Chapter 1

Discrete Time-Space Models

The first three sections introduces diffusion of heat in one direction. This is an example of model evolution with the simplest model being for the temperature of a well stirred liquid where the temperature does not vary with space. The model is then enhanced by allowing the mass to have different temperatures in different locations. Because heat flows from hot to cold regions, the subsequent model will be more complicated. In section four a similar model is considered, and the application will be to the prediction of the pollutant concentration in a stream resulting from a source of pollution up stream. Both these models are discrete versions of the continuous model that are partial differential equations. Section five indicates how these models can be extended to heat and mass transfer in two directions, which is discussed in more detail in chapters three and four. In the last section variations of the mean value theorem are used to estimate the errors made by replacing the continuous model by a discrete model. Additional introductory materials can be found in G. D. Smith [18], and in R. L. Burden and J. D. Faires [3].

1.1 Newton Cooling Models

1.1.1 Introduction

Many quantities change as time progresses such as money in a savings account or the temperature of a refreshing drink or any cooling mass. Here we will be interested in making predictions about such changing quantities. A simple mathematical model has the form $u^+ = au + b$ where a and b are given real numbers, u is the present amount and u^+ is the next amount. This calculation is usually repeated a number of times and is a simple example of an of algorithm. Because of the large number of repeated calculations, we usually use a

of computing tool.

Computers use a finite subset of the rational numbers (a ratio of two integers) to approximate any real number. This set of numbers may depend on the computer being used. However, they do have the same general form and are called floating point numbers. Any real number x can be represented by an infinite decimal expansion $x = \pm(.x_1.....x_d....)10^e$, and by truncating this we can define the chopped floating point numbers.

Let x be any real number and denote a *floating point number* by

$$\begin{aligned} fl(x) &= \pm.x_1.....x_d 10^e \\ &= \pm(x_1/10 + \cdots + x_d/10^d)10^e. \end{aligned}$$

This is a floating point number with base equal to 10 where x_1 is not equal to zero, x_i are integers between 0 and 9, the exponent e is also a bounded integer and d is an integer called the precision of the floating point system. Associated with each real number, x, and its floating point approximate number, $fl(x)$, is the *floating point error*, $fl(x) - x$. In general, this error decreases as the *precision*, d, increases. Each computer calculation has some floating point or roundoff error. Moreover, as additional calculations are done, there is an accumulation of these roundoff errors.

Example. Let $x = -1.5378$ and $fl(x) = -0.154\ 10^1$ where $d = 3$. The roundoff error is

$$fl(x) - x = -.0022.$$

The error will accumulate with any further operations containing $fl(x)$, for example, $fl(x)^2 = .237\ 10^{-1}$ and

$$fl(x)^2 - x^2 = 2.37 - 2.36482884 = .00517116.$$

Repeated calculations using floating point numbers can accumulate significant roundoff errors.

1.1.2 Applied Area

Consider the cooling of a well stirred liquid so that the temperature does not depend on space. Here we want to predict the temperature of the liquid based on some initial observations. Newton's law of cooling is based on the observation that for small changes of time, h, the change in the temperature is nearly equal to the product of the some constant c, the h and the difference in the room temperature and the present temperature of the coffee. Consider the following quantities: u_k equals the temperature of a well stirred cup of coffee at time t_k, u_{sur} equals the surrounding room temperature, and c measures the insulation ability of the cup and is a positive constant. The *discrete form of Newton's law of cooling* is

$$\begin{aligned} u_{k+1} - u_k &= ch(u_{sur} - u_k) \\ u_{k+1} &= (1 - ch)u_k + ch\ u_{sur} \\ &= au_k + b\ where\ a = 1 - ch\ and\ b = ch\ u_{sur}. \end{aligned}$$

The long run solution should be the room temperature, that is, u_k should converge to u_{sur} as k increases. Moreover, when the room temperature is constant, then u_k should converge monotonically to the room temperature. This does happen if we impose the

$$0 < a = 1 - ch$$

constraint, called a *stability condition*, on the time step h. Since both c and h are positive, $a < 1$.

1.1.3 Model

The model in this case appears to be very simple. It consists of three constants u_0, a, b and the formula

$$u_{k+1} = au_k + b \tag{1.1.1}$$

The formula must be used repeatedly, but with different u_k being put into the right side. Often a and b are derived from formulating how u_k changes as k increases (k reflects the time step). The change in the amount u_k is often modeled by $du_k + b$

$$u_{k+1} - u_k = du_k + b$$

where $d = a-1$. This model given in (1.1.1) is called a *first order finite difference* model for the sequence of numbers u_{k+1}. Later we will generalize this to a sequence of column vectors where a will be replaced by a square matrix.

1.1.4 Method

The "iterative or recursive" calculation of (1.1.1) is the most common approach to solving (1.1.1). For example, if $a = \frac{1}{2}, b = 2$ and $u_0 = 10$, then

$$\begin{aligned} u_1 &= \frac{1}{2}\ 10 + 2 = 7.0 \\ u_2 &= \frac{1}{2}\ 7 + 2 = 5.5 \\ u_3 &= \frac{1}{2}\ 5.5 + 2 = 4.75 \\ u_4 &= \frac{1}{2}\ 4.75 + 2 = 4.375 \end{aligned}$$

If one needs to compute u_{k+1} for large k, this can get a little tiresome. On the other hand, if the calculations are being done with a computer, then the floating point errors may generate significant accumulation errors.

An alternative method is to use the following "telescoping" calculation and the geometric summation. Recall the geometric summation

$$1 + r + r^2 + \cdots + r^k \text{ and } (1 + r + r^2 + \cdots + r^k)(1 - r) = 1 - r^{k+1}$$

Or, for r not equal to 1

$$(1 + r + r^2 + \cdots + r^k) = (1 - r^{k+1})/(1 - r).$$

Consequently, if $|r| < 1$, then

$$1 + r + r^2 + \cdots + r^k + \cdots = 1/(1 - r)$$

is a convergent *geometric series*.

In (1.1.1) we can compute u_k by decreasing k by 1 so that $u_k = au_{k-1} + b$. Put this into (1.1.1) and repeat the substitution to get

$$\begin{aligned} u_{k+1} &= a(au_{k-1} + b) + b \\ &= a^2u_{k-1} + ab + b \\ &= a^2(au_{k-2} + b) + ab + b \\ &= a^3u_{k-2} + a^2b + ab + b \\ &\vdots \\ &= a^{k+1}u_0 + b(a^k + \cdots + a^2 + a + 1) \\ &= a^{k+1}u_0 + b(1 - a^{k+1})/(1 - a) \\ &= a^{k+1}(u_0 - b/(1 - a)) + b/(1 - a). \end{aligned} \tag{1.1.2}$$

The error for the steady state solution will be small if $|a|$ is small, or k is large, or the initial guess u_0 is close to the steady state solution $b/(1 - a)$.

Theorem 1.1.1 *(Steady State Theorem) If a is not equal to 1, then the solution of (1.1.1) has the form given in (1.1.2). Moreover, if $|a| < 1$, then the solution of (1.1.1) will converge to the steady state solution $u = au + b$, that is, $u = b/(1 - a)$. More precisely, the error is*

$$u_{k+1} - u = a^{k+1}(u_0 - b/(1 - a)).$$

Example. Let $a = 1/2, b = 2, u_0 = 10$ and $k = 3$. Then (1.1.2) gives

$$u_{3+1} = (1/2)^4(10 - 2/(1 - 1/2)) + 2/(1 - 1/2) = 6/16 + 4 = 4.375.$$

The steady state solution is $u = 2/(1 - \frac{1}{2}) = 4$ and the error for $k = 3$ is

$$u_4 - u = 4.375 - 4 = (\frac{1}{2})^4(10 - 4).$$

1.1.5 Implementation

The reader should be familiar with the information in MATLAB's tutorial. The input segment of the MATLAB code fofdh.m is done in lines 1-12, the execution is done in lines 16-19, and the output is done in line 20. In the following m-file t is the time array whose first entry is the initial time. The array y stores the

approximate temperature values whose first entry is the initial temperature. The value of c is based on a second observed temperature, y_obser, at time equal to h_obser. The value of c is calculated in line 10. Once a and b have been computed, the algorithm is executed by the for loop in lines 16-19. Since the time step $h = 1$, $n = 300$ will give an approximation of the temperature over the time interval from 0 to 300. If the time step were to be changed from 1 to 5, then we could change n from 300 to 60 and still have an approximation of the temperature over the same time interval. Within the for loop we could look at the time and temperature arrays by omitting the semicolon at the end of the lines 17 and 18. It is easier to examine the graph of approximate temperature versus time, which is generated by the MATLAB command plot(t,y).

MATLAB Code fofdh.m

```
1.   % This code is for the first order finite difference algorithm.
2.   % It is applied to Newton's law of cooling model.
3.   clear;
4.   t(1) = 0;      % initial time
5.   y(1) = 200.;      % initial temperature
6.   h = 1;      % time step
7.   n = 300;      % number of Time steps of length h
8.   y_obser = 190;      % observed temperature at time h_obser
9.   h_obser = 5;
10.   c = ((y_obser - y(1))/h_obser)/(70 - y(1))
11.   a = 1 - c*h
12.   b = c*h*70
13.   %
14.   % Execute the FOFD Algorithm
15.   %
16.   for k = 1:n
17.        y(k+1) = a*y(k) + b;
18.        t(k+1) = t(k) + h;
19.   end
20.   plot(t,y)
```

An application to heat transfer is as follows. Consider a cup of coffee, which is initially at 200 degrees and is in a room with temperature equal to 70, and after 5 minutes it cools to 190 degrees. By using $h = h\_obser = 5$, $u_0 = 200$ and $u_1 = u\_obser = 190$, we compute from (1.1.1) that $c = 1/65$. The first calculation is for this c and $h = 5$ so that $a = 1 - ch = 60/65$ and $b = ch70 = 350/65$. Figure 1.1.1 indicates the expected monotonic decrease to the steady state room temperature, $u_{sur} = 70$.

The next calculation is for a larger $c = 2/13$, which is computed from a new second observed temperature of $u\_obser = 100$ after $h\_obser = 5$ minutes. In this case for larger time step $h = 10$ so that $a = 1 - (2/13)10 = -7/13$ and $b = ch70 = (2/13)10\ 70 = 1400/13$. In Figure 1.1.2 notice that the

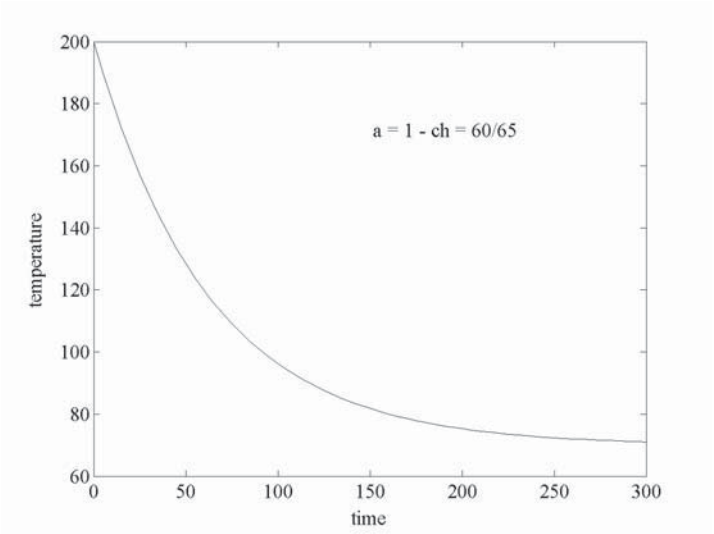

Figure 1.1.1: Temperature versus Time

computed solution no longer is monotonic, but it does converge to the steady state solution.

The model continues to degrade as the magnitude of a increases. In the Figure 1.1.3 the computed solution oscillates and blows up! This is consistent with formula (1.1.2). Here we kept the same c, but let the step size increase to $h = 15$ and in this case $a = 1 - (2/13)15 = -17/13$ and $b = ch70 = (2/13)1050 = 2100/13$. The vertical axis has units multiplied by 10^4.

1.1.6 Assessment

Models of savings plans or loans are discrete in the sense that changes only occur at the end of each month. In the case of the heat transfer problem, the formula for the temperature at the next time step is only an approximation, which gets better as the time step h decreases. The cooling process is continuous because the temperature changes at every instant in time. We have used a discrete model of this, and it seems to give good predictions provided the time step is suitably small. Moreover there are other modes of transferring heat such as diffusion and radiation.

There may be significant accumulation of roundoff error. On a computer (1.1.1) is done with floating point numbers, and at each step there is some new roundoff error $\overline{R}_{k+1}$. Let $U_0 = fl(u_0), A = fl(a)$ and $B = fl(b)$ so that

$$U_{k+1} = AU_k + B + \bar{R}_{k+1}. \tag{1.1.3}$$

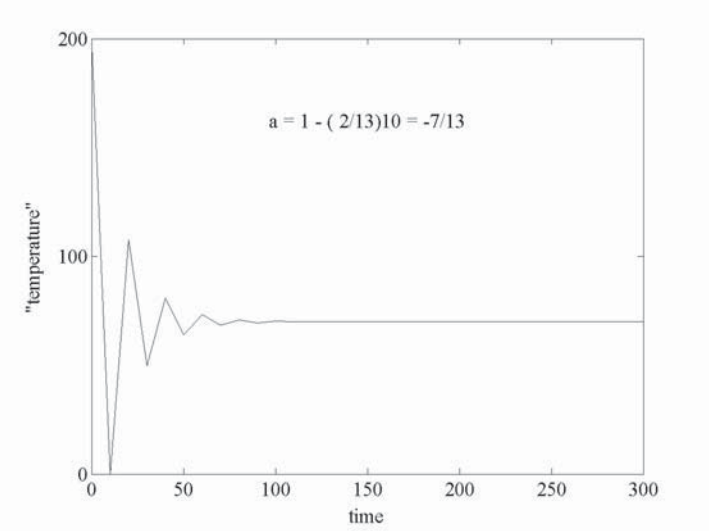

Figure 1.1.2: Steady State Temperature

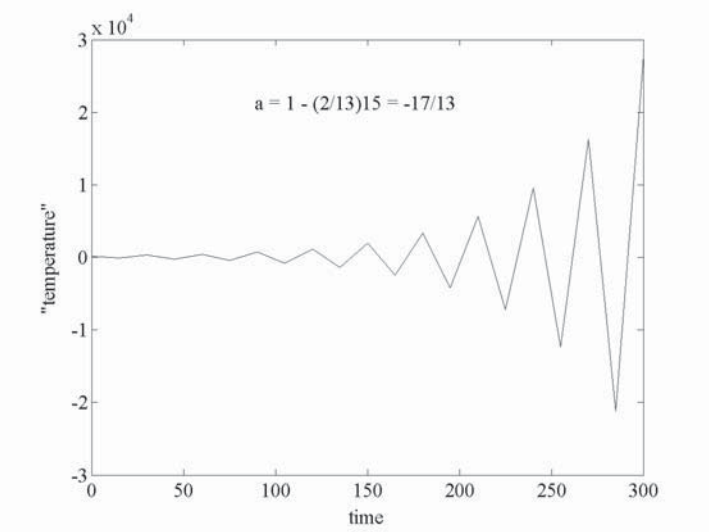

Figure 1.1.3: Unstable Computation

Next, we want to estimate the

$$accumulation\ errors = U_{k+1} - u_{k+1}$$

under the assumption that the roundoff errors are uniformly bounded

$$|\overline{R}_{k+1}| \leq R < \infty.$$

For ease of notation, we will assume the roundoff errors associated with a and b have been put into the R_{k+1} so that $U_{k+1} = aU_k + b + R_{k+1}$. Subtract (1.1.1) and this variation of (1.1.3) to get

$$\begin{aligned} U_{k+1} - u_{k+1} &= a(U_k - u_k) + R_{k+1} \\ &= a[a(U_{k-1} - u_{k-1}) + R_k] + R_{k+1} \\ &= a^2(U_{k-1} - u_{k-1}) + aR_k + R_{k+1} \\ &\vdots \\ &= a^{k+1}(U_0 - u_0) + a^k R_1 + \cdots + R_{k+1} \end{aligned} \tag{1.1.4}$$

Now let $r = |a|$ and R be the uniform bound on the roundoff errors. Use the geometric summation and the triangle inequality to get

$$|U_{k+1} - u_{k+1}| \leq r^{k+1}|U_0 - u_0| + R(r^{k+1} - 1)/(r - 1). \tag{1.1.5}$$

Either r is less than one, or greater, or equal to one. An analysis of (1.1.4) and (1.1.5) immediately yields the next theorem.

Theorem 1.1.2 *(Accumulation Error Theorem) Consider the first order finite difference algorithm. If $|a| < 1$ and the roundoff errors are uniformly bounded by R, then the accumulation error is uniformly bounded. Moreover, if the roundoff errors decreases uniformly, then the accumulation error decreases.*

1.1.7 Exercises

1. Using fofdh.m duplicate the calculations in Figures 1.1.1-1.1.3.
2. Execute fofdh.m four times for $c = 1/65$, variable $h = 64$, 32, 16, 8 with $n = 5$, 10, 20 and 40, respectively. Compare the four curves by placing them on the same graph; this can be done by executing the MATLAB command "hold on" after the first execution of fofdh.m
3. Execute fofdh.m five times with $h = 1$, variable $c = 8/65$, $4/65$, $2/65$, $1/65$, and $.5/65$, and $n = 300$. Compare the five curves by placing them on the same graph; this can be done by executing the MATLAB command "hold on" after the first execution of fofdh.m
4. Consider the application to Newton's discrete law of cooling. Use (1.1.2) to show that if $hc < 1$, then u_{k+1} converges to the room temperature.
5. Modify the model used in Figure 1.1.1 to account for a room temperature that starts at 70 and increases at a constant rate equal to one degree every five

minutes. Use the $c = 1/65$ and $h = 1$. Compare the new curve with Figure 1.1.1.

6. We wish to calculate the amount of a *savings plan* for any month, k, given a fixed interest rate, r, compounded monthly. Denote these quantities as follows: u_k be the amount in an account at month k, r equals the interest rate compounded monthly, and d equals the monthly deposit. The amount at the end of the next month will be the old amount plus the interest on the old amount plus the deposit. In terms of the above variables this is with $a = 1 + r/12$ and $b = d$

$$\begin{aligned} u_{k+1} &= u_k + u_k\ r/12 + d \\ &= au_k + b. \end{aligned}$$

(a). Use (1.1.2) to determine the amount in the account by depositing \$100 each month in an account, which gets 12% compounded monthly, and over a time interval of 30 and 40 years (360 and 480 months).

(b). Use a modified version of fofdh.m to calculate and graph the amounts in the account from 0 to 40 years.

7. Show (1.1.5) follows from (1.1.4).

8. Prove the second part of the accumulation error theorem.

1.2 Heat Diffusion in a Wire

1.2.1 Introduction

In this section we consider heat conduction in a thin electrical wire, which is thermally insulated on its surface. The model of the temperature has the form $u^{k+1} = Au^k + b$ where u^k is a column vector whose components are temperatures for the previous time step, $t = k\Delta t$, at various positions within the wire. The square matrix will determine how heat flows from warm regions to cooler regions within the wire. In general, the matrix A can be extremely large, but it will also have a special structure with many more zeros than nonzero components.

1.2.2 Applied Area

In this section we present a second model of heat transfer. In our first model we considered heat transfer via a discrete version of Newton's law of cooling which involves temperature as only a discrete function of time. That is, we assumed the mass was uniformly heated with respect to space. In this section we allow the temperature to be a function of both discrete time and discrete space.

The model for the diffusion of heat in space is based on empirical observations. The *discrete Fourier heat law* in one direction is

(a). heat flows from hot to cold,

(b). the change in heat is proportional to the cross sectional area,

10. In flow2d.m experiment with different decay rates $dec = .01, .02$ and .04. Be sure to make any adjustments to the time step so that the stability condition holds.

11. Experiment with the wind velocity in the MATLAB code flow2d.m.

(a). Adjust the magnitudes of the velocity components and observe stability as a function of wind velocity.

(b). If the wind velocity is not from the south and west, then the finite difference model in (1.5.10) will change. Let the wind velocity be from the north and west, say wind velocity = (.2, -.4). Modify the finite difference model.

(c). Modify the MATLAB code flow2d.m to account for this change in wind direction.

12. Suppose pollutant is being generated at a rate of 3 units of heat per unit volume per unit time.

(a). How is the model for the 2D problem in equation (1.5.10) modified to account for this?

(b). Modify flow2d.m to implement this source of pollution.

(c). Experiment with different values for the heat source $f = 0, 1, 2, 3$.

1.6 Convergence Analysis

1.6.1 Introduction

Initial value problems have the form

$$u_t = f(t, u) \text{ and } u(0) = \text{given}. \tag{1.6.1}$$

The simplest cases can be solved by separation of variable, but in general they do not have to have closed form solutions. Therefore, one is forced to consider various approximation methods. In this section we study the simplest numerical method, the Euler finite difference method. We shall see that under appropriate assumptions the error made by this type of approximation is bounded by a constant times the step size.

1.6.2 Applied Area

Again consider a well stirred liquid such as a cup of coffee. Assume that the temperature is uniform with respect to space, but the temperature may be changing with respect to time. We wish to predict the temperature as a function of time given some initial observations about the temperature.

1.6.3 Model

The continuous model of Newton's empirical law of cooling states that the rate of change of the temperature is proportional to the difference in the surrounding temperature and the temperature of the liquid

$$u_t = c(u_{sur} - u). \tag{1.6.2}$$

If $c = 0$, then there is perfect insulation, and the liquid's temperature must remain at its initial value. For large c the liquid's temperature will rapidly approach the surrounding temperature. The closed form solution of this differential equation can be found by the separation of variables method and is, for u_{sur} equal a constant,

$$u(t) = u_{sur} + (u(0) - u_{sur})e^{-ct}. \tag{1.6.3}$$

If the c is not given, then it can be found from a second observation $u(t_1) = u_1$. If u_{sur} is a function of t, one can still find a closed form solution provided the integrations steps are not too complicated.

1.6.4 Method

Euler's method involves the approximation of u_t by the finite difference

$$(u^{k+1} - u^k)/h$$

where $h = T/K$, u^k is an approximation of $u(kh)$ and f is evaluated at (kh, u^k). If T is not finite, then h will be fixed and k may range over all of the positive integers. The differential equation (1.6.1) can be replaced by either

$$\begin{aligned} (u^{k+1} - u^k)/h &= f((k+1)h, u^{k+1}) \\ \text{or, } (u^{k+1} - u^k)/h &= f(kh, u^k). \end{aligned} \tag{1.6.4}$$

The choice in (1.6.4) is the simplest because it does not require the solution of a possibly nonlinear problem at each time step. The scheme given by (1.6.4) is called *Euler's method,* and it is a discrete model of the differential equation in (1.6.2). For the continuous Newton's law of cooling differential equation where $f(t, u) = c(u_{sur} - u)$ Euler's method is the same as the first order finite difference method for the discrete Newton's law of cooling.

The *improved Euler method* is given by the following two equations

$$(utemp - u^k)/h = f(kh, u^k) \tag{1.6.5}$$

$$(u^{k+1} - u^k)/h = 1/2(f(kh, u^k) + f((k+1)h, utemp)). \tag{1.6.6}$$

Equation (1.6.5) gives a first estimate of the temperature at time kh, and then it is used in equation (1.6.6) where an average of the time derivative is computed. This is called improved because the errors for Euler's method are often bounded by a constant times the time step, while the errors for the improved Euler method are often bounded by a constant times the time step *squared.*

1.6.5 Implementation

One can easily use MATLAB to illustrate that as the time step decreases the solution from the discrete models approaches the solution to the continuous model. This is depicted in both graphical and table form. In the MATLAB

code eulerr.m we experiment with the number of time steps and fixed final time. Newton's law of cooling for a constant surrounding temperature is considered so that the exact solution is known. The exact solution is compared with both the Euler and improved Euler approximation solutions.

In the MATLAB code eulerr.m lines 1-13 contain the input data. The arrays for the exact solution, Euler approximate solution and the improved Euler approximate solution are, respectively, uexact, ueul and uieul, and they are computed in time loop in lines 14-25. The output is given in lines 26-29 where the errors are given for the final time.

MATLAB Code eulerr.m

```
1.  % This code compares the discretization errors.
2.  % The Euler and improved Euler methods are used.
3.  clear;
4.  maxk = 5;       % number of time steps
5.  T = 10.0;       % final time
6.  dt = T/maxk;
7.  time(1) = 0;
8.  u0 = 200.;      % initial temperature
9.  c = 2./13.;     % insulation factor
10.  usur = 70.;     % surrounding temperature
11.  uexact(1) = u0;
12.  ueul(1) = u0;
13.  uieul(1) = u0;
14.  for k = 1:maxk      %time loop
15.      time(k+1) = k*dt;
16.      % exact solution
17.      uexact(k+1) = usur + (u0 - usur)*exp(-c*k*dt);
18.      % Euler numerical approximation
19.      ueul(k+1) = ueul(k) +dt*c*(usur - ueul(k));
20.      % improved Euler numerical approximation
21.      utemp = uieul(k) +dt*c*(usur - uieul(k));
22.      uieul(k+1)= uieul(k)
                    + dt/2*(c*(usur - uieul(k))+c*(usur - utemp));
23.      err_eul(k+1) = abs(ueul(k+1) - uexact(k+1));
24.      err_im_eul(k+1) = abs(uieul(k+1) - uexact(k+1));
25.  end
26.  plot(time, ueul)
27.  maxk
28.  err_eul_at_T = err_eul(maxk+1)
29.  err_im_eul_at_T = err_im_eul(maxk+1)
```

Figure 1.6.1 contains the plots of for the Euler method given in the arrays ueul for $maxk = 5, 10, 20$ and 40 times steps. The curve for $maxk = 5$ is not realistic because of the oscillations, but it does approach the surrounding

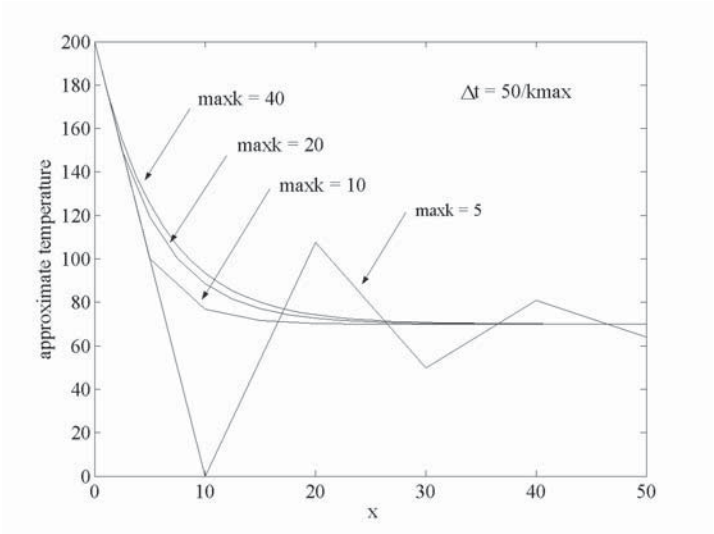

Figure 1.6.1: Euler Approximations

Table 1.6.1: Euler Errors at t = 10

Time Steps	**Euler Error**	**Improved Euler Error**
5	7.2378	0.8655
10	3.4536	0.1908
20	1.6883	0.0449
40	0.8349	0.0109

temperature. The other three plots for all points in time increase towards the exact solution.

Another way of examining the error is to fix a time and consider the difference in the exact temperature and the approximate temperatures given by the Euler methods. Table 1.6.1 does this for time equal to 10. The Euler errors are cut in half whenever the number of time steps are doubled, that is, the Euler errors are bounded be a constant times the time step size. The improved Euler errors are cut in one quarter when the number of time steps are doubled, that is, the improved Euler errors are bounded by a constant times the time step size *squared*.

1.6.6 Assessment

In order to give an explanation of the discretization error, we must review the mean value theorem and an extension. The mean value theorem, like the

intermediate value theorem, appears to be clearly true once one draws the picture associated with it. Drawing the a picture does make some assumptions. For example, consider the function given by $f(x) = 1 - |x|$. Here there is a "corner" in the graph at $x = 0$, that is, $f(x)$ does not have a derivative at $x = 0$.

Theorem 1.6.1 *(Mean Value Theorem) Let $f : [a, b] \to \mathbb{R}$ be continuous on $[a, b]$. If f has a derivative at each point of (a, b), then there is a c between a and x such that $f'(c) = (f(b) - f(a))/(b - a)$.*

If b is replaced by x and we solve for $f(x)$ in $f'(c) = (f(x) - f(a))/(x - a)$, then provided $f(x)$ has a derivative

$$f(x) = f(a) + f'(c)(x - a)$$

for some c between a and x. Generally, one does not know the exact value of c, but if the derivative is bounded by M, then the following inequality holds

$$|f(x) - f(a)| \leq M |x - a| .$$

An extension of this linear approximation to a quadratic approximation of $f(x)$ is stated in the next theorem.

Theorem 1.6.2 *(Extended Mean Value Theorem) If $f : [a, b] \to \mathbb{R}$ has two continuous derivatives on $[a, b]$, then there is a c between a and x such that*

$$f(x) = f(a) + f'(a)(x - a) + f''(c)(x - a)^2/2. \tag{1.6.7}$$

Clearly Euler's method is a very inexpensive algorithm to execute. However, there are sizable errors. There are two types of errors:

$$\textit{Discretization error} \equiv E_d^k = u^k - u(kh)$$

where u^k is from Euler's algorithm (1.6.4) with no roundoff error and $u(kh)$ is from the exact continuum solution (1.6.1).

$$\textit{Accumulation error} \equiv E_r^k = U^k - u^k$$

where U^k is from Euler's algorithm, but with round errors.
The *overall error* contains both errors and is $E_r^k + E_d^k = U^k - u(kh)$. In Table 1.6.1 the discretization error for Euler's method is bounded by a constant times h, and the discretization error for the improved Euler method is bounded by an constant times h *squared*.

Now we will give the discretization error analysis for the Euler method applied to equation (1.6.2). The relevant terms for the error analysis are

$$u_t(kh) = c(u_{sur} - u(kh)) \tag{1.6.8}$$
$$u^{k+1} = u^k + hc(u_{sur} - u^k) \tag{1.6.9}$$

Use the extended mean value theorem on $u(kh+h)$ where a is replaced by kh and x is replaced by $kh+h$

$$u((k+1)h) = u(kh) + u_t(kh)h + u_{tt}(c_{k+1})h^2/2 \tag{1.6.10}$$

Use the right side of (1.6.8) for $u_t(kh)$ in (1.6.10), and combine this with (1.6.9) to get

$$\begin{aligned} E_d^{k+1} &= u^{k+1} - u((k+1)h) \\ &= [u^k + hc(u_{sur} - u^k)] - \\ &\quad [u(kh) + c(u_{sur} - u(kh))h + u_{tt}(c_{k+1})h^2/2] \\ &= aE_d^k + b_{k+1}h^2/2 \qquad (1.6.11) \\ \text{where } a &= 1 - ch \text{ and } b_{k+1} = -u_{tt}(c_{k+1}). \end{aligned}$$

Suppose $a = 1 - ch > 0$ and $|b_{k+1}| \leq M$. Use the triangle inequality, a "telescoping" argument and the partial sums of the geometric series $1 + a + a^2 + \cdots + a^k = (a^{k+1} - 1)/(a-1)$ to get

$$\begin{aligned} |E_d^{k+1}| &\leq a|E_d^k| + Mh^2/2 \\ &\leq a(a|E_d^{k-1}| + Mh^2/2) + Mh^2/2 \\ &\quad \vdots \\ &\leq a^{k+1|}|E_d^0| + (a^{k+1} - 1)/(a-1)\ Mh^2/2. \qquad (1.6.12) \end{aligned}$$

Assume $E_d^0 = 0$ and use the fact that $a = 1 - ch$ with $h = T/K$ to obtain

$$\begin{aligned} |E_d^{k+1}| &\leq [(1 - cT/K)^K - 1]/(-ch)\ Mh^2/2 \\ &\leq 1/c\ Mh/2. \qquad (1.6.13) \end{aligned}$$

We have proved the following theorem, which is a special case of a more general theorem about Euler's method applied to ordinary differential equations of the form (1.6.1), see [3, chapter 5.2]

Theorem 1.6.3 *(Euler Error Theorem) Consider the continuous (1.6.2) and discrete (1.6.4) Newton cooling models. Let T be finite, $h = T/K$ and let solution of (1.6.2) have two derivatives on the time interval $[0,T]$. If the second derivative of the solution is bounded by M, and the initial condition has no roundoff error and $1 - ch > 0$, then the discretization error is bounded by $(M/2c)h$.*

In the previous sections we consider discrete models for heat and pollutant transfer

$$\begin{aligned} \text{Pollutant Transfer} &: \quad u_t = f - au_x - cu, \qquad (1.6.14) \\ &\qquad u(0,t) \text{ and } u(x,0) \text{ given.} \\ \text{Heat Diffusion} &: \quad u_t = f + (\kappa u_x)_x - cu, \qquad (1.6.15) \\ &\qquad u(0,t), u(L,t) \text{ and } u(x,0) \text{ given.} \end{aligned}$$

Table 1.6.2: Errors for Flow

Δt	**Δx**	**Flow Errors in (1.6.14)**
1/10	1/20	0.2148
1/20	1/40	0.1225
1/40	1/60	0.0658
1/80	1/80	0.0342

Table 1.6.3: Errors for Heat

Δt	**Δx**	**Heat Errors in (1.6.15)**
1/50	1/5	$9.2079\ 10^{-4}$
1/200	1/10	$2.6082\ 10^{-4}$
1/800	1/20	$0.6630\ 10^{-4}$
1/3200	1/40	$0.1664\ 10^{-4}$

The *discretization errors* for (1.6.14) and (1.6.15), where the solutions depend both on space and time, have the form

$$\begin{aligned} E_i^{k+1} &\equiv u_i^{k+1} - u(i\Delta x, (k+1)\Delta t) \\ \left\|E^{k+1}\right\| &\equiv \max_i \left|E_i^{k+1}\right|. \end{aligned}$$

$u(i\Delta x, (k+1)\Delta t)$ is the exact solution, and u_i^{k+1} is the numerical or approximate solution. In the following examples the discrete models were from the explicit finite difference methods used in Chapter 1.3 and 1.4.

Example for (1.6.14). Consider the MATLAB code flow1d.m (see flow1derr.m and equations (1.4.2-1.4.4)) that generates the numerical solution of (1.6.14) with $c = dec = .1$, $a = vel = .1$, $f = 0$, $u(0,t) = sin(2\pi(0 - vel\ t))$ and $u(x,0) = sin(2\pi x)$. It is compared over the time interval $t = 0$ to $t = T = 20$ and at $x = L = 1$ with the exact solution $u(x,t) = e^{-dec\ t}\ sin(2\pi(x - vel\ t))$. Note the error in Table 1.6.2 is proportional to $\Delta t + \Delta x$.

Example for (1.6.15). Consider the MATLAB code heat.m (see heaterr.m and equations (1.2.1)-1.2.3)) that computes the numerical solution of (1.6.15) with $k = 1/\pi^2$, $c = 0$, $f = 0$, $u(0,t) = 0$, $u(1,t) = 0$ and $u(x,0) = sin(\pi x)$. It is compared at $(x,t) = (1/2, 1)$ with the exact solution $u(x,t) = e^{-t}sin(\pi x)$. Here the error in Table 1.6.3 is proportional to $\Delta t + \Delta x^2$.

In order to give an explanation of the discretization errors, one must use higher order *Taylor polynomial approximation*. The proof of this is similar

to the extended mean value theorem. It asserts if $f : [a, b] \to \mathbb{R}$ has $n + 1$ continuous derivatives on $[a, b]$, then there is a c between a and x such that

$$\begin{aligned} f(x) &= f(a) + f^{(1)}(a)(x-a) + \cdots + f^{(n)}(a)/n!\ (x-a)^n \\ &\quad + f^{(n+1)}(c)/(n+1)!\ (x-a)^{n+1}. \end{aligned}$$

Theorem 1.6.4 *(Discretization Error for (1.6.14)) Consider the continuous model (1.6.14) and its explicit finite difference model. If a, c and $(1 - a\Delta t/\Delta x - \Delta t\ c)$ are nonnegative, and u_{tt} and u_{xx} are bounded on $[0, L] \times [0, T]$, then there are constants C_1 and C_2 such that*

$$\left\| E^{k+1} \right\| \leq (C_1 \Delta x + C_2 \Delta t)T\ .$$

Theorem 1.6.5 *(Discretization Error for (1.6.15)) Consider the continuous model (1.6.15) and its explicit finite difference model. If $c > 0, \kappa > 0, \alpha = (\Delta t/\Delta x^2)\kappa$ and $(1 - 2\alpha - \Delta t\ c) > 0$, and u_{tt} and u_{xxxx} are bounded on $[0, L] \times [0, T]$, then there are constants C_1 and C_2 such that*

$$\left\| E^{k+1} \right\| \leq (C_1 \Delta x^2 + C_2 \Delta t)T.$$

1.6.7 Exercises

1. Duplicate the calculations in Figure 1.6.1, and find the graphical solution when $maxk = 80$.
2. Verify the calculations in Table 1.6.1, and find the error when $maxk = 80$.
3. Assume the surrounding temperature initially is 70 and increases at a constant rate of one degree every ten minutes.
   (a). Modify the continuous model in (1.6.2) and find its solution via the MATLAB command desolve.
   (b). Modify the discrete model in (1.6.4).
4. Consider the time dependent surrounding temperature in problem 3.
   (a). Modify the MATLAB code eulerr.m to account for the changing surrounding temperature.
   (b). Experiment with different number of time steps with $maxk = 5$, 10, 20, 40 and 80.
5. In the proof of the Theorem 1.6.3 justify the (1.6.11) and $|b_{k+1}| \leq M$.
6. In the proof of the Theorem 1.6.3 justify the (1.6.12) and (1.6.13).
7. Modify Theorem 1.6.3 to account for the case where the surrounding temperature can depend on time, $u_{sur} = u_{sur}(t)$. What assumptions should be placed on $u_{sur}(t)$ so that the discretization error will be bounded by a constant times the step size?
8. Verify the computations in Table 1.6.14. Modify flow1d.m by inserting an additional line inside the time-space loops for the error (see flow1derr.m).
9. Verify the computations in Table 1.6.15. Modify heat.m by inserting an additional line inside the time-space loops for the error (see heaterr.m).
10. Consider a combined model for (1.6.14)-(1.6.15): $u_t = f + (\kappa u_x)_x - a u_x - c u$. Formulate suitable boundary conditions, an explicit finite difference method, a MATLAB code and prove an error estimate.