# OPTIzelle

## COMPUTATIONAL OPTIMIZATION

Joseph Young

Version: 1.2.0

Release Date: October 24, 2016



OPTIMO joe

www.optimojoe.com

# Optizelle v1.2.0

OptimoJoe
Joseph Young

# Contents

# 1 Introduction

Optizelle [op-t*uh*-**zel**] is an open source software library designed to solve general purpose nonlinear optimization problems of the form

| **Unconstrained** | **Equality Constrained** |
|---|---|
| $\min_{x\in X} \quad f(x)$ | $\min_{x\in X} \quad f(x)$ <br> $\text{st} \quad g(x)=0$ |
| **Inequality Constrained** | **Constrained** |
| $\min_{x\in X} \quad f(x)$ <br> $\text{st} \quad h(x)\succeq 0$ | $\min_{x\in X} \quad f(x)$ <br> $\text{st} \quad g(x)=0$ <br> $h(x)\succeq 0$ |

It features

- **State of the art algorithms**
  - Unconstrained – steepest descent, preconditioned nonlinear-CG (Fletcher-Reeves, Polak-Ribiere, Hestenes-Stiefel), BFGS, Newton-CG, SR1, trust-region Newton, Barzilai-Borwein two-point approximation
  - Equality constrained – inexact composite-step SQP.
  - Inequality constrained – primal-dual interior point method for cone constraints (linear, second-order cone, and semidefinite), log-barrier method for cone constraints
  - Constrained – any combination of the above
- **Open source**
  - Released under the 2-Clause BSD License
  - Free and ready to use with both open and closed sourced commercial codes
- **Multilanguage support**
  - Interfaces to C++, MATLAB/Octave, and Python
- **Robust computations and repeatability**
  - Can stop, archive, and restart the computation from any optimization iteration
  - Combined with the multilanguage support, the optimization can be started in one language and migrated to another. For example, archived optimization runs that started in Python can be migrated and completed in C++.
- **User-defined parallelism**
  - Fully compatible with OpenMP, MPI, or GPUs

- **Extensible linear algebra**
  - Supports user-defined vector algebra and preconditioners
  - Enables sparse, dense, and matrix-free computations
  - Ability to define custom inner products and compatibility with preconditioners such as algebraic multigrid make Optizelle well-suited for PDE constrained optimization
- **Sophisticated Control of the Optimization Algorithms**
  - Allows the user to insert arbitrary code into the optimization algorithm, which enables custom heuristics to be embedded without modifying the source. For example, in signal processing applications, the optimization iterates could be run through a band-pass filter at the end of each optimization iteration.

## 1.1 Licensing

Optizelle is copyrighted by OptimoJoe and licensed under the 2-Clause BSD License:



In short, Optizelle is free to use in both open and closed sourced codes. If you do so, we ask that you provide a citation or link to `http://www.optimojoe.com`.

## 1.2 Support

News, updates, and download information for Optizelle can be found at

`http://www.optimojoe.com/products/optizelle`

Our user forum can be found at

`http://forum.optimojoe.com`

Finally, if you are interested in paid support and consulting, please contact us at contact@optimojoe.com.

## 1.3 Brief example

In order to see a short example of Optizelle in action, consider the unconstrained minimization of the Rosenbrock function

$$\min_{x\in\mathbb{R}^2} \qquad (1-x_1)^2 + 100(x_2 - x_1^2)^2.$$

In order to optimize this function, we use the following code and parameters, which generates the subsequent output.

**Language** C++

**Code**

```
// In this example, we setup and minimize the Rosenbrock function.

#include <vector>
#include <iostream>
#include <string>
#include <cstdlib>
#include "optizelle/optizelle.h"
#include "optizelle/vspaces.h"
#include "optizelle/json.h"

//---Objective0---
// Squares its input
template <typename Real>
Real sq(Real x){
    return x*x;
}

// Define the Rosenbrock function where
//
// f(x,y)=(1-x)^2+100(y-x^2)^2
//
struct Rosenbrock
    : public Optizelle::ScalarValuedFunction <double,Optizelle::Rm>
{
    typedef Optizelle::Rm <double> X;

    // Evaluation of the Rosenbrock function
    double eval(X::Vector const & x) const {
        return sq(1.-x[0])+100.*sq(x[1]-sq(x[0]));
    }

    // Gradient
    void grad(
        X::Vector const & x,
        X::Vector & grad
    ) const {
        grad[0]=-400.*x[0]*(x[1]-sq(x[0]))-2.*(1.-x[0]);
        grad[1]=200.*(x[1]-sq(x[0]));
    }

    // Hessian-vector product
    void hessvec(
        X::Vector const & x,
        X::Vector const & dx,
        X::Vector & H_dx
    ) const {
        H_dx[0]=(1200.*sq(x[0])-400.*x[1]+2)*dx[0]-400.*x[0]*dx[1];
```

```
        H_dx[1]=-400.*x[0]*dx[0]+200.*dx[1];
    }
};
//---Objective1---

//---Preconditioner0---
// Define a perfect preconditioner for the Hessian
struct RosenHInv :
    public Optizelle::Operator <double,Optizelle::Rm,Optizelle::Rm>
{
public:
    typedef Optizelle::Rm <double> X;
    typedef X::Vector X_Vector;
private:
    X_Vector& x;
public:
    RosenHInv(X::Vector& x_) : x(x_) {}
    void eval(X_Vector const & dx,X_Vector & result) const {
        auto one_over_det=1./(80000.*sq(x[0])-80000.*x[1]+400.);
        result[0]=one_over_det*(200.*dx[0]+400.*x[0]*dx[1]);
        result[1]=one_over_det*
            (400.*x[0]*dx[0]+(1200.*x[0]*x[0]-400.*x[1]+2.)*dx[1]);
    }
};
//---Preconditioner1---

int main(int argc,char* argv[]){
    // Read in the name for the input file
    if(argc!=2) {
        std::cerr << "rosenbrock <parameters>" << std::endl;
        exit(EXIT_FAILURE);
    }
    auto fname = argv[1];

    //---State0---
    // Generate an initial guess for Rosenbrock
    auto x = std::vector <double> {-1.2, 1.};

    // Create an unconstrained state based on this vector
    Optizelle::Unconstrained <double,Optizelle::Rm>::State::t state(x);
    //---State1---

    //---Parameters0---
    // Read the parameters from file
    Optizelle::json::Unconstrained <double,Optizelle::Rm>::read(fname,state);
    //---Parameters1---

    //---Functions0---
    // Create the bundle of functions
    Optizelle::Unconstrained <double,Optizelle::Rm>::Functions::t fns;
    fns.f.reset(new Rosenbrock);
    fns.PH.reset(new RosenHInv(state.x));
    //---Functions1---

    //---Solver0---
    // Solve the optimization problem
    Optizelle::Unconstrained <double,Optizelle::Rm>::Algorithms
```

```
        ::getMin(Optizelle::Messaging::stdout,fns,state);
    //---Solver1---

    //---Extract0---
    // Print out the reason for convergence
    std::cout << "The algorithm converged due to: " <<
        Optizelle::OptimizationStop::to_string(state.opt_stop) <<
        std::endl;

    // Print out the final answer
    std::cout << "The optimal point is: (" << state.x[0] << ','
        << state.x[1] << ')' << std::endl;
    //---Extract1---

    // Write out the final answer to file
    Optizelle::json::Unconstrained <double,Optizelle::Rm>::write_restart(
        "solution.json",state);

    // Successful termination
    return EXIT_SUCCESS;
}
```

**Language** Python

**Code**

```
# In this example, we setup and minimize the Rosenbrock function.

import Optizelle
import numpy
import sys

#---Objective0---
# Squares its input
sq = lambda x:x*x

# Define the Rosenbrock function where
#
# f(x,y)=(1-x)^2+100(y-x^2)^2
#
class Rosenbrock(Optizelle.ScalarValuedFunction):
    # Evaluation of the Rosenbrock function
    def eval(self,x):
        return sq(1.-x[0])+100.*sq(x[1]-sq(x[0]))

    # Gradient
    def grad(self,x,grad):
        grad[0]=-400*x[0]*(x[1]-sq(x[0]))-2*(1-x[0])
        grad[1]=200*(x[1]-sq(x[0]))

    # Hessian-vector product
    def hessvec(self,x,dx,H_dx):
        H_dx[0] = (1200*sq(x[0])-400*x[1]+2)*dx[0]-400*x[0]*dx[1]
        H_dx[1] = -400*x[0]*dx[0] + 200*dx[1]
#---Objective1---

#---Preconditioner0---
# Define a perfect preconditioner for the Hessian
```

```python
class RosenHInv(Optizelle.Operator):
    def eval(self,state,dx,result):
        x = state.x
        one_over_det=1./(80000.*sq(x[0])-80000.*x[1]+400.)
        result[0]=one_over_det*(200.*dx[0]+400.*x[0]*dx[1])
        result[1]=(one_over_det*
            (400.*x[0]*dx[0]+(1200.*x[0]*x[0]-400.*x[1]+2.)*dx[1]))
#---Preconditioner1---

# Read in the name for the input file
if len(sys.argv)!=2:
    sys.exit("python rosenbrock.py <parameters>")
fname = sys.argv[1]

#---State0---
# Generate an initial guess for Rosenbrock
x = numpy.array([-1.2,1.0])

# Create an unconstrained state based on this vector
state=Optizelle.Unconstrained.State.t(Optizelle.Rm,x)
#---State1---

#---Parameters0---
# Read the parameters from file
Optizelle.json.Unconstrained.read(Optizelle.Rm,fname,state)
#---Parameters1---

#---Functions0---
# Create the bundle of functions
fns=Optizelle.Unconstrained.Functions.t()
fns.f=Rosenbrock()
fns.PH=RosenHInv()
#---Functions1---

#---Solver0---
# Solve the optimization problem
Optizelle.Unconstrained.Algorithms.getMin(
    Optizelle.Rm,Optizelle.Messaging.stdout,fns,state)
#---Solver1---

#---Extract0---
# Print out the reason for convergence
print "The algorithm converged due to: %s" % (
    Optizelle.OptimizationStop.to_string(state.opt_stop))

# Print out the final answer
print "The optimal point is: (%e,%e)" % (state.x[0],state.x[1])
#---Extract1---

# Write out the final answer to file
Optizelle.json.Unconstrained.write_restart(Optizelle.Rm,"solution.json",state)
```

**Language** MATLAB/Octave

**Code**

```matlab
% In this example, we setup and minimize the Rosenbrock function.
function rosenbrock(fname)
```

```
    % Read in the name for the input file
    if nargin ~=1
        error('rosenbrock <parameters>');
    end

    % Execute the optimization
    main(fname);
end

%---Objective0---
% Squares its input
function z = sq(x)
    z=x*x;
end

% Define the Rosenbrock function where
%
% f(x,y)=(1-x)^2+100(y-x^2)^2
%
function self = Rosenbrock()

    % Evaluation of the Rosenbrock function
    self.eval = @(x) sq(1.-x(1))+100.*sq(x(2)-sq(x(1)));

    % Gradient
    self.grad = @(x) [
        -400.*x(1)*(x(2)-sq(x(1)))-2.*(1.-x(1));
        200.*(x(2)-sq(x(1)))];

    % Hessian-vector product
    self.hessvec = @(x,dx) [
        (1200.*sq(x(1))-400.*x(2)+2)*dx(1)-400.*x(1)*dx(2);
        -400.*x(1)*dx(1)+200.*dx(2)];
end
%---Objective1---

%---Preconditioner0---
% Define a perfect preconditioner for the Hessian
function self = RosenHInv()
    self.eval = @(state,dx) eval(state,dx);
end
function result = eval(state,dx)
    x = state.x;
    one_over_det=1./(80000.*sq(x(1))-80000.*x(2)+400.);
    result = [
        one_over_det*(200.*dx(1)+400.*x(1)*dx(2));
        one_over_det*...
            (400.*x(1)*dx(1)+(1200.*x(1)*x(1)-400.*x(2)+2.)*dx(2))];
end
%---Preconditioner1---

% Actually runs the program
function main(fname)

    % Grab the Optizelle library
    global Optizelle;
    setupOptizelle();
```

```
    %---State0---
    % Generate an initial guess for Rosenbrock
    x = [-1.2;1.];

    % Create an unconstrained state based on this vector
    state=Optizelle.Unconstrained.State.t(Optizelle.Rm,x);
    %---State1---

    %---Parameters0---
    % Read the parameters from file
    state=Optizelle.json.Unconstrained.read(Optizelle.Rm,fname,state);
    %---Parameters1---

    %---Functions0---
    % Create the bundle of functions
    fns=Optizelle.Unconstrained.Functions.t;
    fns.f=Rosenbrock();
    fns.PH=RosenHInv();
    %---Functions1---

    %---Solver0---
    % Solve the optimization problem
    state = Optizelle.Unconstrained.Algorithms.getMin( ...
        Optizelle.Rm,Optizelle.Messaging.stdout,fns,state);
    %---Solver1---

    %---Extract0---
    % Print out the reason for convergence
    fprintf('The algorithm converged due to: %s\n', ...
        Optizelle.OptimizationStop.to_string(state.opt_stop));

    % Print out the final answer
    fprintf('The optimal point is: (%e,%e)\n',state.x(1),state.x(2));
    %---Extract1---

    % Write out the final answer to file
    Optizelle.json.Unconstrained.write_restart( ...
        Optizelle.Rm,'solution.json',state);
end
```

**Language** Optizelle Parameters

**Code**

```
   "Optizelle" :{
      "msg_level" :1,
      "iter_max" :50,
      "eps_trunc" :1e-12
   },
```

**Language** Optizelle Output

**Code**

```
iter        f(x)        ||grad||    ||dx||
1           2.42e+01    2.33e+02    .
2           4.73e+00    4.64e+00    3.81e-01
```

```
.           4.73e+00   4.64e+00   1.00e+00
3           4.00e+00   1.74e+01   5.00e-01
4           3.34e+00   2.33e+01   5.00e-01
5           2.58e+00   8.77e+00   2.04e-01
.           2.58e+00   8.77e+00   4.91e-01
6           2.09e+00   8.48e+00   2.45e-01
7           1.65e+00   9.60e+00   2.45e-01
8           1.22e+00   5.00e+00   1.46e-01
9           9.64e-01   9.06e+00   2.13e-01
10          6.22e-01   1.77e+00   1.10e-01
.           6.22e-01   1.77e+00   3.44e-01
11          4.40e-01   4.42e+00   1.72e-01
12          2.81e-01   3.72e+00   1.68e-01
13          1.71e-01   3.60e+00   1.72e-01
14          9.43e-02   3.58e+00   1.80e-01
15          4.49e-02   2.47e+00   1.60e-01
16          1.82e-02   2.40e+00   1.58e-01
17          5.16e-03   1.02e+00   1.11e-01
18          8.94e-04   7.81e-01   9.41e-02
19          4.86e-05   1.17e-01   3.89e-02
20          2.49e-07   1.55e-02   1.36e-02
21          7.47e-12   4.80e-05   7.98e-04
22          7.06e-21   2.62e-09   5.63e-06
The algorithm converged due to: GradientSmall
The optimal point is: (1,1)
```

## 1.4 History

Optizelle originated in 2010 as a code called PEOpt (Parameter Estimation Using Optimization) written by Joseph Young at Sandia National Laboratories. There, it was used as the computational driver for a variety of both internal and external customers. Due to the scale of the problems involved and the nuances of high-performance computing environments, PEOpt was designed specifically to integrate with large, existing code bases as quickly and unobtrusively as possible. Later, Sandia approved the open source release of PEOpt on two separate occasions in 2012 and 2013 under the 2-Clause BSD License. It was from this released code that Joseph continued work on Optizelle through a new company called OptimoJoe.

# 2 Installation

In the following chapter, we discuss how to download, build, and incorporate Optizelle into a new project.

## 2.1 Downloading

Optizelle can be downloaded from

`http://www.optimojoe.com/products/optizelle`

in a variety of precompiled packages. Here, we also provide direct access to our source code repository.

## 2.2 Installing and Uninstalling

The installation method depends on the platform, but generally involves opening the installer and following the specified instructions

**Windows** Open the installer

**macOS**

1. Open the installer
2. Drag the Optizelle folder to Applications
3. Copy the file

   `/Applications/Optizelle/share/optizelle/com.optimojoe.optizelle.plist`

   to

   `/Library/LaunchAgents/`
4. Close the Terminal application if open and reboot

Note, we summarize these steps and provide additional information in the `ReadMe.txt` file provided after opening the installer.

**Linux/Unix**

1. Unzip the `tar.gz` file to a local directory or use the appropriate package manager to install the `rpm` or `deb` package directly
2. Add `/some/path/share/optizelle/matlab` to the `MATLABPATH`
3. Add `/some/path/share/optizelle/octave` to the `OCTAVE_PATH`
4. Add `/some/path/share/optizelle/python` to the `PYTHONPATH`

where `/some/path` denotes the Optizelle install location. By default, the `deb` and `rpm` files install Optizelle to `/usr/local/`. Note, on most Linux distributions, we add a variable to the path by adding

```
export SOMEVARIABLE=$SOMEVARIABLE:NEWPATH
```

to the file `~/.bashrc`. In other words, if we install Optizelle to `/usr/local`, we add the following to text `~/.bashrc`

```
export MATLABPATH=$MATLABPATH:/usr/local/share/optizelle/matlab
export OCTAVE_PATH=$OCTAVE_PATH:/usr/local/share/optizelle/octave
export PYTHONPATH=$PYTHONPATH:/usr/local/share/optizelle/python
```

Remember to execute `source ~/.bashrc` on all active shells, log out and back in, or reboot for the changes to take affect.

Similar to installation, how we uninstall Optizelle depends on the platform

**Windows** Click the menus Start → Settings → System → Apps & features → Optizelle → Uninstall

**macOS**

1. Drag the folder `/Applications/Optizelle` to the trash
2. Drag the file `/Library/LaunchAgents/com.optimojoe.optizelle.plist` to the trash

**Linux/Unix**

1. When installed locally using the `tar.gz` package, delete the installation folder
2. When installed using the `rpm` or `deb` packages, use the package manager to remove Optizelle
3. Delete any modifications to the path made in the file `~/.bashrc` or other similar configuration file

## 2.3 Dependencies

Depending on its configuration, Optizelle uses the following software packages

| Package | Version | License | C++ | Python | MATLAB | Octave | Docs | Windows |
|---|---|---|---|---|---|---|---|---|
| Optizelle | 1.2.0 | BSD | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| JsonCpp | 0.10.6 | Public | ✓ | ✓ | ✓ | ✓ | | ✓ |
| BLAS/LAPACK | 3.5.0 | BSD | ✓ | ✓ | ✓ | ✓ | | ✓ |
| CMake | 3.1 | BSD | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| WiX | 3.10 | MS-RL | | | | | | ✓ |
| GCC | 4.9 | GPL | ✓ | ✓ | ✓ | ✓ | | ✓ |
| TeX Live | 2014 | Various | | | | | ✓ | |
| Python | 2.7 | Python | | ✓ | | | | |
| NumPy | 1.10 | BSD | | ✓ | | | | |
| MATLAB | R2016a | Custom | | | ✓ | | | |
| JSONlab | 1.0-RC1 | BSD | | | ✓ | ✓ | | |
| Octave | 4.0 | GPL | | | | ✓ | | |

Note, we generally depend on GCC for both its C++ and Fortran compiler, but an alternative compiler such as Clang works as well. Since we do not modify GCC, the GCC Runtime Library Exception applies. In addition, Optizelle remains compatible with most high-performance varieties of BLAS and LAPACK.

## 2.4 Building

Optizelle uses CMake as its build system. On Linux, Unix, macOS, Cygwin, or MSYS, execute the following commands from the base Optizelle directory:

1. `mkdir build`
2. `cd build`
3. `ccmake ..`
4. Configure the build.

5. `make`

On Windows, if not using Cygwin or MSYS, execute the following commands:

1. Using Windows Explorer, create a directory called `build` in the base Optizelle directory.
2. Run `cmake-gui.exe`
3. Set the source directory to the base Optizelle directory.
4. Set the build directory to the `build` folder created above.
5. Configure the build.
6. Build the code (`make` with Cygwin or MSYS.)

Rather than using `ccmake`, we can also run `cmake` directly in order to configure the build. This allows us to skip the CMake menu system and configure Optizelle directly, which can be advantageous when compiling Optizelle on multiple, but similar systems. In order to accomplish this, we execute a command such as

```
cmake \
    -DENABLE_OPENMP:BOOL=ON \
    -DENABLE_BUILD_JSONCPP:BOOL=ON \
    -DJSONCPP_ARCHIVE:FILEPATH=/path/to/jsoncpp.zip \
    -DENABLE_BUILD_BLAS_AND_LAPACK:BOOL=ON \
    -DLAPACK_ARCHIVE:FILEPATH=/path/to/lapack.tgz \
    -DENABLE_CPP_EXAMPLES:BOOL=ON \
    -DENABLE_CPP_UNIT:BOOL=ON \
    -DENABLE_PYTHON:BOOL=ON \
    -DENABLE_PYTHON_EXAMPLES:BOOL=ON \
    -DENABLE_PYTHON_UNIT:BOOL=ON \
    -DENABLE_MATLAB:BOOL=ON \
    -DMATLAB_EXECUTABLE:FILEPATH=/path/to/matlab \
    -DMATLAB_INCLUDE_DIR:PATH=/path/to/extern/include \
    -DMATLAB_LIBRARY:FILEPATH=/path/to/bin/glnxa64/libmex.so \
    -DMATLAB_MEX_EXTENSION:STRING=mexa64 \
    -DENABLE_MATLAB_EXAMPLES:BOOL=ON \
    -DENABLE_MATLAB_UNIT:BOOL=ON \
    -DENABLE_OCTAVE:BOOL=ON \
    -DOCTAVE_EXECUTABLE:FILEPATH=/path/to/octave \
    -DOCTAVE_INCLUDE_DIR:PATH=/path/to/octave \
    -DOCTAVE_LIBRARY:FILEPATH=/path/to/liboctinterp.so \
    -DENABLE_OCTAVE_EXAMPLES:BOOL=ON \
    -DENABLE_OCTAVE_UNIT:BOOL=ON \
    -DENABLE_BUILD_JSONLAB:BOOL=ON \
    -DJSONLAB_ARCHIVE:FILEPATH=/path/to/jsonlab.zip \
..
```

where the actual paths, libraries, and archives depend on the individual system. Generally, we put this command inside a shell script or batch file in order to make it easier to edit. As far as the available options, we list them in the next section.

After building Optizelle, installation is as simple as executing

`make install`

from the CMake build directory using GNU Make, MSYS, or Cygwin. If using a different Make utility, call it on the `install` target. For a complete list of installed files, see

`install_manifest.txt`

located in the CMake build directory.
After installation via `make install`, we must also

1. Add `/some/path/share/optizelle/matlab` to the `MATLABPATH`
2. Add `/some/path/share/optizelle/octave` to the `OCTAVE_PATH`
3. Add `/some/path/share/optizelle/python` to the `PYTHONPATH`

where `/some/path` denotes the path found in the `CMAKE_INSTALL_PREFIX` configuration variable described below. How we set environment variables depends on the platform

**Windows** Modify each environment variable via the sequence

1. Open File Explorer
2. Right click This PC
3. Click the menus Advanced System Settings → System Properties → Environment Variables → New (if the variable doesn't exist) or Edit (if the variable does exist)
4. Modify `PATH` with `C:\some\path\lib` and `C:\some\path\share\optizelle\thirdparty\lib`
5. Modify `MATLABPATH` with `C:\some\path\share\optizelle\matlab`
6. Modify `OCTAVE_PATH` with `C:\some\path\share\optizelle\octave`
7. Modify `PYTHONPATH` with `C:\some\path\share\optizelle\python`

where `C:\some\path` denotes the installation path found in the CMake variable `CMAKE_INSTALL_PREFI`

**macOS** Add a plist file to `/Library/LaunchAgents` or `~/Library/LaunchAgents`. For example

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0"><dict>
    <key>Label</key>
    <string>Optizelle.startup</string>
    <key>ProgramArguments</key>
    <array>
        <string>sh</string>
        <string>-c</string>
        <string>
            launchctl setenv MATLABPATH $MATLABPATH:/some/path/share/optizelle/ma
            launchctl setenv OCTAVE_PATH $OCTAVE_PATH:/some/path/share/optizelle/
            launchctl setenv PYTHONPATH $PYTHONPATH:/some/path/share/optizelle/py
        </string>
    </array>
  <key>RunAtLoad</key>
  <true/>
</dict></plist>
```

where `/some/path` denotes the installation path found in the CMake variable `CMAKE_INSTALL_PREFIX`. Note, we must close the Terminal application and then reboot for the changes to take affect.

**Linux/Unix** When using the Bash shell, we add

```
export MATLABPATH=$MATLABPATH:/some/path/share/optizelle/matlab
export OCTAVE_PATH=$OCTAVE_PATH:/some/path/share/optizelle/octave
export PYTHONPATH=$PYTHONPATH:/some/path/share/optizelle/python
```

to ~/.bashrc where /some/path denotes the installation path found in the CMake variable CMAKE_INSTALL_PREFIX. Note, we must also execute the command source ~/.bashrc on all active shells, log out and back in, or reboot for the changes to take affect.

As a final note, CMake does not provide a native uninstallation process when installing Optizelle in this manner. Nevertheless, on Linux, Unix, macOS, MSYS, or Cygwin, the command

```
xargs rm < install_manifest.txt
```

will remove the installation. Also, don't forget to remove each of the environment variables added in the above installation process.

## 2.5 Configuring

Optizelle provides several different options within CMake in order to customize the build. We describe these flags in the table below:

| | |
|---|---|
| **Flag** | CMAKE_INSTALL_PREFIX |
| **Type** | PATH |
| **Default** | Varies |
| **Dependency** | None |
| **Enables** | None |
| **Autodetect?** | No |
| **Description** | Install location of Optizelle. |

| | |
|---|---|
| **Flag** | ENABLE_DOCUMENTATION |
| **Type** | BOOL |
| **Default** | OFF |
| **Dependency** | None |
| **Enables** | PDFLATEX_COMPILER, ENABLE_A4_PAPER |
| **Autodetect?** | No |
| **Description** | Enables the build of the Optizelle manual from the LaTeX source. It builds a pdf file of the manual. |

| | |
|---|---|
| **Flag** | PDFLATEX_COMPILER |
| **Type** | FILEPATH |
| **Default** | None |
| **Dependency** | ENABLE_DOCUMENTATION |
| **Enables** | None |
| **Autodetect?** | Yes |
| **Description** | Complete path and executable for pdflatex. |

| | |
|---|---|
| **Flag** | ENABLE_A4_PAPER |
| **Type** | BOOL |
| **Default** | OFF |
| **Dependency** | ENABLE_DOCUMENTATION |
| **Enables** | None |
| **Autodetect?** | No |
| **Description** | When ON, the manual uses A4 paper. Otherwise, the manual uses Letter paper. |

| | |
|---|---|
| **Flag** | ENABLE_CPP |
| **Type** | BOOL |
| **Default** | OFF |
| **Dependency** | None |
| **Enables** | CMAKE_CXX_FLAGS, CMAKE_BUILD_TYPE, ENABLE_OPENMP, ENABLE_BUILD_BLAS_AND_LAPACK, ENABLE_BUILD_JSONCPP, BLAS_LIBRARY, LAPACK_LIBRARY, JSONCPP_INCLUDE_DIR, JSONCPP_LIBRARY, ENABLE_CPP_EXAMPLES, ENABLE_CPP_UNIT, ENABLE_PYTHON, ENABLE_MATLAB, ENABLE_OCTAVE |
| **Autodetect?** | No |
| **Description** | Enables the Optizelle C++ library. |

| | |
|---|---|
| **Flag** | CMAKE_CXX_FLAGS |
| **Type** | STRING |
| **Default** | None |
| **Dependency** | ENABLE_CPP |
| **Enables** | None |
| **Autodetect?** | No |
| **Description** | C++ compiler specific flags. |

| | |
|---|---|
| **Flag** | CMAKE_BUILD_TYPE |
| **Type** | STRING |
| **Default** | None |
| **Dependency** | ENABLE_CPP |
| **Enables** | None |
| **Autodetect?** | No |
| **Description** | Generally set to either RELEASE or DEBUG. Set to RELEASE for production libraries. Set to DEBUG to allow profiling through utilities such as OProfile. |

| | |
|---|---|
| **Flag** | ENABLE_OPENMP |

| | |
|---|---|
| **Type** | BOOL |
| **Default** | OFF |
| **Dependency** | ENABLE_CPP |
| **Enables** | None |
| **Autodetect?** | No |
| **Description** | Enable OpenMP/threaded support for the default, dense vector spaces. Note, many BLAS and LAPACK libraries such as those from ATLAS benefit from OpenMP directives. |

| | |
|---|---|
| **Flag** | ENABLE_BUILD_BLAS_AND_LAPACK |
| **Type** | BOOL |
| **Default** | OFF |
| **Dependency** | ENABLE_CPP |
| **Enables** | LAPACK_ARCHIVE |
| **Autodetect?** | No |
| **Description** | Builds BLAS and LAPACK from source in case an optimized version is not available. |

| | |
|---|---|
| **Flag** | LAPACK_ARCHIVE |
| **Type** | FILEPATH |
| **Default** | None |
| **Dependency** | ENABLE_BUILD_BLAS_AND_LAPACK |
| **Enables** | None |
| **Autodetect?** | No |
| **Description** | Location of the LAPACK archive downloaded from Netlib. |

| | |
|---|---|
| **Flag** | ENABLE_BUILD_JSONCPP |
| **Type** | BOOL |
| **Default** | OFF |
| **Dependency** | ENABLE_CPP |
| **Enables** | JSONCPP_ARCHIVE |
| **Autodetect?** | No |
| **Description** | Builds JsonCpp from source. |

| | |
|---|---|
| **Flag** | JSONCPP_ARCHIVE |
| **Type** | FILEPATH |
| **Default** | None |

| | |
|---|---|
| **Dependency** | ENABLE_BUILD_JSONCPP |
| **Enables** | None |
| **Autodetect?** | No |
| **Description** | Location of the JsonCpp archive downloaded from GitHub. |

| | |
|---|---|
| **Flag** | BLAS_LIBRARY |
| **Type** | FILEPATH |
| **Default** | None |
| **Dependency** | ENABLE_CPP |
| **Enables** | None |
| **Autodetect?** | Yes |
| **Description** | A semicolon separated list of the complete path and library used to provide BLAS. This must include all required libraries in order to successfully compile a BLAS dependent application. For example, using ATLAS BLAS, one possible entry is:<br><br>`/usr/lib/libf77blas.a;/usr/lib/libatlas.a` |

| | |
|---|---|
| **Flag** | LAPACK_LIBRARY |
| **Type** | FILEPATH |
| **Default** | None |
| **Dependency** | ENABLE_CPP |
| **Enables** | None |
| **Autodetect?** | Yes |
| **Description** | A semicolon separated list of the complete path and library used to provide LAPACK. This must include all required libraries, except for BLAS libraries specified in BLAS_LIBRARY, in order to successfully compile a LAPACK dependent application. For example, using ATLAS LAPACK, one possible entry is:<br><br>`/usr/lib/liblapack.a;/usr/lib/libgfortran.a`<br><br>Note, this example assumes that we include `libatlas.a` in our BLAS_LIBRARY filepath. |

| | |
|---|---|
| **Flag** | JSONCPP_INCLUDE_DIR |
| **Type** | PATH |
| **Default** | None |
| **Dependency** | ENABLE_CPP |
| **Enables** | None |
| **Autodetect?** | Yes |
| **Description** | A path that indicates where the jsoncpp headers have been installed. The actual headers must be found in `$JSONCPP_INCLUDE_DIR/json/` |

.

| | |
|---|---|
| **Flag** | JSONCPP_LIBRARY |
| **Type** | FILEPATH |
| **Default** | None |
| **Dependency** | ENABLE_CPP |
| **Enables** | None |
| **Autodetect?** | Yes |
| **Description** | Complete path and library for JsonCpp. |

| | |
|---|---|
| **Flag** | ENABLE_CPP_EXAMPLES |
| **Type** | BOOL |
| **Default** | OFF |
| **Dependency** | ENABLE_CPP |
| **Enables** | None |
| **Autodetect?** | No |
| **Description** | Enables the build and installation of simple examples that demonstrate the use of Optizelle. |

| | |
|---|---|
| **Flag** | ENABLE_CPP_UNIT |
| **Type** | BOOL |
| **Default** | OFF |
| **Dependency** | ENABLE_CPP |
| **Enables** | None |
| **Autodetect?** | No |
| **Description** | Enables the build of unit tests that help validate the Optizelle code. Execute these unit tests by running `ctest` in the CMake build directory. |

| | |
|---|---|
| **Flag** | ENABLE_PYTHON |
| **Type** | BOOL |
| **Default** | OFF |
| **Dependency** | ENABLE_CPP |
| **Enables** | PYTHON_INCLUDE_DIR, PYTHON_LIBRARY, PYTHON_EXECUTABLE, ENABLE_PYTHON_EXAMPLES, ENABLE_PYTHON_UNIT |
| **Autodetect?** | No |
| **Description** | Enables the build of the Python wrappers for Optizelle. |

| | |
|---|---|
| **Flag** | PYTHON_INCLUDE_DIR |
| **Type** | FILEPATH |
| **Default** | None |
| **Dependency** | ENABLE_PYTHON |
| **Enables** | None |
| **Autodetect?** | Yes |
| **Description** | A path that indicates where the Python 2.7 headers have been installed. We do not prefix these headers, so we look directly in the directory provided here. |

| | |
|---|---|
| **Flag** | PYTHON_LIBRARY |
| **Type** | FILEPATH |
| **Default** | None |
| **Dependency** | ENABLE_PYTHON |
| **Enables** | None |
| **Autodetect?** | Yes |
| **Description** | Complete path and library for Python 2.7. |

| | |
|---|---|
| **Flag** | PYTHON_EXECUTABLE |
| **Type** | FILEPATH |
| **Default** | None |
| **Dependency** | ENABLE_PYTHON |
| **Enables** | None |
| **Autodetect?** | Yes |
| **Description** | Complete path and executable for Python 2.7. |

| | |
|---|---|
| **Flag** | ENABLE_PYTHON_EXAMPLES |
| **Type** | BOOL |
| **Default** | OFF |
| **Dependency** | ENABLE_PYTHON |
| **Enables** | None |
| **Autodetect?** | No |
| **Description** | Enables the build and installation of simple examples that demonstrate the use of the Python wrappers for Optizelle. |

| | |
|---|---|
| **Flag** | ENABLE_PYTHON_UNIT |
| **Type** | BOOL |

**Default** OFF

**Dependency** ENABLE_PYTHON

**Enables** None

**Autodetect?** No

**Description** Enables the build of unit tests that help validate the Python wrappers for the Optizelle code. Execute these unit tests by running `ctest` in the CMake build directory.

**Flag** ENABLE_MATLAB

**Type** BOOL

**Default** OFF

**Dependency** ENABLE_CPP

**Enables** MATLAB_MEX_EXTENSION, MATLAB_INCLUDE_DIR, MATLAB_LIBRARY, MATLAB_EXECUTABLE, ENABLE_BUILD_JSONLAB, JSONLAB_DIR, ENABLE_MATLAB_EXAMPLES, ENABLE_MATLAB_UNIT

**Autodetect?** No

**Description** Enables the build of the MATLAB wrappers for Optizelle.

**Flag** MATLAB_MEX_EXTENSION

**Type** STRING

**Default** None

**Dependency** ENABLE_MATLAB

**Enables** None

**Autodetect?** No

**Description** Extension of mex files on the system. This can be found by typing in the command 'mexext' inside of MATLAB.

**Flag** MATLAB_INCLUDE_DIR

**Type** FILEPATH

**Default** None

**Dependency** ENABLE_MATLAB

**Enables** None

**Autodetect?** Yes

**Description** Path that indicates where the MATLAB header `mex.h` has been installed. We do not prefix these headers, so we look directly in the directory provided here. Generally, this is generally the `extern/include` directory inside the primary MATLAB directory.

**Flag** MATLAB_LIBRARY

| | |
|---|---|
| **Type** | `FILEPATH` |
| **Default** | None |
| **Dependency** | `ENABLE_MATLAB` |
| **Enables** | None |
| **Autodetect?** | Yes |
| **Description** | Complete path and library for MATLAB, `mex`. Sometimes, we have to include the `mx` library as well. If compilation fails and there are several undefined symbols with prefixed with mx, then add the `mx` library and separate it from `mex` with a semicolon. Generally, these libraries are generally found nested within the `bin` directory in the primary MATLAB folder. |

| | |
|---|---|
| **Flag** | `MATLAB_EXECUTABLE` |
| **Type** | `FILEPATH` |
| **Default** | None |
| **Dependency** | `ENABLE_MATLAB` |
| **Enables** | None |
| **Autodetect?** | No |
| **Description** | Complete path and executable for MATLAB. |

| | |
|---|---|
| **Flag** | `ENABLE_MATLAB_EXAMPLES` |
| **Type** | `BOOL` |
| **Default** | `OFF` |
| **Dependency** | `ENABLE_MATLAB` |
| **Enables** | None |
| **Autodetect?** | No |
| **Description** | Enables the build and installation of simple examples that demonstrate the use of the MATLAB wrappers for Optizelle. |

| | |
|---|---|
| **Flag** | `ENABLE_MATLAB_UNIT` |
| **Type** | `BOOL` |
| **Default** | `OFF` |
| **Dependency** | `ENABLE_MATLAB` |
| **Enables** | None |
| **Autodetect?** | No |
| **Description** | Enables the build of unit tests that help validate the MATLAB wrappers for the Optizelle code. Execute these unit tests by running `ctest` in the CMake build directory. |

| | |
|---|---|
| **Flag** | ENABLE_OCTAVE |
| **Type** | BOOL |
| **Default** | OFF |
| **Dependency** | ENABLE_CPP |
| **Enables** | OCTAVE_INCLUDE_DIR, OCTAVE_LIBRARY, OCTAVE_EXECUTABLE, ENABLE_BUILD_JSONLAB, JSONLAB_DIR, ENABLE_OCTAVE_EXAMPLES, ENABLE_OCTAVE_UNIT |
| **Autodetect?** | No |
| **Description** | Enables the build of the Octave wrappers for Optizelle. |

| | |
|---|---|
| **Flag** | OCTAVE_INCLUDE_DIR |
| **Type** | FILEPATH |
| **Default** | None |
| **Dependency** | ENABLE_OCTAVE |
| **Enables** | None |
| **Autodetect?** | Yes |
| **Description** | Path that indicates where the Octave header `mex.h` has been installed. We do not prefix these headers, so we look directly in the directory provided here. Generally, this is the folder called `octave-x.x.x/octave` inside the system `include` directory where `x.x.x` denotes the version number. |

| | |
|---|---|
| **Flag** | OCTAVE_LIBRARY |
| **Type** | FILEPATH |
| **Default** | None |
| **Dependency** | ENABLE_OCTAVE |
| **Enables** | None |
| **Autodetect?** | Yes |
| **Description** | Complete path and library for Octave, `octinterp`. Generally, this library is found nested within the `octave` directory inside the system `lib` directory. |

| | |
|---|---|
| **Flag** | OCTAVE_EXECUTABLE |
| **Type** | FILEPATH |
| **Default** | None |
| **Dependency** | ENABLE_OCTAVE |
| **Enables** | None |
| **Autodetect?** | No |
| **Description** | Complete path and executable for Octave. |

| | |
|---|---|
| **Flag** | `ENABLE_OCTAVE_EXAMPLES` |
| **Type** | `BOOL` |
| **Default** | `OFF` |
| **Dependency** | `ENABLE_OCTAVE` |
| **Enables** | None |
| **Autodetect?** | No |
| **Description** | Enables the build and installation of simple examples that demonstrate the use of the Octave wrappers for Optizelle. |

| | |
|---|---|
| **Flag** | `ENABLE_OCTAVE_UNIT` |
| **Type** | `BOOL` |
| **Default** | `OFF` |
| **Dependency** | `ENABLE_OCTAVE` |
| **Enables** | None |
| **Autodetect?** | No |
| **Description** | Enables the build of unit tests that help validate the Octave wrappers for the Optizelle code. Execute these unit tests by running `ctest` in the CMake build directory. |

| | |
|---|---|
| **Flag** | `ENABLE_BUILD_JSONLAB` |
| **Type** | `BOOL` |
| **Default** | `OFF` |
| **Dependency** | `ENABLE_MATLAB` or `ENABLE_OCTAVE` |
| **Enables** | `JSONLAB_ARCHIVE` |
| **Autodetect?** | No |
| **Description** | Builds jsonlab from source. |

| | |
|---|---|
| **Flag** | `JSONLAB_ARCHIVE` |
| **Type** | `FILEPATH` |
| **Default** | None |
| **Dependency** | `ENABLE_BUILD_JSONLAB` |
| **Enables** | None |
| **Autodetect?** | No |
| **Description** | Location of the json archive downloaded from GitHub. |

| | |
|---|---|
| **Flag** | `JSONLAB_DIR` |

**Type** `PATH`

**Default** None

**Dependency** `ENABLE_MATLAB` or `ENABLE_OCTAVE`

**Enables** None

**Autodetect?** Yes

**Description** A path that indicates where the jsonlab library has been installed. This is automatically set when `ENABLE_BUILD_JSONLAB` is enabled.

## 2.6 Platform Specific Configuration

Due to a variety of platform specific quirks, some additional compilation flags may be necessary. In order to use these flags, place them in the `CMAKE_CXX_FLAGS` variable, separated by spaces, in the CMake configuration.

**Flag** `-include math.h`

**Platform** Windows

**Interface** Python

**Indication** During compilation, `error: '::hypot' has not been declared`

**Description** Fixes a bug inside of Python where `hypot` has been renamed

**Flag** `-DMS_WIN64`

**Platform** Windows

**Interface** Python

**Indication** During compilation, `undefined reference to '__imp_Py_InitModule4'`

**Description** Tells Python to use Windows 64-bit specific code

# 3 Basic API

We organize Optizelle's algorithms into four different categories:

| **Unconstrained** | **Equality Constrained** |
|---|---|
| $\min_{x\in X} \quad f(x)$ | $\min_{x\in X} \quad f(x)$ <br> $\text{st} \quad g(x)=0$ |
| **Inequality Constrained** | **Constrained** |
| $\min_{x\in X} \quad f(x)$ <br> $\text{st} \quad h(x)\succeq 0$ | $\min_{x\in X} \quad f(x)$ <br> $\text{st} \quad g(x)=0$ <br> $h(x)\succeq 0$ |

Since these formulations necessitate different algorithms, we segregate the overall structure of Optizelle and the algorithms themselves into these categories. In order to optimize these formulations, we follow the general procedure:

1. Import Optizelle
2. Import or define the appropriate vector spaces
3. Define the objective function
4. (Optional) Define the constraints
5. (Optional) Define the preconditioners
6. Create the optimization state
7. Set the optimization parameters
8. Accumulate the functions
9. Call the optimization solver
10. Extract the solution
11. Compile/run the program

We discuss how to implement each of the above steps below.

## 3.1 Import Optizelle

Each interface uses its own method to import Optizelle:

**Language** C++

**Code**

```
#include "optizelle/optizelle.h"
#include "optizelle/vspaces.h"
#include "optizelle/json.h"
```

**Language** Python

**Code**

```
import Optizelle
```

**Language** MATLAB/Octave

**Code**

```
global Optizelle
setupOptizelle();
```

In C++, we always require `optizelle/optizelle.h`, but only require `optizelle/json.h` when working with JSON and `optizelle/vspaces.h` when using our default vector spaces such as `Optizelle::Rm` and `Optizelle::SQL`. In Python, we simply need to include the `Optizelle` module and everything else is automatically imported. Finally, in MATLAB/Octave, we encapsulate all of the required functions in the global variable `Optizelle`.

## 3.2 Import or define the appropriate vector spaces

In the optimization problems

| **Unconstrained** | **Equality Constrained** |
|---|---|
| $\min_{x\in X} \quad f(x)$ | $\min_{x\in X} \quad f(x)$ <br> $\text{st} \quad g(x) = 0$ |
| **Inequality Constrained** | **Constrained** |
| $\min_{x\in X} \quad f(x)$ <br> $\text{st} \quad h(x) \succeq 0$ | $\min_{x\in X} \quad f(x)$ <br> $\text{st} \quad g(x) = 0$ <br> $\quad h(x) \succeq 0$ |

we require that

$$\begin{aligned} f :& X \to \mathbb{R} \\ g :& X \to Y \\ h :& X \to Z. \end{aligned}$$

Here, the spaces $X$, $Y$, and $Z$ denote *vector spaces*, more specifically, Hilbert spaces. For most problems, these vector spaces just denote $\mathbb{R}^m$, but we also allow these vector spaces to be spaces of functions such as $L^2(\Omega)$ or matrices such as $\mathbb{R}^{m\times n}$ as long as they contain the necessary operations that we describe in the section Customized vector spaces. A vector space consists of two separate pieces: the actual vector and the operations required to compute on this vector. In Optizelle, we maintain this separation. For $\mathbb{R}^m$, we provide a reasonable implementation of the vector space with the following:

**Language** C++

**Vector** `std::vector`

**Operations** `Optizelle::Rm`

**Language** C++

**Vector** `numpy.array`

**Operations** `Optizelle.Rm`

**Language** MATLAB/Octave

**Vector** `[]` (column vector)

**Operations** `Optizelle.Rm`

To be precise, each of these vector spaces uses the inner product $\langle x, y\rangle = x^T y$ and defines inequalities pointwise, $x \succeq y \iff x_i \geq y_i$ for all $1 \leq i \leq m$. Note, we don't require users to use these vector operations in their code. Simply, if we're happy using the above vectors, we can use these operations exclusively in Optizelle and forget their details.

## 3.3 Define the objective function

In the optimization problems

| **Unconstrained** | **Equality Constrained** |
|---|---|
| $\min_{x\in X} \quad f(x)$ | $\min_{x\in X} \quad f(x)$ <br> $\text{st} \quad g(x) = 0$ |
| **Inequality Constrained** | **Constrained** |
| $\min_{x\in X} \quad f(x)$ <br> $\text{st} \quad h(x) \succeq 0$ | $\min_{x\in X} \quad f(x)$ <br> $\text{st} \quad g(x) = 0$ <br> $\quad h(x) \succeq 0$ |

the function $f : X \to \mathbb{R}$ denotes the *objective function*. Note, we restrict ourselves to scalar-valued functions and do not consider multi-objective optimization problems. In order to optimize with this function, we require information about its evaluation and derivatives. Specifically, we require its evaluation, $f(x)$, and gradient, $\nabla f(x)$. In order to use second-order algorithms, we also require the Hessian-vector product, $\nabla^2 f(x)\delta x$. In the case that $f : \mathbb{R}^m \to \mathbb{R}$, we can obtain each of these quantities from its partial derivatives. Specifically, we write

$$f(x) = f(x_1, \dots, x_m).$$

Then, we have that

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(x) \\ \vdots \\ \frac{\partial f}{\partial x_m}(x) \end{bmatrix},$$

$$\nabla^2 f(x)\delta x = \begin{bmatrix} \frac{\partial f}{\partial x_{11}}(x) & \dots & \frac{\partial f}{\partial x_{1m}}(x) \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial x_{m1}}(x) & \dots & \frac{\partial f}{\partial x_{mm}}(x) \end{bmatrix} \delta x.$$

In code, we specify this function as:

**Language** C++

**Structure** `Optizelle::ScalarValuedFunction`

**Interface** Inheritance

**Code**

```
namespace Optizelle{
    // A scalar valued function interface, f : X -> R
    template <
        typename Real,
        template <typename> class XX
```

```
    >
    struct ScalarValuedFunction {
        // Create some type shortcuts
        typedef XX <Real> X;
        typedef typename X::Vector Vector;

        // <- f(x)
        virtual Real eval(Vector const & x) const = 0;

        // grad = grad f(x)
        virtual void grad(Vector const & x,Vector & grad) const = 0;

        // H_dx = hess f(x) dx
        virtual void hessvec(Vector const & x,Vector const & dx,Vector & H_dx)
            const = 0;

        // Allow a derived class to deallocate memory
        virtual ~ScalarValuedFunction() {}
    };
}
```

**Language** Python

**Structure** `Optizelle.ScalarValuedFunction`

**Interface** Inheritance

**Code**

```
class ScalarValuedFunction(object):
    """A simple scalar valued function interface, f : X -> R"""

    def _err(self,fn):
        """Produces an error message for an undefined function"""
        raise Exception.t("%s function is not defined in a " % (fn) +
            "ScalarValuedFunction")

    def eval(self,x):
        """<- f(x)"""
        _err(self,"eval")

    def grad(self,x,grad):
        """<- grad f(x)"""
        _err(self,"grad")

    def hessvec(self,x,dx,H_dx):
        """<- hess f(x) dx"""
        _err(self,"grad")
```

**Language** MATLAB/Octave

**Structure** `Optizelle.ScalarValuedFunction`

**Interface** Members present

**Code**

```
% A simple scalar valued function interface, f : X -> R
err_svf=@(x)error(sprintf( ...
    'The %s function is not defined in a ScalarValuedFunction.',x));
Optizelle.ScalarValuedFunction = struct( ...
    'eval',@(x)err_svf('eval'), ...
    'grad',@(x)err_svf('grad'), ...
    'hess_vec',@(x,dx)err_svf('hess_vec'));
```

Note, we require that the Hessian-vector product always be present. If one is not available, we simply return zero. As an example, in our Rosenbrock example, we minimize the function $f : \mathbb{R}^2 \to \mathbb{R}$ where

$$f(x) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2.$$

This function has a gradient of

$$\nabla f(x) = \begin{bmatrix} -400x_1(x_2 - x_1^2) - 2(1 - x_1) \\ 200(x_2 - x_1^2) \end{bmatrix}$$

and Hessian-vector product of

$$\nabla^2 f(x)\delta x = \begin{bmatrix} 1200x_1^2 - 400x_2 + 2 & -400x_1 \\ -400x_1 & 200 \end{bmatrix} \delta x.$$

Using Optizelle's internal vector spaces, we implement these functions as:

**Language** C++

**Code**

```
// Squares its input
template <typename Real>
Real sq(Real x){
    return x*x;
}

// Define the Rosenbrock function where
//
// f(x,y)=(1-x)^2+100(y-x^2)^2
//
struct Rosenbrock
    : public Optizelle::ScalarValuedFunction <double,Optizelle::Rm>
{
    typedef Optizelle::Rm <double> X;

    // Evaluation of the Rosenbrock function
    double eval(X::Vector const & x) const {
        return sq(1.-x[0])+100.*sq(x[1]-sq(x[0]));
    }

    // Gradient
    void grad(
        X::Vector const & x,
        X::Vector & grad
    ) const {
        grad[0]=-400.*x[0]*(x[1]-sq(x[0]))-2.*(1.-x[0]);
        grad[1]=200.*(x[1]-sq(x[0]));
    }

    // Hessian-vector product
    void hessvec(
        X::Vector const & x,
```

```
        X::Vector const & dx,
        X::Vector & H_dx
    ) const {
        H_dx[0]=(1200.*sq(x[0])-400.*x[1]+2)*dx[0]-400.*x[0]*dx[1];
        H_dx[1]=-400.*x[0]*dx[0]+200.*dx[1];
    }
};
```

**Language** Python

**Code**

```
# Squares its input
sq = lambda x:x*x

# Define the Rosenbrock function where
#
# f(x,y)=(1-x)^2+100(y-x^2)^2
#
class Rosenbrock(Optizelle.ScalarValuedFunction):
    # Evaluation of the Rosenbrock function
    def eval(self,x):
        return sq(1.-x[0])+100.*sq(x[1]-sq(x[0]))

    # Gradient
    def grad(self,x,grad):
        grad[0]=-400*x[0]*(x[1]-sq(x[0]))-2*(1-x[0])
        grad[1]=200*(x[1]-sq(x[0]))

    # Hessian-vector product
    def hessvec(self,x,dx,H_dx):
        H_dx[0] = (1200*sq(x[0])-400*x[1]+2)*dx[0]-400*x[0]*dx[1]
        H_dx[1] = -400*x[0]*dx[0] + 200*dx[1]
```

**Language** MATLAB/Octave

**Code**

```
% Squares its input
function z = sq(x)
    z=x*x;
end

% Define the Rosenbrock function where
%
% f(x,y)=(1-x)^2+100(y-x^2)^2
%
function self = Rosenbrock()

    % Evaluation of the Rosenbrock function
    self.eval = @(x) sq(1.-x(1))+100.*sq(x(2)-sq(x(1)));

    % Gradient
    self.grad = @(x) [
        -400.*x(1)*(x(2)-sq(x(1)))-2.*(1.-x(1));
        200.*(x(2)-sq(x(1)))];

    % Hessian-vector product
```

```
    self.hessvec = @(x,dx) [
        (1200.*sq(x(1))-400.*x(2)+2)*dx(1)-400.*x(1)*dx(2);
        -400.*x(1)*dx(1)+200.*dx(2)];
end
```

In our Simple equality constrained example, we have an objective $f : \mathbb{R}^2 \to \mathbb{R}$ where

$$f(x) = x_1^2 + x_2^2$$

This function has a gradient of

$$\nabla f(x) = \begin{bmatrix} 2x_1 \\ 2x_2 \end{bmatrix}$$

and Hessian-vector product of

$$\nabla^2 f(x)\delta x = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \delta x.$$

We implement this function with the code:

**Language** C++

**Code**

```cpp
// Squares its input
template <typename Real>
Real sq(Real const & x){
    return x*x;
}

// Define a simple objective where
//
// f(x,y)=x^2+y^2
//
struct MyObj
    : public Optizelle::ScalarValuedFunction <double,Optizelle::Rm>
{
    typedef Optizelle::Rm <double> X;

    // Evaluation
    double eval(X::Vector const & x) const {
        return sq(x[0])+sq(x[1]);
    }

    // Gradient
    void grad(
        X::Vector const & x,
        X::Vector & grad
    ) const {
        grad[0]=2.*x[0];
        grad[1]=2.*x[1];
    }

    // Hessian-vector product
    void hessvec(
        X::Vector const & x,
        X::Vector const & dx,
        X::Vector & H_dx
    ) const {
        H_dx[0]=2.*dx[0];
        H_dx[1]=2.*dx[1];
    }
};
```

**Language** Python

**Code**

```
# Squares its input
sq = lambda x:x*x

# Define a simple objective where
#
# f(x,y)=x^2+y^2
#
class MyObj(Optizelle.ScalarValuedFunction):

    # Evaluation
    def eval(self,x):
        return sq(x[0])+sq(x[1])

    # Gradient
    def grad(self,x,grad):
        grad[0]=2.*x[0]
        grad[1]=2.*x[1]

    # Hessian-vector product
    def hessvec(self,x,dx,H_dx):
        H_dx[0]=2.*dx[0]
        H_dx[1]=2.*dx[1]
```

**Language** MATLAB/Octave

**Code**

```
% Squares its input
function z = sq(x)
    z=x*x;
end

% Define a simple objective where
%
% f(x,y)=x^2+y^2
%
function self = MyObj()

    % Evaluation
    self.eval = @(x) sq(x(1))+sq(x(2));

    % Gradient
    self.grad = @(x) [ ...
        2.*x(1); ...
        2.*x(2)];

    % Hessian-vector product
    self.hessvec = @(x,dx) [ ...
        2.*dx(1); ...
        2.*dx(2)];
end
```

## 3.4 (Optional) Define the constraints

In the optimization problems

| **Unconstrained** | **Equality Constrained** |
|---|---|
| $\min_{x\in X} \; f(x)$ | $\min_{x\in X} \; f(x)$ st $g(x)=0$ |
| **Inequality Constrained** | **Constrained** |
| $\min_{x\in X} \; f(x)$ st $h(x)\succeq 0$ | $\min_{x\in X} \; f(x)$ st $g(x)=0$, $h(x)\succeq 0$ |

the vector-valued functions $g : X \to Y$ and $h : X \to Z$ denote the *equality* and *inequality constraints*, respectfully. Here, we allow the equality constraints to be nonlinear, but require that the inequality constraints be affine. Recall, an affine function is one where $h(\alpha x + (1-\alpha)x) = \alpha h(x) + (1-\alpha)h(y)$ for all $\alpha \in \mathbb{R}$ or equivalently where $h''(x) = 0$. We require affine inequality constraints in order to simplify the line search that maintains the nonnegativity of the inequality constraints. In case we have a nonlinear inequality constraint, we must reformulate the problem in order to make it affine. The easiest method for doing so is through the reformulations

$$\left.\begin{array}{rl} \min\limits_{x\in X} & f(x) \\ \text{st} & h(x)\succeq 0 \end{array}\right\} \rightsquigarrow \left\{\begin{array}{rl} \min\limits_{x\in X, z\in Z} & f(x) \\ \text{st} & h(x)-z=0 \\ & z\succeq 0 \end{array}\right.$$

and

$$\left.\begin{array}{rl} \min\limits_{x\in X} & f(x) \\ \text{st} & g(x)=0 \\ & h(x)\succeq 0 \end{array}\right\} \rightsquigarrow \left\{\begin{array}{rl} \min\limits_{x\in X, z\in Z} & f(x) \\ \text{st} & \begin{bmatrix} g(x) \\ h(x)-z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ & z\succeq 0. \end{array}\right.$$

Similar to the objective function, we require derivative information in order optimize effectively. Specifically, we require the evaluation, $g(x)$, Fréchet (total) derivative applied to a vector, $g'(x)\delta x$, and the adjoint of the Fréchet derivative applied to a vector, $g'(x)^*\delta y$. In order to use second order algorithms, we also require the second derivative operation $(g''(x)\delta x)^*\delta y$. Note, we require the same operations from $h$, but since $h$ is affine, $(h''(x)\delta x)^*\delta z = 0$. In the case that $g : \mathbb{R}^m \to \mathbb{R}^n$ and we use the inner product $\langle x, y\rangle = x^T y$ for both $\mathbb{R}^m$ and $\mathbb{R}^n$, the derivation of these derivatives is quite simple. Here, we write $g$ as

$$g(x) = \begin{bmatrix} g_1(x) \\ \vdots \\ g_n(x) \end{bmatrix}.$$

This means that we have

$$\begin{aligned} g'(x)\delta x &= \begin{bmatrix} \nabla g_1(x)^T \\ \vdots \\ \nabla g_n(x)^T \end{bmatrix} \delta x \\ g'(x)^*\delta y &= \begin{bmatrix} \nabla g_1(x) & \dots & \nabla g_n(x) \end{bmatrix} \delta y \\ (g''(x)\delta x)^*\delta y &= \sum_{i=1}^n \delta y_i \nabla^2 g_i(x)\delta x. \end{aligned}$$

In code, these derivatives become:

**Language** C++

**Structure** `Optizelle::VectorValuedFunction`

**Interface** Inheritance

**Code**

```
namespace Optizelle{
    // A vector valued function interface, f : X -> Y
    template <
        typename Real,
        template <typename> class XX,
        template <typename> class YY
    >
    struct VectorValuedFunction {
        // Create some type shortcuts
        typedef XX <Real> X;
        typedef typename X::Vector X_Vector;
        typedef YY <Real> Y;
        typedef typename Y::Vector Y_Vector;

        // y=f(x)
        virtual void eval(X_Vector const & x,Y_Vector & y) const = 0;

         // y=f'(x)dx
         virtual void p(
             X_Vector const & x,
             X_Vector const & dx,
             Y_Vector & y
         ) const = 0;

         // z=f'(x)*dy
         virtual void ps(
             X_Vector const & x,
             Y_Vector const & dy,
             X_Vector & z
         ) const= 0;

         // z=(f''(x)dx)*dy
         virtual void pps(
             X_Vector const & x,
             X_Vector const & dx,
             Y_Vector const & dy,
             X_Vector & z
         ) const = 0;

         // Allow a derived class to deallocate memory
         virtual ~VectorValuedFunction() {}
    };
}
```

**Language** Python

**Structure** `Optizelle.VectorValuedFunction`

**Interface** Inheritance

**Code**

```
class VectorValuedFunction(object):
    """A vector valued function interface, f : X -> Y"""

    def _err(self,fn):
        """Produces an error message for an undefined function"""
        raise Exception.t("%s function is not defined in a " % (fn) +
```

```
                "VectorValuedFunction")

    def eval(self,x,y):
        """y <- f(x)"""
        _err(self,"eval")

    def p(self,x,dx,y):
        """y <- f'(x)dx"""
        _err(self,"p")

    def ps(self,x,dx,z):
        """z <- f'(x)dx"""
        _err(self,"ps")

    def pps(self,x,dx,dy,z):
        """z <- (f''(x)dx)*dy"""
        _err(self,"pps")
```

**Language** MATLAB/Octave

**Structure** `Optizelle.VectorValuedFunction`

**Interface** Members present

**Code**

```
% A vector valued function interface, f : X -> Y
err_vvf=@(x)error(sprintf( ...
    'The %s function is not defined in a VectorValuedFunction.',x));
Optizelle.VectorValuedFunction = struct( ...
    'eval',@(x)err_vvf('eval'), ...
    'p',@(x,dx)err_vvf('p'), ...
    'ps',@(x,dy)err_vvf('ps'), ...
    'pps',@(x,dx,dy)err_vvf('pps'));
```

Note, we require that the second derivative always be present. If one is not available, we simply return zero. For example, in our Simple equality constrained example, we define a simple equality constraint as

$$g(x) = \left[(x_1-2)^2 + (x_2-2)^2 - 1\right].$$

Then, we have that

$$\begin{aligned} g'(x)\delta x =& \begin{bmatrix} 2(x_1-2) & 2(x_2-2) \end{bmatrix} \delta x \\ g'(x)^*\delta y =& \begin{bmatrix} 2(x_1-2) \\ 2(x_2-2) \end{bmatrix} \delta y \\ (g''(x)\delta x)^*\delta y =& \delta y \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \delta x \end{aligned}$$

Using Optizelle's internal vector spaces, we implement these functions as:

**Language** C++

**Code**

```
// Define a simple equality constraint
//
// g(x,y)= [ (x-2)^2 + (y-2)^2 = 1 ]
//
struct MyEq
    :public Optizelle::VectorValuedFunction<double,Optizelle::Rm,Optizelle::Rm>
{
```

```
    typedef Optizelle::Rm <double> X;
    typedef Optizelle::Rm <double> Y;

    // y=g(x)
    void eval(
        X::Vector const & x,
        Y::Vector & y
    ) const {
        y[0] = sq(x[0]-2.)+sq(x[1]-2.)-1.;
    }

    // y=g'(x)dx
    void p(
        X::Vector const & x,
        X::Vector const & dx,
        Y::Vector & y
    ) const {
        y[0] = 2.*(x[0]-2.)*dx[0]+2.*(x[1]-2.)*dx[1];
    }

    // xhat=g'(x)*dy
    void ps(
        X::Vector const & x,
        Y::Vector const & dy,
        X::Vector & xhat
    ) const {
        xhat[0] = 2.*(x[0]-2.)*dy[0];
        xhat[1] = 2.*(x[1]-2.)*dy[0];
    }

    // xhat=(g''(x)dx)*dy
    void pps(
        X::Vector const & x,
        X::Vector const & dx,
        Y::Vector const & dy,
        X::Vector & xhat
    ) const {
        xhat[0] = 2.*dx[0]*dy[0];
        xhat[1] = 2.*dx[1]*dy[0];
    }
};
```

**Language** Python

**Code**

```
# Define a simple equality constraint
#
# g(x,y)= [ (x-2)^2 + (y-2)^2 = 1 ]
#
class MyEq(Optizelle.VectorValuedFunction):

    # y=g(x)
    def eval(self,x,y):
        y[0] = sq(x[0]-2.)+sq(x[1]-2.)-1.

    # y=g'(x)dx
    def p(self,x,dx,y):
```

```
        y[0] = 2.*(x[0]-2.)*dx[0]+2.*(x[1]-2.)*dx[1]

    # xhat=g'(x)*dy
    def ps(self,x,dy,xhat):
        xhat[0] = 2.*(x[0]-2.)*dy[0]
        xhat[1] = 2.*(x[1]-2.)*dy[0]

    # xhat=(g''(x)dx)*dy
    def pps(self,x,dx,dy,xhat):
        xhat[0] = 2.*dx[0]*dy[0]
        xhat[1] = 2.*dx[1]*dy[0]
```

**Language** MATLAB/Octave

**Code**

```
% Define a simple equality constraint
%
% g(x,y)= [ (x-2)^2 + (y-2)^2 = 1 ]
%
function self = MyEq()

    % y=g(x)
    self.eval = @(x) [ ...
        sq(x(1)-2.)+sq(x(2)-2.)-1.];

    % y=g'(x)dx
    self.p = @(x,dx) [ ...
        2.*(x(1)-2.)*dx(1)+2.*(x(2)-2.)*dx(2)];

    % xhat=g'(x)*dy
    self.ps = @(x,dy) [ ...
        2.*(x(1)-2.)*dy(1); ...
        2.*(x(2)-2.)*dy(1)];

    % xhat=(g''(x)dx)*dy
    self.pps = @(x,dx,dy) [ ...
        2.*dx(1)*dy(1); ...
        2.*dx(2)*dy(1) ];
end
```

## 3.5 (Optional) Define the preconditioners

Since Optizelle is fully matrix-free, its performance depends highly on the quality of the preconditioners provided to it by the user. To that end, there are two places where preconditioning matters: the Hessian of the objective function and a KKT system that relates to the equality constraints. Specifically, we benefit when we can define $P_H : X \to X$ such that

$$P_H \approx \nabla^2 f(x)^{-1}$$

and $P_l : Y \to Y$ along with $P_r : Y \to Y$ such that

$$P_l(g'(x)g'(x)^*)P_r \approx I.$$

Before we discuss these operators in detail, let us emphasize two points. First, as we describe below, we require only the action of this preconditioner on a vector. This enables Optizelle to continue to be matrix-free. Second, even though we use matrix-free abstractions, most of the time, we're better off just using matrices.

At this point in time, both dense and sparse linear algebra libraries are extremely fast. Unless we have a large PDE constrained optimization problem, just form a matrix of the operator, factor it, and move on.

In the objective function, we use a preconditioner for the Hessian in several different places. Foremost, we use it to precondition linear systems related to second-order algorithms such as Newton's method. In addition, we use it within first-order algorithms such as nonlinear-CG and steepest descent. Certainly, $\nabla^2 f(x)^{-1}$ represents the best such preconditioner, but the Hessian may become singular during the course of optimization, so we must take care in how we generate this preconditioner. As such, even though $\nabla^2 f(x)$ is self-adjoint, the LU factorization provides a simple, effective manner to factorize the Hessian. In other words, we find operators $L$ and $U$ such that

$$LU = \nabla^2 f(x).$$

Then, our preconditioner $P_H : X \to X$ approximates

$$P_H \delta x \approx U^{-1} L^{-1} \delta x.$$

We say approximate because either $U^{-1}$ or $L^{-1}$ may not exist. In this case, we note that the action of $U^{-1}$ and $L^{-1}$ on a vector denotes a back and forward solve, respectively. When the inverse does not exist, we can simply modify these solves to ignore any variables that cause problems. As a note, we only benefit from a Hessian preconditioner in unconstrained and inequality constrained problems. For problems with equality constraints, we use a composite-step SQP method. Here, the tangential subproblem requires a null-space projection that replaces the Hessian preconditioner. If preconditioning the quantities in the objective is important to the performance of the problem, then we need to reformulate the problem, so that these quantities appear as equality constraints and then use an appropriate Schur preconditioner below. For example, we can reformulate the problem

$$\min_{x\in X}\{f(x) : g(x) = 0\}$$

as

$$\min_{x\in X, x_0\in\mathbb{R}}\{x_0 : x_0 = f(x), g(x) = 0\}.$$

Note, this transformation may destroy convexity of the problem, so a different transformation may be more appropriate. For the equality constraints, our algorithms require the repeated solution of a system whose operator is

$$\begin{bmatrix} I & g'(x)^* \\ g'(x) & 0 \end{bmatrix}.$$

As it happens, if $g'(x)$ is full-rank, the preconditioner

$$\begin{bmatrix} I & 0 \\ 0 & (g'(x)g'(x)^*)^{-1} \end{bmatrix}$$

allows a Krylov method to solve the above system in three iterations. As such, Optizelle focuses on preconditioning the operator

$$g'(x)g'(x)^*.$$

Note, unlike the Hessian, we allow both left and right preconditioners for this operator. In addition, this operator depends on the inner product used by the vector space because it involves an adjoint. If we're working in $\mathbb{R}^m$ with the inner product $\langle x, y\rangle = x^T y$, we can ignore this nuance. Otherwise, we must modify our factorizations to correctly account for the change in inner product. Outside of this difficulty, we note the operator is always symmetric and positive-semidefinite. However, like the Hessian, it can and likely will become singular during the course of optimization. As such, we propose two ways of dealing with this. In one case, we use a $QR$ factorization of $g'(x)^*$,

$$QR = g'(x)^*,$$

then form the preconditioners $P_l : Y \to Y$ and $P_r : Y \to Y$ where

$$\begin{aligned} P_l \delta x \approx& R^{-1} R^{-*} \delta x, \\ P_r \delta x =& \delta x. \end{aligned}$$

Again, we must take care in case $R$ is singular. Alternatively, we can just form $g'(x)g'(x)^*$ and find its $LU$ factorization like we do with the Hessian,

$$LU = g'(x)g'(x)^*.$$

This gives us the preconditioners

$$P_l\delta x \approx U^{-1}L^{-1}\delta x,$$
$$P_r\delta x = \delta x.$$

In theory, we can use a Choleski factorization to solve this system. The problem with this approach is that the Choleski factorization will fail when $g'(x)$ is not full rank. Generally, we find it easier to fix a failing forward or back solve, as is the case with a QR or LU factorization, than to fix a failing factorization. In code, we represent preconditioners as a generic linear operator:

**Language** C++

**Structure** `Optizelle::Operator`

**Interface** Inheritance

**Code**

```
namespace Optizelle {
    // A linear operator specification, A : X->Y
    template <
        typename Real,
        template <typename> class X,
        template <typename> class Y
    >
    struct Operator {
        // Create some type shortcuts
        typedef typename X <Real>::Vector X_Vector;
        typedef typename Y <Real>::Vector Y_Vector;

        // y = A(x)
        virtual void eval(X_Vector const & x,Y_Vector &y) const = 0;

        // Allow a derived class to deallocate memory
        virtual ~Operator() {}
    };
}
```

**Language** Python

**Structure** `Optizelle.Operator`

**Interface** Inheritance

**Code**

```
class Operator(object):
    """A linear operator specification, A : X->Y"""

    def _err(self,fn):
        """Produces an error message for an undefined function"""
        raise Exception.t("%s function is not defined in an " % (fn) +
            "Operator")

    def eval(self,state,x,y):
        """y <- A(x)"""
        _err(self,"eval")
```

**Language** MATLAB/Octave

**Structure** `Optizelle.Operator`

**Interface** Members present

**Code**

```
% A linear operator specification, A : X->Y
err_op=@(x)error(sprintf( ...
    'The %s function is not defined in an Operator.',x));
Optizelle.Operator = struct( ...
    'eval',@(state,x)err_op('eval'));
```

As we can see, there is a slight difference when we compare C++ to Python and MATLAB/Octave. In Python and MATLAB/Octave, we provide the preconditioner with the variable `state` that we describe in the section Create the optimization state. We omit this variable in C++. If we need access to the state in C++, we can simply pass in a reference to it during the operator's construction. In Python and MATLAB/Octave, this is not an option, so we must pass the state directly. To be clear, access to the variable `state` is important for most preconditioners. Recall, we must either evaluate an approximation to $\nabla^2 f(x)^{-1}\delta x$ or $(g'(x)g'(x)^*)^{-1}\delta y$. When Optizelle calls the preconditioner, it provides $\delta x$ and $\delta y$ and expects $P_H\delta x$, $P_l\delta y$, and $P_r\delta y$ as its return. Optizelle does **not** call the preconditioner on the variables $x$ and $y$. If we want access to these variables, we must find them in the state. As another important note, Optizelle can not optimize user defined factorizations. Meaning, during the course of an optimization iteration, we call these preconditioners several different times at the same optimization iterate, $x$. As such, if we factorize $\nabla^2 f(x)$ or $g'(x)g'(x)^*$, it is critical to our performance that we cache these factorizations. The easiest way to tell when a new factorization is needed is to monitor the variable `x` inside of `state`. This variable represents the current optimization iterate and it does not change until we take a new step in the optimization algorithms. Recall, in our Rosenbrock example, we have a Hessian-vector product of

$$\nabla^2 f(x)\delta x = \begin{bmatrix} 1200x_1^2 - 400x_2 + 2 & -400x_1 \\ -400x_1 & 200 \end{bmatrix} \delta x.$$

This allows us to find the inverse using Cramer's rule

$$\nabla^2 f(x)^{-1}\delta x = \frac{1}{80000x_1^2 - 80000x_2 + 400} \begin{bmatrix} 200 & 400x_1 \\ 400x_1 & 1200x_1^2 - 400x_2 + 2 \end{bmatrix} \delta x.$$

Generally, we claim using Cramer's rule is a bad idea when compared to an LU factorization, but it works fine on this small example. Using this formulation, we define our preconditioner to the Hessian with the code:

**Language** C++

**Code**

```
// Define a perfect preconditioner for the Hessian
struct RosenHInv :
    public Optizelle::Operator <double,Optizelle::Rm,Optizelle::Rm>
{
public:
    typedef Optizelle::Rm <double> X;
    typedef X::Vector X_Vector;
private:
    X_Vector& x;
public:
    RosenHInv(X::Vector& x_) : x(x_) {}
    void eval(X_Vector const & dx,X_Vector & result) const {
        auto one_over_det=1./(80000.*sq(x[0])-80000.*x[1]+400.);
        result[0]=one_over_det*(200.*dx[0]+400.*x[0]*dx[1]);
        result[1]=one_over_det*
            (400.*x[0]*dx[0]+(1200.*x[0]*x[0]-400.*x[1]+2.)*dx[1]);
    }
};
```

**Language** Python

**Code**

```
# Define a perfect preconditioner for the Hessian
class RosenHInv(Optizelle.Operator):
    def eval(self,state,dx,result):
        x = state.x
        one_over_det=1./(80000.*sq(x[0])-80000.*x[1]+400.)
        result[0]=one_over_det*(200.*dx[0]+400.*x[0]*dx[1])
        result[1]=(one_over_det*
            (400.*x[0]*dx[0]+(1200.*x[0]*x[0]-400.*x[1]+2.)*dx[1]))
```

**Language** MATLAB/Octave

**Code**

```
% Define a perfect preconditioner for the Hessian
function self = RosenHInv()
    self.eval = @(state,dx) eval(state,dx);
end
function result = eval(state,dx)
    x = state.x;
    one_over_det=1./(80000.*sq(x(1))-80000.*x(2)+400.);
    result = [
        one_over_det*(200.*dx(1)+400.*x(1)*dx(2));
        one_over_det*...
            (400.*x(1)*dx(1)+(1200.*x(1)*x(1)-400.*x(2)+2.)*dx(2))];
end
```

For our simple equality constraint

$$g(x) = \left[(x_1-2)^2+(x_2-2)^2-1\right].$$

We have that

$$g'(x)\delta x = \begin{bmatrix}2(x_1-2) & 2(x_2-2)\end{bmatrix}\delta x$$
$$g'(x)^*\delta y = \begin{bmatrix}2(x_1-2)\\2(x_2-2)\end{bmatrix}\delta y.$$

This means that

$$g'(x)g'(x)^*\delta y = (4(x_1-2)^2+4(x_2-2)^2)\delta y$$

and we have a perfect preconditioner

$$(g'(x)g'(x)^*)^{-1}\delta y = \frac{1}{4(x_1-2)^2+4(x_2-2)^2}\delta y.$$

We implement this in our Simple equality constrained example with the code:

**Language** C++

**Code**

```
// Define a Schur preconditioner for the equality constraints
struct MyPrecon:
    public Optizelle::Operator <double,Optizelle::Rm,Optizelle::Rm>
{
public:
    typedef Optizelle::Rm <double> X;
    typedef X::Vector X_Vector;
    typedef Optizelle::Rm <double> Y;
    typedef Y::Vector Y_Vector;
private:
```

```
    X_Vector& x;
public:
    MyPrecon(X::Vector& x_) : x(x_) {}
    void eval(Y_Vector const & dy,Y_Vector & result) const {
        result[0]=dy[0]/sq(4.*(x[0]-2.)+4.*sq(x[1]-2.));
    }
};
```

**Language** Python

**Code**

```
# Define a Schur preconditioner for the equality constraints
class MyPrecon(Optizelle.Operator):
    def eval(self,state,dy,result):
        result[0]=dy[0]/sq(4.*(x[0]-2.)+4.*sq(x[1]-2.))
```

**Language** MATLAB/Octave

**Code**

```
% Define a Schur preconditioner for the equality constraints
function self = MyPrecon()
    self.eval=@(state,dy)dy(1)/sq(4.*(state.x(1)-2.)+4.*sq(state.x(2)-2.));
end
```

## 3.6 Create the optimization state

In Optizelle, the optimization state contains an entire description of the current state of the optimization algorithm. This is unique to the particular optimization formulation, but all algorithms in a particular formulations share the same state. Most algorithms do not require information about all of these pieces, but they are present to make it easier to switch from one algorithm to another. For example, trust-region and line-search algorithms share several components, but the trust-region radius is unique to trust-region algorithms and the line-search step length is unique to line-search algorithms. Nevertheless, we may want to switch from one algorithm to another, so they share the same components. In order to define an optimization state, we instantiate the state class within the particular class of formulation we require. The syntax is:

**Language** C++

**Code**

```
Optizelle::Unconstrained <Real,XX>::State::t state(x);

Optizelle::EqualityConstrained <Real,XX,YY>::State::t state(x,y);

Optizelle::InequalityConstrained <Real,XX,ZZ>::State::t state(x,z);

Optizelle::Constrained <Real,XX,YY,ZZ>::State::t state(x,y,z);
```

**Language** Python

**Code**

```
state = Optizelle.Unconstrained.State.t(XX,x)

state = Optizelle.EqualityConstrained.State.t(XX,YY,x,y)

state = Optizelle.InequalityConstrained.State.t(XX,ZZ,x,z)

state = Optizelle.Constrained.State.t(XX,YY,ZZ,x,y,z)
```

**Language** MATLAB/Octave

**Code**

```
state = Optizelle.Unconstrained.State.t(XX,x);

state = Optizelle.EqualityConstrained.State.t(XX,YY,x,y);

state = Optizelle.InequalityConstrained.State.t(XX,ZZ,x,z);

state = Optizelle.Constrained.State.t(XX,YY,ZZ,x,y,z);
```

Here, `XX`, `YY`, and ZZ correspond to the vector spaces $X$, $Y$, and $Z$ described in the section Import or define the appropriate vector spaces. Likely, they are just `Optizelle::Rm` or `Optizelle.Rm`. Next, the variable `x` denotes an initial guess for the optimization problem. This guess is very important to the performance of the algorithms, so choose wisely. The variables `y` and `z` represent arbitrary elements in the codomain of $g$ and $h$, respectively. We do not use the values of these variables, so any properly allocated vector works fine. As an example, we create the optimization state in the Rosenbrock example with the following code:

**Language** C++

**Code**

```
// Generate an initial guess for Rosenbrock
auto x = std::vector <double> {-1.2, 1.};

// Create an unconstrained state based on this vector
Optizelle::Unconstrained <double,Optizelle::Rm>::State::t state(x);
```

**Language** Python

**Code**

```
# Generate an initial guess for Rosenbrock
x = numpy.array([-1.2,1.0])

# Create an unconstrained state based on this vector
state=Optizelle.Unconstrained.State.t(Optizelle.Rm,x)
```

**Language** MATLAB/Octave

**Code**

```
% Generate an initial guess for Rosenbrock
x = [-1.2;1.];

% Create an unconstrained state based on this vector
state=Optizelle.Unconstrained.State.t(Optizelle.Rm,x);
```

In our Simple equality constrained example, we have:

**Language** C++

**Code**

```
// Generate an initial guess
auto x = std::vector <double> {2.1, 1.1};

// Allocate memory for the equality multiplier
auto y = std::vector <double> (1);

// Create an optimization state
Optizelle::EqualityConstrained <double,Rm,Rm>::State::t state(x,y);
```

**Language** Python

**Code**

```
# Generate an initial guess
x = numpy.array([2.1,1.1])

# Allocate memory for the equality multiplier
y = numpy.array([0.])

# Create an optimization state
state=Optizelle.EqualityConstrained.State.t(Optizelle.Rm,Optizelle.Rm,x,y)
```

**Language** MATLAB/Octave

**Code**

```
% Generate an initial guess
x = [2.1;1.1];

% Allocate memory for the equality multiplier
y = [0.];

% Create an optimization state
state= Optizelle.EqualityConstrained.State.t(Optizelle.Rm,Optizelle.Rm,x,y);
```

## 3.7 Set the optimization parameters

For each optimization problem, the parameters required for an efficient optimization solve can vary wildly. Nevertheless, the parameters that guide this process reside within the state object. There are two mechanisms for modifying these entries. First, the state object created in the section Create the optimization state is simply an object with a variety of elements that can be modified directly. Alternatively, and preferably, we can use the JSON reader. The syntax for reading a parameter file in JSON format from file is:

**Language** C++

**Code**

```
Optizelle::json::Unconstrained <Real,XX>::read(fname,state);

Optizelle::json::EqualityConstrained <Real,XX,YY>::read(fname,state);

Optizelle::json::InequalityConstrained <Real,XX,ZZ>::read(fname,state);

Optizelle::json::Constrained <Real,XX,YY,ZZ>::read(fname,state);
```

**Language** Python

**Code**

```
Optizelle.json.Unconstrained.read(XX,fname,state)

Optizelle.json.EqualityConstrained.read(XX,YY,fname,state)

Optizelle.json.InequalityConstrained.read(XX,ZZ,fname,state)

Optizelle.json.Constrained.read(XX,YY,ZZ,fname,state)
```

**Language** MATLAB/Octave

**Code**

```
state = Optizelle.json.Unconstrained.read(XX,fname,state);

state = Optizelle.json.EqualityConstrained.read(XX,YY,fname,state);

state = Optizelle.json.InequalityConstrained.read(XX,ZZ,fname,state);

state = Optizelle.json.Constrained.read(XX,YY,ZZ,fname,state);
```

Here, most of the parameters required are identical to those required in the section Create the optimization state. The lone, new parameter is `fname`, which corresponds to a string of the file name where we read the JSON formatted parameters. As to what these parameters are, we discuss that in the chapter Optimization parameters. In our Rosenbrock example, we use the following code to read the optimization parameters:

**Language** C++

**Code**

```
// Read the parameters from file
Optizelle::json::Unconstrained <double,Optizelle::Rm>::read(fname,state);
```

**Language** Python

**Code**

```
# Read the parameters from file
Optizelle.json.Unconstrained.read(Optizelle.Rm,fname,state)
```

**Language** MATLAB/Octave

**Code**

```
% Read the parameters from file
state=Optizelle.json.Unconstrained.read(Optizelle.Rm,fname,state);
```

This becomes the following in our Simple equality constrained example:

**Language** C++

**Code**

```
// Read the parameters from file
Optizelle::json::EqualityConstrained <double,Optizelle::Rm,Optizelle::Rm>
    ::read(fname,state);
```

**Language** Python

**Code**

```
# Read the parameters from file
Optizelle.json.EqualityConstrained.read(Optizelle.Rm,Optizelle.Rm,fname,state)
```

**Language** MATLAB/Octave

**Code**

```
% Read the parameters from file
state = Optizelle.json.EqualityConstrained.read( ...
    Optizelle.Rm,Optizelle.Rm,fname,state);
```

## 3.8 Accumulate the functions

In order to pass the functions used in the optimization to Optizelle, we accumulate each of them into a bundle of functions. These bundles are simple structures that contain the appropriate function. The syntax for creating these objects is:

**Language** C++

**Code**

```
Optizelle::Unconstrained <Real,XX>::Functions::t fns;

Optizelle::EqualityConstrained <Real,XX,YY>::Functions::t fns;

Optizelle::InequalityConstrained <Real,XX,ZZ>::Functions::t fns;

Optizelle::Constrained <Real,XX,YY,ZZ>::Functions::t fns;
```

**Language** Python

**Code**

```
fns = Optizelle.Unconstrained.Functions.t()

fns = Optizelle.EqualityConstrained.Functions.t()

fns = Optizelle.InequalityConstrained.Functions.t()

fns = Optizelle.Constrained.Functions.t()
```

**Language** MATLAB/Octave

**Code**

```
fns = Optizelle.Unconstrained.Functions.t;

fns = Optizelle.EqualityConstrained.Functions.t;

fns = Optizelle.InequalityConstrained.Functions.t;

fns = Optizelle.Constrained.Functions.t;
```

As was the case in the section Create the optimization state, `XX`, `YY`, and ZZ correspond to the vector spaces $X$, $Y$, and $Z$ described in the section Import or define the appropriate vector spaces. Likely, they are just `Optizelle::Rm` or `Optizelle.Rm`. Now, each of structures contains a number of required and optional elements. We summarize these elements as follows:

**Element** `f`

**Type** `ScalarValuedFunction`

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**Required** Yes

**Description** Objective function.

**Element** `PH`

**Type** `Operator`

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**Required** No

**Description** Preconditioner for the Hessian of the objective function, $\nabla^2 f(x)$.

**Element** `g`

**Type** `VectorValuedFunction`

**Problem Class** Equality Constrained, Constrained

**Required** Yes

**Description** Equality constraint.

**Element** `PSchur_left`

**Type** `Operator`

**Problem Class** Equality Constrained, Constrained

**Required** No

**Description** Left Schur preconditioner for derivative of the equality constraint, $g'(x)g'(x)^*$.

**Element** `PSchur_right`

**Type** `Operator`

**Problem Class** Equality Constrained, Constrained

**Required** No

**Description** Right Schur preconditioner for derivative of the equality constraint, $g'(x)g'(x)^*$.

**Element** `h`

**Type** `VectorValuedFunction`

**Problem Class** Inequality Constrained, Constrained

**Required** Yes

**Description** Inequality constraint.

In C++, we represent each of the these elements as a `std::unique_ptr` using the type specified above. In Python, we use simple class elements. In MATLAB/Octave, we use a structure array. As a final note, since they are optional, we do **not** utilize `PH`, `PSchur_left`, or `PSchur_right` by default even when they are defined. In order to active these functions, we must modify the `PH_type`, `PSchur_left_type`, and `PSchur_right_type` elements in the state, respectively. We describe these variables in the chapter Optimization parameters. In our Rosenbrock example, we accumulate our functions with the following code:

**Language** C++

**Code**

```
// Create the bundle of functions
Optizelle::Unconstrained <double,Optizelle::Rm>::Functions::t fns;
fns.f.reset(new Rosenbrock);
fns.PH.reset(new RosenHInv(state.x));
```

**Language** Python

**Code**

```
# Create the bundle of functions
fns=Optizelle.Unconstrained.Functions.t()
fns.f=Rosenbrock()
fns.PH=RosenHInv()
```

**Language** MATLAB/Octave

**Code**
```
% Create the bundle of functions
fns=Optizelle.Unconstrained.Functions.t;
fns.f=Rosenbrock();
fns.PH=RosenHInv();
```

As another example, we accomplish the same task in our Simple equality constrained example with the code:

**Language** C++

**Code**
```
// Create a bundle of functions
Optizelle::EqualityConstrained <double,Rm,Rm>::Functions::t fns;
fns.f.reset(new MyObj);
fns.g.reset(new MyEq);
fns.PSchur_left.reset(new MyPrecon(state.x));
```

**Language** Python

**Code**
```
# Create a bundle of functions
fns=Optizelle.EqualityConstrained.Functions.t()
fns.f=MyObj()
fns.g=MyEq()
fns.PSchur_left=MyPrecon()
```

**Language** MATLAB/Octave

**Code**
```
% Create a bundle of functions
fns=Optizelle.EqualityConstrained.Functions.t;
fns.f=MyObj();
fns.g=MyEq();
fns.PSchur_left=MyPrecon();
```

## 3.9 Call the optimization solver

Once the state, parameters, and functions are set, calling the optimization solver is straightforward. Simply, we call one of the four commands:

**Language** C++

**Code**
```
Optizelle::Unconstrained<Real,XX>::Algorithms::getMin(
    msg,fns,state);

Optizelle::EqualityConstrained<Real,XX,YY>::Algorithms::getMin(
    msg,fns,state);

Optizelle::InequalityConstrained<Real,XX,ZZ>::Algorithms::getMin(
    msg,fns,state);

Optizelle::Constrained<Real,XX,YY,ZZ>::Algorithms::getMin(
    msg,fns,state);
```

**Language** Python

**Code**

```
Optizelle.Unconstrained.Algorithms.getMin(XX,msg,fns,state)

Optizelle.EqualityConstrained.Algorithms.getMin(XX,YY,msg,fns,state)

Optizelle.InequalityConstrained.Algorithms.getMin(XX,ZZ,msg,fns,state)

Optizelle.Constrained.Algorithms.getMin(XX,YY,ZZ,msg,fns,state)
```

**Language** MATLAB/Octave

**Code**

```
state = Optizelle.Unconstrained.Algorithms.getMin(XX,msg,fns,state);

state = Optizelle.EqualityConstrained.Algorithms.getMin(XX,YY,msg,fns,state);

state = Optizelle.InequalityConstrained.Algorithms.getMin(XX,ZZ,msg,fns,state);

state = Optizelle.Constrained.Algorithms.getMin(XX,YY,ZZ,msg,fns,state);
```

As was the case in the section Create the optimization state, `XX`, `YY`, and ZZ correspond to the vector spaces $X$, $Y$, and $Z$ described in the section Import or define the appropriate vector spaces. Likely, they are just `Optizelle::Rm` or `Optizelle.Rm`. Next, we call the function with a `Messaging` object, `msg`. In the simple case, we can simply use `Optizelle::Messaging::stdout` in C++, `Optizelle.Messaging.stdout` in Python, and `Optizelle.Messaging.stdout` in MATLAB/Octave. For more complicated cases, see the section User-defined messaging. Finally, we pass in the state and bundle of functions that we discussed in the sections Create the optimization state and Accumulate the functions, respectively.

In our Rosenbrock example, we call Optizelle's solver with the code:

**Language** C++

**Code**

```
// Solve the optimization problem
Optizelle::Unconstrained <double,Optizelle::Rm>::Algorithms
    ::getMin(Optizelle::Messaging::stdout,fns,state);
```

**Language** Python

**Code**

```
# Solve the optimization problem
Optizelle.Unconstrained.Algorithms.getMin(
    Optizelle.Rm,Optizelle.Messaging.stdout,fns,state)
```

**Language** MATLAB/Octave

**Code**

```
% Solve the optimization problem
state = Optizelle.Unconstrained.Algorithms.getMin( ...
    Optizelle.Rm,Optizelle.Messaging.stdout,fns,state);
```

With the Simple equality constrained example, this becomes:

**Language** C++

**Code**

```
// Solve the optimization problem
Optizelle::EqualityConstrained <double,Rm,Rm>::Algorithms::getMin(
    Optizelle::Messaging::stdout,fns,state);
```

**Language** Python

**Code**

```
# Solve the optimization problem
Optizelle.EqualityConstrained.Algorithms.getMin(
    Optizelle.Rm,Optizelle.Rm,Optizelle.Messaging.stdout,fns,state)
```

**Language** MATLAB/Octave

**Code**

```
% Solve the optimization problem
state = Optizelle.EqualityConstrained.Algorithms.getMin( ...
    Optizelle.Rm,Optizelle.Rm,Optizelle.Messaging.stdout,fns,state);
```

## 3.10 Extract the solution

After the optimization routine concludes, the solution resides inside of the optimization state in a variable called x and the reason we stopped the optimization resides in a variable called opt_stop. At this point, we can examine our solution and run any post optimization diagnostics we require. In our Rosenbrock example, we print out the final solution with the code:

**Language** C++

**Code**

```
// Print out the reason for convergence
std::cout << "The algorithm converged due to: " <<
    Optizelle::OptimizationStop::to_string(state.opt_stop) <<
    std::endl;

// Print out the final answer
std::cout << "The optimal point is: (" << state.x[0] << ','
    << state.x[1] << ')' << std::endl;
```

**Language** Python

**Code**

```
# Print out the reason for convergence
print "The algorithm converged due to: %s" % (
    Optizelle.OptimizationStop.to_string(state.opt_stop))

# Print out the final answer
print "The optimal point is: (%e,%e)" % (state.x[0],state.x[1])
```

**Language** MATLAB/Octave

**Code**

```
% Print out the reason for convergence
fprintf('The algorithm converged due to: %s\n', ...
    Optizelle.OptimizationStop.to_string(state.opt_stop));

% Print out the final answer
fprintf('The optimal point is: (%e,%e)\n',state.x(1),state.x(2));
```

In our Simple equality constrained example, this becomes:

**Language** C++

**Code**

```
// Print out the reason for convergence
std::cout << "The algorithm converged due to: " <<
    Optizelle::OptimizationStop::to_string(state.opt_stop) <<
    std::endl;

// Print out the final answer
std::cout << std::scientific << std::setprecision(16)
    << "The optimal point is: (" << state.x[0] << ','
<< state.x[1] << ')' << std::endl;
```

**Language** Python

**Code**

```
# Print out the reason for convergence
print "The algorithm converged due to: %s" % (
    Optizelle.OptimizationStop.to_string(state.opt_stop))

# Print out the final answer
print "The optimal point is: (%e,%e)" % (state.x[0],state.x[1])
```

**Language** MATLAB/Octave

**Code**

```
% Print out the reason for convergence
fprintf('The algorithm converged due to: %s\n', ...
    Optizelle.OptimizationStop.to_string(state.opt_stop));

% Print out the final answer
fprintf('The optimal point is: (%e,%e)\n',state.x(1),state.x(2));
```

## 3.11 Compile/run the program

As a final step, we need to either compile or run the program. Each language has its own nuances that we describe below.

### 3.11.1 C++

By default, we install the C++ relevant headers and libraries to

```
/some/path
├─ lib
│  ├─ liboptizelle.a
│  ├─ optizelle.lib
│  ├─ liboptizelle.so
│  ├─ liboptizelle.dylib
│  └─ optizelle.dll
└─ include
   └─ optizelle
      ├─ optizelle.h
      ├─ json.h
      └─ vspaces.h
```

where `/some/path` denotes the installation directory. Therefore, in order to compile an Optizelle program, we must add the directory

`/some/path/include`

to the list of include directories and

```
/some/path/lib
```

to the list of library directories. For the static library, we link either `liboptizelle.a` or `optizelle.lib`. For the dynamic library, we link either `liboptizelle.so`, `liboptizelle.dylib`, or `optizelle.dll`. Note, Optizelle depends on JsonCpp, BLAS, and LAPACK as well. Therefore, these headers and libraries must be included in any compilation command as well. For example, in GCC, we may have the following set of build flags

```
-I/usr/include -L/usr/lib -L/usr/share/optizelle/thirdparty/lib -loptizelle -ljson -lblas
-llapack
```

where we assume `CMAKE_INSTALL_PREFIX`=`/usr`.

### 3.11.2 Python/MATLAB/Octave

We require no compilation.

# 4 Optimization parameters

The parameters that guide the optimization solver have a dramatic effect its performance. To that end, we find each of these parameters within the optimization state that we initially discussed in the section Create the optimization state. These parameters are based on the canonical formulations

| **Unconstrained** | **Equality Constrained** |
|---|---|
| $\min_{x\in X} \quad f(x)$ | $\min_{x\in X} \quad f(x)$<br>$\text{st} \quad g(x) = 0$ |
| **Inequality Constrained** | **Constrained** |
| $\min_{x\in X} \quad f(x)$<br>$\text{st} \quad h(x) \succeq 0$ | $\min_{x\in X} \quad f(x)$<br>$\text{st} \quad g(x) = 0$<br>$\quad h(x) \succeq 0$ |

and come in one of nine types:

**Type** `Real`

**Description** Floating point numbers. In C++, this may be a type such as `double` or `float` as long as it matches the template parameters used in items such as `state` and `fns`. In Python and Matlab/Octave, we use the default floating point representation.

**Type** `Natural`

**Description** Nonnegative integer. In C++, we use the type `Optizelle::Natural`, which we set to be `size_t`. In Python, we use the default integer representation. In MATLAB/Octave, we use the default floating point representation.

**Type** `Enumerated`

**Description** Enumerated type. In C++, we use an `enum` type called `t` wrapped inside a namespace that we type explicitly to `Natural`. For example, we refer to the algorithm class as `AlgorithmClass` and define its type as `AlgorithmClass::t`. Then, we refer to the enumerated values as:

- `AlgorithmClass::TrustRegion`
- `AlgorithmClass::LineSearch`
- `AlgorithmClass::UserDefined`

In Python, we use integers, which we wrap inside of a class. For example, the class `AlgorithmClass` contains three integer values that we access with:

- AlgorithmClass.TrustRegion
- AlgorithmClass.LineSearch
- AlgorithmClass.UserDefined

In MATLAB/Octave, we use floating point numbers, which we wrap inside of a structured array. For example, the structure array `AlgorithmClass` contains three floating point values that we designate as:

- AlgorithmClass.TrustRegion
- AlgorithmClass.LineSearch
- AlgorithmClass.UserDefined

In all cases, we also provide a function called `to_string` in the class or namespace that converts the enumerated type to a string with the name of the enumerated element. Using our previous example of `AlgorithmClass`, in C++ we use

`AlgorithmClass::to_string`

whereas in Python and MATLAB/Octave we use

`AlgorithmClass.to_string`.

As to our specific enumerated types, we elaborate on them below.

**Type** `X_Vector`

**Description** User defined vector within the vector space $X$, the domain of our objective function, $f : X \rightarrow \mathbb{R}$.

**Type** `Y_Vector`

**Description** User defined vector within the vector space $Y$, the codomain of our equality constraint, $g : X \rightarrow Y$ with $g(x) = 0$.

**Type** `Z_Vector`

**Description** User defined vector within the vector space $Z$, the codomain of our inequality constraint, $h : X \rightarrow Z$ with $h(x) \succeq 0$.

**Type** `List`

**Description** List of a specified kind of vectors. In C++, this denotes a `std::list`. In Python, this becomes a `list`. Finally, in MATLAB/Octave, we use a cell array.

**Type** `Function`

**Description** Function of a specified kind of variable. This type represents a function inside the state structure that we use to set a number of similar parameters. However, in the JSON parameter files, we set it like it was just another parameter.

We further classify our enumerated types into the following:

**Type** `AlgorithmClass`

**Values**

```
TrustRegion,         // Trust-Region algorithms
LineSearch,          // Line-search algorithms
UserDefined          // User provides the iterate
```

**Type** `OptimizationStop`

**Values**

```
NotConverged,          // Algorithm did not converge
GradientSmall,         // Gradient was sufficiently small
StepSmall,             // Change in the step is small
MaxItersExceeded,      // Maximum number of iterations exceeded
InteriorPointInstability,// Instability in the interior point method
GlobalizationFailure,  // Too many failed globalization iterations
UserDefined            // Some user defined stopping condition
```

**Type** `Operators`

**Values**

```
Identity,        // Identity approximation
Zero,            // Zero approximation
ScaledIdentity,  // Scaled identity approximation
                 //
                 // || grad || / (2 delta) I
                 //
                 // Use this for trust-region steepest descent
                 // rather than Identity since it forces the
                 // iterate into the trust region
BFGS,            // BFGS approximation
InvBFGS,         // Inverse BFGS approximation
SR1,             // SR1 approximation
InvSR1,          // Inverse SR1 approximation
UserDefined      // User defined operator (such as the true
                 // Hessian for Newton's method)
```

**Type** `LineSearchDirection`

**Values**

```
// Note, all methods here, save BFGS, are preconditioned. This
// includes steepest descent, where dx = -PH grad. This is a good
// way to implement a user-defined search direction. For example,
// when we define PH to be the inverse of the Hessian, we get
// a globalized Newton method.

SteepestDescent,        // SteepestDescent
FletcherReeves,         // Fletcher-Reeves CG
PolakRibiere,           // Polak-Ribiere CG
HestenesStiefel,        // HestenesStiefel CG
BFGS,                   // Limited-memory BFGS
NewtonCG                // Newton-CG
```

**Type** `LineSearchKind`

**Values**

```
GoldenSection,   // Golden-section search
BackTracking,    // BackTracking search
TwoPointA,       // Barzilai and Borwein's method A
TwoPointB        // Barzilai and Borwein's method B
```

**Type** `OptimizationLocation`

**Values**

```
// Occurs at the start of the optimization function
BeginningOfOptimization,

// Occurs before the initial function and gradient evaluation
BeforeInitialFuncAndGrad,

// Occurs after the initial function and gradient evaluation
AfterInitialFuncAndGrad,

// Occurs just before the main optimization loop
BeforeOptimizationLoop,

// Occurs at the beginning of the optimization loop
BeginningOfOptimizationLoop,

// Occurs just before we take the optimization step x+dx
BeforeSaveOld,

// Occurs just before we take the optimization step x+dx
BeforeStep,

// Occurs before we calculate our new step.
BeforeGetStep,

// Occurs during a user defined get step calculation.
GetStep,

// Occurs after we take the optimization step x+dx, but before
// we calculate the gradient based on this new step. In addition,
// after this point we set the objective value, f_x, to be
// f_xpdx.
AfterStepBeforeGradient,

// Occurs just after the gradient computation with the new
// trial step
AfterGradient,

// Occurs before we update our quasi-Newton information.
BeforeQuasi,
```

```
// Occurs after we update our quasi-Newton information.
AfterQuasi,

// This occurs after we check our stopping condition. This is
// where the equality and inequality algorithms adjust the
// stopping conditions.
AfterCheckStop,

// This occurs last in the optimization loop. At this point,
// we have already incremented our optimization iteration and
// checked our stopping condition
EndOfOptimizationIteration,

// This occurs prior to the computation of the line search
BeforeLineSearch,

// This occurs after a rejected trust-region step
AfterRejectedTrustRegion,

// This occurs after a rejected line-search step
AfterRejectedLineSearch,

// This occurs prior to checking the predicted versus actual
// reduction in a trust-region method.
BeforeActualVersusPredicted,

// This occurs at the end of all optimization
EndOfOptimization
```

**Type** `ProblemClass`

**Values**

```
Unconstrained,         // Unconstrained optimization
EqualityConstrained,   // Equality constrained optimization
InequalityConstrained, // Inequality constrained optimization
Constrained            // Fully constrained optimization
```

**Type** `DiagnosticScheme`

**Values**

```
Never,            // Never compute our diagnostic checks
DiagnosticsOnly,  // No optimization. Only diagnostics.
EveryIteration    // Every iteration at the start of the iteration
```

**Type** `FunctionDiagnostics`

**Values**

```
NoDiagnostics,    // No diagnostic checks
FirstOrder,       // First-order function checks
SecondOrder       // Second-order function checks
```

**Type** `VectorSpaceDiagnostics`

**Values**

```
NoDiagnostics,    // No diagnostic checks
Basic,            // Test our basic vector space operations
EuclideanJordan   // Test our Euclidean-jordan algebraic
```

**Type** ToleranceKind

**Values**
```
Relative,          // Relative stopping tolerances
Absolute,          // Absolute stopping tolerances
```

**Type** QuasinormalStop

**Values**
```
Newton,                // Obtained the full Newton point
CauchyTrustRegion,     // Cauchy point truncated by the TR
CauchySafeguard,       // Cauchy point truncated by the safeguard
DoglegTrustRegion,     // Dogleg point truncated by the TR
DoglegSafeguard,       // Dogleg point truncated by the safeguard
NewtonTrustRegion,     // Newton point truncated by the TR
NewtonSafeguard,       // Newton point truncated by the safeguard
Feasible,              // Skipped due to feasibility
CauchySolved,          // Cauchy point solved g'(x)dx_cp+g(x)=0
LocalMin,              // Skipped due to a local min in the
                       // least-squares formulation, min 0.5 ||
                       // g'(x)dx + g(x) ||^2, or g'(x)*g(x)=0
NewtonFailed           // Augmented system solve for the Newton
                       // point failed, so we regressed to the
                       // Cauchy point
```

**Type** TruncatedStop

**Values**
```
NotConverged,               // Algorithm has not converged
NegativeCurvature,          // Negative curvature detected
RelativeErrorSmall,         // Relative error is small
MaxItersExceeded,           // Maximum number of iterations exceeded
TrustRegionViolated,        // Trust-region radius violated
NanOperator,                // NaN detected in the operator
NanPreconditioner,          // NaN detected in the preconditioner
NonProjectorPreconditioner,// Detected a nonprojecting
                            // preconditioner when one is required.
                            // Too much inexactness in the
                            // composite-step SQP method can trigger
                            // this.
NonSymmetricPreconditioner,// Detected a nonsymmetric preconditioner
NonSymmetricOperator,       // Detected a nonsymmetric operator
LossOfOrthogonality,        // Loss of orthogonality between the
                            // Krylov vectors detected
OffsetViolatesTrustRegion, // Offset is chosen such that
                            // || x_offset || > delta where
                            // delta is the trust-region radius
OffsetViolatesSafeguard, // Offset violates the safeguard
TooManyFailedSafeguard,  // Too many safeguarded steps have failed
ObjectiveIncrease           // CG objective, 0.5 <ABx,Bx> - <b,Bx>
                            // increased between iterations, which
                            // shouldn't happen.
```

**Type** Cone

| | | |
|---|---|---|
| **Values** | `Linear,` | `// Nonnegative orthant` |
| | `Quadratic,` | `// Second order cone` |
| | `Semidefinite` | `// Cone of positive semidefinite matrices` |

Based on these types, we catalog the precise meaning of our parameters below. As a note, the field **JSON Param** denotes whether or not we allow the parameter to be set in the JSON file described in the section Set the optimization parameters. Generally, these settable parameters correspond to parameters that tune the behavior the algorithms. The other parameters correspond to internal quantities that assist in diagnostics or advanced heuristics.

**Name** `eps_grad`

**Type** `Real`

**Valid Value** `state.eps_grad > Real(0.)`

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**JSON Param** Yes

**Default** `1e-8`

**Description** Tolerance for the gradient stopping criteria reported in `opt_stop`. We satisfy this stopping criteria when

| | |
|---|---|
| **Unconstrained** | $\|\nabla f(\mathtt{x}))\| \leq \mathtt{eps\_grad} \cdot \mathtt{norm\_gradtyp}$, |
| **Equality** | $\|\nabla f(\mathtt{x}) + g'(\mathtt{x})^*\mathtt{y}\| \leq \mathtt{eps\_grad} \cdot \mathtt{norm\_gradtyp}$, |
| **Inequality** | $\|\nabla f(\mathtt{x}) - h'(\mathtt{x})^*\mathtt{z}\| \leq \mathtt{eps\_grad} \cdot \mathtt{norm\_gradtyp}$, |
| **Constrained** | $\|\nabla f(\mathtt{x}) + g'(\mathtt{x})^*\mathtt{y} - h'(\mathtt{x})^*\mathtt{z}\| \leq \mathtt{eps\_grad} \cdot \mathtt{norm\_gradtyp}$. |

At each iteration, we output the norm on the left of the inequality under the label `||grad||`.

**Name** `eps_dx`

**Type** `Real`

**Valid Value** `state.eps_dx > Real(0.)`

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**JSON Param** Yes

**Default** `1e-16`

**Description** Tolerance for the step length stopping criteria reported in `opt_stop`. We satisfy this stopping criteria when

$$\|\mathtt{dx}\| < \mathtt{eps\_dx} \cdot \mathtt{norm\_dxtyp}.$$

At each iteration, we output the norm on the left of the inequality under the label `||dx||`.

**Name** `stored_history`

**Type** `Natural`

**Valid Value** `// Any`

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**JSON Param** Yes

**Default** `0`

**Description** Number of vectors stored for use with quasi-Newton methods such as SR1 and BFGS.

**Name** `iter`

**Type** `Natural`

**Valid Value** `state.iter > 0`

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**JSON Param** No

**Default** `1`

**Description** Current optimization iteration. We output `iter` at each iteration under the label `iter`.

**Name** `iter_max`

**Type** `Natural`

**Valid Value** `state.iter_max > 0`

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**JSON Param** Yes

**Default** `std::numeric_limits <Integer>::max()`

**Description** Maximum number of optimization iterations for the stopping criteria reported in `opt_stop`. We satisfy this stopping criteria when

$$\texttt{iter} \geq \texttt{iter\_max}$$

**Name** `glob_iter`

**Type** `Natural`

**Valid Value** `// Any`

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**JSON Param** No

**Default** `0`

**Description** Current globalization iteration. Here, globalization means the current iteration of the trust-region or line-search method and involves operations such as checking the actual versus predicted reduction or the sufficient decrease condition.

**Name** `glob_iter_max`

**Type** `Natural`

**Valid Value** `state.glob_iter_max > 0`

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**JSON Param** Yes

**Default** `10`

**Description** Maximum number of globalization iterations that we take before we exit the optimization. In other words, we only allow this many failed trust-region or line-search iterations before we exit the algorithm.

**Name** `glob_iter_total`

**Type** `Natural`

**Valid Value** `// Any`

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**JSON Param** No

**Default** `0`

**Description** Total number of globalization iterations taken across all iterations. This information is helpful when determining the overall expense of the algorithm. When we properly setup an equality constrained problem, we generally do one factorization of $g'(x)g'(x)^*$ every globalization iteration. In addition, evaluating the globalization routines for trust-region methods requires one Hessian-vector product every globalization iteration. We output `glob_iter_total` at each iteration under the label `glb_itr_tot`.

**Name** `opt_stop`

**Type** `OptimizationStop`

**Valid Value** `// Any`

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**JSON Param** No

**Default** `OptimizationStop::NotConverged`

**Description** Why we've stopping the optimization. We use the following logic when determining when to stop

1. If the optimization iteration exceeds the maximum number of iterations, stop. We control this with the parameter `iter_max`.
2. If size of the optimization step becomes too small, stop. We control this with the parameter `eps_dx`.
3. If we have inequality constraints and the estimated interior point parameter `mu_est` becomes negative, stop.
4. If the size of the gradient becomes too small and we satisfy the following additional conditions, stop. We control this with the parameter `eps_grad`.
   (a) For problems with equality constraints, we require that the norm of the equality constraint be small. We control this with the parameter `eps_constr`.

(b) For problems with inequality constraints, we require that estimated interior point parameter be small. We control this with the parameter eps_mu.

| | |
|---|---|
| **Name** | trunc_iter |
| **Type** | Natural |
| **Valid Value** | // Any |
| **Problem Class** | Unconstrained, Equality Constrained, Inequality Constrained, Constrained |
| **JSON Param** | No |
| **Default** | 0 |
| **Description** | Current number of iterations taken by truncated CG when solving the optimality conditions. We output trunc_iter at each iteration under the label trunc_iter. |

| | |
|---|---|
| **Name** | trunc_iter_max |
| **Type** | Natural |
| **Valid Value** | state.trunc_iter_max > 0 |
| **Problem Class** | Unconstrained, Equality Constrained, Inequality Constrained, Constrained |
| **JSON Param** | Yes |
| **Default** | 10 |
| **Description** | Maximum number of iterations taken by truncated CG when solving the optimality conditions. |

| | |
|---|---|
| **Name** | trunc_iter_total |
| **Type** | Natural |
| **Valid Value** | // Any |
| **Problem Class** | Unconstrained, Equality Constrained, Inequality Constrained, Constrained |
| **JSON Param** | No |
| **Default** | 0 |
| **Description** | Total number of iterations ever taken by the truncated CG when solving the optimality conditions. This gives information about the total amount computational effort taken by Optizelle as we evaluate one Hessian-vector product each iteration. We output trunc_iter_total at each iteration under the label trc_itr_tot. |

| | |
|---|---|
| **Name** | trunc_orthog_storage_max |
| **Type** | Natural |
| **Valid Value** | state.trunc_orthog_storage_max > 0 |

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**JSON Param** Yes

**Default** `1`

**Description** Number of vectors stored and used in the orthogonalization of truncated CG. In theory, we only need 1 for unconstrained and inequality constrained problems, but this leads to numerical instabilities. In practice, if memory is available, it may be worthwhile to over orthogonalize.

**Name** `trunc_orthog_iter_max`

**Type** `Natural`

**Valid Value** `state.trunc_orthog_iter_max > 0`

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**JSON Param** Yes

**Default** `1`

**Description** Maximum number of orthogonalization iterations that we use in truncated-CG. In theory, 1 should be enough, which means that we orgthogonalize against all the stored previous directions once. In practice, we'll eventually lose orthogonality, so using 2 may help at the cost of additional computation.

**Name** `trunc_stop`

**Type** `TruncatedStop`

**Valid Value** `// Any`

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**JSON Param** No

**Default** `TruncatedStop::RelativeErrorSmall`

**Description** Reason why truncated CG exited when solving the optimality system. We output `trunc_stop` at each iteration under the label `trunc_stop`.

**Name** `trunc_err`

**Type** `Real`

**Valid Value** `// Any`

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**JSON Param** No

**Default** `std::numeric_limits<Real>::quiet_NaN()`

**Description** Relative error in the solution returned by the truncated CG when solving the optimality system. We output `trunc_err` at each iteration under the label `trunc_err`.

| | |
|---|---|
| **Name** | `eps_trunc` |
| **Type** | `Real` |
| **Valid Value** | `state.eps_trunc > Real(0.)` |
| **Problem Class** | Unconstrained, Equality Constrained, Inequality Constrained, Constrained |
| **JSON Param** | Yes |
| **Default** | `1e-2` |
| **Description** | Relative stopping criteria for truncated CG. In truncated CG, when solving the system $Ax = b$ with preconditioner $B$, we use the stopping criteria $\|B(Ax_k - b)\| \leq$ `eps_trunc`$\|B(Ax_0 - b)\|$. |

| | |
|---|---|
| **Name** | `algorithm_class` |
| **Type** | `AlgorithmClass` |
| **Valid Value** | `// Any` |
| **Problem Class** | Unconstrained, Equality Constrained, Inequality Constrained, Constrained |
| **JSON Param** | Yes |
| **Default** | `AlgorithmClass::TrustRegion` |
| **Description** | Class of algorithm used in optimization. |

| | |
|---|---|
| **Name** | `PH_type` |
| **Type** | `Operators` |
| **Valid Value** | `// Any` |
| **Problem Class** | Unconstrained, Equality Constrained, Inequality Constrained, Constrained |
| **JSON Param** | Yes |
| **Default** | `Operators::Identity` |
| **Description** | Preconditioner used when solving the optimality conditions. Note, in order to accommodate the null space projection, we currently ignore this quantity if problems with equality constraints. |

| | |
|---|---|
| **Name** | `H_type` |
| **Type** | `Operators` |
| **Valid Value** | `// Any` |
| **Problem Class** | Unconstrained, Equality Constrained, Inequality Constrained, Constrained |
| **JSON Param** | Yes |

| | |
|---|---|
| **Default** | `Operators::UserDefined` |
| **Description** | Hessian approximation for the objective function. |

| | |
|---|---|
| **Name** | `norm_gradtyp` |
| **Type** | `Real` |
| **Valid Value** | `state.norm_gradtyp >= Real(0.)`<br>`\|\| (state.iter==1 && state.norm_gradtyp!=state.norm_gradtyp)` |
| **Problem Class** | Unconstrained, Equality Constrained, Inequality Constrained, Constrained |
| **JSON Param** | No |
| **Default** | `std::numeric_limits<Real>::quiet_NaN()` |

**Description**

Norm of a typical gradient defined as

| | |
|---|---|
| **Unconstrained** | $\|\nabla f(x_0)\|$, |
| **Equality** | $\|\nabla f(x_0) + g'(x_0)^* y_0\|$, |
| **Inequality** | $\|\nabla f(x_0) - h'(x_0)^* z_0\|$, |
| **Constrained** | $\|\nabla f(x_0) + g'(x_0)^* y_0 - h'(x_0)^* z_0\|$, |

where $x_0$, $y_0$, and $z_0$ denote our variables at the first iteration. Sometimes, we use `norm_gradtyp` with the stopping criteria described in `eps_grad`. Specifically, we only refer to this quantity when `eps_kind` is set to Relative. When `eps_kind` is set to Absolute, we ignore this value and instead use 1.0.

| | |
|---|---|
| **Name** | `norm_dxtyp` |
| **Type** | `Real` |
| **Valid Value** | `state.norm_dxtyp >= Real(0.)`<br>`\|\| (state.iter==1 && state.norm_dxtyp!=state.norm_dxtyp)` |
| **Problem Class** | Unconstrained, Equality Constrained, Inequality Constrained, Constrained |
| **JSON Param** | No |
| **Default** | `std::numeric_limits<Real>::quiet_NaN()` |
| **Description** | Norm of a typical optimization step. Similar to `norm_gradtyp`, we set this to be the gradient found at the initial guess and use it in the stopping criteria described in `eps_dx`. Since an optimization algorithm may have numerical issues on the first optimization iteration, we do not use the first optimization step generated. By using the norm of the gradient, we approximate the norm of a step taken by the steepest descent algorithm. As a note, we only refer to this quantity when `eps_kind` is set to Relative. When `eps_kind` is set to Absolute, we ignore this value and instead use 1.0. |

| | |
|---|---|
| **Name** | `x` |
| **Type** | `X_Vector` |
| **Valid Value** | `// Any` |

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**JSON Param** No

**Default** `X::copy(x_user,x);`

**Description** Optimization variable.

**Name** `grad`

**Type** `X_Vector`

**Valid Value** `// Any`

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**JSON Param** No

**Default** `X::init(x_user)`

**Description** Gradient of the objective, $\nabla f(x)$.

**Name** `dx`

**Type** `X_Vector`

**Valid Value** `// Any`

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**JSON Param** No

**Default** `X::init(x_user)`

**Description** Step taken during the optimization iteration. Every iteration we set `x=x+dx`. In addition, we output the norm of this vector at each iteration under the label `||dx||`.

**Name** `x_old`

**Type** `X_Vector`

**Valid Value** `// Any`

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**JSON Param** No

**Default** `X::init(x_user)`

**Description** Optimization variable from the last iteration.

**Name** `grad_old`

**Type** `X_Vector`

**Valid Value** `// Any`

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**JSON Param** No

**Default** `X::init(x_user)`

**Description** Gradient of the objective from the last iteration.

**Name** `dx_old`

**Type** `X_Vector`

**Valid Value** `// Any`

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**JSON Param** No

**Default** `X::init(x_user)`

**Description** Optimization step from the last iteration.

**Name** `oldY`

**Type** `List(X_Vector)`

**Valid Value** `// Any`

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**JSON Param** No

**Default** `// Empty`

**Description** Difference in prior gradients,

$$[\nabla f(\texttt{x}_{\texttt{iter}}) - \nabla f(\texttt{x}_{\texttt{iter}-1}), \ldots, \nabla f(\texttt{x}_{\texttt{iter}-\texttt{stored\_history}}) - \nabla f(\texttt{x}_{\texttt{iter}-\texttt{stored\_history}-1})].$$

We use this list in our quasi-Newton methods.

**Name** `oldS`

**Type** `List(X_Vector)`

**Valid Value** `// Any`

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**JSON Param** No

**Default** `// Empty`

**Description** Difference in prior optimization steps,

$$[x_{\texttt{iter}} - x_{\texttt{iter}-1}, \ldots, x_{\texttt{iter}-\texttt{stored\_history}} - x_{\texttt{iter}-\texttt{stored\_history}-1}].$$

We use this list in our quasi-Newton methods.

**Name** `f_x`

**Type** `Real`

**Valid Value** `state.f_x == state.f_x || state.iter==1`

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**JSON Param** No

**Default** `std::numeric_limits<Real>::quiet_NaN()`

**Description** Current value of the objective function, $f(\mathtt{x})$. We output `f_x` at each iteration under the label `f(x)`.

**Name** `f_xpdx`

**Type** `Real`

**Valid Value** `state.f_xpdx == state.f_xpdx || state.iter==1`

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**JSON Param** No

**Default** `std::numeric_limits<Real>::quiet_NaN()`

**Description** Value of the objective function at the trial step, $f(\mathtt{x} + \mathtt{dx})$.

**Name** `msg_level`

**Type** `Natural`

**Valid Value** `// Any`

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**JSON Param** Yes

**Default** `1`

**Description** Messaging level. To turn messages off, use 0. For normal messaging, set to 1. For more detailed information, set to 2. For linear solver information, set to 3. To see precise information about what information we display, refer to the chapter entitled Output.

**Name** `safeguard_failed_max`

**Type** `Natural`

**Valid Value** `state.safeguard_failed_max >=1`

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**JSON Param** Yes

**Default** `5`

**Description** Number of failed safe-guard steps before exiting truncated CG. We use this exclusively for our inequality constraints.

**Name** `safeguard_failed`

**Type** `Natural`

**Valid Value** `// Any`

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**JSON Param** No

**Default** `0`

**Description** Number of failed safe-guard steps during the last iteration. We output `safeguard_failed` at each iteration under the label `safe_fail`.

**Name** `safeguard_failed_total`

**Type** `Natural`

**Valid Value** `// Any`

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**JSON Param** No

**Default** `0`

**Description** Total number of failed safe-guard steps.

**Name** `alpha_x`

**Type** `Real`

**Valid Value** `// Any`

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**JSON Param** No

**Default** `std::numeric_limits <Real>::quiet_NaN()`

**Description** How much we truncate `dx` in an interior point method in order to maintain strict feasibility. When 1.0, we do not truncate and take a full step. We output `alpha_x` at each iteration under the label `alpha_x`.

**Name** `alpha_x_qn`

**Type** `Real`

**Valid Value** `// Any`

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**JSON Param** No

| | |
|---|---|
| **Default** | `std::numeric_limits <Real>::quiet_NaN()` |
| **Description** | How much we truncate `dx_n` in an interior point method in order to maintain strict feasibility. When 1.0, we do not truncate and take a full step.<br>We output `alpha_x_qn` at each iteration under the label `alpha_x_qn`. |
| **Name** | `eps_kind` |
| **Type** | `ToleranceKind` |
| **Valid Value** | `// Any` |
| **Problem Class** | Unconstrained, Equality Constrained, Inequality Constrained, Constrained |
| **JSON Param** | Yes |
| **Default** | `ToleranceKind::Absolute` |
| **Description** | Kind of stopping tolerance used by the algorithms. |

| | |
|---|---|
| **Name** | `delta` |
| **Type** | `Real` |
| **Valid Value** | `state.delta >= Real(0.)` |
| **Problem Class** | Unconstrained, Equality Constrained, Inequality Constrained, Constrained |
| **JSON Param** | Yes |
| **Default** | `1.` |
| **Description** | Trust region radius. We use this as a starting value. Later, we adjust the radius depending on the behavior of the algorithms. As a note, we output `delta` at each iteration under the label `delta`. |

| | |
|---|---|
| **Name** | `eta1` |
| **Type** | `Real` |
| **Valid Value** | `state.eta1 > Real(0.) && state.eta1 < Real(1.)` |
| **Problem Class** | Unconstrained, Equality Constrained, Inequality Constrained, Constrained |
| **JSON Param** | Yes |
| **Default** | `.1` |
| **Description** | When the actual versus predicted reduction for a trust-region method is below this threshold, we reject the step. Otherwise, we accept it. |

| | |
|---|---|
| **Name** | `eta2` |
| **Type** | `Real` |
| **Valid Value** | `state.eta2 > state.eta1 && state.eta2 < Real(1.)` |

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**JSON Param** Yes

**Default** `.9`

**Description** When the actual versus predicted reduction for a trust-region method is above this threshold and the size of the trial step equals the trust-region radius, we increase the size of the trust-region radius.

**Name** `ared`

**Type** `Real`

**Valid Value** `// Any`

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**JSON Param** No

**Default** `std::numeric_limits<Real>::quiet_NaN()`

**Description** Actual reduction in the merit function between the current iterate and the iterate after taking the trial step,

$$\texttt{ared} \equiv \texttt{merit}(\texttt{x}) - \texttt{merit}(\texttt{x} + \texttt{dx}).$$

We use the following merit functions, $\texttt{merit} : \texttt{X\_Vector} \rightarrow \texttt{Real}$,

**Unconstrained** $f(\texttt{x})$,

**Equality** $f(\texttt{x}) + \langle \texttt{y}, g(\texttt{x})\rangle + \texttt{rho}||g(\texttt{x})||^2$,

**Inequality** $f(\texttt{x}) - \texttt{mu} \cdot \texttt{barr}(h(\texttt{x}))$,

**Constrained** $f(\texttt{x}) + \langle \texttt{y}, g(\texttt{x})\rangle + \texttt{rho}||g(\texttt{x})||^2 - \texttt{mu} \cdot \texttt{barr}(h(\texttt{x}))$.

Here, `barr` refers to the barrier function, which we describe in the section Customized vector spaces. As a note, we output the value of the merit function at each iteration under the label `merit(x)` and `ared` under the label `ared`.

**Name** `pred`

**Type** `Real`

**Valid Value** `// Any`

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**JSON Param** No

**Default** `std::numeric_limits<Real>::quiet_NaN()`

**Description** Predicted reduction in the merit function between the current iterate and the iterate after taking the trial step,

$$\texttt{pred} \equiv \texttt{model}(0) - \texttt{model}(\texttt{dx}).$$

We use the following model functions, $\texttt{model} : \texttt{X\_Vector} \rightarrow \texttt{Real}$,

**Unconstrained**

$$f(\texttt{x}) + \langle \nabla f(\texttt{x}), \texttt{dx}\rangle + \frac{1}{2}\langle H(\texttt{x})\texttt{dx}, \texttt{dx}\rangle,$$

**Equality**

$$f(\mathtt{x}) + \langle \mathtt{y}, g(\mathtt{x})\rangle + \mathtt{rho}||g(\mathtt{x})||^2$$
$$+\langle \nabla f(\mathtt{x}) + g'(\mathtt{x})^*\mathtt{y}, \mathtt{dx}\rangle$$
$$+\tfrac{1}{2}\langle H(\mathtt{x})\mathtt{dx} + (g''(\mathtt{x})\mathtt{dx})^*\mathtt{y}, \mathtt{dx}\rangle,$$

**Inequality**

$$f(\mathtt{x}) - \mathtt{mu}\cdot\mathtt{barr}(h(\mathtt{x}))$$
$$+\langle \nabla f(\mathtt{x}) - \mathtt{mu}\cdot h'(\mathtt{x})^* L(h(\mathtt{x}))^{-1}e, \mathtt{dx}\rangle$$
$$+\tfrac{1}{2}\langle H(\mathtt{x})\mathtt{dx} + h'(\mathtt{x})^* L(h(\mathtt{x}))^{-1}(h'(\mathtt{x})\mathtt{dx}\circ\mathtt{z}), \mathtt{dx}\rangle,$$

**Constrained**

$$f(\mathtt{x}) + \langle \mathtt{y}, g(\mathtt{x})\rangle + \mathtt{rho}||g(\mathtt{x})||^2 - \mathtt{mu}\cdot\mathtt{barr}(h(\mathtt{x}))$$
$$+\langle \nabla f(\mathtt{x}) + g'(\mathtt{x})^*\mathtt{y} - \mathtt{mu}\cdot h'(\mathtt{x})^* L(h(\mathtt{x}))^{-1}e, \mathtt{dx}\rangle$$
$$+\tfrac{1}{2}\langle H(\mathtt{x})\mathtt{dx} + (g''(\mathtt{x})\mathtt{dx})^*\mathtt{y} + h'(\mathtt{x})^* L(h(\mathtt{x}))^{-1}(h'(\mathtt{x})\mathtt{dx}\circ\mathtt{z}), \mathtt{dx}\rangle.$$

Here, $\circ$ denotes the Jordan product, `prod`; $L(h(x))^{-1}$ denotes the inverse of the linear operator induced by the Jordan product, `linv`; $e$ denotes the identity element in the pseudo-Euclidean-Jordan algebra, `id`; and `barr` denotes the barrier function. We describe each of these operations further in the section Customized vector spaces. As a note, we output `pred` at each iteration under the label `pred`.

**Name** `alpha0`

**Type** `Real`

**Valid Value** `state.alpha0 >= Real(0.)`

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**JSON Param** Yes

**Default** `1.`

**Description** Base line-search step length. Generally, our line-search methods search for a scaling `alpha` $\in [0, 2\cdot$ `alpha0`$]$. Once we find `alpha`, we increase `alpha0` when our search always brackets to the right and decrease it when our search always brackets to the left. As a note, we output `alpha0` at each iteration under the label `alpha0`.

**Name** `alpha`

**Type** `Real`

**Valid Value** `// Any`

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**JSON Param** No

**Default** `std::numeric_limits <Real>::quiet_NaN()`

**Description** Actual line-search step length. After our line-search process completes, we modify our step `dx` $\leftarrow$ `alpha` $\cdot$ `dx`. As a note, we output `alpha` at each iteration under the label `alpha`.

**Name** `c1`

**Type** `Real`

**Valid Value** `state.c1 > Real(0.) && state.c1 < Real(1.)`

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**JSON Param** Yes

**Default** `1e-4`

**Description** Sufficient decrease parameter. When we set `algorithm_class` to LineSearch, we only take a step when $\texttt{merit}(\texttt{x}+\texttt{alpha}\cdot\texttt{dx}) < \texttt{merit}(\texttt{x})+\texttt{c1}\cdot\texttt{alpha}\langle\tilde{x},\texttt{dx}\rangle$ where we define $\tilde{x}$ as

| | |
|---|---|
| **Unconstrained** | $\nabla f(\texttt{x})$, |
| **Equality** | $\nabla f(\texttt{x}) + g'(\texttt{x})^*\texttt{y}$, |
| **Inequality** | $\nabla f(\texttt{x}) - \texttt{mu}\cdot h'(\texttt{x})^* L(h(\texttt{x}))^{-1}e$, |
| **Constrained** | $\nabla f(\texttt{x}) + g'(\texttt{x})^*\texttt{y} - \texttt{mu}\cdot h'(\texttt{x})^* L(h(\texttt{x}))^{-1}e$. |

Here, $L(h(x))^{-1}$ denotes the inverse of the linear operator induced by the Jordan product, `linv`; and $e$ denotes the identity element in the pseudo-Euclidean-Jordan algebra, `id`. We describe each of these operations further in the section Customized vector spaces.

**Name** `ls_iter`

**Type** `Natural`

**Valid Value** `// Any`

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**JSON Param** No

**Default** `0`

**Description** Current number of iterations used in the line search. We use this to determine the amount of computational effort used by Optizelle during the last iteration. As a note, we output `ls_iter` at each iteration under the label `ls_iter`.

**Name** `ls_iter_max`

**Type** `Natural`

**Valid Value** `state.ls_iter_max > 0`

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**JSON Param** Yes

**Default** `5`

**Description** Maximum number of iterations used in the line search before checking the sufficient decrease condition. We use this to tune the amount of work done by the line search.

**Name** `ls_iter_total`

**Type** `Natural`

**Valid Value** `// Any`

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**JSON Param** No

**Default** `0`

**Description** Total number of iterations ever taken by the line search. We use this to determine the amount of computational effort used by Optizelle.

**Name** `eps_ls`

**Type** `Real`

**Valid Value** `state.eps_ls > Real(0.)`

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**JSON Param** Yes

**Default** `1e-2`

**Description** Relative stopping tolerance used by the line search. At the moment, we do not use this parameter.

**Name** `dir`

**Type** `LineSearchDirection`

**Valid Value** `// Any`

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**JSON Param** Yes

**Default** `LineSearchDirection::SteepestDescent`

**Description** Line-search direction taken by the line-search algorithm.

**Name** `kind`

**Type** `LineSearchKind`

**Valid Value**

```
(state.kind!=LineSearchKind::GoldenSection
    || state.ls_iter_max >= 2) &&
(state.kind!=LineSearchKind::TwoPointA ||
    state.kind!=LineSearchKind::TwoPointB ||
    state.dir==LineSearchDirection::SteepestDescent)
```

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**JSON Param** Yes

**Default** `LineSearchKind::GoldenSection`

**Description** Kind of line-search used in the line-search algorithm.

| | |
|---|---|
| **Name** | `f_diag` |
| **Type** | `FunctionDiagnostics` |
| **Valid Value** | `// Any` |
| **Problem Class** | Unconstrained, Equality Constrained, Inequality Constrained, Constrained |
| **JSON Param** | Yes |
| **Default** | `FunctionDiagnostics::NoDiagnostics` |
| **Description** | Function diagnostics on $f$. |

| | |
|---|---|
| **Name** | `L_diag` |
| **Type** | `FunctionDiagnostics` |
| **Valid Value** | `// Any` |
| **Problem Class** | Unconstrained, Equality Constrained, Inequality Constrained, Constrained |
| **JSON Param** | Yes |
| **Default** | `FunctionDiagnostics::NoDiagnostics` |
| **Description** | Function diagnostics on the Lagrangian. |

| | |
|---|---|
| **Name** | `x_diag` |
| **Type** | `VectorSpaceDiagnostics` |
| **Valid Value** | `// Any` |
| **Problem Class** | Unconstrained, Equality Constrained, Inequality Constrained, Constrained |
| **JSON Param** | Yes |
| **Default** | `VectorSpaceDiagnostics::NoDiagnostics` |
| **Description** | Vector space diagnostics on X. |

| | |
|---|---|
| **Name** | `dscheme` |
| **Type** | `DiagnosticScheme` |
| **Valid Value** | `// Any` |
| **Problem Class** | Unconstrained, Equality Constrained, Inequality Constrained, Constrained |
| **JSON Param** | Yes |
| **Default** | `DiagnosticScheme::Never` |
| **Description** | Which diagnostic scheme, if any, to employ. |

| | |
|---|---|
| **Name** | `y` |
| **Type** | `Y_Vector` |
| **Valid Value** | `// Any` |
| **Problem Class** | Equality Constrained, Constrained |
| **JSON Param** | No |
| **Default** | `// Equality constrained`<br>`// argmin_y \|\| grad f(x) + g'(x)*y \|\|`<br>`//`<br>`// Constrained`<br>`// argmin_y \|\| grad f(x) + g'(x)*y - h'(x)*z \|\|` |
| **Description** | Equality multiplier (dual variable or Lagrange multiplier.) |

| | |
|---|---|
| **Name** | `dy` |
| **Type** | `Y_Vector` |
| **Valid Value** | `// Any` |
| **Problem Class** | Equality Constrained, Constrained |
| **JSON Param** | No |
| **Default** | `Y::init(y_user)` |
| **Description** | Step in the equality multiplier. Every iteration we set `y`=`y`+`dy`. |

| | |
|---|---|
| **Name** | `zeta` |
| **Type** | `Real` |
| **Valid Value** | `state.zeta > Real(0.) && state.zeta < Real(1.)` |
| **Problem Class** | Equality Constrained, Constrained |
| **JSON Param** | Yes |
| **Default** | `0.8` |
| **Description** | Fraction of the total trust region used in the quasi-normal step. |

| | |
|---|---|
| **Name** | `eta0` |
| **Type** | `Real` |
| **Valid Value** | `state.eta0 > Real(0.) && state.eta0 < Real(1.)-state.eta1` |
| **Problem Class** | Equality Constrained, Constrained |
| **JSON Param** | Yes |
| **Default** | `0.5` |

| | |
|---|---|
| **Description** | Trust-region parameter that bounds the error in the predicted-reduction. |

| | |
|---|---|
| **Name** | `rho` |
| **Type** | `Real` |
| **Valid Value** | `state.rho >= Real(1.)` |
| **Problem Class** | Equality Constrained, Constrained |
| **JSON Param** | Yes |
| **Default** | `1.0` |
| **Description** | Penalty parameter for the augmented-Lagrangian. In problems with equality constraints, this term appears in the merit functions |

**Equality** $f(\texttt{x}) + \langle \texttt{y}, g(\texttt{x}) \rangle + \texttt{rho}||g(\texttt{x})||^2$,

**Constrained** $f(\texttt{x}) + \langle \texttt{y}, g(\texttt{x}) \rangle + \texttt{rho}||g(\texttt{x})||^2 - \texttt{mu} \cdot \texttt{barr}(h(\texttt{x}))$.

Here, `barr` refers to the barrier function, which we describe in the section Customized vector spaces.

| | |
|---|---|
| **Name** | `rho_old` |
| **Type** | `Real` |
| **Valid Value** | `state.rho_old >= Real(1.)` |
| **Problem Class** | Equality Constrained, Constrained |
| **JSON Param** | No |
| **Default** | `rho` |
| **Description** | Penalty parameter from the last iteration. |

| | |
|---|---|
| **Name** | `rho_bar` |
| **Type** | `Real` |
| **Valid Value** | `state.rho_bar > Real(0.)` |
| **Problem Class** | Equality Constrained, Constrained |
| **JSON Param** | Yes |
| **Default** | `1e-8` |
| **Description** | Fixed increase in the penalty parameter in the augmented Lagrangian merit function. |

| | |
|---|---|
| **Name** | `eps_constr` |
| **Type** | `Real` |
| **Valid Value** | `state.eps_constr > Real(0.)` |

**Problem Class** Equality Constrained, Constrained

**JSON Param** Yes

**Default** 1e-8

**Description** Relative stopping tolerance for feasibility with respect to the equality constraint reported in opt_stop. We satisfy this stopping criteria when

$$\|g(\mathtt{x})\| < \mathtt{eps\_constr} \cdot \mathtt{norm\_gxtyp}.$$

At each iteration, we output the norm on the left of the inequality under the label `||g(x)||`. Note, since this value tunes a *relative* stopping criteria, if we start with a feasible solution, we need to adjust this value to be something like `1.0`. This states that we do not seek relative improvement in the infeasibility.

**Name** `xi_qn`

**Type** `Real`

**Valid Value** `state.xi_qn > Real(0.) && state.xi_qn < Real(1.)`

**Problem Class** Equality Constrained, Constrained

**JSON Param** Yes

**Default** 1e-4

**Description** Relative stopping tolerance for the augmented system solve associated with the quasi-Newton step.

**Name** `xi_pg`

**Type** `Real`

**Valid Value** `state.xi_pg > Real(0.) && state.xi_pg < Real(1.)`

**Problem Class** Equality Constrained, Constrained

**JSON Param** Yes

**Default** 1e-4

**Description** Relative stopping tolerance for the augmented system solve associated with the projection of the gradient prior to solving the tangential subproblem.

**Name** `xi_proj`

**Type** `Real`

**Valid Value** `state.xi_proj > Real(0.) && state.xi_proj < Real(1.)`

**Problem Class** Equality Constrained, Constrained

**JSON Param** Yes

**Default** 1e-4

**Description** Relative stopping tolerance for the augmented system solve associated with the null-space projection of the iterate in the tangential subproblem.

**Name** `xi_tang`

**Type** `Real`

**Valid Value** `state.xi_tang > Real(0.) && state.xi_tang < Real(1.)`

**Problem Class** Equality Constrained, Constrained

**JSON Param** Yes

**Default** `1e-4`

**Description** Relative stopping tolerance for the augmented system solve associated with the tangential step computation after solving the tangential subproblem.

**Name** `xi_lmh`

**Type** `Real`

**Valid Value** `state.xi_lmh > Real(0.) && state.xi_lmh < Real(1.)`

**Problem Class** Equality Constrained, Constrained

**JSON Param** Yes

**Default** `1e-4`

**Description** Relative stopping tolerance for the augmented system solve associated with the equality multiplier computation.

**Name** `xi_all`

**Type** `Function(Real)`

**Valid Value** `// state.xi_all > Real(0.) && state.xi_all < Real(1.)`

**Problem Class** Equality Constrained, Constrained

**JSON Param** Yes

**Default** `// None`

**Description** Relative stopping tolerance for all of the augmented system solves, `xi_qn`, `xi_pg`, `xi_proj`, `xi_proj`, `xi_tang`, and `xi_lmh`.

**Name** `xi_lmg`

**Type** `Real`

**Valid Value** `state.xi_lmg > Real(0.)`

**Problem Class** Equality Constrained, Constrained

**JSON Param** Yes

**Default** `1e4`

**Description** Absolute tolerance on the residual of the equality multiplier solve.

**Name** `xi_4`

**Type** `Real`

**Valid Value** `state.xi_4 > Real(1.)`

**Problem Class** Equality Constrained, Constrained

**JSON Param** Yes

**Default** `2.`

**Description** Tolerance for how much error is acceptable after computing the tangential step given the result from the tangential subproblem.

**Name** `rpred`

**Type** `Real`

**Valid Value** `// Any`

**Problem Class** Equality Constrained, Constrained

**JSON Param** No

**Default** `std::numeric_limits<Real>::quiet_NaN()`

**Description** Residual term in the predicted reduction. We use this quantity to determine if we computed a tangential step that is accurate enough.

**Name** `PSchur_left_type`

**Type** `Operators`

**Valid Value**

```
state.PSchur_left_type == Operators::Identity ||
state.PSchur_left_type == Operators::UserDefined
```

**Problem Class** Equality Constrained, Constrained

**JSON Param** Yes

**Default** `Operators::Identity`

**Description** Left preconditioner for the augmented system. For a full discussion of this preconditioner, see the section (Optional) Define the preconditioners.

**Name** `PSchur_right_type`

**Type** `Operators`

**Valid Value**

```
state.PSchur_right_type == Operators::Identity ||
state.PSchur_right_type == Operators::UserDefined
```

**Problem Class** Equality Constrained, Constrained

**JSON Param** Yes

**Default** `Operators::Identity`

**Description** Right preconditioner for the augmented system. For a full discussion of this preconditioner, see the section (Optional) Define the preconditioners.

**Name** `augsys_iter_max`

**Type** `Natural`

**Valid Value** `state.augsys_iter_max > 0`

**Problem Class** Equality Constrained, Constrained

**JSON Param** Yes

**Default** `100`

**Description** Maximum number of GMRES iterations allowed when solving an augmented system.

**Name** `augsys_rst_freq`

**Type** `Natural`

**Valid Value** `// Any`

**Problem Class** Equality Constrained, Constrained

**JSON Param** Yes

**Default** `0`

**Description** How often we restart the augmented system solve. We restart GMRES every specified number of iterations in order to save memory. When 0, we do not restart.

**Name** `augsys_qn_iter`

**Type** `Natural`

**Valid Value** `// Any`

**Problem Class** Equality Constrained, Constrained

**JSON Param** No

**Default** `0`

**Description** Number of iterations taken during the last iterate by the augmented system solve for the quasi-normal step.

**Name** `augsys_pg_iter`

**Type** `Natural`

| | |
|---|---|
| **Valid Value** | `// Any` |
| **Problem Class** | Equality Constrained, Constrained |
| **JSON Param** | No |
| **Default** | `0` |
| **Description** | Number of iterations taken during the last iterate by the augmented system solve when projecting the gradient prior to the tangential subproblem. |

| | |
|---|---|
| **Name** | `augsys_proj_iter` |
| **Type** | `Natural` |
| **Valid Value** | `// Any` |
| **Problem Class** | Equality Constrained, Constrained |
| **JSON Param** | No |
| **Default** | `0` |
| **Description** | Number of iterations taken during the last iterate by the augmented system solve during the nullspace projection in the tangential subproblem. Since there are likely many projections, this is the total number of iterations over all projections. |

| | |
|---|---|
| **Name** | `augsys_tang_iter` |
| **Type** | `Natural` |
| **Valid Value** | `// Any` |
| **Problem Class** | Equality Constrained, Constrained |
| **JSON Param** | No |
| **Default** | `0` |
| **Description** | Number of iterations taken during the last iterate by the augmented system solve during the tangential step. |

| | |
|---|---|
| **Name** | `augsys_lmh_iter` |
| **Type** | `Natural` |
| **Valid Value** | `// Any` |
| **Problem Class** | Equality Constrained, Constrained |
| **JSON Param** | No |
| **Default** | `0` |
| **Description** | Number of iterations taken during the last iterate by the augmented system solve during the equality multiplier solve. |

| | |
|---|---|
| **Name** | `augsys_qn_iter_total` |
| **Type** | `Natural` |
| **Valid Value** | `// Any` |
| **Problem Class** | Equality Constrained, Constrained |
| **JSON Param** | No |
| **Default** | `0` |
| **Description** | Total number of iterations taken by the augmented system solve for the quasi-normal step. |

| | |
|---|---|
| **Name** | `augsys_pg_iter_total` |
| **Type** | `Natural` |
| **Valid Value** | `// Any` |
| **Problem Class** | Equality Constrained, Constrained |
| **JSON Param** | No |
| **Default** | `0` |
| **Description** | Total number of iterations taken by the augmented system solve when projecting the gradient prior to the tangential subproblem. |

| | |
|---|---|
| **Name** | `augsys_proj_iter_total` |
| **Type** | `Natural` |
| **Valid Value** | `// Any` |
| **Problem Class** | Equality Constrained, Constrained |
| **JSON Param** | No |
| **Default** | `0` |
| **Description** | Total number of iterations taken by the augmented system solve during the nullspace projection in the tangential subproblem. |

| | |
|---|---|
| **Name** | `augsys_tang_iter_total` |
| **Type** | `Natural` |
| **Valid Value** | `// Any` |
| **Problem Class** | Equality Constrained, Constrained |
| **JSON Param** | No |
| **Default** | `0` |

**Description** Total number of iterations taken by the augmented system solve during the tangential step.

**Name** `augsys_lmh_iter_total`

**Type** `Natural`

**Valid Value** `// Any`

**Problem Class** Equality Constrained, Constrained

**JSON Param** No

**Default** `0`

**Description** Total number of iterations taken by the augmented system solve during the equality multiplier solve.

**Name** `augsys_iter_total`

**Type** `Natural`

**Valid Value** `// Any`

**Problem Class** Equality Constrained, Constrained

**JSON Param** No

**Default** `0`

**Description** Total number of iterations taken by all augmented system solves.

**Name** `augsys_qn_err`

**Type** `Real`

**Valid Value** `// Any`

**Problem Class** Equality Constrained, Constrained

**JSON Param** No

**Default** `0.`

**Description** Error in the last augmented system solve for the quasi-normal step.

**Name** `augsys_pg_err`

**Type** `Real`

**Valid Value** `// Any`

**Problem Class** Equality Constrained, Constrained

**JSON Param** No

**Default** `0.`

**Description** Error in the last augmented system solve when projecting the gradient prior to the tangential subproblem.

**Name** augsys_proj_err

**Type** Real

**Valid Value** // Any

**Problem Class** Equality Constrained, Constrained

**JSON Param** No

**Default** 0.

**Description** Error in the last augmented system solve during the nullspace projection in the tangential subproblem. Note, since there are likely many projections during a single tangential subproblem, this represents the error from the last such solve.

**Name** augsys_tang_err

**Type** Real

**Valid Value** // Any

**Problem Class** Equality Constrained, Constrained

**JSON Param** No

**Default** 0.

**Description** Error in the last augmented system solve during the tangential step.

**Name** augsys_lmh_err

**Type** Real

**Valid Value** // Any

**Problem Class** Equality Constrained, Constrained

**JSON Param** No

**Default** 0.

**Description** Error in the last augmented system solve during the equality multiplier solve.

**Name** augsys_qn_err_target

**Type** Real

**Valid Value** // Any

**Problem Class** Equality Constrained, Constrained

**JSON Param** No

**Default** 0.

**Description** Target error in the last augmented system solve for the quasi-normal step.

**Name** augsys_pg_err_target

**Type** Real

**Valid Value** // Any

**Problem Class** Equality Constrained, Constrained

**JSON Param** No

**Default** 0.

**Description** Target error in the last augmented system solve when projecting the gradient prior to the tangential subproblem.

**Name** augsys_proj_err_target

**Type** Real

**Valid Value** // Any

**Problem Class** Equality Constrained, Constrained

**JSON Param** No

**Default** 0.

**Description** Target error in the last augmented system solve during the nullspace projection in the tangential subproblem. Note, since there are likely many projections during a single tangential subproblem, this represents the target error from the last such solve.

**Name** augsys_tang_err_target

**Type** Real

**Valid Value** // Any

**Problem Class** Equality Constrained, Constrained

**JSON Param** No

**Default** 0.

**Description** Target error in the last augmented system solve during the tangential step.

**Name** augsys_lmh_err_target

**Type** Real

**Valid Value** // Any

**Problem Class** Equality Constrained, Constrained

**JSON Param** No

**Default** `0.`

**Description** Target error in the last augmented system solve during the equality multiplier solve.

**Name** `augsys_qn_failed`

**Type** `Real`

**Valid Value** `// Any`

**Problem Class** Equality Constrained, Constrained

**JSON Param** No

**Default** `0.`

**Description** Number of failed quasinormal augmented system solves.

**Name** `augsys_pg_failed`

**Type** `Real`

**Valid Value** `// Any`

**Problem Class** Equality Constrained, Constrained

**JSON Param** No

**Default** `0.`

**Description** Number of failed projected gradient augmented system solves.

**Name** `augsys_proj_failed`

**Type** `Real`

**Valid Value** `// Any`

**Problem Class** Equality Constrained, Constrained

**JSON Param** No

**Default** `0.`

**Description** Number of failed nullspace projection augmented system solves.

**Name** `augsys_tang_failed`

**Type** `Real`

**Valid Value** `// Any`

**Problem Class** Equality Constrained, Constrained

**JSON Param** No

**Default** 0.

**Description** Number of tangential step augmented system solves.

**Name** augsys_lmh_failed

**Type** Real

**Valid Value** // Any

**Problem Class** Equality Constrained, Constrained

**JSON Param** No

**Default** 0.

**Description** Number of equality multiplier augmented system solves.

**Name** augsys_failed_total

**Type** Real

**Valid Value** // Any

**Problem Class** Equality Constrained, Constrained

**JSON Param** No

**Default** 0.

**Description** Total number of failed augmented system solves. In short, the theory for convergence to a local minima requires that augmented system solves meet their specified tolerance. Sometimes, a lower tolerance can be used and these tolerances are controlled by xi_all, xi_qn, xi_pg, xi_proj, xi_tang, and xi_lmh. However, even with a lower specified tolerance, the inexact composite step SQP method can still require a tighter tolerance in order to guarantee convergence. Generally, the algorithms are tolerant to a few failed solves. However, if there are failed solves at every iteration, then there's a problem with the given preconditioner or no preconditioner was specified. See the section (Optional) Define the preconditioners for more information on how to implement an appropriate preconditioner.

**Name** g_x

**Type** Y_Vector

**Valid Value** // Any

**Problem Class** Equality Constrained, Constrained

**JSON Param** No

**Default** Y::init(y_user)

**Description** Equality constraint evaluated a x, $g(x)$. We use this in the quasi-normal step as well as in the computation of the linear Taylor series at x in the direction dx_n. As a note, we output the norm of this vector each iteration under the label ||g(x)||.

**Name** `norm_gxtyp`

**Type** `Real`

**Valid Value**
```
state.norm_gxtyp >= Real(0.)
|| (state.iter==1 && state.norm_gxtyp!=state.norm_gxtyp)
```

**Problem Class** Equality Constrained, Constrained

**JSON Param** No

**Default** `std::numeric_limits<Real>::quiet_NaN()`

**Description** Norm of a typical equality constraint, which we define to be the norm of the equality constraint at the first iteration. Sometimes, we use `norm_gxtyp` with the stopping criteria described in `eps_constr`. Specifically, we only refer to this quantity when `eps_kind` is set to Relative. When `eps_kind` is set to Absolute, we ignore this value and instead use 1.0.

**Name** `norm_gpsgxtyp`

**Type** `Real`

**Valid Value**
```
state.norm_gpsgxtyp >= Real(0.)
|| (state.iter==1 &&
    state.norm_gpsgxtyp!=state.norm_gpsgxtyp)
```

**Problem Class** Equality Constrained, Constrained

**JSON Param** No

**Default** `std::numeric_limits<Real>::quiet_NaN()`

**Description** Norm of a typical value of $g'(\mathtt{x})^*g(\mathtt{x})$, which we define to be the value of this quantity at the first iteration. When we compute the quasinormal step, we compute the Cauchy point by finding the least-squares solution to the linearized equality constraint, $\min_{\partial x} \frac{1}{2}\|g'(x)\partial x + g(x)\|^2$. Here, the gradient is $g'(x)^*g'(x)\partial x + g'(x)^*g(x)$. Now, for the Cauchy point, we start with $\partial x = 0$, so the steepest descent direction becomes $\partial x = -g'(x)^*g(x)$. We find the Cauchy point, by doing an exact line-search along this direction in the objective for the least-squares problem above. Now, when $g'(x)^*g(x) = 0$, we sit at a local minima to the least-squares problem above. Generally, this is bad since we're not feasible and we don't have good information as to where to move to improve our infeasibility. Nevertheless, the tangential step will likely move us off that point unless we've already achieved optimality with respect to the Lagrangian. In any case, we require `norm_gpsgxtyp` to determine when the relative norm of $g'(x)^*g(x)$ is small and hence fall into this local minima.

**Name** `gpxdxn_p_gx`

**Type** `Y_Vector`

**Valid Value** `// Any`

**Problem Class** Equality Constrained, Constrained

**JSON Param** No

**Default** Y::init(y_user)

**Description** Linear Taylor series at x in the direction dx_n. We use this both in the predicted reduction as well as the residual predicted reduction.

**Name** gpxdxt

**Type** Y_Vector

**Valid Value** // Any

**Problem Class** Equality Constrained, Constrained

**JSON Param** No

**Default** Y::init(y_user)

**Description** Derivative of the constraint applied to the tangential step this is used in the residual predicted reduction.

**Name** norm_gpxdxnpgx

**Type** Real

**Valid Value** // Any

**Problem Class** Equality Constrained, Constrained

**JSON Param** No

**Default** std::numeric_limits<Real>::quiet_NaN()

**Description** Norm of gpxdxn_p_gx. We use this in the penalty parameter computation and predicted reduction.

**Name** dx_n

**Type** X_Vector

**Valid Value** // Any

**Problem Class** Equality Constrained, Constrained

**JSON Param** No

**Default** X::init(x_user)

**Description** Normal step. We output the norm of this vector at each iteration under the label ||dx_n||.

**Name** dx_ncp

**Type** X_Vector

**Valid Value** // Any

**Problem Class** Equality Constrained, Constrained

**JSON Param** No

**Default** `X::init(x_user)`

**Description** Cauchy point for normal step.

**Name** `dx_t`

**Type** `X_Vector`

**Valid Value** `// Any`

**Problem Class** Equality Constrained, Constrained

**JSON Param** No

**Default** `X::init(x_user)`

**Description** (Corrected) tangential step. We output the norm of this vector at each iteration under the label `||dx_t||`.

**Name** `dx_t_uncorrected`

**Type** `X_Vector`

**Valid Value** `// Any`

**Problem Class** Equality Constrained, Constrained

**JSON Param** No

**Default** `X::init(x_user)`

**Description** Tangential step prior to correction.

**Name** `dx_tcp_uncorrected`

**Type** `X_Vector`

**Valid Value** `// Any`

**Problem Class** Equality Constrained, Constrained

**JSON Param** No

**Default** `X::init(x_user)`

**Description** Cauchy point for tangential step prior to correction.

**Name** `H_dxn`

**Type** `X_Vector`

**Valid Value** `// Any`

**Problem Class** Equality Constrained, Constrained

**JSON Param** No

**Default** `X::init(x_user)`

**Description** Hessian applied to the normal step. We require this in `W_gradpHdxn` as well as the predicted reduction.

**Name** `W_gradpHdxn`

**Type** `X_Vector`

**Valid Value** `// Any`

**Problem Class** Equality Constrained, Constrained

**JSON Param** No

**Default** `X::init(x_user)`

**Description** Quantity grad $f(\mathtt{x})+g'(\mathtt{x})*\mathtt{y}+H\mathtt{dx\_n}$ projected into the null-space of the constraints. We require this in the tangential subproblem and the predicted reduction.

**Name** `H_dxtuncorrected`

**Type** `X_Vector`

**Valid Value** `// Any`

**Problem Class** Equality Constrained, Constrained

**JSON Param** No

**Default** `X::init(x_user)`

**Description** Hessian applied to the uncorrected tangential step. We require this in the predicted reduction.

**Name** `g_diag`

**Type** `FunctionDiagnostics`

**Valid Value** `// Any`

**Problem Class** Equality Constrained, Constrained

**JSON Param** Yes

**Default** `FunctionDiagnostics::NoDiagnostics`

**Description** Function diagnostics on $g$.

**Name** `y_diag`

**Type** `VectorSpaceDiagnostics`

**Valid Value** `// Any`

**Problem Class** Equality Constrained, Constrained

**JSON Param** Yes

**Default** `VectorSpaceDiagnostics::NoDiagnostics`

**Description** Vector space diagnostics on Y.

**Name** `qn_stop`

**Type** `QuasinormalStop`

**Valid Value** `// Any`

**Problem Class** Equality Constrained, Constrained

**JSON Param** No

**Default** `QuasinormalStop::Feasible`

**Description** Reason why the quasinormal problem exited.

**Name** `z`

**Type** `Z_Vector`

**Valid Value** `// Any`

**Problem Class** Inequality Constrained, Constrained

**JSON Param** No

**Default** `// mu inv(L(h(x))) e`

**Description** Inequality multiplier (dual variable or Lagrange multiplier.)

**Name** `dz`

**Type** `Z_Vector`

**Valid Value** `// Any`

**Problem Class** Inequality Constrained, Constrained

**JSON Param** No

**Default** `Z::init(z_user)`

**Description** Step in the inequality multiplier. Every iteration we set `z=z+dz`.

**Name** `h_x`

**Type** `Z_Vector`

**Valid Value** `// Any`

**Problem Class** Inequality Constrained, Constrained

**JSON Param** No

**Default** `Z::init(z_user)`

**Description** The inequality constraint evaluated at x. In theory, we can always just evaluate this when we need it. However, we require its computation both in the gradient as well as Hessian calculations. More specifically, when computing with SDP constraints, we require a factorization of this quantity. By caching it, we have the ability to cache the factorization.

**Name** `mu`

**Type** `Real`

**Valid Value** `state.mu > Real(0.)`

**Problem Class** Inequality Constrained, Constrained

**JSON Param** Yes

**Default** `1.0`

**Description** Interior point parameter. We use this as the target for the interior-point parameter estimate `mu_est`. As the interior point method progresses, we drive this value toward zero. As a note, we output `mu` at each iteration under the label `mu`.

**Name** `mu_est`

**Type** `Real`

**Valid Value** `state.mu_est == state.mu_est || state.iter == 1`

**Problem Class** Inequality Constrained, Constrained

**JSON Param** No

**Default** `std::numeric_limits<Real>::quiet_NaN()`

**Description** Current interior-point estimate. We define this as

$$\texttt{mu\_est} \equiv \frac{\langle \texttt{z}, \texttt{h\_x} \rangle}{\langle e, e \rangle}.$$

As a note, we output `mu_est` at each iteration under the label `mu_est`. Also note, we require this value to be small relative to `mu_typ` for convergence and control the relative decrease required with the parameter `eps_mu`.

**Name** `mu_typ`

**Type** `Real`

**Valid Value** `state.mu_typ > Real(0.) || state.iter==1`

**Problem Class** Inequality Constrained, Constrained

**JSON Param** No

**Default** `std::numeric_limits<Real>::quiet_NaN()`

**Description** Typical value for `mu`, which we define as the value of `mu_est` at the first iteration. Sometimes, we use `mu_typ` with the stopping criteria described in `eps_mu`. Specifically, we only refer to this quantity when `eps_kind` is set to Relative. When `eps_kind` is set to Absolute, we ignore this value and instead use 1.0.

**Name** `eps_mu`

**Type** `Real`

**Valid Value** `state.eps_mu > Real(0.)`

**Problem Class** Inequality Constrained, Constrained

**JSON Param** Yes

**Default** `1e-8`

**Description** Relative stopping tolerance for satisfying the complementary slackness condition for the inequality constraint. We satisfy this stopping criteria when

1. $|\texttt{mu} - \texttt{mu\_typ} \cdot \texttt{eps\_mu}| \leq \texttt{mu\_typ} \cdot \texttt{eps\_mu}$
2. $|\texttt{mu} - \texttt{mu\_est}| \leq \texttt{mu}$

**Name** `sigma`

**Type** `Real`

**Valid Value** `state.sigma > Real(0.) && state.sigma < Real(1.)`

**Problem Class** Inequality Constrained, Constrained

**JSON Param** Yes

**Default** `0.1`

**Description** Rate that we decrease the interior point parameter.

**Name** `gamma`

**Type** `Real`

**Valid Value** `state.gamma > Real(0.) && state.gamma < Real(1.)`

**Problem Class** Inequality Constrained, Constrained

**JSON Param** Yes

**Default** `0.99`

**Description** How close we move to the boundary during a single step. A step of 1.0 allows a step to touch the boundary of the inequality constraint in a single step, which is disallowed by the interior point algorithm.

**Name** `alpha_z`

**Type** `Real`

**Valid Value** `// Any`

**Problem Class** Inequality Constrained, Constrained

**JSON Param** No

**Default** `std::numeric_limits <Real>::quiet_NaN()`

**Description** How much we truncate `dz` in an interior point method in order to maintain strict feasibility. When 1.0, we do not truncate and take a full step. We output `alpha_z` at each iteration under the label `alpha_z`.

**Name** `h_diag`

**Type** `FunctionDiagnostics`

**Valid Value** `// Any`

**Problem Class** Inequality Constrained, Constrained

**JSON Param** Yes

**Default** `FunctionDiagnostics::NoDiagnostics`

**Description** Function diagnostics on $h$.

**Name** `z_diag`

**Type** `VectorSpaceDiagnostics`

**Valid Value** `// Any`

**Problem Class** Inequality Constrained, Constrained

**JSON Param** Yes

**Default** `VectorSpaceDiagnostics::NoDiagnostics`

**Description** Vector space diagnostics on Z.

# 5 Output

Optizelle generates a series of diagnostics while running that give information about the behavior and performance of the underlying algorithm. This information is organized into columns that are exactly 12 characters wide. When no information is available, we print a single dot, `.`. In this way, each column always has some sort of information, which makes the output easy to parse using standard Unix utilities such as `cut` or `awk`. For example, to only print the iteration, objective value, and norm of the step on the Rosenbrock example, we use the following commands on POSIX compliant systems:

```
./rosenbrock tr_newton.json | awk '{printf "%-12s%-12s%-12s\n", $1,$2,$4}'
```

and

```
./rosenbrock tr_newton.json | cut -c1-12,13-24,37-48
```

As far as the information in the columns themselves, we detail their meaning below. In terms of convergence, we require the values `||grad||`, `||g(x)||`, and `mu_est` be small relative to their starting value and control the relative decrease required with the parameters `eps_grad`, `eps_constr`, and `eps_mu`, respectively. In addition, if the value `||dx||` becomes too small relative to its starting value, we terminate the optimization. We control the amount of relative decrease allowed in `||dx||` with the parameter `eps_dx`.

| | |
|---|---|
| **Name** | `iter` |
| **State Param** | `iter` |
| **Problem Class** | Unconstrained, Equality Constrained, Inequality Constrained, Constrained |
| **Min** `msg_level` | 1 |
| **Description** | Current optimization iteration. If the value of this entry is `*`, then either a trust-region algorithm has rejected a step due to an unfavorable actual versus predicted reduction or a line-search algorithm has rejected a step due to a lack of sufficient decrease. In a trust-region method, we tune the rejection of steps with the parameter `eta1`. In a line-search method, we tune the rejection of steps with the parameter `c1`. |

| | |
|---|---|
| **Name** | `f(x)` |
| **State Param** | `f_x` |
| **Problem Class** | Unconstrained, Equality Constrained, Inequality Constrained, Constrained |
| **Min** `msg_level` | 1 |
| **Description** | Value of the objective function at the start of the specified iteration. |

**Name** ||grad||

**State Param** None

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**Min msg_level** 1

**Description** Norm of the gradient of either the objective function or the Lagrangian, which we describe in the description of eps_grad. We use this value within our gradient stopping condition described by the parameter eps_grad. In general, we need this value to be small relative to the starting value for convergence.

**Name** ||dx||

**State Param** None

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**Min msg_level** 1

**Description** Norm of the step taken during the last iteration. We calculate this value by taking the norm of the value found in dx and use this within our stopping condition controlled by eps_dx. As a safeguard, we exit the optimization if this value becomes too small relative to the starting value.

**Name** ||g(x)||

**State Param** None

**Problem Class** Equality Constrained, Constrained

**Min msg_level** 1

**Description** Norm of the equality constraint at the start of the optimization iteration, which we calculate in g_x. We use this value within our equality constraint feasibility stopping condition described by the parameter eps_constr. In short, we need this value to be small relative to the starting value for convergence. If the starting value is already acceptably small, then we have started with a feasible solution. In this case, we may need to adjust eps_constr to something like 1.0, which states that we do not seek relative improvement in the infeasibility.

**Name** mu_est

**State Param** mu_est

**Problem Class** Inequality Constrained, Constrained

**Min msg_level** 1

**Description** Current interior-point estimate. We use this value within our complementary slackness stopping condition described by the parameter eps_mu. In short, we need this value to be small relative to its starting value for convergence. We control the relative decrease required with the parameter eps_mu.

**Name** merit(x)

**State Param** None

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**Min msg_level** 2

**Description** Value of the merit function at the start of the specified iteration. We specify the various merit functions in the description of the parameter ared.

**Name** trunc_iter

**State Param** trunc_iter

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**Min msg_level** 2

**Description** Number of iterations used by truncated CG when solving the optimality system. We tune the maximum number of truncated CG iterations with the parameter trunc_iter_max.

**Name** trunc_err

**State Param** trunc_err

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**Min msg_level** 2

**Description** Error in truncated CG when solving the optimality system. We control this error with the parameter eps_trunc and indirectly affect it with the parameters trunc_orthog_storage_max and trunc_orthog_iter_max.

**Name** trunc_stop

**State Param** trunc_stop

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**Min msg_level** 2

**Description** Why truncated CG terminated. Although we shorten the strings, we describe each possible outcome in the enumerated type TruncatedStop.

**Name** ared

**State Param** ared

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**Min msg_level** 2

**Description** Actual reduction in the merit function between the current iterate and the iterate after taking the trial step.

**Name** pred

**State Param** pred

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**Min** msg_level 2

**Description** Predicted reduction in the merit function between the current iterate and the iterate after taking the trial step.

**Name** ared/pred

**State Param** None

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**Min** msg_level 2

**Description** Actual versus predicted reduction. Simply, we divide the outputs ared and pred. For a perfect model, this ratio is 1.0.

**Name** delta

**State Param** delta

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**Min** msg_level 2

**Description** Trust-region radius.

**Name** ls_iter

**State Param** ls_iter

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**Min** msg_level 2

**Description** Number of iterations taken by the line search. We tune the maximum number of line-search iterations with the parameter ls_iter_max and indirectly control the number of iterations with the parameter eps_ls.

**Name** alpha

**State Param** alpha

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**Min** msg_level 2

**Description** Actual line-search step length.

**Name** alpha0

**State Param** alpha0

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**Min** msg_level 2

**Description** Base line-search step length.

**Name** qn_stop

**State Param** qn_stop

**Problem Class** Equality Constrained, Constrained

**Min msg_level** 2

**Description** Reason why the quasinormal problem exited.

**Name** aug_fail

**State Param** augsys_failed_total

**Problem Class** Equality Constrained, Constrained

**Min msg_level** 2

**Description** Total number of failed augmented system solves.

**Name** mu

**State Param** mu

**Problem Class** Inequality Constrained, Constrained

**Min msg_level** 2

**Description** Interior point parameter.

**Name** alpha_x

**State Param** alpha_x

**Problem Class** Inequality Constrained, Constrained

**Min msg_level** 2

**Description** Amount we truncate dx in order to maintain feasibility with respect to the inequality constraint.

**Name** alpha_z

**State Param** alpha_z

**Problem Class** Inequality Constrained, Constrained

**Min msg_level** 2

**Description** Amount we truncate dz in order to maintain feasibility with respect to the inequality multiplier. Note, we only reference this when we are using a primal-dual interior point method.

**Name** safe_fail

**State Param** safeguard_failed

**Problem Class** Inequality Constrained, Constrained

**Min msg_level** 2

**Description** Number of failed safe-guard steps during the last iteration. Note, we only reference this when using a trust-region method.

**Name** alpha_x_qn

**State Param** alpha_x_qn

**Problem Class** Constrained

**Min msg_level** 2

**Description** Amount we truncate dx_n in order to maintain feasibility with respect to the inequality constraint.

**Name** glb_itr_tot

**State Param** glob_iter_total

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**Min msg_level** 3

**Description** Total number of globalization iterations taken across all iterations.

**Name** trc_itr_tot

**State Param** trunc_iter_total

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**Min msg_level** 3

**Description** Total number of iterations used by truncated CG when solving the optimality system. We typically use this to determine how many Hessian-vector products we've computed over the entire optimization run.

**Name** ||dx_n||

**State Param** Equality Constrained, Constrained

**Problem Class** None

**Min msg_level** 3

**Description** Norm of the quasinormal step, dx_n, taken during the last iteration.

**Name** ||dx_t||

**State Param** Equality Constrained, Constrained

**Problem Class** None

**Min msg_level** 3

| | |
|---|---|
| **Description** | Norm of the tangential step, dx_t, taken during the last iteration. |

| | |
|---|---|
| **Name** | qn_iter |
| **State Param** | augsys_qn_iter |
| **Problem Class** | Equality Constrained, Constrained |
| **Min msg_level** | 3 |
| **Description** | Number of iterations taken during the last iterate by the augmented system solve for the quasi-normal step. |

| | |
|---|---|
| **Name** | qn_iter_tot |
| **State Param** | augsys_qn_iter_total |
| **Problem Class** | Equality Constrained, Constrained |
| **Min msg_level** | 3 |
| **Description** | Total number of iterations taken by the augmented system solve for the quasi-normal step. |

| | |
|---|---|
| **Name** | qn_err |
| **State Param** | augsys_qn_err |
| **Problem Class** | Equality Constrained, Constrained |
| **Min msg_level** | 3 |
| **Description** | Error in the last augmented system solve for the quasi-normal step. |

| | |
|---|---|
| **Name** | qn_err_trg |
| **State Param** | augsys_qn_err_target |
| **Problem Class** | Equality Constrained, Constrained |
| **Min msg_level** | 3 |
| **Description** | Target error in the last augmented system solve for the quasi-normal step. |

| | |
|---|---|
| **Name** | qn_fail |
| **State Param** | augsys_qn_failed |
| **Problem Class** | Equality Constrained, Constrained |
| **Min msg_level** | 3 |
| **Description** | Number of failed quasinormal augmented system solves. |

| | |
|---|---|
| **Name** | pg_iter |
| **State Param** | augsys_pg_iter |

**Problem Class** Equality Constrained, Constrained

**Min** msg_level 3

**Description** Number of iterations taken during the last iterate by the augmented system solve when projecting the gradient prior to the tangential subproblem.

**Name** pg_iter_tot

**State Param** augsys_pg_iter_total

**Problem Class** Equality Constrained, Constrained

**Min** msg_level 3

**Description** Total number of iterations taken by the augmented system solve when projecting the gradient prior to the tangential subproblem.

**Name** pg_err

**State Param** augsys_pg_err

**Problem Class** Equality Constrained, Constrained

**Min** msg_level 3

**Description** Error in the last augmented system solve when projecting the gradient prior to the tangential subproblem.

**Name** pg_err_trg

**State Param** augsys_pg_err_target

**Problem Class** Equality Constrained, Constrained

**Min** msg_level 3

**Description** Target error in the last augmented system solve when projecting the gradient prior to the tangential subproblem.

**Name** pg_fail

**State Param** augsys_pg_failed

**Problem Class** Equality Constrained, Constrained

**Min** msg_level 3

**Description** Number of failed projected gradient augmented system solves.

**Name** pr_iter

**State Param** augsys_proj_iter

**Problem Class** Equality Constrained, Constrained

**Min** msg_level 3

**Description** Number of iterations taken during the last iterate by the augmented system solve during the nullspace projection in the tangential subproblem. Since there are likely many projections, this is the total number of iterations over all projections.

**Name** pr_iter_tot

**State Param** augsys_proj_iter_total

**Problem Class** Equality Constrained, Constrained

**Min** msg_level 3

**Description** Total number of iterations taken by the augmented system solve during the nullspace projection in the tangential subproblem.

**Name** pr_err

**State Param** augsys_proj_err

**Problem Class** Equality Constrained, Constrained

**Min** msg_level 3

**Description** Error in the last augmented system solve during the nullspace projection in the tangential subproblem. Note, since there are likely many projections during a single tangential subproblem, this represents the error from the last such solve.

**Name** pr_err_trg

**State Param** augsys_proj_err_target

**Problem Class** Equality Constrained, Constrained

**Min** msg_level 3

**Description** Target error in the last augmented system solve during the tangential step.

**Name** pr_fail

**State Param** augsys_proj_failed

**Problem Class** Equality Constrained, Constrained

**Min** msg_level 3

**Description** Number of failed nullspace projection augmented system solves.

**Name** tg_iter

**State Param** augsys_tang_iter

**Problem Class** Equality Constrained, Constrained

**Min** msg_level 3

**Description** Number of iterations taken during the last iterate by the augmented system solve during the tangential step.

| | |
|---|---|
| **Name** | tg_iter_tot |
| **State Param** | augsys_tang_iter_total |
| **Problem Class** | Equality Constrained, Constrained |
| **Min** msg_level | 3 |
| **Description** | Total number of iterations taken by the augmented system solve during the tangential step. |

| | |
|---|---|
| **Name** | tg_err |
| **State Param** | augsys_tang_err |
| **Problem Class** | Equality Constrained, Constrained |
| **Min** msg_level | 3 |
| **Description** | Error in the last augmented system solve during the tangential step. |

| | |
|---|---|
| **Name** | tg_err_trg |
| **State Param** | augsys_tang_err_target |
| **Problem Class** | Equality Constrained, Constrained |
| **Min** msg_level | 3 |
| **Description** | Target error in the last augmented system solve during the tangential step. |

| | |
|---|---|
| **Name** | tg_fail |
| **State Param** | augsys_tang_failed |
| **Problem Class** | Equality Constrained, Constrained |
| **Min** msg_level | 3 |
| **Description** | Number of failed tangential step augmented system solves. |

| | |
|---|---|
| **Name** | lm_iter |
| **State Param** | augsys_lmh_iter |
| **Problem Class** | Equality Constrained, Constrained |
| **Min** msg_level | 3 |
| **Description** | Number of iterations taken during the last iterate by the augmented system solve during the equality multiplier solve. |

| | |
|---|---|
| **Name** | lm_iter_tot |
| **State Param** | augsys_lmh_iter_total |
| **Problem Class** | Equality Constrained, Constrained |

**Min** msg_level 3

**Description** Total number of iterations taken by the augmented system solve during the equality multiplier solve.

**Name** lm_err

**State Param** augsys_lmh_err

**Problem Class** Equality Constrained, Constrained

**Min** msg_level 3

**Description** Error in the last augmented system solve during the equality multiplier solve.

**Name** lm_err_trg

**State Param** augsys_lmh_err_target

**Problem Class** Equality Constrained, Constrained

**Min** msg_level 3

**Description** Target error in the last augmented system solve during the equality multiplier solve.

**Name** lm_fail

**State Param** augsys_lmh_failed

**Problem Class** Equality Constrained, Constrained

**Min** msg_level 3

**Description** Number of failed equality multiplier augmented system solves.

**Name** aug_itr_tot

**State Param** augsys_iter_total

**Problem Class** Equality Constrained, Constrained

**Min** msg_level 3

**Description** Total number of iterations taken by all augmented system solves. We use this to help determine the overall expense of the augmented system solver and its precondition.

# 6 Advanced API

Optizelle contains many additional features such as customizing the output and defining custom vector spaces. We detail these features below.

## 6.1 User-defined messaging

By default, we output messages from Optizelle to `stdout`. However, in some environments, we require different behavior. For example,

- When we use Optizelle in a program with a GUI, we may not to display the output to a separate window.
- When using MPI in a distributed, parallel environment we likely want to restrict our output to only the rank 0 processor.

In these cases, we want to define a new messaging object. Messaging objects are simply functions that accept a string and print it accordingly. In code, we specify this object as:

**Language** C++

**Structure** `Optizelle::Messaging::t`

**Interface** Function matches type

**Code**

```
namespace Optizelle{
    // Defines how we output messages to the user
    namespace Messaging {
        // At its core, we take in a string and then write it somewhere
        typedef std::function<void(std::string const & msg)> t;
    }
}
```

**Language** Python

**Structure** `Optizelle.Messaging.t`

**Interface** Function matches type

**Code**

```
def t(msg):
    """At its core, we take in a string and then write it somewhere"""
    raise Optizelle.Exception.t("Undefined messaging function")
```

**Language** MATLAB/Octave

**Structure** `Optizelle.Messaging.t`

**Interface** Function matches type

**Code**

```
% At its core, we take in a string and then write it somewhere
Optizelle.Messaging.t = @(x)error('Undefined messaging function');
```

Once we define a custom messaging object, we are free to pass it to Optizelle, which occurs when we call the function `getMin`. We describe this process in the section Call the optimization solver. As an example, we modify the messaging object in our Rosenbrock advanced API example:

**Language** C++

**Code**

```
// Define a custom messaging object
void mymessaging(std::string const & msg) {
    std::cout << "PRINT: " << msg << std::endl;
}
```

**Language** Python

**Code**

```
# Define a custom messaging object
def mymessaging(msg):
    """Prints out normal diagnostic information"""
    sys.stdout.write("PRINT: %s\n" %(msg))
```

**Language** MATLAB/Octave

**Code**

```
% Define a custom messaging object
function MyMessaging(msg)
    fprintf('PRINT: %s\n',msg);
end
```

## 6.2 Handling errors

In general, Optizelle handles algorithmic errors gracefully and will exit the optimization with the current best solution. However, errors in the problem setup or functions provided by the user cause Optizelle to exit its routines immediately.

The mechanism for handling errors depends on the type and interface. For errors that originate with Optizelle, we use the following scheme

**Language** C++

**Structure** `Optizelle::Exception::t`

**Interface** Exception handling

**Code**

```
namespace Optizelle {namespace Exception {
    struct t : public std::runtime_error {
        using std::runtime_error::runtime_error;
    };
}}
```

**Language** Python

**Structure** `Optizelle.Exception.t`

**Interface** Exception handling

**Code**

```python
class t(Exception):
    """Type for Optizell's exceptions"""
    pass
```

**Language** MATLAB/Octave

**Structure** `error`

**Interface** Native error function

For errors that originate within the user code, we exit Optizelle and propagate the original error back to parent code. Typically, the best way to throw an error in the user code is by exceptions in C++ and Python and the `error` function in MATLAB/Octave. As an example, reading an invalid parameter from file raises an Optizelle error. We catch this error with the following code

**Language** C++

**Code**

```cpp
    // Read parameters from file
    try {
        Optizelle::json::Unconstrained <Real,XX>::read(fname,state);
    } catch(Optizelle::Exception::t const & e) {
        // Convert the error message to a string
        msg = Optizelle::Exception::to_string(e);

        // Print the error message directly
        Optizelle::Exception::to_stderr(e);
    }
```

**Language** Python

**Code**

```python
# Read parameters from file
try:
    Optizelle.json.Unconstrained.read(XX,fname,state);
except Optizelle.Exception.t as e:
    # Convert the error message to a string
    msg = e.message

    # Print the error message directly
    print e
```

**Language** MATLAB/Octave

**Code**

```matlab
    try
        state = Optizelle.json.Unconstrained.read(XX,fname,state);
    catch e
        % Convert the error message into a string
        msg = e.message;

        % Print the message directly
        disp(e.message);
    end
```

## 6.3 Customized vector spaces

In continuous optimization, we most often optimize over a simple vector of numbers in $\mathbb{R}^m$. If that's the case, we provide a reasonable implementation of this vector space and describe it in section Import or define the appropriate vector spaces. However, in some situations we want to use a different space. For example:

- In PDE constrained optimization, we may want to optimize over a space of functions such as $L^2(\Omega)$.
- In certain relaxations to discrete optimization problems, we must optimize over the space of symmetric, positive definite matrices.
- When the variables in $\mathbb{R}^m$ have radically different scalings, we may need to alter the inner product to normalize our variables.
- On large-scale problems with billions of variables, we must store the vectors in parallel and compute operations using a messaging system such as MPI.

In each of these cases, we need to define a custom vector space for our problem. Each custom vector space requires us to define the following operations:

**Name** `init`

**Definition**

| | | |
|---|---|---|
| **C++** | `init(x)` | $init \leftarrow \xi(W)$ where $x \in W$ |
| **Python** | `init(x)` | $init \leftarrow \xi(W)$ where $x \in W$ |
| **MATLAB/Octave** | `init(x)` | $init \leftarrow \xi(W)$ where $x \in W$ |

**Description** Initializes memory for a new vector. Here, the function $\xi : \{X, Y, Z\} \to X \cup Y \cup Z$ denotes a choice function that selects an arbitrary element from the appropriate set. Essentially, this states that we want a valid element in the vector space, but we don't care what the element is.

**Name** `copy`

**Definition**

| | | |
|---|---|---|
| **C++** | `copy(x,y)` | $y \leftarrow x$ |
| **Python** | `copy(x,y)` | $y \leftarrow x$ |
| **MATLAB/Octave** | `copy(x)` | $copy \leftarrow x$ |

**Description** In C++ and Python, a shallow copy of the vector $x$ into the vector $y$. In MATLAB/Octave, return the vector $x$

.

**Name** `scal`

**Definition**

| | | |
|---|---|---|
| **C++** | `scal(alpha,x)` | $x \leftarrow \alpha x$ |
| **Python** | `scal(alpha,x)` | $x \leftarrow \alpha x$ |
| **MATLAB/Octave** | `scal(alpha,x)` | $scal \leftarrow \alpha x$ |

**Description** In C++ and Python, overwrite $x$ with $\alpha x$. In MATLAB/Octave, return $\alpha x$.

**Name** `axpy`

**Definition**

| | |
|---|---|
| **C++** | `axpy(alpha,x,y)` |
| | $y \leftarrow \alpha x + y$ |
| **Python** | `axpy(alpha,x,y)` |
| | $y \leftarrow \alpha x + y$ |
| **MATLAB/Octave** | `axpy(alpha,x,y)` |
| | $axpy \leftarrow \alpha x + y$ |

**Description** In C++ and Python, overwrite $y$ with $\alpha x + y$. In MATLAB/Octave, return $\alpha x + y$.

**Name** `innr`

**Definition**

| | |
|---|---|
| **C++** | `innr(x,y)` |
| | $innr \leftarrow \langle x, y\rangle$ |
| **Python** | `innr(x,y)` |
| | $innr \leftarrow \langle x, y\rangle$ |
| **MATLAB/Octave** | `innr(x,y)` |
| | $innr \leftarrow \langle x, y\rangle$ |

**Description** Return the inner product between $x$ and $y$.

**Name** `zero`

**Definition**

| | |
|---|---|
| **C++** | `zero(x)` |
| | $x \leftarrow 0$ |
| **Python** | `zero(x)` |
| | $x \leftarrow 0$ |
| **MATLAB/Octave** | `zero(x)` |
| | $zero \leftarrow 0$ |

**Description** In C++ and Python, overwrite $x$ with 0. In MATLAB/Octave, return 0. Note, this is not necessarily the same as `scal(0.,x)` since, in practice, $x$ may contain NaNs and Infs. As such, we consider `zero` to be a safe operation that returns 0. whereas `scal` may be an unsafe operation.

**Name** `rand`

**Definition**

| | |
|---|---|
| **C++** | `rand(x)` |
| | $x \leftarrow \psi(W)$ where $x \in W$ |
| **Python** | `rand(x)` |
| | $x \leftarrow \psi(W)$ where $x \in W$ |
| **MATLAB/Octave** | `rand(x)` |
| | $rand \leftarrow \psi(W)$ where $x \in W$ |

**Description** In C++ and Python, overwrite $x$ with a random vector. In MATLAB/Octave, return a random vector. Here, the function $\psi : \{X, Y, Z\} \rightarrow X \cup Y \cup Z$ denotes a stochastic choice function that randomly selects an element from the appropriate set. Essentially, this states that we want a valid, random element in the vector space. Primarily, we use these vectors for our diagnostic tests controlled by the parameters `f_diag`, `g_diag`, and `h_diag`.

In addition, the vector space associated with the codomain of the inequality constraints, $Z$, requires the following operations:

**Name** `prod`

**Definition**

| | | |
|---|---|---|
| **C++** | `prod(x,y,z)` | $z \leftarrow x \circ y$ |
| **Python** | `prod(x,y,z)` | $z \leftarrow x \circ y$ |
| **MATLAB/Octave** | `prod(x,y)` | $prod \leftarrow x \circ y$ |

**Description** In C++ and Python, overwrite $z$ with $x \circ y$. In MATLAB/Octave, return $x \circ y$. Here, $\circ$ denotes a pseudo-Jordan product between two elements. We say pseudo-Jordan in the sense that we do not require a full Euclidean-Jordan algebra. Instead, we drop the requirement for commutativity. Hence, for linear bound constraints, we define that

$$[x \circ y]_i = x_i y_i.$$

Hence, the product denotes the pointwise or Hadamard product. For second-order cone constraints, we define that

$$\begin{bmatrix} x_0 \\ \bar{x} \end{bmatrix} \circ \begin{bmatrix} y_0 \\ \bar{y} \end{bmatrix} = \begin{bmatrix} x_0 y_0 + \bar{x}^T \bar{y} \\ x_0 \bar{y} + y_0 \bar{x} \end{bmatrix}.$$

For semidefinite programming, we have that

$$X \circ Y = XY.$$

Alternatively, we can define that

$$X \circ Y = \frac{XY + YX}{2},$$

but the inverse operation `linv` below becomes far less efficient.

**Name** `id`

**Definition**

| | | |
|---|---|---|
| **C++** | `id(x)` | $x \leftarrow e$ |
| **Python** | `id(x)` | $x \leftarrow e$ |
| **MATLAB/Octave** | `id(x)` | $id \leftarrow e$ |

**Description** In C++ and Python, overwrite $x$ with $e$. In MATLAB/Octave, return $e$. In this function, $e$ denotes the identity element for the Jordan algebra. Hence, this function creates element $e$ so that $x \circ e = x$. For linear bound constraints, $e$ denotes the vector of all ones. For second-order cone constraints, $e = \begin{bmatrix} 1 & 0 & \dots & 0 \end{bmatrix}^T$. For semidefinite constraints, $e = I$

**Name** `linv`

**Definition**

| | | |
|---|---|---|
| **C++** | `linv(x,y,z)` | |
| | $z \leftarrow L(x)^{-1}y$ | |
| **Python** | `linv(x,y,z)` | |
| | $z \leftarrow L(x)^{-1}y$ | |
| **MATLAB/Octave** | `linv(x,y)` | |
| | $linv \leftarrow L(x)^{-1}y$ | |

**Description** In C++ and Python, overwrite $z$ with $L(x)^{-1}y$. In MATLAB/Octave, return $L(x)^{-1}y$. Here, the function `linv` denotes the inverse operation to `prod`. Note, `prod` defines a bilinear operation so that there exists a linear operator $L(x)$ such that $x \circ y = L(x)y$. The function `linv` computes the action of the *inverse* of $L(x)$ on a vector. For linear bound constraints, $L(x) = \text{Diag}(x)$, where $\text{Diag}(x)$ denotes the diagonal matrix with $x$ on the diagonal. For second-order cone constraints, $L(x) = \text{Arw}(x)$ where we define $\text{Arw}(x)$ as

$$\text{Arw}\left(\begin{bmatrix} x_0 \\ \bar{x} \end{bmatrix}\right) = \begin{bmatrix} x_0 & \bar{x}^T \\ \bar{x} & x_0 I \end{bmatrix}.$$

For semidefinite constraints, we can either define that $L(X) = X$ or that $L(X) = \frac{X \cdot + \cdot X}{2}$. Generally, it is preferable to use the first definition since $L(X)^{-1} = X^{-1}$. In the second case, we require the solution of the Sylvester equations.

**Name** `barr`

**Definition**

| | | |
|---|---|---|
| **C++** | `barr(x)` | |
| | $barr \leftarrow \phi(x)$ where $x \circ \nabla\phi(x) = e$ | |
| **Python** | `barr(x)` | |
| | $barr \leftarrow \phi(x)$ where $x \circ \nabla\phi(x) = e$ | |
| **MATLAB/Octave** | `barr(x)` | |
| | $barr \leftarrow \phi(x)$ where $x \circ \nabla\phi(x) = e$ | |

**Description** Return the result of the barrier function applied to a vector. Here, the function $\phi : Z \to \mathbb{R}$ denotes the barrier function, which we require to satisfy

$$x \circ \nabla\phi(x) = e.$$

For linear bound constraints, this is simply the log-barrier function

$$\phi(x) = \sum_{i=1}^{m} \log(x_i).$$

For second-order cone constraints, we define this as

$$\phi\left(\begin{bmatrix} x_0 \\ \bar{x} \end{bmatrix}\right) = \frac{1}{2}\log(x_0^2 - \langle \bar{x}, \bar{x} \rangle).$$

For semidefinite constraints, we define this as

$$\phi(X) = \log(\det(X))$$

where $\det(X)$ denotes the determinant of $X$.

**Name** `srch`

**Definition**

| | | |
|---|---|---|
| | C++ | `srch(x,y)` |
| | | $srch \leftarrow \arg\max\{\alpha \in \mathbb{R} : \alpha x + y \succeq 0, \alpha \geq 0\}$ |
| | Python | `srch(x,y)` |
| | | $srch \leftarrow \arg\max\{\alpha \in \mathbb{R} : \alpha x + y \succeq 0, \alpha \geq 0\}$ |
| | MATLAB/Octave | `srch(x,y)` |
| | | $srch \leftarrow \arg\max\{\alpha \in \mathbb{R} : \alpha x + y \succeq 0, \alpha \geq 0\}$ |

**Description** Return how far we can move in the direction $x$ from the point $y$ before violating nonnegativity. In other words, the function `srch` denotes the search function used to maintain strict feasibility with respect to the inequality constraint. We define this as

$$\arg\max\{\alpha \in \mathbb{R} : \alpha x + y \succeq 0, \alpha \geq 0\}$$

where we assume $y \succ 0$. Hence, $\alpha$ denotes the maximum distance we can move in the direction $x$ from $y$ so that $\alpha x + y$ remains feasible. Note, sometimes this number is infinite. If this is the case, we must return `Inf`.

**Name** `symm`

**Definition**

| | | |
|---|---|---|
| | C++ | `symm(x)` |
| | | $x \leftarrow \pi(x)$ where $\pi(x \circ y) = \pi(y \circ x)$ |
| | Python | `symm(x)` |
| | | $x \leftarrow \pi(x)$ where $\pi(x \circ y) = \pi(y \circ x)$ |
| | MATLAB/Octave | `symm(x)` |
| | | $symm \leftarrow \pi(x)$ where $\pi(x \circ y) = \pi(y \circ x)$ |

**Description** In C++ and Python, overwrite $x$ with its symmetrization. In MATLAB/Octave, return the symmetrization of $x$. Here, the function $\pi : Z \to Z$ denotes the symmetrization operator. We require this operator since we relax the commutativity requirement from the Euclidean-Jordan algebra. For linear bound constraints and second-order cone constraints, this operation does nothing. In addition, for semidefinite constraints where $X \circ Y = \frac{XY+YX}{2}$, this operation does nothing. However, for semidefinite constraints where $X \circ Y = XY$, we may use symmetrization,

$$\pi(X) = \frac{X + X^T}{2},$$

or more generally the *similar symmetrization* operator,

$$\pi_P(X) = \frac{(PXP^{-1} + (PXP^{-1})^T}{2},$$

where we require $P$ to be nonsingular.

Next, we require these vector-space functions be encapsulated in the following structures:

**Language** C++

**Interface** Templated struct with static members and a single typedef called `Vector`

**Description** A vector space in C++ must be declared as a templated struct with static members. As far as the template parameter, we template on our real scalar type and require that each of the functions that accept or return a scalar use this type. This template parameter allows us to insure that each of the vector spaces uses the same real type, which is important for consistency. Next, each of the above functions must be included and declared static. This allows us to access the functions without instantiating the struct. We also require a single typedef called `Vector`. This defines the vector type used by each of the vector-space functions. In addition to the typedef, we require that this vector type implement move semantics, which includes both the move constructor as well as move semantics for the assignment operator. Note, items in the standard library all properly implement move semantics. As such, as long as we use `std::vector`, `std::unique_ptr`, or `std::shared_ptr`, we satisfy this requirement.

**Language** Python

**Interface** Class with static methods

**Description** A vector space in Python must be declared as a class consisting entirely of static methods. In other words, we require a class that implements all of the above vector-space functions where we decorate each function definition with the decorator `@staticmethod`.

**Language** MATLAB/Octave

**Interface** Structure array

**Description** A vector space in MATLAB/Octave must be declared as a structure array with all of the above methods present.

As an example, we define and use a custom vector space for $\mathbb{R}^m$ in our Rosenbrock advanced API example:

**Language** C++

**Code**

```
// Defines the vector space used for optimization.
template <typename Real>
struct MyVS {
    typedef std::vector <Real> Vector;

    // Memory allocation and size setting
    static Vector init(Vector const & x) {
        return std::move(Vector(x.size()));
    }

    // y <- x (Shallow. No memory allocation.)
    static void copy(Vector const & x, Vector & y) {
        for(Natural i=0;i<x.size();i++){
            y[i]=x[i];
        }
    }

    // x <- alpha * x
    static void scal(const Real& alpha, Vector & x) {
        for(Natural i=0;i<x.size();i++){
            x[i]=alpha*x[i];
        }
    }

    // x <- 0
    static void zero(Vector & x) {
        for(Natural i=0;i<x.size();i++){
            x[i]=0.;
        }
    }
```

```
// y <- alpha * x + y
static void axpy(const Real& alpha, Vector const & x, Vector & y) {
    for(Natural i=0;i<x.size();i++){
        y[i]=alpha*x[i]+y[i];
    }
}

// innr <- <x,y>
static Real innr(Vector const & x,Vector const & y) {
    Real z=0;
    for(Natural i=0;i<x.size();i++)
        z+=x[i]*y[i];
    return z;
}

// x <- random
static void rand(Vector & x){
    std::mt19937 gen(1);
    std::uniform_real_distribution<Real> dis(Real(0.),Real(1.));
    for(Natural i=0;i<x.size();i++)
        x[i]=Real(dis(gen));
}
// Jordan product, z <- x o y.
static void prod(Vector const & x, Vector const & y, Vector & z) {
    for(Natural i=0;i<x.size();i++)
        z[i]=x[i]*y[i];
}

// Identity element, x <- e such that x o e = x.
static void id(Vector & x) {
    for(Natural i=0;i<x.size();i++)
        x[i]=Real(1.);
}

// Jordan product inverse, z <- inv(L(x)) y where L(x) y = x o y.
static void linv(Vector const & x,Vector const & y,Vector & z) {
    for(Natural i=0;i<x.size();i++)
        z[i]=y[i]/x[i];
}

// Barrier function, barr <- barr(x) where x o grad barr(x) = e.
static Real barr(Vector const & x) {
    Real z=Real(0.);
    for(Natural i=0;i<x.size();i++)
        z+=log(x[i]);
    return z;
}

// Line search, srch <- argmax {alpha \in Real >= 0 : alpha x + y >= 0}
// where y > 0.
static Real srch(Vector const & x,Vector const & y) {
    // Line search parameter
    Real alpha=std::numeric_limits <Real>::infinity();

    // Search for the optimal linesearch parameter.
    for(Natural i=0;i<x.size();i++) {
```

```
            if(x[i] < Real(0.)) {
                Real alpha0 = -y[i]/x[i];
                alpha = alpha0 < alpha ? alpha0 : alpha;
            }
        }

        return alpha;
    }

    // Symmetrization, x <- symm(x) such that L(symm(x)) is a symmetric
    // operator.
    static void symm(Vector & x) { }
};
```

**Language** Python

**Code**

```
# Defines the vector space used for optimization.
class MyVS(object):
    @staticmethod
    def init(x):
        """Memory allocation and size setting"""
        return copy.deepcopy(x)

    @staticmethod
    def copy(x,y):
        """y <- x (Shallow. No memory allocation.)"""
        y[:]=x[:]

    @staticmethod
    def scal(alpha,x):
        """x <- alpha * x"""
        for i in xrange(0,len(x)):
            x[i]=alpha*x[i]

    @staticmethod
    def zero(x):
        """x <- 0"""
        for i in xrange(0,len(x)):
            x[i]=0.

    @staticmethod
    def axpy(alpha,x,y):
        """y <- alpha * x + y"""
        for i in xrange(0,len(x)):
            y[i]=alpha*x[i]+y[i]

    @staticmethod
    def innr(x,y):
        """<- <x,y>"""
        return reduce(lambda z,xy:xy[0]*xy[1]+z,zip(x,y),0.)

    @staticmethod
    def rand(x):
        """x <- random"""
        for i in xrange(0,len(x)):
            x[i]=random.uniform(0.,1.)
```

```
    @staticmethod
    def prod(x,y,z):
        """Jordan product, z <- x o y"""
        for i in xrange(0,len(x)):
            z[i]=x[i]*y[i]

    @staticmethod
    def id(x):
        """Identity element, x <- e such that x o e = x"""
        for i in xrange(0,len(x)):
            x[i]=1.

    @staticmethod
    def linv(x,y,z):
        """Jordan product inverse, z <- inv(L(x)) y where L(x) y = x o y"""
        for i in xrange(0,len(x)):
            z[i]=y[i]/x[i]

    @staticmethod
    def barr(x):
        """Barrier function, <- barr(x) where x o grad barr(x) = e"""
        return reduce(lambda x,y:x+math.log(y),x,0.)

    @staticmethod
    def srch(x,y):
        """Line search, <- argmax {alpha \in Real >= 0 : alpha x + y >= 0} where y >
        alpha = float("inf")
        for i in xrange(0,len(x)):
            if x[i] < 0:
                alpha0 = -y[i]/x[i]
                if alpha0 < alpha:
                    alpha=alpha0
        return alpha

    @staticmethod
    def symm(x):
        """Symmetrization, x <- symm(x) such that L(symm(x)) is a symmetric operator
        pass
```

**Language** MATLAB/Octave

**Code**

```
% Convert a vector to structure
function y = tostruct(x)
    y = struct('data',x);
end

% Defines the vector space used for optimization.
function self = MyVS()

    % Memory allocation and size setting
    self.init = @(x) x;

    % <- x (Shallow. No memory allocation.)
    self.copy = @(x) x;
```

```
    % <- alpha * x
    self.scal = @(alpha,x) tostruct(alpha*x.data);

    % <- 0
    self.zero = @(x) tostruct(zeros(size(x.data)));

    % <- alpha * x + y
    self.axpy = @(alpha,x,y) tostruct(alpha * x.data + y.data);

    %<- <x,y>
    self.innr = @(x,y)x.data'*y.data;

    % <- random
    self.rand = @(x)tostruct(randn(size(x.data)));

    % Jordan product, z <- x o y.
    self.prod = @(x,y)tostruct(x.data .* y.data);

    % Identity element, x <- e such that x o e = x.
    self.id = @(x)tostruct(ones(size(x.data)));

    % Jordan product inverse, z <- inv(L(x)) y where L(x) y = x o y.
    self.linv = @(x,y)tostruct(y.data ./ x.data);

    % Barrier function, barr <- barr(x) where x o grad barr(x) = e.
    self.barr = @(x)sum(log(x.data));

    % Line search, srch <- argmax {alpha \in Real >= 0 : alpha x + y >= 0}
    % where y > 0.
    self.srch = @(x,y) feval(@(z)min([min(z(find(z>0)));inf]),-y.data ./x.data);

    % Symmetrization, x <- symm(x) such that L(symm(x)) is a symmetric
    % operator.
    self.symm = @(x)x;
end
```

## 6.4 Symmetric cone programming

In the case of C++ and MATLAB/Octave, we provide a built-in vector space for semidefinite, second-order cone, and linear (SQL) programs:

| | |
|---|---|
| **Language** | C++ |
| **Vector** | `Optizelle::SQL::Vector` |
| **Operations** | `Optizelle::SQL` |

| | |
|---|---|
| **Language** | MATLAB/Octave |
| **Vector** | `Optizelle.SQL.create` (produces a structure array) |
| **Operations** | `Optizelle.SQL` |

In order to create a C++ `SQL::Vector`, we use the following constructor

```
namespace Optizelle {
    template <typename Real>
    struct SQL {
        struct Vector {
            // We require a vector of cone types and their sizes.
            Vector (
                std::vector <Cone::t> const & types_,
                std::vector <Natural> const & sizes_
            )
        };
    };
}
```

Here, `Cone::t` corresponds to the enumerated type `Cone` and `Natural` refers to the architecture specific unsigned integer defined in `Optizelle::Natural`. The constructor creates an SQL variable with the specified types and sizes of cones. Specifically, a linear cone of size $m$ denotes a vector in $\mathbb{R}^m$ that lies in the nonnegative orthant. A quadratic cone of size $m$ denotes a vector in $\mathbb{R}^m$ that lies in the quadratic cone. Finally, a semidefinite cone of size $m$ denotes a matrix in $\mathbb{R}^{m\times m}$ that lies in the cone of positive semidefinite matrices. Note, even though we ultimately find a symmetric matrix, we compute with a full $m \times m$ matrix and not just the upper or lower half. Using a full matrix affects how we define the derivatives of our inequality constraint $h$, so take care. Specifically, $h'(x)$ and $h'(x)^*$ need to assume that their arguments are not symmetric, so consider both upper and lower triangular parts of the matrices. In order to create a MATLAB/Octave `SQL` vector, we use the function

```
z = Optizelle.SQL.create(types,sizes);
```

where `types` is a vector containing elements from the enumerated type `Cone` and `sizes` is a vector denoting the size of the cones. For example, in order to define a `SQL` vector with a semidefinite, quadratic, and linear cone with sizes 2, 2, and 1, we use the syntax

```
types = ...
    [Optizelle.Cone.Semidefinite, ...
     Optizelle.Cone.Quadratic, ...
     Optizelle.Cone.Linear];
sizes = [2,2,1];
```

Otherwise, we define the meaning of each of these cones to be the same as the C++ case above. In order to access the elements of a C++ SQL vector, `x`, we use the following indexing functions

| Number of cones | Type of cone | Type of Indexing | Use |
|---|---|---|---|
| Single | Quadratic/Linear | Specific element | `x(i)` |
| Multiple | Quadratic/Linear | Specific element | `x(k,i)` |
| Multiple | Semidefinite | Specific element | `x(k,i,j)` |
| Multiple | Semidefinite/Quadratic/Linear | First element | `x.front(k)` |
| Multiple | Quadratic | First element | `x.naught(k)` |
| Multiple | Quadratic | Second element | `x.bar(k)` |

Finally, we have a couple of query functions

| Purpose | Use |
|---|---|
| Size of block | `x.blkSize(k)` |
| Type of block | `x.blkType(k)` |
| Number of blocks | `x.numblocks()` |

In order to access the elements of a MATLAB/Octave SQL vector, `x`, we note that the cones are stored in the cell array `x.data` where each element in the cell array denotes a different cone. We store quadratic and linear elements as column vectors and semidefinite elements as matrices. For example, to access the $i$th element of the $k$th block when this block is quadratic or linear, we use the syntax `x.data{k}(i)`. To access the $(i, j)$th element of the $k$th block when the block is semidefinite, we use the syntax `x.data{k}(i,j)`. As an example, we setup and solve a simple second-order cone program in our simple quadratic cone example:

**Language** C++

**Code**

```
// Optimize a simple problem with an optimal solution of (2.5,2.5)

#include <iostream>
#include <iomanip>
#include "optizelle/optizelle.h"
#include "optizelle/vspaces.h"
#include "optizelle/json.h"

// Create some type shortcuts
using Optizelle::Rm;
using Optizelle::SQL;
typedef double Real;

// Squares its input
template <typename Real>
Real sq(Real x){
    return x*x;
}

// Define a simple objective where
//
// f(x,y)=(x-3)^2+(y-2)^2
//
struct MyObj : public Optizelle::ScalarValuedFunction <Real,Rm> {
    typedef Rm <Real> X;

    // Evaluation
    double eval(X::Vector const & x) const {
        return sq(x[0]-Real(3.))+sq(x[1]-Real(2.));
    }

    // Gradient
    void grad(
        X::Vector const & x,
        X::Vector & grad
    ) const {
        grad[0]=2*x[0]-6;
        grad[1]=2*x[1]-4;
    }

    // Hessian-vector product
    void hessvec(
        X::Vector const & x,
        X::Vector const & dx,
        X::Vector & H_dx
    ) const {
        H_dx[0]= Real(2.)*dx[0];
        H_dx[1]= Real(2.)*dx[1];
    }
};

// Define a simple SOCP inequality
//
// h(x,y) = [ y >= |x| ]
// h(x,y) = (y,x) >=_Q 0
```

```
//
struct MyIneq : public Optizelle::VectorValuedFunction <Real,Rm,SQL> {
    typedef Rm <Real> X;
    typedef SQL <Real> Z;

    // z=h(x)
    void eval(
        X::Vector const & x,
        Z::Vector & z
    ) const {
        z(1,1)=x[1];
        z(1,2)=x[0];
    }

    // z=h'(x)dx
    void p(
        X::Vector const & x,
        X::Vector const & dx,
        Z::Vector & z
    ) const {
        z(1,1) = dx[1];
        z(1,2) = dx[0];
    }

    // xhat=h'(x)*dz
    void ps(
        X::Vector const & x,
        Z::Vector const & dz,
        X::Vector & xhat
    ) const {
        xhat[0] = dz(1,2);
        xhat[1] = dz(1,1);
    }

    // xhat=(h''(x)dx)*dz
    void pps(
        X::Vector const & x,
        X::Vector const & dx,
        Y::Vector const & dz,
        X::Vector & xhat
    ) const {
        X::zero(xhat);
    }
};

int main(int argc,char* argv[]){
    // Create some type shortcuts
    typedef Rm <Real>::Vector Rm_Vector;
    typedef SQL <Real>::Vector SQL_Vector;

    // Read in the name for the input file
    if(argc!=2) {
        std::cerr << "simple_quadratic_cone <parameters>" << std::endl;
        exit(EXIT_FAILURE);
    }
    auto fname = argv[1];
```

```
    // Generate an initial guess for the primal
    auto x = Rm_Vector({1.2,3.1});

    // Allocate memory for the dual
    auto z = SQL_Vector ({Optizelle::Cone::Quadratic},{2});

    // Create an optimization state
    Optizelle::InequalityConstrained <Real,Rm,SQL>::State::t state(x,z);

    // Read the parameters from file
    Optizelle::json::InequalityConstrained <Real,Rm,SQL>::read(fname,state);

    // Create a bundle of functions
    Optizelle::InequalityConstrained <Real,Rm,SQL>::Functions::t fns;
    fns.f.reset(new MyObj);
    fns.h.reset(new MyIneq);

    // Solve the optimization problem
    Optizelle::InequalityConstrained <Real,Rm,SQL>
        ::Algorithms::getMin(Optizelle::Messaging::stdout,fns,state);

    // Print out the reason for convergence
    std::cout << "The algorithm converged due to: " <<
        Optizelle::OptimizationStop::to_string(state.opt_stop) << std::endl;

    // Print out the final answer
    std::cout << std::setprecision(16) << std::scientific
        << "The optimal point is: (" << state.x[0] << ','
    << state.x[1] << ')' << std::endl;

    // Write out the final answer to file
    Optizelle::json::InequalityConstrained <Real,Rm,SQL>
        ::write_restart("solution.json",state);

    // Successful termination
    return EXIT_SUCCESS;
}
```

**Language** MATLAB/Octave

**Code**

```
% Optimize a simple problem with an optimal solution of (2.5,2.5)
function simple_quadratic_cone(fname)
    % Read in the name for the input file
    if nargin ~=1
        error('simple_quadratic_cone <parameters>');
    end

    % Execute the optimization
    main(fname);
end

% Squares its input
function z = sq(x)
    z=x*x;
end
```

```
% Define a simple objective where
%
% f(x,y)=(x-3)^2+(y-2)^2
%
function self = MyObj()

    % Evaluation
    self.eval = @(x) sq(x(1)-3.)+sq(x(2)-2.);

    % Gradient
    self.grad = @(x) [
        2.*x(1)-6;
        2.*x(2)-4];

    % Hessian-vector product
    self.hessvec = @(x,dx) [
        2.*dx(1);
        2.*dx(2)];
end

% Define a simple SOCP inequality
%
% h(x,y) = [ y >= |x| ]
% h(x,y) = (y,x) >=_Q 0
%
function self = MyIneq()

    % y=h(x)
    self.eval = @(x)MyIneq_eval(x);

    % z=h'(x)dx
    self.p = @(x,dx)MyIneq_p(x,dx);

    % xhat=h'(x)*dz
    self.ps = @(x,dz) [
        dz.data{1}(2);
        dz.data{1}(1)];

    % xhat=(h''(x)dx)*dz
    self.pps = @(x,dx,dz) [
        0;
        0];
end

% z=h(x)
function z=MyIneq_eval(x)
    global Optizelle;
    z = Optizelle.SQL.create([Optizelle.Cone.Quadratic],[2]);
    z.data{1} = [
        x(2);
        x(1)];
end

% z=h'(x)dx
function z=MyIneq_p(x,dx)
    global Optizelle;
    z = Optizelle.SQL.create([Optizelle.Cone.Quadratic],[2]);
```

```
    z.data{1} = [
        dx(2);
        dx(1)];
end

% Actually runs the program
function main(fname)

    % Grab the Optizelle library
    global Optizelle;
    setupOptizelle();

    % Generate an initial guess for the primal
    x = [1.2; 3.1];

    % Generate an initial guess for the dual
    z = Optizelle.SQL.create([Optizelle.Cone.Quadratic],[2]);

    % Create an optimization state
    state=Optizelle.InequalityConstrained.State.t( ...
        Optizelle.Rm,Optizelle.SQL,x,z);

    % Read the parameters from file
    state=Optizelle.json.InequalityConstrained.read( ...
        Optizelle.Rm,Optizelle.SQL,fname,state);

    % Create a bundle of functions
    fns=Optizelle.InequalityConstrained.Functions.t;
    fns.f=MyObj();
    fns.h=MyIneq();

    % Solve the optimization problem
    state=Optizelle.InequalityConstrained.Algorithms.getMin( ...
        Optizelle.Rm,Optizelle.SQL,Optizelle.Messaging.stdout,fns,state);

    % Print out the reason for convergence
    fprintf('The algorithm converged due to: %s\n', ...
        Optizelle.OptimizationStop.to_string(state.opt_stop));

    % Print out the final answer
    fprintf('The optimal point is: (%e,%e)\n',state.x(1),state.x(2));

    % Write out the final answer to file
    Optizelle.json.InequalityConstrained.write_restart( ...
        Optizelle.Rm,Optizelle.SQL,'solution.json',state);
end
```

Similarly, we setup and solve a simple semidefinite program in our simple SDP cone example:

**Language** C++

**Code**

```
// Optimize a simple problem with an optimal solution of (0.5,.25)

#include <iostream>
#include <iomanip>
#include "optizelle/optizelle.h"
#include "optizelle/vspaces.h"
#include "optizelle/json.h"
```

```
// Create some type shortcuts
using Optizelle::Rm;
using Optizelle::SQL;
typedef double Real;

// Define a simple objective where
//
// f(x,y)=-x+y
//
struct MyObj : public Optizelle::ScalarValuedFunction <Real,Rm> {
    typedef Rm <Real> X;

    // Evaluation
    double eval(X::Vector const & x) const {
        return -x[0]+x[1];
    }

    // Gradient
    void grad(
        X::Vector const & x,
        X::Vector & grad
    ) const {
        grad[0]=Real(-1.);
        grad[1]=Real(1.);
    }

    // Hessian-vector product
    void hessvec(
        X::Vector const & x,
        X::Vector const & dx,
        X::Vector & H_dx
    ) const {
        H_dx[0]= Real(0.);
        H_dx[1]= Real(0.);
    }
};

// Define a simple SDP inequality
//
// h(x,y) = [ y x ] >= 0
//          [ x 1 ]
//
struct MyIneq : public Optizelle::VectorValuedFunction <Real,Rm,SQL> {
    typedef Rm <Real> X;
    typedef SQL <Real> Z;

    // z=h(x)
    void eval(
        X::Vector const & x,
        Z::Vector & z
    ) const {
        z(1,1,1)=x[1];
        z(1,1,2)=x[0];
        z(1,2,1)=x[0];
        z(1,2,2)=Real(1.);
    }
```

```
    // z=h'(x)dx
    void p(
        X::Vector const & x,
        X::Vector const & dx,
        Z::Vector & z
    ) const {
        z(1,1,1)=dx[1];
        z(1,1,2)=dx[0];
        z(1,2,1)=dx[0];
        z(1,2,2)=Real(0.);
    }

    // xhat=h'(x)*dz
    void ps(
        X::Vector const & x,
        Z::Vector const & dz,
        X::Vector & xhat
    ) const {
        xhat[0]= dz(1,1,2)+dz(1,2,1);
        xhat[1]= dz(1,1,1);
    }

    // xhat=(h''(x)dx)*dz
    void pps(
        X::Vector const & x,
        X::Vector const & dx,
        Z::Vector const & dz,
        X::Vector & xhat
    ) const {
        X::zero(xhat);
    }
};

int main(int argc,char* argv[]){
    // Create some type shortcuts
    typedef Rm <Real>::Vector Rm_Vector;
    typedef SQL <Real>::Vector SQL_Vector;

    // Read in the name for the input file
    if(argc!=2) {
        std::cerr << "simple_sdp_cone <parameters>" << std::endl;
        exit(EXIT_FAILURE);
    }
    auto fname = argv[1];

    // Generate an initial guess for the primal
    auto x = Rm_Vector({1.2,3.1});

    // Allocate memory for the dual
    auto z = SQL_Vector ({Optizelle::Cone::Semidefinite},{2});

    // Create an optimization state
    Optizelle::InequalityConstrained <Real,Rm,SQL>::State::t state(x,z);

    // Read the parameters from file
    Optizelle::json::InequalityConstrained <Real,Rm,SQL>::read(fname,state);
```

```
    // Create a bundle of functions
    Optizelle::InequalityConstrained <Real,Rm,SQL>::Functions::t fns;
    fns.f.reset(new MyObj);
    fns.h.reset(new MyIneq);

    // Solve the optimization problem
    Optizelle::InequalityConstrained <Real,Rm,SQL>
        ::Algorithms::getMin(Optizelle::Messaging::stdout,fns,state);

    // Print out the reason for convergence
    std::cout << "The algorithm converged due to: " <<
        Optizelle::OptimizationStop::to_string(state.opt_stop) << std::endl;

    // Print out the final answer
    std::cout << std::setprecision(16) << std::scientific
        << "The optimal point is: (" << state.x[0] << ','
        << state.x[1] << ')' << std::endl;

    // Write out the final answer to file
    Optizelle::json::InequalityConstrained <Real,Rm,SQL>
        ::write_restart("solution.json",state);

    // Successful termination
    return EXIT_SUCCESS;
}
```

**Language** MATLAB/Octave

**Code**

```
% Optimize a simple problem with an optimal solution of (0.5,.25)

function simple_sdp_cone(fname)
    % Read in the name for the input file
    if nargin ~=1
        error('simple_sdp_cone <parameters>');
    end

    % Execute the optimization
    main(fname);
end

% Define a simple objective where
%
% f(x,y)=-x+y
%
function self = MyObj()

    % Evaluation
    self.eval = @(x) -x(1)+x(2);

    % Gradient
    self.grad = @(x) [
        -1.;
        1.];

    % Hessian-vector product
```

```
    self.hessvec = @(x,dx) [
        0;
        0];
end

% Define a simple SDP inequality
%
% h(x,y) = [ y x ] >= 0
%          [ x 1 ]
%
function self = MyIneq()

    % z=h(x)
    self.eval = @(x)MyIneq_eval(x);

    % z=h'(x)dx
    self.p = @(x,dx)MyIneq_p(x,dx);

    % xhat=h'(x)*dz
    self.ps = @(x,dz) [
        dz.data{1}(2,1)+dz.data{1}(1,2);
        dz.data{1}(1,1)];

    % xhat=(h''(x)dx)*dz
    self.pps = @(x,dx,dz) [
        0;
        0];
end

% z=h(x)
function z=MyIneq_eval(x)
    global Optizelle;
    z = Optizelle.SQL.create([Optizelle.Cone.Semidefinite],[2]);
    z.data{1} = [
        x(2) x(1);
        x(1) 1. ];
end

% z=h'(x)dx
function z=MyIneq_p(x,dx)
    global Optizelle;
    z = Optizelle.SQL.create([Optizelle.Cone.Semidefinite],[2]);
    z.data{1} = [
        dx(2) dx(1);
        dx(1) 0.  ];
end

% Actually runs the program
function main(fname)

    % Grab the Optizelle library
    global Optizelle;
    setupOptizelle();

    % Generate an initial guess for the primal
    x = [1.2; 3.1];
```

```
    % Generate an initial guess for the dual
    z = Optizelle.SQL.create([Optizelle.Cone.Semidefinite],[2]);

    % Create an optimization state
    state=Optizelle.InequalityConstrained.State.t( ...
        Optizelle.Rm,Optizelle.SQL,x,z);

    % Read the parameters from file
    state=Optizelle.json.InequalityConstrained.read( ...
        Optizelle.Rm,Optizelle.SQL,fname,state);

    % Create a bundle of functions
    fns=Optizelle.InequalityConstrained.Functions.t;
    fns.f=MyObj();
    fns.h=MyIneq();

    % Solve the optimization problem
    state=Optizelle.InequalityConstrained.Algorithms.getMin( ...
        Optizelle.Rm,Optizelle.SQL,Optizelle.Messaging.stdout,fns,state);

    % Print out the reason for convergence
    fprintf('The algorithm converged due to: %s\n', ...
        Optizelle.OptimizationStop.to_string(state.opt_stop));

    % Print out the final answer
    fprintf('The optimal point is: (%e,%e)\n',state.x(1),state.x(2));

    % Write out the final answer to file
    Optizelle.json.InequalityConstrained.write_restart( ...
        Optizelle.Rm,Optizelle.SQL,'solution.json',state);
end
```

## 6.5 State manipulation

State manipulation is a process that allows us to insert arbitrary code into the optimization algorithms. We use this to add new features such as the following:

- Real-time optimal control systems require hard computational time limit. After this time, we must exit the optimization cleanly and return our most current solution.
- For a particular application, we may want to use a custom line-search, but not recode the rest of the optimization algorithms.
- In signal processing, we may know our optimal solution does not have any frequencies above a certain threshold. When this is difficult to formulate as a constraint, we can simply run a high-pass filter on the optimization variable at the end of each iteration.
- When our algorithms perform poorly, we may want to run some custom diagnostics at the end of each optimization iteration.
- In order to replicate our optimization runs, we need to write a restart file at the end of each optimization iteration. We describe this process in the section Restarts.
- Internally, we use state manipulation to add algorithms such as the interior point method to the composite-step SQP method.

In each of these situations, we make use of the `StateManipulator`. In order to manipulate the state, we use an object called the `StateManipulator`. During the optimization computation, we repeatedly call this object with the bundle of functions, optimization state, and the location. At this point, we may do any computation and modify the state as desired. In C++ and Python, we implicitly return these changes to the state. In MATLAB/Octave, we must return the state explicitly. In code, we specify the StateManipulator as:

**Language** C++

**Structure** `Optizelle::StateManipulator`

**Interface** Inheritance

**Code**

```
namespace Optizelle{
    // A function that has free reign to manipulate or analyze the state.
    template <typename ProblemClass>
    struct StateManipulator {
        // Disallow constructors
        NO_COPY_ASSIGNMENT(StateManipulator)

        // Give an empty default constructor
        StateManipulator() {}

        // Application
        virtual void eval(
            typename ProblemClass::Functions::t const & fns,
            typename ProblemClass::State::t & state,
            OptimizationLocation::t const & loc
        ) const = 0;

        // Allow the derived class to deallocate memory
        virtual ~StateManipulator() {}
    };
}
```

**Language** Python

**Structure** `Optizelle.StateManipulator`

**Interface** Inheritance

**Code**

```
class StateManipulator(object):
    """A function that has free reign to manipulate or analyze the state"""
    def eval(self,fns,state,loc):
        """Application"""
        pass
```

**Language** MATLAB/Octave

**Structure** `Optizelle.StateManipulator`

**Interface** Members present

**Code**

```
% A function that has free reign to manipulate or analyze the state.
Optizelle.StateManipulator = struct('eval',@(fns,state,loc)state);
```

Once we define the StateManipulator, we call the optimization solver with one of the following four commands, which differs slightly from those defined in the section Call the optimization solver. In essence, we add the StateManipulator as the last argument to `getMin`:

**Language** C++

**Code**

```
Optizelle::Unconstrained<Real,XX>::Algorithms::getMin(
    msg,fns,state,smanip);

Optizelle::EqualityConstrained<Real,XX,YY>::Algorithms::getMin(
    msg,fns,state,smanip);

Optizelle::InequalityConstrained<Real,XX,ZZ>::Algorithms::getMin(
    msg,fns,state,smanip);

Optizelle::Constrained<Real,XX,YY,ZZ>::Algorithms::getMin(
    msg,fns,state,smanip);
```

**Language** Python

**Code**

```
Optizelle.Unconstrained.Algorithms.getMin(XX,msg,fns,state,smanip)

Optizelle.EqualityConstrained.Algorithms.getMin(XX,YY,msg,fns,state,smanip)

Optizelle.InequalityConstrained.Algorithms.getMin(XX,ZZ,msg,fns,state,smanip)

Optizelle.Constrained.Algorithms.getMin(XX,YY,ZZ,msg,fns,state,smanip)
```

**Language** MATLAB/Octave

**Code**

```
state = Optizelle.Unconstrained.Algorithms.getMin( ...
    XX,msg,fns,state,smanip);

state = Optizelle.EqualityConstrained.Algorithms.getMin( ...
    XX,YY,msg,fns,state,smanip);

state = Optizelle.InequalityConstrained.Algorithms.getMin( ...
    XX,ZZ,msg,fns,state,smanip);

state = Optizelle.Constrained.Algorithms.getMin( ...
    XX,YY,ZZ,msg,fns,state,smanip);
```

As an example, we use the `StateManipulator` to add restarts to our Rosenbrock advanced API example. We discuss restarts in the section entitled Restarts.

**Language** C++

**Code**

```
// Define a state manipulator that writes out the optimization state at
// each iteration.
struct MyRestartManipulator
    : Optizelle::StateManipulator <Optizelle::Unconstrained <double,MyVS> >
{
    void eval(
        typename Optizelle::Unconstrained <double,MyVS>
            ::Functions::t const & fns,
        typename Optizelle::Unconstrained <double,MyVS>
            ::State::t & state,
        Optizelle::OptimizationLocation::t const & loc
    ) const {
        switch(loc) {
        // At the end of the optimization iteration, write the restart file
        case Optizelle::OptimizationLocation::EndOfOptimizationIteration: {
            // Create a reasonable file name
```

```
                std::stringstream ss;
                ss << "rosenbrock_advanced_api_";
                ss << std::setw(4) << std::setfill('0') << state.iter;
                ss << ".json";

                // Write the restart file
                Optizelle::json::Unconstrained <double,MyVS>::write_restart(
                    ss.str(),state);
                break;
            } default:
                break;
            }
        }
};
```

**Language** Python

**Code**

```python
# Define a state manipulator that writes out the optimization state at
# each iteration.
class MyRestartManipulator(Optizelle.StateManipulator):
    def eval(self,fns,state,loc):
        # At the end of the optimization iteration, write the restart file
        if loc == Optizelle.OptimizationLocation.EndOfOptimizationIteration:
            # Create a reasonable file name
            ss = "rosenbrock_advanced_api_%04d.json" % (state.iter)

            # Write the restart file
            Optizelle.json.Unconstrained.write_restart(MyVS,ss,state)
```

**Language** MATLAB/Octave

**Code**

```matlab
% Define a state manipulator that writes out the optimization state at
% each iteration.
function smanip=MyRestartManipulator()
    smanip=struct('eval',@(fns,state,loc)MyRestartManipulator_(fns,state,loc));
end
function state=MyRestartManipulator_(fns,state,loc)
    global Optizelle;

    % At the end of the optimization iteration, write the restart file
    if(loc == Optizelle.OptimizationLocation.EndOfOptimizationIteration)
        % Create a reasonable file name
        ss = sprintf('rosenbrock_advanced_api_%04d.json',state.iter);

        % Write the restart file
        Optizelle.json.Unconstrained.write_restart(MyVS(),ss,state);
    end
end
```

In order to use this StateManipulator, we call Optizelle's solver with the code:

**Language** C++

**Code**

```
// Solve the optimization problem
Optizelle::Unconstrained <double,MyVS>::Algorithms
    ::getMin(mymessaging,fns,state,MyRestartManipulator());
```

**Language** Python

**Code**

```
# Solve the optimization problem
Optizelle.Unconstrained.Algorithms.getMin(
    MyVS,mymessaging,fns,state,MyRestartManipulator())
```

**Language** MATLAB/Octave

**Code**

```
% Solve the optimization problem
state=Optizelle.Unconstrained.Algorithms.getMin( ...
    MyVS(),@MyMessaging,fns,state,MyRestartManipulator());
```

## 6.6 Restarts

Restarts are a mechanism to read, write, and archive the progress and solution of an optimization algorithm. In other words, restarts allow us to save the state of an optimization algorithm before it finishes computing. We do this for several reasons:

- In scientific or engineering tasks, we may need to replicate or reproduce our work.
- Large, computationally expensive problems typically require parallel computing clusters. With thousands of computers working in concert, the chance that a hardware failure occurs increases. One way to recover from these failures it to restart the computation after a crash.
- Parallel computing clusters generally share their computing resources between several users. In order to fairly divide use, batch jobs require us to specify the amount of time required to run a job. If we guess this number poorly, restarts allow us to complete the computation later.
- For many problems, it's unclear what algorithm we should use. Second-order methods such as Newton's method are only guaranteed to converge quadratically near the solution. As such, we may be well served to start the computation with a first-order method and then switch to a second-order method as we approach optimality. We can accomplish this by writing a restart file, modifying the specified algorithm, and then resuming the computation.
- Often an algorithm makes progress toward a solution, but then stagnates. In order to diagnose why the algorithm stagnated, we may examine the restart file at the iteration of stagnation. Furthermore, if we have insight into the underlying problem structure, we could modify the solution by hand or with an outside tool and then restart the computation.

Each of these situations requires restarts. As long as we use our built-in vector spaces such as `Rm` and `SQL`, we can easily read and write the state to a JSON formatted file with the commands:

**Language** C++

**Code**

```
Optizelle::json::Unconstrained <Real,XX>::write_restart(
    fname,state);
Optizelle::json::Unconstrained <Real,XX>::read_restart(
    fname,x,state);

Optizelle::json::EqualityConstrained <Real,XX,YY>::write_restart(
    fname,state);
Optizelle::json::EqualityConstrained <Real,XX,YY>::read_restart(
    fname,x,y,state);

Optizelle::json::InequalityConstrained <Real,XX,ZZ>::write_restart(
    fname,state);
Optizelle::json::InequalityConstrained <Real,XX,ZZ>::read_restart(
    fname,x,z,state);

Optizelle::json::Constrained <Real,XX,YY,ZZ>::write_restart(
    fname,state);
Optizelle::json::Constrained <Real,XX,YY,ZZ>::read_restart(
    fname,x,y,z,state);
```

**Language** Python

**Code**

```
Optizelle.json.Unconstrained.write_restart(XX,fname,state);
Optizelle.json.Unconstrained.read_restart(XX,fname,x,state);

Optizelle.json.EqualityConstrained.write_restart(XX,YY,fname,state);
Optizelle.json.EqualityConstrained.read_restart(XX,YY,fname,x,y,state);

Optizelle.json.InequalityConstrained.write_restart(XX,ZZ,fname,state);
Optizelle.json.InequalityConstrained.read_restart(XX,ZZ,fname,x,z,state);

Optizelle.json.Constrained.write_restart(XX,YY,ZZ,fname,state);
Optizelle.json.Constrained.read_restart(XX,YY,ZZ,fname,x,y,z,state);
```

**Language** MATLAB/Octave

**Code**

```
Optizelle.json.Unconstrained.write_restart(XX,fname,state);
state = Optizelle.json.Unconstrained.read_restart(XX,fname,x);

Optizelle.json.EqualityConstrained.write_restart(XX,YY,fname,state);
state = Optizelle.json.EqualityConstrained.read_restart(XX,YY,fname,x,y);

Optizelle.json.InequalityConstrained.write_restart(XX,ZZ,fname,state);
state = Optizelle.json.InequalityConstrained.read_restart(XX,ZZ,fname,x,z);

Optizelle.json.Constrained.write_restart(XX,YY,ZZ,fname,state);
state = Optizelle.json.Constrained.read_restart(XX,YY,ZZ,fname,x,y,z);
```

As was the case before, `XX`, `YY`, and ZZ correspond to the vector spaces $X$, $Y$, and $Z$ described in the section Import or define the appropriate vector spaces. Likely, they are just `Rm` or `SQL`. Next, we call the function with a `Messaging` object, `msg`. Third, the string `fname` denotes the file name that we read or write the restart. Next, the variable `state` denotes a `State` object. During a write, we write the provided state to file. During a read, we read the restart file into the specified state. Finally, the variables `x`, `y`, and `z` denote variables in the spaces `XX`, `YY`, and `ZZ`, respectively. We only use them to initialize memory, so any valid vector works. As an example, we return to our Rosenbrock advanced API example. We already showed how to write a restart file at the end of each optimization iteration in our discussion of `StateManipulator`s. Specifically, we used the write_restart command in our StateManipulator example. To compliment that code, we read an optional restart file prior to optimization with the code:

**Language** C++

**Code**

```
    // If we have a restart file, read in the parameters
    if(argc==3)
        Optizelle::json::Unconstrained <double,MyVS>::read_restart(
            rname,x,state);

    // Read additional parameters from file
    Optizelle::json::Unconstrained <double,MyVS>::read(pname,state);
```

**Language** Python

**Code**

```
# If we have a restart file, read in the parameters
if len(sys.argv)==3:
    Optizelle.json.Unconstrained.read_restart(MyVS,rname,x,state)

# Read additional parameters from file
Optizelle.json.Unconstrained.read(MyVS,pname,state)
```

**Language** MATLAB/Octave

**Code**

```
    % If we have a restart file, read in the parameters
    if(nargin==2)
        state = Optizelle.json.Unconstrained.read_restart(MyVS(),rname,x);
    end

    % Read additional parameters from file
    state=Optizelle.json.Unconstrained.read(MyVS(),pname,state);
```

As a note, we call the JSON reader after we read the restart file. If we do this in the reverse order, the restart read process overwrites all of our parameters. For Rm and SQL, the above process works seamlessly. In fact, C++, Python, and MATLAB/Octave all use the same format for Rm, which means we can write a restart file in one language and then read the same restart file in a different language. However, for customized vector spaces, we must provide Optizelle information on how to translate a vector to a JSON formatted file using the following commands:

**Language** C++

**Code**

```
namespace Optizelle {
    namespace json {
        template <>
        struct Serialization <Real,WW> {
            static std::string serialize(
                typename WW <Real>::Vector const & x,
                std::string const & name,
                Natural const & iter
            ) { throw; }
            static typename WW <Real>::Vector deserialize(
                typename WW <Real>::Vector const & x,
                std::string const & x_json
            ) { throw; }
        };
    }
}
```

**Language** Python

**Code**
```
Optizelle.json.Serialization.serialize.register(serialize,vector_type)
Optizelle.json.Serialization.deserialize.register(deserialize,vector_type)
```

**Language** MATLAB/Octave

**Code**
```
Optizelle.json.Serialization.serialize('register',serialize,check);
Optizelle.json.Serialization.deserialize('register',deserialize,check);
```

In each command, the `serialize` and `deserialize` functions work in a similar manner. The `serialize` function accepts a vector, the vector's name, and the current iteration. Then, `serialize` returns a valid JSON structure corresponding to this vector. For `Rm`, we use simple JSON vector notation such as `[1.2, 2.3, 3.4]`, but this can be significantly more complicated. In fact, for large-scale optimization problems, we suggest storing the vector in a separate binary file and returning a JSON structure that denotes the name of the file. In order to make process of defining these file names easier, we provide access to the variable name and iteration number as the second and third arguments, respectively. Next, the `deserialize` function accepts two arguments and returns a vector. The first argument denotes a vector in the same vector space as the vector we need translated. The second argument denotes a JSON formatted string of the vector we need to translate. Generally, we use the first argument to initialize memory for the vector we eventually return. Then, we use the JSON formatted string to fill in the appropriate information. In C++, we accomplish this process through template specialization. In Python, we call the `serialize` and `deserialize` functions in the `Optizelle.json.Serialization` module with the `"registration"` string. Then, we provide our custom `serialize` and `deserialize` routines along with the type of the vector that we want to serialize in the variable `vector_type`. We obtain this information with the `type` command and require it in order to disambiguate multiple serialization routines. In MATLAB/Octave, we call the `serialize` and `deserialize` functions in the `Optizelle.json.Serialization` structure with the `'registration'` string. Then, similar to Python, we provide our custom `serialize` and `deserialize` routines along with a function `check`. The function `check` accepts a single argument and returns `1` when called with the kind of vector we want to serialize and `0` otherwise. We require the `check` function to disambiguate the different serialization functions, so we try to make it as specific as possible. As an example, we return to our Rosenbrock advanced API example. There, we define custom serialization routines with the code:

**Language** C++

**Code**
```
// Define serialization routines for MyVS
namespace Optizelle {
    namespace json {
        template <>
        struct Serialization <double,MyVS> {
            static std::string serialize(
                typename MyVS <double>::Vector const & x,
                std::string const & name,
                Natural const & iter
            ) {
                // Create a string with the format
                // [ x1, x2, ..., xm ].
                std::stringstream x_json;
                x_json.setf(std::ios::scientific);
                x_json.precision(16);
                x_json << "[ ";
                for(Natural i=0;i<x.size()-1;i++)
                    x_json << x[i] << ", ";
                x_json << x.back() << " ]";

                // Return the string
```

```
            return x_json.str();
        }
        static MyVS <double>::Vector deserialize(
            typename MyVS <double>::Vector const & x_,
            std::string const & x_json_
        ) {
            // Make a copy of x_json_
            auto x_json = x_json_;

            // Filter out the commas and brackets from the string
            char formatting[] = "[],";
            for(Natural i=0;i<3;i++)
                x_json.erase(
                    std::remove(x_json.begin(),x_json.end(),formatting[i]),
                    x_json.end());

            // Create a new vector that we eventually return
            auto x = std::vector <double>(x_.size());

            // Create a stream out of x_json
            std::stringstream ss(x_json);

            // Read in each of the elements
            for(auto i=0;i<x.size();i++)
                ss >> x[i];

            // Return the result
            return std::move(x);
        }
    };
}
}
```

**Language** Python

**Code**

```
def serialize_MyVS(x,name,iter):
    """Serializes an array for the vector space MyVS"""

    # Create the json representation
    x_json="[ "
    for i in xrange(0,len(x)):
        x_json += str(x[i]) + ", "
    x_json=x_json[0:-2]
    x_json +=" ]"

    return x_json

def deserialize_MyVS(x,x_json):
    """Deserializes an array for the vector space MyVS"""

    # Eliminate all whitespace
    x_json="".join(x_json.split())

    # Check if we're a vector
    if x_json[0:1]!="[" or x_json[-1:]!="]":
        raise TypeError("Attempted to deserialize a non-array vector.")
```

```python
    # Eliminate the initial and final delimiters
    x_json=x_json[1:-1]

    # Create a list of the numbers involved
    x_json=x_json.split(",")

    # Convert the strings to numbers
    x_json=map(lambda x:float(x),x_json)

    # Create a MyVS vector
    return array.array('d',x_json)

# Register the serialization routines for arrays
def MySerialization():
    Optizelle.json.Serialization.serialize.register(
        serialize_MyVS,array.array)
    Optizelle.json.Serialization.deserialize.register(
        deserialize_MyVS,array.array)
```

**Language** MATLAB/Octave

**Code**

```matlab
% Define serialization routines for MyVS
function MySerialization()
    global Optizelle;
    Optizelle.json.Serialization.serialize( ...
        'register', ...
        @(x,name,iter)strrep(mat2str(x.data'),' ',', '), ...
        @(x)isstruct(x) && isfield(x,'data') && isvector(x.data));
    Optizelle.json.Serialization.deserialize( ...
        'register', ...
        @(x,x_json)tostruct(str2num(x_json)'), ...
        @(x)isstruct(x) && isfield(x,'data') && isvector(x.data));
end
```

As another example, we refer to our Simple constrained advanced API example. This differs from the previous example since we write our vectors to a separate file. In order to accomplish this, we define custom serialization routines with the code:

**Language** C++

**Code**

```cpp
// Define serialization routines for MyVS
namespace Optizelle {
    namespace json {
        template <>
        struct Serialization <double,MyVS> {
            static std::string serialize(
                typename MyVS <double>::Vector const & x,
                std::string const & name,
                Natural const & iter
            ) {
                // Create the filename where we put our vector
                std::stringstream fname;
                fname << "./restart/";
                fname << name << ".";
                fname << std::setw(4) << std::setfill('0') << iter;
```

```
        fname << ".txt";

        // Actually write the vector there
        std::ofstream fout(fname.str());
        if(fout.fail()) {
            std::stringstream msg;
            msg << "While writing the variable " << name
                << " to file on iteration " << iter
                << ", unable to open the file: "
                << fname.str() << ".";
            throw Optizelle::Exception::t(msg.str());
        }
        fout.setf(std::ios::scientific);
        fout.precision(16);
        for(Natural i=0;i<x.size();i++)
            fout << x[i] << std::endl;

        // Close out the file
        fout.close();

        // Use this filename as the json string
        std::stringstream x_json;
        x_json << "\"" << fname.str() << "\"";
        return x_json.str();
    }
    static MyVS <double>::Vector deserialize(
        typename MyVS <double>::Vector const & x_,
        std::string const & x_json_
    ) {
        // Make a copy of x_json_
        auto x_json = x_json_;

        // Filter out the quotes and newlines from the string
        auto formatting = "\"\n";
        for(auto i=0;i<2;i++)
            x_json.erase(
                std::remove(x_json.begin(),x_json.end(),formatting[i]),
                x_json.end());

        // Open the file for reading
        std::ifstream fin(x_json.c_str());
        if(!fin.is_open())
            throw Optizelle::Exception::t(
                "Error while opening the file " + x_json + ": " +
                strerror(errno));

        // Create a new vector that we eventually return
        auto x = std::vector <double> (x_.size());

        // Read in each of the elements
        for(auto i=0;i<x.size();i++)
            fin >> x[i];

        // Return the result
        return std::move(x);
    }
};
```

```
    }
}
```

**Language** Python

**Code**

```
def serialize_MyVS(x,name,iter):
    """Serializes an array for the vector space MyVS"""

    # Create the filename where we put our vector
    fname = "./restart/%s.%04d.txt" % (name,iter)

    # Actually write the vector there
    fout = open(fname,"w");
    for i in xrange(0,len(x)):
        fout.write("%1.16e\n" % x[i])

    # Close out the file
    fout.close()

    # Use this filename as the json string
    x_json = "\"%s\"" % fname
    return x_json

def deserialize_MyVS(x_,x_json):
    """Deserializes an array for the vector space MyVS"""

    # Eliminate all whitespace
    x_json="".join(x_json.split())

    # Eliminate the initial and final delimiters
    x_json=x_json[1:-1]

    # Open the file for reading
    fin = open(x_json,"r")

    # Allocate a new vector to return
    x = copy.deepcopy(x_)

    # Read in each of the elements
    for i in xrange(0,len(x)):
        x[i] = float(fin.readline())

    # Close out the file
    fin.close()

    # Return the result
    return x

# Register the serialization routines for arrays
def MySerialization():
    Optizelle.json.Serialization.serialize.register(
        serialize_MyVS,array.array)
    Optizelle.json.Serialization.deserialize.register(
        deserialize_MyVS,array.array)
```

**Language** MATLAB/Octave

**Code**

```
% Define the serialize routine for MyVS
function x_json=serialize_MyVS(x,name,iter)
    % Create the filename where we put our vector
    fname=sprintf('./restart/%s.%04d.txt',name,iter);

    % Actually write the vector there
    dlmwrite(fname,x.data);

    % Use this filename as the json string
    x_json = sprintf('\"%s\"',fname);
end

% Define the deserialize routine for MyVS
function x=deserialize_MyVS(x_,x_json)
    % Filter out the quotes and newlines from the string
    x_json = strrep(x_json,'"','');
    x_json = strrep(x_json,sprintf('\n'),'');

    % Read the data into x
    x=tostruct(dlmread(x_json));
end

% Define serialization routines for MyVS
function MySerialization()
    global Optizelle;
    Optizelle.json.Serialization.serialize( ...
        'register', ...
        @(x,name,iter)serialize_MyVS(x,name,iter), ...
        @(x)isstruct(x) && isfield(x,'data') && isvector(x.data));
    Optizelle.json.Serialization.deserialize( ...
        'register', ...
        @(x,x_json)deserialize_MyVS(x,x_json), ...
        @(x)isstruct(x) && isfield(x,'data') && isvector(x.data));
end
```

In some situations, we want to avoid using JSON all together. Generally, this occurs when integrating Optizelle into an existing application with rigid I/O requirements. In this case, we provide an alternative mechanism to generate restarts.

At its core, restarts consist of two mechanisms: release and capture. Release transforms the state into a collection of lists that contain all of the optimization information. Capture reverses this process. Generally, we do a release, write these lists containing the state information to file, and then capture the state. The idea behind this process is that we don't expect ourselves to remember all of the optimization variables. Certainly, this collection of variables changes whenever we update the code or add new algorithms. However, if we know how to write a list of variables to file, we can simply iterate over the list and take the appropriate action. More specifically, the capture and release functions operate on lists of tuples. As far as the type used for the lists, we have:

**Language** C++

**Type** `std::list`

**Language** Python

**Type** `list`

**Language** MATLAB/Octave

**Type** `cell`

For the type used by the tuples, we have

**Language** C++

**Type** `std::pair`

**Language** Python

**Type** `tuple`

**Language** MATLAB/Octave

**Type** `cell`

In these tuples, we always use a string for the first element. This represents the unique label for the item. The second items depends on the type involved and we enumerate these possibilities below:

**Type** `Reals`

**Description** List of `Real` numbers and labels.

**Type** `Naturals`

**Description** List of `Natural` numbers and labels.

**Type** `Params`

**Description** List of strings and labels. These strings correspond to the various `Enumerated` types that have been converted to strings using the `to_string` function, which we also describe in the `Enumerated` type documentation.

**Type** `X_Vectors`

**Description** List of `X_Vector` vectors and labels.

**Type** `Y_Vectors`

**Description** List of `Y_Vector` vectors and labels.

**Type** `Z_Vectors`

**Description** List of `Z_Vector` vectors and labels.

Based on the above types, we release and capture the state with the following code:

**Language** C++

**Code**

```
Optizelle::Unconstrained <Real,XX>::Restart::X_Vectors xs;
Optizelle::Unconstrained <Real,XX>::Restart::Reals reals;
Optizelle::Unconstrained <Real,XX>::Restart::Naturals nats;
Optizelle::Unconstrained <Real,XX>::Restart::Params params;
Optizelle::Unconstrained <Real,XX>::Restart
    ::release(state,xs,reals,nats,params);
Optizelle::Unconstrained <Real,XX>::Restart
    ::capture(state,xs,reals,nats,params);
Optizelle::EqualityConstrained <Real,XX,YY>::Restart::X_Vectors xs;
Optizelle::EqualityConstrained <Real,XX,YY>::Restart::Y_Vectors ys;
Optizelle::EqualityConstrained <Real,XX,YY>::Restart::Reals reals;
Optizelle::EqualityConstrained <Real,XX,YY>::Restart::Naturals nats;
Optizelle::EqualityConstrained <Real,XX,YY>::Restart::Params params;
Optizelle::EqualityConstrained <Real,XX,YY>::Restart
    ::release(state,xs,ys,reals,nats,params);
Optizelle::EqualityConstrained <Real,XX,YY>::Restart
    ::capture(state,xs,ys,reals,nats,params);

Optizelle::InequalityConstrained <Real,XX,ZZ>::Restart::X_Vectors xs;
Optizelle::InequalityConstrained <Real,XX,ZZ>::Restart::Z_Vectors zs;
Optizelle::InequalityConstrained <Real,XX,ZZ>::Restart::Reals reals;
Optizelle::InequalityConstrained <Real,XX,ZZ>::Restart::Naturals nats;
Optizelle::InequalityConstrained <Real,XX,ZZ>::Restart::Params params;
Optizelle::InequalityConstrained <Real,XX,ZZ>::Restart
    ::release(state,xs,zs,reals,nats,params);
Optizelle::InequalityConstrained <Real,XX,ZZ>::Restart
    ::capture(state,xs,zs,reals,nats,params);

Optizelle::Constrained <Real,XX,YY,ZZ>::Restart::X_Vectors xs;
Optizelle::Constrained <Real,XX,YY,ZZ>::Restart::Y_Vectors ys;
Optizelle::Constrained <Real,XX,YY,ZZ>::Restart::Z_Vectors zs;
Optizelle::Constrained <Real,XX,YY,ZZ>::Restart::Reals reals;
Optizelle::Constrained <Real,XX,YY,ZZ>::Restart::Naturals nats;
Optizelle::Constrained <Real,XX,YY,ZZ>::Restart::Params params;
Optizelle::Constrained <Real,XX,YY,ZZ>::Restart
    ::release(state,xs,ys,zs,reals,nats,params);
Optizelle::Constrained <Real,XX,YY,ZZ>::Restart
    ::capture(state,xs,ys,zs,reals,nats,params);
```

**Language** Python

**Code**

```
xs = Optizelle.Unconstrained.Restart.X_Vectors()
reals = Optizelle.Unconstrained.Restart.Reals()
nats = Optizelle.Unconstrained.Restart.Naturals()
params = Optizelle.Unconstrained.Restart.Params()
Optizelle.Unconstrained.Restart.release(XX,state,xs,reals,nats,params)
Optizelle.Unconstrained.Restart.capture(XX,state,xs,reals,nats,params)
xs = Optizelle.EqualityConstrained.Restart.X_Vectors()
ys = Optizelle.EqualityConstrained.Restart.Y_Vectors()
reals = Optizelle.EqualityConstrained.Restart.Reals()
nats = Optizelle.EqualityConstrained.Restart.Naturals()
params = Optizelle.EqualityConstrained.Restart.Params()
Optizelle.EqualityConstrained.Restart.release(
    XX,YY,state,xs,ys,reals,nats,params)
Optizelle.EqualityConstrained.Restart.capture(
    XX,YY,state,xs,ys,reals,nats,params)
```

```
xs = Optizelle.InequalityConstrained.Restart.X_Vectors()
zs = Optizelle.InequalityConstrained.Restart.Z_Vectors()
reals = Optizelle.InequalityConstrained.Restart.Reals()
nats = Optizelle.InequalityConstrained.Restart.Naturals()
params = Optizelle.InequalityConstrained.Restart.Params()
Optizelle.InequalityConstrained.Restart.release(
    XX,ZZ,state,xs,zs,reals,nats,params)
Optizelle.InequalityConstrained.Restart.capture(
    XX,ZZ,state,xs,zs,reals,nats,params)

xs = Optizelle.Constrained.Restart.X_Vectors()
ys = Optizelle.Constrained.Restart.Y_Vectors()
zs = Optizelle.Constrained.Restart.Z_Vectors()
reals = Optizelle.Constrained.Restart.Reals()
nats = Optizelle.Constrained.Restart.Naturals()
params = Optizelle.Constrained.Restart.Params()
Optizelle.Constrained.Restart.release(
    XX,YY,ZZ,state,xs,ys,zs,reals,nats,params)
Optizelle.Constrained.Restart.capture(
    XX,YY,ZZ,state,xs,ys,zs,reals,nats,params)
```

**Language** MATLAB/Octave

**Code**

```
xs = Optizelle.Unconstrained.Restart.X_Vectors;
reals = Optizelle.Unconstrained.Restart.Reals;
nats = Optizelle.Unconstrained.Restart.Naturals;
params = Optizelle.Unconstrained.Restart.Params;
[xs reals nats params] = Optizelle.Unconstrained.Restart.release( ...
    XX,state);
state = Optizelle.Unconstrained.Restart.capture( ...
    XX,state,xs,reals,nats,params);
xs = Optizelle.EqualityConstrained.Restart.X_Vectors;
ys = Optizelle.EqualityConstrained.Restart.Y_Vectors;
reals = Optizelle.EqualityConstrained.Restart.Reals;
nats = Optizelle.EqualityConstrained.Restart.Naturals;
params = Optizelle.EqualityConstrained.Restart.Params;
[xs ys reals nats params] = Optizelle.EqualityConstrained.Restart.release( ...
    XX,YY,state);
state = Optizelle.EqualityConstrained.Restart.capture( ...
    XX,YY,state,xs,ys,reals,nats,params);

xs = Optizelle.InequalityConstrained.Restart.X_Vectors;
zs = Optizelle.InequalityConstrained.Restart.Z_Vectors;
reals = Optizelle.InequalityConstrained.Restart.Reals;
nats = Optizelle.InequalityConstrained.Restart.Naturals;
params = Optizelle.InequalityConstrained.Restart.Params;
[xs zs reals nats params] = Optizelle.InequalityConstrained.Restart.release( ...
    XX,ZZ,state);
state = Optizelle.InequalityConstrained.Restart.capture( ...
    XX,ZZ,state,xs,zs,reals,nats,params);
```

```
xs = Optizelle.Constrained.Restart.X_Vectors;
ys = Optizelle.Constrained.Restart.Y_Vectors;
zs = Optizelle.Constrained.Restart.Z_Vectors;
reals = Optizelle.Constrained.Restart.Reals;
nats = Optizelle.Constrained.Restart.Naturals;
params = Optizelle.Constrained.Restart.Params;
[xs ys zs reals nats params] = Optizelle.Constrained.Restart.release( ...
    XX,YY,ZZ,state);
state = Optizelle.Constrained.Restart.capture( ...
    XX,YY,ZZ,state,xs,ys,zs,reals,nats,params);
```

As with `read_restart` and `write_restart`, we most likely use this functions within a `StateManipulator`. However, when possible, we are likely better off just using the JSON formatted restart mechanisms within `read_restart` and `write_restart`.

## 6.7 Caching Computations

Internally, Optizelle caches many operations in order to reduce unnecessary computation. This includes computations such as the objective or gradient evaluations. Nevertheless, there are operations that should be cached that Optizelle does not control due to its matrix-free nature. These operations must be cached by the user's code. In the following section, we detail what these operations are and how they should be cached. The following table summarizes the different pieces of the code that can be cached, the number of items that should be stored, and the priority of caching this particular element.

**Computation** Objective evaluation during the first gradient solve

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**Priority** Low

**Number Stored** 1

**Description** During initialization, Optizelle evaluates the gradient before the objective function. Depending on the problem, it may be possible to evaluate and cache the objective function at the same time as this computation. Specifically, when the objective function has the form $J(x) = f(g(x))$, we calculate the gradient as

$$\nabla J(x) = g'(x)^* \nabla f(g(x)).$$

When the evaluation of $g(x)$ is expensive, such as solving a PDE or computing an inverse, we can use this calculation for both the gradient and the objective function by simultaneously computing both $f(g(x))$ and $\nabla f(g(x))$.

Despite this utility, we do not typically prioritize this optimization. We only benefit from saving this computation on the first iteration since Optizelle automatically caches the appropriate objective evaluations from the globalization, be that from line-search or trust-region algorithms, for the rest of the algorithm. Therefore, subsequent gradient evaluations don't need to cache information about the objective since it's already been cached. Nevertheless, when we repeatedly run the first iteration of an optimization problem in order to check the problem setup, this caching can save in the overall computation.

**Computation** Nested computations and state solves

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**Priority** High

**Number Stored** 1

**Description** During the discussion of caching the objective, we spoke of objective functions of the form $J(x) = f(g(x))$. As we noted before, we have that

$$\nabla J(x) = g'(x)^* \nabla f(g(x)),$$

but we also note that

$$\nabla^2 J(x)\partial x = (g''(x)\partial x)^* \nabla f(g(x)) + g'(x)^* \nabla^2 f(g(x)) g'(x)\partial x.$$

Here, we see that we repeatedly use the quantity $g(x)$. When the evaluation of $g(x)$ is expensive, such as solving a PDE or computing an inverse, then caching this element allows us to save significantly on the computational cost. When the evaluation of $g(x)$ corresponds to a PDE solve, we refer to its evaluation as a state solve.

**Computation** Hessian

**Problem Class** Unconstrained, Equality Constrained, Inequality Constrained, Constrained

**Priority** Low

**Number Stored** 1

**Description** Although Optizelle implements matrix-free algorithms, we can still use a precomputed Hessian when one is available. Since calculating a Hessian can be expensive, we should only calculate it once per iteration and use it both in computing in the Hessian-vector product as well as the Hessian preconditioner.

Overall, we do not prioritize computing the Hessian explicitly as it tends to require a lot of memory. In addition, we rely on Newton's method in order to obtain quadratic convergence, but this fast convergence only occurs when close to the optimal solution. When far away from the optimal solution, we waste computational effort when fully computing second-order information. Generally, truncated-CG does a good job at determining how many Hessian-vector products are required and this does not require a fully computed Hessian.

**Computation** Factorization, inverse, or approximate inverse of the Hessian

**Problem Class** Unconstrained, Inequality Constrained

**Priority** Low

**Number Stored** 1

**Description** For problems without equality constraints, Optizelle allows the user to define a preconditioner for the Hessian. Recall, the null space projection inherent to the composite-step SQP method precludes a Hessian preconditioner from being used on problems with equality constraints. For more details see the section (Optional) Define the preconditioners. In any case, barring some kind of problem specific preconditioner, we can always compute and then factorize the Hessian to be used as a preconditioner. If we do this, we should also cache the Hessian computation itself.

Overall, we do not prioritize caching this information. Similar to the discussion of caching the Hessian, far from the optimal solution, Newton's method does not guarantee quadratic convergence. Therefore, we waste computational effort when computing the Hessian and factorizing it every iteration in order to force a pure Newton step.

**Computation** Total derivative (Jacobian) of the equality constraints

**Problem Class** Equality Constrained, Constrained

**Priority** High

**Number Stored** 2

**Description** Although Optizelle only requires the action of the derivative of the equality constraints on a vector, $g'(x)\partial x$, we benefit greatly from computing the total derivative $g'(x)$ and caching the result. First, depending on the inner product, when $g'(x)$ or $g'(x)^*$ is explicitly available, we can quickly compute its adjoint. For example, when using the inner product $\langle x, y\rangle = x^T y$, we simply have to transpose the matrix. Second, each augmented system solve requires the repeated application of $g'(x)\partial x$ and $g'(x)^*\partial y$. Combined with the first point, we can compute these operations by simply multiplying the cached result by a vector. Third, when solving a problem with more than tens of variables, we require a preconditioner for the augmented system, which can be accomplished by finding a preconditioner for the operator $g'(x)g'(x)^*$. When these derivatives are explicitly available, we can easily form and factorize this matrix. As we discuss below, we should also cache this factorization.

Note, unlike most of the other caching, we require two cached elements for an efficient code. During globalization, we compute a new equality multiplier, which requires an augmented system solve at the trial point. If we accept the point, we can reuse the new cached derivative. However, if we reject the point, we will continue to require the current cached derivative. As a final note, it's often easier to cache and store $g'(x)^*$ as opposed to $g'(x)$. For example, given the inner product $\langle x, y\rangle = x^T y$ and a function of the form

$$g(x) = \begin{bmatrix} g_1(x) \\ \vdots \\ g_m(x) \end{bmatrix},$$

we can compute $g'(x)^*$ as

$$g'(x)^* = \begin{bmatrix} \nabla g_1(x) & \dots & \nabla g_m(x) \end{bmatrix}.$$

Especially with tools like automatic differentiation, this form becomes somewhat more natural to compute since we don't have to compute an extra transpose, which we undo later. Further, if we decide to compute the Schur preconditioner using a QR factorization, we actually factorize $g'(x)^*$ and not $g'(x)$. Though, as we stated above, we can quickly compute one form from the other, so we always use what's easiest to compute and calculate. For more information on preconditioning, see the section (Optional) Define the preconditioners.

**Computation** Factorization, inverse, or approximate inverse for the Schur preconditioner

**Problem Class** Equality Constrained, Constrained

**Priority** High

**Number Stored** 2

**Description** As we discuss in the section (Optional) Define the preconditioners, we require a Schur preconditioner for equality constrained problems that contain more than tens of variables. To accomplish this, we generally factorize $g'(x)g'(x)^*$, but we can use a problem specific preconditioner as well. In either case, it's important that we cache this computation since we repeatedly require it and it's likely expensive to compute. Similar to our discussion of caching the total derivative of the equality constraints, we require two cached factorizations for an efficient code.

**Computation** Adjoint of the second derivative of the equality constraints applied to a vector

**Problem Class** Equality Constrained, Constrained

**Priority** Low

**Number Stored** 1

**Description** During the tangential subproblem, which solves the optimality conditions, we require the repeated computation of $(g''(x)\partial x)^* y$. Sometimes, we can precompute part of this computation, which can accelerate this application. For example, when we use the inner product $\langle x, y\rangle = x^T y$ and have a function of the form

$$g(x) = \begin{bmatrix} g_1(x) \\ \vdots \\ g_m(x) \end{bmatrix},$$

we have that

$$(g''(x)\partial x)^* y = \left(\sum_{i=1}^{m} y_i \nabla^2 g_i(x)\right) \partial x.$$

In this case, we can cache the quantity

$$\sum_{i=1}^{m} y_i \nabla^2 g_i(x)$$

to accelerate the computation.

Most of the time, we do not prioritize caching this operator. This operator has the same size as the Hessian, which tends to require a lot of memory. Further, when far from the optimal solution, we may only require the action of this operator on a vector a few times each iteration. Therefore, computing the entire operator can be wasteful.

In order to illustrate these caching techniques, let us setup and solve a simple parameter estimation problem. In parameter estimation, we seek an unknown parameter, $k$, that characterizes a model, which is often a PDE describing some kind of physical system. In order to find these parameters, we run a series of experiments on the physical system and collect the measurable data, $d$. Then, we match this data to the output of the model, $u$. For example, we can model a parameter estimation problem governed by the steady-state convection-diffusion equations in 1-D as

$$\begin{array}{ll} \displaystyle\min_{k\in\mathbb{R}^2, u\in C^2([0,1])} & \frac{1}{2}\|u-d\|^2 \\ \text{st} & k_1 \nabla\cdot(\nabla u) + k_2 \nabla \cdot u = f \\ & u(0) = a \\ & u(1) = b. \end{array}$$

To be sure, we give the simplest possible case here. Really, there should be a time component and $k$ should represent material properties that vary spatially like $u$. Nevertheless, this problem will demonstrate that even a problem with only two variables can be very expensive to solve and that intermediate quantities should be cached appropriately. To that end, our strategy for this example will be to

1. Discretize the differential equation using a finite-difference method
2. Implement caching on the reduced-space (unconstrained) formulation
3. Implement caching on the full-space (equality constrained) formulation

This includes code written in MATLAB/Octave demonstrating the caching called `computation_caching` in the examples directory. We explain the terms *reduced-space* and *full-space* below.

**Discretization**

In order to discretize the diffusion operator, $\nabla \cdot \nabla$, we use the second-order accurate finite-difference operator

$$A = \frac{1}{\partial x^2} \begin{bmatrix} -2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{bmatrix}.$$

In order to accommodate the Dirichlet boundary conditions, we also define a vector that we use to modify the right hand side with information about the boundary conditions,

$$\hat{A} = \frac{1}{\partial x^2} \begin{bmatrix} -a \\ 0 \\ \vdots \\ 0 \\ -b \end{bmatrix}.$$

Normally, we just subtract this quantity from the discretized $f$, but since we have unknown material properties $k$, we represent it explicitly. Next, we discretize the convection operator, $\nabla \cdot$, using the first-order accurate finite difference operator

$$B = \frac{1}{\partial x} \begin{bmatrix} 1 & & & \\ -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{bmatrix}$$

As before, we accommodate the Dirichlet boundary condition with a vector to modify the right hand side with information about the boundary conditions,

$$\hat{B} = \frac{1}{\partial x} \begin{bmatrix} -a \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

This allows us to specify the discretized parameter estimation problem as

$$\begin{aligned} \min_{k \in \mathbb{R}^2, u \in \mathbb{R}^m} \quad & \tfrac{1}{2}\|u - d\|^2 \\ \text{st} \quad & (k_1 A + k_2 B)u = f - k_1 \hat{A} - k_2 \hat{B}. \end{aligned}$$

For brevity, we specify that

$$\begin{aligned} C(k) =& k_1 A + k_2 B \\ g(k) =& f - k_1 \hat{A} - k_2 \hat{B}, \end{aligned}$$

which allows us to reformulate the discretized parameter estimation problem as

$$\begin{aligned} \min_{k \in \mathbb{R}^2, u \in \mathbb{R}^m} \quad & \tfrac{1}{2}\|u - d\|^2 \\ \text{st} \quad & C(k)u = g(k). \end{aligned}$$

We call the above formulation the *full-space formulation.* Alternatively, we can solve for $u$ in the constraints and instead solve

$$\min_{k \in \mathbb{R}^2} \quad \tfrac{1}{2}\|C(k)^{-1} g(k) - d\|^2$$

which we call the *reduced-space formulation.*

**Caching the reduced-space (unconstrained) formulation**
In the reduced-space formulation, let us set

$$J(k) = \frac{1}{2}\|C(k)^{-1}g(k) - d\|^2.$$

In order to optimize with this function, we require the gradient and the Hessian-vector product. In order to derive the gradient, we calculate the partial derivative with respect to $k_i$ as

$$\begin{aligned}J_i'(k) =& \langle C(k)^{-1}g(k) - d, -C(k)^{-1}C_i'(k)C(k)^{-1}g(k) + C(k)^{-1}g_i'(k)\rangle\\ =& \langle C(k)^{-1}g(k) - d, -C(k)^{-1}(C_i'(k)C(k)^{-1}g(k) - g_i'(k))\rangle\end{aligned}$$

where

$$\begin{aligned}C_1'(k) &= A, & C_2'(k) &= B,\\ g_1'(k) &= -\hat{A}, & g_2'(k) &= -\hat{B}.\end{aligned}$$

Then,

$$\nabla J(k) = \begin{bmatrix} J_1'(k)\\ J_2'(k)\end{bmatrix}.$$

In order to calculate the Hessian-vector product, we continue this process and compute the full Hessian. We see that the second partial derivative of $J$ with respect to $k_i$ and $k_j$ is

$$\begin{aligned}J_{ij}''(k) =& \langle -C(k)^{-1}(C_j'(k)C(k)^{-1}g(k) - g_j'(k)), -C(k)^{-1}(C_i'(k)C(k)^{-1}g(k) - g_i'(k))\rangle\\ &+ \langle C(k)^{-1}g(k) - d, C(k)^{-1}C_j'(k)C(k)^{-1}(C_i'(k)C(k)^{-1}g(k) - g_i'(k))\rangle\\ &+ \langle C(k)^{-1}g(k) - d, C(k)^{-1}(C_i'(k)C(k)^{-1}C_j'(k)C(k)^{-1}g(k))\rangle\\ &+ \langle C(k)^{-1}g(k) - d, -C(k)^{-1}(C_i'(k)C(k)^{-1}g_j'(k)\rangle.\end{aligned}$$

Certainly, we could group terms more optimally, but this formulation is good enough for our purposes. Then, we have that

$$\nabla^2 J(k) = \begin{bmatrix} J_{11}''(k) & J_{12}''(k)\\ J_{21}''(k) & J_{22}''(k)\end{bmatrix}.$$

At this point, we can implement the necessary optimization functions and cache effectively. We begin with caching the initial objective function solve in the code

```
% Evaluates the objective
function z = obj_eval(params,x)
    % Cached objective evaluation. Really, this only saves us the first
    % objective evaluation as the subsequent evaluations are cached by
    % Optizelle
    global ocache

    % Performance diagnostis
    global diagnostics

    % Grab the cached objective evaluation when possible
    if ~isempty(ocache) && isequal(x,ocache.x)
        z = ocache.eval;
        diagnostics.used_cached_objective = diagnostics.used_cached_objective+1;
    else
        % We don't use the caching state solve here because the objective
        % may be evaluated at multiple points during a single optimization
        % iteration, primarily for globalization. This differs from the
        % gradient and Hessian-vector product, which are both evalated at a
        % fixed point each iteration.
        u = state_uncached(params,x,rhs(params,x));

        % Evaluate the objective
        z = 0.5 * norm(u-params.d)^2;
```

```
    end
end

% Evaluates the gradient
function grad = obj_grad(params,x)
    % Cached objective evaluation
    global ocache

    % Solve for the current solution
    u = state(params,x,rhs(params,x));

    % Cached the state solution globally for the objective
    if isempty(ocache) || ~isequal(x,ocache.x)
        ocache.x = x;
        ocache.eval = 0.5 * norm(u-params.d)^2;
    end

    % Set each element of the gradient
    grad = zeros(2,1);
    for i=1:2
        grad(i) = innr( ...
            u-params.d, ...
            -state(params,x,op_p(i,params,x)*u - rhs_p(i,params,x)));
    end
end
```

In the function `obj_grad`, we compute the objective during the gradient solve and store it in the global variable `ocache`. Then, the function `obj_eval` uses this cached value when possible. Note, it's possible to accomplish the same effect without global variables by using an intermediate function with persistent variables, but this method works well enough. Next, we cache the state solves with the code

```
% Solves the discretized PDE with caching
function z = state(params,x,rhs)
    % Keep track of where the solve occurs
    persistent cache

    % Performance diagnostics
    global diagnostics

    % Cache the factorization when required
    if isempty(cache) || ~isequal(x,cache.x)
        % Save the point we're factorizing at
        cache.x = x;

        % Factorize the operator
        [cache.l cache.u cache.p cache.q cache.r] = ...
            lu(op(params,x),'vector');

        % Keep track that we did a new factorization
        diagnostics.state_factorization_cached = ...
            diagnostics.state_factorization_cached+1;
    end

    % Solve the linear system
    z = zeros(size(rhs));
    z(cache.q) = cache.u\(cache.l\(cache.r(:,cache.p)\rhs));
end
```

We greatly improve the code's performance with this routine because it insures that we only factorize the linear system associated with the discretized convection-diffusion equations once per iteration. It accomplishes this by storing the cached results in the persistent variable `cache`. As far as the second-order information, we see how to compute and cache the Hessian-vector product with the code

```
% Evaluates the Hessian-vector product
function hv = obj_hv(params,x,dx)
    hv = hessian(params,x)*dx;
end

% Finds the Hessian
function H = hessian(params,x)
    % Keep track of where the construction occurs
    persistent cache

    % Performance diagnostics
    global diagnostics

    % Cache the Hessian when required
    if isempty(cache) || ~isequal(x,cache.x)
        % Save the point we're evaluating the Hessian at
        cache.x = x;

        % Solve for the current solution
        u = state(params,x,rhs(params,x));

        % Calculate the Hessian
        cache.H = zeros(2);
        innr = @(x,y)x'*y;
        for j=1:2
            for i=1:j
                cache.H(i,j) = ...
                    innr( ...
                        -state(params,x, ...
                            op_p(j,params,x)*u - rhs_p(j,params,x)), ...
                        -state(params,x, ...
                            op_p(i,params,x)*u - rhs_p(i,params,x))) + ...
                    innr(u-params.d, ...
                        state(params,x, ...
                            op_p(j,params,x) * state(params,x, ...
                                op_p(i,params,x)*u-rhs_p(i,params,x))))+ ...
                    innr(u-params.d, ...
                        state(params,x, ...
                            op_p(i,params,x) * state(params,x, ...
                                op_p(j,params,x) * u))) + ...
                    innr(u-params.d, ...
                        -state(params,x, ...
                            op_p(i,params,x) * ...
                                state(params,x,rhs_p(j,params,x))));

            end
        end
        cache.H(2,1)=cache.H(1,2);

        % Keep track that we cache a Hessian
        diagnostics.hessian_cached = diagnostics.hessian_cached+1;
    end
```

```
    % Evaluate the Hessian-vector product
    H = cache.H;
end
```

Notice that we compute and cache the dense Hessian in the routine `hessian`, which makes the Hessian-vector product a simple multiplication. As before, we accomplish this caching with the persistent variable `cache`. Also note, this code relies on the cached state solves we describe above for fast performance. Finally, we implement and cache a Hessian preconditioner using the inverse of the Hessian computed with the code

```
% Evaluates the inverse of the Hessian applied to a vector
function ihv = obj_hv_inv(params,x,dx)
    % Keep track of where the factorization occurs
    persistent cache

    % Performance diagnostics
    global diagnostics

    % Cache the Hessian factorization when required
    if isempty(cache) || ~isequal(x,cache.x)
        % Save the point we're factorizing the Hessian factorization at
        cache.x = x;

        % Grab the current Hessian
        H = hessian(params,x);

        % Factorize the Hessian
        [cache.l cache.u cache.p]=lu(H,'vector');

        % Keep track that we cache a Hessian factorization
        diagnostics.hessian_factorization_cached = ...
            diagnostics.hessian_factorization_cached+1;
    end

    % Apply the inverse to the direction
    ihv = cache.u\(cache.l\dx(cache.p));
end
```

Similar to the other functions, we cache the intermediate results in the persistent variable `cache`. Also note that we rely on the cached Hessian in the code listed above.

**Caching the full-space (equality constrained) formulation**

If the full-space formulation, we focus on the constraint

$$G(k,u) = C(k)u - g(k)$$

In order to derive the total derivative, we note that

$$\begin{aligned} G'_{k_i}(k,u) =& C'_i u - g'_i(k) \\ G'_u(k,u) =& C(k). \end{aligned}$$

This implies that the total derivative and its adjoint are

$$\begin{aligned} G'(k,u) &= \begin{bmatrix} C'_1 u - g'_1(k) & C'_2 u - g'_2(k) & C(k) \end{bmatrix} \\ G'(k,u)^* &= \begin{bmatrix} (C'_1 u - g'_1(k))^T \\ (C'_2 u - g'_2(k))^T \\ C(k)^T \end{bmatrix}. \end{aligned}$$

We wrote out the adjoint explicitly because it makes it easier to derive the adjoint of the second-derivative in a more cacheable form

$$(G''(k,u)(\partial k, \partial u))^* \partial y = \begin{bmatrix} 0 & 0 & \partial y^T C'_1(k) \\ 0 & 0 & \partial y^T C'_2(k) \\ C'_1(k)^T \partial y & C'_2(k)^T \partial y & 0 \end{bmatrix} \begin{bmatrix} \partial k_1 \\ \partial k_2 \\ \partial u \end{bmatrix}$$

Now, let us look at the code that caches these operations effectively. First, we start with the code that caches the total derivative of $G$

```
% Evaluates the derivative of the equality constraint
function z = eq_p(params,x,dx)
    z = deriv(params,x)*dx;
end

% Evaluates the adjoint of the derivative of the equality constraint
function z = eq_ps(params,x,dy)
    z = deriv(params,x)'*dy;
end

% Finds the total derivative of the equality constraints
function D = deriv(params,x)
    % Keep track of where the evaluation occurs
    persistent cache

    % Performance diagnostics
    global diagnostics

    % Figure out if we match a cached element
    [cache iscached]=cache_search(cache,x);

    % If we don't have a match, cache a new factorization
    if ~iscached
        % Save the current location
        cache{1}.x = x;

        % Find the total derivative
        cache{1}.D = [ ...
            op_p(1,params,x)*x(params.idx.u)-rhs_p(1,params,x) ...
            op_p(2,params,x)*x(params.idx.u)-rhs_p(2,params,x) ...
            op(params,x)];

        % Keep track that we cached a derivative
        diagnostics.first_derivative_cached = ...
            diagnostics.first_derivative_cached+1;
    end

    % Return the derivative
    D = cache{1}.D;
end

% Prepares our cached element according to the following scheme
%
% 1. Item not cached, copy first cached element to the second. Return that no
%    cached item found.
%
% 2. Item found in first cached element. Return that cached item found.
%
% 3. Item found in second cached element. Exchange first and second cached
%    elements. Return that cached item found.
function [cache iscached] = cache_search(cache,x)
    % Determine what cached item matches x
    which = 0;
    if ~isempty(cache)
        for i=1:length(cache)
```

```
            if isequal(x,cache{i}.x)
                which = i;
                break;
            end
        end
    end

    % No items match
    if which==0
        iscached = 0;
        if ~isempty(cache)
            cache{2} = cache{1};
        end

    % First item matches
    elseif which==1
        iscached = 1;

    % Second item matches
    elseif which==2
        iscached = 1;
        cache(2:-1:1)=cache;
    end
end
```

Here, we see that our reliance on computing an explicit representation for the total derivative of $G$ simplifies the functions `eq_p` and `eq_ps` to simple multiplications. Next, as before, we store the cached information in a persistent variable called `cache`. However, unlike before, we store two separate cached items and manage them with the function `cache_search`. In this function, we keep the most recently used cached item as the first element in the cache. When we evaluate the derivative at a new point, we discard the the second item. Recall, we require two cached items due to an additional augmented system solve for the equality multiplier during globalization. In a similar manner, we define the code that implements and caches the Schur preconditioner as

```
% Evaluates the Schur preconditioner
function z = eq_schur(params,x,dx)
    % Keep track of where the evaluation occurs
    persistent cache

    % Performance diagnostics
    global diagnostics

    % Here, we need to cache two elements due to the equality multiplier solve.
    % Basically, the equality multiplier solve during globalization requires a
    % solve at a new iterate. If globalization accepts this point, we can
    % reuse this factorization. However, if globalization rejects this point,
    % we want to use our old factorization.

    % Figure out if we match a cached element
    [cache iscached]=cache_search(cache,x);

    % If we don't have a match, cache a new factorization
    if ~iscached
        % Save the current location
        cache{1}.x = x;

        % Exact Schur preconditioner
        if params.approx_schur==0
```

```
            % Factorize the total derivative of g'
            [q cache{1}.r] = qr(deriv(params,x)',0);

        % Approximate Schur preconditioner
        else
            % Factorize the differential operator
            [cache{1}.l cache{1}.u cache{1}.p cache{1}.q cache{1}.r] = ...
                lu(op(params,x),'vector');
        end

        % Keep track that we did a new factorization
        diagnostics.factorization_cached = diagnostics.factorization_cached+1;
    end

    % Solve the linear system
    if params.approx_schur==0
        z = cache{1}.r\(cache{1}.r'\dx);
    else
        % Forward
        z=zeros(params.nx,1);
        z(cache{1}.q) = cache{1}.u\(cache{1}.l\(cache{1}.r(:,cache{1}.p)\dx));

        % Adjoint
        z = cache{1}.r(:,cache{1}.p)'\(cache{1}.l'\(cache{1}.u'\z(cache{1}.q)));
    end
end
```

Like the code that cached the total derivative of $G$, we cache two elements in the persistent variable `cache`. However, here, we store the factorization of the system $G'(k,u)G'(k,u)^*$. Note, caching the total derivative above helps accelerate this code as well. As a side note, we actually define two different preconditioners in this code. The true Schur preconditioner factorizes the system $G'(k,u)G'(k,u)^*$, which typically yields a dense factorization due to the derivatives with respect to $k$. Alternatively, we can define an approximate Schur preconditioner from the factorization of $G'_u(k,u)G'_u(k,u)^*$. Although we can no longer solve the augmented system in exactly three iterations, this preconditioner allows us to factorize $G'_u(k,u)$ directly, which yields a sparse decomposition. Finally, we cache the adjoint of the second derivative applied to the equality multiplier with the code

```
% Evaluates the adjoint of second derivative of the equality constraint
function z = eq_pps(params,x,dx,dy)
    z = deriv2(params,x,dy)*dx;
end

% Finds the second total derivative adjoint of the equality constraints applied
% to the equality multiplier
function D2 = deriv2(params,x,dy)
    % Keep track of where the evaluation occurs
    persistent cache
    global diagnostics

    % Cache the total derivative when possible
    if isempty(cache) || ~isequal(x,cache.x) || ~isequal(dy,cache.dy)
        % Save the current location
        cache.x = x;
        cache.dy = dy;

        % Find the adjoint of the second derivative applied to the equality
        % multiplier
        cache.D2 = sparse(params.nx+2,params.nx+2);
```

```
        cache.D2(params.idx.k(1),params.idx.u) = dy'*op_p(1,params,x);
        cache.D2(params.idx.k(2),params.idx.u) = dy'*op_p(2,params,x);
        cache.D2(params.idx.u,params.idx.k(1)) = op_p(1,params,x)'*dy;
        cache.D2(params.idx.u,params.idx.k(2)) = op_p(2,params,x)'*dy;

        % Keep track that we cache a derivative
        diagnostics.second_derivative_cached = ...
            diagnostics.second_derivative_cached+1;
    end

    % Return the derivative
    D2 = cache.D2;
end
```

As before, we store the cached information in a persistent variable called `cache`. The nuance in this case is that we should check both `x` and `dy` when determining whether we've moved to a new point and need to recompute the second derivative.

# 7 Additional examples

During the configure process, we compile and install a variety of examples whenever the ENABLE_CPP_EXAMPLES, ENABLE_PYTHON_EXAMPLES, or ENABLE_MATLAB_EXAMPLES are turned to ON. For reference, we include some of these examples here.

## 7.1 Simple equality constrained

In our Simple equality constrained example, we optimize the formulation

$$\begin{array}{ll} \min\limits_{x\in\mathbb{R}^2} & x^2+y^2 \\ \text{st} & (x-2)^2+(y-2)^2=1 \end{array}$$

with the code:

**Language** C++

**Code**

```
// Optimize a simple optimization problem with an optimal solution
// of (2-sqrt(2)/2,2-sqrt(2)/2).

#include "optizelle/optizelle.h"
#include "optizelle/vspaces.h"
#include "optizelle/json.h"
#include <iostream>
#include <iomanip>
#include <cstdlib>

//---Objective0---
// Squares its input
template <typename Real>
Real sq(Real const & x){
    return x*x;
}

// Define a simple objective where
//
// f(x,y)=x^2+y^2
//
struct MyObj
    : public Optizelle::ScalarValuedFunction <double,Optizelle::Rm>
{
    typedef Optizelle::Rm <double> X;
```

```
    // Evaluation
    double eval(X::Vector const & x) const {
        return sq(x[0])+sq(x[1]);
    }

    // Gradient
    void grad(
        X::Vector const & x,
        X::Vector & grad
    ) const {
        grad[0]=2.*x[0];
        grad[1]=2.*x[1];
    }

    // Hessian-vector product
    void hessvec(
        X::Vector const & x,
        X::Vector const & dx,
        X::Vector & H_dx
    ) const {
        H_dx[0]=2.*dx[0];
        H_dx[1]=2.*dx[1];
    }
};
//---Objective1---

//---EqualityConstraint0---
// Define a simple equality constraint
//
// g(x,y)= [ (x-2)^2 + (y-2)^2 = 1 ]
//
struct MyEq
    :public Optizelle::VectorValuedFunction<double,Optizelle::Rm,Optizelle::Rm>
{
    typedef Optizelle::Rm <double> X;
    typedef Optizelle::Rm <double> Y;

    // y=g(x)
    void eval(
        X::Vector const & x,
        Y::Vector & y
    ) const {
        y[0] = sq(x[0]-2.)+sq(x[1]-2.)-1.;
    }

    // y=g'(x)dx
    void p(
        X::Vector const & x,
        X::Vector const & dx,
        Y::Vector & y
    ) const {
        y[0] = 2.*(x[0]-2.)*dx[0]+2.*(x[1]-2.)*dx[1];
    }

    // xhat=g'(x)*dy
    void ps(
        X::Vector const & x,
```

```
        Y::Vector const & dy,
        X::Vector & xhat
    ) const {
        xhat[0] = 2.*(x[0]-2.)*dy[0];
        xhat[1] = 2.*(x[1]-2.)*dy[0];
    }

    // xhat=(g''(x)dx)*dy
    void pps(
        X::Vector const & x,
        X::Vector const & dx,
        Y::Vector const & dy,
        X::Vector & xhat
    ) const {
        xhat[0] = 2.*dx[0]*dy[0];
        xhat[1] = 2.*dx[1]*dy[0];
    }
};
//---EqualityConstraint1---

//---Preconditioner0---
// Define a Schur preconditioner for the equality constraints
struct MyPrecon:
    public Optizelle::Operator <double,Optizelle::Rm,Optizelle::Rm>
{
public:
    typedef Optizelle::Rm <double> X;
    typedef X::Vector X_Vector;
    typedef Optizelle::Rm <double> Y;
    typedef Y::Vector Y_Vector;
private:
    X_Vector& x;
public:
    MyPrecon(X::Vector& x_) : x(x_) {}
    void eval(Y_Vector const & dy,Y_Vector & result) const {
        result[0]=dy[0]/sq(4.*(x[0]-2.)+4.*sq(x[1]-2.));
    }
};
//---Preconditioner1---

int main(int argc,char* argv[]){
    // Read in the name for the input file
    if(argc!=2) {
        std::cerr << "simple_equalty <parameters>" << std::endl;
        exit(EXIT_FAILURE);
    }
    auto fname = argv[1];

    // Create a type shortcut
    using Optizelle::Rm;

    //---State0---
    // Generate an initial guess
    auto x = std::vector <double> {2.1, 1.1};

    // Allocate memory for the equality multiplier
    auto y = std::vector <double> (1);
```

```
    // Create an optimization state
    Optizelle::EqualityConstrained <double,Rm,Rm>::State::t state(x,y);
    //---State1---

    //---Parameters0---
    // Read the parameters from file
    Optizelle::json::EqualityConstrained <double,Optizelle::Rm,Optizelle::Rm>
        ::read(fname,state);
    //---Parameters1---

    //---Functions0---
    // Create a bundle of functions
    Optizelle::EqualityConstrained <double,Rm,Rm>::Functions::t fns;
    fns.f.reset(new MyObj);
    fns.g.reset(new MyEq);
    fns.PSchur_left.reset(new MyPrecon(state.x));
    //---Functions1---

    //---Solver0---
    // Solve the optimization problem
    Optizelle::EqualityConstrained <double,Rm,Rm>::Algorithms::getMin(
        Optizelle::Messaging::stdout,fns,state);
    //---Solver1---

    //---Extract0---
    // Print out the reason for convergence
    std::cout << "The algorithm converged due to: " <<
        Optizelle::OptimizationStop::to_string(state.opt_stop) <<
        std::endl;

    // Print out the final answer
    std::cout << std::scientific << std::setprecision(16)
        << "The optimal point is: (" << state.x[0] << ','
    << state.x[1] << ')' << std::endl;
    //---Extract1---

    // Write out the final answer to file
    Optizelle::json::EqualityConstrained <double,Optizelle::Rm,Optizelle::Rm>
        ::write_restart("solution.json",state);

    // Return that we've exited successfuly
    return EXIT_SUCCESS;
}
```

**Language** Python

**Code**

```
# Optimize a simple optimization problem with an optimal solution
# of (2-sqrt(2)/2,2-sqrt(2)/2).

import Optizelle
import numpy
import sys

#---Objective0---
# Squares its input
```

```
sq = lambda x:x*x

# Define a simple objective where
#
# f(x,y)=x^2+y^2
#
class MyObj(Optizelle.ScalarValuedFunction):

    # Evaluation
    def eval(self,x):
        return sq(x[0])+sq(x[1])

    # Gradient
    def grad(self,x,grad):
        grad[0]=2.*x[0]
        grad[1]=2.*x[1]

    # Hessian-vector product
    def hessvec(self,x,dx,H_dx):
        H_dx[0]=2.*dx[0]
        H_dx[1]=2.*dx[1]
#---Objective1---

#---EqualityConstraint0---
# Define a simple equality constraint
#
# g(x,y)= [ (x-2)^2 + (y-2)^2 = 1 ]
#
class MyEq(Optizelle.VectorValuedFunction):

    # y=g(x)
    def eval(self,x,y):
        y[0] = sq(x[0]-2.)+sq(x[1]-2.)-1.

    # y=g'(x)dx
    def p(self,x,dx,y):
        y[0] = 2.*(x[0]-2.)*dx[0]+2.*(x[1]-2.)*dx[1]

    # xhat=g'(x)*dy
    def ps(self,x,dy,xhat):
        xhat[0] = 2.*(x[0]-2.)*dy[0]
        xhat[1] = 2.*(x[1]-2.)*dy[0]

    # xhat=(g''(x)dx)*dy
    def pps(self,x,dx,dy,xhat):
        xhat[0] = 2.*dx[0]*dy[0]
        xhat[1] = 2.*dx[1]*dy[0]
#---EqualityConstraint1---

#---Preconditioner0---
# Define a Schur preconditioner for the equality constraints
class MyPrecon(Optizelle.Operator):
    def eval(self,state,dy,result):
        result[0]=dy[0]/sq(4.*(x[0]-2.)+4.*sq(x[1]-2.))
#---Preconditioner1---

# Read in the name for the input file
```

```python
if len(sys.argv)!=2:
    sys.exit("simple_equality.py <parameters>")
fname=sys.argv[1]

#---State0---
# Generate an initial guess
x = numpy.array([2.1,1.1])

# Allocate memory for the equality multiplier
y = numpy.array([0.])

# Create an optimization state
state=Optizelle.EqualityConstrained.State.t(Optizelle.Rm,Optizelle.Rm,x,y)
#---State1---

#---Parameters0---
# Read the parameters from file
Optizelle.json.EqualityConstrained.read(Optizelle.Rm,Optizelle.Rm,fname,state)
#---Parameters1---

#---Functions0---
# Create a bundle of functions
fns=Optizelle.EqualityConstrained.Functions.t()
fns.f=MyObj()
fns.g=MyEq()
fns.PSchur_left=MyPrecon()
#---Functions1---

#---Solver0---
# Solve the optimization problem
Optizelle.EqualityConstrained.Algorithms.getMin(
    Optizelle.Rm,Optizelle.Rm,Optizelle.Messaging.stdout,fns,state)
#---Solver1---

#---Extract0---
# Print out the reason for convergence
print "The algorithm converged due to: %s" % (
    Optizelle.OptimizationStop.to_string(state.opt_stop))

# Print out the final answer
print "The optimal point is: (%e,%e)" % (state.x[0],state.x[1])
#---Extract1---

# Write out the final answer to file
Optizelle.json.EqualityConstrained.write_restart(
    Optizelle.Rm,Optizelle.Rm,"solution.json",state)
```

**Language** MATLAB/Octave

**Code**

```matlab
% Optimize a simple optimization problem with an optimal solution
% of (2-sqrt(2)/2,2-sqrt(2)/2).
function simple_equality(fname)
    % Read in the name for the input file
    if nargin ~=1
        error('simple_equality <parameters>');
    end
```

```
    % Execute the optimization
    main(fname);
end

%---Objective0---
% Squares its input
function z = sq(x)
    z=x*x;
end

% Define a simple objective where
%
% f(x,y)=x^2+y^2
%
function self = MyObj()

    % Evaluation
    self.eval = @(x) sq(x(1))+sq(x(2));

    % Gradient
    self.grad = @(x) [ ...
        2.*x(1); ...
        2.*x(2)];

    % Hessian-vector product
    self.hessvec = @(x,dx) [ ...
        2.*dx(1); ...
        2.*dx(2)];
end
%---Objective1---

%---EqualityConstraint0---
% Define a simple equality constraint
%
% g(x,y)= [ (x-2)^2 + (y-2)^2 = 1 ]
%
function self = MyEq()

    % y=g(x)
    self.eval = @(x) [ ...
        sq(x(1)-2.)+sq(x(2)-2.)-1.];

    % y=g'(x)dx
    self.p = @(x,dx) [ ...
        2.*(x(1)-2.)*dx(1)+2.*(x(2)-2.)*dx(2)];

    % xhat=g'(x)*dy
    self.ps = @(x,dy) [ ...
        2.*(x(1)-2.)*dy(1); ...
        2.*(x(2)-2.)*dy(1)];

    % xhat=(g''(x)dx)*dy
    self.pps = @(x,dx,dy) [ ...
        2.*dx(1)*dy(1); ...
        2.*dx(2)*dy(1) ];
end
```

```
%---EqualityConstraint1---

%---Preconditioner0---
% Define a Schur preconditioner for the equality constraints
function self = MyPrecon()
    self.eval=@(state,dy)dy(1)/sq(4.*(state.x(1)-2.)+4.*sq(state.x(2)-2.));
end
%---Preconditioner1---

% Actually runs the program
function main(fname)

    % Grab the Optizelle library
    global Optizelle;
    setupOptizelle();

    %---State0---
    % Generate an initial guess
    x = [2.1;1.1];

    % Allocate memory for the equality multiplier
    y = [0.];

    % Create an optimization state
    state= Optizelle.EqualityConstrained.State.t(Optizelle.Rm,Optizelle.Rm,x,y);
    %---State1---

    %---Parameters0---
    % Read the parameters from file
    state = Optizelle.json.EqualityConstrained.read( ...
        Optizelle.Rm,Optizelle.Rm,fname,state);
    %---Parameters1---

    %---Functions0---
    % Create a bundle of functions
    fns=Optizelle.EqualityConstrained.Functions.t;
    fns.f=MyObj();
    fns.g=MyEq();
    fns.PSchur_left=MyPrecon();
    %---Functions1---

    %---Solver0---
    % Solve the optimization problem
    state = Optizelle.EqualityConstrained.Algorithms.getMin( ...
        Optizelle.Rm,Optizelle.Rm,Optizelle.Messaging.stdout,fns,state);
    %---Solver1---

    %---Extract0---
    % Print out the reason for convergence
    fprintf('The algorithm converged due to: %s\n', ...
        Optizelle.OptimizationStop.to_string(state.opt_stop));

    % Print out the final answer
    fprintf('The optimal point is: (%e,%e)\n',state.x(1),state.x(2));
    %---Extract1---

    % Write out the final answer to file
```

```
    Optizelle.json.EqualityConstrained.write_restart( ...
        Optizelle.Rm,Optizelle.Rm,'solution.json',state);
end
```

## 7.2 Simple inequality constrained

In our Simple inequality constrained example, we optimize the formulation

$$\begin{array}{rl} \min\limits_{x\in\mathbb{R}^2} & (x+1)^2+(y+1)^2 \\ \text{st} & x+2y\geq 1 \\ & 2x+y\geq 1 \end{array}$$

with the code:

**Language** C++

**Code**

```cpp
// Optimize a simple optimization problem with an optimal solution
// of (1/3,1/3)

#include "optizelle/optizelle.h"
#include "optizelle/vspaces.h"
#include "optizelle/json.h"
#include <iostream>
#include <iomanip>
#include <cstdlib>

// Squares its input
template <typename Real>
Real sq(Real const & x){
    return x*x;
}

// Define a simple objective where
//
// f(x,y)=(x+1)^2+(y+1)^2
//
struct MyObj
    : public Optizelle::ScalarValuedFunction <double,Optizelle::Rm>
{
    typedef Optizelle::Rm <double> X;

    // Evaluation
    double eval(const X::Vector& x) const {
        return sq(x[0]+1.)+sq(x[1]+1.);
    }

    // Gradient
    void grad(
        X::Vector const & x,
        X::Vector & grad
    ) const {
        grad[0]=2.*x[0]+2.;
        grad[1]=2.*x[1]+2.;
    }

    // Hessian-vector product
    void hessvec(
```

```
        X::Vector const & x,
        X::Vector const & dx,
        X::Vector & H_dx
    ) const {
        H_dx[0]=2.*dx[0];
        H_dx[1]=2.*dx[1];
    }
};

// Define simple inequalities
//
// h(x,y)= [ x + 2y >= 1 ]
//         [ 2x + y >= 1 ]
//
struct MyIneq
    :public Optizelle::VectorValuedFunction<double,Optizelle::Rm,Optizelle::Rm>
{
    typedef Optizelle::Rm <double> X;
    typedef Optizelle::Rm <double> Y;

    // y=h(x)
    void eval(
        X::Vector const & x,
        Y::Vector & y
    ) const {
        y[0]=x[0]+2.*x[1]-1.;
        y[1]=2.*x[0]+x[1]-1.;
    }

    // y=h'(x)dx
    void p(
        X::Vector const & x,
        X::Vector const & dx,
        Y::Vector & y
    ) const {
        y[0]= dx[0]+2.*dx[1];
        y[1]= 2.*dx[0]+dx[1];
    }

    // z=h'(x)*dy
    void ps(
        X::Vector const & x,
        Y::Vector const & dy,
        X::Vector & z
    ) const {
        z[0]= dy[0]+2.*dy[1];
        z[1]= 2.*dy[0]+dy[1];
    }

    // z=(h''(x)dx)*dy
    void pps(
        X::Vector const & x,
        X::Vector const & dx,
        Y::Vector const & dy,
        X::Vector & z
    ) const {
        X::zero(z);
```

```
    }
};

int main(int argc,char* argv[]){
    // Read in the name for the input file
    if(argc!=2) {
        std::cerr << "simple_inequality <parameters>" << std::endl;
        exit(EXIT_FAILURE);
    }
    auto fname = argv[1];

    // Create a type shortcut
    using Optizelle::Rm;

    // Generate an initial guess
    auto x = std::vector <double> {2.1, 1.1};

    // Allocate memory for the inequality multipler
    auto z = std::vector <double>(2);

    // Create an optimization state
    Optizelle::InequalityConstrained <double,Rm,Rm>::State::t state(x,z);

    // Read the parameters from file
    Optizelle::json::InequalityConstrained <double,Optizelle::Rm,Optizelle::Rm>
        ::read(fname,state);

    // Create a bundle of functions
    Optizelle::InequalityConstrained <double,Rm,Rm>::Functions::t fns;
    fns.f.reset(new MyObj);
    fns.h.reset(new MyIneq);

    // Solve the optimization problem
    Optizelle::InequalityConstrained <double,Rm,Rm>::Algorithms
        ::getMin(Optizelle::Messaging::stdout,fns,state);

    // Print out the reason for convergence
    std::cout << "The algorithm converged due to: " <<
        Optizelle::OptimizationStop::to_string(state.opt_stop) <<
        std::endl;

    // Print out the final answer
    std::cout << std::scientific << std::setprecision(16)
        << "The optimal point is: (" << state.x[0] << ','
    << state.x[1] << ')' << std::endl;

    // Write out the final answer to file
    Optizelle::json::InequalityConstrained<double,Rm,Rm>
        ::write_restart("solution.json",state);

    // Return that the program exited properly
    return EXIT_SUCCESS;
}
```

**Language** Python

**Code**

```
# Optimize a simple optimization problem with an optimal solution
# of (1/3,1/3)

import Optizelle
import numpy
import sys

# Squares its input
sq = lambda x:x*x

# Define a simple objective where
#
# f(x,y)=(x+1)^2+(y+1)^2
#
class MyObj(Optizelle.ScalarValuedFunction):

    # Evaluation
    def eval(self,x):
        return sq(x[0]+1.)+sq(x[1]+1.)

    # Gradient
    def grad(self,x,grad):
        grad[0]=2.*x[0]+2.
        grad[1]=2.*x[1]+2.

    # Hessian-vector product
    def hessvec(self,x,dx,H_dx):
        H_dx[0]=2.*dx[0]
        H_dx[1]=2.*dx[1]

# Define simple inequalities
#
# h(x,y)= [ x + 2y >= 1 ]
#         [ 2x + y >= 1 ]
#
class MyIneq(Optizelle.VectorValuedFunction):

    # z=h(x)
    def eval(self,x,z):
        z[0]=x[0]+2.*x[1]-1.
        z[1]=2.*x[0]+x[1]-1.

    # z=h'(x)dx
    def p(self,x,dx,z):
        z[0]= dx[0]+2.*dx[1]
        z[1]= 2.*dx[0]+dx[1]

    # xhat=h'(x)*dz
    def ps(self,x,dz,xhat):
        xhat[0]= dz[0]+2.*dz[1]
        xhat[1]= 2.*dz[0]+dz[1]

    # xhat=(h''(x)dx)*dz
    def pps(self,x,dx,dz,xhat):
        xhat.fill(0.)

# Read in the name for the input file
```

```python
if len(sys.argv)!=2:
    sys.exit("simple_inequality.py <parameters>")
fname=sys.argv[1]

# Generate an initial guess
x = numpy.array([2.1,1.1])

# Allocate memory for the inequality multiplier
z = numpy.array([0.,0.])

# Create an optimization state
state=Optizelle.InequalityConstrained.State.t(Optizelle.Rm,Optizelle.Rm,x,z)

# Read the parameters from file
Optizelle.json.InequalityConstrained.read(Optizelle.Rm,Optizelle.Rm,fname,state)

# Create a bundle of functions
fns=Optizelle.InequalityConstrained.Functions.t()
fns.f=MyObj()
fns.h=MyIneq()

# Solve the optimization problem
Optizelle.InequalityConstrained.Algorithms.getMin(
    Optizelle.Rm,Optizelle.Rm,Optizelle.Messaging.stdout,fns,state)

# Print out the reason for convergence
print "The algorithm converged due to: %s" % (
    Optizelle.OptimizationStop.to_string(state.opt_stop))

# Print out the final answer
print "The optimal point is: (%e,%e)" % (state.x[0],state.x[1])

# Write out the final answer to file
Optizelle.json.InequalityConstrained.write_restart(
    Optizelle.Rm,Optizelle.Rm,"solution.json",state)
```

**Language** MATLAB/Octave

**Code**

```matlab
% Optimize a simple optimization problem with an optimal solution
% of (1/3,1/3)
function simple_inequality(fname)
    % Read in the name for the input file
    if nargin ~=1
        error('simple_inequality <parameters>');
    end

    % Execute the optimization
    main(fname);
end

% Squares its input
function z = sq(x)
    z=x*x;
end

% Define a simple objective where
```

```
%
% f(x,y)=(x+1)^2+(y+1)^2
%
function self = MyObj()

    % Evaluation
    self.eval = @(x) sq(x(1)+1.)+sq(x(2)+1.);

    % Gradient
    self.grad = @(x) [
        2.*x(1)+2.;
        2.*x(2)+2.];

    % Hessian-vector product
    self.hessvec = @(x,dx) [
        2.*dx(1);
        2.*dx(2)];
end

% Define simple inequalities
%
% h(x,y)= [ x + 2y >= 1 ]
%         [ 2x + y >= 1 ]
%
function self = MyIneq()

    % z=h(x)
    self.eval = @(x) [
        x(1)+2.*x(2)-1. ;
        2.*x(1)+x(2)-1. ];

    % z=h'(x)dx
    self.p = @(x,dx) [
        dx(1)+2.*dx(2) ;
        2.*dx(1)+dx(2) ];

    % xhat=h'(x)*dz
    self.ps = @(x,dz) [
        dz(1)+2.*dz(2) ;
        2.*dz(1)+dz(2) ];

    % xhat=(h''(x)dx)*dz
    self.pps = @(x,dx,dz) zeros(2,1);
end

% Actually runs the program
function main(fname)

    % Grab the Optizelle library
    global Optizelle;
    setupOptizelle();

    % Generate an initial guess
    x = [2.1;1.1];

    % Allocate memory for the inequality multiplier
    z = [0.;0.];
```

```
    % Create an optimization state
    state=Optizelle.InequalityConstrained.State.t( ...
        Optizelle.Rm,Optizelle.Rm,x,z);

    % Read the parameters from file
    state=Optizelle.json.InequalityConstrained.read( ...
        Optizelle.Rm,Optizelle.Rm,fname,state);

    % Create a bundle of functions
    fns=Optizelle.InequalityConstrained.Functions.t;
    fns.f=MyObj();
    fns.h=MyIneq();

    % Solve the optimization problem
    state=Optizelle.InequalityConstrained.Algorithms.getMin( ...
        Optizelle.Rm,Optizelle.Rm,Optizelle.Messaging.stdout,fns,state);

    % Print out the reason for convergence
    fprintf('The algorithm converged due to: %s\n', ...
        Optizelle.OptimizationStop.to_string(state.opt_stop));

    % Print out the final answer
    fprintf('The optimal point is: (%e,%e)\n',state.x(1),state.x(2));

    % Write out the final answer to file
    Optizelle.json.InequalityConstrained.write_restart( ...
        Optizelle.Rm,Optizelle.Rm,'solution.json',state);
end
```

## 7.3 Simple constrained

In our Simple constrained example, we optimize the formulation

$$\begin{array}{rl} \min\limits_{x\in\mathbb{R}^2} & (x+1)^2+(y+1)^2 \\ \text{st} & x+2y=1 \\ & 2x+y\geq 1 \end{array}$$

with the code:

**Language** C++

**Code**

```
// Optimize a simple optimization problem with an optimal
// solution of (1/3,1/3)

#include "optizelle/optizelle.h"
#include "optizelle/vspaces.h"
#include "optizelle/json.h"
#include <iostream>
#include <iomanip>
#include <cstdlib>

// Squares its input
template <typename Real>
Real sq(Real const & x){
    return x*x;
}
```

```
// Define a simple objective where
//
// f(x,y)=(x+1)^2+(y+1)^2
//
struct MyObj
    : public Optizelle::ScalarValuedFunction <double,Optizelle::Rm>
{
    typedef Optizelle::Rm <double> X;

    // Evaluation
    double eval(X::Vector const & x) const {
        return sq(x[0]+1.)+sq(x[1]+1.);
    }

    // Gradient
    void grad(
        X::Vector const & x,
        X::Vector & grad
    ) const {
        grad[0]=2*x[0]+2;
        grad[1]=2*x[1]+2;
    }

    // Hessian-vector product
    void hessvec(
        X::Vector const & x,
        X::Vector const & dx,
        X::Vector & H_dx
    ) const {
        H_dx[0]=2.*dx[0];
        H_dx[1]=2.*dx[1];
    }
};

// Define a simple equality
//
// g(x,y)= [ x + 2y = 1 ]
//
struct MyEq
    :public Optizelle::VectorValuedFunction<double,Optizelle::Rm,Optizelle::Rm>
{
    typedef Optizelle::Rm <double> X;
    typedef Optizelle::Rm <double> Y;

    // y=g(x)
    void eval(
        X::Vector const & x,
        Y::Vector & y
    ) const {
        y[0]=x[0]+2.*x[1]-1.;
    }

    // y=g'(x)dx
    void p(
        X::Vector const & x,
        X::Vector const & dx,
```

```
        Y::Vector & y
    ) const {
        y[0]= dx[0]+2.*dx[1];
    }

    // xhat=g'(x)*dy
    void ps(
        X::Vector const & x,
        Y::Vector const & dy,
        X::Vector & xhat
    ) const {
        xhat[0]= dy[0];
        xhat[1]= 2.*dy[0];
    }

    // xhat=(g''(x)dx)*dy
    void pps(
        X::Vector const & x,
        X::Vector const & dx,
        Y::Vector const & dy,
        X::Vector & xhat
    ) const {
        X::zero(xhat);
    }
};

// Define a simple inequality
//
// h(x,y)= [ 2x + y >= 1 ]
//
struct MyIneq
    :public Optizelle::VectorValuedFunction<double,Optizelle::Rm,Optizelle::Rm>
{
    typedef Optizelle::Rm <double> X;
    typedef Optizelle::Rm <double> Z;

    // z=h(x)
    void eval(
        X::Vector const & x,
        Z::Vector & z
    ) const {
        z[0]=2.*x[0]+x[1]-1.;
    }

    // z=h'(x)dx
    void p(
        X::Vector const & x,
        X::Vector const & dx,
        Z::Vector & z
    ) const {
        z[0]= 2.*dx[0]+dx[1];
    }

    // xhat=h'(x)*dz
    void ps(
        X::Vector const & x,
        Z::Vector const & dz,
```

```
        X::Vector & xhat
    ) const {
        xhat[0]= 2.*dz[0];
        xhat[1]= dz[0];
    }

    // xhat=(h''(x)dx)*dz
    void pps(
        X::Vector const & x,
        X::Vector const & dx,
        Z::Vector const & dz,
        X::Vector & xhat
    ) const {
        X::zero(xhat);
    }
};

int main(int argc,char* argv[]){
    // Read in the name for the input file
    if(argc!=2) {
        std::cerr << "simple_constrained <parameters>" << std::endl;
        exit(EXIT_FAILURE);
    }
    auto fname = argv[1];

    // Create a type shortcut
    using Optizelle::Rm;

    // Generate an initial guess for the primal
    auto x = std::vector <double> {2.1, 1.1};

    // Allocate memory for equality multiplier
    auto y = std::vector <double> (1);

    // Allocate memory for the inequality multiplier
    auto z = std::vector <double> (1);

    // Create an optimization state
    Optizelle::Constrained <double,Rm,Rm,Rm>::State::t state(x,y,z);

    // Read the parameters from file
    Optizelle::json::Constrained <double,Rm,Rm,Rm>::read(fname,state);

    // Create a bundle of functions
    Optizelle::Constrained <double,Rm,Rm,Rm>::Functions::t fns;
    fns.f.reset(new MyObj);
    fns.g.reset(new MyEq);
    fns.h.reset(new MyIneq);

    // Solve the optimization problem
    Optizelle::Constrained <double,Rm,Rm,Rm>::Algorithms
        ::getMin(Optizelle::Messaging::stdout,fns,state);

    // Print out the reason for convergence
    std::cout << "The algorithm converged due to: " <<
        Optizelle::OptimizationStop::to_string(state.opt_stop) <<
        std::endl;
```

```
    // Print out the final answer
    std::cout << std::scientific << std::setprecision(16)
        << "The optimal point is: (" << state.x[0] << ','
        << state.x[1] << ')' << std::endl;

    // Write out the final answer to file
    Optizelle::json::Constrained <double,Rm,Rm,Rm>::write_restart(
        "solution.json",state);

    // Successful termination
    return EXIT_SUCCESS;
}
```

**Language** Python

**Code**

```python
# Optimize a simple optimization problem with an optimal solution
# of (1/3,1/3)

import Optizelle
import numpy
import sys

# Squares its input
sq = lambda x:x*x

# Define a simple objective where
#
# f(x,y)=(x+1)^2+(y+1)^2
#
class MyObj(Optizelle.ScalarValuedFunction):

    # Evaluation
    def eval(self,x):
        return sq(x[0]+1.)+sq(x[1]+1.)

    # Gradient
    def grad(self,x,grad):
        grad[0]=2.*x[0]+2.
        grad[1]=2.*x[1]+2.

    # Hessian-vector product
    def hessvec(self,x,dx,H_dx):
        H_dx[0]=2.*dx[0]
        H_dx[1]=2.*dx[1]

# Define a simple equality
#
# g(x,y)= [ x + 2y = 1 ]
#
class MyEq(Optizelle.VectorValuedFunction):

    # y=g(x)
    def eval(self,x,y):
        y[0]=x[0]+2.*x[1]-1.
```

```python
    # y=g'(x)dx
    def p(self,x,dx,y):
        y[0]= dx[0]+2.*dx[1]

    # xhat=g'(x)*dy
    def ps(self,x,dy,xhat):
        xhat[0]= dy[0]
        xhat[1]= 2.*dy[0]

    # xhat=(g''(x)dx)*dy
    def pps(self,x,dx,dy,xhat):
        xhat.fill(0.)

# Define simple inequalities
#
# h(x,y)= [ 2x + y >= 1 ]
#
class MyIneq(Optizelle.VectorValuedFunction):

    # z=h(x)
    def eval(self,x,z):
        z[0]=2.*x[0]+x[1]-1.

    # z=h'(x)dx
    def p(self,x,dx,z):
        z[0]= 2.*dx[0]+dx[1]

    # xhat=h'(x)*dz
    def ps(self,x,dz,xhat):
        xhat[0]= 2.*dz[0]
        xhat[1]= dz[0]

    # xhat=(h''(x)dx)*dz
    def pps(self,x,dx,dz,xhat):
        xhat.fill(0.)

# Read in the name for the input file
if len(sys.argv)!=2:
    sys.exit("simple_constrained.py <parameters>")
fname=sys.argv[1]

# Generate an initial guess
x = numpy.array([2.1,1.1])

# Allocate memory for the equality multiplier
y = numpy.array([0.])

# Allocate memory for the inequality multiplier
z = numpy.array([0.])

# Create an optimization state
state=Optizelle.Constrained.State.t(
    Optizelle.Rm,Optizelle.Rm,Optizelle.Rm,x,y,z)

# Read the parameters from file
Optizelle.json.Constrained.read(
    Optizelle.Rm,Optizelle.Rm,Optizelle.Rm,fname,state)
```

```
# Create a bundle of functions
fns=Optizelle.Constrained.Functions.t()
fns.f=MyObj()
fns.g=MyEq()
fns.h=MyIneq()

# Solve the optimization problem
Optizelle.Constrained.Algorithms.getMin(
    Optizelle.Rm,Optizelle.Rm,Optizelle.Rm,Optizelle.Messaging.stdout,fns,state)

# Print out the reason for convergence
print "The algorithm converged due to: %s" % (
    Optizelle.OptimizationStop.to_string(state.opt_stop))

# Print out the final answer
print "The optimal point is: (%e,%e)" % (state.x[0],state.x[1])

# Write out the final answer to file
Optizelle.json.Constrained.write_restart(Optizelle.Rm,Optizelle.Rm,Optizelle.Rm,
    "solution.json",state)
```

**Language** MATLAB/Octave

**Code**

```
% Optimize a simple optimization problem with an optimal solution
% of (1/3,1/3)
function simple_constrained(fname)
    % Read in the name for the input file
    if nargin ~=1
        error('simple_constrained <parameters>');
    end

    % Execute the optimization
    main(fname);
end

% Squares its input
function z = sq(x)
    z=x*x;
end

% Define a simple objective where
%
% f(x,y)=(x+1)^2+(y+1)^2
%
function self = MyObj()

    % Evaluation
    self.eval = @(x) sq(x(1)+1.)+sq(x(2)+1.);

    % Gradient
    self.grad = @(x) [
        2.*x(1)+2.;
        2.*x(2)+2.];

    % Hessian-vector product
```

```
    self.hessvec = @(x,dx) [
        2.*dx(1);
        2.*dx(2)];
end

% Define a simple equality
%
% g(x,y)= [ x + 2y = 1 ]
%
function self = MyEq()

    % y=g(x)
    self.eval = @(x) [x(1)+2.*x(2)-1.];

    % y=g'(x)dx
    self.p = @(x,dx) [dx(1)+2.*dx(2)];

    % xhat=g'(x)*dy
    self.ps = @(x,dy) [
        dy(1);
        2.*dy(1)];

    % xhat=(g''(x)dx)*dy
    self.pps = @(x,dx,dy) zeros(2,1);
end

% Define simple inequalities
%
% h(x,y)= [ 2x + y >= 1 ]
%
function self = MyIneq()

    % z=h(x)
    self.eval = @(x) [
        2.*x(1)+x(2)-1];

    % z=h'(x)dx
    self.p = @(x,dx) [
        2.*dx(1)+dx(2)];

    % xhat=h'(x)*dz
    self.ps = @(x,dz) [
        2.*dz(1)
        dz(1)];

    % xhat=(h''(x)dx)*dz
    self.pps = @(x,dx,dz) [ 0. ];
end

% Actually runs the program
function main(fname)

    % Grab the Optizelle library
    global Optizelle;
    setupOptizelle();

    % Generate an initial guess
```

```
    x = [2.1;1.1];

    % Allocate memory for the equality multiplier
    y = [0.];

    % Allocate memory for the inequality multiplier
    z = [0.];

    % Create an optimization state
    state = Optizelle.Constrained.State.t( ...
        Optizelle.Rm,Optizelle.Rm,Optizelle.Rm,x,y,z);

    % Read the parameters from file
    state = Optizelle.json.Constrained.read( ...
        Optizelle.Rm,Optizelle.Rm,Optizelle.Rm,fname,state);

    % Create a bundle of functions
    fns = Optizelle.Constrained.Functions.t;
    fns.f = MyObj();
    fns.g = MyEq();
    fns.h = MyIneq();

    % Solve the optimization problem
    state = Optizelle.Constrained.Algorithms.getMin( ...
        Optizelle.Rm,Optizelle.Rm,Optizelle.Rm,Optizelle.Messaging.stdout, ...
        fns,state);

    % Print out the reason for convergence
    fprintf('The algorithm converged due to: %s\n', ...
        Optizelle.OptimizationStop.to_string(state.opt_stop));

    % Print out the final answer
    fprintf('The optimal point is: (%e,%e)\n',state.x(1),state.x(2));

    % Write out the final answer to file
    Optizelle.json.Constrained.write_restart( ...
        Optizelle.Rm,Optizelle.Rm,Optizelle.Rm,'solution.json',state);
end
```

## 7.4 Rosenbrock advanced API

In our Rosenbrock advanced API example, we optimize the formulation

$$\min_{x\in\mathbb{R}^2} \qquad (1-x_1)^2 + 100(x_2 - x_1^2)^2.$$

using the features described in our chapter Advanced API. We accomplish this with the code:

**Language** C++

**Code**

```
// In this example, we duplicate the Rosenbrock example while demonstrating
// some of the more advanced API features such as custom vector spaces,
// messaging objects, and restarts.

#include <vector>
#include <iostream>
#include <iomanip>
#include <string>
```

```
#include <sstream>
#include <algorithm>
#include "optizelle/optizelle.h"
#include "optizelle/json.h"

// Grab Optizelle's Natural type
using Optizelle::Natural;

//---VectorSpace0---
// Defines the vector space used for optimization.
template <typename Real>
struct MyVS {
    typedef std::vector <Real> Vector;

    // Memory allocation and size setting
    static Vector init(Vector const & x) {
        return std::move(Vector(x.size()));
    }

    // y <- x (Shallow. No memory allocation.)
    static void copy(Vector const & x, Vector & y) {
        for(Natural i=0;i<x.size();i++){
            y[i]=x[i];
        }
    }

    // x <- alpha * x
    static void scal(const Real& alpha, Vector & x) {
        for(Natural i=0;i<x.size();i++){
            x[i]=alpha*x[i];
        }
    }

    // x <- 0
    static void zero(Vector & x) {
        for(Natural i=0;i<x.size();i++){
            x[i]=0.;
        }
    }

    // y <- alpha * x + y
    static void axpy(const Real& alpha, Vector const & x, Vector & y) {
        for(Natural i=0;i<x.size();i++){
            y[i]=alpha*x[i]+y[i];
        }
    }

    // innr <- <x,y>
    static Real innr(Vector const & x,Vector const & y) {
        Real z=0;
        for(Natural i=0;i<x.size();i++)
            z+=x[i]*y[i];
        return z;
    }

    // x <- random
    static void rand(Vector & x){
```

```
        std::mt19937 gen(1);
        std::uniform_real_distribution<Real> dis(Real(0.),Real(1.));
        for(Natural i=0;i<x.size();i++)
            x[i]=Real(dis(gen));
    }
    // Jordan product, z <- x o y.
    static void prod(Vector const & x, Vector const & y, Vector & z) {
        for(Natural i=0;i<x.size();i++)
            z[i]=x[i]*y[i];
    }

    // Identity element, x <- e such that x o e = x.
    static void id(Vector & x) {
        for(Natural i=0;i<x.size();i++)
            x[i]=Real(1.);
    }

    // Jordan product inverse, z <- inv(L(x)) y where L(x) y = x o y.
    static void linv(Vector const & x,Vector const & y,Vector & z) {
        for(Natural i=0;i<x.size();i++)
            z[i]=y[i]/x[i];
    }

    // Barrier function, barr <- barr(x) where x o grad barr(x) = e.
    static Real barr(Vector const & x) {
        Real z=Real(0.);
        for(Natural i=0;i<x.size();i++)
            z+=log(x[i]);
        return z;
    }

    // Line search, srch <- argmax {alpha \in Real >= 0 : alpha x + y >= 0}
    // where y > 0.
    static Real srch(Vector const & x,Vector const & y) {
        // Line search parameter
        Real alpha=std::numeric_limits <Real>::infinity();

        // Search for the optimal linesearch parameter.
        for(Natural i=0;i<x.size();i++) {
            if(x[i] < Real(0.)) {
                Real alpha0 = -y[i]/x[i];
                alpha = alpha0 < alpha ? alpha0 : alpha;
            }
        }

        return alpha;
    }

    // Symmetrization, x <- symm(x) such that L(symm(x)) is a symmetric
    // operator.
    static void symm(Vector & x) { }
};
//---VectorSpace1---

// Squares its input
template <typename Real>
Real sq(Real x){
```

```
    return x*x;
}

// Define the Rosenbrock function where
//
// f(x,y)=(1-x)^2+100(y-x^2)^2
//
struct Rosenbrock : public Optizelle::ScalarValuedFunction <double,MyVS> {
    typedef MyVS <double> X;

    // Evaluation of the Rosenbrock function
    double eval(X::Vector const & x) const {
        return sq(1.-x[0])+100.*sq(x[1]-sq(x[0]));
    }

    // Gradient
    void grad(
        X::Vector const & x,
        X::Vector & g
    ) const {
        g[0]=-400*x[0]*(x[1]-sq(x[0]))-2*(1-x[0]);
        g[1]=200*(x[1]-sq(x[0]));
    }

    // Hessian-vector product
    void hessvec(
        X::Vector const & x,
        X::Vector const & dx,
        X::Vector & H_dx
    ) const {
        H_dx[0]= (1200*sq(x[0])-400*x[1]+2)*dx[0]-400*x[0]*dx[1];
        H_dx[1]= -400*x[0]*dx[0] + 200*dx[1];
    }
};

// Define a perfect preconditioner for the Hessian
struct RosenHInv : public Optizelle::Operator <double,MyVS,MyVS> {
public:
    typedef MyVS <double> X;
    typedef X::Vector X_Vector;
private:
    X_Vector& x;
public:
    RosenHInv(X::Vector& x_) : x(x_) {}
    void eval(const X_Vector& dx,X_Vector &result) const {
        auto one_over_det=1./(80000.*sq(x[0])-80000.*x[1]+400.);
        result[0]=one_over_det*(200.*dx[0]+400.*x[0]*dx[1]);
        result[1]=one_over_det*
            (400.*x[0]*dx[0]+(1200.*x[0]*x[0]-400.*x[1]+2.)*dx[1]);
    }
};

//---Messaging0---
// Define a custom messaging object
void mymessaging(std::string const & msg) {
    std::cout << "PRINT: " << msg << std::endl;
}
```

```
//---Messaging1---

//---Serialization0---
// Define serialization routines for MyVS
namespace Optizelle {
    namespace json {
        template <>
        struct Serialization <double,MyVS> {
            static std::string serialize(
                typename MyVS <double>::Vector const & x,
                std::string const & name,
                Natural const & iter
            ) {
                // Create a string with the format
                // [ x1, x2, ..., xm ].
                std::stringstream x_json;
                x_json.setf(std::ios::scientific);
                x_json.precision(16);
                x_json << "[ ";
                for(Natural i=0;i<x.size()-1;i++)
                    x_json << x[i] << ", ";
                x_json << x.back() << " ]";

                // Return the string
                return x_json.str();
            }
            static MyVS <double>::Vector deserialize(
                typename MyVS <double>::Vector const & x_,
                std::string const & x_json_
            ) {
                // Make a copy of x_json_
                auto x_json = x_json_;

                // Filter out the commas and brackets from the string
                char formatting[] = "[],";
                for(Natural i=0;i<3;i++)
                    x_json.erase(
                        std::remove(x_json.begin(),x_json.end(),formatting[i]),
                        x_json.end());

                // Create a new vector that we eventually return
                auto x = std::vector <double>(x_.size());

                // Create a stream out of x_json
                std::stringstream ss(x_json);

                // Read in each of the elements
                for(auto i=0;i<x.size();i++)
                    ss >> x[i];

                // Return the result
                return std::move(x);
            }
        };
    }
}
//---Serialization1---
```

```
//---RestartManipulator0---
// Define a state manipulator that writes out the optimization state at
// each iteration.
struct MyRestartManipulator
    : Optizelle::StateManipulator <Optizelle::Unconstrained <double,MyVS> >
{
    void eval(
        typename Optizelle::Unconstrained <double,MyVS>
            ::Functions::t const & fns,
        typename Optizelle::Unconstrained <double,MyVS>
            ::State::t & state,
        Optizelle::OptimizationLocation::t const & loc
    ) const {
        switch(loc) {
        // At the end of the optimization iteration, write the restart file
        case Optizelle::OptimizationLocation::EndOfOptimizationIteration: {
            // Create a reasonable file name
            std::stringstream ss;
            ss << "rosenbrock_advanced_api_";
            ss << std::setw(4) << std::setfill('0') << state.iter;
            ss << ".json";

            // Write the restart file
            Optizelle::json::Unconstrained <double,MyVS>::write_restart(
                ss.str(),state);
            break;
        } default:
            break;
        }
    }
};
//---RestartManipulator1---

int main(int argc,char* argv[]) {
    // Read in the name for the parameters and optional restart file
    if(!(argc==2 || argc==3)) {
        std::cerr << "rosenbrock_advanced_api <parameters>" << std::endl;
        std::cerr << "rosenbrock_advanced_api <parameters> <restart>"
            << std::endl;
        exit(EXIT_FAILURE);
    }
    auto pname = argv[1];
    auto rname = argc==3 ? argv[2] : "";

    // Generate an initial guess for Rosenbrock
    auto x = std::vector <double> {-1.2, 1.};

    // Create an unconstrained state based on this vector
    Optizelle::Unconstrained <double,MyVS>::State::t state(x);

    //---ReadRestart0---
    // If we have a restart file, read in the parameters
    if(argc==3)
        Optizelle::json::Unconstrained <double,MyVS>::read_restart(
            rname,x,state);
```

```
    // Read additional parameters from file
    Optizelle::json::Unconstrained <double,MyVS>::read(pname,state);
    //---ReadRestart1---

    // Create the bundle of functions
    Optizelle::Unconstrained <double,MyVS>::Functions::t fns;
    fns.f.reset(new Rosenbrock);
    fns.PH.reset(new RosenHInv(state.x));

    //---Solver0---
    // Solve the optimization problem
    Optizelle::Unconstrained <double,MyVS>::Algorithms
        ::getMin(mymessaging,fns,state,MyRestartManipulator());
    //---Solver1---

    // Print out the reason for convergence
    std::cout << "The algorithm converged due to: " <<
        Optizelle::OptimizationStop::to_string(state.opt_stop) << std::endl;

    // Print out the final answer
    std::cout << "The optimal point is: (" << state.x[0] << ','
    << state.x[1] << ')' << std::endl;

    //---WriteRestart0---
    // Write out the final answer to file
    Optizelle::json::Unconstrained <double,MyVS>::write_restart(
        "solution.json",state);
    //---WriteRestart1---
}
```

**Language** Python

**Code**

```
# In this example, we duplicate the Rosenbrock example while demonstrating
# some of the more advanced API features such as custom vector spaces,
# messaging objects, and restarts.

import Optizelle
import sys
import copy
import array
import math

#---VectorSpace0---
# Defines the vector space used for optimization.
class MyVS(object):
    @staticmethod
    def init(x):
        """Memory allocation and size setting"""
        return copy.deepcopy(x)

    @staticmethod
    def copy(x,y):
        """y <- x (Shallow. No memory allocation.)"""
        y[:]=x[:]

    @staticmethod
```

```python
def scal(alpha,x):
    """x <- alpha * x"""
    for i in xrange(0,len(x)):
        x[i]=alpha*x[i]

@staticmethod
def zero(x):
    """x <- 0"""
    for i in xrange(0,len(x)):
        x[i]=0.

@staticmethod
def axpy(alpha,x,y):
    """y <- alpha * x + y"""
    for i in xrange(0,len(x)):
        y[i]=alpha*x[i]+y[i]

@staticmethod
def innr(x,y):
    """<- <x,y>"""
    return reduce(lambda z,xy:xy[0]*xy[1]+z,zip(x,y),0.)

@staticmethod
def rand(x):
    """x <- random"""
    for i in xrange(0,len(x)):
        x[i]=random.uniform(0.,1.)

@staticmethod
def prod(x,y,z):
    """Jordan product, z <- x o y"""
    for i in xrange(0,len(x)):
        z[i]=x[i]*y[i]

@staticmethod
def id(x):
    """Identity element, x <- e such that x o e = x"""
    for i in xrange(0,len(x)):
        x[i]=1.

@staticmethod
def linv(x,y,z):
    """Jordan product inverse, z <- inv(L(x)) y where L(x) y = x o y"""
    for i in xrange(0,len(x)):
        z[i]=y[i]/x[i]

@staticmethod
def barr(x):
    """Barrier function, <- barr(x) where x o grad barr(x) = e"""
    return reduce(lambda x,y:x+math.log(y),x,0.)

@staticmethod
def srch(x,y):
    """Line search, <- argmax {alpha \in Real >= 0 : alpha x + y >= 0} where y >
    alpha = float("inf")
    for i in xrange(0,len(x)):
        if x[i] < 0:
```

```
                alpha0 = -y[i]/x[i]
                if alpha0 < alpha:
                    alpha=alpha0
        return alpha

    @staticmethod
    def symm(x):
        """Symmetrization, x <- symm(x) such that L(symm(x)) is a symmetric operator
        pass
#---VectorSpace1---

# Squares its input
sq = lambda x:x*x

# Define the Rosenbrock function where
#
# f(x,y)=(1-x)^2+100(y-x^2)^2
#
class Rosenbrock(Optizelle.ScalarValuedFunction):
    # Evaluation of the Rosenbrock function
    def eval(self,x):
        return sq(1.-x[0])+100.*sq(x[1]-sq(x[0]))

    # Gradient
    def grad(self,x,grad):
        grad[0]=-400*x[0]*(x[1]-sq(x[0]))-2*(1-x[0])
        grad[1]=200*(x[1]-sq(x[0]))

    # Hessian-vector product
    def hessvec(self,x,dx,H_dx):
        H_dx[0] = (1200*sq(x[0])-400*x[1]+2)*dx[0]-400*x[0]*dx[1]
        H_dx[1] = -400*x[0]*dx[0] + 200*dx[1]

# Define a perfect preconditioner for the Hessian
class RosenHInv(Optizelle.Operator):
    def eval(self,state,dx,result):
        x = state.x
        one_over_det=1./(80000.*sq(x[0])-80000.*x[1]+400.)
        result[0]=one_over_det*(200.*dx[0]+400.*x[0]*dx[1])
        result[1]=(one_over_det*
            (400.*x[0]*dx[0]+(1200.*x[0]*x[0]-400.*x[1]+2.)*dx[1]))

#---Messaging0---
# Define a custom messaging object
def mymessaging(msg):
    """Prints out normal diagnostic information"""
    sys.stdout.write("PRINT: %s\n" %(msg))
#---Messaging1---

#---Serialization0---
def serialize_MyVS(x,name,iter):
    """Serializes an array for the vector space MyVS"""

    # Create the json representation
    x_json="[ "
    for i in xrange(0,len(x)):
        x_json += str(x[i]) + ", "
```

```
    x_json=x_json[0:-2]
    x_json +=" ]"

    return x_json

def deserialize_MyVS(x,x_json):
    """Deserializes an array for the vector space MyVS"""

    # Eliminate all whitespace
    x_json="".join(x_json.split())

    # Check if we're a vector
    if x_json[0:1]!="[" or x_json[-1:]!="]":
        raise TypeError("Attempted to deserialize a non-array vector.")

    # Eliminate the initial and final delimiters
    x_json=x_json[1:-1]

    # Create a list of the numbers involved
    x_json=x_json.split(",")

    # Convert the strings to numbers
    x_json=map(lambda x:float(x),x_json)

    # Create a MyVS vector
    return array.array('d',x_json)

# Register the serialization routines for arrays
def MySerialization():
    Optizelle.json.Serialization.serialize.register(
        serialize_MyVS,array.array)
    Optizelle.json.Serialization.deserialize.register(
        deserialize_MyVS,array.array)
#---Serialization1---

#---RestartManipulator0---
# Define a state manipulator that writes out the optimization state at
# each iteration.
class MyRestartManipulator(Optizelle.StateManipulator):
    def eval(self,fns,state,loc):
        # At the end of the optimization iteration, write the restart file
        if loc == Optizelle.OptimizationLocation.EndOfOptimizationIteration:
            # Create a reasonable file name
            ss = "rosenbrock_advanced_api_%04d.json" % (state.iter)

            # Write the restart file
            Optizelle.json.Unconstrained.write_restart(MyVS,ss,state)
#---RestartManipulator1---

# Register the serialization routines
MySerialization()

# Read in the name for the input file
if not(len(sys.argv)==2 or len(sys.argv)==3):
    sys.exit("python rosenbrock_advanced_api.py <parameters>\n" +
             "python rosenbrock_advanced_api.py <parameters> <restart>")
pname = sys.argv[1]
```

```
rname = sys.argv[2] if len(sys.argv)==3 else ""

# Generate an initial guess for Rosenbrock
x = array.array('d',[-1.2,1.0])

# Create an unconstrained state based on this vector
state=Optizelle.Unconstrained.State.t(MyVS,x)

#---ReadRestart0---
# If we have a restart file, read in the parameters
if len(sys.argv)==3:
    Optizelle.json.Unconstrained.read_restart(MyVS,rname,x,state)

# Read additional parameters from file
Optizelle.json.Unconstrained.read(MyVS,pname,state)
#---ReadRestart1---

# Create the bundle of functions
fns=Optizelle.Unconstrained.Functions.t()
fns.f=Rosenbrock()
fns.PH=RosenHInv()

#---Solver0---
# Solve the optimization problem
Optizelle.Unconstrained.Algorithms.getMin(
    MyVS,mymessaging,fns,state,MyRestartManipulator())
#---Solver1---

# Print out the reason for convergence
print("The algorithm converged due to: %s" % (
    Optizelle.OptimizationStop.to_string(state.opt_stop)))

# Print out the final answer
print("The optimal point is: (%e,%e)" % (state.x[0],state.x[1]))

#---WriteRestart0---
# Write out the final answer to file
Optizelle.json.Unconstrained.write_restart(MyVS,"solution.json",state)
#---WriteRestart1---
```

**Language** MATLAB/Octave

**Code**

```
% In this example, we duplicate the Rosenbrock example while demonstrating
% some of the more advanced API features such as custom vector spaces,
% messaging objects, and restarts.

function rosenbrock_advanced_api(pname,rname)
    % Read in the name for the input file
    if ~(nargin==1 || nargin==2)
        error(sprintf('%s\n%s', ...
            'rosenbrock_advanced_api(parameters)\n', ...
            'rosenbrock_advanced_api(parameters,restart)'));
    end

    % Execute the optimization
    if nargin==1
```

```
        main(pname);
    else
        main(pname,rname);
    end

end

%---VectorSpace0---
% Convert a vector to structure
function y = tostruct(x)
    y = struct('data',x);
end

% Defines the vector space used for optimization.
function self = MyVS()

    % Memory allocation and size setting
    self.init = @(x) x;

    % <- x (Shallow. No memory allocation.)
    self.copy = @(x) x;

    % <- alpha * x
    self.scal = @(alpha,x) tostruct(alpha*x.data);

    % <- 0
    self.zero = @(x) tostruct(zeros(size(x.data)));

    % <- alpha * x + y
    self.axpy = @(alpha,x,y) tostruct(alpha * x.data + y.data);

    %<- <x,y>
    self.innr = @(x,y)x.data'*y.data;

    % <- random
    self.rand = @(x)tostruct(randn(size(x.data)));

    % Jordan product, z <- x o y.
    self.prod = @(x,y)tostruct(x.data .* y.data);

    % Identity element, x <- e such that x o e = x.
    self.id = @(x)tostruct(ones(size(x.data)));

    % Jordan product inverse, z <- inv(L(x)) y where L(x) y = x o y.
    self.linv = @(x,y)tostruct(y.data ./ x.data);

    % Barrier function, barr <- barr(x) where x o grad barr(x) = e.
    self.barr = @(x)sum(log(x.data));

    % Line search, srch <- argmax {alpha \in Real >= 0 : alpha x + y >= 0}
    % where y > 0.
    self.srch = @(x,y) feval(@(z)min([min(z(find(z>0)));inf]),-y.data ./x.data);

    % Symmetrization, x <- symm(x) such that L(symm(x)) is a symmetric
    % operator.
    self.symm = @(x)x;
end
```

```
%---VectorSpace1---

% Squares its input
function z = sq(x)
    z=x*x;
end

% Define the Rosenbrock function where
%
% f(x,y)=(1-x)^2+100(y-x^2)^2
%
function self = Rosenbrock()

    % Evaluation of the Rosenbrock function
    self.eval = @(x) feval(@(x)sq(1.-x(1))+100.*sq(x(2)-sq(x(1))),x.data);

    % Gradient
    self.grad = @(x) tostruct(feval(@(x)[
        -400.*x(1)*(x(2)-sq(x(1)))-2.*(1.-x(1));
        200.*(x(2)-sq(x(1)))],x.data));

    % Hessian-vector product
    self.hessvec = @(x,dx) tostruct(feval(@(x,dx)[
        (1200.*sq(x(1))-400.*x(2)+2)*dx(1)-400.*x(1)*dx(2);
        -400.*x(1)*dx(1)+200.*dx(2)],x.data,dx.data));
end

% Define a perfect preconditioner for the Hessian
function self = RosenHInv()
    self.eval = @(state,dx) eval(state,dx);
end
function result = eval(state,dx)
    x = state.x.data;
    dx = dx.data;
    one_over_det=1./(80000.*sq(x(1))-80000.*x(2)+400.);
    result = tostruct([
        one_over_det*(200.*dx(1)+400.*x(1)*dx(2));
        one_over_det*...
            (400.*x(1)*dx(1)+(1200.*x(1)*x(1)-400.*x(2)+2.)*dx(2))]);
end

%---Messaging0---
% Define a custom messaging object
function MyMessaging(msg)
    fprintf('PRINT: %s\n',msg);
end
%---Messaging1---

%---Serialization0---
% Define serialization routines for MyVS
function MySerialization()
    global Optizelle;
    Optizelle.json.Serialization.serialize( ...
        'register', ...
        @(x,name,iter)strrep(mat2str(x.data'),' ',', '), ...
        @(x)isstruct(x) && isfield(x,'data') && isvector(x.data));
    Optizelle.json.Serialization.deserialize( ...
```

```
        'register', ...
        @(x,x_json)tostruct(str2num(x_json)'), ...
        @(x)isstruct(x) && isfield(x,'data') && isvector(x.data));
end
%---Serialization1---

%---RestartManipulator0---
% Define a state manipulator that writes out the optimization state at
% each iteration.
function smanip=MyRestartManipulator()
    smanip=struct('eval',@(fns,state,loc)MyRestartManipulator_(fns,state,loc));
end
function state=MyRestartManipulator_(fns,state,loc)
    global Optizelle;

    % At the end of the optimization iteration, write the restart file
    if(loc == Optizelle.OptimizationLocation.EndOfOptimizationIteration)
        % Create a reasonable file name
        ss = sprintf('rosenbrock_advanced_api_%04d.json',state.iter);

        % Write the restart file
        Optizelle.json.Unconstrained.write_restart(MyVS(),ss,state);
    end
end
%---RestartManipulator1---

% Actually runs the program
function main(pname,rname)

    % Grab the Optizelle library
    global Optizelle;
    setupOptizelle();

    % Register the serialization routines
    MySerialization();

    % Generate an initial guess for Rosenbrock
    x = tostruct([-1.2;1.]);

    % Create an unconstrained state based on this vector
    state=Optizelle.Unconstrained.State.t(MyVS(),x);

    %---ReadRestart0---
    % If we have a restart file, read in the parameters
    if(nargin==2)
        state = Optizelle.json.Unconstrained.read_restart(MyVS(),rname,x);
    end

    % Read additional parameters from file
    state=Optizelle.json.Unconstrained.read(MyVS(),pname,state);
    %---ReadRestart1---

    % Create the bundle of functions
    fns=Optizelle.Unconstrained.Functions.t;
    fns.f=Rosenbrock();
    fns.PH=RosenHInv();
```

```
    %---Solver0---
    % Solve the optimization problem
    state=Optizelle.Unconstrained.Algorithms.getMin( ...
        MyVS(),@MyMessaging,fns,state,MyRestartManipulator());
    %---Solver1---

    % Print out the reason for convergence
    fprintf('The algorithm converged due to: %s\n', ...
        Optizelle.OptimizationStop.to_string(state.opt_stop));

    % Print out the final answer
    fprintf('The optimal point is: (%e,%e)\n',state.x.data(1),state.x.data(2));

    %---WriteRestart0---
    % Write out the final answer to file
    Optizelle.json.Unconstrained.write_restart(MyVS(),'solution.json',state);
    %---WriteRestart1---
end
```

## 7.5 Simple constrained advanced API

In our Simple constrained advanced API example, we optimize the formulation

$$\begin{array}{ll} \min\limits_{x\in\mathbb{R}^2} & (x+1)^2+(y+1)^2 \\ \text{st} & x+2y=1 \\ & 2x+y\geq 1 \end{array}$$

which uses the same formulation as our example Simple constrained. It differs in that we implement a restart mechanism that writes our variables to an external file. By using the features described in our chapter Advanced API, we accomplish this with the code:

**Language** C++

**Code**

```
// Optimize a simple optimization problem with an optimal
// solution of (1/3,1/3)

#include "optizelle/optizelle.h"
#include "optizelle/vspaces.h"
#include "optizelle/json.h"
#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <cstring>

// Grab Optizelle's Natural type
using Optizelle::Natural;

// Defines the vector space used for optimization.
template <typename Real>
struct MyVS {
    typedef std::vector <Real> Vector;

    // Memory allocation and size setting
    static Vector init(Vector const & x) {
        return std::move(Vector(x.size()));
    }
```

```
// y <- x (Shallow. No memory allocation.)
static void copy(Vector const & x, Vector & y) {
    for(Natural i=0;i<x.size();i++){
        y[i]=x[i];
    }
}

// x <- alpha * x
static void scal(Real const & alpha, Vector & x) {
    for(Natural i=0;i<x.size();i++){
        x[i]=alpha*x[i];
    }
}

// x <- 0
static void zero(Vector & x) {
    for(Natural i=0;i<x.size();i++){
        x[i]=0.;
    }
}

// y <- alpha * x + y
static void axpy(Real const & alpha, Vector const & x, Vector & y) {
    for(Natural i=0;i<x.size();i++){
        y[i]=alpha*x[i]+y[i];
    }
}

// innr <- <x,y>
static Real innr(Vector const & x,Vector const & y) {
    Real z=0;
    for(Natural i=0;i<x.size();i++)
        z+=x[i]*y[i];
    return z;
}

// x <- random
static void rand(Vector & x){
    std::mt19937 gen(1);
    std::uniform_real_distribution<Real> dis(Real(0.),Real(1.));
    for(Natural i=0;i<x.size();i++)
        x[i]=Real(dis(gen));
}
// Jordan product, z <- x o y.
static void prod(Vector const & x, Vector const & y, Vector & z) {
    for(Natural i=0;i<x.size();i++)
        z[i]=x[i]*y[i];
}

// Identity element, x <- e such that x o e = x.
static void id(Vector & x) {
    for(Natural i=0;i<x.size();i++)
        x[i]=Real(1.);
}

// Jordan product inverse, z <- inv(L(x)) y where L(x) y = x o y.
static void linv(Vector const & x,Vector const & y,Vector & z) {
```

```
        for(Natural i=0;i<x.size();i++)
            z[i]=y[i]/x[i];
    }

    // Barrier function, barr <- barr(x) where x o grad barr(x) = e.
    static Real barr(Vector const & x) {
        Real z=Real(0.);
        for(Natural i=0;i<x.size();i++)
            z+=log(x[i]);
        return z;
    }

    // Line search, srch <- argmax {alpha \in Real >= 0 : alpha x + y >= 0}
    // where y > 0.
    static Real srch(Vector const & x,Vector const & y) {
        // Line search parameter
        Real alpha=std::numeric_limits <Real>::infinity();

        // Search for the optimal linesearch parameter.
        for(Natural i=0;i<x.size();i++) {
            if(x[i] < Real(0.)) {
                Real alpha0 = -y[i]/x[i];
                alpha = alpha0 < alpha ? alpha0 : alpha;
            }
        }

        return alpha;
    }

    // Symmetrization, x <- symm(x) such that L(symm(x)) is a symmetric
    // operator.
    static void symm(Vector & x) { }
};

// Squares its input
template <typename Real>
Real sq(Real const & x){
    return x*x;
}

// Define a simple objective where
//
// f(x,y)=(x+1)^2+(y+1)^2
//
struct MyObj : public Optizelle::ScalarValuedFunction <double,MyVS> {
    typedef MyVS <double> X;

    // Evaluation
    double eval(const X::Vector& x) const {
        return sq(x[0]+1.)+sq(x[1]+1.);
    }

    // Gradient
    void grad(
        X::Vector const & x,
        X::Vector & grad
    ) const {
```

```
        grad[0]=2*x[0]+2;
        grad[1]=2*x[1]+2;
    }

    // Hessian-vector product
    void hessvec(
        X::Vector const & x,
        X::Vector const & dx,
        X::Vector & H_dx
    ) const {
        H_dx[0]=2.*dx[0];
        H_dx[1]=2.*dx[1];
    }
};

// Define a simple equality
//
// g(x,y)= [ x + 2y = 1 ]
//
struct MyEq :public Optizelle::VectorValuedFunction<double,MyVS,MyVS> {
    typedef MyVS <double> X;
    typedef MyVS <double> Y;

    // y=g(x)
    void eval(
        X::Vector const & x,
        Y::Vector & y
    ) const {
        y[0]=x[0]+2.*x[1]-1.;
    }

    // y=g'(x)dx
    void p(
        X::Vector const & x,
        X::Vector const & dx,
        Y::Vector & y
    ) const {
        y[0]= dx[0]+2.*dx[1];
    }

    // xhat=g'(x)*dy
    void ps(
        X::Vector const & x,
        Y::Vector const & dy,
        X::Vector & xhat
    ) const {
        xhat[0]= dy[0];
        xhat[1]= 2.*dy[0];
    }

    // xhat=(g''(x)dx)*dy
    void pps(
        X::Vector const & x,
        X::Vector const & dx,
        Y::Vector const & dy,
        X::Vector & xhat
    ) const {
```

```
        X::zero(xhat);
    }
};

// Define a simple inequality
//
// h(x,y)= [ 2x + y >= 1 ]
//
struct MyIneq :public Optizelle::VectorValuedFunction<double,MyVS,MyVS> {
    typedef MyVS <double> X;
    typedef MyVS <double> Z;

    // z=h(x)
    void eval(
        X::Vector const & x,
        Z::Vector & z
    ) const {
        z[0]=2.*x[0]+x[1]-1.;
    }

    // z=h'(x)dx
    void p(
        X::Vector const & x,
        X::Vector const & dx,
        Z::Vector & z
    ) const {
        z[0]= 2.*dx[0]+dx[1];
    }

    // xhat=h'(x)*dz
    void ps(
        X::Vector const & x,
        Z::Vector const & dz,
        X::Vector & xhat
    ) const {
        xhat[0]= 2.*dz[0];
        xhat[1]= dz[0];
    }

    // xhat=(h''(x)dx)*dz
    void pps(
        X::Vector const & x,
        X::Vector const & dx,
        Z::Vector const & dz,
        X::Vector & xhat
    ) const {
        X::zero(xhat);
    }
};

//---Serialization0---
// Define serialization routines for MyVS
namespace Optizelle {
    namespace json {
        template <>
        struct Serialization <double,MyVS> {
            static std::string serialize(
```

```
    typename MyVS <double>::Vector const & x,
    std::string const & name,
    Natural const & iter
) {
    // Create the filename where we put our vector
    std::stringstream fname;
    fname << "./restart/";
    fname << name << ".";
    fname << std::setw(4) << std::setfill('0') << iter;
    fname << ".txt";

    // Actually write the vector there
    std::ofstream fout(fname.str());
    if(fout.fail()) {
        std::stringstream msg;
        msg << "While writing the variable " << name
            << " to file on iteration " << iter
            << ", unable to open the file: "
            << fname.str() << ".";
        throw Optizelle::Exception::t(msg.str());
    }
    fout.setf(std::ios::scientific);
    fout.precision(16);
    for(Natural i=0;i<x.size();i++)
        fout << x[i] << std::endl;

    // Close out the file
    fout.close();

    // Use this filename as the json string
    std::stringstream x_json;
    x_json << "\"" << fname.str() << "\"";
    return x_json.str();
}
static MyVS <double>::Vector deserialize(
    typename MyVS <double>::Vector const & x_,
    std::string const & x_json_
) {
    // Make a copy of x_json_
    auto x_json = x_json_;

    // Filter out the quotes and newlines from the string
    auto formatting = "\"\n";
    for(auto i=0;i<2;i++)
        x_json.erase(
            std::remove(x_json.begin(),x_json.end(),formatting[i]),
            x_json.end());

    // Open the file for reading
    std::ifstream fin(x_json.c_str());
    if(!fin.is_open())
        throw Optizelle::Exception::t(
            "Error while opening the file " + x_json + ": " +
            strerror(errno));

    // Create a new vector that we eventually return
    auto x = std::vector <double> (x_.size());
```

```
                // Read in each of the elements
                for(auto i=0;i<x.size();i++)
                    fin >> x[i];

                // Return the result
                return std::move(x);
            }
        };
    }
}
//---Serialization1---

// Define a state manipulator that writes out the optimization state at
// each iteration.
struct MyRestartManipulator : Optizelle::StateManipulator <
    Optizelle::Constrained <double,MyVS,MyVS,MyVS> >
{
    void eval(
        typename Optizelle::Constrained <double,MyVS,MyVS,MyVS>
            ::Functions::t const & fns,
        typename Optizelle::Constrained <double,MyVS,MyVS,MyVS>
            ::State::t & state,
        Optizelle::OptimizationLocation::t const & loc
    ) const {
        switch(loc) {
        // At the end of the optimization iteration, write the restart file
        case Optizelle::OptimizationLocation::EndOfOptimizationIteration: {
            // Create a reasonable file name
            std::stringstream ss;
            ss << "simple_constrained_advanced_api_";
            ss << std::setw(4) << std::setfill('0') << state.iter;
            ss << ".json";

            // Write the restart file
            Optizelle::json::Constrained <double,MyVS,MyVS,MyVS>::write_restart(
                ss.str(),state);
            break;
        } default:
            break;
        }
    }
};

int main(int argc,char* argv[]){
    // Read in the name for the parameters and optional restart file
    if(!(argc==2 || argc==3)) {
        std::cerr << "simple_constrained_advanced_api <parameters>"<< std::endl;
        std::cerr << "simple_constrained_advanced_api <parameters> <restart>"
            << std::endl;
        exit(EXIT_FAILURE);
    }
    auto pname = argv[1];
    auto rname = argc==3 ? argv[2] : "";

    // Generate an initial guess for the primal
    auto x = std::vector <double> {2.1, 1.1};
```

```
    // Allocate memory for equality multiplier
    auto y = std::vector <double> (1);

    // Allocate memory for the inequality multiplier
    auto z = std::vector <double> (1);

    // Create an optimization state
    Optizelle::Constrained <double,MyVS,MyVS,MyVS>::State::t state(x,y,z);

    // If we have a restart file, read in the parameters
    if(argc==3)
        Optizelle::json::Constrained <double,MyVS,MyVS,MyVS>::read_restart(
            rname,x,y,z,state);

    // Read the parameters from file
    Optizelle::json::Constrained <double,MyVS,MyVS,MyVS>::read(pname,state);

    // Create a bundle of functions
    Optizelle::Constrained <double,MyVS,MyVS,MyVS>::Functions::t fns;
    fns.f.reset(new MyObj);
    fns.g.reset(new MyEq);
    fns.h.reset(new MyIneq);

    // Solve the optimization problem
    Optizelle::Constrained <double,MyVS,MyVS,MyVS>::Algorithms
        ::getMin(Optizelle::Messaging::stdout,fns,state,MyRestartManipulator());

    // Print out the reason for convergence
    std::cout << "The algorithm converged due to: " <<
        Optizelle::OptimizationStop::to_string(state.opt_stop) <<
        std::endl;

    // Print out the final answer
    std::cout << std::scientific << std::setprecision(16)
        << "The optimal point is: (" << state.x[0] << ','
        << state.x[1] << ')' << std::endl;

    // Write out the final answer to file
    Optizelle::json::Constrained <double,MyVS,MyVS,MyVS>::write_restart(
        "solution.json",state);

    // Successful termination
    return EXIT_SUCCESS;
}
```

**Language** Python

**Code**

```
# Optimize a simple optimization problem with an optimal solution
# of (1/3,1/3)

import Optizelle
import sys
import copy
import array
import math
```

```python
# Defines the vector space used for optimization.
class MyVS(object):
    @staticmethod
    def init(x):
        """Memory allocation and size setting"""
        return copy.deepcopy(x)

    @staticmethod
    def copy(x,y):
        """y <- x (Shallow. No memory allocation.)"""
        y[:]=x[:]

    @staticmethod
    def scal(alpha,x):
        """x <- alpha * x"""
        for i in xrange(0,len(x)):
            x[i]=alpha*x[i]

    @staticmethod
    def zero(x):
        """x <- 0"""
        for i in xrange(0,len(x)):
            x[i]=0.

    @staticmethod
    def axpy(alpha,x,y):
        """y <- alpha * x + y"""
        for i in xrange(0,len(x)):
            y[i]=alpha*x[i]+y[i]

    @staticmethod
    def innr(x,y):
        """<- <x,y>"""
        return reduce(lambda z,xy:xy[0]*xy[1]+z,zip(x,y),0.)

    @staticmethod
    def rand(x):
        """x <- random"""
        for i in xrange(0,len(x)):
            x[i]=random.uniform(0.,1.)

    @staticmethod
    def prod(x,y,z):
        """Jordan product, z <- x o y"""
        for i in xrange(0,len(x)):
            z[i]=x[i]*y[i]

    @staticmethod
    def id(x):
        """Identity element, x <- e such that x o e = x"""
        for i in xrange(0,len(x)):
            x[i]=1.

    @staticmethod
    def linv(x,y,z):
        """Jordan product inverse, z <- inv(L(x)) y where L(x) y = x o y"""
```

```
        for i in xrange(0,len(x)):
            z[i]=y[i]/x[i]

    @staticmethod
    def barr(x):
        """Barrier function, <- barr(x) where x o grad barr(x) = e"""
        return reduce(lambda x,y:x+math.log(y),x,0.)

    @staticmethod
    def srch(x,y):
        """Line search, <- argmax {alpha \in Real >= 0 : alpha x + y >= 0} where y >
        alpha = float("inf")
        for i in xrange(0,len(x)):
            if x[i] < 0:
                alpha0 = -y[i]/x[i]
                if alpha0 < alpha:
                    alpha=alpha0
        return alpha

    @staticmethod
    def symm(x):
        """Symmetrization, x <- symm(x) such that L(symm(x)) is a symmetric operator
        pass

# Squares its input
sq = lambda x:x*x

# Define a simple objective where
#
# f(x,y)=(x+1)^2+(y+1)^2
#
class MyObj(Optizelle.ScalarValuedFunction):

    # Evaluation
    def eval(self,x):
        return sq(x[0]+1.)+sq(x[1]+1.)

    # Gradient
    def grad(self,x,grad):
        grad[0]=2.*x[0]+2.
        grad[1]=2.*x[1]+2.

    # Hessian-vector product
    def hessvec(self,x,dx,H_dx):
        H_dx[0]=2.*dx[0]
        H_dx[1]=2.*dx[1]

# Define a simple equality
#
# g(x,y)= [ x + 2y = 1 ]
#
class MyEq(Optizelle.VectorValuedFunction):

    # y=g(x)
    def eval(self,x,y):
        y[0]=x[0]+2.*x[1]-1.
```

```
    # y=g'(x)dx
    def p(self,x,dx,y):
        y[0]= dx[0]+2.*dx[1]

    # xhat=g'(x)*dy
    def ps(self,x,dy,xhat):
        xhat[0]= dy[0]
        xhat[1]= 2.*dy[0]

    # xhat=(g''(x)dx)*dy
    def pps(self,x,dx,dy,xhat):
        MyVS.zero(xhat)

# Define simple inequalities
#
# h(x,y)= [ 2x + y >= 1 ]
#
class MyIneq(Optizelle.VectorValuedFunction):

    # z=h(x)
    def eval(self,x,z):
        z[0]=2.*x[0]+x[1]-1.

    # z=h'(x)dx
    def p(self,x,dx,z):
        z[0]= 2.*dx[0]+dx[1]

    # xhat=h'(x)*dz
    def ps(self,x,dz,xhat):
        xhat[0]= 2.*dz[0]
        xhat[1]= dz[0]

    # xhat=(h''(x)dx)*dz
    def pps(self,x,dx,dz,xhat):
        MyVS.zero(xhat)

#---Serialization0---
def serialize_MyVS(x,name,iter):
    """Serializes an array for the vector space MyVS"""

    # Create the filename where we put our vector
    fname = "./restart/%s.%04d.txt" % (name,iter)

    # Actually write the vector there
    fout = open(fname,"w");
    for i in xrange(0,len(x)):
        fout.write("%1.16e\n" % x[i])

    # Close out the file
    fout.close()

    # Use this filename as the json string
    x_json = "\"%s\"" % fname
    return x_json

def deserialize_MyVS(x_,x_json):
    """Deserializes an array for the vector space MyVS"""
```

```
    # Eliminate all whitespace
    x_json="".join(x_json.split())

    # Eliminate the initial and final delimiters
    x_json=x_json[1:-1]

    # Open the file for reading
    fin = open(x_json,"r")

    # Allocate a new vector to return
    x = copy.deepcopy(x_)

    # Read in each of the elements
    for i in xrange(0,len(x)):
        x[i] = float(fin.readline())

    # Close out the file
    fin.close()

    # Return the result
    return x

# Register the serialization routines for arrays
def MySerialization():
    Optizelle.json.Serialization.serialize.register(
        serialize_MyVS,array.array)
    Optizelle.json.Serialization.deserialize.register(
        deserialize_MyVS,array.array)
#---Serialization1---

# Define a state manipulator that writes out the optimization state at
# each iteration.
class MyRestartManipulator(Optizelle.StateManipulator):
    def eval(self,fns,state,loc):
        # At the end of the optimization iteration, write the restart file
        if loc == Optizelle.OptimizationLocation.EndOfOptimizationIteration:
            # Create a reasonable file name
            ss = "simple_constrained_advanced_api_%04d.json" % (state.iter)

            # Write the restart file
            Optizelle.json.Constrained.write_restart(
               MyVS,MyVS,MyVS,ss,state)

# Register the serialization routines
MySerialization()

# Read in the name for the input file
if not(len(sys.argv)==2 or len(sys.argv)==3):
    sys.exit("python simple_constrained_advanced_api.py <parameters>\n" +
             "python simple_constrained_advanced_api.py <parameters> <restart>")
pname = sys.argv[1]
rname = sys.argv[2] if len(sys.argv)==3 else ""

# Generate an initial guess
x = array.array('d',[2.1,1.1])
```

```python
# Allocate memory for the equality multiplier
y = array.array('d',[0.])

# Allocate memory for the inequality multiplier
z = array.array('d',[0.])

# Create an optimization state
state=Optizelle.Constrained.State.t(MyVS,MyVS,MyVS,x,y,z)

# If we have a restart file, read in the parameters
if len(sys.argv)==3:
    Optizelle.json.Constrained.read_restart(MyVS,MyVS,MyVS,rname,x,y,z,state)

# Read the parameters from file
Optizelle.json.Constrained.read(MyVS,MyVS,MyVS,pname,state)

# Create a bundle of functions
fns=Optizelle.Constrained.Functions.t()
fns.f=MyObj()
fns.g=MyEq()
fns.h=MyIneq()

# Solve the optimization problem
Optizelle.Constrained.Algorithms.getMin(
    MyVS,MyVS,MyVS,Optizelle.Messaging.stdout,fns,state,MyRestartManipulator())

# Print out the reason for convergence
print("The algorithm converged due to: %s" % (
    Optizelle.OptimizationStop.to_string(state.opt_stop)))

# Print out the final answer
print("The optimal point is: (%e,%e)" % (state.x[0],state.x[1]))

# Write out the final answer to file
Optizelle.json.Constrained.write_restart(MyVS,MyVS,MyVS,"solution.json",state)
```

**Language** MATLAB/Octave

**Code**

```matlab
% Optimize a simple optimization problem with an optimal solution
% of (1/3,1/3)

function simple_constrained_advanced_api(pname,rname)
    % Read in the name for the input file
    if ~(nargin==1 || nargin==2)
        error(sprintf('%s\n%s', ...
            'simple_constrained_advanced_api(parameters)\n', ...
            'simple_constrained_advanced_api(parameters,restart)'));
    end

    % Execute the optimization
    if nargin==1
        main(pname);
    else
        main(pname,rname);
    end
end
```

```
% Convert a vector to structure
function y = tostruct(x)
    y = struct('data',x);
end

% Defines the vector space used for optimization.
function self = MyVS()

    % Memory allocation and size setting
    self.init = @(x) x;

    % <- x (Shallow. No memory allocation.)
    self.copy = @(x) x;

    % <- alpha * x
    self.scal = @(alpha,x) tostruct(alpha*x.data);

    % <- 0
    self.zero = @(x) tostruct(zeros(size(x.data)));

    % <- alpha * x + y
    self.axpy = @(alpha,x,y) tostruct(alpha * x.data + y.data);

    %<- <x,y>
    self.innr = @(x,y)x.data'*y.data;

    % <- random
    self.rand = @(x)tostruct(randn(size(x.data)));

    % Jordan product, z <- x o y.
    self.prod = @(x,y)tostruct(x.data .* y.data);

    % Identity element, x <- e such that x o e = x.
    self.id = @(x)tostruct(ones(size(x.data)));

    % Jordan product inverse, z <- inv(L(x)) y where L(x) y = x o y.
    self.linv = @(x,y)tostruct(y.data ./ x.data);

    % Barrier function, barr <- barr(x) where x o grad barr(x) = e.
    self.barr = @(x)sum(log(x.data));

    % Line search, srch <- argmax {alpha \in Real >= 0 : alpha x + y >= 0}
    % where y > 0.
    self.srch = @(x,y) feval(@(z)min([min(z(find(z>0)));inf]),-y.data ./x.data);

    % Symmetrization, x <- symm(x) such that L(symm(x)) is a symmetric
    % operator.
    self.symm = @(x)x;
end

% Squares its input
function z = sq(x)
    z=x*x;
end

% Define a simple objective where
```

```
%
% f(x,y)=(x+1)^2+(y+1)^2
%
function self = MyObj()

    % Evaluation
    self.eval = @(x) feval(@(x)sq(x(1)+1.)+sq(x(2)+1.),x.data);

    % Gradient
    self.grad = @(x) tostruct(feval(@(x)[
        2.*x(1)+2.;
        2.*x(2)+2.],x.data));

    % Hessian-vector product
    self.hessvec = @(x,dx) tostruct(feval(@(x,dx)[
        2.*dx(1);
        2.*dx(2)],x.data,dx.data));
end

% Define a simple equality
%
% g(x,y)= [ x + 2y = 1 ]
%
function self = MyEq()

    % y=g(x)
    self.eval = @(x) tostruct(feval(@(x)[x(1)+2.*x(2)-1.],x.data));

    % y=g'(x)dx
    self.p = @(x,dx) tostruct(feval(@(x,dx)[dx(1)+2.*dx(2)],x.data,dx.data));

    % xhat=g'(x)*dy
    self.ps = @(x,dy) tostruct(feval(@(x,dy)[
        dy(1);
        2.*dy(1)],x.data,dy.data));

    % xhat=(g''(x)dx)*dy
    self.pps = @(x,dx,dy) tostruct(zeros(2,1));
end

% Define simple inequalities
%
% h(x,y)= [ 2x + y >= 1 ]
%
function self = MyIneq()

    % z=h(x)
    self.eval = @(x) tostruct(feval(@(x)[
        2.*x(1)+x(2)-1],x.data));

    % z=h'(x)dx
    self.p = @(x,dx) tostruct(feval(@(x,dx)[
        2.*dx(1)+dx(2)],x.data,dx.data));

    % xhat=h'(x)*dz
    self.ps = @(x,dz) tostruct(feval(@(x,dz)[
        2.*dz(1)
```

```
        dz(1)],x.data,dz.data));

    % hat=(h''(x)dx)*dz
    self.pps = @(x,dx,dz) tostruct([ 0. ]);
end

%---Serialization0---
% Define the serialize routine for MyVS
function x_json=serialize_MyVS(x,name,iter)
    % Create the filename where we put our vector
    fname=sprintf('./restart/%s.%04d.txt',name,iter);

    % Actually write the vector there
    dlmwrite(fname,x.data);

    % Use this filename as the json string
    x_json = sprintf('\"%s\"',fname);
end

% Define the deserialize routine for MyVS
function x=deserialize_MyVS(x_,x_json)
    % Filter out the quotes and newlines from the string
    x_json = strrep(x_json,'"','');
    x_json = strrep(x_json,sprintf('\n'),'');

    % Read the data into x
    x=tostruct(dlmread(x_json));
end

% Define serialization routines for MyVS
function MySerialization()
    global Optizelle;
    Optizelle.json.Serialization.serialize( ...
        'register', ...
        @(x,name,iter)serialize_MyVS(x,name,iter), ...
        @(x)isstruct(x) && isfield(x,'data') && isvector(x.data));
    Optizelle.json.Serialization.deserialize( ...
        'register', ...
        @(x,x_json)deserialize_MyVS(x,x_json), ...
        @(x)isstruct(x) && isfield(x,'data') && isvector(x.data));
end
%---Serialization1---

% Define a state manipulator that writes out the optimization state at
% each iteration.
function smanip=MyRestartManipulator()
    smanip=struct('eval',@(fns,state,loc)MyRestartManipulator_(fns,state,loc));
end
function state=MyRestartManipulator_(fns,state,loc)
    global Optizelle;

    % At the end of the optimization iteration, write the restart file
    if(loc == Optizelle.OptimizationLocation.EndOfOptimizationIteration)
        % Create a reasonable file name
        ss = sprintf('simple_constrained_advanced_api_%04d.json',state.iter);

        % Write the restart file
```

```
        Optizelle.json.Constrained.write_restart( ...
            MyVS(),MyVS(),MyVS(),ss,state);
    end
end

% Actually runs the program
function main(pname,rname)

    % Grab the Optizelle library
    global Optizelle;
    setupOptizelle();

    % Register the serialization routines
    MySerialization();

    % Generate an initial guess
    x = tostruct([2.1;1.1]);

    % Allocate memory for the equality multiplier
    y = tostruct([0.]);

    % Allocate memory for the inequality multiplier
    z = tostruct([0.]);

    % Create an optimization state
    state = Optizelle.Constrained.State.t(MyVS(),MyVS(),MyVS(),x,y,z);

    % If we have a restart file, read in the parameters
    if(nargin==2)
        state = Optizelle.json.Constrained.read_restart( ...
            MyVS(),MyVS(),MyVS(),rname,x,y,z);
    end

    % Read the parameters from file
    state = Optizelle.json.Constrained.read(MyVS(),MyVS(),MyVS(),pname,state);

    % Create a bundle of functions
    fns = Optizelle.Constrained.Functions.t;
    fns.f = MyObj();
    fns.g = MyEq();
    fns.h = MyIneq();

    % Solve the optimization problem
    state = Optizelle.Constrained.Algorithms.getMin( ...
        MyVS(),MyVS(),MyVS(),Optizelle.Messaging.stdout,fns,state, ...
        MyRestartManipulator());

    % Print out the reason for convergence
    fprintf('The algorithm converged due to: %s\n', ...
        Optizelle.OptimizationStop.to_string(state.opt_stop));

    % Print out the final answer
    fprintf('The optimal point is: (%e,%e)\n',state.x.data(1),state.x.data(2));

    % Write out the final answer to file
    Optizelle.json.Constrained.write_restart( ...
        MyVS(),MyVS(),MyVS(),'solution.json',state);
```

```
end
```

# 8

# Algorithmic discussion

In the following chapter, we give a brief discussion of the algorithms we include within Optizelle and references to more detailed descriptions.

**Algorithm** Barzilai-Borwein

**Description** We implement the Barzilai-Borwein algorithm by setting `dir` to `SteepestDescent` and `kind` to either `TwoPointA` or `TwoPointB`. Specifically, `TwoPointA` and `TwoPointB` refer to the algorithms generated by equation (5) and (6) in Barzilai and Borwein's paper, respectively. Since this algorithm requires two points before it may commence, we use a `GoldenSection` search on the first iteration.

- Jonathan Barzilai and Jonathan M. Borwein. Two-point step size gradient methods. *IMA Journal of Numerical Analysis*, 8(1):141–148, 1988.

**Algorithm** Golden-section search

**Description** We implement a straightforward golden-section search. For historical significance, we refer to Kiefer's paper, but a much more complete description can be found in Bazaraa, Sherali, and Shetty's book.

- J. Kiefer. Sequential minimax search for a maximum. *Proceedings of the American Mathematical Society*, 4(3):502–506, 1953.
- Mokhtar S. Bazaraa, Hanif D. Sherali, and C. M. Shetty. *Nonlinear Programming: Theory And Algorithms.* Wiley-Interscience, 3rd edition, 2006.

**Algorithm** BFGS

**Description** Our BFGS implementation uses a limited-memory, iterative reformulation of the algorithm based on a generic inner product. Our limited-memory implementation differs from that of Byrd, Nocedal, and Schnabel's because we do not form a compact representation, but instead use a scratch space whose size is equal to `stored_history`. In addition, since we do not check the Wolfe conditions, we do a hard check to insure that BFGS operator remains positive definite. We refer to the collection of 1970s papers for historical significance, but note that a much better presentation of the algorithm can be found in Nocedal and Wright's book.

- C. G. Broyden. The convergence of a class of double-rank minimization algorithms: 2. the new algorithm. *IMA Journal of Applied Mathematics*, 6(3):222–231, 1970.
- R. Fletcher. A new approach to variable metric algorithms. *The Computer Journal*, 13(3):317–322, 1970.

- D. Goldfarb. A family of variable metric updates derived by variational means. *Mathematics of Computation*, 24:23–26, 1970.
- D. F. Shanno. Conditioning of quasi-Newton methods for function minimization. *Mathematics of Computation*, 24(111):647–656, 1970.
- Richard H. Byrd, Jorge Nocedal, and Robert B. Schnabel. Representations of quasi-Newton matrices and their use in limited memory methods. *Mathematical Programming*, 63(2):129–156, 1994.
- Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, 2nd edition, 2006.

**Algorithm** SR1

**Description** Similar to BFGS, our SR1 implementation uses a limited-memory, iterative reformulation of the algorithm based on a generic inner product. As before, our limited-memory implementation differs from that of Byrd, Nocedal, and Schnabel's because we do not form a compact representation, but instead use a scratch space whose size is equal to `stored_history`. We refer to Broyden's paper for historical significance, but note that a much better presentation of the algorithm can be found in Nocedal and Wright's book.

- C. G. Broyden. Quasi-Newton methods and their application to function minimization. *Mathematics of Computation*, 21:368–381, 1967.
- Richard H. Byrd, Jorge Nocedal, and Robert B. Schnabel. Representations of quasi-Newton matrices and their use in limited memory methods. *Mathematical Programming*, 63(2):129–156, 1994.
- Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, 2nd edition, 2006.

**Algorithm** Nonlinear-CG

**Description** We use a standard implementation of nonlinear-CG. On the first iteration, we move in the steepest descent direction, but use the specified nonlinear-CG direction on subsequent iterations. Since we do not check the strong-Wolfe condition, we do a hard check to insure a descent direction. If we do not, we negate the search direction. Although we reference the original papers from Hestenes and Stiefel, Fletcher and Reeves, and Polak and Ribiere, Nocedal and Wright give a nicer presentation. In addition, Hager and Zhang present a nice overview of the different nonlinear-CG variations in their survey paper.

- Magnus R. Hestenes and Eduard Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49(6):409–436, 1952.
- R. Fletcher and C. M. Reeves. Function minimization by conjugate gradients. *The Computer Journal*, 7(2):149–154, 1964.
- E. Polak and G. Ribiere. Note sur la convergence de méthodes de directions conjuguées. *Revue Française d'Informatique et de Recherche Opérationnelle*, 16:35–43, 1969.
- Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, 2nd edition, 2006.
- William W. Hager and Hongchao Zhang. A survey of nonlinear conjugate gradient methods. *Pacific Journal of Optimization*, 2(1):35–58, January 2006.

**Algorithm** Trust-region Newton

**Description** Our trust-region Newton implementation uses truncated-CG to solve the trust-region subproblem. Both Conn, Gould, and Toint's as well as Nocedal and Wright's book give good descriptions of the algorithm.

- Andrew R. Conn, Nicholas I. M. Gould, and Philippe L. Toint. *Trust-Region Methods*. SIAM, 2000.
- Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, 2nd edition, 2006.

**Algorithm** Newton-CG

**Description** We base our Newton-CG algorithm on truncated CG and not a Hessian modification. Nocedal and Wright's book describes this algorithm.

- Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, 2nd edition, 2006.

**Algorithm** Truncated CG

**Description** Our version of truncated CG actually possesses the ability to over orthogonalize against previous Krylov vectors, which is controlled by the parameters `trunc_orthog_storage_max` and `trunc_orthog_iter_max`. In addition, we have added a safeguard procedure for the interior point method that insures truncated CG always produces a solution feasible with respect to the inequality constraint. This safeguard process is similar to the one used by Byrd, Hribar, and Nocedal in their NITRO algorithm. Finally, for the inexact composite-step SQP method, we use the heuristic described in appendix B by Heinkenschloss and Ridzal to detect instability in the algorithm. Historically, Toint and Steihaug give a description of truncated-CG in their respective papers. For a modern treatment of truncated-CG see Conn, Gould, and Toint's book.

- Ph. L. Toint. Towards an efficient sparsity exploiting Newton method for minimization. pages 57–88. 1981.
- Trond Steihaug. The conjugate gradient method and trust regions in large scale optimization. *SIAM Journal on Numerical Analysis*, 20(3):626–637, 1983.
- Andrew R. Conn, Nicholas I. M. Gould, and Philippe L. Toint. *Trust-Region Methods*. SIAM, 2000.
- Richard H. Byrd, Mary E. Hribar, and Jorge Nocedal. An interior point algorithm for large-scale nonlinear programming. *SIAM Journal on Optimization*, 9(4):877–900, 1999.
- Matthias Heinkenschloss and Denis Ridzal. A matrix-free trust-region sqp method for equality constrained optimization. *SIAM Journal on Optimization*, 24(3):1507–1541, 2014.

**Algorithm** Interior-point method

**Description** Our interior point method is based on a new derivation of the primal-dual interior point equations based on pseudo-Euclidean-Jordan algebras. We say *pseudo* because we do not require commutativity in the Jordan product. Specifically, our derivation begins from the optimality conditions

$$\begin{aligned}
\nabla f(x) - h'(x)^* z =& 0,\\
h(x) \succeq& 0,\\
z \succeq& 0,\\
h(x) \circ z =& 0
\end{aligned}$$

in the case of inequality constrained problems and

$$\begin{aligned}\nabla f(x) + g'(x)^* y - h'(x)^* z =& 0,\\ g(x) =& 0,\\ h(x) \succeq& 0,\\ z \succeq& 0,\\ h(x) \circ z =& 0\end{aligned}$$

in the case of constrained problems. Here, $\circ$ denotes the Jordan product that we refer to as `prod`. Since we use a composite-step SQP method for constrained problems, we ignore the feasibility condition, $g(x) = 0$, in the constrained problem. Simply, we handle feasibility with respect to this constraint with the quasi-normal step. This allows us to reduce both sets of optimality conditions to

$$\begin{aligned}\operatorname{grad}(x,y) - h'(x)^* z =& 0,\\ h(x) \succeq& 0,\\ z \succeq& 0,\\ h(x) \circ z =& 0\end{aligned}$$

where

$$\operatorname{grad}(x,y) = \begin{cases} \nabla f(x) & \text{Inequality constrained problems,}\\ \nabla f(x) + g'(x)^* y & \text{Constrained problems.}\end{cases}$$

Then, using a standard interior-point formulation, we perturb the optimality conditions into

$$\begin{aligned}\operatorname{grad}(x,y) - h'(x)^* z =& 0\\ h(x) \succ& 0\\ z \succ& 0\\ h(x) \circ z =& \mu e.\end{aligned}$$

where $e$ denotes the identity element in the pseudo-Euclidean-Jordan algebra, which we refer to as `id`. Next, we apply Newton's method to the nonlinear system of equations

$$\begin{aligned}\operatorname{grad}(x,y) - h'(x)^* z =& 0,\\ h(x) \circ z =& \mu e,\end{aligned}$$

which yields the system

$$\begin{bmatrix}\operatorname{hess}(x,y) & -h'(x)^*\\ h'(x)\cdot \circ z & h(x)\circ \cdot\end{bmatrix}\begin{bmatrix}\delta x\\ \delta z\end{bmatrix} = \begin{bmatrix}-\operatorname{grad}(x,y) + h'(x)^* z\\ -h(x)\circ z + \mu e\end{bmatrix}$$

where

$$\operatorname{hess}(x,y) = \begin{cases} \nabla^2 f(x) & \text{Inequality constrained problems,}\\ \nabla^2 f(x) + (g''(x)\cdot)^* y & \text{Constrained problems.}\end{cases}$$

Using the second equation in the Newton system, we solve for $\delta z$ and find that

$$\delta z = -z + L(h(x))^{-1}(-h'(x)\delta x \circ z + \mu e)$$

where $L(h(x))^{-1}$ denotes the inverse of the linear operator induced by the Jordan product, $\circ$, which we refer to as `linv`. In other words, $h(x)\circ z = L(h(x))z$. Then, we plug this equation into the first equation and reduce our Newton system to

$$[\operatorname{hess}(x,y) + h'(x)^*(L(h(x))^{-1}(h'(x)\cdot \circ z))]\delta x = -\operatorname{grad}(x,y) + \mu h'(x)^*(L(h(x))^{-1} e).$$

At this point, we solve the Newton system using truncated CG. As a note, when using a line-search method that is not Newton-CG, we using a different scheme and instead set

$$\texttt{z} = \texttt{mu} \cdot L(h(\texttt{x}))^{-1} e.$$

This gives us a log-barrier algorithm for these methods. In short, without solving a Newton system, the equations for `dz` don't make sense, so we instead fallback on a log-barrier method, which does not require them. In order to maintain strict feasibility of $h(x)$ and $z$ we safeguard our steps `dx` and `dz` using the fraction to the boundary rule

$$\begin{aligned} h(\texttt{x} + \texttt{alpha\_x} \cdot \texttt{dx}) \geq& (1 - \texttt{gamma})h(x) \\ \texttt{z} + \texttt{alpha\_z} \cdot \texttt{dz} \geq& (1 - \texttt{gamma})z) \\ h(\texttt{x} + \texttt{alpha\_x\_qn} \cdot \texttt{dx\_n}) \geq& (1 - \texttt{gamma} \cdot \texttt{zeta})h(x) \end{aligned}$$

Note, the last inequality only occurs in constrained problems. When we enforce these rules depends on the algorithm. Specifically, trust-region methods enforce these bounds during the truncated-CG solve of the optimality conditions. Since truncated CG may violate the inequality bounds periodically throughout the optimality solve, we save the last feasible iterate during the computation. When we exit, we take the last feasible iterate and step and compute the safeguard search, which satisfies the fraction to the boundary rule above. In order to prevent too many discarded steps due to the safeguard, we limit the maximum number of infeasible steps that we allow to be `safeguard_failed_max`. Although our process is slightly different than their paper, how we embed the safeguard into truncated CG is similar to what Byrd, Hribar, and Nocedal do in their implementation of NITRO. In a line-search method, we safeguard the step prior to the line search. Specifically, we shorten `alpha0` so that the maximum step length taken by the line search does not exceed our fraction to the boundary rule. Finally, in the inexact composite step SQP method, we also safeguard our quasi-normal step by enforcing the fraction to the boundary rule during the dogleg computation. In each case, we calculate the distance to the boundary with the user-defined function `srch`. In our `Rm` and `SQL` vector spaces, we use a closed form formula for linear and second-order cones and the Arnoldi algorithm for semidefinite cones. We reduce `mu` prior to the truncated-CG solve for the optimality system and set $\texttt{mu} = \texttt{sigma} \cdot \texttt{mu}$ when one of the following global or local convergence criteria is satisfied

1. $\log(\texttt{norm\_gradtyp}) - \log(\|\text{gradstep}(x,y,z)\|) < \log(\texttt{mu\_typ}) - \log(\texttt{mu\_est})$
2. $\|\text{gradstep}(x,y,z)\| < \texttt{eps\_grad} \cdot \texttt{norm\_gradtyp}$
3. $\log(\texttt{norm\_gradtyp}) - \log(\|\text{gradstop}(x,y,z)\|) < \log(\texttt{mu\_typ}) - \log(\texttt{mu\_est})$
4. $\|\text{gradstop}(x,y,z)\| < \texttt{eps\_grad} \cdot \texttt{norm\_gradtyp}$

where

$$\text{gradstop}(x,y,z) = \begin{cases} \nabla f(x) - h'(x)^* z & \text{Inequality constrained,} \\ \nabla f(x) + g'(x)^* y - h'(x)^* z & \text{Constrained.} \end{cases}$$

$$\text{gradstep}(x,y,z) = \begin{cases} \nabla f(x) - \mu h'(x)^* L(h(x))^{-1} e & \text{Inequality constrai} \\ \nabla f(x) + W(\nabla^2 f(x)\texttt{dx\_n}) + g'(x)^* y - \mu h'(x)^* L(h(x))^{-1} e & \text{Constrained.} \end{cases}$$

and $W$ denotes the projection onto nullspace of $g'(x)$. Next, we must satisfy one of the following global or local convergence criteria

1. $\log(\texttt{norm\_gxtyp}) - \log(\|g(\texttt{x})\|) < \log(\texttt{mu\_typ}) - \log(\texttt{mu\_est})$
2. $\|g(\texttt{x})\| < \texttt{eps\_constr} \cdot \texttt{norm\_gxtyp}$

In addition, we must converge `mu_est` locally

$$|\texttt{mu} - \texttt{mu\_est}| < \texttt{mu}$$

and not have converged $\texttt{mu}$ globally

$$|\texttt{mu} - \texttt{eps\_mu} \cdot \texttt{mu\_typ}| \geq \texttt{eps\_mu} \cdot \texttt{mu\_typ}.$$

Finally, we also require that $\texttt{iter} > 1$, so that we don't reduce `mu` on the first iteration. For globalization, in both trust-region and line-search methods, we modify our merit function with the addition of a barrier function, which we refer to as `barr`. Specifically, we allow any barrier function such that $x \circ \nabla\texttt{barr}(x) = e$. In our `Rm` and `SQL` vector spaces, we use the log-barrier functions:

$$\begin{array}{ll} \text{Linear} & \langle \log(x), e\rangle, \\ \text{Quadratic} & \frac{1}{2}\log(x_0^2 - \langle \bar{x}, \bar{x}\rangle), \\ \text{Semidefinite} & \log(\det(x)). \end{array}$$

where $\langle \cdot, \cdot \rangle$ refers to the $\ell^2$ inner product. In order to compute the semidefinite barrier function, we Choleski factor $x$ into $u^T u$ since

$$\log(\det(x)) = \log(\det(u^T u)) = \log(\det(u^T)\det(u)) = \log(\det(u)^2) = 2\log(\det(u))$$

and the determinant of an upper triangular matrix can be calculated quickly. As our final step, since we don't require our Jordan product to be commutative, we forcibly symmetrize both $\delta x$ and $\delta z$ using the `symm` operator in our vector space. As far as the initial inequality multiplier, we set

$$\texttt{z} = \texttt{mu} \cdot L(h(\texttt{x}))^{-1} e.$$

This guarantees that

1. $h(\texttt{x}) \circ \texttt{z} = \texttt{mu} \cdot e$
2. $\texttt{mu\_est} = \texttt{mu}$

In other words, our initial inequality multiplier puts us on the central path specified by the parameter `mu`. Historically, we are not the first to use Euclidean-Jordan algebras in an interior point algorithm. Alizadeh and Schmieta describe their use for semidefinite programming and Alizadeh and Goldfarb describe their use in second-order cone programming. Nevertheless, we drop the commutativity requirement in our algorithm. Part of the reason we drop the commutativity requirement is that in the SDP case we essentially generate the same optimality conditions as equation (4.10) in Helmberg, Rendl, Vanderbei, and Wolkowicz's SDP paper. In fact, our symmetrization in the SQL vector space is identical to equation (4.30) in the same paper, which later became known as the HKM search direction. Beyond the HKM symmetrization, we allow any similar symmetrization operator, which Zhang describes in his paper.


- Farid Alizadeh and Stefan Schmieta. Symmetric cones, potential reduction methods and word-by-word extensions. In Henry Wolkowicz, Romesh Saigal, and Lieven Vandenberghe, editors, *Handbook of Semidefinite Programming*, volume 27 of *International Series in Operations Research & Management Science*, pages 195–233. Springer US, 2000.
- F. Alizadeh and D. Goldfarb. Second-order cone programming. *Mathematical Programming*, 95(1):3–51, 2003.
- Christoph Helmberg, Franz Rendl, Robert J. Vanderbei, and Henry Wolkowicz. An interior-point method for semidefinite programming. *SIAM Journal on Optimization*, 6(2):342–361, 1996.
- Yin Zhang. On extending some primal-dual interior-point algorithms from linear programming to semidefinite programming. *SIAM Journal on Optimization*, 8(2):365–386, 1998.
- Richard H. Byrd, Mary E. Hribar, and Jorge Nocedal. An interior point algorithm for large-scale nonlinear programming. *SIAM Journal on Optimization*, 9(4):877–900, 1999.

**Algorithm** Inexact composite-step SQP

**Description** We implement a modified version of inexact composite-step SQP method that Ridzal devised in his Ph.D. thesis and later refined in a technical report by Ridzal, Aguiló, and Heinkenschloss. Our implementation adds several safe-guards in order to more directly account for round-off error within the algorithm, which generally affects the augmented system solves. As one example, when solving the augmented system for the quasi-normal step, if the Cauchy point brings us to optimality, GMRES may not be able to practically satisfy the tolerances the algorithm specifies. Therefore, we detect this case directly and terminate the augmented system solve.


- Denis Ridzal. *Trust-Region SQP Methods with Inexact Linear System Solves for Large-Scale Optimization*. PhD thesis, Rice University, 2006.
- Denis Ridzal, Miguel Aguiló, and Matthias Heinkenschloss. Numerical study of matrix-free trust-region SQP method for equality constrained optimization. Technical Report SAND2011-9346, Sandia National Laboratories, 2011.
- Matthias Heinkenschloss and Denis Ridzal. A matrix-free trust-region sqp method for equality constrained optimization. *SIAM Journal on Optimization*, 24(3):1507–1541, 2014.

# 9 Licenses

In the following chapter, we detail Optizelle's license as well as the license of all of its dependencies.

## 9.1 Optizelle



## 9.2 JsonCpp

In jurisdictions which recognize Public Domain property, the user of this software may choose to accept it either as 1) Public Domain, 2) under the conditions of the MIT License (see below), or 3) under the terms of dual Public Domain/MIT License conditions described here, as they choose.

The MIT License is about as close to Public Domain as a license can get, and is described in clear, concise terms at:

   http://en.wikipedia.org/wiki/MIT_License

The full text of the MIT License follows:

========================================================================

Copyright (c) 2007-2010 Baptiste Lepilleur

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

========================================================================

(END LICENSE TEXT)

The MIT license is compatible with both the GPL and commercial software, affording one all of the rights of Public Domain with the minor nuisance of being required to keep the above copyright notice and license text in the source code. Note also that by accepting the Public Domain "license" you can re-license your copy using whatever license you like.

## 9.3 BLAS/LAPACK

```
Copyright (c) 1992-2011 The University of Tennessee and The University
                        of Tennessee Research Foundation.  All rights
                        reserved.
Copyright (c) 2000-2011 The University of California Berkeley. All
                        rights reserved.
Copyright (c) 2006-2012 The University of Colorado Denver.  All rights
                        reserved.
```

$COPYRIGHT$

```
Additional copyrights may follow

$HEADER$

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

- Redistributions of source code must retain the above copyright
  notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright
  notice, this list of conditions and the following disclaimer listed
  in this license in the documentation and/or other materials
  provided with the distribution.

- Neither the name of the copyright holders nor the names of its
  contributors may be used to endorse or promote products derived from
  this software without specific prior written permission.

The copyright holders provide no reassurances that the source code
provided does not infringe any patent, copyright, or any other
intellectual property rights of third parties.  The copyright holders
disclaim any liability to any recipient for claims brought against
recipient by any third party for infringement of that parties
intellectual property rights.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

## 9.4 CMake

```
CMake - Cross Platform Makefile Generator
Copyright 2000-2016 Kitware, Inc.
Copyright 2000-2011 Insight Software Consortium
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

* Redistributions of source code must retain the above copyright
  notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright
  notice, this list of conditions and the following disclaimer in the
  documentation and/or other materials provided with the distribution.
```

## 9.5 WiX

any person that distributes its contribution under this license. "Licensed patents" are a contributor's patent claims that read directly on its contribution.
2. Grant of Rights (A) Copyright Grant- Subject to the terms of this license, including the license conditions and limitations in section 3, each contributor grants you a non-exclusive, worldwide, royalty-free copyright license to reproduce its contribution, prepare derivative works of its contribution, and distribute its contribution or any derivative works that you create. (B) Patent Grant- Subject to the terms of this license, including the license conditions and limitations in section 3, each contributor grants you a non-exclusive, worldwide, royalty-free license under its licensed patents to make, have made, use, sell, offer for sale, import, and/or otherwise dispose of its contribution in the software or derivative works of the contribution in the software.
3. Conditions and Limitations (A) Reciprocal Grants- For any file you distribute that contains code from the software (in source code or binary format), you must provide recipients the source code to that file along with a copy of this license, which license will govern that file. You may license other files that are entirely your own work and do not contain code from the software under any terms you choose. (B) No Trademark License- This license does not grant you rights to use any contributors' name, logo, or trademarks. (C) If you bring a patent claim against any contributor over patents that you claim are infringed by the software, your patent license from such contributor to the software ends automatically. (D) If you distribute any portion of the software, you must retain all copyright, patent, trademark, and attribution notices that are present in the software. (E) If you distribute any portion of the software in source code form, you may do so only under this license by including a complete copy of this license with your distribution. If you distribute any portion of the software in compiled or object code form, you may only do so under a license that complies with this license. (F) The software is licensed "as-is." You bear the risk of using it. The contributors give no express warranties, guarantees or conditions. You may have additional consumer rights under your local laws which this license cannot change. To the extent permitted under your local laws, the contributors exclude the implied warranties of merchantability, fitness for a particular purpose and non-infringement.

## 9.6 GCC

```
    GNU GENERAL PUBLIC LICENSE
       Version 2, June 1991

 Copyright (C) 1989, 1991 Free Software Foundation, Inc.
     51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

    Preamble

  The licenses for most software are designed to take away your
freedom to share and change it.  By contrast, the GNU General Public
License is intended to guarantee your freedom to share and change free
software--to make sure the software is free for all its users.  This
General Public License applies to most of the Free Software
Foundation's software and to any other program whose authors commit to
using it.  (Some other Free Software Foundation software is covered by
the GNU Library General Public License instead.)  You can apply it to
your programs, too.

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
```

this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you

conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to

this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author

to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>
Copyright (C) <year>  <name of author>

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this
when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) year  name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’.
This is free software, and you are welcome to redistribute it
under certain conditions; type ‘show c’ for details.

The hypothetical commands ‘show w’ and ‘show c’ should show the appropriate
parts of the General Public License.  Of course, the commands you use may
be called something other than ‘show w’ and ‘show c’; they could even be
mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your
school, if any, to sign a "copyright disclaimer" for the program, if
necessary.  Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program
‘Gnomovision’ (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989
Ty Coon, President of Vice

This General Public License does not permit incorporating your program into
proprietary programs.  If your program is a subroutine library, you may
consider it more useful to permit linking proprietary applications with the
library.  If this is what you want to do, use the GNU Library General
Public License instead of this License.

GCC RUNTIME LIBRARY EXCEPTION

Version 3.1, 31 March 2009

Copyright (C) 2009 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this
license document, but changing it is not allowed.

This GCC Runtime Library Exception ("Exception") is an additional
permission under section 7 of the GNU General Public License, version
3 ("GPLv3"). It applies to a given file (the "Runtime Library") that
bears a notice placed by the copyright holder of the file stating that
the file is governed by GPLv3 along with this Exception.

When you use GCC to compile a program, GCC may combine portions of
certain GCC header files and runtime libraries with the compiled
program. The purpose of this Exception is to allow compilation of
non-GPL (including proprietary) programs to use, in this way, the
header files and runtime libraries covered by this Exception.

0. Definitions.

A file is an "Independent Module" if it either requires the Runtime

Library for execution after a Compilation Process, or makes use of an
interface provided by the Runtime Library, but is not otherwise based
on the Runtime Library.

"GCC" means a version of the GNU Compiler Collection, with or without
modifications, governed by version 3 (or a specified later version) of
the GNU General Public License (GPL) with the option of using any
subsequent versions published by the FSF.

"GPL-compatible Software" is software whose conditions of propagation,
modification and use would permit combination with GCC in accord with
the license of GCC.

"Target Code" refers to output from any compiler for a real or virtual
target processor architecture, in executable form or suitable for
input to an assembler, loader, linker and/or execution
phase. Notwithstanding that, Target Code does not include data in any
format that is used as a compiler intermediate representation, or used
for producing a compiler intermediate representation.

The "Compilation Process" transforms code entirely represented in
non-intermediate languages designed for human-written code, and/or in
Java Virtual Machine byte code, into Target Code. Thus, for example,
use of source code generators and preprocessors need not be considered
part of the Compilation Process, since the Compilation Process can be
understood as starting with the output of the generators or
preprocessors.

A Compilation Process is "Eligible" if it is done using GCC, alone or
with other GPL-compatible software, or if it is done without using any
work based on GCC. For example, using non-GPL-compatible Software to
optimize any GCC intermediate representations would not qualify as an
Eligible Compilation Process.

1. Grant of Additional Permission.

You have permission to propagate a work of Target Code formed by
combining the Runtime Library with Independent Modules, even if such
propagation would otherwise violate the terms of GPLv3, provided that
all Target Code was generated by Eligible Compilation Processes. You
may then convey such a combination under terms of your choice,
consistent with the licensing of the Independent Modules.

2. No Weakening of GCC Copyleft.

The availability of this Exception does not imply any general
presumption that third-party software is unaffected by the copyleft
requirements of the license of GCC.

## 9.7 TeX Live

$Id: LICENSE.TL 22793 2011-06-05 15:38:08Z karl $

COPYING CONDITIONS FOR TeX Live:

To the best of our knowledge, all software in the TeX Live distribution is freely redistributable (libre, that is, not necessarily gratis), within the Free Software Foundation's definition and the Debian Free Software Guidelines. Where the two conflict, we generally follow the FSF. If you find any non-free files included, please contact us (references given at the end).

That said, TeX Live has neither a single copyright holder nor a single license covering its entire contents, since it is a collection of many independent packages. Therefore, you may copy, modify, and/or redistribute software from TeX Live only if you comply with the requirements placed thereon by the owners of the respective packages.

To most easily learn these requirements, we suggest checking the TeX Catalogue at: http://www.ctan.org/tex-archive/help/Catalogue/ (or any CTAN mirror). Of course the legal statements within the packages themselves are the final authority.

In some cases, TeX Live is distributed with a snapshot of the CTAN archive, which is entirely independent of and separable from TeX Live itself. (The TeX Collection DVD is one example of this.) Please be aware that the CTAN snapshot contains many files which are *not* freely redistributable; see LICENSE.CTAN for more information.

GUIDELINES FOR REDISTRIBUTION:

In general, you may redistribute TeX Live, with or without modification, for profit or not, according to the usual free software tenets. Here are some general guidelines for doing this:

- If you make any changes to the TeX Live distribution or any package it contains, besides complying with any licensing requirements, you must prominently mention such changes in your modified distribution so that users do not take your work for ours, and know to contact you, not us, in case of questions or problems. A new top-level file README.<yourwork> is a good place to describe the general situation.

- Especially (but not necessarily) if changes or additions are made, we recommend a clearly different title, such as "<your work> DVD, based on TeX Live YYYY", where YYYY is the year of TeX Live you are using. This credits both our work and yours.

- You absolutely may *not* place your own copyright on the entire distribution, since it is not your work. Statements such as "all rights reserved" and "may not be reproduced" are especially reprehensible, since they are antithetical to the free software principles under which TeX Live is produced.

- You may use any cover or media label designs that you wish. Such packaging and marketing details are not covered by any TeX Live license.

- Finally, we make the following requests (not legal requirements):

a) Acknowledging that TeX Live is developed as a joint effort by all TeX user groups, and encouraging the user/reader to join their user group of choice, as listed on the web page http://tug.org/usergroups.html.

```
b) Referencing the TeX Live home page: http://tug.org/texlive/

Such information may be placed on the label of your media, your cover,
and/or in accompanying text (for instance, in the acknowledgements
section of a book).

Finally, although it is again not a requirement, we'd like to invite any
redistributors to make a donation to the project, whether cash or
in-kind, for example via https://www.tug.org/donate/dev.html.  Thanks.


If you have any questions or comments, *please* contact us.  In general,
we appreciate being given the chance to review any TeX Live-related
material in advance of publication, simply to avoid mistakes.  It is
much better to correct text on a CD label or in a book before thousands
of copies are made!

We are also happy to keep anyone planning a publication informed as to
our deadlines and progress.  Just let us know.  However, be aware that
TeX Live is produced entirely by volunteers, and no dates can be
guaranteed.


LICENSING FOR NEW PACKAGES:

Finally, we are often asked what license to use for new work.  To be
considered for inclusion on TeX Live, a package must use a free software
license, such as the LaTeX Project Public License, the GNU General
Public License, the modified BSD license, etc.  (Please use an existing
license instead of making up your own.)  Furthermore, all sources must
be available, including for documentation files.  Please see
http://tug.org/texlive/pkgcontrib.html for more information, and other
considerations.

Thanks for your interest in TeX.

- Karl Berry, for the TeX Live project

------------------------------------------------------------
TeX Live mailing list: http://lists.tug.org/tex-live
TeX Live home page:    http://tug.org/tex-live/

The FSF's free software definition: http://www.gnu.org/philosophy/free-sw.html
Debian Free Software Guidelines:    http://www.debian.org/intro/free
FSF commentary on existing licenses:
  http://www.gnu.org/licenses/license-list.html

LPPL: http://latex-project.org/lppl.html or texmf-dist/doc/latex/base/lppl.txt
LPPL rationale: texmf-dist/doc/latex/base/modguide.pdf
```

## 9.8 Python

```
A. HISTORY OF THE SOFTWARE
==========================
```

Python was created in the early 1990s by Guido van Rossum at Stichting Mathematisch Centrum (CWI, see http://www.cwi.nl) in the Netherlands as a successor of a language called ABC. Guido remains Python's principal author, although it includes many contributions from others.

In 1995, Guido continued his work on Python at the Corporation for National Research Initiatives (CNRI, see http://www.cnri.reston.va.us) in Reston, Virginia where he released several versions of the software.

In May 2000, Guido and the Python core development team moved to BeOpen.com to form the BeOpen PythonLabs team. In October of the same year, the PythonLabs team moved to Digital Creations (now Zope Corporation, see http://www.zope.com). In 2001, the Python Software Foundation (PSF, see http://www.python.org/psf/) was formed, a non-profit organization created specifically to own Python-related Intellectual Property. Zope Corporation is a sponsoring member of the PSF.

All Python releases are Open Source (see http://www.opensource.org for the Open Source Definition). Historically, most, but not all, Python releases have also been GPL-compatible; the table below summarizes the various releases.

| Release | Derived from | Year | Owner | GPL-compatible? (1) |
|---|---|---|---|---|
| 0.9.0 thru 1.2 | | 1991-1995 | CWI | yes |
| 1.3 thru 1.5.2 | 1.2 | 1995-1999 | CNRI | yes |
| 1.6 | 1.5.2 | 2000 | CNRI | no |
| 2.0 | 1.6 | 2000 | BeOpen.com | no |
| 1.6.1 | 1.6 | 2001 | CNRI | yes (2) |
| 2.1 | 2.0+1.6.1 | 2001 | PSF | no |
| 2.0.1 | 2.0+1.6.1 | 2001 | PSF | yes |
| 2.1.1 | 2.1+2.0.1 | 2001 | PSF | yes |
| 2.2 | 2.1.1 | 2001 | PSF | yes |
| 2.1.2 | 2.1.1 | 2002 | PSF | yes |
| 2.1.3 | 2.1.2 | 2002 | PSF | yes |
| 2.2.1 | 2.2 | 2002 | PSF | yes |
| 2.2.2 | 2.2.1 | 2002 | PSF | yes |
| 2.2.3 | 2.2.2 | 2003 | PSF | yes |
| 2.3 | 2.2.2 | 2002-2003 | PSF | yes |
| 2.3.1 | 2.3 | 2002-2003 | PSF | yes |
| 2.3.2 | 2.3.1 | 2002-2003 | PSF | yes |
| 2.3.3 | 2.3.2 | 2002-2003 | PSF | yes |
| 2.3.4 | 2.3.3 | 2004 | PSF | yes |
| 2.3.5 | 2.3.4 | 2005 | PSF | yes |
| 2.4 | 2.3 | 2004 | PSF | yes |
| 2.4.1 | 2.4 | 2005 | PSF | yes |
| 2.4.2 | 2.4.1 | 2005 | PSF | yes |
| 2.4.3 | 2.4.2 | 2006 | PSF | yes |
| 2.4.4 | 2.4.3 | 2006 | PSF | yes |
| 2.5 | 2.4 | 2006 | PSF | yes |
| 2.5.1 | 2.5 | 2007 | PSF | yes |
| 2.5.2 | 2.5.1 | 2008 | PSF | yes |
| 2.5.3 | 2.5.2 | 2008 | PSF | yes |
| 2.6 | 2.5 | 2008 | PSF | yes |

| | | | | |
|---|---|---|---|---|
| 2.6.1 | 2.6 | 2008 | PSF | yes |
| 2.6.2 | 2.6.1 | 2009 | PSF | yes |
| 2.6.3 | 2.6.2 | 2009 | PSF | yes |
| 2.6.4 | 2.6.3 | 2009 | PSF | yes |
| 2.6.5 | 2.6.4 | 2010 | PSF | yes |
| 2.7 | 2.6 | 2010 | PSF | yes |

Footnotes:

(1) GPL-compatible doesn't mean that we're distributing Python under the GPL. All Python licenses, unlike the GPL, let you distribute a modified version without making your changes open source. The GPL-compatible licenses make it possible to combine Python with other software that is released under the GPL; the others don't.

(2) According to Richard Stallman, 1.6.1 is not GPL-compatible, because its license has a choice of law clause. According to CNRI, however, Stallman's lawyer has told CNRI's lawyer that 1.6.1 is "not incompatible" with the GPL.

Thanks to the many outside volunteers who have worked under Guido's direction to make these releases possible.

## B. TERMS AND CONDITIONS FOR ACCESSING OR OTHERWISE USING PYTHON

### PYTHON SOFTWARE FOUNDATION LICENSE VERSION 2

1. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and the Individual or Organization ("Licensee") accessing and otherwise using this software ("Python") in source or binary form and its associated documentation.

2. Subject to the terms and conditions of this License Agreement, PSF hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python alone or in any derivative version, provided, however, that PSF's License Agreement and PSF's notice of copyright, i.e., "Copyright (c) 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010 Python Software Foundation; All Rights Reserved" are retained in Python alone or in any derivative version prepared by Licensee.

3. In the event Licensee prepares a derivative work that is based on or incorporates Python or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python.

4. PSF is making Python available to Licensee on an "AS IS" basis. PSF MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, PSF MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

```
5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON
FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS
A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON,
OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material
breach of its terms and conditions.

7. Nothing in this License Agreement shall be deemed to create any
relationship of agency, partnership, or joint venture between PSF and
Licensee.  This License Agreement does not grant permission to use PSF
trademarks or trade name in a trademark sense to endorse or promote
products or services of Licensee, or any third party.

8. By copying, installing or otherwise using Python, Licensee
agrees to be bound by the terms and conditions of this License
Agreement.


BEOPEN.COM LICENSE AGREEMENT FOR PYTHON 2.0
-------------------------------------------

BEOPEN PYTHON OPEN SOURCE LICENSE AGREEMENT VERSION 1

1. This LICENSE AGREEMENT is between BeOpen.com ("BeOpen"), having an
office at 160 Saratoga Avenue, Santa Clara, CA 95051, and the
Individual or Organization ("Licensee") accessing and otherwise using
this software in source or binary form and its associated
documentation ("the Software").

2. Subject to the terms and conditions of this BeOpen Python License
Agreement, BeOpen hereby grants Licensee a non-exclusive,
royalty-free, world-wide license to reproduce, analyze, test, perform
and/or display publicly, prepare derivative works, distribute, and
otherwise use the Software alone or in any derivative version,
provided, however, that the BeOpen Python License is retained in the
Software, alone or in any derivative version prepared by Licensee.

3. BeOpen is making the Software available to Licensee on an "AS IS"
basis.  BEOPEN MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR
IMPLIED.  BY WAY OF EXAMPLE, BUT NOT LIMITATION, BEOPEN MAKES NO AND
DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS
FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE WILL NOT
INFRINGE ANY THIRD PARTY RIGHTS.

4. BEOPEN SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF THE
SOFTWARE FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS
AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE SOFTWARE, OR ANY
DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

5. This License Agreement will automatically terminate upon a material
breach of its terms and conditions.

6. This License Agreement shall be governed by and interpreted in all
respects by the law of the State of California, excluding conflict of
law provisions.  Nothing in this License Agreement shall be deemed to
create any relationship of agency, partnership, or joint venture
```

between BeOpen and Licensee. This License Agreement does not grant permission to use BeOpen trademarks or trade names in a trademark sense to endorse or promote products or services of Licensee, or any third party. As an exception, the "BeOpen Python" logos available at http://www.pythonlabs.com/logos.html may be used according to the permissions granted on that web page.

7. By copying, installing or otherwise using the software, Licensee agrees to be bound by the terms and conditions of this License Agreement.

CNRI LICENSE AGREEMENT FOR PYTHON 1.6.1
---------------------------------------

1. This LICENSE AGREEMENT is between the Corporation for National Research Initiatives, having an office at 1895 Preston White Drive, Reston, VA 20191 ("CNRI"), and the Individual or Organization ("Licensee") accessing and otherwise using Python 1.6.1 software in source or binary form and its associated documentation.

2. Subject to the terms and conditions of this License Agreement, CNRI hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python 1.6.1 alone or in any derivative version, provided, however, that CNRI's License Agreement and CNRI's notice of copyright, i.e., "Copyright (c) 1995-2001 Corporation for National Research Initiatives; All Rights Reserved" are retained in Python 1.6.1 alone or in any derivative version prepared by Licensee. Alternately, in lieu of CNRI's License Agreement, Licensee may substitute the following text (omitting the quotes): "Python 1.6.1 is made available subject to the terms and conditions in CNRI's License Agreement. This Agreement together with Python 1.6.1 may be located on the Internet using the following unique, persistent identifier (known as a handle): 1895.22/1013. This Agreement may also be obtained from a proxy server on the Internet using the following URL: http://hdl.handle.net/1895.22/1013".

3. In the event Licensee prepares a derivative work that is based on or incorporates Python 1.6.1 or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python 1.6.1.

4. CNRI is making Python 1.6.1 available to Licensee on an "AS IS" basis. CNRI MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, CNRI MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON 1.6.1 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. CNRI SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 1.6.1 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 1.6.1, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material

```
breach of its terms and conditions.

7. This License Agreement shall be governed by the federal
intellectual property law of the United States, including without
limitation the federal copyright law, and, to the extent such
U.S. federal law does not apply, by the law of the Commonwealth of
Virginia, excluding Virginia's conflict of law provisions.
Notwithstanding the foregoing, with regard to derivative works based
on Python 1.6.1 that incorporate non-separable material that was
previously distributed under the GNU General Public License (GPL), the
law of the Commonwealth of Virginia shall govern this License
Agreement only as to issues arising under or with respect to
Paragraphs 4, 5, and 7 of this License Agreement.  Nothing in this
License Agreement shall be deemed to create any relationship of
agency, partnership, or joint venture between CNRI and Licensee.  This
License Agreement does not grant permission to use CNRI trademarks or
trade name in a trademark sense to endorse or promote products or
services of Licensee, or any third party.

8. By clicking on the "ACCEPT" button where indicated, or by copying,
installing or otherwise using Python 1.6.1, Licensee agrees to be
bound by the terms and conditions of this License Agreement.

        ACCEPT


CWI LICENSE AGREEMENT FOR PYTHON 0.9.0 THROUGH 1.2
--------------------------------------------------

Copyright (c) 1991 - 1995, Stichting Mathematisch Centrum Amsterdam,
The Netherlands.  All rights reserved.

Permission to use, copy, modify, and distribute this software and its
documentation for any purpose and without fee is hereby granted,
provided that the above copyright notice appear in all copies and that
both that copyright notice and this permission notice appear in
supporting documentation, and that the name of Stichting Mathematisch
Centrum or CWI not be used in advertising or publicity pertaining to
distribution of the software without specific, written prior
permission.

STICHTING MATHEMATISCH CENTRUM DISCLAIMS ALL WARRANTIES WITH REGARD TO
THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS, IN NO EVENT SHALL STICHTING MATHEMATISCH CENTRUM BE LIABLE
FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT
OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
```

## 9.9 NumPy

```
Copyright (c) 2005-2016, NumPy Developers.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
```

## 9.10 MATLAB

Use of MATLAB is subject to the Mathworks, Inc. Software License Agreement, which can be found in license_agreement.txt in the MATLAB installation. Compilation and distribution of MATLAB code and MEX files are subject to the Deployment Addendum in Software License Agreement. Specifically,

2. USER CREATED FILES. This Addendum does not apply to MATLAB code files, Simulink model files, MEX-files, MAT-files, VHDL-files, Verilog-files, FIG-files and P-files that are created by Licensee and that do not include any code obtained from MATLAB code files, Simulink model files, MAT-files, P-code, C/C++ files, VHDL-files, Verilog-files, TLC-files, or other Source Code files supplied with the Programs ("User Files"). Licensee may distribute or sublicense without restriction, User Files provided that a principal purpose of the distribution or sublicense is not to replace or replicate a Program or any part of a Program.

Optizelle does not include or distribute any Mathworks code or object files. Optizelle only distributes its own MATLAB code and MEX files and hence may distribute or sublicense itself without restriction.

## 9.11 JSONlab

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The views and conclusions contained in the software and documentation are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of the copyright holders.

## 9.12 Octave

GNU GENERAL PUBLIC LICENSE
Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the

User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE

USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>
Copyright (C) <year>  <name of author>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
```

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
<program>  Copyright (C) <year>  <name of author>
This program comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’.
This is free software, and you are welcome to redistribute it
under certain conditions; type ‘show c’ for details.
```

The hypothetical commands ‘show w’ and ‘show c’ should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <http://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <http://www.gnu.org/philosophy/why-not-lgpl.html>.

# Index