Ordinary Differential Equations (ODEs)

1 Computer Simulations

Why is computation becoming so important in physics? One reason is that most of our analytical tools such as differential calculus are best suited to the analysis of linear problems.

For example, you probably have analyzed the motion of a particle attached to a spring by assuming a linear restoring force and solving Newton's second law of motion. In this case, a small change in the displacement of the particle leads to a small change in the force. However, many natural phenomena are nonlinear, and a small change in a variable might produce a large change in another. Because relatively few nonlinear problems can be solved by analytical methods, the computer gives us a new tool to explore nonlinear phenomena.

Another reason for the importance of computation is the growing interest in systems with many variables or with many degrees of freedom. Computer simulations are sometimes referred to as computer experiments because they share much in common with laboratory experiments.

The starting point of a computer simulation is the development of an idealized model of a physical system of interest. We then need to specify a procedure or algorithm for implementing the model on a computer and decide what quantities to measure. The results of a computer simulation can serve as a bridge between laboratory experiments and theoretical calculations. In some cases we can obtain essentially exact results by simulating an idealized model that has no laboratory counterpart.

The results of the idealized model can serve as a stimulus to the development of the theory. On the other hand, we sometimes can do simulations of a more realistic model than can be done theoretically, and hence make a more direct comparison with laboratory experiments. Computation has become a third way of doing physics and complements both theory and experiment.

Computer simulations, like laboratory experiments, are not substitutes for thinking, but are tools that we can use to understand natural phenomena. The goal of all our investigations of fundamental phenomena is to seek explanations of natural phenomena that can be stated concisely.

2 Differential equations

Differential equations play a central role in many subjects of physics. It is therefore clear that numerical solutions to differential equations are a central issue in computational physics.

Many so called textbook cases studied in mathematical courses or physics are not very often found in real physics life. The simple models we can solve using math have in many cases many assumptions that make the explicit solution

feasible. If we would put all our physics knowledge into our problem, it would be very difficult to solve.

For example, we know very well the physics of wave propagation in a perfectly elastic medium. Once we put 3D structure, the mathematical solution because increasingly cumbersome. Think of adding anelasticity or anisotropy. This problems cannot be solved directly and one escape is to ask for help from the computer.

Most of the time the interesting equations are either trivial or impossible to solve analytically. Here we want to concentrate on standard techniques as well as more complicated cases. There is no *best* method to solve a differential equation.

There are five main types of differential equations,

- ordinary differential equations (ODEs), discussed in this chapter for initial value problems only. They contain functions of one independent variable, and derivatives in that variable. The next chapter deals with ODEs and boundary value problems.

- Partial differential equations with functions of multiple independent variables and their partial derivatives, covered in chapter 15.

- So-called delay differential equations that involve functions of one dependent variable, derivatives in that variable, and depend on previous states of the dependent variables.

- Stochastic differential equations (SDEs) are differential equations in which one or more of the terms is a stochastic process, thus resulting in a solution which is itself a stochastic process.

- Finally we have so-called differential algebraic equations (DAEs). These are differential equation comprising differential and algebraic terms, given in implicit form.

In this first section we restrict the attention to ordinary differential equations. We focus on initial value problems and present some of the more commonly used methods for solving such problems numerically. One example will likely deal with two-point boundary problems. The physical systems which are discussed range from the classical pendulum with non-linear terms to the physics of motion to stars. We will later focus on partial differential equations, where we have to be careful in studying them. We will focus on wave propagation in simple media.

- The order of the ODE refers to the order of the derivative on the left-hand side in the equation

$$\frac{dy}{dt} = f(t, y) \tag{1}$$

This equation is of first order and f is an arbitrary function. A second-order equation goes typically like

$$\frac{d^2y}{dt^2} = f(t, dy, y) \tag{2}$$

A well-known second-order equation is Newtons second law

$$m\frac{d^2x}{dt^2} = -kx \tag{3}$$

where k is the force constant. ODE depend only on one variable, whereas

- partial differential equations like the time-dependent wave equation

$$\frac{\partial^2x}{\partial t^2} = c^2\nabla^2 u \tag{4}$$

may depend on several variables. In certain cases, the wave function can be factorized in functions of the separate variables, so that the equation can be rewritten in terms of sets of ordinary differential equations.

- We distinguish also between linear and non-linear differential equation where e.g.,

$$\frac{dy}{dtr} = g^3(t)y(t) \tag{5}$$

is an example of a linear equation, while

$$\frac{dy}{dt} = g^3(t)y(t) - g(t)y^2(t) \tag{6}$$

is a non-linear ODE.

- Another concept which dictates the numerical method chosen for solving an ODE, is that of initial and boundary conditions. To give an example, in a study of stars, we may need to solve two coupled first-order differential equations, one for the total mass m and one for the pressure P as functions of ρ

$$\frac{dm}{dr} = g^3(t)y(t) - g(t)y^2(t) \tag{7}$$

and

$$\frac{dP}{dr} = -\frac{Gm(r)}{r^2}\rho(r)/c^2 \tag{8}$$

where $\rho(r)$ is the mass-energy density. The initial conditions are dictated by the mass being zero at the center of the star, i.e., when $r = 0$, yielding $m(r = 0) = 0$. The other condition is that the pressure vanishes at the surface of the star. This means that at the point where we have $P = 0$ in the solution of the integral equations, we have the total radius R of the star and the total mass $m(r = R)$. These two conditions dictate the

solution of the equations. Since the differential equations are solved by stepping the radius from $r = 0$ to $r = R$, so-called one-step methods (see the next section) or Runge-Kutta methods may yield stable solutions.

In the solution of the wave equation for a particle in a potential, we may need to apply boundary conditions as well, such as demanding continuity of the wave function and its derivative.

- In many cases it is possible to rewrite a second-order differential equation in terms of two first order differential equations. Consider again the case of Newton's second law. If we define the position $x(t) = y^{(1)}(t)$ and the velocity $v(t) = y^{(2)}(t)$ as its derivative

$$\frac{dy^{(1)}}{dt} = \frac{dx}{dt} = y^{(2)}(t) \tag{9}$$

we can rewrite Newton's second law as two coupled first-order differential equations

$$m\frac{dy^{(2)}}{dt} = -kx(t) = -ky^{(1)} \tag{10}$$

and

$$\frac{dy^{(1)}}{dt} = y^{(2)}(t) \tag{11}$$

3 Solution of ODEs

Consider the differential equation

$$\frac{dy}{dx} = y'(x) = f(x, y) \tag{12}$$

If f is a function of x alone, we can solve for y

$$\frac{dy}{dx} = f(x, y) \tag{13}$$

$$\int dy = \int f(x)dx \tag{14}$$

$$y(x) = \int f(x)dx \tag{15}$$

but this would be an uninteresting case, so we assume f is a function of variables x and y.

Let's consider the Taylor series expansion of $y(x)$

$$y(x) = y(x_0) + (x - x_0)y'(x_0) + \frac{x - x_0)^2}{2}y''(x_0) + \cdots \tag{16}$$

and we could clearly start adding more and more derivative terms. But the original differential equation gives us the derivative y' at any point. If we also know the value of y at some point x_0, $y_0 = y(x_0)$ we could approximate the function by a Taylor series expansion truncated to two terms

$$y(x) = y_0 + (x - x_0)y'(x_0) \tag{17}$$

3.1 Euler method

If we introduce a *step-size* variable h, such that $h = x - x_0$ then we can rewrite the previous expression

$$y(x_0 + h) = y_0 + hf(x_0, y_0) \quad (18)$$
$$= y_0 + hf_0 \quad (19)$$

where we simple have introduced simple abbreviations.

This scheme presented is known as the *simple Euler method*, first introduced by Leonhard Euler (Swiss, 1707 - 1783), one of the most influential mathematicians ever. Even though half- and eventually fully blind he published 500 books and a total of 886 studies

It works as *moving the solution along, one step at a time*. Typically one would divide the interval one is interested in into steps of size h, solve for the solution at $x = x_0 + h$ and use the solution as a new y_0 until the end is reached. Making the step size smaller increases the accuracy and can be used for result checking.

As an example, let's consider

$$y'(x) = y^2 + 1$$

on the interval $0 < x < 1$ with the boundary condition $y(0) = 0$. The analytic result is

$$y(x) = \tan(x)$$

Note that in order to solve this ODE we NEED one boundary condition. In the case of higher order ODEs or couples of ODEs we will need more boundary conditions to get a unique answer.

Coding this is actually pretty simple, namely

```
!  Initialize variables
   xi = 0.d0
   xe = 1.0d0
   h =  0.05d0
   N = int((xe-xi)/h) + 1

!  Initial boundary condition

      xs = 0.d0
      ys = 0.d0
      xeu(1)     = xs
      yeu(1)     = ys

!  Perform Euler solution

      do i = 1, N-1
         call euler(fun1,xeu(i),yeu(i),xeu(i+1),yeu(i+1),h)
      enddo
```

and the subroutine `euler` is

```
ys_prime = fun(xs,ys)

xf = xs+h

yf = ys + h*ys_prime
```

and we need a function called `fun1` that represents our $y'(x)$

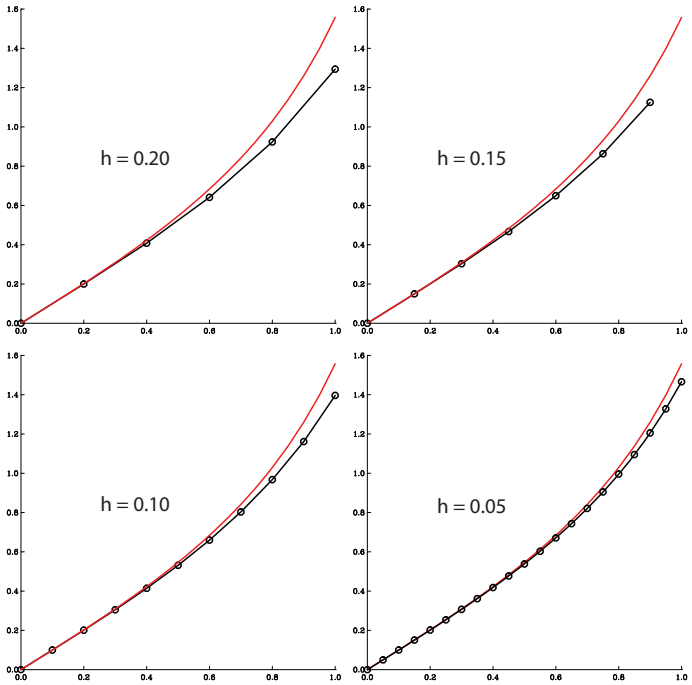
```
real(8) function fun1(x,y) result(z)
...
...

z = y**2 + 1.d0

end function fun1
```

You may note that I have used double precision in my calculations. It is probably not necessary in this case, but it is good practice to always do it with higher precision, because sometimes that may be the difference between the right result and a pretty bad one.

Below, the figure that shows the effect of the choice of h. The red line is the



exact solution, and the other our approximate solution. As expected, as h gets smaller so do the errors.

It becomes also pretty clear that the Euler method has several deficiencies.

- The errors heavily depend on the choice of h.
- As we make h smaller, we get a better result, but we may also need more calculations.
- The error, once introduced, is carried on progressively.
- It would be better to work with *median* values, e.g. the derivative halfway through the step.

4 Modified Euler or the Runge-Kutta of 2nd order

One possible refinement of the Euler method as expressed above is to use the derivative half-way through the step. The problem with this kind of refinement is that one has to evaluate the derivative at the midpoint of the step interval, but the required value of y there is not known beforehand.

We could however use the Simple Euler Method to approximate the value at the middle of the step interval, say

$$x_a = x_0 + \frac{h}{2} \tag{20}$$

leading to

```
y_a = y_0 + h/2.d0*y'(x_0)
```

and with this expression we can now evaluate the derivative at the midpoint $f(x_a, y_a)$ and now use this as a refined approximation of the derivative over the entire interval.

While Euler's Simple Method corresponds to drawing a straight line with derivative $f(x_0, y_0)$ through the point (x_0, y_0) the Runge-Kutta (Modified Euler Method) puts a line through (x_0, y_0) but with approximately the derivative at the midpoint of the interval.

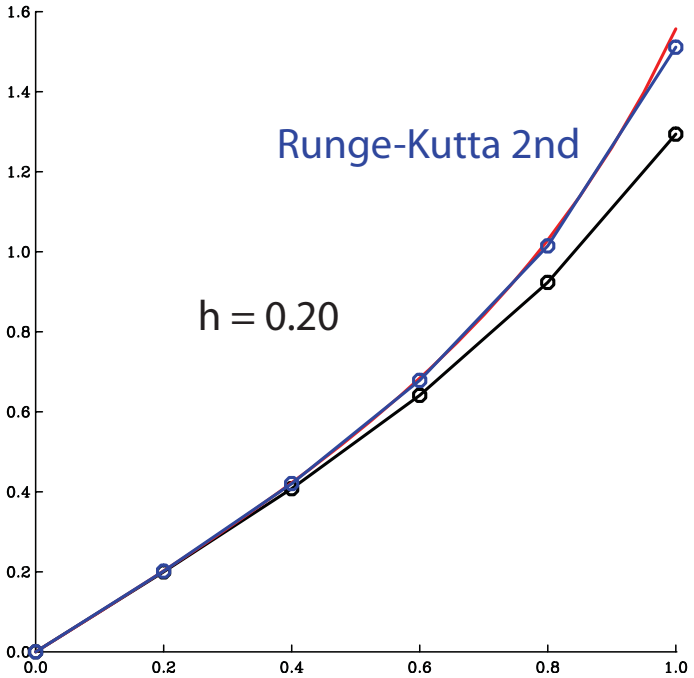
The code below shows the adaptation for performing the Runge-Kutta Method of 2nd order for our example problem.

```
ys_prime = fun(xs,ys)

xa = xs + h/2.d0
ya = ys + h/2.d0*ys_prime

ya_prime = fun(xa,ya)

xf = xs + h
yf = ys + h*ya_prime
```



Note how we basically perform Euler twice, but as shown in the figure below, we have better results that with the simple Euler. With the same sampling h, the RK of 2nd order has a significant improvement compared to Euler. The results are slightly better than those obtained using $h = 0.05$ and Euler, and less number of calculations have been carried out.

Therefore the Modified Euler Method as well as the Improved Euler Method (not described here) both agree with the Taylor series through terms involving h^2 and are said to be second-order Runga-Kutta methods.

5 4th order Runge-Kutta

An even better algorithm (one that is widely used in numerical analysis in science) is the 4th order Runge-Kutta. It is fourth order meaning that the error per step is on the order of h^5, while the total accumulated error has order h^4.

The algorithm is based on a weighted average of various slope estimates. Thus, the next value y_{n+1} is determined by the present value y_n plus the product of the size of the interval (h) and an estimated slope. The slope is a weighted average of slopes

- k_1 is the slope at the beginning of the interval;
- k_2 is the slope at the midpoint of the interval, using slope k_1 to determine the value of y at the point $x_0 + h/2$ using Euler's method

- k_3 is again the slope at the midpoint, but now using the slope k_2 to determine the y-value;
- k_4 is the slope at the end of the interval, with its y-value determined using k_3.

In averaging the four slopes, greater weight is given to the slopes at the midpoint:

$$\text{slope} = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4).$$

Here is my F90 version of the 4th order Runge-Kutta.

```
subroutine rk4(fun,xs,ys,xf,yf,h)

!
! Subroutine using the 4th order Runge-Kutta algorithm
! to progress the solution of a 1st order differential
! equation of the form
!
! dy/dx = f(x,y)
!
! for the starting point xs to the point xf = xs + h.
!
! INPUT
!       xs      starting point
!       ys      y(xs)
!       h       step to perform progression
!
! OUTPUT
!       xf      end point
!       yf      yf = y(xf), the value at point xf
!
!*******************************

   implicit none

   real(8), intent(in)  :: xs, ys, h
   real(8), intent(out) :: xf, yf

   real(8) :: fun

!  Middle steps

   real(8) :: xa, xb, xc, ya, yb, yc
   real(8) :: ys_prime, ya_prime, yb_prime, yc_prime

!*******************************
```

```
   ys_prime = fun(xs,ys)

!  First step

   xa = xs + h/2.d0
   ya = ys + h/2.d0 * ys_prime

   ya_prime = fun(xa,ya)

! Second step

   xb = xs + h/2.d0
   yb = ys + h/2.d0 * ya_prime

   yb_prime = fun(xb,yb)

! Third step

   xc = xs + h
   yc = ys + h * yb_prime

   yc_prime = fun(xc,yc)

! Fourth step

   xf = xs + h
   yf = ys + h/6.d0 * &
      (ys_prime + 2.d0*ya_prime + 2.d0*yb_prime + yc_prime)

   return

end subroutine rk4
```

6 Coupled ODEs

A second order Runge-Kutta subroutine for coupled first-order differential equations.

```
subroutine crk2(f1,f2,xs,y1s,y2s,xf,y1f,y2f,h)

!
! Subroutine using the 2nd order Runge-Kutta algorithm
! to progress the solution of a pair of coupled 1st
! order differential equations of the form
!
```

```
! dy1/dx = f1(x,y1,y2)
! dy2/dx = f2(x,y1,y2)
!
! for the starting point xs to the point xf = xs + h.
!
! INPUT
!       xs      starting point
!       y1s     y1(xs)
!       y2s     y2(xs)
!       h       step to perform progression
!
! OUTPUT
!       xf      end point
!       y1f      y1f = y1(xf), the value at point xf
!       y2f      y2f = y2(xf), the value at point xf
!
!*********************************

   implicit none

   real(8), intent(in)  :: xs, y1s, y2s, h
   real(8), intent(out) :: xf, y1f,y2f

   real(8) :: f1, f2

!  Middle steps

   real(8) :: xa, y1a, y2a
   real(8) :: y1s_prime, y1a_prime, y2s_prime,  y2a_prime

!*******************************

   y1s_prime = f1(xs,y1s,y2s)
   y2s_prime = f2(xs,y1s,y2s)

   xa  = xs + h/2.d0
   y1a = y1s + h/2.d0*y1s_prime
   y2a = y2s + h/2.d0*y2s_prime

   y1a_prime = f1(xa,y1a,y2a)
   y2a_prime = f2(xa,y1a,y2a)

   xf = xs + h
   y1f = y1s + h*y1a_prime
   y2f = y2s + h*y2a_prime
```

```
   return

end subroutine crk2
```

Note that we are simply using the RK2 methodology twice, once for every differential equation. This is valid for linear differential equations.

7 Second order ODEs

Until now we always managed to get solutions to our differential equations. But so far we mainly were concerned with first-order equations of the form

$$y' = f(x, y) \tag{21}$$

In physics second-order differential equations are more common. Therefore we need to start dealing with equations of the form:

$$y'' = f(x, y, y') \tag{22}$$

At first sight this seems to be an entirely new class of problems. But it is possible to reduce this problem to a more familiar form.

Let us introduce two new variables

$$y_1 = y \tag{23}$$

and

$$y_2 = y' \tag{24}$$

We may thus write the original second-order differential equation as *a set of two first-order equations*,

$$y_1' = y_2 \tag{25}$$
$$y_2' = f(x, y_1, y_2) \tag{26}$$

Another way to look at the previous equations is to envision them as vector quantities. We could introduce two more variables,

$$f_1 = y_2 \qquad \text{and} \qquad f_2 = f(x, y_1, y_2) \tag{27}$$

we further obtain

$$\begin{bmatrix} y_1' \\ y_2' \end{bmatrix} = \begin{bmatrix} y_2 \\ f(x, y_1, y_2) \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} \tag{28}$$
$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}' = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} \tag{29}$$
$$\mathbf{y}' = \mathbf{f} \tag{30}$$

Thus the problem is not so much different than what we have seen so far. It just has more components.

Instead of stepping our one solution along we need to modify the existing Runge-Kutta or Euler codes so that it steps our two components along.

One possible required modifications will consist of replacing scalar quantities with arrays, but it is easier to just do the calculations twice, similar to the coupled ODEs Runge Kutta method used in our previous Homework.

The independent variable x will remain a scalar, but the dependent variables $y, \hat{y}, f0, f1, \cdots$ will all become arrays.

Let us try to solve a simple example using an adaptation of the Coupled Runge-Kutta code displayed above. Let us the notation

$$\begin{aligned} f_1 &= y_2 \\ f_2 &= f(x, y_1, y_2) \end{aligned}$$

where f_2 represents the second-order differential equation, which we know. f_1 is not know instead, and we use the value of the derivative f_2 from the previous step. Do not try to calculate f_1 by integration. In some cases this is easy, but we are using computer just because we can't really solve al ODEs, right?

So, our first variation of `crk2` is

```
y1s_prime = y2s
y2s_prime = f2(xs,y1s,y2s)
```

where `f2` and `xs, y1s, y2s` are known.They represent the initial conditions. Now, we estimate the values at the middle step just as `rk2` does.

```
xa  = xs + h/2.d0
y1a = y1s + h/2.d0*y1s_prime
y2a = y2s + h/2.d0*y2s_prime
```

and use this values to get the derivatives at the middle point $h/2$.

```
y1a_prime = y2a
y2a_prime = f2(xa,y1a,y2a)
```

Note that again for the derivative for `f1` we use the value `y2` at the middle point. You can easily continue this process to get the 2nd order Runge-Kutta and even the $4th$ order RK.

7.1 Example 1

We want to solve the following problem

$$y'' = -4y \tag{31}$$

with boundary conditions

$$\begin{aligned} y(0) &= 1 \\ y'(0) &= 0 \end{aligned}$$

Note that since we have a seond-order ODE, we need TWO boundary conditions to obtain a unique solution. We know the solution to this problem, namely,

$$y = \cos(2x) \tag{32}$$

I define

$$\begin{aligned} f_1 &= y_2 \\ f_2 &= f(x, y_1, y_2) = -4y \end{aligned}$$

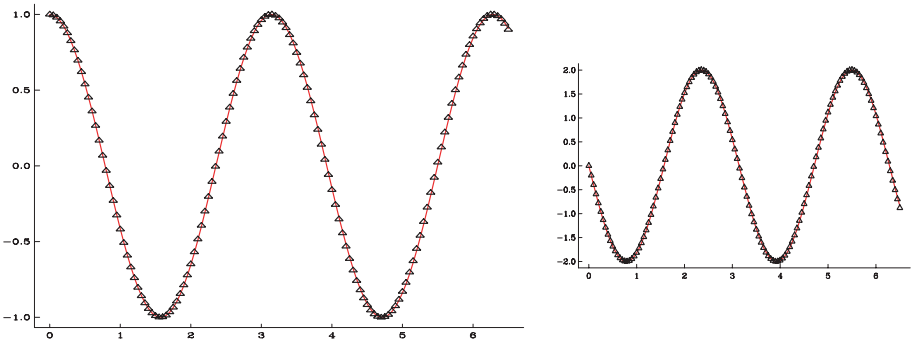
and use the starting conditions

```
xs  = 0.d0
y1s = 1.d0
y2s = 0.d0
```

and loop over x by calling the subroutine called `drk2`,

```
 do i = 1, N-1
   call drk2(f2,xs(i),y1s(i),y2s(i),   &
         xf(i+1),y1f(i+1),y2f(i+1),h)
enddo
```

Note that I only need to have the external function `f2`, since `f1` is not known and not needed. Also, as an added bonus, the method provides as an output the estimates of $y'(x)$ as well as the requested $y(x)$. Below, I plot the comparison of the analytical and numerical solution. On the left, the solution of the $2nd$ order ODE and on the right, the added bonus, the values of $y'(x)$. Note the boundary conditions are correct.



8 Two-point boundary problems

Here the two boundary conditions are not at the same point, typically they are two values of y, given at different values of x. So, our boundary conditions are $y = y_a$ at $x = a$ and $y = y_b$ at $x = b$. What do we do then?

We need to pick one of these two values of x, say $x = a$, use the one boundary condition there, that $y = y_a$ there, together with a guess at y' there. Using this guess we integrate the equation to obtain the resulting value of $y = y_b$ at $x = b$. This will not equal y_b so we need to vary our guess at y' at $x = a$ until we get $y = y_b$ at $x = b$. This varying one number, y' at $x = a$, to make a function $y(x = b) - y_b$ zero is what is known as root finding.

Thus we will make a short digression to consider root finding before we tackle solving a second-order equation with two-point boundary conditions.

8.1 Root finding and Newton-Raphson method

To get us started let us consider the simple algebraic equation $y = x^2 - 2$. We want to know the values of x for which $y = 0$, i.e., the roots of $x^2 - 2 = 0$. There are two roots of course $x = \pm\sqrt{2}$. If we consider only positive values of x, then we have only one root $x = \sqrt{2}$.

The Newton-Raphson method takes advantage of the derivative of the function. The tangent of the curve is constructed at each succesive value of x_n and the next value x_{n+1} taken as the point at which the tangent cuts the axis $f(x) = 0$.

If the nth value is x_n, the tangent to the curve of $f(x)$ at that point has slope $f'(x)$ and passes through the point $x = x_n$, $y = f(x_n)$. Its equation is thus

$$y(x) = (x - x_n)f'(x_n) + f(x_n) \tag{33}$$

The value at x at which $y = 0$ is then taken as x_{n+1}, so solving for x_{n+1} leaves us with

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{34}$$

and this is the Newton-Raphson iteration formula.

For our example above we have

$$y(x) = x^2 - 2 \tag{35}$$

and

$$y'(x) = 2x \tag{36}$$

Let's start with a guess of our result, say $x_1 = 1.0$. The table below shows the steps taken by just following our Newton-Raphson formulae.

Table 1: default

x	y	y'
1.0000000	-1.0000000	2.0000000
1.5000000	0.25000000	3.0000000
1.4166666	6.94441795E-03	2.8333333
1.4142157	5.96046448E-06	2.8284314
1.4142135	-1.19209290E-07	2.8284271