Data Analysis in Geophysics
ESCI 7205

Bob Smalley
Room 103 in 3892 (long building), x-4929

Tu/Th - 13:00-14:30
CERI MAC (or STUDENT) LAB

Lab – 4, 09/05/13

Finish off last time

(if you have not figured it out already)

The math

We need to make a matrix and a vector

$$\vec{u}(t_m : t_{m+k}) = \frac{a_0}{2} + \begin{pmatrix} \cos(\omega_1 t_m) & \cos(\omega_2 t_m) & \cos(\omega_3 t_m) & \cdots & \cos(\omega_n t_m) \\ \cos(\omega_1 t_{m+1}) & \cos(\omega_2 t_{m+1}) & \cos(\omega_3 t_{m+1}) & \cdots & \cos(\omega_n t_{m+1}) \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cos(\omega_1 t_{m+k}) & \cos(\omega_2 t_{m+k}) & \cos(\omega_3 t_{m+k}) & \cdots & \cos(\omega_n t_{m+k}) \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ \cdots \\ a_n \end{pmatrix}$$

$$\vec{u}(t_m : t_{m+k}) = \frac{a_0}{2} + \vec{W}\,\vec{a}$$

$$\left(Note:\ \omega_n = n * \omega_0\right)$$

$$a_n = \frac{1}{2}\left(\sin(n\pi x_s / L)\exp\left[-(\omega_n \tau)^2 / 4\right]\right)\sin(n\pi x_r / L)$$

Look at weights first

Look at arguments of the sines

$$\left(Note: \ \omega_n = n * \omega_0\right)$$

$$a_n = \frac{1}{2}\left(\sin\left(n\pi x_s / L\right)\exp\left[-\left(\omega_n \tau\right)^2 / 4\right]\right)\sin\left(n\pi x_r / L\right)$$

$$\begin{array}{lcccc} n\pi x_s / L & = & \boxed{n} & \pi / L & x_s \\ n\pi x_r / L & & \boxed{n} & \pi / L & x_r \end{array}$$

We need two vectors of integers (that count/id spatial frequencies)

Scaled by π/L, and either x_s or x_r.

Look at weights first

Look at argument of exp.

$$\left(Note:\ \omega_n = n * \omega_0\right)$$

$$a_n = \frac{1}{2}\left(\sin\left(n\pi x_s / L\right)\exp\left[-\left(\omega_n \tau\right)^2 / 4\right]\right)\sin\left(n\pi x_r / L\right)$$

$$\begin{array}{lclll} n\pi x_s / L & = & \boxed{n} & \pi / L & x_s \\ n\pi x_r / L & & \boxed{n} & \pi / L & x_r \end{array}$$

We need one vector of integers (that count/ id temporal frequencies)

Scaled by $\pi c/L$

Now can make time independent part

```
timeindep= sin(?) ? sin(?) ? exp(?);
```

Now make matrix

$$\begin{pmatrix} \cos(\omega_1 t_m) & \cos(\omega_2 t_m) & \cos(\omega_3 t_m) & \cdots & \cos(\omega_n t_m) \\ \cos(\omega_1 t_{m+1}) & \cos(\omega_2 t_{m+1}) & \cos(\omega_3 t_{m+1}) & \cdots & \cos(\omega_n t_{m+1}) \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cos(\omega_1 t_{m+k}) & \cos(\omega_2 t_{m+k}) & \cos(\omega_3 t_{m+k}) & \cdots & \cos(\omega_n t_{m+k}) \end{pmatrix}$$

Each column is $cos(\omega_n t)$

We already have a vector of the ω_n

We need a vector for the times (integer count * dt).

Now make matrix

$$\begin{pmatrix} \cos(\omega_1 t_m) & \cos(\omega_2 t_m) & \cos(\omega_3 t_m) & \cdots & \cos(\omega_n t_m) \\ \cos(\omega_1 t_{m+1}) & \cos(\omega_2 t_{m+1}) & \cos(\omega_3 t_{m+1}) & \cdots & \cos(\omega_n t_{m+1}) \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cos(\omega_1 t_{m+k}) & \cos(\omega_2 t_{m+k}) & \cos(\omega_3 t_{m+k}) & \cdots & \cos(\omega_n t_{m+k}) \end{pmatrix}$$

Now we need to make the matrix of arguments to the cos

$(\omega_n t_m)$

Do this by multiplying the two vectors

Take the cos of the matrix

Now do the matrix multiply

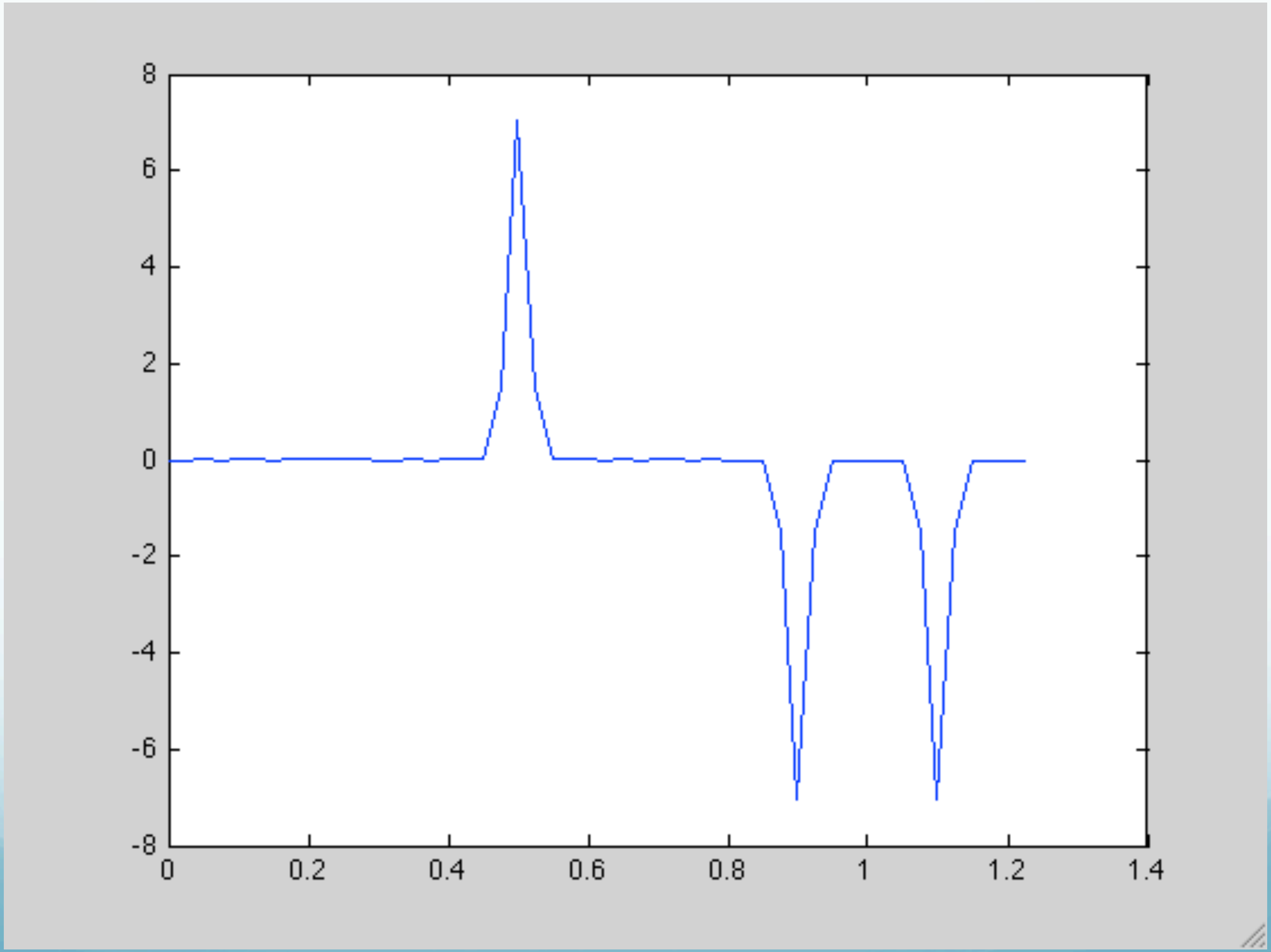
$$\vec{u}\left(t_m : t_{m+k}\right) = \frac{a_0}{2} + \begin{pmatrix} \cos(\omega_1 t_m) & \cos(\omega_2 t_m) & \cos(\omega_3 t_m) & \cdots & \cos(\omega_n t_m) \\ \cos(\omega_1 t_{m+1}) & \cos(\omega_2 t_{m+1}) & \cos(\omega_3 t_{m+1}) & \cdots & \cos(\omega_n t_{m+1}) \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cos(\omega_1 t_{m+k}) & \cos(\omega_2 t_{m+k}) & \cos(\omega_3 t_{m+k}) & \cdots & \cos(\omega_n t_{m+k}) \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ \cdots \\ a_n \end{pmatrix}$$

$$\vec{u}\left(t_m : t_{m+k}\right) = \frac{a_0}{2} + \vec{W}\ \vec{a}$$

Don't worry about the first term – it just moves the seismogram up and down.
Don't worry about the factor of 2 we forgot, it just scales it.

We are after how to do it.

Synthetic seismogram produced by Matlab code on previous slide.



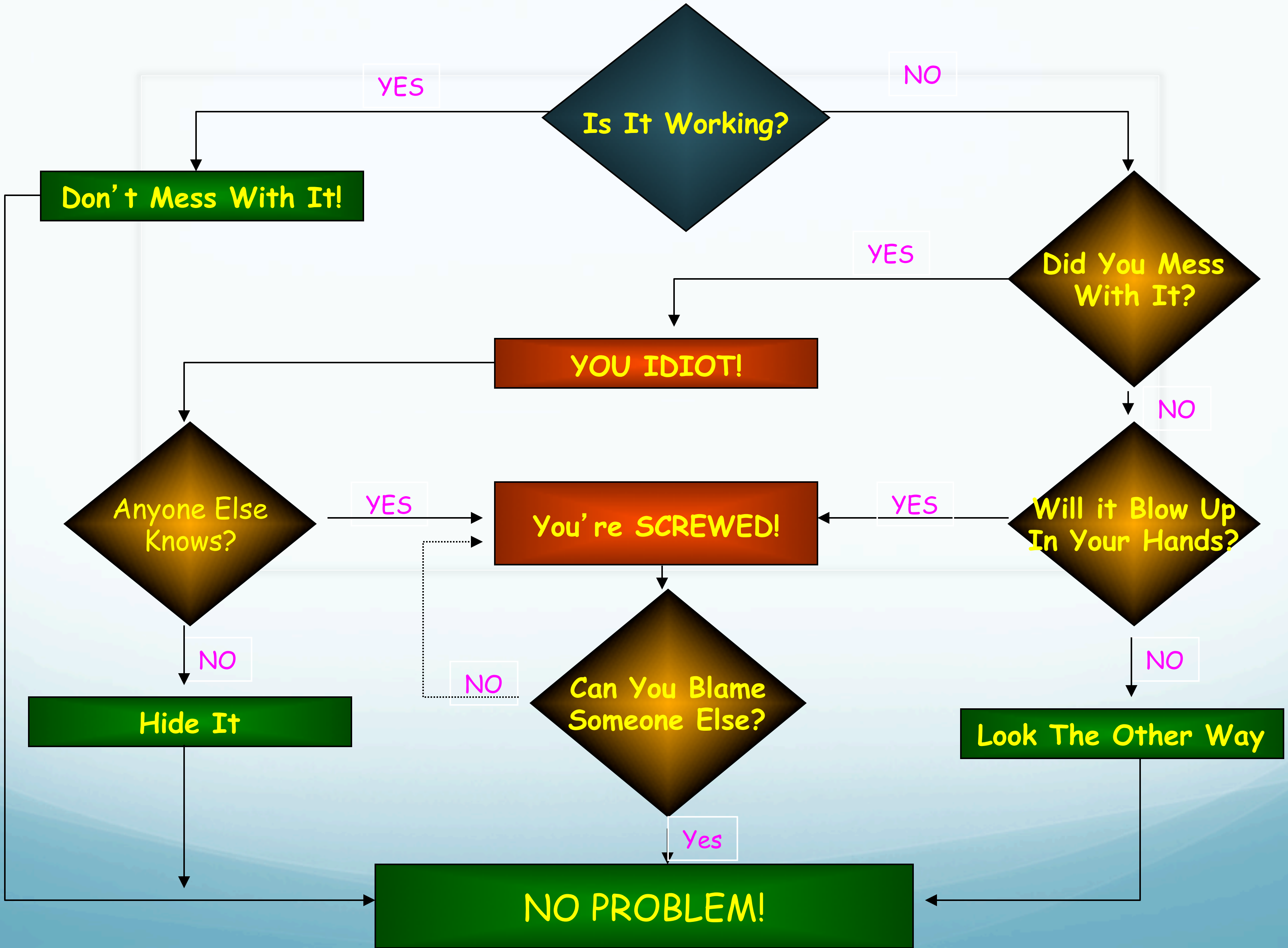
Intro to programming

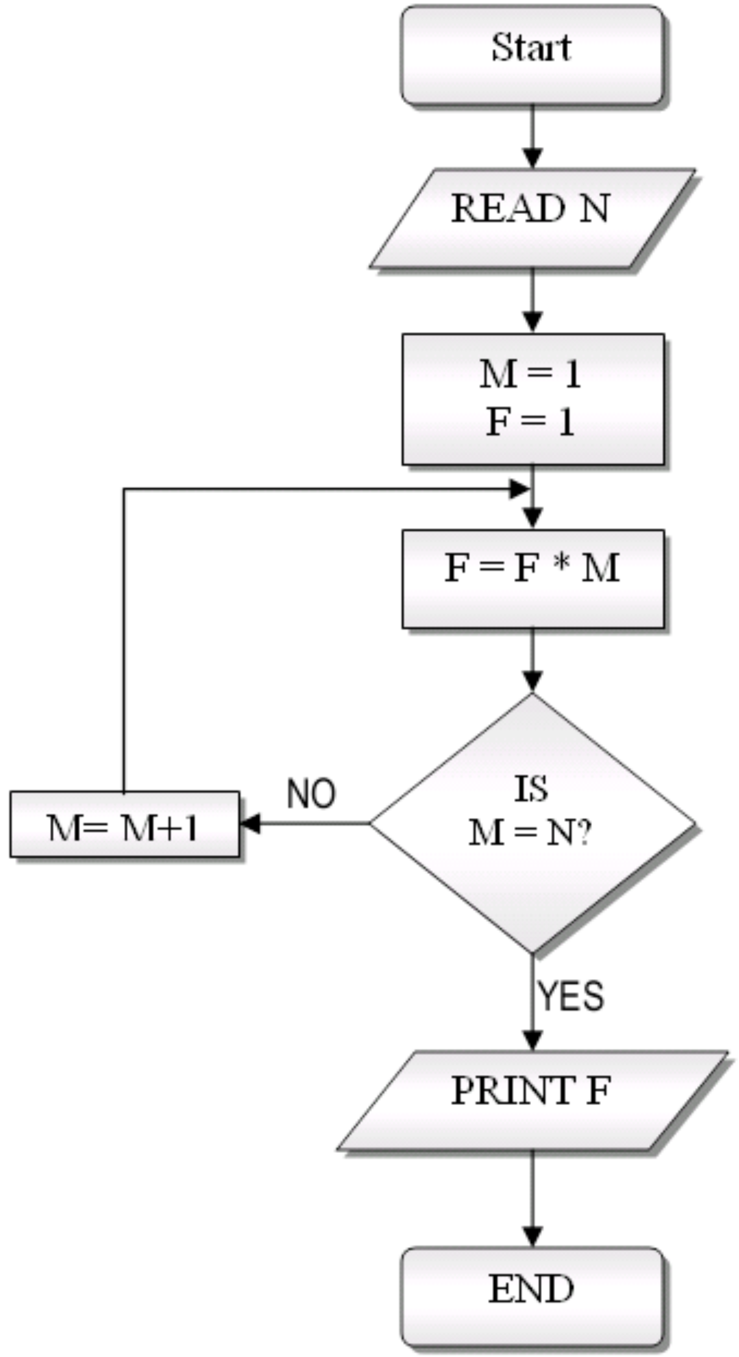
So far our "programming" has been just using the computer or Matlab as a big calculator.

We just gave it equations to evaluate.
The computer does not get bored doing the same thing over and over in a loop (or multiplying matrices).

But what about situations where what we do next depends on the previous results.
Iterating to a solution for example.

Flowchart For Problem Resolution





Flowchart for computing N!

Has
- tests/decisions
- loop

Matlab documentation

http://www.mathworks.com/help/matlab/

Plus thousands of pdf, powerpoints, etc. found on the web

Plus thousands of programs at

http://www.mathworks.com/matlabcentral/fileexchange/

And lots of individual web sites.

Let's say we want to know how many times we have to add 0.1 to get to 1

What would you do?

You could try something like

```
x=0.1;
rsum=0;
cnt=0;
while 1
    rsum=rsum+x;
    cnt=cnt+1;
    if rsum == 1, break
    end
end
display(rsum)
display(cnt)
```

OK, that's not working for some reason. Try something more reasonable since we know the answer should be 10. Replace the red lines with the green line.

```
x=0.1;
rsum=0;
cnt=0;
while 1
    cnt=cnt+1;
    rsum=rsum+x;
if rsum == 1, break
    end
end
display(rsum)
display(cnt)
```

```
x=0.1;
rsum=0;
for cnt=1:20
    rsum=rsum+x;
    if rsum == 1, break
    end
end
display(rsum)
display(cnt)
```

So what is going on?

Math on the computer is not the same as Math in your Math classes!

Finite precision representation of numbers on the computer.

So what is going on?

1/3 is never ending decimal number

On computer is approximation at number of digits the computer stores
(you would also get the wrong answer adding up 0.333 three times!)

Computer dies things in base 2, not base 10

In base 2, both 1/3 and 1/10 are both never ending decimals.

So – will rarely get "real" (non integer) numbers to be "equal" on the computer

(can get integers to be equal, count how many times you do a loop for example, can test for equals and it will always work)

Two kinds of numbers in non Matlab world

Integer – counting numbers
Floating point – subset of rational numbers

Integers on computer are pretty much simple base 2 number
(actually it is a little bit more complicated to handle + and – without wasting a bit to store sign, but detail we don't need now).

Real numbers (non round, non integer) need something more complicated.

Based on regular way we write numbers, plus scientific notation.

So what is this?

10.1

Ten and one tenth
(or two and a half?)

With the idea of zero (incredibly important) and positional notation this is

$1 \times 10^1 + 0 * 10^0 + 1 * 10^{-1}$
(or $1 \times 2^1 + 0 * 2^0 + 1 * 2^{-1}$)

On the computer we represent non integer numbers in something called floating point.

It is in base 2

$-1^s \times (a.b) \times 2^n$

Where
s is one of 0 or 1
a.b is a number with m total digits and an asumed decimal point (who's position is in a predetermined place)
And n is an n digit exponent.

So we have m digits to specify the number (this determines the precision of our numbers).

With m base two digits we can count from

0 to 2^{n-1}

So for m=3, I can count from 0 to 7, a total of 8 values

000,001, 010, 011, 100, 101, 110, 111

And we have n digits to specify the exponent.

So we get

A number with n significant base 2 digits followed by 2 raised to some power.

This is just scientific notation (in base 2) with a stated number of significant digits.

We need to pick n and m.

To make using memory easier the bits (base 2 digits, a 0 or 1) are organized into larger units that are typically handled together

Byte – 8 bits
Word – 2 bytes (16 bits, early computers)
Word – 4 bytes (current computers)

(taking the mouth analogy too far a nibble is 4 bits)

A floating point number is made up of 4 bytes (32 bits) (by convention) with

1 sign bit for the number

The number part (called the mantissa) with 23 bits

and

an exponent with 8 bits

And it is known as single precision floating point.

Matlab does all airthmetic in double precision floating point arithmetic.

(another thing that slows it down, integer arithmetic is faster – gets done in the cpu, floating point arithmetic is slow – has to go to floating point procesor).

Matlab uses double precision floating point.

Take what we had before, but "double" everything.

Use 8 words (64 bits) with

1 sign bit
52 mantissa bits
11 exponent bits

Gives more significant digits and larger range of exponents.