Mathematica

Symbolic Linear Algebra

Nick Maclaren

nmm1@cam.ac.uk

May 2009

Introduction (1)

This course is NOT going to be easy

- But you can get results you can't get otherwise

It is a seriously tricky computational technique

- I can't give you simple recipes, and won't try to

- It will show you how to tackle such problems

Don't worry if the details are confusing

You can go back and work through the examples
There are more than we shall go through now

Introduction (2)

Prerequisite: Mathematica – Linear Algebra etc.
OR:
You already know what that course covers

It includes nothing on the basic use of Mathematica
And little on elementary linear algebra, either

If you don't understand, please interrupt
Don't feel afraid to ask any question you want to

Yes, I have to refer to the documentation, too!

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_2.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_2.txt

Symbolic Manipulation

This is an automation of standard mathematics
You use formulae with variables, not just values

For simple manipulations, it's almost trivial, but:

- Simplifying (cancelling) formulae isn't
- Avoiding combinatorial explosion isn't

Managing those two is where the difficulties lie

But careful, step–by–step, methodology works
Check the effect of each step before proceeding

Trivial Example (1)

```
( x^3 – y^3 ) / ( x – y )    //    InputForm

( x^3 – y^3 ) / ( x – y )
```

Well, that doesn't do what we want it to . . .

```
Cancel [ ( x^3 – y^3 ) / ( x – y ) ]    //    InputForm

x^2 + x*y + y^2
```

Cancel is a built–in transformation function

Trivial Example (2)

Many built–in functions work on symbolic formulae

```
Solve [ x^3 − a + 1 == 0 , x ]   //   InputForm

{ { x −> ( − 1 + a ) ^ (1/3) } ,
  { x −> − ( (−1) ^ (1/3) * (−1 + a) ^ (1/3) ) } ,
  { x −> (−1) ^ (2/3) * (−1 + a) ^ (1/3) } }
```

Now try the following – do you know why it fails?

```
Solve [ x^5 + a * x^3 + 1 == 0 , x ]   //   InputForm
```

Simple Matrix Algebra (1)

Let's use the following matrix and vectors

$$a = \{\{4.2, 2.2+p, -3.9, 9.3, 0.1\}, \{8.6-p, r, 0.7, -2.3, -0.3\}, \{8.4, -5.9, -8.1+q, 9.6, 3.8\}, \{-0.8, -9.4, -9.9, 9.9, 5.0-r\}, \{-1.3, -8.1, 0.6, -9.2+r, -7.3\}\}$$

$$b = \{-6.8, 2.3, 2.7, -7.0, 2.0\}$$

$$c = \{p, 1.0, p+q, 1.0, q\}$$

Simple Matrix Algebra (2)

If we want to transform a known vector, it's trivial
That's just the most elementary matrix algebra

```
a . b

{ - 103.99 + 2.3 ( 2.2 + p ) ,
  17.39 - 6.8 ( 8.6 - p ) + 2.3 r ,
  - 130.29 + 2.7 ( - 8.1 + q ) ,
  - 112.21 + 2. ( 5. - r ) ,
  - 22.77 - 7. ( - 9.2 + r ) }
```

You can do exactly the same with `a . c`

Perturbation Analysis

- Error analysis of a numerical solution

Conventional error analysis is in terms of norms
This is a well–understood area, with useful results

- Use the formulae for it in books etc.

Mathematica helps with non–standard analyses
How specific variations affect the results

Useful for understanding specific points in detail
This is easier to show than to explain

Low-Order Approximations

Consider small variations around a solution
Ignore (say) all second order terms and above

See how solution will vary with perturbations
Can be used to answer questions like:

- Is parameter A critical to the result?
- What's the best way to change the results?
- Does it matter if parameters A and B are linked?

Alternative Approaches

There are a lot of existing results on this
E.g. sensitivity analysis in statistics

Where those answer your question, they are better
Use this technique when they don't help

The specific examples I use are fairly simple
There are other ways, perhaps better ones

An example at the end where only this works

Jumping Ahead

For example: c . MatrixExp[a] . c to second order

$$-1486.11 - 3585.23\,p + 2150.23\,p^2 + 3953.26\,q + 12339.7\,p\,q + 1800.52\,q^2 - 3154.22\,r - 6192.86\,p\,r + 408.955\,q\,r - 410.955\,r^2$$

Getting there isn't hard, but needs background

We shall work up to it slowly, explaining why and how

How Mathematica Works

Mathematica's operation is very simple (even dumb)
You need to know how it works for symbolic work

Everything is an expression, stored as a tree

```
FullForm [ 1.23 * x + y + 4.56 * z ]

Plus [ Times [ 1.23‘ , x ] , y , Times [ 4.56‘ , z ] ]
```

Everything works by transforming that tree

Assignment (1)

It expands the RHS recursively
This is more–or–less literal expansion

It replaces variables with their values
Any unset names are left in symbolic form

Remember that values are always expressions

```
x = 1.23 * a + 4.56
y = 7.89 * x + 0.12 + z
```

```
0.12 + 7.89 ( 4.56 + 1.23 a ) + z
```

Assignment (2)

You can use the LHS variable in the RHS
Provided it already has a value

```
x = 1.23 * a + 4.56
x = 1.23 * x + 4.56
```

```
4.56 + 1.23 ( 4.56 + 1.23 a )
```

But this goes into an infinite loop

```
x =.        (* Ensure that x has no value *)
x = 1.23 * x + 4.56
```

How To Stay Sane

Keep the following two as separate as possible:

- Symbols you use for physical quantities

For example, e, m and c in e = m c^2

- Symbols you use for Mathematica variables

I.e. ones used for programming language variables

Unfortunately, often need to use one as the other
It is very easy to get confused doing that
Minimise and localise such uses

Transformations (1)

There are a great many built–in ones
Look under "Formula Manipulation" for more

There is an "easy to use" function Simplify
Not used here, as it doesn't say what it does

There is almost always something that helps
You search the documentation, and then experiment

If not, you can manipulate the tree yourself
This course does not cover such advanced use

Transformations (2)

Shall use only the few that we actually need
These are the ones used in the examples:

Cancel [expr]: cancels out common factors in the numerator and denominator of expr

CoefficientList [poly , var]: gives a list of coefficients of powers of var in poly, starting with power 0

Collect [expr , var]: collects together terms involving the same powers of objects matching var

Transformations (3)

Expand [expr]: expands out products and positive integer powers in expr

ExpandAll [expr]: expands out all products and integer powers in any part of expr

Together [expr]: puts terms in a sum over a common denominator, and cancels factors in the result

Rules (1)

Rules are user−defined and explicit transformations

Set all powers of x beyond the third to zero

```
rule = x ^ k_ /; k > 3 −> 0
Expand [ ( 1 + x ) ^ 7 ] /. rule
```

$$1 + 7\,x + 21\,x^2$$

Rules (2)

Now, how does that work? Let’s go through it slowly

```
rule = x ^ k_ /; k > 3 -> 0
```

“x ^ k_” is a pattern that matches powers of x

“k” (no underscore) is set to the actual power

“/; k > 3” is a condition that says k > 3

“-> 0” says replace the pattern by 0

Rules (3)

```
Expand [ ( 1 + x ) ^ 7 ] /. rule
```

Expand [(1 + x) ^ 7] is an arbitrary expression

```
                   2        3         4
1 + 7 x + 21 x  + 35 x  + 35 x + . . .
```

"/. rule" applies the rule to that expression

```
                   2
1 + 7 x + 21 x
```

Remember The Tree?

Everything works on the expression tree
If that isn't in the right form, they don't work

```
x * y + x * z   /. ( y + z ) -> 0

x y + x z
```

Try one rule at once; if it doesn't work, either:

- Change the rule to work better
- Transform the tree so it does

Delayed Rules

The RHS is evaluated during the definition
That’s not right if it includes a function call

There’s a variant syntax for a delayed rule
It just replaces the “->” by “:>”

```
x * y + x * z   /. ( y + z ) :> 0

x y + x z
```

In this case, it makes no difference, of course

Repeated Rules

There's a variant syntax for a repeated rule
It applies it repeatedly until there is no change

Just replace the "/." by "//."

That's all we shall use of rules and patterns
Mathematica has quite a lot more features

Worked Example (1)

Let’s use the previous matrix and vectors
And calculate c . a . a . a . c to first order

For simplicity, use a single infinitesimal (say ‘e’)
Other values transformed to ‘constant × e’

Then define a rule to kill high powers of e

```
x = c . MatrixPower [ a , 3 ] . c ;
p = p1 * e ; q = q1 * e ; r = r1 * e ;
rule = e ^ k_ /; k > 1 -> 0
y = x /. rule
```

Worked Example (2)

Well, that doesn't do what we want it to . . .
We need to expand the powered expressions

```
p = p1 * e ; q = q1 * e ; r = r1 * e ;
rule = e ^ k_ /; k > 1 -> 0
y = Expand [ x ] /. rule
```

– 1125.78 – 2593.47 e p1 – 1261.99 e q1 + 67.65 e r1

Hey, presto!

Worked Example (3)

We can now restore the original variables
Use some rules to convert ‘p1 e’ to ‘p’ etc.

Remember to unset the effect of ‘p’ etc. first!

```
p =. ; q =. ; r =. ;
z = y /. e p1 -> p /. e q1 -> q /. e r1 -> r
```

$-1125.78 - 2593.47\, p - 1261.99\, q + 67.65\, r$

Worked Example (4)

Let's try second order – it's much the same

We need a different approach to restoration

```
p = p1 * e ; q = q1 * e ; r = r1 * e ;
rule = e ^ k_ /; k > 2 -> 0
y = Expand [ x ] /. rule
p =. ; q =. ; r =. ;
p1 = p / e ; q1 = q / e ; r1 = r / e ;
z = y
p1 =. ; q1 =. ; r1 =. ;
```

Worked Example (5)

$$-1125.78 - 2593.47\, p - 2628.74\, p^2 - 1261.99\, q$$

$$+ 9.05\, p\, q + 191.281\, q^2 + 67.65\, r$$

$$+ 120.61\, p\, r - 109.2\, q\, r - 4.7\, r^2$$

Aside: Syntactic Gotcha

You separate statements by semicolons (“;”)
Usually, omitting them before a newline prints

```
arg = { 1 , 2 , 3 }
{ 1 , 2 , 3 }
```

But not always (especially in function definitions) . . .

```
arg = { 1 , 2 , 3 }
fred [ ] := ( x = arg [[ 1 ]]
y = x )
```

Syntax::newl: The newline character after "fred[] := (x = arg[[1]]"
is understood as a multiplication operator.

Something Harder

What about c . MatrixExp[a] . c to first order?
That's truly horrible (and useless to us) . . .

Not surprising, given our simplistic approach
MatrixExp[a] doesn't have a finite expansion

But matrix exponentiation is just like numeric

$$exponent(A) = \sum_{k \geq 0} A^k / k!$$

What Can We Do?

Solution is to write our own MatrixExp
Which clobbers second order terms at each stage

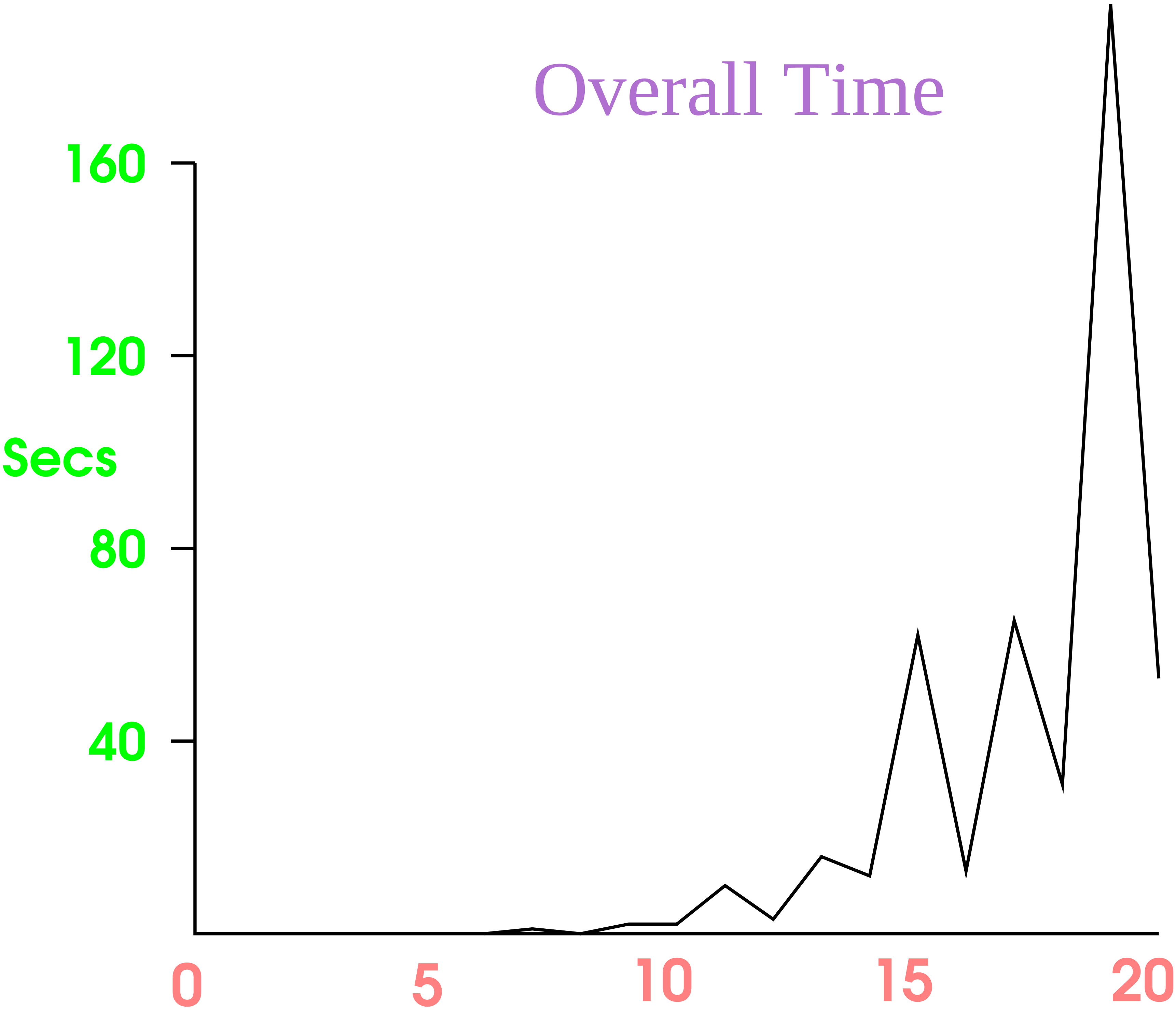
This is a general technique in this area
Reduce was the same, 30 years ago

Mathematica is less automatic than Reduce
The handouts explain why we can't use MatrixPower

Using MatrixPower

Let's time $A^k/k!$ and its simplification first

```
For [ k = 0 , k < 21, ++k , Print [ Timing [
      x = c . MatrixPower [ a , k ] . c ;
      p = p1 * e ; q = q1 * e ; r = r1 * e ;
      rule = e ^ k_ /; k > 1 −> 0 ;
      y = Expand [ x ] /. rule ;
      p =. ; q =. ; r =. ;
      p1 = p / e ; q1 = q / e ; r1 = r / e ;
      z = y ;
      p1 =. ; q1 =. ; r1 =. ;
      z
] ] ]
```


Overall Time
160
120
Secs
80
40
0
5
10
15
20

Back To Basics (1)

How would we do it in Fortran and floating–point?

```
PURE FUNCTION exponent ( arg )
    . . .
    exponent = 1.0 ; value = 1.0
    DO k = 1 , HUGE ( k )
        value = value * arg / k
        temp = exponent + value
        IF ( temp == exponent ) EXIT
    END DO
END FUNCTION exponent
```

Back To Basics (2)

And this is the Mathematica equivalent:

```
matexp [ z_ , r_ ] := Module [ { k, w, x, y } ,
        x = z ;
        y = IdentityMatrix [ Dimensions [ z ] [[ 1 ]] ] ;
        For [ k = 1 , k < 50 , ++k ,
                x = Expand [ x . z / k ] /. r ;
                w = y + x ;    (* No rule needed *)
                y = If [ SameQ [ y, w ] , Break [ ] , w ]
        ] ;
        y
]
```

We will come back to this in a moment

Timing That

```
Timing [
        p = p1 * e ; q = q1 * e ; r = r1 * e ;
        a1 = a ;    rule = e ^ k_ /; k > 1 -> 0 ;
        x = c . matexp [ a1, rule ] . c ;
        y = Expand [ x ] /. rule ;
        p =. ; q =. ; r =. ;
        p1 = p / e ; q1 = q / e ; r1 = r / e ;
        z = y ;    p1 =. ; q1 =. ; r1 =. ;    z
]

{ 0.304019 , -1486.11 - 3585.23 p + 3953.26 q - 3154.22 r }
```

Under 1/3 of a second – that’s not too not bad
Now let’s go back and work through the function

Second order (1)

```
Timing [
     p = p1 * e ; q = q1 * e ; r = r1 * e ;
     a1 = a ;
     rule = e ^ k_ /; k > 2 -> 0 ;          (* Changed *)
     x = c . matexp [ a1, rule ] . c ;
     y = Expand [ x ] /. rule ;
     p =. ; q =. ; r =. ;
     p1 = p / e ; q1 = q / e ; r1 = r / e ;
     z = y ;
     p1 =. ; q1 =. ; r1 =. ;
     z
]
```

Second order (2)

$$\{0.820051, -1486.11 - 3585.23\,p + 2150.23\,p^2 + 3953.26\,q + 12339.7\,p\,q + 1800.52\,q^2 - 3154.22\,r - 6192.86\,p\,r + 408.955\,q\,r - 410.955\,r^2\}$$

Which still takes under a second!

MatrixPower

Here is how you would do MatrixPower, incidentally
It takes well under 0.1 seconds in each case

```
matpower [ z_ , n_ , r_ ] := Module [ { k, x, y } ,
        k = n ; x = z ;
        y = IdentityMatrix [ Dimensions [ z ] [[ 1 ]] ] ;
        While [ k > 0 ,
                y = If [ OddQ [ k ] ,
                        Expand [ y . x ] /. r , y ] ;
                k = IntegerPart [ k / 2 ] ;
                x = Expand [ x . x ] /. r
        ] ;
        y
]
```

Fourier Transforms (1)

Fourier transforms are just matrix multiplication
With matrices of a special form, of course

```
a = { 0.906 , 4.528 , 4.693 , 1.515 + p , −2.190 , −3.239 ,
   −0.724 , 3.238 , 5.253 − 2 * p , 3.463 , −0.906 , −4.528 ,
   −4.693 , −1.515 + p , 2.190 , 3.239 , 0.724 , −3.238 ,
   −5.253 − 2 * p , −3.463 }
Fourier [ a ]
```

```
Fourier::fftl: Argument { 0.906, 4.528, <<17>>, −3.463}
  is not a nonempty list or rectangular array of
     numeric quantities.
```

Boring – so we have to do it by hand

Fourier Transforms (2)

See http://en.wikipedia.org/wiki/...
...`/Discrete_Fourier_transform/`

```
n = 20
w = Cos [ 2.0 * Pi / n ] + I * Sin [ 2.0 * Pi / n ] ;
t = Table [ w ^ ( i * j ) , { i , 0 , n-1 } , { j , 0 , n-1 } ] /
    Sqrt [ 1.0 * n ] ;
Collect [ t . a , p ]
```

Tedious, but not exactly difficult
Now, why can't Mathematica do that?

Solving Equations (1)

Solving linear equations is a little trickier
We use the matrix and vector we had earlier

```
a = { {  4.2 ,     2.2 + p , -3.9 ,      9.3 ,       0.1     } ,
      {  8.6 - p , r ,         0.7 ,    -2.3 ,      -0.3     } ,
      {  8.4 ,    -5.9 ,      -8.1 + q , 9.6 ,       3.8     } ,
      { -0.8 ,    -9.4 ,      -9.9 ,     9.9 ,       5.0 - r } ,
      { -1.3 ,    -8.1 ,       0.6 ,    -9.2 + r ,  -7.3     } }

c = { p , 1.0 , p + q , 1.0 , q }

v = LinearSolve [ a , c ]      (* That's not pretty *)
```

Solving Equations (2)

So let's calculate a first order solution
Obviously need to expand powers top and bottom
Hence use ExpandAll rather than Expand

```
p = p1 * e ; q = q1 * e ; r = r1 * e ;
rule1 = e ^ k_ /; k > 1 -> 0
w =  ExpandAll [ v ] /. rule1
```

Unfortunately, ExpandAll doesn't seem to work
There are products of expressions still not expanded

What do we do? We try a slightly different approach

Solving Equations (3)

Let's convert it to a common denominator first

```
w = ExpandAll [ Together [ v ] ] /. rule1
```

That's a lot better, and worth working from
So we shall now work from that expansion, w

This is a very important general technique
Try bypassing problems by using a variant approach
If it works, it works – don't worry too much why

Solving Equations (4)

We need to convert a/(b+c*e) to (a/b)*(1−(c/b)*e)

```
rule2 = k1_ / ( k2_ + k3_ * e ) ->
        k1 * ( 1 - ( k3 / k2 ) * e ) / k2
x = w /. rule2
```

Well, that improves things, but doesn't get there
We need to apply the rule repeatedly until no change

```
x = w //. rule2
```

Hah! We are in sight of victory! Use x from now

Solving Equations (5)

We just need to expand again and apply rule1

```
y = Expand [ x ] /. rule1
```

And then convert back to our original variables

```
p =. ; q =. ; r =. ;
z = y /. e p1 -> p /. e q1 -> q /. e r1 -> r
```

Solving Equations (6)

Which gives us the final result:

$$-0.137433 + 0.211331\,p + 0.851368\,q + 0.0507513\,r$$

$$2.19542 - 2.31209\,p - 7.83925\,q - 2.67286\,r$$

$$-4.41552 + 4.43115\,p + 15.082\,q + 5.05721\,r$$

$$-2.31043 + 2.17963\,p + 7.79656\,q + 2.73433\,r$$

$$0.137312 + 0.145104\,p - 0.176435\,q - 0.390099\,r$$

Solving Equations (7)

So our complete program becomes:

```
v = LinearSolve [ a , c ] ;
p = p1 * e ; q = q1 * e ; r = r1 * e ;
rule1 = e ^ k_ /; k > 1 -> 0 ;
w = ExpandAll [ Together [ v ] ] /. rule1 ;
rule2 = k1_ / ( k2_ + k3_ * e ) ->
        k1 * ( 1 - ( k3 / k2 ) * e ) / k2 ;
x = w //. rule2 ;
y = Expand [ x ] /. rule1 ;
p =. ; q =. ; r =. ;
z = y /. e p1 -> p /. e q1 -> q /. e r1 -> r
```

Solving Equations (8)

Check that we got it right by reverse substitution:

```
p = p1 * e ; q = q1 * e ; r = r1 * e ;
s = Expand [ a . z ] /. rule1 ;
p =. ; q =. ; r =. ;
t = s /. e p1 -> p /. e q1 -> q /. e r1 -> r
```

I.e. c with rounding errors – also try this:

```
t /. x_ /; Abs[x] < 1.0*^-12 -> 0
```

Second Order (1)

Let's try just adding a rule for a/(b+c*x+d*x^2)

We need to collect terms in x to do this

```
v = LinearSolve [ a , c ] ;
p = p1 * e ; q = q1 * e ; r = r1 * e ;
rule1 = e ^ k_ /; k > 2 -> 0 ;
w = ExpandAll [ Together [ v ] ] /. rule1 ;
rule2 = k1_ / ( k2_ + k3_ * e ) ->
      k1 * ( 1 - ( k3 / k2 ) * e ) / k2 ;
rule3 = k1_ / ( k2_ + k3_ * ( e | e ^ * k4_ ) ->
      k1 * ( 1 - ( k3 / k2 ) * e -
            ( ( k3 / k2 ) ^ 2 ) * e ^ 2 -
            ( k4 / k2 ) * e ^ 2 ) / k2 ;
x = Collect [ w , e ] //. rule2 //. rule3
```

Second Order (2)

Sigh. Collect applies only at top level

We need to go back a step and write a function
We start with a/(b+c*x+d*x^2)

And we turn it into (a/b)*(1−(c/b)*x+((b/c)^2−d)*x^2)

```
invert [ arg_ ] := Module [ { x, y , z } ,
        x = CoefficientList [ arg , e ] ;
        y = x [[ 2 ]] / x [[ 1 ]] ; z = x [[ 3 ]] / x [[ 1 ]] ;
        (1.0 / x [[ 1 ]] ) *
                ( 1.0 – y * e + ( y ^ 2 – z ) * e ^ 2 )
]
```

Second Order (3)

Now use that – note that we need a delayed rule
Start from after the w = ExpandAll line

Tidy up by restoring the original variables

```
rule2 = k1_ / k2_ :> k1 * invert [ k2 ] ;
x = Expand [ w //. rule2 ] /. rule1

p =. ; q =. ; r =. ;
p1 = p / e ; q1 = q / e ; r1 = r / e ;
y = x ;
p1 =. ; q1 =. ; r1 =. ;
y
```

Second Order (4)

Check that we got it right by reverse substitution:

```
p = p1 * e ; q = q1 * e ; r = r1 * e ;
s = Expand [ a . y ] /. rule1 ;
p =. ; q =. ; r =. ;
p1 = p / e ; q1 = q / e ; r1 = r / e ;
t = s ;
p1 =. ; q1 =. ; r1 =. ;
Print [ t ] ;
t /. x_ /; Abs[x] < 1.0*^-12 -> 0
```

Which is c with some rounding errors

Practicalities

How did I work that out? Just as I showed you

- Lots of trial and error, step–by–step

As we saw, not everything works as specified
The documentation makes it unnecessarily hard

Nothing is specified precisely or completely
There is no comprehensive index to a topic
The tutorials are almost always too simple

- But that can be handled, with care

And I haven't found anything not documented

User's Guide (1)

- Firstly, work out the next step
Make sure it's really, really simple

- Then look in the documentation
Remember that there is a search facility

- Look into sections with more information
Check all the other useful–looking functions

- Don't forget to look in the tutorials
But check the description of what you find

User's Guide (2)

- Run an experiment on each new feature
Check that you understand the description

- Then try it for real, and see if it helps
Check on what it has done (if anything)
Then tweak it, if you can see something to do

- If it works, good – if not, iterate!
Go back and check its description and tutorials
Or go right back and look for another feature

Yes, it's tedious

User's Guide (3)

But not as tedious as doing it by hand!

Try it if you don't believe me . . .

Performance (1)

The above is OK for for small, simple matrices
But the time can build exponentially

```
For [ k = 1 , k < 10, ++ k ,
        a = Table [ RandomReal [ ] , { k } , { k } ] +
                Table [ RandomReal [ ] , { k } , { k } ] * p ;
        b = Table [ RandomReal [ ] , { k } , { k } ] ;
        Print [ k , Timing [ LinearSolve [ a , b ] ; ] ]
]
```

Ouch. We need to take a different approach
Start by describing the general technique

Performance (2)

As with MatrixExp, you go back to basics
Gaussian Elimination solves such equations
Lots of Web pages that describe it, including:

http://mathworld.wolfram.com/...
.../GaussianElimination.html

Test using floating–point, because it's much easier

You can use Mathematica to generate test results
In this case, compare against LinearSolve

Performance (3)

You have a working Gaussian Elimination solver
Now start converting it for symbolic work

Create a suitable symbolic test case
Add simplification rules, step–by–step

For tricky problems, save values using InputForm
Then you can work on just that, in a separate program

Then use the answer in your main code
And carry on working on that . . .

Gaussian Elimination (1)

There are some programs that do the first step
There's nothing tricky about this, just tedious

NOT how best to code Gaussian Elimination
E.g. we shall omit all pivoting, for simplicity

All files are in directory Examples_2

File gaussian_1.math uses LinearSolve
File gaussian_2.f90 is Fortran 90 code
File gaussian_3.math is Mathematica code

Gaussian Elimination (2)

You have learnt enough to add the simplifications
This is left as an exercise, with an answer provided

$\Leftarrow$ You should start with file gaussian_3.math

$\Rightarrow$ File gaussian_4.math is the specimen answer

File worked_1.math uses LinearSolve
Just the example worked through above, for reference

Gaussian Elimination (3)

There are also files containing second order solvers

File worked_2.math uses LinearSolve

Just the example worked through above, for reference

Function invert is a bit more complicated

To handle expressions that are not quadratic

$\Leftarrow$ You should start with file gaussian_4.math

File gaussian_5.math uses Gaussian Elimination

$\Rightarrow$ This is the result you should be aiming for

Eigenvalues etc. (1)

Harder than linear equation solving – why?

Numerical ones are fairly well–understood
But applies to real ($\mathbb{R}$) and complex ($\mathbb{C}$) ONLY

They are solutions of the characteristic polynomial

CharacteristicPolynomial [HankelMatrix [5] , x]

$$3125 - 686\,x - 398\,x^2 + 72\,x^3 + 9\,x^4 - x^5$$

Eigenvalues etc. (2)

You can solve a quartic in radicals (roots)
Galois proved you can't solve a general quintic

Eigenvalues of symbolic matrices need a solution
So you can't get them, in general, above 4×4

```
Solve [ CharacteristicPolynomial [
        HankelMatrix [ 4 ] , x ] == 0 , x ]

Solve [ CharacteristicPolynomial [
        HankelMatrix [ 5 ] , x ] == 0 , x ]
```

Eigenvalues etc. (3)

But we know that we can get them, numerically!

You would adapt one of the iterative solvers
And would start from the numerical solution

Symbolic calculations are very like that
Few things are impossible, but many are hard

And you almost always have to go back to basics

Where I Came In (1)

My initial interest in this area was statistics
I had some complicated combinatoric distributions
Think hypergeometric, only a bit more complicated

$$probability = \frac{\binom{K}{k}\binom{N-K}{n-k}}{\binom{N}{K}}$$

That is asymptotically normal under many conditions
I wanted the first 3–4 terms in the approximation

Where I Came In (2)

You expand factorials using Stirling's formula

```
factorial [ n_ ] = Sqrt [ 2 * pi * n ] * ( n / e ) ^ n *
        (1 + 1 / ( 12 * n ) + 1 / ( 288 * n ^ 2 ) ) ;
choose [ n_ , k_ ] :=
        factorial [ k ] * factorial [ n - k ] / factorial [ n ] ;
choose [ K , k ] * choose [ N - K , n - k ] /
        choose [ N , K ]
```

That's bad enough, and my problem was worse!
After 2 terms using pencil and paper, I gave up

Where I Came In (3)

Reduce was painful – but better than by hand!
The key to solving my problem was twofold:

- Deal with highest rank terms first

N^N > a^N > N^a etc.

- Cancel the highest rank terms

Then reduce the residue to the next lower rank

I can no longer remember the details!
But please contact me if you have this problem