

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:

$$p(x) = a_0 + a_1x + a_2x^2 + \dots a_{n-1}x^{n-1} + a_nx^n$$

Calculation of functions

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

$$p(x) = a_0 + a_1x + a_2x^2 + \dots + a_{n-1}x^{n-1} + a_nx^n$$

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:

$$p(x) = a_0 + a_1x + a_2x^2 + \dots + a_{n-1}x^{n-1} + a_nx^n$$

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 $P_1(x) = a_0 + a_1x$.

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 $P_1(x) = a_0 + a_1x$.

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.

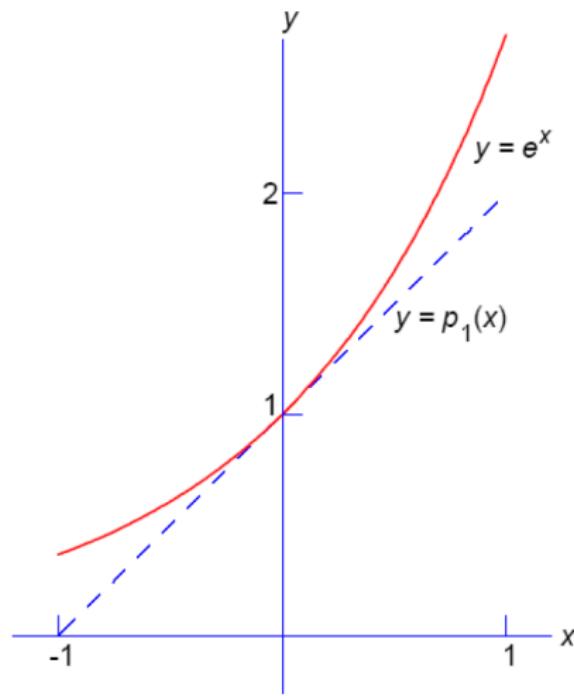
Look for a polynomial $p(x)$ whose values will be the same as those of e^x to within acceptable accuracy.

Begin with a linear polynomial $P_1(x) = a_0 + a_1x$.

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 $P_2(x) = a_0 + a_1x + a_2x^2$

Taylor polynomial approximations

Look for a quadratic polynomial $P_2(x) = a_0 + a_1x + a_2x^2$

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 $P_2(x) = a_0 + a_1x + a_2x^2$

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 $P_2(x) = a_0 + a_1x + a_2x^2$

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

$$P_n(x) = a_0 + a_1x + a_2x^2 + \dots a_{n-1}x^{n-1} + a_nx^n$$

Taylor polynomial approximations

We continue this pattern, looking for a polynomial

$$P_n(x) = a_0 + a_1x + a_2x^2 + \dots a_{n-1}x^{n-1} + a_nx^n$$

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

$$P_n(x) = a_0 + a_1x + a_2x^2 + \dots + a_{n-1}x^{n-1} + a_nx^n$$

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

$$P_n(x) = a_0 + a_1x + a_2x^2 + \dots + a_{n-1}x^{n-1} + a_nx^n$$

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

$$P_n(x) = a_0 + a_1x + a_2x^2 + \dots + a_{n-1}x^{n-1} + a_nx^n$$

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 $P_n(x)$:

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

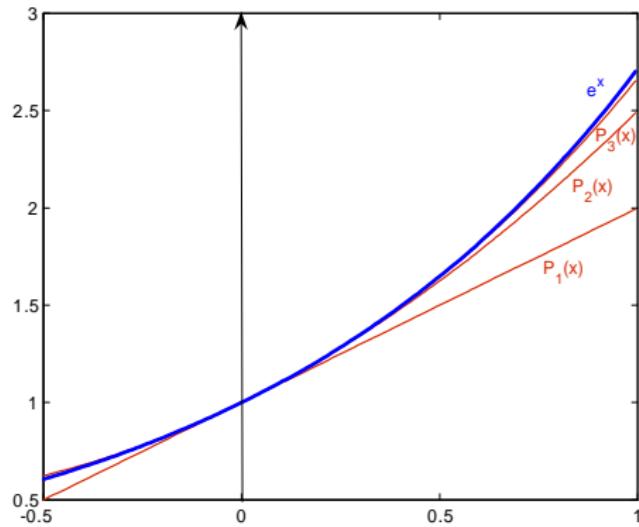
Moreover, as $n \rightarrow \infty$ it can be shown that $P_n(x) \rightarrow e^x$

Taylor polynomial approximations

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

Taylor polynomial approximations

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

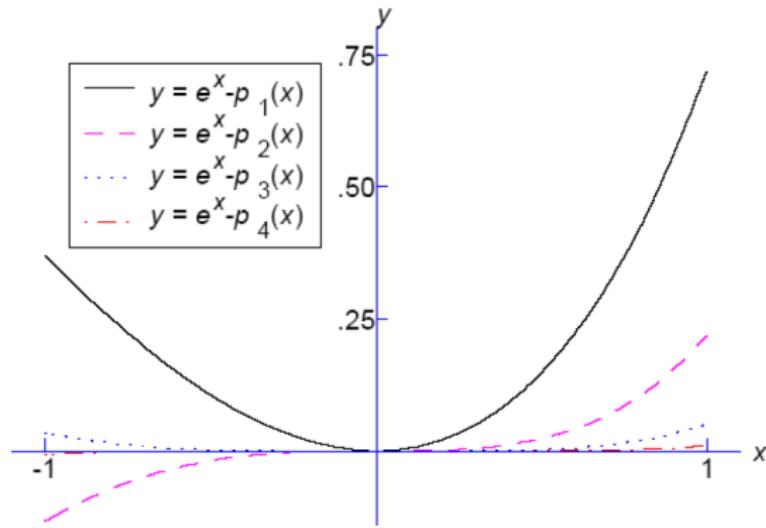


Taylor polynomial approximations

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 .

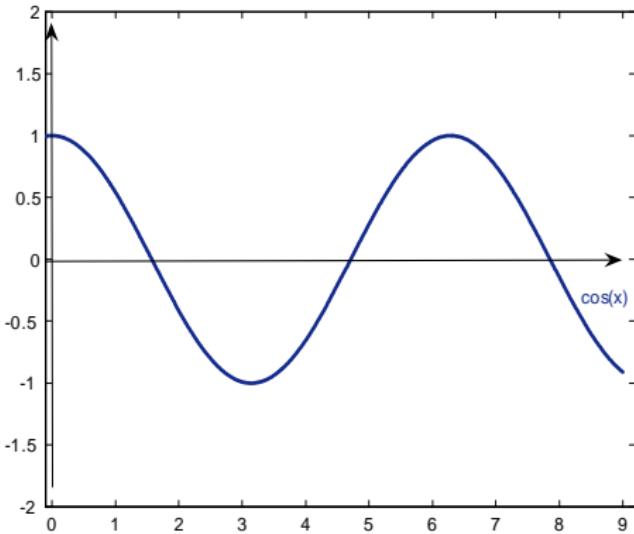
Note that for $x \approx 0$, we must have $\xi \approx 0$ and

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

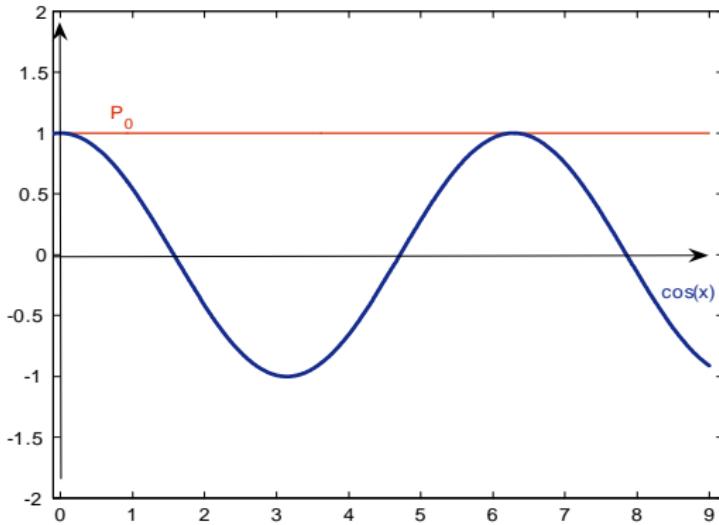
This last term is also the final term in $P_{n+1}(x)$, and thus

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

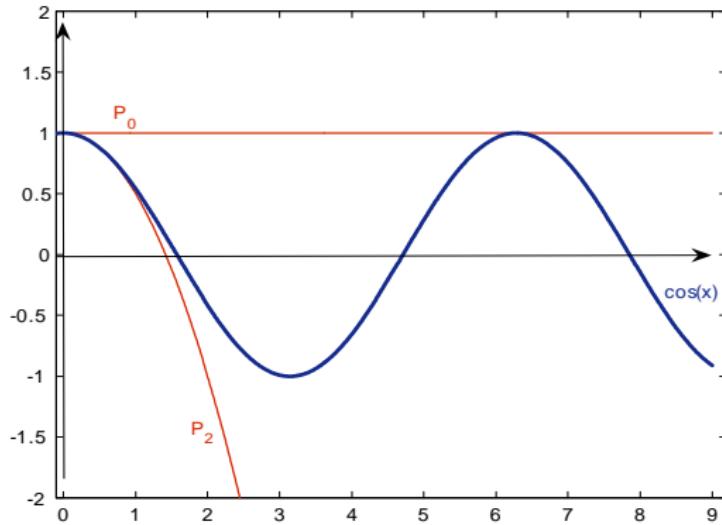
$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + \frac{x^{12}}{12!} - \frac{x^{14}}{14!} + \frac{x^{16}}{16!} - \frac{x^{18}}{18!} + \dots$$



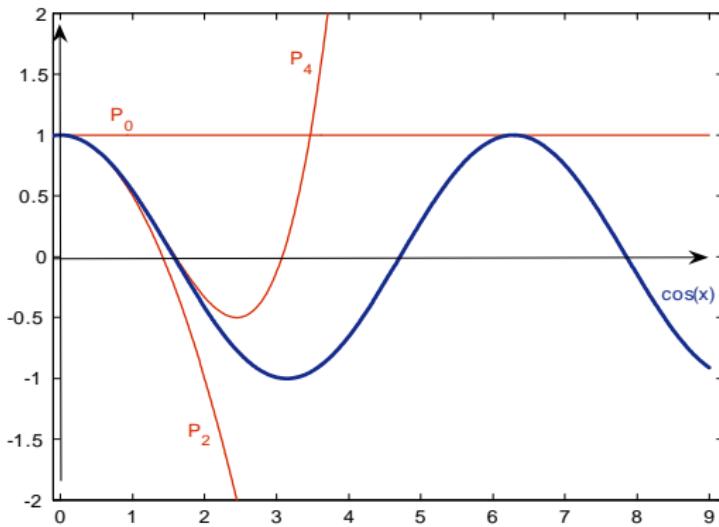
$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + \frac{x^{12}}{12!} - \frac{x^{14}}{14!} + \frac{x^{16}}{16!} - \frac{x^{18}}{18!} + \dots$$



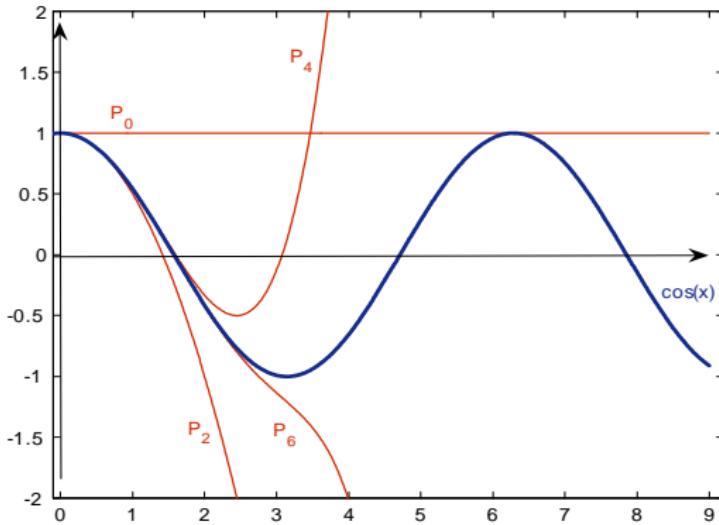
$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + \frac{x^{12}}{12!} - \frac{x^{14}}{14!} + \frac{x^{16}}{16!} - \frac{x^{18}}{18!} + \dots$$



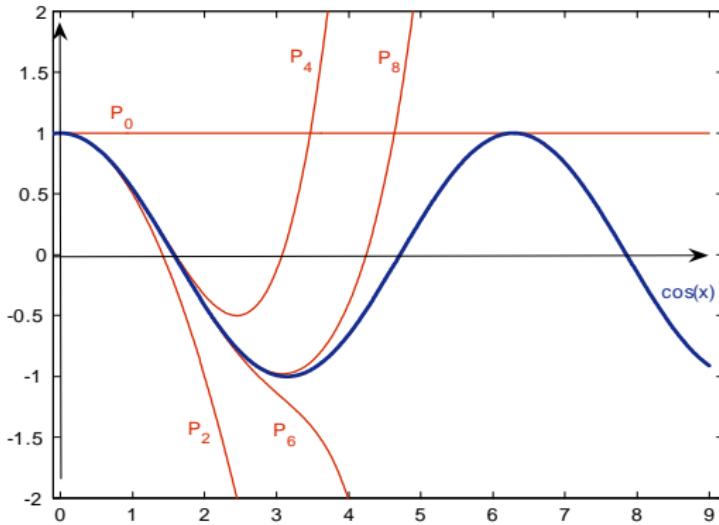
$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + \frac{x^{12}}{12!} - \frac{x^{14}}{14!} + \frac{x^{16}}{16!} - \frac{x^{18}}{18!} + \dots$$



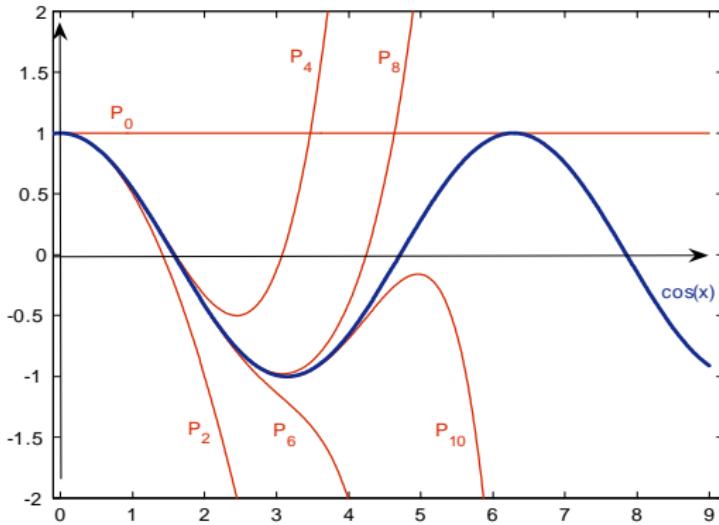
$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + \frac{x^{12}}{12!} - \frac{x^{14}}{14!} + \frac{x^{16}}{16!} - \frac{x^{18}}{18!} + \dots$$



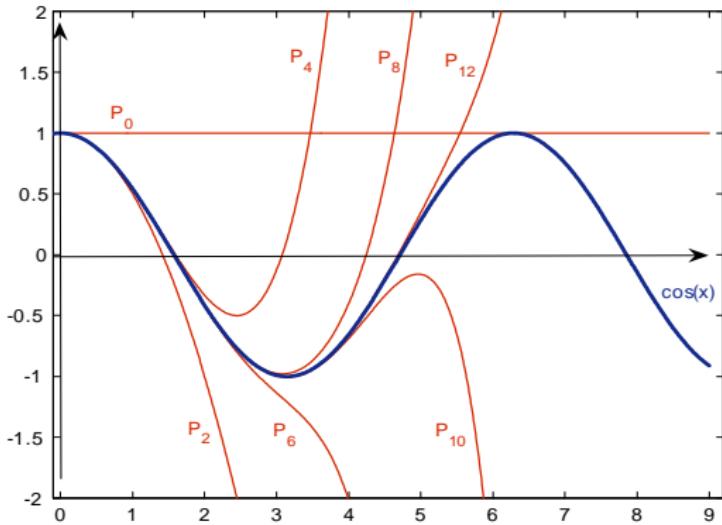
$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + \frac{x^{12}}{12!} - \frac{x^{14}}{14!} + \frac{x^{16}}{16!} - \frac{x^{18}}{18!} + \dots$$



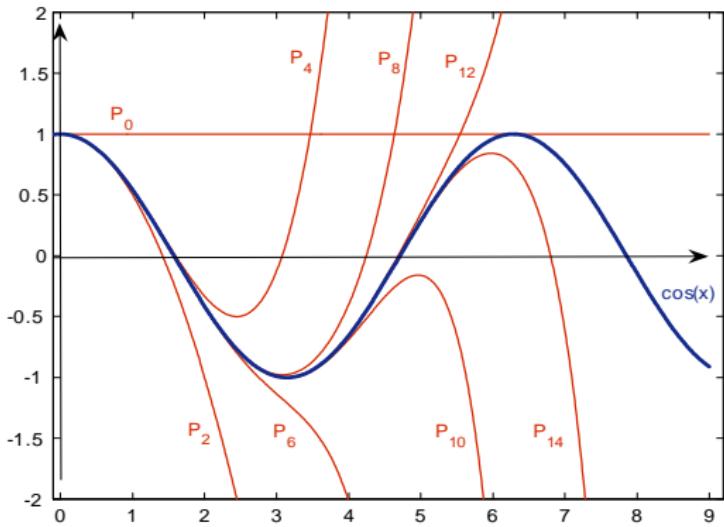
$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + \frac{x^{12}}{12!} - \frac{x^{14}}{14!} + \frac{x^{16}}{16!} - \frac{x^{18}}{18!} + \dots$$



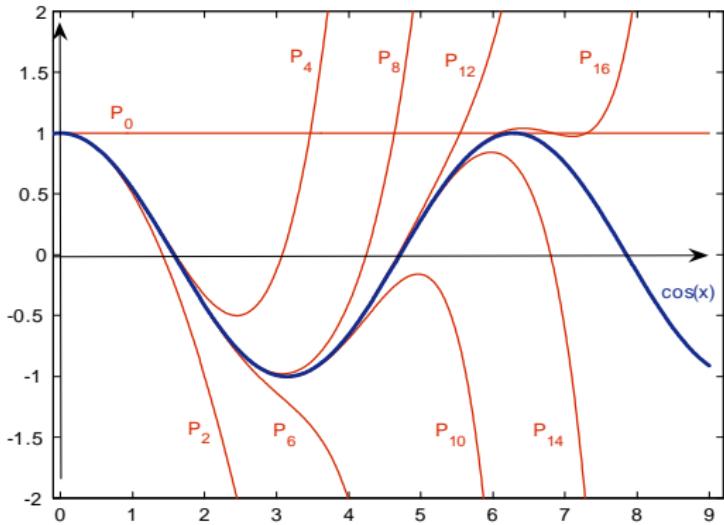
$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + \frac{x^{12}}{12!} - \frac{x^{14}}{14!} + \frac{x^{16}}{16!} - \frac{x^{18}}{18!} + \dots$$



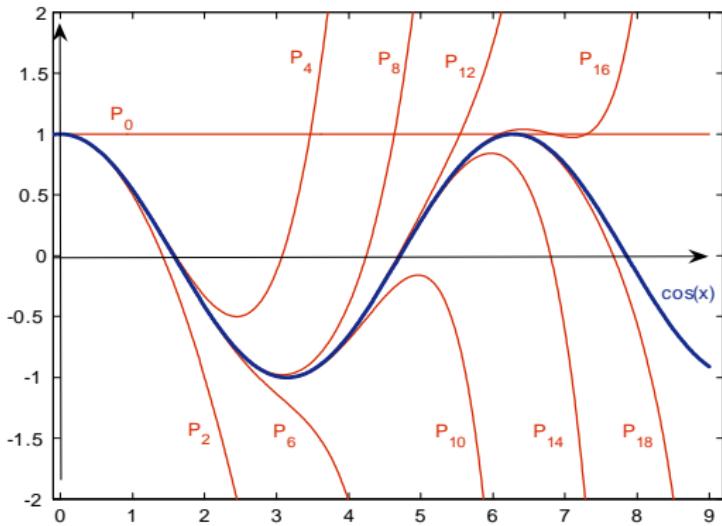
$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + \frac{x^{12}}{12!} - \frac{x^{14}}{14!} + \frac{x^{16}}{16!} - \frac{x^{18}}{18!} + \dots$$



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

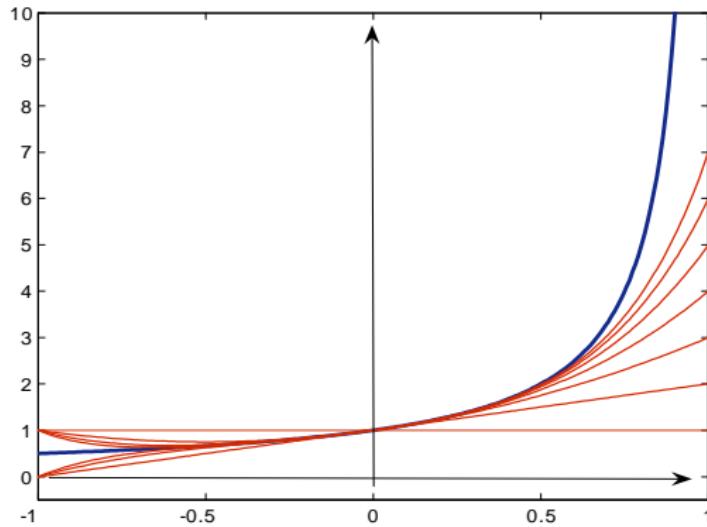


$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + \frac{x^{12}}{12!} - \frac{x^{14}}{14!} + \frac{x^{16}}{16!} - \frac{x^{18}}{18!} + \dots$$



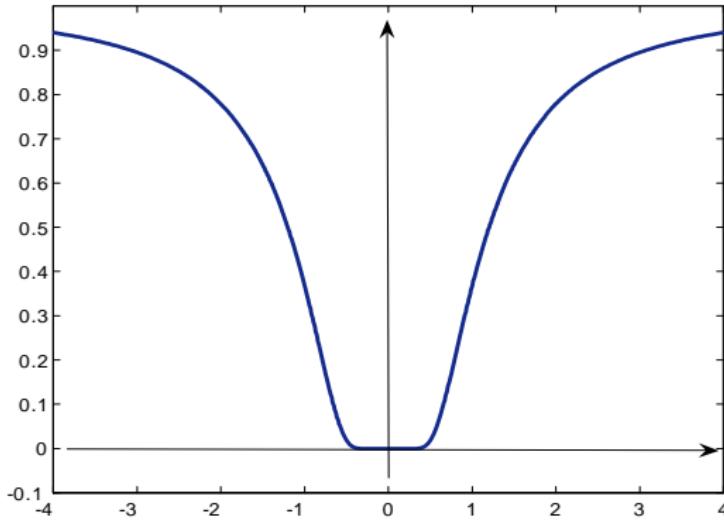
Taylor polynomial approximations

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



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

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



Computing approximation to e

Coming back to earlier approximation

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

Computing approximation to e

Coming back to earlier approximation

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

Then let $x = 1$ to get

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

Computing approximation to e

Coming back to earlier approximation

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

Then let $x = 1$ to get

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

For the error,

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

Computing approximation to e

Coming back to earlier approximation

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

Then let $x = 1$ to get

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

For the error,

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

To bound the error, we have

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

Computing approximation to e

Coming back to earlier approximation

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

Then let $x = 1$ to get

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

For the error,

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

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 $P_8(1)$:

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

Taylor polynomial approximations

In fact,

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

Then calculate $P_8(1)$:

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

and true error is

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

Evaluation of polynomials

Consider having a polynomial

$$p(x) = a_0 + a_1x + a_2x^2 + \dots a_{n-1}x^{n-1} + a_nx^n$$

Evaluation of polynomials

Consider having a polynomial

$$p(x) = a_0 + a_1x + a_2x^2 + \dots a_{n-1}x^{n-1} + a_nx^n$$

which you need to evaluate for many values of x .

Evaluation of polynomials

Consider having a polynomial

$$p(x) = a_0 + a_1x + a_2x^2 + \dots a_{n-1}x^{n-1} + a_nx^n$$

which you need to evaluate for many values of x .

How do you evaluate it?

Evaluation of polynomials

Consider having a polynomial

$$p(x) = a_0 + a_1x + a_2x^2 + \dots a_{n-1}x^{n-1} + a_nx^n$$

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:

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

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 $\{x^2, x^3, \dots, x^n\}$ 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 $\{x^2, x^3, \dots, x^n\}$ 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 $\{x^2, x^3, \dots, x^n\}$ 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

additions : n

multiplications : $n + n - 1 = 2n - 1$

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

$$n = 2 : p(x) = a_0 + x(a_1 + a_2x)$$

$$n = 3 : p(x) = a_0 + x(a_1 + x(a_2 + a_3x))$$

$$n = 4 : p(x) = a_0 + x(a_1 + x(a_2 + x(a_3 + a_4x)))$$

Evaluation of polynomials

Consider **nested multiplication**.

As examples of particular degrees, write

$$n = 2 : p(x) = a_0 + x(a_1 + a_2x)$$

$$n = 3 : p(x) = a_0 + x(a_1 + x(a_2 + a_3x))$$

$$n = 4 : p(x) = a_0 + x(a_1 + x(a_2 + x(a_3 + a_4x)))$$

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

Evaluation of polynomials

Consider **nested multiplication**.

As examples of particular degrees, write

$$n = 2 : p(x) = a_0 + x(a_1 + a_2x)$$

$$n = 3 : p(x) = a_0 + x(a_1 + x(a_2 + a_3x))$$

$$n = 4 : p(x) = a_0 + x(a_1 + x(a_2 + x(a_3 + a_4x)))$$

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

$$n = 2 : p(x) = a_0 + x(a_1 + a_2x)$$

$$n = 3 : p(x) = a_0 + x(a_1 + x(a_2 + a_3x))$$

$$n = 4 : p(x) = a_0 + x(a_1 + x(a_2 + x(a_3 + a_4x)))$$

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

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

Evaluation of polynomials

For the general case, write

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

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

Evaluation of polynomials

For the general case, write

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

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

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

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 .

Approximating function SF(x)

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

Approximating function SF(x)

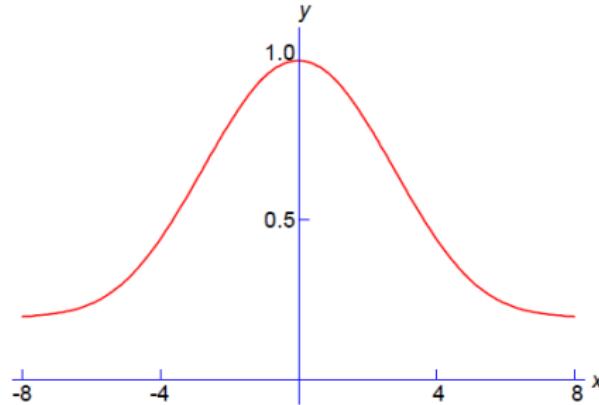
$$SF(x) = \frac{1}{x} \int_0^x \frac{\sin t}{t} dt, \quad x \neq 0$$

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

Approximating function SF(x)

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

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



Approximating function SF(x)

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

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

Approximating function SF(x)

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

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

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

Approximating function SF(x)

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

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

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

Approximating function SF(x)

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

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

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

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

$$\int_0^x \frac{\sin t}{t} dt = \int_0^x 1 dt - \frac{1}{6} \int_0^x t^2 dt + \frac{1}{120} \int_0^x t^4 \cos \xi dt$$

Approximating function SF(x)

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

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

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

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

$$\int_0^x \frac{\sin t}{t} dt = \int_0^x 1 dt - \frac{1}{6} \int_0^x t^2 dt + \frac{1}{120} \int_0^x t^4 \cos \xi dt$$

$$\int_0^x \frac{\sin t}{t} dt = x - \frac{1}{18}x^3 + \frac{1}{120} \int_0^x t^4 \cos \xi dt$$

Approximating function SF(x)

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

Approximating function SF(x)

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

Approximating 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

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

Approximating 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 $x \in [-1, 1]$?

Approximating function SF(x)

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

Approximating function SF(x)

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

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

Approximating 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 $x < 0$.

Approximating 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

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

Approximating 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

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

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

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

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

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

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

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

Number σ is called **sign**, e is **exponent**, and \bar{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

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

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

Number σ is called **sign**, e is **exponent**, and \bar{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

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

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

Number σ is called **sign**, e is **exponent**, and \bar{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 \bar{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 \bar{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 \bar{x} and the size of the exponent e .

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

Decimal floating point numbers

Computers have a finite space for storage.

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

The number of digits in \bar{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

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

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

Binary floating point numbers

Write

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

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

For example,

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

Binary floating point numbers

Write

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

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

For example,

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

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

Binary floating point numbers

Write

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

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

For example,

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

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

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

Binary floating point numbers

Write

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

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

For example,

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

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

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

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 precision 10,

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

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 precision 10,

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

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 precision 10,

$$fl\left(\frac{1}{3}\right) = fl(0.3333333\dots) = (+1) \cdot (3.333333333)_10 \cdot 10^{-1}$$

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 $f(x)$.

On a calculator with precision 10,

$$f\left(\frac{1}{3}\right) = f(0.3333333\dots) = (+1) \cdot (3.333333333)_10 \cdot 10^{-1}$$

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 $f(x)$.

On a calculator with precision 10,

$$f\left(\frac{1}{3}\right) = f(0.3333333\dots) = (+1) \cdot (3.333333333)_10 \cdot 10^{-1}$$

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 precision 10,

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

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

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

Binary floating point numbers

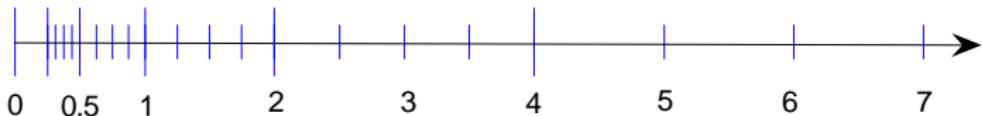


Binary floating point numbers



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

$$(0.01)_2 \cdot 2^{-1} = \frac{1}{16} = (0.0625)_{10}$$

$$(0.10)_2 \cdot 2^{-1} = \frac{2}{16} = (0.125)_{10}$$

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

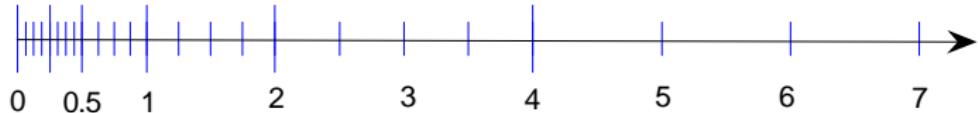
Binary floating point numbers

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

$$x = \sigma \cdot 1.a_1a_2 \dots a_{23} \cdot 2^e.$$

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

$$x = \sigma \cdot 1.a_1a_2 \dots a_{23} \cdot 2^e.$$

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

$$x = \sigma \cdot 1.a_1a_2 \dots a_{23} \cdot 2^e.$$

Basically, we store σ as a single bit, the significand \bar{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

$$x = \sigma \cdot 1.a_1a_2 \dots a_{23} \cdot 2^e.$$

Basically, we store σ as a single bit, the significand \bar{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				\bar{x}			
b_1	b_2	\dots	b_9	b_{10}	\dots	b_{30}	\dots	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				\bar{x}			
b_1	b_2	\dots	b_9	b_{10}	\dots	b_{32}		

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

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
$(11111101)_2 = (253)_{10}$	$(126)_{10}$	$\pm(1.b_{10} \dots b_{32})_2 \cdot 2^{126}$
$(11111110)_2 = (254)_{10}$	$(127)_{10}$	$\pm(1.b_{10} \dots b_{32})_2 \cdot 2^{127}$
$(11111111)_2 = (255)_{10}$	$(128)_{10}$	$\pm\infty, \text{ dacă } b_i = 0$ $NaN, \text{ otherwise}$

IEEE double precision floating point representation

$$x = \sigma \cdot 1.a_1a_2 \dots a_{52} \cdot 2^e.$$

with $E = e + 1023$

σ	E				\bar{x}		
b_1	b_2	\dots	b_{12}	b_{13}	\dots	b_{64}	

IEEE double precision floating point representation

$E = (b_2 \dots b_{12})_2$	e	x
$(000000000000)_2 = (0)_{10}$	$-(1023)_{10}$	$\pm(0.b_{13} \dots b_{64})2^{-1022}$
$(000000000001)_2 = (1)_{10}$	$-(1022)_{10}$	$\pm(1.b_{13} \dots b_{64})2^{-1022}$
$(000000000010)_2 = (2)_{10}$	$-(1021)_{10}$	$\pm(1.b_{13} \dots b_{64})2^{-1021}$
⋮	⋮	⋮
$(011111111111)_2 = (1023)_{10}$	$(0)_{10}$	$\pm(1.b_{13} \dots b_{64})2^0$
$(100000000000)_2 = (1024)_{10}$	$(1)_{10}$	$\pm(1.b_{13} \dots b_{64})2^1$
⋮	⋮	⋮
$(111111111101)_2 = (2045)_{10}$	$(1022)_{10}$	$\pm(1.b_{13} \dots b_{64})2^{1022}$
$(111111111110)_2 = (2046)_{10}$	$(1023)_{10}$	$\pm(1.b_{13} \dots b_{64})2^{1023}$
$(111111111111)_2 = (2047)_{10}$	$(1024)_{10}$	$\pm\infty, \quad b_i = 0$ $NaN, \quad otherwise$

Characteristics of floating point representation

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

Characteristics of floating point representation

What is the connection of the 24 bits in the significand \bar{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 $fl(x) = x$

Characteristics of floating point representation

What is the connection of the 24 bits in the significand \bar{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 $fl(M + 1) \neq M + 1$

Characteristics of floating point representation

What is the connection of the 24 bits in the significand \bar{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 $fl(M + 1) \neq M + 1$

Characteristics of floating point representation

What is the connection of the 24 bits in the significand \bar{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

$$(\underbrace{111\dots11}_\text{23 ones})_2$$

Characteristics of floating point representation

What is the connection of the 24 bits in the significand \bar{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

$$\underbrace{(111\dots11)_2}_{23 \text{ ones}} = (1.11\dots1)_2 \cdot 2^{23}$$

Characteristics of floating point representation

What is the connection of the 24 bits in the significand \bar{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

$$\begin{aligned} (\underbrace{111\dots11}_{23 \text{ ones}})_2 &= (1.11\dots1)_2 \cdot 2^{23} \\ &= 2^{23} + 2^{22} + \dots + 2^0 \end{aligned}$$

Characteristics of floating point representation

What is the connection of the 24 bits in the significand \bar{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

$$\begin{aligned} (\underbrace{111\dots11}_{23 \text{ ones}})_2 &= (1.11\dots1)_2 \cdot 2^{23} \\ &= 2^{23} + 2^{22} + \dots + 2^0 \\ &= 2^{24} - 1 \end{aligned}$$

Characteristics of floating point representation

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

Characteristics of floating point representation

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

$$2^{24} = (1.00\dots00)_2 \cdot 2^{24}$$

Characteristics of floating point representation

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

$$2^{24} = (1.00\dots00)_2 \cdot 2^{24}$$

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

Characteristics of floating point representation

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

$$2^{24} = (1.00\dots00)_2 \cdot 2^{24}$$

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

$$2^{24} + 1 = (1.\underbrace{00\dots01}_\text{23 zeros})_2 \cdot 2^{24}$$

Characteristics of floating point representation

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

$$2^{24} = (1.00\dots00)_2 \cdot 2^{24}$$

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

$$2^{24} + 1 = (1.\underbrace{00\dots01}_\text{23 zeros})_2 \cdot 2^{24}$$

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

Characteristics of floating point representation

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

$$2^{24} = (1.00\dots00)_2 \cdot 2^{24}$$

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

$$2^{24} + 1 = (1.\underbrace{00\dots01}_\text{23 zeros})_2 \cdot 2^{24}$$

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

Characteristics of floating point representation

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

$$2^{24} = (1.00\dots00)_2 \cdot 2^{24}$$

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

$$2^{24} + 1 = (1.\underbrace{00\dots01}_\text{23 zeros})_2 \cdot 2^{24}$$

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

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

Characteristics of floating point representation

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

$$2^{24} = (1.00\dots00)_2 \cdot 2^{24}$$

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

$$2^{24} + 1 = (1.\underbrace{00\dots01}_\text{23 zeros})_2 \cdot 2^{24}$$

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

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

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

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

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

The number 1 has the simple floating point representation

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

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

$$1 + 2^{-23} = (1.0\dots 01)_2 \cdot 2^0 > 1$$

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

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

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

$$1 + 2^{-23} = (1.0\dots 01)_2 \cdot 2^0 > 1$$

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

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

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

$$1 + 2^{-23} = (1.0\dots 01)_2 \cdot 2^0 > 1$$

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

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

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

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

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

Given a general number x

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

Chopping and rounding in decimal

We write a computer floating point number z as

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

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

Given a general number x

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

we must shorten it to fit within the computer.

Chopping and rounding in decimal

We write a computer floating point number z as

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

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

Given a general number x

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

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

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

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

Given a general number x

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

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

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

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

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

Chopping and rounding in decimal

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

A simplified formula is

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

The term $(0.00 \dots 01)_{10}$ 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

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

with all a_i equal to 0 or 1.

Chopping and rounding in binary

Let

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

with all a_i equal to 0 or 1.

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

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

Chopping and rounding in binary

Let

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

with all a_i equal to 0 or 1.

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

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

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

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

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

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

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

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

$$-\frac{1}{2^{n-1}} \leq \varepsilon \leq 0, \quad \text{chopping}$$

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

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

$$-\frac{1}{2^{n-1}} \leq \varepsilon \leq 0, \quad \text{chopping}$$

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	<i>a</i>	1010
3	0011	<i>b</i>	1011
4	0100	<i>c</i>	1100
5	0101	<i>d</i>	1101
6	0110	<i>e</i>	1110
7	0111	<i>f</i>	1111

MATLAB floating point representation

401c800000000000

MATLAB floating point representation

401c800000000000

0100000000001110010000000000...0000

4 0 1 c 8 0 0 0

MATLAB floating point representation

401c800000000000

0100000000001110010000000000...0000

4 0 1 c 8 0 0 0

σ	E				\bar{x}			
b_1	b_2	\dots	b_{12}	b_{13}	\dots	b_{63}	b_{64}	

MATLAB floating point representation

401c800000000000

0100000000011100100000000000...0000

4 0 1 c 8 0 0 0

σ	E				\bar{x}		
b_1	b_2	\dots	b_{12}	b_{13}	\dots	b_{64}	

0 1000000000111001000000000000...0000

σ E $1.b_{13}b_{14}\dots b_{64} = \bar{x}$

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:

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

$$err(x_A) = e - \frac{19}{7} \approx 0.003996$$

$$rel(x_A) \approx \frac{0.003996}{e} \approx 0.00147$$

Errors in floating point representation

Errors in floating point representation

Example: Suppose the distance between two cities is $D_T = 100\text{km}$

Errors in floating point representation

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

Errors in floating point representation

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

In this case,

$$\text{err}(D_A) = D_T - D_A = 1\text{km},$$

Errors in floating point representation

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

In this case,

$$\text{err}(D_A) = D_T - D_A = 1\text{km},$$

$$\text{rel}(D_A) = \frac{\text{err}(D_A)}{D_T} = \frac{1}{100} = 0.01 = 1\%$$

Errors in floating point representation

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

In this case,

$$\text{err}(D_A) = D_T - D_A = 1\text{km},$$

$$\text{rel}(D_A) = \frac{\text{err}(D_A)}{D_T} = \frac{1}{100} = 0.01 = 1\%$$

Now, suppose that distance is $d_T = 2\text{km}$

Errors in floating point representation

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

In this case,

$$\text{err}(D_A) = D_T - D_A = 1\text{km},$$

$$\text{rel}(D_A) = \frac{\text{err}(D_A)}{D_T} = \frac{1}{100} = 0.01 = 1\%$$

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

Errors in floating point representation

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

In this case,

$$\text{err}(D_A) = D_T - D_A = 1\text{km},$$

$$\text{rel}(D_A) = \frac{\text{err}(D_A)}{D_T} = \frac{1}{100} = 0.01 = 1\%$$

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

$$\text{err}(d_A) = d_T - d_A = 1\text{km},$$

Errors in floating point representation

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

In this case,

$$\text{err}(D_A) = D_T - D_A = 1\text{km},$$

$$\text{rel}(D_A) = \frac{\text{err}(D_A)}{D_T} = \frac{1}{100} = 0.01 = 1\%$$

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

$$\text{err}(d_A) = d_T - d_A = 1\text{km},$$

$$\text{rel}(d_A) = \frac{\text{err}(d_A)}{d_T} = \frac{1}{2} = 0.5 = 50\%$$

Errors in floating point representation

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

In this case,

$$\text{err}(D_A) = D_T - D_A = 1\text{km},$$

$$\text{rel}(D_A) = \frac{\text{err}(D_A)}{D_T} = \frac{1}{100} = 0.01 = 1\%$$

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

$$\text{err}(d_A) = d_T - d_A = 1\text{km},$$

$$\text{rel}(d_A) = \frac{\text{err}(d_A)}{d_T} = \frac{1}{2} = 0.5 = 50\%$$

In both cases the error is the same.

Errors in floating point representation

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

In this case,

$$\text{err}(D_A) = D_T - D_A = 1\text{km},$$

$$\text{rel}(D_A) = \frac{\text{err}(D_A)}{D_T} = \frac{1}{100} = 0.01 = 1\%$$

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

$$\text{err}(d_A) = d_T - d_A = 1\text{km},$$

$$\text{rel}(d_A) = \frac{\text{err}(d_A)}{d_T} = \frac{1}{2} = 0.5 = 50\%$$

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 through 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 through the earth's atmosphere. A simple and often 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 through the earth's atmosphere. A simple and often used description of projectile motion is given by

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

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 through the earth's atmosphere. A simple and often used description of projectile motion is given by

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

with $b \geq 0$. In this, $\vec{r}(t)$ 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 through the earth's atmosphere. A simple and often used description of projectile motion is given by

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

with $b \geq 0$. In this, $\vec{r}(t)$ 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 through the earth's atmosphere. A simple and often used description of projectile motion is given by

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

with $b \geq 0$. In this, $\vec{r}(t)$ 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 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

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

Sources of Error

2. Physical / Observational / Measurement Error.

The radius of an electron is given by

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

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

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

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. Finiteness of precision errors

Sources of Error

7. Finiteness of precision errors

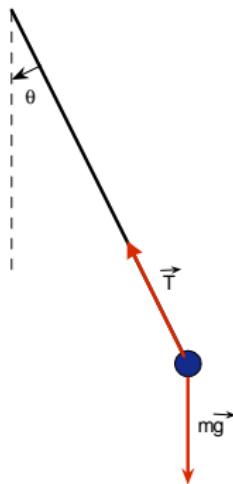
All the numbers stored in computer memory are subject to the finiteness of allocated space for storage.

Pendulum Example

Original problem in engineering or in science to be solved:

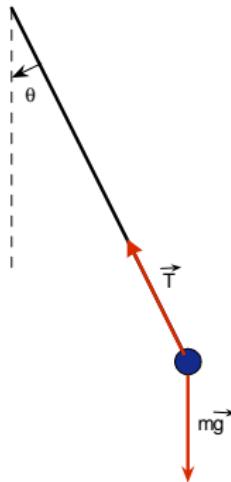
Pendulum Example

Original problem in engineering or in science to be solved:



Pendulum Example

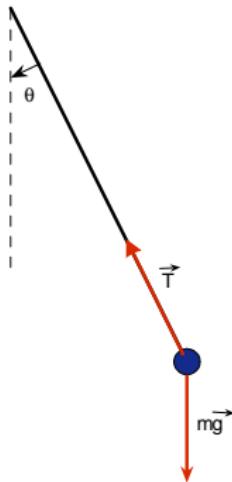
Original problem in engineering or in science to be solved:



Model this physical problem mathematically.

Pendulum Example

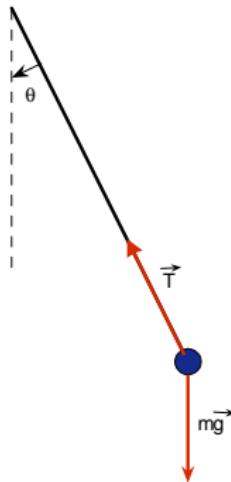
Original problem in engineering or in science to be solved:



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

Pendulum Example

Original problem in engineering or in science to be solved:

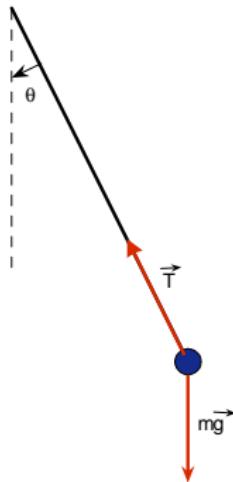


Model this physical problem mathematically.
Second Newton law provides us with:

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

Pendulum Example

Original problem in engineering or in science to be solved:



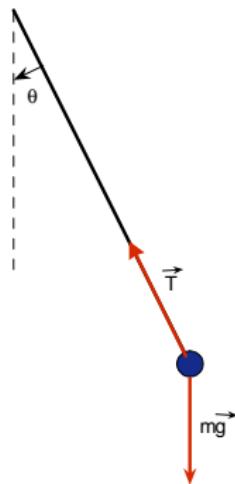
Model this physical problem mathematically.
Second Newton law provides us with:

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

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

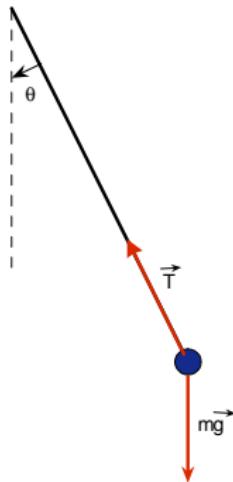
Pendulum Example

Problem of continuous mathematics:



Pendulum Example

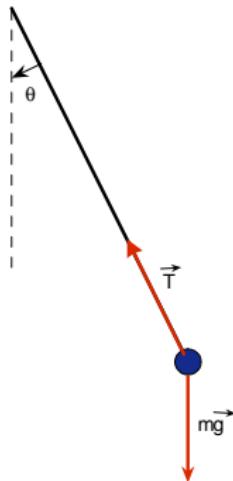
Problem of continuous mathematics:



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

Pendulum Example

Problem of continuous mathematics:

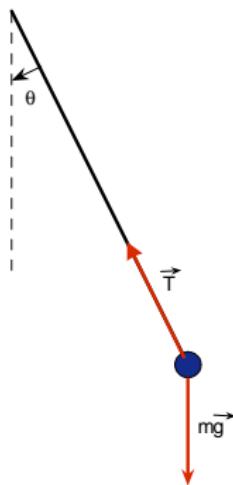


$$\begin{cases} \dot{\theta} = \omega \\ \dot{\omega} = -\frac{g}{l} \sin \theta \end{cases}$$

- Modeling Errors
- Physical Errors

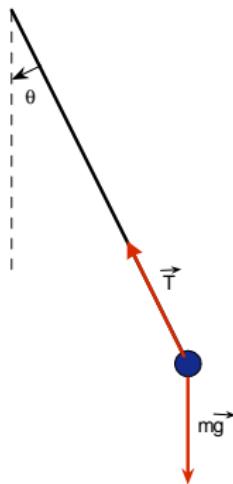
Pendulum Example

Mathematical Algorithms:



Pendulum Example

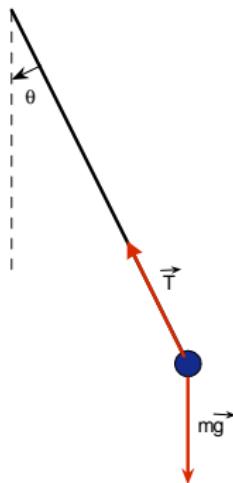
Mathematical Algorithms:



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

Pendulum Example

Mathematical Algorithms:

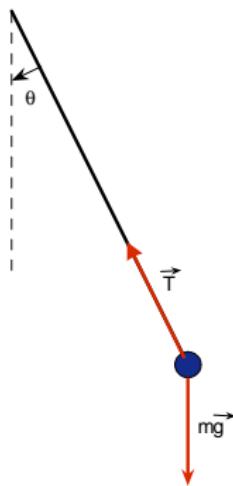


$$\begin{cases} \theta_{n+1} = \theta_n + h\omega_{n+1} \\ \omega_{n+1} = \omega_n - h \frac{g}{l} \sin(\theta_n) \end{cases}$$

- Discretisation Errors
- Finiteness of Algorithm Errors

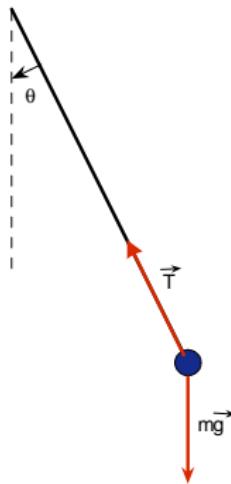
Pendulum Example

Computer Implementation:



Pendulum Example

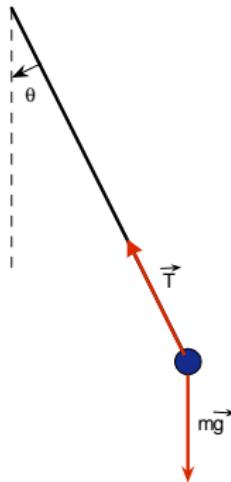
Computer Implementation:



```
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 $|\text{err}(x_A)| \leq 5$ 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 $|err(x_A)| \leq 5$ units in the $(m+1)$ -st digit, beginning with the first nonzero digit in x_T .

Example

Let

$$x_T = e = 2.71828182845904523\dots,$$

$$x_A = \frac{19}{7} = 2.714285714285714285\dots,$$

$$err(x_A) \approx 0.003996\dots$$

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

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

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

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

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

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.

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

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.

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.

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.

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 .

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

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

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

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

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.

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.

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.

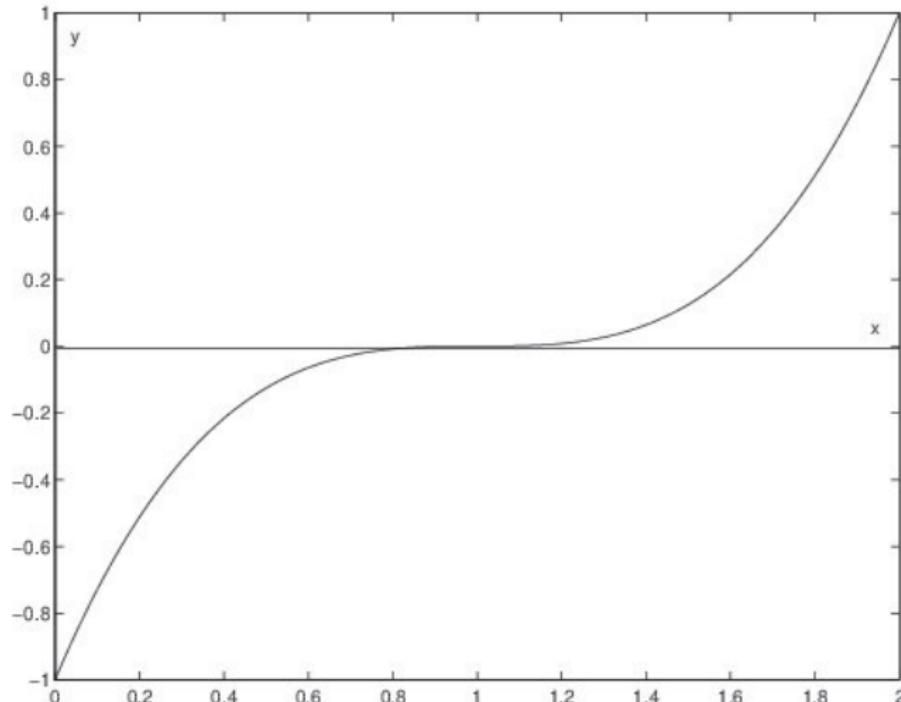
General principle

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.

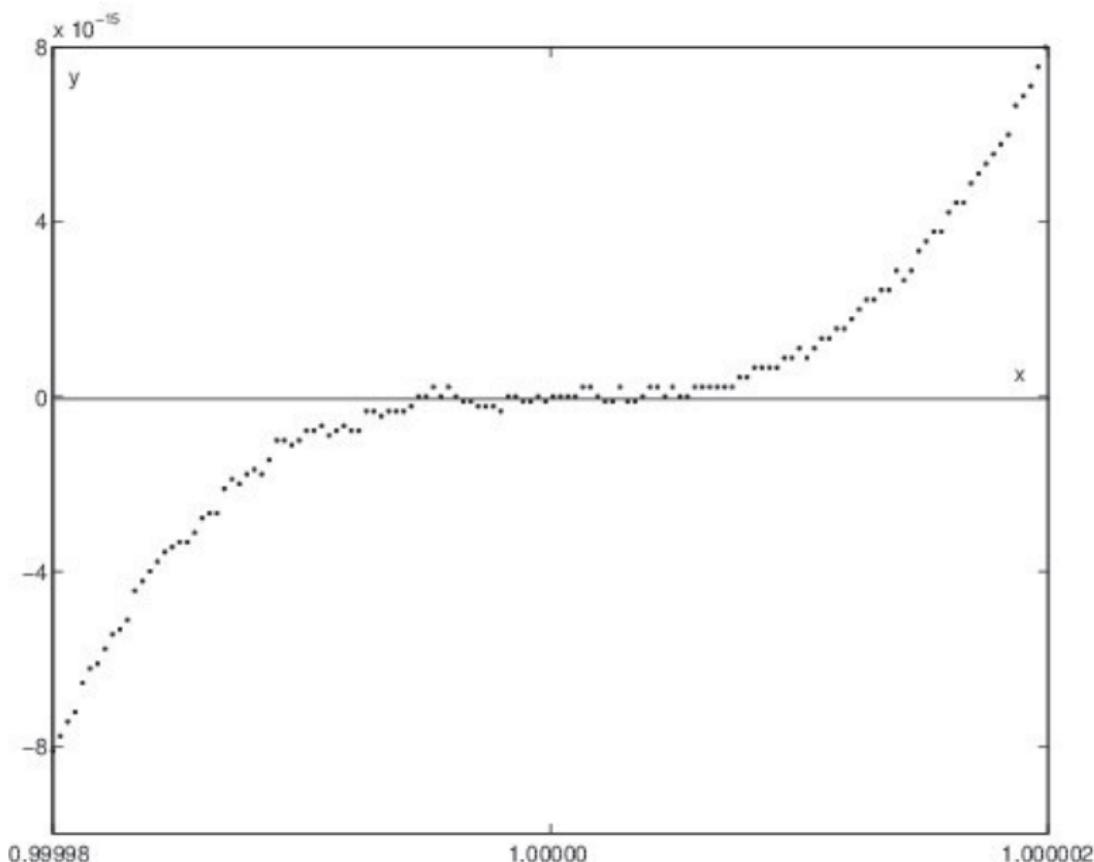
Noise in function evaluation

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



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

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.

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.

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

$$x^{10} < m,$$

$$|x| < m^{\frac{1}{10}},$$

$$|x| < 1.61 \times 10^{-4},$$

$$-0.000161 < x < 0.000161.$$

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

$$x^{10} > m,$$

$$|x| > m^{\frac{1}{10}},$$

$$|x| > 7131.6$$

Propagation of arithmetic operations errors



Let ω denote arithmetic operation such as $+$, $-$, $*$, or $/$.

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 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 $y_A \approx y_T$.

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 $x_A \omega^* y_A$.

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 $x_T \omega y_T - x_A \omega^* y_A$.

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:

$$x_T \omega y_T - x_A \omega^* y_A = [x_T \omega y_T - x_A \omega y_A] + [x_A \omega y_A - x_A \omega^* y_A]$$

The last term is the error introduced by the inexactness of the machine arithmetic, since it can be shown that

$$\text{Rel}(x_A \omega^* y_A) = -\varepsilon.$$

With rounded binary arithmetic having n digits in the mantissa,

$$-2^{-n} \leq \varepsilon \leq 2^{-n}.$$

With rounded binary arithmetic having n digits in the mantissa,

$$-2^{-n} \leq \varepsilon \leq 2^{-n}.$$

Coming back to error formula we have

$$x_T \omega y_T - x_A \omega^* y_A = [x_T \omega y_T - x_A \omega y_A] + \underbrace{[x_A \omega y_A - x_A \omega^* y_A]}_{\text{Relative error is } -\varepsilon}.$$

With rounded binary arithmetic having n digits in the mantissa,

$$-2^{-n} \leq \varepsilon \leq 2^{-n}.$$

Coming back to error formula we have

$$x_T \omega y_T - x_A \omega^* y_A = [x_T \omega y_T - x_A \omega y_A] + \underbrace{[x_A \omega y_A - x_A \omega^* y_A]}_{\text{Relative error is } -\varepsilon}.$$

The first term from the right side

$$x_T \omega y_T - x_A \omega y_A$$

is called the **propagated error**.

With rounded binary arithmetic having n digits in the mantissa,

$$-2^{-n} \leq \varepsilon \leq 2^{-n}.$$

Coming back to error formula we have

$$x_T \omega y_T - x_A \omega^* y_A = [x_T \omega y_T - x_A \omega y_A] + \underbrace{[x_A \omega y_A - x_A \omega^* y_A]}_{\text{Relative error is } -\varepsilon}.$$

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.

With rounded binary arithmetic having n digits in the mantissa,

$$-2^{-n} \leq \varepsilon \leq 2^{-n}.$$

Coming back to error formula we have

$$x_T \omega y_T - x_A \omega^* y_A = [x_T \omega y_T - x_A \omega y_A] + \underbrace{[x_A \omega y_A - x_A \omega^* y_A]}_{\text{Relative error is } -\varepsilon}.$$

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

$$x_T = x_A + \xi, \quad y_T = y_A + \eta,$$

where ξ and η are the approximation errors in x_A and y_A .

Then for the relative error in $x_A \cdot y_A$

$$\begin{aligned}\text{Rel}(x_A * y_A) &= \frac{x_T * y_T - x_A * y_A}{x_T * y_T} \\&= \frac{x_T * y_T - (x_T - \xi) * (y_T - \eta)}{x_T * y_T} \\&= \frac{x_T \eta + y_T \xi - \xi \eta}{x_T * y_T} \\&= \frac{\xi}{x_T} + \frac{\eta}{y_T} - \frac{\xi}{x_T} \cdot \frac{\eta}{y_T} \\&= \text{Rel}(x_A) + \text{Rel}(y_A) - \text{Rel}(x_A) \cdot \text{Rel}(y_A).\end{aligned}$$

Then for the relative error in $x_A \cdot y_A$

$$\begin{aligned}\text{Rel}(x_A \cdot y_A) &= \frac{x_T \cdot y_T - x_A \cdot y_A}{x_T \cdot y_T} \\ &= \frac{x_T \cdot y_T - (x_T - \xi) \cdot (y_T - \eta)}{x_T \cdot y_T} \\ &= \frac{x_T \eta + y_T \xi - \xi \eta}{x_T \cdot y_T} \\ &= \frac{\xi}{x_T} + \frac{\eta}{y_T} - \frac{\xi}{x_T} \cdot \frac{\eta}{y_T} \\ &= \text{Rel}(x_A) + \text{Rel}(y_A) - \text{Rel}(x_A) \cdot \text{Rel}(y_A).\end{aligned}$$

Usually we have $|\text{Rel}(x_A)| \ll 1$, $|\text{Rel}(y_A)| \ll 1$. Therefore, we can skip the last term $\text{Rel}(x_A) \cdot \text{Rel}(y_A)$, since it is much smaller compared with previous two.

$$\begin{aligned}\text{Rel}(x_A \cdot y_A) &= \text{Rel}(x_A) + \text{Rel}(y_A) - \text{Rel}(x_A) \cdot \text{Rel}(y_A) \\ &\approx \text{Rel}(x_A) + \text{Rel}(y_A).\end{aligned}$$

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 $x_A * y_A$.

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.

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:

$$\text{Rel}(x_A y_A) \approx \text{Rel}(x_A) - \text{Rel}(y_A)$$

provided $|\text{Rel}(y_A)| \ll 1$.

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:

$$\text{Rel}(x_A y_A) \approx \text{Rel}(x_A) - \text{Rel}(y_A)$$

provided $|\text{Rel}(y_A)| \ll 1$.

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, 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:

$$\text{Rel}(x_A y_A) \approx \text{Rel}(x_A) - \text{Rel}(y_A)$$

provided $|\text{Rel}(y_A)| \ll 1$.

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.**

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:

$$\text{Rel}(x_A y_A) \approx \text{Rel}(x_A) - \text{Rel}(y_A)$$

provided $|\text{Rel}(y_A)| \ll 1$.

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 $\tilde{f}(x)$.

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 $\tilde{f}(x)$.

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 $\tilde{f}(x)$.

Now, suppose that we have a number $x_A \approx x_T$.

We want to calculate $f(x_T)$, but instead we evaluate $\tilde{f}(x_A)$.

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 $\tilde{f}(x)$.

Now, suppose that we have a number $x_A \approx x_T$.

We want to calculate $f(x_T)$, but instead we evaluate $\tilde{f}(x_A)$.

What can we say about the error in this latter computed quantity?

Rewrite the error

$$f(x_T) - \tilde{f}(x_A) = [f(x_T) - f(x_A)] + [f(x_A) - \tilde{f}(x_A)].$$

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 $\tilde{f}(x)$.

Now, suppose that we have a number $x_A \approx x_T$.

We want to calculate $f(x_T)$, but instead we evaluate $\tilde{f}(x_A)$.

What can we say about the error in this latter computed quantity?

Rewrite the error

$$f(x_T) - \tilde{f}(x_A) = [f(x_T) - f(x_A)] + [f(x_A) - \tilde{f}(x_A)].$$

The quantity $f(x_A) - \tilde{f}(x_A)$ 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 $\tilde{f}(x)$.

Now, suppose that we have a number $x_A \approx x_T$.

We want to calculate $f(x_T)$, but instead we evaluate $\tilde{f}(x_A)$.

What can we say about the error in this latter computed quantity?

Rewrite the error

$$f(x_T) - \tilde{f}(x_A) = [f(x_T) - f(x_A)] + [f(x_A) - \tilde{f}(x_A)].$$

The quantity $f(x_A) - \tilde{f}(x_A)$ 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 and x_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 and x_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 and x_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.

Example. Define $f(x) = b^x$, where b is a positive real number.
Then, last formula yields

$$b^{x_T} - b^{x_A} \approx (\ln b) b^{x_T} (x_T - x_A).$$

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,

$$\begin{aligned}\text{Rel}(b^{x_A}) &\approx \frac{(\ln b) b^{x_T} (x_T - x_A)}{b^{x_T}} \\ &= \frac{(\ln b)(x_T - x_A)x_T}{x_T} \\ &= x_T \ln b \cdot \text{Rel}(x_A) \\ &= K \cdot \text{Rel}(x_A).\end{aligned}$$

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,

$$\begin{aligned}\text{Rel}(b^{x_A}) &\approx \frac{(\ln b) b^{x_T} (x_T - x_A)}{b^{x_T}} \\ &= \frac{(\ln b)(x_T - x_A)x_T}{x_T} \\ &= x_T \ln b \cdot \text{Rel}(x_A) \\ &= K \cdot \text{Rel}(x_A).\end{aligned}$$

Note that if $K = 10^4$ and $\text{Rel}(x_A) = 10^{-7}$, then $\text{Rel}(b^{x_A}) \approx 10^{-3}$.

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,

$$\begin{aligned}\text{Rel}(b^{x_A}) &\approx \frac{(\ln b) b^{x_T} (x_T - x_A)}{b^{x_T}} \\&= \frac{(\ln b)(x_T - x_A)x_T}{x_T} \\&= x_T \ln b \cdot \text{Rel}(x_A) \\&= K \cdot \text{Rel}(x_A).\end{aligned}$$

Note that if $K = 10^4$ and $\text{Rel}(x_A) = 10^{-7}$, then $\text{Rel}(b^{x_A}) \approx 10^{-3}$.

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,

$$\begin{aligned}\text{Rel}(b^{x_A}) &\approx \frac{(\ln b) b^{x_T} (x_T - x_A)}{b^{x_T}} \\ &= \frac{(\ln b)(x_T - x_A)x_T}{x_T} \\ &= x_T \ln b \cdot \text{Rel}(x_A) \\ &= K \cdot \text{Rel}(x_A).\end{aligned}$$

Note that if $K = 10^4$ and $\text{Rel}(x_A) = 10^{-7}$, then $\text{Rel}(b^{x_A}) \approx 10^{-3}$.

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

$$S = a_1 + a_2 + \dots + a_n, \quad (1)$$

where $\{a_j\}_{j=1}^n$ are floating point numbers.

Summation

Let S be a sum with a relatively large number of terms

$$S = a_1 + a_2 + \dots + a_n, \quad (1)$$

where $\{a_j\}_{j=1}^n$ are floating point numbers. The summation process in (1) consists of $n - 1$ consecutive additions:

$$S = (((\dots(a_1 + a_2) + a_3) + \dots + a_{n-1}) + a_n).$$

Summation

Let S be a sum with a relatively large number of terms

$$S = a_1 + a_2 + \dots + a_n, \quad (1)$$

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

$$S_2 = fl(a_1 + a_2),$$

$$S_3 = fl(S_2 + a_3),$$

$$S_4 = fl(S_3 + a_4),$$

$$\vdots$$

$$S_n = fl(S_{n-1} + a_n).$$

Recall the formula

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

Summation

$$S_2 = (a_1 + a_2)(1 + \varepsilon_2),$$

$$S_3 = (S_2 + a_3)(1 + \varepsilon_3),$$

$$S_4 = (S_3 + a_4)(1 + \varepsilon_4),$$

$$\vdots$$

$$S_n = (S_{n-1} + a_n)(1 + \varepsilon_n).$$

Summation

$$S_2 = (a_1 + a_2)(1 + \varepsilon_2),$$

$$S_3 = (S_2 + a_3)(1 + \varepsilon_3),$$

$$S_4 = (S_3 + a_4)(1 + \varepsilon_4),$$

⋮

$$S_n = (S_{n-1} + a_n)(1 + \varepsilon_n).$$

Then,

$$S_3 = (S_2 + a_3)(1 + \varepsilon_3),$$

$$= ((a_1 + a_2)(1 + \varepsilon_2) + a_3)(1 + \varepsilon_3),$$

$$\approx (a_1 + a_2 + a_3) + a_1(\varepsilon_2 + \varepsilon_3),$$

$$+ a_2(\varepsilon_2 + \varepsilon_3) + a_3\varepsilon_3,$$

$$S_4 \approx (a_1 + a_2 + a_3 + a_4) + a_1(\varepsilon_2 + \varepsilon_3 + \varepsilon_4)$$

$$+ a_2(\varepsilon_2 + \varepsilon_3 + \varepsilon_4) + a_3(\varepsilon_3 + \varepsilon_4) + a_4\varepsilon_4.$$

Summation

And finally we get

$$\begin{aligned} S_n \approx & (a_1 + a_2 + \dots + a_n) + a_1(\varepsilon_2 + \dots + \varepsilon_n) \\ & + a_2(\varepsilon_2 + \dots + \varepsilon_n) + a_3(\varepsilon_3 + \dots + \varepsilon_n) \\ & + a_4(\varepsilon_4 + \dots + \varepsilon_n) + \dots + a_n\varepsilon_n. \end{aligned}$$

Summation

And finally we get

$$\begin{aligned}S_n \approx & (a_1 + a_2 + \dots + a_n) + a_1(\varepsilon_2 + \dots + \varepsilon_n) \\& + a_2(\varepsilon_2 + \dots + \varepsilon_n) + a_3(\varepsilon_3 + \dots + \varepsilon_n) \\& + a_4(\varepsilon_4 + \dots + \varepsilon_n) + \dots + a_n\varepsilon_n.\end{aligned}$$

We are interested in the error $S - S_n$:

$$\begin{aligned}S - S_n \approx & -a_1(\varepsilon_2 + \dots + \varepsilon_n) - a_2(\varepsilon_2 + \dots + \varepsilon_n) - a_3(\varepsilon_3 + \dots + \varepsilon_n) \\& - a_4(\varepsilon_4 + \dots + \varepsilon_n) - \dots - a_n\varepsilon_n.\end{aligned}$$

Summation

And finally we get

$$\begin{aligned}S_n \approx & (a_1 + a_2 + \dots + a_n) + a_1(\varepsilon_2 + \dots + \varepsilon_n) \\& + a_2(\varepsilon_2 + \dots + \varepsilon_n) + a_3(\varepsilon_3 + \dots + \varepsilon_n) \\& + a_4(\varepsilon_4 + \dots + \varepsilon_n) + \dots + a_n\varepsilon_n.\end{aligned}$$

We are interested in the error $S - S_n$:

$$\begin{aligned}S - S_n \approx & -a_1(\varepsilon_2 + \dots + \varepsilon_n) - a_2(\varepsilon_2 + \dots + \varepsilon_n) - a_3(\varepsilon_3 + \dots + \varepsilon_n) \\& - a_4(\varepsilon_4 + \dots + \varepsilon_n) - \dots - a_n\varepsilon_n.\end{aligned}$$

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:

$$|a_1| \leq |a_2| \leq |a_3| \leq \dots \leq |a_n|.$$

Summation

And finally we get

$$\begin{aligned} S_n \approx & (a_1 + a_2 + \dots + a_n) + a_1(\varepsilon_2 + \dots + \varepsilon_n) \\ & + a_2(\varepsilon_2 + \dots + \varepsilon_n) + a_3(\varepsilon_3 + \dots + \varepsilon_n) \\ & + a_4(\varepsilon_4 + \dots + \varepsilon_n) + \dots + a_n\varepsilon_n. \end{aligned}$$

We are interested in the error $S - S_n$:

$$\begin{aligned} S - S_n \approx & -a_1(\varepsilon_2 + \dots + \varepsilon_n) - a_2(\varepsilon_2 + \dots + \varepsilon_n) - a_3(\varepsilon_3 + \dots + \varepsilon_n) \\ & - a_4(\varepsilon_4 + \dots + \varepsilon_n) - \dots - a_n\varepsilon_n. \end{aligned}$$

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:

$$|a_1| \leq |a_2| \leq |a_3| \leq \dots \leq |a_n|.$$

In this rearrangement, smaller numbers a_1 and a_2 will be multiplied with larger numbers $\varepsilon_2 + \dots + \varepsilon_n$, 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:

$$S = \sum_{i=1}^n \frac{1}{i} = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}.$$

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:

$$S = \sum_{i=1}^n \frac{1}{i} = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}.$$

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.

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.

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 than chopping. Just compare the errors in the first and second tables.

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 than 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 $f : [a, b] \rightarrow \mathbb{R}$ 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 $f : [a, b] \rightarrow \mathbb{R}$ 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.

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 $x_1, x_2, x_3, x_4, \dots, x_n, x_{n+1}, \dots$ such that

$$x_n \rightarrow \alpha \text{ as } n \rightarrow \infty.$$

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 $x_1, x_2, x_3, x_4, \dots, x_n, x_{n+1}, \dots$ such that

$$x_n \rightarrow \alpha \text{ as } n \rightarrow \infty.$$

We begin with the simplest of such methods, **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 $x_1, x_2, x_3, x_4, \dots, x_n, x_{n+1}, \dots$ such that

$$x_n \rightarrow \alpha \text{ as } n \rightarrow \infty.$$

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.

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 $x_1, x_2, x_3, x_4, \dots, x_n, x_{n+1}, \dots$ such that

$$x_n \rightarrow \alpha \text{ as } n \rightarrow \infty.$$

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 $\tilde{\alpha} \in [a, b]$ for which

$$|\alpha - \tilde{\alpha}| < \varepsilon.$$

Bisection method is based on the following theorem:

Bisection method is based on the following theorem:

Theorem

If $f : [a, b] \rightarrow \mathbb{R}$ 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 is based on the following theorem:

Theorem

If $f : [a, b] \rightarrow \mathbb{R}$ 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 Algorithm: $\text{Bisect}(f, a, b, \varepsilon)$

Bisection Algorithm: $\text{Bisect}(f, a, b, \varepsilon)$

Step 1: Define

$$c = \frac{a + b}{2}$$

Bisection Algorithm: $\text{Bisect}(f, a, b, \varepsilon)$

Step 1: Define

$$c = \frac{a + b}{2}$$

Step 2: If $b - c \leq \varepsilon$, accept c as our root, and then stop.

Bisection Algorithm: $\text{Bisect}(f, a, b, \varepsilon)$

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

$$\text{sign}(f(a)) \cdot \text{sign}(f(b)) \leq 0$$

then replace a with c ; Otherwise, replace b with c .

Bisection Algorithm: $\text{Bisect}(f, a, b, \varepsilon)$

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

$$\text{sign}(f(a)) \cdot \text{sign}(f(b)) \leq 0$$

then replace a with c ; Otherwise, replace b with c .

Step 4: Return to Step 1.

Bisection Algorithm: $\text{Bisect}(f, a, b, \varepsilon)$

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

$$\text{sign}(f(a)) \cdot \text{sign}(f(b)) \leq 0$$

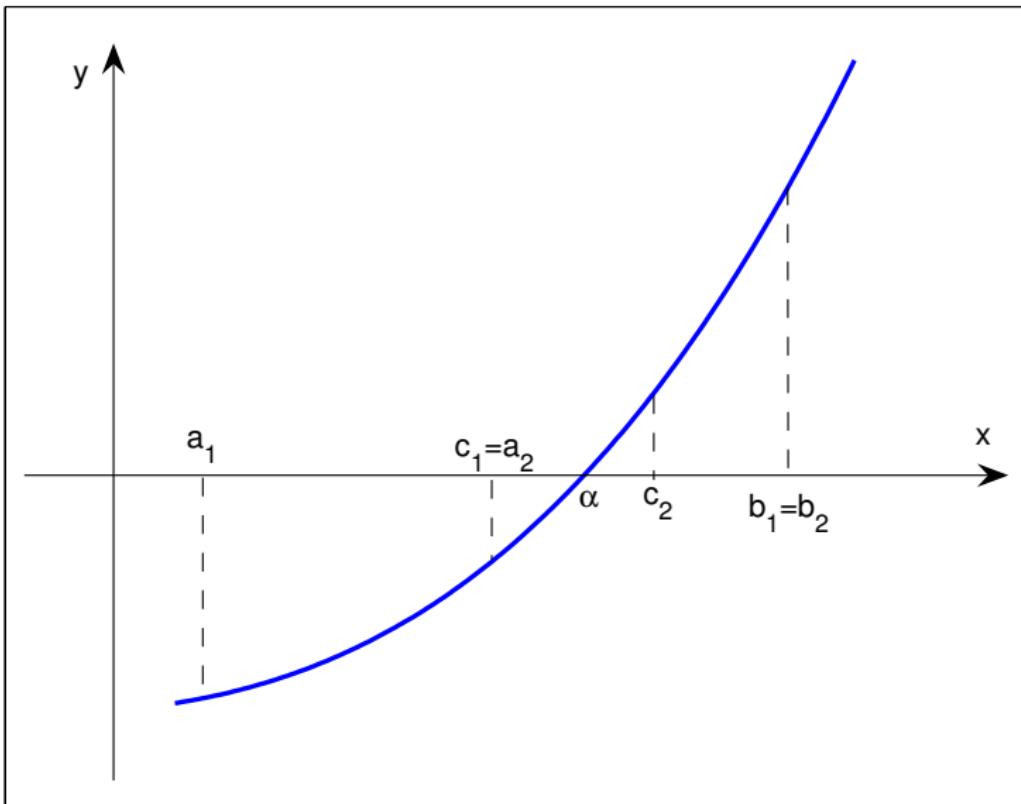
then replace a with c ; Otherwise, replace b with c .

Step 4: Return to Step 1.

Note that, it is preferable to check the sign using condition $\text{sign}(f(a)) \cdot \text{sign}(f(b)) \leq 0$ instead of using $\text{sign}(f(a) \cdot f(b)) \leq 0$.

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

$$|\alpha - c_n| \leq c_n - a_n = b_n - c_n = \frac{1}{2}(b_n - a_n) = \frac{1}{2^n}(b - a).$$

Therefore, if $\alpha \in [a, b]$ is the root, and c_n is the approximation provided by bisection method at step n , we have

$$|\alpha - c_n| \leq \frac{1}{2^n} (b - a).$$

This relation provides us with a stopping criterion for bisection method. Moreover, it follows that $c_n \rightarrow \alpha$ as $n \rightarrow \infty$.

Therefore, if $\alpha \in [a, b]$ is the root, and c_n is the approximation provided by bisection method at step n , we have

$$|\alpha - c_n| \leq \frac{1}{2^n} (b - a).$$

This relation provides us with a stopping criterion for bisection method. Moreover, it follows that $c_n \rightarrow \alpha$ as $n \rightarrow \infty$.

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

$$|\alpha - c_n| \leq \frac{1}{2^n} (b - a).$$

This relation provides us with a stopping criterion for bisection method. Moreover, it follows that $c_n \rightarrow \alpha$ as $n \rightarrow \infty$.

Suppose we want to estimate the number of iterations in bisection method necessary to find the root with an error tolerance ε ,

$$\begin{aligned} |\alpha - c_n| &\leq \varepsilon \\ \frac{1}{2^n} (b - a) &\leq \varepsilon \\ n &\geq \frac{\ln\left(\frac{b-a}{\varepsilon}\right)}{\ln 2}. \end{aligned}$$

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

$$|\alpha - c_n| \leq \frac{1}{2^n} (b - a).$$

This relation provides us with a stopping criterion for bisection method. Moreover, it follows that $c_n \rightarrow \alpha$ as $n \rightarrow \infty$.

Suppose we want to estimate the number of iterations in bisection method necessary to find the root with an error tolerance ε ,

$$\begin{aligned} |\alpha - c_n| &\leq \varepsilon \\ \frac{1}{2^n} (b - a) &\leq \varepsilon \\ n &\geq \frac{\ln\left(\frac{b-a}{\varepsilon}\right)}{\ln 2}. \end{aligned}$$

Thus, using the last inequality for the previous example we get

$$n \geq \frac{\ln\left(\frac{1}{0.001}\right)}{\ln 2} \approx 9.97.$$

Advantages:

- 1** It always converges.

Advantages:

- 1 It always converges.
- 2 You have a guaranteed error bound, and it decreases with each successive iteration.

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:

- 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:

- 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:

- 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:

- 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:

- 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

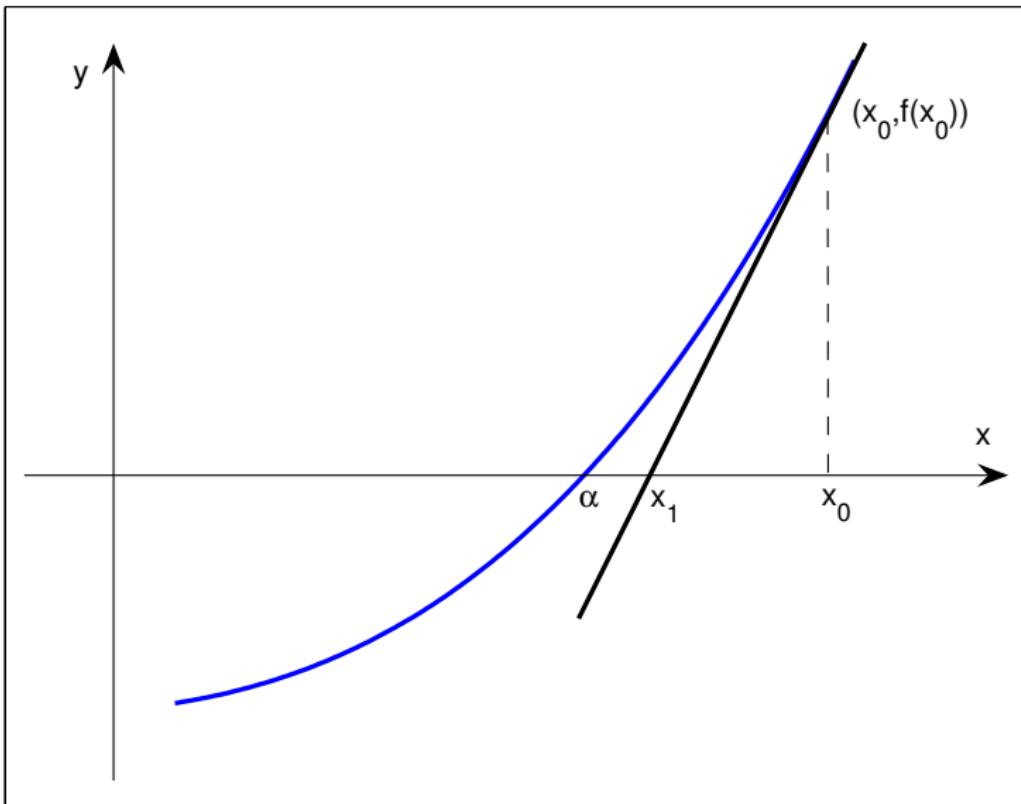
If you cannot solve the given problem, then solve a "nearby problem".

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



Let x_0 be an initial guess, sufficiently closed to the root α .

Let x_0 be an initial guess, sufficiently closed to the root α .
Consider the tangent line to the graph of $f(x)$ in $(x_0, f(x_0))$.

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 ,

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}.$$

Repeat this process to obtain the sequence x_1, x_2, x_3, \dots that hopefully will converge to root α .

Newton's method

Repeat this process to obtain the sequence x_1, x_2, x_3, \dots that hopefully will converge to root α .

General scheme for Newton's method

Starting with initial guess x_0 , compute iterated sequence:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \quad n = 0, 1, 2, \dots$$

Newton's method

Repeat this process to obtain the sequence x_1, x_2, x_3, \dots that hopefully will converge to root α .

General scheme for Newton's method

Starting with initial guess x_0 , compute iterated sequence:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \quad n = 0, 1, 2, \dots$$

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 $\alpha = 1.134724138$.

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.

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

$$x_{n+1} = x_n - \frac{b - \frac{1}{x_n}}{\frac{1}{x_n^2}}.$$

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

$$x_{n+1} = x_n(2 - bx_n), \quad n \geq 0.$$

Initial guess x_0 must be close enough to the true solution and of course $x_0 > 0$. Consider the error

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,

$$\begin{aligned}\text{Rel}(x_{n+1}) &= \frac{\alpha - x_{n+1}}{\alpha} \\ &= 1 - bx_{n+1}.\end{aligned}$$

It can be shown (try it!) that

$$\text{Rel}(x_{n+1}) = (\text{Rel}(x_n))^2.$$

It can be shown (try it!) that

$$\text{Rel}(x_{n+1}) = (\text{Rel}(x_n))^2.$$

In order to guarantee convergence $x_n \rightarrow \alpha$, we ask

$$|\text{Rel}(x_0)| < 1$$

Newton's method. Division example

It can be shown (try it!) that

$$\text{Rel}(x_{n+1}) = (\text{Rel}(x_n))^2.$$

In order to guarantee convergence $x_n \rightarrow \alpha$, we ask

$$|\text{Rel}(x_0)| < 1$$

or equivalently

$$0 < x_0 < \frac{2}{b}.$$

Newton's method. Division example

It can be shown (try it!) that

$$\text{Rel}(x_{n+1}) = (\text{Rel}(x_n))^2.$$

In order to guarantee convergence $x_n \rightarrow \alpha$, we ask

$$|\text{Rel}(x_0)| < 1$$

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.

Newton's method. Division example

It can be shown (try it!) that

$$\text{Rel}(x_{n+1}) = (\text{Rel}(x_n))^2.$$

In order to guarantee convergence $x_n \rightarrow \alpha$, we ask

$$|\text{Rel}(x_0)| < 1$$

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 $|\text{Rel}(x_0)| = 0.1$. Then

Newton's method. Division example

It can be shown (try it!) that

$$\text{Rel}(x_{n+1}) = (\text{Rel}(x_n))^2.$$

In order to guarantee convergence $x_n \rightarrow \alpha$, we ask

$$|\text{Rel}(x_0)| < 1$$

or equivalently

$$0 < x_0 < \frac{2}{b}.$$

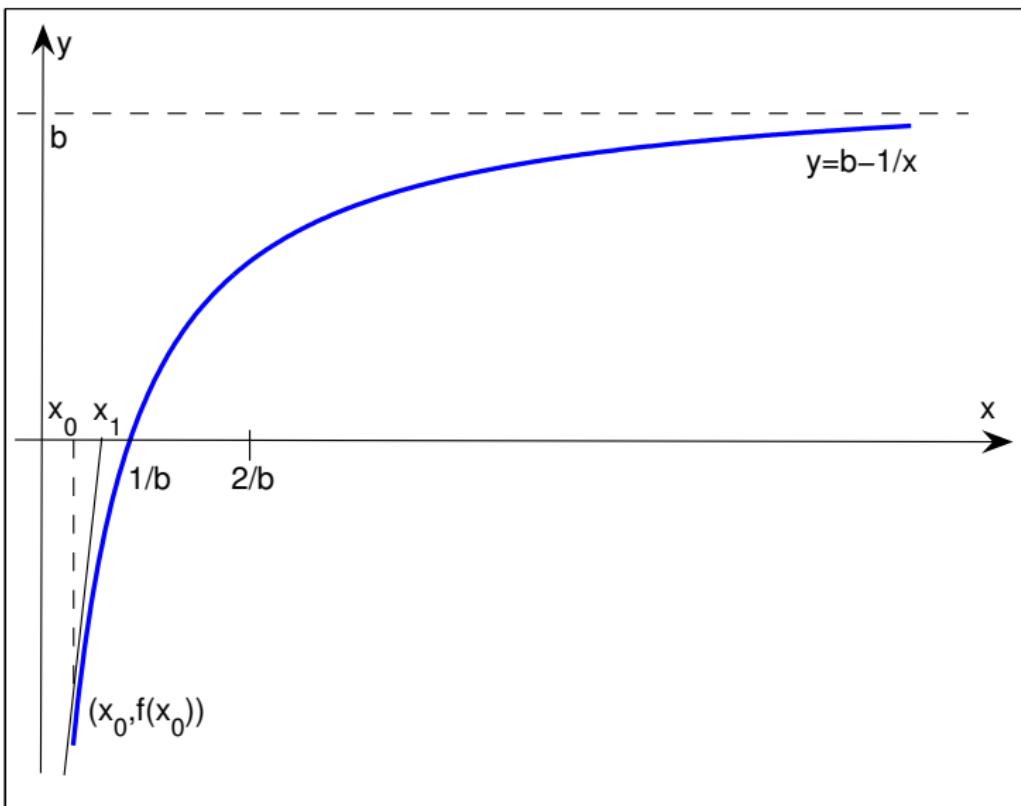
In other words, if above condition is satisfied, then the sequence generated by Newton method for division example will converge.

For example, suppose that $|\text{Rel}(x_0)| = 0.1$. Then

$$\text{Rel}(x_1) = 10^{-2}, \quad \text{Rel}(x_2) = 10^{-4}$$

$$\text{Rel}(x_3) = 10^{-8}, \quad \text{Rel}(x_4) = 10^{-16}.$$

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

$$\frac{-f''(\xi_n)}{2f'(x_n)} \approx \frac{-f''(\alpha)}{2f'(\alpha)} \equiv M.$$

$$\alpha - x_{n+1} \approx M(\alpha - x_n)^2.$$

$$M(\alpha - x_{n+1}) \approx (M(\alpha - x_n))^2.$$

Error analysis for Newton's method



If iteration x_n is close to α , we could denote

$$\frac{-f''(\xi_n)}{2f'(x_n)} \approx \frac{-f''(\alpha)}{2f'(\alpha)} \equiv M.$$

$$\alpha - x_{n+1} \approx M(\alpha - x_n)^2.$$

$$M(\alpha - x_{n+1}) \approx (M(\alpha - x_n))^2.$$

Inductively,

$$M(\alpha - x_{n+1}) \approx (M(\alpha - x_0))^{2^n}, \quad \forall n \in \mathbb{N}.$$

Error analysis for Newton's method

If iteration x_n is close to α , we could denote

$$\frac{-f''(\xi_n)}{2f'(x_n)} \approx \frac{-f''(\alpha)}{2f'(\alpha)} \equiv M.$$

$$\alpha - x_{n+1} \approx M(\alpha - x_n)^2.$$

$$M(\alpha - x_{n+1}) \approx (M(\alpha - x_n))^2.$$

Inductively,

$$M(\alpha - x_{n+1}) \approx (M(\alpha - x_0))^{2^n}, \quad \forall n \in \mathbb{N}.$$

In other words, in order to guarantee the convergence of Newton's method we should have

$$|M(\alpha - x_0)| < 1,$$

$$|\alpha - x_0| < \frac{1}{|M|} = \left| \frac{2f'(\alpha)}{f''(\alpha)} \right|.$$

Advantages of Newton's method:

- 1 Fast convergence (usually quadratic convergence).

Advantages of Newton's method:

- 1 Fast convergence (usually quadratic convergence).
- 2 There is a stopping criterion.

Advantages of Newton's method:

- 1 Fast convergence (usually quadratic convergence).
- 2 There is a stopping criterion.

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).

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.

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.

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

$$q(x) = a_0 + a_1 x, \text{ such that}$$

$$q(x_0) = f(x_0), \quad q(x_1) = f(x_1).$$

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

$$q(x) = a_0 + a_1 x, \text{ such that}$$

$$q(x_0) = f(x_0), \quad q(x_1) = f(x_1).$$

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 :

$$x_2 = x_1 - f(x_1) : \frac{f(x_1) - f(x_0)}{x_1 - x_0}.$$

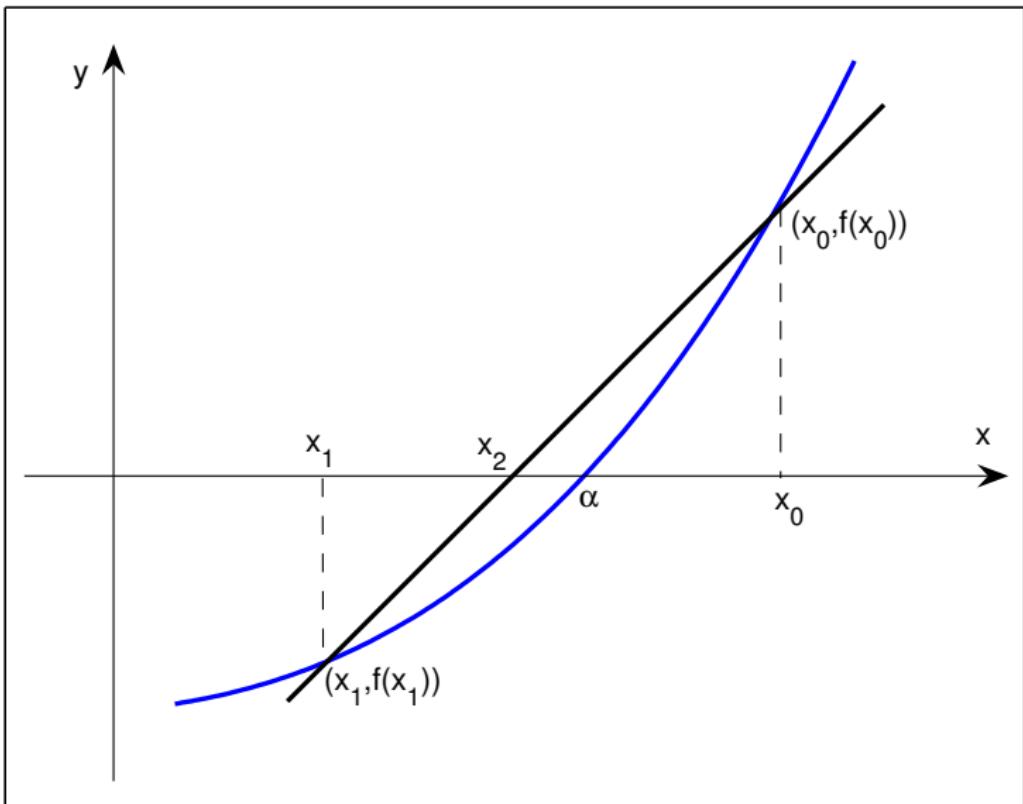
We can now repeat the whole process: use x_1 and x_2 to produce another secant line, and then use its root to approximate α .

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 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:

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

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 $\{x_{n-1}, x_n, \alpha\}$.

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 $x_n \rightarrow \alpha$ as $n \rightarrow \infty$ 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 $x_n \rightarrow \alpha$ as $n \rightarrow \infty$ 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

$$\lim_{n \rightarrow \infty} \frac{|\alpha - x_{n+1}|}{|\alpha - x_n|^r} = \left| \frac{f''(\alpha)}{2f'(\alpha)} \right|^{r-1}$$

where $r = \frac{1+\sqrt{5}}{2} \approx 1.62$.

The last formula says that when we are close to α , that

$$|\alpha - x_{n+1}| \approx C |\alpha - x_n|^r \text{ with } C = \left| \frac{f''(\alpha)}{2f'(\alpha)} \right|^{r-1} \text{ and } r \approx 1.62.$$

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:

$$|\alpha - x_{n+1}| \approx M|\alpha - x_n|^2 \text{ with } M = \left| \frac{f''(\alpha)}{2f'(\alpha)} \right|$$

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 $r = 2$).

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

$$\lim_{n \rightarrow \infty} \frac{|x_{n+1} - x_n|}{|\alpha - x_n|} = 1$$

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

$$\lim_{n \rightarrow \infty} \frac{|x_{n+1} - x_n|}{|\alpha - x_n|} = 1$$

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

$$\lim_{n \rightarrow \infty} \frac{|x_{n+1} - x_n|}{|\alpha - x_n|} = 1$$

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:

$$x_{n+1} = x_n - \frac{f(x_n)(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})}.$$

Observe that the Newton's method

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

requires two function evaluations per iteration.

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.

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.

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

$$\frac{T_S}{T_N} = \frac{t_f}{t_f + st_f} \cdot \frac{\ln 2}{\ln r}, \quad r \approx 1.62,$$

$$\frac{T_S}{T_N} = \frac{1}{1+s} \cdot \frac{\ln 2}{\ln r}.$$

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:

- 1 Secant method converges at faster than a linear rate, so that it is more rapidly convergent than the bisection 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:

- 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:

- 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:

- 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:

- 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:

- 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:

- 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.

Richard Brent devised a method combining the advantages of the bisection method and the secant method.

- 1 It is guaranteed to converge.

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.

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.

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.

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.

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

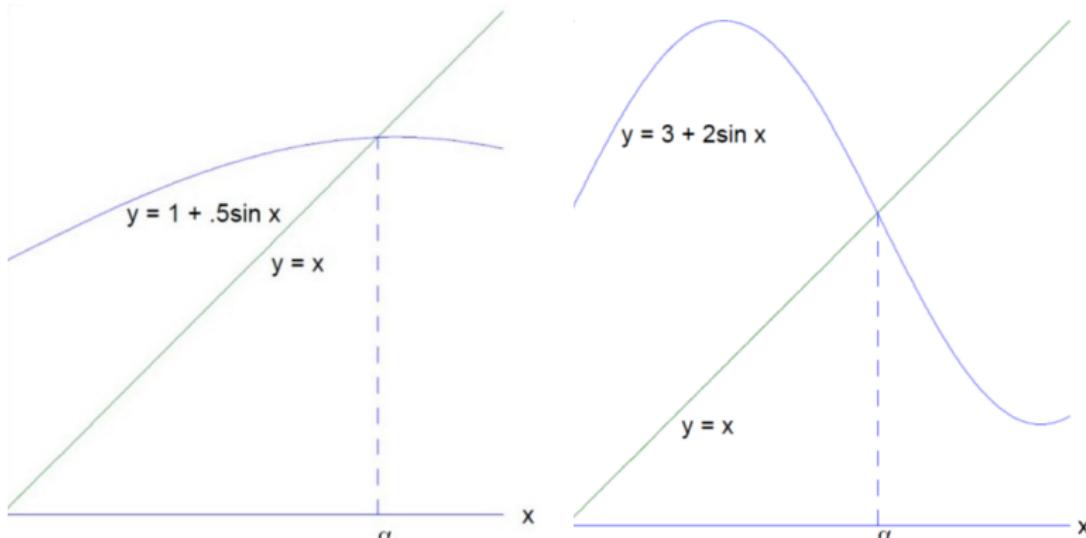
Fixed point iteration

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.

$$x_1 = 1 + 0.5 \sin x_0,$$

$$x_2 = 1 + 0.5 \sin x_1,$$

...

$$x_n = 1 + 0.5 \sin x_{n-1},$$

...

This is repeated until convergence occurs or until the iteration is terminated. The above iterations can be written symbolically as

$$E1 : \quad x_{n+1} = 1 + 0.5 \sin x_n, \quad n = 0, 1, 2, \dots$$

$$E2 : \quad x_{n+1} = 3 + 2 \sin x_n, \quad n = 0, 1, 2, \dots$$

Fixed point iterations

	$E1$	$E2$
n	x_n	x_n
0	0.000000000000000	3.000000000000000
1	1.000000000000000	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 $E2$.

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.

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:

$$I_1: \quad x_{n+1} = 5 + x_n - x_n^2;$$

$$I_2: \quad x_{n+1} = \frac{5}{x_n};$$

$$I_3: \quad x_{n+1} = 1 + x_n - \frac{1}{5}x_n^2;$$

$$I_4: \quad x_{n+1} = \frac{1}{2} \left(x_n + \frac{5}{x_n} \right).$$

It can be shown for all 4 sequences that if $\lim_{n \rightarrow \infty} = \alpha$ exist, then $\alpha = \sqrt{5}$.

It can be shown for all 4 sequences that if $\lim_{n \rightarrow \infty} = \alpha$ 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:

$$x_{n+1} = g(x_n) \quad \text{with } n = 0, 1, 2, 3, \dots$$

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:

$$x_{n+1} = g(x_n) \quad \text{with } n = 0, 1, 2, 3, \dots$$

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.

$$g(\alpha) = \alpha.$$

Begin by asking whether there is a solution to equation

$$x = g(x).$$

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. $\alpha = g(\alpha)$.

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

If $a \leqslant x \leqslant b$ then $a \leqslant g(x) \leqslant b$,

i.e. $g([a, b]) \subset [a, b]$.

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

$$\lambda \stackrel{\text{def}}{=} \max_{a \leq x \leq b} |g'(x)| < 1.$$

Then:

(S1) $\exists! \alpha \in [a, b]$ s.t. $\alpha = g(\alpha)$.

(S2) $\forall x_0 \in [a, b]$, the iteration $x_{n+1} = g(x_n) \rightarrow \alpha$.

$$(S3) \quad |\alpha - x_n| \leq \frac{\lambda^n}{1-\lambda} |x_1 - x_0|, \quad n \geq 1.$$

$$(S4) \quad \lim_{n \rightarrow \infty} \frac{|\alpha - x_{n+1}|}{|\alpha - x_n|} = g'(\alpha).$$

Fixed point iterations

The statement

$$(S4) \quad \lim_{n \rightarrow \infty} \frac{|\alpha - x_{n+1}|}{|\alpha - x_n|} = g'(\alpha)$$

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

$$(S4) \quad \lim_{n \rightarrow \infty} \frac{|\alpha - x_{n+1}|}{|\alpha - x_n|} = g'(\alpha)$$

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

$$(S4) \quad \lim_{n \rightarrow \infty} \frac{|\alpha - x_{n+1}|}{|\alpha - x_n|} = g'(\alpha)$$

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.

The statement

$$(S4) \quad \lim_{n \rightarrow \infty} \frac{|\alpha - x_{n+1}|}{|\alpha - x_n|} = g'(\alpha)$$

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.

The statement

$$(S4) \quad \lim_{n \rightarrow \infty} \frac{|\alpha - x_{n+1}|}{|\alpha - x_n|} = g'(\alpha)$$

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.

Consider first fixed point example:

$$E1 : \quad x = 1 + 0.5 \sin x,$$

$$E2 : \quad x = 3 + 2 \sin x.$$

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

$$g'(x) = 0.5 \cos x,$$

$$g'(\alpha) \leq 0.5$$

and therefore $x_{n+1} = 1 + 0.5 \sin x_n$ will converge to α .

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

$$g'(x) = 0.5 \cos x,$$

$$g'(\alpha) \leq 0.5$$

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.

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 (I₁) : $g(x) = 5 + x - x^2$.

$$g'(x) = 1 - 2x,$$

$$g'(\sqrt{5}) = 1 - 2\sqrt{5} < -1.$$

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 (I₁) : $g(x) = 5 + x - x^2$.

$$g'(x) = 1 - 2x,$$

$$g'(\sqrt{5}) = 1 - 2\sqrt{5} < -1.$$

Thus, $x_n = g(x_{n-1})$ will not converge to $\sqrt{5}$.

Case (I₂) : $g(x) = \frac{5}{x}$.

$$g'(x) = -\frac{5}{x^2},$$

$$g'(\sqrt{5}) = -1.$$

So, $x_n = g(x_{n-1})$ may converge or diverge, but results show that it doesn't converge to $\sqrt{5}$.

Case (I₃) : $g(x) = 1 + x - \frac{1}{5}x^2$.

$$g'(x) = 1 - \frac{2}{5}x,$$

$$g'(\sqrt{5}) = 1 - \frac{2}{5}\sqrt{5} \approx 0.106.$$

Therefore, $x_n = g(x_{n-1}) \rightarrow \sqrt{5}$. 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 (I₄) : $g(x) = \frac{1}{2} \left(x + \frac{5}{x} \right)$.

$$g'(x) = \frac{1}{2} \left(1 - \frac{5}{x^2} \right),$$

$$g'(\sqrt{5}) = 0.$$

Thus, $x_n = g(x_{n-1}) \rightarrow \sqrt{5}$ is convergent with an order of convergence higher than 1.

Fixed point iterations

Case (I₄) : $g(x) = \frac{1}{2} \left(x + \frac{5}{x} \right).$

$$g'(x) = \frac{1}{2} \left(1 - \frac{5}{x^2} \right),$$

$$g'(\sqrt{5}) = 0.$$

Thus, $x_n = g(x_{n-1}) \rightarrow \sqrt{5}$ 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 $x^2 - 5 = 0$.

Sometimes it is difficult to express initial equation $f(x) = 0$ in the form $x = g(x)$ such that the resulting iterates will converge.

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

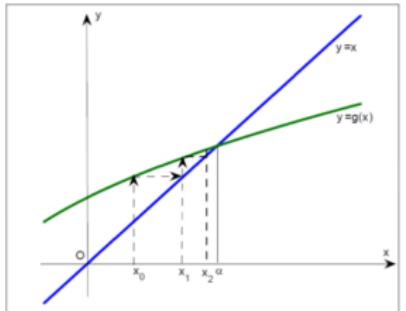
$$x = \frac{1}{1+x^2}.$$

That will provide us with iterations

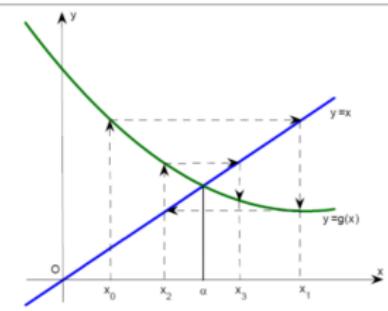
$$x_0 = 1, \quad x_{n+1} = \frac{1}{1+x_n^2},$$

which will converge to $\alpha \approx 0.6823$.

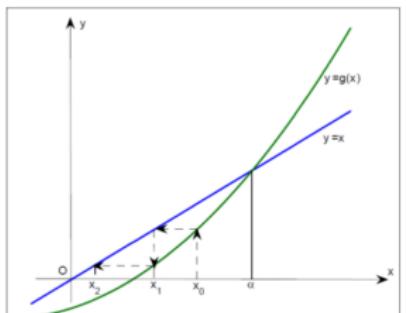
Fixed point iterations



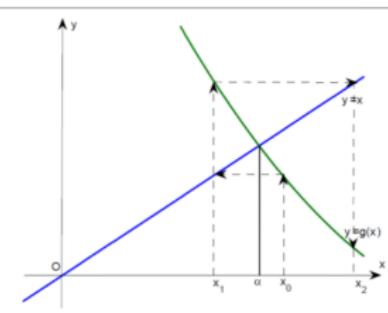
$$0 < g'(\alpha) < 1$$



$$-1 < g'(\alpha) < 0$$



$$g'(\alpha) > 1$$



$$g'(\alpha) < -1$$

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 $g'(\alpha)$.

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 $g'(\alpha)$:

$$g'(x) = -\frac{f(x)f''(x)}{(f'(x))^2}, \quad g'(\alpha) = 0.$$

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:

$$\lim_{n \rightarrow \infty} \frac{\alpha - x_n}{\alpha - x_{n-1}} = g'(\alpha) = \lambda.$$

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:

$$\lim_{n \rightarrow \infty} \frac{\alpha - x_n}{\alpha - x_{n-1}} = g'(\alpha) = \lambda.$$

Thus,

$$\alpha - x_n \approx \lambda(\alpha - x_{n-1})$$

with $|\lambda| < 1$.

If λ is known, then solve for α to get

$$\begin{aligned}\alpha &\approx \frac{x_n - \lambda x_{n-1}}{1 - \lambda} \\&= \frac{x_n - \lambda x_n}{1 - \lambda} + \frac{\lambda x_n - \lambda x_{n-1}}{1 - \lambda} \\&= x_n + \frac{\lambda}{1 - \lambda} [x_n - x_{n-1}].\end{aligned}$$

Thus, if λ is known, we have on the right side the so-called **extrapolation** of x_{n-1} and x_n :

$$\alpha \approx x_n + \frac{\lambda}{1-\lambda} [x_n - x_{n-1}].$$

Thus, if λ is known, we have on the right side the so-called **extrapolation** of x_{n-1} and x_n :

$$\alpha \approx x_n + \frac{\lambda}{1-\lambda} [x_n - x_{n-1}].$$

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 $x_n \rightarrow \alpha$, we have that $\lambda_n \rightarrow \lambda$ as $n \rightarrow \infty$.

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**:

$$\hat{x}_n = x_n + \frac{\lambda_n}{1 - \lambda_n} [x_n - x_{n-1}],$$

where

$$\lambda_n = \frac{x_n - x_{n-1}}{x_{n-1} - x_{n-2}}.$$

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**:

$$\hat{x}_n = x_n + \frac{\lambda_n}{1 - \lambda_n} [x_n - x_{n-1}],$$

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,

$$\alpha - x_n \approx \hat{x}_n - x_n = \frac{\lambda_n}{1 - \lambda_n} [x_n - x_{n-1}],$$

which is known as **Aitken's error estimation formula** and can be used to estimate the error in fixed point iterations.

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 $(\alpha - x_n) \rightarrow 0$, since $\lambda_n \rightarrow \lambda = g'(\alpha) \approx 0.9644$ 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 $(\alpha - x_n) \rightarrow 0$, since $\lambda_n \rightarrow \lambda = g'(\alpha) \approx 0.9644$ 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.

$$\alpha - x_n \approx \hat{x}_n - x_n = \frac{\lambda_n}{1 - \lambda_n} [x_n - x_{n-1}].$$

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 .

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 :

$$\lambda_2 = \frac{x_2 - x_1}{x_1 - x_0},$$

$$x_3 = x_2 + \frac{\lambda_2}{1 + \lambda_2} [x_2 - x_1].$$

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 :

$$\lambda_2 = \frac{x_2 - x_1}{x_1 - x_0},$$

$$x_3 = x_2 + \frac{\lambda_2}{1 + \lambda_2} [x_2 - x_1].$$

- 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 :

$$\lambda_2 = \frac{x_2 - x_1}{x_1 - x_0},$$

$$x_3 = x_2 + \frac{\lambda_2}{1 + \lambda_2} [x_2 - x_1].$$

- 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 :

$$\lambda_2 = \frac{x_2 - x_1}{x_1 - x_0},$$
$$x_3 = x_2 + \frac{\lambda_2}{1 + \lambda_2} [x_2 - x_1].$$

- 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

$$\alpha - x_3 = 7.98e - 04, \quad \alpha - x_6 = 2.27e - 06.$$

On the other hand, using original fixed point iterations we got

$$\alpha - x_3 = 1.38e - 02, \quad \alpha - x_6 = 1.23e - 02.$$

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

$$\alpha - x_3 = 7.98e - 04, \quad \alpha - x_6 = 2.27e - 06.$$

On the other hand, using original fixed point iterations we got

$$\alpha - x_3 = 1.38e - 02, \quad \alpha - x_6 = 1.23e - 02.$$

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,

$$\lim_{x \rightarrow 0} h(x) = \lim_{x \rightarrow 0} \frac{e^{x^2} - 1}{x^2} = 1.$$

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,

$$\lim_{x \rightarrow 0} h(x) = \lim_{x \rightarrow 0} \frac{e^{x^2} - 1}{x^2} = 1.$$

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,

$$\begin{aligned} f'(x) &= 3(x - \alpha)^2 h(x) + (x - \alpha)^3 h'(x) \\ &= (x - \alpha)^2 [3h(x) + (x - \alpha)h'(x)] = (x - \alpha)^2 q(x) \end{aligned}$$

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,

$$\begin{aligned} f'(x) &= 3(x - \alpha)^2 h(x) + (x - \alpha)^3 h'(x) \\ &= (x - \alpha)^2 [3h(x) + (x - \alpha)h'(x)] = (x - \alpha)^2 q(x) \end{aligned}$$

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,

$$\begin{aligned} f'(x) &= 3(x - \alpha)^2 h(x) + (x - \alpha)^3 h'(x) \\ &= (x - \alpha)^2 [3h(x) + (x - \alpha)h'(x)] = (x - \alpha)^2 q(x) \end{aligned}$$

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

$$f(\alpha) = 0,$$

$$f'(\alpha) = 0,$$

$$f''(\alpha) = 0,$$

$$f'''(\alpha) \neq 0.$$

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

$$\begin{aligned}f(\alpha) &= 0, \\f'(\alpha) &= 0, \\f''(\alpha) &= 0, \\f'''(\alpha) &\neq 0.\end{aligned}$$

We can prove the following generalizing theorem:

Theorem

Number α is a root of $f(x)$ of multiplicity m if and only if

$$\begin{aligned}f(\alpha) = f'(\alpha) = f''(\alpha) = \dots = f^{(m-1)}(\alpha) &= 0, \\f^{(m)}(\alpha) &\neq 0.\end{aligned}$$

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.

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.

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.

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

$$\begin{aligned} g(x) &= x - \frac{(x - \alpha)^m h(x)}{m(x - \alpha)^{m-1} h(x) + (x - \alpha)^m h'(x)} \\ &= x - \frac{(x - \alpha) h(x)}{m h(x) + (x - \alpha) h'(x)}. \end{aligned}$$

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.

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

$$x_n = g(x_{n-1}) \quad \text{with} \quad g(x) = x - m \frac{f(x)}{f'(x)}$$

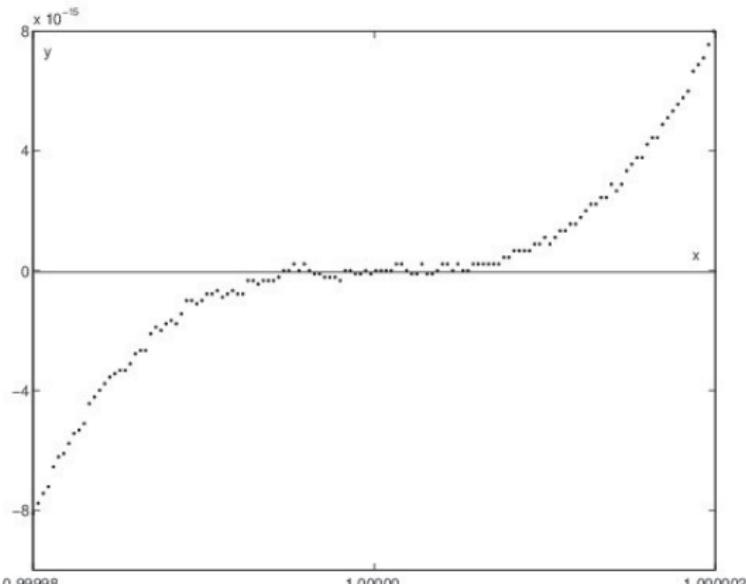
for which $g'(\alpha) = 0$ and the iteration method will converge quadratically.

Both possibilities presume that multiplicity m is known in advance.

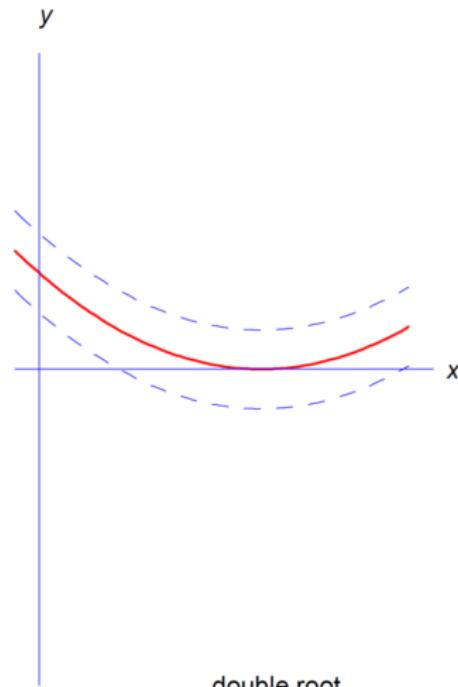
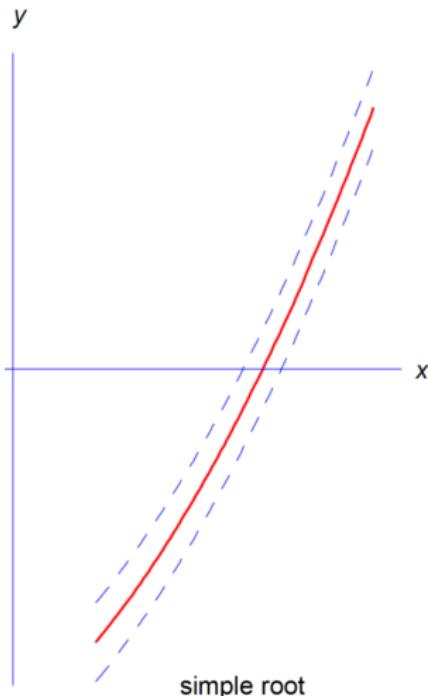
Recall that λ_n (that are computable) converge to $\lambda = \frac{m-1}{m}$.

Therefore, m can be guessed from numerical computations.

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:

$$\begin{aligned}F(x) &= x^7 - \textcolor{red}{28.002}x^6 + 322x^5 - 1960x^4 + 6769x^3 - 13132x^2 \\&\quad + 13068x - 5040.\end{aligned}$$

As it can be seen, this is a relatively small change in one coefficient with a relative error of

$$\text{Rel}(28) = \frac{28 - 28.002}{28} = -\frac{0.002}{28} \approx 7.14e-05.$$

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:

$$\begin{aligned}F(x) &= x^7 - \textcolor{red}{28.002}x^6 + 322x^5 - 1960x^4 + 6769x^3 - 13132x^2 \\&\quad + 13068x - 5040.\end{aligned}$$

As it can be seen, this is a relatively small change in one coefficient with a relative error of

$$\text{Rel}(28) = \frac{28 - 28.002}{28} = -\frac{0.002}{28} \approx 7.14e-05.$$

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

$$x_0 = 0, \quad x_1 = \frac{\pi}{4}, \quad x_2 = \frac{\pi}{2}$$

and

$$y_i = \cos x_i, \quad i = 0, 1, 2$$

This gives us the three points

$$(0, 1), \quad \left(\frac{\pi}{4}, \frac{1}{\sqrt{2}}\right), \quad \left(\frac{\pi}{2}, 0\right)$$

Now find a quadratic polynomial

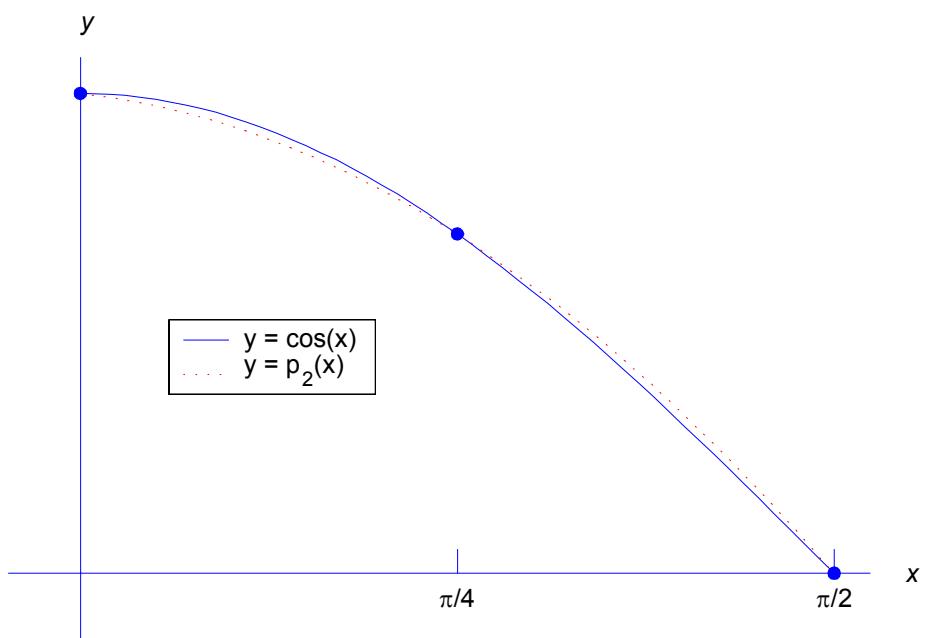
$$p(x) = a_0 + a_1 x + a_2 x^2$$

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 $\cos(x)$



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 $\{a_j\}$ 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 an 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

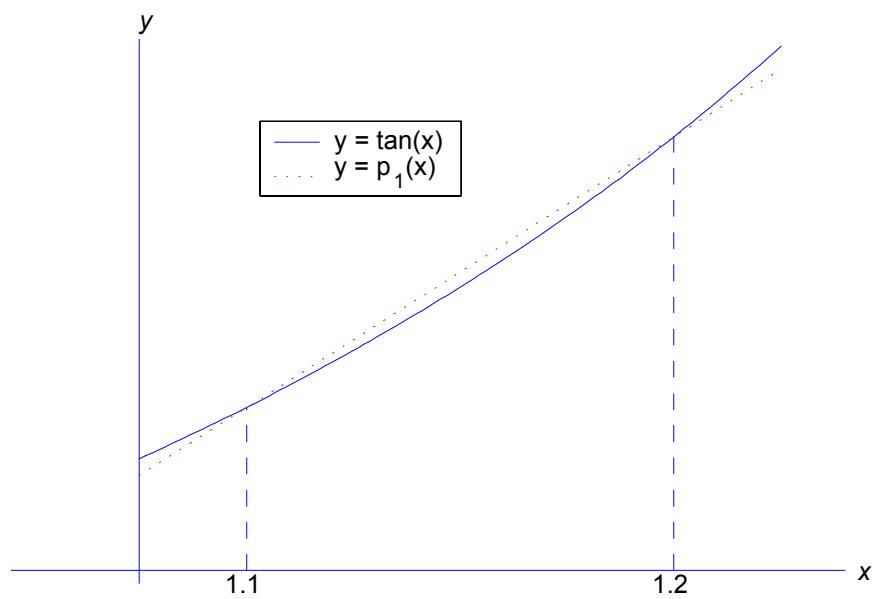
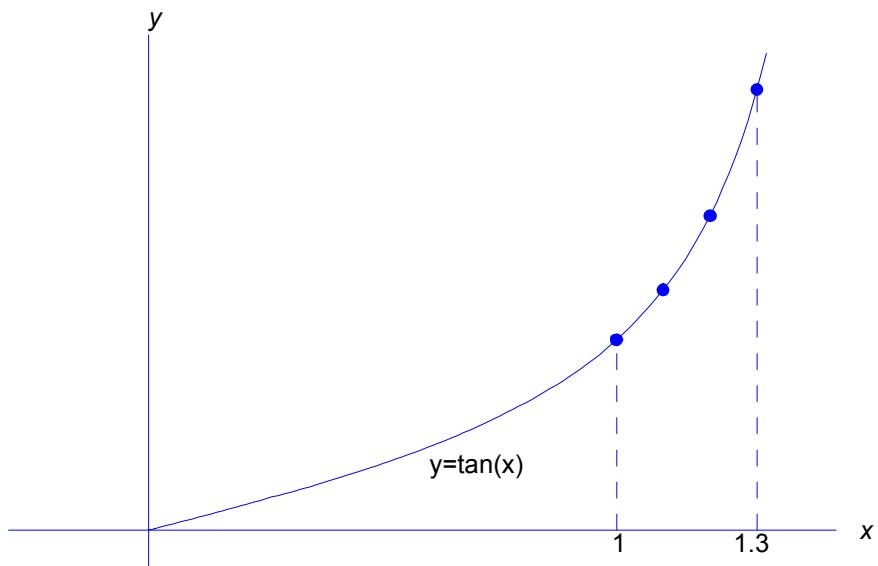
$$x_0 = 1.1, \quad x_1 = 1.2$$

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.



QUADRATIC INTERPOLATION

We want to find a polynomial

$$P_2(x) = a_0 + a_1x + a_2x^2$$

which satisfies

$$P_2(x_i) = y_i, \quad i = 0, 1, 2$$

for given data points $(x_0, y_0), (x_1, y_1), (x_2, y_2)$. One formula for such a polynomial follows:

$$P_2(x) = y_0L_0(x) + y_1L_1(x) + y_2L_2(x) \quad (**)$$

with

$$\begin{aligned} L_0(x) &= \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}, & L_1(x) &= \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} \\ L_2(x) &= \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)} \end{aligned}$$

The formula $(**)$ is called *Lagrange's form* of the interpolation polynomial.

LAGRANGE BASIS FUNCTIONS

The functions

$$L_0(x) = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}, \quad L_1(x) = \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}$$
$$L_2(x) = \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}$$

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

$$\begin{aligned}\deg(Q) &\leq 2 \\ Q(x_i) &= y_i, \quad i = 0, 1, 2\end{aligned}$$

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

$$\begin{aligned} \deg(P_n) &\leq n \\ P_n(x_i) &= y_i, \quad i = 0, 1, \dots, n \end{aligned} \tag{**}$$

with given data points

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

The solution is given by Lagrange's formula

$$P_n(x) = y_0 L_0(x) + y_1 L_1(x) + \dots + y_n L_n(x)$$

The Lagrange basis functions are given by

$$L_k(x) = \frac{(x - x_0) \dots (x - x_{k-1})(x - x_{k+1}) \dots (x - x_n)}{(x_k - x_0) \dots (x_k - x_{k-1})(x_k - x_{k+1}) \dots (x_k - x_n)}$$

for $k = 0, 1, 2, \dots, n$. 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

$$L_k(x) = \frac{(x - x_0) \dots (x - x_{k-1})(x - x_{k+1}) \dots (x - x_n)}{(x_k - x_0) \dots (x_k - x_{k-1})(x_k - x_{k+1}) \dots (x_k - x_n)}$$

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

$$P_n(x) = y_0 L_0(x) + y_1 L_1(x) + \dots + y_n L_n(x)$$

satisfies the interpolation problem of finding a solution to

$$\begin{aligned} \deg(P_n) &\leq n \\ P_n(x_i) &= y_i, \quad i = 0, 1, \dots, n \end{aligned}$$

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 is 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

$$f[x_0, x_1] = \frac{.45360 - .54030}{1.1 - 1} = -.86700$$

$$f[x_1, x_2] = \frac{.36236 - .45360}{1.1 - 1} = -.91240$$

$$\begin{aligned} f[x_0, x_1, x_2] &= \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} \\ &= \frac{-.91240 - (-.86700)}{1.2 - 1.0} = -.22700 \end{aligned}$$

For comparison,

$$f' \left(\frac{x_1 + x_0}{2} \right) = -\sin(1.05) = -.86742$$

$$\frac{1}{2} f''(x_1) = -\frac{1}{2} \cos(1.1) = -.22680$$

GENERAL DIVIDED DIFFERENCES

Given $n + 1$ distinct points x_0, \dots, x_n , with $n \geq 2$, define

$$f[x_0, \dots, x_n] = \frac{f[x_1, \dots, x_n] - f[x_0, \dots, x_{n-1}]}{x_n - x_0}$$

This is a recursive definition of the n^{th} -order divided difference of $f(x)$, using divided differences of order n . Its relation to the derivative is as follows:

$$f[x_0, \dots, x_n] = \frac{1}{n!} f^{(n)}(c)$$

for some c intermediate to the points $\{x_0, \dots, x_n\}$. Let I denote the interval

$$I = [\min \{x_0, \dots, x_n\}, \max \{x_0, \dots, x_n\}]$$

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

$$D^k f(x_i) = f[x_i, \dots, x_{i+k}]$$

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				

These were computed using the recursive definition

$$f[x_0, \dots, x_n] = \frac{f[x_1, \dots, x_n] - f[x_0, \dots, x_{n-1}]}{x_n - x_0}$$

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 $f[x_0, \dots, x_n]$ is independent of the order of the arguments $\{x_0, \dots, x_n\}$, even though the intermediate steps in its calculations using

$$f[x_0, \dots, x_n] = \frac{f[x_1, \dots, x_n] - f[x_0, \dots, x_{n-1}]}{x_n - x_0}$$

are order dependent.

We can show

$$f[x_0, \dots, x_n] = f[x_{i_0}, \dots, x_{i_n}]$$

for any permutation (i_0, i_1, \dots, i_n) of $(0, 1, \dots, n)$.

COINCIDENT NODES

What happens when some of the nodes $\{x_0, \dots, x_n\}$ 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

$$\lim_{x_1 \rightarrow x_0} f[x_0, x_1] = \lim_{x_1 \rightarrow x_0} \frac{f(x_1) - f(x_0)}{x_1 - x_0} = f'(x_0)$$

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 $x_1 \rightarrow x_0$ and $x_2 \rightarrow x_0$, we must also have that $c \rightarrow x_0$. Therefore,

$$\lim_{\substack{x_1 \rightarrow x_0 \\ x_2 \rightarrow x_0}} f[x_0, x_1, x_2] = \frac{1}{2} f''(x_0)$$

We therefore define

$$f[x_0, x_0, x_0] = \frac{1}{2} f''(x_0)$$

For the case of general $f[x_0, \dots, x_n]$, recall that

$$f[x_0, \dots, x_n] = \frac{1}{n!} f^{(n)}(c)$$

for some c intermediate to $\{x_0, \dots, x_n\}$. Then

$$\lim_{\{x_1, \dots, x_n\} \rightarrow x_0} f[x_0, \dots, x_n] = \frac{1}{n!} f^{(n)}(x_0)$$

and we define

$$f[\underbrace{x_0, \dots, x_0}_{n+1 \text{ times}}] = \frac{1}{n!} f^{(n)}(x_0)$$

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

$$\begin{aligned}\deg(P_n) &\leq n \\ P_n(x_i) &= y_i, \quad i = 0, 1, \dots, n\end{aligned}$$

with given data points

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

and with $\{x_0, \dots, x_n\}$ distinct points.

In §5.1, we gave the solution as *Lagrange's formula*

$$P_n(x) = y_0L_0(x) + y_1L_1(x) + \dots + y_nL_n(x)$$

with $\{L_0(x), \dots, L_n(x)\}$ the Lagrange basis polynomials. Each L_j is of degree n and it satisfies

$$L_j(x_i) = \begin{cases} 1, & j = i \\ 0, & j \neq i \end{cases}$$

for $i = 0, 1, \dots, n$.

THE NEWTON DIVIDED DIFFERENCE FORM OF THE INTERPOLATION POLYNOMIAL

Let the data values for the problem

$$\deg(P_n) \leq n \\ P_n(x_i) = y_i, \quad i = 0, 1, \dots, n$$

be generated from a function $f(x)$:

$$y_i = f(x_i), \quad i = 0, 1, \dots, n$$

Using the divided differences

$$f[x_0, x_1], \quad f[x_0, x_1, x_2], \dots, f[x_0, \dots, x_n]$$

we can write the interpolation polynomials

$$P_1(x), \quad P_2(x), \dots, P_n(x)$$

in a way that is simple to compute.

$$\begin{aligned} P_1(x) &= f(x_0) + f[x_0, x_1](x - x_0) \\ P_2(x) &= f(x_0) + f[x_0, x_1](x - x_0) \\ &\quad + f[x_0, x_1, x_2](x - x_0)(x - x_1) \\ &= P_1(x) + f[x_0, x_1, x_2](x - x_0)(x - x_1) \end{aligned}$$

For the case of the general problem

$$\deg(P_n) \leq n$$
$$P_n(x_i) = y_i, \quad i = 0, 1, \dots, n$$

we have

$$\begin{aligned} P_n(x) &= f(x_0) + f[x_0, x_1](x - x_0) \\ &\quad + f[x_0, x_1, x_2](x - x_0)(x - x_1) \\ &\quad + f[x_0, x_1, x_2, x_3](x - x_0)(x - x_1)(x - x_2) \\ &\quad + \dots \\ &\quad + f[x_0, \dots, x_n](x - x_0) \cdots (x - x_{n-1}) \end{aligned}$$

From this we have the recursion relation

$$P_n(x) = P_{n-1}(x) + f[x_0, \dots, x_n](x - x_0) \cdots (x - x_{n-1})$$

in which $P_{n-1}(x)$ interpolates $f(x)$ at the points in $\{x_0, \dots, x_{n-1}\}$.

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 $D^k f(x_i) = f[x_i, \dots, x_{i+k}]$, $k = 1, 2, 3, 4$. 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

$$\begin{aligned}d_1 &= f[x_0, x_1] \\d_2 &= f[x_0, x_1, x_2] \\&\vdots \\d_n &= f[x_0, \dots, x_n]\end{aligned}$$

Then the formula

$$\begin{aligned}P_n(x) &= f(x_0) + f[x_0, x_1](x - x_0) \\&\quad + f[x_0, x_1, x_2](x - x_0)(x - x_1) \\&\quad + f[x_0, x_1, x_2, x_3](x - x_0)(x - x_1)(x - x_2) \\&\quad + \dots \\&\quad + f[x_0, \dots, x_n](x - x_0) \cdots (x - x_{n-1})\end{aligned}$$

can be written as

$$\begin{aligned}P_n(x) &= f(x_0) + (x - x_0)(d_1 + (x - x_1)(d_2 + \dots \\&\quad + (x - x_{n-2})(d_{n-1} + (x - x_{n-1})d_n) \dots)\end{aligned}$$

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 $a \leq x \leq b$,

$$f(x) - P_1(x) = \frac{(x - x_0)(x - x_1)}{2} f''(c_x)$$

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 $1 \leq x, x_0, x_1 \leq 10$. 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 $x_0 \leq x \leq x_1$; and in that case, we have

$$(x - x_0)(x_1 - x) \geq 0, \quad x_0 \leq c_x \leq x_1$$

$$(x - x_0)(x_1 - x) \geq 0, \quad x_0 \leq c_x \leq x_1$$

and therefore

$$\begin{aligned} (x - x_0)(x_1 - x) \left[\frac{\log_{10} e}{2x_1^2} \right] &\leq \log_{10} x - P_1(x) \\ &\leq (x - x_0)(x_1 - x) \left[\frac{\log_{10} e}{2x_0^2} \right] \end{aligned}$$

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,

$$\max_{x_0 \leq x \leq x_1} (x - x_0)(x_1 - x) \leq \frac{h^2}{4}$$

Therefore,

$$0 \leq \log_{10} x - P_1(x) \leq \frac{h^2}{4} \left[\frac{\log_{10} e}{2x_0^2} \right] \doteq .0543 \frac{h^2}{x_0^2}$$

If we want a *uniform bound* for all points $1 \leq x_0 \leq 10$, we have

$$0 \leq \log_{10} x - P_1(x) \leq \frac{h^2 \log_{10} e}{8} \doteq .0543 h^2$$

$$0 \leq \log_{10} x - P_1(x) \leq .0543 h^2$$

For $h = .01$, as is typical of the high school text book tables of $\log_{10} x$,

$$0 \leq \log_{10} x - P_1(x) \leq 5.43 \times 10^{-6}$$

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 $\deg(P_n) \leq n$

$$P_n(x_i) = f(x_i), \quad i = 0, 1, \dots, n$$

with distinct node points $\{x_0, \dots, x_n\}$ 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, \dots, 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 $\{x, x_0, \dots, x_n\}$.

$$f(x) - P_n(x) = \frac{(x - x_0)(x - x_1) \cdots (x - x_n)}{(n + 1)!} f^{(n+1)}(c_x)$$

As shorthand, introduce

$$\Psi_n(x) = (x - x_0)(x - x_1) \cdots (x - x_n)$$

a polynomial of degree $n + 1$ with roots $\{x_0, \dots, x_n\}$.

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

$$f(x) - P_2(x) = \frac{(x - x_0)(x - x_1)(x - x_2)}{3!} f^{(3)}(c_x) \quad (*)$$

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:

$$x_1 = x_0 + h, \quad x_2 = x_1 + h$$

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

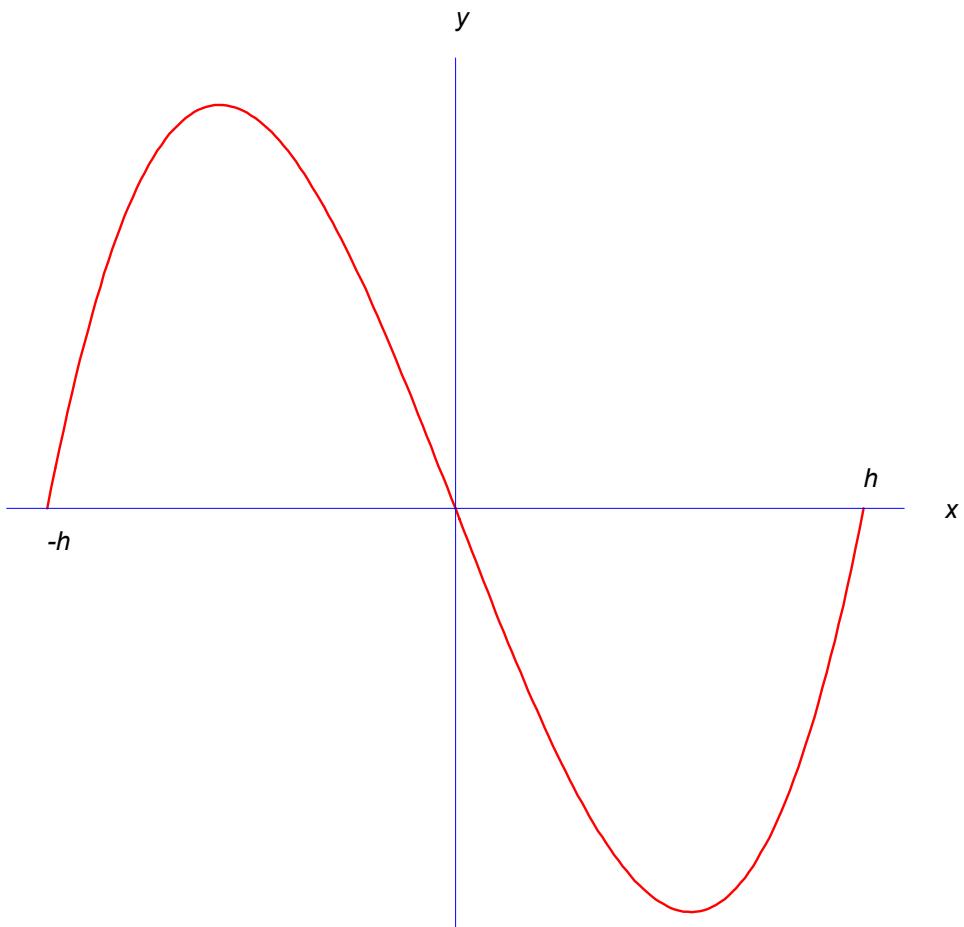
$$\Psi_2(x) = (x - x_0)(x - x_1)(x - x_2)$$

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 $|f^{(3)}(c_x)|$ with a maximum of $|f^{(3)}(x)|$ as x varies over $x_0 \leq x \leq x_2$. This yields

$$|f(x) - P_2(x)| \leq \frac{|\Psi_2(x)|}{3!} \max_{x_0 \leq x \leq x_2} |f^{(3)}(x)| \quad (**)$$

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,

$$\max_{x_0 \leq x \leq x_2} |\Psi_2(x)| = \frac{2h^3}{3 \sqrt{3}}, \quad \text{at } x = x_1 \pm \frac{h}{\sqrt{3}}$$

Combined with (**), this yields

$$|f(x) - P_2(x)| \leq \frac{h^3}{9 \sqrt{3}} \max_{x_0 \leq x \leq x_2} |f^{(3)}(x)|$$

for $x_0 \leq x \leq x_2$.

For $f(x) = \log_{10} x$, with $1 \leq x_0 \leq x \leq x_2 \leq 10$, this leads to

$$\begin{aligned} |\log_{10} x - P_2(x)| &\leq \frac{h^3}{9 \sqrt{3}} \cdot \max_{x_0 \leq x \leq x_2} \frac{2 \log_{10} e}{x^3} \\ &= \frac{.05572 h^3}{x_0^3} \end{aligned}$$

For the case of $h = .01$, we have

$$|\log_{10} x - P_2(x)| \leq \frac{5.57 \times 10^{-8}}{x_0^3} \leq 5.57 \times 10^{-8}$$

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 $\{x, x_0, \dots, x_n\}$. When bounding the error we replace $f^{(n+1)}(c_x)$ with its maximum over the interval containing $\{x, x_0, \dots, x_n\}$, 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 $\{x, x_0, \dots, x_n\}$. 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 $|f^{(n+1)}(x)|$ 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?

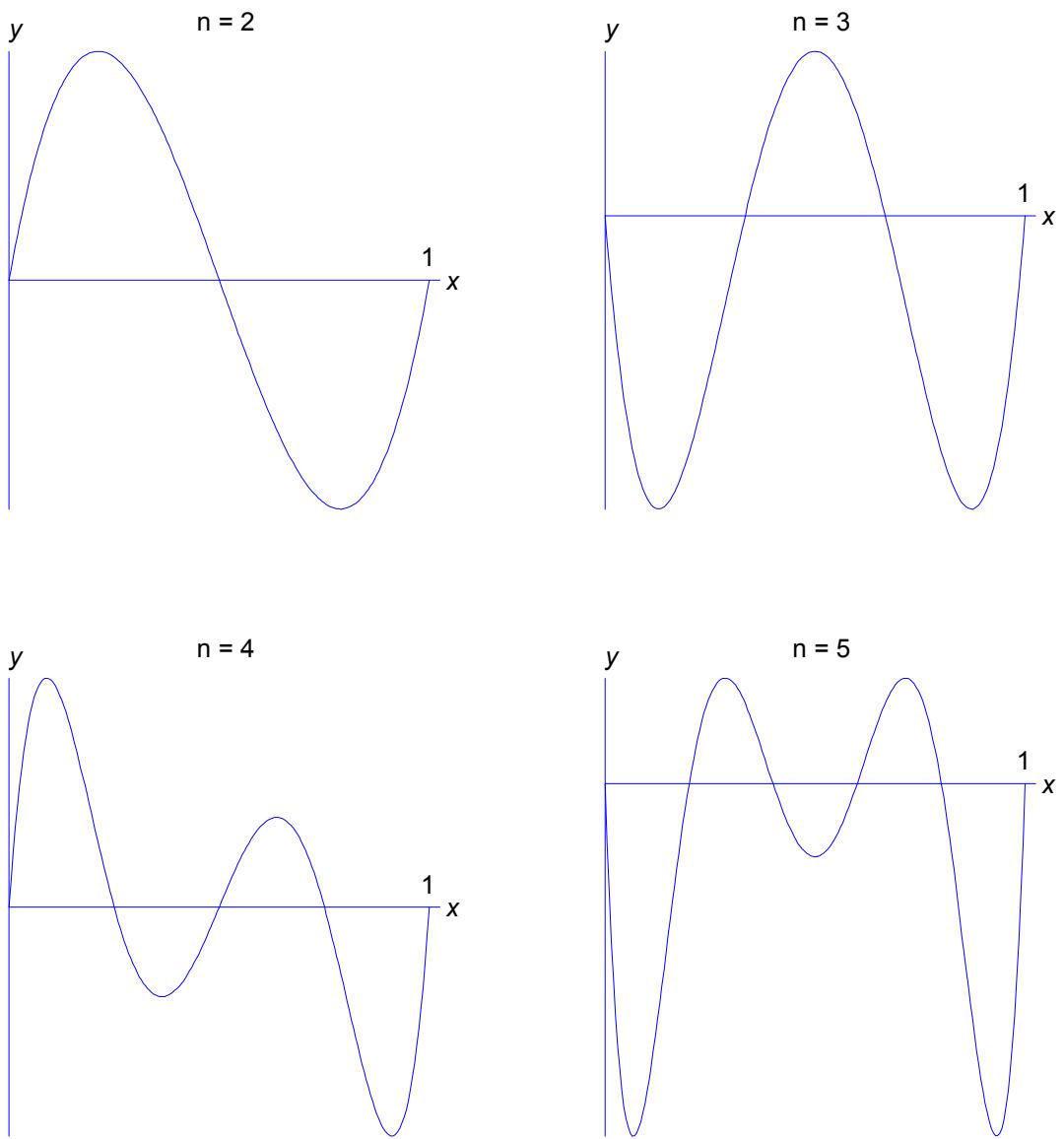
The interpolation nodes are determined by using

$$h = \frac{1}{n}, \quad x_0 = 0, \quad x_1 = h, \quad x_2 = 2h, \dots, \quad x_n = nh = 1$$

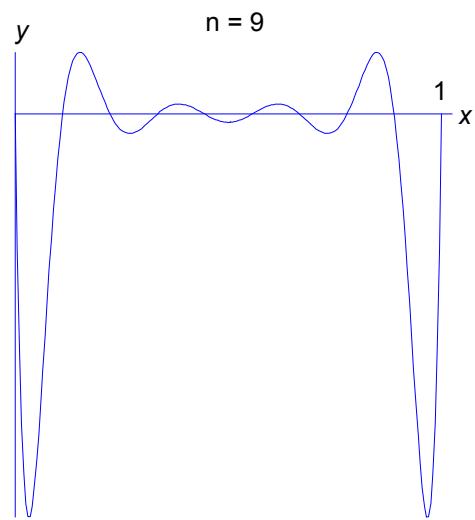
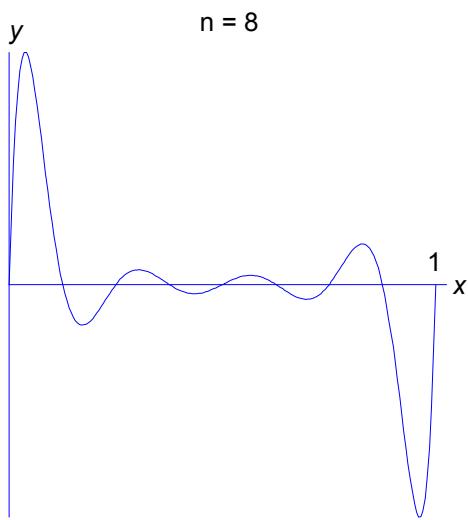
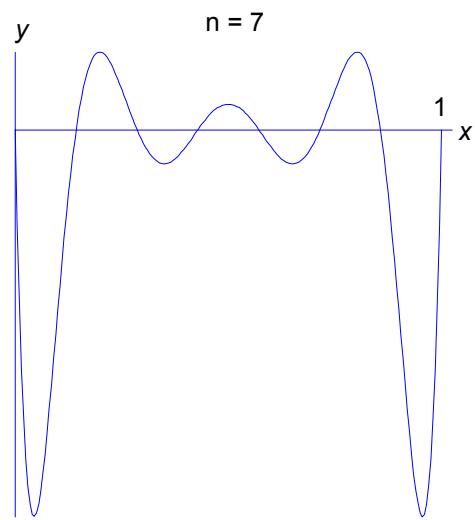
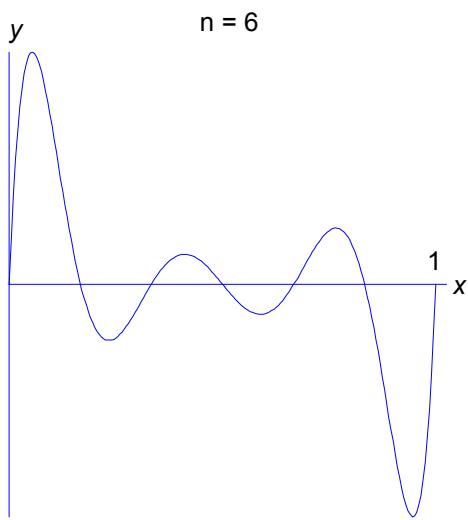
For this case,

$$\Psi_n(x) = x(x - h)(x - 2h) \cdots (x - 1)$$

Our graphs are the cases of $n = 2, \dots, 9$.



Graphs of $\Psi_n(x)$ on $[0, 1]$ for $n = 2, 3, 4, 5$

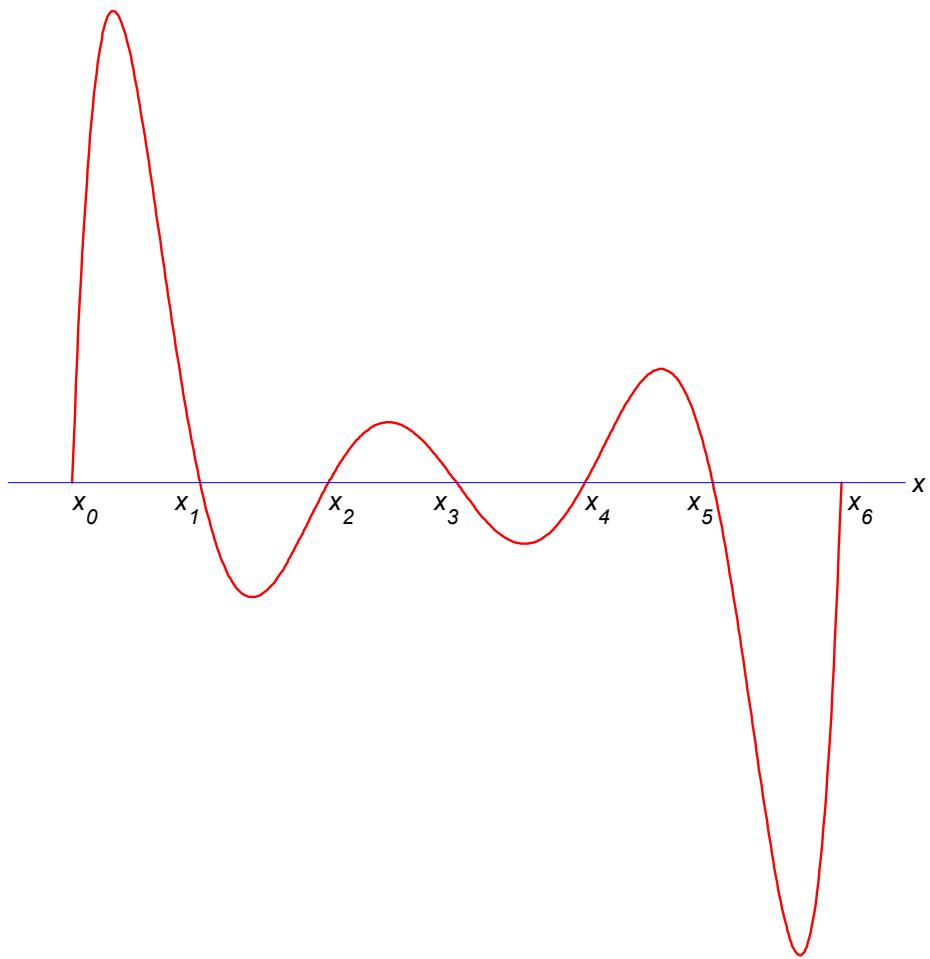


Graphs of $\Psi_n(x)$ on $[0, 1]$ for $n = 6, 7, 8, 9$

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,

$$\begin{aligned}\max_{x_0 \leq x \leq x_1} \frac{|\Psi_n(x)|}{(n+1)!} &= 3.39 \times 10^{-11} \\ \max_{x_4 \leq x \leq x_5} \frac{|\Psi_n(x)|}{(n+1)!} &= 6.89 \times 10^{-13}\end{aligned}$$

and the ratio of these two errors is approximately 49. Thus the interpolation error is likely to be around 49 times larger when $x_0 \leq x \leq x_1$ as compared to the case when $x_4 \leq x \leq x_5$. When doing table interpolation, the point x at which you are interpolating should be centrally located with respect to the interpolation nodes $\{x_0, \dots, x_n\}$ 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

$$a \leq x_0 < x_1 < \cdots < x_{n-1} < x_n \leq b$$

and produce the interpolation polynomial $P_n(x)$ that interpolates $f(x)$ at the given node points. We would like to have

$$\max_{a \leq x \leq b} |f(x) - P_n(x)| \rightarrow 0 \quad \text{as} \quad n \rightarrow \infty$$

Does it happen?

Recall the error bound

$$\begin{aligned} \max_{a \leq x \leq b} |f(x) - P_n(x)| \\ \leq \max_{a \leq x \leq b} \frac{|\Psi_n(x)|}{(n+1)!} \cdot \max_{a \leq x \leq b} |f^{(n+1)}(x)| \end{aligned}$$

We begin with an example using evenly spaced node points.

RUNGE'S EXAMPLE

Use evenly spaced node points:

$$h = \frac{b - a}{n}, \quad x_i = a + ih \quad \text{for } i = 0, \dots, n$$

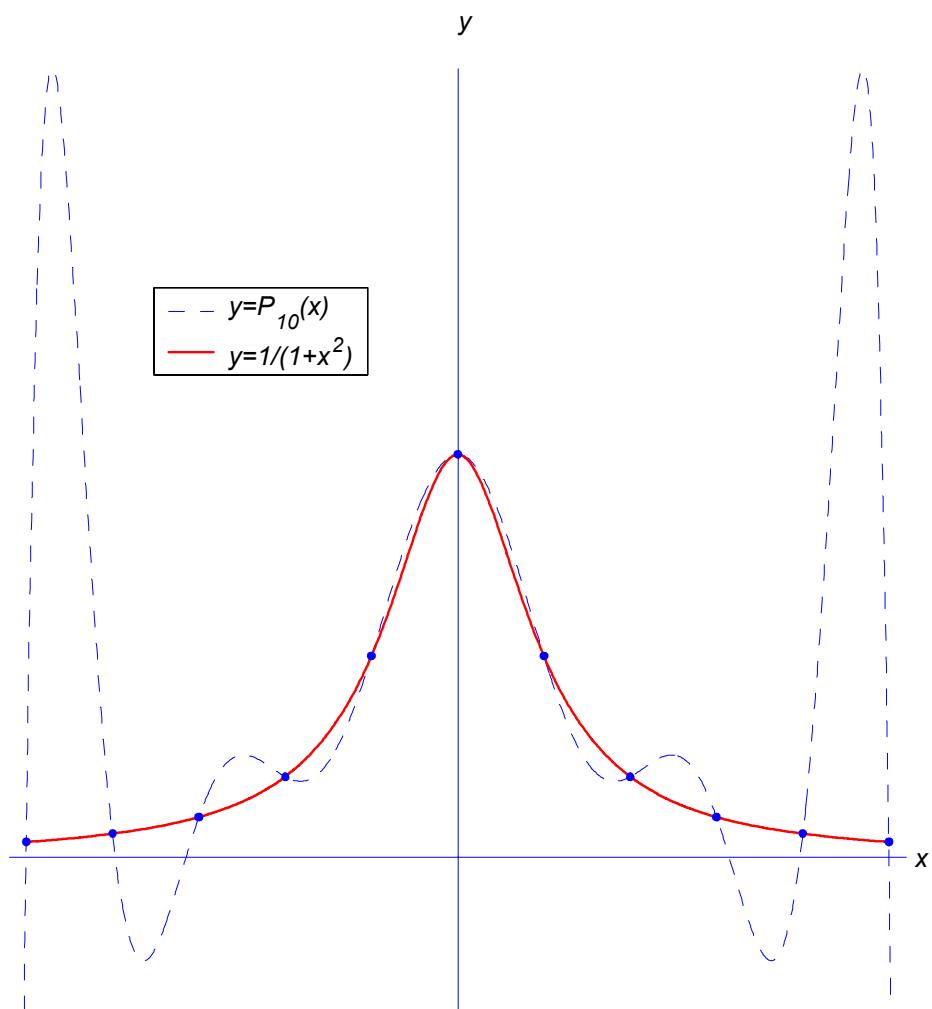
For some functions, such as $f(x) = e^x$, the maximum error goes to zero quite rapidly. But the size of the derivative term $f^{(n+1)}(x)$ in

$$\begin{aligned} & \max_{a \leq x \leq b} |f(x) - P_n(x)| \\ & \leq \max_{a \leq x \leq b} \frac{|\Psi_n(x)|}{(n+1)!} \cdot \max_{a \leq x \leq b} |f^{(n+1)}(x)| \end{aligned}$$

can badly hurt or destroy the convergence of other cases.

In particular, we show the graph of $f(x) = 1/(1+x^2)$ 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

$$\max_{a \leq x \leq b} |f(x) - P_n(x)| \leq \max_{a \leq x \leq b} \frac{|\Psi_n(x)|}{(n+1)!} \cdot \max_{a \leq x \leq b} |f^{(n+1)}(x)|$$

There is nothing we really do with the derivative term for f ; but we can examine the way of defining the nodes $\{x_0, \dots, x_n\}$ 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 $\{x_0, \dots, x_n\}$ 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

$$\max_{a \leq x \leq b} |\Psi_n(x)| = \left(\frac{b-a}{2}\right)^{n+1} 2^{-n}$$

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 $\{x_0, \dots, x_n, x\}$.

A second formula is given by

$$f(x) - P_n(x) = \Psi_n(x) f[x_0, \dots, x_n, x]$$

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 $\{x_0, \dots, x_n, x_{n+1}\}$.

Then

$$\begin{aligned} P_{n+1}(x) &= P_n(x) \\ &+ f[x_0, \dots, x_n, x_{n+1}] (x - x_0) \cdots (x - x_n) \end{aligned}$$

Substituting $x = x_{n+1}$, and using the fact that $P_{n+1}(x)$ interpolates $f(x)$ at x_{n+1} , we have

$$f(x_{n+1}) = P_n(x_{n+1}) + f[x_0, \dots, x_n, x_{n+1}] (x_{n+1} - x_0) \cdots (x_{n+1} - x_n)$$

$$f(x_{n+1}) = P_n(x_{n+1}) + f[x_0, \dots, x_n, x_{n+1}] (x_{n+1} - x_0) \cdots (x_{n+1} - x_n)$$

In this formula, the number x_{n+1} is completely arbitrary, other than being distinct from the points in $\{x_0, \dots, x_n\}$. To emphasize this fact, replace x_{n+1} by x throughout the formula, obtaining

$$\begin{aligned} f(x) &= P_n(x) + f[x_0, \dots, x_n, x] (x - x_0) \cdots (x - x_n) \\ &= P_n(x) + \Psi_n(x) f[x_0, \dots, x_n, x] \end{aligned}$$

provided $x \neq x_0, \dots, x_n$.

The formula

$$\begin{aligned}f(x) &= P_n(x) + f[x_0, \dots, x_n, x] (x - x_0) \cdots (x - x_n) \\&= P_n(x) + \Psi_n(x) f[x_0, \dots, x_n, x]\end{aligned}$$

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

$$f(x) - P_n(x) = \Psi_n(x) f[x_0, \dots, x_n, x]$$

Compare the two error formulas

$$f(x) - P_n(x) = \Psi_n(x) f[x_0, \dots, x_n, x]$$

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

Then

$$\Psi_n(x) f[x_0, \dots, x_n, x] = \frac{\Psi_n(x)}{(n+1)!} f^{(n+1)}(c)$$

$$f[x_0, \dots, x_n, x] = \frac{f^{(n+1)}(c)}{(n+1)!}$$

for some c between the smallest and largest of the numbers in $\{x_0, \dots, x_n, x\}$.

To make this somewhat symmetric in its arguments, let $m = n + 1$, $x = x_{n+1}$. Then

$$f[x_0, \dots, x_{m-1}, x_m] = \frac{f^{(m)}(c)}{m!}$$

with c an unknown number between the smallest and largest of the numbers in $\{x_0, \dots, x_m\}$. 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

$$E(p) = \max_{a \leq x \leq b} |f(x) - p(x)|$$

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:

$$\begin{aligned}\rho_n(f) &= \min_{\deg(p) \leq n} E(p) \\ &= \min_{\deg(p) \leq n} \left[\max_{a \leq x \leq b} |f(x) - p(x)| \right]\end{aligned}$$

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 $E(m_n)$.

n	Maximum Error in:	
	$t_n(x)$	$m_n(x)$
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

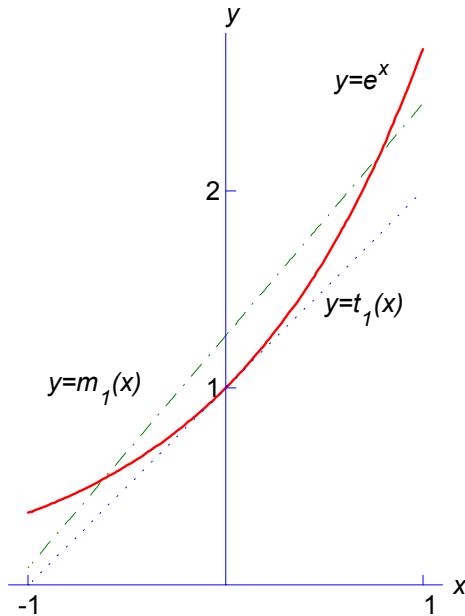
Consider graphically how we can improve on the Taylor polynomial

$$t_1(x) = 1 + x$$

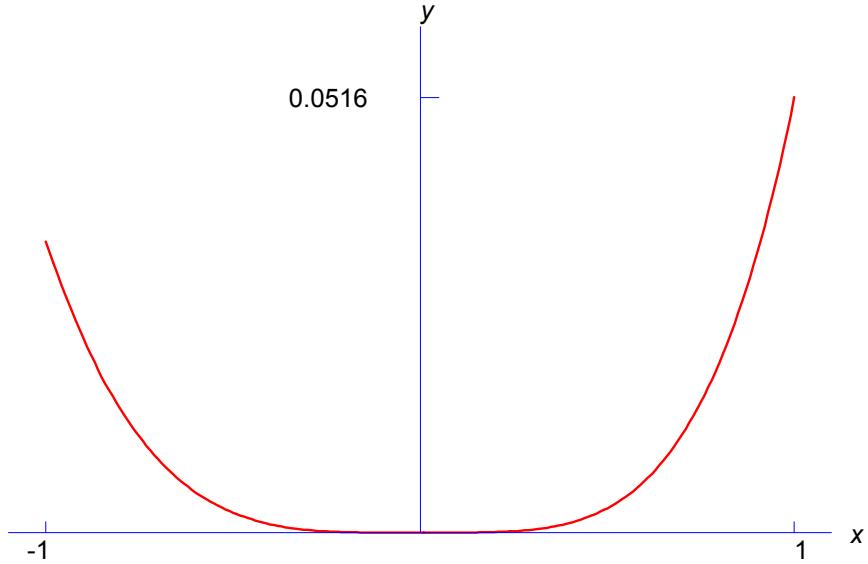
as a uniform approximation to e^x on the interval $[-1, 1]$.

The linear minimax approximation is

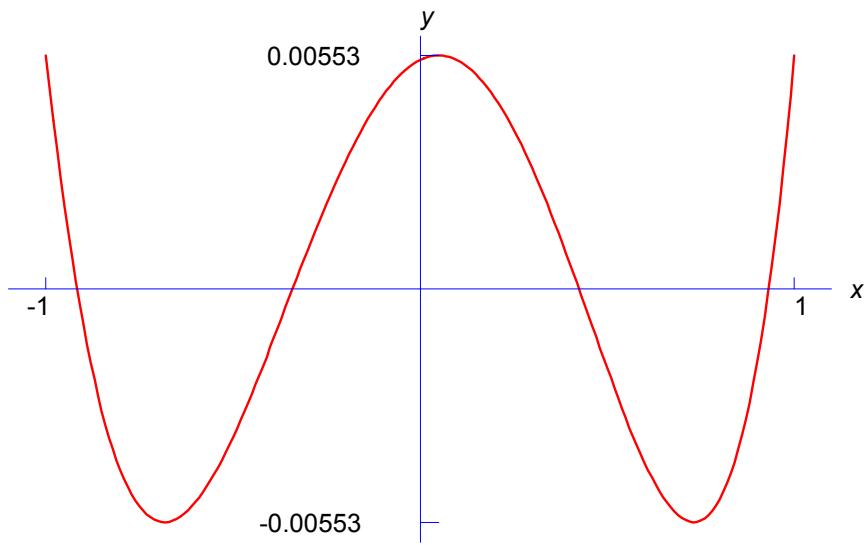
$$m_1(x) = 1.2643 + 1.1752x$$



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.

$$\rho_n(f) \leq \frac{[(b-a)/2]^{n+1}}{(n+1)!2^n} \max_{a \leq x \leq b} |f^{(n+1)}(x)|$$

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

$$\rho_n(e^x) \leq \frac{e}{(n+1)!2^n} \tag{*}$$

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

$$T_n(x) = \cos(n \cos^{-1} x), \quad -1 \leq x \leq 1 \quad (1)$$

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

$$\theta = \cos^{-1}(x) \quad \text{or} \quad x = \cos(\theta), \quad 0 \leq \theta \leq \pi \quad (2)$$

Then

$$T_n(x) = \cos(n\theta) \quad (3)$$

Example. $n = 0$

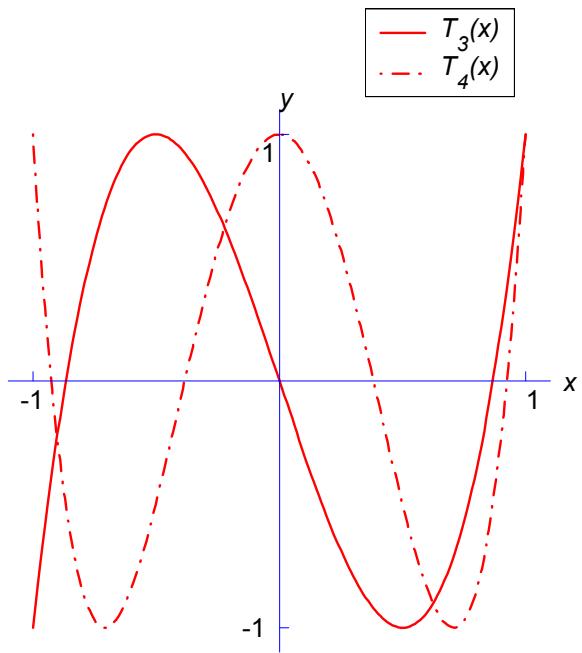
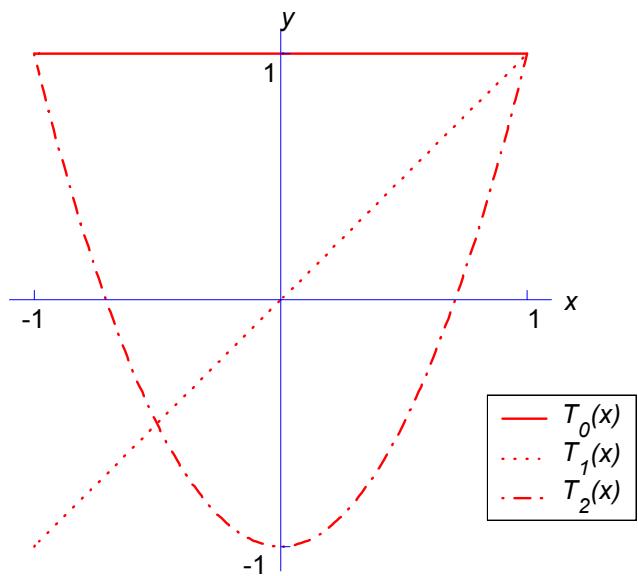
$$T_0(x) = \cos(0 \cdot \theta) = 1$$

$n = 1$

$$T_1(x) = \cos(\theta) = x$$

$n = 2$

$$T_2(x) = \cos(2\theta) = 2\cos^2(\theta) - 1 = 2x^2 - 1$$



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

$$\begin{aligned} T_{n+1}(x) + T_{n-1}(x) &= 2\cos(n\theta)\cos(\theta) = 2xT_n(x) \\ T_{n+1}(x) &= 2xT_n(x) - T_{n-1}(x), \quad n \geq 1 \end{aligned} \tag{4}$$

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

$$T_0(x) = 1, \quad T_1(x) = x$$

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x), \quad n \geq 1$$

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

$$|T_n(x)| \leq 1, \quad -1 \leq x \leq 1 \quad (5)$$

for all $n \geq 0$. Also, note that

$$T_n(x) = 2^{n-1}x^n + \text{lower degree terms}, \quad n \geq 1 \quad (6)$$

This can be proven using the triple recursion relation and mathematical induction.

Introduce a modified version of $T_n(x)$,

$$\tilde{T}_n(x) = \frac{1}{2^{n-1}}T_n(x) = x^n + \text{lower degree terms} \quad (7)$$

From (5) and (6),

$$|\tilde{T}_n(x)| \leq \frac{1}{2^{n-1}}, \quad -1 \leq x \leq 1, \quad n \geq 1 \quad (8)$$

Example.

$$\tilde{T}_4(x) = \frac{1}{8} (8x^4 - 8x^2 + 1) = x^4 - x^2 + \frac{1}{8}$$

A polynomial whose highest degree term has a coefficient of 1 is called a *monic polynomial*. Formula (8) says the monic polynomial $\tilde{T}_n(x)$ has size $1/2^{n-1}$ on $-1 \leq x \leq 1$, and this becomes smaller as the degree n increases. In comparison,

$$\max_{-1 \leq x \leq 1} |x^n| = 1$$

Thus x^n is a monic polynomial whose size does not change with increasing n .

Theorem. Let $n \geq 1$ 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 $\tilde{T}_n(x)$, 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

$$f(x) - c_3(x) = \frac{\omega(x)}{4!} f^{(4)}(\xi_x), \quad -1 \leq x \leq 1 \quad (1)$$

$$\omega(x) = (x - x_0)(x - x_1)(x - x_2)(x - x_3) \quad (2)$$

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

$$\max_{-1 \leq x \leq 1} |\omega(x)| \quad (3)$$

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

$$\omega(x) = \tilde{T}_4(x) = \frac{T_4(x)}{2^3} = \frac{1}{8}(8x^4 - 8x^2 + 1) \quad (4)$$

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

$$T_4(x) = \cos(4\theta) = 0, \quad x = \cos(\theta)$$

$$\begin{aligned} 4\theta &= \pm\frac{\pi}{2}, \pm\frac{3\pi}{2}, \pm\frac{5\pi}{2}, \pm\frac{7\pi}{2}, \dots \\ \theta &= \pm\frac{\pi}{8}, \pm\frac{3\pi}{8}, \pm\frac{5\pi}{8}, \pm\frac{7\pi}{8}, \dots \\ x &= \cos\left(\frac{\pi}{8}\right), \cos\left(\frac{3\pi}{8}\right), \cos\left(\frac{5\pi}{8}\right), \dots \end{aligned} \quad (5)$$

using $\cos(-\theta) = \cos(\theta)$.

$$x = \cos\left(\frac{\pi}{8}\right), \cos\left(\frac{3\pi}{8}\right), \cos\left(\frac{5\pi}{8}\right), \cos\left(\frac{7\pi}{8}\right), \dots$$

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

$$\{x_0, x_1, x_2, x_3\} = \{\pm 0.382683, \pm 0.923880\} \quad (6)$$

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

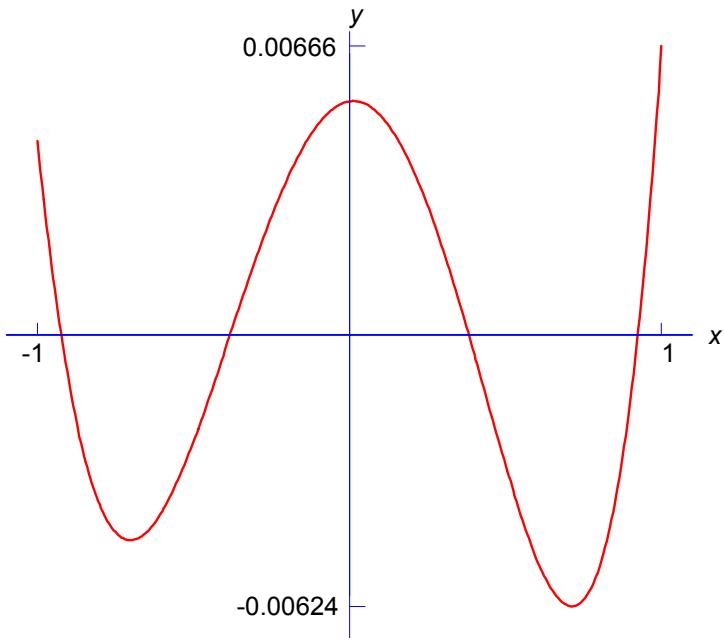
$$\begin{aligned} \max_{-1 \leq x \leq 1} |f(x) - c_3(x)| &\leq \frac{1/2^3}{4!} \max_{-1 \leq x \leq 1} e^{\xi_x} \\ &\leq \frac{e}{192} \doteq 0.014158 \end{aligned}$$

By direct calculation,

$$\max_{-1 \leq x \leq 1} |e^x - c_3(x)| \doteq 0.00666$$

Interpolation Data: $f(x) = e^x$

i	x_i	$f(x_i)$	$f[x_0, \dots, x_i]$
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 $\{x_0, \dots, x_n\}$ in $[-1, 1]$. Denote the interpolation polynomial by $c_n(x)$. The interpolation error on $[-1, 1]$ is given by

$$\begin{aligned} f(x) - c_n(x) &= \frac{\omega(x)}{(n+1)!} f^{(n+1)}(\xi_x) \quad (7) \\ \omega(x) &= (x - x_0) \cdots (x - x_n) \end{aligned}$$

with ξ_x and unknown point in $[-1, 1]$. In order to minimize the interpolation error, we seek to minimize

$$\max_{-1 \leq x \leq 1} |\omega(x)| \quad (8)$$

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

$$\tilde{T}_{n+1}(x) = \frac{1}{2^n} T_{n+1}(x)$$

Thus the interpolation nodes are the zeros of $T_{n+1}(x)$; and by the procedure that led to (5), they are given by

$$x_j = \cos\left(\frac{2j+1}{2n+2}\pi\right), \quad j = 0, 1, \dots, n \quad (9)$$

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 e^x - c_n(x) $	$\rho_n(e^x)$
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

$$\max_{-1 \leq x \leq 1} |f(x) - c_n(x)| \leq \frac{\max_{-1 \leq x \leq 1} |\omega(x)|}{(n+1)!} \max_{-1 \leq \xi \leq 1} |f(\xi)|$$

From the theorem of the preceding section,

$$\max_{-1 \leq x \leq 1} |\tilde{T}_{n+1}(x)| = \max_{-1 \leq x \leq 1} |\omega(x)| = \frac{1}{2^n}$$

in this case. Thus

$$\max_{-1 \leq x \leq 1} |f(x) - c_n(x)| \leq \frac{1}{(n+1)!2^n} \max_{-1 \leq \xi \leq 1} |f(\xi)|$$

OTHER INTERVALS

Consider approximating $f(x)$ on the finite interval $[a, b]$. Introduce the linear change of variables

$$x = \frac{1}{2}[(1-t)a + (1+t)b] \quad (10)$$

$$t = \frac{2}{b-a} \left[x - \frac{b+a}{2} \right] \quad (11)$$

Introduce

$$F(t) = f\left(\frac{1}{2}[(1-t)a + (1+t)b]\right), \quad -1 \leq t \leq 1$$

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

$$F(t) = \cos\left(\frac{1+t}{4}\pi\right), \quad -1 \leq t \leq 1$$

PIECEWISE POLYNOMIAL INTERPOLATION

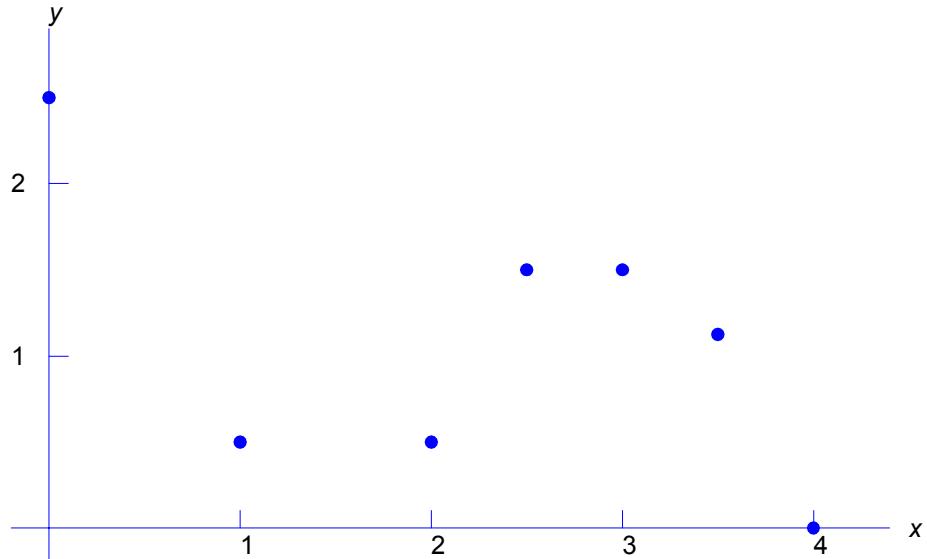
Recall the examples of higher degree polynomial interpolation of the function $f(x) = (1 + x^2)^{-1}$ on $[-5, 5]$. The interpolants $P_n(x)$ oscillated a great deal, whereas the function $f(x)$ was nonoscillatory. To obtain interpolants that are better behaved, we look at other forms of interpolating functions.

Consider the data

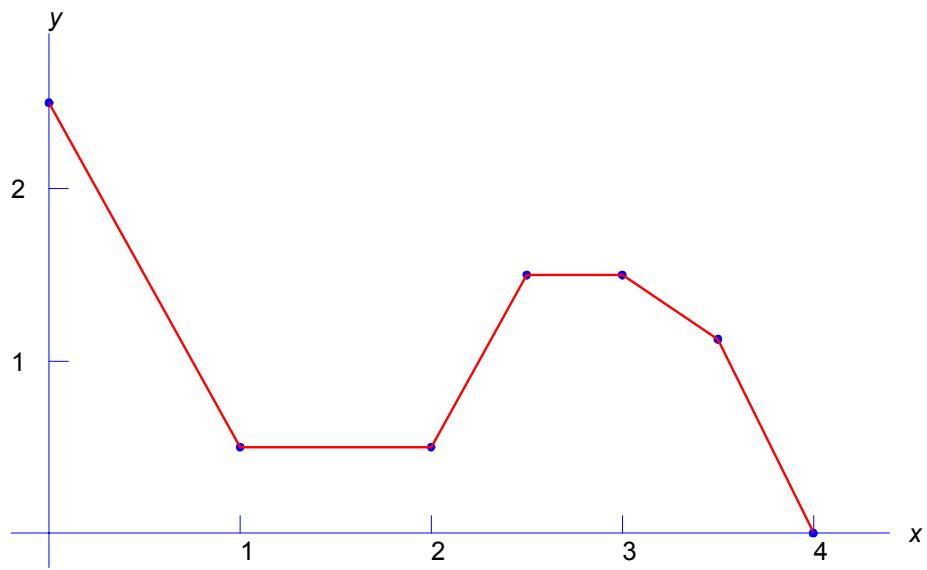
x	0	1	2	2.5	3	3.5	4
y	2.5	0.5	0.5	1.5	1.5	1.125	0

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.

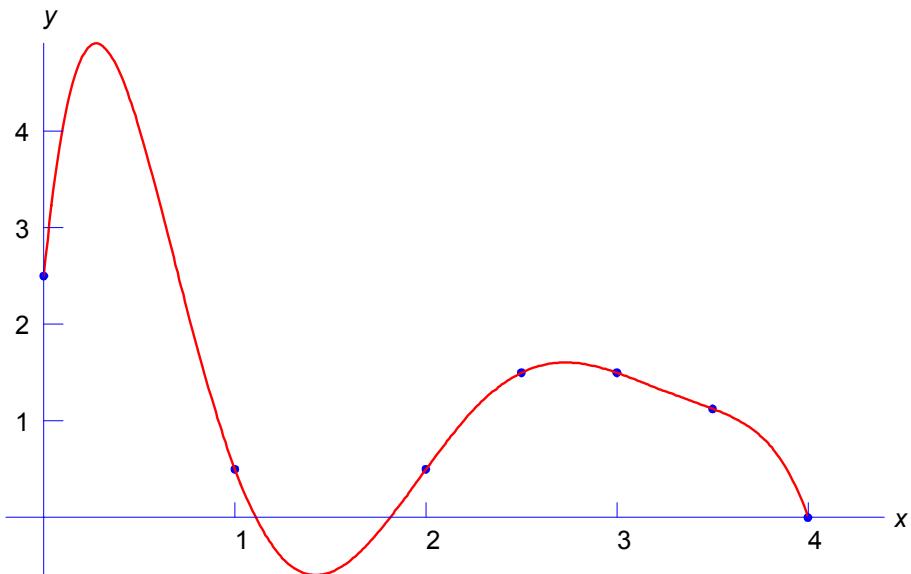
Since we only have the data to consider, we would generally want to use an interpolant that had somewhat the shape of that of the piecewise linear interpolant.



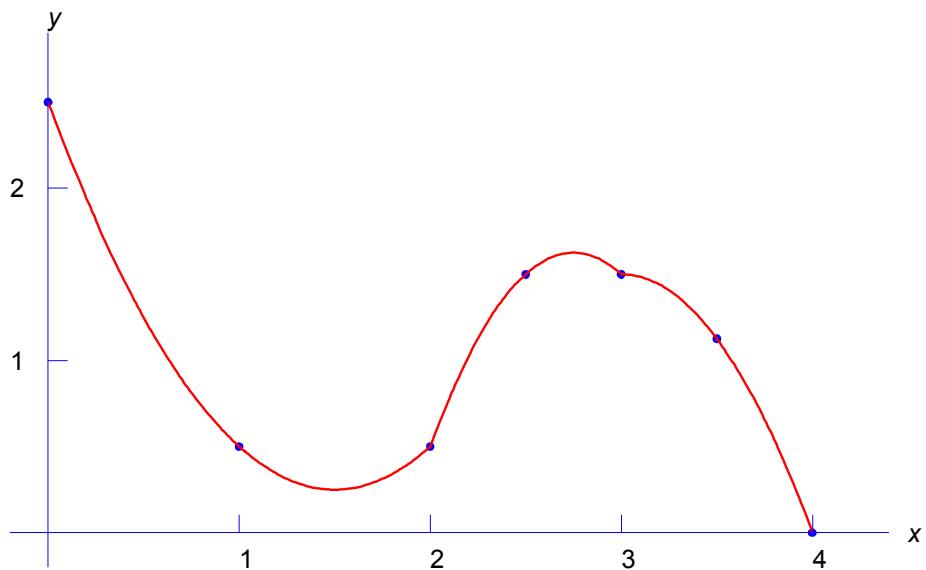
The data points



Piecewise linear interpolation



Polynomial Interpolation



Piecewise quadratic interpolation

PIECEWISE POLYNOMIAL FUNCTIONS

Consider being given a set of data points $(x_1, y_1), \dots, (x_n, y_n)$, with

$$x_1 < x_2 < \dots < x_n$$

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 $\{(x_j, y_j)\}$. 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

$$[x_1, x_3], [x_3, x_5], [x_5, x_7], \dots$$

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 $(x_1, y_1), \dots, (x_n, y_n)$ be given, as let

$$x_1 < x_2 < \dots < x_n$$

Consider finding functions $s(x)$ for which the following properties hold:

- (1) $s(x_i) = y_i, \quad i = 1, \dots, n$
- (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

$$\int_{x_1}^{x_n} |s''(x)|^2 dx$$

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

$$[a, x_1], [x_1, x_2], \dots, [x_{n-1}, x_n], [x_n, b]$$

$s(x)$ 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 $(x_1, y_1), \dots, (x_n, y_n)$, we say $s(x)$ is a cubic interpolating spline function for this data if

3. $s(x_i) = y_i, \quad i = 1, \dots, n.$

Return to the earlier problem of choosing an interpolating function $s(x)$ to minimize the integral

$$\int_{x_1}^{x_n} |s''(x)|^2 dx$$

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 $[x_1, x_2]$,

$$s''(x) = \frac{(x_2 - x) M_1 + (x - x_1) M_2}{x_2 - x_1}, \quad x_1 \leq x \leq x_2$$

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

$$s(x_1) = y_1, \quad s(x_2) = y_2 \quad (*)$$

Then

$$\begin{aligned} s(x) &= \frac{(x_2 - x)^3 M_1 + (x - x_1)^3 M_2}{6(x_2 - x_1)} \\ &\quad + \frac{(x_2 - x)y_1 + (x - x_1)y_2}{x_2 - x_1} \\ &\quad - \frac{x_2 - x_1}{6} [(x_2 - x)M_1 + (x - x_1)M_2] \end{aligned}$$

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 $x_2 \leq x \leq x_3$,

$$s(x) = \frac{(x_3 - x)^3 M_2 + (x - x_2)^3 M_3}{6(x_3 - x_2)} \\ + \frac{(x_3 - x)y_2 + (x - x_2)y_3}{x_3 - x_2} \\ - \frac{x_3 - x_2}{6} [(x_3 - x)M_2 + (x - x_2)M_3]$$

For $x_3 \leq x \leq x_4$,

$$s(x) = \frac{(x_4 - x)^3 M_3 + (x - x_3)^3 M_4}{6(x_4 - x_3)} \\ + \frac{(x_4 - x)y_3 + (x - x_3)y_4}{x_4 - x_3} \\ - \frac{x_4 - x_3}{6} [(x_4 - x)M_3 + (x - x_3)M_4]$$

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

$$s(x_2) = y_2, \quad s''(x_2) = M_2$$

The formula on $[x_2, x_3]$ also yields

$$s(x_2) = y_2, \quad s''(x_2) = M_2$$

All that is lacking is to make $s'(x)$ continuous at x_2 and x_3 . Thus we require

$$\begin{aligned} s'(x_2 + 0) &= s'(x_2 - 0) \\ s'(x_3 + 0) &= s'(x_3 - 0) \end{aligned} \tag{**}$$

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

$$\begin{aligned} s(x) &= \frac{(x_2 - x)^3 M_1 + (x - x_1)^3 M_2}{6h} \\ &\quad + \frac{(x_2 - x) y_1 + (x - x_1) y_2}{h} \\ &\quad - \frac{h}{6} [(x_2 - x) M_1 + (x - x_1) M_2] \end{aligned}$$

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.

$$\begin{aligned} \frac{h}{6}M_1 + \frac{2h}{3}M_2 + \frac{h}{6}M_3 \\ = \frac{y_3 - y_2}{h} - \frac{y_2 - y_1}{h} \\ \frac{h}{6}M_2 + \frac{2h}{3}M_3 + \frac{h}{6}M_4 \\ = \frac{y_4 - y_3}{h} - \frac{y_3 - y_2}{h} \end{aligned}$$

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

$$M_2 = \frac{1}{2}, \quad M_3 = 0$$

This leads to the spline function formula on each subinterval.

On $[1, 2]$,

$$\begin{aligned}
 s(x) &= \frac{(x_2 - x)^3 M_1 + (x - x_1)^3 M_2}{6h} \\
 &\quad + \frac{(x_2 - x) y_1 + (x - x_1) y_2}{h} \\
 &\quad - \frac{h}{6} [(x_2 - x) M_1 + (x - x_1) M_2] \\
 &= \frac{(2 - x)^3 \cdot 0 + (x - 1)^3 \left(\frac{1}{2}\right)}{6} + \frac{(2 - x) \cdot 1 + (x - 1) \left(\frac{1}{2}\right)}{1} \\
 &\quad - \frac{1}{6} \left[(2 - x) \cdot 0 + (x - 1) \left(\frac{1}{2}\right) \right] \\
 &= \frac{1}{12} (x - 1)^3 - \frac{7}{12} (x - 1) + 1
 \end{aligned}$$

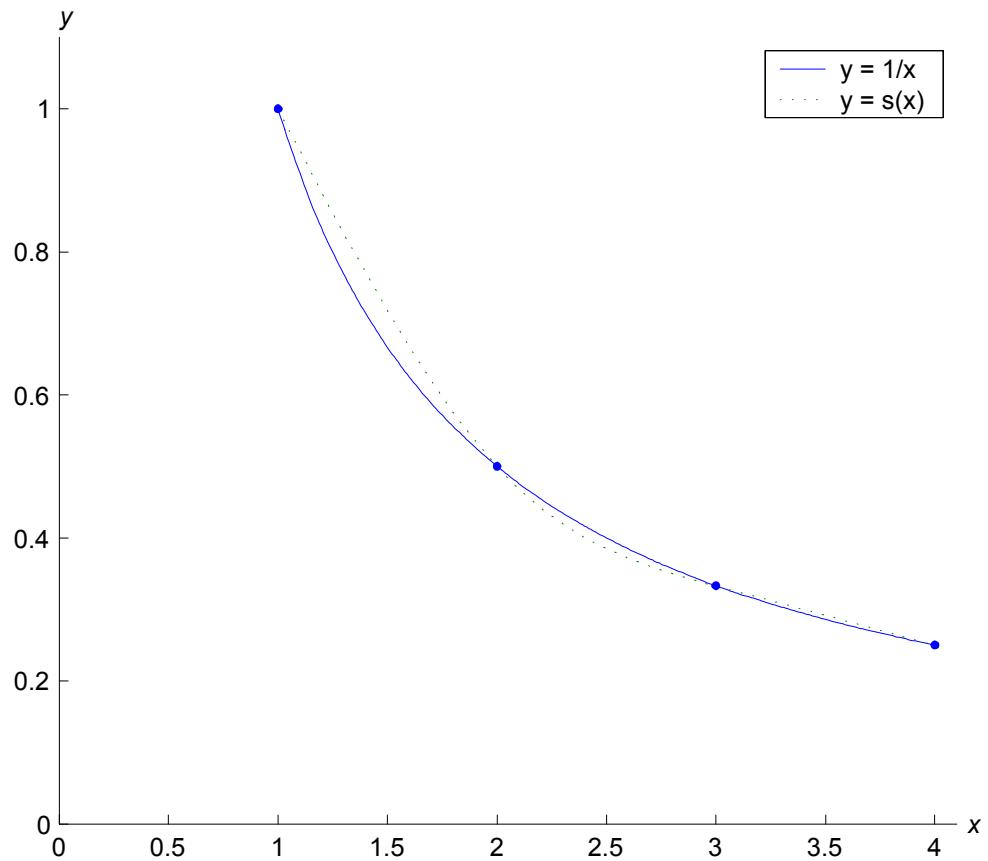
Similarly, for $2 \leq x \leq 3$,

$$s(x) = \frac{-1}{12} (x - 2)^3 + \frac{1}{4} (x - 2)^2 - \frac{1}{3} (x - 1) + \frac{1}{2}$$

and for $3 \leq x \leq 4$,

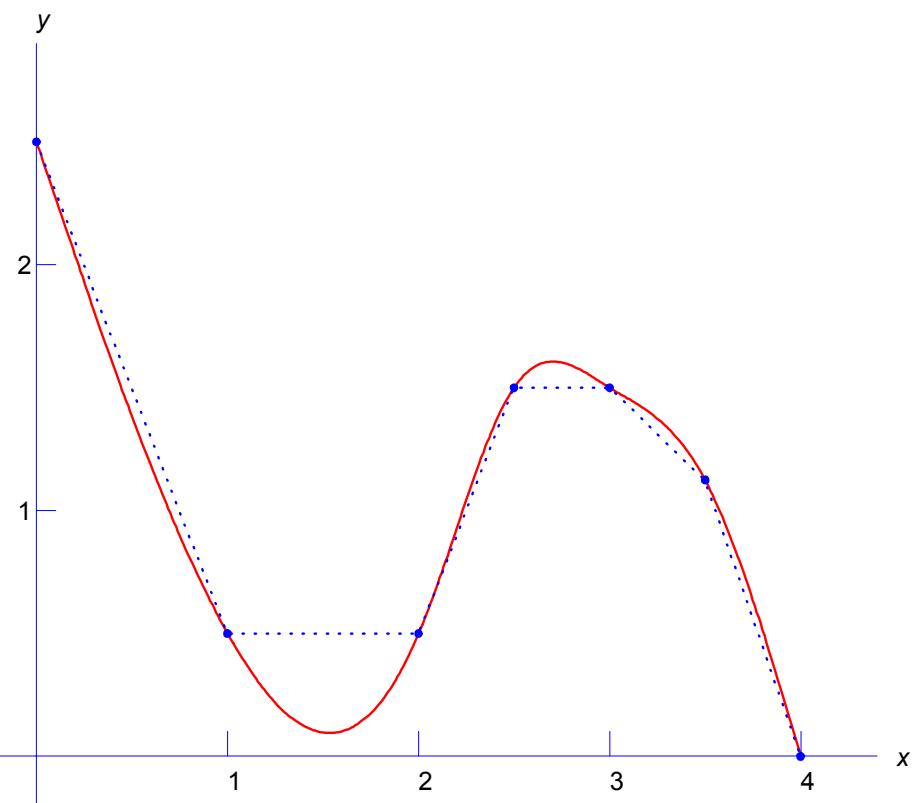
$$s(x) = \frac{-1}{12} (x - 4) + \frac{1}{4}$$

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 $x_1 \leq x \leq x_2$,

$$s(x) = \frac{(x_2 - x)^3 M_1 + (x - x_1)^3 M_2}{6h} \\ + \frac{(x_2 - x) y_1 + (x - x_1) y_2}{h} \\ - \frac{h}{6} [(x_2 - x) M_1 + (x - x_1) M_2]$$

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,

$$f(x) = \frac{1}{x}, \quad f\left(\frac{3}{2}\right) = \frac{2}{3}, \quad s\left(\frac{3}{2}\right) = .65260$$

This is quite a decent approximation.

THE GENERAL PROBLEM

Consider the spline interpolation problem with n nodes

$$(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$$

and assume the node points $\{x_i\}$ are evenly spaced,

$$x_j = x_1 + (j - 1) h, \quad j = 1, \dots, n$$

We have that the interpolating spline $s(x)$ on $x_j \leq x \leq x_{j+1}$ is given by

$$\begin{aligned} s(x) = & \frac{(x_{j+1} - x)^3 M_j + (x - x_j)^3 M_{j+1}}{6h} \\ & + \frac{(x_{j+1} - x) y_j + (x - x_j) y_{j+1}}{h} \\ & - \frac{h}{6} [(x_{j+1} - x) M_j + (x - x_j) M_{j+1}] \end{aligned}$$

for $j = 1, \dots, n - 1$.

To enforce continuity of $s'(x)$ at the interior node points x_2, \dots, x_{n-1} , the second derivatives $\{M_j\}$ 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 $j = 2, \dots, n - 1$. 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 $\{M_1, \dots, M_n\}$. 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

$$s'(x_1) = y'_1, \quad s'(x_n) = y'_n$$

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

$$(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$$

We separate these points into two categories. For constructing the interpolating cubic spline function, we use the points

$$(x_1, y_1), (x_3, y_3), \dots, (x_{n-2}, y_{n-2}), (x_n, y_n)$$

Thus deleting two of the points. We now have $n - 2$ points, and the interpolating spline $s(x)$ can be determined on the intervals

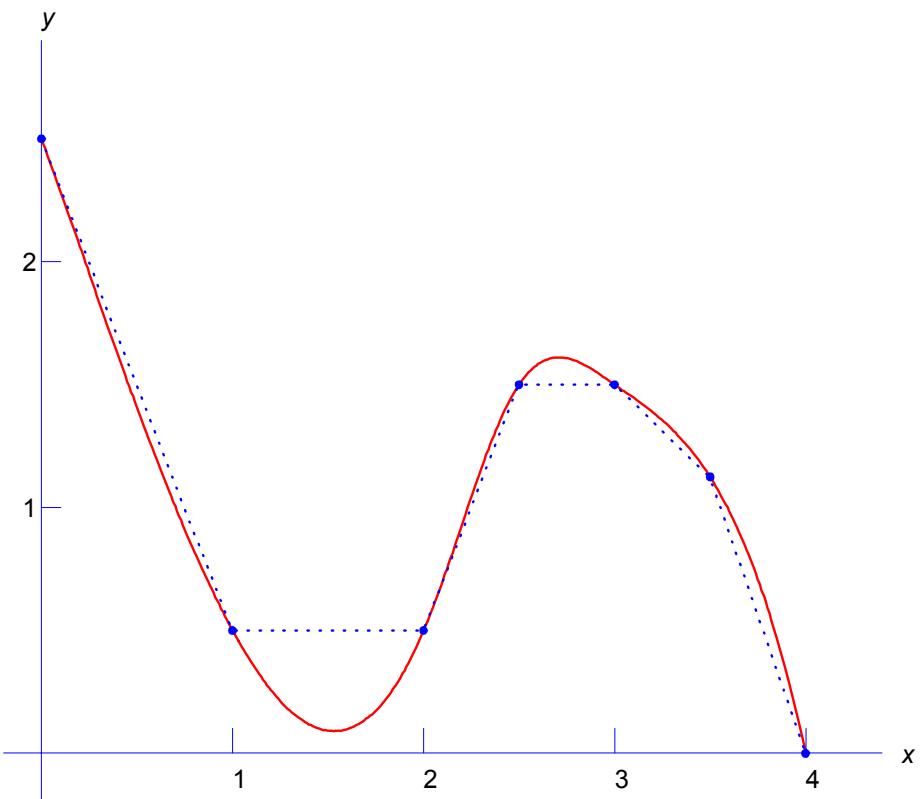
$$[x_1, x_3], [x_3, x_4], \dots, [x_{n-3}, x_{n-2}], [x_{n-2}, x_n]$$

This leads to $n - 4$ equations in the $n - 2$ unknowns $M_1, M_3, \dots, M_{n-2}, M_n$. The two additional boundary conditions are

$$s(x_2) = y_2, \quad s(x_{n-1}) = y_{n-1}$$

These translate into two additional equations, and we obtain a system of $n - 2$ linear simultaneous equations in the $n - 2$ unknowns $M_1, M_3, \dots, M_{n-2}, M_n$.

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

$$(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$$

type arrays containing the x and y coordinates:

$$\begin{aligned}x &= [x_1 \ x_2 \ \dots x_n] \\y &= [y_1 \ y_2 \ \dots y_n] \\plot(x, y, 'o')\end{aligned}$$

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

$$h = (x_n - x_1) / (10 * n); \quad xx = x_1 : h : x_n;$$

use

$$\begin{aligned}yy &= spline(x, y, xx) \\plot(x, y, 'o', xx, yy)\end{aligned}$$

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

$$h = \frac{b - a}{n - 1}, \quad x_j = a + (j - 1)h, \quad j = 1, \dots, n$$

Suppose we want to approximate a given function $f(x)$ on the interval $[a, b]$ using cubic spline interpolation. Define

$$y_i = f(x_i), \quad j = 1, \dots, n$$

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 ,

$$E_n \equiv \max_{a \leq x \leq b} |f(x) - s_n(x)| \leq ch^4$$

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

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

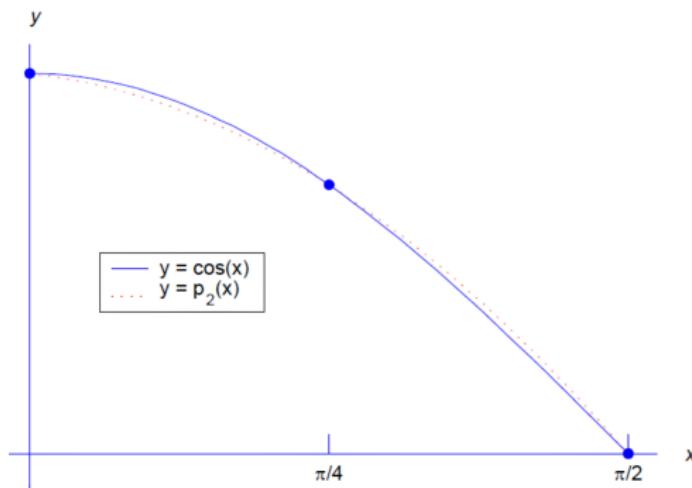
with a function given analytically. Usually this is a polynomial function. In other words, **Purpose 1** consists in finding a polynomial of degree n that passes every given data point.

- 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:

$$P_n(x_i) = y_i, \quad i = 0, 1, \dots, n$$



Polynomial of degree 10 passing 11 data points

Polynomial interpolation

This UNIQUE polynomial can be obtained by (see Lecture 7):

- 1 Lagrange formula:

$$P_n(x) = \sum_{i=0}^n y_i L_i(x), \text{ where } L_i(x) = \prod_{j=0, j \neq i}^n \frac{x - x_j}{x_i - x_j}.$$

- 2 Newton difference formula

$$\begin{aligned} P_n(x) &= P_{n-1}(x) + \\ &+ f[x_0, x_1, x_2, \dots, x_n](x - x_0)(x - x_1) \dots (x - x_{n-1}). \end{aligned}$$

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.

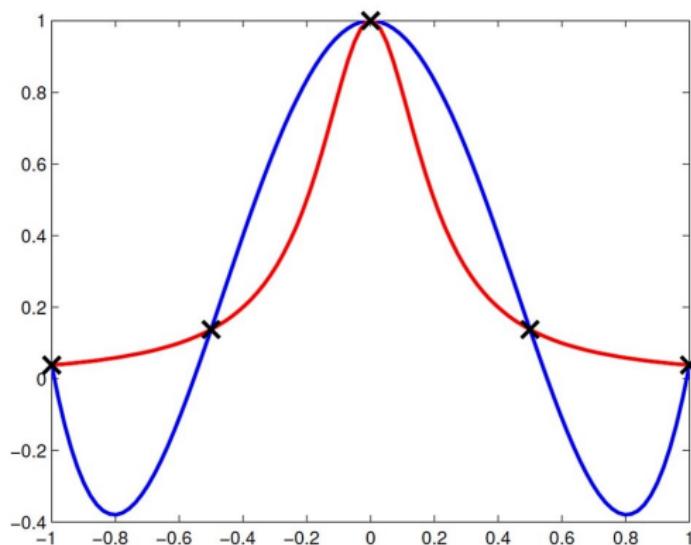
We might use Taylor polynomials, but the error increases as we move toward endpoints and it will be of the same sign (see Taylor polynomial discussion in Lecture 1)

Another approach is by interpolation:

Consider the partition of $[a, b]$: $a \leq x_0 < x_1 < x_2 < \dots < x_n \leq b$ and the data points $\{(x_0, f(x_0)), (x_1, f(x_1)), \dots, (x_n, f(x_n))\}$.

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

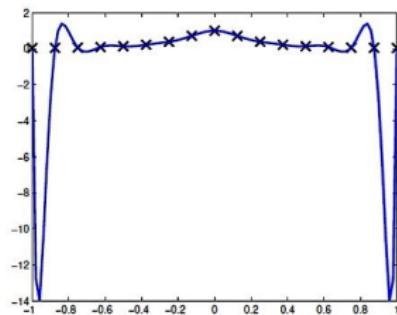
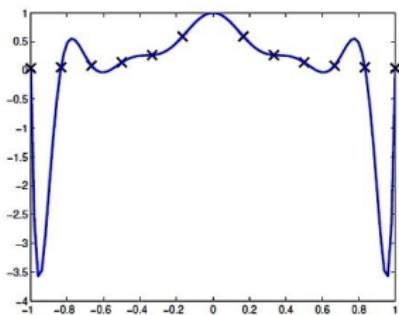
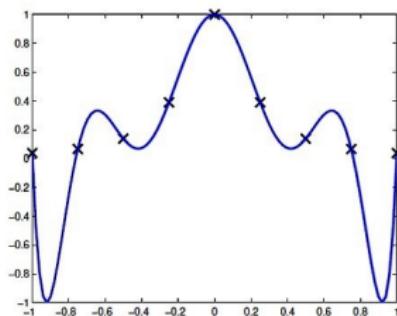
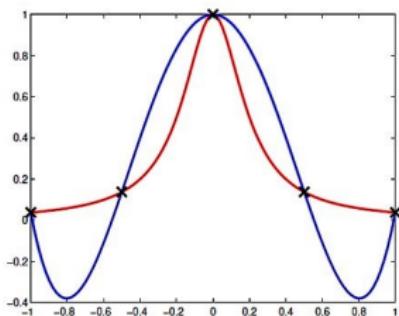
Generally, if we have more interpolation points, and hence the degree of the interpolation polynomial is higher, then the error will be smaller:

$$\|e(x)\|_{\infty} = \|f(x) - P_n(x)\|_{\infty} \rightarrow 0 \text{ as } n \rightarrow \infty.$$

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:

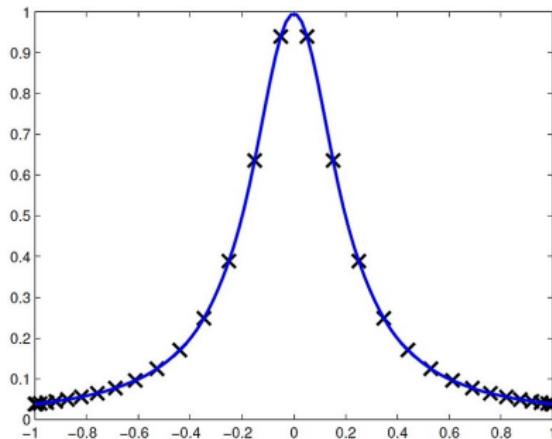
$$\rho_n(f) = \min_{P_n} \|f(x) - P_n(x)\|_\infty = \min_{P_n} \max_{x \in [a,b]} |f(x) - P_n(x)|.$$

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:

$$\mathcal{P} = \{x_0, x_1, \dots, x_n\} \subset [a, b].$$

The Lebesgue constant of \mathcal{P} is defined as,

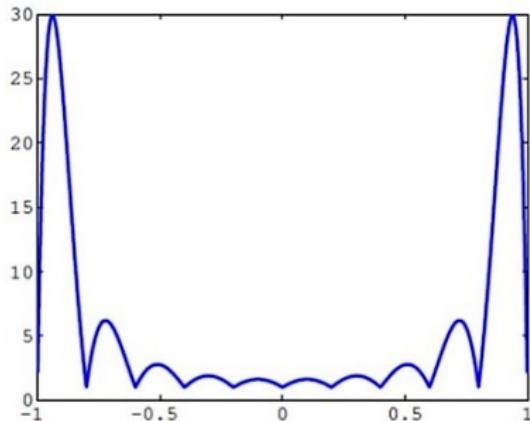
$$\Lambda_n(\mathcal{P}) = \max_{x \in [a, b]} \sum_{k=0}^n |L_k(x)|,$$

where $L_k(x)$ are the Lagrange basis functions associated with interpolation set \mathcal{P} (see Lecture 4).

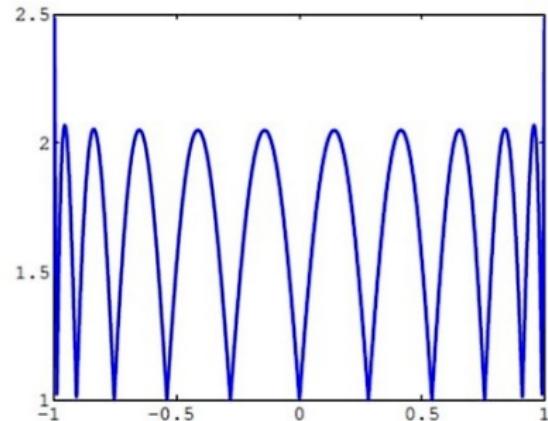
Small Lebesgue constant means that our interpolation can't be much worse than the best polynomial approximation!

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 $[-1, 1]$.



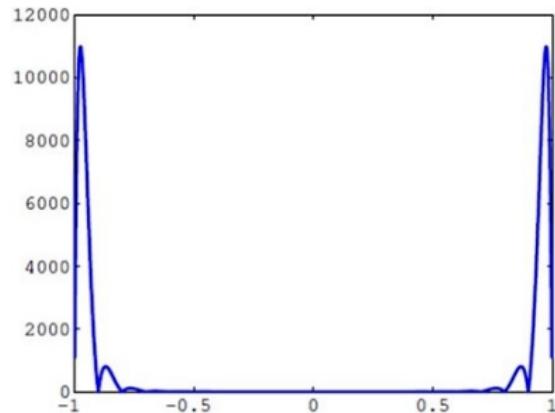
$$\Lambda_{10}(\mathcal{P}_{unif}) \approx 29.9$$



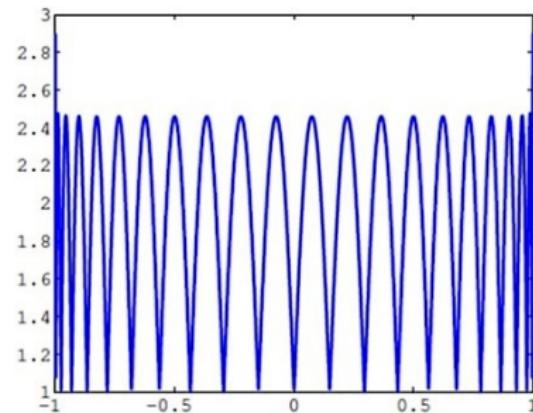
$$\Lambda_{10}(\mathcal{P}_{cheb}) \approx 2.49$$

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 $[-1, 1]$.



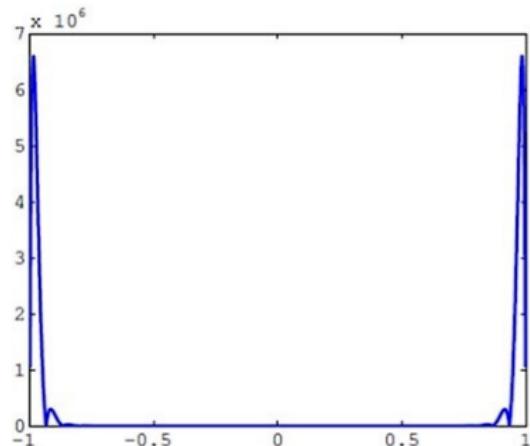
$$\Lambda_{20}(\mathcal{P}_{unif}) \approx 10987$$



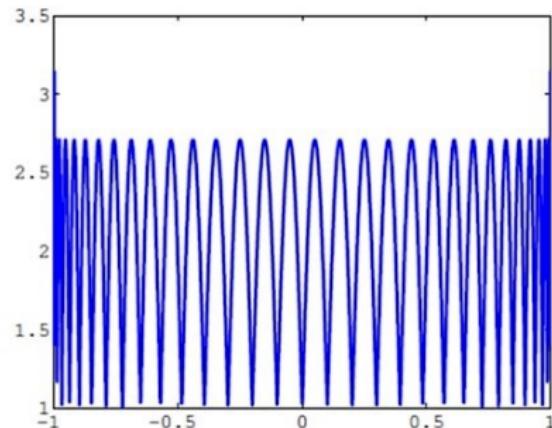
$$\Lambda_{20}(\mathcal{P}_{cheb}) \approx 2.9$$

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 $n \rightarrow \infty$,

$$\Lambda_n(\mathcal{P}_{unif}) \approx \frac{2^n}{e n \log n}, \quad \text{BAD!}$$

whereas

$$\Lambda_n(\mathcal{P}_{cheb}) < \frac{2}{\pi} \log(n+1) + 1. \quad \text{GOOD!}$$

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)

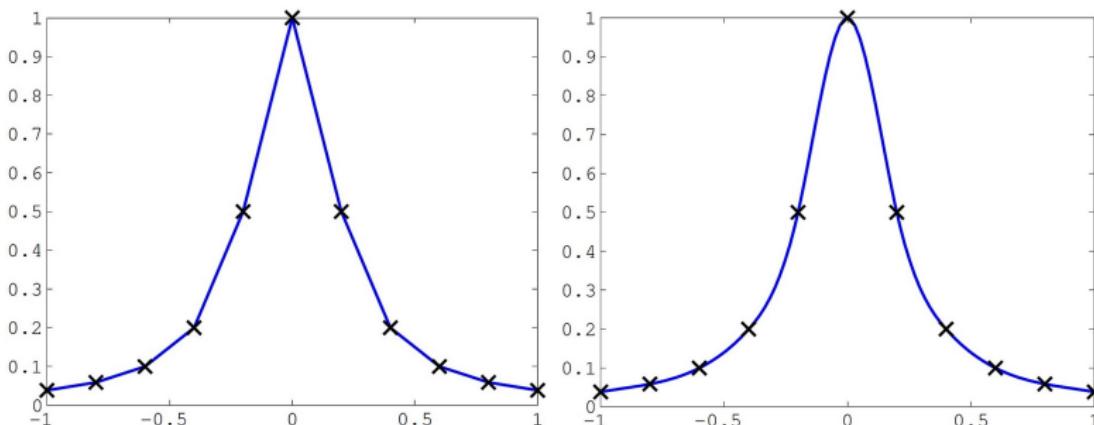
- For a given set of interpolation points, use point 1 methodology to build the approximation $P_n(x)$.
- Interpolation points play an important role on the size of error $\|f(x) - P_n(x)\|_\infty$ (keep in mind Runge example).

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).

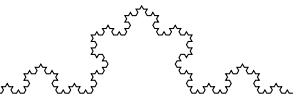


Piecewise linear and cubic spline interpolations of Runge function

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

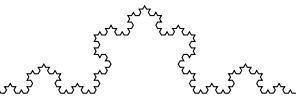
Consider a binary floating-point representation with mantissa containing 3 digits without hiding the leading 1 and $-3_{10} \leq e \leq 3_{10}$.

- 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) $x_A = 6435.4012$, $x_T = 6435.401163$;
- b) $x_A = 0.007245$, $x_T = 0.00723816$;
- c) $x_A = 355/113$, $x_T = \pi$;
- a) $x_A = 2.236$, $x_T = \sqrt{5}$.

**Problem 1.6**

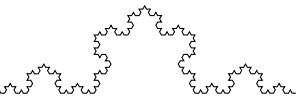
Avoid loss-of-significance errors in the following formulas

- a) $\log(x) - \log(x - 1)$ for large values of x ;
- b) $\frac{e^x - 1}{x}$ for small values of x ;
- c) $\cos(x + a) - \cos(a)$ for small values of x ;

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 $Rel(x_A)$:

- a) $\sin(0.521)$;
- b) $e^{3.22}$;
- c) $\sqrt{0.0011}$;
- d) $\arcsin(0.5)$.



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

$$x_{n+1} = \frac{1}{2} \left(x_n + \frac{a}{x_n} \right), \quad n \geq 0.$$

Derive the error and relative error formulas:

$$\begin{aligned} \sqrt{a} - x_{n+1} &= -\frac{1}{2x_n} (\sqrt{a} - x_n)^2, \\ \text{Rel}(x_{n+1}) &= -\frac{\sqrt{a}}{2x_n} (\text{Rel}(x_n))^2. \end{aligned}$$

For initial guess x_0 near \sqrt{a} , the last formula becomes

$$\text{Rel}(x_{n+1}) \approx -\frac{1}{2} (\text{Rel}(x_n))^2$$

Assuming $\text{Rel}(x_0) = 0.1$, use this formula to estimate the relative error in $x_i, i = 1, 2, 3, 4$.

Problem 2.5

Derive formula

$$\text{Rel}(x_{n+1}) = (\text{Rel}(x_n))^2$$

for the Newton's iterations used in computing $\frac{1}{b}$ for given b (formula was discussed in class without proof).

Problem 2.6

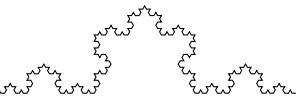
How many solutions are there to the equation $x = e^{-x}$? Will the iteration $x_{n+1} = e^{-x_n}$ converge for a suitable choice of x_0 ? Use Aitken extrapolation formula to estimate the error $\alpha - x_3$ for $x_0 = 0.57$.

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 ?

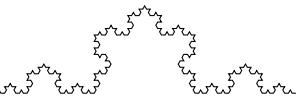
n	x_n	$x_{n-1} - x_n$
0	0.75	
1	0.752710	0.00271
2	0.754795	0.00208
3	0.756368	0.00157
4	0.757552	0.00118
5	0.758441	0.00089

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:

$$\begin{aligned} |\alpha - c_n| &\leq \frac{3}{2^n} \leq \varepsilon = 10^{-9} \\ \Leftrightarrow n &\geq \frac{\ln\left(\frac{3}{10^{-9}}\right)}{\ln 2} \approx 31.48. \end{aligned}$$

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

$$\begin{aligned} n &\geq \frac{\ln\left(\frac{1}{2^{-m}}\right)}{\ln 2} \\ &= \frac{\ln(2^m)}{\ln 2} \\ &= m, \end{aligned}$$

clearly the number of halvings will be m .

Problem 2.3

Solution, obviously is $\alpha = 0$. Newton's method is

$$\begin{aligned} x_{n+1} &= x_n - \frac{x_n^2}{2x_n} \\ &= \frac{1}{2}x_n = \frac{1}{2}\left(\frac{1}{2}x_{n-1}\right) = \frac{1}{2^2}\left(\frac{1}{2}x_{n-2}\right) \\ &= \dots \\ &= \frac{x_0}{2^{n+1}}, \end{aligned}$$

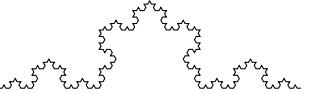
where x_0 is the initial guess. It can be seen that $x_{n+1} \rightarrow 0$ for any value of x_0 , in other words, the Newton's method will converge, no matter what initial guess is chosen. In order to see the speed of convergence, first we can observe a similar behavior to the behavior of the bisection 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

$$\begin{aligned} \sqrt{a} - x_{n+1} &= -\frac{(\sqrt{a} - x_n)^2}{2} \cdot \frac{f''(c_n)}{f'(x_n)} \\ &= -\frac{(\sqrt{a} - x_n)^2}{2} \cdot \frac{2}{2x_n} \\ &= -\frac{(\sqrt{a} - x_n)^2}{2x_n}. \\ \text{Rel}(x_{n+1}) &= \frac{\sqrt{a} - x_{n+1}}{\sqrt{a}} \\ &= \frac{-\frac{(\sqrt{a} - x_n)^2}{2x_n}}{\sqrt{a}} \\ &= -\frac{(\sqrt{a} - x_n)^2 \cdot \sqrt{a}}{2x_n (\sqrt{a})^2} \\ &= -\frac{\sqrt{a}}{2x_n} \left(\frac{\sqrt{a} - x_n}{\sqrt{a}} \right)^2 \\ &= -\frac{\sqrt{a}}{2x_n} (\text{Rel}(x_n))^2. \end{aligned}$$

For initial guess x_0 near \sqrt{a} , the method was shown to converge and therefore $x_n \rightarrow \sqrt{a}$. Thus for n big enough, $x_n \approx \sqrt{a}$ and the last formula becomes

$$\text{Rel}(x_{n+1}) \approx -\frac{1}{2} (\text{Rel}(x_n))^2$$

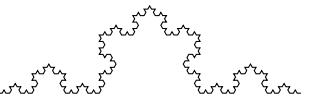
Let $\text{Rel}(x_0) = 0.1$, Then

$$\begin{aligned} \text{Rel}(x_1) &\approx -\frac{1}{2} (\text{Rel}(x_0))^2 = -0.005; \\ \text{Rel}(x_2) &\approx -\frac{1}{2} (\text{Rel}(x_1))^2 \approx -1.25 \cdot 10^{-5}; \\ \text{Rel}(x_3) &\approx -\frac{1}{2} (\text{Rel}(x_2))^2 \approx -7.8125 \cdot 10^{-11}; \\ \text{Rel}(x_4) &\approx -\frac{1}{2} (\text{Rel}(x_3))^2 \approx -3.0518 \cdot 10^{-21}. \end{aligned}$$

Problem 2.5

Let $f(x) = b - \frac{1}{x}$. Root is $\alpha = \frac{1}{b}$. The relative error according to its definition is

$$\begin{aligned} \text{Rel}(x_{n+1}) &= \frac{\alpha - x_{n+1}}{\alpha} \\ &= \frac{\frac{1}{b} - x_{n+1}}{\frac{1}{b}} \\ &= 1 - bx_{n+1}. \end{aligned}$$



In class we showed that Newton's method for the equation $b - \frac{1}{x} = 0$ becomes

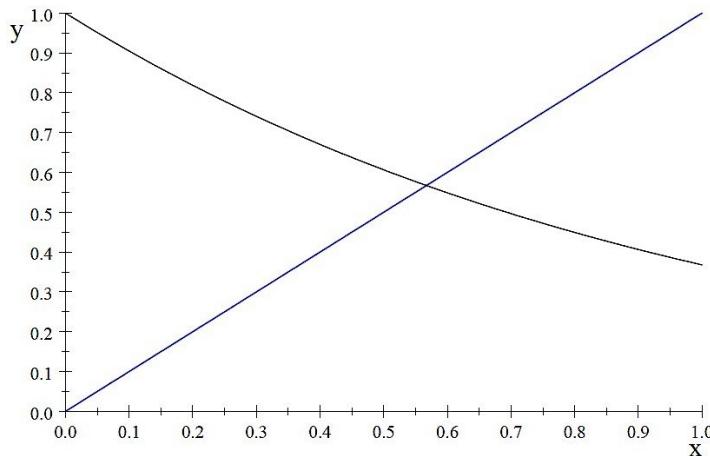
$$x_{n+1} = x_n(2 - bx_n).$$

Thus, combining the last two formulas we have

$$\begin{aligned}\text{Rel}(x_{n+1}) &= 1 - bx_{n+1} \\ &= 1 - bx_n(2 - bx_n) \\ &= 1 - 2bx_n + b^2x_n^2 \\ &= (1 - bx_n)^2 \\ &= (\text{Rel}(x_n))^2\end{aligned}$$

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

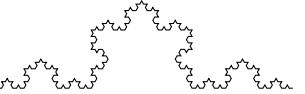
$$\lambda_3 = \frac{x_3 - x_2}{x_2 - x_1}.$$

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

$$|\text{Rel}(-1960.14)| = \frac{-1960 - (-1960.14)}{-1960} = 7.1429E - 5$$

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

$$\begin{aligned} g'(x) &= \frac{9x^4 - 30x^2 + 75}{(3x^2 + 5)^2} \\ g'(\sqrt{5}) &= 0. \end{aligned}$$

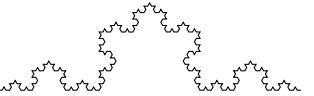
Thus, the order of convergence is at least quadratic.

$$\begin{aligned} g''(x) &= \frac{12x(-3x^4 + 31x^2 - 80)}{(3x^2 + 5)^4} \\ g''(\sqrt{5}) &= 0. \end{aligned}$$

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 $\lambda_n \rightarrow 0.75 = \frac{3}{4}$. 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 obviously are converging to 0.707, we have $g'(\alpha) \approx 0.707$ and

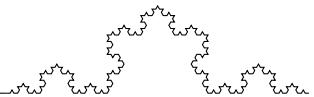
$$\alpha - x_{n+1} \approx 0.707(\alpha - x_{n+1})$$

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 :

$$F_0 = 1, \quad F_1 = 1, \quad F_n = F_{n-1} + F_{n-2}, \quad n = 2, 3, \dots$$

Let $R_n = \frac{F_{n+1}}{F_n}$. It can be shown that

$$\lim_{n \rightarrow \infty} R_n = \frac{1 + \sqrt{5}}{2} \equiv \phi,$$

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

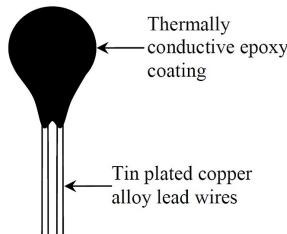


Figure 1: A typical thermistor

the resistance R of the thermistor and the temperature T is given by

$$\frac{1}{T} = 1.129241 \times 10^{-3} + 2.341077 \times 10^{-4} \log R + 8.775468 \times *10^{-8} (\log R)^3$$

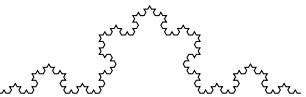
where T is in Kelvin and R is in Ohms, and \log denotes the natural logarithm. A thermistor error of no more than $\pm 0.01^\circ C$ is acceptable. To find the range of the resistance that is within this acceptable limit at $19^\circ C$, we need to solve

$$\frac{1}{19.01 + 273.15} = 1.129241 \times 10^{-3} + 2.341077 \times 10^{-4} \log R + 8.775468 \times *10^{-8} (\log R)^3 \quad (1)$$

$$\frac{1}{18.99 + 273.15} = 1.129241 \times 10^{-3} + 2.341077 \times 10^{-4} \log R + 8.775468 \times *10^{-8} (\log R)^3 \quad (2)$$

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 $f(x) = e^{x-\pi} + \cos x - x + \pi$.

- Plot its graph on interval $[0, 5]$.
- 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?
- Write a modified routine that will ensure **quadratic** convergence and apply it.
- 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

- Compute the fixed point $x_{n+1} = \cos x_n - 1 + x_n$ with initial guess $x_0 = 0.1$.
- What can be said about the speed of convergence? Compare it with bisection method.
- 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.

- What can be said about the order of convergence? Is it slower or faster than bisection method?
- What can be said about the root α to explain this convergence?
- Knowing function $f(x)$, how would you speed up the convergence?

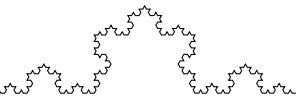
n	x_n	$x_n - x_{n-1}$
0	2.0	
1	2.1248	0.124834
2	2.2148	0.089944
3	2.2805	0.065698
4	2.3289	0.048386
5	2.3647	0.035827
6	2.3913	0.026624
7	2.4111	0.019835
8	2.4260	0.014803
9	2.4370	0.011062
10	2.4453	0.0082745

Problem 2.7

For solving the equation $x + \ln x = 0$, there were proposed three methods:

$$\begin{aligned}(a) \quad x &= -\ln x \\ (b) \quad x &= e^{-x} \\ (c) \quad x &= \frac{x + e^{-x}}{2}\end{aligned}$$

- Which of the formulas can be used?
- Which of the formulas should be used?
- 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. Benoit 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, \dots 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, $|z_n| \rightarrow \infty$ as $n \rightarrow \infty$).

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 .

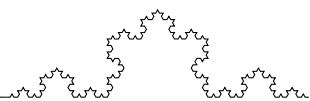
a) It has been shown that if $|z_n| > 2$ for some n , then it is guaranteed that the orbit will tend to infinity. The value of n for which this becomes true is called the **escape velocity** of a particular z_0 . Write a function that returns the escape velocity of a given z_0 and c . The function declaration should be: $n = \text{EscVel}(z_0, c, N)$, 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

$$M = \text{JuliaSet}(z_{Max}, c, N),$$

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 $c = -0.297491 + i * 0.641051$.

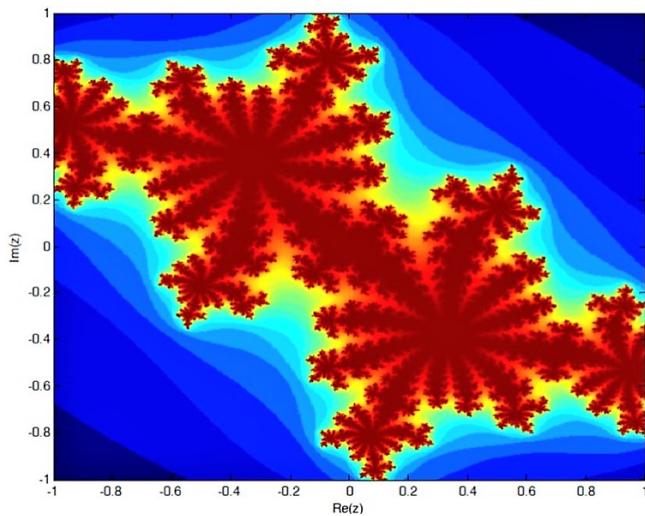


Figure 2: $M = \text{JuliaSet}(1, -0.297491 + i * 0.641051, 100)$

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.

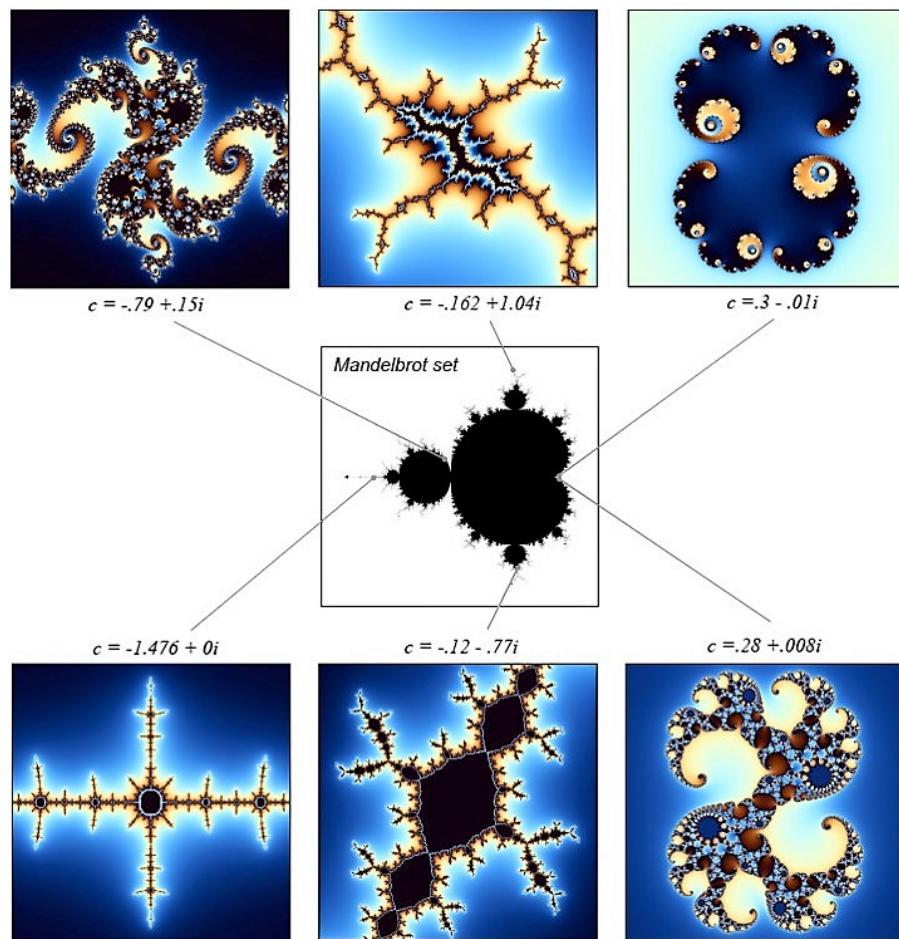
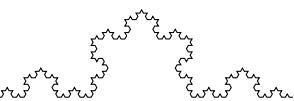
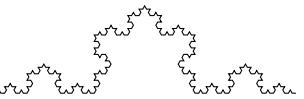


Figure 3: Different Julia Sets and their relation to Mandelbrot Set.



Practice Problems Set 3

Problem 3.1

Find a polynomial $P(x)$ of degree ≤ 3 for which

$$\begin{aligned} P(0) &= y_1 & P(1) &= y_2 \\ P'(0) &= y'_1 & P'(1) &= y'_2 \end{aligned}$$

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.

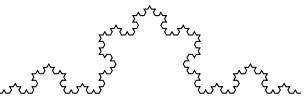
Problem 3.6

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 $T_0(x), T_1(x), T_2(x), \dots, T_n(x)$ for a particular value of x ?

Problem 3.11

Let $q(x)$ be a polynomial of degree $\leq n - 1$, and consider

$$\max_{-1 \leq x \leq 1} |x^n - q(x)|$$

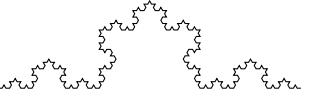
What is the smallest possible value for this quantity? Solve for the $q(x)$ for which this value is attained.

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

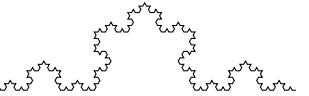
$$\begin{aligned} P(x) &= y_1 + y'_1 x + (-3y_1 + 3y_2 - 2y'_1 - y'_2)x^2 + (2y_1 - 2y_2 + y'_1 + y'_2)x^3 \\ &= (2x^3 - 3x^2 + 1)y_1 + (-2x^3 + 3x^2)y_2 + (x^3 - 2x^2 + x)y'_1 + (x^3 - x^2)y'_2 \\ &= (1+2x)(1-x)^2 y_1 + x^2(3-2x)y_2 + x(1-x)^2 y'_1 + x^2(x-1)y'_2 \\ &= H_1(x)y_1 + H_2(x)y_2 + H_3(x)y'_1 + H_4(x)y'_2 \end{aligned}$$

with

$$\begin{aligned} H_1(x) &= (1+2x)(1-x)^2, \\ H_2(x) &= x^2(3-2x), \\ H_3(x) &= x(1-x)^2, \\ H_4(x) &= x^2(x-1), \end{aligned}$$

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

$$\begin{cases} a + b \cos(\pi \cdot 0) + c \sin(\pi \cdot 0) = Q_1(0) = 2 \\ a + b \cos(\pi \cdot \frac{1}{2}) + c \sin(\pi \cdot \frac{1}{2}) = Q_1(\frac{1}{2}) = 5 \\ a + b \cos(\pi \cdot 1) + c \sin(\pi \cdot 1) = Q_1(1) = 4 \end{cases}$$

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 $P_2(x) = 2 + 6x - 8x(x - \frac{1}{2}) = -8x^2 + 10x + 2$.

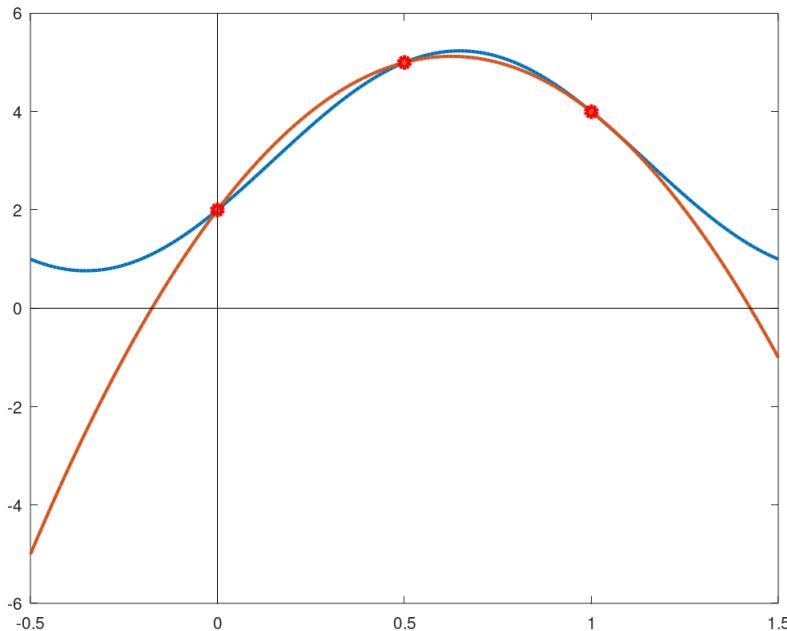


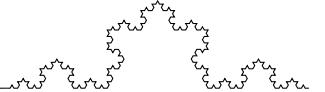
Figure 1: Graphs of quadratic interpolant $P_2(x) = -8x^2 + 10x + 2$ (red) and trigonometric interpolant $Q_1(x) = 3 - \cos(\pi x) + 2 \sin(\pi x)$ (blue).

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 $L_i(x)$, $i = 0, 1, \dots, N$ be $N + 1$ associated Lagrange basis functions. Need to prove that

$$\sum_{i=0}^N L_i(x) = 1. \quad (1)$$

Proof.

Rewrite the left side of the identity (1) in the form

$$1 \cdot L_0(x) + 1 \cdot L_1(x) + 1 \cdot L_2(x) + \cdots + 1 \cdot L_N(x) \quad (2)$$

and compare it with the formula for interpolation polynomial for the data (x_i, y_i) , $i = 0, 1, \dots, N$:

$$y_0 \cdot L_0(x) + y_1 \cdot L_1(x) + y_2 \cdot L_2(x) + \cdots + y_N \cdot L_N(x).$$

Obviously (2) represents the interpolation polynomial that interpolates points $\{(x_0, 1), (x_1, 1), (x_2, 1), \dots, (x_N, 1)\}$. These points lie on the line $y = 1$, therefore, 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

$$e^{-x^2} - P_2(x) = \frac{x(x - \frac{1}{2})(x - 1)}{6} \left(e^{-x^2} \right)^{'''}(θ) \quad (3)$$

for some $θ ∈ [0, 1]$. Let $h = \frac{1}{2}$. Using error formula (3) we get

$$\begin{aligned} |e^{-x^2} - P_2(x)| &= \frac{|x(x - h)(x - 2h)|}{6} \left| \left(e^{-x^2} \right)^{'''}(θ) \right| \\ &\leq \frac{1}{6} \cdot \max_{x \in [0, 1]} |x(x - h)(x - 2h)| \cdot \max_{x \in [0, 1]} \left| \left(e^{-x^2} \right)^{'''}(x) \right| \end{aligned} \quad (4)$$

From Lecture 9 (pages 8 – 10) we know that

$$\max_{x_0 \leq x \leq x_2} |(x - x_0)(x - x_1)(x - x_2)| = \frac{2h^3}{3\sqrt{3}}, \text{ with } h = x_1 - x_0 = x_2 - x_1.$$

Thus,

$$\max_{x \in [0, 1]} |x(x - h)(x - 2h)| = \frac{2 \cdot \frac{1}{2^3}}{3\sqrt{3}} = \frac{1}{12\sqrt{3}} \approx 0.048113 \quad (5)$$

Next compute the 3rd derivative of e^{-x^2} :

$$\begin{aligned} \left(e^{-x^2} \right)' &= -2xe^{-x^2} \\ \left(e^{-x^2} \right)'' &= (4x^2 - 2)e^{-x^2} \\ \left(e^{-x^2} \right)''' &= (-8x^3 + 12x)e^{-x^2} \end{aligned}$$

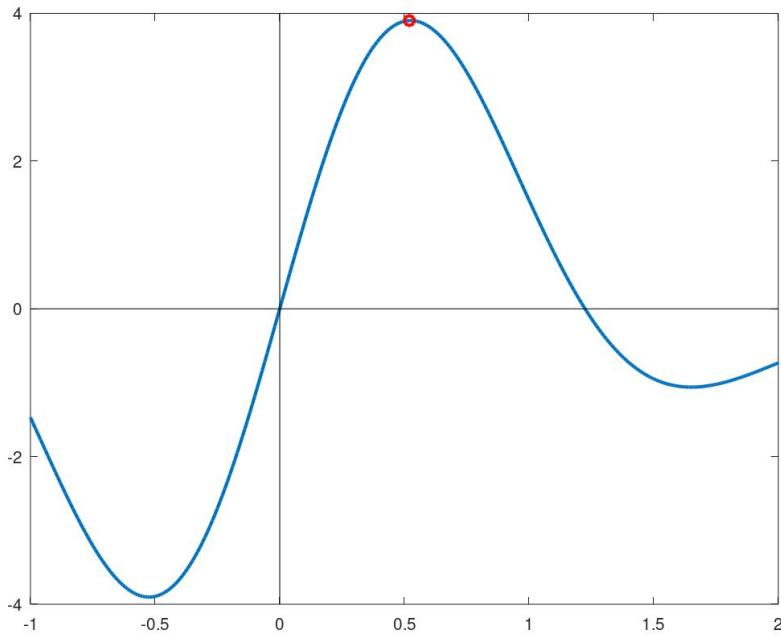
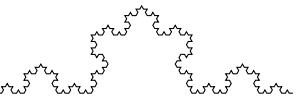


Figure 2: Graph of $(-8x^3 + 12x)e^{-x^2}$

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

$$\max_{x \in [0, 1]} \left| \left(e^{-x^2} \right)^{'''}(x) \right| \approx 3.9032 \quad \text{at } x \approx 0.52 \quad (6)$$

Substituting (6) and (5) in inequality (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

$$s(x) = \begin{cases} (x-1)^3, & 0 \leq x \leq 1 \\ 2(x-1)^3, & 1 \leq x \leq 2 \end{cases}$$

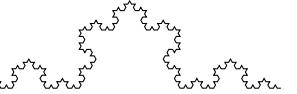
Obviously condition 1 is satisfied. Then

$$s'(x) = \begin{cases} 3(x-1)^2, & 0 \leq x \leq 1 \\ 6(x-1)^2, & 1 \leq x \leq 2 \end{cases}$$

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

$$P_{1,p}(x) = \begin{cases} \frac{1}{2}x, & 0 \leq x \leq \frac{1}{2} \\ \frac{3}{2}x - \frac{1}{2}, & \frac{1}{2} \leq x \leq 1 \\ -2x + 3, & 1 \leq x \leq 2 \\ -1, & 2 \leq x \leq 3 \end{cases}$$

(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:

$$P_{1,p}(x) = \begin{cases} x^2, & 0 \leq x \leq 1 \\ x^2 - 5x + 5, & 1 \leq x \leq 3 \end{cases}$$

(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.

$$sn(x) = \begin{cases} 1.80952x^3 + 0.04762x, & 0 \leq x \leq \frac{1}{2} \\ -5.04762(x - 0.5)^3 + 2.71429(x - 0.5)^2 + 1.40476(x - 0.5) + 0.25, & \frac{1}{2} \leq x \leq 1 \\ 2.52381(x - 1)^3 - 4.85714(x - 1)^2 + 0.33333(x - 1) + 1.0, & 1 \leq x \leq 2 \\ -0.90476(x - 2)^3 + 2.71429(x - 2)^2 - 1.80952(x - 2) - 1.0, & 2 \leq x \leq 3 \end{cases}$$

Let's plot all three interpolants:

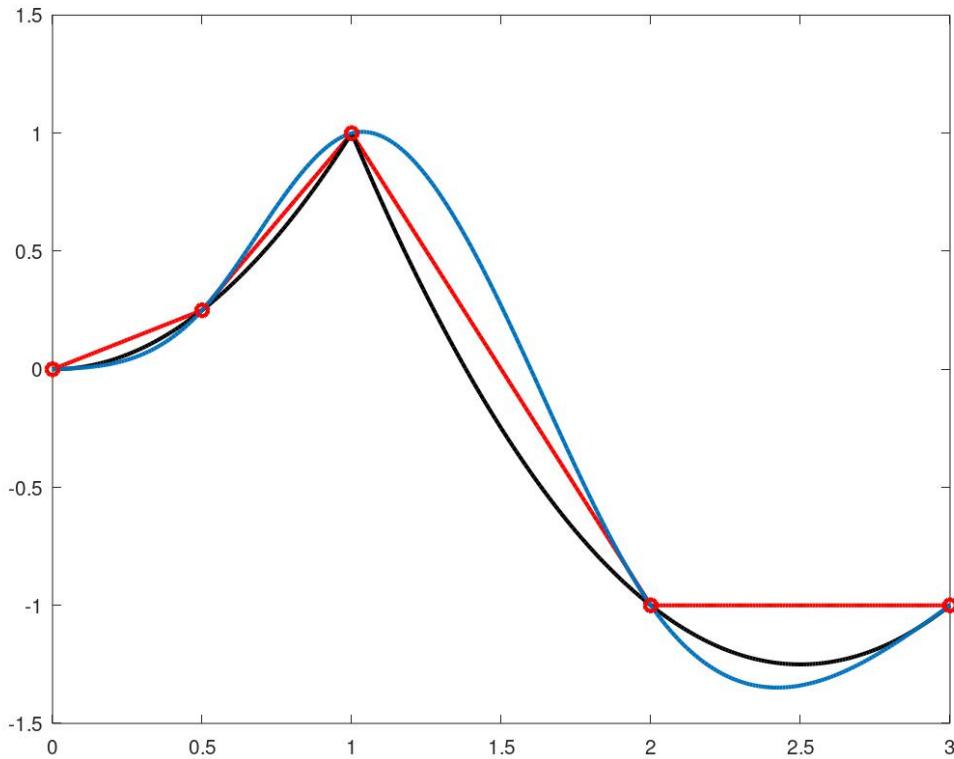
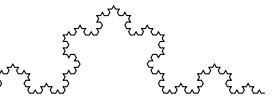


Figure 3: Graphs of piecewise linear interpolant (red), piecewise quadratic interpolant (black) and natural cubic spline interpolant (blue)

IMPORTANT REMARK:

Beware and read help/reference documentation initially, since, as it was mentioned in class lecture, there are various 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 (obtained using *spline* function).

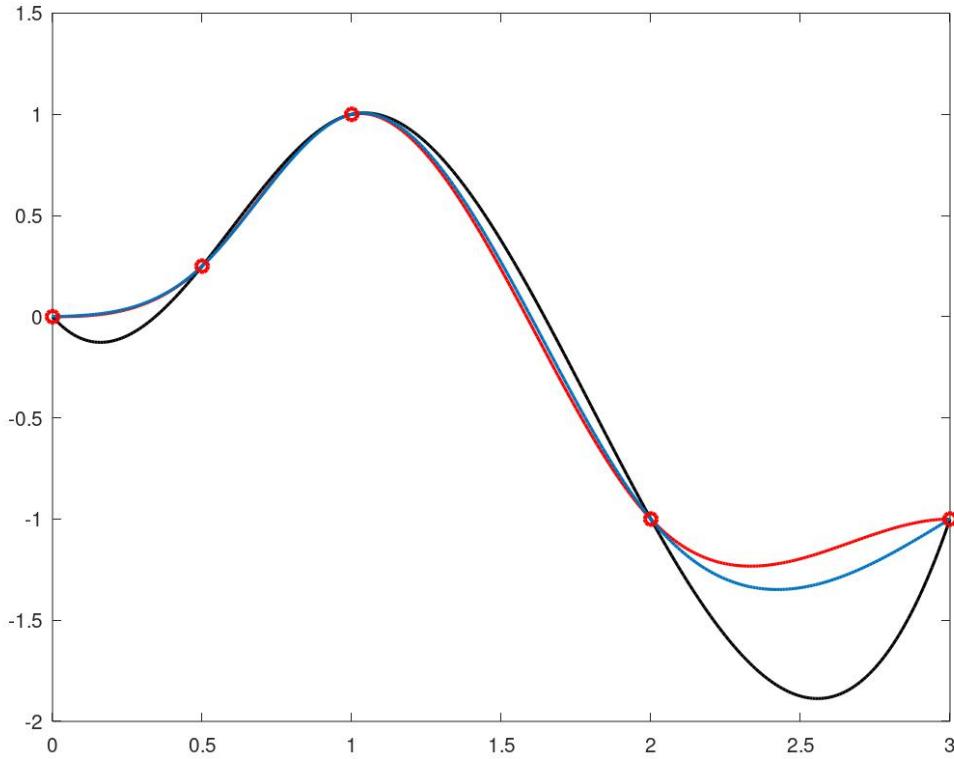
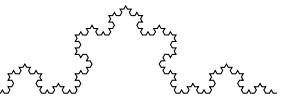


Figure 4: Natural cubic spline (blue), clamped cubic spline (red), not-a-knot cubic spline (black)

Problem 3.9

Consider the error formula for minimax approximation (see Lecture 10, page 5)

$$\rho_n(f) \leq \frac{\left(\frac{b-a}{2}\right)^{n+1}}{(n+1)! 2^n} \max_{x \in [a,b]} |f^{(n+1)}(x)|$$

Substituting $a = -1$, $b = 2$, $n = 5$ we get error estimate formula

$$\rho_5(f) \leq \frac{\left(\frac{3}{2}\right)^6}{6! 2^5} \max_{x \in [-1,2]} |f^{(6)}(x)| = \frac{3^6}{6! 2^{11}} \max_{x \in [-1,2]} |f^{(6)}(x)|,$$

where $f(x) = e^{3x-1}$. Compute derivatives

$$(e^{3x-1})' = 3e^{3x-1}, \quad (e^{3x-1})'' = 9e^{3x-1}, \quad \dots, \quad (e^{3x-1})^{(6)} = 3^6 e^{3x-1}$$

Since

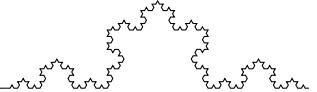
$$\max_{x \in [-1,2]} |f^{(6)}(x)| = 3^6 \max_{x \in [-1,2]} |e^{3x-1}| = 3^6 \cdot e^{3 \cdot 2 - 1} = 3^6 e^5$$

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 recursion formula for Chebyshev polynomials (see Lecture 10, 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 $\tilde{T}_n(x)$, and its maximum value on $[-1, 1]$ is $\frac{1}{2^{n-1}}$. Thus,

$$\max_{x \in [-1, 1]} |x^n - q(x)| = \frac{1}{2^{n-1}}$$

and it is achieved for $q(x) = x^n - \tilde{T}_n(x)$.

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

$$\begin{aligned} \int_{-1}^1 \frac{T_n(x)T_m(x)}{\sqrt{1 - x^2}} dx &= \int_{\pi}^0 \frac{\cos(n\theta)\cos(m\theta)(-\sin \theta) d\theta}{\sin \theta} \\ &= \int_0^\pi \cos(n\theta)\cos(m\theta) d\theta \\ &= 0, \text{ if } n \neq m. \end{aligned} \quad \blacksquare$$