

BSCM: A C++ Library for the Lattice Solution of Boundary Value Problems

Jeffery A. Solheim

Abstract—In their paper that appeared in the *Journal of Computational Physics*, Umar, Wu, Strayer, and Bottcher described the Basis-Spline Collocation Method, a technique which they developed for the lattice solution of boundary value problems. Umar *et al.* demonstrate how their method can be applied to a variety of problems in Physics, such as calculating bound states of the radial Schrödinger equation, and solving the Poisson equation.

Based on the paper of Umar *et al.*, we have developed BSCM, a C++ library intended to facilitate utilization of the basis-spline collocation method for the lattice solution of boundary problems. In this paper, we describe our BSCM library, and subsequently discuss its application to find an iterative solution to the heat equation. Finally, we describe a visualization tool that we have developed that utilizes our BSCM library. We believe that our library can be a useful tool for researchers in Physics, Mathematics, and other disciplines.

Index Terms—Basis spline, boundary value, C++, heat equation, lattice.

I. INTRODUCTION

In their paper appearing in the *Journal of Computational Physics*, Umar, Wu, Strayer, and Bottcher describe how the basis-spline collocation method can be used in the lattice solution of boundary value problems [1]. Although space limitations do not permit a complete description of their method in this paper, we next describe those aspects of their method that are most pertinent to our later discussion of the BSCM library.

The setting of Umar *et al.* requires specification of a finite sequence of “knot points,” $\{x_i\}$, along a horizontal axis. Subsequently, Umar *et al.* construct an interpolating spline function of “order” M to be a linear combination of an indexed collection of continuous piecewise polynomial functions $B_i^M(x)$, each of degree $M-1$. (For each i , $B_i^M(x)$ is nonzero only on the interval $[x_i, x_i+M]$.) The interpolating function is designed so that it passes through a set of chosen collocation points, $\{(x_\alpha, y_\alpha)\}$.

Umar *et al.* go on to construct a system of linear equations relating the interpolating function to the aforementioned piecewise polynomials $B_i^M(x)$

$$f_\alpha = \sum_{i=1}^{N+M-1} B_{\alpha i} \cdot c_i \quad (1)$$

where $f_\alpha = f(x_\alpha)$, N is the number of collocation points,

$B_{\alpha i} \equiv B_i^M(x_\alpha)$, and the c_i are real constants that are determined by (1). Equation (1) can also be expressed using matrix notation as

$$\mathbf{f} = \mathbf{B} \cdot \mathbf{c} \quad (2)$$

Later in their paper, Umar *et al.* describe a technique whereby fixed boundary conditions are imposed that govern the behavior of derivatives of the interpolating function at the leftmost and rightmost boundaries of the physical region being simulated. Boundary conditions are specified via an M by $M-1$ matrix that is denoted K_{rp} .

Umar *et al.* then define a matrix β_{ri} in terms of both K_{rp} and the values of derivatives of $B_i^M(x)$ evaluated at the boundaries

$$\beta_{ri} = \sum_{p \geq 0} K_{rp} \cdot \left[\frac{\partial^p}{\partial x^p} B_i^M(x) \right]_{\text{boundary}} \quad (3)$$

Later in their paper, Umar *et al.* define a matrix $\tilde{\mathbf{B}}$ formed by placing the rows of $\mathbf{B}$ from (2) above the rows of β_{ri} from (3)

$$\tilde{\mathbf{B}} = \begin{pmatrix} \mathbf{B} \\ \beta_{ri} \end{pmatrix} \quad (4)$$

In this paper, we describe BSCM, which is our C++ implementation of the techniques described by Umar *et al.* We also demonstrate the BSCM library’s applicability by utilizing it to solve the heat equation. Finally, we discuss a BSCM-based visualization tool we have developed for the one-dimensional heat equation.

II. UTILIZING THE BSCM LIBRARY

The interpolating functionality described by Umar *et al.* [1] is encapsulated in our BSCM library’s Spline class. A C++ programmer utilizing the BSCM library will instantiate this class, then use the resulting object to access to the interpolating function and related entities.

A. The Spline Constructor

We have designed the constructor of BSCM’s Spline class to accept three arguments: the spline order M , a Standard Template Library (STL) [2], [3] vector of knot points, and an Eigen object of type MatrixXi specifying fixed boundary conditions. In our BSCM C++ library, the vector of knot points uses zero-based indexing, which differs from the paper of Umar *et al.*, which uses one-based indexing. (There are a few other places in the BSCM C++ library that utilize

zero-based indexing, and they are noted in BSCM's HTML documentation.)

The constructor calculates and initializes various data members of the Spline class. For example, the public data member B matrix is initialized. This matrix is the matrix $B_{\alpha i}$ of (1). The constructor also initializes data member β_{ri} , the matrix β_{ri} of (3). Several additional data members are calculated and initialized, including $\tilde{B}$ of (4).

B. Overloaded Function B

The BSCM library provides two versions of function named B . In the notation of Umar et al., these two functions calculate and return $B_i^k(x)$ for arbitrary real x , and $B_i^k(x_\alpha)$ for collocation point x_α , respectively.

Each of these two overloaded versions takes three parameters. The first parameter of each, k , specifies a spline order. The second parameter of each, i , selects the i^{th} basis function of the given spline order k . The two overloaded versions of B are differentiated by their third parameters. When calculating B_i^k at arbitrary real x , the third parameter is of type double; in contrast, when calculating B_i^k at collocation point x_α , the third parameter, α , is of type `size_t`, and is an index into a vector of collocation points $\{x_\alpha\}$.

C. Overloaded Function D_B

The BSCM library provides two versions of function named D_B . The application programmer invokes either of these functions in order to determine the value of a derivative of a basis function $B_i^M(x)$. Each of the two overloaded versions of D_B has four parameters, but the two versions of the function are differentiated by their fourth parameters.

In one version of D_B , the fourth parameter is double. When the function call $D_B(p, k, i, x)$ is made (where x is of type double), the function returns $\partial^p B_i^k(x) / \partial x^p$, the value of the p^{th} derivative of $B_i^k(x)$, evaluated at x . In this version of the function, the fourth parameter, x , is an arbitrary real number, rather than an index into the vector of collocation points.

In the second version of D_B , the fourth parameter is of data type `size_t`, and the function call $D_B(p, k, i, \alpha)$ returns $\partial^p B_i^k(x_\alpha) / \partial x^p$, the value of the p^{th} derivative of B_i^k , evaluated at x_α . In this case, the fourth parameter, α , is an index into a vector of collocation points $\{x_\alpha\}$.

D. Matrix Representation of Differential Operator O_α^β

The function operator `Matrix` is a BSCM function most likely to be of great utility to the researcher who is utilizing the basis-spline collocation method for the lattice solution of boundary value problems. A call to this instance method of the Spline class returns O_α^β , the collocation space matrix representation of a differentiation operator. The one parameter of this method is `derivativeOrder`, which indicates which derivative operator is required. For example, the function call `operator Matrix(2)` would return the matrix representation of the second derivative operator, $\partial^2 / \partial x^2$.

E. Public Attributes of the Spline Class

The Spline class exposes several attributes as public instance variables. The spline order M is available as an instance variable named `order`. Note that M must be odd so that K_matrix (described below) will have an even number of

rows. The sequence of knots is exposed as the instance variable `knot X`, and the data member `numKnots` indicates the number of knot points along the x -axis. The sequence the abscissas of the collocation points are accessed via instance variable `collocation X`, and data member `N` indicates the number of collocation points. Data member `B_matrix` (of type `Eigen::MatrixXd`) corresponds to the matrix $B_{\alpha i}$ of Umar's (14). The data member `beta_matrix` (of data type `Eigen::MatrixXd`) corresponds to the matrix β_{ri} of Umar's (18). Boundary constraints are specified via the data member `K_matrix`, which is of data type `Eigen::MatrixXi`. This matrix determines which linear combinations of derivatives of interpolating function f shall be forced to be zero. Within `K_matrix`, the subscript r enumerates the fixed boundary conditions and varies 0 ($M-2$). This differs from the paper of Umar et al. [1], in which r varies 1 ($M-1$). Also within the first $(M-1)/2$ rows of `K_matrix`, derivatives are evaluated at the left physical boundary, while in the last $(M-1)/2$ rows, derivatives are evaluated at the right physical boundary.

F. HTML Documentation

We have included numerous D oxygen comments in our C++ source code. Based on these D oxygen comments, we have generated HTML documentation of the BSCM library's data members and member functions. This HTML documentation describes all public member functions and public attributes of our Spline class in detail. Although the space limitations of this paper prevent inclusion of the entirety of our HTML documentation, in Fig. 1 we provide a sampling of the documentation for one of the function of the BSCM library. Other functions of the BSCM library are documented similarly within the HTML documentation.

III. APPLYING THE BSCM LIBRARY TO SOLVE THE HEAT EQUATION

For the purposes of understanding the heat equation in one dimension [4], imagine a long, thin metal rod that conducts heat energy, paired with infinite reservoirs at its two ends that can effectively determine boundary conditions such as the temperatures at the two ends, or the rates at which heat flows into or out of the two ends. For our purposes, it is assumed that an initial distribution of heat within the rod is given.

The heat equation is a partial differential equation [5] that describes the diffusion of heat within this rod over time. In the one-dimensional case that lacks a heat source, this equation can be expressed as $\partial u / \partial t = \alpha \cdot \partial^2 u / \partial x^2$, where $u(x, t)$ is the temperature at position x along the rod and at time t , and α is the thermal diffusivity of the metal [6].

A. An Iterative Simulation Algorithm to Solve the Heat Equation

In the setting just described, a solution to the heat equation is $u(x, t) = e^{(D-\alpha t)} \cdot u(x, 0)$, where $D = \partial^2 / \partial x^2$. This solution suggests the following iterative algorithm for simulating the diffusion of heat within the rod. (We use the symbol $\leftarrow$ to indicate assignment.)

- 1) Instantiate the Spline class
- 2) $D \leftarrow$ the second derivative operator matrix, obtained via the function call operator `Matrix(2)`

- 3) $A \leftarrow e^{[\alpha D]}$, the exponential of the matrix $\alpha \cdot D$
- 4) $u \leftarrow a$ column vector containing the initial temperatures at collocation points x_a
- 5) for $t = 1, 2, 3, 4, 5, \dots$
 - $u \leftarrow A \cdot u$ (i.e., the new heat vector u at time t is obtained by left-multiplying the preceding u by matrix A)

Member Function Documentation

```
double Spline::B ( size_t k,
                  size_t i,
                  double x
                  )
```

$B_i^k(x)$, basis function B_i^k , evaluated at x

$B_i^k(x)$ is defined only when x is between the first & last knots.

Also, $B_i^k(x) \neq 0$ only when x is between knots x_i & x_{i+k} (i.e., when $x_i \leq x < x_{i+k}$).

See Umar's Equations (1), (2), and (3), p. 428.

Parameters

- k** Spline order, ranging from 1, ..., **M**
- i** Spline index, ranging 0, ..., (*numKnots* - *k* - 1).
- x** Location along horizontal axis

Returns

double

Note

This version of **B** has third parameter of type double, whereas another version of overloaded **B** has third parameter of type size_t.

Fig. 1. Sample HTML documentation.

```
#include "Spline.h"
#include <unsupported/Eigen/MatrixFunctions>

int main ( )
{
    unsigned int splineOrder ;
    std::vector<double> knotVector ;
    Eigen::MatrixXi boundaryConditionsMatrix ( 2, 3 ) ;

    // Here, the programmer needs to initialize
    // splineOrder, knotVector, and boundaryConditionsMatrix.

    // Next, instantiate the Spline class.
    BSCM::Spline testSpline
    ( splineOrder, knotVector, boundaryConditionsMatrix ) ;

    // Obtain the matrix representation of the second derivative.
    Eigen::MatrixXd D = testSpline.operatorMatrix ( 2 ) ;

    double thermalDiffusivity ;

    // Here, the programmer must initialize thermalDiffusivity.

    // Use Eigen to calculate exponential of the thermalDiffusivity
    // times the 2nd derivative operator matrix.
    Eigen::MatrixXd A = (thermalDiffusivity * D).exp() ;

    Eigen::VectorXd u ; // a vector storing temperatures at collocation points

    // Resize vector u to reflect # of collocation points.
    u.resize ( testSpline.numKnots - 2 * testSpline.order + 1 ) ;

    // Initialize u with temperatures at collocation points at time 0.
    u << 1.0, 0.0, 0.5 ; // or other initial temperatures as the programmer sees fit

    // Iterate through several time periods to simulate diffusion of heat
    // within the rod. Each increment in t corresponds to the elapsing
    // of one time period.
    for ( int t = 1 ; t <= 5 ; t++ )
    {
        u = A * u ; // simply multiply A times u to advance one time period
        cout << "after " << t << " time periods, "
        << "temperatures at collocation points are " << u ;
    }
}
```

Fig. 2. Utilizing BSCM within C++.

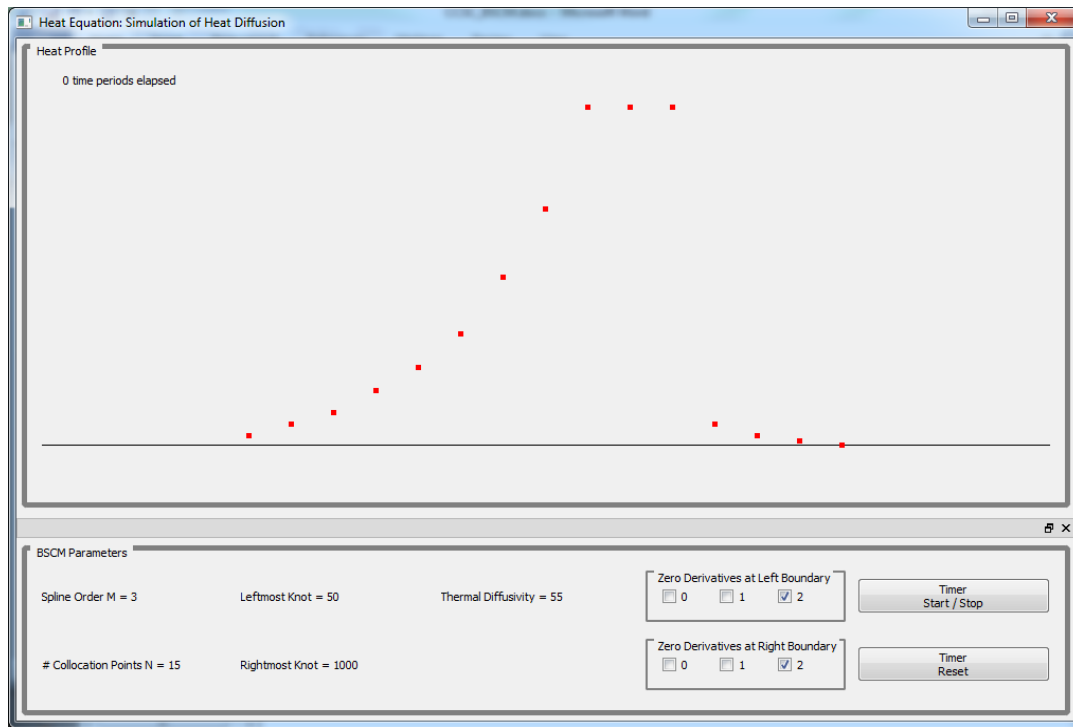


Fig. 3. A BSCM-based tool for visualizing heat diffusion.

B. Utilizing the BSCM Library in C++

In order that the reader may see an example of how to utilize the BSCM library within a C++ program, in Fig. 2 we provide a short C++ program that implements the algorithm just discussed. The C++ programmer can also consult the HTML documentation that accompanies the BSCM library.

C. A BSCM-Based Visualization Tool for Heat Diffusion

In addition to our BSCM library, we have also developed a Qt [7], [8] application that permits its user to visualize the diffusion of heat in the one-dimensional case. This visualization tool utilizes our BSCM library and implements the algorithm described above.

Our visualization tool permits its user to specify spline order, a sequence of knot values, and boundary conditions. In Fig. 3, we provide a screenshot showing how this tool may be utilized to view the distribution of heat within a one-dimensional medium over time. As Fig. 3 shows, the user can click a button to start or to pause a timer that simulates the passage of time. The user may also click a button to reset the time to time $t = 0$. As time proceeds, the user will see the small squares in the top portion of Fig. 3 move upward or downward, as the temperature at each collocation point rises or falls.

IV. CONCLUSION

In this paper, we have described the components and use of the BSCM C++ library, which is intended to facilitate utilization of the basis-spline collocation method for the lattice solution of boundary problems. Also, we have demonstrated applicability of our BSCM library by simulating the diffusion of heat in the one-dimensional case. The interested reader is invited to contact the author to obtain the C++ source code and HTML documentation of the

BSCM library and our visualization tool.

REFERENCES

- [1] A. Umar, J. Wu, M. Strayer, and C. Bottcher, "Basis-spline collocation method for the lattice solution of boundary value problems," *Journal of Computational Physics*, vol. 93, pp. 426-448, April 1991.
- [2] N. Josuttis, *The C++ Standard Library: A Tutorial and Reference*, 2nd ed., Upper Saddle River, NJ: Addison-Wesley Professional, ch. 7, p. 270, 2012.
- [3] P. Plauger, A. Stepanov, M. Lee, and D. Musser, *The C++ Standard Template Library*, 1st ed. Upper Saddle River, NJ: Prentice Hall, ch. 11, p. 290, 2000.
- [4] J. Cannon, *The One-Dimensional Heat Equation*, 1st ed., Cambridge, UK: Cambridge University Press, 1984.
- [5] A. Polyanin, *Handbook of Linear Partial Differential Equations for Engineers and Scientists*, 1st ed., Boca Raton, FL: Chapman & Hall / CRC Press, p. 16, 2002.
- [6] S. Sobolev, *Partial Differential Equations of Mathematical Physics*, 1st ed., Mineola, NY: Pergamon Press, 1964.
- [7] J. Blanchette and M. Summerfield, *C++ GUI Programming with Qt 4*, 2nd ed., Upper Saddle River, NJ: Prentice Hall, 2008.
- [8] M. Summerfield, *Advanced Qt Programming: Creating Great Software With C++ and Qt 4*, 1st ed., Upper Saddle River, NJ: Prentice Hall, 2010.



Jeffery A. Solheim received his bachelor of science degree in computer science from South Dakota School of Mines and Technology (1982, Rapid City, South Dakota, U.S.A.), his master of science degree in mathematics from Montana State University (1984, Bozeman, Montana, U.S.A.), and his Ph.D. in mathematics and computer science from the University of Wyoming (1992, Laramie, Wyoming, U.S.A.). He presently serves as assistant professor of mathematics and computer at Fort Hays State University, Hays, Kansas, U.S.A. His previous publications include *Technology in Second-Language Learning: a Student's Perspective and An Empirical Study of Testing and Integration Strategies Using Artificial Software Systems* (coauthored with J. Rowland). His research interests include the development of software tools to aid in the teaching and learning of mathematics.