A Verification Driven Process for Rapid Development of CFD Software

Marshall C. Galbraith
galbramc@mit.edu

Steven R. Allmaras
allmaras@mit.edu

David L. Darmofal
darmofal@mit.edu
Motivation

- Begun development of new CFD software
  - Focus on finite elements and mesh adaptation
- Verification Driven Development
  - Principals of Verification and Validation
  - Ensure functionality
  - Significantly reduce debugging time
- Continuous Integration
  - Collaboration
  - Share progress, but not mistakes
- Automatic Differentiation
  - Linearization is tedious and error prone for CFD
Outline

- Verification Driven Development

- Continuous Integration

- Automatic Differentiation

Bugging today, debugging tomorrow

Teamwork
Working harder not smarter

Just because you have to, does not mean you need to.
Verification Driven Development

- **Verification**: Are the equations solved correctly?
- **Validation**: Are the correct equations solved?
- Outline of OUR adoption of Verification
  - Build configurations
    - Different machines, Different Compilers
  - Building/Executing unit tests
    - Unit test make targets
  - Coverage information
    - GNU + Lcov
  - Memory checking
    - Valgrind
  - Uniform coding style
    - Vera++ style checking
  - Static Analysis
    - Cppcheck & Clang
  - Regression testing
    - Method of Manufactured Solutions
Build Configurations

• Build Configuration Software
  – CMake, autotools, Scons, Waf, etc.
  – Finds dependencies
  – Machine specific customize configurations

• Not all compilers are created equal

• CMake automatic configuration
  – GNU, Intel, Clang
  – Enable compiler warnings: Wall, Wextra, Weverything

• Build directory guides configuration

  **Commands GNU+debug flags**

  ```
  cd SANS
  mkdir -p build/debug_gnu
  cd build/debug_gnu
  cmake ../..
  ```

  ```
  build/
  debug_gnu
  release_gnu
  debug_intel
  release_intel
  debug_clang
  release_clang
  ```
Unit Test Make Targets

• Unit Testing
  – Ensures functionality of the code
    • Given known input check for known output
  – Want lots of them
  – Easy to create/execute
  – Many developers create them intuitively with print statements

• Unit testing frameworks simplify testing
  – C++: Boost-test, Google-test
  – C: CuTest, Cunit
  – Fortran: pFUnit, FRUIT
  – Matlab, Python: Built-in

• All unit tests in unit folder
  – unit folder structure mimics src folder

• Executables automatically created for each file/folder
• Automatically added to overall unit executable
Unit Test Make Targets

**Unit test folder structure**

unit/DenseLinAlg/
  CMakeLists.txt
  MatrixS_btest.cpp
  MatrixD_btest.cpp

unit/Surreal/
  CMakeLists.txt
  SurrealS1_btest.cpp
  SurrealS4_btest.cpp
  SurrealD1_btest.cpp
  SurrealD4_btest.cpp

**Command Line:**

```
  cd build/debug_gnu
  cmake .
  make SurrealD1_btest
  make Surreal
  make unit
```

**Brand new unit test file just created**

```cpp
BOOST_AUTO_TEST_SUITE( SurrealD1_test_suite )

BOOST_AUTO_TEST_CASE( add )
{
  SurrealD v1(1, 2), v2(v1);
  SurrealD v3 = v1 + v2;
  BOOST_CHECK_EQUAL( 2, v3.value() );
  BOOST_CHECK_EQUAL( 4, v3.deriv(0) );
  BOOST_CHECK_EQUAL( 1, v3.size() );
}

BOOST_AUTO_TEST_SUITE_END()
```
Coverage Information

• Unit tests ensure that tested code is working
• How do you KNOW what is tested?
  – Coverage Information
• GNU gcov and lcov
  – g++ --coverage
  – http://ltp.sourceforge.net/coverage/lcov.php

Commands GNU+coverage

```
mkdir -p build/coverage
cd build/coverage
cmake ../..
make unit
make coverage
make coverage_show
```
## Lcov Coverage Information

### LCOV - code coverage report

<table>
<thead>
<tr>
<th>Directory</th>
<th>Line Coverage</th>
<th>Functions</th>
<th>Branches</th>
</tr>
</thead>
<tbody>
<tr>
<td>BasisFunction</td>
<td>95.7 %</td>
<td>84.0 %</td>
<td>23.7 %</td>
</tr>
<tr>
<td>DenseLinAlg</td>
<td>100.0 %</td>
<td>100.0 %</td>
<td>-</td>
</tr>
<tr>
<td>DenseLinAlg/DynamicSize</td>
<td>99.1 %</td>
<td>98.0 %</td>
<td>35.1 %</td>
</tr>
<tr>
<td>DenseLinAlg/DynamicSize/MatMul</td>
<td>98.8 %</td>
<td>87.5 %</td>
<td>32.8 %</td>
</tr>
<tr>
<td>DenseLinAlg/DynamicSize/lapack</td>
<td>100.0 %</td>
<td>100.0 %</td>
<td>28.4 %</td>
</tr>
<tr>
<td>DenseLinAlg/StaticSize</td>
<td>100.0 %</td>
<td>93.3 %</td>
<td>51.3 %</td>
</tr>
<tr>
<td>DenseLinAlg/StaticSize/MatMul</td>
<td>97.5 %</td>
<td>98.9 %</td>
<td>52.2 %</td>
</tr>
<tr>
<td>DenseLinAlg/tools</td>
<td>100.0 %</td>
<td>90.0 %</td>
<td>85.7 %</td>
</tr>
<tr>
<td>Python</td>
<td>100.0 %</td>
<td>94.4 %</td>
<td>43.0 %</td>
</tr>
<tr>
<td>Quadrature</td>
<td>100.0 %</td>
<td>100.0 %</td>
<td>55.0 %</td>
</tr>
<tr>
<td>Residual</td>
<td>79.3 %</td>
<td>93.4 %</td>
<td>33.2 %</td>
</tr>
<tr>
<td>SparseLinAlg</td>
<td>99.6 %</td>
<td>88.6 %</td>
<td>34.6 %</td>
</tr>
<tr>
<td>SparseLinAlg/Direct</td>
<td>96.0 %</td>
<td>97.8 %</td>
<td>50.0 %</td>
</tr>
<tr>
<td>SparseLinAlg/Krylov</td>
<td>97.1 %</td>
<td>61.3 %</td>
<td>23.5 %</td>
</tr>
<tr>
<td>SparseLinAlg/Preconditioners</td>
<td>100.0 %</td>
<td>51.7 %</td>
<td>33.3 %</td>
</tr>
<tr>
<td>SparseLinAlg/tools</td>
<td>100.0 %</td>
<td>88.9 %</td>
<td>62.5 %</td>
</tr>
<tr>
<td>Surreal</td>
<td>99.7 %</td>
<td>98.4 %</td>
<td>39.2 %</td>
</tr>
<tr>
<td>pde/CauchyRiemann</td>
<td>83.3 %</td>
<td>57.1 %</td>
<td>25.4 %</td>
</tr>
<tr>
<td>pde/NS</td>
<td>93.8 %</td>
<td>96.6 %</td>
<td>46.6 %</td>
</tr>
<tr>
<td>tools</td>
<td>82.0 %</td>
<td>80.0 %</td>
<td>49.5 %</td>
</tr>
</tbody>
</table>

**Generated by:** [LCOV version 1.10](https://lcamtuf.ccc.in.id/)
**lcov Coverage Information**

---

**LCOV - code coverage report**

Current view: top level - tools - SANSException.cpp (source / functions)  
Test: coverage.info  
Date: 2014-05-02

<table>
<thead>
<tr>
<th>Lines</th>
<th>Hit</th>
<th>Total</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>14</td>
<td>29</td>
<td>48.3 %</td>
</tr>
<tr>
<td>Functions</td>
<td>5</td>
<td>7</td>
<td>71.4 %</td>
</tr>
<tr>
<td>Branches</td>
<td>13</td>
<td>50</td>
<td>26.0 %</td>
</tr>
</tbody>
</table>

Legend:  
| Lines: hit | not hit | Branches: + taken | - not taken | # not executed |

---

**Source Code:**

```cpp
#include "SANSException.h"
#include "iostream"
#include "stdarg.h"

int main() {
    // Your code goes here...
}
```
Memory Checking

- Valgrind
  - Memory leaks
  - Uninitialized variables
  - Array bounds
  - Profiling
  - More...

Command Line:
```
cd build/debug_gnu
make SurrealD1_btest_memcheck
make Surreal_memcheck
make unit_memcheck
make memcheck
```

Run valgrind on any executable by adding _memcheck to the make target
Vera++ Style Checking

- Enforce consistent formatting across developers
  - Brace Placement
  - Spacings
  - Maximum Line Length
  - No Trailing Spaces
  - No Tabs
  - and more...

- All .h and .cpp files automatically checked with CMake
Outline

• Verification Driven Development
  Bugging today, debugging tomorrow

• Continuous Integration
  Teamwork
  Working harder not smarter

• Automatic Differentiation
  Just because you have to, does not mean you need to.
Continuous Integration

- Collaborative Development of Software
  - Source Code Management software (cvs, svn, git, mercurial, etc.)
  - Share progress, but not mistakes
    - Automatic execution of tests
  - Collaborate on new features
    - Commit anytime and often (small incremental changes)
  - Testing broken code on multiple machines
    - It works on my computer, why not on the testing server?
  - Always have access to code that has passed ALL tests

- Pre-commit Checking
- Post-commit Checking
- Pre-merge Checking
Continuous Integration
Pre-commit Checking

- Nothing is allowed into the repository unless ALL tests pass!
- Manual Workstation vs. Server Checks
  - Manual Workstation Testing
    - Developers are asked to check before committing
    - Humans are poor automatic systems
  - Server Testing
    - Automated

✓ Code in repository always passes ALL tests! (this is a big deal!)

✗ Collaboration
  ✓ Can't commit broken code

✗ Discourages small incremental changes
  ✓ Can't commit broken code

✗ Testing Broken Code on Multiple Machines
  ✓ Can't commit broken code
Continuous Integration
Post-Commit Checking

- Everything is always accepted into the repository
- Tests run automatically after commit

✔ Runs All Test Automatically
✔ Notification of Errors
✔ Possible to collaborate

✗ Errors Committed to trunk
  ✓ Code with errors is shared with all
  ✗ Collaboration (can't commit broken code)
  ✓ Comment/Uncomment code before committing

✗ Discourages Small Incremental Changes
  ✓ Reluctance to Commit before Completion of Feature

✗ Testing Broken Code on Multiple Machines
  ✓ Tests Pass/Fail on different machines. Difficult to Synchronize.
✗ No Guarantee that Code in Repository Passes ALL tests
Continuous Integration
Pre-merge Checking

- Everything is always accepted into the repository
- Code is only shared when tests pass
  - Unit+regression tests with GNU, Clang, and Intel
  - Coverage, Vera++, Cppcheck
  - Two Machines: Ubuntu and OSX
- Leverages branching/merging abilities of git
  - Multiple versions of code in one repository
- Personal working branches
  - galbramc/develop
  - allmaras/develop
- develop branch used to share code amongst developers
- Server merges working branches into develop if tests pass
- Developers update from the develop branch
- master branch holds code that has passed ALL tests
- Possible to implement with svn repositories as well
Jenkins Continuous Integration

• Jenkins ( jenkins-ci.org )
  – Large user base
  – Web based interface
  – Simple to install
  – Simple to configure

• Pre-Tested Commits with Jenkins
  – https://www.youtube.com/watch?v=LvCVw5gnAo0
Continuous Integration Pre-merge Checking

- Server merges passing commits from personal working branches into develop

```
allmaras@peclet.mit.edu
allmaras/develop

acdl.mit.edu
develop
galbramc/develop
allmaras/develop

Commit changes to galbramc/develop
galbramc/galbramc/develop

allmaras@reynolds.mit.edu
allmaras/develop

jenkins@reynolds.mit.edu
Server merges into develop
Runs tests on develop

galbramc@gripen.mit.edu
galbramc/develop
```
Continuous Integration
Pre-merge Checking

- Server merges passing commits from personal working branches into develop.

Test Pass!
Push to acdl

Test Pass!
Run tests on develop

Server merges into develop

acdl.mit.edu
allmaras/develop
galbramc@gripen.mit.edu
galbramc/develop

allmaras@peclet.mit.edu
allmaras/develop

jenkins@reynolds.mit.edu
Server merges into develop

allmaras@reynolds.mit.edu
allmaras/develop
Continuous Integration
Pre-merge Checking

- Server merges passing commits from personal working branches into develop

```
allmaras@peclet.mit.edu
allmaras/develop

Updates from galbramc
Nothing to update

acdl.mit.edu
develop
galbramc/develop
allmaras/develop

allmaras@reynolds.mit.edu
allmaras/develop

galbramc@gripen.mit.edu
galbramc/develop

jenkins@reynolds.mit.edu
Server merges into develop
Runs tests on develop
```
Continuous Integration
Pre-merge Checking

- Code changes that fail tests

```
allmaras@peclet.mit.edu
allmaras/develop
Commit changes to allmaras/develop

acdl.mit.edu
develop
galbramc/develop
allmaras/develop

jenkins@reynolds.mit.edu
Server merges into develop
Runs tests on develop

galbramc@gripen.mit.edu
galbramc/develop
```
Continuous Integration
Pre-merge Checking

- Code changes that fail tests
Continuous Integration
Pre-merge Checking

• Code changes that fail tests

- Broken code not in develop. Nothing to update.
- Nothing to update from galbramac/develop.
- Broken code only in allmaras/develop.
- Server merges into develop.
- Runs tests on develop.
Continuous Integration
Pre-merge Checking

• Code changes that fail tests
Continuous Integration
Pre-merge Checking

- Code changes that fail tests

- Commit fixed code to allmaras/develop

- Server merges into develop
  - Runs tests on develop

- adcl.mit.edu
  - develop
  - galbramc/develop
  - allmaras/develop

- galbramc@gripen.mit.edu
  - galbramc/develop

- allmaras@peclet.mit.edu
  - allmaras/develop

- allmaras@reynolds.mit.edu
  - allmaras/develop

- jenkins@reynolds.mit.edu
  - Server merges into develop
  - Runs tests on develop
Continuous Integration
Pre-merge Checking

- Code changes that fail tests

![Diagram]

- allmaras@peclet.mit.edu
  allmaras/develop

- acdl.mit.edu
  develop
  galbramc/develop
  allmaras/develop

- jenkins@reynolds.mit.edu
  Server merges into develop
  Runs tests on develop

- galbramc@gripen.mit.edu
  galbramc/develop

- allmaras@reynolds.mit.edu
  allmaras/develop

Test Pass!
Push to acdl
Continuous Integration
Pre-merge Checking

- Code changes that fail tests
Continuous Integration
Pre-merge Checking

• Collaboration on code that does not pass test

- allmaras@peclet.mit.edu
  allmaras/develop

- Commit failing code to allmaras/develop

- acdl.mit.edu
  develop
  galbramc/develop
  allmaras/develop

- galbramc@gripen.mit.edu
  galbramc/develop

- Test Fail
  Email allmaras

- jenkins@reynolds.mit.edu
  Server merges into develop
  Runs tests on develop

- allmaras@reynolds.mit.edu
  allmaras/develop
Continuous Integration
Pre-merge Checking

- Collaboration on code that does not pass test
Continuous Integration
Pre-merge Checking

- Collaboration on code that does not pass test
Continuous Integration

- Pre-merge Checking

- Collaboration on code that does not pass test
Continuous Integration
Pre-merge Checking

- master branch holds code that has passed ALL tests

Nightly Testing

- Code in apprentice has passed commit and nightly tests

  Test Pass!
  Push to apprentice

  Nightly Tests on develop

Weekly Testing

- Code in master has always passed ALL tests

  Test Pass!
  Push to master

  Weekly Tests on apprentice

- Test Fail
  Email team

  reynolds.mit.edu
  Runs tests on develop

  reynolds.mit.edu
  Runs tests on apprentice

  acdl.mit.edu
  master
  apprentice
develop
galbramc/develop
allmaras/develop

Massachusetts Institute of Technology
Outline

- Verification Driven Development
  
  Bugging today, debugging tomorrow

- Continuous Integration
  
  Teamwork
  Working harder not smarter

- Automatic Differentiation
  
  Just because you have to, does not mean you need to.
Types of Differentiation

- Hand Differentiation
- Symbolic Differentiation
  - Symbolic Manipulation Software (Mathematica, Maple, etc.)
  - Translate to Code
- Numerical Differentiation
  - Finite Difference
  - Complex Step
- Automatic Differentiation
  - Analytically Differentiates Code Directly from Source
  - Preprocessor
  - Operator Overloading
Automatic Differentiation

- Write source code for primary function
- Derivative function source code generated automatically

- Preprocessor
  - Parses code and generates differentiated code
  - ADIFOR, ADIC and many more
  - Often requires 'tweaking' of generated code

- Operator Overloading
  - Custom data type to track values and derivatives
  - Overloaded operators apply chain rule
  - Packages are available (Wikipedia has a long list)
Automatic Differentiation
Operator Overloading

Function
\[
\frac{\partial f}{\partial x_1} = a + x_2
\]
\[
\frac{\partial f}{\partial x_2} = x_1
\]

Surreal Datatype
\[
s = \left[ \begin{array}{c} \hat{s} \\ \frac{\partial s}{\partial x_1}, \frac{\partial s}{\partial x_2} \end{array} \right]
\]
\[\text{value}\]
\[\text{derivatives}\]

Instances of the variables
\[
x_1 = \left[ \begin{array}{c} \hat{x}_1 \\ \frac{\partial x_1}{\partial x_1}, \frac{\partial x_1}{\partial x_2} \end{array} \right] = \left[ \begin{array}{c} \hat{x}_1 \\ [1, 0] \end{array} \right]
\]
\[
x_2 = \left[ \begin{array}{c} \hat{x}_2 \\ \frac{\partial x_2}{\partial x_1}, \frac{\partial x_2}{\partial x_2} \end{array} \right] = \left[ \begin{array}{c} \hat{x}_2 \\ [0, 1] \end{array} \right]
\]

Computing function and derivatives
\[
f = a x_1 + x_1 x_2
\]
\[
= a \left[ \begin{array}{c} \hat{x}_1 \\ [1, 0] \end{array} \right] + \left[ \begin{array}{c} \hat{x}_1 \\ [1, 0] \end{array} \right] \left[ \begin{array}{c} \hat{x}_2 \\ [0, 1] \end{array} \right]
\]
\[
= a \left[ \begin{array}{c} \hat{x}_1 \\ [a, 0] \end{array} \right] + \left[ \begin{array}{c} \hat{x}_1 \hat{x}_2 \\ [1, 0] \hat{x}_2 + \hat{x}_1 [0, 1] \end{array} \right]
\]
\[
= a \left[ \begin{array}{c} \hat{x}_1 \\ [a, 0] \end{array} \right] + \left[ \begin{array}{c} \hat{x}_1 \hat{x}_2 \\ [\hat{x}_2, \hat{x}_1] \end{array} \right]
\]
\[
= a \left[ \begin{array}{c} \hat{x}_1 + \hat{x}_1 \hat{x}_2 \\ [a + \hat{x}_2, \hat{x}_1] \end{array} \right]
\]
\[
= \left[ \begin{array}{c} \hat{f} \\ \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2} \end{array} \right]
\]
template<class T>
void flux( const T *q, double nx, double ny, double gam, double l, int n, T *)&
{
  T cs, ubar, u, v, p;
  ubar = nx*q[1]/q[0]+ny*q[2]/q[0];
  u = q[1]/q[0];
  v = q[2]/q[0];
  p = (gam - 1.0)*(q[3] - .5*(q[1]*q[1] + q[2]*q[2])/q[0]);
  cs = pow((gam*p)/q[0],.5);
  f[0] = l*0.25*q[0]*cs*(ubar/cs+1.0)*(ubar/cs+1.0);
  f[1] = f[0]*((-1.0*ubar)+(nx/gam)*(2.0*cs)+u);
  f[2] = f[0]*((-1.0*ubar)+(ny/gam)*(2.0*cs)+v);
  f[3] = f[0]*((gam-1.0)*ubar*ubar+2.0*(gam-1.0)*ubar*cs+2.0*cs*cs)/
       (gam*gam-1.0)+0.5*(u*u)+0.5*(v*v));
}

void Jacobian( double q[], double nx, double ny, double gam, double l, int n, double dfp_dq[][4] )
{
  double ubar,c,fp1,rho,u,v,e,p;
  double rho_q[],u_q[],v_q[],p_q[],ubar_q[],c_q[],e_q[];  
  rho = q[0];  u = q[1]/rho;  v = q[2]/rho;  e = q[3];  p = (gam-1.0)*(e-0.5*rho*(u*u+v*v));
  c = sqrt((gam*(gam-1.0)*(q[3]-0.5*(q[1]*q[1]+q[2]*q[2])/q[0])/q[0]);
  ubar = nx*q[1]/q[0]+ny*q[2]/q[0];
  fp1 = l*0.25*rho*c*pow((ubar/c+1.0),2);
  for( int i=1; i < n; i++ ) { rho_q[i] = 0.0; e_q[i] = 0.0; }
  rho_q[0] = 1.0;  
  e_q[3] = 1.0;
  u_q[0] = -q[1]/(q[0]*q[0]);  u_q[1] = 1.0/q[0];  u_q[2] = 0.0;  u_q[3] = 0.0;
  v_q[0] = -q[2]/(q[0]*q[0]);  v_q[1] = 0.0;  v_q[2] = 1.0/q[0];  v_q[3] = 0.0;
  p_q[0] = 0.5*(gam-1.0)*(q[1]*q[1]+q[2]*q[2])/q[0];  
  p_q[1] = -(gam-1.0)*q[1]/q[0];  
  p_q[2] = -(gam-1.0)*q[2]/q[0];  p_q[3] = gam-1.0;
  for( int i=0; i < n; i++ ) {
    ubar_q[i] = nx*u_q[i]+ny*v_q[i];
    c_q[i] = 0.5*(gam/c)*(p_q[i]/rho-p*rho_q[i]/(rho*rho));
  }
  for( it i = 0; i < n; i++ ) {
    dfp_dq[0][i]= l*0.25*((rho_q[i]*c+rho*c_q[i])*pow(ubar/c+1.0,2)+
      rho*c**2.0*(ubar/c+1.0)*(ubar_q[i]+c-ubar*c_q[i]/(c*c)));  
    dfp_dq[1][i] = dfp_dq[0][i]**((nx/gam)*(ubar+2.0*c)+u)+fp1*(nx/gam)**(ubar_q[i]+2.0*c_q[i])+u_q[i];
    dfp_dq[2][i] = dfp_dq[0][i]**((ny/gam)*(ubar+2.0*c)+v)+fp1*(ny/gam)**(ubar_q[i]+2.0*c_q[i])+v_q[i];
    dfp_dq[3][i] = dfp_dq[0][i]**((gam-1.0)**(ubar+2.0*c)+2.0*c_c)+0.5*(u*u+v*v)+fp1*((gam-1.0)**(ubar+2.0*c)+2.0*c_q[i]+v_q[i]);
  }
}

Compute the Flux+Jacobian: Hand Coded

fluct<double>(q,nx,ny,gam,l,n,f);

Jacobian(q,nx,ny,gam,l,n,df_dq);

Compute the Flux+Jacobian: Automatic

//f will contain both flux and jacobian
fluct< Surreal<4> >(q,nx,ny,gam,l,n,f);
## Automatic Differentiation Timing

### van Leer Flux 100x10^6 Evaluations

<table>
<thead>
<tr>
<th></th>
<th>Hand Coded Flux+Jacobian</th>
<th>Automatic</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C (s)</td>
<td>Fortran (s)</td>
<td>C++ (s)</td>
</tr>
<tr>
<td>GNU</td>
<td>8.35</td>
<td>10.55</td>
<td>15.04</td>
</tr>
<tr>
<td>Intel</td>
<td>10.72</td>
<td>10.46</td>
<td>7.65</td>
</tr>
<tr>
<td>Clang</td>
<td>11.61</td>
<td>-</td>
<td>11.72</td>
</tr>
</tbody>
</table>

### Roe Flux 25x10^6 Evaluations

<table>
<thead>
<tr>
<th></th>
<th>Hand Coded Flux+Jacobian</th>
<th>Automatic</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C (s)</td>
<td>Fortran (s)</td>
<td>C++ (s)</td>
</tr>
<tr>
<td>GNU</td>
<td>5.18</td>
<td>5.15</td>
<td>5.99</td>
</tr>
<tr>
<td>Intel</td>
<td>4.45</td>
<td>5.18</td>
<td>6.06</td>
</tr>
<tr>
<td>Clang</td>
<td>5.10</td>
<td>-</td>
<td>7.83</td>
</tr>
</tbody>
</table>
Thank you
Back Slides

LISP IS OVER HALF A CENTURY OLD AND IT STILL HAS THIS PERFECT, TIMELESS AIR ABOUT IT.

I WONDER IF THE CYCLES WILL CONTINUE FOREVER.

A FEW CODERS FROM EACH NEW GENERATION RE-DISCOVERING THE LISP ARTS.

THESE ARE YOUR FATHER’S PARENTHESES.

ELEGANT WEAPONS FOR A MORE... CIVILIZED AGE.
Example Unit Test
Linear Algebra

BOOST_AUTO_TEST_SUITE( SparseLinAlg )

BOOST_AUTO_TEST_CASE( LinearSolver_UMFPACK )
{
    LinearSolver< Matrix_TriDiag > Solver( "UMFPACK" );

    //Create the sparse matrix and vectors
    const unsigned int nRow = 5;
    Matrix_TriDiag A(nRow);
    Vector x(nRow), b(nRow), b2(nRow);

    //Initialize the tri-diagonal matrix with rows of -1, 2, -1
    A.init(-1, 2, -1);

    //Create a vector
    b[0] = 0.5;
    b[1] = 1;
    b[2] = 2;
    b[3] = 1;
    b[4] = 0.5;

    //Solve the linear system Ax = b.
    x = Solver.inverse(A)*b;

    //Compute another vector from the solution
    b2 = A*x;

    //The vectors should now be the same!
    for (unsigned int i = 0; i < nRow; i++)
        BOOST_CHECK_CLOSE( b[i], b2[i], 1e-12 );
}

BOOST_AUTO_TEST_SUITE_END()
All Headers Compile

- All header files compile independently
- Avoid complicated include patterns
- Avoid copies of “chucks” of includes
- CMake automatically generates .h.cxx files with only one include statement and compiles that file
Example Unit Test
Linear Algebra

BOOST_AUTO_TEST_CASE( LinearSolver_UMFPACK )
{
    typedef SparseMatrix_CRS<double> Matrix_type;
    LinearSolver< Matrix_type, double> Solver("UMFPACK");

    //Create the sparse matrix and vectors
    const unsigned int nRow = 5;
    std::shared_ptr< Matrix_type >
        A( new SparseMatrix_TriDiag<double>(nRow) );
    SparseVector<double> x(nRow), b(nRow), b2(nRow);

    //Initialize the tri-diagonal matrix with rows of -1, 2, -1
    Heat1D<double>::init(A);

    //Create a vector
    b[0] = 0.5;
    b[1] = 1;
    b[2] = 2;
    b[3] = 1;
    b[4] = 0.5;

    //Solve the linear system Ax = b.
    x = Solver.inverse(A)*b;

    //Compute another vector from the solution
    b2 = A*x;

    //The vectors should now be the same!
    for (unsigned int i = 0; i < nRow; i++)
        BOOST_CHECK_CLOSE( b[i], b2[i], 1e-12 );
}
Pros/Cons
Pre-merge Checking

✔ Runs All Test Automatically
✔ Notification of Errors
✔ Errors are not Shared Automatically
✔ Collaboration
✔ Commit Often with Small Incremental Changes
✔ Synchronizing for Development on Multiple Machines
✔ Holds Code that Passes ALL Tests

✗ Standard 'git pull' not robust when merging multiple branches
   × Custom 'git update' when working on personal branch
✗ Custom configuration required for each cloned repository
   × Script for cloning and automated configuration