# Mathematica

*Numerical Linear Algebra*

Nick Maclaren

**nmm1@cam.ac.uk**

October 2008

# Please Interrupt

This course assumes a fair amount of background

1: that you already know some Mathematica
E.g. the arcane 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

Mathematica/

http://reference.wolfram.com/mathematica/...
.../guide/Mathematica.html

# Logging In

- No practicals, as such, but are examples

Recommended to try them as I describe the topics
Can use cut-and-paste from file Examples_1.txt

If using Microsoft Windows, find mathematica
Somewhere in Applications

If not using Microsoft Windows, CTRL-ALT-DEL
Select Restart, then Linux and log in
Start by mathematica or math

You should also display Examples_1.txt

# 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 Mathematica
Far more than in packages like Matlab

- As always, follow the motto "festina lente"

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

# Structure Of Course

## This part

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

## Second part (not covered)

- Examples of symbolic linear algebra
- Development and debugging techniques

# What Are Matrices?

Effectively a 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 matrices are also called vectors

n–D matrices are also called tensors
Won't cover them, but they are easy to use

Yes, mathematicians, I know – over–simplification!

# 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

That's almost all we are going to use, though

All most scientists want to work with

# Symbolic and Rational Matrices

- Simple matrix operations work as usual

I.e. O(N) combinations of +, − and *

Others (e.g. eigenvalues) may change element type

Others (e.g. determinant) merely run like drains
Often O(N!), where N is number of elements

- So proceed very cautiously

This course will give some guidelines

# Integer Matrices etc.

Elements can be an Abelian (commutative) ring
E.g. integers ($\mathbb{Z}$) or polynomials
Difference is an Abelian ring has no division

- Simple matrix operations work as usual

Others ones may change the element type
Or they may run more slowly than you expect

```
Eigenvalues[{{1,2,3},{4,5,6},{7,8,9}}]

     3 (5 + Sqrt[33])      3 (5 - Sqrt[33])
{--------------------, --------------------, 0}
           2                     2
```

# Reminder

123456789 is an integer

12345/6789 is an rational number

12345.6789 is a real number

123.45+678.9*I is a complex number

123.45+678.9*p is a polynomial

# 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
Mathematica slower than NAG or even Matlab

- Working with expressions can be much slower

E.g. may need Cramer and not Cholesky

But you can often get much more information

- So the approaches are complementary

# 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 ...

# Mathematica and Matrices

Will describe how Mathematica 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 $A$
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 |

$A_{3,2}$ 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 Mathematica

```
a = { { 11 , 12 , 13 } , { 21 , 22 , 23 } , { 31 , 32 , 33 } , { 41 , 42 , 43 } }
b = { 7 , 8 , 9 }

TableForm [ a ]
            11  12  13
            21  22  23
            31  32  33
            41  42  43

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 $\sum_i A_i \ldots B_{ij}$

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

# Row Major or Column Major?

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

- Mathematica is like Matlab and C

Last subscript varies fastest

```
a = { { 11 , 12 , 13 } , { 21 , 22 , 23 } , { 31 , 32 , 33 } }
a [[ 3 , 2 ]]   ->   32
```

Warning: Fortran is different!

# Index Ranges (1)

Mathematica calls this the Span function

- It works inside indices only

a;;b means all values from a to b

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

Span [ 5 , 7 ]   ->   5 ;; 7

a [[ 5 ;; 7 ]]   ->   { 5 , 6 , 7 }
a [[ Span [ 5 , 7 ] ]]   ->   { 5 , 6 , 7 }
```

# Index Ranges (2)

You can omit either or both of a and b

```
a [[ 5 ;; ]]   ->   { 5 , 6 , 7 , 8 , 9 , 10 }
a [[ ;; 5 ]]   ->   { 1 , 2 , 3 , 4 , 5 }
a [[ ;; ]]   ->   { 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 10 }
```

A third argument sets a step

```
a [[ 3 ;; 8 ;; 2 ]]   ->   { 3 , 5 , 7 }
a [[ Span [ 3 , 8 , 2 ] ]]   ->   { 3 , 5 , 7 }
```

# 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 } } // TableForm
```

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

# Matrices as Lists (2)

Values need not be simple numbers

```
c = {{1 + x ^ 2, x * y}, {- x * y, 1 + y ^ 2}}

        2                          2
{{1 + x , x y}, {- (x y), 1 + y }}

TableForm[c]

             2
        1 + x     x y
                       2
        -(x y)    1 + y
```

# Matrix Constructors (1)

```
a = ConstantArray[p + q , {5}]
    {p + q, p + q, p + q, p + q, p + q}

a = ConstantArray[p + q , {2, 2}]
    {{p + q, p + q}, {p + q, p + q}}

a = IdentityMatrix[3]
    {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}}

a = DiagonalMatrix[{P, Q, R}]
    {{P, 0, 0}, {0, Q, 0}, {0, 0, R}}
```

# Matrix Constructors (2)

```
a = HilbertMatrix [ 3 ]
```

$$\left\{\left\{1, \frac{1}{2}, \frac{1}{3}\right\}, \left\{\frac{1}{2}, \frac{1}{3}, \frac{1}{4}\right\}, \left\{\frac{1}{3}, \frac{1}{4}, \frac{1}{5}\right\}\right\}$$

HankelMatrix, ToeplitzMatrix, RotationMatrix, ScalingMatrix, ShearingMatrix, ReflectionMatrix, UnitVector, Range, RandomReal, ...

- Look them up when and if you need them!

# Importing from Matlab (1)

Let's create a MAT format file in Matlab

```
A = [ 1.0 , 2.1 , 3.2 ; 4.3 , 5.4 , 6.5 ; 7.6 , 8.7 , 9.8 ]
A =
   1.0000   2.1000   3.2000
   4.3000   5.4000   6.5000
   7.6000   8.7000   9.8000

B = [ 1.23 , 4.56 , 6.78 ]
B =
   1.2300   4.5600   6.7800

save matthew A  B
```

# Importing from Matlab (2)

```
c = Import [ "matthew.mat" ]
{ { { 1. , 2.1 , 3.2 } , { 4.3 , 5.4 , 6.5 } ,
   { 7.6 , 8.7 , 9.8 } } , { { 1.23 , 4.56 , 6.78 } } }

a = c [[ 1 ]]
{ { 1. , 2.1 , 3.2 } , { 4.3 , 5.4 , 6.5 } , { 7.6 , 8.7 , 9.8 } }

b = c [[ 2 ]] [[ 1 ]]
{ 1.23 , 4.56 , 6.78 }
```

Don't ask me why the vector becomes a matrix
It may well be a bug and fixed in next version

# Importing, Generally

There are a zillion other import/export formats
A documentation page “Listing of All Formats”

- Don’t trust import/export without testing

The Matlab example didn’t work in Mathematica 6.0

Very often a version incompatibility problem
E.g. Matlab makes an incompatible change ...
Or Mathematica assumes something that ain’t so

- Applies to ALL combinations of applications

# Importing in CSV

For your data, consider “Comma Separated Value”

```
1.0 , 2.1 , 3.2
4.3 , 5.4 , 6.5
7.6 , 8.7 , 9.8
```

```
Import [ "matthew.csv" ]
```

```
{ { 1. , 2.1 , 3.2 } , { 4.3 , 5.4 , 6.5 } , { 7.6 , 8.7 , 9.8 } }
```

This course does not cover Mathematica’s I/O

- You need that for anything more advanced

# Matrices from Expressions (1)

Table[expr,{count}] $\equiv$ ConstantArray[expr,count]
Table[expr,{var,count}] sets var to 1...count
Table[expr,{var,lwb,upb}] sets var to lwb...upb
Table[expr,{var,lwb,upb,step}] increments by step

```
Table [ 1.23 , { 3 } ]   ->   { 1.23 , 1.23 , 1.23 }
Table [ n ^ 2 , { n , 5 } ]   ->   { 1 , 4 , 9 , 16 , 25 }
Table [ n ^ 2 , { n , 3 , 5 } ]   ->   { 9 , 16 , 25 }
Table [ n ^ 2 , { n , 1 , 5 , 3 } ]   ->   { 1 , 16 }
Table [ n ^ 2 , { n , 1 , 5 , -3 } ]   ->   { }
Table [ n ^ 2 , { n , 5 , 1 , -3 } ]   ->   { 25 , 4 }
```

# Matrices from Expressions (2)

Repeated loop terms give a nested list
The last term varies fastest

```
Table [ n ^ 2 + 100 * 2 ^ m , { m , 3 } , { n , 5 } ]

{ { 201 , 204 , 209 , 216 , 225 } ,
  { 401 , 404 , 409 , 416 , 425 } ,
  { 801 , 804 , 809 , 816 , 825 } }

Table [ n ^ 2 + 100 * 2 ^ m , { m , 3 } , { n , m } ]

{ { 201 } , { 401 , 404 } , { 801 , 804 , 809 } }
```

# Matrices from Expressions (3)

```
Table [ p ^ m + q ^ n , { m , 3 } , { n , 3 } ] // TableForm

                2          3
p + q     p + q      p + q

 2          2     2    2     3
p  + q    p  + q     p  + q

 3          3     2    3     3
p  + q    p  + q     p  + q
```

There is also a related Array function
I find it more confusing and more restrictive

# Displaying Arrays

We have used TableForm several times already
It works for any number of dimensions

MatrixForm and Grid are near–synonyms
The differences are visible only in GUI mode

Row is the default display mode, as above
Column is on separate lines, by first dimension

There are also graphical display facilities
I don't mean graph–drawing ones – see ArrayPlot

# Elementwise Arithmetic (1)

The basic ‘numeric’ operations: +, −, * and ^
Normal mathematical functions: Exp, Sin etc.

Obviously, the matrix shapes must match exactly
But you can combine matrices and scalars

```
a = { { Pi , x + y ^ 2 } , { y / x , 0 } }
TableForm [ Sin [ a + Pi / 2 ] ]
                                     2
            -1              Cos [ x + y ]

                 y
            Cos [ - ]
                 x         1
```

# Elementwise Arithmetic (2)

The error handling is not nice at all

```
a = { { Pi , x + y } , { x * y , 0 } }
b = { Pi , x + y , x * y , 0 }
Sine [ a + b ]
```

Thread::tdlen: Objects of unequal length in
{ { Pi , x + y } , { x y , 0 } } + { Pi , x + y , x y , 0 } cannot be
combined.

Sine [ { { Pi , x + y } , { x y , 0 } } + { Pi , x + y , x y , 0 } ]

- So develop your program in parts, checking each

# Elementwise Arithmetic (3)

```
a = { { x ^ 2 - y ^ 2 , x ^ 3 + y ^ 3 } ,
    { x ^ 3 + y ^ 3 , x ^ 3 - y ^ 3 } }
b = { { x - y , x + y } , { x + y , x - y } }
Cancel [ a / b ] // TableForm
```

| | |
|---|---|
| $x + y$ | $x^2 - x y + y^2$ |
| $x^2 - x y + y^2$ | $x^2 + x y + y^2$ |

# True Matrix Operations

Matrix multiplication is the dot product (.)
You may prefer to use the Dot function

```
Dot [ a , b , c ]   ->   a . b . c
```

It works the same for vectors and matrices
You have to match dimensions, of course

```
a = { { 1 , 2 } , { 3 , 4 } , { 5 , 6 } }
b = { 9 , 8 }

a . b   ->   { 25 , 59 , 93 }
```

# Vector Operations

You can do most of the usual ones

```
a = Normalize [ { 1.0 , 2.0 , 3.0 } ]
{ 0.267261 , 0.534522 , 0.801784 }
Norm [ a ]
1.
```

Cross, Total, VectorAngle, Projection, KroneckerProduct, Orthogonalize, ...

Orthogonalize is simple for real and complex ONLY

# Simple Matrix Operations

You can do most of the usual ones

```
a = Transpose[{{p,q},{r,s}}]
{{p,r},{q,s}}

Tr[a]        (* Note that this is the trace *)
p + s
```

There aren't all that many more simple ones

ConjugateTranspose, KroneckerProduct, ...

# Enquiry Function

Dimensions returns the vector of dimensions

Dimensions [ { 1, 4, 7 } ] -> { 3 }

Dimensions [ { { 1, 2 } , { 3 , 4}, { 5 , 6 } } ] -> { 3 , 2}

a = Table [ n ^ 2 + 100 * 2 ^ m , { m , 3 } , { n , m } ]
{ { 201 } , { 401 , 404 } , { 801 , 804 , 809 } }

Dimensions [ a ] -> { 3 }

# Matrix Powering

You have matrix powering and exponential

```
MatrixPower [ { { 1 , x } , { y , 1 } } , 3 ]

{ { 1 + 3 x y , 2 x + x (1 + x y) } ,
  { 2 y + y (1 + x y) , 1 + 3 x y } }

MatrixExp [ { { 1.0 , 2.0 } , { 3.0 , 4.0 } } ]

{ { 51.969 , 74.7366 } , { 112.105 , 164.074 } }
```

- MatrixExp is simple for real and complex ONLY

# Numeric Linear Algebra

For now, we consider only real and complex
That is in IEEE 754 64–bit format – c.15 sig. figs

- This has some special mathematics to itself

Can do a lot more than for general matrices

- Generally, Mathematica is “automagical”

Doesn’t ask questions – just delivers the answer

You can do specific analyses if you want, though

# Matrix Inversion and Division

Division and inversion are mathematically tricky
There is an Inverse function when you need it

- DON'T invert matrices unless you have to!
Solving equations is usually the right approach

But you often need to in multivariate statistics

You can also get the PseudoInverse if wanted

# Inverse

```
Inverse [ { { 1.2 , 3.4 } , { 5.6 , 7.8 } } ]
{ { - 0.805785 , 0.35124 } , { 0.578512 , - 0.123967 } }

Inverse [ { { 1.2 - I , 3.4 } , { 5.6 , 7.8 + 2.0 * I } } ]
{ { - 0.802157 + 0.3036 I , 0.296248 - 0.208299 I } ,
      { 0.487938 - 0.343081 I ,
            - 0.0432936 + 0.160649 I } }

Inverse [ { { 1.2 , 3.4 } , { 2.4 , 6.8 } } ]
Inverse::sing: Matrix {{1.2, 3.4}, {2.4, 6.8}} is singular.

PseudoInverse [ { { 1.2 , 3.4 } , { 2.4 , 6.8 } } ]
{ { 0.0184615 , 0.0369231 } , { 0.0523077 , 0.104615 } }
```

# Determinant

Even insane requests usually work ....

```
Det [ { { 1.2 , 3.4 } , { 5.6 , 7.8 } } ]
-9.68
```

```
Det [ { { 1.2 - I , 3.4 } , { 5.6 , 7.8 + 2.0 * I } } ]
-7.68 - 5.4 I
```

```
Det [ HilbertMatrix [ 2000 ] * 1.0 ]
```

$-7.552209418370832\ 10^{-34079}$

# Enquiry Functions

HermitianMatrixQ tests for being Hermitian
And, similarly, PositiveDefiniteMatrixQ

Warning: answer is not numerically well–defined

```
PositiveDefiniteMatrixQ [ 1.0 * HilbertMatrix [ 10 ] ]

True

PositiveDefiniteMatrixQ [ 1.0 * HilbertMatrix [ 20 ] ]

False
```

# Rank And Null Space

You can calculate the rank directly
And a set of vectors spanning the null space
[ the ones corresponding to zero eigenvalues ]

- But neither is well–defined, numerically ....

```
a = HilbertMatrix [ 100 ] * 1.0 ;

{ MatrixRank [ a ] , MatrixRank [ a . a ] }
{ 18 , 10 }

Dimensions [ NullSpace [ a . a ] ]
{ 90 , 100 }
```

# Linear Equations (1)

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 }

LinearSolve [ a , b ]

{ 1.45411 , -12.4949 , 24.5078 , 11.8408 , 0.422917 }
```

# Linear Equations (2)

Complex matrices are equally easy

```
a = {{ 4.2 + 2.2 I ,  - 3.9 + 9.3 I,    0.1 + 0.0 I } ,
     { 8.6 + 0.0 I ,    0.7  - 2.3 I ,  0.0  - 0.3 I } ,
     { 8.4  - 5.9 I ,  - 8.1 + 9.6 I ,   3.8 - 0.8 I } }

b = { - 6.8 + 2.3 I , 2.7 - 7.0 I, 2.0 + 0.0 I }

LinearSolve [ a , b ]

{ 0.0361936 - 0.531091 I , 0.719502 + 0.614737 I ,
  4.02693 + 1.57063 I }
```

# Linear Equations (3)

Insoluble problems get a suitable diagnostic

LinearSolve [ { { 1.2 , 3.4 } , { 2.4 , 6.8 } } , { 1.0, 1.0 } ]

LinearSolve::nosol: Linear equation encountered that has
no solution.

LinearSolve [ HilbertMatrix [ 10 ] * 1.0 , \
RandomReal [ { −1.0 , 1.0 } , { 10 , 10 } ]

LinearSolve::luc:
Result for LinearSolve of badly conditioned matrix
{{1., 0.5, 0.333333, 0.25, <<4>>, 0.111111, 0.1}, ...
may contain significant numerical errors.

# Linear Equations (4)

Don’t rely on its diagnostics, though!

```
a = { { 1.0, 1.0 } , { 1.0, 1.0 } }

a . { 1.0 , 0.0 }
{ 1. , 1. }

a . { 0.0 , 1.0 }
{ 1. , 1. }

LinearSolve [ a , {1.0, 1.0} ]
{ 0.5 , 0.5 }
```

# Decompositions

If you are using the same matrix many times
With lots of different right hand sides
LinearSolveFunction may be faster

You can also generate decompositions directly:

LUDecomposition, CholeskyDecomposition,
SingularValueDecomposition, QRDecomposition,
SchurDecomposition, JordanDecomposition,
HessenbergDecomposition, ...

# Fourier Transforms (1)

a = { – 0.92 , 9.1 , 2.3 , 5.7 , 4.9 , –2.8 , – 5.6 ,
6.7 , –7.0 , 9.0 }

b = Fourier [ a ]
{ 6.76095 + 0. I , 3.73317 + 4.46649 I ,
– 1.44596 – 1.2133 I , 3.13213 + 1.6452 I ,
– 4.87543 – 5.03082 I , – 10.7581 + 0. I ,
– 4.87543 + 5.03082 I , 3.13213 – 1.6452 I ,
– 1.44596 + 1.2133 I , 3.73317 – 4.46649 I }

Fourier [ b ]
{ – 0.92 , 9. , –7., 6.7 , –5.6 , –2.8 , 4.9 , 5.7 , 2.3 , 9.1 }

# Fourier Transforms (2)

Mathematica doesn't call them linear algebra
Under Image Processing and Signal Processing

There is also an inverse, InverseFourier

You can generate only the cosine or sine parts
FourierDCT, FourierDST

There are also several related facilities

# Eigenvalues (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
As do all complex Hermitian ones
Not all other matrices do, though

# Eigenvalues (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}}

Eigenvalues[a]

{6.45845 + 9.89753 I, 6.45845 - 9.89753 I,
    -7.28396 + 4.45457 I, -7.28396 - 4.45457 I,
    0.351016}
```

# Eigenvalues (3)

Eigenvectors [ a ]    (* omitted as it is a bit messy *)

Eigenvalues [ HilbertMatrix [ 3 ] * 1.0 ]

{ 1.40832 , 0.122327 , 0.00268734 }

Eigenvectors [ HilbertMatrix [ 3 ] * 1.0 ]

| | | |
|---|---|---|
| – 0.827045 | – 0.459864 | – 0.323298 |
| 0.547448 | – 0.52829 | – 0.649007 |
| 0.127659 | – 0.713747 | 0.688672 |

# Eigenvalues (4)

a = { { 4.2 + 2.2 I , - 3.9 + 9.3 I , 0.1 + 8.6 I } ,
{ 0.0 + 0.7 I , - 2.3 - 0.3 I , 8.4 - 5.9 I } ,
{ - 8.1 + 9.6 I , 3.8 - 0.8 I , - 9.4 - 9.9 I } }

Eigenvalues [ a ]

{ 3.66761 - 13.7127 I , −12.3134 - 3.43015 I ,
1.1458 + 9.14282 I }

Eigenvectors [ a ] (* omitted as it is a bit messy *)

# Eigenvalues (5)

Again, don't rely on the diagnostics

```
a = { { 1.0 , 1.0 } , { 0.0 , 1.0 } }

Eigenvalues [ a ]

{ 1. , 1. }

Eigenvectors [ a ]

{ { 1. , 0. } , { 0. , 0. } }
```

That can cause chaos if you use the second one

# 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}}

CharacteristicPolynomial[a, p]
```

$$3574.06 - 9798.33\,p - 1084.54\,p^2 - 23.82\,p^3 - 1.3\,p^4 - p^5$$

# Singular Values (1)

SVD or Singular value decomposition
Essentially an extension of eigenanalysis

Gives the same results in the simple cases
I.e. square matrices with all eigenvectors

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

```
Eigenvalues [ a ]
SingularValueList [ a ]

Eigenvectors [ a ]
b = SingularValueDecomposition [ a ]
b [[ 1 ]] . b [[ 2 ]] . Transpose [ b [[ 3 ]] ]
```

# Singular Values (3)

a = HilbertMatrix [ 3 ] * 1.0

Eigenvalues [ a ]
{ 1.40832 , 0.122327 , 0.00268734 }

SingularValueList [ a ]
{ 1.40832 , 0.122327 , 0.00268734 }

Eigenvectors [ a ]

| | | |
|---|---|---|
| – 0.827045 | – 0.459864 | – 0.323298 |
| 0.547448 | – 0.52829 | – 0.649007 |
| 0.127659 | – 0.713747 | 0.688672 |

# Singular Values (4)

```
b = SingularValueDecomposition [ a ]
{ { { – 0.827045 ,   0.547448 ,   0.127659 } ,
   { – 0.459864 , – 0.52829   , – 0.713747 } ,
   { – 0.323298 , – 0.649007 ,   0.688672 } },
{ { 1.40832 , 0. , 0. } , { 0. , 0.122327 , 0. } ,
        { 0. , 0. , 0.00268734 } } ,
 { { – 0.827045 ,   0.547448 ,   0.127659 } ,
   { – 0.459864 , – 0.52829   , – 0.713747 } ,
   { – 0.323298 , – 0.649007 ,   0.688672 } } }

b [[ 1 ]] . b [[ 2 ]] . Transpose [ b [[ 3 ]] ]
{ { 1. , 0.5 , 0.333333 } , { 0.5 , 0.333333 , 0.25 } ,
   { 0.333333 , 0.25 , 0.2 } }
```

# Singular Values (5)

```
a = { { 1.0 , 1.0 } , { 0.0 , 1.0 } }

SingularValueList [ a ]
{ 1.61803 , 0.618034 }

b = SingularValueDecomposition [ a ]
{ { { − 0.850651 , −0.525731 } ,
        { − 0.525731 , 0.850651 } } ,
  { { 1.61803 , 0. } , { 0. , 0.618034 } } ,
  { { − 0.525731 , − 0.850651 } ,
        { − 0.850651 , 0.525731 } } }

b [[ 1 ]] . b [[ 2 ]] . Transpose [ b [[ 3 ]] ]
{ { 1. , 1. } , { 0. , 1. } }
```

# 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

- 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

Mathematica allows choice of precision – aha!
Should make it possible to do some easy checking

Unfortunately, it’s not easy to use ....

# Arbitrary Precision (1)

1.23‘50 is 0.123 with 50 sig. figs

N[expression,P] evaluates ‘expression’ in P sig. figs
So does SetPrecision[expression,P], but differently
Block[{$MinPrecision=P,$MaxPrecision=P},expr.]

Precision[expression] indicates the actual precision
sometimes the storage precision
and sometimes the estimated significance

- I haven’t found any precise specifications

# Arbitrary Precision (2)

- Precisions are reduced, unlike most languages
  e.g. Precision[1.23‘50*4.56‘100] –> 50

- Plain numbers (e.g. 1.23) are MachinePrecision

That is somewhere between 15 and 18 digits

I haven’t found any way of extending precision

Use arbitrary precision with great care

- And NEVER use plain real numbers

Unfortunately, almost useless for imported data

# Solution of Equations (1)

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

But, in rational arithmetic, the 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 floating−point

```
d = LinearSolve [ a + 0.0, b ]
                                                                  6
         { −9.99792  ,  989.82  ,  −23756.2  ,  240205.  ,  -
-1.26109 10 ,
                  6                    6                 6
      3.78332 10 ,  −6.72597 10 , 7.00056 10 ,
                    6
      −3.93784 10 , 923697. }
```

# Solution of Equations (3)

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

Now we do it in extended precision

```
d = N [ LinearSolve [ N [ a , 30] , b ] , 6 ]
                                                         6
     { −10.0000 , 990.000 , −23760.0 , 240240. , -
-1.26126 10 ,
                 6           6          6
   3.78378 10 , −6.72672 10 , 7.00128 10 ,
                 6
   −3.93822 10 , 923780.}
```

# 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.

Mathematica helps with non–standard analyses
- Understanding specific points in more detail

Covered in the second half of this course

Linear algebra with symbolic matrices

# Manipulating Arrays

There are a lot of facilities for manipulating arrays

Minors calculates the matrix of minors

You can reshape them using ordinary indexing
But using built–in functions is preferable

Part, Take, Drop, Diagonal, RotateLeft, RotateRight, Reverse, Join, Position, Extract, ReplacePart

# Sparse Arrays

SparseArray creates a sparse array from rules

ArrayRules creates rules from an array

Normal converts a sparse array to dense form

CoefficientArrays creates a sparse array
from a multinomial

# And More

There are some functions for optimisation

There's almost certainly stuff I haven't found

Most will be fairly specialist

If you want to work with symbolic matrices
PLEASE ask for it on your green form

It's not easy, but can be very useful