Why Getting Numerical Analysis Right Matters:

Or a Layman's View of Economic Disaster

Definitions

- CDO: Collateralized Debt Obligation - A mixed up collection of mortgages or other debts (designed to reduce risk).
- Tranche: A way of dividing up a CDO to make pieces less (senior) or more (mezzanine) risky.
- CDO^2: A recombination of mezzanine CDO's into new senior and mezzanine tranches.
- CDS: Credit Default Swap - A weird sort of life insurance for bonds and mortgages (designed to reduce risk).
- Copula: The covariance of default risk.

The CDO Tranches

- Major investors such as pension funds can only invest in AAA bonds, as they are risk averse.
- Subprime mortgages are intrinsically risky (that's why they are subprime!).
- To make an investment grade bond, a bunch of subprime mortgages are combined into a pool, diversifying the risk.
- This pool is then subdivided into *tranches*. Any defaults are assigned to the lowest tranche first (potentially wiping it out). The senior tranches are the last to get defaults.
- The bottom (mezzanine) tranches were then pooled together, and then redivided into tranches again, producing the CDO^2 financial product.
- The top tranches were sold off to banks and pension funds, creating a huge market for subprime loans.

The CDS

- Banks like to control risk by insuring against loss, which is an important way of protecting the financial system.
- CDO's were insured via something called a *credit default swap*.
- Insurance was sold that would pay out if the CDO defaulted, and the insurance company got a premium for their risk.
- Rather than just selling the CDS to the bond owner (which makes sense), AIG and other insurers sold CDS's to anyone, including hedge funds which recognized that the insurers were underestimating the risk.
- This is sort of like taking out a life insurance policy on your neighbor, without even letting him know…
- The total exposure on CDS's was actually far greater than the total CDO market.
- Goldman and deal makers made fees on every transaction.

Risk Estimation

- In order to price the insurance and rate the risk of the CDO's, it was necessary to determine the probability of default.
- Because they were pooled investments, the default rate was critically dependent on the covariance of the subprime mortgages.
- Rather than examining the bonds making up the CDO, David Li proposed that the market price of the related CDS could be used to determine the *copula*.
- This worked great, except all the available data was during a period of rising home prices - when the covariance was naturally small.
- When the housing market started to go down, really bad things happened…

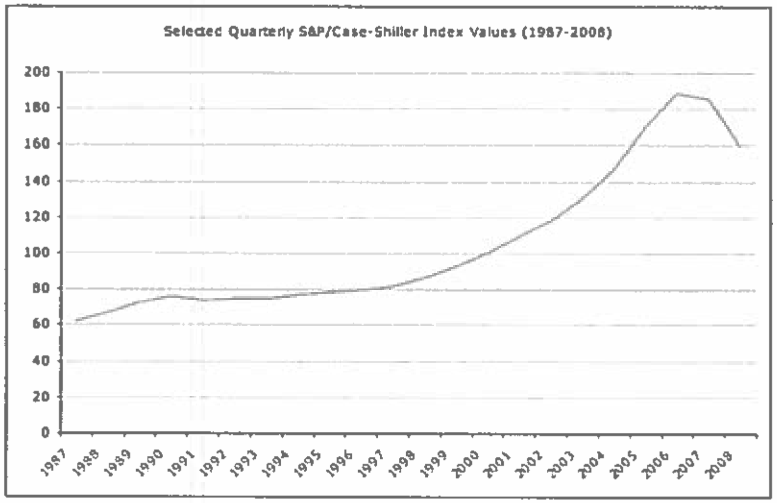
The Effect of Greed!

- Wall Street was rewarded via fees for brokering deals.
- Because a market was created for sub-prime (read "bad") loans, lenders solicited wildly inappropriate borrowing and passed the bad loans off to others. This increased the total amount of borrowing far beyond a sustainable level.
- Insurers would issue CDS guarantees in an amount far greater than the CDO being insured: again, they would get lots of fees.
- Provided that housing prices kept going up, the banks and the insurers thought they had a great deal!

Why the error?

- CDO correlation used CDS valuation, only available during a period of rising home prices!
- Covariance is massively underestimated in an adverse housing market.
- Reality was too painful to believe, and easier to ignore.
- Senior management used the “black box” calculations and didn’t really understand the implications of the underlying assumptions.

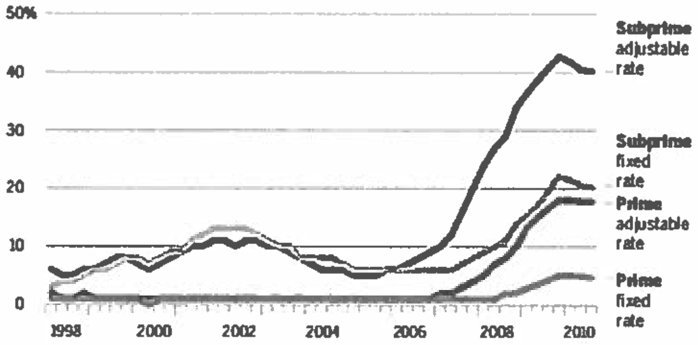
Housing Price Variation



Mortgage Delinquencies by Loan Type

Serious delinquencies started earlier and were substantially higher among subprime adjustable-rate loans, compared with other loan types.

IN PERCENT, BY TYPE



NOTE: Serious delinquencies include mortgages 90 days or more past due and those in foreclosure.
SOURCE: Mortgage Bankers Association National Delinquency Survey

Leverage: Compounding the error

- Leverage means borrowing money to invest in a higher yielding opportunity
- Leverage can be very good, or very very bad.
- You have to accurately asses the risk of the investments
- Financial institutions underestimated the risk of CDO's, so what they *thought* was a sure thing, was really a sure disaster
- At the end, ML was leveraged 30:1 in CDO's, with predictable results…

A Timeline

- 1997 housing price index = 80
- 1997 CDS invented by JP Morgan Chase
- 2000 Paper by David Li “On Default Correlation: A Copula Function Approach” - determined default correlation from CDS pricing
- 2000 “Commodity Futures Modernization Act of 2000”- A “late night” bill deregulating credit default swaps
- 2001 $920 billion in CDS outstanding, $275 billion in CDO’s
- 2006 peak housing price index of 190 (2.4x in 9 years!)
- 2006 CDO market hits $4.7 trillion (17x increase in 5 years!)
- 2007 CDS market hits $62 trillion (67x increase in 6 years!)
- 2008 Lehman Brothers declares bankruptcy
- 2009 Troubled Asset Relief Program costs the taxpayer nearly $1T

The Take Home Message:

- Numerical & Statistical Analysis can be very useful to understand complex systems!

- You must -always- question the assumptions that go into any numerical model, particularly one involving evaluation of risk/uncertainty.

- Most systems and models will 1) assume some level of independence, and 2) underestimate the degree of covariance.

- DON'T BELIEVE THE ANSWER JUST BECAUSE THE COMPUTER GIVES YOU A NICE GRAPH OR NUMBER! UNDERSTAND WHAT YOU ARE DOING!

Further Reading

- Michael Lewis, “The Big Short”, 2010.
- Felix Salmon, “Recipe for Disaster: The Formula That Killed Wall Street” Wired.com 2009.
- Anna Barnett-Hart, “The Story of the CDO Market Meltdown: An Empirical Analysis” Senior Thesis, Harvard University 2009.

Review of Matrix Ops:

A matrix is a way of representing a large amount of information in a compact way.

Matlab ≡ Matrix Laboratory

⇒ it's all about manipulating matrices!

Suppose we look at the HW grades for a class. Let:

G_{ij} ≡ grade of the i^{th} student on the j^{th} homework

So the matrix $\underset{\sim}{G}$ might be:

$$\underset{\sim}{G} = \left[\begin{array}{cccc} 9 & 8 & 7 & 8 \\ 10 & 6 & 9 & 10 \\ 2 & 3 & 1 & 0 \\ \uparrow & \uparrow & & \end{array}\right] \begin{array}{l} \leftarrow 1^{st}\text{ student} \\ \leftarrow 2^{nd}\text{ student} \\ \\ \\ \end{array}$$

2^{nd} assignment

1^{st} assignment

Suppose we want to calc. the <u>average</u> for each student.

Let $\underset{\approx}{G}$ be an $n \times m$ matrix (n rows, m columns)

Then the <u>vector</u> $\underset{\sim}{AG}$ is:

$$\underset{\sim}{AG} = \underset{\approx}{G} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \frac{1}{m} = n \times 1 \text{ vector!}$$

$n \times m$ matrix $\longleftrightarrow$ $m \times 1$ vector

these <u>must</u> match!

We multiply each <u>row</u> of <u>first</u> matrix by each <u>column</u> of <u>2nd</u> matrix (only one here)

So $$\underset{\approx}{A}\,\underset{\approx}{B} = \sum_j A_{ij} B_{jk} = C_{ik}$$

which is a series of nested loops!

In Matlab, this is a lot more compact!

$$\underset{\sim}{AG} = \underset{\approx}{G} * \text{ones}(4,1) * \frac{1}{4}$$

$$\underset{\sim}{AG} = \text{sum}(\underset{\approx}{G}')' * \frac{1}{4}$$

(sum command sums over columns! we use ' to take transpose!)

$$\underset{\sim}{AG} = \text{sum}(\underset{\approx}{G}, \underset{\uparrow}{2}) \frac{1}{4}$$

forces sum over rows!

All yield the same result!

Final note: Matlab scripts & functions

Run exanple 1

- Break your code into <u>functions</u> and <u>scripts</u>
- comment your code so you can remember it!
- define function I/O & purpose in comment on function (1^{st} line)

How do computers represent numbers?

⇒ stored in a finite amount of memory!

A computer can't directly store a base 10 number – each cell is either on or off

∴ Binary!

0 = 0
1 = 1
2 = 10
3 = 11
11 = 1011
etc...

each binary digit is a bit

8 bits = 1 byte

single precision = 32 bits (4 bytes)

For integers you can store them exactly in a finite range!

$$-2^{31} < N < 2^{31} - 1$$

(one bit used for sign)

2^{31} isn't very big!

$2^{31} = 2,147,483,648$

(less than the amount the gov't spends in one day...)

Double precision (default for Matlab) doubles the # of bits to 64

To store larger or non-integer #'s, you use floating point scientific notation!

In single precision, use 24 bits mantissa, 8 bits for exponent

one of each needed for signs!

So the largest single precision

$\# = 1.111\ldots \times 2^{+127} \approx 10^{38}$

(base 2)

Any thing larger leads to overflow!

The 24 bit mantissa means that you only get ~7 sig digits (base 10).

For double precision, you do much better - about 16 sig digits, and a max # of $\sim 10^{308}$

This seems great -
10^{308} is huge (as a ref, there are $\sim 10^{50}$ atoms in earth)

You can still have problems!

In probability formulas you often need to use factorials

(to determine # possible permutations)

$171! \sim 10^{309} = \text{Inf}$

not a big # !

The smallest number in matlab is $\sim 10^{-308}$

Anything smaller is taken as zero!

How do we avoid underflow & overflow errors?

⇒ well-designed algorithms!

Example: Probability!

Suppose the probability of some outcome occuring is p

(coin flip, heads prob is $p = 0.5$)

Now suppose you do this N times. What is the probability of this occuring exactly K times?

$$X(p, K, N) = \binom{N}{K} p^K (1-p)^{N-K}$$

The quantity $\binom{N}{K}$ is the binomial coefficient:

$$\binom{N}{K} = \frac{N!}{K!(N-K)!} = \frac{N(N-1)\cdots(N-K+1)}{K(K-1)\cdots 2\cdot 1}$$

A natural way to do this is to use the Matlab command factorial.m

You could also iterate over K:

$$\binom{N}{K} = \frac{\lambda_2}{\lambda_1}; \quad \lambda_2 = \prod_{i=N-K+1}^{N} i$$

$$\lambda_1 = \prod_{i=1}^{K} i$$

If N is larger than 170 you are toast!

In single precision N has to be much smaller!

Take coin flips - flip it 50 times, what is the prob it comes up heads exactly 25 times?

So: $N = 50$, $K = 25$, $p = 0.5$

$$X = \frac{50!}{25!\,(50-25)!}\,(0.5)^{25}\,(0.5)^{50-25}$$

If we calc. λ_1 & λ_2 we have trouble in single precision:

$$\lambda_2 = \prod_{i=k+1}^{N} i = \frac{N!}{k!} = 1.96 \times 10^{39}$$

(larger than max in single)

But the answer is just

$$X = 0.112 \quad \text{or} \sim 11\% \,!$$

How can we avoid the error? Simple! As we go through the calculation, we just add lines of code that check when the # is too large or small, then mult or divide by a large #. If we

Keep track of the # of times we do it, we can recover the correct result at the end!

(Run example)

So much for the exponent, what about the mantissa? $\Rightarrow$ finite precision! (roundoff error!)

Single Precision $\sim 10^{-7}$

Double Precision $\sim 10^{-16}$

These are small: single ~ 13ft error in distance to moon

Double ~ 0.1 μm

But you can still have trouble!

Most common: Algorithms which subtract large #s to get small ones!

Example: Suppose we want to calc $f(x) = e^{-x}$

We can use the *Taylor Series*

$$f(x) \approx f(x_0) + f'(x_0)(x-x_0) + \frac{1}{2} f''(x_0)(x-x_0)^2 + \frac{1}{3!} f'''(x_0)(x-x_0)^3 + \ldots$$

For e^{-x} if we take $x_0 = 0$ this is just:

$$f(x) = \sum_{i=0}^{\infty} \frac{(-x)^i}{i!} = 1 - x + \frac{x^2}{2} - \frac{x^3}{6} + \ldots$$

Suppose each term has some precision $\underline{\varepsilon}$ (rel. to mag)

The summed error is thus:

$$\text{error} \sim \varepsilon \cdot \underline{1 +} \ \varepsilon \cdot x + \varepsilon \frac{x^2}{2} + \dots$$

errors are additive!

$$\therefore \ \text{error} \sim \varepsilon e^{x}$$

$$\text{so} \ \left(e^{-x}\right)_{num} = e^{-x} \pm \varepsilon e^{x}$$

$$\text{or} \ = e^{-x}\left(1 \pm \varepsilon e^{2x}\right)$$

Even for double precision, error is $O(1)$ for $x \sim 18$!

Note that the expansion was exact as it converges (eventually) for all x, but due to roundoff errors it fails for large (ish) x!

Never design an algorithm (or experiment!) where you subtract large #s to get small ones! This kills your precision!

How do we fix this? Simple!

$$e^{-x} = \frac{1}{e^{x}} = \frac{1}{\sum_{i=0}^{\infty} \frac{x^{i}}{i!}}$$

error is always $O(\varepsilon)$ for $x > 0$!

Another example of error is the combination of algorithm & round-off (precision) errors!

Suppose we have some $f(x)$ and we want the derivative $\left.\frac{df}{dx}\right|_{x_0}$

We can get this using the **forward difference** approximation:

$$\left.\frac{\partial f}{\partial x}\right|_{x_0} = \lim_{h \to 0} \frac{f(x_0+h)-f(x_0)}{h}$$

What is the error in this formula?

Algorithm error is from h being too large, while numerical error is from h being too small!

Algorithm error:

We use the Taylor Series:

$$f(x) = f(x_0) + f'(x_0)(x-x_0) + \frac{1}{2}f''(x_0)(x-x_0^2)$$
$$+ \ldots$$

We can **truncate** the series if we introduce some point ξ:

$$\xi \in [x_0, x]$$

So ξ lies between x_0 & x

So: $f(x) = f(x_0) + f'(x_0)(x-x_0) + \frac{1}{2}(x-x_0)^2 f''(\xi)$

unknown point

From the forward difference equation:

$$\left(\frac{df}{dx}\Big|_{x_0}\right)_{num} \equiv \frac{f(x_0+h)-f(x_0)}{h}$$

$$= \frac{\cancel{f(x_0)} + h f'(x_0) + \frac{1}{2}h^2 f''(\xi) - \cancel{f(x_0)}}{h}$$

$$= f'(x_0) + \frac{1}{2} h f''(\xi)$$

So the <u>algorithm</u> <u>error</u> is $\sim h$

What about the numerical error? Both $f(x_0)$ and $f(x_0+h)$ have error of $O(\varepsilon f(x_0))$ due to round off error!

Thus, round off error $\sim O\left(\frac{2\varepsilon f(x_0)}{h}\right)$

This blows up as $h \to 0$

The total error is a minimum for some intermediate h, about where both are the same magnitude.

So: $$\frac{2\varepsilon f(x_0)}{h_{opt}} \approx \frac{1}{2} h_{opt} f''(\xi)$$

So we have: $$h_{opt} \sim O\left(4\varepsilon \frac{f(x_0)}{f''(\xi)}\right)^{1/2}$$

(for small h_{opt}, $\xi \approx x_0$)

Thus, the best you can do with this formula is:

$$\left(\frac{\delta f}{\delta x}\bigg|_{x_0}\right)_{num} = f'(x_0)\left[1 + O\left(\varepsilon \frac{f f''}{f'^2}\right)^{1/2}\right]$$

So if $\varepsilon \sim 10^{-7}$, error in derivative is $O(\varepsilon^{1/2}) \sim 0.03\%$

double precision effectively becomes single precision in accuracy...

Run the Matlab example

You can do better with better algorithms. A center difference approx uses points on either side of x_0:

$$\left(\frac{\partial f}{\partial x}\bigg|_{x_0}\right)_{num} = \frac{f(x_0+h)-f(x_0-h)}{2h}$$

The roundoff error is still $O(h^{-1})$, but the algorithm error is now $O(h^2)$ ⇒ Use the Taylor Series to prove this! (also plug into example) This means the optimum h is larger, and the minimum total error is $\sim O(\varepsilon^{2/3})$.

Another class of problems where round-off errors come into play are ill-conditioned problems.

Here the algorithm is fine, the computation of the answer is precise, but the answer is wrong!

=> Because of round-off error and the nature of the problem, you are solving a slightly different problem exactly!

Suppose we want the roots of a polynomial:

$$\left(x - \tfrac{2}{3}\right)^4 = 0$$

This has the repeated root

$$x = \tfrac{2}{3}$$

Now suppose we solve this by attacking the polynomial when multiplied out:

$$x^4 - 4\tfrac{2}{3}x^3 + 6\tfrac{4}{9}x^2 - 4\tfrac{8}{27}x + \tfrac{16}{81} = 0$$

and we store all #'s to single precision. (floating pt. rep.)

If we solve the resulting problem, we get:

0.6861

$0.6663 + 0.0191i \qquad (i \equiv \sqrt{-1})$

$0.6663 - 0.0191i$

0.6480

or an error of 2.8%!

(even w/ double precision error is ~0.01%

Why? problem was ill conditioned

The computer solved a problem which was very similar to the one we wanted, but had a very different answer!

Let's see what happens if we plug one of the roots back in:

$$x = 0.6861, \quad \left(x - \tfrac{2}{3}\right)^4 = 1.4 \times 10^{-7}$$

which is within roundoff error of zero! (error for double was 1.8×10^{-17})

In an ill-conditioned problem, a small change in the coefficients yields a large change in the result.

The difficulty is in the <u>problem</u>, not the <u>algorithm</u>

Try to avoid ill-conditioned problems as much as possible.

A constant focus of the course will be how to avoid all of these types of numerical errors.

Systems of Linear Equations

We all know that 2pts determine a line, 3pts a parabola, 4pts a cubic, etc. How do we get the curves?

Say we have 3pts:

(1,0), (2,-1), (3,2)

We want to determine the coefficients of the parabola:

$p(x) = a + bx + cx^2$

Just plug in the points!

$p(1)=0, \ p(2)=-1, \ p(3)=2$

So:

$$\begin{aligned} a + b + c &= 0 \\ a + 2b + 4c &= -1 \\ a + 3b + 9c &= 2 \end{aligned}$$

We can write this in matrix form:

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 0 \\ -1 \\ 2 \end{pmatrix}$$

How do we solve this? This is of the form:

$$\underset{\approx}{A} \, \underset{\sim}{x} = \underset{\sim}{b}$$

where the coef.'s $\underset{\sim}{x}$ are unknown!

This is a <u>system</u> of linear equations!

We <u>can</u> solve by inverting $\underset{\approx}{A}$:

let $\underset{\approx}{A}^{-1}$ be a matrix s.t.

$$\underset{\approx}{A}^{-1} \underset{\approx}{A} = \underset{\approx}{I} \quad \text{(identity matrix)}$$

Then:

$$\underset{\approx}{A}^{-1} \underset{\approx}{A} \, \underset{\sim}{x} = \underset{\approx}{A}^{-1} \underset{\sim}{b}$$

$$\text{or} \quad \underset{\sim}{x} = \underset{\approx}{A}^{-1} \underset{\sim}{b}$$

This is <u>correct</u>, but <u>inefficient</u> since we don't <u>need</u> to invert $\underset{\approx}{A}$ to solve the problem! Use Gaussian elimination instead.

How does this work? The same way you would naturally do it!

Say:

$$x_1 + x_2 + x_3 = 0$$

$$x_1 + 2x_2 + 4x_3 = -1$$

$$x_1 + 3x_2 + 9x_3 = 2$$

Subtract 1st eq'n from second 2, elim. x_1 from eq'ns. Thus:

$$\begin{aligned} x_1 + x_2 + x_3 &= 0 \\ x_2 + 3x_3 &= -1 \\ 2x_2 + 8x_3 &= 2 \end{aligned}$$

Now subtract twice 2nd eq'n from third:

$$\begin{aligned} x_1 + x_2 + x_3 &= 0 \\ x_2 + 3x_3 &= -1 \\ 2x_3 &= 4 \end{aligned}$$

This is the new problem:

$$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 3 \\ 0 & 0 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0 \\ -1 \\ 4 \end{pmatrix}$$

Which is in the upper triangular form:

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ & a_{22} & & \vdots \\ 0 & & \ddots & \vdots \\ & & & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix}$$

This can be solved by back substitution

$$x_i = \begin{cases} c_n / a_{nn} \\ \left(c_i - \sum_{j=i+1}^{n} a_{ij} x_j\right) / a_{ii} & i < n \end{cases}$$

For this case,

$$x_3 = 2, \quad x_2 = -7, \quad x_1 = 5$$

How much time does this take?

We require $\sim n^3/3$ multiplications for Gaussian elimination

Back substitution takes $\sim n^2$ operations

$\therefore$ dominated by elimination!

What does this mean?

Suppose we have 100×100 matrix

$\rightarrow$ need $O(100^3) \sim 10^6$ operations

How about a 1000×1000 system?

need 1000 times longer!

A <u>big</u> problem in large scale simulations

Often matrices are <u>sparse</u> or have a banded structure that allows the use of special algorithms $\Rightarrow$ faster

A matrix is <u>sparse</u> if only a few elements are non-zero

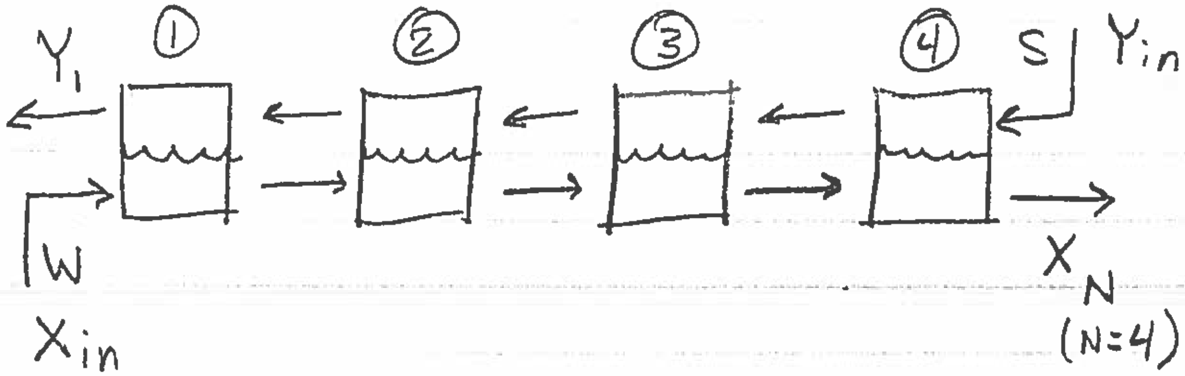
A matrix is <u>banded</u> if non-zero elements are along the diagonals

Special algorithms, faster than $O(n^3)$ can be used to solve these types of problems!

Let's examine a ChemEng. example:

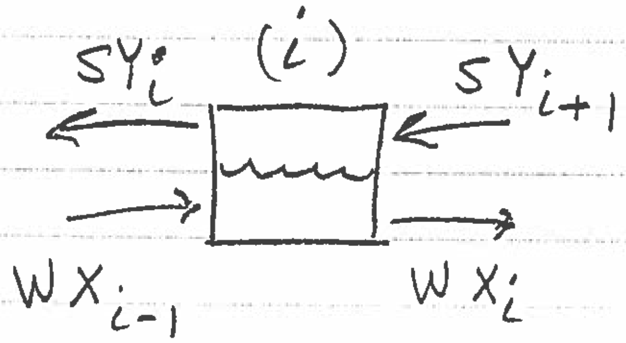
Liquid-Liquid Extraction

We have a four-stage extractor:



We are stripping a dilute solute from a water stream W by contacting this stream with a solvent stream S. The solute concentration in the water stream is X, and in the solvent stream is Y. We want to determine the outlet concentration of the water stream X_N (N=4 in this case).

In each stage we have a mass balance:



The solute in = solute out, thus:

$$W X_{i-1} + S Y_{i+1} = W X_i + S Y_i$$

We also have the equilibrium relation (Henry's Law):

$$Y_i = K X_i$$

Combining these two we get the relation for the interior stages:

$$W X_{i-1} + S K X_{i+1} = (W + SK) X_i$$

or, dividing by W and rearranging:

$$X_{i-1} - \left(1 + \frac{SK}{W}\right) X_i + \frac{SK}{W} X_{i+1} = 0$$

We need different equations for the first and last contactors:

$$W X_{in} + SK X_2 = (W + SK) X_1$$

or

$$-\left(1 + \frac{SK}{W}\right) X_1 + \frac{SK}{W} X_2 = -X_{in}$$

for the first contactor, and:

$$W X_{N-1} + S Y_{in} = (W + SK) X_N$$

or

$$X_{N-1} - \left(1 + \frac{SK}{W}\right) X_N = -\frac{S}{W} Y_{in}$$

So the problem depends on X_{in}, Y_{in} and the key ratio:

$$\chi \equiv \frac{SK}{W}$$

Let's take another look at this example

we have the equation for the first contactor:

$$-(1+\chi)x_1 + \chi x_2 = -x_{in}$$

In matrix form we get:

$$\begin{pmatrix} -(1+\chi) & \chi & 0 & \cdots & 0 \\ & & & & \\ & & & & \\ & & & & \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix} = \begin{pmatrix} -x_{in} \\ \vdots \\ \vdots \end{pmatrix}$$

Now we add in the add'l eq'ns:

$$x_{i-1} - (1+\chi)x_i + \chi x_{i+1} = 0$$

and the last eq'n

$$x_{N-1} - (1+\chi)x_N = -\chi \frac{Y_{in}}{K}$$

thus:

$$\begin{pmatrix} -(1+\chi) & \chi & 0 & 0 & 0 & 0 & \cdots & 0 \\ 1 & -(1+\chi) & \chi & 0 & 0 & \cdots & & \vdots \\ 0 & \ddots & 1 & -(1+\chi) & \chi & 0 & \ddots & 0 \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & \chi \\ & \bigcirc & & & & 0 & 1 & -(1+\chi) \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ \vdots \\ x_N \end{pmatrix} = \begin{pmatrix} -x_{in} \\ 0 \\ \vdots \\ 0 \\ -\chi \frac{Y_{in}}{K} \end{pmatrix}$$

This is a tri-diagonal matrix which can be rapidly solved using Thomas method.

Example program just uses matlab's A\b command. (a bit slower, but for a small matrix it doesn't matter)

Ok, systems of eq'ns are useful but do we always get the right answer?

Sometimes, have significant numerical error. Let's see where it comes from.

First, we look at the algorithm itself suppose we have the problem:

$$\begin{pmatrix} 10 & -7 & 0 \\ -3 & 2 & 6 \\ 5 & -1 & 5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7 \\ 4 \\ 6 \end{pmatrix}$$

Apply elim. (1st step):

$$\begin{pmatrix} 10 & -7 & 0 \\ 0 & -0.1 & 6 \\ 0 & 2.5 & 5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7 \\ 6.1 \\ 2.5 \end{pmatrix}$$

At the 2nd step we would mult 2nd row by 25 (lge #) and add to third:

$$\begin{pmatrix} 10 & -7 & 0 \\ 0 & -0.1 & 6 \\ 0 & 0 & 155 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7 \\ 6.1 \\ 155 \end{pmatrix}$$

This has the sol'n $x_3 = 1$, but the #'s are getting large - can lead to roundoff errors in lge matrices

What if the 2nd coef. is zero??

Avoid this w/ pivoting

Each row corresp. to an equation, so we just reorder them!

flip 2nd & 3rd rows:

$$\begin{pmatrix} 10 & -7 & 0 \\ 0 & 2.5 & 5 \\ 0 & -0.1 & 6 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7 \\ 2.5 \\ 6.1 \end{pmatrix}$$

Now do elimination:

$$\begin{pmatrix} 10 & -7 & 0 \\ 0 & 2.5 & 5 \\ 0 & 0 & 6.2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7 \\ 2.5 \\ 6.2 \end{pmatrix}$$

So $x_3 = 1$, $x_2 = -1$, $x_1 = 0$

In general, in $K^{\underline{th}}$ step of GE, interchange rows so that largest element of $K^{\underline{th}}$ column and $i \geq K$ rows is in $K^{\underline{th}}$ row

Then $\Rightarrow$ $$a_{ij}^{(K+1)} = a_{ij}^{(K)} - \left(a_{iK}^{(K)} / a_{KK}^{(K)} \right) a_{Kj}^{(K)}$$

for $i, j > K$

This operation is equivalent to a $\underset{\approx}{P}\underset{\approx}{L}\underset{\approx}{U}$ factorization!

$\underset{\approx}{L}$ is lower triangular matrix

$\underset{\approx}{U}$ is an upper triangular matrix

$\underset{\approx}{P}$ corresp. to pivoting. So:

$$\underset{\underset{\approx}{A}}{\begin{pmatrix} 10 & -7 & 0 \\ -3 & 2 & 6 \\ 5 & -1 & 5 \end{pmatrix}} = \underset{\underset{\approx}{P}}{\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}} \underset{\underset{\approx}{L}}{\begin{pmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0 \\ -0.3 & -0.04 & 1 \end{pmatrix}}$$

$$\cdot \underset{\underset{\approx}{U}}{\begin{pmatrix} 10 & -7 & 0 \\ 0 & 2.5 & 5 \\ 0 & 0 & 6.2 \end{pmatrix}}$$

Once we have a PLU decomposition, we can easily get the solution!

We had $\underset{\approx}{A}\,\underset{\sim}{x} = \underset{\sim}{b}$

or: $\underset{\approx}{P}\,\underset{\approx}{L}\,\underset{\approx}{U}\,\underset{\sim}{x} = \underset{\sim}{b}$

Let $\underset{\sim}{z}$ be an array s.t. $\underset{\approx}{P}\,\underset{\sim}{z} = \underset{\sim}{b}$

And $\underset{\sim}{y}$ s.t. $\underset{\approx}{L}\,\underset{\sim}{y} = \underset{\sim}{z}$

Then we have the series of problems:

$\underset{\approx}{P}\,\underset{\sim}{z} = \underset{\sim}{b}$ (rearrange $\underset{\sim}{b}$ to get $\underset{\sim}{z}$)

$\underset{\approx}{L}\,\underset{\sim}{y} = \underset{\sim}{z}$ (solve via forward substitution)

$\underset{\approx}{U}\,\underset{\sim}{x} = \underset{\sim}{y}$ (solve via back substitution)

You only have to factor $\underset{\approx}{A}$ <u>once</u>, then can solve for <u>many</u> vectors $\underset{\sim}{b}$!

Even w/ pivoting, still sensitive to errors! Look at simple example:

$$\begin{pmatrix} 0.780 & 0.563 \\ 0.457 & 0.330 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0.217 \\ 0.127 \end{pmatrix}$$

We solve this using GE where we store all #'s to 3 decimal places (mult $\equiv \frac{0.457}{0.780} \approx 0.586$

$\therefore$

$$\begin{pmatrix} 0.780 & 0.563 \\ 0 & 0.0000820 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0.217 \\ -0.000162 \end{pmatrix}$$

Solving via back substitution:

$x_2 = -1.98$, $x_1 = 1.71$

Checking the residual, we get:

$\underset{\sim}{r} = \underset{\sim}{b} - \underset{\approx}{A}\,\underset{\sim}{x} = \begin{pmatrix} -0.00206 \\ -0.00107 \end{pmatrix}$ which is about 10^{-2} to 10^{-3} times $\underset{\sim}{b}$, about right for 3 sig digit calc.

But the exact answer was really

$x_1 = 1$, $x_2 = -1$

we were off by 2x! Why? The problem was ill-conditioned.

$\underset{\approx}{A}$ was close to being singular

we always get:

size residual ~ size sol'n × size (A) × ε_{mach}

<u>but</u>

size error ~ size sol'n × cond# $(\underset{\approx}{A}) \times \varepsilon_{mach}$

What is cond $(\underset{\approx}{A})$? It is a measure of how close $\underset{\approx}{A}$ is to <u>singularity</u>

What happens if $\underset{\approx}{A}$ is singular?

First, det $(\underset{\approx}{A}) = 0$

Second, for some b's there will be no sol'n for $\underset{\sim}{x}$, while for others there will be an inf. #

Ex: $$\begin{pmatrix} 1 & 1 \\ 2 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

eq'ns are <u>linearly</u> <u>dependent</u>

any sol'n $x_2 = 1 - x_1$ will work!

and:
$$\begin{pmatrix} 1 & 1 \\ 2 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

has <u>no</u> sol'n (requires $1=0$)

To understand cond $(\underset{\sim}{A})$ we must look at norms

3 common types

Euclidean Norm (2-norm)

$$\|\underset{\sim}{x}\|_2 = \left[\sum_{i=1}^{N} (x_i^2)\right]^{1/2}$$

This is the usual physical vector length

Infinity Norm (∞-norm)

$$\|\underset{\sim}{x}\|_\infty = \max_i |x_i|$$

Gives a bound on the size of the elements of $\underset{\sim}{x}$

Manhattan Norm (1-norm)

$$\|\underset{\sim}{x}\| = \sum_{i=1}^{N} |x_i|$$

We will use this one!

All of these are specific examples of the general norm:

$$\|\underset{\sim}{x}\|_n = \left(\sum_{i=1}^{N} (x_i)^n\right)^{1/n}$$

Only $n = 1, 2, \infty$ are in common usage

What does this have to do with the condition ⁑?

We can form the scalar:

$$\frac{\|\underset{\approx}{A}\,\underset{\sim}{x}\|}{\|\underset{\sim}{x}\|} \leftarrow \text{1-norm!}$$

This depends on $\underset{\sim}{x}$!

Let $$M = \max_{\underset{\sim}{x}} \frac{\|\underset{\approx}{A}\,\underset{\sim}{x}\|}{\|\underset{\sim}{x}\|}$$

and $$m = \min_{\underset{\sim}{x}} \frac{\|\underset{\approx}{A}\,\underset{\sim}{x}\|}{\|\underset{\sim}{x}\|}$$

If $\underset{\approx}{A}$ is singular, $m = 0$!

The condition number of $\underset{\approx}{A}$ is defined as:

$$\text{cond}(\underset{\approx}{A}) \equiv \frac{M}{m} \quad !$$

Ok, now what do we do with it??

Let $\underset{\approx}{A}\,\underset{\sim}{x} = \underset{\sim}{b}$

and $\underset{\approx}{A}\,(\underset{\sim}{x} + \underset{\sim}{\Delta x}) = (\underset{\sim}{b} + \underset{\sim}{\Delta b})$

we want to see how much $\underset{\sim}{x}$ changes with a small change in $\underset{\sim}{b}$, e.g. the rel. between $\underset{\sim}{\Delta x}$ and $\underset{\sim}{\Delta b}$.

We know that:

$$\underset{\approx}{A}\,\underset{\sim}{\Delta x} = \underset{\sim}{\Delta b}$$ (subtract off original eq'n)

We <u>know</u> that

$$\|b\| \leq M\|\underset{\sim}{x}\|$$

since $M \geq \dfrac{\|\underset{\sim}{b}\|}{\|\underset{\sim}{x}\|}$ by $\text{def}^{\underline{n}}$

Similarly,

$$\|\Delta \underset{\sim}{b}\| \geq m \|\Delta \underset{\sim}{x}\|$$

$\therefore$ for all $m \neq 0$ (non-singular $\underset{\approx}{A}$):

$$\frac{\|\Delta \underset{\sim}{x}\|}{\|\underset{\sim}{x}\|} \leq \text{cond}(\underset{\approx}{A}) \frac{\|\Delta \underset{\sim}{b}\|}{\|\underset{\sim}{b}\|}$$

How do we calculate $\text{cond}(\underset{\approx}{A})$?

First, we <u>define</u> $\|\underset{\approx}{A}\| \equiv M = \max\limits_{\underset{\sim}{x} \neq 0} \dfrac{\|\underset{\approx}{A}\underset{\sim}{x}\|}{\|\underset{\sim}{x}\|}$

It turns out that:

$$\|\underset{\approx}{A}\| = \max_j \|\underset{\sim}{a}_j\| \quad \text{where}$$

$\underset{\sim}{a}_j$ are the <u>columns</u> of $\underset{\approx}{A}$

The min m is:

$$m = \frac{1}{\|\underset{\approx}{A}^{-1}\|}$$

so: $\text{cond}(\underset{\approx}{A}) = \dfrac{M}{m} = \|\underset{\approx}{A}\| \, \|\underset{\approx}{A}^{-1}\|$

Rather than calculating $\underset{\approx}{A}^{-1}$ directly (more work than LU decomposition!) the computer instead estimates it using $O(n^2)$ algorithms.

Let's look at some properties:

What's cond $(\underset{\approx}{I})$??

we have $\dfrac{\|\underset{\approx}{I}\,\underset{\sim}{x}\|}{\|\underset{\sim}{x}\|} = \dfrac{\|\underset{\sim}{x}\|}{\|\underset{\sim}{x}\|} = 1$ for <u>all</u> $\underset{\sim}{x}$

so $M = m = 1 \therefore \text{cond}\,(\underset{\approx}{I}) = 1$

The same goes for permutation matrix $\underset{\approx}{P}$:

$$\text{cond}(\underset{\approx}{P}) = 1$$

Note: cond $(\underset{\approx}{A}) \geq 1$ by definition!

What about diagonal matrices?

$$\underset{\approx}{D} \equiv \begin{pmatrix} d_{11} & & & 0 \\ & d_{22} & & \\ & & \ddots & \\ 0 & & & d_{nn} \end{pmatrix}$$

Thus $\text{cond}(\underset{\approx}{D}) = \dfrac{\max|d_{ii}|}{\min|d_{ii}|}$

Ok, for an error in $\underset{\sim}{b}$ we get the relation:

$$\frac{\|\Delta \underset{\approx}{x}\|}{\|\underset{\sim}{x}\|} \le \text{cond}(\underset{\approx}{A}) \frac{\|\Delta b\|}{\|b\|}$$

What if the error is in $\underset{\approx}{A}$ instead?

This is a bit more tricky!

Let $\underset{\sim}{x}^*$ be the solution of:

$$(\underset{\approx}{A} + \underset{\approx}{E})\underset{\sim}{x}^* = \underset{\sim}{b}$$

where $\underset{\approx}{E}$ is some error in $\underset{\approx}{A}$.

we expect $\dfrac{\|E\|}{\|A\|} = C \cdot \varepsilon_{mach}$ where

where C is some O(n) cst where n is the matrix size.

Let's look at the residual first. We have:

$$\| \underset{\sim}{b} - \underset{\approx}{A}\,\underset{\sim}{x}^{*} \| = \| \underset{\approx}{E}\,\underset{\sim}{x}^{*} \| \leq \| \underset{\approx}{E} \| \, \| \underset{\sim}{x}^{*} \|$$

by the definition of $\| \underset{\approx}{E}^{*} \|$

Dividing by $\| \underset{\approx}{A} \| \, \| \underset{\sim}{x}^{*} \|$ we get:

$$\frac{\| \underset{\sim}{b} - \underset{\approx}{A}\,\underset{\sim}{x}^{*} \|}{\| \underset{\approx}{A} \| \, \| \underset{\sim}{x}^{*} \|} = \frac{\| \underset{\approx}{E}\,\underset{\sim}{x}^{*} \|}{\| \underset{\approx}{A} \| \, \| \underset{\sim}{x}^{*} \|} \leq \frac{\| \underset{\approx}{E} \|}{\| \underset{\approx}{A} \|} = C\,\varepsilon_{mach}$$

so the residual is pretty small. Now for the error:

$$\underset{\approx}{A}\,\underset{\sim}{x} - \underset{\approx}{A}\,\underset{\sim}{x}^{*} = \underset{\sim}{b} - \underset{\approx}{A}\,\underset{\sim}{x}^{*}$$

$$\therefore \quad \underset{\sim}{x} - \underset{\sim}{x}^{*} = \underset{\approx}{A}^{-1} (\underset{\sim}{b} - \underset{\approx}{A}\,\underset{\sim}{x}^{*})$$

$$\| \underset{\sim}{x} - \underset{\sim}{x}^{*} \| = \| \underset{\approx}{A}^{-1} (\underset{\sim}{b} - \underset{\sim}{A}\,\underset{\sim}{x}^{*}) \|$$

$$\le \|\underset{\approx}{A}^{-1}\| \, \|\underset{\sim}{b} - \underset{\approx}{A}\underset{\sim}{x}^*\|$$

$$= \|\underset{\approx}{A}^{-1}\| \, \|\underset{\approx}{E}\,\underset{\sim}{x}^*\|$$

$$\le \|\underset{\approx}{A}^{-1}\| \, \|\underset{\approx}{E}\| \, \|\underset{\sim}{x}^*\|$$

Thus,

$$\frac{\|\underset{\sim}{x} - \underset{\sim}{x}^*\|}{\|\underset{\sim}{x}^*\|} \le \|\underset{\approx}{A}^{-1}\| \, \|\underset{\approx}{E}\| \equiv \|\underset{\approx}{A}^{-1}\| \, \|\underset{\approx}{A}\| \, C \, \varepsilon_{Mach}$$

$$= \mathrm{cond}(\underset{\approx}{A}) \, C \, \varepsilon_{mach}.$$

So again the error is on the order of the condition*, but is typically $O(n)$ larger than for error in the RHS vector $\underset{\sim}{b}$.

So far we have looked at problems

$$\underset{\approx}{A}\,\underset{\sim}{x} = \underset{\sim}{b}$$

where $\underset{\approx}{A}$ is a square non-singular matrix! There are many problems where $\underset{\approx}{A}$ is not square!

$$[n\ m] = \text{size}\,(\underset{\approx}{A})$$

if $n > m$ problem is over-determin

This usually occurs in regression: You are fitting a curve to datapoints & the number of parameters (m) is less than the number of data points (n)

Thus, there is no exact solution!

If $n < m$, problem is under-determi

This often happens in control: You want to reach a set temperature, and you have a heater and a cooler. You can heat it, you can cool it, but you can also do both at the same time! Thus, there are an infinite # of solutions!

Ok, how do we solve these problems? We shall use QR factorization - row & column reduction via orthogonal transformations.

Remember the permutation matrix? It is an example of an orthogonal matrix!

$$\underset{\sim}{P}^{T}\underset{\sim}{P} = \underset{\sim}{I}$$

(switch rows back!)

There are <u>many</u> examples of orthogonal matrices:

$$\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix};\ \frac{1}{2}\begin{pmatrix}1\\1\\1\\1\end{pmatrix};\ \begin{pmatrix}3/5 & -4/5\\ -4/5 & -3/5\end{pmatrix}$$

all satisfy $\underset{\approx}{P}^T \underset{\approx}{P} = \underset{\approx}{I}$

An important property of orthogonal matrices is that they preserve the 2-Norm!

$$\|\underset{\approx}{P}\underset{\sim}{x}\|_2 = \left[(\underset{\approx}{P}\underset{\sim}{x})^T(\underset{\approx}{P}\underset{\sim}{x})\right]^{1/2}$$

$$= \left[\underset{\sim}{x}^T \underset{\approx}{P}^T \underset{\approx}{P}\underset{\sim}{x}\right]^{1/2} = \left[\underset{\sim}{x}^T \underset{\sim}{x}\right]^{1/2} = \|\underset{\sim}{x}\|_2 \,!$$

This property makes orthogonal matrices useful in regression

The regression problem was:

$$\min_{\underset{\sim}{x}} \|\underset{\approx}{A}\underset{\sim}{x} - \underset{\sim}{b}\|_2$$

we can multiply this by an orthogonal matrix $\underset{\approx}{P}$:

$$\|\underset{\approx}{A}\underset{\sim}{x}-\underset{\sim}{b}\|_2=\|\underset{\approx}{P}(\underset{\approx}{A}\underset{\sim}{x}-\underset{\sim}{b})\|_2$$

$$=\|(\underset{\approx}{P}\underset{\approx}{A})\underset{\sim}{x}-(\underset{\approx}{P}\underset{\sim}{b})\|_2$$

we want to find $\underset{\approx}{P}$ s.t. $\underset{\approx}{P}\underset{\approx}{A}$ is an <u>upper</u> <u>triangular</u> form. Why??

Suppose we have a matrix $\underset{\approx}{R}$ which is of the form:

$$\underset{\approx}{R}=\begin{pmatrix}3&4&1\\0&2&2\\0&0&7\\0&0&0\\0&0&0\end{pmatrix}\quad\text{and}\quad\underset{\sim}{c}=\begin{pmatrix}14\\10\\21\\6\\2\end{pmatrix}$$

we want to minimize:

$$\|\underset{\approx}{R}\underset{\sim}{x}-\underset{\sim}{c}\|_2 \quad \text{where } \underset{\sim}{x} \text{ is } 3\times1$$

Note that we can solve the first three equations exactly and that no $\underset{\sim}{x}$ has any effect on the last 2!

$$\|\underset{\approx}{R}\,\underset{\sim}{x} - \underset{\sim}{c}\|_2^2 = \left\| \begin{pmatrix} 3 & 4 & 1 \\ 0 & 2 & 2 \\ 0 & 0 & 7 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} - \begin{pmatrix} 14 \\ 10 \\ 21 \end{pmatrix} \right\|_2^2$$

$$+ \left\| \begin{matrix} 6 \\ 2 \end{matrix} \right\|_2^2$$

We can solve the first part via back substitution:

$$x_3 = \frac{21}{7} = 3, \quad x_2 = \frac{10-6}{2} = 2,$$

$$x_1 = \frac{14-3-8}{3} = 1$$

The residual is just:

$$\left\| \begin{matrix} 6 \\ 2 \end{matrix} \right\|_2 = \sqrt{40}$$

So this sort of problem is easy to solve!

How do we find some matrix P s.t. $\underset{\approx}{P}\underset{\approx}{A} = \underset{\approx}{R}$ in upper triangular form?? We use Householder Transformations!

You use a sequence of orthogonal transformations $\underset{\approx}{P}$ which reduces $\underset{\approx}{A}$ to upper triangular form one column at a time. Thus:

$$\underset{\approx}{P}_n \cdots \underset{\approx}{P}_3 \underset{\approx}{P}_2 \underset{\approx}{P}_1 \underset{\approx}{A} = \underset{\approx}{R}$$

Because $\underset{\approx}{P}_i$ is orthogonal we have:

$$(\underset{\approx}{P}_n \cdots \underset{\approx}{P}_2 \underset{\approx}{P}_1)^T \underset{\approx}{R} \equiv \underset{\approx}{P}_1^T \underset{\approx}{P}_2^T \cdots \underbrace{\underset{\approx}{P}_n^T \underset{\approx}{P}_n}_{\underset{\approx}{I}} \cdots \underset{\approx}{P}_2 \underset{\approx}{P}_1 \underset{\approx}{A}$$

$$\equiv \underset{\approx}{A}$$

We define $\underset{\approx}{Q} \equiv (\underset{\approx}{P}_n \cdots \underset{\approx}{P}_2 \underset{\approx}{P}_1)^T$

Thus

$$\underset{\approx}{A} \equiv \underset{\approx}{Q}\,\underset{\approx}{R}$$

$$\text{s.t. } \underset{\approx}{Q}^T \underset{\approx}{Q} = \underset{\approx}{I}$$

Each matrix $\underset{\approx}{P}$ can be easily formed.

Suppose we want to reduce a column of $\underset{\approx}{A}$ given by $\underset{\sim}{a}$:

$$\underset{\approx}{P}\,\underset{\sim}{a} = \alpha\,(1, 0, 0 \cdots 0)^T \equiv \alpha\,\underset{\sim}{\hat{e}}_1$$

Remember from orthogonality that

$$\|\underset{\approx}{P}\,\underset{\sim}{a}\|_2 = \|\underset{\sim}{a}\|_2, \text{ thus } \alpha = \pm\|\underset{\sim}{a}\|_2$$

It can be shown that :

$$\underset{\approx}{P} = \underset{\approx}{I} - 2\,\frac{\underset{\sim}{v}\,\underset{\sim}{v}^T}{\underset{\sim}{v}^T\underset{\sim}{v}}$$

where $\underset{\sim}{v} = \underset{\sim}{a} - \alpha\,\underset{\sim}{\hat{e}}_1$

accomplishes this reduction.

Let's examine this matrix.

First, $\underset{\approx}{P}$ is symmetric such that

$$\underset{\approx}{P}^T = \underset{\approx}{P} \quad \text{or} \quad P_{ij} = P_{ji}$$

Second, it is orthogonal:

$$\underset{\approx}{P}^T\underset{\approx}{P} = \left(\underset{\approx}{I} - 2\frac{\underset{\sim}{v}\,\underset{\sim}{v}^T}{(\underset{\sim}{v}\cdot\underset{\sim}{v})}\right)\left(\underset{\approx}{I} - 2\frac{\underset{\sim}{v}\,\underset{\sim}{v}^T}{(\underset{\sim}{v}\cdot\underset{\sim}{v})}\right)$$

$$= \underset{\approx}{I} - 4\frac{\underset{\sim}{v}\,\underset{\sim}{v}^T}{(\underset{\sim}{v}\cdot\underset{\sim}{v})} + 4\frac{\underset{\sim}{v}\cancel{(\underset{\sim}{v}^T\underset{\sim}{v})}\underset{\sim}{v}^T}{(\underset{\sim}{v}\cdot\underset{\sim}{v})^{\cancel{2}}}$$

where the $\underset{\sim}{v}\cdot\underset{\sim}{v}$'s cancel out! Thus:

$$\underset{\approx}{P}^T\underset{\approx}{P} = \underset{\approx}{I}$$

Finally, we want to prove that our choice for $\underset{\approx}{P}$ accomplishes the desired reduction. It's actually easier to determine the choice for $\underset{\sim}{v}$ which does the job!

We seek a vector $\underset{\sim}{v}$ such that

$$\underset{\approx}{P}\,\underset{\sim}{a} = \alpha\,\underset{\sim}{\hat{e}}_1 \quad \text{where} \quad \alpha \equiv \pm \|\underset{\sim}{a}\|_2$$

Note that this <u>must</u> be true since $\underset{\approx}{P}$ is orthogonal and thus preserves the 2-norm!

$$\text{So:}\quad \underset{\approx}{P}\,\underset{\sim}{a} = \left(\underset{\approx}{I} - 2\,\frac{\underset{\sim}{v}\,\underset{\sim}{v}^T}{(\underset{\sim}{v}\cdot\underset{\sim}{v})}\right)\underset{\sim}{a}$$

$$= \underset{\sim}{a} - 2\,\frac{\underset{\sim}{v}\,\underset{\sim}{v}^T\underset{\sim}{a}}{(\underset{\sim}{v}\cdot\underset{\sim}{v})} = \alpha\,\underset{\sim}{\hat{e}}_1$$

Rearranging:

$$\underbrace{2\,\frac{(\underset{\sim}{v}\cdot\underset{\sim}{a})}{(\underset{\sim}{v}\cdot\underset{\sim}{v})}}_{\text{scalar}}\,\underset{\sim}{v} = \underset{\sim}{a} - \alpha\,\underset{\sim}{\hat{e}}_1$$

Now we note that $\underset{\approx}{P}$ is independent of the <u>magnitude</u> of $\underset{\sim}{v}$ — you can multiply it by any scalar, and the resulting $\underset{\approx}{P}$ will be unchanged! Thus any vector proportional to:

$$\underset{\sim}{V} = \underset{\sim}{a} - \alpha \, \underset{\sim}{\hat{e}}_1$$

<u>will</u> accomplish the desired reduction.

OK, so how do we use this?

We have the problem:

$$\min_{\underset{\sim}{x}} \| \underset{\approx}{A} \underset{\sim}{x} - \underset{\sim}{b} \|_2$$

$$= \min_{\underset{\sim}{x}} \| \underset{\approx}{Q} \underset{\approx}{R} \underset{\sim}{x} - \underset{\sim}{b} \|_2$$

$$= \min_{\underset{\sim}{x}} \| \underset{\approx}{R} \underset{\sim}{x} - \underset{\approx}{Q}^T \underset{\sim}{b} \|_2 \quad \leftarrow \text{by } \underline{\text{orthagonality}}$$

$$= \min_{\underset{\sim}{x}} \| \underset{\approx}{R} \underset{\sim}{x} - \underset{\sim}{c} \|_2$$

$$\text{where } \underset{\sim}{c} \equiv \underset{\approx}{Q}^T \underset{\sim}{b}$$

we solve the first m rows <u>exactly</u>, while the last n-m elements of $\underset{\sim}{c}$ is the <u>residual</u>!

accomplishes this reduction

~~Work out matlab demonstration of QR factorization.~~

Degenerate Problems

What happens if the problem is under determined??

Corresponds to matrix $\underset{\sim}{R}$ not being of full rank

rank ≡ largest submatrix w/ non-zero determinant.

In linear regression this arises if the modelling functions are linearly dependent

Example:

$$b(t) = x_1 \cdot (1) + x_2 \cdot (t) + x_3 \cdot (2t+1)$$

For this problem there will be an

infinite number of ways to approximate data w/ the same residual (minimum)

In this case:

$$\alpha\left[1\cdot(1)+2(t)-1\cdot(2t+1)\right]=0$$

for all α, thus

$$\underset{\sim}{x}=(x_1,x_2,x_3)$$

can be replaced with

$$\underset{\sim}{x}^{*}=\underset{\sim}{x}+\alpha\,(1,2,-1)$$

without changing the residual

A linear regression problem will be degenerate if the columns of $\underset{\approx}{A}$ (an $m\times n$ matrix) are linearly dependent

skip

The line between degeneracy & non-degeneracy may be blurred by round off error!

Suppose we had:

$$\begin{pmatrix} 1 & 1 \\ 0 & 10^{-k} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

This has the solution:

$$x_2 = 10^k, \quad x_1 = \frac{1-10^k}{1} \approx -10^k$$

Now suppose $k >> 1$ and A_{22} is non-zero *only* due to roundoff error

We would prefer to fit the data with:

$$x_2 = 0, \quad x_1 = 1$$

The *size* of $\|\underset{\sim}{x}\|_2 = 1$

vs. $\sqrt{2}\ 10^k$!

You usually want to keep your

modelling parameters <u>small</u>, thus we seek a vector with the shortest length for degenerate problems!

Sometimes it's hard to tell how close a matrix is to degeneracy by looking at it. Take:

$$\underset{\sim}{A} = \begin{pmatrix} 1 & -1 & -1 & -1 & \cdots & -1 \\ & 1 & -1 & -1 & \cdots & -1 \\ & & 1 & -1 & \cdots & -1 \\ & 0 & & 1 & \cdots & -1 \\ & & & & \ddots & \vdots \\ & & & & & 1 \end{pmatrix}$$

The det. of this matrix is $\det(A) = 1$

If $A_{n1} = 2^{-(n-2)}$ then the matrix is singular!

Although $\underset{\sim}{A}$ is in upper triangular form already, we can do a QR on it w/ column pivoting!

Pick the column w/ lgest norm, move to column 1 & reduce it. Then pick next largest column of $(n-1)\times(n-1)$ submatrix & move to column 2, etc.

This yields:

$$\begin{pmatrix} 1 & -1 & -1 & -1 & -1 \\ & 1 & -1 & -1 & -1 \\ & & 1 & -1 & -1 \\ & 0 & & 1 & -1 \\ & & & & 1 \end{pmatrix} \Rightarrow \begin{pmatrix} 2.24 & 0.89 & .45 & 0 & -.45 \\ & 1.79 & .34 & 0 & -.34 \\ & & -1.64 & 0 & .42 \\ & 0 & & -1.4 & .71 \\ & & & & 0.108 \end{pmatrix}$$

Note that the last diagonal element is an order of magnitude smaller than the other diag elements! This is because A is close to singularity!

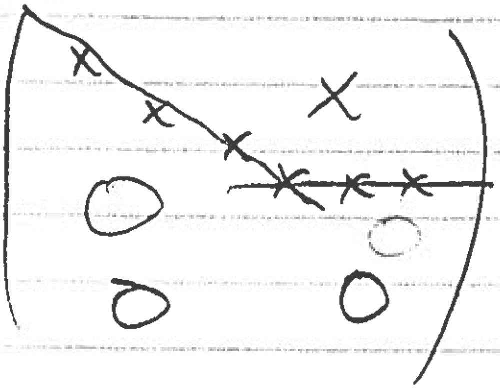
This is equivalent to the factorization:

$$\underset{\sim}{A} = \underset{\sim}{Q}\,\underset{\sim}{R}\,\underset{\sim}{P} \leftarrow \text{rt. mult interchanging columns of } \underset{\sim}{A}$$

skip

If A is singular, R is of form:

(w/ QRP)



The number of non-zero rows in R is the rank of A

We thus reduce the regression problem to:

$$\| A x - b \|_2 = \| R P x - Q^T b \|_2$$

To solve this, let:

$$y = P x \quad \text{and} \quad c = Q^T b$$

Then

$$\min_{y} \| R y - c \|_2 \text{ may be}$$

solved via back substitution

We can go even further than this, and do Singular Value Decomposition!

We operated on $\underset{\approx}{A}$ to factor it into an orthogonal matrix and an upper triangular matrix. We can operate on $\underset{\approx}{R}$ to factor *it* into a diagonal matrix and an orthogonal matrix!

Thus:

$$\underset{\substack{\nearrow \\ m\times n}}{\underset{\approx}{A}} = \underset{\substack{\nearrow \\ \text{orthogonal} \\ m\times m}}{\underset{\approx}{U}} \; \overset{\substack{\text{diagonal} \\ m\times n \\ \downarrow}}{\underset{\approx}{\Sigma}} \; \underset{\substack{\uparrow \\ \text{orthogonal} \\ n\times n}}{\underset{\approx}{V}}^{T}$$

Using pivoting, we order the elements of Σ s.t.

$$|\sigma_1| \geq |\sigma_2| \geq |\sigma_3| \geq \cdots \geq |\sigma_n|$$

If $\sigma_n = 0$ then $\underset{\approx}{A}$ is not of full rank! $(m > n)$

If A is square, then $|\sigma_1/\sigma_n|$ is the condition number of $\underline{\underline{A}}$

SVD is useful for many purposes. First, we have the data fitting/regression problem:

$$\min_{\underset{\sim}{x}} \|\underset{\approx}{A}\,\underset{\sim}{x} - \underset{\sim}{b}\|_2 = \min_{\underset{\sim}{x}} \|\underset{\approx}{U}\,\underset{\approx}{\Sigma}\,\underset{\approx}{V}^T \underset{\sim}{x} - \underset{\sim}{b}\|_2$$

$$= \min_{\underset{\sim}{x}} \|\underset{\approx}{\Sigma}\,\underset{\approx}{V}^T \underset{\sim}{x} - \underset{\approx}{U}^T \underset{\sim}{b}\|_2$$

let $\underset{\approx}{V}^T \underset{\sim}{x} = \underset{\sim}{z}$, $\underset{\approx}{U}^T \underset{\sim}{b} = \underset{\sim}{d}$

$$\therefore \underset{\approx}{V}\,\underset{\approx}{V}^T \underset{\sim}{x} \equiv \underset{\sim}{x} = \underset{\approx}{V}\,\underset{\sim}{z}$$

and $z_i = d_i/\sigma_i$

and $r^2 = \sum_{n+1}^{m} d_i^2$ *if* $\sigma_n \neq 0$

If $\sigma_n = 0$ then the problem is *singular*

If $\sigma_n \approx 0$ (or of order ε_{mach}) we may want to regard it as zero so it doesn't mess up the calculation of $\underset{\sim}{x}$!

Thus we let $$z_i = \begin{cases} d_i/\sigma_i & \sigma_i > \varepsilon \\ 0 & \sigma_i < \varepsilon \end{cases}$$

This has the property that among the set of vectors $\underset{\sim}{x}^*$ which minimize the residual:

$$\|\underset{\approx}{A}\underset{\sim}{x} - \underset{\sim}{b}\|_2$$

That computed via SVD will have the minimum norm:

$$\|\underset{\sim}{x}_{SVD}\|_2 \le \|\underset{\sim}{x}\|_2$$

skip

Another use for SVD is data compression

Recall $\underset{\approx}{A} = \underset{\approx}{U}\,\underset{\approx}{\Sigma}\,\underset{\approx}{V}^T$

SVD: Application to Control

Control problems often reduce to a set of linear equations where a limited number of control actions (parameters) are chosen to achieve some desired outcomes. Even if the underlying problem is non-linear, use of the Taylor Series can lead to linearization

These problems are often addressed via SVD. Suppose we want to minimize:

$$\underset{\sim}{r} = \overset{\overset{\text{model}}{\downarrow}}{\underset{\approx}{A}}\;\underset{\underset{\underset{\text{control parameters}}{\uparrow}}{\sim}}{x} - \overset{\overset{\text{target}}{\swarrow}}{\underset{\sim}{b}}$$

We can "solve" this in a least-squares sense via SVD as we just showed!

In control, however, the problem is often close to singular – thus since the smallest singular values are close to zero, the solution vector $\underset{\sim}{x}$ blows up! We wind up with large errors in this case, and can have unstable control due to problems w/ the solution vector.

Example: plastic bag fabrication
Addition of more actuators for control of molding / bag thickness actually led to worse performance! (Richard Braatz, MIT)

We can fix this by doing SVD and setting the smallest singular values to zero! This is equiv. to controlling actuators in a reduced pattern (not independetly)

SVD: Sensitivity Analysis

We look at a vector function of a vector ($\underset{\sim}{x}$)

$$\underset{\sim}{z} = \underset{\sim}{f}(\underset{\sim}{x})$$

Expand about $\underset{\sim}{x}_0$:

$$\underset{\sim}{z}_0 = \underset{\sim}{f}(\underset{\sim}{x}_0) \quad , \quad \underset{\sim}{z}' = \underset{\sim}{z} - \underset{\sim}{z}_0 \, , \quad \underset{\sim}{x}' = \underset{\sim}{x} - \underset{\sim}{x}_0$$

$$\underset{\sim}{z} = \underset{\sim}{f}(\underset{\sim}{x}_0) + \underset{\sim}{\nabla}\underset{\sim}{f}\,(\underset{\sim}{x} - \underset{\sim}{x}_0) + O(\underset{\sim}{x}'^2)$$

so

$$\underset{\sim}{z}' \simeq \underset{\sim}{\nabla}\underset{\sim}{f}\;\underset{\sim}{x}'$$

$\underset{\sim}{\nabla}\underset{\sim}{f}$ details the sensitivity of $\underset{\sim}{z}$ to changes in $\underset{\sim}{x}$!

What is $\underset{\sim}{\nabla}\underset{\sim}{f}$?

$$\underset{\sim}{\nabla}\underset{\sim}{f} \equiv \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_m} \\ \vdots & & & \vdots \\ \frac{\partial f_n}{\partial x_1} & & \cdots & \frac{\partial f_n}{\partial x_m} \end{bmatrix}$$

We can <u>study</u> <u>this</u> using SVD!

Let $\underset{\approx}{A} = \underset{\sim}{\nabla} \underset{\sim}{f}$

So $\underset{\sim}{z}' = \underset{\approx}{A} \underset{\sim}{x}'$

Let $\underset{\approx}{U} \underset{\approx}{\Sigma} \underset{\approx}{V}^T = \underset{\approx}{A}$

where $\underset{\approx}{U}^T \underset{\approx}{U} = \underset{\approx}{I}$

$\underset{\approx}{V}^T \underset{\approx}{V} = \underset{\approx}{I}$

and $\underset{\approx}{\Sigma} = \begin{bmatrix} \sigma_{11} & & 0 \\ & \ddots & \\ 0 & & \sigma_{nn} \end{bmatrix}$

min of n, m if matrix isn't square

Now because $\underset{\approx}{U}$ and $\underset{\approx}{V}$ are orthogonal, they preserve the 2-norm of any vector.

Thus: $\|\underset{\approx}{U} \underset{\sim}{y}\|_2 = \|\underset{\sim}{y}\|_2$

This means that the columns of $\underset{\approx}{U}$ and $\underset{\approx}{V}$ are also orthogonal!

This is easy to show. Let's look at the product:

$$\underset{\approx}{C} = \underset{\approx}{U}^{T} \underset{\approx}{U} = \underset{\approx}{I}$$

Now the element C_{ij} is the inner product of the i^{th} column of $\underset{\approx}{U}$ (e.g., $\underset{\sim}{u}_i$) w/ the j^{th} column!

$$C_{ij} = \underset{\sim}{u}_i^{T} \underset{\sim}{u}_j = I_{ij}$$

But if $i \neq j$ this is an off-diag. element of the identity matrix!

$$\text{So} : \quad \underset{\sim}{u}_i^{T} \underset{\sim}{u}_j = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases}$$

The same applies to $\underset{\approx}{V}$!

We can use the values (magnitude) of σ_{ii} to explore the sensitivity of $\underset{\sim}{z}$ to variations in $\underset{\sim}{X}$.

Suppose the model $\underset{\sim}{z} = \underset{\sim}{f}(\underset{\sim}{x})$ represents a multiple input / multiple output model of plant operations.

$\underset{\sim}{x}$ might be the mass flow rate of input streams

$\underset{\sim}{z}$ might be the resulting products needed elsewhere in the plant.

In general, you want to minimize $\underset{\sim}{z}'$ - the deviation from some optimal output $\underset{\sim}{z}_0$.

Question: what fluctuations $\underset{\sim}{x}'$ give rise to the maximum change in $\underset{\sim}{z}'$?

This is addressed by SVD, looking at the largest singular value σ_{11}.

The most dangerous mode is any vector of disturbances x' proportional to the first column of $\underset{\approx}{\tilde{V}}$

The resulting output vector is proportional to the first column of $\underset{\sim}{U}$

The amplification factor is the corresponding singular value σ_{11}

If the i^{th} singular value is zero, then ~~that~~ a disturbance vector $\underset{\sim}{x}' \sim V(:,i)$ (i^{th} column of $\underset{\sim}{V}$) has no effect on $\underset{\sim}{z}$!

In the homework project 3 you are doing the same thing, only in reverse: what is the magnitude of $\underset{\sim}{x}$ to get some output $\underset{\sim}{z}$.

Thus, you are interested in the smallest singular values, rather than (as in the usual case) the largest.

This all is an example of Principal Component Analysis, or PCA.

Data Compression

Let $\underset{\sim}{u}_i$ be the <u>columns</u> of $\underset{\approx}{U}$ and let $\underset{\sim}{v}_i$ be the <u>columns</u> of $\underset{\approx}{V}$

(or the rows of $\underset{\approx}{V}^T$)

Then:

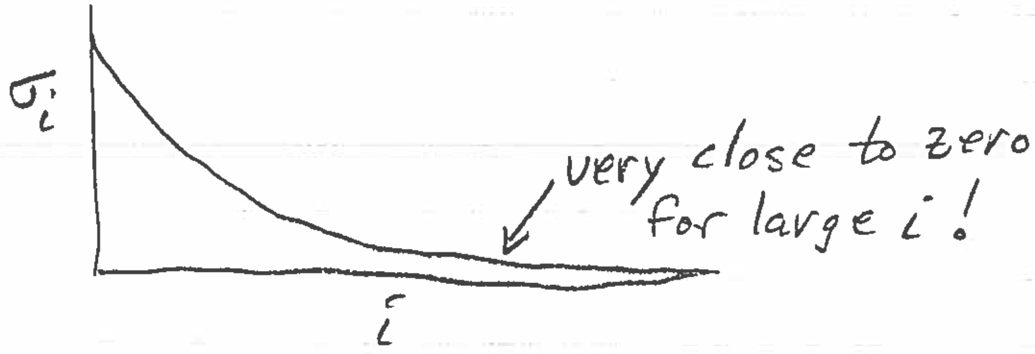
$$\underset{\approx}{A} = \sum_{i=1}^{n} \sigma_i \, \underset{\sim}{u}_i \, \underset{\sim}{v}_i^T$$

vector composition or outer product

Suppose $\underset{\approx}{A}$ is a huge matrix, say

$\underset{\approx}{A} \equiv 512 \times 512$ (usual picture size)

If we plot σ_i vs. i we get:



very close to zero for large i!

Thus if we truncate picture after $i \cong 20$ we can reproduce the original picture!

Take $$\underset{\approx}{A} \cong \sum_{i=1}^{20} \sigma_i \, \underset{\sim}{u}_i \, \underset{\sim}{v}_i^T$$

This has a total of

$$20 \times (1 + 512 + 512) = 20{,}500$$

vs. 262,144 elements!

For larger matrices you get even greater reduction in size

Now SVD requires a large # of computations, so it's a tradeoff between storage/transmission considerations and comp. speed!

Recently, transmission rates have not kept up w/ processor speed, so equation has shifted toward computationally expensive compression

An extreme example: the Gallileo Jupiter probe

- stuck high gain antenna, so must use wimpy low gain antenna

- transmission rate = $O(10 \text{ bps})$!

⇒ Use on-board computers to compress images, then transmit very slowly!

They are using JPEG compression - not quite SVD, but related. It's using fourier transform of data & only transmitting some of the modes.

You really appreciate data compression when downloading images over a modem!

~~Run matlab example~~

we'll look at SVD again later

Systems of linear equations

Summary of Solution approaches

$$\underset{\approx}{A}\,\underset{\sim}{x} = \underset{\sim}{b}$$

1) If $\underset{\approx}{A}$ is <u>square</u> & <u>well conditioned</u>

a) PLU factorization (Gaussian elimination

b) $\underset{\approx}{A}^{-1}$ so $\underset{\sim}{x} = \underset{\approx}{A}^{-1}\underset{\sim}{b}$

- convenient, but
- squares $\text{cond}(\overset{\approx}{A})$ & takes 3x longer...

c) If $\underset{\approx}{A}$ is <u>sparse</u> special techniques are much fa

2) If $\underset{\approx}{A}$ is <u>square</u> but close to singularity

a) QRP factorization
(zero out rows of $\underset{\approx}{R}$ which are very small)

b) SVD
(zero out singular values which are very small)
⇒ computationally intensive, but guaranteed to get sol'n w/ min 2-norm

3) If $\underset{\approx}{A}$ is overdetermined $(n>m)$
– often occurs in regression

a) QRP

b) $\underset{\approx}{A}^T \underset{\approx}{A} \underset{\sim}{x} = \underset{\approx}{A}^T \underset{\sim}{b}$

both minimize 2-norm of residual

- square. $m \times m$ problem
- solve via LU, inverse, etc.
- if m is large, cond maybe a problem

4) If $\underset{\approx}{A}$ is underdetermined $(m>n)$

- Use SVD (min 2-norm of $\underset{\sim}{x}$)

SVD: Uses

1) Solve linear equations, particularly those close to singularity

2) Get <u>approx</u> solutions which yield more robust control by zeroing out smallest singular values

3) Analyze transfer functions:

$\underset{\approx}{U}$ ≡ orthonormal basis set of output vectors

$\underset{\approx}{V}$ = orthonormal basis set of input vectors

$\underset{\approx}{\Sigma}$ ≡ amplification factors

4) Compress images:

outer product yields matrix

singular values

$$\underset{\approx}{A} = \underset{i}{\text{sum}} \left\{ \underset{\sim}{u}_i \, \underset{\sim}{v}_i^T \, \sigma_{ii} \right\}$$

col. of $\underset{\approx}{U}$ col of $\underset{\approx}{V}$

- computationally expensive, but good for very large $\underset{\approx}{A}$

5) Face Detection:

The columns of $\underset{\approx}{U}$ are an ordered (by importance) set of eigenfaces

6) Principal Component Analysis:
By examining the <u>projection</u> of a data set on $\underset{\approx}{U}$, can distinguish pop:

Statistics & Probability

What is error?

Two kinds: Random & Systematic

Random error is the scatter you get from repeated measurements

Systematic error is <u>systematic</u>: it happens to the same degree <u>each time</u> you make a measurement.

Multiple measurements will average out (reduce) random error, but <u>won't</u> affect systematic error!

If you do many measurements, the error in the avg. will be <u>completely</u> dominated by systematic error, but this is <u>very</u> difficult to estimate!

Be wary of error estimates — always look to see how it is calculated.

As an engineer, a detailed understanding of statistics and error is <u>critical</u>; You **must** master this material!

OK, now for some elementary statistics:

The most important probability distribution is the Gaussian or normal distribution:

$$p(x) = \frac{1}{\sqrt{2\pi}\,\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$p(x) \equiv$ probability density

$$= \frac{\text{probability of meas. being in interval } [x, x+dx]}{dx}$$

We can define the cumulative probability:

$$P(x) = \int_{-\infty}^{x} p(x')dx'$$

This is the probability that a meas. is less than some value x

The normal distribution is characterized by 2 quantities:

$\mu \equiv$ mean of distribution

$\sigma \equiv$ population standard deviation

About 69% of observations lie within 1σ of μ, 95% within 2σ, and 99% within 3σ.

In working w/ statistics, we define an <u>expectation</u> <u>value</u>

$E(x) \equiv$ what you expect to get if you do a meas. many times and average it together!

Mathematically:

$$E(x) \equiv \int_{-\infty}^{\infty} \underset{\substack{\uparrow \\ \text{meas. quantity}}}{x} \; \underset{\substack{\uparrow \\ \text{prob. density}}}{p(x)} \, dx$$

$$= \int_{-\infty}^{\infty} \frac{x}{\sqrt{2\pi}\,\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx$$

$$= \int_{-\infty}^{\infty} \frac{x-\mu}{\sqrt{2\pi}\,\sigma^2} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx + \mu \int_{-\infty}^{\infty} p(x) dx$$

Let $z \equiv x - \mu$

$$\therefore E(x) = \int_{-\infty}^{\infty} \underbrace{\frac{z}{\sqrt{2\pi}\,\sigma^2}}_{\text{odd}} \underbrace{e^{-\frac{z^2}{2\sigma^2}}}_{\text{even}} dz + \mu \underbrace{\int_{-\infty}^{\infty} p(x) dx}_{=1}$$

The integral of an <u>odd</u> function over an <u>even</u> domain is <u>zero</u>, so first integral vanishes!

Thus $E(x) = \mu$

What about the mean of N observations?

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

$$\therefore E(\bar{x}) = E\left(\frac{1}{N}\sum_{i=1}^{N} x_i\right)$$

$$= \frac{1}{N}\sum_{i=1}^{N} E(x_i) = \frac{1}{N}\sum_{i=1}^{N}\mu = \mu!$$

This may have been self-evident, but we can use the same approach for the variance:

What is $E((x-\mu)^2)$?

$$E((x-\mu)^2) \equiv \int_{-\infty}^{\infty} (x-\mu)^2 \, p(x)\,dx$$

$$= \int_{-\infty}^{\infty} \frac{(x-\mu)^2}{\sqrt{2\pi}\,\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}\,dx$$

Let $z = (x-\mu)$

$$\therefore \quad E((x-\mu)^2) = \int_{-\infty}^{\infty} \frac{z^2}{\sqrt{2\pi}\,\sigma} e^{-\frac{z^2}{2\sigma^2}} dz$$

Integrate by parts:

$$= \underbrace{-\frac{\sigma^2 z}{\sqrt{2\pi}\,\sigma} e^{-\frac{z^2}{2\sigma^2}} \Big|_{-\infty}^{\infty}}_{\text{vanishes at } \pm\infty} + \sigma^2 \underbrace{\int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}\,\sigma} e^{-\frac{z^2}{2\sigma^2}} dz}_{=1}$$

$$\therefore \underline{\underline{E((x-\mu)^2) \equiv \sigma^2}}$$

What about the variance of the <u>mean</u> of N measurements?

$$E((\bar{x}-\mu)^2) \equiv \sigma_{\bar{x}}^2$$

$$= E\left(\left[\frac{1}{N}\sum_{i=1}^{N} x_i - \mu\right]^2\right)$$

$$= \frac{1}{N^2} E\left(\left[\sum_{i=1}^{N} x_i - N\mu\right]^2\right) = \frac{1}{N^2} E\left(\left(\sum_{i=1}^{N} (x_i-\mu)\right)^2\right)$$

$$= \frac{1}{N^2} E\left(\sum_{i=1}^{N} (x_i-\mu)^2 + \sum_{i=1}^{N} \sum_{j \neq i} (x_i-\mu)(x_j-\mu)\right)$$

$$= \frac{1}{N} E\left(\frac{1}{N}\sum_{i=1}^{N} (x_i-\mu)^2\right)$$

$$+ E\left(\frac{1}{N^2}\sum_{i=1}^{N} \sum_{j\neq i} (x_i-\mu)(x_j-\mu)\right)$$

$$= \frac{1}{N^2}\sum_{i=1}^{N} E\left((x_i-\mu)^2\right)$$

$$+ \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j\neq i} E\left((x_i-\mu)(x_j-\mu)\right)$$

First expectation is just σ^2

Second term is <u>zero</u> provided x_i & x_j are <u>independent</u>

$$E\left((x_i-\mu)(x_j-\mu)\right) \equiv \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} (x_i-\mu)(x_j-\mu) \cdot p(x_i)\,P(x_j)\,dx_i\,dx_j$$

$$\equiv \int_{-\infty}^{\infty} (x_i-\mu)\,P(x_i)\,dx_i \int_{-\infty}^{\infty} (x_j-\mu)\,P(x_j) = 0 \quad i \neq j$$

Thus:

$$\sigma_{\bar{x}}^2 \equiv \frac{\sigma^2}{N}$$

That is why multiple measurements reduce the random error!

Ok, how do we estimate μ and σ?

In general we have a sample of N observations drawn from an underlying population of possible observations!

we define the sample mean

$$\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$$

we have already shown that

$$E(\bar{x}) = \mu$$

That is, $\bar{x}$ is an unbiased estimate of μ.

Now for the variance:

We Define the sample variance:

$$S_x^2 \equiv \frac{1}{N-1} \sum_{i=1}^{N} (x_i - \bar{x})^2$$

$$= \frac{1}{N-1} \sum_{i=1}^{N} \left(x_i^2 - 2x_i\bar{x} + \bar{x}^2 \right)$$

$$= \frac{1}{N-1} \left\{ \sum_{i=1}^{N} x_i^2 - 2\bar{x} \underbrace{\sum_{i=1}^{N} x_i}_{\equiv N\bar{x}} + N\bar{x}^2 \right\}$$

$$= \frac{1}{N-1} \sum_{i=1}^{N} (x_i^2 - \bar{x}^2)$$

Note that to calculate this we don't have to store all the x_i's!

We just need to keep N, $\sum_{i=1}^{N} x_i$ and $\sum_{i=1}^{N} x_i^2$ ⇒ that's how a calculator with only a few memories can calc. S_x^2.

Ok, how is s_x^2 related to σ^2?

We need the following identities:

$$\sum_{i=1}^{N} (x_i-\mu)^2 = \sum_{i=1}^{N} (x_i-\bar{x})^2 + N(\bar{x}-\mu)^2$$

(this is because $\sum_{i=1}^{N}(x_i-\bar{x}) = 0$ by the definition of $\bar{x}$)

Also, we had:

$$E\left((\bar{x}-\mu)^2\right) \equiv \sigma_{\bar{x}}^2 = \frac{\sigma^2}{N}$$

and $E\left((x_i-\mu)^2\right) = \sigma^2$

Thus:

$$E(s_x^2) \equiv E\left(\frac{1}{N-1}\sum_{i=1}^{N}(x_i-\bar{x})^2\right)$$

$$= \frac{1}{N-1}\left\{E\left(\sum_{i=1}^{N}(x_i-\mu)^2\right) - N\,E\left((\bar{x}-\mu)^2\right)\right\}$$

$$= \frac{1}{N-1}\left\{N\sigma^2 - \frac{N}{N}\sigma^2\right\} = \sigma^2 !$$

So S_x^2 is an unbiased estimate for σ^2!

The $N-1$ part comes about because we are calculating the <u>mean</u> from the same sample that we are calculating the <u>variance</u>

We have reduced the number of <u>degrees</u> <u>of</u> <u>freedom</u> by one.

Propagation of Errors:

Usually when doing experimental calculations you must combine several measurements to get the final result.

Each individual meas. has some error associated with it. How do these combine?

Let x_i, y_i be random variables w/ mean $\bar{x}, \bar{y}$ and st. dev. s_x & s_y

Let
$$z_i = c_1 x_i + c_2 y_i$$

where c_1 & c_2 are csts.

$$\therefore \bar{z} = \frac{1}{N} \sum_{i=1}^{N} z_i = c_1 \bar{x} + c_2 \bar{y}$$

What about the variance?

$$s_z^2 \equiv \frac{1}{N-1} \sum_{i=1}^{N} (z_i - \bar{z})^2$$

$$= \frac{1}{N-1} \left\{ \sum_{i=1}^{N} (z_i^2 - \bar{z}^2) \right\}$$

Substituting in:

$$z_i^2 = c_1^2 x_i^2 + c_2^2 y_i^2 + 2 c_1 c_2 x_i y_i$$

$$\bar{z}^2 = c_1^2 \bar{x}^2 + c_2^2 \bar{y}^2 + 2 c_1 c_2 \bar{x} \bar{y}$$

$$\therefore S_z^2 \equiv \frac{1}{N-1} \sum_{i=1}^{N} \Big(C_1^2 (x_i^2 - \bar{x}^2) + C_2^2 (y_i^2 - \bar{y}^2)$$

$$+ 2 C_1 C_2 (x_i y_i - \bar{x}\bar{y}) \Big)$$

$$= C_1^2 S_x^2 + C_2^2 S_y^2 + \frac{2 C_1 C_2}{N-1} \sum_{i=1}^{N} (x_i y_i - \bar{x}\bar{y})$$

Let's look at this last term. We define:

$$S_{xy}^2 \equiv \frac{1}{N-1} \sum_{i=1}^{N} (x_i y_i - \bar{x}\,\bar{y})$$

$$= \frac{1}{N-1} \sum_{i=1}^{N} (x_i - \bar{x})(y_i - \bar{y})$$

This is the covariance of x and y. If they are independent, then:

$$E(S_{xy}^2) = 0$$

So for addition:

$$S_z^2 = C_1^2 S_x^2 + C_2^2 S_y^2 + 2 C_1 C_2 S_{xy}^2$$

If we subtract two variables we get a similar result:

$$z_i = x_i - y_i$$

then $\bar{z} = \bar{x} - \bar{y}$; $s_z^2 = s_x^2 + s_y^2 - 2 s_{xy}^2$

note (-) sign!

A positive covariance (note: s_{xy}^2 may be + or -) reduces the error in the difference between variables.

OK, how about multiplication and division?

Let $z_i = x_i / y_i$

where x_i and y_i are normally distributed.

Note that z is not normally distributed! It only approaches this if $s_x / \bar{x}$ and $s_y / \bar{y} \ll 1$!

Let's look at this more closely.

Let $x_i = \underset{\overset{\uparrow}{\text{mean}}}{\bar{x}} + \underset{\overset{\uparrow}{\text{deviation}}}{x_i'}$

and $y_i = \bar{y} + y_i'$

Suppose $s_x/\bar{x} \ll 1$ and $s_y/\bar{y} \ll 1$

This means that x_i' and y_i' are small:

$$z_i = \frac{x_i}{y_i} = \frac{\bar{x} + x_i'}{\bar{y} + y_i'} = \frac{\bar{x}}{\bar{y}} \, \frac{(1 + x_i'/\bar{x})}{(1 + y_i'/\bar{y})}$$

$$\approx \frac{\bar{x}}{\bar{y}} \left(1 + \frac{x_i'}{\bar{x}}\right)\left(1 - \frac{y_i'}{\bar{y}} + O\left(\left(\frac{y_i'}{\bar{y}}\right)^2\right)\right)$$

$$\approx \frac{\bar{x}}{\bar{y}} \left(1 + \frac{x_i'}{\bar{x}} - \frac{y_i'}{\bar{y}} + O\left(\frac{x_i' y_i'}{\bar{x}\bar{y}}, \left(\frac{y_i'}{\bar{y}}\right)^2\right)\right).$$

We ignore the higher order terms!

$$\bar{z} \equiv \frac{1}{N}\sum_{i=1}^{N}\frac{x_i}{y_i} \approx \frac{\bar{x}}{\bar{y}}\frac{1}{N}\sum_{i=1}^{N}\left(1+\underbrace{\cancel{\frac{x_i'}{\bar{x}}}+\cancel{\frac{y_i'}{\bar{y}}}}_{\text{Sum to zero}}\right)$$

$$\therefore\ \bar{z} \approx \frac{\bar{x}}{\bar{y}}$$

Now for s_z^2:

$$s_z^2 \approx \underbrace{\left(\frac{\bar{x}^2}{\bar{y}^2}\right)}_{\approx \bar{z}^2}\left(\frac{s_x^2}{\bar{x}^2}+\frac{s_y^2}{\bar{y}^2}-2\frac{s_{xy}^2}{(\bar{x}\,\bar{y}}\right)$$

Thus:

$$\frac{s_z^2}{\bar{z}^2} \cong \frac{s_x^2}{\bar{x}^2}+\frac{s_y^2}{\bar{y}^2}-2\frac{s_{xy}^2}{\bar{x}\,\bar{y}}$$

so for division you add the *fractional* or *relative* variance!

For multiplication you get the same:

$$z_i \equiv x_i y_i$$

$$\bar{z} \approx \bar{x}\,\bar{y} \quad \text{provided} \quad \frac{s_x}{\bar{x}}, \frac{s_y}{\bar{y}} \ll 1$$

and

$$\frac{s_z^2}{\bar{z}^2} \cong \frac{s_x^2}{\bar{x}^2} + \frac{s_y^2}{\bar{y}^2} \underset{\substack{\uparrow \\ \text{note } (+) \text{ sign}}}{+} 2\,\frac{s_{xy}^2}{\bar{x}\,\bar{y}}$$

What about a more general functional relationship?

Suppose $z_i = f(x_i, y_i)$ where f is some nasty function

Let $x_i = \bar{x} + x_i'$, $y_i = \bar{y} + y_i'$

We shall expand f in a 2-D

Taylor series (Note: the T.S. can be generalized to n-dimensions)

So:

$$f(x_i, y_i) = f(\bar{x}, \bar{y}) + x_i' \frac{\partial f}{\partial x}\Big|_{\bar{x},\bar{y}} + y_i' \frac{\partial f}{\partial y}\Big|_{\bar{x},\bar{y}} + O(x_i'^2, y_i'^2, x_i' y_i')$$

(Ignore 2<u>nd</u> order terms!)

$$\therefore \bar{z} \equiv \frac{1}{N}\sum_{i=1}^{N} f(x_i, y_i)$$

$$\approx f(\bar{x}, \bar{y}) + \frac{\partial f}{\partial x}\Big|_{\bar{x},\bar{y}} \cancel{\frac{1}{N}\sum_{i=1}^{N} x_i'} + \frac{\partial f}{\partial y}\Big|_{\bar{x},\bar{y}} \cancel{\frac{1}{N}\sum_{i=1}^{N} y_i'}$$

sum to zero!

$$= f(\bar{x}, \bar{y})$$

The <u>error</u> is $O(x_i'^2, y_i'^2, x_i' y_i' \cdot \underset{\uparrow \atop \text{dyadic}}{\nabla\nabla f})$

Similarly, we get for the variance:

$$s_z^2 \cong \left(\frac{\partial f}{\partial x}\bigg|_{\bar{x},\bar{y}}\right)^2 s_x^2 + \left(\frac{\partial f}{\partial y}\bigg|_{\bar{x},\bar{y}}\right)^2 s_y^2$$

$$+ 2\left(\frac{\partial f}{\partial x}\bigg|_{\bar{x},\bar{y}}\right)\left(\frac{\partial f}{\partial y}\bigg|_{\bar{x},\bar{y}}\right) s_{xy}^2$$

again providing that the higher order terms can be neglected.

This formula reduces to the earlier ones for addition, subtr., mult. & div.

Only for add^n & subtr. is it exact because $\nabla\nabla f \equiv 0$ for this case and the higher order terms vanish.

Let's generalize this result to an n-dimensional (n-variable) system

Let $\underset{\sim}{x} = x_1, x_2, \ldots, x_n$

$z = f(\underset{\sim}{x})$

we need to find the variance of each variable x_1, x_2, etc. and the covariance of all pairs $s^2_{x_1 x_2}$, etc.

We can define the matrix of variance & covariance

$$V_{ij} = \frac{N}{N-1}\left(\overline{x_i x_j} - \bar{x}_i \bar{x}_j\right)$$

↑ number of elements contributing to each x_i, x_j

Now if $\frac{V_{ii}}{\bar{x}_i^{2}} \ll 1$ for all i

(this corresponds to a small error in each variable, e.g. $\frac{s_x^2}{\bar{x}^2} \ll 1$, $\frac{s_y^2}{\bar{y}^2} \ll 1$, etc.)

And if we define:

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial x_3}, \ldots \frac{\partial f}{\partial x_n}\right)$$

Then we get the result:

$$s_z^2 = (\underset{\sim}{\nabla} f) \; \underset{\approx}{V} \; (\underset{\sim}{\nabla} f)^T$$

Which gives us an easy way of calculating the variance in $z = f(\underset{\sim}{x})$!

Note: in this derivation the subscripts i and j denote different variables entirely, not different measurements contributing to the same average.

Let's look at the example $f(\underset{\sim}{x}) = x_1 x_2$

Let $x_1 \equiv x$, $x_2 \equiv y$

$$\therefore \; V_{ij} = \frac{N}{N-1}\left(\overline{x_i x_j} - \bar{x}_i \bar{x}_j\right)$$

so $$V_{xx} = \frac{N}{N-1}\left(\overline{xx} - \bar{x}\,\bar{x}\right)$$

$$= \frac{1}{N-1} \sum^{N} \left(x^2 - \bar{x}^2\right) = s_x^2$$

$$V_{xy} = \frac{N}{N-1}\left(\overline{xy} - \bar{x}\,\bar{y}\right) = \frac{1}{N-1}\sum^{N}\left(xy - \bar{x}\,\bar{y}\right)$$

$$\equiv S_{xy}^2$$

$$\underset{\approx}{V} = \begin{pmatrix} S_x^2 & S_{xy}^2 \\ S_{xy}^2 & S_y^2 \end{pmatrix}$$

$$\underset{\sim}{\nabla} f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right) \equiv (y, x)$$

$$\underset{\sim}{\nabla} f \ \underset{\approx}{V} \left(\underset{\sim}{\nabla} f\right)^T = (y \quad x)\ \underset{\approx}{V} \begin{pmatrix} y \\ x \end{pmatrix}$$

$$= \left(y S_x^2 + x S_{xy}^2,\ y S_{xy}^2 + x S_y^2\right)\begin{pmatrix} y \\ x \end{pmatrix}$$

$$= y^2 S_x^2 + xy S_{xy}^2 + xy S_{xy}^2 + x^2 S_y^2$$

$$= y^2 S_x^2 + x^2 S_y^2 + 2xy\, S_{xy}^2$$

So

$$\frac{S_z^2}{(xy)^2} = \frac{S_x^2}{x^2} + \frac{S_y^2}{y^2} + 2\,\frac{S_{xy}^2}{xy}$$

!

Finally, we examine vector functions of vector variables:

Let $\underset{\sim}{z} = \underset{\sim}{f}(\underset{\sim}{x})$ be a vector function of $\underset{\sim}{x}$. We define:

$$\underset{\sim}{\mu}_z \equiv E(\underset{\sim}{z}), \quad \underset{\sim}{\mu}_x = E(\underset{\sim}{x})$$

and the matrix of covariance for each:

$$\underset{\approx}{\Sigma}_x^2 \equiv E\left[(\underset{\sim}{x} - \underset{\sim}{\mu}_x)(\underset{\sim}{x} - \underset{\sim}{\mu}_x)^T\right]$$

$$\underset{\approx}{\Sigma}_z^2 = E\left[(\underset{\sim}{z} - \underset{\sim}{\mu}_z)(\underset{\sim}{z} - \underset{\sim}{\mu}_z)^T\right]$$

we have taken $\underset{\sim}{z}$ and $\underset{\sim}{x}$ to be column vectors.

Suppose we know $\underset{\sim}{x}$ and $\underset{\approx}{\Sigma}_x^2$. We wish to calculate $\underset{\approx}{\Sigma}_z^2$.

We do a multivariable Taylor series expansion of $\underset{\sim}{z}$ about $\underset{\sim}{\mu}_x$:

$$\underset{\sim}{z} = \underset{\sim}{f}(\underset{\sim}{x}) = \underset{\sim}{f}(\underset{\sim}{\mu}_x) + \underset{\sim}{\nabla}\underset{\sim}{f}\Big|_{\underset{\sim}{\mu}_x} \cdot (\underset{\sim}{x} - \underset{\sim}{\mu}_x)$$
$$+ \tfrac{1}{2}\, \underset{\sim}{\nabla}\underset{\sim}{\nabla}\underset{\sim}{f} : (\underset{\sim}{x} - \underset{\sim}{\mu}_x)(\underset{\sim}{x} - \underset{\sim}{\mu}_x)^T + \dots$$

If the deviation between $\underset{\sim}{x}$ and $\underset{\sim}{\mu}_x$ is **small** (e.g. that $\underset{\approx}{\Sigma}_x^2$ is small), or $\underset{\sim}{f}(\underset{\sim}{x})$ is **linear** in $\underset{\sim}{x}$, then we can ignore the quadratic or higher order terms in the Taylor series!

Thus:

$$\underset{\sim}{z} \cong \underset{\sim}{f}(\underset{\sim}{\mu}_x) + \underset{\sim}{\nabla}\underset{\sim}{f}\Big|_{\underset{\sim}{\mu}_x} \cdot (\underset{\sim}{x} - \underset{\sim}{\mu}_x)$$

and thus:

$$\underset{\sim}{\mu}_z \equiv E(\underset{\sim}{z}) \cong \underset{\sim}{f}(\underset{\sim}{\mu}_x) + \underset{\sim}{\nabla}\underset{\sim}{f}\Big|_{\underset{\sim}{\mu}_x} \cdot E(\underset{\sim}{x} - \underset{\sim}{\mu}_x)$$

but $E(\underset{\sim}{x} - \underset{\sim}{\mu}_x) \equiv 0$ by definition!

Thus $\underset{\sim}{\mu}_z \cong \underset{\sim}{f}(\underset{\sim}{\mu}_x)$ + quadratic terms!

Putting this together, we get:

$$\underset{\sim}{z} - \underset{\sim}{\mu}_z \approx \underset{\sim}{\nabla} \underset{\sim}{f}\Big|_{\underset{\sim}{\mu}_x} (\underset{\sim}{x} - \underset{\sim}{\mu}_x)$$

So:

$$\underset{\approx}{\Sigma}_z^2 \equiv E\left((\underset{\sim}{z} - \underset{\sim}{\mu}_z)(\underset{\sim}{z} - \underset{\sim}{\mu}_z)^T \right)$$

$$\cong E\left[(\underset{\sim}{\nabla} \underset{\sim}{f})\ (\underset{\sim}{x} - \underset{\sim}{\mu}_x)(\underset{\sim}{x} - \underset{\sim}{\mu}_x)^T (\underset{\sim}{\nabla} \underset{\sim}{f})^T \right]$$

$$= (\underset{\sim}{\nabla} \underset{\sim}{f})\ E\left((\underset{\sim}{x} - \underset{\sim}{\mu}_x)(\underset{\sim}{x} - \underset{\sim}{\mu}_x)^T \right) (\underset{\sim}{\nabla} \underset{\sim}{f})^T$$

$$= (\underset{\sim}{\nabla} \underset{\sim}{f})\ \underset{\approx}{\Sigma}_x^2\ (\underset{\sim}{\nabla} \underset{\sim}{f})^T$$

<u>provided</u> that $\frac{1}{2} \underset{\sim}{\nabla} \underset{\sim}{\nabla} \underset{\sim}{f} : \underset{\approx}{\Sigma}_x^2 \ll \underset{\sim}{f}$

so that we can neglect the quadratic term in the Taylor series expansion.

We will use this formulation again in calculating linear regression error!

Ok, let's review error calc

Sample mean $\bar{x} \equiv \frac{1}{N} \sum x_i$

population mean $\mu \equiv E(X)$
(or $E(\bar{x})$)

Sample variance

$$s_x^2 = \frac{1}{N-1} \sum (x_i - \bar{x})^2$$

$$= \frac{1}{N-1} \left(\sum x_i^2 - N \bar{x}^2 \right)$$

Pop. variance $\sigma^2 = E(s_x^2)$

st dev = σ ($\sqrt{\ }$ of variance)

Matrix of covariance:

$$\underset{\sim}{V} = \begin{bmatrix} s_{x_1}^2 & s_{x_1 x_2}^2 & \cdots \\ \vdots & \ddots & \vdots \\ \vdots & \cdots & s_{x_n}^2 \end{bmatrix}$$

where $s_{x_1 x_2}^2 = \frac{1}{N-1} (x_1 - \bar{x}_1)(x_2 - \bar{x}_2)$

where x_1, x_2 are dif types of meas (say, apples & oranges)

$$\underset{\sim}{\Sigma}_x^2 = E(\underset{\sim}{V})$$

Error propagation:

If $y = c_1 x_1 + c_2 x_2$

then $\sigma_y^2 = c_1^2 \sigma_{x_1}^2 + c_2^2 \sigma_{x_2}^2 + 2 c_1 c_2 \underset{\substack{\uparrow \\ \text{covariance}}}{\sigma_{x_1 x_2}^2}$

If $y = x_1 x_2$

then $\dfrac{\sigma_y^2}{y^2} \underset{\substack{\uparrow \\ \text{approx!}}}{\cong} \dfrac{\sigma_{x_1}^2}{x_1^2} + \dfrac{\sigma_{x_2}^2}{x_2^2} + 2 \dfrac{\sigma_{x_1 x_2}^2}{x_1 x_2}$

↳ ignore higher order terms

In general, if

$y = f(\underset{\sim}{x})$

Then $\sigma_y^2 \cong \underset{\sim}{\nabla} f \, \underset{\sim}{\Sigma}_x^2 (\underset{\sim}{\nabla} f)^T$

For a vector function:

$$\underset{\sim}{y} = \underset{\sim}{f}(\underset{\sim}{x})$$

we get:

$$\underset{\approx}{\Sigma}^2_{\underset{\sim}{y}} \cong (\underset{\sim}{\nabla} \underset{\sim}{f}) \underset{\approx}{\Sigma}^2_x (\underset{\sim}{\nabla} \underset{\sim}{f})^T$$

$$\text{where } \underset{\sim}{\nabla} \underset{\sim}{f} \equiv \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

OK, given all this, what do we use it for?

— Usually hypothesis testing!

⇒ Do measurements match theory within error? To what conf. level,

⇒ Is a disease (say cancer) assoc. w/ diet? Lifestyle?

⇒ Is a new drug regimen superior to standard treatment? Placebo?

We usually test this by examining the Null Hypothesis

What is the prob. that the meas. mean/response/value, etc. is really from the same dist?

— If prob of null hyp. H_0 < threshold value (usually 0.05) reject null hyp. & accept alt hyp.

How do we do this? Suppose we know pop. mean & S.D., What is prob. meas not from a population?

Form Z-statistic

$$Z = \frac{x - \mu}{\sigma}$$

For multivariable system: col vector

$$Z^2 = (\underset{\sim}{x} - \underset{\sim}{\mu})^T \underset{\sim}{\Sigma}_x^{-2} (\underset{\sim}{x} - \underset{\sim}{\mu})$$

$\rightarrow$ inverse of $\underset{\sim}{\Sigma}_x^2$

What is the level of significance?

α	conf interval	z
0.10	90%	±1.645
0.05	95%	±1.96 ($\sim 2\sigma$ level)
0.01	99%	±2.575
0.005	99.5%	±2.81
0.001	99.9%	±3.27

Ok, how can we use this?
Suppose we want to test a theory: Stokes sedimentation velocity

At low Re,

$$U_s = \frac{2}{9}\frac{\Delta\rho g a^2}{\mu}$$

Lets test this!

Let $$y = \frac{U_s}{\frac{\Delta\rho g a^2}{\mu}} = f(\underset{\sim}{x})$$

We measure each of these values!

Say $x_1 = U_s$, $x_2 = \Delta g$, $x_3 = g$, etc.

We figure out uncertainty in each!

$$\sigma_{U_s}^2, \quad \sigma_a^2, \quad \text{etc.}\ldots$$

$$\underset{\sim}{\Sigma}_x^2 = \begin{bmatrix} \sigma_{U_s}^2 & & 0 \\ & \ddots & \\ 0 & & \sigma_\mu^2 \end{bmatrix}$$

(assume indep)

$$\underset{\sim}{\nabla} f = \left(\frac{\partial f}{\partial U_s}, \frac{\partial f}{\partial \Delta g}, \ldots \right)$$

⇒ take numerically or analytically

$$\sigma_y^2 \approx \underset{\sim}{\nabla} f \ \underset{\sim}{\Sigma}_x^2 \left(\underset{\sim}{\nabla} f \right)^T$$

form z-statistic $z = \dfrac{y - \frac{2}{g}}{\sigma_y}$

Usually, det. if "agrees to within 2σ" — or not!

Note: random chance will cause it to be outside 2σ interval 5% of time!

Actually, it's more complex than this! Usually the # meas. is fairly small. Thus, we don't know σ, instead have est s_x

We can form the t-statistic

$$t = \frac{y-\mu}{s_x}$$

If the sample size is small, t is not normally distributed, This is because s_x is a random variable too!

There is a sig chance that S_x is too small, which makes t too big => increase prob of tail of dist

Normally, prob of $|Z| > 3$ is very small — but if s_x is underestimated, t can easily be > 3!

Run Example

t-test has interesting history: May know as "student t-test" or "Student's t-test"

=> Really should be Gosset t-test — was a brewer working for Guinness, had corporate freeze on publication. Had to publish anonomously under name "Student" in '08

Ok, what does this mean?

If we know σ, 95% meas within $\pm 2\sigma$ of mean

1 deg freedom

If est s_x from 2 meas,

95% prob within $\pm 12.7 s_x$!

If $n > 5$ and look at 1σ limit, nearly identical — only affects tails, at small n

Ok, what about other tests?

Suppose we have 2 samples (say, response from 2 treatments) — is the response any dif?

$\Rightarrow$ Calc. $\bar{x}_1$, $\bar{x}_2$

var: $s_{x_1}^2$, $s_{x_2}^2$

Null Hyp : $\Delta X = \bar{X}_1 - \bar{X}_2 = 0$

What is the SD in Δx?

$$\Delta x = \bar{x}_1 - \bar{x}_2$$

$$\therefore S^2_{\Delta x} = \underbrace{S^2_{\bar{x}_1} + S^2_{\bar{x}_2}}_{\text{var}_2 \text{ in } \underline{\underline{\text{mean}}}}$$

$$S^2_{x_1} = \frac{S^2_{x_1}}{n_1}, \quad S^2_{\bar{x}_2} = \frac{S^2_{x_2}}{n_2}$$

we **define** $t = \dfrac{\Delta x}{S_{\Delta x}}$

If > threshold, **reject** null hyp.

If n_1, n_2 **large** 95% conf $|t| \gtrsim 2$

for smaller n_1, n_2 use t-tables

Actually, easier to use matlab $f^{\underline{n}}$ (stat. toolbox)

Common functions

cdf, normcdf, norminv

⇒ based on Gaussian dist.

tcdf - t cum dist $f^{\underline{n}}$

$\Rightarrow$ can do hyp. testing w/
ztest, ttest, ttest2
↑
2 samples

Ok, what if we want to test *many* samples? suppose we look to see if there is a difference among *many groups*

We *could* do a t-test between each sample in a combinatoric fashion, but you are *guaranteed* a false positive! In text, talks about looking at relation between 8 groups => 7 x 8/2 = 28 combinations

Even *random chance* at 95% conf. level would lead to at least 1 false pos.

A better way is ANOVA, analysis of Variance

Basically, you look at the variance of "supergroup" and variance of all the subgroups.

If the variance of supergroup is larger than expected from subgroups, than at least one of the subgroups doesn't belong.

This ratio yields the F-statistic which has a dist. depending on the number of degrees of freedom.

Matlab has a canned routine anova1(A) which does this to a matrix of values.

Most of these formulas require independent normally distributed data

If the data is not indep, $s_{\bar{x}}$ will be underestimated! (Effectively, n is reduced)

If the data isn't normally dist, sometimes you can fix it!

Example: Dist of particle size

- often have # density of small particles >> larger ones
- could look at radius vs mass fraction, not # of particles
- Also - log normal dist - work w/ log of variable rather than direct meas
- plot dist & see what works!

Linear Regression

In HW problem, wrote routine to fit a parabola to 3 points. In this case there was only one answer.

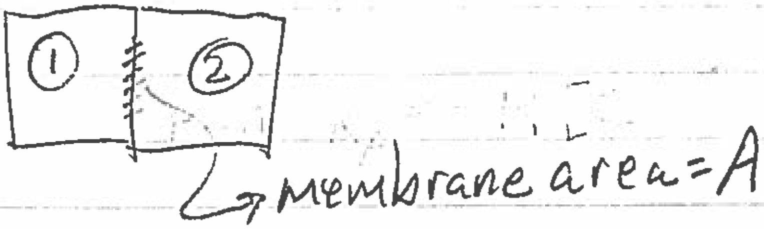
Often in data analysis you want to fit a curve to many data points

How do we do this? Use (usually) linear regression

This does not mean that we fit data with a line, rather that we use an algorithm which reduces the problem to a set of linear equations!

Let's look at a simple problem: measure mass transfer coefficients

Suppose we have a membrane between two reservoirs



We start with some concentration in reservoir ① of C_0 and we assume that reservoir ② is maintained at a constant C_{eq}. If the volume of reservoir ① is V, then:

$$V \frac{\partial C}{\partial t} = -Ah(C - C_{eq})$$

where $C\Big|_{t=0} = C_0$

We can solve this equation:

$$C = C_{eq} + (C_0 - C_{eq})\, e^{-\left(\frac{htA}{V}\right)}$$

We wish to determine h by measuring C as a function of time

In order for us to use linear regression, the model must be linear in the modelling parameter,

In this case, the modelling parameter is $\underline{\underline{h}}$. We can rewrite the eq'n:

$$(c - c_{eq}) = (c_0 - c_{eq})\, e^{-\frac{hA}{V}t}$$

$$\ln(c - c_{eq}) = \ln(c_0 - c_{eq}) - \frac{hA}{V}t$$

So if we plot $\ln(c - c_{eq})$ vs $\frac{At}{V}$ we get a line w/ intercept of $\ln(c_0 - c_{eq})$ and a slope of h!

How do we get the best fit line?

We look at the deviation of the points from the line, and try to minimize this in some way.

Let's take:

$$y \equiv \ln(c - c_{eq}), \quad a \equiv \frac{Ah}{V},$$

$$b \equiv \ln(c_0 - c_{eq})$$

We want to fit a series of points (t_i, y_i) by the model $y = at + b$

We shall use the method of least squares. We form the sum:

$$\text{Sum} = \sum_{i=1}^{N} \left(y_i - (at_i + b)\right)^2$$

which is the sum of the squared distance between datapt. and model in the y direction.

We pick a & b so that this is a minimum

We let:

$$\left.\begin{array}{l} \dfrac{\partial \text{sum}}{\partial a} = 0 \\[2ex] \dfrac{\partial \text{sum}}{\partial b} = 0 \end{array}\right\} \text{2 eq}^{n}\text{s for } a, b\,!$$

$$\frac{\partial \text{sum}}{\partial a} = \sum_{i=1}^{N} -2\left(y_i - (at_i + b)\right) t_i$$

$$= \sum_{i=1}^{N} -2y_i t_i + 2a \sum_{i=1}^{N} t_i^2 + 2b \sum_{i=1}^{N} t_i = 0$$

and:

$$\frac{\partial \text{sum}}{\partial b} = \sum_{i=1}^{N} -2\left(y_i - (at_i + b)\right)$$

$$= \sum_{i=1}^{N} -2y_i + 2a \sum_{i=1}^{N} t_i + 2Nb = 0$$

This is a pair of linear equations for a & b!

We have the solution:

$$a\overline{t^2} + b\overline{t} = \overline{yt} \qquad \left(\text{let } \overline{x} \equiv \frac{1}{N}\sum_{i=1}^{N} x_i\right)$$

$$a\overline{t} + b = \overline{y}$$

$$\therefore \quad a = \frac{\overline{yt} - \overline{y}\,\overline{t}}{\overline{t^2} - \overline{t}^2}$$

$$\text{and} \quad b = \overline{y} - a\overline{t}$$

which can be used to get the mass transfer coefficient!

This is not the only way to approach the problem. Let's look at it as a system of equations!

We are trying to satisfy:

$$
\begin{aligned}
y_1 &= a\,t_1 + b \\
y_2 &= a\,t_2 + b \\
&\vdots \\
y_N &= a\,t_N + b
\end{aligned}
$$

Let's write this in <u>matrix form</u>:

$$
\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix} \approx \begin{pmatrix} t_1 & 1 \\ t_2 & 1 \\ \vdots & \vdots \\ t_N & 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix}
$$

we shall use $\underset{\sim}{X}$ to be the vector containing the modelling parameters:

$$
X_1 \equiv a \qquad X_2 \equiv b
$$

If we let the matrix of modelling functions be $\underset{\approx}{A}$ (e.g. $A_{11} = t_1$, etc.)

then we want to satisfy:

$$\underset{\sim}{y} \approx \underset{\approx}{A}\,\underset{\sim}{x}$$

Actually, we want to minimize the residual:

$$\underset{\sim}{r} = \underset{\sim}{y} - \underset{\approx}{A}\,\underset{\sim}{x}$$

Or, for least squares, adjust $\underset{\sim}{x}$ s.t.:

$$\underset{\underset{\text{Euclidean norm}}{\uparrow}}{\|\underset{\sim}{r}\|_2} \equiv \|\underset{\sim}{y} - \underset{\approx}{A}\,\underset{\sim}{x}\|_2 \quad \text{is as small as possible}$$

We want to pick $\underset{\sim}{x}$ s.t.

$$\underset{\sim}{r}^T\underset{\sim}{r} \equiv (\underset{\sim}{y} - \underset{\approx}{A}\,\underset{\sim}{x})^T(\underset{\sim}{y} - \underset{\approx}{A}\,\underset{\sim}{x})$$

is a minimum.

This is *exactly* *equivalent* to what we did before!

Let's multiply this out:

$$r^2 \equiv \underset{\sim}{y}^T \underset{\sim}{y} - \underset{\sim}{y}^T \underset{\approx}{A} \underset{\sim}{x} - \underset{\sim}{x}^T \underset{\approx}{A}^T \underset{\sim}{y} + \underset{\sim}{x}^T \underset{\approx}{A}^T \underset{\approx}{A} \underset{\sim}{x}$$

We take the *gradient* of this w.r.t. $\underset{\sim}{x}$ and set it equal to zero!

$$\underset{\sim}{\nabla}_{\underset{\sim}{x}} r^2 = 0 = -2 \underset{\approx}{A}^T \underset{\sim}{y} + 2 \underset{\approx}{A}^T \underset{\approx}{A} \underset{\sim}{x} = 0$$

Thus we have the problem:

$$\underset{\approx}{A}^T \underset{\approx}{A} \underset{\sim}{x} = \underset{\approx}{A}^T \underset{\sim}{y} \qquad !$$

This gives us a set of $\underline{\underline{n}}$ eq'ns for the n unknowns in $\underset{\sim}{x}$!

These are known as the normal equations.

Linear Regression Error

Let's apply all this back to linear regression. Suppose we have some data and we are trying to fit it, and calc. the uncertainty in the modelling parameters.

We measur a lead weight eas. its pos'n a

We get

skip this

(go straight to vector approach)

t_i (s	
0	
1	1.8
2	20.7
3	15.9
4	2.0

We fit this to the equation:

$$h = h_0 + v_0 t - \frac{1}{2} g t^2$$

We want to both calculate g, <u>and</u> determine the uncertainty in g.

First let's get g. We have the problem:

$$h_i = (1)\underset{\substack{\uparrow\\ x_1}}{h_0} + (t)\underset{\substack{\uparrow\\ x_2}}{V_0} + \left(-\tfrac{1}{2}t^2\right)\underset{\substack{\uparrow\\ x_3}}{g}$$

So:

$$\underset{\approx}{A}\nearrow\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & -.5 \\ 1 & 2 & -2 \\ 1 & 3 & -\frac{9}{2} \\ 1 & 4 & -8 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \overset{\swarrow \underset{\sim}{h}}{\begin{pmatrix} 0.3 \\ 14.8 \\ 20.7 \\ 15.9 \\ 2.0 \end{pmatrix}}$$

Using the normal eq'ns we get:

$$\underset{\approx}{A}^T\underset{\approx}{A}\,\underset{\sim}{x} = \underset{\approx}{A}^T\underset{\sim}{h}$$

$$\underset{\sim}{x} = \left[\underset{\approx}{A}^T\underset{\approx}{A}\right]^{-1}\underset{\approx}{A}^T\underset{\sim}{h}$$

Or numerically:

$$\underset{\sim}{x} = \begin{pmatrix} .886 & .257 & -.086 & -.143 & .086 \\ -.771 & .186 & .571 & .386 & -.371 \\ -.286 & .143 & .286 & .143 & -.286 \end{pmatrix} \begin{pmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \end{pmatrix}$$

This yields:

$$x_1 = 0.197 \,, \quad x_2 = 19.7 \quad x_3 = 9.64$$

Now we want to examine the error in the position measurements h_i.

To do this we look at the deviation:

$$S_h^2 \approx \underset{\substack{\uparrow \\ \text{because 3} \\ \text{adjustable} \\ \text{parameters} \\ \text{det. from data}}}{\frac{1}{N-3}} \sum_{i=1}^{N} \Big(h_i - \underset{\substack{\uparrow \\ \text{using fitted} \\ \text{parameters}}}{h}(t_i) \Big)^2$$

The quantity $h_i - h(t_i)$ is just the residual;

$$\underset{\sim}{r} = \underset{\sim}{h} - \underset{\approx}{A}\,\underset{\sim}{x} \quad !$$

$$\therefore \; s_h^2 = \underbrace{\frac{1}{N-3}}_{\text{↳ you should really have more pts for accuracy!}} \|\underset{\sim}{r}\|_2^2 = 0.110$$

Ok, how do we use this to get the error in g (x_3)?

We had:

$$\underset{\sim}{x} = \underbrace{\left\{\left[\underset{\approx}{A}^T\underset{\approx}{A}\right]^{-1}\underset{\approx}{A}^T\right\}}_{\equiv\, \underset{\approx}{K}} \underset{\sim}{h}$$

Thus $x_i \equiv \sum_{j=1}^{N} \underbrace{K_{ij}\; h_j}_{i\text{'th row of } \underset{\approx}{K}\,!}$

We can thus use the formula for a linear combination of random variables!

$$S^2_{x_i} = \sum_{i=1}^{N} K^2_{ij}\, S^2_{h_i} = 0.0315$$

$\uparrow$ these are all the same in this problem!

Thus $g = 9.64 \pm 0.18$

to the 69% confidence level (1σ)

Often we want to develop correlations for their predictive value, for example what is the expected ht. of the lead wt. at some time t?

$$h_m = x_1 + x_2 t - \tfrac{1}{2} t^2 x_3 \quad !$$

But what is the error? We have a linear combination of x_1, x_2, x_3 each of which have some error.

You might expect that:

$$S_{h_m}^2 = S_{x_1}^2 + (t)^2 S_{x_2}^2 + \left(-\tfrac{1}{2}t^2\right)^2 S_{x_3}^2$$

<u>But</u> this is <u>incorrect</u> !!

The problem is that x_1, x_2, x_3 are <u>not</u> independent! Their covariance is non-zero!

Thus we should have:

$$S_{h_m}^2 = S_{x_1}^2 + (t)^2 S_{x_2}^2 + \left(-\tfrac{1}{2}t^2\right)^2 S_{x_3}^2$$
$$+ 2\,(1)(t)\, S_{x_1 x_2}^2 + 2\,(1)\left(-\tfrac{1}{2}t^2\right) S_{x_1 x_3}^2$$
$$+ 2\,(t)\left(-\tfrac{1}{2}t^2\right) S_{x_2 x_3}^2$$

How do we calculate the covariance?

Recall:

$$x_i \equiv \sum_{j=1}^{N} K_{ij}\, h_j$$

where $\underset{\approx}{K} \equiv \left[\underset{\approx}{A}^T \underset{\approx}{A}\right]^{-1} \underset{\approx}{A}^T$

Thus we let:

$$x_1 = \sum_{j=1}^{N} \alpha_j h_j$$ where α is 1st row of $\underset{\approx}{K}$

and

$$x_2 = \sum_{j=1}^{N} \beta_j h_j$$ where $\underset{\sim}{\beta}$ is the 2nd row of $\underset{\approx}{K}$

Now we need to determine the covariance

$$\sigma^2_{x_1 x_2} = E\left(\left(x_1 - \underset{\uparrow}{\mu_{x_1}}\right)\left(x_2 - \underset{\uparrow}{\mu_{x_2}}\right)\right)$$

value for x_1 & x_2 expected for an infinite sample

$$\mu_{x_1} = \sum_{j=1}^{N} \alpha_j \underset{\uparrow}{\mu_{h_j}}$$

from infinite sample

$$\mu_{x_2} = \sum_{j=1}^{N} \beta_j \mu_{h_j}$$

So:

$$\sigma_{x_1x_2}^2 \equiv E\left(\left(\sum_{j=1}^{N}\alpha_j\left(h_j-\mu_{h_j}\right)\right)\left(\sum_{j=1}^{N}\beta_j\left(h_j-\mu_{h_j}\right)\right)\right)$$

Now if all of the h_j are <u>independent</u> then

$$E\left(\left(h_j-\mu_{h_j}\right)\left(h_i-\mu_{h_i}\right)\right)=\underline{\underline{0}}$$

for $i \neq j$

$$\therefore \sigma_{x_1x_2}^2 = E\left(\sum_{j=1}^{N}\alpha_j\beta_j\left(h_j-\mu_{h_j}\right)^2\right)$$

$$= \sum_{j=1}^{N}\alpha_j\beta_j\,\underset{\uparrow}{\sigma_h^2}$$

provided all the h_j have the same uncertainty

$$\therefore \sigma_{x_1x_2}^2 = \underset{\sim}{\alpha}\,\underset{\sim}{\beta}^T\,\sigma_h^2$$

So for this problem if we let

$$\underset{\sim}{\gamma} \equiv 3^{\underline{rd}} \text{ row of } \underset{\approx}{K}$$

$$x_1 = \underset{\sim}{\alpha}\,\underset{\sim}{h} \ , \quad x_2 = \underset{\sim}{\beta}\,\underset{\sim}{h} \ , \quad x_3 = \underset{\sim}{\gamma}\,\underset{\sim}{h}$$

$$s_h^2 = \frac{1}{N-3}\,\|\underset{\sim}{r}\|_2^2 \ , \quad \underset{\sim}{r} = \underset{\sim}{h} - \underset{\approx}{A}\,\underset{\sim}{X}$$

$$s_{x_1}^2 = (\underset{\sim}{\alpha}\,\underset{\sim}{\alpha}^T)\, s_h^2$$

$$s_{x_2}^2 = (\underset{\sim}{\beta}\,\underset{\sim}{\beta}^T)\, s_h^2 \quad \text{etc.}$$

and:

$$h_m = (1)\,x_1 + (t)\,x_2 + (\tfrac{1}{2}t^2)\,x_3$$

$$\begin{aligned}
\sigma_{h_m}^2 &= (1)^2\sigma_{x_1}^2 + (t)^2\sigma_{x_2}^2 + (-\tfrac{1}{2}t^2)^2\sigma_{x_3}^2 \\
&\quad + 2(1)(t)\,\underset{\sim}{\alpha}\,\underset{\sim}{\beta}^T\sigma_h^2 + 2(1)(-\tfrac{1}{2}t^2)\,\underset{\sim}{\alpha}\,\underset{\sim}{\gamma}^T\sigma_h^2 \\
&\quad + 2(t)(-\tfrac{1}{2}t^2)\,\underset{\sim}{\beta}\,\underset{\sim}{\gamma}^T\sigma_h^2 \\
&= \left\| \left[(1)\,\underset{\sim}{\alpha} + (t)\,\underset{\sim}{\beta} + (-\tfrac{1}{2}t^2)\,\underset{\sim}{\gamma} \right] \right\|_2^2 \sigma_h^2
\end{aligned}$$

If we represent the vector of modelling <u>functions</u> as $\underset{\sim}{a}(t)$, we can collapse this:

$$\underset{\sim}{a}(t) \equiv \left(1, t, -\tfrac{1}{2}t^2\right)$$

Then

$$\sigma_{h_m}^{2} = \|\underset{\sim}{a}\,\underset{\approx}{K}\|_2^{2}\,\sigma_h^{2}$$

Let's generalize the regression analysis we developed above.

We have n observations b_i which we wish to fit with m parameters and modelling functions:

$$b_i \approx \sum_{j=1}^{m} x_j \phi_j(t_i)$$

or :

$$\underset{\sim}{b} \approx \underset{\approx}{A}\,\underset{\sim}{x}$$

where $A_{ij} = \phi_j(t_i)$

We define the residual :

$$\underset{\sim}{r} \equiv \underset{\sim}{b} - \underset{\approx}{A}\,\underset{\sim}{x}$$

We define the best fit modelling parameters $\underset{\sim}{x}$ by :

$$\min_{\underset{\sim}{x}} \|\underset{\sim}{r}\|_2^2 \equiv \min_{\underset{\sim}{x}} \left(\underset{\sim}{r}^T \underset{\sim}{r}\right)$$

$\underset{\sim}{x}$ has the solution :

$$\underset{\approx}{A}^T \underset{\approx}{A}\,\underset{\sim}{x} = \underset{\approx}{A}^T \underset{\sim}{b}$$

or

$$\underset{\sim}{x} = \left[\underset{\approx}{A}^T \underset{\approx}{A}\right]^{-1} \underset{\approx}{A}^T \underset{\sim}{b} \equiv \underset{\approx}{K}\,\underset{\sim}{b}$$

Note that $\underset{\approx}{A}$ is an $n \times m$ matrix, $\underset{\sim}{b}$ is a $n \times 1$ array, $\underset{\sim}{x}$ is a $m \times 1$ array and $\underset{\approx}{K}$ is a $m \times n$ matrix

If we recall that :

$$\sigma_z^2 = E\left\{(z - \mu_z)^2\right\}$$

where $\mu_z = E(z)$

Then the matrix of covariance of the regression parameters is:

$$\underset{\approx}{\Sigma}_x^2 \equiv E\left\{ (\underset{\sim}{x} - \underset{\sim}{\mu}_x)(\underset{\sim}{x} - \underset{\sim}{\mu}_x)^T \right\}$$

substituting $\underset{\sim}{x} = \underset{\approx}{K}\,\underset{\sim}{b}$ and $\underset{\sim}{\mu}_x = \underset{\approx}{K}\,\underset{\sim}{\mu}_b$ we get:

$$\underset{\approx}{\Sigma}_x^2 \equiv \underset{\approx}{K}\; E\left\{ (\underset{\sim}{b} - \underset{\sim}{\mu}_b)(\underset{\sim}{b} - \underset{\sim}{\mu}_b)^T \right\} \underset{\approx}{K}^T$$

$$= \underset{\approx}{K}\; \underset{\approx}{\Sigma}_b^2\; \underset{\approx}{K}^T$$

where $\underset{\approx}{\Sigma}_b^2$ is the matrix of covariance of the observations $\underset{\sim}{b}$.

If the observations are <u>independent</u> then $\underset{\approx}{\Sigma}_b^2$ will be a diagonal matrix.

If all of the observations have the same variance, then:

$$\underset{\approx}{\Sigma}_b^2 = \underset{\approx}{I}\;\sigma_b^2$$

and we get

$$\underset{\approx}{\Sigma}_x^2 = \sigma_b^2 \, \underset{\approx}{K} \, \underset{\approx}{K}^T$$

We may estimate σ_b^2 from:

$$\sigma_b^2 \approx s_b^2 \equiv \frac{1}{n-m} \| \underset{\approx}{A} \underset{\sim}{\hat{x}} - \underset{\sim}{b} \|_2^2$$

$$= \frac{1}{n-m} \underset{\sim}{r}^T \underset{\sim}{r}$$

We may calculate the error in model predictions similarly. Suppose we wish to determine the model value at K times $\underset{\sim}{t}^{(m)}$. Thus:

$$\underset{\sim}{b}^{(m)} = \underset{\approx}{A}^{(m)} \underset{\sim}{\hat{x}}$$

where $A_{ij}^{(m)} = \phi_j(t_i^{(m)})$

Note that $\underset{\approx}{A}^{(m)}$ is a $K \times m$ matrix.

What is the error in $\underset{\sim}{b}^{(m)}$? We have the matrix of covariance:

$$\underset{\approx}{\Sigma}^2_{b^{(m)}} \equiv E\left\{ \left(\underset{\sim}{b}^{(m)} - \underset{\sim}{\mu}_{b^{(m)}}\right)\left(\underset{\sim}{b}^{(m)} - \underset{\sim}{\mu}_{b^{(m)}}\right)^T \right\}$$

$$= \underset{\approx}{A}^{(m)} E\left\{ \left(\underset{\sim}{x} - \underset{\sim}{\mu}_x\right)\left(\underset{\sim}{x} - \underset{\sim}{\mu}_x\right)^T \right\} \underset{\approx}{A}^{(m)T}$$

$$= \underset{\approx}{A}^{(m)} \underset{\approx}{\Sigma}^2_x \underset{\approx}{A}^{(m)T}$$

where $\underset{\approx}{\Sigma}^2_x$ is the matrix of covariance determined before! In general, only the diagonal elements of $\underset{\approx}{\Sigma}^2_{b^{(m)}}$ are of interest:

$$\underset{\sim}{\sigma}^2_{b^{(m)}} = \text{Diag}\left\{ \underset{\approx}{\Sigma}^2_{b^{(m)}} \right\}$$

which yields the error in the model predictions.

As a final note on linear regression, let us examine the case where the variance of $\underset{\sim}{\sigma}^2_b$ is <u>not</u> constant!

In this case we wish to weight the points in our regression analysis differently.

This is <u>weighted</u> linear regression!

Suppose we have the <u>diagonal</u> matrix $\underset{\approx}{\Sigma}_b^2$. The weighted problem is just:

$$\min_{\underset{\sim}{x}} \; (\underset{\approx}{A}\underset{\sim}{x}-\underset{\sim}{b})^T \left(\underset{\approx}{\Sigma}_b^2\right)^{-1} (\underset{\approx}{A}\underset{\sim}{x}-\underset{\sim}{b})$$

$$= \min_{\underset{\sim}{x}} \; \underset{\sim}{r}^T \left(\underset{\approx}{\Sigma}_b^2\right)^{-1} \underset{\sim}{r}$$

where $\left(\underset{\approx}{\Sigma}_b^2\right)^{-1}$ is a diagonal matrix whose elements are just $\frac{1}{\sigma_{b_i}^2}$.

Thus we weight each point by the inverse of the variance of that point. This makes sense. Suppose the reason why σ_{b_i} varied was that m_i points were <u>averaged</u> together to get b_i.

Then we have $\sigma_{b_i}^2 = \frac{\sigma_b^2}{m_i}$ and

thus the above regression yields:

$$\frac{1}{\sigma_b^2}\min_{\underset{\sim}{x}} \underset{\sim}{r}^T \begin{pmatrix} m_1 & & 0 \\ & m_2 & \\ 0 & \ddots & m_n \end{pmatrix} \underset{\sim}{r} \,,$$

or the i^{th} point is counted m_i times. The result for $\underset{\sim}{x}$ would be the same (on average) as if you had used all of the points individually!

OK, what is the formula for $\underset{\sim}{x}$?

Just as before we take the gradient:

$$\underset{\sim}{\nabla}_x \left\{ \underset{\sim}{r}^T \left(\underset{\approx}{\Sigma}_b^2\right)^{-1} \underset{\sim}{r} \right\} = 0$$

which yields:

$$\underset{\approx}{A}^T \left(\underset{\approx}{\Sigma}_b^2\right)^{-1} \underset{\approx}{A}\, \underset{\sim}{x} = \underset{\approx}{A}^T \left(\underset{\approx}{\Sigma}_b^2\right)^{-1} \underset{\sim}{b}$$

or $\underset{\sim}{x} = \underset{\approx}{K}_w \underset{\sim}{b} \equiv \left\{ \underset{\approx}{A}^T \left(\underset{\approx}{\Sigma}_b^2\right)^{-1} \underset{\approx}{A} \right\}^{-1} \underset{\approx}{A}^T \left(\underset{\approx}{\Sigma}_b^2\right)^{-1} \underset{\sim}{b}$

with error:

$$\underset{\approx}{\Sigma}_x^2 = \underset{\approx}{K}_w \underset{\approx}{\Sigma}_b^2 \underset{\approx}{K}_w^T$$

ex 15

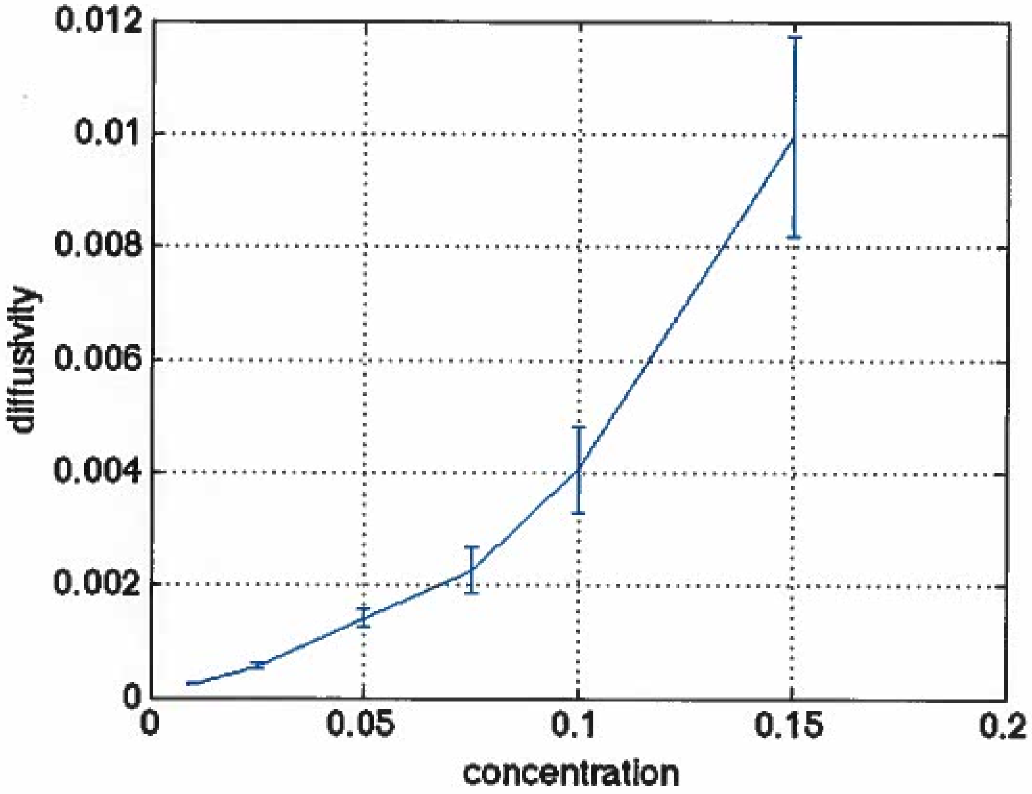
```
clear
echo on
format compact
% In this problem we examine the behavior of a dilute
% suspension of particles which is being sheared.  As the
% suspension is sheared (the motion which is produced
% when a fluid is confined between two concentric cylinders
% and one of them is rotated) the particles will tumble over
% one another.  This will lead to a random walk of the
% particles which can be characterized by a diffusion
% coefficient, much like a molecular diffusivity.  We can
% measure this quantity - the shear-induced self-diffusivity -
% by examining the random walk of a single tracer particle in
% a suspension of otherwise identical particles.  Because the
% random motion is due to the interaction of the tracer with
% other particles, the diffusivity will be identically zero if the
% concentration of the other particles is zero.
pause
% In our lab we measured the shear-induced self-diffusivity in
% the dilute limit.  If the particles are perfect spheres, theory
% suggests that the leading order term in the diffusivity
% should go as c^2 where c is the concentration.  Other
% models suggest that if the spheres are not perfect, the
% diffusivity should be proportional to c to leading order.  More
% recently, models have suggested that even for smooth spheres the
% presence of bounding walls gives rise to an O(c) diffusivity.
%
% In this example we examine data to determine which model best
% describes the experiments:
%
% c       diffusivity     error
% 0.01    2.37e-4         2.2e-5
% 0.025   5.63e-4         6.5e-5
% 0.05    1.42e-3         1.6e-4
% 0.075   2.27e-3         4.0e-4
% 0.10    4.06e-3         7.6e-4
% 0.15    9.96e-3         1.8e-3
%
% The errors given above are the one standard deviation errors
% in the measured diffusivities calculated from the statistics
% governing the measurement process.
pause
% Using this data, we wish to fit a constitutive equation for the
% diffusivity of the form:
%
%         Diffusivity = c * x(1) + c^2 * x(2) + c^3 * x(3)
%
% using weighted linear regression, and determine the error
% in each of the fitting coefficients.  In particular, we need to
% determine if the difference between the O(c) coefficient x(1) and
% zero is statistically significant.

% First we put in the data:
c=[.01,.025,.05,.075,.1,.15]';
```

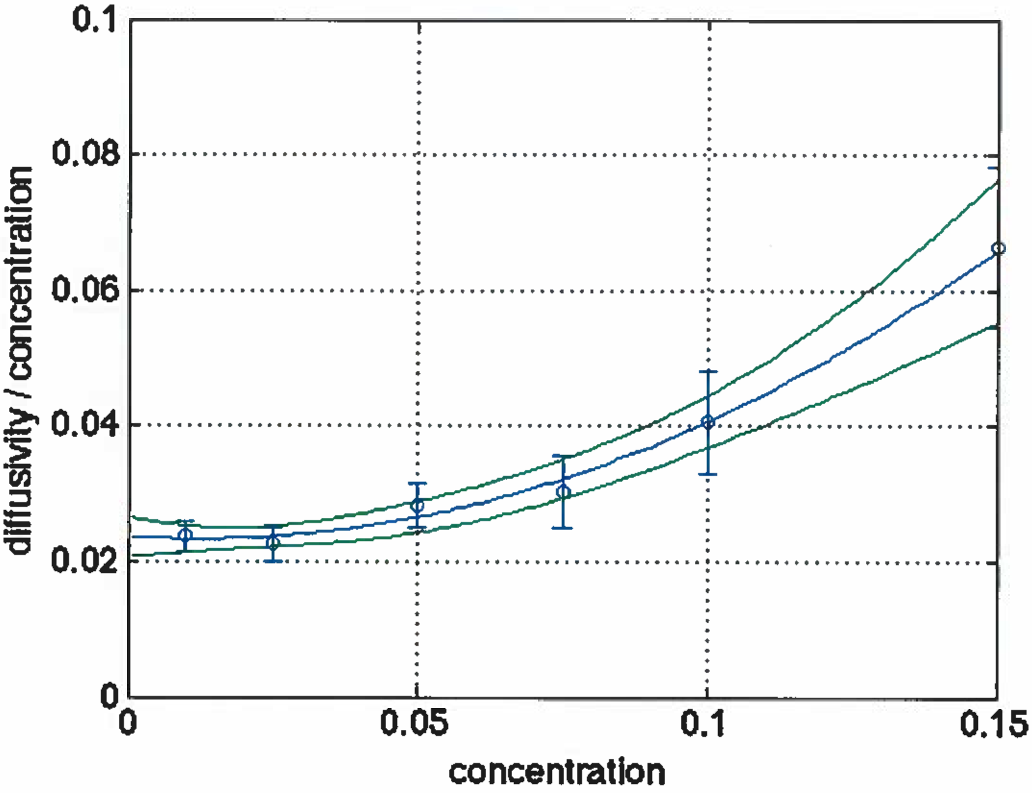
```
diff=[2.37e-4,5.63e-4,1.4153e-3,2.27e-3,4.055e-3,9.965e-3]';
sigma=[2.2e-5,6.491228e-5, 1.617643e-4,3.98208e-4,7.605929e-4,1.775587e-3]';
pause
%Let's plot this data up:
figure(1)
errorbar (c,diff,sigma)
set(gca,'FontSize',18)
xlabel('concentration')
ylabel('diffusivity')
grid on
%As you can see, it certainly goes to zero as the
%concentration goes to zero.
pause
%The asymptotic behavior can be visualized better
%by plotting diff/c rather than just diff:
figure(2)
errorbar(c,diff./c,sigma./c,sigma./c,'o')
set(gca,'FontSize',18)
xlabel('concentration')
ylabel('diffusivity / concentration')
grid on
hold on
%Here there is clearly a non-zero asymptotic value
%of diff/c as c goes to zero.  This is what we are
%trying to capture.
pause
%OK, now for the weighted linear regression.  The
%unweighted linear regression problem is of the
%form:
%
%   min || A x - b ||
%
%In the weighted regression problem we weight each
%data point by the inverse of its variance when forming
%the sum of the squares of the deviation.  Let's do this:
%
%First we set up the matrix of modeling functions.  We
%have a linear term, a quadratic term, and a cubic term:
a=[c,c.^2,c.^3]
pause
%This is the weighting function:
vardiff=diag(sigma.^2);
weight=inv(vardiff)
pause
%We now define a matrix kw:
kw=inv(a'*weight*a)*a'*weight
%which can be used the conventional way:
x=kw*diff
%which has a lead coefficient different from zero.
pause
%Now for the error:
varx=kw*vardiff*kw'
xerror=diag(varx).^.5;
%So the coefficients and errors are:
[x,xerror]
%and in particular,
x(1)/xerror(1)
%So the order c coefficient is statistically different from
%zero (about 8 standard deviations).
pause
%Let's finish off with a bit of graphics.  Let's plot up
%the model predictions together with its uncertainty.
cp=[.001:.001:.15]';
ap=[cp,cp.^2,cp.^3];
dp=ap*x;
perror=diag(ap*varx*ap').^.5;
figure(2)
plot(cp,dp./cp,'b')
plot(cp,(dp+perror)./cp,'g')
plot(cp,(dp-perror)./cp,'g')
axis([0 .15 0 .1]);
%where we have plotted the 1-sigma confidence interval
%of the model function, based on the given standard deviations.
hold off
echo off
```

```
format compact
% In this problem we examine the behavior of a dilute
% suspension of particles which is being sheared.  As the
% suspension is sheared (the motion which is produced
% when a fluid is confined between two concentric cylinders
% and one of them is rotated) the particles will tumble over
% one another.  This will lead to a random walk of the
% particles which can be characterized by a diffusion
% coefficient, much like a molecular diffusivity.  We can
% measure this quantity - the shear-induced self-diffusivity -
% by examining the random walk of a single tracer particle in
% a suspension of otherwise identical particles.  Because the
% random motion is due to the interaction of the tracer with
% other particles, the diffusivity will be identically zero if the
% concentration of the other particles is zero.
pause
% In our lab we measured the shear-induced self-diffusivity in
% the dilute limit.  If the particles are perfect spheres, theory
% suggests that the leading order term in the diffusivity
% should go as c^2 where c is the concentration.  Other
% models suggest that if the spheres are not perfect, the
% diffusivity should be proportional to c to leading order.  More
% recently, models have suggested that even for smooth spheres the
% presence of bounding walls gives rise to an O(c) diffusivity.
%
% In this example we examine data to determine which model best
% describes the experiments:
%
% c        diffusivity     error
% 0.01     2.37e-4         2.2e-5
% 0.025    5.63e-4         6.5e-5
% 0.05     1.42e-3         1.6e-4
% 0.075    2.27e-3         4.0e-4
% 0.10     4.06e-3         7.6e-4
% 0.15     9.96e-3         1.8e-3
```

```
%
% The errors given above are the one standard deviation errors
% in the measured diffusivities calculated from the statistics
% governing the measurement process.
pause
% Using this data, we wish to fit a constitutive equation for the
% diffusivity of the form:
%
%         Diffusivity = c * x(1) + c^2 * x(2) + c^3 * x(3)
%
% using weighted linear regression, and determine the error
% in each of the fitting coefficients.  In particular, we need to
% determine if the difference between the O(c) coefficient x(1) and
% zero is statistically significant.
% First we put in the data:
c=[.01,.025,.05,.075,.1,.15]';
diff=[2.37e-4,5.63e-4,1.4153e-3,2.27e-3,4.055e-3,9.965e-3]';
sigma=[2.2e-5,6.491228e-5, 1.617643e-4,3.98208e-4,7.605929e-4,1.775587e-3]';
pause
%Let's plot this data up:
figure(1)
errorbar (c,diff,sigma)
set(gca,'FontSize',18)
xlabel('concentration')
ylabel('diffusivity')
grid on
%As you can see, it certainly goes to zero as the
%concentration goes to zero.
pause
%The asymptotic behavior can be visualized better
%by plotting diff/c rather than just diff:
figure(2)
errorbar(c,diff./c,sigma./c,sigma./c,'o')
set(gca,'FontSize',18)
xlabel('concentration')
ylabel('diffusivity / concentration')
grid on
hold on
%Here there is clearly a non-zero asymptotic value
%of diff/c as c goes to zero.  This is what we are
%trying to capture.
pause
%OK, now for the weighted linear regression.  The
%unweighted linear regression problem is of the
%form:
%
%   min || A x - b ||
%
%In the weighted regression problem we weight each
%data point by the inverse of its variance when forming
%the sum of the squares of the deviation.  Let's do this:
%
%First we set up the matrix of modeling functions.  We
%have a linear term, a quadratic term, and a cubic term:
a=[c,c.^2,c.^3]
a =
    0.0100    0.0001    0.0000
    0.0250    0.0006    0.0000
    0.0500    0.0025    0.0001
    0.0750    0.0056    0.0004
    0.1000    0.0100    0.0010
    0.1500    0.0225    0.0034
pause
%This is the weighting function:
vardiff=diag(sigma.^2);
weight=inv(vardiff)
weight =
   1.0e+09 *
    2.0661         0         0         0         0         0
         0    0.2373         0         0         0         0
         0         0    0.0382         0         0         0
         0         0         0    0.0063         0         0
         0         0         0         0    0.0017         0
         0         0         0         0         0    0.0003
pause
%We now define a matrix kw:
kw=inv(a'*weight*a)*a'*weight
kw =
   1.0e+04 *
    0.0115    0.0009   -0.0005   -0.0002   -0.0000    0.0000
   -0.3532    0.0251    0.0474    0.0126    0.0017   -0.0038
    2.1167   -0.3116   -0.3388   -0.0639    0.0127    0.0472
%which can be used the conventional way:
x=kw*diff
x =
    0.0237
   -0.0539
    2.2376
%which has a lead coefficient different from zero.
pause
%Now for the error:
varx=kw*vardiff*kw'
varx =
    0.0000   -0.0004    0.0024
   -0.0004    0.0195   -0.1500
    0.0024   -0.1500    1.3349
xerror=diag(varx).^.5;
%So the coefficients and errors are:
[x,xerror]
ans =
    0.0237    0.0030
   -0.0539    0.1395
    2.2376    1.1554
%and in particular,
x(1)/xerror(1)
ans =
    7.9682
%So the order c coefficient is statistically different from
%zero (about 8 standard deviations).
pause
%Let's finish off with a bit of graphics.  Let's plot up
%the model predictions together with its uncertainty.
```

```
cp=[.001:.001:.15]';
ap=[cp,cp.^2,cp.^3];
dp=ap*x;
perror=diag(ap*varx*ap').^.5;
figure(2)
plot(cp,dp./cp,'b')
plot(cp,(dp+perror)./cp,'g')
plot(cp,(dp-perror)./cp,'g')
axis([0 .15 0 .1]);
%where we have plotted the 1-sigma confidence interval
%of the model function, based on the given standard deviations.
hold off
echo off
```





Published with MATLAB® 7.12

OK, this works if you Know $\sigma_{\underset{\sim}{b}}$ - but what if you don't?

This often occurs in model transformations when you are linearizing the model - Even if all the σ_b's are the same before the start, you don't have them the same afterwards!

If you can estimate the way error varies for meas., you can use error propagation to improve linear regression!

Let's look at Michaelis-Menten kinetics:

$$v = \frac{V_{max}[S]}{K_m + [S]} \leftarrow \text{substrate conc.}$$

V_{max} ≡ max rate

K_m ≡ Michaelis Menten cst — measure of protein binding cst

The classic way of linearizing this is to take the inverse!

$$\frac{1}{\nu} = \frac{K_m + [S]}{V_{max}[S]}$$

$$= \underbrace{\frac{1}{V_{max}}}_{x_1} + \underbrace{\frac{K_m}{V_{max}}}_{x_2} \frac{1}{[S]}$$

This is known as a Lineweaver-Burk plot. It's very convenient, but it *really* distorts the data!

Let $b = \frac{1}{\nu}$

Suppose we have some σ_ν. What is σ_b? $\Rightarrow$ Use error propagation:

$$\sigma_b^2 = \left(\frac{\partial b}{\partial \nu}\right)^2 \sigma_\nu^2$$

$$\frac{\partial b}{\partial \nu} = \frac{-1}{\nu^2} \quad \text{so} \quad \sigma_b^2 = \frac{\sigma_\nu^2}{\nu^4}$$

A *huge* dependence! How can

we avoid this? We use weighted regression!

Assume σ_ν^2 is constant

$$b_i = \frac{1}{\nu_i}$$

$$\therefore \underset{\approx}{\Sigma}_b^2 = \sigma_\nu^2 \, diag\left(\frac{1}{\underset{\sim}{\nu}^4}\right)$$

And:

$$\underset{\approx}{K}_W \equiv \left\{\underset{\approx}{A}^T \left(diag(\underset{\sim}{\nu}^4)\right)\underset{\approx}{A}\right\}^{-1}\underset{\approx}{A}^T diag(\underset{\sim}{\nu}^4)$$

Note that multiplying $\underset{\approx}{\Sigma}_b^2$ by any constant doesn't affect $\underset{\approx}{K}_W$ (cancels out) - so you don't need to know σ_ν, just assume it's constant!

This weights meas. at high rxn rate more, in Lineweaver Burk plot as they are more accurate!

Ex 17

```
clear
echo on
format compact
%This example demonstrates the consequences of
%systematic error in parameter estimation and
%error calculations.  Suppose we are trying to
%determine the position of a ball at some time
%tp.  It is proposed that we do this by making
%a series of measurements at different times t
%near this value, fitting a line to the data, and
%then evaluating this line at time tp.  We shall
%use standard error propagation formulae to determine
%the error in our prediction.
pause

%First we generate the data set artificially:
t=[0:.1:5]';
c=[.1 1 1]';
signoise=.15;
b=[t.^2,t,ones(size(t))]*c+signoise*randn(size(t));
%and we plot this up:
figure(1)
plot(t,b,'o')
set(gca,'FontSize',18)
xlabel('time')
ylabel('position')
pause

%Next we fit a line to the data:
a=[t,ones(size(t))];
k=inv(a'*a)*a';
x=k*b;
figure(1)
hold on
plot(t,a*x,'g')
hold off
%which fits the data pretty well.
pause

%We also calculate the error making the usual
%assumption that 1) the error is random, and 2)
%the model itself is perfect.
r=b-a*x;
n=length(t);
varb=r'*r/(n-2);
varx=k*k'*varb;
sigx=diag(varx).^.5;
%This yields the slope and intercept with one SD error:
[x,sigx]
pause

%The predicted value at tp = 2.5, with error, is:
tp=2.5;
ap=[tp,1];
```

```
bp=ap*x;
bpvar=ap*varx*ap';
[bp,bpvar.^.5]
%vs. the actual value:
actual=[tp^2,tp,1]*c
%which is off by:
abs(bp-actual)/bpvar^.5
%which is very large!
pause

%The explanation for this is that we have dramatically
%underestimated our standard deviation by assuming that
%the error is simply random - we have ignored our systematic
%error!  We can see that the systematic error is significant
%by plotting the residuals:
figure(2)
plot(t,r,'og')
set(gca,'FontSize',18)
xlabel('time')
ylabel('residual')
pause

%We can quantify the degree of non-randomness by looking at the
%normalized covariance of adjacent points:
covariance=r(1:n-1)'*r(2:n)/varb/(n-1)
%which is close to unity.  This value would be much smaller if the
%data were, in fact, random.
pause

%Judging from the plot of residuals, our error cannot be calculated
%correctly using our standard statistical formulae.  Rather than
%the error being random, instead the deviation between the model
%and the data is because we have left something out of the -model-!
%In this case we can get a better estimate (and error estimate) if
%we include the quadratic term in our model:
a=[t.^2,t,ones(size(t))];
ap=[tp.^2,tp,ones(size(tp))];
k=inv(a'*a)*a';
x=k*b;
figure(1)
hold on
plot(t,a*x,'r')
hold off
%which is pretty similar to the linear fit.
pause

%Now for the error calculation and residual:
r=b-a*x;
varb=r'*r/(n-3);
varx=k*k'*varb;
bp=ap*x;
bpvar=ap*varx*ap';
%We have the new predicted value:
[bp,bpvar.^.5]
%vs. the actual value:
actual=[tp^2,tp,1]*c
```

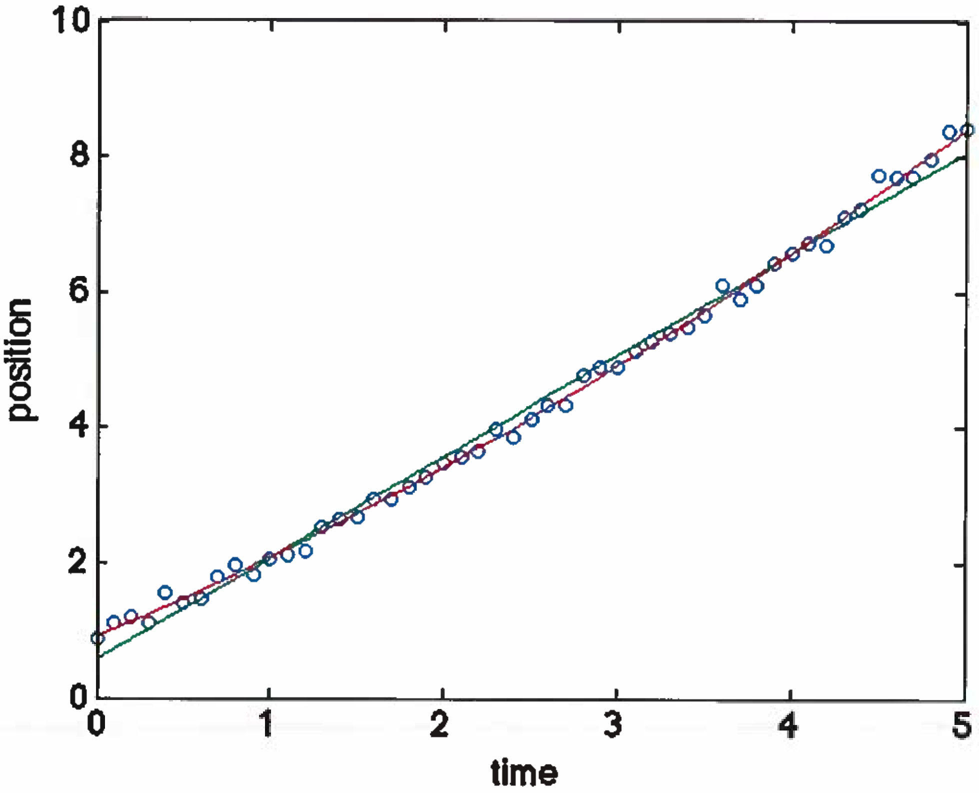
```
%which is off by:
abs(bp-actual)/bpvar^.5
%which is now order one!
pause

%We can also plot up the new residual:
figure(2)
hold on
plot(t,r,'or')
hold off
%which is now random, as was assumed in our
%error calculation.  We can also look at the
%normalized covariance:
covariance=r(1:n-1)'*r(2:n)/varb/(n-1)
%In conclusion, *always* plot your residuals when
%doing curve fitting and error calculations!
pause
%In general, if you ignore systematic error in the
%residual, you will -overestimate- the error in the data points
%themselves, but you will (often dramatically) -underestimate-
%the error in the modeling parameters.  That can lead to
%completely incorrect conclusions from your experiments!
%
%            DON'T MAKE THIS ERROR!!!
echo off
```

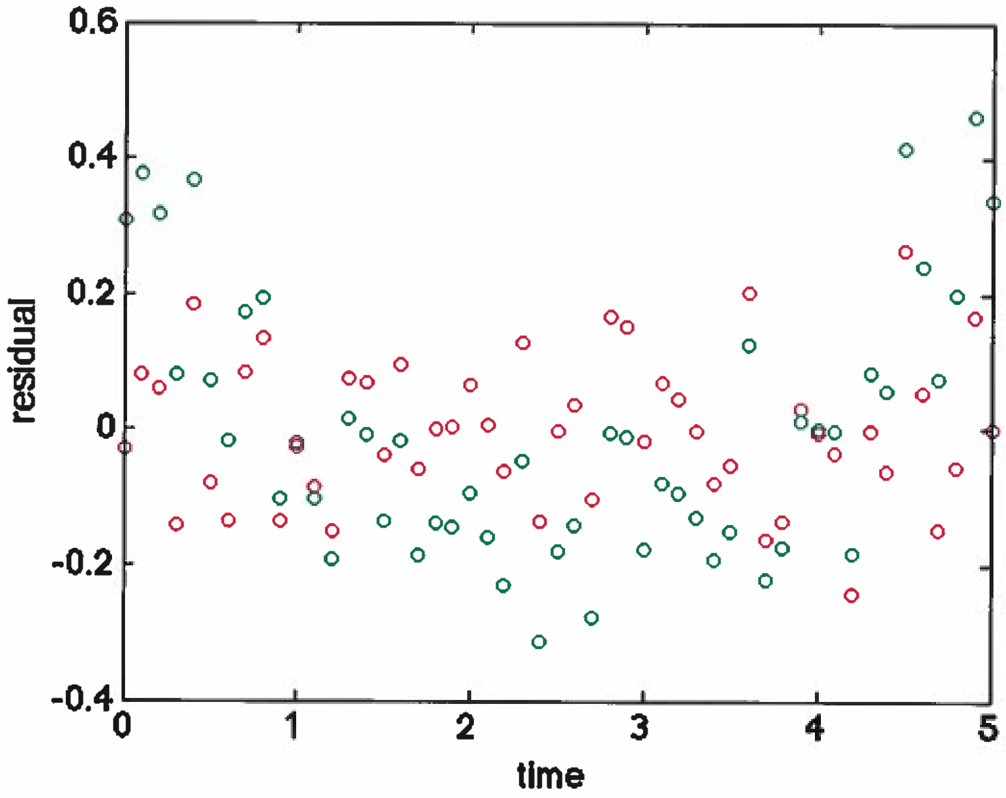
```
format compact
%This example demonstrates the consequences of
%systematic error in parameter estimation and
%error calculations.  Suppose we are trying to
%determine the position of a ball at some time
%tp.  It is proposed that we do this by making
%a series of measurements at different times t
%near this value, fitting a line to the data, and
%then evaluating this line at time tp.  We shall
%use standard error propagation formulae to determine
%the error in our prediction.
pause

%First we generate the data set artificially:
t=[0:.1:5]';
c=[.1 1 1]';
signoise=.15;
b=[t.^2,t,ones(size(t))]*c+signoise*randn(size(t));
%and we plot this up:
figure(1)
plot(t,b,'o')
set(gca,'FontSize',18)
xlabel('time')
ylabel('position')
pause

%Next we fit a line to the data:
a=[t,ones(size(t))];
k=inv(a'*a)*a';
```

```
x=k*b;
figure(1)
hold on
plot(t,a*x,'g')
hold off
%which fits the data pretty well.
pause

%We also calculate the error making the usual
%assumption that 1) the error is random, and 2)
%the model itself is perfect.
r=b-a*x;
n=length(t);
varb=r'*r/(n-2);
varx=k*k'*varb;
sigx=diag(varx).^.5;
%This yields the slope and intercept with one SD error:
[x,sigx]
ans =
    1.5000    0.0186
    0.5703    0.0541
pause

%The predicted value at tp = 2.5, with error, is:
tp=2.5;
ap=[tp,1];
bp=ap*x;
bpvar=ap*varx*ap';
[bp,bpvar.^.5]
ans =
    4.3204    0.0274
%vs. the actual value:
actual=[tp^2,tp,1]*c
actual =
    4.1250
%which is off by:
abs(bp-actual)/bpvar^.5
ans =
    7.1227
%which is very large!
pause

%The explanation for this is that we have dramatically
%underestimated our standard deviation by assuming that
%the error is simply random - we have ignored our systematic
%error!  We can see that the systematic error is significant
%by plotting the residuals:
figure(2)
plot(t,r,'og')
set(gca,'FontSize',18)
xlabel('time')
ylabel('residual')
pause

%We can quantify the degree of non-randomness by looking at the
%normalized covariance of adjacent points:
```

```
covariance=r(1:n-1)'*r(2:n)/varb/(n-1)
covariance =
    0.5640
%which is close to unity.  This value would be much smaller if the
%data were, in fact, random.
pause

%Judging from the plot of residuals, our error cannot be calculated
%correctly using our standard statistical formulae.  Rather than
%the error being random, instead the deviation between the model
%and the data is because we have left something out of the -model-!
%In this case we can get a better estimate (and error estimate) if
%we include the quadratic term in our model:
a=[t.^2,t,ones(size(t))];
ap=[tp.^2,tp,ones(size(tp))];
k=inv(a'*a)*a';
x=k*b;
figure(1)
hold on
plot(t,a*x,'r')
hold off
%which is pretty similar to the linear fit.
pause

%Now for the error calculation and residual:
r=b-a*x;
varb=r'*r/(n-3);
varx=k*k'*varb;
bp=ap*x;
bpvar=ap*varx*ap';
%We have the new predicted value:
[bp,bpvar.^.5]
ans =
    4.1421    0.0232
%vs. the actual value:
actual=[tp^2,tp,1]*c
actual =
    4.1250
%which is off by:
abs(bp-actual)/bpvar^.5
ans =
    0.7355
%which is now order one!
pause

%We can also plot up the new residual:
figure(2)
hold on
plot(t,r,'or')
hold off
%which is now random, as was assumed in our
%error calculation.  We can also look at the
%normalized covariance:
covariance=r(1:n-1)'*r(2:n)/varb/(n-1)
covariance =
   -0.1588
```

```
%In conclusion, *always* plot your residuals when
%doing curve fitting and error calculations!
pause
%In general, if you ignore systematic error in the
%residual, you will -overestimate- the error in the data points
%themselves, but you will (often dramatically) -underestimate-
%the error in the modeling parameters.  That can lead to
%completely incorrect conclusions from your experiments!
%
%            DON'T MAKE THIS ERROR!!!
echo off
```





Published with MATLAB® 7.12

Review of Linear Regression

1) Identify unknowns in model

~~Is model linear in?~~

2) If necessary, transform model so that it is linear in (new) unknown parameters

3) Set up ~~[illegible]~~ problem ~~of~~ in ~~modeling fn~~ matrix form

$\underset{\approx}{A}$ = matrix of modeling functions

$\underset{\sim}{b}$ = column vector ~~(or matrix)~~ of observations

$\underset{\sim}{x}$ = column vector of unknown parameters

4) Solve via normal equations:

$$\underset{\sim}{x} = \underset{\approx}{A} \backslash \underset{\sim}{b} \quad (QR)$$

$$\underset{\approx}{A}^T \underset{\approx}{A} \underset{\sim}{x} = \underset{\approx}{A}^T \underset{\sim}{b} \quad \text{(solve via LU)}$$

$$\underset{\sim}{x} = \left[\left\{ \underset{\approx}{A}^T \underset{\approx}{A} \right\}^{-1} \underset{\approx}{A}^T \right] \underset{\sim}{b} \equiv \underset{\approx}{K} \underset{\sim}{b}$$

5) Define residuals $\underset{\sim}{r} = \underset{\approx}{A}\,\underset{\sim}{x} - \underset{\sim}{b}$

6) Plot Your Residuals! Are they random? Are they uniform in magnitude

⇒ If they aren't random, reexamine model - something is missing!

⇒ If they aren't uniform in magnitude, consider using weighted regression.

7) Estimate $\underset{\approx}{\Sigma}_b^2 = \sigma_b^2\,\underset{\approx}{I}$

where $\sigma_b^2 = \dfrac{\underset{\sim}{r}^T\underset{\sim}{r}}{n-m}$

8) Calculate matrix of covariance of $\underset{\sim}{x}$:

$$\underset{\approx}{\Sigma}_x^2 = \underset{\approx}{K}\,\underset{\approx}{\Sigma}_b^2\,\underset{\approx}{K}^T$$

9) Re cast in terms of original desired model parameters

$$\underset{\sim}{z} = \underset{\sim}{f}(\underset{\sim}{x})$$

10) Calculate Matrix of covariance of original parameters:

$$\underset{\approx}{\Sigma}_z^2 \cong (\underset{\sim}{\nabla} \underset{\sim}{f}) \underset{\approx}{\Sigma}_x^2 (\underset{\sim}{\nabla} \underset{\sim}{f})^T$$

11) Model Predictions : $\underset{\sim}{b}_p = \underset{\approx}{A}_p \underset{\sim}{x}$; $\sigma_{\sim b_p} = \mathrm{diag}(\underset{\approx}{A}_p \underset{\approx}{\Sigma}_x^2 \underset{\approx}{A}_p^T)^{1/2}$

If residual is not uniform,

or if error in $\underset{\sim}{b}$ is known, use weighted regression!

Weight each ~~term (equation)~~ datapoint by the inverse of it's variance!

$$\min_{\underset{\sim}{x}} (\underset{\approx}{A} \underset{\sim}{x} - \underset{\sim}{b})^T (\underset{\approx}{\Sigma}_b^2)^{-1} (\underset{\approx}{A} \underset{\sim}{x} - \underset{\sim}{b})$$

or: $\underset{\approx}{A}^T (\underset{\approx}{\Sigma}_b^2)^{-1} \underset{\approx}{A} \underset{\sim}{x} = \underset{\approx}{A}^T (\underset{\approx}{\Sigma}_b^2)^{-1} \underset{\sim}{b}$

or

$$\underset{\sim}{x} = \underset{\approx}{K}_w \underset{\sim}{b} = \left[\left\{ \underset{\approx}{A}^T (\underset{\approx}{\Sigma}_b^2)^{-1} \underset{\approx}{A} \right\}^{-1} \underset{\approx}{A}^T (\underset{\approx}{\Sigma}_b^2)^{-1} \right] \underset{\sim}{b}$$

Non-Linear Regression

So far we've focussed on *linear regression*: problems where the model is *linear* in the unknown *modelling parameters*. This is convenient, but not really necessary! Suppose we have the non-linear model:

$$\underline{b} = f(\underset{\sim}{x}, t)$$

where $\underset{\sim}{x}$: is an array of modelling functions.

We may still define the *residual* between observed and fitted values:

$$\underset{\text{residual}}{\underset{\uparrow}{r_i}} = \underset{\text{meas}}{\underset{\uparrow}{b_i}} - f(\underset{\text{model}}{\underset{\uparrow}{\underset{\sim}{x}}}, t_i)$$

Thus we can form the sum of squares:

$$F(\underset{\sim}{x}) = \sum_{i=1}^{N} (r_i)^2$$

The best fitted values for $\underset{\sim}{x}$ are obtained via finding the minimum of $F(\underset{\sim}{x})$!

Be Wary: Non-linear optimization problems may have local minima which can trap the solver! Also, make sure you have set up the parameters so that $F(\underset{\sim}{x})$ is well conditioned. The dependence on each parameter should be of similar magnitude. Let's look at an example:

Arrhenius Kinetics

$$\text{rate} \sim K_0 e^{-E/RT}$$

In senior lab you will measure the rate of oxidation of methane as a function of temperature, and use it to try to get the activation energy E for the catalyst. If we could measure this rate, holding everything but T constant, we could use linear regression. Unfortunately, the rate also depends on concentration, and that depends (experimentally) on T as well!

The exp't is written up in Chem Eng. Education, 36 (1) p. 34-40. Suppose we feed a reactor (well-mixed) a concentration of Methane C_{r_0} at flow rate q_r. We measure some outlet concentration C_r. The subscript "r" is because the reactor is <u>hot</u>: due to expansion, the <u>actual</u> concentration is $\frac{C}{C_r} = \frac{T_r}{T}$ from the ideal gas law - where T_r is some ref. temp (°K).

The system is further complicated be cause the reaction is <u>fractional order</u>, e.g.

$$rate \sim C^n$$

where $n \neq 1$ (not first order)

we need to figure out n too!

From a mass balance, we get the rate of rxn per gram of catalyst:

$$\frac{q_r}{m}\left(C_{r0}-C_r\right) = \left(C_r \frac{T_r}{T}\right)^n K_0 e^{-\frac{E}{RT}}$$

where m is the mass of catalyst. We can also look at the conversion ratio given by:

$$X = 1 - \frac{C_r}{C_{r_0}}$$

which yields (after rearrangement):

$$\frac{X}{(1-X)^n} = \frac{m}{q_r} C_{0r}^{n-1}\left(\frac{T_r}{T}\right)^n K_0 e^{-\frac{E}{RT}}$$

If you were to fix T and plot X vs. C_{0r}, you find it decreases w/ C_{0r}, thus $n<1$ for this system.

Ok, suppose we vary C_{ro} and T, and measure C_r. How do we get the unknowns n, k_o, and E?

The classic approach is to do it in two steps:

1) Fix T and plot

$$\ln\left\{\frac{q_r}{m}(C_{or}-C_r)\right\} \text{ vs. } \ln\{C_r\}$$

You should get a straight line with slope n!

2) With this in hand, you can then do an Arrhenius Plot of:

$$\ln\left\{\frac{q_r}{m}\frac{(C_{or}-C_r)}{C_r^{\,n}}\left(\frac{T_r}{T}\right)^{-n}\right\} = \ln k_o - \frac{E}{R}\frac{1}{T}$$

Thus, the slope (vs. $\frac{1}{T}$) is just E/R and the intercept is $\ln k_o$!

This works fine if the data is good. Unfortunately, this is not usually the case. The problem is that C_r is a much stronger function of T than C_{r_o}, and thus you get large errors in the first step (n), leading to large errors in the second step (E & K_o). The fitting parameters are also strongly biased by errors at low conversions.

We can avoid this by using non-linear regression! We simply define the deviation from the model Δ:

$$\Delta = C_r - C_{r_0} + \frac{m}{q_r} C_r^n K_0 e^{-\frac{E}{RT}} \left(\frac{T_r}{T}\right)^n$$

and minimize the objective function:

$$F(K_0, E, n) \equiv \sum \Delta_i^2$$

over all the experiments! The sensitivity of the fitting parameters to error can then be easily calculated using the non-linear error propagation/gradient method, accounting for the error in both C_r and T. Note that you should work with n, $\ln K_0$, and $\frac{E}{RT_r}$ as fitting parameters so that the optimization problem is well-conditioned!

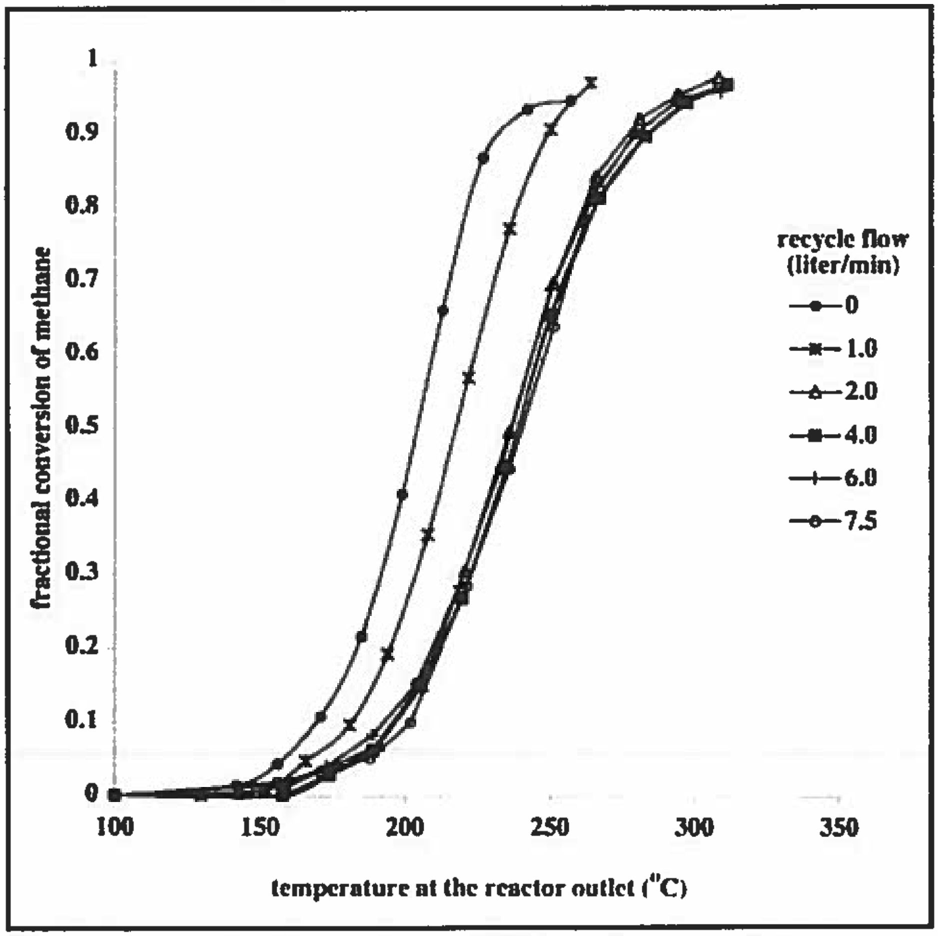


Figure 2. *The effect of recycle flow on methane conversion for a feed concentration, C_{or}, of $1.976x10^{-4}$ mol/liter.*

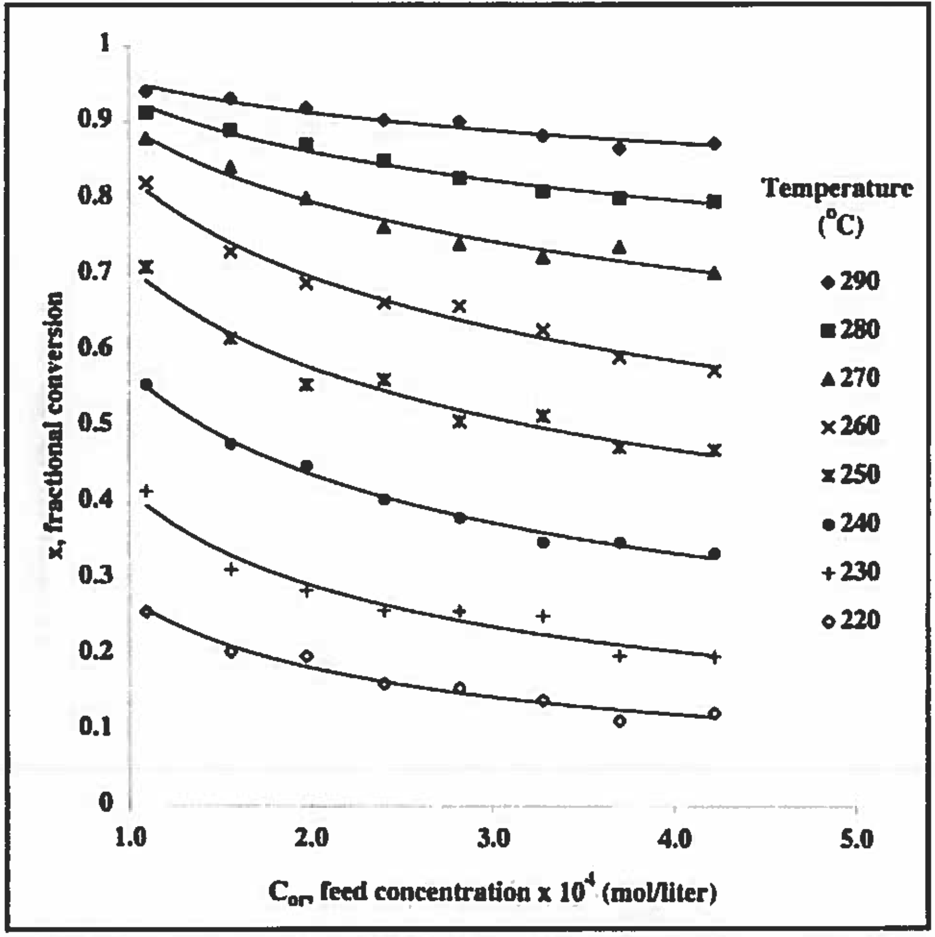


Figure 3. *The effect of feed concentration on methane conversion under gradientless conditions. The curves are least-squares polynomial representations of the data.*

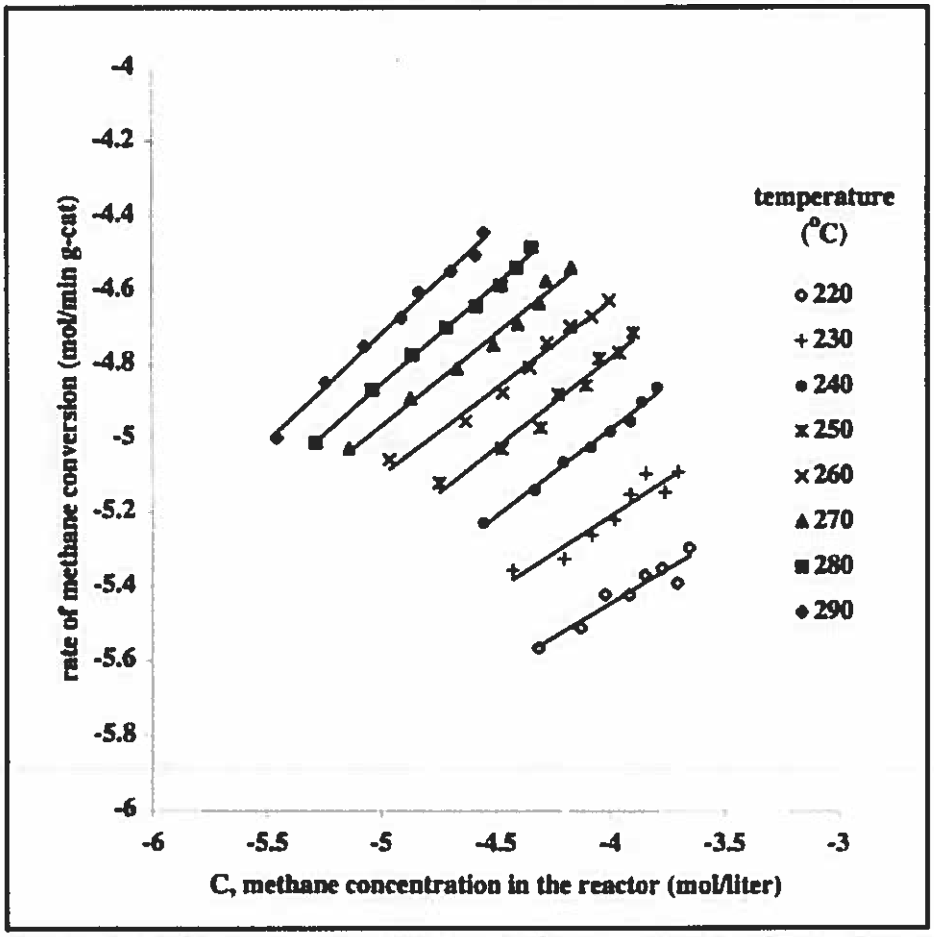


Figure 4. Methane conversion rates. The straight lines are least-squares representations of the data shown. The numbers on the coordinates are exponents of 10.

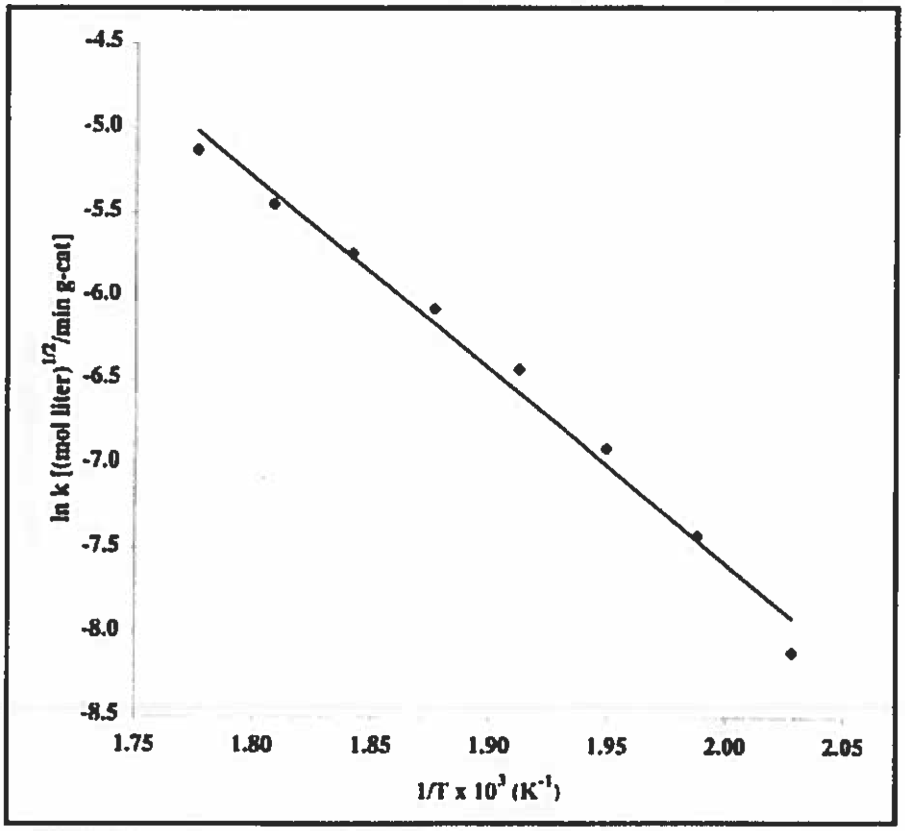
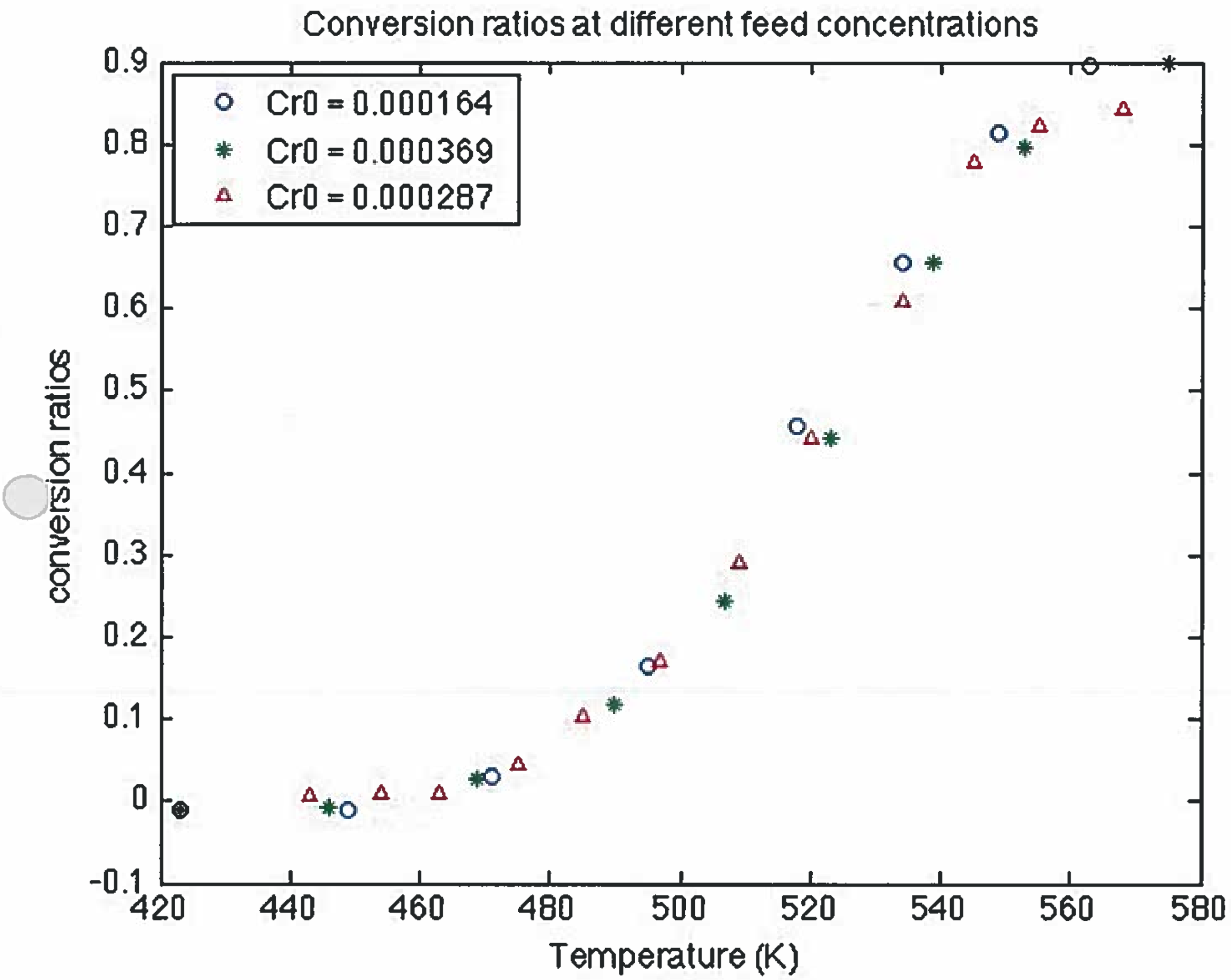


Figure 5. *An Arrhenius plot. The values of k_o and E/R, determined from the least-squares line shown, are given in Eqs. (6) and (7).*


Conversion ratios at different feed concentrations
Cr0 = 0.000164
Cr0 = 0.000369
Cr0 = 0.000287
conversion ratios
Temperature (K)
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-0.1
420
440
460
480
500
520
540
560
580

```
function y=delta(guess)
%This function takes in guesses for the unknown parameters n, ln(k0) and
%E/RTr, and returns the deviation between the model and the data.  We bring
%the data in through the "global" command:
global crpass cr0pass Tpass

Tr=298; %The reference temperature.
qr=0.1; %The flow rate (liters/min)
m=1; %The amount of catalyst (g)

n=guess(1); %The first parameter
k0=exp(guess(2)); %The second parameter
ERTr=guess(3); %The third parameter

miss=crpass-cr0pass+m/qr*crpass.^n*k0.*(Tr./Tpass).^n.*exp(-ERTr*Tr./Tpass);

%OK, the question is how to weight each of the data points.  As it is
%currently written, it tends to accentuate the weighting at higher
%temperatures where the conversion is the largest.  This is because cr is
%lowest, and it is multiplied by a large value to make it balance cr0.  It
%also places a stronger weight on the runs with higher intial
%concentrations.  On the other hand, for lower cr we should have more
%accurate measurements (if the fractional error is fixed, for example).
%Different weightings will yield different "solutions" for optimal parameters.

%A reasonable choice is to weight each of the runs with the inverse of the
%initial concentration.  This is essentially equivalent to assuming an error
%proportional to the concentration measured, and each data point should be
%of O(1).  Thus:
miss=miss./cr0pass;

%and thus we get the objective function:
y=sum(miss.*miss);
```

Ex. 18

```
clear
echo on
%In this example we analyze the catalytic oxidation data obtained by a
%group of students in senior lab.  Under CSTR conditions they measured outlet
%concentrations for three different reactor feed concentrations.  The data
%for the three feeds cr0 are given below:

Ta=[423 449 471 495 518 534 549 563];
cra=[1.66E-04   1.66E-04   1.59E-04   1.37E-04   8.90E-05   5.63E-05   3.04E-0
cr0a=1.64E-4;

Tb=[423 446 469 490 507 523 539 553 575];
crb=[3.73E-04   3.72E-04   3.59E-04   3.26E-04   2.79E-04   2.06E-04   1.27E-0
cr0b=3.69e-4;

Tc=[443 454 463 475 485 497 509 520 534 545 555 568];
crc=[2.85E-04   2.84E-04   2.84E-04   2.74E-04   2.57E-04   2.38E-04   2.04E-0
cr0c=2.87e-4;
pause

%We can calculate conversion ratios for these three runs:
xa=1-cra/cr0a;
xb=1-crb/cr0b;
xc=1-crc/cr0c;

%and we can plot them up:
figure(1)
plot(Ta,xa,'o',Tb,xb,'*',Tc,xc,'^')
xlabel('Temperature (K)','FontSize',14)
ylabel('conversion ratios','FontSize',14)
legend(['Cr0 = ',num2str(cr0a)],['Cr0 = ',num2str(cr0b)],['Cr0 = ',num2str(cr0c)],'I
title('Conversion ratios at different feed concentrations','FontSize',14)
set(gca,'FontSize',14)
pause

%Looking at this plot, we can immediately see why the standard technique
%for analyzing the reaction data will run into trouble: Even with
%interpolation, it will be very hard to get accurate values of the
%conversion at different concentrations for fixed temperatures.  To use
%non-linear regression to get at the fitting parameters, we will have to
%define an objective function for minimization, as well as some initial
%guesses for the parameters.  We can pass the data into the objective
%function using the "global" meat axe:

global crpass cr0pass Tpass

Tpass=[Ta,Tb,Tc];
crpass=[cra,crb,crc];
cr0pass=[cr0a*ones(size(Ta)),cr0b*ones(size(Tb)),cr0c*ones(size(Tc))];

%and you will have to save the function "delta.m" which returns the
%objective function to be minimized.
pause
```

```
%OK, let's do it.  We have the initial guesses:
guess=zeros(3,1);
guess(1)=.5; %This is the guess for n
guess(2)=15; %This is the guess for ln(k0)
guess(3)=38; %This is the guess for E/RTr

%And we go:
guess=fminsearch('delta',guess)
pause
%Looking at these values, they aren't too far off of those in the
%literature.  In particular, the exponent is quite close to the
%expected value of 0.5, and the activation energy is only off by 7%!
pause

%We can plot the model up too.  We need the other parameters
%(both here and in the function delta.m).
Tr=298; %The reference temperature.
qr=0.1; %The flow rate (liters/min)
m=1; %The amount of catalyst (g)

n=guess(1);
k0=exp(guess(2));
ERTr=guess(3);

Trange=[min(Tpass):max(Tpass)]; %A plotting range
xmodel=m/qr*(Tr./Trange).^n*k0.*exp(-ERTr*Tr./Trange);

xafn=xa./(1-xa).^n/cr0a^(n-1);
xbfn=xb./(1-xb).^n/cr0b^(n-1);
xcfn=xc./(1-xc).^n/cr0c^(n-1);

%OK, we've got the model and the data for the function x/(1-x)^n/cr0^(n-1).
%It should be independent of the concentration. Let's plot it up:
pause
figure(2)
plot(Ta,xafn,'o',Tb,xbfn,'*',Tc,xcfn,'+',Trange,xmodel)
xlabel('Temperature (K)','FontSize',14)
ylabel('x/(1-x)^n/cr0^(n-1)','FontSize',14)
legend(['Cr0 = ',num2str(cr0a)],['Cr0 = ',num2str(cr0b)],['Cr0 = ',num2str(cr0c)],'m
title(['Comparison of data to model, n = ',num2str(n)],'FontSize',14)
set(gca,'FontSize',14)
%Which shows that we get pretty much perfect collapse of the data.
pause

%Now we turn to the trickier error calculations.  First, we need to get a
%measure of the uncertainty in the concentration measurements.  We can get
%this from the magnitude of the "miss" in the data:
miss=crpass-cr0pass+m/qr*crpass.^n*k0.*(Tr./Tpass).^n.*exp(-ERTr*Tr./Tpass);

%We must adjust this to account for the relative weighting of the data.  In
%this case, a rough correction for the actual fractional deviation in cr
%is given by:
miss=miss./cr0pass.*(crpass./cr0pass);

%Thus we get the fractional standard deviation (assuming randomness) of:
crstdev=norm(miss)/(length(Tpass)-3)^.5
```

```
%Which yields a fractional error of around 3% - not too bad.  Note that these
%deviations could have been due to errors in the temperature just as readily!
pause

%OK, it is always important to plot up the residuals to see if the error is
%really random.  It is useful to plot up the actual cr's and predicted
%cr's.  Alas, we have an implicit equation for the predicted cr's which
%cannot be solved analytically.  Instead, we shall use the "miss" from the
%minimization routine.  We need the range of indices corresponding to each
%data set:
a=[1:length(Ta)];
b=[max(a)+1:max(a)+length(Tb)];
c=[max(b)+1:max(b)+length(Tc)];

figure(3)
plot(Ta,miss(a),'o',Tb,miss(b),'*',Tc,miss(c),'+')
hold on
plot(Trange,zeros(size(Trange)))
hold off
xlabel('Temperature (K)','FontSize',14)
ylabel('Residual (dimensionless fractional deviation)','FontSize',14)
title('Plot of Residuals','FontSize',14)
legend(['Cr0 = ',num2str(cr0a)],['Cr0 = ',num2str(cr0b)],['Cr0 = ',num2str(cr0c)],'
set(gca,'FontSize',14)
pause

%As you can see, there is a pretty significant systematic deviation between
%the model and the data.  That means that the random error in cr is
%-overestimated- (it's less than 3%) while assuming the data to be
%dominated by independent random error will lead to errors in fitting parameters
%to be underestimated!  It also means that there is something going on which is
%not captured by the model - not a big surprise.
pause

%OK, we still want to measure the sensitivity of the calculated values to
%errors in measured concentrations.  This can be done by taking the
%derivative of the fitted values with respect to each of the data points
%(not forgetting the initial concentrations, which have error too!).
%First, let's do the cr's:

crkeep=crpass;
gradfcr=zeros(3,length(Tpass)); %The array where we stuff the gradient.
for j=1:length(Tpass)
    crpass=crkeep;
    ep=crpass(j)*crstdev; %The amount we change the j'th data point by.
    crpass(j)=crpass(j)+ep;
    %Now we calculate new values of the fitted parameters:
    newguess=fminsearch('delta',guess);
    gradfcr(:,j)=(newguess-guess)/ep; %The gradient.
    echo off
end
echo on

%And we do the same for the initial concentrations:
crpass=crkeep;
```

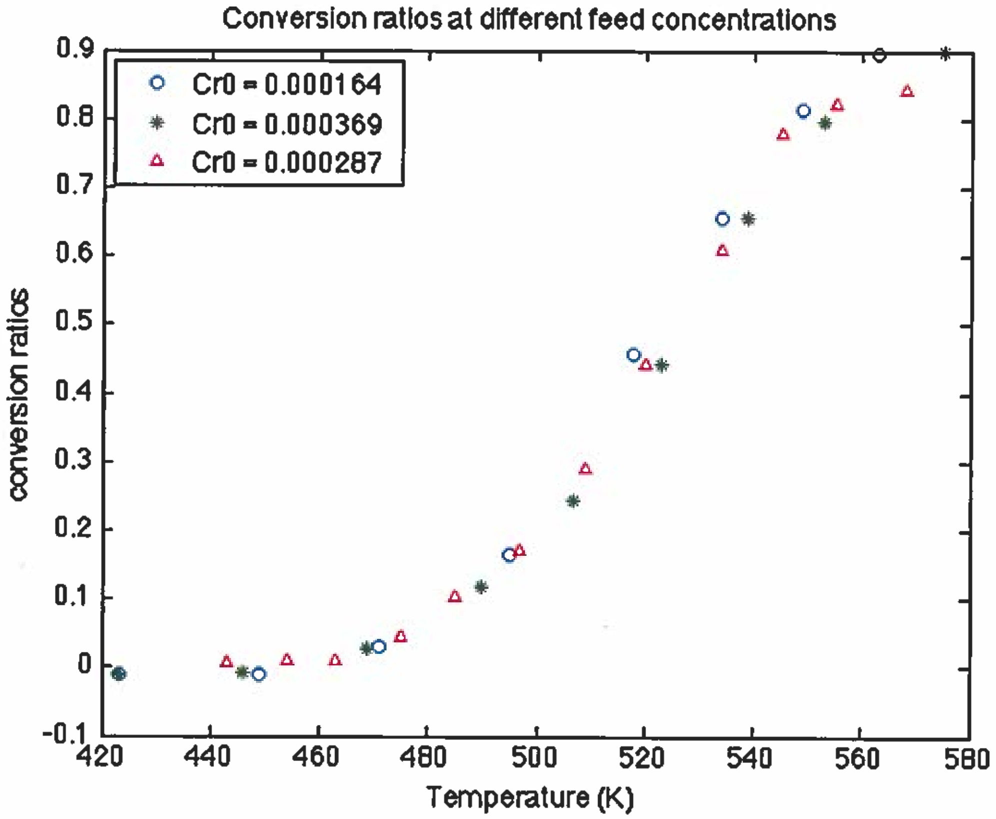
```
gradfcr0=zeros(3,3); %We had three initial concentrations.

cr0pass=[cr0a*(1+crstdev)*ones(size(Ta)),cr0b*ones(size(Tb)),cr0c*ones(size(Tc))];
gradfcr0(:,1)=(fminsearch('delta',guess)-guess)/(crstdev*cr0a);

cr0pass=[cr0a*ones(size(Ta)),cr0b*(1+crstdev)*ones(size(Tb)),cr0c*ones(size(Tc))];
gradfcr0(:,2)=(fminsearch('delta',guess)-guess)/(crstdev*cr0b);

cr0pass=[cr0a*ones(size(Ta)),cr0b*ones(size(Tb)),cr0c*(1+crstdev)*ones(size(Tc))];
gradfcr0(:,3)=(fminsearch('delta',guess)-guess)/(crstdev*cr0c);

%and we put cr0pass back again:
cr0pass=[cr0a*ones(size(Ta)),cr0b*ones(size(Tb)),cr0c*ones(size(Tc))];

pause

%That gives us the sensitivity gradients.  To complete the problem, we need
%to determine the matrix of covariance of the concentration measurements.
%This is a little more "iffy" because we -know- that they are not really
%random!  Still, if we make the randomness assumption, we can get an
%estimate of the matrix of covariance of the fitting parameters.

varcr=diag((crstdev*crpass).^2);
varcr0=diag(([cr0a,cr0b,cr0c]*crstdev).^2);
%Note that we multiply by the value of cr, etc., as crstdev was an
%estimate of the -fractional- standard deviation!
pause

%These will both contribute to the uncertainty.  We can look at each
%separately.  First, from cr:
var1=gradfcr*varcr*gradfcr'

%and now from cr0:
var2=gradfcr0*varcr0*gradfcr0'
pause

%Note that the error due to the initial concentrations is actually greater
%than that due to all the reactor outlet measurements put together!  That's
%because it is used in every calculated value of the fitting parameters,
%while the "randomness" of the outlet measurements gets averaged out.

%Putting these two together yields an estimate of the variance:
var=var1+var2

%and the fitting parameters + error:
[guess,diag(var).^.5]

%which, I would guess, gives a reasonable measure of the random uncertainty
%in the values.  This is because the uncertainty in cr0 (which is probably
%overestimated) dominates the calculation, while the "randomness
%assumption" underestimates the contribution due to error in cr.  The total
%error will be greater due to other contributions not considered here.  In
%particular, calibration errors will yield systematic error not observable
%from the residuals!
pause
```

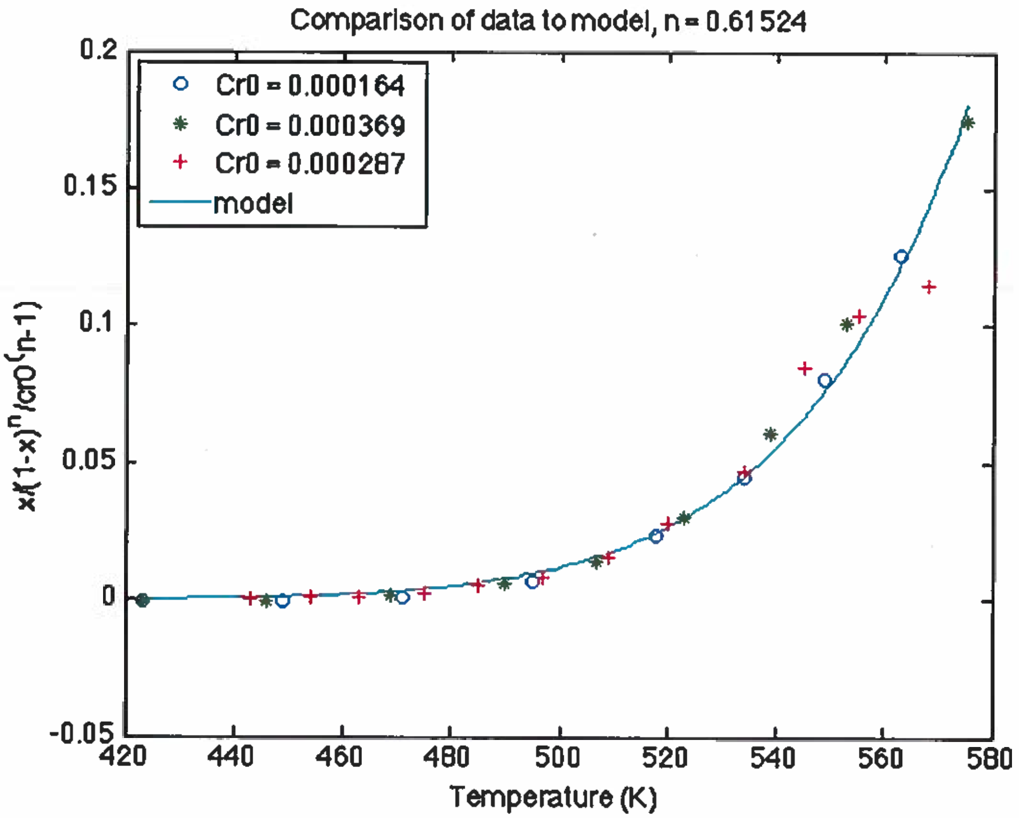
```
%So, in conclusion, the fractional exponent lies within approximately two
%standard deviations of the literature value of 0.5, and the activation
%energy is also within two sigma of 38.5 (at Tr=298K).  Error estimates
%could be improved by having independent estimates of the uncertainty in
%cr0 (the dominant source), by modeling the matrix of covariance in cr,
%and by studying the effect of errors in the other parameters in the
%problem, such as T, qr, and m.  You can also study how the modeling
%parameters change if you leave cr0 as an additional adjustable parameter
%in the model, and simply add its normalized deviation from the measured
%value as an additional contribution to the objective function.  That would
%decrease the model dependence on these few particular data points, and
%might actually decrease the parameter error bars and reduce the systematic
%deviation in the residual.  To get the most out of your data, you need to
%think about the analysis procedure: what assumptions and relative
%weighting it is putting on particular data points.  You will have fun with
%this experiment senior year!
echo off
```

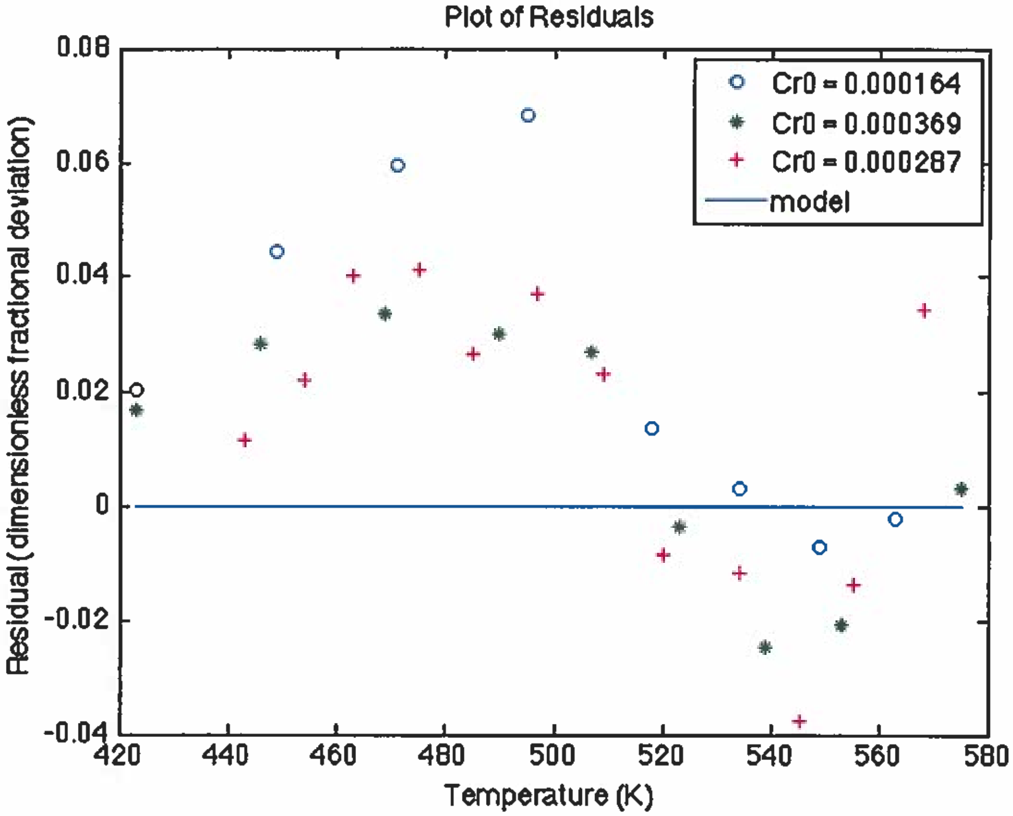
```
%In this example we analyze the catalytic oxidation data obtained by a
%group of students in senior lab.  Under CSTR conditions they measured outlet
%concentrations for three different reactor feed concentrations.  The data
%for the three feeds cr0 are given below:
Ta=[423 449 471 495 518 534 549 563];
cra=[1.66E-04	1.66E-04	1.59E-04	1.37E-04	8.90E-05	5.63E-05	3.04E-05
cr0a=1.64E-4;

Tb=[423 446 469 490 507 523 539 553 575];
crb=[3.73E-04	3.72E-04	3.59E-04	3.26E-04	2.79E-04	2.06E-04	1.27E-04
cr0b=3.69e-4;

Tc=[443 454 463 475 485 497 509 520 534 545 555 568];
crc=[2.85E-04	2.84E-04	2.84E-04	2.74E-04	2.57E-04	2.38E-04	2.04E-04
cr0c=2.87e-4;
pause

%We can calculate conversion ratios for these three runs:
xa=1-cra/cr0a;
xb=1-crb/cr0b;
xc=1-crc/cr0c;

%and we can plot them up:
figure(1)
plot(Ta,xa,'o',Tb,xb,'*',Tc,xc,'^')
xlabel('Temperature (K)','FontSize',14)
ylabel('conversion ratios','FontSize',14)
legend(['Cr0 = ',num2str(cr0a)],['Cr0 = ',num2str(cr0b)],['Cr0 = ',num2str(cr0c)],'I
title('Conversion ratios at different feed concentrations','FontSize',14)
set(gca,'FontSize',14)
pause

%Looking at this plot, we can immediately see why the standard technique
%for analyzing the reaction data will run into trouble: Even with
%interpolation, it will be very hard to get accurate values of the
%conversion at different concentrations for fixed temperatures.  To use
%non-linear regression to get at the fitting parameters, we will have to
```

```
%define an objective function for minimization, as well as some initial
%guesses for the parameters.  We can pass the data into the objective
%function using the "global" meat axe:
global crpass cr0pass Tpass

Tpass=[Ta,Tb,Tc];
crpass=[cra,crb,crc];
cr0pass=[cr0a*ones(size(Ta)),cr0b*ones(size(Tb)),cr0c*ones(size(Tc))];

%and you will have to save the function "delta.m" which returns the
%objective function to be minimized.
pause

%OK, let's do it.  We have the initial guesses:
guess=zeros(3,1);
guess(1)=.5; %This is the guess for n
guess(2)=15; %This is the guess for ln(k0)
guess(3)=38; %This is the guess for E/RTr

%And we go:
guess=fminsearch('delta',guess)
guess =
    0.6152
   15.0920
   36.0888
pause
%Looking at these values, they aren't too far off of those in the
%literature.  In particular, the exponent is quite close to the
%expected value of 0.5, and the activation energy is only off by 7%!
pause

%We can plot the model up too.  We need the other parameters
%(both here and in the function delta.m).
Tr=298; %The reference temperature.
qr=0.1; %The flow rate (liters/min)
m=1; %The amount of catalyst (g)

n=guess(1);
k0=exp(guess(2));
ERTr=guess(3);

Trange=[min(Tpass):max(Tpass)]; %A plotting range
xmodel=m/qr*(Tr./Trange).^n*k0.*exp(-ERTr*Tr./Trange);

xafn=xa./(1-xa).^n/cr0a^(n-1);
xbfn=xb./(1-xb).^n/cr0b^(n-1);
xcfn=xc./(1-xc).^n/cr0c^(n-1);

%OK, we've got the model and the data for the function x/(1-x)^n/cr0^(n-1).
%It should be independent of the concentration. Let's plot it up:
pause
figure(2)
plot(Ta,xafn,'o',Tb,xbfn,'*',Tc,xcfn,'+',Trange,xmodel)
xlabel('Temperature (K)','FontSize',14)
ylabel('x/(1-x)^n/cr0^(n-1)','FontSize',14)
legend(['Cr0 = ',num2str(cr0a)],['Cr0 = ',num2str(cr0b)],['Cr0 = ',num2str(cr0c)],'m
title(['Comparison of data to model, n = ',num2str(n)],'FontSize',14)
set(gca,'FontSize',14)
%Which shows that we get pretty much perfect collapse of the data.
pause

%Now we turn to the trickier error calculations.  First, we need to get a
%measure of the uncertainty in the concentration measurements.  We can get
%this from the magnitude of the "miss" in the data:
miss=crpass-cr0pass+m/qr*crpass.^n*k0.*(Tr./Tpass).^n.*exp(-ERTr*Tr./Tpass);

%We must adjust this to account for the relative weighting of the data.  In
%this case, a rough correction for the actual fractional deviation in cr
%is given by:
miss=miss./cr0pass.*(crpass./cr0pass);

%Thus we get the fractional standard deviation (assuming randomness) of:
crstdev=norm(miss)/(length(Tpass)-3)^.5
crstdev =
    0.0312

%Which yields a fractional error of around 3% - not too bad.  Note that these
%deviations could have been due to errors in the temperature just as readily!
pause

%OK, it is always important to plot up the residuals to see if the error is
%really random.  It is useful to plot up the actual cr's and predicted
%cr's.  Alas, we have an implicit equation for the predicted cr's which
%cannot be solved analytically.  Instead, we shall use the "miss" from the
%minimization routine.  We need the range of indices corresponding to each
%data set:
a=[1:length(Ta)];
b=[max(a)+1:max(a)+length(Tb)];
c=[max(b)+1:max(b)+length(Tc)];

figure(3)
plot(Ta,miss(a),'o',Tb,miss(b),'*',Tc,miss(c),'+')
hold on
plot(Trange,zeros(size(Trange)))
hold off
xlabel('Temperature (K)','FontSize',14)
ylabel('Residual (dimensionless fractional deviation)','FontSize',14)
title('Plot of Residuals','FontSize',14)
legend(['Cr0 = ',num2str(cr0a)],['Cr0 = ',num2str(cr0b)],['Cr0 = ',num2str(cr0c)],'n
set(gca,'FontSize',14)
pause

%As you can see, there is a pretty significant systematic deviation between
%the model and the data.  That means that the random error in cr is
%-overestimated- (it's less than 3%) while assuming the data to be
%dominated by independent random error will lead to errors in fitting parameters
%to be underestimated!  It also means that there is something going on which is
%not captured by the model - not a big surprise.
pause

%OK, we still want to measure the sensitivity of the calculated values to
%errors in measured concentrations.  This can be done by taking the
```

```
%derivative of the fitted values with respect to each of the data points
%(not forgetting the initial concentrations, which have error too!).
%First, let's do the cr's:
crkeep=crpass;
gradfcr=zeros(3,length(Tpass)); %The array where we stuff the gradient.
for j=1:length(Tpass)
    crpass=crkeep;
    ep=crpass(j)*crstdev; %The amount we change the j'th data point by.
    crpass(j)=crpass(j)+ep;
    %Now we calculate new values of the fitted parameters:
    newguess=fminsearch('delta',guess);
    gradfcr(:,j)=(newguess-guess)/ep; %The gradient.
    echo off

%And we do the same for the initial concentrations:
crpass=crkeep;
gradfcr0=zeros(3,3); %We had three initial concentrations.

cr0pass=[cr0a*(1+crstdev)*ones(size(Ta)),cr0b*ones(size(Tb)),cr0c*ones(size(Tc))];
gradfcr0(:,1)=(fminsearch('delta',guess)-guess)/(crstdev*cr0a);

cr0pass=[cr0a*ones(size(Ta)),cr0b*(1+crstdev)*ones(size(Tb)),cr0c*ones(size(Tc))];
gradfcr0(:,2)=(fminsearch('delta',guess)-guess)/(crstdev*cr0b);

cr0pass=[cr0a*ones(size(Ta)),cr0b*ones(size(Tb)),cr0c*(1+crstdev)*ones(size(Tc))];
gradfcr0(:,3)=(fminsearch('delta',guess)-guess)/(crstdev*cr0c);

%and we put cr0pass back again:
cr0pass=[cr0a*ones(size(Ta)),cr0b*ones(size(Tb)),cr0c*ones(size(Tc))];

pause

%That gives us the sensitivity gradients.  To complete the problem, we need
%to determine the matrix of covariance of the concentration measurements.
%This is a little more "iffy" because we -know- that they are not really
%random!  Still, if we make the randomness assumption, we can get an
%estimate of the matrix of covariance of the fitting parameters.
varcr=diag((crstdev*crpass).^2);
varcr0=diag(([cr0a,cr0b,cr0c]*crstdev).^2);
%Note that we multiply by the value of cr, etc., as crstdev was an
%estimate of the -fractional- standard deviation!
pause

%These will both contribute to the uncertainty.  We can look at each
%separately.  First, from cr:
var1=gradfcr*varcr*gradfcr'
var1 =
    0.0005    0.0117    0.0127
    0.0117    0.3550    0.4275
    0.0127    0.4275    0.5431

%and now from cr0:
var2=gradfcr0*varcr0*gradfcr0'
var2 =
    0.0027    0.0697    0.0759
    0.0697    1.8849    2.1491
    0.0759    2.1491    2.5335
pause

%Note that the error due to the initial concentrations is actually greater
%than that due to all the reactor outlet measurements put together!  That's
%because it is used in every calculated value of the fitting parameters,
%while the "randomness" of the outlet measurements gets averaged out.
%Putting these two together yields an estimate of the variance:
var=var1+var2
var =
    0.0032    0.0814    0.0886
    0.0814    2.2400    2.5766
    0.0886    2.5766    3.0765

%and the fitting parameters + error:
[guess,diag(var).^.5]
ans =
    0.6152    0.0565
   15.0920    1.4966
   36.0888    1.7540

%which, I would guess, gives a reasonable measure of the random uncertainty
%in the values.  This is because the uncertainty in cr0 (which is probably
%overestimated) dominates the calculation, while the "randomness
%assumption" underestimates the contribution due to error in cr.  The total
%error will be greater due to other contributions not considered here.  In
%particular, calibration errors will yield systematic error not observable
%from the residuals!
pause

%So, in conclusion, the fractional exponent lies within approximately two
%standard deviations of the literature value of 0.5, and the activation
%energy is also within two sigma of 38.5 (at Tr=298K).  Error estimates
%could be improved by having independent estimates of the uncertainty in
%cr0 (the dominant source), by modeling the matrix of covariance in cr,
%and by studying the effect of errors in the other parameters in the
%problem, such as T, qr, and m.  You can also study how the modeling
%parameters change if you leave cr0 as an additional adjustable parameter
%in the model, and simply add its normalized deviation from the measured
%value as an additional contribution to the objective function.  That would
%decrease the model dependence on these few particular data points, and
%might actually decrease the parameter error bars and reduce the systematic
%deviation in the residual.  To get the most out of your data, you need to
%think about the analysis procedure: what assumptions and relative
%weighting it is putting on particular data points.  You will have fun with
%this experiment senior year!
echo off
```


Conversion ratios at different feed concentrations
Cr0 = 0.000164
Cr0 = 0.000369
Cr0 = 0.000287
conversion ratios
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-0.1
420
440
460
480
500
520
540
560
580
Temperature (K)


Comparison of data to model, n = 0.61524
Cr0 = 0.000164
Cr0 = 0.000369
Cr0 = 0.000287
model
x/(1-x)^n/cr0^(n-1)
0.2
0.15
0.1
0.05
0
-0.05
420
440
460
480
500
520
540
560
580
Temperature (K)



Published with MATLAB® 7.12

So far we've examined error propagation in terms of linear combinations of variables.

For non-linear functions, we used a truncated Taylor series to linearize it (valid for small variations/errors).

There's another way to do all this: Resampling Methods

The basic idea of all these methods is the same: That there exists some "population" of possible data measurements. The actual data measurements are drawn from this population. Thus, if we sample the data in different ways, we can get parameter estimates that give us a measure of error!

we'll look at 4 methods:

1) The Jackknife: Sequentially drop one data point and determine how parameters change

2) The Bootstrap: Draw a sample of n data points from a periodically replicated set of original data, determine how parameters change

3) Undersampling: Divide a large data set into smaller samples, calculate parameters & see how they vary

4) Monte Carlo simulation: Add random noise (of approp. mag) to data, calc parameters & see how they change.

First we look at the Jackknife

— first suggested ~1949, term coined '58

Suppose we have N meas

we wish to calculate the statistics of $f(\bar{x})$ (this can be non-linear)

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

we may define f_j s.t.

$$f_j \equiv f\left(\frac{1}{N-1}\sum_{i\neq j} x_i\right)$$

↳ avg. of x excluding jth point!

The quantity $f(\bar{x})$ is estimated by the average of f_j!

$$f(\bar{x}) \approx \frac{1}{N}\sum_{j=1}^{N} f_j$$

↳ less bias than $\overline{f(x_i)}$ for non-linear prob.

The thing is, we can also use this to get the variance!

The formula is:

$$\sigma^2_{f(\bar{x})} = \frac{N-1}{N} \sum_{j=1}^{N} \left(f_j - \bar{f}_j \right)^2$$

This looks a little strange, but it makes sense from error propagation

Consider the application to Linear regression. We have:

↳ or non-linear.

$$\underset{\sim}{X} = \underset{\sim}{f}(\underset{\sim}{b}) = \underset{\approx}{K}\,\underset{\sim}{b} \quad (\text{for linear})$$

we <u>know</u> that

$$\underset{x}{\Sigma}{}^2_X = \underset{\sim}{\nabla}\underset{\sim}{f}\ \underset{\approx}{\Sigma}{}^2_b \left(\underset{\sim}{\nabla}\underset{\sim}{f}\right)^T$$

If the b_i are all <u>indep</u> then this is:

$$\underset{\approx}{\Sigma}{}^2_X = \sum_{i=1}^{W} \left(\frac{\partial \underset{\sim}{f}}{\partial b_i}\right)\left(\frac{\partial \underset{\sim}{f}}{\partial b_i}\right)^T \sigma^2_{b_i}$$

$$\approx \sum_{i=1}^{N} \underbrace{\left(\frac{\underset{\sim}{f}(\underset{\sim}{\mu}_b + \sigma_{b_i} \hat{\underset{\sim}{e}}_i) - f(\underset{\sim}{\mu}_b)}{\sigma_{b_i}} \right)}_{\text{↳ numerical derivative}} \left(\quad \right)^T \sigma^2_{b_i}$$

$$\approx \sum_{i=1}^{N} \left(f(\underset{\sim}{\mu}_b + \sigma_{b_i}\hat{\underset{\sim}{e}}_i) - f(\underset{\sim}{\mu}_b)\right)\left(\quad\right)^T$$

which is the same formula (for large N anyway) since $f_j = \bar{f}_j$ (without the jth pt) ***if*** the curve went right through the jth data point (no change).

The $\frac{N-1}{N}$ factor comes from deg. of freedom issues.

Run example

Note that, just like other methods, it will get the error wrong if there is any data covariance ⟹ It's assuming independence!

It does ***not*** require residuals to have same magnitude – but if they aren't the same, use weighted regression!

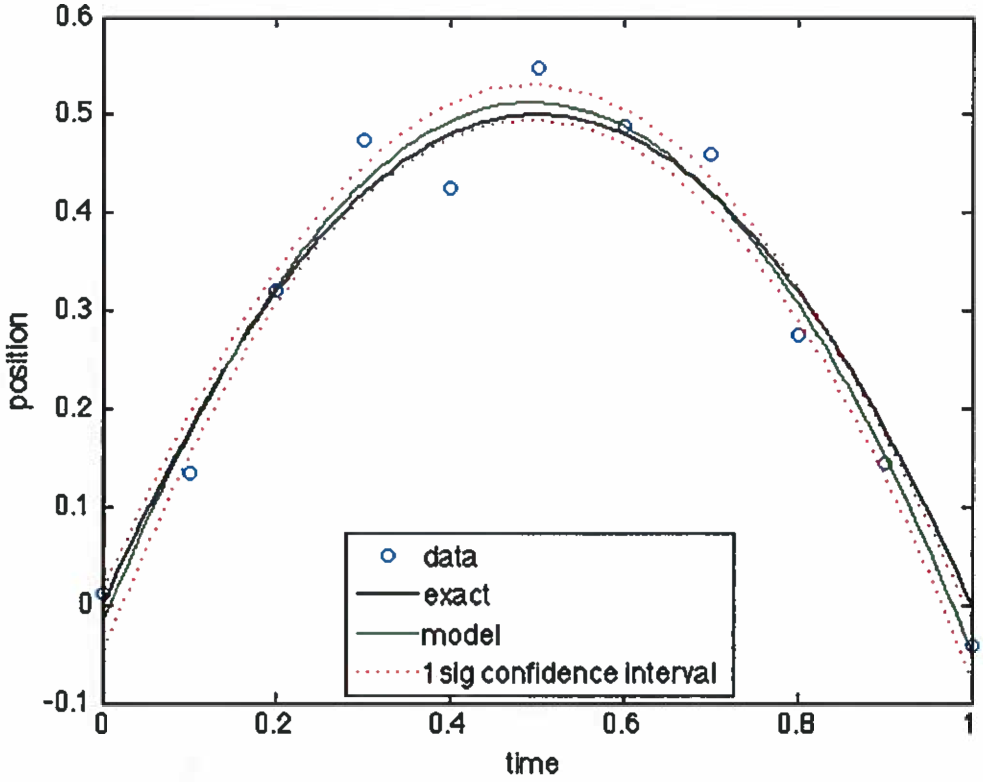
ex 19

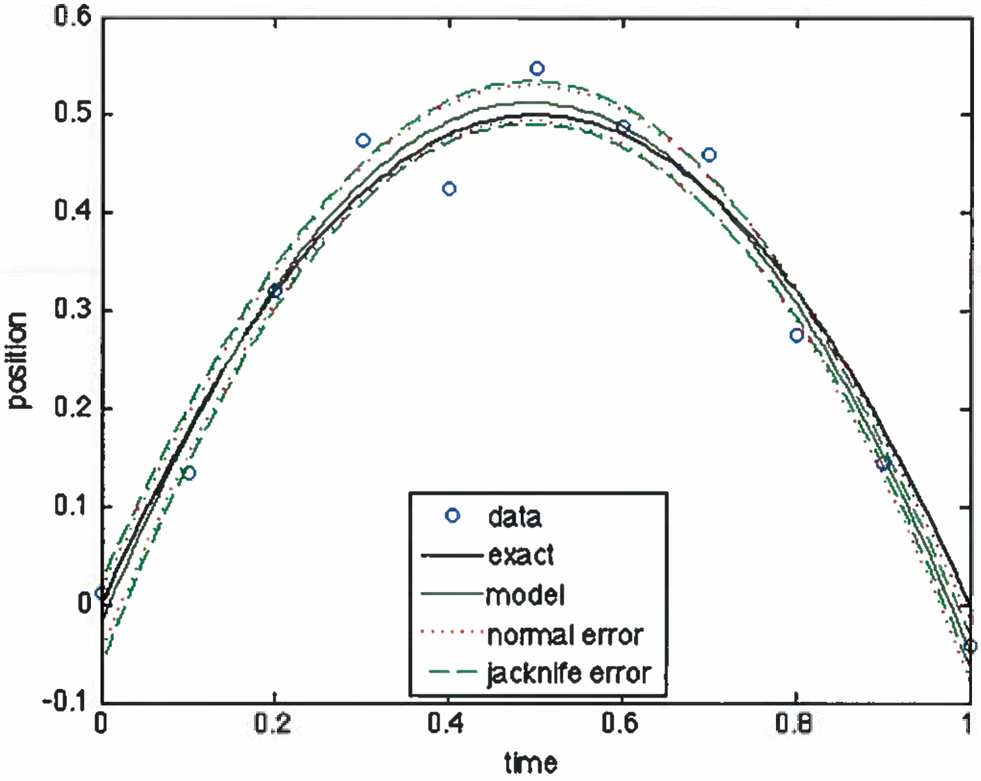
```
clear
format compact
echo on
%%The Jackknife
%In this example, we demonstrate the jackknife - a way of estimating the
%matrix of covariance of a set of fitted parameters without requiring
%modeling of the residuals directly.  We do this by solving the problem
%over and over, leaving out one data point each time.  We then calculate
%the matrix of covariance of the fitted values.  This has the advantage of
%not relying on the residuals being normally distributed, although if the
%residuals are not of uniform error the same issues with bias of the
%regression will occur.

%OK, we shall take as our example the "ball in air" problem we've looked at
%before:
t=[0:.1:1]';
%and we generate some "data":
xexact=[0,2,-2]'; %The exact values
a=[ones(size(t)),t,t.^2];
noise=0.05; %The amplitude of the noise
b=a*xexact+noise*randn(size(t)); %Our artificial data set.

pause

%%Basic Regression:
%We solve this using the usual regression formula:
k=inv(a'*a)*a';
x=k*b

%We calculate the error in the usual way:
r=a*x-b;
varb=r'*r/(length(r)-3); %Three degrees of freedom lost
sigb=varb^.5
%Which is (usually) close to the value of noise we put in
varx=k*varb*k'
%and in particular we have the 2-sigma confidence intervals of the x
%values:
xinterval=[x-2*diag(varx).^.5,x+2*diag(varx).^.5]
%which usually contains the exact values:
xexact
%Where probabilities are governed by the t-distribution:
prob=tcdf(2,length(b)-3)-tcdf(-2,length(b)-3)
%So the 2-sigma probability in the 90% range (depending on n-m).


pause

%Now let's plot this up. We want some smooth plotting:
tp=[0:.01:1]';
ap=[ones(size(tp)),tp,tp.^2];
bp=ap*x;
sigbp=diag(ap*varx*ap').^.5;
figure(1)
plot(t,b,'o',tp,ap*xexact,'k',tp,bp,tp,bp+sigbp,':r',tp,bp-sigbp,':r')
```

```
set(gca,'FontSize',14)
xlabel('time')
ylabel('position')
legend('data','exact','model','1sig confidence interval','Location','South')

pause

%%The Jackknife
%Now we look at how to estimate the matrix of covariance using the jackknife
%approach.  We simply leave off one of the data points and resolve for x.
%We then subtract it from the value obtained by looking at all the points,
%and determine the matrix of covariance:
[n m]=size(a);
xjksqall=zeros(m,m); %We initialize the array.
xjkall=zeros(m,1);
for i=1:n
    iall=[1:n];
    ikeep=find(iall~=i); %We keep all but the ith data point!
    ajk=a(ikeep,:);
    bjk=b(ikeep);
    xjk=ajk\bjk;
    xjksqall=xjksqall+xjk*xjk';
    xjkall=xjkall+xjk;
    echo off
end
echo on
%And we get the final result:
xjk=xjkall/n
%with the matrix of covariance:
varxjk=(xjksqall-n*xjk*xjk')*(n-1)/n
%And that generates the matrix, without having to assume anything about the
%matrix of covariance of b - other than assuming that the data points are
%independent, of course.

pause

%Let's compare this to the values we obtained using the error propagation
%formula:
var_ratio=varxjk./varx

%Which is (usually) close to one - it will change every time you run it. If
%you have a large number of data points it will converge exactly to one, as
%expected, but it takes about a thousand or so.

pause

%We can also add this to our plot:
bpjk=ap*xjk;
sigbpjk=diag(ap*varxjk*ap').^.5;
figure(2)
plot(t,b,'o',tp,ap*xexact,'k',tp,bpjk,tp,bp+sigbp,':r',tp,bpjk+sigbpjk,'--g',tp,bp-s
set(gca,'FontSize',14)
xlabel('time')
ylabel('position')
legend('data','exact','model','normal error','jacknife error','Location','South')
```

```
pause

%As a final note, the Jackknife will work both for linear and non-linear
%regression, although it does involve solving the problem n times.  It does
%not require determining the residuals (other than assuming independence),
%but you can't use it if you -require- one of the data points in the model
%fitting (such as cr0 in Tuesday's reaction engineering problem).  It also
%may be more prone to "strange results" if the number of data points is
%small, whereas the exact expressions (if the number of data points is
%still sufficient to estimate the variance in the data, or if you can get
%it in other ways) are less so.  For example, if you have a small number of
%data points and "leave off the one on the end", you will tend to get a
%much different value for fitting parameters, leading to (on average) an
%overestimate of the variance.
echo off
```

```
%%The Jackknife
%In this example, we demonstrate the jackknife - a way of estimating the
%matrix of covariance of a set of fitted parameters without requiring
%modeling of the residuals directly.  We do this by solving the problem
%over and over, leaving out one data point each time.  We then calculate
%the matrix of covariance of the fitted values.  This has the advantage of
%not relying on the residuals being normally distributed, although if the
%residuals are not of uniform error the same issues with bias of the
%regression will occur.
%OK, we shall take as our example the "ball in air" problem we've looked at
%before:
t=[0:.1:1]';
%and we generate some "data":
xexact=[0,2,-2]'; %The exact values
a=[ones(size(t)),t,t.^2];
noise=0.05; %The amplitude of the noise
b=a*xexact+noise*randn(size(t)); %Our artificial data set.

pause

%%Basic Regression:
%We solve this using the usual regression formula:
k=inv(a'*a)*a';
x=k*b
x =
   -0.0174
    2.1482
   -2.1776

%We calculate the error in the usual way:
r=a*x-b;
varb=r'*r/(length(r)-3); %Three degrees of freedom lost
sigb=varb^.5
sigb =
    0.0401
%Which is (usually) close to the value of noise we put in
varx=k*varb*k'
varx =
    0.0009   -0.0035    0.0028
   -0.0035    0.0202   -0.0187
    0.0028   -0.0187    0.0187
%and in particular we have the 2-sigma confidence intervals of the x
%values:
xinterval=[x-2*diag(varx).^.5,x+2*diag(varx).^.5]
xinterval =
   -0.0785    0.0437
    1.8640    2.4325
   -2.4514   -1.9038
%which usually contains the exact values:
xexact
xexact =
     0
     2
    -2
%Where probabilities are governed by the t-distribution:
prob=tcdf(2,length(b)-3)-tcdf(-2,length(b)-3)
prob =
    0.9195
%So the 2-sigma probability in the 90% range (depending on n-m).
pause

%Now let's plot this up. We want some smooth plotting:
tp=[0:.01:1]';
ap=[ones(size(tp)),tp,tp.^2];
bp=ap*x;
sigbp=diag(ap*varx*ap').^.5;
figure(1)
plot(t,b,'o',tp,ap*xexact,'k',tp,bp,tp,bp+sigbp,':r',tp,bp-sigbp,':r')
set(gca,'FontSize',14)
xlabel('time')
ylabel('position')
legend('data','exact','model','1sig confidence interval','Location','South')

pause

%%The Jackknife
%Now we look at how to estimate the matrix of covariance using the jackknife
%approach.  We simply leave off one of the data points and resolve for x.
%We then subtract it from the value obtained by looking at all the points,
%and determine the matrix of covariance:
[n m]=size(a);
xjksqall=zeros(m,m); %We initialize the array.
xjkall=zeros(m,1);
for i=1:n
    iall=[1:n];
    ikeep=find(iall~=i); %We keep all but the ith data point!
    ajk=a(ikeep,:);
    bjk=b(ikeep);
    xjk=ajk\bjk;
    xjksqall=xjksqall+xjk*xjk';
    xjkall=xjkall+xjk;
    echo off
%And we get the final result:
xjk=xjkall/n
xjk =
   -0.0189
    2.1549
   -2.1833
%with the matrix of covariance:
varxjk=(xjksqall-n*xjk*xjk')*(n-1)/n
varxjk =
    0.0018   -0.0067    0.0052
   -0.0067    0.0337   -0.0290
    0.0052   -0.0290    0.0258
%And that generates the matrix, without having to assume anything about the
%matrix of covariance of b - other than assuming that the data points are
%independent, of course.
pause

%Let's compare this to the values we obtained using the error propagation
%formula:
var_ratio=varxjk./varx
var_ratio =
    1.9536    1.8994    1.8468
    1.8994    1.6686    1.5467
    1.8468    1.5467    1.3782

%Which is (usually) close to one - it will change every time you run it. If
%you have a large number of data points it will converge exactly to one, as
%expected, but it takes about a thousand or so.
pause

%We can also add this to our plot:
bpjk=ap*xjk;
sigbpjk=diag(ap*varxjk*ap').^.5;
figure(2)
plot(t,b,'o',tp,ap*xexact,'k',tp,bpjk,tp,bp+sigbp,':r',tp,bpjk+sigbpjk,'--g',tp,bp-s
set(gca,'FontSize',14)
xlabel('time')
ylabel('position')
legend('data','exact','model','normal error','jacknife error','Location','South')

pause

%As a final note, the Jackknife will work both for linear and non-linear
%regression, although it does involve solving the problem n times.  It does
%not require determining the residuals (other than assuming independence),
%but you can't use it if you -require- one of the data points in the model
%fitting (such as cr0 in Tuesday's reaction engineering problem).  It also
%may be more prone to "strange results" if the number of data points is
%small, whereas the exact expressions (if the number of data points is
%still sufficient to estimate the variance in the data, or if you can get
%it in other ways) are less so.  For example, if you have a small number of
%data points and "leave off the one on the end", you will tend to get a
%much different value for fitting parameters, leading to (on average) an
%overestimate of the variance.
echo off
```





Published with MATLAB® 7.12

Now for the Bootstrap. This is simpler! We have N meas. Let's assume that these are rep. of population of all possible meas! Thus, if we draw N meas at random from inf. dist, can calculate parameters! We do this p times

we thus get p realizations of $\underset{\sim}{x}$!

$\hookrightarrow \underset{\sim}{x}^{(k)} = k^{th}$ realization

We have:

$$\Sigma^2_{\underset{\sim}{x}_i} = \frac{1}{p-1} \sum_{k=1}^{p} \left(\underset{\sim}{x}^{(k)} \underset{\sim}{x}^{(k)T} - \bar{\underset{\sim}{x}} \bar{\underset{\sim}{x}}^T \right)$$

Note that this runs into trouble if N is small or p is too big! We could just be sampling the same point N times! — prob is $\left(\frac{1}{N}\right)^{N-1}$ though...

ex 20

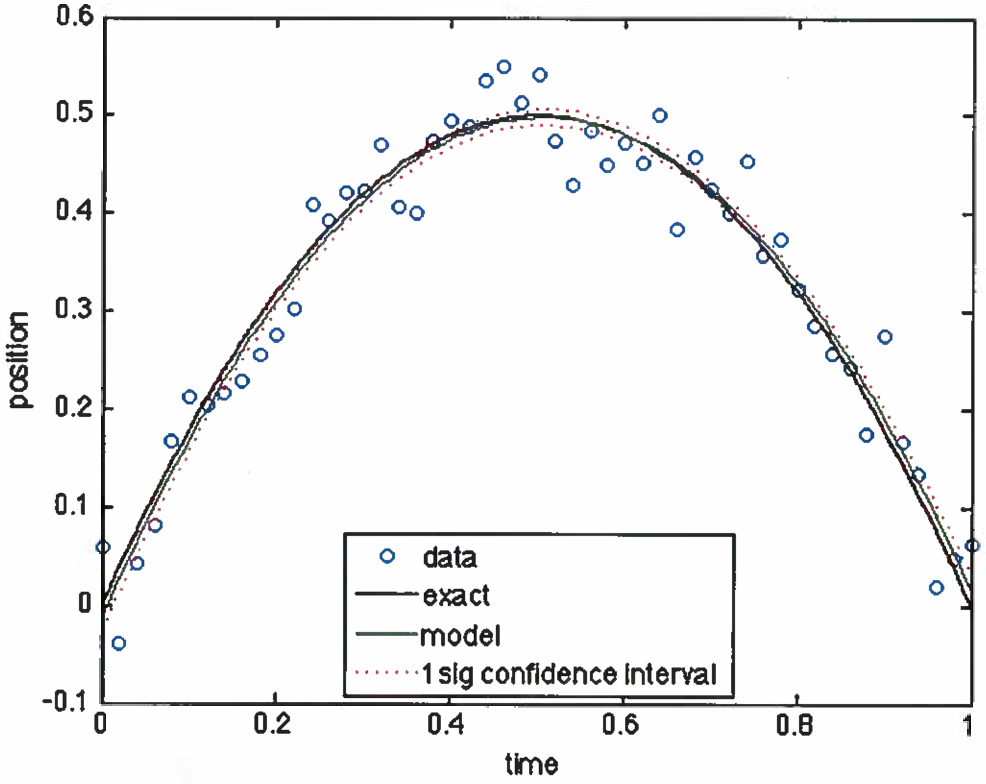
```
clear
format compact
echo on
%%The Bootstrap
%In this example, we demonstrate the bootstrap technique to estimate the
%matrix of covariance.  It is similar to the jacknife, except this time we
%analyze the data by taking a random sample of data drawn from an
%infinitely replicated data set.  The assumption is that our data is drawn
%from an infinite array of possible observations, and that the actual data
%is a good representation of this data set.

%We use the "ball in air" problem again:
t=[0:.02:1]';
%and we generate some "data":
xexact=[0,2,-2]'; %The exact values
a=[ones(size(t)),t,t.^2];
noise=0.05; %The amplitude of the noise
b=a*xexact+noise*randn(size(t)); %Our artificial data set.

pause

%%Basic Regression:
%We solve this using the usual regression formula:
k=inv(a'*a)*a';
x=k*b

%We calculate the error in the usual way:
r=a*x-b;
varb=r'*r/(length(r)-3); %Three degrees of freedom lost
sigb=varb^.5
%Which is (usually) close to the value of noise we put in
varx=k*varb*k'
%and in particular we have the 2-sigma confidence intervals of the x
%values:
xinterval=[x-2*diag(varx).^.5,x+2*diag(varx).^.5]
%which usually contains the exact values:
xexact
%Where probabilities are governed by the t-distribution:
prob=tcdf(2,length(b)-3)-tcdf(-2,length(b)-3)
%So the 2-sigma probability in the 90% range (depending on n-m).
%Now let's plot this up. We want some smooth plotting:
tp=[0:.01:1]';
ap=[ones(size(tp)),tp,tp.^2];
bp=ap*x;
sigbp=diag(ap*varx*ap').^.5;
figure(1)
plot(t,b,'o',tp,ap*xexact,'k',tp,bp,tp,bp+sigbp,':r',tp,bp-sigbp,':r')
set(gca,'FontSize',14)
xlabel('time')
ylabel('position')
legend('data','exact','model','1sig confidence interval','Location','South')

pause
```

```
%%The Bootstrap
%Now we look at how to estimate the matrix of covariance using the
%bootstrap approach.  We keep our matrix a, but we "sample" it nbs times.
%For this to work, nbs has to be a pretty large number.

[n m]=size(a);
xbssqall=zeros(m,m); %We initialize the array.
xbsall=zeros(m,1);
nbs=100;
for i=1:nbs
    ikeep=ceil(rand(n,1)*n); %The indices we keep this time around
    a_bs=a(ikeep,:);
    b_bs=b(ikeep);
    xbs=a_bs\b_bs;
    xbsall=xbsall+xbs;
    xbssqall=xbssqall+xbs*xbs';
    echo off
end
echo on
%Now we calculate the mean and matrix of covariance of these values:
xbs=xbsall/nbs
%And the covariance:
varxbs=(xbssqall-nbs*xbs*xbs')/(nbs-1)
%And that generates the matrix, without having to assume anything about the
%matrix of covariance of b - other than assuming that the data points are
%independent, of course.

pause

%Let's compare this to the values we obtained using the error propagation
%formula:
var_ratio=varxbs./varx

%Which is (usually) close to one - it will change every time you run it. If
%you have a large number of data points it will converge exactly to one, as
%expected, but it takes about a thousand or so.

pause

%We can also add this to our plot:
bpbs=ap*xbs;
sigbpbs=diag(ap*varxbs*ap').^.5;
figure(2)
plot(t,b,'o',tp,ap*xexact,'k',tp,bp,tp,bp+sigbp,':r',tp,bpbs+sigbpbs,'--g',tp,bp-sig
set(gca,'FontSize',14)
xlabel('time')
ylabel('position')
legend('data','exact','model','normal error','bootstrap error','Location','South')

pause

%The bootstrap is an interesting approach, but it suffers from the problem
%that you have to solve the problem a fairly large number of times to get a
%reasonable estimate of the variance.  Even more than the jacknife, it runs
%into trouble with small data sets: there is a finite probability that all
%n data points it picks will be the same one, leading to a degenerate
```

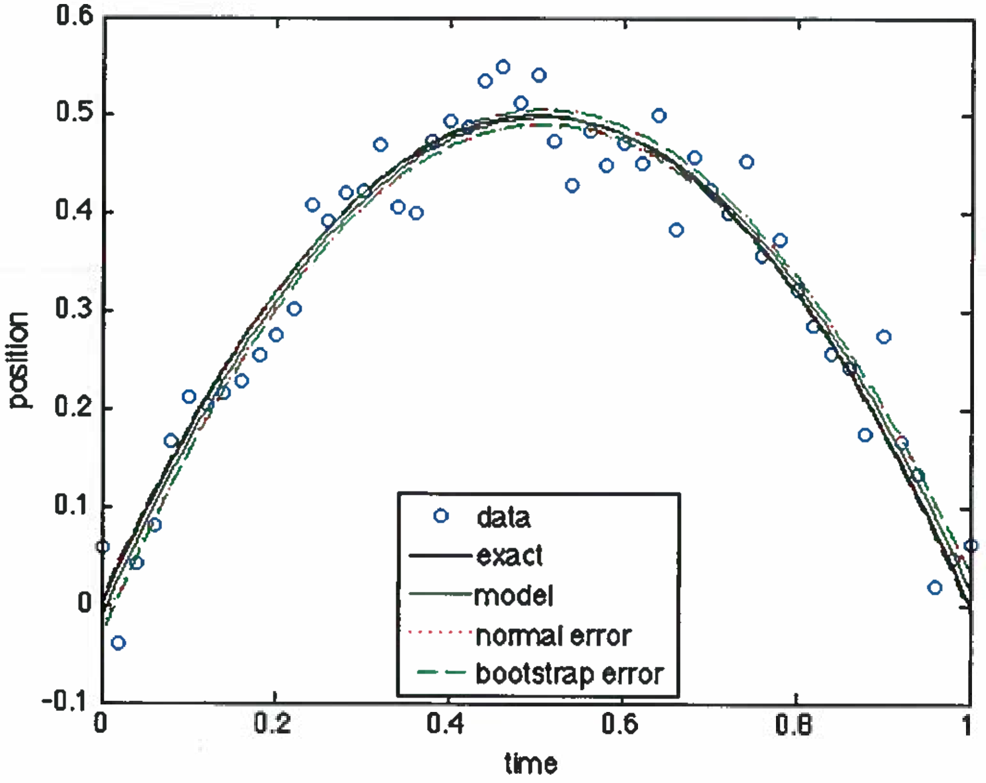
```
%matrix and lots of warning messages!  The variance calculated in this way
%is usually higher than the "correct" variance because of this effect.  If
%both n and nbs are large, however, the variances will be the same.
echo off
```

```
%%The Bootstrap
%In this example, we demonstrate the bootstrap technique to estimate the
%matrix of covariance.  It is similar to the jacknife, except this time we
%analyze the data by taking a random sample of data drawn from an
%infinitely replicated data set.  The assumption is that our data is drawn
%from an infinite array of possible observations, and that the actual data
%is a good representation of this data set.
%We use the "ball in air" problem again:
t=[0:.02:1]';
%and we generate some "data":
xexact=[0,2,-2]'; %The exact values
a=[ones(size(t)),t,t.^2];
noise=0.05; %The amplitude of the noise
b=a*xexact+noise*randn(size(t)); %Our artificial data set.

pause

%%Basic Regression:
%We solve this using the usual regression formula:
k=inv(a'*a)*a';
x=k*b
x =
   -0.0121
    2.0129
   -1.9847

%We calculate the error in the usual way:
r=a*x-b;
varb=r'*r/(length(r)-3); %Three degrees of freedom lost
sigb=varb^.5
sigb =
    0.0384
%Which is (usually) close to the value of noise we put in
varx=k*varb*k'
varx =
    0.0002   -0.0010    0.0008
   -0.0010    0.0051   -0.0048
    0.0008   -0.0048    0.0048
%and in particular we have the 2-sigma confidence intervals of the x
%values:
xinterval=[x-2*diag(varx).^.5,x+2*diag(varx).^.5]
xinterval =
   -0.0431    0.0189
    1.8695    2.1562
   -2.1234   -1.8461
%which usually contains the exact values:
xexact
xexact =
     0
     2
    -2
%Where probabilities are governed by the t-distribution:
prob=tcdf(2,length(b)-3)-tcdf(-2,length(b)-3)
prob =
    0.9488
%So the 2-sigma probability in the 90% range (depending on n-m).
%Now let's plot this up. We want some smooth plotting:
tp=[0:.01:1]';
ap=[ones(size(tp)),tp,tp.^2];
bp=ap*x;
sigbp=diag(ap*varx*ap').^.5;
figure(1)
plot(t,b,'o',tp,ap*xexact,'k',tp,bp,tp,bp+sigbp,':r',tp,bp-sigbp,':r')
set(gca,'FontSize',14)
xlabel('time')
ylabel('position')
legend('data','exact','model','1sig confidence interval','Location','South')

pause

%%The Bootstrap
%Now we look at how to estimate the matrix of covariance using the
%bootstrap approach.  We keep our matrix a, but we "sample" it nbs times.
%For this to work, nbs has to be a pretty large number.
[n m]=size(a);
xbssqall=zeros(m,m); %We initialize the array.
xbsall=zeros(m,1);
nbs=100;
for i=1:nbs
    ikeep=ceil(rand(n,1)*n); %The indices we keep this time around
    a_bs=a(ikeep,:);
    b_bs=b(ikeep);
    xbs=a_bs\b_bs;
    xbsall=xbsall+xbs;
    xbssqall=xbssqall+xbs*xbs';
    echo off
%Now we calculate the mean and matrix of covariance of these values:
xbs=xbsall/nbs
xbs =
   -0.0122
    2.0178
   -1.9922
%And the covariance:
varxbs=(xbssqall-nbs*xbs*xbs')/(nbs-1)
varxbs =
    0.0004   -0.0015    0.0012
   -0.0015    0.0069   -0.0063
    0.0012   -0.0063    0.0063
%And that generates the matrix, without having to assume anything about the
%matrix of covariance of b - other than assuming that the data points are
%independent, of course.
pause

%Let's compare this to the values we obtained using the error propagation
%formula:
var_ratio=varxbs./varx
```

```
var_ratio =
    1.6126    1.5491    1.5653
    1.5491    1.3368    1.3211
    1.5653    1.3211    1.3148

%Which is (usually) close to one - it will change every time you run it. If
%you have a large number of data points it will converge exactly to one, as
%expected, but it takes about a thousand or so.
pause

%We can also add this to our plot:
bpbs=ap*xbs;
sigbpbs=diag(ap*varxbs*ap').^.5;
figure(2)
plot(t,b,'o',tp,ap*xexact,'k',tp,bp,tp,bp+sigbp,':r',tp,bpbs+sigbpbs,'--g',tp,bp-sig
set(gca,'FontSize',14)
xlabel('time')
ylabel('position')
legend('data','exact','model','normal error','bootstrap error','Location','South')

pause

%The bootstrap is an interesting approach, but it suffers from the problem
%that you have to solve the problem a fairly large number of times to get a
%reasonable estimate of the variance.  Even more than the jacknife, it runs
%into trouble with small data sets: there is a finite probability that all
%n data points it picks will be the same one, leading to a degenerate
%matrix and lots of warning messages!  The variance calculated in this way
%is usually higher than the "correct" variance because of this effect.  If
%both n and nbs are large, however, the variances will be the same.
echo off
```



0.6
0.5
0.4
0.3
0.2
0.1
0
-0.1
position
0
0.2
0.4
0.6
0.8
1
time
data
exact
model
1 sig confidence interval



0.6
0.5
0.4
0.3
0.2
0.1
0
-0.1
position
0
0.2
0.4
0.6
0.8
1
time
data
exact
model
normal error
bootstrap error

For really large data sets, undersampling works! We have N meas.; divide into p sets & compute $\underset{\sim}{x}$ from each! (e.g., $\underset{\sim}{x}^{(K)}$ is $\underset{\sim}{x}$ from K^{th} set.

Thus:

$$\underset{\sim}{\Sigma}_x^2 = \frac{1}{p}\,\frac{1}{p-1}\sum_{K=1}^{p}\left(\underset{\sim}{x}^{(K)}\underset{\sim}{x}^{(K)T} - \bar{\underset{\sim}{x}}\,\bar{\underset{\sim}{x}}^{T}\right)$$

extra factor of p since we want error in average of p subsets. We didn't get that for Bootstrap because they were all from the same data set resampled!

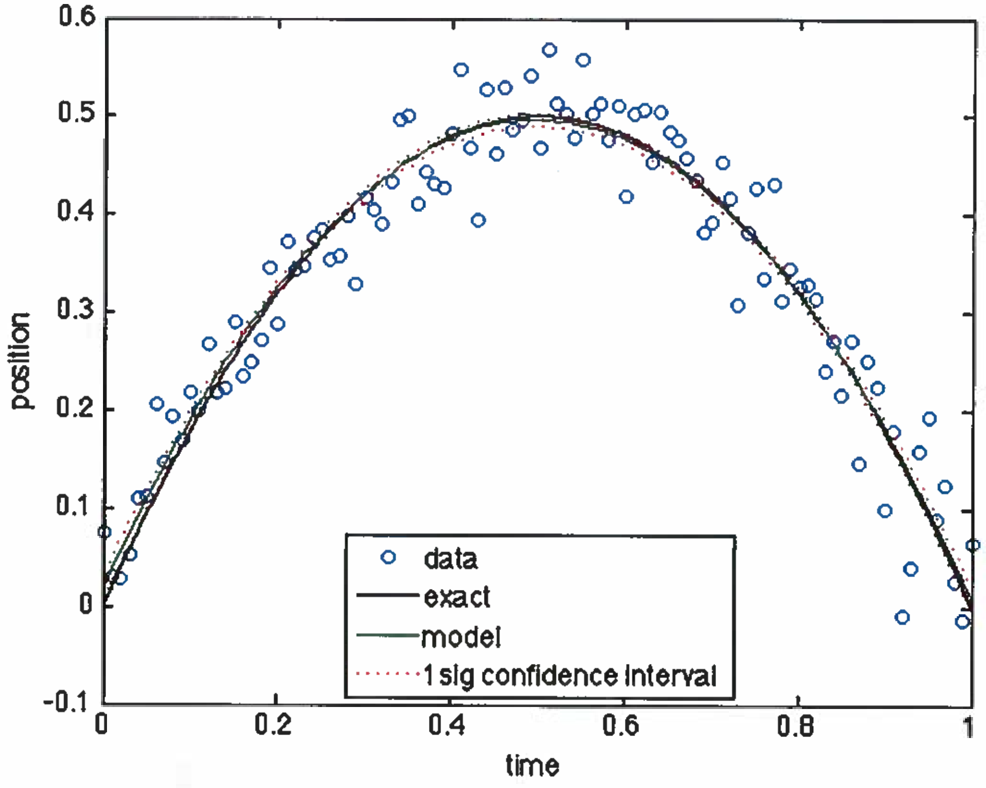
ex 21

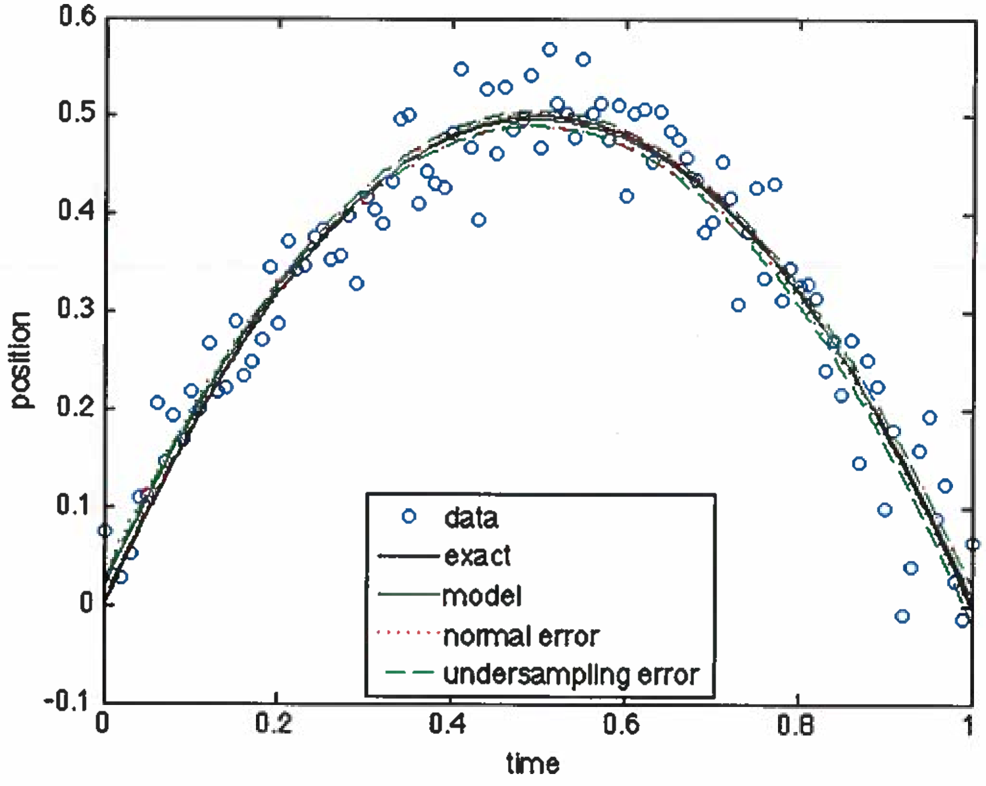
```
clear
format compact
echo on
%%Undersampling
%In this example, we look at the technique of undersampling.  This is
%really only appropriate for very large data sets.  The idea is that you
%break your dataset into many subsets, calculate the parameters for each,
%and then take the mean and standard deviations of these calculated values.
%Because you need a significant number of data points in each set (for any
%reasonable statistics) and a significant number of subsets, the total
%number of data points has to be really large!

%We use the "ball in air" problem again:
t=[0:.01:1]';
%and we generate some "data":
xexact=[0,2,-2]'; %The exact values
a=[ones(size(t)),t,t.^2];
noise=0.05; %The amplitude of the noise
b=a*xexact+noise*randn(size(t)); %Our artificial data set.

pause

%%Basic Regression:
%We solve this using the usual regression formula:
k=inv(a'*a)*a';
x=k*b

%We calculate the error in the usual way:
r=a*x-b;
varb=r'*r/(length(r)-3); %Three degrees of freedom lost
sigb=varb^.5
%Which is (usually) close to the value of noise we put in
varx=k*varb*k'
%and in particular we have the 2-sigma confidence intervals of the x
%values:
xinterval=[x-2*diag(varx).^.5,x+2*diag(varx).^.5]
%which usually contains the exact values:
xexact
%Where probabilities are governed by the t-distribution:
prob=tcdf(2,length(b)-3)-tcdf(-2,length(b)-3)
%So the 2-sigma probability in the 90% range (depending on n-m).
%Now let's plot this up. We want some smooth plotting:
tp=[0:.01:1]';
ap=[ones(size(tp)),tp,tp.^2];
bp=ap*x;
sigbp=diag(ap*varx*ap').^.5;
figure(1)
plot(t,b,'o',tp,ap*xexact,'k',tp,bp,tp,bp+sigbp,':r',tp,bp-sigbp,':r')
set(gca,'FontSize',14)
xlabel('time')
ylabel('position')
legend('data','exact','model','1sig confidence interval','Location','South')

pause
```

```
%%Undersampling
%Now we estimate our fitting parameters by undersampling.
[n m]=size(a);
xussqall=zeros(m,m); %We initialize the array.
xusall=zeros(m,1);
p=10
for i=1:p
    ikeep=[i:p:n]; %The indices we keep this time around
    a_us=a(ikeep,:);
    b_us=b(ikeep);
    xus=a_us\b_us;
    xusall=xusall+xus;
    xussqall=xussqall+xus*xus';
    echo off
end
echo on
%Now we calculate the mean and matrix of covariance of these values:
xus=xusall/p
%And the covariance:
varxus=(xussqall-p*xus*xus')/(p-1)/p
%And that generates the matrix, without having to assume anything about the
%matrix of covariance of b - other than assuming that the data points are
%independent, of course.  Note that we are getting the matrix of covariance
%of the -mean- of p samples of our dataset.

pause

%Let's compare this to the values we obtained using the error propagation
%formula:
var_ratio=varxus./varx

%Which is (usually) close to one - it will change every time you run it. If
%you have a large number of data points it will converge exactly to one, as
%expected, but it takes about a thousand or so.

pause

%We can also add this to our plot:
bpus=ap*xus;
sigbpus=diag(ap*varxus*ap').^.5;
figure(2)
plot(t,b,'o',tp,ap*xexact,'k',tp,bp,tp,bp+sigbp,':r',tp,bpus+sigbpus,'--g',tp,bp-sig
set(gca,'FontSize',14)
xlabel('time')
ylabel('position')
legend('data','exact','model','normal error','undersampling error','Location','Soutl

pause

%Undersampling is a very simple way of getting at the variance of
%measurements, but it does require a very large number of datapoints.  You
%certainly can't do it with just a few!  Like the other resampling methods,
%it works as well for both linear and non-linear regression problems, and
%will also behave well for non-normal residuals (although it is
%questionable whether non-weighted regression is appropriate if your
%residuals are not uniform).  You can also use these techniques to estimate
%the -distribution- of the fitting parameters: statistics beyond mean and
%variance, as fitting parameters are usually not normally distributed as
%well.
echo off
```

```
%%Undersampling
%In this example, we look at the technique of undersampling.  This is
%really only appropriate for very large data sets.  The idea is that you
%break your dataset into many subsets, calculate the parameters for each,
%and then take the mean and standard deviations of these calculated values.
%Because you need a significant number of data points in each set (for any
%reasonable statistics) and a significant number of subsets, the total
%number of data points has to be really large!
%We use the "ball in air" problem again:
t=[0:.01:1]';
%and we generate some "data":
xexact=[0,2,-2]'; %The exact values
a=[ones(size(t)),t,t.^2];
noise=0.05; %The amplitude of the noise
b=a*xexact+noise*randn(size(t)); %Our artificial data set.

pause

%%Basic Regression:
%We solve this using the usual regression formula:
k=inv(a'*a)*a';
x=k*b
x =
    0.0191
    1.9202
   -1.9315

%We calculate the error in the usual way:
r=a*x-b;
varb=r'*r/(length(r)-3); %Three degrees of freedom lost
sigb=varb^.5
sigb =
    0.0440
%Which is (usually) close to the value of noise we put in
varx=k*varb*k'
varx =
    0.0002   -0.0007    0.0005
   -0.0007    0.0035   -0.0033
    0.0005   -0.0033    0.0033
%and in particular we have the 2-sigma confidence intervals of the x
%values:
xinterval=[x-2*diag(varx).^.5,x+2*diag(varx).^.5]
xinterval =
   -0.0067    0.0448
    1.8013    2.0391
   -2.0466   -1.8165
%which usually contains the exact values:
xexact
xexact =
     0
     2
    -2
%Where probabilities are governed by the t-distribution:
prob=tcdf(2,length(b)-3)-tcdf(-2,length(b)-3)
prob =
    0.9517
%So the 2-sigma probability in the 90% range (depending on n-m).
%Now let's plot this up. We want some smooth plotting:
tp=[0:.01:1]';
ap=[ones(size(tp)),tp,tp.^2];
bp=ap*x;
sigbp=diag(ap*varx*ap').^.5;
figure(1)
plot(t,b,'o',tp,ap*xexact,'k',tp,bp,tp,bp+sigbp,':r',tp,bp-sigbp,':r')
set(gca,'FontSize',14)
xlabel('time')
ylabel('position')
legend('data','exact','model','1sig confidence interval','Location','South')

pause

%%Undersampling
%Now we estimate our fitting parameters by undersampling.
[n m]=size(a);
xussqall=zeros(m,m); %We initialize the array.
xusall=zeros(m,1);
p=10
p =
    10
for i=1:p
    ikeep=[i:p:n]; %The indices we keep this time around
    a_us=a(ikeep,:);
    b_us=b(ikeep);
    xus=a_us\b_us;
    xusall=xusall+xus;
    xussqall=xussqall+xus*xus';
    echo off
%Now we calculate the mean and matrix of covariance of these values:
xus=xusall/p
xus =
    0.0160
    1.9405
   -1.9552
%And the covariance:
varxus=(xussqall-p*xus*xus')/(p-1)/p
varxus =
    0.0001   -0.0005    0.0005
   -0.0005    0.0031   -0.0032
    0.0005   -0.0032    0.0035
%And that generates the matrix, without having to assume anything about the
%matrix of covariance of b - other than assuming that the data points are
%independent, of course.  Note that we are getting the matrix of covariance
%of the -mean- of p samples of our dataset.
pause
```

```
%Let's compare this to the values we obtained using the error propagation
%formula:
var_ratio=varxus./varx
var_ratio =
    0.4948    0.6851    0.8419
    0.6851    0.8844    0.9665
    0.8419    0.9665    1.0511

%Which is (usually) close to one - it will change every time you run it. If
%you have a large number of data points it will converge exactly to one, as
%expected, but it takes about a thousand or so.
pause

%We can also add this to our plot:
bpus=ap*xus;
sigbpus=diag(ap*varxus*ap').^.5;
figure(2)
plot(t,b,'o',tp,ap*xexact,'k',tp,bp,tp,bp+sigbp,':r',tp,bpus+sigbpus,'--g',tp,bp-sig
set(gca,'FontSize',14)
xlabel('time')
ylabel('position')
legend('data','exact','model','normal error','undersampling error','Location','Sout

pause

%Undersampling is a very simple way of getting at the variance of
%measurements, but it does require a very large number of datapoints.  You
%certainly can't do it with just a few!  Like the other resampling methods,
%it works as well for both linear and non-linear regression problems, and
%will also behave well for non-normal residuals (although it is
%questionable whether non-weighted regression is appropriate if your
%residuals are not uniform).  You can also use these techniques to estimate
%the -distribution- of the fitting parameters: statistics beyond mean and
%variance, as fitting parameters are usually not normally distributed as
%well.
echo off
```





Published with MATLAB® 7.12

Note that *none* of these methods required knowing σ_b^2 (or even if all $\sigma_{b_i}^2$ were the same! We *do* assume independence, though! You also need decent sized data sets (or huge, for undersampling) to make it work.

For Monte Carlo, you can work w/ *small* data sets, but you need to *know* σ_b!

If we *know* σ_b, simulate by adding in *noise* $\sim \sigma_b$ & recalc. x!

We get $$\underset{\approx}{\Sigma}_x^2 = \frac{1}{p-1}\sum_{K=1}^{p}\left(\underset{\sim}{x}^{(K)}\underset{\sim}{x}^{(K)T} - \bar{\underset{\sim}{x}}\,\bar{\underset{\sim}{x}}^{T}\right)$$

(same as bootstrap)

We can do this for a really large # of trials! You can get both $\underset{\sim}{\bar{x}}$, $\underset{\approx}{\Sigma}_x^2$ but *also* the distribution! Note that if $\underset{\sim}{b}$ is not normally distributed, you can simulate just as readily! simply add same noise as distrib.

Only 2 downsides

1) Requires a *lot* of comp.

2) You *have* to know σ_b — but can est. from residual

Also – answer changes every time you run it unless p is huge (same prob. as w/ Bootstrap) ex22

```
clear
format compact
echo on
%%Monte Carlo Simulation
%In this example, we look at the approach of Monte Carlo simulation for
%error estimation.  This method requires knowledge of the residual (error)
%in the data set.  If we know this (via calculation or via other
%experiments) we can create "artificial" data sets with data that has an
%added error characterized by this residual.  We determine the fitting
%parameters for these, average them together, and compute the statistics.

%We use the "ball in air" problem again:
t=[0:.1:1]';
%and we generate some "data":
xexact=[0,2,-2]'; %The exact values
a=[ones(size(t)),t,t.^2];
noise=0.05; %The amplitude of the noise
b=a*xexact+noise*randn(size(t)); %Our artificial data set.

pause

%%Basic Regression:
%We solve this using the usual regression formula:
k=inv(a'*a)*a';
x=k*b

%We calculate the error in the usual way:
r=a*x-b;
varb=r'*r/(length(r)-3); %Three degrees of freedom lost
sigb=varb^.5
%Which is (usually) close to the value of noise we put in
varx=k*varb*k'
%and in particular we have the 2-sigma confidence intervals of the x
%values:
xinterval=[x-2*diag(varx).^.5,x+2*diag(varx).^.5]
%which usually contains the exact values:
xexact
%Where probabilities are governed by the t-distribution:
prob=tcdf(2,length(b)-3)-tcdf(-2,length(b)-3)
%So the 2-sigma probability in the 90% range (depending on n-m).
%Now let's plot this up. We want some smooth plotting:
tp=[0:.01:1]';
ap=[ones(size(tp)),tp,tp.^2];
bp=ap*x;
sigbp=diag(ap*varx*ap').^.5;
figure(1)
plot(t,b,'o',tp,ap*xexact,'k',tp,bp,tp,bp+sigbp,':r',tp,bp-sigbp,':r')
set(gca,'FontSize',14)
xlabel('time')
ylabel('position')
legend('data','exact','model','1sig confidence interval','Location','South')

pause
```

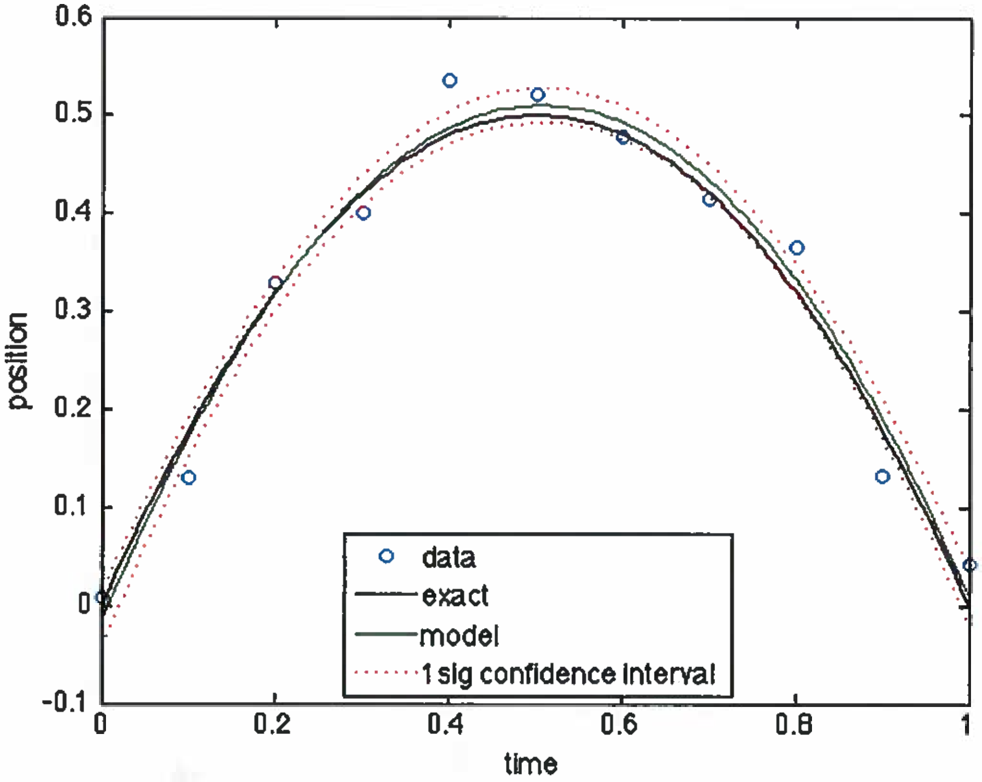
```
%%Monte Carlo
%Now we estimate our fitting parameters from Monte Carlo simulation.  We
%want to use a fair number of samples to get reasonable statistics.
[n m]=size(a);
xmcsqall=zeros(m,m); %We initialize the array.
xmcall=zeros(m,1);
nmc=100
for i=1:nmc
    bmc=b+randn(n,1)*sigb; %We add in noise based on our residuals
    xmc=k*bmc; %We use all the same times, so k doesn't change.
    xmcall=xmcall+xmc;
    xmcsqall=xmcsqall+xmc*xmc';
    echo off
end
echo on
%Now we calculate the mean and matrix of covariance of these values:
xmc=xmcall/nmc
%And the covariance:
varxmc=(xmcsqall-nmc*xmc*xmc')/(nmc-1)
%And that generates the matrix.

pause

%Let's compare this to the values we obtained using the error propagation
%formula:
var_ratio=varxmc./varx

%Which is (usually) close to one - it will change every time you run it.
%You don't need a large number of data points, but you do need to have a
%large number of monte carlo simulation runs!

pause

%We can also add this to our plot:
bpmc=ap*xmc;
sigbpmc=diag(ap*varxmc*ap').^.5;
figure(2)
plot(t,b,'o',tp,ap*xexact,'k',tp,bp,tp,bp+sigbp,':r',tp,bpmc+sigbpmc,'--g',tp,bp-sig
set(gca,'FontSize',14)
xlabel('time')
ylabel('position')
legend('data','exact','model','normal error','monte carlo error','Location','South')

pause

%Monte Carlo Simulation is an easy technique for estimating the statistics
%of the fitting parameters.  Unlike other resampling techniques, however,
%it does require knowledge of the residual: exactly the same information
%required of the normal error propagation formulas.  The computational
%requirement is much higher than that of other methods (at least 100
%simulations for decent statistics), and yields the exact same matrix
%of covariance (if the number of simulations is high enough).  There are
%two advantages: it does not require taking any gradients (this may be
%significant in non-linear regression, but is irrelevant in linear
%regression), and it can yield the distribution of the fitting parameters
%(more than just the covariance).  Of the resampling approaches, it is the
```

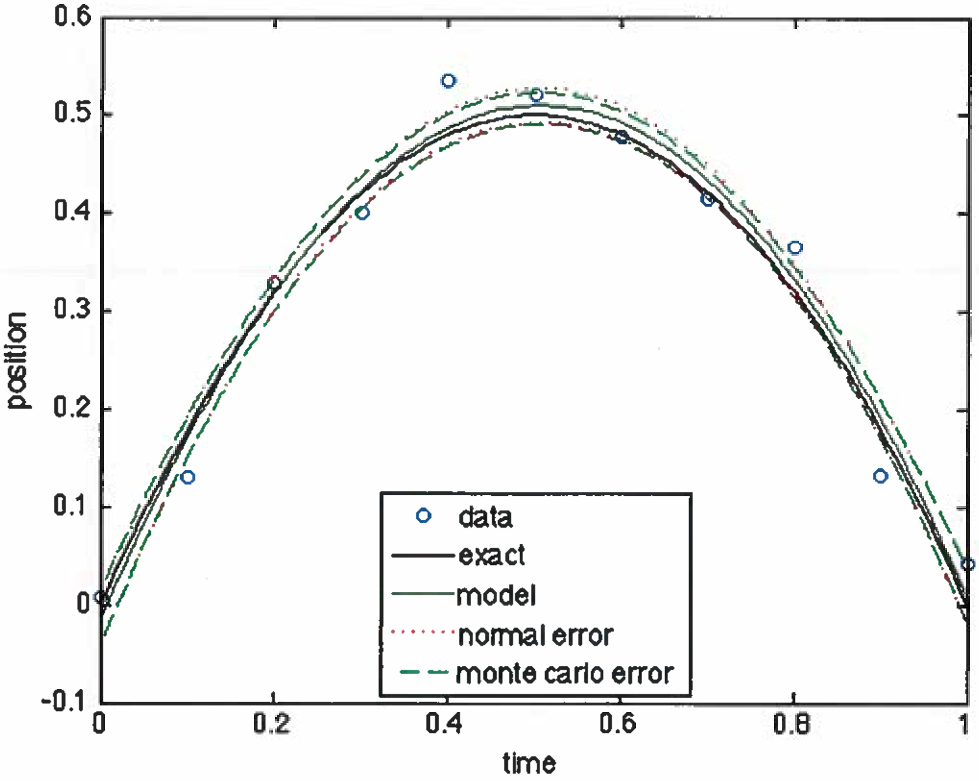
```
%only one which is suitable for small data sets.
echo off
```

```
%%Monte Carlo Simulation
%In this example, we look at the approach of Monte Carlo simulation for
%error estimation.  This method requires knowledge of the residual (error)
%in the data set.  If we know this (via calculation or via other
%experiments) we can create "artificial" data sets with data that has an
%added error characterized by this residual.  We determine the fitting
%parameters for these, average them together, and compute the statistics.
%We use the "ball in air" problem again:
t=[0:.1:1]';
%and we generate some "data":
xexact=[0,2,-2]'; %The exact values
a=[ones(size(t)),t,t.^2];
noise=0.05; %The amplitude of the noise
b=a*xexact+noise*randn(size(t)); %Our artificial data set.

pause

%%Basic Regression:
%We solve this using the usual regression formula:
k=inv(a'*a)*a';
x=k*b
x =
   -0.0145
    2.0749
   -2.0510

%We calculate the error in the usual way:
r=a*x-b;
varb=r'*r/(length(r)-3); %Three degrees of freedom lost
sigb=varb^.5
sigb =
    0.0382
%Which is (usually) close to the value of noise we put in
varx=k*varb*k'
varx =
    0.0008   -0.0032    0.0026
   -0.0032    0.0183   -0.0170
    0.0026   -0.0170    0.0170
%and in particular we have the 2-sigma confidence intervals of the x
%values:
xinterval=[x-2*diag(varx).^.5,x+2*diag(varx).^.5]
xinterval =
   -0.0728    0.0437
    1.8041    2.3458
   -2.3119   -1.7901
%which usually contains the exact values:
xexact
xexact =
     0
     2
    -2
%Where probabilities are governed by the t-distribution:
```

```
prob=tcdf(2,length(b)-3)-tcdf(-2,length(b)-3)
prob =
    0.9195
%So the 2-sigma probability in the 90% range (depending on n-m).
%Now let's plot this up. We want some smooth plotting:
tp=[0:.01:1]';
ap=[ones(size(tp)),tp,tp.^2];
bp=ap*x;
sigbp=diag(ap*varx*ap').^.5;
figure(1)
plot(t,b,'o',tp,ap*xexact,'k',tp,bp,tp,bp+sigbp,':r',tp,bp-sigbp,':r')
set(gca,'FontSize',14)
xlabel('time')
ylabel('position')
legend('data','exact','model','1sig confidence interval','Location','South')

pause

%%Monte Carlo
%Now we estimate our fitting parameters from Monte Carlo simulation.  We
%want to use a fair number of samples to get reasonable statistics.
[n m]=size(a);
xmcsqall=zeros(m,m); %We initialize the array.
xmcall=zeros(m,1);
nmc=100
nmc =
   100
for i=1:nmc
    bmc=b+randn(n,1)*sigb; %We add in noise based on our residuals
    xmc=k*bmc; %We use all the same times, so k doesn't change.
    xmcall=xmcall+xmc;
    xmcsqall=xmcsqall+xmc*xmc';
    echo off
%Now we calculate the mean and matrix of covariance of these values:
xmc=xmcall/nmc
xmc =
   -0.0121
    2.0532
   -2.0312
%And the covariance:
varxmc=(xmcsqall-nmc*xmc*xmc')/(nmc-1)
varxmc =
    0.0008   -0.0026    0.0019
   -0.0026    0.0143   -0.0131
    0.0019   -0.0131    0.0133
%And that generates the matrix.
pause

%Let's compare this to the values we obtained using the error propagation
%formula:
var_ratio=varxmc./varx
var_ratio =
    0.9054    0.7977    0.7268
    0.7977    0.7782    0.7676
    0.7268    0.7676    0.7792
```

```
%Which is (usually) close to one - it will change every time you run it.
%You don't need a large number of data points, but you do need to have a
%large number of monte carlo simulation runs!
pause

%We can also add this to our plot:
bpmc=ap*xmc;
sigbpmc=diag(ap*varxmc*ap').^.5;
figure(2)
plot(t,b,'o',tp,ap*xexact,'k',tp,bp,tp,bp+sigbp,':r',tp,bpmc+sigbpmc,'--g',tp,bp-sig
set(gca,'FontSize',14)
xlabel('time')
ylabel('position')
legend('data','exact','model','normal error','monte carlo error','Location','South')

pause

%Monte Carlo Simulation is an easy technique for estimating the statistics
%of the fitting parameters.  Unlike other resampling techniques, however,
%it does require knowledge of the residual: exactly the same information
%required of the normal error propagation formulas.  The computational
%requirement is much higher than that of other methods (at least 100
%simulations for decent statistics), and yields the exact same matrix
%of covariance (if the number of simulations is high enough).  There are
%two advantages: it does not require taking any gradients (this may be
%significant in non-linear regression, but is irrelevant in linear
%regression), and it can yield the distribution of the fitting parameters
%(more than just the covariance).  Of the resampling approaches, it is the
%only one which is suitable for small data sets.
echo off
```





Published with MATLAB® 7.12

As a final note on statistics, we have assumed throughout that the deviation between the observations and the model is random! This is, in general,

Not True!

Thus, you tend to underestimate your error by ignoring the non-zero covariance in your data.

It is important to always plot your residuals to see if there is a systematic deviation.

A systematic deviation means that something is missing from your model and/or there is a systematic error in your data.

Never forget that for large N parameter error is dominated by systematic error!!

Non-Linear Equations

So far we have focussed on <u>linear equations</u>. This is because we can <u>solve</u> these equations in a finite # of steps!

$$\underset{\approx}{A}\,\underset{\sim}{x} = \underset{\sim}{b} \quad \text{is solved in } O(n^3) \text{ steps!}$$

Often, however, our physical problem is non-linear!

Example: Suppose we are trying to fit some reaction rate data as a $f^{\underline{n}}(T)$. If we have a <u>single irreversible</u> reaction:

$$K = \underset{\substack{\uparrow \\ \text{pre-exp. factor}}}{a}\; e^{-E_a/KT}$$

Boltzmann's cst

activation energy

If we have a series of measured rxn rates $K_i(T_i)$ we can solve for a & E_a using linear regression:

Transform eq'n:

$$\underset{\overset{\uparrow}{b}}{\ln K} = \underset{\overset{\uparrow}{x_1}}{\ln \alpha} - \underset{\overset{\uparrow}{x_2}}{\frac{E_a}{K}} \frac{1}{T}$$

So $$\min_{\underset{\sim}{x}} \left\| \left(\underset{\approx}{A} \, \underset{\sim}{x} - \underset{\sim}{b} \right) \right\|_2$$

where $b \equiv \begin{pmatrix} \ln K_1 \\ \vdots \\ \ln K_n \end{pmatrix}$, $\underset{\approx}{A} = \begin{pmatrix} 1 & \frac{1}{T_1} \\ \vdots & \vdots \\ 1 & \frac{1}{T_n} \end{pmatrix}$

So you know how to solve this. Suppose we have a more complex problem in which a substance can follow 2 rxn pathways. In this case:

$$K = a_1 e^{-E_{a_1}/KT} + a_2 e^{-E_{a_2}/KT}$$

You can't linearize this!

We can still set it up as a least squares problem:

$$\min_{\underset{\sim}{x}} \| K_i - K(T_i) \|_2 \equiv S(\underset{\sim}{x})$$

where $\underset{\sim}{x} \equiv \begin{pmatrix} a_1 \\ Ea_1 \\ a_2 \\ Ea_2 \end{pmatrix}$

We can determine the minimum by requiring:

$$\frac{\partial S(\underset{\sim}{x})}{\partial \underset{\sim}{x}} = 0 \quad : \text{4 eq'ns w/ 4 unknowns}$$

But this is a *non-linear problem*

Non-linear problems are *nasty* because:

1) it may not have a unique solution! You may have multiple roots, local minima, etc.
For linear problems (non-singular) you

are <u>guaranteed</u> a unique solution!

2) For linear systems, you can solve it in a finite # of steps ($O(n^3)$)
For non-linear problems there is <u>no</u> guarantee - sometimes algorithms will fail to converge on <u>any</u> sol^n!

Let's focus on the problem $f(x)=0$

(we'll generalize this to the system $\underset{\sim}{f}(\underset{\sim}{x})=0$ later)

Let x^* satisfy $f(x^*)=0$

We will say $\bar{x}$ "solves" $f(x)=0$

if:

$$|f(\bar{x})| \approx 0 \quad \text{or} \quad |\bar{x}-x^*| \approx 0$$

less than some set tolerance

we need this dual def$^{\underline{n}}$ because $f(x)$ may have a steep slope near x^*

The approaches to solving $f(x)=0$ are iterative - we obtain a series (sequence) of approximations to the solution:

$$x_1, x_2, x_3, \ldots, x_n$$

which hopefully converges on x^*!

How do we do this?

The simplest approach is the method of bisection

Suppose $f(x)$ is continuous over the interval

$$x \in [a, b]$$

and that $f(a)f(b) < 0$

Then we know that there is at least one root in this interval!

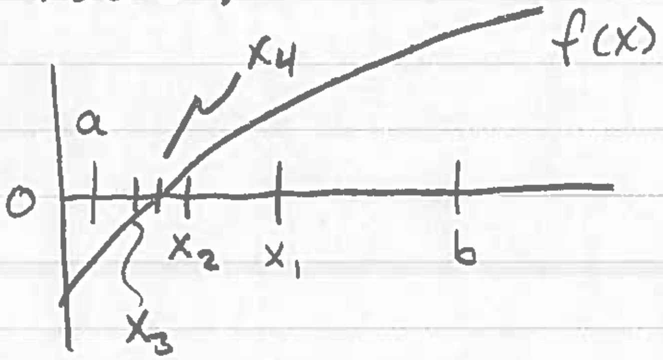
We can get within some interval of x^* in a finite number of steps!

We divide the interval in half and test to see which is satisfied:

$$f(a)f(m) < 0 \ \underline{\text{or}} \ f(m)f(b) < 0$$

where $m = \dfrac{a+b}{2}$

We just keep the half that has the root!



How fast does this converge?

Each iteration divides the interval in <u>half</u>, the midpoint is an estimate of x^*, thus:

$|m - x^*|$ is cut in half (approximately) at each iteration

The error is less than 10^{-7} of the original interval after about 24 iterations.

Let's take e_i as the error at the i^{th} iteration (e.g., $m - x^*$)

Then for bisection:

$$\frac{|e_{i+1}|}{|e_i|} \approx \frac{1}{2} \quad \text{on average}$$

In general, a method is said to converge at a rate <u>r</u> if:

$$\lim_{i \to \infty} \frac{|e_{i+1}|}{|e_i|^r} = C$$

In general, $|e_i| << 1$ so we want r to be <u>large</u> (>1) and C to be small.

If C is too large the method won't converge!

If $r=1$ the method is linear

If $r=2$ the method is quadratic

If $r>1$ the method is super linear

For bisection, $r=1$ (linear convergence)

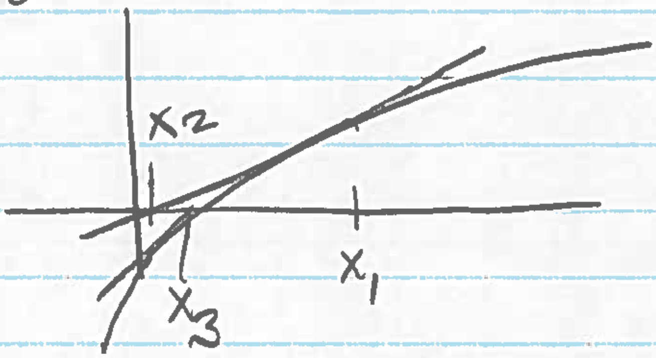
Let's look at a method with faster convergence: Newton's Method

In this method we compute both $f(x_i)$ and $f'(x_i)$

We fit the function with a tangent line (e.g., truncate the Taylor series after the linear term) and find its root.

This gives us our next guess!

So:



How do we get x_{i+1}?

$$f(x) = f(x_i) + f'(x_i)(x - x_i) + \dots$$

↑ truncate here

So set:

$$0 = f(x_i) + f'(x_i)(x_{i+1} - x_i)$$

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Let's use this method to calculate $\sqrt{2}$:

This is equivalent to the root of

$$f(x) = x^2 - 2 = 0$$

Thus $f'(x) = 2x$

and $\dfrac{f(x)}{f'(x)} = \dfrac{x^2-2}{2x}$

so $x_{i+1} = x_i - \dfrac{x_i^2 - 2}{2x_i}$

We start at $x_0 = 1$

Thus $x_1 = 1 - \dfrac{1-2}{2} = 1.5$

$$x_2 = 1.5 - \frac{(1.5)^2-2}{3} = 1\tfrac{5}{12} = 1.41666\ldots$$

$$x_3 = 1\tfrac{5}{12} - \frac{(1\tfrac{5}{12})^2-2}{2\,(1\tfrac{5}{12})} = \frac{577}{408}$$

$$= 1.4142157\ldots$$

vs. exact sol'n $x^* = 1.4142136\ldots$

How fast did this converge?

$e_0 = 0.4142136$
$e_1 = 0.085786$
$e_2 = 0.002453038$
$e_3 = 2.138 \times 10^{-6}$

much faster than linear convergence!

Actually, it's quadratic:

$$\frac{e_1}{e_0^2} = 0.5$$

$$\frac{e_2}{e_1^2} = 0.333$$

$$\frac{e_3}{e_2^2} = 0.355\ldots$$

Let's look at this in general

we have:

$$f(x) = f(x_i) + f'(x_i)(x - x_i) + \tfrac{1}{2} f''(\xi)(x - x_i)^2$$

Let $x = x^*$

Thus:

$$0 = f(x_i) + f'(x_i)(x^* - x_i) + \tfrac{1}{2} f''(\xi)(x^* - x_i)^2$$

Divide by $f'(x_i)$ and rearrange:

$$x^* - \underbrace{\left(x_i - \frac{f(x_i)}{f'(x_i)}\right)}_{x_{i+1}\,!} = \frac{f''(\xi)}{2f'(x_i)}(x^* - x_i)^2$$

$$\therefore \quad \frac{|e_{i+1}|}{|e_i|^2} = \left|\frac{f''(\xi)}{2f'(x_i)}\right| \cong C$$

So as $x_i \to x^*$,

$$C \sim \left|\frac{f''(x^*)}{2f'(x^*)}\right|$$

For the example problem,

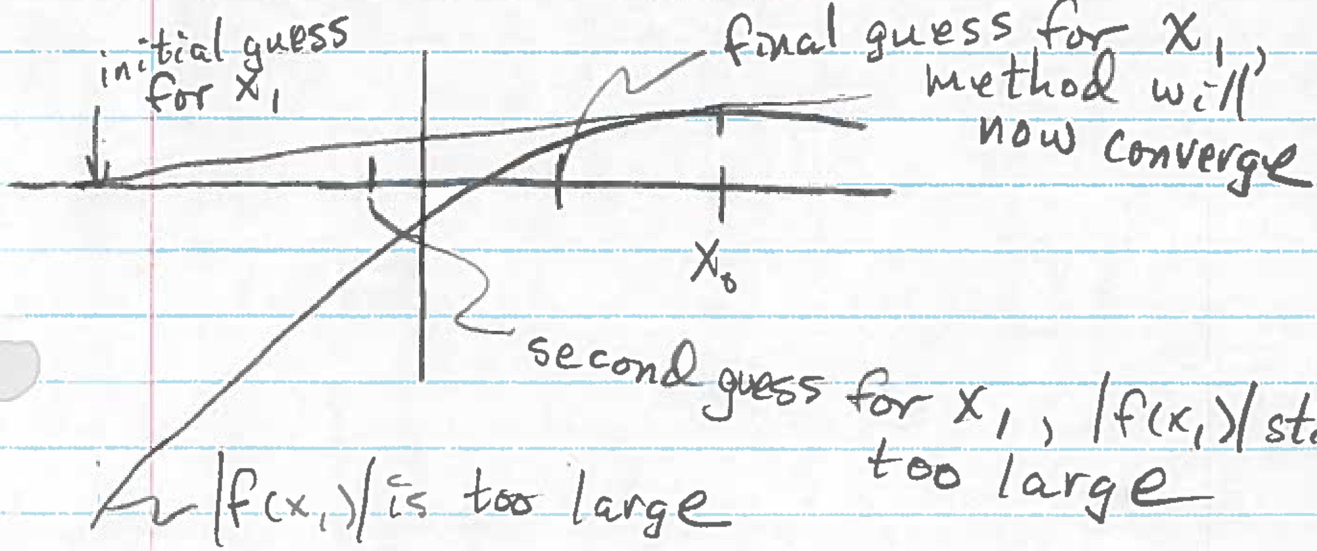
$f''(x^*) = 2$, $f'(x^*) = 2\sqrt{2}$ $\therefore$ $C = 0.354$

Newton's method does not always converge!

If $f'(x_i) \approx 0$ then $\frac{f(x_i)}{f'(x_i)}$ may be very large. If so, we will get a way out prediction for x_{i+1}. This is equivalent to

$C >> 1$.

We can fix this by requiring that $|f(x_{i+1})| < |f(x_i)|$ and cutting down on the correction by halves until this is satisfied.


initial guess for x_1
final guess for x_1, method will now converge
x_0
second guess for x_1, $|f(x_1)|$ still too large
$|f(x_1)|$ is too large

In general, you get into trouble if the first guess is too far off.

Newton's method has the drawback that it requires derivatives

we can avoid this using the secant method

Start with two points x_{i-1}, x_i

Compute line thru points

$$(x_{i-1}, f(x_{i-1})), \ (x_i, f(x_i))$$

find intercept to get x_{i+1} and then repeat

The line is given by:

$$y = f(x_i) + (x - x_i) \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}$$

Thus:

$$x_{i+1} = x_i - f(x_i)\,\frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})}$$

This is just Newton's method where we approximate the derivative by:

$$f'(x_i) \approx \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}$$

This converges a bit more slowly.

You can show that:

$$\frac{|e_{i+1}|}{|e_i|^{\frac{1}{2}(1+\sqrt{5})}} = \left|\frac{f''(x^*)}{2f'(x^*)}\right|^{\frac{1}{\frac{1}{2}(1+\sqrt{5})}}$$

where $\frac{1}{2}(1+\sqrt{5}) = 1.618\ldots > 1$

This method can fail to converge in the same ways as Newton's method.

Thus far we have looked at continuous methods for finding roots. These methods can fail for case of multiple roots and/or $f'(x) = 0$ in search domain.

Stadtherr is working on a different approach : Interval Methods

We look at Interval Newton Method: guaranteed to find all roots of a differentiable non-linear algebraic $f^{\underline{n}}$ over some domain.

Recall we had Newton's formula:

$$\underbrace{f'(x^k)}_{\substack{\text{deriv at} \\ k^{th} \text{ guess}}} \Big(\underbrace{x^{k+1}}_{k+1^{\underline{st}} \text{ guess}} - x^k \Big) = -f(x^k)$$

In the interval Newton method we do the same thing, but with intervals

Suppose we have a search domain

X^k at step k — the range in x we are interested in.

We evaluate the range of $f'(x)$ over this domain (this may be comp. expensive):

$$F'(X^k) \equiv \left[(f'(x))_{min} , (f'(x))_{max} \right]$$

where $x \in X^k$

Then we get Newton's equation for the new interval N^k:

$$F'(X^k)\left[N^k - \underset{\substack{\uparrow \\ \text{midpoint} \\ \text{of } X^k}}{x^k}\right] = -f(x^k)$$

We can learn stuff about the roots by comparing N^k and X^k

1) All roots in X^k are in $N^k \cap X^k$ (intersection), thus if N^k & X^k are disjoint, there are no roots in X^k!

2) If $N^k \subset X^k$ (subset) then there is one unique root in X^k!

How do we proceed? If we have the reduction:

$$(N^k \cap X^k) < X^k$$

then we take $X^{k+1} = N^k \cap X^k$ and do it again.

If $(N^k \cap X^k) \approx X^k$ then we bisect the interval and work on each half separately. Each half will either produce one or more roots, or will be shown to produce no roots via the disjoint test.

The interval Newton method is implemented in mathematica along with other interval techniques.

A reference for this work is:

Schnepper & Stadtherr, Computers in Chem. Engineering, 20 (2), 187-199, 1996,

Systems of Equations

So far we have just looked at a single equation. What if we had a <u>system</u> of equations?

suppose we have the set:

$$f_1(x_1, \ldots, x_n) = 0$$
$$\vdots$$
$$f_n(x_1, \ldots, x_n) = 0$$

or $\underset{\sim}{f}(\underset{\sim}{x}) = 0$

we wish to determine $\underset{\sim}{x} \equiv (x_1, \cdots, x_n)$

s.t. $\underset{\sim}{f}(\underset{\sim}{x}) \equiv (f_1, f_2, \ldots, f_n) = 0$

We shall use Newton's method:

expand $\underset{\sim}{f}(\underset{\sim}{x})$ in a Taylor series about $\underset{\sim}{x}^{(k)}$

Note: $\underset{\sim}{x}^{(k)}$ is the $k^{\underline{th}}$ guess for the root.

So:

$$\underset{\sim}{f}(\underset{\sim}{x}) = \underset{\sim}{f}(\underset{\sim}{x}^{(k)}) + \underset{\sim}{\nabla}\,\underset{\sim}{f}\Big|_{\underset{\sim}{x}^{(k)}} \cdot (\underset{\sim}{x} - \underset{\sim}{x}^{(k)}) + \ldots$$

The matrix ∇f is known as the <u>Jacobian</u> of $\underset{\sim}{f}$:

$$\underset{\approx}{J} \equiv \underset{\sim}{\nabla}\,\underset{\sim}{f}$$

or:

$$\underset{\approx}{J} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & & & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \cdots & \frac{\partial f_n}{\partial x_n} \end{pmatrix}$$

We truncate the Taylor series after the linear term & set it equal to zero to get the next guess!

$$0 = \underset{\sim}{f}^{(k)} + \underset{\approx}{J}^{k}(\underset{\sim}{x}^{k+1} - \underset{\sim}{x}^{k})$$

Thus:

$$\underset{\sim}{x}^{k+1} = \underset{\sim}{x}^{k} - \left[\underset{\approx}{J}^{k}\right]^{-1} \underset{\sim}{f}^{k}$$

Actually, this would be solved using Gaussian elimination.

If the Jacobian is <u>singular</u>, Newton's method will fail to converge!

This is equivalent to the first deriv. $f'(x_k)=0$ for a one-dim Newton's method problem!

Let's do a simple example:

$$f_1(x_1, x_2) = x_1 x_2 - x_2^3 - 1 = 0$$

$$f_2(x_1, x_2) = x_1^2 x_2 + x_2 - 5 = 0$$

What is the Jacobian?

$$J_{ij} = \frac{\partial f_i}{\partial x_j}$$

Thus:

$$\underset{\approx}{J} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{pmatrix} = \begin{pmatrix} x_2 & x_1 - 3x_2^2 \\ 2x_1x_2 & x_1^2+1 \end{pmatrix}$$

So:

$$\underset{\sim}{x}^{(k+1)} = \underset{\sim}{x}^{(k)} - \left.\begin{pmatrix} x_2 & x_1 - 3x_2^2 \\ 2x_1\,x_2 & x_1^2+1 \end{pmatrix}^{-1}\right|_{\underset{\sim}{x}^{(k)}} \begin{pmatrix} f_1(\underset{\sim}{x}^{(k)}) \\ f_2(\underset{\sim}{x}^{(k)}) \end{pmatrix}$$

Let's take an initial guess of:

$$\underset{\sim}{x}^{(0)} = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$$

$$\therefore \underset{\approx}{J}^{(0)} = \begin{pmatrix} 3 & -25 \\ 12 & 5 \end{pmatrix} \quad \& \quad \underset{\sim}{f}^{(0)} = \begin{pmatrix} -22 \\ 10 \end{pmatrix}$$

Thus:

$$\underset{\sim}{x}^{(1)} = \begin{pmatrix} 2 \\ 3 \end{pmatrix} - \begin{pmatrix} 3 & -25 \\ 12 & 5 \end{pmatrix}^{-1} \begin{pmatrix} -22 \\ 10 \end{pmatrix}$$

$$= \begin{pmatrix}2\\3\end{pmatrix} - \begin{pmatrix}4/9\\42/45\end{pmatrix} = \begin{pmatrix}14/9\\93/45\end{pmatrix} = \begin{pmatrix}1{,}55\ldots\\2{,}066\ldots\end{pmatrix}$$

The solution will converge to (2, 1) after about 8 iterations. Note that it doesn't go straight to the sol'n!

We started at $x_1 = 2$, which was the solution, but we moved away from it first!

Why? ⇒ because locally the tangent may not point in the correct direction!

Run matlab example!

OK, how do we improve on Newton's method?

We wish to guard against failure to converge we shall require that:

1) the algorithm makes progress to the sol'n at every step

2) The steps are never too large.

Thus we require:

$$\|\underset{\sim}{f}(\underset{\sim}{x}^{(k+1)})\|_2 < \|\underset{\sim}{f}(\underset{\sim}{x}^{(k)})\|_2$$

and that

$$\|\underset{\sim}{x}^{(k+1)} - \underset{\sim}{x}^{(k)}\|_2 < \delta$$

where δ is picked by the algorithm

if the truncated Taylor series (including the linear term) is a good approx to the function, we make δ large, if not we reduce δ

Suppose we have :

$$\underset{\sim}{x}^{(K+1)} = \underset{\sim}{x}^{(K)} + \underset{\underset{\text{correction at step } K}{\uparrow}}{\underset{\sim}{p}}$$

From Newton's method we expect :

$$\underset{\sim}{p} = -\left(\underset{\approx}{J}^{(K)}\right)^{-1} \underset{\sim}{f}^{(K)}$$

but this may not satisfy $\|\underset{\sim}{p}\|_2 < \delta$

In addition, if $\underset{\approx}{J}^{(K)}$ is singular, $\underset{\sim}{p}$ may not exist!

Instead, let's solve the constrained optimization problem :

$$\min_{\underset{\sim}{p}} \|\underset{\approx}{J}^{(K)} \underset{\sim}{p} + \underset{\sim}{f}^{(K)}\|_2$$

subject to $\|\underset{\sim}{p}\|_2 < \delta$

This works even for singular $\underset{\approx}{J}^{(K)}$!

This is a bit different from the linear regression problem. Linear regression is an unconstrained optimization problem, what we have is a constrained problem.

We'll look into solving this in a bit. If we have such a solver, we would

1) if $\underset{\sim}{f}(\underset{\sim}{x}^{k}) \approx 0$ then stop

2) calc. $\underset{\sim}{p}$ via constrained optimization solution

3) If $\|\underset{\sim}{f}(x^{(k)}+\underset{\sim}{p})\|_2 < \|f(\underset{\sim}{x}^{(k)})\|_2$ then accept step & go to 1. You may also want to increase δ.

4) If not, then reduce δ & go back to step 2.

This sort of algorithm will reliably find roots, and will be trapped by local minima.

Key: start close to the answer!

Optimization

In optimization we are trying to minimize or maximize an objective function, usually subject to a set of constraints

This is known as a constrained optimization problem

If there are no constraints, then we have an unconstrained optimization problem.

Example : suppose we are designing soup cans. We want to minimize the metal usage for volume enclosed, e.g., minimize surface/volume ratio

Obviously, a sphere would do this, but it's difficult to make & open.

Instead, we choose a cylindrical shape:

$$\text{Metal Area} = \underbrace{2\pi r^2}_{\text{2 ends}} + \underbrace{2\pi r h}_{\text{sides}} = A$$

$$\text{Volume} = \pi r^2 h = V$$

We wish to minimize A subject to the constraint that $V = 16\text{ oz}$ (say)

This is a 2-D non-linear constrained optimization problem

We can convert this to a 1-D unconstrained problem by recognizing:

$$h = \frac{V}{\pi r^2}$$

$$\therefore \; A = 2\pi r^2 + \frac{2V}{r}$$

Which has the sol'n $r = 1.08$, $h = 2.17$

$$A = 22.14$$

Actually, cans don't fit this ideal, as there are additional constraints such as labelling and packing considerations.

We have the general problem:

minimize $F(\underset{\sim}{x}) \equiv F(x_1, x_2 \ldots x_n)$

over some domain S in n-dim. space

For the can problem,

$$F(x_1, x_2) = 2\pi x_1^2 + 2\pi x_1 x_2$$

subject to $\underset{\sim}{x} \varepsilon S = \{(x_1, x_2) \mid 2\pi x_1^2 x_2 = V\}$

If $S \equiv \mathbb{R}_N$ (all space) then problem is unconstrained

$\underset{\sim}{x} \varepsilon S$ are the feasible solutions

Least squares data fitting is an unconstrained optimization problem.

If we have some point $\underset{\sim}{x}$ s.t. $\underset{\sim}{\nabla} F = 0$

then $\underset{\sim}{x}$ is a critical pt

It may be a minimum, a maximum, or a saddle point!

We define a point $\underset{\sim}{x}^*$ to be a <u>local minimum</u> if:

$$\underset{\sim}{x}^* \in S \quad \text{s.t.} \quad F(\underset{\sim}{x}^*) < F(\underset{\sim}{x}^* + \underset{\sim}{\delta})$$

$\underset{\sim}{x}^*$ is a <u>global minimum</u> if

$$F(\underset{\sim}{x}^*) < F(\underset{\sim}{x}) \; ; \; \underset{\sim}{x} \in S \quad (\text{all } \underset{\sim}{x})$$

You can't guarantee to find the global minimum! (or - interval techiques <u>do</u> allow this over some domain)

One-Dim. problems

There are three basic approaches analogous to the three root finding techniques. They are:

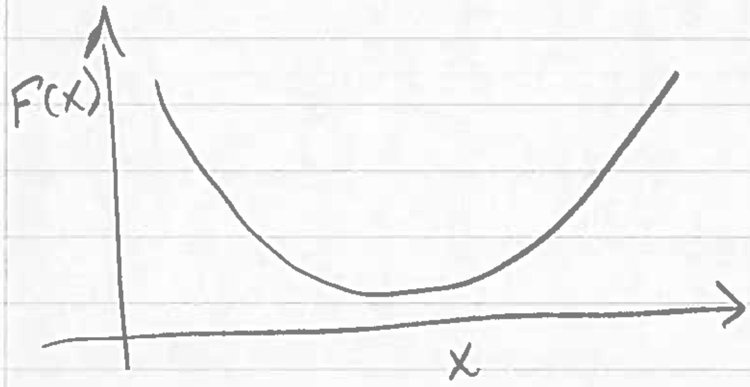
1) Newton's Method

2) Successive Parabolic Interpolation

3) Golden Search (Fibonacci Search)

We start with Newton's method

We have the one-D function F(x):



It is natural to approximate this with a parabola. We thus truncate its Taylor series after the quadratic term, and use the approx. to get the next estimate of the critical point:

$$F(x) = F(x_k) + (x-x_k)F'(x_k) + \tfrac{1}{2}(x-x_k)^2 F''(x_k) + \ldots$$

We seek x^* where $\underline{\underline{F'(x^*) = 0}}$

Thus:

$$x_{k+1} = x_k - \frac{F'(x_k)}{F''(x_k)}$$

This is virtually identical to root solving via Newton's method ⇒ in fact it is root solving, just the root of $F'(x)$

Successive Parabolic Interpolation is very similar to the secant method for root solving.

Instead of using two points to get a line, we use three points to get a parabola!

Thus if we have x_k, x_{k-1}, x_{k-2}

we fit $y = ax^2 + bx + c$ to the points:

$$(x_k, F(x_k)), (x_{k-1}, F(x_{k-1})), (x_{k-2}, F(x_{k-2}))$$

Once we have a, b & c we seek the critical pt.

$$\frac{dy}{dx} = 0 = 2ax + b$$

Thus:

$$x_{k+1} = \frac{-b}{2a}$$

and we iterate forward keeping the last three points

The rate of convergence is 1.324

This is super linear, but not as fast as Newton's method (r = 2)

Both techniques go to critical pts - either maxima or minima - and have convergence problems if you don't start close enough to the correct answer!

The Golden Search (modified Fibonacci search) is analogous to root finding using bisection, In this case rather than knowing some interval [a, b] where $f(a)f(b)<0$ as in bisection, here we require that the function $F(x)$ be unimodal over $[a,b]$

This means that (if we are looking for a minimum):

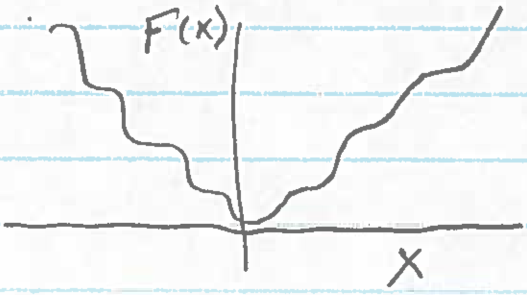
$$F'(x) \begin{cases} < 0 & x < x^* \\ > 0 & x > x^* \end{cases}$$

Note that F'' may change sign in this interval! If $F''=0$ Newton's method (and successive parabolic interpolation) will fail, but the golden search is unaffected!

An example of such a function:

$$F(x) = x^2 + a\cos(\omega x)^2$$

provided $\omega^2 a < 1$ there is only one critical point at $x^*=0$

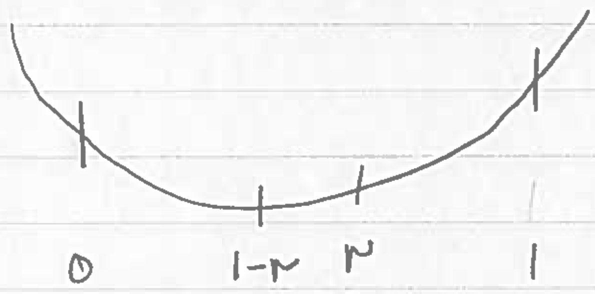


For large x, however, F'' is both positive and negative. Unless Newton's method is started close to the min. it will not converge!

The golden search relies on evaluating the function at points within the interval $[a, b]$ and determining in which part of the sub-interval the minimum lies.

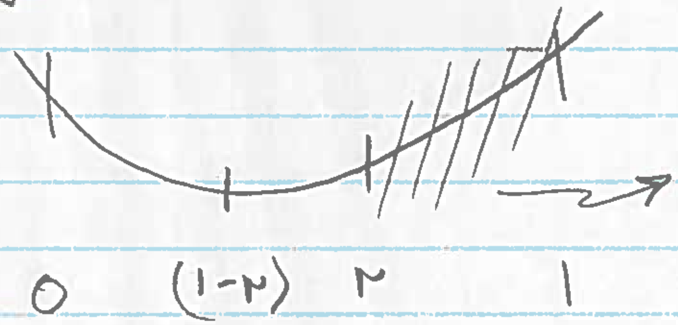
Suppose we have the interval $[0, 1]$ over which the function is unimodal, and which contains the minimum.

We evaluate the function at two additional points in the interior $(1-r)$, r s.t. $r > 0.5$. Thus:



Now if $F(r) < F(1-r)$ then the minimum will lie in the interval: $[(1-r), 1]$ and if $F(r) > F(1-r)$ then it will lie in the interval $[0, r]$.

We discard the balance of the original interval and do it again:



$F(r) > F(1-r)$ so right part is discarded!

The trick is to choose r so that in the next step we only need one new evaluation rather than two!

This means that we want the point $(1-r)$ on the left side of the original interval to map onto the point r on the right hand side of the new subinterval!

Thus we require:

$$\underset{\substack{\uparrow \\ \text{left hand} \\ \text{pt original} \\ \text{interval}}}{(1-r)} = \underset{\substack{\uparrow \\ \text{right hand} \\ \text{point of} \\ \text{sub interval}}}{(r)}\;\underset{\substack{\nwarrow \\ \text{width of sub-} \\ \text{interval}}}{(r)}$$

Thus $r^2 + r - 1 = 0$

$$r = \frac{\sqrt{5}-1}{2} = 0.6180 \qquad \swarrow \text{Golden Ratio} \quad \frac{1}{r} = r+1$$

You get the same value if you discard the left side of the interval and map the old rhs point onto the new left hand side point.

each iteration discards 38% of the current interval, thus:

$$\frac{|e_{i+1}|}{|e_i|} = 0.618$$

So we have <u>linear</u> rate of convergence, but C is higher than in bisection.

Run Matlab Example!

Multi-Dimensional Optimization

Life gets much more complex for higher $(N > 1)$ dimensional optimization.

In general, we start from a given point, pick a search direction and do a 1-D search in that direction.

Method of steepest descent:

If we want to get to a minimum, it makes sense to go down hill

Search direction $\equiv -\underset{\sim}{\nabla} F$

Thus we solve the one-D problem

$$\min_{\alpha} F\left(\underset{\sim}{x}^{(k)} - \alpha \underset{\sim}{\nabla} F(\underset{\sim}{x}^{(k)})\right)$$

to get

$$\underset{\sim}{x}^{(k+1)} = \underset{\sim}{x}^{(k)} - \alpha_{opt} \underset{\sim}{\nabla} F(\underset{\sim}{x}^{(k)})$$

where α_{opt} is the desired minimum

This method tends to be rather slow. The gradient often does not point towards the minimum! Let's work an example:

$$\text{Let } F(\underset{\sim}{x}) = x_1^2 + 10x_2^2 + 100x_3^2$$

This has the global minimum of $\underset{\sim}{x}^* = 0 \quad (x_1^* = x_2^* = x_3^* = 0)$

$$\text{Now} \quad \underset{\sim}{\nabla} F \equiv (2x_1, 20x_2, 200x_3)^T$$

Thus at each iteration we want to solve:

$$\min_{\alpha} F(\underset{\sim}{x} - \alpha \nabla F) = \min_{\alpha} \Big\{ (x_1 - 2\alpha x_1)^2$$

$$+ 10(x_2 - 20\alpha x_2)^2 + 100(x_3 - 200\alpha x_3)^2 \Big\}$$

We can actually get a linear equation for α for this particular problem.

$$\alpha = \frac{x_1^2 + 10^2 x_2^2 + 10^4 x_3^2}{2(x_1^2 + 10^3 x_2^2 + 10^6 x_3^2)}$$

After <u>180</u> iterations we get:

$x^{(0)} = (1, 1, 1)$ ← starting point

$x^{(180)} = (1.07 \times 10^{-4},\ 0\ ,\ 3.01 \times 10^{-6})$

Why? The problem has different curvature in different directions.

You can think of the algorithm as wandering back and forth across a narrow river valley and only slowly rolling down to the sea.

We can do better (sometimes) with a multi-dim. Newton's method.

Let's keep an extra term in the Taylor Series:

$$F(\underset{\sim}{x}) = F(\underset{\sim}{x}_0) + \underset{\sim}{\nabla} F(\underset{\sim}{x}_0) \cdot (\underset{\sim}{x} - x_0)$$

$$+ \frac{1}{2} \underset{\sim}{\nabla}\underset{\sim}{\nabla} F(\underset{\sim}{x}_0) \cdot (\underset{\sim}{x} - \underset{\sim}{x}_0)(\underset{\sim}{x} - \underset{\sim}{x}_0)$$

$$+ \ldots$$

The matrix $\underset{\sim}{\nabla}\underset{\sim}{\nabla}F$ is the <u>Hessian</u> <u>Matrix</u>

$$\underset{\sim}{\nabla}\underset{\sim}{\nabla}F \equiv \frac{\partial^2 F}{\partial x_i \partial x_j}$$

Remember, we seek the critical points of F. Thus:

$$\underset{\sim}{\nabla}F = 0 = \underset{\sim}{\nabla}F(x_0) + \underset{\sim}{\nabla}\underset{\sim}{\nabla}F(x_0)\,(\underset{\sim}{x} - \underset{\sim}{x}_0) + \ldots$$

If we truncate after this term, we get an equation for the next guess at the critical point:

$$\underset{\sim}{x}^{(k+1)} = \underset{\sim}{x}^{(k)} - \left(\underset{\sim}{\nabla}\underset{\sim}{\nabla}F(\underset{\sim}{x}^{k})\right)^{-1} \underset{\sim}{\nabla}F(\underset{\sim}{x}^{(k)})$$

These are Newton's equations for this problem!

The method will converge quadratically near the critical point, but it can fail to converge.

Suppose we want to find a minimum. We want an algorithm which does the following:

1) Computes $F(\underset{\sim}{x}^{(k)})$, $\underset{\sim}{\nabla} F(x^{(k)})$

2) Computes some descent direction $\underset{\sim}{p}$ such that:

$$F(\underset{\sim}{x}^{(k)} + \varepsilon \underset{\sim}{p}) < F(x^{(k)}),$$

for small ε

We can do this via steepest descent:

$$\underset{\sim}{p} = -\underset{\sim}{\nabla} F(x^{(k)})$$

or via Newton's method:

$$\underset{\sim}{p} = -\left[\underset{\sim}{\nabla}\underset{\sim}{\nabla} F\right]^{-1} \underset{\sim}{\nabla} F$$

Note that Newton's method does not necessarily point to a minimum!

3) Line search along vector $\underset{\sim}{p}$:

find some α s.t.

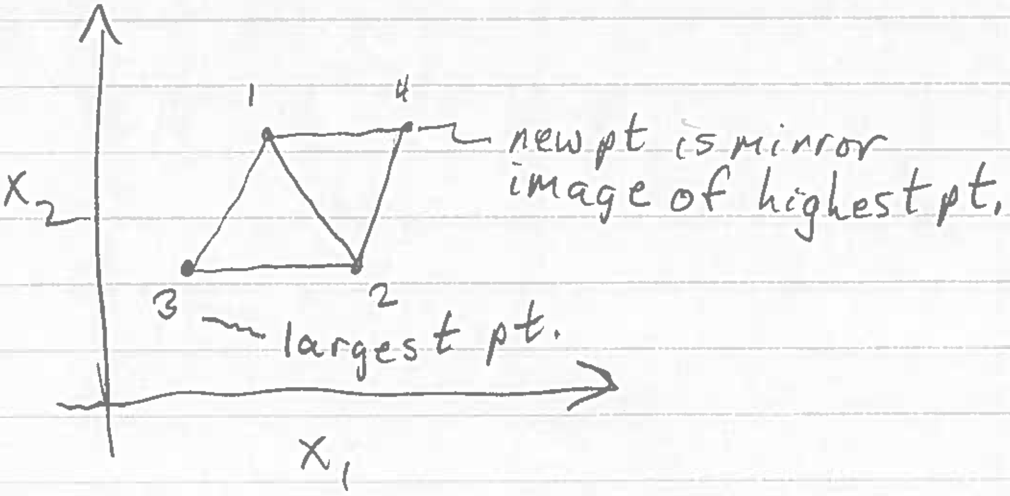
$$F(\underset{\sim}{x}^{(k)} + \alpha \underset{\sim}{p}) \text{ is minimized}$$

This is a one-D optimization problem!

Then return to step (1).

Simplex method:

A completely different algorithm is the simplex method. This algorithm uses a search based on triangles in 2-space and multi-dim. pyramids in n-space. We look at the 2-D problem:



We pick 3 pts in form of equilateral triangle. We discard the largest pt. (when looking for a minimum) and pick a new point which is its mirror image. We then wander through space until the min. is reached!

Technique works well if all elements of ∇F are comparable.

Tip: Rescale variables in an ill-cond. problem so that this is realized

Method is modified when minimum is approached (size of triangle is reduced.

Nelder-Mead implementation also changes triangle shape => makes triangle longer in the direction of the minimum.

Matlab function fminsearch uses this approach.

Constrained Optimization

So far, we have dealt w/ unconstrained optimization methods

Most of the time (except in regression problems) we have constraints on feasible solutions. How do we deal with this?

One-Dim.

In one D we may have inequality constraints (say $r > 0$). In this case we calculate the optimum in the interior and then compare to function on bdy of feasible sol'n region.

If F is lower on the boundary, then that point is the answer!

Multi-Dim.

- If N-dim., the problem is much more complex.

We can have either <u>equality constraints</u> or inequality constraints.

Look at equality constraints first

In general, we seek

$$\min_{\underset{\sim}{x}} F(\underset{\sim}{x}) \quad \text{subject to} \quad \underset{\sim}{g}(\underset{\sim}{x}) = 0$$

The <u>best</u> way of treating this is to use the m constraints $\underset{\sim}{g} = 0$ to <u>eliminate</u> m variables from $F(\underset{\sim}{x})$!

We did this with the soup can example:

$$F(x_1, x_2) = 2\pi x_1^2 + 2\pi x_1 x_2$$

$$\underset{\sim}{g}(\underset{\sim}{x}) = 2\pi x_1^2 x_2 - V = 0$$

we used this to eliminate x_2:

$$x_2 = \frac{V}{2\pi x_1^2}$$

$$F = 2\pi x_1^2 + \frac{V}{x_1}$$

So F is now an unconstrained 1-D problem!

Usually you can't get away with this. One approach is using Lagrange multipliers!

Let $F^* = F + \underset{\sim}{\lambda} \cdot \underset{\sim}{g}$

where $\underset{\sim}{\lambda}$ contains m multipliers for the m constraints $\underset{\sim}{g}(\underset{\sim}{x}) = 0$

The optimization problem was the solution to

$$\underset{\sim}{\nabla} F = 0$$

Here we have the augmented problem:

$$\underset{\sim}{\nabla}^* F^* = 0$$

$$\text{where} \quad \underset{\sim}{\nabla}^* \equiv \begin{pmatrix} \underset{\sim}{\nabla}_{\underset{\sim}{x}} \\ \underset{\sim}{\nabla}_{\underset{\sim}{\lambda}} \end{pmatrix}$$

This is because

$$\underset{\sim}{\nabla}_{\underset{\sim}{\lambda}} F^{*} = \underset{\sim}{g}(\underset{\sim}{x}) = 0$$

or just the equality constraints!

Now for inequality constraints

we have: $\underset{\sim}{g}(\underset{\sim}{x}) \leq 0$

We can convert inequality constraints to equality constraints by adding slack variables

Let $\underset{\sim}{g} + \underset{\sim}{s} = 0$

where $s_i = x_{n+i}^2 \geq 0$

↑
convenient choice insuring that $s_i \geq 0$

We then treat the problem using Lagrange multipliers again!

Finally, look at penalty functions

We wish to have an unconstrained optimization problem. We can do this even with constraints in an artificial manner. Suppose we have:

$$\min_{\underset{\sim}{x}} F(\underset{\sim}{x}) \quad , \quad \underset{\sim}{g}(\underset{\sim}{x}) = 0$$

We may define

$$F^{*}(\underset{\sim}{x}) = F(\underset{\sim}{x}) + P \, \| \underset{\sim}{g}(\underset{\sim}{x}) \|_2^2$$

where P is a positive number

We proceed by obtaining the solution for $\underset{\sim}{x}$ at moderate values of P, and then slowly increase it.

$P \to \infty$ corresponds to $\underset{\sim}{g}(\underset{\sim}{x}) \Rightarrow 0$!

This can also be used with inequality constraints through the use of slack variables.

Numerical Quadrature

We now look at two related topics: quadrature and integration

quadrature: evaluation of integral of a known function over a specified domain.

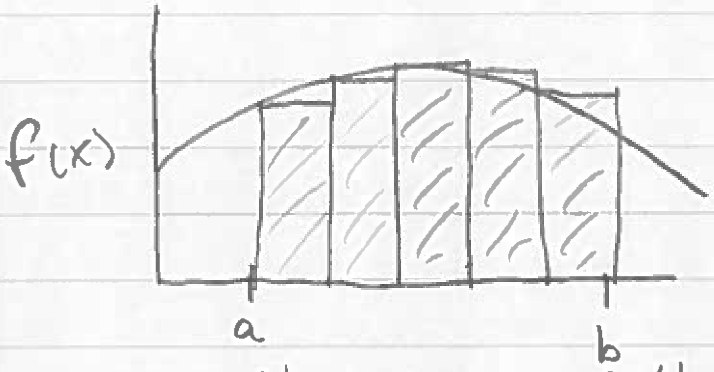
integration: integration of a set of differential equations

Let's look at quadrature first. We want to solve:

$$I = \int_a^b f(x)\,dx$$

How do we do this? We evaluate f(x) at several points in the domain [a, b] and combine them to estimate I

Thus:



we sum the area of the rectangles as an estimate of I

$$I = \sum_{i=1}^{n} w_i f(x_i) + R_n$$

$w_i \equiv$ weights

$x_i \equiv$ nodes

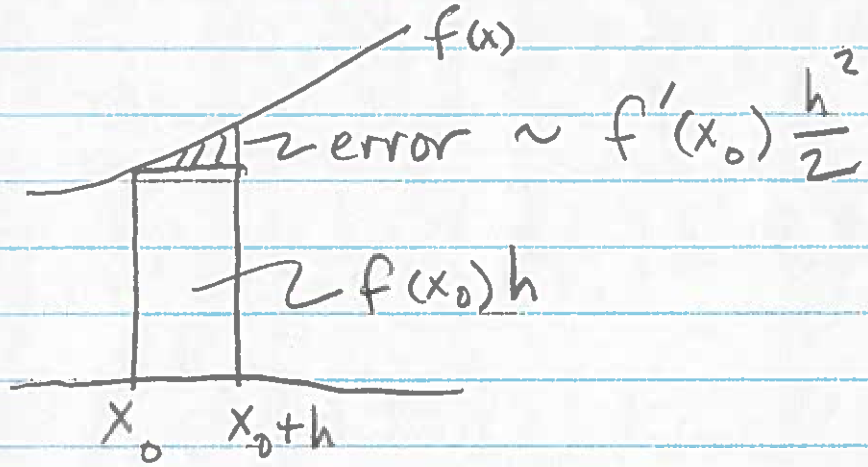
$R_n \equiv$ error in approximation

In this case, with n panels,

we have:

$$w_i = \frac{b-a}{n} \quad , \quad x_i = a + \frac{b-a}{n}(i-1)$$

What is R_n? We make some error in each interval:



Thus in each panel the error is

$$O\left(f'\frac{h^2}{2}\right)$$

The number of panels is $n = \frac{b-a}{h}$ ↳ panel width

Thus the total error is:

$$O\left(f' h^2 \frac{b-a}{h}\right) \sim O(f' h (b-a))$$

So this rule is of order $h^{(1)}$

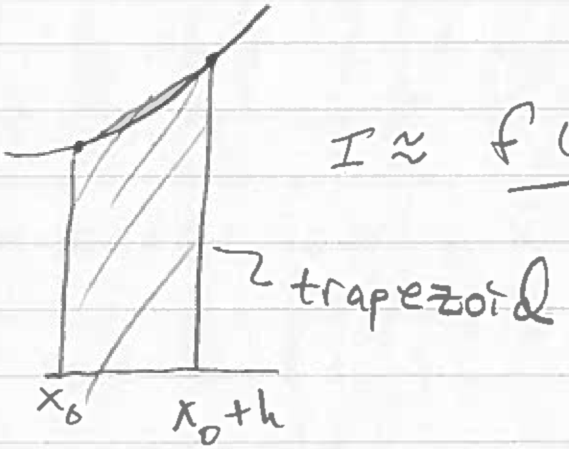
$$R_n = O\left(\frac{(b-a)^2}{n} f'\right)$$

If $f' \equiv 0$ ($f = cst$) then the rule is exact ($R_n = 0$)

We call a quadrature rule of polynomial degree d if it gets all polynomials of degree d exactly, but makes errors in polynomials of degree $(d+1)$

This rule was of degree zero

What other rules are there? How about the trapezoidal rule:



$$I \approx \frac{f(x_0) + f(x_0+h)}{2} h$$

what is the error?

We are approximating $f(x)$ with a line from $f(x_0)$ to $f(x_0+h)$. The error in this is proportional to the curvature, or f''!

$$\text{error} \sim O(h^3 f'')$$

↳ require h^3 for dimensions to work out.

Since the number of intervals is again:

$$n = \frac{b-a}{h}$$

we get for the total error:

$$R_n = O\left((b-a)h^2 f''\right) = O\left(\frac{(b-a)^3}{n^2} f''\right)$$

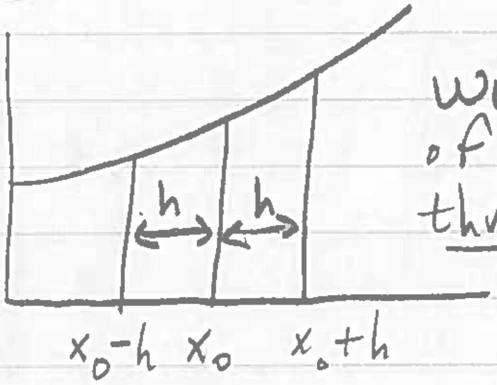
with:

$$w_i = \frac{b-a}{n}\begin{cases} \frac{1}{2} & i=0,n \\ 1 & 1 \le i \le n-1 \end{cases}$$

$$x_i = a + \frac{(b-a)}{n} i$$

The trapezoidal rule is of polynomial degree 1 because it gets lines correct but makes errors on quadratic functions!

We can do better using Simpson's Rule:



we take a pair of panels with three function evaluations

we fit a parabola through these three points:

$$Q(x) = f(x_0-h)\,\frac{(x-x_0)(x-x_0-h)}{2h^2}$$

$$+\ f(x_0)\,\frac{(x-x_0-h)(x-x_0+h)}{-h^2}$$

$$+\ f(x_0+h)\,\frac{(x-x_0+h)(x-x_0)}{2h^2}$$

So we integrate $Q(x)$ over x_0-h to x_0+h

to get the quadrature rule

$$I \approx f(x_0-h)\,w_1 + f(x_0)\,w_2 + f(x_0+h)\,w_3$$

where $w_1 = w_3 = \frac{h}{3}$ $\quad w_2 = \frac{4h}{3}$

Thus in general:

$$I = \sum_{i=0}^{2n} w_i \, f\left(a + \frac{b-a}{2n}\, i\right) + R_{2n}$$

$$w_i = \frac{h}{3}\begin{cases} 1 & i = 0,\, 2n \\ 4 & i = \text{odd} \\ 2 & i = \text{even (other than } i = 0,\, 2n\,) \end{cases}$$

The error is:

$$R_n = \mathcal{O}\left(f^{iv}\, h^4\, (b-a)\right)$$

So Simpson's rule is of polynomial degree $\underline{\underline{3}}$

It gets cubics right even though we fit it with a parabola.

This occurs because of the <u>symmetry</u> of the rule.

Ok, let's try this rule out:

$$\int_0^2 x^3 \, dx = \frac{1}{4}\left(2^4 - 0\right) = \frac{16}{4} = 4$$

We use a 2 <u>panel</u> approximation:

$$I \approx \frac{2-0}{6}\left(f(0) + 4f(1) + f(2)\right)$$

$$= \frac{24}{6} = 4$$ - no error

This is because Simpson's rule gets <u>all</u> cubics exactly! $f^{IV}(x) = 0$

Now let's try integrating x^4:

$$\int_0^2 x^4 \, dx = \frac{1}{5}\left(2^5 - 0\right) = \frac{32}{5} = 6.4$$

So:

$$I \cong \frac{2-0}{6}\left(f(0) + 4f(1) + f(2)\right) = \frac{40}{6} = 6.67$$

If we use the trapezoidal rule we put them at a & b. This is **not** the optimum choice!

Instead we let

$$I \approx w_1 f(x_1) + w_2 f(x_2)$$

and choose all four parameters such that we can integrate the highest degree polynomial possible!

We have 4 parameters, so we can integrate an arbitrary cubic polynomial with 4 constants!

Let $m = \dfrac{b+a}{2}$

we want to pick x_1, x_2, w_1, w_2 so that we integrate without error:

$$f(x) = 1 = (x-m)^0$$
$$f(x) = (x-m)^1$$
$$f(x) = (x-m)^2$$
$$f(x) = (x-m)^3$$

The error is thus $0.266\ldots = \frac{4}{15}$

Let's cut h in half:

$$\int_0^2 x^4 \, dx \approx \frac{1}{6}\left(0 + 4\left(\frac{1}{2}\right)^4 + 2(1)^4 + 4\left(\frac{3}{2}\right)^4 + (2)^4\right)$$

$$= \frac{77}{12} = 6.417$$

The error is now only $0.0166\ldots = \frac{1}{60}$ which is much smaller.

Note that when we halved the interval we decreased the error by a factor of 16. This was because the error was $O(h^4)$

Now for Gaussian Quadrature

Suppose we want to integrate

$I = \int_a^b f(x) \, dx$ using only 2 pts.

Where do we put them?

Any cubic is a linear combination of these four functions. If we get these right, we get them all right!

We get the four equations:

$$(1)\ \int_a^b (x-m)^0\,dx = (b-a) = w_1 + w_2$$

$$(2)\ \int_a^b (x-m)^1\,dx = 0 = w_1(x_1-m) + w_2(x_2-m)$$

$$(3)\ \int_a^b (x-m)^2\,dx = \frac{(b-a)^3}{12} = w_1(x_1-m)^2 + w_2(x_2-m)^2$$

and:

$$(4)\ \int_a^b (x-m)^3\,dx = 0 = w_1(x_1-m)^3 + w_2(x_2-m)^3$$

The second and fourth equations enforce symmetry:

$$w_1 = w_2, \quad (x_1-m) = -(x_2-m)$$

From the first equation $w_1 = w_2 = \dfrac{b-a}{2}$

and from the third we get:

$$x_1 = \frac{a+b}{2} - \frac{b-a}{2\sqrt{3}}, \quad x_2 = \frac{a+b}{2} + \frac{b-a}{2\sqrt{3}}$$

So we get the quadrature rule:

$$I \approx \frac{b-a}{2}\left[f\left(\frac{a+b}{2} - \frac{b-a}{2\sqrt{3}}\right) + f\left(\frac{a+b}{2} + \frac{b-a}{2\sqrt{3}}\right)\right]$$

Let's use this to integrate x^4:

$$I = \int_0^2 x^4 dx \approx \frac{2-0}{2}\left[\left(1 - \frac{1}{\sqrt{3}}\right)^4 + \left(1 + \frac{1}{\sqrt{3}}\right)^4\right]$$

$$= \frac{56}{9} = 6.222..$$

The error is just $\frac{8}{45}$ which is less than the 3pt. Simpson's rule!

In general:

$$\int_a^b f(x) dx = \sum_{i=1}^{n} w_i f(x_i) + R_n$$

where $R_n = \dfrac{(b-a)^{2n+1}(n!)^4}{(2n+1)[(2n)!]^3} f^{2n}(\xi)$

where $\xi \in [a,b]$

provided $f(x)$ has $2n$ continuous derivatives on $[a,b]$

If $f(x)$ has singularities in itself or any of its derivatives the error can be much larger.

The Gaussian quadrature rules are of polynomial degree $2n-1$. This is because it has $2n$ adjustable parameters.

What are the properties of Gaussian Quadrature?

1) The weights & nodes are, in general, irrational numbers. The exceptions are:

$n=1$: the midpoint rule

$$w_1 = (b-a) \quad , \quad x_1 = m = \frac{b+a}{2}$$

$$n=2 : \quad w_1 = w_2 = \frac{(b-a)}{2}$$

$n = $ odd : the midpoint m is a node
As a consequence, lists of weights and nodes are usually provided in subroutines that do the quadrature.

2) Gaussian rules are <u>open</u>: the function is never evaluated at the edges of the domain of integration. This is convenient for integrals like:

$$\int_0^1 \frac{\sin x}{x}\, dx$$

which is well behaved at $x=0$, but may lead to errors in closed rule quadrature.

3) Nodes for the n-point rule are <u>disjoint</u> from those of the m-point rule, with the exception that the midpoint is always a node for odd n, m.

Two sets are <u>disjoint</u> if they have no elements in common.

4) Among all rules using n-point evaluation the n-point Gaussian rule will produce the most accurate estimate for a smooth function.

5) The weights in gaussian quadrature are always positive. This would not be true if the nodes were evenly spaced and the weights determined optimally. Thus increasing the number of nodes always improves accuracy. For evenly spaced nodes at large n you can get large positive and negative weights leading to roundoff error.

6) The n Gauss nodes of the n-point rule are the roots of the $n^{\underline{th}}$ Legendre polynomial.

In Gaussian quadrature routines the weights and nodes are given based on the interval $[-1, 1]$. The nodes are thus <u>symmetric</u>:

Thus for the four point rule:

x_i^*	w_i^*
$\pm 0.8611363\ldots$	$0.3478548\ldots$
$\pm 0.3399810\ldots$	$0.6521452\ldots$

Suppose we have some general interval $[a,b]$. We can use the stored values of x_i^* and w_i^* to get the values of x_i and w_i on the general interval via a <u>mapping</u>:

$$x_i = \frac{a+b}{2} + \frac{b-a}{2}\, x_i^*$$

$$w_i = \frac{b-a}{2}\, w_i^*$$

and:

$$\int_a^b f(x)\,dx = \sum_{i=1}^{n} w_i f(x_i) + R_n$$

yields the desired result.

Note that even if the nodes aren't optimally spaced, we could still do better just by adjusting our weights!

Suppose all n nodes are evenly spaced! For $n=1$ we just have a node in the center — e.g., the midpoint rule:

$$I \approx w_1 f(x_1)\,(b-a) \qquad x_1 = \frac{b-a}{2}$$

By symmetry, (since - by chance - the node <u>was</u> optimal) it is of degree 1

Ok, for $n=2$

$$x_1 = a, \quad x_2 = b$$

Thus, we choose weights by fitting a line to the two points:

$$\underline{I} \approx (b-a)\left[w_1 f(a) + w_2 f(b)\right]$$

180 B

This just yields TR!

$$w_1 = w_2 = \frac{1}{2}$$

For $n=3$ we have

$$x_1 = a, \quad x_2 = \frac{b-a}{2}, \quad x_3 = b$$

We get a rule of degree $\underline{\underline{3}}$ because the midpoint is optimal!

$$\therefore \quad \underline{I} \approx (b-a)\left[\frac{1}{6}f(x_1) + \frac{4}{6}f(x_2) + \frac{1}{6}f(x_3)\right]$$

This is Simpson's Rule!

For the $n=4$ rule we have:

$$x_1 = a, \quad x_2 = a + \frac{b-a}{3}, \quad x_3 = a + \frac{2}{3}(b-a), \; x_4 = b$$

None of the nodes are optimal, so again we are of degree 3!

$$\underset{\sim}{w} = (b-a)\,\frac{1}{8}(1, 3, 3, 1)$$

180C

For $n=5$ we can fit a 5^{th} degree polynomial, as once again the mid-point is optimal

For this case: $\underset{\sim}{x} = a + \frac{b-a}{4}(0,1,2,3,4)^T$

and we get:

$$\underset{\sim}{w} = \frac{b-a}{90}\left[7, 32, 12, 32, 7\right]$$

Note that while this rule is of degree 5, the 5 point GQ rule is of degree $5 \times 2 - 1 = \underline{\underline{9}}$

Note on integrating data: Usually nodes are evenly spaced! (They're also fixed)

Thus: if even # panels, can use SR

If odd then use TR (also if unevenly spaced). Note that error in integral may be dominated by data error, not h! (thus TR may be better!)

Error Estimation

It's not enough just to compute some estimate for an integral. We must also estimate how accurate the estimate is!

One approach: for the trapezoidal rule, just cut the interval in half!

Let's look at the example

$$\int_0^2 x^3\,dx = \frac{1}{4}(2)^4 = 4$$

Suppose we use n panels:

$$n=2: \quad I \approx \frac{1}{2}(0)^3 + 1(1)^3 + \frac{1}{2}(2)^3 = 5$$

$$n=4: \quad I \approx \frac{1}{2}\left[\frac{1}{2}(0)^3 + \left(\frac{1}{2}\right)^3 + (1)^3 + \left(\frac{3}{2}\right)^3 + \frac{1}{2}(2)^3\right] = 4\frac{1}{4}$$

$$n=8: \quad I \approx \frac{1}{4}\left[\qquad\qquad\right] = 4\frac{1}{16}$$

Each time the error was decreased by a factor of 4! This is because the error was $O(\frac{1}{n^2})$.

We can actually use this to improve our result via repeated Richardson Extrapolation:

$$I = I_n + \frac{\lambda}{n^2} + \underset{\substack{\uparrow \\ \text{next order in} \\ \text{error term.}}}{O\left(\frac{1}{n^4}\right)}$$

Note that odd powers in n are killed off due to symmetry

Suppose we took the n=2 and n=4 results:

$$I = I_2 + \frac{\lambda}{4} + O\left(\frac{1}{2^4}\right)$$

$$I = I_4 + \frac{\lambda}{16} + O\left(\frac{1}{4^4}\right)$$

If we multipy the second equation by 4 and subtract from the first we can eliminate λ:

$$3I = 4I_4 - I_2 + O\left(\frac{1}{2^4}\right)$$

$$\therefore I = \frac{1}{3}\left(4I_4 - I_2\right) + O\left(h^4\right)$$

This actually just gives us Simpson's rule back again!

We have:

$$I_2 = \frac{b-a}{2}\left(\frac{1}{2}f(a) + f\left(a + \frac{b-a}{2}\right) + \frac{1}{2}f(b)\right)$$

$$I_4 = \frac{b-a}{4}\left(\frac{1}{2}f(a) + f\left(a + \frac{b-a}{4}\right) + f\left(a + \frac{b-a}{2}\right) + f\left(a + \frac{3}{4}(b-a)\right) + \frac{1}{2}f(b)\right)$$

Let $h = \frac{b-a}{4}$

Thus:

$$\frac{1}{3}\left(4I_4 - I_2\right) = \frac{1}{3}h\left[f(a) + 4f(a+h) + 2f(a+2h) + 4f(a+3h) + f(b)\right]$$

Note that by doubling the number

of panels we only have to evaluate the function at n new points: we can use all of the points from the first evaluation over again! This can make a big difference if it takes alot of time to make a function evaluation.

If we had just increased the number of panels by one, we wouldn't have gotten a significant increase in accuracy and all of the interior nodes would change!

We can take this combination process to a higher level. Suppose we have an n panel and $2n$ panel pair of Simpson's rule estimates. The error in each is $1/n^4$. Thus:

$$I = S_n + \frac{\lambda}{n^4} + O\left(\frac{1}{n^6}\right)$$

$$I = S_{2n} + \frac{\lambda}{(2n)^4} + O\left(\frac{1}{n^6}\right)$$

We can thus get an $O(\frac{1}{n^6})$ rule from the combination:

$$I = \frac{1}{2^4-1}\left(2^4 S_{2n} - S_n\right) + O\left(\frac{1}{n^6}\right)$$

This process can be repeated again and again in what is known as repeated Richardson extrapolation, or Romberg Integration, and the ultimate result is known as the Newton-Cotes formula.

Let's see how this works. Suppose we calculate an integral of $f(x)$ over $[a, b]$ using $n = 1, 2, 4, 8, 16, \ldots$ panel Trapezoidal rules. This gives us the series of estimates:

$$T_1, T_2, T_4, T_8, T_{16}, \ldots$$

Note that if we quit at $n = 16$ panels, we would only need 17 function evaluations — the nodes for the $n/2$ panel rule are included in those for the n-panel rule.

We may thus combine T_1 & T_2 to get S_2; T_2 & T_4 to get S_4, and so on.

$$S_2 = \frac{1}{2^2-1}\left(2^2 T_2 - T_1\right)$$

$$S_4 = \frac{1}{2^2-1}\left(2^2 T_2 - T_1\right)$$

The simpson's rule estimates can be combined as well. If we look at this in matrix form we get:

$$\begin{array}{llllll} T_1 \\ T_2 & S_2 \\ T_4 & S_4 & P_4 \\ T_8 & S_8 & P_8 & Q_8 \\ T_{16} & S_{16} & P_{16} & Q_{16} & R_{16} & \ddots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{array}$$

Thus, the iteration process produces a lower triangular matrix. The most accurate integral estimate is given by the bottom-right element, and the estimate of the error is given by the difference between the last two elements of the main diagonal!

If we start with 2^k panels (a total of 2^k+1 evaluations) then our extrapolation matrix is of size $(k+1)\times(k+1)$, and the error in the last estimate is:

$$\text{error} \sim O\left(h^{2(k+1)} f^{2(k+1)}(\xi)\,(b-a)\right)$$

where $h = \frac{b-a}{2^k}$

If $\underset{\approx}{M}$ is our matrix, then an estimate for this error is:

$$\text{error} < \left| M_{(k+1),(k+1)} - M_{k,k} \right|$$

How about error estimation in Gaussian quadrature? Because the nodes are disjoint, doubling the number of points does not allow you to reuse any! A way around this was suggested by Kronod:

Use the same n points over again, but add $n+1$ new ones and $2n+1$ new weights, chosen optimally!

Thus if we have:

$$G_n = \sum_{i=1}^{n} w_i f(x_i)$$

Then:

new weights

$$K_{2n+1} = \sum_{i=1}^{n} a_i f(\underset{\substack{\uparrow \\ \text{old nodes}}}{x_i}) + \sum_{j=1}^{n+1} b_j f(\underset{\substack{\uparrow \\ \text{new nodes}}}{y_j})$$

G_n is of degree $2n-1$

If a_i, b_j, y_j are determined properly K_{2n+1} is of degree $3n+2$!

The cost to compute the Gauss-Kronrod pair (G_n, K_{2n+1}) is $2n+1$ function evaluations. Computing G_{2n+1} requires the same number, is of degree $4n+3$ (much better than K_{2n+1}) but doesn't give an error estimate

What is the error in K_{2n+1}?

we expect the error to be less than the difference:

$$\text{error} < |K_{2n+1} - G_n|$$

but this way overestimates the error!

A better error estimate may be found as:

$$\text{error} \approx (b-a)\,\|f\|\left(200\,\frac{|G_n - K_{2n+1}|}{\|f\|(b-a)}\right)^{1.5}$$

where $\|f\|$ is some measure of the magnitude of $f(x)$ over the interval. This can be calculated from the function evaluations at the quadrature nodes.

Usually codes use the $G_7\,K_{15}$ pair

For smooth functions, the error in K_{15} is $O(10^{-17}) \Rightarrow$ less than roundoff error in double precision.

If the function has discontinuities or singularities, the error may be much greater!

If you use $G_7 K_{15}$ rule, or a 32 panel repeated Richardson extrapolation (see example) the error for a smooth function is $O(10^{-15} - 10^{-17})$ or so – on the order of double precision!

If the function is not smooth, the error will be much greater!

Consider Simpson's rule: The error is $O((b-a) f^{IV} h^4)$

Even though h^4 may be small, f^{IV} may blow up!

This is particularly true for functions with integrable singularities

Suppose we wish to evaluate:

$$I = \int_0^1 x^{-p} \, dx$$

For $x = 0$, $p > 0$ the integrand is singular

However, if $p < 1$ we can still integrate it: the integral has a finite value

$$I = \int_0^1 x^{-p} dx = \frac{1}{1-p}\left(1 - (0)^{1-p}\right)$$

if $p = 1$ this blows up!

other example: $f(x) = \ln x$ is an integrable singularity

How do we integrate such a function?

First, we can't use a closed rule as that requires function evaluation at the singularity!

Second, even an open rule would have problems as a polynomial approximation is poor.

All quadrature rules are based on approximating a function with a polynomial of high degree. If this approximation is poor, the result will be inaccurate.

The best solution is to remove the singularity analytically

Example:

$$I = \int_0^1 \frac{\ln x}{1+x^2}\, dx \qquad \text{as } x \to 0 \quad f(x) \to \ln x$$

Thus we subtract off the singularity!

$$I = \int_0^1 \frac{\ln x}{1+x^2} + \ln x - \ln x \left(\frac{1+x^2}{1+x^2}\right) dx$$

$$= \int_0^1 \ln x \, dx + \int_0^1 \frac{-x^2 \ln x}{1+x^2}\, dx$$

We evaluate the first part analytically and the non-singular part numerically!

Even if $f(x)$ is non-singular, it's derivatives may be! To integrate these functions we require adaptive quadrature.

Basically, adaptive quadrature is just a way of making the discretization finer where a polynomial approximation is poor.

An adaptive quadrature algorithm has two parts

1) A local quadrature module which computes the integral over the domain $[\alpha, \beta]$ and also provides an error estimate.

2) A control strategy which decides what portion of the interval to subdivide.

This algorithm works as follows:

1) start by feeding $[a,b]$ to the LQM. If the error $< \varepsilon$ (desired tolerance) then quit. This will probably happen for smooth functions!

2) If the total error $> \varepsilon$, divide the interval with the largest contribution to the error in 2 pieces, and send both to the LQM.

3) Add up the error. If it's less than ε then quit, if it's greater go to (2).

Usually such an algorithm will have a stop built in to avoid too fine a level of discretization.

Matlab offers two adaptive quadrature functions, 'quad' and 'quadgK'. The first uses Simpson's rule for the LQM, the second uses the G_7K15 Gauss-Kronod rule. Simpson's rule is closed, so singularities must be removed analytically for that one!

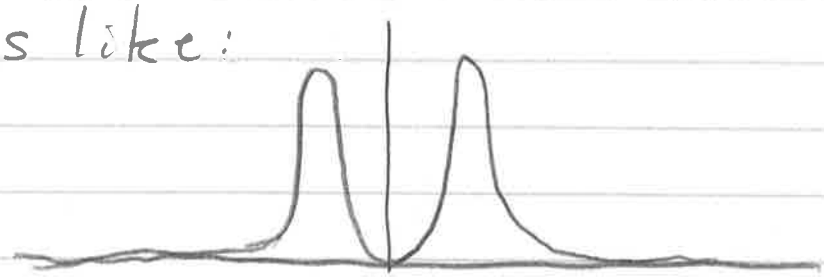
but Matlab has now "fixed" this for "quad"

In general, quadrature algorithms have trouble with steeply varying functions. For example, suppose we want to calculate the integral:

$$I = \frac{1}{\sqrt{2\pi}} \int_{-N}^{N} t^2 e^{-t^2/2} dt$$

which is (as $N \to \infty$) the variance of the normal (Gaussian) distribution.

If we plug in $N = 26$ we can run into trouble! The function looks like:



Using 'quad' on this function yields 5.2×10^{-6} - but if you used $N = 25$ you would get the correct answer $I = 1$!
(Note - Quadgk works even for large N - it chokes for $N > 6035$)
There is no substitute for knowing the answer before you start!

Often we want to integrate over infinite & semi-infinite domains. How do we deal with this?

Example:

$$I = \int_0^\infty e^{-x} \cos^2 x^2 \, dx$$

One way is to simply truncate at some large number A:

Say $I \approx Q = \int_0^A e^{-x} \cos^2 x^2 \, dx$

This produces an error:

$$I - Q = \int_A^\infty e^{-x} \cos^2 x^2 \, dx < \int_A^\infty e^{-x} \, dx = e^{-A}$$

which is exponentially small!

Unfortunately, this is not usually true.

In general $I - Q = \int_A^\infty f(x) \, dx$

if $f(x) \cong x^{-3/2}$ at large x then

$$\int_A^\infty x^{-3/2}\,dx = -\frac{1}{2}A^{-1/2}$$

Thus for $A = 10^4$, error is still $\mathcal{O}(10^{-2})$!

A better way is to transform the integral by mapping the domain.

Let $x = p(t)$

If we pick $p(t) = -\ln t$ or $\frac{t}{1-t}$

we map the domain $[0, \infty]$ onto $[0, 1]$

e.g. if $x = -\ln t$

then: $\displaystyle\int_0^\infty f(x)\,dx = -\int_1^0 f(-\ln t)\,\frac{dt}{t}$

$$= \int_0^1 f(-\ln t)\,\frac{dt}{t}$$

This can work <u>very</u> well:

if $f(x) = e^{-x}$ then:

$$\int_0^\infty e^{-x}\, dx = \int_0^1 e^{\ln t} \frac{dt}{t} = \int_0^1 dt = 1 !$$

or rather poorly:

if $f(x) = \frac{1}{1+x^2}$

$$\int_0^\infty \frac{1}{1+x^2}\, dx = \int_0^1 \frac{1}{1+(\ln t)^2} \frac{1}{t}\, dt$$

which has an integrable singularity at the origin! In general, the choice of mapping depends on the function - you want to choose a mapping that captures the asymptotic behavior of the function!

You can also combine mapping with truncation: map the integrand so that it is exponentially small in t, and then truncate it.

So far we have just looked at one-dimensional quadrature. Often we need to integrate over 2 or more dimensions, e.g.

$$I = \iint_D f(x,y)\,dA$$

Say, calculate the lift on a wing by integrating the pressure over the wing surface!

We can write the n-point 2-D quadrature rule as:

$$I = \sum_{i=1}^{n} w_i f(x_i, y_i) + R_n$$

This formula is of polynomial degree d if $R_n = 0$ for any bivariate polynomial of degree d but non-zero for some polynomial of degree $d+1$

What is a bivariate polynomial?

It is a linear combination of terms

$x^p y^q$ where $p+q \leq d$

For example, the most general polynomial of bivariate degree 2 is:

$$ax^2 + by^2 + cxy + dx + ey + f = 0$$

There are $\underline{6}$ parameters, thus to integrate this exactly we require at least $\underline{2}$ nodes:

$$I = w_1 f(x_1, y_1) + w_2 f(x_2, y_2)$$

For each node we have $\underline{3}$ adjustable parameters: x_i, y_i, and w_i

The problem is that the location and weights are dependent on the domain shape!

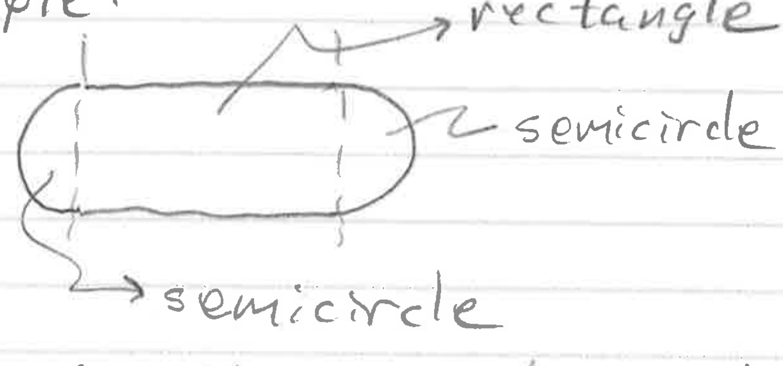
Optimum node locations and weights have been derived for rectangles, circles, and triangles.

What happens for more complex shapes?

There are a number of choices:

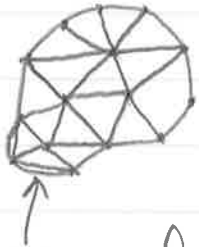
1) Map onto a simple domain via a change of variables. For example, a simple stretching maps an ellipse onto a circle, and polar coords. is a map of a circle onto a rectangle! Complex variables, through a technique called <u>conformal mapping</u> provides many useful mappings.

2) Break the domain of integration into a union of simpler shapes if possible. Example:



Solve for the integral on each of these bits separately

3) Approximate the domain with a union of triangles or other shapes:



error due to approximation.

The error gets smaller as the triangles are smaller (finer discretization)

4) Imbed the domain in a rectangle or other shape and let $f(x,y)=0$ outside the domain. This is very simple to do, but leads to a discontinuity in $f(x,y)$ – and hence to quadrature errors

In general, mapping is the best approach – if you can do it! As always, there is a tradeoff between the difficulty in setting the problem up (e.g. a complex mapping) and the amount of computer work required (e.g., dealing with discontinuities).

Suppose the domain is a rectangle (or a rectangular panel in a more complex domain)

We wish to solve:

$$\int_a^b \int_\alpha^\beta f(x,y)\,dx\,dy$$

Let $\int_\alpha^\beta f(x,y)\,dx = \sum_{i=1}^{n} w_i\, f(x_i, y) + R_1(y)$

and $\int_a^b g(y)\,dy = \sum_{j=1}^{n} p_j\, g(y_j) + R_2$

Thus:

$$\int_a^b \int_\alpha^\beta f(x,y)\,dx\,dy = \int_a^b \left[\sum_{i=1}^{n} w_i f(x_i, y) + R_1(y)\right] dx$$

$$= \sum_{i=1}^{n} w_i \int_a^b f(x_i, y)\,dy + \int_a^b R_1(y)\,dy$$

Let $f(x_i, y) = g_i(y)$

Thus:

$$I = \sum_{i=1}^{n} w_i \left[\sum_{j=1}^{m} p_j f(x_i, y_j) + R_2 \right] + \int_a^b R_1 \, dy$$

$$= \sum_{i=1}^{n} \sum_{j=1}^{m} w_i p_j f(x_i, y_j) + \underbrace{\sum_{i=1}^{n} w_i R_2 + \int_a^b R_1 \, dy}_{R \text{ (total error)}}$$

This is a product formula: essentially we evaluate the integral over one variable first and then over the other. n & m are, in general, different.

If the two one-dim. rules are of degree $d_1 = 2n-1$, $d_2 = 2m-1$ (e.g., we use Gaussian quadrature) then the product rule will be exact for polynomials of the form:

$$x^p y^q : \quad p \le d_1, \; q \le d_2$$

The product rule is exact for a bivariate polynomial of degree $\min(d_1, d_2)$

but it will be exact for <u>some</u> polynomials of degree d_1+d_2.

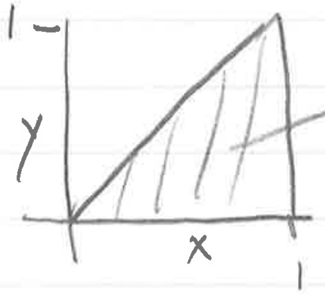
While this type of product rule is simple to code, it may not be the best choice. A 3x3 product rule using Gaussian quadrature is of bivariate degree 5 with 9 nodes.

There exists some rule of degree 5 with only 7 nodes! It's probably not worth trying to obtain it, however!

Mappings:

The most effective technique for doing multi-dim. quadrature is mapping. Usually we want to map a function onto a rectangle. This may do strange things.

Example: map a triangle onto a rectangle:



$$0 \leq x \leq 1$$
$$0 \leq y \leq x$$

We can map this onto a rectangle:

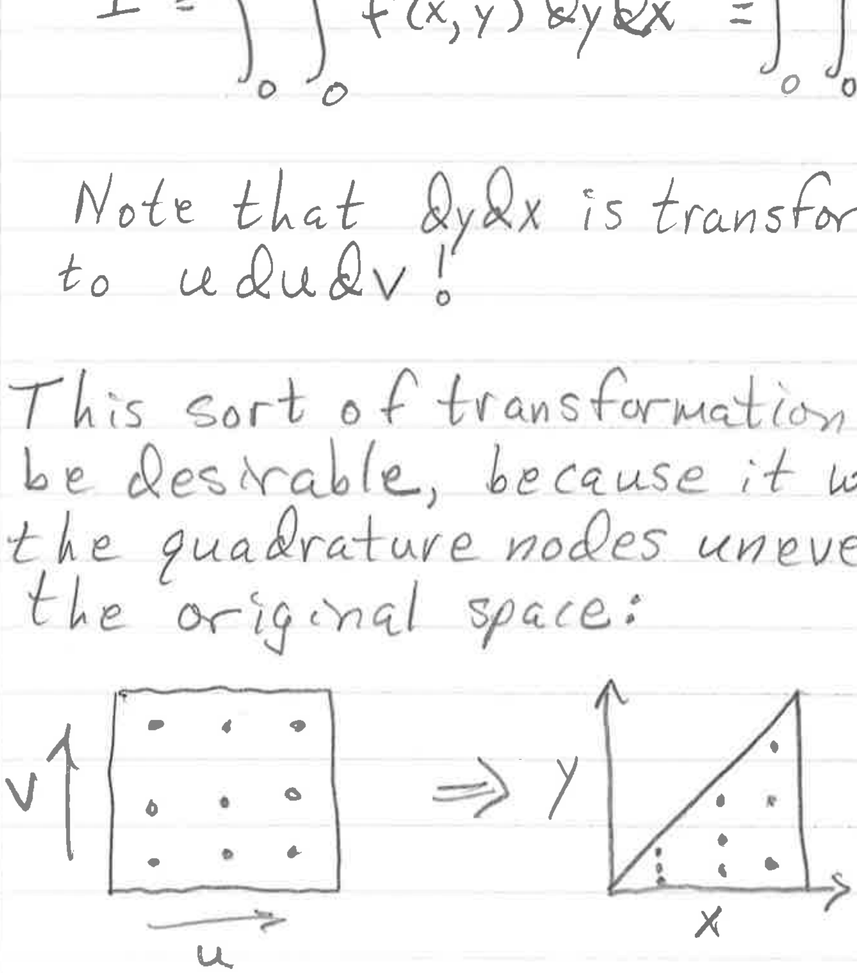
$$u = x \quad , \quad v = y/x$$

So that:

$$I \equiv \int_0^1 \int_0^x f(x,y)\,dy\,dx = \int_0^1 \int_0^1 f(u,uv)\,u\,du\,dv$$

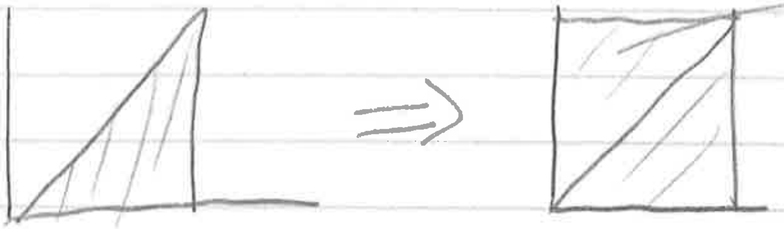
Note that $dy\,dx$ is transformed to $u\,du\,dv$!

This sort of transformation may not be desirable, because it will place the quadrature nodes unevenly in the original space:



Thus the function may be undersampled in some regions and oversampled in others. You should take this (and the spatial behavior of the function) into account when deciding on a mapping!

As an alternative, we could have <u>continued</u> the function by <u>reflection</u>



choose function to be the same as its mirror image!

Thus we pick $f(x,y)\Big|_{y>x} = f(y,x)$

and evaluate the integral over $0<x<1$ and $0<y<1$. The original integral will be half of this value.

While this leaves the points evenly distributed, it introduces a discontinuity in the derivative on the line $y=x$.

How can we apply adaptive quadrature to a 2-D integral?

If we have mapped to a rectangle, this is very simple:

we first evaluate over one variable, and then (in an outer loop or program) evaluate over the other!

Thus:
$$I = \int_a^b \left[\int_\alpha^\beta f(x,y)\,dy \right] dx = \int_a^b g(x)\,dx$$

where $g(x) = \int_\alpha^\beta f(x,y)\,dy$

Suppose the quadrature routine over y returns:

$$g(x) = g^*(x) + E_y(x)$$

where $E_y(x)$ is the error and g^* is the numerical result.

In the second integral (over x) we are not integrating g(x), but rather the numerical result $g^*(x)$:

$$\int_a^b g^*(x)\,dx = \int_a^b \left[g(x) - E_y(x) \right] dx$$

and this will have some error as well!

Thus the combined result returns:

$$I = \int_a^b g(x)\,dx = I^* + \int_a^b E_y(x)\,dx + E_x$$

Thus both the integral over y <u>and</u> x contribute to the error.

We require (in adaptive quadrature) that the total error be less than some threshold ε:

$$\left| \int_a^b E_y(x)\,dx + E_x \right| < \varepsilon$$

This is accomplished by setting individual tolerances for both the x and y quadratures. Thus:

$$|E_y| < \varepsilon_y \ , \ |E_x| < \varepsilon_x$$

where:

$$|(b-a)\varepsilon_y + \varepsilon_x| < \varepsilon$$

In general we require the inner tolerance to be smaller than the outer tolerance, say:

$$(b-a)\varepsilon_y \approx 0.1\,\varepsilon, \quad \varepsilon_x \approx 0.9\,\varepsilon$$

Feeding these values to the adaptive quadrature routines will result in the desired accuracy.

Monte Carlo Integration

So far all quadrature methods we have looked at are polynomial based - we are approximating a function locally with a high degree polynomial. Monte Carlo integration is completely different.

Suppose we have:

$$I = \int_a^b f(x)\,dx \equiv (b-a)\langle f(x)\rangle$$

where $\langle f(x)\rangle$ is the average of f over $[a, b]$.

The integral is thus just the average of the function times the width of the interval.

In Monte Carlo integration we want to compute this average!

We just pick N points chosen at random over the domain and then average the corresponding function values.

$$I \approx \Theta_N \equiv (b-a)\,\frac{1}{N}\sum_{i=1}^{N} f(x_i)$$

As N goes to infinity, the error in Θ_N will go to zero.

What is the error? For large N, Θ_N approximates a normal distribution.

Thus:

$$E(\Theta_M) = I$$

$$\sigma^2_{\Theta_N} = E\left\{ \frac{(b-a)^2}{N}\,\frac{1}{N}\sum_{i=1}^{N}\left(f(x_i) - \bar{f}\right)^2 \right\}$$

$$= \frac{(b-a)^2}{N}\,\sigma_f^2$$

where σ_f^2 is the variance of $f(x)$ over the interval.

The problem with this method is that it converges <u>very</u> slowly with N.

Even the Trapezoidal rule has an error which goes as $\frac{1}{N^2}$, while Monte Carlo integration error goes as $\sim 1/N^{1/2}$!

To get 3 significant digits, you need around 10^6 points!

The primary advantage of this procedure is in multi-dimensional integrals:

$$I = \int_D f(\underset{\sim}{x})\,dV$$

where $\underset{\sim}{x}$ is an m-dimensional vector. If m is large, this is difficult to code using polynomial based quadrature. Also, if (say) $m = 12$ and we use just 3 point quadrature in each dimension, we require over 500,000 points! with this number of points Monte Carlo integration may well be more accurate!

The most important use is in simulation of random media and related topics.

You choose a number of random configurations of particles distributed in a matrix (say) and calculate effective properties such as thermal conductivity or dielectric constant for each configuration. You then just average the values together!

Monte Carlo integration was originally developed by von Neumann to solve the neutron diffusion problem important in designing the A-bomb. We use similar approaches for measuring shear-induced diffusion in suspensions.

Prof. Maginn uses these techniques to examine shape/size selectivity of zeolite catalysts.

Ordinary Differential Equations

What is an ODE? It is a differential equation of a single variable

We begin with first order, linear ordinary differential equations

$$\frac{dy}{dt} = f(t)$$

This is very simple: we can solve it just using ordinary quadrature routines:

$$y = y(t_0) + \int_{t_0}^{t} f(t')\,dt'$$

where $y(t_0)$ is the initial condition

Often the problem is more complex, however. Suppose we have:

$$\frac{dy}{dt} = f(y,t)$$

If $f(y,t)$ is linear in y then the ODE is linear - it doesn't matter if it's non-linear in t.

We can't solve this using quadrature schemes, instead we must integrate it step by step!

Let's discretize the interval in time. Thus we have the series:

$$t_0, t_1, t_2, \ldots, t_n$$

with corresponding estimates of the dependent variable:

$$y_0, y_1, y_2, \ldots, y_n$$

we'll get into how these are made in a bit.

Often rather than a single equation we have multiple equations which are coupled:

Suppose we are examining the population of foxes and rabbits in a valley. Rabbits reproduce at a rate proportional to their number and get eaten by foxes, thus:

$$\frac{dr}{dt} = \underset{\substack{\uparrow \\ \text{reproduction} \\ \text{rate}}}{2r} - \underset{\substack{\nwarrow \\ \text{rate of} \\ \text{consumption}}}{\alpha r f}$$

Foxes also reproduce if there is enough food (rabbits) and die out without it, thus:

$$\frac{df}{dt} = (\alpha r - 1) f$$

These comprise a pair of coupled first order non-linear ODE's!

In general we have:

$$\begin{aligned} y_1' &= g_1(t, y_1, \ldots, y_n) \\ &\vdots \\ y_n' &= g_n(t, y_1, \ldots, y_n) \end{aligned}$$

In the last case $n=2$, thus:

$$y_1(t) = r(t), \quad y_2(t) = f(t)$$

$$g_1(t, y_1, y_2) = 2y_1 - \alpha y_1 y_2$$

$$g_2(t, y_1, y_2) = -y_2 + \alpha y_1 y_2$$

or $\underset{\sim}{y}' = \underset{\sim}{g}(t, \underset{\sim}{y})$

in vector form!

We can also write a single n^{th} order ODE as a system of first order ODE's

Consider the damped pendulum:

$$ML \frac{d^2\theta}{dt^2} = -Mg\sin\theta - 6\pi\mu a L \frac{d\theta}{dt}$$

The last term is the resistance due to viscosity of a sphere of radius a moving through a viscous liquid with velocity

$$L \frac{d\theta}{dt}$$

We can rewrite this as a pair of first order equations:

$$y_1 = \Theta \;,\quad y_2 = \frac{d\theta}{dt}$$

Thus:

$$y_1' = y_2 \qquad \text{(by definition)}$$

and $$y_2' = -\frac{Mg}{ML}\sin y_1 - \frac{6\pi\mu a L}{ML}\, y_2$$

For an arbitrary $n^{\underline{th}}$ order ODE we have:

$$y_1 \equiv y \,,\quad y_2 \equiv \frac{dy}{dt} \,, \quad \ldots \quad y_n = \frac{d^{(n-1)}y}{dt^{n-1}}$$

and:
$$y_1' = y_2$$
$$y_2' = y_3$$
$$\vdots \qquad \vdots$$
$$y_{n-1}' = y_n$$
$$y_n' = f(t, y_1, \ldots, y_n)$$

An important aspect of numerical integration is stability. Some equations will be easy to solve, while others may be very difficult.

Let's look at the single first order ODE:

$$\frac{dy}{dt} = f(y,t)$$

One solution technique is the explicit Euler method:

$$y_{k+1}^{EM} = y_k + f(t_k, y_k) \cdot h_k$$

where $h_k = t_{k+1} - t_k$ is the step size in time.

This method is equivalent to using the first two terms in a Taylor series for $y(t)$ at t_k and using the linear approximation to estimate $y(t_{k+1})$.

Methods which use information at t_k to predict y_{k+1} are called explicit

Let's see how this integration scheme works for a simple function:

$$\frac{dy}{dt} = y \;,\; y(0) = 1$$

This has the solution $y = e^{t}$

Let's see what the Euler Method does to this if $h_k = t_{k+1} - t_k = 0.1$:

	Numerical	Exact	Error
$y_0 =$	1	1	0
$y_1 = 1 + (1)(0.1) =$	1.1	1.1052	5.2×10^{-3}
$y_2 = 1.1 + (1.1)(0.1) =$	1.21	1.221	1.14×10^{-2}
$y_3 = (1.1)^3 =$	1.331	1.3499	1.89×10^{-2}
$\vdots$			
$y_{10} = (1.1)^{10} =$	2.593	2.7183	0.125
$\vdots$			
$y_{100} = (1.1)^{100} =$	1.38×10^{4}	2.20×10^{4}	0.82×10^{4}

So at $t = 10$ the error is nearly a factor of 2!

This amplification of error occured because of the differential equation itself. The equation

$\frac{dy}{dt} = y$ has the family of solutions:

$$y = ce^{t}$$

where c must be determined from the initial condition.

When you make an error in integration, you are wandering among this family of curves. Thus, if the curves are diverging in time, all errors will be amplified. Such a differential equation is unstable

Suppose we have instead that

$$\frac{dy}{dt} = e^{t}, \quad y(0) = 1$$

This has the same solution. What about the numerical error?

$$y_0 = 1$$

$$y_1 = 1 + e^0 \cdot (0.1) = 1.1$$

$$y_2 = 1.1 + 0.1\, e^{0.1} = 1.2105$$

$$\vdots$$

$$y_{10} = 1 + 0.1 \sum_{i=1}^{10} e^{(i-1)/10} = 2.634$$

w/ error 0.0845, which is about 2/3 the error of the previous equation, but:

$$y_{100} = 1 + 0.1 \sum_{i=1}^{100} e^{(i-1)/10} = 2.0943 \times 10^4$$

which is only off by about 5% rather than a factor of 2!

Why? In this case the family of curves is not diverging!

$$y = e^t + \underline{\underline{\underline{C}}}$$

Even though the solution is the same (for the same initial condition) the behavior is very different. For equations like this, the ODE is called neutrally stable.

What if we had the equation:

$$\frac{dy}{dt} = -y$$

In this case all of the family of solutions converge, thus the error tends to be wiped out. Such equations are stable

The stability is determined by the Jacobian:

$$J = \frac{\partial f}{\partial y}$$

Note that the derivative is with respect to the dependent variable!

If $J < 0$ the equation is stable

if $J > 0$ it's unstable

and if $J = 0$ it's neutrally stable!

We may have an equation which is stable in places and unstable in others. For example:

$$\frac{\partial y}{\partial t} = (1-t)y \qquad y(0) = 1$$

$$J > 0 \text{ for } t < 1 \qquad \text{unstable}$$

$$J < 0 \text{ for } t > 1 \qquad \text{stable}$$

We have the same for systems of equations. In this case the Jacobian is a matrix

$$\underset{\approx}{J} \equiv J_{ij} = \frac{\partial f_i}{\partial y_j}$$

The stability of a system of equations is related to the eigenvalues of $\underset{\approx}{J}$

The eigenvalues are a set of numbers λ_n which are the roots of the n^{th} order polynomial given by the determinant:

$$\left|\underset{\sim}{J} - \lambda \underset{\approx}{I}\right| = 0$$

determinant

identity matrix

The eigenvalues govern the stability. If all of the eigenvalues have a negative real part then the equations will be stable. If any have a positive real part then it will be unstable!

Let's work an example.

Let's look at the linearized damped pendulum. We linearize it for small Θ so that $\sin\Theta \cong \Theta$. Thus in dimensionless form:

$$\Theta'' = -2\underset{\uparrow}{d}\,\Theta' - \Theta$$

d is the damping coefficient

We let: $y_1 = \Theta$, $y_2 = \Theta'$

Thus:

$$y_1' = y_2$$

$$y_2' = -2\ell y_2 - y_1$$

The Jacobian is given by:

$$\underset{\approx}{J} = \begin{pmatrix} 0 & 1 \\ -1 & -2\ell \end{pmatrix}$$

Thus the eigenvalues are found from:

$$|\underset{\approx}{J} - \lambda \underset{\approx}{I}| = 0 = \begin{vmatrix} -\lambda & 1 \\ -1 & -2\ell-\lambda \end{vmatrix} = 0$$

or

$$(\lambda)(2\ell+\lambda)+1=0$$

which has the solutions:

$$\lambda = \frac{-2\ell \pm \sqrt{4\ell^2-4}}{2} = -\ell \pm \sqrt{\ell^2-1}$$

For all $\ell > 0$ the real part of λ_1 and λ_2 is negative, thus the equations are <u>stable</u>

If $d=0$, $\lambda_1 = i$, $\lambda_2 = -i$ where $i \equiv \sqrt{-1}$ and the real part is <u>zero</u>, so we are neutrally stable

If $d<0$ (not physical) then the real part is positive, and the equations are <u>unstable</u>

OK, how do we solve ODE's? As with quadrature, most methods are based on polynomial approximations!

The simplest method is just the Euler method. We examine the Taylor series expansion:

$$y(t+h) = y(t) + h\,y'(t) + \tfrac{1}{2}h^2 y''(\xi)$$

Recall that $y'(t) \equiv f(t, y(t))$

Thus:

$$y(t+h) = y(t) + h\,f(t, y(t)) + \tfrac{1}{2}h^2 y''(\xi)$$

In the Euler method we truncate after the linear term!

Let y_K be the estimate at $y(t_K)$

and thus:

$$y_{K+1}^{EM} = y_K + h f(t_K, y_K)$$

As we stated before, this rule is <u>explicit</u>, because y_{K+1} can be written as an explicit function of known quantities (t_K & y_K).

What is the error? Just subtract off the Taylor series!

$$y_{K+1}^{EM} - y(t_{K+1}) = y_K^{EM} - y(t_K)$$

$$+ h_K \left[f(t_K, y_K) - f(t_K, y(t_K)) \right]$$

$$- \frac{1}{2} h_K^2 \, y''(\xi)$$

The second term survives because there is some error from previous terms. Let's look at the term in brackets:

$$\left[f(t_k, y_k) - f(t_k, y(t_k))\right]$$

$$= \left[\cancel{f(t_k, y(t_k))} + (y_k - y(t_k)) \underset{\underset{\text{Jacobian}}{\uparrow}}{\frac{\partial f}{\partial y}} - \cancel{f(t_k, y(t_k))}\right]$$

$$= (y_k - y(t_k)) \frac{\partial f}{\partial y}$$

Thus the error at step $k+1$ is given by:

$$y_{k+1}^{EM} - y(t_{k+1}) = \left(y_k^{EM} - y(t_k)\right)\left(1 + h_k J\right) - \frac{1}{2} h_k^2 \, y''(\xi)$$

The term $\frac{1}{2} h_k^2 y''(\xi)$ is the <u>local error</u> at each step.

The quantity $(1 + hJ)$ is the <u>amplification factor</u>. Note that for all <u>unstable</u> ODE's this factor is greater than 1.

The key result is that if J is very negative, then Euler's method may be unstable as well!

If $hJ < -2$ then $|1+hJ| > 1$ and our method is numerically unstable.

Equations for which $J << -1$ are termed stiff

The interval of stability is given by:

$$-2 < hJ < 0$$

If you have a stiff equation, your step size had better not be too large!

For systems of equations, the Euler method will be stable if

$$\| \underset{\approx}{I} + h \underset{\approx}{J} \| < 1$$

This will be satisfied if

$$|1 + h\lambda| < 1$$

for all eigen values. This is a bit more complex since λ may have an imaginary part. Thus the interval of stability is now a region. The method will be stable if all $h\lambda$ lie within a circle of radius 1 centered about $z=-1$ in the complex plane.

How do we deal with stiff problems? We use implicit methods. The simplest is the Backward Euler method:

$$y^{BE}_{k+1} = y_k + h_k f(t_{k+1}, \underset{\uparrow}{y_{k+1}})$$

note implicit dependence

A method is called implicit if the equation for y_{k+1} depends on a function of y_{k+1}!

Ok, what does error propagation look like in this case? We determine it the same way:

$$y_{k+1}^{BE} = y_k + h\, f(t_{k+1}, y_{k+1})$$

$$y(t_k) = y(t_{k+1}) - h\, f(t_{k+1}, y(t_{k+1})) + \frac{1}{2}h^2 y''(\xi)$$

where we have expanded about t_{k+1} rather than t_k!

Subtracting, we get:

$$y_{k+1}^{BE} - y(t_{k+1}) \pm h\big(f(t_{k+1}, y_{k+1}) - f(t_{k+1}, y(t_{k+1}))\big) + y_k - y(t_k) \quad + \frac{1}{2}h^2 y''(\xi)$$

Again expanding $f(t_{k+1}, y_{k+1})$ we get:

$$y_{k+1}^{BE} - y(t_{k+1}) = h\frac{\partial f}{\partial y}\big(y_{k+1} - y(t_{k+1})\big) + y_k - y(t_k) + \frac{1}{2}h^2 y''(\xi)$$

Rearranging weget:

$$y_{k+1}^{BE} - y(t_{k+1}) = \frac{1}{1-hJ}\left(y_k - y(t_k) + \frac{1}{2}h^2 y''(\xi)\right)$$

In this case the amplification factor is less than 1 in magnitude for all $J<0$, nomatter how large h is!

Thus the interval of stability is:

$$hJ < 0$$

Note that if $J>0$ the problem is still unstable.

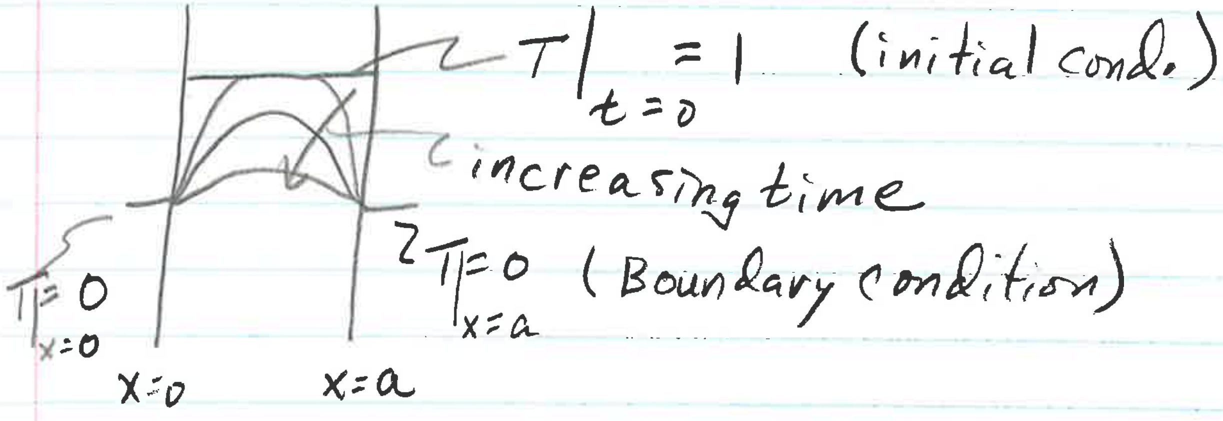
Implicit methods are very useful for stiff equations, but they do require the solution to a potentially non-linear equation at every step!

A very important application of these linear matrix methods is in the solution of Partial D.E.'s

Consider the heat equation:

$$\frac{\partial T}{\partial t} = \alpha \frac{\partial^2 T}{\partial x^2} \qquad (\alpha = \text{thermal diffusivity})$$

Suppose we look at the quenching of a hot slab of thickness a:


$T\big|_{t=0} = 1$ (initial cond.)
increasing time
$T\big|_{x=a} = 0$ (Boundary condition)
$T\big|_{x=0} = 0$
$x=0$
$x=a$

We can most easily solve this by discretizing the domain in x and tracking $T_i = T(x_i)$ as a function of time. This yields a system of first order ODE's.

Thus let:

$$x_i = i\Delta x \;;\; i = 0, 1, \ldots, n \;;\; \Delta x = \frac{a}{n}$$

Now the spatial derivative is:

$$\left.\left(\frac{\partial^2 T}{\partial x^2}\right)\right|_{x_i} \cong \frac{T_{i-1} + T_{i+1} - 2T_i}{\Delta x^2}$$

So:

$$\frac{\partial T_i}{\partial t} = \frac{\alpha}{\Delta x^2}\left\{T_{i-1} + T_{i+1} - 2T_i\right\}$$

for $i = 1, 2, \ldots, n-1$

and $T_0 = T_n = 0$ (Boundary Conditions)

Thus we have $n-1$ (excluding B.C.'s) coupled equations which can be solved as a <u>system</u> of first order <u>ODE's</u>

Matrix methods work very well for this:

$$\frac{\partial \underset{\sim}{T}}{\partial t} = \underset{\approx}{A}\,\underset{\sim}{T}$$

where :

$$\underset{\approx}{A} = \frac{\alpha}{\Delta x^2}\begin{bmatrix} 0 & - & - & - & - & 0 \\ 1 & -2 & 1 & 0 & \cdots & - \\ 0 & 1 & -2 & 1 & 0 & \cdots \\ & & & & & \\ 0 & \cdots & - & 0 & 1 & -2 \quad 1 \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 \end{bmatrix} \begin{array}{l} \leftarrow \\ \\ \\ \\ \\ \leftarrow \end{array}$$

first & last replaced by B.C.'s

The matrix A is just a tri-diagonal.

One problem with this, however, is that the Jacobian varies as $\alpha/\Delta x^2$, thus the set of equations is very stiff.

For an explicit numerical solution approach to be stable, we require

$$\Delta t < \frac{1}{2}\frac{\Delta x^2}{\alpha} \quad \text{(von Neumann Stability Crit.)}$$

which can get pretty small! Fortunately, implicit techniques are easy to code up for a linear system.

We have the explicit formulation:

$$\underset{\sim}{T}^{(K+1)} = \underset{\sim}{T}^{(K)} + \delta t \, \underset{\approx}{A} \, \underset{\sim}{T}^{(K)}$$

where the first and last equations (rows) are modified to account for the B.C.'s

An implicit formulation (B.E.) is:

$$\underset{\sim}{T}^{(K+1)} = \underset{\sim}{T}^{(K)} + \delta t \, \underset{\approx}{A} \, \underset{\sim}{T}^{(K+1)}$$

Rearranging yields:

$$\left(\underset{\approx}{I} - \delta t \, \underset{\approx}{A} \right) \underset{\sim}{T}^{(K+1)} = \underset{\approx}{I} \, \underset{\sim}{T}^{(K)}$$

where again the first and last rows are modified for the B.C.'s.

Since the problem is <u>linear</u>, it may be easily solved.

The implicit formulation is <u>much</u> more stable!

Thus far we have looked at explicit and implicit methods where the local error is $O(h^2 y'')$. We can take larger step sizes if we use higher order algorithms.

First, let's look at explicit methods.

The most common higher order methods are the Runge-Kutta techniques.

Recall that we have the Taylor series expansion:

$$y(t_{k+1}) = y(t_k) + y'(t_k)h_k + \frac{1}{2}h_k^2 y''(t_k) + y'''(\xi)\frac{h_k^3}{6}$$

Before in the Euler method we truncated after the linear term and had a local error which was $O(h^2)$. If we estimate y'', we can cause the local error to be $O(h^3)$! How can we do this?

One approach is the 2-stage Runge-Kutta technique.

Let:

$$K_1 = h f(t_n, y_n) \approx h \cdot y'(t_n)$$

$$K_2 = h f(t_n + h, y_n + K_1) \approx h \cdot y'(t_{n+1})$$

Thus $\frac{K_2 - K_1}{h^2} \approx y''(t_n)$

is an estimate of the second derivative. Plugging this back into the Taylor series leads to the formula:

$$y(t_n + h) = y(t_n) + h y'(t_n) + \frac{1}{2} h^2 y''(t_n) + O(h^3 y''')$$

So:

$$y(t_{n+1}) = y_n + K_1 + \frac{1}{2}(K_2 - K_1) + O(h^3 y''')$$

$$= \underbrace{y_n + \frac{1}{2}(K_1 + K_2)}_{\text{2-stage R-K formula}} + O(h^3 y''')$$

(It's called 2-stage because there are two function evaluations)

The local error is $O(h^3)$, the number of steps is $n \sim (b-a)/h$, thus this rule is second order

The Runge Kutta rule is more accurate than the Euler Method, but it is still explicit. It will run into trouble for very stiff equations.

Usually codes don't use 2-stage R-K rules, but rather 4-stage rules:

$$K_1 = h f(t_n, y_n)$$

$$K_2 = h f(t_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}K_1)$$

$$K_3 = h f(t_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}K_2)$$

$$K_4 = h f(t_n + h, y_n + K_3)$$

$$y_{n+1} = y_n + \frac{1}{6}(K_1 + 2K_2 + 2K_3 + K_4)$$

The local error is $O(h^5)$ and the overall rule is fourth order in the step size!

It is interesting to note that the R-K rules are not unique! In particular, Fehlberg came up with a 6-stage R-K technique which had an error that was 5th order, and then showed that 4 of these stages could be recombined to get a 4th order rule.

This is very useful because the difference between the two estimates gives an error estimate which can be used in adaptive step size control.

This Fehlberg R-K technique is implemented in the matlab routine ode45.m

All R-K techniques are explicit and hence run into trouble with stiff problems. What are the higher order implicit techniques?

The simplest is the trapezoidal rule!

The euler method uses the slope of $y(t)$ at t_k, the BE method uses the slope at t_{k+1}. If we average the two we can generate a second order rule!

Thus:

$$y^{TR}_{k+1} = \left(y^{EM}_{k+1} + y^{BE}_{k+1}\right)\frac{1}{2}$$

$$= y_k + \frac{1}{2}h\left(f(t_k, y_k) + f(t_{k+1}, y_{k+1})\right)$$

which is implicit

The error is a bit difficult to calculate, but eventually you get:

$$y^{TR}_{k+1} - y(t_{k+1}) = \frac{1+\frac{1}{2}hJ}{1-\frac{1}{2}hJ}\left(y_k - y(t_k)\right) + O(h^3)$$

Like the BE method, TR is stable for all $J<0$, but the amplification factor will approach -1 as $hJ \to -\infty$

$$\text{amp}^{BE} \approx \frac{1}{|Jh|} \quad \text{as} \quad Jh \to -\infty$$

$$\text{amp}^{TR} \approx \frac{\frac{1}{2}hJ}{-\frac{1}{2}hJ} = -1$$

which is close to instability! Thus the Trapezoidal rule may be no better than the BE method for sufficiently stiff problems!

What about higher order implicit techniques? If we just extended the TR to higher order, we would lose the infinite stability interval. Instead, we use a different approach: Multi-Value methods.

Consider the Taylor Series expansion:

$$y(t_{n+1}) = y(t_n) + h\,y'(t_n) + \frac{h^2}{2}y''(t_n) + \frac{h^3}{6}y'''(t_n) + \frac{h^4}{24}y^{IV}$$

The multivalue approach involves tracking each of these terms (the function and its derivatives) at every step. We can define the vector:

$$\tilde{y}(t_n) = \begin{pmatrix} y(t_n) \\ h y'(t_n) \\ \frac{1}{2}h^2 y''(t_n) \\ \frac{1}{6}h^3 y'''(t_n) \end{pmatrix}$$

We want to estimate $\tilde{y}(t_{n+1})$ given that we know $\tilde{y}(t_n)$. We can expand each of these derivatives in a Taylor Series as well:

$$h y'(t_{n+1}) = h y'(t_n) + h^2 y''(t_n) + \frac{h^3}{2} y'''(t_n) + \frac{h^4}{6} y^{IV}$$

$$\frac{h^2}{2} y''(t_{n+1}) = \frac{h^2}{2} y''(t_n) + \frac{h^3}{2} y'''(t_n) + \frac{h^4}{4} y^{IV}$$

$$\frac{h^3}{6} y'''(t_{n+1}) = \frac{h^3}{6} y'''(t_n) + \frac{h^4}{6} y^{IV}$$

where we have ignored terms which are of $O(h^5)$.

We can write this set of equations in vector notation:

$$\tilde{y}(t_{n+1}) = \underbrace{\begin{pmatrix}1&1&1&1\\0&1&2&3\\0&0&1&3\\0&0&0&1\end{pmatrix}}_{\underset{\approx}{B}} \tilde{y}(t_n) + \begin{pmatrix}1\\4\\6\\4\end{pmatrix}\frac{h^4}{24}\, y^{IV}$$

Let us define $\tilde{y}^{p}_{n+1} \equiv \underset{\approx}{B}\,\tilde{y}(t_n)$

Thus we have:

$$\tilde{y}_{n+1} = \tilde{y}^{p}_{n+1} + \begin{pmatrix}1\\4\\6\\4\end{pmatrix}\left(y^{IV}\frac{h^4}{24}\right)$$

Unfortunately, the quantity $\frac{h^4}{24}y^{IV}$ is unknown. We shall take it to be some constant a_n and pick it so that we satisfy our original ODE!

Recall that $y' = f(t, y)$ is our differential equation. Thus the second element of $\tilde{y}_{n+1}$ must satisfy this equation!

So:

$$\left(\tilde{y}_{n+1}\right)_2 \equiv \left(\tilde{y}^{p}_{n+1}\right)_2 + 4a_n = hf(t_{n+1}, y_{n+1})$$

↳ 2nd element

Hence we obtain:

$$4a_n = hf(t_{n+1}, y_{n+1}) - \left(y^{p}_{n+1}\right)_2$$

If we plug this back into the expression for $\tilde{y}_{n+1}$ we get:

$$\tilde{y}_{n+1} = \underset{\approx}{B}\,\tilde{y}_n + \underset{\sim}{r}\left[hf(t_{n+1}, y_{n+1}) - \left(\tilde{y}^{p}_{n+1}\right)_2\right]$$

where $\underset{\sim}{r} = \begin{pmatrix} 1/4 \\ 1 \\ 3/2 \\ 1 \end{pmatrix}$

Since we have ignored the $O(h^5)$ terms in the Taylor series, the local error is $O(h^5)$ and the method is 4th order!

The method is also implicit in the equation for y_{n+1}.

What about the stability of this method? It is rather messy, but you can show that it is always numerically unstable!

Just as Runge Kutta formulae are not unique, so are the choices for the vector $\underset{\sim}{r}$ which specifies the rule!

Two commonly used choices are:

Adams-Moulton : $\underset{\sim}{r} = \left(\frac{3}{8}, 1, \frac{3}{4}, \frac{1}{6}\right)^T$

Gear : $\underset{\sim}{r} = \left(\frac{6}{11}, 1, \frac{6}{11}, \frac{1}{11}\right)^T$

Adams-Moulton has a finite stability region, and is good for non-stiff problems

Gear has an infinite stability region and (although somewhat less accurate than Adams-Moulton) is the method most often used for stiff problems.

So we have the key integration methods:

<u>Explicit</u>

EM: $y_{n+1} = y_n + hf(t_n, y_n)$

– error is $O(h^2)$, overall $O(h)$

2sRK: $k_1 = hf(t_n, y_n)$

$k_2 = hf(t_n + h, y_n + k_1)$ ← EM est of y_{n+1}

$$y_{n+1} = y_n + \frac{1}{2}(k_1 + k_2)$$

local error $O(h^3)$, overall $O(h^2)$

4sRK – As in notes, local error $O(h^5)$ overall $O(h^4)$

<u>All</u> explicit methods have problems for stiff eq'ns, but are <u>easy</u> to code up!

Implicit

BE: $y_{n+1} = y_n + h f(t_n+h, y_{n+1})$

– error is same as EM (locally) but stable for stiff eq'ns!

TR: $y_{n+1} = y_n + \frac{h}{2}\left(f(t_n, y_n) + f(t_n+h, y_{n+1})\right)$

local error is same as 2s RK, but stable for stiff eq'n's!

Higher order methods are Adams-Moulton & Gear: You are unlikely to need to code these – use a canned routine...

Adaptive Step Size Control:

As in the case of quadrature, we want to pick a step size which will achieve some desired accuracy. Unfortunately the required step size may change from point to point as the integration proceeds. This leads to the need for adaptive schemes for controlling the step size.

To change the step size we need an estimate of the local integration error. Consider the 2-stage Runge Kutta rule:

$$K_1 = hf(t_n, y_n)$$

$$K_2 = hf(t_n + h, y_n + h)$$

$$y_{n+1} = y_n + \frac{1}{2}(K_1 + K_2)$$

The local error of this rule is $O(h^3)$ (overall, rule is second order)

Suppose we make one more evaluation:

$$K_3 = hf\left(t_n + \tfrac{h}{2},\ y_n + (k_1 + k_2)/4\right)$$

This is an estimate of the derivative at the midpoint of the interval!

We can combine this with the other two evaluations to get a third order rule:

$$y_{n+1} = y_n + \frac{1}{6}\left(k_1 + 4k_3 + k_2\right)$$

Note the similarity of this rule to Simpson's rule!

The local error is less than the difference between these two estimates:

$$\text{error} \sim \frac{1}{2}(k_1 + k_2) - \frac{1}{6}\left(k_1 + 4k_3 + k_2\right)$$

$$= \frac{1}{3}\left(k_1 - 2k_3 + k_2\right)$$

If the error is greater than some

chosen threshold, we reduce the step size. If the error is less than the tolerance, we may increase the step size.

How do we determine the optimum step size? The local error in the 2-stage rule (which dominates the error estimate) is locally $O(h^3)$

If the error estimate is Δ and the allowed error is τ, then:

$$\left(\frac{h_{opt}}{h}\right)^3 \approx \left(\frac{\tau}{\Delta}\right)$$

Thus $h_{opt} \approx h\left(\frac{\tau}{\Delta}\right)^{1/3}$

In practice, we pick our new h to be a little smaller than this value, say:

$$h_{opt} = 0.9\, h\left(\frac{\tau}{\Delta}\right)^{1/3}$$

We must also set an upper and lower limit to h. The upper limit will just be some fraction of the total interval of integration. The lower limit serves as a flag that we are approaching a singularity!

This method is implemented in the matlab routine ode23.m.

So far we have just looked at initial value problems - we specified the initial value of some vector $\underset{\sim}{y}$ and used the derivatives $\underset{\sim}{f}(t, \underset{\sim}{y})$ to march forward.

Often, however, we will not have an initial value problem. Instead, we have a boundary value problem: some of the values of $\underset{\sim}{y}$ are specified at both t_0 and t_{final}.

One way of solving this sort of problem is the shooting method. Basically, you assume some value for the unknown initial conditions and then integrate to the other boundary. The shooting parameters (the assumed initial conditions) are adjusted until the boundary conditions at the ending boundary are satisfied!

The name 'shooting method' comes about because this is the method used to determine elevation angle in firing artillery: the angle is adjusted by examining the trajectory of ranging shots until the target is hit!

Let's look at a simple example:

Let $y'' = y^2 - 1$ with $y(0) = 0$, $y(1) = 1$

We set this up as a system of equations:

$$y_1 \equiv y \,,\quad y_2 \equiv y'$$

$$\left.\begin{array}{l} y_1' = y_2 \\ y_2' = y_1^2 - 1 \end{array}\right\} \underset{\sim}{f} = \left\{\begin{array}{l} y_2 \\ y_1^2 - 1 \end{array}\right.$$

So $\underset{\sim}{y}' = \underset{\sim}{f}(t, \underset{\sim}{y})$

(although $\underset{\sim}{f}$ is not explicitly a function of t)

We have the initial condition

$$y_1(0) = 0$$

but we have no initial condition on y_2! We thus assume some value:

$$y_2(0) = X$$

We can now integrate the above expression. If we do this we will get some $y_1(1)$. We define:

$$g(x) = \underset{\substack{\nearrow \\ \text{Value at } t=1 \\ \text{for the initial} \\ \text{condition } y_2(0)=X}}{y_1(1)} - \underbrace{1}_{\substack{\hookrightarrow \text{boundary condition} \\ \text{we are trying} \\ \text{to satisfy}}}$$

The correct initial condition for y_2 is thus the solution to $g(x)=0$

We can thus solve this problem by using an integrator and a root solver. This is demonstrated in today's example.

A very different approach can be used to solve linear boundary value problems.

Suppose we have the linear ODE:

$$y'' + p(x)\,y' + q(x)\,y = g(x)$$

where p, q, and g are known functions.

we have the boundary conditions:

$$y(0) = a \;,\quad y(1) = b$$

We can solve this problem by reducing it to a set of linear algebraic equations.

We do this by using finite difference approximations for the derivatives.

First we discretize the domain:

$$x_1, \ldots, x_n \quad \text{s.t.} \quad x_{i+1} - x_i = h$$

and x_1 is the left edge while x_n is the right edge. We also have the corresponding y values:

$$y_1, \dots, y_n$$

such that $y_1 = a$ and $y_n = b$ from the boundary conditions.

We need to evaluate the derivatives using finite difference expressions:

$$(y'')_K \approx \frac{y_{K+1} - 2y_K + y_{K-1}}{h^2} + O(h^2 y^{IV})$$

$$(y')_K \approx \frac{y_{K+1} - y_{K-1}}{2h} + O(h^2 y''')$$

Note that both of these approximations are <u>second order</u> in the step size h. We thus get:

$$y_K'' + p(x_K)\, y_K' + q(x_k)\, y_K = g(x_k)$$

or, substituting in the above expressions:

$$y_{k+1}\left(\frac{1}{h^2}+\frac{h}{2}p(x_k)\right)+y_k\left(q(x_k)-\frac{2}{h^2}\right)$$

$$+\,y_{k-1}\left(\frac{1}{h^2}-\frac{h}{2}p(x_k)\right)=g(x_k)$$

This equation is valid for $2 \le k \le n-1$

It isn't valid at $k=1$ or n (the boundaries) because it requires knowledge of y_{k+1} and y_{k-1} and for $k=1$, y_0 is not defined, while for $k=n$ y_{n+1} is not defined.

Instead the equations at $k=1$ and n are replaced by the boundary conditions

$y_1 = a$ and $y_n = b$!

We now have n linear equations for the n unknowns $y_1, \ldots, y_n$! We can write this in vector form:

$$\underset{\approx}{A}\,\underset{\sim}{y}=\underset{\sim}{b}$$

and solve it in the usual way.

The matrix $\underset{\approx}{A}$ is tridiagonal

The main diagonal comprises the vector:

$$\left[1, \left(q(x_2)-\frac{2}{h^2}\right), \ldots, \left(q(x_{n-1})-\frac{2}{h^2}\right), 1\right]$$

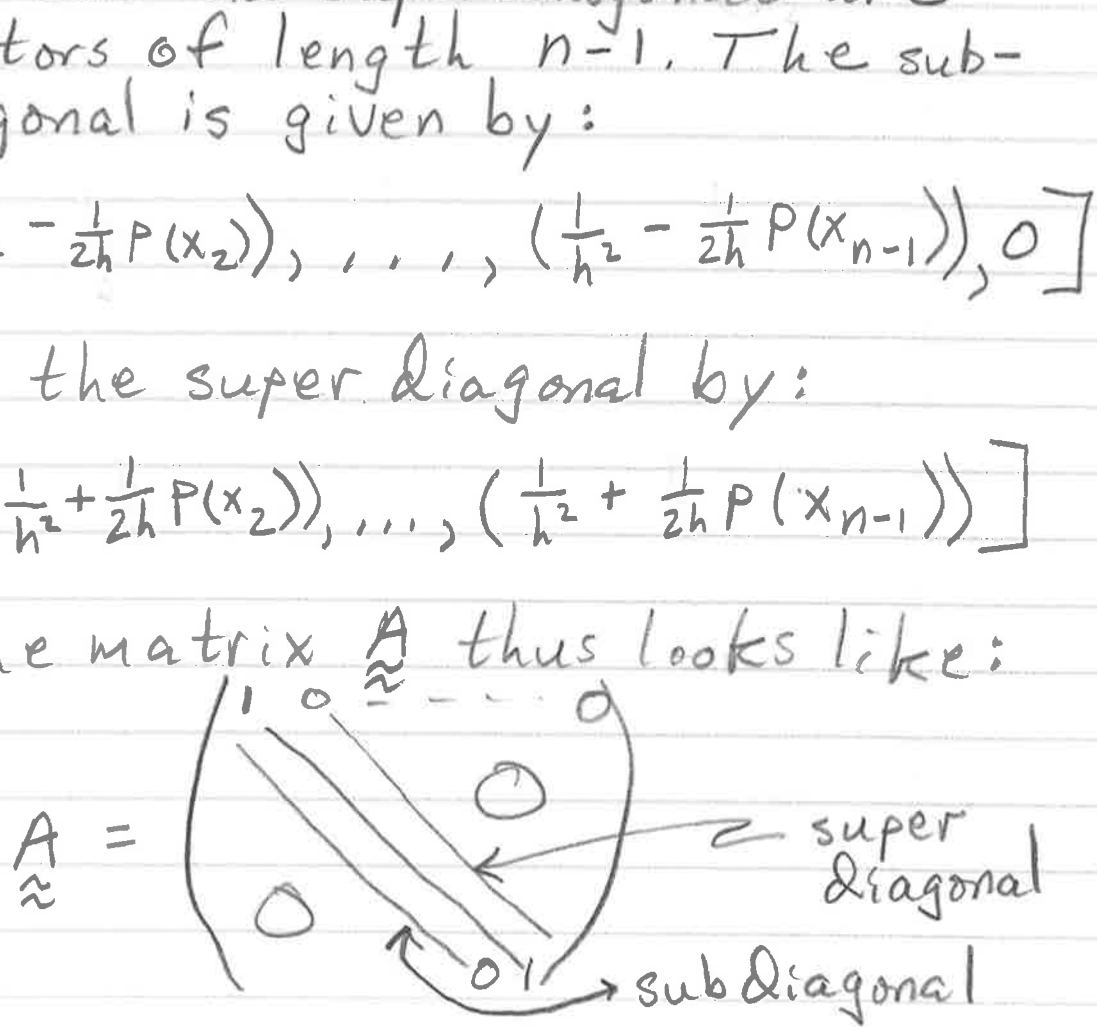
The sub and superdiagonals are vectors of length $n-1$. The sub-diagonal is given by:

$$\left[\left(\frac{1}{h^2}-\frac{1}{2h}P(x_2)\right), \ldots, \left(\frac{1}{h^2}-\frac{1}{2h}P(x_{n-1})\right), 0\right]$$

and the super diagonal by:

$$\left[0, \left(\frac{1}{h^2}+\frac{1}{2h}P(x_2)\right), \ldots, \left(\frac{1}{h^2}+\frac{1}{2h}P(x_{n-1})\right)\right]$$

The matrix $\underset{\approx}{A}$ thus looks like:



The first and last rows of $\underset{\approx}{A}$ come from the boundary conditions

The right hand side vector $\underset{\sim}{b}$ is just:

$$\underset{\sim}{b} = \begin{pmatrix} a \\ g(x_2) \\ \vdots \\ g(x_{n-1}) \\ b \end{pmatrix} \begin{matrix} \rightarrow \text{left B.C.} \\ \\ \\ \\ \rightarrow \text{right B.C.} \end{matrix}$$

Let's work a simple example:

$$y'' = 1 \qquad y(0) = 0, \; y(1) = \frac{1}{2}$$

The solution is just $y(x) = \frac{1}{2}x^2$

We will discretize the domain into three points ($h = 1/2$)

Thus $\underset{\approx}{A}\,\underset{\sim}{y} = \underset{\sim}{b}$ becomes:

$$\begin{pmatrix} 1 & 0 & 0 \\ 4 & -8 & 4 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1/2 \end{pmatrix}$$

Using Gaussian Elimination we get the solution:

$$y_1 = 0, \quad y_2 = \frac{1}{8}, \quad y_3 = \frac{1}{2}$$

Note that the value for y_2 is exact. There was no error because our approximation for the derivative y'' had an error $O(h^2 y^{IV})$. For this problem y^{IV} is zero, so the error vanished.

The solution procedure described above is equivalent to using a trapezoidal rule method where we solve all of the implicit equations simultaneously.

Sometimes we have boundary conditions on the derivative y' rather than the function y itself. This will modify the first and last row of $\underset{\approx}{A}$.

If we had $y'(0)=a$ we could use:

$$(y')_1 \approx \frac{y_2 - y_1}{h} + O(h y'')$$

but this rule, since it is not centered on x_1, is of order hy'' rather than $O(h^2 y''')$. We can improve this using an estimate of the second derivative:

$$(y')_1 \approx \frac{y_2 - y_1}{h} - \frac{h}{2}\,\frac{y_3 - 2y_2 + y_1}{h^2} + O(h^2 y''')$$

This reduces to the equation:

$$\frac{1}{h}\left[-\frac{1}{2}y_3 + 2y_2 - \frac{3}{2}y_1\right] = a$$

The right-hand-edge boundary condition $y'(1)=b$ has the equivalent expression:

$$\frac{1}{h}\left[\frac{3}{2}y_n - 2y_{n-1} + \frac{1}{2}y_{n-2}\right] = b$$

Arguably the most important class of second degree linear ODE's is the Sturm Liouville boundary value problem. These problems arise in many areas of mechanics, particularly in transport phenomena. For example, the PDE describing the time-dependent temperature profile in a hot slab whose surface is suddenly quenched can be described as a combination of Sturm-Liouville boundary value problems over the spatial variable and a simple exponential in time. You will see this next year.

These problems are characterized by the differential equation:

$$\left[p(x)\, y' \right]' - q(x)\, y + \lambda w(x)\, y = 0$$

where $\quad 0 \leq x \leq 1$

and: $\quad a_1 y(0) + a_2 y'(0) = 0$

$$b_1 y(1) + b_2 y'(0) = 0$$

and where p, p', q, and w are continuous functions of x on the interval $0 \le x \le 1$ and that $p(x) > 0$ and $w(x) > 0$ on $[0,1]$

This problem has a homogeneous differential equation and homogeneous boundary conditions! Thus the trivial solution $y = 0$ satisfies the problem!

For certain values of λ (called the eigenvalue), however, there will be non-trivial solutions $y(x)$ (called eigenfunctions). There will be an infinite number of each, with each eigenfunction corresponding to each eigenvalue.

We could solve this problem using the shooting method, varying λ as the shooting parameter. We can get an estimate of many eigenvalues at once by setting the problem up using the finite difference approach!

We can generate the finite difference representation of the derivative:

$$\left(\left[p(x)\, y' \right]' \right)_k \simeq$$

$$\frac{\dfrac{p(x_k + \frac{h}{2})(y_{k+1} - y_k)}{h} - \dfrac{p(x_k - \frac{h}{2})(y_k - y_{k-1})}{h}}{h}$$

$$= \frac{1}{h^2} \Big\{ p(x_k + \tfrac{h}{2})\, y_{k+1} - \big(p(x_k + \tfrac{h}{2}) + p(x_k - \tfrac{h}{2}) \big) y_k$$
$$+ p(x_k - \tfrac{h}{2})\, y_{k-1} \Big\} + O(h^2)$$

This, combined with discretization of q and w will give us the problem:

$$\underset{\approx}{A}\, \underset{\sim}{y} = \lambda\, \underset{\approx}{W}\, \underset{\sim}{y}$$

where $\underset{\approx}{W} = \begin{pmatrix} 0 & & & \\ & -w(x_2) & & 0 \\ & & \ddots & \\ 0 & & -w(x_{n-1}) & \\ & & & 0 \end{pmatrix}$

is a diagonal matrix.

This problem can be solved using the matlab eigenvalue solver

```
[v, d] = eig (A, W)
```

where d is a diagonal matrix containing the eigenvalues and V is the matrix of eigenvectors.

The example linked into today's notes solves a general Sturm-Liouville eigenvalue problem - you may want to hang on to this for use next year.

Have a Nice Summer!