

# DIFFERENCE SCHEMES FOR ODE

BY  
MICHAEL E. TAYLOR

## Contents:

1. Difference schemes based on power series.
2. Numerical integration and improved Euler methods.
3. The Runge-Kutta scheme.

## Introduction

We discuss several methods of approximating solutions to a first order system of ODEs

$$(1) \quad X'(t) = F(X), \quad X(0) = X_0,$$

using difference schemes. The method consists of approximating  $X(nh)$  by  $X_n$ ,  $n = 1, 2, 3, \dots$ , defined inductively by

$$(2) \quad X_{n+1} = \Phi_h(X_n),$$

starting with  $X_0$ , given by (1). We say (2) is  $k^{\text{th}}$  order accurate provided that, if  $X(t)$  actually is a solution to (1), then

$$(3) \quad \Phi_h(X(t)) = X(t+h) + r(X(t), h),$$

where the remainder term  $r(X, h)$  satisfies the estimate

$$(4) \quad |r(X, h)| \leq C(X)h^{k+1}.$$

In such a case, under appropriate conditions, the approximation given by (2) differs from the solution to (1) by  $O(h^k)$ , in the sense that, given that (1) has a solution for  $0 \leq t \leq T$ ,

$$(5) \quad |X(nh) - X_n| \leq C(T)h^k, \text{ for } 0 \leq nh \leq T.$$

Two methods of devising difference schemes are based on power series and on numerical integration, particularly via the trapezoidal rule and via Simpson's rule. We devote §1 to a discussion of power series methods and §2-3 to methods based on integration, with emphasis on the famous Runge-Kutta method. In fact, methods of §2 are inferior to the Runge-Kutta method, and would tend not to be used; the study of these methods serves only a pedagogical purpose of preparing one to understand the truly useful Runge-Kutta method. There is an even more primitive method, called, probably inappropriately, the Euler method. This method, which is first order accurate, is actually defined after formula (6) in §1. Here is all there is to say about that method:

Don't use it.

We illustrate these methods with several computer programs, including versions in Pascal, BASIC, and C. We include some figures, produced using Gauss, some of which illustrate the accuracy of various methods.

While we do not make specific mention of it, standard methods of converting higher order ODEs to first order systems allow one to adapt rather easily the difference schemes discussed here to such higher order equations. Also, the extension to the treatment of non-autonomous ODEs, e.g., of the type (1) with  $F(X)$  replaced by  $F(t, X)$ , is straightforward.

There are a number of methods for constructing difference schemes, particularly ‘multistep’ methods, such as predictor-corrector methods, not touched on here. Discussion of these can be found in books listed in the references.

## 1. Difference schemes based on power series

We describe one method of constructing a  $k^{\text{th}}$  order accurate difference approximation to the solution of a first order system of ODEs

$$(1) \quad X'(t) = F(X), \quad X(0) = X_0.$$

It derives from the expansion

$$(2) \quad X(t+h) = X(t) + hX'(t) + \frac{1}{2}h^2X''(t) + \cdots + \frac{1}{k!}h^kX^{(k)}(t) + O(h^{k+1}).$$

To begin, differentiate the equation (1), producing

$$(3) \quad X''(t) = F_2(X, X'),$$

where  $F_2(X, X') = F'(X)X'$ , and continue differentiating, producing

$$(4) \quad X^{(j)}(t) = F_j(X, X', \dots, X^{(j-1)}), \quad j \leq k.$$

Then the difference scheme for an approximation  $X_n$  to  $X(nh)$  is of the form

$$(5) \quad X_{n+1} = X_n + hX'_n + \frac{1}{2}h^2X''_n + \cdots + \frac{1}{k!}h^kX_n^{(k)}$$

where

$$X'_n = F(X_n), \quad X''_n = F_2(X_n, X'_n),$$

and, inductively,

$$(6) \quad X_n^{(j)} = F_j(X_n, X'_n, \dots, X_n^{(j-1)}).$$

If one takes  $k = 1$ , the method is what is called the Euler method. It should be noted that often one does not want to use this method for  $k = 4$ , say, because the formula for  $F_4(X, X', X'', X^{(3)})$  might be too complicated. However, if such a formula happens not to be very complicated, this is a good method.

To give one explicit example, consider the linear ODE

$$(7) \quad X'(t) = AX,$$

and take  $k = 3$ . Then the difference scheme is

$$(8) \quad X_{n+1} = X_n + hX'_n + \frac{1}{2}h^2X''_n + \frac{1}{6}h^3X_n^{(3)},$$

with

$$(9) \quad X'_n = AX_n, \quad X''_n = AX'_n = A^2X_n, \quad X_n^{(3)} = AX''_n = A^3X_n.$$

Another ODE amenable to this method is the set of Lorenz equations

$$(10) \quad \begin{aligned} x' &= s(y - x) \\ y' &= x(r - z) - y \\ z' &= xy - bz. \end{aligned}$$

A typical set of values for the parameters  $s, r, b$  is

$$(11) \quad s = 10, \quad r = 28, \quad b = 8/3.$$

Differentiating (10) produces

$$(12) \quad \begin{aligned} x'' &= s(y' - x') \\ y'' &= x'(r - z) - xz' - y' \\ z'' &= xy' + x'y - bz' \end{aligned}$$

and differentiating again yields

$$(13) \quad \begin{aligned} x''' &= s(y'' - x'') \\ y''' &= x''(r - z) - 2x'z' - xz'' - y'' \\ z''' &= xy'' + 2x'y' + x''y - bz'' \end{aligned}$$

From this one produces a third order accurate difference scheme for the Lorenz equations.

### Computer program examples

The following are programs written respectively for Borland's Turbo C, Borland's Turbo Pascal (version 5.5), Borland's Turbo Basic, and Aptech's Gauss. Each one produces numerical approximations to solutions of the Lorenz equations (10), using a third order accurate difference scheme, of the form (5), with  $k = 3$ . In view of the calculations (11)-(13), we have

$$(14) \quad \begin{aligned} x_{n+1} &= x_n + hx'_n + \frac{h^2}{2}x''_n + \frac{h^3}{6}x'''_n \\ y_{n+1} &= y_n + hy'_n + \frac{h^2}{2}y''_n + \frac{h^3}{6}y'''_n \\ z_{n+1} &= z_n + hz'_n + \frac{h^2}{2}z''_n + \frac{h^3}{6}z'''_n \end{aligned}$$

where, by (10),

$$(15) \quad \begin{aligned} x'_n &= s(y_n - x_n) \\ y'_n &= x_n(r - z_n) - y_n \\ z'_n &= x_n y_n - bz_n, \end{aligned}$$

by (12),

$$(16) \quad \begin{aligned} x_n'' &= s(y_n' - x_n') \\ y_n'' &= x_n'(r - z_n) - x_n z_n' - y_n' \\ z_n'' &= x_n y_n' - x_n' y_n - b z_n' \end{aligned}$$

and, by (13),

$$(17) \quad \begin{aligned} x_n''' &= s(y_n'' - x_n'') \\ y_n''' &= x_n''(r - z_n) - 2x_n' z_n' - x_n z_n'' - y_n'' \\ z_n''' &= x_n y_n'' + 2x_n' y_n' + x_n'' y_n - b z_n'' \end{aligned}$$

These calculations can be seen in the inner loops of the following programs. We have used the labels  $dx$ ,  $dy$ ,  $dz$  for  $x'$ ,  $y'$ ,  $z'$ , the labels  $x2$ ,  $y2$ ,  $z2$  for  $x''$ ,  $y''$ ,  $z''$ , and the labels  $x3$ ,  $y3$ ,  $z3$  for  $x'''$ ,  $y'''$ ,  $z'''$ . Also, we set  $hh$  equal to  $h^2/2$  and  $hc$  equal to  $h^3/6$ .

Each program computes 21 orbits, with initial points

$$(18) \quad x(0) = -20 + 2n, \quad y(0) = 0, \quad z(0) = 5, \quad 0 \leq n \leq 20,$$

except the Gauss program, which plots 20 orbits, i.e., for  $0 \leq n < 20$ . To plot the orbits on the 2-dimensional screen, a linear map from  $\mathbb{R}^3$  to  $\mathbb{R}^2$  is used:

$$u = x + y/2, \quad v = z - y/2.$$

The program written in Gauss is capable of producing a high resolution printout on a laser printer. Such a printout appears after the listing of the Gauss program.

In the C and Pascal programs are commands to include a file called 'screen.c' (or picture.pas), containing a function (or procedure) called 'picture,' which prepares the computer to display graphics. Since specific ways to do this vary so much from one version of a personal computer language to another, it doesn't seem useful to specify what commands are in this procedure. A copy of the file picture.pas can be found in [T].

### C program for Lorenz equation

```
# include <graphics.h>
# include <screen.c>
main()
{
double s,r,b,h,hh,hc,u,v;
double x,y,z,dx,dy,dz,x2,y2,z2,x3,y3,z3;
int n,k,i,m1,n1;
picture(0);
```

```

s=10;
r=28;
b=2.667;
h=0.003;
hh=h*h/2;
hc=h*hh/3;
k=4000;
for(n=0;n<=20;n++){
  x=-20+2*n;
  y=0;
  z=5;
  for(i=1;i<=k;i++){
    dx=s*(y-x);
    dy=x*(r-z)-y;
    dz=x*y-z*b;
    x2=s*(dy-dx);
    y2=-x*dz+dx*(r-z)-dy;
    z2=x*dy+y*dx-b*dz;
    x3=s*(y2-x2);
    y3=-x*z2+x2*(r-z)-2*dx*dz-y2;
    z3=x*y2+y*x2+2*dx*dy-b*z2;
    x=x+h*dx+hh*x2+hc*x3;
    y=y+h*dy+hh*y2+hc*y3;
    z=z+h*dz+hh*z2+hc*z3;
    u=x+0.5*y;
    v=z-0.5*y;
    u=320+7*u;
    v=400-7*v;
    m1=u;
    n1=v;
    putpixel(m1,n1,n+1);
  }
}
}

```

### Pascal program

```

program lorenz;
uses
  Crt,Graph;
var
  s,r,b,h,hh,hc,u,v:double;
  x,y,z,dx,dy,dz,x2,y2,z2,x3,y3,z3:double;
  n,k,i,m1,n1:integer;
{$I picture.pas}

```

```

begin
picture(0,4);
s:=10;
r:=28;
b:=2.667;
h:=0.003;
hh:=h*h/2;
hc:=h*hh/3;
k:=4000;
for n:=0 to 20 do
begin
  x:=-20+2*n;
  y:=0;  z:=5;
  for i:=1 to k do
  begin
    dx:=s*(y-x);
    dy:=x*(r-z)-y;
    dz:=x*y-z*b;
    x2:=s*(dy-dx);
    y2:=-x*dz+dx*(r-z)-dy;
    z2:=x*dy+y*dx-b*dz;
    x3:=s*(y2-x2);
    y3:=-x*z2+x2*(r-z)-2*dx*dz-y2;
    z3:=x*y2+y*x2+2*dx*dy-b*z2;
    x:=x+h*dx+hh*x2+hc*x3;
    y:=y+h*dy+hh*y2+hc*y3;
    z:=z+h*dz+hh*z2+hc*z3;
    u:=x+0.5*y;
    v:=z-0.5*y;
    m1:=trunc(320+6*u);
    n1:=trunc(320-6*v);
    PutPixel(m1,n1,n);
  end;
end;
end.

```

### Basic program

```

defint n,i,k
SCREEN 12
PALETTE
s=10
r=28
b=2.667

```

```

h=0.003
hh=h*h/2
hc=h*hh/3
k=4000
for n=0 to 20
  x=-20+2*n
  y=0
  z=5
  for i=1 to k
    dx=s*(y-x)
    dy=x*(r-z)-y
    dz=x*y-z*b
    x2=s*(dy-dx)
    y2=-x*dz+dx*(r-z)-dy
    z2=x*dy+y*dx-b*dz
    x3=s*(y2-x2)
    y3=-x*z2+x2*(r-z)-2*dx*dz-y2
    z3=x*y2+y*x2+2*dx*dy-b*z2
    x=x+h*dx+hh*x2+hc*x3
    y=y+h*dy+hh*y2+hc*y3
    z=z+h*dz+hh*z2+hc*z3
    u=x+0.5*y
    v=z-0.5*y
    u=320+7*u
    v=400-7*v
    u%=u
    v%=v
    pset(u%,v%) ,n+1
  next i
next n
end

```

### Gauss program

```

library pgraph;
s=10;
let r[1,20]=28;
b=2.667;
h=0.003;
hh=h*h/2;
hc=h*hh/3;
xx=seqa(-20,2,20);
x=xx';
let y[1,20]=0;

```

```

let z[1,20]=5;
t=x+0.5*y;
w=z-0.5*y;
i=0;
do until i==4000;
  i=i+1;
  dx=s*(y-x);
  dy=x.*(r-z)-y;
  dz=x.*y-z.*b;
  x2=s*(dy-dx);
  y2=-x.*dz+dx.*(r-z)-dy;
  z2=x.*dy+y.*dx-b*dz;
  x3=s*(y2-x2);
  y3=-x.*z2+x2.*(r-z)-2*dx.*dz-y2;
  z3=x.*y2+y.*x2+2*dx.*dy-b*z2;
  x=x+h*dx+hh*x2+hc*x3;
  y=y+h*dy+hh*y2+hc*y3;
  z=z+h*dz+hh*z2+hc*z3;
  u=x+0.5*y;
  v=z-0.5*y;
  t=t|u;
  w=w|v;
endo;
_plttype = 6;
xy(t,w);

```

## 2. Numerical integration and improved Euler methods

We will derive some second order accurate difference schemes via second order accurate methods of numerical integration. The connection between the two comes from rewriting the ODE

$$(1) \quad X'(t) = F(X)$$

as

$$(2) \quad X(t+h) = X(t) + \int_0^h F(X(t+s))ds.$$

Approximating the integral by  $hF(X(t)) + O(h^2)$  yields the Euler method  $X_{n+1} = X_n + hF(X_n)$ , mentioned in the last section. Consider instead, methods of approximating

$$(3) \quad \int_0^h g(s)ds$$

better than  $hg(0) + O(h^2)$ . Two simple improvements are

$$(4) \quad \frac{h}{2} [g(0) + g(h)] + O(h^3),$$

the trapezoidal method, and

$$(5) \quad hg\left(\frac{h}{2}\right) + O(h^3),$$

the midpoint method. These lead respectively to

$$(6) \quad X(t+h) = X(t) + \frac{h}{2} [F(X(t)) + F(X(t+h))] + O(h^3)$$

and

$$(7) \quad X(t+h) = X(t) + hF\left(X\left(t + \frac{h}{2}\right)\right) + O(h^3).$$

Neither of them immediately converts to an explicit difference scheme, but in (6) we can substitute  $F(X(t+h)) = F(X(t) + hF(X(t))) + O(h^2)$  and in (7) we can substitute  $F(X(t + \frac{h}{2})) = F(X(t) + \frac{h}{2}F(X(t))) + O(h^2)$ , to obtain the second order accurate difference schemes

$$(8) \quad X_{n+1} = X_n + \frac{h}{2} [F(X_n) + F(X_n + hF(X_n))]$$

and

$$(9) \quad X_{n+1} = X_n + hF\left(X_n + \frac{h}{2}F(X_n)\right).$$

Often (8) is called Heun's method and (9) a modified Euler method.

Note that expanding the right side of (8) in powers of  $h$  gives

$$(10) \quad \begin{aligned} & X_n + \frac{h}{2} [2F(X_n) + hF'(X_n)F(X_n)] + O(h^3) \\ &= X_n + hF(X_n) + \frac{h^2}{2} F'(X_n)F(X_n) + O(h^3), \end{aligned}$$

which agrees mod  $O(h^3)$  with (1.5). A similar result holds for the expansion of the right side of (9) in powers of  $h$ .

### Computer program example

We will give an example of using the improved Euler method, defined by (8), to approximate solutions to the ODE

$$(11) \quad X''(t) = -X/|X|^3,$$

for  $X = (x, y) \in \mathbb{R}^2$ . This models a central force problem, describing for example the orbit of a single planet about a star. It converts to the first order system

$$(12) \quad (X, V)' = (V, -X/|X|^3)$$

with  $V = (v, w) = (x', y')$ . In other words,

$$(13) \quad \begin{aligned} x' &= v \\ y' &= w \\ v' &= -x(x^2 + y^2)^{-\frac{3}{2}} \\ w' &= -y(x^2 + y^2)^{-\frac{3}{2}}. \end{aligned}$$

The iteration, from  $(X, V)(t) = (x, y, v, w)$  to  $(X, V)(t+h)$ , via (8), is accomplished by setting

$$(14) \quad a_1 = f(x, y) = -x(x^2 + y^2)^{-\frac{3}{2}}, \quad b_1 = g(x, y) = -y(x^2 + y^2)^{-\frac{3}{2}},$$

letting

$$(15) \quad x_2 = x + hv, \quad y_2 = y + hw, \quad v_2 = v + ha_1, \quad w_2 = w + hb_1,$$

and then

$$(16) \quad a_2 = f(x_2, y_2), \quad b_2 = g(x_2, y_2),$$

and finally

$$(17) \quad \begin{aligned} x(t+h) &= x + \frac{h}{2}(v + v_2) \\ y(t+h) &= y + \frac{h}{2}(w + w_2) \\ v(t+h) &= v + \frac{h}{2}(a_1 + a_2) \\ w(t+h) &= w + \frac{h}{2}(b_1 + b_2). \end{aligned}$$

This iterative method is used in the program written for Borland's Turbo C, listed below. Note that the same program works for a general ODE of the form

$$(18) \quad (x'', y'') = (f(x, y), g(x, y));$$

one need only change the part defining  $f_{nf}(x, y)$  and  $f_{ng}(x, y)$ .

Accompanying the program listing are several graphs, showing the orbits for various choices of  $h$ , produced by the analogous program in Gauss. In the listed program, the step size  $h$  is .01, and 2500 steps are used. This is seen not to capture well the known feature that planetary orbits are closed. In pictures showing step sizes  $h = .005$  and  $h = .002$ , for which the number of steps  $N$  was adjusted to keep the length of the  $t$ -interval  $Nh$  constant, a steady improvement is seen. The Runge-Kutta method, discussed in the next section, will be seen to give a considerably more accurate picture.

### C program using improved Euler method

```
# include <graphics.h>
# include <math.h>
# include <screen.c>
double fnf(double x, double y);
double fng(double x, double y);
main()
{
double x,y,v,w,h,ha;
double a1,b1,x2,y2,v2,w2,a2,b2;
int j,kx,ky;
picture(0);
x=.7;
y=0;
v=0;
w=.8;
h=.01;
ha=h/2;
putpixel(320,240,4);
for(j=1;j<=2500;j++){
    kx=320+240*x;
    ky=240-240*y;
    putpixel(kx,ky,2);
    a1=fnf(x,y);
    b1=fng(x,y);
    x2=x+h*v;
    y2=y+h*w;
    v2=v+h*a1;
    w2=w+h*b1;
    a2=fnf(x2,y2);
    b2=fng(x2,y2);
    x=x+ha*(v+v2);
    y=y+ha*(w+w2);
```

```

    v=v+ha*(a1+a2);
    w=w+ha*(b1+b2);
}
}

```

```

double fnf(double x, double y)
{
double r,s,z;
s=x*x+y*y;
r=s*sqrt(s);
z=-x/r;
return z;
}

```

```

double fng(double x, double y)
{
double r,s,z;
s=x*x+y*y;
r=s*sqrt(s);
z=-y/r;
return z;
}

```

### 3. The Runge-Kutta scheme

The Runge-Kutta scheme for an autonomous first order system of ODEs

$$(1) \quad X' = F(X), \quad X(0) = X_0$$

is specified as follows. The approximation  $X_n$  to  $X(nh)$  is given recursively by

$$(2) \quad X_{n+1} = X_n + \frac{h}{6} (K_{n1} + 2K_{n2} + 2K_{n3} + K_{n4}),$$

where

$$(3) \quad \begin{aligned} K_{n1} &= F(X_n) \\ K_{n2} &= F\left(X_n + \frac{1}{2}hK_{n1}\right) \\ K_{n3} &= F\left(X_n + \frac{1}{2}hK_{n2}\right) \\ K_{n4} &= F\left(X_n + hK_{n3}\right). \end{aligned}$$

This scheme is  $4^{th}$  order accurate. It is one of the most popular and important difference schemes used for numerical studies of ODE, so it is very important to understand why it has the accuracy stated.

One way to establish this is to perform a power series expansion in  $h$ , through  $h^4$ , and verify that the appropriate coefficients arise. This verification has the unsatisfactory aspect of not enabling one to see how such a difference scheme was derived. Thus, we will consider a method of deriving  $4^{th}$  order accurate difference schemes, based on Simpson's formula

$$(4) \quad \int_0^h g(s)ds = \frac{h}{6} \left( g(0) + 4g\left(\frac{h}{2}\right) + g(h) \right) + O(h^5).$$

This is derived by producing a quadratic polynomial  $p(s)$  such that  $p(s) = g(s)$  at  $s = 0$ ,  $h/2$ , and  $h$ , and then exactly integrating  $p(s)$ . The formula can be verified by rewriting it as

$$(5) \quad \int_{-h}^h G(s)ds = \frac{h}{3} \left[ G(-h) + 4G(0) + G(h) \right] + O(h^5).$$

The main part on the right is exact for all odd  $G(s)$ , and it is also exact for  $G(s) = 1$  and  $G(s) = s^2$ , so it is exact when  $G(s)$  is a polynomial of degree  $\leq 3$ . Making a power series expansion  $G(s) = \sum_{j=0}^3 a_j s^j + O(s^4)$  then yields (5).

Now, write the ODE (1) as an integral equation

$$(6) \quad X(t+h) = X(t) + \int_0^h F(X(t+s))ds.$$

By (4),

$$(7) \quad \int_0^h F(X(t+s))ds = \frac{h}{6} \left[ F(X(t)) + 4F\left(X\left(t + \frac{h}{2}\right)\right) + F(X(t+h)) \right] + O(h^5).$$

We then have as an immediate consequence the following result on producing accurate difference schemes.

**Proposition 1.** *Suppose the approximation*

$$(8) \quad X(t+h) = X(t) + \Phi(X(t), h) = \mathcal{X}(X(t), h)$$

produces a  $j^{\text{th}}$  order accurate difference scheme for the solution to (1). If  $j \leq 3$ , then a difference scheme accurate of order  $j + 1$  is given by

$$(9) \quad X_{n+1} = X_n + \frac{h}{6} \left[ F(X_n) + 4F(\mathcal{X}(X_n, \frac{h}{2})) + F(\mathcal{X}(X_n, h)) \right].$$

Furthermore, if  $X(t+h) = \mathcal{X}_\ell(X(t), h)$  both work in (8),  $\ell = 0, 1$ , then you can use

$$(10) \quad X_{n+1} = X_n + \frac{h}{6} \left[ F(X_n) + 4F(\mathcal{X}_0(X_n, \frac{h}{2})) + F(\mathcal{X}_1(X_n, h)) \right].$$

We will first apply this to two second order methods derived before:

$$(11) \quad \mathcal{X}_0(X_n, h) = X_n + \frac{h}{2} \left[ F(X_n) + F(X_n + hF(X_n)) \right], \text{ Heun,}$$

and

$$(12) \quad \mathcal{X}_1(X_n, h) = X_n + hF(X_n + \frac{1}{2}hF(X_n)), \text{ modified Euler.}$$

Thus a third order accurate scheme is produced. The last term in (9) becomes

$$(13) \quad \frac{h}{6} \left[ F(X_n) + 4F\left(X_n + \frac{h}{4}[F(X_n) + F(X_n + \frac{h}{2}F)]\right) + F(X_n + hF(X_n + \frac{h}{2}F)) \right],$$

where  $F = F(X_n)$ . In terms of  $K_{n1}$ ,  $K_{n2}$  as defined in (3), we have

$$(14) \quad \frac{h}{6} \left[ K_{n1} + 4F(X_n + \frac{h}{4}[K_{n1} + K_{n2}]) + F(X_n + hK_{n2}) \right].$$

This could be used in a  $3^{\text{rd}}$  order accurate scheme, but some simplification of the middle term is desirable. Note that, for smooth  $H$ ,

$$(15) \quad H(x + \frac{1}{2}\eta) = \frac{1}{2}H(x) + \frac{1}{2}H(x + \eta) + O(|\eta|^2).$$

Consequently, as  $|K_{n1} - K_{n2}| = O(h)$ , by (3),

$$(16) \quad F(X_n + \frac{h}{4}[K_{n1} + K_{n2}]) = \frac{1}{2}F(X_n + \frac{h}{2}K_{n1}) + \frac{1}{2}F(X_n + \frac{h}{2}K_{n2}) + O(h^4).$$

Therefore we have the following.

**Lemma 2.** A 3<sup>rd</sup> order accurate difference scheme for (1) is given by

$$(17) \quad X_{n+1} = X_n + \frac{h}{6}[K_{n1} + 2K_{n2} + 2K_{n3} + L_{n4}]$$

where  $K_{n1}$ ,  $K_{n2}$ ,  $K_{n3}$  are given by (3) and

$$(18) \quad L_{n4} = F(X_n + hK_{n2}).$$

We can now produce a 4<sup>th</sup> order accurate difference scheme by applying Proposition 1 with  $\mathcal{X}(X_n, h)$  defined by (17). Thus we obtain the difference scheme.

$$(19) \quad X_{n+1} = X_n + \frac{h}{6} \left[ K_{n1} + 4F \left( X_n + \frac{h}{12} [K_{n1} + 2k_{n2} + 2k_{n3} + \ell_{n4}] \right) \right. \\ \left. + F \left( X_n + \frac{h}{6} [K_{n1} + 2K_{n2} + 2K_{n3} + L_{n4}] \right) \right],$$

where  $K_{nj}$ ,  $L_{n4}$  are as above and

$$(20) \quad \begin{aligned} k_{n2} &= F \left( X_n + \frac{h}{4} K_{n1} \right) \\ k_{n3} &= F \left( X_n + \frac{h}{4} k_{n2} \right) \\ \ell_{n4} &= F \left( X_n + \frac{h}{2} k_{n2} \right). \end{aligned}$$

This formula is more complicated than the Runge-Kutta formula (2). It requires 9 evaluations of  $F$  rather than just 4. Rather than try to modify (19) mod  $O(h^5)$  to get (2), we will instead look at how (17) fails to be 4<sup>th</sup> order accurate, by expanding (17) in powers of  $h$ , through  $h^4$ . To begin this power series expansion, note that

$$(21) \quad \begin{aligned} K_{n2} &= F(X_n) + \frac{1}{2}hF'(X_n)K_{n1} + \frac{1}{8}h^2F''(X_n)K_{n1}^2 + \frac{1}{48}h^3F'''(X_n)K_{n1}^3 + O(h^4), \\ K_{n3} &= F(X_n) + \frac{1}{2}hF'(X_n)K_{n2} + \frac{1}{8}h^2F''(X_n)K_{n2}^2 + \frac{1}{48}h^3F'''(X_n)K_{n2}^3 + O(h^4), \end{aligned}$$

and

$$(22) \quad L_{n4} = F(X_n) + hF'(X_n)K_{n2} + \frac{1}{2}h^2F''(X_n)K_{n2}^2 + \frac{1}{6}h^3F'''(X_n)K_{n2}^3 + O(h^4).$$

To simplify the notation, we set  $F = F(X_n)$ ,  $F' = F'(X_n)$ ,  $F'' = F''(X_n)$ , etc. Note that  $K_{n1} = F$ . A straightforward substitution yields

$$(23) \quad \begin{aligned} \frac{h}{6} [K_{n1} + 2K_{n2} + 2K_{n3} + L_{n4}] &= hF + \frac{h^2}{6}F' [K_{n1} + 2K_{n2}] \\ &+ \frac{h^3}{6}F'' \left[ \frac{1}{4}K_{n1}^2 + \frac{3}{4}K_{n2}^2 \right] \\ &+ \frac{h^4}{6}F''' \left[ \frac{1}{24}K_{n1}^3 + \frac{5}{24}K_{n2}^3 \right] + O(h^5). \end{aligned}$$

To proceed further, note from (21) that

$$(24) \quad K_{n2} = K_{n1} + \frac{h}{2}F'K_{n1} + \frac{h^2}{8}F''K_{n1}^2 + O(h^3).$$

Further substitution into (23) then shows that (23) is equal to

$$(25) \quad \begin{aligned} hF + \frac{h^2}{6}F'[3F + hF'F + \frac{h^2}{4}F''F^2] \\ + \frac{h^3}{6}F''[F^2 + \frac{3}{4}hF^2F'] + \frac{h^4}{6} \cdot \frac{1}{4}F'''F^3 + O(h^5). \end{aligned}$$

We want to match

$$(26) \quad hX' + \frac{h^2}{2}X'' + \frac{h^3}{6}X''' + \frac{h^4}{24}X^{(4)} + O(h^5).$$

Obviously the coefficients of  $h$  match, by the differential equation (1). Further differentiation of (1) yields

$$(27) \quad \begin{aligned} X'' &= F'X' \\ X''' &= F''X'^2 + F'X'' \\ &= F''X'^2 + F'^2X' \\ X^{(4)} &= F'''X'^3 + 2F''X''X' + F'^2X'' + 2F''F'X'^2 \\ &= F'''X'^3 + 4F''F'X'^2 + F'^3X'. \end{aligned}$$

Now a comparison shows that (25) and (26) match up mod  $O(h^4)$ , but

$$(28) \quad [26] = [25] + \frac{h^4}{24}F'^3F + O(h^5),$$

where [N] stands for the content of formula (N).

Thus we will get a 4<sup>th</sup> order accurate scheme if we can modify  $L_{n4}$  by adding  $\frac{h^3}{4}F'^3F + O(h^4)$ . Recalling that  $L_{n4} = F(X_n + hF(X_n + \frac{h}{2}F))$ , note that

$$(29) \quad F(X_n + hF(X_n + \frac{h}{2}F + h^2A)) = L_{n4} + h^3F'^2A + O(h^4),$$

so we pick  $A = \frac{1}{4}F'F + O(h)$ , so  $h^2A = \frac{h^2}{4}F'F + O(h^3)$ . Now

$$(30) \quad \frac{h}{2}F + \frac{h^2}{4}F'F = \frac{h}{2}K_{n2} + O(h^3)$$

by (24), so

$$(31) \quad F\left(X_n + hF\left(X_n + \frac{h}{2}K_{n2}\right)\right) = L_{n4} + \frac{h^3}{4}F'^3F + O(h^4).$$

Now the left side of (31) is precisely  $F(X_n + hK_{n3}) = K_{n4}$ . Thus we have produced the Runge-Kutta scheme (2) and shown that it is 4<sup>th</sup> order accurate.

### Computer program example

We apply the Runge-Kutta scheme to the study of the central force problem used as an example in the last section, i.e., to

$$(11) \quad \begin{aligned} x' &= v \\ y' &= w \\ v' &= -x(x^2 + y^2)^{-\frac{3}{2}} \\ w' &= -y(x^2 + y^2)^{-\frac{3}{2}}. \end{aligned}$$

This time, the iteration, from  $(X, V)(t) = (x, y, v, w)$  to  $(X, V)(t + h)$ , is accomplished by setting

$$(12) \quad a_1 = f(x, y) = -x(x^2 + y^2)^{-\frac{3}{2}}, \quad b_1 = g(x, y) = -y(x^2 + y^2)^{-\frac{3}{2}},$$

and

$$(13) \quad x_2 = x + \frac{h}{2}v, \quad y_2 = y + \frac{h}{2}w, \quad v_2 = v + \frac{h}{2}a_1, \quad w_2 = w + \frac{h}{2}b_1,$$

then setting

$$(14) \quad a_2 = f(x_2, y_2), \quad b_2 = g(x_2, y_2)$$

and

$$(15) \quad x_3 = x + \frac{h}{2}v_2, \quad y_3 = y + \frac{h}{2}w_2, \quad v_3 = v + \frac{h}{2}a_2, \quad w_3 = w + \frac{h}{2}b_2,$$

then setting

$$(16) \quad a_3 = f(x_3, y_3), \quad b_3 = g(x_3, y_3)$$

and

$$(17) \quad x_4 = x + hv_3, \quad y_4 = y + hw_3, \quad v_4 = v + ha_3, \quad w_4 = w + hb_3,$$

then

$$(18) \quad a_4 = f(x_4, y_4), \quad b_4 = g(x_4, y_4).$$

Finally,

$$(19) \quad \begin{aligned} x(t+h) &= x + \frac{h}{6}(v + 2v_2 + 2v_3 + v_4) \\ y(t+h) &= y + \frac{h}{6}(w + 2w_2 + 2w_3 + w_4) \\ v(t+h) &= v + \frac{h}{6}(a_1 + 2a_2 + 2a_3 + a_4) \\ w(t+h) &= w + \frac{h}{6}(b_1 + 2b_2 + 2b_3 + b_4). \end{aligned}$$

This iterative method is used in the program written for Borland's Turbo C, listed below. There is also a listing of the analogous program for Borland's Turbo Pascal, version 5.5. As before, the same sort of program works for a general ODE of the form

$$(20) \quad (x'', y'') = (f(x, y), g(x, y)).$$

The reader might experiment, changing the definition of  $f(x, y)$  and of  $g(x, y)$ , to study a variety of such ODEs, modeling problems in Newtonian mechanics with two degrees of freedom.

Accompanying the program listing is a graph of the orbit produced. Note how much more accurately it depicts the true closed elliptical orbit than was done using the second order accurate scheme in the last section.

### C program using Runge-Kutta

```
# include <graphics.h>
# include <math.h>
# include <screen.c>
double fnf(double x, double y);
double fng(double x, double y);
main()
{
double x,y,v,w,h,ha,hs;
double x1,x2,x3,x4,y1,y2,y3,y4;
double v1,v2,v3,v4,w1,w2,w3,w4;
double a1,a2,a3,a4,b1,b2,b3,b4;
int j,kx,ky;
picture(0);
```

```

x=.7;
y=0;
v=0;
w=.8;
h=.01;
ha=h/2;
hs=h/6;
putpixel(320,240,4);
for (j=1;j<=2500;j++){
    kx=320+240*x;
    ky=240-240*y;
    putpixel(kx,ky,2);
    v1=v;
    w1=w;
    a1=fnf(x,y);
    b1=fng(x,y);
    v2=v+ha*a1;
    w2=w+ha*b1;
    x2=x+ha*v1;
    y2=y+ha*w1;
    a2=fnf(x2,y2);
    b2=fng(x2,y2);
    v3=v+ha*a2;
    w3=w+ha*b2;
    x3=x+ha*v2;
    y3=y+ha*w2;
    a3=fnf(x3,y3);
    b3=fng(x3,y3);
    v4=v+h*a3;
    w4=w+h*b3;
    x4=x+h*v3;
    y4=y+h*w3;
    a4=fnf(x4,y4);
    b4=fng(x4,y4);
    x=x+hs*(v1+2*v2+2*v3+v4);
    y=y+hs*(w1+2*w2+2*w3+w4);
    v=v+hs*(a1+2*a2+2*a3+a4);
    w=w+hs*(b1+2*b2+2*b3+b4);
}
}

double fnf(double x, double y)
{
double r,s,z;

```

```

s=x*x+y*y;
r=s*sqrt(s);
z=-x/r;
return z;
}

double fng(double x, double y)
{
double r,s,z;
s=x*x+y*y;
r=s*sqrt(s);
z=-y/r;
return z;
}

```

### Pascal program using Runge-Kutta

```

Program orbit1;
Uses
  CRT,Graph;

Var
  x,y,v,w,h,ha,hs: double;
  x1,x2,x3,x4,y1,y2,y3,y4:double;
  v1,v2,v3,v4,w1,w2,w3,w4:double;
  a1,a2,a3,a4,b1,b2,b3,b4:double;
  r,s,z:double;
  j,kx,ky:integer;
  {$I picture.pas}

Function f(x,y:double): double;
begin
  s:=x*x+y*y;
  r:=s*sqrt(s);
  f:=-x/r
end;

Function g(x,y:double): double;
begin
  s:=x*x+y*y;
  r:=s*sqrt(s);
  g:=-y/r
end;

```

```
Begin
picture(0,2);
x:=0.7;
y:=0;
v:=0;
w:=0.8;
h:=0.01;
ha:=h/2;
hs:=h/6;
Putpixel(320,240,4);

for j:=1 to 12500 do
begin
  kx:=trunc(320+240*x);
  ky:=trunc(240-240*y);
  Putpixel(kx,ky,2);
  v1:=v;
  w1:=w;
  a1:=f(x,y);
  b1:=g(x,y);
  v2:=v+ha*a1;
  w2:=w+ha*b1;
  x2:=x+ha*v1;
  y2:=y+ha*w1;
  a2:=f(x2,y2);
  b2:=g(x2,y2);
  v3:=v+ha*a2;
  w3:=w+ha*b2;
  x3:=x+ha*v2;
  y3:=y+ha*w2;
  a3:=f(x3,y3);
  b3:=g(x3,y3);
  v4:=v+h*a3;
  w4:=w+h*b3;
  x4:=x+h*v3;
  y4:=y+h*w3;
  a4:=f(x4,y4);
  b4:=g(x4,y4);
  x:=x+hs*(v1+2*v2+2*v3+v4);
  y:=y+hs*(w1+2*w2+2*w3+w4);
  v:=v+hs*(a1+2*a2+2*a3+a4);
```

```
w:=w+hs*(b1+2*b2+2*b3+b4);  
end;  
end.
```

## References

- [At] K. Atkinson, An Introduction to Numerical Analysis, Wiley, New York, 1978.
- [BD] W. Boyce and R. DiPrima, Elementary Differential Equations and Boundary Value Problems, Wiley, New York, 1969.
- [BR] G. Birkhoff and G.-C. Rota, Ordinary Differential Equations, Wiley, New York, 1978.
- [CK] W. Cheney and D. Kincaid, Numerical Mathematics and Computing, Brooks/Cole, Monterey, Calif., 1980.
- [Ki] J. King, Introduction to Numerical Computation, McGraw-Hill, New York, 1984.
- [PFTV] W. Press, B. Flannery, S. Teukolsky, and W. Vetterling, Numerical Recipes, Cambridge, 1987.
- [T] M. Taylor, Turbo Pascal in a Hurry for Mathematical Computation, MET Press, 1990.