

Lecture Notes for Mathematics 1070: Numerical Analysis

William Layton
Math. Dept, Univ. of Pittsburgh

August, 2005

Overview:

These notes are written solely for the convenience of my current students in Math 1070, Numerical Analysis and are intended as a supplement to the class lectures. They are mostly typed; you will surely find some errors, typographical and otherwise, herein. This year's version has been updated from last year's as follows: much more is typed and much less handwritten, several of the presentations have been simplified, page numbers have been inserted on all the pages, several new topics are included and more proofs included. My previous student's were interested in the mathematics behind the algorithms and asked for more proofs! I have included proofs which give real insight into "why" something works; we will do more such proofs in class.

My current students are welcome to make a copy of these notes for their own use. All other rights are reserved!

**Welcome to the interesting and beautiful world
of numerical analysis!**

Numerical Analysis.

William Layton

Contents.

Chapter 1. Computer arithmetic in numerical analysis.

- 1.1. Roundoff error.
- 1.2. Measuring and controlling error.
- 1.3. Stability.
- 1.4. Mathematical preliminaries.
- 1.5. A case study: numerical differentiation.
- 1.6. A case study: numerical integration.
- 1.7. A case study: polynomial interpolation.
- 1.8. A case study: solving $f(x) = 0$.
- 1.9. A case study: solving an initial value problem.

Chapter 2. The numerical solution of nonlinear equations.

- 2.1. Introduction.
- 2.2. Simple iteration.
- 2.3. Newton's method.
- 2.4. Nonlinear simultaneous equations.
- 2.5. Globalization strategies: the example of homotopy methods.

Chapter 3. Numerical differentiation and integration

- 3.1. Introduction: Approximations of linear functionals.
- 3.2. Numerical differentiation.
- 3.3. Numerical integration: Basic Newton-Cotes rules.
- 3.4. Numerical integration: Gauss quadrature.
- 3.5. Adaptive quadrature

Chapter 4. Numerical methods for ordinary differential equations.

- 4.1. Introduction.
- 4.2. More on Euler's method.
- 4.3. Convergence of Euler's method.
- 4.4. Runge-Kutta methods.
- 4.5. Adaptive time step selection.
- 4.6. Stiffness and implicit methods.

Chapter 5. Curve fitting.

- 5.1. Introduction.
- 5.2. The interpolating polynomial.
- 5.3. Least squares approximation.
- 5.4. Orthogonal polynomials and least squares approximations.
- 5.5. Cubic splines.

Computer Arithmetic in Numerical Analysis

William J. Layton

1.1 Roundoff Error.

The basic problems in numerical analysis are:

1. Representing a function on a computer;
2. Solving linear systems of equations;
3. Solving nonlinear systems of equations;
4. Differentiating and integrating numerically;
5. Solving initial value problems for differential equations;
6. Solving boundary value problems for differential equations.

Computers work in finite precision (base 2) number system. This fact causes extra problems in accomplishing (1) - (6). Every operation a computer performs introduces errors (**roundoff errors**). The worse cases of round off errors are usually introduced in your program in one of the following ways:

1. Input errors.

For example, the following two statements were found in FORTRAN programs on a machine with 24 significant digits base 10.

$$\left. \begin{array}{l} PI = 3.1416 \\ PI = 22.0/7.0 \end{array} \right\} \text{WRONG!}$$

To preserve the machine's accuracy π must be input to 24 significant digits.

$$PI = 3.14159 \dots \text{ (to 24 digits)}$$

A sneaky way around this is:

$$PI = 4.0 * ATAN(1.0)$$

2. Formatting Errors.

This includes such things as using single precision variables in double precision calculations.

3. Subtracting Nearly Equal Numbers.

This is a frequent cause of roundoff error since such subtraction causes a loss of significant digits. For example, in a 4-digit mantissa base 10 computer, suppose we do:

$$.1234 \times 10^1 - .1233 \times 10^1 = .1000 \times 10^{-3}.$$

We go from four significant digits to one. Thus, a 1% error in $.1233 \times 10^1$ can become a 1000% error in the answer!

4. Adding a Large Number to a Small One.

This causes the effect of the small number to be completely lost. For example, suppose that in our 4-digit computer we perform

$$X = .1234 * 10^3 + .1200 * 10^{-2}.$$

This is done by making the exponents alike and adding the mantissas:

$$\begin{array}{r} .1234 * 10^3 \\ + \quad \begin{array}{l} \text{CHOP} \\ \swarrow \\ .0000 / 01200 * 10^3 \\ \searrow \\ \text{OR ROUND} \end{array} \\ \hline .1234 * 10^3 \end{array}$$

This can have profound effects when summing a series.

Exercise: On a PC ^{either} sum the geometric series (approximating 2) forward and backward: ^{or by hand using 4 digit arithmetic (with fewer terms)}
1.1.1. Compare and explain the results.

$$2 \cong \sum_{k=0}^{10,000} \frac{1}{2^k}$$

$$2 \cong \sum_{k=10,000}^0 \frac{1}{2^k}$$

Computer Exercise

The Fibonacci sequence are generated by the recursion

$$a_1 = 1, a_2 = 1, a_{n+1} = a_n + a_{n-1}, n = 2, 3, \dots$$

Here's a bit of FORTRAN to generate the first 100 Fibonacci numbers

```
PROGRAM FIBO
INTEGER FIBO(100)
FIBO (1) = 1
FIBO (2) = 1
DO 10 I = 3,100
FIBO (I) = FIBO (I-1) + FIBO (I-2)
10 CONTINUE
DO 20 J = 1, 100, 10
PRINT *, "I = ", I, " Ith FIBONACCI NUMBER = ", FIBO (I)
20 CONTINUE
STOP
END
```

Try running this. What problems does this have? Convert FIBO(100) to a real array and try again. If we define $RATIO(I) = FIBO(I) / FIBO(I-1)$ it is known that $RATIO(I)$ converges to the golden mean as $I \rightarrow \infty$. Add another loop to compute and output $RATIO(I)$, the percent difference between $RATIO(I)$ and the golden mean. How fast does it approach the golden mean?

5. Dividing by a Small Number.

This has the effect of magnifying errors. A small percent error can become a large percent error when divided by a small number.

Example

Suppose we compute, using four significant digits, the following:

$$x = A - B/C,$$

where

$$A = 0.1102 \times 10^9,$$

$$B = 0.1000 \times 10^6,$$

$$C = 0.9000 \times 10^{-3}.$$

We obtain $B/C = .1111 \times 10^9$ and $x = 0.9000 \times 10^6$.

Suppose instead that there is a 0.01% error in calculating C , namely

$$C = 0.9001 \times 10^{-3}.$$

Then we calculate instead

$$B/C = 0.1110 \times 10^9 \text{ so } x = 0.1000 \times 10^7.$$

Thus we have an 11% error in the result!

Exercise 1.12. What are the main causes of serious roundoff error. Give a (new) example of each.

1.2 Measuring and Controlling Roundoff Error.

Since every arithmetic operation induces roundoff error it is useful to come to grips with it on a quantitative basis. Suppose a quantity is calculated by some approximate process.

The result, x_{COMPUTED} , is seldom the exact or true result, x_{TRUE} . Thus, we measure errors by the following convenient standards:

$$e = \text{ERROR} := x_{\text{TRUE}} - x_{\text{COMPUTED}},$$

$$e_{\text{ABSOLUTE}} := |x_{\text{TRUE}} - x_{\text{COMPUTED}}|,$$

$$e_{\text{RELATIVE}} := |x_{\text{TRUE}} - x_{\text{COMPUTED}}| / |x_{\text{TRUE}}|,$$

$$e_{\text{PERCENT}} := e_{\text{RELATIVE}} * 100.$$

Of course, we seldom know the true solution so it is useful to get a “ballpark” estimate of error sizes. Here are some universally used “standard” ways to do this:

1. (Estimating roundoff errors) Repeat the calculation in double precision. The digit where the two results differ represents the place where roundoff error has influenced the single precision calculation.
2. (Estimating errors in the arithmetic model) Solve the problem at hand twice—once with a given model and second with a more “refined” or accurate arithmetic model. The difference between the two can be taken as a ballpark measure of the error in the less accurate discrete model.
3. (Interval Arithmetic for estimating roundoff and other errors) As a calculation proceeds, we track not only the arithmetic result but also a “confidence interval” in which the solution sought is guaranteed to lie. The evolution of this “confidence interval” is predicted via a worse case type of calculation at every step.
4. (Significant Digit Arithmetic) Similarly to interval Arithmetic, the number of significant digits are tracked through each computation.
5. (Backward error analysis for studying sensitivity of problem to roundoff error) For many types of computations, it has been shown rigorously that “*the solution computed using finite precision arithmetic is precisely the exact solution in exact arithmetic to a perturbation of the original problem.*”. Thus the sensitivity of a calculation to roundoff error can be examined by studying the sensitivity of the continuous problem to perturbations in its data.

6. (Statistical analysis of roundoff error propagation) Rather than a worse-case type scenario, roundoff errors are more or less normally distributed. The statistical analysis of roundoff error proceeds under assumption of a normal distribution. Results typically agree well with practice.

Exercise 1.2.1. Relate the number of significant digits of accuracy ^{in an approximation} to having a relative error smaller than 10^{-k} .

1.3 Stability.

Since scientific calculations involve millions of operations in sequence, some way must be found to measure the effect of a small roundoff error on the error in later calculations. If such an error grows as further calculations are performed the algorithm is called *unstable*. If roundoff errors are damped by the method (or don't grow) the method is called *stable*.

The underlying problem can also be *stable* or *unstable* also. Consider the following (case study) of simply evaluating a function.

Example: We seek $f(x^*)$ where $f(x)$ can be calculated exactly (a *very ideal* situation) but x^* cannot be represented exactly on the computer. Thus, we calculate $f(x)$ where x is "close to" x^* .

We call the quotient of the relative change in f induced by a relative change in x^* , the condition of f at x^* .

Definition 1.3. The condition of $f(x)$ at x^* is defined to be:

$$\text{cond } f(x^*) := \lim_{x \rightarrow x^*} \frac{\left| \frac{f(x^*) - f(x)}{f(x^*)} \right|}{\left| \frac{x^* - x}{x^*} \right|}.$$

This is a very instructive number to get a grip on. For example, if $\text{cond } f(x^*) = 10$, then a 1% error in x^* becomes a 10% error in f .

We can estimate $\text{cond } f(x^*)$ by noting that the mean value theorem tells us:

$$\begin{aligned} \text{cond } f(x^*) &= \left| \frac{f(x^*) - f(x)}{x^* - x} \right| \cdot \left| \frac{x^*}{f(x^*)} \right| \\ &\cong \left| \frac{f'(x^*)x^*}{f(x^*)} \right| \end{aligned}$$

since x is "close to x^* ".

1.3.1.

Exercise: (a) Find $\text{cond}(e^x)$ at $x = \pm 10$. (b) Let $S_N(x) = \sum_{n=0}^N \frac{x^n}{n!}$, and find $\text{cond} S_N(x)$ at $x = \pm 10$, for $N = 10, 20, 30$. Compare $\text{cond} e^x$ with $\text{cond} S_N(x)$.

1.3.2. If $\text{cond}(f(x)) = 10$ and x has a 1% error, what is the anticipated error in $f(x)$?

Examples: 1. If $f(x) = \sqrt{x}$, then $f'(x) = \frac{1}{2}x^{-\frac{1}{2}}$ and

$$\text{cond}(f(x)) = \left| \frac{f'(x)x}{f(x)} \right| = \frac{\frac{1}{2}x^{-\frac{1}{2}}x}{x^{\frac{1}{2}}} = \frac{1}{2}.$$

2. If $f(x) = \frac{10}{1-x^2}$, then $f'(x) = \frac{20x}{(1-x^2)^2}$ so that $\text{cond}(f(x)) = \frac{2x^2}{|1-x^2|}$. Note that if x is near 1 $\text{cond}(f)$ is very large and $\text{cond}(f) \rightarrow \infty$ as $x \rightarrow 1$.

1.3.3.

Exercise: Let $\beta_{1,2}$ be the roots of

$$a^*\beta^2 + b^*\beta + c^* = 0$$

given by the quadratic formula

$$\beta_{1,2} = \frac{-b^* \pm \sqrt{b^{*2} - 4a^*c^*}}{2a^*}.$$

Thus, $\beta = \beta(a, b, c)$. Suppose $\beta_1 = \beta_2$ that is: $(b^{*2} - 4a^*c^* = 0)$ and find the condition of β with respect to a, b and c individually. (**Hint:** use partial derivatives.)

1.4 Mathematical Preliminaries.

The notion of condition of a function at a point is related to the idea of continuity. If a function $f(x)$ is not continuous at x^* , then any small in x can yield large changes in the function value (the condition is "infinite".) Conversely, if $f(x)$ is continuous at x^* then, in principle at least and with infinite precision arithmetic, etc., small changes in x near x^* cannot produce large changes in the function value. Mathematically, continuity at x^* is defined as follows.

Definition. $f(x)$ is continuous at x^* if, for any to positive tolerance ϵ in the function, there is a positive tolerance δ in the argument such that whenever $|x - x^*| < \delta$ it follows that $|f(x) - f(x^*)| < \epsilon$.

The "big O" notation

Definition 1.4. We say that $f(h)$ is $O(h^\alpha)$ as $h \rightarrow 0$ if $f(h)$ is bounded by:

$$|f(h)| \leq C|h|^\alpha \quad 0 \leq h \leq h_0.$$

for some $C > 0$.

Many examples of "O" type notation arise from Taylor's theorem. As a first example,

Theorem 1.42. Suppose $f(x)$ is C^1 . Then,

$$f(x+h) - f(x) = O(h) \quad \text{as } h \rightarrow 0.$$

Proof. The mean value theorem tells us that

$$\frac{f(x+h) - f(x)}{h} = f'(\xi), \quad x < \xi < x+h.$$

Letting $C = \max_{x < t < x+h_0} |f'(t)|$ we then have by rearranging the previous equation,

$$|f(x+h) - f(x)| \leq Ch, \quad \text{as } h \rightarrow 0. \quad \square$$

Theorem 1.43. (Taylor's Theorem with Remainder) Suppose $f(x)$ has $n+1$ continuous derivatives on $A \leq x \leq B$, and that c, x lie on $[A, B]$. Then, for every such c, x there is a ξ between A and B , $A < \xi < B$, such that

$$\begin{aligned} f(x) &= f(c) + f'(c)(x-c) + \frac{f''(c)}{2!}(x-c)^2 + \dots \\ &\dots + \frac{f^{(n)}(c)}{n!}(x-c)^n + E_n. \end{aligned}$$

The error term E_n is given by

$$E_n = \frac{f^{(n+1)}(\xi)}{(n+1)!} (x-c)^{n+1}. \quad \square$$

Example: Expand $f(x+h)$ in a Taylor series centered at x in the variable h up to 5 terms:

$$f(x+h) = f(x) + f'(x)(x+h-x) + \frac{f''(x)}{2!} (x+h-x)^2 + \dots + \frac{f^{(4)}(x)}{4!} (x+h-x)^4 + E_4$$

where

$E_4 = O(h^5)$ Specifically,

$$E_4 = \frac{f^{(5)}(\xi)}{5!} h^5$$

(insert here)

Exercises

1.4.1. If $f(x)$ is smooth, show that

$$\frac{1}{2}(f(a+h) + f(a-h)) = f(a) + O(h^2).$$

1.4.2. If $f = f(x, y)$ is a smooth function of two variables, expand $f(x+h, y+h)$ and $f(x-h, y+h)$ in Taylor series through $O(h^4)$ terms.

1.5 A Case Study: Numerical Differentiation.

One fundamental question in numerical analysis is to estimate rates of change given function data. In other words, given $f(x)$ at certain points x_j (possibly with error) estimate $f'(x)$ at certain points.

Recalling

$$f'(a) := \lim_{h \rightarrow 0} \frac{f(a+h) - f(a)}{h}$$

we can estimate $f'(a)$ by approximating for h "small".

$$(1.5.1) \quad f'(a) \cong \frac{f(a+h) - f(a)}{h} = (f(a+h)/h) - (f(a)/h).$$

This calculation contains *two* sources of possibly serious roundoff errors:

1. Subtraction of two nearly equal numbers ($f(a+h) - f(a)$).

2. Division by a small number (h).

In performing (1.5.1) two sources of error can be thus significant:

Discretization error (already in the approximation (1.5.1) and

Roundoff error (when (1.5.1) is executed in finite precision Arithmetic.)

Discretization Errors

The discretization error in (1.5.1) can be estimated using Taylor's theorem with remainder as follows. Expanding

$$f(a+h) = f(a) + f'(a)h + \frac{f''(\xi)}{2!} h^2, \text{ some } \xi \text{ between } a \text{ and } a+h,$$

gives

$$\frac{f(a+h) - f(a)}{h} = f'(a) + h \frac{f''(\xi)}{2!}$$

so the absolute error is bounded by:

$$(1.5.2) \quad \left| f'(a) - \frac{f(a+h) - f(a)}{h} \right| \leq \frac{h}{2} \max_{a \leq x \leq a+h} |f''(x)|.$$

This is *linear convergence* since it predicts that whenever h is cut in half the error is also cut in half. It is also written that "the error in (1.5.1) is $O(h)$."

Roundoff error in numerical differentiation

When (1.5.1) is performed the values $f(a+h)$ and $f(a)$ cannot be assumed to be exact: they each have errors ϵ_1, ϵ_2 of O (machine precision) (or worse). Thus we actually compute:

$$(1.5.3) \quad f'(a) \cong \frac{f(a+h) \pm \epsilon_1 - f(a) \pm \epsilon_2}{h} =: f'(a)_{\text{COMPUTED}}.$$

There is no reason to suppose $\pm\epsilon_1, \pm\epsilon_2$ to exactly cancel thus (1.5.3) can be written as:

$$(1.5.4) \quad f'(a) \cong \frac{f(a+h) - f(a)}{h} + \frac{2\epsilon}{h}, \epsilon = O(\text{machine precision}).$$

We have thus for the absolute error (using (1.5.2) and (1.5.4))

$$(1.5.5) \quad |f'(a) - f'(a)_{\text{COMPUTED}}| \leq C_1 \frac{h}{2} + \frac{2\epsilon}{h},$$

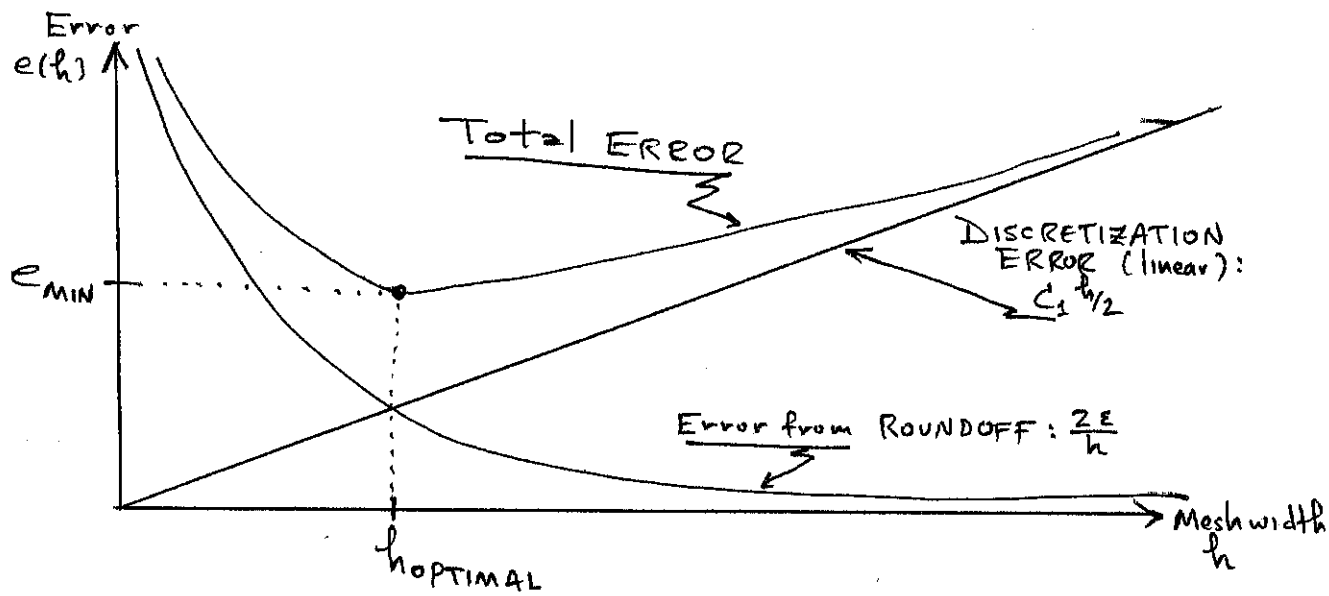
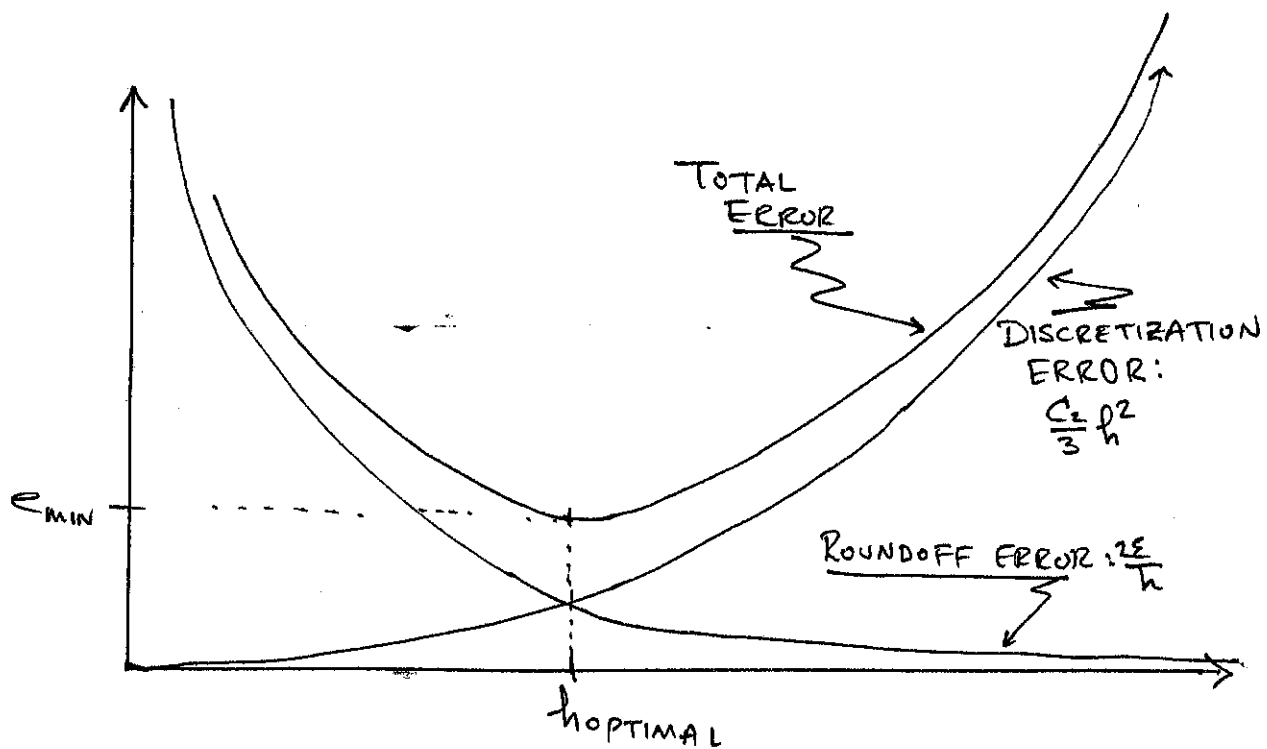


FIGURE 1.5.1: TOTAL ERROR FOR LINEAR, $O(h)$, METHOD.

FIGURE 1.5.2: TOTAL ERROR FOR $O(h^2)$ METHOD



where $C_1 = \max_{a < x < a+h} |f''(x)|$ is an $O(1)$ constant.

(1.5.5) is an important inequality to understand so let's graph the RHS $e(h) = C_1 \frac{h}{2} + \frac{2\epsilon}{h}$ as the sum of two curves.

We can find using calculus (set $e'(h) = 0$ and solve for h).

$$h_{\text{OPTIMAL}} = \frac{2}{\sqrt{C_1}} \sqrt{\epsilon} \cong O(\sqrt{\text{machine precision}}),$$

$$e_{\text{min}} = 2\sqrt{C_1} \sqrt{\epsilon} \cong O(\sqrt{\text{machine precision}}).$$

In viewing such a figure you should always ask yourself which portion is particular to the formula (1.5.1) and which is universal. We shall see that the general shape and the contribution of roundoff error is universal. The discretization error curve and the values of h_{OPTIMAL} and e_{min} are particular to (1.5.1).

Let's now consider other methods which give a *smaller* value of e_{min} for a *larger* value of h (since the only way to increase the accuracy of e_{min} for this method is to use extended precision).

Consider the *central difference approximation*:

$$f'(a) \cong \frac{f(a+h) - f(a-h)}{2h}.$$

Repeating the previous analysis gives:

$$\left| f'(a) - \frac{f(a+h) - f(a-h)}{2h} \right| \leq C_2 \frac{h^2}{6}, \text{ and}$$

$$\left| f'(a) - f'(a)_{\text{COMPUTED}} \right| \leq C_2 \frac{h^2}{3} + \frac{\epsilon}{h}.$$

which is depicted in the next figure.

Again we find for the central difference approximation:

$$h_{\text{COMPUTED}} = \left(\frac{3}{2}C_2\right)^{\frac{1}{3}} \sqrt[3]{\epsilon} = O([\text{machine precision}]^{\frac{1}{3}}),$$
$$e_{\text{min}} = O(\epsilon^{\frac{3}{4}}) = O([\text{machine precision}]^{\frac{3}{4}}).$$

Note the improvement attainable using a $O(h^2)$ accurate method (central differences) vs. an $O(h)$ accurate method.

$$h_{\text{OPTIMAL}} = \left(\frac{3}{2}C_2\right)^{\frac{1}{3}} \sqrt[3]{\epsilon} = O([\text{machine precision}]^{\frac{1}{3}}),$$
$$e_{\text{min}} = O(\epsilon^{\frac{3}{2}}) = O([\text{machine precision}]^{\frac{3}{2}}).$$

Note the improvement attainable using an $O(h^2)$ accurate method (central differences) vs. an $O(h)$ accurate method.

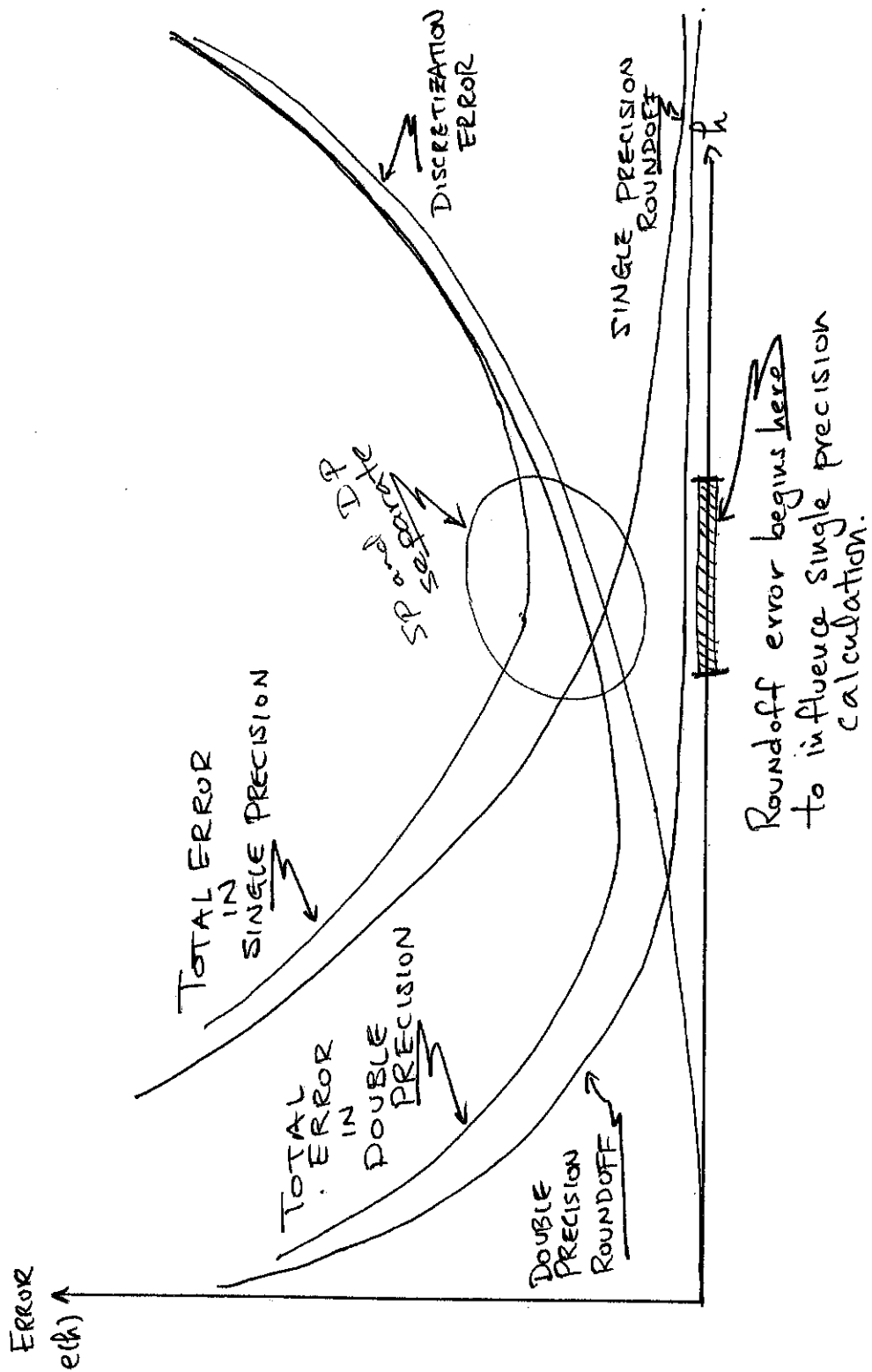


FIGURE 1.5.3: ESTIMATION OF EFFECTS OF ROUND OFF.

Exercises

1.5.1. Consider the central difference approximation to $f'(a)$

$$f'(a) \sim (f(a+h) - f(a-h)) / 2h$$

Show that if $f \in C^\infty$ then the error has a Taylor expansion in h with only even powers of h :

$$\begin{aligned} E(h) &:= f'(a) - (f(a+h) - f(a-h)) / 2h \\ &= a_2 h^2 + a_4 h^4 + a_6 h^6 + \dots \end{aligned}$$

1.5.2. Using the result of problem 1, find an $O(h^4)$ approximation to $f'(a)$ as follows.

$$\text{Let } D_h f(a) = (f(a+h) - f(a-h)) / 2h$$

$$\text{So that } D_{2h} f(a) = (f(a+2h) - f(a-2h)) / 4h.$$

$$\text{Consider } D^\theta f(a) = \theta D_h f(a) + (1-\theta) D_{2h} f(a).$$

Show that the error in D_θ satisfies

$$f'(a) - D^\theta f(a) = (\theta a_2 + (1-\theta)a_2 2^2)h^2 + (\theta a_4 + (1-\theta)a_4 2^4)h^4 + \dots$$

Find a θ value for which the $O(h^2)$ terms vanish.

1.5.3. Suppose $f'(a) = 3.2 \times 10^{-5}$ and approximations to $f'(a)$ are computed for various values of h , giving

h	Approximation to $f'(a)$	Error $e(h)$
1/10	9.1×10^{-5}	
1/20	4.6×10^{-5}	
1/30	3.9×10^{-5}	

Find the (experimental) rate of convergence of the (unknown) method.

Hint: Calculate the errors and fill in the above table. Assume $e(h) = Ch^\alpha$ in each case and solve for C and α for each pair of observed errors. You are trying to find the “ α ” value.

1.5.4. Consider $f'(a) \sim D_h f(a) := (f(a+h) - f(a-h)) / 2h$. Upon what degree polynomials is this exact? Hint: Begin checking:

$1' = 0$ and $D_h 1 = (\text{what})$, $x' = 1$, $D_h x = (\text{what})$, and so on.

COMPUTER EXERCISE

Try running the short program below. It computes two approximations to $f'(a)$

PROGRAM DIFF

$$F(x) = x * \exp(x)$$

$$FP(x) = x * \exp(x) + \exp(x)$$

$$a = 1.0$$

$$H = 0.5$$

$$\text{DO } 10 \text{ I} = 0, 10$$

$$H = H / (2.0 * I)$$

$$D1F = (F(a+H) - F(a)) / H$$

$$\text{DOF} = (F(a+H) - F(a-H)) / (2.0 * H)$$

$$E1 = \text{ABS}(FP(a) - D1F)$$

$$E0 = \text{ABS}(FP(a) - \text{DOF})$$

PRINT *, H, D1F, E1, DOF, E0

10 CONTINUE

STOP

END

Run this program for different values of H .
What trends do you see experimentally? Do you see the predicted $O(h)$ and $O(h^2)$ convergence rates?

Try a different $F(x)$ and number of levels!

Does roundoff error influence any of your calculations? Explain³ your conclusions carefully.

1.6. A Case Study : Numerical Intigration

The basic idea of numerical integration is similar to numerical differentiation : a function is locally replaced by a polynomial which is integrated exactly. As an example, consider the “composite” trapezoid rule.

The interval $[a,b]$ is divided into n subintervals (x_j, x_{j+1}) where:

$$A = x_0 < x_1 < x_2 < \dots < x_n = b.$$

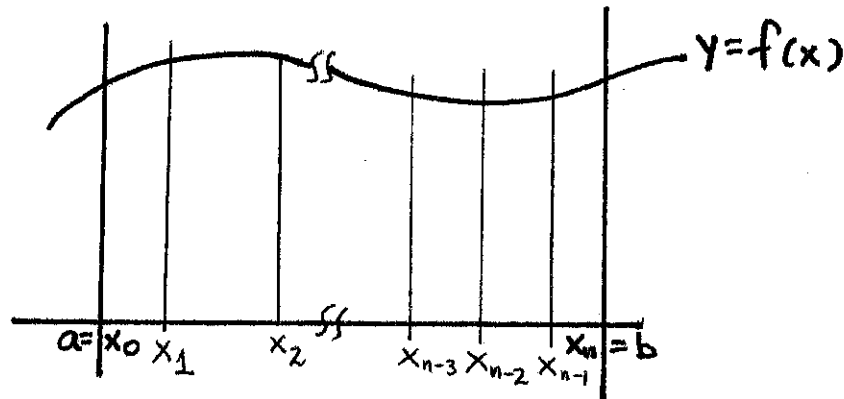


Figure : A composite integration rule.

Write:

$$\int_a^b f(x) dx = \sum_{j=0}^{n-1} \int_{x_j}^{x_{j+1}} f(x) dx$$

Each small intervals integral is approximated as above. Consider the linear case first.

On $[x_j, x_{j+1}]$ approximate $f(x)$ by a linear polynomial : the secant line through $(x_j, f(x_j))$

and $(x_{j+1}, f(x_{j+1}))$:

$$f(x) \doteq P_1(x) := \frac{f(x_{j+1}) - f(x_j)}{x_{j+1} - x_j} (x - x_j) + f(x_j), \quad x \in [x_j, x_{j+1}]$$

Then approximate :

$$\begin{aligned} \int_{x_j}^{x_{j+1}} f(x) dx &\doteq \int_{x_j}^{x_{j+1}} P_1(x) dx = (\text{after a simple calculation}) \\ &= \frac{(x_{j+1} - x_j)}{2} f(x_j) + \frac{(x_{j+1} - x_j)}{2} f(x_{j+1}). \end{aligned}$$

Using this approximation on each sub-interval gives :

$$\int_a^b f(x) dx \doteq \sum_{j=0}^{n-1} (x_{j+1} - x_j) \left(\frac{f(x_{j+1}) + f(x_j)}{2} \right).$$

This is called the "trapezoid rule" for the following reason. The area under $y = f(x)$ on $[x_j, x_{j+1}]$ is approximated by the area under the secant line – which is a trapezoid :

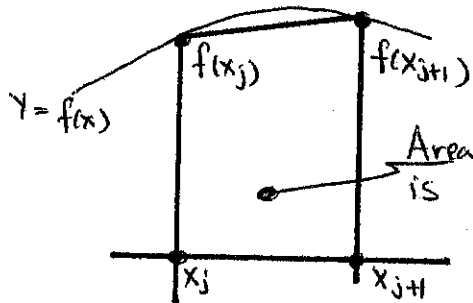


Figure: Area of trapezoid approximates area under curve.

$$\text{Area of trapezoid is } (x_{j+1} - x_j) \left(\frac{f(x_j) + f(x_{j+1})}{2} \right)$$

Note that the x_j 's do not have to be equally spaced. In fact, it's almost never good to let them be equally spaced – as we shall see!

The accuracy of this method has been rigorously proven:

Theorem 1.6.1 Let $h = \max (x_{j+1} - x_j)$ and suppose f'' is continuous. Then,

$$\left| \int_a^b f(x) dx - \sum_{j=0}^{n-1} (x_{j+1} - x_j) \frac{f(x_j) + f(x_{j+1})}{2} \right| \leq \frac{b-a}{12} h^2 \max_{a < x < b} |f''(x)|.$$

There is another way to derive the key approximation step. Suppose we seek a formula of the form we used before:

$$\int_{x_j}^{x_{j+1}} f(x) dx \doteq w_j f(x_j) + w_{j+1} f(x_{j+1}).$$

If we ask this be exact on $f(x)=1$ and $f(x)=x$ we get

$$\begin{aligned} \int_{x_j}^{x_{j+1}} 1 dx &= (x_{j+1} - x_j) = w_j \cdot 1 + w_{j+1} \cdot 1 \\ \int_{x_j}^{x_{j+1}} x dx &= \frac{x_{j+1}^2}{2} - \frac{x_j^2}{2} = w_j x_j + w_{j+1} x_{j+1} \end{aligned}$$

If we solve this 2x2 system for w_j, w_{j+1} we get

$$w_j = (x_{j+1} - x_j) / 2$$

$$w_{j+1} = (x_{j+1} - x_j) / 2$$

exactly as before!

What are the main issues in numerical integration? Like always, they are accuracy, cost and reliability. The first goal is to get much greater accuracy than the trapezoid rule for the same cost. "Cost" is measured in the number of function evaluations performed since that is by far the most computer time consuming operation.

The reliability question is : given a user-specified target accuracy, compute an approximation to $\int_a^b f(x) dx$ (by picking the points x_j) which attains this guaranteed accuracy with near minimal cost.

1.6.1 Suppose a small amount of (round off) error E_j is committed each time $f(x_j)$ is evaluated. Find an expression for the global round off error. Using the discretization error estimate and assuming $E_j = E = O$ (machine precision) find the minimal attainable error and "optimal" h .

1.6.2. Take $f(x) = 1 / (1+x^2)$ and compute an approximation to $\int_0^1 f(x) dx$ using the trapezoid rule on the mesh

$$a = x_0 < x_1 < x_2 < \dots < x_n = b, \quad x_1 = \frac{1}{4}, \quad x_2 = \frac{2}{3}, \quad x_3 = x_n = 1.$$

What is the error? Cut this mesh in half and repeat. How are the two errors related?

1.6.3. Write the Trapezoid rule algorithm on the non uniform mesh

$$a = x_0 < x_1 < x_2 < \dots < x_n = b$$

in pseudo-code for approximating $\int_a^b f(x) dx$.

PROGRAM trapezoid

This program approximates the integral
from x=A to x=B of a function F(X) input below.
It uses the trapezoid rule.

INTEGER COUNTER

"COUNTER" will count the number of steps taken.
In many older programs such a counter will be
called "ICOUNT" and no "INTEGER" statement used.
It's better to call it what it is and declare
"COUNTER" to be an integer.

F(X)= 1.0/(1.0+X*X)

The first executable statement comes next.
Thus, any function statements MUST come above here!

The interval of integration is [A,B]

A=0.0
B=1.0

To alter the problem change above here.

N=100
H=(B-A)/REAL(N)

"SUM" will accumulate the value of the integral.

SUM=0.0
COUNTER = 0
HNEW=H
XL=A

XR=A+HNEW
10 TRAP=(XR-XL)*(F(XL)+F(XR))/2.0
SUM=SUM+TRAP
COUNTER=COUNTER+1
XL=XR
XR=XR+HNEW
IF(XR+HNEW.GE.B) GO TO 20

If you want to try changing the step size, you
would add a few statements next modifying HNEW.

GO TO 10

Now compute the integral on the last subinterval.

20 XR=B
TRAP=(XR-XL)*(F(XL)+F(XR))/2.0
SUM=SUM+TRAP
COUNTER=COUNTER+1
PRINT*, " The number of steps taken was: ",COUNTER
PRINT*, "A=",A," B=",B," INTEGRAL FROM A TO B IS:",SUM
STOP
END

1.7. A Case Study : Polynomial Interpolation

One basic problem in numerical analysis is to take a set of function values

$$(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$$

and produce a smooth function which agrees with these values. If the $(n+1)$ values are exact then one appropriate approach is to interpolate them with a polynomial of degree n .

With only two values, the solution is easy:

$$P_1(x) = \frac{y_1 - y_0}{x_1 - x_0} (x - x_0) + y_0.$$

The general case of $(n+1)$ values was solved by Lagrange when he was 16 years old.

Form the "Lagrange Functions" for the points x_0, x_1, \dots, x_n :

$$l_0(x) := \frac{(x-x_1)(x-x_2)\dots(x-x_n)}{(x_0-x_1)(x_0-x_2)\dots(x_0-x_n)} = \prod_{\substack{j=1 \\ j \neq 0}}^n \frac{(x-x_j)}{(x_0-x_j)}$$

$$l_k(x) := \prod_{j=1, j \neq k}^n \frac{(x-x_j)}{(x_k-x_j)}$$

$$l_n(x) := \frac{(x-x_0)(x-x_1)\dots(x-x_{n-1})}{(x_n-x_0)(x_n-x_1)\dots(x_n-x_{n-1})}$$

Note that each $l_k(x)$ satisfies:

- (i) $l_k(x)$ is a polynomial of degree n
- (ii) $l_k(x_k) = 1$
- (iii) $l_k(x_j) = 0, j \neq k$

The solution of the interpolation problem is then easily written down:

$$P_n(x) = y_0 l_0(x) + y_1 l_1(x) + \dots + y_n l_n(x).$$

Example : Three Points

If we have data at only three points:

$$(x_0, y_0), (x_1, y_1), (x_2, y_2),$$

the interpolating polynomial is (of course) quadratic. Consider the three (quadratic)

Lagrange functions for these three points:

$$l_0(x) = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)},$$

$$l_1(x) = \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)},$$

$$l_2(x) = \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}.$$

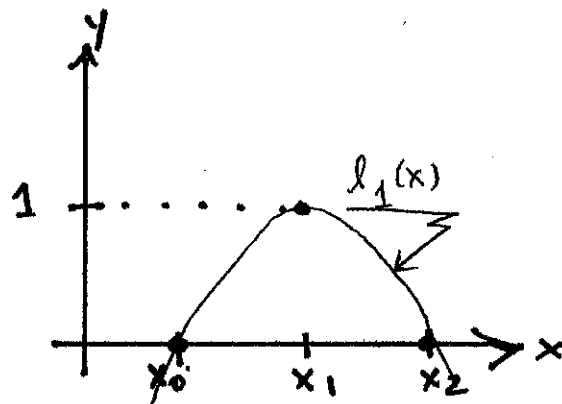
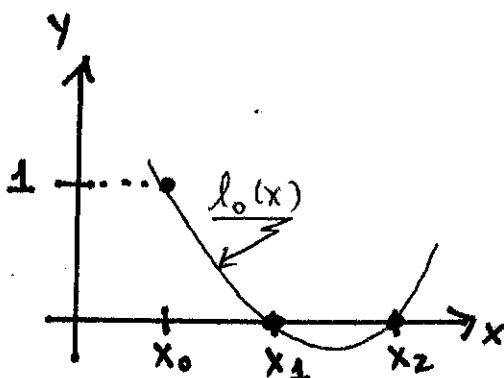
Note that

$$l_0(x) = 1, l_0(x_1) = 0, l_0(x_2) = 0, \text{ and}$$

$$l_1(x) = 0, l_1(x_1) = 1, l_1(x_2) = 0, \text{ and}$$

$$l_2(x) = 0, l_2(x_1) = 0, l_2(x_2) = 1.$$

It's easy to graph these: each is a quadratic determined at three points.



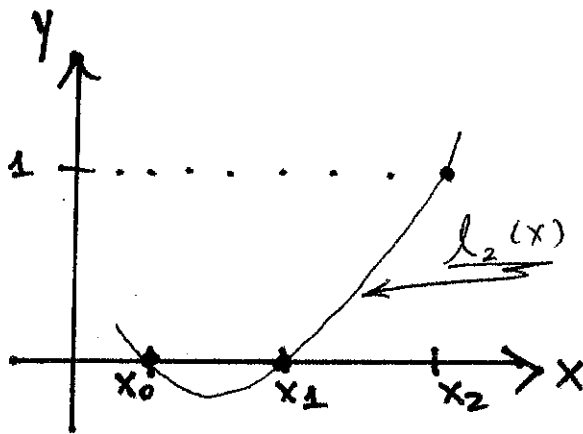


Figure: The Lagrange functions for three points. Note $l_j(x_k) = 0, j \neq k$.

In the case of any number of points the Lagrange functions are constructed to have exactly these properties.

Exercises:

- 1.7.1. Verify that $p_n(x_j) = y_j$.
- 1.7.2. If the points x_j are all distinct, prove that the polynomial interpolant is unique.

This simple and beautiful construction of Lagrange does not close the field.

Many problems remain open; in particular:

Indeed,

- cost: how do we evaluate $p_n(x)$ efficiently
- incoming data: if the data is continuously being augmented, the entire Lagrange interpolant must be recomputed.
- observation errors: small experimental or observational errors in the y_j can be magnified in $p_n(x)$.

The following is known about the error in polynomial interpolation.

Theorem 1.7.1. Suppose $y_j = f(x_j)$ and $f(x)$ is a smooth function (in particular, f has $(n+1)$ continuous derivatives.) Let $p_n(x)$ interpolate $f(x)$ at $x_0 < x_1 < x_2 < \dots < x_n$. Then, for any $x \in [x_0, x_n]$ there is a $\xi \in (x_0, x_n)$ such that

$$f(x) - p_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} (x-x_0)(x-x_1)\dots(x-x_n).$$

The error is thus super small if n is large ($1/(n+1)$ is small) and if the points are close together (so each $|x-x_j| < 1$).

Exercise

1.7.3. Consider the following data:

x_j	-1	0	1	$\frac{3}{2}$	2	3	4
y_j	8	3	0	$-\frac{3}{4}$	-1	0	3

- (a) Find the polynomial interpolant of this data using the Lagrange functions.
- (b) This data actually arises from a quadratic polynomial. Can you think of some way to discover this from the Lagrange form of the interpolant?

Proof of Theorem 1.7.1 (Sketch). Consider $w(x) = \prod_{j=0}^n (x-x_j)$. $w(x) = x^{n+1} +$ lower order terms and $w(x_j) = 0, j=0, \dots, n$. Thus, $w(x)$ resembles the sketch

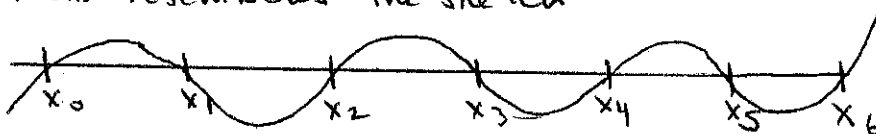


Figure: $w(x)$ when $n=6$, i.e., at 7 points.

Let $p_n(x)$ interpolate $f(x)$ at the points x_j . Consider $\phi(x) = f(x) - p_n(x) - \frac{e(y)}{w(y)} w(x)$. Note that ϕ has $n+2$ roots:

$$\phi(x_j) = 0, j=0, 1, \dots, n, \text{ and } \phi(y) = 0 \text{ at } n+2 \text{ points.}$$

By induction, we have that ϕ' has $n+1$ roots, etc., and $\phi^{(n+1)}$ has at least one root, $\phi^{(n+1)}(\xi) = 0$.

Computing $\phi^{(n+1)}(\xi)$ gives

$$0 = \frac{d^{n+1}}{dx^{n+1}} \left[f(x) - p_n(x) - \frac{e(y)}{w(y)} w(x) \right] \Big|_{x=\xi} = f^{(n+1)}(\xi) - \frac{e(y)}{w(y)} (n+1)! \quad \square$$

1.8. A Case Study : Solving $f(x) = 0$

If $f: \mathbb{R} \rightarrow \mathbb{R}$ the problem is to solve $f(\bar{x}) = 0$ for \bar{x} . This is equivalent to finding where the curve $y = f(x)$ touches the x axis. The intermediate value theorem implies that if $f(a)$ and $f(b)$ have opposite signs then $f(x)$ must be zero somewhere between a and b .

The Bisection Method

Input X_L and X_R such that $f(X_L) \cdot f(X_R) < 0$,

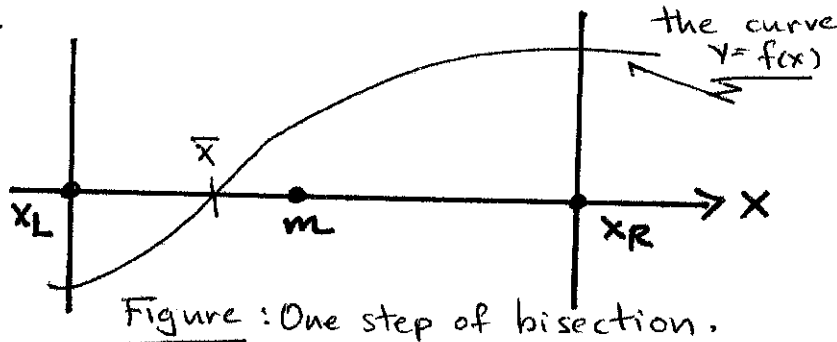
(*) $X_m = (X_L + X_R) / 2.0$

If $f(X_L) \cdot f(X_m) \leq 0$ set $X_R = m$ Otherwise set $X_L = m$.

Test if $|X_L - X_R| < \text{ToL}$ (a pre-specified tolerance)

If not, go to (*)

If so, stop.



In bisection, the root is always bracketed so convergence is guaranteed (although slow). Each step cuts the previous error by half so each step adds one binary digit of accuracy.

Exercise

1.8.1 Suppose we define w differently as the place where the (secant) line through the points

$$(X_L, f(X_L)), (X_R, f(X_R))$$

hits the x-axis. Write down this method (known as "false position") as an algorithm in pseudo-code.

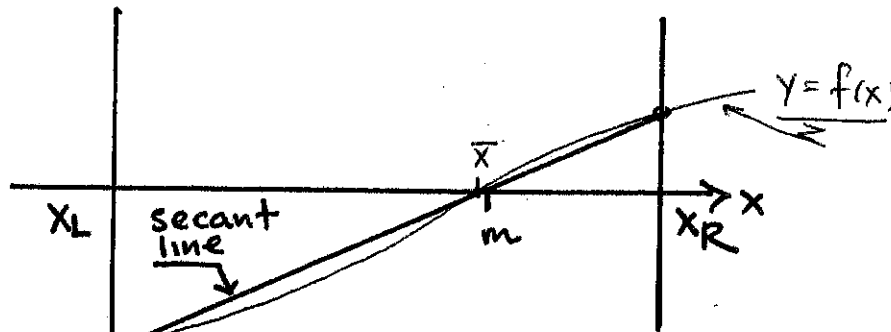


Figure: False position.

There is no secret to finding good initial guesses. One basic idea is to search: pick a Δx and an interval $[a, b]$ and look for a subinterval $(a+n\Delta x, a+(n+1)\Delta x)$ where $f(x)$ changes sign.

One improvement of the bisection method is described in the last exercise: the update is calculated using the secant line through $(X_L, f(X_L))$ and $(X_R, f(X_R))$. The second, and most important improvement is to use instead the tangent line through $(X_{old}, f(X_{old}))$. Indeed, this line is:

$$Y = f'(X_{OLD}) (X - X_{OLD}) + f(X_{OLD}).$$

This approximates $y = f(x)$ so its root can be taken as an updated approximation for x .

Setting $y = 0$ and solving for x give the famous Newton iteration:

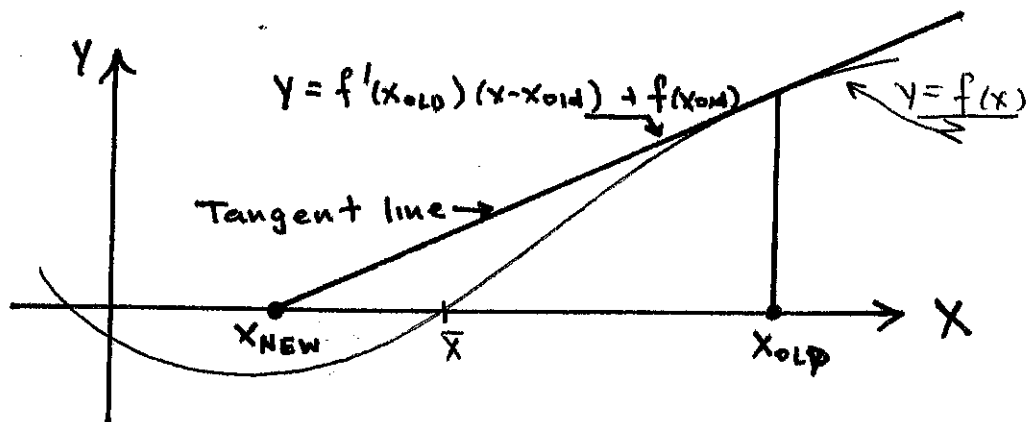


Figure: One Newton step.

1.8.2. Try to generalize bisection, false position and Newton's method to the 2-d problem:

$$f(x,y) = 0$$

$$g(x,y) = 0, \text{ solve for } (x,y).$$

You will soon see that there is no direct or natural generalization of bisection methods.

1.8.3. Consider calculating $(\sqrt{3})$ by solving $f(x) = x^2 - 3 = 0$. (a) Take $X_L = 1$, $X_R = 3$ and do three steps of the bisection method. (b) Write the bisection algorithm carefully in either pseudo-code or the language of your choice.

1.8.4. Repeat problem 2 for Newton's method.

1.9. A Case Study : Solving an Initial Value Problem

One basic task of computational science is the following. Knowing the initial state

$$y(0) = y_0 \text{ (} y_0 \text{ is a known number)}$$

and the "laws" governing a system

$$y'(t) = f(t, y(t)), \text{ for } 0 < t \leq T_{\text{final}},$$

predict the future! Specifically, find $y(t)$ for $t > 0$. This could be a single equation, a system of equations or involve higher derivatives. The ideas are all the same. Thus, we will first consider a scalar problem, like the above.

Naturally, we can't expect to "solve" for $y(t)$ in closed form. We pick a stepsize called Δt or h . The variables t_j and y_j denote $t_j = jh$ and y_j is the approximation we compute to $y(t_j)$:

$$\Delta t = h = \text{stepsize,}$$

$$t_j = jh = j^{\text{th}} \text{ time step,}$$

$$y_j \approx y(t_j).$$

The simplest way to find y_j is a method used by Euler to actually prove that initial value problems have solutions (ie., that the future exists!). It's constructive, so we can use Euler's method for calculations. It is motivated as follows: Suppose we know $y(t_j)$ exactly and want $y(t_{j+1}) \doteq y(t_j + \Delta t)$. Expanding y in a Taylor series at t_j gives:

$$y(t_j + h) = y(t_j) + y'(t_j)h + \frac{y''(\xi)}{2} h^2, \text{ for some } \xi, t_j < \xi < t_{j+1}.$$

Now $y'(t_j) = f(t_j, y(t_j))$ from the equation.

Thus:

$$y(t_{j+1}) = y(t_j) + h f(t_j, y(t_j)) + \frac{h^2}{2} y''(\xi),$$

where $t_j < \xi < t_{j+1}$.

The last term is "unknowable" but it is small if h is small. Just dropping this last term is

Euler's method:

$$y_{j+1} = y_j + h f(t_j, y_j), \quad y_0 \text{ given.}$$

Note that by doing this we commit an error $O(h^2)$ every step. This is called the "local truncation error" = the error in performing one step of the method:

$$\begin{aligned} \text{Local truncation error of Euler's method} &= y(t_{j+1}) - [y(t_j) + h f(t_j, y(t_j))] \\ &= \frac{h^2}{2} y''(\xi), \quad \text{where } t_j < \xi < t_{j+1} \end{aligned}$$

$$= (\text{by Taylor's theorem}) = \frac{h^2}{2} y''(t_j) + O(h^3)$$

$$(\text{Note that } \frac{d}{dt} f(t, y) = f_t(t, y) + f_y(t, y) y')$$

$$= (\text{by the equation, } y' = f(t, y) \text{ so } y'' = \frac{d}{dt} f(t, y(t)), \text{ and } y' = f(t, y))$$

$$\text{Local Truncation Error} = \frac{h^2}{2} [f_t(t, y(t)) + f_y(t, y(t)) f(t, y(t))] + O(h^3).$$

To calculate the solution from $t = 0$ to $t = T$ we take $J = T/h (= O(h^{-1}))$ steps. Since errors are additive we thus expect an error in $y(T)$ of $O(h)$. Indeed, we shall prove this is true.

Programming Euler's method is very simple. Here's an algorithm for it:

Algorithm Euler's method for $y' = f(t,y)$

$y(0) = y_0$ over $0 < t < T$.

Define the function $f(t,y)$

Input J , the number of steps

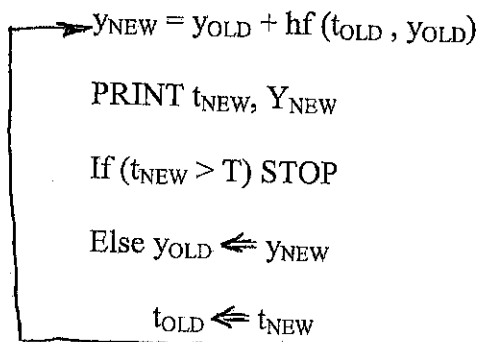
Input y_0 , the initial condition

Input T , the final time

Calculate $h = T/J$, the step size

$t_{\text{OLD}} = 0.0$

$y_{\text{OLD}} = y_0$



The main issues is doing better than Euler's method are the ones we have seen time and time again:

- Accuracy: Euler's method is only $O(h)$ accurate. Thus it is nearly impossible to get more than about two significant digits of accuracy with it in the presence of round off error.
- Efficiency: Can we obtain (greater) accuracy with less work?
- Reliability: What is the actual error in a computed approximation?

The error in Euler's method can (remarkably!) be calculated inside the algorithm.

Recall that the local truncation error is:

$$\text{Local Error} = \frac{h^2}{2} [f_t(t,y) + f_y(t,y) f(t,y)] + O(h^3).$$

The global error can be calculated inside the algorithm by adding a few more lines:

• • •

Define the functions

$$FT(T,Y) := f_t(t,y)$$

$$FY(T,Y) := f_y(t,y)$$

$$YPP(T,Y) := FT(T,Y) + FY(T,Y) F(T,Y)$$

• • •

$$ERREST = 0.0$$

• • •

$$LOCERR = \text{ABS}(0.5 * H * H * YPP(T_{\text{NEW}}, Y_{\text{NEW}}))$$

$$ERREST \leftarrow ERREST + LOCERR$$

• • •

PRINT T_{NEW}, Y_{NEW}, ERREST

Exercise

1.9.1. Insert these statements inside the Euler's method algorithm at the correct locations.

When we are solving a system of equations this is too expensive as FY is an N x N Jacobi matrix. Thus, important questions include: How to calculate an estimate of the error more efficiently? How can it be used to improve the accuracy and efficiency of the program as the calculation progresses?

Exercises

1.9.2. Consider the linear pendulum

$$\theta'' + \theta = 0, \theta(0) = \pi/4, \theta'(0) = \pi/4.$$

If this is written as a first order system via $x(t) = \theta(t)$ $y(t) = \theta'(t)$ we obtain:

$$X' = Y, \quad X(0) = \pi/4$$

$$Y' = -X, \quad Y(0) = \pi/4.$$

Take $h = 1/3$ and compute an approximation to $\theta(1)$ and $\theta'(1)$ using Euler's method. Find the error. (Hint: The exact solution takes the form $\theta(t) = C_1 \cos(t) + C_2 \sin(t)$, where $C_{1,2}$ depend on $\theta(0)$ and $\theta'(0)$.)

1.9.3. Repeat problem 1 for the nonlinear pendulum $\theta''(t) + \sin\theta(t) = 0$. (The error cannot be calculated explicitly.)

```
PROGRAM euler
This program solves
x'=f(t,x,y)
y'=g(t,x,y)
for a specific choice-the linear pendulum.
```

```
F(T,X,Y)=Y
G(T,X,Y)=-X
XTRUE(T)=(ATAN(1.0))*(COS(T)+SIN(T))
YTRUE(T)=(ATAN(1.0))*(COS(T)-SIN(T))
```

```
NEXT IS THE FIRST EXECUTABLE STATEMENT.
```

```
T0=0.0
PI=4.0*ATAN(1.0)
X0=PI/4.0
Y0=PI/4.0
TMAX=5.0
H=0.1
```

```
TO ALTER THE PROBLEM, MAKE CHANGES ABOVE HERE.
```

```
TOLD=T0
XOLD=X0
YOLD=Y0
```

```
NOW BEGIN COMPUTING.
```

```
10 XNEW= XOLD+H*F(TOLD,XOLD,YOLD)
YNEW=YOLD+H*G(TOLD,XOLD,YOLD)
TNEW=TOLD+H
ERRX=ABS(XTRUE(TNEW)-XNEW)
ERRY=ABS(YTRUE(TNEW)-YNEW)
PRINT*,"T=",TNEW,"X=",XNEW,"Y=",YNEW
PRINT*,"ERR IN X=",ERRX,"ERRY=",ERRY
IF(TNEW.GE.TMAX) GO TO 20
XOLD=XNEW
YOLD=YNEW
TOLD=TNEW
GO TO 10
20 PRINT*,"MAXIMUM TIME IS NOW REACHED."
STOP
END
```

2. The Numerical Solution of Nonlinear Equations.

2.1 Introduction.

One basic problem in scientific computing is to solve a nonlinear equation:

$$(2.1.1) \quad f(x) = 0 \quad \text{find } x, \text{ where } f : \mathbb{R} \rightarrow \mathbb{R}$$

or system of equations:

$$\begin{aligned} f_1(x_1, \dots, x_n) &= 0 \\ f_2(x_1, \dots, x_n) &= 0 \\ \dots & \\ f_n(x_1, \dots, x_n) &= 0 \end{aligned}, \quad \text{find } x,$$

A closely related problem is *optimization* of a scalar function. If $\phi(x_1, \dots, x_n) : \mathbb{R}^n \rightarrow \mathbb{R}$ find the minimizer of ϕ . Find (x_1^*, \dots, x_n^*) such that

$$\phi(x_1^*, \dots, x_n^*) \leq \phi(x_1, \dots, x_n), \text{ for all } (x_1, \dots, x_n) \in \mathbb{R}^n.$$

This leads to the nonlinear system for the critical points of ϕ

$$\begin{aligned} \phi_{x_1}(x_1^*, \dots, x_n^*) &= 0 \\ \phi_{x_2}(x_1^*, \dots, x_n^*) &= 0 \\ \dots & \\ \phi_{x_n}(x_1^*, \dots, x_n^*) &= 0 \end{aligned}, \quad \text{find } (x_1^*, \dots, x_n^*)$$

We will begin by studying the 1 - d problem (2.1.1).

2.2 Simple Iteration.

One way to attempt to solve $f(x) = 0$ for the root \bar{x} is by simple iteration. We rewrite $f(x) = 0$ as $x = g(x)$ and iterate:

GUESS $x_{\text{OLD}} \quad (= x_0)$

ITERATE $x_{\text{NEW}} = g(x_{\text{OLD}}) \quad (x_{n+1} = g(x_n))$

(until satisfied with x_{NEW})

If not satisfied with x_{NEW} , set $x_{\text{OLD}} = x_{\text{NEW}}$.

Of course, there are infinitely many ways to rewrite $f(x) = 0$ as $x = g(x)$. Here is an example.

Example. The roots of $f(x) \equiv x^2 - 5x + 4 = 0$ are $x = 1$ and $x = 4$. We write this as $x = g(x)$ to attempt to find these roots as follows:

$$5x = x^2 + 4 \quad \text{or} \quad x_{n+1} = \frac{1}{5} (x_n^2 + 4)$$

Take

$$x_0 = 2,$$

$$x_1 = \frac{2^2 + 4}{5} = \frac{8}{5} = 1.6,$$

$$x_2 = \frac{(1.6)^2 + 4}{5} = 1.312,$$

$$x_3 = \dots = 1.144,$$

$$x_4 = \dots = 1.062,$$

$$x_5 = \dots = 1.025,$$

$$x_6 = \dots = 1.010,$$

$$x_7 = \dots = 1.004.$$

This converges to the root $x = 1$ (although slowly!) To obtain the second root $x = 4$, try

$$x_0 = 5:$$

$$x_0 = 5,$$

$$x_1 = \frac{5^2 + 4}{5} = \frac{29}{5} = 5.8,$$

$$x_2 = \dots = 7.528,$$

$$x_4 = \dots = 12.12,$$

which clearly diverges.

There is a graphical interpretation of the simple iteration $x_{n+1} = g(x_n)$.

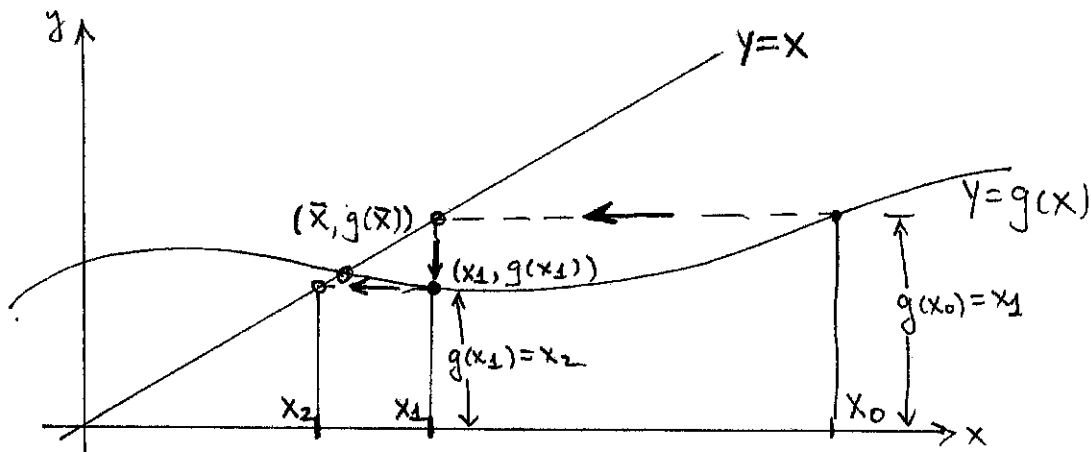


FIGURE: GRAPHICAL INTERPRETATION OF THE SIMPLE ITERATION
 $x_{\text{NEW}} = g(x_{\text{OLD}})$..

If we program a simple iteration the simplest possible program resembles:

INPUT the initial guess x_{OLD} and MAX-ITERATIONS

Counter = 0

(*) Compute $x_{\text{NEW}} = g(x_{\text{OLD}})$

Counter = 0

Test if satisfied with x_{NEW}

(**) If so, print x_{NEW} , Counter and other statistics.

If Counter > MAX-ITERATIONS, then exit program

If not, $x_{\text{OLD}} \leftarrow x_{\text{NEW}}$ and go to (*)

The test for "satisfaction" is interesting to consider because *all* iteration programs contain essentially the same three tests:

Test 1: Too many iterations. If Counter > MAX ITERATIONS the scheme is likely diverging and you should exit the program and signal failure by printing "Program exited after "counter" iterations".

Test 2: Update small. If $x_n \rightarrow \bar{x}$ then x_n is a Cauchy sequence. This implies $|x_n - x_{n+1}| \rightarrow 0$ as $n \rightarrow \infty$. Thus, one common stopping criterion is:

STOP if $|x_{\text{NEW}} - x_{\text{OLD}}| < \text{TOL } 1$

where TOL 1 is pre-set stopping tolerance. (For example, $\text{TOL } 1 = 10^{-6}$). Another version is to stop based on the relative update:

STOP if $|x_{\text{NEW}} - x_{\text{OLD}}|/|x_{\text{NEW}}| < \text{TOL } 1$.

Test 3: Small residual. If we are solving $f(x) = 0$, then we preassign a tolerance TOL 2 and

STOP if $|f(x_{\text{NEW}})| < \text{TOL } 2$.

Naturally, the safest course is to stop only if Test 2 *and* Test 3 are *both* satisfied.

One statistic that's often interesting to track and print is the experimental contraction constant:

$$\alpha_n := \frac{|x_{n+1} - x_n|}{|x_n - x_{n-1}|}.$$

If $\alpha_n = \frac{1}{2}$ (say) then $|x_{n+1} - x_n| = \frac{1}{2}|x_n - x_{n-1}|$ and (it can be shown that)

$$|x_{n+1} - \bar{x}| \sim \frac{1}{2}|x_n - \bar{x}|.$$

In other words x_{n+1} has 1 significant digit base 2 more accuracy than x_n .

Remarks about Fixed Point Problems.

Consider the fixed point problem

$$\bar{x} = g(\bar{x}) + y$$

where we assume $g(0) = 0$ (if not then write $x = (g(x) - g(0)) + (y + g(0))$). The basic iterative method is:

$$(2.2.1) \quad x_{n+1} = g(x_n) + y, \quad x_0 \text{ given.}$$

The convergence of this to \bar{x} is equivalent to the convergence of the telescoping series:

$$\sum_{n=0}^{\infty} (x_{n+1} - x_n)$$

Note that by subtraction

$$x_{n+1} - x_n = g(x_n) - g(x_{n-1})$$

If $g'(\xi)$ is bounded:

$$|g'(\xi)| \leq \alpha$$

then the telescoping series can be dominated by:

$$\sum_{n=0}^{\infty} \alpha^n |x_1 - x_0|$$

as

$$|x_{n+1} - x_n| \leq \alpha |x_n - x_{n-1}| \leq \cdots \leq \alpha^n |x_1 - x_0|.$$

Thus convergence follows easily if $\alpha < 1$. If g is a matrix (more generally, a linear operator)

$$g(\bar{x}) = A\bar{x}$$

then the iteration becomes:

$$x_{n+1} = A^{n+1}x_0 + \sum_{i=0}^n A^i \bar{y}.$$

$\sum_{i=0}^{\infty} A^i \bar{y}$ is called the Neumann series of A , it equals $(I - A)^{-1}\bar{y}$ if $\|A\| < 1$.

The iteration (2.2.1) is frequently called the Picard-Poincaré-Neumann method. However, Liouville used this iteration in 1837 to solve integral equations as did Neumann later

in 1877., Liouville indicates in his paper that this method was well known in his time. In fact, Heron of Alexandria used this method in the 2nd century B.C. to solve $x^2 = 3$ via:

$$x_{n+1} = \frac{1}{2} \left(x_n + \frac{3}{x_n} \right).$$

If $|g'(x)| \leq 1$, (rather than " < 1 ") then sometimes the iteration: $z_0 = y$,

$$z_{n+1} = \frac{n}{n+1} g(z_n) + \frac{1}{n+1} g(y) + y$$

will converge when (2.2.1) fails to converge.

Remark: It is an interesting exercise to relate the z_n 's to the x_n 's.

Convergence of Simple Iteration.

We begin with another example:

Example. Find all \bar{x} such that $f(\bar{x}) = 0$ where $f(x) = e^x - 3x$.

This can be interpreted geometrically. Indeed, the curves $y = 3x$ and $y = e^x$ do intersect twice, at $\bar{x} \cong 0.62$ and at $\bar{x} \cong 1.51$.

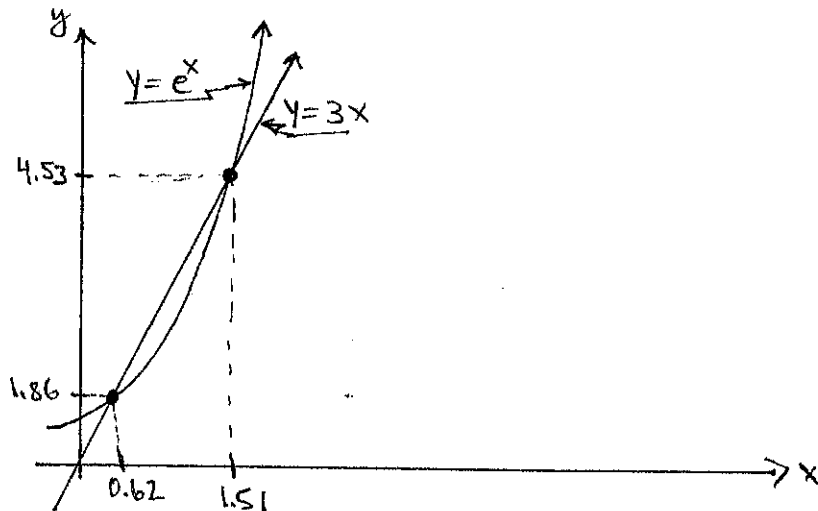


FIGURE: LOCATION OF THE ROOTS OF $f(x) = 0$.

We attempt to find the roots via the iteration $x_{\text{NEW}} = \frac{1}{3} \exp(x_{\text{OLD}})$.

$$x_{\text{NEW}} = \frac{1}{3} e^{x_{\text{OLD}}}$$

$$\begin{aligned}
x_0 &= 0 \cdots x_0 = 2 \\
x_1 &= \frac{e^0}{3} \cong 0.333 \cdots x_1 = \frac{e^2}{3} \cong 2.46 \\
x_2 &= \cdots \cong 0.465 \cdots x_2 = \frac{e^{2.46}}{3} \cong 3.91 \\
x_3 &= \cdots \cong 0.530 \cdots x_3 = \cdots \cong 16.7 \\
x_4 &= \cdots \cong 0.567 \cdots x_4 = \cong 6 \text{ million}
\end{aligned}$$

converges (slowly) to 0.62 diverges very rapidly.

The explanation for when this method will converge is given in the contraction mapping theorem.

Contraction Mapping Theorem.

Theorem 2.2.1. Suppose $g(x)$ is C^1 and there is some interval $I = (a, b)$ such that

- (i) If $x \in I, g(x) \in I$.
- (ii) $|g'(x)| \leq \alpha < 1$, for some α , for all $x \in I$.

Then,

- (1) There is a unique $\bar{x} \in I$ with $\bar{x} = g(\bar{x})$.
- (2) If $x_0 \in I$ then $x_n \in I$ for all n and $x_n \rightarrow \bar{x}$ as $n \rightarrow \infty$.
- (3) The error $|x_n - \bar{x}|$ satisfies

$$|x_n - \bar{x}| \leq \alpha^n |x_0 - \bar{x}|. \quad \square$$

We will prove only the error estimate (3) in the theorem since it is an instructive application of the mean value theorem. Note that one consequence of the theorem is the following global result.

Corollary 2.2.1. Suppose $g(x)$ is C^1 and that there is an $\alpha < 1$ such that

$$|g'(x)| \leq \alpha < 1, \text{ for all } x \in \mathbb{R}.$$

Then, (1) there is a unique fixed point $\bar{x} = g(\bar{x})$.

(2) The simple iteration $x_{n+1} = g(x_n)$ converges for every choice of x_0 .

(3) $|x_n - \bar{x}| \leq \alpha^n |x_0 - \bar{x}|. \quad \square$

In solving nonlinear problems there is a recurring dilemma: global convergence results like this one impose hard to verify and unrealistic conditions on $g(\cdot)$ while local convergence results are easy to verify theoretically but require an initial guess "close enough" to the root. Before proving the contraction mapping theorem, we give the local analogue.

Theorem 2.2.2. Suppose $g(x) \in C^1$ and $g(x)$ had a fixed point \bar{x} , i.e.

$$\bar{x} = g(\bar{x}).$$

Suppose $|g'(x)| < 1$. Then, if x_0 is close enough to x , the iteration

$$x_{n+1} = g(x_n)$$

converges to x . For x_0 close enough to x , it also satisfies the error estimate:

$$|\bar{x} - x_n| \leq \alpha^n |\bar{x} - x_0|, \text{ where } \alpha < 1.$$

Proof of theorem 2.2.2: By continuity of $g'(\cdot)$ if $g'(\bar{x}) < 1$, there is an $\alpha < 1$ and an interval $(a, b) = (\bar{x} - \delta, \bar{x} + \delta)$ (where $\delta > 0$) in which $|g'(x)| \leq \alpha < 1$ for $x \in (a, b)$. The mean value theorem implies that if $x \in (a, b)$ then $g(x) \in (a, b)$. Indeed, $x \in (a, b)$ means $|x - \bar{x}| < \delta$. Now $|g(x) - \bar{x}| = |g(x) - g(\bar{x})| \leq \alpha |x - \bar{x}| \leq \alpha \delta$. The conclusion now follows by taking $I = (a, b)$ in the contraction mapping theorem. \square

Exercises

2.2.1. Consider the iteration used by Heron to calculate $\sqrt{3}$:

$$x_{n+1} = (x_n + 3 / x_n) / 2.$$

Apply the contraction mapping theorem to find an interval $I = (a,b)$ such that if $x_0 \in I$,

$$x_n \rightarrow \sqrt{3}.$$

2.2.2. Consider the simple iteration

$$x_{n+1} = (1 - x_n^2) / 2.$$

What are the fixed points? Find an interval $I = (a,b)$ so that if $x_0 \in I$, $x_n \rightarrow x$, a fixed point.

2.2.3. Suppose $\bar{x} = g(\bar{x})$ is solved by $x_{n+1} = g(x_n)$. If $g'(\bar{x}) = 0$, show that if $|x_0 - \bar{x}|$ is small enough, $x_n \rightarrow \bar{x}$.

2.2.4. What three stopping criteria should be included in all programs which solve some problem by iteration?

Proof of Part (3) of the Contraction Mapping Theorem.

$\bar{x} = g(\bar{x})$ and $x_n = g(x_{n-1})$. Subtracting these two equations and applying the mean value theorem gives:

$$|x_n - \bar{x}| = |g'(\xi_n)| |x_{n-1} - \bar{x}| \leq \alpha |x_{n-1} - \bar{x}|.$$

Further, $|x_{n-1} - \bar{x}| \leq \alpha |x_{n-2} - \bar{x}|$ so we can iterate backwards:

$$|x_n - \bar{x}| \leq \alpha^2 |x_{n-2} - \bar{x}| \leq \alpha^3 |x_{n-3} - \bar{x}| \leq \cdots \leq \alpha^n |x_0 - \bar{x}|. \quad \square$$

Example. We can analyze the simple iteration $x_{\text{NEW}} = \frac{1}{3} \exp(x_{\text{OLD}})$ via this theorem. Indeed, if we ask that $|g'(\xi)| \leq \alpha < 1$, we obtain

$$-1 < \frac{1}{3} e^x < 1.$$

Since e^x is always positive we can simplify this to

$$0 < e^x < 3 \quad \text{or} \quad -\infty < x < \ln 3.$$

So if $I = (-\infty, \ln(3))$ condition (ii) is satisfied. Further, if $x \in I$ then we can work backward to verify (i):

$$-\infty < x < \ln(3) \quad \text{or} \quad 0 < e^x < 3.$$

So that $0 < g(x) < 1$.

Thus, if $x \in I$, $g(x) \in (0, 1) \subseteq I$, and condition (i) holds. \square

2.3 Newton's Method.

We wish to solve $f(\bar{x}) = 0$ for \bar{x} . Simple iteration can converge very slowly. Newton's method typically converge much more rapidly. In fact, we will show that the "approximate" contraction constant α_n in Newton's method satisfies $\alpha_n \rightarrow 0$ as $n \rightarrow \infty$. First, let us review the geometric idea behind the method.

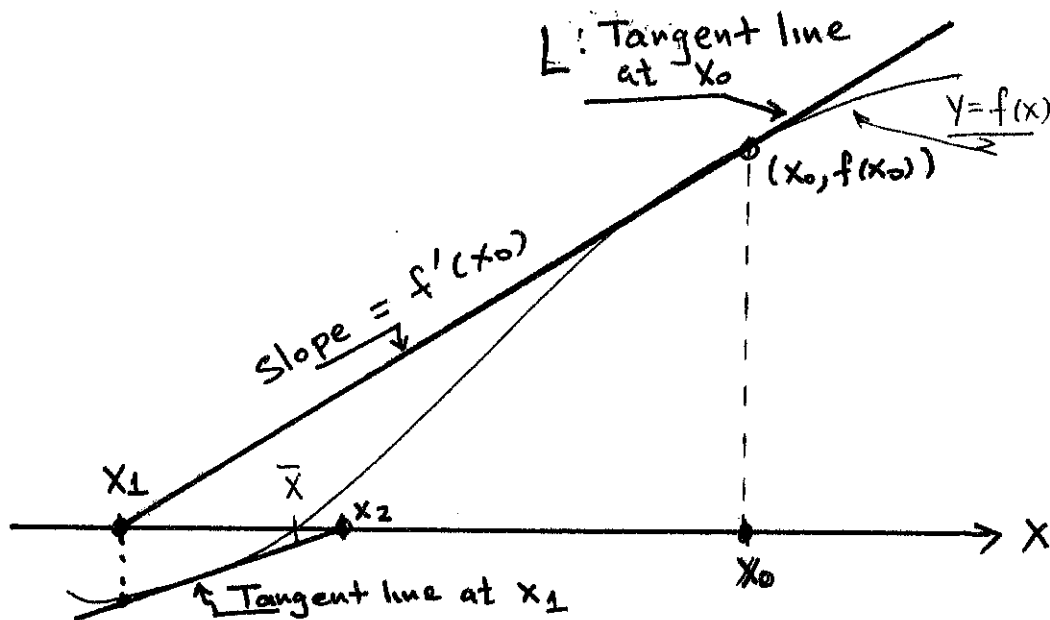


FIGURE: GOING DOWN THE TANGENT LINES.

Geometric Idea:

The equation of the tangent line L at $(x_i, f(x_i))$ is:

$$y - f(x_i) = f'(x_i)(x - x_i).$$

To find its root set $y = 0$ and solve. When $y = 0, x = x_{i+1}$ or:

$$x_{i+1} = x_i - f(x_i)/f'(x_i)$$

This is precisely the Newton iteration:

$$(2.3.1) \quad \boxed{x_{i+1} = x_i - f(x_i)/f'(x_i)}.$$

This is an iteration of the form $x_{i+1} = g(x_i)$

Example. $f(x) = e^x - 3x = 0$ with $x_0 = -1, 0, +1, +2$. This takes 38 iterations to converge with *simple iteration*. However,

COMPUTER EXERCISE. Use the Newton method program to verify this claim of only 4-5 Newton steps.

4 or 5 iterations produces both roots with Newton's method.

Algorithm Guess x_0 (intelligently). \triangleright HERE \triangleleft

Compute $f(x_i), f'(x_i)$

Is $f'(x_i) = 0$? (i.e. $|f'(x_i)| < \epsilon_1$)? - YES

If YES, signal failure.

If NO, proceed:

$$x_{i+1} = x_i - f(x_i)/f'(x_i)$$

TEST if: (a) $|f(x_{i+1})| < \epsilon_2$

(b) Fixed of iterations

(c) $|x_{i+1} - x_i| < \epsilon$,

CONTINUE if not true.

Convergence of Newton's Method.

Theorem of Global Convergence.
Theorem 2.3.1. (Newton-Baluev) Assume

- (i) $f: \mathbb{R} \rightarrow \mathbb{R}$ is C^1 and convex;
- (ii) $f'(x) > 0$ for all x
- (iii) $f(x) = 0$ has the solution $x = \bar{x}$.

Then \bar{x} is unique and the Newton iterates

$$x^{k+1} = x^k - f(x^k)/f'(x^k)$$

converge to \bar{x} for any x_0 . Moreover

$$\bar{x} \leq x^{k+1} \leq x^k \quad k = 1, 2, \dots$$

a generalization of

This theorem is a "global" convergence result and it holds if $f: \mathbb{R}^n \rightarrow \mathbb{R}^n$ as well. There is also very famous and important local convergence theorem called the "Newton-Kantorovich Theorem". We can't give it here in its full generality and power. Nevertheless, local convergence of Newton's method in 1-d is easy to prove using Theorem 2.2.2.

Theorem 2.3.2. [Local Convergence of Newton's Method].

Let $f \in C^2$ and suppose \bar{x} is a simple root of $f(x)$. (Specifically, $f(\bar{x}) = 0$ but, $f'(\bar{x}) \neq 0$.) Then, if x_0 is close enough to \bar{x} , the Newton iterates converge to \bar{x} .

Proof Newton's method is a simple iteration with $g(x) = x - f(x) / f'(x)$. Thus, Theorem

2.2.2. can be applied. Checking $g'(\bar{x})$ shows:

$$g'(x) \Big|_{x=\bar{x}} = 1 - \frac{f'(\bar{x})f'(\bar{x}) - f''(\bar{x})f(\bar{x})}{(f'(\bar{x}))^2} = 1 - \frac{f'(\bar{x})^2}{f'(\bar{x})^2} = 0 (< 1).$$

Since $|g'(\bar{x})| = 0 < 1$, local convergence follows immediately.

Exercises

2.3.1. $f(x) = xe^x$ has a root at $x = 0$. Take $x_0 = 1$ and do five steps of Newton's method. Calculate and tabulate for each step: n (the step number), $|f(x_n)|$ (residual), $|x_{n+1} - x_n|$ (the update), $\alpha_n = |x_{n+1} - x_n| / |x_n - x_{n-1}|$ (the contraction constant).

What patterns do you see?

2.3.2. Write Newton's method as a pseudo code algorithm.

2.3.3. \bar{x} is a double root of $f(\bar{x}) = 0$ if $f(\bar{x}) = 0$ and $f'(\bar{x}) = 0$ but $f''(\bar{x}) \neq 0$. Construct a function $f(x)$ with a known double root \bar{x} . Repeat problem 1 for this function.

Pitfalls of Newton's Method.

The diagrams below indicate some commonly occurring pitfalls in using Newton's method. It's good to be aware of these rare possibilities.

Pitfall 1: No real root. If $f(x)$ is a real function when x is real then all the Newton iterates will remain real. Thus, to find *complex* roots we must start with a *complex* initial guess.

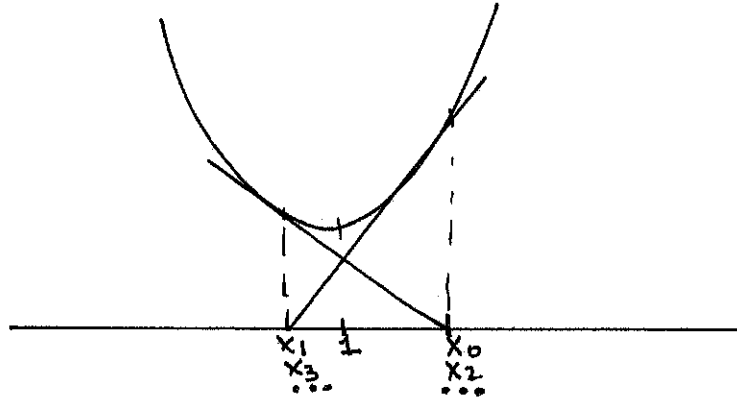
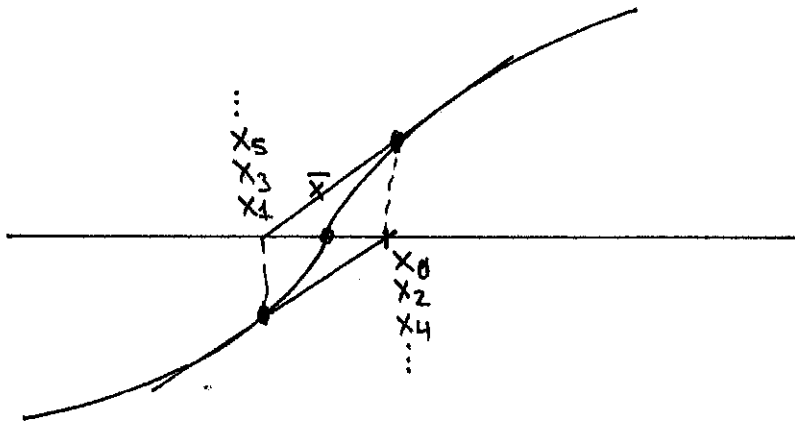


FIGURE: $f(x) = (x - 1)^2 + 1 = 0$.

Pitfall 2: $f''(\bar{x}) = 0$. In this case, the method can be stuck in an infinite loop near the root.

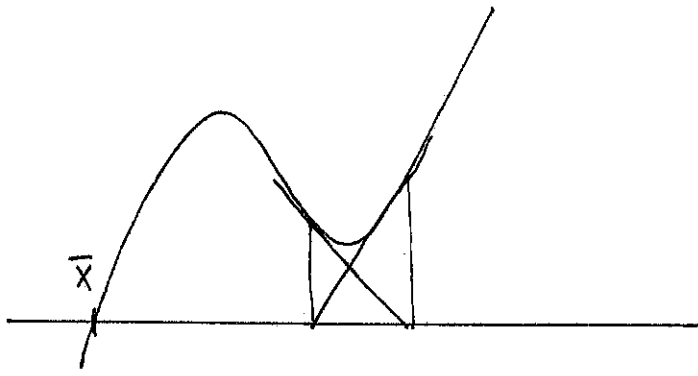


Pitfall 3: Bad initial guess Some other portion of the function $f(x)$ can "trap" the iteration.

Order of Convergence.

If we have the simple iteration

$$x_{n+1} = g(x_n), \quad \bar{x} = g(\bar{x})$$



PITFALL 3! BAD INITIAL GUESS!

we then have

$$x_{n+1} - \bar{x} = g'(\xi_n)(x_n - \bar{x})$$

or, with $e_n = n^{\text{th}}$ error = $x_n - \bar{x}$,

$$e_{n+1} = g'(\xi_n) e_n.$$

As $x_n \rightarrow \bar{x}$, $\xi_n \rightarrow \bar{x}$ also so we can say that as $x_n \rightarrow \bar{x}$

$$e_{n+1} \cong g'(\bar{x})e_n.$$

If g'' is continuous, we can write also

$$e_{n+1} \cong g'(\bar{x})e_n + \frac{1}{2}g''(\bar{x})e_n^2 + \dots$$

Definition. The order of convergence of $x_{n+1} = g(x_n)$ is the order of the first nonzero derivative of g at the root \bar{x} .

Alternatively:

Definition. Suppose

$$\lim_{n \rightarrow \infty} \frac{|e_{n+1}|}{|e_n|^p} \leq C$$

then the method is of order p . C is called the asymptotic error constant.

Examples. 1. $f(x) = e^x - 3x = 0$ rewritten as:

$$x = e^x/3 = g(x)$$

then $g'(x) = \frac{e^x}{3}$ so this first order since $g' \neq 0$.

2. $f(x) = x^2 - 5x + 4 = 0$, 2 roots: 1, 4. Consider

$$x = g(x) \equiv \frac{x^2 - 4}{5}$$

so $g'(x) = \frac{2x}{5}$ which is not 0 at $x = 1$, and $x = 4$ so this is first order.

Convergence Order of Newton's Method.

Consider Newton's method: $x_{i+1} = g(x_i)$ where $g(x) = x - f(x)/f'(x)$

Differentiate $g(x)$:

$$g'(x) = 1 - \frac{f'(x)f'(x) - f(x)f''(x)}{[f'(x)]^2}$$

at $x = \bar{x}$ and using $f(\bar{x}) = 0$ we have:

Case 1: Simple root (i.e. $f'(\bar{x}) \neq 0$), $g'(\bar{x}) = 0$ and $g''(\bar{x}) \neq 0$. Thus Newton's method is *second order* at a simple root.

Case 2: Multiple root (i.e. $f'(\bar{x}) = 0$). Here $g'(\bar{x}) = \frac{0}{0}$, (formally) L'Hospital's rule reveals $g'(\bar{x}) \sim \frac{1}{2}$ so Newton's Method is *first order* at a multiple root.

The Secant Method The secant method is *almost* as fast as Newton's method and does *not* need $f'(x)$ to work.

Notes about the secant method:

1. $f(x_1), f(x_0)$ do not have to be of opposite sign.
2. This converges faster than bisection and false position but more slowly than Newton's method.
3. The secant method generalities to systems.
4. The ratio:

$$\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} \approx \frac{1}{f'(x_n)}$$

The secant method needs two initial guesses. It replaces the function with the secant line through the previous two iterates. The root of this line is the new approximation (see the next figure). Mathematically, it takes the form:

$$x_0, x_1, \text{ given,}$$
$$x_{n+1} = x_n - f(x_n) / \left(\frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}} \right).$$

Computer Exercise

Consider the short Newton method program you have previously used. Convert it to the secant method and repeat the experiment you did earlier. Compare the results of the two methods.

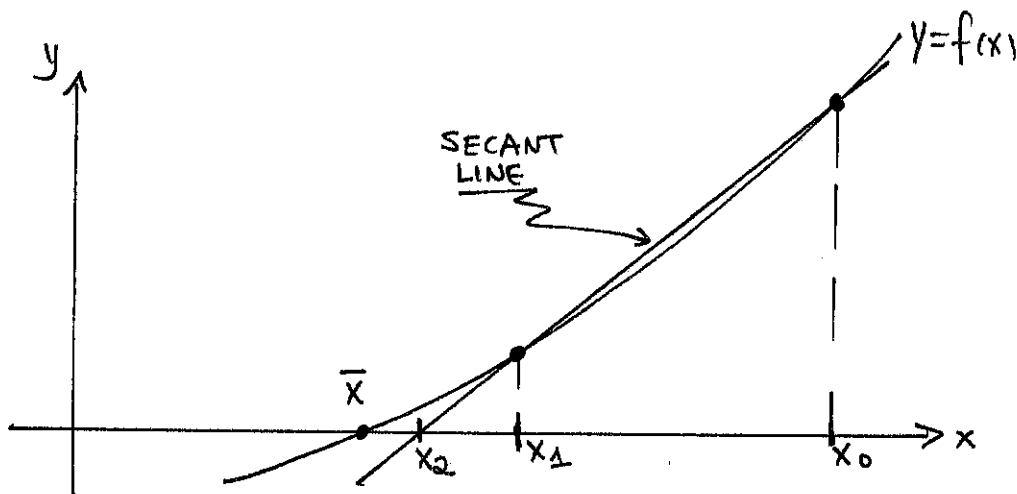


FIGURE: THE SECANT METHOD, $x_{n+1} := x_n - f(x_n) \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}$.

so that this can be thought of as replacing $f'(x_n)$ by a difference quotient.

Error in the Secant Method

The error in the secant method satisfies:

$$e_{n+1} \approx -\frac{1}{2} \frac{f''(c)}{f'(c)} e_n e_{n-1}$$

This translates to an order of convergence between 1 and 2, about 1.6.

<u>METHOD</u>	<u>ORDER</u>
Newton	2 at simple root 1 at even order root
Secant	1.618 (actually, it is <u>exactly</u> the golden mean)
Simple Iteration	1

2.4 Nonlinear Simultaneous Equations.

Suppose that we wish to solve

$$(2.4.1) \quad \begin{cases} f_1(x, y) = 0, \\ f_2(x, y) = 0. \end{cases}$$

There are many possibilities as the two curves $f_1(x, y) = 0$ and $f_2(x, y) = 0$ can intersect in virtually any manner. Recall the definition of Jacobian, and Jacobi matrix.

a vector function of a vector variables

Definition 2.4.1. Let $\vec{F}(x_1, \dots, x_n) = (f_1, f_2, \dots, f_n)$ with $f_i = f_i(x_1, \dots, x_n) \in C^1$. Then, the Jacobi matrix of \vec{F} , written \vec{F}' , is defined to be the $n \times n$ matrix

$$\vec{F}'(\vec{x}) = \left(\frac{\partial f_i}{\partial x_j} \right) \quad i, j = 1, \dots, n.$$

The Jacobian of \vec{F} is defined to be $J = \det(\vec{F}')$.

Exercise: $f_1(x, y) = x^2 + y^2$, $f_2(x, y) = x + 3xy$ find \vec{F}' .

Exercises: (1) Suppose $\vec{F}(x_1, \dots, x_n) = \text{grad } \phi(x_1, \dots, x_n)$. Show that $\vec{F}'(\vec{x})$ is symmetric.

(2) Show that if $J(\vec{x}_n) \neq 0$ then $\vec{F}'(\vec{x}_n)$ is an invertible matrix.

Newton's method for the system (2.4.1) then reads

Guess x_0, y_0

Given x_n, y_n , let $x_{n+1} = x_n + \Delta x, y_{n+1} = y_n + \Delta y$

Where, with $J = J(f_1(x_n, y_n), f_2(x_n, y_n)) \neq 0$,

$$\vec{F}'(x_n, y_n) \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = - \begin{pmatrix} f_1(x_n, y_n) \\ f_2(x_n, y_n) \end{pmatrix}.$$

More generally, for $n \times n$ nonlinear systems we can write Newton's method for \vec{F} as

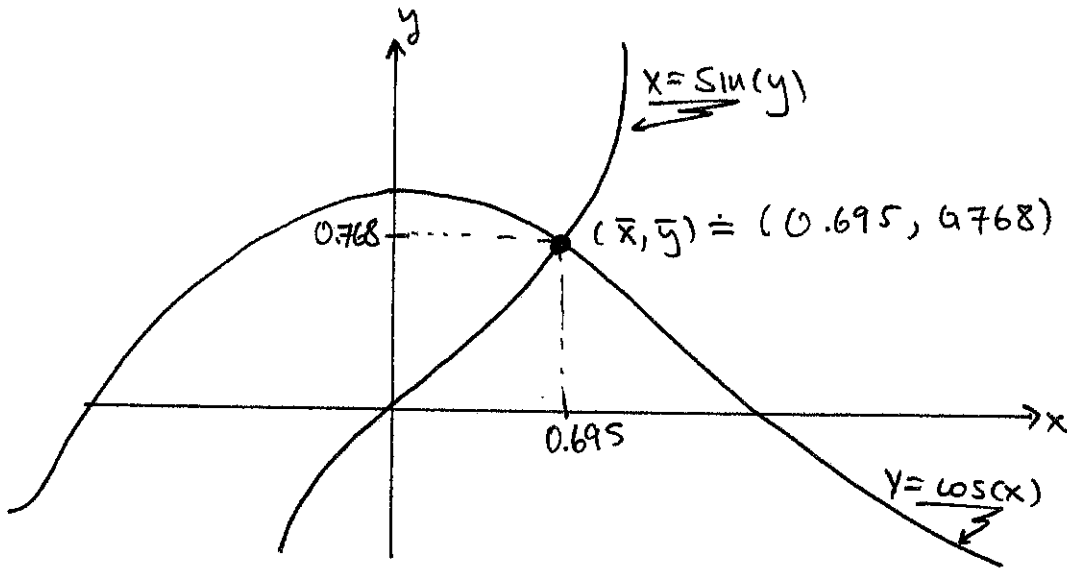
$$\underbrace{\vec{F}'(x_n)}_{n \times n \text{ Jacobi matrix}} (\vec{x}_{n+1} - \vec{x}_n) = -\vec{F}(x_n).$$

Note that at each iteration we must solve an $n \times n$ linear system for the update, $\Delta x = \vec{x}_{n+1} - \vec{x}_n$.

Example. $y = \cos x, x = \sin y$, solve for (x, y) using Newton's method.

Solution:

$$\begin{cases} f_1(x, y) \equiv \cos x - y = 0 \\ f_2(x, y) \equiv x - \sin y = 0 \end{cases}$$



The next figure shows the approximate location of the root:

We have

$$\begin{aligned} \frac{\partial f_1}{\partial x} &= -\sin x, & \frac{\partial f_2}{\partial x} &= 1 \\ \frac{\partial f_1}{\partial y} &= -1, & \frac{\partial f_2}{\partial y} &= -\cos y \end{aligned}$$

so that

$$\bar{F}'(x, y) = \begin{pmatrix} -\sin x & -1 \\ 1 & -\cos y \end{pmatrix}$$

and $J = \sin x \cos y + 1 \neq 0$, for (x, y) in the box $0 \leq x \leq 1$, $0 \leq y \leq 1$.

Given (x_n, y_n) we solve

$$\begin{pmatrix} -\sin x_n & -1 \\ 1 & -\cos y_n \end{pmatrix} \begin{pmatrix} \Delta x_n \\ \Delta y_n \end{pmatrix} = - \begin{pmatrix} \cos(x_n) - y_n \\ x_n - \sin(y_n) \end{pmatrix}$$

and then $x_{n+1} = x_n + \Delta x_n$, $y_{n+1} = y_n + \Delta y_n$. This gives the following results:

n	x_n	y_n
0	1.000	1.00
1	0.72027285	0.77568458
2	0.69495215	0.76832706
3	0.69481970	0.76816915
4	0.69481969	0.76816915
5	0.69481969	0.76816916

Exercise: Repeat this example for

$y = x + \sin y$ $3 \sin(x) - y = 0$

Problems with Newton's Method in \mathbb{R}^n .

- (Expense) At each iteration we must form an $n \times n$ matrix $\vec{F}'(\vec{x}_n)$ and solve a *NEW* linear system with it.
- (Pitfalls) There are a lot more possible pitfalls in \mathbb{R}^n than in \mathbb{R}^1 . Thus, we have to be very careful when we are away from the roots.

Some Solutions to these Problems

Solution 1: (Modified Newton's Method) Once we are near the root \vec{x}_n doesn't change much so $\vec{F}'(\vec{x}_n)$ doesn't either. Thus, we may "freeze" $\vec{F}'(\vec{x}_n)$ for a few iterates at a time (*near the root*): (*modified Newton Method*).

$$\vec{F}'(\vec{x}_n)(\vec{x}_{n+k+1} - \vec{x}_{n+k}) = -\vec{F}(\vec{x}_{n+k}).$$

Solution 2: (Damped Newton Method) We monitor $|\vec{F}(\vec{x}_{n+1})|$ and refuse to accept \vec{x}_{n+1} as the new approximation when $|\vec{F}(\vec{x}_{n+1})| \geq |\vec{F}(\vec{x}_n)|$.

Algorithm: Damped Newton Method for $\vec{F}(\vec{x}) = 0$.

Guess: \vec{x}_0 .

For $n = 0, 1, 2, \dots$, until satisfied, do:

$$\vec{\Delta x}_n := -\vec{F}'(\vec{x}_n)^{-1} \vec{F}(\vec{x}_n)$$

$$(*) \quad \begin{cases} \text{Find smallest } j \text{ such that} \\ |\vec{F}(\vec{x}_m + \vec{\Delta x}_m / 2^j)| < |\vec{F}(\vec{x}_m)| \\ \vec{x}_{m+1} := \vec{x}_m + \vec{\Delta x}_m / 2^j \end{cases}$$

It is possible to show that, in theory at least, (*) can always be executed. Of course, if you program it, you specify a j_{\max} (= 10 for example) and exit if $j > j_{\max}$. \square

COMPUTER EXERCISE

Take the Newton method program you have used earlier. Modify it to include a damping step. Now try ~~some~~ to construct some really difficult nonlinear equations to solve. Study the number of damping steps and convergence with and without damping. Does the residual really decrease monotonically?

MORE

1. Secant Method in \mathbb{R}^2 and \mathbb{R}^n
2. Broyden's Method
3. GLOBALIZATION STRATEGIES
 - DAMPING

2.5. Globalization Strategies: Homotopy Methods.

The Example of

Suppose we wish to solve n nonlinear equations in n variables

$$\left. \begin{array}{l} f_1(x_1, \dots, x_n) = 0 \\ \dots \\ f_n(x_1, \dots, x_n) = 0 \end{array} \right\} \Leftrightarrow \vec{F}(\vec{x}) = \vec{0}.$$

One difficult question is related to the lack of global convergence of Newton's method. This is, of course, related to the question of finding good initial guesses.

Homotopy and continuation methods are one globalization strategy for Newton's method. In the most simple context, it works as follows. A nonlinear problem

$$\vec{G}(\vec{x}) = \vec{0}, \quad n \text{ equations in } n \text{ variables,}$$

is picked which is "easy" to solve and which is "close" to $\vec{F}(\vec{x}) = \vec{0}$. These two problems are connected by a curve (a "homotopy"), such as:

$$\vec{H}(t, \vec{x}) = t \vec{F}(\vec{x}) + (1-t) \vec{G}(\vec{x}) = 0.$$

Note that

- $\vec{H}(t, \vec{x}) = 0$ is n equations in $(n+1)$ variables x_1, \dots, x_n and t so the solution is a curve

$$\vec{x}(t) \quad 0 \leq t \leq 1.$$

- When $t=0$ it reduces to $\vec{G}(\vec{x})=0$
so

$$\vec{x}(0) = \text{solution of } : \vec{G}(\vec{x})=0$$

- When $t=1$ it reduces to $\vec{F}(\vec{x})=0$
so

$$\vec{x}(1) = \text{solution of } : \vec{F}(\vec{x})=0.$$

One simple strategy is as follows:

Solve $\vec{G}(\vec{x}) = 0$ for $\vec{x}_{old}, t_{old} = 0$

Pick Δt

$$(*) \quad t_{new} = t_{old} + \Delta t$$

Solve

$$\vec{H}(t_{new}, \vec{x}_{new}) = 0$$

using Newton's method with \vec{x}_{old} as initial guess.

If NM does not converge, reduce Δt
and return to (*)

If NM converges, $\vec{x}_{old} \leftarrow \vec{x}_{new}$ and return to (*)

STOP when $t_{new} = 1$.

There are many possible refinements to this basic strategy.

Techniques from O.D.E. methods can even be used as follows. If the total derivative of $H(t, x(t)) = 0$ is taken, we obtain:

$$\underbrace{\vec{H}_t(t, x(t))}_{n\text{-vector}} + \underbrace{\vec{H}_x(t, x(t))}_{n \times n \text{ Jacobian matrix}} \cdot \vec{x}'(t) = 0,$$

or: Find $x(t)$ where:

$$\begin{cases} x'(t) = - \left(H_x(t, x(t)) \right)^{-1} \vec{H}_t(t, x(t)). \\ x(0) = \text{solution of: } \vec{G}(\vec{x}) = 0. \end{cases}$$

This naturally still involves the inversion of an $n \times n$ matrix or the solution of an $n \times n$ linear system.

Exercise 2.5.1. Show that if we solve the O.D.E. above with Euler's method with step size 1, we get exactly Newton's method!

3. Numerical Differentiation and Integration.

3.1 Introduction.

Often functions are "known" as a table of values rather than a closed form expression. This table of values might be known from experiments or observations. It might also be the result of a long computer program where "x" represents some input parameter and "y" the result of 20,000 lines of FORTRAN calculation. In either case, the problem is: given data

$$(3.1.1) \quad (x_0, y_0), (x_1, y_1), \dots, (x_n, y_n) \quad x_j \in [a, b]$$

calculate an approximation to

$$y'(\bar{x}), \bar{x} \in [a, b], \text{ or } \int_a^b y(x) dx,$$

from this data.

One basic approach is to simply interpolate the data (3.1.1) with a polynomial $p_n(x)$

$$(3.1.2) \quad p_n(x) = y_0 \ell_0(x) + y_1 \ell_1(x) + \dots + y_n \ell_n(x).$$

where $\ell_j(x) = \prod_{\substack{k=1 \\ k \neq j}}^n \frac{x - x_k}{x_j - x_k}$ are the Lagrange functions, and then just differentiating or integrating $p_n(x)$.

In fact, this is an example of a general procedure for approximating "linear functions".

Definition. L is a linear functional acting on continuous functions, if $L(f)$ is a real number and for any two continuous functions $f(x), g(x)$ and any real number α :

$$L(f + g) = L(f) + L(g) \text{ and } L(\alpha f) = \alpha L(f).$$

There are numerous examples of linear functionals in applied mathematics:

- differentiation: $L(f) = f'(a)$ for $f \in C^1(\mathbb{R})$.

- Integration: $L(f) = \int_a^b f(x) dx$
- limits: $L(f) = \lim_{h \rightarrow 0} f(a+h)$
- Fourier sine coefficients: $L_N(f) = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(x) \sin(Nx) dx$

Assuming we can calculate $L(p(x))$ for any polynomial $p(x)$, we may proceed as follows:

Given $y_j = f(x_j)$ in the table (3.1.1),

Approximate $f(x)$ by $p_n(x)$ given by (3.1.2), and from this, approximate

$$\begin{aligned} L(f) &\cong L(p_n) = L(y_0 \ell_0(x) + \cdots + y_n \ell_n(x)) = \\ &= \text{(using linearity)} = y_0 L(\ell_0(x)) + \cdots + y_n L(\ell_n(x)). \end{aligned}$$

The “weights” $w_j := L(\ell_j(x))$ are calculated and tabulated. The approximation formula

$$(3.1.3) \quad L(f) \cong w_0 f(x_0) + w_1 f(x_1) + \cdots + w_n f(x_n)$$

results. $L(f)$ is approximated by a weighted sum of function values of f at the points x_0, \dots, x_n .

The weights typically

$$w_j = L(\ell_j(x))$$

are computed and tabulated. However, there is another way to think of how these are determined. To see this note that if $f(x)$ is a polynomial of degree $\leq n$ (written $f \in \Pi_n$) then it is ^{its} own interpolant:

$$f(x) = p_n(x) = f(x_0) \ell_0(x) + \cdots + f(x_n) \ell_n(x), \quad f \in \Pi_n.$$

Thus,

$$L(f) = L(p_n) = w_0 f(x_0) + \cdots + w_n f(x_n), \quad f \in \Pi_n$$

In other words, the approximation (3.1.3) is *exact* on Π_n .

Exercises

3.1.1. Consider the central difference approximation to $Lf := f'(a)$:

$$Lf \doteq Df(a) := (f(a+h) - f(a-h)) / 2h.$$

Verify that this is exact on $1, x$ and x^2 .

3.1.2. Consider a one-sided approximation to $f'(a) = L(f)$:

$$L(f) = f'(a) \doteq Df := w_1 f(a) + w_2 f(a+h) + w_3 f(a+2h).$$

(a) Require this to be exact on $1, x$ and x^2 and derive the resulting 3×3 linear system for w_1, w_2 and w_3 .

(b) Solve the system for w_1, w_2 and w_3 . By expanding the error in a Taylor series in h , verify its $O(h^2)$ accuracy.

3.1.3. Find an approximation to $f'''(a)$ which is ~~is~~ compact (i.e., uses as few points as possible). Calculate its error using a Taylor series expansion. Upon what degree polynomials is it exact?

We have already seen that in practical calculations we cannot let $h \rightarrow 0$ since roundoff error will make the total error blow up. In many practical calculations there is (worse) noise or other errors inherent in the data. In these settings, numerical differentiation *must* be *preceded* by a procedure known as “data smoothing” to filter out some of the noise. We shall discuss some approaches to that later.

3.3 Numerical Integration: Basic Newton-Cotes Rules.

The idea of the introduction, applied to calculating

$$\int_a^b f(x) dx$$

is to pick $n + 1$ points on $[a, b]$ x_0, x_1, \dots, x_n interpolate $f(x)$ at those points by $p_n(x) \in \Pi_n[a, b]$ and

. approximate .

$$\int_a^b f(x) dx \cong \int_a^b p_n(x) dx.$$

This procedure does *NOT* work well without one additional refinement. If $f(x)$ is not *very* smooth (i.e., $f(x) \notin C^\infty(a, b)$) then it is quite possible that $p_n(x) \not\rightarrow f(x)$ as $n \rightarrow \infty$.

The refinement needed is to use this idea inside a “composite” formula: the interval $[a, b]$ is broken into small subintervals and this idea applied, for fixed polynomial degree, on each subinterval.

The basic idea of a composite formula is to pick the meshpoints on $[a, b]$:

$$a = x_0 < x_1 < x_2 < \dots < x_n = b$$

and divide the integral accordingly:

$$\int_a^b f(x) dx = \sum_{j=0}^{n-1} \int_{x_j}^{x_{j+1}} f(x) dx.$$

Each sub-integral $\int_{x_j}^{x_{j+1}} f(x) dx$ is approximated by a weighted sum of function values. Methods differ only on how this weighted sum is selected.

Example 1. : (The trapezoid rule) If we seek a two point approximation on each subinterval:

$$\int_{x_j}^{x_{j+1}} f(x) dx \cong w_j f(x_j) + w_{j+1} f(x_{j+1}),$$

the coefficients w_j, w_{j+1} are determined by requiring this be *exact* on Π_1 (equivalently, $f(x) = 1$ and $f(x) = x$.) This gives the 2×2 system:

$$\begin{aligned} \int_{x_j}^{x_{j+1}} 1 dx &= (x_{j+1} - x_j) = w_j \cdot 1 + w_{j+1} \cdot 1 \\ \int_{x_j}^{x_{j+1}} x dx &= \frac{x_{j+1}^2}{2} - \frac{x_j^2}{2} = w_j x_j + w_{j+1} x_{j+1}. \end{aligned}$$

The solution of the system is $w_j = w_{j+1} = \frac{x_{j+1} - x_j}{2}$. This gives the rule:

$$\int_{x_j}^{x_{j+1}} f(x) dx \cong \left(\frac{x_{j+1} - x_j}{2} \right) f(x_j) + \left(\frac{x_{j+1} - x_j}{2} \right) f(x_{j+1})$$

and the approximation

$$(3.3.1) \quad \int_a^b f(x) dx \cong \sum_{j=0}^{n-1} \left[\left(\frac{x_{j+1} - x_j}{2} \right) f(x_j) + \left(\frac{x_{j+1} - x_j}{2} \right) f(x_{j+1}) \right].$$

3.3.1.

Exercise: If the points are equally spaced, $h = x_{j+1} - x_j$ for all j , show that this reduces to:

$$\int_a^b f(x) dx \cong h \left[\frac{1}{2} f(x_0) + f(x_1) + f(x_2) + \cdots + f(x_{n-1}) + \frac{1}{2} f(x_n) \right].$$

Theorem 3.3.1. The local error on the j^{th} subinterval is $O(x_{j+1} - x_j)^3$:

$$\int_{x_j}^{x_{j+1}} f(x) dx - \left[(x_{j+1} - x_j) \frac{f(x_j) + f(x_{j+1})}{2} \right] = \frac{(x_{j+1} - x_j)^3}{12} f''(\xi_j), \text{ for some } \xi_j \in (x_j, x_{j+1}).$$

The global error is $O(h^2)$, if $h = \max_j (x_{j+1} - x_j)$,

$$\left| \int_a^b f(x) dx - \sum_{j=0}^{n-1} (x_{j+1} - x_j) \left(\frac{f(x_j) + f(x_{j+1})}{2} \right) \right| \leq \frac{b-a}{12} h^2 \max_{a < x < b} |f''(x)|.$$

Proof. The first follows by integrating from x_j to x_{j+1} the error formula for linear interpolation.

The second, global error bound follows by summation:

$$\begin{aligned}
 |\text{Total error}| &\leq \sum_{j=0}^{n-1} |\text{Error on } (x_j, x_{j+1})| \leq (\text{inserting the first bound}) \\
 &\leq \sum_{j=0}^{n-1} \frac{1}{12} (x_{j+1} - x_j)^3 f''(\xi_j) \\
 &\leq \max_{a < x < b} |f''(x)| \frac{\max_j (x_{j+1} - x_j)^2}{12} \sum_{j=0}^{n-1} (x_{j+1} - x_j) \\
 &\leq \frac{h^2}{12} (b - a) \max_{a < x < b} |f''(x)|. \quad \square
 \end{aligned}$$

Example. : (Simpson's Rule) As before, we decompose

$$\int_a^b f(x) dx = \sum_{j=0}^{n-1} \int_{x_j}^{x_{j+1}} f(x) dx.$$

The trapezoid rule arises as a *two*-point formula on (x_j, x_{j+1}) , exact on the *two* functions $f(x) = 1$ and $f(x) = x$. The path to more accuracy is now clear: Simpson's rule (the next step) is a *three*-point formula exact on $1, x$ and x^2 .

Let us seek an approximation:

$$\int_{x_j}^{x_{j+1}} f(x) dx \cong w_j f(x_j) + w_{j+1/2} f(x_{j+1/2}) + w_{j+1} f(x_{j+1}).$$

where $x_{j+1/2}$ is the mid-point: $x_{j+1/2} := (x_j + x_{j+1})/2$. If this is *exact* on $\Pi_2 = \text{span}\{1, x, x^2\}$ we have:

$$\begin{aligned}
 \int_{x_j}^{x_{j+1}} 1 \, dx &= (x_{j+1} - x_j) = w_j \cdot 1 + w_{j+1/2} \cdot 1 + w_{j+1} \cdot 1, \\
 \int_{x_j}^{x_{j+1}} x \, dx &= \frac{x_{j+1}^2}{2} - \frac{x_j^2}{2} = w_j x_j + w_{j+1/2} x_{j+1/2} + w_{j+1} x_{j+1} \\
 \int_{x_j}^{x_{j+1}} x^2 \, dx &= \frac{x_{j+1}^3}{3} - \frac{x_j^3}{3} = w_j x_j^2 + w_{j+1/2} x_{j+1/2}^2 + w_{j+1} x_{j+1}^2.
 \end{aligned}$$

This is a 3×3 linear system for the 3 weights $w_j, w_{j+1/2}$ and w_{j+1} . It's solution is:

$$w_j = \frac{(x_{j+1} - x_j)}{6}, \quad w_{j+1/2} = \frac{4}{6} (x_{j+1} - x_j), \quad w_{j+1} = \frac{1}{6} (x_{j+1} - x_j).$$

The corresponding integration rules (Simpson's rule) are:

$$\int_{x_j}^{x_{j+1}} f(x) dx \cong \frac{1}{6} (x_{j+1} - x_j) f(x_j) + \frac{4}{6} (x_{j+1} - x_j) f(x_{j+1/2}) + \frac{1}{6} (x_{j+1} - x_j) f(x_{j+1}),$$

and

$$(3.3.2) \quad \int_a^b f(x) dx \cong \sum_{j=0}^{n-1} \frac{1}{6} (x_{j+1} - x_j) \left[f(x_j) + 4f\left(\frac{x_j + x_{j+1}}{2}\right) + f(x_{j+1}) \right].$$

This formula is also (by chance) exact on x^3 so it contains hidden accuracy. It's local error on (x_j, x_{j+1}) is $O(h^5)$ and its global error is $O(h^4)$.

There is another, equivalent, way to derive Simpson's rule: Interpolate $y = f(x)$ at the points:

$$(x_j, f(x_j)), (x_{j+1/2}, f(x_{j+1/2})) \text{ and } (x_{j+1}, f(x_{j+1}))$$

to give a quadratic polynomial $p_2(x)$. In each (x_j, x_{j+1}) we approximate

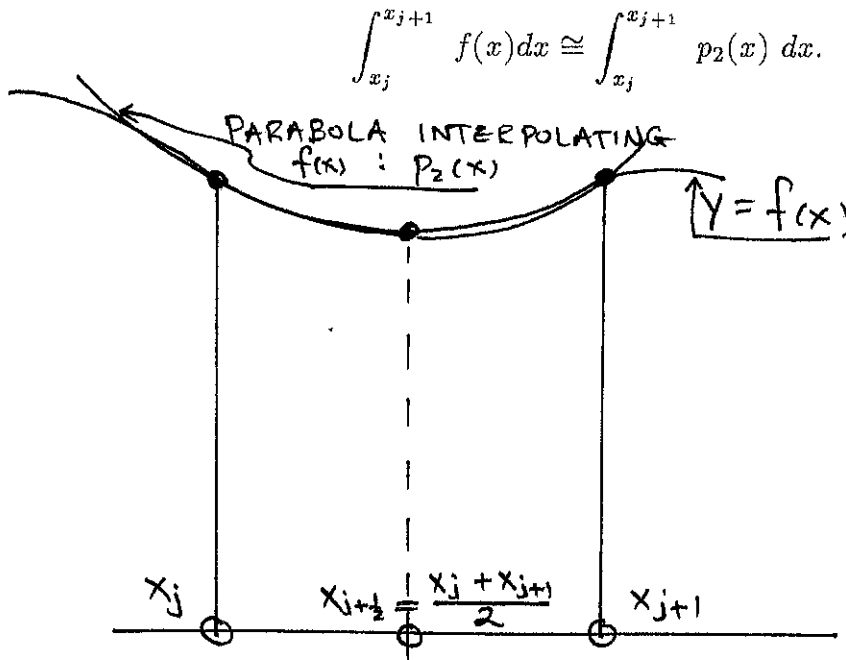


FIGURE: A GEOMETRIC INTERPOLATION OF SIMPSON'S RULE..

Theorem 3.3.2. Let $h = \max_j (x_{j+1} - x_j)$. The global error in Simpson's rule is $O(h^4)$:

$$\left| \int_a^b f(x) dx - \sum_{j=0}^{n-1} \frac{1}{6} (x_{j+1} - x_j) \left[f(x_j) + 4 \overbrace{f(x_{j+\frac{1}{2}})}^{+f(x_j)} \right] \right| \leq \frac{b-a}{2880} h^4 \max_{a < x < b} |f'''(x)|. \quad \square$$

More Examples By following the above procedure it's easy to generate (so-called) Newton-Cotes integration rules of any order.

It's interesting to make a simple comparison between the above two rules when the meshwidth is a constant $h = x_{j+1} - x_j$. Simpson's rule (as we have presented it) takes roughly twice as many function evaluations as the trapezoid rule. It's error is far smaller through:

$$\frac{\text{error in Trapezoid}}{\text{error in Simpson}} \sim \left(\frac{h^2}{240} \right)^{-1}.$$

If, for example, $h = \frac{1}{100}$, Simpson's rule error is smaller by 2.4×10^6 ! As h decreases the advantage of Simpson's rule increases further.

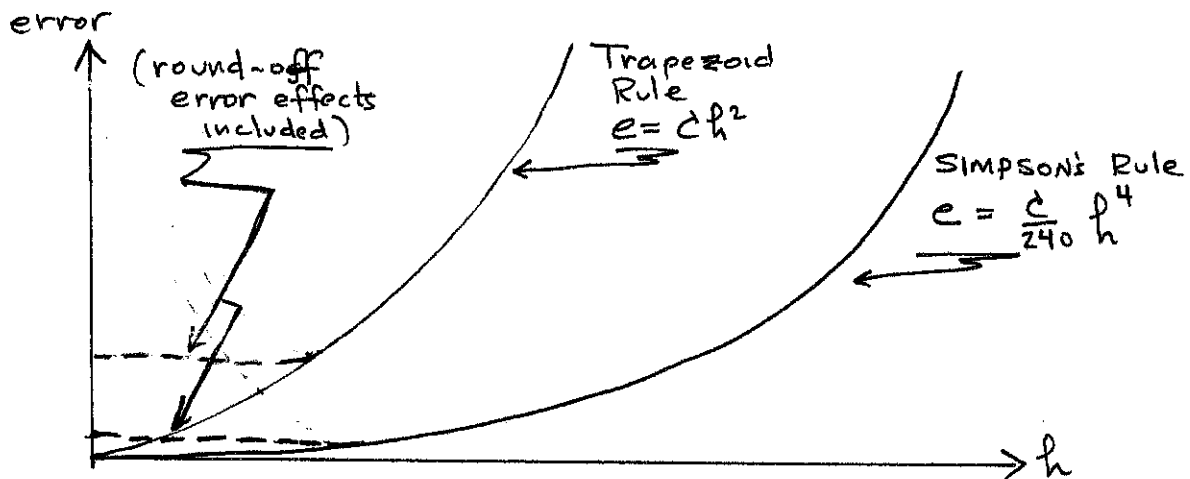


FIGURE: SIMPSON'S RULE IS BETTER!.

Of course, this picture neglects roundoff error. If each function evaluation contains a small bit of roundoff error, E_{\pm} , the total expected error of each is then:

$$\begin{aligned} \text{Total error in T.R.} &\sim Ch^2 + |E_{\pm}|(b-a), \\ \text{Total error in S.R.} &\sim \frac{Ch^4}{240} + 2|E_{\pm}|(b-a). \end{aligned}$$

Including these last terms accounts for the dotted lines in the previous figure. Once again:

Higher order formulas (like Simpson's) are superior!

Computer Exercise! Convert your trapezoid rule program to Simpson's rule. Repeat the experiment you did before and compare the results to those of the trapezoid rule.

3.4 Gauss Quadrature.

All quadrature schemes begin with the decomposition

$$\int_a^b f(x) dx = \sum_{j=0}^{n-1} \int_{x_j}^{x_{j+1}} f(x) dx.$$

The integral over (x_j, x_{j+1}) is approximated by a weighted sum of function values. The brilliant idea of Gauss was to pick *both* the *points* at which $f(x)$ is evaluated (called the quadrature points) *and* the *weights* to optimize the methods accuracy.

Example. : (Midpoint Rule) Suppose we seek a one-point approximation of optimal accuracy:

$$\int_{x_j}^{x_{j+1}} f(x) dx \cong w_j f(q_j).$$

This has two degrees of freedom (w_j and q_j) so we can ask it be *exact* on Π_1 . (Thus, it will have accuracy comparable to the trapezoid rule.) This yields

$$\begin{aligned} \int_{x_j}^{x_{j+1}} 1 dx &= (x_{j+1} - x_j) = w_j \cdot 1 \\ \int_{x_j}^{x_{j+1}} x dx &= \frac{x_{j+1}^2}{2} - \frac{x_j^2}{2} = w_j q_j. \end{aligned}$$

This is a 2×2 *nonlinear* system with solution $w_j = (x_{j+1} - x_j)$, $q_j = \frac{x_{j+1} + x_j}{2}$ ($= x_{j+1/2}$).

The midpoint rule is then:

$$(3.4.1) \quad \int_a^b f(x) dx \cong \sum_{j=0}^{n-1} (x_{j+1} - x_j) f\left(\frac{x_{j+1} + x_j}{2}\right).$$

This is exact on Π_1 so it attains $O(h^2)$ accuracy.

If we seek greater accuracy with a k -point formula

$$\int_{x_j}^{x_{j+1}} f(x) dx \cong \sum_{\ell=1}^k w_{j,\ell} f(q_{j,\ell}),$$

COMPUTER EXERCISE

Reconsider your program approximating $f'(a)$ by central differences. ~~Try to find it~~ a computable error. Modify it to include a computable estimate of the error, done two ways:

Method 1. Error $\doteq \left| \frac{h^2}{6} f'''(a) \right|$ where $f'''(a)$

is approximated by a finite difference approximation to $f'''(a)$ (which you should develop).

Method 2. Let $D_h f(a) = (f(a+h) - f(a-h)) / 2h$. A computable estimate is then

$$\text{Error} \doteq |D_h f(a) - D_{h/2} f(a)|.$$

Compute both and the true error. Which is closer?

Do all three approach zero like $O(h^2)$ as $h \rightarrow 0$?

With this in mind, we must have:

$$w_0 \cdot 1 + w_1 \cdot 1 + \cdots + w_n \cdot 1 = L(1), \quad (\text{exact on } 1)$$

$$w_0 \cdot x_0 + w_1 x_1 + \cdots + w_n x_n = L(x), \quad (\text{exact on } x),$$

$$w_0 x_0^2 + w_1 x_1^2 + \cdots + w_n x_n^2 = L(x^2), \quad (\text{exact on } x^2)$$

...

$$w_0 x_0^n + w_1 x_1^n + \cdots + w_n x_n^n = L(x^n), \quad (\text{exact on } x^n).$$

This is an $(n+1) \times (n+1)$ linear system for the weights w_0, \dots, w_n . If it's easy to solve using Gaussian elimination, then w_0, \dots, w_n can be determined this way.

→ (insert page HERE) ←

3.2 Numerical Differentiation.

The classic approach to numerical differentiation is to expand everything in sight in a Taylor series and cancel everything possible. However, the approach of the introduction is equally valid. We shall see how they are complementary.

Example. : (Forward Difference) Suppose we seek an approximation to $f'(a)$ using two points a and $a+h$.

$$f'(a) \cong w_0 f(a) + w_1 f(a+h).$$

Here $n=1$ so w_0 and w_1 are determined by requiring the formula be *exact* on linears, i.e., $f(x) \equiv 1$ and $f(x) = x$.

$$\text{Exact on } 1 : 1' = 0 = w_0 \cdot 1 + w_1 \cdot 1$$

$$\text{Exact on } x : x' = 1 = w_0 \cdot a + w_1(a+h).$$

This gives a 2×2 linear system for w_0 and w_1 whose solution is $w_0 = -\frac{1}{h}$, $w_1 = \frac{1}{h}$ and

$$f'(a) \cong -\frac{1}{h} f(a) + \frac{1}{h} f(a+h) = \frac{f(a+h) - f(a)}{h},$$

as expected!

The error in this approximation can be calculated by expanding the RHS in a Taylor series, as in Chapter 1. This gives:

$$\text{error} = f'(a) - \frac{f(a+h) - f(a)}{h} = -\frac{1}{2}h f''(\xi), \text{ some } \xi \text{ between } a \text{ and } a+h.$$

The following difference approximations can be derived exactly analogously to the last example. Their error properties are summarized.

Example. : (Central Difference)

$$f'(a) \cong \frac{f(a+h) - f(a-h)}{2h} \quad 1 \text{ exact on } 1, x, x^2,$$

$$\text{error} = f'(a) - \frac{f(a+h) - f(a-h)}{2h} = -\frac{h^2}{6} f'''(\xi), \text{ for some } \xi \in (a-h, a+h).$$

Example. : (Second Central Difference)

$$f''(a) \cong \frac{f(a+h) - 2f(a) + f(a-h)}{h^2}, \text{ exact on } 1, x, x^2, x^3$$

$$\text{error} = f''(a) - \frac{f(a+h) - 2f(a) + f(a-h)}{h^2} = -\frac{h^2}{12} f^{(4)}(\xi), \text{ some } \xi \in (a-h, a+h).$$

Numerical Differentiation: Roundoff.

Suppose we compute

$$f'(a) \cong \frac{f(a+h) - f(a-h)}{2h}$$

Then $f'(a) = \frac{f(a+h) - f(a-h)}{2h} + \frac{-h^2}{6} f'''(\eta)$. In general, there is an error in $f(a+h)$: E_+ and in $f(a-h)$: E_- ; errors can occur in measurement or storage or roundoff (if $f(a \pm h)$ is the result of other computations). Then, as

$$f_{\text{COMP}}(a \pm h) = f(a \pm h) + E_{\pm},$$

we have

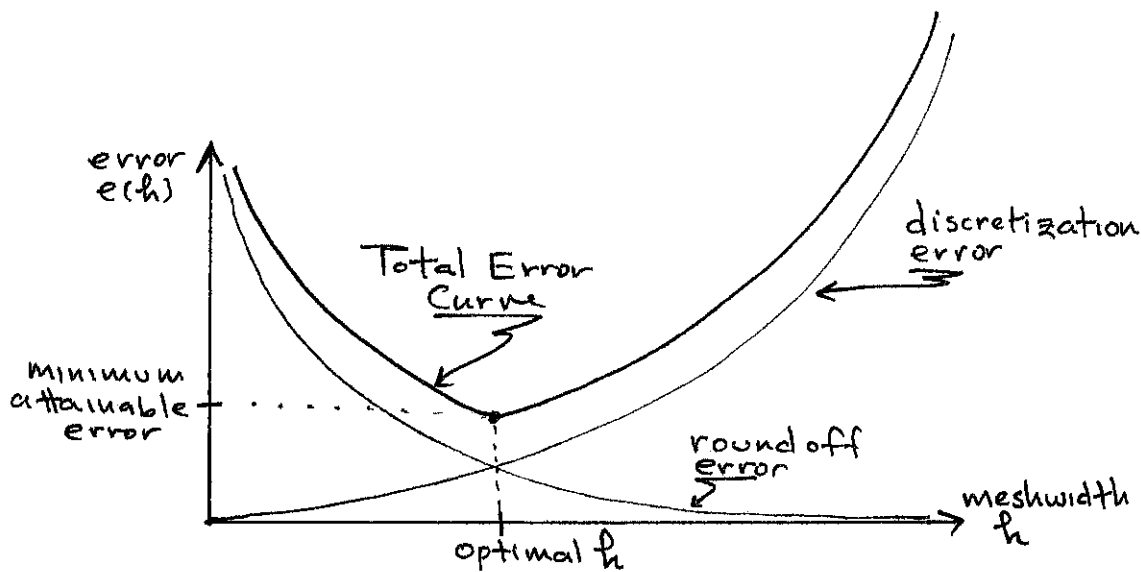
$$f'_{\text{COMP}} = \frac{f(a+h) - f(a-h)}{2h} + \frac{E_+ - E_-}{2h}$$

Hence, the total error (discretization plus-roundoff) is:

$$f'(a) - f'_{\text{COMP}} = -\frac{E_+ - E_-}{2h} - \frac{h^2 f'''(\eta)}{6}.$$

Notes: As $h \rightarrow 0$ the DISCRETIZATION ERROR $\rightarrow 0$.

The roundoff errors E_{\pm} do not decrease as $h \rightarrow 0$ so that $\frac{E_+ - E_-}{2h} \rightarrow \infty$ as $h \rightarrow 0$.



Example. Suppose the error in computing e^x for $-1 \leq x \leq 1$ is $\pm 1 \times 10^{-15}$ so $E_+ - E_- \approx \pm 2 \times 10^{-15}$. The roundoff error is approximately

$$\text{Roundoff} \cong \pm \frac{2 \times 10^{-15}}{2h}.$$

As for the discretization error: $(e^x)''' \leq 3$, so that

$$\text{Discretization error} \cong \frac{-h^2 3}{6} = \frac{-h^2}{2}.$$

Thus, we have a rough estimate for:

$$\begin{aligned} |\text{Total error}| &= |R| + |D| = \frac{2 \times 10^{-15}}{2h} + \frac{h^2}{2} \\ &= 10^{-15} h^{-1} + \frac{1}{2} h^2 \quad (\equiv g(h)) \end{aligned}$$

What is the minimum value of total error:

$$\begin{aligned} g'(h) &= -10^{-15} h^{-2} + h = 0 \Rightarrow -10^{-15} + h^3 = 0 \\ \text{or } h &= \sqrt[3]{10^{-15}} = 10^{-5} \text{ so } h_{\text{OPTIMAL}} \approx 10^{-5} \end{aligned}$$

and the minimum total error is roughly 10^{-10} .

Higher order difference approximations to derivatives are easy to derive by the same procedure.

we have $2k$ undetermined parameters. Thus, such a formula can be exact on $1, x, x^2, \dots, x^{2k-1}$. The overall rate of convergence will also be very high: $O(h^{2k})$. The only difficulty is solving the $2k \times 2k$ nonlinear system of equations on each (x_j, x_{j+1}) for the points and weights! Clearly, this needs to be done once and the weights and points tabulated. This is accomplished as follows:

Make a change of variables mapping (x_j, x_{j+1}) to $(-1, +1)$:

$$\int_{x_j}^{x_{j+1}} f(x) dx = \int_{-1}^{+1} g(t) dt.$$

This change of variable is important because then the weights and points are independent of (x_j, x_{j+1}) . A higher level of universality is achieved! This is simply done by:

$$x = \frac{x_{j+1} - x_j}{2} t + \frac{x_{j+1} + x_j}{2},$$

$$dx = \frac{x_{j+1} - x_j}{2} dt.$$

Thus, we have

$$(3.4.2) \quad \int_a^b f(x) dx = \sum_{j=0}^{n-1} \int_{x_j}^{x_{j+1}} f(x) dx = \sum_{j=0}^{n-1} \left(\frac{x_{j+1} - x_j}{2} \right) \int_{-1}^{+1} f \left(\frac{x_{j+1} - x_j}{2} t + \frac{x_{j+1} + x_j}{2} \right) dt.$$

Now we only need a quadrature rule on $(-1, +1)$:

$$\int_{-1}^{+1} g(t) dt \cong w_1 g(q_1) + \dots + w_k g(q_k)$$

which is used in (3.4.2). Note that with this approach we need only store k points and k weights. These are used for *every* subinterval!

How are these points and weights determined? By exactness on Π_{2k-1} naturally! We give two examples.

Example. (Two point Gauss) We seek a rule of the form:

$$\int_{-1}^{+1} g(t) dt \cong w_1 g(q_1) + w_2 g(q_2).$$

The $w_{1,2}, q_{1,2}$ are determined by asking this be exact on $1, t, t^2$ and t^3 :

$$\begin{aligned} \int_{-1}^{+1} 1 \, dx &= 2 = w_1 \cdot 1 + w_2 \cdot 1, \\ \int_{-1}^{+1} x \, dx &= 0 = w_1 \cdot q_1 + w_2 \cdot q_2, \\ \int_{-1}^{+1} x^2 \, dx &= \frac{2}{3} = w_1 \cdot q_1^2 + w_2 \cdot q_2^2, \\ \int_{-1}^{+1} x^3 \, dx &= 0 = w_1 \cdot q_1^3 + w_2 \cdot q_2^3. \end{aligned}$$

This is a 4×4 nonlinear system that *happens* to be easy to solve. (By symmetry, we expect $w_1 = w_2$ and $q_1 = -q_2$. Try solving this yourself!) The solution is:

$$w_1 = 1, \quad w_2 = 1, \quad q_1 = -\frac{1}{\sqrt{3}}, \quad q_2 = \frac{1}{\sqrt{3}}.$$

This gives the base rule

$$\int_{-1}^{+1} g(t) dt = 1 \cdot g\left(-\frac{1}{\sqrt{3}}\right) + 1 \cdot g\left(+\frac{1}{\sqrt{3}}\right).$$

Used in (3.4.2) as a composite rule we have:

$$(3.4.3) \quad \begin{cases} \int_a^b f(x) dx \cong \sum_{j=0}^{n-1} \left(\frac{x_{j+1} - x_j}{2} \right) \left[w_1 f\left(\frac{x_{j+1} - x_j}{2} q_1 + \frac{x_{j+1} + x_j}{2} \right) + \right. \\ \qquad \qquad \qquad \left. + w_2 f\left(\frac{x_{j+1} - x_j}{2} q_2 + \frac{x_{j+1} + x_j}{2} \right) \right], \\ q_1 = -\frac{1}{\sqrt{3}}, \quad q_2 = +\frac{1}{\sqrt{3}}, \quad w_1 = 1, \quad w_2 = 1. \end{cases}$$

This requires only two function evaluations per subinterval yet yields $O(h^4)$ accuracy (since it's exact on Π_3 .)

Remark: (3.4.3) points to the general programming procedure. We store a small array of weights $w(k)$, points $q(k)$. The RHS of (3.4.3) is programmed typically as 2 nested do loops: over the subintervals (x_j, x_{j+1}) then over the points $(w(\ell), q(\ell))$.

Computer Exercise: Convert your trapezoid rule program to use the two point Gauss rule. Repeat the experiment and compare the results to the Trapezoid rule and Simpson's rule.

Example. : (Three point Gauss) A three point Gauss formula takes the form:

$$\int_{-1}^{+1} g(t) dt \cong w_1 g(q_1) + w_2 g(q_2) + w_3 g(q_3).$$

The w_j, q_j are determined by asking this be exact on Π_5 :

$$\begin{aligned} \text{exact on } 1 &\Rightarrow 2 = w_1 + w_2 + w_3, \\ \text{exact on } x &\Rightarrow 0 = w_1 q_1 + w_2 q_2 + w_3 q_3, \\ \text{exact on } x^2 &\Rightarrow \frac{2}{5} = w_1 q_1^2 + w_2 q_2^2 + w_3 q_3^2, \\ \text{exact on } x^3 &\Rightarrow 0 = w_1 q_1^3 + w_2 q_2^3 + w_3 q_3^3, \\ \text{exact on } x^4 &\Rightarrow \frac{2}{5} = w_1 q_1^4 + w_2 q_2^4 + w_3 q_3^4, \\ \text{exact on } x^5 &\Rightarrow 0 = w_1 q_1^5 + w_2 q_2^5 + w_3 q_3^5. \end{aligned}$$

We expect, by symmetry, $w_1 = w_3$ - $q_1 = q_3$ and $q_2 = 0$. Nonetheless, this system starts to seem a bit intricate. It's solution is:

$$\begin{aligned} w_1 &= \frac{5}{9} & w_2 &= \frac{8}{9} & w_3 &= \frac{5}{9} \\ q_1 &= -\sqrt{\frac{3}{5}} & q_2 &= 0 & q_3 &= +\sqrt{\frac{3}{5}} \end{aligned}$$

Three point (Gauss, used as a composite rule, attains an overall error of $O(h^6)$.

The weights and points of the general k -point Gauss scheme are available in most mathematical handbooks for example, Gaussian Quadrature Formulas, by Stroud and Secrest.

We give one last example : the quite popular four-point Gauss scheme.

Example. : (Four point Gauss) Four point Gauss is exact on Π_7 and $O(h^8)$ accurate as a composite rule. It's weight and points are (to 10 significant digits) :

$$\begin{aligned} w_1 &= 0.3478548451, w_2 = 0.6521451549, w_3 = w_2, w_4 = w_1, \\ q_1 &= -0.8611363116, q_2 = -0.3399810436, q_3 = -q_2, q_4 = -q_1. \end{aligned}$$

Example: (5 point Gauss). Five point gauss rules are exact on Π_4 . To ten significant digits, the points and weights are given by:

$$w_1 = w_5 = 0.2369268851,$$

$$w_2 = w_4 = 0.4786286705,$$

$$w_3 = 0.5688888888,$$

$$-q_1 = q_5 = 0.9061798459$$

$$-q_2 = q_4 = 0.5384693101$$

$$q_3 = 0.0$$

Example: (6 point Gauss). Six point Gauss is exact on Π_5 . To ten significant digits, its points and weights are:

$$w_1 = w_6 = 0.1713244924$$

$$w_2 = w_5 = 0.3607615730$$

$$w_3 = w_4 = 0.4679139346$$

$$-q_1 = q_6 = 0.9324695142$$

$$-q_2 = q_5 = 0.6612093865$$

$$-q_3 = q_4 = 0.2386191861.$$

Exercises

3.4.1. Consider $\int_0^1 f(x) dx$ where $f(x) = \frac{1}{1+x^2}$.
Take the mesh

$$0 = x_0 < x_1 = \frac{1}{4} < x_2 = \frac{2}{3} < x_3 = 1.$$

(a) Calculate an approximation using Simpson's rule on this mesh. What is the error.

(b) For an arbitrary mesh

$$a = x_0 < x_1 < x_2 < \dots < x_N = b,$$

write down in pseudo code the Simpson's rule algorithm for approximating $\int_a^b f(x) dx$.

3.4.2. Repeat problem 1 for ^{the} two point Gauss rule

3.4.3. ~~The~~ The , so-called, Gauss-Lobatto quadrature use the interval's end points (like Newton-Cotes methods) and a ~~f~~ set of internal points ~~being~~ picked to obtain optimal accuracy (like Gauss methods). Consider a Gauss-Lobatto formula:

$$\int_{-1}^1 f(t) dt \doteq w_1 f(-1) + w_2 f(q_2) + w_3 f(q_3) + w_4 f(1).$$

(a) What accuracy do you anticipate is attainable by a formula of this type, used as a composite rule?

(b) Derive the nonlinear system for w_1, \dots, w_4 , and q_2, q_3 arising by imposing exactness on T_6 .

3.5 Adaptive Quadrature.

The problem of adaptivity is to compute

$$\int_a^b f(x)dx \doteq I_h(f) \text{ (some approximation)}$$

with *minimal or near minimal cost* and guaranteed accuracy

$$\left| \int_a^b f(x)dx - I_h(f) \right| < \epsilon, \text{ a user-supplied error tolerance.}$$

Thus, we want an algorithm that will automatically select the points x_j

$$a = x_0 < x_1 < \dots < x_n = b$$

so that the *error* and *cost* requirements are satisfied.

Suppose that we use a composite rule that is exact on Π_k . We thus decompose

$$(3.5.1) \quad \int_a^b f(x)dx = \sum_{j=0}^{n-1} \int_{x_j}^{x_{j+1}} f(x)dx \doteq \sum_{j=0}^{n-1} I_j(f)$$

where $I_j(f) \cong \int_{x_j}^{x_{j+1}} f(x)dx$. If $I_j(f)$ is *exact* on Π_k , we have seen that the global approximation (3.5.1) is $O(h^{k+1})$ accurate. On each (x_j, x_{j+1}) it is $O((x_{j+1} - x_j)^{k+2})$ accurate:

$$\int_{x_j}^{x_{j+1}} f(x)dx - I_j(f) = C(x_{j+1} - x_j)^{k+2} f^{(k+2)}(\xi_j), \text{ for some } \xi_j \in (x_j, x_{j+1}).$$

Suppose we can approximate $\int_{x_j}^{x_{j+1}} f(x)dx$ to an error smaller than $\epsilon(x_{j+1} - x_j)/(b - a)$:

$$\left| \int_{x_j}^{x_{j+1}} f(x)dx - I_j(f) \right| < \frac{\epsilon(x_{j+1} - x_j)}{b - a}.$$

Summing this from $j = 0$ to $n - 1$ then implies:

$$|\text{error}| = \left| \int_a^b f(x)dx - \sum_{j=0}^{n-1} I_j(f) \right| < \frac{\epsilon}{b - a} \sum_{j=0}^{n-1} (x_{j+1} - x_j) = \epsilon,$$

our target accuracy! Thus, a local error of $\frac{\epsilon(x_{j+1} - x_j)}{b - a}$ ensures a global error of ϵ .

There are thus *three* components of an adaptive strategy:

- Estimation: A computable estimate for the local error: $\int_{x_j}^{x_{j+1}} f(x)dx - I_j(f)$.
- Accuracy: A strategy for cutting the mesh and recomputing the approximation when the local error is bigger than $\epsilon(x_{j+1} - x_j)/(b - a)$.
- Efficiency: A strategy for *increasing* the mesh size when the local error is much smaller than necessary ($\ll \epsilon(x_{j+1} - x_j)/(b - a)$).

Estimation of the Local Errors.

The simplest way to estimate local errors is to approximate $\int_{x_j}^{x_{j+1}} f(x)dx$ by one method and then by a more accurate one. The number of significant digits of agreement are a good estimate of the accuracy of the first method and a conservative estimate of the accuracy of the more accurate one. (One common method is to cut (x_j, x_{j+1}) in half for the more accurate approximation.)

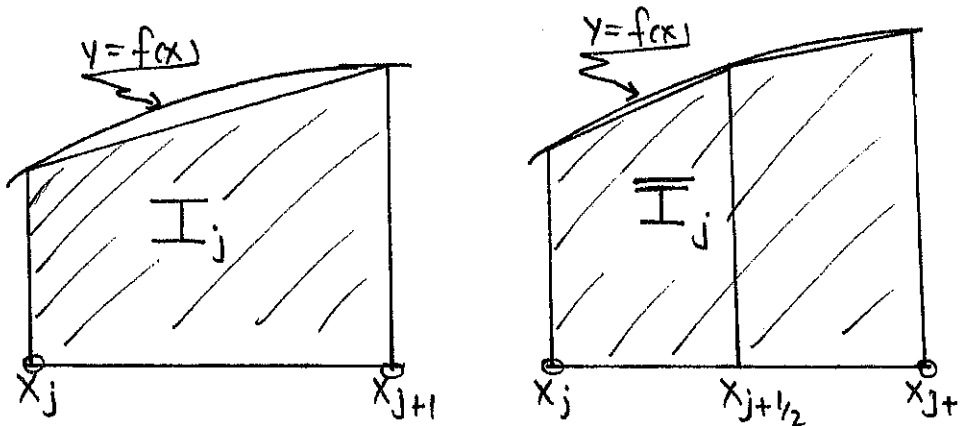


FIGURE: A TRAPEZOID APPROXIMATION AND A MORE ACCURATE ONE.

Example. : (The Trapezoid Rule) For the trapezoid rule $|I_j - \bar{I}_j|$ is a crude but reliable estimate for $|\int_{x_j}^{x_{j+1}} f(x)dx - \bar{I}_j|$. However, it is not an efficient estimator.

This estimate can be improved a lot as follows. From Theorem 3.3.1 we know

$$\int_{x_j}^{x_{j+1}} f(x)dx - I_j = \frac{(x_{j+1} - x_j)^3}{12} f''(\xi_j).$$

Similarly, applying it over $(x_j, x_{j+1/2})$ and $(x_{j+1/2}, x_{j+1})$ then adding gives:

$$\int_{x_j}^{x_{j+1}} f(x)dx - \bar{I}_j = \frac{(x_{j+1/2} - x_j)^3}{12} f''(\tilde{\xi}_j) + \frac{(x_{j+1} - x_{j+1/2})^3}{12} f''(\tilde{\xi}_j).$$

It's reasonable to suppose f'' does not vary much over (x_j, x_{j+1}) . Let $h_j = (x_{j+1} - x_j)$ (so $(x_{j+1/2} - x_j) = (x_{j+1} - x_{j+1/2}) = h_j/2$). The last two equations give:

$$(3.5.2) \quad \begin{aligned} \int_{x_j}^{x_{j+1}} f(x)dx - I_j &= \frac{h_j^3}{12} f'' \quad (= \text{local error in } I_j), \\ \int_{x_j}^{x_{j+1}} f(x)dx - \bar{I}_j &= \frac{1}{4} \frac{h_j^3}{12} f'' \quad (= \text{local error in } \bar{I}_j). \end{aligned}$$

Subtracting these two equations gives:

$$\bar{I}_j - I_j = \frac{3}{4} \frac{h_j^3}{12} f''.$$

Thus, $\frac{1}{3}(\bar{I}_j - I_j) = \frac{1}{4} \frac{h_j^3}{12} f'' =$ (compare with (3.5.2)) $=$ the local error in \bar{I}_j . We thus have:

$$(3.5.3) \quad \int_{x_j}^{x_{j+1}} f(x)dx - \bar{I}_j = \frac{\bar{I}_j - I_j}{3} \quad (+ \text{ Higher Order Terms}).$$

Thus, $\frac{1}{3}(\bar{I}_j - I_j)$ is a computable local error estimator for the error in \bar{I}_j . This estimation is smaller by a factor of 3 than the simpleminded one of $|I_j - \bar{I}_j|$.

More generally, if the scheme used on (x_j, x_{j+1}) is exact on Π_k we can do the analogous computation:

$$\begin{aligned} I_j &: \text{apply scheme once on } (x_j, x_{j+1}) \\ \bar{I}_j &: \text{cut } (x_j, x_{j+1}) \text{ in half, apply on} \\ &\quad (x_j, x_{j+1/2}), (x_{j+1/2}, x_{j+1}) \text{ and sum.} \end{aligned}$$

Then, a sharp estimator for \bar{I}_j is:

$$(3.5.4) \quad \int_{x_j}^{x_{j+1}} f(x)dx - \bar{I}_j = \frac{1}{2^{k+1} - 1} (\bar{I}_j - I_j) \quad (+ \text{ Higher Order terms}).$$

Examples. :

Simpson's Rule, $k = 3$, $\frac{1}{15} (\bar{I}_j - I_j)$.

Four Point Gauss, $k = 7$, $\frac{1}{255} (\bar{I}_j - I_j)$.

3.5.1
Exercises: Consider the trapezoid rule. Show that $|\bar{I}_j - I_j|$ overestimates the error in \bar{I}_j by a factor of three (so it is reliable but not efficient for \bar{I}_j) but underestimates the error in I_j by a factor of $\frac{3}{4}$ (so it is efficient but not reliable for I_j).

Verify (3.5.4) and the above two examples.
3.5.2.

Accuracy and Efficiency: A Mesh Refinement and de-Refinement Strategy.

The strategy is pretty simple: If the estimator on (x_j, x_{j+1}) is too big \bar{I}_j is rejected, the interval is cut (for example, x_{j+1} is replaced by $x_{j+1}^{\text{NEW}} = x_j + \frac{1}{2} (x_{j+1}^{\text{OLD}} - x_j)$ and the calculation repeated on $(x_j, x_{j+1}^{\text{NEW}})$. If the estimated error is much too small, \bar{I}_j is accepted but the size of the next interval is doubled. There are only two choices that must be made. First, to accept a simple mesh halving and doubling or use a more sophisticated adjustment. Second, to set the tolerances far enough apart that there is no "flip-flopping" (half \rightarrow double \rightarrow half \rightarrow double $\rightarrow \dots$).

With the mesh halving and doubling the algorithm proceeds as follows.

Algorithm: Adaptive Quadrature.

Set upper error tolerance: ϵ_{max}

Compute lower error tolerance: $\epsilon_{\text{min}} = \frac{1}{2^{k+3}} \epsilon_{\text{max}}$.

Initialize: Integral = 0.0, $x_{\text{LEFT}} = a$.

Set initial meshwidth: h

4 Numerical Methods for Ordinary Differential Equations.

4.1 Introduction.

Suppose we are modeling a system that is changing. If its state at time t is represented by a number $y(t)$ (or a collection of numbers $\vec{y}(t)$) it's reasonable to suppose that its rate of change depends upon its current state. This general situation leads to the initial-value problem: find $y(t)$ satisfying

$$(4.1.1) \quad \begin{cases} y'(t) = f(t, y(t)), \text{ for } t > 0, \\ y(0) = y_0, \text{ (known)}. \end{cases}$$

Higher order initial value problems (IVP's for short) often occur through Newton's laws, for example. A particles position $s(t)$ satisfies a second order IVP:

$$\begin{cases} s''(t) = g(t, s(t), s'(t)), t > 0, \\ s(0) = s_0, \\ s'(0) = s_1. \end{cases}$$

This, and in fact **any** higher order IVP, can be written in the form (4.1.1). Let $y_1(t) = s(t), y_2(t) = s'(t)$. Then $y'_1 = y_2$ and $y'_2 (= s'' = g(t, s, s')) = g(t, y_1, y_2)$. This gives the IVP: $y_1(0) = s_0, y_2(0) = s_1$ and:

$$\begin{aligned} y'_1 &= y_2 && \iff \vec{y}' = \vec{f}(t, \vec{y}), \vec{y}(0) = \vec{y}_0. \\ y'_2 &= g(t, y_1, y_2) \end{aligned}$$

The problem is now, given $\vec{f} : \mathbb{R} \times \mathbb{R}^n \rightarrow \mathbb{R}^n$ and $\vec{y}_0 \in \mathbb{R}^n$, solve approximately the IVP:

$$(4.1.2) \quad \vec{y}' = \vec{f}(t, \vec{y}), \quad \vec{y}(0) = \vec{y}_0, \text{ or equivalently.}$$

$$(4.1.3) \quad \begin{cases} y'_1 = f_1(t, y_1, y_2, \dots, y_n), \\ y'_2 = f_2(t, y_1, y_2, \dots, y_n), \\ \dots \\ y'_n = f_n(t, y_1, y_2, \dots, y_n). \end{cases}$$

One basic approach to approximating the solution of (4.1.1) and (4.1.2) is to replace all the derivatives by differences and compute. We begin with the two examples.

Example: Euler's Method for $y' = f(t, y)$.

Let h denote the time-step and t_j the j th time. Thus, $h = T/N$ for some integer N and $t_j = jh = t_{j-1} + h$. Since $y'(t_j) = \lim_{h \rightarrow 0} \frac{y(t_j+h) - y(t_j)}{h}$, we can take h "small" and approximate the derivative with the difference quotient. In (4.1.1) this gives:

$$(4.1.4) \quad y_0 \text{ given}, \quad \frac{y_{j+1} - y_j}{h} = f(t_j, y_j).$$

The method (4.1.4) is very simple to program. It is known as Euler's method. As a concrete example, consider

$$y' = y, y(0) = 1, \quad (\text{true solution : } y(t) = e^t).$$

Euler's method is:

$$\frac{y_{n+1} - y_n}{h} = y_n, y(0) = 1, \text{ so}$$

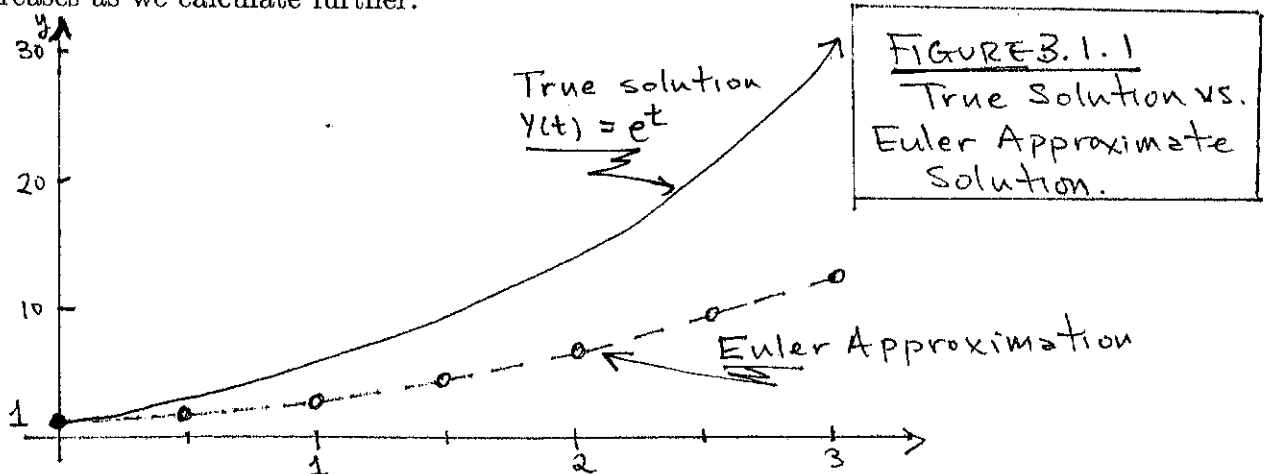
$$y_{n+1} = (1+h)y_n \quad \text{and} \quad y_n = (1+h)^n.$$

Example: Recall that $t_n = nh$ so $n = t_n/h$. Then $y_n = (1+h)^n = [(1+h)^{1/h}]^{t_n} \rightarrow e^{t_n}$ as $h \rightarrow 0$ for t_n fixed.

If we take $h = \frac{1}{2}$ (much too large for practical calculation) we get the following

n	0	1	2	3	4	5	6
t_n	0	$\frac{1}{2}$	1	$\frac{3}{2}$	2	$\frac{5}{2}$	3
y_n	1	1.5	2.225	3.375	5.0625	7.59375	11.390625

This has the correct general behavior but there is an error which, not surprisingly, increases as we calculate further.



Example: A Second Order IVP. As another brief example, consider the IVP from Newtonian mechanics:

$$s''(t) = g(t, s(t), s'(t)), \text{ for } 0 < t \leq T, s(0) = \alpha, s'(0) = \beta.$$

Let the stepsize $h = T/N$, $t_n = nh$ and s_n be the approximation to $s(t_n)$. Replacing derivatives by differences gives:

$$\left\{ \begin{array}{l} s_0 = \alpha \\ \frac{s_1 - s_0}{h} = \beta, \quad s_0 \quad s_1 = \alpha + h\beta \\ \frac{s_{n+1} - 2s_n + s_{n-1}}{h^2} = g(t_n, s_n, \frac{s_n - s_{n-1}}{h}), \text{ for } n = 2, \dots, N. \end{array} \right.$$

Note that this is again **explicit** calculation.

In both of these examples, it is not clear if the solutions of the differences approximations are close to the solutions of the IVP's.

Example: An Implicit Method.

Consider $y' = f(t, y)$ again. If y' is replaced by a difference quotient (as before) but $f(t, y)$ is evaluated at (t_{n+1}, y_{n+1}) we obtain the, so-called, "backward Euler" method:

$$\frac{y_{n+1} - y_n}{h} = f(t_{n+1}, y_{n+1}) \quad , y_0 \text{ given.}$$

Note that to find the y_{n+1} we must solve the nonlinear system: given t_{n+1}, y_n find y_{n+1} :

$$y_{n+1} - hf(t_{n+1}, y_{n+1}) = y_n ,$$

at each timestep. This is more costly than simply evaluating $f(t_n, y_n)$ (as in Euler's method) but it is not particularly difficult using the methods in Chapter 2. Note that in numerical methods for IVP's we **always** have lots of good initial guesses for y_{n+1} including

- $y_{n+1}^{\text{guess}} = y_n$.
- linear extrapolation: $y_{n+1}^{\text{guess}} = 2y_n - y_{n-1}$
- an explicit method: $y_{n+1}^{\text{guess}} = y_n + hf(t_n, y_n)$.

We shall see that simple implicit methods can succeed where all explicit methods fail miserably.

Example: A Multi-Step Method

A one-step method determines y_{n+1} from y_n . A "multi-step" method uses more time levels. One common approach to deriving multi-step method is through numerical integration. Integrate from t_{n-m} to t_{n+1}

Example: Stability of Explicit vs. Implicit Methods.

Why would anyone want to solve a nonlinear system each time step when (with an explicit method) you could just do a simple calculation and move on? The answer is that explicit methods are sometimes unstable for interesting problems!

The simplest example where a difference occurs is:

$$y' = -\lambda y, \quad y(0) = 1, \quad \text{solution: } y(t) = e^{-\lambda t}.$$

For $\lambda > 0$, $y(t) \rightarrow 0$ as $t \rightarrow \infty$.

Thus, any growth in an approximation to this problem is an instability.

Consider the two Euler methods:

$$\text{Forward: } y_{n+1} = y_n - h\lambda y_n, \quad y_n = (1 - h\lambda)^n.$$

$$\text{Backward: } y_{n+1} = y_n - h\lambda y_{n+1}, \quad y_n = \left(\frac{1}{1+h\lambda}\right)^n.$$

For $\lambda > 0$, $\left|\frac{1}{1+h\lambda}\right| < 1$ so the backward Euler approximation $y_n \rightarrow 0$ as $n \rightarrow \infty$. It is always stable!

For $\lambda > 0$, the (explicit) forward Euler method is stable if and only if $|1 - h\lambda| < 1$. This is equivalent to

$$-1 < 1 - h\lambda < +1, \text{ or } h < \frac{2}{|\lambda|}.$$

Thus, if the (rate constant) λ is very large (e.g., $\lambda = 1000$):

{ The true solution $y(t) \rightarrow 0$ very fast.
The Euler approximation $y_n \rightarrow \infty$
unless h is very small (e.g., $h < \frac{1}{500}$).

$$y'(t) = f(t, y(t)) \rightarrow y(t_{n+1}) - y(t_{n-m}) = \int_{t_{n-m}}^{t_{n+1}} f(t, y(t)) dt.$$

If the integral is approximated by a weighted sum of function values

$$w_0 f(t_{n+1}, y_{n+1}) + w_1 f(t_n, y_n) + \dots + w_{m+1} f(t_{n-m}, y_{n-m})$$

a multi-step method results.

As an example, (purposely chosen to be an example of what can go wrong) consider the 1-point Gauss rule (i.e., the midpoint rule):

$$\int_{x_{n-1}}^{x_{n+1}} f(t, y(t)) dt = (x_{n+1} - x_{n-1}) f(t_n, y_n)$$

This yields the two-step method:

$$y_{n+1} - y_{n-1} = 2h f(t_n, y_n)$$

where $h = x_{n+1} - x_n = x_n - x_{n-1}$.

This scheme is more difficult to start than Euler's method (how do we calculate y_1 ?). It is more difficult to change the step size from one step to another. On the other hand, it is much more accurate - $O(h^2)$ vs $O(h)$ and quite inexpensive. Unfortunately, it is also unstable!

To see this, consider the simple IVP:

$$y' = -ay, \quad y(0) = 1, \quad a > 0 \text{ is constant};$$

the solution is:

$$y(t) = e^{-at} \rightarrow 0, \text{ as } t \rightarrow \infty.$$

Thus, any growth in the approximate solution is introduced only through the numerical method. Applying the multi-step method to this equation gives:

$$y_{n+1} - y_{n-1} \stackrel{?}{=} 2ha y_n, \text{ or,}$$

$$y_{n+1} + 2ha y_n - y_{n-1} = 0.$$

This is a **linear, constant coefficient, homogeneous, difference equation**. Solutions to these are known to be linear combinations of **power** functions ($y_n = c_1 r_1^n + c_2 r_2^n$). In the case of ordinary differential equations the analogous solution is $y(t) = c_1 e^{r_1 t} + c_2 e^{r_2 t}$.

Substituting $y_n = r^n$ into the difference equation gives: $r^{n+1} + 2ahr^n - r^{n-1} = 0$
or :

$$r^2 + 2ahr - 1 = 0.$$

The roots $r_{1,2}$ are

$$r_1 = -ah + \sqrt{1 + (ah)^2}, r_2 = -ah - \sqrt{1 + (ah)^2}.$$

The approximate solution produced by this method is then:

$$y_n = c_1 r_1^n + c_2 r_2^n.$$

Note that $|r_1| < 1$ but $r_2 < 0$ and $|r_2| > 1$. Thus,

$$|y_n| \rightarrow +\infty \quad \text{while} \quad y(t_n) \rightarrow 0.$$

Exercise: Consider the last example problem:

$$y' = -ay, y(0) = 1, y(t) = e^{-at} \rightarrow 0 \text{ as } t \rightarrow \infty, \text{ for } a > 0.$$

Approximate this by Euler's method and show $y_n \rightarrow 0$ as $n \rightarrow \infty$ for h small enough. Repeat this analysis for the backward Euler method. Show $y_n \rightarrow 0$ as $t_n \rightarrow \infty$ for any $h > 0$.

4.2 More on Euler's Method.

Euler's Method isn't really used in any practical calculations. Still it is very handy to use it to introduce in a very clear way ideas that are commonly used for other methods in practical calculations and in the mathematical theory that supports them. In this section, we will use Euler's method to show

- solving **systems** of equations is as easy as solving one equation.
- **roundoff error** can be controlled in a practical calculation.
- **convergence** of y_n to the true solution $y(t_n)$ reduces to **stability** of the numerical methods used-which is tested for $y' = -\lambda y$.

Systems of Equations

Euler's method for a scalar problem

$$y' = f(t, y)$$

at its most simple might read something like this:

$$y_{\text{new}} = y_{\text{old}} + h \cdot f(t_{\text{old}}, y_{\text{old}}),$$

which becomes something like the following.

ALGORITHM: Euler's method

```

FUNCTION  F(T, Y) specified
Y0    specified
H        specified
TFINAL   specified
Yold = Y0
T = 0.0
COMPUTE  UNTIL  T ≥ TFINAL
(*)      Ynew = Yold + H · F(T, Yold)
          T ← T + H
          PRINT  T, Ynew
          Yold ← Ynew

```

4.1.4

Exercise: Suppose we wish to change H step by step by some formula such as $H_{\text{new}} = g(H_{\text{old}})$. Rewrite this algorithm for varying stepsizes.

If we are solving a system of equations with N equations the addition required is that we must define $Y(N), Y_{\text{old}}(N), Y_0(N)$ to be vectors and $F(N, T, Y)$ to be a vector function with N components. The statement (*) is replaced by a loop:

```

For  J = 1, ..., N
    Ynew(J) = Yold(J) + H F(J, T, Yold)

```

otherwise the program is essentially the same.

It's sometimes useful to write the algorithm **without** indices the case of the two equations:

$$x'(t) = g(t, x(t), y(t)), \quad y'(t) = f(t, x(t), y(t)), \quad x(0) = x_0, \quad y(0) = y_0.$$

In this case the algorithm reads:

ALGORITHM: Euler's method for $x' = g(t, x, y), y' = f(t, x, y)$.

```

FUNCTION: F(t,X,Y) specified
FUNCTION: G(t,X,Y) specified
H specified
X0 specified
Y0 specified
Xold = X0
Yold = Y0
T = 0.0
COMPUTE UNTIL T ≥ TFINAL
    Xnew = Xold + H · G(T, Xold, Yold)
    Ynew = Yold + H · F(T, Xold, Yold)
    T ← T+H
    PRINT T, Xnew, Ynew
    Xold ← Xnew
    Yold ← Ynew

```

Control of Round off Error

Suppose we wish to solve

$$y'(t) = f(t, y) \quad y(0) = y_0 \quad ,$$

for $0 \leq t \leq 1$, by Euler's Method:

$$\frac{y_{n+1} - y_n}{h} = f(t_n, y_n) \quad , y_0 = y_0, Nh = T.$$

At each step we actually compute

$$\tilde{y}_{n+1} = \tilde{y}_n + h\Delta\tilde{y}_n + \varepsilon_n,$$

where,

$$\Delta\tilde{y}_n = f(x_n, y_n)$$

ε_n = round off error contribution = 0(machine accuracy).

The effect of the accumulation of the ε_n depends on a number of things, including

1. the kind of arithmetic used,
2. the way the machine rounds,

3. the order of the operations,
4. the numerical method used,

and so on.

If, for example, the error is additive, then the total effect of rounding error is going from $t = 0$ to $t = 1$ is $O(\epsilon/h)$ (machine tolerance ϵ).

On the other hand, a stable method applied to a stable problem may not experience an additive accumulation, only a loss of accuracy of $O(\epsilon)$. The effect of these roundoff errors on unstable methods or unstable initial value problems, however, is catastrophic.

One way to control the effect of roundoff errors is through **partial double precision accumulation**:

- (i) Each y_n is stored in double precision.
- (ii) $h\Delta y_n$ is computed in single precision (only the single precision part of y_n is used for function evaluations.)
- (iii) $y_n + h\Delta y_n$ is formed in double precision and stored in double precision y_{n+1} .

Cost: Only 1 double precision sum/step.

Notice that we evaluate $f(t, y)$ in single precision and form the sum $y_{n+1} = y_n + h\Delta y_n$ in double. This is very economical and minimizes the roundoff resulting from adding a large number to a small number (since $y_n + h\Delta y_n = O(1) + O(h)$).

COMPUTER EXERCISE 4.1.5.

Consider your Euler's method program. Use it to solve $y' = ay$, $a = -100, -1, \text{ and } +1$. How does the error behave for fixed h as t_n increases?

Convergence of Euler's Method. Let us consider again the scalar problem:

$$y' = f(t, y), \quad 0 < t \leq T, \quad y(0) = y_0,$$

and Euler's method. Pick $0 < h < 1$ and compute:

$$y_{n+1} = y_n + hf(t_n, y_n), \quad t_n = nh.$$

Theorem 4.2.1 Suppose y'' and $\frac{\partial f}{\partial y}$ are bounded:

$$|y''(t)| \leq Y, \quad \left| \frac{\partial f}{\partial y}(t, y) \right| \leq L, \quad \text{for } 0 \leq t \leq T.$$

Then, the error in Euler's method satisfies:

$$|y(t_n) - y_n| \leq h \frac{Y}{2L} [e^{Lt} - 1], \quad \text{for } t > 0.$$

Note that this implies:

- the error is $O(h)$ as $h \rightarrow 0$.
- the estimate of the error increases exponentially as we leave $t = 0$.

This last effect can be pessimistic. In the first example with Euler's method it did, however, occur. *We will give a detailed proof of this theorem in section 4.3,*

Systems of Equations

The problem with **any** first order accurate method is that it is very hard to get more than two significant digits of accuracy before roundoff error swamps the calculation. To attain higher accuracy we need a method which is (of course) stable and whose local truncation error is $O(h^3)$ or smaller. Just as in numerical differentiation, truncation error is assessed by a Taylor series calculation with the true solution.

Let $y(t)$ be the true solution of

$$(4.3.1) \quad y'(t) = f(t, y(t))$$

Expanding $y(t_{n+1}) = y(t_n + h)$ in Taylor series about

$$y(t_{n+1}) = y(t_n + h) = y(t_n) + hy'(t_n)$$

$$(4.3.2) \quad + \frac{h^2}{2} y''(t_n) + \frac{h^3}{3!} y'''(t_n) + \dots$$

Interestingly, all derivatives of y can be calculated from (4.3.1):

$$\begin{cases} y'(t) = f(t_n, y(t_n)) \\ y''(t_n) = \frac{d}{dt}[f(t_n, y(t_n))] = \frac{\partial f}{\partial t}(t_n, y(t_n)) + \frac{\partial f}{\partial y}(t_n, y(t_n)) y'(t_n) \\ \quad = f_t(t_n, y(t_n)) + f_y(t_n, y(t_n)) f(t_n, y(t_n)). \\ y'''(t_n) = \frac{d}{dt}[f_t + f_y f] = [f_{tt} + 2f_{ty}f + f_{yy}f^2 + f_t f_y + f_y^2 f]|_{t_n, y(t_n)} \end{cases}$$

and so on.

This expansion is most important for deriving good higher order methods such as the Runge-Kutta methods of the next section. It can also be used to generate methods that, while higher order, are too costly. Here are a few examples:

Example: Euler's method.

If (4.3.2) is truncated after the $0(h)$ term we get:

$$\begin{cases} y(t_{n+1}) = y(t_n) + hy'(t_n) \quad (+0(h^2)). \\ \text{Using } y' = f(t, y) \text{ gives Euler's method:} \\ y_{n+1} = y_n + hf(t_n, y_n) \quad (+0(h^2)). \end{cases}$$

Its "truncation error" is, $0(h^2)$, and its global error is $0(h)$.

Example: A Globally $0(h^2)$ accurate method.

Taking one more term gives:

$$y(t_{n+1}) = y(t_n) + hy'(t_n) + \frac{h^2}{2}y''(t_n) \quad (+0(h^3)).$$

Using the above formulas for y' and y'' gives:

$$y_{n+1} = y_n + hf'(t_n, y_n) + \frac{h^2}{2}[f_t(t_n, y_n) + f_y(t_n, y_n)f(t_n, y_n)],$$

which has $0(h^3)$ local truncation error and $0(h^2)$ global accuracy.

For scalar equations, this quickly becomes too expensive as we seek still higher accuracy. For $n \times n$ systems even the second example is too costly, because f_y is now the $n \times n$ Jacobi matrix. Evaluating $f_y(t_n, y_n)$ thus requires n^2 function evaluation!

4.2.1.)
Exercise: It is instructive, although painful, to calculate $y''''(t_n)$ as above. Equally painful is to write down explicitly a Taylor series method for a system of two ordinary differential equations. Find the $0(h^2)$ accurate Taylor series method for $x' = f(t, x, y)$, $y' = g(t, x, y)$.

4.3. Convergence of Euler's Method.

This section presents a detailed proof of the convergence theorem, Theorem 4.2.1. The proof is interesting because it shows that convergence follows by a longish but standard argument from two principles that are easy to verify (or show false!)

(i) Consistency: this is verified by a Taylor series ~~calculation~~ calculation.

(ii) Stability: ^{of the method} for the linear, constant coefficient problem

$$\phi' = L\phi + Y, \quad L, Y \text{ constants.}$$

This is verified by simply back-solving the recursion coming from the method.

This reduction of a difficult but essential requirement of convergence into two routine and much easier steps is often expressed as:

"Stability + Consistency \Rightarrow Convergence"!

(We will repeat this ^{reduction} at several key times in the proof.)

Proof of Theorem 4.2.1.

The proof of Theorem 4.2.1 is very interesting because it breaks down into two steps

1. consistency, i.e., the local truncation error $\rightarrow 0$ as $h \rightarrow 0$.
2. Stability, for the linear constant coefficient problem: $y' = Ly + f$, (L, f constants).

Each of these can be easily checked, individually. Often this decomposition is described by:

$$\text{Stability} + \text{Consistency} \Rightarrow \text{Convergence.}$$

With that in mind, let us begin the proof.

Step 1: Use Consistency to derive a difference equation for the error.

The true solution $y(t)$ satisfies:

$$y(t_n + h) = y(t_n) + h y'(t_n) + \frac{h^2}{2} y''(\xi_n),$$

for some ξ_n between t_n and t_{n+1} .

Thus, with $e_n = y(t_n) - Y_n$, we subtract:

- P -

$$y(t_{n+1}) = y(t_n) + h f(t_n, y(t_n)) + \frac{h^2}{2} y''(\xi_n)$$

$$\underline{y_{n+1} = y_n + h f(t_n, y_n)}$$

$$e_{n+1} = e_n + h [f(t_n, y(t_n)) - f(t_n, y_n)] + \frac{h^2}{2} y''(\xi_n)$$

Step 2: Convert this to a linear, constant coefficient, difference inequality.

This uses the two assumed bounds in the theorem. First, since $|y''| \leq Y$ (constant)

$$\left| \frac{h^2}{2} y''(\xi_n) \right| \leq \frac{h^2}{2} Y, \text{ for any } n.$$

Second, by the mean value theorem:

$$f(t_n, y(t_n)) - f(t_n, y_n) = \frac{\partial f}{\partial y}(t_n, \xi_n) (y(t_n) - y_n),$$

where ξ_n is (another) unknown point between y_n and $y(t_n)$. This implies the inequality:

$$|f(t_n, y(t_n)) - f(t_n, y_n)| \leq L |y(t_n) - y_n|,$$

(which is called a "Lipschitz condition on f ")
where

$$L = \max \left| \frac{\partial f}{\partial y} \right|$$

Using these last two bounds in the error equation gives

$$\begin{aligned} |e_{n+1}| &\leq |e_n| + h |f(t_n, y(t_n)) - f(t_n, y_n)| \\ &\quad + \frac{h^2}{2} |y''(\xi_n)| \leq \\ &\leq |e_n| + hL |e_n| + \frac{h^2}{2} \underline{Y}, \end{aligned}$$

or, collecting terms

$$|e_{n+1}| \leq (1+hL) |e_n| + \frac{h^2}{2} \underline{Y}.$$

Remark: This can be rewritten as

$$\frac{|e_{n+1}| - |e_n|}{h} \leq L |e_n| + \frac{h}{2} \underline{Y},$$

so it looks like Euler's method for $y' = Ly + \frac{h}{2} \underline{Y}$. The last step will use stability of Euler's method for this equation.

Step 3: Use stability to bound e_n by $O(h)$ terms.

Recall $|e_n|$ satisfies:

$|e_0| = 0$, and

$$|e_{n+1}| \leq (1+hL)|e_n| + \frac{h^2}{2} \Upsilon, \quad \text{for } n=0,1,2,\dots$$

There are two possible ways to proceed here.

Method 1: Direct assault!

Backsolving this inequality gives:

$$|e_n| \leq (1+hL)|e_{n-1}| + \frac{h^2}{2} \Upsilon, \quad (\text{but } |e_{n-1}| \leq (1+hL)|e_{n-2}| + \frac{h^2}{2} \Upsilon)$$

$$\leq (1+hL) \left[(1+hL)|e_{n-2}| + \frac{h^2}{2} \Upsilon \right] + \frac{h^2}{2} \Upsilon$$

$$\quad (\text{but } |e_{n-2}| \leq (1+hL)|e_{n-3}| + \frac{h^2}{2} \Upsilon)$$

$$\leq (1+hL)^n |e_0| + \sum_{k=0}^{n-1} (1+hL)^k \frac{h^2}{2} \Upsilon,$$

Since $e_0 = 0$ this gives:

$$|e_n| \leq \frac{h^2}{2} \Upsilon \sum_{k=0}^{n-1} (1+hL)^k.$$

Method 2: Majorization.

This gives the same bound as above so we will sketch the steps.

~~Comparing~~

Viewing the difference inequality:

$$|e_0| = 0,$$

$$|e_{n+1}| \leq (1+hL) |e_n| + \frac{h^2}{2} \Upsilon,$$

suggests considering the difference equation

$$\phi_0 = |e_0|$$

$$\phi_{n+1} = (1+hL) \phi_n + \frac{h^2}{2} \Upsilon.$$

(This is exactly Euler's method for $\phi' = L\phi + \frac{h^2}{2}\Upsilon$, $\phi(0) = |e_0| = 0$.)

Lemma $|e_n| \leq \phi_n$ for every n .

proof: This is a very simple induction argument. \square

Now all that is needed is to solve the difference equality for ϕ . This is very easy to do. Its solution is:

$$\begin{aligned} \phi_n &= (1+hL)^n \phi_0 + \sum_{l=0}^{n-1} (1+hL)^l \frac{h^2}{2} \Upsilon \\ &= (\text{since } \phi_0=0) = \frac{h^2}{2} \Upsilon \sum_{l=0}^{n-1} (1+hL)^l. \end{aligned}$$

Step 4. Put the estimate in more agreeable form.

Consider our bound

$$|e_n| \leq \frac{h^2}{2} \Upsilon \sum_{l=0}^{n-1} (1+hL)^l.$$

The geometric series can be summed exactly.
This gives

$$|e_n| \leq \frac{h^2}{2} \Upsilon \left(\frac{(1+hL)^n - 1}{hL} \right) = \frac{h}{2} \Upsilon \left(\frac{(1+hL)^n - 1}{L} \right).$$

Now, note that

$$e^{hL} = 1 + hL + \frac{(hL)^2}{2!} + \dots$$


so that

$$(1+hL) \leq e^{hL}$$

or $(1+hL)^n \leq e^{L(nh)}$. Recall that $t_n = nh$.

This gives

$$|e_n| \leq \frac{h}{2} \Upsilon \left(\frac{e^{Lt_n} - 1}{L} \right),$$

proving the theorem. 

4.4 Runge-Kutta Methods.

Runge-Kutta methods attain the high accuracy of Taylor series methods without needing derivatives of f . The simplest interesting R-K method can be derived as follows. Suppose we seek an update formula of the form:

$$y_{n+1} = y_n + a k_1 + b k_2$$

where

$$(4.4.1) \quad k_1 = hf(t_n, y_n),$$

$$k_2 = hf(t_n + \alpha h, y_n + \beta k_1)$$

We determine a, b, α, β so that (4.4.1) will agree with as high order Taylor series method as possible.

Expand:

$$y(t_{n+1}) = y(t_n) + hy'(t_n) + \frac{h^2}{2}y''(t_n) + \frac{h^3}{6}y'''(t_n) + \dots$$

$$= y(t_n) + hf(t_n, y_n) + \frac{h^2}{2}(f_{tt} + ff_{yy})|_{(t_n, y_n)}$$

$$(4.4.2) \quad + \frac{h^3}{6}(f_{ttt} + 2ff_{ty} + f_{yy}f^2 + f_t f_y + f_y^2 f)|_{(t_n, y_n)} + O(h^4)$$

On the other hand, another Taylor expansion gives

$$\begin{aligned} \frac{k_2}{h} &= f(t_n + \alpha h, y_n + \beta k_1) \\ &= f(t_n, y_n) + \alpha h f_t + \beta k_1 f_y + \\ &+ \frac{\alpha^2 h^2}{2} f_{tt} + \alpha h \beta k_1 f_{ty} + \frac{\beta^2 k_1^2}{2} f_{yy} + O(h^3). \end{aligned}$$

Insert this in (4.4.1) for k_2 , and use $k_1 = hf(x_n, y_n)$. Collecting terms gives:

$$(4.4.3) \quad y_{n+1} = y_n + (a+b)hf + \underbrace{h^2}_{b}(\alpha f_t + \beta f f_y) + bh^3(\frac{\alpha^2}{2}f_{tt} + \alpha\beta f_{ty} + \frac{\beta^2}{2}f^2 f_{yy}) + O(h^4).$$

The coefficients a, b, α, β in (4.4.1) are chosen to make (4.4.3) as accurate as possible by matching (4.4.2) as far as possible. This gives the equations:

$$(4.4.4) \quad a + b = 1, \quad b\alpha = b\beta = \frac{1}{2}$$

This is three equations for four variables. Typically, there are an infinite number of solutions. One commonly used parameter set is:

$$a = b = \frac{1}{2}, \quad \alpha = \beta = 1.$$

Exercise: ^{4.4.1)} Verify that one further term cannot be matched.

Any of the solutions of (4.4.4) gives a second order RK method. This last solution is often called "the" second order RK formula:

Example: The RK2 method:

Given h and y_n compute :

$$\begin{aligned} k_1 &= hf(t_n, y_n) \\ k_2 &= hf(t_n + h, y_n + k_1) \\ y_{n+1} &= y_n + \frac{1}{2}k_1 + \frac{1}{2}k_2 \end{aligned}$$

The local error in this method is $O(h^3)$ and the global error $O(h^2)$. It is also quite inexpensive, costing only two function evaluations per time step.

There are, in principle, RK methods of every order accuracy. The most common ones are the RK4 method and the special RK-F methods of the next section.

Example: The RK4 Method.

Given h and y_n compute:

$$\begin{aligned} k_1 &= hf(t_n, y_n) \\ k_2 &= hf(t_n + \frac{h}{2}, y_n + \frac{1}{2}k_1), \\ k_3 &= hf(t_n + \frac{1}{2}h, y_n + \frac{1}{2}k_2), \\ k_4 &= hf(t_n + h, y_n + k_3), \\ y_{n+1} &= y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{aligned}$$

RK-4 is globally fourth order accurate ($O(h^4)$) and costs only four function evaluations per step.

The higher order RK methods are fantastically attractive in that:

- they are explicit, one step and high order methods.
- adaptive error control can be done easily and efficiently.

— Insert on page 13 —

Exercises

- 4.4. 2. Consider the nonlinear pendulum $\theta'' + \sin\theta = 0$, $\theta(0) = \pi/4$, $\theta'(0) = 0$. (a) Take $h = 1/2$ and compute an approximation to $\theta(1)$ using RK2. (b) Write the RK2 algorithm for a system of two equations in pseudocode.

4.4.3. COMPUTER EXERCISE

Reconsider your Euler's method program. Compute, at each step, both the Euler's method approximation and the RK2 approximation. Repeat the calculation you did before of the linear pendulum.

How do the results compare. For each t_n add a calculation of the following estimate of the error:

$$EST(t_n) = \sum_{k=0}^n \left| \frac{\text{EulerApprox}(t_n) - \text{RK2Approx}(t_n)}{h} \right|.$$

How close is this $\sum_{k=0}^n$ to the true error?

Their only drawback, which we will see later, is for so-called stiff initial value problems.

→ insert here ←

The Runge-Kutta-Fehlberg RKF Methods

Runge-Kutta methods are explicit, simple, higher order accurate, one-step methods. Thus, they hardly seem improvable. However, they are at least two very important further developments in Runge-Kutta methods. We shall see in a later section that diagonally implicit RK methods are an important tool for the types of problems ("stiff" systems) upon which all explicit methods fail. The other important extension is the RKF pairs.

RKF formulas are, like RK methods, high order accurate, one-step methods. They also give an automatic estimate of the local error at each step at no extra cost. Recall from the section on adaptive quadrature that the most basic way to estimate a local error is to compute two approximations of different accuracy. The number of significant digits of agreement is a conservative estimate of the accuracy of the **less** accurate approximation. (A more careful analysis can, as we will see, produce an estimate of the local error in the **more** accurate approximation.)

The brilliant idea of Fehlberg was to use the fact that the nonlinear equations for the coefficients in the RK formula, like (4.4.4) have an infinite number of solutions. He found pairs of RKF methods where the function evaluations of the lower order ones are used again for a higher order approximation. Thus, one can compute a lower and higher order approximation for the cost of the one higher order one alone.

The most popular RKF formula is the fourth and fifth order RKF pair.

ALGORITHM: The Fourth and Fifth Order RKF formulas.

These proceed as follows. Given y_n , compute a y_n and \bar{y}_n by the following fourth and fifth RKF methods:

$$k_1 = f(t_n, y_n),$$

$$k_2 = f\left(t_n + \frac{h}{4}, y_n + \frac{1}{4}h k_1\right),$$

$$k_3 = f\left(t_n + \frac{3}{8}h, y_n + h\left(\frac{3}{32}k_1 + \frac{9}{32}k_2\right)\right),$$

$$k_4 = f\left(t_n + \frac{12}{13}h, y_n + h\left(\frac{1932}{2197}k_1 - \frac{7200}{2197}k_2 + \frac{7296}{2197}k_3\right)\right),$$

$$k_5 = f\left(t_n + h, y_n + h\left(\frac{439}{216}k_1 - 8k_2 + \frac{3680}{513}k_3 - \frac{845}{4104}k_4\right)\right)$$

and

$$k_6 = f\left(t_n + \frac{1}{2}h, y_n + h\left(-\frac{8}{27}k_1 + 2k_2 - \frac{3544}{2565}k_3 - \frac{1859}{4104}k_4 - \frac{11}{40}k_5\right)\right)$$

After these six function evaluations, compute

$$y_{n+1} = y_n + h \left[\frac{25}{216} k_1 + \frac{1408}{2565} k_3 + \frac{2197}{4104} k_4 - \frac{1}{5} k_5 \right]$$

$$\bar{y}_{n+1} = y_n + h \left[\frac{16}{135} k_1 + \frac{6656}{12825} k_3 + \frac{28561}{56430} k_4 - \frac{9}{50} k_5 + \frac{2}{55} k_6 \right]$$

This requires four function evaluations to get an approximation y_{n+1} to $y(t_{n+1})$. With two extra ones we obtain a higher order approximation \bar{y}_{n+1} and the following estimate of the local error

$$y_{n+1} - y(t_{n+1}) \doteq -h \left[\frac{1}{360} k_1 - \frac{128}{4275} k_3 - \frac{2197}{75240} k_4 + \frac{1}{50} k_5 + \frac{2}{55} k_6 \right].$$

→ insert here ←

4.5 Adaptive Time-Step Selection.

This chapter stresses one step methods over (often more efficient) multistep methods because of their ease of use in an adaptive program. The modern view of scientific software is that a program should not just compute a table of numbers "approximating" something. It should compute an approximation with guaranteed accuracy! It is, of course, possible to use multistep methods adaptively. In fact, adaptively variable step and order multistep methods are near the leading edge in initial value problem solvers. Nevertheless, one step methods are quite efficient and the ideas of an adaptive program based on a single step method are very simple and clean-hence our focus on them here.

As usual, adaptivity needs two ingredients : a method of estimating errors and a strategy for changing the step size in response to those estimates.

Estimating Local Errors

Suppose we want the global error $< \varepsilon$, e.g.,

$$\max |y(t_n) - y_n| < \varepsilon, \quad t_N = N \cdot h = T, 0 \leq t_n \leq T$$

then we wish to make the local errors/unit step $< N\varepsilon$. So we must estimate the local errors.

Example: Euler's method.

The local error is given by

$$(4.5.1) \quad E_L = \frac{1}{2} h^2 y''(\xi)$$

so where y'' is large we must take h small and vice versa. We can estimate (4.5.1) directly since

$$\begin{aligned} y' &= f(t, y(t)) \\ y''(t) &= f_t(t, y(t)) + f_y(t, y(t)) f(t, y(t)) \end{aligned}$$

— Insert on p.14 —

Exercise

HH4 (a) Find an estimator like the above RK4-5 pair using RK1 (Euler's method) and RK2, for $y' = f(t, y)$

(b) Repeat (a) for a system of two equations: $x' = f(t, x, y)$, $y' = g(t, x, y)$.

Thus given, $y_n \cong y(t_n)$ we pick h_{new} such that $\frac{1}{2}h_{new}^2 \{|f_t(t_n, y_n) + f_y(t_n, y_n)f(t_n, y_n)|\} \leq N\epsilon$ and calculate $y_{n+1} \cong y(t_n + h_{new})$ by:

$$y_{n+1} = y_n + h_{new} f(t_n, y_n).$$

This method doesn't work well for systems of equations or higher order methods; e.g., for 2nd order Runge-Kutta the local error is

$$\frac{1}{6}h^3[\frac{3}{2}(f_{yy}f^2 + 2f_{ty}f + f_{tt} - f_t - ff_y)]$$

which is not so easy to compute. We usually settle for a more easily computed estimate for the **local error**. We now consider two methods for doing this.

Method 1. We calculate an approximate to $y(x_{n+1})$ twice, using a method $O(h^p)$ accurate and an $O(h^q)$ method $q > p$, giving \bar{y}_{n+1} .

The first local error is expanded by Taylor series as

$$E_{1L} = \tau_n h^{p+1} + O(h^{p+2}),$$

and similarly $E_{2,L}$. Ignoring the error at t_n (i.e., assuming $y_n \cong \bar{y}_n$) we expand

$$\begin{aligned} y_{n+1} &= y(t_{n+1}) + \tau_n h^{p+1} + O(h^{p+2}) \\ \bar{y}_{n+1} &= y(t_{n+1}) + \bar{\tau}_n h^{q+1} + O(h^{q+2}), \text{ subtract:} \\ y_{n+1} - \bar{y}_{n+1} &= (\tau_n - \bar{\tau}_n h^{q-p})h^{p+1} + O(h^{p+2}), q > p, \\ y_{n+1} - \bar{y}_{n+1} &\cong \underbrace{\tau_n h^{p+1}}_{E'_{1L} \text{ s leading order term}} + O(h^{p+2}), (\text{recall } q > p). \end{aligned}$$

Thus, if $p < q$ we may take as an approximation to $E_{1,L}$

$$E_{1,L} \cong y_{n+1} - \bar{y}_{n+1} \quad (+\text{Higher Order Terms})$$

Exercise: ^{4.5.1} If $p = q$ we must be more careful. Show that we can estimate the local error as

$$\bar{E}_L \cong [\bar{\tau}_n / (\tau_n - \bar{\tau}_n)](y_{n+1} - \bar{y}_{n+1}).$$

Example: Fourth and Fifth Order Runge-Kutta Fehlberg formulas

$$y_{n+1} = y_n + h \left[\frac{25}{216} k_1 + \frac{1408}{2565} k_3 + \frac{2197}{4104} k_4 + \frac{1}{5} k_5 \right]$$

$$\bar{y}_{n+1} = y_n + h \left[\frac{16}{135} k_1 + \frac{6656}{12825} k_3 + \frac{28561}{56430} k_4 - \frac{9}{50} k_5 + \frac{2}{55} k_6 \right]$$

where,

$$k_1 = f_n,$$

$$k_2 = f\left(t_n + \frac{h}{4}, y_n + \frac{1}{4} h k_1\right),$$

$$k_3 = f\left(t_n + \frac{3}{8} h, y_n + h\left(\frac{3}{32} k_1 + \frac{9}{32} k_2\right)\right),$$

$$k_4 = f\left(t_n + \frac{12}{13} h, y_n + h\left(\frac{1932}{2197} k_1 - \frac{7200}{2197} k_2 + \frac{7296}{2197} k_3\right)\right),$$

$$k_5 = f\left(t_n + h, y_n + h\left(\frac{439}{216} k_1 - 8k_2 + \frac{3680}{513} k_3 - \frac{845}{4104} k_4\right)\right)$$

and

$$k_6 = f\left(t_n + \frac{1}{2} h, y_n + h\left(-\frac{8}{27} k_1 + 2k_2 - \frac{3544}{2565} k_3 + \frac{1859}{4104} k_4 - \frac{11}{40} k_5\right)\right).$$

Using these, we have an estimate for the local error in y_{n+1} :

$$y_{n+1} - y(t_{n+1}) \doteq y_{n+1} - \bar{y}_{n+1} = -h \left(\frac{1}{360} k_1 - \frac{128}{4275} k_3 - \frac{2197}{75240} k_4 + \frac{1}{50} k_5 + \frac{2}{55} k_6 \right).$$

Method 2 In this approach, we compute two approximations to $y(t_{n+1})$ with a single $O(h^p)$ accurate method by halving the stepsize. Let

$$y_{n+1} = y(t_{n+1}) + \tau_n h^{p+1} + O(h^{p+2}),$$

Then, since \bar{y}_{n+1} is computed with two steps of size $h/2$:

$$\bar{y}_{n+1} = y(t_{n+1}) + 2\tau_n \left(\frac{h}{2}\right)^{p+1} + O\left(\left(\frac{h}{2}\right)^{p+2}\right),$$

Thus, we have by subtraction $y_{n+1} - \bar{y}_{n+1} = 2\tau_n \left(\frac{h}{2}\right)^{p+1} \{2^p - 1\} + \text{H.O.T.}$, and:

$$(4.5.2) \quad 2\tau_n \left(\frac{h}{2}\right)^{p+1} \doteq \text{local error in } \bar{y}_{n+1} \cong \frac{1}{2^p - 1} (y_{n+1} - \bar{y}_{n+1}) (+ \text{higher order terms}).$$

This method is particularly simple and produces an estimate of the error in the more accurate approximation.

Step-Size Control

Given an estimate of a local error in the n^{th} step, EST, the strategy is to keep the local error per unit below the preset tolerance by cutting h . If EST is so small

that h can be increased and still keep the local error per unit step smaller than the tolerance then h is increased.

The simplest implementation of this idea is mesh halving and doubling. Suppose we are using an $O(h^p)$ accurate method. Then, the local error is $O(h^{p+1})$ and if h is doubled/halved the error in one step is changed by $2^{p+1}/\frac{1}{2^{p+1}}$. Given our preset tolerance ToL , we compute $ToL' = ToL/2^{p+2}$. We seek to maintain:

$$ToL' = ToL/2^{p+2} < \frac{Est}{h} < ToL$$

There are three cases:

Case 1. $ToL' < \frac{Est}{h} < ToL$.

Accept y_{n+1} and continue onto the next time step with the same h .

Case 2. $\frac{Est}{h} \geq ToL$.

The error is too large. Return to (t_n, y_n) and reduce h , say $h \leftarrow \frac{h}{2}$, and recompute y_{n+1} .

Case 3. $\frac{Est}{h} \leq ToL$.

The error is too small so the program is performing more calculations than needed. Accept (t_{n+1}, y_{n+1}) but an increase h , say $h \leftarrow 2h$, to compute y_{n+2} .

Remark. It should be clear that we don't have to restrict the program to mesh halving and doubling.

For example, suppose we compute one estimate by (4.5.2) by computing

$$y_{n+1} = \text{One Step with } O(h^p) \text{ method with } \Delta t = h,$$

$$\bar{y}_{n+1} = \text{Two steps of } O(h^p) \text{ method with } \Delta t = h/2,$$

$$EST(\bar{y}_{n+1}) = \frac{|\bar{y}_{n+1} - y_{n+1}|}{2^{p+1} - 1}.$$

If the local error in y_{n+1} is

$$y(t_{n+1}) - y_{n+1} \doteq C_n h^{p+1}$$

then the local error in \bar{y}_{n+1} is roughly

$$y(t_{n+1}) - \bar{y}_{n+1} \doteq 2C_n \left(\frac{h}{2}\right)^{p+1} = C_n \frac{h^{p+1}}{2^p}.$$

Since the local EST (\bar{y}_{n+1}) is computed, we can get a rough estimate of the unknown constant " C_n " in the above:

$$EST(\bar{y}_{n+1}) \doteq C_n \frac{h^{p+1}}{2^p} \rightarrow C_n \doteq \frac{2^p}{h^{p+1}} EST(\bar{y}_{n+1})$$

To pick the new stepsize h_{new} so that the new error per unit step is roughly ToL set

$$\frac{\text{New Error}}{\text{New Step}} \doteq \frac{2 C_n h_{New}^{p+1}}{h_{new} 2^{p+1}} \doteq ToL.$$

Solving for h_{new} gives:

$$h_{new} \doteq 2 \left[\frac{ToL}{C_n} \right]^{1/p} \doteq 2 \left[\frac{ToL}{\frac{2^p}{h^{p+1}} EST(\bar{y}_{n+1})} \right]^{1/p}.$$

Simplifying gives:

$$h_{new} \doteq h_{old} \left[h_{old} \frac{ToL}{EST(\bar{y}_{n+1})} \right]^{1/p}$$

4.6 Stiffness and Implicit Methods

We have seen that explicit methods (such as Euler's method) converge over bounded time intervals for h small enough. This is certainly important but it leaves open two unanswered questions:

- How small must h be for the error to begin decreasing?
- What happens to the error and the approximate solution as the problem is solved over longer and longer time intervals?

Numerical analysts search for the most simple example for which these two questions are interesting. This test problem is:

$$(4.6.1) \quad y'(t) = +\lambda y(t), y(0) = 1, \text{ where } \lambda < 0.$$

The solution is:

$$y(t) = e^{\lambda t} \rightarrow 0 \quad t \rightarrow \infty.$$

The more negative λ is, the faster $y(t) \rightarrow 0$. Thus, any growth in an approximate solution to (4.6.1) is a serious error. More generally, we can consider the test problem (4.6.1) with

$$Re(\lambda) < 0$$

since then too

$$y(t) = e^{\lambda t} = e^{Re(\lambda)t} (\cos(Im(\lambda)t) + i \sin(Im(\lambda)t)) \rightarrow 0$$

as $t \rightarrow \infty$. Let us consider a few examples applied to (4.6.1)

Example 1: Euler's Method This reads:

$$\frac{y_{n+1} - y_n}{h} = \lambda y_n, y_0 = 1,$$

or $y_n = (1 + h\lambda)^n$. For $|y_n| \rightarrow 0$ as $t_n \rightarrow \infty$ we must have $|1 + h\lambda| < 1$. If λ is real this means $-1 < |1 + h\lambda| < +1$ or :

$$-2 < h\lambda < 0, \text{ for } \lambda \text{ real.}$$

If $\lambda = -4$ then for stability over $(0, \infty)$ we must have

$$h < \frac{1}{2} \quad \text{for stability with } \lambda = -4.$$

This is no serious restriction. Consider however $\lambda = -1000$ the true solution is e^{-1000t} . It approaches zero so fast it isn't very interesting. On the other hand, Euler's method requires

$$h < \frac{1}{500} \quad \text{for stability with } \lambda = -1000.$$

If $h > \frac{1}{500}$ the approximate solution will blow up exponentially as $t_n \rightarrow \infty$!

The bad side of $h < \frac{1}{500}$ is that we are forced to pick a very small stepsize for stability rather than to attain a desired accuracy.

Example 2: RK 4

By a similar calculation it can be shown RK4 will be stable as $t_n \rightarrow \infty$ if:

$$-2.8 < h\lambda < 0.$$

The RK4 approximation will blow up exponentially if h isn't small enough! This blowup is especially common with the systems of equations.

Example 3: Consider the second order IVP:

$$y'' + 1001y' + 1000y = 0, \quad y(0) = 1, \quad y'(0) = -1.$$

The solution is

$$y(t) = C_1 e^{-t} + C_2 e^{-1000t},$$

where $C_{1,2}$ are picked to satisfy the initial conditions. If this is written as a first order system: ($y_1 = y, y_2 = y'$)

$$\begin{aligned} y_1' &= y_2 & , y_1(0) &= 1 \\ y_2' &= -1001y_2 - 1000y_1 & , y_2(0) &= -1, \end{aligned}$$

it can be approximated by RK4. The RK4 approximate solution will converge nicely if $h \leq 0.002$ and smaller and blow-up if $h \geq 0.003$. This system is said to be stiff. It's stiffness is measured by the eigenvalues of the coefficient matrix:

$$\begin{pmatrix} 0 & 1 \\ -1000 & -1001 \end{pmatrix}, \quad \lambda_1 = -1000, \lambda_2 = -1.$$

This is typical behavior of a numerical method for a stiff system: the scheme blows up and up and up until the mesh width is reduced to a ridiculously small size. Many **implicit** methods, such as the backward Euler method:

Backward Euler: $\frac{y_{n+1} - y_n}{h} = f(t_{n+1}, y_{n+1})$,

are stable and convergent for all mesh sizes. On the other hand, at each step a nonlinear system must be solved.

With these examples in mind, consider a one step method applied to (4.6.1). The one step method produces an approximate solution $y_n = \beta^n$ where $\beta = \beta(h\lambda)$.

Definition. A one step method applied to (4.6.1) is

- **strongly stable** if $|\beta| < 1$ for all negative real λ
- **A-Stable** if $|\beta| < 1$ for all complex λ with $Re(\lambda) < 0$.

Definition. The **stability region** of a onestep method is the region in the complex plane:

$$\{z : z = h\lambda \text{ and } |\beta(h\lambda)| < 1\}.$$

For stiff systems, it is clear that A-stable methods are the ones that should be used. Unfortunately, there are no A-stable, explicit methods.

Theorem 4.6.1. [Dahlquist]

- There are no A-stable, explicit single or linear multi-step methods.
- A single or multi-step method that is A-stable has at most second order accuracy.
- The A-stable method with the smallest truncation error is the trapezoid rule.

Example 3. **The trapezoid rule:** If we integrate (4.6.1) from t_n to t_{n+1} :

$$y_{n+1} - y_n = \int_{t_n}^{t_{n+1}} \lambda y dt.$$

If the integral is approximated by trapezoid rule the "trapezoid rule" for numerical integration arises:

$$y_{n+1} - y_n = h \lambda \left(\frac{y_{n+1} + y_n}{2} \right).$$

For the nonlinear problem $y' = f(t, y)$ there are two natural interpretations (called a "two-leg" and "one-leg" trapezoid rule):

$$\frac{y_{n+1} - y_n}{h} = (f(t_n, y_n) + f(t_{n+1}, y_{n+1}))/2, \text{ ("two-leg")}$$

$$\frac{y_{n+1} - y_n}{h} = f\left(\frac{t_n + t_{n+1}}{2}, \frac{y_n + y_{n+1}}{2}\right), \text{ ("one-leg")}$$

The trapezoid rule has some amazing properties. For example, its stability region is exactly the same as for the continuous initial value problem:

$$\{z : Re(z) < 0\}$$

Examples.

The stability regions for RK1 (which is Euler's method), RK2, RK3 (which we have not derived) and RK4 can be derived by applying the method to

$$y' = \lambda y, \quad y(0) = 1, \quad \lambda = \alpha + i\beta$$

The stability region is the set of $z = h\lambda \in \mathbb{C}$ such that the RK approximation $\rightarrow 0$ as $tn \rightarrow \infty$. These can be easily calculated (using Maple for example) to be:

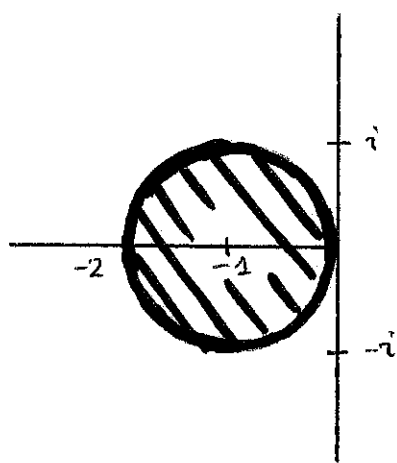


Figure: Stability Region of RK1 = Euler's Method.

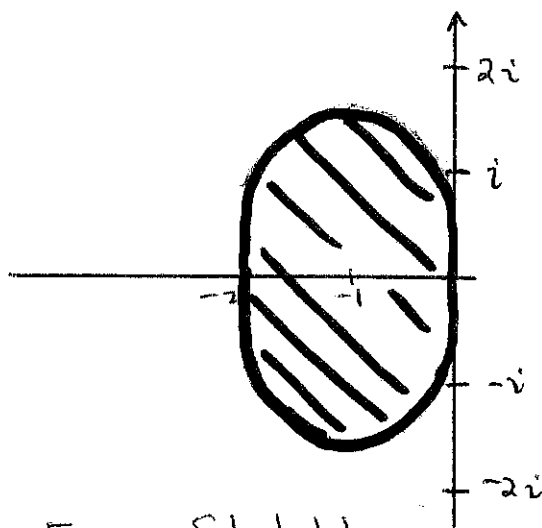
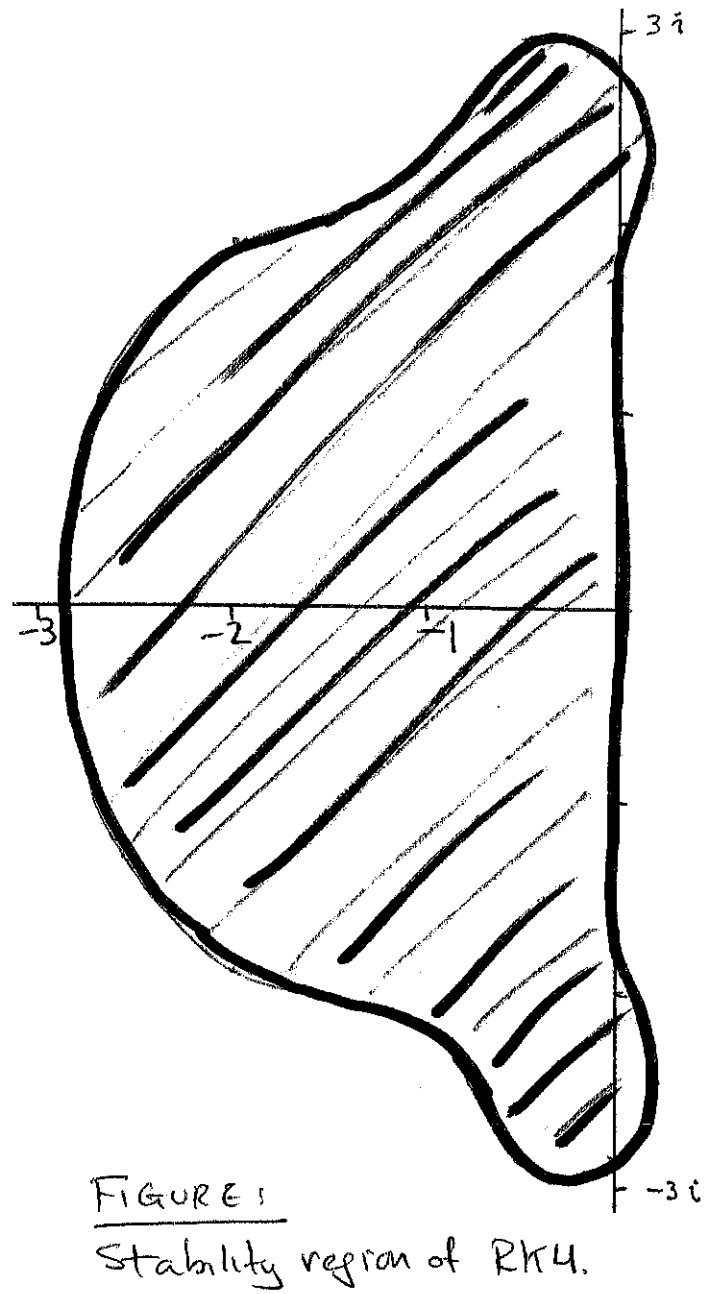
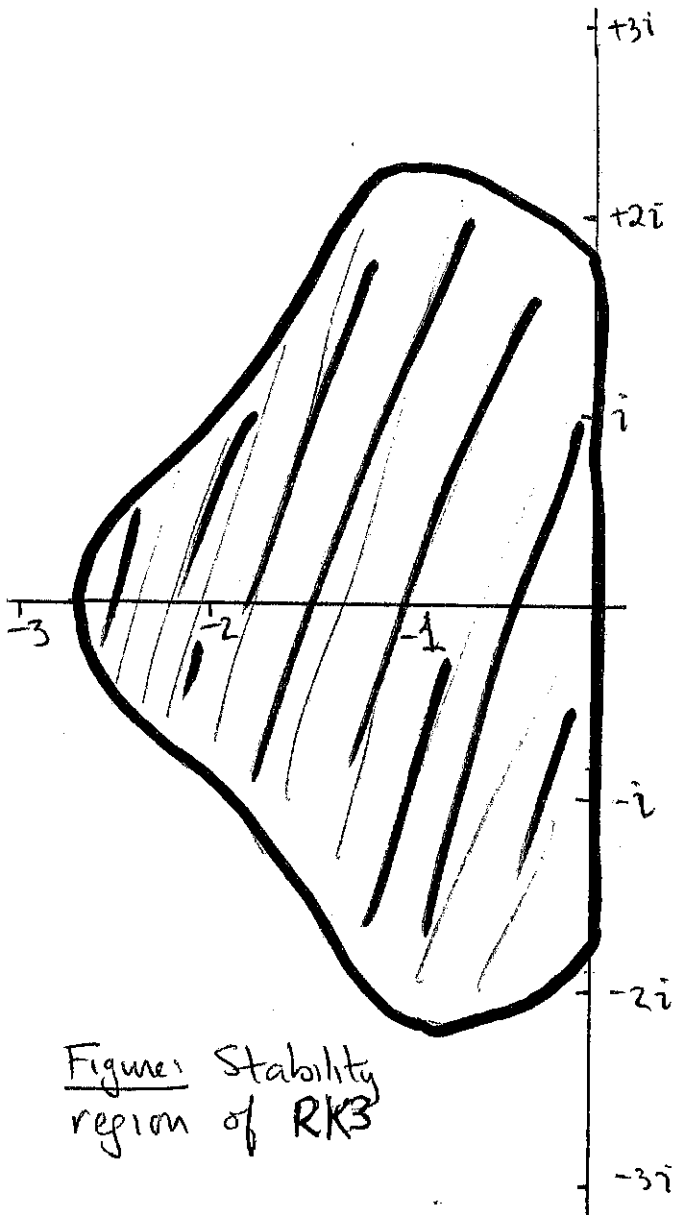


Figure: Stability region of RK2



There are other good choices for stiff systems - the "diagonally implicit Runge-Kutta methods." These are all implicit methods. There are some interesting possibilities for solving the associated nonlinear system.

Solving the Nonlinear System

→ Insert page here ←

Suppose we are solving the IVP

$$y' = f(t, y), 0 < t \leq T, y(0) = y_0,$$

by the (for example) "two-leg" form of the trapezoid rule:

$$\frac{y_{n+1} - y_n}{h} = \frac{1}{2}f(t_n, y_n) + \frac{1}{2}f(t_{n+1}, y_{n+1}).$$

At each time step we must solve the nonlinear system

$$(4.6.2) \quad F(y) := y - \frac{h}{2}f(t_{n+1}, y) = y_n + \frac{h}{2}f(t_n, y_n).$$

for $y = y_{n+1}$. One simple method is to use a simple iteration:

$$(4.6.3) \quad y^{new} - \frac{h}{2}f(t_{n+1}, y^{old}) = y_n + \frac{h}{2}f(t_n, y_n).$$

and, upon convergence, set $y_{n+1} \leftarrow y^{new}$. If h is "small" enough this is often all that is required.

Theorem. Suppose $f(t_{n+1}, y)$ is Lipschitz continuous:

$$(4.6.4) \quad |f(t_{n+1}, y) - f(t_{n+1}, z)| \leq L|y - z|.$$

If $\frac{hL}{2} < 1$, or, equivalently,

$$h < \frac{2}{L}$$

the simple iteration (4.6.3) converges for y_{n+1} for any initial guess.

Proof: This follows from the contraction mapping theorem applied to (4.6.3).

The simple iteration (4.6.3) requires an initial guess and a stopping criteria. We can begin with $y^{old} = y_n$ but it's better to **predict** y_{n+1} with one step of an explicit method. The stopping criteria should be a **relative** one since solutions can grow or decay exponentially - hence are not $O(1)$. The combination looks like:

Given y_n and stopping tolerance TOL

Predict $y_{n+1}^{old} = y_n + h f(t_n, y_n)$

→ Insert on page 21 ←

Exercises

4.6.1. Consider $y' = -10y$, $y(0) = 1$.

Show that $y(t) \rightarrow 0$ as $t \rightarrow \infty$.

(a) Write down the RK2 approximation to this equation and show that $|Y_n| \rightarrow 0$ as $n \rightarrow \infty$ if h is small enough but $|Y_n| \rightarrow \infty$ as $n \rightarrow \infty$ if h is too large.

(b) How small is "small enough"?

4.6.2 (a) Repeat problem 1 for the predictor-corrector method:

$$Y_{n+1}^P = Y_n + h f(t_n, Y_n),$$

$$Y_{n+1} = Y_n + \frac{1}{2}h f(t_n, Y_n) + \frac{1}{2}h f(t_{n+1}, Y_{n+1}^P).$$

(b) Write down a pseudo code realization of this method for solving $x' = f(t, x, y)$, $y' = g(t, x, y)$.

4.6.3. Consider the trapezoid rule for $y' = f(t, y)$:

$$Y_{n+1} - Y_n = \frac{h}{2} (f(t_n, Y_n) + f(t_{n+1}, Y_{n+1})).$$

Show that it is strongly stable.

4.6.4. Repeat problem 3 for the backward Euler method. Why is strong stability important?

Compute

$$y_{n+1}^{new} - \frac{h}{2} f(t_{n+1}, y_{n+1}^{old}) = y_n + \frac{h}{2} f(t_n, y_n)$$

Stop when:

$$\frac{|y_{n+1}^{new} - y_{n+1}^{old}|}{|y_{n+1}^{new}|} < TOL$$

Such a process:

Predict with explicit, unstable method,

Correct with implicit, stable method.

is called a "predictor-corrector" method. This particular combination is known as "modified Euler" or "Heun's method" if we stop after only Δ correction step.

The above combination can still fail to converge for "stiff" problems. Reconsider $y' = \lambda y$, i.e.,

$$f(t, y) = \lambda y \rightarrow \text{Lipschitz constant} = |\lambda|.$$

If λ is **large** and **negative** then the condition $h < \frac{2}{|\lambda|}$ is exactly the sort of time-step restriction we are seeking to avoid by using an A-stable, implicit method!

Fortunately, Newton's method works extremely well for systems like (4.6.2) when $\frac{\partial f}{\partial y} < 0$. Newton's method for (4.6.2) reads:

$$y^{new} = y_{old} + (F'(y^{old}))^{-1} [y^{old} - \frac{h}{2} f(t_{n+1}, y^{old}) - y_n - \frac{h}{2} f(t_n, y_n)],$$

where $F'(\cdot)$ is given by:

$$F'(y^{old}) = 1 - \frac{h}{2} \frac{\partial f}{\partial y}(t_{n+1}, y^{old}).$$

Note that if $\frac{\partial f}{\partial y}$ is **negative**, $F'(y^{old})$ is positive and the more negative $\frac{\partial f}{\partial y}$ is, the more positive $F'(y^{old})$ becomes.

Thus, even the most stiff IVP's will succumb to the overall strategy of PREDICTION for initial guess, Newton's method for nonlinear system and Relative update for stopping criteria.

4.6.5. COMPUTER EXERCISE

Consider the following system of two equations modeling the population levels of Rabbits ($R(t) \sim x(t)$) and Foxes ($F(t) \sim y(t)$): ~~the~~

$$R'(t) = R(t) - R(t)F(t), \quad R(0) = 3$$

$$F'(t) = -F(t) + R(t)F(t), \quad F(0) = 1.$$

The ~~the~~ solutions ~~is~~ ^{is} ~~are~~ periodic for positive initial data. Try using your Euler-RK2 program to solve this over $0 \leq t \leq 20$. How does the error behave as t increases?

Try solving it again using the adaptive Euler-RK2 program. How does the error behave with t increasing? ~~look~~ ^{look} carefully.

~~Plot~~ Plot a graph of t vs. number of steps N . ~~Can~~ Compare and try to explain the results you see!

5. Curve Fitting.

5.1 Introduction.

In this section, we consider the following problem:

$$(5.1.1) \quad \begin{array}{l} \text{Given data } x_j, y_j \quad j = 0, \dots, N, \text{ find} \\ \text{a polynomial } p_N(x) \text{ of degree } \leq N \text{ such that} \\ p_n(x_j) = y_j. \end{array}$$

We then say that " $p_N(x)$ interpolates y at the data points $\{x_j\}$ ". The coefficients of the polynomial can, in principal at least, be determined directly. Indeed, expand $p_N(x)$

$$p_N(x) = a_0 + a_1x + \dots + a_Nx^N$$

and insert into (5.1.1):

$$\begin{array}{l} p_N(x_0) \equiv a_0 + a_1x_0 + \dots + a_Nx_0^N = y_0 \\ \vdots \\ p_N(x_N) \equiv \underbrace{a_0 + a_1x_N + \dots + a_Nx_N^N}_{(N+1) \times (N+1) \text{ Linear system}} = y_N \end{array}$$

$$(5.1.2) \quad \begin{array}{l} V\bar{a} = \bar{y}, V_{ij} = x_i^j, i, j = 0, \dots, N, \\ \bar{a} = (a_0, \dots, a_N)^T, \bar{y} = (y_0, \dots, y_N)^T. \end{array}$$

Here V is called the Van der Monde matrix.

Example: Suppose we seek a cubic fit $p_3(x) = a_0 + a_1x + a_2x^2 + a_3x^3$ of data measured at $x = 0, 30$ feet, 60 feet and 90 feet. The Van der Monde matrix resulting from these points is approximately:

$$V = \begin{bmatrix} 0 & 0 & 0 & 1.0 \\ 3 \times 10^4 & 9 \times 10^2 & 30 & 1.0 \\ 2 \times 10^5 & 4 \times 10^3 & 60 & 1.0 \\ 7 \times 10^5 & 8 \times 10^3 & 90 & 1.0 \end{bmatrix}$$

Note the widely varying sizes of the entries. Solution of any linear system with this coefficient matrix introduces, due to these widely varying coefficients, so much roundoff error that any result is extremely untrustworthy.

Theorem 5.1.1. Suppose $x_i \neq x_j$ for all i, j . Then,

$$\det V \neq 0.$$

That is, (1) has a unique solution. \square

However, solving the linear system (5.1.2) to solve (5.1.1) is not realizable because V is almost always very unstable (ill-conditioned). Thus, we cannot solve (5.1.2) in practice and expect any accuracy at all and we must search for other methods of finding $p_N(x)$.

5.2

The Interpolating Polynomial.

Polynomials are so simple to manipulate by hand but they are very difficult for the computer to work with. We now review some basic properties.

Example: Loss of accuracy:

Suppose we have the data

x_j	6000	6001
y_j	1/3	2/3

then in 5 decimal place arithmetic,

$$p_1(x) = 6000.3 - x$$

But $p(x)|_{x=6000} = 0.3$, $p_1(x)|_{6001} = -0.7!$

Efficiency: Horner's Rule.

If we evaluate $p_N(x)$ directly and count the "flops":

$$p_N(x) = a_0 + \underbrace{a_1 x}_1 + \underbrace{a_2 x^2}_2 + \underbrace{a_3 x^3}_3 + \dots +$$

$+ \dots + N = O(N^2)$ multiplications,

N additions.

Compare this with Horner's method

$$p_N(x) = a_0 + \underbrace{x[a_1 + x[a_2 + \cdots + x_1[a_{N-1} + a_N x]] \cdots]}_{\substack{N \text{ multiplications} \\ N \text{ addition}}}$$

Control of Roundoff Error: Newton's Form.

We pick centers $c_1 \cdots c_N$ and rewrite $p_N(x)$ as:

$$p_N(x) = a_0 + a_1(x - c_1) + a_2(x - c_1)(x - c_2) + \cdots + a_n(x - c_1)(x - c_2) \cdots (x - c_n)$$

and evaluate it via:

$$p_N(x) = a_0 + (x - c_1)[a_1 + (x - c_2)[a_2 + \cdots + (x - c_{N-1})[a_{N-1} + a_N(x - c_N)] \cdots]].$$

where we choose the centers to minimize roundoff.

5.2.1.
Exercise: Show that if $c_1 = c_2 = \cdots = c_N = c$ then

$$a_i = P_N^{(i)}(c)/(i!) \quad i = 0, \dots, N.$$

Algorithm: [Nested Multiplication for Newton Form].

Given coefficients a_0, \dots, a_n and centers c_1, \dots, c_n and the number z .

$$\tilde{a}_n := a_n$$

For $i = n - 1, n - 2, \dots, 0$, Do:

$$\tilde{a}_i := a_i + (z - c_{i+1})\tilde{a}_{i+1}.$$

Then, $\tilde{a}_0 = p_n(z)$.

The Lagrange Form of the Interpolant.

Consider the interpolation problem: Find $p_n(x)$ such that

$$(5.2.1) \quad p_n(x_j) = y_j, \quad j = 0, \dots, n,$$

Lagrange found the solution to this via writing $p_n(x)$ in the form

$$(5.2.2) \quad p_n(x) = \alpha_0 \ell_0(x) + \alpha_1 \ell_1(x) + \dots + \alpha_n \ell_n(x)$$

where

$$(5.2.3) \quad \ell_k(x) = \prod_{\substack{i=0 \\ i \neq k}}^n \frac{(x - x_i)}{(x_k - x_i)}$$

are the Lagrange polynomials for the points x_0, \dots, x_n . Note that ℓ_k has the properties:

$$\text{degree } \ell_k(x) = n,$$

$$\ell_k(x_k) = 1,$$

$$\ell_k(x_j) = 0, \quad j \neq k.$$

Thus, the solution to (1) is (2) where

$$(5.2.4) \quad \alpha_k = p_n(x_k) = y_k, \text{ i.e.} \\ p_n(x) = \sum_{k=0}^n y_k \ell_k(x).$$

Theorem. If the points $\{x_k\}$ are all distinct then (5.2.1) has a unique solution, given by

(5.2.4). \square

5.2.2
Exercise: Prove that the solution is unique.

Cost of Evaluation (5.2.4). Once the denominators are calculated and divided into y_k ,

(5.2.4) costs, using Horner's rule,

$(2n + 1)$ multiplications and $(2n + 1)$ additions.

Further, if a new data point is added then we must begin from scratch and recalculate all the coefficients.

Remark: This is a very elegant solution to the interpolation problem. Lagrange found this at age 16!

Newton's Solution to the Interpolation Problem

Newton derived the following formula, which gives the solution of the interpolation problem:

$$(5.2.5) \quad p_n(x) = f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + \cdots \\ + f[x_0, x_1, \dots, x_n](x - x_0)(x - x_1) \cdots (x - x_{n-1}).$$

Here the $f[\cdot, \cdot, \cdot]$'s are the divided differences of the function $f(x)$, defined via

$$(5.2.6) \quad f[x_j, \dots, x_{j+k}] = \frac{f[x_{j+1}, \dots, x_{j+k}] - f[x_j, \dots, x_{j+k-1}]}{x_{j+k} - x_j}$$

where $f[x_j] = f(x_j) = y_j$ begins the difference table.

Theorem. The solution to the interpolation problem (5.2.1) is given by (1) above, where the coefficients are given through (5.2.6).

We postpone the proof of the above for a moment while we discuss its implementation and work an example.

Example: Given the following data, construct the difference table and write down the interpolating polynomial.

x_j	$f(x_j)$ y_j	$f[x_0, j]$	$f[x_0, j]$	$f[x_0, j]$	$f[x_0, j]$	$f[x_0, j]$
0	3					
1	7	4				
2	-1	-8	-6			
3	2	3	$11/2$	$23/6$		
4	1	-1	-2	$-15/6$	$-19/12$	
5	3	2	$3/2$	$7/6$	$22/24$	$1/2$
	\otimes	\otimes	\circ	\circ	\circ	\circ

New data points New calculations needed

Note that only the underlined numbers need be stored to process a new data point.

The solution to the interpolation problem is then

$$p_5(x) = 3 + 4(x-0) - 6(x-0)(x-1) + \frac{23}{6}(x-0)(x-1)(x-2) - \frac{19}{12}(x-0)(x-1)(x-2)(x-3) + \frac{1}{2}(x-0)(x-1)(x-2)(x-3)(x-4)$$

The important algorithmic fact about Newton's discovery is that if a new data point is added, the table need not be recomputed. In fact, only the last diagonal need be stored for use with new data points.

Algorithm. This computes $f[x_0, \dots, x_k], k = 0, \dots, n$.

- | |
|---|
| <ol style="list-style-type: none"> 1. Input $\{x_0, \dots, x_n, f_0, \dots, f_n\}$ 2. For $J = 0, \dots, n$
 $d_j := f_j$ 3. For $k = 0, \dots, n - 1$
 For $j = n, n - 1, \dots, k + 1$
 $d_j := (d_j - d_{j-1}) / (x_j - x_{j-k})$ 4. Output $\{d_1, \dots, d_{n+1}\}$ |
|---|

This algorithm costs

$$n^2 + n \text{ additions}$$

$$(n^2 + n) / 2 \text{ divisions}$$

The Newton form of the interpolant has the tremendous advantage that if a new point is added only the coefficient $f[x_0, \dots, x_n, \bar{x}]$ need be computed, i.e. all the previous coefficients remain the same.

The error $f(x) - p_n(x)$ is described by the following theorem.

Theorem. Let $p_n(x)$ interpolate $f(x)$ at x_i on $[a, b], i = 0, \dots, n$. Suppose $f(x)$ is $C^{n+1}[a, b]$, then for any $\bar{x} \in [a, b]$ there is a $\xi \in [a, b]$ such that

$$f(\bar{x}) - p_n(\bar{x}) = \left[\frac{f^{n+1}(\xi)}{(n+1)!} \right] (\bar{x} - x_0) \cdots (\bar{x} - x_n). \quad \square$$

This theorem has several interesting consequences:

1. Convergence for "smooth" f follows on bounded intervals.
2. Extrapolation is unstable.
3. The best choice of the x_i are the *Chebyshev points*. On $[a, b]$, these are given by

$$x_i = \left[a + b + (a - b) \cos \left(\frac{2i + 1}{2n + 2} \pi \right) \right] / 2, \quad i = 0, \dots, n$$

This choice minimizes $\max |\psi_n(\bar{x})| = |(\bar{x} - x_0) \cdots (\bar{x} - x_n)|$

— Insert page here ←

5.2 Least Squares Approximations.

Given a table of data $x_j, f_j, j = 1, \dots, N$ the least squares approximation is defined as follows. We specify functions $\phi_1(x), \dots, \phi_k(x)$ and positive weights w_1, \dots, w_N . The approximation is determined via

$$F(x) = c_1 \phi_1(x) + \cdots + c_k \phi_k(x)$$

where the c 's are chosen to minimize the deviation:

$$\|\bar{d}\|_2^2 = \sum_{j=1}^N w_j [f_j - (c_1 \phi_1(x_j) + \cdots + c_k \phi_k(x_j))]^2.$$

This leads to the following linear system for the c_j 's.

$$A_{k \times k} \bar{c}_{k \times 1} = \bar{f} \Leftrightarrow [\text{The "Normal" Equations}],$$

where

$$A_{ij} = \sum_{n=1}^N w_n \phi_j(x_n) \phi_i(x_n), \quad \bar{c} = (c_1, \dots, c_k)^{Tr} (i, j = 1, \dots, k)$$

$$\bar{f}_j = \sum_{n=1}^N w_n f_n \phi_j(x_n).$$

If the "standard" choice of basis for polynomials is chosen $\phi_1 = 1, \phi_2(x) = x, \phi_3(x) = x^2, \dots, \phi_k(x) = x^{k-1}$, the matrix \mathcal{A} is terribly ill-conditioned. This is easy to understand - a quick look at some examples shows that with this choice of basis the matrix \mathcal{A} has elements of vastly different orders of magnitudes. This leads to a lot of roundoff error under Gaussian elimination.

Thus, for an efficient algorithm, we must pick our basis functions carefully. Notice that if we define the dot product:

$$\langle \bar{u}; \bar{v} \rangle \stackrel{\text{def}}{=} \begin{pmatrix} u_1 \\ \vdots \\ u_N \end{pmatrix} \cdot \begin{pmatrix} v_1 \\ \vdots \\ v_N \end{pmatrix} \equiv u_1 v_1 w_1 + u_2 v_2 w_2 + \cdots + u_N v_N w_N$$

Exercise

5.2.3. Consider the data

X_j	-1	0	1	$3/2$	2	3	4
Y_j	8	3	0	$-3/4$	-1	0	3

- (a) Write down the Lagrange interpolant.
(You need not ~~write~~ simplify the resulting algebraic expressions).
- (b) Compute the difference table of this data and find the interpolant ~~in~~ using the Newton form.
- (c) What information comes readily from the latter (Newton form) and not from the former (Lagrange form).

5.2.4. Consider the data :

X_j	0	1	2	3	4
Y_j	1100	1080	1040	960	840

- (a) Compute the difference table of this data and the cubic interpolant.
- (b) Can you argue that the cubic term has little influence and a quadratic approximation might be more appropriate?

then

$$A_{ji} = \langle \phi_j; \phi_i \rangle.$$

Thus, if we choose a basis for \mathcal{P}_{k+1} that is orthonormal in $\langle \cdot; \cdot \rangle$ then A becomes I !

Beginning with the basis $\{1, \dots, x^{k-1}\}$ for \mathcal{P}_{k+1} we can use, e.g., Gram-Schmidt to construct an orthogonal basis.

Example: A heavy object is dropped from 1100 feet. It's height after x seconds (ignoring air resistance) is:

$$h(x) = 1100 - 16x^2.$$

If we actually do the experiment we observe the following:

Time in Seconds	Height (feet) Observed	Formula Prediction $h(x)$	Least Squares Prediction Height
0	1100	1100	1097.7
1	1080	1084	1085.1
2	1040	1036	1038.3
3	960	956	957.1
4	840	844	841.7

Fit this with $\phi_0(x) = 1, \phi_1(x) = x, \phi_2(x) = x^2$ i.e.

$$p_2(x) = c_0 \cdot 1 + c_1 \cdot x + c_2 \cdot x^2, k = 3, N = 5 \text{ points.}$$

This gives a 3×3 linear system $A\bar{c} = \bar{b}$. For example:

$$A_{01} = \sum_{i=0}^4 \phi_1(x_i)\phi_0(x_i) = 0 + 1 + 2 + 3 + 4 = 10$$

.....

$$A_{22} = \sum_{i=0}^4 \phi_2(x_i)\phi_2(x_i) = \sum x_i^2 x_i^2 = 0^4 + 1^4 + 2^4 + 3^4 + 4^4 = 354.$$

Giving the 3×3 linear system:

$$\begin{bmatrix} 5 & 10 & 30 \\ 10 & 30 & 100 \\ 30 & 100 & 354 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 5020 \\ 9400 \\ 27340 \end{bmatrix} \Leftrightarrow \begin{cases} 30c_2 + 10c_1 + 5c_0 = 5020 & c_2 = -17.14285714 \\ 100c_2 + 30c_1 + 10c_0 = 9400 & c_1 = 4.573428573 \\ 354c_2 + 100c_1 + 30c_0 = 27340 & c_0 = 1097.714286 \end{cases}$$

and

$$p_2(x) = -17.14285714x^2 + 4.573428573x + 1097.714286.$$

This $p_2(x)$ is a very good approximation *OVER THE RANGE OF THE ORIGINAL DATA!* We can check that $p_2(x)$ is the correct form by doing a cubic fit ($p_3(x)$) and checking that the cubic coefficient is "small".

5.4. Orthogonal Polynomials and Least Squares.

The problem of computing a least squares approximation reduces to finding a basis $\{\tilde{p}_0, \tilde{p}_1, \dots, \tilde{p}_n\}$ for the polynomials of degree $\leq n$ in which the normal equations matrix A is diagonal. We shall see that this is done by a simple three term iteration (recursion). First we will need some notation.

Keep in mind that we are especially interested in finding this basis for $n =$ the polynomial degree "small" and for $N =$ the number of data points "large", with

$$\langle p, q \rangle = \sum_{j=1}^N p(x_j) q(x_j) w_j \quad (\text{each } w_j > 0).$$

Definition. \mathcal{P}_n is the set of all polynomials of degree $\leq n$. (This is an $(n+1)$ dimensional vector space.)

is a map $\langle \cdot, \cdot \rangle: \mathbb{P}_n \times \mathbb{P}_n \rightarrow \mathbb{R}$. It

Preliminaries

An inner product $\langle \cdot, \cdot \rangle$ is a generalization of the idea of a dot-product in \mathbb{R}^n . It must satisfy three conditions to be usable in least squares problems. These are: for all $p(x), q(x), r(x) \in \mathbb{P}_n$

1) "definiteness"

$$\langle p, p \rangle \geq 0 \quad \text{and} \quad \langle p, p \rangle = 0 \\ \text{if and only if} \quad p = 0.$$

2) "Symmetry"

$$\langle p, q \rangle = \langle q, p \rangle$$

3) "bi-linearity" for all real numbers α and β

$$\langle \alpha p(x) + \beta q(x), r(x) \rangle = \alpha \langle p, r \rangle + \beta \langle q, r \rangle$$

and

$$\langle p, \alpha q + \beta r \rangle = \alpha \langle p, q \rangle + \beta \langle p, r \rangle.$$

This last condition just means that $\langle p, q \rangle$ is linear in p for q frozen and linear in q for p fixed.

There are many interesting examples of inner-products on \mathbb{P}_n .

Examples of inner products on $\mathcal{P}_n[a, b]$:

1. If $[a, b] = [-1, 1]$, then

$$\langle p, q \rangle := \int_{-1}^{+1} p(x) q(x) dx$$

is an inner product on $\mathcal{P}_n[-1, 1]$.

2. A weight function can be used in the above integral as well. If $w(x) \geq 0$ and $w(x) > 0$ except for a finite number of points where w vanishes, then

$$\langle p, q \rangle := \int_{-1}^{+1} p(x) q(x) w(x) dx$$

is also an inner product on $\mathcal{P}_n[-1, 1]$.

3. Given positive weights w_j and distinct points $x_0 < x_1 < \dots < x_m$, then define:
on $[a, b]$:

$$\langle p, q \rangle := \sum_{j=0}^m p(x_j) q(x_j) w_j.$$

Proposition. This $\langle \cdot, \cdot \rangle$ is an inner product on $\mathcal{P}_n[a, b]$ provided enough points are used:
for $M \geq n$.

The problem is now as follows. We are given data points and these define our inner product $\langle \cdot, \cdot \rangle$.

We seek a basis for \mathcal{P}_n , $\{p_0, \dots, p_n\}$ which is *orthogonal* with respect to $\langle \cdot, \cdot \rangle$, specifically,

$$\langle p_i, p_j \rangle = 0, \text{ if } i \neq j$$

If additionally $\langle p_i, p_i \rangle = 1$ then the basis is said to be *orthonormal*.

Given such a basis the computation of the least squares approximation is very easy.

Expanding

$$q_n^*(x) = a_0 p_0(x) + a_1 p_1(x) + \dots + a_n p_n(x)$$

we obtain the equations for the undetermined coefficients, as follows.

Setting $p = p_i$ gives:

$$\langle a_0 p_0(x) + \dots + a_n p_n(x), p_i(x) \rangle = \langle f(x), p_i(x) \rangle,$$

whereupon orthogonality of the basis yields:

$$\langle p_i, p_i \rangle a_i = \langle f, p_i \rangle \text{ or } a_i = \frac{\langle f, p_i \rangle}{\langle p_i, p_i \rangle}.$$

Thus, the normal equations become a diagonal system when an orthogonal basis is used.

Problem: Given $w(x)$ how do we construct an orthogonal basis for \mathcal{P}_n ?

The solution is by using the Gram-Schmidt procedure, suitably interpreted. Define:

$$\tilde{p}_0(x) = 1 \quad \tilde{p}_1(x) = x - \frac{\langle 1, x \rangle}{\langle 1, 1 \rangle}.$$

Note that $\langle \tilde{p}_0, \tilde{p}_1 \rangle = 0$. \tilde{p}_2 is defined by:

$$\tilde{p}_2(x) = (x - \alpha_2)\tilde{p}_1(x) - \beta_2 \tilde{p}_0(x)$$

where α_2, β_2 are chosen to make \tilde{p}_2 orthogonal to \tilde{p}_0, \tilde{p}_1 . This leads to the following equations

$$\begin{aligned} \langle \tilde{p}_2, \tilde{p}_1 \rangle &= \langle x\tilde{p}_1, \tilde{p}_1 \rangle - \alpha_2 \langle \tilde{p}_1, \tilde{p}_1 \rangle \\ &\quad - \beta_2 \langle \tilde{p}_0, \tilde{p}_1 \rangle, \quad i = 0, 1. \end{aligned}$$

When $i = 1$ one term drops out, leaving

$$\langle \tilde{p}_2, \tilde{p}_0 \rangle = \langle x\tilde{p}_1, \tilde{p}_0 \rangle - \alpha_2 \langle \tilde{p}_1, \tilde{p}_0 \rangle$$

which vanishes provided:

$$\alpha_2 = \frac{\langle x\tilde{p}_1, \tilde{p}_0 \rangle}{\langle \tilde{p}_1, \tilde{p}_1 \rangle}$$

Choose also

$$\beta_2 = \frac{\langle x\tilde{p}_1, \tilde{p}_0 \rangle}{\langle \tilde{p}_0, \tilde{p}_0 \rangle}$$

when $i = 0$ this choice of β_2 ensures that $\langle \tilde{p}_2, \tilde{p}_0 \rangle = 0$ and $\{\tilde{p}_0, \tilde{p}_1, \tilde{p}_2\}$ are an orthogonal basis for \mathcal{P}_2 .

In the general case (\mathcal{P}_k), suppose $\{\tilde{p}_0, \dots, \tilde{p}_k\}$ is an orthogonal basis for \mathcal{P}_k . We now construct \tilde{p}_{k+1} such that $\{\tilde{p}_0, \dots, \tilde{p}_k, \tilde{p}_{k+1}\}$ is an orthogonal basis for \mathcal{P}_{k+1} . Following the previous example of \mathcal{P}_2 , set

$$(5.3.1) \quad \tilde{p}_{k+1}(x) = (x - \alpha_k)\tilde{p}_k(x) - \beta_k \tilde{p}_{k-1}(x)$$

where

$$\alpha_k = \frac{\langle x\tilde{p}_k, \tilde{p}_k \rangle}{\langle \tilde{p}_k, \tilde{p}_k \rangle}, \quad \beta_k = \frac{\langle x\tilde{p}_k, \tilde{p}_{k-1} \rangle}{\langle \tilde{p}_{k-1}, \tilde{p}_{k-1} \rangle}.$$

As before this ensures

$$\langle \tilde{p}_{k+1}, \tilde{p}_j \rangle = 0, \quad j = k \text{ and } j = k - 1.$$

If $0 \leq j \leq k - 2$, $x\tilde{p}_j \in \mathcal{P}_{k-1}$, whereupon it may be expanded:

$$x\tilde{p}_j = \gamma_0\tilde{p}_0 + \gamma_1\tilde{p}_1 + \cdots + \gamma_{k-1}\tilde{p}_{k-1}.$$

Thus $\langle x\tilde{p}_j, p_k \rangle = 0 = \langle x\tilde{p}_k, \tilde{p}_j \rangle$.

From this fact and (5.3.1), for $j = 0, \dots, k - 2$,

$$\begin{aligned} \langle \tilde{p}_{k+1}, \tilde{p}_j \rangle &= \langle x\tilde{p}_k, \tilde{p}_j \rangle - \alpha_k \langle \tilde{p}_k, \tilde{p}_j \rangle + \beta_k \langle \tilde{p}_{k-1}, \tilde{p}_j \rangle \\ &= 0 - 0 + 0 = 0. \end{aligned}$$

Thus, \tilde{p}_{k+1} is orthogonal to p_0, \dots, p_k and we have an orthogonal basis for \mathcal{P}_{k+1} .

Example: The Jacobi Polynomials Choose $w(x) = (1-x)^\alpha(1+x)^\beta$, $-1 < x < +1$, where $\alpha, \beta > -1$. The Jacobi polynomials are usually normalized by requiring

$$(5.3.2) \quad p_j^{(\alpha, \beta)}(1) = \frac{(\alpha+1)(\alpha+2)\cdots(\alpha+j)}{j!}$$

Special Case: $\alpha = 0, \beta = 0, w(x) \equiv 1$ the $p_j^{(0,0)} = p_j$ are then the *Legendre polynomials*.

These are normalized by $p_j(1) = 1$. Constructing $\tilde{p}_j(x)$ via the general Gram-Schmidt procedure we evaluate:

$$\tilde{p}_k(1) = 2^k / \binom{2k}{k},$$

(which can be proven by induction).

Thus,

$$p_j = \binom{2j}{j} 2^{-j} \tilde{p}_j, \quad j = 0, 1, 2, \dots,$$

and the recursion relation for p_j becomes after simplification:

$$p_{j+1}(x) = \frac{2j+1}{j+1} x p_j(x) - \frac{j}{j+1} p_{j-1}(x)$$

or

$$(j+1)p_{j+1}(x) = (2j+1)xp_j(x) - jp_{j-1}(x).$$

For example, the first five become:

$$\begin{aligned} p_0(x) &= 1, & p_1(x) &= x, & p_2(x) &= 3/2x^2 - 1/2, \\ p_3(x) &= 5/2x^3 - 3/2x, & p_4(x) &= 35/8x^4 - 15/4x^2 + 3/8. \end{aligned}$$

Special Case 2: $\alpha = \beta = -1/2$ so $w(x) = (1-x^2)^{-1/2}, p_j^{(1/2, 1/2)}(x)$ are then the Chebyshev polynomials, multiplied by a normalizing factor. Specifically,

$$T_n(x) = \cos(n\theta), \text{ where } x = \cos \theta, \quad 0 \leq \theta \leq \pi,$$

and

$$p_j^{(-1/2, -1/2)} = \frac{1.3.5 \cdots (2j-1)}{2^j j!} T_j(x).$$

Example: Equally Spaced Points on $[-1, 1]$.

Suppose the points are equally spaced:

$$x_j = -1 + \frac{2(j-1)}{N-1}, \quad j = 1, \dots, N,$$

and suppose the weights are all 1:

$$w_j = 1, \quad j = 1, \dots, N.$$

The resulting polynomials are then the *Gram polynomials*. It is easy to see that the choice of points and weights give $\alpha_\ell = 0$ for all ℓ , so, in effect, we have two uncoupled recursions.

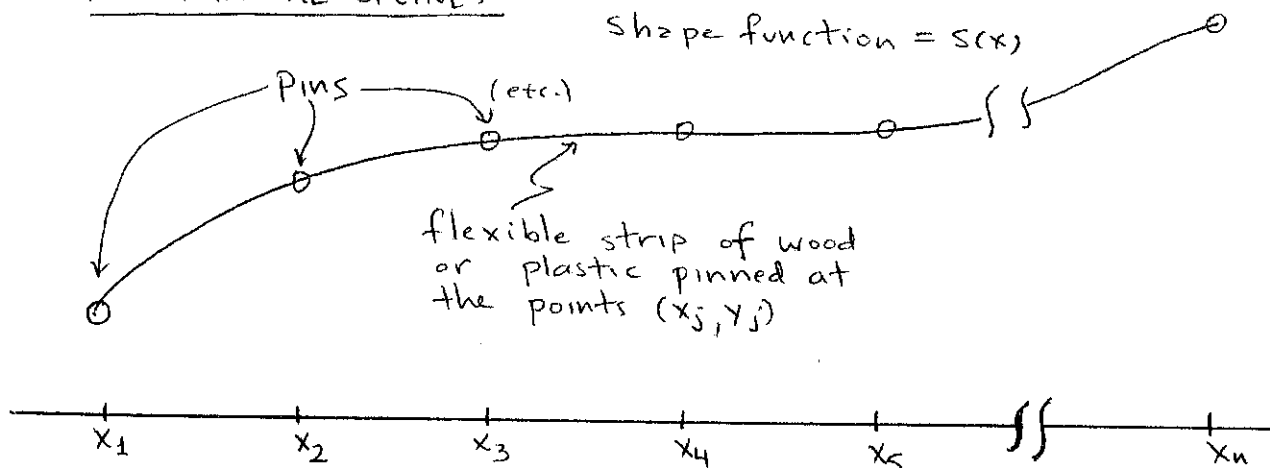
One is for even degree polynomials and one is for odd. For example, with $\tilde{g}_0 = 1, \tilde{g}_1 = x$ it follows that $\tilde{g}_2 = x^2 - (N+1)/3(N-1)$.

5.4.1.
 Exercise \times Compute the quadratic least square approximation to $f(x) = |x|$ at the points $\{-1, -1/2, 0, 1/2, +1\}$, directly (solving the 3×3 normal equations) and using the orthogonal Gram polynomials.

5.5 Cubic Splines.

So far, we are faced with (smooth) low degree interpolation with large truncation error and high degree interpolation with lots of "wiggles". The solution of the problem of finding a smooth interpolant with high accuracy is provided by *Spline Interpolation*.

MECHANICAL SPLINES



- (i) Spline must pass through knots: $\Leftrightarrow s(x_k) = f_k \quad k = 1, \dots, n.$
- (ii) $s(x)$ is smooth: it doesn't break or bend at sharp angles: $\Leftrightarrow s, s', s''$ are all continuous for $a \leq x \leq b.$
- (iii) In each interval $x_k \leq x \leq x_{k+1}, s(x)$ satisfies the thin beam equation $s'''' = 0$: $\Leftrightarrow s(x)$ is (approximately) cubic on each $x_k \leq x \leq x_{k+1}.$
- (iv) Among possible shapes, $s(x)$ minimizes potential energy: \Leftrightarrow if $g(x)$ is any other C^2 piecewise cubic passing through the knots: $\int_a^b s''(x)^2 dx \leq \int_a^b g''(x)^2 dx$ and $s''(a) = s''(b) = 0.$

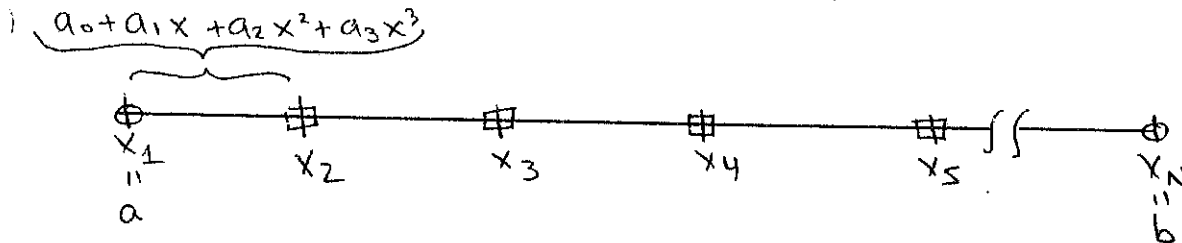
Thus, the *cubic spline interpolation* $s(x)$, to the data $(x_j, f_j), j = 1, \dots, N$, is a C^2 function that is a cubic polynomial on each interval $x_k \leq x \leq x_{k+1}$ (between the knots) and satisfies

$$s(x_k) = f_k, k = 1, \dots, N, s''(a) = 0 = s''(b).$$

Constructing the Cubic Spline.

Method 1: Crude Approach.

We set up a huge linear system for the total degrees of freedom and constraints



On each interval $s(x) = a_0 + a_1x + a_2x^2 + a_3x^3$ so there are

$$4 \text{ coefficient/interval} \times (N - 1) \text{ intervals} = 4N - 4 \text{ degrees of freedom.}$$

Constraints: At each \square there are

$$\left. \begin{array}{l} 3 \text{ continuity constraints} = 3(N - 2) \\ \text{at each } \theta : 1 = 2 \text{ constraints} \\ \text{at each knot } s(x_k) = f_k = N \text{ constraints} \end{array} \right\} \text{TOTAL Constraints} = 4N - 4.$$

The Error Estimate.

Let $\Delta x_i = x_i - x_{i-1}$.

S.S.I.
Theorem: Suppose $f \in C^4[a, b]$ then for $a \leq x \leq b$.

$$(1) \quad |f(x) - s(x)| \leq \frac{5}{384} (\max_i \Delta x_i)^4 \max_{a \leq \xi \leq b} |f^{(4)}(\xi)|,$$

$$(2.) \quad |f'(x) - s'(x)| \leq \frac{1}{24} (\max_i \Delta x_i)^3 \max_{a \leq \xi \leq b} |f^{(4)}(\xi)|,$$

If $\Delta x_i \equiv h$ (constant-uniform mesh), at the knots we have: for $f \in C^5[a, b]$:

$$|f'(x_i) - s'(x_i)| \leq \frac{1}{60} h^4 \max_{\xi} |f^{(5)}(\xi)|. \quad \square$$

Work: Solving for s''_k :

$5n - 14$ additions

$5n - 13$ multiplications

$4n - 8$ divisions

Evaluating $s(\bar{x})$ (once $x_k < \bar{x} < x_{n+1}$)

$$7A + 8M + 4D$$

Minimal Curvature Property.

Theorem 5.42. Let g satisfy (i), (ii) then

$$\int_a^b (g'')^2 dt = \int_a^b (s'')^2 dt + \int_a^b (g'' - s'')^2 dt$$

pf:

$$\int_a^b (g'' - s'')^2 = \int_a^b (g'')^2 - \int_a^b (s'')^2 - 2 \int_a^b (g'' - s'')s''$$

This last term is zero. Indeed, write:

$$\int_a^b (g - s)'' s'' = \sum_{k=1}^n \int_{I_k} (g - s)'' s'' \text{ and integrate by parts } \int_{I_k} = \int_{I_k} (g - s)s'''' + \text{boundary terms which all cancel} = 0.$$

Implementation

Since $s(x)$ is cubic on each $[x_i, x_{i+1}]$, $s''(x)$ is linear

$$(5.41) \quad s'(x) = s'(x_i) + s''(x_i)(x - x_i) + \frac{s''(x_{i+1}) - s''(x_i)}{2(x_{i+1} - x_i)}(x - x_i)^2$$
$$s''(x) = s''(x_i) + \frac{x - x_i}{x_{i+1} - x_i}(s''(x_{i+1}) - s''(x_i))$$

Integrating and using $s(x_i) = f_i$ we obtain:

$$(5.42) \quad s(x) = f_i + s'(x_i)(x - x_i) + s''(x_i)\frac{1}{2}(x - x_i)^2 + \frac{s''(x_{i+1}) - s''(x_i)}{6(x_{i+1} - x_i)}(x - x_i)^3, \quad x_i \leq x \leq x_{i+1}.$$

Thus, we need to solve for s'_i and s''_i , let $h_i = x_{i+1} - x_i$. In (5.4.2) set $x = x_{i+1}$, and solve for s'_i :

$$(5.4.3) \quad s'_i = \frac{f_{i+1} - f_i}{h_i} - s''_{i+1} \frac{h_i}{6} - s''_i \frac{h_i}{3}.$$

Thus, we need only solve for the s''_i 's as the s'_i 's are then determined from the f_i 's and s''_i 's. Inserting (5.4.1) into (5.4.3) gives the linear system:

$$h_{i+1}s''_{i-1} + 2(h_{i-1} + h_i)s''_i + h_i s''_{i+1} = \left(\frac{f_{i+1} - f_i}{h_i} - \frac{f_i - f_{i-1}}{h_{i-1}} \right)$$
$$i = 2, 3, \dots, n-1$$

and $(N-2) \times (N-2)$ linear system for s''_i .