The Automatic Construction and Solution of a Partial Differential Equation from the Strong Form*

Joseph Young[1]**

Institutt for informatikk
Universitetet i Bergen
joseph.young@ii.uib.no

Abstract. In the last ten years, there has been significant improvement and growth in tools that aid the development of finite element methods for solving partial differential equations. These tools assist the user in transforming a weak form of a differential equation into a computable solution. Despite these advancements, solving a differential equation remains challenging. Not only are there many possible weak forms for a particular problem, but the most accurate or most efficient form depends on the problem's structure. Requiring a user to generate a weak form by hand creates a significant hurdle for someone who understands a model, but does not know how to solve it.
We present a new algorithm that finds the solution of a partial differential equation when modeled in its strong form. We accomplish this by applying a first order system least squares algorithm using triangular Bézier patches as our shape functions. After describing our algorithm, we validate our results by presenting a numerical example.

1 Introduction

The variety of algorithms used to solve a partial differential equation has been both an asset as well as a burden. On one hand, we have an assortment of extremely sophisticated tools that allow us to solve a diverse set of problems. On the other, the heterogeneous nature of these algorithms makes it challenging to design a general modeling tool. Within the realm of finite element methods, there has been considerable progress towards this goal. Modeling tools such as deal.II [1], FEniCS [2, 3], FreeFEM [4], GetDP [5], and Sundance [6] allow the user to specify the weak form of a differential equation by hand. Then, given a specific kind of element, these tools either assist in or automate the construction of the linear system that arises from the discretization.

In spite of their usefulness, these tools assume that their user possesses the technical expertise to find the weak form of a differential equation. Unfortunately, this can be a difficult task. Ideally, we would like a system that can transform the original strong form of the differential equation into a computable solution. This would allow a user with far less technical knowledge to solve a problem than is currently possible. While it is doubtful that such a perfect mechanism exists for all differential equations, we focus on a system that can achieve this goal for a relatively broad class of problems.

* This research was financed by the Research Council of Norway through the SAGA-geo project
** Special thanks to Magne Haveraaen for his suggestions and guidance.

Specifically, we automate a first order system least squares algorithm using triangular Bézier patches as our shape functions. Neither our choice of the straightforward least squares algorithm [7, 8] nor our choice of Bézier patches is unique [9–11]. Nonetheless, we combine these pieces in such way that we can automate the construction and solution of any polynomial differential equation where every function can be adequately approximated by a surface composed of several Bézier patches. This includes all smooth functions as well as, in a practical sense, some discontinuous functions. We do not intend nor claim that this system will provide the best possible solution in all cases. Simply, it provides a smooth solution given relatively little analytical work by the user. In this way, we view it as a tool that allows an end user to rapidly prototype a problem and then determine whether further investigation into an alternative algorithm is necessary.

2 A Calculus for Bézier Patches and Surfaces

The key to this method is the manipulation of surfaces composed of Bézier patches. In the following section, we introduce and develop a calculus for these surfaces.

2.1 Basic Definition

Let us define the Ith Bernstein polynomial of degree k over the jth simplex within the set t as $b^k_{I,j,t} : \mathbb{R}^p \to \mathbb{R}$

$$b^k_{I,j,t}(x) = \begin{cases} \binom{k}{I}(T_j(x))^I & [T(x)]_i \geq 0 \text{ for all } i \\ 0 & \text{otherwise} \end{cases}$$

where $I \in \mathbb{N}_0^{p+1}$, $|I| = k$, and $T_j(x)$ denotes the solution y of the $(p+1) \times (p+1)$ linear system

$$\begin{bmatrix} z_1 & z_2 & \dots & z_{p+1} \\ 1 & 1 & \dots & 1 \end{bmatrix} y = \begin{bmatrix} x \\ 1 \end{bmatrix}$$

where $t_j \in \mathbb{R}^{p+1 \times p} = (z_1, \dots, z_{p+1})$ and z_i denotes a corners of the simplex j. Based on these polynomials, we form Bézier patches by taking the sum over all possible polynomials of degree k. We form a surface by summing over all possible simplices.

2.2 Sums, Negation, Subtraction, and Multiplication

Our core manipulations include the sum, negation, subtraction, and multiplication of patches. Assume we have two patches, f and g, defined over the same simplex where

$$f = \sum_{|I|=k_1} \alpha_I b_I^{k_1} \qquad g = \sum_{|I|=k_2} \beta_I b_I^{k_2}.$$

When $k_1 = k_2$, we find the sum $f + g$ by simply adding the corresponding coefficients. However, when the degrees differ, we must elevate the degree of one of the patches until

they match. Without loss of generality, assume $k_1 > k_2$. Then, let

$$\gamma_I = \sum_{\substack{|J| = k_2 \\ (I-J)_i \geq 0 \forall i}} \beta_J \frac{\binom{k_2}{J}\binom{k_1-k_2}{I-J}}{\binom{k_1}{I}}$$

where $|I| = k_1$. Then, we have that [12–14]

$$g = \sum_{|I|=k_2} \beta_I b_I^{k_2} = \sum_{|I|=k_1} \gamma_I b_I^{k_1}.$$

In a similar manner, in order to negate a patch, we simply negate the coefficients. Then, in order to subtract one patch from another, we negate the second patch and add it to the first.

Another useful manipulation is the product between two different Bézier patches which is equal to [12, 14]

$$h = \sum_{|I|=k_1+k_2} \gamma_I b_I^{k_1+k_2}$$

where

$$\gamma_I = \sum_{\substack{|I_1| = k_1, \\ |I_2| = k_2, \\ I = I_1 + I_2}} \frac{\binom{k_1}{I_1}\alpha_{I_1}\binom{k_2}{I_2}\beta_{I_2}}{\binom{k_1+k_2}{I_1+I_2}}.$$

2.3 Derivatives

Next, we give the directional derivative in the direction h of a Bézier patch as [12–14]

$$\left(\sum_{|I|=k} \alpha_I b_{I,j,t}^k\right)'(x)(h) = \sum_{|\tilde{I}|=k-1} \gamma_{\tilde{I}} b_{\tilde{I},j,t}^{k-1}(x)$$

where

$$\gamma_{\tilde{I}} = \sum_{|I|=k} \alpha_I \beta_{\tilde{I}} \qquad \beta_{\tilde{I}} = \begin{cases} k[T_j^0(h)]_i & I - \tilde{I} = e_i \\ 0 & \text{otherwise} \end{cases}.$$

In addition, p describes the length of x, e_i denotes the ith canonical vector, $b_{I-e_i,j,t}^{k-1}(x) = 0$ for all x when $I - e_i$ contains negative indices, and $T_j^0(x)$ denotes the solution to the linear system

$$\begin{bmatrix} z_1 & z_2 & \dots & z_{p+1} \\ 1 & 1 & \dots & 1 \end{bmatrix} y = \begin{bmatrix} x \\ 0 \end{bmatrix}$$

where $t_j \in \mathbb{R}^{p+1 \times p} = (z_1, \dots, z_{p+1})$. This is the same transformation as T_j except that the right hand side includes a 0 in the final element rather than 1. In other words, the derivative of a Bézier patch is simply a Bézier patch of degree one lower.

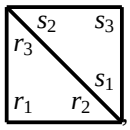
2.4 Smoothness

Although a single Bézier patch is smooth and differentiable, a surface composed of many patches may not be differentiable nor continuous between patches. In order to stitch these patches together, we leverage the blossom of the Bernstein polynomials. This introduces a set of linear constraints on the coefficients where we can tune the amount of smoothness between patches.

We compute the blossom of the Ith Bernstein polynomial of degree k, $\hat{b}_{I,j,t} : \prod_{i=1}^{k} \mathbb{R}^p \to \mathbb{R}$, through the recurrence relation

$$\hat{b}^k_{I,j,t}(x_1, \ldots, x_k) = \sum_{i=1}^{p+1} [T(x_1)]_i \hat{b}^k_{I-e_i,j,t}(x_2, \ldots, x_k)$$

where we define $\hat{b}^0_{I,j,t}(x) = 1$ for all x and $\hat{b}^k_{I,j,t}(x) = 0$ for all x when I contains negative indices. Next, let us consider two simplices, defined by the corners r and s, that share a boundary. The face that defines this boundary requires p points. Thus, without loss of generality let s_{p+1} denote the single point not needed to define this boundary. For example, given the triangulation



the points s_1 and s_2 define the boundary while s_3 remains unneeded. Then, the boundary between the Bézier patches defined over the simplices r and s is q times continuously differentiable when [13, 14]

$$\alpha_{Is} = \sum_{\hat{I}} \alpha_{\hat{I}r} \hat{b}^k_{\hat{I},r,t}(\underbrace{s_1, \ldots, s_1}_{I_1}, \underbrace{s_2, \ldots, s_2}_{I_2}, \ldots, \underbrace{s_{p+1}, \ldots, s_{p+1}}_{I_{p+1}})$$

for all I where $I_{p+1} \leq q$ and α_{Ir} and α_{Is} denotes the coefficients of the Bézier patch over the simplices r and s respectively.

2.5 Symbolic Coefficients

When we use Bézier surfaces within a differential equation, some coefficients are constant whereas others are symbolic. As a result, we require a mechanism to track our symbolic variables. Fortunately, each manipulation we define above simply forms a new Bézier surface where the new coefficients are polynomials of the old. Thus, assume we have n unknown variables x_i. We represent each coefficient by a tuple $(\alpha, a, A, \ldots)$ where α is a constant, a is a vector, A is a matrix, and higher order terms are represented as tensors. In this manner, the actual value of this coefficient is

$$\alpha + \langle a, x \rangle + \langle Ax, x \rangle + \ldots$$

Therefore, we must also define operations such as addition, subtraction, and multiplication on symbolic coefficients. We define addition as

$$(\alpha, a, A, \ldots) + (\beta, b, B, \ldots) = (\alpha + \beta, a + b, A + B, \ldots)$$

multiplication of linear terms as

$$(\alpha, a)(\beta, b) = \left(\alpha\beta, \alpha b + \beta a, (ab^T + ba^T)/2\right)$$

and the other operations analogously.

3 Algorithm

In the following section, we describe our algorithm used to solve the differential equation. We describe this process in three steps. First, we decompose the problem into a first order system. Second, we replace all functions with their Bézier surface equivalents. Third, we construct and solve the least-squares problem. This involves combining all functions together using our calculus defined above, integrating the resulting surfaces, and solving the final optimization problem.

3.1 Decomposition Into a First Order System

As with most least squares approaches, we decompose our problem into a first order system. In the straightforward least squares algorithm, this is not necessary, but yields many benefits such as reducing the condition number of the final optimality system [7, 8]. We emphasize that there are an infinite number of ways to decompose most problems and choosing one particular scheme may have benefits over the other. Our decomposition method is very simple and simply demonstrates that this decomposition is possible to automate.

Our scheme mirrors the trick used to decompose an ordinary differential equation into a system of first order equations. Let us consider a kth order PDE,

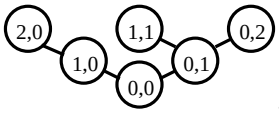
$$F(x, u, D^{\alpha_1}u, \ldots, D^{\alpha_m}u) = 0$$

where α_i is a multiindex in $\mathbb{N}_0^p$. We construct a graph where we label each node by α, for $|\alpha| \leq k$, and connect two nodes α_i and α_j when $\|\alpha_i - \alpha_j\|_1 = 1$. Thus, we connect two nodes if they differ by a single index. Further, we only allow nodes α when $\alpha \preceq \alpha_i$ for some α_i where $\preceq$ denotes pointwise inequality. By its construction, this graph must be connected and contain a spanning tree. The spanning tree gives us our decomposition

$$F(x, u, D^{\alpha_1-\beta_1}u_{\beta_1}, \ldots, D^{\alpha_m-\beta_m}u_{\beta_m}) = 0 \qquad D^{\delta-\gamma}u_\gamma = u_\delta.$$

In the first equality, the constant β_i is a label where $\alpha_i \succ \beta_i$ and both α_i and β_i are connected in the spanning tree. The second equality must hold for all labels δ and γ such that $\beta_i \succeq \delta \succ \gamma$ and both δ and γ are connected in the spanning tree. Finally, we replace any instance of u_0 by u.

For example, consider the problem $D^{20}u + D^{11}u + D^{02}u = 0$. The spanning tree generated by the problem is



.

Thus, according to this decomposition, we would rewrite our problem as

$$D^{10}u_{10} + D^{10}u_{01} + D^{01}u_{01} = 0 \qquad D^{10}u = u_{10} \qquad D^{01}u = u_{01}\,.$$

3.2 Approximating Non-Bézier Surfaces

Our algorithm requires that all functions be Bézier surfaces. Therefore, all non-Bézier surfaces must be approximated. In order to accomplish this, we force the error in the approximation to be orthogonal to the Bernstein polynomials used to construct our Bézier patches. This leads to the system of equations generated by

$$\sum_{j=1}^{|t|} \sum_{I=|k|} \alpha_{Ij} \left\langle b^k_{I,j,t}, b^k_{\hat{I},\hat{j},t} \right\rangle_{L^2(\Omega)} = \left\langle f, b^k_{\hat{I},\hat{j},t} \right\rangle_{L^2(\Omega)}$$

for all $|\hat{I}| = k$ and all $1 \leq \hat{j} \leq |t|$. The domain Ω is a square domain that encompasses the domain of the differential equation. After solving, the variable α defines the coefficients of a Bézier surface that approximates f.

3.3 Constructing the Least Squares Problem

Given a first order decomposition, we rewrite a system of differential equations

$$F_i(x, u, D^{\alpha_1}u, \ldots, D^{\alpha_m}u) = 0 \text{ on } \Omega \qquad G_j(x, u, D^{\alpha_1}u, \ldots, D^{\alpha_m}u) = 0 \text{ on } \partial\Omega$$

as the least squares problem

$$\min_u \sum_i \|F_i(x, u, D^{\alpha_1}u, \ldots, D^{\alpha_m}u)\|^2_{L^2(\Omega)} + \sum_j \|G_j(x, u, D^{\alpha_1}u, \ldots, D^{\alpha_m}u)\|^2_{L^2(\partial\Omega)}.$$

We have chosen the L^2 norm since we implement the straightforward least squares algorithm. From a constructive point of view, we require a norm that uses our algebra on Bézier surfaces. As long as we restrict ourselves to the real numbers, this includes all $W^{k,p}$ norms for an integer k and even p. As a result, it may be possible to automate a more complicated least-squares algorithm as long as it adheres to these norms. However, we do not explore this idea further within this paper.

Next, we replace all non-Bézier surfaces with Bézier surface approximations. We also discretize u using Bézier surfaces as our shape functions. The coefficients for each of these surfaces uses our symbolic scheme from above. In the case of our unknown coefficients, initially all terms are zero except for the linear term which is the ith canonical vector, e_i, where i denotes the index of that unknown coefficient.

Once we have made this substitution, we use our calculus on Bézier patches from above to combine all patches into a single patch. In other words, we simplify all subtraction, addition, multiplication, negation, and differentiation operations including the squaring of the problem due to the L^2 norm. Since we require this simplification, we focus solely on polynomial differential equations composed of these operations. In the end, we are left with the optimization problem

$$\min_{x \in \mathbb{R}^n} \int_\Omega \left(\sum_{j=1}^{|t|} \sum_{|I|=k_i} p_{I,j}(x) b^{k_i}_{I,j,t} \right) + \int_{\partial\Omega} \left(\sum_{j=1}^{|t|} \sum_{|I|=\hat{k}_i} q_{I,j}(x) b^{\hat{k}_i}_{I,j,t} \right)$$

where $p_{I,j}$ and $q_{I,j}$ are polynomials whose coefficients are still represented using our symbolic scheme. Then, we integrate away all the Bézier surfaces and obtain a polynomial r based on these symbolic coefficients. Finally, we add stitching constraints based on the blossoming scheme described above. Thus, our complete problem has the form

$$\min_{x \in \mathbb{R}^n} r(x) \text{ st } Ax = 0$$

where A is a matrix generated by including all possible stitching constraints. As a note, we require the differentiability between patches to be equal to the highest order derivative in the problem. Even though we decompose our problem into first order form, we can not guarantee convergence unless all functions conform to the original space of functions. For example, if we solve Poisson's equation, $\Delta u = f$, we require two derivatives between each patch.

In the end, we have a linearly constrained polynomial program. In the case of a linear differential equation, this yields a linearly constrained quadratic program.

4 Numerical Experiments

In order to experimentally verify the convergence of our method, we solve Poisson's equation in two dimensions on the unit square

$$\begin{aligned} u_{xx} + u_{yy} = x, y \mapsto -8\pi^2 \cos(2\pi x) \sin(2\pi y) \text{ on } [0,1] \times [0,1] \\ u = x, y \mapsto \sin(2\pi y) \text{ when } x = 0 \text{ or } x = 1 \\ u = 0 \text{ when } y = 0 \text{ or } y = 1. \end{aligned}$$

The solution to this problem is the function u where $u(x, y) = \cos(2\pi x) \sin(2\pi y)$.

We solved the problem formulated as both a first order system as well as an undecomposed problem using the original formulation. In both cases, we used fourth order Bézier surfaces. The table below summarizes the relative error in these solutions as well as the condition number of the KKT conditions

	First Order System			Original Formulation		
Subdivisions	Error	Rate	κ	Error	Rate	κ
1	6.00e-1	-	6.02e+3	6.43e-1	-	1.84e+3
2	4.03e-1	0.574	5.19e+3	2.85e-1	1.17	5.57e+3
4	1.18e-1	1.77	1.79e+4	1.88e-1	.600	8.21e+4
8	-	-	-	4.30e-2	2.13	1.28e+6

We do not have results for the first order system on an eight time subdivided surface since we ran out of memory. Although a better implementation would greatly alleviate the problem, this highlights that the symbolic manipulations are memory intensive. As a result, decomposing the problem into a first order system significantly increases the amount of memory required. In spite of this drawback, the method seems to converge.

5 Conclusion

We have presented an algorithm that takes a differential equation modeled in the strong form and automatically produces a solution. It accomplishes this by applying a first order system least squares finite element method. As our shape functions, we use surfaces composed of triangular Bézier patches. We use these functions because their special properties allow us to combine all functions in the differential equation into a single Bézier surface whose coefficients are simply polynomials of our unknown variables. This allows us to produce a linearly constrained polynomial program which, when solved, produces our solution.

Certainly, this algorithm will not work in all cases. However, it provides a mechanism that produces a solution with a minimal amount of effort. Even in cases when the true solution contains features that are difficult to model, we can use this method to quickly produce a rough approximation to the true solution.

References

1. Bangerth, W., Hartmann, R., Kanschat, G.: deal.II–a general purpose object-oriented finite element library. ACM Transactions on Mathematical Software **33**(2) (August 2007)
2. Dupont, T., Hoffman, J., Johnson, C., Kirby, R., Larson, M., Logg, A., Scott, R.: The FEniCS project. (PREPRINT 2003-21) (2003)
3. Logg, A.: Automating the finite element method. Sixth Winter School in Computational Mathematics (March 2006)
4. Hecht, F., Pironneau, O., Hyaric, A.L., Ohtsuka, K.: Freefem++. www.freefem.org Second Edition, Version 2.24-2-2.
5. Dular, P., Geuzaine, C., Henrotte, F., Legros, W.: A general environment for the treatment of discrete problems and its application to the finite element method. IEEE Transactions on Magnetics **34**(5) (September 1998) 3395–3398
6. Long, K.: Sundance 2.0 tutorial. Technical Report SAND2004-4793, Sandia National Laboratory (July 2004)
7. Bochev, P.B., Gunzburger, M.D.: Finite element methods of least-squares type. SIAM Review **40**(4) (December 1998) 789–837
8. Bochev, P.B., Gunzburger, M.D.: Least-Squares: Finite Element Methods. Springer (2009)
9. Zheng, J., Sederberg, T.W., Johnson, R.W.: Least squares methods for solving differential equations using Bézier control points. Applied Numerical Mathematics (48) (2004) 237–252
10. Awanou, G., Lai, M.J., Wenston, P.: The multivariate spline method for scattered data fitting and numerical solution of partial differential equations. In Chen, G., Lai, M.J., eds.: Wavelets and Splines: Athens 2005. Nashboro Press (2006) 24–74
11. Schumaker, L.L.: Computing bivariate splines in scattered data fitting and the finite-element method. Numerical Algorithms **48** (2008) 237–260
12. de Boor, C.: B-form basics. In Farin, G., ed.: Geometric Modeling: Algorithms and New Trends. (1987)
13. Farin, G.: Triangular Bernstein-Bézier patches. Computed Aided Geometric Design 3 (1986) 83–127
14. Prautzsch, H., Boehm, W., Paluszny, M.: Bézier and B-Spline Techniques. Springer (2002)