1.13 Predictor-corrector methods

The trapezoidal rule differs from the other two that we've looked at in that it does not explicitly tell us what the next value of the unknown function is, but instead gives us an equation that must be solved in order to find it. At first sight this seems like a nuisance, but in fact it is a boon, because it enables us to regulate the step size during the courses of a calculation, as we will discuss in section 1.15.

Let's take a look at the process by which we refine a guessed value of y_{n+1} to an improved value, using the trapezoidal formula

$$y_{n+1} = y_n + \frac{h}{2}(f(x_n, y_n) + f(x_{n+1}, y_{n+1})). \tag{1.13.1}$$

Suppose we let $y_{n+1}^{(k)}$ represent some guess to the value of y_{n+1} that satisfies (1.13.1). Then the improved value $y_{n+1}^{(k+1)}$ is computed from

$$y_{n+1}^{(k+1)} = y_n + \frac{h}{2}(f(x_n, y_n) + f(x_{n+1}, y_{n+1}^{(k)})). \tag{1.13.2}$$

We want to find out about how rapidly the successive values $y_{n+1}^{(k)}$, $k = 1, 2, \ldots$ approach a limit, if at all. To do this, we rewrite equation (1.13.2), this time replacing k by $k-1$ to obtain

$$y_{n+1}^{(k)} = y_n + \frac{h}{2}(f(x_n, y_n) + f(x_{n+1}, y_{n+1}^{(k-1)})) \tag{1.13.3}$$

and then subtract (1.13.3) from (1.13.2) to get

$$y_{n+1}^{(k+1)} - y_{n+1}^{(k)} = \frac{h}{2}(f(x_{n+1}, y_{n+1}^{(k)}) - f(x_{n+1}, y_{n+1}^{(k-1)})). \tag{1.13.4}$$

Next we use the mean-value theorem on the difference of f values on the right-hand side, yielding

$$y_{n+1}^{(k+1)} - y_{n+1}^{(k)} = \frac{h}{2}\left.\frac{\partial f}{\partial y}\right|_{(x_{n+1},\eta)} \left(y_{n+1}^{(k)} - y_{n+1}^{(k-1)}\right), \tag{1.13.5}$$

where η lies between $y_{n+1}^{(k)}$ and $y_{n+1}^{(k-1)}$.

From the above we see at once that the difference between two consecutive iterated values of y_{n+1} will be $\frac{h}{2}\frac{\partial f}{\partial y}$ times the difference between the previous two iterated values.

It follws that the iterative process will converge if h is kept small enough so that $\frac{h}{2}\frac{\partial f}{\partial y}$ is less than 1 in absolute value. We refer to $\frac{h}{2}\frac{\partial f}{\partial y}$ as the *local convergence factor* of the trapezoidal rule.

If the factor is a lot less than 1 (and this can be assured by keeping h small enough), then the convergence will be extremely rapid.

In actual practice, one uses an iterative formula together with another formula (the predictor) whose mission is to provide an intelligent first guess for the iterative method to use. The predictor formula will be explicit, or non-iterative. If the predictor formula is clever enough, then it will happen that just a single application of the iterative refinement (corrector formula) will be sufficient, and we won't have to get involved in a long convergence process.

If we use the trapezoidal rule for a corrector, for instance, then a clever predictor would be the midpoint rule. The reason for this will become clear if we look at both formulas together with their error terms. We will see in the next section that the error terms are as follows:

$$y_{n+1} = y_{n-1} + 2hy'_n + \frac{h^3}{3}y'''(X_m) \tag{1.13.6}$$

$$y_{n+1} = y_n + \frac{h}{2}(y'_n + y'_{n+1}) - \frac{h^3}{12}y'''(X_t). \tag{1.13.7}$$

Now the exact locations of the points X_m and X_t are unknown, but we will assume here that h is small enough that we can regard the two values of y''' that appear as being the same.

As far as the powers of h that appear in the error terms go, we see that the third power occurs in both formulas. We say then, that the midpoint predictor and the trapezoidal corrector constitute a *matched pair*. The error in the trapezoidal rule is about one fourth as large as, and of opposite sign from, the error in the midpoint method.

The midpont guess is therefore quite "intelligent". The subsequent iterative refinement of that guess needs to reduce the error only by a factor of four. Now let y_P denote the midpoint predicted value, $y^{(1)}_{n+1}$ denote the first refined value, and y_{n+1} be the final converged value given by the trapezoidal rule. Then we have

$$\begin{aligned} y_{n+1} &= y_n + \frac{h}{2}\left(y'_n + f(x_{n+1}, y_{n+1})\right) \\ y^{(1)}_{n+1} &= y_n + \frac{h}{2}\left(y'_n + f(x_{n+1}, y_P)\right) \end{aligned}$$

and by subtraction

$$y^{(1)}_{n+1} - y_{n+1} = \frac{h}{2}\frac{\partial f}{\partial y}\left(y_P - y_{n+1}\right).$$

This shows that, however far from the converged value the first guess was, the refined value is $\frac{h}{2}\frac{\partial f}{\partial y}$ times closer. Hence if we can keep $\frac{h}{2}\frac{\partial f}{\partial y}$ no bigger than about $1/4$, then the distance from the first refined value to the converged value will be no larger than the size of the error term in the method, and so there would be little point in gilding the iteration any further.

The conclusion is that when we are dealing with a matched predictor-corrector pair, we need do only a single refinement of the corrector if the step size is kept moderately small. Furthermore, "moderately small" means that the step size times the local value of $\frac{\partial f}{\partial y}$ should be small compared to 1. For this reason, iteration to full convergence is rarely done in practice.

1.14 Truncation error and step size

We have so far regarded the step size h as a silent partner, more often than not choosing it to be equal to 0.05, for no particular reason. It is evident, however, that the accuracy of the calculation is strongly affected by the step size. If h is chosen too large, the computed solution may be quite far from the true solution of the differential equation, if too small then the calculation will become unnecessarily time-consuming, and roundoff errors may build up excessively because of the numerous arithmetic operations that are being carried out.

Speaking in quite general terms, if the true solution of the differential equation is rapidly changing, then we will need a small values of h, that is, small compared to the local relaxation length (see section 1.12), and if the solution changes slowly, then a larger value of h will do.

Frequently in practice we deal with equations whose solutions change very rapidly over part of the range of integration and slowly over another part. Examples of this are provided by the study of the switching on of a complicated process, such as beginning a multi-stage chemical reaction, turning on a piece of electronic equipment, starting a power reactor, etc. In such cases there usually are rapid and ephemeral or "transient" phenomena that occur soon after startup, and that disappear quickly. If we want to follow these transients accurately, we may need to choose a very tiny step size. After the transients die out, however, the steady-state solution may be a very quiet, slowly varying or nearly constant function, and then a much larger value of h will be adequate.

If we are going to develop software that will be satisfactory for such problems, then the program will obviously have to choose, and re-choose its own step size as the calculation proceeds. While following a rapid transient it should use a small mesh size, then it should gradually increase h as the transient fades, use a large step while the solution is steady, decreae it again if further quick changes appear, and so forth, all without operator intervention.

Before we go ahead to discuss methods for achieving this step size control, let's observe that one technique is already available in the material of the previous section. Recall that if we want to, we can implement the trapezoidal rule by first guessing, or predicting, the unknown at the next point by using Euler's formula, and then correcting the guess to complete convergence by iteration.

The first guess will be relatively far away from the final converged value if the solution is rapidly varying, but if the solution is slowly varying, then the guess will be rather good. It follows that the number of iterations required to produce convergence is one measure of the appropriateness of the current value of the step size: if many iterations are needed, then the step size is too big. Hence one way to get some control on h is to follow a policy of cutting the step size in half whenever more than, say, one or two iterations are necessary.

This suggestion is not sufficiently sensitive to allow doubling the stepsize when only one iteration is needed, however, and somewhat more delicacy is called for in that situation. Furthermore this is a very time-consuming approach since it involves a complete iteration to convergence, when in fact a single turn of the crank is enough if the step size is kept small enough.

The discussion does, however, point to the fundamental idea that underlies the automatic control of step size during the integration. That basic idea is precisely that we can estimate the correctness of the step size by whatching how well the first guess in our iterative process agrees with the corrected value. The correction process itself, when viewed this way, is seen to be a powerful ally of the software user, rather than the "pain in the neck" it seemed to be when we first met it.

Indeed, why would anyone use the cumbersome procedure of guessing and refining (*i.e.*, prediction and correction) as we do in the trapezoidal rule, when many other methods are available that give the next value of the unknown immediately and explicitly? No doubt the question crossed the reader's mind, and the answer is now emerging. It will appear that not only does the disparity between the first prediction and the corrected value help us to control the step size, it actually can give us a quantitative estimate of the local error in the integration, so that if we want to, we can print out the approximate size of the error along with the solution.

Our next job will be to make these rather qualitative remarks into quantitative tools, and so we must discuss the estimation of the error that we commit by using a particular difference approximation to a differential equation, instead of that equation itself, on one step of the integration process. This is the *single-step truncation error*. It does not tell us how far our computed solution is from the true solution, but only how much error is committed in a single step.

The easiest example, as usual, is Euler's method. In fact, in equation (1.7.2) we have already seen the single-step error of this metnod. That equation was

$$y(x_n + h) = y(x_n) + hy'(x_n) + h^2 \frac{y''(X)}{2} \tag{1.14.1}$$

where X lies between x_n and $x_n + h$. In Euler's procedure, we drop the third term on the right, the "remainder term", and compute the solution from the rest of the equation. In doing this we commit a single-step trunction error that is equal to

$$E = h^2 \frac{y''(X)}{2} \qquad x_n < X < x_n + h. \tag{1.14.2}$$

Thus, Euler's method is exact ($E = 0$) if the solution is a polynomial of degree 1 or less ($y'' = 0$). Otherwise, the single-step error is proportional to h^2, so if we cut the step size in half, the local

error is reduced to 1/4 of its former value, approximately, and if we double h the error is multiplied by about 4.

We could use (1.14.2) to estimate the error by somehow computing an estimate of y''. For instance, we might differentiate the differential equation $y' = f(x, y)$ once more, and compute y'' directly from the resulting formula. This is usually more trouble than it is worth, though, and we will prefer to estimate E by more indirect methods.

Next we derive the local error of the trapezoidal rule. There are various special methods that might be used to do this, but instead we are going to use a very general method that is capable of dealing with the error terms of almost every integration rule that we intend to study.

First, let's look a little more carefully at the capability of the trapezoidal rule, in the form

$$y_{n+1} - y_n - \frac{h}{2}(y'_n + y'_{n+1}) = 0. \tag{1.14.3}$$

Of course, this is a recurrence by meand of which we propagate the approximate solution to the right. It certainly is not exactly true if y_n denotes the value of the true solution at the point x_n unless that true solution is very special. How special?

Suppose the true solution is $y(x) = 1$ for all x. Then (1.14.3) would be exactly valid. Suppose $y(x) = x$. Then (1.14.3) is again exactly satisfied, as the reader should check. Furthermore, if $y(x) = x^2$, then a brief calculation reveals that (1.14.3) holds once more. How long does this continue? Our run of good luck has just expired, because if $y(x) = x^3$ then (check this) the left side of (1.14.3) is not 0, but is instead $-h^3/2$.

We might say that the trapezoidal rule is exact on 1, x, and x^2, but not x^3, *i.e.*, that it is an integration rule of *order two* ("order" is an overworked word in differential equations). It follows by linearity that the rule is exact on any quadratic polynomial.

By way of contrast, it is easy to verify that Euler's method is exact for a linear function, but fails on x^2. Since the error term for Euler's method in (1.14.2) is of the form $\text{const} * h^2 * y''(X)$, it is perhaps reasonable to expect the error term for the trapezoidal rule to look like $\text{const} * h^3 * y'''(X)$.

Now we have to questions to handle, and they are respectively easy and hard:

(a) If the error term in the trapezoidal rule really is $\text{const} * h^3 * y'''(X)$, then what is "const"?

(b) Is it true that the error term is $\text{const} * h^3 * y'''(X)$?

We'll do the easy one first, anticipating that the answer to (b) is affirmative so the effort won't be wasted. If the error term is of the form stated, then the trapezoidal rule can be written as

$$y(x_h) - y(x) - \frac{h}{2}\left(y'(x+h) + y'(x)\right) = c * h^3 * y'''(X), \tag{1.14.4}$$

where X lies between x and $x + h$. To find c all we have to do is substitute $y(x) = x^3$ into (1.14.4) and we find at once that $c = -1/12$. The single-step truncation error of the trapezoidal rule would therefore be

$$E = -h^3 \frac{y'''(X)}{12} \qquad x < X < x + h. \tag{1.14.5}$$

Now let's see that question (b) has the answer "yes" so that (1.14.5) is really right.

To do this we start with a truncated Taylor series with the integral form of the remainder, rather than with the differential form of the remainder. In general the series is

$$y(x) = y(0) + xy'(0) + x^2 \frac{y''(0)}{2!} + \cdots + x^n \frac{y^{(n)}(0)}{n!} + R_n(x) \tag{1.14.6}$$

where

$$R_n(x) = \frac{1}{n!}\int_0^x (x-s)^n y^{(n-1)}(s)\,ds. \tag{1.14.7}$$

Indeed, one of the nice ways to prove Taylor's theorem begins with the right-hand side of (1.14.7), plucked from the blue, and then repeatedly integrates by parts, lowereing the order of the derivative of y and the power of $(x-s)$ until both reach zero.

In (1.14.6) we choose $n = 2$, because the trapezoidal rule is exact on polynomials of degree 2, and we write it in the form

$$y(x) = P_2(x) + R_2(x) \tag{1.14.8}$$

where $P_2(x)$ is the quadratic (in x) polynomial $P_2(x) = y(0) + xy'(0) + x^2y''(0)/2$.

Next we define a certain operation that transforms a function $y(x)$ into a new function, namely into the left-hand side of equation (1.14.4). We call the operator L and so it is defined by

$$\mathrm{L}y(x) = y(x+h) - y(x) - \frac{h}{2}\left(y'(x) + y'(x+h)\right). \tag{1.14.9}$$

Now we apply the operator L to both sides of equation (1.14.8), and we notice immediately that $\mathrm{L}P_2(x) = 0$, because the rule is exact on quadratic polynomials (this is why we chose $n = 2$ in (1.14.6)). Hence we have

$$\mathrm{L}y(x) = \mathrm{L}R_2(x). \tag{1.14.10}$$

Notice that we have here a remainder formula for the trapezoidal rule. It isn't in a very satisfactory form yet, so we will now make it a bit more explicit by computing $\mathrm{L}R_2(x)$. First, in the integral expression (1.14.7) for $R_2(x)$ we want to replace the upper limit of the integral by $+\infty$. We can do this by writing

$$R_2(x) = \frac{1}{2!}\int_0^\infty H_2(x-s)y'''(s)\,ds \tag{1.14.11}$$

where $H_2(t) = t^2$ if $t > 0$ and $H_2(t) = 0$ if $t < 0$.

Now if we bear in mind the fact that the operator L acts only on x, and that s is a dummy variable of integration, we find that

$$\mathrm{L}R_2(x) = \frac{1}{2!}\int_0^\infty \mathrm{L}H_2(x-s)y'''(s)\,ds. \tag{1.14.12}$$

Choose $x = h$. Then if s lies between 0 and h we find

$$\begin{aligned}\mathrm{L}H_2(x-s) &= (h-s)^2 - \frac{h}{2}\left(2(h-s)\right)\\ &= -s(h-s)\end{aligned}$$

(*Caution:* Do not read past the right-hand side of the first equals sign unless you can verify the correctness of what you see there!), whereas if $s > h$ then $\mathrm{L}H_2(x-s) = 0$.

Then (1.14.12) becomes

$$\mathrm{L}R_2(h) = -\frac{1}{2}\int_0^h s(h-s)y'''(s)\,ds. \tag{1.14.13}$$

This is a much better form for the remainder, but we still do not have the "hard" question (b). To finish it off we need a form of the mean-value theorem of integral calculus, namely

Theorem 1.14.1: *If $p(x)$ is non-negative, and $g(x)$ is continuous, then*

$$\int_a^b p(x)g(x)\,dx = g(X)\int_a^b p(x)\,dx \tag{1.14.14}$$

where X lies between a and b.

The theorem asserts that a weighted average of the values of a continuous function is itself one of the values of that function. The vital hypothesis is that the "weight" $p(x)$ does not change sign.

Now in (1.14.13), the function $s(h-s)$ does not change sign on the s-interval $(0,h)$, and so

$$\mathrm{L}R_2(h) = -\frac{1}{2}y'''(X)\int_0^h s(h-s)\,ds \tag{1.14.15}$$

and if we do the integral we obtain, finally,

Theorem 1.14.2: *The trapezoidal rule with remainder term is given by the formula*

$$y(x_{n+1}) - y(x_n) = \frac{h}{2}\left(y'(x_n) + y'(x_{n+1})\right) - \frac{h^3}{12}y'''(X), \tag{1.14.16}$$

where X lies between x_n and x_{n+1}.

The proof of this theorem involved some ideas tha carry over almost unchanged to very general kinds of integration rules. Therefore it is important to make sure that you completely understand its derivation.

1.15 Controlling the step size

In equation (1.14.5) we saw that if we can estimate the size of the third derivative during the calculation, then we can estimate the error in the trapezoidal rule as we go along, and modify the step size h if necessary, to keep that error within preassigned bounds.

To see how this can be done, we will quote, without proof, the result of a similar derivation for the midpoint rule. It says that

$$y(x_{n+1}) - y(x_{n-1}) = 2hy'(x_n) + \frac{h^3}{3}y'''(X), \tag{1.15.1}$$

where X is between x_{n-1} and x_{n+1}. Thus the midpoint rule is also of second order. If the step size were halved, the local error would be cut to one eighth of its former value. The error in the midpoint rule is, however, about four times as large as that in the trapezoidal formula, and of opposite sign.

Now suppose we adopt an overall strategy of predicting the value y_{n+1} of the unknown by means of the midpoint rule, and then refining the prediction to convergence with the trapezoidal corrector. We want to estiamte the size of the single-step truncation error, using only the following data, both of which are available during the calculation: (a) the initial guess, from the midpoint method, and (b) the converged corrected value, from the trapezoidal rule.

We begin by defining three different kinds of values of the unknown function at the "next" point x_{n+1}. They are

(i) the quantity p_{n+1} is defined as the predicted value of $y(x_{n+1})$ obtained from using the midpoint rule, except that backwards values are not the computed ones, but are the exact ones instead. In symbols,

$$p_{n+1} = y(x_{n+1}) + 2hy'(x_n). \tag{1.15.2}$$

Of course, p_{n+1} is not available during an actual computation.

(ii) the quantity q_{n+1} is the value that we would compute from the trapezoidal corrector if for the backward value we use the exact solution $y(x_n)$ instead of the calculated solution y_n. Thus q_{n+1} satisfies the equation

$$q_{n+1} = y(x_n) + \frac{h}{2}\left(f(x_n, y(x_n)) + f(x_{n+1}, q_{n+1})\right). \tag{1.15.3}$$

Again, q_{n+1} is not available to us during calculation.

(iii) the quantity $y(x_{n+1})$, which is the exact solution itself. It staisfies two different equations, one of which is

$$y(x_{n+1}) = y(x_n) + \frac{h}{2}\left(f(x_n, y(x_n)) + f(x_{n+1}, y(x_{n+1}))\right) - \frac{h^3}{12}y'''(X) \tag{1.15.4}$$

and the other of which is (1.15.1). Note that the two X's may be different.

Now, from (1.15.1) and (1.15.2) we have at once that

$$y(x_{n+1}) = p_{n+1} + \frac{h^3}{3}y'''(X). \tag{1.15.5}$$

Next, from (1.15.3) and (1.15.4) we get

$$\begin{aligned} y(x_{n+1}) &= \frac{h}{2}\left(f(x_{n+1}, y(x_{n+1})) - f(x_{n+1}, q_{n+1})\right) - \frac{h^3}{12}y'''(X) \\ &= q_{n+1}\frac{h}{2}(y(x_{n+1}) - q_{n+1})\frac{\partial f}{\partial y}(x_{n+1}, Y) - \frac{h^3}{12}y'''(X) \end{aligned}$$

where we have used the mean-value theorem, and Y lies between $y(x_{n+1})$ and q_{n+1}. Now if we subtract q_{n+1} from both sides, we will observe that $y(x_{n+1}) - q_{n+1}$ will then appear on both sides of the equation. Hence we will be able to solve for it, with the result that

$$y(x_{n+1}) = q_{n+1} - \frac{h^3}{12}y'''(X) + \text{ terms involving } h^4. \tag{1.15.6}$$

Now let's make the working hypothesis that y''' is constant over the range of values of x considered, namely from $x_n - h$ to $x_n + h$. The $y'''(X)$ in (1.15.6) is thereby decreed to be equal to the $y'''(X)$ in (1.15.5), even though the X's are different. Under this assumption, we can eliminate $y(x_{n+1})$ between (1.15.5) and (1.15.6) and obtain

$$q_{n+1} - p_{n+1} = \frac{5}{12}h^3y''' + \text{ terms involving } h^4. \tag{1.15.7}$$

We see that this expresses the unknown, but assumed constant, value of y''' in terms of the difference between the initial prediction and the final converged value of $y(x_{n+1})$. Now we ignore the "terms involving h^4" in (1.15.7), solve for y''', and then for the estimated single-step truncation error we have

$$\begin{aligned} \text{Error} &= -\frac{h^3}{12}y''' \\ &\approx -\frac{1}{12}\frac{12}{5}\left(q_{n+1} - p_{n+1}\right) \\ &= -\frac{1}{5}(q_{n+1} - p_{n+1}). \end{aligned} \tag{1.15.8}$$

The quantity $q_{n+1} - p_{n+1}$ is not available during the calculation, but as an estimator we can use the compted predicted value and the compted converged value, because these differ only in that they use computed, rather than exact backwards values of the unknown function.

Hence, we have here an estimate of the single-step trunction error that we can conveniently compute, print out, or use to control the step size.

The derivation of this formula was of course dependent on the fact that we used the midpoint metnod for the predoctor and the trapezoidal rule for the corrector. If we had used a different pair, however, the same argument would have worked, provided only that the error terms of the predictor

and corrector formulas both involved the same derivative of y, *i.e.*, both formulas were of the same order.

Hence, "matched pairs"" of predictor and corrector formulas, *i.e.*, pairs in which both are of the same order, are most useful for carrying out extended calculations in which the local errors are continually monitored and the step size is regulated accordingly.

Let's pause to see how this error estimator would have turned out in the case of a general matched pair of predictor-corrector formulas, insted of just for the midpoint and trapezoidal rule combination. Suppose the predictor formula has an error term

$$y_{exact} - y_{predicted} = \lambda h^q y^{(q)}(X) \tag{1.15.9}$$

and suppose that the error in the corrector formula is given by

$$y_{exact} - y_{corrected} = \mu h^q y^{(q)}(X). \tag{1.15.10}$$

Then a derivation similar to the one that we have just done will show that the estimator for the single-step error that is available during the progress of the computation is

$$\text{Error} \approx \frac{\mu}{\lambda - \mu}(y_{predicted} - y_{corrected}). \tag{1.15.11}$$

In the table below we show the result of integrating the differential equation $y' = -y$ with $y(0) = 1$ using the midpoint and trapezoidal formulas with $h = 0.05$ as the predictor and corrector, as described above. The successive columns show x, the predicted value at x, the converged corrected value at x, the single-step error estimated from the approximation (1.15.8), and the actual single-step error obtained by computing

$$y(x_{n+1}) - y(x_n) - \frac{h}{2}(y'(x_n) + y'(x_{n+1}))$$

using the true solution $y(x) = e^{-x}$. The calculation was started by (cheating and) using the exact solution at 0.00 and 0.05.

x	Pred(x)	Corr(x)	Errest(x)	Error(x)
0.00	———-	———-	———-	———-
0.05	———-	———-	———-	———-
0.10	0.904877	0.904828	98×10^{-7}	94×10^{-7}
0.15	0.860747	0.860690	113×10^{-7}	85×10^{-7}
0.20	0.818759	0.818705	108×10^{-7}	77×10^{-7}
0.25	0.778820	0.778768	102×10^{-7}	69×10^{-7}
0.30	0.740828	0.740780	97×10^{-7}	61×10^{-7}
0.35	0.704690	0.704644	93×10^{-7}	55×10^{-7}
0.40	0.670315	0.670271	88×10^{-7}	48×10^{-7}
0.45	0.637617	0.637575	84×10^{-7}	43×10^{-7}
0.50	0.606514	0.606474	80×10^{-7}	37×10^{-7}
⋮	⋮	⋮	⋮	⋮
0.95	0.386694	0.386669	51×10^{-7}	5×10^{-7}
1.00	0.367831	0.367807	48×10^{-7}	3×10^{-7}

TABLE 1.15.1

Now that we have a simple device for estimating the single-step truncation error, namely by using one fifth of the distance between the first guess and the corrected value, we can regulate the step size so as to keep the error between preset limits. Suppose we would like to keep the single-step

error in the neighborhood of 10^{-8}. We might then adopt, say 5×10^{-8} as the upper limit of tolerable error and, for instance, 10^{-9} as the lower limit.

Why should we have a lower limit? If the calculation is being done with more precision than necessary, the step size will be smaller than needed, and we will be wasting computer time as well as possibly building up roundoff error.

Now that we have fixed these limits we should be under no delusion that our computed solution will depart from the true solution by no more than 5×10^{-8}, or whatever. What we are controlling is the one-step truncation error, a lot bettern than nothing, but certainly not the same as the total accumulated error.

With the upper and lower tolerances set, we embark on the computation. First, since the midpoint method needs two backward values before it can be used, something special will have to be done to get the procedure moving at the start. This is typical of numerical solution of differential equations. Special procedures are needed at the beginning to build up enough computed values so that the predictor-corrector formulas can be used thereafter, or at least until the step size is changed.

In the present case, since we're going to use the trapezoidal corrector anyway, we might as well use the trapezoidal rule, unassisted by midpoint, with complete iteration to convergence, to get the value of y at the first point $x_0 + h$ beyond the initial point x_0.

Now we have two consecutive values of y, and the generic calculation can begin. From any two consecutive values, the midpoint rule is used to predict the next value of y. This predicted value is also saved for future use in error estimation. The predicted value is then refined by the trapezoidal rule.

With the trapezoidal value in hand, the local error is then estimated by calculating one-fifth of the difference between that value and the midpoint guess.

If the absolute value of the local error lies between the preset limits 10^{-9} and 5×10^{-8}, then we just go on to the next step. This means that we augment x by h, and move back the newer values of the unknown function to the locations that hold older values (we remember, at any moment, just two past values).

Otherwise, suppose the local error was too large. Then we must reduce the step size h, say by cutting it in half. When all this is done, some message should be printed out that announces the change, and then we should restart the procedure, with the new value of h, from the "farthest backward" value of x for which we still have the corresponding y in memory. One reason for this is that we may find out right at the outset that our very first choice of h is too large, and perhaps it may need to be halved, say, three times before the errors are tolerable. Then we would like to restart each time from the same originally given data point x_0, rather than let the computation creep forward a little bit with step sizes that are too large for comfort.

Finally, suppose the local error was too small. Then we double the step size, print a message to that effect, and restart, again from the smallest possible value of x.

Now let's apply the philosophy of structured programming to see how the whole thing should be organized. We ask first for the major logical blocks into which the computation is divided. In this case we see

(i) a procedure `midpt`. Input to this procedure will be x, h, y_{n-1}, y_n. Output from it will be y_{n+1} computed from the midpoint formula. No arrays are involved. The three values of y in question occupy just three memory locations. The leading statement in this routine might be

```
midpt:=proc(x,h,y0,ym1,n);
```

and its `return` statement would be `return(yp1);`. One might think that it is scarcely necessary to have a separate subroutine for such a simple calculation. The spirit of structured

programming dictates otherwise. Someday one might want to change from the midpoint predictor to some other predictor. If organized as a subrouting, then it's quite easy to disentangle it from the program and replace it. This is the "modular" arpproach.

(ii) a procedure `trapez`. This routine will be called from two or three different places in the main routine: when starting, when restarting with a new value of h, and in a generic step, wher eit is the corrector formula. Operation of thie routine is different, too, depending on the circumstances. When starting or restarting, there is no externally supplied guess to help it. It must find its own way to convergence. On a generic step of the integration, however, we want it to use the prediction supplied by `midpt`, and then just do a single correction.

One way to handle this is to use a logical variable `start`. If `trapez` is called with `start = TRUE`, then the subroutine would supply a final converged value without looking for any input guess. Suppose the first line of `trapez` is

```
trapez:=proc(x,h,yin,yguess,n,start,eps);
```

and its `return` statement is `return(yout);`. When called with `start = TRUE`, then the routine might use `yin` as its first guess to *yout*, and iterate to convergence from there, using `eps` as its convergence criterion. If `start = FALSE`, it will take `yguess` as an estimate of `yout`, then use the trapezoidal rule just once, to refine the value as `yout`.

The combination of these two modules plus a small main program that calls them as needed, constitutes the whole package. Each of the subroutines and the main routine should be heavily documented in a self-contained way. That is, the descriptions of `trapez`, and of `midpt`, should precisely explain their operation as separate modules, with their own inputs and outputs, quite independently of the way they happen to be used by the main program in this problem. It should be possible to unplug a module from this application and use it without change in another. The documentation of a procedure should never make reference to how it is used by a certain calling program, but should describe only how it transforms its own inputs into its own outputs.

In the next section we are going to take an intermission from the study of integration rules in order to discuss an actual physical problem, the flight of a spacecraft to the moon. This problem will need the best methods that we can find for a successful, accurate solution. Then in section 1.17, we'll return to the modules discussed above, and will display complete computed programs that carry them out. In the meantine, the reader might enjoy trying to write such a complete program now, and comparing it with the one in that section.

1.16 Case study: Rocket to the moon

Now we have a reasonably powerful apparatus for integration of initial-value problems and systems, including the automatic regulation of step size and built-in error estimation. In order to try out this software on a problem that will use all of its capability, in this section we are going to derive the differential equations that govern the flight of a rocket to the moon. We do this first in a one-dimensional model, and then in two dimensions. It will be very useful to have these equations available for testing various proposed integration methods. Great accuracy will be needed, and the ability to chenge the step size, both to increase it and the devrease it, will be essential, or else the computation will become intolerably long. The variety of solutions that can be obtained is quite remarkable.

First, in the one-dimensional simplified model, we place the center of the earth at the origin of the x-axis, and let R denote the earth's radius. At the point $x = D$ we place the moon, and we let its radius be r. Finally, at a position $x = x(t)$ is our rocket, making its way towards the moon.

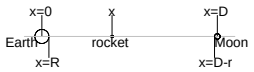

FIGURE 1.16.1

We will use Newton's law of gravitation to find the net gravitational force on the rocket,and equate it to the mass of the rocket times its acceleration (Newton's second law of motion). According to Newton's law of gravitation, the gravitational force exerted by one body on another is proportional to the product of their masses and inversely proportional to the square of the distance between them. If we use K for the constant of proportionality, then the force on the rocket due to the earth is

$$-K\frac{M_E m}{x^2},$$

whereas the force on the rocket due to the moon's gravity is

$$K\frac{M_M m}{(D-x)^2}$$

where M_E, M_M and m are, respectively, the masses of the earth, the moon and the rocket.

The acceleration of the rocket is of course $x''(t)$, and so the assertion that the net force is equal to mass times acceleration takes the form:

$$mx'' = -K\frac{M_E m}{x^2} + K\frac{M_M m}{(D-x)^2}. \tag{1.16.1}$$

This is a (nasty) differential equation of second order in the unknown function $x(t)$, the position of the rocket at time t. Note the nonlinear way in which this unknown function appears on the right-hand side.

A second-order differential equations deserves two initial values, and we will oblige. First. let's agree that at time $t = 0$ the rocket was onthe surfact of the earth, and second, that the rocket was fired at the moon with a certain initial velocity V. Hence, the initial conditions that go with (1.16.1) are

$$x(0) = R\,; \qquad x'(0) = V. \tag{1.16.2}$$

Now, just a quick glance at (1.16.1) shows that m cancels out, so let's remove it, but not before pointing out the immense significance of that fact. It implies that the motion of the rocket is independent of its mass. For performing a now-legendary experiment with rocks of different sizes dropping from the Tower of Pisa, Galileo demonstrated that fact to an incredulous world.

At any rate, (1.16.1) now reads as

$$x'' = -\frac{KM_E}{x} + \frac{KM_M}{(D-x)^2}. \tag{1.16.3}$$

We can make this equation a good bit prettier by changing the units of distance and time from miles and seconds (or whatever) to a set of more natural units for the problem.

For our unit of distance we choose R, the radius of the earth. If we divide (1.16.3) through by R, we can write the result as

$$\left(\frac{x}{R}\right)'' = -\frac{\frac{KM_E}{R^3}}{\left(\frac{X}{R}\right)^2} + \frac{\frac{KM_M}{R^3}}{\left(\frac{D}{R} - \frac{x}{R}\right)^2}. \tag{1.16.4}$$

Now instead of the unknown function $x(t)$, we define $y(t) = x(t)/R$. Then $y(t)$ is the position of the rocket, expressed in earth radii, at time t. Further, the ratio D/R that occurs in (1.16.4) is a

dimensionless quantity, whose numerical value is about 60. Hence (1.16.4) has now been transformed to

$$y'' = -\frac{\frac{KM_E}{R^3}}{y^2} + \frac{\frac{KM_M}{R^3}}{(60-y)^2}. \tag{1.16.5}$$

Next we tackle the new time units. Since y is now dimensionless, the dimension of the left side of the equation is the reciprocal of the square of a time. If we look next at the first term on the right, which of course has the same dimension, we see that the quantity R^3/KM_E is the square of a time, and so

$$T_0 = \sqrt{\frac{R^3}{KM_E}} \tag{1.16.6}$$

is a time. Its numerical value is easier to calculate if we change the formula first, as follows.

Consider a body of mass m on the surface of the earth. Its weight is the magnitude of the force exerted on it by the earth's gravity, namely KM_Em/R^2. Its weight is also equal to m times the acceleration of the body, namely the acceleration due to gravity, usually denoted by g, and having the value 32.2 feet/sec^2.

It follows that

$$\frac{KM_em}{R^2} = mg,$$

and if we substitute into (1.16.6) we find that our time unit is

$$T_0 = \sqrt{\frac{R}{g}}\,. \tag{1.16.7}$$

We take $R = 4000$ miles, and find T_0 is about 13 minutes and 30 seconds. We propose to measure time in units of T_0. To that end, we multiply through equation (1.16.5) by T_0 and get

$$T_0^2 y'' = -\frac{1}{y^2} + \frac{\frac{M_M}{M_E}}{(60-y)^2}. \tag{1.16.8}$$

The ratio of the mass M_M of the moon to the mass M_E of the earth is about 0.012. Furthermore, we will now introduce a new independent variable τ and a new dependent variable $u = u(\tau)$ by the relations

$$u(\tau) = y(\tau T_0)\,; \qquad t = \tau T_0\,. \tag{1.16.9}$$

Thus, $u(\tau)$ represents the position of the rocket, measured in units of the radius of the earth, at a time τ that is measured in units of T_0, *i.e.*, in units of 13.5 minutes.

The substitution of (1.16.9) into (1.16.8) yields the differential equation for the scaled distance $u(\tau)$ as a function of the scaled time τ in the form

$$u'' = -\frac{1}{u^2} + \frac{0.012}{(60-u)^2}. \tag{1.16.10}$$

Finally we must translate the initial conditions (1.16.2) into conditions on the new variables. The first condition is easy: $u(0) = 1$. Next, if we differentiate (1.16.9) and set $\tau = 0$ we get

$$u'(0) = \frac{T_0 V}{R} = \frac{V}{R/T_0}. \tag{1.16.11}$$

This is a ratio of two velocities. In the numerator is the velocity with which the rocket is launched. What is the significance of the velocity R/T_0?

We claim that it is, aside from a numerical factor, the escape velocity from the earth, if there were no moon. Perhaps the quickest way to see this is to go back to equation (1.16.8) and drop the second term on the right-hand side (the one that comes from the moon). Then we will be looking at the differential equation that would govern the motion if the moon were absent. This equation can be solved. Multiply bth sides by $2y'$, and it becomes

$$T_0^2\left((y')^2\right)' = \left(\frac{2}{y}\right)',$$

and integration yields

$$T_0^2(y')^2 = \frac{2}{y} + C.$$

Now let $t = 0$ and find that $C = T_0^2V^2/R^2 - 2$, and so

$$T_0^2(y')^2 = \frac{2}{y} - \left(\frac{T_0^2V^2}{R^2} - 2\right). \tag{1.16.2}$$

Suppose the rocket is launched with sufficient initial velocity to escape from the earth. Then the function $y(t)$ will grow without bound. Hence let $y \to \infty$ on the right side of (1.16.12). For all values of y, the left side is a square, and therefore a non-negative quantity. Hence the right side, which approaches the constant C, must also be non-negative. Thus $C \geq 0$ or, equivalently

$$V \geq \sqrt{2}\frac{R}{T_0}. \tag{1.16.13}$$

Thus, if the rocket escapes, then (1.16.13) is true, and the converse is easy to show also. Hence the quantity $\sqrt{2}\,R/T_0$ is the *escape velocity* from the earth. We shall denote it by V_{esc}. Its numerical value is approximately $25,145$ miles per hour.

Now we can return to (1.16.9) to translate the initial conditions on $x'(t)$ into initial conditions on $u'(\tau)$. In terms of the escape velocity, it becomes $u'(0) = \sqrt{2}\,V/V_{esc}$. We might say that if we choose to measure distance in units of earth radii, and time in units of T_0, then velocities turn out to be measured in units of escape velocity, aside from the $\sqrt{2}$.

In summary, the differential equation and the initial conditions have the final form

$$\begin{aligned} u'' &= -\frac{1}{u^2} + \frac{0.012}{(60-u)^2} \\ u(0) &= 1 \\ u'(0) &= \sqrt{2}\frac{V}{V_{esc}} \end{aligned} \tag{1.16.14}$$

Since that was all so easy, let's try the two-dimensional case next. Here, the earth is centered at the origin of the xy-plane, and the moon is moving. Let the coordinates of the moon at time t be $(x_m(t), y_m(t))$. For example, if we take the orbit of the moon to be a circle of radius D, then we would have $x_m = D\cos(\omega t)$ and $y_m(t) = D\sin(\omega t)$.

If we put the rocket at a generic position $(x(t), y(t))$ on the way to the moon, then we have the configuration shown in figure 1.16.2.

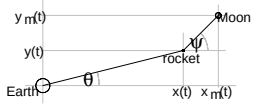

FIGURE 1.16.2

Consider the net force on the rocket in the x direction. It is given by

$$F_x = -\frac{KM_E m\cos\theta}{x^2+y^2} + \frac{KM_M m\cos\psi}{(x-x_m)^2+(y-y_m)^2}, \tag{1.16.15}$$

where the angles θ and ψ are shown in figure 1.16.2. From that figure, we see that

$$\cos\theta = x\sqrt{x^2+y^2}$$

and

$$\cos\psi = \frac{x_m - x}{sqrt(x_m-x)^2+)y_m-y)^2}.$$

Now we substitute into (1.16.15), and equate the force in the x direction to $mx''(t)$, to obtain the differential equation

$$mx''(t) = -\frac{KM_E mx}{(x^2+y^2)^{3/2}} + \frac{KM_M m(x_m-x)}{((x_m-x)^2+(y_m-y)^2)^{3/2}}. \tag{1.16.16}$$

If we carry out a similar analysis for the y-component of the force on the rocket, we get

$$my''(t) = -\frac{KM_E my}{(x^2+y^2)^{3/2}} + \frac{KM_M m(y_m-y)}{((x_m-x)^2+)y_m-y)^2)^{3/2}}. \tag{1.16.17}$$

We are now looking at two (even nastier) simultaneous differential equations of the second order in the two unknown functions $x(t),\ y(t)$ that describe the position of the rocket. To go with these equations, we need four initial conditions. We will suppose that at time $t=0$, the rocket is on the earth's surface, at the point $(R,0)$. Further, at time $t=0$, it will be fired with an initial speed of V, in a direction that makes an angle α with the positive x-axis. Thus, our initial conditions are

$$\begin{cases} x(0) = R\,; & y(0) = 0 \\ x'(0) = V\cos\alpha\,; & y'(0) = V\sin\alpha \end{cases}. \tag{1.16.18}$$

The problem has now been completely defined. It remains to change the units into the same natural dimensions of distance and time that were used in the one-dimensional problem. This time we leave the details to the reader, and give only the results. If $u(\tau)$ and $v(\tau)$ denote the x and y coordinates of the rocket, measured in units of earth radii, at a time τ measured in units of T_0 (see (1.16.7)), then it turns out the u and v satisfy the differential equations

$$\begin{aligned} u'' &= -\frac{u}{(u^2+v^2)^{3/2}} + \frac{0.012(u_m-u)}{((u_m-u)^2+(v_m-v)^2)^{3/2}} \\ v'' &= -\frac{v}{(u^2+v^2)^{3/2}} + \frac{0.012(v_m-v)}{((u_m-u)^2+(v_m-v)^2)^{3/2}}. \end{aligned} \tag{1.16.19}$$

Furthermore, the initial data (1.16.18) take the form

$$\begin{cases} u(0) = 1\,; & v(0) = 0 \\ u'(0) = \sqrt{2}\dfrac{V\cos\alpha}{V_{esc}}\,; & v'(0) = \sqrt{2}\dfrac{V\sin\alpha}{V_{esc}}. \end{cases} \tag{1.16.20}$$

In these equations, the functions $u_m(\tau)$ and $v_m(\tau)$ are the x and y coordinates of the moon, in units of R, at the time τ. Just to be specific, let's decree that the moom is in a circular orbit of radius $60R$, and that it completes a revolution every twenty-eight days. Then, after a brief session with a hand calculator or a computer, we discover that the equations

$$\begin{aligned} u_m(\tau) &= 60\cos(0.002103745\tau) \\ v_m(\tau) &= 60\sin(0.002103745\tau) \end{aligned} \tag{1.16.21}$$

represent the position of the moon.