
MATH/COSC 3416, Numerical Methods

Introductory Notes on Numerical Pathology

January 2010 (MATLAB version)

(B. G. Adams)

Contents

1	Roundoff and truncation errors	2
1.1	Absolute error	2
1.2	Relative error	2
1.3	Estimating errors	2
1.4	Truncation error	3
2	Roundoff error in numerical calculations	4
2.1	Accumulation of roundoff error	4
3	Roundoff error in subtractions	5
3.1	Loss of significance example	5
3.2	Fixing the loss of significance	6
4	Taylor series	7
4.1	Some important series	8
4.2	Approximation of a function near its center of convergence	8
4.3	Formal Taylor series	9
4.4	Formal power series	9
4.5	Taylor's theorem with error term	9
4.6	Taylor polynomials	9
4.7	Alternate form of Taylor series	10
4.8	The mean value theorem	10
4.9	General error estimates	11
4.10	Example: error estimates in calculation of e	11
4.11	Example: compute square root of 1.00001	12
4.12	Example: compute square root of 0.99999	12
4.13	Alternating series	13
4.14	Error estimates for alternating series	13
4.15	Example: calculation of $\ln 2$ using $\ln 1+x$	13
4.16	Example: calculation of $\ln 2$ using a better series	13

5	Examples of loss of significance in subtractions	14
5.1	Loss of precision in the $x - \sin x$ example	15
5.2	Loss of precision theorem	15
5.3	Fixing the $x - \sin x$ example	16
5.4	Finding real roots of a quadratic equation	18
6	Series summation and convergence rates	18
6.1	Calculating partial sums of a series	18
6.2	A much better series for $\ln 2$	20
7	Calculating derivatives using standard limit definition	22
7.1	Derivative of $\sin x$	22
8	Recurrence relations and iterative algorithms	24
8.1	A stable recurrence relation	24
8.2	An unstable recurrence relation	26
8.3	Backward recurrence	28
9	Slow and fast convergence	29
9.1	Slow convergence	30
9.2	Fast convergence	31

1 Roundoff and truncation errors

1.1 Absolute error

If x is the exact value of some quantity, a root of an equation for example, and x_{approx} is an approximation then the absolute error is

$$E_{\text{abs}} = |x - x_{\text{approx}}| \quad (1.1)$$

1.2 Relative error

The relative error is given by

$$E_{\text{rel}} = \frac{E_{\text{abs}}}{|x|} \quad (1.2)$$

1.3 Estimating errors

In practice we don't know the exact value so we need to calculate approximate values for the absolute or relative errors. One way to estimate errors is to calculate upper bounds on the error. For example

we may be able to show that $E_{\text{abs}} < c$ for some $c > 0$ and similarly for the relative error. A useful form of bound for the relative error is

$$E_{\text{rel}} \leq 0.5 \times 10^{-n} \quad (1.3)$$

This means that the approximate value is accurate to at least n significant figures.

Many numerical methods are iterative. To approximate some exact quantity x we begin with a starting approximation x_0 and iteratively compute better approximations x_1, x_2, \dots . If the process converges we can stop when the difference between two successive approximations is less than some desired error. Given a sequence of iterations we can use the following approximations

$$E_{\text{abs}} = |x_n - x_{n-1}|, \quad E_{\text{rel}} = \frac{E_{\text{abs}}}{|x_n|}$$

as a measure of convergence.

1.4 Truncation error

Consider the two functions $f_1(x) = \sin x$ and $f_2(x) = x - x^3/6$. Then f_2 is an approximation to f_1 obtained by keeping the first two terms of the Taylor series expansion of f_1 . By omitting the remaining terms in the infinite series we are making what is called a truncation error given by the absolute or relative errors

$$E_{\text{abs}} = |\sin x - x + x^3/6|, \quad E_{\text{rel}} = \frac{E_{\text{abs}}}{|\sin x|} \quad (1.4)$$

At $x = 0.05$ we obtain

$$\begin{aligned} E_{\text{abs}} &= 2.6 \times 10^{-9}, & E_{\text{rel}} &= 0.052 \times 10^{-6} \approx 0.5 \times 10^{-7} \\ f_1(x) &= 0.04997916927 \\ f_2(x) &= 0.04997916667 \end{aligned}$$

The results agree to at least 7 significant figures.

At $x = 0.25$ we obtain

$$\begin{aligned} E_{\text{abs}} &= 8.1 \times 10^{-6}, & E_{\text{rel}} &= 0.33 \times 10^{-4} \leq 0.5 \times 10^{-4} \\ f_1(x) &= 0.2474039593 \\ f_2(x) &= 0.2473958333 \end{aligned}$$

The results agree to about 4 significant figures.

This example shows that the relative error is a measure of the number of significant figures. The following MATLAB script `err.m` illustrates this.

```
for x = 0 : 0.05 : 0.25
    exact = sin(x);
    approx = x - x^3/6;
```

```

abs_err = abs(exact - approx);
if x == 0
    rel_err = 0.0;
else
    rel_err = abs_err / abs(x);
end
fprintf('%8.2f%17.9e%17.9e%17.9e%17.9e\n', ...
        x, exact, approx, abs_err, rel_err);
end

```

The output is

```

>> err
    0.00 0.000000000e+000 0.000000000e+000 0.000000000e+000 0.000000000e+000
    0.05 4.997916927e-002 4.997916667e-002 2.604011659e-009 5.208023318e-008
    0.10 9.983341665e-002 9.983333333e-002 8.331349481e-008 8.331349481e-007
    0.15 1.494381325e-001 1.494375000e-001 6.324735992e-007 4.216490661e-006
    0.20 1.986693308e-001 1.986666667e-001 2.664128395e-006 1.332064197e-005
    0.25 2.474039593e-001 2.473958333e-001 8.125921190e-006 3.250368476e-005

```

2 Roundoff error in numerical calculations

Each operation (+, -, *, /, for example) normally produces a small roundoff error because most real numbers cannot be stored exactly in computer memory. Such numbers must be truncated or rounded to a certain number of decimal or binary digits.

Most computers use binary arithmetic in hardware with a fixed number of bits such as 32 (single precision) or 64 (double precision). Numbers that can be stored exactly in these formats are called **machine numbers**.

2.1 Accumulation of roundoff error

Consider the computation of

$$s = \frac{1}{10} + \cdots + \frac{1}{10} = \sum_{k=1}^{10,000} \frac{1}{10} \quad (2.1)$$

Using a decimal machine and arithmetic the answer will be exactly 1,000 since $1/10$ is the machine number 0.1.

On a binary machine using binary arithmetic $1/10$ is not a machine number. The base 2 representation is the infinite repeating binary number

$$(0.1)_{10} = (0.\overline{00011})_2 = (0.0\ 0011\ 0011\ 0011\ \cdots)_2 \quad (2.2)$$

Therefore $1/10$ must be truncated or rounded to a machine number. To see this using MATLAB run the script [roundoff1.m](#):

```

s = 0;
max_index = 10000;
for k = 1 : max_index
    s = s + 0.1;
end
s

```

The result is 1000.000000000159 so the total roundoff error in doing the sum is 0.16×10^{-9} corresponding to a relative error of $0.16 \times 10^{-9}/10^3 = 0.16 \times 10^{-12}$ which is an error of 0.16×10^{-16} per addition.

On a binary machine using the hardware arithmetic there are usually two floating point representations.

Single precision (32 bits) \approx 7–8 decimal digits
 Double precision (64 bits) \approx 15–16 decimal digits

3 Roundoff error in subtractions

Watch out for subtractions of nearly equal quantities. The result will have less significant figures. In fact all significant figures can be lost.

3.1 Loss of significance example

Consider the function $f_1(x) = \sqrt{1+x} - 1$. When x is near 0 this is a subtraction of nearly equal values and we can lose significant figures rapidly as x approaches 0. Here are some Maple results using the default of 10 digits.

$$\begin{array}{ll}
 f_1(0.123456789 \times 10^{-1}) = 0.6153904 \times 10^{-2} & 7 \text{ sig figs} \\
 f_1(0.123456789 \times 10^{-5}) = 0.617 \times 10^{-6} & 3 \text{ sig figs} \\
 f_1(0.123456789 \times 10^{-6}) = 0.61 \times 10^{-7} & 2 \text{ sig figs} \\
 f_1(0.123456789 \times 10^{-7}) = 0.6 \times 10^{-7} & 1 \text{ sig figs} \\
 f_1(0.123456789 \times 10^{-8}) = 0 & \text{no sig figs}
 \end{array}$$

Here we start with 9 significant figures and quickly lose them. Eventually we lose all figures and get 0 as the value of the function.

Note With the default 10 digits in Maple we have

$$\begin{array}{l}
 1 + 10^{-9} = 1.000000001 \\
 1 + 10^{-10} = 1.0000000001
 \end{array}$$

The first result requires 10 digits but the second one requires 11 digits so it cannot be represented using 10 digit numbers. Instead Maple returns 1 for this addition. Thus there are non-zero numbers x such that $1+x=1$. The smallest non-zero x such that $1+x>1$ is called the **machine epsilon**. In this Maple example the machine epsilon is 10^{-9} . For a binary machine we would get a power of 2 as the machine epsilon. The following MATLAB script `machine_eps.m` computes the machine epsilon for base 2.

```
e = 1;
while 1+e > 1
    e = e / 2;
end
e = e * 2;
disp(e);
disp(eps);
```

This gives the value $2.220446049250313 \times 10^{-16}$ for the binary machine epsilon. This value agrees with the built-in MATLAB constant `eps`.

3.2 Fixing the loss of significance

In the preceding example the loss of significance is easily fixed using the following equivalent version.

$$f_2(x) = \left(\sqrt{1+x}-1\right) \left(\frac{\sqrt{1+x}+1}{\sqrt{1+x}+1}\right) = \frac{x}{\sqrt{1+x}+1} \quad (3.1)$$

Now there are no subtractions so we can use $f_2(x)$ for all values of x . Now the Maple results are

$$\begin{aligned} f_2(0.123456789 \times 10^{-1}) &= 0.6153904182 \times 10^{-2} \\ f_2(0.123456789 \times 10^{-5}) &= 0.6172837546 \times 10^{-6} \\ f_2(0.123456789 \times 10^{-6}) &= 0.6172839262 \times 10^{-7} \\ f_2(0.123456789 \times 10^{-7}) &= 0.6172839431 \times 10^{-7} \\ f_2(0.123456789 \times 10^{-8}) &= 0.6172839450 \times 10^{-9} \end{aligned}$$

Eventually we are getting $x/2$ if x is small enough so that $1+x=1$.

To illustrate this loss of significance on a binary machine it is necessary to display results in 64-bit binary form (16 hex digits) to see the loss of significance. To do this MATLAB has a function `num2hex` which converts a decimal number to its internal binary form. The following MATLAB script `roundoff2.m` illustrates this.

```
f1 = @(x) sqrt(1 + x) - 1;
num2hex(f1(0.123456789012345E-1))
num2hex(f1(0.123456789012345E-5))
num2hex(f1(0.123456789012345E-10))
num2hex(f1(0.123456789012345E-12))
num2hex(f1(0.123456789012345E-15))
```

The output is

```
ans =  
3f7934d6134a1700  
ans =  
3ea4b66d54e00000  
ans =  
3d9b260000000000  
ans =  
3d31600000000000  
ans =  
0000000000000000
```

Again we have a catastrophic loss of significant figures as x gets closer to 0.

The following MATLAB script `roundoff3.m` shows how the loss of significance is eliminated by using `f2`.

```
f2 = @(x) x / (sqrt(1 + x) + 1);  
num2hex(f2(0.123456789012345E-1))  
num2hex(f2(0.123456789012345E-5))  
num2hex(f2(0.123456789012345E-10))  
num2hex(f2(0.123456789012345E-12))  
num2hex(f2(0.123456789012345E-15))
```

The output is

```
ans =  
3f7934d6134a1727  
ans =  
3ea4b66d54de01a1  
ans =  
3d9b25ffd6368fc7  
ans =  
3d315fffe541de12  
ans =  
3c91cac067affea1
```

showing that there is no loss of significance.

4 Taylor series

Taylor series are very important in numerical methods. They are useful for

- evaluating functions using Taylor polynomials

- defining new functions
- providing error bounds

4.1 Some important series

Here are some series and their intervals of convergence that you should know.

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots = \sum_{k=0}^{\infty} \frac{x^k}{k!} \quad (-\infty, \infty) \quad (4.1)$$

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots = \sum_{k=0}^{\infty} \frac{(-1)^k x^{2k+1}}{(2k+1)!} \quad (-\infty, \infty) \quad (4.2)$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \dots = \sum_{k=0}^{\infty} \frac{(-1)^k x^{2k}}{(2k)!} \quad (-\infty, \infty) \quad (4.3)$$

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \dots = \sum_{k=1}^{\infty} \frac{(-1)^{k+1} x^k}{k} \quad (-1, 1] \quad (4.4)$$

$$\frac{1}{1-x} = 1 + x + x^2 + \dots = \sum_{k=0}^{\infty} x^k \quad (-1, 1) \quad (4.5)$$

These are often called Maclaurin series since the center of convergence is $x = 0$. Other points can be chosen. For example the series for $\ln(x)$ can be expanded about $x = 1$ and the result is

$$\ln(x) = (x-1) - \frac{1}{2}(x-1)^2 + \frac{1}{3}(x-1)^3 - \dots = \sum_{k=1}^{\infty} \frac{(-1)^{k+1} (x-1)^k}{k} \quad (0, 2] \quad (4.6)$$

with interval of convergence $(0, 2]$.

4.2 Approximation of a function near its center of convergence

The Taylor series expansion is only useful for evaluating the function if x is near the center of convergence. For example 5 terms of the series for $\ln(1+x)$ gives 5 significant figures if $x = 0.1$ but if $x = 2$ at the end of the convergence interval then 200,000 terms are required to get 5 significant figures. These results follow from the fact that the error in truncating an alternating series is less than the first term omitted.

$$\ln(1.1) \approx 0.1 - \frac{0.01}{2} + \frac{0.001}{3} - \frac{0.0001}{4} + \frac{0.00001}{5} = 0.09531033333$$

The exact value is 0.09531017980.... The actual error is 0.15353×10^{-6} and the absolute value of the last term omitted is 0.16666×10^{-6} .

4.3 Formal Taylor series about $x = x_0$

A formal Taylor series of a function f about $x = x_0$ is the infinite series

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \cdots + \frac{1}{n!}f^{(n)}(x_0)(x - x_0)^n + \cdots \quad (4.7)$$

$$= \sum_{k=0}^{\infty} \frac{1}{k!}f^{(k)}(x_0)(x - x_0)^k \quad (4.8)$$

If a Taylor series exists for a function it converges on some interval called the **interval of convergence** centered at the point x_0 . The radius of this interval is called the **radius of convergence** and may be finite or infinite.

4.4 Formal power series

Given x_0 and any sequence a_0, a_1, \dots we can define a formal power series as a function of x .

$$f(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)^2 + \cdots$$

When this series converges on some interval it defines a function f on this interval. This may be a known function or it may be a new function. This is how new functions can be obtained from power series.

4.5 Taylor's theorem with error term

Suppose $f(x)$ has continuous derivatives of all orders on the interval $[a, b]$. Then for any $x_0 \in [a, b]$ and $x \in [a, b]$

$$f(x) = \sum_{k=0}^n \frac{1}{k!}f^{(k)}(x_0)(x - x_0)^k + E_{n+1} \quad (4.9)$$

where

$$E_{n+1} = \frac{1}{(n+1)!}f^{(n+1)}(\xi)(x - x_0)^{n+1} \quad \text{and } \xi \in (x_0, x) \quad (4.10)$$

Here E_{n+1} is the remainder or error term. Of course ξ depends on x and x_0 . It is not known and the error term must be estimated. E_{n+1} looks like the next term in the series.

4.6 Taylor polynomials

If the error term E_{n+1} is omitted we obtain the n^{th} degree Taylor polynomial

$$P(x) = \sum_{k=0}^n \frac{1}{k!}f^{(k)}(x_0)(x - x_0)^k \quad (4.11)$$

as an approximation to $f(x)$.

4.7 Alternate form of Taylor series

Instead of using x_0 and x we can use x and $x+h$ so the center of convergence is x and a nearby point is $x+h$. This gives the series

$$f(x+h) = f(x) + \frac{h}{1!}f'(x) + \frac{h^2}{2!}f''(x) + \cdots + \frac{1}{n!}f^{(n)}(x)h^n + E_{n+1}(x,h) \quad (4.12)$$

where the error term is given by

$$E_{n+1}(x,h) = \frac{1}{(n+1)!}f^{(n+1)}(\xi)h^{n+1} \quad (4.13)$$

Here h can be positive or negative. If $h > 0$ then $\xi \in (x, x+h)$ and if $h < 0$ then $\xi \in (x+h, x)$.

If we use only the Taylor polynomial of degree 1 then

$$f(x+h) = f(x) + hf'(x) + O(h^2) \quad (4.14)$$

Dividing by h and taking limits gives

$$\lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h} = f'(x) \quad (4.15)$$

which is just the usual limit definition of the derivative from calculus.

The notation $O(h^n)$ represents any function $g(h)$ such that $|g(h)| \leq c|h|^n$ for some $c > 0$ and sufficiently small h . Using this “Big O” notation we can write the error term as

$$E_{n+1} = O(h^{n+1}) \quad (4.16)$$

so there exists a constant $c > 0$ such that $|E_{n+1}| \leq c|h|^{n+1}$. In practice c can be any positive constant such that

$$\frac{1}{(n+1)!} \left| f^{(n+1)}(\xi) \right| \leq c, \quad \text{for } \xi \in (x, x+h) \quad (4.17)$$

4.8 The mean value theorem

The mean value theorem is a special case of Taylor’s theorem for $n = 1$. If f is continuous on $[a, b]$ and f' exists on (a, b) , there exists $\xi \in (a, b)$ such that $f(b) = f(a) + (b-a)f'(\xi)$. This is just Taylor’s theorem with $x = a$, $h = b-a$, $n = 1$, and can be expressed in the form

$$f'(\xi) = \frac{f(b) - f(a)}{b - a} \quad (4.18)$$

Geometrically this means there is a number ξ such that the tangent line to the graph of f at ξ has the same slope as the secant line joining the endpoints $(a, f(a))$ and $(b, f(b))$.

4.9 General error estimates

Since the value of ξ in the error term is unknown we need to approximate the error. One way is to maximize the error term:

$$\begin{aligned} |E_{n+1}| &= \left| \frac{1}{(n+1)!} f^{(n+1)}(\xi) (x-x_0)^{n+1} \right| \\ &\leq \max_{\xi \in [x_0, x]} \left| f^{(n+1)}(\xi) \right| \cdot \frac{|x-x_0|^{n+1}}{(n+1)!} = \frac{M |x-x_0|^{n+1}}{(n+1)!} \end{aligned} \quad (4.19)$$

This technique for estimating the error bound is useful if derivatives can easily be calculated and maximized.

4.10 Example: error estimates in calculation of e

Let us use the series for e^x to evaluate e with a given error bound. Start with

$$e = 1 + 1 + \frac{1}{2!} + \cdots + \frac{1}{n!} + E_{n+1}, \quad \text{where } E_{n+1} = \frac{e^\xi}{(n+1)!} \quad (4.20)$$

Here $0 \leq \xi \leq 1$, $x = 1$, $x_0 = 0$. Then $e^\xi \leq e^1$ so $E_{n+1} \leq e/(n+1)!$. It can be shown that $e < 3$. Therefore $E_{n+1} \leq 3/(n+1)!$. Suppose we want an error less than 10^{-6} . Then we require that $3/(n+1)! < 10^{-6}$ or $(n+1)! > 3 \times 10^6$. The first value of n to satisfy this inequality is $n = 9$ so we obtain the approximation

$$e \approx 1 + 1 + \frac{1}{2!} + \cdots + \frac{1}{9!} = 2.718281526 \quad (4.21)$$

The exact value is $2.718281828 \dots$ and the actual error is 0.3×10^{-6} . These results can be obtained with the MATLAB statements

```
>> e = 0;
>> for k = 0:9
    e = e + 1/factorial(k);
end
>> e
e =
    2.71828152557319
>> err = abs(e - exp(1.0))
err =
    3.028858532871936e-007
```

Using vectorization e can be calculated in one statement using

```
>> e = sum(1.0 ./ factorial(0:9))
e =
    2.71828152557319
```

Here the argument of the `sum` function is the vector $[1/0!, 1/1!, 1/2!, \dots, 1/9!]$ and `sum` produces the sum of the elements of this vector.

4.11 Example: compute $\sqrt{1.00001}$

Let $f(x) = \sqrt{1+x}$. Then $f'(x) = \frac{1}{2}(1+x)^{-1/2}$, $f''(x) = -\frac{1}{4}(1+x)^{-3/2}$, $f'''(x) = \frac{3}{8}(1+x)^{-5/2}$. Therefore

$$\begin{aligned} f(x) &= f(0) + f'(0)x + \frac{1}{2!}f''(0)x^2 + \frac{1}{3!}f'''(\xi)x^3, \quad \text{where } 0 < \xi < x \\ &= 1 + \frac{1}{2}x - \frac{1}{8}x^2 + \frac{1}{3!} \cdot \frac{3}{8} \cdot \frac{x^3}{(1+\xi)^{5/2}} \end{aligned} \quad (4.22)$$

Since $0 < \xi < x$ we get the error estimate

$$E_3 = \frac{1}{3!} \cdot \frac{3}{8} \cdot \frac{x^3}{(1+\xi)^{5/2}} < \frac{1}{16}x^3 \quad (4.23)$$

For $x = h = 10^{-5}$ this gives a maximum error bound $E_3 < \frac{1}{16}10^{-15}$. Therefore

$$\sqrt{1.00001} = 1 + \frac{1}{2}10^{-5} - \frac{1}{8}10^{-10} = 1.000004999987500 \quad (4.24)$$

This agrees with the exact value to the same number of digits as shown by the following MATLAB statements

```
>> exact = sqrt(1.00001)
exact =
    1.00000499998750
>> approx = 1.0 + 1E-5/2.0 - 1E-10/8
approx =
    1.00000499998750
>> err = abs(exact - approx)
err =
    0
```

4.12 Example: compute $\sqrt{0.99999}$

This is similar to the previous example but using $f(x) = \sqrt{1-x}$. The same error bound $|x|^3/16$ is valid and we get

$$\sqrt{0.99999} = 1 - \frac{1}{2}10^{-5} - \frac{1}{8}10^{-10} = 0.999994999987500 \quad (4.25)$$

4.13 Alternating series

Suppose the sequence a_1, a_2, a_3, \dots is decreasing so that $a_1 \geq a_2 \geq a_3 \dots \geq a_n \geq \dots$ for all $n \geq 1$. Also suppose that $\lim_{n \rightarrow \infty} a_n = 0$. Then the alternating series $\sum_{k=1}^{\infty} (-1)^{k+1} a_k$ converges. This means that $S = \lim_{n \rightarrow \infty} S_n$ is the sum of the series where the n^{th} partial sum is $S_n = \sum_{k=1}^n (-1)^{k+1} a_k$

4.14 Error estimates for alternating series

An important property of alternating series that does not hold for other kinds of series is that

$$|S - S_n| \leq a_{n+1} \quad (4.26)$$

In other words the absolute error in using the n^{th} partial sum S_n as an approximation to the exact sum S is less or equal to the absolute value of the first term omitted.

4.15 Example: calculation of $\ln 2$ using $\ln(1+x)$

Since the series $\ln(1+x)$ has convergence interval $(-1, 1]$ we expect slow convergence at $x = 1$ since this is as far from the center of convergence $x = 0$ as we can get:

$$\begin{aligned} \ln(1+x) &= x - \frac{x^2}{2} + \frac{x^3}{3} - \dots \\ \ln(2) &= 1 - \frac{1}{2} + \frac{1}{3} - \dots + \frac{(-1)^{n+1}}{n} + \dots = S_n + \dots \end{aligned} \quad (4.27)$$

Since this is an alternating series then $|\ln(2) - S_n| \leq \frac{1}{n+1}$. If we want an absolute error less than 0.5×10^{-5} then n must satisfy $\frac{1}{n+1} < 0.5 \times 10^{-5}$ which gives $n = 200,000$. Thus the series at $x = 1$ converges too slowly to be useful.

4.16 Example: calculation of $\ln 2$ using a better series

Consider the two series

$$\begin{aligned} \ln(1+x) &= x - \frac{x^2}{2} + \frac{x^3}{3} - \dots \\ \ln(1-x) &= -x - \frac{x^2}{2} - \frac{x^3}{3} + \dots \end{aligned} \quad (4.28)$$

Subtract to obtain the series

$$\ln\left(\frac{1+x}{1-x}\right) = \sum_{k=0}^{\infty} \frac{2}{2k+1} x^{2k+1} = \sum_{k=0}^{n-1} \frac{2}{2k+1} x^{2k+1} + E_n \quad (4.29)$$

Since $\frac{2}{2k+1} \leq \frac{2}{2n+1}$ for $k \geq n$ we have the error estimate

$$\begin{aligned} E_n &= \sum_{k=n}^{\infty} \frac{2}{2k+1} x^{2k+1} \leq \sum_{k=n}^{\infty} \frac{2}{2n+1} x^{2k+1} = \frac{2}{2n+1} [x^{2n+1} + x^{2n+3} + \dots] \\ &= \frac{2}{2n+1} x^{2n+1} [1 + x^2 + x^4 + \dots] = \frac{2}{2n+1} x^{2n+1} \left(\frac{1}{1-x^2} \right) \end{aligned} \quad (4.30)$$

Now substitute $x = 1/3$ corresponding to $\frac{1+x}{1-x} = 2$ to obtain the error estimate

$$E_n \leq \frac{2}{2n+1} \left(\frac{1}{3} \right)^{2n+1} \cdot \frac{9}{8} = \frac{3}{4} \cdot \frac{1}{2n+1} \cdot \frac{1}{9^n} < 0.5 \times 10^{-5} \quad (4.31)$$

The smallest value of n satisfying this inequality is $n = 6$. Therefore we obtain the following estimate for $\ln(2)$.

$$\begin{aligned} \ln(2) &\approx \sum_{k=0}^5 \frac{2}{2k+1} \left(\frac{1}{3} \right)^{2k+1} = \frac{2}{3} + \frac{2}{3} \left(\frac{1}{3} \right)^3 + \frac{2}{5} \left(\frac{1}{3} \right)^5 + \frac{2}{7} \left(\frac{1}{3} \right)^7 + \frac{2}{9} \left(\frac{1}{3} \right)^9 + \frac{2}{11} \left(\frac{1}{3} \right)^{11} \\ &= 0.6931470738 \end{aligned} \quad (4.32)$$

The exact value is 0.6931471806. The error bound obtained is 0.1068×10^{-6} . The following MATLAB statements show this.

```
>> f = @(k) (2./(2*k+1)).*(1/3).^ (2*k+1);
>> exact = log(2)
exact =
    0.69314718055995
>> approx = sum(f(0:5))
approx =
    0.69314707375979
>> err = abs(exact - approx)
err =
    1.068001601600699e-007
```

Note that the natural logarithm (base e) is `log` and not `ln` in MATLAB.

5 Examples of loss of significance in subtractions

We have already considered the function $\sqrt{1+x} - 1$ and it was easy to fix the loss of significance near $x = 0$ by rewriting the function as $x/(\sqrt{1+x} + 1)$.

5.1 Loss of precision in the $x - \sin x$ example

The function $f(x) = x - \sin x$ is a subtraction of nearly equal quantities near $x = 0$ since $\sin x = x - x^3/6 + \dots \approx x$ near $x = 0$. Try the following MATLAB script `sin1.m` to see what happens:

```
f = @(x) x - sin(x);
for x = [0.0001234567890123456, 0.001, 0.1, 1, 1.9, 2.0]
    fprintf('%25.15e %25.15e %s\n', x, f(x), num2hex(f(x)));
end
```

We need to look at the values in binary (hex) form to see the loss of significance. The output is

```
>> sin1
1.234567890123456e-004    3.136127154409335e-013    3d56118b60000000
1.000000000000000e-003    1.666666583390042e-010    3de6e80fcd000000
1.000000000000000e-001    1.665833531718508e-004    3f25d59c1b9cdc00
1.000000000000000e+000    1.585290151921035e-001    3fc44aad3dbcc48
1.900000000000000e+000    9.536999123125854e-001    3fee84b5adb29475
2.000000000000000e+000    1.090702573174318e+000    3ff173848a9725dd
```

Note that the first three values of x are close enough to zero that they suffer a loss of significant digits: the hex representations show that there are zeros entering from the right. For example the first value loses at least $7 \times 4 = 28$ bits of precision and the second value loses at least 6 bits of precision.

On the other hand the last three values of x are far enough away from 0 that no significant figures are lost. Later we show that the formula $x - \sin x$ is good if $x \geq 1.9$.

5.2 Loss of precision theorem

To understand this theorem we need the concept of a normalized machine number. Let x, y be normalized binary machine numbers. For example in base 2 a normalized machine number has the form $0.b_1b_2b_3 \dots \times b_n$ with $b_1 > 0$. Another way to say this is that x is a normalized binary machine number if it has the form $x = r \cdot 2^n$ where $1/2 \leq r < 1$.

Thus the number 0.101×2^4 is normalized but the number 0.000101×2^{-4} is not normalized. To normalize it is necessary to shift the decimal point to obtain 0.101×2^{-7} .

We can now state the base b version of the theorem

Loss of significance theorem Let x and y be normalized machine numbers such that $0 < y < x$. If for some $p > 0$ and $q > 0$ we have

$$\frac{1}{b^p} \leq 1 - \frac{y}{x} \leq \frac{1}{b^q} \quad (5.1)$$

then at most p and at least q significant base b digits are lost in the subtraction $x - y$.

In the important special case of base 2 we have

$$\frac{1}{2^p} \leq 1 - \frac{y}{x} \leq \frac{1}{2^q} \quad (5.2)$$

Note that the quantity $1 - y/x = (x - y)/x$ is like the relative error of y with respect to x .

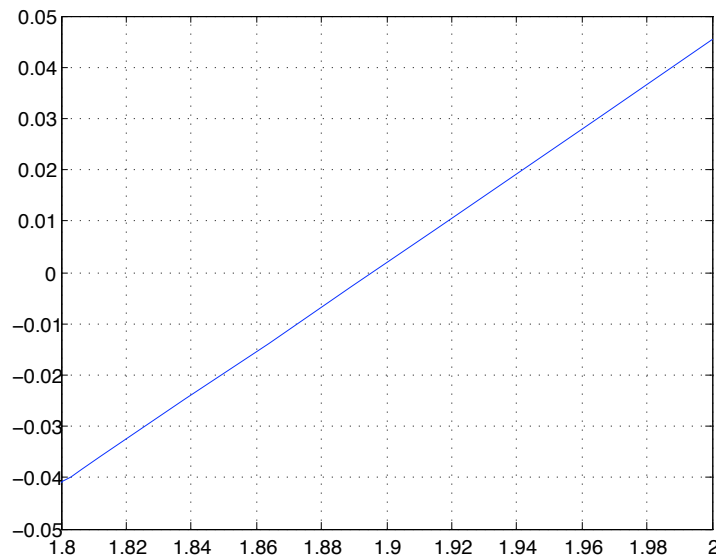
5.3 Fixing the $x - \sin x$ example

We can use this result in the $x - \sin x$ example. We can assume that $x \geq 0$ since $x - \sin x$ is an odd function. When s is too close to 0 we can use the first few terms of the Taylor series as an approximation. Thus we can define

$$G(x) = \begin{cases} F(x) = x - \sin x, & x > a \\ T(x) = \frac{x^3}{3!} - \frac{x^5}{5!} + \dots, & 0 \leq x \leq a \end{cases} \quad (5.3)$$

We need to determine a and the number of terms to use in the series for $T(x)$. The value of a can be determined using the loss of significance theorem. Let us assume base 2 and choose a value of a so that at most one significant bit is lost. Then we need to solve the inequality $1 - \sin x/x \geq 1/2$. We can do this in MATLAB by graphing the function $y = 1 - \sin x/x - 1/2$ and finding the smallest $y \geq 0$. A suitable value is $a = 1.9$ which can be obtained using the MATLAB script `sin2.m`:

```
y = @(x) 1 - sin(x)/x - 1/2;
fplot(y, [1.8, 2.0])
grid on;
```



Now consider the number of terms to use in the series

$$T(x) = \frac{x^3}{3!} - \frac{x^5}{5!} + \frac{x^7}{7!} - \dots + \frac{(-1)^{n+1} x^{2n+1}}{(2n+1)!} \quad (5.4)$$

Since this is an alternating series we can evaluate enough terms so that

$$\frac{x^{2n+1}}{(2n+1)!} < 10^{-16} \quad (5.5)$$

at $x = a$. This gives $n = 11$ for the first term omitted so we can choose $n = 10$ and use x^{21} as the last term. This gives the function

$$G(x) = \begin{cases} F(x) = x - \sin x, & x > 1.9 \\ T(x) = \frac{x^3}{3!} - \frac{x^5}{5!} + \frac{x^7}{7!} - \frac{x^9}{9!} + \cdots - \frac{x^{21}}{21!} & 0 \leq x \leq 1.9 \end{cases} \quad (5.6)$$

We can evaluate the series as follows.

$$T(x) = \sum_{k=1}^{10} (-1)^{k+1} \frac{x^{2k+1}}{(2k+1)!} = \sum_{k=1}^{10} t_k \quad (5.7)$$

Now the term t_{k+1} can be expressed in terms of t_k as follows

$$\frac{t_{k+1}}{t_k} = \frac{(-1)^{k+2} x^{2k+3}}{(2k+3)!} \cdot \frac{(2k+1)!}{(-1)^{k+1} x^{2k+1}} = -x^2 \cdot \frac{1}{(2k+2)(2k+3)} \quad (5.8)$$

Here is a pseudo-code algorithm for the function $G(x)$.

```

x ← ?
n ← 10
IF |x| ≥ 1.9 THEN
    s ← x - sin x
ELSE
    t ← x3/6
    s ← t
    FOR k ← 1 TO n - 1 DO
        t ← -tx2/[(2k+2)(2k+3)]
        s ← s + t
    END FOR
END IF
RETURN s

```

The MATLAB script `sinf.m` that defines the function $G(x)$ is given by

```

function [s] = sinf(x)
    n = 10;
    if abs(x) >= 1.9
        s = x - sin(x);
    else
        t = x*x*x/6;
        s = t;
        for k = 1 : (n-1)
            t = -t*x*x / ((2*k+2) * (2*k+3));
            s = s + t;
        end
    end
end

```

5.4 Finding real roots of a quadratic equation

The roots of a quadratic equation $ax^2 + bx + c = 0$ are

$$x_1 = \frac{1}{2a} \left(-b + \sqrt{b^2 - 4ac} \right) \quad (5.9)$$

$$x_2 = \frac{1}{2a} \left(-b - \sqrt{b^2 - 4ac} \right) \quad (5.10)$$

If $b > 0$ then the calculation of x_1 could involve a subtraction of nearly equal results. Similarly, if $b < 0$ then the calculation of x_2 could involve a subtraction of nearly equal results.

In either case we calculate the root whose formula does not have a subtraction and find the other root using the fact that the product of the roots of a quadratic equation are given by

$$x_1 x_2 = \frac{c}{a} \quad (5.11)$$

6 Series summation and convergence rates

Here we discuss how to evaluate the partial sums of a series using two examples for $\ln(2)$.

6.1 Calculating partial sums of a series

Let the partial sums of a series be given by

$$S_n = \sum_{k=1}^n t_k$$

The t_k are the terms of the series and we have started with index 1 although any other index can be used (0 for example). First try to express each term in terms of the previous one:

$$t_1 = a(x), \quad t_k = b(k, x)t_{k-1} \quad (6.1)$$

where a is some function of x and b is some function of k and x .

Example: For the $\ln(1+x)$ series

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \dots = \sum_{k=1}^{\infty} \frac{(-1)^{k+1} x^k}{k}$$

we have

$$t_1 = x, \quad \frac{t_k}{t_{k-1}} = - \left(\frac{k-1}{k} \right) x. \quad (6.2)$$

Here is a pseudo-code algorithm for computing partial sums:

$n \leftarrow ?$	<i>input number of terms in partial sum</i>
$t \leftarrow a(x)$	<i>initialize the term</i>
$s \leftarrow t$	<i>initialize partial sum to first term</i>
FOR $k \leftarrow 2$ TO n DO	
$t \leftarrow b(k,x)t$	<i>compute next term from previous one</i>
$s \leftarrow s+t$	<i>add it to partial sum</i>
PRINT k,s,t	<i>to see all partial sums</i>
END FOR	

Here is the algorithm for $\ln(1+x)$ at $x = 1$.

$n \leftarrow ?$	<i>input number of terms in partial sum</i>
$t \leftarrow 1.0$	<i>initialize the term</i>
$s \leftarrow t$	<i>initialize partial sum to first term</i>
FOR $k \leftarrow 2$ TO n DO	
$t \leftarrow -\left(\frac{k-1}{k}\right)t$	<i>compute next term from previous one</i>
$s \leftarrow s+t$	<i>add it to partial sum</i>
PRINT k,s,t	<i>to see all partial sums</i>
END FOR	

Here is a MATLAB script `logsum1.m` for computing and displaying partial sums

```
n = input('Enter number of terms ');

% The loop calculates each term from the preceding term
% and adds it onto the partial sum.

t = 1.0; % first term
s = t; % first partial sum
fprintf('%5d %20.10e %20.10e\n',1,t,s);
for k = 2 : n
    t = -((k-1)/k)*t;
    s = s + t;
    fprintf('%5d %20.10e %20.10e\n',k,t,s);
end
```

Here is some output with some lines omitted.

```
>> logsum1
Enter number of terms 10000
    1    1.0000000000e+000    1.0000000000e+000
```

```

2    -5.0000000000e-001    5.0000000000e-001
3     3.3333333333e-001    8.3333333333e-001
4    -2.5000000000e-001    5.8333333333e-001
5     2.0000000000e-001    7.8333333333e-001
...
997   1.0030090271e-003    6.9364843357e-001
998  -1.0020040080e-003    6.9264642956e-001
999   1.0010010010e-003    6.9364743056e-001
1000  -1.0000000000e-003    6.9264743056e-001
...
9996  -1.0004001601e-004    6.9309716305e-001
9997   1.0003000900e-004    6.9319719306e-001
9998  -1.0002000400e-004    6.9309717306e-001
9999   1.0001000100e-004    6.9319718306e-001
10000 -1.0000000000e-004    6.9309718306e-001
>>

```

The convergence is slow since we need 200,000 terms for an error 0.5×10^{-5} corresponding to 5 significant figures. This follows since the series is alternating so $|\ln(2) - S_n| \leq \frac{1}{n+1}$.

6.2 A much better series for $\ln 2$

A much better series for calculating $\ln(2)$ is obtained by subtracting the series for $\ln(1-x)$ from the series for $\ln(1+x)$ to obtain (see page 13)

$$\ln(1+x) - \ln(1-x) = \ln\left(\frac{1+x}{1-x}\right) = 2\left(x + \frac{x^3}{3} + \frac{x^5}{5} + \dots\right) = 2\sum_{k=0}^{\infty} \frac{1}{2k+1} x^{2k+1} \quad (6.3)$$

To calculate $\ln(2)$ solve $(1+x)/(1-x) = 2$ to get $x = 1/3$, which is much closer than $x = 1$ to the center of convergence at $x = 0$. The new series is

$$\ln(2) = 2\sum_{k=0}^{\infty} \frac{1}{2k+1} \left(\frac{1}{3}\right)^{2k+1} = \sum_{k=0}^{\infty} t_k \quad (6.4)$$

where the terms are

$$t_0 = \frac{2}{3}, \quad t_k = \frac{1}{9} \cdot \frac{2k-1}{2k+1} t_{k-1}, \quad \text{for } k \geq 1 \quad (6.5)$$

Pseudo-code algorithm The following pseudo-code algorithm displays the terms and partial sums.

```

n ← ?
exact ← ln(2)
c ← 1.0/9.0
t ← 2.0/3.0           this is t0
s ← t               this is S0
err ← |s - exact|
PRINT 0,s,t
FOR k ← 1 TO n - 1 DO
    
$$t \leftarrow c \cdot \frac{2k-1}{2k+1} \cdot t$$

    s ← s + t
    err ← |s - exact|
    PRINT k,s,t,err
END FOR

```

Here is a MATLAB script `logsum2.m` for computing and displaying partial sums:

```

n = input('Enter number of terms ');

c = 1.0 / 9.0;
t = 2.0 / 3.0;
s = t;
fprintf('%5d %20.10e %20.10e\n', 0, t, s);
for k = 1 : n-1
    t = c * ((2*k-1)/(2*k+1)) * t;
    s = s + t;
    fprintf('%5d %20.10e %20.10e\n', k, t, s);
end
fprintf('Exact value is %.10e\n', log(2));

```

Some output is

```

>> logsum2
Enter number of terms 12
    0    6.6666666667e-001    6.6666666667e-001
    1    2.4691358025e-002    6.9135802469e-001
    2    1.6460905350e-003    6.9300411523e-001
    3    1.3064210595e-004    6.9313475733e-001
    4    1.1290058539e-005    6.9314604739e-001
    5    1.0263689581e-006    6.9314707376e-001
    6    9.6496226829e-008    6.9314717026e-001
    7    9.2922292502e-009    6.9314717955e-001
    8    9.1100286766e-010    6.9314718046e-001

```

```

9      9.0567536551e-011    6.9314718055e-001
10     9.1046729866e-012    6.9314718056e-001
11     9.2366247691e-013    6.9314718056e-001
Exact value is 6.9314718056e-001
>>

```

Here S_5 gives 0.6931470738 compared to the exact value 0.6931471806 so we get 5 significant figures.

Conclusion Some series are useful in practice, others are not. From the mathematical point of view both series for $\ln(2)$ converge. From the numerical analysis point of view only the second series is useful. Thus, the rate of convergence is important.

7 Calculating derivatives using standard limit definition

We can try to calculate $f'(x)$ by simulating the limit definition as follows

$$f'(x) = \lim_{n \rightarrow \infty} \frac{f(x+h_n) - f(x)}{h_n}, \quad \text{where } h_n = \frac{1}{4^n}, n \geq 0. \quad (7.1)$$

7.1 Derivative of $\sin x$

For $f(x) = \sin x$ at $x = 0.5$ we have

$$f'(x) = \lim_{n \rightarrow \infty} \frac{\sin(0.5 + h_n) - \sin(0.5)}{h_n} \quad (7.2)$$

The exact answer is $\cos(0.5) \approx 0.877582562$.

Pseudo-code algorithm Here is a pseudo-code algorithm for simulating the limit.

```

exact ← cos(0.5)
h ← 1
FOR n ← 1 TO 20 DO
    r ←  $\frac{\sin(0.5 + h) - \sin(0.5)}{h}$ 
    err ← exact - r
    PRINT n, h, r, err
    h ← h/4
END FOR

```

Here is a MATLAB script `deriv.m` for simulating the limit.

```

num_iter = input('Enter number of iterations ');

exact = cos(0.5);
fprintf('Exact value = %.10e\n', exact);
fprintf('%5s %18s %18s %18s\n', 'n', 'h', 'r', 'err');
h = 1.0;
for n = 1 : num_iter
    r = (sin(0.5+h) - sin(0.5)) / h;
    err = exact - r;
    fprintf('%5d %18.10e %18.10e %18.10e\n', n, h, r, err);
    h = h / 4.0;
end

```

Here is some output:

```

>> deriv
Enter number of iterations 30
Exact value = 8.7758256189e-001

```

n	h	r	err
1	1.0000000000e+000	5.1806944800e-001	3.5951311389e-001
2	2.5000000000e-001	8.0885288568e-001	6.8729676214e-002
3	6.2500000000e-002	8.6203415891e-001	1.5548402981e-002
4	1.5625000000e-002	8.7380141758e-001	3.7811443083e-003
5	3.9062500000e-003	8.7664395327e-001	9.3860862058e-004
...			
12	2.3841857910e-007	8.7758250465e-001	5.7243520146e-008
13	5.9604644775e-008	8.7758254725e-001	1.4635512358e-008
14	1.4901161194e-008	8.7758255750e-001	4.3909640368e-009
15	3.7252902985e-009	8.7758256495e-001	-3.0596165601e-009
16	9.3132257462e-010	8.7758255005e-001	1.1841544634e-008
17	2.3283064365e-010	8.7758255005e-001	1.1841544634e-008
18	5.8207660913e-011	8.7758255005e-001	1.1841544634e-008
19	1.4551915228e-011	8.7758255005e-001	1.1841544634e-008
20	3.6379788071e-012	8.7757873535e-001	3.8265388103e-006
21	9.0949470177e-013	8.7756347656e-001	1.9085327873e-005
22	2.2737367544e-013	8.7744140625e-001	1.4115564037e-004
23	5.6843418861e-014	8.7792968750e-001	-3.4712560963e-004
24	1.4210854715e-014	8.7890625000e-001	-1.3236881096e-003
25	3.5527136788e-015	8.7500000000e-001	2.5825618904e-003
26	8.8817841970e-016	8.7500000000e-001	2.5825618904e-003
27	2.2204460493e-016	7.5000000000e-001	1.2758256189e-001
28	5.5511151231e-017	0.0000000000e+000	8.7758256189e-001
29	1.3877787808e-017	0.0000000000e+000	8.7758256189e-001

30 3.4694469520e-018 0.0000000000e+000 8.7758256189e-001

Can you explain why we eventually get zero for an answer?

8 Recurrence relations and iterative algorithms

Recurrence relations can often be used to define sequences and functions. Some are stable and others are unstable but can be made stable by backward recurrence. Here we consider one recurrence relation of each type.

8.1 A stable recurrence relation

Consider the integrals y_n defined by

$$y_n = \int_0^1 \frac{x^n}{4x+1} dx, \quad n \geq 0. \quad (8.1)$$

We can obtain a recurrence relation for these integrals. The first one is

$$y_0 = \int_0^1 \frac{dx}{4x+1} = \frac{1}{4} \int_0^1 \frac{4dx}{4x+1} = \frac{1}{4} \ln(1+4x) \Big|_0^1 = \frac{1}{4} \ln(5) - \frac{1}{4} \ln(1) = \frac{1}{4} \ln(5) \quad (8.2)$$

Other values of y_n are difficult to calculate directly but we can develop a recurrence relation for them as follows

$$\begin{aligned} y_{n+1} &= \int_0^1 \frac{x^{n+1}}{4x+1} dx = \frac{1}{4} \int_0^1 x^n \left(\frac{4x}{4x+1} \right) dx = \frac{1}{4} \int_0^1 x^n \left(\frac{4x+1-1}{4x+1} \right) dx \\ &= \frac{1}{4} \int_0^1 x^n dx - \frac{1}{4} \int_0^1 \frac{x^n}{4x+1} dx \\ &= \frac{1}{4} \left(\frac{1}{n+1} - y_n \right) \end{aligned} \quad (8.3)$$

Therefore we obtain the recurrence relation

$$y_{n+1} = \frac{1}{4} \left(\frac{1}{n+1} - y_n \right), \quad y_0 = \frac{1}{4} \ln(5), \quad n \geq 0 \quad (8.4)$$

As n increases we expect this recurrence relation to be stable: any error in y_n should decrease since y_n is multiplied by $1/4$ with each iteration.

Pseudo-code algorithm Here is a pseudo-code algorithm for computing y_n beginning with $n = 0$:

```

y ← ln(5)/4
FOR n ← 0 TO 20 DO
  PRINT n,y
  
$$y \leftarrow \frac{1}{4} \left( \frac{1}{n+1} - y \right)$$

END FOR

```

Here is a MATLAB script `recurrencel.m` that evaluates the y_n and also compares them with the numerical (exact) results obtained using the MATLAB `quadl` function for numerical integration.

```

f = @(x,n) x.^n./(4*x+1); % integrand
y = log(5)/4; % first integral y0

fprintf('%5s %23s %23s %23s\n', 'n', 'y', 'yexact', 'yerror');
for n = 0:20
    yexact = quadl(@(x) f(x,n), 0, 1, 1E-15);
    yerror = yexact - y;
    fprintf('%5d %23.15e %23.15e %23.15e\n', n, y, yexact, yerror);
    y = 0.25*(1/(n+1) - y);
end

```

Some output is

```

recurrencel
   n                y                yexact                yerror
   0  4.023594781085251e-001  4.023594781085251e-001  0.000000000000000e+000
   1  1.494101304728687e-001  1.494101304728687e-001  0.000000000000000e+000
   2  8.764746738178282e-002  8.764746738178282e-002  0.000000000000000e+000
   3  6.142146648788763e-002  6.142146648788764e-002  1.387778780781446e-017
   4  4.714463337802809e-002  4.714463337802810e-002  6.938893903907228e-018
   5  3.821384165549298e-002  3.821384165549298e-002  0.000000000000000e+000
   6  3.211320625279342e-002  3.211320625279342e-002  6.938893903907228e-018
   7  2.768598415108736e-002  2.768598415108736e-002 -3.469446951953614e-018
   8  2.432850396222816e-002  2.432850396222828e-002  1.249000902703301e-016
   9  2.169565178722074e-002  2.169565178722071e-002 -2.428612866367530e-017
  10  1.957608705319482e-002  1.957608705319483e-002  1.040834085586084e-017
  11  1.783325096397402e-002  1.783325096397402e-002  0.000000000000000e+000
  12  1.637502059233983e-002  1.637502059233983e-002  0.000000000000000e+000
  13  1.513701408268428e-002  1.513701408268427e-002 -1.734723475976807e-018
  14  1.407288933647179e-002  1.407288933647179e-002  1.734723475976807e-018
  15  1.314844433254872e-002  1.314844433254872e-002  0.000000000000000e+000
  16  1.233788891686282e-002  1.233788891686282e-002  0.000000000000000e+000
  17  1.162141012372547e-002  1.162141012372547e-002  0.000000000000000e+000
  18  1.098353635795752e-002  1.098353635795760e-002  7.979727989493313e-017

```

```

19 1.041201064735273e-002 1.041201064735276e-002 3.642919299551295e-017
20 9.896997338161819e-003 9.896997338161833e-003 1.387778780781446e-017

```

The results have full accuracy. This is expected since the recurrence relation shows that any error in y_n is multiplied by $1/4$ in the calculation of y_{n+1} .

8.2 An unstable recurrence relation

Now consider the integrals E_n defined as follows.

$$E_n = \int_0^1 x^n e^{x-1} dx, \quad n \geq 0 \quad (8.5)$$

Let us develop a recurrence relation for them. The first integral is

$$E_0 = \int_0^1 e^{x-1} dx = e^{x-1} \Big|_0^1 = 1 - \frac{1}{e} \quad (8.6)$$

A recurrence relation for E_n can be obtained using integration by parts:

$$E_n = \int_0^1 x^n d(e^{x-1}) = x^n e^{x-1} \Big|_0^1 - n \int_0^1 x^{n-1} e^{x-1} dx = 1 - nE_{n-1} \quad (8.7)$$

This gives the recurrence relation

$$E_0 = 1 - \frac{1}{e}, \quad E_{n+1} = 1 - (n+1)E_n, \quad n = 0, 1, 2, \dots \quad (8.8)$$

Now any error in E_n is magnified by $n+1$ at each iteration. Thus any error is magnified at each iteration. After a few iteration the results should be meaningless.

Pseudo-code algorithm Here is a pseudo-code algorithm for computing E_n beginning with $n = 0$:

```

E ← 1 - 1/e
FOR n ← 0 TO 20 DO
  PRINT n, E
  E ← 1 - (n+1)E
END FOR

```

Here is a MATLAB script `recurrence2.m` for computing the sequence.

```

n_max = input('Enter number of iterations ');
f = @(x,n) x.^n.*exp(x-1);
E = 1 - exp(-1);
fprintf(1, '%5s %18s %18s %18s\n', 'n', 'E', 'E_exact', 'E_error');

```

```

for n = 0:n_max
    E_exact = quadl(@(x) f(x,n), 0, 1, 1E-15);
    E_error = E_exact - E;
    fprintf(1, '%5d %18.10e %18.10e %18.10e\n', n, E, E_exact, E_error);
    E = 1 - (n+1)*E;
end

```

Some output is

```

>> recurrence2
Enter number of iterations 25

```

n	E	E_exact	E_error
0	6.3212055883e-001	6.3212055883e-001	-1.1102230246e-016
1	3.6787944117e-001	3.6787944117e-001	0.0000000000e+000
2	2.6424111766e-001	2.6424111766e-001	5.5511151231e-017
3	2.0727664703e-001	2.0727664703e-001	-5.5511151231e-017
4	1.7089341189e-001	1.7089341189e-001	3.3306690739e-016
5	1.4553294057e-001	1.4553294057e-001	-1.4710455076e-015
6	1.2680235656e-001	1.2680235656e-001	8.9650509238e-015
7	1.1238350407e-001	1.1238350407e-001	-6.2630456377e-014
8	1.0093196745e-001	1.0093196745e-001	5.0112691774e-013
9	9.1612292994e-002	9.1612292990e-002	-4.5101422597e-012
10	8.3877070058e-002	8.3877070103e-002	4.5101464230e-011
11	7.7352229359e-002	7.7352228863e-002	-4.9611606490e-010
12	7.1773247695e-002	7.1773253648e-002	5.9533928898e-009
13	6.6947779970e-002	6.6947702576e-002	-7.7394107650e-008
14	6.2731080424e-002	6.2732163941e-002	1.0835175069e-006
15	5.9033793642e-002	5.9017540879e-002	-1.6252762604e-005
16	5.5459301730e-002	5.5719345931e-002	2.6004420167e-004
17	5.7191870597e-002	5.2771119169e-002	-4.4207514283e-003
18	-2.9453670752e-002	5.0119854958e-002	7.9573525710e-002
19	1.5596197443e+000	4.7722755796e-002	-1.5118969885e+000
20	-3.0192394886e+001	4.5544884076e-002	3.0237939770e+001
21	6.3504029260e+002	4.3557434408e-002	-6.3499673516e+002
22	-1.3969886437e+004	4.1736443028e-002	1.3969928174e+004
23	3.2130838805e+005	4.0061810357e-002	-3.2130834799e+005
24	-7.7114003133e+006	3.8516551423e-002	7.7114003518e+006
25	1.9278500883e+008	3.7086214424e-002	-1.9278500880e+008

As expected the results are meaningless. In fact it is clear that each E_n is the integral of a positive function so the E_n should all be positive yet some of the values we get are negative. Also we can show that $\lim_{n \rightarrow \infty} E_n = 0$ as follows

$$0 \leq E_n = \int_0^1 x^n e^{x-1} dx \leq \int_0^1 x^n dx, \quad \text{since } e^{x-1} \leq e^{1-1} = 1 \text{ for } 0 \leq x \leq 1$$

$$= \frac{1}{n+1}$$

Therefore we have $0 \leq E_n \leq \frac{1}{n+1}$ which shows that $\lim_{n \rightarrow \infty} E_n = 0$. However the values we get seem to be going to infinity.

8.3 Backward recurrence

We can convert this unstable forward recurrence relation to a stable backward recurrence relation. Since $\lim_{n \rightarrow \infty} E_n = 0$ let us assume for example that $E_{30} = 0$ and try to calculate $E_{29}, E_{28}, \dots, E_0$ using the backward recurrence

$$E_n = \frac{1}{n+1} (1 - E_{n+1}), \quad E_{30} = 0, \quad n = 29, 28, \dots, 0 \quad (8.9)$$

Notice that any error in E_{n+1} is now multiplied by $1/(n+1)$ as n decreases so any initial error in E_{30} will disappear after a few iterations.

Pseudo-code algorithm Here is a pseudo-code algorithm for computing E_n beginning with $n = 30$ and going backward:

```

E ← 0
FOR n ← 30 TO 1 BY -1 DO
  PRINT n, E
  E ← (1/n) * (1 - E)
END FOR

```

Here is a MATLAB script `recurrence3.m` for computing the sequence

```

f = @(x,n) x.^n.*exp(x-1);
E = 0; % E(30)

fprintf(1, '%5s %18s %18s %18s\n', 'n', 'E', 'E_exact', 'E_error');
for n = 30:-1:1
    E_exact = quadl(@(x) f(x,n), 0, 1, 1E-15);
    E_error = E_exact - E;
    fprintf(1, '%5d %18.10e %18.10e %18.10e\n', n, E, E_exact, E_error);
    E = (1/n)*(1 - E);
end

```

Some output is

```

>> recurrence3
      n                E                E_exact                E_error

```

30	0.0000000000e+000	3.1279673932e-002	3.1279673932e-002
29	3.3333333333e-002	3.2290677536e-002	-1.0426557977e-003
28	3.3333333333e-002	3.3369286982e-002	3.5953648198e-005
27	3.4523809524e-002	3.4522525465e-002	-1.2840588642e-006
26	3.5758377425e-002	3.5758424983e-002	4.7557735715e-008
25	3.7086216253e-002	3.7086214424e-002	-1.8291436768e-009
24	3.8516551350e-002	3.8516551423e-002	7.3165744296e-011
23	4.0061810360e-002	4.0061810357e-002	-3.0485752811e-012
22	4.1736443028e-002	4.1736443028e-002	1.3254675135e-013
21	4.3557434408e-002	4.3557434408e-002	-6.0160210147e-015
20	4.5544884076e-002	4.5544884076e-002	3.1225022568e-016
19	4.7722755796e-002	4.7722755796e-002	4.1633363423e-017
18	5.0119854958e-002	5.0119854958e-002	1.1796119637e-016
17	5.2771119169e-002	5.2771119169e-002	0.0000000000e+000
16	5.5719345931e-002	5.5719345931e-002	0.0000000000e+000
15	5.9017540879e-002	5.9017540879e-002	-6.9388939039e-018
14	6.2732163941e-002	6.2732163941e-002	0.0000000000e+000
13	6.6947702576e-002	6.6947702576e-002	-1.3877787808e-017
12	7.1773253648e-002	7.1773253648e-002	-1.3877787808e-017
11	7.7352228863e-002	7.7352228863e-002	1.3877787808e-017
10	8.3877070103e-002	8.3877070103e-002	0.0000000000e+000
9	9.1612292990e-002	9.1612292990e-002	0.0000000000e+000
8	1.0093196745e-001	1.0093196745e-001	0.0000000000e+000
7	1.1238350407e-001	1.1238350407e-001	1.3877787808e-017
6	1.2680235656e-001	1.2680235656e-001	2.7755575616e-017
5	1.4553294057e-001	1.4553294057e-001	2.7755575616e-017
4	1.7089341189e-001	1.7089341189e-001	0.0000000000e+000
3	2.0727664703e-001	2.0727664703e-001	2.7755575616e-017
2	2.6424111766e-001	2.6424111766e-001	5.5511151231e-017
1	3.6787944117e-001	3.6787944117e-001	0.0000000000e+000

The results except for the first few values $E_{30}, E_{29}, \dots, E_k$ are accurate to at least 10 significant figures. To find out what k is just redo the calculations using say $E_{40} = 0$ and compare.

9 Slow and fast convergence

Let us find the root of $x = e^{-x}$. Graphically it is at the intersection of the line $y = x$ and the curve $y = e^{-x}$ near $x = 0.5$.

9.1 Slow convergence

We can use an initial approximation $x_0 = 0.5$ and find better approximations by using the iteration process $x_n = e^{-x_{n-1}}$ for $n \geq 1$.

Pseudo-code algorithm Here is a pseudo-code algorithm to compute these iterations

```
x ← 0.5
FOR n ← 1 TO 20 DO
  x ← e-x
  PRINT n,x
END FOR
```

Here is a MATLAB script `slowroot.m` to do the iterations.

```
x0 = 0.5;
x = x0;
for n = 1:20
    x = exp(-x);
    fprintf(1, '%5d %25.15e\n', n, x);
end
```

Some output is

```
>> slowroot
 1    6.065306597126334e-001
 2    5.452392118926051e-001
 3    5.797030948780683e-001
 4    5.600646279389019e-001
 5    5.711721489772151e-001
 6    5.648629469803235e-001
 7    5.684380475700662e-001
 8    5.664094527469208e-001
 9    5.675596342622424e-001
10    5.669072129354714e-001
11    5.672771959707785e-001
12    5.670673518537281e-001
13    5.671863600876381e-001
14    5.671188642569858e-001
15    5.671571437076446e-001
16    5.671354336592732e-001
17    5.671477463306249e-001
18    5.671407632698067e-001
19    5.671447236620769e-001
20    5.671424775509449e-001
```

Convergence here is slow since 20 iterations are needed to obtain the exact value 0.56714 to 5 significant figures.

9.2 Fast convergence

A faster sequence of iterations is given by

$$x_n = g(x_{n-1}), \quad \text{where } g(x) = x + \frac{e^{-x} - x}{e^{-x} + 1} \quad (9.1)$$

Here is a pseudo-code algorithm to compute these iterations.

```

x ← 0.5
FOR n ← 1 TO 6 DO
    x ← x +  $\frac{e^{-x} - x}{e^{-x} + 1}$ 
    PRINT n, x
END FOR

```

Here is a MATLAB script `fastroot.m` to compute these iterations.

```

x0 = 0.5;
x = x0;
for n = 1:6
    e = exp(-x);
    x = x + (e - x)/(e + 1);
    fprintf(1, '%5d %25.15e\n', n, x);
end:

```

Some output is

```

>> fastroot
    1    5.663110031972182e-001
    2    5.671431650348623e-001
    3    5.671432904097810e-001
    4    5.671432904097838e-001
    5    5.671432904097840e-001
    6    5.671432904097838e-001

```

Convergence here is much faster since only 3 iterations are needed to obtain the exact value 0.56714 to 5 significant figures. This iteration scheme was obtained using Newton's method.