Matlab

Linear Algebra etc.

Nick Maclaren

Computing Service

nmm1@cam.ac.uk, ext. 34761

October 2008

Please Interrupt

This course assumes a fair amount of background

1: that you already happy using Matlab
E.g. basic use, syntax and error handling

2: that you already know some linear algebra
At least up to elementary use of matrices
It will refer to a bit more, but will explain

- If you don't understand, please interrupt

Don't feel afraid to ask any question you want to

Beyond the Course

Email scientific–computing@ucs for advice

- Ask almost anything about scientific computing

http://www–uxsup.csx.cam.ac.uk/courses/...
...//Matlab/

http://www.mathworks.com/access/helpdesk/...
.../help/techdoc/

What Is Linear Algebra?

Could call it the arithmetic of matrices
It's more general than you might think

Need to explain some mathematics
Don't Panic – it will be over–simplified!

You can do a great deal in Matlab

- As always, follow the motto "festina lente"

"Make haste slowly" – i.e. start with simple uses

Structure Of Course

- Overview of what analyses are possible
- Basic matrix facilities in Matlab
- Real and complex linear algebra
- Summary of more advanced matrix facilities

What Are Matrices?

Effectively a 2–D rectangular grid of elements

1.2	2.3	3.4	4.5
5.6	6.7	7.8	8.9
9.0	0.1	1.2	2.3

1–D arrays are also called vectors

n–D arrays may be called tensors

Won't cover them, because poor in Matlab

Yes, mathematicians, I know – over–simplification!

Terminology

Some people use matrix to mean only 2–D
Others talk about matrices of any rank
The standard term is matrix algebra not array algebra

Matlab tends to use the term array generically

- But it also uses array for other purposes

- This course often uses matrix generically

But uses vector or n–D where it matters

Elements of Matrices

- These are not limited to real numbers

Can actually belong to any mathematical field
Examples:

- Real ($\mathbb{R}$) or complex ($\mathbb{C}$) numbers
- Ratios of integers (rational numbers)
- Ratios of polynomials/multinomials
- And more

Course covers real but mentions complex
You use them in almost exactly the same way

Reminder

123456789 is an integer

12345.6789 is a real number

123.45+678.9i is a complex number

complex(123.45,678.9) creates the same number

What Can We Do?

All of basic matrix arithmetic, obviously
Including some quite complicated operations

Solution of simultaneous linear equations
Eigenvalues and eigenvectors
Matrix decompositions of quite a few types

Plus (with more hassle) their error analysis

Fourier transforms are just linear algebra, too

Physics, Chemistry etc.

Anything expressible in normal matrix notation

- That's almost everything, really!

But that isn't always practically possible
Matlab is slower than NAG, but you can call NAG

- Working with expressions is not covered

You need to use Mathematica for that
But you can often get much more information

Statistical Uses

- Regression and analysis of variance
- Multivariate probability functions

Calculating the errors is the tricky bit
It's NOT the same as in most physics!

- Also Markov processes – finite state machines

This is where transitions are probabilistic
Working with these is just more matrix algebra

- Standard textbooks give the matrix formulae

You just carry on from there …

Matlab and Matrices

Will describe how Matlab provides them

And explain how to construct and display them

And perform other basic matrix operations

Matrix Notation (1)

Conventional layout of a 4x3 matrix $\boldsymbol{A}$
Multiplied by a 3 vector

11	12	13		7		290
21	22	23	X	8	->	530
31	32	33		9		770
41	42	43				1010

$\boldsymbol{A_{3,2}}$ is the value 32

$\mathbf{530}$ is $\mathbf{21 \times 7 + 22 \times 8 + 23 \times 9}$

Matrix Notation (2)

Now we do the same thing in Matlab
All of the details will be explained in due course

```
a = [ 11 , 12 , 13 ;  21 , 22 , 23 ;  31 , 32 , 33 ;  41 , 42 , 43 ]
          11   12   13
          21   22   23
          31   32   33
          41   42   43

b = [ 7 ;   8 ;   9 ]

a ( 3 , 2 )   ->   32
a * b   ->   [ 290 , 530 , 770 , 1010 ]
```

Notation in Papers

There are a zillion – one for each sub–area
'Standard' tensor notation has changed, too
Here is another over–simplification

A_i, A^i, $\bar{A}$ or $\tilde{A}$ is a vector
A_i may also refer to element i of vector A
B_{ij} or B_i^j is a matrix

$A_i B_{ij}$ often means $\sum_i A_i \times B_{ij}$

Algorithms may use A(i) or A[i] and B(i,j) or B[i,j]

Matrices as Lists (1)

We can input a list of values

a = [9.8 , 8.7 , 7.6 , 6.5 , 5.4 , 4.3 , 3.2 , 2.1 , 1.0]

b = [1.2 , 2.3 , 3.4 , 4.5 ;   5.6 , 6.7 , 7.8 , 8.9 ;
      9.0 , 0.1 , 1.2 , 2.3]

1.2	2.3	3.4	4.5
5.6	6.7	7.8	8.9
9.	0.1	1.2	2.3

You use commas (,) between numbers in rows
And semicolons (;) between successive columns

Matrices as Lists (2)

Values need not be simple numbers
Expressions using defined variables are allowed

```
x = 1.23

y = 4.56

c = [1+x^2,x*y ;  -x*y,1+y^2]

  2.5129   5.6088
 -5.6088  21.7936
```

Row Major or Column Major?

I find those terms seriously confusing
We want to know which subscript varies fastest

- Matlab like Mathematica and C, NOT Fortran

First subscript (rows) varies slowest
Last subscript (columns) varies fastest

```
a = [ 11 , 12 , 13 ;  21 , 22 , 23 ;  31 , 32 , 33 ;  41 , 42 , 43 ]
a ( 3 , 2 )   ->   32
```

11 , 12 , 13 is the first of 4 rows
11 , 21 , 31 , 41 is the first of 3 columns

Vectors

A vector can be either a row or column

- And you need to use the right one

```
a = [ 1.2 , 2.3 , 3.4 ]
      1.2   2.3   3.4

b = [ 1.2 ; 2.3 ; 3.4 ]
      1.2
      2.3
      3.4

b'
      1.2   2.3   3.4
```

Index Ranges (1)

a : b means all values from a to b

```
a=[1,2,3,4,5,6,7,8,9,10]

a(5:7)  ->  [5,6,7]


a=[1,2,3; 4,5,6; 7,8,9]

a(1:2,2:3)  ->  [2,3;5,6]
```

Index Ranges (2)

Just a : means all values in a row or column
Can be combined with indices and index ranges

```
a=[1,2,3; 4,5,6; 7,8,9]

a(:,2:3)
    2   3
    5   6
    8   9

a(2,:)  ->  [4,5,6]
    4   5   6
```

Matrix Constructors (1)

```
zeros ( 2 )    ->   [ 0 , 0 ;   0 , 0 ]
zeros ( 1 , 2 )   ->   [ 0  ,  0 ]
zeros ( 2 , 1 )   ->   [ 0  ;  0 ]

1.23 * ones ( 2 )   ->   [ 1.23 , 1.23 ;   1.23 , 1.23 ]

rand ( 2 )   ->   [ 0.9649 , 0.9706 ;   0.1576 , 0.9572 ]

eye ( 3 )   ->   [ 1 , 0 , 0 ;   0 , 1 , 0 ;   0 , 0 , 1 ]
```

Most have several options and extra features

Matrix Constructors (2)

There are also many specialised standard matrices
These are Hilbert and inverse Hilbert ones

```
hilb ( 3 )
      1.0000      0.5000      0.3333
      0.5000      0.3333      0.2500
      0.3333      0.2500      0.2000
invhilb ( 3 )
        9   –36    30
      –36   192  –180
       30  –180   180
```

Also companion, Hankel, Toeplitz matrices, etc.

- Look them up when and if you need them!

Matrix Constructors (3)

You can construct linear sequence vectors

```
linspace ( – 2.3 , 8.9 , 7 )
–2.3000   –0.4333   1.4333   3.3000   5.1667
    7.0333   8.9000
```

And then assign them to rows or columns
Try this and see what it does:

```
a = zeros ( 7 , 3 ) ;
a ( : , 1 ) = linspace ( 1 , 7 , 7 ) ;
a ( : , 2 ) = linspace ( – 2.3 , 8.9 , 7 ) ;
a( : , 3) = linspace ( 5.6 , 7.8 , 7 ) ;
```

Advanced Construction

Matlab has lots of more advanced features
If you can't do what you want, look for something

```
a = [ 11 , 12 , 13 ;   21 , 22 , 23 ;   31 , 32 , 33 ]
[ a , a , a ]
[ a ; a ; a ]
repmat ( a , 5 , 7 )
```

Look in Elementary Matrices and Arrays
And in Array Manipulation

Importing Data

Unfortunately, Matlab isn't brilliant at this
This course won't go into how to do it
General advice:

- Use load and save for storage

Uses Matlab's own format – version dependent

- Use importdata for numbers in text form

Input them in the format that Matlab expects

- Use Python (or Perl) for other formats

Write the data out in importdata format

Basic Functions (1)

disp(A) displays A – compare that with A

```
a = [ 11 , 12 , 13  ]

a

    a =
          11    12    13

disp(a)

          11    12    13
```

Basic Functions (2)

ndims is the rank (number of dimensions)
size gives the actual dimensions

```
a = [ 11 , 12 , 13 ;    21 , 22 , 23 ]

ndims ( a )
     2

size ( a )
     2   3
```

A Vector 'Gotcha'

The rank of a vector is one (1), right?

- Not usually in Matlab, it isn't

Most 'vectors' are actually degenerate matrices

```
a = [ 11 , 12 , 13 ]
b = [ 11  ;  12  ;  13 ]

ndims ( a )    ->    2
size ( a )    ->    1    3
ndims ( b )    ->    2
size ( b )    ->    3    1
```

IEEE 754 Features

Matlab has a large collection of IEEE 754 functions
Matlab's support is not IEEE 754–compliant
Nor is Java's, or most other languages'

- I strongly recommend NOT using them

See the course How Computers Handle Numbers

The reasons are far to complicated to go into here
But not even experts can use them reliably
Please ask if you want to know more details

Basic Operations

A + B	addition
A – B	subtraction
A * B	matrix multiplication
A ^ K	Kth power of matrix
A '	normal transpose (conjugated)
A .'	transpose (non–conjugated)

All of those work as in normal mathematics
Generally, use only positive integer power K

Matrix Multiplication

Remember that this is NOT commutative
A∗B is generally not equal to B∗A

Try the following if you need to convince yourself

```
A = rand(5)
B = rand(5)
A*B
B*A
```

And the product of two symmetric matrices
will usually be unsymmetric

Matrix Division

A / B   matrix (right) division

A .\ B   matrix left division

- Take care about numerical problems with these

Serious ones with both accuracy and range

- They are effectively linear equation solvers

We will discuss that in more detail later

It is easy to get confused when using these

Elementwise Operations

You can also do elementwise operations

A .* B	elementwise multiplication
A .^ K	elementwise Kth power
A ./ B	elementwise (right) division
A .\ B	elementwise left division

You can use any power that makes sense

Diagonalisation (1)

Can construct a diagonal matrix from a vector
Or convert a matrix into its diagonal vector

```
diag ( [ 1.23  ,  4.56 ] )   ->   [ 1.23 , 0 ;   0 , 4.56 ]
diag ( [ 1.23  ;  4.56 ] )   ->   [ 1.23 , 0 ;   0 , 4.56 ]

b = [ 1.2 , 2.3 , 3.4 ;  4.5 , 5.6 , 6.7 ;  7.8 , 8.9 , 9.0 ]
diag ( b )   ->   [ 1.2  ;  5.6  ;  9.0 ]
```

That dual use is counter–intuitive, but this is useful:

```
diag ( diag ( b ) )
```

Diagonalisation (2)

You can also construct block diagonal matrices
The block matrices can be of any shape

```
a = [ 11 , 12 , 13 ;    21 , 22 , 23 ;    31 , 32 , 33 ]
b = [ 100 , 200 ;    300 , 400 ]
c = 5
blkdiag ( a , b , c , c , c , b , a )
```

I strongly recommend experimenting before use
Especially if any of your blocks aren't square

More Functions (1)

max and min collapse along one dimension

```
a = [ 11 , 12 , 13 ;   21 , 22 , 23 ;   31 , 32 , 33 ]

max ( a )
    31   32   33

min ( a )
    11   12   13

max ( a , [ ] , 2 )
    13
    23
    33
```

More Functions (2)

prod and sum use a different syntax

```
a = [ 11 , 12 , 13 ;    21 , 22 , 23 ;    31 , 32 , 33 ]

sum ( a )
    63    66    69

prod ( a )
    7161        8448        9867

sum ( a , 2 )
    36
    66
    96
```

More Functions (3)

Most basic mathematical functions work elementwise

```
a = [ 1 , 2 , 3  ]
b = [ –2 , 0 , 1 ]

sqrt ( a )
      1.0000    1.4142    1.7321
exp ( a )
      2.7183    7.3891   20.0855
atan2 ( a , b )
     2.6779    1.5708    1.2490
```

arrayfun can be used for your functions

More Functions (4)

The usual inner and cross products of vectors

dot(V1,V2) and cross(V1,V2)

But no outer product nor normalisation

You have to program those yourself, by hand; e.g.

```
a = [ 1 , 2 , 3 ]
b = a / sqrt ( dot ( a , a ) )
    0.2673    0.5345    0.8018
```

Other absences for multi–dimensional matrices

Practical Break

We shall now stop for some practical exercises
These are mainly for inexperienced Matlab users

- I don't care which system you use

Just don't expect any help with Microsoft!

I use matlab –nodisplay under Linux

Numeric Linear Algebra

This is defined only for real and complex
Matlab uses IEEE 754 64–bit – c.15 sig. figs

- This has some special mathematics to itself

Can do a lot more than for general matrices

- Are going to cover only the simplest analyses

There's a huge amount more in Matlab
And even more in books and papers

Determinant

det gives the determinant
As usual, watch out for numerical problems

det (hilb (10)) –> 2.1644e–53
det (invhilb (10)) –> 4.6207e+52

So far, so good. Now let's try a harder problem

det (hilb (20)) –> –1.1004e–195
det (invhilb (20)) –> 3.1788e+254

Norms

norm gives a choice of induced norms
Calculate other norms using the usual formulae

norm(A) is the 2–norm of A
    I.e. the largest singular value of A
norm(A,1) is the 1–norm or largest column sum
norm(A,inf) is the ∞–norm or largest row sum
norm(A,'fro') is the Frobenius norm
normest(A) is an approximation to norm

norm(V,p) is the p–norm for vector V and integer p

Condition Numbers

cond(A) is the 2–norm condition number
cond(A,opt) is the opt–norm condition number
    opt can be anything allowed in norm

rcond(A) is LAPACK approx. reciprocal cond. no

condeig(A) is the vector of eigenvalue cond. nos
Relevant only for unsymmetric matrices

Other Functions

trace(A) is the sum of diagonal elements

expm(A) is the matrix exponential

inv(A) is the matrix inverse
You very rarely should be using this

rank, null, orth etc. for subspaces
Most of these are for relatively advanced use

- Watch out for serious numerical 'gotchas'

Linear Equations (1)

In the common, simple cases, Just Do It

```
a = [  4.2 ,   2.2 , – 3.9 ,   9.3 ,   0.1 ;
       8.6 ,   0.0 ,   0.7 , – 2.3 , – 0.3 ;
       8.4 , – 5.9 , – 8.1 ,   9.6 ,   3.8
     – 0.8 , – 9.4 , – 9.9 ,   9.9 ,   5.0 ;
     – 1.3 , – 8.1 ,   0.6 , – 9.2 , – 7.3 ]

b = [ – 6.8 , 2.3 , 2.7 , – 7.0 , 2.0 ]

linsolve ( a , b' )

[ 1.45411 , –12.4949 , 24.5078 , 11.8408 , 0.422917 ]
```

Linear Equations (2)

How do these relate to matrix division?
Yes, I find this very confusing, too!

```
a  =  rand ( 5 )
b = rand ( 5 )

linsolve ( a , b )
a \ b
( b' / a' )'
```

All of the above are more–or–less equivalent

Linear Equations (3)

How did I work that out? Trying every combination ...
I then worked out what Matlab's authors were thinking

- Always run a cross–check when doing that

I.e. check you have done the right calculation

```
a*linsolve(a,b)  ->  b
(b/a)*a  ->  b
```

Linear Equations (4)

It's critical to look for its warnings

```
linsolve ( [ 1.2 , 3.4 ; 2.4 , 6.8 ] , [ 1.0 ; 1.0 ] )

Warning: Matrix is singular to working precision.
–Inf
 Inf

linsolve ( hilb ( 20 ) , rand( 20 , 1 ) )

Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. RCOND = 1.995254e–19.
```

Linear Equations (5)

- If any, the results are often complete nonsense
They may be NaNs, infinities, zeroes
⇒ Or plausible values that are just plain wrong

- Matlab won't stop by itself on a warning
You have to do the checking, by hand

The command dbstop if warning looks useful here
But works only if you are running the debugger!

- Naturally, this applies to all warnings
Not just ones that occur in linear equations

Decompositions

You can get just the decompositions

The main ones are chol(A) and lu(A)
Cholesky for real positive definite symmetric only
LU for pretty well any square matrix

qr(A) is also used for eigenvalues

pinv(A) is the Moore–Penrose pseudo–inverse

Least Squares Solutions (1)

Matlab uses these for over–determined problems
Generally, everything does what you expect

```
a = rand ( 10 , 3 )
b = rand ( 10 , 1 )
linsolve ( a , b )

   0.5731
   0.1851
   0.2837
```

You will get different random results, of course

Least Squares Solutions (2)

lscov can handle known covariances
Look it up if you need to do that

But what if you need to do more advanced work?
Including any serious statistics or regression

- Use the Matlab statistics toolbox (extra cash)
- Statistical package, like Genstat or SPSS
- Program the formulae yourself, if you can

Fourier Transforms (1)

Matlab doesn't call them linear algebra
Look under Data Analysis for `fft*`

a = [–0.92 , 9.1 , 2.3 , 5.7 , 4.9 , –2.8 , –5.6 ,
6.7 , –7.0 , 9.0]
b = fftn (a)
21.4 , 11.8 – 14.1i , –4.6 + 3.8i , 9.9 – 5.2i ,
–15.4 + 15.9i , –34.0 , –15.4 – 15.9i ,
9.9 + 5.2i , –4.6 – 3.8i , 11.8 + 14.1i
ifftn (b)
–0.92 , 9.1 , 2.3 , 5.7 , 4.9 , –2.8 , –5.6 ,
6.7 , –7.0 , 9.0

Fourier Transforms (2)

- fftn and fftn work on matrices, too

They are the multi–dimensional array forms

fft/ifft and fft2/ifft2 look more obvious

- But they have a very serious ‘gotcha’

Unexpected behaviour on too many dimensions

fft/ifft work on the first dimension only

I.e. they apply multiple 1–dimensional FFTs

I haven’t experimented with what fft2/ifft2 do

Fourier Transforms (3)

You can make fft/ifft work on another dimension
Using the same nasty syntax as for max/min
The following two operations are equivalent – try them

```
a = rand ( 5 )

fft2 ( a )

fft ( fft ( a , [ ] , 2 ) )
```

Eigenanalysis (1)

- Things start to get a bit hairier, here
That is because the mathematics does

- All square matrices have all eigenvalues
But real matrices may have complex eigenvalues

- All real symmetric matrices have all eigenvectors
And those are always orthogonal
The same applies to all complex Hermitian ones

- But unsymmetric matrices may be nasty

Eigenanalysis (2)

Simple use is, er, simple

```
a = [ 4.2 , 2.2 , –3.9 , 9.3 , 0.1 ;
      8.6 , 0.0 , 0.7 , –2.3 , –0.3 ;
      8.4 , –5.9 , –8.1 , 9.6 , 3.8 ;
      –0.8 , –9.4 , –9.9 , 9.9 , 5.0 ;
      –1.3 , –8.1 , 0.6 , –9.2 , –7.3 ]
eig ( a )
      6.4585 + 9.8975i
      6.4585 – 9.8975i
     –7.2840 + 4.4546i
     –7.2840 – 4.4546i
      0.3510
```

Characteristic Polynomial

Eigenvalues are the roots of that
You can calculate it directly, if you want

```
a = [ 4.2 , 2.2 , –3.9 , 9.3 , 0.1 ;
      8.6 , 0.0 , 0.7 , –2.3 , –0.3 ;
      8.4 , –5.9 , –8.1 , 9.6 , 3.8 ;
      –0.8 , –9.4 , –9.9 , 9.9 , 5.0 ;
      –1.3 , –8.1 , 0.6 , –9.2 , –7.3 ]

poly ( a )

  1.0e+03 *
    0.0010  0.0013  0.0238  1.0845  9.7983 –3.5741
```

Eigenvectors (1)

You can get the eigenvectors almost as easily

If you don’t know the following syntax, just use it
It just assigns two results, rather than one

```
a = [ 4.2 , 2.2 , –3.9 ;   8.6 , 0.3 , 0.7 ;
        8.4 , –5.9 , –8.1 ]

[ p , q ] = eig ( a )
```

Returns the eigenvectors in the columns of p
And the eigenvalues as a diagonal matrix in q

Eigenvectors (2)

The output is omitted because of verbosity – try it

Let's check that we have got the right answer

p ∗ q / p should be the matrix a

```
a = ...
4.2000    2.2000   -3.9000
8.6000    0.3000    0.7000
8.4000   -5.9000   -8.1000
p ∗ q / p
4.2000 + 0.0000i    2.2000 - 0.0000i  -3.9000
8.6000 + 0.0000i    0.3000 - 0.0000i   0.7000 - 0.0000i
8.4000 + 0.0000i   -5.9000 - 0.0000i  -8.1000
```

Eigenvectors (3)

Here, really don’t rely on the diagnostics

```
a = [ 1.0 , 1.0  ;  0.0 , 1.0 ]
        1     1
        0     1

[ p , q ] = eig ( a )

p * q / p
        1     0
        0     1
```

Oops!   Not even a warning

Eigenvectors (4)

Let's try another of the same level of nastiness

```
a = [ 0.0 , 1.0  ;  0.0 , 0.0 ]

[ p , q ] = eig ( a )

p * q / p
Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. RCOND = 2.004168e-292.
             0     0
             0     0
```

More Eigenanalysis (1)

There are a lot of extra features and functions
I haven't investigated, and it's not one of my areas

Matlab says eig has problems with sparse matrices
There's another function, eigs, which may do better

You can get the Hessenberg and Schur forms

And then there's the question of whether to balance

More Eigenanalysis (2)

- Real symmetric and complex Hermitian are easy

You will be unlucky to have problems with those ones

Mere numerical inaccuracy excepted, of course

- Otherwise: Be Very Careful

Unsymmetric eigenanalysis is full of ‘gotchas’

- Many experts say you shouldn’t do it at all

You should use Singular value decomposition

Singular Values (1)

Singular value decomposition is called SVD
A more robust extension of eigenanalysis

Gives the same results in the simple cases
The term for these is normal matrices

Also handles non–square matrices
And ones with missing eigenvectors

If you don't know it, don't worry about it
But it's an important technique in many fields

Singular Values (2)

Try the following with a variety of matrices a

```
eig (a)
svd ( a )

[ p , q ] = eig ( a )
p * q / p   –>   a

[ r , s , t ] = svd ( a )
r * s * t'   –>   a
```

Singular Values (3)

The trivial cases give exactly the same answers
Matlab uses the reverse orders of eigenvalues

```
a = hilb ( 3 )

eig (a)
      [ 0.0027 ; 0.1223 ; 1.4083 ]

svd ( a )
      [ 1.4083 ; 0.1223 ; 0.0027 ]
```

Singular Values (4)

But SVD handles the foul cases I used correctly

```
a = [ 1.0 , 1.0 ;     0.0 , 1.0 ]
[ r , s , t ] = svd ( a )
r * s * t'
    1.0000    1.0000
    0.0000    1.0000

a = [ 0.0 , 1.0 ;     0.0 , 0.0 ]
[ r , s , t ] = svd ( a )
r * s * t'
    0    1
    0    0
```

Complex Numbers (1)

FFT and eigenalysis have produced these
We have completely ignored them – why?

- Because there has been nothing to say!

Really do use complex numbers just like real ones
Their differences don't show up in linear algebra

⇒ There is only one critical warning
Regard complex exceptions as pure poison
That means mainly overflow and division by zero

See the course How Computers Handle Numbers

Complex Numbers (2)

a = [4.2 + 2.2i , –3.9 + 9.3i, 0.1 + 0.0i ;
8.6 + 0.0i , 0.7 – 2.3i , 0.0 – 0.3i ;
8.4 – 5.9i , –8.1 + 9.6i , 3.8 – 0.8i]
b = [–6.8 + 2.3i ; 2.7 – 7.0i ; 2.0 + 0.0i]

linsolve (a , b)
0.0362 – 0.5311i
0.7195 + 0.6147i
4.0269 + 1.5706i

eig (a)
–3.2517 – 8.1715i
7.9506 + 7.7844i
4.0011 – 0.5129i

A Bit of Numerical Analysis

Very roughly, the error in linear algebra is:

$$N \times cond.number \times epsilon$$

Where N is the size of the matrix
$Cond.number$ is how 'nasty' the matrix is
$epsilon$ is the error in the values/arithmetic

- Almost always, the main error is in the input data

Good linear algebra algorithms are very accurate

$\Rightarrow$ Rounding error isn't usually the problem

Real vs Floating-Point

See "How Computers Handle Numbers"
Only significant problem is loss of accuracy

Not going to teach much numerical analysis
But it's well–understood for much of linear algebra

- PLEASE don't use single precision

And do watch out for growth of inaccuracy

Solution of Equations (1)

Let’s look at a classic numerically foul problem
The Hilbert matrix is positive definite
And horribly ill–conditioned ...

In Mathematica rational arithmetic, result is exact

```
a = HilbertMatrix [ 10 ]
b = ConstantArray [ 1 , 10 ]

c = LinearSolve [ a , b ]
{ –10 , 990 , –23760 , 240240 , –1261260 , 3783780 ,
   –6726720 , 7001280 , –3938220 , 923780 }
```

Solution of Equations (2)

{ –10 , 990 , –23760 , 240240 , –1261260 , 3783780 , –6726720 , 7001280, –3938220 , 923780 }

Now we do it in Matlab's floating–point

```
a = hilb ( 10 )
b = ones (10 , 1 )
num2str ( linsolve ( a , b ) , 6 )
```

–9.99803 , 989.83 , –23756.4 , 240207 ,
–1.2611e+06 , 3.78335e+06 , –6.72601e+06 ,
7.0006e+06 , –3.93786e+06 , 923701

Error Analysis

Traditionally, this is overall error analysis
Usually in terms of norms etc.
It is a well–understood area, with useful results

- Use the formulae for it in books etc.

Sorry, but there is no time to go into it further

Sparse Arrays (1)

Matlab has good support for sparse arrays
You use them just like dense arrays – ha, ha!

- Don't be fooled – this is a problem area

It can be done very well, but don't just rush in

First problem is many operations cause infilling
A 16 MB array can expand to 16 GB

- Sparse algorithms are designed to minimise that

Sparse Arrays (2)

Problem is that nothing comes for free

- Very often the sacrifice is numerical robustness

Some problems may give inaccurate answers, or fail
I mentioned Matlab's warning about the eig function

So proceed cautiously with sparse arrays

- Find out what best practice is before proceeding

But lots of people do it, very successfully

Esoteric Analyses

The above are the basic methods of linear algebra

- Every field has its own specialised methods

There are thousands of them and each field differs

- Matlab has a few, otherwise program them

Almost always formulae in reference books

- The only warning is to watch out for errors

Both algorithm failure and numerical inaccuracy

Practical Break

We shall now stop for some practical exercises
These use the actual linear algebra functions

- The same comments apply as before

But, before that, one more slide …

Feedback

Please fill in your green forms
I am particularly interested in the following:

- Would you have liked an intermediate course?
- Would you like a course on sparse arrays?
- Would you like a course on the NAG toolbox?
- Any other related courses that you would like?

On Matlab or any other scientific computing