Verification of Complex Codes

Curtis Ober

Exploratory Simulation Technology
Sandia National Laboratories
Albuquerque, New Mexico
ccober@sandia.gov

Oil and Gas High Performance Computing Workshop 2009
Thursday, March 5, 2009
Rice University

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.
Starting Point to Verification


The T Experiments:
Errors In Scientific Software

LES HATTON
Oakwood Computing

Extensive tests showed that many software codes widely used in science and engineering are not as accurate as we would like to think. Better software engineering practices would help solve this problem, but realizing that the problem exists is an important first step.

• Major Points
  – Why should we trust complex codes?
  – Nine seismic migrations $\rightarrow$ results varied (one significant digit)
V&V Definitions

• **Verification** (“Is the model working correctly?”)
  – From the AIAA V&V Guide (similar definition from ASME)
    • Verification is the process of determining that a model implementation accurately represents the developer’s conceptual description of the model and the solution to the model.
  – Mathematical and numerical question

• **Validation** (“Is it the correct model?”)
  – From the AIAA V&V Guide (exactly the same definition for ASME)
    • Validation is the process of determining the degree to which a model is an accurate representation of the real world from the perspective of the intended uses of the model.
  – Physics and experimental question
Levels of Verification

• **Benchmarks**
  - Comparison against
    • Standard solutions (analytic, series expansions, simple geometries or BCs)
    • Legacy/reference codes
    • Experimental data? This usually causes confusion with validation.
  - Error measured by “Are you on the ballpark?” or “eye-ball norm”
  - These tests are very useful during early development of software
  - Single temporal and spatial refinement
  - Many verification studies stop here

• **Solution Verification**
  - Temporal and/or spatial refinement study
  - Richardson-extrapolated solution as the reference solution
  - *Approximate error* is generated from Richardson-extrapolated solution
  - Useful in real simulations of complex geometries and/or BCs

• **Order Verification**
  - Temporal and/or spatial refinement study
  - Exact solutions as the reference solution
    • Analytic solutions (usually simplified geometries, BCs, …)
    • Method of Manufactured Solutions (MMSs) (complex geometries, BCs, …)
  - *Exact error* is known
  - Can generally test all the code capabilities
\[ \frac{\partial E}{\partial t} - \frac{\partial}{\partial x} \left( D_e \frac{\partial E}{\partial x} \right) = \sigma_a \left( T^4 - E \right) \]

\[ \frac{\partial T}{\partial t} = -\sigma_a \left( T^4 - E \right) \]

\[ D_e = \frac{1}{\left( 3\sigma_a + \frac{1}{E} \left| \frac{\partial E}{\partial x} \right| \right)} , \sigma_a = \frac{1}{T^3} \]

E – Radiation energy density
T – Temperature
Dr – Flux-limited diffusion coefficient

(Ober and Shadid, JCP, 2004)
Compressible Fluid Dynamics

Used the Method of Manufactured Solutions (MMS) to investigate

- Gradient calculations
- Boundary conditions

(Bond, Ober and Knupp, AIAA Journal 2007)
Magneto-hydrodynamics

- Operator-split MHD problem
  - 1D, Lagrangian slab
  - Aluminum 6061-T6
  - MG US UP
  - Knoepfel electrical conductivity

- Solution verification using Richardson Extrapolation

- Supercycling ($\propto$ hydro/ 1 mag)
Study Motivation

• Illustrate the impact of bugs on the numerical error (NE) and uncertainty.
• NE is the difference between numerical and exact solution.
  – Discretization error (DE)
  – Round-off error (RE)
  – Incomplete iterative convergence error (IICE)
  – Implementation correctness error (ICE) (coding mistakes or bugs)
    \[ NE = F(DE, RE, IICE, ICE) \]
• ICE
  – Is an epistemic uncertainty (lack-of-knowledge uncertainty)
  – May be small or large compared to DE
    • Small => ICE can be ignored
    • Large => ICE is important!
• Not all bugs create NE, but the ones that do should be detected.
• Bugs should be eliminated from codes, but this study shows what happens if a bug slips through the testing process.
We Used a Model Code Instead of a Real Code

• Ideally would perform study on real production codes.

• Impractical to use large complex physics code
  – Need to remove “ALL” bugs from the code
  – Requires large computational effort (asymptotic range)
  – Complicated by mixed-order methods (boundary conditions)
  – Possibility of obscuring any general trends

• Therefore solve model problem (1D Poisson’s equation)
  – Pros
    • By-pass above difficulties
    • Exact solution allows evaluation of NE and uncertainty
  – Cons
    • Careful extrapolation of results to real codes (later slides)
Our “Complex” Model Problem

A simple governing equation to mimic complex PDEs

\[ u_{xx} = f \quad x = [a, b] \]

with boundary conditions

\[ \left. \frac{\partial u}{\partial x} \right|_{x=a} = \gamma \]

\[ u(b) = U_b \]

where

\[ a = -1.2 \]
\[ b = 3.2 \]
\[ \gamma = 2.0 \]
\[ U_b = 0.12 \]

Exact solution is known
The Model Problem

- Exact Solution
  \[ u(x) = \alpha + \beta x - L^2 e^{x/L} \]
  \[ \begin{align*}
  \alpha &= U_b - \beta b + L^2 e^{b/L} \\
  \beta &= \gamma + L e^{a/L} \\
  L &= b - a
  \end{align*} \]

- Allows easily evaluation of
  - Errors
  - System response quantities (SRQs)

- Caveats for a simplistic linear model problem
  - Degree to which observations carry over to complex codes
  - Magnitude of ICE compares to modeling and experimental errors
The “Bug 0” Code

• Implemented in Fortran and C
• Simple finite difference
  – Second order
  – Uniform spacing
• Gauss-Siedel Iteration
  – Tolerance $\Delta u < 10^{-14}$
• Boundary Condition
  – Second order
  – One-sided differencing

```c
// Initialization
double beta = gamma + ell*exp(a/ell);
double alpha = ub - beta*b + pow(ell,2.0)*exp(b/ell);
for (int k=0; k<n; k++) {
    x[k] = a + k*dx;
    u[k] = x[k];
    f[k] = -exp(x[k]/ell);
    uexact[k] = alpha + beta*x[k] - pow(ell,2.0)*exp(x[k]/ell);
}

// BCs
u[n-1] = ub;

// Solve
int i = 0;
double dumax = 1.0 + tol;
while ( ( i < imax ) && ( dumax > tol) ) {
    i++;
    dumax = 0.0;
    int kbeg = 1;
    int kend = n-1;
    for (int k=kbeg; k<kend; k++) {
        double uold = u[k];
        u[k] = (u[k+1] + u[k-1] - dx*dx*f[k] )^0.5;
        double du = abs(u[k]-uold);
        if ( du > dumax ) dumax=du;
    }
    // BCs
double uold = u[0];
    u[0] = (-2.0*dx *gamma + 4.0*u[1] - u[2])/3.0;
    double du = abs(u[0]-uold);
    if ( du > dumax ) dumax=du;
```
Order Verification of “Bug 0” Code

• Second order
  – $L_2$-norm
    \[ e_2 = \sqrt{\frac{1}{N} \sum_k (U_k - u(x_k))^2} \]
    \[ e_2^r = \frac{e_2}{\sqrt{\int_a^b u^2(s)ds}} \]
  – $L_\infty$-norm
    \[ e_\infty = \max_k |U_k - u(x_k)| \]
    \[ e_\infty^r = \frac{e_\infty}{|u(x_{k_{\text{max}}})|} \]

• Monotone
  – No non-asymptotic range!
  – Linear problem
27 Samples from the Set of Codes with Realistic Bugs

<table>
<thead>
<tr>
<th>Bug No.</th>
<th>Incorrect Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$u(0) = (-2*, dx + \gamma + 4 + u(1) - u(2))/3$</td>
</tr>
<tr>
<td>2</td>
<td>$u(0) = (+2*, dx + \gamma + 4 + u(1) - u(2))/3$</td>
</tr>
<tr>
<td>3</td>
<td>$u(0) = (-2*, dx + dx + \gamma + 4 + u(1) - u(2))/3$</td>
</tr>
<tr>
<td>4</td>
<td>$u(0) = (-2*, dx + \gamma + 4 + u(1) - u(2))/3$</td>
</tr>
<tr>
<td>5</td>
<td>$u(0) = (+2*, dx + \gamma + 4 + u(1) + u(2))/3$</td>
</tr>
<tr>
<td>6</td>
<td>$u(0) = (-2*, dx + \gamma + 4 + u(1) - u(2))/3$</td>
</tr>
<tr>
<td>7</td>
<td>$u(0) = (-2*, dx + \gamma + 4 + u(1)/u(2))/3$</td>
</tr>
<tr>
<td>8</td>
<td>$u(0) = (+2*, dx + \gamma + 4 + u(1) - u(2))/4$</td>
</tr>
<tr>
<td>9</td>
<td>$u(0) = (-2*, dx + \gamma + 3 + u(1) - u(2))/3$</td>
</tr>
<tr>
<td>10</td>
<td>$u(k) = (u(k+1) + u(k-1) - (dx),f(k))/2$</td>
</tr>
<tr>
<td>11</td>
<td>$u(k) = (u(k+1) + u(k-1) - 2(dx)^2,f(k))/2$</td>
</tr>
<tr>
<td>12</td>
<td>$u(k) = (u(k+1) + u(k-1) - (dx)^2,f(k))/2$</td>
</tr>
<tr>
<td>13</td>
<td>$u(k) = (u(k+1) + u(k-1) - (dx)^2,f(k))/2$</td>
</tr>
<tr>
<td>14</td>
<td>$u(k) = (u(k+1) + u(k+1) - (dx)^2,f(k))/2$</td>
</tr>
<tr>
<td>15</td>
<td>$u(k) = (u(k+1) + u(k-1) - (dx)^2,f(k))/2$</td>
</tr>
<tr>
<td>16</td>
<td>$u(k) = (u(k+1) + u(k-1) - (dx)^2,f(k-1))/2$</td>
</tr>
<tr>
<td>17</td>
<td>$u(k) = (u(k+1) + u(k-1) - (dx)^2,f(k))/4$</td>
</tr>
<tr>
<td>18</td>
<td>$u(k) = (u(k+1) + u(k-1) + (dx)^2,f(k))/2$</td>
</tr>
<tr>
<td>19</td>
<td>do $k=2,n-1$</td>
</tr>
<tr>
<td>20</td>
<td>do $k=1,n-2$</td>
</tr>
<tr>
<td>21</td>
<td>$u(k) = -(u(k+1) + u(k-1) - (dx)^2,f(k))/2$</td>
</tr>
<tr>
<td>22</td>
<td>$u(k) = (u(k+1) + u(k-1) - (dxf(k))/2)$</td>
</tr>
<tr>
<td>23</td>
<td>Bug 5 and Bug 16</td>
</tr>
<tr>
<td>24</td>
<td>Bug 8 and Bug 16</td>
</tr>
<tr>
<td>25</td>
<td>Bug 10 and Bug 19</td>
</tr>
<tr>
<td>26</td>
<td>Bug 10 and Bug 17</td>
</tr>
</tbody>
</table>

• To study ICE, we created 27 samples/instantiations
  – Bugs in coefficients/signs, extra/wrong terms, operators, and indices. Example:
    $$u(k) = (u(k+1) + u(k-1) - (dx)^2\,f(k-1))/2$$
  – Three different lines contain bugs: Boundary Condition, Governing Equation, Loop Indices, and Combinations.
  – “Bug 0” is ICE free, and used to determine size of DE.

• In practice, we have 1 code instantiation, so does it have any bugs?
One Sample of the Solution Space

- How do you know it's the correct solution?
- May not have
  - Complete analysis, or
  - Experimental data to help us (we are mimicking a complex code).
27 Samples of the Solution Space

• Without testing, we could have gotten any one of these 27 solutions.

• Range of solutions is quite large, indicating ICE is important.
Strategies for Detecting Bugs in Samples

- The following will show what would happen if a single code were subjected to a sequence of strategies.

1. Numerical artifacts
   - Non-smooth behavior
   - Oscillations

2. Unphysical behavior
   - Negative or positive values
   - Satisfies boundary conditions
   - Value of the solution (e.g., x=a)

3. Solution verification
   - Compare against your best solution
     - Richardson Extrapolation
     - Best resolution

4. Order verification
   - Compare against the exact solution

- Similar to benchmark comparisons.
- Fairly cheap

- Rarely done
- Used for “important calculations”
Detection Strategy: Numerical Artifacts

• Failure criteria: kinks in the solution.
Detection Strategy: Numerical Artifacts

• Failure criteria: kinks in the solution.

• Eliminates
  – No Boundary Condition Bugs
  – Governing Equation Bugs 12, 14, 15, 17, and 21
  – Loop Bug 20
  – Combination Bugs 25 and 26

• Total samples eliminated: 8
Detection Strategy: Unphysical Behavior

• Failure criteria:
  – Negative flux at $x=a$
  – Negative $u(a)$
Detection Strategy: Unphysical Behavior

• Failure criteria:
  – Negative flux at \(x=a\)
  – Negative \(u(a)\)

• Eliminates
  – Boundary Condition Bugs 1, 2, 4, 5, 6 and 9
  – Governing Equation Bugs 13, 18 and 22
  – Loop Bug 19
  – Combination Bug 23

• Total samples eliminated: 11
• Failure criteria:
  – Negative flux at $x=a$
  – Negative $u(a)$

• Eliminates
  – Boundary Condition Bugs 1, 2, 4, 5, 6 and 9
  – Governing Equation Bugs 13, 18 and 22
  – Loop Bug 19
  – Combination Bug 23

• Total samples eliminated: 11
Detection Strategy: Solution Verification

- Error calculated with Richardson Extrapolation
- Failure criteria:
  - Slopes < two
- Solutions are converging but not to correct solution
Detection Strategy: Solution Verification

• Error calculated with Richardson Extrapolation
• Failure criteria:
  – Slopes < two
• Solutions are converging but not to correct solution
• Eliminates
  – Boundary Condition Bugs 3, 7 and 8
  – Governing Equation Bugs 10 and 16
  – No Loop Bugs
  – Combination Bug 24
• Total samples eliminated: 6
Detection Strategy: Solution Verification

• Error calculated with Richardson Extrapolation
• Failure criteria:
  – Slopes < two
• Solutions are converging but not to correct solution
• Eliminates
  – Boundary Condition Bugs 3, 7 and 8
  – Governing Equation Bugs 10 and 16
  – No Loop Bugs
  – Combination Bug 24
• Total samples eliminated: 6
• Two remaining 2nd-order accurate solutions are quite different so the remaining bug MUST be eliminated!!
Detection Strategy: Order Verification

- Error calculated using exact solution
- Failure criteria:
  - Slopes < two
Detection Strategy: Order Verification

- Error calculated using exact solution
- Failure criteria:
  - Slopes < two
- Eliminates
  - Boundary Condition Bugs 3, 7 and 8
  - Governing Equation Bugs 10, 11 and 16
  - No Loop Bugs
  - Combination Bug 24
- Total samples eliminated: 7
Detection Strategy: Order Verification

- Error calculated using exact solution
- Failure criteria:
  - Slopes < two
- Eliminates
  - Boundary Condition Bugs 3, 7 and 8
  - Governing Equation Bugs 10, 11 and 16
  - No Loop Bugs
  - Combination Bug 24
- Total samples eliminated: 7
Effectiveness of Detection Methods

<table>
<thead>
<tr>
<th>Detection Method</th>
<th>Numerical Artifacts</th>
<th>Unphysical Behavior</th>
<th>Solution Verification</th>
<th>Order Verification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential Detection</td>
<td>8 (31%)</td>
<td>19 (73%)</td>
<td>25 (96%)</td>
<td>26 (100%)</td>
</tr>
<tr>
<td>Gain</td>
<td></td>
<td>11</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>Individual Detection</td>
<td>8 (31%)</td>
<td>16 (62%)</td>
<td>19 (73%)</td>
<td>26 (100%)</td>
</tr>
</tbody>
</table>

• A large number of bugs can be detected using simple, cheap tests
• Sequential detection shows diminishing returns
• Only OV can eliminate them all (starting with OV gets 100%)
• ~25% of bugs converged at second-order accuracy with SV!
  (side note: ~50% of bugs converged at first-order accuracy with SV!)
• If one does not have the resources to do SV or OV, can one live with the 100-73% = 27% chance that a bug still exists in the code?
DE Error Bars on Samples can be Misleading

- Using GCI, $r=2$, $p=2$, and $F_s = 1.25$
- Error Bars for solutions with ICE do not overlap the exact solution.
  - A Richard Extrapolation assumption is violated
    - Asymptotically approaching correct solution
- Error Bars can give a false sense of accuracy for solutions with ICE, even though some
  - Show the correct trends
  - Are quite small
Eliminating half the bugs does not reduce the uncertainty by half

- Use 27 Samples to quantify the uncertainty due to ICE
- Methods for creating error bars
  - Full range
    \[ U^1_k = \frac{1}{2} \left[ \max_{j \in \text{Bugs}} (\rho_k)_j - \min_{j \in \text{Bugs}} (\rho_k)_j \right] \]
  - Standard deviation
    \[ U^2_k = 2\sigma \]
  - Quartile deviation
    \[ U^3_k = \frac{1}{2} (Q_{3,k} - Q_{1,k}) \]
- Uncertainties are not drastically reduced
  - 1/2 Bugs ≠ 1/2 ICE
- ICE is an epistemic uncertainty that can only be eliminated by using OV

Sequential Bug Detection Strategy
Summary

• For verification at Sandia, many projects use
  – Benchmarks → Cheap and effective, but only good to the eyeball norm
  – Solution Verification → More expensive, but moves beyond visual inspection
  – Order Verification → Guarantees to find bugs below the formal order of accuracy

• Additionally many projects use a variety of other tests
  – Unit Tests
  – Nightly Regressions
  – Performance Tests
  – Platform Testing