Need:
infinite_binary_test.m
Loss_of_sig.m
loop_error.m

DAY 3.

Error.

Hand in HW 1
HW 2 assigned.
Hand in using DropBox!
Type up in LATEX.

Overview – Other Forms of Error
Ch. 2 p. 33–70

Today I will just go over some important points. Please read the chapter to better understand all the ways error can accumulate in a calculation.

Numerical Analysis — How to preform a sequence of operations to approximate the solution to a difficult problem.

Sources of Error:

1. Modeling Error — when we write down equations that model physical systems we make assumptions.
   - If the model has error the numerical approximation will also have error!
   - Know the limitations of the models you use!
2. Blunders and Mistakes — It is easy to make a simple mistake [+ instead of −]
   - Always start by running your code for a simple case that you either know the soln or can do by hand.
3. Physical Measurement Error — Calculations based on physical data will always contain error from measurement.
   - We try to limit the propagation of this error and know the limitations of our output SIGNIFICANT FIGURES.

4. Machine representation and arithmetic errors. — computers and calculators must round or chop numbers. They can't store an infinite number of decimal places.

5. Method Error — These are the major terms of error that we will consider all semester Called Truncation or Discretization error this type of error arises from the method itself.

FIRST — HOW DO COMPUTERS STORE DATA ?

Integers ... pretty straight forward.
Convert the number to binary and store.

Floating point numbers ... we need to store decimals but don't want to waste space with zeros.
0.0002 only need 1 number and some placeholders.

We store very large and very small numbers in the form

$$x = \sigma \cdot \bar{x} \cdot 10^{e}$$

σ = sign ± 1
$\bar{x}$ = significand $1 \leq \bar{x} \leq 10$
e = exponent, an integer

Ex. 156.32 we store $\sigma = 1$ $\bar{x} = 1.5632$ $e = 2$

Now computers do this except they use binary #'s.

WHY? Binary, or base 2, numbers are great for computers because computers are just big on/off switches.

10011100 .... computer can store as on or off.

Ex How do we read $156 = (156)_{10}$

Each space is a power of TEN

0	0	1	5	6
10,000	1,000	100	10	1

$= 1 \times 100 + 5 \times 10 + 6 \times 1 = 156$

Decimals 0.10

0	1	0	...
1	1/10	1/100	

$= 0.10$

How does a computer read $156 = (10011100)_2$

Each space is a power of TWO

1	0	0	1	1	1	0	0
128	64	32	16	8	4	2	1

$= 1 \times 128 + 1 \times 16 + 1 \times 8 + 1 \times 4$

$= 156$

Decimals.

1	1/2	1/4	1/8	1/16	...

This can acutally lead to some problems

Convert $X = 0.10$ to binary:

0.10×2	$=$	0.20	1/2 none
0.20×2	$=$	0.40	1/4 none
0.40×2	$=$	0.80	1/8 none
0.80×2	$=$	1.60	we need 1 1/16 plus...
0.60×2	$=$	1.20	we need 1 1/32 plus...
0.02×2	$=$	0.40	

SO $0.10 = (0.0001100110011\ldots)_2$

a finite decimal fraction can become an infinite binary fraction.

This can be a problem... depending on how many bits of storage space your computer/program uses.

SINGLE PRECISION: 32 bits of storage for each #
DOUBLE PRECISION: 64 bits of storage for each #

Freemat automatically double precision.
Fortran or C you must specify.

extended precision 80
quadruple. 128

What problems can this cause? Freemat Example. Infinite_binary_test.

TODAY — Different Types of Machine and Arithmetic Errors:

A. ROUNDING vs. CHOPPING.

- when our #'s have more digits than we can store we must do something.

NaN or Infty — when the number is larger that the machine can store.

CHOPPING — just drop the extra digits.
- Up to twice the error of rounding.
- The sign of the error is always same as sign of the #

ROUNDING — Chop of the number but round based on the last digit.
- can have 1/2 the error of chopping
- Sometimes round up sometimes down so these errors can cancel out.
  Matlab/FreeMat do this.

How much you chop or round depends on precision.

B. Loss of Significance Errors. — happens when you subtract two numbers that are nearly equal.

Ex Evaluate $f(x) = x\left[\sqrt{x+1} - \sqrt{x}\right]$

Keep just 4 decimal places:

x = 100

$\sqrt{100} = 10 \quad \sqrt{101} = 10.0499$

$\sqrt{101} - \sqrt{100} = 0.0499$

$100 \times 0.0499 = 4.9900$

BUT the real answer is more like 4.98756 we actually rounded so that our accuracy is only to two decimal places.

Freemat Example

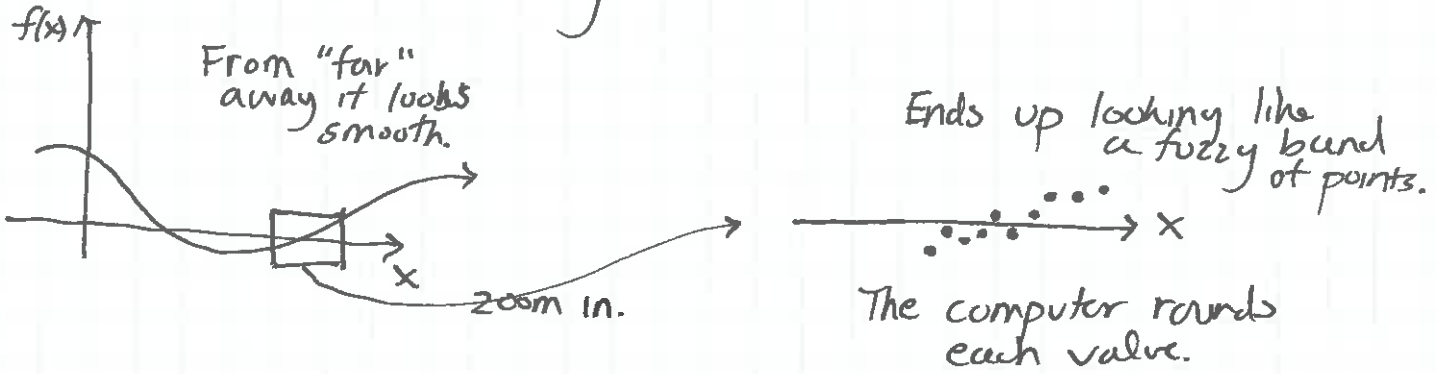
LOSS_OF_SIG.

It is much better to rewrite $f(x)$ to avoid subtraction:

$$f(x) = x\left[\sqrt{x+1} - \sqrt{x}\right]\frac{\left[\sqrt{x+1} + \sqrt{x}\right]}{\sqrt{x+1} + \sqrt{x}} = \frac{x}{\sqrt{x+1} + \sqrt{x}}$$

C. NOISE IN FUNCTION EVALUATION.

Even when evaluating continuous functions we can get numerical noise because of rounding errors.



D. UNDERFLOW and OVERFLOW

→ numbers are too small

→ numbers are too big.

Each of these types of error can propagate through a calculation. We want to be careful how we program things.

Ex LOOP ERROR. — ROUNDING

compare:

For $j = 1:100$

$x = a + jh$ and $x = x + h$

two operations and only two chances for rounding errors.

$h = 0.1$ non finite binary expression. ROUNDING.

j additions so there are j possible rounding errors.

Freefmt Example Loop_error.

Absolute Error = True Value − Approx Value = $X_T - X_A$

Relative Error = $\frac{X_T - X_A}{X_T}$ a better predictor of the severity of the error!