Sudan University of Science &Technology

College of Graduate Studies

Numerical Methods for Solving Some Types of Differential Equations

الطرق العددية لحل بعض أنواع المعادلات التفاضلية

By:

Diaa Eldeen Mohamed Ahmed Elgzoley Boshra

Supervisor:

Dr. Mohamed Hassan Mohamed Khabir

2015

الآية

قال تعالى:

بسم الله الرحمن الرحيم

﴿ربِ أوزعني أن أشكُرَ نِعمَتَكَ التي أنعَمتَ عَلَيَ وَعَلي وَالِدَيَ وَأن أعمَلَ صالِحآ تَرضَاهُ وَأدخِلني بِرَحمَتِكَ في عِبادِكَ الصَالِحينَ﴾

صدق الله العظيم

سورة النمل الآية (19)

DEDICATION

To my mother .....My first teacher

To my father .... My hero

To my brothers, sisters

To my friends

To all those unbelievable persons

I am trying to say thank you

Acknowledgement

Firstly, I am deeply thankful to Allah for his almighty...

Also I am grateful to Dr. ***Mohamed Hassan Mohamed Khabir***

For his advices and help until I finished this study...

Finally thanks for every person in my life...

The contains

Abstract:

We will Apply this thesis to apply some numerical methods for solving ordinary and partial differential equations. In chapter One we solve numerically some ordinary differential equations using Euler and improved Euler methods, then we apply Runge Kutta method. In chapter Two, we find the numerical solution for systems of ordinary differential equation using same methods used in chapter one. We discuss some ways for the solution of partial differential equation using finite difference methods; In particular we solve a parabolic type of PDEs namely the heat equation using Explicit, Implicit and Crank Nicholson method. For the solution we use the programming language (Matlab) and we write some programs in The appendix.

الخلاصة:

تناولنا في هذا البحث بعض الطرق العددية لحل المعادلات التفاضلية العادية والجزئية. ناقشنا في الباب الاول الحلول العددية للمعادلات التفاضلية العادية باستخدام طريقة اويلر و طريقة اويلر المحسنة ثم طريقة رونج كوتا.وناقشنا في الباب الثاني الحلول العددية لأنظمة المعادلات التفاضلية باستخدام نفس الطرق السابقة المستخدمة في الباب الأول.في الباب الثالث ناقشنا بعض الطرق العددية لحل المعادلات التفاضلية الجزئية باستخدام الفروق المقسمة حيث تناولنا فيها المعادلات التفاضلية الجزئية المكافئة وكمثال اخذنا معادلة الحرارة واوجدنا حلها باستخدام الفروق الأمامية والمركزية والخلفية.أخيراً أرفقنا هذا البحث ببرامج لهذه الطرق باستخدام برنامج الماتلاب.

Chapter one

Numerical Methods for ODEs

Chapter 1

Numerical Methods from ODEs

1.1 The Euler Method[1]

It is the exception rather than the rule when a differential equation of the general form $\frac{dy}{dx} = f(x, y)$ can be solved exactly and explicitly by elementary methods. For example, consider the simple equation

$$\frac{dy}{dx} = e^{-x^2}. \tag{1.1}$$

A solution of equation (1.1) is simply an anti-derivative of e^{-x^2}. But it is known that every antiderivative of $f(x) = e^{-x^2}$ is a nonelementary function-one that cannot be expressed as a finite combination of the familiar functions of elementary calculus. Hence no particular solution of equation (1.1) is finitely expressible in terms of elementary functions.

As a possible alternative, an old-fashioned computer plotter-one that uses an ink pen to draw curves mechanically-can be programmed to draw a solution curve that starts at the initial point (x_0, y_0) and attempts to thread its way through the slope field of a given differential equation $y' = I(x, y)$. The procedure the plotter carries out can be described as follows.

- The plotter pen starts at the initial point (x_0, y_0) and moves a tiny distance along the slope segment through (x_0, y_0). This takes it to the point (x_1, y_1).
- At (x_1, y_1) the pen changes direction, and now moves a tiny distance along the slope segment through this new starting point (x_1, y_1). This takes it to the next starting point (x_2, y_2).
- At (x_2, y_2) the pen changes direction again, and now moves a tiny distance along the slope segment through (x_2, y_2) . This takes it to the next starting point (x_3, y_3) .

Figure 1.1 illustrates the result of continuing in this fashion-by a sequence of discrete straight-line steps from one starting point to the next. In this figure we see a polygonal curve consisting of line segments that connect the successive points

$(x_0, y_0), (x_1, y_1), (x_2, y_2), (x_3, y_3)$, ... However, suppose that each "tiny distance" the pen travels along a slope segment before the midcourse correction that sends it along a fresh new slope segment is so very small that the naked eye cannot distinguish the individual line segments constituting the polygonal curve. Then the resulting polygonal curve looks like a smooth, continuously turning solution curve of the differential equation.

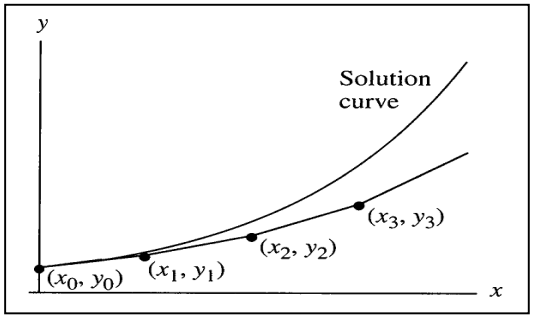

Figure 1.1 The first few steps in approximating a solution curve.

Leonhard Euler-the great 18th-century mathematician for whom so many mathematical concepts, formulas, methods, and results are named-did not have a computer plotter, and his idea was to do all this numerically rather than graphically. In order to approximate the solution of the initial value problem

$$\frac{dy}{dx} = f(x, y), \quad y(x_0) = y_0, \tag{1.2}$$

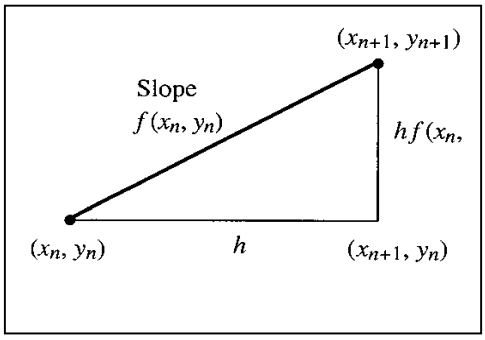

Figure1.2 The step from (x_n, y_n) to (x_{n+1}, y_{n+1}).

We first choose a fixed (horizontal) step size h to use in making each step from one point to the next. Suppose we've started at the initial point (x_0, y_0) and after n steps have reached the point (x_n, y_n). Then the step from (x_n, y_n) to the next point (x_{n+1}, y_{n+1}) is illustrated in Figure 1.2 The slope of the direction segment through (x_n, y_n) is $m = f(x_n, y_n)$. Hence a horizontal change of h from x_n to x_{n+1} corresponds to a vertical change of $m \cdot h = h \cdot f(x_n, y_n)$ from y_n to y_{n+1}. Therefore the coordinates of the new point (x_{n+1}, y_{n+1}) are given in terms of the old coordinates by

$$x_{n+1} = x_n + h, \;\; y_{n+1} = y_n + h \cdot f(x_n, y_n).$$

Given the initial value problem in (2), Euler's method with step size h consists of starting with the initial point (x_0, y_0) and applying the formulas

$$\begin{array}{ll} x_1 = x_0 + h & y_1 = y_0 + h \cdot f(x_0, y_0) \\ x_2 = x_1 + h & y_2 = y_1 + h \cdot f(x_1, y_1) \\ x_3 = x_2 + h & y_3 = y_2 + h \cdot f(x_2, y_2) \\ \vdots & \vdots \\ \vdots & \vdots \end{array}$$

To calculate successive points $(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots$ on an approximate solution curve.

However, we ordinarily do not sketch the corresponding polygonal approximation. Instead, the numerical result of applying Euler's method is the sequence of approximations

$$y_1, y_2, y_3, \ldots, y_n, \ldots$$

to the true values

$$y(x_1), y(x_2), y(x_3), \ldots, y(x_n), \ldots$$

at the points $x_1, x_2, x_3, \ldots, x_n$ ... of the exact (though unknown) solution $y(x)$ of the initial value problem. These results typically are presented in the form of a table of approximate values of the desired solution.

Algorithm: The Euler Method

Given the initial value problem

$$\frac{dy}{dx} = f(x,y), \;\; y(x_0) = y_0, \qquad (1.2)$$

Euler's method with step size h consists of applying the iterative formula

$$y_{n+1} = y_n + h \cdot f(x_n, y_n) \quad (n \geq 0) \qquad (1.3)$$

to calculate successive approximations $y_1, y_2, y_3, \ldots$ to the [true] values $y(x_1), y(x_2), y(x_3), \ldots$, of the [exact] solution $y = y(x)$ at the points $x_1, x_2, x_3, \ldots$, respectively.

The iterative formula in (1.3) tells us how to make the typical step from y_n to y_{n+1} and is the heart of Euler's method. Although the most important applications of Euler's method are to nonlinear equations, we first illustrate the method with a simple initial value problem whose exact solution is available, just for the purpose of comparison of approximate and actual solutions.

Example (1.1):

Apply Euler's method to approximate the solution of the initial value problem

$$\frac{dy}{dx} = x + \frac{1}{5}y, \;\; y(0) = -3, \qquad (1.4)$$

a) first with step size $h = 1$ on the interval [0,5],
b) then with step size $h = 0.2$ on the interval [0,1] .

Solution:

a) With $x_0 = 0, y_0 = -3, f(x,y) = x + \frac{1}{5}y$, and $h = 1$ the iterative formula in (1.3) yields the approximate values

$$y_1 = y_0 + h \cdot \left[x_0 + \frac{1}{5}y_0\right] = (-3) + (1)\left[0 + \frac{1}{5}(-3)\right] = -3.6,$$
$$y_2 = y_1 + h \cdot \left[x_1 + \frac{1}{5}y_1\right] = (-3.6) + (1)\left[1 + \frac{1}{5}(-3.6)\right] = -3.32,$$
$$y_3 = y_2 + h \cdot \left[x_2 + \frac{1}{5}y_2\right] = (-3.32) + (1)\left[2 + \frac{1}{5}(-3.32)\right] = -1.984,$$
$$y_4 = y_3 + h \cdot \left[x_3 + \frac{1}{5}y_3\right] = (-1.984) + (1)\left[3 + \frac{1}{5}(-1.984)\right] = 0.6192, \text{and}$$
$$y_5 = y_4 + h \cdot \left[x_4 + \frac{1}{5}y_4\right] = (0.6912) + (1)\left[4 + \frac{1}{5}(0.6912)\right] = 4.7430,$$

at the points $x_1 = 1, x_2 = 2, x_3 = 3, x_4 = 4$, and $x_5 = 5$. Note how the result of each calculation feeds into the next one. The resulting table of approximate values in

x	0	1	2	3	4	5
Approx. y	-3	-3.6	-3.32	-1.984	0.6912	4.7430

Table1.1

Figure 1.3 shows the graph of this approximation, together with the graphs of the Euler approximations obtained with step sizes $h = 0.2$ and 0.05, as well as the graph of the exact solution

$$y(x) = 22e^{x/5} - 5x - 25$$

that is readily found using the linear-equation technique. We see that decreasing the step size increases the accuracy, but with any single approximation, the accuracy decreases with distance from the initial point.

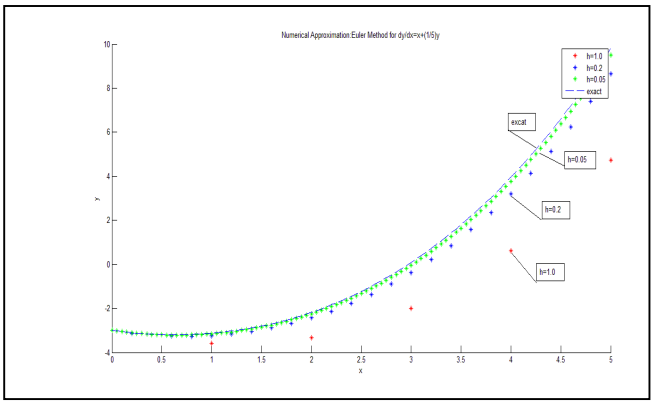

Figure 1.3 Graphs of Euler approximations with step sizes $h = 1, h = 0.2$, and $h = 0.05$.

b) Starting afresh with $x_0 = 0, y_0 = -3, f(x,y) = x + \frac{1}{5}y$, and $h = 0.2$, we get the approximate values

$$y_1 = y_0 + h \cdot \left[x_0 + \tfrac{1}{5}y_0\right] = (-3) + (0.2)\left[0 + \tfrac{1}{5}(-3)\right] \approx -3.12,$$

$$y_2 = y_1 + h \cdot \left[x_1 + \tfrac{1}{5}y_1\right] = (-3.12) + (0.2)\left[0.2 + \tfrac{1}{5}(-3.12)\right] \approx -3.205,$$

$$y_3 = y_2 + h \cdot \left[x_2 + \tfrac{1}{5}y_2\right] = (-3.205) + (0.2)\left[0.4 + \tfrac{1}{5}(-3.205)\right] \approx -3.253,$$

$$y_4 = y_3 + h \cdot \left[x_3 + \tfrac{1}{5}y_3\right] = (-3.253) + (0.2)\left[0.6 + \tfrac{1}{5}(-3.253)\right] \approx -3.263,$$

$$y_5 = y_4 + h \cdot \left[x_4 + \tfrac{1}{5}y_4\right] = (-3.263) + (0.2)\left[0.8 + \tfrac{1}{5}(-3.263)\right] \approx -3.234,$$

at the points $x_1 = 0.2, x_2 = 0.4, x_3 = 0.6, x_4 = 0.8$, and $x_5 = 1$. The resulting table of approximate values in

Table 1.2

x	0	0.2	0.4	0.6	0.8	1
Approx. y	-3	-3.12	-3.205	-3.253	-3.263	-3.234

High accuracy with Euler's method usually requires a very small step size and hence a larger number of steps than can reasonably be carried out by hand. The application material for this section contains calculator and computer programs for automating Euler's method. One of these programs was used to calculate the table entries shown in Table1.3 We see that 500 Euler steps (with step size $h = 0.002$) from $x = 0$ to $x = 1$ yield values that are accurate to within 0.001.

Example (1.2):[1]

Suppose the baseball in [1] is simply dropped (instead of being thrown downward) from the helicopter . Then its velocity $v(t)$ after t seconds satisfies the initial value problem

$$\frac{dv}{dt} = 32 - 0.16v, \quad v(0) = 0. \qquad (1.5)$$

We use Euler's method with $h = 1$ to track the ball's increasing velocity at 1-second intervals for the first 10 seconds of fall with $t_0 = 0, v_0 = 0, \quad F(t, v) = 32 - 0.16v$,

x	Approx y with $h = 0.2$	Approx y with $h = 0.02$	Approx y with $h = 0.002$	Actual value of y
0	-3.000	-3.000	-3.000	-3.000
0.2	-3.120	-3.104	-3.102	-3.102
0.4	-3.205	-3.172	-3.168	-3.168
0.6	-3.253	-3.201	-3.196	-3.195
0.8	-3.263	-3.191	-3.184	-3.183
1	-3.234	-3.140	-3.130	-3.129

Table 1.3 Euler approximations with step sizes $h = 0.2, h = 0.02$, and $h = 0.002$.

and $h = 1$ the iterative formula in (1.3) yields the approximate values

$v_1 = v_0 + h \cdot [32 - 0.16v_0] = (0) + (1)[32 - 0.16(0)] \approx 32,$
$v_2 = v_1 + h \cdot [32 - 0.16v_1] = (32) + (1)[32 - 0.16(32)] \approx 32,$
$v_3 = v_2 + h \cdot [32 - 0.16v_2] = (58.88) + (1)[32 - 0.16(58.88)] \approx 81.46,$
$v_4 = v_3 + h \cdot [32 - 0.16v_3] = (81.46) + (1)[32 - 0.16(81.46)] \approx 100.43,$ and
$v_5 = v_4 + h \cdot [32 - 0.16v_4] = (100.43) + (1)[32 - 0.16(100.43)] \approx 116.36,$

Continuing in this fashion, we complete the $h = 1$ column of v-values shown in the table of Table1.4-where we have rounded off velocity entries to the nearest foot per second. The values corresponding to $h = 0.1$ were calculated using a computer, and we see that they are accurate to within about 1 ft/s. Note also that after 10 seconds the falling ball has attained about 80% of its limiting velocity of 200 ft/s.

t	Approx v with $h = 1$	Approx y with $h = 0.1$	Actual value of v
1	32	30	30
2	59	55	55
3	81	77	76
4	100	95	95
5	116	111	110
6	130	124	123
7	141	135	135
8	150	145	144
9	158	153	153
10	165	160	160

Table 1.4 Euler approximations in Example 2 with step sizes $h = 1$ and $h = 0.1$.

Local and Cumulative Errors:

There are several sources of error in Euler's method that may make the approximation y_n to $y(x_n)$ unreliable for large values of n , those for which x_n is not sufficiently close to x_0. The error in the linear approximation formula

$$y(x_{n+1}) \approx y_n + h \cdot f(x_n, y_n) = y_{n+1} \qquad (1.6)$$

is the amount by which the tangent line at (x_n, y_n) departs from the solution curve through (x_n, y_n) , as illustrated in Fig1.4 This error, introduced at each step in the process, is called the local error in Euler's method.

The local error indicated in Fig1.4 would be the total error in y_{n+1} if the starting point y_n in (1.6) were an exact value, rather than merely an approximation to the actual value (x_n) . But y_n itself suffers from the accumulated effects of all the local errors introduced at the previous steps. Thus the tangent line in Fig1.4 is tangent to the "wrong" solution curve-the one through (x_n, y_n) rather than the actual solution curve through the initial point (x_0, y_0). Figure1.5 illustrates this cumulative error in Euler's method; it is the amount by which the polygonal stepwise path from (x_0, y_0) departs from the actual solution curve through (x_0, y_0).

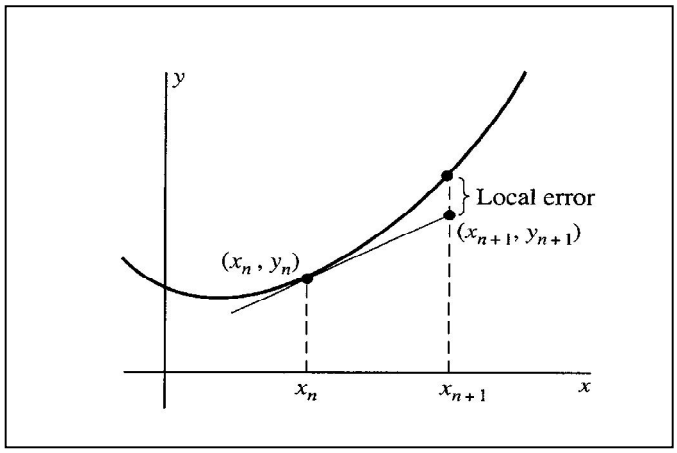

Figure1.4 The local error in Euler's method

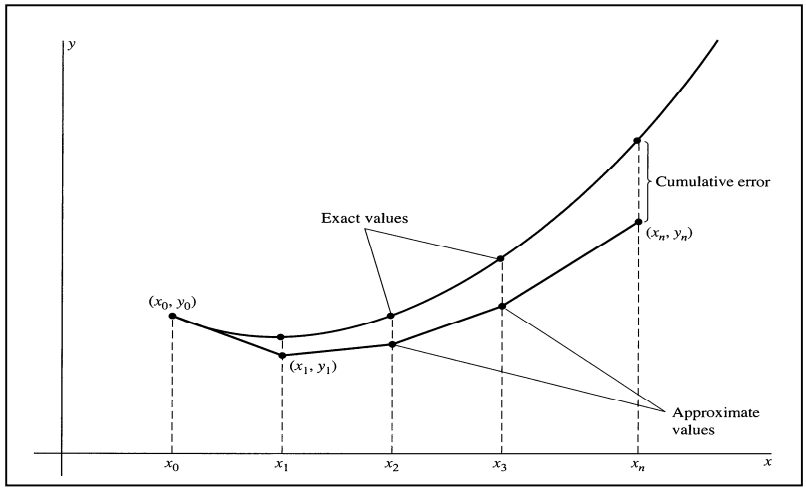

Figure 1.5 The cumulative error in Euler's method.

The usual way of attempting to reduce the cumulative error in Euler's method is to decrease the step size h . The table inTable1.5 shows the results obtained in approximating the exact solution $y(x) = 2e^x - x - 1$ of the initial value problem

$$\frac{dy}{dx} = x + y, \qquad y(0) = 1\,,$$

using the successively smaller step sizes $h = 0.1, h = 0.02, h = 0.005,$ and $h = 0.001$. We show computed values only at intervals of $\Delta = 0.1$. For instance,

with $h = 0.001$, the computation required 1000 Euler steps, but the value y_n is shown only when n is a multiple of 100, so that x_n is an integral multiple of 0.1.

B y scanning the columns in Table1.5 we observe that, for each fixed step size h, the error $y_{\text{actual}} - y_{\text{approx}}$ increases as x gets farther from the starting point $x_0 = 0$. But by scanning the rows of the table we see that for each fixed x, the error decreases as the step size h is reduced. Thus the smaller the step size, the more slowly does the error grow with increasing distance from the starting point.

x	y with $h = 0.1$	y with $h = 0.02$	y with $h = 0.005$	y with $h = 0.001$	Actual y
0.1	1.1000	1.1082	1.1098	1.1102	1.1103
0.2	1.2200	1.2380	1.2416	1.2426	1.2428
0.3	1.3620	1.3917	1.3977	1.3993	1.3997
0.4	1.5282	1.5719	1.5807	1.5831	1.5836
0.5	1.7210	1.7812	1.7933	1.7966	1.7974
0.6	1.9461	2.0227	2.0388	2.0431	2.0442
0.7	2.1974	2.2998	2.3205	2.3261	2.3275
0.8	2.4872	2.6161	2.6422	2.6493	2.6511
0.9	2.8159	2.9757	3.0082	3.0170	3.0192
1.0	3.1875	3.3832	3.4230	3.4238	3.4266

Table 1.5 Approximating the solution of $dy/dx = x + y, y(0) = 1$ with successively smaller step sizes.

The column of data for $h = 0.1$ in Table1.5 requires only 10 steps, so Euler's method can be carried out with a hand-held calculator. But 50 steps are required to reach $x = 1$ with $h = 0.02$, 200 steps with $h = 0.005$, and 1000 steps with $h = 0.001$. A computer is almost always used to implement Euler's method when more than 10 or 20 steps are required. Once an appropriate computer program has been written, One step size is-in principle-just as convenient as another; after all, the computer hardly cares how many steps it is asked to carry out.

Why, then, do we not simply choose an exceedingly small step size (such as $h = 10^{-12}$), with the expectation that very great accuracy will result? There are two reasons for not doing so. The first is obvious: the time required for the computation. For example, the data in Table1.5 were obtained using a hand-held calculator that carried out nine Euler steps per second. Thus it required slightly

over one second to approximate $y(1)$ with $h = 0.1$ and about 1 min 50 sec with $h = 0.001$. But with $h = 10^{-12}$ it would require over 3000 years !

The second reason is more subtle. the computer itself will contribute round off error at each stage because only finitely many significant digits can be used in each calculation. An Euler's method computation with $h = 0.0001$ will introduce roundoff errors 1000 times as often as one with $h = 0.1$. Hence with certain differential equations, $h = 0.1$ might actually produce more accurate results than those obtained with $h = 0.0001$, because the cumulative effect of roundoff error in the latter case might exceed combined cumulative and round off error in the case $h = 0.1$.

The "best" choice of h is difficult to determine in practice as well as in theory. It depends on the nature of the function $f(x, y)$ in the initial value problem in (1.2), on the exact code in which the program is written, and on the specific computer used. With a step size that is too large, the approximations inherent in Euler's method may not be sufficiently accurate, whereas if h is too small, then roundoff errors may accumulate to an unacceptable degree or the program may require too much time to be practical. The subject of error propagation in numerical algorithms is treated in numerical analysis courses and books.

The computations in Table1.5 illustrate the common strategy of applying a numerical algorithm, such as Euler's method, several times in succession, beginning with a selected number n of subintervals for the first application, then doubling n for each succeeding application of the method. Visual comparison of successive results often can provide an "intuitive feel" for their accuracy. In the next two examples we present graphically the results of successive applications of Euler's method.

Example (1.3):

The exact solution of the logistic initial value problem

$$\frac{dy}{dx} = \frac{1}{3}y(8 - y), \qquad y(0) = 1$$

is $y(x) = 8/\left(1 + 7e^{-8x/3}\right)$. Figure1.6 shows both the exact solution curve and approximate solution curves obtained by applying Euler's method on the interval

$0 \leqq x \leqq 5$ with $n = 5, n = 10$, and $n = 20$ subintervals. Each of these "curves" actually consists of line segments joining successive points (x_n, y_n) and $(x_{n+1} y_{n+1})$. The Euler approximation with 5 subintervals is poor, and the approximation with 10 subintervals also overshoots the limiting value $y = 8$ of the solution before leveling off, but with 20 subintervals we obtain fairly good qualitative agreement with the actual behavior of the solution.

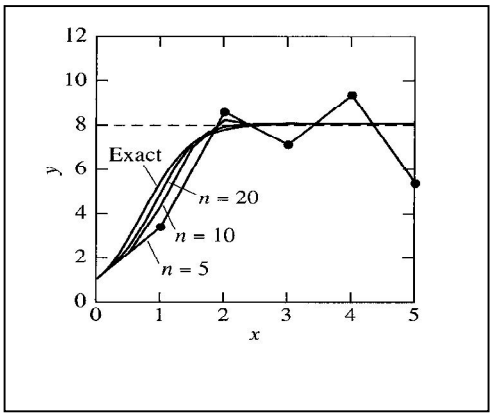

Figure 1.6 Approximating a logistic solution using Euler's method with $n = 5, n = 10$, and $n = 20$ subintervals.

Example (1.4):

The exact solution of the initial value problem

$$\frac{dy}{dx} = y\cos x, \qquad y(0) = 1$$

Is the periodic function $y(x) = e^{\sin x}$. Figure 1.7 shows both the exact solution curve and approximate solution curves obtained by applying Euler's method on the interval $0 \leqq x \leqq 6\pi$ with $n = 50, n = 100, n = 200$, and $n = 400$ subintervals. Even with this many subintervals, Euler's method evidently has considerable difficulty keeping up with the oscillations in the actual solution. Consequently, the more accurate methods discussed in succeeding sections are needed for serious numerical investigations.

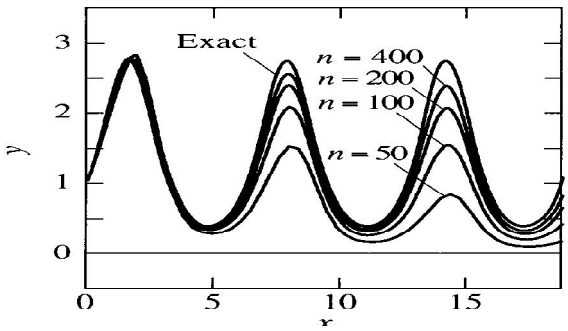

Figure1.7 Approximating the exact solution $y = e^{\sin x}$ using Euler's method with 50,100,200, and 400 subinterval.

A Word of Caution:

The data shown in Table1.5 indicate that Euler's method works well in approximating the solution of $dy/dx = x + y, y(0) = 1$ on the interval [0,1]. That is, for each fixed x it appears that the approximate values approach the actual value of $y(x)$ as the step size h is decreased. For instance, the approximate values in the rows corresponding to $x = 0.3$ and $x = 0.5$ suggest that $y(0.3) \approx 1.40$ and $y(0.5) \approx 1.80$, in accord with the actual values shown in the final column of the table.

Example (**1.5**), in contrast, shows that some initial value problems are not so well behaved.

Example (1.5):

Use Euler's method to approximate the solution of the initial value problem

$$\frac{dy}{dx} = x^2 + y^2, \qquad y(0) = 1 \tag{1.7}$$

on the interval [0,1].

Solution:

Here $f(x, y) = x^2 + y^2$, so the iterative formula of Euler's method is

$$y_{n+1} = y_n + h \cdot (x_n^2 + y_n^2). \tag{1.8}$$

With step size $h = 0.1$ we obtain

$$y_1 = 1 + (0.1) \cdot [(0)^2 + (1)^2] = 1.1,$$
$$y_2 = 1.1 + (0.1) \cdot [(0.1)^2 + (1.1)^2] = 1.222,$$
$$y_3 = 1.222 + (0.1).[(0.2)^2 + (1.222)^2]1.3753,$$

and so forth. Rounded to four decimal places, the first ten values obtained in this manner are

$$y_1 = 1.1000 \quad y_6 = 2.1995$$
$$y_2 = 1.2220 \quad y_7 = 2.7193$$
$$y_3 = 1.3753 \quad y_8 = 3.5078$$
$$y_4 = 1.5735 \quad y_9 = 4.8023$$
$$y_5 = 1.8371 \quad y_{10} = 7.1895$$

But instead of naively accepting these results as accurate approximations, we decided to use a computer to repeat the computations with smaller values of h. The table in Table1.6 shows the results obtained with step sizes $h = 0.1, h = 0.02,$ and $h = 0.005$. Observe that now the "stability" of the procedure in Example (1) is missing. Indeed, it seems obvious that something is going wrong near $x = 1$.

Figure 1.8 provides a graphical clue to the difficulty. It shows a slope field for $dy/dx = x^2 + y^2$, together with a solution curve through $(0,1)$ plotted using one of the more accurate approximation methods of the following two sections. It appears that this solution curve may have a vertical asymptote near $x = 0.97$. Indeed, an exact solution using Bessel functions (see [1]) can be used to show that $y(x) \to +\infty$ as $x \to 0.969811$ (approximately). Although Euler's method gives values (albeit spurious ones) at $x = 1$, the actual solution does not exist on the entire interval $[0,1]$. Moreover, Euler's method is unable to "keep up" with the rapid changes in $y(x)$ that occur as x approaches the infinite discontinuity near 0.969811.

x	y with $h = 0.1$	y with $h = 0.2$	y with $h = 0.005$
0.1	1.1000	1.1088	1.1108
0.2	1.2220	1.2458	1.2512
0.3	1.3753	1.4243	1.4357
0.4	1.5735	1.6658	1.6882
0.5	1.8371	2.0074	2.0512
0.6	2.1995	2.5201	2.6104
0.7	2.7193	3.3612	3.5760
0.8	3.5078	4.9601	5.5763
0.9	4.8023	9.000	12.2061
1.0	7.1895	30.9167	1502.2090

Table1.6 Attempting to approximate the solution of $\frac{dy}{dx} = x^2 + y^2, y(0) = 1$.

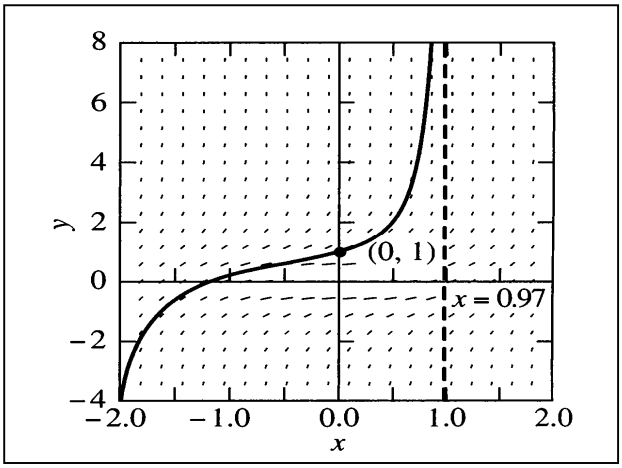

Figure1.8 Solution of $dy/dx = x^2 + y^2, y(0) = 1$

The moral of Example (1.5) is that there are pitfalls in the numerical solution of certain initial value problems. Certainly it's pointless to attempt to approximate a solution on an interval where it doesn't even exist (or where it is not unique, in which case there's no general way to predict which way the numerical approximations will branch at a point of nonuniqueness). One should never accept as accurate the results of applying Euler's method with a single fixed step size h. A second "run" with smaller step size ($h/2$, say, or $h/5$, or $h/10$) may give seemingly consistent results, thereby suggesting their accuracy, or it may-as in

Example (5)-reveal the presence of some hidden difficulty in the problem. Many problems simply require the more accurate and powerful methods that are discussed in the final two sections of this chapter.

Application: Implementing Euler's Method

Construction of a calculator or computer program to implement a numerical algorithm can sharpen one's understanding of the algorithm. Program 1 implementing Euler's method to approximate the solution of the initial value problem

$$\frac{dy}{dx} = x + y, \qquad y(0) = 1$$

To increase the number of steps (and thereby decrease the step size) you need only change the value of N specified in the the program [1]. To apply Euler's method to a different equation $dy/dx = f(x, y)$, you need change only the single line that calculates the function value F.

For instance, the MATLAB command

$$[\mathrm{X\ Y}] = \mathrm{euler}\ (0, 1, 1, 10)$$

Then generates the x_n - and y_n -data shown in the first two columns of the table of Table 1.5

You should begin this project by implementing Euler's method with your own calculator or computer system. Test your program by first applying it to the initial value problem in Example (1), then to some of the problems for this section.

Famous Numbers Investigation:

The following problems describe the numbers $e \approx 2.71828$, Ln $2 \approx 0.69315$, and $\pi \approx 3.14159$ as specific values of solutions of certain initial value problems. In each case, apply Euler's method with $n = 50,100,200,\ldots$ subintervals (doubling n each time). How many subintervals are needed to obtain-twice in succession- the correct value of the target number rounded to three decimal places?

1. The number $e = y(1)$, where $y(x)$ is the solution of the initial value problem $dy/dx = y, y(0) = 1$.
2. The number in $2 = y(2)$, where $y(x)$ is the solution of the initial value problem $dy/dx = 1/x, y(1) = 0$.
3. The number $\pi = y(l)$, where $y(x)$ is the solution of the initial value problem $dy/dx = 4/(1 + x^2), y(0) = 0$.

Also explain in each problem what the point is-why the indicated famous number is the expected numerical result.

1.2 A Closer Look at the Euler Method:

The Euler method as presented in Section1 is not often used in practice, mainly because more accurate methods are available. But Euler's method has the advantage of simplicity, and a careful study of this method yields insights into the workings of more accurate methods, because many of the latter are extensions or refinements of the Euler method.

To compare two different methods of numerical approximation, we need some way to measure the accuracy of each. Theorem (1) tells what degree of accuracy we can expect when we use Euler's method. [5]

Theorem (1): The Error in the Euler Method

Suppose that the initial value problem

$$\frac{dy}{dx} = f(x, y), \qquad y(x_0) = y_0 \tag{1.9}$$

has a unique solution $y(x)$ on the closed interval $[a, b]$ with $a = x_0$, and assume that $y(x)$ has a continuous second derivative on $[a, b]$. (This would follow from the assumption that f, f_x, and f_y are all continuous for $a \leqq x \leqq b$ and $c \leqq y \leqq d$, where $c \leqq y(x) \leqq d$ for all x in $[a, b]$.) Then there exists a constant C such that the following is true: If the approximations $y_1, y_2, y_3, \ldots, y_k$ to the actual values $y(x_1), y(x_2), y(x_3), \ldots, y(x_k)$ at points of $[a, b]$ are computed using Euler's method with step size $h > 0$, then

$$|y_n - y(x_n)| \leqq Ch \tag{1.10}$$

for each $n = 1,2,3,\ldots,k$.

Remark: The error

$$y_{\text{actual}} - y_{\text{approx}} = y(x_n) - y_n$$

in (1.10) denotes the [cumulative] error in Euler's method after n steps in the approximation, exclusive of roundoff error (as though we were using a perfect machine that made no roundoff errors). The theorem can be summarized by saying that the error in Euler's method is of order h; that is, the error is bounded by a [predetermined] constant C multiplied by the step size h. It follows, for instance, that (on a given closed interval) halving the step size cuts the maximum error in half; similarly, with step size $h/10$ we get 10 times the accuracy (that is, $1/10$ the maximum error) as with step size h. Consequently, we can-in principle-get any degree of accuracy we want by choosing h sufficiently small.

We will omit the proof of this theorem, but one can be found in [5]. The constant C deserves some comment. Because C tends to increase as the maximum value of $|y''(x)|$ on $[a, b]$ increases, it follows that C must depend in a fairly complicated way on y, and actual computation of a value of C such that the inequality in (1.10) holds is usually impractical. In practice, the following type of procedure is commonly employed.

1. Apply Euler's method to the initial value problem in (1.9) with a reasonable value of h.
2. Repeat with $h/2, h/4$, and so forth, at each stage halving the step size for the next application of Euler's method.
3. Continue until the results obtained at one stage agree-to an appropriate num-ber of significant digits-with those obtained at the previous stage. Then the approximate values obtained at this stage are considered likely to be accurate to the indicated number of significant digits.

Example (1.6):

Carry out this procedure with the initial value problem

$$\frac{dy}{dx} = -\frac{2xy}{1+x^2}, \quad y(0) = 1 \qquad (1.11)$$

to approximate accurately the value $y(1)$ of the solution at $x = 1$.

Solution:

Using an Euler method in program [1]we begin with a step size $h = 0.04$ requiring $n = 25$ steps'to reach $x = 1$. The table 1.7 shows the approximate values of $y(1)$ obtained with successively smaller values of h. The data suggest that the true value of $y(1)$ is exactly 0.5. Indeed, the exact solution of the initial value problem in (1.11) is $y(x) = 1/(1+x^2)$, so the true value of $y(1)$ is exactly $\frac{1}{2}$.

h	Approximate $y(1)$	Actual $y(1)$	$\|Error\|/h$
0.04	0.50451	0.50000	0.11
0.02	0.50220	0.50000	0.11
0.01	0.50109	0.50000	0.11
0.005	0.50054	0.50000	0.11
0.0025	0.50027	0.50000	0.11
0.00125	0.50013	0.50000	0.10
0.000625	0.50007	0.50000	0.11
0.0003125	0.50003	0.50000	0.10

Table 1.7 of values in Example (5)

The final column of the table in 1.7 displays the ratio of the magnitude of the error to h; that is, $\left|y_{\text{actual}} - y_{\text{approx}}\right|/h$. Observe how the data in this column substantiate Theorem (1)-in this computation, the error bound in (1.10) appears to hold with a value of C slightly larger than 0.1.

1.2 An Improvement in Euler's Method: [1]

As Fig 1.11 shows, Euler's method is rather unsymmetrical. It uses the predicted slope $k = f(x_n, y_n)$ of the graph of the solution at the left-hand endpoint of the interval $[x_n, x_n + h]$ as if it were the actual slope of the solution over that entire interval. We now turn our attention to a way in which increased accuracy can easily be obtained; it is known as the improved Euler method.

Given the initial value problem

$$\frac{dy}{dx} = f(x, y), \qquad y(x_0) = y_0, \tag{1.12}$$

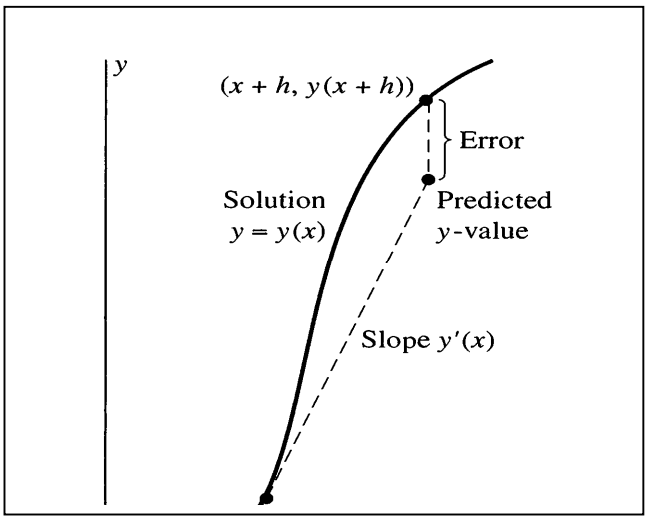

Figure 1.11 True and predicted values in Euler's method.

suppose that after carrying out n steps with step size h we have computed the approximation y_n to the actual value $y(x_n)$ of the solution at $x_n = x_0 + nh$. We can use the Euler method to obtain a first estimate which we now call u_{n+1} rather than y_{n+1}-of the value of the solution at $x_{n+1} = x_n + h$. Thus

$$u_{n+1} = y_n + h \cdot f(x_n, y_n) = y_n + h \cdot k_1$$

Now that $u_{n+1} \approx y(x_{n+1})$ has been computed, we can take

$$k_2 = f(x_{n+1}, u_{n+1})$$

as a second estimate of the slope of the solution curve $y = y(x)$ at $x = x_{n+1}$.

Of course, the approximate slope $k_1 = f(x_n, y_n)$ at $x = x_n$ has already been calculated. Why not average these two slopes to obtain a more accurate estimate of the average slope of the solution curve over the entire subinterval $[x_n, x_{n+1}]$? This idea is the essence of the improved Euler method. Figure 1.12 shows the geometry behind this method.

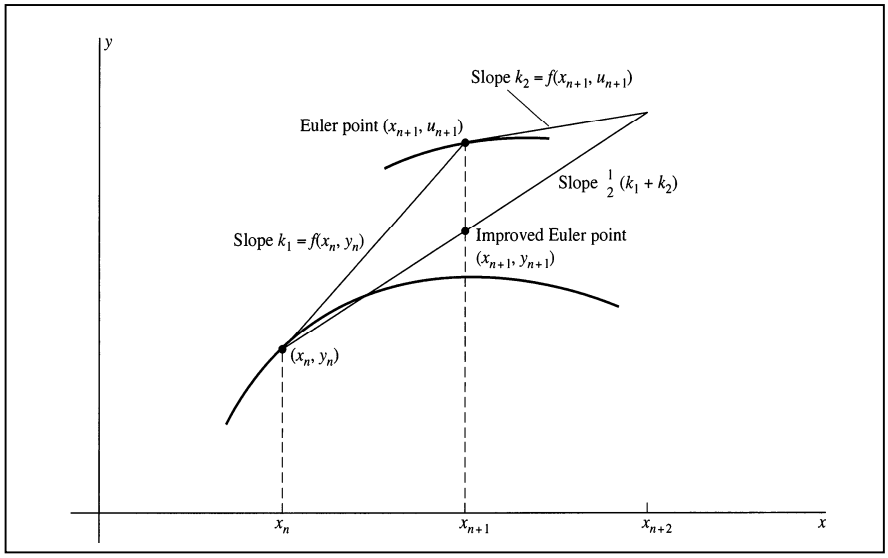

Figure 1.12 The improved Euler method: Average the slopes of the tangent lines at (x_n, y_n) and (x_{n+1}, u_{n+1}).

Algorithm: The Improved Euler Method

Given the initial value problem

$$\frac{dy}{dx} = f(x, y), \qquad y(x_0) = y_0,$$

the improved Euler method with step size h consists in applying the iterative formulas.

$$\begin{aligned} k_1 &= f(x_n, y_n), \\ u_{n+1} &= y_n + h \cdot k_1, \\ k_2 &= f(x_{n+1}, u_{n+1}), \\ y_{n+1} &= y_n + h \cdot \tfrac{1}{2}(k_1 + k_2) \end{aligned} \tag{1.13}$$

to compute successive approximations $y_1, y_2, y_3, \ldots$ to the [true] values $y(x_1), y(x_2), y(x_3)$, of the [exact] solution $y = y(x)$ at the points $x_1, x_2, x_3, \ldots$ respectively.

Remark:

The final formula in (13) takes the "Euler form"

$$y_{n+1} = y_n + h \cdot k$$

if we write

$$k = \frac{k_1 + k_2}{2}$$

for the approximate average slope on the interval $[x_n, x_{n+1}]$.

The improved Euler method is one of a class of numerical techniques known as predictor-corrector methods. First a predictor u_{n+1} of the next y-value is computed; then it is used to correct itself. Thus the improved Euler method with step size h consists of using the predictor

$$u_{n+1} = y_n + h \cdot f(x_n, y_n) \tag{1.14}$$

and the corrector

$$y_{n+1} = y_n + h \cdot \tfrac{1}{2}[f(x_n, y_n) + f(x_{n+1}, u_{n+1})] \tag{1.15}$$

iteratively to compute successive approximations y_1, y_2, y_3 ... to the values $y(x_1), y(x_2), y(y_3), \ldots$ of the actual solution of the initial value problem in (4).

Example (1.7):

Figure 1.8 shows results of applying Euler's method to the initial value problem

$$\frac{dy}{dx} = x + y, \qquad y(0) = 1 \tag{1.16}$$

with exact solution $y(x) = 2e^x - x - 1$. With $f(x, y) = x + y$ in Eqs. (1.14) and (1.15), the predictor-corrector formulas for the improved Euler method are

$$u_{n+1} = y_n + h \cdot (x_n + y_n),$$
$$y_{n+1} = y_n + h \cdot \frac{1}{2}[(x_n + y_n) + (x_{n+1} + u_{n+1})].$$

With step size $h = 0.1$ we calculate

$$u_1 = 1 + (0.1).(0 + 1) = 1.1,$$

$$y_1 = 1 + (0.05).[(0 + 1) + (0.1 + 1.1)] = 1.11,$$

$$u_2 = 1.11 + (0.1).(0.1 + 1.11) = 1.231,$$

$$y_2 = 1.11 + (0.05).[(0.1 + 1.11) + (0.2 + 1.231)] = 1.24205,$$

And so forth. The table 1.8 compares the results obtained using the improved Euler method with those obtained previously using the "unimproved" Euler method. When the same step size $h = 0.1$ is used, the error in the Euler approximation to $y(1)$ is 7.25%, but the error in the improved Euler approximation is only 0.24%.

x	Euler Method, $h = 0.1$ values of y	Euler Method, $h = 0.005$ values of y	Improved Euler, $h = 0.1$ values of y	Actual y
0.1	1.1000	1.1098	1.1100	1.1103
0.2	1.2200	1.2416	1.2421	1.2428
0.3	1.3620	1.3977	1.3985	1.3997
0.4	1.5282	1.5807	1.5818	1.5836
0.5	1.7210	1.7933	1.7949	1.7974
0.6	1.9431	2.0388	2.0409	2.0442
0.7	2.1974	2.3205	2.3231	2.3275
0.8	2.4872	2.6422	2.6456	2.6511
0.9	2.8159	3.0082	3.0124	3.0192
1.0	3.1875	3.4230	3.4282	3.4266

Table 1.8 Euler and improved Euler approximations to the solution of $dy/dx = x + y, y(0) = 1$.

Indeed, the improved Euler method with $h = 0.1$ is more accurate (in this example) than the original Euler method with $h = 0.005$. The latter requires 200 evaluations of the function $f(x, y)$, but the former requires only 20 such evaluations, so in this case the improved Euler method yields greater accuracy with only about one-tenth the work.

Table1.9 shows the results obtained when the improved Euler method is applied to the initial value problem in (1.16) using step size $h = 0.005$. Accuracy of five significant figures is apparent in the table. This suggests that, in contrast with the original Euler method, the improved Euler method is sufficiently accurate for certain practical applications-such as plotting solution curves.

x	Improved Euler Approximate y	Actual y
0.0	1.0000	1.00000
0.1	1.11034	1.11034
0.2	1.24280	1.24281
0.3	1.39971	1.39972
0.4	1.58364	1.58365
0.5	1.79744	1.79744
0.6	2.04423	2.04424
0.7	2.32749	2.32751
0.8	2.65107	2.65108
0.9	3.01919	3.01921
1.0	3.43654	3.43656

Table 1.9 Improved Euler approximation to the solution of Eg. (1.16) with step size $h = 0.005$.

An improved Euler program (similar to the ones listed in the project material for this section) was used to compute approximations to the exact value $y(1) = 0.5$ of the solution $y(x) = 1/(1 + x^2)$ of the initial value problem

$$\frac{dy}{dx} = -\frac{2xy}{1+x^2}, \qquad y(0) = 1 \qquad (1.17)$$

of Example (1). The results obtained by successively halving the step size appear in the table 1.10. Note that the final column of this table impressively corroborates the form of the error bound $\left|y_{(x_n)} - y\right| \leq ch^2$ and that each halving of the step size reduces the error by a factor of almost exactly4, as should happen if the error is proportional to h^2.

In the following two examples we exhibit graphical results obtained by employing this strategy of successively halving the step size, and thus doubling the number of subintervals of a fixed interval on which we are approximating a solution.

Example (1.8):

In Example (1.3) of Section (1.1) we applied Euler's method to the logistic initial value problem

$$\frac{dy}{dx} = \frac{1}{3}y(8 - y), \qquad y(0) = 1.$$

h	Improved Euler Approximation to $y(1)$	Error	$\lvert Error\rvert/h^2$
0.04	0.500195903	−0.000195903	0.12
0.02	0.500049494	−0.000049494	0.12
0.01	0.500012437	−0.000012437	0.12
0.005	0.500003117	−0.000003117	0.12
0.0025	0.500000780	−0.000000780	0.12
0.00125	0.500000195	−0.000000195	0.12
0.000625	0.500000049	−0.000000049	0.12
0.0003125	0.000000012	−0.000000012	0.12

Table 1.10 Improved Euler approximation to y(l) for $dy/dx = -2xy/(1 + x^2), y(0) = 1$

Figure 1.6 shows an obvious difference between the exact solution $y(x) = 8/(1 + 7e^{-8x/3})$ and the Euler approximation on $0 \leqq x \leqq 5$ using $n = 20$ subintervals. Figure 1.13 shows approximate solution curves plotted using the improved Euler's method.

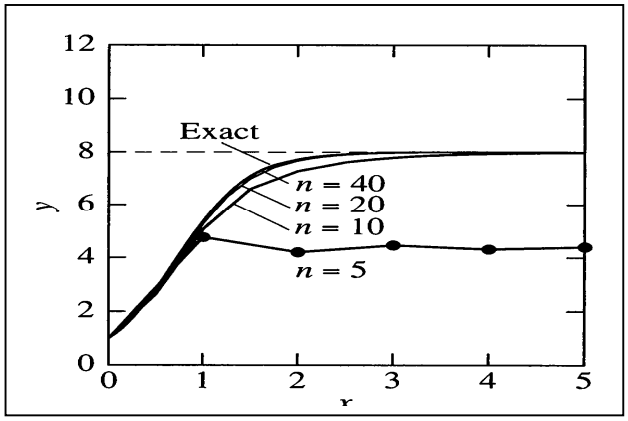

Figure 1.13 Approximating a logistic solution using the improved Euler method with $n = 5, n = 10, n = 20$, and $n = 40$ subintervals.

The approximation with five subintervals is still bad-perhaps worse! It appears to level off considerably short of the actual limiting population $M = 8$. We should carry out at least the first two improved Euler steps manually to see how it happens that, after increasing appropriately during the first step, the approximate solution decreases in the second step rather than continuing to increase (as it should).

In contrast, the approximate solution curve with $n = 20$ subintervals tracks the exact solution curve rather closely, and with $n = 40$ subintervals the exact and approximate solution curves are indistinguishable in Fig. 1.13 The table 1.11 indicates that the improved Euler approximation with $n = 200$ subintervals is accurate rounded to three decimal places (that is, four significant digits) on the interval $0 \leqq x \leqq 5$. Because discrepancies in the fourth significant digit are not visually apparent at the resolution of an ordinary computer screen, the improved Euler method (using several hundred subintervals) is considered adequate for many graphical purposes.

Table 1.11 Using the improved Euler method to approximate the actual solution of the initial value problem in Example (1.3).

Example (1.9):

In Example (4) of Section (1.1) we applied Euler's method to the initial value problem

$$\frac{dy}{dx} = y\cos x, \qquad y(0) = 1.$$

Figure 1.10 shows obvious visual differences between the periodic exact solution $y(x) = e^{\sin x}$ and the Euler approximations on $0 \leqq x \leqq 6\pi$ with as many as $n = 400$ subintervals.

Figure 1.14 shows the exact solution curve and approximate solution curves plotted using the improved Euler method with $n = 50, n = 100$, and $n = 200$ subintervals. The approximation obtained with $n = 200$ is indistinguishable from the exact solution curve, and the approximation with $n = 100$ is only barely distinguishable from it.

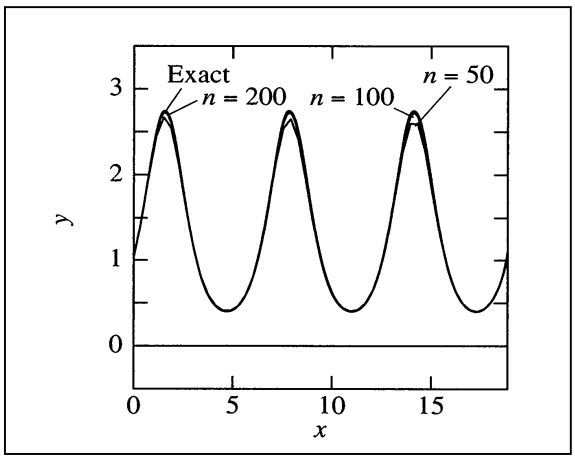

Figure1.14Approximating the exact solution $y = e^{\sin x}$ using the improved Euler method with $n = 50,100$, and 200 subintervals.

Although Figs1.13 and 1.14 indicate that the improved Euler method can provide accuracy that suffices for many graphical purposes, it does not provide the higher-precision numerical accuracy that sometimes is needed for more careful investigations. For instance, consider again the initial value problem

$$\frac{dy}{dx} = -\frac{2xy}{1+x^2}, \qquad y(0) = 1$$

of Example (1.6). The final column of the table 1.10 suggests that, if the improved Euler method is used on the interval $0 \leqq x \leqq 1$ with n subintervals and step size $h = 1/n$, then the resulting error E in the final approximation $y_n \approx y(1)$ is given by

$$E = |y(1) - y_n| \approx (0.12)h^2 = \frac{0.12}{n^2}.$$

If so, then 12-place accuracy (for instance) in the value $y(1)$ would require that $(0.12)n^{-2} < 5 \times 10^{-13}$, which means that $n \geqq 489{,}898$. Thus, roughly half a million steps of length $h \approx 0.000002$ would be required. Aside from the possible impracticality of this many steps (using available computational resources), the roundoff error resulting from so many successive steps might well overwhelm the cumulative error predicted by theory (which assumes exact computations in each separate step). Consequently, still more accurate methods than the improved Euler method are needed for such high-precision computations. Such a method is presented in [1]

Application: Improved Euler Implementation

Program [2] implementing the improved Euler method to approximate the solution of the initial value problem

$$\frac{dy}{dx} = x + y, \qquad y(0) = 1$$

considered in Example (1.7) of this section. See program[2].

To apply the improved Euler method to a differential equation $dy/dx = f(x, y)$, one need only change the initial line of the program, in which the function f is defined. To increase the number of steps (and thereby decrease the step size) one need only change the value of N specified in the second line of the program.

Program[2] exhibits one MATLAB implementation of the improved Euler method. The res2 function takes as input the initial value x, the initial value y, the final value x1 of x, and the desired number n of subintervals. As output it produces the resulting column vectors x and Y of x- and y-values. For instance, the MATLAB command

$$[\mathrm{X}, \mathrm{Y}] = \mathrm{res2}\ (0, 1, 1, 10)$$

Then generates the first and fourth columns of data shown in Table 1.8.

Famous Numbers Revisited:

The following problems describe the numbers $e \approx 2.71\ 8281\ 8$, $\mathrm{In}2 \approx 0.6931472$, and $\pi \approx 3.1415927$ as specific values of certain initial value problems. In each case, apply the improved Euler method with $n = 10,20,40,\ldots$ subintervals (doubling n each time). How many subintervals are needed to obtain-twice in succession the correct value of the target number rounded to five decimal places?

1. The number $e = y(1)$, where $y(x)$ is the solution of the initial value problem $dy/dx = y, y(0) = 1$.
2. The number $\mathrm{In}2 = y(2)$, where $y(x)$ is the solution of the initial value problem $dy/dx = 1/x, y(1) = 0$.

3. The number $\pi = y(1)$, where $y(x)$ is the solution of the initial value problem $dy/dx = 4/(1 + x^2), y(0) = 0$.

Logistic Population Investigation:

We the improved Euler program to the initial value problem $dy/dx = \frac{1}{3}y(8 - y), y(0) = 1$ of Example (1.3).In particular,We verify that the approximate solution with step size $h = 1$ levels off at $y \approx 4.3542$ rather than at the limiting value $y = 8$ of the exact solution. Perhaps a table of values for $0 \leqq x \leqq 100$ will make this apparent.

x	Actual y	Improved Euler with $n = 200$
0	1.0000	1.0000
1	5.3822	5.3809
2	7.7385	7.7379
3	7.9813	7.9812
4	7.9987	7.9987
5	7.9999	7.9999

Table 1.11 Using the improved Euler method to approximate

The actual solution of the initial value problem in example (1.3)

1.3 The Runge-Kutta Method:

We now discuss a method for approximating the solution $y = y(x)$ of the initial value problem

$$\frac{dy}{dx} = f(x, y), \qquad y(x_0) = y_0 \tag{1.16}$$

that is considerably more accurate than the improved Euler method and is more widely used in practice than any of the numerical methods discussed in Sections (1.1) and (1.2). It is called the Runge-Kutta method, after the German mathematicians who developed it, Carl Runge (1856-1927) and Wilhelm Kutta (1867-1944).

With the usual notation, suppose that we have computed the approximations $y_1, y_2, y_3, \dots, y_n$ to the actual values $y(x_1), y(x_2), y(x_3), \dots, y(x_n)$ and now want to compute $y_{n+1} \approx y(x_{n+1})$. Then

$$y(x_{n+1}) - y(x_n) = \int_{x_n}^{x_{n+1}} y'(x)dx = \int_{x_n}^{x_n+h} y'(x)dx \qquad (1.17)$$

by the fundamental theorem of calculus. Next, Simpson's rule for numerical integration yields

$$y(x_{n+1}) - y(x_n) \approx \frac{h}{6}\left[y'(x_n) + 4y'\left(x_n + \frac{h}{2}\right) + y'(x_{n+1})\right] \qquad (1.18)$$

Hence we want to define y_{n+1} so that

$$y_{n+1} \approx y_n + \frac{h}{6}\left[y'(x_n) + 2y'\left(x_n + \frac{h}{2}\right) + 2y'\left(x_n + \frac{h}{2}\right) + y'(x_{n+1})\right]; \quad (1.19)$$

we have split $4y'(x_n + \frac{1}{2}h)$ into a sum of two terms because we intend to approximate the slope $y'(x_n + \frac{1}{2}h)$ at the midpoint $x_n + \frac{1}{2}h$ of the interval $[x_n, x_{n+1}]$ in two different ways.

On the right-hand side in (1.19), we replace the [true] slope values $y'(x_n)$, $y'\left(x_n + \frac{1}{2}h\right)$, $y'(x_n + \frac{1}{2}h)$, and $y'(x_{n+1})$, respectively, with the following estimates.

$$k_1 = f(x_n, y_n). \qquad (1.20a)$$

- This is the Euler method slope at x_n.

$$k_2 = f\left(x_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}h\right). \qquad (1.20b)$$

- This is an estimate of the slope at the midpoint of the interval $[x_n, x_{n+1}]$ using the Euler method to predict the ordinate there**:**

$$k_3 = f\left(x_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}hk_2\right). \qquad (1.20c)$$

- This is an improved Euler value for the slope at the midpoint**.**

$$k_4 = f(x_{n+1} + y_n + hk_3) \qquad (1.20d)$$

- This is the Euler method slope at x_{n+1}, using the improved slope k_3 at the midpoint to step to x_{n+1}.

When these substitutions are made in (1.19), the result is the iterative formula

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \qquad (1.21)$$

The use of this formula to compute the approximations $y_1, y_2, y_3, \ldots$ successively constitutes the Runge-Kutta method. Note that Eq. (1.21) takes the "Euler form"

$$y_{n+1} = y_n + h \cdot k$$

if we write

$$k = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \qquad (1.22)$$

For the approximate average slope on the interval $[x_n, x_{n+1}]$.

The Runge-Kutta method is a fourth-order method it can be proved that the cumulative error on a bounded interval $[a, b]$ with $a = x_0$ is of order h^4. Thus the iteration in (1.21) is sometimes called the fourth-order Runge-Kutta method because it is possible to develop Runge-Kutta methods of other orders.)That is,

$$|y(x_n) - y_n| \leqq Ch^4, \qquad (1.23)$$

where the constant C depends on the function $f(x, y)$ and the interval $[a, b]$, but does not depend on the step size h. The following example illustrates this high accuracy in comparison with the lower-order accuracy of our previous numerical methods.

Example (1.10):

We first apply the Runge-Kutta method to the illustrative initial value problem

$$\frac{dy}{dx} = x + y, \qquad y(0) = 1, \qquad (1.24)$$

that we considered in Fig. (1.8) of Section (1.1) and again in Example (2) of Section (1.2). The exact solution of this problem is $y(x) = 2e^x - x - 1$. To make a point we use $h = 0.5$, a larger step size than in any previous example, so only two steps are required to go from $x = 0$ to $x = 1$.

In the first step we use the formulas in (1.20) and (1.21) to calculate and then

$$k_1 = 0 + 1 = 1,$$
$$k_2 = (0 + 0.25) + \big(1 + (0.25).(1)\big) = 1.5,$$
$$k_3 = (0 + 0.25) + (1 + (0.25).(1.5)) = 1.625,$$
$$k_4 = (0.5) + \big(1 + (0.5).(1\ .625)\big) = 2.3125,$$

and then

$$y_1 = 1 + \frac{0.5}{6}[1 + 2 \cdot (1.5) + 2 \cdot (1.625) + 2.3125] \approx 1.7969 \cdot$$

Similarly, the second step yields $y_2 \approx 3.4347$.

Table 1.12 presents these results together with the results (Table 1.8) of applying the improved Euler method with step size $h = 0.1$. We see that even with the larger step size, the Runge-Kutta method gives (for this problem) four to five times the accuracy (in terms of relative percentage errors) of the improved Euler method.

Improved Euler			Runge-Kutta		
x	y with $h = 0.1$	Percent Error	y with $h = 0.5$	Percent Error	Actual y
0.0	1.0000	0.00%	1.0000	0.00%	1.0000
0.5	1.7949	0.14%	1.7969	0.03%	1.7974
1.0	3.4282	0.24%	3.4347	0.05%	3.4366

Table 1.12 Runge-Kutta and improved Euler results for the initial value problem $dy/dx = x + y, y(0) = 1.$

It is customary to measure the computational labor involved in solving $dy/dx = f(x, y)$ numerically by counting the number of evaluations of the function $f(x, y)$ that are required. In Example **(1.10)**, the Runge-Kutta method required eight evaluations of $f(x, y) = x + y$ (four at each step), whereas the improved

Euler method required 20 such evaluations (two for each of 10 steps). Thus the Runge-Kutta method gave over four times the accuracy with only 40% of the labor.

Computer programs implementing the Runge-Kutta method are listed in the project material for this section. Table 1.13 shows the results obtained by applying the improved Euler and Runge-Kutta methods to the problem $dy/dx = x + y, y(0) = 1$ with the same step size $h = 0.1$. The relative error in the improved Euler value at $x = 1$ is about 0.24%, but for the Runge-Kutta value it is 0.00012%. In this comparison the Runge-Kutta method is about 2000 times as accurate, but requires only twice as many function evaluations, as the improved Euler method.

x	Improved Euler y	Runge-Kutta y	Actual y
0.1	1.1000	1.110342	1.110342
0.2	1.2421	1.242805	1.242806
0.3	1.3985	1.399717	1.399718
0.4	1.5818	1.583648	1.583649
0.5	1.7949	1.797441	1.797443
0.6	2.0409	2.044236	2.044238
0.7	2.3231	2.327503	2.327505
0.8	2.6456	2.651079	2.651082
0.9	3.0124	3.019203	3.019206
1.0	3.4282	3.436559	3.436564

Table 1.13 Runge-Kutta and improved Euler results for the initial value problem $dy/dx = x + y, y(0) = 1$, with the same step size $h = 0.1$

The error bound

$$|y(x_n) - y_n| \leqq Ch^4 \tag{1.23}$$

for the Runge-Kutta method results in a rapid decrease in the magnitude of errors when the step size h is reduced (except for the possibility that very small step sizes may result in unacceptable roundoff errors). It follows from the inequality in (1.23) that (on a fixed bounded interval) halving the step size decreases the absolute error by a factor of $\left(\frac{1}{2}\right)^4 = \frac{1}{16}$. Consequently, the common practice of

successively halving the step size until the computed results "stabilize" is particularly effective with the Runge-Kutta method.

Example (1.11):

In Example (5) of Section (1.1) we saw that Euler's method is not adequate to approximate the solution $y(x)$ of the initial value problem

$$\frac{dy}{dx} = x^2 + y^2, \quad y(0) = 1 \tag{1.25}$$

as x approaches the infinite discontinuity near $x = 0.969811$ (see Fig 1.16). Now we apply the Runge-Kutta method to this initial value problem.

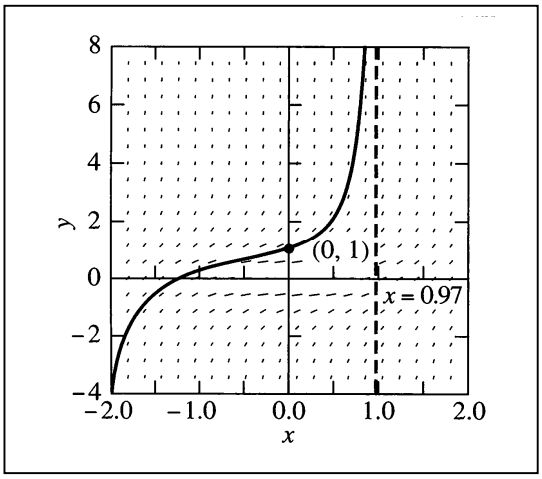

Figure 1.16 Solutions of $dy/dx = x^2 + y^2, y(0) = 1$.

Table 1.14 shows Runge-Kutta results on the interval $[0.0, 0.9]$, computed with step sizes $h = 0.1, h = 0.05$, and $h = 0.025$. There is still some difficulty near $x = 0.9$, but it seems safe to conclude from these data that $y(0.5) \approx 2.0670$.

x	y with $h = 0.1$	y with $h = 0.05$	y with $h = 0.025$
0.1	1.1115	1.1115	1.1115
0.3	1.4397	1.4397	1.4397
0.5	2.0670	2.0670	2.0670
0.7	3.6522	3.6529	3.6529
0.9	14.0218	14.2712	14.3021

Table1.14 Approximating the solution of the initial value problem in Eq. (1.25).

Example (1.12):

We therefore begin and apply the Runge-Kutta method to the initial value problem

$$\frac{dy}{dx} = x^2 + y^2, \qquad y(0.5) = 2.0670. \tag{1.26}$$

Table1.15 shows results on the interval [0.5, 0.9], obtained with step sizes $h = 0.01, h = 0.005$, and $h = 0.0025$. We now conclude that $y(0.9) \approx 14.3049$.

x	y with $h = 0.1$	y with $h = 0.005$	y with $h = 0.0025$
0.5	2.0670	2.0670	2.0670
0.6	2.6440	2.6440	2.6440
0.7	3.6529	3.6529	3.6529
0.8	5.8486	5.8486	5.8486
0.9	14.3048	14.3049	14.3049

Table1.15 Approximating the solution of the initial value problem in Eq. (1.26).

Finally, Table1.16 shows results on the interval $[0.90, 0.95]$ for the initial value problem

$$\frac{dy}{dx} = x^2 + y^2, \qquad y(0.9) = 14.3049, \tag{1.27}$$

Obtained using step sizes $h = 0.002, h = 0.001$, and $h = 0.0005$. Our final approximate result is $y(0.95) \approx 50.4723$. The actual value of the solution at $x = 0.95$ is $y(0.95) \approx 50.471\,867$. Our slight overestimate results mainly from the fact that the four-place initial value in (1.27) is (in effect) the result of rounding up the actual value $y(0.9) \approx 14.304864$; such errors are magnified considerably as we approach the vertical asymptote.

x	y with $h = 0.1$	y with $h = 0.05$	y with $h = 0.025$
0.90	14.3049	14.3049	14.3049
0.91	16.7024	16.7024	16.7024
0.92	20.0617	20.0617	20.0617
0.93	25.1073	25.1073	25.1073
0.94	33.5363	33.5363	33.5363
0.95	50.4722	50.4723	50.4723

Table1.16 Approximating the solution of the initial value problem in Eq. (1.27).

Example (1.13):

Consider the seemingly innocuous initial value problem

$$\frac{dy}{dx} = 5y - 6e^{-x}, \qquad y(0) = 1 \tag{1.32}$$

whose exact solution is $y(x) = e^{-x}$. The table Table 1.18 shows the results obtained by applying the Runge-Kutta method on the interval $[0, 4]$ with step sizes $h = 0.2, h = 0.1$, and $h = 0.05$. Obviously these attempts are spectacularly un-successful. Although $y(x) = e^{-x} \to 0$ as $x \to +\infty$, it appears that our numerical approximations are headed toward $-\infty$ rather than zero.

The explanation lies in the fact that the general solution of the equation (1.32) is

$$y(x) = e^{-x} + Ce^{5x} \tag{1.33}$$

The particular solution of (1.32) satisfying the initial condition $y(0) = 1$ is obtained with $C = 0$. But any departure, however small, from the exact solution $y(x) = e^{-x}$-even if due only to roundoff error-introduces [in effect] a nonzero value of C in Eq.(1.33). And as indicated in Fig 1.18, all solution curves of the form in (1.33) with $C \neq 0$ diverge rapidly away from the one with $C = 0$, even if their initial values are close to 1 .

x	Runge-Kutta y with $h = 0.2$	Runge-Kutta y with $h = 0.1$	Runge-Kutta y with $h = 0.1$	Actual y
0.4	0.66880	0.67020	0.67031	0.67032
0.8	0.43713	0.44833	0.44926	0.44933
1.2	0.21099	0.29376	0.30067	0.30199
1.6	−0.46019	0. 14697	0. 19802	0.20190
2.5	−4.72142	−0.27026	0. 10668	0. 13534
2.6	−35.53415	−2.90419	−0. 12102	0.09072
2.8	−261 .25023	−22.05352	−1 .50367	0.06081
3.2	−1,916.69395	−163.25077	−11.51868	0.04076
3.6	−14059.35494	−1205.71249	−85.38156	0.02732
4.0	−103, 126.5270	−8903.12866	−631 .03934	0.01832

Table 1.18 Runge-Kutta attempts to solve numerically the initial value problem in Eq. (1.32).

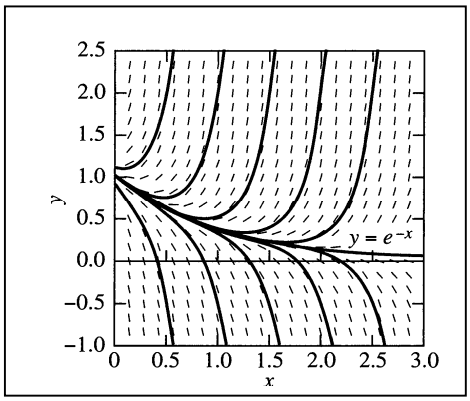

Figure1.18 Direction field and solution curves for $dy/dx = 5y - 6e^{-x}$.

Difficulties of the sort illustrated by Example (**1.13**) sometimes are unavoidable, but one can at least hope to recognize such a problem when it appears. Approximate values whose order of magnitude varies with changing step size are a common indicator of such instability. These difficulties are discussed in numerical analysis books and are the subject of current research in the field.

Application: Runge-Kutta Implementation

Program[3] implementing the Runge-Kutta method to approximate the solution of the initial value problem

$$\frac{dy}{dx} = x + y, \qquad y(0) = 1$$

Considered in Example (**1.10**) of this section.

Program [3] exhibits a MATLAB implementation of the Runge-Kutta method. Suppose that the function f describing the differential equation $y' = f(x, y)$ has been defined. Then the **res3** function takes as input the initial value **x**, the initial value **y**, the final value **xl** of **x**, and the desired number n of subintervals. As output it produces the resulting column vectors **X** and **Y** of x- and y-values. For instance, the MATLAB command

```
[X, Y] = res3(0, 1, 1, 10)
```

Then generates the first and third columns of data shown in the table 1.13

You should begin this project by implementing the Runge-Kutta method with your own calculator or computer system. Test your program by applying it first to the initial value problem in Example (1), then to some of the problems for this section.

Chapter two

Numerical Methods for Systems of ODEs

Chapter 2

Numerical Methods for Systems of ODEs

2.1 **Introduction**

We now discuss the numerical approximation of solutions of systems of differential equations. Our goal is to apply the methods of Sections 1.1 through 1.3 to the initial value problem

$$\mathbf{x}' = \mathbf{f}(t, \boldsymbol{x}) \quad \mathbf{x}(t_0) = \mathbf{x_0} \tag{2.1}$$

for a system of m first-order differential equations. In (1) the independent variable is the scalar t, and

$$\boldsymbol{x} = (x_1, x_2, \ldots, x_m) \text{ and } \quad \mathbf{f} = (f_1 f_2, \ldots, f_m)$$

are vector-valued functions. If the component functions of **f** and their first-order partial derivatives are all continuous in a neighborhood of the point $(t_0, \mathbf{x}_0)$, then [5] the existence and uniqueness of a solution $\mathbf{x} = \mathbf{x}(t)$ of (2.1) on some subinterval [of the t-axis] containing in [5]. With this assurance we can proceed to discuss the numerical approximation of this solution.

Beginning with step size h, we want to approximate the values of $\mathbf{x}(t)$ at the points $t_1, t_2, t_3, \ldots$, where $t_{n+1} = t_n + h$ for $n \geqq 0$. Suppose that we have already computed the approximations

$$\mathbf{x}_1, \quad \mathbf{x}_2, \quad \mathbf{x}_3, \quad \ldots, \quad \mathbf{x}_n$$

to the actual values

$$\mathbf{x}(t_1), \quad \mathbf{x}(t_2), \quad \mathbf{x}(t_3), \quad \ldots, \quad \mathbf{x}(t_n)$$

of the exact solution of the system in (2.1). We can then make the step from $\mathbf{x}_n$ to the next approximation $\mathbf{x}_{n+1} \approx \mathbf{x}(t_{n+1})$ by any one of the methods of Sections(1.1) through (1.3). Essentially all that is required is to write the iterative formula of the selected method in the vector notation of the present discussion.

2.2 **Euler Methods for Systems:**

For example, the iterative formula of Euler's method for systems is

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\mathbf{f}(t, \mathbf{x}_n). \tag{2.2}$$

To examine the case $m = 2$ of a pair of first-order differential equations, let us write

$$\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix} \quad \text{and} \quad \mathbf{f} = \begin{bmatrix} f \\ \mathrm{g} \end{bmatrix}. \tag{2.3}$$

Then the initial value problem in (2.1) is

$$\begin{aligned} x' &= f(t, x, y), \quad x(t_0) = x_0, \\ y' &= \mathrm{g}(t, x, y), \quad y(t_0) = y_0, \end{aligned} \tag{2.4}$$

and the scalar components of the vector formula in (2.2) are

$$\begin{aligned} x_{n+1} &= x_n + hf(t_n, x_n, y_n), \\ y_{n+1} &= y_n + h\mathrm{g}(t_n, x_n, y_n). \end{aligned} \tag{2.5}$$

Note that each iterative formula in (2.5) has the form of a single Euler iteration, but with y_n inserted like a parameter in the first formula (for x_{n+1}) and with x_n inserted like a parameter in the second formula (for y_{n+1}). The generalization to the system in (2.4) of each of the other methods in Sections (1. 1) through (1.3) follows a similar pattern.

The improved Euler method for systems consists at each step of calculating first the predictor

$$\mathbf{u}_{n+1} = \mathbf{x}_n + h\mathbf{f}(t_n, \mathbf{x}_n) \tag{2.6}$$

and then the corrector

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \frac{h}{2}[f(t_n, \mathbf{x}_n) + f(t_{n+1}, \mathbf{u}_{n+1})]. \tag{2.7}$$

For the case of the two-dimensional initial value problem in (2.4), the scalar components of the formulas in (2.6) and (2.7) are

$$u_{n+1} = x_n + hf(t_n, x_n, y_n),$$
$$v_{n+1} = y_n + hg(t_n, x_n, y_n). \tag{2.8}$$

and

$$x_{n+1} = x_n + \frac{h}{2}[f(t_n, x_n, y_n) + f(t_{n+1}, u_{n+1}, v_{n+1})],$$
$$y_{n+1} = y_n + \frac{h}{2}[f(t_n, x_n, y_n) + g(t_{n+1}, u_{n+1}, v_{n+1})]. \tag{2.9}$$

Example (2.1):

Consider the initial value problem

$$x' = 3x - 2y, \quad x(0) = 3;$$
$$y' = 5x - 4y, \quad y(0) = 6. \tag{2.10}$$

The exact solution of the system in (2.10) is

$$x(t) = 2e^{-2t} + e^t \quad y(t) = 5e^{-2t} + e^t \tag{2.11}$$

Here we have $f(x, y) = 3x - 2y$ and $g(x, y) = 5x - 4y$ in (2.3), so the Euler iterative formulas in (2.4) are

$$x_{n+1} = x_n + h \cdot (3x_n - 2y_n), \quad y_{n+1} = y_n + h \cdot (5x_n - 4y_n).$$

With step size $h = 0.1$ we calculate

$$x_1 = 3 + (0.1) \cdot [3 \cdot 3 - 2 \cdot 6] = 2.7,$$
$$y_1 = 6 + (0.1) \cdot [5 \cdot 3 - 4 \cdot 6] = 5.1$$

and

$$x_2 = 2.7 + (0.1) \cdot [3 \cdot (2.7) - 2 \cdot (5.1)] = 2.49,$$
$$y_2 = 5.1 + (0.1) \cdot [5 \cdot (2.7) - 4 \cdot (5.1)] = 4.41$$

The actual values at $t_2 = 0.2$ given by (2.10) are $x(0.2) \approx 2.562$ and $y(0.2) \approx 4.573$.

2.3 **The improved Euler methods for system:**

To compute the improved Euler approximations to $x(0.2)$ and $y(0.2)$ with a single step of size $h = 0.2$, we first calculate the predictors

$$u_1 = 3 + (0.2) \cdot [3 \cdot 3 - 2 \cdot 6] = 2.4,$$
$$v_2 = 6 + (0.2) \cdot [5 \cdot 3 - 4 \cdot 6] = 4.2.$$

Then the corrector formulas in (2.9) yield

$$x_1 = 3 + (0.1) \cdot ([3 \cdot 3 - 2 \cdot 6] + [3 \cdot (2.4) - 2 \cdot (4.2)]) = 2.58,$$
$$y_1 = 6 + (0.1) \cdot ([5 \cdot 3 - 4 \cdot 6] + [5 \cdot (2.4) - 4 \cdot (4.2)]) = 4.62.$$

As we would expect, a single improved Euler step gives better accuracy than two ordinary Euler steps.

2.4 **The Runge-Kutta Method and Second-Order Equations:**

The vector version of the iterative formula for the Runge-Kutta method is

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \frac{h}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4) \tag{2.12}$$

Where the vectors $\mathbf{k}_1, \mathbf{k}_2, \mathbf{k}_3$, and $\mathbf{k}_4$ are defined (by analogy with Eqs. $1.20a$)-(1.20d) of Section (1.3) as follows:

$$\begin{aligned} \mathbf{k}_1 &= \mathbf{f}(t_n, \mathbf{x}_n), \\ \mathbf{k}_2 &= \mathbf{f}\left(t_n + \tfrac{1}{2}h, \mathbf{x}_n + \tfrac{1}{2}h\mathbf{k}_1\right), \\ \mathbf{k}_3 &= \mathbf{f}\left(t_n + \tfrac{1}{2}h, \mathbf{x}_n + \tfrac{1}{2}h\mathbf{k}_2\right), \\ \mathbf{k}_4 &= \mathbf{f}(t_n + h, \mathbf{x}_n + \mathbf{k}_3). \end{aligned} \tag{2.13}$$

To describe in scalar notation the Runge-Kutta method for the two-dimensional initial value problem

$$\begin{aligned} x' &= f(t, x, y), \quad x(t_0) = x_0, \\ y' &= \mathrm{g}(t, x, , y), \quad y(t_0) = y_0, \end{aligned} \tag{2.4}$$

let us write

$$\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}, \quad \mathbf{f} = \begin{bmatrix} f \\ \mathrm{g} \end{bmatrix}, \quad \text{and} \quad \mathbf{k}_i = \begin{bmatrix} F_i \\ G_i \end{bmatrix}.$$

Then the Runge-Kutta iterative formulas for the step from (x_n, y_n) to the next approximation $(x_{n+1}, y_{n+1}) \approx (x(t_{n+1}), y(t_1))$ are

$$\begin{aligned} x_{n+1} &= x_n + \frac{h}{6}(F_1 + 2F_2 + 2F_3 + F_4), \\ y_{n+1} &= y_n + \frac{h}{6}(G_1 + 2G_2 + 2G_3 + G_4) \end{aligned} \tag{2.14}$$

where the values F_1, F_2, F_3 and F_4 of the function f are

$$\begin{aligned} F_1 &= f(t_n, x_n, y_n), \\ F_2 &= f\left(t_n + \tfrac{1}{2}h, x_n + \tfrac{1}{2}hF_1, y_n + \tfrac{1}{2}hG_1\right), \\ F_3 &= f\left(t_n + \tfrac{1}{2}h, x_n + \tfrac{1}{2}hF_2, y_n + \tfrac{1}{2}hG_2\right), \\ F_4 &= f\left(t_n + h, x_n + \tfrac{1}{2}hF_3, y_n + hG_3\right); \end{aligned} \tag{2.15}$$

G_1, G_2, G_3, and G_4 are the similarly defined values of the function g.

Perhaps the most common application of the two-dimensional Runge-Kutta method is to the numerical solution of second-order initial value problems of the form

$$\begin{aligned} x'' &= \mathrm{g}(t, x, x'), \\ x(t_0) &= x_0, x'(t_0) = y_0. \end{aligned} \tag{2.16}$$

If we introduce the auxiliary variable $y = x'$, then the problem in (2.16) translates into the two-dimensional first-order problem

$$\begin{aligned} x' &= y, & x(t_0) &= x_0, \\ y' &= \mathrm{g}(t, x, y), & y(t_0) &= y_0. \end{aligned} \tag{2.17}$$

This is a problem of the form in (2.4) with $f(t, x, y) = y$.

Example (2.2):

The exact solution of the initial value problem

$$x'' = -x; \qquad x(0) = 0,\ x'(0) = 1 \tag{2.18}$$

is $x(t) = \sin t$. The substitution $y = x'$ translates (2.17) into the two-dimensional problem

$$\begin{aligned} x' &= y, \quad & x(0) &= 0; \\ y' &= -x, \quad & y(0) &= 1, \end{aligned} \tag{2.19}$$

Which has the form in (2.4) with $f(t, x, y) = y$ and $g(t, x, y) = -x$. Table 2.1 shows the results produced for $0 \leqq t \leqq 5$ (radians) using Program RK2DIM with step size $h = 0.05$. The values shown for $x = \sin t$ and $y = \cos t$ are all accurate to five decimal places.

t	$x = \sin t$	$y = \cos t$
0.5	+0.47943	+0.87758
1.0	+0.84147	+0.54030
1.5	+0.99749	+0.07074
2	+0.90930	−0.41615
2.5	+0.59847	−0.801 14
3.0	+0. 14112	−0.98999
3.5	−0.35078	−0.93646
4.0	−0.75680	−0.65364
4.5	−0.97753	−0.21080
5.0	−0.95892	+0.28366

Table 2.1. Runge-Kutta values (with $h = 0.05$) for the problem in Eq. (2.19).

2.5 **Higher-Order Systems:**

As we saw in Section 2.4, any system of higher-order differential equations can be replaced with an equivalent system of first-order differential equations. For example consider the system

$$\begin{aligned} x'' &= F(t, x, y, x', y'), \\ y'' &= G(t, x, y, x', y') \end{aligned} \tag{2.20}$$

of second-order equations. If we substitute

$$x = x_1, \quad y = x_2, \quad x' = x_3 = x_1', \quad y' = x_4 = x_2'$$

then we get the equivalent system

$$\begin{aligned} x_1' &= x_3, \\ x_2' &= x_4, \\ x_3' &= F(t, x_1, x_2, x_3, x_4), \\ x_4' &= G(t, x_1, x_2, x_3, x_4) \end{aligned} \tag{2.21}$$

Of four first-order equations in the unknown functions $x_1(t) = x(t), x_2(t) = y(t), x_3(t)$, and $x_4(t)$. It would be a routine (if slightly tedious) matter to write a four-dimensional version of program RK2DIM for the purpose of solving such a system. But in a programming language that accommodates vectors, an n-dimensional Runge-Kutta program is scarcely more complicated than a one-dimensional program. For instance, the application material for this section lists in the MATLAB program [**5**].

Example (2.3):[1]

Suppose that a base ball starts at $x_0 = 0, y_0 = 0$ with initial velocity $v_0 = 160$ ft/s and with initial angle of inclination $\theta = 30°$. If air resistance is ignored, we find by the elementary methods of Section 1.2 that the baseball travels a [horizontal] distance of $400\sqrt{3}$ ft (approximately 693 ft) in 5 s before striking the ground. Now suppose that in addition to a downward gravitational acceleration ($g = 32$ ft$/s^2$), the baseball experiences an acceleration due to air resistance of $(0.0025)v^2$ feet per second, directed opposite to its instantaneous direction of

motion. Determine how far the baseball will travel horizontally under these conditions.

Solution:

The equations of motion of the baseball are

$$\frac{d^2x}{dt^2} = -cv\frac{dx}{dt}, \quad \frac{d^2y}{dt^2} = -cv\frac{dy}{dt} - \mathrm{g} \tag{2.22}$$

Where $v = \sqrt{(x')^2 + (y')^2}$ is the speed of the ball, and where $c = 0.0025$ and $\mathrm{g} = 32$ in fps units. We convert to a first-order system as in (2.21) and thereby obtain the system

$$\begin{aligned} x_1' &= x_3, \\ x_2' &= x_4, \\ x_3' &= -cx_3\sqrt{x_3^2 + x_4^2} \\ x_4' &= -cx_4\sqrt{x_3^2 + x_4^2} - \mathrm{g} \end{aligned} \tag{2.23}$$

of four first-order differential equations with

$$\begin{aligned} x_1(0) &= x_2(0) = 0, \\ x_3(0) &= 80\sqrt{3}, \qquad x_4(0) = 80. \end{aligned} \tag{2.24}$$

Note that $x_3(t)$ and $x_4(t)$ are simply the x- and y-components of the baseball's velocity vector, so $v = \sqrt{x_3^2 + x_4^2}$. We proceed to apply the Runge-Kutta method to investigate the motion of the batted baseball described by the initial value problem in (2.23) and (2.24), first taking $c = 0$ to ignore air resistance and then using $c = 0.0025$ to take air resistance into account.

WITHOUT AIR RESISTANCE:

Table2.4 shows the numerical results obtained when a Runge-Kutta program such as **program [5]** is applied with step size $h = 0.1$ and with $c = 0$ (no air resistance). For convenience in interpreting the results, the printed output at each selected step consists of the horizontal and vertical coordinates x and y of the baseball, its velocity v, and the angle of inclination of its velocity vector (in

degrees measured from the horizontal). These results agree with the exact solution when $c = 0$. The ball travels a horizontal distance of $400\sqrt{3} \approx 692.82$ ft in exactly 5 s, having reached a maximum height of 100 ft after 2.5 s. Note also that the ball strikes the ground at the same angle and with the same speed as its initial angle and speed.

t	x	y	v	α
0.0	0.00	0.00	160.00	+30
0.5	69.28	36.00	152.63	+25
1.0	138.56	46.00	146.64	+19
1.5	207.85	84.00	142.21	+13
2.0	277.13	96.00	139.48	+7
2.5	346.41	100.00	138.56	+0
3.0	415.69	96.00	139.48	−7
3.5	484.97	84.00	142.21	−13
4.0	554.26	64.00	146.64	−19
4.5	623.54	36.00	152.63	−25
5.0	692.82	0.00	160.00	−30

Table 2.3 The batted baseball with no air resistance ($c = 0$).

t	x	y	v	α
0.0	0.00	0.00	160.00	+30
0.5	63.25	32.74	127.18	+24
1.0	117.11	53.20	104.86	+17
1.5	164.32	63.60	89.72	+8
2.0	206.48	65.30	80.17	−3
2.5	244.61	59.22	75.22	−15
3.0	279.29	46.05	73.99	−27
3.5	310.91	26.41	75.47	−37
4.0	339.67	0.91	78.66	−46

Table 2.4The batted baseball with air resistance ($c = 0.0025$).

WITH AIR RESISTANCE:

Table 2.4 shows the results obtained with the fairly realistic value of $c = 0.0025$ for the air resistance for a batted baseball. To within a hundredth of a foot in either direction, the same results are obtained with step sizes $h = 0.05$ and

$h = 0.025$. We now see that with air resistance the ball travels a distance well under 400 ft in just over 4 s. The more refined data in Table 2.5 show that the ball travels horizontally only about 340 ft and that its maximum height is only about 66 ft. As illustrated in Table 2.5, air resistance has converted a massive home run into a routine fly ball (if hit straightaway to center field). Note also that when the ball strikes the ground, it has slightly under half its initial speed (only about 79 ft/s) and is falling at a steeper angle (about 46°). Every baseball fan has observed empirically these aspects of the trajectory of a fly ball. (see fig(2.1)).

t	x	y	v	α	
1.5	164.32	63.60	89.72	+8	
1.6	173.11	64.60	87.40	+5	
1.7	181.72	65.26	85.29	+3	
1.8	190.15	65.60	83.39	+1	
1.9	198.40	65.61	81 .68	−1	Apex
2.0	206.48	65.30	80.17	−3	
⋮	⋮	⋮	⋮	⋮	
3.8	328.50	11 .77	77.24	−42	
3.9	334.14	6.45	77.93	−44	
4.0	339.67	0.91	78.66	−46	Impact
4.1	345.10	−4.84	79.43	−47	
4.2	350.41	−10.79	80.22	−49	

Table 2.5 The batted ball's apex and its impact with the ground.

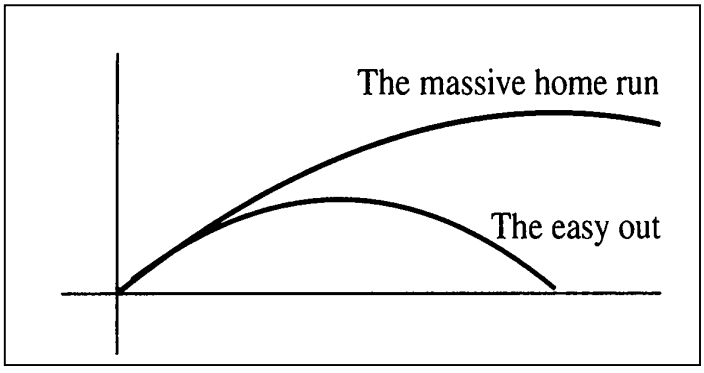

Figure2.1 An "easy out" or a home run?

Variable Step Size Methods:

The Runge-Kutta method for a large system requires an appreciable amount of computational labor, even when a computer is employed. Therefore, just as the step size h should not be so large that the resulting error in the solution is unacceptable, h ought not to be so small that too many steps are needed, hence requiring an unacceptable amount of computation. Thus the practical numerical solution of differential equations involves a tradeoff between accuracy and efficiency.

To facilitate this tradeoff, modern variable step size methods vary the step size h as the solution process proceeds. Large steps are taken in regions where the dependent variables are changing slowly; smaller steps are taken when these variables are changing rapidly, in order to prevent large errors.

An adaptable or variable step size Runge-Kutta method employs both a pre-assigned minimum error tolerance $MinTol$ and a maximum error tolerance MaxTol to attempt to ensure that the error made in the typical step from x_n to x_{n+1} is neither too large (and hence inaccurate) nor too small (and hence inefficient). A fairly simple scheme for doing this may be outlined as follows:

- Having reached x_n with a Runge-Kutta step of length $t_n - t_{n-1} = h$, let $\mathbf{x}^{(1)}$ denote the result of a further Runge-Kutta step of length h and let $\mathbf{x}^{(2)}$denote the result of two successive Runge-Kutta steps each of length $h/2$.
- On the grounds that $\mathbf{x}^{(2)}$ should be a more accurate approximation to $\mathbf{x}(t_n + h)$ than is $\mathbf{x}^{(1)}$, take

$$Err = \left|\mathbf{x}^{(1)} - \mathbf{x}^{(2)}\right|$$

As an estimate of the error in $\mathbf{x}^{(1)}$.

- If $MinTol \leqq Err \leqq MaxTol$, then let $\mathbf{x}_{n+1} = \mathbf{x}^{(1)}, t_{n+1} = t_n + h$, and proceed to the next step.
- If $Err < MinTol$, then the error is too small ! Hence let $\mathbf{x}_{n+1} = \mathbf{x}^{(1)}, t_{n+1} = t_n + h$, but double the step size to $2h$ before making the next step.
- If $Err > MaxTol$, then the error is too large. Hence reject $\mathbf{x}^{(1)}$and start afresh at x_n with the halved step size $h/2$.

The detailed implementation of such a scheme can be complicated. For a much more complete but readable discussion of adaptive Runge-Kutta methods, see Section 15.2 of [4] Several widely available scientific computing packages (such as Maple, Math ematica, and MATLAB) include sophisticated variable step size programs that will accommodate an essentially arbitrary number of simultaneous differential equations. Such a general purpose program might be used, for example, to model numerically the major components of the solar system: the sun and the nine (known) major planets. If m_i denotes the mass and $\mathbf{r}_i = (x_i, y_i, z_i)$ denotes the position vector of the ith one of these 10 bodies, then by Newton's laws the equation of motion of m_i is

$$m_i \mathrm{r}_i'' = \sum_{j \neq i} \frac{G m_i m_j}{(r_{ij})^3} (\mathbf{r}_j - \mathbf{r}_i) \qquad (2.25)$$

where $r_{ij} = |\mathbf{r}_j - \mathbf{r}_i|$ denotes the distance between m_i and m_j. For each $i = 1,2,\ldots,10$, the summation in Eq. (2.25) is over all values of $j \neq i$ from 1 to 10. The 10 vector equations in (2.25) constitute a system of 30 second-order scalar equations, and the equivalent first-order system consists of 60 differential equations in the coordinates and velocity components of the 10 major bodies in the solar system. Mathematical models that involve this many (or more) differential equations and that require sophisticated software and hardware for their numerical analysis are quite common in science, engineering, and applied technology.

Earth-Moon Satellite Orbits:

For an example of a program whose efficient solution requires adaptive step size methods, we consider an Apollo satellite in orbit about the Earth E and Moon M. Figure 2.2 shows an x_1x_2-coordinate system whose origin lies at the center of mass of the Earth and the Moon and which rotates at the rate of one revolution per "moon month" of approximately $\tau = 27.32$ days, so the Earth and Moon remain fixed in their positions on the x_1 -axis. If we take as unit distance the distance (about 384,000 kilometers, assumed constant) between the Earth and Moon centers, then their coordinates are $E(-\mu, 0)$ and $M(1-\mu, 0)$, where $\mu = m_M/(m_E + m_M)$ in terms of the Earth mass m_E and Moon mass m_M. If we take the total mass $m_E + m_M$ as the unit of mass and $\tau/(2\pi) \approx 4.35$ days as the unit

oftime, then the gravitational constant is $G = 1$ in Eq. (2.26), and the equations of motion of the satellite position $S(x_1, x_2)$ are

$$x_1'' = x_1 + 2x_2' - \frac{(1-\mu)(x_1+\mu)}{(r_E)^3} - \frac{\mu(x_1 - 1 + \mu)}{(r_M)^3},$$
$$x_2'' = x_2 - 2x_1' - \frac{(1-\mu)x_2}{(r_E)^3} - \frac{\mu x_2}{(r_M)^3}, \qquad (2.27)$$

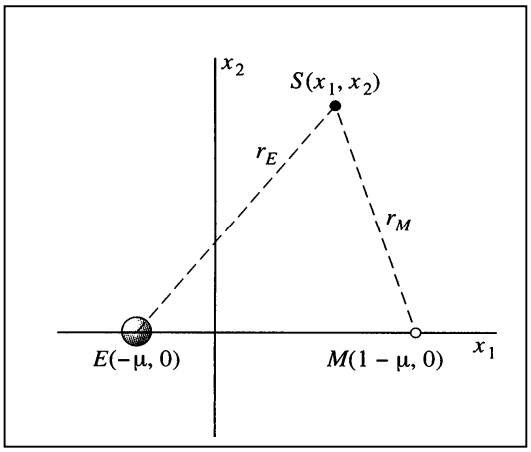

Figure 2.2 The Earth-Moon center-of-mass coordinate system.

Where r_E and r_M denote the satellite's distance to the Earth and Moon (indicated in Fig2.2 The initial two terms on the right-hand side of each equation result from the rotation of the coordinate system. In the system of units described here, the lunar mass is approximately $m_M = 0.012277471$. The second-order system in (2.27) can be converted to an equivalent first-order system (of four differential equations) by substituting

$$x_1' = x_3, x_2' = x_4, \qquad \text{so that } x_1' = x_3', \qquad x_2'' = x_4'.$$

Suppose that the satellite initially is in a clockwise circular orbit of radius about 2400 kilometers about the Moon. At its farthest point from the Earth ($x_1 = 0.994$) it is "launched" into Earth-Moon orbit with initial velocity v_0. The corresponding initial conditions are

$$x_1(0) = 0.994, \quad x_2(0) = 0, \quad x_3(0) = 0, \quad x_4(0) = -v_0$$

An adaptive step size method (**ode45**) in the MATLAB software system was used to solve numerically the system in (2.27). The orbits in Figs. 2.3and 2.4 were obtained with

$$v_0 = 2.031732629557 \quad \text{and} \quad v_0 = 2.001585106379,$$

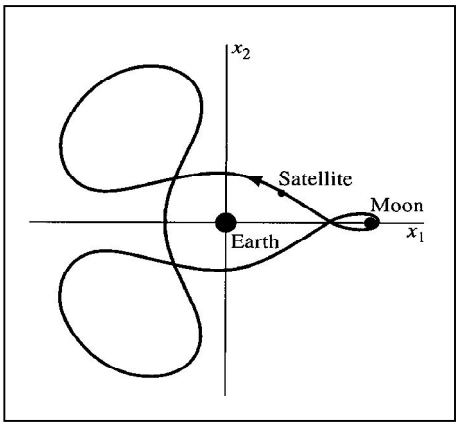

Figure 2.3 Apollo MoonEarth bus orbit with insertion velocity $v_0 = 7476$ km/h.

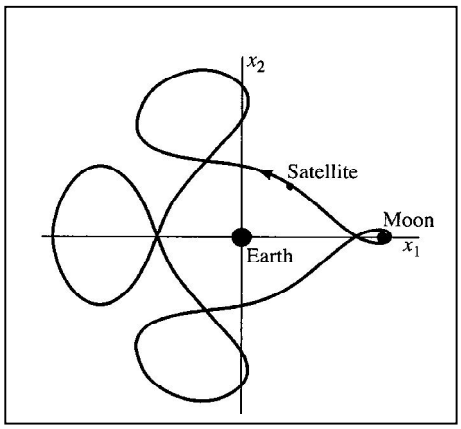

Figure2.4 Apollo MoonEarth bus orbit with insertion velocity $v_0 = 7365$ km/h.

Respectively. [In the system of units used here, the unit of velocity is approxi-mately 3680 km/h.] In each case a closed but multilooped periodic trajectory about the Earth and the Moon-a so-called bus orbit-is obtained, but a relatively small change in the initial velocity changes the number of loops! For more information, see [7].

So-called Moon-Earth "bus orbits" are periodic-that is, are closed trajectories traversed repeatedly by the satellite-only in a rotating x_1x_2-coordinate system as discussed above. The satellite of Fig2.3 traverses its closed orbit and returns to

rendezvous with the Moon about 48.4 days after its insertion into orbit. Figures 2.5 and 2.6 illustrate the motion of the same satellite-but in an ordinary nonrotating xy-coordinate system centered at the Earth, in which the Moon encircles the Earth counterclockwise in a near-circular orbit, completing one revolution in about 27.3 days. The Moon starts at point S, and after 48.4 days it has completed a bit over 1.75 revolutions about the Earth and reaches the point R at which its rendezvous with the satellite occurs. Figure 2.5 shows the positions of Moon and satellite a day and a half after the satellite's insertion into its orbit, each traveling in a generally counterclockwise direction around the Earth. Figure 2.6 shows their positions a day and a half before their rendezvous at point R, the satellite mean-while having encircled the Earth about 2.5 times in an orbit that (in the indicated xy-coordinate system) appears to resemble a slowly varying ellipse.

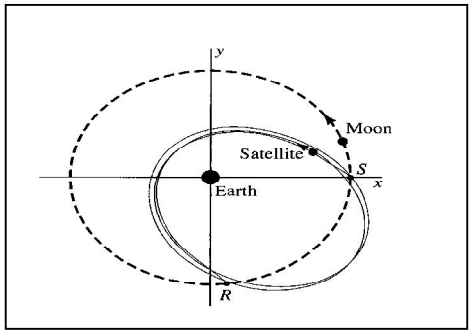

Figure2.5. The moon and satellite in a nonrotating coordinate system, 1.5 days after orbital insertion of the satellite at starting point S.

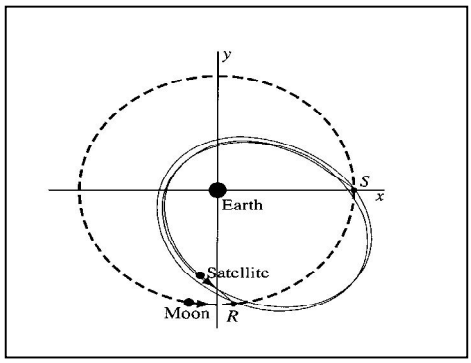

Figure2.6 The moon and satellite in a nonrotating coordinate system, 1.5 days before their rendezvous at point R.

Application: Comets and Spacecraft

Figure 2.7 lists TI-85 and BASIC versions of the two-dimensional Runge-Kutta program RK2DIM. You should note that it closely parallels the one-dimensional Runge-Kutta program listed in Fig000, with a single line there replaced (where appropriate) with two lines here to calculate a pair of x- and y-values or slopes. Note also that the notation used is essentially that of Eqs. (13) and (14) in this section. The first several lines define the functions and initial data needed for Example (1).

Figure 2.8 exhibits an n-dimensional MATLAB implementation of the Runge-Kutta method. The MATLAB function **f** defines the vector of right-hand sides of the differential equations in the system $x' = \mathbf{f}(t, x)$ to be solved. The **rkn** function then takes as input the initial t-value **t**, the column vector **x** of initial **x**-values, the final t-value **t1**, and the desired number **n** of subintervals. As output it produces the resulting column vector **T** of t-values and the matrix **x** whose rows give the corresponding x-values. For instance, with f as indicated in the figure, the MATLAB command

[T , X] = rkn (0 , [0 ; 1] , 5 , 5 0)

then generates the data shown in the table of Fig. 2.1 (which lists only every fifth value of each variable).

You can use Examples (2.1) through (2.3) in this section to test your own implementation of the Runge-Kutta method. Then investigate the comet and spacecraft problems described next. Additional application material at the Web site **www.prenhall.com/edwards** describes additional numerical ODE investigations ranging from batted baseballs to the Apollo orbits shown in Figs. 2.5 and 2.6

Your Spacecraft Landing:

Your spacecraft is traveling at constant velocity V, approaching a distant earthlike planet with mass M and radius R. When activated, your deceleration system provides a constant thrust T until impact with the surface of the planet. During the period of deceleration, your distance $x(t)$ from the center of the planet satisfies the differential equation

$$\frac{d^2x}{dt^2} = T - \frac{G\,M}{x^2} \tag{2.28}$$

where $G \approx 6.6726 \times 10^{-11} N.(m/\text{kg})^2$ as in Example (2.3). Your question is this: At what altitude above the surface should your deceleration system be activated in order to achieve a soft touchdown? For a reasonable problem, you can take

$$M = 5.97 \times 10^{24}(\text{kg}),$$
$$R = 6.38 \times 10^{6}(\text{m}).$$
$$V = p \times 10^{4}(\text{km/h}),$$
$$T = \text{g} + q(\text{m}/s^2)$$

where $\text{g} = GM/R^2$ is the surface gravitational acceleration of the planet. Choose p to be the smallest nonzero digit and q the next-to-smallest nonzero digit in your ID number. Find the "ignition altitude" accurate to the nearest meter and the resulting "descent time" accurate to the nearest tenth of a second.

Kepler's Law of Planetary (or Satellite) Motion:

Consider a satellite in elliptical orbit around a planet of mass M, and suppose that physical units are so chosen that $GM = 1$ (where G is the gravitational constant). If the planet is located at the origin in the xy-plane, then the equations of motion of the satellite are

$$\frac{d^2x}{dt^2} = -\frac{x}{(x^2+y^2)^{3/2}}, \quad \frac{d^2y}{dt^2} = -\frac{y}{(x^2+y^2)^{3/2}} \tag{2.29}$$

Let T denote the period of revolution of the satellite. Kepler's third law says that the square of T is proportional to the cube of the major semiaxis a of its elliptical orbit. In particular, if $G\,M = 1$, then

$$T^2 = 4\pi^2 a^3. \tag{2.30}$$

(For details, see Section 11.6 of [8].if the satellite's x-and y-components of velocity, $x_3 = x' = x_1'$; and $x_4 = y' = x_2'$, are introduced, then the system in (2.29) translates into a system of four first-order differentiaJ equations having the form of those in Eq. (22) of this section.

(a) Solve this 4×4 system numerically with the initial conditions

$$x\,(0) = 1, \qquad y(0) = 0, \qquad x'(0) = 0, \qquad y'(0) = 1$$

that correspond theoretically to a circular orbit of radius $a = 1$, so Eq. (2.30) gives $T = 2\pi$. Is this what you get?

(b) Now solve the system numerically with the initial conditions

$$x(0) = 1\,, \;\; y(0) = 0, \qquad x'(0) = 0, \qquad y'(0) = \tfrac{1}{2}\sqrt{6}$$

that correspond theoretically to an elliptical orbit with major semi axis $a = 2$, so Eq. (2.30) gives $T = 4\pi\sqrt{2}$. Is this what you get?

Halley's Comet:

Halley's comet last reached perihelion (its point of closest approach to the sun at the origin) on February 9, 1986. Its position and velocity components at that time were

$$\mathbf{P}_0 = (0.325514, -0.459460, 0.166229) \text{ and}$$
$$\mathbf{V}_0 = (-9.096111, -6.916686, -1.305721)$$

(respectively), with position in AU (astronomical units, in which the unit of distance is the major semiaxis of the earth's orbit) and time in years. In this system, the three-dimensional equations of motion of the comet are

$$\frac{d^2x}{dt^2} = -\frac{\mu x}{r^3}, \quad \frac{d^2y}{dt^2} = -\frac{\mu y}{r^3}, \quad \frac{d^2z}{dt^2} = -\frac{\mu z}{r^3} \qquad (2.31)$$

where

$$\mu = 4\pi^2 \quad \text{and} \quad r = \sqrt{x^2 + y^2 + z^2}$$

Solve the equations in (2.31) numerically to verify the appearance of the yz-projection of the orbit of Halley's comet shown in Fig2.9 Plot the xy- and xz-projections as well.

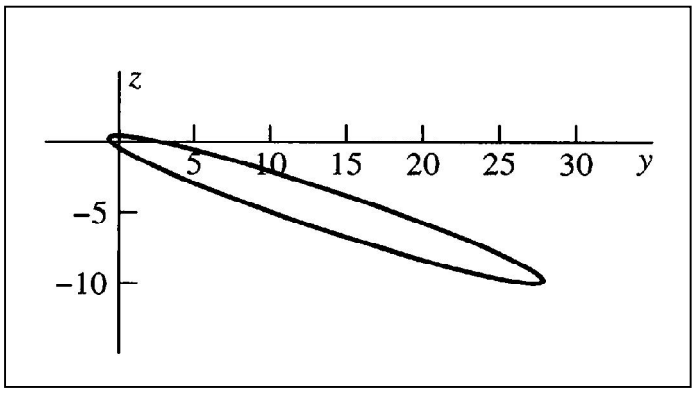

Figure2.9 yz-projection of the orbit of Halley's comet.

Figure 2.10 shows the graph of the distance ret) of Halley's comet from the sun. Inspection of this graph indicates that Halley's comet reaches a maximum distance (at aphelion) of about 35 AU in a bit less than 40 years and returns to perihelion after about three-quarters of a century. The closer look in Fig. 2.11 indicates that the period of revolution of Halley's comet is about 76 years. Use your numerical solution to refine these observations. What is your best estimate of the calendar date of the comet's next perihelion passage?

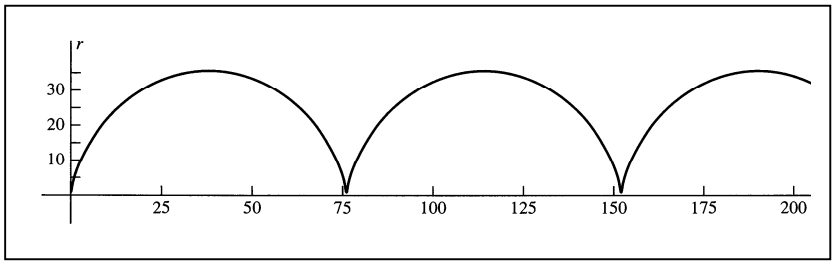

Figure2.10. 200-year plot of the distance $r(t)$ of Halley's comet from the sun. Is there a cusp near $t = 75$?

Your Own Comet:

The night before your birthday in 2007 you set up your telescope on a nearby mountaintop. It was a clear night, and you had a stroke of luck: At 12:30 A.M. you spotted a new comet. After repeating the observation on successive nights, you were able to calculate its solar system coordinates $\mathbf{P}_0 = (x_0, y_0, z_0)$ and its velocity vector $\mathbf{V}_0 = (v_{x0}, v_{y0}, v_{z0})$ on that first night. Using this information, determine the following:

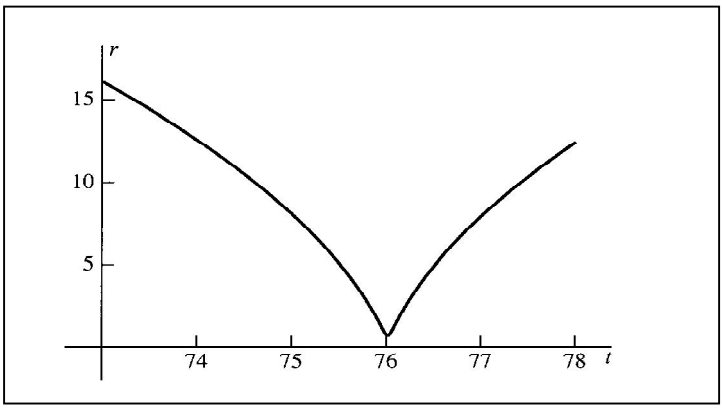

Figure2.11. A closer look at Halley's perihelion passage after about 76 years.

- the comet's perihelion (point nearest the sun) and aphelion (point farthest from the sun),
- the comet's velocity at perihelion and at aphelion,
- the comet's period of revolution around the sun, and
- the comet's next two dates of perihelion passage.

Using units of length in AU and time in earth years, the equations of motion of your comet are given in (2.31). For your personal comet, begin with random initial position and velocity vectors with the same order of magnitude as those of Halley's comet. Repeat the random selection of initial position and velocity vectors, if necessary, until you get a plausible eccentric orbit that ranges well outside the earth's orbit (as most real comets do).

Chapter three

Numerical Methods for PDEs(Parabolic Heat Equation)

Chapter 3

3.1 Introduction:

This section provides an introduction to the finite difference method (FDM) for solving partial differential equations (PDEs). In addition to specific FDM details, general concepts such as stability, boundary conditions, verification, validation and grid independence are presented which are important for anyone wishing to solve PDEs by using other numerical methods and/or commercial software packages. Material is presented in order of increasing complexity and supplementary theory is included in appendices.

Partial Differential Equations:

The following equation is an example of a PDE:

$$a(t,x,y)U_t + b(t,x,y)U_x + c(t,x,y)U_{yy} = f(t,x,y) \qquad (3.1)$$

where,

- t, x, y are the *independent* variables (often time and space)
- a, b, c and f are known functions of the independent variables,
- U is the *dependent* variable and is an unknown function of the independent variables.
- partial derivatives are denoted by subscripts: $U_t = \frac{\partial U}{\partial t}, U_x = \frac{\partial U}{\partial x}, U_{yy} = \frac{\partial^2 U}{\partial y^2}$ etc.

The order of a PDE is the order of its highest derivative. A PDE is linear if U and all its partial derivatives occur to the first power only and there are no products involving more than one of these terms. (3.1) is second order and is linear. The dimension of a PDE is the number of independent spatial variables it contains. eq(3.1) is 2D if x and y are spatial variables.

Solution to a Partial Differential Equation:

Solving a PDE means finding the unknown function U. An analytical (i.e. exact) solution of a PDE is a function that satisfies the PDE and also satisfies any boundary and/or initial conditions given with the PDE (more about these later). Most PDEs of interest do not have analytical solutions so a numerical procedure must be used to find an approximate solution. The approximation is made at discrete values of the independent variables and the approximation scheme is implemented via a computer program. The FDM replaces all partial derivatives and other terms in the PDE by approximations. After some manipulation, a finite difference scheme (FDS) is created from which the approximate solution is obtained. The FDM depends fundamentally on Taylor's beautiful theorem (circa 1712!) which is stated in the next chapter.

PDE Models:

PDEs describe many of the fundamental natural laws (e.g. conservation of mass) so describe a wide range of physical phenomena. Examples include Laplace's equation for steady state heat conduction, the advection-diffusion equation for pollutant transport, Maxwell's equations for electromagnetic waves, the Navier–Stokes equations for fluid flow and many, many more.

Classification of PDEs:

The Second order linear PDEs can be formally classified into 3 generic types: elliptic, parabolic and hyperbolic. The simplest examples are:

a) Elliptic: e.g. $U_{xx} + U_{yy} = f(x, y)$.
   This is Poisson's equation or Laplace's equation (when $f(x, y) = 0$) which may be used to model the steady state temperature distribution in a plate or incompressible potential flow. Notice there is no time derivative.
b) Parabolic: e.g. $U_t = kU_{xx}$.
   This is the 1D diffusion equation and can be used to model the time-dependent temperature distribution along a heated 1D bar.
c) Hyperbolic: e.g. $U_{tt} = c^2 U_{xx}$.
   This is the wave equation and may be used to model a vibrating guitar string or 1D supersonic flow.

d) $U_t = -cU_x$.
This first order PDE is called the advection equation. Solutions of d) also satisfy c).

e) $U_t + cU_x = kU_{xx}$.
This is the advection-diffusion equation and may be used to model transport of a pollutant in a river. The coefficients k, c in the above PDEs quantify material properties that relate to the problem being solved e.g. k could be the coefficient of thermal conductivity in the case of a heated bar, or 1D diffusion coefficient in the case of pollutant transport; c is a wave speed, usually, in fluid flow, the speed of sound.

Initial and Boundary Conditions:

PDEs require proper initial conditions (ICs) and boundary conditions (BCs) in order to define what is known as a well-posed problem. If too many conditions are specified then there will be no solution; if too few conditions are specified the solution will not be unique. If the ICs/BCs are specified in the wrong place or at the wrong time then the solution will not depend smoothly on the ICs/BCs and small errors in the ICs/BCs will bring about large changes in the solution. This is referred to as an ill-posed problem. The PDEs encountered in practice are often non-linear and multi-dimensional and cannot be reduced to the simple so-called canonical forms of a) - e). However, we need to understand the properties of the solution to these simple model PDEs before attempting to solve more complicated PDEs.

A second order elliptic PDE such as a) requires a boundary condition on U at each point on the boundary. Thus, these are called Boundary Value (BV) problems. The BC may be a value of U on the boundary or the value of its derivative (see Appendix B). Linear parabolic equations such a b) require ICs at the initial start time (usually $t = 0$) and one BC at each end-point of the spatial domain (e.g. at the ends of the heated bar). Technically linear hyperbolic equations such as d) require ICs and as many BCs as there are inward-pointing characteristics (this is an advanced topic which we will not cover) which depend on the sign of wave speed c, thus:

If $c > 0$, we need ICs: $U(0, x) = f(x)$ and BCs: $U(t, 0) = g(t)$;

If $c < 0$, we need ICs: $U(0,x) = f(x)$ but no BCs.

These are called Initial Boundary Value Problems (IBV) problems.

Domain of Dependence:

The differences between the types of PDEs can be illustrated by sketching their respective domains of dependence. So for example, in the hyperbolic case d), point $P(x_0, t_0)$ in Figure 3.1 can only be influenced by points lying within the region bounded by the two characteristics $x + ct = \text{const}$ and $x - ct = \text{const}$ and $t < t_0$. This region is called the domain of dependence. In turn, point P can influence points at later times lying within its zone of influence. In the parabolic case, shown in Figure 3.2 information travels downstream (or forward in time) only and so the domain of dependence of point $P(x_0, t_0)$ in this case is the region $t < t_0$ and the zone of influence is all points for which $t < t_0$.

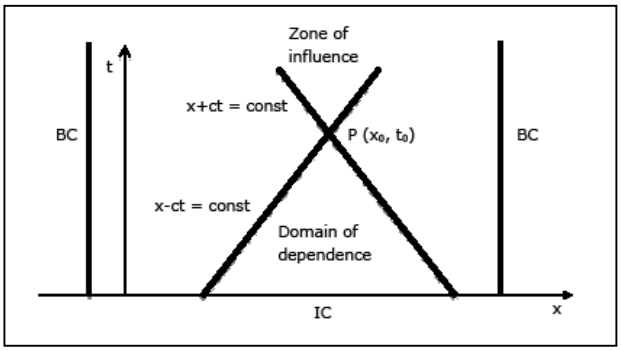

Figure 3.1 Domain of dependence: hyperbolic case.

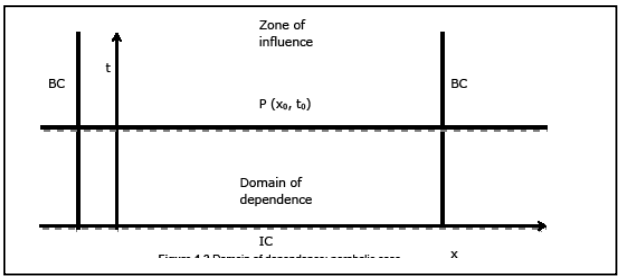

Figure 3.2 Domain of dependence: parabolic case.

In the elliptic case, corresponding to subsonic flow (Figure 3.3), information travels in all directions at infinite speed so the solution at point $P(x_0, t_0)$ influences all points within the domain and vice versa.

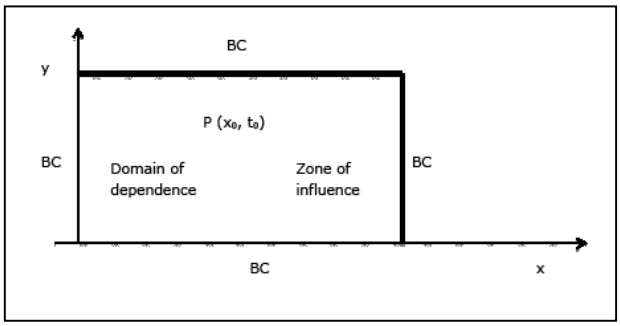

Figure 3.3 Domain of dependence: elliptic case.

Notice in this case that the whole region bounded by the BCs is both a domain of dependence and zone of influence.

The type of PDE fundamentally influences the choice of solution strategy. Time dependent hyperbolic problems and parabolic problems illustrated by Figures 3.1 and 3.2 are solved numerically by time marching methods which involves, as its name suggests, obtaining the numerical solution at a later time from that at an earlier time starting from given ICs.

Elliptic problems, as illustrated in Figure 3.3 are solved numerically by so-called relaxation methods.

Discrete Notation:

We will use upper case U to denote the analytic (exact) solution of the PDE and lower case u to denote the numerical (approximate) solution. Subscripts will denote discrete points in space and superscripts discrete levels in time. e.g. $u_{i,j}^{n}$ denotes the numerical solution at grid point (i, j) in a 2D region at time level n.

Checking Results:

Before applying a numerical scheme to real life situations modelled by PDEs there are two important steps that should always be undertaken.

Verification:

The computer program implementing the scheme must be verified. This is a check to see if the program is doing what it is supposed to do. Comparing results

from pen and paper calculations at a small number of points to equivalent computer output is a way to (partially) verify a program. Give or take a small amount of rounding error the numbers should be the same. Another way to verify the program is to find an exact solution to the PDE for a simpler problem (if one exists) and compare numerical and exact results. Complete program verification involves testing that all branches, program elements and statements are executed and produce the expected outcomes. For large programs there exist software verification programs to facilitate the verification process. For a commercial solver it may not be possible to completely verify the program if the source code is unavailable.

Validation:

Validation is really a check on whether the PDE is a good model for the real problem being studied. Validation means comparing numerical results with results from similar physical problems. Physical results may come from measurements from real life or from small-scale laboratory experiments. Either way, due to measurement errors, scaling problems and the inevitable failure of the PDEs to capture all the underlying physics, agreement between numerical and physical results will not be perfect and the user will have to decide what is 'close enough'.

Fundamentals:

The finite difference method (FDM) works by replacing the region over which the independent variables in the PDE are defined by a finite grid (also called a *mesh*) of points at which the dependent variable is approximated. The partial derivatives in the PDE at each grid point are approximated from neighbouring values by using Taylor's theorem.

Taylor's Theorem:

Let $U(x)$ have n continuous derivatives over the interval (a, b). Then for $a < x_0, x_0 + h < b$,

$$U(x_0 + h) = U(x_0) + hU_x(x_0) + h^2 \frac{U_{xx}(x_0)}{2!} + \cdots + h^{n-1} \frac{U_{(n-1)}(x_0)}{(n-1)!} + O(h^n), (3.2)$$

Where,

- $U_x = \frac{dU}{dx}, \quad U_{xx} = \frac{d^2U}{dx^2}, \ldots, U_{(n-1)} = \frac{d^{n-1}U}{dx^{n-1}}$
- $U_x(x_0)$ is the derivative of U with respect to x evaluated at $x = x_0$.
- $O(h^n)$ is an unknown error term.

The usual interpretation of Taylor's theorem says that if we know the value of U and the values of its derivatives at point x_0 then we can write down the equation (3.2) for its value at the (nearby) point $x_0 + h$. This expression contains an unknown quantity which is written in as $O(h^n)$ and pronounced 'order h to then'. If we discard the term $O(h^n)$ in (3.2) (i.e. truncate the right hand side of (3.2)) we get an approximation to $U(x_0 + h)$. The error in this approximation is $O(h^n)$.

Taylor's Theorem Applied to the Finite Difference Method (FDM):

In the FDM we know the U values at the grid points and we want to replace partial derivatives in the PDE we are solving by approximations at these grid points. We do this by interpreting (3.2) in another way. In the FDM both xo and $x_0 + h$ are grid points and $U(x_0)$. and $U(x_0 + h)$. are known. This allows us to rearrange equation (3.2) to get so-called Finite Difference (FD) approximations to derivatives which have $O(h^n)$ errors. Appendix A explains the meaning of $O(h^n)$ notation.

3.2 Finite Difference Approximation:

Simple Finite Difference Approximation to a Derivative:

Truncating (3.2) after the first derivative term gives,

$$U(x_0 + h) = U(x_0) + hU_x(x_0) + O(h^2) \tag{3.3}$$

Rearranging (2.2) gives,

$$U_x(x_0) = \frac{U(x_0 + h) - U(x_0)}{h} + \frac{O(h^2)}{h}$$

$$= \frac{U(x_0 + h) - U(x_0)}{h} + O(h)$$

Neglecting the $O(h)$ term gives,

$$U_x(x_0) \approx \frac{U(x_0 + h) - U(x_0)}{h} \tag{3.4}$$

Eq (3.4) is called a first order FD approximation to $U_x(x_0)$ as the approximation $error = O(h)$ which depends on the first power of h. This approximation is called a forward FD approximation since we start at x_0 and step forwards to the point $x_0 + h$. h is called the step size ($h > 0$).

Example: Simple Finite Difference Approximations to a Derivative:

This simple example shows that our forward difference approximation works and has the stated order of accuracy. We choose a simple function for U. Let $U(x) = x^2$. We will find the first order forward FD approximation to $U_x(3)$ using step size $h = 0.1$. From (3.4) the general first order forward FD approximation formula is,

$$U_x(x_0) \approx \frac{U(x_0 + h) - U(x_0)}{h} \tag{3.5}$$

Substituting for U gives,

$$U_x(x_0) \approx \frac{(x_0 + h)^2 - {x_0}^2}{h}$$

Replacing x_0 by 3 and h by 0.1 gives,

$$U_x(3) \approx \frac{(3 + 0.1)^2 - 3^2}{0.1} = 6.1$$

The exact answer from basic Calculus is clearly $U_x(3) = 6$ so the error in the approximation is $6.1 - 6 = 0.1$. Repeating the problem with $h = 0.05$ (i.e. half the step size) gives,

$$U_x(3) \approx \frac{(3 + 0.05)^2 - 3^2}{0.05} = 6.05$$

The error is $6.05 - 6 = 0.05$. The approximation formula (3.5) is first order so the errors should be proportional to h which is seen to be the case: halving the step size results in a halving of the error.

3.3 Explicit schemes:

Constructing a Finite Difference Toolkit:

We now construct common FD approximations to common partial derivatives. For simplicity we suppose that U is a function of only two variables, t and x. We will approximate the partial derivatives of U with respect to x. As t is held constant U is effectively a function of the single variable x so we can use Taylor's formula (3.1) where the ordinary derivative terms are now partial derivatives and the arguments are (t, x) instead of x. Finally we will replace the step size h by Δx (to indicate a change in x) so that (3.1) becomes,

$$U(t, x_0 + \Delta x) = U(t, x_0) + \Delta x U_x(t, x_0) + \frac{\Delta x^2}{2!} U_{xx}(t, x_0) + \cdots + \frac{\Delta x^{n-1}}{(n-1)!} U_{(n-1)}(t, x_0) + O(\Delta x^n) \quad (3.6a)$$

Truncating (2.5a) to $O(\Delta x^2)$ gives,

$$U(t, x_0 + \Delta x) = U(t, x_0) + \Delta x U_x(t, x_0) O(\Delta x^2) \quad (3.6b)$$

Now we derive some FD approximations to partial derivatives. Rearranging (3.6b) gives,

$$U_x(t, x_0) = \frac{U(t, x_0 + \Delta x) - U(t, x_0)}{\Delta x} - \frac{O(\Delta x^2)}{\Delta x}$$
$$\therefore U_x(t, x_0) = \frac{U(t, x_0 + \Delta x) - U(t, x_0)}{\Delta x} - O(\Delta x) \quad (3.7a)$$

Equation (3.7a) holds at any point (t, x_0). In numerical schemes for solving PDEs we are restricted to a grid of discrete x values, $x_1, x_2, \ldots, x_N$, and discrete t levels $0 = t_0, t_1, \ldots$. We will assume a constant grid spacing, Δx, in x, so that $x_{i+1} = x_i + \Delta x$. Evaluating Equation (3.7a) for a point, (t_n, x_i), on the grid gives,

$$U_x(t_n, x_i) = \frac{U(t_n, x_{i+1}) - U(t_n, x_i)}{\Delta x} - O\Delta x \quad (3.7b)$$

We will use the common subscript/superscript notation,

$$U_i^n = U(t_n, x_i) \tag{3.7c}$$

so that dropping the $O(\Delta x)$ error term, (3.6b) becomes,

$$U_x\,(t_n\,, x_i\,) \approx \frac{U_{i+1}^n - U_i^n}{\Delta x} \tag{3.7d}$$

(3.7d) is the first order forward difference approximation to $U_x\,(t_n\,, x_i\,)$ that we derived previously in approximation (3.5). We now derive another FD approximation to $U_x\,(t_n\,, x_i\,)$. Replacing Δx by $-\Delta x$ in (3.6b) gives,

$$U(t, x_0\,, -\Delta x) = U(t, x_0) - \Delta x U_x\,(t\,, x_0\,) + O(\Delta x^2\,) \tag{3.8a}$$

Evaluating (2.7a) at $(t_n\,, x_i\,)$ and rearranging as previously gives,

$$U_x\,(t_n\,, x_i\,) \approx \frac{U_i^n - U_{i-1}^n}{\Delta x} \tag{3.8b}$$

(3.8b) is the first order backward difference approximation to $U_x\,(t_n\,, x_i\,)$.

Our first two FD approximations are first order in x but we can increase the order (and so make the approximation more accurate) by taking more terms in the Taylor series as follows. Truncating (3.6a) to $O(\Delta x^3)$, then replacing Δx by $-\Delta x$ and subtracting this new expression from (3.6a) and evaluating at $(t_n\,, x_i\,)$ gives, after some algebra,

$$U_x\,(t_n\,, x_i\,) \approx \frac{U_{i+1}^n - U_{i-1}^n}{2\Delta x} \tag{3.9}$$

(3.89) is called the second order central difference FD approximation to $U_x\,(t_n\,, x_i\,)$.

We could construct even higher order FD approximations to U_x by taking even more terms in the Taylor series but we will stop at second order approximations to first order derivatives.

Many PDEs of interest contain second order (and higher) partial derivatives so we need to derive approximations to them. We will restrict our attention to second order unmixed partial derivatives i.e. U_{xx}.Truncating (3.6a) to $O(\Delta x^4)$ gives,

$$U(t, x_0 + \Delta x)$$
$$= U(t, x_0) + \Delta x U_x(t, x_0) + \frac{\Delta x^2}{2!} U_{xx}(t, x_0) + \frac{\Delta x^3}{3!} U_{xxx}(t, x_0)$$
$$+ O(\Delta x^4) \qquad (3.10a)$$

Replacing Δx by $-\Delta x$ in (3.10a) gives,

$$U(t, x_0 - \Delta x)$$
$$= U(t, x_0) + \Delta x U_x(t, x_0) + \frac{\Delta x^2}{2!} U_{xx}(t, x_0) - \frac{\Delta x^3}{3!} U_{xxx}(t, x_0)$$
$$+ O(\Delta x^4) \qquad (3.10b)$$

Adding (3.10a) and (3.10b) gives,

$$U(t, x_0 + \Delta x) = U(t, x_0 - \Delta x) = 2U(t, x_0) + \Delta x^2 U_{xx}(t, x_0) + O(\Delta x^4) \quad (3.11a)$$

Evaluating (3.11a) at (t_n, x_i) and using our discrete notation gives,

$$U_{i+1}^n + U_{i-1}^n$$
$$= 2U_i^n + \Delta x^2 U_{xx}(t_n, x_i) + O(\Delta x^4) \qquad (3.11b)$$

Rearranging (3.11b) and dropping the $O(\Delta x^2)$ error term gives,

$$U_{xx}(t_n, x_i) \approx \frac{U_{i+1}^n - 2U_i^n + U_{i-1}^n}{\Delta x^2} \qquad (3.12)$$

(3.12) is the second order symmetric difference FD approximation to $U_{xx}(t_n, x_i)$. These results are put into Table 2.1 to form a FD approximation toolkit. FD approximations to partial derivatives with respect to t are derived in a similar manner and are included in Table 2.1.

partial derivative	finite difference approximation	Type	Order
$\frac{\partial U}{\partial x} = U_x$	$\frac{U_{i+1}^n - U_i^n}{\Delta x}$	forward	first in x
$\frac{\partial U}{\partial x} = U_x$	$\frac{U_i^n - U_{i-1}^n}{\Delta x}$	backward	first in x
$\frac{\partial U}{\partial x} = U_x$	$\frac{U_{i+1}^n - U_{i-1}^n}{2\Delta x}$	central	second in x
$\frac{\partial^2 U}{\partial x^2} = U_{xx}$	$\frac{U_{i+1}^n - 2U_i^n U_{i-1}^n}{\Delta x^2}$	symmetric	second in x
$\frac{\partial U}{\partial t} = U_t$	$\frac{U_i^{n+1} - U_i^n}{\Delta t}$	forward	first in t
$\frac{\partial U}{\partial t} = U_t$	$\frac{U_i^n - U_i^{n-1}}{\Delta t}$	backward	first in t
$\frac{\partial U}{\partial t} = U_t$	$\frac{U_i^{n+1} - U_i^{n-1}}{2\Delta t}$	central	first in t
$\frac{\partial^2 U}{\partial t^2} = U_{tt}$	$\frac{U_i^{n+1} - 2U_i^n U_i^{n-1}}{\Delta t^2}$	symmetric	second in t

Table 3.1 Finite Difference Toolkit for Partial Derivatives

3.4 Implicit Schemes:

The previous schemes are called *explicit* schemes because data at the next time level is obtained from an explicit formula involving data from previous time levels. This leads to a (stability) restriction on the maximum allowable time step, Δt. Now we come to a different type of scheme – implicit, in which data from the next time level occurs on both sides of the difference scheme that necessitates solving a system of linear equations. There is no stability restriction on the maximum time step which may be much larger than an explicit scheme for the same problem. In implicit schemes the time step is chosen on the basis of accuracy considerations.

Example 6: Crank-Nicolson Scheme:

This is an implicit scheme. In previous examples spatial derivatives are approximated at time level n. However values change between time level n and time level $n + 1$ so a better approximation to spatial derivatives could make use of data at both time levels. Let,

$$\delta_x u_i^n = \alpha \frac{u_{i+1}^n - u_{i-1}^n}{2\Delta x} + (1-\alpha)\frac{u_{i+1}^{n+1} - u_{i+1}^{n-1}}{2\Delta x}, \qquad 0 \le \alpha \le 1. \tag{3.13}$$

This is a weighted average of central difference approximations at times levels n and $n+1$. Choose $\delta_{xx} u_i^n = 0$. Then for $\alpha = ½$ (3.13) becomes,

$$u_i^{n+1} = u_i^n - \frac{\mathrm{v}\Delta t}{2\Delta x}\left(\frac{u_{i+1}^n - u_{i-1}^n}{2} + \frac{u_{i+1}^{n+1} - u_{i-1}^{n+1}}{2}\right) (4.17)$$

This is the Crank-Nicolson scheme and it has the following stencil.

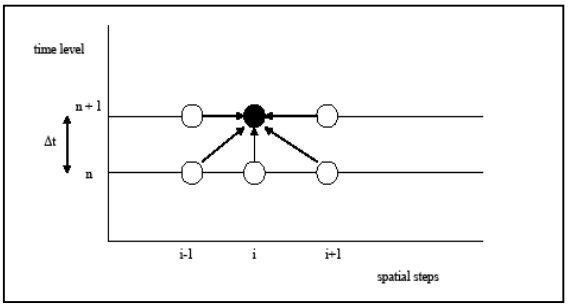

Figure 3.4 Stencil for the Crank-Nicolson Scheme Notes:

Notees:

1) The scheme is first order in time and second order in space.
2) Ghost values are required at both left and right ends of the computational domain.
3) The scheme is implicit so values at time level $n+1$ are found by solving a system of linear equations.

Implementation of the Crank-Nicolson Scheme:

Letting $c = \frac{\mathrm{v}\Delta t}{\Delta x}$, (3.14) is,

$$u_i^{n+1} = u_i^n - \frac{c}{2}\left(\frac{u_{i+1}^n - u_{i-1}^n}{2} + \frac{u_{i+1}^{n+1} - u_{i-1}^{n+1}}{2}\right) \tag{3.15}$$

Re-writing (3.16) gives,

$$4u_i^{n+1} = 4u_i^n - c\left(u_{i+1}^n - u_{i-1}^n + u_{i+1}^{n+1} - u_{i-1}^{n+1}\right) \tag{3.17a}$$

Rearranging so that data from the same time level is on the same side gives,

$$-cu_{i-1}^{n+1} + 4u_i^{n+1} + cu_{i+1}^{n+1} = cu_{i-1}^{n} + 4u_i^{n} - cu_{i+1}^{n} \qquad (4.19b)$$

The data at time level n is assumed known so (3.17b) is simplified by replacing the right hand side by d_i^n to give,

$$cu_{i-1}^{n+1} + 4u_i^{n+1} + cu_{i+1}^{n+1} = d_i^n \qquad (4.19c)$$

For each grid point $i = 1,2, \ldots, N$, we write out (3.17c) to give,

$$\begin{aligned}
-cu_0^{n+1} + 4u_1^{n+1} + cu_2^{n+1} &= d_1^{n'} \\
-cu_1^{n+1} + 4u_2^{n+1} + cu_3^{n+1} &= d_2^{n} \\
-cu_2^{n+1} + 4u_3^{n+1} + cu_4^{n+1} &= d_3^{n} \\
\vdots \qquad\qquad & \vdots \\
-cu_{N-2}^{n+1} + 4u_{N-1}^{n+1} + cu_N^{n+1} &= d_{N-1}^{n} \\
-cu_{N-1}^{n+1} + 4u_N^{n+1} + cu_{N+1}^{n+1} &= d_N^{n'}
\end{aligned} \qquad (4.19d)$$

(3.17d) is a system of N linear equations in what looks like $N + 2$ unknowns! However u_0^{n+1} and u_{N+1}^{n+1} on the left hand side of (3.17d) are ghost values which may be known directly or can be calculated in terms of neighbouring values depending on the type of boundary condition given in the problem (see Appendix B). In the former case we can move these known values to the right hand side of (3.17d) and, letting $d_1^{n'} = d_1^n + cu_0^{n+1}, d_N^{n'} = d_N^n - cu_{N+1}^{n+1}$, (3.17d) becomes,

$$\begin{aligned}
4u_1^{n+1} + cu_2^{n+1} &= d_1^{n'} \\
-cu_1^{n+1} + 4u_2^{n+1} + cu_3^{n+1} &= d_2^{n} \\
-cu_2^{n+1} + 4u_3^{n+1} + cu_4^{n+1} &= d_3^{n} \\
\vdots \qquad\qquad & \vdots \\
-cu_{N-2}^{n+1} + 4u_{N-1}^{n+1} + cu_N^{n+1} &= d_{N-1}^{n} \\
-cu_{N-1}^{n+1} + 4u_N^{n+1} &= d_N^{n'}
\end{aligned} \qquad (3.17e)$$

This system is expressed as the matrix equation,

$$\mathrm{A}\,\underline{u}^{n+1} = \underline{d}^{n} \qquad (3.17f)$$

Where,

$$A = \begin{pmatrix} 4 & c & 0 & \cdots & 0 & & & \\ -c & 4 & c & 0 & \cdots & 0 & & \\ 0 & -c & 4 & c & 0 & \cdots & 0 & \\ 0 & 0 & -c & 4 & c & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & & & 0 & -c & 4 & c \\ 0 & \cdots & & & & 0 & -c & 4 \end{pmatrix}, \underline{u}^{n+1} \begin{pmatrix} u_1^{n+1} \\ u_2^{n+1} \\ u_{N-1}^{n+1} \\ u_N^{n+1} \end{pmatrix}, \underline{d}^n \begin{pmatrix} d_1^{n'} \\ d_2^n \\ d_{N-1}^n \\ d_N^{n'} \end{pmatrix}$$

The solution to (3.17f) is obviously,

$$\underline{u}^{n+1} = A^{-1}\,\underline{d}^n \qquad (4.19g)$$

Notes:

1- (3.17g) is solved at each time step and the solution updated iteratively.
2- In practical problems A may be very large (e.g. 1000 x 1000) so an efficient matrix inversion method may be needed (see Chapter 3).
3- In this example boundary values are known so A is constant and needs only to be inverted once. For problems where the left hand side boundary values are calculated in terms of neighbouring values (e.g. Derivative boundary conditions, see Appendix B) the first and last rows of A will vary from time step to time step.
4- In our example A has a special structure – it is tridiagonal. This special structure permits the use of the efficient Thomas algorithm for matrix inversion algorithm (see downloadable code for Chapter 3).

3.5 Finite-difference Methods For Parabolic PDE

the Heat (or Diffusion) Parabolic PDE

The Diffusive Problem: (Heat or diffusion Equation)

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(D\frac{\partial u}{\partial x}\right) \qquad (3.18)$$

Where D is constant:

Explicit Methods:

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2}$$

By replacing

$$\frac{\partial u}{\partial t} = \frac{u_{n,j+1} - u_{n,j}}{\Delta t} + O(\Delta t) \quad \text{and} \quad \frac{\partial^2 u}{\partial x^2} = \frac{u_{n+1,j} - 2u_{n,j} + u_{n-1,j}}{(\Delta x)^2} + O(\Delta x)$$

$$\frac{u_{n,j+1} - u_{n,j}}{\Delta t} = D\frac{u_{n+1,j} - 2u_{n,j} + u_{n-1,j}}{(\Delta x)^2}$$

Or

$$u_{n,j+1} = \alpha u_{n+1,j} + (1 - 2\alpha)u_{n,j} + \alpha u_{n-1,j}$$

With $\alpha = \frac{D\Delta t}{(\Delta x)^2}$

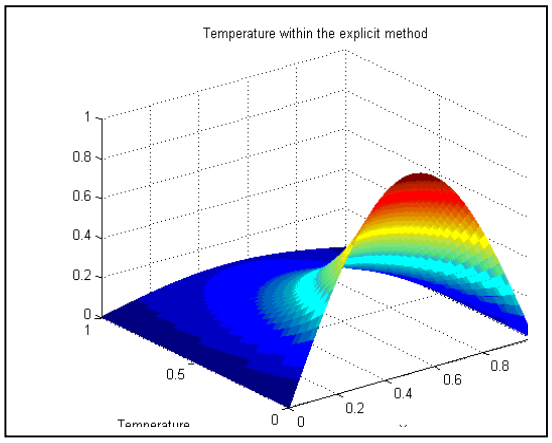

Figure3.5a

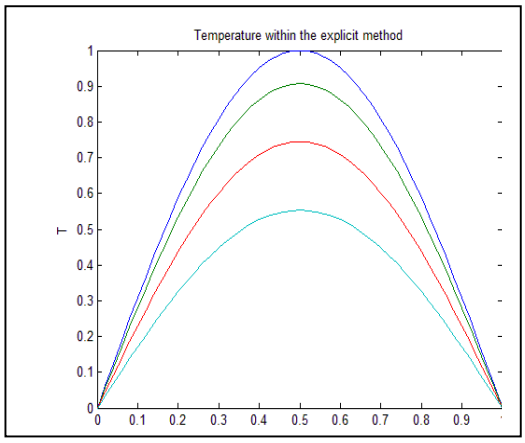

Figure3.5b

Stability Analysis: Von Neumann

This is a FTCS scheme again, but having a second derivative makes a world of difference. The FTCS was unstable for the advection equation (hyperbolic), but trying independent solutions of the form $u_{n,j} = \xi^j(k)e^{ikn\Delta x}$, we have that $\xi(k) = 1 - \frac{4D\Delta t}{(\Delta x)^2}\sin^2\left(\frac{k\Delta x}{2}\right)$, so that the requirement $|\xi(k)| \leq 1$, leads to the stability criterion:

$$\frac{2D\Delta t}{\Delta x^2} \leq 1$$

However, this condition implies hug limitation on the calculation procedure. For example, if we want to analyse with great detail in space ($\Delta x \ll 1$) implies that $\Delta t \ll 1$ so that a huge number of steps will be required until something

interesting happens. The computational requirement may be enormous and, therefore, new methods are required.

Implicit Methods:

A. Fully implicit scheme (or backward in time):

$$\frac{u_{n,j+1} - u_{n+1,j+1}}{\Delta t} = D\frac{u_{n+1,j+1} - 2u_{n,j+1} + u_{n-1,j+1}}{\Delta x^2} \quad (3.19)$$

This is like the FTCS scheme except that the spatial derivatives on the right-hand side are evaluated at time step $j + 1$.

One has to solve a set of simultaneous linear equation at each time step for the $u_{n,j+1}$. Fortunately, this is a simple problem because the system is tridiagonal: just group the terms in equation appropriately:

$$u_{n,j} = -\alpha u_{n-1,j+1} + (1 + 2\alpha)u_{n,j+1} - \alpha u_{n+1,j+1},$$
$$\text{with } n = 1,2, \dots, N-1 \text{ and } \alpha = \frac{D\Delta t}{\Delta x^2}$$

supplemented by Dirichlet or Neumann boundary conditions at $n = 0$ and $n = N$. These equations will be discussed in depth shortly.

What about stability?

The amplification factor is $\xi(k) = \frac{1}{1+4\alpha \sin^2\left(\frac{k\Delta x}{2}\right)}$, which clearly $|\xi(k)| \leq 1$ for any Δt. The scheme is unconditionally stable. The details of the small-scale evolution form the initial conditions are obviously inaccurate for large Δt (it is only first-order in time), but the correct equilibrium solution is obtained for $\Delta \to \infty$.

Keeping the mesh fixed and changing the conductivity:

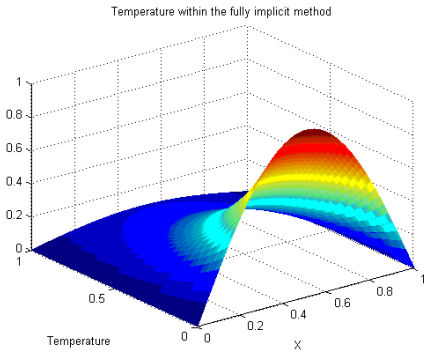

Figure3.6a

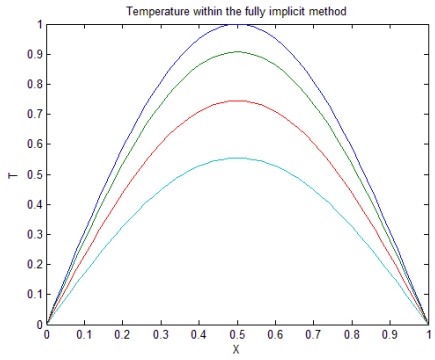

Figure3.6b

B. Crank-Nicholson Method (highly recommended):

Combines the stability of an implicit method with the accuracy of a method that is second-order in both space and time. Simply from the average of the explicit and implicit FTCS schemes (left-and right hand side are centered at time step $j + 1/2$):

$$\frac{u_{n,j+1} - u_{n,j}}{\Delta t} = \frac{D}{2}\left(\frac{\left(u_{n+1,j+1} - 2u_{n,j+1} + u_{n-1,j+1}\right) + \left(u_{n+1,j} - 2u_{n,j} + u_{n-1,j}\right)}{\Delta x^2}\right) \quad (3.20)$$

What about stability?

The amplification factor is: $\xi(k) = \frac{1-2\alpha \sin^2\left(\frac{k\Delta x}{2}\right)}{1+2\alpha \sin^2\left(\frac{k\Delta x}{2}\right)}$, which clearly $|\xi(k)| \leq 1$ for any Δt. The scheme is unconditionally stable and second-order both in time and space. It is worthy to analyse it more deeply.

The Crank-Nicholson can be written as:

$$\alpha u_{n-1,j} + (2 - 2\alpha)u_{n,j} + \alpha u_{n+1,j} = -\alpha u_{n-1,j+1} + (2 + 2\alpha)u_{n,j+1} - \alpha u_{n+1,j+1}$$

Theses equations only holds for $1 \leq n \leq N-1$. The boundary conditions again supply the two missing equations. They are harder to handle than in the explicit method and I will discuss them.

The Crank-Nicholson method can be written in a matrix form:

$$\begin{pmatrix} -\alpha & 2+2\alpha & -\alpha & 0 & \cdot & \cdot & \cdot \\ 0 & -\alpha & 2+2\alpha & -\alpha & \cdot & \cdot & \cdot \\ \cdot & 0 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 2+2\alpha & -\alpha & 0 \\ \cdot & \cdot & \cdot & \cdot & -\alpha & 2+2\alpha & -\alpha \end{pmatrix} \begin{pmatrix} u_{0,j+1} \\ u_{1,j+1} \\ \cdot \\ \cdot \\ \cdot \\ u_{N-1,j+1} \\ u_{N,j+1} \end{pmatrix}$$

$$= \begin{pmatrix} \alpha & 2-2\alpha & \alpha & 0 & \cdot & \cdot & \cdot \\ 0 & \alpha & 2-2\alpha & \alpha & \cdot & \cdot & \cdot \\ \cdot & 0 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 2-2\alpha & \alpha & 0 \\ \cdot & \cdot & \cdot & \cdot & \alpha & 2-2\alpha & \alpha \end{pmatrix} \begin{pmatrix} u_{0,j} \\ u_{1,j} \\ \cdot \\ \cdot \\ \cdot \\ u_{N-1,j} \\ u_{N,j} \end{pmatrix}$$

The two matrices have $N-1$ rows and $N+1$ columns, which is a representations of the $N-1$ equations and $N+1$ unknowns. The two equations that we are missing come from the boundary conditions. Using these conditions, I am going to convert this system of equations into a system of equations involving a square matrix. The aim is to write a system of equations in the form:

$$\mathbf{M}_{j+1}^{L}\mathbf{u}_{j+1} + \mathbf{r}_{j+1}^{L} = \mathbf{M}_{j}^{R}u_{j} + \mathbf{r}_{j}^{R}$$

For known square matrices $\mathbf{M}_{j+1}^{L}$ and $\mathbf{M}_{j}^{R}$, and a known vectors $\mathbf{r}$, where the details of the boundary conditions have been fully incorporated.

Example of boundary condition: given $u_{0,j+1}$ and $u_{N,j+1}$: Sometimes we know the value of the u function in the boundary ($n = 0$ and $n = N$). In this case, write:

$$\begin{pmatrix} -\alpha & 2+2\alpha & -\alpha & 0 & \cdot & \cdot & \cdot \\ 0 & -\alpha & 2+2\alpha & -\alpha & \cdot & \cdot & \cdot \\ \cdot & 0 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 2+2\alpha & -\alpha & 0 \\ \cdot & \cdot & \cdot & \cdot & -\alpha & 2+2\alpha & -\alpha \end{pmatrix} \begin{pmatrix} u_{0,j+1} \\ u_{1,j+1} \\ \cdot \\ \cdot \\ \cdot \\ u_{N-1,j+1} \\ u_{N,j+1} \end{pmatrix}$$

as

$$\begin{pmatrix} 2+2\alpha & -\alpha & \cdot & \cdot & \cdot & \cdot \\ -\alpha & 2+2\alpha & \cdot & \cdot & \cdot & \cdot \\ 0 & \cdot & \cdot & \cdot & -\alpha & \cdot \\ \cdot & \cdot & \cdot & -\alpha & 2+2\alpha & -\alpha \\ \cdot & \cdot & \cdot & 0 & -\alpha & 2+2\alpha \end{pmatrix} \begin{pmatrix} u_{1,j+1} \\ \cdot \\ \cdot \\ \cdot \\ u_{N-1,j+1} \end{pmatrix} + \begin{pmatrix} -\alpha u_{0,j+1} \\ \cdot \\ \cdot \\ \cdot \\ -\alpha u_{N,j+1} \end{pmatrix}$$

$$= \mathbf{M}_{j+1}^{L} + \mathbf{u}_{j+1} + \mathbf{r}_{j+1}^{L}$$

and the same for the matrices on the right.

Whichever of the boundary conditions we have, the Crank-Nicholson scheme, with boundary conditions incorporated is:

$$\mathbf{u}_{j+1} = \left(\mathbf{M}_{j+1}^{L}\right)^{-1}\left(\mathbf{r}_{j}^{R} - \mathbf{r}_{j+1}^{L} + \mathbf{M}_{j}^{R}\mathbf{u}_{j}\right)$$

However, matrix inversion is very time consuming and computationally inefficient. Two much better ways will be explained below:

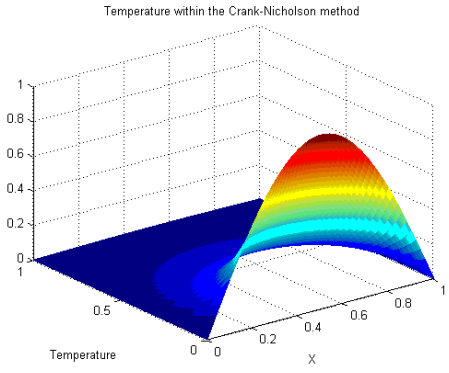

Figure3.7a

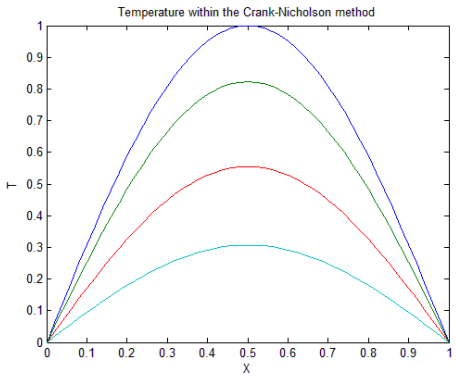

Figure3.7b

References

[1] C. H. Edwards and D. E. Penney, Elementry Diffrential Equations-Sixth Edition, 2008.

[2] J. F. Kuritis D.Fink , Numerical Methods Using Matlab-Third Edition

California State University, Northwest Missoun State University ,1999.

[3] D. M. Causon and C. G. Mingham ,Introductory Finite Difference Methods for PDEs ,2010.

[4] A. Bergara, Finite-difference Numerical Methods of Partial Differential Equations in Finance with Matlab [Lecture notes]. Retrieved from 'http://www.ehu.eus/aitor/irakas/fin/apuntes/pde.pdf'. 2014.

[5] Birkhoff and G.-c. Rota, Ordinary Differential Equations,,John Wiley, New York ,1989.

[6] H. William Press et aI , Numerical Recipes: The Art of Scientific Computing New York: Cambridge University Press, 1986.

[7] O. B. Francis, Jr. et al. for the NASA-George , Study of the Methods for the Numerical Solution of Ordinary Differential Equations , NASA Contractor Report CR-61139, C. Marshall Space Flight Center, June 7, 1966.

[8] E.Ps. Penney , Calculus: Early Transcendentals, 7th ed, Upper Saddle River, NJ: Prentice HalI, 2008.

Appendix

Program (1):

```
z=-5:5:10;
y = zeros(size(z));
y(1) = -3;
h=1.0;
   x = 0.0:h:5.0;
   for i = 1:(length(x)-1)
       y(i+1) = y(i) + h*(x(i)+1/5.*y(i));
   end

hold on

  plot(x,y,'r*')
    title('Numerical Approximation:Euler Method for dy/dx=x+(1/5)y')
    xlabel('x')
    ylabel('y')
    hold off
    h1=0.2;
   x = 0.0:h1:5.0;
   for i = 1:(length(x)-1)
       y(i+1) = y(i) + h1*(x(i)+1/5.*y(i));
   end

hold on

  plot(x,y,'b*')

    hold off
    h2=0.05;
   x = 0.0:h2:5.0;
   for i = 1:(length(x)-1)
       y(i+1) = y(i) + h2*(x(i)+1/5.*y(i));
   end

hold on

  plot(x,y,'g*')

    hold off

hold on
 exact = 22.*exp(x./5)-5.*x-25;
 plot(x,exact,'b--')
 legend('h=1.0','h=0.2','h=0.05','exact')
 hold off
```

Program (2):

```
%function [x,y] = res2(x,y,x1,n)
x=0,x1=1,y=1;n=10;

h=0.005;
%h=(x1-x)/n;
X=x;
Y=y;
for i=1:n
    k1=f(x,y);
    k2=f(x+h,y+h*k1);
    k=(k1+k2)/2;
    x=x+h;
    y=y+h*k;
    X=[X;x]
    Y=[Y;y]

end
hold on
exact=2*exp(X)-X-1;
plot(X,Y,'r*',X,exact,'b-')
xlabel('x')
ylabel('y')
title('The Improvement Euler Method for dy/dx=x+y')
legend('h=0.005','exact')
hold off
%end

%x=0,y=1,x1=1,n=10
```

Program (3):

```
%function [ x,y] =res3( x,y,x1,n )
x = 0; y = 1; x1 = 1; n = 10;
xx=x;
Y=y;
xx = zeros(n,1);
yy = zeros(n,1);
h=(x1-x)/n;
for i=1:n
    k1=f(x,y);
    k2=f(x+h/2,y+h*k1/2);
    k3=f(x+h/2,y+h*k2/2);
    k4=f(x+h,y+h*k3);
    k=(k1+2*k2+2*k3+k4)/6;
    x=x+h;
    y=y+h*k;
    xx(i)=x
    yy(i)=y

end
hold on
plot(xx,yy,'r*')
xlabel('x')
ylabel('y')
title('The Runge-Kutta(4 step) Method for dy/dx=x+y')
excat=2.*exp(xx)-xx-1;
plot(xx,excat)
legend('h=0.1','exact')
hold off
%end
%x=0,y=1,x1=1,n=10
```

Program (4):

```
clc
clear
x0=3;y0=6;h=0.1;
x=zeros(2);
y=zeros(2);
x(1)=x0;
y(1)=y0;
for j = 1:2
x(j+1) = x(j)+h*(3*x(j)-2*y(j));
y(j+1) = y(j)+h*(5*x(j)-4*y(j));
x(1)=x(2);
y(1)=y(2);
end
%Euler method for systems
a=[x(2) y(2);x(3) y(3)];
X=a(1:2,1), Y=a(1:2,2)
```

Program (5):

```
function [ T Z] = res6(F,a,b,Za,M  )
%Rung kutta for system
h=(b-a)/M;
T=zeros(1,M+1);
Z=zeros(M+1,length(Za));
T=a:h:b;
Z(1,:)=Za;
for j=1:M
    k1=h*feval(F,T(j),Z(j,:));
    k2=h*feval(F,T(j)+h/2,Z(j,:)+k1/2);
    k3=h*feval(F,T(j)+h/2,Z(j,:)+k2/2);
    k4=h*feval(F,T(j)+h,Z(j,:)+k3);
    Z(j+1,:)=Z(j,:)+(k1+2*k2+2*k3+k4)/6
end
end
%function[f g]=aaa()
%f=inline('x+2*y');
%g=inline('3*x+2*y');
%end
%Za=[6,4];
%M=10;
%a=0.0,b=0.2;
%res6(aaa,a,b,Za,M)
```

Program (6):

```
% Matlab Program 5: Heat Diffusion in one dimensional wire within the
% Explicit Method
clear;
% Parameters to define the heat equation and the range in space and time
L = 1.; % Length of the wire
T =1.; % Final time
% Parameters needed to solve the equation within the explicit method
maxk = 2500; % Number of time steps
dt = T/maxk;
n = 50; % Number of space steps
dx = L/n;
cond = 1/4; % Conductivity
b = 2.*cond*dt/(dx*dx); % Stability parameter (b=<1)
% Initial temperature of the wire: a sinus.
for i = 1:n+1
x(i) =(i-1)*dx;
u(i,1) =sin(pi*x(i));
end
% Temperature at the boundary (T=0)
for k=1:maxk+1
u(1,k) = 0.;
u(n+1,k) = 0.;
time(k) = (k-1)*dt;
end
% Implementation of the explicit method
for k=1:maxk % Time Loop
for i=2:n; % Space Loop
u(i,k+1) =u(i,k) + 0.5*b*(u(i-1,k)+u(i+1,k)-2.*u(i,k));
end
end
% Graphical representation of the temperature at different selected times
figure(1)
plot(x,u(:,1),'-',x,u(:,100),'-',x,u(:,300),'-',x,u(:,600),'-')
title('Temperature within the explicit method')
xlabel('X')
ylabel('T')
figure(2)
mesh(x,time,u')
title('Temperature within the explicit method')
xlabel('X')
ylabel('Temperature')
%analytic solution
 u0=zeros(25,100);
for i = 1:n
xx(i) =(i-1)*dx;
end

for k=1:maxk
timee(k) = (k-1)*dt;
end

for k=1:maxk % Time Loop
for i=1:n; % Space Loop

u0(i,k) = sin(pi.*xx(i))*exp(-pi*pi.*timee(k));
end
end
figure(3)
mesh(xx,timee,u0')
title('Temperature analytic solution')
xlabel('X')
ylabel('Temperature')
```

Program (7):

```
% Matlab Program i: Heat Diffusion in one dimensional wire within the Fully
% Implicit Method
clear;
% Parameters to define the heat equation and the range in space and time
L = 1.; % Lenth of the wire
T =1.; % Final time
% Parameters needed to solve the equation within the fully implicit method
maxk = 2500; % Number of time steps
dt = T/maxk;
n = 50.; % Number of space steps
dx = L/n;
cond = 1./4.; % Conductivity
b = cond*dt/(dx*dx); % Parameter of the method
% Initial temperature of the wire: a sinus.
for i = 1:n+1
x(i) =(i-1)*dx;
u(i,1) =sin(pi*x(i));
end
% Temperature at the boundary (T=0)
for k=1:maxk+1
u(1,k) = 0.;
u(n+1,k) = 0.;
time(k) = (k-1)*dt;
end
aa(1:n-2)=-b;
bb(1:n-1)=1.+2.*b;
cc(1:n-2)=-b;
MM=inv(diag(bb,0)+diag(aa,-1)+diag(cc,1));
% Implementation of the implicit method
for k=2:maxk % Time Loop
uu=u(2:n,k-1);
u(2:n,k)=MM*uu;
end
% Graphical representation of the temperature at different selected times
figure(1)
plot(x,u(:,1),'-',x,u(:,100),'-',x,u(:,300),'-',x,u(:,600),'-')
title('Temperature within the fully implicit method')
xlabel('X')
ylabel('T')
figure(2)
mesh(x,time,u')
title('Temperature within the fully implicit method')
xlabel('X')
ylabel('Temperature')
```

Program (8):

```
% Matlab Program : Heat Diffusion in one dimensional wire within the
% Crank-Nicholson Method
clear;
% Parameters to define the heat equation and the range in space and time
L = 1.; % Lenth of the wire
T =1.; % Final time
% Parameters needed to solve the equation within the Crank-Nicholson method
maxk = 2500; % Number of time steps
dt = T/maxk;
n = 50.; % Number of space steps
dx = L/n;
cond = 1/2; % Conductivity
b = cond*dt/(dx*dx); % Parameter of the method
% Initial temperature of the wire: a sinus.
for i = 1:n+1
x(i) =(i-1)*dx;
u(i,1) =sin(pi*x(i));
end
% Temperature at the boundary (T=0)
for k=1:maxk+1
u(1,k) = 0.;
u(n+1,k) = 0.;
time(k) = (k-1)*dt;
end
% Defining the Matrices M_right and M_left in the method
aal(1:n-2)=-b;
bbl(1:n-1)=2.+2.*b;
ccl(1:n-2)=-b;
MMl=diag(bbl,0)+diag(aal,-1)+diag(ccl,1);
aar(1:n-2)=b;
bbr(1:n-1)=2.-2.*b;
ccr(1:n-2)=b;
MMr=diag(bbr,0)+diag(aar,-1)+diag(ccr,1);
% Implementation of the Crank-Nicholson method
for k=2:maxk % Time Loop
uu=u(2:n,k-1);
u(2:n,k)=inv(MMl)*MMr*uu;
end
% Graphical representation of the temperature at different selected times
figure(1)
plot(x,u(:,1),'-',x,u(:,100),'-',x,u(:,300),'-',x,u(:,600),'-')
title('Temperature within the Crank-Nicholson method')
xlabel('X')
ylabel('T')
figure(2)
mesh(x,time,u')
title('Temperature within the Crank-Nicholson method')
xlabel('X')
ylabel('Temperature')
```