Work in Progress: Using Symbolic Execution to Formally Verify the Accuracy of Numerical Approximations

Timothy K. Zirkel Louis F. Rossi Stephen F. Siegel

University of Delaware

July 14, 2011

Automatic verification of order of accuracy

Using symbolic execution and theorem proving techniques, it is possible to provide automatic formal verification of the accuracy of a numerical program.

- Extend the Toolkit for Accurate Scientific Software (TASS)
  - Symbolic execution tool
  - `http://vsl.cis.udel.edu/tass`

Error in numerical programs

- From numerical method (discretization error)

Error in numerical programs

- From numerical method (discretization error)
- From floating-point computations (round-off error)

Error in numerical programs

- From numerical method (discretization error)
- From floating-point computations (round-off error)
- From defects

Error in numerical programs

- From numerical method (discretization error)
- From floating-point computations (round-off error)
- From defects

"To put it baldly, most scientific results are corrupted and perhaps fatally so by undiscovered mistakes in the software used to calculate and present those results." (Les Hatton)

Error in numerical programs

- From numerical method (discretization error)
- From floating-point computations (round-off error)
- From defects

> *"To put it baldly, most scientific results are corrupted and perhaps fatally so by undiscovered mistakes in the software used to calculate and present those results."* *(Les Hatton)*

This project is focused on the discretization error.

Big-O

Definition

Let $I = (0, a), a > 0$. Suppose we have two functions $\phi : I \to \mathbb{R}$ and $\psi : I \to \mathbb{R}$. We write

$$\phi(h) = O(\psi(h)) \text{ as } h \to 0$$

if there exist positive real numbers C and ϵ such that $|\phi(h)| \leq C|\psi(h)|$ whenever $0 < h < \epsilon$.

Order of accuracy

Let $D \subseteq \mathbb{R}, I = (0, a), a > 0$.

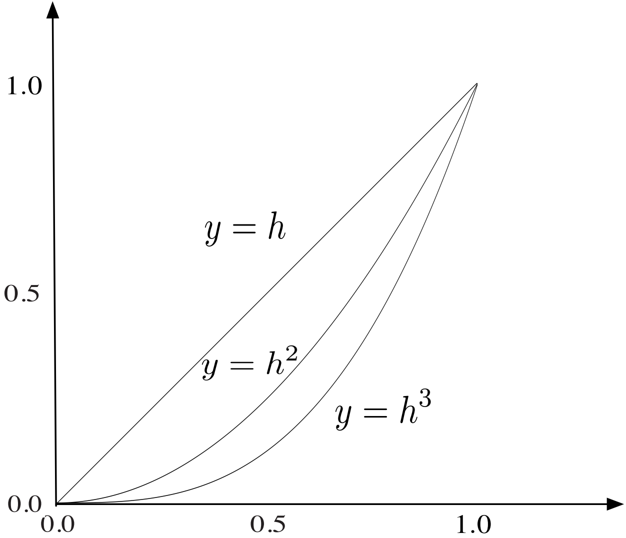
Definition

Let n be a positive integer. Given a function $f : D \to \mathbb{R}$, consider a function $g : D \times I \to \mathbb{R}$. Fix $x \in D$. We say g is an n^{th} *order accurate approximation to* f *at* x if

$$f(x) - g(x, h) = O(h^n) \text{ as } h \to 0.$$

Order of accuracy

Note that a higher n is better.



Uniformly n^{th} order accurate

Definition

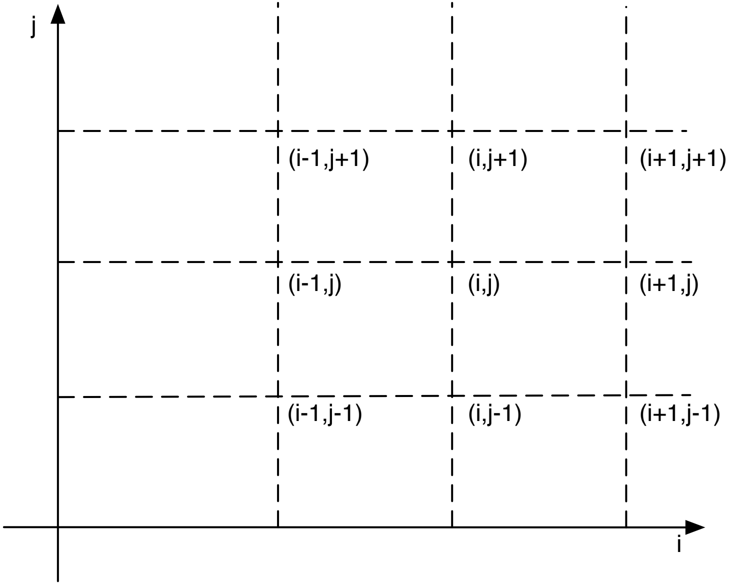
Let n be a positive integer. Given a function $f : D \to \mathbb{R}$, consider a function $g : D \times I \to \mathbb{R}$. Define $\phi : I \to \mathbb{R}$ by

$$\phi(h) = \sup_{x \in D} |f(x) - g(x, h)|.$$

We say that g is a *uniformly n^{th} order accurate approximation of f on D* if

$$\phi(h) = O(h^n) \text{ as } h \to 0.$$

Grid approximations



Grid approximations

Definition

Let n be a positive integer, $D \subseteq \mathbb{R}$, and f a function from $D \to \mathbb{R}$. Let $I = (0, a)$, where a is a positive real number and suppose $\Delta \colon I \to \wp(D)$. Let $S = \bigcup_{h \in I} (\Delta(h) \times \{h\}) \subseteq D \times I$. Suppose $g \colon S \to \mathbb{R}$. Define $\phi \colon I \to \mathbb{R}$ by

$$\phi(h) = \sup_{x \in \Delta(h)} |f(x) - g(x, h)|.$$

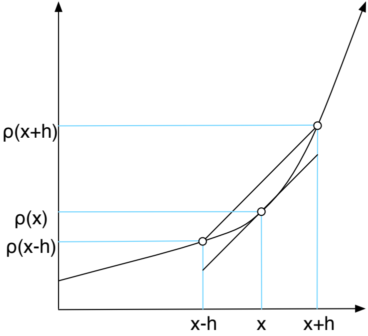
We say g is a Δ-*uniformly* n^{th} *order accurate approximation of* f if

$$\phi(h) = O(h^n) \text{ as } h \to 0.$$

Example: derivative using central difference

Approximate a derivative by taking the slope through neighboring points.

$$\rho'(x) \approx \frac{\rho(x+h) - \rho(x-h)}{2h}$$



Definition	Example
D	$\mathbb{R}$
I	$(0, a)$
$f(x)$	$\rho'(x)$
$g(x, h)$	$\frac{\rho(x+h)-\rho(x-h)}{2h}$
$\Delta(h)$	$\{ih \mid i \in \mathbb{Z}\}$
S	$\bigcup_{h\in I} (\Delta(h) \times \{h\})$
ϕ	$\sup_{x\in\Delta(h)} \left\lvert \rho'(x) - \frac{\rho(x+h) - \rho(x-h)}{2h} \right\rvert$

Code

In this example, $\Delta(h) = \{ih | i \in \mathbb{Z}, 0 \leq i < n\}$

```
void differentiate(double h, int n, double[] y, double[] result) {
   int i;
   for(i = 1; i < n-1; i++) {
      result[i] = (y[i+1]-y[i-1])/(2*h);
   }
   result[0] = (y[1]-y[0])/h;
   result[m-1] = (y[n-1]-y[n-2])/h;
}
```

We want to show this is a Δ-uniformly 2^{nd} order accurate approximation of ρ'.

Current approaches

- Prove manually
  - Prove bounds on truncation error in the numerical method
  - Limitations
    - ⋆ Manual proof could have an error
    - ⋆ Program might not match the proved method
  - Assume correct translation to code
- Do convergence studies
  - Run for various values of h and x
  - Limitations
    - ⋆ Looking at a finite set of values for h does not prove anything about the limit
    - ⋆ Might not be valid for all x in the input space

How the manual proof works

Show that

$$\frac{\rho(x+h)-\rho(x-h)}{2h} - \rho'(x) = O(h^2).$$

How the manual proof works

Show that

$$\frac{\rho(x+h)-\rho(x-h)}{2h}-\rho'(x)=O(h^2).$$

Use Taylor's theorem with Lagrangian remainders:

$$\rho(x+h)=\rho(x)+\rho'(x)h+\frac{1}{2}\rho''(x)h^2+\frac{1}{6}\rho'''(\xi_1)h^3$$
$$\rho(x-h)=\rho(x)-\rho'(x)h+\frac{1}{2}\rho''(x)h^2-\frac{1}{6}\rho'''(\xi_2)h^3.$$

How the manual proof works

Suppose $\forall x. |\rho'''(x)| \leq M$. Then

$$\begin{aligned}\left|\frac{\rho(x+h)-\rho(x-h)}{2h} - \rho'(x)\right| &= \frac{1}{12}\left|\rho'''(\xi_1) + \rho'''(\xi_2)\right| h^2 \\ &\leq \frac{1}{6} M h^2.\end{aligned}$$

Therefore

$$\frac{\rho(x+h)-\rho(x-h)}{2h} - \rho'(x) = O(h^2).$$

Automatic verification of order of accuracy

Using symbolic execution and theorem proving techniques, it is possible to provide automatic formal verification of the accuracy of a numerical program.

Automatic verification of order of accuracy

Using symbolic execution and theorem proving techniques, it is possible to provide automatic formal verification of the accuracy of a numerical program.

- Develop a tool
  - Extend TASS, a powerful symbolic execution tool
  - Operate on the semantics of real numbers
- Prove relation between *code* and function
  - Bound the truncation error
  - Check for bugs

Automatic verification of order of accuracy

Using symbolic execution and theorem proving techniques, it is possible to provide automatic formal verification of the accuracy of a numerical program.

- Develop a tool
  - Extend TASS, a powerful symbolic execution tool
  - Operate on the semantics of real numbers
- Prove relation between *code* and function
  - Bound the truncation error
  - Check for bugs
- Automatic (almost)
  - Let the computer do similar work to manual proof
  - Need annotations

Annotations

- Abstract functions

```
#pragma TASS abstract continuous(3) bound(3) double rho(double x);
```

- Derivatives

```
\D[rho,{x,1}]
```

- Quantifiers

```
forall {int j} a[j] == j*j;
forall {int j | 0 <= j && j < n} a[j] == j*j;
forall {j=0..n-1} a[j] == j*j;
```

- Assumptions

```
#pragma TASS assume x==0.0;
```

- Assertions

```
#pragma TASS assert x==0.0;
```

- Big-O

```
\O(h)
```

- Uniform

```
#pragma TASS assert uniform {j=1..n-2} \
      result[j]-\D[rho,{x,1}](j*h) == \O(h^2);
```

Annotated code

```
void differentiate(double h, int m, double[] y, double[] result) {
#pragma TASS abstract continuous(3) bound(3) double rho(double x);
#pragma TASS assume forall {j=0..m-1} y[j]==rho(j*h);
   int i;
   for(i = 1; i < m-1; i++) {
      result[i] = (y[i+1]-y[i-1])/(2*h);
   }
   result[0] = (y[1]-y[0])/h;
   result[m-1] = (y[m-1]-y[n-2])/h;
#pragma TASS assert uniform {j=1..m-2} \
          result[j]-\D[rho,{x,1}](j*h) == \O(h^2);
}
```

Symbolic execution

```
void differentiate(double h, int m, double[] y, double[] result) {
#pragma TASS abstract continuous(3) bound(3) double rho(double x);
#pragma TASS assume forall {j=0..m-1} y[j]==rho(j*h);
   int i;
   for(i = 1; i < m-1; i++) {
      result[i] = (y[i+1]-y[i-1])/(2*h);
   }
   result[0] = (y[1]-y[0])/h;
   result[m-1] = (y[m-1]-y[m-2])/h;
#pragma TASS assert uniform {j=1..m-2} \
          result[j]-\D[rho,{x,1}](j*h) == \O(h^2);
}
```

Variable	Symbolic Value
h	X_h
m	X_m
y	$X_y\langle(0, X_y[0]), \ldots, (m-1, X_y[X_m - 1])\rangle$
result	$X_{result}\langle\rangle$

Path condition: $true$

Symbolic execution

```
void differentiate(double h, int m, double[] y, double[] result) {
#pragma TASS abstract continuous(3) bound(3) double rho(double x);
#pragma TASS assume forall {j=0..m-1} y[j]==rho(j*h);
   int i;
   for(i = 1; i < m-1; i++) {
      result[i] = (y[i+1]-y[i-1])/(2*h);
   }
   result[0] = (y[1]-y[0])/h;
   result[m-1] = (y[m-1]-y[m-2])/h;
#pragma TASS assert uniform {j=1..m-2} \
          result[j]-\D[rho,{x,1}](j*h) == \O(h^2);
}
```

Variable	Symbolic Value
h	X_h
m	3
y	$X_y\langle(0, X_y[0]), (1, X_y[1]), (2, X_y[2])\rangle$
result	$X_{result}\langle\rangle$

Path condition: $true$

Symbolic execution

```
void differentiate(double h, int m, double[] y, double[] result) {
#pragma TASS abstract continuous(3) bound(3) double rho(double x);
#pragma TASS assume forall {j=0..m-1} y[j]==rho(j*h);
   int i;
   for(i = 1; i < m-1; i++) {
      result[i] = (y[i+1]-y[i-1])/(2*h);
   }
   result[0] = (y[1]-y[0])/h;
   result[m-1] = (y[m-1]-y[m-2])/h;
#pragma TASS assert uniform {j=1..m-2} \
          result[j]-\D[rho,{x,1}](j*h) == \O(h^2);
}
```

Variable	Symbolic Value
h	X_h
m	3
y	$X_y\langle(0, X_y[0]), (1, X_y[1]), (2, X_y[2])\rangle$
result	$X_{result}\langle\rangle$

Path condition: $y[0] = \rho(0) \wedge y[1] = \rho(h) \wedge y[2] = \rho(2h)$

Symbolic execution

```
void differentiate(double h, int m, double[] y, double[] result) {
#pragma TASS abstract continuous(3) bound(3) double rho(double x);
#pragma TASS assume forall {j=0..m-1} y[j]==rho(j*h);
   int i;
   for(i = 1; i < m-1; i++) {
      result[i] = (y[i+1]-y[i-1])/(2*h);
   }
   result[0] = (y[1]-y[0])/h;
   result[m-1] = (y[m-1]-y[m-2])/h;
#pragma TASS assert uniform {j=1..m-2} \
          result[j]-\D[rho,{x,1}](j*h) == \O(h^2);
}
```

Variable	Symbolic Value
h	X_h
m	3
y	$X_y\langle(0, X_y[0]), (1, X_y[1]), (2, X_y[2])\rangle$
result	$X_{result}\langle(1, \frac{\rho(2h)-\rho(0)}{2h})\rangle$

Path condition: $y[0] = \rho(0) \wedge y[1] = \rho(h) \wedge y[2] = \rho(2h)$

Symbolic execution

```
void differentiate(double h, int m, double[] y, double[] result) {
#pragma TASS abstract continuous(3) bound(3) double rho(double x);
#pragma TASS assume forall {j=0..m-1} y[j]==rho(j*h);
   int i;
   for(i = 1; i < m-1; i++) {
      result[i] = (y[i+1]-y[i-1])/(2*h);
   }
   result[0] = (y[1]-y[0])/h;
   result[m-1] = (y[m-1]-y[m-2])/h;
#pragma TASS assert uniform {j=1..m-2} \
          result[j]-\D[rho,{x,1}](j*h) == \O(h^2);
}
```

Variable	Symbolic Value
h	X_h
m	3
y	$X_y\langle(0, X_y[0]), (1, X_y[1]), (2, X_y[2])\rangle$
result	$X_{result}\langle(0, \frac{\rho(h)-\rho(0)}{h}), (1, \frac{\rho(2h)-\rho(0)}{2h})\rangle$

Path condition: $y[0] = \rho(0) \wedge y[1] = \rho(h) \wedge y[2] = \rho(2h)$

Symbolic execution

```
void differentiate(double h, int m, double[] y, double[] result) {
#pragma TASS abstract continuous(3) bound(3) double rho(double x);
#pragma TASS assume forall {j=0..m-1} y[j]==rho(j*h);
   int i;
   for(i = 1; i < m-1; i++) {
      result[i] = (y[i+1]-y[i-1])/(2*h);
   }
   result[0] = (y[1]-y[0])/h;
   result[m-1] = (y[m-1]-y[m-2])/h;
#pragma TASS assert uniform {j=1..m-2} \
          result[j]-\D[rho,{x,1}](j*h) == \O(h^2);
}
```

Variable	Symbolic Value
h	X_h
m	3
y	$X_y\langle(0, X_y[0]), (1, X_y[1]), (2, X_y[2])\rangle$
result	$X_{result}\langle(0, \frac{\rho(h)-\rho(0)}{h}), (1, \frac{\rho(2h)-\rho(0)}{2h}), (2, \frac{\rho(2h)-\rho(h)}{h})\rangle$

Path condition: $y[0] = \rho(0) \wedge y[1] = \rho(h) \wedge y[2] = \rho(2h)$

Simplification

```
result[1]-\D[rho, {x,1}](h) == \O(h^2)
```

$$
\begin{aligned}
\frac{\rho(2h)-\rho(0)}{2h} - \rho'(h) =& \frac{\rho(h) + \rho'(h)h + \frac{1}{2}\rho''(h)h^2 + \frac{1}{6}\rho'''(\xi_1)h^3}{2h} \\
&- \frac{\rho(h) - \rho'(h)h + \frac{1}{2}\rho''(h)h^2 - \frac{1}{6}\rho'''(\xi_2)h^3}{2h} - \rho'(h) \\
=& \left(\rho'(h) + \frac{1}{12}\left(\rho'''(\xi_1) + \rho'''(\xi_2)\right) h^2\right) - \rho'(h) \\
\leq& Ch^2
\end{aligned}
$$

CVC3 Interaction

- Input

```
h, M, xi1, xi2, v : REAL;
r, rx1, rx2, rx3: (REAL) -> REAL;
y : ARRAY INT OF REAL;

ASSERT h > 0 AND M > 0;
ASSERT r(2*h) = r(h)+rx1(h)*h+(1/2)*rx2(h)*h*h+(1/6)*rx3(xi1)*h*h*h;
ASSERT r(0) = r(h)-rx1(h)*h+(1/2)*rx2(h)*h*h-(1/6)*rx3(xi2)*h*h*h;
ASSERT FORALL (x : REAL): rx3(x)<= M;
ASSERT v = (r(2*h)-r(0))-rx1(h)*2*h;

QUERY v <= (M/3)*h*h*h;
```

- Output

```
Valid.
```

Challenges

Specification

- Mathematical functions
- Derivatives
- Differentiability
- Bounded Derivatives
- Big-O notation
- Relationship to program variables
- Minimize annotation effort

Verification

- Value representation
- Taylor expansion point
- Taylor expansion degree
- Theorem proving problems