Interval and Multiple Double Arithmetic

1. Numerical Evaluation of Expressions
   - a motivating example
   - interval arithmetic in SageMath
   - interval arithmetic with `mpmath`
   - evaluation with interval arithmetic
2. Naive Interval Arithmetic and Data Dependency
   - Newton's method with interval operations
   - avoiding re-use of computed data
3. Multiple Double Arithmetic
   - the QD library
   - error free transformations

MCS 507 Lecture 7
Mathematical, Statistical and Scientific Software
Jan Verschelde, 6 September 2023

Interval and Multiple Double Arithmetic

1. Numerical Evaluation of Expressions
   - a motivating example
   - interval arithmetic in SageMath
   - interval arithmetic with `mpmath`
   - evaluation with interval arithmetic
2. Naive Interval Arithmetic and Data Dependency
   - Newton's method with interval operations
   - avoiding re-use of computed data
3. Multiple Double Arithmetic
   - the QD library
   - error free transformations

an expression

Problem: Evaluate $f(x, y) =$

$$(333.75 - x^2)y^6 + x^2(11x^2y^2 - 121y^4 - 2) + 5.5y^8 + x/(2y)$$

at $(77617, 33096)$.

An example of Stefano Taschini: Interval Arithmetic: Python Implementation and Applications. In the Proceedings of the 7th Python in Science Conference (SciPy 2008).

Siegfried M. Rump: Verification methods: Rigorous results using floating-point arithmetic. *Acta Numerica* 19:287-449, 2010.

numerical evaluation

```
F = lambda x, y: (333.75 - x**2)*y**6 \
  + x**2*(11*x**2*y**2 - 121*y**4 - 2) \
  + 5.5*y**8 + x/(2*y)
(A, B) = (77617, 33096)
Z = F(float(A), float(B))
print('numerical value :', Z)
```

shows

```
numerical value : 1.17260394005
```

verification with SymPy

```
import sympy as sp
X, Y = sp.var('x, y')
G = (sp.Rational(33375)/100 - X**2)*Y**6 \
  + X**2*(11*X**2*Y**2 - 121*Y**4 - 2) \
  + sp.Rational(55)/10*Y**8 \
  + sp.Rational(1)*X/(2*Y)
print('evaluating', G, 'at', (A, B))
E = sp.Subs(G, (X, Y), (A, B)).doit()
E15 = E.evalf(15)
print('numerical value :', Z)
print('exact value :', E, '~', E15)
print('error :', abs(E15 - Z))
```

shows

```
numerical value : 1.17260394005
exact value : -54767/66192 ~ -0.827396059946821
error : 2.00000000000000
```

doubling the precision

```
mpmath.mp.dps = 30
MP_F = lambda x, y: (mpmath.mpf('333.75') \
     - x**2)*y**6 \
     + x**2*(11*x**2*y**2 - 121*y**4 - 2) \
     + mpmath.mpf('5.5')*y**8 + x/(2*y)
MP_A30 = mpmath.mpf(str(A))
MP_B30 = mpmath.mpf(str(B))
Z30 = MP_F(MP_A30, MP_B30)
print('using 30 digits :', Z30)
```

shows

```
using 30 digits : 1.172603940053178631858834904520
```

the right working precision

```
mpmath.mp.dps = 35
...
mpmath.mp.dps = 36
...
```

shows

```
using 35 digits : 1.17260394005317863185883490452018 37
using 36 digits : -0.827396059946821368141165095479816292
```

Problem: when is the precision insufficient?

Interval and Multiple Double Arithmetic

1. Numerical Evaluation of Expressions
   - a motivating example
   - interval arithmetic in SageMath
   - interval arithmetic with `mpmath`
   - evaluation with interval arithmetic
2. Naive Interval Arithmetic and Data Dependency
   - Newton's method with interval operations
   - avoiding re-use of computed data
3. Multiple Double Arithmetic
   - the QD library
   - error free transformations

root finding with SageMath

```
from sage.rings.polynomial.real_roots import real_roots
x = polygen(ZZ)
real_roots(x^2 - 2, retval='interval', \
  max_diameter=1/2^100)
[(-1.41421356237309504880168872420970?, 1), \
  (1.41421356237309504880168872420970?, 1)]
real_roots((x-1)*(x-2)*(x-3)*(x-4))
[((11/16, 33/32), 1), ((11/8, 33/16), 1), \
 ((11/4, 99/32), 1), ((55/16, 33/8), 1)]
real_roots((x-1)*(x-2)*(x-3)*(x-4), \
  max_diameter=1/2^10)
[((137438953465/137438953472, \
 34359738369/34359738368), 1), \
 ((137438953465/68719476736, \
 34359738369/17179869184), 1), \
 ((206158430203/68719476736, \
 103079215107/34359738368), 1), \
...
```

solving linear systems

```
sage: R = RealIntervalField(53)
sage: A = matrix(R,2,2,[[R(2,2),R(-1.1,1)],\
                        [R(-1,1),R(3.7,4)]])
sage: A^(-1)
[0.5?  0.?]
[ 0.? 0.3?]
sage: b = vector(R,[R(1,1),R(2,2)])
sage: x = A.solve_left(b)
sage: x
(1.?, 1.?)
sage: x[0].absolute_diameter()
0.809523809523810
sage: x[1].absolute_diameter()
0.479853479853480
```

Interval and Multiple Double Arithmetic

1. Numerical Evaluation of Expressions
   - a motivating example
   - interval arithmetic in SageMath
   - interval arithmetic with `mpmath`
   - evaluation with interval arithmetic
2. Naive Interval Arithmetic and Data Dependency
   - Newton's method with interval operations
   - avoiding re-use of computed data
3. Multiple Double Arithmetic
   - the QD library
   - error free transformations

mpmath

From `http://mpmath.org/`:

- **mpmath** is a free (BSD licensed) Python library for real and complex floating-point arithmetic with arbitrary precision.
- developed by Fredrik Johansson since 2007, with help from many contributors.
- **mpmath** works with both Python 2 and Python 3, with no other required dependencies.
- `https://github.com/fredrik-johansson/mpmath`

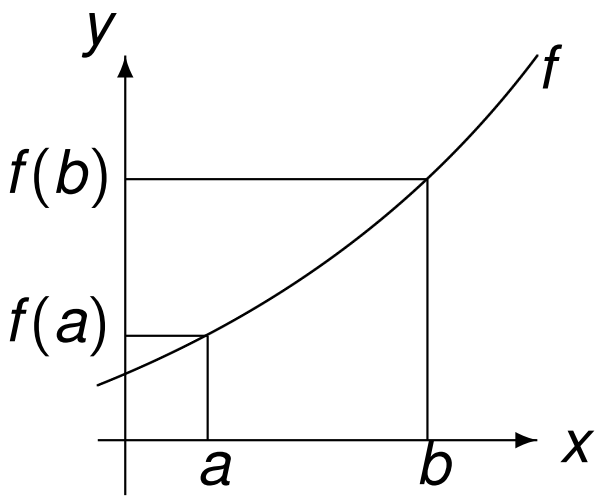
interval arithmetic

```
from mpmath import iv
iv.dps = 15
x = iv.mpf(3)
print(x, 'has type', type(x))
y = iv.mpf([3,4])
z = x/y
print(x, '/', y, '=', z)
```

shows

```
  [3.0, 3.0] has type \
<class 'mpmath.ctx_iv.ivmpf'>
  [3.0, 3.0] / [3.0, 4.0] = [0.75, 1.0]
```

functions of intervals



For continuous f over $[a, b]$:

$$f([a,b]) = \left[\min_{x \in [a,b]} f(x), \max_{x \in [a,b]} f(x) \right].$$

mathematical functions

```
from mpmath import iv
iv.dps = 15
E = iv.exp(1)
print('e :', E)
print('log(e) :', iv.log(E))
print('sin(e) :', iv.sin(E))
print('cos(e) :', iv.cos(E))
P = iv.pi
print('pi :', P)
print('sin(pi) :', iv.sin(P))
print('cos(pi) :', iv.cos(P))
```

```
$ python usempmathivfun.py
e : [2.7182818284590450908, \
2.7182818284590453349]
log(e) : [0.99999999999999988898, \
1.00000000000000222]
sin(e) : [0.41078129050290840274, \
0.41078129050290884683]
cos(e) : [-0.91173391478696530488, \
-0.91173391478696497181]
pi : [3.1415926535897932116, \
3.1415926535897935601]
sin(pi) : [-3.2162452993532732201e-16, \
1.2246467991473532072e-16]
cos(pi) : [-1.0, -0.99999999999999988898]
```

instantiating with strings

```
a = iv.mpf(0.1)
b = iv.mpf('0.1')
print('Observe strings!')
print(a)
print(b)
```

shows

```
Observe strings!
[0.10000000000000000555, 0.10000000000000000555]
[0.09999999999999991673, 0.10000000000000000555]
```

properties of intervals

```
print('some properties of', b)
print('middle :', b.mid)
print('width :', b.delta)
print('left bound :', b.a)
print('right bound :', b.b)
```

shows

```
  some properties of [0.09999999999999991673, \
0.10000000000000000555]
  middle : [0.10000000000000000555, \
0.10000000000000000555]
  width : [1.387778780781445675529e-17, \
1.387778780781445675529e-17]
  left bound : [0.09999999999999991673, \
0.09999999999999991673]
  right bound : [0.10000000000000000555, \
0.10000000000000000555]
```

the internal representation

```
print('internal representation of', b)
print(b.__dict__)
fraction = b.__dict__['_mpi_'][0][1]
exponent = b.__dict__['_mpi_'][0][2]
print('fraction =', fraction)
print('exponent =', exponent)
lb = fraction*2.0**exponent
print(lb)
```

shows

```
  internal representation of \
[0.09999999999999991673, 0.10000000000000000555]
  {'_mpi_': ((0, 7205759403792793L, -56, 53), \
(0, 3602879701896397L, -55, 52))}
  fraction = 7205759403792793
  exponent = -56
  0.1
```

Interval and Multiple Double Arithmetic

1. Numerical Evaluation of Expressions
   - a motivating example
   - interval arithmetic in SageMath
   - interval arithmetic with `mpmath`
   - evaluation with interval arithmetic
2. Naive Interval Arithmetic and Data Dependency
   - Newton's method with interval operations
   - avoiding re-use of computed data
3. Multiple Double Arithmetic
   - the QD library
   - error free transformations

multiprecision interval arithmetic

```
from mpmath import iv
print('using 35 decimal places ...')
iv.dps = 35
IV_F = lambda x, y: (iv.mpf('333.75') \
     - x**2)*y**6 \
     + x**2*(iv.mpf('11')*x**2*y**2 \
     - iv.mpf('121')*y**4 - iv.mpf('2')) \
     + iv.mpf('5.5')*y**8 + x/(iv.mpf('2')*y);
IV_A = iv.mpf(str(A))
IV_B = iv.mpf(str(B))
IV_Z = iv_f(IV_A, IV_B)
print(IV_Z)
```

shows

```
  using 35 decimal places ...
  [-6.827396059946821368141165095479816292382, \
1.172603940053178631858834904520183709123]
```

the right working precision

```
print('using 36 decimal places ...')
iv.dps = 36
...
```

shows

```
  using 36 decimal places ...
  [-0.827396059946821368141165095479816292005 49, \
   -0.827396059946821368141165095479816291817 41]
```

$$= -\left(0.827396059946821368141165095479816292^{00549}_{81741} \right)$$

width of interval = upper bound on error

Interval and Multiple Double Arithmetic

1. Numerical Evaluation of Expressions
   - a motivating example
   - interval arithmetic in SageMath
   - interval arithmetic with `mpmath`
   - evaluation with interval arithmetic
2. Naive Interval Arithmetic and Data Dependency
   - Newton's method with interval operations
   - avoiding re-use of computed data
3. Multiple Double Arithmetic
   - the QD library
   - error free transformations

Newton's method

To approximate $\sqrt{2}$ with quadratic convergence,
we apply Newton's method on $x^2 - 2 = 0$:

$$x := x - \frac{f(x)}{f'(x)} = x - \frac{x^2 - 2}{2x}$$

starting with the interval $[1.4, 1.5]$.

Naive interval arithmetic: replace each operation
in an algorithm by the corresponding interval operation.

the script

```
from mpmath import iv
iv.dps = 15
print('naive interval Newton :')
X = iv.mpf(['1.4', '1.5'])
for i in range(5):
    X = X - (X**2 - 2)/(2*X)
    print(X)
```

increasing width

```
naive interval Newton :
[1.3107142857142854986, 1.5142857142857114457]
[1.1989198131568696848, 1.6218713507201252266]
[0.93598868491625553112, 1.8564955830691347582]
[0.16323583369742133975, 2.4568902244900931997]
[-12.200194069435248423, 8.501378183717767100 8]
```

Experiments like this contribute to the dubious reputation of interval arithmetic: *the intervals only just get wider!*

Why? In division `(X**2 - 2)/(2*X)` one uses for `X` the upper bound in the numerator and the lower bound in the denominator.

But this is an inappropriate use of interval arithmetic.
Principle: avoid re-use of computed data.

Interval and Multiple Double Arithmetic

1. Numerical Evaluation of Expressions
   - a motivating example
   - interval arithmetic in SageMath
   - interval arithmetic with `mpmath`
   - evaluation with interval arithmetic
2. Naive Interval Arithmetic and Data Dependency
   - Newton's method with interval operations
   - avoiding re-use of computed data
3. Multiple Double Arithmetic
   - the QD library
   - error free transformations

inclusion/exclusion of intervals

Denote $N_f(x, [a, b]) := x - \dfrac{f(x)}{f'([a, b])}$.

Theorem:

- If $N_f(x, [a, b]) \subseteq [a, b]$,
  then $[a, b]$ contains a unique root of f.
- If $N_f(x, [a, b]) \cap [a, b] = \emptyset$, then $f(x) \neq 0$ for all $x \in [a, b]$.

Application: take midpoint of $[a, b]$ in each Newton step.

proper interval Newton

```
print('proper interval Newton :')
X = iv.mpf(['1.4', '1.5'])
for i in range(5):
    X = X.mid
    X = X - (X**2 - 2)/(2*X)
    print(X)
```

converging intervals

```
proper interval Newton :
[1.4146551724137930162, 1.4146551724137932382]
[1.4142136313013509152, 1.4142136313013513593]
[1.4142135623730964777, 1.4142135623730969218]
[1.4142135623730949234, 1.4142135623730951455]
[1.4142135623730949234, 1.4142135623730951455]
```

converged to $1.414213562373095^{51455}_{49234}$

Interval and Multiple Double Arithmetic

1. Numerical Evaluation of Expressions
   - a motivating example
   - interval arithmetic in SageMath
   - interval arithmetic with `mpmath`
   - evaluation with interval arithmetic
2. Naive Interval Arithmetic and Data Dependency
   - Newton's method with interval operations
   - avoiding re-use of computed data
3. Multiple Double Arithmetic
   - the QD library
   - error free transformations

quad double arithmetic

A quad double is an unevaluated sum of 4 doubles,
improves working precision from 2.2×10^{-16} to 2.4×10^{-63}.

Y. Hida, X.S. Li, and D.H. Bailey: **Algorithms for quad-double precision floating point arithmetic.** In *15th IEEE Symposium on Computer Arithmetic* pages 155–162. IEEE, 2001. Software at
`http://crd.lbl.gov/~dhbailey/mpdist/qd-2.3.22.tar.gz`.

A quad double builds on `double double`, some features:

- The least significant part of a `double double` can be interpreted as a compensation for the roundoff error.
- Predictable overhead: working with `double double` is of the same cost as working with complex numbers.

Interval and Multiple Double Arithmetic

1. Numerical Evaluation of Expressions
   - a motivating example
   - interval arithmetic in SageMath
   - interval arithmetic with `mpmath`
   - evaluation with interval arithmetic
2. Naive Interval Arithmetic and Data Dependency
   - Newton's method with interval operations
   - avoiding re-use of computed data
3. Multiple Double Arithmetic
   - the QD library
   - error free transformations

multiple double precision — error free transformations

Computing the 2-norm of a vector of dimension 64
of random complex numbers on the unit circle equals 8.
Observe the second double of the multiple double 2-norm.

```
double double : 8.00000000000000E+00 - 6.47112461314111E-32
  quad double : 8.00000000000000E+00 + 3.20475411419393E-65
  octo double : 8.00000000000000E+00 - 9.72609915198313E-129
```

- CAMPARY by M. Joldes, J.-M. Muller, V. Popescu, and W. Tucker.
  CAMPARY: Cuda Multiple Precision Arithmetic Library and Applications.
  In *Mathematical Software – ICMS 2016, the 5th Internatical Conference on Mathematical Software*, pages 232–240, Springer-Verlag, 2016.
- J.-M. Muller, N. Brunie, F. de Dinechin, C.-P. Jeannerod, M. Joldes, V. Lefèvre, G. Melquiond, N. Revol, and S. Torres.
  Handbook of Floating-Point Arithmetic.
  Springer-Verlag, second edition, 2018.

Summary + Exercises

Interval arithmetic enables rigorous computing.
Double double and multiple double arithmetic extend double precision.

1. Consider $p(x) = x^2 - 4x$, $q(x) = x(x-4)$ and $[a, b] = [1, 4]$. Compare the straightforward interval evaluations of $[a, b]$ in p and q with the graph of the function. Which form, p or q, yields the best result?
2. Compute $1/x$ for $x = [-1, +1]$. Compare the outcome with the graph of $1/x$. Can you improve the outcome?
3. Generate a 2-by-2 matrix A of intervals that is close to a singular matrix. Experiment with A^{-1} in SageMath.
4. Consider the application of Newton's method to approximate the $\sqrt{2}$ in quad double arithmetic.
   Use the C interface to the QD library to write a simple test.