Numerical Analysis

Prof.dr.hab. Bostan Viorel

Calculation of functions

Using hand calculations, a hand calculator, or a computer, what are the basic operations of which we are capable?

Calculation of functions

Using hand calculations, a hand calculator, or a computer, what are the basic operations of which we are capable?

Addition, subtraction, multiplication, and division

Calculation of functions

Using hand calculations, a hand calculator, or a computer, what are the basic operations of which we are capable?

Addition, subtraction, multiplication, and division

(and even this will usually require a truncation of the quotient at some point).

Calculation of functions

Using hand calculations, a hand calculator, or a computer, what are the basic operations of which we are capable?

Addition, subtraction, multiplication, and division

(and even this will usually require a truncation of the quotient at some point).

In addition, we can make logical decisions, such as deciding:

$$a > b, \quad a = b, \quad a < b$$

Calculation of functions

Using hand calculations, a hand calculator, or a computer, what are the basic operations of which we are capable?

Addition, subtraction, multiplication, and division

(and even this will usually require a truncation of the quotient at some point).

In addition, we can make logical decisions, such as deciding:

$$a > b, \quad a = b, \quad a < b$$

Furthermore, we can carry out only a finite number of such operations.

Calculation of functions

In evaluating functions $f(x)$ we are limited to the evaluation of polynomials:

Calculation of functions

In evaluating functions $f(x)$ we are limited to the evaluation of polynomials:

$$p(x) = a_0 + a_1 x + a_2 x^2 + \ldots a_{n-1} x^{n-1} + a_n x^n$$

Calculation of functions

In evaluating functions $f(x)$ we are limited to the evaluation of polynomials:

$$p(x) = a_0 + a_1 x + a_2 x^2 + \dots a_{n-1} x^{n-1} + a_n x^n$$

We will discuss later how to evaluate **efficiently** the polynomials;

Calculation of functions

In evaluating functions $f(x)$ we are limited to the evaluation of polynomials:

$$p(x) = a_0 + a_1 x + a_2 x^2 + \dots a_{n-1} x^{n-1} + a_n x^n$$

We will discuss later how to evaluate **efficiently** the polynomials;

How to evaluate other functions such as

$$e^x, \quad \cos x, \quad \log x, \quad \tan x, \quad \text{etc?}$$

Taylor polynomial approximations

Consider function $f(x) = e^x$

Taylor polynomial approximations

Consider function $f(x) = e^x$

Consider evaluating it for x near to 0.

Taylor polynomial approximations

Consider function $f(x) = e^x$

Consider evaluating it for x near to 0.

Look for a polynomial p(x) whose values will be the same as those of e^x within acceptable accuracy.

Taylor polynomial approximations

Consider function $f(x) = e^x$

Consider evaluating it for x near to 0.

Look for a polynomial p(x) whose values will be the same as those of e^x to within acceptable accuracy.

Begin with a linear polynomial $P_1(x) = a_0 + a_1 x$.

Taylor polynomial approximations

Consider function $f(x) = e^x$

Consider evaluating it for x near to 0.

Look for a polynomial p(x) whose values will be the same as those of e^x to within acceptable accuracy.

Begin with a linear polynomial $P_1(x) = a_0 + a_1 x$.

Then to make its graph look like that of e^x, we ask that the graph of $y = P_1(x)$ be tangent to that of $y = e^x$ at $x = 0$.

Taylor polynomial approximations

Consider function $f(x) = e^x$

Consider evaluating it for x near to 0.

Look for a polynomial p(x) whose values will be the same as those of e^x to within acceptable accuracy.

Begin with a linear polynomial $P_1(x) = a_0 + a_1 x$.

Then to make its graph look like that of e^x, we ask that the graph of $y = P_1(x)$ be tangent to that of $y = e^x$ at $x = 0$.

and we get $P_1(x) = 1 + x$.

Taylor polynomial approximations

Taylor polynomial approximations

Look for a quadratic polynomial $P_2(x) = a_0 + a_1 x + a_2 x^2$

Taylor polynomial approximations

Look for a quadratic polynomial $P_2(x) = a_0 + a_1 x + a_2 x^2$

Make it tangent; and to determine a_2, ask that $P_2(x)$ and e^x have the same "curvature" at the origin.

Taylor polynomial approximations

Look for a quadratic polynomial $P_2(x) = a_0 + a_1 x + a_2 x^2$

Make it tangent; and to determine a_2, ask that $P_2(x)$ and e^x have the same "curvature" at the origin.

Combining these requirements, we have for $f(x) = e^x$ that

$$P_2(0) = f(0), \quad P_2'(0) = f'(0), \quad P_2''(0) = f''(0)$$

Taylor polynomial approximations

Look for a quadratic polynomial $P_2(x) = a_0 + a_1 x + a_2 x^2$

Make it tangent; and to determine a_2, ask that $P_2(x)$ and e^x have the same "curvature" at the origin.

Combining these requirements, we have for $f(x) = e^x$ that

$$P_2(0) = f(0), \quad P_2'(0) = f'(0), \quad P_2''(0) = f''(0)$$

This yields the approximation

$$P_2(x) = 1 + x + \frac{1}{2}x^2.$$

Taylor polynomial approximations

We continue this pattern, looking for a polynomial

$$P_n(x) = a_0 + a_1 x + a_2 x^2 + \dots a_{n-1} x^{n-1} + a_n x^n$$

Taylor polynomial approximations

We continue this pattern, looking for a polynomial

$$P_n(x) = a_0 + a_1 x + a_2 x^2 + \dots a_{n-1} x^{n-1} + a_n x^n$$

Such that

$$P_n(0) = f(0), \quad P_n'(0) = f'(0), \quad P_n''(0) = f''(0), \quad P_n^{(n)}(0) = f^{(n)}(0)$$

Taylor polynomial approximations

We continue this pattern, looking for a polynomial

$$P_n(x) = a_0 + a_1 x + a_2 x^2 + \dots a_{n-1} x^{n-1} + a_n x^n$$

Such that

$$P_n(0) = f(0), \quad P_n'(0) = f'(0), \quad P_n''(0) = f''(0), \quad P_n^{(n)}(0) = f^{(n)}(0)$$

This leads to the formula

$$P_n(x) = 1 + x + \frac{1}{2!}x^2 + \frac{1}{3!}x^3 + \frac{1}{4!}x^4 + \dots + \frac{1}{n!}x^n, \quad x \in \mathbb{R}$$

Taylor polynomial approximations

We continue this pattern, looking for a polynomial

$$P_n(x) = a_0 + a_1 x + a_2 x^2 + \dots a_{n-1} x^{n-1} + a_n x^n$$

Such that

$$P_n(0) = f(0), \quad P_n'(0) = f'(0), \quad P_n''(0) = f''(0), \quad P_n^{(n)}(0) = f^{(n)}(0)$$

This leads to the formula

$$P_n(x) = 1 + x + \frac{1}{2!}x^2 + \frac{1}{3!}x^3 + \frac{1}{4!}x^4 + \dots + \frac{1}{n!}x^n, \quad x \in \mathbb{R}$$

In the vicinity of 0 we will have an approximation of $f(x) = e^x$ by polynomials $P_n(x)$:

$$e^x \approx P_n(x)$$

Taylor polynomial approximations

We continue this pattern, looking for a polynomial

$$P_n(x) = a_0 + a_1 x + a_2 x^2 + \dots a_{n-1} x^{n-1} + a_n x^n$$

Such that

$$P_n(0) = f(0), \quad P_n'(0) = f'(0), \quad P_n''(0) = f''(0), \quad P_n^{(n)}(0) = f^{(n)}(0)$$

This leads to the formula

$$P_n(x) = 1 + x + \frac{1}{2!}x^2 + \frac{1}{3!}x^3 + \frac{1}{4!}x^4 + \dots + \frac{1}{n!}x^n, \quad x \in \mathbb{R}$$

In the vicinity of 0 we will have an approximation of $f(x) = e^x$ by polynomials $P_n(x)$:

$$e^x \approx P_n(x)$$

Moreover, as $n \to \infty$ it can be shown that $P_n(x) \to e^x$

Taylor polynomial approximations

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \ldots, \quad x \in \mathbb{R}$$

Taylor polynomial approximations

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \ldots, \quad x \in \mathbb{R}$$

Taylor polynomial approximations

What are the problems when evaluating points x that are far from 0?

Taylor polynomial approximations

What are the problems when evaluating points x that are far from 0?

Taylor polynomial error formula

Let $f(x)$ be a given function, and assume it has derivatives around some point $x = a$ (with as many derivatives as we find necessary).

Taylor polynomial error formula

Let $f(x)$ be a given function, and assume it has derivatives around some point $x = a$ (with as many derivatives as we find necessary).

For the error in the Taylor polynomial $P_n(x)$, we have the formulas

Taylor polynomial error formula

Let $f(x)$ be a given function, and assume it has derivatives around some point $x = a$ (with as many derivatives as we find necessary).

For the error in the Taylor polynomial $P_n(x)$, we have the formulas

$$f(x) - P_n(x) = \frac{1}{(n+1)!}(x-a)^{n+1} f^{(n+1)}(\xi)$$

Taylor polynomial error formula

Let $f(x)$ be a given function, and assume it has derivatives around some point $x = a$ (with as many derivatives as we find necessary).

For the error in the Taylor polynomial $P_n(x)$, we have the formulas

$$f(x) - P_n(x) = \frac{1}{(n+1)!}(x-a)^{n+1} f^{(n+1)}(\xi)$$

or

$$f(x) - P_n(x) = \frac{1}{n!}\int_a^x (x-t)^n f^{n+1}(t)\,dt$$

Taylor polynomial error formula

Let $f(x)$ be a given function, and assume it has derivatives around some point $x = a$ (with as many derivatives as we find necessary).

For the error in the Taylor polynomial $P_n(x)$, we have the formulas

$$f(x) - P_n(x) = \frac{1}{(n+1)!}(x-a)^{n+1} f^{(n+1)}(\xi)$$

or

$$f(x) - P_n(x) = \frac{1}{n!}\int_a^x (x-t)^n f^{n+1}(t)\,dt$$

where point ξ is restricted to the interval bounded by x and a, and otherwise ξ is unknown.

Taylor polynomial error formula

For the case $a = 0$ we have

Taylor polynomial error formula

For the case $a = 0$ we have

For the error in the Taylor polynomial $P_n(x)$, we have the formulas

$$f(x) - P_n(x) = \frac{1}{(n+1)!} x^{n+1} f^{(n+1)}(\xi)$$

or

$$f(x) - P_n(x) = \frac{1}{n!} \int_0^x (x-t)^n f^{n+1}(t) dt$$

Taylor polynomial error formula

For the case $a = 0$ we have

For the error in the Taylor polynomial $P_n(x)$, we have the formulas

$$f(x) - P_n(x) = \frac{1}{(n+1)!} x^{n+1} f^{(n+1)}(\xi)$$

or

$$f(x) - P_n(x) = \frac{1}{n!} \int_0^x (x-t)^n f^{n+1}(t) dt$$

where point ξ is restricted to the interval bounded by x and 0, and otherwise ξ is unknown.

Taylor polynomial approximations

For the case of considered function $f(x) = e^x$ we get

Taylor polynomial approximations

For the case of considered function $f(x) = e^x$ we get

$$e^x - P_n(x) = \frac{1}{(n+1)!} x^n e^{\xi}$$

with ξ located between 0 and x.

Taylor polynomial approximations

For the case of considered function $f(x) = e^x$ we get

$$e^x - P_n(x) = \frac{1}{(n+1)!} x^n e^{\xi}$$

with ξ located between 0 and x.

Note that for $x \approx 0$, we must have $\xi \approx 0$ and

$$e^x - P_n(x) \approx \frac{1}{(n+1)!} x^n$$

Taylor polynomial approximations

For the case of considered function $f(x) = e^x$ we get

$$e^x - P_n(x) = \frac{1}{(n+1)!} x^n e^{\xi}$$

with ξ located between 0 and x.

Note that for $x \approx 0$, we must have $\xi \approx 0$ and

$$e^x - P_n(x) \approx \frac{1}{(n+1)!} x^n$$

This last term is also the final term in $P_{n+1}(x)$, and thus

$$e^x - P_n(x) \approx P_{n+1}(x) - P_n(x)$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + \frac{x^{12}}{12!} - \frac{x^{14}}{14!} + \frac{x^{16}}{16!} - \frac{x^{18}}{18!} + \dots$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + \frac{x^{12}}{12!} - \frac{x^{14}}{14!} + \frac{x^{16}}{16!} - \frac{x^{18}}{18!} + \dots$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + \frac{x^{12}}{12!} - \frac{x^{14}}{14!} + \frac{x^{16}}{16!} - \frac{x^{18}}{18!} + \dots$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + \frac{x^{12}}{12!} - \frac{x^{14}}{14!} + \frac{x^{16}}{16!} - \frac{x^{18}}{18!} + \dots$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + \frac{x^{12}}{12!} - \frac{x^{14}}{14!} + \frac{x^{16}}{16!} - \frac{x^{18}}{18!} + \dots$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + \frac{x^{12}}{12!} - \frac{x^{14}}{14!} + \frac{x^{16}}{16!} - \frac{x^{18}}{18!} + \dots$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + \frac{x^{12}}{12!} - \frac{x^{14}}{14!} + \frac{x^{16}}{16!} - \frac{x^{18}}{18!} + \dots$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + \frac{x^{12}}{12!} - \frac{x^{14}}{14!} + \frac{x^{16}}{16!} - \frac{x^{18}}{18!} + \dots$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + \frac{x^{12}}{12!} - \frac{x^{14}}{14!} + \frac{x^{16}}{16!} - \frac{x^{18}}{18!} + \dots$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + \frac{x^{12}}{12!} - \frac{x^{14}}{14!} + \frac{x^{16}}{16!} - \frac{x^{18}}{18!} + \dots$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + \frac{x^{12}}{12!} - \frac{x^{14}}{14!} + \frac{x^{16}}{16!} - \frac{x^{18}}{18!} + \dots$$

Taylor polynomial approximations

$$\frac{1}{1-x} = 1 + x + x^2 + x^3 + x^4 + x^5 + x^6 + \ldots, \quad |x| < 1$$

Function whose Taylor series doesn't converge to the function itself

$$f(x) = \begin{cases} 0, & if \quad x = 0 \\ e^{-\frac{1}{x^2}}, & if \quad x \neq 0 \end{cases}$$

Computing approximation to e

Coming back to earlier approximation

$$e^x \approx P_n(x) = 1 + x + \frac{1}{2!}x^2 + \frac{1}{3!}x^3 + \frac{1}{4!}x^4 + \ldots + \frac{1}{n!}x^n, \quad x \in \mathbb{R}$$

Computing approximation to e

Coming back to earlier approximation

$$e^x \approx P_n(x) = 1 + x + \frac{1}{2!}x^2 + \frac{1}{3!}x^3 + \frac{1}{4!}x^4 + \ldots + \frac{1}{n!}x^n, \quad x \in \mathbb{R}$$

Then let $x = 1$ to get

$$e \approx P_n(1) = 1 + 1 + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \ldots + \frac{1}{n!}, \quad x \in \mathbb{R}$$

Computing approximation to e

Coming back to earlier approximation

$$e^x \approx P_n(x) = 1 + x + \frac{1}{2!}x^2 + \frac{1}{3!}x^3 + \frac{1}{4!}x^4 + \ldots + \frac{1}{n!}x^n, \quad x \in \mathbb{R}$$

Then let $x = 1$ to get

$$e \approx P_n(1) = 1 + 1 + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \ldots + \frac{1}{n!}, \quad x \in \mathbb{R}$$

For the error,

$$e - P_n(1) = \frac{1}{(n+1)!}e^{\xi}, \qquad 0 \leq \xi \leq 1$$

Computing approximation to e

Coming back to earlier approximation

$$e^x \approx P_n(x) = 1 + x + \frac{1}{2!}x^2 + \frac{1}{3!}x^3 + \frac{1}{4!}x^4 + \ldots + \frac{1}{n!}x^n, \quad x \in \mathbb{R}$$

Then let $x = 1$ to get

$$e \approx P_n(1) = 1 + 1 + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \ldots + \frac{1}{n!}, \quad x \in \mathbb{R}$$

For the error,

$$e - P_n(1) = \frac{1}{(n+1)!}e^{\xi}, \qquad 0 \leq \xi \leq 1$$

To bound the error, we have

$$e^0 \leq e^{\xi} \leq e^1$$

Computing approximation to e

Coming back to earlier approximation

$$e^x \approx P_n(x) = 1 + x + \frac{1}{2!}x^2 + \frac{1}{3!}x^3 + \frac{1}{4!}x^4 + \ldots + \frac{1}{n!}x^n, \quad x \in \mathbb{R}$$

Then let $x = 1$ to get

$$e \approx P_n(1) = 1 + 1 + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \ldots + \frac{1}{n!}, \quad x \in \mathbb{R}$$

For the error,

$$e - P_n(1) = \frac{1}{(n+1)!}e^{\xi}, \qquad 0 \leq \xi \leq 1$$

To bound the error, we have

$$e^0 \leq e^{\xi} \leq e^1$$

$$\frac{1}{(n+1)!} \leq e - P_n(1) \leq \frac{1}{(n+1)!}e$$

Taylor polynomial approximations

Suppose we would like to have an approximation accurate for e within 10^{-5}.

Taylor polynomial approximations

Suppose we would like to have an approximation accurate for e within 10^{-5}.

How large we should choose n?

Taylor polynomial approximations

Suppose we would like to have an approximation accurate for e within 10^{-5}.

How large we should choose n?

From the formula

$$\frac{1}{(n+1)!} \leq e - P_n(1) \leq \frac{1}{(n+1)!} e$$

Taylor polynomial approximations

Suppose we would like to have an approximation accurate for e within 10^{-5}.

How large we should choose n?

From the formula

$$\frac{1}{(n+1)!} \leq e - P_n(1) \leq \frac{1}{(n+1)!} e$$

we choose n large enough to have

$$\frac{e}{(n+1)!} \leq 10^{-5}$$

Taylor polynomial approximations

Suppose we would like to have an approximation accurate for e within 10^{-5}.

How large we should choose n?

From the formula

$$\frac{1}{(n+1)!} \leq e - P_n(1) \leq \frac{1}{(n+1)!} e$$

we choose n large enough to have

$$\frac{e}{(n+1)!} \leq 10^{-5}$$

which is true if $n \geq 8$.

Taylor polynomial approximations

In fact,

$$e - P_8(1) \leq \frac{e}{9!} \approx 7.5 \cdot 10^{-6}$$

Taylor polynomial approximations

In fact,

$$e - P_8(1) \leq \frac{e}{9!} \approx 7.5 \cdot 10^{-6}$$

Then calculate $P_8(1)$:

$$P_8(1) = 2 + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \ldots + \frac{1}{8!} = 2.71827877,$$

Taylor polynomial approximations

In fact,

$$e - P_8(1) \leq \frac{e}{9!} \approx 7.5 \cdot 10^{-6}$$

Then calculate $P_8(1)$:

$$P_8(1) = 2 + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \ldots + \frac{1}{8!} = 2.71827877,$$

and true error is

$$e - P_8(1) \approx 3.06 \cdot 10^{-6}$$

Evaluation of polynomials

Consider having a polynomial

$$p(x) = a_0 + a_1 x + a_2 x^2 + \dots a_{n-1} x^{n-1} + a_n x^n$$

Evaluation of polynomials

Consider having a polynomial

$$p(x) = a_0 + a_1 x + a_2 x^2 + \dots a_{n-1} x^{n-1} + a_n x^n$$

which you need to evaluate for many values of x.

Evaluation of polynomials

Consider having a polynomial

$$p(x) = a_0 + a_1 x + a_2 x^2 + \dots a_{n-1} x^{n-1} + a_n x^n$$

which you need to evaluate for many values of x.

How do you evaluate it?

Evaluation of polynomials

Consider having a polynomial

$$p(x) = a_0 + a_1 x + a_2 x^2 + \dots a_{n-1} x^{n-1} + a_n x^n$$

which you need to evaluate for many values of x.

How do you evaluate it?

This may seem a strange question, but the answer is not as obvious as you might think.

Evaluation of polynomials

The standard way, written in a loose algorithmic format is:

Evaluation of polynomials

The standard way, written in a loose algorithmic format is:

```
poly = a(0)
for j = 1 : n
    poly = poly + a(j)*x^j
end
```

Evaluation of polynomials

The standard way, written in a loose algorithmic format is:

```
poly = a(0)
for j = 1 : n
    poly = poly + a(j)*x^j
end
```

To compare the costs of different numerical methods, we do an operations count, and then we compare these for the competing methods.

Evaluation of polynomials

The standard way, written in a loose algorithmic format is:

```
poly = a(0)
for j = 1 : n
    poly = poly + a(j)*x^j
end
```

To compare the costs of different numerical methods, we do an operations count, and then we compare these for the competing methods.

Above, the counts are as follows:

$$\begin{aligned} \textit{additions} &: \quad n \\ \textit{multiplications} &: \quad 1+2+3+\cdots+n = \frac{n(n+1)}{2} \end{aligned}$$

Evaluation of polynomials

Next, do the terms x^j recursively:

$$x^j = x \cdot x^{j-1}$$

Evaluation of polynomials

Next, do the terms x^j recursively:

$$x^j = x \cdot x^{j-1}$$

Then to compute $\{x^2, x^3, \ldots, x^n\}$ will cost $n - 1$ multiplications.

Evaluation of polynomials

Next, do the terms x^j recursively:

$$x^j = x \cdot x^{j-1}$$

Then to compute $\{x^2, x^3, \ldots, x^n\}$ will cost $n-1$ multiplications. The evaluation algorithm becomes

```
poly = a(0) + a(1)*x
power = x
for j = 2 : n
   power = x*power
   poly = poly + a(j)*power
end
```

Evaluation of polynomials

Next, do the terms x^j recursively:

$$x^j = x \cdot x^{j-1}$$

Then to compute $\{x^2, x^3, \ldots, x^n\}$ will cost $n-1$ multiplications. The evaluation algorithm becomes

```
poly = a(0) + a(1)*x
power = x
for j = 2 : n
   power = x*power
   poly = poly + a(j)*power
end
```

The total operations cost is

$$\begin{aligned} \textit{additions} &: n \\ \textit{multiplications} &: n + n - 1 = 2n - 1 \end{aligned}$$

Evaluation of polynomials

When n is evenly moderately large, this is much less than for the first method of evaluating $p(x)$.

Evaluation of polynomials

When n is evenly moderately large, this is much less than for the first method of evaluating $p(x)$.

For example, with $n = 20$,

Evaluation of polynomials

When n is evenly moderately large, this is much less than for the first method of evaluating $p(x)$.

For example, with $n = 20$,

the first method has 210 multiplications,

Evaluation of polynomials

When n is evenly moderately large, this is much less than for the first method of evaluating $p(x)$.

For example, with $n = 20$,

the first method has 210 multiplications,

whereas the second has 39 multiplications.

Evaluation of polynomials

Evaluation of polynomials

Consider **nested multiplication**.

Evaluation of polynomials

Consider **nested multiplication**.

As examples of particular degrees, write

$$
\begin{aligned}
n &= 2 : p(x) = a_0 + x(a_1 + a_2 x) \\
n &= 3 : p(x) = a_0 + x(a_1 + x(a_2 + a_3 x)) \\
n &= 4 : p(x) = a_0 + x(a_1 + x(a_2 + x(a_3 + a_4 x)))
\end{aligned}
$$

Evaluation of polynomials

Consider **nested multiplication**.

As examples of particular degrees, write

$$\begin{aligned} n &= 2 : p(x) = a_0 + x(a_1 + a_2 x) \\ n &= 3 : p(x) = a_0 + x(a_1 + x(a_2 + a_3 x)) \\ n &= 4 : p(x) = a_0 + x(a_1 + x(a_2 + x(a_3 + a_4 x))) \end{aligned}$$

These contain, respectively, $2, 3$, and 4 multiplications.

Evaluation of polynomials

Consider **nested multiplication**.

As examples of particular degrees, write

$$\begin{aligned} n &= 2 : p(x) = a_0 + x(a_1 + a_2 x) \\ n &= 3 : p(x) = a_0 + x(a_1 + x(a_2 + a_3 x)) \\ n &= 4 : p(x) = a_0 + x(a_1 + x(a_2 + x(a_3 + a_4 x))) \end{aligned}$$

These contain, respectively, $2, 3$, and 4 multiplications.

This is less than the second method, which would have need $3, 5$, and 7 multiplications,

Evaluation of polynomials

Consider **nested multiplication**.

As examples of particular degrees, write

$$\begin{aligned} n &= 2 : p(x) = a_0 + x(a_1 + a_2 x) \\ n &= 3 : p(x) = a_0 + x(a_1 + x(a_2 + a_3 x)) \\ n &= 4 : p(x) = a_0 + x(a_1 + x(a_2 + x(a_3 + a_4 x))) \end{aligned}$$

These contain, respectively, $2, 3$, and 4 multiplications.

This is less than the second method, which would have need $3, 5$, and 7 multiplications,

The first method will need $3, 6$, and 10 multiplications.

Evaluation of polynomials

For the general case, write

$$p(x) = a_0 + x(a_1 + x(a_2 + \ldots + x(a_{n-1} + a_n x) \ldots))$$

Evaluation of polynomials

For the general case, write

$$p(x) = a_0 + x(a_1 + x(a_2 + \ldots + x(a_{n-1} + a_n x) \ldots))$$

This requires n multiplications, which is only about half that for the preceding method.

Evaluation of polynomials

For the general case, write

$$p(x) = a_0 + x(a_1 + x(a_2 + \ldots + x(a_{n-1} + a_n x) \ldots))$$

This requires n multiplications, which is only about half that for the preceding method.

For an algorithm, write

```
poly = a(n)
for j = n-1 : -1 : 0
   poly = a(j) + x *poly
end
```

Evaluation of polynomials

For the general case, write

$$p(x) = a_0 + x(a_1 + x(a_2 + \ldots + x(a_{n-1} + a_n x) \ldots))$$

This requires n multiplications, which is only about half that for the preceding method.

For an algorithm, write

```
poly = a(n)
for j = n-1 : -1 : 0
   poly = a(j) + x *poly
end
```

With all three methods, the number of additions is n; but the number of multiplications can be dramatically different for large values of n.

Approxiamting function SF(x)

$$SF(x) = \frac{1}{x}\int_0^x \frac{\sin t}{t}\,dt, \qquad x \neq 0$$

Approxiamting function SF(x)

$$SF(x) = \frac{1}{x}\int_0^x \frac{\sin t}{t}\,dt, \qquad x \neq 0$$

We use Taylor polynomials to approximate this function, to obtain a way to compute it with accuracy and simplicity.

Approxiamting function SF(x)

$$SF(x) = \frac{1}{x}\int_0^x \frac{\sin t}{t}\,dt, \qquad x \neq 0$$

We use Taylor polynomials to approximate this function, to obtain a way to compute it with accuracy and simplicity.

Approxiamting function SF(x)

As an example, begin with the degree 3 Taylor approximation to $\sin t$, expanded about $t = 0$:

$$\sin t = t - \frac{1}{6}t^3 + \frac{1}{120}t^5 \cos \xi, \qquad 0 \leq \xi \leq t$$

Approxiamting function SF(x)

As an example, begin with the degree 3 Taylor approximation to $\sin t$, expanded about $t = 0$:

$$\sin t = t - \frac{1}{6}t^3 + \frac{1}{120}t^5 \cos \xi, \qquad 0 \le \xi \le t$$

$$\frac{\sin t}{t} = 1 - \frac{1}{6}t^2 + \frac{1}{120}t^4 \cos \xi$$

Approxiamting function SF(x)

As an example, begin with the degree 3 Taylor approximation to $\sin t$, expanded about $t = 0$:

$$\sin t = t - \frac{1}{6}t^3 + \frac{1}{120}t^5 \cos\xi, \qquad 0 \le \xi \le t$$

$$\begin{aligned} \frac{\sin t}{t} &= 1 - \frac{1}{6}t^2 + \frac{1}{120}t^4 \cos\xi \\ \int_0^x \frac{\sin t}{t}\,dt &= \int_0^x \left(1 - \frac{1}{6}t^2 + \frac{1}{120}t^4 \cos\xi\right) dt \end{aligned}$$

Approxiamting function SF(x)

As an example, begin with the degree 3 Taylor approximation to $\sin t$, expanded about $t = 0$:

$$\sin t = t - \frac{1}{6}t^3 + \frac{1}{120}t^5 \cos\xi, \qquad 0 \le \xi \le t$$

$$\begin{aligned}
\frac{\sin t}{t} &= 1 - \frac{1}{6}t^2 + \frac{1}{120}t^4 \cos\xi \\
\int_0^x \frac{\sin t}{t}\,dt &= \int_0^x \left(1 - \frac{1}{6}t^2 + \frac{1}{120}t^4 \cos\xi\right) dt \\
\int_0^x \frac{\sin t}{t}\,dt &= \int_0^x 1\,dt - \frac{1}{6}\int_0^x t^2\,dt + \frac{1}{120}\int_0^x t^4 \cos\xi\,dt
\end{aligned}$$

Approxiamting function SF(x)

As an example, begin with the degree 3 Taylor approximation to $\sin t$, expanded about $t = 0$:

$$\sin t = t - \frac{1}{6}t^3 + \frac{1}{120}t^5 \cos\xi, \qquad 0 \le \xi \le t$$

$$\begin{aligned}
\frac{\sin t}{t} &= 1 - \frac{1}{6}t^2 + \frac{1}{120}t^4 \cos\xi \\
\int_0^x \frac{\sin t}{t}\,dt &= \int_0^x \left(1 - \frac{1}{6}t^2 + \frac{1}{120}t^4 \cos\xi\right) dt \\
\int_0^x \frac{\sin t}{t}\,dt &= \int_0^x 1\,dt - \frac{1}{6}\int_0^x t^2\,dt + \frac{1}{120}\int_0^x t^4 \cos\xi\,dt \\
\int_0^x \frac{\sin t}{t}\,dt &= x - \frac{1}{18}x^3 + \frac{1}{120}\int_0^x t^4 \cos\xi\,dt
\end{aligned}$$

Approxiamting function SF(x)

$$SF(x) = \frac{1}{x}\int_0^x \frac{\sin t}{t}\,dt$$

Approxiamting function SF(x)

$$SF(x) = \frac{1}{x}\int_0^x \frac{\sin t}{t}dt = 1 - \frac{1}{18}x^2 + R_2(x),$$

Approxiamting function SF(x)

$$SF(x) = \frac{1}{x}\int_0^x \frac{\sin t}{t}dt = 1 - \frac{1}{18}x^2 + R_2(x),$$

where the remainder (error term) is given by

$$R_2(x) = \frac{1}{120}\frac{1}{x}\int_0^x t^4 \cos\xi dt$$

Approxiamting function SF(x)

$$SF(x) = \frac{1}{x}\int_0^x \frac{\sin t}{t}dt = 1 - \frac{1}{18}x^2 + R_2(x),$$

where the remainder (error term) is given by

$$R_2(x) = \frac{1}{120}\frac{1}{x}\int_0^x t^4 \cos\xi dt$$

How large is the error in approximation

$$SF(x) \approx 1 - \frac{1}{18}x^2$$

on interval $x \in [-1, 1]$?

Approxiamting function SF(x)

For $x > 0$ we have $0 \leq \cos \xi \leq 1$,

Approxiamting function SF(x)

For $x > 0$ we have $0 \leq \cos\xi \leq 1$, therefore

$$0 \leq R_2(x) = \frac{1}{120}\frac{1}{x}\int_0^x t^4 \cos\xi dt \leq \frac{1}{120}\frac{1}{x}\int_0^x t^4 dt = \frac{1}{600}x^4$$

Approxiamting function SF(x)

For $x > 0$ we have $0 \leq \cos\xi \leq 1$, therefore

$$0 \leq R_2(x) = \frac{1}{120}\frac{1}{x}\int_0^x t^4 \cos\xi dt \leq \frac{1}{120}\frac{1}{x}\int_0^x t^4 dt = \frac{1}{600}x^4$$

and the same result can be shown for $x < 0$.

Approxiamting function SF(x)

For $x > 0$ we have $0 \leq \cos\xi \leq 1$, therefore

$$0 \leq R_2(x) = \frac{1}{120}\frac{1}{x}\int_0^x t^4 \cos\xi dt \leq \frac{1}{120}\frac{1}{x}\int_0^x t^4 dt = \frac{1}{600}x^4$$

and the same result can be shown for $x < 0$.

Then for $|x| \leq 1$, we have

$$0 \leq R_2(x) \leq \frac{1}{600}$$

Approxiamting function SF(x)

For $x > 0$ we have $0 \leq \cos\xi \leq 1$, therefore

$$0 \leq R_2(x) = \frac{1}{120}\frac{1}{x}\int_0^x t^4 \cos\xi dt \leq \frac{1}{120}\frac{1}{x}\int_0^x t^4 dt = \frac{1}{600}x^4$$

and the same result can be shown for $x < 0$.

Then for $|x| \leq 1$, we have

$$0 \leq R_2(x) \leq \frac{1}{600}$$

To obtain a more accurate approximation, we can proceed exactly as above, but simply use a higher degree approximation to $\sin t$.

Numerical Analysis

Prof.dr. hab. Bostan Viorel

Decimal floating point numbers

Floating point notation is similar to what is called scientific notation.

Decimal floating point numbers

Floating point notation is similar to what is called scientific notation.

For a nonzero number x, we can write it in the form

$$x = \sigma \cdot \overline{x} \cdot 10^e,$$

Decimal floating point numbers

Floating point notation is similar to what is called scientific notation.

For a nonzero number x, we can write it in the form

$$x = \sigma \cdot \overline{x} \cdot 10^e,$$

where $\sigma = \pm 1$, e is an integer, and $1_{10} \leq \overline{x} < 10_{10}$.

Decimal floating point numbers

Floating point notation is similar to what is called scientific notation.

For a nonzero number x, we can write it in the form

$$x = \sigma \cdot \overline{x} \cdot 10^e,$$

where $\sigma = \pm 1$, e is an integer, and $1_{10} \leq \overline{x} < 10_{10}$.

Number σ is called **sign**, e is **exponent**, and $\overline{x}$ is called **mantissa** or **significand**.

Decimal floating point numbers

Floating point notation is similar to what is called scientific notation.

For a nonzero number x, we can write it in the form

$$x = \sigma \cdot \overline{x} \cdot 10^e,$$

where $\sigma = \pm 1$, e is an integer, and $1_{10} \leq \overline{x} < 10_{10}$.

Number σ is called **sign**, e is **exponent**, and $\overline{x}$ is called **mantissa** or **significand**.

For example

$$345.78 = (+1) \cdot 3.4578 \cdot 10^2,$$

Decimal floating point numbers

Floating point notation is similar to what is called scientific notation.

For a nonzero number x, we can write it in the form

$$x = \sigma \cdot \overline{x} \cdot 10^e,$$

where $\sigma = \pm 1$, e is an integer, and $1_{10} \leq \overline{x} < 10_{10}$.

Number σ is called **sign**, e is **exponent**, and $\overline{x}$ is called **mantissa** or **significand**.

For example

$$345.78 = (+1) \cdot 3.4578 \cdot 10^2,$$

On a decimal computer or calculator, we store x by instead storing σ, e and $\overline{x}$.

Decimal floating point numbers

Computers have a finite space for storage.

Decimal floating point numbers

Computers have a finite space for storage.

We must restrict the number of digits in $\overline{x}$ and the size of the exponent e.

Decimal floating point numbers

Computers have a finite space for storage.

We must restrict the number of digits in $\overline{x}$ and the size of the exponent e.

The number of digits in $\overline{x}$ is called **precision**.

Decimal floating point numbers

Computers have a finite space for storage.

We must restrict the number of digits in $\overline{x}$ and the size of the exponent e.

The number of digits in $\overline{x}$ is called **precision**.

For example on calculator HP-15, the number of digits in mantissa is 10 and

$$-99 \leq e \leq 99$$

Binary floating point numbers

Binary floating point numbers

Write

$$x = \sigma \cdot \overline{x} \cdot 2^e$$

with $1_2 \leq \overline{x} < (10)_2 = (2)_{10}$ and e an integer.

Binary floating point numbers

Write

$$x = \sigma \cdot \overline{x} \cdot 2^e$$

with $1_2 \leq \overline{x} < (10)_2 = (2)_{10}$ and e an integer.

For example,

$$(10011.01101)_2 = (+1) \cdot (1.001101101)_2 \cdot 2^{(100)_2}$$

Binary floating point numbers

Write

$$x = \sigma \cdot \overline{x} \cdot 2^e$$

with $1_2 \leq \overline{x} < (10)_2 = (2)_{10}$ and e an integer.

For example,

$$(10011.01101)_2 = (+1) \cdot (1.001101101)_2 \cdot 2^{(100)_2}$$

Or another example for $(0.1)_{10} = (0.000110011001100\ldots)_2$

Binary floating point numbers

Write

$$x = \sigma \cdot \overline{x} \cdot 2^e$$

with $1_2 \leq \overline{x} < (10)_2 = (2)_{10}$ and e an integer.

For example,

$$(10011.01101)_2 = (+1) \cdot (1.001101101)_2 \cdot 2^{(100)_2}$$

Or another example for $(0.1)_{10} = (0.000110011001100\ldots)_2$

$$(0.000110011001100\ldots)_2 = (+1) \cdot (1.10011001100\ldots)_2 \cdot 2^{-(100)_2}$$

Binary floating point numbers

Write

$$x = \sigma \cdot \overline{x} \cdot 2^e$$

with $1_2 \leq \overline{x} < (10)_2 = (2)_{10}$ and e an integer.

For example,

$$(10011.01101)_2 = (+1) \cdot (1.001101101)_2 \cdot 2^{(100)_2}$$

Or another example for $(0.1)_{10} = (0.000110011001100\ldots)_2$

$$(0.000110011001100\ldots)_2 = (+1) \cdot (1.10011001100\ldots)_2 \cdot 2^{-(100)_2}$$

Notice, that the first digit in mantissa is always 1.

Floating point numbers

When a number x outside a computer or calculator is converted into a machine number, we denote it by $fl(x)$.

Floating point numbers

When a number x outside a computer or calculator is converted into a machine number, we denote it by $fl(x)$.

On a calculator with precission 10,

$$fl\left(\frac{1}{3}\right) = fl(0.3333333\ldots)$$

Floating point numbers

When a number x outside a computer or calculator is converted into a machine number, we denote it by $fl(x)$.

On a calculator with precission 10,

$$fl\left(\frac{1}{3}\right) = fl(0.3333333\ldots) = (+1)\cdot(3.333333333)_{10}\cdot 10^{-1}$$

Floating point numbers

When a number x outside a computer or calculator is converted into a machine number, we denote it by $fl(x)$.

On a calculator with precission 10,

$$fl\left(\frac{1}{3}\right) = fl(0.3333333\ldots) = (+1)\cdot(3.333333333)_{10}\cdot 10^{-1}$$

The decimal fraction of finite length will not fit in the registers of the calculator, but the latter 10−digit number will fit.

Floating point numbers

When a number x outside a computer or calculator is converted into a machine number, we denote it by $fl(x)$.

On a calculator with precission 10,

$$fl\left(\frac{1}{3}\right) = fl(0.3333333\ldots) = (+1)\cdot(3.333333333)_{10}\cdot 10^{-1}$$

The decimal fraction of finite length will not fit in the registers of the calculator, but the latter 10−digit number will fit.

Some calculators actually carry more digits internally than they allow to be displayed.

Floating point numbers

When a number x outside a computer or calculator is converted into a machine number, we denote it by $fl(x)$.

On a calculator with precission 10,

$$fl\left(\frac{1}{3}\right) = fl(0.3333333\ldots) = (+1)\cdot(3.333333333)_{10}\cdot 10^{-1}$$

The decimal fraction of finite length will not fit in the registers of the calculator, but the latter 10−digit number will fit.

Some calculators actually carry more digits internally than they allow to be displayed.

On a binary computer, we use a similar notation.

Floating point numbers

When a number x outside a computer or calculator is converted into a machine number, we denote it by $fl(x)$.

On a calculator with precission 10,

$$fl\left(\frac{1}{3}\right) = fl(0.3333333\ldots) = (+1)\cdot(3.333333333)_{10}\cdot 10^{-1}$$

The decimal fraction of finite length will not fit in the registers of the calculator, but the latter 10−digit number will fit.

Some calculators actually carry more digits internally than they allow to be displayed.

On a binary computer, we use a similar notation.

We will concentrate on a particular form of computer floating point number, that is called the **IEEE floating point standard**.

Binary floating point numbers

Example 1 Consider a binary floating point representation with precision 3 and $e_{min} = -2 \leq e \leq 2 = e_{max}$

Binary floating point numbers

Example 1 Consider a binary floating point representation with precision 3 and $e_{min} = -2 \leq e \leq 2 = e_{max}$

All the numbers admitted by this representation are presented in the table:

Binary floating point numbers

Example 1 Consider a binary floating point representation with precision 3 and $e_{min} = -2 \leq e \leq 2 = e_{max}$

All the numbers admitted by this representation are presented in the table:

		e				
		-2	-1	0	1	2
$\overline{x}$	$(1.00)_2$	$(0.25)_{10}$	$(0.5)_{10}$	$(1)_{10}$	$(2)_{10}$	$(4)_{10}$
	$(1.01)_2$	$(0.3125)_{10}$	$(0.625)_{10}$	$(1.25)_{10}$	$(2.5)_{10}$	$(5)_{10}$
	$(1.10)_2$	$(0.375)_{10}$	$(0.75)_{10}$	$(1.5)_{10}$	$(3)_{10}$	$(6)_{10}$
	$(1.11)_2$	$(0.4375)_{10}$	$(0.875)_{10}$	$(1.75)_{10}$	$(3.5)_{10}$	$(7)_{10}$

$$x = \sigma \cdot \overline{x} \cdot 2^e$$

Binary floating point numbers

Binary floating point numbers

This representation can be extended to include smaller numbers called **denormalized** numbers.

Binary floating point numbers

This representation can be extended to include smaller numbers called **denormalized** numbers.

These numbers are obtained if $e = e_{min}$ and the first digit of the significand is 0.

Binary floating point numbers

Example 2 Previous example plus denormalized numbers

Binary floating point numbers

Example 2 Previous example plus denormalized numbers

$$(0.01)_2 \cdot 2^{-1} = \frac{1}{16} = (0.0625)_{10}$$
$$(0.10)_2 \cdot 2^{-1} = \frac{2}{16} = (0.125)_{10}$$
$$(0.11)_2 \cdot 2^{-1} = \frac{3}{16} = (0.1875)_{10}$$

Binary floating point numbers

Example 2 Previous example plus denormalized numbers

$$(0.01)_2 \cdot 2^{-1} = \frac{1}{16} = (0.0625)_{10}$$
$$(0.10)_2 \cdot 2^{-1} = \frac{2}{16} = (0.125)_{10}$$
$$(0.11)_2 \cdot 2^{-1} = \frac{3}{16} = (0.1875)_{10}$$

IEEE single precision floating point representation

In IEEE (Institute of Electrical and Electronics Engineers) single precision standard 32 bits are used to store numbers.

IEEE single precision floating point representation

In IEEE (Institute of Electrical and Electronics Engineers) single precision standard 32 bits are used to store numbers.

IEEE single precision has in mantissa 24 binary digits,

IEEE single precision floating point representation

In IEEE (Institute of Electrical and Electronics Engineers) single precision standard 32 bits are used to store numbers.

IEEE single precision has in mantissa 24 binary digits, exponent e has values between -126 and 127,

IEEE single precision floating point representation

In IEEE (Institute of Electrical and Electronics Engineers) single precision standard 32 bits are used to store numbers.

IEEE single precision has in mantissa 24 binary digits,
exponent e has values between -126 and 127,
or in binary $-(1111110)_2 \leq e \leq (1111111)_2$.

IEEE single precision floating point representation

In IEEE (Institute of Electrical and Electronics Engineers) single precision standard 32 bits are used to store numbers.

IEEE single precision has in mantissa 24 binary digits,
exponent e has values between -126 and 127,
or in binary $-(1111110)_2 \leq e \leq (1111111)_2$.

Thus number x has the representation

$$x = \sigma \cdot 1.a_1 a_2 \ldots a_{23} \cdot 2^e.$$

IEEE single precision floating point representation

In IEEE (Institute of Electrical and Electronics Engineers) single precision standard 32 bits are used to store numbers.

IEEE single precision has in mantissa 24 binary digits,
exponent e has values between -126 and 127,
or in binary $-(1111110)_2 \leq e \leq (1111111)_2$.

Thus number x has the representation

$$x = \sigma \cdot 1.a_1 a_2 \ldots a_{23} \cdot 2^e.$$

Basically, we store σ as a single bit,

IEEE single precision floating point representation

In IEEE (Institute of Electrical and Electronics Engineers) single precision standard 32 bits are used to store numbers.

IEEE single precision has in mantissa 24 binary digits,
exponent e has values between -126 and 127,
or in binary $-(1111110)_2 \leq e \leq (1111111)_2$.

Thus number x has the representation

$$x = \sigma \cdot 1.a_1 a_2 \ldots a_{23} \cdot 2^e.$$

Basically, we store σ as a single bit, the significand $\overline{x}$ as 24 bits (only 23 need be stored),

IEEE single precision floating point representation

In IEEE (Institute of Electrical and Electronics Engineers) single precision standard 32 bits are used to store numbers.

IEEE single precision has in mantissa 24 binary digits,
exponent e has values between -126 and 127,
or in binary $-(1111110)_2 \leq e \leq (1111111)_2$.

Thus number x has the representation

$$x = \sigma \cdot 1.a_1 a_2 \ldots a_{23} \cdot 2^e.$$

Basically, we store σ as a single bit, the significand $\overline{x}$ as 24 bits (only 23 need be stored), and the exponent fills out 8 bits, including both negative and positive integers.

IEEE single precision floating point representation

In order to avoid the sign for exponent, denote

$$E = e + 127$$

IEEE single precision floating point representation

In order to avoid the sign for exponent, denote

$$E = e + 127$$

Obviously, $1 \leq E \leq 254$ with two additional values 0 and 255.

IEEE single precision floating point representation

In order to avoid the sign for exponent, denote

$$E = e + 127$$

Obviously, $1 \leq E \leq 254$ with two additional values 0 and 255.

σ	E			$\overline{x}$		
b_1	b_2	$\ldots$	b_9	b_{10}	$\ldots$	b_{32}

IEEE single precision floating point representation

In order to avoid the sign for exponent, denote

$$E = e + 127$$

Obviously, $1 \leq E \leq 254$ with two additional values 0 and 255.

σ	E			$\overline{x}$		
b_1	b_2	$\ldots$	b_9	b_{10}	$\ldots$	b_{32}

Number $x = 0$ is stored in the following way: $E = 0$, $\sigma = 0$ and $b_{10} b_{11} \ldots b_{32} = (00 \ldots 0)_2$.

IEEE single precision floating point representation

$E = (b_2 \dots b_9)_2$	e	x
$(00000000)_2 = (0)_{10}$	$-(127)_{10}$	$\pm(0.b_{10} \dots b_{32})_2 \cdot 2^{-126}$
$(00000001)_2 = (1)_{10}$	$-(126)_{10}$	$\pm(1.b_{10} \dots b_{32})_2 \cdot 2^{-126}$
$(00000010)_2 = (2)_{10}$	$-(125)_{10}$	$\pm(1.b_{10} \dots b_{32})_2 \cdot 2^{-125}$
$\vdots$	$\vdots$	$\vdots$
$(01111111)_2 = (127)_{10}$	$(0)_{10}$	$\pm(1.b_{10} \dots b_{32})_2 \cdot 2^{0}$
$(10000000)_2 = (128)_{10}$	$(1)_{10}$	$\pm(1.b_{10} \dots b_{32})_2 \cdot 2^{1}$
$\vdots$	$\vdots$	$\vdots$
$(11111101)_2 = (253)_{10}$	$(126)_{10}$	$\pm(1.b_{10} \dots b_{32})_2 \cdot 2^{126}$
$(11111110)_2 = (254)_{10}$	$(127)_{10}$	$\pm(1.b_{10} \dots b_{32})_2 \cdot 2^{127}$
$(11111111)_2 = (255)_{10}$	$(128)_{10}$	$\pm\infty, \quad \textit{dacă } b_i = 0$ $NaN, \quad \textit{otherwise}$

IEEE double precision floating point representation

$$x = \sigma \cdot 1.a_1 a_2 \ldots a_{52} \cdot 2^e.$$

with $E = e + 1023$

σ	E			$\overline{x}$		
b_1	b_2	$\ldots$	b_{12}	b_{13}	$\ldots$	b_{64}

IEEE double precision floating point representation

$E = (b_2 \dots b_{12})_2$	e	x
$(00000000000)_2 = (0)_{10}$	$-(1023)_{10}$	$\pm(0.b_{13} \dots b_{64})2^{-1022}$
$(00000000001)_2 = (1)_{10}$	$-(1022)_{10}$	$\pm(1.b_{13} \dots b_{64})2^{-1022}$
$(00000000010)_2 = (2)_{10}$	$-(1021)_{10}$	$\pm(1.b_{13} \dots b_{64})2^{-1021}$
$\vdots$	$\vdots$	$\vdots$
$(01111111111)_2 = (1023)_{10}$	$(0)_{10}$	$\pm(1.b_{13} \dots b_{64})2^{0}$
$(10000000000)_2 = (1024)_{10}$	$(1)_{10}$	$\pm(1.b_{13} \dots b_{64})2^{1}$
$\vdots$	$\vdots$	$\vdots$
$(11111111101)_2 = (2045)_{10}$	$(1022)_{10}$	$\pm(1.b_{13} \dots b_{64})2^{1022}$
$(11111111110)_2 = (2046)_{10}$	$(1023)_{10}$	$\pm(1.b_{13} \dots b_{64})_2{}^{1023}$
$(11111111111)_2 = (2047)_{10}$	$(1024)_{10}$	$\pm\infty, \quad b_i = 0$ $NaN, \quad otherwise$

Caracteristics of floating point representation

What is the connection of the 24 bits in the significand $\overline{x}$ to the number of decimal digits in the storage of a number x into floating point form?

Caracteristics of floating point representation

What is the connection of the 24 bits in the significand $\overline{x}$ to the number of decimal digits in the storage of a number x into floating point form?

One way of answering this is to find the integer M for which

1. $0 < x \leq M$ and x an integer implies $fl(x) = x$

Caracteristics of floating point representation

What is the connection of the 24 bits in the significand $\overline{x}$ to the number of decimal digits in the storage of a number x into floating point form?

One way of answering this is to find the integer M for which

1. $0 < x \leq M$ and x an integer implies $fl(x) = x$
2. $fl(M+1) \neq M+1$

Caracteristics of floating point representation

What is the connection of the 24 bits in the significand $\overline{x}$ to the number of decimal digits in the storage of a number x into floating point form?

One way of answering this is to find the integer M for which

1. $0 < x \leq M$ and x an integer implies $fl(x) = x$
2. $fl(M+1) \neq M+1$

Caracteristics of floating point representation

What is the connection of the 24 bits in the significand $\overline{x}$ to the number of decimal digits in the storage of a number x into floating point form?

One way of answering this is to find the integer M for which

1. $0 < x \leq M$ and x an integer implies $fl(x) = x$
2. $fl(M+1) \neq M+1$

This integer M is at least as big as

$$(\underbrace{111\ldots11}_{\text{23 ones}})_2$$

Caracteristics of floating point representation

What is the connection of the 24 bits in the significand $\overline{x}$ to the number of decimal digits in the storage of a number x into floating point form?

One way of answering this is to find the integer M for which

1. $0 < x \leq M$ and x an integer implies $fl(x) = x$
2. $fl(M+1) \neq M+1$

This integer M is at least as big as

$$(\underbrace{111\ldots11}_{\text{23 ones}})_2 \quad = \quad (1.11\ldots1)_2 \cdot 2^{23}$$

Caracteristics of floating point representation

What is the connection of the 24 bits in the significand $\overline{x}$ to the number of decimal digits in the storage of a number x into floating point form?

One way of answering this is to find the integer M for which

1. $0 < x \leq M$ and x an integer implies $fl(x) = x$
2. $fl(M+1) \neq M+1$

This integer M is at least as big as

$$\begin{aligned}(\underbrace{111\ldots11}_{\text{23 ones}})_2 &= (1.11\ldots1)_2 \cdot 2^{23} \\ &= 2^{23} + 2^{22} + \ldots + 2^0\end{aligned}$$

Caracteristics of floating point representation

What is the connection of the 24 bits in the significand $\overline{x}$ to the number of decimal digits in the storage of a number x into floating point form?

One way of answering this is to find the integer M for which

1. $0 < x \leq M$ and x an integer implies $fl(x) = x$
2. $fl(M+1) \neq M+1$

This integer M is at least as big as

$$\begin{aligned}(\underbrace{111\ldots11}_{\text{23 ones}})_2 &= (1.11\ldots1)_2 \cdot 2^{23} \\ &= 2^{23} + 2^{22} + \ldots + 2^0 \\ &= 2^{24} - 1\end{aligned}$$

Caracteristics of floating point representation

Also, there is one more number to be stored exactly:

Caracteristics of floating point representation

Also, there is one more number to be stored exactly:

$$2^{24} = (1.00\ldots00)_2 \cdot 2^{24}$$

Caracteristics of floating point representation

Also, there is one more number to be stored exactly:

$$2^{24} = (1.00\ldots00)_2 \cdot 2^{24}$$

Next integer $2^{24} + 1$ cannot be stored exactly since its significand will contain $24 + 1$ binary digits:

Caracteristics of floating point representation

Also, there is one more number to be stored exactly:

$$2^{24} = (1.00\ldots00)_2 \cdot 2^{24}$$

Next integer $2^{24} + 1$ cannot be stored exactly since its significand will contain $24 + 1$ binary digits:

$$2^{24} + 1 = (1.\underbrace{00\ldots0}_{\text{23 zeros}}1)_2 \cdot 2^{24}$$

Caracteristics of floating point representation

Also, there is one more number to be stored exactly:

$$2^{24} = (1.00\ldots00)_2 \cdot 2^{24}$$

Next integer $2^{24} + 1$ cannot be stored exactly since its significand will contain $24 + 1$ binary digits:

$$2^{24} + 1 = (1.\underbrace{00\ldots0}_{\text{23 zeros}}1)_2 \cdot 2^{24}$$

Therefore, for single precision $M = 2^{24}$.

Caracteristics of floating point representation

Also, there is one more number to be stored exactly:

$$2^{24} = (1.00\ldots00)_2 \cdot 2^{24}$$

Next integer $2^{24} + 1$ cannot be stored exactly since its significand will contain $24 + 1$ binary digits:

$$2^{24} + 1 = (1.\underbrace{00\ldots0}_{23 \text{ zeros}}1)_2 \cdot 2^{24}$$

Therefore, for single precision $M = 2^{24}$. Any integer less or equal to M will be stored exactly.

Caracteristics of floating point representation

Also, there is one more number to be stored exactly:

$$2^{24} = (1.00\ldots00)_2 \cdot 2^{24}$$

Next integer $2^{24} + 1$ cannot be stored exactly since its significand will contain $24 + 1$ binary digits:

$$2^{24} + 1 = (1.\underbrace{00\ldots0}_{23 \text{ zeros}}1)_2 \cdot 2^{24}$$

Therefore, for single precision $M = 2^{24}$. Any integer less or equal to M will be stored exactly.

So $M = 2^{24} = 16777216$

Caracteristics of floating point representation

Also, there is one more number to be stored exactly:

$$2^{24} = (1.00\ldots00)_2 \cdot 2^{24}$$

Next integer $2^{24} + 1$ cannot be stored exactly since its significand will contain $24 + 1$ binary digits:

$$2^{24} + 1 = (1.\underbrace{00\ldots0}_{23 \text{ zeros}}1)_2 \cdot 2^{24}$$

Therefore, for single precision $M = 2^{24}$. Any integer less or equal to M will be stored exactly.

So $M = 2^{24} = 16777216$
For double precision $M = 2^{53} \approx 9.0 \cdot 10^{15}$

Machine epsilon

Let y be the smallest number representable in the machine arithmetic that is greater than 1 in the machine.

Machine epsilon

Let y be the smallest number representable in the machine arithmetic that is greater than 1 in the machine.

The machine epsilon is $\eta = y - 1$.

Machine epsilon

Let y be the smallest number representable in the machine arithmetic that is greater than 1 in the machine.

The machine epsilon is $\eta = y - 1$.

It is a widely used to measure the accuracy possible in representing numbers in the machine.

Machine epsilon

Let y be the smallest number representable in the machine arithmetic that is greater than 1 in the machine.

The machine epsilon is $\eta = y - 1$.

It is a widely used to measure the accuracy possible in representing numbers in the machine.

The number 1 has the simple floating point representation

$$1 = (1.00\ldots0)_2 \cdot 2^0$$

Machine epsilon

Let y be the smallest number representable in the machine arithmetic that is greater than 1 in the machine.

The machine epsilon is $\eta = y - 1$.

It is a widely used to measure the accuracy possible in representing numbers in the machine.

The number 1 has the simple floating point representation

$$1 = (1.00\ldots0)_2 \cdot 2^0$$

What is the smallest number that is greater than 1?

Machine epsilon

Let y be the smallest number representable in the machine arithmetic that is greater than 1 in the machine.

The machine epsilon is $\eta = y - 1$.

It is a widely used to measure the accuracy possible in representing numbers in the machine.

The number 1 has the simple floating point representation

$$1 = (1.00\ldots0)_2 \cdot 2^0$$

What is the smallest number that is greater than 1? It is

$$1 + 2^{-23} = (1.0\ldots01)_2 \cdot 2^0 > 1$$

Machine epsilon

Let y be the smallest number representable in the machine arithmetic that is greater than 1 in the machine.

The machine epsilon is $\eta = y - 1$.

It is a widely used to measure the accuracy possible in representing numbers in the machine.

The number 1 has the simple floating point representation

$$1 = (1.00\ldots0)_2 \cdot 2^0$$

What is the smallest number that is greater than 1? It is

$$1 + 2^{-23} = (1.0\ldots01)_2 \cdot 2^0 > 1$$

and the machine epsilon in IEEE single precision floating point format is $\eta = 2^{-23}$

Machine epsilon

Let y be the smallest number representable in the machine arithmetic that is greater than 1 in the machine.

The machine epsilon is $\eta = y - 1$.

It is a widely used to measure the accuracy possible in representing numbers in the machine.

The number 1 has the simple floating point representation

$$1 = (1.00\ldots0)_2 \cdot 2^0$$

What is the smallest number that is greater than 1? It is

$$1 + 2^{-23} = (1.0\ldots01)_2 \cdot 2^0 > 1$$

and the machine epsilon in IEEE single precision floating point format is $\eta = 2^{-23} \approx 1.19 \cdot 10^{-7}$

The unit round

Consider the smallest number $\delta > 0$ that is representable in the machine

The unit round

Consider the smallest number $\delta > 0$ that is representable in the machine and for which

$$1 + \delta > 1$$

in the arithmetic of the machine.

The unit round

Consider the smallest number $\delta > 0$ that is representable in the machine and for which

$$1 + \delta > 1$$

in the arithmetic of the machine.

For any number $0 < \alpha < \delta$ the result of $1 + \alpha$ is exactly 1 in the machines arithmetic.

The unit round

Consider the smallest number $\delta > 0$ that is representable in the machine and for which

$$1 + \delta > 1$$

in the arithmetic of the machine.

For any number $0 < \alpha < \delta$ the result of $1 + \alpha$ is exactly 1 in the machines arithmetic.

Thus α drops off the end of the floating point representation in the machine.

The unit round

Consider the smallest number $\delta > 0$ that is representable in the machine and for which

$$1 + \delta > 1$$

in the arithmetic of the machine.

For any number $0 < \alpha < \delta$ the result of $1 + \alpha$ is exactly 1 in the machines arithmetic.

Thus α drops off the end of the floating point representation in the machine.

The size of δ is another way of describing the accuracy attainable in the floating point representation of the machine.

The unit round

Consider the smallest number $\delta > 0$ that is representable in the machine and for which

$$1 + \delta > 1$$

in the arithmetic of the machine.

For any number $0 < \alpha < \delta$ the result of $1 + \alpha$ is exactly 1 in the machines arithmetic.

Thus α drops off the end of the floating point representation in the machine.

The size of δ is another way of describing the accuracy attainable in the floating point representation of the machine.

The machine epsilon has been replacing it in recent years.

Chopping and rounding in decimal

Chopping and rounding in decimal

We write a computer floating point number z as

$$z = \sigma \cdot \overline{z} \cdot 10^e \equiv \sigma \cdot (a_1.a_2 \ldots a_n)_{10} \cdot 10^e$$

with $a_1 \neq 0$, so that there are n decimal digits in the significand.

Chopping and rounding in decimal

We write a computer floating point number z as

$$z = \sigma \cdot \overline{z} \cdot 10^e \equiv \sigma \cdot (a_1.a_2 \ldots a_n)_{10} \cdot 10^e$$

with $a_1 \neq 0$, so that there are n decimal digits in the significand.

Given a general number x

$$x = \sigma \cdot \overline{x} \cdot 10^e \equiv \sigma \cdot (a_1.a_2 \ldots a_n a_{n+1} \ldots)_{10} \cdot 10^e, \quad a_1 \neq 0.$$

Chopping and rounding in decimal

We write a computer floating point number z as

$$z = \sigma \cdot \overline{z} \cdot 10^e \equiv \sigma \cdot (a_1.a_2 \ldots a_n)_{10} \cdot 10^e$$

with $a_1 \neq 0$, so that there are n decimal digits in the significand.

Given a general number x

$$x = \sigma \cdot \overline{x} \cdot 10^e \equiv \sigma \cdot (a_1.a_2 \ldots a_n a_{n+1} \ldots)_{10} \cdot 10^e, \quad a_1 \neq 0.$$

we must shorten it to fit within the computer.

Chopping and rounding in decimal

We write a computer floating point number z as

$$z = \sigma \cdot \overline{z} \cdot 10^e \equiv \sigma \cdot (a_1.a_2 \ldots a_n)_{10} \cdot 10^e$$

with $a_1 \neq 0$, so that there are n decimal digits in the significand.

Given a general number x

$$x = \sigma \cdot \overline{x} \cdot 10^e \equiv \sigma \cdot (a_1.a_2 \ldots a_n a_{n+1} \ldots)_{10} \cdot 10^e, \quad a_1 \neq 0.$$

we must shorten it to fit within the computer.

This is done by either chopping or rounding.

Chopping and rounding in decimal

We write a computer floating point number z as

$$z = \sigma \cdot \overline{z} \cdot 10^e \equiv \sigma \cdot (a_1.a_2 \ldots a_n)_{10} \cdot 10^e$$

with $a_1 \neq 0$, so that there are n decimal digits in the significand.

Given a general number x

$$x = \sigma \cdot \overline{x} \cdot 10^e \equiv \sigma \cdot (a_1.a_2 \ldots a_n a_{n+1} \ldots)_{10} \cdot 10^e, \quad a_1 \neq 0.$$

we must shorten it to fit within the computer.

This is done by either chopping or rounding.

The floating point chopped version of x is given by

$$fl(x) = \sigma \cdot \overline{x} \cdot 10^e \equiv \sigma \cdot (a_1.a_2 \ldots a_n)_{10} \cdot 10^e,$$

where we assume that e fits within the bounds required by the computer or calculator.

Chopping and rounding in decimal

For the rounded version, we must decide whether to round up or round down.

Chopping and rounding in decimal

For the rounded version, we must decide whether to round up or round down.

A simplified formula is

$$fl(x) = \begin{cases} \sigma \cdot (a_1.a_2 \ldots a_n)_{10} \cdot 10^e, & \textit{if} \quad a_{n+1} < 5 \\ \sigma \cdot (a_1.a_2 \ldots a_n)_{10} \cdot 10^e + (0.00 \ldots 01)_{10}, & \textit{if} \quad a_{n+1} \geq 5 \end{cases}$$

Chopping and rounding in decimal

For the rounded version, we must decide whether to round up or round down.

A simplified formula is

$$fl(x) = \begin{cases} \sigma \cdot (a_1.a_2 \ldots a_n)_{10} \cdot 10^e, & \textit{if} \quad a_{n+1} < 5 \\ \sigma \cdot (a_1.a_2 \ldots a_n)_{10} \cdot 10^e + (0.00 \ldots 01)_{10}, & \textit{if} \quad a_{n+1} \geq 5 \end{cases}$$

The term $(0.00 \ldots 01)_{10}$ denotes 10^{-n+1}, giving the ordinary sense of rounding with which you are familiar.

Chopping and rounding in binary

Chopping and rounding in binary

Let

$$x = \sigma \cdot \overline{x} \cdot 2^e \equiv \sigma \cdot (1.a_2 \ldots a_n a_{n+1} \ldots)_2 \cdot 2^e,$$

with all a_i equal to 0 or 1.

Chopping and rounding in binary

Let

$$x = \sigma \cdot \overline{x} \cdot 2^e \equiv \sigma \cdot (1.a_2 \ldots a_n a_{n+1} \ldots)_2 \cdot 2^e,$$

with all a_i equal to 0 or 1.

Then for a **chopped** floating point representation, we have

$$fl(x) = \sigma \cdot \overline{x} \cdot 2^e \equiv \sigma \cdot (1.a_2 \ldots a_n)_2 \cdot 2^e.$$

Chopping and rounding in binary

Let

$$x = \sigma \cdot \overline{x} \cdot 2^e \equiv \sigma \cdot (1.a_2 \ldots a_n a_{n+1} \ldots)_2 \cdot 2^e,$$

with all a_i equal to 0 or 1.

Then for a **chopped** floating point representation, we have

$$fl(x) = \sigma \cdot \overline{x} \cdot 2^e \equiv \sigma \cdot (1.a_2 \ldots a_n)_2 \cdot 2^e.$$

For a **rounded** floating point representation, we have

$$fl(x) = \begin{cases} \sigma \cdot (1.a_2 \ldots a_n)_2 \cdot 2^e, & \textit{if} \quad a_{n+1} = 0 \\ \sigma \cdot (1.a_2 \ldots a_n)_2 \cdot 2^e + (0.00 \ldots 01)_2\,, & \textit{if} \quad a_{n+1} = 1 \end{cases}$$

Errors in floating point representation

The error $x - fl(x)$

Errors in floating point representation

The error $x - fl(x) = 0$ when x needs no change to be put into the computer or calculator, in other words x is stored exactly.

Errors in floating point representation

The error $x - fl(x) = 0$ when x needs no change to be put into the computer or calculator, in other words x is stored exactly.

Of more interest is the case when the error is nonzero.

Errors in floating point representation

The error $x - fl(x) = 0$ when x needs no change to be put into the computer or calculator, in other words x is stored exactly.

Of more interest is the case when the error is nonzero.

Consider first the case $x > 0$ (meaning $\sigma = +1$).

Errors in floating point representation

The error $x - fl(x) = 0$ when x needs no change to be put into the computer or calculator, in other words x is stored exactly.

Of more interest is the case when the error is nonzero.

Consider first the case $x > 0$ (meaning $\sigma = +1$).

With $x \neq fl(x)$,

Errors in floating point representation

The error $x - fl(x) = 0$ when x needs no change to be put into the computer or calculator, in other words x is stored exactly.

Of more interest is the case when the error is nonzero.

Consider first the case $x > 0$ (meaning $\sigma = +1$).

With $x \neq fl(x)$, and using chopping,

Errors in floating point representation

The error $x - fl(x) = 0$ when x needs no change to be put into the computer or calculator, in other words x is stored exactly.

Of more interest is the case when the error is nonzero.

Consider first the case $x > 0$ (meaning $\sigma = +1$).

With $x \neq fl(x)$, and using chopping, we have $fl(x) < x$

Errors in floating point representation

The error $x - fl(x) = 0$ when x needs no change to be put into the computer or calculator, in other words x is stored exactly.

Of more interest is the case when the error is nonzero.

Consider first the case $x > 0$ (meaning $\sigma = +1$).

With $x \neq fl(x)$, and using chopping, we have $fl(x) < x$ and the error $x - fl(x)$ is always positive.

Errors in floating point representation

The error $x - fl(x) = 0$ when x needs no change to be put into the computer or calculator, in other words x is stored exactly.

Of more interest is the case when the error is nonzero.

Consider first the case $x > 0$ (meaning $\sigma = +1$).

With $x \neq fl(x)$, and using chopping, we have $fl(x) < x$ and the error $x - fl(x)$ is always positive.

This fact has major consequences in extended numerical computations.

Errors in floating point representation

The error $x - fl(x) = 0$ when x needs no change to be put into the computer or calculator, in other words x is stored exactly.

Of more interest is the case when the error is nonzero.

Consider first the case $x > 0$ (meaning $\sigma = +1$).

With $x \neq fl(x)$, and using chopping, we have $fl(x) < x$ and the error $x - fl(x)$ is always positive.

This fact has major consequences in extended numerical computations.

With $x \neq fl(x)$ and rounding

Errors in floating point representation

The error $x - fl(x) = 0$ when x needs no change to be put into the computer or calculator, in other words x is stored exactly.

Of more interest is the case when the error is nonzero.

Consider first the case $x > 0$ (meaning $\sigma = +1$).

With $x \neq fl(x)$, and using chopping, we have $fl(x) < x$ and the error $x - fl(x)$ is always positive.

This fact has major consequences in extended numerical computations.

With $x \neq fl(x)$ and rounding, the error $x - fl(x)$ is negative for half the values of x, and it is positive for the other half of possible values of x.

Errors in floating point representation

We often write the **relative error** as

$$\frac{x - fl(x)}{x} = -\varepsilon$$

Errors in floating point representation

We often write the **relative error** as

$$\frac{x - fl(x)}{x} = -\varepsilon$$

This can be expanded to obtain

$$fl(x) = (1 + \varepsilon)x$$

Errors in floating point representation

We often write the **relative error** as

$$\frac{x - fl(x)}{x} = -\varepsilon$$

This can be expanded to obtain

$$fl(x) = (1 + \varepsilon)x$$

Thus $fl(x)$ can be considered as a perturbed value of x.

Errors in floating point representation

We often write the **relative error** as

$$\frac{x - fl(x)}{x} = -\varepsilon$$

This can be expanded to obtain

$$fl(x) = (1 + \varepsilon)x$$

Thus $fl(x)$ can be considered as a perturbed value of x.
This formula is used in many analyses of the effects of chopping and rounding errors in numerical computations.

Errors in floating point representation

We often write the **relative error** as

$$\frac{x - fl(x)}{x} = -\varepsilon$$

This can be expanded to obtain

$$fl(x) = (1 + \varepsilon)x$$

Thus $fl(x)$ can be considered as a perturbed value of x.
This formula is used in many analyses of the effects of chopping and rounding errors in numerical computations.

For bounds on ε, we have

$$-\frac{1}{2^n} \quad \leq \quad \varepsilon \leq \frac{1}{2^n}, \quad \textit{rounding}$$

Errors in floating point representation

We often write the **relative error** as

$$\frac{x - fl(x)}{x} = -\varepsilon$$

This can be expanded to obtain

$$fl(x) = (1 + \varepsilon)x$$

Thus $fl(x)$ can be considered as a perturbed value of x.
This formula is used in many analyses of the effects of chopping and rounding errors in numerical computations.

For bounds on ε, we have

$$\begin{aligned} -\frac{1}{2^n} &\leq \varepsilon \leq \frac{1}{2^n}, \quad \textit{rounding} \\ -\frac{1}{2^{n-1}} &\leq \varepsilon \leq 0, \quad \textit{chopping} \end{aligned}$$

Errors in floating point representation

We often write the **relative error** as

$$\frac{x - fl(x)}{x} = -\varepsilon$$

This can be expanded to obtain

$$fl(x) = (1 + \varepsilon)x$$

Thus $fl(x)$ can be considered as a perturbed value of x.
This formula is used in many analyses of the effects of chopping and rounding errors in numerical computations.

For bounds on ε, we have

$$\begin{aligned} -\frac{1}{2^n} &\leq \varepsilon \leq \frac{1}{2^n}, \quad \textit{rounding} \\ -\frac{1}{2^{n-1}} &\leq \varepsilon \leq 0, \quad \textit{chopping} \end{aligned}$$

There is also an extended representation, having $n = 69$ digits in its significand.

MATLAB floating point representation

MATLAB can be used to generate the binary floating point representation of a number.

MATLAB floating point representation

MATLAB can be used to generate the binary floating point representation of a number.

Execute in MATLAB the command:
>>format hex

MATLAB floating point representation

MATLAB can be used to generate the binary floating point representation of a number.

Execute in MATLAB the command:
>>format hex

This will cause all subsequent numerical output to the screen to be given in hexadecimal format (base 16).

MATLAB floating point representation

MATLAB can be used to generate the binary floating point representation of a number.

Execute in MATLAB the command:
>>format hex

This will cause all subsequent numerical output to the screen to be given in hexadecimal format (base 16).

For example, listing the number 7.125

MATLAB floating point representation

MATLAB can be used to generate the binary floating point representation of a number.

Execute in MATLAB the command:
>>format hex

This will cause all subsequent numerical output to the screen to be given in hexadecimal format (base 16).

For example, listing the number 7.125 results in an output of
$>> 401c800000000000$

MATLAB floating point representation

MATLAB can be used to generate the binary floating point representation of a number.

Execute in MATLAB the command:
>>format hex

This will cause all subsequent numerical output to the screen to be given in hexadecimal format (base 16).

For example, listing the number 7.125 results in an output of
$>> 401c800000000000$

The 16 hexadecimal digits are
$\{0; 1; 2; 3; 4; 5; 6; 7; 8; 9; a; b; c; d; e; f\}$

MATLAB floating point representation

MATLAB can be used to generate the binary floating point representation of a number.

Execute in MATLAB the command:
>>format hex

This will cause all subsequent numerical output to the screen to be given in hexadecimal format (base 16).

For example, listing the number 7.125 results in an output of
$>> 401c800000000000$

The 16 hexadecimal digits are
$\{0; 1; 2; 3; 4; 5; 6; 7; 8; 9; a; b; c; d; e; f\}$

To obtain the binary representation, convert each hexadecimal digit to a four digit binary number according to the table on next slide:

MATLAB floating point representation

Format hex	Format binary	Format hex	Format binary
0	0000	8	1000
1	0001	9	1001
2	0010	*a*	1010
3	0011	*b*	1011
4	0100	*c*	1100
5	0101	*d*	1101
6	0110	*e*	1110
7	0111	*f*	1111

MATLAB floating point representation

$$401c800000000000$$

MATLAB floating point representation

$$401c800000000000$$

$$\overbrace{0100}^{4}\overbrace{0000}^{0}\overbrace{0011}^{1}\overbrace{1100}^{c}\overbrace{1000}^{8}\overbrace{0000}^{0}\overbrace{0000}^{0}\ldots\overbrace{0000}^{0}$$

MATLAB floating point representation

$$401c800000000000$$

$$\overbrace{0100}^{4}\overbrace{0000}^{0}\overbrace{0011}^{1}\overbrace{1100}^{c}\overbrace{1000}^{8}\overbrace{0000}^{0}\overbrace{0000}^{0}\ldots\overbrace{0000}^{0}$$

σ	E			$\overline{x}$		
b_1	b_2	$\ldots$	b_{12}	b_{13}	$\ldots$	b_{64}

MATLAB floating point representation

$$401c800000000000$$

$$\overbrace{0100}^{4}\overbrace{0000}^{0}\overbrace{0011}^{1}\overbrace{1100}^{c}\overbrace{1000}^{8}\overbrace{0000}^{0}\overbrace{0000}^{0}\ldots\overbrace{0000}^{0}$$

σ	E			$\overline{x}$		
b_1	b_2	$\ldots$	b_{12}	b_{13}	$\ldots$	b_{64}

$$\underbrace{0}_{\sigma}\ \underbrace{10000000001}_{E}\ \underbrace{1100100000000000\ldots0000}_{1.b_{13}b_{14}\ldots b_{64}=\overline{x}}$$

Errors

Let x_T denote the true value of some number, usually unknown in practice

Errors

Let x_T denote the true value of some number, usually unknown in practice and let x_A denote an approximation of x_T.

Errors

Let x_T denote the true value of some number, usually unknown in practice and let x_A denote an approximation of x_T.

The **error** in x_A is

$$err(x_A) = x_T - x_A$$

Errors

Let x_T denote the true value of some number, usually unknown in practice and let x_A denote an approximation of x_T.

The **error** in x_A is

$$err(x_A) = x_T - x_A$$

The **relative error** in x_A is

$$rel(x_A) = \frac{err(x_A)}{x_T} = \frac{x_T - x_A}{x_T}$$

Errors

Let x_T denote the true value of some number, usually unknown in practice and let x_A denote an approximation of x_T.

The **error** in x_A is

$$err(x_A) = x_T - x_A$$

The **relative error** in x_A is

$$rel(x_A) = \frac{err(x_A)}{x_T} = \frac{x_T - x_A}{x_T}$$

Example:

$$x_T = e, \quad x_A = \frac{19}{7}$$

Errors

Let x_T denote the true value of some number, usually unknown in practice and let x_A denote an approximation of x_T.

The **error** in x_A is

$$err(x_A) = x_T - x_A$$

The **relative error** in x_A is

$$rel(x_A) = \frac{err(x_A)}{x_T} = \frac{x_T - x_A}{x_T}$$

Example:

$$x_T = e, \quad x_A = \frac{19}{7}$$

$$\begin{aligned} err(x_A) &= e - \frac{19}{7} \approx 0.003996 \\ rel(x_A) &\approx \frac{0.003996}{e} \approx 0.00147 \end{aligned}$$

Errors in floating point representation

Errors in floating point representation

Example: Suppose the distance between two cities is $D_T = 100km$

Errors in floating point representation

Example: Suppose the distance between two cities is $D_T = 100km$ and let this distance be approximated with $D_A = 99km$.

Errors in floating point representation

Example: Suppose the distance between two cities is $D_T = 100km$ and let this distance be approximated with $D_A = 99km$.

In this case,

$$err(D_A) \quad = \quad D_T - D_A = 1km,$$

Errors in floating point representation

Example: Suppose the distance between two cities is $D_T = 100km$ and let this distance be approximated with $D_A = 99km$.

In this case,

$$\begin{aligned} err(D_A) &= D_T - D_A = 1km, \\ rel(D_A) &= \frac{err(D_A)}{D_T} = \frac{1}{100} = 0.01 = 1\% \end{aligned}$$

Errors in floating point representation

Example: Suppose the distance between two cities is $D_T = 100km$ and let this distance be approximated with $D_A = 99km$.

In this case,

$$\begin{aligned} err(D_A) &= D_T - D_A = 1km, \\ rel(D_A) &= \frac{err(D_A)}{D_T} = \frac{1}{100} = 0.01 = 1\% \end{aligned}$$

Now, suppose that distance is $d_T = 2km$

Errors in floating point representation

Example: Suppose the distance between two cities is $D_T = 100km$ and let this distance be approximated with $D_A = 99km$.

In this case,

$$\begin{aligned} err(D_A) &= D_T - D_A = 1km, \\ rel(D_A) &= \frac{err(D_A)}{D_T} = \frac{1}{100} = 0.01 = 1\% \end{aligned}$$

Now, suppose that distance is $d_T = 2km$ and estimate it with $d_A = 1km$.

Errors in floating point representation

Example: Suppose the distance between two cities is $D_T = 100km$ and let this distance be approximated with $D_A = 99km$.

In this case,

$$\begin{aligned} err(D_A) &= D_T - D_A = 1km, \\ rel(D_A) &= \frac{err(D_A)}{D_T} = \frac{1}{100} = 0.01 = 1\% \end{aligned}$$

Now, suppose that distance is $d_T = 2km$ and estimate it with $d_A = 1km$. Then

$$err(d_A) = d_T - d_A = 1km,$$

Errors in floating point representation

Example: Suppose the distance between two cities is $D_T = 100km$ and let this distance be approximated with $D_A = 99km$.

In this case,

$$\begin{aligned} err(D_A) &= D_T - D_A = 1km, \\ rel(D_A) &= \frac{err(D_A)}{D_T} = \frac{1}{100} = 0.01 = 1\% \end{aligned}$$

Now, suppose that distance is $d_T = 2km$ and estimate it with $d_A = 1km$. Then

$$\begin{aligned} err(d_A) &= d_T - d_A = 1km, \\ rel(d_A) &= \frac{err(d_A)}{d_T} = \frac{1}{2} = 0.5 = 50\% \end{aligned}$$

Errors in floating point representation

Example: Suppose the distance between two cities is $D_T = 100km$ and let this distance be approximated with $D_A = 99km$.

In this case,

$$\begin{aligned} err(D_A) &= D_T - D_A = 1km, \\ rel(D_A) &= \frac{err(D_A)}{D_T} = \frac{1}{100} = 0.01 = 1\% \end{aligned}$$

Now, suppose that distance is $d_T = 2km$ and estimate it with $d_A = 1km$. Then

$$\begin{aligned} err(d_A) &= d_T - d_A = 1km, \\ rel(d_A) &= \frac{err(d_A)}{d_T} = \frac{1}{2} = 0.5 = 50\% \end{aligned}$$

In both cases the error is the same.

Errors in floating point representation

Example: Suppose the distance between two cities is $D_T = 100km$ and let this distance be approximated with $D_A = 99km$.

In this case,

$$\begin{aligned} err(D_A) &= D_T - D_A = 1km, \\ rel(D_A) &= \frac{err(D_A)}{D_T} = \frac{1}{100} = 0.01 = 1\% \end{aligned}$$

Now, suppose that distance is $d_T = 2km$ and estimate it with $d_A = 1km$. Then

$$\begin{aligned} err(d_A) &= d_T - d_A = 1km, \\ rel(d_A) &= \frac{err(d_A)}{d_T} = \frac{1}{2} = 0.5 = 50\% \end{aligned}$$

In both cases the error is the same.

But, obviously $D_A$is a better approximation of D_T , then d_A of d_T.

Sources of Error

Sources of Error

The sources of error in the computation of the solution of a mathematical model for some physical situation can be roughly characterised as follows:

Sources of Error

The sources of error in the computation of the solution of a mathematical model for some physical situation can be roughly characterised as follows:

1. Modelling Error.

Sources of Error

The sources of error in the computation of the solution of a mathematical model for some physical situation can be roughly characterised as follows:

1. Modelling Error.

Consider the example of a projectile of mass m that is travelling thrugh the earth's atmosphere.

Sources of Error

The sources of error in the computation of the solution of a mathematical model for some physical situation can be roughly characterised as follows:

1. Modelling Error.

Consider the example of a projectile of mass m that is travelling thrugh the earth's atmosphere. A simple and oftenly used description of projectile motion is given by

Sources of Error

The sources of error in the computation of the solution of a mathematical model for some physical situation can be roughly characterised as follows:

1. Modelling Error.

Consider the example of a projectile of mass m that is travelling thrugh the earth's atmosphere. A simple and oftenly used description of projectile motion is given by

$$m\frac{d^2\overrightarrow{r}}{dt^2}(t) = -mg\overrightarrow{k} - b\frac{d\overrightarrow{r}}{dt}$$

with $b \geq 0$.

Sources of Error

The sources of error in the computation of the solution of a mathematical model for some physical situation can be roughly characterised as follows:

1. Modelling Error.

Consider the example of a projectile of mass m that is travelling thrugh the earth's atmosphere. A simple and oftenly used description of projectile motion is given by

$$m\frac{d^2\overrightarrow{r}}{dt^2}(t) = -mg\overrightarrow{k} - b\frac{d\overrightarrow{r}}{dt}$$

with $b \geq 0$. In this, $\overrightarrow{r}(t)$ is the vector position of the projectile;

Sources of Error

The sources of error in the computation of the solution of a mathematical model for some physical situation can be roughly characterised as follows:

1. Modelling Error.

Consider the example of a projectile of mass m that is travelling thrugh the earth's atmosphere. A simple and oftenly used description of projectile motion is given by

$$m\frac{d^2\overrightarrow{r}}{dt^2}(t) = -mg\overrightarrow{k} - b\frac{d\overrightarrow{r}}{dt}$$

with $b \geq 0$. In this, $\overrightarrow{r}(t)$ is the vector position of the projectile; and the final term in the equation represents friction force in air.

Sources of Error

The sources of error in the computation of the solution of a mathematical model for some physical situation can be roughly characterised as follows:

1. Modelling Error.

Consider the example of a projectile of mass m that is travelling thrugh the earth's atmosphere. A simple and oftenly used description of projectile motion is given by

$$m\frac{d^2\overrightarrow{r}}{dt^2}(t) = -mg\overrightarrow{k} - b\frac{d\overrightarrow{r}}{dt}$$

with $b \geq 0$. In this, $\overrightarrow{r}(t)$ is the vector position of the projectile; and the final term in the equation represents friction force in air. If there is an error in this a model of a physical situation, then the numerical solution of this equation is not going to improve the results.

Sources of Error

2. **Physical / Observational / Measurement Error**.

Sources of Error

2. Physical / Observational / Measurement Error.
The radius of an electron is given by

$$(2.81777 + \varepsilon) \times 10^{-13} cm, \quad |\varepsilon| \leq 0.00011$$

Sources of Error

2. Physical / Observational / Measurement Error.
The radius of an electron is given by

$$(2.81777 + \varepsilon) \times 10^{-13} cm, \quad |\varepsilon| \leq 0.00011$$

This error cannot be removed, and it must affect the accuracy of any computation in which it is used.

Sources of Error

2. Physical / Observational / Measurement Error.
The radius of an electron is given by

$$(2.81777 + \varepsilon) \times 10^{-13} cm, \quad |\varepsilon| \leq 0.00011$$

This error cannot be removed, and it must affect the accuracy of any computation in which it is used.

We need to be aware of these effects and to so arrange the computation as to minimize the effects.

Sources of Error

3. **Approximation Error**.

Sources of Error

3. Approximation Error.

This is also called **"discretization error"** and **"truncation error"**;

Sources of Error

3. Approximation Error.
This is also called **"discretization error"** and **"truncation error"**; and it is the main source of error with which we deal in this course.

Sources of Error

3. Approximation Error.
This is also called **"discretization error"** and **"truncation error"**; and it is the main source of error with which we deal in this course. Such errors generally occur when we replace a computationally unsolvable problem with a nearby problem that is more tractable computationally.

Sources of Error

3. Approximation Error.
This is also called **"discretization error"** and **"truncation error"**; and it is the main source of error with which we deal in this course. Such errors generally occur when we replace a computationally unsolvable problem with a nearby problem that is more tractable computationally.
For example, the Taylor polynomial approximation

$$e^x \approx 1 + x + \frac{1}{2}x^2$$

contains an "approximation error".

Sources of Error

3. Approximation Error.
This is also called **"discretization error"** and **"truncation error"**; and it is the main source of error with which we deal in this course. Such errors generally occur when we replace a computationally unsolvable problem with a nearby problem that is more tractable computationally.
For example, the Taylor polynomial approximation

$$e^x \approx 1 + x + \frac{1}{2}x^2$$

contains an "approximation error".
The numerical integration

$$\int_0^1 f(x)dx \approx \frac{1}{N}\sum_{j=1}^{N} f\left(\frac{j}{N}\right)$$

contains an approximation error.

Sources of Error

4. **Finiteness of Algorithm Error**

Sources of Error

4. Finiteness of Algorithm Error

This is an error due to stopping an algorithm after a finite number of iterations.

Sources of Error

4. Finiteness of Algorithm Error
This is an error due to stopping an algorithm after a finite number of iterations.

Even if theoretically an algorithm can run for indefinite time, after a finite (usually specified) number of iterations the algorithm will be stopped.

Sources of Error

5. **Blunders.**

Sources of Error

5. Blunders.

In the pre-computer era, blunders were mostly arithmetic errors.

Sources of Error

5. Blunders.
In the pre-computer era, blunders were mostly arithmetic errors. In the earlier years of the computer era, the typical blunder was a programming bug.

Sources of Error

5. Blunders.
In the pre-computer era, blunders were mostly arithmetic errors. In the earlier years of the computer era, the typical blunder was a programming bug. Present day "blunders" are still often programming errors.

Sources of Error

5. Blunders.
In the pre-computer era, blunders were mostly arithmetic errors. In the earlier years of the computer era, the typical blunder was a programming bug. Present day "blunders" are still often programming errors. But now they are often much more difficult to find, as they are often embedded in very large codes which may mask their effect.

Sources of Error

5. Blunders.
In the pre-computer era, blunders were mostly arithmetic errors. In the earlier years of the computer era, the typical blunder was a programming bug. Present day "blunders" are still often programming errors. But now they are often much more difficult to find, as they are often embedded in very large codes which may mask their effect.

Sources of Error

5. Blunders.

In the pre-computer era, blunders were mostly arithmetic errors. In the earlier years of the computer era, the typical blunder was a programming bug. Present day "blunders" are still often programming errors. But now they are often much more difficult to find, as they are often embedded in very large codes which may mask their effect.

Some simple rules to decrease the risk of having a bug in the code:

Sources of Error

5. Blunders.

In the pre-computer era, blunders were mostly arithmetic errors. In the earlier years of the computer era, the typical blunder was a programming bug. Present day "blunders" are still often programming errors. But now they are often much more difficult to find, as they are often embedded in very large codes which may mask their effect.

Some simple rules to decrease the risk of having a bug in the code:

- Break programs into small testable subprograms;
- Run test cases for which you know the outcome;
- When running the full code, maintain a skeptical eye on the output, checking whether the output is reasonable or not.

Sources of Error

6. **Rounding/chopping Error**.

Sources of Error

6. Rounding/chopping Error.
This is the main source of many problems, especially problems in solving systems of linear equations.

Sources of Error

6. Rounding/chopping Error.
This is the main source of many problems, especially problems in solving systems of linear equations.

We later look at the effects of such errors.

Sources of Error

7. **Finitness of precision errors**

Sources of Error

7. Finitness of precision errors

All the numbers stored in computer memory are subject to the finiteness of allocated space for storage.

Pendulum Example

Original problem in engineering or in science to be solved:

Pendulum Example

Original problem in engineering or in science to be solved:

Pendulum Example

Original problem in engineering or in science to be solved:

Model this physical problem mathematically.

Pendulum Example

Original problem in engineering or in science to be solved:

Model this physical problem mathematically.
Second Newton law provides us with:

Pendulum Example

Original problem in engineering or in science to be solved:

Model this physical problem mathematically.
Second Newton law provides us with:

$$\ddot{\theta} = -\frac{g}{l}\sin\theta$$

Pendulum Example

Original problem in engineering or in science to be solved:

Model this physical problem mathematically.
Second Newton law provides us with:

$$\ddot{\theta} = -\frac{g}{l}\sin\theta$$

$$\begin{cases} \dot{\theta} = \omega \\ \dot{\omega} = -\frac{g}{l}\sin\theta \end{cases}$$

Pendulum Example

Problem of continuous mathematics:

Pendulum Example

Problem of continuous mathematics:

$$\begin{cases} \dot{\theta} &= \omega \\ \dot{\omega} &= -\frac{g}{l}\sin\theta \end{cases}$$

Pendulum Example

Problem of continuous mathematics:

$$\begin{cases} \dot{\theta} &= \omega \\ \dot{\omega} &= -\frac{g}{l}\sin\theta \end{cases}$$

- Modeling Errors
- Physical Errors

Pendulum Example

Mathematical Algorithms:

Pendulum Example

Mathematical Algorithms:

$$\begin{cases} \theta_{n+1} = \theta_n + h\omega_{n+1} \\ \omega_{n+1} = \omega_n - h\frac{g}{l}\sin(\theta_n) \end{cases}$$

Pendulum Example

Mathematical Algorithms:

$$\begin{cases} \theta_{n+1} = \theta_n + h\omega_{n+1} \\ \omega_{n+1} = \omega_n - h\frac{g}{l}\sin(\theta_n) \end{cases}$$

- Discretisation Errors
- Finiteness of Algorithm Errors

Pendulum Example

Computer Implementation:

Pendulum Example

Computer Implementation:

```
for i=1:Nmax
        Omega = Omega - H*g/L*sin(Theta);
        Theta = Theta + H*Omega
end
```

Pendulum Example

Computer Implementation:

```
for i=1:Nmax
        Omega = Omega - H*g/L*sin(Theta);
        Theta = Theta + H*Omega
end
```

- Rounding / Chopping Errors
- Bugs in the Code
- Finite Precision Errors

Numerical Analysis / Numerical Methods

Prof.univ. dr.hab. Viorel Bostan

Lecture 1 (part 3), Spring 2022

Significant digits

Let x_T and x_A be true and, respectively approximated values.

Definition

We say that approximation x_A has m **significant digits** with respect to true value x_T if $|err(x_A)| \leqslant 5$ units in the $(m+1)$-st digit, beginning with the first nonzero digit in x_T.

Significant digits

Let x_T and x_A be true and, respectively approximated values.

Definition

We say that approximation x_A has m **significant digits** with respect to true value x_T if $|err(x_A)| \leqslant 5$ units in the $(m+1)$-st digit, beginning with the first nonzero digit in x_T.

Example

Let

$$\begin{aligned} x_T &= e = 2.718281828459045 23\ldots, \\ x_A &= \tfrac{19}{7} = 2.714285714285714285\ldots, \\ err(x_A) &\approx 0.003996\ldots \end{aligned}$$

Therefore, there are 3 significant digits in this approximation.

Significant digits

Another way to look at significant digits (also called significant figures) regardless whether we have an approximation or not is each of the digits of a number that are used to express it to the required degree of accuracy, starting from the first non-zero digit.

There are several rules:

- All non-zero numbers are significant.
  The number 37.9 has 3 significant digits.
- Zeros between two non-zero digits are significant.
  The number 4001.7 has 5 significant digits,
  while the number 2005 has 4 significant digits.
- Leading zeros are not significant.
  The number 0.89 has only 2 significant digits, and 0.00017 also has 2 significant digits.
- Trailing zeros to the right of the decimal are significant.
  There are 4 significant digits in 92.00 and 5 significant digits in 3.0000.

Loss of significance errors

This can be considered a source of error or a consequence of the finiteness of calculator and computer arithmetic.

Loss of significance errors

This can be considered a source of error or a consequence of the finiteness of calculator and computer arithmetic.

Example 1. Define

$$f(x) = x\left(\sqrt{x+1} - \sqrt{x}\right)$$

and consider evaluating it on a 6-digit decimal calculator which uses rounded arithmetic.

Loss of significance errors

This can be considered a source of error or a consequence of the finiteness of calculator and computer arithmetic.

Example 1. Define

$$f(x) = x\left(\sqrt{x+1} - \sqrt{x}\right)$$

and consider evaluating it on a 6-digit decimal calculator which uses rounded arithmetic.

x	Computed $f(x)$	True $f(x)$	Error
1	0.4142210	0.414214	$7.0000e-006$
10	1.54340	1.54347	$-7.0000e-005$
100	4.99000	4.98756	0.0024
1000	15.8000	15.8074	-0.0074
10000	50.0000	49.9988	0.0012
100000	100.000	158.113	-58.1130

Loss of significance errors. Example 1

In order to localize the error, consider the case $x = 100$.

Loss of significance errors. Example 1

In order to localize the error, consider the case $x = 100$.

The calculator with 6 decimal digits will provide us with the following values

$$\sqrt{100} = 10, \quad \sqrt{101} = 10.0499.$$

Loss of significance errors. Example 1

In order to localize the error, consider the case $x = 100$.

The calculator with 6 decimal digits will provide us with the following values

$$\sqrt{100} = 10, \quad \sqrt{101} = 10.0499.$$

Then,

$$\sqrt{x+1} - \sqrt{x} = \sqrt{101} - \sqrt{100} = 0.0499000,$$

while the exact value is 0.0498756.

Loss of significance errors. Example 1

In order to localize the error, consider the case $x = 100$.

The calculator with 6 decimal digits will provide us with the following values

$$\sqrt{100} = 10, \quad \sqrt{101} = 10.0499.$$

Then,

$$\sqrt{x+1} - \sqrt{x} = \sqrt{101} - \sqrt{100} = 0.0499000,$$

while the exact value is 0.0498756.

Three significant digits in $\sqrt{x+1} = \sqrt{101}$ have been lost from $\sqrt{x} = \sqrt{100}$.

Loss of significance errors. Example 1

In order to localize the error, consider the case $x = 100$.

The calculator with 6 decimal digits will provide us with the following values

$$\sqrt{100} = 10, \quad \sqrt{101} = 10.0499.$$

Then,

$$\sqrt{x+1} - \sqrt{x} = \sqrt{101} - \sqrt{100} = 0.0499000,$$

while the exact value is 0.0498756.

Three significant digits in $\sqrt{x+1} = \sqrt{101}$ have been lost from $\sqrt{x} = \sqrt{100}$.

The loss of precision is due to the form of the function $f(x)$ and the finiteness of the precision of the 6 digit calculator.

Loss of significance errors. Example 2

In this particular case, we can avoid the loss of precision by rewriting the function as follows:

Loss of significance errors. Example 2

In this particular case, we can avoid the loss of precision by rewriting the function as follows:

$$\begin{aligned} f(x) &= x\frac{\sqrt{x+1}+\sqrt{x}}{\sqrt{x+1}+\sqrt{x}} \cdot \frac{\sqrt{x+1}-\sqrt{x}}{1} \\ &= \frac{x}{\sqrt{x+1}+\sqrt{x}}. \end{aligned}$$

Thus we will avoid the subtraction on near quantities.

Doing so gives us

$$f(100) = 4.98756,$$

a value with 6 significant digits.

Loss of significance errors. Example 2

Example 2. Define

$$g(x) = \frac{1 - \cos x}{x^2}$$

and consider evaluating it on a 10-digit decimal calculator which uses rounded arithmetic.

Loss of significance errors. Example 2

Example 2. Define

$$g(x) = \frac{1 - \cos x}{x^2}$$

and consider evaluating it on a 10-digit decimal calculator which uses rounded arithmetic.

x	Computed $f(x)$	True $f(x)$	Error
0.1	0.4995834700	0.4995834722	$-2.2000e-009$
0.01	0.4999960000	0.4999958333	$1.6670e-007$
0.001	0.5000000000	0.4999999583	$4.1700e-008$
0.0001	0.5000000000	0.4999999996	$4.0000e-010$
0.00001	0.0	0.5000000000	0.5

Loss of significance errors. Example 2

Consider one case, that of $x = 0.001$.

Loss of significance errors. Example 2

Consider one case, that of $x = 0.001$.

Then on the calculator:

$$\begin{aligned} \cos(0.001) &= 0.9999994999 \\ 1 - \cos(0.001) &= 5.001 \times 10^{-7} \\ \frac{1 - \cos(0.001)}{(0.001)^2} &= 0.5001000000 \end{aligned}$$

Loss of significance errors. Example 2

Consider one case, that of $x = 0.001$.

Then on the calculator:

$$\begin{aligned} \cos(0.001) &= 0.9999994999 \\ 1 - \cos(0.001) &= 5.001 \times 10^{-7} \\ \frac{1 - \cos(0.001)}{(0.001)^2} &= 0.5001000000 \end{aligned}$$

The true answer is

$$f(0.001) = 0.4999999583$$

Loss of significance errors. Example 2

Consider one case, that of $x = 0.001$.

Then on the calculator:

$$\begin{aligned} \cos(0.001) &= 0.9999994999 \\ 1 - \cos(0.001) &= 5.001 \times 10^{-7} \\ \frac{1 - \cos(0.001)}{(0.001)^2} &= 0.5001000000 \end{aligned}$$

The true answer is

$$f(0.001) = 0.4999999583$$

The relative error in our answer is

$$\frac{0.4999999583 - 0.5001}{0.4999999583} = \frac{-0.0001000417}{0.4999999583} = -0.0002$$

Loss of significance errors. Example 2

Consider one case, that of $x = 0.001$.

Then on the calculator:

$$\begin{aligned} \cos(0.001) &= 0.9999994999 \\ 1 - \cos(0.001) &= 5.001 \times 10^{-7} \\ \frac{1 - \cos(0.001)}{(0.001)^2} &= 0.5001000000 \end{aligned}$$

The true answer is

$$f(0.001) = 0.4999999583$$

The relative error in our answer is

$$\frac{0.4999999583 - 0.5001}{0.4999999583} = \frac{-0.0001000417}{0.4999999583} = -0.0002$$

There are 3 significant digits in the answer.
How can such a straightforward and short calculation lead to such a large error (relative to the accuracy of the calculator)?

Loss of significance errors. Example 2

When two numbers are nearly equal and we subtract them, then we suffer a "loss of significance error" in the calculation.

Loss of significance errors. Example 2

When two numbers are nearly equal and we subtract them, then we suffer a "loss of significance error" in the calculation.

In some cases, these can be quite subtle and difficult to detect.

Loss of significance errors. Example 2

When two numbers are nearly equal and we subtract them, then we suffer a "loss of significance error" in the calculation.

In some cases, these can be quite subtle and difficult to detect.

And even after they are detected, they may be difficult to fix.

Loss of significance errors. Example 2

When two numbers are nearly equal and we subtract them, then we suffer a "loss of significance error" in the calculation.

In some cases, these can be quite subtle and difficult to detect.

And even after they are detected, they may be difficult to fix.

The last example, fortunately, can be fixed in a number of ways. Easiest is to use a trigonometric identity:

$$\cos(x) = 1 - 2\sin^2(x/2),$$

$$f(x) = \frac{1-\cos x}{x^2} = \frac{2\sin^2(x/2)}{x^2} = \frac{1}{2}\left(\frac{\sin(x/2)}{x/2}\right)^2.$$

This latter formula, with $x = 0.001$, yields a computed value of 0.4999999584, nearly the true answer. We could also have used a Taylor polynomial for $\cos(x)$ around $x = 0$ to obtain a better approximation to $f(x)$ for small values of x.

Loss of significance errors. Example 3

Example 3. Evaluate e^{-5} using a Taylor polynomial approximation:

$$e^{-5} = 1 + \frac{(-5)}{1!} + \frac{(-5)^2}{2!} + \frac{(-5)^3}{3!} + \frac{(-5)^4}{4!} + \frac{(-5)^5}{5!} + \frac{(-5)^6}{6!} \dots$$

Loss of significance errors. Example 3

Example 3. Evaluate e^{-5} using a Taylor polynomial approximation:

$$e^{-5} = 1 + \frac{(-5)}{1!} + \frac{(-5)^2}{2!} + \frac{(-5)^3}{3!} + \frac{(-5)^4}{4!} + \frac{(-5)^5}{5!} + \frac{(-5)^6}{6!} \dots$$

With $n = 25$, the error is

$$\left| \frac{(-5)^{-26}}{26!} e^c \right| \leq 10^{-8}.$$

Loss of significance errors. Example 3

Example 3. Evaluate e^{-5} using a Taylor polynomial approximation:

$$e^{-5} = 1 + \frac{(-5)}{1!} + \frac{(-5)^2}{2!} + \frac{(-5)^3}{3!} + \frac{(-5)^4}{4!} + \frac{(-5)^5}{5!} + \frac{(-5)^6}{6!} \dots$$

With $n = 25$, the error is

$$\left| \frac{(-5)^{-26}}{26!} e^c \right| \leq 10^{-8}.$$

Imagine calculating this polynomial using a computer with 4 digit decimal arithmetic and rounding.

Loss of significance errors. Example 3

Example 3. Evaluate e^{-5} using a Taylor polynomial approximation:

$$e^{-5} = 1 + \frac{(-5)}{1!} + \frac{(-5)^2}{2!} + \frac{(-5)^3}{3!} + \frac{(-5)^4}{4!} + \frac{(-5)^5}{5!} + \frac{(-5)^6}{6!} \dots$$

With $n = 25$, the error is

$$\left| \frac{(-5)^{-26}}{26!} e^c \right| \leq 10^{-8}.$$

Imagine calculating this polynomial using a computer with 4 digit decimal arithmetic and rounding.

To make the point about cancellation more strongly, imagine that each of the terms in the above polynomial is calculated exactly and then rounded to the arithmetic of the computer. We add the terms exactly and then we round to four digits.

Loss of significance errors. Example 3

Degree	Term	Sum	Degree	Term	Sum
0	1.000	1.000	13	-0.1960	-0.04230
1	-5.000	-4.000	14	$0.7001e-1$	0.02771
2	12.50	8.500	15	$-0.2334e-1$	0.004370
3	-20.83	-12.33	16	$0.7293e-2$	0.01166
4	26.04	13.71	17	$-0.2145e-2$	0.009518
5	-26.04	-12.33	18	$0.5958e-3$	0.01011
6	21.70	9.370	19	$-0.1568e-3$	0.009957
7	-15.50	-6.130	20	$0.3920e-4$	0.009996
8	9.688	3.558	21	$-0.9333e-5$	0.009987
9	-5.382	-1.824	22	$0.2121e-5$	0.009989
10	2.691	0.8670	23	$-0.4611e-6$	0.009989
11	-1.223	-0.3560	24	$0.9670e-7$	0.009989
12	0.5097	0.1537	25	$-0.1921e-7$	0.009989

Loss of significance errors. Example 3

Degree	Term	Sum	Degree	Term	Sum
0	1.000	1.000	13	-0.1960	-0.04230
1	-5.000	-4.000	14	$0.7001e-1$	0.02771
2	12.50	8.500	15	$-0.2334e-1$	0.004370
3	-20.83	-12.33	16	$0.7293e-2$	0.01166
4	26.04	13.71	17	$-0.2145e-2$	0.009518
5	-26.04	-12.33	18	$0.5958e-3$	0.01011
6	21.70	9.370	19	$-0.1568e-3$	0.009957
7	-15.50	-6.130	20	$0.3920e-4$	0.009996
8	9.688	3.558	21	$-0.9333e-5$	0.009987
9	-5.382	-1.824	22	$0.2121e-5$	0.009989
10	2.691	0.8670	23	$-0.4611e-6$	0.009989
11	-1.223	-0.3560	24	$0.9670e-7$	0.009989
12	0.5097	0.1537	25	$-0.1921e-7$	0.009989

True answer is 0.006738!

Loss of significance errors. Example 3

Look at the numbers being added and their accuracy, ex. 3rd term:

$$\frac{(-5)^3}{3!} = -\frac{125}{6} = -20.83$$

in the 4 digit decimal calculation, with an error of magnitude 0.00333.

Loss of significance errors. Example 3

Look at the numbers being added and their accuracy, ex. 3rd term:

$$\frac{(-5)^3}{3!} = -\frac{125}{6} = -20.83$$

in the 4 digit decimal calculation, with an error of magnitude 0.00333.

Note that this error in an intermediate step is of same magnitude as the true answer 0.006738 being sought.

Loss of significance errors. Example 3

Look at the numbers being added and their accuracy, ex. 3rd term:

$$\frac{(-5)^3}{3!} = -\frac{125}{6} = -20.83$$

in the 4 digit decimal calculation, with an error of magnitude 0.00333.

Note that this error in an intermediate step is of same magnitude as the true answer 0.006738 being sought.

Other similar errors are present in calculating other coefficients, and thus they cause a major error in the final answer being calculated.

Loss of significance errors. Example 3

Look at the numbers being added and their accuracy, ex. 3rd term:

$$\frac{(-5)^3}{3!} = -\frac{125}{6} = -20.83$$

in the 4 digit decimal calculation, with an error of magnitude 0.00333.

Note that this error in an intermediate step is of same magnitude as the true answer 0.006738 being sought.

Other similar errors are present in calculating other coefficients, and thus they cause a major error in the final answer being calculated.

General principle

Whenever a sum is being formed in which the final answer is much smaller than some of the terms being combined, then a loss of significance error is occurring.

Noise in function evaluation

Consider plotting the function

$$f(x) = (x-1)^3 = x^3 - 3x^2 + 3x - 1 = -1 + x(3 + x(-3 + x)).$$

Noise in function evaluation

Noise in function evaluation

Whenever a function $f(x)$ is evaluated, there are arithmetic operations carried out which involve rounding or chopping errors.

Noise in function evaluation

Whenever a function $f(x)$ is evaluated, there are arithmetic operations carried out which involve rounding or chopping errors.

This means that what the computer eventually returns as an answer contains noise.

Noise in function evaluation

Whenever a function $f(x)$ is evaluated, there are arithmetic operations carried out which involve rounding or chopping errors.

This means that what the computer eventually returns as an answer contains noise.

This noise is generally "random" and small.

Noise in function evaluation

Whenever a function $f(x)$ is evaluated, there are arithmetic operations carried out which involve rounding or chopping errors.

This means that what the computer eventually returns as an answer contains noise.

This noise is generally "random" and small.

But it can affect the accuracy of other calculations which depend on $f(x)$.

Underflow errors

Consider evaluating the function

$$f(x) = x^{10}$$

for values of x close to 0.

Underflow errors

Consider evaluating the function

$$f(x) = x^{10}$$

for values of x close to 0.

When using IEEE single precision arithmetic, the smallest nonzero positive number expressible (stored exactly) in normalized floating-point format is

$$m = 2^{-126} \approx 1.18 \times 10^{-38}.$$

Underflow errors

Consider evaluating the function

$$f(x) = x^{10}$$

for values of x close to 0.

When using IEEE single precision arithmetic, the smallest nonzero positive number expressible (stored exactly) in normalized floating-point format is

$$m = 2^{-126} \approx 1.18 \times 10^{-38}.$$

Thus, $f(x)$ will be stored as zero if

$$\begin{aligned} x^{10} &< m, \\ |x| &< m^{\frac{1}{10}}, \\ |x| &< 1.61 \times 10^{-4}, \\ -0.000161 < x &< 0.000161. \end{aligned}$$

Storing a nonzero number as a zero leads to **underflow error**.

Overflow errors

Attempts to use numbers that are too large for the floating-point format will lead to **overflow errors**.

Overflow errors

Attempts to use numbers that are too large for the floating-point format will lead to **overflow errors**.

These are generally fatal errors on most computers. With the IEEE floating-point format, overflow errors can be carried along as having a value of $\pm\infty$ or NaN, depending on the context.

Overflow errors

Attempts to use numbers that are too large for the floating-point format will lead to **overflow errors**.

These are generally fatal errors on most computers. With the IEEE floating-point format, overflow errors can be carried along as having a value of $\pm\infty$ or NaN, depending on the context.

Usually an overflow error is an indication of a more significant problem and the user needs to be aware of such errors.

Overflow errors

Attempts to use numbers that are too large for the floating-point format will lead to **overflow errors**.

These are generally fatal errors on most computers. With the IEEE floating-point format, overflow errors can be carried along as having a value of $\pm\infty$ or NaN, depending on the context.

Usually an overflow error is an indication of a more significant problem and the user needs to be aware of such errors.

When using IEEE single precision arithmetic, the largest nonzero positive number expressible in normalized floating point format is

$$m = 2^{128}\left(1 - 2^{-24}\right) = 3.40 \times 10^{38}$$

Overflow errors

Attempts to use numbers that are too large for the floating-point format will lead to **overflow errors**.

These are generally fatal errors on most computers. With the IEEE floating-point format, overflow errors can be carried along as having a value of $\pm\infty$ or NaN, depending on the context.

Usually an overflow error is an indication of a more significant problem and the user needs to be aware of such errors.

When using IEEE single precision arithmetic, the largest nonzero positive number expressible in normalized floating point format is

$$m = 2^{128}\left(1 - 2^{-24}\right) = 3.40 \times 10^{38}$$

Thus, $f(x)$ will overflow if

$$\begin{aligned} x^{10} &> m, \\ |x| &> m^{\frac{1}{10}}, \\ |x| &> 7131.6 \end{aligned}$$

Propagation of arithmetic operations errors

Let ω denote arithmetic operation such as $+, -, *$,or $/$.

Propagation of arithmetic operations errors

Let ω denote arithmetic operation such as $+, -, *$,or $/$.

Let ω^* denote the same arithmetic operation as it is actually carried out in the computer, including rounding or chopping error.

Propagation of arithmetic operations errors

Let ω denote arithmetic operation such as $+, -, *$,or $/$.

Let ω^* denote the same arithmetic operation as it is actually carried out in the computer, including rounding or chopping error.

Let $x_A \approx x_T$ and $y_A \approx y_T$.

Propagation of arithmetic operations errors

Let ω denote arithmetic operation such as $+, -, *$,or $/$.

Let ω^* denote the same arithmetic operation as it is actually carried out in the computer, including rounding or chopping error.

Let $x_A \approx x_T$ and $y_A \approx y_T$.

We want to obtain $x_T \, \omega \, y_T$, but we actually obtain $x_A \, \omega^* \, y_A$.

Propagation of arithmetic operations errors

Let ω denote arithmetic operation such as $+, -, *$,or $/$.

Let ω^* denote the same arithmetic operation as it is actually carried out in the computer, including rounding or chopping error.

Let $x_A \approx x_T$ and $y_A \approx y_T$.

We want to obtain $x_T \, \omega \, y_T$, but we actually obtain $x_A \, \omega^* \, y_A$.

The error in $x_A \, \omega^* \, y_A$ is given by $x_T \, \omega \, y_T - x_A \, \omega^* \, y_A$.

Propagation of arithmetic operations errors

Let ω denote arithmetic operation such as $+, -, *$,or $/$.

Let ω^* denote the same arithmetic operation as it is actually carried out in the computer, including rounding or chopping error.

Let $x_A \approx x_T$ and $y_A \approx y_T$.

We want to obtain $x_T \, \omega \, y_T$, but we actually obtain $x_A \, \omega^* \, y_A$.

The error in $x_A \, \omega^* \, y_A$ is given by $x_T \, \omega \, y_T - x_A \, \omega^* \, y_A$.
The error in $x_A \, \omega^* \, y_A$ can be rewritten as:

$$x_T \, \omega \, y_T - x_A \, \omega^* \, y_A = \left[x_T \, \omega \, y_T - x_A \, \omega \, y_A\right] + \left[x_A \, \omega \, y_A - x_A \, \omega^* \, y_A\right]$$

The last term is the error introduced by the inexactness of the machine arithmetic, since it can be shown that

$$\mathrm{Rel}(x_A \, \omega^* \, y_A) = -\varepsilon.$$

Propagation of arithmetic operations errors

With rounded binary arithmetic having n digits in the mantissa,

$$-2^{-n} \leq \varepsilon \leq 2^{-n}.$$

Propagation of arithmetic operations errors

With rounded binary arithmetic having n digits in the mantissa,

$$-2^{-n} \leq \varepsilon \leq 2^{-n}.$$

Coming back to error formula we have

$$x_T\,\omega\,y_T - x_A\,\omega^*\,y_A = \left[x_T\,\omega\,y_T - x_A\,\omega\,y_A\right] + \underbrace{\left[x_A\,\omega\,y_A - x_A\,\omega^*\,y_A\right]}_{\text{Relative error is } -\varepsilon}.$$

Propagation of arithmetic operations errors

With rounded binary arithmetic having n digits in the mantissa,

$$-2^{-n} \leq \varepsilon \leq 2^{-n}.$$

Coming back to error formula we have

$$x_T\,\omega\,y_T - x_A\,\omega^*\,y_A = \left[x_T\,\omega\,y_T - x_A\,\omega\,y_A\right] + \underbrace{\left[x_A\,\omega\,y_A - x_A\,\omega^*\,y_A\right]}_{\text{Relative error is } -\varepsilon}.$$

The first term from the right side

$$x_T\,\omega\,y_T - x_A\,\omega\,y_A$$

is called the **propagated error**.

Propagation of arithmetic operations errors

With rounded binary arithmetic having n digits in the mantissa,

$$-2^{-n} \leq \varepsilon \leq 2^{-n}.$$

Coming back to error formula we have

$$x_T\,\omega\,y_T - x_A\,\omega^*\,y_A = \left[x_T\,\omega\,y_T - x_A\,\omega\,y_A\right] + \underbrace{\left[x_A\,\omega\,y_A - x_A\,\omega^*\,y_A\right]}_{\text{Relative error is } -\varepsilon}.$$

The first term from the right side

$$x_T\,\omega\,y_T - x_A\,\omega\,y_A$$

is called the **propagated error**.

In what follows we examine it for particular cases.

Propagation of arithmetic operations errors

With rounded binary arithmetic having n digits in the mantissa,

$$-2^{-n} \leq \varepsilon \leq 2^{-n}.$$

Coming back to error formula we have

$$x_T\,\omega\,y_T - x_A\,\omega^*\,y_A = \left[x_T\,\omega\,y_T - x_A\,\omega\,y_A\right] + \underbrace{\left[x_A\,\omega\,y_A - x_A\,\omega^*\,y_A\right]}_{\text{Relative error is } -\varepsilon}.$$

The first term from the right side

$$x_T\,\omega\,y_T - x_A\,\omega\,y_A$$

is called the **propagated error**.

In what follows we examine it for particular cases.

Let $\omega = *$ (i.e. multiplication). Write

$$x_T = x_A + \xi, \qquad y_T = y_A + \eta,$$

where ξ and η are the approximation errors is x_A and y_A.

Propagation of arithmetic operations errors

Then for the relative error in $x_A\, y_A$

$$
\begin{aligned}
\mathrm{Rel}(x_A * y_A) &= \frac{x_T * y_T - x_A * y_A}{x_T * y_T} \\
&= \frac{x_T * y_T - (x_T - \xi) * (y_T - \eta)}{x_T * y_T} \\
&= \frac{x_T \eta + y_T \xi - \xi \eta}{x_T * y_T} \\
&= \frac{\xi}{x_T} + \frac{\eta}{y_T} - \frac{\xi}{x_T} \cdot \frac{\eta}{y_T} \\
&= \mathrm{Rel}(x_A) + \mathrm{Rel}(y_A) - \mathrm{Rel}(x_A) \cdot \mathrm{Rel}(y_A).
\end{aligned}
$$

Propagation of arithmetic operations errors

Then for the relative error in $x_A\, y_A$

$$\begin{aligned}
\mathrm{Rel}(x_A * y_A) &= \frac{x_T * y_T - x_A * y_A}{x_T * y_T} \\
&= \frac{x_T * y_T - (x_T - \xi) * (y_T - \eta)}{x_T * y_T} \\
&= \frac{x_T \eta + y_T \xi - \xi \eta}{x_T * y_T} \\
&= \frac{\xi}{x_T} + \frac{\eta}{y_T} - \frac{\xi}{x_T} \cdot \frac{\eta}{y_T} \\
&= \mathrm{Rel}(x_A) + \mathrm{Rel}(y_A) - \mathrm{Rel}(x_A) \cdot \mathrm{Rel}(y_A).
\end{aligned}$$

Usually we have $|\mathrm{Rel}(x_A)| \ll 1, \quad |\mathrm{Rel}(y_A)| \ll 1$. Therefore, we can skip the last term $\mathrm{Rel}(x_A) \cdot \mathrm{Rel}(\, y_A)$, since it is much smaller compared with previous two.

$$\begin{aligned}
\mathrm{Rel}(x_A y_A) &= Rel(x_A) + Rel(y_A) - \mathrm{Rel}(x_A) \cdot \mathrm{Rel}(y_A) \\
&\approx \mathrm{Rel}(x_A) + \mathrm{Rel}(y_A).
\end{aligned}$$

Propagation of arithmetic operations errors

Thus, in multiplication small relative errors in the arguments x_A and y_A lead to a small relative error (the sum of relative errors from factors) in the product $x_A * y_A$.

Propagation of arithmetic operations errors

Thus, in multiplication small relative errors in the arguments x_A and y_A lead to a small relative error (the sum of relative errors from factors) in the product $x_A * y_A$.

Also, note that there will be some cancellation, if these relative errors are of opposite sign.

Propagation of arithmetic operations errors

Thus, in multiplication small relative errors in the arguments x_A and y_A lead to a small relative error (the sum of relative errors from factors) in the product $x_A * y_A$.

Also, note that there will be some cancellation, if these relative errors are of opposite sign.

There is a similar result for division:

$$\mathrm{Rel}(x_A\, y_A) \approx \mathrm{Rel}(x_A) - \mathrm{Rel}(\, y_A)$$

provided $|\mathrm{Rel}(y_A)| \ll 1$.

Propagation of arithmetic operations errors

Thus, in multiplication small relative errors in the arguments x_A and y_A lead to a small relative error (the sum of relative errors from factors) in the product $x_A * y_A$.

Also, note that there will be some cancellation, if these relative errors are of opposite sign.

There is a similar result for division:

$$\mathrm{Rel}(x_A\, y_A) \approx \mathrm{Rel}(x_A) - \mathrm{Rel}(\, y_A)$$

provided $|\mathrm{Rel}(y_A)| \ll 1$.

For ω equal to $-$ or $+$, we have

$$[x_T \pm y_T] - [x_A \pm y_A] = [x_T - x_A] \pm [y_T - y_A].$$

Propagation of arithmetic operations errors

Thus, in multiplication small relative errors in the arguments x_A and y_A lead to a small relative error (the sum of relative errors from factors) in the product $x_A * y_A$.

Also, note that there will be some cancellation, if these relative errors are of opposite sign.

There is a similar result for division:

$$\mathrm{Rel}(x_A\, y_A) \approx \mathrm{Rel}(x_A) - \mathrm{Rel}(\, y_A)$$

provided $|\mathrm{Rel}(y_A)| \ll 1$.

For ω equal to $-$ or $+$, we have

$$[x_T \pm y_T] - [x_A \pm y_A] = [x_T - x_A] \pm [y_T - y_A].$$

Thus, **the error in a sum is the sum of the errors in the original arguments, and similarly for subtraction.**

Propagation of arithmetic operations errors

Thus, in multiplication small relative errors in the arguments x_A and y_A lead to a small relative error (the sum of relative errors from factors) in the product $x_A * y_A$.

Also, note that there will be some cancellation, if these relative errors are of opposite sign.

There is a similar result for division:

$$\mathrm{Rel}(x_A\, y_A) \approx \mathrm{Rel}(x_A) - \mathrm{Rel}(\, y_A)$$

provided $|\mathrm{Rel}(y_A)| \ll 1$.

For ω equal to $-$ or $+$, we have

$$[x_T \pm y_T] - [x_A \pm y_A] = [x_T - x_A] \pm [y_T - y_A].$$

Thus, **the error in a sum is the sum of the errors in the original arguments, and similarly for subtraction.**

However, there is a more subtle error occurring here.

Errors in function evaluations

Suppose we are evaluating a function $f(x)$ in the machine.

Errors in function evaluations

Suppose we are evaluating a function $f(x)$ in the machine.

Then, the result is generally not $f(x)$, but rather an approximate of it, which we denote by $\widetilde{f}(x)$.

Errors in function evaluations

Suppose we are evaluating a function $f(x)$ in the machine.

Then, the result is generally not $f(x)$, but rather an approximate of it, which we denote by $\widetilde{f}(x)$.

Now, suppose that we have a number $x_A \approx x_T$.

Errors in function evaluations

Suppose we are evaluating a function $f(x)$ in the machine.

Then, the result is generally not $f(x)$, but rather an approximate of it, which we denote by $\widetilde{f}(x)$.

Now, suppose that we have a number $x_A \approx x_T$.

We want to calculate $f(x_T)$, but instead we evaluate $\widetilde{f}(x_A)$.

Errors in function evaluations

Suppose we are evaluating a function $f(x)$ in the machine.

Then, the result is generally not $f(x)$, but rather an approximate of it, which we denote by $\widetilde{f}(x)$.

Now, suppose that we have a number $x_A \approx x_T$.

We want to calculate $f(x_T)$, but instead we evaluate $\widetilde{f}(x_A)$.

What can we say about the error in this latter computed quantity?

Rewrite the error

$$f(x_T) - \widetilde{f}(x_A) = \left[f(x_T) - f(x_A)\right] + \left[f(x_A) - \widetilde{f}(x_A)\right].$$

Errors in function evaluations

Suppose we are evaluating a function $f(x)$ in the machine.

Then, the result is generally not $f(x)$, but rather an approximate of it, which we denote by $\widetilde{f}(x)$.

Now, suppose that we have a number $x_A \approx x_T$.

We want to calculate $f(x_T)$, but instead we evaluate $\widetilde{f}(x_A)$.

What can we say about the error in this latter computed quantity?

Rewrite the error

$$f(x_T) - \widetilde{f}(x_A) = \left[f(x_T) - f(x_A)\right] + \left[f(x_A) - \widetilde{f}(x_A)\right].$$

The quantity $f(x_A) - \widetilde{f}(x_A)$ is the **noise** in the evaluation of $f(x_A)$ in the computer.

Errors in function evaluations

Suppose we are evaluating a function $f(x)$ in the machine.

Then, the result is generally not $f(x)$, but rather an approximate of it, which we denote by $\widetilde{f}(x)$.

Now, suppose that we have a number $x_A \approx x_T$.

We want to calculate $f(x_T)$, but instead we evaluate $\widetilde{f}(x_A)$.

What can we say about the error in this latter computed quantity?

Rewrite the error

$$f(x_T) - \widetilde{f}(x_A) = \left[f(x_T) - f(x_A)\right] + \left[f(x_A) - \widetilde{f}(x_A)\right].$$

The quantity $f(x_A) - \widetilde{f}(x_A)$ is the **noise** in the evaluation of $f(x_A)$ in the computer.

The quantity $f(x_T) - f(x_A)$ is called the **propagated error**. It is the error that results from using perfect arithmetic in the evaluation of the function.

Errors in function evaluations

If the function $f(x)$ is differentiable, then we can use the **Mean-value Theorem** from calculus to write

$$f(x_T) - f(x_A) = f'(\xi)(x_T - x_A)$$

for some ξ between x_T andx_A.

Errors in function evaluations

If the function $f(x)$ is differentiable, then we can use the **Mean-value Theorem** from calculus to write

$$f(x_T) - f(x_A) = f'(\xi)(x_T - x_A)$$

for some ξ between x_T andx_A.

Since usually x_T and x_A are close together, we can say ξ is close to either of them, and

$$\begin{aligned} f(x_T) - f(x_A) &= f'(\xi)(x_T - x_A) \\ &\approx f'(x_T)(x_T - x_A) \\ &\approx f'(x_A)(x_T - x_A). \end{aligned}$$

Errors in function evaluations

If the function $f(x)$ is differentiable, then we can use the **Mean-value Theorem** from calculus to write

$$f(x_T) - f(x_A) = f'(\xi)(x_T - x_A)$$

for some ξ between x_T andx_A.

Since usually x_T and x_A are close together, we can say ξ is close to either of them, and

$$\begin{aligned} f(x_T) - f(x_A) &= f'(\xi)(x_T - x_A) \\ &\approx f'(x_T)(x_T - x_A) \\ &\approx f'(x_A)(x_T - x_A). \end{aligned}$$

This last approximation can be used in practice to estimate the propagated error once function f and its derivative are known.

Errors in function evaluations. Example

Example. Define $f(x) = b^x$, where b is a positive real number. Then, last formula yields

$$b^{x_T} - b^{x_A} \approx (\ln b) b^{x_T} (x_T - x_A).$$

Errors in function evaluations. Example

Example. Define $f(x) = b^x$, where b is a positive real number. Then, last formula yields

$$b^{x_T} - b^{x_A} \approx (\ln b) b^{x_T} (x_T - x_A).$$

Therefore,

$$\begin{aligned}
\mathrm{Rel}\left(b^{x_A}\right) &\approx \frac{(\ln b) b^{x_T} (x_T - x_A)}{b^{x_T}} \\
&= \frac{(\ln b)(x_T - x_A) x_T}{x_T} \\
&= x_T \ln b \cdot \mathrm{Rel}(x_A) \\
&= K \cdot \mathrm{Rel}(x_A).
\end{aligned}$$

Errors in function evaluations. Example

Example. Define $f(x) = b^x$, where b is a positive real number. Then, last formula yields

$$b^{x_T} - b^{x_A} \approx (\ln b) b^{x_T} (x_T - x_A).$$

Therefore,

$$\begin{aligned}
\mathrm{Rel}\left(b^{x_A}\right) &\approx \frac{(\ln b) b^{x_T} (x_T - x_A)}{b^{x_T}} \\
&= \frac{(\ln b)(x_T - x_A) x_T}{x_T} \\
&= x_T \ln b \cdot \mathrm{Rel}(x_A) \\
&= K \cdot \mathrm{Rel}(x_A).
\end{aligned}$$

Note that if $K = 10^4$ and $\mathrm{Rel}(x_A) = 10^{-7}$, then $\mathrm{Rel}(b^{x_A}) \approx 10^{-3}$.

Errors in function evaluations. Example

Example. Define $f(x) = b^x$, where b is a positive real number. Then, last formula yields

$$b^{x_T} - b^{x_A} \approx (\ln b) b^{x_T} (x_T - x_A).$$

Therefore,

$$\begin{aligned}
\mathrm{Rel}\left(b^{x_A}\right) &\approx \frac{(\ln b) b^{x_T} (x_T - x_A)}{b^{x_T}} \\
&= \frac{(\ln b)(x_T - x_A) x_T}{x_T} \\
&= x_T \ln b \cdot \mathrm{Rel}(x_A) \\
&= K \cdot \mathrm{Rel}(x_A).
\end{aligned}$$

Note that if $K = 10^4$ and $\mathrm{Rel}(x_A) = 10^{-7}$, then $\mathrm{Rel}(b^{x_A}) \approx 10^{-3}$.

This is a large decrease in accuracy; and it is independent of how we actually calculate b^x.

Errors in function evaluations. Example

Example. Define $f(x) = b^x$, where b is a positive real number. Then, last formula yields

$$b^{x_T} - b^{x_A} \approx (\ln b) b^{x_T} (x_T - x_A).$$

Therefore,

$$\begin{aligned}
\mathrm{Rel}\left(b^{x_A}\right) &\approx \frac{(\ln b) b^{x_T} (x_T - x_A)}{b^{x_T}} \\
&= \frac{(\ln b)(x_T - x_A) x_T}{x_T} \\
&= x_T \ln b \cdot \mathrm{Rel}(x_A) \\
&= K \cdot \mathrm{Rel}(x_A).
\end{aligned}$$

Note that if $K = 10^4$ and $\mathrm{Rel}(x_A) = 10^{-7}$, then $\mathrm{Rel}(b^{x_A}) \approx 10^{-3}$.

This is a large decrease in accuracy; and it is independent of how we actually calculate b^x.

The number K is called a **condition number** for the computation.

Summation

Let S be a sum with a relatively large number of terms

$$S = a_1 + a_2 + \dots a_n, \tag{1}$$

where $\{a_j\}_{j=1}^{n}$ are floating point numbers.

Summation

Let S be a sum with a relatively large number of terms

$$S = a_1 + a_2 + \ldots a_n, \tag{1}$$

where $\{a_j\}_{j=1}^{n}$ are floating point numbers. The summation process in (1) consists of $n-1$ consecutive additions:

$$S = (((\ldots(a_1 + a_2) + a_3) + \ldots + a_{n-1}) + a_n.$$

Summation

Let S be a sum with a relatively large number of terms

$$S = a_1 + a_2 + \dots a_n, \tag{1}$$

where $\{a_j\}_{j=1}^n$ are floating point numbers. The summation process in (1) consists of $n-1$ consecutive additions:

$$S = (((\dots(a_1 + a_2) + a_3) + \dots + a_{n-1}) + a_n.$$

Define

$$\begin{aligned} S_2 &= fl(a_1 + a_2), \\ S_3 &= fl(S_2 + a_3), \\ S_4 &= fl(S_3 + a_4), \\ &\vdots \\ S_n &= fl(S_{n-1} + a_n). \end{aligned}$$

Recall the formula

$$fl(x) = x(1 + \varepsilon).$$

Summation

$$
\begin{aligned}
S_2 &= (a_1 + a_2)(1 + \varepsilon_2), \\
S_3 &= (S_2 + a_3)(1 + \varepsilon_3), \\
S_4 &= (S_3 + a_4)(1 + \varepsilon_4), \\
&\vdots \\
S_n &= (S_{n-1} + a_n)(1 + \varepsilon_n).
\end{aligned}
$$

Summation

$$
\begin{aligned}
S_2 &= (a_1 + a_2)(1 + \varepsilon_2), \\
S_3 &= (S_2 + a_3)(1 + \varepsilon_3), \\
S_4 &= (S_3 + a_4)(1 + \varepsilon_4), \\
&\vdots \\
S_n &= (S_{n-1} + a_n)(1 + \varepsilon_n).
\end{aligned}
$$

Then,

$$
\begin{aligned}
S_3 &= (S_2 + a_3)(1 + \varepsilon_3), \\
&= ((a_1 + a_2)(1 + \varepsilon_2) + a_3)(1 + \varepsilon_3), \\
&\approx (a_1 + a_2 + a_3) + a_1(\varepsilon_2 + \varepsilon_3), \\
&+ a_2(\varepsilon_2 + \varepsilon_3) + a_3\varepsilon_3, \\
S_4 &\approx (a_1 + a_2 + a_3 + a_4) + a_1(\varepsilon_2 + \varepsilon_3 + \varepsilon_4) \\
&+ a_2(\varepsilon_2 + \varepsilon_3 + \varepsilon_4) + a_3(\varepsilon_3 + \varepsilon_4) + a_4\varepsilon_4.
\end{aligned}
$$

Summation

And finally we get

$$\begin{aligned} S_n \approx &(a_1 + a_2 + \ldots + a_n) + a_1(\varepsilon_2 + \ldots + \varepsilon_n) \\ &+ a_2(\varepsilon_2 + \ldots + \varepsilon_n) + a_3(\varepsilon_3 + \ldots + \varepsilon_n) \\ &+ a_4(\varepsilon_4 + \ldots + \varepsilon_n) + \ldots + a_n\varepsilon_n. \end{aligned}$$

Summation

And finally we get

$$\begin{aligned} S_n \approx (a_1 + a_2 + \ldots + a_n) &+ a_1(\varepsilon_2 + \ldots + \varepsilon_n) \\ &+ a_2(\varepsilon_2 + \ldots + \varepsilon_n) + a_3(\varepsilon_3 + \ldots + \varepsilon_n) \\ &+ a_4(\varepsilon_4 + \ldots + \varepsilon_n) + \ldots + a_n\varepsilon_n. \end{aligned}$$

We are interested in the error $S - S_n$:

$$\begin{aligned} S - S_n \approx -a_1(\varepsilon_2 + \ldots + \varepsilon_n) &- a_2(\varepsilon_2 + \ldots + \varepsilon_n) - a_3(\varepsilon_3 + \ldots + \varepsilon_n) \\ &- a_4(\varepsilon_4 + \ldots + \varepsilon_n) - \ldots - a_n\varepsilon_n. \end{aligned}$$

Summation

And finally we get

$$\begin{aligned} S_n \approx (a_1 + a_2 + \ldots + a_n) &+ a_1(\varepsilon_2 + \ldots + \varepsilon_n) \\ &+ a_2(\varepsilon_2 + \ldots + \varepsilon_n) + a_3(\varepsilon_3 + \ldots + \varepsilon_n) \\ &+ a_4(\varepsilon_4 + \ldots + \varepsilon_n) + \ldots + a_n\varepsilon_n. \end{aligned}$$

We are interested in the error $S - S_n$:

$$\begin{aligned} S - S_n \approx &-a_1(\varepsilon_2 + \ldots + \varepsilon_n) - a_2(\varepsilon_2 + \ldots + \varepsilon_n) - a_3(\varepsilon_3 + \ldots + \varepsilon_n) \\ &- a_4(\varepsilon_4 + \ldots + \varepsilon_n) - \ldots - a_n\varepsilon_n. \end{aligned}$$

Using the last relation, in order to minimize the error $S - S_n$, we can establish the strategy for summation:
initially rearrange the terms in increasing order:

$$|a_1| \leq |a_2| \leq |a_3| \leq \ldots \leq |a_n|.$$

Summation

And finally we get

$$\begin{aligned} S_n \approx (a_1 + a_2 + \ldots + a_n) &+ a_1(\varepsilon_2 + \ldots + \varepsilon_n) \\ &+ a_2(\varepsilon_2 + \ldots + \varepsilon_n) + a_3(\varepsilon_3 + \ldots + \varepsilon_n) \\ &+ a_4(\varepsilon_4 + \ldots + \varepsilon_n) + \ldots + a_n\varepsilon_n. \end{aligned}$$

We are interested in the error $S - S_n$:

$$\begin{aligned} S - S_n \approx &-a_1(\varepsilon_2 + \ldots + \varepsilon_n) - a_2(\varepsilon_2 + \ldots + \varepsilon_n) - a_3(\varepsilon_3 + \ldots + \varepsilon_n) \\ &- a_4(\varepsilon_4 + \ldots + \varepsilon_n) - \ldots - a_n\varepsilon_n. \end{aligned}$$

Using the last relation, in order to minimize the error $S - S_n$, we can establish the strategy for summation:
initially rearrange the terms in increasing order:

$$|a_1| \leq |a_2| \leq |a_3| \leq \ldots \leq |a_n|.$$

In this rearrangement, smaller numbers a_1 and a_2 will be multiplied with larger numbers $\varepsilon_2 + \ldots + \varepsilon_n$, and a larger number a_n will be multiplied with a smaller number ε_n.

Summation. Example with chopping

Four digit calculator with chopping is being used to compute the following sum:

$$S = \sum_{i=1}^{n} \frac{1}{i} = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}.$$

Number of terms, n	Exact value	SL	Error	LS	Error
10	2.929	2.928	0.001	2.927	0.002
25	3.816	3.813	0.003	3.806	0.010
50	4.499	4.491	0.008	4.470	0.020
100	5.187	5.170	0.017	5.142	0.045
200	5.878	5.841	0.037	5.786	0.092
500	6.793	6.692	0.101	6.569	0.224
1000	7.486	7.284	0.202	7.069	0.417

SL: *smallest to largest* strategy; LS: *largest to smallest* strategy.

Summation. Example with rounding

Four digit calculator with rounding is being used to compute the following sum:

$$S = \sum_{i=1}^{n} \frac{1}{i} = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}.$$

Number of terms, n	Exact value	SL	Error	LS	Error
10	2.929	2.929	0	2.929	0
25	3.816	3.816	0	3.817	−0.001
50	4.499	4.500	−0.001	4.498	0.001
100	5.187	5.187	0	5.187	0
200	5.878	5.878	0	5.876	0.002
500	6.793	6.794	−0.001	6.783	0.010
1000	7.486	7.486	0	7.449	0.037

SL: *smallest-to-largest* strategy; LS: *largest-to-smallest* strategy.

Summation

Conclusions:

1. *Smallest-to-largest* strategy is more preferable than the *largest-to-smallest*, since the error in the first strategy is at least twice as small.

Summation

Conclusions:

1. *Smallest-to-largest* strategy is more preferable than the *largest-to-smallest*, since the error in the first strategy is at least twice as small.
2. Rounding is much better that chopping. Just compare the errors in the first and second tables.

Summation

Conclusions:

1. *Smallest-to-largest* strategy is more preferable than the *largest-to-smallest*, since the error in the first strategy is at least twice as small.
2. Rounding is much better that chopping. Just compare the errors in the first and second tables.
3. Observe that in the rounding case, some of the errors are 0, since rounding errors can be either positive or negative, and thus they might cancel each other.

Special Mathematics 2

Numerical Analysis / Numerical Methods

Spring 2022 Lecture 2

Rootfinding

Want to find the numbers x for which

$$f(x) = 0$$

with $f : [a, b] \to \mathbb{R}$ a given real-valued function.

Here, we denote such roots or zeroes by the Greek letter α. So

$$f(\alpha) = 0.$$

Rootfinding

Want to find the numbers x for which

$$f(x) = 0$$

with $f : [a, b] \to \mathbb{R}$ a given real-valued function.

Here, we denote such roots or zeroes by the Greek letter α. So

$$f(\alpha) = 0.$$

Rootfinding problems occur in many contexts.
Sometimes they are a direct formulation of some physical situation, but more often, they are an intermediate step in solving a much larger problem.

Bisection method

Most methods for solving $f(x) = 0$ are **iterative methods**.

This means that such a method given an initial guess x_0 will provide us with a sequence of consecutively computed solutions $x_1, x_2, x_3, x_4, \ldots, x_n, x_{n+1}, \ldots$ such that

$$x_n \to \alpha \text{ as } n \to \infty.$$

Bisection method

Most methods for solving $f(x) = 0$ are **iterative methods**.

This means that such a method given an initial guess x_0 will provide us with a sequence of consecutively computed solutions $x_1, x_2, x_3, x_4, \ldots, x_n, x_{n+1}, \ldots$ such that

$$x_n \to \alpha \text{ as } n \to \infty.$$

We begin with the simplest of such methods, **Bisection method**.

Bisection method

Most methods for solving $f(x) = 0$ are **iterative methods**.

This means that such a method given an initial guess x_0 will provide us with a sequence of consecutively computed solutions $x_1, x_2, x_3, x_4, \ldots, x_n, x_{n+1}, \ldots$ such that

$$x_n \to \alpha \text{ as } n \to \infty.$$

We begin with the simplest of such methods, **Bisection method**.

Suppose we are given a function $f(x)$ and we assume we have an interval $[a, b]$ containing the root, on which the function is continuous.

Bisection method

Most methods for solving $f(x) = 0$ are **iterative methods**.

This means that such a method given an initial guess x_0 will provide us with a sequence of consecutively computed solutions $x_1, x_2, x_3, x_4, \ldots, x_n, x_{n+1}, \ldots$ such that

$$x_n \to \alpha \text{ as } n \to \infty.$$

We begin with the simplest of such methods, **Bisection method**.

Suppose we are given a function $f(x)$ and we assume we have an interval $[a, b]$ containing the root, on which the function is continuous.

We also assume we are given an error tolerance $\varepsilon > 0$, and we want an approximated root $\widetilde{\alpha} \in [a, b]$ for which

$$|\alpha - \widetilde{\alpha}| < \varepsilon.$$

Bisection method

Bisection method is based on the following theorem:

Bisection method

Bisection method is based on the following theorem:

Theorem

If $f : [a, b] \to \mathbb{R}$ is a continuous function on closed and bounded interval $[a, b]$ and

$$f(a) \cdot f(b) < 0$$

then there exists $\alpha \in [a, b]$ such that $f(\alpha) = 0$.

Bisection method

Bisection method is based on the following theorem:

Theorem

If $f : [a, b] \to \mathbb{R}$ is a continuous function on closed and bounded interval $[a, b]$ and

$$f(a) \cdot f(b) < 0$$

then there exists $\alpha \in [a, b]$ such that $f(\alpha) = 0$.

Therefore, in order for Bisection Method to "work" we need to assume that the function $f(x)$ changes sign on interval $[a, b]$.

Bisection method

Bisection Algorithm: Bisect(f, a, b, ε)

Bisection method

Bisection Algorithm: Bisect(f, a, b, ε)

Step 1: Define

$$c = \frac{a+b}{2}$$

Bisection method

Bisection Algorithm: Bisect(f, a, b, ε)

Step 1: Define

$$c = \frac{a+b}{2}$$

Step 2: If $b - c \leq \varepsilon$, accept c as our root, and then stop.

Bisection method

Bisection Algorithm: Bisect(f, a, b, ε)

Step 1: Define

$$c = \frac{a+b}{2}$$

Step 2: If $b - c \leq \varepsilon$, accept c as our root, and then stop.
Step 3: If $b - c > \varepsilon$, then compare the sign of $f(c)$ to that of $f(a)$ and $f(b)$. If

$$sign(f(a)) \cdot sign(f(b)) \leqslant 0$$

then replace a with c; Otherwise, replace b with c.

Bisection method

Bisection Algorithm: Bisect(f, a, b, ε)

Step 1: Define

$$c = \frac{a+b}{2}$$

Step 2: If $b - c \leq \varepsilon$, accept c as our root, and then stop.
Step 3: If $b - c > \varepsilon$, then compare the sign of $f(c)$ to that of $f(a)$ and $f(b)$. If

$$sign(f(a)) \cdot sign(f(b)) \leqslant 0$$

then replace a with c; Otherwise, replace b with c.
Step 4: Return to Step 1.

Bisection method

Bisection Algorithm: Bisect(f, a, b, ε)

Step 1: Define

$$c = \frac{a+b}{2}$$

Step 2: If $b - c \leq \varepsilon$, accept c as our root, and then stop.
Step 3: If $b - c > \varepsilon$, then compare the sign of $f(c)$ to that of $f(a)$ and $f(b)$. If

$$sign(f(a)) \cdot sign(f(b)) \leqslant 0$$

then replace a with c; Otherwise, replace b with c.
Step 4: Return to Step 1.

Note that, it is preferable to check the sign using condition $sign(f(a)) \cdot sign(f(b)) \leqslant 0$ instead of using $sign(f(a) \cdot f(b)) \leqslant 0$. Why?

Bisection method

Bisection method

Example

Consider the function

$$f(x) = x^6 - x - 1.$$

We want to find the largest root with accuracy of $\varepsilon = 0.001$.

It can be seen from the graph of this function that the root is located in interval $[1, 2]$.

Also, note that the function is continuous.

Let $a = 1$ and $b = 2$.

Since $f(a) = -1$ and $f(b) = 61$, the given function changes its sign and thus all conditions for bisection method are being satisfied. Results are presented on next page.

Bisection method

n	a_n	b_n	c_n	$f(c_n)$	$b_n - c_n$
1	1.00000	2.00000	1.50000	$8.891e+00$	$5.000e-01$
2	1.00000	1.50000	1.25000	$1.565e+00$	$2.500e-01$
3	1.00000	1.25000	1.12500	$-9.771e-02$	$1.250e-01$
4	1.12500	1.25000	1.18750	$6.167e-01$	$6.250e-02$
5	1.12500	1.18750	1.15625	$2.333e-01$	$3.125e-02$
6	1.12500	1.15625	1.14063	$6.158e-02$	$1.563e-02$
7	1.12500	1.14063	1.13281	$-1.958e-02$	$7.813e-03$
8	1.13281	1.14063	1.13672	$2.062e-02$	$3.906e-03$
9	1.13281	1.13672	1.13477	$4.268e-04$	$1.953e-03$
10	1.13281	1.13477	1.13379	$-9.598e-03$	$9.766e-04$

Error analysis for bisection method

Let a_n, b_n and c_n be the values provided by bisection method at iteration n. Evidently,

$$b_{n+1} - a_{n+1} = \frac{1}{2}(b_n - a_n).$$

Error analysis for bisection method

Let a_n, b_n and c_n be the values provided by bisection method at iteration n. Evidently,

$$b_{n+1} - a_{n+1} = \frac{1}{2}(b_n - a_n).$$

Using induction we get

$$\begin{aligned} b_n - a_n &= \frac{1}{2}(b_{n-1} - a_{n-1}), \\ &= \frac{1}{2^2}(b_{n-2} - a_{n-2}), \\ &= \dots \\ &= \frac{1}{2^{n-1}}(b-a). \end{aligned}$$

Since either $\alpha \in [a_n, c_n]$ or $\alpha \in [c_n, b_n]$, we have

$$|\alpha - c_n| \leqslant c_n - a_n = b_n - c_n = \frac{1}{2}(b_n - a_n) = \frac{1}{2^n}(b-a).$$

Error analysis for bisection method

Therefore, if $\alpha \in [a, b]$ is the root, and c_n is the approximation provided by bisection method at step n, we have

$$|\alpha - c_n| \leqslant \frac{1}{2^n}(b - a).$$

This relation provides us with a stopping criterion for bisection method. Moreover, it follows that $c_n \to \alpha$ as $n \to \infty$.

Error analysis for bisection method

Therefore, if $\alpha \in [a, b]$ is the root, and c_n is the approximation provided by bisection method at step n, we have

$$|\alpha - c_n| \leqslant \frac{1}{2^n}(b - a).$$

This relation provides us with a stopping criterion for bisection method. Moreover, it follows that $c_n \to \alpha$ as $n \to \infty$.
Suppose we want to estimate the number of iterations in bisection method necessary to find the root with an error tolerance ε,

Error analysis for bisection method

Therefore, if $\alpha \in [a, b]$ is the root, and c_n is the approximation provided by bisection method at step n, we have

$$|\alpha - c_n| \leqslant \frac{1}{2^n}(b - a).$$

This relation provides us with a stopping criterion for bisection method. Moreover, it follows that $c_n \to \alpha$ as $n \to \infty$.
Suppose we want to estimate the number of iterations in bisection method necessary to find the root with an error tolerance ε,

$$\begin{aligned} |\alpha - c_n| &\leqslant \varepsilon \\ \frac{1}{2^n}(b - a) &\leqslant \varepsilon \\ n &\geqslant \frac{\ln\left(\frac{b-a}{\varepsilon}\right)}{\ln 2}. \end{aligned}$$

Error analysis for bisection method

Therefore, if $\alpha \in [a, b]$ is the root, and c_n is the approximation provided by bisection method at step n, we have

$$|\alpha - c_n| \leqslant \frac{1}{2^n}(b - a).$$

This relation provides us with a stopping criterion for bisection method. Moreover, it follows that $c_n \to \alpha$ as $n \to \infty$.
Suppose we want to estimate the number of iterations in bisection method necessary to find the root with an error tolerance ε,

$$|\alpha - c_n| \leqslant \varepsilon$$

$$\frac{1}{2^n}(b - a) \leqslant \varepsilon$$

$$n \geqslant \frac{\ln\left(\frac{b-a}{\varepsilon}\right)}{\ln 2}.$$

Thus, using the last inequality for the previous example we get

$$n \geqslant \frac{\ln\left(\frac{1}{0.001}\right)}{\ln 2} \approx 9.97.$$

Advantages and Disadvantages of Bisection method

Advantages:

1. It always converges.

Advantages and Disadvantages of Bisection method

Advantages:

1. It always converges.
2. You have a guaranteed error bound, and it decreases with each successive iteration.

Advantages and Disadvantages of Bisection method

Advantages:

1. It always converges.
2. You have a guaranteed error bound, and it decreases with each successive iteration.
3. You have a guaranteed rate of convergence. The error bound decreases by $1/2$ with each iteration.

Advantages and Disadvantages of Bisection method

Advantages:

1. It always converges.
2. You have a guaranteed error bound, and it decreases with each successive iteration.
3. You have a guaranteed rate of convergence. The error bound decreases by $1/2$ with each iteration.

Advantages and Disadvantages of Bisection method

Advantages:

1. It always converges.
2. You have a guaranteed error bound, and it decreases with each successive iteration.
3. You have a guaranteed rate of convergence. The error bound decreases by $1/2$ with each iteration.

Disadvantages:

1. It is relatively slow when compared with other rootfinding methods we will study, especially when the function $f(x)$ has several continuous derivatives about the root α.

Advantages and Disadvantages of Bisection method

Advantages:

1. It always converges.
2. You have a guaranteed error bound, and it decreases with each successive iteration.
3. You have a guaranteed rate of convergence. The error bound decreases by $1/2$ with each iteration.

Disadvantages:

1. It is relatively slow when compared with other rootfinding methods we will study, especially when the function $f(x)$ has several continuous derivatives about the root α.
2. The algorithm has no check to see whether the ε is too small for the computer arithmetic being used.

Advantages and Disadvantages of Bisection method

Advantages:

1. It always converges.
2. You have a guaranteed error bound, and it decreases with each successive iteration.
3. You have a guaranteed rate of convergence. The error bound decreases by $1/2$ with each iteration.

Disadvantages:

1. It is relatively slow when compared with other rootfinding methods we will study, especially when the function $f(x)$ has several continuous derivatives about the root α.
2. The algorithm has no check to see whether the ε is too small for the computer arithmetic being used.

Advantages and Disadvantages of Bisection method

Advantages:

1. It always converges.
2. You have a guaranteed error bound, and it decreases with each successive iteration.
3. You have a guaranteed rate of convergence. The error bound decreases by $1/2$ with each iteration.

Disadvantages:

1. It is relatively slow when compared with other rootfinding methods we will study, especially when the function $f(x)$ has several continuous derivatives about the root α.
2. The algorithm has no check to see whether the ε is too small for the computer arithmetic being used.

We also need to assume that the function $f(x)$ is continuous on the given interval $[a, b]$; but there is no way for the computer to confirm this.

Rootfinding

Want to find the root α of a given function $f(x)$.

In other words, we want to find the point x at which the graph of $y = f(x)$ intersects the x-axis.

Rootfinding

Want to find the root α of a given function $f(x)$.

In other words, we want to find the point x at which the graph of $y = f(x)$ intersects the x-axis.

One of the principles of numerical analysis is the following:

Numerical Analysis Principle

If you cannot solve the given problem, then solve a "nearby problem".

Rootfinding

Want to find the root α of a given function $f(x)$.

In other words, we want to find the point x at which the graph of $y = f(x)$ intersects the x-axis.

One of the principles of numerical analysis is the following:

Numerical Analysis Principle

If you cannot solve the given problem, then solve a "nearby problem".

How do we obtain a nearby problem for $f(x) = 0$?

Rootfinding

Want to find the root α of a given function $f(x)$.

In other words, we want to find the point x at which the graph of $y = f(x)$ intersects the x-axis.

One of the principles of numerical analysis is the following:

Numerical Analysis Principle

If you cannot solve the given problem, then solve a "nearby problem".

How do we obtain a nearby problem for $f(x) = 0$?

Begin first by asking for types of problems which we can solve easily. At the top of the list should be that of finding where a straight line intersects the x-axis.

Rootfinding

Want to find the root α of a given function $f(x)$.

In other words, we want to find the point x at which the graph of $y = f(x)$ intersects the x-axis.

One of the principles of numerical analysis is the following:

Numerical Analysis Principle

If you cannot solve the given problem, then solve a "nearby problem".

How do we obtain a nearby problem for $f(x) = 0$?

Begin first by asking for types of problems which we can solve easily. At the top of the list should be that of finding where a straight line intersects the x-axis.

Thus, we seek to replace $f(x) = 0$ by that of solving $p(x) = 0$ for some linear polynomial $p(x)$ that approximates $f(x)$ in the vicinity of the root α.

Rootfinding

Newton's method

Let x_0 be an initial guess, sufficiently closed to the root α.

Newton's method

Let x_0 be an initial guess, sufficiently closed to the root α.
Consider the tangent line to the graph of $f(x)$ in $(x_0, f(x_0))$.

Newton's method

Let x_0 be an initial guess, sufficiently closed to the root α.
Consider the tangent line to the graph of $f(x)$ in $(x_0, f(x_0))$.
Tangent intersects x-axis at x_1, a closer point to α.
Tangent has equation

$$p_1(x) = f(x_0) + f'(x_0)(x - x_0).$$

Newton's method

Let x_0 be an initial guess, sufficiently closed to the root α.
Consider the tangent line to the graph of $f(x)$ in $(x_0, f(x_0))$.
Tangent intersects x-axis at x_1, a closer point to α.
Tangent has equation

$$p_1(x) = f(x_0) + f'(x_0)(x - x_0).$$

Since $p_1(x_1) = 0$, we get

$$f(x_0) + f'(x_0)(x_1 - x_0) = 0.$$

Newton's method

Let x_0 be an initial guess, sufficiently closed to the root α.
Consider the tangent line to the graph of $f(x)$ in $(x_0, f(x_0))$.
Tangent intersects x-axis at x_1, a closer point to α.
Tangent has equation

$$p_1(x) = f(x_0) + f'(x_0)(x - x_0).$$

Since $p_1(x_1) = 0$, we get

$$f(x_0) + f'(x_0)(x_1 - x_0) = 0.$$

Solve for x_1

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

Newton's method

Let x_0 be an initial guess, sufficiently closed to the root α.
Consider the tangent line to the graph of $f(x)$ in $(x_0, f(x_0))$.
Tangent intersects x-axis at x_1, a closer point to α.
Tangent has equation

$$p_1(x) = f(x_0) + f'(x_0)(x - x_0).$$

Since $p_1(x_1) = 0$, we get

$$f(x_0) + f'(x_0)(x_1 - x_0) = 0.$$

Solve for x_1

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

Similarly, we could get x_2,

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}.$$

Newton's method

Repeat this process to obtain the sequence $x_1, x_2, x_3, \ldots$ that hopefully will converge to root α.

Newton's method

Repeat this process to obtain the sequence $x_1, x_2, x_3, \ldots$ that hopefully will converge to root α.

General scheme for Newton's method

Starting with initial guess x_0, compute iterated sequence:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \quad n = 0, 1, 2, \ldots$$

Newton's method

Repeat this process to obtain the sequence $x_1, x_2, x_3, \ldots$ that hopefully will converge to root α.

General scheme for Newton's method

Starting with initial guess x_0, compute iterated sequence:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \quad n = 0, 1, 2, \ldots$$

Example

Apply Newton's method to function $f(x) = x^6 - x - 1$. Substitute $f(x)$ and its derivative $f'(x) = 6x^5 - 1$ in Newton's method formula to get

$$x_{n+1} = x_n - \frac{x_n^6 - x_n - 1}{6x_n^5 - 1}, \quad \forall n \in \mathbb{N}.$$

Newton's method

Use initial guess $x_0 = 1.5$.

n	x_n	$f(x_n)$	$x_n - x_{n-1}$	$\alpha - x_n$
0	1.50000000	$8.89e+1$		
1	1.30049088	$2.54e+1$	$-2.00e-1$	$-3.65e-1$
2	1.18148042	$5.38e-1$	$-1.19e-1$	$-1.66e-1$
3	1.13945559	$4.92e-2$	$-4.20e-2$	$-4.68e-2$
4	1.13477763	$5.50e-4$	$-4.68e-3$	$-4.73e-3$
5	1.13472415	$7.11e-8$	$-5.35e-5$	$-5.35e-5$
6	1.13472414	$1.55e-15$	$-6.91e-9$	$-6.91e-9$

True solution is $\alpha = 1.134724138$.

Newton's method

Use initial guess $x_0 = 1.5$.

n	x_n	$f(x_n)$	$x_n - x_{n-1}$	$\alpha - x_n$
0	1.50000000	$8.89e+1$		
1	1.30049088	$2.54e+1$	$-2.00e-1$	$-3.65e-1$
2	1.18148042	$5.38e-1$	$-1.19e-1$	$-1.66e-1$
3	1.13945559	$4.92e-2$	$-4.20e-2$	$-4.68e-2$
4	1.13477763	$5.50e-4$	$-4.68e-3$	$-4.73e-3$
5	1.13472415	$7.11e-8$	$-5.35e-5$	$-5.35e-5$
6	1.13472414	$1.55e-15$	$-6.91e-9$	$-6.91e-9$

True solution is $\alpha = 1.134724138$.

Notice that

$$x_n - x_{n-1} \approx \alpha - x_n.$$

Newton's method

Use initial guess $x_0 = 1.5$.

n	x_n	$f(x_n)$	$x_n - x_{n-1}$	$\alpha - x_n$
0	1.50000000	$8.89e+1$		
1	1.30049088	$2.54e+1$	$-2.00e-1$	$-3.65e-1$
2	1.18148042	$5.38e-1$	$-1.19e-1$	$-1.66e-1$
3	1.13945559	$4.92e-2$	$-4.20e-2$	$-4.68e-2$
4	1.13477763	$5.50e-4$	$-4.68e-3$	$-4.73e-3$
5	1.13472415	$7.11e-8$	$-5.35e-5$	$-5.35e-5$
6	1.13472414	$1.55e-15$	$-6.91e-9$	$-6.91e-9$

True solution is $\alpha = 1.134724138$.

Notice that

$$x_n - x_{n-1} \approx \alpha - x_n.$$

Thus, $x_n - x_{n-1}$ can be used as estimate of the true error and therefore as a stopping criterion in Newton's iterations.

Newton's method. Division example

Here we consider a division algorithm (based on Newton's method) implemented in some computers in the past.

Newton's method. Division example

Here we consider a division algorithm (based on Newton's method) implemented in some computers in the past.

Say, we are interested in computing $\frac{a}{b} = a \cdot \frac{1}{b}$, where $\frac{1}{b}$ is computed using Newton's method.

$$f(x) \equiv b - \frac{1}{x} = 0,$$

with b positive. The root of this equation is: $\alpha = \frac{1}{b}$.

Newton's method. Division example

Here we consider a division algorithm (based on Newton's method) implemented in some computers in the past.

Say, we are interested in computing $\frac{a}{b} = a \cdot \frac{1}{b}$, where $\frac{1}{b}$ is computed using Newton's method.

$$f(x) \equiv b - \frac{1}{x} = 0,$$

with b positive. The root of this equation is: $\alpha = \frac{1}{b}$.

$$f'(x) = \frac{1}{x^2}$$

Newton's method. Division example

Here we consider a division algorithm (based on Newton's method) implemented in some computers in the past.

Say, we are interested in computing $\frac{a}{b} = a \cdot \frac{1}{b}$, where $\frac{1}{b}$ is computed using Newton's method.

$$f(x) \equiv b - \frac{1}{x} = 0,$$

with b positive. The root of this equation is: $\alpha = \frac{1}{b}$.

$$f'(x) = \frac{1}{x^2}$$

and Newton's method for this problem becomes

$$x_{n+1} = x_n - \frac{b - \frac{1}{x_n}}{\frac{1}{x_n^2}}.$$

Newton's method. Division example

Here we consider a division algorithm (based on Newton's method) implemented in some computers in the past.

Say, we are interested in computing $\frac{a}{b} = a \cdot \frac{1}{b}$, where $\frac{1}{b}$ is computed using Newton's method.

$$f(x) \equiv b - \frac{1}{x} = 0,$$

with b positive. The root of this equation is: $\alpha = \frac{1}{b}$.

$$f'(x) = \frac{1}{x^2}$$

and Newton's method for this problem becomes

$$x_{n+1} = x_n - \frac{b - \frac{1}{x_n}}{\frac{1}{x_n^2}}.$$

Simplifying

$$x_{n+1} = x_n(2 - bx_n), \quad n \geq 0.$$

Newton's method. Division example

Initial guess x_0 must be close enough to the true solution and of course $x_0 > 0$. Consider the error

Newton's method. Division example

Initial guess x_0 must be close enough to the true solution and of course $x_0 > 0$. Consider the error

$$\begin{aligned}
\alpha - x_{n+1} &= \frac{1}{b} - x_{n+1} \\
&= \frac{1 - bx_{n+1}}{b} \\
&= \frac{1 - bx_n(2 - bx_n)}{b} \\
&= \frac{(1 - bx_n)^2}{b}.
\end{aligned}$$

On the other hand,

$$\begin{aligned}
\mathrm{Rel}(x_{n+1}) &= \frac{\alpha - x_{n+1}}{\alpha} \\
&= 1 - bx_{n+1}.
\end{aligned}$$

Newton's method. Division example

It can be shown (try it!) that

$$\mathrm{Rel}(x_{n+1}) = (\mathrm{Rel}(x_n))^2 .$$

Newton's method. Division example

It can be shown (try it!) that

$$\mathrm{Rel}(x_{n+1}) = \left(\mathrm{Rel}(x_n)\right)^2 .$$

In order to guarantee convergence $x_n \to \alpha$, we ask

$$\left|\mathrm{Rel}(x_0)\right| < 1$$

Newton's method. Division example

It can be shown (try it!) that

$$\mathrm{Rel}(x_{n+1}) = \left(\mathrm{Rel}(x_n)\right)^2 .$$

In order to guarantee convergence $x_n \to \alpha$, we ask

$$\left|\mathrm{Rel}(x_0)\right| < 1$$

or equivalently

$$0 < x_0 < \frac{2}{b}.$$

Newton's method. Division example

It can be shown (try it!) that

$$\mathrm{Rel}(x_{n+1}) = \left(\mathrm{Rel}(x_n)\right)^2 .$$

In order to guarantee convergence $x_n \to \alpha$, we ask

$$\left|\mathrm{Rel}(x_0)\right| < 1$$

or equivalently

$$0 < x_0 < \frac{2}{b}.$$

In other words, if above condition is satisfied, then the sequence generated by Newton method for division example will converge.

Newton's method. Division example

It can be shown (try it!) that

$$\mathrm{Rel}(x_{n+1}) = \left(\mathrm{Rel}(x_n)\right)^2 .$$

In order to guarantee convergence $x_n \to \alpha$, we ask

$$|\mathrm{Rel}(x_0)| < 1$$

or equivalently

$$0 < x_0 < \frac{2}{b}.$$

In other words, if above condition is satisfied, then the sequence generated by Newton method for division example will converge.

For example, suppose that $|\mathrm{Rel}(x_0)| = 0.1$. Then

Newton's method. Division example

It can be shown (try it!) that

$$\mathrm{Rel}(x_{n+1}) = \left(\mathrm{Rel}(x_n)\right)^2 .$$

In order to guarantee convergence $x_n \to \alpha$, we ask

$$|\mathrm{Rel}(x_0)| < 1$$

or equivalently

$$0 < x_0 < \frac{2}{b}.$$

In other words, if above condition is satisfied, then the sequence generated by Newton method for division example will converge.

For example, suppose that $|\mathrm{Rel}(x_0)| = 0.1$. Then

$$\mathrm{Rel}(x_1) = 10^{-2}, \qquad \mathrm{Rel}(x_2) = 10^{-4}$$
$$\mathrm{Rel}(x_3) = 10^{-8}, \qquad \mathrm{Rel}(x_4) = 10^{-16}.$$

Newton's method. Division example

Error analysis for Newton's method

Let $f(x) \in C^2[a, b]$ and $\alpha \in [a, b]$. Also let $f'(\alpha) \neq 0$.

Error analysis for Newton's method

Let $f(x) \in C^2[a, b]$ and $\alpha \in [a, b]$. Also let $f'(\alpha) \neq 0$.
Consider Taylor formula for function $f(x)$ about point x_n:

$$f(x) = f(x_n) + (x - x_n)f'(x_n) + \frac{(x - x_n)^2}{2} f''(\xi_n),$$

where ξ_n is some point between x and x_n.

Error analysis for Newton's method

Let $f(x) \in C^2[a,b]$ and $\alpha \in [a,b]$. Also let $f'(\alpha) \neq 0$.
Consider Taylor formula for function $f(x)$ about point x_n:

$$f(x) = f(x_n) + (x - x_n)f'(x_n) + \frac{(x - x_n)^2}{2} f''(\xi_n),$$

where ξ_n is some point between x and x_n. Take $x = \alpha$ to get:

$$f(\alpha) = f(x_n) + (\alpha - x_n)f'(x_n) + \frac{(\alpha - x_n)^2}{2} f''(\xi_n).$$

Error analysis for Newton's method

Let $f(x) \in C^2[a,b]$ and $\alpha \in [a,b]$. Also let $f'(\alpha) \neq 0$.
Consider Taylor formula for function $f(x)$ about point x_n:

$$f(x) = f(x_n) + (x - x_n)f'(x_n) + \frac{(x - x_n)^2}{2} f''(\xi_n),$$

where ξ_n is some point between x and x_n. Take $x = \alpha$ to get:

$$f(\alpha) = f(x_n) + (\alpha - x_n)f'(x_n) + \frac{(\alpha - x_n)^2}{2} f''(\xi_n).$$

Since $f(\alpha) = 0$, we have

$$0 = \frac{f(x_n)}{f'(x_n)} + (\alpha - x_n) + (\alpha - x_n)^2 \frac{f''(\xi_n)}{2f'(x_n)}.$$

Error analysis for Newton's method

Let $f(x) \in C^2[a,b]$ and $\alpha \in [a,b]$. Also let $f'(\alpha) \neq 0$.
Consider Taylor formula for function $f(x)$ about point x_n:

$$f(x) = f(x_n) + (x - x_n)f'(x_n) + \frac{(x - x_n)^2}{2} f''(\xi_n),$$

where ξ_n is some point between x and x_n. Take $x = \alpha$ to get:

$$f(\alpha) = f(x_n) + (\alpha - x_n)f'(x_n) + \frac{(\alpha - x_n)^2}{2} f''(\xi_n).$$

Since $f(\alpha) = 0$, we have

$$0 = \frac{f(x_n)}{f'(x_n)} + (\alpha - x_n) + (\alpha - x_n)^2 \frac{f''(\xi_n)}{2f'(x_n)}.$$

$$\alpha - x_{n+1} = (\alpha - x_n)^2 \left[\frac{-f''(\xi_n)}{2f'(x_n)} \right].$$

Error analysis for Newton's method

For previous example, $f''(x) = 30x^4$, and we get

$$\frac{-f''(\xi_n)}{2f'(x_n)} \approx \frac{-f''(\alpha)}{2f'(\alpha)} = \frac{-30\alpha^4}{2(6\alpha^5 - 1)} \approx -2.42.$$

Error analysis for Newton's method

For previous example, $f''(x) = 30x^4$, and we get

$$\frac{-f''(\xi_n)}{2f'(x_n)} \approx \frac{-f''(\alpha)}{2f'(\alpha)} = \frac{-30\alpha^4}{2(6\alpha^5 - 1)} \approx -2.42.$$

Therefore, we could write that

$$\alpha - x_{n+1} \approx -2.42(\alpha - x_n)^2.$$

Error analysis for Newton's method

For previous example, $f''(x) = 30x^4$, and we get

$$\frac{-f''(\xi_n)}{2f'(x_n)} \approx \frac{-f''(\alpha)}{2f'(\alpha)} = \frac{-30\alpha^4}{2(6\alpha^5 - 1)} \approx -2.42.$$

Therefore, we could write that

$$\alpha - x_{n+1} \approx -2.42(\alpha - x_n)^2.$$

For example, if $n = 3$, we get $\alpha - x_3 \approx -4.73e - 03$ and using the above estimation we obtain

$$\alpha - x_4 \approx -2.42(\alpha - x_3)^2 \approx -5.42e - 05,$$

a result which is in accordance with the result from the table:

$$\alpha - x_4 \approx -5.35e - 05.$$

Error analysis for Newton's method

If iteration x_n is close to α, we could denote

$$\frac{-f''(\xi_n)}{2f'(x_n)} \approx \frac{-f''(\alpha)}{2f'(\alpha)} \equiv M.$$

Error analysis for Newton's method

If iteration x_n is close to α, we could denote

$$\frac{-f''(\xi_n)}{2f'(x_n)} \approx \frac{-f''(\alpha)}{2f'(\alpha)} \equiv M.$$

$$\alpha - x_{n+1} \approx M(\alpha - x_n)^2.$$

Error analysis for Newton's method

If iteration x_n is close to α, we could denote

$$\frac{-f''(\xi_n)}{2f'(x_n)} \approx \frac{-f''(\alpha)}{2f'(\alpha)} \equiv M.$$

$$\alpha - x_{n+1} \approx M(\alpha - x_n)^2.$$

$$M(\alpha - x_{n+1}) \approx \left(M(\alpha - x_n)\right)^2.$$

Error analysis for Newton's method

If iteration x_n is close to α, we could denote

$$\frac{-f''(\xi_n)}{2f'(x_n)} \approx \frac{-f''(\alpha)}{2f'(\alpha)} \equiv M.$$

$$\alpha - x_{n+1} \approx M(\alpha - x_n)^2.$$

$$M(\alpha - x_{n+1}) \approx \left(M(\alpha - x_n)\right)^2.$$

Inductively,

$$M(\alpha - x_{n+1}) \approx \left(M(\alpha - x_0)\right)^{2^n}, \quad \forall n \in \mathbb{N}.$$

Error analysis for Newton's method

If iteration x_n is close to α, we could denote

$$\frac{-f''(\xi_n)}{2f'(x_n)} \approx \frac{-f''(\alpha)}{2f'(\alpha)} \equiv M.$$

$$\alpha - x_{n+1} \approx M(\alpha - x_n)^2.$$

$$M(\alpha - x_{n+1}) \approx \left(M(\alpha - x_n)\right)^2.$$

Inductively,

$$M(\alpha - x_{n+1}) \approx \left(M(\alpha - x_0)\right)^{2^n}, \quad \forall n \in \mathbb{N}.$$

In other words, in order to guarantee the convergence of Newton's method we should have

$$|M(\alpha - x_0)| < 1,$$

$$|\alpha - x_0| < \frac{1}{|M|} = \left|\frac{2f'(\alpha)}{f''(\alpha)}\right|.$$

Newton's method advantages vs disadvantages

Advantages of Newton's method:

1. Fast convergence (usually quadratic convergence).

Newton's method advantages vs disadvantages

Advantages of Newton's method:

1. Fast convergence (usually quadratic convergence).
2. There is a stopping criterion.

Newton's method advantages vs disadvantages

Advantages of Newton's method:

1. Fast convergence (usually quadratic convergence).
2. There is a stopping criterion.

Newton's method advantages vs disadvantages

Advantages of Newton's method:

1. Fast convergence (usually quadratic convergence).
2. There is a stopping criterion.

Disadvantages of Newton's method:

1. It may fail to converge (for example, when initial guess is not close enough).

Newton's method advantages vs disadvantages

Advantages of Newton's method:

1. Fast convergence (usually quadratic convergence).
2. There is a stopping criterion.

Disadvantages of Newton's method:

1. It may fail to converge (for example, when initial guess is not close enough).
2. It needs also the derivative of the function.

Newton's method advantages vs disadvantages

Advantages of Newton's method:

1. Fast convergence (usually quadratic convergence).
2. There is a stopping criterion.

Disadvantages of Newton's method:

1. It may fail to converge (for example, when initial guess is not close enough).
2. It needs also the derivative of the function.
3. There is no guaranteed error bound for the computed iterates.

Newton's method advantages vs disadvantages

Advantages of Newton's method:

1. Fast convergence (usually quadratic convergence).
2. There is a stopping criterion.

Disadvantages of Newton's method:

1. It may fail to converge (for example, when initial guess is not close enough).
2. It needs also the derivative of the function.
3. There is no guaranteed error bound for the computed iterates.
4. It is likely to have difficulty if $f'(\alpha) = 0$. This means the $x-$axis is tangent to the graph of $y = f(x)$ at $x = \alpha$.

Secant method

Newton's method was based on using the line tangent to the curve of $y = f(x)$ at point $(x_0, f(x_0))$.

Secant method

Newton's method was based on using the line tangent to the curve of $y = f(x)$ at point $(x_0, f(x_0))$.

When $x_0 \approx \alpha$, the graph of the tangent line is approximately the same as the graph of $y = f(x)$ around $x = \alpha$.

Secant method

Newton's method was based on using the line tangent to the curve of $y = f(x)$ at point $(x_0, f(x_0))$.

When $x_0 \approx \alpha$, the graph of the tangent line is approximately the same as the graph of $y = f(x)$ around $x = \alpha$.

Then the root of the tangent line is used to approximate α.

Secant method

Newton's method was based on using the line tangent to the curve of $y = f(x)$ at point $(x_0, f(x_0))$.

When $x_0 \approx \alpha$, the graph of the tangent line is approximately the same as the graph of $y = f(x)$ around $x = \alpha$.

Then the root of the tangent line is used to approximate α.

Consider using an approximating line based on "interpolation".

Secant method

Newton's method was based on using the line tangent to the curve of $y = f(x)$ at point $(x_0, f(x_0))$.

When $x_0 \approx \alpha$, the graph of the tangent line is approximately the same as the graph of $y = f(x)$ around $x = \alpha$.

Then the root of the tangent line is used to approximate α.

Consider using an approximating line based on "interpolation".

Assume there are two estimates of the root α, say x_0 and x_1.

Secant method

Newton's method was based on using the line tangent to the curve of $y = f(x)$ at point $(x_0, f(x_0))$.

When $x_0 \approx \alpha$, the graph of the tangent line is approximately the same as the graph of $y = f(x)$ around $x = \alpha$.

Then the root of the tangent line is used to approximate α.

Consider using an approximating line based on "interpolation".

Assume there are two estimates of the root α, say x_0 and x_1.

Then, produce a linear function

$$q(x) = a_0 + a_1 x, \text{ such that}$$
$$q(x_0) = f(x_0), \qquad q(x_1) = f(x_1).$$

Secant method

Newton's method was based on using the line tangent to the curve of $y = f(x)$ at point $(x_0, f(x_0))$.

When $x_0 \approx \alpha$, the graph of the tangent line is approximately the same as the graph of $y = f(x)$ around $x = \alpha$.

Then the root of the tangent line is used to approximate α.

Consider using an approximating line based on "interpolation".

Assume there are two estimates of the root α, say x_0 and x_1.

Then, produce a linear function

$$q(x) = a_0 + a_1 x, \text{ such that}$$
$$q(x_0) = f(x_0), \qquad q(x_1) = f(x_1).$$

This line is called a **secant line** through points x_0 and x_1.

Secant method

Equation of secant line is given by

$$q(x) = \frac{(x_1 - x)f(x_0) + (x - x_0)f(x_1)}{x_1 - x_0}.$$

Secant method

Equation of secant line is given by

$$q(x) = \frac{(x_1 - x)f(x_0) + (x - x_0)f(x_1)}{x_1 - x_0}.$$

This equation is linear in x.

Secant method

Equation of secant line is given by

$$q(x) = \frac{(x_1 - x)f(x_0) + (x - x_0)f(x_1)}{x_1 - x_0}.$$

This equation is linear in x.

Solve the equation $q(x) = 0$, denoting its root by x_2:

$$x_2 = x_1 - f(x_1) : \frac{f(x_1) - f(x_0)}{x_1 - x_0}.$$

Secant method

Equation of secant line is given by

$$q(x) = \frac{(x_1 - x)f(x_0) + (x - x_0)f(x_1)}{x_1 - x_0}.$$

This equation is linear in x.

Solve the equation $q(x) = 0$, denoting its root by x_2:

$$x_2 = x_1 - f(x_1) : \frac{f(x_1) - f(x_0)}{x_1 - x_0}.$$

We can now repeat the whole process: use x_1 and x_2 to produce another secant line, and then use its root to approximate α.

Secant method

Equation of secant line is given by

$$q(x) = \frac{(x_1 - x)f(x_0) + (x - x_0)f(x_1)}{x_1 - x_0}.$$

This equation is linear in x.

Solve the equation $q(x) = 0$, denoting its root by x_2:

$$x_2 = x_1 - f(x_1) : \frac{f(x_1) - f(x_0)}{x_1 - x_0}.$$

We can now repeat the whole process: use x_1 and x_2 to produce another secant line, and then use its root to approximate α.

This yields the general iteration formula:

$$x_{n+1} = x_n - f(x_n) : \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}, \quad \forall n \in \mathbb{N}.$$

Secant method

Equation of secant line is given by

$$q(x) = \frac{(x_1 - x)f(x_0) + (x - x_0)f(x_1)}{x_1 - x_0}.$$

This equation is linear in x.

Solve the equation $q(x) = 0$, denoting its root by x_2:

$$x_2 = x_1 - f(x_1) : \frac{f(x_1) - f(x_0)}{x_1 - x_0}.$$

We can now repeat the whole process: use x_1 and x_2 to produce another secant line, and then use its root to approximate α.

This yields the general iteration formula:

$$x_{n+1} = x_n - f(x_n) : \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}, \quad \forall n \in \mathbb{N}.$$

This is called the **secant method** for solving equation $f(x) = 0$.

Secant method

Secant method

Example

Solve $f(x) \equiv x^6 - x - 1 = 0$ with initial guesses $x_0 = 2$ and $x_1 = 0$.

n	x_n	$f(x_n)$	$x_n - x_{n-1}$	$\alpha - x_n$
0	2.0	61.0		
1	1.0	-1.0	-1.0	
2	1.01612903	$-9.15e-1$	$1.61e-2$	$1.35e-1$
3	1.19057777	$6.57e-1$	$1.74e-1$	$1.19e-1$
4	1.11765583	$-1.68e-1$	$-7.29e-2$	$-5.59e-2$
5	1.13253155	$-2.24e-2$	$1.49e-2$	$1.71e-2$
6	1.13481681	$9.54e-4$	$2.29e-3$	$2.19e-3$
7	1.13472414	$-5.07e-6$	$-9.32e-5$	$-9.72e-5$
8	1.13472414	$-1.13e-9$	$4.92e-7$	$4.29e-7$

As in Newton's method example, we can see that

$$x_n - x_{n-1} \approx \alpha - x_n.$$

Secant method

Secant method requires more iterates than Newton method.

Secant method

Secant method requires more iterates than Newton method.

But, note that the secant method does not require a knowledge of $f'(x)$, whereas Newton's method requires both $f(x)$ and $f'(x)$.

Secant method

Secant method requires more iterates than Newton method.

But, note that the secant method does not require a knowledge of $f'(x)$, whereas Newton's method requires both $f(x)$ and $f'(x)$.

Note also that the secant method

$$x_{n+1} = x_n - \frac{f(x_n)}{\frac{f(x_n)-f(x_{n-1})}{x_n - x_{n-1}}}$$

Secant method

Secant method requires more iterates than Newton method.

But, note that the secant method does not require a knowledge of $f'(x)$, whereas Newton's method requires both $f(x)$ and $f'(x)$.

Note also that the secant method

$$x_{n+1} = x_n - \frac{f(x_n)}{\frac{f(x_n)-f(x_{n-1})}{x_n-x_{n-1}}}$$

can be considered as an approximation of the Newton's method:

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

by using the approximation of the derivative:

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

Secant method: Convergence Analysis

It can be shown that secant method error satisfies

$$\alpha - x_{n+1} = (\alpha - x_n)(\alpha - x_{n-1}) \cdot \left(-\frac{f''(\xi_n)}{2f'(\zeta_n)} \right)$$

with ξ_n and ζ_n unknown points located between the minimum and maximum of $\{x_{n-1}, x_n, \alpha\}$.

Secant method: Convergence Analysis

It can be shown that secant method error satisfies

$$\alpha - x_{n+1} = (\alpha - x_n)(\alpha - x_{n-1}) \cdot \left(-\frac{f''(\xi_n)}{2f'(\zeta_n)} \right)$$

with ξ_n and ζ_n unknown points located between the minimum and maximum of $\{x_{n-1}, x_n, \alpha\}$.

Recall that the Newton's method iterates satisfied

$$\alpha - x_{n+1} = (\alpha - x_n)^2 \cdot \left(-\frac{f''(\xi_n)}{2f'(x_n)} \right)$$

Secant method: Convergence Analysis

It can be shown that secant method error satisfies

$$\alpha - x_{n+1} = (\alpha - x_n)(\alpha - x_{n-1}) \cdot \left(-\frac{f''(\xi_n)}{2f'(\zeta_n)} \right)$$

with ξ_n and ζ_n unknown points located between the minimum and maximum of $\{x_{n-1}, x_n, \alpha\}$.

Recall that the Newton's method iterates satisfied

$$\alpha - x_{n+1} = (\alpha - x_n)^2 \cdot \left(-\frac{f''(\xi_n)}{2f'(x_n)} \right)$$

From the error formula for secant method it follows that $x_n \to \alpha$ as $n \to \infty$ assuming that x_0 and x_1 are chosen sufficiently close to α.

Secant method: Convergence Analysis

It can be shown that secant method error satisfies

$$\alpha - x_{n+1} = (\alpha - x_n)(\alpha - x_{n-1}) \cdot \left(-\frac{f''(\xi_n)}{2f'(\zeta_n)} \right)$$

with ξ_n and ζ_n unknown points located between the minimum and maximum of $\{x_{n-1}, x_n, \alpha\}$.

Recall that the Newton's method iterates satisfied

$$\alpha - x_{n+1} = (\alpha - x_n)^2 \cdot \left(-\frac{f''(\xi_n)}{2f'(x_n)} \right)$$

From the error formula for secant method it follows that $x_n \to \alpha$ as $n \to \infty$ assuming that x_0 and x_1 are chosen sufficiently close to α.

Moreover, it can be proved (assuming that $f \in C^2([a, b])$) that

$$\lim_{n\to\infty} \frac{|\alpha - x_{n+1}|}{|\alpha - x_n|^r} = \left| \frac{f''(\alpha)}{2f'(\alpha)} \right|^{r-1}$$

where $r = \frac{1+\sqrt{5}}{2} \approx 1.62$.

Secant method: Convergence Analysis

The last formula says that when we are close to α, that

$$|\alpha - x_{n+1}| \approx C|\alpha - x_n|^r \text{ with } C = \left|\frac{f''(\alpha)}{2f'(\alpha)}\right|^{r-1} \text{ and } r \approx 1.62.$$

Secant method: Convergence Analysis

The last formula says that when we are close to α, that

$$|\alpha - x_{n+1}| \approx C|\alpha - x_n|^r \text{ with } C = \left|\frac{f''(\alpha)}{2f'(\alpha)}\right|^{r-1} \text{ and } r \approx 1.62.$$

This looks very much like the Newton's method result:

$$|\alpha - x_{n+1}| \approx M|\alpha - x_n|^2 \text{ with } M = \left|\frac{f''(\alpha)}{2f'(\alpha)}\right|$$

Secant method: Convergence Analysis

The last formula says that when we are close to α, that

$$|\alpha - x_{n+1}| \approx C|\alpha - x_n|^r \text{ with } C = \left|\frac{f''(\alpha)}{2f'(\alpha)}\right|^{r-1} \text{ and } r \approx 1.62.$$

This looks very much like the Newton's method result:

$$|\alpha - x_{n+1}| \approx M|\alpha - x_n|^2 \text{ with } M = \left|\frac{f''(\alpha)}{2f'(\alpha)}\right|$$

Obviously, the secant method converge slower than Newton's method ($r = 1.62$ versus $r = 2$).

Secant method: Convergence Analysis

The last formula says that when we are close to α, that

$$|\alpha - x_{n+1}| \approx C|\alpha - x_n|^r \text{ with } C = \left|\frac{f''(\alpha)}{2f'(\alpha)}\right|^{r-1} \text{ and } r \approx 1.62.$$

This looks very much like the Newton's method result:

$$|\alpha - x_{n+1}| \approx M|\alpha - x_n|^2 \text{ with } M = \left|\frac{f''(\alpha)}{2f'(\alpha)}\right|$$

Obviously, the secant method converge slower than Newton's method ($r = 1.62$ versus $r = 2$).

Both the secant and Newton's methods converge at faster than a linear rate, and they are called super-linear methods.

Secant method: Convergence Analysis

It can be proved that

$$\lim_{n\to\infty} \frac{|x_{n+1} - x_n|}{|\alpha - x_n|} = 1$$

and therefore,

$$|\alpha - x_n| \approx |x_{n+1} - x_n|$$

is a good error estimator for the true error.

Secant method: Convergence Analysis

It can be proved that

$$\lim_{n\to\infty} \frac{|x_{n+1} - x_n|}{|\alpha - x_n|} = 1$$

and therefore,

$$|\alpha - x_n| \approx |x_{n+1} - x_n|$$

is a good error estimator for the true error.

Warning!: Do not combine the secant method formula and write it in the form

$$x_{n+1} = \frac{f(x_n)x_{n-1} - f(x_{n-1})x_n}{f(x_n) - f(x_{n-1})}.$$

Secant method: Convergence Analysis

It can be proved that

$$\lim_{n\to\infty} \frac{|x_{n+1} - x_n|}{|\alpha - x_n|} = 1$$

and therefore,

$$|\alpha - x_n| \approx |x_{n+1} - x_n|$$

is a good error estimator for the true error.

Warning!: Do not combine the secant method formula and write it in the form

$$x_{n+1} = \frac{f(x_n)x_{n-1} - f(x_{n-1})x_n}{f(x_n) - f(x_{n-1})}.$$

This has enormous loss of significance error as compared with the earlier formulation:

$$x_{n+1} = x_n - \frac{f(x_n)\,(x_n - x_{n-1})}{f(x_n) - f(x_{n-1}).}$$

Costs of Secant and Newton's methods

Observe that the Newton's method

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

requires two function evaluations per iteration.

Costs of Secant and Newton's methods

Observe that the Newton's method

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

requires two function evaluations per iteration.

Whilst the secant method

$$x_{n+1} = x_n - \frac{f(x_n)\,(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})}$$

requires only one function evaluation per iteration, following the initial step.

Costs of Secant and Newton's methods

Observe that the Newton's method

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

requires two function evaluations per iteration.

Whilst the secant method

$$x_{n+1} = x_n - \frac{f(x_n)\,(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})}$$

requires only one function evaluation per iteration, following the initial step.

For this reason, the secant method is often faster in time, even though more iterates are needed with it than with Newton's method to attain a similar accuracy.

Costs of Secant and Newton's methods

Let T_N and T_S be CPU times used to compute the root α with a given accuracy using Newton's method and Secant method, respectively. Then, it can be shown that

$$\frac{T_S}{T_N} = \frac{t_f}{t_f + st_f} \cdot \frac{\ln 2}{\ln r}, \quad r \approx 1.62,$$
$$\frac{T_S}{T_N} = \frac{1}{1+s} \cdot \frac{\ln 2}{\ln r}.$$

Costs of Secant and Newton's methods

Let T_N and T_S be CPU times used to compute the root α with a given accuracy using Newton's method and Secant method, respectively. Then, it can be shown that

$$\frac{T_S}{T_N} = \frac{t_f}{t_f + s t_f} \cdot \frac{\ln 2}{\ln r}, \quad r \approx 1.62,$$
$$\frac{T_S}{T_N} = \frac{1}{1+s} \cdot \frac{\ln 2}{\ln r}.$$

Secant method will be faster if

$$\frac{T_S}{T_N} = \frac{1}{1+s} \cdot \frac{\ln 2}{\ln r} < 1,$$
$$s > \frac{\ln 2}{\ln r} - 1 \approx 0.44.$$

In other words if CPU time needed to evaluate $f'(x)$ is bigger than 44% from the time needed to evaluate $f(x)$, then we should choose secant method.

Advantages and disadvantages of secant method

Advantages:

1. Secant method converges at faster than a linear rate, so that it is more rapidly convergent than the bisection method.

Advantages and disadvantages of secant method

Advantages:

1. Secant method converges at faster than a linear rate, so that it is more rapidly convergent than the bisection method.
2. It does not require use of the derivative of the function, something that is not available in a number of applications.

Advantages and disadvantages of secant method

Advantages:

1. Secant method converges at faster than a linear rate, so that it is more rapidly convergent than the bisection method.
2. It does not require use of the derivative of the function, something that is not available in a number of applications.
3. It requires only one function evaluation per iteration, as compared with Newton's method which requires two.

Advantages and disadvantages of secant method

Advantages:

1. Secant method converges at faster than a linear rate, so that it is more rapidly convergent than the bisection method.
2. It does not require use of the derivative of the function, something that is not available in a number of applications.
3. It requires only one function evaluation per iteration, as compared with Newton's method which requires two.

Advantages and disadvantages of secant method

Advantages:

1. Secant method converges at faster than a linear rate, so that it is more rapidly convergent than the bisection method.
2. It does not require use of the derivative of the function, something that is not available in a number of applications.
3. It requires only one function evaluation per iteration, as compared with Newton's method which requires two.

Disadvantages:

1. Secant method may not converge.

Advantages and disadvantages of secant method

Advantages:

1. Secant method converges at faster than a linear rate, so that it is more rapidly convergent than the bisection method.
2. It does not require use of the derivative of the function, something that is not available in a number of applications.
3. It requires only one function evaluation per iteration, as compared with Newton's method which requires two.

Disadvantages:

1. Secant method may not converge.
2. There is no guaranteed error bound for the computed iterates.

Advantages and disadvantages of secant method

Advantages:

1. Secant method converges at faster than a linear rate, so that it is more rapidly convergent than the bisection method.
2. It does not require use of the derivative of the function, something that is not available in a number of applications.
3. It requires only one function evaluation per iteration, as compared with Newton's method which requires two.

Disadvantages:

1. Secant method may not converge.
2. There is no guaranteed error bound for the computed iterates.
3. It is likely to have difficulty if $f'(\alpha) = 0$. This means the $x-$axis is tangent to the graph of $y = f(x)$ at $x = \alpha$.

Advantages and disadvantages of secant method

Advantages:

1. Secant method converges at faster than a linear rate, so that it is more rapidly convergent than the bisection method.
2. It does not require use of the derivative of the function, something that is not available in a number of applications.
3. It requires only one function evaluation per iteration, as compared with Newton's method which requires two.

Disadvantages:

1. Secant method may not converge.
2. There is no guaranteed error bound for the computed iterates.
3. It is likely to have difficulty if $f'(\alpha) = 0$. This means the $x-$axis is tangent to the graph of $y = f(x)$ at $x = \alpha$.
4. Newton's method generalizes more easily to new methods for solving simultaneous systems of nonlinear equations.

Brent's method

Richard Brent devised a method combining the advantages of the bisection method and the secant method.

1. It is guaranteed to converge.

Brent's method

Richard Brent devised a method combining the advantages of the bisection method and the secant method.

1. It is guaranteed to converge.
2. It has an error bound which will converge to zero in practice.

Brent's method

Richard Brent devised a method combining the advantages of the bisection method and the secant method.

1. It is guaranteed to converge.
2. It has an error bound which will converge to zero in practice.
3. For most problems $f(x) = 0$, with $f(x)$ differentiable about the root α, the method behaves like the secant method.

Brent's method

Richard Brent devised a method combining the advantages of the bisection method and the secant method.

1. It is guaranteed to converge.
2. It has an error bound which will converge to zero in practice.
3. For most problems $f(x) = 0$, with $f(x)$ differentiable about the root α, the method behaves like the secant method.
4. In the worst case, it is not too much worse in its convergence than the bisection method.

Brent's method

Richard Brent devised a method combining the advantages of the bisection method and the secant method.

1. It is guaranteed to converge.
2. It has an error bound which will converge to zero in practice.
3. For most problems $f(x) = 0$, with $f(x)$ differentiable about the root α, the method behaves like the secant method.
4. In the worst case, it is not too much worse in its convergence than the bisection method.

Brent's method

Richard Brent devised a method combining the advantages of the bisection method and the secant method.

1. It is guaranteed to converge.
2. It has an error bound which will converge to zero in practice.
3. For most problems $f(x) = 0$, with $f(x)$ differentiable about the root α, the method behaves like the secant method.
4. In the worst case, it is not too much worse in its convergence than the bisection method.

In Matlab, it is implemented as *fzero*; and it is present in most Fortran numerical analysis libraries.

Order of convergence

Besides the convergence of iterations generated by a numerical method, often we would like to know how fast is the sequence x_n converging to a solution α. In other words, we want to know how fast the error $\alpha - x_n$ is decreasing.

Order of convergence

Besides the convergence of iterations generated by a numerical method, often we would like to know how fast is the sequence x_n converging to a solution α. In other words, we want to know how fast the error $\alpha - x_n$ is decreasing.

Definition

We say that sequence $\{x_n\}_{n=0}^{\infty}$ converges to α with order of convergence $p \geq 1$, if

$$|\alpha - x_{n+1}| \leq C|\alpha - x_n|^p, \quad \forall n \in \mathbb{N},$$

where $C \geq 0$ is a constant. Cases $p = 1, 2, 3$ are called **linear**, **quadratic** and **cubic** convergences. In case of linear convergence, constant C is called the **rate of linear convergence**.

Thus, bisection method is linearly convergent with rate 0.5, Newton's method is quadratically convergent, and secant method has order of convergence $p = \frac{1+\sqrt{5}}{2} \approx 1.62$.

Numerical Methods

Prof. Dr.hab. Viorel Bostan

Spring 2022 Lecture 3

Fixed point iteration

Another approach in deriving rootfinding methods is based on so-called **fixed point iterations**.
Consider solving the following two equations with solution α

$$E1: \quad x = 1 + 0.5\sin x, \quad \alpha = 1.49870113351785,$$
$$E2: \quad x = 3 + 2\sin x, \quad \alpha = 3.09438341304928.$$

Fixed point iteration

In order to solve these equations, we are going to use a numerical scheme called "fixed point iteration".

Fixed point iteration

In order to solve these equations, we are going to use a numerical scheme called "fixed point iteration".

It amounts to making an initial guess of x_0 and substituting this into the right side of the equation.

$$
\begin{aligned}
x_1 &= 1 + 0.5 \sin x_0, \\
x_2 &= 1 + 0.5 \sin x_1, \\
&\dots \\
x_n &= 1 + 0.5 \sin x_{n-1}, \\
&\dots
\end{aligned}
$$

This is repeated until convergence occurs or until the iteration is terminated. The above iterations can be written symbolically as

$$
\begin{aligned}
E1: &\quad x_{n+1} = 1 + 0.5 \sin x_n, \quad n = 0, 1, 2, \dots \\
E2: &\quad x_{n+1} = 3 + 2 \sin x_n, \quad n = 0, 1, 2, \dots
\end{aligned}
$$

Fixed point iterations

	$E1$	$E2$
n	x_n	x_n
0	0.00000000000000	3.00000000000000
1	1.00000000000000	3.28224001611973
2	1.42073549240395	2.71963177181556
3	1.49438099256432	3.81910025488514
4	1.49854088439917	1.74629389651652
5	1.49869535552190	4.96927957214762
6	1.49870092540704	1.06563065299216
7	1.49870112602244	4.75018861639465
8	1.49870113324789	1.00142864236516
9	1.49870113350813	4.68448404916097
10	1.49870113351750	1.00077863465869

Fixed point iterations

Clearly convergence is occurring with $E1$, but not with $E2$.

Fixed point iterations

Clearly convergence is occurring with $E1$, but not with $E2$.
Why does one of these iterations converge, but not the other?
The graphs show similar behavior, so why the difference?
Before answering these questions, consider another example.

Fixed point iterations

Clearly convergence is occurring with $E1$, but not with $E2$.
Why does one of these iterations converge, but not the other?
The graphs show similar behavior, so why the difference?
Before answering these questions, consider another example.
Suppose we are solving the equation $x^2 - 5 = 0$ with exact root $\alpha = \sqrt{5} \approx 2.2361$ using iterates of the form

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

Fixed point iterations

Clearly convergence is occurring with $E1$, but not with $E2$.
Why does one of these iterations converge, but not the other?
The graphs show similar behavior, so why the difference?
Before answering these questions, consider another example.
Suppose we are solving the equation $x^2 - 5 = 0$ with exact root $\alpha = \sqrt{5} \approx 2.2361$ using iterates of the form

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

Consider four different iterations:

$$\begin{aligned}
\mathrm{I}_1\colon &\quad x_{n+1} = 5 + x_n - x_n^2; \\
\mathrm{I}_2\colon &\quad x_{n+1} = \frac{5}{x_n}; \\
\mathrm{I}_3\colon &\quad x_{n+1} = 1 + x_n - \frac{1}{5}x_n^2; \\
\mathrm{I}_4\colon &\quad x_{n+1} = \frac{1}{2}\left(x_n + \frac{5}{x_n}\right).
\end{aligned}$$

Fixed point iterations

It can be shown for all 4 sequences that if $\lim_{n\to\infty} = \alpha$ exist, then $\alpha = \sqrt{5}$.

Fixed point iterations

It can be shown for all 4 sequences that if $\lim_{n\to\infty} = \alpha$ exist, then $\alpha = \sqrt{5}$.

Numerical results are presented below:

	I_1	I_2	I_3	I_4
n	x_n	x_n	x_n	x_n
0	$1.0e+00$	1.0	1.0	1.0
1	$5.0000e+00$	5.0	1.8000	3.0000
2	$-1.5000e+01$	1.0	2.1520	2.3333
3	$-2.3500e+02$	5.0	2.2258	2.2381
4	$-5.5455e+04$	1.0	2.2350	2.2361
5	$-3.0753e+09$	5.0	2.2360	2.2361
6	$-9.4575e+18$	1.0	2.2361	2.2361
7	$-8.9445e+37$	5.0	2.2361	2.2361
8	$-8.0004e+75$	1.0	2.2361	2.2361

Fixed point iterations

As another example of a fixed point iteration, note that the Newton's method

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

is also a fixed point iteration, for the equation

$$x = x - \frac{f(x)}{f'(x)}.$$

Fixed point iterations

As another example of a fixed point iteration, note that the Newton's method

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

is also a fixed point iteration, for the equation

$$x = x - \frac{f(x)}{f'(x)}.$$

In general, we are interested in solving equations of the form

$$x = g(x)$$

by means of fixed point iterations:

$$x_{n+1} = g(x_n) \quad \text{with } n = 0, 1, 2, 3, \ldots$$

Fixed point iterations

As another example of a fixed point iteration, note that the Newton's method

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

is also a fixed point iteration, for the equation

$$x = x - \frac{f(x)}{f'(x)}.$$

In general, we are interested in solving equations of the form

$$x = g(x)$$

by means of fixed point iterations:

$$x_{n+1} = g(x_n) \quad \text{with } n = 0, 1, 2, 3, \ldots$$

It is called '**fixed point iteration**' because the root α is a fixed point of the function $g(x)$, meaning that α is a number s.t.

$$g(\alpha) = \alpha.$$

Fixed point iterations: Existence of fixed point

Begin by asking whether there is a solution to equation

$$x = g(x).$$

Fixed point iterations: Existence of fixed point

Begin by asking whether there is a solution to equation

$$x = g(x).$$

For this to occur, the graphs of $y = x$ and $y = g(x)$ must intersect. There are several lemmas and theorems that give conditions under which we are guaranteed that there is a fixed point i.e. $\alpha = g(\alpha)$.

Fixed point iterations: Existence of fixed point

Begin by asking whether there is a solution to equation

$$x = g(x).$$

For this to occur, the graphs of $y = x$ and $y = g(x)$ must intersect. There are several lemmas and theorems that give conditions under which we are guaranteed that there is a fixed point i.e. $\alpha = g(\alpha)$.

Lemma

Let function $g(x) \in C[a, b]$ and suppose it satisfies the property

$$\text{If } a \leqslant x \leqslant b \text{ then } a \leqslant g(x) \leqslant b,$$
$$i.e. \quad g([a, b]) \subset [a, b].$$

Then, the equation $x = g(x)$ has at least one solution in the interval $[a, b]$.

Fixed point iterations

The following theorem ensures the convergence of fixed point iterations.

Theorem

Assume $g(x) \in C^1[a,b]$ *and* $g([a,b]) \subset [a,b]$.
Further assume that

$$\lambda \stackrel{def}{=} \max_{a \leqslant x \leqslant b} |g'(x)| < 1.$$

Then:
$(S1)$ $\exists! \alpha \in [a,b]$ *s.t.* $\alpha = g(\alpha)$.
$(S2)$ $\forall x_0 \in [a,b]$, *the iteration* $x_{n+1} = g(x_n) \to \alpha$.

$$(S3) \quad |\alpha - x_n| \leqslant \frac{\lambda^n}{1-\lambda} |x_1 - x_0|, \quad n \geqslant 1.$$

$$(S4) \quad \lim_{n \to \infty} \frac{|\alpha - x_{n+1}|}{|\alpha - x_n|} = g'(\alpha).$$

Fixed point iterations

The statement

$$(S4) \qquad \lim_{n\to\infty} \frac{|\alpha - x_{n+1}|}{|\alpha - x_n|} = g'(\alpha)$$

tells us that if x_n is close to α then

$$|\alpha - x_{n+1}| \approx g'(\alpha)\,|\alpha - x_n|\,.$$

Fixed point iterations

The statement

$$(S4) \quad \lim_{n\to\infty} \frac{|\alpha - x_{n+1}|}{|\alpha - x_n|} = g'(\alpha)$$

tells us that if x_n is close to α then

$$|\alpha - x_{n+1}| \approx g'(\alpha)\,|\alpha - x_n|\,.$$

i.e. the errors will decrease by a constant factor of $g'(\alpha)$.

Fixed point iterations

The statement

$$(S4) \quad \lim_{n\to\infty} \frac{|\alpha - x_{n+1}|}{|\alpha - x_n|} = g'(\alpha)$$

tells us that if x_n is close to α then

$$|\alpha - x_{n+1}| \approx g'(\alpha)\,|\alpha - x_n|\,.$$

i.e. the errors will decrease by a constant factor of $g'(\alpha)$.

If this is negative, then the errors will oscillate between positive and negative, and the iterates will be approaching α from both sides.

Fixed point iterations

The statement

$$(S4) \quad \lim_{n\to\infty} \frac{|\alpha - x_{n+1}|}{|\alpha - x_n|} = g'(\alpha)$$

tells us that if x_n is close to α then

$$|\alpha - x_{n+1}| \approx g'(\alpha)\,|\alpha - x_n|\,.$$

i.e. the errors will decrease by a constant factor of $g'(\alpha)$.

If this is negative, then the errors will oscillate between positive and negative, and the iterates will be approaching α from both sides.

When $g'(\alpha)$ is positive, the iterates will approach α from only one side.

Fixed point iterations

The statement

$$(S4) \quad \lim_{n\to\infty} \frac{|\alpha - x_{n+1}|}{|\alpha - x_n|} = g'(\alpha)$$

tells us that if x_n is close to α then

$$|\alpha - x_{n+1}| \approx g'(\alpha)\,|\alpha - x_n|\,.$$

i.e. the errors will decrease by a constant factor of $g'(\alpha)$.

If this is negative, then the errors will oscillate between positive and negative, and the iterates will be approaching α from both sides.

When $g'(\alpha)$ is positive, the iterates will approach α from only one side.

Also we can see what happens when $g'(\alpha) > 1$. Then, the errors will increase with each new iteration rather than decrease in size.

Fixed point iterations. Examples explained

Consider first fixed point example:

$$E1: \quad x = 1 + 0.5 \sin x,$$
$$E2: \quad x = 3 + 2 \sin x.$$

Fixed point iterations. Examples explained

Consider first fixed point example:

$$E1: \quad x = 1 + 0.5 \sin x,$$
$$E2: \quad x = 3 + 2 \sin x.$$

For equation $E1$ we have $g(x) = 1 + 0.5 \sin x$ and

$$g'(x) = 0.5 \cos x,$$
$$g'(\alpha) \leqslant 0.5$$

and therefore $x_{n+1} = 1 + 0.5 \sin x_n$ will converge to α.

Fixed point iterations. Examples explained

Consider first fixed point example:

$$E1: \quad x = 1 + 0.5 \sin x,$$
$$E2: \quad x = 3 + 2 \sin x.$$

For equation $E1$ we have $g(x) = 1 + 0.5 \sin x$ and

$$g'(x) = 0.5 \cos x,$$
$$g'(\alpha) \leqslant 0.5$$

and therefore $x_{n+1} = 1 + 0.5 \sin x_n$ will converge to α.

For equation $E2$ we have $g(x) = 3 + 2 \sin x$ and

$$g'(x) = 2 \cos x,$$
$$g'(\alpha) = g'(3.09438341304928) \approx -1.998$$

and consequently the fixed point iteration $x_{n+1} = 3 + 2 \sin x_n$ will diverge.

Fixed point iterations

Consider second example $x^2 = 5$ with solution $\alpha = \sqrt{5}$.

Fixed point iterations

Consider second example $x^2 = 5$ with solution $\alpha = \sqrt{5}$.

Case (I_1) : $\quad g(x) = 5 + x - x^2$.

$$g'(x) = 1 - 2x,$$
$$g'(\sqrt{5}) = 1 - 2\sqrt{5} < -1.$$

Thus, $x_n = g(x_{n-1})$ will not converge to $\sqrt{5}$.

Fixed point iterations

Consider second example $x^2 = 5$ with solution $\alpha = \sqrt{5}$.

Case (I_1) : $\quad g(x) = 5 + x - x^2$.

$$g'(x) = 1 - 2x,$$
$$g'(\sqrt{5}) = 1 - 2\sqrt{5} < -1.$$

Thus, $x_n = g(x_{n-1})$ will not converge to $\sqrt{5}$.

Case (I_2) : $\quad g(x) = \frac{5}{x}$.

$$g'(x) = -\frac{5}{x^2},$$
$$g'(\sqrt{5}) = -1.$$

So, $x_n = g(x_{n-1})$ may converge or diverge, but results show that it doesn't converge to $\sqrt{5}$.

Fixed point iterations. Examples explained

Case (I_3) : $\quad g(x) = 1 + x - \frac{1}{5}x^2$.

$$g'(x) = 1 - \frac{2}{5}x,$$
$$g'(\sqrt{5}) = 1 - \frac{2}{5}\sqrt{5} \approx 0.106.$$

Therefore, $x_n = g(x_{n-1}) \to \sqrt{5}$. Moreover,

$$\left|\sqrt{5} - x_{n+1}\right| \approx 0.106 \left|\sqrt{5} - x_n\right|,$$

This is a linear convergence with linear rate 0.106.

Fixed point iterations

Case (I_4) : $\quad g(x) = \frac{1}{2}\left(x + \frac{5}{x}\right)$.

$$g'(x) = \frac{1}{2}\left(1 - \frac{5}{x^2}\right),$$
$$g'(\sqrt{5}) = 0.$$

Thus, $x_n = g(x_{n-1}) \to \sqrt{5}$ is convergent with an order of convergence higher than 1.

Fixed point iterations

Case (I_4) : $\quad g(x) = \frac{1}{2}\left(x + \frac{5}{x}\right)$.

$$g'(x) = \frac{1}{2}\left(1 - \frac{5}{x^2}\right),$$
$$g'(\sqrt{5}) = 0.$$

Thus, $x_n = g(x_{n-1}) \to \sqrt{5}$ is convergent with an order of convergence higher than 1.

It can be observed that the last fixed point iteration is nothing else but Newton's method applied to equation $x^2 - 5 = 0$.

Fixed point iterations

Sometimes it is difficult to express initial equation $f(x) = 0$ in the form $x = g(x)$ such that the resulting iterates will converge.

Fixed point iterations

Sometimes it is difficult to express initial equation $f(x) = 0$ in the form $x = g(x)$ such that the resulting iterates will converge.

Such a process is presented in the following examples.

Fixed point iterations

Sometimes it is difficult to express initial equation $f(x) = 0$ in the form $x = g(x)$ such that the resulting iterates will converge.

Such a process is presented in the following examples.

Example

Let equation $x^4 - x - 1 = 0$ be rewritten as

$$x = \sqrt[4]{1 + x},$$

which will provide us with iterations

$$x_0 = 1, \quad x_{n+1} = \sqrt[4]{1 + x_n}.$$

This sequence will converge to $\alpha \approx 1.2207$.

Fixed point iterations

Example

Let equation $x^3 + x - 1 = 0$ be rewritten as

$$x = \frac{1}{1 + x^2}.$$

That will provide us with iterations

$$x_0 = 1, \quad x_{n+1} = \frac{1}{1 + x_n^2},$$

which will converge to $\alpha \approx 0.6823$.

Fixed point iterations

$0 < g'(\alpha) < 1$

$-1 < g'(\alpha) < 0$

$g'(\alpha) > 1$

$g'(\alpha) < -1$

Order of convergence of fixed point iterations

If $|g'(\alpha)| < 1$, then we have that iterations $x_{n+1} = g(x_n)$ are at least linear convergent. If additionally, $g'(\alpha) \neq 0$, then we have exactly linear convergence with rate $g'(\alpha)$.

Order of convergence of fixed point iterations

If $|g'(\alpha)| < 1$, then we have that iterations $x_{n+1} = g(x_n)$ are at least linear convergent. If additionally, $g'(\alpha) \neq 0$, then we have exactly linear convergence with rate $g'(\alpha)$.

Consider Newton's method:

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

as a fixed point method with

$$g(x) = x - \frac{f(x)}{f'(x)}.$$

Compute $g'(\alpha)$:

$$g'(x) = -\frac{f(x)f''(x)}{\left(f'(x)\right)^2}, \quad g'(\alpha) = 0.$$

Therefore, Newton's method will converge and it will be faster than linear convergence.

Error analysis of fixed point iterations

Consider the fixed point iteration $x_{n+1} = g(x_n)$ and suppose that it is convergent to fixed point $\alpha = g(\alpha)$. Recall the formula:

$$\lim_{n\to\infty} \frac{\alpha - x_n}{\alpha - x_{n-1}} = g'(\alpha) = \lambda.$$

Thus,

$$\alpha - x_n \approx \lambda(\alpha - x_{n-1})$$

with $|\lambda| < 1$.

Error analysis of fixed point iterations

Consider the fixed point iteration $x_{n+1} = g(x_n)$ and suppose that it is convergent to fixed point $\alpha = g(\alpha)$. Recall the formula:

$$\lim_{n\to\infty} \frac{\alpha - x_n}{\alpha - x_{n-1}} = g'(\alpha) = \lambda.$$

Thus,

$$\alpha - x_n \approx \lambda(\alpha - x_{n-1})$$

with $|\lambda| < 1$.

If λ is known, then solve for α to get

$$\begin{aligned}
\alpha &\approx \frac{x_n - \lambda x_{n-1}}{1-\lambda} \\
&= \frac{x_n - \lambda x_n}{1-\lambda} + \frac{\lambda x_n - \lambda x_{n-1}}{1-\lambda} \\
&= x_n + \frac{\lambda}{1-\lambda}\Big[x_n - x_{n-1}\Big].
\end{aligned}$$

Error analysis of fixed point iterations

Thus, if λ is known, we have on the right side the so-called **extrapolation** of x_{n-1} and x_n:

$$\alpha \approx x_n + \frac{\lambda}{1-\lambda}\Big[x_n - x_{n-1}\Big].$$

Error analysis of fixed point iterations

Thus, if λ is known, we have on the right side the so-called **extrapolation** of x_{n-1} and x_n:

$$\alpha \approx x_n + \frac{\lambda}{1-\lambda}\Big[x_n - x_{n-1}\Big].$$

How to compute λ? From formula

$$\alpha - x_n \approx \lambda(\alpha - x_{n-1}),$$

solving for λ we get

$$\lambda \approx \frac{\alpha - x_n}{\alpha - x_{n-1}}.$$

Consider sequence of ratios:

$$\lambda_n = \frac{x_n - x_{n-1}}{x_{n-1} - x_{n-2}},$$

and since $x_n \to \alpha$, we have that $\lambda_n \to \lambda$ as $n \to \infty$.

Error analysis of fixed point iterations

Thus, once x_{n-2}, x_{n-1} and x_n are known, by substituting in extrapolation formula λ with λ_n, we get the so-called **Aitken's extrapolation formula**:

$$\hat{x}_n = x_n + \frac{\lambda_n}{1-\lambda_n}\Big[x_n - x_{n-1}\Big],$$

where

$$\lambda_n = \frac{x_n - x_{n-1}}{x_{n-1} - x_{n-2}}.$$

Error analysis of fixed point iterations

Thus, once x_{n-2}, x_{n-1} and x_n are known, by substituting in extrapolation formula λ with λ_n, we get the so-called **Aitken's extrapolation formula**:

$$\hat{x}_n = x_n + \frac{\lambda_n}{1-\lambda_n}\Big[x_n - x_{n-1}\Big],$$

where

$$\lambda_n = \frac{x_n - x_{n-1}}{x_{n-1} - x_{n-2}}.$$

Obviously, $\hat{x}_n$ is an approximation of α, much better than x_n. Moreover,

$$\alpha - x_n \approx \hat{x}_n - x_n = \frac{\lambda_n}{1-\lambda_n}\Big[x_n - x_{n-1}\Big],$$

which is known as **Aitken's error estimation formula** and can be used to estimate the error in fixed point iterations.

Error analysis. Example

Example

Consider solving for fixed point the equation $x = 2\pi + \sin x$ using fixed point iterations

$$x_n = 2\pi + \sin x_{n-1}, \quad \forall n \in \mathbb{N}$$

with initial guess $x_0 = 6$.

Error analysis. Example

Example

Consider solving for fixed point the equation $x = 2\pi + \sin x$ using fixed point iterations

$$x_n = 2\pi + \sin x_{n-1}, \quad \forall n \in \mathbb{N}$$

with initial guess $x_0 = 6$.

n	x_n	λ_n	$\alpha - x_n$	Aitken Estimate
0	6.0000000		$1.55e-02$	
1	6.0005845		$1.49e-02$	
2	6.0011458	0.9603	$1.44e-02$	$1.36e-02$
3	6.0016848	0.9604	$1.38e-02$	$1.31e-02$
4	6.0022026	0.9606	$1.33e-02$	$1.26e-02$
5	6.0027001	0.9607	$1.28e-02$	$1.22e-02$
6	6.0031780	0.9609	$1.23e-02$	$1.17e-02$
7	6.0036374	0.9610	$1.18e-02$	$1.13e-02$

Error analysis. Example

1. Observe a very slow linear convergence $(\alpha - x_n) \to 0$, since $\lambda_n \to \lambda = g'(\alpha) \approx 0.9644$ and thus

$$\alpha - x_n \approx 0.9644(\alpha - x_{n-1}),$$

which confirms the linear convergence with rate 0.9644. This rate (close to 1) tells us that with each iteration the error decreases only by 3.56%.

Error analysis. Example

1. Observe a very slow linear convergence $(\alpha - x_n) \to 0$, since $\lambda_n \to \lambda = g'(\alpha) \approx 0.9644$ and thus

$$\alpha - x_n \approx 0.9644(\alpha - x_{n-1}),$$

   which confirms the linear convergence with rate 0.9644. This rate (close to 1) tells us that with each iteration the error decreases only by 3.56%.

2. The Aitken estimate (last column in the table) approximates well the true error (the 4th column), and therefore it can be used to estimate the error in fixed point iteration methods.

$$\alpha - x_n \approx \hat{x}_n - x_n = \frac{\lambda_n}{1 - \lambda_n}\left[x_n - x_{n-1}\right].$$

Aitken algorithm

Based on the Aitken extrapolation, the following improvement to fixed point iteration, known as **Aitken algorithm**, can be proposed:

1. Start with initial guess x_0.

Aitken algorithm

Based on the Aitken extrapolation, the following improvement to fixed point iteration, known as **Aitken algorithm**, can be proposed:

1. Start with initial guess x_0.
2. Compute $x_1 = g(x_0)$ and $x_2 = g(x_1)$.

Aitken algorithm

Based on the Aitken extrapolation, the following improvement to fixed point iteration, known as **Aitken algorithm**, can be proposed:

1. Start with initial guess x_0.
2. Compute $x_1 = g(x_0)$ and $x_2 = g(x_1)$.
3. Compute Aitken extrapolation of x_0, x_1 and x_2:

$$\lambda_2 = \frac{x_2 - x_1}{x_1 - x_0},$$
$$x_3 = x_2 + \frac{\lambda_2}{1 + \lambda_2}\left[x_2 - x_1\right].$$

Aitken algorithm

Based on the Aitken extrapolation, the following improvement to fixed point iteration, known as **Aitken algorithm**, can be proposed:

1. Start with initial guess x_0.
2. Compute $x_1 = g(x_0)$ and $x_2 = g(x_1)$.
3. Compute Aitken extrapolation of x_0, x_1 and x_2:

$$\lambda_2 = \frac{x_2 - x_1}{x_1 - x_0},$$
$$x_3 = x_2 + \frac{\lambda_2}{1 + \lambda_2}\left[x_2 - x_1\right].$$

4. Compute $x_4 = g(x_3)$ and $x_5 = g(x_4)$.

Aitken algorithm

Based on the Aitken extrapolation, the following improvement to fixed point iteration, known as **Aitken algorithm**, can be proposed:

1. Start with initial guess x_0.
2. Compute $x_1 = g(x_0)$ and $x_2 = g(x_1)$.
3. Compute Aitken extrapolation of x_0, x_1 and x_2:

$$\lambda_2 = \frac{x_2 - x_1}{x_1 - x_0},$$
$$x_3 = x_2 + \frac{\lambda_2}{1 + \lambda_2}\left[x_2 - x_1\right].$$

4. Compute $x_4 = g(x_3)$ and $x_5 = g(x_4)$.
5. Compute again x_6 as Aitken extrapolation of x_3, x_4 and x_5.

Aitken algorithm

Based on the Aitken extrapolation, the following improvement to fixed point iteration, known as **Aitken algorithm**, can be proposed:

1. Start with initial guess x_0.
2. Compute $x_1 = g(x_0)$ and $x_2 = g(x_1)$.
3. Compute Aitken extrapolation of x_0, x_1 and x_2:

$$\lambda_2 = \frac{x_2 - x_1}{x_1 - x_0},$$
$$x_3 = x_2 + \frac{\lambda_2}{1 + \lambda_2}\left[x_2 - x_1\right].$$

4. Compute $x_4 = g(x_3)$ and $x_5 = g(x_4)$.
5. Compute again x_6 as Aitken extrapolation of x_3, x_4 and x_5.
6. And so on, until stopping criterion for fixed point iteration method is satisfied.

Aitken algorithm

Example

Consider again the fixed point iterations

$$x_n = 2\pi + \sin x_{n-1}, \quad \forall n \in \mathbb{N}$$

with initial guess $x_0 = 6$.

Using the Aitken algorithm we will get

$$\alpha - x_3 = 7.98e - 04, \qquad \alpha - x_6 = 2.27e - 06.$$

On the other hand, using original fixed point iterations we got

$$\alpha - x_3 = 1.38e - 02, \qquad \alpha - x_6 = 1.23e - 02.$$

Aitken algorithm

Example

Consider again the fixed point iterations

$$x_n = 2\pi + \sin x_{n-1}, \quad \forall n \in \mathbb{N}$$

with initial guess $x_0 = 6$.

Using the Aitken algorithm we will get

$$\alpha - x_3 = 7.98e - 04, \qquad \alpha - x_6 = 2.27e - 06.$$

On the other hand, using original fixed point iterations we got

$$\alpha - x_3 = 1.38e - 02, \qquad \alpha - x_6 = 1.23e - 02.$$

Therefore, Aitken algorithm based on Aitken extrapolation can accelerate greatly the convergence of linearly convergent fixed point iterations $x_n = g(x_{n-1})$.

Multiple roots

In some cases, the numerical methods for rootfinding will not converge as predicted by theory. For example, Newton's method, instead of converging quadratically, will converge linearly, even worse than bisection method.For example, this will happen if the root has a multiplicity greater than 1.

Multiple roots

In some cases, the numerical methods for rootfinding will not converge as predicted by theory. For example, Newton's method, instead of converging quadratically, will converge linearly, even worse than bisection method.For example, this will happen if the root has a multiplicity greater than 1.

Definition

We say that a root α of $f(x)$ has **multiplicity** $m \in \mathbb{N}$, if the function $f(x)$ can be factored as

$$f(x) = (x - \alpha)^m h(x),$$

where $h(x)$ is a continuous function such that $h(\alpha) \neq 0$. Roots of multiplicity 1 are called simple roots.

Example

Polynomial $f(x) = (x + 2.8)^2(x - 1.1)^3(x - 7)$ has root -2.8 of multiplicity 2, root 1.1 of multiplicity 3 and a simple root 7.

Multiple roots

Example

Let $f(x) = e^{x^2} - 1$. Obviously $x = 0$ is a root of this function.

Multiple roots

Example

Let $f(x) = e^{x^2} - 1$. Obviously $x = 0$ is a root of this function. But it is not that obvious that root $x = 0$ has multiplicity 2.

Multiple roots

Example

Let $f(x) = e^{x^2} - 1$. Obviously $x = 0$ is a root of this function. But it is not that obvious that root $x = 0$ has multiplicity 2. Rewrite $f(x)$ as

$$f(x) = x^2 \cdot \frac{e^{x^2} - 1}{x^2} = x^2 h(x)$$

and let's show that $h(0) \neq 0$. Indeed,

$$\lim_{x \to 0} h(x) = \lim_{x \to 0} \frac{e^{x^2} - 1}{x^2} = 1.$$

Multiple roots

Example

Let $f(x) = e^{x^2} - 1$. Obviously $x = 0$ is a root of this function. But it is not that obvious that root $x = 0$ has multiplicity 2. Rewrite $f(x)$ as

$$f(x) = x^2 \cdot \frac{e^{x^2} - 1}{x^2} = x^2 h(x)$$

and let's show that $h(0) \neq 0$. Indeed,

$$\lim_{x \to 0} h(x) = \lim_{x \to 0} \frac{e^{x^2} - 1}{x^2} = 1.$$

For simple roots, indeed the rootfinding numerical methods considered up to this moment will have a behavior as discussed. Let's consider now the case of multiple roots.

Multiple roots

Let α be a root of multiplicity m of function $f(x)$.
For simplicity, suppose it has multiplicity 3. Therefore,

$$f(x) = (x - \alpha)^3 h(x) \quad \text{and} \quad h(\alpha) \neq 0.$$

Multiple roots

Let α be a root of multiplicity m of function $f(x)$.
For simplicity, suppose it has multiplicity 3. Therefore,

$$f(x) = (x-\alpha)^3 h(x) \quad \text{and} \quad h(\alpha) \neq 0.$$

Then,

$$\begin{aligned} f'(x) &= 3(x-\alpha)^2 h(x) + (x-\alpha)^3 h'(x) \\ &= (x-\alpha)^2 \big[3h(x) + (x-\alpha)h'(x)\big] = (x-\alpha)^2 q(x) \end{aligned}$$

Multiple roots

Let α be a root of multiplicity m of function $f(x)$.
For simplicity, suppose it has multiplicity 3. Therefore,

$$f(x) = (x-\alpha)^3 h(x) \quad \text{and} \quad h(\alpha) \neq 0.$$

Then,

$$\begin{aligned} f'(x) &= 3(x-\alpha)^2 h(x) + (x-\alpha)^3 h'(x) \\ &= (x-\alpha)^2 \big[3h(x) + (x-\alpha)h'(x)\big] = (x-\alpha)^2 q(x) \end{aligned}$$

Obviously, $f'(\alpha) = 0$ and $q(\alpha) = 3h(\alpha) \neq 0$ and therefore α is a root of $f'(x)$ of multiplicity 2.

Multiple roots

Let α be a root of multiplicity m of function $f(x)$.
For simplicity, suppose it has multiplicity 3. Therefore,

$$f(x) = (x-\alpha)^3 h(x) \quad \text{and} \quad h(\alpha) \neq 0.$$

Then,

$$\begin{aligned} f'(x) &= 3(x-\alpha)^2 h(x) + (x-\alpha)^3 h'(x) \\ &= (x-\alpha)^2 \big[3h(x) + (x-\alpha)h'(x)\big] = (x-\alpha)^2 q(x) \end{aligned}$$

Obviously, $f'(\alpha) = 0$ and $q(\alpha) = 3h(\alpha) \neq 0$ and therefore α is a root of $f'(x)$ of multiplicity 2.

Differentiating a second time, we get

$$f''(x) = (x-\alpha)r(x),$$

such that $r(\alpha \neq 0)$ and thus, α is a simple root of $f''(x)$.

Multiple roots

Differentiating a third time, we obtain that $f'''(\alpha) \neq 0$.

Multiple roots

Differentiating a third time, we obtain that $f'''(\alpha) \neq 0$.

Thus, we if have for a root of $f(x)$ of multiplicity 3, then

$$\begin{aligned} f(\alpha) &= 0, \\ f'(\alpha) &= 0, \\ f''(\alpha) &= 0, \\ f'''(\alpha) &\neq 0. \end{aligned}$$

Multiple roots

Differentiating a third time, we obtain that $f'''(\alpha) \neq 0$.

Thus, we if have for a root of $f(x)$ of multiplicity 3, then

$$\begin{aligned} f(\alpha) &= 0, \\ f'(\alpha) &= 0, \\ f''(\alpha) &= 0, \\ f'''(\alpha) &\neq 0. \end{aligned}$$

We can prove the following generalizing theorem:

Theorem

Number α is a root of $f(x)$ of multiplicity m if and only if

$$\begin{aligned} &f(\alpha) = f'(\alpha) = f''(\alpha) = \ldots = f^{(m-1)}(\alpha) = 0, \\ &f^{(m)}(\alpha) \neq 0. \end{aligned}$$

Difficulties in multiple roots calculation

In the case of multiple roots ($m > 1$) we distinguish two main difficulties in applying numerical methods for rootfinding:

1. Newton's and secant methods will converge more slowly than for the case of a simple root.

Difficulties in multiple roots calculation

In the case of multiple roots ($m > 1$) we distinguish two main difficulties in applying numerical methods for rootfinding:

1. Newton's and secant methods will converge more slowly than for the case of a simple root.
2. Due to the noise in function evaluation there will be a large interval of uncertainty for the location of a multiple root.

Difficulties in multiple roots calculation

In the case of multiple roots ($m > 1$) we distinguish two main difficulties in applying numerical methods for rootfinding:

1. Newton's and secant methods will converge more slowly than for the case of a simple root.
2. Due to the noise in function evaluation there will be a large interval of uncertainty for the location of a multiple root.

Difficulties in multiple roots calculation

In the case of multiple roots ($m > 1$) we distinguish two main difficulties in applying numerical methods for rootfinding:

1. Newton's and secant methods will converge more slowly than for the case of a simple root.
2. Due to the noise in function evaluation there will be a large interval of uncertainty for the location of a multiple root.

Newton's method can be viewed as a fixed point iteration method:

$$x_n = g(x_{n-1}) \quad \text{with} \quad g(x) = x - \frac{f(x)}{f'(x)}.$$

On the other hand, since α is a root of multiplicity m, we have $f(x) = (x-\alpha)^m h(x)$. Substitute this expression in the last formula to get

$$\begin{aligned} g(x) &= x - \frac{(x-\alpha)^m h(x)}{m(x-\alpha)^{m-1}h(x) + (x-\alpha)^m h'(x)} \\ &= x - \frac{(x-\alpha)h(x)}{mh(x) + (x-\alpha)h'(x)}. \end{aligned}$$

Difficulties in multiple roots calculation

Differentiate $g(x)$ and evaluate it at α to obtain

$$g'(\alpha) = \frac{m-1}{m},$$

where m is the multiplicity of α.

Difficulties in multiple roots calculation

Differentiate $g(x)$ and evaluate it at α to obtain

$$g'(\alpha) = \frac{m-1}{m},$$

where m is the multiplicity of α.

Observe that in this case $\lambda = g'(\alpha) \neq 0$ and therefore Newton's method will be only linear convergent:

$$\alpha - x_n \approx \lambda(\alpha - x_{n-1}) \quad \text{with} \quad \lambda = \frac{m-1}{m}.$$

Difficulties in multiple roots calculation

Differentiate $g(x)$ and evaluate it at α to obtain

$$g'(\alpha) = \frac{m-1}{m},$$

where m is the multiplicity of α.

Observe that in this case $\lambda = g'(\alpha) \neq 0$ and therefore Newton's method will be only linear convergent:

$$\alpha - x_n \approx \lambda(\alpha - x_{n-1}) \quad \text{with} \quad \lambda = \frac{m-1}{m}.$$

Same result can be proven for secant method as well.

Difficulties in multiple roots calculation

Differentiate $g(x)$ and evaluate it at α to obtain

$$g'(\alpha) = \frac{m-1}{m},$$

where m is the multiplicity of α.

Observe that in this case $\lambda = g'(\alpha) \neq 0$ and therefore Newton's method will be only linear convergent:

$$\alpha - x_n \approx \lambda(\alpha - x_{n-1}) \quad \text{with} \quad \lambda = \frac{m-1}{m}.$$

Same result can be proven for secant method as well.

Is it possible to accelerate the convergence of Newton's and secant method in case of multiple roots?
For example, to have a quadratic convergence as in the case of simple roots?

Difficulties in multiple roots calculation

Let α be a root of $f(x)$ of multiplicity $m > 1$.
Then, there are two possibilities:

1. Instead of solving $f(x) = 0$, solve $f^{(m-1)}(x) = 0$ for which α will be a simple root (of multiplicity 1).

Difficulties in multiple roots calculation

Let α be a root of $f(x)$ of multiplicity $m > 1$.
Then, there are two possibilities:

1. Instead of solving $f(x) = 0$, solve $f^{(m-1)}(x) = 0$ for which α will be a simple root (of multiplicity 1).
2. Consider the fixed iteration method

$$x_n = g(x_{n-1}) \quad \text{with} \quad g(x) = x - m\frac{f(x)}{f'(x)}$$

for which $g'(\alpha) = 0$ and the iteration method will converge quadratically.

Difficulties in multiple roots calculation

Let α be a root of $f(x)$ of multiplicity $m > 1$.
Then, there are two possibilities:

1. Instead of solving $f(x) = 0$, solve $f^{(m-1)}(x) = 0$ for which α will be a simple root (of multiplicity 1).
2. Consider the fixed iteration method

$$x_n = g(x_{n-1}) \quad \text{with} \quad g(x) = x - m\frac{f(x)}{f'(x)}$$

for which $g'(\alpha) = 0$ and the iteration method will converge quadratically.

Difficulties in multiple roots calculation

Let α be a root of $f(x)$ of multiplicity $m > 1$.
Then, there are two possibilities:

1. Instead of solving $f(x) = 0$, solve $f^{(m-1)}(x) = 0$ for which α will be a simple root (of multiplicity 1).
2. Consider the fixed iteration method

$$x_n = g(x_{n-1}) \quad \text{with} \quad g(x) = x - m\frac{f(x)}{f'(x)}$$

   for which $g'(\alpha) = 0$ and the iteration method will converge quadratically.

Both possibilities presume that multiplicity m is known in advance.

Difficulties in multiple roots calculation

Let α be a root of $f(x)$ of multiplicity $m > 1$.
Then, there are two possibilities:

1. Instead of solving $f(x) = 0$, solve $f^{(m-1)}(x) = 0$ for which α will be a simple root (of multiplicity 1).
2. Consider the fixed iteration method

$$x_n = g(x_{n-1}) \quad \text{with} \quad g(x) = x - m\frac{f(x)}{f'(x)}$$

   for which $g'(\alpha) = 0$ and the iteration method will converge quadratically.

Both possibilities presume that multiplicity m is known in advance.

Recall that λ_n (that are computable) converge to $\lambda = \frac{m-1}{m}$.

Therefore, m can be guessed from numerical computations.

Noise in function evaluation for multiple roots

Recall that in evaluating a function, due to rounding/chopping errors, its graph is rather a band with random noise rather than a smooth curve. In the case of multiple roots, this will affect the accuracy of computation in a greater manner than in the case of simple roots.

Noise in function evaluation for multiple roots

Multiple root. Example

Example

Let $f(x) = 2.7951 - 8.954x + 10.56x^2 - 5.4x^3 + x^4$ and α its root on interval $[0, 2]$.
Apply Newton's method with initial guess $x_0 = 0.8$.
Numerical results are presented below.

n	x_n	$f(x_n)$	$x_n - x_{n-1}$	λ_n
0	0.800000	0.03510		
1	0.892857	0.01073	0.092857	
2	0.958176	0.00325	0.065319	0.7034
3	1.00344	0.00099	0.045264	0.6930
4	1.03486	0.00029	0.03142	0.6942
5	1.05581	0.00009	0.02095	0.6668
6	1.07028	0.00003	0.01447	0.6907
7	1.08092	0.0	0.01064	0.7353

Multiple root. Example

Observe that convergence is quite slow.

Multiple root. Example

Observe that convergence is quite slow.

Numerical results are in accordance with linear convergence, since $\lambda_n = \frac{x_n - x_{n-1}}{x_{n-1} - x_{n-2}}$ are in interval $[0.65, 0.75]$ and thus $\lambda = g'(\alpha) \neq 0$:

$$\alpha - x_n \approx \lambda(\alpha - x_{n-1}).$$

Also, observe that the convergence is even slower than bisection method, since for bisection method the linear rate is 0.5, whilst in this case the rate is approximately 0.7.

Multiple root. Example

Observe that convergence is quite slow.

Numerical results are in accordance with linear convergence, since $\lambda_n = \frac{x_n - x_{n-1}}{x_{n-1} - x_{n-2}}$ are in interval $[0.65, 0.75]$ and thus $\lambda = g'(\alpha) \neq 0$:

$$\alpha - x_n \approx \lambda(\alpha - x_{n-1}).$$

Also, observe that the convergence is even slower than bisection method, since for bisection method the linear rate is 0.5, whilst in this case the rate is approximately 0.7.

The explanation for this behavior is probably the multiplicity $m > 1$ of the root. If this is the case then $\lambda \approx \frac{m-1}{m}$ and we can guess from numerical results that $m = 3$.

Multiple root. Example

Observe that convergence is quite slow.

Numerical results are in accordance with linear convergence, since $\lambda_n = \frac{x_n - x_{n-1}}{x_{n-1} - x_{n-2}}$ are in interval $[0.65, 0.75]$ and thus $\lambda = g'(\alpha) \neq 0$:

$$\alpha - x_n \approx \lambda(\alpha - x_{n-1}).$$

Also, observe that the convergence is even slower than bisection method, since for bisection method the linear rate is 0.5, whilst in this case the rate is approximately 0.7.

The explanation for this behavior is probably the multiplicity $m > 1$ of the root. If this is the case then $\lambda \approx \frac{m-1}{m}$ and we can guess from numerical results that $m = 3$.

Newton's method applied to $f''(x) = 21.12 - 32.4x + 12x^2$ with initial guess $x_0 = 1$ will lead to a rapid convergence to $\alpha = 1.1$.

Stability of the roots

Consider polynomial

$$\begin{aligned} f(x) &= x^7 - 28x^6 + 322x^5 - 1960x^4 + 6769x^3 - 13132x^2 \\ &\quad + 13068x - 5040 \\ &= (x-1)(x-2)(x-3)(x-4)(x-5)(x-6)(x-7). \end{aligned}$$

This polynomial has exactly 7 simple roots: 1, 2, 3, 4, 5, 6, 7.

Stability of the roots

Consider polynomial

$$\begin{aligned} f(x) &= x^7 - 28x^6 + 322x^5 - 1960x^4 + 6769x^3 - 13132x^2 \\ &\quad + 13068x - 5040 \\ &= (x-1)(x-2)(x-3)(x-4)(x-5)(x-6)(x-7). \end{aligned}$$

This polynomial has exactly 7 simple roots: 1, 2, 3, 4, 5, 6, 7.

Now, consider **the perturbed** polynomial:

$$\begin{aligned} F(x) &= x^7 - 28.002x^6 + 322x^5 - 1960x^4 + 6769x^3 - 13132x^2 \\ &\quad + 13068x - 5040. \end{aligned}$$

As it can be seen, this is a relatively small change in one coefficient with a relative error of

$$\mathrm{Rel}(28) = \frac{28 - 28.002}{28} = -\frac{0.002}{28} \approx 7.14e - 05.$$

Stability of the roots

Consider polynomial

$$\begin{aligned} f(x) &= x^7 - 28x^6 + 322x^5 - 1960x^4 + 6769x^3 - 13132x^2 \\ &\quad + 13068x - 5040 \\ &= (x-1)(x-2)(x-3)(x-4)(x-5)(x-6)(x-7). \end{aligned}$$

This polynomial has exactly 7 simple roots: 1, 2, 3, 4, 5, 6, 7.

Now, consider **the perturbed** polynomial:

$$\begin{aligned} F(x) &= x^7 - 28.002x^6 + 322x^5 - 1960x^4 + 6769x^3 - 13132x^2 \\ &\quad + 13068x - 5040. \end{aligned}$$

As it can be seen, this is a relatively small change in one coefficient with a relative error of

$$\mathrm{Rel}(28) = \frac{28 - 28.002}{28} = -\frac{0.002}{28} \approx 7.14e - 05.$$

How this small error in one coefficient changed the roots of $f(x)$?

Stability of the roots

Roots of $f(x)$	Roots of $F(x)$	Error
1	1.0000028	$-2.8e-06$
2	1.9989382	$+1.1e-03$
3	3.0331253	$-3.3e-02$
4	3.8195692	$+1.8e-01$
5	$5.4586758 + 0.54012578i$	$-0.46 - 0.54i$
6	$5.4586758 - 0.54012578i$	$-0.46 + 0.54i$
7	7.2330128	$+2.3e-01$

Stability of the roots

Roots of $f(x)$	Roots of $F(x)$	Error
1	1.0000028	$-2.8e-06$
2	1.9989382	$+1.1e-03$
3	3.0331253	$-3.3e-02$
4	3.8195692	$+1.8e-01$
5	$5.4586758 + 0.54012578i$	$-0.46 - 0.54i$
6	$5.4586758 - 0.54012578i$	$-0.46 + 0.54i$
7	7.2330128	$+2.3e-01$

It can be seen that a small error (perturbation) (of 10^{-5}) in one coefficient lead to much bigger errors in the roots of perturbed polynomial. Moreover, two of the roots now are complex numbers.

Stability of the roots

Roots of $f(x)$	Roots of $F(x)$	Error
1	1.0000028	$-2.8e-06$
2	1.9989382	$+1.1e-03$
3	3.0331253	$-3.3e-02$
4	3.8195692	$+1.8e-01$
5	$5.4586758 + 0.54012578i$	$-0.46 - 0.54i$
6	$5.4586758 - 0.54012578i$	$-0.46 + 0.54i$
7	7.2330128	$+2.3e-01$

It can be seen that a small error (perturbation) (of 10^{-5}) in one coefficient lead to much bigger errors in the roots of perturbed polynomial. Moreover, two of the roots now are complex numbers.

This phenomena (small changes in input lead to big changes in output) is an example of so-called **unstable** or **ill-conditioned** rootfinding problem.

Stability analysis

Suppose that for a given function $f(x)$ there is perturbed function

$$F(x) = f(x) + \varepsilon g(x).$$

For example, for the previous case $g(x) = x^6$ and $\varepsilon = -0.002$.

Stability analysis

Suppose that for a given function $f(x)$ there is perturbed function

$$F(x) = f(x) + \varepsilon g(x).$$

For example, for the previous case $g(x) = x^6$ and $\varepsilon = -0.002$.

Let α be a simple root of $f(x)$ and α_ε will be a root of $F(x)$ close to α. It can be shown that

$$\alpha - \alpha_\varepsilon \approx \varepsilon \frac{g(\alpha)}{f'(\alpha)}.$$

Stability analysis

Suppose that for a given function $f(x)$ there is perturbed function

$$F(x) = f(x) + \varepsilon g(x).$$

For example, for the previous case $g(x) = x^6$ and $\varepsilon = -0.002$.

Let α be a simple root of $f(x)$ and α_ε will be a root of $F(x)$ close to α. It can be shown that

$$\alpha - \alpha_\varepsilon \approx \varepsilon \frac{g(\alpha)}{f'(\alpha)}.$$

This last approximation allows us to estimate the changes in the root $\alpha - \alpha_\varepsilon$, if a small error ε is being introduced. Thus, it can be considered as an indicator of stability. For our example, $\alpha = 3$ and $\varepsilon = -0.002$:

$$3 - \alpha_\varepsilon \approx 0.002 \cdot \frac{3^6}{48} \approx 0.0304$$

and we can see that it is close to actual error of 0.033.

INTERPOLATION

Interpolation is a process of finding a formula (often a polynomial) whose graph will pass through a given set of points (x, y).

As an example, consider defining

$$x_0 = 0, \qquad x_1 = \frac{\pi}{4}, \qquad x_2 = \frac{\pi}{2}$$

and

$$y_i = \cos x_i, \qquad i = 0, 1, 2$$

This gives us the three points

$$(0, 1)\,, \quad \left(\frac{\pi}{4}, \frac{1}{\mathrm{sqrt}(2)}\right), \quad \left(\frac{\pi}{2}, 0\right)$$

Now find a quadratic polynomial

$$p(x) = a_0 + a_1 x + a_2 x^2$$

for which

$$p(x_i) = y_i, \quad i = 0, 1, 2$$

The graph of this polynomial is shown on the accompanying graph. We later give an explicit formula.

Quadratic interpolation of cos(x)
y
x
π/4
π/2
y = cos(x)
y = p2(x)

PURPOSES OF INTERPOLATION

1. Replace a set of data points $\{(x_i, y_i)\}$ with a function given analytically.

2. Approximate functions with simpler ones, usually polynomials or 'piecewise polynomials'.

Purpose #1 has several aspects.

- The data may be from a known class of functions. Interpolation is then used to find the member of this class of functions that agrees with the given data. For example, data may be generated from functions of the form

  $$p(x) = a_0 + a_1 e^x + a_2 e^{2x} + \cdots + a_n e^{nx}$$

  Then we need to find the coefficients $\left\{a_j\right\}$ based on the given data values.

- We may want to take function values $f(x)$ given in a table for selected values of x, often equally spaced, and extend the function to values of x not in the table.

  For example, given numbers from a table of logarithms, estimate the logarithm of a number x not in the table.

- Given a set of data points $\{(x_i, y_i)\}$, find a curve passing thru these points that is "pleasing to the eye". In fact, this is what is done continually with computer graphics. How do we connect a set of points to make a smooth curve? Connecting them with straight line segments will often give a curve with many corners, whereas what was intended was a smooth curve.

Purpose #2 for interpolation is to approximate functions $f(x)$ by simpler functions $p(x)$, perhaps to make it easier to integrate or differentiate $f(x)$. That will be the primary reason for studying interpolation in this course.

As as example of why this is important, consider the problem of evaluating

$$I = \int_0^1 \frac{dx}{1 + x^{10}}$$

This is very difficult to do analytically. But we will look at producing polynomial interpolants of the integrand; and polynomials are easily integrated exactly.

We begin by using polynomials as our means of doing interpolation. Later in the chapter, we consider more complex 'piecewise polynomial' functions, often called 'spline functions'.

LINEAR INTERPOLATION

The simplest form of interpolation is probably the straight line, connecting two points by a straight line. Let two data points (x_0, y_0) and (x_1, y_1) be given. There is a unique straight line passing through these points. We can write the formula for a straight line as

$$P_1(x) = a_0 + a_1 x$$

In fact, there are other more convenient ways to write it, and we give several of them below.

$$\begin{aligned} P_1(x) &= \frac{x - x_1}{x_0 - x_1} y_0 + \frac{x - x_0}{x_1 - x_0} y_1 \\ &= \frac{(x_1 - x)\, y_0 + (x - x_0)\, y_1}{x_1 - x_0} \\ &= y_0 + \frac{x - x_0}{x_1 - x_0} [y_1 - y_0] \\ &= y_0 + \left(\frac{y_1 - y_0}{x_1 - x_0} \right) (x - x_0) \end{aligned}$$

Check each of these by evaluating them at $x = x_0$ and x_1 to see if the respective values are y_0 and y_1.

Example. Following is a table of values for $f(x) = \tan x$ for a few values of x.

x	1	1.1	1.2	1.3
$\tan x$	1.5574	1.9648	2.5722	3.6021

Use linear interpolation to estimate tan(1.15). Then use

$$x_0 = 1.1, \quad x_1 = 1.2$$

with corresponding values for y_0 and y_1. Then

$$\tan x \approx y_0 + \frac{x - x_0}{x_1 - x_0}[y_1 - y_0]$$

$$\begin{aligned} \tan x &\approx y_0 + \frac{x - x_0}{x_1 - x_0}[y_1 - y_0] \\ \tan(1.15) &\approx 1.9648 + \frac{1.15 - 1.1}{1.2 - 1.1}[2.5722 - 1.9648] \\ &= 2.2685 \end{aligned}$$

The true value is $\tan 1.15 = 2.2345$. We will want to examine formulas for the error in interpolation, to know when we have sufficient accuracy in our interpolant.

y
y=tan(x)
1
1.3
x

y
y = tan(x)
y = p1(x)
1.1
1.2
x

QUADRATIC INTERPOLATION

We want to find a polynomial

$$P_2(x) = a_0 + a_1 x + a_2 x^2$$

which satisfies

$$P_2(x_i) = y_i, \qquad i = 0, 1, 2$$

for given data points $(x_0, y_0), (x_1, y_1), (x_2, y_2)$. One formula for such a polynomial follows:

$$P_2(x) = y_0 L_0(x) + y_1 L_1(x) + y_2 L_2(x) \qquad (**)$$

with

$$L_0(x) = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}, \qquad L_1(x) = \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}$$

$$L_2(x) = \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}$$

The formula $(**)$ is called *Lagrange's form* of the interpolation polynomial.

LAGRANGE BASIS FUNCTIONS

The functions

$$L_0(x) = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}, \qquad L_1(x) = \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}$$

$$L_2(x) = \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}$$

are called 'Lagrange basis functions' for quadratic interpolation. They have the properties

$$L_i(x_j) = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$$

for $i, j = 0, 1, 2$. Also, they all have degree 2. Their graphs are on an accompanying page.

As a consequence of each $L_i(x)$ being of degree 2, we have that the interpolant

$$P_2(x) = y_0 L_0(x) + y_1 L_1(x) + y_2 L_2(x)$$

must have degree ≤ 2.

UNIQUENESS

Can there be another polynomial, call it $Q(x)$, for which

$$\deg(Q) \leq 2$$
$$Q(x_i) = y_i, \qquad i = 0, 1, 2$$

Thus, is the Lagrange formula $P_2(x)$ unique?

Introduce

$$R(x) = P_2(x) - Q(x)$$

From the properties of P_2 and Q, we have $\deg(R) \leq 2$. Moreover,

$$R(x_i) = P_2(x_i) - Q(x_i) = y_i - y_i = 0$$

for all three node points x_0, x_1, and x_2. How many polynomials $R(x)$ are there of degree at most 2 and having three distinct zeros? The answer is that only the zero polynomial satisfies these properties, and therefore

$$R(x) = 0 \quad \text{for all } x$$

$$Q(x) = P_2(x) \quad \text{for all } x$$

SPECIAL CASES

Consider the data points

$$(x_0, 1), (x_1, 1), (x_2, 1)$$

What is the polynomial $P_2(x)$ in this case?

Answer: We must have the polynomial interpolant is

$$P_2(x) \equiv 1$$

meaning that $P_2(x)$ is the constant function. Why? First, the constant function satisfies the property of being of degree ≤ 2. Next, it clearly interpolates the given data. Therefore by the uniqueness of quadratic interpolation, $P_2(x)$ must be the constant function 1.

Consider now the data points

$$(x_0, mx_0), (x_1, mx_1), (x_2, mx_2)$$

for some constant m. What is $P_2(x)$ in this case? By an argument similar to that above,

$$P_2(x) = mx \quad \text{for all } x$$

Thus the degree of $P_2(x)$ can be less than 2.

HIGHER DEGREE INTERPOLATION

We consider now the case of interpolation by polynomials of a general degree n. We want to find a polynomial $P_n(x)$ for which

$$\begin{aligned} &\deg(P_n) \leq n \\ P_n(x_i) = y_i, \qquad &i = 0, 1, \cdots, n \end{aligned} \qquad (**)$$

with given data points

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

The solution is given by <u>Lagrange's formula</u>

$$P_n(x) = y_0 L_0(x) + y_1 L_1(x) + \cdots + y_n L_n(x)$$

The Lagrange basis functions are given by

$$L_k(x) = \frac{(x - x_0) .. (x - x_{k-1})(x - x_{k+1}) .. (x - x_n)}{(x_k - x_0) .. (x_k - x_{k-1})(x_k - x_{k+1}) .. (x_k - x_n)}$$

for $k = 0, 1, 2, ..., n$. The quadratic case was covered earlier.

In a manner analogous to the quadratic case, we can show that the above $P_n(x)$ is the only solution to the problem $(**)$.

In the formula

$$L_k(x) = \frac{(x - x_0)\,..(x - x_{k-1})(x - x_{k+1})..\,(x - x_n)}{(x_k - x_0)\,..(x_k - x_{k-1})(x_k - x_{k+1})..\,(x_k - x_n)}$$

we can see that each such function is a polynomial of degree n. In addition,

$$L_k(x_i) = \begin{cases} 1, & k = i \\ 0, & k \neq i \end{cases}$$

Using these properties, it follows that the formula

$$P_n(x) = y_0 L_0(x) + y_1 L_1(x) + \cdots + y_n L_n(x)$$

satisfies the interpolation problem of finding a solution to

$$\begin{aligned} \deg(P_n) &\leq n \\ P_n(x_i) = y_i, \qquad & i = 0, 1, \cdots, n \end{aligned}$$

EXAMPLE

Recall the table

x	1	1.1	1.2	1.3
$\tan x$	1.5574	1.9648	2.5722	3.6021

We now interpolate this table with the nodes

$$x_0 = 1,\ x_1 = 1.1,\ x_2 = 1.2,\ x_3 = 1.3$$

Without giving the details of the evaluation process, we have the following results for interpolation with degrees $n = 1, 2, 3$.

n	1	2	3
$P_n(1.15)$	2.2685	2.2435	2.2296
$Error$	$-.0340$	$-.0090$	.0049

It improves with increasing degree n, but not at a very rapid rate. In fact, the error becomes worse when n is increased further. Later we will see that interpolation of a much higher degree, say $n \geq 10$, is often poorly behaved when the node points $\{x_i\}$ are evenly spaced.

A FIRST ORDER DIVIDED DIFFERENCE

For a given function $f(x)$ and two distinct points x_0 and x_1, define

$$f[x_0, x_1] = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

This is called a *first order divided difference* of $f(x)$.

By the Mean-value theorem,

$$f(x_1) - f(x_0) = f'(c)\,(x_1 - x_0)$$

for some c between x_0 and x_1. Thus

$$f[x_0, x_1] = f'(c)$$

and the divided difference in very much like the derivative, especially if x_0 and x_1 are quite close together. In fact,

$$f'\left(\frac{x_1 + x_0}{2}\right) \approx f[x_0, x_1]$$

is quite an accurate approximation of the derivative (see §5.4).

SECOND ORDER DIVIDED DIFFERENCES

Given three distinct points x_0, x_1, and x_2, define

$$f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0}$$

This is called the *second order divided difference* of $f(x)$.

By a fairly complicated argument, we can show

$$f[x_0, x_1, x_2] = \frac{1}{2} f''(c)$$

for some c intermediate to x_0, x_1, and x_2. In fact, as we investigate in §5.4,

$$f''(x_1) \approx 2f[x_0, x_1, x_2]$$

in the case the nodes are evenly spaced,

$$x_1 - x_0 = x_2 - x_1$$

EXAMPLE

Consider the table

x	1	1.1	1.2	1.3	1.4
$\cos x$	.54030	.45360	.36236	.26750	.16997

Let $x_0 = 1$, $x_1 = 1.1$, and $x_2 = 1.2$. Then

$$\begin{aligned} f[x_0, x_1] &= \frac{.45360 - .54030}{1.1 - 1} = -.86700 \\ f[x_1, x_2] &= \frac{.36236 - .45360}{1.1 - 1} = -.91240 \end{aligned}$$

$$\begin{aligned} f[x_0, x_1, x_2] &= \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} \\ &= \frac{-.91240 - (-.86700)}{1.2 - 1.0} = -.22700 \end{aligned}$$

For comparison,

$$\begin{aligned} f'\left(\frac{x_1 + x_0}{2}\right) &= -\sin(1.05) = -.86742 \\ \frac{1}{2} f''(x_1) &= -\frac{1}{2}\cos(1.1) = -.22680 \end{aligned}$$

GENERAL DIVIDED DIFFERENCES

Given $n+1$ distinct points $x_0, ..., x_n$, with $n \geq 2$, define

$$f[x_0, ..., x_n] = \frac{f[x_1, ..., x_n] - f[x_0, ..., x_{n-1}]}{x_n - x_0}$$

This is a recursive definition of the nth-order divided difference of $f(x)$, using divided differences of order n. Its relation to the derivative is as follows:

$$f[x_0, ..., x_n] = \frac{1}{n!} f^{(n)}(c)$$

for some c intermediate to the points $\{x_0, ..., x_n\}$. Let I denote the interval

$$I = [\min\{x_0, ..., x_n\}, \max\{x_0, ..., x_n\}]$$

Then $c \in I$, and the above result is based on the assumption that $f(x)$ is n-times continuously differentiable on the interval I.

EXAMPLE

The following table gives divided differences for the data in

x	1	1.1	1.2	1.3	1.4
$\cos x$	.54030	.45360	.36236	.26750	.16997

For the column headings, we use

$$D^k f(x_i) = f[x_i, ..., x_{i+k}]$$

i	x_i	$f(x_i)$	$Df(x_i)$	$D^2 f(x_i)$	$D^3 f(x_i)$	$D^4 f(x_i)$
0	1.0	.54030	-.8670	-.2270	.1533	.0125
1	1.1	.45360	-.9124	-.1810	.1583	
2	1.2	.36236	-.9486	-.1335		
3	1.3	.26750	-.9753			
4	1.4	.16997				

These were computed using the recursive definition

$$f[x_0, ..., x_n] = \frac{f[x_1, ..., x_n] - f[x_0, ..., x_{n-1}]}{x_n - x_0}$$

ORDER OF THE NODES

Looking at $f[x_0, x_1]$, we have

$$f[x_0, x_1] = \frac{f(x_1) - f(x_0)}{x_1 - x_0} = \frac{f(x_0) - f(x_1)}{x_0 - x_1} = f[x_1, x_0]$$

The order of x_0 and x_1 does not matter. Looking at

$$f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0}$$

we can expand it to get

$$\begin{aligned} f[x_0, x_1, x_2] = & \frac{f(x_0)}{(x_0 - x_1)(x_0 - x_2)} \\ & + \frac{f(x_1)}{(x_1 - x_0)(x_1 - x_2)} + \frac{f(x_2)}{(x_2 - x_0)(x_2 - x_1)} \end{aligned}$$

With this formula, we can show that the order of the arguments x_0, x_1, x_2 does not matter in the final value of $f[x_0, x_1, x_2]$ we obtain. Mathematically,

$$f[x_0, x_1, x_2] = f[x_{i_0}, x_{i_1}, x_{i_2}]$$

for any permutation (i_0, i_1, i_2) of $(0, 1, 2)$.

We can show in general that the value of $f[x_0, ..., x_n]$ is independent of the order of the arguments $\{x_0, ..., x_n\}$, even though the intermediate steps in its calculations using

$$f[x_0, ..., x_n] = \frac{f[x_1, ..., x_n] - f[x_0, ..., x_{n-1}]}{x_n - x_0}$$

are order dependent.

We can show

$$f[x_0, ..., x_n] = f[x_{i_0}, ..., x_{i_n}]$$

for any permutation $(i_0, i_1, ..., i_n)$ of $(0, 1, ..., n)$.

COINCIDENT NODES

What happens when some of the nodes $\{x_0, ..., x_n\}$ are not distinct. Begin by investigating what happens when they all come together as a single point x_0.

For first order divided differences, we have

$$\lim_{x_1 \to x_0} f[x_0, x_1] = \lim_{x_1 \to x_0} \frac{f(x_1) - f(x_0)}{x_1 - x_0} = f'(x_0)$$

We extend the definition of $f[x_0, x_1]$ to coincident nodes using

$$f[x_0, x_0] = f'(x_0)$$

For second order divided differences, recall

$$f[x_0, x_1, x_2] = \frac{1}{2} f''(c)$$

with c intermediate to x_0, x_1, and x_2.

Then as $x_1 \to x_0$ and $x_2 \to x_0$, we must also have that $c \to x_0$. Therefore,

$$\lim_{\substack{x_1 \to x_0 \\ x_2 \to x_0}} f[x_0, x_1, x_2] = \frac{1}{2} f''(x_0)$$

We therefore define

$$f[x_0, x_0, x_0] = \frac{1}{2} f''(x_0)$$

For the case of general $f[x_0, ..., x_n]$, recall that

$$f[x_0, ..., x_n] = \frac{1}{n!} f^{(n)}(c)$$

for some c intermediate to $\{x_0, ..., x_n\}$. Then

$$\lim_{\{x1,...,x_n\} \to x_0} f[x_0, ..., x_n] = \frac{1}{n!} f^{(n)}(x_0)$$

and we define

$$f[\underbrace{x_0, ..., x_0}_{n+1 \text{ times}}] = \frac{1}{n!} f^{(n)}(x_0)$$

What do we do when only some of the nodes are coincident. This too can be dealt with, although we do so here only by examples.

$$\begin{aligned} f[x_0, x_1, x_1] &= \frac{f[x_1, x_1] - f[x_0, x_1]}{x_1 - x_0} \\ &= \frac{f'(x_1) - f[x_0, x_1]}{x_1 - x_0} \end{aligned}$$

The recursion formula can be used in general in this way to allow all possible combinations of possibly coincident nodes.

LAGRANGE'S FORMULA FOR THE INTERPOLATION POLYNOMIAL

Recall the general interpolation problem: find a polynomial $P_n(x)$ for which

$$\begin{aligned} \deg(P_n) &\le n \\ P_n(x_i) = y_i, &\qquad i = 0, 1, \cdots, n \end{aligned}$$

with given data points

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

and with $\{x_0, ..., x_n\}$ distinct points.

In §5.1, we gave the solution as *Lagrange's formula*

$$P_n(x) = y_0 L_0(x) + y_1 L_1(x) + \cdots + y_n L_n(x)$$

with $\{L_0(x), ..., L_n(x)\}$ the Lagrange basis polynomials. Each L_j is of degree n and it satisfies

$$L_j(x_i) = \begin{cases} 1, & j = i \\ 0, & j \ne i \end{cases}$$

for $i = 0, 1, ..., n$.

THE NEWTON DIVIDED DIFFERENCE FORM OF THE INTERPOLATION POLYNOMIAL

Let the data values for the problem

$$\begin{gathered} \deg(P_n) \leq n \\ P_n(x_i) = y_i, \qquad i = 0, 1, \cdots, n \end{gathered}$$

be generated from a function $f(x)$:

$$y_i = f(x_i), \qquad i = 0, 1, ..., n$$

Using the divided differences

$$f[x_0, x_1],\ f[x_0, x_1, x_2], ..., f[x_0, ..., x_n]$$

we can write the interpolation polynomials

$$P_1(x),\ P_2(x), ..., P_n(x)$$

in a way that is simple to compute.

$$\begin{aligned} P_1(x) &= f(x_0) + f[x_0, x_1]\,(x - x_0) \\ P_2(x) &= f(x_0) + f[x_0, x_1]\,(x - x_0) \\ &\qquad + f[x_0, x_1, x_2]\,(x - x_0)\,(x - x_1) \\ &= P_1(x) + f[x_0, x_1, x_2]\,(x - x_0)\,(x - x_1) \end{aligned}$$

For the case of the general problem

$$\begin{aligned} \deg(P_n) &\leq n \\ P_n(x_i) = y_i, \qquad & i = 0, 1, \cdots, n \end{aligned}$$

we have

$$\begin{aligned} P_n(x) \;=\; & f(x_0) + f[x_0, x_1]\,(x - x_0) \\ & + f[x_0, x_1, x_2]\,(x - x_0)\,(x - x_1) \\ & + f[x_0, x_1, x_2, x_3]\,(x - x_0)\,(x - x_1)\,(x - x_2) \\ & + \cdots \\ & + f[x_0, ..., x_n]\,(x - x_0) \cdots (x - x_{n-1}) \end{aligned}$$

From this we have the recursion relation

$$P_n(x) = P_{n-1}(x) + f[x_0, ..., x_n]\,(x - x_0) \cdots (x - x_{n-1})$$

in which $P_{n-1}(x)$ interpolates $f(x)$ at the points in $\{x_0, ..., x_{n-1}\}$.

Example: Recall the table

i	x_i	$f(x_i)$	$Df(x_i)$	$D^2f(x_i)$	$D^3f(x_i)$	$D^4f(x_i)$
0	1.0	.54030	-.8670	-.2270	.1533	.0125
1	1.1	.45360	-.9124	-.1810	.1583	
2	1.2	.36236	-.9486	-.1335		
3	1.3	.26750	-.9753			
4	1.4	.16997				

with $D^k f(x_i) = f[x_i, ..., x_{i+k}], \quad k = 1, 2, 3, 4$. Then

$$\begin{aligned}
P_1(x) &= .5403 - .8670\,(x-1) \\
P_2(x) &= P_1(x) - .2270\,(x-1)\,(x-1.1) \\
P_3(x) &= P_2(x) + .1533\,(x-1)\,(x-1.1)\,(x-1.2) \\
P_4(x) &= P_3(x) \\
&\quad +.0125\,(x-1)\,(x-1.1)\,(x-1.2)\,(x-1.3)
\end{aligned}$$

Using this table and these formulas, we have the following table of interpolants for the value $x = 1.05$. The true value is $\cos(1.05) = .49757105$.

n	1	2	3	4
$P_n(1.05)$	.49695	.49752	.49758	.49757
$Error$	6.20E−4	5.00E−5	−1.00E−5	0.0

EVALUATION OF THE DIVIDED DIFFERENCE INTERPOLATION POLYNOMIAL

Let

$$\begin{aligned} d_1 &= f[x_0, x_1] \\ d_2 &= f[x_0, x_1, x_2] \\ &\vdots \\ d_n &= f[x_0, ..., x_n] \end{aligned}$$

Then the formula

$$\begin{aligned} P_n(x) &= f(x_0) + f[x_0, x_1]\,(x - x_0) \\ &\quad + f[x_0, x_1, x_2]\,(x - x_0)\,(x - x_1) \\ &\quad + f[x_0, x_1, x_2, x_3]\,(x - x_0)\,(x - x_1)\,(x - x_2) \\ &\quad + \cdots \\ &\quad + f[x_0, ..., x_n]\,(x - x_0) \cdots (x - x_{n-1}) \end{aligned}$$

can be written as

$$\begin{aligned} P_n(x) &= f(x_0) + (x - x_0)\,(d_1 + (x - x_1)\,(d_2 + \cdots \\ &\quad + (x - x_{n-2})\,(d_{n-1} + (x - x_{n-1})\,d_n) \cdots) \end{aligned}$$

Thus we have a nested polynomial evaluation, and this is quite efficient in computational cost.

ERROR IN LINEAR INTERPOLATION

Let $P_1(x)$ denote the linear polynomial interpolating $f(x)$ at x_0 and x_1, with $f(x)$ a given function (e.g. $f(x) = \cos x$). What is the error $f(x) - P_1(x)$?

Let $f(x)$ be twice continuously differentiable on an interval $[a, b]$ which contains the points $\{x_0, x_1\}$. Then for $a \le x \le b$,

$$f(x) - P_1(x) = \frac{(x - x_0)(x - x_1)}{2} f''(c_x)$$

for some c_x between the minimum and maximum of x_0, x_1, and x.

If x_1 and x are 'close to x_0', then

$$f(x) - P_1(x) \approx \frac{(x - x_0)(x - x_1)}{2} f''(x_0)$$

Thus the error acts like a quadratic polynomial, with zeros at x_0 and x_1.

EXAMPLE

Let $f(x) = \log_{10} x$; and in line with typical tables of $\log_{10} x$, we take $1 \le x, x_0, x_1 \le 10$. For definiteness, let $x_0 < x_1$ with $h = x_1 - x_0$. Then

$$f''(x) = -\frac{\log_{10} e}{x^2}$$

$$\begin{aligned} \log_{10} x - P_1(x) &= \frac{(x - x_0)(x - x_1)}{2}\left[-\frac{\log_{10} e}{c_x^2}\right] \\ &= (x - x_0)(x_1 - x)\left[\frac{\log_{10} e}{2c_x^2}\right] \end{aligned}$$

We usually are interpolating with $x_0 \le x \le x_1$; and in that case, we have

$$(x - x_0)(x_1 - x) \ge 0, \qquad x_0 \le c_x \le x_1$$

$$(x - x_0)(x_1 - x) \geq 0, \qquad x_0 \leq c_x \leq x_1$$

and therefore

$$(x - x_0)(x_1 - x)\left[\frac{\log_{10} e}{2x_1^2}\right] \leq \log_{10} x - P_1(x)$$
$$\leq (x - x_0)(x_1 - x)\left[\frac{\log_{10} e}{2x_0^2}\right]$$

For $h = x_1 - x_0$ small, we have for $x_0 \leq x \leq x_1$

$$\log_{10} x - P_1(x) \approx (x - x_0)(x_1 - x)\left[\frac{\log_{10} e}{2x_0^2}\right]$$

Typical high school algebra textbooks contain tables of $\log_{10} x$ with a spacing of $h = .01$. What is the error in this case? To look at this, we use

$$0 \leq \log_{10} x - P_1(x) \leq (x - x_0)(x_1 - x)\left[\frac{\log_{10} e}{2x_0^2}\right]$$

By simple geometry or calculus,

$$\max_{x_0 \le x \le x_1} (x - x_0)(x_1 - x) \le \frac{h^2}{4}$$

Therefore,

$$0 \le \log_{10} x - P_1(x) \le \frac{h^2}{4}\left[\frac{\log_{10} e}{2x_0^2}\right] \doteq .0543\frac{h^2}{x_0^2}$$

If we want a *uniform bound* for all points $1 \le x_0 \le 10$, we have

$$0 \le \log_{10} x - P_1(x) \le \frac{h^2 \log_{10} e}{8} \doteq .0543h^2$$

$$0 \le \log_{10} x - P_1(x) \le .0543h^2$$

For $h = .01$, as is typical of the high school text book tables of $\log_{10} x$,

$$0 \le \log_{10} x - P_1(x) \le 5.43 \times 10^{-6}$$

If you look at most tables, a typical entry is given to only four decimal places to the right of the decimal point, e.g.

$$\log 5.41 \doteq .7332$$

Therefore the entries are in error by as much as .00005. Comparing this with the interpolation error, we see the latter is less important than the rounding errors in the table entries.

From the bound

$$0 \leq \log_{10} x - P_1(x) \leq \frac{h^2 \log_{10} e}{8x_0^2} \doteq .0543\frac{h^2}{x_0^2}$$

we see the error decreases as x_0 increases, and it is about 100 times smaller for points near 10 than for points near 1.

AN ERROR FORMULA: THE GENERAL CASE

Recall the general interpolation problem: find a polynomial $P_n(x)$ for which $\deg(P_n) \leq n$

$$P_n(x_i) = f(x_i), \qquad i = 0, 1, \cdots, n$$

with distinct node points $\{x_0, ..., x_n\}$ and a given function $f(x)$. Let $[a, b]$ be a given interval on which $f(x)$ is $(n+1)$-times continuously differentiable; and assume the points $x_0, ..., x_n$, and x are contained in $[a, b]$. Then

$$f(x) - P_n(x) = \frac{(x - x_0)(x - x_1) \cdots (x - x_n)}{(n+1)!} f^{(n+1)}(c_x)$$

with c_x some point between the minimum and maximum of the points in $\{x, x_0, ..., x_n\}$.

$$f(x) - P_n(x) = \frac{(x - x_0)(x - x_1)\cdots(x - x_n)}{(n+1)!} f^{(n+1)}(c_x)$$

As shorthand, introduce

$$\Psi_n(x) = (x - x_0)(x - x_1)\cdots(x - x_n)$$

a polynomial of degree $n + 1$ with roots $\{x_0, ..., x_n\}$. Then

$$f(x) - P_n(x) = \frac{\Psi_n(x)}{(n+1)!} f^{(n+1)}(c_x)$$

THE QUADRATIC CASE

For $n = 2$, we have

$$f(x) - P_2(x) = \frac{(x - x_0)(x - x_1)(x - x_2)}{3!} f^{(3)}(c_x) \qquad (*)$$

with c_x some point between the minimum and maximum of the points in $\{x, x_0, x_1, x_2\}$.

To illustrate the use of this formula, consider the case of evenly spaced nodes:

$$x_1 = x_0 + h, \qquad x_2 = x_1 + h$$

Further suppose we have $x_0 \leq x \leq x_2$, as we would usually have when interpolating in a table of given function values (e.g. $\log_{10} x$). The quantity

$$\Psi_2(x) = (x - x_0)(x - x_1)(x - x_2)$$

can be evaluated directly for a particular x.

Graph of

$$\Psi_2(x) = (x+h)\,x\,(x-h)$$

using $(x_0, x_1, x_2) = (-h, 0, h)$:

In the formula $(*)$, however, we do not know c_x, and therefore we replace $\left|f^{(3)}(c_x)\right|$ with a maximum of $\left|f^{(3)}(x)\right|$ as x varies over $x_0 \leq x \leq x_2$. This yields

$$|f(x) - P_2(x)| \leq \frac{|\Psi_2(x)|}{3!} \max_{x_0 \leq x \leq x_2} \left|f^{(3)}(x)\right| \quad (**)$$

If we want a uniform bound for $x_0 \leq x \leq x_2$, we must compute

$$\max_{x_0 \leq x \leq x_2} |\Psi_2(x)| = \max_{x_0 \leq x \leq x_2} |(x - x_0)(x - x_1)(x - x_2)|$$

Using calculus,

$$\max_{x_0 \leq x \leq x_2} |\Psi_2(x)| = \frac{2h^3}{3 \text{ sqrt}(3)}, \qquad \text{at } x = x_1 \pm \frac{h}{\text{sqrt}(3)}$$

Combined with $(**)$, this yields

$$|f(x) - P_2(x)| \leq \frac{h^3}{9 \text{ sqrt}(3)} \max_{x_0 \leq x \leq x_2} \left|f^{(3)}(x)\right|$$

for $x_0 \leq x \leq x_2$.

For $f(x) = \log_{10} x$, with $1 \le x_0 \le x \le x_2 \le 10$, this leads to

$$\begin{aligned} |\log_{10} x - P_2(x)| &\le \frac{h^3}{9 \text{ sqrt}(3)} \cdot \max_{x_0 \le x \le x_2} \frac{2 \log_{10} e}{x^3} \\ &= \frac{.05572\, h^3}{x_0^3} \end{aligned}$$

For the case of $h = .01$, we have

$$|\log_{10} x - P_2(x)| \le \frac{5.57 \times 10^{-8}}{x_0^3} \le 5.57 \times 10^{-8}$$

Question: How much larger could we make h so that quadratic interpolation would have an error comparable to that of linear interpolation of $\log_{10} x$ with $h = .01$? The error bound for the linear interpolation was 5.43×10^{-6}, and therefore we want the same to be true of quadratic interpolation. Using a simpler bound, we want to find h so that

$$|\log_{10} x - P_2(x)| \leq .05572\, h^3 \leq 5 \times 10^{-6}$$

This is true if $h = .04477$. Therefore a spacing of $h = .04$ would be sufficient. A table with this spacing and quadratic interpolation would have an error comparable to a table with $h = .01$ and linear interpolation.

For the case of general n,

$$\begin{aligned} f(x) - P_n(x) &= \frac{(x-x_0)\cdots(x-x_n)}{(n+1)!} f^{(n+1)}(c_x) \\ &= \frac{\Psi_n(x)}{(n+1)!} f^{(n+1)}(c_x) \\ \Psi_n(x) &= (x-x_0)(x-x_1)\cdots(x-x_n) \end{aligned}$$

with c_x some point between the minimum and maximum of the points in $\{x, x_0, ..., x_n\}$. When bounding the error we replace $f^{(n+1)}(c_x)$ with its maximum over the interval containing $\{x, x_0, ..., x_n\}$, as we have illustrated earlier in the linear and quadratic cases.

Consider now the function

$$\frac{\Psi_n(x)}{(n+1)!}$$

over the interval determined by the minimum and maximum of the points in $\{x, x_0, ..., x_n\}$. For evenly spaced node points on $[0, 1]$, with $x_0 = 0$ and $x_n = 1$, we give graphs for $n = 2, 3, 4, 5$ and for $n = 6, 7, 8, 9$ on accompanying pages.

DISCUSSION OF ERROR

Consider the error

$$\begin{aligned} f(x) - P_n(x) &= \frac{(x-x_0)\cdots(x-x_n)}{(n+1)!} f^{(n+1)}(c_x) \\ &= \frac{\Psi_n(x)}{(n+1)!} f^{(n+1)}(c_x) \\ \Psi_n(x) &= (x-x_0)(x-x_1)\cdots(x-x_n) \end{aligned}$$

as n increases and as x varies. As noted previously, we cannot do much with $f^{(n+1)}(c_x)$ except to replace it with a maximum value of $\left|f^{(n+1)}(x)\right|$ over a suitable interval. Thus we concentrate on understanding the size of

$$\frac{\Psi_n(x)}{(n+1)!}$$

ERROR FOR EVENLY SPACED NODES

We consider first the case in which the node points are evenly spaced, as this seems the 'natural' way to define the points at which interpolation is carried out. Moreover, using evenly spaced nodes is the case to consider for table interpolation. What can we learn from the given graphs?

The interpolation nodes are determined by using

$$h = \frac{1}{n},\ x_0 = 0,\ x_1 = h,\ x_2 = 2h, ...,\ x_n = nh = 1$$

For this case,

$$\Psi_n(x) = x\,(x-h)\,(x-2h)\cdots(x-1)$$

Our graphs are the cases of $n = 2, ..., 9$.

Graphs of $\Psi_n(x)$ on $[0, 1]$ for $n = 2, 3, 4, 5$

Graphs of $\Psi_n(x)$ on $[0, 1]$ for $n = 6, 7, 8, 9$

Graph of

$$\Psi_6(x) = (x - x_0)(x - x_1)\cdots(x - x_6)$$

with evenly spaced nodes:

Using the following table

,

n	M_n	n	M_n
1	1.25E−1	6	4.76E−7
2	2.41E−2	7	2.20E−8
3	2.06E−3	8	9.11E−10
4	1.48E−4	9	3.39E−11
5	9.01E−6	10	1.15E−12

we can observe that the maximum

$$M_n \equiv \max_{x_0 \leq x \leq x_n} \frac{|\Psi_n(x)|}{(n+1)!}$$

becomes smaller with increasing n.

From the graphs, there is enormous variation in the size of $\Psi_n(x)$ as x varies over $[0, 1]$; and thus there is also enormous variation in the error as x so varies. For example, in the $n = 9$ case,

$$\begin{aligned} \max_{x_0 \le x \le x_1} \frac{|\Psi_n(x)|}{(n+1)!} &= 3.39 \times 10^{-11} \\ \max_{x_4 \le x \le x_5} \frac{|\Psi_n(x)|}{(n+1)!} &= 6.89 \times 10^{-13} \end{aligned}$$

and the ratio of these two errors is approximately 49. Thus the interpolation error is likely to be around 49 times larger when $x_0 \le x \le x_1$ as compared to the case when $x_4 \le x \le x_5$. When doing table interpolation, the point x at which you are interpolating should be centrally located with respect to the interpolation nodes m$\{x_0, ..., x_n\}$ being used to define the interpolation, if possible.

AN APPROXIMATION PROBLEM

Consider now the problem of using an interpolation polynomial to approximate a given function $f(x)$ on a given interval $[a, b]$. In particular, take interpolation nodes

$$a \le x_0 < x_1 < \cdots < x_{n-1} < x_n \le b$$

and produce the interpolation polynomial $P_n(x)$ that interpolates $f(x)$ at the given node points. We would like to have

$$\max_{a \le x \le b} |f(x) - P_n(x)| \to 0 \quad \text{as} \quad n \to \infty$$

Does it happen?

Recall the error bound

$$\max_{a \le x \le b} |f(x) - P_n(x)| \le \max_{a \le x \le b} \frac{|\Psi_n(x)|}{(n+1)!} \cdot \max_{a \le x \le b} \left| f^{(n+1)}(x) \right|$$

We begin with an example using evenly spaced node points.

RUNGE'S EXAMPLE

Use evenly spaced node points:

$$h = \frac{b-a}{n}, \qquad x_i = a + ih \quad \text{for} \quad i = 0, ..., n$$

For some functions, such as $f(x) = e^x$, the maximum error goes to zero quite rapidly. But the size of the derivative term $f^{(n+1)}(x)$ in

$$\max_{a \le x \le b} |f(x) - P_n(x)|$$
$$\le \max_{a \le x \le b} \frac{|\Psi_n(x)|}{(n+1)!} \cdot \max_{a \le x \le b} \left| f^{(n+1)}(x) \right|$$

can badly hurt or destroy the convergence of other cases.

In particular, we show the graph of $f(x) = 1/\left(1 + x^2\right)$ and $P_n(x)$ on $[-5, 5]$ for the cases $n = 8$ and $n = 12$. The case $n = 10$ is in the text on page 127. It can be proven that for this function, the maximum error on $[-5, 5]$ does not converge to zero. Thus the use of evenly spaced nodes is not necessarily a good approach to approximating a function $f(x)$ by interpolation.

Runge's example with $n = 10$:

OTHER CHOICES OF NODES

Recall the general error bound

$$\max_{a \le x \le b} |f(x) - P_n(x)| \le \max_{a \le x \le b} \frac{|\Psi_n(x)|}{(n+1)!} \cdot \max_{a \le x \le b} \left| f^{(n+1)}(x) \right|$$

There is nothing we really do with the derivative term for f; but we can examine the way of defining the nodes $\{x_0, ..., x_n\}$ within the interval $[a, b]$. We ask how these nodes can be chosen so that the maximum of $|\Psi_n(x)|$ over $[a, b]$ is made as small as possible.

This problem has quite an elegant solution, and it is taken up in §4.6. The node points $\{x_0, ..., x_n\}$ turn out to be the zeros of a particular polynomial $T_{n+1}(x)$ of degree $n+1$, called a *Chebyshev polynomial*. These zeros are known explicitly, and with them

$$\max_{a \le x \le b} |\Psi_n(x)| = \left(\frac{b-a}{2}\right)^{n+1} 2^{-n}$$

This turns out to be smaller than for evenly spaced cases; and although this polynomial interpolation does not work for all functions $f(x)$, it works for all differentiable functions and more.

ANOTHER ERROR FORMULA

Recall the error formula

$$\begin{aligned} f(x) - P_n(x) &= \frac{\Psi_n(x)}{(n+1)!} f^{(n+1)}(c) \\ \Psi_n(x) &= (x - x_0)(x - x_1) \cdots (x - x_n) \end{aligned}$$

with c between the minimum and maximum of $\{x_0, ..., x_n, x\}$.
A second formula is given by

$$f(x) - P_n(x) = \Psi_n(x)\, f[x_0, ..., x_n, x]$$

To show this is a simple, but somewhat subtle argument.

Let $P_{n+1}(x)$ denote the polynomial of degree $\leq n+1$ which interpolates $f(x)$ at the points $\{x_0, ..., x_n, x_{n+1}\}$. Then

$$\begin{aligned} P_{n+1}(x) = P_n(x) \\ + f[x_0, ..., x_n, x_{n+1}]\,(x - x_0) \cdots (x - x_n) \end{aligned}$$

Substituting $x = x_{n+1}$, and using the fact that $P_{n+1}(x)$ interpolates $f(x)$ at x_{n+1}, we have

$$\begin{aligned} f(x_{n+1}) &= P_n(x_{n+1}) \\ &\quad + f[x_0, ..., x_n, x_{n+1}]\,(x_{n+1} - x_0) \cdots (x_{n+1} - x_n) \end{aligned}$$

$$\begin{aligned} f(x_{n+1}) &= P_n(x_{n+1}) \\ &\quad + f[x_0, ..., x_n, x_{n+1}]\,(x_{n+1} - x_0) \cdots (x_{n+1} - x_n) \end{aligned}$$

In this formula, the number x_{n+1} is completely arbitrary, other than being distinct from the points in $\{x_0, ..., x_n\}$. To emphasize this fact, replace x_{n+1} by x throughout the formula, obtaining

$$\begin{aligned} f(x) &= P_n(x) + f[x_0, ..., x_n, x]\,(x - x_0) \cdots (x - x_n) \\ &= P_n(x) + \Psi_n(x)\, f[x_0, ..., x_n, x] \end{aligned}$$

provided $x \neq x_0, ..., x_n$.

The formula

$$\begin{aligned} f(x) &= P_n(x) + f[x_0, ..., x_n, x]\,(x - x_0)\cdots(x - x_n) \\ &= P_n(x) + \Psi_n(x)\, f[x_0, ..., x_n, x] \end{aligned}$$

is easily true for x a node point. Provided $f(x)$ is differentiable, the formula is also true for x a node point.

This shows

$$f(x) - P_n(x) = \Psi_n(x)\, f[x_0, ..., x_n, x]$$

Compare the two error formulas

$$f(x) - P_n(x) = \Psi_n(x)\, f[x_0, ..., x_n, x]$$

$$f(x) - P_n(x) = \frac{\Psi_n(x)}{(n+1)!} f^{(n+1)}(c)$$

Then

$$\Psi_n(x)\, f[x_0, ..., x_n, x] = \frac{\Psi_n(x)}{(n+1)!} f^{(n+1)}(c)$$

$$f[x_0, ..., x_n, x] = \frac{f^{(n+1)}(c)}{(n+1)!}$$

for some c between the smallest and largest of the numbers in $\{x_0, ..., x_n, x\}$.

To make this somewhat symmetric in its arguments, let $m = n + 1$, $x = x_{n+1}$. Then

$$f[x_0, ..., x_{m-1}, x_m] = \frac{f^{(m)}(c)}{m!}$$

with c an unknown number between the smallest and largest of the numbers in $\{x_0, ..., x_m\}$. This was given in an earlier lecture where divided differences were introduced.

BEST APPROXIMATION

Given a function $f(x)$ that is continuous on a given interval $[a, b]$, consider approximating it by some polynomial $p(x)$. To measure the error in $p(x)$ as an approximation, introduce

$$E(p) = \max_{a \le x \le b} |f(x) - p(x)|$$

This is called the *maximum error* or *uniform error* of approximation of $f(x)$ by $p(x)$ on $[a, b]$.

With an eye towards efficiency, we want to find the 'best' possible approximation of a given degree n. With this in mind, introduce the following:

$$\begin{aligned} \rho_n(f) &= \min_{\deg(p) \le n} E(p) \\ &= \min_{\deg(p) \le n} \left[\max_{a \le x \le b} |f(x) - p(x)| \right] \end{aligned}$$

The number $\rho_n(f)$ will be the smallest possible uniform error, or *minimax error*, when approximating $f(x)$ by polynomials of degree at most n. If there is a polynomial giving this smallest error, we denote it by $m_n(x)$; thus $E(m_n) = \rho_n(f)$.

Example. Let $f(x) = e^x$ on $[-1, 1]$. In the following table, we give the values of $E(t_n)$, $t_n(x)$ the Taylor polynomial of degree n for e^x about $x = 0$, and $E(m_n)$.

	Maximum Error in:	
n	$t_n(x)$	$m_n(x)$
1	7.18E − 1	2.79E − 1
2	2.18E − 1	4.50E − 2
3	5.16E − 2	5.53E − 3
4	9.95E − 3	5.47E − 4
5	1.62E − 3	4.52E − 5
6	2.26E − 4	3.21E − 6
7	2.79E − 5	2.00E − 7
8	3.06E − 6	1.11E − 8
9	3.01E − 7	5.52E − 10

Consider graphically how we can improve on the Taylor polynomial

$$t_1(x) = 1 + x$$

as a uniform approximation to e^x on the interval $[-1, 1]$.

The linear minimax approximation is

$$m_1(x) = 1.2643 + 1.1752x$$

Linear Taylor and minimax approximations to e^x

Error in cubic Taylor approximation to e^x

Error in cubic minimax approximation to e^x

Accuracy of the minimax approximation.

$$\rho_n(f) \leq \frac{[(b-a)/2]^{n+1}}{(n+1)!2^n} \max_{a \leq x \leq b} \left| f^{(n+1)}(x) \right|$$

This error bound does not always become smaller with increasing n, but it will give a fairly accurate bound for many common functions $f(x)$.

Example. Let $f(x) = e^x$ for $-1 \leq x \leq 1$. Then

$$\rho_n(e^x) \leq \frac{e}{(n+1)!2^n} \qquad (*)$$

n	*Bound* (*)	$\rho_n(f)$
1	6.80E − 1	2.79E − 1
2	1.13E − 1	4.50E − 2
3	1.42E − 2	5.53E − 3
4	1.42E − 3	5.47E − 4
5	1.18E − 4	4.52E − 5
6	8.43E − 6	3.21E − 6
7	5.27E − 7	2.00E − 7

CHEBYSHEV POLYNOMIALS

Chebyshev polynomials are used in many parts of numerical analysis, and more generally, in applications of mathematics. For an integer $n \geq 0$, define the function

$$T_n(x) = \cos\left(n \cos^{-1} x\right), \qquad -1 \leq x \leq 1 \tag{1}$$

This may not appear to be a polynomial, but we will show it is a polynomial of degree n. To simplify the manipulation of (1), we introduce

$$\theta = \cos^{-1}(x) \quad \text{or} \quad x = \cos(\theta), \qquad 0 \leq \theta \leq \pi \tag{2}$$

Then

$$T_n(x) = \cos(n\theta) \tag{3}$$

Example. $n = 0$

$$T_0(x) = \cos(0 \cdot \theta) = 1$$

$n = 1$

$$T_1(x) = \cos(\theta) = x$$

$n = 2$

$$T_2(x) = \cos(2\theta) = 2\cos^2(\theta) - 1 = 2x^2 - 1$$

y
1
-1
1
x
-1
$T_0(x)$
$T_1(x)$
$T_2(x)$

$T_3(x)$
$T_4(x)$
y
1
-1
1
x
-1

The triple recursion relation. Recall the trigonometric addition formulas,

$$\cos(\alpha \pm \beta) = \cos(\alpha)\cos(\beta) \mp \sin(\alpha)\sin(\beta)$$

Let $n \geq 1$, and apply these identities to get

$$\begin{aligned} T_{n+1}(x) &= \cos[(n+1)\theta] = \cos(n\theta + \theta) \\ &= \cos(n\theta)\cos(\theta) - \sin(n\theta)\sin(\theta) \\ T_{n-1}(x) &= \cos[(n-1)\theta] = \cos(n\theta - \theta) \\ &= \cos(n\theta)\cos(\theta) + \sin(n\theta)\sin(\theta) \end{aligned}$$

Add these two equations, and then use (1) and (3) to obtain

$$\begin{aligned} T_{n+1}(x) + T_{n-1} &= 2\cos(n\theta)\cos(\theta) = 2xT_n(x) \\ T_{n+1}(x) &= 2xT_n(x) - T_{n-1}(x), \qquad n \geq 1 \end{aligned} \tag{4}$$

This is called the *triple recursion relation* for the Chebyshev polynomials. It is often used in evaluating them, rather than using the explicit formula (1).

Example. Recall

$$T_0(x) = 1, \qquad T_1(x) = x$$

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x), \qquad n \geq 1$$

Let $n = 2$. Then

$$\begin{aligned} T_3(x) &= 2xT_2(x) - T_1(x) \\ &= 2x(2x^2 - 1) - x \\ &= 4x^3 - 3x \end{aligned}$$

Let $n = 3$. Then

$$\begin{aligned} T_4(x) &= 2xT_3(x) - T_2(x) \\ &= 2x(4x^3 - 3x) - (2x^2 - 1) \\ &= 8x^4 - 8x^2 + 1 \end{aligned}$$

The minimum size property. Note that

$$|T_n(x)| \leq 1, \qquad -1 \leq x \leq 1 \tag{5}$$

for all $n \geq 0$. Also, note that

$$T_n(x) = 2^{n-1}x^n + \text{lower degree terms}, \qquad n \geq 1 \tag{6}$$

This can be proven using the triple recursion relation and mathematical induction.

Introduce a modified version of $T_n(x)$,

$$\widetilde{T}_n(x) = \frac{1}{2^{n-1}} T_n(x) = x^n + \text{lower degree terms} \tag{7}$$

From (5) and (6),

$$\left|\widetilde{T}_n(x)\right| \leq \frac{1}{2^{n-1}}, \qquad -1 \leq x \leq 1, \qquad n \geq 1 \tag{8}$$

Example.

$$\widetilde{T}_4(x) = \frac{1}{8}\left(8x^4 - 8x^2 + 1\right) = x^4 - x^2 + \frac{1}{8}$$

A polynomial whose highest degree term has a coefficient of 1 is called a *monic polynomial*. Formula (8) says the monic polynomial $\widetilde{T}_n(x)$ has size $1/2^{n-1}$ on $-1 \le x \le 1$, and this becomes smaller as the degree n increases. In comparison,

$$\max_{-1 \le x \le 1} |x^n| = 1$$

Thus x^n is a monic polynomial whose size does not change with increasing n.

Theorem. Let $n \ge 1$ be an integer, and consider all possible monic polynomials of degree n. Then the degree n monic polynomial with the smallest maximum on $[-1, 1]$ is the modified Chebyshev polynomial $\widetilde{T}_n(x)$, and its maximum value on $[-1, 1]$ is $1/2^{n-1}$.

This result is used in devising applications of Chebyshev polynomials. We apply it to obtain an improved interpolation scheme.

A NEAR-MINIMAX APPROXIMATION METHOD

Let $f(x)$ be continuous on $[a, b] = [-1, 1]$. Consider approximating f by an interpolatory polynomial of degree at most $n = 3$. Let x_0, x_1, x_2, x_3 be interpolation node points in $[-1, 1]$; let $c_3(x)$ be of degree ≤ 3 and interpolate $f(x)$ at $\{x_0, x_1, x_2, x_3\}$. The interpolation error is

$$f(x) - c_3(x) = \frac{\omega(x)}{4!} f^{(4)}(\xi_x), \quad -1 \leq x \leq 1 \tag{1}$$

$$\omega(x) = (x - x_0)(x - x_1)(x - x_2)(x - x_3) \tag{2}$$

with ξ_x in $[-1, 1]$. We want to choose the nodes $\{x_0, x_1, x_2, x_3\}$ so as to minimize the maximum value of $|f(x) - c_3(x)|$ on $[-1, 1]$.

From (1), the only general quantity, independent of f, is $\omega(x)$. Thus we choose $\{x_0, x_1, x_2, x_3\}$ to minimize

$$\max_{-1 \leq x \leq 1} |\omega(x)| \tag{3}$$

Expand to get

$$\omega(x) = x^4 + \text{lower degree terms}$$

This is a monic polynomial of degree 4. From the theorem in the preceding section, the smallest possible value for (3) is obtained with

$$\omega(x) = \widetilde{T}_4(x) = \frac{T_4(x)}{2^3} = \frac{1}{8}(8x^4 - 8x^2 + 1) \qquad (4)$$

and the smallest value of (3) is $1/2^3$ in this case. The equation (4) defines implicitly the nodes $\{x_0, x_1, x_2, x_3\}$: they are the roots of $T_4(x)$.

In our case this means solving

$$T_4(x) = \cos(4\theta) = 0, \qquad x = \cos(\theta)$$

$$\begin{aligned} 4\theta &= \pm\frac{\pi}{2}, \pm\frac{3\pi}{2}, \pm\frac{5\pi}{2}, \pm\frac{7\pi}{2}, \ldots \\ \theta &= \pm\frac{\pi}{8}, \pm\frac{3\pi}{8}, \pm\frac{5\pi}{8}, \pm\frac{7\pi}{8}, \ldots \\ x &= \cos\left(\frac{\pi}{8}\right), \cos\left(\frac{3\pi}{8}\right), \cos\left(\frac{5\pi}{8}\right), \ldots \qquad (5) \end{aligned}$$

using $\cos(-\theta) = \cos(\theta)$.

$$x = \cos\left(\frac{\pi}{8}\right), \cos\left(\frac{3\pi}{8}\right), \cos\left(\frac{5\pi}{8}\right), \cos\left(\frac{7\pi}{8}\right), \ldots$$

The first four values are distinct; the following ones are repetitive. For example,

$$\cos\left(\frac{9\pi}{8}\right) = \cos\left(\frac{7\pi}{8}\right)$$

The first four values are

$$\{x_0, x_1, x_2, x_3\} = \{\pm 0.382683, \pm 0.923880\} \qquad (6)$$

Example. Let $f(x) = e^x$ on $[-1, 1]$. Use these nodes to produce the interpolating polynomial $c_3(x)$ of degree 3. From the interpolation error formula and the bound of $1/2^3$ for $|\omega(x)|$ on $[-1, 1]$, we have

$$\begin{aligned} \max_{-1 \le x \le 1} |f(x) - c_3(x)| &\le \frac{1/2^3}{4!} \max_{-1 \le x \le 1} e^{\xi_x} \\ &\le \frac{e}{192} \doteq 0.014158 \end{aligned}$$

By direct calculation,

$$\max_{-1 \le x \le 1} |e^x - c_3(x)| \doteq 0.00666$$

Interpolation Data: $f(x) = e^x$

i	x_i	$f(x_i)$	$f[x_0, \ldots, x_i]$
0	0.923880	2.5190442	2.5190442
1	0.382683	1.4662138	1.9453769
2	-0.382683	0.6820288	0.7047420
3	-0.923880	0.3969760	0.1751757

The error $e^x - c_3(x)$

For comparison, $E(t_3) \doteq 0.0142$ and $\rho_3(e^x) \doteq 0.00553$.

THE GENERAL CASE

Consider interpolating $f(x)$ on $[-1,1]$ by a polynomial of degree $\leq n$, with the interpolation nodes $\{x_0, \ldots, x_n\}$ in $[-1,1]$. Denote the interpolation polynomial by $c_n(x)$. The interpolation error on $[-1,1]$ is given by

$$\begin{aligned} f(x) - c_n(x) &= \frac{\omega(x)}{(n+1)!} f^{(n+1)}(\xi_x) \qquad (7) \\ \omega(x) &= (x - x_0) \cdots (x - x_n) \end{aligned}$$

with ξ_x and unknown point in $[-1,1]$. In order to minimize the interpolation error, we seek to minimize

$$\max_{-1 \leq x \leq 1} |\omega(x)| \qquad (8)$$

The polynomial being minimized is monic of degree $n+1$,

$$\omega(x) = x^{n+1} + \text{ lower degree terms}$$

From the theorem of the preceding section, this minimum is attained by the monic polynomial

$$\widetilde{T}_{n+1}(x) = \frac{1}{2^n} T_{n+1}(x)$$

Thus the interpolation nodes are the zeros of $T_{n+1}(x)$; and by the procedure that led to (5), they are given by

$$x_j = \cos\left(\frac{2j+1}{2n+2}\pi\right), \qquad j = 0, 1, \ldots, n \tag{9}$$

The near-minimax approximation $c_n(x)$ of degree n is obtained by interpolating to $f(x)$ at these $n+1$ nodes on $[-1, 1]$.

The polynomial $c_n(x)$ is sometimes called a *Chebyshev approximation*.

Example. Let $f(x) = e^x$. the following table contains the maximum errors in $c_n(x)$ on $[-1, 1]$ for varying n. For comparison, we also include the corresponding minimax errors. These figures illustrate that for practical purposes, $c_n(x)$ is a satisfactory replacement for the minimax approximation $m_n(x)$.

n	max $\lvert e^x - c_n(x)\rvert$	$\rho_n(e^x)$
1	3.72E − 1	2.79E − 1
2	5.65E − 2	4.50E − 2
3	6.66E − 3	5.53E − 3
4	6.40E − 4	5.47E − 4
5	5.18E − 5	4.52E − 5
6	3.80E − 6	3.21E − 6

THEORETICAL INTERPOLATION ERROR

For the error

$$f(x) - c_n(x) = \frac{\omega(x)}{(n+1)!} f^{(n+1)}(\xi_x)$$

we have

$$\max_{-1 \le x \le 1} |f(x) - c_n(x)| \le \frac{\max\limits_{-1 \le x \le 1} |\omega(x)|}{(n+1)!} \max_{-1 \le \xi \le 1} |f(\xi)|$$

From the theorem of the preceding section,

$$\max_{-1 \le x \le 1} \left| \widetilde{T}_{n+1}(x) \right| = \max_{-1 \le x \le 1} |\omega(x)| = \frac{1}{2^n}$$

in this case. Thus

$$\max_{-1 \le x \le 1} |f(x) - c_n(x)| \le \frac{1}{(n+1)! 2^n} \max_{-1 \le \xi \le 1} |f(\xi)|$$

OTHER INTERVALS

Consider approximating $f(x)$ on the finite interval $[a, b]$. Introduce the linear change of variables

$$x = \frac{1}{2}\left[(1-t)\,a + (1+t)\,b\right] \tag{10}$$

$$t = \frac{2}{b-a}\left[x - \frac{b+a}{2}\right] \tag{11}$$

Introduce

$$F(t) = f\left(\frac{1}{2}\left[(1-t)\,a + (1+t)\,b\right]\right), \quad -1 \le t \le 1$$

The function $F(t)$ on $[-1, 1]$ is equivalent to $f(x)$ on $[a, b]$, and we can move between them via (10)-(11). We can now proceed to approximate $f(x)$ on $[a, b]$ by instead approximating $F(t)$ on $[-1, 1]$.

Example. Approximating $f(x) = \cos x$ on $[0, \pi/2]$ is equivalent to approximating

$$F(t) = \cos\left(\frac{1+t}{4}\pi\right), \quad -1 \le t \le 1$$

PIECEWISE POLYNOMIAL INTERPOLATION

Recall the examples of higher degree polynomial interpolation of the function $f(x) = \left(1 + x^2\right)^{-1}$ on $[-5, 5]$. The interpolants $P_n(x)$ oscillated a great deal, whereas the function $f(x)$ was nonoscillatory. To obtain interpolants that are better behaved, we look at other forms of interpolating functions.

Consider the data

x	0	1	2	2.5	3	3.5	4
y	2.5	0.5	0.5	1.5	1.5	1.125	0

What are methods of interpolating this data, other than using a degree 6 polynomial. Shown in the text are the graphs of the degree 6 polynomial interpolant, along with those of *piecewise linear* and a *piecewise quadratic* interpolating functions.

Since we only have the data to consider, we would generally want to use an interpolant that had somewhat the shape of that of the piecewise linear interpolant.

The data points

Piecewise linear interpolation

Polynomial Interpolation

Piecewise quadratic interpolation

PIECEWISE POLYNOMIAL FUNCTIONS

Consider being given a set of data points $(x_1, y_1), ...,$ (x_n, y_n), with

$$x_1 < x_2 < \cdots < x_n$$

Then the simplest way to connect the points (x_j, y_j) is by straight line segments. This is called a *piecewise linear interpolant* of the data $\Big\{(x_j, y_j)\Big\}$. This graph has "corners", and often we expect the interpolant to have a smooth graph.

To obtain a somewhat smoother graph, consider using *piecewise quadratic interpolation*. Begin by constructing the quadratic polynomial that interpolates

$$\{(x_1, y_1), (x_2, y_2), (x_3, y_3)\}$$

Then construct the quadratic polynomial that interpolates

$$\{(x_3, y_3), (x_4, y_4), (x_5, y_5)\}$$

Continue this process of constructing quadratic interpolants on the subintervals

$$[x_1, x_3], [x_3, x_5], [x_5, x_7], ...$$

If the number of subintervals is even (and therefore n is odd), then this process comes out fine, with the last interval being $[x_{n-2}, x_n]$. This was illustrated on the graph for the preceding data. If, however, n is even, then the approximation on the last interval must be handled by some modification of this procedure. Suggest such!

With piecewise quadratic interpolants, however, there are "corners" on the graph of the interpolating function. With our preceding example, they are at x_3 and x_5. How do we avoid this?

Piecewise polynomial interpolants are used in many applications. We will consider them later, to obtain numerical integration formulas.

SMOOTH NON-OSCILLATORY INTERPOLATION

Let data points $(x_1, y_1), ..., (x_n, y_n)$ be given, as let

$$x_1 < x_2 < \cdots < x_n$$

Consider finding functions $s(x)$ for which the following properties hold:
(1) $s(x_i) = y_i, \qquad i = 1, ..., n$
(2) $s(x)$, $s'(x)$, $s''(x)$ are continuous on $[x_1, x_n]$.
Then among such functions $s(x)$ satisfying these properties, find the one which minimizes the integral

$$\int_{x_1}^{x_n} \left| s''(x) \right|^2 \, dx$$

The idea of minimizing the integral is to obtain an interpolating function for which the first derivative does not change rapidly. It turns out there is a unique solution to this problem, and it is called a *natural cubic spline* function.

SPLINE FUNCTIONS

Let a set of node points $\{x_i\}$ be given, satisfying

$$a \leq x_1 < x_2 < \cdots < x_n \leq b$$

for some numbers a and b. Often we use $[a, b] = [x_1, x_n]$. A cubic spline function $s(x)$ on $[a, b]$ with "breakpoints" or "knots" $\{x_i\}$ has the following properties:

1. On each of the intervals

$$[a, x_1], [x_1, x_2], ..., [x_{n-1}, x_n], [x_n, b]$$

$s(x)$ is a polynomial of degree ≤ 3.

2. $s(x)$, $s'(x)$, $s''(x)$ are continuous on $[a, b]$.

In the case that we have given data points (x_1, y_1),..., (x_n, y_n), we say $s(x)$ is a cubic interpolating spline function for this data if

3. $s(x_i) = y_i, \quad i = 1, ..., n.$

Return to the earlier problem of choosing an interpolating function $s(x)$ to minimize the integral

$$\int_{x_1}^{x_n} \left|s''(x)\right|^2 \, dx$$

There is a unique solution to problem. The solution $s(x)$ is a cubic interpolating spline function, and moreover, it satisfies

$$s''(x_1) = s''(x_n) = 0$$

Spline functions satisfying these *boundary conditions* are called "natural" cubic spline functions, and the solution to our minimization problem is a "natural cubic interpolatory spline function". We will show a method to construct this function from the interpolation data.

Motivation for these boundary conditions can be given by looking at the physics of bending thin beams of flexible materials to pass thru the given data. To the left of x_1 and to the right of x_n, the beam is straight and therefore the second derivatives are zero at the transition points x_1 and x_n.

CONSTRUCTION OF THE INTERPOLATING SPLINE FUNCTION

To make the presentation more specific, suppose we have data

$$(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)$$

with $x_1 < x_2 < x_3 < x_4$. Then on each of the intervals

$$[x_1, x_2], [x_2, x_3], [x_3, x_4]$$

$s(x)$ is a cubic polynomial. Taking the first interval, $s(x)$ is a cubic polynomial and $s''(x)$ is a linear polynomial. Let

$$M_i = s''(x_i), \quad i = 1, 2, 3, 4$$

Then on $[x_1, x_2]$,

$$s''(x) = \frac{(x_2 - x)\, M_1 + (x - x_1)\, M_2}{x_2 - x_1}, \quad x_1 \le x \le x_2$$

We can find $s(x)$ by integrating twice:

$$s(x) = \frac{(x_2 - x)^3 M_1 + (x - x_1)^3 M_2}{6\,(x_2 - x_1)} + c_1 x + c_2$$

We determine the constants of integration by using

$$s(x_1) = y_1, \quad s(x_2) = y_2 \qquad (*)$$

Then

$$\begin{aligned} s(x) = & \frac{(x_2 - x)^3 M_1 + (x - x_1)^3 M_2}{6\,(x_2 - x_1)} \\ & + \frac{(x_2 - x)\, y_1 + (x - x_1)\, y_2}{x_2 - x_1} \\ & - \frac{x_2 - x_1}{6} \left[(x_2 - x)\, M_1 + (x - x_1)\, M_2 \right] \end{aligned}$$

for $x_1 \leq x \leq x_2$.

Check that this formula satisfies the given interpolation condition (*)!

We can repeat this on the intervals $[x_2, x_3]$ and $[x_3, x_4]$, obtaining similar formulas.

For $x_2 \leq x \leq x_3$,

$$
\begin{aligned}
s(x) = & \frac{(x_3 - x)^3 M_2 + (x - x_2)^3 M_3}{6\,(x_3 - x_2)} \\
& + \frac{(x_3 - x)\, y_2 + (x - x_2)\, y_3}{x_3 - x_2} \\
& - \frac{x_3 - x_2}{6} \left[(x_3 - x)\, M_2 + (x - x_2)\, M_3 \right]
\end{aligned}
$$

For $x_3 \leq x \leq x_4$,

$$
\begin{aligned}
s(x) = & \frac{(x_4 - x)^3 M_3 + (x - x_3)^3 M_4}{6\,(x_4 - x_3)} \\
& + \frac{(x_4 - x)\, y_3 + (x - x_3)\, y_4}{x_4 - x_3} \\
& - \frac{x_4 - x_3}{6} \left[(x_4 - x)\, M_3 + (x - x_3)\, M_4 \right]
\end{aligned}
$$

We still do not know the values of the second derivatives $\{M_1, M_2, M_3, M_4\}$. The above formulas guarantee that $s(x)$ and $s''(x)$ are continuous for $x_1 \leq x \leq x_4$. For example, the formula on $[x_1, x_2]$ yields

$$s(x_2) = y_2, \qquad s''(x_2) = M_2$$

The formula on $[x_2, x_3]$ also yields

$$s(x_2) = y_2, \qquad s''(x_2) = M_2$$

All that is lacking is to make $s'(x)$ continuous at x_2 and x_3. Thus we require

$$\begin{aligned} s'(x_2 + 0) &= s'(x_2 - 0) \\ s'(x_3 + 0) &= s'(x_3 - 0) \end{aligned} \qquad (**)$$

This means

$$\lim_{x \searrow x_2} s'(x) = \lim_{x \nearrow x_2} s'(x)$$

and similarly for x_3.

To simplify the presentation somewhat, I assume in the following that our node points are evenly spaced:

$$x_2 = x_1 + h, \quad x_3 = x_1 + 2h, \quad x_4 = x_1 + 3h$$

Then our earlier formulas simplify to

$$\begin{aligned} s(x) = & \frac{(x_2 - x)^3 M_1 + (x - x_1)^3 M_2}{6h} \\ & + \frac{(x_2 - x) y_1 + (x - x_1) y_2}{h} \\ & - \frac{h}{6} \left[(x_2 - x) M_1 + (x - x_1) M_2 \right] \end{aligned}$$

for $x_1 \leq x \leq x_2$, with similar formulas on $[x_2, x_3]$ and $[x_3, x_4]$.

Without going thru all of the algebra, the conditions (**) leads to the following pair of equations.

$$\begin{aligned}
&\frac{h}{6}M_1 + \frac{2h}{3}M_2 + \frac{h}{6}M_3 \\
&\qquad = \frac{y_3 - y_2}{h} - \frac{y_2 - y_1}{h} \\
&\frac{h}{6}M_2 + \frac{2h}{3}M_3 + \frac{h}{6}M_4 \\
&\qquad = \frac{y_4 - y_3}{h} - \frac{y_3 - y_2}{h}
\end{aligned}$$

This gives us two equations in four unknowns. The earlier boundary conditions on $s''(x)$ gives us immediately

$$M_1 = M_4 = 0$$

Then we can solve the linear system for M_2 and M_3.

EXAMPLE

Consider the interpolation data points

x	1	2	3	4
y	1	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{4}$

In this case, $h = 1$, and linear system becomes

$$\begin{aligned} \frac{2}{3}M_2 + \frac{1}{6}M_3 &= y_3 - 2y_2 + y_1 = \frac{1}{3} \\ \frac{1}{6}M_2 + \frac{2}{3}M_3 &= y_4 - 2y_3 + y_2 = \frac{1}{12} \end{aligned}$$

This has the solution

$$M_2 = \frac{1}{2}, \qquad M_3 = 0$$

This leads to the spline function formula on each subinterval.

On $[1, 2]$,

$$
\begin{aligned}
s(x) = & \frac{(x_2 - x)^3 M_1 + (x - x_1)^3 M_2}{6h} \\
& + \frac{(x_2 - x) y_1 + (x - x_1) y_2}{h} \\
& - \frac{h}{6} \left[(x_2 - x) M_1 + (x - x_1) M_2 \right] \\
= & \frac{(2 - x)^3 \cdot 0 + (x - 1)^3 \left(\frac{1}{2}\right)}{6} + \frac{(2 - x) \cdot 1 + (x - 1) \left(\frac{1}{2}\right)}{1} \\
& - \frac{1}{6} \left[(2 - x) \cdot 0 + (x - 1) \left(\frac{1}{2}\right) \right] \\
= & \tfrac{1}{12} (x - 1)^3 - \tfrac{7}{12} (x - 1) + 1
\end{aligned}
$$

Similarly, for $2 \leq x \leq 3$,

$$
s(x) = \frac{-1}{12} (x - 2)^3 + \frac{1}{4} (x - 2)^2 - \frac{1}{3} (x - 1) + \frac{1}{2}
$$

and for $3 \leq x \leq 4$,

$$
s(x) = \frac{-1}{12} (x - 4) + \frac{1}{4}
$$

x	1	2	3	4
y	1	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{4}$

Graph of example of natural cubic spline interpolation

x	0	1	2	2.5	3	3.5	4
y	2.5	0.5	0.5	1.5	1.5	1.125	0

Interpolating natural cubic spline function

We can now write the functions $s(x)$ for each of the subintervals $[x_1, x_2]$, $[x_2, x_3]$, and $[x_3, x_4]$. Recall for $x_1 \leq x \leq x_2$,

$$\begin{aligned} s(x) = & \frac{(x_2 - x)^3 M_1 + (x - x_1)^3 M_2}{6h} \\ & + \frac{(x_2 - x) y_1 + (x - x_1) y_2}{h} \\ & - \frac{h}{6} \left[(x_2 - x) M_1 + (x - x_1) M_2 \right] \end{aligned}$$

We can substitute in from the data

x	1	2	3	4
y	1	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{4}$

and the solutions $\{M_i\}$. Doing so, consider the error $f(x) - s(x)$. As an example,

$$f(x) = \frac{1}{x}, \quad f\left(\frac{3}{2}\right) = \frac{2}{3}, \qquad s\left(\frac{3}{2}\right) = .65260$$

This is quite a decent approximation.

THE GENERAL PROBLEM

Consider the spline interpolation problem with n nodes

$$(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$$

and assume the node points $\{x_i\}$ are evenly spaced,

$$x_j = x_1 + (j-1)\,h, \qquad j = 1, ..., n$$

We have that the interpolating spline $s(x)$ on $x_j \leq x \leq x_{j+1}$ is given by

$$\begin{aligned} s(x) = & \frac{\left(x_{j+1} - x\right)^3 M_j + \left(x - x_j\right)^3 M_{j+1}}{6h} \\ & + \frac{\left(x_{j+1} - x\right) y_j + \left(x - x_j\right) y_{j+1}}{h} \\ & - \frac{h}{6}\left[\left(x_{j+1} - x\right) M_j + \left(x - x_j\right) M_{j+1}\right] \end{aligned}$$

for $j = 1, ..., n-1$.

To enforce continuity of $s'(x)$ at the interior node points $x_2, ..., x_{n-1}$, the second derivatives $\left\{M_j\right\}$ must satisfy the linear equations

$$\frac{h}{6}M_{j-1} + \frac{2h}{3}M_j + \frac{h}{6}M_{j+1} = \frac{y_{j-1} - 2y_j + y_{j+1}}{h}$$

for $j = 2, ..., n-1$. Writing them out,

$$\begin{aligned}
\frac{h}{6}M_1 + \frac{2h}{3}M_2 + \frac{h}{6}M_3 &= \frac{y_1 - 2y_2 + y_3}{h} \\
\frac{h}{6}M_2 + \frac{2h}{3}M_3 + \frac{h}{6}M_4 &= \frac{y_2 - 2y_3 + y_4}{h} \\
&\vdots \\
\frac{h}{6}M_{n-2} + \frac{2h}{3}M_{n-1} + \frac{h}{6}M_n &= \frac{y_{n-2} - 2y_{n-1} + y_n}{h}
\end{aligned}$$

This is a system of $n-2$ equations in the n unknowns $\{M_1, ..., M_n\}$. Two more conditions must be imposed on $s(x)$ in order to have the number of equations equal the number of unknowns, namely n. With the added boundary conditions, this form of linear system can be solved very efficiently.

BOUNDARY CONDITIONS

"Natural" boundary conditions

$$s''(x_1) = s''(x_n) = 0$$

Spline functions satisfying these conditions are called "natural cubic splines". They arise out the minimization problem stated earlier. But generally they are not considered as good as some other cubic interpolating splines.

"Clamped" boundary conditions We add the conditions

$$s'(x_1) = y_1', \qquad s'(x_n) = y_n'$$

with y_1', y_n' given slopes for the endpoints of $s(x)$ on $[x_1, x_n]$. This has many quite good properties when compared with the natural cubic interpolating spline; but it does require knowing the derivatives at the endpoints.

"Not a knot" boundary conditions This is more complicated to explain, but it is the version of cubic spline interpolation that is implemented in Matlab.

THE "NOT A KNOT" CONDITIONS

As before, let the interpolation nodes be

$$(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$$

We separate these points into two categories. For constructing the interpolating cubic spline function, we use the points

$$(x_1, y_1), (x_3, y_3), ..., (x_{n-2}, y_{n-2}), (x_n, y_n)$$

Thus deleting two of the points. We now have $n-2$ points, and the interpolating spline $s(x)$ can be determined on the intervals

$$[x_1, x_3], [x_3, x_4], ..., [x_{n-3}, x_{n-2}], [x_{n-2}, x_n]$$

This leads to $n-4$ equations in the $n-2$ unknowns $M_1, M_3, ..., M_{n-2}, M_n$. The two additional boundary conditions are

$$s(x_2) = y_2, \qquad s(x_{n-1}) = y_{n-1}$$

These translate into two additional equations, and we obtain a system of $n-2$ linear simultaneous equations in the $n-2$ unknowns $M_1, M_3, ..., M_{n-2}, M_n$.

x	0	1	2	2.5	3	3.5	4
y	2.5	0.5	0.5	1.5	1.5	1.125	0

Interpolating cubic spline function with "not-a_knot" boundary conditions

MATLAB SPLINE FUNCTION LIBRARY

Given data points

$$(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$$

type arrays containing the x and y coordinates:

$$\begin{array}{l} x = [x_1\ x_2\ ...x_n] \\ y = [y_1\ y_2\ ...y_n] \\ plot\,(x, y, \text{'o'}) \end{array}$$

The last statement will draw a plot of the data points, marking them with the letter 'oh'. To find the interpolating cubic spline function and evaluate it at the points of another array xx, say

$$h = (x_n - x_1)\,/\,(10 * n)\,; \qquad xx = x_1 : h : x_n;$$

use

$$\begin{array}{l} yy = spline\,(x, y, xx) \\ plot\,(x, y, \text{'o'}, xx, yy) \end{array}$$

The last statement will plot the data points, as before, and it will plot the interpolating spline $s(x)$ as a continuous curve.

ERROR IN CUBIC SPLINE INTERPOLATION

Let an interval $[a, b]$ be given, and then define

$$h = \frac{b-a}{n-1}, \qquad x_j = a + (j-1)h, \quad j = 1, ..., n$$

Suppose we want to approximate a given function $f(x)$ on the interval $[a, b]$ using cubic spline interpolation. Define

$$y_i = f(x_i), \qquad j = 1, ..., n$$

Let $s_n(x)$ denote the cubic spline interpolating this data and satisfying the "not a knot" boundary conditions. Then it can be shown that for a suitable constant c,

$$E_n \equiv \max_{a \le x \le b} |f(x) - s_n(x)| \le ch^4$$

The corresponding bound for natural cubic spline interpolation contains only a term of h^2 rather than h^4; it does not converge to zero as rapidly.

EXAMPLE

Take $f(x) = \arctan x$ on $[0, 5]$. The following table gives values of the maximum error E_n for various values of n. The values of h are being successively halved.

n	E_n	$E_{\frac{1}{2}n}/E_n$
7	7.09E−3	
13	3.24E−4	21.9
25	3.06E−5	10.6
49	1.48E−6	20.7
97	9.04E−8	16.4

Numerical Analysis / Numerical Methods

Spring 2022 Lecture 6 (Review)

Review of interpolation and approximation

PURPOSES OF INTERPOLATION

1. Replace a set of data points

$$\{(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)\}$$

   with a function given analytically. Usually this is a polynomial function. In other words, **Purpose 1** consists in finding a polynomial of degree n that passes every given data point.

2. Approximate functions (at least continuous) with simpler ones, usually polynomials or 'piecewise polynomials'. Here interpolation is used to form polynomials that accurately approximate continuous functions, and next question is how accurately we can do it?

Polynomial interpolation

Once data set with $n+1$ points $\{(x_i, y_i)\}_{i=0}^{n}$ is given, there is a UNIQUE polynomial $P_n(x)$ of degree at most n that passes them:

$$P_n(x_i) = y_i, \quad i = 0, 1, \ldots n$$

Polynomial of degree 10 passing 11 data points

Polynomial interpolation

This UNIQUE polynomial can be obtained by (see Lecture 7):

1. Lagrange formula:

$$P_n(x) = \sum_{i=0}^{n} y_i L_i(x), \quad \text{where } L_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}.$$

2. Newton difference formula

$$P_n(x) = P_{n-1}(x) + \\ + f[x_0, x_1, x_2, \ldots, x_n](x - x_0)(x - x_1)\ldots(x - x_{n-1}).$$

Never use Lagrange formula for practical reasons!

So, we have essentially solved the polynomial interpolation problem of discrete data perfectly.

Approximation Purpose

Polynomial approximation problem

Given a function $f(x)$ on $[a, b]$, usually at least continuous, find a polynomial $P(x)$ of certain degree that accurately approximates it.

We might use Taylor polynomials, but the error increases as we move toward endpoints and it will be of the same sign (see Taylor polynomial discussion in Lecture 1)

Another approach is by interpolation:

Consider the partition of $[a, b]$: $a \leqslant x_0 < x_1 < x_2 < \cdots < x_n \leqslant b$ and the data points $\{(x_0, f(x_0)), (x_1, f(x_1)), \ldots, (x_n, f(x_n))\}$.

Construct the interpolation polynomial passing these $n+1$ data points to get $P_n(x)$ (polynomial of degree n).

Interpolation for Function Approximation

Function $f(x) = \frac{1}{1+25x^2}$ (red) and polynomial $P_4(x)$ (blue) of degree 4 passing 5 data points (black)

Polynomial approximation clearly depends on the interpolation partition (data points) considered.

Interpolation for Function Approximation

Approximation accuracy is given by error $e(x) = f(x) - P_n(x)$.
One measure of the error is its **infinity norm**:

$$\|e(x)\|_\infty = \|f(x) - P_n(x)\|_\infty = \max_{x \in [a,b]} |f(x) - P_n(x)|$$

Error for polynomial interpolation is given by:

$$\begin{aligned} f(x) - P_n(x) &= (x - x_0)(x - x_1)\dots(x - x_n)\frac{f^{(n+1)}(c_x)}{(n+1)!} \\ &= \Psi_n(x)\frac{f^{(n+1)}(c_x)}{(n+1)!}, \end{aligned}$$

where c_x is some point on $[a, b]$.
Behavior (distribution and magnitude) of error depends on the shape of function $\Psi_n(x)$ (see Lecture 4)

Interpolation for Function Approximation

Generally, if we have more interpolation points, and hence the degree of the interpolation polynomial is higher, then the error will be smaller:

$$\|e(x)\|_\infty = \|f(x) - P_n(x)\|_\infty \to 0 \text{ as } n \to \infty.$$

But ... interpolation points should be chosen appropriately.

If the interpolation partition is uniform (equally spaced) then pathological Runge example might happen (see Lecture 4).

Another Runge Example

Function $f(x) = \frac{1}{1+25x^2}$ (red) and its polynomial approximations on evenly spaced points P_4, P_8, P_{12} and P_{16}

Approximation Problem

We can build different approximating polynomials of the same degree n for the same function $f(x)$:

- Taylor polynomial of degree n.
- Interpolation polynomials (based on various $n+1$ data points).

Question

Do we have the best approximation among all polynomials of degree n for a given function $f(x)$?

Best means the smallest error measure:

$$\rho_n(f) = \min_{P_n} \|f(x) - P_n(x)\|_\infty = \min_{P_n} \max_{x\in[a,b]} |f(x) - P_n(x)|.$$

Answer is YES!
(Best approximation, Lecture 5),
but practical implementation is cumbersome.

Near MiniMax Approximation

A practical alternative for best approximation is the near-minimax approximation (also known as Chebyshev approximation) based on Chebyshev polynomials (Lecture 5).

Chebyshev approximation of the function $f(x) = \frac{1}{1+25x^2}$

Notice that, using Chebyshev points, we overcame the pathology of Runge Example.

Lebesgue constant

In other words, interpolation points chosen for approximation have big effect on how accurately the interpolation polynomial approximates $f(x)$.

Chebyshev interpolation points were chosen in order to minimize the interpolation error formula for $|f(x) - P_n(x)|$.

This formula depends on function $f(x)$ and interpolation partition.

On the other hand, we would prefer to have a measure of interpolation accuracy that is independent of f.

Something that would measure the quality of interpolation points. This is provided by so-called **Lebesgue constant**.

Lebesgue constant

Definition

Let $\mathcal{P}$ denote a set of interpolation points:

$$\mathcal{P} = \{x_0, x_1, \ldots, x_n\} \subset [a, b].$$

The Lebesgue constant of $\mathcal{P}$ is defined as,

$$\Lambda_n(\mathcal{P}) = \max_{x \in [a,b]} \sum_{k=0}^{n} |L_k(x)|,$$

where $L_k(x)$ are the Lagrange basis functions associated with interpolation set $\mathcal{P}$ (see Lecture 4).

Small Lebesgue constant means that our interpolation can't be much worse that the best polynomial approximation!

Lebesgue constant

Plot of $\sum_{k=0}^{10} |L_k(x)|$ for uniform partition $\mathcal{P}_{unif}$ and Chebyshev partition $\mathcal{P}_{cheb}$ with 11 data points in $[-1, 1]$.

$\Lambda_{10}(\mathcal{P}_{unif}) \approx 29.9$

$\Lambda_{10}(\mathcal{P}_{cheb}) \approx 2.49$

Lebesgue constant

Plot of $\sum_{k=0}^{20} |L_k(x)|$ for uniform partition $\mathcal{P}_{unif}$ and Chebyshev partition $\mathcal{P}_{cheb}$ with 21 data points in $[-1, 1]$.

$\Lambda_{20}(\mathcal{P}_{unif}) \approx 10987$

$\Lambda_{20}(\mathcal{P}_{cheb}) \approx 2.9$

Lebesgue constant

Plot of $\sum_{k=0}^{30} |L_k(x)|$ for uniform partition $\mathcal{P}_{unif}$ and Chebyshev partition $\mathcal{P}_{cheb}$ with 31 data points in $[-1, 1]$.

$\Lambda_{30}(\mathcal{P}_{unif}) \approx 6600000$

$\Lambda_{30}(\mathcal{P}_{cheb}) \approx 3.15$

Lebesgue constant

The explosive growth of $\Lambda_n(\mathcal{P}_{unif})$ is an explanation for Runge example pathology.

Also, it has been shown that as $n \to \infty$,

$$\Lambda_n(\mathcal{P}_{unif}) \approx \frac{2^n}{e\, n \log n}, \qquad \text{BAD!}$$

whereas

$$\Lambda_n(\mathcal{P}_{cheb}) < \frac{2}{\pi} \log(n+1) + 1. \qquad \text{GOOD!}$$

Conclusions

1. **Polynomial interpolation purpose 1 (fitting discrete data)**
   - There is a unique polynomial $P_n(x)$ that fits the data.
   - Should use Newton divided difference formula.
   - Avoid equally spaced (uniform) partition of data points.
2. **Polynomial interpolation purpose 2 (approximating functions)**
   - For a given set of interpolation points, use point 1 methodology to build the approximation $P_n(x)$.
   - Interpolation points play an important role on the size of error $\|f(x) - P_n(x)\|_\infty$ (keep in mind Runge example).

Piecewise polynomial interpolation

Problem

Is it possible, given a set of data points, to build a function (piecewise polynomial) s.t. the "shape" of the data is preserved?

Answer is YES: Cubic spline functions (piecewise cubic polynomials with additional properties) (See Lecture 6).

Piecewise linear and cubic spline interpolations of Runge function

Numerical Analysis for Calculus

In the next lectures, we will discuss the development and application of numerical methods to problems of Calculus:

- Integration;
- Differentiation;
- Solving ODE (ordinary differential equations);
- Solving PDE (partial differential equations) (few slides);
- Optimization.

Homework 1

Due February 11, 19.00

Problem 1.1

All readings are posted on the course web page.

a) Read Lectures 0, 1.1, 1.2, 1.3.

b) Review how to do binary to decimal and vice-versa conversions.

c) Read definitions of Numerical Analysis by K. Atkinson and L. Trethenen.

d) Read the history of IEEE standard 754.

e) Read the paper on some common bugs related to computer representation of numbers.

Problem 1.2

Write the binary single precision IEEE floating-point expression for the number 12.1875. Specify sign σ, exponent E and mantissa.

Problem 1.3

Some microcomputers in the past used a binary floating-point format with 7 bits for the exponent and 1 bit for the sign σ. The mantissa contained 16 bits, with no hiding of the leading bit 1. The arithmetic used chopping. Determine the accuracy of the representation by finding the following:

a) machine epsilon;

b) integer M;

c) accuracy of the chopping operation.

Problem 1.4

Consider a binary floating-point representation with mantissa containing 3 digits without hiding the leading 1 and $-3_{10} \leqslant e \leqslant 3_{10}$.

a) List all numbers that can be stored exactly together with their decimal value.

b) Plot these numbers on real axis.

c) For this arithmetic, specify what are the corresponding floating-point representation of $\pi/3$ and $12/7$ if rounding is used.

d) Repeat c) if chopping is being used.

Problem 1.5

Calculate the error, relative error and the number of significant digits in the following approximations $x_A \approx x_T$.

a) $x_A = 6435.4012, \quad x_T = 6435.401163;$

b) $x_A = 0.007245, \quad x_T = 0.00723816;$

c) $x_A = 355/113, \quad x_T = \pi;$

a) $x_A = 2.236, \quad x_T = \sqrt{5}.$

Problem 1.6

Avoid loss-of-significance errors in the following formulas

$$
\begin{aligned}
&a) \quad \log(x) - \log(x-1) \quad \text{for large values of } x;\\
&b) \quad \frac{e^x - 1}{x} \quad \text{for small values of } x;\\
&c) \quad \cos(x+a) - \cos(a) \quad \text{for small values of } x;
\end{aligned}
$$

Problem 1.7

In the following function evaluations $f(x_A)$, assume the numbers x_A are correctly rounded to the number of digits shown. Bound the error $f(x_T) - f(x_A)$ and the relative error $Rel(x_A)$:

$$
\begin{aligned}
&a) \quad \sin(0.521);\\
&b) \quad e^{3.22};\\
&c) \quad \sqrt{0.0011};\\
&d) \quad \arcsin(0.5).
\end{aligned}
$$

Practice problems 2

Problem 2.1

Let the interval used in the bisection method have the length $b - a = 3$. Find the number of midpoints c_n that must be calculated with the bisection method to obtain an approximate root within an error tolerance of 10^{-9}.

Problem 2.2

Imagine you are finding a root α satisfying $1 < \alpha < 2$. If you are using a binary computer with m digits in its significand, what is the smallest error tolerance that makes sense in finding an approximation to α? If the original interval is $[1, 2]$ how many halving are needed to find an approximation to α with the maximum accuracy possible for this computer?

Problem 2.3

Work out what the Newton iteration is for $f(x) = x^2$. What is the solution to $f(x) = 0$? Will the sequence generated by Newton method converge to solution? How quickly? Relate this to the theory of Newton method.

Problem 2.4

On most computers, the computation of $\sqrt{a}$ is based on Newton's method. Set up the Newton's iteration for solving $x^2 - a = 0$, and show that it can be written in the form

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right), \qquad n \geq 0.$$

Derive the error and relative error formulas:

$$\begin{aligned} \sqrt{a} - x_{n+1} &= -\frac{1}{2x_n}\left(\sqrt{a} - x_n\right)^2, \\ \mathrm{Rel}(x_{n+1}) &= -\frac{\sqrt{a}}{2x_n}\left(\mathrm{Rel}(x_n)\right)^2. \end{aligned}$$

For initial guess x_0 near $\sqrt{a}$, the last formula becomes

$$\mathrm{Rel}(x_{n+1}) \approx -\frac{1}{2}\left(\mathrm{Rel}(x_n)\right)^2$$

Assuming $\mathrm{Rel}(x_0) = 0.1$, use this formula to estimate the relative error in $x_i, i = 1, 2, 3, 4$.

Problem 2.5

Derive formula

$$\mathrm{Rel}(x_{n+1}) = \left(\mathrm{Rel}(x_n)\right)^2$$

for the Newton's iterations used in computing $\frac{1}{b}$ for given b (formula was discussed in class without proof).

Problem 2.6

How many solutions are there to the equation $x = e^{-x}$? Will the iteration $x_{n+1} = e^{-x_n}$ converge for a suitable choice of x_0? Use Aitken extrapolation formula to estimate the error $\alpha - x_3$ for $x_0 = 0.57$.

Problem 2.7

The iteration

$$x_{n+1} = 2 - (1 + c)x_n + cx_n^3$$

will converge to $\alpha = 1$ for some values of c (provided that initial guess x_0 is chosen sufficiently close to α). Find the values of c for which convergence occurs. For what values of c, if any, convergence will be quadratic?

Problem 2.8

Consider the equation

$$x^7 - 28x^6 + 322x^5 - 1960x^4 + 6769x^3 - 13132x^2 - 5040 = 0$$

Change the coefficient of x^4 from -1960 to -1960.14. What is relative perturbation error in the coefficient of x^4? Calculate $\alpha(\varepsilon)$ for $\alpha(0) = 3$ and $\alpha(0) = 5$.

Problem 2.9

What is the order of convergence of the iteration

$$x_{n+1} = \frac{x_n(x_n^2 + 15)}{3x_n^2 + 5}$$

as it converges to the fixed point $\alpha = \sqrt{5}$?

Problem 2.10

Newton's method is used to find the root of $f(x) = 0$. The first few iterates are shown in the following table, giving a very slow speed of convergence. What can be said about the root α to explain the convergence? Knowing $f(x)$, how would you find an accurate value for α?

n	x_n	$x_{n-1} - x_n$
0	0.75	
1	0.752710	0.00271
2	0.754795	0.00208
3	0.756368	0.00157
4	0.757552	0.00118
5	0.758441	0.000889

Problem 2.11

Consider the following table of iterates from an iteration method which is convergent to a fixed point α of the function $g(x)$:

n	x_n	$x_n - x_{n-1}$
0	1.30499998	
1	1.25340617	$-5.159E-2$
2	1.21676284	$-3.664E-2$
3	1.19087998	$-2.588E-2$
4	1.17257320	$-1.831E-2$
5	1.15962919	$-1.294E-2$

(a) Does this appear to be a linearly convergent iteration method? If so, then estimate the rate of linear convergence. (b) Estimate the error in x_5 . (c) Give an improved estimate of α .

Practice set 2

ANSWERS

Problem 2.1

Use the error formula for bisection method to get:

$$|\alpha - c_n| \leq \frac{3}{2^n} \leq \varepsilon = 10^{-9}$$
$$\Leftrightarrow n \geq \frac{\ln\left(\frac{3}{10^{-9}}\right)}{\ln 2} \approx 31.48.$$

Therefore $n \geq 32$.

Problem 2.2

The smallest error tolerance that makes sense (since the root is between 1 and 2, no subnormal numbers are used) is the machine epsilon, in this case $\varepsilon = 2^{-m}$. Since we should have

$$\begin{aligned} n &\geq \frac{\ln\left(\frac{1}{2^{-m}}\right)}{\ln 2} \\ &= \frac{\ln\left(2^m\right)}{\ln 2} \\ &= m, \end{aligned}$$

clearly the number of halvings will be m.

Problem 2.3

Solution, obviously is $\alpha = 0$. Newton's method is

$$\begin{aligned} x_{n+1} &= x_n - \frac{x_n^2}{2x_n} \\ &= \frac{1}{2}x_n = \frac{1}{2}\left(\frac{1}{2}x_{n-1}\right) = \frac{1}{2^2}\left(\frac{1}{2}x_{n-2}\right) \\ &= \dots \\ &= \frac{x_0}{2^{n+1}}, \end{aligned}$$

where x_0 is the initial guess. It can be seen that $x_{n+1} \to 0$ for any value of x_0, in other words, the Newton's method will converge, no matter what initial guess is chosen. In order to see the speed of convergence, first we can observe a similar behavior to the behavior of the bisection method (look for the error formula for bisection method).

Therefore, a linear convergence will be expected. Actually, we can easily derive a formula for the error in this case:

$$\begin{aligned} x_{n+1} - \alpha &= x_{n+1} \\ &= \frac{1}{2}x_n \\ &= \frac{1}{2}(x_n - \alpha) \end{aligned}$$

Thus, we have a linear convergence with linear rate $\frac{1}{2}$.

Theoretically, the convergence for Newton's method should be quadratic, but in this case the convergence speed is lower. (Because of the multiplicity of the root!)

Problem 2.4

Let a be given and set $\alpha = \sqrt{a}$ and $f(x) = x^2 - a$. Then for any $n \geq 0$ we have

$$\begin{aligned} x_{n+1} &= x_n - \frac{f(x_n)}{f'(x_n)} \\ &= x_n - \frac{x_n^2 - a}{2x_n} \\ &= \frac{1}{2}\left(x_n + \frac{a}{x_n}\right). \end{aligned}$$

In class we have shown the formula

$$\begin{aligned} \sqrt{a} - x_{n+1} &= -\frac{(\sqrt{a} - x_n)^2}{2} \cdot \frac{f''(c_n)}{f'(x_n)} \\ &= -\frac{(\sqrt{a} - x_n)^2}{2} \cdot \frac{2}{2x_n} \\ &= -\frac{(\sqrt{a} - x_n)^2}{2x_n}. \\ \mathrm{Rel}(x_{n+1}) &= \frac{\sqrt{a} - x_{n+1}}{\sqrt{a}} \\ &= \frac{-\frac{(\sqrt{a}-x_n)^2}{2x_n}}{\sqrt{a}} \\ &= -\frac{(\sqrt{a} - x_n)^2 \cdot \sqrt{a}}{2x_n (\sqrt{a})^2} \\ &= -\frac{\sqrt{a}}{2x_n}\left(\frac{\sqrt{a} - x_n}{\sqrt{a}}\right)^2 \\ &= -\frac{\sqrt{a}}{2x_n}(\mathrm{Rel}(x_n))^2. \end{aligned}$$

For initial guess x_0 near $\sqrt{a}$, the method was shown to converge and therefore $x_n \to \sqrt{a}$. Thus for n big enough, $x_n \approx \sqrt{a}$ and the last formula becomes

$$\mathrm{Rel}(x_{n+1}) \approx -\frac{1}{2}(\mathrm{Rel}(x_n))^2$$

Let $\mathrm{Rel}(x_0) = 0.1$, Then

$$\begin{aligned} \mathrm{Rel}(x_1) &\approx -\frac{1}{2}(\mathrm{Rel}(x_0))^2 = -0.005; \\ \mathrm{Rel}(x_2) &\approx -\frac{1}{2}(\mathrm{Rel}(x_1))^2 \approx -1.25 \cdot 10^{-5}; \\ \mathrm{Rel}(x_3) &\approx -\frac{1}{2}(\mathrm{Rel}(x_2))^2 \approx -7.8125 \cdot 10^{-11}; \\ \mathrm{Rel}(x_4) &\approx -\frac{1}{2}(\mathrm{Rel}(x_3))^2 \approx -3.0518 \cdot 10^{-21}. \end{aligned}$$

Problem 2.5

Let $f(x) = b - \frac{1}{x}$. Root is $\alpha = \frac{1}{b}$. The relative error according to its definition is

$$\begin{aligned} \mathrm{Rel}(x_{n+1}) &= \frac{\alpha - x_{n+1}}{\alpha} \\ &= \frac{\frac{1}{b} - x_{n+1}}{\frac{1}{b}} \\ &= 1 - bx_{n+1}. \end{aligned}$$

In class we showed that Newton's method for the equation $b - \frac{1}{x} = 0$ becomes

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

Thus, combining the last two formulas we have

$$\begin{aligned}
\mathrm{Rel}(x_{n+1}) &= 1 - bx_{n+1} \\
&= 1 - bx_n(2 - bx_n) \\
&= 1 - 2bx_n + b^2x_n^2 \\
&= (1 - bx_n)^2 \\
&= (\mathrm{Rel}(x_n))^2
\end{aligned}$$

Problem 2.6

From the graph it can be seen that we have only one root α between 0.5 and 0.7.

Let $g(x) = e^{-x} \Rightarrow g'(x) = -e^{-x}$. Since

$$\max_{x\in[0.5,0.7]} |g'(x)| = e^{-0.5} \approx 0.6065 < 1,$$

the fixed point iterates $x_{n+1} = e^{-x_n}$ will converge for $x_0 \in [0.5, 0.7]$ (Actually, any $x_0 > 0$ will do fine.). The Aitken extrapolation formula for the error $\alpha - x_3$ is

$$\alpha - x_3 \approx \frac{\lambda_3}{1 - \lambda_3}(x_3 - x_2),$$

where

$$\lambda_3 = \frac{x_3 - x_2}{x_2 - x_1}.$$

We have

i	x_i	λ_i
0	0.57	
1	$5.6553E-1$	
2	$5.6806E-1$	
3	$5.6662E-1$	$-5.6734E-1$

Thus

$$\alpha - x_3 \approx 5.2084E - 4.$$

Problem 2.7

Convergence will happen if

$$|g'(\alpha)| < 1$$

where $g(x) = 2 - (1+c)x + cx^3$ and $\alpha = 1$. Therefore, we have the following condition

$$\begin{aligned} |-(1+c)+3c| &< 1 \Leftrightarrow \\ |-1+2c| &< 1 \Leftrightarrow \\ -1 &< -1+2c < 1 \Leftrightarrow \\ 0 &< c < 1 \end{aligned}$$

Convergence will be at least quadratic, if $g'(1) = 0 \Leftrightarrow -1+2c = 0 \Leftrightarrow c = \frac{1}{2}$. It should be remarked that $g''(1) = 3x \Rightarrow g''(1) \neq 0$. Therefore, convergence will be exactly quadratic.

Problem 2.8

$$|\mathrm{Rel}(-1960.14)| = \frac{-1960-(-1960.14)}{-1960} = 7.1429E-5$$

Consider the root $\alpha(0) = 3$. Let $g(x) = x^4$ and $f(x) = x^7 - 28x^6 + 322x^5 - 1960x^4 + 6769x^3 - 13132x^2 + 13680x - 5040$. Then $f'(x) = 7x^6 - 168x^5 + 1610x^4 - 7840x^3 + 20307x^2 - 26264x + 13680$ and $f'(3) = 660$. From the formula

$$\begin{aligned} \alpha(\varepsilon) &\approx \alpha(0) - \varepsilon\frac{g(\alpha(0))}{f'(\alpha(0))} \\ &= 3 - \varepsilon\frac{g(3)}{f'(3)} \\ &= 3 - 0.14 \cdot \frac{3^4}{660} \\ &\approx 2.9828. \end{aligned}$$

Consider the root $\alpha(0) = 5$. Then $f'(5) = 660$. Similarly,

$$\begin{aligned} \alpha(\varepsilon) &\approx 5 - \varepsilon\frac{g(5)}{f'(5)} \\ &= 5 - 0.14 \cdot \frac{5^4}{660} \\ &\approx 4.8674. \end{aligned}$$

Problem 2.9

Let

$$g(x) = \frac{x(x^2+15)}{3x^2+5}.$$

First, check that $\sqrt{5}$ is a fixed point for function g. Indeed $g(\sqrt{5}) = \sqrt{5}$. Then compute

$$\begin{aligned} g'(x) &= \frac{9x^4 - 30x^2 + 75}{(3x^2+5)^2} \\ g'(\sqrt{5}) &= 0. \end{aligned}$$

Thus, the order of convergence is at least quadratic.

$$\begin{aligned} g''(x) &= \frac{12x(-3x^4 + 31x^2 - 80)}{(3x^2+5)^4} \\ g''(\sqrt{5}) &= 0. \end{aligned}$$

Therefore, the order of convergence is at least cubic. It can be checked that

$$g^{(3)}(\sqrt{5}) \neq 0$$

So, the order of convergence is cubic.

Problem 2.10

n	x_n	$x_{n-1} - x_n$	λ_n
0	0.75		
1	0.752710	0.00271	
2	0.754795	0.00208	0.76753
3	0.756368	0.00157	0.75481
4	0.757552	0.00118	0.75159
5	0.758441	0.000889	0.75339

We can see that $\lambda_n \to 0.75 = \frac{3}{4}$. Therefore, we can say that our root have multiplicity 4. In order to find an accurate value of α, we compute the $f^{(3)}(x)$ and apply the Newton's method to this equation $f^{(3)}(x) = 0$. Since α is a simple root of $f^{(3)}(x)$,Newton$'$s method should converge much faster.

Problem 2.11

n	x_n	$x_n - x_{n-1}$	λ_n
0	1.30499998		
1	1.25340617	$-5.159E-2$	
2	1.21676284	$-3.664E-2$	$7.102E-1$
3	1.19087998	$-2.588E-2$	$7.063E-1$
4	1.17257320	$-1.831E-2$	$7.075E-1$
5	1.15962919	$-1.294E-2$	$7.06717E-1$

Since λ_n obviusly are converging to 0.707, we have $g'(\alpha) \approx 0.707$ and

$$\alpha - x_{n+1} \approx 0.707(\alpha - x_{n+1})$$

which means that convergence is linear with linear rate approximately 0.707. By Aitken error estimation formula we have

$$\begin{aligned} \alpha - x_5 &\approx \frac{\lambda_5}{1-\lambda_5}(x_5 - x_4) \\ &\approx \frac{0.706717}{1-0.76717} \cdot (-0.001294) \\ &\approx -3.92772E-3. \end{aligned}$$

By Aitken extrapolation formula

$$\begin{aligned} \alpha &\approx x_5 + \frac{\lambda_5}{1-\lambda_5}(x_5 - x_4) \\ &\approx 1.15962919 + (-3.92772E-3) \\ &= 1.15570147. \end{aligned}$$

Homework 2

Due March 18, 19:00

Problem 2.1

The **error function** (also called Gauss error function) is defined by

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\, dt.$$

Use Taylor series to approximate $\operatorname{erf}(x)$ with a polynomial $T_n(x)$. What is n, if the desired accuracy is 10^{-5}? Using this approximation, plot the graph of $\operatorname{erf}(x)$ on $[-3, 3]$.

Problem 2.2

Consider the sequence of Fibonacci numbers F_n:

$$F_0 = 1, \quad F_1 = 1, \quad F_n = F_{n-1} + F_{n-2}, \quad n = 2, 3, \ldots$$

Let $R_n = \frac{F_{n+1}}{F_n}$. It can be shown that

$$\lim_{n\to\infty} R_n = \frac{1+\sqrt{5}}{2} \equiv \phi,$$

which is known as **golden ratio**. Write a code that will compute numerically the first 40 terms of the sequence R_n together with errors $\phi - R_n$. In computations make sure that you are using IEEE double precision. Comment your results. What can be said on the order of convergence?

Problem 2.3

Thermistors are temperature-measuring devices based on the principle that the thermistor material exhibits a change in electrical resistance with a change in temperature. By measuring the resistance of the thermistor material, one can then determine the temperature. For a 10K3A Betatherm thermistor, the relationship between

Figure 1: A typical thermistor

the resistance R of the thermistor and the temperature T is given by

$$\frac{1}{T} = 1.129241 \times 10^{-3} + 2.341077 \times 10^{-4} \log R + 8.775468 \times *10^{-8} \Big(\log R\Big)^3$$

where T is in Kelvin and R is in Ohms, and log denotes the natural logarithm. A thermistor error of no more than $\pm 0.01^oC$ is acceptable. To find the range of the resistance that is within this acceptable limit at 19^oC, we need to solve

$$\frac{1}{19.01 + 273.15} = 1.129241 \times 10^{-3} + 2.341077 \times 10^{-4} \log R + 8.775468 \times *10^{-8} \Big(\log R\Big)^3 \tag{1}$$

$$\frac{1}{18.99 + 273.15} = 1.129241 \times 10^{-3} + 2.341077 \times 10^{-4} \log R + 8.775468 \times *10^{-8} \Big(\log R\Big)^3 \tag{2}$$

Write a computer routine implementing Newton's method and solve equations (1) and (2) using Newton's method with initial guess $R_0 = 15000$ and error tolerance of 10^{-5}.
What is the obtained range for resistance values?

Problem 2.4

Consider the function $f(x) = e^{x-\pi} + \cos x - x + \pi$.

a) Plot its graph on interval $[0, 5]$.

b) Apply Newton's method routine you developed in **Problem 2.3** to solve equation $f(x) = 0$ on interval $[0, 5]$. What can be said about its order of convergence? Argue why this is happening? How its convergence order can be improved?

c) Write a modified routine that will ensure **quadratic** convergence and apply it.

d) Instead of solving $f(x) = 0$, try to apply the fixed point iterations $x_{n+1} = e^{x_n-\pi} + \cos x_n + \pi$. Comment on your results.

Problem 2.5

a) Compute the fixed point $x_{n+1} = \cos x_n - 1 + x_n$ with initial guess $x_0 = 0.1$.

b) What can be said about the speed of convergence? Compare it with bisection method.

c) Write a modified computer routine that will speed up the convergence.

Problem 2.6

Newton's method is used to find the root α of $f(x) = 0$. The first 10 iterates are shown in the table below.

(1) What can be said about the order of convergence? Is it slower or faster than bisection method?
(2) What can be said about the root α to explain this convergence?
(3) Knowing function $f(x)$, how would you speed up the convergence?

n	x_n	$x_n - x_{n-1}$
0	2.0	
1	2.1248	0.124834
2	2.2148	0.089944
3	2.2805	0.065698
4	2.3289	0.048386
5	2.3647	0.035827
6	2.3913	0.026624
7	2.4111	0.019835
8	2.4260	0.014803
9	2.4370	0.011062
10	2.4453	0.0082745

Problem 2.7

For solving the equation $x + \ln x = 0$, there were proposed three methods:

$$\begin{aligned} (a) \quad x &= -\ln x \\ (b) \quad x &= e^{-x} \\ (c) \quad x &= \frac{x + e^{-x}}{2} \end{aligned}$$

(1) Which of the formulas can be used?
(2) Which of the formulas should be used?
(3) Give an even better formula!

Problem 2.8

Consider the following table of iterates from an iteration method which is convergent to a fixed point α of the function $g(x)$:

n	x_n	$x_n - x_{n-1}$
0	1.00	
1	0.36788	$-6.3212E-01$
2	0.69220	$3.2432E-01$
3	0.50047	$-1.9173E-01$
4	0.60624	$1.0577E-01$
5	0.54540	$-6.0848E-02$
6	0.57961	$3.4217E-02$

(1) Show that this is a linearly convergent iteration method.
(2) Find its rate of linear convergence. Is this method faster or slower than bisection method?
(3) Propose a way to accelerate the convergence of this method?

Problem 2.9

BONUS. Benout B. Mandelbrot, a famous mathematician is known as the inventor of *fractals* and this problem is dedicated to him. In this problem you will generate the so-called **quadratic Julia Sets**, a well-known fractal example.

Given two complex numbers, c and z_0 the following recursion (it is similar to fixed point iterations) is defined

$$z_n = z_{n-1}^2 + c.$$

For an arbitrary given choice of c and z_0, this recursion leads to a sequence of complex numbers z_1, z_2, $z_3, \ldots$ called the **orbit** of z_0. Depending on the exact choice of c and z_0, a large range of orbit patterns are possible.

For a given fixed c, most choices of z_0 yield orbits that tend towards infinity. (That is, $|z_n| \to \infty$ as $n \to \infty$).

For some values of c certain choices of z_0 yield orbits that eventually go into a periodic loop. Finally, some starting values yield orbits that appear to dance around the complex plane, apparently at random (an example of chaos). These initial values of z_0 make up the Julia set of this recursion, denoted by J_c.

Write a MATLAB/GNU Octave/Python script that visualizes a slightly different set, called the filled-in Julia set denoted by K_c, which is the set of all z_0 with orbits which do not tend towards infinity. The "normal" Julia set J_c is the edge of the filled-in Julia set. The figure below illustrates a filled-in Julia Set for one particular value of c.

a) It has been shown that if $|z_n| > 2$ for some n, then it is guaranteed that the orbit will tend to infinity. The value of n for which this becomes true is called the **escape velocity** of a particular z_0. Write a function that returns the escape velocity of a given z_0 and c. The function declaration should be: $n =$`EscVel`(z_0, c, N), where N is the maximum allowed escape velocity (i.e. if $|z_n| \leq 2$ for $n < N$, return N as the escape velocity, so you will prevent infinite loops).

b) To generate the filled-in Julia Set, write the following function

$$M = \texttt{JuliaSet}(z_{Max}, c, N),$$

where z_{Max} will be the maximum of the real and imaginary parts of the various values of z_0 for which we will compute escape velocities, c and N are the same as defined above, and M is the matrix that contains the escape velocity of various z_0.

- In this function, you first want to make a 500×500 matrix that contains complex numbers with real part between $-z_{Max}$ and z_{Max}, and imaginary part between $-z_{Max}$ and z_{Max}. Call this matrix Z. Make the imaginary part vary along the y-axis of this matrix. You can most easily do this by using `linspace` and `meshgrid` commands from MATLAB/GNU Octave, but you can also do it with a loop.
- For each element of Z, compute the escape velocity (by calling your `EscVel` function) and store it in the same location in a matrix M. When done, the matrix M should be the same size as Z and contain escape velocities with values between 1 and N.
- Run your `JuliaSet` function with various z_{Max}, c and N values to generate various fractals. To display the fractal nicely, use `imagesc` command to visualize $\arctan(0.1 * M)$, (taking the arctangent of M makes the image look nicer, you also can use `axisxy` command so that y values aren't flipped).

The figure below shows a Julia Set for $c = -.297491 + i * 0.641051$.

Figure 2: $M =$`JuliaSet`$(1, -.297491 + i * 0.641051, 100)$

Julia Sets are the boundaries of more general Mandelbrot sets. There is a chapter in Clive Moler textbook "Experiments with MATLAB" `www.mathworks.com/moler/exm/chapters.html` dedicated to Mandelbrot sets. The difference between Julia Sets and Mandelbrot set is presented in `www.karlsims.com/julia.html`, the figure below was taken from there.

Figure 3: Different Julia Sets and their relation to Mandelbrot Set.

Practice Problems Set 3

Problem 3.1

Find a polynomial $P(x)$ of degree ≤ 3 for which

$$\begin{aligned} P(0) &= y_1 \qquad & P(1) &= y_2 \\ P'(0) &= y_1' \qquad & P'(1) &= y_2' \end{aligned}$$

with y_1, y_2, y_1', y_2' given constants.
The resulting polynomial is called **cubic Hermite interpolating polynomial**.
HINT: Write $P(x) = y_1 H_1(x) + y_2 H_2(x) + y_1' H_3(x) + y_2' H_4(x)$ with H_i cubic polynomials satisfying appropriate properties, in analogy with Lagrange interpolating polynomials.

Problem 3.2

Find the function $P(x) = a + b\cos(\pi x) + c\sin(\pi x)$, which interpolates the data

x	0	0.5	1
y	2	5	4

This is so-called **trigonometric interpolation**. Also, find the quadratic polynomial interpolating this data. In each instance, draw the graph of the interpolating function.

Problem 3.3

Find cubic polynomial interpolating the data

x	0	1	2	5
y	-1	4	2	6

Problem 3.4

Find the polynomial interpolating the data

x	-3	1	2	4	5
y	9	1	4	16	25

Problem 3.5

Prove that

$$\sum_{i=0}^{N} L_i(x) = 1,$$

where $L_i(x)$ are Lagrange basis functions associated to $N+1$ interpolation points.

Problem 3.6

Consider the polynomial interpolation of the function $f(x) = e^{-x^2}$ on $[0,1]$ at the points $x_0 = 0$, $x_1 = 0.5$ and $x_2 = 1$. Estimate the maximum of the polynomial interpolation error for $x \in [0,1]$, i.e. give an upper bound for this error.

Problem 3.7

Is the following a cubic spline on the interval $0 \leq x \leq 2$?

$$s(x) = \begin{cases} (x-1)^3, & 0 \leq x \leq 1 \\ 2(x-1)^3, & 1 \leq x \leq 2 \end{cases}$$

Problem 3.8

Consider the data

x	0	1/2	1	2	3
y	0	1/4	1	-1	-1

(a) Find the piecewise linear interpolating function for the data; (b) Find the piecewise quadratic interpolating function for the data. (c) Find the natural cubic spline that interpolates the data. Graph all three graphs for $0 \leq x \leq 3$.

Problem 3.9

Compute the error bound for the minimax approximation of the function $f(x) = e^{3x-1}$ on the $[-1, 2]$ and $n = 5$.

Problem 3.10

How many multiplications and additions are needed to compute Chebyshev polynomials $T_0(x), T_1(x), T_2(x), \ldots, T_n(x)$ for a particular value of x?

Problem 3.11

Let $q(x)$ be a polynomial of degree $\leq n-1$, and consider

$$\max_{-1 \leq x \leq 1} |x^n - q(x)|$$

What is the smallest possible value for this quantity? Solve for the $q(x)$ for which this value is attained.

Problem 3.12

For $n, m \geq 0$ and $n \neq m$ show

$$\int_{-1}^{1} \frac{T_n(x) T_m(x)}{\sqrt{1-x^2}} dx = 0$$

This is called the orthogonality property for the Chebyshev polynomials.

Practice Set 3

ANSWERS

Problem 3.1

Consider a general cubic polynomial

$$P(x) = a + bx + cx^2 + dx^3.$$

Then

$$P'(x) = b + 2cx + 3dx^2.$$

From conditions

$$y_1 = P(0), \quad y_2 = P(1), \quad y_1' = P'(0), \quad y_2' = P'(1),$$

we get

$$\begin{aligned} y_1 &= P(0) = a, \\ y_2 &= P(1) = a + b + c + d, \\ y_1' &= P'(0) = b, \\ y_2' &= P'(1) = b + 2c + 3d, \end{aligned}$$

Solve this system with unknowns a, b, c and d

$$\begin{aligned} y_1 &= a, \\ y_2 &= a + b + c + d, \\ y_1' &= b, \\ y_2' &= b + 2c + 3d, \end{aligned}$$

and obtain solution

$$\begin{aligned} a &= y_1, \\ b &= y_1', \\ c &= -3y_1 + 3y_2 - 2y_1' - y_2', \\ d &= 2y_1 - 2y_2 + y_1' + y_2', \end{aligned}$$

Thus we have

$$\begin{aligned} P(x) &= y_1 + y_1'x + \big(-3y_1 + 3y_2 - 2y_1' - y_2'\big)x^2 + \big(2y_1 - 2y_2 + y_1' + y_2'\big)x^3 \\ &= \big(2x^3 - 3x^2 + 1\big)y_1 + \big(-2x^3 + 3x^2\big)y_2 + \big(x^3 - 2x^2 + x\big)y_1' + \big(x^3 - x^2\big)y_2' \\ &= \big(1 + 2x\big)\big(1 - x\big)^2 y_1 + x^2\big(3 - 2x\big)y_2 + x\big(1 - x\big)^2 y_1' + x^2\big(x - 1\big)y_2' \\ &= H_1(x)y_1 + H_2(x)y_2 + H_3(x)y_1' + H_4(x)y_2' \end{aligned}$$

with

$$\begin{aligned} H_1(x) &= \big(1 + 2x\big)\big(1 - x\big)^2, \\ H_2(x) &= x^2\big(3 - 2x\big), \\ H_3(x) &= x\big(1 - x\big)^2, \\ H_4(x) &= x^2\big(x - 1\big), \end{aligned}$$

Observe that

$$\begin{array}{llll} H_1(0) = 1, & H_1(1) = 0, & H_1'(0) = 0, & H_1'(1) = 0, \\ H_2(0) = 0, & H_2(1) = 1, & H_2'(0) = 0, & H_2'(1) = 0, \\ H_3(0) = 0, & H_3(1) = 0, & H_3'(0) = 1, & H_3'(1) = 0, \\ H_4(0) = 0, & H_4(1) = 0, & H_4'(0) = 0, & H_4'(1) = 1, \end{array}$$

Problem 3.2

Look for function $Q_1(x) = a + b\cos(\pi x) + c\sin(\pi x)$ such that

$$\begin{cases} a + b\cos(\pi \cdot 0) + c\sin(\pi \cdot 0) = Q_1(0) = 2 \\ a + b\cos(\pi \cdot \frac{1}{2}) + c\sin(\pi \cdot \frac{1}{2}) = Q_(\frac{1}{2}) = 5 \\ a + b\cos(\pi \cdot 1) + c\sin(\pi \cdot 1) = Q_1(1) = 4 \end{cases}$$

This leads to the following system

$$\begin{cases} a + b = 2 \\ a + c = 5 \\ a - b = 4 \end{cases}$$

that has solution $a = 3$, $b = -1$, $c = 2$. Therefore the function is $Q_1(x) = 3 - \cos(\pi x) + 2\sin(\pi x)$.

In order to find the quadratic interpolation polynomial use Newton's divided difference formula. First, need to compute Newton's divided differences:

x	y	D_1	D_2
0	2	6	–8
0.5	5	–2	
1	4		

And interpolating polynomial is $P_2(x) = 2 + 6x - 8x(x - \frac{1}{2}) = -8x^2 + 10x + 2$.

Figure 1: Graphs of quadratic interpolant $P_2(x) = -8x^2 + 10x + 2$ (red) and trigonometric interpolant $Q_1(x) = 3 - \cos(\pi x) + 2\sin(\pi x)$ (blue).

Problem 3.3

Use Newton's divided difference formula

x	y	D_1	D_2	D_3
0	–1	5	–7/2	13/15
1	4	–2	5/6	
2	2	4/3		
5	6			

Interpolating cubic polynomial is $P_3(x) = -1 + 5x - \frac{7}{2}x(x-1) + \frac{13}{15}x(x-1)(x-2)$.

Problem 3.4

Observe that data are satisfying relation $y = x^2$, i.e. interpolating points are located on parabola $y = x^2$. Since interpolation polynomial is unique, the answer is $P(x) = x^2$.

Problem 3.5

Let $N+1$ interpolation points be

$$x_0 < x_1 < x_2 < \cdots < x_{N-1} < x_N$$

and let $L_i(x)$, $i = 0, 1, \ldots, N$ be $N+1$ associated Lagrange basis functions. Need to prove that

$$\sum_{i=0}^{N} L_i(x) = 1. \tag{1}$$

Proof.
Rewrite the left side of the identity (1) in the form

$$1 \cdot L_0(x) + 1 \cdot L_1(x) + 1 \cdot L_2(x) + \cdots + 1 \cdot L_N(x) \tag{2}$$

and compare it with the formula for interpolation polynomial for the data (x_i, y_i), $i = 0, 1, \ldots, N$:

$$y_0 \cdot L_0(x) + y_1 \cdot L_1(x) + y_2 \cdot L_2(x) + \cdots + y_N \cdot L_N(x).$$

Obviuosly (2) represents the interpolation polynomial that interpolates points $\{(x_0, 1), (x_1, 1), (x_2, 1), \ldots, (x_N, 1)\}$. These points lie on the line $y = 1$, therfore, since interpolation polynomial is unique, it follows that interpolation polynomial given by (2) is identical function 1. So

$$1 \cdot L_0(x) + 1 \cdot L_1(x) + 1 \cdot L_2(x) + \cdots + 1 \cdot L_N(x) \equiv 1.$$

and identity (1) is proved. ■

Problem 3.6

Let $P_2(x)$ be the quadratic interpolation polynomial of function e^{-x^2} at the points $x_0 = 0$, $x_1 = \frac{1}{2}$ and $x_2 = 1$. The interpolation error is given by

$$e^{-x^2} - P_2(x) = \frac{x(x-\frac{1}{2})(x-1)}{6}\left(e^{-x^2}\right)'''(\theta) \tag{3}$$

for some $\theta \in [0, 1]$. Let $h = \frac{1}{2}$. Using error formula (3) we get

$$\begin{aligned}\left|e^{-x^2} - P_2(x)\right| &= \frac{\left|x(x-h)(x-2h)\right|}{6}\left|\left(e^{-x^2}\right)'''(\theta)\right| \\ &\le \frac{1}{6} \cdot \max_{x\in[0,1]} \left|x(x-h)(x-2h)\right| \cdot \max_{x\in[0,1]} \left|\left(e^{-x^2}\right)'''(x)\right|\end{aligned} \tag{4}$$

From Lecture 9 (pages $8-10$) we know that

$$\max_{x_0 \le x \le x_2} \left|(x-x_0)(x-x_1)(x-x_2)\right| = \frac{2h^3}{3\sqrt{3}}, \text{ with } h = x_1 - x_0 = x_2 - x_1.$$

Thus,

$$\max_{x\in[0,1]} \left|x(x-h)(x-2h)\right| = \frac{2 \cdot \frac{1}{2^3}}{3\sqrt{3}} = \frac{1}{12\sqrt{3}} \approx 0.048113 \tag{5}$$

Next compute the 3rd derivative of e^{-x^2}:

$$\begin{aligned}\left(e^{-x^2}\right)' &= -2xe^{-x^2} \\ \left(e^{-x^2}\right)'' &= (4x^2 - 2)e^{-x^2} \\ \left(e^{-x^2}\right)''' &= (-8x^3 + 12x)e^{-x^2}\end{aligned}$$

Figure 2: Graph of $(-8x^3 + 12x)e^{-x^2}$

In order to find the maximum of $(-8x^3 + 12x)e^{-x^2}$ on $[0, 1]$, plot its graph.
It follows from the graph that

$$\max_{x\in[0,1]} \left| \left(e^{-x^2}\right)'''(x) \right| \approx 3.9032 \quad \text{at } x \approx 0.52 \tag{6}$$

Substituting (6) and (5) in inequaity (4) we obtain

$$\left| e^{-x^2} - P_2(x) \right| \leq \frac{1}{6} \cdot 0.048113 \cdot 3.9032 \approx 0.031299 = 3.13E - 2$$

Problem 3.7

In order to be a cubic spline, function $s(x)$ should have the properties (see Lecture 11):

1. $s(x)$ is a piecewise cubic polynomial;
2. $s(x)$, $s'(x)$ and $s''(x)$ should be continuous functions.

Let

$$s(x) = \begin{cases} (x-1)^3, & 0 \leq x \leq 1 \\ 2(x-1)^3, & 1 \leq x \leq 2 \end{cases}$$

Obviously condition 1 is satisfied. Then

$$s'(x) = \begin{cases} 3(x-1)^2, & 0 \leq x \leq 1 \\ 6(x-1)^2, & 1 \leq x \leq 2 \end{cases}$$

and

$$s''(x) = \begin{cases} 6(x-1), & 0 \leq x \leq 1 \\ 12(x-1), & 1 \leq x \leq 2 \end{cases}$$

It can be easily checked that $s_-(1) = 0 = s_+(1)$, $s'_-(1) = 0 = s'_+(1)$ and $s''_-(1) = 0 = s''_+(1)$. Thus, condition 2 is also satisfied. Therefore the function $s(x)$ is a cubic spline. Moreover, since $s''(0) = -6 \neq 0$ and $s''(2) = 12 \neq 0$, this is not a "natural" cubic spline.

Problem 3.8

(a) Piecewise linear interpolant $P_{1,p}(x)$ is

$$P_{1,p}(x) = \begin{cases} \frac{1}{2}x, & 0 \le x \le \frac{1}{2} \\ \frac{3}{2}x - \frac{1}{2}, & \frac{1}{2} \le x \le 1 \\ -2x+3, & 1 \le x \le 2 \\ -1, & 2 \le x \le 3 \end{cases}$$

(b) Use Newton's divided differences to find piecewise quadratic interpolant $P_{2,p}(x)$.

x	y	D_1	D_2
0	0	1/2	1
1/2	1/4	3/2	
1	1		

x	y	D_1	D_2
1	1	–2	1
2	–1	0	
3	–1		

On $\{0, 1/2, 1\}$ quadratic polynomial is $0 + 1/2(x-0) + 1(x-0)(x-1/2) = x^2$
and on $\{1, 2, 3\}$ quadratic polynomial is $1 + (-2)(x-1) + 1(x-1)(x-2) = x^2 - 5x + 5$.
Therefore, piecewise quadratic interpolant is:

$$P_{1,p}(x) = \begin{cases} x^2, & 0 \le x \le 1 \\ x^2 - 5x + 5, & 1 \le x \le 3 \end{cases}$$

(c) Since it was not specifically required to obtain the natural cubic spline analytically, we will use GNU Octave/MATLAB built-in functions to get it.

In GNU Octave define the interpolating data and apply *csape* function:

```
>>xx=[0, 0.5, 1, 2, 3];
>>yy=[0, 0.25, 1, -1, -1];
>>pp=csape(xx, yy, 'variational')
pp =
  scalar structure containing the fields:

    form = pp

    breaks =
       0.00000   0.50000   1.00000   2.00000   3.00000

    coefs =
       1.80952   0.00000   0.04762   0.00000
      -5.04762   2.71429   1.40476   0.25000
       2.52381  -4.85714   0.33333   1.00000
      -0.90476   2.71429  -1.80952  -1.00000

    pieces =  4
    order =  4
    dim =  1
```

Matrix `coefs` contains the coefficients of piecewise cubic polynomial:
if row i of matrix `coefs` is $[a\,b\,c\,d]$ then $s(x) = a(x-x_i)^3 + b(x-x_i)^2 + c(x-x_i) + d$ if $x \in [x_i, x_{i+1}]$

Therefore, we have the following natural cubic spline function that interpolates our data.

$$sn(x) = \begin{cases} 1.80952x^3 + 0.04762x, & 0 \le x \le \frac{1}{2} \\ -5.04762(x-0.5)^3 + 2.71429(x-0.5)^2 + 1.40476(x-0.5) + 0.25, & \frac{1}{2} \le x \le 1 \\ 2.52381(x-1)^3 - 4.85714(x-1)^2 + 0.33333(x-1) + 1.0, & 1 \le x \le 2 \\ -0.90476(x-2)^3 + 2.71429(x-2)^2 - 1.80952(x-2) - 1.0, & 2 \le x \le 3 \end{cases}$$

Let's plot all three interpolants:

Figure 3: Graphs of piecewise linear interpolant (red), piecewise quadratic interpolant (black) and natural cubic spline interpolant (blue)

IMPORTANT REMARK:
Beware and read help/refference documentation initially, since, as it was mentioned in class lecture, there are variuous boundary conditions, and thus many types of spline functions exist. For example,

- **natural** (known in Matlab and GNU Octave as 'variational') cubic spline uses condition $s''(x_1) = s''(x_N) = 0$.
- **not-a-knot** cubic spline uses condition that $s'''(x)$ is continuous at x_2 and x_{N-1}.
- **complete** (also known as clamped) cubic spline uses condition $s'(x_1) = A$ and $s'(x_2) = B$ with values A and B provided beforehand.
- **periodic** cubic spline uses condition $s'(x_1) = s'(x_N)$.
- **second** cubic spline uses condition $s'(x_1) = C$ and $s'(x_2) = D$ with values A and B provided beforehand.

In order to obtain natural cubic spline we will use *csape* function provided by MATLAB library. For using it in GNU Octave you need to download and install additional package *splines*. Let Google be with you! I was able to do it, so will you. The available by default with GNU Octave basic distribution function *spline* can do only not-a-knot and clamped cubic splines.

As a bonus let's compare 3 different cubic spline interpolants.

- natural cubic spline obtained previously;
- clamped cubic spline (obtained using `spline` function from GNU Octave);
- not-a-knot cubic spline (aobtained using `spline` function).

Figure 4: Natural cubic spline (blue), clamped cubic spline (red), not-a-knot cubic spline (black)

Problem 3.9

Consider the error formula for minimax approximation (see Lecture 10, page 5)

$$\rho_n(f) \leq \frac{\left(\frac{b-a}{2}\right)^{n+1}}{(n+1)!\,2^n} \max_{x\in[a,b]} \left|f^{(n+1)}(x)\right|$$

Substituting $a = -1$, $b = 2$, $n = 5$ we get error estimate formula

$$\rho_5(f) \leq \frac{\left(\frac{3}{2}\right)^6}{6!\,2^5} \max_{x\in[-1,2]} \left|f^{(6)}(x)\right| = \frac{3^6}{6!\,2^{11}} \max_{x\in[-1,2]} \left|f^{(6)}(x)\right|,$$

where $f(x) = e^{3x-1}$. Compute derivatives

$$\left(e^{3x-1}\right)' = 3e^{3x-1}, \quad \left(e^{3x-1}\right)'' = 9e^{3x-1}, \quad \ldots, \quad \left(e^{3x-1}\right)^{(6)} = 3^6 e^{3x-1}$$

Since

$$\max_{x\in[-1,2]} \left|f^{(6)}(x)\right| = 3^6 \max_{x\in[-1,2]} \left|e^{3x-1}\right| = 3^6 \cdot e^{3\cdot 2-1} = 3^6 e^5$$

error estiamte formula becomes

$$\rho_5(e^{3x-1}) \leq \frac{3^{12}e^5}{6!\,2^{11}} \approx 53.489$$

It seems that this is a very crude estimate, but keep in mind that fucntion e^{3x-1} on interval $[-1, 2]$ grows fast from 0.018 up to 148.41. So we can repeat the same computations for $n = 10$ and $n = 15$ to get

$$\rho_{10}(e^{3x-1}) \leq \frac{3^{22}e^5}{11!\,2^{21}} \approx 5.564E-2, \quad \rho_{15}(e^{3x-1}) \leq \frac{3^{32}e^5}{16!\,2^{31}} \approx 6.1207E-6$$

Problem 3.10

Recall the triple recurrsion formula for Chebyshev polynomials (see Lecture10, page 8)

$$\begin{aligned} T_0(x) &= 1, \\ T_1(x) &= x, \\ T_{n+1}(x) &= (2x) \cdot T_n(x) - T_{n-1}(x), \end{aligned}$$

Evaluation of T_0 and T_1 is straightforward, then T_2 will need 2 multiplications and 1 addition, and every next polynomial will need 1 more multiplication and 1 more addition. Therefore, evaluation at a particular x of Chebyshev polynomials $T_0(x), T_1(x), T_2(x), \dots T_n(x)$ will need n multiplications and $n-1$ additions.

Problem 3.11

Let $q(x)$ be a polynomial of degree $n-1$ and consider the polynomial $x^n - q(x)$. It is a polynomial of degree n, moreover it is a monic polynomial (see Lecture 10). According to the **Theorem on minimum size property** from page 11, the degree n monic polynomial with the smallest maximum on $[-1,1]$ is the modified Chebyshev polynomial $\widetilde{T}_n(x)$, and its maximum value on $[-1,1]$ is $\frac{1}{2^{n-1}}$. Thus,

$$\max_{x\in[-1,1]} \left|x^n - q(x)\right| = \frac{1}{2^{n-1}}$$

and it is achieved for $q(x) = x^n - \widetilde{T}_n(x)$.

Problem 3.12

Proof.
Consider substitution

$$x = \cos\theta, \quad \theta = \arccos x, \quad \sqrt{1-x^2} = \sqrt{1-\cos^2\theta} = \sin\theta, \quad dx = d(\cos\theta) = -\sin\theta\, d\theta.$$

Then

$$\begin{aligned} \int_{-1}^{1} \frac{T_n(x)T_m(x)}{\sqrt{1-x^2}}\, dx &= \int_{\pi}^{0} \frac{\cos(n\theta)\cos(m\theta)(-\sin\theta)\, d\theta}{\sin\theta} \\ &= \int_0^{\pi} \cos(n\theta)\cos(m\theta)\, d\theta \\ &= 0, \text{ if } n \neq m. \end{aligned}$$

■

Lupascu Felicia

Homework 2

Ex.1

Problem 1.2

$$erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

The factor of $\pi/2$ guarantees that $\lim_{x \to \infty} erf(x) = 1$

We can integrate power series $\Rightarrow e^{-t^2} = 1 - t^2 + \frac{t^4}{2!} - \frac{t^6}{3!} + \dots \quad (R = \infty)$

we can now find a power series expansion for the error function

$$erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt = \frac{2}{\sqrt{\pi}} \int_0^x \left(1 - t^2 + \frac{t^4}{2!} - \frac{t^6}{3!} + \dots\right) dt =$$

$$= \frac{2}{\sqrt{\pi}} \left[t - \frac{t^3}{3} + \frac{t^5}{5 \cdot 2!} - \frac{t^7}{7 \cdot 3!} + \dots \right]_0^x = \frac{2}{\sqrt{\pi}} \left(x - \frac{x^3}{3} + \frac{x^5}{5 \cdot 2!} - \frac{x^7}{7 \cdot 3!} + \dots \right)$$

We have to find such n, so the difference between I_n and $erf(x)$ should be less than 10^{-5}

Lupascu Felicia

Ex.2

main.py

```python
memoized = {0: 1, 1: 1}

def fib(n):
  if (n in memoized):
    return(memoized[n])
  else:
    memoized[n] = fib(n-1) + fib(n-2)
    return(memoized[n])

for i in range(40):
  print(i+1,fib(i))
```

Program that generates Fibonacci numbers

main.py

```python
memoized = {0: 1, 1: 1}

def fib(n):
  if (n in memoized):
    return(memoized[n])
  else:
    memoized[n] = fib(n-1) + fib(n-2)
    print("  Golden ratio=",fib(n-1)/fib(n-2))
    return(memoized[n])

for i in range(40):
  print(i+1,fib(i))
```

The ratio of successive Fibonacci numbers approaches the golden ratio. Here's a program to explore this connection.

```
1 1
2 1
3 2
4 3
5 5
6 8
7 13
8 21
9 34
10 55
11 89
12 144
13 233
14 377
15 610
16 987
17 1597
18 2584
19 4181
20 6765
21 10946
22 17711
23 28657
24 46368
25 75025
26 121393
27 196418
28 317811
29 514229
30 832040
31 1346269
32 2178309
33 3524578
34 5702887
35 9227465
36 14930352
37 24157817
38 39088169
39 63245986
40 102334155
```

```
1 1
2 1
    Golden ratio= 1.0
3 2
    Golden ratio= 2.0
4 3
    Golden ratio= 1.5
5 5
    Golden ratio= 1.6666666666666667
6 8
Golden ratio= 1.6
7 13
    Golden ratio= 1.625
8 21
    Golden ratio= 1.6153846153846154
9 34
    Golden ratio= 1.619047619047619
10 55
    Golden ratio= 1.6176470588235294
11 89
    Golden ratio= 1.6181818181818182
12 144
    Golden ratio= 1.6179775280898876
13 233
    Golden ratio= 1.6180555555555556
14 377
    Golden ratio= 1.6180257510729614
15 610
    Golden ratio= 1.6180371352785146
16 987
    Golden ratio= 1.618032786885246
17 1597
    Golden ratio= 1.618034447821682
18 2584
    Golden ratio= 1.6180338134001253
19 4181
    Golden ratio= 1.618034055727554
20 6765
    Golden ratio= 1.6180339631667064
21 10946
    Golden ratio= 1.6180339985218033
22 17711
    Golden ratio= 1.618033985017358
```

```
23 28657
    Golden ratio= 1.6180339901755971
24 46368
    Golden ratio= 1.618033988205325
25 75025
    Golden ratio= 1.618033988957902
26 121393
    Golden ratio= 1.6180339886704431
27 196418
    Golden ratio= 1.6180339887802426
28 317811
    Golden ratio= 1.618033988738303
29 514229
    Golden ratio= 1.6180339887543225
30 832040
    Golden ratio= 1.6180339887482036
31 1346269
    Golden ratio= 1.6180339887505408
32 2178309
    Golden ratio= 1.6180339887496482
33 3524578
    Golden ratio= 1.618033988749989
34 5702887
    Golden ratio= 1.618033988749859
35 9227465
    Golden ratio= 1.6180339887499087
36 14930352
    Golden ratio= 1.6180339887498896
37 24157817
    Golden ratio= 1.618033988749897
38 39088169
    Golden ratio= 1.618033988749894
39 63245986
    Golden ratio= 1.6180339887498951
40 102334155
```

Thus we have found that the ratio of successive terms of a Fibonacci sequence a_{n+1}/a_n, which is equal to b_n/a_n, converges to the Golden Ratio.

Lupascu Felicia

Ex.3

the resistance R of the thermistor and the temperature T is given by

$$\frac{1}{T} = 1.129241 \times 10^{-3} + 2.341077 \times 10^{-4} \log R + 8.775468 \times *10^{-8} \Big(\log R \Big)^3$$

$$\frac{1}{19.01 + 273.15} = 1.129241 \times 10^{-3} + 2.341077 \times 10^{-4} \log R + 8.775468 \times *10^{-8} \Big(\log R \Big)^3 \quad (1)$$

$$\frac{1}{18.99 + 273.15} = 1.129241 \times 10^{-3} + 2.341077 \times 10^{-4} \log R + 8.775468 \times *10^{-8} \Big(\log R \Big)^3 \quad (2)$$

$$\frac{1}{18.99+273.15} = 1.129241 \cdot 10^{-3} + 2.341077 \cdot 10^{-4} \ln(R) + 8.775468 \cdot 10^{-8} \{\ln(R)\}^3$$

$f(R) = 2.341077 \cdot 10^{-4} \ln(R) = 0$, as $R_{-1} = 14000$, $R_0 = 15000$

Iteration 1

The estimate of the root is:

$$R_1 = R_0 - \frac{f(R_0)(R_0 - R_{-1})}{f(R_0) - f(R_{-1})}$$

$$f(R_0) = 2.341077 \cdot 10^{-4} \ln(R_0) + 8.775468 \cdot 10^{-8} \{\ln(R_0)\}^3 - 2.293775 \cdot 10^{-3} =$$
$$= 2.341077 \cdot 10^{-4} \ln(1500) + 8.775468 \cdot 10^{-8} (\ln(15000))^3 - 2.293775 \cdot 10^{-3} =$$
$$= \underline{3.5383 \cdot 10^{-5}}$$

$$f(R_{-1}) = 2.341077 \cdot 10^{-4} \ln(R_{-1}) + 8.775468 \cdot 10^{-8} \{\ln(R_{-1})\}^3 - 2.293775 \cdot 10^{-3} =$$
$$= 2.341077 \cdot 10^{-4} \ln(14000) + 8.775468 \cdot 10^{-8} \{\ln(14000)\}^3 - 2.293775 \cdot 10^{-3} =$$
$$= \underline{1.7563 \cdot 10^{-5}}$$

$$R_1 = 1500 - \frac{(3.5383 \cdot 10^{-5})(15000 - 14000)}{(3.5383 \cdot 10^{-5})(-1.7563 \cdot 10^{-5})} = \underline{13014}$$

The absolute relative approximate error $|\epsilon_a|$ at the end of iteration 1

$$|\epsilon_a| = \left| \frac{R_1 - R_0}{R_1} \right| \cdot 100\% = \left| \frac{13014 - 15000}{13014} \right| \cdot 100 = \underline{15.257\%}$$

The number of significant digits at least correct is zero, as we need an absolute relative approximate error of less than 5% for one significant digit to be correct in our result.

Lupascu Felicia

Iteration 2

The estimate of the root is

$$R_2 = R_1 - \frac{f(R_1)(R_1 - R_0)}{f(R_1) - f(R_0)}$$

$$f(R_1) = 2.341077 \cdot 10^{-4} \ln(R_1) + 8.775468 \cdot 10^{-5} \{\ln(R_1)\}^3 - 2.293775 \cdot 10^{-3} =$$
$$= 2.341077 \cdot 10^{-4} \ln(13014) \{\}^3 - 2.293775 \cdot 10^{-3} = \underline{13083}$$

The absolute relative approximate error $|\varepsilon_a|$ at the end of iteration 2

$$|\varepsilon_a| = \left|\frac{R_2 - R_1}{R_2}\right| \cdot 100 = \left|\frac{13083 - 13014}{13083}\right| \cdot 100\% = 0{,}52422\%$$

The number of significant digits at least correct is 1, because the absolute relative approximate error is less than 5%.

Iteration 3

The estimate of the root is

$$R_3 = R_2 - \frac{f(R_2)(R_2 - R_1)}{f(R_2) - f(R_1)}$$

$$f(R_2) = 2.341077 \cdot 10^{-4} \ln(R_2) + 8.775468 \cdot 10^{-8} \{\ln(R_2)\}^3 - 2.293775 \cdot 10^{-3} =$$
$$= 2.341077 \cdot 10^{-4} \ln(13083) + 8.775468 \cdot 10^{-8} \{\ln(13083)\}^3 - 2.293775 \cdot 10^{-3} =$$
$$= 8.8907 \cdot 10^{-8}$$

$$R_3 = 13083 - \frac{(8.8911 \cdot 10^{-8})(13083 - 13014)}{(8.8911 \cdot 10^{-8}) - (-1.2658 \cdot 10^{-6})} = \underline{13078}$$

The absolute relative approximate error $|\varepsilon_a|$ at the end of iteration 3

$$|\varepsilon_a| = \left|\frac{R_3 - R_2}{R_1}\right| \cdot 100 = \left|\frac{13078 - 13083}{13078}\right| \cdot 100\% = 0.034415\%$$

The number of significant digits at least correct is 3, because the absolute relative approximate error is less than 0.05%

```
clear all
clc
syms R
% write the equation to be solved
f = log(R)*2.341077*10^(-4)+ log(R)^3*8.775468*10^(-8)-2.293775*10^(-3);
% derivative of the function
g = diff(f);
n = 3; % number of decimal places
epsilon = 0.5*10^(2-n);
R0 = 15000; % initial guess
%%
for i=1:100
f0 = vpa(subs(f,R,R0)); %function value at R0
f0_der = vpa(subs(g,R,R0));%function derivative at R0
y = R0 - f0/f0_der; % update the next guess
err = abs(y-R0);
if err<epsilon
break
end
R0 = y;
end
%%
y = y - rem(y,10^-n)
%fprintf('The Root is : %f ',y);
```

Ex.3 Matlab

Lupascu Felicia

Ex.4

Order of convergence= 1.9617575539126844

It tends to 2 . It is quadratic.

c) $f(x) = e^{x-\pi} + \cos x - x + \pi$

$f'(x) = -\sin(x) + e^{x-\pi} - 1$

If the function has a multiplicity 2 or higher then it will converge slower than bisection, but if we take the derivative $f'(x)$ or higher, it will have the same solution as $f(x)$ and it will converge quadratically

Lupascu Felicia

```
function dummy=bisc_newton_secant()
clc;
clear all;

f=@(e^(x-pi)+cos(x)-x+pi); %function
fp=@(-sin(x)+e^(x-pi)-1); % derivative of unction

tol=1e-8;

a=0;
b=5; % interval
x0=5;
disp('Root by Bisection method')
y=bisecion(f,a,b,tol) % function calling

a1=0;
b1=5; % interval
disp('Root by fixed point method')
C1=fixedp(g,x0,tol) % function calling

disp('Root by Newton method')
y2=newt(f,x0,tol)% function calling

a2=0;
b2=5; % interval

disp('Root by Secant method')
y1=secn(f,a2,b2,tol)% function calling
```

Ex.5

a) $x_{n+1} = \cos x_n - 1 + x_n$

The formula gives $x_1 = \cos x_0 - 1 + x_0$

Plug in $x_0 = 0.1$ to find x_1.

$x_1 = \cos(0.1) - 1 + 0.1 = 0.999984769$

$x_2 = \cos(0.999984769) - 1 + 0.999984769 = 0.9998324688$

$x_3 = \cos(0.9998324688) - 1 + 0.9998324688 =$

...

$x = f(x)$

$x = \cos(x) - 1 + x$.

It gives 0 as a fixed point.

Lupascu Felicia

```cpp
// C++ program for implementation of Newton Raphson Method for
// solving equations
#include<bits/stdc++.h>
#define EPSILON 0.001
using namespace std;

// An example function whose solution is determined using
// Bisection Method. The function is xn+1 = cos xn – 1 + xn
double func(double x)
{
    return cos(x)-1+x;
}

// Derivative of the above function which is 1-sin(x)
double derivFunc(double x)
{
    return  1-sin(x);
}

// Function to find the root
void newtonRaphson(double x)
{
    double h = func(x) / derivFunc(x);
    while (abs(h) >= EPSILON)
    {
        h = func(x)/derivFunc(x);

        // x(i+1) = x(i) - f(x) / f'(x)
        x = x - h;
    }

    cout << "The value of the root is : " << x;
}

// Driver program to test above
int main()
{
    double x0 = -0.1; // Initial values assumed
    newtonRaphson(x0);
    return 0;
}
```

Ex.6

$$\lambda n = \frac{x_n - x_{n-1}}{x_{n-1} - x_{n-2}}$$

n	x_n	$x_n - x_{n-1}$	λn
0	2.0		
1	2.1248	0.124834	
2	2.2148	0.089944	0,720705
3	2.2805	0.065698	0.72997777
4	2.3289	0.048386	0.736468797
5	2.3647	0.035827	0,74022727272
6	2.3913	0.026624	0.74567669172
7	2.4111	0.019835	0.74567669172
8	2.4260	0.014803	0,747626262
9	2.4370	0.011062	0,74224161073
10	2.4453	0.0082745	0,752227

Lupascu Felicia

$$\lambda_{n_3} = \frac{0,065698}{2.2718 - 2.1248} = 0,7299777777$$

$$\lambda_{n_4} = \frac{0,048386}{2.2805 - 2.2718} = 0,7364687975 6$$

$$\lambda_{n_5} = \frac{0,035827}{2.3289 - 2.2805} = 0,7402272727 2$$

$$\lambda_{n_6} = \frac{0,026624}{2,3647 - 2.3289} = 0,7436871508 38$$

$$\lambda_{n_7} = \frac{0,019835}{2,3913 - 2.3647} = 0,7456766917 2$$

$$\lambda_{n_8} = \frac{0,014803}{2.4111 - 2.3913} = 0,7476262626 2$$

$$\lambda_9 = \frac{0,011062}{2.4260 - 2.4111} = 0,7422416107 38$$

$$\lambda_{10} = \frac{0,0082745}{2.4370 - 2.4260} = 0,752227$$

a)An iterative method is called convergent if the corresponding sequence converges for given initial approximations. Out function is convergent because it tends to the initial value 2.

b) Yes.

c) When the condition is satisfied, Newton's method converges, and it also converges faster than almost any other alternative iteration scheme based on other methods of coverting the original f(x) to a function with a fixed point.

Lupascu Felicia

Ex.7

Given function is $f(x) = x + \ln x = 0$

Graph of $f(x) = x + \ln(x)$

domain $(0, \infty)$
Range $(-\infty; +\infty)$

① i) $x = -\ln(x) = \varphi(x) \Rightarrow \varphi(x) = -\ln(x)$

$|\varphi'(x)| = |-\frac{1}{x}|$

Therefore $|\varphi'(x)| = |\frac{1}{x}| \not< 1$ in its domain.

$\Rightarrow x$ can not converge

③ iii) $x = \frac{x + e^{-x}}{2}$

$\varphi(x) = \frac{x + e^{-x}}{2}$; $|\varphi'(x)| = \frac{1}{2}|1 - e^{-x}| < 1$

$\forall x$ in the domain of f.

$\Rightarrow x = \frac{x + e^{-x}}{2}$ will converge.

② ii) $x = e^{-x} = \varphi(x) = \varphi(x)$

$\varphi'(x) = |-e^{-x}| = |e^{-x}| < 1$ $\forall x$ in the domain of f.

$\Rightarrow \varphi(x) = x = e^{-x}$ will converge definitely

② and ③ can be used to find the root of f.

A better formula to solve this equation by Newton-Raphson meth

Newton-Raphson method is given by:

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

We have $f(x) = x + \ln(x)$
$f'(x) = 1 + \frac{1}{x}$
$a_0 = 0.5$

Putting $n = 0$ in the above iterative formula we get

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 0.5 - \frac{0.5 + \ln(0.5)}{1 + \frac{1}{0.5}} = 0.5 - \frac{-0.1931478}{3}$$

i.e x_1 = first iteration $= 0.5643823 93$

Second iteration

$$x_3 = x_2 - \frac{f(x_2)}{f'(x_2)} = 0.567138987 - \frac{0.567138987 + \ln(0.567138987)}{1 + \frac{1}{0.567178987}} =$$

$$= 0.567145831$$

Relative error $= |x_3 - x_2| = 0.000006844543222$

$x_3 = 0.567145831$ is correct up to 5 digits.

x_3 is a root of the eq. $x + \ln(x) = 0$.

Lupascu Felicia

Ex.9

```python
# Python code for Julia Fractal
from PIL import Image

# driver function
if __name__ == "__main__":

    # setting the width, height and zoom
    # of the image to be created
    w, h, zoom = 1920,1080,1

    # creating the new image in RGB mode
    bitmap = Image.new("RGB", (w, h), "white")

    # Allocating the storage for the image and
    # loading the pixel data.
    pix = bitmap.load()

    # setting up the variables according to
    # the equation to create the fractal
    cX, cY = -0.7, 0.27015
    moveX, moveY = 0.0, 0.0
    maxIter = 255

    for x in range(w):
        for y in range(h):
            zx = 1.5*(x - w/2)/(0.5*zoom*w) + moveX
            zy = 1.0*(y - h/2)/(0.5*zoom*h) + moveY
            i = maxIter
            while zx*zx + zy*zy < 4 and i > 1:
                tmp = zx*zx - zy*zy + cX
                zy,zx = 2.0*zx*zy + cY, tmp
                i -= 1

            # convert byte to RGB (3 bytes), kinda
            # magic to get nice colors
            pix[x,y] = (i << 21) + (i << 10) + i*8

    # to display the created fractal
    bitmap.show()

```

Ex.8

n	x_n	$x_n - x_{n-1}$
0	1.00	
1	0.36788	$-6.3212E-01$
2	0.69220	$3.2432E-01$
3	0.50047	$-1.9173E-01$
4	0.60624	$1.0577E-01$
5	0.54540	$-6.0848E-02$
6	0.57961	$3.4217E-02$

8.

$$\lambda_2 = \frac{-6.3212E-01}{0,36788-1} =$$

$$\lambda_3 = \frac{-1,9173E-01}{0,69220-0,36788} =$$

$$\lambda_4 = \frac{1,0577E-01}{0,50047-0,69220} = 1,0577e+\frac{100000}{19173}$$

$$\lambda_5 = \frac{-6,0848E-02}{0,60624-0,50042} = -6,048e-\frac{100000}{5291}$$

$$\lambda_6 = \frac{3,4217E-02}{0,54540-0,600624} \approx 3,4217e+\frac{250000}{6903}$$

$f'(R)=0$, so the sequence converges linearly to the fixed point.

We can also use Aitken extrapolation formula

Lupascu Felicia FAF212

Homework 1

Exercise 1.2

Number 12.1875

Value actually stored in float 12.1875

Binary representation 01000001010000110000000000000000

Hexadecimal representation 0x41430000

	Sign	Exponent	Mantissa
Value	+1	2^3	1.5234375
Encoded as	0	130	43090912

Exercise 1.3

Exponent: 7 bits
Sign 0 : 1 bit
Mantissa : 16 bits with no heading of the leading bit 1
The arithmetic used chopping

The single ~~form~~ precision IEEE format of x consists of a precision of 24 binary digits.

The exponent e is limited by $-126 < e < 127$

Exercise 1.5

Relative error $= \frac{x - fe(x)}{x}$

a) $x_A = 6435.4012$
$x_T = 6435.401163$

Relative error $= 0,000000575\% = 5,749447291467 \cdot e^{-9}$
Absolute error $= 0,000037$
Error $= 0,000037$
Significant digits $x_A \approx x_T$: 2 (from error)
The significant digits are 3,7

b) $x_A = 0,007245$ $x_T = 0,00723816$
Relative error $= 9,440893788819876e^{-4}$
Error $= 0,00000684$
Significant digits $x_A \approx x_T = 3$ (error)
The significant figures are 6,8,4 for error

c) $x_A = \frac{355}{113}$ $x_T = \pi$
$=$
3.141592920353982 3.141592653589793
Relative error $= 8,491367142816902e^{-8}$
Error $= 2,667641893008850e^{-7} = 0,000000266764189$
Significant digits $x_A \approx x_T$: 9

d) $x_A = 2.236$ $x_T = \sqrt{5} = 2,236067977$
Relative Error: $3.040138631023990e^{-5}$
Error : $6.797749928969641e^{-5}$
Significant digits $x_A \approx x_T =$

x_A has m significant digits with respect to x_T if the magnitude of error (x_A) is ≤ 5 units in the $(m+1)^{th}$ digit, beginning with the first nonzero digit in x_T

Exercise 14.

3 digits in mantissa: 1.00 1.01. 1.10 1.11

We should always have 1 as a leading bit

Possibilities for e $-3, -2, -1, 0, 1, 2, 3$

		e						
		-3	-2	-1	0	1	2	3
x	$(1.00)_2$	$(0.125)_{10}$	$(0.25)_{10}$	$(0.5)_{10}$	$(1)_{10}$	$(2)_{10}$	$(4)_{10}$	$(8)_{10}$
	$(1.01)_2$	$(0.15625)_{10}$	$(0.3125)_{10}$	$(0.625)_{10}$	$(1.25)_{10}$	$(2.5)_{10}$	$(5)_{10}$	$(10)_{10}$
	$(1.10)_2$	$(0.1875)_{10}$	$(0.375)_{10}$	$(0.75)_{10}$	$(1.5)_{10}$	$(3)_{10}$	$(6)_{10}$	$(12)_{10}$
	$(1.11)_2$	$(0.21875)_{10}$	$(0.4375)_{10}$	$(0.875)_{10}$	$(1.75)_{10}$	$(3.5)_{10}$	$7(10)$	$(14)_{10}$

c) $\frac{\pi}{3} = 1{,}0471975512 \approx 1{,}047$

$\frac{12}{7} = 1{,}71428571429 \approx 1{,}714$

Lupascu Felicia FAF212

Exercise 1.6

Avoid loss of significance errors in formulas

a) $\log(x) - \log(x-1)$ for large values of x

- We can avoid the loss of precision by multyplying the function by conjugate

$$\frac{\log(x) - \log(x-1)}{1} \cdot \frac{\log(x) + \log(x-1)}{\log(x) + \log(x-1)} =$$

$$= \frac{(\log(x) - \log(x-1))(\log(x) + \log(x-1))}{\log(x) + \log(x-1)}.$$

$f(100) =$ 0,0043648054
in first case

$f(100) =$ 0,00436480540245 0004
in second case

b) $\frac{e^x - 1}{x}$ We have to multply by the conjugate!

Other similar errors are present in calculating other coefficients, and thus they cause a major error in the final answer being calculated

c) $\cos(x+\alpha) - \cos(\alpha)$ for small values

We have to substitute the expression with something simmilar

$$\cos(\alpha) = 1 - 2\sin^2\left(\frac{\alpha}{2}\right).$$

Exercise 7.

Assume $|x_T - x_A| < 00\ldots05$

10's depend on number of the digits of x_A)

$$f(x_T) - f(x_A) = f'(\xi)(x_T - x_A)$$
$$\approx f'(x_T)(x_T - x_A)$$
$$\approx f'(x_A)(x_T - x_A)$$

a) $\sin(0.521)$

Maxeim size of rounding error 0.0005

$x_T - x_A = 0{,}0005$

$f'(x_A) \cdot 0{,}0005 \approx f(x_T) - f(x_A)$

b) $x_T \in (e^{\wedge 3.215} \cdot e^{\wedge 3.225})$

c) $\sqrt{0.0011}$ Maxeim size of error is 0,00005

d) $\arcsin(0{,}5)$

Maxeim size of rounding error is 0,05

Homework 3

Ex.3.1

x	2.0	4.5	5.25	7.81	9.2	10.6
y	7.2	7.1	6.0	5.0	3.5	5.0

For fifth polynomial interpolation we choose the value of y given by

$$y(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5$$

Using the six points.

$x_0 = 2.00$ $\quad y(x_0) = 7.2$
$x_1 = 4.25$ $\quad y(x_1) = 7.1$
$x_2 = 5.25$ $\quad y(x_2) = 6.0$
$x_3 = 7.81$ $\quad y(x_3) = 5.0$
$x_4 = 9.20$ $\quad y(x_4) = 3.5$
$x_5 = 10.60$ $\quad y(x_5) = 5.0$

gives:

$$y(2.00) = a_0 + a_1(2.00) + a_2(2.00)^2 + a_3(2.00)^3 + a_4(2.00)^4 + a_5(2.00)^5 = 7.2$$
$$y(4.25) = a_0 + a_1(4.25) + a_2(4.25)^2 + a_3(4.25)^3 + a_4(4.25)^4 + a_5(4.25)^5 = 7.1$$
$$y(5.25) = a_0 + a_1(5.25) + a_2(5.25)^2 + a_3(5.25)^3 + a_4(5.25)^4 + a_5(5.25)^5 = 6.0$$
$$y(7.81) = a_0 + a_1(7.81) + a_2(7.81)^2 + a_3(7.81)^3 + a_4(7.81)^4 + a_5(7.81)^5 = 5.0$$
$$y(9.20) = a_0 + a_1(9.20) + a_2(9.20)^2 + a_3(9.20)^2 + a_4(9.20)^4 + a_5(9.20)^5 = 3.5$$
$$y(10.60) = a_0 + a_1(10.60) + a_2(10.60)^2 + a_3(10.60)^3 + a_4(10.60)^4 + a_5(10.60)^5 = 5.0$$

Writing the six equations in matrix form:

$$\begin{bmatrix} 1 & 2.00 & 4 & 8 & 16 & 32 \\ 1 & 4.25 & 18.063 & 76.766 & 326.25 & 1386.6 \\ 1 & 5.25 & 27.563 & 144.70 & 759.69 & 3988.4 \\ 1 & 7.81 & 60.996 & 476.38 & 3720.5 & 29057 \\ 1 & 9.20 & 84.64 & 778.69 & 7163.9 & 65908 \\ 1 & 10.60 & 112.36 & 1191.0 & 12625 & 133820 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} = \begin{bmatrix} 7.2 \\ 7.1 \\ 6.0 \\ 5.0 \\ 3.5 \\ 5.0 \end{bmatrix}$$

Solving the above six equations gives

$a_0 = -30.898$
$a_1 = 41.344$
$a_2 = -15.855$
$a_3 = 2.7862$
$a_4 = -0.23091$
$a_5 = 0.0072923$

Hence

$$y(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5$$

$$= -30.898 + 41.344x - 15.855x^2 + 2.7862x^3 - 0.23091x^4 +$$

$$+ 0.0072923x^5 , \qquad 2 \le x \le 16$$

Figure 1 Location of holes on the rectangular plate.

Cubic Spline formula is

$$f(x) = \frac{(x_i - x)^3}{6h} M_{i-1} + \frac{(x - x_{i-1})^3}{6h} M_i + \frac{(x_i - x)}{h}\left(y_{i-1} - \frac{h^2}{6} M_{i-1}\right) + \frac{(x - x_{i-1})}{h}\cdot\left(y_i - \frac{h^2}{6} M_i\right)$$

We have $M_{i-1} + 4M_i + M_{i+1} = \frac{6}{h^2}(y_{i-1} - 2y_i - y_{i+1})$

Here $h = 2.5$ $n = 5$ $M_0 = 0$ $M_5 = 0$

Substitute $i = 1$ in equation

$$M_0 + 4M_1 + M_2 = \frac{6}{h^2}(y_0 - 2y_1 + y_2) \Rightarrow 4M_1 + M_2 = -0.96$$

$$M_1 + 4M_2 + M_3 = \frac{6}{h^2}(y_1 - 2y_2 + y_3) \Rightarrow M_1 + 4M_2 + M_3 = 0.096$$

$$M_2 + 4M_3 + M_4 = \frac{6}{h^2}(y_2 - 2y_3 + y_4) \Rightarrow M_2 + 4M_3 + M_4 = -0.48$$

$$M_3 + 4M_4 + M_5 = \frac{6}{h^2}(y_3 - 2y_4 + y_5) \Rightarrow M_3 + 4M_4 = 2.88$$

Substitute $i = 1$ in eq, we get cubic spline 1st interval $[2.4 - 5]$

$f_1(x) = -0.0191x^3 + 0.1148x^2 - 0.1501x + 7.1939$ for $2.0 \le x \le 4.5$

$f_2(x) = 0.0317x^3 - 0.4709x^2 + 1.7072x + 1.776$ for $4.5 \le x \le 5.25$

$f_3(x) = -0.0372x^3 + 0.6828x^2 - 4.5051x + 16.3729$ for $5.25 \le x \le 7.81$

$f_4(x) = 0.0789x^3 - 1.9504x^2 + 15.087x - 33.4957$ for $7.81 \le x \le 9.2$

$f_5(x) = -0.0542x^3 + 1.7777x^2 - 17.3222x + 57.3723$ for $9.2 \le x \le 10.6$

Ex.3.2

a)

n	1	2	3	4	5
$\Gamma(n)$	1	1	2	6	24

```
import numpy as np
import matplotlib.pyplot as plt

plt.style.use('seaborn-poster')

%matplotlib inline
def divided_diff(x, y):
    '''
    function to calculate the divided
    differences table
    '''
    n = len(y)
    coef = np.zeros([n, n])
    # the first column is y
    coef[:,0] = y

    for j in range(1,n):
        for i in range(n-j):
            coef[i][j] = \
           (coef[i+1][j-1] - coef[i][j-1]) / (x[i+j]-x[i])

    return coef

def newton_poly(coef, x_data, x):
    '''
    evaluate the newton polynomial
    at x
    '''
    n = len(x_data) - 1
    p = coef[n]
    for k in range(1,n+1):
        p = coef[n-k] + (x -x_data[n-k])*p
    return p
x = np.array([1,2,3,4,5])
y = np.array([1,1,2,6,24])
# get the divided difference coef
a_s = divided_diff(x, y)[0, :]

# evaluate on new data points
#x_new = np.arange(1, 2,3,4,5)
#Y_new = newton_poly(a_s, x, x_new)

plt.figure(figsize = (12, 8))
plt.plot(x, y, 'bo')
#plt.plot(x_new, y_new)
```

$$
\begin{array}{llllll}
x_0 & y_0 & & & & \\
 & & f[x_1,x_0] & & & \\
x_1 & y_1 & & f[x_2,x_1,x_0] & & \\
 & & f[x_2,x_1] & & f[x_3,x_2,x_1,x_0] & \\
x_2 & y_2 & & f[x_3,x_2,x_1] & & f[x_4,x_3,x_2,x_1,x_0] \\
 & & f[x_3,x_2] & & f[x_4,x_3,x_2,x_1] & \\
x_3 & y_3 & & f[x_4,x_3,x_2] & & \\
 & & f[x_4,x_3] & & & \\
x_4 & y_4 & & & &
\end{array}
$$

x	y	1st order	2nd order	3rd order	4th order
1	1				
		0			
2	1		0.5		
		1		0.333333	
3	2		1.5		-0.458333
		4		-1.5	
4	6		-3		
		-2			
5	4				

Newton's divided difference interpolation formula

$$f(x) = y_0 + (x - x_0)f[x_0, x_1] + (x - x_0)(x - x_1)f[x_0, x_1, x_2] + (x - x_0)(x - x_1)(x - x_2)f[x_0, x_1, x_2, x_3] + (x - x_0)(x - x_1)(x - x_2)(x - x_3)f[x_0, x_1, x_2, x_3, x_4]$$

$$f(x) = 1 + (x - 1) \times 0 + (x - 1)(x - 2) \times 0.5 + (x - 1)(x - 2)(x - 3) \times 0.333333 + (x - 1)(x - 2)(x - 3)(x - 4) \times -0.458333$$

$$f(x) = 1 + (x - 1) \times 0 + \left(x^2 - 3x + 2\right) \times 0.5 + \left(x^3 - 6x^2 + 11x - 6\right) \times 0.333333 + \left(x^4 - 10x^3 + 35x^2 - 50x + 24\right) \times -0.458333$$

$$f(x) = 1 + (0) + \left(0.5x^2 - 1.5x + 1\right) + \left(0.333333x^3 - 2x^2 + 3.666667x - 2\right) + \left(-0.458333x^4 + 4.583333x^3 - 16.041667x^2 + 22.916667x - 11\right)$$

$$f(x) = -0.458333x^4 + 4.916667x^3 - 17.541667x^2 + 25.083333x - 11$$

$$f(x) = \frac{(x_i - x)^3}{6h} M_{i-1} + \frac{(x - x_{i-1})^3}{6h} M_i + \frac{(x_i - x)}{h}\left(y_{i-1} - \frac{h^2}{6} M_{i-1}\right) + \frac{(x - x_{i-1})}{h}\left(y_i - \frac{h^2}{6} M_i\right)$$

b)

We have $M_{i-1} + 4M_i + M_{i+1} = \frac{6}{h^2}(y_{i-1} - 2y_i + y_{i+1})$

$h=1 \quad h=4 \quad M_0=0 \quad M_4=0$

$M_0 + 4M_1 + M_2 = \frac{6}{h^2}(y_0 - 2y_1 + y_2). \Rightarrow 4M_1 + M_2 = 6$

$M_1 + 4M_2 + M_3 = \frac{6}{h^2}(y_1 - 2y_2 + y_3) \Rightarrow M_1 + 4M_2 + M_3 = 18$

$M_2 + 4M_3 + M_4 = \frac{6}{h^2}(y_2 - 2y_3 + y_4) \Rightarrow M_2 + 4M_3 = 84$

$f_1(x) = 0.3036x^3 - 0.9107x^2 + 0.6071x + 1$ for $1 \le x \le 2$

$f_2(x) = -0.5178x^3 + 4.0178x^2 - 9.2498x + 7.5713$ for $2 \le x \le 3$

$f_3(x) = 3.7678x^3 - 34.5535x^2 + 106.4641x - 108.1426$ for $3 \le x \le 4$

$f_4(x) = -3.5536x^3 + 53.3035x^2 - 244.9639x + 360.428$ for $4 \le x \le 5$

c)

n	1	2	3	4	5
$\log\Gamma(n)$	0	0	$\log 2$	$\log 6$	$\log 24$

c) $\frac{1}{24}x^4\log 24 - \frac{1}{6}x^4\log 6 + \frac{1}{4}\log 2 - \frac{5}{12}x^3\log 24 + \frac{11}{6}x^3\log 6 -$

$- 3x^3\log 2 + \frac{35}{24}x^2\log(24) - \frac{41}{6}x^2\log 6 + \frac{49}{4}x^2\log 2 - \frac{25}{12}x\log 24 +$

$\frac{61}{6}x\log 6 - \frac{39}{2}x\log 2 + \log 24 - 5\log 6 + 10\log 2.$

$$Q_4(x) = 0.0071x^4 - 0.1187x^3 + 0.8820x^2 - 1.9211x + 1.1507$$

$$g(x) = e^{Q(x)}$$

Ex.3.3

$$f(x) = \sqrt{x+1}$$

interval $[-1, 1]$

```
>> pkg load symbolic
>> syms n x
```

```
>> C=chebyshevU(8,x)
C = (sym)

       8         6         4        2
  256*x  - 448*x  + 240*x  - 40*x  + 1
```

The function to approximate is $f(x) = \sqrt{1+x}$ and the approx polynomial is $p(x) = \sum_{i=0}^{d} p_i x^i$ of degree $d \geq 1$

Required that $p(0) = f(0) = 1$ and
$p(1) = f(1) = \sqrt{2}$

So we need to compute coefficients p_i that $1 \leq i \leq d-1$

An initial guess for $p(x)$ is $p(i/d) = f(i/d)$ for $1 \leq i \leq d-1$ in which case $g(x) = f(x) - p(x)$ is oscillatory.

For degree 7, the conditions $p(0) = f(0)$
$p(1/7) = f(1/7)$
$\vdots$
$p(1) = f(1)$

d	coefficients			d	coefficients		
1	p_0	=	$+1$	2	p_0	=	$+1$
	p_1	=	$+4.1421356237309505 * 10^{-1}$		p_1	=	$+4.8563183076125260 * 10^{-1}$
	e	=	$+1.7766952966368793 * 10^{-2}$		p_2	=	$-7.1418268388157458 * 10^{-2}$
					e	=	$+1.1795695163108744 * 10^{-3}$
3	p_0	=	$+1$	4	p_0	=	$+1$
	p_1	=	$+4.9750045320242231 * 10^{-1}$		p_1	=	$+4.9955939832918816 * 10^{-1}$
	p_2	=	$-1.0787308044477850 * 10^{-1}$		p_2	=	$-1.2024066151943025 * 10^{-1}$
	p_3	=	$+2.4586189615451115 * 10^{-2}$		p_3	=	$+4.5461507257698486 * 10^{-2}$
	e	=	$+1.1309620116468910 * 10^{-4}$		p_4	=	$-1.0566681694362146 * 10^{-2}$
					e	=	$+1.2741170151556180 * 10^{-5}$
5	p_0	=	$+1$	6	p_0	=	$+1$
	p_1	=	$+4.9992197660031912 * 10^{-1}$		p_1	=	$+4.9998616695784914 * 10^{-1}$
	p_2	=	$-1.2378506719245053 * 10^{-1}$		p_2	=	$-1.2470733323278438 * 10^{-1}$
	p_3	=	$+5.6122776972699739 * 10^{-2}$		p_3	=	$+6.0388587356982271 * 10^{-2}$
	p_4	=	$-2.3128836281145482 * 10^{-2}$		p_4	=	$-3.1692053551807930 * 10^{-2}$
	p_5	=	$+5.0827122737047148 * 10^{-3}$		p_5	=	$+1.2856590305148075 * 10^{-2}$
	e	=	$+1.5725568940708201 * 10^{-6}$		p_6	=	$-2.6183954624343642 * 10^{-3}$
					e	=	$+2.0584155535630089 * 10^{-7}$
7	p_0	=	$+1$	8	p_0	=	$+1$
	p_1	=	$+4.9999754817809228 * 10^{-1}$		p_1	=	$+4.9999956583056759 * 10^{-1}$
	p_2	=	$-1.2493243476353655 * 10^{-1}$		p_2	=	$-1.2498490369914350 * 10^{-1}$
	p_3	=	$+6.1859954146370910 * 10^{-2}$		p_3	=	$+6.2318494667579216 * 10^{-2}$
	p_4	=	$-3.6091595023208356 * 10^{-2}$		p_4	=	$-3.7982961896432244 * 10^{-2}$
	p_5	=	$+1.9483946523450868 * 10^{-2}$		p_5	=	$+2.3642612312869460 * 10^{-2}$
	p_6	=	$-7.5166134568007692 * 10^{-3}$		p_6	=	$-1.2529377587270574 * 10^{-2}$
	p_7	=	$+1.4127567687864939 * 10^{-3}$		p_7	=	$+4.5382426960713929 * 10^{-3}$
	e	=	$+2.8072302919734948 * 10^{-8}$		p_8	=	$-7.8810995273670414 * 10^{-4}$
					e	=	$+3.9460605685825989 * 10^{-9}$

The numbers are coefficient pi for the polynomyal p(x), The table shows the maximum error for the approximation.

```
% construct N=8 data points
N=8;
xdata=linspace(-1,1,N);
```

```
ydata=sqrt(1+x)(xdata);

% construct many test points
xval=linspace(???,???,4001);
% construct the true test point values, for reference
yvalTrue=sinh(???);

% use Lagrange polynomial interpolation to evaluate
% the interpolant at the test points
yval=eval_lag(???,???,xval);

% plot reference values in thick green
plot(xval,yvalTrue,'g','linewidth',4);
hold on
% plot interpolation data points
plot(xdata,ydata,'k+');
% plot interpolant in thin black
plot(xval,yval,'k');
hold off

% estimate the approximation error of the interpolant
approximationError=max(abs(yvalTrue-yval))/max(abs(yvalTrue))
```

Figure 1 sqrt(x+1)

Homework 1

Due February 11, 19.00

Problem 1.1

All readings are posted on the course web page.

a) Read Lectures 0, 1.1, 1.2, 1.3.

b) Review how to do binary to decimal and vice-versa conversions.

c) Read definitions of Numerical Analysis by K. Atkinson and L. Trethenen.

d) Read the history of IEEE standard 754.

e) Read the paper on some common bugs related to computer representation of numbers.

Problem 1.2

Write the binary single precision IEEE floating-point expression for the number 12.1875. Specify sign σ, exponent E and mantissa.

Problem 1.3

Some microcomputers in the past used a binary floating-point format with 7 bits for the exponent and 1 bit for the sign σ. The mantissa contained 16 bits, with no hiding of the leading bit 1. The arithmetic used chopping. Determine the accuracy of the representation by finding the following:

a) machine epsilon;

b) integer M;

c) accuracy of the chopping operation.

Problem 1.4

Consider a binary floating-point representation with mantissa containing 3 digits without hiding the leading 1 and $-3_{10} \leqslant e \leqslant 3_{10}$.

a) List all numbers that can be stored exactly together with their decimal value.

b) Plot these numbers on real axis.

c) For this arithmetic, specify what are the corresponding floating-point representation of $\pi/3$ and $12/7$ if rounding is used.

d) Repeat c) if chopping is being used.

Problem 1.5

Calculate the error, relative error and the number of significant digits in the following approximations $x_A \approx x_T$.

a) $x_A = 6435.4012, \quad x_T = 6435.401163;$

b) $x_A = 0.007245, \quad x_T = 0.00723816;$

c) $x_A = 355/113, \quad x_T = \pi;$

a) $x_A = 2.236, \quad x_T = \sqrt{5}.$

Problem 1.6

Avoid loss-of-significance errors in the following formulas

$$
\begin{aligned}
&a) \quad \log(x) - \log(x-1) \quad \text{for large values of } x;\\
&b) \quad \frac{e^x - 1}{x} \quad \text{for small values of } x;\\
&c) \quad \cos(x+a) - \cos(a) \quad \text{for small values of } x;
\end{aligned}
$$

Problem 1.7

In the following function evaluations $f(x_A)$, assume the numbers x_A are correctly rounded to the number of digits shown. Bound the error $f(x_T) - f(x_A)$ and the relative error $Rel(x_A)$:

$$
\begin{aligned}
&a) \quad \sin(0.521);\\
&b) \quad e^{3.22};\\
&c) \quad \sqrt{0.0011};\\
&d) \quad \arcsin(0.5).
\end{aligned}
$$

Homework 2

Due March 18, 19:00

Problem 2.1

The **error function** (also called Gauss error function) is defined by

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\, dt.$$

Use Taylor series to approximate $\operatorname{erf}(x)$ with a polynomial $T_n(x)$. What is n, if the desired accuracy is 10^{-5}? Using this approximation, plot the graph of $\operatorname{erf}(x)$ on $[-3, 3]$.

Problem 2.2

Consider the sequence of Fibonacci numbers F_n:

$$F_0 = 1, \quad F_1 = 1, \quad F_n = F_{n-1} + F_{n-2}, \quad n = 2, 3, \ldots$$

Let $R_n = \frac{F_{n+1}}{F_n}$. It can be shown that

$$\lim_{n\to\infty} R_n = \frac{1+\sqrt{5}}{2} \equiv \phi,$$

which is known as **golden ratio**. Write a code that will compute numerically the first 40 terms of the sequence R_n together with errors $\phi - R_n$. In computations make sure that you are using IEEE double precision. Comment your results. What can be said on the order of convergence?

Problem 2.3

Thermistors are temperature-measuring devices based on the principle that the thermistor material exhibits a change in electrical resistance with a change in temperature. By measuring the resistance of the thermistor material, one can then determine the temperature. For a 10K3A Betatherm thermistor, the relationship between

Figure 1: A typical thermistor

the resistance R of the thermistor and the temperature T is given by

$$\frac{1}{T} = 1.129241 \times 10^{-3} + 2.341077 \times 10^{-4} \log R + 8.775468 \times *10^{-8} \Big(\log R\Big)^3$$

where T is in Kelvin and R is in Ohms, and log denotes the natural logarithm. A thermistor error of no more than $\pm 0.01^oC$ is acceptable. To find the range of the resistance that is within this acceptable limit at 19^oC, we need to solve

$$\frac{1}{19.01 + 273.15} = 1.129241 \times 10^{-3} + 2.341077 \times 10^{-4} \log R + 8.775468 \times *10^{-8} \Big(\log R\Big)^3 \tag{1}$$

$$\frac{1}{18.99 + 273.15} = 1.129241 \times 10^{-3} + 2.341077 \times 10^{-4} \log R + 8.775468 \times *10^{-8} \Big(\log R\Big)^3 \tag{2}$$

Write a computer routine implementing Newton's method and solve equations (1) and (2) using Newton's method with initial guess $R_0 = 15000$ and error tolerance of 10^{-5}.
What is the obtained range for resistance values?

Problem 2.4

Consider the function $f(x) = e^{x-\pi} + \cos x - x + \pi$.

a) Plot its graph on interval $[0, 5]$.

b) Apply Newton's method routine you developed in **Problem 2.3** to solve equation $f(x) = 0$ on interval $[0, 5]$. What can be said about its order of convergence? Argue why this is happening? How its convergence order can be improved?

c) Write a modified routine that will ensure **quadratic** convergence and apply it.

d) Instead of solving $f(x) = 0$, try to apply the fixed point iterations $x_{n+1} = e^{x_n-\pi} + \cos x_n + \pi$. Comment on your results.

Problem 2.5

a) Compute the fixed point $x_{n+1} = \cos x_n - 1 + x_n$ with initial guess $x_0 = 0.1$.

b) What can be said about the speed of convergence? Compare it with bisection method.

c) Write a modified computer routine that will speed up the convergence.

Problem 2.6

Newton's method is used to find the root α of $f(x) = 0$. The first 10 iterates are shown in the table below.

(1) What can be said about the order of convergence? Is it slower or faster than bisection method?
(2) What can be said about the root α to explain this convergence?
(3) Knowing function $f(x)$, how would you speed up the convergence?

n	x_n	$x_n - x_{n-1}$
0	2.0	
1	2.1248	0.124834
2	2.2148	0.089944
3	2.2805	0.065698
4	2.3289	0.048386
5	2.3647	0.035827
6	2.3913	0.026624
7	2.4111	0.019835
8	2.4260	0.014803
9	2.4370	0.011062
10	2.4453	0.0082745

Problem 2.7

For solving the equation $x + \ln x = 0$, there were proposed three methods:

$$\begin{aligned} (a) \quad x &= -\ln x \\ (b) \quad x &= e^{-x} \\ (c) \quad x &= \frac{x + e^{-x}}{2} \end{aligned}$$

(1) Which of the formulas can be used?
(2) Which of the formulas should be used?
(3) Give an even better formula!

Problem 2.8

Consider the following table of iterates from an iteration method which is convergent to a fixed point α of the function $g(x)$:

n	x_n	$x_n - x_{n-1}$
0	1.00	
1	0.36788	$-6.3212E-01$
2	0.69220	$3.2432E-01$
3	0.50047	$-1.9173E-01$
4	0.60624	$1.0577E-01$
5	0.54540	$-6.0848E-02$
6	0.57961	$3.4217E-02$

(1) Show that this is a linearly convergent iteration method.
(2) Find its rate of linear convergence. Is this method faster or slower than bisection method?
(3) Propose a way to accelerate the convergence of this method?

Problem 2.9

BONUS. Benout B. Mandelbrot, a famous mathematician is known as the inventor of *fractals* and this problem is dedicated to him. In this problem you will generate the so-called **quadratic Julia Sets**, a well-known fractal example.

Given two complex numbers, c and z_0 the following recursion (it is similar to fixed point iterations) is defined

$$z_n = z_{n-1}^2 + c.$$

For an arbitrary given choice of c and z_0, this recursion leads to a sequence of complex numbers z_1, z_2, $z_3, \ldots$ called the **orbit** of z_0. Depending on the exact choice of c and z_0, a large range of orbit patterns are possible.

For a given fixed c, most choices of z_0 yield orbits that tend towards infinity. (That is, $|z_n| \to \infty$ as $n \to \infty$).

For some values of c certain choices of z_0 yield orbits that eventually go into a periodic loop. Finally, some starting values yield orbits that appear to dance around the complex plane, apparently at random (an example of chaos). These initial values of z_0 make up the Julia set of this recursion, denoted by J_c.

Write a MATLAB/GNU Octave/Python script that visualizes a slightly different set, called the filled-in Julia set denoted by K_c, which is the set of all z_0 with orbits which do not tend towards infinity. The "normal" Julia set J_c is the edge of the filled-in Julia set. The figure below illustrates a filled-in Julia Set for one particular value of c.

a) It has been shown that if $|z_n| > 2$ for some n, then it is guaranteed that the orbit will tend to infinity. The value of n for which this becomes true is called the **escape velocity** of a particular z_0. Write a function that returns the escape velocity of a given z_0 and c. The function declaration should be: $n =$`EscVel`(z_0, c, N), where N is the maximum allowed escape velocity (i.e. if $|z_n| \leq 2$ for $n < N$, return N as the escape velocity, so you will prevent infinite loops).

b) To generate the filled-in Julia Set, write the following function

$$M = \texttt{JuliaSet}(z_{Max}, c, N),$$

where z_{Max} will be the maximum of the real and imaginary parts of the various values of z_0 for which we will compute escape velocities, c and N are the same as defined above, and M is the matrix that contains the escape velocity of various z_0.

- In this function, you first want to make a 500×500 matrix that contains complex numbers with real part between $-z_{Max}$ and z_{Max}, and imaginary part between $-z_{Max}$ and z_{Max}. Call this matrix Z. Make the imaginary part vary along the y-axis of this matrix. You can most easily do this by using `linspace` and `meshgrid` commands from MATLAB/GNU Octave, but you can also do it with a loop.
- For each element of Z, compute the escape velocity (by calling your `EscVel` function) and store it in the same location in a matrix M. When done, the matrix M should be the same size as Z and contain escape velocities with values between 1 and N.
- Run your `JuliaSet` function with various z_{Max}, c and N values to generate various fractals. To display the fractal nicely, use `imagesc` command to visualize $\arctan(0.1 * M)$, (taking the arctangent of M makes the image look nicer, you also can use `axisxy` command so that y values aren't flipped).

The figure below shows a Julia Set for $c = -.297491 + i * 0.641051$.

Figure 2: $M =$`JuliaSet`$(1, -.297491 + i * 0.641051, 100)$

Julia Sets are the boundaries of more general Mandelbrot sets. There is a chapter in Clive Moler textbook "Experiments with MATLAB" `www.mathworks.com/moler/exm/chapters.html` dedicated to Mandelbrot sets. The difference between Julia Sets and Mandelbrot set is presented in `www.karlsims.com/julia.html`, the figure below was taken from there.

Figure 3: Different Julia Sets and their relation to Mandelbrot Set.

Homework 3

Due April 1, 19:00

Problem 3.1

A robot arm with a rapid laser is used to do a quick quality check, such as the radius of hole, on six holes located on a rectangular plate. Coordinates of the centers of the holes are given below

x	2.0	4.5	5.25	7.81	9.2	10.6
y	7.2	7.1	6.0	5.0	3.5	5.0

The path of the robot going from one point to another point needs to be smooth so as to avoid sharp jerks in the arm that can otherwise create premature wear and tear of the robot arm.

a) One such path can be the interpolating polynomial $P_5(x)$ that passes those points. Find it.

b) Other paths can be obtained by cubic spline interpolation. Find several cubic splines (see Answers to homework 3) interpolating these data, plot them together with $P_5(x)$ and propose the one with the shortest path.

Problem 3.2

The gamma function is defined as

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t}\, dt$$

and it can be shown (integrate by parts) that $\Gamma(n) = (n-1)!\ \forall n \in \mathbb{N}$.
a) Write a computer programm that will calculate Newton's divided differences and then find (using Newton's formula) the interpolating polynomial $P_4(x)$ of the Γ function at the following points:

n	1	2	3	4	5
$\Gamma(n)$	1	1	2	6	24

b) Find a natural cubic spline $S(x)$ (using an available spline routine) that interpolates the same data.

c) Another approximation can be obtained by first calculating the polynomial $Q_4(x)$ that interpolates points $(n, \log\Gamma(n))$:

n	1	2	3	4	5
$\log\Gamma(n)$	0	0	$\log 2$	$\log 6$	$\log 24$

(here log denotes the natural logarithm) and then, consider $q(x) = e^{Q_4(x)}$.

d) Plot in the same figure the graphs of $\Gamma(x)$, $P_4(x)$, $S(x)$ and $q(x)$ on $[1, 5]$.

e) Compute with accuracy of at least 3 significant digits

$$\max_{x\in[1,5]} \Big|\Gamma(x) - P_4(x)\Big|, \quad \max_{x\in[1,5]} \Big|\Gamma(x) - S(x)\Big| \text{and} \quad \max_{x\in[1,5]} \Big|\Gamma(x) - q(x)\Big|.$$

Which of the 3 approximations is more accurate on $[1, 5]$?

HINTS. In MATLAB or GNU Octave the Γ function is known as `gamma`. In Python it is available in the `scipy.special` library. In order to complete d) you can compute the maximum from the graph, just make sure that sampling is done at sufficiently dense points that will ensure the needed accuracy.

Problem 3.3

Consider function $f(x) = \sqrt{x+1}$ on interval $[-1, 1]$. Find the near minimax polynomial approximation of degree 7 m_7 for this function. Also, find the polynomial interpolant $P_7(x)$ on evenly spaced points. Plot together function $f(x), m_7(x)$ and $P_7(x)$.

Homework 3

Due April 1, 19:00

Problem 3.1

A robot arm with a rapid laser is used to do a quick quality check, such as the radius of hole, on six holes located on a rectangular plate. Coordinates of the centers of the holes are given below

x	2.0	4.5	5.25	7.81	9.2	10.6
y	7.2	7.1	6.0	5.0	3.5	5.0

The path of the robot going from one point to another point needs to be smooth so as to avoid sharp jerks in the arm that can otherwise create premature wear and tear of the robot arm.

a) One such path can be the interpolating polynomial $P_5(x)$ that passes those points. Find it.

b) Other paths can be obtained by cubic spline interpolation. Find several cubic splines (see Answers to homework 3) interpolating these data, plot them together with $P_5(x)$ and propose the one with the shortest path.

Problem 3.2

The gamma function is defined as

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t}\, dt$$

and it can be shown (integrate by parts) that $\Gamma(n) = (n-1)!\ \forall n \in \mathbb{N}$.
a) Write a computer programm that will calculate Newton's divided differences and then find (using Newton's formula) the interpolating polynomial $P_4(x)$ of the Γ function at the following points:

n	1	2	3	4	5
$\Gamma(n)$	1	1	2	6	24

b) Find a natural cubic spline $S(x)$ (using an available spline routine) that interpolates the same data.

c) Another approximation can be obtained by first calculating the polynomial $Q_4(x)$ that interpolates points $(n, \log\Gamma(n))$:

n	1	2	3	4	5
$\log\Gamma(n)$	0	0	$\log 2$	$\log 6$	$\log 24$

(here log denotes the natural logarithm) and then, consider $q(x) = e^{Q_4(x)}$.

d) Plot in the same figure the graphs of $\Gamma(x)$, $P_4(x)$, $S(x)$ and $q(x)$ on $[1, 5]$.

e) Compute with accuracy of at least 3 significant digits

$$\max_{x\in[1,5]} \Big|\Gamma(x) - P_4(x)\Big|, \quad \max_{x\in[1,5]} \Big|\Gamma(x) - S(x)\Big| \text{and} \quad \max_{x\in[1,5]} \Big|\Gamma(x) - q(x)\Big|.$$

Which of the 3 approximations is more accurate on $[1, 5]$?

HINTS. In MATLAB or GNU Octave the Γ function is known as `gamma`. In Python it is available in the `scipy.special` library. In order to complete d) you can compute the maximum from the graph, just make sure that sampling is done at sufficiently dense points that will ensure the needed accuracy.

Problem 3.3

Consider function $f(x) = \sqrt{x+1}$ on interval $[-1, 1]$. Find the near minimax polynomial approximation of degree 7 m_7 for this function. Also, find the polynomial interpolant $P_7(x)$ on evenly spaced points. Plot together function $f(x), m_7(x)$ and $P_7(x)$.

Practice problems 2

Problem 2.1

Let the interval used in the bisection method have the length $b - a = 3$. Find the number of midpoints c_n that must be calculated with the bisection method to obtain an approximate root within an error tolerance of 10^{-9}.

Problem 2.2

Imagine you are finding a root α satisfying $1 < \alpha < 2$. If you are using a binary computer with m digits in its significand, what is the smallest error tolerance that makes sense in finding an approximation to α? If the original interval is $[1, 2]$ how many halving are needed to find an approximation to α with the maximum accuracy possible for this computer?

Problem 2.3

Work out what the Newton iteration is for $f(x) = x^2$. What is the solution to $f(x) = 0$? Will the sequence generated by Newton method converge to solution? How quickly? Relate this to the theory of Newton method.

Problem 2.4

On most computers, the computation of $\sqrt{a}$ is based on Newton's method. Set up the Newton's iteration for solving $x^2 - a = 0$, and show that it can be written in the form

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right), \qquad n \geq 0.$$

Derive the error and relative error formulas:

$$\begin{aligned} \sqrt{a} - x_{n+1} &= -\frac{1}{2x_n}\left(\sqrt{a} - x_n\right)^2, \\ \mathrm{Rel}(x_{n+1}) &= -\frac{\sqrt{a}}{2x_n}\left(\mathrm{Rel}(x_n)\right)^2. \end{aligned}$$

For initial guess x_0 near $\sqrt{a}$, the last formula becomes

$$\mathrm{Rel}(x_{n+1}) \approx -\frac{1}{2}\left(\mathrm{Rel}(x_n)\right)^2$$

Assuming $\mathrm{Rel}(x_0) = 0.1$, use this formula to estimate the relative error in $x_i, i = 1, 2, 3, 4$.

Problem 2.5

Derive formula

$$\mathrm{Rel}(x_{n+1}) = \left(\mathrm{Rel}(x_n)\right)^2$$

for the Newton's iterations used in computing $\frac{1}{b}$ for given b (formula was discussed in class without proof).

Problem 2.6

How many solutions are there to the equation $x = e^{-x}$? Will the iteration $x_{n+1} = e^{-x_n}$ converge for a suitable choice of x_0? Use Aitken extrapolation formula to estimate the error $\alpha - x_3$ for $x_0 = 0.57$.

Problem 2.7

The iteration

$$x_{n+1} = 2 - (1 + c)x_n + cx_n^3$$

will converge to $\alpha = 1$ for some values of c (provided that initial guess x_0 is chosen sufficiently close to α). Find the values of c for which convergence occurs. For what values of c, if any, convergence will be quadratic?

Problem 2.8

Consider the equation

$$x^7 - 28x^6 + 322x^5 - 1960x^4 + 6769x^3 - 13132x^2 - 5040 = 0$$

Change the coefficient of x^4 from -1960 to -1960.14. What is relative perturbation error in the coefficient of x^4? Calculate $\alpha(\varepsilon)$ for $\alpha(0) = 3$ and $\alpha(0) = 5$.

Problem 2.9

What is the order of convergence of the iteration

$$x_{n+1} = \frac{x_n(x_n^2 + 15)}{3x_n^2 + 5}$$

as it converges to the fixed point $\alpha = \sqrt{5}$?

Problem 2.10

Newton's method is used to find the root of $f(x) = 0$. The first few iterates are shown in the following table, giving a very slow speed of convergence. What can be said about the root α to explain the convergence? Knowing $f(x)$, how would you find an accurate value for α?

n	x_n	$x_{n-1} - x_n$
0	0.75	
1	0.752710	0.00271
2	0.754795	0.00208
3	0.756368	0.00157
4	0.757552	0.00118
5	0.758441	0.000889

Problem 2.11

Consider the following table of iterates from an iteration method which is convergent to a fixed point α of the function $g(x)$:

n	x_n	$x_n - x_{n-1}$
0	1.30499998	
1	1.25340617	$-5.159E-2$
2	1.21676284	$-3.664E-2$
3	1.19087998	$-2.588E-2$
4	1.17257320	$-1.831E-2$
5	1.15962919	$-1.294E-2$

(a) Does this appear to be a linearly convergent iteration method? If so, then estimate the rate of linear convergence. (b) Estimate the error in x_5 . (c) Give an improved estimate of α .

Practice set 2

ANSWERS

Problem 2.1

Use the error formula for bisection method to get:

$$|\alpha - c_n| \leq \frac{3}{2^n} \leq \varepsilon = 10^{-9}$$
$$\Leftrightarrow n \geq \frac{\ln\left(\frac{3}{10^{-9}}\right)}{\ln 2} \approx 31.48.$$

Therefore $n \geq 32$.

Problem 2.2

The smallest error tolerance that makes sense (since the root is between 1 and 2, no subnormal numbers are used) is the machine epsilon, in this case $\varepsilon = 2^{-m}$. Since we should have

$$\begin{aligned} n &\geq \frac{\ln\left(\frac{1}{2^{-m}}\right)}{\ln 2} \\ &= \frac{\ln\left(2^m\right)}{\ln 2} \\ &= m, \end{aligned}$$

clearly the number of halvings will be m.

Problem 2.3

Solution, obviously is $\alpha = 0$. Newton's method is

$$\begin{aligned} x_{n+1} &= x_n - \frac{x_n^2}{2x_n} \\ &= \frac{1}{2}x_n = \frac{1}{2}\left(\frac{1}{2}x_{n-1}\right) = \frac{1}{2^2}\left(\frac{1}{2}x_{n-2}\right) \\ &= \dots \\ &= \frac{x_0}{2^{n+1}}, \end{aligned}$$

where x_0 is the initial guess. It can be seen that $x_{n+1} \to 0$ for any value of x_0, in other words, the Newton's method will converge, no matter what initial guess is chosen. In order to see the speed of convergence, first we can observe a similar behavior to the behavior of the bisection method (look for the error formula for bisection method).

Therefore, a linear convergence will be expected. Actually, we can easily derive a formula for the error in this case:

$$\begin{aligned} x_{n+1} - \alpha &= x_{n+1} \\ &= \frac{1}{2}x_n \\ &= \frac{1}{2}(x_n - \alpha) \end{aligned}$$

Thus, we have a linear convergence with linear rate $\frac{1}{2}$.

Theoretically, the convergence for Newton's method should be quadratic, but in this case the convergence speed is lower. (Because of the multiplicity of the root!)

Problem 2.4

Let a be given and set $\alpha = \sqrt{a}$ and $f(x) = x^2 - a$. Then for any $n \geq 0$ we have

$$\begin{aligned} x_{n+1} &= x_n - \frac{f(x_n)}{f'(x_n)} \\ &= x_n - \frac{x_n^2 - a}{2x_n} \\ &= \frac{1}{2}\left(x_n + \frac{a}{x_n}\right). \end{aligned}$$

In class we have shown the formula

$$\begin{aligned} \sqrt{a} - x_{n+1} &= -\frac{(\sqrt{a} - x_n)^2}{2} \cdot \frac{f''(c_n)}{f'(x_n)} \\ &= -\frac{(\sqrt{a} - x_n)^2}{2} \cdot \frac{2}{2x_n} \\ &= -\frac{(\sqrt{a} - x_n)^2}{2x_n}. \\ \mathrm{Rel}(x_{n+1}) &= \frac{\sqrt{a} - x_{n+1}}{\sqrt{a}} \\ &= \frac{-\frac{(\sqrt{a}-x_n)^2}{2x_n}}{\sqrt{a}} \\ &= -\frac{(\sqrt{a} - x_n)^2 \cdot \sqrt{a}}{2x_n (\sqrt{a})^2} \\ &= -\frac{\sqrt{a}}{2x_n}\left(\frac{\sqrt{a} - x_n}{\sqrt{a}}\right)^2 \\ &= -\frac{\sqrt{a}}{2x_n}(\mathrm{Rel}(x_n))^2. \end{aligned}$$

For initial guess x_0 near $\sqrt{a}$, the method was shown to converge and therefore $x_n \to \sqrt{a}$. Thus for n big enough, $x_n \approx \sqrt{a}$ and the last formula becomes

$$\mathrm{Rel}(x_{n+1}) \approx -\frac{1}{2}(\mathrm{Rel}(x_n))^2$$

Let $\mathrm{Rel}(x_0) = 0.1$, Then

$$\begin{aligned} \mathrm{Rel}(x_1) &\approx -\frac{1}{2}(\mathrm{Rel}(x_0))^2 = -0.005; \\ \mathrm{Rel}(x_2) &\approx -\frac{1}{2}(\mathrm{Rel}(x_1))^2 \approx -1.25 \cdot 10^{-5}; \\ \mathrm{Rel}(x_3) &\approx -\frac{1}{2}(\mathrm{Rel}(x_2))^2 \approx -7.8125 \cdot 10^{-11}; \\ \mathrm{Rel}(x_4) &\approx -\frac{1}{2}(\mathrm{Rel}(x_3))^2 \approx -3.0518 \cdot 10^{-21}. \end{aligned}$$

Problem 2.5

Let $f(x) = b - \frac{1}{x}$. Root is $\alpha = \frac{1}{b}$. The relative error according to its definition is

$$\begin{aligned} \mathrm{Rel}(x_{n+1}) &= \frac{\alpha - x_{n+1}}{\alpha} \\ &= \frac{\frac{1}{b} - x_{n+1}}{\frac{1}{b}} \\ &= 1 - bx_{n+1}. \end{aligned}$$

In class we showed that Newton's method for the equation $b - \frac{1}{x} = 0$ becomes

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

Thus, combining the last two formulas we have

$$\begin{aligned}
\mathrm{Rel}(x_{n+1}) &= 1 - bx_{n+1} \\
&= 1 - bx_n(2 - bx_n) \\
&= 1 - 2bx_n + b^2x_n^2 \\
&= (1 - bx_n)^2 \\
&= (\mathrm{Rel}(x_n))^2
\end{aligned}$$

Problem 2.6

From the graph it can be seen that we have only one root α between 0.5 and 0.7.

Let $g(x) = e^{-x} \Rightarrow g'(x) = -e^{-x}$. Since

$$\max_{x\in[0.5,0.7]} |g'(x)| = e^{-0.5} \approx 0.6065 < 1,$$

the fixed point iterates $x_{n+1} = e^{-x_n}$ will converge for $x_0 \in [0.5, 0.7]$ (Actually, any $x_0 > 0$ will do fine.). The Aitken extrapolation formula for the error $\alpha - x_3$ is

$$\alpha - x_3 \approx \frac{\lambda_3}{1 - \lambda_3}(x_3 - x_2),$$

where

$$\lambda_3 = \frac{x_3 - x_2}{x_2 - x_1}.$$

We have

i	x_i	λ_i
0	0.57	
1	$5.6553E-1$	
2	$5.6806E-1$	
3	$5.6662E-1$	$-5.6734E-1$

Thus

$$\alpha - x_3 \approx 5.2084E - 4.$$

Problem 2.7

Convergence will happen if

$$|g'(\alpha)| < 1$$

where $g(x) = 2 - (1+c)x + cx^3$ and $\alpha = 1$. Therefore, we have the following condition

$$\begin{aligned} |-(1+c)+3c| &< 1 \Leftrightarrow \\ |-1+2c| &< 1 \Leftrightarrow \\ -1 &< -1+2c < 1 \Leftrightarrow \\ 0 &< c < 1 \end{aligned}$$

Convergence will be at least quadratic, if $g'(1) = 0 \Leftrightarrow -1+2c = 0 \Leftrightarrow c = \frac{1}{2}$. It should be remarked that $g''(1) = 3x \Rightarrow g''(1) \neq 0$. Therefore, convergence will be exactly quadratic.

Problem 2.8

$$|\mathrm{Rel}(-1960.14)| = \frac{-1960-(-1960.14)}{-1960} = 7.1429E-5$$

Consider the root $\alpha(0) = 3$. Let $g(x) = x^4$ and $f(x) = x^7 - 28x^6 + 322x^5 - 1960x^4 + 6769x^3 - 13132x^2 + 13680x - 5040$. Then $f'(x) = 7x^6 - 168x^5 + 1610x^4 - 7840x^3 + 20307x^2 - 26264x + 13680$ and $f'(3) = 660$. From the formula

$$\begin{aligned} \alpha(\varepsilon) &\approx \alpha(0) - \varepsilon\frac{g(\alpha(0))}{f'(\alpha(0))} \\ &= 3 - \varepsilon\frac{g(3)}{f'(3)} \\ &= 3 - 0.14 \cdot \frac{3^4}{660} \\ &\approx 2.9828. \end{aligned}$$

Consider the root $\alpha(0) = 5$. Then $f'(5) = 660$. Similarly,

$$\begin{aligned} \alpha(\varepsilon) &\approx 5 - \varepsilon\frac{g(5)}{f'(5)} \\ &= 5 - 0.14 \cdot \frac{5^4}{660} \\ &\approx 4.8674. \end{aligned}$$

Problem 2.9

Let

$$g(x) = \frac{x(x^2+15)}{3x^2+5}.$$

First, check that $\sqrt{5}$ is a fixed point for function g. Indeed $g(\sqrt{5}) = \sqrt{5}$. Then compute

$$\begin{aligned} g'(x) &= \frac{9x^4 - 30x^2 + 75}{(3x^2+5)^2} \\ g'(\sqrt{5}) &= 0. \end{aligned}$$

Thus, the order of convergence is at least quadratic.

$$\begin{aligned} g''(x) &= \frac{12x(-3x^4 + 31x^2 - 80)}{(3x^2+5)^4} \\ g''(\sqrt{5}) &= 0. \end{aligned}$$

Therefore, the order of convergence is at least cubic. It can be checked that

$$g^{(3)}(\sqrt{5}) \neq 0$$

So, the order of convergence is cubic.

Problem 2.10

n	x_n	$x_{n-1} - x_n$	λ_n
0	0.75		
1	0.752710	0.00271	
2	0.754795	0.00208	0.76753
3	0.756368	0.00157	0.75481
4	0.757552	0.00118	0.75159
5	0.758441	0.000889	0.75339

We can see that $\lambda_n \to 0.75 = \frac{3}{4}$. Therefore, we can say that our root have multiplicity 4. In order to find an accurate value of α, we compute the $f^{(3)}(x)$ and apply the Newton's method to this equation $f^{(3)}(x) = 0$. Since α is a simple root of $f^{(3)}(x)$,Newton$'$s method should converge much faster.

Problem 2.11

n	x_n	$x_n - x_{n-1}$	λ_n
0	1.30499998		
1	1.25340617	$-5.159E-2$	
2	1.21676284	$-3.664E-2$	$7.102E-1$
3	1.19087998	$-2.588E-2$	$7.063E-1$
4	1.17257320	$-1.831E-2$	$7.075E-1$
5	1.15962919	$-1.294E-2$	$7.06717E-1$

Since λ_n obviusly are converging to 0.707, we have $g'(\alpha) \approx 0.707$ and

$$\alpha - x_{n+1} \approx 0.707(\alpha - x_{n+1})$$

which means that convergence is linear with linear rate approximately 0.707. By Aitken error estimation formula we have

$$\begin{aligned} \alpha - x_5 &\approx \frac{\lambda_5}{1-\lambda_5}(x_5 - x_4) \\ &\approx \frac{0.706717}{1-0.76717} \cdot (-0.001294) \\ &\approx -3.92772E-3. \end{aligned}$$

By Aitken extrapolation formula

$$\begin{aligned} \alpha &\approx x_5 + \frac{\lambda_5}{1-\lambda_5}(x_5 - x_4) \\ &\approx 1.15962919 + (-3.92772E-3) \\ &= 1.15570147. \end{aligned}$$

Practice Set 3

ANSWERS

Problem 3.1

Consider a general cubic polynomial

$$P(x) = a + bx + cx^2 + dx^3.$$

Then

$$P'(x) = b + 2cx + 3dx^2.$$

From conditions

$$y_1 = P(0), \quad y_2 = P(1), \quad y_1' = P'(0), \quad y_2' = P'(1),$$

we get

$$\begin{aligned} y_1 &= P(0) = a, \\ y_2 &= P(1) = a + b + c + d, \\ y_1' &= P'(0) = b, \\ y_2' &= P'(1) = b + 2c + 3d, \end{aligned}$$

Solve this system with unknowns a, b, c and d

$$\begin{aligned} y_1 &= a, \\ y_2 &= a + b + c + d, \\ y_1' &= b, \\ y_2' &= b + 2c + 3d, \end{aligned}$$

and obtain solution

$$\begin{aligned} a &= y_1, \\ b &= y_1', \\ c &= -3y_1 + 3y_2 - 2y_1' - y_2', \\ d &= 2y_1 - 2y_2 + y_1' + y_2', \end{aligned}$$

Thus we have

$$\begin{aligned} P(x) &= y_1 + y_1'x + \big(-3y_1 + 3y_2 - 2y_1' - y_2'\big)x^2 + \big(2y_1 - 2y_2 + y_1' + y_2'\big)x^3 \\ &= \big(2x^3 - 3x^2 + 1\big)y_1 + \big(-2x^3 + 3x^2\big)y_2 + \big(x^3 - 2x^2 + x\big)y_1' + \big(x^3 - x^2\big)y_2' \\ &= \big(1 + 2x\big)\big(1 - x\big)^2 y_1 + x^2\big(3 - 2x\big)y_2 + x\big(1 - x\big)^2 y_1' + x^2\big(x - 1\big)y_2' \\ &= H_1(x)y_1 + H_2(x)y_2 + H_3(x)y_1' + H_4(x)y_2' \end{aligned}$$

with

$$\begin{aligned} H_1(x) &= \big(1 + 2x\big)\big(1 - x\big)^2, \\ H_2(x) &= x^2\big(3 - 2x\big), \\ H_3(x) &= x\big(1 - x\big)^2, \\ H_4(x) &= x^2\big(x - 1\big), \end{aligned}$$

Observe that

$$\begin{array}{llll} H_1(0) = 1, & H_1(1) = 0, & H_1'(0) = 0, & H_1'(1) = 0, \\ H_2(0) = 0, & H_2(1) = 1, & H_2'(0) = 0, & H_2'(1) = 0, \\ H_3(0) = 0, & H_3(1) = 0, & H_3'(0) = 1, & H_3'(1) = 0, \\ H_4(0) = 0, & H_4(1) = 0, & H_4'(0) = 0, & H_4'(1) = 1, \end{array}$$

Problem 3.2

Look for function $Q_1(x) = a + b\cos(\pi x) + c\sin(\pi x)$ such that

$$\begin{cases} a + b\cos(\pi \cdot 0) + c\sin(\pi \cdot 0) = Q_1(0) = 2 \\ a + b\cos(\pi \cdot \frac{1}{2}) + c\sin(\pi \cdot \frac{1}{2}) = Q_(\frac{1}{2}) = 5 \\ a + b\cos(\pi \cdot 1) + c\sin(\pi \cdot 1) = Q_1(1) = 4 \end{cases}$$

This leads to the following system

$$\begin{cases} a + b = 2 \\ a + c = 5 \\ a - b = 4 \end{cases}$$

that has solution $a = 3$, $b = -1$, $c = 2$. Therefore the function is $Q_1(x) = 3 - \cos(\pi x) + 2\sin(\pi x)$.

In order to find the quadratic interpolation polynomial use Newton's divided difference formula. First, need to compute Newton's divided differences:

x	y	D_1	D_2
0	2	6	–8
0.5	5	–2	
1	4		

And interpolating polynomial is $P_2(x) = 2 + 6x - 8x(x - \frac{1}{2}) = -8x^2 + 10x + 2$.

Figure 1: Graphs of quadratic interpolant $P_2(x) = -8x^2 + 10x + 2$ (red) and trigonometric interpolant $Q_1(x) = 3 - \cos(\pi x) + 2\sin(\pi x)$ (blue).

Problem 3.3

Use Newton's divided difference formula

x	y	D_1	D_2	D_3
0	–1	5	–7/2	13/15
1	4	–2	5/6	
2	2	4/3		
5	6			

Interpolating cubic polynomial is $P_3(x) = -1 + 5x - \frac{7}{2}x(x-1) + \frac{13}{15}x(x-1)(x-2)$.

Problem 3.4

Observe that data are satisfying relation $y = x^2$, i.e. interpolating points are located on parabola $y = x^2$. Since interpolation polynomial is unique, the answer is $P(x) = x^2$.

Problem 3.5

Let $N+1$ interpolation points be

$$x_0 < x_1 < x_2 < \cdots < x_{N-1} < x_N$$

and let $L_i(x)$, $i = 0, 1, \ldots, N$ be $N+1$ associated Lagrange basis functions. Need to prove that

$$\sum_{i=0}^{N} L_i(x) = 1. \tag{1}$$

Proof.
Rewrite the left side of the identity (1) in the form

$$1 \cdot L_0(x) + 1 \cdot L_1(x) + 1 \cdot L_2(x) + \cdots + 1 \cdot L_N(x) \tag{2}$$

and compare it with the formula for interpolation polynomial for the data (x_i, y_i), $i = 0, 1, \ldots, N$:

$$y_0 \cdot L_0(x) + y_1 \cdot L_1(x) + y_2 \cdot L_2(x) + \cdots + y_N \cdot L_N(x).$$

Obviuosly (2) represents the interpolation polynomial that interpolates points $\{(x_0, 1), (x_1, 1), (x_2, 1), \ldots, (x_N, 1)\}$. These points lie on the line $y = 1$, therfore, since interpolation polynomial is unique, it follows that interpolation polynomial given by (2) is identical function 1. So

$$1 \cdot L_0(x) + 1 \cdot L_1(x) + 1 \cdot L_2(x) + \cdots + 1 \cdot L_N(x) \equiv 1.$$

and identity (1) is proved. ■

Problem 3.6

Let $P_2(x)$ be the quadratic interpolation polynomial of function e^{-x^2} at the points $x_0 = 0$, $x_1 = \frac{1}{2}$ and $x_2 = 1$. The interpolation error is given by

$$e^{-x^2} - P_2(x) = \frac{x(x-\frac{1}{2})(x-1)}{6}\left(e^{-x^2}\right)'''(\theta) \tag{3}$$

for some $\theta \in [0, 1]$. Let $h = \frac{1}{2}$. Using error formula (3) we get

$$\begin{aligned}\left|e^{-x^2} - P_2(x)\right| &= \frac{\left|x(x-h)(x-2h)\right|}{6}\left|\left(e^{-x^2}\right)'''(\theta)\right| \\ &\le \frac{1}{6} \cdot \max_{x\in[0,1]} \left|x(x-h)(x-2h)\right| \cdot \max_{x\in[0,1]} \left|\left(e^{-x^2}\right)'''(x)\right|\end{aligned} \tag{4}$$

From Lecture 9 (pages $8-10$) we know that

$$\max_{x_0 \le x \le x_2} \left|(x-x_0)(x-x_1)(x-x_2)\right| = \frac{2h^3}{3\sqrt{3}}, \text{ with } h = x_1 - x_0 = x_2 - x_1.$$

Thus,

$$\max_{x\in[0,1]} \left|x(x-h)(x-2h)\right| = \frac{2 \cdot \frac{1}{2^3}}{3\sqrt{3}} = \frac{1}{12\sqrt{3}} \approx 0.048113 \tag{5}$$

Next compute the 3rd derivative of e^{-x^2}:

$$\begin{aligned}\left(e^{-x^2}\right)' &= -2xe^{-x^2} \\ \left(e^{-x^2}\right)'' &= (4x^2 - 2)e^{-x^2} \\ \left(e^{-x^2}\right)''' &= (-8x^3 + 12x)e^{-x^2}\end{aligned}$$

Figure 2: Graph of $(-8x^3 + 12x)e^{-x^2}$

In order to find the maximum of $(-8x^3 + 12x)e^{-x^2}$ on $[0, 1]$, plot its graph.
It follows from the graph that

$$\max_{x\in[0,1]} \left| \left(e^{-x^2}\right)'''(x) \right| \approx 3.9032 \quad \text{at } x \approx 0.52 \tag{6}$$

Substituting (6) and (5) in inequaity (4) we obtain

$$\left| e^{-x^2} - P_2(x) \right| \leq \frac{1}{6} \cdot 0.048113 \cdot 3.9032 \approx 0.031299 = 3.13E - 2$$

Problem 3.7

In order to be a cubic spline, function $s(x)$ should have the properties (see Lecture 11):

1. $s(x)$ is a piecewise cubic polynomial;
2. $s(x)$, $s'(x)$ and $s''(x)$ should be continuous functions.

Let

$$s(x) = \begin{cases} (x-1)^3, & 0 \leq x \leq 1 \\ 2(x-1)^3, & 1 \leq x \leq 2 \end{cases}$$

Obviously condition 1 is satisfied. Then

$$s'(x) = \begin{cases} 3(x-1)^2, & 0 \leq x \leq 1 \\ 6(x-1)^2, & 1 \leq x \leq 2 \end{cases}$$

and

$$s''(x) = \begin{cases} 6(x-1), & 0 \leq x \leq 1 \\ 12(x-1), & 1 \leq x \leq 2 \end{cases}$$

It can be easily checked that $s_-(1) = 0 = s_+(1)$, $s'_-(1) = 0 = s'_+(1)$ and $s''_-(1) = 0 = s''_+(1)$. Thus, condition 2 is also satisfied. Therefore the function $s(x)$ is a cubic spline. Moreover, since $s''(0) = -6 \neq 0$ and $s''(2) = 12 \neq 0$, this is not a "natural" cubic spline.

Problem 3.8

(a) Piecewise linear interpolant $P_{1,p}(x)$ is

$$P_{1,p}(x) = \begin{cases} \frac{1}{2}x, & 0 \le x \le \frac{1}{2} \\ \frac{3}{2}x - \frac{1}{2}, & \frac{1}{2} \le x \le 1 \\ -2x+3, & 1 \le x \le 2 \\ -1, & 2 \le x \le 3 \end{cases}$$

(b) Use Newton's divided differences to find piecewise quadratic interpolant $P_{2,p}(x)$.

x	y	D_1	D_2
0	0	1/2	1
1/2	1/4	3/2	
1	1		

x	y	D_1	D_2
1	1	–2	1
2	–1	0	
3	–1		

On $\{0, 1/2, 1\}$ quadratic polynomial is $0 + 1/2(x-0) + 1(x-0)(x-1/2) = x^2$
and on $\{1, 2, 3\}$ quadratic polynomial is $1 + (-2)(x-1) + 1(x-1)(x-2) = x^2 - 5x + 5$.
Therefore, piecewise quadratic interpolant is:

$$P_{1,p}(x) = \begin{cases} x^2, & 0 \le x \le 1 \\ x^2 - 5x + 5, & 1 \le x \le 3 \end{cases}$$

(c) Since it was not specifically required to obtain the natural cubic spline analytically, we will use GNU Octave/MATLAB built-in functions to get it.

In GNU Octave define the interpolating data and apply *csape* function:

```
>>xx=[0, 0.5, 1, 2, 3];
>>yy=[0, 0.25, 1, -1, -1];
>>pp=csape(xx, yy, 'variational')
pp =
  scalar structure containing the fields:

    form = pp

    breaks =
       0.00000   0.50000   1.00000   2.00000   3.00000

    coefs =
       1.80952   0.00000   0.04762   0.00000
      -5.04762   2.71429   1.40476   0.25000
       2.52381  -4.85714   0.33333   1.00000
      -0.90476   2.71429  -1.80952  -1.00000

    pieces =  4
    order =  4
    dim =  1
```

Matrix `coefs` contains the coefficients of piecewise cubic polynomial:
if row i of matrix `coefs` is $[a\,b\,c\,d]$ then $s(x) = a(x-x_i)^3 + b(x-x_i)^2 + c(x-x_i) + d$ if $x \in [x_i, x_{i+1}]$

Therefore, we have the following natural cubic spline function that interpolates our data.

$$sn(x) = \begin{cases} 1.80952x^3 + 0.04762x, & 0 \le x \le \frac{1}{2} \\ -5.04762(x-0.5)^3 + 2.71429(x-0.5)^2 + 1.40476(x-0.5) + 0.25, & \frac{1}{2} \le x \le 1 \\ 2.52381(x-1)^3 - 4.85714(x-1)^2 + 0.33333(x-1) + 1.0, & 1 \le x \le 2 \\ -0.90476(x-2)^3 + 2.71429(x-2)^2 - 1.80952(x-2) - 1.0, & 2 \le x \le 3 \end{cases}$$

Let's plot all three interpolants:

Figure 3: Graphs of piecewise linear interpolant (red), piecewise quadratic interpolant (black) and natural cubic spline interpolant (blue)

IMPORTANT REMARK:
Beware and read help/refference documentation initially, since, as it was mentioned in class lecture, there are variuous boundary conditions, and thus many types of spline functions exist. For example,

- **natural** (known in Matlab and GNU Octave as 'variational') cubic spline uses condition $s''(x_1) = s''(x_N) = 0$.
- **not-a-knot** cubic spline uses condition that $s'''(x)$ is continuous at x_2 and x_{N-1}.
- **complete** (also known as clamped) cubic spline uses condition $s'(x_1) = A$ and $s'(x_2) = B$ with values A and B provided beforehand.
- **periodic** cubic spline uses condition $s'(x_1) = s'(x_N)$.
- **second** cubic spline uses condition $s'(x_1) = C$ and $s'(x_2) = D$ with values A and B provided beforehand.

In order to obtain natural cubic spline we will use *csape* function provided by MATLAB library. For using it in GNU Octave you need to download and install additional package *splines*. Let Google be with you! I was able to do it, so will you. The available by default with GNU Octave basic distribution function *spline* can do only not-a-knot and clamped cubic splines.

As a bonus let's compare 3 different cubic spline interpolants.

- natural cubic spline obtained previously;
- clamped cubic spline (obtained using `spline` function from GNU Octave);
- not-a-knot cubic spline (aobtained using `spline` function).

Figure 4: Natural cubic spline (blue), clamped cubic spline (red), not-a-knot cubic spline (black)

Problem 3.9

Consider the error formula for minimax approximation (see Lecture 10, page 5)

$$\rho_n(f) \leq \frac{\left(\frac{b-a}{2}\right)^{n+1}}{(n+1)!\,2^n} \max_{x\in[a,b]} \left|f^{(n+1)}(x)\right|$$

Substituting $a = -1$, $b = 2$, $n = 5$ we get error estimate formula

$$\rho_5(f) \leq \frac{\left(\frac{3}{2}\right)^6}{6!\,2^5} \max_{x\in[-1,2]} \left|f^{(6)}(x)\right| = \frac{3^6}{6!\,2^{11}} \max_{x\in[-1,2]} \left|f^{(6)}(x)\right|,$$

where $f(x) = e^{3x-1}$. Compute derivatives

$$\left(e^{3x-1}\right)' = 3e^{3x-1}, \quad \left(e^{3x-1}\right)'' = 9e^{3x-1}, \quad \ldots, \quad \left(e^{3x-1}\right)^{(6)} = 3^6 e^{3x-1}$$

Since

$$\max_{x\in[-1,2]} \left|f^{(6)}(x)\right| = 3^6 \max_{x\in[-1,2]} \left|e^{3x-1}\right| = 3^6 \cdot e^{3\cdot 2-1} = 3^6 e^5$$

error estiamte formula becomes

$$\rho_5(e^{3x-1}) \leq \frac{3^{12}e^5}{6!\,2^{11}} \approx 53.489$$

It seems that this is a very crude estimate, but keep in mind that fucntion e^{3x-1} on interval $[-1, 2]$ grows fast from 0.018 up to 148.41. So we can repeat the same computations for $n = 10$ and $n = 15$ to get

$$\rho_{10}(e^{3x-1}) \leq \frac{3^{22}e^5}{11!\,2^{21}} \approx 5.564E-2, \quad \rho_{15}(e^{3x-1}) \leq \frac{3^{32}e^5}{16!\,2^{31}} \approx 6.1207E-6$$

Problem 3.10

Recall the triple recurrsion formula for Chebyshev polynomials (see Lecture10, page 8)

$$\begin{aligned} T_0(x) &= 1, \\ T_1(x) &= x, \\ T_{n+1}(x) &= (2x) \cdot T_n(x) - T_{n-1}(x), \end{aligned}$$

Evaluation of T_0 and T_1 is straightforward, then T_2 will need 2 multiplications and 1 addition, and every next polynomial will need 1 more multiplication and 1 more addition. Therefore, evaluation at a particular x of Chebyshev polynomials $T_0(x), T_1(x), T_2(x), \dots T_n(x)$ will need n multiplications and $n-1$ additions.

Problem 3.11

Let $q(x)$ be a polynomial of degree $n-1$ and consider the polynomial $x^n - q(x)$. It is a polynomial of degree n, moreover it is a monic polynomial (see Lecture 10). According to the **Theorem on minimum size property** from page 11, the degree n monic polynomial with the smallest maximum on $[-1,1]$ is the modified Chebyshev polynomial $\widetilde{T}_n(x)$, and its maximum value on $[-1,1]$ is $\frac{1}{2^{n-1}}$. Thus,

$$\max_{x\in[-1,1]} \left|x^n - q(x)\right| = \frac{1}{2^{n-1}}$$

and it is achieved for $q(x) = x^n - \widetilde{T}_n(x)$.

Problem 3.12

Proof.
Consider substitution

$$x = \cos\theta, \quad \theta = \arccos x, \quad \sqrt{1-x^2} = \sqrt{1-\cos^2\theta} = \sin\theta, \quad dx = d(\cos\theta) = -\sin\theta\, d\theta.$$

Then

$$\begin{aligned} \int_{-1}^{1} \frac{T_n(x)T_m(x)}{\sqrt{1-x^2}}\, dx &= \int_{\pi}^{0} \frac{\cos(n\theta)\cos(m\theta)(-\sin\theta)\, d\theta}{\sin\theta} \\ &= \int_0^{\pi} \cos(n\theta)\cos(m\theta)\, d\theta \\ &= 0, \text{ if } n \neq m. \end{aligned}$$

■

Practice Problems Set 3

Problem 3.1

Find a polynomial $P(x)$ of degree ≤ 3 for which

$$\begin{aligned} P(0) &= y_1 \qquad & P(1) &= y_2 \\ P'(0) &= y_1' \qquad & P'(1) &= y_2' \end{aligned}$$

with y_1, y_2, y_1', y_2' given constants.
The resulting polynomial is called **cubic Hermite interpolating polynomial**.
HINT: Write $P(x) = y_1 H_1(x) + y_2 H_2(x) + y_1' H_3(x) + y_2' H_4(x)$ with H_i cubic polynomials satisfying appropriate properties, in analogy with Lagrange interpolating polynomials.

Problem 3.2

Find the function $P(x) = a + b\cos(\pi x) + c\sin(\pi x)$, which interpolates the data

x	0	0.5	1
y	2	5	4

This is so-called **trigonometric interpolation**. Also, find the quadratic polynomial interpolating this data. In each instance, draw the graph of the interpolating function.

Problem 3.3

Find cubic polynomial interpolating the data

x	0	1	2	5
y	-1	4	2	6

Problem 3.4

Find the polynomial interpolating the data

x	-3	1	2	4	5
y	9	1	4	16	25

Problem 3.5

Prove that

$$\sum_{i=0}^{N} L_i(x) = 1,$$

where $L_i(x)$ are Lagrange basis functions associated to $N+1$ interpolation points.

Problem 3.6

Consider the polynomial interpolation of the function $f(x) = e^{-x^2}$ on $[0,1]$ at the points $x_0 = 0$, $x_1 = 0.5$ and $x_2 = 1$. Estimate the maximum of the polynomial interpolation error for $x \in [0,1]$, i.e. give an upper bound for this error.

Problem 3.7

Is the following a cubic spline on the interval $0 \leq x \leq 2$?

$$s(x) = \begin{cases} (x-1)^3, & 0 \leq x \leq 1 \\ 2(x-1)^3, & 1 \leq x \leq 2 \end{cases}$$

Problem 3.8

Consider the data

x	0	1/2	1	2	3
y	0	1/4	1	-1	-1

(a) Find the piecewise linear interpolating function for the data; (b) Find the piecewise quadratic interpolating function for the data. (c) Find the natural cubic spline that interpolates the data. Graph all three graphs for $0 \leq x \leq 3$.

Problem 3.9

Compute the error bound for the minimax approximation of the function $f(x) = e^{3x-1}$ on the $[-1, 2]$ and $n = 5$.

Problem 3.10

How many multiplications and additions are needed to compute Chebyshev polynomials $T_0(x), T_1(x), T_2(x), \ldots, T_n(x)$ for a particular value of x?

Problem 3.11

Let $q(x)$ be a polynomial of degree $\leq n-1$, and consider

$$\max_{-1 \leq x \leq 1} |x^n - q(x)|$$

What is the smallest possible value for this quantity? Solve for the $q(x)$ for which this value is attained.

Problem 3.12

For $n, m \geq 0$ and $n \neq m$ show

$$\int_{-1}^{1} \frac{T_n(x) T_m(x)}{\sqrt{1-x^2}} dx = 0$$

This is called the orthogonality property for the Chebyshev polynomials.

Homework 1

Problem 1.1

Let d_1d_2/m_1m_2 be your birthday. Find the binary single precision IEEE floating-point expression for the number $(d_1m_2)_{10}$. Also, find the binary double precision IEEE floating-point expression for the number $(d_2m_1)_{10}$. In both cases, specify significand, exponent and σ. Convert $(111\ldots1)_2$ to decimal form with the parentheses enclosing $(21 - m_1m_2)$ ones.

Problem 1.2

Some microcomputers in the past used a binary floating-point format with 8 bits for the exponent and 1 bit for the sign σ. The significand contained 31 bits, with no hiding of the leading bit 1. The arithmetic used rounding. To determine the accuracy of the representation, find the machine epsilon, integer M, and the largest number that can be represented exactly in this floating-point format. Also, find the accuracy of the rounding operation.

Problem 1.3

Consider a binary floating-point representation with significand containing 3 digits without hiding the leading 1 and $-1_{10} \leq e \leq 3_{10}$. List all numbers that can be stored exactly together with their decimal value. Plot these numbers on real axis. For this arithmetic, specify what are the corresponding floating-point representation of $\pi/4$ and $14/5$ if a) rounding is used; b) chopping is used?

Problem 1.4

Calculate the error, relative error and the number of significant digits in the following approximations $x_A \approx x_T$.

$$
\begin{aligned}
&a) \quad x_A = 28.271, \quad x_T = 28.254;\\
&b) \quad x_A = 0.028271, \quad x_T = 0.028254;\\
&c) \quad x_A = 19/7, \quad x_T = e;\\
&d) \quad x_A = 1.414, \quad x_T = \sqrt{2}.
\end{aligned}
$$

Problem 1.5

Avoid loss-of-significance errors in the following formulas

$$
\begin{aligned}
&a) \quad \log(x+1) - \log(x) \quad \text{for large values of } x;\\
&b) \quad \frac{e^x - 1}{x} \quad \text{for small values of } x;\\
&c) \quad \sin(x+a) - \sin(a) \quad \text{for small values of } x;\\
&d) \quad \sqrt[3]{x+1} - \sqrt[3]{x} \quad \text{for large values of } x;\\
&e) \quad \sqrt{1 + \frac{1}{x} - 1} \quad \text{for large values of } x.
\end{aligned}
$$

Problem 1.6

In the following function evaluations $f(x_A)$, assume the numbers x_A are correctly rounded to the number of digits shown. Bound the error $f(x_T) - f(x_A)$ and the relative error $Rel(x_A)$:

$$
\begin{aligned}
&a) \quad cos(1.473);\\
&b) \quad e^{2.231};\\
&c) \quad \sqrt{0.0275};\\
&d) \quad \arctan(4.7869).
\end{aligned}
$$

Problem 1.7

Extra Bonus point Bound

$$\int_0^{\pi} \frac{t^2}{1+t^4}\,dt - \int_0^{22/7} \frac{t^2}{1+t^4}\,dt.$$

Hint: Define function

$$f(x) = \int_0^{x} \frac{t^2}{1+t^4}\,dt.$$

and then bound $f(\pi) - f(22/7)$.

Numerical Analysis

Homework 1. Answers to some problems.

Problem 1

$(2.8)_{10} = (10.1100110011001100110011 0011\ldots)_2$
$= (1.0110011001100110011001100110011\ldots)_2 \cdot 2^1$
$\sigma = +1;\ e = 1; \overline{x} = (1.0110011001100110011001100110011\ldots)_2$

In single precision IEEE format $E = e + 127 = 128_{10} = (10000000)_2$;

1	10000000	01100110011001100110011

Double precission is done similarly.

$\underbrace{(111\ldots1)_2}_{21-8=13} = 2^{12} + 2^{11} + \ldots 2 + 2^0 = \frac{2^{13}-1}{2-1} = 2^{13} - 1 = 8191$

Problem 2 *Machine epsilon is the difference between* 1 *and the next larger number that can be stored in the given format. Therefore, in this format,*

Machine epsilon $= (1.00000\ldots.0001)_2 - 1_2$ (29 *zeros*) $= (0.00000\ldots0001)_2 = 2^{-30}$

M, by definition given in class, is the largest integer having the property that any integer x smaller than M can be stored exactly in this floating-point format.

If 31 is the number of binary digits in the significand, then all integers less than or equal to $(11\ldots1)_2$ can be stored exactly, where this significand contains 31 digits, all 1. It equals to $2^{31} - 1$.

In addition, the number $2^{31} = (1.0\ldots0)_2 \cdot 2^{31}$ also stores exactly.

Note that there not enough digits to store $2^{31} + 1$, as this would require 32 binary digits in the significand.

Therefore $M = 2^{31} \approx 2.147 \cdot 10^9$.

The largest number that can be represented in this format is $(1.111\ldots111)_2 \cdot 2^e$, where significand contains 31 ones, and the exponent $e = (11111111)_2 - 127_{10} = 255 - 127 = 128_{10}$. It is approximately 10^{37}.

Since rounding is used, the bounds for ε are

$$-2^{-31} \leq \varepsilon \leq 2^{-31}$$

Problem 3 *List of numbers in decimal format:*

$0.5, 0.625, 0.75, 0.875,$
$1, 1.25, 1.5, 1.75,$
$2, 2.5,$
$3, 3.5,$
$4, 5, 6, 7, 8,$
$10, 12, 14$

$\pi/4 \approx 0.785398163$. In this arithmetic it will be stored as 0.75 if both rounding and chopping are used.

$14/5 = 2.8$ In this arithmetic it will be stored as 3 if rounding is used.and 2.5 in case of chopping.

Problem 4 *a)* $Error = 28.254 - 28.271 = -0.017; \quad Rel(x_A) = -\frac{0.017}{28.254} = -0.00060168.$

Three significant digits;
Similarly
b) Three significant digits; c) Three significant digits; d) Four significant digits.

Problem 5

$$a) \quad Use \quad \log(x+1) - \log(x) = \log\frac{x+1}{x}$$
$$b) \quad Use \quad Taylor \quad decomp. \quad for \quad e^x$$
$$c) \quad Use \ \sin(x+a) - \sin(a) = 2\cos\frac{x+2a}{2}\sin\frac{x}{2}$$
$$d) \quad Use \quad a - b = \frac{a^3 - b^3}{a^2 + ab + b^2}$$
$$e) \quad Use \quad = \frac{\left(\sqrt{1+\frac{1}{x}} - 1\right)\left(\sqrt{1+\frac{1}{x}} + 1\right)}{\sqrt{1+\frac{1}{x}} + 1} = \frac{\sqrt{x}}{\sqrt{x+1} + \sqrt{x}}$$

Problem 6 *Use formula*

$$|f(x_T) - f(x_A)| \approx |f"(x_A)| \cdot |x_T - x_A|$$

a) Since x_A was correctly rounded, $|x_T - x_A| \leq 0.0005$. Therefore, $|f(x_T) - f(x_A)| \leq |\sin 1.4713| \cdot 0.0005 \approx 4.9761 \cdot 10^{-4}$
$|rel(f(x_A))| \approx \frac{|f"(x_A)| \cdot |x_T - x_A|}{|f(x_A)|} \approx 0.0051$
b,c,d) are done similarly.

Problem 7 *Define*

$$f(x) = \int_0^x \frac{t^2}{1+t^4} dt. \quad Then \quad f'(x) = \frac{x^2}{1+x^4}$$

Thus $f(\pi) - f(22/7) \approx f'(x_A)(x_T - x_A) = f'\left(\frac{22}{7}\right)\left(\pi - \frac{22}{7}\right) = \frac{\left(\frac{22}{7}\right)^2}{1+\left(\frac{22}{7}\right)^4}\left(\pi - \frac{22}{7}\right) \approx -1.2672 \cdot 10^{-4}$.

THE EIGENVALUE PROBLEM

Let A be an $n \times n$ square matrix. If there is a number λ and a column vector $v \neq 0$ for which

$$Av = \lambda v$$

then we say λ is an *eigenvalue* of A and v is an associated *eigenvector*. Note that if v is an eigenvector, then any nonzero multiple of v is also an eigenvector for the same eigenvalue λ.

Example: Let

$$A = \begin{bmatrix} 1.25 & 0.75 \\ 0.75 & 1.25 \end{bmatrix} \tag{1}$$

The eigenvalue-eigenvector pairs for A are

$$\begin{aligned} \lambda_1 &= 2, \qquad v^{(1)} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\ \lambda_2 &= 0.5, \qquad v^{(2)} = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \end{aligned} \tag{2}$$

Eigenvalues and eigenvectors are often used to give additional intuition to the function

$$F(x) = Ax, \qquad x \in \mathbb{R}^n \text{ or } \mathbb{C}^n$$

Example. The eigenvectors in the preceding example (2) form a *basis* for $\mathbb{R}^2$. For $x = [x_1, x_2]^T$,

$$\begin{aligned} x &= c_1 v^{(1)} + c_2 v^{(2)} \\ c_1 &= \frac{x_1 + x_2}{2}, \quad c_2 = \frac{x_2 - x_1}{2} \end{aligned}$$

Using (2), the function

$$F(x) = Ax, \qquad x \in \mathbb{R}^2$$

can be written as

$$\begin{aligned} F(x) &= c_1 A v^{(1)} + c_2 A v^{(2)} \\ &= 2c_1 v^{(1)} + \frac{1}{2} c_2 v^{(2)}, \quad x \in \mathbb{R}^2 \end{aligned}$$

The eigenvectors provide a better way to understand the meaning of $F(x) = Ax$.

See the following Figure 1 and Figure 2.

Figure 1: Decomposition of x

Figure 2: Decomposition of Ax

How to calculate λ and v? Rewrite $Av = \lambda v$ as

$$(\lambda I - A)v = 0, \qquad v \neq 0 \tag{3}$$

a homogeneous system of linear equations with the coefficient matrix $\lambda I - A$ and the nonzero solution v. This can be true if and only if

$$f(\lambda) \equiv \det(\lambda I - A) = 0$$

The function $f(\lambda)$ is called the *characteristic polynomial* of A, and its roots are the eigenvalues of A.
Assuming A has order n,

$$f(\lambda) = \lambda^n + \alpha_{n-1}\lambda^{n-1} + \cdots + \alpha_1\lambda + \alpha_0 \tag{4}$$

For the case $n = 2$,

$$\begin{aligned} f(\lambda) &= \det \begin{bmatrix} \lambda - a_{11} & -a_{12} \\ -a_{21} & \lambda - a_{22} \end{bmatrix} \\ &= (\lambda - a_{11})(\lambda - a_{22}) - a_{21}a_{12} \\ &= \lambda^2 - (a_{11} + a_{22})\lambda + a_{11}a_{22} - a_{21}a_{12} \end{aligned}$$

The formula (4) shows that a matrix A of order n can have at most n distinct eigenvalues.

Example. Let

$$A = \begin{bmatrix} -7 & 13 & -16 \\ 13 & -10 & 13 \\ -16 & 13 & -7 \end{bmatrix} \tag{5}$$

Then

$$\begin{aligned} f(\lambda) &= \det \begin{bmatrix} \lambda + 7 & -13 & 16 \\ -13 & \lambda + 10 & -13 \\ 16 & -13 & \lambda + 7 \end{bmatrix} \\ &= \lambda^3 + 24\lambda^2 - 405\lambda + 972 \end{aligned}$$

is the characteristic polynomial of A.

The roots are

$$\lambda_1 = -36, \qquad \lambda_2 = 9, \qquad \lambda_3 = 3 \tag{6}$$

Finding an eigenvector. For $\lambda = -36$, find an associated eigenvector v by solving $(\lambda I - A)v = 0$, which becomes

$$(-36I - A)v = 0$$

$$\begin{bmatrix} -29 & -13 & 16 \\ -13 & -26 & -13 \\ 16 & -13 & -29 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

If $v_1 = 0$, then the only solution is $v = 0$. Thus $v_1 \neq 0$, and we arbitrarily choose $v_1 = 1$. This leads to the system

$$\begin{aligned} -13v_2 + 16v_3 &= 29 \\ -26v_2 - 13v_3 &= 13 \\ -13v_2 - 29v_3 &= -16 \end{aligned}$$

The solution is $v_2 = -1$, $v_3 = 1$. Thus the eigenvector v for $\lambda = -36$ is

$$v^{(1)} = [1, -1, 1]^T \tag{7}$$

or any nonzero multiple of this vector.

SYMMETRIC MATRICES

Recall that A *symmetric* means $A^{\mathsf{T}} = A$, assuming that A contains only real number entries. Such matrices are very common in applications.

Example. A general 3×3 symmetric matrix looks like

$$A = \begin{bmatrix} a & b & c \\ b & d & e \\ c & e & f \end{bmatrix}$$

A general $n \times n$ matrix $A = \left[a_{i,j}\right]$ is symmetric if and only if

$$a_{i,j} = a_{j,i}, \qquad 1 \le i, j \le n$$

Following is a very important set of results for symmetric matrices, explaining much of their special character.

THEOREM. Let A be a real, symmetric, $n \times n$ matrix. Then there is a set of n eigenvalue-eigenvector pairs $\{\lambda_i, v^{(i)}\}$, $1 \leq i \leq n$ that satisfy the following properties.

(i) The numbers $\lambda_1, \lambda_2, \ldots, \lambda_n$ are all of the roots of the characteristic polynomial $f(\lambda)$ of A, repeated according to their multiplicity. Moreover, all the λ_i are real numbers.

(ii) If the column matrices $v^{(i)}$, $1 \leq i \leq n$ are regarded as vectors in n-dimensional space, then they are mutually perpendicular and of length 1:

$$\begin{aligned} v^{(i)\mathsf{T}} v^{(j)} &= 0, \qquad 1 \leq i, j \leq n, \quad i \neq j \\ v^{(i)\mathsf{T}} v^{(i)} &= 1, \qquad 1 \leq i \leq n \end{aligned}$$

(iii) For each column vector $x = [x_1, x_2, \ldots, x_n]^\mathsf{T}$, there is a unique choice of constants $c_1, \ldots, c_n$ for which

$$x = c_1 v^{(1)} + \cdots + c_n v^{(n)}$$

The constants are given by

$$c_i = \sum_{j=1}^{n} x_j v_j^{(i)} = x^\mathsf{T} v^{(i)}, \qquad 1 \le i \le n$$

where $v^{(i)} = [v_1^{(i)}, \ldots, v_n^{(i)}]^\mathsf{T}$.

(iv) Define the matrix U of order n by

$$U = [v^{(1)}, v^{(2)}, \ldots, v^{(n)}] \tag{8}$$

Then

$$U^\mathsf{T} A U = D \equiv \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \lambda_n \end{bmatrix}$$

and

$$UU^\mathsf{T} = U^\mathsf{T} U = I \tag{9}$$

$A = UDU^\mathsf{T}$ is a useful decomposition of A.

Example. Recall

$$A = \begin{bmatrix} 1.25 & 0.75 \\ 0.75 & 1.25 \end{bmatrix}$$

and its eigenvalue-eigenvector pairs

$$\lambda_1 = 2, \qquad v^{(1)} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\lambda_2 = 0.5, \qquad v^{(2)} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

Figure 1 illustrates that the eigenvectors are perpendicular. To have them be of length 1, replace the above by

$$\lambda_1 = 2, \qquad v^{(1)} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

$$\lambda_2 = 0.5, \qquad v^{(2)} = \begin{bmatrix} \frac{-1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

that are multiples of the original $v^{(i)}$.

The matrix U of (8) is given by

$$U = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$$

Easily, $U^{\mathsf{T}} U = I$.

Also,

$$\begin{aligned} & U^{\mathsf{T}} A U \\ &= \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 1.25 & 0.75 \\ 0.75 & 1.25 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \\ &= \begin{bmatrix} 2 & 0 \\ 0 & 0.5 \end{bmatrix} \end{aligned}$$

as specified in (9).

NONSYMMETRIC MATRICES

Nonsymmetric matrices have a wide variety of possible behaviour. We illustrate with two simple examples some of the possible behaviour.

Example. We illustrate the existence of complex eigenvalues. Let

$$A = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

The characteristic polynomial is

$$f(\lambda) = \det \begin{bmatrix} \lambda & -1 \\ 1 & \lambda \end{bmatrix} = \lambda^2 + 1$$

The roots of $f(\lambda)$ are complex,

$$\lambda_1 = i \equiv \sqrt{-1}, \qquad \lambda_2 = -i$$

and corresponding eigenvectors are

$$v^{(1)} = \begin{bmatrix} i \\ -1 \end{bmatrix}, \qquad v^{(2)} = \begin{bmatrix} i \\ 1 \end{bmatrix}$$

Example. For A an $n \times n$ nonsymmetric matrix, there may not be n independent eigenvectors. Let

$$A = \begin{bmatrix} a & 1 & 0 \\ 0 & a & 1 \\ 0 & 0 & a \end{bmatrix}$$

where a is a constant.

Then $\lambda = a$ is the eigenvalue of A with multiplicity 3, and any associated eigenvector must be of the form

$$v = c\,[1,\ 0,\ 0]^T$$

for some $c \neq 0$.

Thus, up to a nonzero multiplicative constant, we have only one eigenvector,

$$v = [1,\ 0,\ 0]^T$$

for the three equal eigenvalues $\lambda_1 = \lambda_2 = \lambda_3 = a$.

THE POWER METHOD

This numerical method is used primarily to find the eigenvalue of largest magnitude, if such exists.

We assume that the eigenvalues $\{\lambda_1, \ldots, \lambda_n\}$ of an $n \times n$ matrix A satisfy

$$|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n| \tag{10}$$

Denote the eigenvector for λ_1 by $v^{(1)}$. We define an iteration method for computing improved estimates of λ_1 and $v^{(1)}$.

Choose $z^{(0)} \approx v^{(1)}$, usually chosen randomly. Define

$$w^{(1)} = Az^{(0)}$$

Let α_1 be the maximum component of $w^{(1)}$, in size. If there is more than one such component, choose the first such component as α_1. Then define

$$z^{(1)} = \frac{1}{\alpha_1} w^{(1)}$$

Repeat the process iteratively. Define

$$w^{(m)} = Az^{(m-1)} \tag{11}$$

Let α_m be the maximum component of $w^{(m)}$, in size. Define

$$z^{(m)} = \frac{1}{\alpha_m} w^{(m)} \tag{12}$$

for $m = 1, 2, \ldots$ Then, roughly speaking, the vectors $z^{(m)}$ will converge to some multiple of $v^{(1)}$.

To find λ_1 by this process, also pick some nonzero component of the vectors $z^{(m)}$ and $w^{(m)}$, say component k; and fix k. Often this is picked as the maximal component of $z^{(l)}$, for some large l. Define

$$\lambda_1^{(m)} = \frac{w_k^{(m)}}{z_k^{(m-1)}}, \qquad m = 1, 2, \ldots \tag{13}$$

where $z_k^{(m-1)}$ denotes component k of $z^{(m-1)}$.

It can be shown that $\lambda_1^{(m)}$ converges to λ_1 as $m \to \infty$.

Example. Recall the earlier example $A = \begin{bmatrix} 1.25 & 0.75 \\ 0.75 & 1.25 \end{bmatrix}$. Double precision was used in the computation, with rounded values shown in the table given here.

Table 1: Power method example

m	$z_1^{(m)}$	$z_2^{(m)}$	$\lambda_1^{(m)}$	$\lambda_1^{(m)} - \lambda_1^{(m-1)}$	*Ratio*
0	1.0	.5			
1	1.0	.84615	1.62500		
2	1.0	.95918	1.88462	$2.60E-1$	
3	1.0	.98964	1.96939	$8.48E-2$	0.33
4	1.0	.99740	1.99223	$2.28E-2$	0.27
5	1.0	.99935	1.99805	$5.82E-3$	0.26
6	1.0	.99984	1.99951	$1.46E-3$	0.25
7	1.0	.99996	1.99988	$3.66E-4$	0.25

Note that in this example, $\lambda_1 = 2$ and $v^{(1)} = [1, 1]^T$, and the numerical results are converging to these values.

The column of successive differences $\lambda_1^{(m)} - \lambda_1^{(m-1)}$ and their successive ratios are included to show that there is a regular pattern to the convergence.

CONVERGENCE OF $\lambda_1^{(m)}$

A similar convergence analysis can be given for $\left\{\lambda_1^{(m)}\right\}$, with the same kind of error bound.

Moreover, if we also assume

$$|\lambda_1| > |\lambda_2| > |\lambda_3| \geq |\lambda_4| \geq \cdots \geq |\lambda_n| \geq 0 \qquad (19)$$

then

$$\lambda_1 - \lambda_1^{(m)} \approx c\left(\frac{\lambda_2}{\lambda_1}\right)^m \qquad (20)$$

for some constant c, as $m \to \infty$.

The error decreases geometrically with a ratio of λ_2/λ_1. In the earlier example with $A = \begin{bmatrix} 1.25 & 0.75 \\ 0.75 & 1.25 \end{bmatrix}$,

$$\lambda_2/\lambda_1 = 0.5/2 = .25$$

which is the ratio observed in Table 1.

Example. Consider the symmetric matrix

$$A = \begin{bmatrix} -7 & 13 & -16 \\ 13 & -10 & 13 \\ -16 & 13 & -7 \end{bmatrix}$$

From earlier, the eigenvalues are

$$\lambda_1 = -36, \qquad \lambda_2 = 9, \qquad \lambda_3 = 3$$

The eigenvector $v^{(1)}$ associated with λ_1 is

$$v^{(1)} = [1, -1, 1]^T$$

The results of using the power method are shown in the following Table 2.

Note that the ratios of the successive differences of $\lambda_1^{(m)}$ are approaching

$$\frac{\lambda_2}{\lambda_1} = -0.25$$

Also note that location of the maximal component of $z^{(m)}$ changes from one iteration to the next.

The initial guess $z^{(0)}$ was chosen closer to $v^{(1)}$ than would be usual in actual practice. It was so chosen for purposes of illustration.

ESTIMATION OF ERROR

Let $\widehat{x}$ denote an approximate solution for $Ax = b$; perhaps $\widehat{x}$ is obtained by Gaussian elimination. Let x denote the exact solution. Then introduce

$$r = b - A\widehat{x}$$

a quantity called the *residual* for $\widehat{x}$. Then

$$\begin{aligned} r &= b - A\widehat{x} \\ &= Ax - A\widehat{x} \\ &= A\left(x - \widehat{x}\right) \\ x - \widehat{x} &= A^{-1}r \end{aligned}$$

or the error $e = x - \widehat{x}$ is the exact solution of

$$Ae = r$$

Thus we can solve this to obtain an estimate $\widehat{e}$ of our error e.

EXAMPLE. Recall the linear system

$$\begin{array}{rcrcrl} .729x_1 & + & .81x_2 & + & .9x_3 & = .6867 \\ x_1 & + & x_2 & + & x_3 & = .8338 \\ 1.331x_1 & + & 1.21x_2 & + & 1.1x_3 & = 1.000 \end{array}$$

The true solution, rounded to four significant digits, is

$$x = [.2245, .2814, .3279]^{\mathsf{T}}$$

Using Gaussian elimination without pivoting and four digit decimal floating point arithmetic with rounding, the resulting solution and error are

$$\widehat{x} = [.2251, .2790, .3295]^{\mathsf{T}}$$
$$e = [-.0006, .0024, -.0016]^{\mathsf{T}}$$

Then

$$r = b - A\widehat{x} = [.00006210, .0002000, .0003519]^{\mathsf{T}}$$

Solving $Ae = r$ by Gaussian elimination, we obtain

$$e \approx \widehat{e} = [-.0004471, .002150, -.001504]^{\mathsf{T}}$$

THE RESIDUAL CORRECTION METHOD

If in the above we had taken $\widehat{e}$ and added it to $\widehat{x}$, then we would have obtained an improved answer:

$$x \approx \widehat{x} + \widehat{e} = [.2247, .2811, .3280]^{\mathsf{T}}$$

Recall

$$x = [.2245, .2814, .3279]^{\mathsf{T}}$$

With the new approximation, we can repeat the earlier process of estimating the error and then using it to improve the answer. This iterative process is called the *residual correction method*. It is illustrated with another example on page 286 in the text.

ERROR ANALYSIS

Begin with a simple example. The system

$$\begin{array}{rcrcr} 7x & + & 10y & = & 1 \\ 5x & + & 7y & = & .7 \end{array}$$

has the solution

$$x = 0, \quad y = .1$$

The perturbed system

$$\begin{array}{rcrcr} 7\widehat{x} & + & 10\widehat{y} & = & 1.01 \\ 5\widehat{x} & + & 7\widehat{y} & = & .69 \end{array}$$

has the solution

$$\widehat{x} = -.17, \quad \widehat{y} = .22$$

Why is there such a difference?

Consider the following Hilbert matrix example.

$$H_3 = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{bmatrix}, \quad \widetilde{H}_3 = \begin{bmatrix} 1.000 & .5000 & .3333 \\ .5000 & .3333 & .2500 \\ .3333 & .2500 & .2000 \end{bmatrix}$$

$$H_3^{-1} = \begin{bmatrix} 9 & -36 & 30 \\ -36 & 192 & -180 \\ 30 & -180 & 180 \end{bmatrix}$$

$$\widetilde{H}_3^{-1} = \begin{bmatrix} 9.062 & -36.32 & 30.30 \\ -36.32 & 193.7 & -181.6 \\ 30.30 & -181.6 & 181.5 \end{bmatrix}$$

We have changed H_3 in the fifth decimal place (by rounding the fractions to four decimal digits). But we have ended with a change in H_3^{-1} in the third decimal place.

VECTOR NORMS

A *norm* is a generalization of the absolute value function, and we use it to measure the "size" of a vector. There are a variety of ways of defining the norm of a vector, with each definition tied to certain applications.

Euclidean norm : Let x be a column vector with n components. Define

$$\|x\|_2 = \left[\sum_{j=1}^{n} \left| x_j \right|^2 \right]^{\frac{1}{2}}$$

This is the standard definition of the length of a vector, giving us the "straight line" distance between head and tail of the vector.

1-norm : Let x be a column vector with n components. Define

$$\|x\|_1 = \sum_{j=1}^{n} \left|x_j\right|$$

For planar applications ($n = 2$), this is sometimes called the "taxi cab norm", as it corresponds to distance as measured when driving in a city laid out with a rectangular grid of streets.

∞-*norm* : Let x be a column vector with n components. Define

$$\|x\|_\infty = \max_{1 \le j \le n} \left|x_j\right|$$

This is also called the *maximum norm* and the *Chebyshev norm*. It is often used in numerical analysis where we want to measure the maximum error component in some vector quantity.

EXAMPLES

Let

$$x = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}^{\mathsf{T}}$$

Then

$$\begin{aligned} \|x\|_1 &= 6 \\ \|x\|_2 &= \text{sqrt}(14) \doteq 3.74 \\ \|x\|_\infty &= 3 \end{aligned}$$

PROPERTIES

Let $\|\cdot\|$ denote a generic norm. Then:

(a) $\|x\| = 0$ if and only if $x = 0$.
(b) $\|cx\| = |c|\,\|x\|$ for any vector x and constant c.
(c) $\|x + y\| \leq \|x\| + \|y\|$, for all vectors x and y.

MATRIX NORMS

We also need to measure the sizes of general matrices, and we need to have some way of relating the sizes of A and x to the size of Ax. In doing this, we will consider only square matrices A.

We say a matrix norm is a way of defining the size of a matrix, again satisfying the properties seen with vector norms. Thus:

1. $\|A\| = 0$ if and only if $A = 0$.

2. $\|cA\| = |c|\,\|A\|$ for any matrix A and constant c.

3. $\|A + B\| \leq \|A\| + \|B\|$, for all matrices A and B of equal order.

In addition, we can multiply matrices, forming AB from A and B. With absolute values, we have $|ab| = |a|\,|b|$ for all complex numbers a and b. There is no way of generalizing exactly this relation to matrices A and B. But we can obtain definitions for which

$$\text{(d)} \qquad \|AB\| \leq \|A\|\,\|B\|$$

Finally, if we are given some vector norm $\|\cdot\|_v$, we can obtain an associated matrix norm definition for which

$$\text{(e)} \qquad \|Ax\|_v \leq \|A\|\,\|x\|_v$$

for all $n \times n$ matrices A and $n \times 1$ vectors x.

Often we use as our definition of $\|A\|$ the smallest number for which this last inequality is satisfied for all vectors x. In that case, we also obtain the useful property

$$\|I\| = 1$$

Let the vector norm be $\|\cdot\|_\infty$ for $n \times 1$ vectors x. Then the associated matrix norm definition is

$$\|A\| = \max_{1 \le i \le n} \sum_{j=1}^{n} \left| a_{i,j} \right|$$

This is sometimes called the "row norm" of a matrix A.

EXAMPLE. Let

$$A = \begin{bmatrix} 1 & 2 \\ 5 & 7 \end{bmatrix}, \quad z = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \quad Az = \begin{bmatrix} -1 \\ -2 \end{bmatrix}$$

Then

$$\|A\| = 12, \quad \|z\|_\infty = 1, \quad \|Az\|_\infty = 2$$

and clearly $\|Az\|_\infty \le \|A\| \, \|z\|_\infty$. Also let $z = \begin{bmatrix} 1, & 1 \end{bmatrix}^{\mathsf{T}}$. Then

$$Az = \begin{bmatrix} 3 \\ 12 \end{bmatrix}, \quad \|z\|_\infty = 1, \quad \|Az\|_\infty = 12$$

and $\|Az\|_\infty = \|A\| \, \|z\|_\infty$.

Let the vector norm be $\|\cdot\|_1$. Then the associated matrix norm definition is

$$\|A\| = \max_{1 \le j \le n} \sum_{i=1}^{n} \left| a_{i,j} \right|$$

This is sometimes called the "column norm" of the matrix A.

Let the vector norm be $\|\cdot\|_2$. Then the associated matrix norm definition is

$$\|A\| = \text{sqrt}\left[r_\sigma(A^\mathsf{T} A) \right]$$

To understand this, let B denote an arbitrary square matrix of order $n \times n$. Then introduce

$$\sigma(B) = \{\lambda \text{ an eigenvalue of } B\}$$

$$r_\sigma(B) = \max_{\lambda \in \sigma(B)} |\lambda|$$

The set $\sigma(B)$ is called the spectrum of B, and it contains all the eigenvalues of B. The number $r_\sigma(B)$ is called the "spectral radius" of B. There are easily computable bounds for $\|A\|$, but the norm itself is difficult to compute.

ERROR BOUNDS

Let $Ax = b$ and $A\widehat{x} = \widehat{b}$, and we are interested in cases with $b \approx \widehat{b}$. Then

$$\frac{\|x - \widehat{x}\|_v}{\|x\|_v} \leq \|A\| \left\|A^{-1}\right\| \frac{\left\|b - \widehat{b}\right\|_v}{\|b\|_v}$$

where $\|\cdot\|_v$ is some vector norm and $\|\cdot\|$ is an associated matrix norm.

Proof:

$$\begin{aligned}
Ax - A\widehat{x} &= b - \widehat{b} \\
A\left(x - \widehat{x}\right) &= b - \widehat{b} \\
x - \widehat{x} &= A^{-1}\left(b - \widehat{b}\right) \\
\|x - \widehat{x}\|_v &= \left\|A^{-1}\left(b - \widehat{b}\right)\right\|_v \\
&\leq \left\|A^{-1}\right\| \left\|b - \widehat{b}\right\|_v \\
\frac{\|x - \widehat{x}\|_v}{\|x\|_v} &\leq \frac{\left\|A^{-1}\right\| \left\|b - \widehat{b}\right\|_v}{\|x\|_v}
\end{aligned}$$

Rewrite this as

$$\frac{\|x-\widehat{x}\|_v}{\|x\|_v} \leq \|A\| \left\|A^{-1}\right\| \frac{\left\|b-\widehat{b}\right\|_v}{\|A\| \, \|x\|_v}$$

Since $Ax = b$, we have

$$\|b\|_v = \|Ax\|_v \leq \|A\| \, \|x\|_v$$

Using this,

$$\frac{\|x-\widehat{x}\|_v}{\|x\|_v} \leq \|A\| \left\|A^{-1}\right\| \frac{\left\|b-\widehat{b}\right\|_v}{\|b\|_v}$$

This completes the proof of the earlier assertion.

The quantity

$$\text{cond}(A) = \|A\| \left\|A^{-1}\right\|$$

is called a <u>condition number</u> for the matrix A.

EXAMPLE. Recall the earlier example:

$$\begin{array}{rcrcr} 7x_1 & + & 10x_2 & = & 1 \\ 5x_1 & + & 7x_2 & = & .7 \end{array} \qquad \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ .1 \end{bmatrix}$$

$$\begin{array}{rcrcr} 7\widehat{x}_1 & + & 10\widehat{x}_2 & = & 1.01 \\ 5\widehat{x}_1 & + & 7\widehat{x}_2 & = & .69 \end{array} \qquad \begin{bmatrix} \widehat{x}_1 \\ \widehat{x}_2 \end{bmatrix} = \begin{bmatrix} -.17 \\ .22 \end{bmatrix}$$

Then

$$\|b\|_\infty = 1, \quad \left\|b - \widehat{b}\right\|_\infty = .01$$

$$\|x\|_\infty = .1, \quad \|x - \widehat{x}\|_\infty = .17$$

$$A = \begin{bmatrix} 7 & 10 \\ 5 & 7 \end{bmatrix}, \qquad A^{-1} = \begin{bmatrix} -7 & 10 \\ 5 & -7 \end{bmatrix}$$

$$\|A\| = 17, \qquad \left\|A^{-1}\right\| = 17, \qquad \text{cond}(A) = 289$$

$$\frac{\|x - \widehat{x}\|_\infty}{\|x\|_\infty} \div \frac{\left\|b - \widehat{b}\right\|_\infty}{\|b\|_\infty} = \frac{1.7}{.01} = 170 \leq \text{cond}(A)$$

$$\frac{\|x - \widehat{x}\|_\infty}{\|x\|_\infty} \leq \text{cond}(A) \frac{\left\|b - \widehat{b}\right\|_\infty}{\|b\|_\infty}$$

The result

$$\frac{\|x-\widehat{x}\|_v}{\|x\|_v} \leq \text{cond}(A)\frac{\left\|b-\widehat{b}\right\|_v}{\|b\|_v}$$

has another aspect which we do not prove here. Given any matrix A, then there is a vector b and a nearby perturbation $\widehat{b}$ for which the above inequality can be replaced by equality. Moreover, there is no simple way to know of these b and $\widehat{b}$ in advance. For such b and $\widehat{b}$, we have

$$\text{cond}(A) = \frac{\|x-\widehat{x}\|_v}{\|x\|_v} \div \frac{\left\|b-\widehat{b}\right\|_v}{\|b\|_v}$$

Thus if cond(A) is very large, say 10^8, then there are b and $\widehat{b}$ for which

$$\frac{\|x-\widehat{x}\|_v}{\|x\|_v} = 10^8 \cdot \frac{\left\|b-\widehat{b}\right\|_v}{\|b\|_v}$$

We call such systems *ill-conditioned*.

Recall an earlier discussion of error in Gaussian elimination. Let $\widehat{x}$ denote an approximate solution for $Ax = b$; perhaps $\widehat{x}$ is obtained by Gaussian elimination. Let x denote the exact solution. Then introduce the *residual*

$$r = b - A\widehat{x}$$

We then obtained $x - \widehat{x} = A^{-1}r$. But we could also have discussed this as a special case of our present results. Write

$$Ax = b \quad \text{and} \quad A\widehat{x} = b - r \equiv \widehat{b}$$

Then

$$\frac{\|x - \widehat{x}\|_v}{\|x\|_v} \leq \text{cond}(A)\frac{\left\|b - \widehat{b}\right\|_v}{\|b\|_v}$$

becomes

$$\frac{\|x - \widehat{x}\|_v}{\|x\|_v} \leq \text{cond}(A)\frac{\|r\|_v}{\|b\|_v}$$

ILL-CONDITIONED EXAMPLE

Define the 4×4 Hilbert matrix:

$$H_4 = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \end{bmatrix}$$

Its inverse is given by

$$H_4^{-1} = \begin{bmatrix} 16 & -120 & 240 & -140 \\ -120 & 1200 & -2700 & 1680 \\ 240 & -2700 & 6480 & -4200 \\ -140 & 1680 & -4200 & 2800 \end{bmatrix}$$

For the matrix row norm,

$$\text{cond}(H_4) = \frac{25}{12} \cdot 13620 = 28375$$

Thus rounding error in defining b should lead to errors in solving $H_4 x = b$ that are larger than the rounding errors by a factor of 10^4 or more.

SOLVING LINEAR SYSTEMS

We want to solve the linear system

$$\begin{aligned} a_{1,1}x_1 + \cdots + a_{1,n}x_n &= b_1 \\ &\vdots \\ a_{n,1}x_1 + \cdots + a_{n,n}x_n &= b_n \end{aligned}$$

This will be done by the method used in beginning algebra, by successively eliminating unknowns from equations, until eventually we have only one equation in one unknown. This process is known as *Gaussian elimination*. To put it onto a computer, however, we must be more precise than is generally the case in high school algebra.

We begin with the linear system

$$\begin{aligned} 3x_1 - 2x_2 - x_3 &= 0 && (E1) \\ 6x_1 - 2x_2 + 2x_3 &= 6 && (E2) \\ -9x_1 + 7x_2 + x_3 &= -1 && (E3) \end{aligned}$$

$$\begin{array}{rcrl} 3x_1 - 2x_2 - x_3 & = & 0 & (E1) \\ 6x_1 - 2x_2 + 2x_3 & = & 6 & (E2) \\ -9x_1 + 7x_2 + x_3 & = & -1 & (E3) \end{array}$$

[1] Eliminate x_1 from equations $(E2)$ and $(E3)$. Subtract 2 times $(E1)$ from $(E2)$; and subtract -3 times $(E1)$ from $(E3)$. This yields

$$\begin{array}{rcrl} 3x_1 - 2x_2 - x_3 & = & 0 & (E1) \\ 2x_2 + 4x_3 & = & 6 & (E2) \\ x_2 - 2x_3 & = & -1 & (E3) \end{array}$$

[2] Eliminate x_2 from equation $(E3)$. Subtract $\frac{1}{2}$ times $(E2)$ from $(E3)$. This yields

$$\begin{array}{rcrl} 3x_1 - 2x_2 - x_3 & = & 0 & (E1) \\ 2x_2 + 4x_3 & = & 6 & (E2) \\ -4x_3 & = & -4 & (E3) \end{array}$$

Using back substitution, solve for x_3, x_2, and x_1, obtaining

$$x_3 = x_2 = x_1 = 1$$

In the computer, we work on the arrays rather than on the equations. To illustrate this, we repeat the preceding example using array notation.

The original system is $Ax = b$, with

$$A = \begin{bmatrix} 3 & -2 & -1 \\ 6 & -2 & 2 \\ -9 & 7 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} 0 \\ 6 \\ -1 \end{bmatrix}$$

We often write these in combined form as an *augmented* matrix:

$$[A \mid b] = \left[\begin{array}{rrr|r} 3 & -2 & -1 & 0 \\ 6 & -2 & 2 & 6 \\ -9 & 7 & 1 & -1 \end{array}\right]$$

In step 1, we eliminate x_1 from equations 2 and 3. We multiply row 1 by 2 and subtract it from row 2; and we multiply row 1 by -3 and subtract it from row 3. This yields

$$\left[\begin{array}{rrr|r} 3 & -2 & -1 & 0 \\ 0 & 2 & 4 & 6 \\ 0 & 1 & -2 & -1 \end{array}\right]$$

$$\left[\begin{array}{rrr|r} 3 & -2 & -1 & 0 \\ 0 & 2 & 4 & 6 \\ 0 & 1 & -2 & -1 \end{array}\right]$$

In step 2, we eliminate x_2 from equation 3. We multiply row 2 by $\frac{1}{2}$ and subtract from row 3. This yields

$$\left[\begin{array}{rrr|r} 3 & -2 & -1 & 0 \\ 0 & 2 & 4 & 6 \\ 0 & 0 & -4 & -4 \end{array}\right]$$

Then we proceed with back substitution as previously.

For the general case, we reduce

$$[A \mid b] = \left[\begin{array}{ccc|c} a_{1,1}^{(1)} & \cdots & a_{1,n}^{(1)} & b_1^{(1)} \\ \vdots & \ddots & \vdots & \vdots \\ a_{n,1}^{(1)} & \cdots & a_{n,n}^{(1)} & b_n^{(1)} \end{array}\right]$$

in $n-1$ steps to the form

$$\left[\begin{array}{cccc|c} a_{1,1}^{(1)} & \cdots & & a_{1,n}^{(1)} & b_1^{(1)} \\ 0 & \ddots & & \vdots & \vdots \\ \vdots & & \ddots & & \vdots \\ 0 & \cdots & 0 & a_{n,n}^{(n)} & b_n^{(n)} \end{array}\right]$$

More simply, and introducing new notation, this is equivalent to the matrix-vector equation $Ux = g$:

$$\begin{bmatrix} u_{1,1} & \cdots & & u_{1,n} \\ 0 & \ddots & & \vdots \\ \vdots & & \ddots & \\ 0 & \cdots & 0 & u_{n,n} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} g_1 \\ \vdots \\ \vdots \\ g_n \end{bmatrix}$$

This is the linear system

$$\begin{aligned} u_{1,1}x_1 + u_{1,2}x_2 + \cdots + u_{1,n-1}x_{n-1} + u_{1,n}x_n &= g_1 \\ &\vdots \\ u_{n-1,n-1}x_{n-1} + u_{n-1,n}x_n &= g_{n-1} \\ u_{n,n}x_n &= g_n \end{aligned}$$

We solve for x_n, then x_{n-1}, and backwards to x_1. This process is called *back substitution*.

$$x_n = \frac{g_n}{u_{n,n}}$$

$$u_k = \frac{g_k - \left\{u_{k,k+1}x_{k+1} + \cdots + u_{k,n}x_n\right\}}{u_{k,k}}$$

for $k = n-1, ..., 1$. What we have done here is simply a more carefully defined and methodical version of what you have done in high school algebra.

How do we carry out the conversion of

$$\left[\begin{array}{ccc|c} a_{1,1}^{(1)} & \cdots & a_{1,n}^{(1)} & b_1^{(1)} \\ \vdots & \ddots & \vdots & \vdots \\ a_{n,1}^{(1)} & \cdots & a_{n,n}^{(1)} & b_n^{(1)} \end{array}\right]$$

to

$$\left[\begin{array}{cccc|c} a_{1,1}^{(1)} & \cdots & & a_{1,n}^{(1)} & b_1^{(1)} \\ 0 & \ddots & & \vdots & \vdots \\ \vdots & & \ddots & & \vdots \\ 0 & \cdots & 0 & a_{n,n}^{(n)} & b_n^{(n)} \end{array}\right]$$

To help us keep track of the steps of this process, we will denote the initial system by

$$[A^{(1)} \mid b^{(1)}] = \left[\begin{array}{ccc|c} \underline{a_{1,1}^{(1)}} & \cdots & a_{1,n}^{(1)} & b_1^{(1)} \\ \vdots & \ddots & \vdots & \vdots \\ a_{n,1}^{(1)} & \cdots & a_{n,n}^{(1)} & b_n^{(1)} \end{array}\right]$$

Initially we will make the assumption that every *pivot element* will be nonzero; and later we remove this assumption.

Step 1. We will eliminate x_1 from equations 2 thru n. Begin by defining the *multipliers*

$$m_{i,1} = \frac{a_{i,1}^{(1)}}{a_{1,1}^{(1)}}, \quad i = 2, ..., n$$

Here we are assuming the *pivot element* $a_{1,1}^{(1)} \neq 0$. Then in succession, multiply $m_{i,1}$ times row 1 (called the *pivot row*) and subtract the result from row i. This yields new matrix elements

$$a_{i,j}^{(2)} = a_{i,j}^{(1)} - m_{i,1} a_{1,j}^{(1)}, \quad j = 2, ..., n$$

$$b_i^{(2)} = b_i^{(1)} - m_{i,1} b_1^{(1)}$$

for $i = 2, ..., n$.

Note that the index j does not include $j = 1$. The reason is that with the definition of the multiplier $m_{i,1}$, it is automatic that

$$a_{i,1}^{(2)} = a_{i,1}^{(1)} - m_{i,1} a_{1,1}^{(1)} = 0, \quad i = 2, ..., n$$

The augmented matrix now is

$$[A^{(2)} \mid b^{(2)}] = \left[\begin{array}{cccc|c} a_{1,1}^{(1)} & a_{1,2}^{(1)} & \cdots & a_{1,n}^{(1)} & b_1^{(1)} \\ 0 & \underline{a_{2,2}^{(2)}} & & a_{2,n}^{(2)} & b_2^{(2)} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & a_{n,2}^{(2)} & \cdots & a_{n,n}^{(2)} & b_n^{(2)} \end{array}\right]$$

Step k: Assume that for $i = 1, ..., k-1$ the unknown x_i has been eliminated from equations $i+1$ thru n. We have the augmented matrix

$$[A^{(k)} \mid b^{(k)}] = \left[\begin{array}{cccccc|c} a_{1,1}^{(1)} & a_{1,2}^{(1)} & & \cdots & & a_{1,n}^{(1)} & b_1^{(1)} \\ 0 & a_{2,2}^{(2)} & & \cdots & & a_{2,n}^{(2)} & b_2^{(2)} \\ & \ddots & \ddots & & & \vdots & \vdots \\ \vdots & & 0 & \underline{a_{k,k}^{(k)}} & \cdots & a_{k,n}^{(k)} & b_k^{(k)} \\ & & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & a_{n,k}^{(k)} & \cdots & a_{n,n}^{(k)} & b_n^{(k)} \end{array}\right]$$

We want to eliminate unknown x_k from equations $k+1$ thru n. Begin by defining the multipliers

$$m_{i,k} = \frac{a_{i,k}^{(k)}}{a_{k,k}^{(k)}}, \quad i = k+1, ..., n$$

The pivot element is $a_{k,k}^{(k)}$, and we assume it is nonzero. Using these multipliers, we eliminate x_k from equations $k+1$ thru n. Multiply $m_{i,k}$ times row k (the *pivot row*) and subtract from row i, for $i = k+1$ thru n.

$$a_{i,j}^{(k+1)} = a_{i,j}^{(k)} - m_{i,k} a_{k,j}^{(k)}, \quad j = k+1, ..., n$$

$$b_i^{(k+1)} = b_i^{(k)} - m_{i,k} b_k^{(k)}$$

for $i = k+1, ..., n$. This yields the augmented matrix

$[A^{(k+1)} \mid b^{(k+1)}]$:

$$\left[\begin{array}{cccccc|c}
a_{1,1}^{(1)} & & & \cdots & & a_{1,n}^{(1)} & b_1^{(1)} \\
0 & \ddots & & & & \vdots & \vdots \\
 & & a_{k,k}^{(k)} & a_{k,k+1}^{(k)} & \cdots & a_{k,n}^{(k)} & b_k^{(k)} \\
\vdots & & 0 & \underline{a_{k+1,k+1}^{(k+1)}} & & a_{k+1,n}^{(k+1)} & b_{k+1}^{(k+1)} \\
 & & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & \cdots & 0 & a_{n,k+1}^{(k+1)} & \cdots & a_{n,n}^{(k+1)} & b_n^{(k+1)}
\end{array}\right]$$

Doing this for $k = 1, 2, ..., n-1$ leads to the upper triangular system with the augmented matrix

$$\left[\begin{array}{cccc|c}
a_{1,1}^{(1)} & \cdots & & a_{1,n}^{(1)} & b_1^{(1)} \\
0 & \ddots & & \vdots & \vdots \\
\vdots & & \ddots & & \vdots \\
0 & \cdots & 0 & a_{n,n}^{(n)} & b_n^{(n)}
\end{array}\right]$$

We later remove the assumption

$$a_{k,k}^{(k)} \neq 0, \qquad k = 1, 2, ..., n$$

QUESTIONS

- How do we remove the assumption on the pivot elements?

- How many operations are involved in this procedure?

- How much error is there in the computed solution due to rounding errors in the calculations?

- How does the machine architecture affect the implementation of this algorithm.

PARTIAL PIVOTING

Recall the reduction of

$$[A^{(1)} \mid b^{(1)}] = \left[\begin{array}{ccc|c} a_{1,1}^{(1)} & \cdots & a_{1,n}^{(1)} & b_1^{(1)} \\ \vdots & \ddots & \vdots & \vdots \\ a_{n,1}^{(1)} & \cdots & a_{n,n}^{(1)} & b_n^{(1)} \end{array}\right]$$

to

$$[A^{(2)} \mid b^{(2)}] = \left[\begin{array}{cccc|c} a_{1,1}^{(1)} & a_{1,2}^{(1)} & \cdots & a_{1,n}^{(1)} & b_1^{(1)} \\ 0 & a_{2,2}^{(2)} & & a_{2,n}^{(2)} & b_2^{(2)} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & a_{n,2}^{(2)} & \cdots & a_{n,n}^{(2)} & b_n^{(2)} \end{array}\right]$$

What if $a_{1,1}^{(1)} = 0$? In that case we look for an equation in which the x_1 is present. To do this in such a way as to avoid zero the maximum extant possible, we do the following.

Look at all the elements in the first column,

$$a_{1,1}^{(1)}, a_{2,1}^{(1)}, ..., a_{n,1}^{(1)}$$

and pick the largest in size. Say it is

$$\left|a_{k,1}^{(1)}\right| = \max_{j=1,\ldots,n} \left|a_{j,1}^{(1)}\right|$$

Then interchange equations 1 and k, which means interchanging rows 1 and k in the augmented matrix $[A^{(1)} \mid b^{(1)}]$. Then proceed with the elimination of x_1 from equations 2 thru n as before.

Having obtained

$$[A^{(2)} \mid b^{(2)}] = \left[\begin{array}{cccc|c} a_{1,1}^{(1)} & a_{1,2}^{(1)} & \cdots & a_{1,n}^{(1)} & b_1^{(1)} \\ 0 & a_{2,2}^{(2)} & & a_{2,n}^{(2)} & b_2^{(2)} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & a_{n,2}^{(2)} & \cdots & a_{n,n}^{(2)} & b_n^{(2)} \end{array}\right]$$

what if $a_{2,2}^{(2)} = 0$? Then we proceed as before.

Among the elements

$$a_{2,2}^{(2)}, a_{3,2}^{(2)}, ..., a_{n,2}^{(2)}$$

pick the one of largest size:

$$\left|a_{k,2}^{(2)}\right| = \max_{j=2,\ldots,n} \left|a_{j,2}^{(2)}\right|$$

Interchange rows 2 and k. Then proceed as before to eliminate x_2 from equations 3 thru n, thus obtaining

$$[A^{(3)} \mid b^{(3)}] = \left[\begin{array}{ccccc|c} a_{1,1}^{(1)} & a_{1,2}^{(1)} & a_{1,3}^{(1)} & \cdots & a_{1,n}^{(1)} & b_1^{(1)} \\ 0 & a_{2,2}^{(2)} & a_{2,3}^{(2)} & \cdots & a_{2,n}^{(2)} & b_2^{(2)} \\ 0 & 0 & a_{3,3}^{(3)} & \cdots & a_{3,n}^{(3)} & b_3^{(3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & a_{n,3}^{(3)} & \cdots & a_{n,n}^{(3)} & b_n^{(3)} \end{array}\right]$$

This is done at every stage of the elimination process. This technique is called *partial pivoting*, and it is a part of most Gaussian elimination programs (including the one in the text).

Consequences of partial pivoting. Recall the definition of the elements obtained in the process of eliminating x_1 from equations 2 thru n.

$$m_{i,1} = \frac{a_{i,1}^{(1)}}{a_{1,1}^{(1)}}, \quad i = 2, ..., n$$

$$\begin{aligned} a_{i,j}^{(2)} &= a_{i,j}^{(1)} - m_{i,1} a_{1,j}^{(1)}, \quad j = 2, ..., n \\ b_i^{(2)} &= b_i^{(1)} - m_{i,1} b_1^{(1)} \end{aligned}$$

for $i = 2, ..., n$. By our definition of the pivot element $a_{1,1}^{(1)}$, we have

$$\left| m_{i,1} \right| \leq 1, \qquad i = 2, ..., n$$

Thus in the calculation of $a_{i,j}^{(2)}$ and $b_i^{(2)}$, we have that the elements do not grow rapidly in size. This is in comparison to what might happen otherwise, in which the multipliers $m_{i,1}$ might have been very large. This property is true of the multipliers at very step of the elimination process:

$$\left| m_{i,k} \right| \leq 1, \qquad i = k+1, ..., n, \quad k = 1, ..., n-1$$

The property

$$\left|m_{i,k}\right| \leq 1, \qquad i = k+1, ..., n$$

leads to good error propagation properties in Gaussian elimination with partial pivoting. The only error in Gaussian elimination is that derived from the rounding errors in the arithmetic operations. For example, at the first elimination step (eliminating x_1 from equations 2 thru n),

$$\begin{aligned} a_{i,j}^{(2)} &= a_{i,j}^{(1)} - m_{i,1} a_{1,j}^{(1)}, \quad j = 2, ..., n \\ b_i^{(2)} &= b_i^{(1)} - m_{i,1} b_1^{(1)} \end{aligned}$$

The above property on the size of the multipliers prevents these numbers and the errors in their calculation from growing as rapidly as they might if no partial pivoting was used.

As an example of the improvement in accuracy obtained with partial pivoting, see the example on pages 262-263.

OPERATION COUNTS

One of the major ways in which we compare the efficiency of different numerical methods is to count the number of needed arithmetic operations. For solving the linear system

$$\begin{aligned} a_{1,1}x_1 + \cdots + a_{1,n}x_n &= b_1 \\ &\vdots \\ a_{n,1}x_1 + \cdots + a_{n,n}x_n &= b_n \end{aligned}$$

using Gaussian elimination, we have the following operation counts.

1. $A \to U$, where we are converting $Ax = b$ to $Ux = g$:

Divisions	$\dfrac{n(n-1)}{2}$
Additions	$\dfrac{n(n-1)(2n-1)}{6}$
Multiplications	$\dfrac{n(n-1)(2n-1)}{6}$

2. $b \to g$:

$$\begin{array}{cc} \text{Additions} & \dfrac{n(n-1)}{2} \\ \text{Multiplications} & \dfrac{n(n-1)}{2} \end{array}$$

3. Solving $Ux = g$:

$$\begin{array}{cc} \text{Divisions} & n \\ \text{Additions} & \dfrac{n(n-1)}{2} \\ \text{Multiplications} & \dfrac{n(n-1)}{2} \end{array}$$

On some machines, the cost of a division is much more than that of a multiplication; whereas on others there is not any important difference. We assume the latter; and then the operation costs are as follows.

$$\begin{aligned} MD(A \to U) &= \frac{n\left(n^2-1\right)}{3} \\ MD(b \to g) &= \frac{n(n-1)}{2} \\ MD(\text{Find } x) &= \frac{n(n+1)}{2} \end{aligned}$$

$$
\begin{aligned}
AS(A \to U) &= \frac{n(n-1)(2n-1)}{6} \\
AS(b \to g) &= \frac{n(n-1)}{2} \\
AS(\text{Find } x) &= \frac{n(n-1)}{2}
\end{aligned}
$$

Thus the total number of operations is

$$
\begin{array}{cc}
\text{Additions} & \dfrac{2n^3 + 3n^2 - 5n}{6} \\
\left(\begin{array}{c} \text{Multiplications} \\ \text{and Divisions} \end{array} \right) & \dfrac{n^3 + 3n^2 - n}{3}
\end{array}
$$

Both are around $\frac{1}{3}n^3$, and thus the total operations account is approximately

$$\frac{2}{3}n^3$$

What happens to the cost when n is doubled?

Solving $Ax = b$ **and** $Ax = c$. What is the cost? Only the modification of the right side is different in these two cases. Thus the additional cost is

$$\begin{pmatrix} MD(b \to g) \\ MD(\text{Find } x) \end{pmatrix} = n^2$$

$$\begin{pmatrix} AS(b \to g) \\ AS(\text{Find } x) \end{pmatrix} = n(n-1)$$

The total is around $2n^2$ operations, which is quite a bit smaller than $\frac{2}{3}n^3$ when n is even moderately large, say $n = 100$.

Thus one can solve the linear system $Ax = c$ at little additional cost to that for solving $Ax = b$. This has important consequences when it comes to estimation of the error in computed solutions.

CALCULATING THE MATRIX INVERSE

Consider finding the inverse of a 3×3 matrix

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} = \Big[A_{*,1}, A_{*,2}, A_{*,3}\Big]$$

We want to find a matrix

$$X = \Big[X_{*,1}, X_{*,2}, X_{*,3}\Big]$$

for which

$$AX = I$$

$$A\Big[X_{*,1}, X_{*,2}, X_{*,3}\Big] = [e_1, e_2, e_3]$$

$$\Big[AX_{*,1}, AX_{*,2}, AX_{*,3}\Big] = [e_1, e_2, e_3]$$

This means we want to solve

$$AX_{*,1} = e_1, \quad AX_{*,2} = e_2, \quad AX_{*,3} = e_3$$

We want to solve three linear systems, all with the same matrix of coefficients A.

In augmented matrix notation, we want to work with

$$[A \mid I]$$

$$\left[\begin{array}{ccc|ccc} a_{1,1} & a_{1,2} & a_{1,3} & 1 & 0 & 0 \\ a_{2,1} & a_{2,2} & a_{2,3} & 0 & 1 & 0 \\ a_{3,1} & a_{3,2} & a_{3,3} & 0 & 0 & 1 \end{array}\right]$$

Then we proceed as before with a single linear system, only now we have three right hand sides. We first introduce zeros into positions 2 and 3 of column 1; and then we introduce zero into position 3 of column 2. Following that, we will need to solve three upper triangular linear systems by back substitution. See the numerical example on pages 264-266.

MATRIX INVERSE EXAMPLE

$$A = \begin{bmatrix} 1 & 1 & -2 \\ 1 & 1 & 1 \\ 1 & -1 & 0 \end{bmatrix}$$

$$\left[\begin{array}{rrr|rrr} 1 & 1 & -2 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & -1 & 0 & 0 & 0 & 1 \end{array}\right]$$

$$m_{2,1} = 1 \downarrow m_{3,1} = 1$$

$$\left[\begin{array}{rrr|rrr} 1 & 1 & -2 & 1 & 0 & 0 \\ 0 & 0 & 3 & -1 & 1 & 0 \\ 0 & -2 & 2 & -1 & 0 & 1 \end{array}\right]$$

$$\downarrow$$

$$\left[\begin{array}{rrr|rrr} 1 & 1 & -2 & 1 & 0 & 0 \\ 0 & -2 & 2 & -1 & 0 & 1 \\ 0 & 0 & 3 & -1 & 1 & 0 \end{array}\right]$$

$$\left[\begin{array}{rrr|rrr} 1 & 1 & -2 & 1 & 0 & 0 \\ 0 & -2 & 2 & -1 & 0 & 1 \\ 0 & 0 & 3 & -1 & 1 & 0 \end{array}\right]$$

Then by using back substitution to solve for each column of the inverse, we obtain

$$A^{-1} = \begin{bmatrix} \frac{1}{6} & \frac{1}{3} & \frac{1}{2} \\ \frac{1}{6} & \frac{1}{3} & -\frac{1}{2} \\ -\frac{1}{3} & \frac{1}{3} & 0 \end{bmatrix}$$

COST OF MATRIX INVERSION

In calculating A^{-1}, we are solving for the matrix $X = \left[X_{*,1}, X_{*,2}, \ldots, X_{*,n}\right]$ where

$$A\left[X_{*,1}, X_{*,2}, \ldots, X_{*,n}\right] = [e_1, e_2, \ldots, e_n]$$

and e_j is column j of the identity matrix. Thus we are solving n linear systems

$$AX_{*,1} = e_1, \quad AX_{*,2} = e_2, \ldots, \quad AX_{*,n} = e_n \quad (1)$$

all with the same coefficient matrix. Returning to the earlier operation counts for solving a single linear system, we have the following.

Cost of triangulating A: approx. $\frac{2}{3}n^3$ operations
Cost of solving $Ax = b$: $2n^2$ operations

Thus solving the n linear systems in (1) costs approximately

$$\frac{2}{3}n^3 + n\left(2n^2\right) = \frac{8}{3}n^3 \quad \text{operations, approximately}$$

It costs approximately four times as many operations to invert A as to solve a single system. With attention to the form of the right-hand sides in (1) this can be reduced to $2n^3$ operations.

MATLAB MATRIX OPERATIONS

To solve the linear system $Ax = b$ in Matlab, use

$$x = A \backslash b$$

In Matlab, the command

$$\text{inv}\,(A)$$

will calculate the inverse of A.

There are many matrix operations built into Matlab, both for general matrices and for special classes of matrices. We do not discuss those here, but recommend the student to investigate these thru the Matlab help options.

GAUSSIAN ELIMINATION - REVISITED

Consider solving the linear system

$$\begin{array}{rcl} 2x_1 + x_2 - x_3 + 2x_4 & = & 5 \\ 4x_1 + 5x_2 - 3x_3 + 6x_4 & = & 9 \\ -2x_1 + 5x_2 - 2x_3 + 6x_4 & = & 4 \\ 4x_1 + 11x_2 - 4x_3 + 8x_4 & = & 2 \end{array}$$

by Gaussian elimination **without** pivoting. We denote this linear system by $Ax = b$. The augmented matrix for this system is

$$[A \mid b] = \left[\begin{array}{rrrr|r} 2 & 1 & -1 & 2 & 5 \\ 4 & 5 & -3 & 6 & 9 \\ -2 & 5 & -2 & 6 & 4 \\ 4 & 11 & -4 & 8 & 2 \end{array}\right]$$

To eliminate x_1 from equations 2, 3, and 4, use multipliers

$$m_{2,1} = 2, \quad m_{3,1} = -1, \quad m_{4,1} = 2$$

To eliminate x_1 from equations 2, 3, and 4, use multipliers

$$m_{2,1} = 2, \quad m_{3,1} = -1, \quad m_{4,1} = 2$$

This will introduce zeros into the positions below the diagonal in column 1, yielding

$$\left[\begin{array}{cccc|r} 2 & 1 & -1 & 2 & 5 \\ 0 & 3 & -1 & 2 & -1 \\ 0 & 6 & -3 & 8 & 9 \\ 0 & 9 & -2 & 4 & -8 \end{array}\right]$$

To eliminate x_2 from equations 3 and 4, use multipliers

$$m_{3,2} = 2, \quad m_{4,2} = 3$$

This reduces the augmented matrix to

$$\left[\begin{array}{cccr|r} 2 & 1 & -1 & 2 & 5 \\ 0 & 3 & -1 & 2 & -1 \\ 0 & 0 & -1 & 4 & 11 \\ 0 & 0 & 1 & -2 & -5 \end{array}\right]$$

To eliminate x_3 from equation 4, use the multiplier

$$m_{4,3} = -1$$

This reduces the augmented matrix to

$$\left[\begin{array}{cccc|r} 2 & 1 & -1 & 2 & 5 \\ 0 & 3 & -1 & 2 & -1 \\ 0 & 0 & -1 & 4 & 11 \\ 0 & 0 & 0 & 2 & 6 \end{array}\right]$$

Return this to the familiar linear system

$$\begin{aligned} 2x_1 + x_2 - x_3 + 2x_4 &= 5 \\ 3x_2 - x_3 + 2x_4 &= -1 \\ -x_3 + 4x_4 &= 11 \\ 2x_4 &= 6 \end{aligned}$$

Solving by back substitution, we obtain

$$x_4 = 3, \quad x_3 = 1, \quad x_2 = -2, \quad x_1 = 1$$

There is a surprising result involving matrices associated with this elimination process. Introduce the upper triangular matrix

$$U = \begin{bmatrix} 2 & 1 & -1 & 2 \\ 0 & 3 & -1 & 2 \\ 0 & 0 & -1 & 4 \\ 0 & 0 & 0 & 2 \end{bmatrix}$$

which resulted from the elimination process. Then introduce the lower triangular matrix

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ m_{2,1} & 1 & 0 & 0 \\ m_{3,1} & m_{3,2} & 1 & 0 \\ m_{4,1} & m_{4,2} & m_{4,3} & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ -1 & 2 & 1 & 0 \\ 2 & 3 & -1 & 1 \end{bmatrix}$$

This uses the multipliers introduced in the elimination process. Then

$$A = LU$$

$$\begin{bmatrix} 2 & 1 & -1 & 2 \\ 4 & 5 & -3 & 6 \\ -2 & 5 & -2 & 6 \\ 4 & 11 & -4 & 8 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ -1 & 2 & 1 & 0 \\ 2 & 3 & -1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & -1 & 2 \\ 0 & 3 & -1 & 2 \\ 0 & 0 & -1 & 4 \\ 0 & 0 & 0 & 2 \end{bmatrix}$$

In general, when the process of Gaussian elimination without pivoting is applied to solving a linear system $Ax = b$, we obtain $A = LU$ with L and U constructed as above.

For the case in which partial pivoting is used, we obtain the slightly modified result

$$LU = PA$$

where L and U are constructed as before and P is a *permutation matrix*. For example, consider

$$P = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

Then

$$PA = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \end{bmatrix} = \begin{bmatrix} A_{3,*} \\ A_{1,*} \\ A_{4,*} \\ A_{2,*} \end{bmatrix}$$

$$
\begin{aligned}
PA &= \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \end{bmatrix} \\
&= \begin{bmatrix} A_{3,*} \\ A_{1,*} \\ A_{4,*} \\ A_{2,*} \end{bmatrix}
\end{aligned}
$$

The matrix PA is obtained from A by switching around rows of A. The result $LU = PA$ means that the LU-factorization is valid for the matrix A with its rows suitably permuted.

Consequences: If we have a factorization

$$A = LU$$

with L lower triangular and U upper triangular, then we can solve the linear system $Ax = b$ in a relatively straightforward way.

The linear system can be written as

$$LUx = b$$

Write this as a two stage process:

$$Lg = b, \qquad Ux = g$$

The system $Lg = b$ is a lower triangular system

$$\begin{array}{ll} g_1 & = \; b_1 \\ \ell_{2,1} g_1 + g_2 & = \; b_2 \\ \ell_{3,1} g_1 + \ell_{3,2} g_2 + g_3 & = \; b_3 \\ & \vdots \\ \ell_{n,1} g_1 + \cdots \ell_{n,n-1} g_{n-1} + g_n & = \; b_n \end{array}$$

We solve it by "forward substitution". Then we solve the upper triangular system $Ux = g$ by back substitution.

VARIANTS OF GAUSSIAN ELIMINATION

If no partial pivoting is needed, then we can look for a factorization

$$A = LU$$

without going thru the Gaussian elimination process.

For example, suppose A is 4×4. We write

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ \ell_{2,1} & 1 & 0 & 0 \\ \ell_{3,1} & \ell_{3,2} & 1 & 0 \\ \ell_{4,1} & \ell_{4,2} & \ell_{4,3} & 1 \end{bmatrix} \begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & u_{1,4} \\ 0 & u_{2,2} & u_{2,3} & u_{2,4} \\ 0 & 0 & u_{3,3} & u_{3,4} \\ 0 & 0 & 0 & u_{4,4} \end{bmatrix}$$

To find the elements $\left\{\ell_{i,j}\right\}$ and $\left\{u_{i,j}\right\}$, we multiply the right side matrices L and U and match the results with the corresponding elements in A.

Multiplying the first row of L times all of the columns of U leads to

$$u_{1,j} = a_{1,j}, \quad j = 1, 2, 3, 4$$

Then multiplying rows 2, 3, 4 times the first column of U yields

$$\ell_{i,1} u_{1,1} = a_{i,1}, \quad i = 2, 3, 4$$

and we can solve for $\left\{\ell_{2,1}, \ell_{3,1}, \ell_{4,1}\right\}$. We can continue this process, finding the second row of U and then the second column of L, and so on. For example, to solve for $\ell_{4,3}$, we need to solve for it in

$$\ell_{4,1} u_{1,3} + \ell_{4,2} u_{2,3} + \ell_{4,3} u_{3,3} = a_{4,3}$$

Why do this? A hint of an answer is given by this last equation. If we had an $n \times n$ matrix A, then we would find $\ell_{n,n-1}$ by solving for it in the equation

$$\ell_{n,1} u_{1,n-1} + \ell_{n,2} u_{2,n-1} + \cdots + \ell_{n,n-1} u_{n-1,n-1} = a_{n,n-1}$$

$$\ell_{n,n-1} = \frac{a_{n,n-1} - \left[\ell_{n,1} u_{1,n-1} + \cdots + \ell_{n,n-2} u_{n-2,n-1}\right]}{u_{n-1,n-1}}$$

Embedded in this formula we have a dot product. This is in fact typical of this process, with the length of the inner products varying from one position to another.

Recalling §2.4 and the discussion of dot products, we can evaluate this last formula by using a higher precision arithmetic and thus avoid many rounding errors. This leads to a variant of Gaussian elimination in which there are far fewer rounding errors.

With ordinary Gaussian elimination, the number of rounding errors is proportional to n^3. This reduces the number of rounding errors, with the number now being proportional to only n^2. This can lead to major increases in accuracy, especially for matrices A which are very sensitive to small changes.

TRIDIAGONAL MATRICES

$$A = \begin{bmatrix} b_1 & c_1 & 0 & 0 & \cdots & 0 \\ a_2 & b_2 & c_2 & 0 & & \\ 0 & a_3 & b_3 & c_3 & & \vdots \\ & & & \ddots & & \\ \vdots & & & a_{n-1} & b_{n-1} & c_{n-1} \\ 0 & & \cdots & & a_n & b_n \end{bmatrix}$$

These occur very commonly in the numerical solution of partial differential equations, as well as in other applications (e.g. computing interpolating cubic spline functions).

We factor $A = LU$, as before. But now L and U take very simple forms. Before proceeding, we note with an example that the same may not be true of the matrix inverse.

EXAMPLE

Define an $n \times n$ tridiagonal matrix

$$A = \begin{bmatrix} -1 & 1 & 0 & 0 & \cdots & 0 \\ 1 & -2 & 1 & 0 & & \\ 0 & 1 & -2 & 1 & & \vdots \\ & & & \ddots & & \\ \vdots & & & 1 & -2 & 1 \\ 0 & & \cdots & & 1 & -\frac{n-1}{n} \end{bmatrix}$$

Then A^{-1} is given by

$$\left(A^{-1}\right)_{i,j} = \max\{i, j\}$$

Thus the sparse matrix A can (and usually does) have a dense inverse.

We factor $A = LU$, with

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ \alpha_2 & 1 & 0 & 0 & & \\ 0 & \alpha_3 & 1 & 0 & & \vdots \\ & & & \ddots & & \\ \vdots & & & \alpha_{n-1} & 1 & 0 \\ 0 & & \cdots & & \alpha_n & 1 \end{bmatrix}$$

$$U = \begin{bmatrix} \beta_1 & c_1 & 0 & 0 & \cdots & 0 \\ 0 & \beta_2 & c_2 & 0 & & \\ 0 & 0 & \beta_3 & c_3 & & \vdots \\ & & & \ddots & & \\ \vdots & & & 0 & \beta_{n-1} & c_{n-1} \\ 0 & & \cdots & & 0 & \beta_n \end{bmatrix}$$

Multiply these and match coefficients with A to find $\{\alpha_i, \gamma_i\}$.

By doing a few multiplications of rows of L times columns of U, we obtain the general pattern as follows.

$$\begin{aligned}
\beta_1 &= b_1 && \text{: row 1 of } LU \\
\alpha_2\beta_1 = a_2, \quad \alpha_2 c_1 + \beta_2 &= b_2 && \text{: row 2 of } LU \\
&\vdots \\
\alpha_n\beta_{n-1} = a_n, \quad \alpha_n c_{n-1} + \beta_n &= b_n && \text{: row } n \text{ of } LU
\end{aligned}$$

These are straightforward to solve.

$$\beta_1 = b_1$$

$$\alpha_j = \frac{a_j}{\beta_{j-1}}, \quad \beta_j = b_j - \alpha_j c_{j-1}, \quad j = 2, ..., n$$

To solve the linear system

$$Ax = f$$

or

$$LUx = f$$

instead solve the two triangular systems

$$Lg = f, \qquad Ux = g$$

Solving $Lg = f$:

$$\begin{aligned} g_1 &= f_1 \\ g_j &= f_j - \alpha_j g_{j-1}, \quad j = 2, ..., n \end{aligned}$$

Solving $Ux = g$:

$$\begin{aligned} x_n &= \frac{g_n}{\beta_n} \\ x_j &= \frac{g_j - c_j x_{j+1}}{\beta_j}, \quad j = n-1, ..., 1 \end{aligned}$$

See the numerical example on page 278.

OPERATIONS COUNT

Factoring $A = LU$.

Additions:	$n-1$
Multiplications:	$n-1$
Divisions:	$n-1$

Solving $Lz = f$ and $Ux = z$:

Additions:	$2n-2$
Multiplications:	$2n-2$
Divisions:	n

Thus the total number of arithmetic operations is approximately $3n$ to factor A; and it takes about $5n$ to solve the linear system using the factorization of A.

If we had A^{-1} at no cost, what would it cost to compute $x = A^{-1}f$?

$$x_i = \sum_{j=1}^{n} \left(A^{-1}\right)_{i,j} f_j, \quad i = 1, ..., n$$

MATLAB MATRIX OPERATIONS

To obtain the *LU*-factorization of a matrix, including the use of partial pivoting, use the Matlab command *lu*. In particular,

$$[L,\ U,\ P] = lu(X)$$

returns the lower triangular matrix L, upper triangular matrix U, and permutation matrix P so that

$$PX = LU$$

LINEAR SYSTEMS

Consider the following example of a linear system:

$$\begin{aligned} x_1 + 2x_2 + 3x_3 &= -5 \\ -x_1 + x_3 &= -3 \\ 3x_1 + x_2 + 3x_3 &= -3 \end{aligned}$$

Its unique solution is

$$x_1 = 1, \quad x_2 = 0, \quad x_3 = -2$$

In general we want to solve n equations in n unknowns. For this, we need some simplifying notation. In particular we introduce *arrays*. We can think of these as means for storing information about the linear system in a computer. In the above case, we introduce

$$A = \begin{bmatrix} 1 & 2 & 3 \\ -1 & 0 & 1 \\ 3 & 1 & 3 \end{bmatrix}, \quad b = \begin{bmatrix} -5 \\ -3 \\ -3 \end{bmatrix}, \quad x = \begin{bmatrix} 1 \\ 0 \\ -2 \end{bmatrix}$$

These arrays completely specify the linear system and its solution. We also know that we can give meaning to multiplication and addition of these quantities, calling them *matrices* and *vectors*. The linear system is then written as

$$Ax = b$$

with Ax denoting a matrix-vector multiplication.

The general system is written as

$$\begin{aligned} a_{1,1}x_1 + \cdots + a_{1,n}x_n &= b_1 \\ &\vdots \\ a_{n,1}x_1 + \cdots + a_{n,n}x_n &= b_n \end{aligned}$$

This is a system of n linear equations in the n unknowns $x_1, ..., x_n$. This can be written in matrix-vector notation as

$$Ax = b$$

$$A = \begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,n} \end{bmatrix}, \; b = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix} \; x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

A TRIDIAGONAL SYSTEM

Consider the *tridiagonal* linear system

$$\begin{array}{rcl} 3x_1 - x_2 & = & 2 \\ -x_1 + 3x_2 - x_3 & = & 1 \\ & \vdots & \\ -x_{n-2} + 3x_{n-1} - x_n & = & 1 \\ -x_{n-1} + 3x_n & = & 2 \end{array}$$

The solution is

$$x_1 = \cdots = x_n = 1$$

This has the associated arrays

$$A = \begin{bmatrix} 3 & -1 & 0 & \cdots & 0 \\ -1 & 3 & -1 & 0 & \\ & & \ddots & & \\ \vdots & & -1 & 3 & -1 \\ 0 & \cdots & & -1 & 3 \end{bmatrix}, \; b = \begin{bmatrix} 2 \\ 1 \\ \vdots \\ 1 \\ 2 \end{bmatrix}, \; x = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{bmatrix}$$

SOLVING LINEAR SYSTEMS

Linear systems $Ax = b$ occur widely in applied mathematics. They occur as direct formulations of "real world" problems; but more often, they occur as a part of the numerical analysis of some other problem. As examples of the latter, we have the construction of spline functions, the numerical solution of systems of nonlinear equations, ordinary and partial differential equations, integral equations, and the solution of optimization problems.

There are many ways of classifying linear systems.

Size: Small, moderate, and large. This of course varies with the machine you are using. Most PCs are now being sold with a memory of 256 to 512 megabytes (MB), and my old HP workstation has 768 MB (it is over four years old).

For a matrix A of order $n \times n$, it will take $8n^2$ bytes to store it in double precision. Thus a matrix of order 8000 will need around 512 MB of storage. The latter would be too large for most present day PCs, if the matrix was to be stored in the computer's memory, although one can easily expand a PC to contain much more memory than this.

Sparse vs. Dense. Many linear systems have a matrix A in which almost all the elements are zero. These matrices are said to be *sparse*. For example, it is quite common to work with tridiagonal matrices

$$A = \begin{bmatrix} a_1 & c_1 & 0 & \cdots & 0 \\ b_2 & a_2 & c_2 & 0 & \vdots \\ 0 & b_3 & a_3 & c_3 & \\ \vdots & & & \ddots & \\ 0 & \cdots & & b_n & a_n \end{bmatrix}$$

in which the order is 10^4 or much more. For such matrices, it does not make sense to store the zero elements; and the sparsity should be taken into account when solving the linear system $Ax = b$. Also, the sparsity need not be as regular as in this example.

BASIC DEFINITIONS & THEORY

A *homogeneous* linear system $Ax = b$ is one for which the right hand constants are all zero. Using vector notation, we say b is the zero vector for a homogeneous system. Otherwise the linear system is call *non-homogeneous*.

Theorem. The following are equivalent statements.
(1) For each b, there is exactly one solution x.
(2) For each b, there is a solution x.
(3) The homogeneous system $Ax = 0$ has only the solution $x = 0$.
(4) $\det(A) \neq 0$.
(5) A^{-1} exists. [The matrix inverse and determinant are introduced in §6.2, but they belong as a part of this theorem.]

EXAMPLE. Consider again the tridiagonal system

$$\begin{array}{rcl} 3x_1 - x_2 & = & 2 \\ -x_1 + 3x_2 - x_3 & = & 1 \\ & \vdots & \\ -x_{n-2} + 3x_{n-1} - x_n & = & 1 \\ -x_{n-1} + 3x_n & = & 2 \end{array}$$

The homogeneous version is simply

$$\begin{array}{rcl} 3x_1 - x_2 & = & 0 \\ -x_1 + 3x_2 - x_3 & = & 0 \\ & \vdots & \\ -x_{n-2} + 3x_{n-1} - x_n & = & 0 \\ -x_{n-1} + 3x_n & = & 0 \end{array}$$

Assume $x \neq 0$, and therefore that x has nonzero components. Let x_k denote a component of maximum size:

$$|x_k| = \max_{1 \leq j \leq n} \left| x_j \right|$$

Consider now equation k, and assume $1 < k < n$. Then

$$\begin{aligned}
-x_{k-1} + 3x_k - x_{k+1} &= 0 \\
x_k &= \tfrac{1}{3}\left(x_{k-1} + x_{k+1}\right) \\
|x_k| &\leq \tfrac{1}{3}\left(|x_{k-1}| + |x_{k+1}|\right) \\
&\leq \tfrac{1}{3}\left(|x_k| + |x_k|\right) \\
&= \tfrac{2}{3}\,|x_k|
\end{aligned}$$

This implies $x_k = 0$, and therefore $x = 0$. A similar proof is valid if $k = 1$ or $k = n$, using the first or the last equation, respectively.

Thus the original tridiagonal linear system $Ax = b$ has a unique solution x for each right side b.

METHODS OF SOLUTION

There are two general categories of numerical methods for solving $Ax = b$.

Direct Methods: These are methods with a finite number of steps; and they end with the exact solution x, provided that all arithmetic operations are exact. The most used of these methods is *Gaussian elimination*, which we begin with. There are other direct methods, but we do not study them here.

Iteration Methods: These are used in solving all types of linear systems, but they are most commonly used with large sparse systems, especially those produced by discretizing partial differential equations. This is an extremely active area of research.

MATRICES in MATLAB

Consider the matrices

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 2 & 3 \\ 3 & 3 & 3 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

In *MATLAB*, A can be created as follows.

```
A = [1   2   3;   2   2   3;   3   3   3];
A = [1,  2,  3;  2,  2,  3;  3,  3,  3];
A = [1   2   3
     2   2   3
     3   3   3] ;
```

Commas can be used to replace the spaces. The vector b can be created by

$$b = ones(3, 1);$$

Consider setting up the matrices for the system $Ax = b$ with

$$A_{i,j} = \max\{i, j\}, \qquad b_i = 1, \qquad 1 \le i, j \le n$$

One way to set up the matrix A is as follows:

```
A = zeros(n, n);
for  i = 1 : n
     A(i, 1 : i) = i;
     A(i, i + 1 : n) = i + 1 : n;
end
```

and set up the vector b by

$$b = ones(n, 1);$$

ITERATION METHODS

These are methods which compute a sequence of progressively accurate iterates to approximate the solution of $Ax = b$.

We need such methods for solving many large linear systems. Sometimes the matrix is too large to be stored in the computer memory, making a direct method too difficult to use.

More importantly, the operations cost of $\frac{2}{3}n^3$ for Gaussian elimination is too large for most large systems. With iteration methods, the cost can often be reduced to something of cost $O\left(n^2\right)$ or less. Even when a special form for A can be used to reduce the cost of elimination, iteration will often be faster.

There are other, more subtle, reasons, which we do not discuss here.

JACOBI'S ITERATION METHOD

We begin with an example. Consider the linear system

$$\begin{array}{rcrcrcl} 9x_1 & + & x_2 & + & x_3 & = & b_1 \\ 2x_1 & + & 10x_2 & + & 3x_3 & = & b_2 \\ 3x_1 & + & 4x_2 & + & 11x_3 & = & b_3 \end{array}$$

In equation $\#k$, solve for x_k:

$$\begin{aligned} x_1 &= \tfrac{1}{9}[b_1 - x_2 - x_3] \\ x_2 &= \tfrac{1}{10}[b_2 - 2x_1 - 3x_3] \\ x_3 &= \tfrac{1}{11}[b_3 - 3x_1 - 4x_2] \end{aligned}$$

Let $x^{(0)} = \left[x_1^{(0)}, x_2^{(0)}, x_3^{(0)}\right]^T$ be an initial guess to the solution x. Then define

$$\begin{aligned} x_1^{(k+1)} &= \tfrac{1}{9}\left[b_1 - x_2^{(k)} - x_3^{(k)}\right] \\ x_2^{(k+1)} &= \tfrac{1}{10}\left[b_2 - 2x_1^{(k)} - 3x_3^{(k)}\right] \\ x_3^{(k+1)} &= \tfrac{1}{11}\left[b_3 - 3x_1^{(k)} - 4x_2^{(k)}\right] \end{aligned}$$

for $k = 0, 1, 2, \ldots$. This is called the *Jacobi iteration method* or the *method of simultaneous replacements*.

NUMERICAL EXAMPLE.

Let $b = [10,\ 19,\ 0]^T$.

The solution is $x = [1,\ 2,\ -1]^T$.

To measure the error, we use

$$Error = \|x - x^{(k)}\| = \max_i \left| x_i - x_i^{(k)} \right|$$

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$	*Error*	*Ratio*
0	0	0	0	2.00E + 0	
1	1.1111	1.9000	0	1.00E + 0	0.500
2	0.9000	1.6778	−0.9939	3.22E − 1	0.322
3	1.0351	2.0182	−0.8556	1.44E − 1	0.448
4	0.9819	1.9496	−1.0162	5.06E − 2	0.349
5	1.0074	2.0085	−0.9768	2.32E − 2	0.462
6	0.9965	1.9915	−1.0051	8.45E − 3	0.364
7	1.0015	2.0022	−0.9960	4.03E − 3	0.477
8	0.9993	1.9985	−1.0012	1.51E − 3	0.375
9	1.0003	2.0005	−0.9993	7.40E − 4	0.489
10	0.9999	1.9997	−1.0003	2.83E − 4	0.382
30	1.0000	2.0000	−1.0000	3.01E − 11	0.447
31	1.0000	2.0000	−1.0000	1.35E − 11	0.447

GAUSS-SEIDEL ITERATION METHOD

Again consider the linear system

$$\begin{array}{rcrcrcl} 9x_1 & + & x_2 & + & x_3 & = & b_1 \\ 2x_1 & + & 10x_2 & + & 3x_3 & = & b_2 \\ 3x_1 & + & 4x_2 & + & 11x_3 & = & b_3 \end{array}$$

and solve for x_k in equation $\#k$:

$$\begin{aligned} x_1 &= \tfrac{1}{9}[b_1 - x_2 - x_3] \\ x_2 &= \tfrac{1}{10}[b_2 - 2x_1 - 3x_3] \\ x_3 &= \tfrac{1}{11}[b_3 - 3x_1 - 4x_2] \end{aligned}$$

Now immediately use every new iterate:

$$\begin{aligned} x_1^{(k+1)} &= \tfrac{1}{9}\left[b_1 - x_2^{(k)} - x_3^{(k)}\right] \\ x_2^{(k+1)} &= \tfrac{1}{10}\left[b_2 - 2x_1^{(k+1)} - 3x_3^{(k)}\right] \\ x_3^{(k+1)} &= \tfrac{1}{11}\left[b_3 - 3x_1^{(k+1)} - 4x_2^{(k+1)}\right] \end{aligned}$$

for $k = 0, 1, 2, \ldots$. This is called the *Gauss-Seidel iteration method* or the *method of successive replacements*.

NUMERICAL EXAMPLE.

Let $b = [10,\ 19,\ 0]^T$.

The solution is $x = [1,\ 2,\ -1]^T$.

To measure the error, we use

$$Error = \|x - x^{(k)}\| = \max_i \left| x_i - x_i^{(k)} \right|$$

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$	*Error*	*Ratio*
0	0	0	0	$2.00E+0$	
1	1.1111	1.6778	-0.9131	$3.22E-1$	0.161
2	1.0262	1.9687	-0.9958	$3.13E-2$	0.097
3	1.0030	1.9981	-1.0001	$3.00E-3$	0.096
4	1.0002	2.0000	-1.0001	$2.24E-4$	0.074
5	1.0000	2.0000	-1.0000	$1.65E-5$	0.074
6	1.0000	2.0000	-1.0000	$2.58E-6$	0.155

The values of *Ratio* do not approach a limiting value with larger values of the iteration index k.

A GENERAL SCHEMA

Rewrite $Ax = b$ as

$$Nx = b + Px \tag{1}$$

with $A = N - P$ a <u>*splitting*</u> of A. Choose N to be nonsingular. Usually we want $Nz = f$ to be easily solvable for arbitray f. The iteration method is

$$Nx^{(k+1)} = b + Px^{(k)}, \qquad k = 0, 1, 2, \ldots, \tag{2}$$

EXAMPLE. Let N be the diagonal of A, and let $P = N - A$. The iteration method is the Jacobi method:

$$a_{i,i}x_i^{(k+1)} = b_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{i,j}x_j^{(k)}, \quad 1 \leq i \leq n$$

for $k = 0, 1, \ldots$.

EXAMPLE. Let N be the lower triangular part of A, including its diagonal, and let $P = N - A$. The iteration method is the Gauss-Seidel method:

$$\sum_{j=1}^{i} a_{i,j} x_j^{(k+1)} = b_i - \sum_{j=i+1}^{n} a_{i,j} x_j^{(k)}, \quad 1 \le i \le n$$

for $k = 0, 1, \ldots$.

EXAMPLE. Another method could be defined by letting N be the tridiagonal matrix formed from the diagonal, super-diagonal, and sub-diagonal of A, with $P = N - A$:

$$N = \begin{bmatrix} a_{1,1} & a_{1,2} & 0 & \cdots & 0 \\ a_{2,1} & a_{2,2} & a_{2,3} & \ddots & \vdots \\ 0 & & \ddots & & 0 \\ \vdots & \ddots & a_{n-1,n-2} & a_{n-1,n-1} & a_{n-1,n} \\ 0 & \cdots & 0 & a_{n,n-1} & a_{n,n} \end{bmatrix}$$

Solving $Nx^{(k+1)} = b + Px^{(k)}$ uses the algorithm for tridiagonal systems from §6.4.

CONVERGENCE

When does the iteration method (2) converge? Subtract (2) from (1), obtaining

$$\begin{aligned} N\left(x - x^{(k+1)}\right) &= P\left(x - x^{(k)}\right) \\ x - x^{(k+1)} &= N^{-1}P\left(x - x^{(k)}\right) \\ e^{(k+1)} &= Me^{(k)}, \quad M = N^{-1}P \end{aligned} \tag{3}$$

with $e^{(k)} \equiv x - x^{(k)}$

Return now to the matrix and vector norms of §6.5. Then

$$\left\|e^{(k+1)}\right\| \leq \|M\| \left\|e^{(k)}\right\|, \qquad k \geq 0$$

Thus the error $e^{(k)}$ converges to zero if $\|M\| < 1$, with

$$\left\|e^{(k)}\right\| \leq \|M\|^k \left\|e^{(0)}\right\|, \qquad k \geq 0$$

EXAMPLE. For the earlier example with the Jacobi method,

$$
\begin{aligned}
x_1^{(k+1)} &= \tfrac{1}{9}\left[b_1 - x_2^{(k)} - x_3^{(k)}\right] \\
x_2^{(k+1)} &= \tfrac{1}{10}\left[b_2 - 2x_1^{(k)} - 3x_3^{(k)}\right] \\
x_3^{(k+1)} &= \tfrac{1}{11}\left[b_3 - 3x_1^{(k)} - 4x_2^{(k)}\right]
\end{aligned}
$$

$$
M = \begin{bmatrix} 0 & -\frac{1}{9} & -\frac{1}{9} \\ -\frac{2}{10} & 0 & -\frac{3}{10} \\ -\frac{3}{11} & -\frac{4}{11} & 0 \end{bmatrix}
$$

$$
\|M\| = \frac{7}{11} \doteq 0.636
$$

This is consistent with the earlier table of values, although the actual convergence rate was better than predicted by (3).

EXAMPLE. For the earlier example with the Gauss-Seidel method,

$$
\begin{aligned}
x_1^{(k+1)} &= \tfrac{1}{9}\left[b_1 - x_2^{(k)} - x_3^{(k)}\right] \\
x_2^{(k+1)} &= \tfrac{1}{10}\left[b_2 - 2x_1^{(k+1)} - 3x_3^{(k)}\right] \\
x_3^{(k+1)} &= \tfrac{1}{11}\left[b_3 - 3x_1^{(k+1)} - 4x_2^{(k+1)}\right]
\end{aligned}
$$

$$
\begin{aligned}
M &= \begin{bmatrix} 9 & 0 & 0 \\ 2 & 10 & 0 \\ 3 & 4 & 11 \end{bmatrix}^{-1} \begin{bmatrix} 0 & -1 & -1 \\ 0 & 0 & -3 \\ 0 & 0 & 0 \end{bmatrix} \\
&= \begin{bmatrix} 0 & -\frac{1}{9} & -\frac{1}{9} \\ 0 & \frac{1}{45} & -\frac{5}{18} \\ 0 & \frac{1}{45} & \frac{13}{99} \end{bmatrix} \\
\|M\| &= 0.3
\end{aligned}
$$

This too is consistent with the earlier numerical results.

DIAGONALLY DOMINANT MATRICES

Matrices A for which

$$\left|a_{i,i}\right| > \sum_{\substack{j=1 \\ j\neq i}}^{n} \left|a_{i,j}\right|, \qquad i = 1, \ldots, n$$

are called *diagonally dominant*. For the Jacobi iteration method,

$$M = \begin{bmatrix} 0 & -\dfrac{a_{1,2}}{a_{1,1}} & \cdots & -\dfrac{a_{1,n}}{a_{1,1}} \\ -\dfrac{a_{2,1}}{a_{2,2}} & 0 & & -\dfrac{a_{2,n}}{a_{2,2}} \\ \vdots & & \ddots & \vdots \\ -\dfrac{a_{n,1}}{a_{n,n}} & \cdots & -\dfrac{a_{n,n-1}}{a_{n,n}} & 0 \end{bmatrix}$$

With diagonally dominant matrices A,

$$\|M\| = \max_{1\leq i\leq n} \sum_{\substack{j=1 \\ j\neq i}}^{n} \left|\frac{a_{i,j}}{a_{i,i}}\right| < 1 \tag{4}$$

Thus the Jacobi iteration method for solving $Ax = b$ is convergent.

GAUSS-SEIDEL ITERATION

Assuming A is diagonally dominant, we can show that the Gauss-Seidel iteration will also converge. However, constructing $M = N^{-1}P$ is not reasonable for this method and an alternative approach is needed. Return to the error equation

$$Ne^{(k+1)} = Pe^{(k)}$$

and write it in component form for the Gauss-Seidel method:

$$\sum_{j=1}^{i} a_{i,j} e_j^{(k+1)} = -\sum_{j=i+1}^{n} a_{i,j} e_j^{(k)}, \quad 1 \le i \le n$$

$$e_i^{(k+1)} = -\sum_{j=1}^{i-1} \frac{a_{i,j}}{a_{i,i}} e_j^{(k+1)} - \sum_{j=i+1}^{n} \frac{a_{i,j}}{a_{i,i}} e_j^{(k)} \qquad (5)$$

Introduce

$$\alpha_i = \sum_{j=1}^{i-1} \left| \frac{a_{i,j}}{a_{i,i}} \right|, \qquad \beta_i = \sum_{j=i+1}^{n} \left| \frac{a_{i,j}}{a_{i,i}} \right|, \quad 1 \le i \le n$$

with $\alpha_1 = \beta_n = 0$. Taking bounds in (5),

$$\left| e_i^{(k+1)} \right| \le \alpha_i \left\| e^{(k+1)} \right\| + \beta_i \left\| e^{(k)} \right\|, \quad i = 1, ..., n \quad (6)$$

Let ℓ be an index for which

$$\left| e_\ell^{(k+1)} \right| = \max_{1 \le i \le n} \left| e_i^{(k+1)} \right| = \left\| e^{(k+1)} \right\|$$

Then using $i = \ell$ in (6),

$$\left\| e^{(k+1)} \right\| \le \alpha_\ell \left\| e^{(k+1)} \right\| + \beta_\ell \left\| e^{(k)} \right\|$$

$$\left\| e^{(k+1)} \right\| \le \frac{\beta_\ell}{1 - \alpha_\ell} \left\| e^{(k)} \right\|$$

Define

$$\eta = \max_i \frac{\beta_i}{1 - \alpha_i}$$

Then

$$\left\| e^{(k+1)} \right\| \le \eta \left\| e^{(k)} \right\|$$

For A diagonally dominant, it can be shown that

$$\eta \leq \|M\| \tag{7}$$

where $\|M\|$ is for the definition of M for the Jacobi method, given earlier in (4) as

$$\|M\| = \max_{1 \leq i \leq n} \sum_{\substack{j=1 \\ j \neq i}}^{n} \left| \frac{a_{i,j}}{a_{i,i}} \right| = \max_{1 \leq i \leq n} (\alpha_i + \beta_i) < 1$$

Consequently, for A diagonally dominant, the Gauss-Seidel method also converges and it does so more rapidly than the Jacobi method in most cases.

Showing (7) follows by showing

$$\frac{\beta_i}{1 - \alpha_i} - (\alpha_i + \beta_i) \geq 0, \qquad 1 \leq i \leq n$$

For our earlier example with A of order 3, we have $\mu = 0.375$ This is not as good as computing $\|M\|$ directly for the Gauss-Seidel method, but it does show that the rate of convergence is better than for the Jacobi method.

CONVERGENCE: AN ADDENDUM

Since

$$\|M\| = \|N^{-1}P\| \leq \|N^{-1}\| \, \|P\|,$$

$\|M\| < 1$ is satisfied if N satisfies

$$\|N^{-1}\| \, \|P\| < 1$$

Using $P = N - A$, this can be rewritten as

$$\|A - N\| < \frac{1}{\|N^{-1}\|}$$

We also want to choose N so that systems $Nz = f$ is 'easily solvable'.

GENERAL CONVERGENCE THEOREM:

$$Nx^{(k+1)} = b + Px^{(k)}, \qquad k = 0, 1, 2, \ldots,$$

will converge, for all right sides b and all initial guesses $x^{(0)}$, if and only if all eigenvalues λ of $M = N^{-1}P$ satisfy

$$|\lambda| < 1$$

This is the basis of deriving other splittings $A = N - P$ that lead to convergent iteration methods.

RESIDUAL CORRECTION METHODS

N be an invertible approximation of the matrix A; let $x^{(0)} \approx x$ for the solution of $Ax = b$. Define

$$r^{(0)} = b - Ax^{(0)}$$

Since $Ax = b$ for the true solution x,

$$\begin{aligned} r^{(0)} &= Ax - Ax^{(0)} \\ &= A(x - x^{(0)}) = Ae^{(0)} \end{aligned}$$

with $e^{(0)} = x - x^{(0)}$. Let $\hat{e}^{(0)}$ be the solution of

$$N\hat{e}^{(0)} = r^{(0)}$$

and then define

$$x^{(1)} = x^{(0)} + \hat{e}^{(0)}$$

Repeat this process inductively.

RESIDUAL CORRECTION

For $k = 0, 1, \ldots,$ define

$$\begin{aligned} r^{(k)} &= b - Ax^{(k)} \\ N\hat{e}^{(k)} &= r^{(k)} \\ x^{(k+1)} &= x^{(k)} + \hat{e}^{(k)} \end{aligned}$$

This is the general *residual correction method*.

To see how this fits into our earlier framework, proceed as follows:

$$\begin{aligned} x^{(k+1)} &= x^{(k)} + \hat{e}^{(k)} = x^{(k)} + N^{-1}r^{(k)} \\ &= x^{(k)} + N^{-1}(b - Ax^{(k)}) \end{aligned}$$

Thus,

$$\begin{aligned} Nx^{(k+1)} &= Nx^{(k)} + b - Ax^{(k)} \\ &= b + (N - A)x^{(k)} \\ &= b + Px^{(k)} \end{aligned}$$

Sometimes the residual correction scheme is a preferable way of approaching the development of an iterative method.

LEAST SQUARES DATA FITTING

Experiments generally have *error* or *uncertainty* in measuring their outcome. *Error* can be human error, but it is more usually due to inherent limitations in the equipment being used to make measurements. *Uncertainty* can be due to lack of precise definition or of human variation in what is being measured. (For example, how do you measure how much you like something?)

We often want to represent the experimental data by some functional expression. Interpolation is often unsatisfactory because it fails to account for the error or uncertainty in the data.

We may know from theory that the data is taken from a particular form of function (e.g. a quadratic polynomial), or we may choose to use some particular type of formula to represent the data.

EXAMPLE

Consider the following data.

Table 1: Empirical data

x_i	y_i	x_i	y_i
1.0	-1.945	3.2	0.764
1.2	-1.253	3.4	0.532
1.4	-1.140	3.6	1.073
1.6	-1.087	3.8	1.286
1.8	-0.760	4.0	1.502
2.0	-0.682	4.2	1.582
2.2	-0.424	4.4	1.993
2.4	-0.012	4.6	2.473
2.6	-0.190	4.8	2.503
2.8	0.452	5.0	2.322
3.0	0.337		

From the following Figure 1 it appears to be approximately linear.

Figure 1: The plot of empirical data

An experiment seeks to obtain an unknown functional relationship

$$y = f(x) \tag{1}$$

involving two related variables x and y. We choose varying values of x, say, $x_1, x_2, \ldots, x_n$. Then we measure a corresponding set of values for y. Let the actual measurements be denoted by $y_1, \ldots, y_n$, and let

$$\epsilon_i = f(x_i) - y_i$$

denote the unknown measurement errors. We want to use the points $(x_1, y_1), \ldots, (x_n, y_n)$ to determine the analytic relationship (1) as accurately as possible.

Often we suspect that the unknown function $f(x)$ lies within some known class of functions, for example, polynomials. Then we want to choose the member of that class of functions that will best approximate the unknown function $f(x)$, taking into account the experimental errors $\{\epsilon_i\}$.

As an example of such a situation, consider the data in Table 1 and the plot of it in Figure 1. From this plot, it is reasonable to expect $f(x)$ to be close to a linear polynomial,

$$f(x) = mx + b \tag{2}$$

Assuming this to be the true form, the problem of determining $f(x)$ is now reduced to that of determining the constants m and b.

We can choose to determine m and b in a number of ways. We list three such ways.

1. Choose m and b so as to minimize the quantity

$$\frac{1}{n}\sum_{i=1}^{n}|f(x_i)-y_i|$$

which can be considered an average approximation error.

2. Choose m and b so as to minimize the quantity

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}[f(x_i)-y_i]^2}$$

which can also be considered an average approximation error. It is called the *root mean square error* in the approximation of the data $\{(x_i, y_i)\}$ by the function $f(x)$.

3. Choose m and b so as to minimize the quantity

$$\max_{1\le i\le n}|f(x_i)-y_i|$$

which is the maximum error of approximation.

All of these can be used, but #**2** is the favorite, and we now comment on why. To do so we need to understand more about the nature of the unknown errors $\{\epsilon_i\}$.

Standard assumption: Each error ϵ_i is a random variable chosen from a *normal probability distribution*. Intuitively, such errors satisfy the following.
(1) If the experiment is repeated many times for the same $x = x_i$, then the associated unknown errors ϵ_i in the empirical values y_i will be likely to have an average of zero.
(2) For this same experimental case with $x = x_i$, as the size of ϵ_i increases, the likelihood of its occurring will decrease rapidly.

This is the *normal error assumption*. We also assume that the individual errors ϵ_i, $1 \leq i \leq n$, are all random variables from the same normal probability distribution function, meaning that the size of ϵ_i is unrelated to the size of x_i or y_i.

Assume $f(x)$ is in a known class of functions, call it C. An example is the assumption that $f(x)$ is linear for the data in Table 1.

Then among all functions $\widehat{f}(x)$ in C, it can be shown that the function $\widehat{f}^*$ that is most likely to equal f will also minimize the expression

$$E = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left[\widehat{f}(x_i) - y_i\right]^2} \tag{3}$$

among all functions $\widehat{f}$ in C.

This is called the *root-mean-square error* in the approximation of the data $\{y_i\}$ by $\widehat{f}(x)$. The function $\widehat{f}^*(x)$ that minimizes E relative to all $\widehat{f}$ in C is called the *least squares approximation* to the data $\{(x_i, y_i)\}$.

EXAMPLE

Return to the data in Table 1, pictured in Figure 1. The least squares approximation is given by

$$\widehat{f}^*(x) = 1.06338x - 2.74605 \tag{4}$$

It is illustrated graphically in Figure 2.

Figure 2: The linear least squares fit $\widehat{f}^*(x)$

CALCULATING THE LEAST SQUARES APPROXIMATION

How did we calculate $\widehat{f}^*(x)$? We want to minimize

$$E \equiv \sqrt{\frac{1}{n}\sum_{i=1}^{n}[f(x_i) - y_i]^2}$$

when considering all possible functions $f(x) = mx + b$. Note that minimizing E is equivalent to minimizing the sum, although the minimum values will be different. Thus we seek to minimize

$$G(b, m) = \sum_{i=1}^{n}[mx_i + b - y_i]^2 \tag{5}$$

as b and m are allowed to vary arbitrarily.

The choices of b and m that minimize $G(b, m)$ will satisfy

$$\frac{\partial G(b, m)}{\partial b} = 0, \qquad \frac{\partial G(b, m)}{\partial m} = 0 \tag{6}$$

Use

$$\frac{\partial G}{\partial b} = \sum_{i=1}^{n} 2\left[mx_i + b - y_i\right]$$

$$\frac{\partial G}{\partial m} = \sum_{i=1}^{n} 2\left[mx_i + b - y_i\right] x_i = \sum_{i=1}^{n} 2\left[mx_i^2 + bx_i - x_i y_i\right]$$

This leads to the linear system

$$\begin{aligned} nb + \left(\sum_{i=1}^{n} x_i\right) m &= \sum_{i=1}^{n} y_i \\ \left(\sum_{i=1}^{n} x_i\right) b + \left(\sum_{i=1}^{n} x_i^2\right) m &= \sum_{i=1}^{n} x_i y_i \end{aligned} \tag{7}$$

This is uniquely solvable if the determinant is nonzero,

$$n \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2 \neq 0 \tag{8}$$

This is true unless

$$x_1 = x_2 = \cdots = x_n = \text{constant}$$

and this is false for our case.

For our example in Table 1,

$$\sum_{i=1}^{n} x_i = 63.0 \qquad \sum_{i=1}^{n} x_i^2 = 219.8$$

$$\sum_{i=1}^{n} y_i = 9.326 \qquad \sum_{i=1}^{n} x_i y_i = 60.7302$$

Using this in (7), the linear system becomes

$$\begin{aligned} 21b + 63.0m &= 9.326 \\ 63.0b + 219.8m &= 60.7302 \end{aligned}$$

The solution is

$$b \doteq -2.74605 \qquad m \doteq 1.06338$$

$$\widehat{f}^*(x) = 1.06338x - 2.74605$$

The root-mean-square-error in $\widehat{f}^*(x)$ is

$$E \doteq 0.171$$

Recall the graph of $\widehat{f}^*(x)$ is given in Figure 2.

GENERALIZATION

To represent the data $\{(x_i, y_i) \mid 1 \le i \le n\}$, let

$$\widehat{f}(x) = a_1\varphi_1(x) + a_2\varphi_2(x) + \cdots + a_m\varphi_m(x) \quad (9)$$

$a_1, a_2, \ldots, a_m$ arbitrary numbers, $\varphi_1(x), \ldots, \varphi_m(x)$ given functions.

If $\widehat{f}(x)$ is to be a quadratic polynomial, write

$$\widehat{f}(x) = a_1 + a_2x + a_3x^2 \quad (10)$$

$$\varphi_1(x) \equiv 1, \qquad \varphi_2(x) = x, \qquad \varphi_3(x) = x^2$$

Under the normal error assumption, the function $\widehat{f}(x)$ is to be chosen to minimize the root-mean-square error

$$E = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left[\widehat{f}(x_i) - y_i\right]^2}$$

Consider $m = 3$. Then

$$\widehat{f}(x) = a_1\varphi_1(x) + a_2\varphi_2(x) + a_3\varphi_3(x)$$

Choose a_1, a_2, a_3 to minimize

$$G(a_1, a_2, a_3) = \sum_{j=1}^{n} [a_1\varphi_1(x_j) + a_2\varphi_2(x_j) + a_3\varphi_3(x_j) - y_j]^2$$

At the minimizing point (a_1, a_2, a_3),

$$\frac{\partial G}{\partial a_1} = 0, \qquad \frac{\partial G}{\partial a_2} = 0, \qquad \frac{\partial G}{\partial a_3} = 0$$

This leads to the three equations. For $i = 1, 2, 3$,

$$0 = \frac{\partial G}{\partial a_i} = \sum_{j=1}^{n} 2[a_1\varphi_1(x_j) + a_2\varphi_2(x_j) + a_3\varphi_3(x_j) - y_j]\varphi_i(x_j)$$

$$\left[\sum_{j=1}^{n} \varphi_1(x_j)\varphi_i(x_j)\right] a_1 + \left[\sum_{j=1}^{n} \varphi_2(x_j)\varphi_i(x_j)\right] a_2 + \left[\sum_{j=1}^{n} \varphi_3(x_j)\varphi_i(x_j)\right] a_3 = \sum_{j=1}^{n} y_j\varphi_i(x_j), \quad (11)$$

Apply this to the quadratic formula

$$\widehat{f}(x) = a_1 + a_2 x + a_3 x^2$$

$$\varphi_1(x) \equiv 1, \qquad \varphi_2(x) = x, \qquad \varphi_3(x) = x^2$$

Then the three equations are

$$\begin{aligned} n a_1 + \left[\sum_{j=1}^{n} x_j\right] a_2 + \left[\sum_{j=1}^{n} x_j^2\right] a_3 &= \sum_{j=1}^{n} y_j \\ \left[\sum_{j=1}^{n} x_j\right] a_1 + \left[\sum_{j=1}^{n} x_j^2\right] a_2 + \left[\sum_{j=1}^{n} x_j^3\right] a_3 &= \sum_{j=1}^{n} y_j x_j \\ \left[\sum_{j=1}^{n} x_j^2\right] a_1 + \left[\sum_{j=1}^{n} x_j^3\right] a_2 + \left[\sum_{j=1}^{n} x_j^4\right] a_3 &= \sum_{j=1}^{n} y_j x_j^2 \end{aligned} \tag{12}$$

This can be shown a nonsingular system due to the assumption that the points $\{x_j\}$ are distinct.

Generalization. Let

$$\widehat{f}(x) = a_1\varphi_1(x) + a_2\varphi_2(x) + \cdots + a_m\varphi_m(x)$$

The root-mean-square error E in (3) is minimized with the coefficients $a_1, \ldots, a_m$ satisfying

$$\sum_{k=1}^{m} a_k \left[\sum_{j=1}^{n} \varphi_k(x_j)\varphi_i(x_j)\right] = \sum_{j=1}^{n} y_j\varphi_i(x_j) \qquad (13)$$

for $i = 1, \ldots, m$. For the special case of a polynomial of degree $(m-1)$,

$$\widehat{f}(x) = a_1 + a_2x + a_3x^2 + \cdots + a_mx^{m-1}$$

write

$$\varphi_1(x) = 1, \quad \varphi_2(x) = x, \quad \varphi_3(x) = x^2, \quad \ldots, \varphi_m(x) = x^{m-1} \qquad (14)$$

System (13) becomes

$$\sum_{k=1}^{m} a_k \left[\sum_{j=1}^{n} x_j^{i+k-2}\right] = \sum_{j=1}^{n} y_j x_j^{i-1}, \qquad i = 1, 2, \ldots, m \qquad (15)$$

When $m = 3$, this yields the system (12) obtained earlier.

ILL-CONDITIONING

This system (15) is nonsingular (for $m < n$). Unfortunately it is increasingly ill-conditioned as the degree $m-1$ increases.

The condition number for the matrix of coefficients can be very large for fairly small values of m, say, $m = 4$.

For this reason, it is seldom advisable to use

$$\varphi_1(x) = 1, \quad \varphi_2(x) = x, \quad \varphi_3(x) = x^2, \\ \ldots, \varphi_m(x) = x^{m-1}$$

to do a least squares polynomial fit, except for degree ≤ 2.

To do a least squares fit to data $\{(x_i, y_i) \mid 1 \le i \le n\}$ with a higher degree polynomial $\widehat{f}(x)$, write

$$\widehat{f}(x) = a_1\varphi_1(x) + \cdots + a_m\varphi_m(x)$$

with $\varphi_1(x), \ldots, \varphi_m(x)$ so chosen that the matrix of coefficients in

$$\sum_{k=1}^{m} a_k \left[\sum_{j=1}^{n} \varphi_k(x_j)\varphi_i(x_j) \right] = \sum_{j=1}^{n} y_j \varphi_i(x_j)$$

is not ill-conditioned.

There are optimal choices of these functions $\varphi_j(x)$, with $\deg(\varphi_j) = j - 1$ and with the coefficient matrix becoming diagonal.

IMPROVED BASIS FUNCTIONS

A nonoptimal but still satisfactory choice in general can be based on the Chebyshev polynomials $\{T_k(x)\}$ of Section 5.5, and a somewhat better choice is the Legendre polynomials of Section 5.7.

Suppose that the nodes $\{x_i\}$ are chosen from an interval $[\alpha, \beta]$. Introduce modified Chebyshev polynomials

$$\varphi_k(x) = T_{k-1}\left(\frac{2x - \alpha - \beta}{\beta - \alpha}\right), \qquad \alpha \le x \le \beta, \qquad k \ge 1 \tag{16}$$

Then degree $(\varphi_k) = k - 1$; and any polynomial $\widehat{f}(x)$ of degree $(m - 1)$ can be written as a combination of $\varphi_1(x), \ldots, \varphi_m(x)$.

EXAMPLE

Consider the following data.

Table 2: Data for a cubic least squares fit

x_i	y_i	x_i	y_i
0.00	0.486	0.55	1.102
0.05	0.866	0.60	1.099
0.10	0.944	0.65	1.017
0.15	1.144	0.70	1.111
0.20	1.103	0.75	1.117
0.25	1.202	0.80	1.152
0.30	1.166	0.85	1.265
0.35	1.191	0.90	1.380
0.40	1.124	0.95	1.575
0.45	1.095	1.00	1.857
0.50	1.122		

From the following Figure 3 it appears to be approximately cubic. We begin by using

$$\widehat{f}(x) = a_1 + a_2 x + a_3 x^2 + a_4 x^3 \tag{17}$$

Figure 3: The plot of data of Table 2

The resulting linear system (15), denoted here by $La = b$, is given by

$$L = \begin{bmatrix} 21 & 10.5 & 7.175 & 5.5125 \\ 10.5 & 7.175 & 5.5125 & 4.51666 \\ 7.175 & 5.5125 & 4.51666 & 3.85416 \\ 5.5125 & 4.51666 & 3.85416 & 3.38212 \end{bmatrix}$$

$$a = [a_1, a_2, a_3, a_4]^T$$

$$b = [24.1180, 13.2345, 9.46836, 7.55944]^T$$

The solution is

$$a = [0.5747,\ 4.7259,\ -11.1282,\ 7.6687]^T$$

The condition number is

$$\text{cond}(L) = \|L\|\,\|L^{-1}\| \doteq 22000 \tag{18}$$

This is very large; it may be difficult to obtain an accurate answer for $La = b$.

To verify this, perturb b above by adding to it the perturbation

$$[0.01,\ -0.01,\ 0.01,\ -0.01]^T$$

This will change b in its second place to the right of the decimal point, within the range of possible perturbations due to errors in the data. The solution of the new perturbed system is

$$a = [0.7408,\ 2.6825,\ -6.1538,\ 4.4550]^T$$

This is very different from the earlier result for a.

The main point here is that use of

$$\widehat{f}(x) = a_1 + a_2 x + a_3 x^2 + a_4 x^3$$

leads to a rather ill-conditioned system of linear equations for determining $\{a_1, a_2, a_3, a_4\}$.

A better basis.

Use the modified Chebyshev functions of (16) on $[\alpha, \beta] = [0, 1]$:

$$f(x) = a_1\varphi_1(x) + a_2\varphi_2(x) + a_3\varphi_3(x) + a_4\varphi_4(x)$$

$$\begin{aligned}
\varphi_1(x) &= T_0(2x-1) \equiv 1 \\
\varphi_2(x) &= T_1(2x-1) = 2x - 1 \\
\varphi_3(x) &= T_2(2x-1) = 8x^2 - 8x + 1 \\
\varphi_4(x) &= T_3(2x-1) = 32x^3 - 48x^2 + 18x - 1
\end{aligned}$$

The values $\{a_1, a_2, a_3, a_4\}$ are completely different than in the representation (17).

The linear system (13) is again denoted by $La = b$:

$$L = \begin{bmatrix} 21 & 0 & -5.6 & 0 \\ 0 & 7.7 & 0 & -2.8336 \\ -5.6 & 0 & 10.4664 & 0 \\ 0 & -2.8336 & 0 & 11.01056 \end{bmatrix}$$

$$b = [24.118,\ 2.351,\ -6.01108,\ 1.523576]^T$$

The solution is

$$a = [1.160969,\ 0.393514,\ 0.046850,\ 0.239646]^T$$

The linear system is very stable with respect to the type of perturbation made in b with the earlier approach to the cubic least squares fit, using (17).

This is implied by the small condition number of L.

$$\text{cond}(L) = \|L\|\,\|L^{-1}\| \doteq (26.6)(0.1804) \doteq 4.8$$

Relatively small perturbations in b will lead to relatively small changes in the solution a.

The graph of $\widehat{f}(x)$ is shown in Figure 4.

Figure 4: The cubic least squares fit for Table 2

To give some idea of the accuracy of $\widehat{f}(x)$ in approximating the data in Table 2, we easily compute the root-mean-square error from (3) to be

$$E \doteq 0.0421$$

a fairly small value when compared with the function values of $\widehat{f}(x)$.

MATRICES

Matrices are rectangular arrays of real or complex numbers. With them, we define arithmetic operations that are generalizations of those for real and complex numbers. The general form a matrix of order $m \times n$ is

$$A = \begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \cdots & a_{m,n} \end{bmatrix}$$

We say it has *order* $m \times n$. Matrices that consist of a single column are called *column vectors*, and those consisting of a single row are called *row vectors*. In both cases, they will have properties identical to the geometric vectors studied earlier in mulitvariable calculus. I assume that most of you have seen this material previously in a course named linear algebra, matrix algebra, or something similar. Section 6.2 in the text is intended as both a quick introduction and review of this material.

MATRIX ADDITION

Let $A = \left[a_{i,j}\right]$ and $B = \left[b_{i,j}\right]$ be matrices of order $m \times n$. Then

$$C = A + B$$

is another matrix of order $m \times n$, with

$$c_{i.j} = a_{i,j} + b_{i,j}$$

EXAMPLE.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} + \begin{bmatrix} 1 & -1 \\ -1 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ 2 & 5 \\ 6 & 5 \end{bmatrix}$$

MULTIPLICATION BY A CONSTANT

$$c\begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \cdots & a_{m,n} \end{bmatrix} = \begin{bmatrix} ca_{1,1} & \cdots & ca_{1,n} \\ \vdots & \ddots & \vdots \\ ca_{m,1} & \cdots & ca_{m,n} \end{bmatrix}$$

EXAMPLE.

$$5\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} = \begin{bmatrix} 5 & 10 \\ 15 & 20 \\ 25 & 30 \end{bmatrix}$$

$$(-1)\begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} -a & -b \\ -c & -d \end{bmatrix}$$

THE ZERO MATRIX 0

Define the *zero matrix* of order $m \times n$ as the matrix of that order having all zero entries. It is sometimes written as $0_{m\times n}$, but more commonly as simply 0. Then for any matrix A of order $m \times n$,

$$A + 0 = 0 + A = A$$

The zero matrix $0_{m\times n}$ acts in the same role as does the number zero when doing arithmetic with real and complex numbers.

EXAMPLE.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

We denote by $-A$ the solution of the equation

$$A + B = 0$$

It is the matrix obtained by taking the negative of all of the entries in A. For example,

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} -a & -b \\ -c & -d \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\Rightarrow \quad -\begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} -a & -b \\ -c & -d \end{bmatrix} = (-1)\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$-\begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} = \begin{bmatrix} -a_{1,1} & -a_{1,2} \\ -a_{2,1} & -a_{2,2} \end{bmatrix}$$

MATRIX MULTIPLICATION

Let $A = \left[a_{i,j}\right]$ have order $m \times n$ and $B = \left[b_{i,j}\right]$ have order $n \times p$. Then

$$C = AB$$

is a matrix of order $m \times p$ and

$$\begin{aligned} c_{i,j} &= A_{i,*}B_{*,j} \\ &= a_{i,1}b_{1,j} + a_{i,2}b_{2,j} + \cdots + a_{i,n}b_{n,j} \end{aligned}$$

or equivalently

$$\begin{aligned} c_{i,j} &= \begin{bmatrix} a_{i,1} & a_{i,2} & \cdots & a_{i,n} \end{bmatrix} \begin{bmatrix} b_{1,j} \\ b_{2,j} \\ \vdots \\ b_{n,j} \end{bmatrix} \\ &= a_{i,1}b_{1,j} + a_{i,2}b_{2,j} + \cdots + a_{i,n}b_{n,j} \end{aligned}$$

EXAMPLES

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} = \begin{bmatrix} 22 & 28 \\ 49 & 64 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} = \begin{bmatrix} 9 & 12 & 15 \\ 19 & 26 & 33 \\ 29 & 40 & 51 \end{bmatrix}$$

$$\begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,n} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} a_{1,1}x_1 + \cdots + a_{1,n}x_n \\ \vdots \\ a_{n,1}x_1 + \cdots + a_{n,n}x_n \end{bmatrix}$$

Thus we write the linear system

$$\begin{aligned} a_{1,1}x_1 + \cdots + a_{1,n}x_n &= b_1 \\ &\vdots \\ a_{n,1}x_1 + \cdots + a_{n,n}x_n &= b_n \end{aligned}$$

as

$$Ax = b$$

THE IDENTITY MATRIX I

For a given integer $n \geq 1$, Define I_n to be the matrix of order $n \times n$ with 1's in all diagonal positions and zeros elsewhere:

$$I_n = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & \dots & & 1 \end{bmatrix}$$

More commonly it is denoted by simply I.

Let A be a matrix of order $m \times n$. Then

$$AI_n = A, \qquad I_m A = A$$

The identity matrix I acts in the same role as does the number 1 when doing arithmetic with real and complex numbers.

THE MATRIX INVERSE

Let A be a matrix of order $n \times n$ for some $n \geq 1$. We say a matrix B is an *inverse* for A if

$$AB = BA = I$$

It can be shown that if an inverse exists for A, then it is unique.

EXAMPLES. If $ad - bc \neq 0$, then

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 \\ 2 & 2 \end{bmatrix}^{-1} = \begin{bmatrix} -1 & 1 \\ 1 & -\frac{1}{2} \end{bmatrix}$$

$$\begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{bmatrix}^{-1} = \begin{bmatrix} 9 & -36 & 30 \\ -36 & 192 & -180 \\ 30 & -180 & 180 \end{bmatrix}$$

Recall the earlier theorem on the solution of linear systems $Ax = b$ with A a square matrix.

Theorem. The following are equivalent statements.

1. For each b, there is exactly one solution x.

2. For each b, there is a solution x.

3. The homogeneous system $Ax = 0$ has only the solution $x = 0$.

4. $\det(A) \neq 0$.

5. A^{-1} exists.

EXAMPLE

$$\det \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = 0$$

Therefore, the linear system

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

is not always solvable, the coefficient matrix does not have an inverse, and the homogeneous system $Ax = 0$ has a solution other than the zero vector, namely

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

The arithmetic properties of matrix addition and multiplication are listed on page 248, and some of them require some work to show. For example, consider showing the distributive law for matrix multiplication,

$$(AB)\,C = A\,(BC)$$

with A, B, C matrices of respective orders $m \times n, n \times p$, and $p \times q$, respectively. Writing this out, we want to show

$$\sum_{k=1}^{p} (AB)_{i,k}\, C_{k,l} = \sum_{j=1}^{n} A_{i,j}\, (BC)_{j,l}$$

for $1 \leq i \leq m$, $1 \leq l \leq q$.

With new situations, we often use notation to suggest what should be true. But this is done only after deciding what actually is true. You should read carefully the properties given in the text on page 248.

PARTITIONED MATRICES

Matrices can be built up from smaller matrices; or conversely, we can decompose a large matrix into a matrix of smaller matrices. For example, consider

$$A = \begin{bmatrix} 1 & 2 & 0 \\ 2 & 1 & 1 \\ 0 & -1 & 5 \end{bmatrix} = \begin{bmatrix} B & c \\ d & e \end{bmatrix}$$

$$B = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} \quad c = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad d = \begin{bmatrix} 0 & -1 \end{bmatrix} \quad e = 5$$

Matlab allows you to build up larger matrices out of smaller matrices in exactly this manner; and smaller matrices can be defined as portions of larger matrices. We will often write an $n \times n$ square matrix in terms of its columns:

$$A = \left[A_{*,1}, ..., A_{*,n}\right]$$

For the $n \times n$ identity matrix I, we write

$$I = [e_1, ..., e_n]$$

with e_j denoting a column vector with a 1 in position j and zeros elsewhere.

ARITHMETIC OF PARTITIONED MATRICES

As with matrices, we can do addition and multiplication with partitioned matrices provided the individual constituent parts have the proper orders.

For example, let A, B, C, D be $n \times n$ matrices. Then

$$\begin{bmatrix} I & A \\ B & I \end{bmatrix} \begin{bmatrix} I & C \\ D & I \end{bmatrix} = \begin{bmatrix} I + AD & C + A \\ B + D & I + BC \end{bmatrix}$$

Let A be $n \times n$ and x be a column vector of length n. Then

$$Ax = \begin{bmatrix} A_{*,1}, ..., A_{*,n} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = x_1 A_{*,1} + \cdots + x_n A_{*,n}$$

Compare this to

$$\begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,n} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} a_{1,1}x_1 + \cdots + a_{1,n}x_n \\ \vdots \\ a_{n,1}x_1 + \cdots + a_{n,n}x_n \end{bmatrix}$$

PARTITIONED MATRICES IN MATLAB

In MATLAB, matrices can be constructed using smaller matrices. For example, let

$A = [1, \quad 2; \quad 3, \quad 4]; \quad x = [5, \quad 6]; \quad y = [7, \quad 8]';$

Then

$$B = [A, \quad y; \quad x, \quad 9];$$

forms the matrix

$$B = \begin{bmatrix} 1 & 2 & 7 \\ 3 & 4 & 8 \\ 5 & 6 & 9 \end{bmatrix}$$

ELEMENTARY ROW OPERATIONS

As preparation for the discussion of Gaussian Elimination in Section 6.3, we introduce three elementary row operations on general rectangular matrices. They are:

i) *Interchange of two rows.*

ii) *Multiplication of a row by a nonzero scalar*.

iii) *Addition of a nonzero multiple of one row to another row.*

Consider the rectangular matrix

$$A = \begin{bmatrix} 3 & 3 & 3 & 1 \\ 2 & 2 & 3 & 1 \\ 1 & 2 & 3 & 1 \end{bmatrix}$$

We add row 2 times (-1) to row 1, and then add row 3 times (-1) to row 2 to obtain the matrix:

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 2 & 3 & 1 \end{bmatrix}$$

Add row 2 times (-1) to row 1, and to row 3 as well:

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 2 & 3 & 1 \end{bmatrix}$$

Add row 1 times (-2) to row 3:

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 3 & 1 \end{bmatrix}$$

Interchange row 1 and row 2:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 3 & 1 \end{bmatrix}$$

Finally, we multiply row 3 by $1/3$:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1/3 \end{bmatrix}$$

This is obtained from A using elementary row operations. A reverse sequence of operations of the same type converts this result back to A.

NUMERICAL METHODS FOR ODEs

Consider the initial value problem

$$y' = f(x, y), \quad x_0 \le x \le b, \quad y(x_0) = Y_0$$

and denote its solution by $Y(x)$. Most numerical methods solve this by finding values at a set of *node points*:

$$x_0 < x_1 < \cdots < x_N \le b$$

The approximating values are denoted in this book in various ways. Most simply, we have

$$y_1 \approx Y(x_1), \ \cdots \ , \ y_N \approx Y(x_N)$$

We also use

$$y(x_i) \equiv y_i, \quad i = 0, 1, ..., N$$

To begin with, and for much of our work, we use a fixed *stepsize* h, and we generate the node points by

$$x_i = x_0 + i\,h, \quad i = 0, 1, ..., N$$

Then we also write

$$y_h(x_i) \equiv y_i \approx Y(x_i), \quad i = 0, 1, ..., N$$

EULER'S METHOD

Euler's method is defined by

$$y_{n+1} = y_n + h\, f(x_n, y_n), \quad n = 0, 1, ..., N-1$$

with $y_0 = Y_0$. Where does this method come from?

There are various perspectives from which we can derive numerical methods for solving

$$y' = f(x, y), \quad x_0 \le x \le b, \quad y(x_0) = Y_0$$

and Euler's method is simplest example of most such perspectives. Moreover, the error analysis for Euler's method is introduction to the error analysis of most more rapidly convergent (and more practical) numerical methods.

A GEOMETRIC PERSPECTIVE

Look at the graph of $y = Y(x)$, beginning at $x = x_0$. Approximate this graph by the line tangent at $(x_0, Y(x_0))$:

$$\begin{aligned} y &= Y(x_0) + (x - x_0)Y'(x_0) \\ &= Y(x_0) + (x - x_0)f(x_0, Y_0) \end{aligned}$$

Evaluate this tangent line at x_1 and use this value to approximate $Y(x_1)$. This yields Euler's approximation.

We could generalize this by looking for more accurate means of approximating a function, e.g. by using a higher degree Taylor approximation.

An illustration of Euler's method derivation

TAYLOR'S SERIES

Approximate $Y(x)$ about x_0 by a Taylor polynomial approximation of some degree:

$$Y(x_0 + h) \approx Y(x_0) + h\,Y'(x_0) + \frac{h^2}{2!}Y''(x_0) + \cdots + \frac{h^p}{p!}Y^{(p)}(x_0)$$

Euler's method is the case $p = 1$:

$$\begin{aligned} Y(x_0 + h) &\approx Y(x_0) + h\,Y'(x_0) \\ &= y_0 + h\,f(x_0, y_0) \equiv y_1 \end{aligned}$$

We have an error formula for Taylor polynomial approximations; and in this case,

$$Y(x_1) - y_1 = \frac{h^2}{2}Y''(\xi_0)$$

with some $x_0 \le \xi_0 \le x_1$.

GENERAL ERROR FORMULA

In general,

$$y_{n+1} = y_n + h\, f(x_n, y_n), \quad n = 0, 1, ..., N-1$$

$$\begin{aligned} Y(x_{n+1}) &= Y(x_n) + h\, Y'(x_n) + \frac{h^2}{2} Y''(\xi_n) \\ &= Y(x_n) + h\, f(x_n, Y(x_n)) + \frac{h^2}{2} Y''(\xi_n) \end{aligned}$$

with some $x_n \le \xi_n \le x_{n+1}$.

We will use this as the starting point of our error analyses of Euler's method. In particular,

$$\begin{aligned} Y(x_{n+1}) - y_{n+1} &= Y(x_n) - y_n \\ &\quad + h\,[\, f(x_n, Y(x_n)) - f(x_n, y_n)] \\ &\quad + \frac{h^2}{2} Y''(\xi_n) \end{aligned}$$

NUMERICAL DIFFERENTIATION

From beginning calculus,

$$Y'(x_n) \approx \frac{Y(x_n + h) - Y(x_n)}{h}$$

This leads to

$$\begin{aligned} Y(x_n + h) &\approx Y(x_n) + h\, Y'(x_n) \\ &= Y(x_n) + h\ f(x_n, Y(x_n)) \\ &\approx y_n + h\, f(x_n, y_n) \end{aligned}$$

Most numerical differentiation approximations can be used to obtain numerical methods for solving the initial value problem. **However**, most such formulas turn out to be poor methods for solving differential equations. We will see an example of this in the one of the following sections of the book.

EXAMPLE. Solve

$$Y'(x) = \frac{Y(x) + x^2 - 2}{x + 1}, \qquad Y(0) = 1$$

The solution is

$$Y(x) = x^2 + 2x + 2 - 2\,(x + 1)\log\,(x + 1)$$

We give selected results for three values of h. Note the behaviour of the error as h is halved.

h	x	$y_h(x)$	*Error*	*Relative Error*
0.2	1.0	2.1592	$6.82E-2$	0.0306
	3.0	5.4332	$4.76E-1$	0.0805
	5.0	14.406	1.09	0.0703
0.1	1.0	2.1912	$3.63E-2$	0.0163
	3.0	5.6636	$2.46E-1$	0.0416
	5.0	14.939	$5.60E-1$	0.0361
0.05	1.0	2.2087	$1.87E-2$	0.00840
	3.0	5.7845	$1.25E-1$	0.0212
	5.0	15.214	$2.84E-1$	0.0183

GENERAL ERROR FORMULA

In general,

$$y_{n+1} = y_n + h\, f(x_n, y_n), \quad n = 0, 1, ..., N-1$$

$$\begin{aligned} Y(x_{n+1}) &= Y(x_n) + h\, Y'(x_n) + \frac{h^2}{2} Y''(\xi_n) \\ &= Y(x_n) + h\, f(x_n, Y(x_n)) + \frac{h^2}{2} Y''(\xi_n) \end{aligned}$$

with some $x_n \le \xi_n \le x_{n+1}$.

We will use this as the starting point of our discussion of the error in Euler's method. In particular,

$$\begin{aligned} Y(x_{n+1}) - y_{n+1} &= Y(x_n) - y_n \\ &\quad + h\,[\, f(x_n, Y(x_n)) - f(x_n, y_n)] \\ &\quad + \frac{h^2}{2} Y''(\xi_n) \end{aligned}$$

ERROR ANALYSIS - SPECIAL CASES

We begin with a couple of special cases, to obtain some additional intuition on the behaviour of the error. Consider

$$y' = 2x, \quad y(0) = 0$$

This has the solution $Y(x) = x^2$. Euler's method becomes

$$y_{n+1} = y_n + 2x_n h, \quad y_0 = 0$$

$$\begin{aligned} y_1 &= y_0 + 2x_0 h &= x_1 x_0 \\ y_2 &= y_1 + 2x_1 h &= x_1 x_0 + 2x_1 h = x_2 x_1 \\ y_3 &= y_2 + 2x_2 h &= x_2 x_1 + 2x_2 h = x_3 x_2 \end{aligned}$$

By an induction argument,

$$y_n = x_n x_{n-1}, \quad n \geq 1$$

For the error,

$$Y(x_n) - y_n = x_n^2 - x_n x_{n-1} = x_n h$$

Note that $Y(x_n) - y_n \propto h$ at each fixed x_n.

Return to our error equation

$$\begin{aligned} Y(x_{n+1}) - y_{n+1} &= Y(x_n) - y_n \\ &\quad + h\,[\,f(x_n, Y(x_n)) - f(x_n, y_n)] \\ &\quad + \frac{h^2}{2} Y''(\xi_n) \end{aligned} \tag{1}$$

With the mean value theorem,

$$f(x_n, Y(x_n)) - f(x_n, y_n) = \frac{\partial f(x_n, \zeta_n)}{\partial y}\,[Y(x_n) - y_n]$$

for some ζ_n between $Y(x_n)$ and y_n. As shorthand, use $e_h(x) = Y(x) - y_h(x)$. Then we can write

$$e_h(x_{n+1}) = \left[1 + h\frac{\partial f(x_n, \zeta_n)}{\partial y}\right] e_h(x_n) + \frac{h^2}{2} Y''(\xi_n) \tag{2}$$

with $e_h(x_0) = 0$. We also will assume henceforth that

$$K \equiv \max_{\substack{x_0 \le x \le b \\ -\infty < y < \infty}} \left| \frac{\partial f(x, y)}{\partial y} \right| < \infty$$

Consider those differential equations with

$$\frac{\partial f(x,y)}{\partial y} \le 0, \quad x_0 \le x \le b, \quad -\infty < y < \infty$$

Then

$$-1 \le 1 + h\frac{\partial f(x_n,\zeta_n)}{\partial y} \le 1$$

provided h is chosen sufficiently small, *e.g.* if

$$h < \frac{1}{K}$$

Using this in our error formula (2),

$$|e_h(x_{n+1})| \le |e_h(x_n)| + \frac{h^2}{2}\left\|Y''\right\|_\infty, \quad n \ge 0 \quad (3)$$

in which

$$\left\|Y''\right\|_\infty = \max_{x_0 \le t \le b}\left|Y''(t)\right|$$

Using induction with (3), we can prove

$$|e_h(x_n)| \le \frac{h}{2}(x_n - x_0)\left\|Y''\right\|_\infty$$

Again the error is bounded by something of the form $c(x_n)h$.

EXAMPLE

Solve the initial value problems

$$y' = -y + \cos x - \sin x, \quad 0 \le x \le 5, \quad y(0) = 2$$

Its true solution is

$$Y(x) = e^{-x} + \cos x$$

Then

$$Y''(x) = e^{-x} - \cos x$$
$$\|Y''\|_\infty = \max_{0 \le x \le 5} |Y''(x)| \doteq 1.0442$$

We solve the problem with Euler's method on $[0, 5]$.

For this differential equation,

$$f(x, y) = -y + \cos x - \sin x, \qquad \frac{\partial f}{\partial y} = -1$$

Then the error bound becomes

$$\begin{aligned} |e_h(x_n)| &\le \frac{h}{2}(x_n - x_0)\,\|Y''\|_\infty \\ &= 0.5221 h x_n \end{aligned}$$

The following table gives the error when solving this problem with $h = .1$ on the interval $[0, 5]$. The final column is the error bound, and it clearly exceeds the magnitude of the actual error at each value of x_n.

x_n	y_n	$Y(x_n) - y_n$	$0.5221hx_n$
0	2.00000000	0.000000	0.00000
1	.91533345	$-.007152$	.05221
2	$-.28535798$	.004546	.10442
3	$-.97071646$	.030511	.15663
4	$-.67539538$	.040067	.20884
5	.27163709	.018763	.26105

GENERAL ERROR ANALYSIS

Return to

$$e_h(x_{n+1}) = \left[1 + h\frac{\partial f(x_n, \zeta_n)}{\partial y}\right] e_h(x_n) + \frac{h^2}{2} Y''(\xi_n)$$

in which

$$e_h(x) = Y(x) - y_h(x)$$

As before, we assume

$$K \equiv \max_{\substack{-\infty<y<\infty \\ x_0 \le x \le b}} \left|\frac{\partial f(x,y)}{\partial y}\right| < \infty$$

This is much more restrictive than is needed, but it simplifies the statement and analysis of the error in solving differential equations. From this we can show

$$|e_h(x_n)| \le e^{(x_n - x_0)K} |e_h(x_0)| + h\frac{e^{(x_n-x_0)K} - 1}{2K} \left\|Y''\right\|_\infty$$

for all points $x_0 \le x_n \le b$. In theory, we would have $e_h(x_0) = 0$. But we can also examine what would happen if we allowed for errors in the choice of y_0.

EXAMPLE

Recall the last example, of solving

$$y' = -y + \cos x - \sin x, \quad 0 \le x \le 5, \quad y(0) = 2$$

whose true solution is $Y(x) = e^{-x} + \cos x$. Again,

$$f(x, y) = -y + \cos x - \sin x, \qquad \frac{\partial f}{\partial y} = -1$$

and then $K = 1$. Recall

$$Y''(x) = e^{-x} - \cos x$$
$$\|Y''\|_\infty = \max_{0 \le x \le 5} |Y''(x)| \doteq 1.0442$$

and also assume $e_h(0) = 0$. Then our general error bound becomes

$$\begin{aligned} |e_h(x_n)| &\le h \frac{e^{(x_n - x_0)K} - 1}{2K} \|Y''\|_\infty \\ &= h \frac{e^{x_n} - 1}{2} (1.0442) \end{aligned}$$

This is consistent with our earlier bound since

$$e^{x_n} - 1 \ge x_n$$

ASYMPTOTIC ERROR FORMULA

We see from the earlier results that the error satisfies

$$|Y(x_n) - y_n| \leq c\,(x_n)\,h, \quad x_0 \leq x_n \leq b$$

for some number $c\,(x_n)$. We can improve upon this. Namely, it can be shown that

$$Y(x_n) - y_h(x_n) = D(x_n)h + O\left(h^2\right)$$

The term $O(h^2)$ denotes a quantity which is bounded by a constant times h^2. We say the term being bounded is of <u>order</u> h^2. Thus

$$Y(x_n) - y_h(x_n) \approx D(x_n)h \tag{4}$$

for smaller values of h. The function $D(x)$ satisfies a particular differential equation, but it can seldom be found in practice since it depends on the solution $Y(x)$. Instead we use (4) to justify using Richardson extrapolation to estimate the error.

RICHARDSON EXTRAPOLATION

Consider solving the initial value problem

$$y' = f(x, y), \quad x_0 \le x \le b, \qquad y(x_0) = Y_0$$

with Euler's method, as suppose we do it twice, using stepsizes of h and $2h$. Denote the respective numerical solutions by $y_h(x_n)$ and $y_{2h}(x_n)$. Then from (4) above, and at a generic node point x,

$$\begin{aligned} Y(x) - y_h(x) &\approx D(x)h \\ Y(x) - y_{2h}(x) &\approx D(x)\,(2h) \end{aligned}$$

Multiply the first equation by 2 and then subtract the second equation. This yields

$$\begin{gathered} Y(x) - 2y_h(x) + y_{2h}(x) \approx 0 \\ Y(x) \approx 2y_h(x) - y_{2h}(x) \end{gathered}$$

This last formula is called "Richardson's extrapolation formula" for Euler's method. We can also use it to estimate the error.

$$\begin{aligned} Y(x) - y_h(x) &\approx [2y_h(x) - y_{2h}(x)] - y_h(x) \\ &= y_h(x) - y_{2h}(x) \end{aligned}$$

The formula

$$Y(x) - y_h(x) \approx y_h(x) - y_{2h}(x)$$

is called "Richardson's error estimation formula" for Euler's method.

<u>EXAMPLE</u>: Recall the last example, of solving

$$y' = -y + \cos x - \sin x, \quad 0 \le x \le 5, \quad y(0) = 2$$

whose true solution is $Y(x) = e^{-x} + \cos x$. We do so with stepsizes $h = .1$ and $2h = .2$.

x_n	$y_h(x_n)$	$Y(x_n) - y_h(x_n)$	$y_h(x) - y_{2h}(x)$
0	2.00000000	0.000000	0.00000
1	.91533345	$-.007152$	-0.007593
2	$-.28535798$	.004546	0.004420
3	$-.97071646$	.030511	0.031741
4	$-.67539538$	.040067	0.041539
5	.27163709	.018763	0.018820

NUMERICAL STABILITY; IMPLICIT METHODS

When solving the initial value problem

$$\begin{aligned} Y'(x) &= f(x, Y(x)), \qquad x_0 \le x \le b \\ Y(x_0) &= Y_0 \end{aligned}$$

we know that small changes in the initial data Y_0 will result in small changes in the solution of the differential equation. More precisely, consider the perturbed problem

$$\begin{aligned} Y_\varepsilon'(x) &= f(x, Y_\varepsilon(x)), \qquad x_0 \le x \le b \\ Y_\varepsilon(x_0) &= Y_0 + \varepsilon \end{aligned}$$

Then assuming $f(x, z)$ and $\partial f(x, z)/\partial z$ are continuous for $x_0 \le x \le b$, $-\infty < z < \infty$, we have

$$\max_{x_0 \le x \le b} |Y_\varepsilon(x) - Y(x)| \le c\,|\varepsilon|$$

for some constant $c > 0$. We would like our numerical methods to have a similar property.

Consider the Euler method

$$\begin{aligned} y_{n+1} &= y_n + hf\left(x_n, y_n\right), \qquad n = 0, 1, \ldots \\ y_0 &= Y_0 \end{aligned}$$

and then consider the perturbed problem

$$\begin{aligned} y_{n+1}^{\varepsilon} &= y_n^{\varepsilon} + hf\left(x_n, y_n^{\varepsilon}\right), \qquad n = 0, 1, \ldots \\ y_0^{\varepsilon} &= Y_0 + \varepsilon \end{aligned}$$

We can show the following.

$$\max_{x_0 \le x_n \le b} \left| y_n^{\varepsilon} - y_n \right| \le \widehat{c} \left| \varepsilon \right|$$

for some constant $\widehat{c} > 0$ and for all sufficiently small values of the stepsize h.

This implies that Euler's method is stable, and in the same manner as was true for the original differential equation problem.

The general idea of *stability* for a numerical method is essentially that given above for Eulers's method.

There is a general theory for numerical methods for solving the initial value problem

$$\begin{aligned} Y'(x) &= f(x, Y(x)), \quad x_0 \leq x \leq b \\ Y(x_0) &= Y_0 \end{aligned}$$

If the *truncation error* in a numerical method has order 2 or greater, then the numerical method is stable **if and only if** it is a convergent numerical method.

Note that Euler's method has a truncation error of order 2.

The numerical methods studied in this chapter are both stable and convergent.

All of these general results on stability and convergence are valid if the stepsize h is *sufficiently small*. What is meant by this?

Rather than studying this for a general problem, we restrict our interest to the *model problem*

$$Y'(x) = \lambda Y(x), \quad x \geq 0, \quad Y(0) = 1$$

The analysis for this problem is generally applicable to the more general differential equation problem.

The true solution is $Y(x) = e^{\lambda x}$. When $\lambda < 0$ or λ is complex with Real$(\lambda) < 0$, we have $Y(x) \to 0$ as $x \to \infty$. We would like the same to be true for the numerical solution of the model problem.

We begin by studying Euler's method applied to the model problem.

EULER'S METHOD AND THE MODEL PROBLEM

Apply Euler's method to the model problem.

$$\begin{aligned} y_{n+1} &= y_n + h\lambda y_n \\ &= (1 + h\lambda)\, y_n, \quad n = 0, 1, \ldots \end{aligned}$$

with $y_0 = 1$. By induction,

$$y_n = (1 + h\lambda)^n, \quad n \geq 0$$

The numerical solution $y_n \to 0$ as $n \to \infty$ if and only if

$$|1 + h\lambda| < 1$$

In the case λ is real and negative, this is equivalent to

$$-2 < h\lambda < 0$$

If $|\lambda|$ is quite large, then h must be correspondingly small. Usually the truncation error does not require such a small value of h; it is needed only for stability.

NUMERICAL EXAMPLE: Consider the problem

$$Y'(x) = \lambda Y(x) + (1-\lambda)\cos(x) - (1+\lambda)\sin(x)$$

with $Y(0) = 1$. The true solution is $Y(x) = \sin(x) + \cos(x)$. We use Euler's method and give numerical results for varying λ and h are given. The truncation error $T_{n+1} = \frac{h^2}{2}Y''(\xi_n)$ does not depend on λ, but the numerical solution does.

λ	x	Error: $h = 0.5$	Error: $h = 0.1$	Error: $h = 0.01$
-1	1	$-2.46\text{E}-1$	$-4.32\text{E}-2$	$-4.22\text{E}-3$
	3	$-2.66\text{E}-2$	$-6.78\text{E}-3$	$-7.22\text{E}-4$
	5	$2.72\text{E}-1$	$4.91\text{E}-2$	$4.81\text{E}-3$
-10	1	$3.98\text{E}-1$	$-6.99\text{E}-3$	$-6.99\text{E}-4$
	3	$1.11\text{E}+2$	$3.86\text{E}-3$	$3.64\text{E}-4$
	5	$2.83\text{E}+4$	$3.78\text{E}-3$	$3.97\text{E}-4$
-50	1	$3.26\text{E}+0$	$1.06\text{E}+3$	$-1.39\text{E}-4$
	3	$1.08\text{E}+6$	$1.17\text{E}+15$	$8.25\text{E}-5$
	5	$3.59\text{E}+11$	$1.28\text{E}+27$	$7.00\text{E}-5$

ABSOLUTE STABILITY

The set of values of $h\lambda$ for which the numerical solution $y_n \to 0$ as $n \to \infty$ is called the *region of absolute stability* of the numerical method. We allow λ to be complex, restricting it with Real$(\lambda) < 0$.

With Euler's method, this region is the set of all complex numbers $z = h\lambda$ for which

$$|1 + z| < 1$$

or equivalently,

$$|z - (-1)| < 1$$

This is a circle of radius one in the complex plane, centered at the complex number $-1 + 0 \cdot i$.

If a numerical method has no restrictions on λ in order to have $y_n \to 0$ as $n \to \infty$, we say the numerical method is *A-stable*.

THE BACKWARD EULER METHOD

Expand the function $Y(x)$ as a linear Taylor polynomial about x_{n+1}:

$$\begin{aligned} Y(x) = Y(x_{n+1}) + (x - x_{n+1})\, Y'(x_{n+1}) \\ + \tfrac{1}{2}\,(x - x_{n+1})^2\, Y''(\zeta_n) \end{aligned}$$

with ζ_n between x and x_{n+1}. Let $x = x_n$, solve for $Y(x_{n+1})$, and use the differential equation to evaluate $Y'(x_{n+1})$:

$$\begin{aligned} Y(x_{n+1}) &= Y(x_n) + hY'(x_{n+1}) - \frac{1}{2}h^2 Y''(\zeta_n) \\ &= Y(x_n) + hf\left(x_{n+1}, Y(x_{n+1})\right) - \frac{1}{2}h^2 Y''(\zeta_n) \end{aligned}$$

with ζ_n between x_n and x_{n+1}. The *backward Euler method* is obtained by dropping the truncation error:

$$\begin{aligned} y_{n+1} &= y_n + hf\left(x_{n+1}, y_{n+1}\right), \qquad n = 0, 1, \ldots \\ y_0 &= Y_0 \end{aligned}$$

The truncation is essentially of the same size as for Euler's method, but of opposite sign.

Apply the backward Euler method to the model problem

$$Y'(x) = \lambda Y(x), \quad x \geq 0, \quad Y(0) = 1$$

This yields

$$\begin{aligned} y_{n+1} &= y_n + h\lambda y_{n+1} \\ y_{n+1} &= \frac{1}{1 - h\lambda} y_n, \quad n = 0, 1, \ldots \end{aligned}$$

with $y_0 = 1$. By induction,

$$y_n = \left(\frac{1}{1 - h\lambda}\right)^n, \quad n = 0, 1, \ldots$$

We want to know when $y_n \to \infty$ as $n \to \infty$. This will be true if

$$\left|\frac{1}{1 - h\lambda}\right| < 1$$

The hypothesis that $\lambda < 0$ or Real(λ) < 0 is sufficient to show this is true, regardless of the size of the stepsize h. Thus the backward Euler method is an A-stable numerical method.

NUMERICAL EXAMPLE

We apply the backward Euler method to the problem

$$Y'(x) = \lambda Y(x) + (1-\lambda)\cos(x) - (1+\lambda)\sin(x)$$

with $Y(0) = 1$. The true solution is $Y(x) = \sin(x) + \cos(x)$. We give numerical results with varying λ and with $h = 0.5$. The truncation error $T_{n+1} = -\frac{h^2}{2}Y''(\xi_n)$ does not depend on λ, but the numerical solution does. But in contrast to Euler's method, there are no stability problems with the numerical solution.

x	*Error:* $\lambda = -1$	*Error:* $\lambda = -10$	*Error:* $\lambda = -50$
2	2.08E − 1	1.97E − 2	3.60E − 3
4	−1.63E − 1	−3.35E − 2	−6.94E − 3
6	−7.04E − 2	8.19E − 3	2.18E − 3
8	2.22E − 1	2.67E − 2	5.13E − 3
10	−1.14E − 1	−3.04E − 2	−6.45E − 3

SOLVING THE BACKWARD EULER METHOD

For a general differential equation, we must solve

$$y_{n+1} = y_n + hf\left(x_{n+1}, y_{n+1}\right) \tag{1}$$

for each n. In most cases, this is a rootfinding problem for the equation

$$z = y_n + hf\left(x_{n+1}, z\right) \tag{2}$$

with the root $z = y_{n+1}$. Such numerical methods (1) for solving differential equations are called *implicit methods*.

Methods in which y_{n+1} is given explicitly are called *explicit methods*. Euler's method is an explicit method.

Fixed point iteration is often used to solve (2):

$$y_{n+1}^{(k+1)} = y_n + hf\left(x_{n+1}, y_{n+1}^{(k)}\right), \quad k = 0, 1, \ldots \tag{3}$$

For an initial guess, we use $y_{n+1}^{(0)} = y_n$ or something even better. The iteration error satisfies

$$y_{n+1} - y_{n+1}^{(k+1)} \approx h\frac{\partial f(x_{n+1}, y_{n+1})}{\partial y}\left[y_{n+1} - y_{n+1}^{(k)}\right] \tag{4}$$

We have convergence if

$$h\left|\frac{\partial f(x_{n+1}, y_{n+1})}{\partial y}\right| < 1 \tag{5}$$

which is true if h is sufficiently small. If this is too restrictive on h, then another rootfinding method must be used to solve (2).

THE TRAPEZOIDAL METHOD

The backward Euler method is stable, but still is lacking in accuracy. A similar but more accurate numerical method is the *trapezoidal method*:

$$y_{n+1} = y_n + \frac{h}{2}\left[f\left(x_n, y_n\right) + f\left(x_{n+1}, y_{n+1}\right)\right], \qquad n = 0, 1, \ldots \tag{6}$$

It is derived by applying the simple trapezoidal numerical integration rule to the equation

$$Y(x_{n+1}) = Y(x_n) + \int_{x_n}^{x_{n+1}} f\left(t, Y(t)\right)\ dt$$

obtaining

$$\begin{aligned} Y(x_{n+1}) &= Y(x_n) + \frac{h}{2}\left[f\left(x_n, Y(x_n)\right)\right. \\ &\quad \left. + f\left(x_{n+1}, Y(x_{n+1})\right)\right] - \frac{h^3}{12} Y'''(\xi_n) \end{aligned}$$

The method (6) results from dropping the truncation error term.

As with the backward Euler method, the equation (6) is a nonlinear equation with a root of y_{n+1}. Again, fixed point iteration can be used to solve it:

$$y_{n+1}^{(j+1)} = y_n + \frac{h}{2}[f(x_n, y_n) + f(x_{n+1}, y_{n+1}^{(j)})]$$

for $j = 0, 1, 2, \ldots$ The iteration will converge if

$$\left| \frac{h}{2} \cdot \frac{\partial f(x_{n+1}, y_{n+1})}{\partial z} \right| < 1$$

provided also that $y_{n+1}^{(0)}$ is a sufficiently good initial guess. Often we use

$$\begin{aligned} y_{n+1}^{(0)} &= y_n + hf\left(x_n, y_n\right) \\ & \quad \text{or} \\ y_{n+1}^{(0)} &= y_n + \frac{h}{2}[3f(x_n, y_n) - f(x_{n-1}, y_{n-1})] \end{aligned}$$

These latter formulas are called *predictor formulas*. The first is simply Euler's method. The second is an *Adams-Bashforth method of order 2*; and it is an *explicit 2-step method*. Other rootfinding methods are used for more difficult problems.

CONVERGENCE: We can show that for all sufficiently small values of h,

$$\max_{x_0 \le x_n \le b} |Y(x_n) - y_n| \le ch^2 \max_{x_0 \le x \le b} \left| Y'''(x) \right|$$

The constant c depends on the Lipschitz constant K for $f(x, z)$:

$$K = \max_{\substack{x_0 \le x \le b \\ -\infty < z < \infty}} \left| \frac{\partial f(x, z)}{\partial z} \right|$$

Thus it converges more rapidly than either the Euler method or the backward Euler method.

STABILITY: The trapezoidal method is *absolutely stable*. Apply the method to the problem

$$Y'(x) = \lambda Y(x), \quad x \geq 0, \quad Y(0) = 1$$

Then

$$y_{n+1} = y_n + \frac{h}{2}\left[\lambda y_n + \lambda y_{n+1}\right]$$

with $y_0 = 1$. Solving for y_{n+1},

$$y_{n+1} = \left(\frac{1 + \frac{h\lambda}{2}}{1 - \frac{h\lambda}{2}}\right) y_n, \quad n \geq 0$$

By induction,

$$y_n = \left(\frac{1 + \frac{h\lambda}{2}}{1 - \frac{h\lambda}{2}}\right)^n, \quad n \geq 0$$

Then for λ real and negative, and also for λ complex with $\text{Real}\,(\lambda) < 0$, we can use this formula to show $y_n \to 0$ as $n \to \infty$. This shows the trapezoidal method is absolutely stable.

NUMERICAL EXAMPLE: We apply the trapezoidal method to the problem

$$Y'(x) = \lambda Y(x) + (1-\lambda)\cos(x) - (1+\lambda)\sin(x)$$

with $Y(0) = 1$. The true solution is $Y(x) = \sin(x) + \cos(x)$. We give numerical results with varying λ and with $h = 0.5$. The truncation error $T_{n+1} = -\frac{h^3}{12}Y'''(\xi_n)$ does not depend on λ, but the numerical solution does. As with the backward Euler method, there are no stability problems with the numerical solution. Moreover, it converges more rapidly than does the backward Euler method.

x	*Error:* $\lambda = -1$	*Error:* $\lambda = -10$	*Error:* $\lambda = -50$
2	$-1.13\text{E}-2$	$-2.78\text{E}-3$	$-7.91\text{E}-4$
4	$-1.43\text{E}-2$	$-8.91\text{E}-5$	$-8.91\text{E}-5$
6	$2.02\text{E}-2$	$2.77\text{E}-3$	$4.72\text{E}-4$
8	$-2.86\text{E}-3$	$-2.22\text{E}-3$	$-5.11\text{E}-4$
10	$-1.79\text{E}-2$	$-9.23\text{E}-4$	$-1.56\text{E}-4$

STIFF PROBLEMS: Linear equations

$$Y'(x) = \lambda Y(x) + g(x)$$

with Real $(\lambda) < 0$, $|\text{Real}\,(\lambda)|$ large, are called *stiff differential equations*. For general equations, the role of λ is replaced by

$$\frac{\partial f(x, Y(x))}{\partial z}$$

With many numerical methods, the truncation error for these equations may be satisfactorily small with not too small a value of h. But the large size of $|\text{Real}\,(\lambda)|$ may force h to be smaller so that λh will be in the stability region of the numerical method.

For stiff differential equations, one must use a numerical method with a large region of absolute stability, or else h must be chosen very small.

The backward Euler method and the trapezoidal method are very desirable because their stability region contains all λh where Real $(\lambda) < 0$. The backward Euler method and the trapezoidal rule have convergence orders 1 and 2, respectively.

The iteration method for calculating y_{n+1} in

$$\begin{aligned} y_{n+1} &= y_n + hf\left(x_{n+1}, y_{n+1}\right) \\ & \quad \text{or} \\ y_{n+1} &= y_n + \frac{h}{2}\left[f\left(x_n, y_n\right) + f\left(x_{n+1}, y_{n+1}\right)\right] \end{aligned}$$

must be something like Newton's method, something where the convergence is not affected by the size of λh.

SYSTEMS OF NONLINEAR EQUATIONS

Widely used in the mathematical modeling of real world phenomena.
We introduce some numerical methods for their solution.

For better intuition, we examine systems of two nonlinear equations and numerical methods for their solution. We then generalize to systems of an arbitrary order.

The Problem: Consider solving a system of two nonlinear equations

$$\begin{aligned} f(x,y) &= 0 \\ g(x,y) &= 0 \end{aligned} \tag{1}$$

Example: Consider solving the system

$$\begin{aligned} f(x,y) &\equiv x^2 + 4y^2 - 9 &&= 0 \\ g(x,y) &\equiv 18y - 14x^2 + 45 &&= 0 \end{aligned} \tag{2}$$

A graph of $z = f(x,y)$ is given in Figure 1, along with the curve for $f(x,y) = 0$.

Figure 1: Graph of $z = x^2 + 4y^2 - 9$, along with $z = 0$ on that surface

To visualize the points (x, y) that satisfy simultaneously both equations in (1), look at the intersection of the zero curves of the functions f and g. Figure 2 illustrates this. The zero curves intersect at four points, each of which corresponds to a solution of the system (2). For example, there is a solution near the point $(1, -1)$.

Figure 2: The graphs of $f(x, y) = 0$ and $g(x, y) = 0$

TANGENT PLANE APPROXIMATION

The graph of $z = f(x, y)$ is a surface in xyz-space.

For $(x, y) \approx (x_0, y_0)$, we approximate $f(x, y)$ by a plane that is tangent to the surface at the point $(x_0, y_0, f(x_0, y_0))$. The equation of this plane is $z = p(x, y)$ with

$$\begin{aligned} p(x, y) \;\equiv\; & f(x_0, y_0) \\ & +(x - x_0) f_x(x_0, y_0) + (y - y_0) f_y(x_0, y_0), \end{aligned} \tag{3}$$

$$f_x(x, y) = \frac{\partial f(x, y)}{\partial x}, \qquad f_y(x, y) = \frac{\partial f(x, y)}{\partial y},$$

the partial derivatives of f with respect to x and y, respectively.

If $(x, y) \approx (x_0, y_0)$, then

$$f(x, y) \approx p(x, y)$$

For functions of two variables, $p(x, y)$ is the linear Taylor polynomial approximation to $f(x, y)$.

NEWTON'S METHOD

Let $\alpha = (\xi, \eta)$ denote a solution of the system

$$\begin{aligned} f(x, y) &= 0 \\ g(x, y) &= 0 \end{aligned}$$

Let $(x_0, y_0) \approx (\xi, \eta)$ be an initial guess at the solution.

Approximate the surface $z = f(x, y)$ with the tangent plane at $(x_0, y_0, f(x_0, y_0))$.

If $f(x_0, y_0)$ is sufficiently close to zero, then the zero curve of $p(x, y)$ will be an approximation of the zero curve of $f(x, y)$ for those points (x, y) near (x_0, y_0).

Because the graph of $z = p(x, y)$ is a plane, its zero curve is simply a straight line.

Example. Consider $f(x, y) \equiv x^2 + 4y^2 - 9$. Then

$$f_x(x, y) = 2x, \qquad f_y(x, y) = 8y$$

At $(x_0, y_0) = (1, -1)$,

$$f(x_0, y_0) = -4, \quad f_x(x_0, y_0) = 2, \quad f_y(x_0, y_0) = -8$$

At $(1, -1, -4)$ the tangent plane to the surface $z = f(x, y)$ has the equation

$$z = p(x, y) \equiv -4 + 2(x - 1) - 8(y + 1)$$

The graphs of the zero curves of $f(x, y)$ and $p(x, y)$ for (x, y) near (x_0, y_0) are given in Figure 3.

Figure 3: $f(x, y) = 0$ and $p(x, y) = 0$

The tangent plane to the surface $z = g(x, y)$ at $(x_0, y_0, g(x_0, y_0))$ has the equation $z = q(x, y)$ with

$$q(x, y) \equiv g(x_0, y_0) + (x - x_0) g_x(x_0, y_0) + (y - y_0) g_y(x_0, y_0)$$

Recall that the solution $\alpha = (\xi, \eta)$ to

$$\begin{aligned} f(x, y) &= 0 \\ g(x, y) &= 0 \end{aligned}$$

is the intersection of the zero curves of $z = f(x, y)$ and $z = g(x, y)$.

Approximate these zero curves by those of the tangent planes $z = p(x, y)$ and $z = q(x, y)$. The intersection of these latter zero curves gives an approximate solution to the above nonlinear system.

Denote the solution to

$$\begin{aligned} p(x, y) &= 0 \\ q(x, y) &= 0 \end{aligned}$$

by (x_1, y_1).

Example. Return to the equations

$$\begin{aligned} f(x,y) &\equiv x^2 + 4y^2 - 9 &&= 0 \\ g(x,y) &\equiv 18y - 14x^2 + 45 &&= 0 \end{aligned}$$

with $(x_0, y_0) = (1, -1)$.

The use of the zero curves of the tangent plane approximations is illustrated in Figure 4.

Figure 4: $f = g = 0$ and $p = q = 0$

Calculating (x_1, y_1). To find the intersection of the zero curves of the tangent planes, we must solve the linear system

$$f(x_0, y_0) + (x - x_0)f_x(x_0, y_0) + (y - y_0)f_y(x_0, y_0) = 0$$
$$g(x_0, y_0) + (x - x_0)g_x(x_0, y_0) + (y - y_0)g_y(x_0, y_0) = 0$$

The solution is denoted by (x_1, y_1). In matrix form,

$$\begin{bmatrix} f_x(x_0, y_0) & f_y(x_0, y_0) \\ g_x(x_0, y_0) & g_y(x_0, y_0) \end{bmatrix} \begin{bmatrix} x - x_0 \\ y - y_0 \end{bmatrix} = - \begin{bmatrix} f(x_0, y_0) \\ g(x_0, y_0) \end{bmatrix}$$

It is actually computed as follows. Define δ_x and δ_y to be the solution of the linear system

$$\begin{bmatrix} f_x(x_0, y_0) & f_y(x_0, y_0) \\ g_x(x_0, y_0) & g_y(x_0, y_0) \end{bmatrix} \begin{bmatrix} \delta_x \\ \delta_y \end{bmatrix} = - \begin{bmatrix} f(x_0, y_0) \\ g(x_0, y_0) \end{bmatrix}$$

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + \begin{bmatrix} \delta_x \\ \delta_y \end{bmatrix}$$

Usually the point (x_1, y_1) is closer to the solution α than is the original point (x_0, y_0).

Continue this process, using (x_1, y_1) as a new initial guess. Obtain an improved estimate (x_2, y_2).

This iteration process is continued until a solution with sufficient accuracy is obtained.

The general iteration is given by

$$\begin{bmatrix} f_x(x_k, y_k) & f_y(x_k, y_k) \\ g_x(x_k, y_k) & g_y(x_k, y_k) \end{bmatrix} \begin{bmatrix} \delta_{x,k} \\ \delta_{y,k} \end{bmatrix} = - \begin{bmatrix} f(x_k, y_k) \\ g(x_k, y_k) \end{bmatrix}$$
$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ y_k \end{bmatrix} + \begin{bmatrix} \delta_{x,k} \\ \delta_{y,k} \end{bmatrix}, \quad k = 0, 1, \ldots \tag{4}$$

This is *Newton's method* for solving

$$\begin{aligned} f(x, y) &= 0 \\ g(x, y) &= 0 \end{aligned}$$

Many numerical methods for solving nonlinear systems are variations on Newton's method.

Example. Consider again the system

$$\begin{aligned} f(x,y) &\equiv x^2 + 4y^2 - 9 &&= 0 \\ g(x,y) &\equiv 18y - 14x^2 + 45 &&= 0 \end{aligned}$$

Newton's method (4) becomes

$$\begin{gathered} \begin{bmatrix} 2x_k & 8y_k \\ -28x_k & 18 \end{bmatrix} \begin{bmatrix} \delta_{x,k} \\ \delta_{y,k} \end{bmatrix} = - \begin{bmatrix} x_k^2 + 4y_k^2 - 9 \\ 18y_k - 14x_k^2 + 45 \end{bmatrix} \\ \begin{bmatrix} x_{k+1} \\ y_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ y_k \end{bmatrix} + \begin{bmatrix} \delta_{x,k} \\ \delta_{y,k} \end{bmatrix} \end{gathered} \tag{5}$$

Choose $(x_0, y_0) = (1, -1)$. The resulting Newton iterates are given in Table 1, along with

$$\text{Error } = \|\alpha - (x_k, y_k)\| \equiv \max\{|\xi - x_k|, |\eta - y_k|\}$$

Table 1: Newton iterates for the system (5)

k	x_k	y_k
0	1.0	-1.0
1	1.170212765957447	-1.457446808510638
2	1.202158829506705	-1.376760321923060
3	1.203165807091535	-1.374083486949713
4	1.203166963346410	-1.374080534243534
5	1.203166963347774	-1.374080534239942

k	*Error*
0	$3.74E-1$
1	$8.34E-2$
2	$2.68E-3$
3	$2.95E-6$
4	$3.59E-12$
5	$2.22E-16$

The final iterate is accurate to the precision of the computer arithmetic.

AN ALTERNATIVE NOTATION. We introduce a more general notation for the preceding work. The problem to be solved is

$$\begin{aligned} F_1(x_1, x_2) &= 0 \\ F_2(x_1, x_2) &= 0 \end{aligned} \tag{6}$$

Introduce

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \qquad F(x) = \begin{bmatrix} F_1(x_1, x_2) \\ F_2(x_1, x_2) \end{bmatrix}$$

$$F'(x) = \begin{bmatrix} \dfrac{\partial F_1}{\partial x_1} & \dfrac{\partial F_1}{\partial x_2} \\ \dfrac{\partial F_2}{\partial x_1} & \dfrac{\partial F_2}{\partial x_2} \end{bmatrix}$$

$F'(x)$ is called the *Frechet derivative* of $F(x)$. It is a generalization to higher dimensions of the ordinary derivative of a function of one variable.

The system (6) can now be written as

$$F(x) = 0 \tag{7}$$

A solution of this equation will be denoted by α.

Newton's method becomes

$$\begin{aligned} F'(x^{(k)})\delta^{(k)} &= -F(x^{(k)}) \\ x^{(k+1)} &= x^{(k)} + \delta^{(k)} \end{aligned} \tag{8}$$

for $k = 0, 1, \ldots$.

A shorter and mathematically equivalent form:

$$x^{(k+1)} = x^{(k)} - \left[F'(x^{(k)})\right]^{-1} F(x^{(k)}) \tag{9}$$

for $k = 0, 1, \ldots$.

This last formula is often used in discussing and analyzing Newton's method for nonlinear systems. But (8) is used for practical computations, since it is usually less expensive to solve a linear system than to find the inverse of the coefficient matrix.

Note the analogy of (9) with Newton's method for a single equation.

THE GENERAL NEWTON METHOD

Consider the system of n nonlinear equations

$$\begin{aligned} F_1(x_1, \ldots, x_n) &= 0 \\ &\vdots \\ F_n(x_1, \ldots, x_n) &= 0 \end{aligned} \tag{10}$$

Define

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \qquad F(x) = \begin{bmatrix} F_1(x_1, \ldots, x_n) \\ \vdots \\ F_n(x_1, \ldots, x_n) \end{bmatrix}$$

$$F'(x) = \begin{bmatrix} \dfrac{\partial F_1}{\partial x_1} & \cdots & \dfrac{\partial F_1}{\partial x_n} \\ \vdots & & \vdots \\ \dfrac{\partial F_n}{\partial x_1} & \cdots & \dfrac{\partial F_n}{\partial x_n} \end{bmatrix}$$

The nonlinear system (10) can be written as

$$F(x) = 0$$

Its solution is denoted by $\alpha \in \mathbb{R}^n$.

Newton's method is

$$\begin{aligned} F'(x^{(k)})\delta^{(k)} &= -F(x^{(k)}) \\ x^{(k+1)} &= x^{(k)} + \delta^{(k)}, \qquad k = 0, 1, \ldots \end{aligned} \tag{11}$$

Alternatively, as before,

$$x^{(k+1)} = x^{(k)} - \left[F'(x^{(k)})\right]^{-1} F(x^{(k)}), \quad k = 0, 1, \ldots \tag{12}$$

This formula is often used in theoretical discussions of Newton's method for nonlinear systems. But (11) is used for practical computations, since it is usually less expensive to solve a linear system than to find the inverse of the coefficient matrix.

CONVERGENCE. Under suitable hypotheses, it can be shown that there is a $c > 0$ for which

$$\left\|\alpha - x^{(k+1)}\right\| \le c \left\|\alpha - x^{(k)}\right\|^2, \quad k = 0, 1, \ldots$$

$$\left\|\alpha - x^{(k)}\right\| \equiv \max_{1 \le i \le n} \left|\alpha_i - x_i^{(k)}\right|$$

Newton's method is *quadratically convergent*.