Computer Algebra Systems: Numerics

Lecture 17

“Symbolic Computation” includes numeric as a subset

- Why do CAS not entirely replace numeric programming environments?

“Symbolic Computation” vs...

- Purely Numeric Systems prosper . Why?
  - loss in efficiency is not tolerated
  - unless something sophisticated is going on, the symbolic system adds more complexity than necessary. (learning curve)
  - CAS systems are “extra cost”
- Other reasons.
  - People are successful in the first approach they learned. They don’t change.
  - How else to explain Fortran

What is the added value for Symb.+Num?

- SENAC-like systems (Computer Algebra, front end help systems)
- Code-generation systems (GENTRAN)
- integrated visualization, interaction, plotting
- exact integer and rational arithmetic
- extra precision (seamlessly)
- interval arithmetic
  - explicit endpoints (range in Maple, Interval in MMa)
  - implicit intervals (significance arithmetic)

Numerics tend to be misunderstood

- Insufficient explanation about what is going on
- Peculiar user expectations. Is 3.000 more accurate than 3.0? Is it more precise?
- Why is sum(0.001,i=1,1000) only 0.99994?
- Mathematica default makes simple convergent processes diverge.

Square root of 9 by Newton Iteration

- s[x_]:=   x- (x^2-9)/(2*x);
- Nest[s,2,5] → (11641532182693481445313/3880510727564493815104... differs from 3 by
- 1/3880510727564493815104
- Nest[s,2.0,5]-3 = "0.0" ;start interation at 2.
- Nest[s,2.000000000000000,5]-3 = 0.x10^{-18}
- Nest[s,2.000000000000000,50]-3 = 0.x10^{-5}
- r=Nest[s,2.000000000000000,70]-3 = 0.

Nest[s,2.0000000000000000000000000000,88]-2 =0.
//umh, you mean the iteration also converges to 2??

It looks like it was getting worse, and then got better

- InputForm[r] is 0``-0.4771
- furthermore, r+1 prints as 0.

Mathematica has gotten more elaborate

- AccuracyGoal
- WorkingPrecision
- SetPrecision
  - beyond simple characterization
- Claims (v 3) to run all routines to enough accuracy to provide (conservatively) as many digits correct as requested. [subsequently retracted?]
- Decisions (e.g. $\sin(\tan(x)) < \tan(\sin(x))$ for x near zero) can be tricky. Taylor series of difference starts as $-x^7/30+ \ldots$)

Other possibilities: IEEE binary FP std

- Start with standard (IEEE float) and extend toward symbolic. IEEE 754, 854 (any radix).
- Problematical: there are symbols like +/- infinity, not-a-number, signed 0, in IEEE, which take on some of the properties of symbols. What to do? In particular….
- Is NaN a way to represent a symbol z? (a symbol is a number that is not a number?)
- Rounding modes (etc) in software are time consuming when implemented poorly.

Start with a numeric programming system

- Matlab: add a “Maple Toolbox”. Allow symbols or expressions as strings in a matrix.
- Limited integration of facilities.
- Excel: add functionalities (again, using strings) as patches to a spreadsheet program.

Explicitly add numeric libraries to CAS

- Treat (say) numeric matrices as a special case: transfer to ordinary double-precision floats to do numerics.
- Put all the work into good interfaces so that the CAS can guide the computation.
- From lisp systems, “foreign function“ calls?

Rewrite all the code in lisp

- How hard would it be to compile C or Fortran into Lisp, and then compile it from Lisp into binary code?
- A program: f2CL exists. Major efforts to pound on it have improved it (credits: Kevin Broughan, Raymond Toy, me..)
- How does this compare to FF?

Non-functional vs functional: the Fortran version

- x = x+1 in Fortran
  - load value of x from location L into a register Ra
  - add 1 into Ra [ignore overflow?]
  - store Ra into location L
  - Three assembler instructions. No memory.

The functional version

- (setf x (+ x 1)) in Lisp   [or other functional style languages]
  - Load pointer to value of x from location L into register Rx
  - Load value of x into register Ra
  - Add 1 into Ra
  - Check for overflow: jump to bignumber routine
  - Check for a HEAP location for the answer: L2
    - If no space available, do garbage collection
  - Store L2 in heap and store (pointer to L2) in L

How functional loses

- a loop like this:

  do 100 times: x← x+1

can use up 100 cells of memory (heap)

Repairing the functional version

- (dsetv x (+ x 1)) in Lisp [or other functional style languages] // macro defining destructive version, generates (e.g. in GMP)

(gmpz_add_ui (inside x) (inside x) 1)

……………………… TARGET addend addend

Repairing using "registers"

How to generate temporary spaces/ registers/ at compile time?

```
(let ((temp1 #.(runtime-allocated-temp))
      (temp2 #.(runtime-allocated-temp))....) ....)
  (…hairy arithmetic needing temporaries temp1, temp2, ..)
```

If compiled nicely, "temp2" might even be allocated on a stack, and the loop might use 1 (or zero) cells of memory.

..

So the Problems can be fixed at some inconvenience.

Superfast GC

Very clever compiler (stack allocate vars etc.)

Special encoding for likely inner-loop stuff like INOB.. small integers stored as “pointers”

Non-functional versions like (add-destroying-arg1 x 1)   ;;;overwrite the location where x is stored...

Compile CAS programs into Fortran, C, (Lisp, assembler). Especially prior to num. integ. or plotting (functions from $R \rightarrow R$ or $R^2 \rightarrow R$)

Even if CAS has bignums, link to outside..

- Consider super-hacked bignums, bigfloats
  - GMP
  - ARPREC

Why might GMP be faster?

- Representation of bignum b is (essentially) a triple:
  - Maximum allocated length in words
  - Actual length in words (times sign of b)
  - Array of words in base $\beta = 2^k$)
    - k might be 16, 29, 31, …
- Hacked mercilessly, with occasional pieces in assembler, depending on which version of Pentium II, III, IV, …AMD, Sparc, etc , cache size, you have, and which compiler, etc

The size of k is critical

- Doing an “n^2” operation where n is the number of words is 4 times faster if you can double the size of k.
- Note that the operation of multiplying 32 bits by 32 bits to get 64 bits tends to be unsupported by higher-level languages, unsupported by hardware, too.
- If you are using ANSI C, you might have to choose k=16. (Done by some Lisp systems).

What about MPFUN, ARPREC ?

- Work by a smaller team (led by David Bailey, first at NASA, now at NERSC/LBL)
- Similar in general outline to GMP
- Takes advantage of IEEE float std
- Uses arrays of 64-bit FLOATS / 48 bit fraction – wastes exponent (☹)
- Supports calc with big-exponent modest precision (for scaling computations)
- Can take advantage of multiple float arithmetic units ☺
- Number theory, experimental mathematics.

Any other clever ideas?

- Double/ doubled-double (quad)
- Doubled-quad (etc)

- Sparse bigfloats e.g. 3£10^{300}+4£ 10^{-300} does not need 600 decimal digits. It seems to need only 2. (Doug Priest, J. Shewchuk)

Why restrict outside libraries to floats?

- Consider super-hacked algebra stuff too, e.g. look around for libraries to do
  - Integer factorization
  - Polynomial factorization
  - Grobner basis reduction (A minor industry!)
  - Plotting (Forever popular)
  - (whatever else).. Web search for Math via Google?