Testing COSY's Interval Arithmetic

Jun Yu and George Corliss
Marquette University, Milwaukee
(Martin Berz and Kyoko Makino)

Outline:
- Testing strategy
- Run the tests with the old COSY
- Show how Maple catches the containment failure
- Rerun the tests with the new COSY
- How Sun faired
- Opportunities for improvement
- COSY responses

3 (fixable) errors in COSY's interval arithmetic

We welcome suggestions for improving our tests

Funded in part from Michigan State University
Presented at Dagstuhl Seminar on Numerical Software with Result Verification, January 20, 2003

Why Test COSY?

Spring 2002, reliable computing list serve `reliable_computing@interval.louisiana.edu` had active discussion of COSY Infinity

Raised concerns about the reliability of interval and Taylor model arithmetics

COSY Infinity [1], by Berz *et al.* available from `http://cosy.pa.msu.edu` [2]

Arbitrary order beam dynamics simulation and analysis code using interval arithmetic and Taylor models for the validated solution of systems of ordinary differential equations

Berz commissioned execution-based testing of COSY interval arithmetic

What Is Testing?

"The purpose of testing is to find errors" - Myers [8].

Execution based testing cannot show the absence of errors, but can only demonstrate their presence

Kit [7], Kaner *et al.* [5], or Whittaker [9] offer best practice in industrial software quality assurance

Error if a program fails to do what it is supposed to do - Myers

Error if a program does what it should not do - Myers

Brain on. Eyes open. Test ... - Whittaker

What Is “Correct?”

Fundamental tenet: **Thou shalt not lie!**

It is an error to

- violate containment
- assert a mathematical falsehood

Two violations of containment in COSY's interval arithmetic:

1. power when the exponent is not an integer, but very close to it
2. (with warning) tan when the interval argument crosses discontinuity

What Is "Correct?"

Assertions of a mathematical falsehood for

1. asin and acos on intervals containing ± 1

Questions of
    Domains for interval operations
    Tightness
    Speed
    Ease of use
    etc.

Not errors. Opportunities for improved performance

Test Strategy: Goal

Goal is limited: Identify violations of containment or assertions of mathematical falsehood.

Developed a set of test cases consisting of

1. an interval vector $[x]$
2. an expression $f(x)$

Expected results are computed *a posteriori* in Maple

$[\widehat{f([x])}]$ is the result of challenging the COSY interval arithmetic to evaluate f on the interval $[x]$

Seek examples $x \in [x]$ for which $f(x)$ is not in $[\widehat{f([x])}]$

Do not need to know the true containment set of $f([x])$

Test Strategy: Maple as "Referee"

Use Maple as the "referee" of containment

1. Read each test case into a COSY driver
2. Construct COSY intervals for the arguments
3. Evaluate the expression using COSY interval arithmetic
4. Write binary values of the arguments and the COSY result
5. Read the binary arguments and COSY results into Maple
6. Perform many point evaluations $f(x)$ for $x \in [x]$
7. Compare Maple's $f(x)$ with COSY enclosure

```
for (i = 0; i <= 10; i++) {
  y = INF(X) + (SUP(X) - INF(X)) * i/10.0
  fx = f(y)
  ERROR if fx is outside COSY result
}
```

Test Strategy: Roundoff?

Most challenging aspect: Prevent inevitable roundoff errors from contaminating our results

Consider an example: Test COSY's sin on [0.1, 0.6]
Impossible to test that, since 0.1 and 0.6 are not exactly representable
Cannot express the question, "What is sin [0.1, 0.6]?" to COSY's sin

Roundoff errors may be introduced

1. Read test cases into COSY driver
2. Construct COSY interval
3. Extract COSY interval bounds
4. Write arguments and COSY results to a file
5. Read arguments and COSY results into Maple
6. Construct Maple variable precision representations
7. Perform Maple operations
8. Report from Maple

Test Strategy: Maple as "Referee"

COSY		Maple referee
Enter $[0.1,\ 0.6]$		
$\Downarrow$ round near		
$[$ internal IEEE 754 $]$	$\Longrightarrow$	$[\ \cdots\ ]$
$\Downarrow$ INTV round out		$\Downarrow$ multiprecision
INTV($\cdots$)		$x \in [\ \cdots\ ]$
$\Downarrow$ f round out		$\Downarrow$ f multiprecision
$\Downarrow$		$f(x)$
funct($\cdots$)	$\Longrightarrow$	enclosed in?

Test Strategy: Test Cases

If an interval arithmetic package gets individual operations and intrinsic functions right, it will get complicated expressions right, too

Tested primarily expressions composed of a single operation or intrinsic function

For elementary operations, not matter how wide the arguments, extrema are always at endpoints, except for division by zero

For elementary functions, extrema are always at endpoints, except for a modest set of exceptions (e.g., sin and cos on arguments that span π or $\pi/2$), which we can enumerate and test

Most likely to find violations of containment at endpoints of an argument intervals

Conjecture: For any rational function, if we get no violation of containment for all possible combinations of argument endpoints (and do not divide by zero), we can get no violation of containment from interior points

Probably straightforward application of the Maximum Principle from complex variable theory

Test Strategy: 2600+ Test Cases

Most came from TOMS 737 [6]. Kearfott *et al.* tested their Fortran 77 INTLIB interval arithmetic operations with a combination of specially constructed and randomly generated arguments

Also used 30 multi-operation expressions taken from tests of a validated quadrature package by Corliss and Rall [3]

To increase the coverage of our tests of binary operations, each pair of arguments was used in several combinations. For example for addition and subtraction, argument intervals $[a]$ and $[b]$ give test cases

- $[a]+[b]$, $[a]-[b]$, $[-a]+[b]$, $[-a]-[b]$
- $[-a]+[-b]$, $[-a]-[-b]$, $[a]+[-b]$, $[a]-[-b]$
- $[b]+[a]$, $[b]-[a]$, $[-b]+[a]$, $[-b]-[a]$
- $[-b]+[-a]$, $[-b]-[-a]$, $[b]+[-a]$, $[b]-[-a]$

Test Strategy: 2600+ Test Cases

For multiplication, with $0 \le [\underline{a}, \overline{a}]$ and $0 \le \left[\underline{b}, \overline{b}\right]$, we test 16 combinations:

- $[\underline{a}, \overline{a}] \times \left[\underline{b}, \overline{b}\right]$, $[-\underline{a}, \overline{a}] \times \left[\underline{b}, \overline{b}\right]$, $[-\overline{a}, -\underline{a}] \times \left[\underline{b}, \overline{b}\right]$, $[-\overline{a}, \underline{a}] \times \left[\underline{b}, \overline{b}\right]$
- $[\underline{a}, \overline{a}] \times \left[-\underline{b}, \overline{b}\right]$, $[-\underline{a}, \overline{a}] \times \left[-\underline{b}, \overline{b}\right]$, $[-\overline{a}, -\underline{a}] \times \left[-\underline{b}, \overline{b}\right]$, $[-\overline{a}, \underline{a}] \times \left[-\underline{b}, \overline{b}\right]$
- $[\underline{a}, \overline{a}] \times \left[-\overline{b}, -\underline{b}\right]$, $[-\underline{a}, \overline{a}] \times \left[-\overline{b}, -\underline{b}\right]$, $[-\overline{a}, -\underline{a}] \times \left[-\overline{b}, -\underline{b}\right]$, $[-\overline{a}, \underline{a}] \times \left[-\overline{b}, -\underline{b}\right]$
- $[\underline{a}, \overline{a}] \times \left[-\overline{b}, \underline{b}\right]$, $[-\underline{a}, \overline{a}] \times \left[-\overline{b}, \underline{b}\right]$, $[-\overline{a}, -\underline{a}] \times \left[-\overline{b}, \underline{b}\right]$, $[-\overline{a}, \underline{a}] \times \left[-\overline{b}, \underline{b}\right]$

Tests near boundaries of domains

Plus 500,000+ random tests (no errors):

```
loops for i and j
  a = RAND(); b = RAND();
  x1 := +- 0.a * 2^+-i;
  x2 := +- 0.b * 2^+-j;
  [X] := [x1, x2];
  expr(X);
```

Run Tests on June 8 COSY

POWER near an integer
- See 2arith_inpPOW.txt
- runtest POW
- See 2arith_resPOW.txt
- Maple 2arith_binPOW.txt

TAN crossing discontinuity
- See 2arith_inpTAN.txt
- runtest TAN
- See 2arith_resTAN.txt

ASIN or ACOS at ± 1
- See 2arith_inpASIN1.txt
- runtest ASIN1
- similarly for [-1, 1] and ACOS

Noteworthy: List is short and fixable

How Did Sun's F95 Compiler Do?

Same tests ported to Sun's F95 compiler

Error: tanh (negative), e.g., tanh ([-4.879, -4.267])
Fails by 1-2 UPL's

See 1sun.f95, Maple/test_fort.mws

Sun fixed within a week

Discrepancy between production and development

POWER: COSY Response

Martin Berz:

Observation: When the power operation is called with an interval I and a floating point exponent p very close to an integer value, the code executes, but gives a result different from I^p.

This is due to the fact that the COSY intrinsic operation "^" raises the incoming interval I to the power nint(2*p)/2, and thus agrees with the commonly known power operation only for full and half integer exponents.

The operation warns the user about this difference to the conventional power operation for exponents p sufficiently far from integer or half integer, but because of the possibility for numerical inaccuracies in the floating point value of p, can not do this for exponents very close to the allowed values.

The details of the definition of "^" in COSY were not provided due to a documentation error in the manual, which is automatically generated from the COSY language independent architecture code management system.

POWER removed from the list of user callable binary operations

Re-run test? No

TAN: COSY Response

Martin Berz:

Observation: When tan and related tools are called with an interval containing a pole of the function, the code diagnoses this situation properly, issues an error message, and warns that subsequent calculations will not be validated.

However, execution continues with a resulting interval boundary that is large, but not infinity. This is connected to the fact that the size of the largest representable number is machine dependent, and there is no machine independent treatment of "infinity" in COSY. While not leading to containment violation in the absence of a diagnostic, this is inelegant and possibly confusing.

Problem is remedied by terminating execution in such cases instead of continuing after the diagnostic message, and consistently disallowing intervals with infinite bounds.

Re-run test with October COSY

ASIN/ACOS: COSY Response

Martin Berz:

Observation: when acos and asin are called with intervals just barely exceeding one, the intrinsic properly diagnoses a domain violation and terminates execution.

However, the error message reads something like “error, acos does not exist for interval [1,1],” while the function acos is of course defined for [1,1]. This situation is due to a limitation of digits in the result of the PRINT command, which was assumed by us to be standardized to output all digits available. Since the routine for the FORTRAN system at hand (and possibly others) apparently does not show all digits, we have modified the output to a proper system independent interval output that is rounded out according to the number of digits actually shown.

Interval output as part of the diagnostic is now consistent
with the properly recognized reason for termination of execution

Re-run test with October COSY

Domains: Opportunity for Improvement?

When one passes arguments outside the domain, there are 3 possible forms of output to consider:

- Value returned: Over/underflow or empty (Sun's approach)
- Print a message?
- Fatal error?

COSY is not consistent. We observe

Program literals are handled differently from values read from data file

Overflow limits:

- Constructor INTV accepts reversed endpoints: INTV(2 1)
  COSY appears to interpret as [1, 2]
  Consider issuing a warning
- ADD, SUB, MUL, DIV, SQR: SUP(result) $\leq$ 1.79769E+308
  else Overflow to IEEE Infinity w/o warning

Domains: Opportunity for Improvement?

- SIN: [-999999.999999, 999999.999999]
  else "Warning in INSINF, the argument interval contains too large number to be accurate"
  Consider re-phrasing 'numbers'
  Result is still valid [-1, 1]
- COS: [-1000001.57, 999998.42]
  else "Warning in INSINF, the argument interval contains too large number to be accurate"
  Result is still valid [-1, 1]
  Warning makes more sense to the programmer than to the user
- TAN: [-999999.999999, 999999.999999]
  else at 1.0E6 "Error in INTANF, the argument interval contains too large number to be accurate"
  Fatal error with SQRT(-1)
  Consider re-phrasing 'numbers'
  else at [1.5, 1.6] "Error in INTANF, the argument interval includes pole"
  Fatal error with SQRT(-1)
- ASIN, ACOS: (-1, 1)
  else at [1, 1] "Error in INASNF, ARC SINE does not exist"
  Fatal error with SQRT(-1)

Domains: Opportunity for Improvement?

- SINH: [-710.47586007, 710.4758600739437]
  else at [710.475860073944, 710.475860073944] no message
  Fatal error with SQRT(-1)
- COSH: [-710.47586007394, 710.475860073943]
  else at [710.475860073944, 710.475860073944] no message
  Fatal error cosh OVERFLOW error
- SQRT: [1.0E-320, 1.79769E+308]
  else at [1.0E-326, 1.0E-326] no message
  Fatal error sqrt DOMAIN error
- ISRT: [1.0E-307, 1.79769E+308]
  else at [1.0E-308, 1.0E-308] INVALID message:
  "Error in INISRT, inverse of square root does not exist for the interval [0.000 0.222E-307]"
  Fatal error sqrt DOMAIN error
- EXP: [-1.79769E+308, 709.78271289338]
  else at [709.7827128934, 709.7827128934] No message
  Fatal error with exp DOMAIN error
- LOG: [1.0E-307, 1.79769E+308]
  else at [1.0E-308, 1.0E-308] error message
  Fatal error sqrt DOMAIN error

Domains: Opportunity for Improvement?

COSY considers it a fatal error to evaluate outside the domain of an expression, e.g., asin(1) or sqrt(0) (outside the domain because COSY enlarges the intervals)

Error if a program fails to do what it is supposed to do - Myers
Error if a program does what it should not do - Myers

Corliss opinion, not shared by Berz:
- asin(1) or sqrt(0) make good mathematical sense. Interval arithmetic should evaluate them
- A program should not experience "unexpected termination of execution because of a diagnostic," especially on anticipated input

Sun's F95 handled many cases COSY did not
Sun considers sqrt ([-1, 1]) to be [0, 1]

Berz opinion: If we can't evaluate sqrt (0.1 - 0.1), why bother about sqrt(0)

Tightness: Opportunity for Improvement?

Estimated ULP's:

See Maple code for detailed definition

```
rewU := COSYres[sup] - maxres;
if ((whichexpression = 1060) and (COSYres[sup] >= 1) ) then
   rewULPU := 0;  # Maple did not hit sin() = 1
elif ((whichexpression = 1070) and (COSYres[sup] >= 1) ) then
   rewULPU := 0;  # Maple did not hit cos() = 1
elif (rewU < 0) or (whichexpression >= 2000) then
   rewULPU := -10^10;   # Do not count
elif (maxres = 0) then
   rewULPU := round (rewU * 2^1022);
else
   rewULPU := round ((rewU / abs(maxres)) * 2^52);
end if;
```

Tightness: Opportunity for Improvement?

```
rewL := minres - COSYres[inf];
if ((whichexpression = 1060) and (COSYres[inf] <= -1) ) then
   rewULPL := 0;
elif ((whichexpression = 1070) and (COSYres[inf] <= -1) ) then
   rewULPL := 0;
elif (rewL < 0) or (whichexpression >= 2000) then
   rewULPL := -10^10;   # Do not count
elif (minres = 0) then
   rewULPL := round (rewL * 2^1022);
else
   rewULPL := round ((rewL / abs(minres)) * 2^52);
end if;
relexcesswid := rewULPL + rewULPU;
```

Show Maple code for relative ULP's

Tightness: Opportunity for Improvement?

	Estimated ULP's				Estimated Relative ULP's			
	COSY June	COSY Oct.	Sun F95	INTLAB ver. 4	COSY June	COSY Oct	Sun F95	INTLAB ver. 4
0	33	33	1277	1197	50	50	1304	1221
1	1	1	697	586	47	46	294	243
2	81	79	251	265	148	145	73	81
4	746	746	26	147	384	383	20	72
8	906	906	1	8	265	265	19	25
16	194	190	0	2	136	113	7	10
32	151	129	0	0	113	107	5	8
64	17	15	0	0	34	33	11	8
128	6	6	0	0	15	15	10	13
256	14	14	0	0	26	26	1	3
512	12	12	0	0	25	23	7	6
1024	100	89	4	4	462	445	172	197
Total	2261	2220	2256	2204	1705	1671	1923	1887

Strongest conclusion I feel comfortable drawing is that if anyone is concerned about tightness, they should look more carefully. Sun and INTLAB show that increased tightness is achievable.

Tightness: Opportunity for Improvement?

Why?

Example: [1, 2] + [3, 4]

COSY: [FC FF FF FF FF FF 0F 40 08,   04 00 00 00 00 00 18 40 08] (hex)
= [3.999 999 999 999 998 223 ...,   6.000 000 000 000 003 552 ...]
is 8 ULP's excess (estimated 5), 4 ULP's relative to width = 2 (estimated 2.5)
    Constructors INTV(1.0, 2.0) and INTV (3.0, 4.0) round out
    Operator ADD rounds out

See 3arith_inpADD.txt, 3arith_resADD.txt
    debug 3arith_binADD.txt ; Use DOS file name

Sun's excess width is 0 ULP's

Suggestions to improve definitions of estimated ULP's and relative ULP's are welcome

Tightness: Opportunity for Improvement?

Why?

Example: [1, 1] - [1, 1]

COSY: [-0.444 ..., +0.444] E-15
Excess is about 6? or ∞?

Speed: Opportunity for Improvement?

Standard Time Unit, or STU, based on 1000 evaluations of the Shekel 5 Function. See pp. 1-15 of Dixon, L. C. W. and Szegö, G. P., Towards Global Optimization 2 [4]

COSY: Intel Celeron @ 400 Mhz, 128 Mb RAM, Windows 98

Sun F95: Sun Enterprise 250, UltraSPARC 3, 1 CPU @ 450 Mhz, 512 Mb RAM

CPU time for 10 M evaluations of Shekel 5:

$$f(x) = -\sum_{i=1}^{m=5} \frac{1}{(x - A_i)(x - A_i)^T + c_i}$$

CPU time for 10 M evaluations of

$$f(x) = \log_{10}(\mathsf{asin}\ (\sin^2(x) + \cos^2(x) - \exp(\mathsf{atan}\ (-x^2 2))))$$

	COSY	Sun F95	COSY	Sun F95
Double precision	920 sec	25.4 sec	73 sec	28.9 sec
Interval	1570 sec	33.2 sec	254 sec	135.8 sec
Ratio:	1.7	1.3	3.5	4.7

Tightness and Speed: COSY Response

Martin Berz:

COSY is designed on the two premises of portability across platforms on the one hand, and use within the Taylor model framework on the other. The desired portability is achieved by building interval intrinsics based on F77 intrinsics, with the necessary safety factors of around 4 ulps because of the inherent precision (or rather lack thereof) of the intrinsics. The use in the TM framework entails that in practically relevant calculations, these slight overestimations usually do not matter since the TM approach is used for large domain intervals where because of dependency, conventional validated methods usually have much larger overestimations in all but the simplest cases. Furthermore, since the vast majority of effort in the Taylor model arithmetic lies in the floating point coefficient arithmetic which is highly optimized in COSY, the efficiency of the interval implementation is of secondary significance.

Isolated Suggestions

Not in testing scope, but things we noticed:

Update version number (and date)

When you open a COSY window, don't require user to enlarge it

Better exception handling than SQRT(-1)
"unexpected termination of execution because of a diagnostic"
vs. "crash"

Better warnings when using non-rigorously

Source restricted to 80 columns

More free-form source

More helpful error messages, e.g., missing data file

More helpful error messages, e.g., miss-declare a variable

More helpful error messages, e.g., missing ')' gave 100's of "COMMAND NOT FOUND"

Isolated Suggestions

If you are interested in improved tightness

First suggestion: Provide a character-based interval constructor so that one can construct intervals of width zero enclosing exactly representable values

Second suggestion: Provide an option using hardware directed rounding, if available.

If We Continue ...

Possible extensions to these tests include

- Refine excess width measurements
- Port tests to INTLAB in Matlab
- Port Gonnet's `www.inf.ethz.ch/personal/gonnet/FPAccuracy/Analysis.html`

Your suggestions?

Summary

Martin Berz:

“We are encouraged by the fact that COSY algorithms performed as designed regarding containment. Indeed, the asin and tan problems will not result in an unrecognized containment violation since they merely represent misleading diagnostics. We hope the documentation error responsible for the latter would not have caused incorrect results for users who in the absence of documentation may have expected a different behavior of the “^” operation. However, it would only lead to a problem if in all calls to the power operation, the exponents are just slightly away from integers or half integers, but not exactly integers or half integers, which one may say is unlikely; further, it would probably have led to suspicion from the user because the exponent to be passed is floating point, and hence necessarily will have a certain inaccuracy.

“All users of the COSY validated interval library are asked to download the latest version of the code, which contains remedies to the above problems as well as some other improvements as documented in the updated manual. Finally, while the tests of COSY were rather extensive and demanding, the fact that the inclusions produced by the code behaved as designed for all tests undertaken does of course not guarantee the absence of algorithmic or coding errors. Since both COSY source code as well as the set of test problems are available, we encourage all users to study the source code carefully or run appropriate variations of the test problems in the case of questions or concerns.”

*

References

[1] Martin Berz. COSY INFINITY Version 8 reference manual. Technical Report MSUCL–1088, National Superconducting Cyclotron Laboratory, Michigan State University, East Lansing, MI 48824, 1997.

[2] Martin Berz. COSY INFINITY web page, 2000.

[3] George F. Corliss. Performance of self-validating quadrature. In Pat Keast and Graeme Fairweather, editors, *Proceedings of the NATO Advanced Workshop on Numerical Integration: Recent Developments, Software, and Applications*, pages 239–259. Reidel, Boston, 1987.

[4] Laurence C. W. Dixon and G. P. Szegö. *Towards Global Optimization 2*. North-Holland, 1978.

[5] Cem Kaner, Jack Falk, and Hung Quoc Nguyen. *Testing Computer Software, Second edition*. Wiley, New York, 1999.

[6] R. B. Kearfott, M. Dawande, Du K.-S., and C.-Y. Hu. INTLIB: A portable FORTRAN 77 interval standard function library. *ACM Transactions on Mathematical Software*, 1994.

[7] Edward Kit. *Software Testing in the Real World: Improving the Process*. Addison Wesley, 1995.

[8] Glenford Myers. *The Art of Software Testing*. Wiley, New York, 1979.

[9] James A. Whittaker. *How to Break Software: A Practical Guide to Testing*. Addison Wesley, Boston, 2003.