The Julian Boundary/Finite Element Package

Adam Powell and Vinod Tewary

Version 0.0.2, built July 23, 1999for sgi mips running irix6.5

Abstract

Presented is a software package for mathematical models using the Boundary Element Method (BEM) and (RSN :-) Finite Element Method (FEM). The library includes the following subroutines:

- Mesh specification routines for building a mesh by specifying coordinates of nodes, the nodes which make up each element, the surface elements which make up each group, and for BEM problems, groups which surround each body.
- A modular Green's function matrix calculation routine, which is easily extended to add modules to solve new equations by subroutines which calculate the Green's function matrix for a pair of elements.
- Modular specification of boundary conditions, including Dirichlet, Neumann and Robin (mixed) boundary conditions.
- Necessary linear algebra required for one-step solution of problems with Dirichlet, Neumann or linear Robin boundary conditions; and Newton-Raphson iterative solution of problems with nonlinear Robin boundary conditions.
- A simple wrapper for matrix multiply, decompose, invert and solve operations for easy substitution of alternative approaches to these tasks (default is the native system BLAS/LAPACK routines).
- A CORBA server interface providing object methods for all of the subroutines contained here.
- GNU autoconf/autmake file system for simple compilation and installation across platforms.

It is hoped that this library will simplify the use of boundary element methods in other software.

Dedication

This package, including the documentation, is dedicated to the memory of the late Professor Julian Szekely of MIT, who was a pioneer in the use of mathematical modeling (including the finite element method) to improve materials processing operations.

Contents

Chapter 1

Introduction

This is a really cool program which you should use. Here we start with the FAQ, then go on to user's manual, and next to the developers' information, including file format and calls to `libelem`

Chapter 2

Frequently Asked Questions

2.1 General install problems

Q: Why does `Julian` not build?

A: You need to use GNU `make` to build `Julian`. On many systems, this is done by invoking `gmake` instead of `make`. You can get GNU `make` from GNU at `http://www.gnu.org/`.

2.1.1 SGI IRIX-specific problems

Q: While building `Julian`, I get an error about `ANG:=f77: not found`. What's the deal?

A: This is a known bug in GNU Autoconf, which thinks the flag `-LANG:=f77` is required to link libraries containing FORTRAN sources. When passed to `libtool` it is interpreted as a `-L` library path directory flag, and rejected because the directory doesn't exist. Open `lapack/Makefile` and `libelem/Makefile` and remove this flag from the lines that start with `BLAS_LIBS` and `FLIBS`.

Q: Why do the scroll bars sometimes not function properly in `make check` (or perhaps elsewhere)?

A: This seems to be a bug in the string handling of the IRIX compilers. Specifically, a string passed from one of the `mydxxx` C functions in `libelem` is not being received properly by the `ilaenv` FORTRAN function in `libmylapack`. To illustrate the bug, insert the following line near the top of `lapack/ilaenv.f` and rebuild:

```
WRITE (6,*) 'SUBROUTINE NAME IS ', NAME
```

As a workaround, try editing `lapack/ilaenv.f`, changing the `ILAENV` assignment on line 123 from 1 to 64.

Q: Why does install not copy files that are clearly there?

A: This is because IRIX `install` is Sys V flavored, whereas GNU autoconf requires a BSD-compatible `install`. IRIX provides this in `bsdinst`, so before configuring, set the environment variable `INSTALL`, for example:

```
INSTALL='bsdinst -c' ./configure
```

or for csh/tcsh users:

```
setenv INSTALL 'bsdinst -c'; ./configure
```

If that doesn't work, try `/usr/bin/X11/bsdinst`.

Chapter 3

User's Manual

This is the user's manual. Isn't it cool?

Chapter 4

Library Calls

The foundation of `Julian` is its library, `libelem`, which is a Unix shared library designed to provide all of the boundary element and finite element functionality to any front end or larger program which decides to use it. `libelem` is composed of the three object classes: `matrix`, `mesh` and `bem`, with a fourth, `fem`, to be completed in the future. It is object-oriented in design, and written in the C language for maximum speed, portability, and efficiency in memory use.

In addition to the C language library calls given here, there are header files which serve as very thin wrappers for a C++ language class library. Class names are capitalized, and method names are given by the uncapitalized C method name stripped of its prefix, e.g. `matDecompose()` becomes `Matrix::decompose()`. Exceptions include constructors, destructors, and operators, such as the matrix multiply operator which allows one to write, for example, `Matrix C = A * B`. Error handling is done by assertion.

4.1 Class `matrix`

The `matrix` class includes methods to create and manipulate dense and (in the future) sparse matrices in ways required for BEM and FEM analysis[1] with the following methods:

- `matCreate` is the basic constructor, it allocates memory for the matrix structure and its data.
- `matDuplicate` is an alternate constructor which creates a new matrix object identical to an existing one.
- `matDestroy` is the basic destructor, which frees all memory associated with a matrix object.

[1] It is not intended to be a complete general-purpose matrix object library, though it may become that in the future if there is sufficient interest

Please see the file matrix.h (installed in prefix/Julian) for error returns.

The only current implementation of the compute-intensive matrix methods consists of wrappers to the native system Basic Linear Algebra Subroutines, or BLAS. Alternative dgemm-based implementations of the dgetrf and dtrsm routines and also dgemm itself (for LU-decomposition, back substitution and matrix multiplication respectively) are provided, which make callbacks to external routines to indicate progress toward completion of their operations. These routines are currently optimized for single processor operation, but at some point may have their backend replaced by subroutines which store and operate on matrix components in parallel.

4.2 Class mesh

The mesh class provides methods to construct a mesh by specifying nodal coordinates and node indices which make up elements, and also routines for generic calculations related to the mesh structure, such as integration of a scalar- or vector-valued function over an element group, or calculation of the mass or stiffness matrix for an element group. It can create meshes in any number of dimensions, and elements of arbitrary polynonial order spectral discretization. Its methods are as follows:

- meshCreate is the basic constructor.
- meshDuplicate
- meshDestroy
- meshAddNode
- meshAddNodes
- meshMoveNode
- meshMoveNodes
- meshAddElementGroup
- meshAddElement
- meshAddElements
- meshAddBody
- meshAddGroupToBody
- meshNumNodes
- meshNumGroups
- meshNumGroupNodes

- `meshNumGroupElems`
- `meshGroupNode`
- `meshNodeCoords`
- `meshNodesPerElem`
- `meshElemNodes`
- `meshNodeGroup`
- `meshScalarIntegralElement`
- `meshVectorIntegralElement`

4.2.1 Import/export API for new mesh formats

This section of the code will eventually use loadable modules.

4.3 Class `bem`

4.3.1 Equation API for new Green's Functions

This section of the code will eventually use loadable modules.

4.4 Class `fem`

This will provide various capabilities in the future.

Chapter 5

File Format

The file format consists of a number of fields, each with a text label and field length in bytes[1], followed by linefeed, then the data. Text labels are not case-dependent and contain no whitespace, though they may be proceeded by whitespace (spaces, tabs, CR, LF), and there must be whitespace between the label and bytecount. Text fields may include multiple lines.

Currently supported labels include:

- `Author` The author(s) of the file. If there are multiple `Author` fields, the last one will be official.
- `Title` The title of the file. If there are multiple `Title` fields, the last one will be official.
- `Description` The text description of the file. If there are multiple `Description` fields, the last one will be official.
- `Comment` A comment, can be inserted anywhere in the file.
- `Dimensions` This must come before `Nodes`, and just gives the integer number of dimensions in the problem.
- `Nodes` Text information on nodes. Each node consists of its integer group number (can be any integer) and D floating point numbers giving its coordinates. Node numbers are assigned sequentially starting with zero, and multiple `Nodes` fields just add to the end of the node list.
- N`-Elements` List of integer node numbers that make up each surface element. Interpolation is done using N-order polynomials, with $\frac{(N+D)!}{(N-1)!D!}$ evenly-spaced nodes per element (assuming 0!=-1!=1). Elements are numbered the same way as nodes are, and it is possible to mix several different orders of elements in a given model.

[1]This bytecount is here in order to accomodate both text and binary data. I know it's a pain, but just use M-= in emacs and you'll get the bytecount as well as the line count.

- `Volume` List of surface elements that define a closed volume. Elements listed with a dash in front of them (including -0) are switched to point outward.
- `Equation` List of integer volume index numbers and equation names to apply to them. Each equation comes with a defined number of variables per node.
- `BoundaryCond` List of boundary conditions, each of which consists of a group number, a variable number, and the boundary condition type and parameters. These are written as (case-insensitive):
  - `D(a)` for constant Dirichlet value u=`a`.
  - `F(function)` for Dirichlet BC given by a function of node position.
  - `N(a)` for constant outward flux `a`.
  - `G(function)` for outward flux as a function of node position.
  - `R(a,b)` for linear Robin BC giving outward flux as `a`+`b`u.
  - `L(function1, function2)` for linear Robin BC giving outward flux as `function1`+`function2`u where `function1` and `function2` take node position as arguments.
  - `P(function)` for generalized Robin BC giving outward flux as a function of node position and u.
  - `S(string)` for special boundary conditions unique to equations. The string is passed to the elemental matrix builder and the boundary condition maker, which are unique to each `Equation`.

  Functions are not yet implemented, so only `D`, `N`, `R` and `S` are presently available. Function syntax will be determined at a later time. Any number of these fields can be used; if more than one boundary condition is supplied for a given node group and variable then the last one will be used.
- `DMatrix` LU-decomposed matrix for this problem, stored in long double-precision format in little endian byte order.
- `IMatrix` Inverted matrix for this problem, same format.

Chapter 6

Theory

Much of this is taken from Brebbia [1].

6.1 Basics

Okay, say we have a volume Ω with boundary $\partial\Omega$, on which we are solving $\nabla^2 u = 0$, with $u = \bar{u}$ on $\partial\Omega_1$ and $\nabla u \cdot \hat{n} = \bar{q}$ on $\partial\Omega_2$ ($\hat{n}$ is the outward-pointing unit normal vector). We can approximately solve it using the following weighted residual equation in the Galerkin Finite Element formulation:

$$\int_\Omega \phi \nabla^2 u dV = \int_{\partial\Omega_2} \phi(\nabla u - \bar{q}\hat{n}) \cdot d\vec{A} - \int_{\partial\Omega_1} (u - \bar{u}) \nabla\phi \cdot d\vec{A}. \tag{6.1}$$

Integrating the left side by parts and subtracting $\int_{\partial\Omega} \phi \nabla u \cdot d\vec{A}$ gives

$$-\int_\Omega \nabla\phi \cdot \nabla u dV = -\int_{\partial\Omega_2} \phi\bar{q} dA - \int_{\partial\Omega_1} \phi\nabla u \cdot d\vec{A} - \int_{\partial\Omega_1} u\nabla\phi \cdot d\vec{A} + \int_{\partial\Omega_1} \bar{u}\nabla\phi \cdot d\vec{A}, \tag{6.2}$$

and another integration plus $\int_{\partial\Omega} u\nabla\phi \cdot d\vec{A}$ gives

$$\int_\Omega u\nabla^2\phi dV = -\int_{\partial\Omega_2} \phi\bar{q} dA - \int_{\partial\Omega_1} \phi\nabla u \cdot d\vec{A} + \int_{\partial\Omega_2} u\nabla\phi \cdot d\vec{A} + \int_{\partial\Omega_1} \bar{u}\nabla\phi \cdot d\vec{A}, \tag{6.3}$$

which this book calls the "starting point for the boundary element method".

6.2 Fundamental Solution

Now, let's consider the "fundamental solution" u_i^*, which is the solution to

$$\nabla^2 u_i^* + \Delta^i = 0, \tag{6.4}$$

where Δ^i is the Dirac delta function at point i, *i.e.*

$$\int_\Omega \Delta^i dV = \begin{cases} 1, & i \in \Omega \\ 0 & \text{otherwise} \end{cases} . \tag{6.5}$$

We know that

$$\int_\Omega u(\nabla^2 u_i^* + \Delta^i) dV = \int_\Omega u \nabla^2 u_i^* dV + u^i, \tag{6.6}$$

so since u_i^* satisfies equation 6.4, equation 6.3 becomes

$$u^i + \int_{\partial\Omega_2} u \nabla u_i^* \cdot d\vec{A} + \int_{\partial\Omega_1} \bar{u} \nabla u_i^* \cdot d\vec{A} = \int_{\partial\Omega_2} u_i^* \bar{q} dA + \int_{\partial\Omega_1} u_i^* \nabla u \cdot d\vec{A}. \tag{6.7}$$

Incidentally, this allows the calculation of u at any internal point i given the u and $\nabla u \cdot \hat{n}$ values on the boundary, according to

$$u^i = \int_{\partial\Omega} u_i^* \nabla u \cdot d\vec{A} - \int_{\partial\Omega} u \nabla u_i^* \cdot d\vec{A}. \tag{6.8}$$

The "fundamental solutions" u_i^* that fit in here are

$$u_i^* = \frac{1}{2\pi} \ln\left(\frac{1}{r}\right) \tag{6.9}$$

in 2-D (r is the distance from point i), and

$$u_i^* = \frac{1}{4\pi r} \tag{6.10}$$

in 3-D. For a point i on the surface, because the singularity intersects the integration path, we modify equation 6.7 to read:

$$\frac{u^i}{2} + \int_{\partial\Omega_2} u \nabla u_i^* \cdot d\vec{A} + \int_{\partial\Omega_1} \bar{u} \nabla u_i^* \cdot d\vec{A} = \int_{\partial\Omega_2} u_i^* \bar{q} dA + \int_{\partial\Omega_1} u_i^* \nabla u \cdot d\vec{A}. \tag{6.11}$$

(Note: if the surface isn't smooth at point i, u^i shouldn't be multiplied by $\frac{1}{2}$ but by something else.) This is valid for both 2-D and 3-D cases.

6.3 Boundary elements

Beginning with constant elements, if one assumes that u and $\nabla u \cdot \hat{n}$ are constant over each element j of which the surface is comprised, then equation 6.11 becomes

$$\frac{u^i}{2} + u_j \int_{\partial\Omega_j} \nabla u_i^* \cdot d\vec{A} = (\nabla u \cdot \hat{n})_j \int_{\partial\Omega_j} u_i^* dA, \tag{6.12}$$

where summation is implied for like indices (Einstein notation). Defining the matrices $\hat{H}_{ij}$ and G_{ij} according to the left- and right-side integrals gives the matrix equation

$$\frac{u^i}{2} + \hat{H}_{ij} u_j = G_{ij} (\nabla u \cdot \hat{n})_j, \tag{6.13}$$

which can be simplified further by defining $H_{ij} = \hat{H}_{ij} + \frac{1}{2}\delta_{ij}$ to give

$$H_{ij} u_j = G_{ij} (\nabla u \cdot \hat{n})_j. \tag{6.14}$$

Rearranging the matrix columns and vector rows so that unknowns go on the left and knowns on the right (i.e. for nodes j in $\partial\Omega_1$, u_j goes on the right and $(\nabla u \cdot \hat{n})_j$ on the left; vice versa for $\partial\Omega_2$) gives the matrix equation which solves the problem.

Note that for constant and linear elements, $\hat{H}_{ii}$ is zero because ∇u_i^* is orthogonal to the surface normal. Also, G_{ii} for constant elements can be calculated analytically, and in 2-D is equal to

$$G_{ii} = \int_{\partial\Omega_i} u_i^* dA = \frac{1}{2\pi} \int_{\partial\Omega_i} \ln\left(\frac{1}{r}\right) dA = \frac{1}{\pi} |r_1| \left\{ \ln\left(\frac{1}{r_1}\right) + 1 \right\} \tag{6.15}$$

where $|r_1|$ is half the element width.

6.3.1 Linear elements

For linear elements, we have a more complicated situation. First, u and $\nabla u \cdot \hat{n}$ aren't constant, so we can't take them out of the integrals. A given element has two or three nodes

Nodes are sitting at non-smooth points, so $H_{ij} - \hat{H}_{ij}$ is not $\frac{1}{2}\delta_{ij}$. Instead, equation 6.13 becomes

$$c_i u_i + \hat{H}_{ij} u_j = G_{ij} (\nabla u \cdot \hat{n})_j \tag{6.16}$$

6.4 Matrix Partitioning

The mesh is built as element groups, and multiple element groups can be assembled into sets which define a partition of the mesh. The matrices for the various set pairs may then be decomposed individually, and re-used multiple times to solve various problems.

6.4.1 Collocation

It is important to note that on boundaries with dissimilar boundary conditions (e.g. Dirichlet and Neumann), one must generate multiple points on that boundary. Special thanks to Prof. Ambar Mitra of Iowa State University for pointing this out.

Chapter 7

License

"This software was developed at the National Institute of Standards and Technology by employees of the Federal Government in the course of their official duties. Pursuant to title 17 Section 105 of the United States Code this software is not subject to copyright protection and is in the public domain.

We would appreciate acknowledgement if the software is used."

This will be the license for all code developed prior to the summer of 1999, after which changes in `libelem` will be covered by the GNU Library General Public License (LGPL), and the front ends by the GNU General Public License (GPL).

Bibliography

[1] C.A. Brebbia, *The Boundary Element Method for Engineers,* New York: John Wiley & Sons, 1978.