CSE 380 Project

Finite Element Solution of the Heat Equation

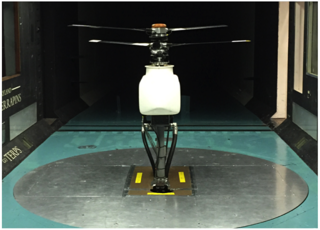
Christopher G. Cameron

The University of Texas at Austin

December 10, 2015

Project motivation and goals . . .

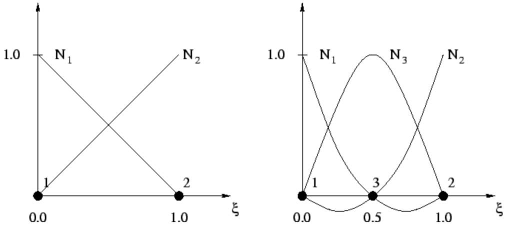
- My work is almost exclusively experimental in nature, giving me freedom in project choice
- Goals
  - Write a significant amount of code from scratch
  - Use a large array of tools from the coursework
  - Object oriented implementation



Finite Element Method . . .

- The finite element method involves solving a variational form of a differential equation
- Test and trial functions are chosen as a summation of function with finite support making all integrations local

$$-\frac{d}{dx}\left(k(x)\frac{du}{dx}\right) = f(x) \longrightarrow \sum_{i=1}^{n}\sum_{j=1}^{n}\int_a^b k(x)\frac{du_i}{dx}\frac{dv_j}{dx}\,dx. = \sum_{i=1}^{n}\int_a^b f(x)u_i\,dx. \tag{1}$$

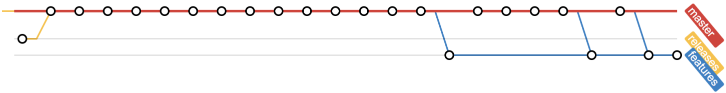


0 Mustafa Radi, 1998

My code ...

- Solves the 1D steady heat equation with 1st and 2nd order finite elements, using GRVY for input parsing and timing
- Objects include:
  - Domain containing vectors of elements, edges, and nodes
    - Elements point to member nodes and edges, have method for calculating stiffness and forcing matrix contributions
    - Edges point to member nodes and have routine for adding nodes for higher order approximations
    - Nodes know their position and global number
  - Solver to hold stiffness matrix and forcing and perform iterative solving
    - Hand coded jacobi and Gauss-Seidel iterative solvers
    - Built in eigen Conjugate Gradient solver
    - Includes output option for comparison with MASA exact solution

Source control and Make System



Project commit tree[1]

- Code written in C++
  - External libraries: MASA, GRVY, Eigen
- Makefile based build system with subfolders for sources, external includes, build files, and binaries
- Git repository on github for version control
  - Most work done on the master branch
  - Branch for GRVY features due to compilation issues on local machine

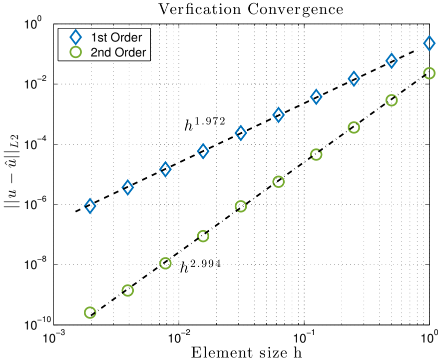
[1] Git vizualization by gitflowchart

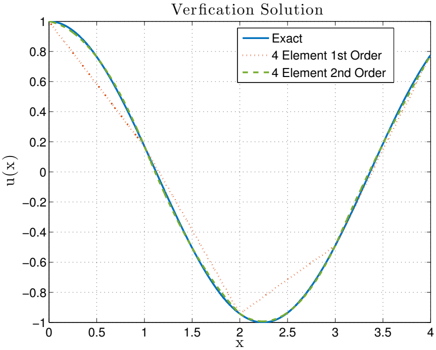
Verification framework

- Catch unit testing library
  - Header only
  - Simple asserts with macros
  - Spread tests across multiple files
  - Sections for reusing setup objects
- MASA manufactured solution library
  - 1d Steady heat equation solution for verification
  - Wrapper object created for forcing evaluation due to late integration into code

```
-------------------------------------------------------------------------------
Line Constructor with Node element
-------------------------------------------------------------------------------
src/test.cpp:60
...............................................................................

src/test.cpp:65:
PASSED:
  CHECK( L1.nodes[0] == &n1 )
with expansion:
  0x00007fff1cb56ec0 == 0x00007fff1cb56ec0

src/test.cpp:66:
PASSED:
  CHECK( L1.nodes[0]->ind == n1.ind )
with expansion:
  0 == 0

src/test.cpp:67:
PASSED:
  CHECK( L1.nodes[1] == &n2 )
with expansion:
  0x00007fff1cb56ee0 == 0x00007fff1cb56ee0

src/test.cpp:68:
PASSED:
  CHECK( L1.nodes[1]->df == n2.df )
with expansion:
  0 == 0

-------------------------------------------------------------------------------
Test addNodes method of Line
-------------------------------------------------------------------------------
src/test.cpp:71
...............................................................................

src/test.cpp:90:
PASSED:
  CHECK( L1.nodes.size() == 4 )
with expansion:
  4 == 4

src/test.cpp:91:
PASSED:
  CHECK( nodes_e.size() == 5 )
with expansion:
  5 == 5

src/test.cpp:92:
PASSED:
  CHECK( nodes_g.size() == 5 )
with expansion:
  5 == 5
```

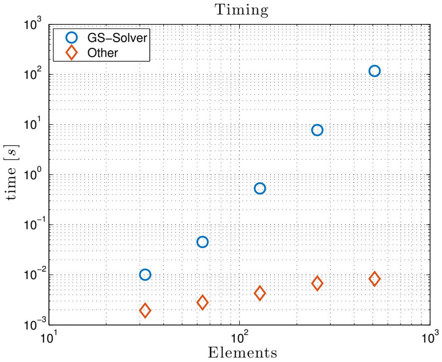
MASA Verification

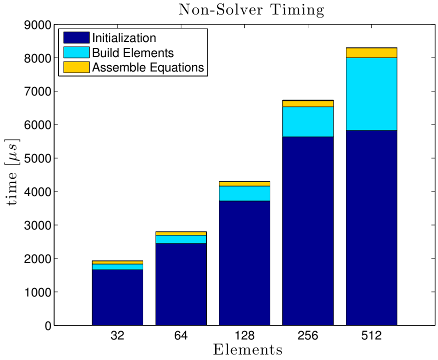




- Convergence of 1st and second order elements is approximately order 2 and 3 respectively, in agreement with theory
- Higher order elements can provide significant gains in accuracy for sufficiently smooth problems

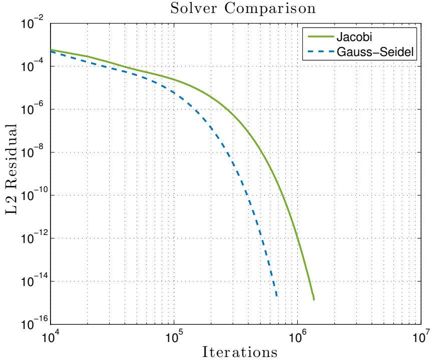
Timing and Performance

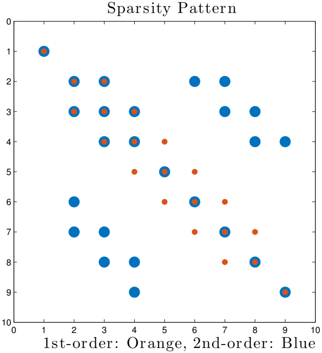




- Program runtime is dominated by the solver routines
- Initialization (parsing input, reserving memory, initializing masa) takes the majority of the rest of the program time
- Meshing and creating nodes, edges and elements takes more time than performing the integrations

Timing and Performance





- Gauss-Seidel converges in fewer iterations than the jacobi solver
- The jacobi solver does not converge for 2nd order finite elements
- Using an eigen built-in Conjugate Gradient solver saw 3000x speedup (vs. GS, n = 256, 2nd order)

Conclusions ...

- Began by taking a basic program structure for the 2D problem and implementing in matlab (1 afternoon)
- C++ coding began with getting unit testing framework setup
- Tests written at the same time as objects
  - Very useful (except when you write your tests wrong)
- Despite having a solid structure in place before beginning, coding went much slower than expected
- Late integration of MASA caused some headaches with evaluating forcing and boundary conditions
- Code is very close to being able to convert to 2d, however the work to make this flexible limited my chances to do more in depth profiling/optimization

Thank you!

Questions?