Modelling and Simulation
EEEN 30150

Electronic, Electrical and Communications Engineering

Dr Paul Curran

Room 145, Engineering and Materials' Science Centre.

paul.curran@ucd.ie

+353-1-7161846

Modelling and Simulation
EEEN 30150

Before we begin some points should be made …

To reduce workload I propose to amend the assessment to this module from that described in the module descriptor, eliminating one component of the assessment, namely the problem sheet.

Modelling and Simulation
EEN 30150

Change to Module Descriptor:

Scheduled Learning (72 hrs): Lectures 24 hrs, Computer Laboratories 12 hrs (6 laboratories each of 2 hr duration), Laboratory reports 12 hrs (4 reports allowing 3 hrs additional completion time each). Minor projects 24 hrs (2 projects with12 hrs expected completion time each).

Autonomous Learning (43 hrs): 30 hrs (1 hr and 15 min per lecture post-familiarisation, additional reading and Matlab practice), 6 hrs (15 min per lecture pre-familarisation and additional reading), 3 hrs (30 min per laboratory preperation). 4 hrs (project research and choice).

Total: **115 hrs**

Modelling and Simulation
EEN 30150

Change to Module Descriptor (to reflect workload):

Workload per laboratory report = 30 min. lab. prep. + 2 hr lab. + 3 hr completion = 5.5 hrs.

Workload per minor project = 2 hrs choice and research + 12 hours development and completion = 14 hrs.

Ratio 14/5.5 = 28/11.

Assessment:

Laboratory Reports: 11% each = 44%

Minor Projects: 28% each = 56%

Modelling and Simulation EEEN 30150

The current assessment assigns 20% of the marks to the problem sheet which as stated I am not proposing to issue for the purposes of reducing the workload. Nonetheless, until the descriptor can be changed, gradebook requires that a grade/mark be issued for this component. Your percentage mark for this now fictitious component will be calculated as follows:

8.8(Laboratory report total percentage)/20 +11.2 (Minor project total percentage)/20

Modelling and Simulation EEEN 30150

The objectives of this module are, essentially, threefold. We wish to make a good start on developing your ability in:

Modelling: translating problems arising in practical engineering into the purely mathematical problem of having to solve mathematical equations of certain kinds.

Simulation: solving these mathematical equations to an acceptable level of approximation numerically, i.e. using a computer, specifically using the computer package Matlab.

Visualisation: presenting these approximate numerical solutions in the best possible manner.

Modelling and Simulation
EEEN 30150

In all three areas, as we are just taking the first steps in this process, we will have to settle for solutions which are considerably less than state-of-the-art. In modelling we will have to make a variety of idealisations, i.e. simplifications, to facilitate acquiring simple models, i.e. simple or at least relatively simple equations. In simulation we will employ methods which can be understood with relatively little previous background. Better methods are available for almost all of the problems which we consider, but they require far more background both to explain and to understand. In visualisation we employ the tools provided by Matlab. These are not the most efficient and do not look amazing. Animations for example will fall well short of the standards of professional movies such as Shrek or Cars. Nonetheless we will make some effort in this regard.

Modelling and Simulation
EEEN 30150

I am cognisant of the fact that the cohort of students attending this module is drawn from more than one specialisation. Accordingly in two of the four graded laboratories and in the two minor projects you will be faced with a list of problems from which you may choose. I am endeavouring to ensure that these problems will reflect a good variety of engineering and other disciplines. My hope is that many of you will find in some list a problem of at least some interest to you. As this module develops I will be looking to expand these lists. If you wish to propose your own problems you may bring such proposals to me. Should I ultimately accept your proposal and add it to the list you will earn my admiration (which is not easily earned and worthless) and you will also earn module credit (which is valuable).

Modelling and Simulation
EEEN 30150

Although idealised to facilitate modelling, the problems I hope, will have at least a reasonable level of practicality. I have no interest in setting essentially purely mathematical problems for non-mathematicians and either pretending or dogmatically stating that these have something to do with the real world. You will find, if you research in the library or on the internet, that it is far more common to find rather contrived problems for solution by the methods of this module, rather than problems which actually arose in some practical problem. One of the reasons for this of course is that practical problems do not generally lead to simple mathematical models, they generally lead to rather complicated ones. This observation notwithstanding I aspire to have at least some level of practicality in the problems of this module.

Modelling and Simulation
EEEN 30150

You will at first perhaps be dismayed by the volume of notes accompanying this module. In general I have far more notes on each topic than I can possibly cover in the number of lectures assigned to the topic. I feel that I should explain myself. According to your specialisation you may choose which examples to study in detail and which examples to study less carefully or indeed not at all. The idea of the module, both in terms of material and assessment is that, whereas there is a certain common core, much of the module is of your own choosing. I do not expect that many students will study in detail all of the material. However, having all of this material does make one point rather forcefully for any who choose to read sufficiently to observe it. All of your disciplines ultimately lead to mathematical models which differ from each other in terms of notation only and the numerical methods for solving the resulting equations are identical for all of you.

Modelling and Simulation
EEEN 30150

The various institutions (national and international) which oversee the standards of engineering education within this university in recent years have been concerned to see development of presentation skills, group work and team work included within the curriculum. These are recognised as being skills which are genuinely useful when working in industry. It seems opportune to include these topics in the minor projects. I do not want to have too great a portion of the grade dependent on team work however. Accordingly only the second minor project will be undertaken as part of a team. Teams will generally consist of three students. A team differs from a group in that members of a team have different roles. The roles of the team members will be model development, numerical solution and visualisation. Each team member will be responsible for one ot these areas but will have knowledge of all. Each team will present their work to a panel of judges and an audience.

Modelling and Simulation
EEEN 30150

Obviously teams will have to be formed. If you wish to have a say in the team in which you will work that is entirely acceptable. Please communicate your wishes to me early giving the names and student numbers of all three team members. Otherwise I will form teams by alphabetical order within specialisations. I will in general not form teams with students from different disciplines, although of course it is arguable that such interdisciplinary teams would likely be amongst the best. Teams, like groups, can have, indeed mostly will have weaker members. In part this is why I favour team rather than group work where a weaker member has a much greater adverse effect. I'm afraid that is just the nature of the beast. You will in your career be faced with the challenge of ensuring a good team performance in spite of a team membership which is not your vision of the ideal.

Modelling and Simulation
EEN 30150

As is always the case each of you will have your own level of engagement with this module. For some perhaps that engagement will be rather weak (all of the lectures being scheduled for 09.00 will play a role here no doubt) or rather ineffective (because your prior knowledge is lacking). To these latter I encourage you to engage with the TAs and with myself. If you are truly trying I assure you that I will gladly do my best to help.

Some I hope will find something in this module of genuine interest to them. Their engagement is likely to be much higher. You will find an abundance of textbooks on numerical methods for all types of engineers and scientists in the library. You will also find a deluge of websites dealing with the various topics which we will cover. Wikepedia will in almost every case offer a very good first point of reference, indeed you will find it one of the sources for some of my information.

Overview of Topics

Solution of equations by iteration.

Optimisation.

Solution of system of linear equations.

Solution of ordinary differential equations: initial value problems.

Numerical Quadrature.

Solution of ordinary differential equations: boundary value problems.

Solution of partial differential equations.

Section 1 – Solution of equations by iteration

- Elementary method: Regula Falsi
- Advanced method: Newton-Raphson method.

Workload: 3 lectures + 1 laboratory + Autonomous learning

Roots of an equation

e.g. $\sqrt{2}$ is a real root of the quadratic function

$$f(x) = x^2 - 2$$

i.e. a solution of the quadratic equation

$$f(x) = x^2 - 2 = 0$$

But what is the value of $\sqrt{2}$ in, for example, decimal form?

Roots of an equation

One of the critical ideas of numerical methods is to seek approximate solutions to problems when exact solutions are unavailable. In this respect first dynasty Babylonia (ca. 1800 - 1600 BC), the world's first great scientific civilisation, offers a sexagesimal (base 60) approximation to $\sqrt{2}$

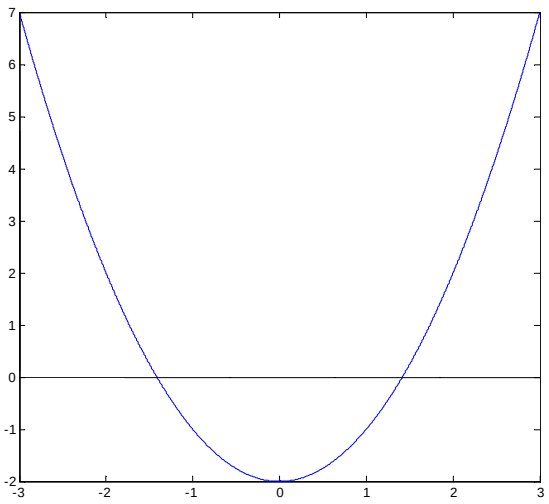
$$\sqrt{2} \cong 1;24,51,10 = 1 + \frac{24}{60} + \frac{51}{60^2} + \frac{10}{60^3} = 1.41421296$$

which is accurate to five decimal digits.

$f(x) = x^2 - 2$

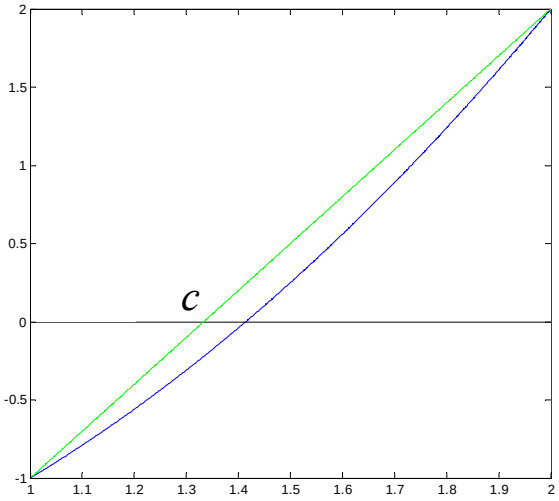
A plot of $f(x)$ vs x reveals that one root occurs between 1 and 2, with the second root being negative.

Clearly as $\sqrt{2}$ is positive it is the root lying between 1 and 2 with these values acting as lower and upper bounds on the root.



$$f(x) = x^2 - 2$$

Note that $f(1) = -1$ and $f(2) = 2$. Draw the *chord* between the points $(1, -1)$ and $(2,2)$. The point where the chord crosses the x-axis is $(c,0)$. Clearly the value c gives a better approximation to $\sqrt{2}$.

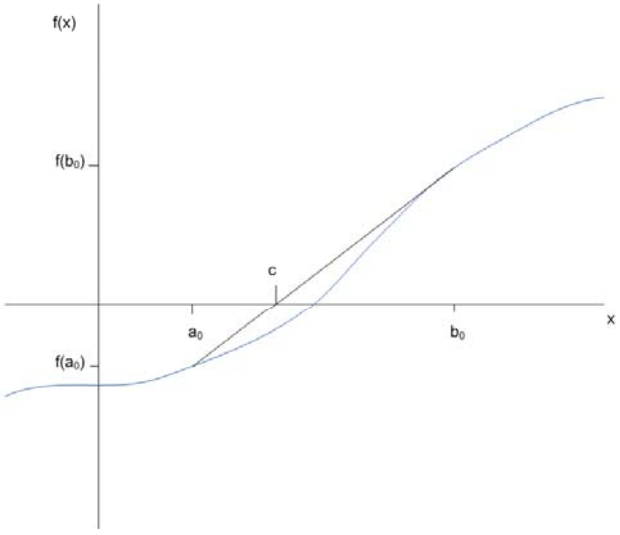


As the point $(c,0)$ is the point of intersection of two lines it is elementary to calculate.

Consider the problem in general.

$$f(a_0)f(b_0) < 0$$

Seek intersection of chord with x-axis.



Point of Intersection

The chord is a line having equation

$$y - f(a_0) = m(x - a_0)$$

It also contains the point $(b_0, f(b_0))$ so that

$$m = \frac{f(b_0) - f(a_0)}{b_0 - a_0}$$

The point of intersection with the x-axis is $(c,0)$ giving: $0 - f(a_0) = m(c - a_0)$

$$c = a_0 - \frac{f(a_0)}{m} = a_0 - \frac{f(a_0)(b_0 - a_0)}{f(b_0) - f(a_0)}$$

Regula Falsi (False Position)

Given a_0 and b_0 such that $f(a_0)f(b_0) < 0$ formula

$$c = a_0 - \frac{f(a_0)(b_0 - a_0)}{f(b_0) - f(a_0)} = \frac{f(b_0)a_0 - f(a_0)b_0}{f(b_0) - f(a_0)}$$

gives a better approximation to the root. For the above example:

$$f(x) = x^2 - 2 \quad , \quad a_0 = 1 \quad , \quad b_0 = 2$$

$$c = 1 - \frac{(-1)(2-1)}{2-(-1)} = 1 + \frac{1}{3} = 1.3333$$

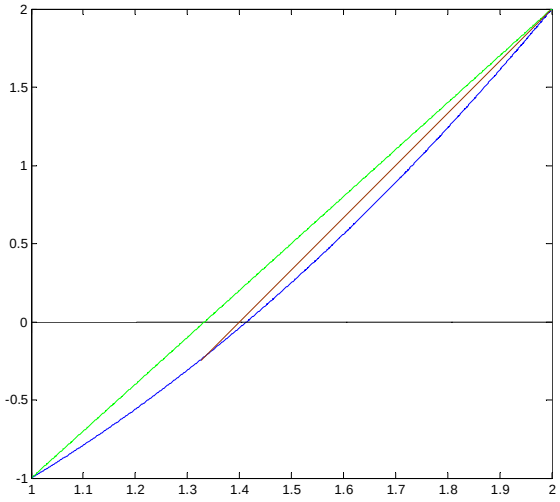
Iteration

$f(c) = (16/9) - 2 = -2/9$. Clearly this is closer to zero than $f(1) = -1$ or $f(2) = 2$. The approximation is an improvement, nevertheless $f(c)$ is *not* zero.

As $f(c) = -2/9 < 0$ and $f(2) = 2 > 0$ it is clear that the root lies between $c = 1.3333$ and 2. Application of the formula has permitted us to improve our lower bound for the root from 1 to 1.3333. Our upper bound of 2 remains unchanged in this case.

Iteration

The critical idea, not just for this example but for numerical analysis in general, is that we can apply Regula Falsi *again*.



Iteration

Let $a_1 = 1.3333$ and $b_1 = 2$, the new lower and upper bounds on the root.

$$f(x) = x^2 - 2 \quad , \quad a_1 = 1.3333 \quad , \quad b_1 = 2$$

$$c = \tfrac{4}{3} - \frac{\left(-\tfrac{2}{9}\right)\left(2 - \tfrac{4}{3}\right)}{2 - \left(-\tfrac{2}{9}\right)} = \tfrac{4}{3} + \tfrac{1}{15} = \frac{7}{5} = 1.4$$

$f(c) = -1/25 < 0, f(2) = 2 > 0$ so the root lies between $c = 1.4$ and 2. The new lower and upper bounds on the root are $a_2 = 1.4$ and $b_2 = 2$.

Iteration using Matlab Command Line

To implement this iteration in Matlab we note that we have two sequences a_n, b_n and an additional internal quantity c. Accordingly in Matlab we will create three variables, **a**, **b** and **c**. For the problem at hand the initial value of **a** (a_0) is 1 and that of **b** (b_0) is 2. We create the variables **a**, **b** and initialise them in one command:

```
>> a = 1;      create and initialise a
>> b = 2;      create and initialise b
```

Iteration using Matlab Command Line

Comment:

```
>> a = 1;      create and initialise a
>> b = 2;      create and initialise b
```

I present comments upon the particular command in green italic. This is *not* Matlab code. You can insert comments into Matlab code and Matlab will colour them green but they must be preceeded by a % symbol to indicate to Matlab that they are comments. Take careful note, **I have left this symbol out**.

Iteration using Matlab Command Line

Before we commence the iteration we need to check that $f(a_0)$ and $f(b_0)$ have opposite signs, since the iteration will not offer an improved solution unless this condition holds:

```
>> fa = (a^2) – 2      find value of f(a)
>> fb = (b^2) – 2      find value of f(b)
```

If $f(a_0)$ and $f(b_0)$ have opposite signs we calculate the additional internal quantity c. We create and evaluate the first value of variable **c** in one command:

```
If f(a) and f(b) have opposite sign:
>> c = a – (fa*(b – a)/(fb – fa)  update c using old a and b
```

Iteration using Matlab Command Line

Find the sign of $f(c)$.

```
>> fc = (c^2) – 2      find value of f(c)
```

If $f(c)$ is zero stop. Answer is **c**. If $f(c)$ and $f(a_0)$ have the same sign we should use **c** as new lower bound, replacing a. If $f(c)$ and $f(b_0)$ have the same sign we should use **c** as new upper bound replacing **b**.

```
If f(c) = 0 stop. If f(c) and f(a) have same sign:
>> a = c;     update a
Else:
>> b = c;     update b
```



```
>> a = 1;       initialise a
>> b = 2;       initialise b
>> fa = (a^2) – 2       determine value of f(a)
>> fb = (b^2) – 2       determine value of f(b)
If f(a) and f(b) have opposite sign:
>> c = a – (fa)*(b – a)/(fb – fa)       update c using old a and b
>> fc = (c^2) – 2       determine value of f(c)
If f(c) = 0 stop. If f(c) and f(a) have same sign:
>> a = c;       update a
Else:
>> b = c;       update b
```

Iteration using Matlab Command Line

Strictly the update when $f(c)$ and $f(a_0)$ have same sign should be: a = c and b = b, i.e. update lower bound replacing it with c and update upper bound *leaving it unchanged*. There is clearly no point in executing the second of these commands, since it does nothing. Now the key idea of iteration is to repeat all but the first two initialisation steps of the previous list of commands, unless we happened to achieve the hard termination condition $f(c) = 0$, which is unlikely. Using blue to identify once off commands and black to identify repeated commands (repeated in same order) yields:


```
>> a = 1;          initialise a
>> b = 2;          initialise b
>> fa = (a^2) – 2      determine sign of f(a)
>> fb = (b^2) – 2      determine sign of f(b)
If f(a) and f(b) have opposite sign:
>> c = a – (fa*(b-a)/(fb – fa))        update c using old a and b
>> fc = (c^2) – 2      determine sign of f(c)
If f(c) = 0 stop. If f(c) and f(a) have same sign:
>> a = c;          update a
Else:
>> b = c;          update b
```

Iteration using Matlab Command Line

The *uparrow* key is a useful one in this regard since it scrolls back up to previously executed commands. One only has to be careful to execute the commands which are to be repeated in the same order every time. The code is not the most efficient. It shows a repeated evaluation of variables fa and fb although one of these only has changed since the previous calculation. Moreover it shows a repeated determination by the user of whether $f(a)$ and $f(b)$ have opposite signs, although the nature of the algorithm is such that if this is true the first time then it is true at all subsequent times.

Iteration

Repeated application of Regula Falsi gives:

$a_0 = 1.0000\ ,\ b_0 = 2.$

$a_1 = 1.3333\ ,\ b_1 = 2.$

$a_2 = 1.4000\ ,\ b_2 = 2.$

$a_3 = 1.4118\ ,\ b_3 = 2.$

$a_4 = 1.4138\ ,\ b_4 = 2.$

$a_5 = 1.41421\ ,\ b_5 = 2.$

$a_6 = 1.4142\ ,\ b_6 = 2.$

It happens in this case that the upper bound b does not change.

Correct to four decimal places after six iterations.

General Regula Falsi

Given $a_n < b_n$ such that $f(a_n)f(b_n) < 0$ let

$$c = a_n - \frac{f(a_n)(b_n - a_n)}{f(b_n) - f(a_n)}$$

If $f(c) = 0$ root equals c. Terminate.

If $f(a_n)f(c) > 0$ let $a_{n+1} = c$, $b_{n+1} = b_n$

If $f(a_n)f(c) < 0$ let $a_{n+1} = a_n$, $b_{n+1} = c$

Deficiencies of Regula Falsi

The initial lower and upper bounds a_0 and b_0 must be found by some unspecified additional method. Their determination does not form part of the Regula Falsi algorithm.

The algorithm is considered to be rather slow.

As presented the algorithm is incomplete. One must identify some kind of termination conditions. The condition of actually finding a root will not suffice as it will very rarely hold. Typical conditions are (i) that $|f(c)|$ is below a prespecified tolerance or (ii) that the number of steps has hit a prespecified limit.

Newton-Raphson Algorithm

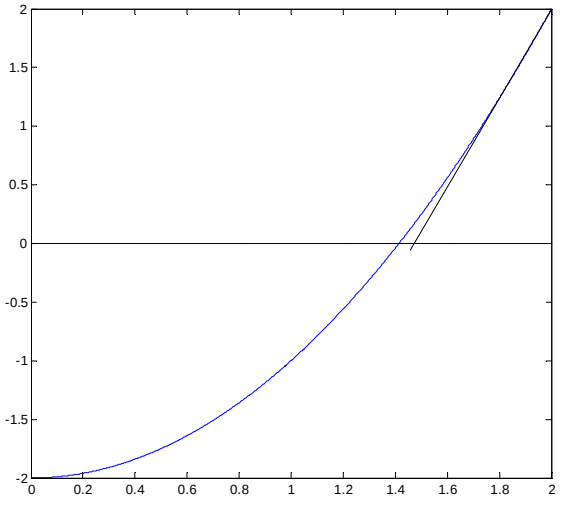
It is the speed (or lack thereof) with which Regula Falsi converges to a root that is its principal drawback. Prior to computers/calculators calculations had to be done by hand with mistakes being easily made (although given their level of practice and the importance of their task, human calculators were far more accurate than they are, in general, today).

Independently Isaac Newton and Joseph Raphson developed an alternative algorithm for finding roots which, in terms of speed, when it worked was greatly superior to all preceding root-finding algorithms. A deficiency was that it did not always work.

$f(x) = x^2 - 2$

Plot $f(x)$ vs x.

2 is a poor approximation to the positive root. Draw tangent to curve at $x = 2$. The intercept of this tangent with the x-axis gives a better approximation to the root.



Point of Intersection/Intercept

In general let the initial guess for the root be x_0. The tangent to the curve at $x = x_0$ is a line having equation

$$y - f(x_0) = f'(x_0)(x - x_0)$$

The point of intersection with the x-axis is $(x_1,0)$ giving $0 - f(x_0) = f'(x_0)(x_1 - x_0)$

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

Iteration

With $f(x) = x^2 - 2$ and $x_0 = 2$ we have:

$$x_1 = 2 - \frac{2}{4} = 1.5$$

where we employ the elementary result:

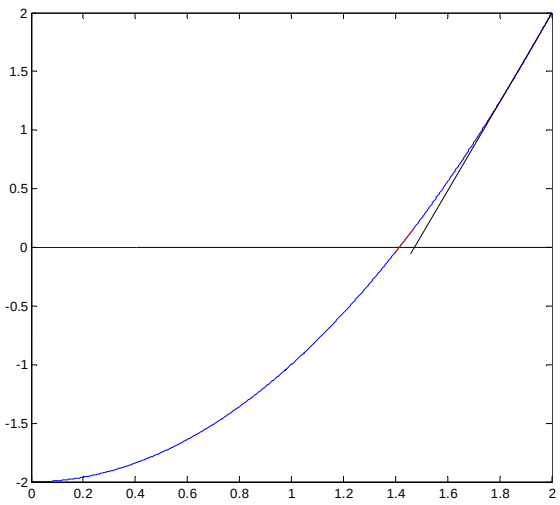
$$f'(x) = 2x$$

As $f(x_0) = f(2) = 2$ and $f(x_1) = f(1.5) = ¼$ we have an improvement in the approximation of the root. Nevertheless the approximation 1.5, although better, remains rather poor.

Iteration

The critical idea once more is to apply Newton-Raphson *again*.

The figure, in which we see no discrepency, indicates that we expect a very good approximation.



Iteration

Repeated application of Newton-Raphson gives:

$x_0 = 2$

$x_1 = 1.5$

$x_2 = 1.4167$

$x_3 = 1.4142$

Correct to four decimal places after three iterations.

Iteration using Matlab Command Line

```
>> x = 2;          initialise x
>> x = x – ((x^2) – 2)/(2*x)   update x using old x
```

As above the format for presentation of command line Matlab code will be as follows:

Blue: initialisation and setting of parameters, commands executed once only.

Green italic: comment on purpose of step.

Black: steps which are subsequently repeated.

Newton-Raphson Algorithm

Given x_n let

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

If $f(x_{n+1}) = 0$ root equals x_{n+1}. Terminate.

If $f(x_{n+1}) \neq 0$ continue until termination condition.

Terminate if $|f(x_{n+1})|$ below prespecified tolerance or number of steps taken reaches prespecified maximum.

Leonardo of Pisa's equation

In 1225 John of Palermo posed to Leonardo of Pisa (Fibonacci) the problem of solving the following cubic equation:

$$f(x) = x^3 + 2x^2 + 10x - 20 = 0$$

which he "borrowed" from Omar Khayyam.

Leonardo makes the following immensely important decision and comment: "… *because it was not possible to solve this equation in any other of the above ways, I worked to reduce the solution to an approximation*".

Leonardo of Pisa's equation

Leonardo worked long before the onset of the modern obsession with the decimal system. He offers a solution in the sexagesimal (base 60) system: $x = 1;22,07,42,33,04,40$ meaning

$$x = 1 + \frac{22}{60} + \frac{7}{60^2} + \frac{42}{60^3} + \frac{33}{60^4} + \frac{4}{60^5} + \frac{40}{60^6}$$

In fact there is an error in this answer. The sixth sexagesimal digit should be 38 or 39 (for lower or upper bound respectively) but assuredly not 40.

In decimal form Leonardo's answer is:
$x = 1.368,808,107$

Leonardo of Pisa's equation

Application of Regula Falsi with $a_0 = 1$ and $b_0 = 2$ gives, as in the previous problem, an unchanging upper bound $b = 2$. The lower bound does change however, giving:

Regula Falsi

$a_0 = 1$

$a_1 = 1.304,347,826$

$a_2 = 1.357,912,304$

$a_3 = 1.366,977,804$

$a_4 = 1.368,500,975$

$a_5 = 1.368,756,579$

$a_6 = 1.368,799,462$

$a_7 = 1.368,806,657$

$a_8 = 1.368,807,864$

$a_9 = 1.368,808,066$

$a_{10} = 1.368,808,100$

$a_{11} = 1.368,808,106$

$a_{12} = 1.368,808,107$

Convergence to nine decimal places in twelve steps.

Newton-Raphson

On the other hand application of Newton-Raphson with $x_0 = 1$ gives:

$x_0 = 1$

$x_1 = 1.411,764,705$

$x_2 = 1.369,336,470$

$x_3 = 1.368,808,188$

$x_4 = 1.368,808,107$

Convergence to nine decimal places in four steps.

Safeguards

Although the Newton-Raphson algorithm converges much more rapidly than Regula Falsi in the example considered it does have a significant potential problem. Regula Falsi generates a lower and upper bound, with the guarantee that at least one root of the function lies between these bounds (provided the function is continuous). Newton-Raphson only provides an estimate of the solution. There is no guarantee that the difference between that estimate and an actual root is small, no guarantee that it does not increase rather than decrease.

Safeguards

In practice, given an algorithm capable of taking very large steps, such as the Newton-Raphson algorithm we should implement safeguards to ensure that the approximate answer does not simply jump out of the region of interest and miss the root entirely. It is common, for this purpose, to combine the Newton-Raphson iteration with a much slower method called *bisection*. The bisection is used simply to keep a check on the Newton-Raphson iteration and to ensure that it is converging, as distinct from simply rapidly changing.

Bisection or Interval-Halving

Given $a_n < b_n$ such that $f(a_n)f(b_n) < 0$ let

$$c = \frac{a_n + b_n}{2}$$

If $f(c) = 0$ root equals c. Terminate.

If $f(a_n)f(c) > 0$ let $a_{n+1} = c$, $b_{n+1} = b_n$

If $f(a_n)f(c) < 0$ let $a_{n+1} = a_n$, $b_{n+1} = c$

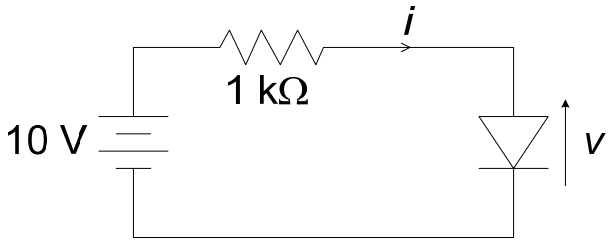
There is guaranteed to be a root in interval $[a_n,b_n]$ for every step n. Clearly the length of the interval *halves* with each step, hence the name.

Safeguards

Bisection has the disadvantage that it is generally very slowly converging, however, it has the great advantage that it *always* converges. If the Newton-Raphson estimate leaves the current interval determined by the bisection method then it has jumped too far and has overlooked a root. A proper reaction to this is to restart the Newton-Raphson with a new initial guess equal to either the upper or lower bound of this interval, whichever gives a value for the function closer to zero. Eventually the approximation will be sufficiently close and from this point onwards Newton-Raphson will converge rapidly to the root. We therefore retain the speed advantage of Newton-Raphson while suppressing its capacity to rapidly diverge.

Application of Newton-Raphson Method: nonlinear electronic circuit

The circuit shown is analysed using Kirchoff's voltage law, Ohm's law and Shockley's ideal diode equation



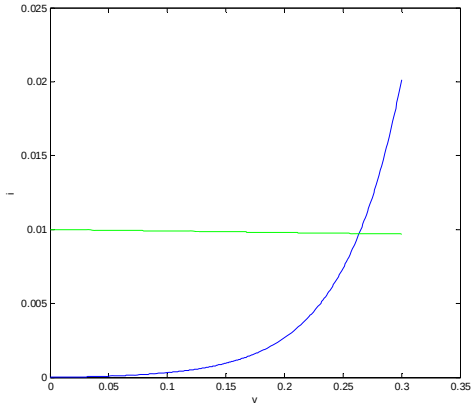
$$10 = v + Ri \qquad R = 1\,\mathrm{k\Omega}$$

$$i = I_s\left(\exp\left(\frac{v}{nV_T}\right) - 1\right) \qquad I_s = 50\,\mu\mathrm{A} \qquad nV_T = 50\,\mathrm{mV}$$

Nonlinear electronic circuit

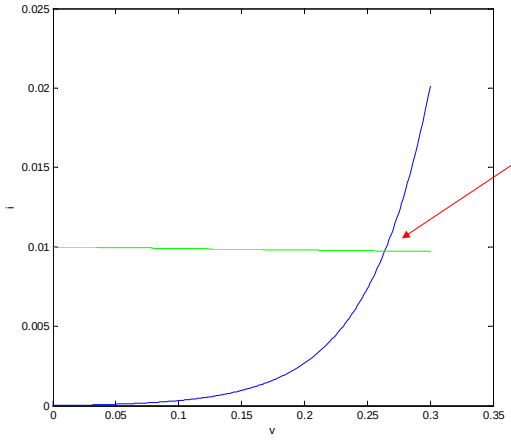
$10 = v + 1000i$ graph is green line

$i = 50 \times 10^{-6}\left(\exp(20v) - 1\right)$ graph is blue curve



Nonlinear electronic circuit

$$f(v) = 10 - v - \tfrac{1}{20}\left(\exp(20v) - 1\right) = 0$$



Solution

Nonlinear electronic circuit

$$f(v) = 10 - v - \tfrac{1}{20}\left(\exp(20v) - 1\right) = 0$$

```
>> v = 0.2;      initialise v
>> v = v – ((10 – v – ((1/20)*(exp(20*v) – 1)))/(– 1 – exp(20*v)))
        update v using old v
```

$v_0 = 0.2$

$v_1 = 0.3281$

$v_2 = 0.2919$

$v_3 = 0.2704$

$v_4 = 0.2642$

$v_5 = 0.2638$

Application of Newton-Raphson Method: floating football

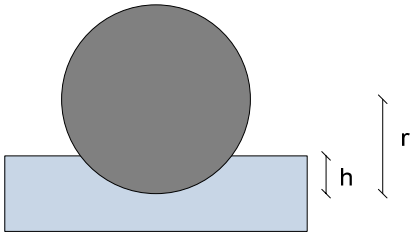
FIFA "laws of the game" specify that a football should be spherical with a circumference between 68 and 70 cm, "weigh" between 410 and 450 g at the start of the match and be made of leather (or similar). In their use of the term "weigh" FIFA are technically incorrect, as are almost all organisations. They mean that the football should have *mass* between 410 and 450 g.

Floating football

If a perfectly spherical FIFA football of circumference 69cm and mass 430g is placed in water, to what depth, h (in cm), does it sink?

Volume of water displaced, ignoring surface tension:

$$\frac{\pi}{3}\left(3rh^2 - h^3\right)$$



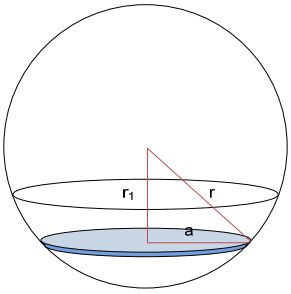
$$r = \frac{69}{2\pi} = 10.9817 \text{ cm}$$

Aside

Calculation of volume of water displaced:

$$a^2 = r^2 - r_1^2$$

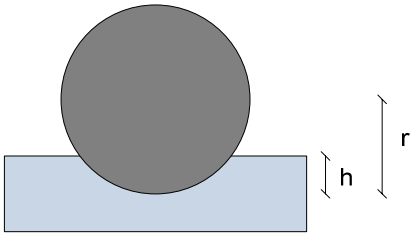
$$Vol = \int_{r-h}^{r} \pi a^2 dr_1 = \int_{r-h}^{r} \pi\left(r^2 - r_1^2\right)dr_1 = \frac{\pi}{3}\left(3rh^2 - h^3\right)$$



Floating football

The density of water is 1 g/cm^3. By Archimedes' Principle:

$$\frac{\pi}{3}\left(32.9451h^2 - h^3\right) = 430$$



$$f(h) = h^3 - 32.9451h^2 + 410.6198 = 0$$

$$h_{n+1} = h_n - \frac{\left(h_n{}^3 - 32.9451h_n{}^2 + 410.6198\right)}{3h_n{}^2 - 65.8902h_n}$$

Floating football

$$f(h) = h^3 - 32.9451h^2 + 410.6198$$

```
>> h = 1;        initialise h
>> h = h – (((h^3) – (32.9451*(h^2)) + 410.6198)/((3*(h^2)) – (32.9451*2*h)))        update h using old h
```

Note that from time to time I will permit Matlab commands to run over more than one line. This is for convenience in presenting the material, but will not work in Matlab itself. You *can* produce Matlab commands over more than one line, but you must use the **…** command.

Matlab polynomials

There is a more convenient command in Matlab for making evaluations of this kind, i.e. evaluations involving polynomials. Firstly we must learn how to create a polynomial in Matlab. For example Matlab creates and stores the polynomial:

$$f(h) = h^3 - 32.9451h^2 + 410.6198$$

as the vector of its coefficients *in descending order*:

$$f = \begin{bmatrix} 1 & -32.9451 & 0 & 410.6198 \end{bmatrix}$$

Note that *all* coefficients must appear, even those which are 0, and that terms are seperated by space.

Matlab polynomials

Now having created the Matlab polynomial we may wish to evaluate it at some particular value (or set of values) of its argument. The Matlab command to achieve this is **polyval**. For example, to evaluate the polynomial

$$f(h) = h^3 - 32.9451h^2 + 410.6198$$

at $h = 2$ use the following commands:

```
>> f = [1  -32.9451  0  410.6198];    create polynomial f
>> polyval(f,2)          evaluate f(h) at h = 2
```

Matlab knows that vector f is actually a polynomial, not a vector, because we are using it with a command which requires and therefore assumes that it is a polynomial.

polyval

$$f(h) = h^3 - 32.9451h^2 + 410.6198$$

$$f'(h) = 3h^2 - 2(32.9451)h$$

```
>> h = 1;        initialise h
>> f = [1  -32.9451  0  410.6198];    initialise f
>> fdash = [3*1  -2*32.9451  0];      initialise fdash
>> h = h – polyval(f,h)/polyval(fdash,h)      update h using old h
```

The initialisation requires considerably more effort, but the iteration becomes easier, even in this simple case.

Floating football

$$f(h) = h^3 - 32.9451h^2 + 410.6198$$

$h_0 = 1$

$h_1 = 7.0212$

$h_2 = 4.2654$

$h_3 = 3.7745$

$h_4 = 3.7504$

$h_5 = 3.7503$

Simplified analysis suggests that the football will sink to a depth of about 3.75 cm

Converges to four decimal places in five iterations.

Application of Newton-Raphson Method: spherical water tank

A spherical water tank of circumference 20 m is to hold the emergency water supply for a building. To what depth must the tank be filled to ensure an emergency supply of 40,000 litres of water? One litre is 1,000 cm^3, i.e. 10cm x 10cm x 10cm. Accordingly 40 m^3 of water is required. The radius of the tank is $r = 20/2\pi = 3.1831$ m. The volume of the tank is more than twice the required volume, being: $\frac{4\pi r^3}{3} = 135.0951 \text{ m}^3$

Water tank

The previous formula applies, giving the equation:

$$\frac{\pi}{3}\left(3rh^2 - h^3\right) = 40$$

$$f(h) = h^3 - 9.5493h^2 + 38.1972$$

$h_0 = 3$

$h_1 = 2.3152$

$h_2 = 2.2946$

Converges to four decimal places in two iterations.

Application of Newton-Raphson Method: air density at altitude

The density of air decreases with altitude h (km). An approximate formula, if air is dry, is given by:

$$\rho = \frac{Mp_0}{RT_0}\left(1-\frac{Lh}{T_0}\right)^{\left(\frac{gM}{RL}-1\right)} = \rho_0\left(1-\frac{Lh}{T_0}\right)^{\left(\frac{gM}{RL}-1\right)}$$

$M = 0.0289644$ kg/mol molar mass of dry air

$R = 8.31447$ J/mol·K ideal gas constant

$L = 6.5$ K/km temperature lapse rate

$T_0 = 288.15$ K $= 15^{\circ}$C sea-level standard temperature

$p_0 = 101{,}325$ Pa sea-level standard pressure

Air density at altitude

At what altitude is the air density half of the value at sea-level?

$$\tfrac{1}{2} = \left(1-\frac{h}{44.331}\right)^{4.2558}$$

$h_0 = 0$

$h_1 = 5.2083$

$h_2 = 6.5773$

$h_3 = 6.6626$

$h_4 = 6.6629$

$$f(h) = \left(1-\frac{h}{44.331}\right)^{4.2558} - \tfrac{1}{2}$$

So answer is about 6.7 km

Converges to four decimal places in four iterations.

Gateway Arch, St Louis

The gateway arch in St Louis is a weighted, inverted catenary.

$$y = (A + f_c) - A\cosh\left(\frac{Cx}{L}\right) \quad \text{for} \quad 0 \le x \le L$$

where L is half the width (of the centroid) at the base. Clearly: $y(0) = (A + f_c) - A\cosh(0) = f_c$

so that f_c is the maximum height (of the centroid). By definition of L: $y(L) = (A + f_c) - A\cosh(C) = 0$

giving: $$C = \cosh^{-1}\left(\frac{A + f_c}{A}\right)$$

Gateway Arch, St Louis

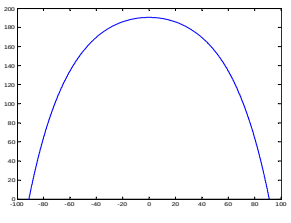
$$A = \frac{f_c}{Q_b/Q_t - 1}$$

$f_c = 190.5282\ \text{m}$

$Q_b = 117.31\ \text{m}^2$

$Q_t = 11.626\ \text{m}^2$

where Q_b is the (maximal) cross sectional area at base and Q_t is the (maximal) cross sectional area at top.



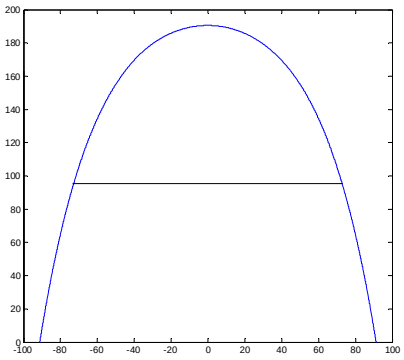
Gateway Arch, St Louis

What is the span when the arch is at half the maximum height?

$$\frac{f_c}{2} = (A + f_c) - A\cosh\left(\frac{Cx}{L}\right)$$

$$0 = 5.5452 - \cosh(0.0329x)$$

Span = $2x$.



$$x_{n+1} = x_n - \left(\frac{5.5452 - \cosh(0.0329x_n)}{-0.0329\sinh(0.0329x_n)}\right)$$

Section: 1.2 Iteration

Gateway Arch, St Louis

$$x_{n+1} = x_n - \left(\frac{5.5452 - \cosh(0.0329x_n)}{-0.0329\sinh(0.0329x_n)}\right)$$

$x_0 = 60$

$x_1 = 76.1525$

$x_2 = 73.0550$

$x_3 = 72.8834$

$x_4 = 72.8829$

Span = $2x$ = 145.7659 m.

Application of Newton-Raphson Method: Free fall

A parachutist free-falls head down from 13,000 ft (3,962.4 m). The parachutist, gear and parachute have a combined mass of 85 kg, drag coefficient of 0.95 and manages to present a cross-sectional area of 0.31 m^2 in this fall position. Terminal velocity occurs when forces due to drag and gravity cancel, using an air density of 1.112 kg/m^3 (dry air at 1,000 m)

$$mg = \frac{\rho C_d A v_T^2}{2} \qquad v_T = \sqrt{\frac{2mg}{\rho C_d A}} = 71.35 \text{ m/s} = 256.86 \text{ km/hr}$$

Free fall

Making the simplifying assumption that the air density is constant at 1.112 kg/m^3 throughout the free fall how long does it take for the parachutist to reach 250 km/hr? Newton's law, the drag equation and $v(0) = 0$ give:

$$m\dot{v} = mg - \left(\frac{\rho C_d A}{2}\right)v^2 \qquad \frac{dv}{\left(1 - \frac{v^2}{v_T^2}\right)} = gdt$$

This differential equation can be solved:

$$v = v_T \left(\frac{\exp\left(\frac{2gt}{v_T}\right) - 1}{\exp\left(\frac{2gt}{v_T}\right) + 1} \right)$$

Free fall

$$\frac{v}{v_T} = \left(\frac{\exp\left(\frac{2gt}{v_T}\right) - 1}{\exp\left(\frac{2gt}{v_T}\right) + 1} \right) = \frac{250}{256.86} = 0.9733$$

Alternatively:

$$x = \frac{gt}{v_T} \quad , \quad \left(\frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} \right) = \tanh(x) = 0.9733$$

$$f(x) = 0.9733 - \tanh(x) \qquad f'(x) = -\text{sech}^2(x)$$

$x_0 = 2$

$x_1 = 2.1312$

$x_3 = 2.1510$

$x_4 = 2.1514$

t = 15.65 s

Free fall

Again making the simplifying assumption that the air density is constant at 1.112 kg/m^3 throughout the free fall how long does it take for the parachutist to free fall 2 km? As above

$$v = v_T \tanh\left(\frac{gt}{v_T}\right)$$

$$h = \int_0^t v\,dt = \int_0^t v_T \tanh\left(\frac{gt}{v_T}\right) dt = \frac{v_T^2}{g} \ln\left(\cosh\left(\frac{gt}{v_T}\right)\right)$$

$$\frac{v_T^2}{g} \ln\left(\cosh\left(\frac{gt}{v_T}\right)\right) = 2{,}000$$

Free fall

$$\ln\left(\cosh\left(\frac{gt}{v_T}\right)\right) = \frac{2{,}000g}{v_T^2} = 3.8527$$

Alternatively: $x = \dfrac{gt}{v_T}$, $\ln(\cosh(x)) = 3.8527$

$f(x) = 3.8527 - \ln(\cosh(x))$ $\quad f'(x) = -\tanh(x)$

$x_0 = 1$

$x_1 = 5.4892$

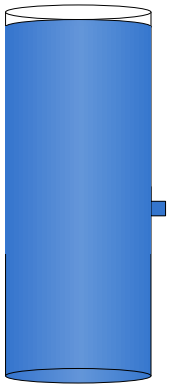
$x_3 = 4.5458$

$x_4 = 4.5457$

$t = 33.07$ s

Application of Newton-Raphson Method: spouting cylinder

A spouting cylinder, i.e. having a circular cross section, of height 1m and circumference 30cm is filled with water up to 4cm from the top. An orifice with circular cross section of circumference 3cm is located in the side of the cylinder half-way down.



Spouting cylinder

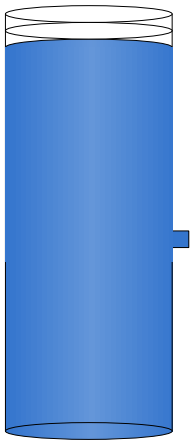
For an incompressible fluid:

$$\frac{1}{2}\Delta m\, v^2 = \Delta m\, gh$$

$v = \sqrt{2gh}$ Torricelli's equation

Due to viscosity, rotation, constriction of jet, *etc* this law fails to hold. Rather, for a circular orifice it is modified to

$v = \alpha\sqrt{gh}$ $\alpha = 0.84$ for water



Spouting cylinder

The water level is governed by the following equation where r is the orifice radius and R the cylinder radius

$$\dot{h} = -\frac{\alpha r^2\sqrt{g}}{R^2}\sqrt{h} = -0.0263\sqrt{h}$$

This differential equation can be solved:

$$\frac{dh}{\sqrt{h}} = -0.0263\,dt$$

$$\frac{1}{2}\sqrt{h}\Big|_{0.96}^{h} = -0.0263t = \frac{1}{2}\sqrt{h} - \frac{1}{2}\sqrt{0.96}$$

Spouting cylinder

$$h = 0.96 - 0.1031t + 0.0028t^2$$

How long does it take the water level to drop by 20 cm, i.e. at what time is $h = 0.76$ m?

$$0.96 - 0.1031t + 0.0028t^2 = 0.76$$

$$f(t) = 0.26 - 0.1031t + 0.0028t^2 = 0$$

$t_0 = 0$

$t_1 = 2.5218$

$t_2 = 2.7220$

$t_3 = 2.7232$

$t = 2.7232$ s

Order of Iteration Method

The previous examples suggest that the Newton-Raphson method converges much faster than the Regula Falsi method. Why?

Suppose Regula Falsi behaves as above, i.e. upper bound b is unchanging. Both methods take the form:

$$x_{n+1} = x_n - F(x_n) \qquad x_n = a_n \text{ for Regula Falsi}$$

$$F(x) = \frac{f(x)}{f'(x)} \quad \text{Newton - Raphson}$$

$$F(x) = \frac{f(x)(b-x)}{f(b) - f(x)} \quad \text{Regula Falsi with } b \text{ constant}$$

Order of Iteration Method

In general a recursion of the form

$$x_{n+1} = x_n - F(x_n)$$

may converge to a *fixed point*, x^*, satisfying

$$x^* = x^* - F(x^*) \qquad \text{i.e.} \quad F(x^*) = 0$$

i.e. a root of $F(x)$. Let the deviation of x_n from the root x^* (the error) be denoted $e_n = x_n - x^*$. It is simple to derive

$$e_{n+1} = e_n - (F(x^* + e_n) - F(x^*))$$

Order of Iteration Method

$$e_{n+1} = e_n - (F(x^* + e_n) - F(x^*))$$

Applying Taylor series gives:

$$e_{n+1} = e_n - F'(x^*)e_n - F''(\xi)e_n^2$$

For Regula Falsi with b constant

$$F(x) = \frac{f(x)(b-x)}{f(b) - f(x)} \qquad \text{so} \quad F(x^*) = 0 \quad \text{if} \quad f(x^*) = 0.$$

$$F'(x^*) = \frac{f'(x^*)(b-x^*)}{f(b)} \qquad e_{n+1} \cong (1 - F'(x^*))e_n$$

Regula Falsi with b constant is a *first order* iterative method.

Order of Iteration Method

$$e_{n+1} = e_n - F'(x^*)e_n - F''(\xi)e_n^2$$

For Newton-Raphson

$$F(x) = \frac{f(x)}{f'(x)} \qquad \text{so} \quad F(x^*) = 0 \quad \text{if} \quad f(x^*) = 0.$$

$$F'(x^*) = \frac{f'(x^*)}{f'(x^*)} - \frac{f(x^*)f''(x^*)}{(f'(x^*))^2} = 1$$

so $e_{n+1} = -F''(\xi)e_n^2$

Newton-Raphson is a *second order* iterative method.

Difficulties with Newton-Raphson Method

No more than any method the Newton-Raphson method is not a panacea. The initial guess must be made by some other procedure, it does not form part of the algorithm. Moreover the initial guess may have a significant influence on whether the algorithm converges (there may be multiple solutions), on the rate of convergence and on whether the algorithm converges at all. There are also cases where, essentially irrespective of the initial guess the algorithm fails to converge and in these cases you will have to apply an alternative method such as Regula Falsi.

Difficulties with Newton-Raphson Method: Failure to converge

Consider

$$f(x) = x^{1/3} \qquad f'(x) = \frac{1}{3}x^{-2/3}$$

$$x_{n+1} = x_n - \frac{x_n^{1/3}}{\frac{1}{3}x_n^{-2/3}} = x_n - 3x_n = -2x_n$$

What has gone wrong here?

$$x^* = 0 \qquad f'(x^*) = 1/0$$

so f is not differentiable at the root.

Section: 1.3 Iteration

Difficulties with Newton-Raphson Method: Failure to converge

Consider $f(x) = x^3 - ax + b = x^3 - 2x + 2$

$$x_{n+1} = x_n - \frac{x_n^3 - 2x_n + 2}{3x_n^2 - 2} = \frac{2x_n^3 - 2}{3x_n^2 - 2} = G(x_n)$$

The selected values $a = 2$ and $b = 2$ are chosen to yield

$$G(0) = \frac{-2}{-2} = 1 \quad \text{and} \quad G(1) = \frac{0}{1} = 0$$

so that the sequence generated by the algorithm for initial guess $x_0 = 0$ is periodic $\{0,1,0,1, \ldots\}$

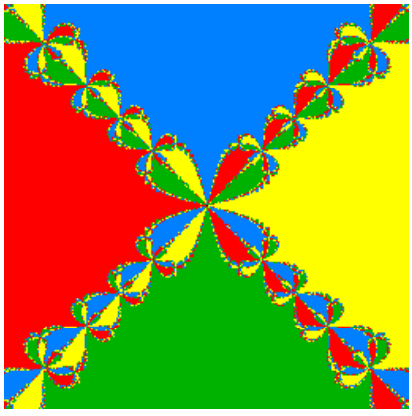
Difficulties with Newton-Raphson Method

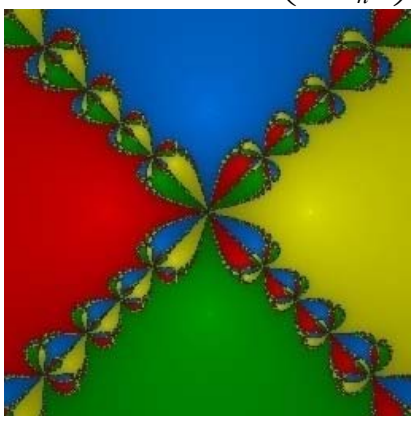
In general the sequence generated by the Newton-Raphson algorithm (or for that matter any iterative scheme) may be highly complicated, including *chaotic* sequences. Even if the sequence converges to a fixed point, which fixed point it converges to will depend upon the initial point and this dependency can also be surprisingly complicated, particularly if the iteration involves complex numbers.

Subtleties of Newton-Raphson Method

$f(z) = z^4 - 1 \qquad z^* = \pm 1 \ , \ \pm i$

$$z_{n+1} = z_n - \left(\frac{z_n^4 - 1}{4 z_n^3} \right)$$





e.g. point coloured blue if sequence starting at this point converges to i. Shading reflects rate of convergence.

Subtleties of Newton-Raphson Method: a new geometry

This type of highly complicated geometrical object is called a *fractal*. There is an entire geometry of fractals and it has been argued, in particular by Mandelbrot, that fractals offer a far more accurate description of the objects found in nature.

Other Methods for Solving Nonlinear Equations

There are many other methods available for solving nonlinear equations. Examples are *secant method*, *Muller's method* (an extension of Regula Falsi), *Halley's method* (an extension of Newton-Raphson), *Householder's methods* (an extension of both Newton-Raphson and Halley) and *Lehmer-Schur method* (particularly suited to finding complex roots). For the special case of polynomial equations specialised methods are available such as *Bairstow's method* and the *QD-algorithm*.

Newton-Raphson Method for systems of equations

Not all problems involve or can be reduced to a single equation in a single unknown. The ideal diode circuit for example involved two equations in two unknowns, the current and the voltage, although granted on that occasion it could be reduced. It transpires that there exists a generalisation of the Newton-Raphson algorithm which is applicable to the problem of solving *systems of equations*.

General Newton-Raphson Method

In general a vector recursion of the form

$$\mathrm{x}_{n+1} = \mathrm{x}_n - F(\mathrm{x}_n)$$

x_n a vector of order m, $F(\mathrm{x})$ a vector function, may converge to a *fixed point*, x*, satisfying

$$\mathrm{x}^* = \mathrm{x}^* - F(\mathrm{x}^*) \qquad \text{i.e.} \quad F(\mathrm{x}^*) = \vec{0}$$

i.e. a solution of $F(\mathrm{x}) = \vec{0}$, a vector equation or system of equations. Let the deviation of x_n from the root x* (the error) be denoted $\mathrm{e}_n = \mathrm{x}_n - \mathrm{x}^*$. Again it is simple to derive

$$\mathrm{e}_{n+1} = \mathrm{e}_n - \left(F(\mathrm{x}^* + \mathrm{e}_n) - F(\mathrm{x}^*)\right)$$

General Newton-Raphson Method

Applying Taylor series for a function of several variables

$$e_{n+1} = e_n - DF(\mathrm{x}^*)e_n + h.o.t.$$

where *h.o.t.* denotes "higher order terms" and where $DF(\mathrm{x})$ denotes the *Jacobian matrix* of vector function $F(\mathrm{x})$. The iteration becomes second order if

$$DF(\mathrm{x}^*) = I$$

where I denotes the identity matrix of suitable order, i.e. of order equal to the order, m, of vector x.

General Newton-Raphson Method

Now suppose the system of equations to be solved is expressible in vector form as $f(\mathrm{x}) = \vec{0}$ and suppose that the recursion

$$\mathrm{x}_{n+1} = \mathrm{x}_n - F(\mathrm{x}_n) = \mathrm{x}_n - H(\mathrm{x}_n)f(\mathrm{x}_n)$$

is to be employed in an attempt to solve the system. Of course if $f(\mathrm{x}^*) = \vec{0}$, i.e. x* is a solution of the system, then $F(\mathrm{x}^*) = H(\mathrm{x}^*)f(\mathrm{x}^*) = \vec{0}$ provided $H(\mathrm{x}^*)$ is finite, i.e. x* is a fixed point of the recursion and it is for this reason that we express $F(\mathrm{x})$ in the factored form $H(\mathrm{x})f(\mathrm{x})$.

General Newton-Raphson Method

The coup-de-gras is delivered by the product rule of differentiation.

$$DF(\mathrm{x}^*) = D(H(\mathrm{x})f(\mathrm{x}))\big|_{\mathrm{x}=\mathrm{x}^*} = (\cdot)f(\mathrm{x}^*) + H(\mathrm{x}^*)Df(\mathrm{x}^*)$$

where the unspecified term represents a tangle of partial derivatives which thankfully we do not need to know since it is multiplied by $f(\mathrm{x}^*) = \vec{0}$, i.e.

$$DF(\mathrm{x}^*) = H(\mathrm{x}^*)Df(\mathrm{x}^*)$$

So the recursion is second order if $I = H(\mathrm{x}^*)Df(\mathrm{x}^*)$

$$\text{i.e. if } \quad H(\mathrm{x}^*) = (Df(\mathrm{x}^*))^{-1}$$

General Newton-Raphson Method

The simplest way to achieve this result is by taking

$$H(\mathrm{x}) = (Df(\mathrm{x}))^{-1}$$

so that the recursion or iteration becomes

$$\mathrm{x}_{n+1} = \mathrm{x}_n - (Df(\mathrm{x}_n))^{-1} f(\mathrm{x}_n)$$

which is the general Newton-Raphson method and which comprises an essentially perfect generalisation to vector functions (a.k.a. systems of equations) of the Newton-Raphson method with the inverse of the Jacobian matrix replacing the reciprocal of the derivative.

Example: Newton-Raphson for system of equations

$$f(x,y)=\begin{bmatrix} f_1(x,y) \\ f_2(x,y) \end{bmatrix}=\begin{bmatrix} x^2+y^2-2 \\ x^2-y^2-1 \end{bmatrix}=\begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$Df(x,y)=\begin{bmatrix} \dfrac{\partial f_1(x,y)}{\partial x} & \dfrac{\partial f_1(x,y)}{\partial y} \\ \dfrac{\partial f_2(x,y)}{\partial x} & \dfrac{\partial f_2(x,y)}{\partial y} \end{bmatrix}=\begin{bmatrix} 2x & 2y \\ 2x & -2y \end{bmatrix}$$

Example: Newton-Raphson for system of equations

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix}=\begin{bmatrix} x_n \\ y_n \end{bmatrix}-\begin{bmatrix} 2x_n & 2y_n \\ 2x_n & -2y_n \end{bmatrix}^{-1}\begin{bmatrix} x_n^2+y_n^2-2 \\ x_n^2-y_n^2-1 \end{bmatrix}$$

$(x_0,y_0)=(1,1)$

$(x_1,y_1)=(1.2500,0.7500)$

$(x_2,y_2)=(1.2250,0.7083)$

$(x_3,y_3)=(1.2247,0.7071)$

Actual four answers

$$(x,y)=\left(\pm\sqrt{\frac{3}{2}},\pm\frac{1}{\sqrt{2}}\right)$$

Example: Newton-Raphson for system of equations using Matlab

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} x_n \\ y_n \end{bmatrix} - \begin{bmatrix} 2x_n & 2y_n \\ 2x_n & -2y_n \end{bmatrix}^{-1} \begin{bmatrix} x_n^2 + y_n^2 - 2 \\ x_n^2 - y_n^2 - 1 \end{bmatrix}$$

```
>> v = [1;1];   initialise vector v = [x y]'
>> f = [(v(1)^2)+(v(2)^2) – 2; (v(1)^2) – (v(2)^2) – 1]
>> Df = [2*v(1)   2*v(2) ; 2*v(1)   – 2*v(2)];
>> v = v – (inv(Df)*f)          update v using old v
```

Note, letting *f* print to screen permits monitoring of convergence or otherwise.

Nonlinear electronic circuit: reprise

$$f(v,i) = \begin{bmatrix} f_1(v,i) \\ f_2(v,i) \end{bmatrix} = \begin{bmatrix} 10 - v - 1000\,i \\ i - 50\times10^{-6}\left(\exp(20v) - 1\right) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$Df(v,i) = \begin{bmatrix} \dfrac{\partial f_1(v,i)}{\partial v} & \dfrac{\partial f_1(v,i)}{\partial i} \\ \dfrac{\partial f_2(v,i)}{\partial v} & \dfrac{\partial f_2(v,i)}{\partial i} \end{bmatrix} = \begin{bmatrix} -1 & -1000 \\ -20\left(50\times10^{-6}\right)\exp(20v) & 1 \end{bmatrix}$$

$$\begin{bmatrix} v_{n+1} \\ i_{n+1} \end{bmatrix} = \begin{bmatrix} v_n \\ i_n \end{bmatrix} - \begin{bmatrix} -1 & -1000 \\ -20\left(50\times10^{-6}\right)\exp(20v_n) & 1 \end{bmatrix}^{-1} \begin{bmatrix} 10 - v_n - 1000 i_n \\ i_n - 50\times10^{-6}\left(\exp(20v_n) - 1\right) \end{bmatrix}$$

Nonlinear electronic circuit: reprise

$$\begin{bmatrix} v_{n+1} \\ i_{n+1} \end{bmatrix} = \begin{bmatrix} v_n \\ i_n \end{bmatrix} - \begin{bmatrix} -1 & -1000 \\ -20\left(50\times10^{-6}\right)\exp(20v_n) & 1 \end{bmatrix}^{-1} \begin{bmatrix} 10 - v_n - 1000 i_n \\ i_n - 50\times10^{-6}\left(\exp(20v_n)-1\right) \end{bmatrix}$$

$(v_0, i_0) = (0.2 , 0.0098)$

$(v_1, i_1) = (0.3281 , 0.0097)$

$(v_2, i_2) = (0.2919 , 0.0097)$

$(v_3, i_3) = (0.2704 , 0.0097)$

$(v_4, i_4) = (0.2642 , 0.0097)$

$(v_5, i_5) = (0.2638 , 0.0097)$

Nonlinear electronic circuit: reprise

In this example the ratio of the largest parameter, 1000, to the smallest, $50\text{x}10^{-6}$, is rather large and this can cause numerical difficulties. It is commonly the case that one can mitigate these difficulties by scaling. In this case suppose the current is measured in mA rather than A, i.e. replace $1000i$ by $\hat{\imath}$. The system of equations becomes

$$f\left(v,\hat{i}\right) = \begin{bmatrix} f_1\left(v,\hat{i}\right) \\ f_2\left(v,\hat{i}\right) \end{bmatrix} = \begin{bmatrix} 10 - v - \hat{i} \\ \hat{i} - 50\times10^{-3}\left(\exp(20v)-1\right) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Nonlinear electronic circuit: reprise

$$\begin{bmatrix} v_{n+1} \\ \hat{i}_{n+1} \end{bmatrix} = \begin{bmatrix} v_n \\ \hat{i}_n \end{bmatrix} - \begin{bmatrix} -1 & -1 \\ -20\left(50\times10^{-3}\right)\exp(20v_n) & 1 \end{bmatrix}^{-1} \begin{bmatrix} 10 - v_n - \hat{i}_n \\ \hat{i}_n - 50\times10^{-3}\left(\exp(20v_n)-1\right) \end{bmatrix}$$

$(v_0\,,\,\hat{i}_0) = (0.2\,,\,9.8)$

$(v_1\,,\,\hat{i}_1) = (0.3281\,,\,9.6719)$

$(v_2\,,\,\hat{i}_2) = (0.2919\,,\,9.7081)$

$(v_3\,,\,\hat{i}_3) = (0.2704\,,\,9.7296)$

$(v_4\,,\,\hat{i}_4) = (0.2642\,,\,9.7358)$

$(v_5\,,\,\hat{i}_5) = (0.2638\,,\,9.7362)$

Section 1 - Conclusion

- In practice we must commonly accept approximate numerical solutions to problems as the only readily available solutions.
- The key concept in acquiring approximate solutions is iteration whereby the approximation improves with each iterate. If you only know one iterative method for solving systems of nonlinear equations it should be the Newton-Raphson method.

Overview of Topics

Solution of equations by iteration.

Optimisation.

Solution of system of linear equations.

Solution of ordinary differential equations: initial value problems.

Numerical Quadrature.

Solution of ordinary differential equations: boundary value problems.

Solution of partial differential equations.

Section 2 - Optimisation

- Unconstrained optimisation.
- Constrained optimisation.

Workload: 3 lectures + Component of Minor Project + Autonomous learning

Local extrema

Although having an ancient history optimisation commences in earnest with the work of Pierre de Fermat on determining the local maxima (and local minima) of functions. He largely establishes the following theorem:

If $f(x)$ is a function which is differentiable on the interval (a,b) then the point c in interval (a,b) is a local extremum (i.e. a local maximum or a local minimum) of $f(x)$ only if it is a *stationary point* of $f(x)$, i.e. $f'(c) = 0$.

Local extrema

It is important to observe that Fermat's theorem transforms the problem of finding local extrema into the problem of solving equations, a problem which we have already considered from the numerical viewpoint. Fermat's theorem provides *necessary* conditions only for a point to be a local extremum. In practice the theorem provides a set of *potential* local maxima (or local minima) but further testing is required to determine which if any of the points in this set are actually local maxima (or local minima).

Fermat's Theorem: Example

Two real numbers sum to 20, what is the largest possible value of their product?

$$x + y = 20 \quad , \quad xy = x(20 - x) = 20x - x^2$$

$$f(x) = 20x - x^2$$

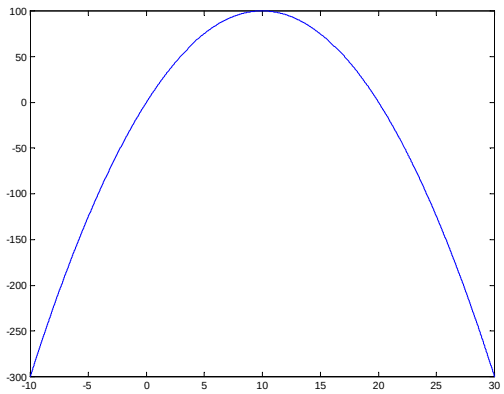
$$f'(x) = 20 - 2x = 0, \quad \text{i.e.} \quad x = 10$$

Fermat's theorem establishes that the only potential local maximum is $x = y = 10$, giving the value $xy = 100$.

Fermat's Theorem: Example

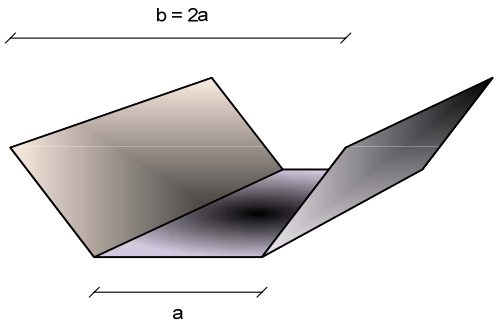
Fermat's theorem does not establish that $x = y = 10$ is a local, indeed global maximum, although a plot of $f(x)$ vs x does make this seem likely.

A stationary point c, i.e. $f'(c) = 0$, is a local maximum if $f''(c) > 0$, a local minimum if $f''(c) < 0$ and a *point of inflection* if $f''(c) = 0$.



Application of Fermat's theorem: Trapezoidal gutter

A gutter is formed by bending a single sheet of metal of width W so that the width at the top b is twice that of the base a as shown.



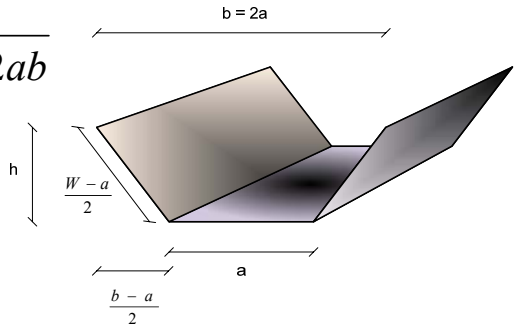
Trapezoidal gutter

What choice of width a for the base of the gutter ensures the greatest capacity?

$$h = \tfrac{1}{2}\sqrt{W^2 - 2aW - b^2 + 2ab}$$

$$= \tfrac{1}{2}\sqrt{W^2 - 2aW}$$

Capacity is greatest when cross sectional area (CSA) is greatest.



$$CSA = \left(a + \frac{(b-a)}{2}\right)h = \frac{(a+b)h}{2} = \frac{3ah}{2} = \frac{3a\sqrt{W^2 - 2aW}}{4}$$

Trapezoidal Gutter

$$f'(a) = \frac{3}{4}\sqrt{W^2 - 2aW} + \frac{3a}{8}\frac{(-2W)}{\sqrt{W^2 - 2aW}} = 0$$

$$W^2 - 2aW - aW = 0$$

$$a = \frac{W}{3}$$

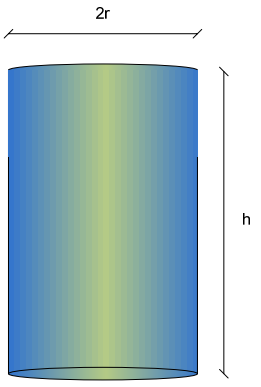
Application of Fermat's theorem: oil can

An oil can is to contain 1 litre of oil. The can is cylindrical in shape. What dimensions of the can will require the least amount of material in its manufacture?

$$Vol = \pi r^2 h = 1{,}000 \text{ cm}^3$$

$$Area = 2\pi r^2 + 2\pi rh$$

$$h = \frac{1{,}000}{\pi r^2} \qquad Area = 2\pi r^2 + \frac{2{,}000}{r}$$



Oil can

$$f'(r) = 4\pi r - \frac{2{,}000}{r^2} = 0$$

Key idea: apply Newton-Raphson to solution of resulting equation.

$$r_{n+1} = r_n - \frac{\left(4\pi r_n - \frac{2{,}000}{r_n^2}\right)}{\left(4\pi + \frac{4{,}000}{r_n^3}\right)}$$

$r_0 = 1$

$r_1 = 1.4953$

$r_2 = 2.2196$

$r_3 = 3.2189$

$r_4 = 4.3704$

$r_5 = 5.1936$

$r_6 = 5.4096$

$r_7 = 5.4192$

$r_8 = 5.4193$

Oil can

$r = 5.4193$ cm, $h = 10.8385$ cm $= 2r$

So the height of the can should equal its diameter.

This method of optimisation, namely apply the Newton-Raphson method to solve Fermat's equation $f'(x) = 0$ is generally called *Newton's method*.

Newton's Method

Given x_n let

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$$

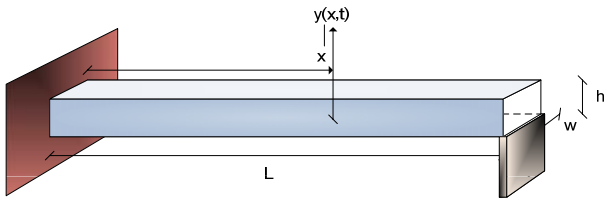
If $f'(x_{n+1}) = 0$ stationary point equals x_{n+1}. Terminate.

If $f'(x_{n+1}) \neq 0$ continue until termination condition.

Terminate if $|f'(x_{n+1})|$ below prespecified tolerance or number of steps taken reaches prespecified maximum.

Application of Newton's Method: Cantilever beam

A cantilever beam is fixed to a wall at one end and simply supported at the other. The beam is of length L.



Assuming both flexural stiffness and mass per unit length to be constant find the point of maximum deflection of the beam.

Application of Newton's Method: Cantilever beam

Assuming flexural stiffness EI to be a constant the deflection is described by the static Euler-Bernoulli beam equation:

$$EI\frac{d^4y}{dx^4} = q(x)$$

where $q(x)$ is the distributed force per unit length. In this case, apart from at the end points, the only force is the weight of the beam and this is constant if the mass per unit length is constant.

Cantilever beam

Assuming the force per unit length, q, also to be a constant the general solution of this equation is

$$y = \frac{q}{24EI}x^4 + c_1x^3 + c_2x^2 + c_3x + c_4$$

As the beam is fixed at $x = 0$ we take for boundary conditions: $y(0) = 0$ and $y'(0) = 0$.

As the beam is simply supported at $x = L$ we take for boundary conditions: $y(L) = 0$ and $y''(L) = 0$ which ensures that there is no bending moment at $x = L$, i.e. that the beam does not rotate.

Cantilever beam

$$y = \frac{q}{24EI}x^4 + c_1x^3 + c_2x^2 + c_3x + c_4$$

$$y(0) = 0 \quad , \quad y'(0) = 0$$
$$y(L) = 0 \quad , \quad y''(L) = 0$$

$$c_4 = 0 \quad , \quad c_3 = 0$$

$$\frac{q}{24EI}L^4 + c_1L^3 + c_2L^2 = 0 \quad , \quad \frac{q}{2EI}L^2 + 6c_1L + 2c_2 = 0$$

$$c_1 = \frac{-5qL}{48EI} \quad , \quad c_2 = \frac{3qL^2}{48EI}$$

Cantilever beam

$$y = \frac{q}{24EI}x^4 - \frac{5qL}{48EI}x^3 + \frac{3qL^2}{48EI}x^2$$

Fermat's theorem gives maximum deflection at a point x such that

$$\frac{dy}{dx} = \frac{q}{48EI}\left(8x^3 - 15Lx^2 + 6L^2x\right) = 0$$

$$8\left(\frac{x}{L}\right)^3 - 15\left(\frac{x}{L}\right)^2 + 6\left(\frac{x}{L}\right) = 0$$

Let $z = x/L$ $\qquad 8z^3 - 15z^2 + 6z = 0$

Cantilever beam

$$8z^3 - 15z^2 + 6z = 0 \qquad z = x/L$$

$z_0 = 1.2$
$z_1 = 1.3263$
$z_2 = 1.2983$
$z_3 = 1.2965$

$z_0 = 0.8$
$z_1 = 0.5333$
$z_2 = 0.5796$
$z_3 = 0.5785$

$z_0 = 0.2$
$z_1 = -0.4917$
$z_2 = -0.2082$
$z_3 = -0.0598$
$z_4 = -0.0072$
$z_5 = -0.0001$
$z_6 = -0.0000$

Cantilever beam

$$8z^3 - 15z^2 + 6z = 0 \qquad z = x/L$$

The Newton-Raphson method offers three roots (perfectly reasonably since the equation is cubic). The root $z = 1.2965$ corresponds to $x = 1.2965L > L$ is a mathematical artefact lacking in physical meaning. The obvious root $z = 0$ corresponds to $x = 0$ and is clearly not a maximum of deflection since there is in fact no deflection here. Accordingly the maximum deflection is at $z = 0.5785$, i.e. $x = 0.5785L$.

Cantilever beam

It should be noted that a potential problem arises in the application of the Newton-Raphson method in this case. The iteration is

$$z_{n+1} = z_n - \frac{\left(8z_n^3 - 15z_n^2 + 6z_n\right)}{\left(24z_n^2 - 30z_n + 6\right)}$$

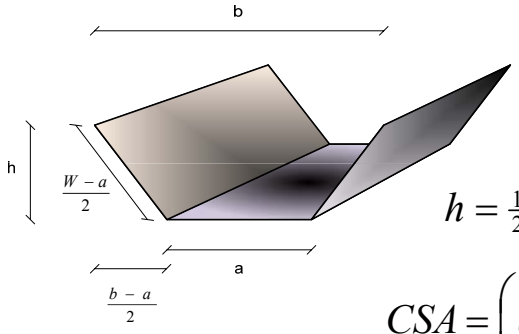
But the denominator polynomial $24z^2$ -$30z$+6 has roots at 1 and 0.25. Accordingly an initial guess of $z_0 = 1$ or 0.25 will result in the algorithm failing to execute with a "divide by zero" warning.

Local extrema

In optimisation the function which is to be either maximized or minimized is called the *objective function*. The quantities which can be adjusted to affect this maximization or minimization are called the *control variables*. The simple examples considered so far where this function is a function of a single control variable are not representative of the optimisation problems which most commonly arise in practice. One will far more commonly be faced with an objective function which is a function of several (m) control variables.

Capacity of trapezoidal gutter depending on two variables

A gutter is formed by bending a single sheet of metal of width W so that the width at the top b is greater than that of the base a as shown.



$$h = \tfrac{1}{2}\sqrt{W^2 - 2aW - b^2 + 2ab}$$

$$CSA = \left(a + \frac{(b-a)}{2}\right)h = \frac{(a+b)h}{2}$$

Local extrema

It transpires that there exists a counterpart to Fermat's theorem for functions of several (m) variables and indeed this counterpart could hardly be a more perfect generalisation. For a function of several variables $f(\mathrm{x}) = f(x_1, \ldots, x_m)$ the gradient of f, denoted grad(f), is the vector function

$$grad(f) = \nabla f(\mathrm{x}) = \begin{bmatrix} \dfrac{\partial f(\mathrm{x})}{\partial x_1} \\ \vdots \\ \dfrac{\partial f(\mathrm{x})}{\partial x_m} \end{bmatrix}$$

Local extrema

Fermat's theorem for functions of several variables, says:

If $f(\mathrm{x})$ is a function which is differentiable on the open set U then the vector c in U is a local extremum (i.e. a local maximum or a local minimum) of $f(\mathrm{x})$ only if it is a stationary point of $f(\mathrm{x})$, i.e.

$$\nabla f(\mathrm{c}) = \vec{0}$$

where $\vec{0}$ denotes the zero vector, i.e. the vector whose components are all zero.

Fermat's Theorem: Example

Three real numbers sum to 20, what is the largest possible value of their product?

$$x + y + z = 20 \quad , \quad xyz = xy(20 - x - y) = 20xy - x^2y - xy^2$$

$$f(x, y) = 20xy - x^2y - xy^2$$

$$\begin{bmatrix} \dfrac{\partial f(x,y)}{\partial x} \\ \dfrac{\partial f(x,y)}{\partial y} \end{bmatrix} = \begin{bmatrix} 20y - 2xy - y^2 \\ 20x - x^2 - 2xy \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Fermat's Theorem: Example

$$\begin{bmatrix} 20y - 2xy - y^2 \\ 20x - x^2 - 2xy \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$ a *system* of equations

$$20 - 2x - y = 0 \quad \text{or} \quad y = 0$$
$$20 - x - 2y = 0 \quad \text{or} \quad x = 0$$

$$(x, y) = \left(\tfrac{20}{3}, \tfrac{20}{3}\right),\ (0,20),\ (20,0) \text{ or } (0,0)$$

$$(x, y, z) = \left(\tfrac{20}{3}, \tfrac{20}{3}, \tfrac{20}{3}\right),\ (0,20,0),\ (20,0,0) \text{ or } (0,0,20)$$

$$xyz = \frac{8{,}000}{27},\ 0,\ 0 \text{ or } 0$$

Fermat's Theorem

In general Fermat's theorem applied to the problem of finding local maxima (or local minima) of functions of several variables will lead to the problem of solving a system of equations. Of course we can apply the Newton-Raphson method to solve this system. However, if all of the equations in the system are linear then the Newton-Raphson method is assuredly not the best method available. We will see how to efficiently solve systems of linear equations later in the module.

General trapezoidal gutter

$$CSA = \frac{(a+b)\sqrt{W^2 - 2aW - b^2 + 2ab}}{4}$$

$$\frac{\partial CSA}{\partial a} = \frac{\sqrt{W^2 - 2aW - b^2 + 2ab}}{4} + \frac{(a+b)(-2W+2b)}{8\sqrt{W^2 - 2aW - b^2 + 2ab}} = 0$$

$$\frac{\partial CSA}{\partial b} = \frac{\sqrt{W^2 - 2aW - b^2 + 2ab}}{4} + \frac{(a+b)(-2b+2a)}{8\sqrt{W^2 - 2aW - b^2 + 2ab}} = 0$$

$$W^2 - 2aW - b^2 + 2ab + (a+b)(-W+b) = 0$$

$$W^2 - 2aW - b^2 + 2ab + (a+b)(-b+a) = 0$$

Trapezoidal gutter

$$g(a,b) = \begin{bmatrix} g_1(a,b) \\ g_2(a,b) \end{bmatrix} = \begin{bmatrix} W^2 - 2aW - b^2 + 2ab + (a+b)(-W+b) \\ W^2 - 2aW - b^2 + 2ab + (a+b)(-b+a) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

It is possible to simplify these equations by a scaling. This kind of simplification, often called normalisation, is commonly achievable and well worth considering. Let

$$a = \alpha W \quad , \quad b = \beta W$$

$$W^2 \begin{bmatrix} 1 - 2\alpha - \beta^2 + 2\alpha\beta + (\alpha+\beta)(-1+\beta) \\ 1 - 2\alpha - \beta^2 + 2\alpha\beta + (\alpha+\beta)(-\beta+\alpha) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Trapezoidal gutter

$$g(\alpha,\beta)=\begin{bmatrix} g_1(\alpha,\beta) \\ g_2(\alpha,\beta) \end{bmatrix}=\begin{bmatrix} 1-2\alpha-\beta^2+2\alpha\beta+(\alpha+\beta)(-1+\beta) \\ 1-2\alpha-\beta^2+2\alpha\beta+(\alpha+\beta)(-\beta+\alpha) \end{bmatrix}=\begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$Dg(\alpha,\beta)=\begin{bmatrix} \dfrac{\partial g_1(\alpha,\beta)}{\partial\alpha} & \dfrac{\partial g_1(\alpha,\beta)}{\partial\beta} \\ \dfrac{\partial g_2(\alpha,\beta)}{\partial\alpha} & \dfrac{\partial g_2(\alpha,\beta)}{\partial\beta} \end{bmatrix}=\begin{bmatrix} -3+3\beta & -1+3\alpha \\ -2+2\beta+2\alpha & -4\beta+2\alpha \end{bmatrix}$$

$$\begin{bmatrix} \alpha_{n+1} \\ \beta_{n+1} \end{bmatrix}=\begin{bmatrix} \alpha_n \\ \beta_n \end{bmatrix}-$$

$$\begin{bmatrix} -3+3\beta_n & -1+3\alpha_n \\ -2+2\beta_n+2\alpha_n & -4\beta_n+2\alpha_n \end{bmatrix}^{-1}\begin{bmatrix} 1-3\alpha_n-\beta_n+3\alpha_n\beta_n \\ 1-2\alpha_n+2\alpha_n\beta_n+\alpha_n^2-2\beta_n^2 \end{bmatrix}$$

Section: 2.1 Optimisation

Trapezoidal gutter

$$\begin{bmatrix} \alpha_{n+1} \\ \beta_{n+1} \end{bmatrix}=\begin{bmatrix} \alpha_n \\ \beta_n \end{bmatrix}-\begin{bmatrix} -3+3\beta_n & -1+3\alpha_n \\ -2+2\beta_n+2\alpha_n & -4\beta_n+2\alpha_n \end{bmatrix}^{-1}\begin{bmatrix} 1-3\alpha_n-\beta_n+3\alpha_n\beta_n \\ 1-2\alpha_n+2\alpha_n\beta_n+\alpha_n^2-2\beta_n^2 \end{bmatrix}$$

$(\alpha_0,\beta_0) = (0.5,0.5)$

$(\alpha_1,\beta_1) = (0.4167,0.7500)$

$(\alpha_2,\beta_2) = (0.3029,0.6588)$

$(\alpha_3,\beta_3) = (0.3327,0.6661)$

$(\alpha_4,\beta_4) = (0.3333,0.6667)$

Actually there are four answers with only the first two being physically meaningful:

$$(\alpha,\beta)=\left(\frac{1}{3},\frac{2}{3}\right)\ ,\ (1,1)$$

$$(-1,1)\ ,\ \left(\frac{1}{3},-\frac{1}{3}\right)$$

Trapezoidal gutter

$$\begin{bmatrix}\alpha_{n+1}\\ \beta_{n+1}\end{bmatrix}=\begin{bmatrix}\alpha_n\\ \beta_n\end{bmatrix}-\begin{bmatrix}-3+3\beta_n & -1+3\alpha_n\\ -2+2\beta_n+2\alpha_n & -4\beta_n+2\alpha_n\end{bmatrix}^{-1}\begin{bmatrix}1-3\alpha_n-\beta_n+3\alpha_n\beta_n\\ 1-2\alpha_n+2\alpha_n\beta_n+\alpha_n^2-2\beta_n^2\end{bmatrix}$$

$(\alpha_0,\beta_0) = (0.9,0.9)$

$(\alpha_1,\beta_1) = (1.0326,1.0234)$

$(\alpha_2,\beta_2) = (1.0017,1.0010)$

$(\alpha_3,\beta_3) = (1.0000,1.0000)$

Which solution the procedure converges to depends on the initial values chosen.

Curve Fitting

A major application of optimisation is the problem of *curve fitting*. We may reasonably often have a set of experimental data points showing the variation of some quantity as a second control quantity is varied. We may imagine that there exists some formula relating the two quantities. If we guess the rough shape of that formula then the problem is to select the parameters in the approximate formula which cause it to be the best possible approximation of its kind to the given data. This is best illustrated by example.

Example: Curve Fitting

The specific heat capacity at constant standard pressure on a per unit mass basis of CO_2 is measured as:

Temperature K	C_p at 0.1 MPa kJ/kg K	Temperature K	C_p at 0.1 MPa kJ/kg K
250	0.791	600	1.075
300	0.846	650	1.102
350	0.895	700	1.126
400	0.939	750	1.148
450	0.978	800	1.169
500	1.014	900	1.204
550	1.046	1000	1.234

Example: Curve Fitting

Firstly let us create two vectors in Matlab which contain respectively all of the temperature values and all of the associated specific heat capacity values:

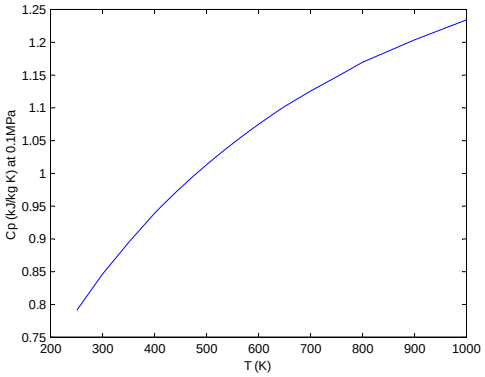
```
>>  T = [[250:50:800] 900 1000];      initialise vector T

>>  Cp =[0.791 0.846 0.895 0.939 0.978 1.014 1.046 1.075 1.102
1.126 1.148 1.169 1.204 1.234];      initialise vector Cp
```

Fundamentally I use the *list method* to generate these vectors, but noting that the first 12 temperatures are all seperated by 50 K allows an easier method of generating them.

Example: Curve Fitting

Clearly the specific heat capacity is not constant and clearly it does not vary linearly with temperature. Simply plotting **Cp** vs **T** with the **plot** command (and changing axis labels appropriately):



Section: 2.2 Optimisation

Example: Curve Fitting

In the event (the rather common case) where we do not have a formula for the tabulated data we may choose to find some approximate formula. Finding a polynomial (usually of relatively low degree) which gives a good approximation to the data is a common enough choice for this formula. In this case, suppose we elect to approximate the data by a cubic polynomial:

$$C_p(T) = a_0 + a_1 T + a_2 T^2 + a_3 T^3$$

How then should we choose the coefficients a_i ?

Example: Curve Fitting

A popular method is to select the coefficients so that the *mean square error* is minimized. Here mean square error refers to the average of the square of the error, the error being the difference between the actual value of C_p at a particular temperature and the value given by the polynomial formula at this same temperature. So, to be clear, the first and second error values for this data set are:

$$0.791 - \left(a_0 + a_1 250 + a_2 250^2 + a_3 250^3\right)$$

$$0.846 - \left(a_0 + a_1 300 + a_2 300^2 + a_3 300^3\right)$$

Example: Curve Fitting

The mean square error is the rather intimidating:

$$\begin{aligned}
&\tfrac{1}{14}\left(0.791 - \left(a_0 + a_1 250 + a_2 250^2 + a_3 250^3\right)\right)^2 + \\
&\tfrac{1}{14}\left(0.846 - \left(a_0 + a_1 300 + a_2 300^2 + a_3 300^3\right)\right)^2 + \cdots + \\
&\tfrac{1}{14}\left(1.234 - \left(a_0 + a_1 1000 + a_2 1000^2 + a_3 1000^3\right)\right)^2
\end{aligned}$$

Fermat's Theorem gives necessary conditions for minimality:

$$\begin{aligned}
\frac{\partial}{\partial a_0} \quad & \tfrac{1}{7}\left(0.791 - \left(a_0 + a_1 250 + a_2 250^2 + a_3 250^3\right)\right)(-1) + \\
& \tfrac{1}{7}\left(0.846 - \left(a_0 + a_1 300 + a_2 300^2 + a_3 300^3\right)\right)(-1) + \cdots + \\
& \tfrac{1}{7}\left(1.234 - \left(a_0 + a_1 1000 + a_2 1000^2 + a_3 1000^3\right)\right)(-1) = 0
\end{aligned}$$

Example: Curve Fitting

$$\frac{\partial}{\partial a_1}: \quad \tfrac{1}{7}\left(0.791-\left(a_0+a_1 250+a_2 250^2+a_3 250^3\right)\right)(-250)+$$
$$\tfrac{1}{7}\left(0.846-\left(a_0+a_1 300+a_2 300^2+a_3 300^3\right)\right)(-300)+\cdots+$$
$$\tfrac{1}{7}\left(1.234-\left(a_0+a_1 1000+a_2 1000^2+a_3 1000^3\right)\right)(-1000)=0$$

$$\frac{\partial}{\partial a_2}: \quad \tfrac{1}{7}\left(0.791-\left(a_0+a_1 250+a_2 250^2+a_3 250^3\right)\right)(-250^2)+$$
$$\tfrac{1}{7}\left(0.846-\left(a_0+a_1 300+a_2 300^2+a_3 300^3\right)\right)(-300^2)+\cdots+$$
$$\tfrac{1}{7}\left(1.234-\left(a_0+a_1 1000+a_2 1000^2+a_3 1000^3\right)\right)(-1000^2)=0$$

$$\frac{\partial}{\partial a_3}: \quad \tfrac{1}{7}\left(0.791-\left(a_0+a_1 250+a_2 250^2+a_3 250^3\right)\right)(-250^3)+$$
$$\tfrac{1}{7}\left(0.846-\left(a_0+a_1 300+a_2 300^2+a_3 300^3\right)\right)(-300^3)+\cdots+$$
$$\tfrac{1}{7}\left(1.234-\left(a_0+a_1 1000+a_2 1000^2+a_3 1000^3\right)\right)(-1000^3)=0$$

Example: Curve Fitting

Consider the first equation. After scaling it becomes:

$$\left(0.791-\left(a_0+a_1 250+a_2 250^2+a_3 250^3\right)\right)+$$
$$\left(0.846-\left(a_0+a_1 300+a_2 300^2+a_3 300^3\right)\right)+\cdots+$$
$$\left(1.234-\left(a_0+a_1 1000+a_2 1000^2+a_3 1000^3\right)\right)=0$$

This takes the form: $k-k_0 a_0-k_1 a_1-k_2 a_2-k_3 a_3=0$

k is the sum of the elements of vector **Cp** and can therefore be calculated using the **sum** command.

```
>> k = sum(Cp);
```

k_0 is just the number of elements of **Cp**, i.e. 14.

Example: Curve Fitting

$$\left(0.791-\left(a_0+a_1 250+a_2 250^2+a_3 250^3\right)\right)(-1)+$$
$$\left(0.846-\left(a_0+a_1 300+a_2 300^2+a_3 300^3\right)\right)(-1)+\cdots+$$
$$\left(1.234-\left(a_0+a_1 1000+a_2 1000^2+a_3 1000^3\right)\right)(-1)=0$$

$$k-k_0 a_0-k_1 a_1-k_2 a_2-k_3 a_3=0$$

k_1 is the sum of the elements of vector **T**

```
>> k1 = sum(T);
```

k_2 is the sum of the squares of the elements of vector **T**

```
>> k2 = sum(T.^2);
```

k_3 is the sum of the cubes of the elements of vector **T**

```
>> k3 = sum(T.^3);
```


Example: Curve Fitting

This gives the first equation in a simple form:

$$14.567=14a_0+8200a_1+5.475\times10^6 a_2+4.0285\times10^9 a_3$$

We must repeat the process for the other three equation.

Example: Curve Fitting

Consider the second equation. After scaling it becomes:

$$\begin{aligned}&\left(0.791-\left(a_0+a_1 250+a_2 250^2+a_3 250^3\right)\right)(-250)+\\&\left(0.846-\left(a_0+a_1 300+a_2 300^2+a_3 300^3\right)\right)(-300)+\cdots+\\&\left(1.234-\left(a_0+a_1 1000+a_2 1000^2+a_3 1000^3\right)\right)(-1000)=0\end{aligned}$$

This again takes the form: $k-k_0a_0-k_1a_1-k_2a_2-k_3a_3=0$

k is the sum of the elements of vector **Cp** multiplied term-by-term by the elements of **-T**. Term-by-term multiplication is achieved by the **.*** command:

```
>> k = sum(Cp.*(-T));
```


Example: Curve Fitting

$$\begin{aligned}&\left(0.791-\left(a_0+a_1 250+a_2 250^2+a_3 250^3\right)\right)(-250)+\\&\left(0.846-\left(a_0+a_1 300+a_2 300^2+a_3 300^3\right)\right)(-300)+\cdots+\\&\left(1.234-\left(a_0+a_1 1000+a_2 1000^2+a_3 1000^3\right)\right)(-1000)=0\end{aligned}$$

$$k-k_0a_0-k_1a_1-k_2a_2-k_3a_3=0$$

k_0 is the sum of the elements of vector **-T**

```
>> k0 = sum(-T);
```

k_1 is minus the sum of the squares of the elements **T**

```
>> k1 = -sum(T.^2);
```

k_2 is the sum of the cubes of the elements of vector **-T**

```
>> k2 = sum((-T).^3);
```

Example: Curve Fitting

k_3 is minus the sum of the fourth powers of the elements of **T**

```
>> k3 = -sum(T.^4);
```

This gives the second equation in a simple form:

$$8926.1 = 8200a_0 + 5.475\times10^6 a_1 + 4.0285\times10^9 a_2 + 3.1779375\times10^{12} a_3$$

Treatment of the remaining two equations is similar.

Example: Curve Fitting

Finally we obtain four linear equations in the four unknown coefficients:

$$14.567 = 14a_0 + 8200a_1 + 5.475\times10^6 a_2 + 4.0285\times10^9 a_3$$

$$8926.1 = 8200a_0 + 5.475\times10^6 a_1 + 4.0285\times10^9 a_2 + 3.1779375\times10^{12} a_3$$

$$6160900 = 5.475\times10^6 a_0 + 4.0285\times10^9 a_1 + 3.1779375\times10^{12} a_2 + 2.63626375\times10^{15} a_3$$

$$4.63918025\times10^9 = 4.0285\times10^9 a_0 + 3.1779375\times10^{12} a_1 + 2.63626375\times10^{15} a_2 + 2.26980421875\times10^{18} a_3$$

Example: Curve Fitting

$$14.567 = 14a_0 + 8200a_1 + 5.475\times10^6 a_2 + 4.0285\times10^9 a_3$$

$$8.9261 = 8.2a_0 + 5.475\times10^3 a_1 + 4.0285\times10^6 a_2 + 3.1779375\times10^9 a_3$$

$$6.1609 = 5.475a_0 + 4.0285\times10^3 a_1 + 3.1779375\times10^6 a_2 + 2.63626375\times10^9 a_3$$

$$4.63918025 = 4.0285a_0 + 3.1779375\times10^3 a_1 + 2.63626375\times10^6 a_2 + 2.26980421875\times10^9 a_3$$

To improve the conditioning scale:

$$b_0 = a_0\,,\ b_1 = 10^3 a_1\,,\ b_2 = 10^6 a_2\,,\ b_3 = 10^9 a_3$$

Example: Curve Fitting

$$14.567 = 14b_0 + 8.2b_1 + 5.475b_2 + 4.0285b_3$$

$$8.9261 = 8.2b_0 + 5.475b_1 + 4.0285b_2 + 3.1779375b_3$$

$$6.1609 = 5.475b_0 + 4.0285b_1 + 3.1779375b_2 + 2.63626375b_3$$

$$4.63918025 = 4.0285b_0 + 3.1779375b_1 + 2.63626375b_2 + 2.26980421875b_3$$

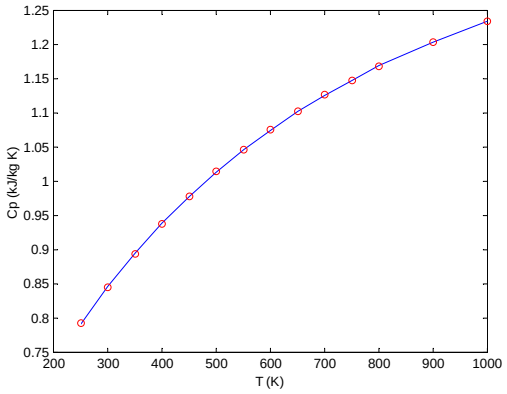
Solve by *Gaussian elimination*. We will see this algorithm later. For now note that the Matlab command **inv** effectively achieves the same result.

$$b_0 = 0.4418\,,\ b_1 = 1.7122\,,\ b_2 = -1.3515\,,\ b_3 = 0.4322$$

Example: Curve Fitting

We have the following approximation:

$$C_p(T) = 0.4418 + 1.7122\left(\frac{T}{1000}\right) - 1.3515\left(\frac{T}{1000}\right)^2 + 0.4322\left(\frac{T}{1000}\right)^3$$



The approximation (red circles) is seen to be very good. Mean Square Error equals 5.1875×10^{-7}. Maximum percentage error equals 0.14%

Curve Fitting

Minimization of the mean square error is certainly a popular method for determining the "best" parameter values in an approximate formula. Moreover there are reasons (probably best offered in terms of Fourier theory for those of you familiar with this mathematical field) why this procedure will often yield approximations which we consider to be very good. Nonetheless, no procedure is a panacea. There are occasions where the approximating formulae generated by this method *do not* yield satisfactory levels of approximation.

Example: Curve Fitting

The *enthalpy* on a molal basis of CO_2 is measured at a constant pressure of 0.1 Mpa. The change in enthalpy from the value at 298K is shown:

T (K)	$\bar{h} - \bar{h}_{298}$ (kJ/kmol)	T (K)	$\bar{h} - \bar{h}_{298}$ (kJ/kmol)	T (K)	$\bar{h} - \bar{h}_{298}$ (kJ/kmol)
298	0	1100	38885	2000	91439
300	69	1200	44473	2200	103562
400	4003	1300	50148	2400	115779
500	8305	1400	55895	2600	128074
600	12906	1500	61705	2800	140435
700	17754	1600	67569	3000	152853
800	22806	1700	73480	3200	165321
900	28030	1800	79432	3400	177836
1000	33397	1900	85420	3600	190394

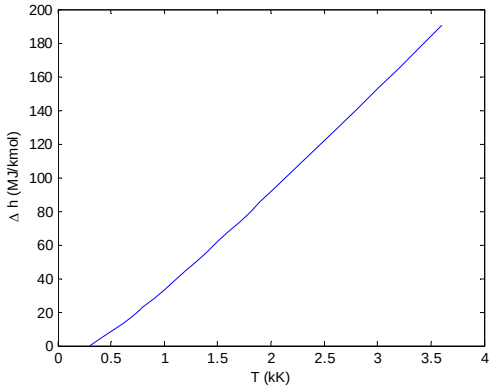
Example: Curve Fitting

It is possible to apply the procedure above to find the coefficients of a cubic polynomial which gives the least mean square error fit. The size of the data points (T being up to 3600 and the change in enthalpy up to 190394) renders the numerical problem rather difficult. We seek a polynomial in $T/1000$ rather than in T (i.e. measure temperature in kK) and we consider the scaled enthalpy data:

$$\frac{\left(\bar{h} - \bar{h}_{298}\right)}{1000}$$

i.e. measure the change in enthalpy in MJ/kmol.

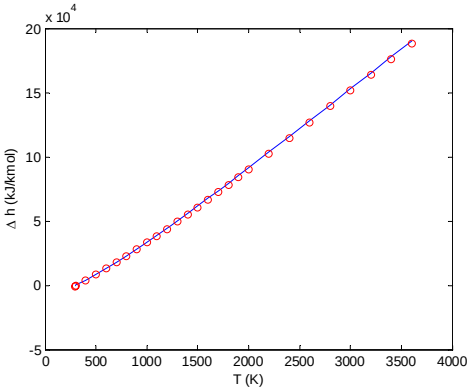
Example: Curve Fitting



The scaled data permits a least mean square error fit by the cubic polynomial:

$$\frac{\overline{h}-\overline{h}_{298}}{1000} \cong -12.88 + 38.3\left(\frac{T}{1000}\right) + 8.813\left(\frac{T}{1000}\right)^2 - 1.096\left(\frac{T}{1000}\right)^3$$

Example: Curve Fitting



The approximation (red circles) is again *seen* to be very good. However the maximum percentage error equals *infinity*.

The percentage error is rather high at the lower temperatures of 298K and 300K because the value of the enthalpy change is small here so that almost any error will comprise a large percentage error.

Curve Fitting

The lesson is that if the goal is to achieve an approximation which gives a small percentage error, then the procedure of minimizing the mean square error will, in general, not achieve this result. If the experimental value is exactly 0 at some setting of the control variable then any here error at all corresponds to a percentage error of infinity. The way around this problem is to impose a *constraint* on the approximating function, requiring that it give exactly 0 for this particular value of the control variable. We are rather naturally led to the problem of *constrained optimisation*.

Newton's Method

When the system of equations to be solved derives from applying Fermat's theorem to find local extrema of a function of several variables, the system will take the form

$$g(\mathrm{x}) = \nabla f(\mathrm{x})$$

where $f(\mathrm{x})$ is the scalar objective function in several variables. The Newton-Raphson method for the solution of the system in this case has the rather complicated looking form

$$\mathrm{x}_{n+1} = \mathrm{x}_n - \left(D\nabla f(\mathrm{x}_n)\right)^{-1} \nabla f(\mathrm{x}_n)$$

Newton's Method

$$\mathrm{x}_{n+1} = \mathrm{x}_n - \left(D\nabla f(\mathrm{x}_n)\right)^{-1} \nabla f(\mathrm{x}_n)$$

We seek clarification of the nature of the matrix

$$D\nabla f(\mathrm{x})$$

i.e. the Jacobian matrix of the gradient of a scalar function of several variables $f(\mathrm{x})$

$$\nabla f(\mathrm{x}) = \begin{bmatrix} \frac{\partial f(\mathrm{x})}{\partial x_1} \\ \vdots \\ \frac{\partial f(\mathrm{x})}{\partial x_m} \end{bmatrix} = g(\mathrm{x}) = \begin{bmatrix} g_1(\mathrm{x}) \\ \vdots \\ g_m(\mathrm{x}) \end{bmatrix}$$

Newton's Method

$$D\nabla f(\mathrm{x}) = Dg(\mathrm{x}) = \begin{bmatrix} \frac{\partial g_1(\mathrm{x})}{\partial x_1} & \cdots & \frac{\partial g_1(\mathrm{x})}{\partial x_m} \\ \vdots & & \vdots \\ \frac{\partial g_m(\mathrm{x})}{\partial x_1} & \cdots & \frac{\partial g_m(\mathrm{x})}{\partial x_m} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{\partial^2 f(\mathrm{x})}{\partial x_1^2} & \cdots & \frac{\partial^2 f(\mathrm{x})}{\partial x_m \partial x_1} \\ \vdots & & \vdots \\ \frac{\partial^2 f(\mathrm{x})}{\partial x_1 \partial x_m} & \cdots & \frac{\partial^2 f(\mathrm{x})}{\partial x_m^2} \end{bmatrix}$$

This matrix of the second partial derivatives is called the *Hessian matrix*. It is commonly symmetric by Clairaut's theorem and is denoted $Hf(x)$.

Newton's Method

$$x_{n+1} = x_n - \left(H f(x_n)\right)^{-1} \nabla f(x_n)$$

The numerical task of finding the Hessian matrix and its inverse in each iteration can comprise a significant portion of the numerical effort when the order *m* of the vector x is large. Accordingly there have been developed a number of *quasi-newton methods* which employ a variety of alternative, more efficient procedures for approximately determining either the Hessian matrix or its inverse. For example the 1970 BFGS (Broyden, Fletcher, Goldfarb, Shanno) method enjoys widespread use.

Constrained Optimisation

The problems considered above continue to be the exception rather than the rule. It is far more common to happen upon a *constrained optimisation problem* rather than an *unconstrained optimisation problem* . Indeed we saw how curve fitting led to such a problem. Two types of constraints arise, *equality constraints* and *inequality constraints*. In fact the first example of an optimisation problem which we considered was actually a constrained optimisation problem.

Example

Two real numbers sum to 20, what is the largest possible value of their product?

$$h(x,y) = x + y - 20 = 0 \quad , \quad f(x,y) = xy$$

Lagrange offers a brilliant result concerning constrained optimisation when the constraints are equalities. In this case the method calls for the construction of an auxiliary function and an increase in the number of control variables

$$\Lambda(x,y,\lambda) = f(x,y) + \lambda h(x,y) = xy + \lambda(x + y - 20)$$

Example

Lagrange's result is the statement that if (x_0,y_0) is a local extremum of the constrained objective function then there exists λ_0 such that (x_0,y_0,λ_0) is a stationary point of the auxiliary function.

$$\frac{\partial \Lambda(x,y,\lambda)}{\partial x} = y + \lambda = 0 \qquad \lambda = -y \quad , \quad \lambda = -x$$

$$\frac{\partial \Lambda(x,y,\lambda)}{\partial y} = x + \lambda = 0 \qquad x = y$$

$$\frac{\partial \Lambda(x,y,\lambda)}{\partial \lambda} = x + y - 20 = 0 \qquad 2x - 20 = 0$$

$$x = y = 10$$

Constrained Optimisation

More generally if x* is a local extremum of the objective function in several control variables $f(\mathrm{x})$ subject to the equality constraints

$$h_1(\mathrm{x}) = 0\,, \ldots, h_r(\mathrm{x}) = 0$$

then there exists λ* such that (x*,λ*) is a stationary point of the auxiliary function in an increased number of control variables

$$\Lambda(\mathrm{x},\lambda) = f(\mathrm{x}) + \lambda_1 h_1(\mathrm{x}) + \cdots + \lambda_r h_r(\mathrm{x})$$

The additional control variables $\lambda_1, \ldots, \lambda_r$ are called *Lagrange multipliers*.

Example

Find maximum of $f(x,y) = xy$ subject to equality constraint

$$h(x,y) = \frac{x^2}{8} + \frac{y^2}{2} - 1 = 0$$

$$\Lambda(x,y,\lambda) = xy + \lambda\left(\frac{x^2}{8} + \frac{y^2}{2} - 1\right)$$

$$\frac{\partial\Lambda(x,y,\lambda)}{\partial x} = y + \frac{\lambda x}{4} = 0$$

$$\frac{\partial\Lambda(x,y,\lambda)}{\partial y} = x + \lambda y = 0$$

$$\frac{\partial\Lambda(x,y,\lambda)}{\partial \lambda} = \frac{x^2}{8} + \frac{y^2}{2} - 1 = 0$$

Stationarity conditions.

Actual solutions

$$(x,y,\lambda) = (2,-1,2), (2,1,-2), (-2,1,2), (-2,-1,2)$$

Example

$$\begin{bmatrix} y + \dfrac{\lambda x}{4} \\ x + \lambda y \\ \dfrac{x^2}{8} + \dfrac{y^2}{2} - 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$ Stationarity conditions

The Hessian of the auxiliary function is

$$\begin{bmatrix} \frac{\lambda}{4} & 1 & \frac{x}{4} \\ 1 & \lambda & y \\ \frac{x}{4} & y & 0 \end{bmatrix}$$

Example

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \\ \lambda_{n+1} \end{bmatrix} = \begin{bmatrix} x_n \\ y_n \\ \lambda_n \end{bmatrix} - \begin{bmatrix} \frac{\lambda_n}{4} & 1 & \frac{x_n}{4} \\ 1 & \lambda_n & y_n \\ \frac{x_n}{4} & y_n & 0 \end{bmatrix}^{-1} \begin{bmatrix} y_n + \frac{\lambda_n x_n}{4} \\ x_n + \lambda_n y_n \\ \frac{x_n^2}{8} + \frac{y_n^2}{2} - 1 \end{bmatrix}$$

$(x_0, y_0, \lambda_0) = (1, 1, 0)$

$(x_1, y_1, \lambda_1) = (3.2500, 0.8125, -3.2500)$

$(x_2, y_2, \lambda_2) = (2.2868, 0.9752, -2.1635)$

$(x_3, y_3, \lambda_3) = (2.0158, 1.0016, -2.0084)$

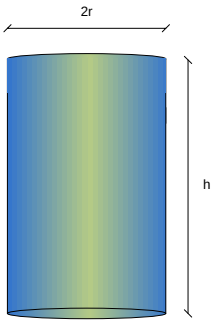
$(x_4, y_4, \lambda_4) = (2.0000, 1.0000, -2.0000)$

Application of Lagranges's theorem: oil can

An oil can is to contain 1 litre of oil. The can is cylindrical in shape. The cost of manufacture of the top and bottom of the can is 10 euro per m^2 and that of the side is 7 euro per m^2. What dimensions necessitate least cost?

$$Vol = \pi r^2 h = 1{,}000 \ \text{cm}^3$$

$$Cost = 10\times10^{-4}\left(2\pi r^2\right)+7\times10^{-4}\left(2\pi rh\right)$$



Application of Lagranges's theorem: oil can

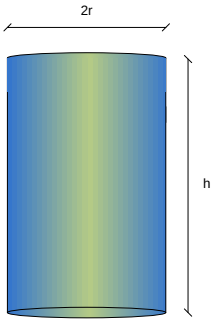
Minimize 10^4 times the cost. Evidently the answer is the same but the numbers are easier. Auxiliary:

$$\Lambda(r,h,\lambda)=10\left(2\pi r^2\right)+7\left(2\pi rh\right)+\lambda\left(\pi r^2h-1{,}000\right)$$

$$\frac{\partial\Lambda(r,h,\lambda)}{\partial r}=40\pi r+14\pi h+\lambda 2\pi rh=0$$

$$\frac{\partial\Lambda(r,h,\lambda)}{\partial h}=14\pi r+\lambda\pi r^2=0$$

$$\frac{\partial\Lambda(r,h,\lambda)}{\partial\lambda}=\pi r^2h-1{,}000=0$$



Application of Lagranges's theorem: oil can

Stationarity conditions:

$$\begin{bmatrix} 20r + 7h + \lambda rh \\ 14r + \lambda r^2 \\ \pi r^2 h - 1{,}000 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

The Hessian of the auxiliary function is

$$\begin{bmatrix} 20 + \lambda h & 7 + \lambda r & rh \\ 14 + 2\lambda r & 0 & r^2 \\ 2\pi rh & \pi r^2 & 0 \end{bmatrix}$$

Example

$$\begin{bmatrix} r_{n+1} \\ h_{n+1} \\ \lambda_{n+1} \end{bmatrix} = \begin{bmatrix} r_n \\ h_n \\ \lambda_n \end{bmatrix} - \begin{bmatrix} 20 + \lambda_n h_n & 7 + \lambda_n r_n & r_n h_n \\ 14 + 2\lambda_n r_n & 0 & r_n^2 \\ 2\pi r_n h_n & \pi r_n^2 & 0 \end{bmatrix}^{-1} \begin{bmatrix} 20r_n + 7h_n + \lambda_n r_n h_n \\ 14r_n + \lambda_n r_n^2 \\ \pi r_n^2 h_n - 1{,}000 \end{bmatrix}$$

$(r_0, h_0, \lambda_0) = (5, 13, 0)$

$(r_1, h_1, \lambda_1) = (5.1350, 12.0305, -2.8756)$

$(r_2, h_2, \lambda_2) = (4.8228, 13.5347, -2.9103)$

$(r_3, h_3, \lambda_3) = (4.8118, 13.7468, -2.9095)$

$(r_4, h_4, \lambda_4) = (4.8118, 13.7480, -2.9095)$

Application of Lagranges's theorem: oil can

The choice of initial condition is rather important here. For many initial conditions the recursion fails to converge.

Example: Curve Fitting

Recall the *enthalpy* on a molal basis of CO_2 is measured at a constant pressure of 0.1 Mpa. The change in enthalpy from the value at 298K is shown:

T (K)	$\bar{h} - \bar{h}_{298}$ (kJ/kmol)	T (K)	$\bar{h} - \bar{h}_{298}$ (kJ/kmol)	T (K)	$\bar{h} - \bar{h}_{298}$ (kJ/kmol)
298	0	1100	38885	2000	91439
300	69	1200	44473	2200	103562
400	4003	1300	50148	2400	115779
500	8305	1400	55895	2600	128074
600	12906	1500	61705	2800	140435
700	17754	1600	67569	3000	152853
800	22806	1700	73480	3200	165321
900	28030	1800	79432	3400	177836
1000	33397	1900	85420	3600	190394

Example: Curve Fitting

Fitting this data by a cubic polynomial which gives the least mean square error fit did not achieve a small maximum percentage error. The problem was that the cubic polynomial was not constrained to agree with the value of 0 at $T = 298$ K. We seek instead a cubic polynomial in $T/1000$ rather than in T which gives least mean square fit to the scaled enthalpy data:

$$\frac{\left(\bar{h} - \bar{h}_{298}\right)}{1000}$$

subject to the constraint that at $T = 298$ K the cubic yields the value 0.

Example: Curve Fitting

Firstly let us create two vectors in Matlab which contain respectively all of the temperature values and all of the associated specific heat capacity values:

```
>> T = [298 [300:100:2000] [2200:200:3600]];   initialise vector T
>> H = [0 60 4003 8305 12906 17754 22806 28030 33397 38885
44473 50148 55895 61705 67569 73480 79432 85420 91439
103562 115779 128074 140435 152853 165321 177836 190394];
        initialise vector H
```

Note the faster method for generating vector **T**.

Example: Curve Fitting

We elect to approximate the data by a cubic polynomial:

$$\frac{\bar{h} - \bar{h}_{298}}{1000} = a_0 + a_1\left(\frac{T}{1000}\right) + a_2\left(\frac{T}{1000}\right)^2 + a_3\left(\frac{T}{1000}\right)^3$$

How then should we choose the coefficients a_i ? Well, firstly we have the constraint:

$$0 = a_0 + a_1\left(\frac{298}{1000}\right) + a_2\left(\frac{298}{1000}\right)^2 + a_3\left(\frac{298}{1000}\right)^3$$

We may either use Lagrange's method to deal with this constraint or explicitly solve it for a_0. We choose the former method.

Example: Curve Fitting

Given our decision concerning scaling we should scale the vectors appropriately:

```
>> Tscaled = T/1000;        scale vector T
>> Hscaled = H/1000;        scale vector H
```

Example: Curve Fitting

The auxiliary function becomes:

$$\begin{aligned}&\left(0.0600-\left(a_0+a_1 0.3+a_2 0.3^2+a_3 0.3^3\right)\right)^2+\\&\left(4.0030-\left(a_0+a_1 0.4+a_2 0.4^2+a_3 0.4^3\right)\right)^2+\cdots+\\&\left(190.394-\left(a_0+a_1 3.6+a_2 3.6^2+a_3 3.6^3\right)\right)^2\\&+\lambda\left(a_0+a_1 0.298+a_2 0.298^2+a_3 0.298^3\right)\end{aligned}$$

Fermat's Theorem gives necessary conditions for minimality:

$$\frac{\partial}{\partial a_0}$$

$$\begin{aligned}&2\left(0.0600-\left(a_0+a_1 0.3+a_2 0.3^2+a_3 0.3^3\right)\right)(-1)+\\&2\left(4.0030-\left(a_0+a_1 0.4+a_2 0.4^2+a_3 0.4^3\right)\right)(-1)+\cdots+\\&2\left(190.394-\left(a_0+a_1 3.6+a_2 3.6^2+a_3 3.6^3\right)\right)(-1)+\lambda=0\end{aligned}$$

Example: Curve Fitting

$$\frac{\partial}{\partial a_1}$$

$$\begin{aligned}&2\left(0.0600-\left(a_0+a_1 0.3+a_2 0.3^2+a_3 0.3^3\right)\right)(-0.3)+\\&2\left(4.0030-\left(a_0+a_1 0.4+a_2 0.4^2+a_3 0.4^3\right)\right)(-0.4)+\cdots+\\&2\left(190.394-\left(a_0+a_1 3.6+a_2 3.6^2+a_3 3.6^3\right)\right)(-3.6)+\lambda(0.298)=0\end{aligned}$$

$$\frac{\partial}{\partial a_2}$$

$$\begin{aligned}&2\left(0.0600-\left(a_0+a_1 0.3+a_2 0.3^2+a_3 0.3^3\right)\right)\left(-0.3^2\right)+\\&2\left(4.0030-\left(a_0+a_1 0.4+a_2 0.4^2+a_3 0.4^3\right)\right)\left(-0.4^2\right)+\cdots+\\&2\left(190.394-\left(a_0+a_1 3.6+a_2 3.6^2+a_3 3.6^3\right)\right)\left(-3.6^2\right)+\lambda\left(0.298^2\right)=0\end{aligned}$$

Example: Curve Fitting

$$\frac{\partial}{\partial a_3}$$

$$2\left(0.0600-\left(a_0+a_1 0.3+a_2 0.3^2+a_3 0.3^3\right)\right)\left(-0.3^3\right)+$$
$$2\left(4.0030-\left(a_0+a_1 0.4+a_2 0.4^2+a_3 0.4^3\right)\right)\left(-0.4^3\right)+\cdots+$$
$$2\left(190.394-\left(a_0+a_1 3.6+a_2 3.6^2+a_3 3.6^3\right)\right)\left(-3.6^3\right)+\lambda\left(0.298^3\right)=0$$

Finally of course we have the constraint:

$$\frac{\partial}{\partial \lambda}$$

$$a_0+a_1 0.298+a_2 0.298^2+a_3 0.298^3=0$$

Example: Curve Fitting

$$2\left(0.0600-\left(a_0+a_1 0.3+a_2 0.3^2+a_3 0.3^3\right)\right)(-1)+$$
$$2\left(4.0030-\left(a_0+a_1 0.4+a_2 0.4^2+a_3 0.4^3\right)\right)(-1)+\cdots+$$
$$2\left(190.394-\left(a_0+a_1 3.6+a_2 3.6^2+a_3 3.6^3\right)\right)(-1)+\lambda=0$$

$$k+k_0 a_0+k_1 a_1+k_2 a_2+k_3 a_3+\lambda=0$$

```
>> k = -2*sum(Hscaled);
>> k0 = 2*(length(Tscaled)-1);
>> k1 = 2*sum(Tscaled);
>> k2 = 2*sum(Tscaled.^2);
>> k3 = 2*sum(Tscaled.^3);
```

$$-3{,}899.922+52a_0+88.396a_1+195.398a_2+507.691a_3+\lambda=0$$

Example: Curve Fitting

$$2\left(0.0600-\left(a_0+a_1 0.3+a_2 0.3^2+a_3 0.3^3\right)\right)\left(-0.3\right)+$$
$$2\left(4.0030-\left(a_0+a_1 0.4+a_2 0.4^2+a_3 0.4^3\right)\right)\left(-0.4\right)+\cdots+$$
$$2\left(190.394-\left(a_0+a_1 3.6+a_2 3.6^2+a_3 3.6^3\right)\right)\left(-3.6\right)+\lambda\left(0.298\right)=0$$
$$k+k_0 a_0+k_1 a_1+k_2 a_2+k_3 a_3+\lambda\left(0.298\right)=0$$

```
>> k = -2*sum(Hscaled.*Tscaled);
>> k0 = 2*sum(Tscaled);
>> k1 = 2*sum(Tscaled.^2);
>> k2 = 2*sum(Tscaled.^3);
>> k3 = 2*sum(Tscaled.^4);
```

$$-9{,}331.399+88.396a_0+195.398a_1+507.691a_2+1{,}446.984a_3+0.298\lambda=0$$

Example: Curve Fitting

$$2\left(0.0600-\left(a_0+a_1 0.3+a_2 0.3^2+a_3 0.3^3\right)\right)\left(-0.3^2\right)+$$
$$2\left(4.0030-\left(a_0+a_1 0.4+a_2 0.4^2+a_3 0.4^3\right)\right)\left(-0.4^2\right)+\cdots+$$
$$2\left(190.394-\left(a_0+a_1 3.6+a_2 3.6^2+a_3 3.6^3\right)\right)\left(-3.6^2\right)+\lambda\left(0.298^2\right)=0$$
$$k+k_0 a_0+k_1 a_1+k_2 a_2+k_3 a_3+\lambda\left(0.298^2\right)=0$$

```
>> k = -2*sum(Hscaled.*(Tscaled.^2));
>> k0 = 2*sum(Tscaled.^2);
>> k1 = 2*sum(Tscaled.^3);
>> k2 = 2*sum(Tscaled.^4);
>> k3 = 2*sum(Tscaled.^5);
```

$$-25{,}149.171+195.398a_0+507.691a_1+1{,}446.984a_2+4{,}365.951a_3+0.298^2\lambda=0$$

Example: Curve Fitting

$$2\left(0.0600-\left(a_0+a_1 0.3+a_2 0.3^2+a_3 0.3^3\right)\right)\left(-0.3^3\right)+$$
$$2\left(4.0030-\left(a_0+a_1 0.4+a_2 0.4^2+a_3 0.4^3\right)\right)\left(-0.4^3\right)+\cdots+$$
$$2\left(190.394-\left(a_0+a_1 3.6+a_2 3.6^2+a_3 3.6^3\right)\right)\left(-3.6^3\right)+\lambda\left(0.298^3\right)=0$$
$$k+k_0 a_0+k_1 a_1+k_2 a_2+k_3 a_3+\lambda\left(0.298^3\right)=0$$

```
>> k = -2*sum(Hscaled.*(Tscaled.^3));
>> k0 = 2*sum(Tscaled.^3);
>> k1 = 2*sum(Tscaled.^4);
>> k2 = 2*sum(Tscaled.^5);
>> k3 = 2*sum(Tscaled.^6);
```

$$-73{,}097.179+507.691a_0+1{,}446.984a_1+4{,}365.951a_2+13{,}672.148a_3+0.298^3\lambda=0$$

Example: Curve Fitting

We obtain five linear equations in five unknowns (four coefficients and a Lagrange multiplier):

$$-3{,}899.922+52a_0+88.396a_1+195.398a_2+507.691a_3+\lambda=0$$

$$-9{,}331.399+88.396a_0+195.398a_1+507.691a_2$$
$$+1{,}446.984a_3+0.298\lambda=0$$

$$-25{,}149.171+195.398a_0+507.691a_1+1{,}446.984a_2$$
$$+4{,}365.951a_3+0.298^2\lambda=0$$

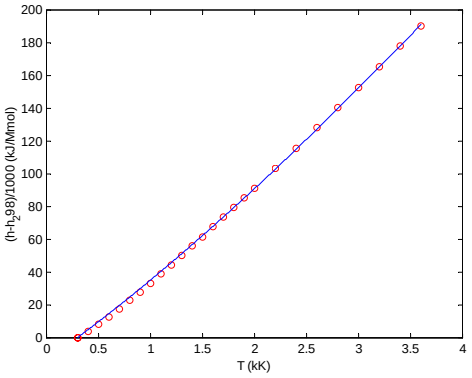
$$-73{,}097.179+507.691a_0+1{,}446.984a_1+4{,}365.951a_2$$
$$+13{,}672.148a_3+0.298^3\lambda=0$$

$$a_0+a_1 0.298+a_2 0.298^2+a_3 0.298^3=0$$

Example: Curve Fitting

Solving yields:

$$\frac{\overline{h}(T)-\overline{h}_{298}}{1000} = -13.752 + 44.9313\left(\frac{T}{1000}\right) + 4.4176\left(\frac{T}{1000}\right)^2 - 0.2226\left(\frac{T}{1000}\right)^3$$



The approximation (red circles) is seen to be very good. Mean Square Error equals 1.107.

Constrained Optimisation

In the case where the constraints are not just equality constraints but also include some inequality constraints it is necessary to employ a more general result than Lagrange's, a result apparently derived first by Karush and then independently by Kuhn and Tucker. The problem considered is

$$\begin{aligned} &\text{Minimize} && f(\mathrm{x}) \\ &\text{subject to} && g_i(\mathrm{x}) \le 0\,,\; h_j(\mathrm{x}) = 0 \end{aligned}$$

Constrained Optimisation

After the manner of Lagrange, Karush, Kuhn and Tucker essentially seek to transform the problem of solving the constrained optimisation problem into that of solving an unconstrained optimisation problem by introducing an extra set of control variables, one for each constraint. In effect they consider the auxiliary function:

$$f(\mathrm{x}) + \sum_i \mu_i g_i(\mathrm{x}) + \sum_j \lambda_j h_j(\mathrm{x})$$

The additional control variables $\mu_1, \ldots, \mu_q, \lambda_1, \ldots, \lambda_r$ are called *Karush-Kuhn-Tucker* or *KKT multipliers*.

Constrained Optimisation

The necessary conditions obtained by Karush, Kuhn and Tucker for x* to be a local minimum are

Stationarity $\nabla f(\mathrm{x}^*) + \sum_i \mu_i \nabla g_i(\mathrm{x}^*) + \sum_j \lambda_j \nabla h_j(\mathrm{x}^*) = \vec{0}$

Primal feasability $g_i(\mathrm{x}^*) \le 0\,,\; h_j(\mathrm{x}^*) = 0$

Dual feasability $\mu_i \ge 0$

Complementary slackness $\mu_i g_i(\mathrm{x}^*) = 0$

Constrained Optimisation

Unfortunately the KKT conditions fail to transform the constrained optimisation problem into an unconstrained optimisation problem. This failure manifests in the feasability conditions which offer inequalities, but not equalities

$$g_i\left(\mathrm{x}^*\right) \leq 0\,,\ \mu_i \geq 0$$

In practice the KKT conditions are best used to confirm optimality of a solution obtained by some other method.

Example (Kreyzig)

In a production process two types of containers K_1 and K_2 are manufactured using two machines M_1 and M_2. To produce a container of type K_1 takes two minutes processing with machine M_1 and four minutes with machine M_2. To produce a container of type K_2 takes eight minutes processing with machine M_1 and four minutes with machine M_2. The net profit for a container K_1 is \$29 and for a K_2 container is \$45. Determine a production schedule which maximizes profit.

Example (Kreyzig)

Assume that we produce x_1 containers of type K_1 and x_2 containers of type K_2 every hour. To maximize profit one must minimize minus the profit per hour, i.e. $f(x_1, x_2) = -29x_1 - 45x_2$

Machine usage establishes the following constraints, all of which are inequality constraints

$$g_1(x_1, x_2) = 2x_1 + 8x_2 - 60 \leq 0$$
$$g_2(x_1, x_2) = 4x_1 + 4x_2 - 60 \leq 0$$
$$g_3(x_1, x_2) = -x_1 \leq 0$$
$$g_4(x_1, x_2) = -x_2 \leq 0$$

Example (Kreyzig)

In the design of an optimal schedule the first task is to determine an optimum target number of containers of each type to produce per hour. It is fairly obvious that a schedule which has either machine lying idle for any period of time is surely not optimal in the sense of profit attained. Accordingly intuition suggests that the first two inequality constraints, those based on machine usage, will actually hold with equality for an optimal solution.

Example (Kreyzig)

This intuition yields the following problem

$$f(x_1, x_2) = -29x_1 - 45x_2$$

$$h_1(x_1, x_2) = 2x_1 + 8x_2 - 60 = 0$$
$$h_2(x_1, x_2) = 4x_1 + 4x_2 - 60 = 0$$
$$g_1(x_1, x_2) = -x_1 \leq 0$$
$$g_2(x_1, x_2) = -x_2 \leq 0$$

Where the linear equality constraints completely specify the solution, namely $(x_1, x_2) = (10,5)$ which we note to be feasible.

Example (Kreyzig)

The KKT conditions for the original problem are

$$-29 + 2\mu_1 + 4\mu_2 - \mu_3 = 0$$
$$-45 + 8\mu_1 + 4\mu_2 - \mu_4 = 0 \quad \text{Stationarity}$$

$$2x_1 + 8x_2 - 60 \leq 0 \quad , \quad 4x_1 + 4x_2 - 60 \leq 0$$
$$x_1 \geq 0 \quad , \quad x_2 \geq 0 \quad \text{Primal feasibility}$$

$$\mu_1 \geq 0 \quad , \quad \mu_2 \geq 0 \quad , \quad \mu_3 \geq 0 \quad , \quad \mu_4 \geq 0$$
Dual feasibility

$$\mu_1(2x_1 + 8x_2 - 60) = 0 \quad , \quad \mu_2(4x_1 + 4x_2 - 60) = 0$$
$$\mu_3 x_1 = 0 \quad , \quad \mu_4 x_2 = 0 \quad \text{Complementary Slackness}$$

Example (Kreyzig)

With $x_1 = 10, x_2 = 5$ primal feasibility clearly holds and complementary slackness is equivalent to $\mu_3 = 0$ and $\mu_4 = 0$. The stationarity conditions are

$$-29 + 2\mu_1 + 4\mu_2 = 0$$
$$-45 + 8\mu_1 + 4\mu_2 = 0$$

giving $\mu_1 = \frac{8}{3} \geq 0 \quad , \quad \mu_2 = \frac{71}{12} \geq 0$

so that dual feasibility is seen to hold.

Constrained Optimisation

The *penalty method* is one of the more commonly employed methods for solving constrained optimisation problems when at least some of the constraints are inequalities. The method replaces the problem not with a single unconstrained optimisation problem but rather with a series of such problems.

Special cases of constrained optimisation

In the event that the objective function and the constraints take on certain special forms (being linear for example) more specialised and more efficient optimisation methods are available. A particularly notable example is linear programming and Dantzig's simplex method.

Section 2 - Conclusion

- The problem of optimisation, constrained or otherwise, often leads to systems of equations.
- The resulting equations are not always readily amenable to solution.

Overview of Topics

Solution of equations by iteration.

Optimisation.

Solution of system of linear equations.

Solution of ordinary differential equations: initial value problems.

Numerical Quadrature.

Solution of ordinary differential equations: boundary value problems.

Solution of partial differential equations.

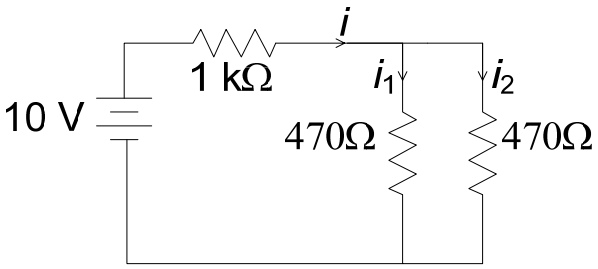
Section 3 – Solution of systems of linear equations

- ❑ Gaussian Elimination.
- ❑ Iteration Method.

Workload: 5 lectures + 1 laboratory + Autonomous learning

Example

The problem of solving systems of linear equations, also commonly known as simultaneous linear equations, is probably one of the earliest, important mathematical problems which you came across in your education. The simple circuit shown gives rise to the following system of equations



$$1000i + 470i_1 = 10$$
$$i - i_1 - i_2 = 0$$
$$470i_1 - 470i_2 = 0$$

Example

Obviously we can easily solve by noting that from the third equation $i_1 = i_2$. From the second $i = 2i_1$. So from the first $2{,}470i_1 = 10$. This gives $i_1 = i_2 = 4.0486$ mA and $i = 8.0972$ mA. But we can also take a more general approach.

Step 1: eliminate i from second and third equations

$$1000i + 470i_1 = 10$$
$$(10 - 470i_1)/1000 - i_1 - i_2 = 0$$
$$470i_1 - 470i_2 = 0$$

$$1000i + 470i_1 = 10$$
$$1.47i_1 + i_2 = 0.01$$
$$470i_1 - 470i_2 = 0$$

Example

Step 2: eliminate i_1 from third equation in new set

$$1000i + 470i_1 = 10$$
$$1.47i_1 + i_2 = 0.01$$
$$470(0.01 - i_2)/1.47 - 470i_2 = 0$$

$$1000i + 470i_1 = 10$$
$$1.47i_1 + i_2 = 0.01$$
$$789.7279i_2 = 3.1973$$

Step 3: solve third equation in new set for i_2

$$i_2 = \frac{3.1973}{789.7279} \cong 4.0486 \times 10^{-3}$$

Example

Step 4: substitute value for i_2 into last set

$$1000i + 470i_1 = 10$$
$$1.47i_1 = 0.0059514$$

Step 5: solve second equation in new set for i_1

$$i_1 = \frac{0.0059514}{1.47} \cong 4.0486 \times 10^{-3}$$

Step 6: substitute value for i_1 into last set

$$1000i = 8.09717$$

Example

Step 7: solve equation for i

$$i = \frac{8.09717}{1000} \cong 8.0972 \times 10^{-3}$$

This method of solving a system of linear equations is called *Gaussian elimination*. This name is standard but not appropriate. The method was essentially known in Han dynasty China almost two millenia before Gauss, appearing in a famous book *Jiuzhang suanshu* from this period. In the present case it is clearly more complicated than required. However, it has the virtue of being general.

System of Linear Equations

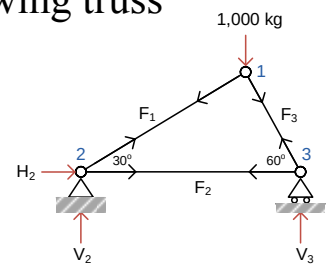
Systems of linear equations arise in all fields of engineering and probably in all fields (although I do not have the expertise to vouch for the latter). Before we more carefully investigate the method of Gaussian elimination let us consider a few more examples. In this regard we saw that the problem of curve fitting using polynomial approximations led to equations of this kind.

Statically Determinate Truss

In the analysis of trusses we assume that all loads are applied at joints only. If the weight of a member is significant then half of this weight is considered to be applied to the joint at each end of the member. It is assumed that every member is in pure compression or pure tension. Shear, bending moments, *etc* are all considered to be negligible and are therefore neglected. Obviously this leads to a rather idealised analysis. The simplest mathematical model arises when the truss is *statically determinate*, i.e. when $2j = m + 3$, j being the number of *joints* and m the number of *members*.

Example: Truss (Chapra and Canale)

With minor modification to metric system Chapra and Canale consider the following truss



$j = 3$, $m = 3$ so it is statically determinate. Forces F_1, F_2 and F_3 denote tension (if positive) or compression (if negative). The roller can apply a vertical force only, the hinge can apply horizontal and vertical forces.

Statically Determinate Truss

Analysis is based on horizontal and vertical force (in N) balance at each joint or node.

$$\begin{aligned}
&\text{Joint 1 : horizontal} && -F_1\cos(30^\circ)+F_3\cos(60^\circ)=0\\
&\text{Joint 1 : vertical} && -F_1\sin(30^\circ)-F_3\sin(60^\circ)-1000g=0\\
&\text{Joint 2 : horizontal} && F_1\cos(30^\circ)+F_2+H_2=0\\
&\text{Joint 2 : vertical} && F_1\sin(30^\circ)+V_2=0\\
&\text{Joint 3 : horizontal} && -F_2-F_3\cos(60^\circ)=0\\
&\text{Joint 3 : vertical} && F_3\sin(60^\circ)+V_3=0
\end{aligned}$$

Statically Determinate Truss

This is a system of six linear equations in six unknowns. In matrix-vector form, with a few sign changes, it becomes:

$$\begin{bmatrix}
\cos(30^\circ) & 0 & -\cos(60^\circ) & 0 & 0 & 0\\
\sin(30^\circ) & 0 & \sin(60^\circ) & 0 & 0 & 0\\
\cos(30^\circ) & 1 & 0 & 1 & 0 & 0\\
\sin(30^\circ) & 0 & 0 & 0 & 1 & 0\\
0 & 1 & \cos(60^\circ) & 0 & 0 & 0\\
0 & 0 & \sin(60^\circ) & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix} F_1\\ F_2\\ F_3\\ H_2\\ V_2\\ V_3 \end{bmatrix}
=
\begin{bmatrix} 0\\ -1000g\\ 0\\ 0\\ 0\\ 0 \end{bmatrix}$$

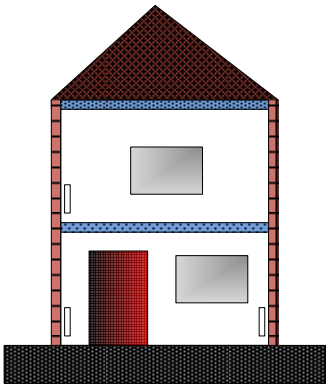
Thermal Resistance

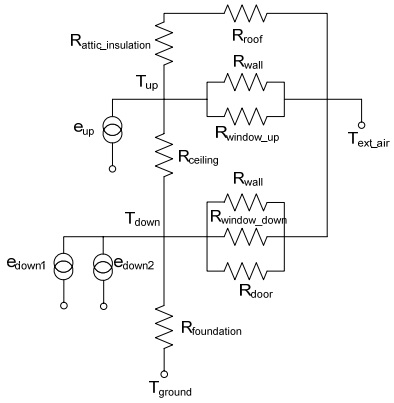
The steady-state heat conduction equation gives rise to the concept of *thermal resistance* (usually given as the *R-value*) and a very useful analogy between steady-state thermal problems and elementary electrical circuits made up of DC voltage sources and ideal resistors. Within this analogy the conductive heat flow through a material is equal to the temperature difference across the material divided by the thermal resistance.

$$\dot{Q} = \frac{T_1 - T_2}{R_{therm}}$$

Thermal Resistance

A highly simplified model and associated equivalent circuit are shown:





Thermal Resistance

Analysis yields:

$$e_{up} = \frac{T_{up} - T_{ext\_air}}{R_{attic\_insulation} + R_{roof}} + \frac{T_{up} - T_{ext}}{R_{parallel\_up}} + \frac{T_{up} - T_{down}}{R_{ceiling}}$$

$$e_{down1} + e_{down2} = \frac{T_{down} - T_{up}}{R_{ceiling}} + \frac{T_{down} - T_{ext}}{R_{parallel\_down}} + \frac{T_{down} - T_{ground}}{R_{foundation}}$$

where $$\frac{1}{R_{parallel\_up}} = \frac{1}{R_{wall}} + \frac{1}{R_{window\_up}}$$

and $$\frac{1}{R_{parallel\_down}} = \frac{1}{R_{wall}} + \frac{1}{R_{window\_down}} + \frac{1}{R_{door}}$$

Thermal Resistance

Note that thermal resistances in series add and thermal conductances (reciprocals of thermal resistance) in parallel add, as do heat sources.

Once all thermal resistances, external air temperature, ground temperature and heat generation by all sources are known these equations comprise two linear equations in two unknowns, the upstairs temperature T_{up} and the downstairs temperature T_{down}. Obviously this is not a serious model, but it is a serious method which can lead to rather good models.

Example (Kreyzig)

$$2w + x + 2y + z = 6$$
$$6w - 6x + 6y + 12z = 36$$
$$4w + 3x + 3y - 3z = -1$$
$$2w + 2x - y + z = 10$$

Step 1: eliminate w from second, third and fourth equations by subtracting multiples of first

$$-9x + 9z = 18$$
$$x - y - 5z = -13$$
$$x - 3y = 4$$

Example (Kreyzig)

$$-9x + 9z = 18$$
$$x - y - 5z = -13$$
$$x - 3y = 4$$

Step 2: eliminate x from second and third equations by subtracting multiples of first

$$-y - 4z = -11$$
$$-3y + z = 6$$

Example (Kreyzig)

$$-y-4z=-11$$
$$-3y+z=6$$

Step 3: eliminate y from second equation by subtracting multiple of first

$$13z=39$$

Final Step: Back-substitution

$$13z=39 \qquad z=3$$
$$-y-4z=-11 \qquad -y-4\cdot 3=-11 \qquad y=-1$$
$$-9x+9z=18 \qquad -9x+9\cdot 3=18 \qquad x=1$$
$$2w+x+2y+z=6 \qquad 2w+1+2\cdot(-1)+3=6 \qquad w=2$$

Gaussian Elimination

We began by employing the first equation to eliminate the first variable from the remaining equations. Of course it was possible to do this because the coefficient of the first variable in the first equation was non-zero, being 2. The coefficient of the first variable (or unknown) in the first equation is called the *pivotal coefficient*. The "first" equation is called the *pivotal equation*. We deduce that the first step of the procedure outlined above cannot be taken if the pivotal coefficient is zero. The situation is more serious. We face numerical difficulties even when the pivotal coefficient is non-zero.

Cautionary Example

$$0.003x_1 + 59.14x_2 = 59.17 \qquad x_1 = 10$$
$$5.291x_1 - 6.130x_2 = 46.78 \qquad x_2 = 1$$

Step 1: eliminate x_1 from second equation by subtracting multiple of first, work with 4 digit precision

$$-6.130 - \left(\frac{5.291}{0.003}\right)59.14 = -6.130 - (1764 \cdot 59.14)$$
$$= -6.130 - 104300 = -104300$$
$$46.78 - \left(\frac{5.291}{0.003}\right)59.17 = 46.78 - (1764 \cdot 59.17)$$
$$= 46.78 - 104400 = -104400$$

Cautionary Example

Giving: $-104300x_2 = -104400$

Final step: Back Substitution

$$x_2 = \frac{-104400}{-104300} = 1.001$$

$$0.003x_1 + (59.14 \cdot 1.001) = 59.17$$

$$x_1 = \frac{59.17 - (59.14 \cdot 1.001)}{0.003} = \frac{59.17 - 59.20}{0.003}$$
$$= \frac{-0.03}{0.003} = -10$$

which is completely wrong.

Gaussian Elimination

In the Kreyzig example we began by employing the first equation to eliminate the first variable from the remaining equations, i.e. we took the first equation to be the pivotal equation and w to be the first variable. It is obvious that we could have made different choices. We could have used the fourth equation to eliminate the first variable for example, or we could have begun by eliminating the second variable. The cautionary example has revealed that not only do we have choices, some may be better than others.

Gaussian Elimination

An algorithm should be clear. We should have a means of deciding in what order we should eliminate the variables and in what order we should employ the equations to achieve this elimination. The variation upon the basic Gaussian elimination procedure which involves making good decisions in both of these regards is called *pivoting*. As we have seen, a small pivotal coefficient can lead to problems when numerical precision is finite (as of course it will be). One approach to pivoting is to look for the coefficient having the largest absolute value and choose the equation and variable order so that this coefficient becomes the pivotal coefficient.

Example (Kreyzig + pivoting)

$$2w + x + 2y + z = 6$$
$$6w - 6x + 6y + 12z = 36$$
$$4w + 3x + 3y - 3z = -1$$
$$2w + 2x - y + z = 10$$

Step 1 pivot: Identify coefficient 12 as having largest absolute value. Reorder

$$12z + 6w - 6x + 6y = 36$$
$$z + 2w + x + 2y = 6$$
$$-3z + 4w + 3x + 3y = -1$$
$$z + 2w + 2x - y = 10$$

Example (Kreyzig + pivoting)

$$12z + 6w - 6x + 6y = 36$$
$$z + 2w + x + 2y = 6$$
$$-3z + 4w + 3x + 3y = -1$$
$$z + 2w + 2x - y = 10$$

Step 1: Eliminate z from second,third and fourth equations by subtracting multiples of first equation

$$1.5w + 1.5x + 1.5y = 3$$
$$5.5w + 1.5x + 4.5y = 8$$
$$1.5w + 2.5x - 1.5y = 7$$

Example (Kreyzig + pivoting)

$$1.5w + 1.5x + 1.5y = 3$$
$$5.5w + 1.5x + 4.5y = 8$$
$$1.5w + 2.5x - 1.5y = 7$$

Step 2 pivot: Identify coefficient 5.5 as having largest absolute value. Reorder

$$5.5w + 1.5x + 4.5y = 8$$
$$1.5w + 1.5x + 1.5y = 3$$
$$1.5w + 2.5x - 1.5y = 7$$

Example (Kreyzig + pivoting)

$$5.5w + 1.5x + 4.5y = 8$$
$$1.5w + 1.5x + 1.5y = 3$$
$$1.5w + 2.5x - 1.5y = 7$$

Step 2: Eliminate w from second and third equation by subtracting multiples of first equation

$$\tfrac{12}{11}x + \tfrac{3}{11}y = \tfrac{9}{11}$$
$$\tfrac{23}{11}x - \tfrac{30}{11}y = \tfrac{53}{11}$$

Example (Kreyzig + pivoting)

$$\frac{12}{11}x + \frac{3}{11}y = \frac{9}{11}$$
$$\frac{23}{11}x - \frac{30}{11}y = \frac{53}{11}$$

Step 3 pivot: Identify coefficient -30/11 as largest absolute value. Reorder

$$-\frac{30}{11}y + \frac{23}{11}x = \frac{53}{11}$$
$$\frac{3}{11}y + \frac{12}{11}x = \frac{9}{11}$$

Example (Kreyzig + pivoting)

$$-\frac{30}{11}y + \frac{23}{11}x = \frac{53}{11}$$
$$\frac{3}{11}y + \frac{12}{11}x = \frac{9}{11}$$

Step 3: Eliminate y from second equation by subtracting multiple of first $\frac{14.3}{11}x = \frac{14.3}{11}$

Final step: Back-substitution

$\frac{14.3}{11}x = \frac{14.3}{11}$ $x = 1$ $-\frac{30}{11}y + \frac{23}{11} = \frac{53}{11}$ $y = -1$

$5.5w + 1.5 - 4.5 = 8$ $w = 2$

$12z + 12 - 6 - 6 = 36$ $z = 3$

Example (Kreyzig + pivoting)

An additional point can be noted. When we arrived at the system of equations

$$5.5w + 1.5x + 4.5y = 8$$
$$1.5w + 1.5x + 1.5y = 3$$
$$1.5w + 2.5x - 1.5y = 7$$

and even more so at the system of equations

$$\tfrac{12}{11}x + \tfrac{3}{11}y = \tfrac{9}{11}$$
$$\tfrac{23}{11}x - \tfrac{30}{11}y = \tfrac{53}{11}$$

the system was practically begging to be *scaled*.

Example (Kreyzig + pivoting)

As it is more extremely the case consider the system of equations

$$\tfrac{12}{11}x + \tfrac{3}{11}y = \tfrac{9}{11}$$
$$\tfrac{23}{11}x - \tfrac{30}{11}y = \tfrac{53}{11}$$

We can multiply the first equation on both sides by 11/3 and the second on both sides by 11. This yields a different, simpler system of equations *having the same solutions*

$$4x + y = 3$$
$$23x - 30y = 53$$

Scaling

This type of scaling is called *row scaling*. It has no impact at all on the solution but it can have a very beneficial impact on the numerical effort involved in determining that solution to a given desired level of precision.

Of course we could also *column scale* where we replace a variable or unknown by a scalar times itself. This type of scaling *does* have an impact on the solution and we must bear that in mind in order to appropriately interpret the answer obtained.

Again it can have a big impact on numerical effort.

We already saw this type of scaling before.

Gaussian Elimination

In practice the preferred algorithm for solving systems of linear equations is to apply Gaussian elimination (also called Gauss elimination) with pivoting and scaling. The pivoting which has been described above, namely *look for the coefficient of largest absolute value in the remaining system of equations* is called *complete* or *total pivoting* and is relatively numerically intensive. A simpler method is to *look for the coefficient of the first remaining unknown of largest absolute value* and reorder equations only so that this becomes the pivotal coefficient. This is called *partial pivoting*. The perceived wisdom is that it works sufficiently well in most cases that the extra numerical effort in implementing total pivoting is not justified by the resulting improvement in performance.

Example (Kreyzig + partial pivoting)

$$2w + x + 2y + z = 6$$
$$6w - 6x + 6y + 12z = 36$$
$$4w + 3x + 3y - 3z = -1$$
$$2w + 2x - y + z = 10$$

Step 1 partial pivot: Identify coefficient 6 as largest absolute value of coefficient of first unknown w.

Reorder

$$6w - 6x + 6y + 12z = 36$$
$$2w + x + 2y + z = 6$$
$$4w + 3x + 3y - 3z = -1$$
$$2w + 2x - y + z = 10$$

Section: 3.2 Systems of linear equations

Example (Kreyzig + partial pivoting)

$$6w - 6x + 6y + 12z = 36$$
$$2w + x + 2y + z = 6$$
$$4w + 3x + 3y - 3z = -1$$
$$2w + 2x - y + z = 10$$

Step 1: Eliminate w from second,third and fourth equations by subtracting multiples of first equation

$$3x - 3z = -6$$
$$7x - y - 11z = -25$$
$$4x - 3y - 3z = -2$$

Example (Kreyzig + partial pivoting)

$$3x - 3z = -6$$
$$7x - y - 11z = -25$$
$$4x - 3y - 3z = -2$$

Step 2 partial pivot: Identify coefficient 7 as largest absolute value of coefficient of first unknown x.
Reorder

$$7x - y - 11z = -25$$
$$3x - 3z = -6$$
$$4x - 3y - 3z = -2$$

Section: 3.2 Systems of linear equations

Example (Kreyzig + partial pivoting)

$$7x - y - 11z = -25$$
$$3x - 3z = -6$$
$$4x - 3y - 3z = -2$$

Step 2: Eliminate x from second and third equations by subtracting multiples of first equation

$$\tfrac{3}{7}y + \tfrac{12}{7}z = \tfrac{33}{7}$$
$$-\tfrac{17}{7}y + \tfrac{23}{7}z = \tfrac{86}{7}$$

Example (Kreyzig + partial pivoting)

$$\tfrac{3}{7}y + \tfrac{12}{7}z = \tfrac{33}{7}$$
$$-\tfrac{17}{7}y + \tfrac{23}{7}z = \tfrac{86}{7}$$

Step 3 partial pivot: Identify coefficient -17/7 as largest absolute value of coefficient of first unknown y. Reorder

$$-\tfrac{17}{7}y + \tfrac{23}{7}z = \tfrac{86}{7}$$
$$\tfrac{3}{7}y + \tfrac{12}{7}z = \tfrac{33}{7}$$

Example (Kreyzig + partial pivoting)

$$-\tfrac{17}{7}y + \tfrac{23}{7}z = \tfrac{86}{7}$$
$$\tfrac{3}{7}y + \tfrac{12}{7}z = \tfrac{33}{7}$$

Step 3: Eliminate y from second equation by subtracting multiples of first equation $\tfrac{39}{17}z = \tfrac{117}{17}$

Final step: Back-substitution

$\tfrac{39}{17}z = \tfrac{117}{17}$ $z = 3$ $-\tfrac{17}{7}y + \tfrac{23\times 3}{7} = \tfrac{86}{7}$ $y = -1$

$7x + 1 - 11\times 3 = -25$ $x = 1$

$6w - 6 - 6 + 12\times 3 = 36$ $w = 2$

Matrix Form

It is common from an algorithmic programming point of view to express the steps involved in and the final effect of Gaussian elimination in matrix terms. This will also assist in developing Matlab code to implement Gaussian elimination. We will discuss by considering Kreyzig's example.

$$
\begin{aligned}
2w + x + 2y + z &= 6 \\
6w - 6x + 6y + 12z &= 36 \\
4w + 3x + 3y - 3z &= -1 \\
2w + 2x - y + z &= 10
\end{aligned}
$$

Matrix Form

First write the equations in matrix-vector form:

$$
\begin{bmatrix} 2 & 1 & 2 & 1 \\ 6 & -6 & 6 & 12 \\ 4 & 3 & 3 & -3 \\ 2 & 2 & -1 & 1 \end{bmatrix} \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 6 \\ 36 \\ -1 \\ 10 \end{bmatrix}
$$

We store the matrix of coefficients and the vector of constant values

$$
A = \begin{bmatrix} 2 & 1 & 2 & 1 \\ 6 & -6 & 6 & 12 \\ 4 & 3 & 3 & -3 \\ 2 & 2 & -1 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} 6 \\ 36 \\ -1 \\ 10 \end{bmatrix}
$$

Matrix Form: Matlab

Create matrices **A** and **b** in Matlab using the list method:

```
>> A = [2 1 2 1; 6 -6 6 12;4 3 3 -3;2 2 -1 1];   create matrix A
>> b = [6; 36; -1; 10];                          create vector b
```


Matrix Form

Step 1: pivot, i.e. swap rows 1 and 2, swap columns 1 and 4. In matrix terms the row swap is achieved by premultiplying by:

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

the column swap is achieved by postmultiplying by:

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Note that each of these matrices is a *permutation matrix* and when squared is the identity.

Matrix Form

We obtain:

$$\begin{bmatrix}0&1&0&0\\1&0&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}2&1&2&1\\6&-6&6&12\\4&3&3&-3\\2&2&-1&1\end{bmatrix}\begin{bmatrix}0&0&0&1\\0&1&0&0\\0&0&1&0\\1&0&0&0\end{bmatrix}\begin{bmatrix}0&0&0&1\\0&1&0&0\\0&0&1&0\\1&0&0&0\end{bmatrix}\begin{bmatrix}w\\x\\y\\z\end{bmatrix}=\begin{bmatrix}0&1&0&0\\1&0&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}6\\36\\-1\\10\end{bmatrix}$$

$$\begin{bmatrix}12&-6&6&6\\1&1&2&2\\-3&3&3&4\\1&2&-1&2\end{bmatrix}\begin{bmatrix}z\\x\\y\\w\end{bmatrix}=\begin{bmatrix}36\\6\\-1\\10\end{bmatrix}$$

$$A_{01}=\begin{bmatrix}12&-6&6&6\\1&1&2&2\\-3&3&3&4\\1&2&-1&2\end{bmatrix},\quad b_{01}=\begin{bmatrix}36\\6\\-1\\10\end{bmatrix}$$

Matrix Form: Matlab

The first part of step 1 involves finding the pivotal coefficient. The **max** command is very useful in this regard:

```
>> [Y,I] = max(abs(A))        find pivotal coefficient in matrix A
Y =
    6    6    6   12
I =
    2    2    2    2
```

This shows for example that the element of largest modulus in the first column of **A** has modulus 6 and is the second element.

Matrix Form: Matlab

A second application of the **max** command gives:

```
>>  [Y1,I1] = max(Y)   find pivotal coefficient in matrix A
Y1 =
   12
I1 =
    4
```

This shows for example that the element of largest modulus in **A** has modulus 12 and is in the fourth column, i.e. the pivotal element of **A** is in column I1 and row I(I1), i.e. column 4 and row 2. Hence it is a_{24}.

Section: 3.3 Systems of linear equations

Matrix Form: Matlab

Accordingly pivoting requires that I move column I1 = 4 to column 1. The simplest way to achieve this is to swap columns 1 and 4. Pivoting also requires that move row I(I1) = 2 to row 1. Again the simplest way to do this is to swap row 1 and 2.

So I wish to swap row 1 and row 2, and to swap column 1 and column 4.

Matrix Form: Matlab

There is a nice method in Matlab for swapping rows or swapping columns. To swap rows 1 and 2 using this method:

```
>> p = [2; 1; 3; 4];      vector p equals vector [1 2 3 4]' with rows 1 and 2 swapped
>> A01 = A(p,:);     A01 is matrix A with rows 1 and 2 swapped
```

Similarly to swap columns 1 and 4:

```
>> q = [4; 2; 3; 1];      vector q equals vector [1 2 3 4]' with rows 1 and 4 swapped
>> A01 = A01(:,q);     A01 is old matrix A01 with columns 1 and 4 swapped
```


Matrix Form: Matlab

Even better, we can do both swaps at the same time:

```
>> p = [2; 1; 3; 4];      vector p equals vector [1 2 3 4]' with rows 1 and 2 swapped
>> q = [4; 2; 3; 1];      vector q equals vector [1 2 3 4]' with rows 1 and 4 swapped
>> A01 = A(p,q)     A01 is matrix A with rows 1 and 2 swapped and with columns 1 and 4 swapped
A01 =
   12   -6    6    6
    1    1    2    2
   -3    3    3    4
    1    2   -1    2
```

Matrix Form: Matlab

Swapping rows 1 and 2 corresponded to swapping the order of the equations, so we will have to similarly swap the element of the vector of constants **b**. Given that vector p has already been created this is simple:

```
>>  bo1 = b(p);     b01 is vector b with rows 1 and 2 swapped
bo1 =
   36
    6
   -1
   10
```


Matrix Form: Matlab

Alternatively, and there is significant merit in this approach, we can generate the required matrices for pre- and post-multiplication. The first is the identity matrix with rows 1 and 2 swapped. Given that we know how to swap rows the following code generates this matrix:

```
>>  J = eye(4);        generate 4X4 identity matrix
>>  p = [2; 1; 3; 4];       vector p equals vector [1 2 3 4]' with rows 1
and 2 swapped
>>  P1 = J(p,:);     P1 is required pre-multiplication matrix
```

Matrix Form: Matlab

The post-multiplication matrix is the identity matrix with columns 1 and 4 swapped.

```
>> q = [4; 2; 3; 1];      vector q equals vector [1 2 3 4]' with rows 1 and 4 swapped
>> Q1 = J(:,q);     Q1 is required post-multiplication matrix
```

It is now relatively simple to generate A_{01} and b_{01}.

```
>> A01 = P1*A*Q1;
>> b01 = P1*b;
```


Matrix Form

Step 1: subtract multiples of first equation from other equations, i.e. premultiply both matrix of coefficients and vector of constants by

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ -\frac{1}{12} & 1 & 0 & 0 \\ \frac{3}{12} & 0 & 1 & 0 \\ -\frac{1}{12} & 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 12 & -6 & 6 & 6 \\ 0 & 1.5 & 1.5 & 1.5 \\ 0 & 1.5 & 4.5 & 5.5 \\ 0 & 2.5 & -1.5 & 1.5 \end{bmatrix} \begin{bmatrix} z \\ x \\ y \\ w \end{bmatrix} = \begin{bmatrix} 36 \\ 3 \\ 8 \\ 7 \end{bmatrix}$$

$$A_1 = \begin{bmatrix} 12 & -6 & 6 & 6 \\ 0 & 1.5 & 1.5 & 1.5 \\ 0 & 1.5 & 4.5 & 5.5 \\ 0 & 2.5 & -1.5 & 1.5 \end{bmatrix}, \quad b_1 = \begin{bmatrix} 36 \\ 3 \\ 8 \\ 7 \end{bmatrix}$$

Matrix Form

Note that the pre-multiplication matrix has the form:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ -\frac{1}{12} & 1 & 0 & 0 \\ \frac{3}{12} & 0 & 1 & 0 \\ -\frac{1}{12} & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} - \frac{1}{12} \begin{bmatrix} 0 \\ 1 \\ -3 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}$$

Matrix Form: Matlab

There are a lot of ways to achieve this in Matlab but the simplest is probably to just generate the matrix by employing the decomposition presented in the previous slide:

```
>>  L1 = eye(4) – ([0;A01([2:4],1)]*[1 zeros(1,3)])/A01(1,1);
L1 =
    1.0000         0         0         0
   -0.0833    1.0000         0         0
    0.2500         0    1.0000         0
   -0.0833         0         0    1.0000
```

Matrix Form: Matlab

```
>> A1 = L1*A01
A1 =
   12.0000   -6.0000    6.0000    6.0000
         0    1.5000    1.5000    1.5000
         0    1.5000    4.5000    5.5000
         0    2.5000   -1.5000    1.5000
>> b1 = L1*b01
b1 =
   36
    3
    8
    7
```


Matrix Form

Step 2: pivot, i.e. swap rows 2 and 3, swap columns 2 and 4. In matrix terms the row swap is achieved by premultiplying by:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

the column swap is achieved by postmultiplying by:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

Again each of these matrices squared is the identity.

Matrix Form: Matlab

Again the first part of step 2 involves finding the pivotal coefficient. In determining the coefficient on this second step *we must ignore* row 1 and column 1.

```
>> [Y,I] = max(abs(A1(2:4,2:4)))      find pivotal coefficient in
matrix A1, ignoring first row and column
Y =
    2.5000    4.5000    5.5000
I =
    3     2     2
```


Matrix Form: Matlab

A second application of the **max** command gives:

```
>> [Y1,I1] = max(Y)   find pivotal coefficient in matrix A1,
ignoring first row and column
Y1 =
    5.5000
I1 =
     3
```

The pivotal element of **A1** (ignoring first row and column) is in column I1 and row I(I1), i.e. column 3 and row 2.

Matrix Form: Matlab

It seems that the pivotal element is in column 3 and row 2. But this is column 3 and row 2 of a (3x3 matrix which was obtained from **A1** by ignoring or elimination the first row and column). The second row of this matrix is the *third* row of **A1**. The third column of this matrix is the *fourth* column of **A1**. Hence in terms of where it lies in matrix **A1** the pivotal element for step 2 is in column 4 and row 3, i.e. it is $(A_1)_{34}$ which is readily seen to be 5.5. We must make row and column swaps to move this pivotal coefficient to row 2 and column 2, i.e. swap rows 2 and 3 and swap columns 2 and 4.

Matrix Form

We obtain:

$$\begin{bmatrix}1&0&0&0\\0&0&1&0\\0&1&0&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}12&-6&6&6\\0&1.5&1.5&1.5\\0&1.5&4.5&5.5\\0&2.5&-1.5&1.5\end{bmatrix}\begin{bmatrix}1&0&0&0\\0&0&0&1\\0&0&1&0\\0&1&0&0\end{bmatrix}\begin{bmatrix}1&0&0&0\\0&0&0&1\\0&0&1&0\\0&1&0&0\end{bmatrix}\begin{bmatrix}z\\x\\y\\w\end{bmatrix}=\begin{bmatrix}1&0&0&0\\0&0&1&0\\0&1&0&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}36\\3\\8\\7\end{bmatrix}$$

$$\begin{bmatrix}12&6&6&-6\\0&5.5&4.5&1.5\\0&1.5&1.5&1.5\\0&1.5&-1.5&2.5\end{bmatrix}\begin{bmatrix}z\\w\\y\\x\end{bmatrix}=\begin{bmatrix}36\\8\\3\\7\end{bmatrix}$$

$$A_{02}=\begin{bmatrix}12&6&6&-6\\0&5.5&4.5&1.5\\0&1.5&1.5&1.5\\0&1.5&-1.5&2.5\end{bmatrix},\quad b_{02}=\begin{bmatrix}36\\8\\3\\7\end{bmatrix}$$

Matrix Form: Matlab

So we swap at the same time:

```
>> p = [1; 3; 2; 4];      vector p equals vector [1 2 3 4]' with rows 2 and 3 swapped

>> q = [1; 4; 3; 2];      vector q equals vector [1 2 3 4]' with rows 2 and 4 swapped

>> A02 = A1(p,q)    A02 is matrix A1 with rows 2 and 3 swapped and with columns 2 and 4 swapped

A02 =

   12.0000    6.0000    6.0000   -6.0000
         0    5.5000    4.5000    1.5000
         0    1.5000    1.5000    1.5000
         0    1.5000   -1.5000    2.5000
```


Matrix Form: Matlab

Again swapping rows 2 and 3 corresponded to swapping the order of the equations, so we will have to similarly swap the element of the vector of constants **b1**. Given that vector p has already been created this is simple:

```
>> bo2 = b1(p);    b02 is vector b1 with rows 2 and 3 swapped

bo2 =

   36
    8
    3
    7
```

Matrix Form: Matlab

Alternatively, and preferably again, we can generate the required matrices for pre- and post-multiplication. The first is the identity matrix with rows 2 and 3 swapped:

```
>> J = eye(4);        generate 4 by 4 identity matrix
>> p = [1; 3; 2; 4];      vector p equals vector [1 2 3 4]' with rows 2 and 3 swapped
>> P2 = J(p,:);     P2 is required pre-multiplication matrix
```


Matrix Form: Matlab

The post-multiplication matrix is the identity matrix with columns 2 and 4 swapped.

```
>> q = [1; 4; 3; 2];      vector q equals vector [1 2 3 4]' with rows 2 and 4 swapped
>> Q2 = J(:,q);     Q2 is required post-multiplication matrix
```

It is now relatively simple to generate A_{01} and b_{01}.

```
>> A02 = P2*A1*Q2;
>> b02 = P2*b1;
```

Matrix Form

Step 2: subtract multiples of second equation from third and fourth equations, i.e. premultiply both matrix of coefficients and vector of constants by

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -\frac{1.5}{5.5} & 1 & 0 \\ 0 & -\frac{1.5}{5.5} & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 12 & 6 & 6 & -6 \\ 0 & 5.5 & 4.5 & 1.5 \\ 0 & 0 & \frac{3}{11} & \frac{12}{11} \\ 0 & 0 & -\frac{30}{11} & \frac{23}{11} \end{bmatrix} \begin{bmatrix} z \\ w \\ y \\ x \end{bmatrix} = \begin{bmatrix} 36 \\ 8 \\ \frac{9}{11} \\ \frac{53}{11} \end{bmatrix}$$

$$A_2 = \begin{bmatrix} 12 & 6 & 6 & -6 \\ 0 & 5.5 & 4.5 & 1.5 \\ 0 & 0 & \frac{3}{11} & \frac{12}{11} \\ 0 & 0 & -\frac{30}{11} & \frac{23}{11} \end{bmatrix}, \quad b_2 = \begin{bmatrix} 36 \\ 8 \\ \frac{9}{11} \\ \frac{53}{11} \end{bmatrix}$$

Matrix Form

As before the pre-multiplication matrix permits the following decomposition:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -\frac{1.5}{5.5} & 1 & 0 \\ 0 & -\frac{1.5}{5.5} & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} - \frac{1}{5.5} \begin{bmatrix} 0 \\ 0 \\ 1.5 \\ 1.5 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}$$

Matrix Form: Matlab

Again, based on previous decomposition, generate the matrix:

```
>> L2 = eye(4) – ([0;0;A02([3:4],2)]*[0 1 zeros(1,2)])/A02(2,2);
L2 =
    1.0000         0         0         0
         0    1.0000         0         0
         0   -0.2727    1.0000         0
         0   -0.2727         0    1.0000
```


Matrix Form: Matlab

```
>> A2 = L2*A02
A2 =
   12.0000    6.0000    6.0000   -6.0000
         0    5.5000    4.5000    1.5000
         0         0    0.2727    1.0909
         0         0   -2.7273    2.0909
>> b2 = L2*b02
b2 =
   36.0000
    8.0000
    0.8182
    4.8182
```

Matrix Form

Step 3: pivot, i.e. swap rows 3 and 4, leave columns alone. In matrix terms the row swap is achieved by premultiplying by

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Again this matrix squared is the identity.

Matrix Form: Matlab

The first part of step 3 involves finding the pivotal coefficient. In determining the coefficient on this second step *we must ignore* rows 1, 2 and columns 1, 2.

```
>> [Y,I] = max(abs(A2(3:4,3:4)))      find pivotal coefficient in
matrix A2, ignoring first two rows and columns
Y =
   2.7273    2.0909
I =
   2     2
```

Matrix Form: Matlab

A second application of the **max** command gives:

```
>> [Y1,I1] = max(Y)   find pivotal coefficient in matrix A2,
ignoring first two rows and columns
Y1 =
   2.7273
I1 =
    1
```

The pivotal element of **A2** (ignoring first two rows and columns) is in column I1 and row I(I1), i.e. column 1 and row 2.

Matrix Form: Matlab

It seems that the pivotal element is in column 1 and row 2. But this is column 1 and row 2 of a (2x2 matrix which was obtained from **A2** by ignoring or elimination the first two rows and columns). The second row of this matrix is the *fourth* row of **A2**. The first column of this matrix is the *third* column of **A2**. Hence in terms of where it lies in matrix **A2** the pivotal element for step 3 is in column 3 and row 4, i.e. it is $(A_2)_{43}$ which is readily seen to be -2.7273. We must make row and column swaps to move this pivotal coefficient to row 3 and column 3, i.e. swap rows 3 and 4 and leave columns unchanged.

Matrix Form

We obtain:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 12 & 6 & 6 & -6 \\ 0 & 5.5 & 4.5 & 1.5 \\ 0 & 0 & \frac{3}{11} & \frac{12}{11} \\ 0 & 0 & -\frac{30}{11} & \frac{23}{11} \end{bmatrix} \begin{bmatrix} z \\ w \\ y \\ x \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 36 \\ 8 \\ \frac{9}{11} \\ \frac{53}{11} \end{bmatrix}$$

$$\begin{bmatrix} 12 & 6 & 6 & -6 \\ 0 & 5.5 & 4.5 & 1.5 \\ 0 & 0 & -\frac{30}{11} & \frac{23}{11} \\ 0 & 0 & \frac{3}{11} & \frac{12}{11} \end{bmatrix} \begin{bmatrix} z \\ w \\ y \\ x \end{bmatrix} = \begin{bmatrix} 36 \\ 8 \\ \frac{53}{11} \\ \frac{9}{11} \end{bmatrix}$$

$$A_{03} = \begin{bmatrix} 12 & 6 & 6 & -6 \\ 0 & 5.5 & 4.5 & 1.5 \\ 0 & 0 & -\frac{30}{11} & \frac{23}{11} \\ 0 & 0 & \frac{3}{11} & \frac{12}{11} \end{bmatrix}, \quad b_{03} = \begin{bmatrix} 36 \\ 8 \\ \frac{53}{11} \\ \frac{9}{11} \end{bmatrix}$$

Matrix Form: Matlab

```
>> p = [1; 2; 4; 3];      vector p equals vector [1 2 3 4]' with rows 3 and 4 swapped
>> q = [1; 2; 3; 4];      vector q equals vector [1 2 3 4]' with rows unchanged
>> P3 = J(p,:);           P3 is required pre-multiplication matrix
>> Q3 = J(:,q);           Q3 is required post-multiplication matrix, actually the identity matrix in this case
```

```
>> A03 = P3*A2*Q3;
>> b03 = P3*b2;
```

Matrix Form

Step 3: subtract multiples of third equation from fourth equation, i.e. premultiply both matrix of coefficients and vector of constants by

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{10} & 1 \end{bmatrix} \qquad \begin{bmatrix} 12 & 6 & 6 & -6 \\ 0 & 5.5 & 4.5 & 1.5 \\ 0 & 0 & -\frac{30}{11} & \frac{23}{11} \\ 0 & 0 & 0 & \frac{14.3}{11} \end{bmatrix} \begin{bmatrix} z \\ w \\ y \\ x \end{bmatrix} = \begin{bmatrix} 36 \\ 8 \\ \frac{53}{11} \\ \frac{14.3}{11} \end{bmatrix}$$

$$A_3 = \begin{bmatrix} 12 & 6 & 6 & -6 \\ 0 & 5.5 & 4.5 & 1.5 \\ 0 & 0 & -\frac{30}{11} & \frac{23}{11} \\ 0 & 0 & 0 & \frac{14.3}{11} \end{bmatrix}, \quad b_3 = \begin{bmatrix} 36 \\ 8 \\ \frac{53}{11} \\ \frac{14.3}{11} \end{bmatrix}$$

Matrix Form

As before the pre-multiplication matrix permits the following decomposition:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{10} & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} - \frac{1}{-2.7273} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0.2727 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix}$$

Matrix Form: Matlab

Again, based on previous decomposition, generate the matrix:

```
>>  L3 = eye(4) – ([0;0;0;A03(4,3)]*[0 0 1 0])/A03(3,3);
L3 =
    1.0000         0         0         0
         0    1.0000         0         0
         0         0    1.0000         0
         0         0    0.1000    1.0000
```


Matrix Form: Matlab

```
>>  A3 = L3*A03
A3 =
   12.0000    6.0000    6.0000   -6.0000
         0    5.5000    4.5000    1.5000
    0.0000    0.0000   -2.7273    2.0909
    0.0000    0.0000   -0.0000    1.3000
 >>  b3 = L3*b03
b3 =
   36.0000
    8.0000
    4.8182
    1.3000
```

Matrix Form

The effect of the algorithm to date is apparent. The matrix of coefficients is brought into *upper triangular form*. Moreover a readback reveals that this has been achieved through pre-multiplying by relatively simple matrices, either permutation matrices or lower triangular matrices and post-multiplying by permutation matrices. The third pivot step, where the action of total pivoting was the same as that of partial pivoting, reveals that when one partial pivots no post-multiplication is required and the vector of unknowns is not reordered, which means we do not have to bother keeping track of its reordering. This explains more clearly the previous assertion that partial pivoting requires less numerical effort.

Matrix Form

The advantage of upper triangular form is that when the matrix of coefficients is in this form the theoretical difficulty of solving the system of equations is greatly reduced since the method of *back substitution* is applicable.

$$\begin{bmatrix} 12 & 6 & 6 & -6 \\ 0 & 5.5 & 4.5 & 1.5 \\ 0 & 0 & -\frac{30}{11} & \frac{23}{11} \\ 0 & 0 & 0 & \frac{14.3}{11} \end{bmatrix} \begin{bmatrix} z \\ w \\ y \\ x \end{bmatrix} = \begin{bmatrix} 36 \\ 8 \\ \frac{53}{11} \\ \frac{14.3}{11} \end{bmatrix} \qquad \begin{bmatrix} z \\ w \\ y \\ x \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \\ -1 \\ 1 \end{bmatrix}$$

Matrix Form: Matlab

```
>> vec = zeros(4,1)    initialise vector to store results
>> vec(4) = b3(4)/A3(4,4)
vec =
     0
     0
     0
     1.0000
```

```
>> vec(3) = (b3(3) – (A3(3,4)*vec(4)))/A3(3,3)
vec =
     0
     0
    -1.0000
     1.0000
```

Section: 3.3 Systems of linear equations

Matrix Form: Matlab

```
>> vec(2) = (b3(2) – (A3(2,4)*vec(4))-(A3(2,3)*vec(3)))/A3(2,2)
vec =
     0
     2.0000
    -1.0000
     1.0000
```

```
>> vec(1) = (b3(1) – (A3(1,4)*vec(4))-(A3(1,3)*vec(3))-
(A3(1,2)*vec(2)))/A3(1,1)
vec =
     3.0000
     2.0000
    -1.0000
     1.0000
```

Matrix Form: Matlab

The vector **vec** contains the answer, i.e. the value of all the variables. But they are not in the same order as originally. The post-multiplication by **Q1** changed the order, as did the post-multiplications by **Q2** and **Q3**. Accordingly the new vector of unkowns is:

$$Q_3Q_2Q_1\begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} z \\ w \\ y \\ x \end{bmatrix} \quad \text{giving} \quad \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} = Q_1Q_2Q_3\begin{bmatrix} z \\ w \\ y \\ x \end{bmatrix}$$

as each matrix **Q1**, **Q2**, **Q3** is its own inverse.

Matrix Form: Matlab

So finally, to recover the original values:

```
>> v = Q1*Q2*Q3*vec
v =
   2.0000        w = 2
   1.0000        x = 1
  -1.0000        y = -1
   3.0000        z = 3
```

Variations on Gaussian Elimination

If the matrix of coefficients has some simplified structure then Gaussian elimination may not be the most efficient method for solving the associated system of equations. For example, if the matrix of coefficients, A, is real and symmetric then there exists a more efficient method essentially due to Cholesky. Jordan introduced a variation, called *Gauss-Jordan elimination*, where the somewhat annoying and error-prone back-substitution can be avoided. The method is generally less efficient numerically however.

Iterative Method

Gaussian elimination belongs to the class of algorithms for solving systems of linear equations which are known as *direct methods*. There is a second class of algorithms known as *indirect* or *iterative methods*. The direct methods yield a solution to a predefined precision necessitating a predefined maximum amount of computation. Indirect methods on the other hand begin with an approximation and, just as in the case of the Newton-Raphson method for nonlinear equations, seek to produce a sequence of ever improving approximations. The computational effort required cannot be predetermined. One must simply persist until the required level of accuracy has been achieved.

Iterative Method

Iterative methods cannot compete with Gaussian elimination in general. However, when the matrix of coefficients is large, i.e. one has many equations in many unknowns, and when a good number of these coefficients are zero, Gaussian elimination becomes relatively cumbersome and iterative methods come into their own. Systems of equations of this kind are said to be *sparse*. So sparse means two things (i) a large number of equations and unknowns and (ii) a large number of zero coefficients. If sparse systems rarely or never occurred we would not care, but we shall see that methods for numerically solving other problems (Partial Differential Equations for example) lead to systems of equations which are indeed sparse.

Example (Kreyzig)

$$w - 0.25x - 0.25y = 50$$
$$-0.25w + x - 0.25z = 50$$
$$-0.25w + y - 0.25z = 25$$
$$-0.25x - 0.25y + z = 25$$

Rewrite

$$w = 0.25x + 0.25y + 50$$
$$x = 0.25w + 0.25z + 50$$
$$y = 0.25w + 0.25z + 25$$
$$z = 0.25x + 0.25y + 25$$

Example (Kreyzig)

Define an affine function in four variables

$$f\left(\begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix}\right) = \begin{bmatrix} 0 & 0.25 & 0.25 & 0 \\ 0.25 & 0 & 0 & 0.25 \\ 0.25 & 0 & 0 & 0.25 \\ 0 & 0.25 & 0.25 & 0 \end{bmatrix} \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} + \begin{bmatrix} 50 \\ 50 \\ 25 \\ 25 \end{bmatrix}$$

The equations take the form of a fixed point equation for an affine function. We have already seen how we go about solving this by iteration.

$$f\left(\begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix}\right) = \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix}$$

Example (Kreyzig)

$$\begin{bmatrix} w_{n+1} \\ x_{n+1} \\ y_{n+1} \\ z_{n+1} \end{bmatrix} = f\left(\begin{bmatrix} w_n \\ x_n \\ y_n \\ z_n \end{bmatrix}\right) = \begin{bmatrix} 0 & 0.25 & 0.25 & 0 \\ 0.25 & 0 & 0 & 0.25 \\ 0.25 & 0 & 0 & 0.25 \\ 0 & 0.25 & 0.25 & 0 \end{bmatrix} \begin{bmatrix} w_n \\ x_n \\ y_n \\ z_n \end{bmatrix} + \begin{bmatrix} 50 \\ 50 \\ 25 \\ 25 \end{bmatrix}$$

This type of iteration is sometimes called *Jacobi iteration*.

$$\begin{bmatrix} w_0 \\ x_0 \\ y_0 \\ z_0 \end{bmatrix} = \begin{bmatrix} 100 \\ 100 \\ 100 \\ 100 \end{bmatrix} \qquad \begin{bmatrix} w_1 \\ x_1 \\ y_1 \\ z_1 \end{bmatrix} = \begin{bmatrix} 100 \\ 100 \\ 75 \\ 75 \end{bmatrix} \qquad \begin{bmatrix} w_2 \\ x_2 \\ y_2 \\ z_2 \end{bmatrix} = \begin{bmatrix} 93.75 \\ 93.75 \\ 68.75 \\ 68.75 \end{bmatrix}$$

Example (Kreyzig)

$$\begin{bmatrix} w_3 \\ x_3 \\ y_3 \\ z_3 \end{bmatrix} = \begin{bmatrix} 90.625 \\ 90.625 \\ 65.625 \\ 65.625 \end{bmatrix} \quad \begin{bmatrix} w_4 \\ x_4 \\ y_4 \\ z_4 \end{bmatrix} = \begin{bmatrix} 89.0625 \\ 89.0625 \\ 64.0625 \\ 64.0625 \end{bmatrix} \quad \begin{bmatrix} w_5 \\ x_5 \\ y_5 \\ z_5 \end{bmatrix} = \begin{bmatrix} 88.2813 \\ 88.2813 \\ 63.2813 \\ 63.2813 \end{bmatrix}$$

$$\begin{bmatrix} w_6 \\ x_6 \\ y_6 \\ z_6 \end{bmatrix} = \begin{bmatrix} 87.8906 \\ 87.8906 \\ 62.8906 \\ 62.8906 \end{bmatrix} \quad \begin{bmatrix} w_7 \\ x_7 \\ y_7 \\ z_7 \end{bmatrix} = \begin{bmatrix} 87.6953 \\ 87.6953 \\ 62.6953 \\ 62.6953 \end{bmatrix}$$

Example (Kreyzig)

$$\begin{bmatrix} w_8 \\ x_8 \\ y_8 \\ z_8 \end{bmatrix} = \begin{bmatrix} 87.5977 \\ 87.5977 \\ 62.5977 \\ 62.5977 \end{bmatrix} \quad \begin{bmatrix} w_9 \\ x_9 \\ y_9 \\ z_9 \end{bmatrix} = \begin{bmatrix} 87.5488 \\ 87.5488 \\ 62.5488 \\ 62.5488 \end{bmatrix}$$

$$\begin{bmatrix} w_{10} \\ x_{10} \\ y_{10} \\ z_{10} \end{bmatrix} = \begin{bmatrix} 87.5244 \\ 87.5244 \\ 62.5244 \\ 62.5244 \end{bmatrix} \quad \begin{bmatrix} w_{11} \\ x_{11} \\ y_{11} \\ z_{11} \end{bmatrix} = \begin{bmatrix} 87.5122 \\ 87.5122 \\ 62.5122 \\ 62.5122 \end{bmatrix}$$

Example (Kreyzig)

$$\begin{bmatrix} w_{12} \\ x_{12} \\ y_{12} \\ z_{12} \end{bmatrix} = \begin{bmatrix} 87.5061 \\ 87.5061 \\ 62.5061 \\ 62.5061 \end{bmatrix} \qquad \begin{bmatrix} w_{13} \\ x_{13} \\ y_{13} \\ z_{13} \end{bmatrix} = \begin{bmatrix} 87.5031 \\ 87.5031 \\ 62.5031 \\ 62.5031 \end{bmatrix}$$

$$\begin{bmatrix} w_{14} \\ x_{14} \\ y_{14} \\ z_{14} \end{bmatrix} = \begin{bmatrix} 87.5015 \\ 87.5015 \\ 62.5015 \\ 62.5015 \end{bmatrix} \qquad \begin{bmatrix} w_{15} \\ x_{15} \\ y_{15} \\ z_{15} \end{bmatrix} = \begin{bmatrix} 87.5008 \\ 87.5008 \\ 62.5008 \\ 62.5008 \end{bmatrix}$$

Example (Kreyzig)

$$\begin{bmatrix} w_{16} \\ x_{16} \\ y_{16} \\ z_{16} \end{bmatrix} = \begin{bmatrix} 87.5004 \\ 87.5004 \\ 62.5004 \\ 62.5004 \end{bmatrix} \qquad \begin{bmatrix} w_{17} \\ x_{17} \\ y_{17} \\ z_{17} \end{bmatrix} = \begin{bmatrix} 87.5002 \\ 87.5002 \\ 62.5002 \\ 62.5002 \end{bmatrix}$$

It is clear that the iteration is converging and it is clear that it is converging to $w = x = 87.5$ and $y = z = 62.5$. It is equally clear that the rate of convergence is extremely poor.

Example (Kreyzig)

There is one simple idea which can (note I say *can* not *will*) be effective in improving convergence. We consider a different form of iteration where, if an updated approximation of an unknown is available it is used.

$$w_{n+1} = 0.25x_n + 0.25y_n + 50$$
$$x_{n+1} = 0.25w_{n+1} + 0.25z_n + 50$$
$$y_{n+1} = 0.25w_{n+1} + 0.25z_n + 25$$
$$z_{n+1} = 0.25x_{n+1} + 0.25y_{n+1} + 25$$

This type of iteration is called *Gauss-Seidel iteration*. The idea can also be applied to nonlinear iterations.

Example (Kreyzig)

$$\begin{bmatrix} w_0 \\ x_0 \\ y_0 \\ z_0 \end{bmatrix} = \begin{bmatrix} 100 \\ 100 \\ 100 \\ 100 \end{bmatrix} \quad \begin{bmatrix} w_1 \\ x_1 \\ y_1 \\ z_1 \end{bmatrix} = \begin{bmatrix} 100 \\ 100 \\ 75 \\ 68.75 \end{bmatrix} \quad \begin{bmatrix} w_2 \\ x_2 \\ y_2 \\ z_2 \end{bmatrix} = \begin{bmatrix} 93.75 \\ 90.625 \\ 65.625 \\ 64.0625 \end{bmatrix}$$

$$\begin{bmatrix} w_3 \\ x_3 \\ y_3 \\ z_3 \end{bmatrix} = \begin{bmatrix} 89.0625 \\ 88.2813 \\ 63.2813 \\ 62.8906 \end{bmatrix} \quad \begin{bmatrix} w_4 \\ x_4 \\ y_4 \\ z_4 \end{bmatrix} = \begin{bmatrix} 87.8906 \\ 87.6953 \\ 62.6953 \\ 62.5977 \end{bmatrix}$$

Example (Kreyzig)

$$\begin{bmatrix} w_5 \\ x_5 \\ y_5 \\ z_5 \end{bmatrix} = \begin{bmatrix} 87.5977 \\ 87.5488 \\ 62.5488 \\ 62.5244 \end{bmatrix} \qquad \begin{bmatrix} w_6 \\ x_6 \\ y_6 \\ z_6 \end{bmatrix} = \begin{bmatrix} 87.5244 \\ 87.5122 \\ 62.5122 \\ 62.5061 \end{bmatrix}$$

$$\begin{bmatrix} w_7 \\ x_7 \\ y_7 \\ z_7 \end{bmatrix} = \begin{bmatrix} 87.5061 \\ 87.5031 \\ 62.5031 \\ 62.5015 \end{bmatrix} \qquad \begin{bmatrix} w_8 \\ x_8 \\ y_8 \\ z_8 \end{bmatrix} = \begin{bmatrix} 87.5015 \\ 87.5008 \\ 62.5008 \\ 62.5004 \end{bmatrix}$$

Example (Kreyzig)

$$\begin{bmatrix} w_9 \\ x_9 \\ y_9 \\ z_9 \end{bmatrix} = \begin{bmatrix} 87.5004 \\ 87.5002 \\ 62.5002 \\ 62.5001 \end{bmatrix} \qquad \begin{bmatrix} w_{10} \\ x_{10} \\ y_{10} \\ z_{10} \end{bmatrix} = \begin{bmatrix} 87.5001 \\ 87.5000 \\ 62.5000 \\ 62.5000 \end{bmatrix}$$

We obtain approximately the same level of precision with approximately half the number of steps. There is undoubtedly an improvement in performance, although convergence remains a problem.

Caveat

The upward (vertical) velocity of a rocket is measured at three consective times, giving:

Time, t (sec)	Velocity, v (m/sec)
5	106.8
8	177.2
12	279.2

The velocity of the rocket is modelled by a quadratic equation: $v(t) = a_1 t^2 + a_2 t + a_3$

Determine appropriate values for the parameters a_1, a_2 and a_3.

Caveat

$$\begin{aligned} a_1(5)^2 + a_2(5) + a_3 &= 106.8 \\ a_1(8)^2 + a_2(8) + a_3 &= 177.2 \\ a_1(12)^2 + a_2(12) + a_3 &= 279.2 \end{aligned}$$

$$\begin{aligned} 25a_1 + 5a_2 + a_3 &= 106.8 \\ 64a_1 + 8a_2 + a_3 &= 177.2 \\ 144a_1 + 12a_2 + a_3 &= 279.2 \end{aligned}$$

Caveat

Rearrange:

$$a_1 = \left(\frac{106.8 - 5a_2 - a_3}{25}\right)$$
$$a_2 = \left(\frac{177.2 - 64a_1 - a_3}{8}\right)$$
$$a_3 = 279.2 - 144a_1 - 12a_2$$

Gauss-Seidel iteration:

$$a_1(n+1) = 4.272 - 0.2a_2(n) - 0.04a_3(n)$$
$$a_2(n+1) = 22.15 - 8a_1(n+1) - 0.125a_3(n)$$
$$a_3(n+1) = 279.2 - 144a_1(n+1) - 12a_2(n+1)$$

Caveat

Initial guess: $\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 5 \end{bmatrix}$

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 3.672 \\ -7.851 \\ -155.356 \end{bmatrix} \quad \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 12.0564 \\ -54.882 \\ -798.3431 \end{bmatrix}$$

After 6 iterations: $\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 3322.57 \\ -19049.48 \\ -249577.9 \end{bmatrix}$

Caveat

The components of a are simply getting bigger. They are not obviously converging to the correct answer which is:

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 0.2905 \\ 19.6905 \\ 1.0857 \end{bmatrix}$$

Evidently the Gauss-Seidel method has failed to yield the solution for this example. We wish to know what has gone wrong, whether we have predicted this problem and whether there is some method of avoiding it.

Caveat

Rearrange Gauss-Seidel iteration:

$$\begin{aligned} a_1(n+1) &= 4.272 - 0.2a_2(n) - 0.04a_3(n) \\ 8a_1(n+1) + a_2(n+1) &= 22.15 - 0.125a_3(n) \\ 144a_1(n+1) + 12a_2(n+1) + a_3(n+1) &= 279.2 \end{aligned}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 8 & 1 & 0 \\ 144 & 12 & 1 \end{bmatrix} \begin{bmatrix} a_1(n+1) \\ a_2(n+1) \\ a_3(n+1) \end{bmatrix} = \begin{bmatrix} 4.272 \\ 22.15 \\ 279.2 \end{bmatrix} - \begin{bmatrix} 0 & 0.2 & 0.04 \\ 0 & 0 & 0.125 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} a_1(n) \\ a_2(n) \\ a_3(n) \end{bmatrix}$$

These equations can be rewritten as:

$$L\vec{a}(n+1) = \vec{b} - U\vec{a}(n)$$

Caveat

$$L\vec{a}(n+1) = \vec{b} - U\vec{a}(n)$$

$$\vec{a}(n+1) = L^{-1}\vec{b} - L^{-1}U\vec{a}(n)$$

$$\vec{a}(1) = L^{-1}\vec{b} - L^{-1}U\vec{a}(0)$$

$$\vec{a}(2) = L^{-1}\vec{b} + \left(-L^{-1}U\right)L^{-1}\vec{b} + \left(L^{-1}U\right)^2\vec{a}(0)$$

$$\vec{a}(3) = L^{-1}\vec{b} + \left(-L^{-1}U\right)L^{-1}\vec{b} + \left(L^{-1}U\right)^2 L^{-1}\vec{b} + \left(-L^{-1}U\right)^3\vec{a}(0)$$

A reasonably obvious pattern has developed.

Caveat

$$\vec{a}(m) =$$
$$L^{-1}\vec{b} + \left(-L^{-1}U\right)L^{-1}\vec{b} + \left(-L^{-1}U\right)^2 L^{-1}\vec{b} + \cdots + \left(L^{-1}U\right)^{m-1} L^{-1}\vec{b} + \left(-L^{-1}U\right)^m\vec{a}(0)$$

It is apparent that the matrix $(-L^{-1}U)$ is playing a very significant role here. It is possible to establish that if a matrix is raised to higher and higher powers then it tends to become the zero matrix if all of its eigenvalues have modulus less than 1 and some of its elements tend to become very large if at least one eigenvalue has modulus exceeding 1.

Caveat

For the curve fitting problem:

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 8 & 1 & 0 \\ 144 & 12 & 1 \end{bmatrix} \qquad U = \begin{bmatrix} 0 & 0.2 & 0.04 \\ 0 & 0 & 0.125 \\ 0 & 0 & 0 \end{bmatrix}$$

$$-L^{-1}U = \begin{bmatrix} 0 & -0.2 & -0.04 \\ 0 & 1.6 & 0.195 \\ 0 & 9.6 & 3.42 \end{bmatrix}$$

and its eigenvalues are 0, 0.8668 and 4.1532. Of course the latter has modulus exceeding 1 and this is why the iteration above gives larger and larger values and fails to converge.

Caveat

So we know what went wrong, an eigenvalue of the critical matrix $-L^{-1}U$ has modulus exceeding 1. We could indeed have predicted that convergence problems would arise by having noted this fact at the outset. The one outstanding question is whether we can do anything about it. How does matrix $-L^{-1}U$ relate to the original problem?

$$\begin{aligned} 25a_1 + 5a_2 + a_3 &= 106.8 \\ 64a_1 + 8a_2 + a_3 &= 177.2 \\ 144a_1 + 12a_2 + a_3 &= 279.2 \end{aligned}$$

Caveat

Scale each equation so that diagonal elements are unity:

$$\begin{aligned} a_1 + 0.2a_2 + 0.04a_3 &= 4.272 \\ 8a_1 + a_2 + 0.125a_3 &= 22.15 \\ 144a_1 + 12a_2 + a_3 &= 279.2 \end{aligned}$$

These equations take the form:

$$(L+U)\vec{a} = \vec{b}$$

where L is lower triangular with, moreover, unit diagonal elements and U is upper triangular with, moreover, zero diagonal elements.

Caveat

$$\begin{aligned} 25a_1 + 5a_2 + a_3 &= 106.8 \\ 64a_1 + 8a_2 + a_3 &= 177.2 \\ 144a_1 + 12a_2 + a_3 &= 279.2 \end{aligned}$$

Now the order in which we place the equations has no bearing at all upon the solution. The following equations are equivalent:

$$\begin{aligned} 144a_1 + 12a_2 + a_3 &= 279.2 \\ 64a_1 + 8a_2 + a_3 &= 177.2 \\ 25a_1 + 5a_2 + a_3 &= 106.8 \end{aligned}$$

Caveat

$$\begin{aligned} 144a_1 + 12a_2 + a_3 &= 279.2 \\ 64a_1 + 8a_2 + a_3 &= 177.2 \\ 25a_1 + 5a_2 + a_3 &= 106.8 \end{aligned}$$

Scale:
$$\begin{aligned} a_1 + 0.0833a_2 + 0.0069a_3 &= 1.9389 \\ 8a_1 + a_2 + 0.125a_3 &= 22.15 \\ 25a_1 + 5a_2 + a_3 &= 106.8 \end{aligned}$$

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 8 & 1 & 0 \\ 25 & 5 & 1 \end{bmatrix} \qquad U = \begin{bmatrix} 0 & 0.0833 & 0.0069 \\ 0 & 0 & 0.125 \\ 0 & 0 & 0 \end{bmatrix}$$

Caveat

$$-L^{-1}U = \begin{bmatrix} 0 & -0.0833 & -0.0069 \\ 0 & 0.6664 & -0.0698 \\ 0 & -1.2495 & 0.5215 \end{bmatrix}$$

Eigenvalues: 0, 0.2899 and 0.8980.

By simply reordering the equations we can apply Gauss-Seidel iteration and see convergence.

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 5 \end{bmatrix} \qquad \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 1.7378 \\ 7.6226 \\ 25.2420 \end{bmatrix} \qquad \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 1.1298 \\ 9.9566 \\ 28.7728 \end{bmatrix}$$

Caveat

After 20 iterations:

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 0.3831 \\ 18.3210 \\ 5.6186 \end{bmatrix}$$

Convergence is very slow, but it does at least occur. In fact the largest modulus eigenvalue raised to the power of 20 is:

$$0.8980^{20} = 0.1163$$

This is not particularly small and this is the reason why convergence has been rather modest after 20 iterations.

Caveat

After 100 iterations:

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 0.2905 \\ 19.6903 \\ 1.0865 \end{bmatrix}$$

which is very close to the correct answer. So convergence has indeed occurred, albeit at a rather poor rate.

Note: Convergence

With regard to Gauss-Seidel iteration some authors like to quote a relatively simple condition which is a *sufficient* condition for the eigenvalues of $-L^{-1}U$ to have modulus less than 1. This condition is stated in terms of *diagonal dominance* or *row diagonal dominance* or *strict diagonal dominance* or *strict row diagonal dominance* depending on the author. Not only is it possible to deduce convergence without calculating the eigenvalues of $-L^{-1}U$, we do not even have to calculate $-L^{-1}U$ as the condition is stated directly in terms of the original matrix $A=L+U$.

Note: Convergence

The condition is that matrix $A = L+U$ should be *row diagonally dominant*, meaning that in each row the diagonal element has a modulus larger than the sum of the moduli of all of the remaining elements in that row. For example above:

$$\begin{aligned} a_1 + 0.2a_2 + 0.04a_3 &= 4.272 \\ 8a_1 + a_2 + 0.125a_3 &= 22.15 \\ 144a_1 + 12a_2 + a_3 &= 279.2 \end{aligned} \qquad A = L + U = \begin{bmatrix} 1 & 0.2 & 0.04 \\ 8 & 1 & 0.125 \\ 144 & 12 & 1 \end{bmatrix}$$

not diagonally dominant in second or third rows.

Note: Convergence

The Gauss-Seidel iteration, even when it converges, commonly does so rather slowly. There is a scheme called *relaxation* applicable to a wide variety of iterative schemes and, in particular, to the Gauss-Seidel iteration, which can lead to an acceleration in the convergence. Two versions of relaxation are available, *underrelaxation* and *overrelaxation*, with the latter being most commonly applied to affect an acceleration of the Gauss-Seidel convergence.

Advanced Methods

Gauss-Seidel iteration can be effective for solving sparse systems of linear equations, but it is not considered to be the best method available. The method of *Successive Over-Relaxation* (SOR), introduced independently by Young and Frankel, is recognised as being superior.

Section 3 - Conclusion

- Systems of linear equations commonly arise. The best general method for numerically solving such systems of equations is Gaussian elimination with scaling and partial pivoting.

- If the equations are large in number, have many variables and are sparse then an iterative procedure may be required for their solution since the back-substitution process can lead to propagation of errors.