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, \( \{ (x_i, y_i) \} \).

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

\[
f_{\alpha} = \sum_{i=1}^{N+M-1} B_{\alpha i} \cdot c_i \tag{1}
\]

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

\[
f = B \cdot c . \tag{2}
\]

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 \( \beta_{ri} \) in terms of both \( K_{rp} \) and the values of derivatives of \( B_i^M(x) \) evaluated at the boundaries

\[
\beta_{ri} = \sum_{p=0}^{M-1} K_{rp} \left[ \frac{\partial^p}{\partial x^p} B_i^M(x) \right]_{\text{boundary}} . \tag{3}
\]

Later in their paper, Umar et al. define a matrix \( \hat{\beta} \) formed by placing the rows of \( B \) from (2) above the rows of \( \beta_{ri} \) from (3)

\[
\hat{\beta} = \begin{pmatrix} \beta & B \end{pmatrix} . \tag{4}
\]

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 Matrix Xi 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 $B_{p,k,i}$
of (1). The constructor also initializes data member
beta_matrix, the matrix $\beta_{p,i}$ of (3). Several additional data
members are calculated and initialized, including $\mathbb{B}$ of (4).

**B. Overloaded Function $D_B$**

The BSCM library provides two versions of function
named $D_B$. In the notation of Umar et al., these two functions
calculate and return $B^k_i(x)$ for arbitrary real $x$, and $B^k_i(x_a)$ for
collocation point $x_a$, 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 $D_B$ are differentiated by their third parameters.
When calculating $B^k_i$ at arbitrary real $x$, the third parameter is
of type double; in contrast, when calculating $B^k_i$ at collocation point $x_a$, the third parameter, $alpha$, is of type size_t,
and is an index into a vector of collocation points $\{x_a\}$.

**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 $B^k_M(x)$. 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 $\frac{\partial^p}{\partial x^p} B^k_i(x)$, the value of
the $p^{th}$ derivative of $B^k_i(x)$, 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
$\frac{\partial^p}{\partial x^p} B^k_i(x_a)$, the value of the $p^{th}$ derivative of $B^k_i$
evaluated at $x_a$. In this case, the fourth parameter, $\alpha$, is an
index into a vector of collocation points $\{x_a\}$.

**D. Matrix Representation of Differential Operator $O_{\alpha,\theta}$**

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 $O_{\alpha,\theta}$, 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, $\frac{\partial^2}{\partial x^2}$.

**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 $K_{\text{matrix}}$ (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
abscess 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 $B_{p,k,i}$
of Umar’s (14). The data member beta_matrix (of data type
Eigen::MatrixXi) corresponds to the matrix $\beta_{p,i}$ of Umar’s
(18). Boundary constraints are specified via the data member
K_matrix, which is of data type Eigen::MatrixXd. 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 $\frac{\partial u}{\partial t} = a \cdot \frac{\partial^2 u}{\partial x^2}$, where $u(x, t)$ is the temperature at position $x$ along the rod and at time $t$, and $a$ 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 $u(x, t) = e^{(D^2 \alpha \theta)} \cdot u(x,0)$, where $D = \frac{\partial^2}{\partial x^2}$. 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) \( A \leftarrow e^{(\alpha \cdot D)} \), the exponential of the matrix \( \alpha \cdot D \)

4) \( u \leftarrow \) a column vector containing the initial temperatures at collocation points \( \lambda_u \)

5) for \( t = 1, 2, 3, 4, 5, \ldots \) \( u \leftarrow A \cdot u \) (i.e., the new heat vector \( u \) at time \( t \) is obtained by left-multiplying the preceding \( u \) by matrix \( A \))

---

**Member Function Documentation**

```cpp
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 \( \lambda_i \) \& \( \lambda_{i+k} \) (i.e., when \( \lambda_i \leq x \leq \lambda_{i+k} \)).
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.

```cpp
#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 thermal diffusivity
    // times the 2nd derivative operator matrix.
    Eigen::MatrixXd D = testSpline.operatorMatrix ( 2 ) ;
    double thermalDiffusivity ;
    // Here, the programmer must initialize thermalDiffusivity.
    // Use Eigen to calculate exponential of the thermal diffusivity
    // 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++.
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 value 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


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.