Numerical Fuzz

A type system for rounding error analysis

Ariel Kellison, Justin Hsu

CSGF Program Review, 16th July 2024

Cornell University

Rounding error analysis

Derive an a priori bound on the effects of floating-point rounding errors on an algorithm – Nick Higham

Fundamental to HPC

When can we **safely** trade **accuracy** for **performance**?

Rounding error analysis

Derive an a priori bound on the effects of floating-point rounding errors on an algorithm – Nick Higham

Challenging to do in practice

Many floating-point formats…

BF16 FP16 FP8 FP64 FP32

…many numerical libraries

few tools for automated rounding error analysis

Rounding error analysis

Derive an a priori bound on the effects of floating-point rounding errors on an algorithm – Nick Higham

Problem

Sound tools that automatically bound rounding errors give **pessimistic bounds** and **scale poorly**

Rounding error analysis

Derive an a priori bound on the effects of floating-point rounding errors on an algorithm – Nick Higham

Problem

Sound tools that automatically bound rounding errors give **pessimistic bounds** and **scale poorly**

true error $\leq$ computed error

Rounding error analysis

Derive an a priori bound on the effects of floating-point rounding errors on an algorithm – Nick Higham

Problem

Sound tools that automatically bound rounding errors give **pessimistic bounds** and **scale poorly**

true error $\leq \infty$

Rounding error analysis

Derive an a priori bound on the effects of floating-point rounding errors on an algorithm – Nick Higham

Problem

Sound tools that automatically bound rounding errors give **pessimistic bounds** and **scale poorly**

$\leq$ 100 floating-point operations

Problem

Sound tools that automatically perform rounding error analyses give **pessimistic bounds** and **scale poorly**

Example: polynomial evaluation

Given $x, a_0, \ldots, a_n \in \mathbb{R}$ **evaluate**

$$P(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

Problem

Sound tools that automatically perform rounding error analyses give **pessimistic bounds** and **scale poorly**

Example: polynomial evaluation

Horner's Scheme

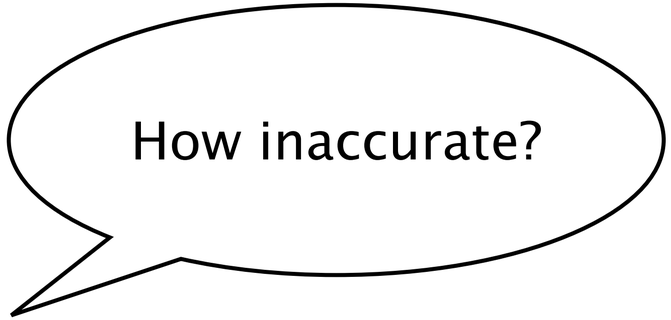
$$P(x) = a_0 + x \cdot (a_1 + x \cdot (a_2 + \ldots)\ldots)$$

Example: polynomial evaluation

v2.0.0 numpy / numpy / lib / _polynomial_impl.py

```
def polyval(p, x):
    """
    Evaluate a polynomial at specific values.

    Horner's scheme [1]_ is used to evaluate the polynomial. Even so,
    for polynomials of high degree the values may be inaccurate due to
    rounding errors. Use carefully.
```



Existing tools generate pessimistic bounds

Benchmark	Ops	Relative Error Bound*	
		Target	FPTaylor†
Horner2	4	**4.44e-16**	6.49e−11
Horner5	10	**1.11e-15**	5.00e+00

† Solovyev+, 2018

*$x, a_0, \ldots, a_n \in [0.1, 1000]$

Problem

Sound tools that automatically perform rounding error analyses give pessimistic bounds and scale poorly

Our Contribution

Numerical Fuzz: a functional programming language featuring a linear type system for rounding error analysis

Sound **Useful** **Fast**

How does it work?

It combines two key ingredients:

1. **Graded effect system**
   tracks **rounding errors**

2. **Sensitivity type system**
   tracks **function sensitivity**

How does it work?

It combines two key ingredients:

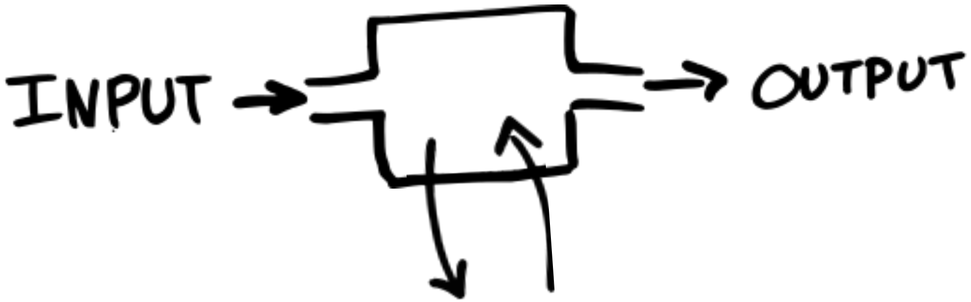
1. **Graded effect system**
   tracks **rounding errors**

2. **Sensitivity type system**
   tracks **function sensitivity**

Graded Effect System

In **functional programming languages**, functions are total

effects = what programs produce during evaluation, other than values



exceptions, errors, i/o

Our key idea: rounding error behaves like an **effect**

floating-point programs produce values and **rounding error**

Graded Effect System

Our key idea: rounding error behaves like an **effect**

$$f : \mathbf{num} \to \mathbf{num}$$

produces a value of type **num**

Graded Effect System

Our key idea: rounding error behaves like an **effect**

graded effect annotation

$$f : \mathbf{num} \to \mathbf{M}[q]\mathbf{num}$$

produces a value of type **num** and

at most q rounding error

$$q \in \mathbb{R}^{\geq 0} \cup \{\infty\}$$

How does it work?

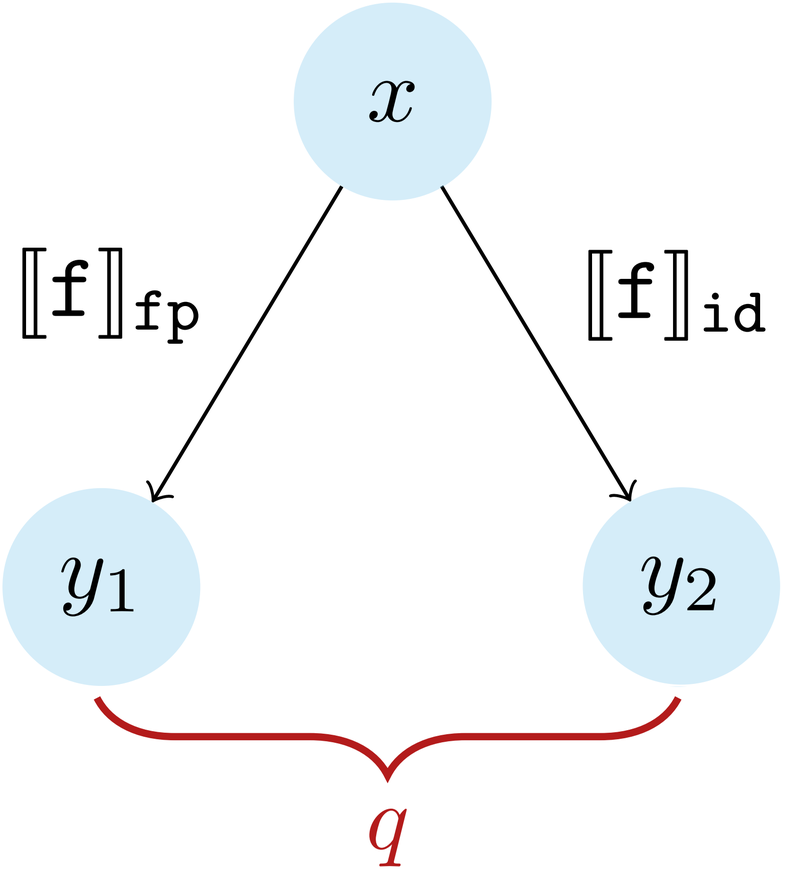
$f : \mathsf{num} \to \mathsf{M}[q]\mathsf{num}$

Ideal semantics $[\![\mathtt{f}]\!]_{\mathtt{id}}$

Infinitely precise operations

FP semantics $[\![\mathtt{f}]\!]_{\mathtt{fp}}$

Finitely precise operations

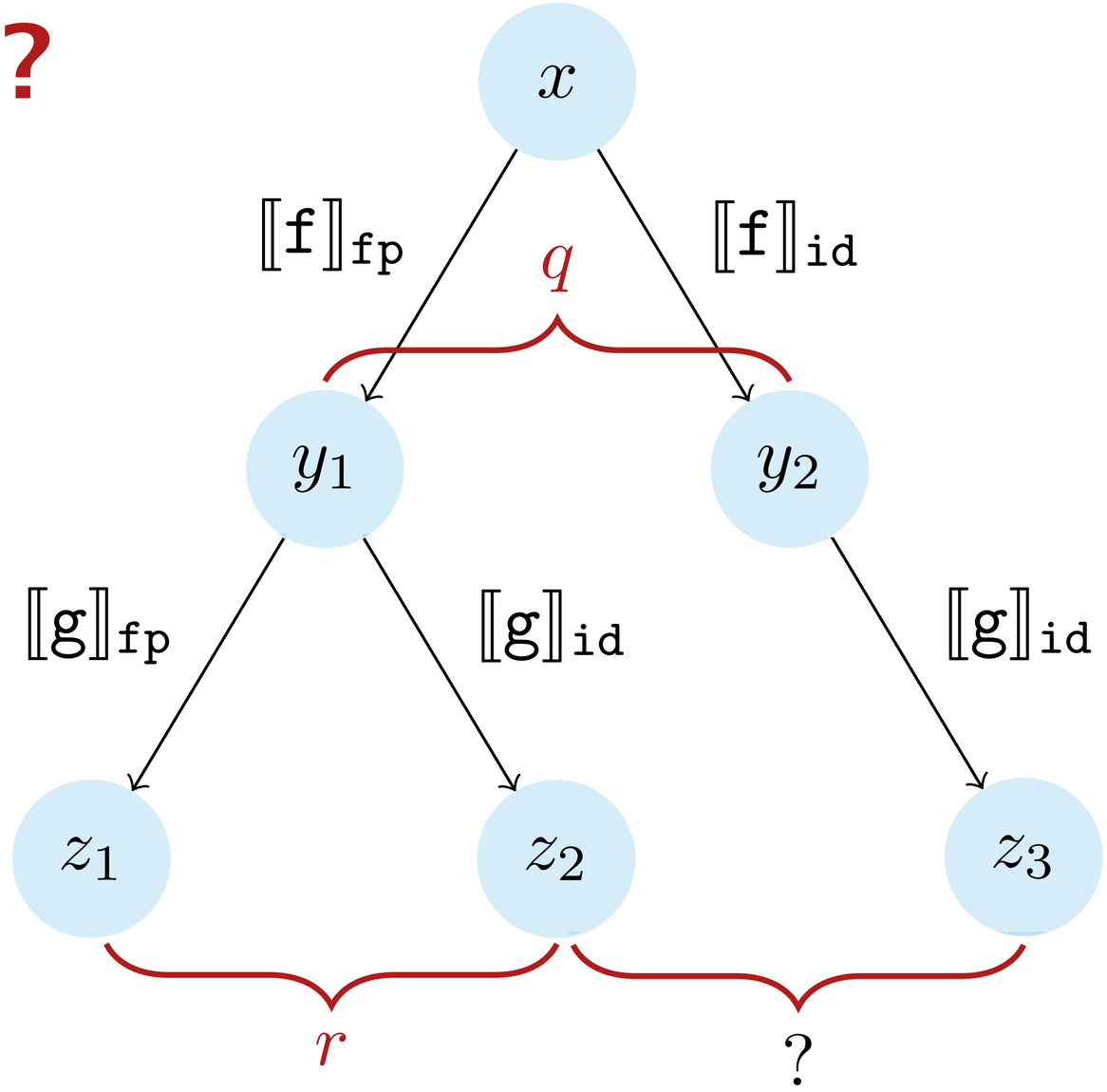


What about composition?

$f : \mathsf{num} \to \mathsf{M}[q]\mathsf{num}$

$g : \mathsf{num} \to \mathsf{M}[r]\mathsf{num}$

$(g \circ f)(x) : \mathsf{M}[?]\mathsf{num}$



How does it work?

It combines two key ingredients:

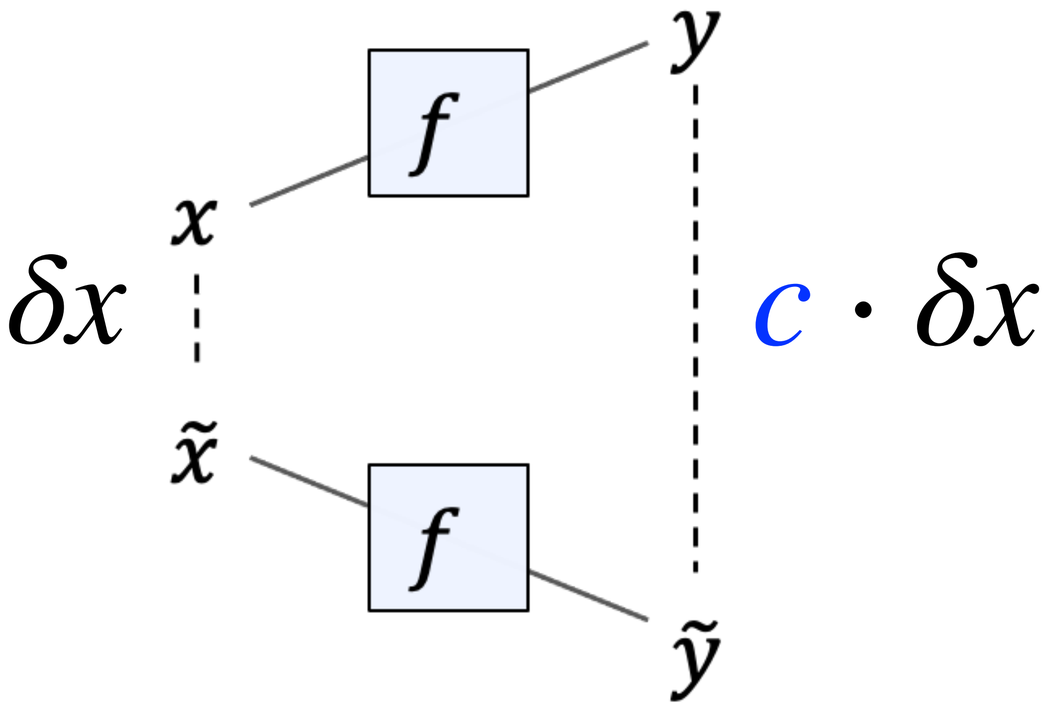
1 **Graded effect system**
tracks **rounding errors**

2 **Sensitivity type system**
tracks **function sensitivity**

Sensitivity Type Systems (Reed & Pierce, 2010)

Key idea: C-sensitivity

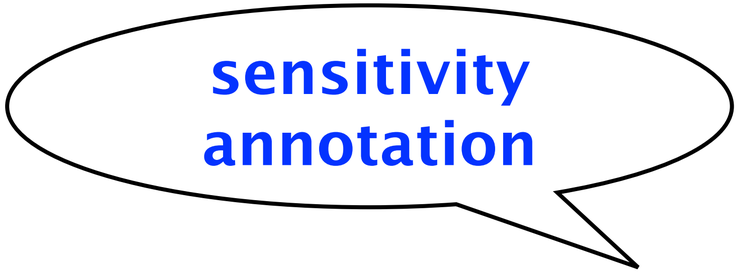
A c-sensitive function amplifies errors by at most c.



Sensitivity Type Systems (Reed & Pierce, 2010)

Key idea: C-sensitivity

A c-sensitive function amplifies errors by at most c.



$$f : ![c]\text{num} \to \text{num}$$

is c-sensitive in its argument

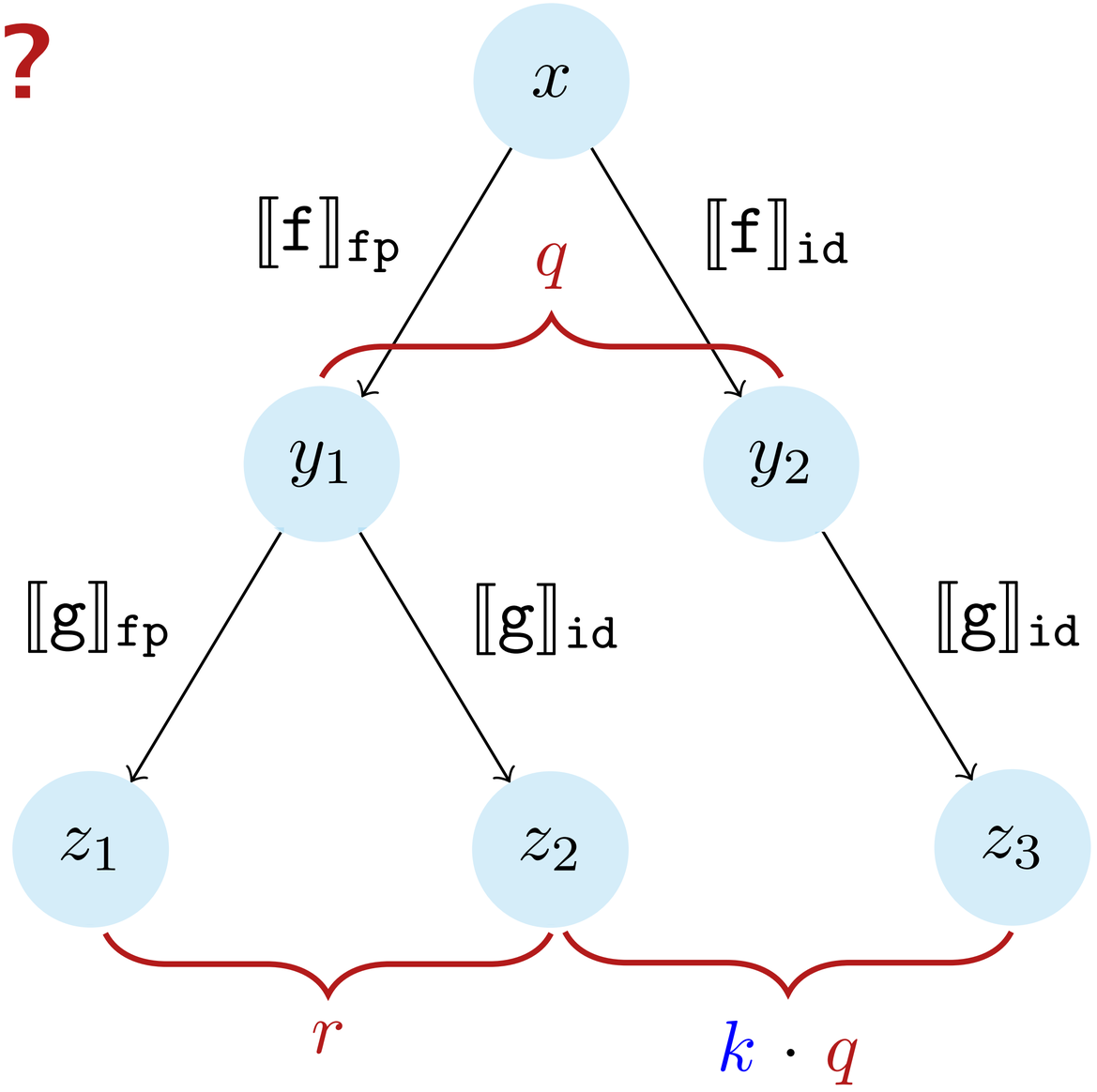
$$c \in \mathbb{R}^{\geq 0} \cup \{\infty\}$$

What about composition?

$f : \mathsf{num} \to \mathsf{M}[q]\mathsf{num}$

$g : !\![k]\mathsf{num} \to \mathsf{M}[r]\mathsf{num}$

$(g \circ f)(x) : \mathsf{M}[r + k \cdot q]\mathsf{num}$



What about composition?

$f : \text{num} \to \mathsf{M}[q]\text{num}$

$g : !\![k]\text{num} \to \mathsf{M}[r]\text{num}$

$(g \circ f)(x) : \mathsf{M}[r + k \cdot q]\text{num}$

Language guarantee

The **relative error** of a program is **bounded** by the value indicated by its **type**.

✓Sound

Our Contribution

Numerical Fuzz: a functional programming language featuring a linear type system for rounding error analysis

✓ Sound ? Useful ? Fast

Examples of Numerical Fuzz Programs

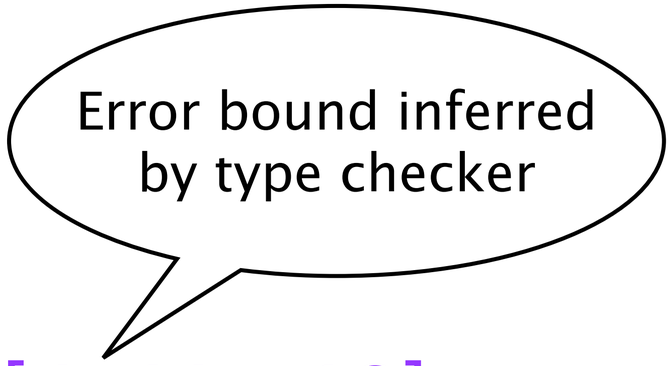
Automating analysis: Horner's scheme

Language feature: enforcing accuracy

Evaluation results

Example: Horner's scheme

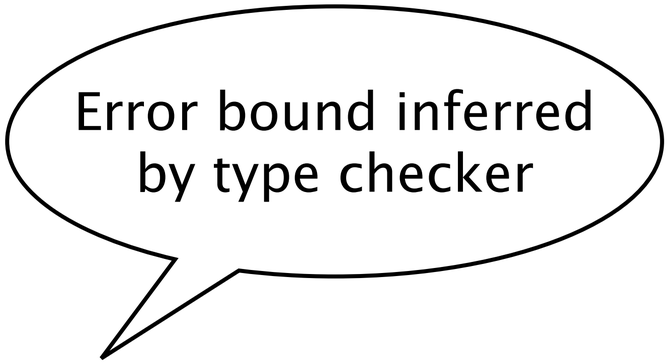
```
function Horner2_FMA (a: array3(num))(x: ![2]num)
 {
   s1 = FMA (a.2,x,a.1);
   FMA (s1,x,a.0)
 }
```




```
Horner2_FMA: array3(num) → ![2]num → M[4.44e-16]num
```

Example: Horner's scheme

```
function Horner2 (a: array3(num))(x: ![2]num)
 {
   # FMA not used in body
 }
```




```
Horner2: array3(num) → ![2]num → M[8.88e-16]num

Horner2_FMA: array3(num) → ![2]num → M[4.44e-16]num
```

Example: Enforcing Accuracy

```
function needs_accurate
 (f: array3(num) → ![2]num → M[4.44e-16]num)
 (a: array3(num)) (x: ![2]num) (……)
 { # body omitted }
```

Type checking succeeds:

```
needs_accurate(Horner2_FMA,-,-,……)
```

Example: Enforcing Accuracy

```
function needs_accurate
 (f: array3(num) → ![2]num → M[4.44e-16]num)
 (a: array3(num)) (x: ![2]num) (……)
 { # body omitted }
```

Type checking fails:

```
needs_accurate(Horner2,-,-,……)
```

```
$ Error: Cannot sat. constraint 8.88e-16 <= 4.44e-16
```

Numerical Fuzz scales to large problems

Only tool to generate relative error bounds at this scale

Benchmark	Ops	NumFuzz Bound	Theoretical Bound	Timing (s)
MatrixMultiply4	112	1.55e-15	8.88e-16	3.0e−03
SerialSum	1023	2.27e-13	2.27e-13	5.0e+00
Poly50	1325	2.94e-13	-	2.0e+00
MatrixMultiply16	7936	6.88e-15	3.55e-15	4.0e−02
MatrixMultiply64	520192	2.82e-14	1.42e-14	1.0e+01
MatrixMultiply128	4177920	5.66e-14	2.84e-14	1.1e+03

Numerical Fuzz scales to large problems

Only tool to generate relative error bounds at this scale

✓ Useful

Benchmark	Ops	NumFuzz Bound	Theoretical Bound	Timing (s)
MatrixMultiply4	112	1.55e-15	8.88e-16	3.0e−03
SerialSum	1023	2.27e-13	2.27e-13	5.0e+00
Poly50	1325	2.94e-13	-	2.0e+00
MatrixMultiply16	7936	6.88e-15	3.55e-15	4.0e−02
MatrixMultiply64	520192	2.82e-14	1.42e-14	1.0e+01
MatrixMultiply128	4177920	5.66e-14	2.84e-14	1.1e+03

Numerical Fuzz scales to large problems

Only tool to generate relative error bounds at this scale

✓ Useful ✓ Fast

Benchmark	Ops	NumFuzz Bound	Theoretical Bound	Timing (s)
MatrixMultiply4	112	1.55e-15	8.88e-16	3.0e−03
SerialSum	1023	2.27e-13	2.27e-13	5.0e+00
Poly50	1325	2.94e-13	-	2.0e+00
MatrixMultiply16	7936	6.88e-15	3.55e-15	4.0e−02
MatrixMultiply64	520192	2.82e-14	1.42e-14	1.0e+01
MatrixMultiply128	4177920	5.66e-14	2.84e-14	1.1e+03

Our Contribution

Numerical Fuzz: a functional programming language featuring a linear type system for rounding error analysis

✓ Sound ✓ Useful ✓ Fast

Key ingredients:

1. **Graded effect system**
   tracks **rounding errors**

2. **Sensitivity type system**
   tracks **function sensitivity**

More in paper

Additional benchmarks **Implementation details**

Probabilistic Rounding Error Analysis **Overflow & exceptional values**