MAX PLANCK INSTITUTE
FOR SOFTWARE SYSTEMS

ANASTASIA VOLKOVA, EVA DARULOVA

SOUND APPROXIMATION OF PROGRAMS WITH ELEMENTARY FUNCTIONS

TRADING ACCURACY FOR PERFORMANCE

Elementary functions

- sin, cos, exp, log, ...
- essential to scientific and financial computations
- may be a performance bottleneck (~75% execution time for SPICE simulator)

TRADING ACCURACY FOR PERFORMANCE

Elementary functions

- sin, cos, exp, log, ...
- essential to scientific and financial computations
- may be a performance bottleneck (~75% execution time for SPICE simulator)

Goal:

- Improve performance at cost of accuracy

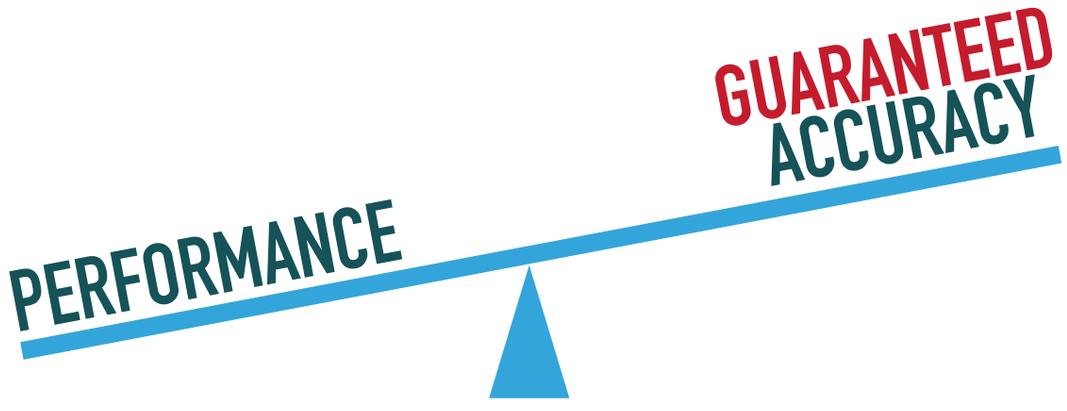
TRADING ACCURACY FOR PERFORMANCE

Elementary functions

- sin, cos, exp, log, ...
- essential to scientific and financial computations
- may be a performance bottleneck (~75% execution time for SPICE simulator)

Goal:

- Improve performance at cost of accuracy
- Give guarantees
- Automatic way, usable by non experts



OVERVIEW OF THE TOOL

Input: program over reals

```
def forwardk2jY(theta1: Real, theta2: Real): Real = {
  require(-3.14 <= theta1 && theta1 <= 3.14 && -3.14 <= theta2 && theta2 <= 3.14)
  val l1: Real = 0.5   val l2: Real = 2.5

  return l1 * sin(theta1) + l2 * sin(theta1 + theta2)
}
```

Output: C code with float64 & worst-case absolute error (3.44e-15)

OVERVIEW OF THE TOOL

Input: program over reals

```
def forwardk2jY(theta1: Real, theta2: Real): Real = {
  require(-3.14 <= theta1 && theta1 <= 3.14 && -3.14 <= theta2 && theta2 <= 3.14)
  val l1: Real = 0.5   val l2: Real = 2.5

  return l1 * sin(theta1) + l2 * sin(theta1 + theta2)
}
```

Output: C code with float64 & worst-case absolute error (3.44e-15)

Assuming libm:

- Roughly 34% of overall time for elementary functions

OVERVIEW OF THE TOOL

Input: program over reals

```
def forwardk2jY(theta1: Real, theta2: Real): Real = {
  require(-3.14 <= theta1 && theta1 <= 3.14 && -3.14 <= theta2 && theta2 <= 3.14)
  val l1: Real = 0.5   val l2: Real = 2.5

  return l1 * sin(theta1) + l2 * sin(theta1 + theta2)
} ensuring(res => res +/- 1e-13)
```

Output: C code with float64 & worst-case absolute error (3.44e-15)

Assuming libm:

- Roughly 34% of overall time for elementary functions

OVERVIEW OF THE TOOL

Input: program over reals

```
def forwardk2jY(theta1: Real, theta2: Real): Real = {
  require(-3.14 <= theta1 && theta1 <= 3.14 && -3.14 <= theta2 && theta2 <= 3.14)
  val l1: Real = 0.5   val l2: Real = 2.5

  return l1 * sin(theta1) + l2 * sin(theta1 + theta2)
} ensuring(res => res +/- 1e-13)
```

Output: C code with float64 & worst-case absolute error (3.44e-15)

Assuming libm:

- Roughly 34% of overall time for elementary functions

Our tool:

- Repartitions the error budget and uses guaranteed approximations

User requirement	Speedup
1e-13	2.9%

OVERVIEW OF THE TOOL

Input: program over reals

```
def forwardk2jY(theta1: Real, theta2: Real): Real = {
  require(-3.14 <= theta1 && theta1 <= 3.14 && -3.14 <= theta2 && theta2 <= 3.14)
  val l1: Real = 0.5   val l2: Real = 2.5

  return l1 * sin(theta1) + l2 * sin(theta1 + theta2)
} ensuring(res => res +/- 1e-13)
```

Output: C code with float64 & worst-case absolute error (3.44e-15)

Assuming libm:

- Roughly 34% of overall time for elementary functions

Our tool:

- Repartitions the error budget and uses guaranteed approximations

User requirement	Speedup
1e-13	2.9%
1e-12	13.4%

OVERVIEW OF THE TOOL

Input: program over reals

```
def forwardk2jY(theta1: Real, theta2: Real): Real = {
  require(-3.14 <= theta1 && theta1 <= 3.14 && -3.14 <= theta2 && theta2 <= 3.14)
  val l1: Real = 0.5   val l2: Real = 2.5

  return l1 * sin(theta1) + l2 * sin(theta1 + theta2)
} ensuring(res => res +/- 1e-13)
```

Output: C code with float64 & worst-case absolute error (3.44e-15)

Assuming libm:

- Roughly 34% of overall time for elementary functions

Our tool:

- Repartitions the error budget and uses guaranteed approximations

User requirement	Speedup
1e-13	2.9%
1e-12	13.4%
1e-11	17.6%

FLOATING-POINT TOOLS

- IEEE 754-2008 standard (formats, operations, exceptions,...)
- Rounding errors must be modelled, analysed and bounded:
  - $(x \text{ op } y) = (x \text{ op } y)(1+\delta), \quad |\delta| \le u, \text{ op} = +, -, \times, /$

$$\max_{x \in [a;b]} |f(x) - \tilde{f}(\tilde{x})|$$

FLOATING-POINT TOOLS

- IEEE 754-2008 standard (formats, operations, exceptions,…)
- Rounding errors must be modelled, analysed and bounded:
  - $\circ(x \text{ op } y) = (x \text{ op } y)(1+\delta), \quad |\delta| \le u, \text{op} = +, -, \times, /$
  - $\max_{x \in [a;b]} |f(x) - \tilde{f}(\tilde{x})|$
- Automated tool support
  - Certified error bounds (Gappa, Fluctuat, FPTaylor, PRECiSA, Real2Float,…)
  - Rewriting (SALSA) and mixed-precision tuning (Herbie)
  - Approximate computing (STOKE)
  - Code generators for small numerical kernels (SPIRAL, Metalibm)

DAISY

- Based on static analysis (dynamic analysis available too)
- Rewriting techniques
- Mixed-precision tuning
- Code generation in floating- and fixed-point by ensuring user-given error

Two-step data flow static analysis:

RANGE ANALYSIS	ROUNDOFF ERROR ANALYSIS
Interval and Affine Arithmetic	Affine Arithmetic

Arithmetic operations and common elementary functions (sin, cos, exp,..) assuming libm

https://github.com/malyzajko/daisy

LIMITATIONS OF STANDARD LIBMS

- Implementations depend on OS, programming language, hardware
- Reference implementation for float32 and float64
  - Arguments on the whole range of representable numbers
  - Result as accurate as target format allows

LIMITATIONS OF STANDARD LIBMS

- Implementations depend on OS, programming language, hardware
- Reference implementation for float32 and float64
  - Arguments on the whole range of representable numbers
  - Result as accurate as target format allows

OVERKILL

LIMITATIONS OF STANDARD LIBMS

- Implementations depend on OS, programming language, hardware
- Reference implementation for float32 and float64
  - Arguments on the whole range of representable numbers
  - Result as accurate as target format allows

OVERKILL

- Common case:
  - Bounded input ranges
  - Imprecise input data => no need for high accuracy

LIMITATIONS OF STANDARD LIBMS

- Implementations depend on OS, programming language, hardware
- Reference implementation for float32 and float64
  - Arguments on the whole range of representable numbers
  - Result as accurate as target format allows

OVERKILL

- Common case:
  - Bounded input ranges
  - Imprecise input data => no need for high accuracy
- Need automation:
  - Huge design space (approximation type, degree, ...)
  - Error-analysis non-trivial

LIMITATIONS OF STANDARD LIBMS

- Implementations depend on OS, programming language, hardware
- Reference implementation for float32 and float64
  - Arguments on the whole range of representable numbers
  - Result as accurate as target format allows

OVERKILL

- Common case:
  - Bounded input ranges
  - Imprecise input data => no need for high accuracy
- Need automation:
  - Huge design space (approximation type, degree, ...)
  - Error-analysis non-trivial

METALIBM

METALIBM – CODE GENERATOR FOR MATH FUNCTIONS

μετά-libm

INPUT

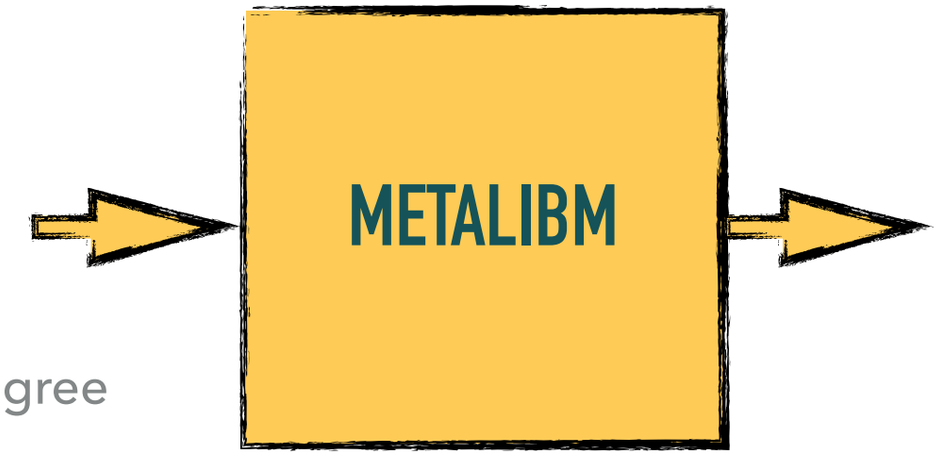
Function

Domain

Target error

Max approx degree

...



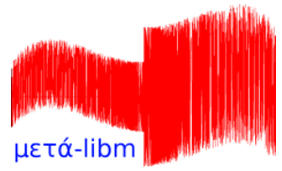
OUTPUT

C code

Gappa certificate

http://www.metalibm.org/lutetia.html

METALIBM – CODE GENERATOR FOR MATH FUNCTIONS



INPUT

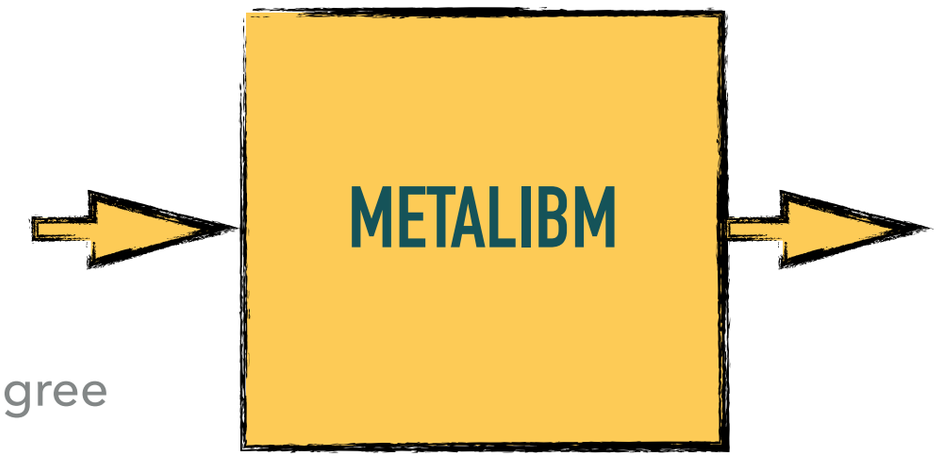
Function

Domain

Target error

Max approx degree

...



OUTPUT

C code

Gappa certificate

Three-stage process:

PROPERTIES DETECTION

Symmetry, period, ...

ARG REDUCTION / DOMAIN SPLITTING

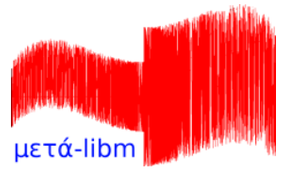
uniform/arbitrary splitting

(PIECE-WISE) POLYNOMIAL APPROXIMATION

fpminimax

http://www.metalibm.org/lutetia.html

METALIBM – CODE GENERATOR FOR MATH FUNCTIONS



INPUT

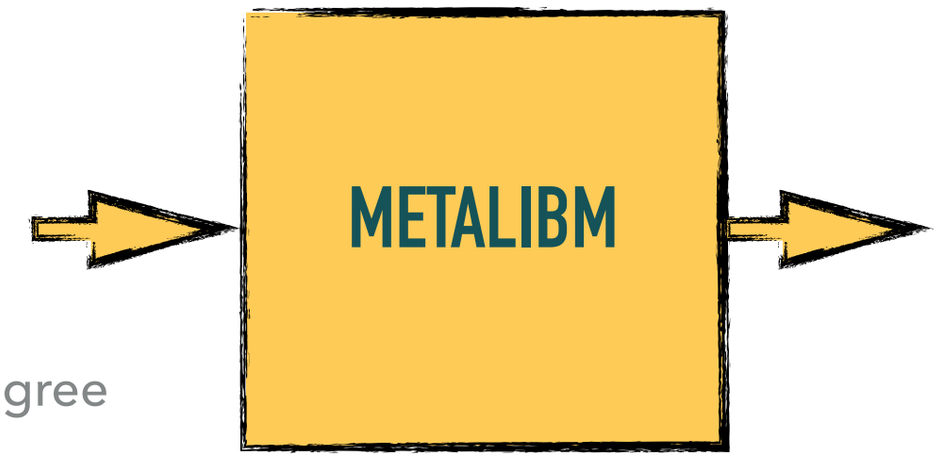
Function

Domain

Target error

Max approx degree

...



OUTPUT

C code

Gappa certificate

Three-stage process:

PROPERTIES DETECTION

Symmetry, period, ...

ARG REDUCTION / DOMAIN SPLITTING

uniform/arbitrary splitting

(PIECE-WISE) POLYNOMIAL APPROXIMATION

fpminimax

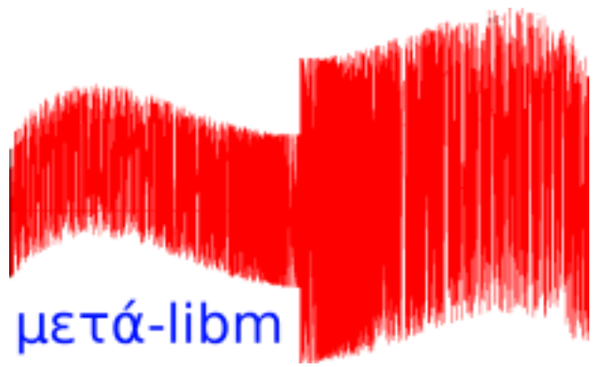
Limitations:

- Dependency performance/parameters is non-trivial and non-linear
- No help in navigating design space => usable mostly by experts

http://www.metalibm.org/lutetia.html

EMPOWERING DAISY BY USING METALIBM

Analyses errors and, given error budget, determines the room for improvement



Provides guaranteed implementations of elementary functions

OVERALL STRUCTURE

OVERALL STRUCTURE

- AST decomposition: each elementary function call is a local variable

OVERALL STRUCTURE

- AST decomposition: each elementary function call is a local variable
- Roundoff error analysis assuming libm

OVERALL STRUCTURE

- AST decomposition: each elementary function call is a local variable
- Roundoff error analysis assuming libm
- Approximation
  - Error budget repartition
  - Call Metalibm with appropriate parameters

OVERALL STRUCTURE

- AST decomposition: each elementary function call is a local variable
- Roundoff error analysis assuming libm
- Approximation
  - Error budget repartition
  - Call Metalibm with appropriate parameters
- Run roundoff analysis again (with error bounds reported by Metalibm)

OVERALL STRUCTURE

- AST decomposition: each elementary function call is a local variable
- Roundoff error analysis assuming libm
- Approximation
  - Error budget repartition
  - Call Metalibm with appropriate parameters
- Run roundoff analysis again (with error bounds reported by Metalibm)
- Code generation

ERROR DISTRIBUTION

Total error budget decomposition:

$$|f(x) - \tilde{f}(\tilde{x})| \leq \underbrace{|f(x) - \hat{f}(x)|}_{\tau_{approx}} + \underbrace{|\hat{f}(x) - \tilde{f}(\tilde{x})|}_{\tau_{fl}}$$

Equal error distribution: $\tau_i = \tau_{approx}/n$ for n calls

Derivative-based distribution: $\tau_i = \tau_{approx} w_i$, where w_i is a weight for each call

ERROR DISTRIBUTION

Total error budget decomposition:

$$|f(x) - \tilde{f}(\tilde{x})| \leq \underbrace{|f(x) - \hat{f}(x)|}_{\tau_{approx}} + \underbrace{|\hat{f}(x) - \tilde{f}(\tilde{x})|}_{\tau_{fl}}$$

Equal error distribution: $\tau_i = \tau_{approx}/n$ for n calls

Derivative-based distribution: $\tau_i = \tau_{approx} w_i$, where w_i is a weight for each call

Total error vs. local error:

$$|f(x) - \tilde{f}(x)| = |f(x) - \hat{f}_1(x)| + |\hat{f}_1(x) - \hat{f}_2(x)| + |\hat{f}_2(x) - \tilde{f}(\tilde{x})|$$

1 call replaced — both calls replaced

Total error budgets τ_1 and τ_2 at the *end of the program*. Metalibm needs *local* errors ε_i:

$$\varepsilon_i = \tau_i / m_i$$

where $m_i = \max_{[a,b]} \frac{\partial f}{\partial x_i}$ and x_i is the function call

DEGREE SELECTION

Problem:

- *“best” degree heavily depends on the target error, the domain, …*

Naive approach:

- *linear search and performance estimation via Metalibm*

Our heuristic:

- Smaller degrees for monotone functions (degrees: 4, 8, 12, 16)
- Larger degrees for non-monotone functions (degrees: 12, 16, 20, 24)

Implementation: default heuristics or manual exploration by the user

COMPOUND FUNCTIONS

Example:

$$\sin(\cos(x) - 1)$$

Idea: avoid subsequent calls and obtain 1 approximation/univariate function

Benefits:

- Need only one polynomial evaluation — more efficiency
- Metalibm overcomes undesired effects — more accuracy to trade

Implementation: call depth is left out as a user parameter

EXPERIMENTAL SETUP & DESIGN SPACE EXPLORATION

Benchmarks

- Functions from Axbench suite
- Rotation and pendulum functions
- Functions from COPRIN project
- Graduate analysis textbook

Error requirements

- 2, 3, 4 order of magnitudes larger than a libm-based implementation

Performance benchmarking

- Automatic test case generation
- 100k random inputs in given domains
- 10% of slowest runs discarded as outliers
- 5 benchmarking runs per test case

EXPERIMENTAL SETUP & DESIGN SPACE EXPLORATION

Benchmarks

- Functions from Axbench suite
- Rotation and pendulum functions
- Functions from COPRIN project
- Graduate analysis textbook

Error requirements

- 2, 3, 4 order of magnitudes larger than a libm-based implementation

Performance benchmarking

- Automatic test case generation
- 100k random inputs in given domains
- 10% of slowest runs discarded as outliers
- 5 benchmarking runs per test case

Error distribution:

- Equal vs derivative-based

Table-based approximation

- 256 vs 0 elements

Compound functions

- depth = 1 vs depth=infty

Float32

- Float64 elementary function + conversion

PERFORMANCE IMPROVEMENTS

Benchmark	Small	Middle	Large					float32
	Equal	Equal	Equal	Deriv	No table	depth=1	depth=\infty	
axisRotationX	12.6	18.9	10.0	18.0	11.1	-	-	-10.1
axisRotationY	5.0	16.7	14.0	18.9	13.4	-	-	-16.0
ex2_1	9.9	12.4	13.2	13.1	11.8	12.8	11.5	7.3
ex2_10	22.2	21.7	22.2	9.7	22.2	21.7	21.9	7.2
ex2_11	-13.5	-6.3	-2.5	-4.9	-0.9	-4.4	-6.1	18.6
ex2_2	0.2	9.1	18.2	18.7	17.7	19.2	18.9	4.5
ex2_3	-	26.7	27.4	26.8	27.3	26.8	27.4	5.7
ex2_4	11.0	14.0	14.8	13.8	14.8	14.9	13.5	7.0
ex2_5	24.3	24.1	25.4	25.0	24.5	25.6	26.1	5.3
ex2_9	2.2	10.2	8.2	9.5	9.9	8.0	10.0	-8.7
ex3_d	4.7	9.4	20.0	20.6	-0.3	20.4	20.6	5.8
forwardk2jX	-0.6	17.1	17.6	18.4	17.7	-	-	-8.3
forwardk2jY	2.9	13.4	17.6	17.9	15.3	-	-	10.2
integrate18257	-10.6	15.6	22.3	13.4	-3.7	18.1	34.1	0.8
integStoutemyer	-11.8	3.8	6.0	1.3	-6.0	4.4	19.8	6.7
pendulum1	-6.0	-4.5	-4.5	-4.2	-7.8	-	-	-9.2
pendulum2	9.1	9.6	11.5	5.7	0.0	21.2	19.7	1.4
rodriguesRotation	9.6	16.1	14.7	13.9	14.6	10.8	11.6	-6.6
sinxx10	6.4	8.3	8.1	8.3	8.3	-15.5	1.3	6.3
xu1	-22.5	14.7	26.7	24.5	27.5	27.4	26.3	8.5
xu2	-1.5	3.5	11.7	12.3	12.1	11.3	25.2	-3.2
Average	2.7	12.1	14.4	13.4	10.9	13.9	17.6	1.6
Average success	9.2	13.9	16.2	15.2	16.2.	17.1	19.1	6.5

PERFORMANCE IMPROVEMENTS

Benchmark	Small	Middle	Large					float32
	Equal	Equal	Equal	Deriv	No table	depth=1	depth=\infty	
axisRotationX	12.6	18.9	10.0	18.0	11.1	-	-	-10.1
axisRotationY	5.0	16.7	14.0	18.9	13.4	-	-	-16.0
ex2_1	9.9	12.4	13.2	13.1	11.8	12.8	11.5	7.3
ex2_10	22.2	21.7	22.2	9.7	22.2	21.7	21.9	7.2
ex2_11	-13.5	-6.3	-2.5	-4.9	-0.9	-4.4	-6.1	18.6
ex2_2	0.2	9.1	18.2	18.7	17.7	19.2	18.9	4.5
ex2_3	-	26.7	27.4	26.8	27.3	26.8	27.4	5.7
ex2_4	11.0	14.0	14.8	13.8	14.8	14.9	13.5	7.0
ex2_5	24.3	24.1	25.4	25.0	24.5	25.6	26.1	5.3
ex2_9	2.2	10.2	8.2	9.5	9.9	8.0	10.0	-8.7
ex3_d	4.7	9.4	20.0	20.6	-0.3	20.4	20.6	5.8
forwardk2jX	-0.6	17.1	17.6	18.4	17.7	-	-	-8.3
forwardk2jY	2.9	13.4	17.6	17.9	15.3	-	-	10.2
integrate18257	-10.6	15.6	22.3	13.4	-3.7	18.1	34.1	0.8
integStoutemyer	-11.8	3.8	6.0	1.3	-6.0	4.4	19.8	6.7
pendulum1	-6.0	-4.5	-4.5	-4.2	-7.8	-	-	-9.2
pendulum2	9.1	9.6	11.5	5.7	0.0	21.2	19.7	1.4
rodriguesRotation	9.6	16.1	14.7	13.9	14.6	10.8	11.6	-6.6
sinxx10	6.4	8.3	8.1	8.3	8.3	-15.5	1.3	6.3
xu1	-22.5	14.7	26.7	24.5	27.5	27.4	26.3	8.5
xu2	-1.5	3.5	11.7	12.3	12.1	11.3	25.2	-3.2
Average	2.7	12.1	14.4	13.4	10.9	13.9	17.6	1.6
Average success	9.2	13.9	16.2	15.2	16.2.	17.1	19.1	6.5

PERFORMANCE IMPROVEMENTS

Benchmark	Small	Middle	Large					float32
	Equal	Equal	Equal	Deriv	No table	depth=1	depth=\infty	
axisRotationX	12.6	18.9	10.0	18.0	11.1	-	-	-10.1
axisRotationY	5.0	16.7	14.0	18.9	13.4	-	-	-16.0
ex2_1	9.9	12.4	13.2	13.1	11.8	12.8	11.5	7.3
ex2_10	22.2	21.7	22.2	9.7	22.2	21.7	21.9	7.2
ex2_11	-13.5	-6.3	-2.5	-4.9	-0.9	-4.4	-6.1	18.6
ex2_2	0.2	9.1	18.2	18.7	17.7	19.2	18.9	4.5
ex2_3	-	26.7	27.4	26.8	27.3	26.8	27.4	5.7
ex2_4	11.0	14.0	14.8	13.8	14.8	14.9	13.5	7.0
ex2_5	24.3	24.1	25.4	25.0	24.5	25.6	26.1	5.3
ex2_9	2.2	10.2	8.2	9.5	9.9	8.0	10.0	-8.7
ex3_d	4.7	9.4	20.0	20.6	-0.3	20.4	20.6	5.8
forwardk2jX	-0.6	17.1	17.6	18.4	17.7	-	-	-8.3
forwardk2jY	2.9	13.4	17.6	17.9	15.3	-	-	10.2
integrate18257	-10.6	15.6	22.3	13.4	-3.7	18.1	34.1	0.8
integStoutemyer	-11.8	3.8	6.0	1.3	-6.0	4.4	19.8	6.7
pendulum1	-6.0	-4.5	-4.5	-4.2	-7.8	-	-	-9.2
pendulum2	9.1	9.6	11.5	5.7	0.0	21.2	19.7	1.4
rodriguesRotation	9.6	16.1	14.7	13.9	14.6	10.8	11.6	-6.6
sinxx10	6.4	8.3	8.1	8.3	8.3	-15.5	1.3	6.3
xu1	-22.5	14.7	26.7	24.5	27.5	27.4	26.3	8.5
xu2	-1.5	3.5	11.7	12.3	12.1	11.3	25.2	-3.2
Average	2.7	12.1	14.4	13.4	10.9	13.9	17.6	1.6
Average success	9.2	13.9	16.2	15.2	16.2.	17.1	19.1	6.5

PERFORMANCE IMPROVEMENTS

Benchmark	Small	Middle	Large					float32
	Equal	Equal	Equal	Deriv	No table	depth=1	depth=\infty	
axisRotationX	12.6	18.9	10.0	18.0	11.1	-	-	-10.1
axisRotationY	5.0	16.7	14.0	18.9	13.4	-	-	-16.0
ex2_1	9.9	12.4	13.2	13.1	11.8	12.8	11.5	7.3
ex2_10	22.2	21.7	22.2	9.7	22.2	21.7	21.9	7.2
ex2_11	-13.5	-6.3	-2.5	-4.9	-0.9	-4.4	-6.1	18.6
ex2_2	0.2	9.1	18.2	18.7	17.7	19.2	18.9	4.5
ex2_3	-	26.7	27.4	26.8	27.3	26.8	27.4	5.7
ex2_4	11.0	14.0	14.8	13.8	14.8	14.9	13.5	7.0
ex2_5	24.3	24.1	25.4	25.0	24.5	25.6	26.1	5.3
ex2_9	2.2	10.2	8.2	9.5	9.9	8.0	10.0	-8.7
ex3_d	4.7	9.4	20.0	20.6	-0.3	20.4	20.6	5.8
forwardk2jX	-0.6	17.1	17.6	18.4	17.7	-	-	-8.3
forwardk2jY	2.9	13.4	17.6	17.9	15.3	-	-	10.2
integrate18257	-10.6	15.6	22.3	13.4	-3.7	18.1	34.1	0.8
integStoutemyer	-11.8	3.8	6.0	1.3	-6.0	4.4	19.8	6.7
pendulum1	-6.0	-4.5	-4.5	-4.2	-7.8	-	-	-9.2
pendulum2	9.1	9.6	11.5	5.7	0.0	21.2	19.7	1.4
rodriguesRotation	9.6	16.1	14.7	13.9	14.6	10.8	11.6	-6.6
sinxx10	6.4	8.3	8.1	8.3	8.3	-15.5	1.3	6.3
xu1	-22.5	14.7	26.7	24.5	27.5	27.4	26.3	8.5
xu2	-1.5	3.5	11.7	12.3	12.1	11.3	25.2	-3.2
Average	2.7	12.1	14.4	13.4	10.9	13.9	17.6	1.6
Average success	9.2	13.9	16.2	15.2	16.2.	17.1	19.1	6.5

PERFORMANCE IMPROVEMENTS

Benchmark	Small	Middle	Large					float32
	Equal	Equal	Equal	Deriv	No table	depth=1	depth=\infty	
axisRotationX	12.6	18.9	10.0	18.0	11.1	-	-	-10.1
axisRotationY	5.0	16.7	14.0	18.9	13.4	-	-	-16.0
ex2_1	9.9	12.4	13.2	13.1	11.8	12.8	11.5	7.3
ex2_10	22.2	21.7	22.2	9.7	22.2	21.7	21.9	7.2
ex2_11	-13.5	-6.3	-2.5	-4.9	-0.9	-4.4	-6.1	18.6
ex2_2	0.2	9.1	18.2	18.7	17.7	19.2	18.9	4.5
ex2_3	-	26.7	27.4	26.8	27.3	26.8	27.4	5.7
ex2_4	11.0	14.0	14.8	13.8	14.8	14.9	13.5	7.0
ex2_5	24.3	24.1	25.4	25.0	24.5	25.6	26.1	5.3
ex2_9	2.2	10.2	8.2	9.5	9.9	8.0	10.0	-8.7
ex3_d	4.7	9.4	20.0	20.6	-0.3	20.4	20.6	5.8
forwardk2jX	-0.6	17.1	17.6	18.4	17.7	-	-	-8.3
forwardk2jY	2.9	13.4	17.6	17.9	15.3	-	-	10.2
integrate18257	-10.6	15.6	22.3	13.4	-3.7	18.1	34.1	0.8
integStoutemyer	-11.8	3.8	6.0	1.3	-6.0	4.4	19.8	6.7
pendulum1	-6.0	-4.5	-4.5	-4.2	-7.8	-	-	-9.2
pendulum2	9.1	9.6	11.5	5.7	0.0	21.2	19.7	1.4
rodriguesRotation	9.6	16.1	14.7	13.9	14.6	10.8	11.6	-6.6
sinxx10	6.4	8.3	8.1	8.3	8.3	-15.5	1.3	6.3
xu1	-22.5	14.7	26.7	24.5	27.5	27.4	26.3	8.5
xu2	-1.5	3.5	11.7	12.3	12.1	11.3	25.2	-3.2
Average	2.7	12.1	14.4	13.4	10.9	13.9	17.6	1.6
Average success	9.2	13.9	16.2	15.2	16.2.	17.1	19.1	6.5

PERFORMANCE IMPROVEMENTS

Benchmark	Small	Middle	Large					float32
	Equal	Equal	Equal	Deriv	No table	depth=1	depth=\infty	
axisRotationX	12.6	18.9	10.0	18.0	11.1	-	-	-10.1
axisRotationY	5.0	16.7	14.0	18.9	13.4	-	-	-16.0
ex2_1	9.9	12.4	13.2	13.1	11.8	12.8	11.5	7.3
ex2_10	22.2	21.7	22.2	9.7	22.2	21.7	21.9	7.2
ex2_11	-13.5	-6.3	-2.5	-4.9	-0.9	-4.4	-6.1	18.6
ex2_2	0.2	9.1	18.2	18.7	17.7	19.2	18.9	4.5
ex2_3	-	26.7	27.4	26.8	27.3	26.8	27.4	5.7
ex2_4	11.0	14.0	14.8	13.8	14.8	14.9	13.5	7.0
ex2_5	24.3	24.1	25.4	25.0	24.5	25.6	26.1	5.3
ex2_9	2.2	10.2	8.2	9.5	9.9	8.0	10.0	-8.7
ex3_d	4.7	9.4	20.0	20.6	-0.3	20.4	20.6	5.8
forwardk2jX	-0.6	17.1	17.6	18.4	17.7	-	-	-8.3
forwardk2jY	2.9	13.4	17.6	17.9	15.3	-	-	10.2
integrate18257	-10.6	15.6	22.3	13.4	-3.7	18.1	34.1	0.8
integStoutemyer	-11.8	3.8	6.0	1.3	-6.0	4.4	19.8	6.7
pendulum1	-6.0	-4.5	-4.5	-4.2	-7.8	-	-	-9.2
pendulum2	9.1	9.6	11.5	5.7	0.0	21.2	19.7	1.4
rodriguesRotation	9.6	16.1	14.7	13.9	14.6	10.8	11.6	-6.6
sinxx10	6.4	8.3	8.1	8.3	8.3	-15.5	1.3	6.3
xu1	-22.5	14.7	26.7	24.5	27.5	27.4	26.3	8.5
xu2	-1.5	3.5	11.7	12.3	12.1	11.3	25.2	-3.2
Average	2.7	12.1	14.4	13.4	10.9	13.9	17.6	1.6
Average success	9.2	13.9	16.2	15.2	16.2.	17.1	19.1	6.5

PERFORMANCE IMPROVEMENTS

Benchmark	Small	Middle	Large					float32
	Equal	Equal	Equal	Deriv	No table	depth=1	depth=\infty	
axisRotationX	12.6	18.9	10.0	18.0	11.1	-	-	-10.1
axisRotationY	5.0	16.7	14.0	18.9	13.4	-	-	-16.0
ex2_1	9.9	12.4	13.2	13.1	11.8	12.8	11.5	7.3
ex2_10	22.2	21.7	22.2	9.7	22.2	21.7	21.9	7.2
ex2_11	-13.5	-6.3	-2.5	-4.9	-0.9	-4.4	-6.1	18.6
ex2_2	0.2	9.1	18.2	18.7	17.7	19.2	18.9	4.5
ex2_3	-	26.7	27.4	26.8	27.3	26.8	27.4	5.7
ex2_4	11.0	14.0	14.8	13.8	14.8	14.9	13.5	7.0
ex2_5	24.3	24.1	25.4	25.0	24.5	25.6	26.1	5.3
ex2_9	2.2	10.2	8.2	9.5	9.9	8.0	10.0	-8.7
ex3_d	4.7	9.4	20.0	20.6	-0.3	20.4	20.6	5.8
forwardk2jX	-0.6	17.1	17.6	18.4	17.7	-	-	-8.3
forwardk2jY	2.9	13.4	17.6	17.9	15.3	-	-	10.2
integrate18257	-10.6	15.6	22.3	13.4	-3.7	18.1	34.1	0.8
integStoutemyer	-11.8	3.8	6.0	1.3	-6.0	4.4	19.8	6.7
pendulum1	-6.0	-4.5	-4.5	-4.2	-7.8	-	-	-9.2
pendulum2	9.1	9.6	11.5	5.7	0.0	21.2	19.7	1.4
rodriguesRotation	9.6	16.1	14.7	13.9	14.6	10.8	11.6	-6.6
sinxx10	6.4	8.3	8.1	8.3	8.3	-15.5	1.3	6.3
xu1	-22.5	14.7	26.7	24.5	27.5	27.4	26.3	8.5
xu2	-1.5	3.5	11.7	12.3	12.1	11.3	25.2	-3.2
Average	2.7	12.1	14.4	13.4	10.9	13.9	17.6	1.6
Average success	9.2	13.9	16.2	15.2	16.2.	17.1	19.1	6.5

VARIANCE OF THE MIN AND MAX CYCLES

Benchmark	libm		Small		Middle		Large					
							Equal		Derivative		No table	
	min	max	min	max	min	max	min	max	min	max	min	max
axisRotationX	340	611	380	426	350	401	356	491	351	451	356	446
axisRotationY	337	541	378	480	352	473	352	419	294	509	352	428
ex2_1	295	448	302	382	282	400	282	348	282	345	282	396
ex2_10	339	510	296	338	324	354	324	339	326	448	324	342
ex2_11	318	525	304	590	296	472	296	495	301	495	296	464
ex2_2	299	490	373	415	340	368	294	363	294	348	294	368
ex2_3	316	622	302	390	278	324	276	329	276	338	275	340
ex2_4	296	493	310	332	298	349	288	356	291	371	288	345
ex2_5	336	547	374	463	311	339	302	339	302	350	302	372
ex2_9	320	561	358	382	346	390	346	445	346	401	346	402
ex3_d	324	465	429	540	334	438	284	356	282	330	283	449
forwardk2jX	337	599	388	490	359	432	360	448	357	414	360	428
forwardk2jY	335	564	528	574	354	416	354	403	356	392	354	477
integrate18257	380	716	380	521	388	526	340	456	336	587	341	638
integStoutemyer	387	447	360	404	321	469	313	457	328	454	319	658
pendulum1	314	410	472	518	356	386	352	400	352	401	352	554
pendulum2	433	616	485	566	462	545	456	543	400	598	460	596
rodriguesRotation	447	697	312	346	454	510	455	512	454	586	456	521
sinxx10	285	424	515	901	306	325	305	344	304	330	305	347
xu1	359	731	343	746	394	508	302	525	300	525	298	491
xu2	368	682	360	700	350	678	343	650	346	606	349	598

NOT ONLY 17% FASTER IN AVERAGE

ACCURACY COMPARISON

Benchmarks	libm	Small	Middle	Large	
		Equal	Equal	Equal	Deriv
sinxx10	2,56E-13	2,64E-12	2,51E-11	2,5E-10	2,50E-10
xu1	6,86E-15	1,65E-14	1,62E-13	2,01E-12	2,38E-12
xu2	9,44E-15	1,5E-14	4,21E-14	3,47E-13	3,79E-13
integrate18257	3,68E-15	1,57E-14	1,65E-13	1,8E-12	1,85E-12
integStoutemyer	1,19E-15	3,12E-15	2,67E-14	2,61E-13	7,91E-14
axisRotationX	5,77E-15	2,51E-14	1,91E-13	2,19E-12	1,88E-12
axisRotationY	1,11E-14	1,94E-13	1,88E-12	2,19E-11	2,50E-11
rodriguesRotation	3,93E-13	2,16E-12	1,9E-11	2,19E-10	1,95E-10
pendulum1	4,61E-16	1,91E-14	1,88E-13	1,88E-12	1,88E-12
pendulum2	7E-15	1,9E-14	1,53E-13	1,49E-12	8,48E-14
forwardk2jX	7,22E-15	2,62E-14	2,23E-13	2,19E-12	1,93E-12
forwardk2jY	3,44E-15	2,09E-14	2,21E-13	1,56E-12	1,98E-12
ex2_1	1,28E-15	1,34E-14	2,5E-13	1,76E-12	1,76E-12
ex2_2	2,66E-15	2,28E-14	2,2E-13	2,5E-12	2,50E-12
ex2_3	1,33E-15	-	2,5E-13	2,5E-12	2,50E-12
ex2_4	2,11E-15	1,38E-14	1,14E-13	1,76E-12	2,50E-12
ex2_5	1,09E-14	8,93E-14	5,39E-13	8,06E-12	8,65E-12
ex2_9	3,37E-15	2,05E-14	1,89E-13	1,88E-12	1,88E-12
ex2_10	3E-15	1,79E-14	1,59E-13	1,56E-12	2,50E-12
ex2_11	6,08E-14	9,36E-14	6,04E-13	5,7E-12	5,12E-12
ex3_d	2,78E-15	9,78E-15	9,79E-14	1,29E-12	1,29E-12

CONCLUSION

- Automatic performance improvements even for non-experts
- Flexible tool for expert scientific computing developers
- Efficient heuristic to select suitable approximation parameters
- Extensive experimental evaluation

FUTURE WORK

- Single-precision implementations
- More control over Metalibm's heuristics
- Improve budget repartitioning

Research report available on https://avolkova.org
Code will be released as Daisy's module https://github.com/malyzajko/daisy

ANALYSIS TIME

Benchmark	Small	Middle	Large					Libm
	Equal	Equal	Equal	Deriv	No table	depth=1	depth=\infty	
sinxx10	0m 30s	0m 29s	0m 32s	0m 33s	0m 28s	7m 58s	6m 56s	6s 203ms
xu1	6m 12s	4m 0s	10m 55s	8m 7s	10m 30s	10m 5s	8m 51s	6s 623ms
xu2	15m 56s	7m 33s	3m 39s	3m 44s	3m 39s	3m 6s	4m 48s	6s 907ms
integrate18257	10m 1s	6m 49s	9m 1s	3m 24s	10m 19s	9m 1s	6m 52s	7s 142ms
integStoutemyer	8m 23s	5m 25s	5m 19s	4m 18s	4m 56s	5m 39s	1m 10s	6s 880ms
axisRotationX	8m 58s	10m 42s	9m 13s	10m 45s	9m 11s	-	-	6s 981ms
axisRotationY	10m 46s	10m 44s	8m 22s	4m 2s	7m 43s	-	-	8s 697ms
rodriguesRotation	11m 5s	10m 43s	7m 28s	6m 14s	7m 30s	6m 57s	6m 59s	8s 576ms
pendulum1	0m 30s	0m 31s	0m 28s	0m 28s	0m 26s	-	-	8s 316ms
pendulum2	1m 22s	1m 24s	1m 18s	0m 54s	1m 57s	1m 36s	1m 30s	7s 331ms
forwardk2jX	5m 44s	3m 51s	7m 58s	4m 10s	6m 51s	-	-	7s 817ms
forwardk2jY	0m 56s	3m 22s	6m 4s	6m 18s	6m 2s	-	-	7s 406ms
ex2_1	5m 25s	12m 11s	5m 35s	5m 33s	5m 36s	5m 32s	5m 32s	7s 362ms
ex2_2	5m 32s	5m 31s	2m 52s	2m 35s	3m 28s	2m 25s	2m 25s	7s 274ms
ex2_3	10m 7s	7m 5s	8m 31s	8m 32s	8m 34s	8m 34s	8m 31s	7s 290ms
ex2_4	5m 36s	5m 54s	5m 33s	0m 50s	5m 33s	5m 33s	5m 34s	7s 993ms
ex2_5	0m 57s	1m 13s	0m 50s	0m 50s	0m 50s	0m 50s	0m 50s	7s 313ms
ex2_9	0m 55s	0m 50s	0m 49s	0m 48s	0m 48s	0m 49s	0m 49s	7s 720ms
ex2_10	0m 53s	0m 45s	0m 44s	0m 26s	0m 46s	0m 45s	0m 45s	6s 976ms
ex2_11	4m 51s	3m 7s	2m 5s	1m 39s	2m 4s	7m 1s	7m 4s	6s 880ms
ex3_d	5m 40s	5m 38s	4m 6s	4m 7s	4m 57s	4m 2s	4m 2s	6s 869ms
total	2h 0m 19s	1h 47m 47s	1h 41m 22s	1h 18m 17s	1h 42m 8s	1h 19m 53s	1h 12m 38s	2m 34s 556ms

BENCHMARKS

```
def xu1(x1: Real, x2: Real): Real = {
  require(-3.14 <= x1 && x1 <= 3.14 && -3.14 <= x2 && x2 <= 3.14)
  2 * sin(x1) + 0.8 * cos(2 * x1) + 7 * sin(x2) - x1
} ensuring(res => res +/- 1e-13)

def xu2(x1: Real, x2: Real): Real = {
  require(-3.14 <= x1 && x1 <= 3.14 && -3.14 <= x2 && x2 <= 3.14)
  1.4 * sin(3 * x2) + 3.1 * cos(2 * x2) - x2 + 4 * sin(2 * x1)
} ensuring(res => res +/- 1e-13)

def integrate18257(x: Real): Real = {
  require(0 <= x && x <= 3.14)
  exp(cos(x)) * cos(x - sin(x))
} ensuring(res => res +/- 1e-13)

def integrateStoutemyer2007(x: Real): Real = {
  require(0.1 <= x && x <= 1)
  log((exp(x) + 2 * sqrt(x) + 1) / 2.0)
} ensuring(res => res +/- 1e-13)

def stoutemyerEq2007(x: Real): Real = {
  require(-1e-8 <= x && x <= 1e-8)
  exp(x) + atan(x)
} ensuring(res => res +/- 1e-13)

def sinxx10(x: Real): Real = {
  require(-3 <= x && x <= 3)
  (3 * x * x * x - 5 * x + 2) * sin(x) * sin(x) + (x * x * x + 5 * x) * sin(x) - 2*x*x - x - 2
} ensuring(res => res +/- 1e-11)
```

BENCHMARKS

```
def axisRotationX(x: Real, y: Real, theta: Real): Real =  {
    require(-2 <= x && x <= 2 && -4 <= y && y <= 4 && -5 <= theta && theta <= 5)
     x * cos(theta) + y * sin(theta)
  } ensuring(res => res +/- 1e-13)

  def axisRotationY(x: Real, y: Real, theta: Real): Real = {
    require(-2 <= x && x <= 2 && -10 <= y && y <= 10 && -5 <= theta && theta <= 5)
    -x * sin(theta) + y * cos(theta)
  } ensuring(res => res +/- 1e-12)

  def rodriguesRotation(v1: Real, v2: Real, v3: Real, k1: Real, k2: Real, k3: Real, theta: Real): Real = {
    require(-2 <= v1 && v1 <= 2 && -2 <= v2 && v2 <= 2 && -2 <= v3 && v3 <= 2 &&
      -5 <= k1 && k1 <= 5 && -5 <= k2 && k2 <= 5 && -5 <= k3 && k3 <= 5 && -5 <= theta && theta <= 5)
    v1 * cos(theta) + (k2 * v3 - k3 * v2) * sin(theta) + k1 * (k1 * v1 + k2 * v2 + k3 * v3) * (1 - cos(theta))
  } ensuring(res => res +/- 1e-11)

  def pendulum1(t: Real, w: Real): Real = {
    require(1 <= t && t <= 3 && -5 <= w && w <= 5)
    val h: Real = 0.01      val L: Real = 2.0
    val m: Real = 1.5       val g: Real = 9.80665
    t + h*(w + h/2*(-g/L * sin(t)))
  } ensuring(res => res +/- 1e-13)

  def pendulum2(t: Real, w: Real): Real = {
    require(-2 <= t && t <= 2 && 1 <= w && w <= 5)
    val h: Real = 0.01      val L: Real = 2.0
    val m: Real = 1.5       val g: Real = 9.80665
    w + h * exp(-g/L * sin(t + h/2*cos(w)))
  } ensuring(res => res +/- 1e-13)
```

BENCHMARKS

```
def forwardk2jY(theta1: Real, theta2: Real): Real = {
    require(-3.14 <= theta1 && theta1 <= 3.14 && -3.14 <= theta2 && theta2 <= 3.14)
    val l1: Real = 0.5     val l2: Real = 2.5
    l1 * sin(theta1) + l2 * sin(theta1 + theta2)
  } ensuring(res => res +/- 1e-13)

  def forwardk2jX(theta1: Real, theta2: Real): Real = {
    require(-3.14 <= theta1 && theta1 <= 3.14 && -3.14 <= theta2 && theta2 <= 3.14)
    val l1: Real = 0.5     val l2: Real = 5.5
   l1 * cos(theta1) + l2 * cos(theta1 + theta2)
  } ensuring(res => res +/- 1e-13)
```

```
  def ex2_1(x: Real): Real = {
    require(x > -1.57079632679 && x < 1.57079632679)
    val x1 = sin(x)
    val x2 = cos(x)
    x1 * x1 * x2 * x2
  } ensuring(res => res +/- 1e-13)

  def ex2_2(x: Real): Real = {
    require(x > -1 && x < 1)
    val x1 = sin(2 * x)
    val x2 = cos(2 * x)
    x1 * x1 * x2 * x2 * x2
  } ensuring(res => res +/- 1e-13)

def ex2_3(x: Real): Real = {
    require(x > 0 && x < 1)
    val x1 = cos(2 * x)
    val x2 = cos(3 * x)
    x1 * x2
  } ensuring(res => res +/- 1e-13)
```

```
def ex2_4(x: Real): Real = {
    require(x > -1.57079632679 && x < 1.57079632679)
    val x1 = sin(x)
    val x2 = cos(x)
    x1 * x1 * x1 * x1 * x1 * x2 * x2
  } ensuring(res => res +/- 1e-13)

  def ex2_5(x: Real): Real = {
    require(17 <= x && x <= 18)
    val x1 = sin(x)
    val x2 = cos(x)
    (x1 + 2 * x2) / (x2 + 2 * x1)
  } ensuring(res => res +/- 1e-12)

  def ex2_6(x: Real): Real = {
    require(-1 <= x && x <= 1)
    val x1 = tan(x)
    x1 * x1
  } ensuring(res => res +/- 1e-13)
```

BENCHMARKS

```
def ex2_7(x: Real): Real = {
  require(x > -1.57079632679 && x < 1.57079632679)
  val x1 = cos(x)
  1 / (3 + 5 * x1)
} ensuring(res => res +/- 1e-13)

def ex2_8(x: Real): Real = {
  require(17 <= x && x <= 21)
  val x1 = sin(x)
  1 / (25 + 24 * x1 * x1)
} ensuring(res => res +/- 1e-11)

def ex2_9(x: Real): Real = {
  require(1 <= x && x < 3.1415)
  val x1 = sin(x)
  val x2 = cos(x)
  1 / (1 - x2 + x1)
} ensuring(res => res +/- 1e-13)

def ex2_10(x: Real): Real = {
  require(-20 <= x && x < -18)
  val x1 = sin(x)
  val x2 = 1 + cos(x)
  x1 / (x2 * x2)
} ensuring(res => res +/- 1e-13)

def ex2_11(x: Real): Real = {
  require(-1.1 <= x && x < 0.9)
  val x1 = 1 / cos(x)
  val x2 = tan(x)
  (x1 * x1) / (4 + x2 * x2)
} ensuring(res => res +/- 1e-12)

def ex2_12(x: Real): Real = {
  require(-17 <= x && x < 7.3)
  val x1 = exp(x)
  x1 / (1 + x1)
} ensuring(res => res +/- 1e-7)

def ex3_c(x: Real): Real = {
  require(0 <= x && x <= 1)
  val x1 = exp(-2 * x)
  x * x * x * x1
} ensuring(res => res +/- 1e-13)

def ex3_d(x: Real): Real = {
  require(0 <= x && x <= 7)
  val x1 = exp(-2 * x)
  val x2 = sin(x)
  x1 * x2
} ensuring(res => res +/- 1e-13)
```