A Primer On Numerical Mathematics

Dr. Achim Kehrein
Dr. Thorsten Camps
William L. Cole

13.08.2019

Contents

Chapter 0

Numerical Mathematics: A Paradigm Shift

> ”Because most things don’t work the same way on a computer - to a scary extent.”

0.1 A ”New” Approach to Math.

So far:
Infinite processes (limits, derivatives, integrals, etc.)
$\longmapsto$ Must truncate these processes to make them finite. (”Truncation Error”)

Infinitely many numbers, infinite precision:
$\longmapsto$ Computer has only finitely many machine numbers with finite precision (”round-off error”)

Error propagation in mathematical recipes: Mathematically equivalent formulas may produce different results on a computer. Which result is more accurate?

Goal of Numerics Course:
Learn to treat each problem individually instead of applying ”standard recipes”.

Chapter 1

Number Representation in Computers and Calculators

Which numbers does a computer use?

Two types:

- INT - Integer
- FLOAT, DOUBLE - Floating point numbers

In the past, calculators only used FLOATs. Modern Calculators use both INTs and FLOATs.

1.1 Integers

...-1, 0, 1, 2, 3...

Infinitely many in math, but computers can only represent finitely many.

INT_{MIN}, INT_{MIN}+1, ..., -1, 0, 1, ..., INT_{MAX}
Computers use a binary system instead of a decimal system of representation.

Decimal:
$$\begin{aligned}123 &= 1 \cdot 100 + 2 \cdot 10 + 3 \cdot 1\\ &= 1 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0\end{aligned}$$

Binary:
$${}_2 1101 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$$

$$= 1 \cdot 8 \ + 1 \cdot 4 \ + 0 \cdot 2 \ + 1 \cdot 1 \ = \ {}_{10}13$$

This scheme corresponds to:

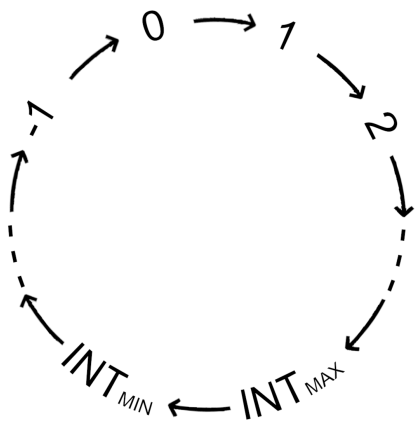


Figure 1.1: In a computer, integers are arranged like a clock.

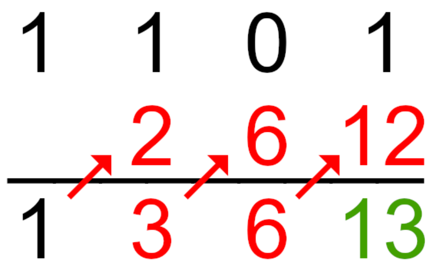
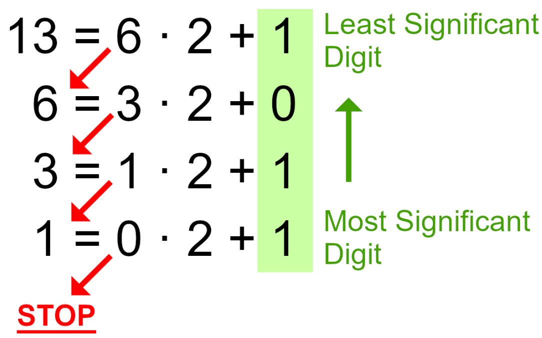


Figure 1.2: One method for binary to decimal conversion is to multiply by 2 and add the next digit.

$$
\begin{aligned}
&1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 \\
&\quad = (2^2 + 1 \cdot 2^1 + 0) \cdot 2 + 1 \\
&\quad = [(1 \cdot 2 + 1) \cdot 2 + 0)] \cdot 2 + 1
\end{aligned}
$$

How do we go in the other direction? How does one obtain the binary representation of a decimal integer? Use division with remainders.



For our calculators
$INT_{MAX} = 2147483647$

$INT_{MIN} = -2137483648$

$2147483647 = 1,073,741,824 \cdot 2 + 1$ [‡]

$1073741823 = 536870911 \cdot 2 + 1$

$$INT_{MAX} = \underbrace{{}_b 111................11}_{\text{31 digits. 1 bit for the sign}}$$

If we leave the range INT_{MAX}, INT_{MIN}, this is called an **overflow error**. *The computer does not notify us about this!* We have to program all of the necessary checks.

> As long as the computations do not produce an overflow error, arithmetic with the datatype INT is exact: Addition, subtraction, multiplication, and division with remainders.

We can extend this to rational numbers:

$$Rational\ Number = \frac{Integer}{Non{-}Zero\ Integer}$$

Just store numerator and denominator separately as integers (INTs).
Arithmetic for rational numbers:

$$\frac{n_1}{d_1} + \frac{n_2}{d_2} = \frac{n_1 d_2 + n_2 d_1}{d_1 d_2}$$

To avoid unnecessary overflow, cancel the fractions as much and as soon as possible when programming your own routine. For a better interpretation, we can convert $\frac{numerator}{denominator}$ into a decimal representation *at the end* of our computations.

Example: $\frac{13}{12}$

$$
\begin{array}{l|l}
1\,3 & 1\,2 \\ \cline{2-2}
\quad 1\,0\,0 & 1.0\,8\,3\,3\,3 \\
\qquad\quad 4\,0 & \\
\qquad\qquad 4\,0 & \\
\qquad\qquad\quad 4\,0 & \\
\qquad\qquad\qquad 4 &
\end{array}
$$

The decimal representation will either terminate or will eventually become periodic, as in the preceding example.

[‡] The last digit of this number should be 3, but your calculator will round it.

1.2 Floating Point Numbers

Now let's look at FLOATs. In the sciences and in engineering, we also need very large numbers and very small numbers, i.e close to zero.

Avogadro's Number	6.022×10^{23}
Electrical Charge of an Elimentary Particle	$q_e = 1.602 \times 10^{-19} C$
Planck's Constant	$h = 6.626 \times 10^{-34} m^2 kg/s$

Another number representation is needed. Every real number possesses a decimal representation:

$$\pm d_k d_{k-1} ... d_0 \underbrace{.}_{\text{decimal}} d_{-1} d_{-2} .... \leftarrow \textit{May never stop or become periodic}$$

This is a convergent series:

$$= d_k \cdot 10^k + d_{k-1} \cdot 10^{k-1} + ... + d_0 \cdot 10^0 + d_{-1} \cdot 10^{-1} + d_{-2} \cdot 10^{-2} + ...$$

On a computer we can only use finitely many digits. The number of digits is often called the **precision**. By moving the decimal point, we standardize these numbers:

$$\boxed{\begin{array}{c} \pm \underbrace{d_b . d_{-1} d_{-2} ... d_{-p+1}}_{\text{significand}} \times 10^{k \leftarrow Exponent}_{\nwarrow Base} \\ \text{This is a } \textbf{floating point number}. \end{array}}$$

It is **normalized** when there is only one digit, d_0 before the decimal point. In other sources, "normalized" is often shown as

$$\mathbf{0}.d_k d_{k-1} ... d_{-3} * 10^{k+1} + \Delta x$$

For our purposes, a number is normalized when $d_0 \neq 0$, for two main reasons:

- This is how FLOATs are stored on computers and calculators.
- Scientific Notation (i.e. Avogadro's Constant = 6.022×10^{23})

Also, a computer doesn't use the decimal system. It uses **binary**. In binary

- base $= 2$
- digits: 0, 1

In base 2 a normalized floating-point number is

$$\mathbf{1}.b_{-1} b_{-2} ... b_{-p+1} * 2^E$$

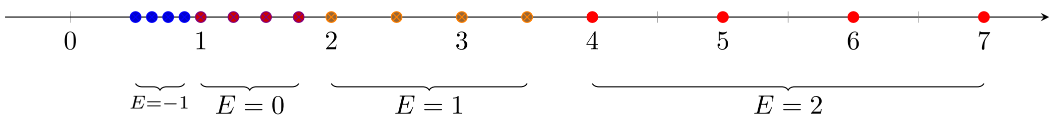
Because the first digit is automatically 1, it is usually not stored in the computer.

1.3 Relative Error

To get a feeling for floating point numbers, let's study a very limited "Toy" number system: $1.b_{-1}b_{-2} \cdot 2^E$

b_{-1}	b_{-2}	E	Decimal Value
0	0	0	$1 + 0 \cdot \frac{1}{2} + 0 \cdot \frac{1}{4} = 1$
0	1	0	$1 + 0 \cdot \frac{1}{2} + 1 \cdot \frac{1}{4} = 1.25$
1	0	0	$1 + 1 \cdot \frac{1}{2} + 0 \cdot \frac{1}{4} = 1.5$
1	1	0	$1 + 1 \cdot \frac{1}{2} + 1 \cdot \frac{1}{4} = 1.75$
0	0	1	$1 \cdot 2^1 = 2$
0	1	1	$1.25 \cdot 2^1 = 2.5$
1	0	1	$1.5 \cdot 2^1 = 3$
1	1	1	$1.75 \cdot 2^1 = 3.5$

1.	b_{-1}	b_{-2}	$E = -1$	$E = 0$	$E = 1$	$E = 2$
	0	0	0.5	1	2	4
	0	1	0.625	1.25	2.5	5
	1	0	0.75	1.5	3	6
	1	1	0.875	1.75	3.5	7



Note that the distance between the numbers depends on their size. Consequentially, the round off error will also be affected by the size of the number. For this reason it's important to use *relative error* instead of absolute error.

Absolute Error	$x + \Delta x$
Relative Error	$x(1 + \frac{\Delta x}{x}) = x(1 + \delta x)$

Often the relative error is the better way to measure the round off error of floating point numbers.

$$fl(x) = x(1 + \delta x) \Leftrightarrow \delta x = \frac{fl(x) - x}{x}$$

1.4 Relative Error Examples

Example 1: Real number, 4 digits, base 10:

$$x = \underbrace{1.234}_{4digits}56$$

$$fl(x) = 1.235$$

$$\delta x = \frac{1.235 - 1.23456}{1.23456}$$

$$= \frac{0.00044}{1.23456} = 0.000356$$

$$\simeq 0.0004 \leftarrow \text{Only first non-zero digit}$$

For errors, only record record the first non-zero (in most cases). When the first non-zero digit is 1, record the next digit (which can be zero).

Example 2: $y = 123.456 = 1.23456 \times 10^2$

$$fl(y) = 1.235 \times 10^2$$

$$\delta y = \frac{1.235 \times 10^2 - 1.23456 \times 10^2}{1.23456 \times 10^2}$$

$$= \frac{0.00044 \times 10^2}{1.23456 \times 10^2} \leftarrow \textit{Absolute Error}$$

$$\simeq 0.0004 \leftarrow \textit{Relative Error}$$

Notice how the relative error size reflects the number of digits used in the system. If a decimal system uses k digits, then the relative round-off error is bounded by 5×10^{-k}.

1.5 Subnormal Numbers and other special FLOAT values

Note in the number line from earlier that there is a gap at zero. What do we do about this gap? How do we express zero itself as a FLOAT? Not only is the number of digits in the mantissa finite, but the number of digits in the exponent is also finite. In C and other programming languages FLOATs are 32 Bits. 23 of these bits are used for the significand, 7 for the exponent, and 1 for the sign.

In our toy system from earlier, let's use $-2 \leq E \leq 4$

Use the maximal exponent for

INF	infinity (overflow error)
NaN	"not a number"

These two are distinguished by using different mantissae.

Use the minimal exponent for ***subnormal numbers:*** $\mathbf{0}.b_{-1}b_{-2} \times 2^{-2}$

0.00×2^{-2}	0
0.01×2^{-2}	$\frac{1}{8}$
0.10×2^{-2}	$\frac{1}{4}$
0.11×2^{-2}	$\frac{3}{8}$

What is the relative error in our toy system?

$$\tfrac{1}{8} = 2^{-3} = \tfrac{1}{2} \times 2^{-2}$$

Go to seven, which in binary is

$$1.11 \times 2^2$$
$$\tfrac{1}{8} \times 2^2$$

Some example of relative errors in our binary toy system:

$$\tfrac{7-7.4}{7.4} \simeq -0.05$$

$$\tfrac{7-6.6}{6.6} \simeq 0.06$$

$$\tfrac{8-8.9}{8.9} \simeq -0.10$$

Why are subnormal numbers treated differently?

$$\frac{fl(\frac{1}{16})-\frac{1}{16}}{\frac{1}{16}} = \frac{\frac{1}{8}-\frac{1}{16}}{\frac{1}{16}} = \frac{\frac{2}{16}-\frac{1}{16}}{\frac{1}{16}} = \frac{\frac{1}{16}}{\frac{1}{16}} = 1$$

Relative error 2^{-3} does not hold for sub-normal numbers, only normalized numbers. Because of this, it is of the utmost importance to ***avoid dividing by numbers close to zero whenever possible!***

1.6 Adding FLOATs

Now we know the general shape of the machine number system FLOAT, how does arithmetic work with it?

$$\text{Toy System: } \left(\left(\left(\left(4 \oplus \tfrac{1}{4}\right) \oplus \tfrac{1}{4}\right) \oplus \tfrac{1}{4}\right) \oplus \tfrac{1}{4}\right) = 4$$

The sum of two machine numbers may not be a machine number. Addition my create more round- off errors.

$$4 \oplus \left(\tfrac{1}{4} \oplus \left(\tfrac{1}{4} \oplus \left(\tfrac{1}{4} \oplus \tfrac{1}{4}\right)\right)\right)$$

$$= 4 \oplus \left(\tfrac{1}{4} \oplus \left(\tfrac{1}{4} \oplus \tfrac{1}{2}\right)\right)$$

$$= 4 \oplus \left(\tfrac{1}{4} \oplus \tfrac{3}{4}\right) = 4 + 1 = 5$$

If you must add lots of numbers on a computer, add them in order from small to large, to minimize round-off error.

Example: Harmonic Series

$$1 + \tfrac{1}{2} + \tfrac{1}{3} + \tfrac{1}{4} + \tfrac{1}{5} + \tfrac{1}{6} + ...$$

- Divergent Series

- Partial sums "approach" infinity.

We try to recreate the harmonic series in a C program:

```
float sum;
sum = 0.0;
for (i=1; i<= 1 000 000; i++){
    sum += 1/(float) i;
}
```

We repeat this same for-loop, increasing the size of the loop from `i<=1 000 000` to 10,000,000 and 100,000,000. The program returns the following results:

```
1 000 000 summands give sum 14.3573579788
10 000 000 summands give sum 15.4036827087
100 000 000 summands give sum 15.4036827087
```

We see that by adding increasingly smaller summands the computer has already run out of precision before 10,000,000 cycles. If we reverse the order of the for-loop so that the script adds summands in ascending order from smallest to largest we obtain the following results:

```
1 000 000 summands give sum 14.3926515579
10 000 000 summands give sum 16.6860313416
100 000 000 summands give sum 18.8079185486
1 000 000 000 summands give sum 18.8079185486
```

By adding the summands from smallest to largest, it takes the script is able to add smaller numbers, and so obtain a larger partial sum. *When programming, if possible try to add smaller numbers first, for more accurate results.*

$$0.1 = \underbrace{b_{-1}}_{=0} \tfrac{1}{2} + b_{-2}\tfrac{1}{2^2} + b_{-3}\tfrac{1}{2^3} + ...$$

Multiply by 2:

$$0.2 = \underbrace{b_{-1}}_{=0} + b_{-2}\tfrac{1}{2} + b_{-3}\tfrac{1}{2^2} + ...$$

$\times 2$:

$$0.4 = b_{-2} + b_{-3}\tfrac{1}{2} + b_{-4}\tfrac{1}{2^2} + ...$$

$\times 2$:

$$0.8 = b_{-3} + b_{-4}\tfrac{1}{2}$$

A particular C program adds `float 0.0 += 0.1`, with the following results:

$$\begin{aligned}
fl(0.1) &= 0\,.\,100\,000\,001\,5\\
fl(0.2) &= 0\,.\,200\,000\,003\,0\\
fl(0.3) &= 0\,.\,300\,000\,011\,9\\
fl(0.4) &= 0\,.\,400\,000\,006\,0\\
fl(0.5) &= 0\,.\,500\,000\,000\,0\\
fl(0.6) &= 0\,.\,600\,000\,023\,8\\
fl(0.7) &= 0\,.\,700\,000\,047\,7\\
fl(0.8) &= 0\,.\,800\,000\,071\,5\\
fl(0.9) &= 0\,.\,900\,000\,095\,4\\
fl(1.0) &= 0\,.\,200\,000\,119\,2
\end{aligned}$$

$$\begin{array}{r|c|l}
b_0 & \mathbf{0} & \mathbf{.1} \leftarrow \textit{Double the fractional part}\\
b_{-1} & 0 & .2\\
b_{-2} & 0 & .4\\
b_{-3} & 0 & .8\\
\textit{Discard whole part} \rightarrow & 1 & .6\\
b_{-5} & 1 & .2 \leftarrow \textit{Numbers repeat}\\
b_{-6} & 0 & .4
\end{array}$$

The whole part becomes the binary representation, hence:

$$\begin{aligned}
fl({}_{10}0.1) &= \underbrace{{}_{2}0\,.}_{b_0}000\,110\,011\,001\,1... =_2 0.000\,\overline{1100}\\
&= \underbrace{{}_{2}1.\,1}_{\text{Normalized}}001\ 1001\ 1001... \times 2^{-4}
\end{aligned}$$

FLOATs use 23 fractional bits, so this non-terminating number will be rounded:

$$fl({}_{10}0.1) = 1.1001\ 1001\ 1001\ 1001\ 1001\ 10\cancel{01}^{1} \times 2^{-4}$$

Let's compute the round-off error:

$$\begin{aligned}
\text{absolute error}\rightarrow \quad \Delta x &= 1 \times 2^{-23} \times 2^{-4} - 1.1001\ 1001\ ... \times 2^{-24} \times 2^{-24}\\
&= 2^{-3} \times 2^{-24} -_{10} 0.1 \times 2^{-24}\\
&= ({}_{10}0.125 -_{10} 0.1) \times 2^{-24}\\
&=_{10} 1.49 \times 10^{-9} \simeq 1.5 \times 10^{-9}
\end{aligned}$$

Now let's add two FLOAT values:

$$
\begin{array}{rrcccccccc}
 & {\scriptstyle 1\,1} & {\scriptstyle 1\,1} & {\scriptstyle 1\,1} & {\scriptstyle 1\,1} & {\scriptstyle 1\,1} & {\scriptstyle 1\,1} & {\scriptstyle 1} & \\
fl(0.1) = & 1. & 1001 & 1001 & 1001 & 1001 & 1001 & 101 & \times 2^{-4} \\
+\, fl(0.1) = & +1. & 1001 & 1001 & 1001 & 1001 & 1001 & 101 & \times 2^{-4} \\
\hline
\text{Not normalized} \rightarrow & 11. & 0011 & 0011 & 0011 & 0011 & 0011 & 010 & \times 2^{-4}
\end{array}
$$

$$fl(0.2) = 1.\ 1001\ 1001\ 1001\ 1001\ 1001\ 101\overset{round-off}{\cancel{0}} \times 2^{-3}$$

To add an additional $fl(0.1) = 1.100... \times 2^{-4}$ we have to shift the mantissa to adjust the exponent.

$$
\begin{array}{rrcccccccc}
 & {\scriptstyle 1\,1} & {\scriptstyle 1\,1} & {\scriptstyle 1\,1} & {\scriptstyle 1\,1} & {\scriptstyle 1\,1} & {\scriptstyle 1\,1} & {\scriptstyle 1\,1} & \\
fl(0.2) = & 1. & 1001 & 1001 & 1001 & 1001 & 1001 & 101 & \times 2^{-3} \\
+\, fl(0.1) = & +\,0. & 1100 & 1100 & 1100 & 1100 & 1100 & 11\overset{1}{\cancel{0}}\cancel{1} & \times 2^{-3} \\
\hline
\text{Normalize} \rightarrow & 10. & 0110 & 0110 & 0110 & 0110 & 0110 & 100 & \times 2^{-3} \\
 & & & & & & & & \\
\Rightarrow & 1. & 0011 & 0011 & 0011 & 0011 & 0011 & 010 & \times 2^{-2}
\end{array}
$$

Is this floating point number representative of ${}_{10}0.3$? Let's check!

$$
\begin{array}{r|l}
\mathbf{0} & \mathbf{.3} \\
0 & .6 \\
1 & .2 \\
0 & .4 \\
0 & .8 \\
1 & .6 \leftarrow \textit{Numbers repeat} \\
\textit{Repeating pattern}: 0011 \rightarrow 1 & .2
\end{array}
$$

To add $fl(0.1)$ at this point, we must adjust the exponent by two. There are two options: First, we can shift by one digit twice, which would lead to two cases of rounding up. The second option is to shift by two digits at once, which leads to rounding down. Looking at the results from the computer, we see that the second method is chosen.

$$
\begin{array}{rcccccccc}
 & {\scriptstyle 1\,1} & {\scriptstyle 1\,1} & {\scriptstyle 1\,1} & {\scriptstyle 1\,1} & {\scriptstyle 1\,1} & {\scriptstyle 1} & \\
1. & 0011 & 0011 & 0011 & 0011 & 0011 & 010 & \times 2^{-2} \\
+\,0. & 0110 & 0110 & 0110 & 0110 & 0110 & 011 & \times 2^{-2} \\
\hline
1. & 1001 & 1001 & 1001 & 1001 & 1001 & 101 & \times 2^{-2} \\
 & & & & & & & = fl(0.4)
\end{array}
$$

$$
\begin{array}{rrcccccccc}
 & {\scriptstyle 1\,1} & {\scriptstyle 1\,1\,1\,1} & {\scriptstyle 1\,1\,1\,1} & {\scriptstyle 1\,1\,1\,1} & {\scriptstyle 1\,1\,1\,1} & {\scriptstyle 1\,1\,1\,1} & {\scriptstyle 1\,1} & \\
fl(0.4) = & 1. & 1001 & 1001 & 1001 & 1001 & 1001 & 101 & \times 2^{-2} \\
+\, fl(0.1) = & +\,0. & 0110 & 0110 & 0110 & 0110 & 0110 & 011 & \times 2^{-2} \\
\hline
\text{Normalize} \rightarrow & 10. & 0000 & 0000 & 0000 & 0000 & 0000 & 000 & \times 2^{-2} \\
 & & & & & & & & \\
fl(0.5) = & 1. & 0000 & 0000 & 0000 & 0000 & 0000 & 000 & \times 2^{-1}
\end{array}
$$

Note: In this program $fl(0.5) = 0.5$

<u>Side Remark:</u> When programming FLOAT results, allow for round-off errors. Don't use conditions like

```
result == 0.0
```

Instead use

```
result <= 0.00....1
```

Don't use values that are <u>too</u> small, though.

One more step:

$$\begin{array}{rllllllll} fl(0.5) = & 1. & 0000 & 0000 & 0000 & 0000 & 0000 & 000 & \times 2^{-1} \\ +\, fl(0.1) = & +\,0. & 0011 & 0011 & 0011 & 0011 & 0011 & \underbrace{010}_{\text{two digits affected}} & \times 2^{-1} \\ \hline & 1. & 0011 & 0011 & 0011 & 0011 & 0011 & 010 & \times 2^{-1} \end{array}$$

Because $fl(0.1)$ needs to be rounded by two digits, there is a larger error in this operation. **Shifting loses significant digits!** Many other math texts ignore this effect.

We may think that the round-off error is gone at 0.5 because it can be represented *exactly* by a machine number, but notice that 1 is also a machine number, and we have a huge round-off error there.

So far we've had

$$fl(0.x) + fl(0.1) = fl(0.x + 1)$$

$$i.e. \quad fl(0.4) + fl(0.1) = fl(0.5)$$

Note that:

$$\underbrace{fl(0.9) + fl(0.1)}_{\text{severe round-off error}} \neq fl(1.0)$$

$$Since \quad fl(1.0) = 1.0.$$

Where does this happen the first time and why? Notice in the preceding example we do not have a simple case of error propagation. The roundoff error of $fl(0.1)$ does not simply accumulate in the sum.

What errors are produced by arithmetic operations? Sometimes a sum, difference, product, or quotient of machine numbers is not a machine number. So there are round-off errors produced by arithmetic, not just number representation. (Other texts provide rather limited information about this.) Additionally, how do errors propagate if we begin with uncertain numbers (i.e. measurement data).

1.7 Finding Zeros of Functions

How do we compute function values? Due to round-off errors, can we really detect zeros? If a function gets very close to zero, how can we determine whether its value is really different from zero, or just a zero with a round-off error?

Example:

In our toy system with normalized floating-point numbers, of the form $b_0.b_{-1}b_{-2} \times 2^E$ we found:

$$4 \oplus \tfrac{1}{4} = 4$$

Why?

$$fl(4) =_2 1.00 \times 2^2$$
$$fl(\tfrac{1}{4}) =_2 1.00 \times 2^{-2}$$

Add them:

Shift the $fl(\frac{1}{4})$ to adjust the exponent:

$$\begin{array}{rcll} fl_{shifted}(\frac{1}{4}) & = & 0.00\,\cancel{01}^{\text{round down}} & \times 2^2 \\ +\ fl(4) & = & 1.00 & \times 2^2 \\ \hline & & 1.00 & \times 2^2 \ = fl(4) \end{array}$$

What size of error do we get here?

$$\delta_{sum} = \frac{4-4.25}{4.25} = \frac{-0.25}{4.25} \simeq -0.06$$

Upper bound for relative error: ${}_20.001 =_{10} 0.125$

Notice, however, that we can make the error arbitrarily large by repeated addition:

$$\left(\left(\left(4 \oplus \tfrac{1}{4}\right) \oplus \tfrac{1}{4}\right) \oplus \tfrac{1}{4}\right) \oplus \tfrac{1}{4}$$
$$\delta = \frac{4-5}{5} = \frac{-1}{5} = -0.2$$

We have already seen that adding smaller summands first improves the accuracy.

1.8 FLOAT multiplication

Example: Use decimal system with three digits:

$$
\begin{aligned}
&(1.23 \times 10^4) \cdot (4.56 \times 10^3) \\
&= 5.6088 \times 10^7 \\
&\Rightarrow 5.61 \times 10^7
\end{aligned}
$$

Rounding is necessary, but within the rounding bounds of the number system. For multiplication, the error propagation can be described rather easily:

$$
\begin{aligned}
&[x(1+\delta x)] \cdot [y(1+\delta y)] \\
&= (x + x \cdot \delta x)(y + y \cdot \delta y) \\
&= xy + xy\delta x + xy\delta y + xy\delta x\delta y \\
&= xy(1 + \delta x + \delta y + \underbrace{\delta x\delta y}_{negligible}) \\
&\simeq xy(1 + \delta x + \delta y)
\end{aligned}
$$

The relative error of the product is (to a very good approximation) the sum of the relative errors of the factors. The relative errors don't explode. They grow slowly.

1.9 FLOAT Division

Example: Use decimal system with three digits:

$$
\begin{aligned}
\frac{1.23\times 10^4}{4.56\times 10^3} &= 0.2697 \times 10^1 \\
&= 2.697 \times 10^0 \leftarrow \textit{Write exponent so it isn't forgotten!} \\
&\Rightarrow 2.70 \times 10^0 \\
\frac{x(1+\delta x)}{y(1+\delta y)} &= \frac{x}{y}\ (1+\delta x)(\underbrace{1 - \delta y + (\delta y)^2 - (\delta y)^3 + - ...}_{\text{Geometric series. } \delta y \text{ can be assumed } <1})
\end{aligned}
$$

Geometric Series:

$$1 + (-\delta y) + (-\delta y)^2 + (-\delta y)^3 + ... = \frac{1}{1-(-\delta y)}$$

$$\text{for } |\delta y| < 1$$

$$
\begin{aligned}
\frac{x(1+\delta x)}{y(1+\delta y)} &= \frac{x}{y}\ (1 + \delta x - \delta y + \textit{[smaller terms]}) \\
&\simeq \frac{x}{y}\ 1 + \delta x - \delta y
\end{aligned}
$$

As with multiplication, the relative errors in division can add up but can also partially cancel each other out. *Multiplication and Division behave nicely* for FLOATS.

1.10 FLOAT Subtraction

Subtraction is very problematic when values are *almost* equal.

$$4.56 \times 10^3 - 4.55 \times 10^3 = \underbrace{0.01}_{\text{Two significant digits lost!}} \times 10^3$$

Normalize:

$$= 1\underbrace{.00}_{\text{These could be 9. We don't know!}} \times 10^1$$

Such results are usually *very* uncertain. *Avoid formulas that lead to differences of almost equal numbers!*

Example 1:

$$\begin{aligned} \ln 5000 - \ln 4999 &\simeq 8.51719391 - 8.516993171 \\ &\simeq 2.0002 \times 10^{-4} \end{aligned}$$

Alternatively:

$$\ln \frac{5000}{4999} \simeq 2.000199947 \times 10^{-4}$$

Example 2:

$$f(t) = \frac{\sqrt{t^2+9}-3}{t^2}$$

This equation can be rewritten:

$$= \frac{\sqrt{t^2+9}-3}{t^2} \cdot \frac{\sqrt{t^2+9}+3}{\sqrt{t^2+9}+3}$$

$$= \frac{t^2+9-9}{t^2(\sqrt{t^2+9}+3)}$$

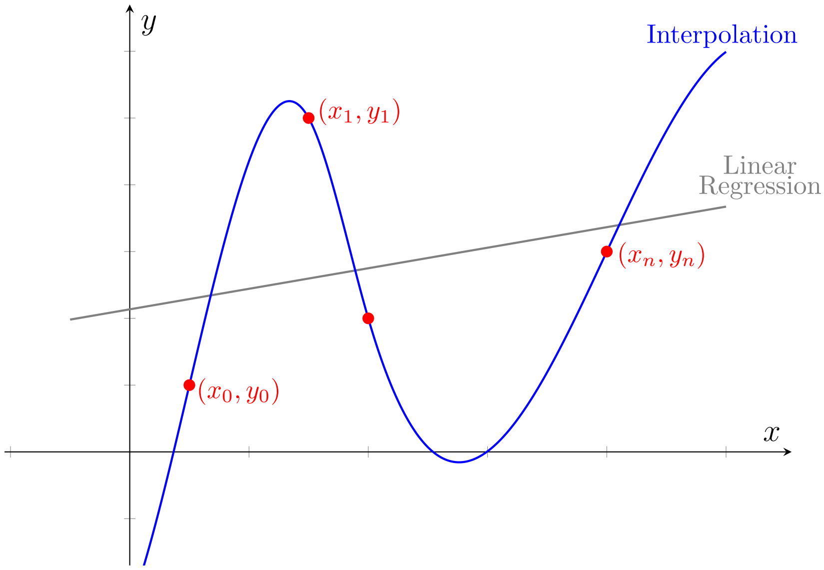
$$= \frac{1}{\sqrt{t^2+9}+3}$$

Computer results:

t	$\frac{\sqrt{t^2+9}-3}{t^2}$	$\frac{1}{\sqrt{t^2+9}+3}$
0.001	1.6667	1.666 6666 2
0.00001	1.6666	

Chapter 2

Interpolation



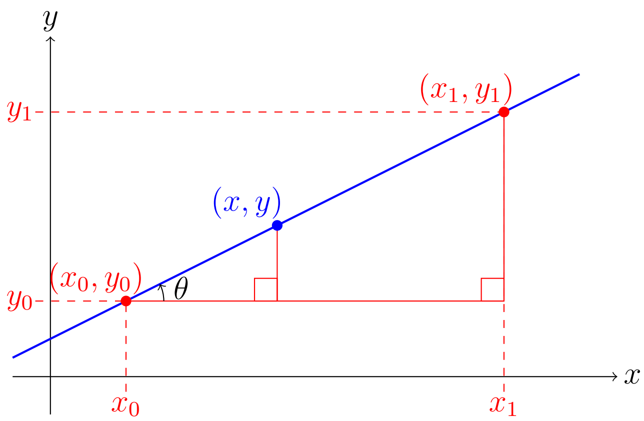
n+1 data points: $(x_0, y_0), (x_1, y_1), ...(x_n, y_n)$. We want an interpolating function, $y = f(x)$ such that $f(x_i) = y_i$ for all $i = 0, 1, ..., n$ Interpolation Conditions.

A totally different approach is the one using the "least squares" method - fit the function as best as possible to the data.

We start with polynomial interpolation, because polynomials are very simple functions with lots of nice properties:

- Can be easily calculated
- Continuous and Differentiable everywhere
- Versatile

Example: Linear interpolation (through two points)



Similar Triangles — Two Point Formula for a line

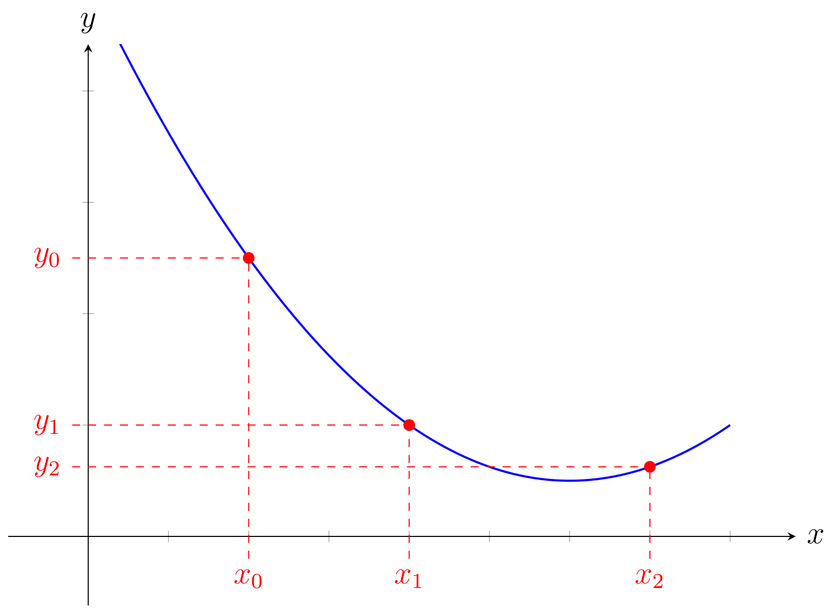
$$\frac{y-y_0}{x-x_0} = \underbrace{\frac{y_1 - y_0}{x_1 - x_0}}_{\text{slope}} \quad \Leftrightarrow \quad y = \frac{y_1-y_0}{x_1-x_0}(x - x_0) + y_0$$

We distinguish between *Interpolation* and*Extrapolation*.

Interpolation: Calculating points *between* smallest and largest x_i values.
Extrapolation: Calculating point *outside* range of x_i values.

2.1 Lagrangian Interpolation

Example: Three Data Points $(x_0, y_0)(x_1, y_1)(x_2, y_2)$



Model: $\quad p(x) = A + Bx + Cx^2$

$\uparrow \quad \uparrow \quad \uparrow$

parameters

How do we compute these parameters? Interpolating conditions lead to:

$$\begin{aligned} y_0 &= p(x_0) = A + Bx_0 + Cx_0^2 \\ y_1 &= p(x_1) = A + Bx_1 + Cx_1^2 \\ y_2 &= p(x_2) = A + Bx_2 + Cx_2^2 \end{aligned}$$

This is a linear system:

$$\begin{pmatrix} 1 & x_0 & x_0^2 \\ 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \end{pmatrix} \begin{pmatrix} A \\ B \\ C \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \end{pmatrix}$$

A Matrix of the form

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{pmatrix}$$

is called a **Vandermonde Matrix**.

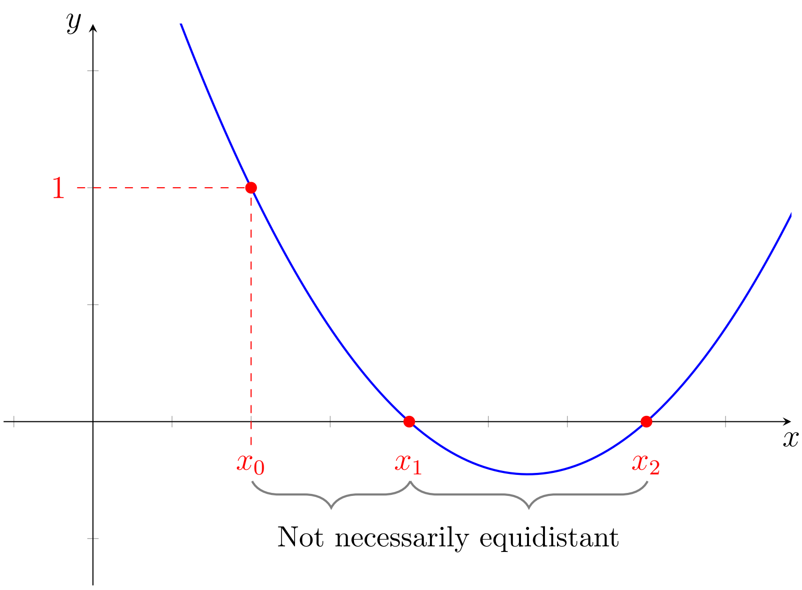
It's determinant is

$$\prod_{i<j} (x_i - x_j)$$

so it will be non-zero when all the x_i's are different. This restriction does not interfere with an interpolation problem, since it also requires different x_i's.

<u>Theorem</u>: For $n+1$ data points $(x_0, y_0), (x_1, y_1), ...(x_n, y_n)$ there is a unique polynomial of degree n interpolating these points.

This polynomial could be computed by Gaussian Elimination, but usually that leads to numerical problems (round-off errors will amplify). *There is a smarter way!*



$$L_0(x) = a(x - x_1)(x - x_2)$$

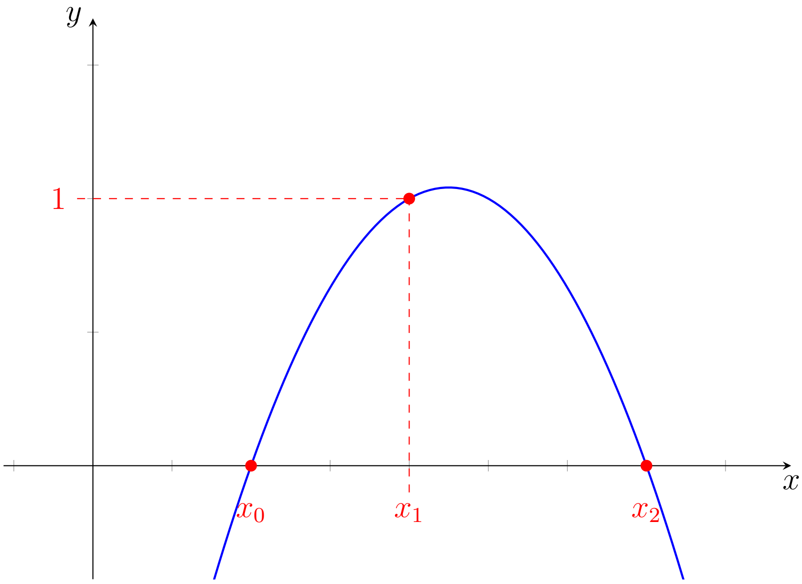
Choose:

$$a = \frac{1}{(x_0 - x_1)(x_0 - x_2)}$$

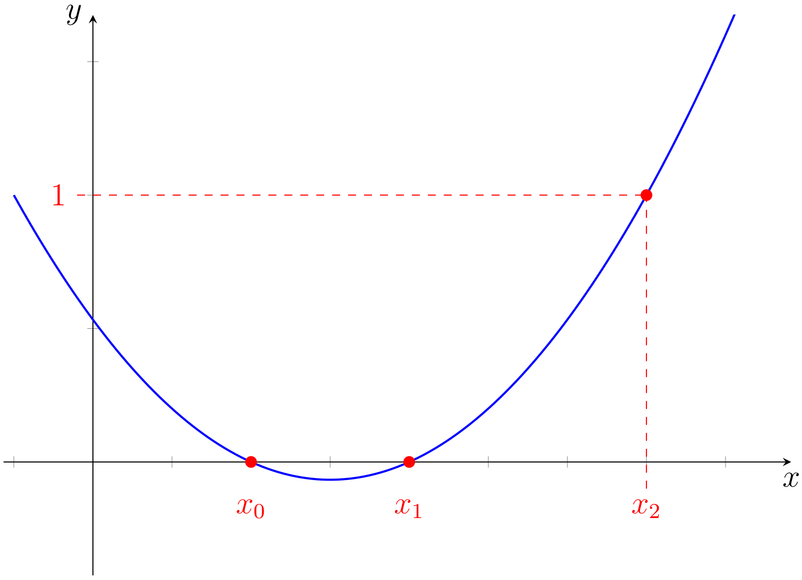
$$\Rightarrow \; L_0(x) = \frac{(x - x_1)(x - x_2)}{x_0 - x_1)(x_0 - x_2)}$$

So:

$$\left.\begin{array}{l} L_0(x_0) = 1 \\ L_0(x_1) = 0 \\ L_0(x_2) = 0 \end{array}\right\} \text{This is the } \textbf{interpolating parablola}$$



$$L_1(x) = \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)}$$



$$L_2(x) = \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)}$$

> Now with (x_0, y_0), (x_1, y_1), and (x_2, y_2), the interpolating polynomial is:
>
> $$p(x) = y_0 \cdot L_0(x) + y_1 \cdot L_1(x) + y_2 \cdot L_2(x)$$

2.2 Newtonian Interpolation

When data points are "added" the Lagrange polynomials have to be computed all over again. The Newton approach is designed in a way that we can use previous calculations when interpolating with additional data points.

Example: Three Coordinates (1,-2), (2,5), and (-1,-4)

Align the coordinates in two separate columns:

x	y
1	-2
2	5
-1	-4

Subtract the first x-coordinate from each subsequent coordinate, and write the difference in the next column to the left.

	x	y
	1	-2
2 - 1 = **1** ←	2	5
-1 - 1 = **-2** ←	-1	-4

In the next column subtract the first value of that column from each subsequent value, just as you did in the first column:

		x	y
		1	-2
	1	2	5
-2 - 1 = **-3** ←	-2	-1	-4

Repeat this process until there is a left-most column with only one value.

For the y-coordinates, the process is a bit different. Just like with the x-coordinates, subtract the first value in each column from all subsequent values in that column. Then, divide the resulting difference by the difference of the corresponding x-values (i.e. the answers on the left side).

$$
\begin{array}{cccccc}
 & & x & y & & \\
\hline
 & & 1 & -2 & & \\
 & 1 & 2 & 5 & \rightarrow \frac{5-(-2)}{1} = \mathbf{7} & \\
-3 & -2 & -1 & -4 & \frac{-4-(-2)}{-2} = \mathbf{1} &
\end{array}
$$

Repeat this process until there is a right-most column with only one value:

$$
\begin{array}{cccccc}
 & & x & y & & \\
\hline
 & & 1 & -2 & & \\
 & 1 & 2 & 5 & 7 & \\
-3 & -2 & -1 & -4 & 1 \rightarrow \frac{1-7}{-3} = \mathbf{2} &
\end{array}
$$

How do we turn this into a polynomial?

$$
\begin{array}{cccccc}
 & & x & y & & \\
\hline
 & & \boxed{1} & \boxed{-2} & & \\
 & 1 & \boxed{2} & 5 & \boxed{7} & \\
-3 & -2 & -1 & -4 & 1 & \boxed{2}
\end{array}
$$

Beginning with your first y-coordinate, multiply the first value in each y-column as a coefficient in a polynomial of increasing degree, starting with degree 0. These polynomials will be composed of expressions of $(x - x_{n-1})$ Where x_{n-1} are your original x-coordinates. Add these polynomials together for the desired formula:

$$
\begin{aligned}
p(x) &= -2 \underbrace{\cdot\ 1}_{\text{degree } 0} + 7\underbrace{(x-1)}_{\text{degree } 1} + 2\underbrace{(x-1)(x-2)}_{\text{degree } 2} \\
&= -2 + 7x - 7 + 2x^2 - 6x + 4 \\
&= 2x^2 + x - 5
\end{aligned}
$$

What happens if we want to add an additional data-point?

Example: Three Coordinates (1,-2), (2,5), (-1,-4), and (-2, -11):

$$
\begin{array}{cccccc}
 & & x & y & & \\
\hline
 & & \mathbf{1} & \mathbf{-2} & & \\
 & 1 & \mathbf{2} & \mathbf{5} & 7 & \\
-3 & -2 & \mathbf{-1} & \mathbf{-4} & 1 & 2 \\
 & & -2 & -11 & &
\end{array}
$$

If we begin building the triangle as we did before, we get the same values we did previously for the first three three coordinates. *The same points in the same order will always result in the same triangle.* Because of this, adding a new coordinate just means adding a new row.

Original values

$$
\begin{array}{cccccccc}
 & & & x & y & & & \\
\hline
 & & & 1 & -2 & & & \\
 & & 1 & 2 & 5 & 7 & & \\
 & -3 & -2 & -1 & -4 & 1 & 2 & \\
-1 & -4 & -3 & \mathbf{-2} & \mathbf{-11} & 3 & 1 & 1
\end{array}
$$

Newly calculated values

Repeat the process from the first three coordinates for the new coordinate to obtain fourth row to the triangle. Just as we were able to add the new coordinates to the last row of the triangle, we can append the new calculated values to the end of our previously obtained polynomial:

$$
\begin{array}{cccccccc}
 & & & x & y & & & \\
\hline
 & & & \boxed{1} & \boxed{-2} & & & \\
 & & 1 & \boxed{2} & 5 & \boxed{7} & & \\
 & -3 & -2 & \boxed{-1} & -4 & 1 & \boxed{2} & \\
-1 & -4 & -3 & -2 & -11 & 3 & 1 & \boxed{1}
\end{array}
$$

$$p(x) = \underbrace{-2 \cdot 1 + 7(x-1) + 2(x-1)(x-2)}_{\text{previously obtained}} + 1 \cdot (x-1)(x-2)(x-(-1))$$

So we see that while the Newton method of interpolation is a bit complicated, it has the nice property of allowing us to add more points of data without having to recalculate our previous values.

2.3 Interpolating Non-Polynomial Functions

Example: Use data points from the function $f(x) = \frac{1}{1+25x^2}$:

x	± 1	± 0.8	± 0.6	± 0.4	± 0.2	0
$f(x)$	0.038	0.058	$\frac{1}{10}$	$\frac{1}{5}$	$\frac{1}{2}$	1

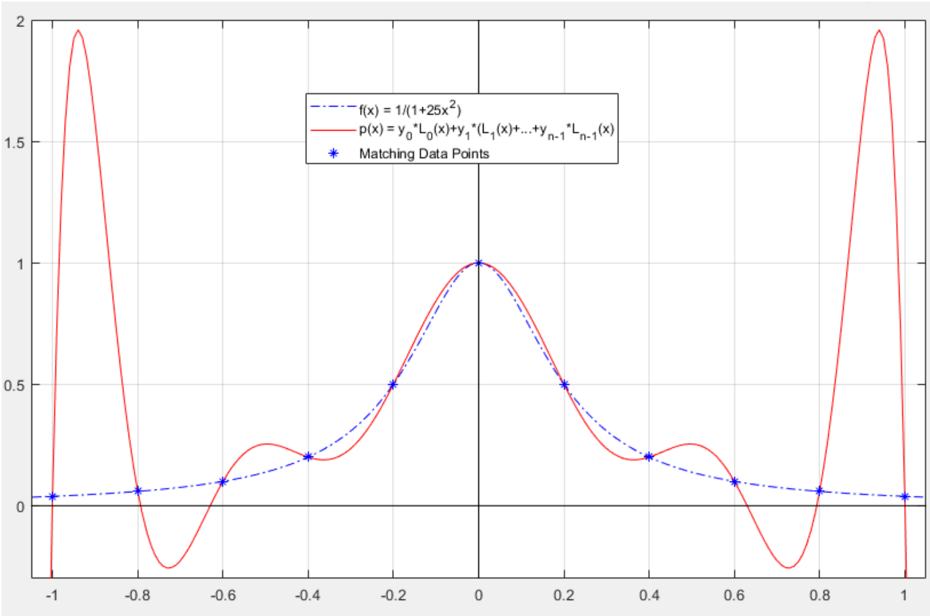


Figure 2.1: We observe high oscillations between the interpolated data points.

What is going on here?

- We try to reconstruct a function that has non-polynomial behavior (horizontal asymptote).
- Equidistant points create extra trouble. *Cluster points at the end points to obtain better results.*

Let's try more midpoints: 0, ± 0.1, $\pm 0.2 \ldots \pm 1$. Now we have 21 points instead of 11:

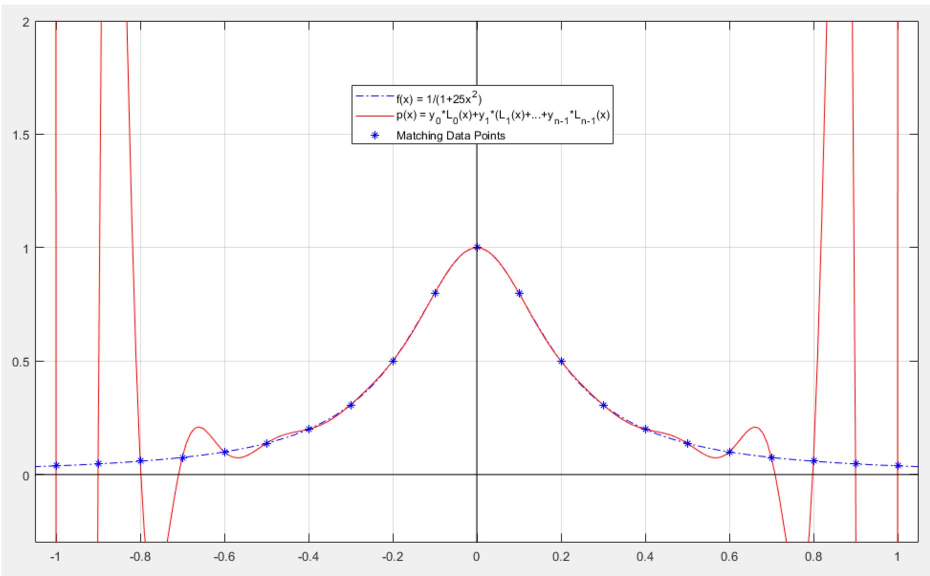
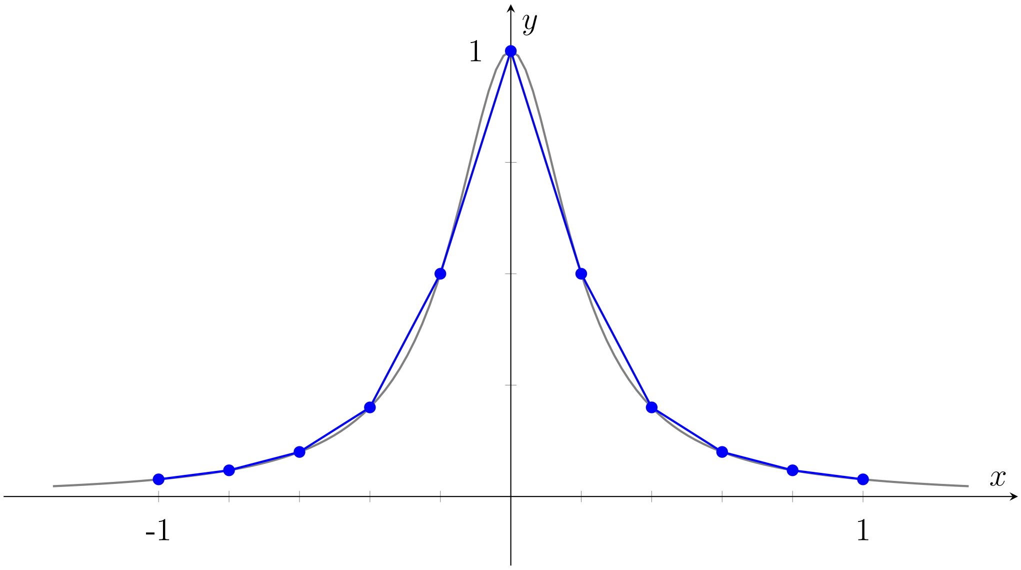


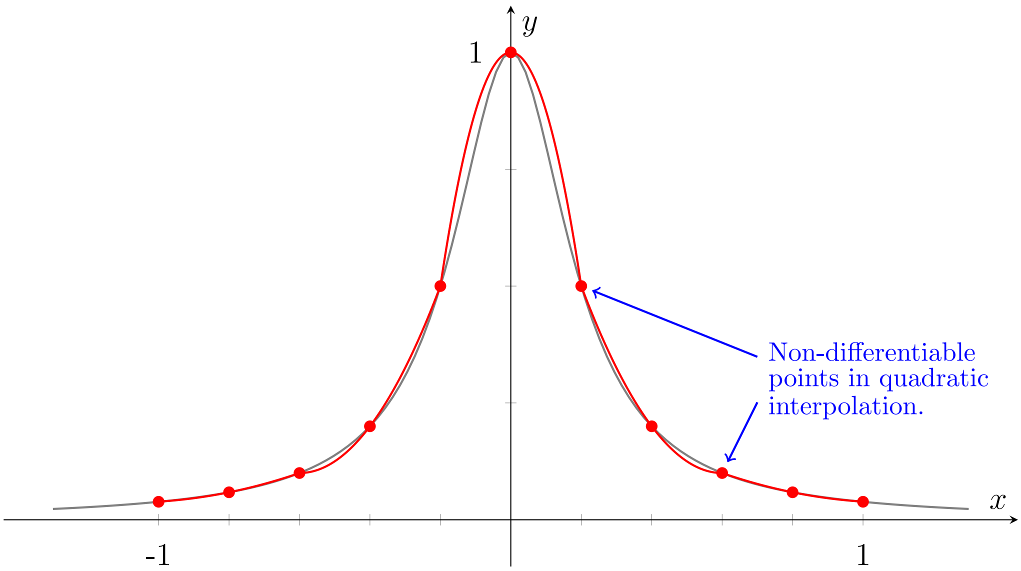
Figure 2.2: Increasing the number of points leads to more oscillations.

2.4 Spline Interpolation

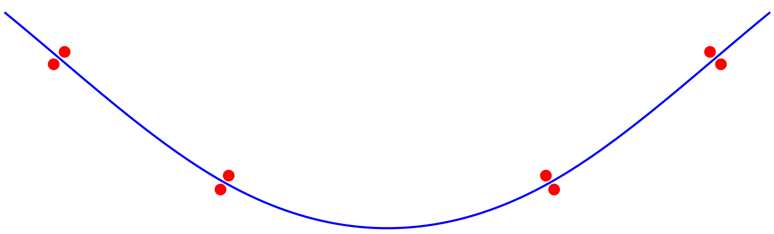
Increasing the number of data points in our interpolating polynomial leads to worse results. What is a practical solution? We could try using piecewise interpolation with small degree polynomials. The simplest case of this would be piecewise linear interpolation.



Piecewise Linear interpolation gets rid of oscillations and is continuous, but usually isn't differentiable at the interpolating points.



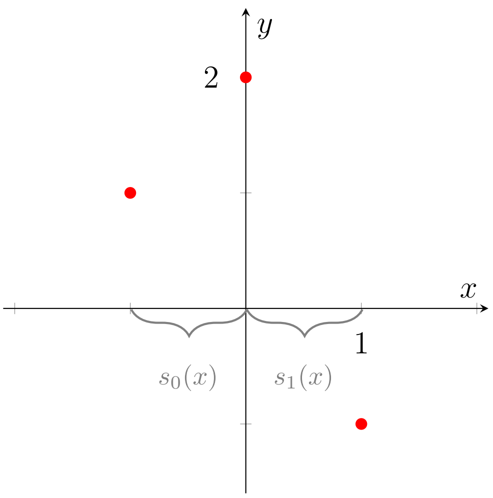
Piecewise Quadratic interpolation has better curvature, much closer to the original function. Unlike piecewise linear interpolation it's differentiable at every other point. Piecewise quadratic interpolation is popular for approximate integration Use function values and integrate the interpolating parabola's. (*Simpson's Rule*).



The term *spline* comes from historic ship building methods, which involved bending long flexible wooden beams into place, using pins. **Spline Interpolation** is a type of piecewise polynomial interpolation, in which the function values, derivatives, curvature, etc. at each of the interpolating points.

The standard method is **Cubic Spline Interpolation**, which uses 3rd order polynomials with matching function values, slope, and curvature.

Example: 3 points: (-1, 1) (0, 2) (1, 0)



The interpolated function will be composed of the following splines:

$$s_0(x) = ax^3 + bx^2 + cx + d \text{ on [-1,0]}$$
$$s_1(x) = ex^3 + fx^2 + gx + h \text{ on [0,1]}$$

We must determine eight parameters:

$$s_0(x_0) = y_0 \quad \Rightarrow \quad s_0(-1) = 1$$
$$\left.\begin{array}{lll} s_0(x_1) = y_1 & \Rightarrow & s_0(0) = 2 \\ s_1(x_1) = y_1 & \Rightarrow & s_1(0) = 2 \end{array}\right\} \text{Matching Values}$$
$$s_1(x_2) = y_2 \quad \Rightarrow \quad s_1(1) = -1$$

Matching slope at x_1:

$$s_0{}'(x_1) = s_1{}'(x_1) \quad \Rightarrow \quad s_0{}'(0) = s_1{}'(0)$$

Matching "Curvature":

$$s_0{}''(x_1) = s_1{}''(x_1) \quad \Rightarrow \quad s_0{}''(0) = s_1{}''(0)$$

Two conditions remain. There are several approaches to "fix" these conditions:

$$s_0{}''(x_0) = 0 \qquad s_1{}''(0) = s_1{}''(x_2) = 0$$

This is referred to as **Natural Cubic Splines**. The curvature at the endpoints is zero. This mimics the behavior of the wooden beams in ship building.

With **Clamped Splines** the second derivative of each of the endpoint set to a specified value.

$$s_0{}''(x_0) = \text{[some value]} \qquad s_1{}''(0) = s_1{}''(x_2) = \text{[some value]}$$

Periodic Conditions can also be specified to match the end points of the interpolated function into one that is periodic.

In our example, let's find the natural cubic spline:

$$
\begin{gathered}
s_0{}'(x) = 3ax^2 + 2bx + c \\
s_0{}''(x) = 6ax + 2b \\
s_1{}'(x) = 3ex^2 + 2fx + g \\
s_1{}''(x) = 6ex + 2fx
\end{gathered}
$$

Interpolation Conditions:

$$s_0(-1) = a \cdot (-1)^3 + b \cdot (-1)^2 + c \cdot (-1) + d = 1$$

$$\Rightarrow \textbf{-a} + \textbf{b} \textbf{-} \textbf{c} + \textbf{d} = \textbf{1}$$

$$s_0(0) = a \cdot 0^3 + b \cdot 0^2 + c \cdot 0 + d = 2$$

$$\Rightarrow \textbf{d} = \textbf{2}$$

$$s_1(0) = e \cdot 0^3 + f \cdot 0^2 + g \cdot 0 + h = 2$$

$$\Rightarrow \textbf{h} = \textbf{2}$$

$$s_1 = e \cdot 1^3 + f \cdot 1^2 + g \cdot 1 + h = -1$$

$$\Rightarrow \textbf{e} + \textbf{f} + \textbf{g} + \textbf{h} = \textbf{-1}$$

Matching Slope:

$$s_0{}'(0) = s_1{}'(0)$$

$$\Rightarrow 3a \cdot 0^2 + 2b \cdot 0 + c = 3e \cdot 0^2 + 2f \cdot 0 + g$$

$$\Rightarrow \textbf{c} = \textbf{g}$$

Matching Curvature:

$$s_0{}''(0) = s_1{}''(0)$$

$$\Rightarrow 6a \cdot 0 + 2b = 6e \cdot 0 + 2f \Rightarrow 2b = 2f \Rightarrow \textbf{b} = \textbf{f}$$

Natural Conditions:

$$s_0{}''(-1) = 6a \cdot (-1) + 2b = 0$$

$$\Rightarrow \textbf{-6a} + \textbf{2b=0}$$

$$s_0{}''(1) = 6e \cdot 1 + 2f = 0$$

$$\Rightarrow \textbf{6e} + \textbf{2f=0}$$

This leads to a linear system with eight equations for eight unknowns. In this spline set up, this system has a unique solution.

$$\begin{aligned} -a + b - c + d &= 1 \\ d &= 2 \\ h &= 2 \\ e + f + g + h &= -1 \\ c &= g \\ b &= f \\ -6a + 2b &= 0 \\ 6e + 2f &= 0 \end{aligned}$$

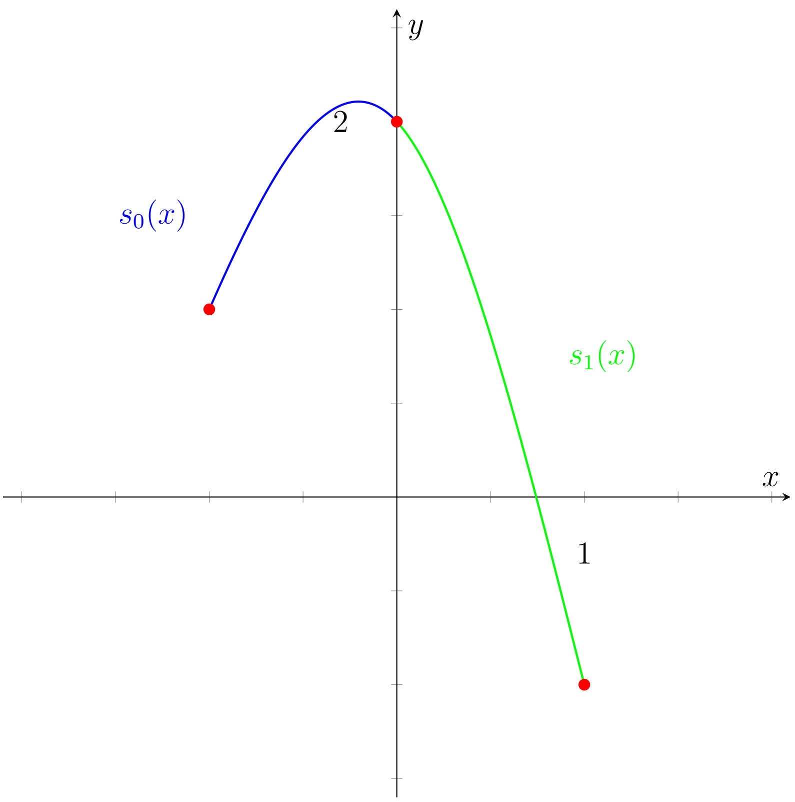
Through Gaussian Elimination we find the solutions:

$$a = -\tfrac{4}{5}$$
$$b = f = -\tfrac{72}{25}$$
$$c = g = -\tfrac{-27}{25}$$
$$d = h = 2$$
$$e = \tfrac{24}{25}$$

When we plug these values into our original equations, we get the following:

$$s_0(x) = -\tfrac{4}{5}x^3 - \tfrac{72}{25}x^2 - \tfrac{27}{25}x + 2$$

$$s_1(x) = \tfrac{24}{25}x^3 - \tfrac{72}{25}x^2 - \tfrac{27}{25}x + 2$$



In the preceding example, we had 3 data points, with 2 interpolating spline sections in between them. This process can be generalized to include more data points and more sections:

Requirements for calculating Cubic Splines

$$(n+1) \text{ data points} \quad \Rightarrow \quad s_k(x) \text{ on } [x_k, x_{k+1}]$$
$$\text{for } k = 0.1, ..., n-1$$

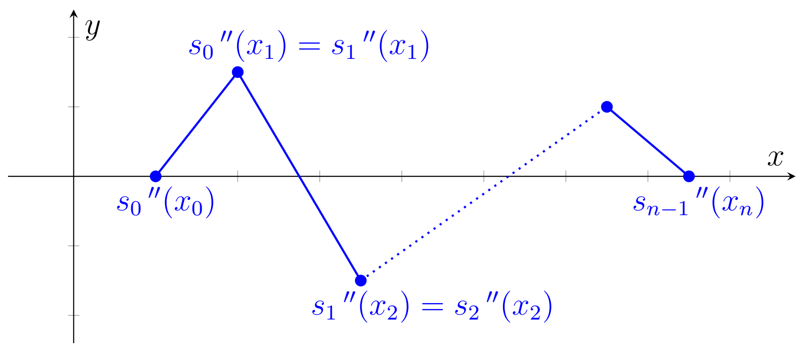
$4 \cdot n$ coefficients:

- $2n$ *Interpolation Conditions*
- $n-1$ *Slope Conditions* for each "internal point"
- $n-1$ *Curvature Conditions* for each internal point

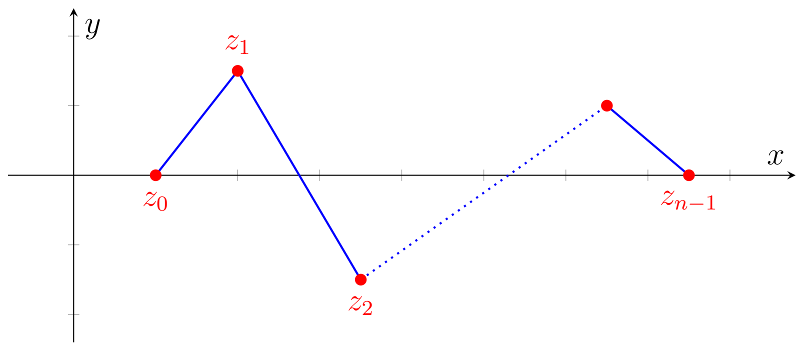
$\Rightarrow \quad 4n-2$ conditions.

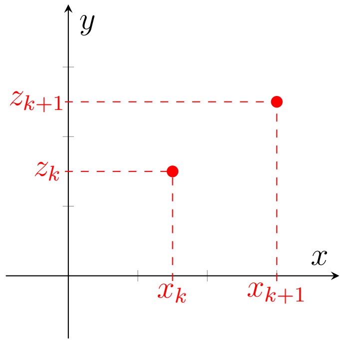
The two remaining "end" conditions can be determined to fit the nature of the data (i.e. "natural" splines, clamped, periodic, etc.)

To find formulas we start with the second derivative of the spline, which is a continuous, piecewise linear function.



Rename the unknown second derivatives at x_k to z_k:





$$s_k{}''(x) = z_k + \frac{z_{k+1}-z_k}{x_{k+1}-x_k}$$

We can abbreviate the expression $x_{k+1} - x_k$ as Δ_k, resulting in:

$$s_k{}''(x) = z_k + \frac{z_{k+1}-z_k}{\Delta_k}$$

Integrate:

$$s_k{}'(x) = C_k + z_k(x - x_k) + \frac{z_{k+1}-z_k}{\Delta_k} \cdot \frac{(x-x_k)^2}{6}$$

Integrate once more:

$$s_k(x) = D_k + C_k(x - x_k) + \frac{z_k}{2}\ (x - x_k)^2 + \frac{z_{k+1}-z_k}{\Delta_k} \cdot \frac{(x-x_k)^3}{6}$$

Use interpolation conditions to find D_k and C_k:

$$y_k = s_k(x_k) = D_k$$

$$y_{k+1} = s_k(x_{k+1}) = y_k + C_k\Delta_k + \frac{z_k}{2}\ \Delta_k{}^2 + \frac{z_{k+1}-z_k}{\cancel{\Delta_k}^{1}} \cdot \frac{\Delta_k^{\cancel{3}\,2}}{6}$$

Solve for C_k:

$$C_k = \frac{y_{k+1}-y_k}{\Delta_k} - \frac{z_k}{2}\Delta_k - \frac{z_{k+1}-z_k}{6}\Delta_k$$

$$= \frac{y_{k+1}-y_k}{\Delta k} - \left(\frac{z_{k+1}}{6} + \frac{z_k}{3}\right)\Delta_k$$

Hence:

$$s_k(x) = y_k + \left[\ \frac{y_{k+1}-y_k}{\Delta_k} - \left(\frac{z_{k+1}}{6} + \frac{z_k}{3}\right)\Delta_k\right]$$

$$s_k{}'(x) = \frac{y_{k+1}-y_k}{\Delta_k} - \left(\frac{z_{k+1}}{6} + \frac{z_k}{3}\right)\Delta_k + z_k(x - x_k) + \frac{z_{k+1}-z_k}{\Delta_k} \cdot \frac{(x-x_k)^2}{2}$$

$$s_{k+1}{}'(x) = \frac{y_{k+2}-y_{k+1}}{\Delta_{k+1}} - \left(\frac{z_{k+2}}{6} + \frac{z_{k+1}}{3}\right)\Delta_{k+1} + z_{k+1}(x - x_{k+1}) + \frac{z_{k+2}-z_{k+1}}{\Delta_{k+1}} \cdot \frac{(x-x_{k+1})^2}{2}$$

Match the slopes at x_{k+1}, i.e. $s_k{}'(x_{k+1}) = s_{k+1}{}'(x_{k+1})$:

$$\frac{y_{k+1}-y_k}{\Delta_k} - \left(\frac{z_{k+1}}{6} - \frac{z_k}{3}\right)\Delta_k + z_k \cdot \Delta_k + \frac{z_{k+1}-z_k}{\cancel{\Delta_k}} \cdot \frac{\Delta_k{}^{\cancel{2}}}{2} = \frac{y_{k+2}-y_{k+1}}{\Delta_{k+1}} - \left(\frac{z_{k+2}}{6} + \frac{z_{k+1}}{3}\right)\Delta_{k+1}$$

$$\text{for } k = 0, 1, \ldots n - 2$$

Rewrite:

$$\boxed{\begin{gathered}\frac{\Delta_k}{6} z_k + \left(\frac{\Delta_{k+1}+\Delta_k}{3}\right) z_{k+1} + \frac{\Delta_{k+1}}{6} z_{k+2} = \frac{y_{k+2}-y_{k+1}}{\Delta_{k+1}} - \frac{y_{k+1}-y_k}{\Delta_k} \\ \text{for } k = 0, 1, \ldots n-2\end{gathered}}$$

Example: $n = 5$

z_0	z_1	z_2	z_3	z_4	z_5	RHS
$\frac{x_1-x_0}{6}$	$\frac{x_2-x_1}{3} + \frac{x_1-x_0}{3}$	$\frac{x_2-x_1}{6}$	0	0	0	$\frac{y_2-y_1}{x_2-x_1} - \frac{y_1-y_0}{x_1-x_0}$
0	$\frac{x_2-x_1}{6}$	$\frac{x_3-x_2}{3} + \frac{x_2-x_1}{3}$	$\frac{x_3-x_2}{6}$	0	0	$\frac{y_3-y_2}{x_3-x_2} - \frac{y_2-y_1}{x_2-x_1}$
0	0	$\frac{x_3-x_2}{6}$	$\frac{x_4-x_3}{3} + \frac{x_3-x_2}{3}$	$\frac{x_4-x_3}{6}$	0	$\frac{y_4-y_3}{x_4-x_3} - \frac{y_3-y_2}{x_3-x_2}$
0	0	0	$\frac{x_4-x_3}{6}$	$\frac{x_5-x_4}{3} + \frac{x_4-x_3}{3}$	$\frac{x_4-x_3}{6}$	$\frac{y_5-y_4}{x_5-x_4} - \frac{y_4-y_3}{x_4-x_3}$

- Symmetric Matrix (good)
- "Positive Definite" (better)
- Many zeros (perfect)

This matrix can be simplified in the following ways:

$$\frac{x_p-x_q}{3} + \frac{x_q-x_r}{3} = \frac{x_p \cancel{-x_q+x_q} -x_r}{3} = \frac{x_p-x_r}{3}$$

Also, if we choose natural splines for our end conditions, we can set z_0 and z_5 to 0, thereby eliminating the first and last columns of the matrix:

z_1	z_2	z_3	z_4	RHS
$\frac{x_2-x_1}{3} + \frac{x_1-x_0}{3}$	$\frac{x_2-x_1}{6}$	0	0	$\frac{y_2-y_1}{x_2-x_1} - \frac{y_1-y_0}{x_1-x_0}$
$\frac{x_2-x_1}{6}$	$\frac{x_3-x_2}{3} + \frac{x_2-x_1}{3}$	$\frac{x_3-x_2}{6}$	0	$\frac{y_3-y_2}{x_3-x_2} - \frac{y_2-y_1}{x_2-x_1}$
0	$\frac{x_3-x_2}{6}$	$\frac{x_4-x_3}{3} + \frac{x_3-x_2}{3}$	$\frac{x_4-x_3}{6}$	$\frac{y_4-y_3}{x_4-x_3} - \frac{y_3-y_2}{x_3-x_2}$
0	0	$\frac{x_4-x_3}{6}$	$\frac{x_5-x_4}{3} + \frac{x_4-x_3}{3}$	$\frac{y_5-y_4}{x_5-x_4} - \frac{y_4-y_3}{x_4-x_3}$

Such a system can be solved very quickly.

- $z_0, z_1, \ldots z_5$ (Second Derivatives)
- Very little error propogation (Positive Definite)

Plug in to obtain splines:

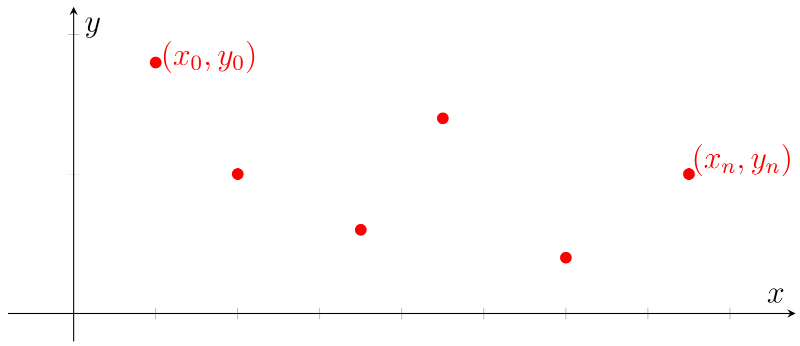
$$s_k(x) = y_k + ... \cdot \frac{(x-x_k)^3}{3}$$

$$s(x) = \begin{cases} s_0(x) & \text{if} x \in [x_0, x_1] \\ s_1(x) & \text{if} x \in [x_1, x_2] \\ \quad\vdots & \\ s_{n-1}(x) & \text{if} x \in [x_{n-1}, x_n] \end{cases}$$

Chapter 3

Numerical Differentiation

Differentiation rules from Calculus require a function. How do we differentiate when we are only given a set of data points?



One option is to use interpolation. We can consider these points as samples of a graph $y = f(x)$.

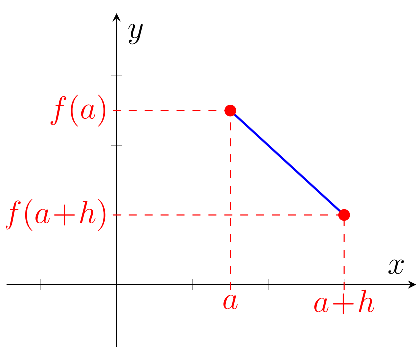
Another option is to look at the definition of the derivative:

$$f'(a) = \lim_{h \to 0} \frac{f(a+h) - f(a)}{h} \approx \frac{f(a+h) - f(a)}{h} \qquad \text{When } h \text{ is close to } 0.$$

Since we cannot get arbitrarily close, let's use the closest value available.

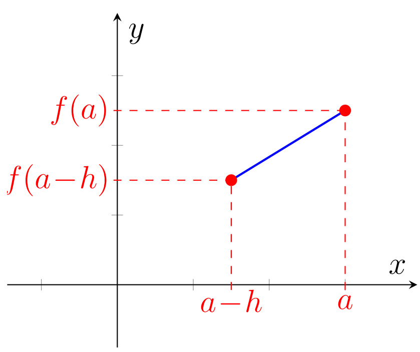
Forward Difference

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



Backward Difference

$$f'(a) \approx \frac{fa-h) - f(a)}{-h}$$



Which form is better? Should we average these values for a better approximation? Certainly, the approximations improve when the so-called *step size* h becomes smaller.

We want some quantitative measure.

We can use Taylor series to find out more details:

Taylor Series:

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 \dots$$

Plug in $x = a + h \Leftrightarrow x - a = h$ to obtain:

$$f(a+h) = f(a) + f'(a)\cdot h + \frac{f''(a)}{2!}h^2 + \frac{f'''(a)}{3!}h^3 + \dots$$

Then our forward difference turns into:

$$\frac{f(a+h)-f(a)}{h} = \frac{\cancel{f(a)} + \left(f'(a)\cancel{h} + \frac{f''}{2!}h^{\cancel{2}} + \dots\right) - \cancel{f(a)}}{\cancel{h}}$$

$$= \underbrace{f'(a)}_{\text{wanted}} + \underbrace{\overbrace{\frac{f''(a)}{2!}h}^{\text{dominating term}} + \frac{f'''(a)}{3!}h^2}_{\text{truncation error}} + \dots$$

The truncation error is approximately of the form $constant \cdot h^1$. If we divide the step size by 10, the error should become smaller by a factor of 10. We say the forward difference is a *first order* approximation for the derivative.

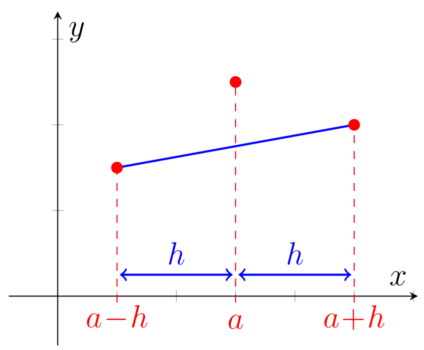
Now, let's analyze the backward difference: Plug in $x = a - h$ in the Taylor series to obtain:

$$f(a-h) = f(a) + f'(a)\cdot(-h) + \frac{f''(a)}{2!}(-h)^2 + \frac{f'''(a)}{3!}(-h)^3 + \dots$$

$$\Rightarrow \frac{f(a-h)-f(a)}{-h} = \frac{\cancel{f(a)} + \left(f'(a)\cancel{(-h)} + \frac{f''}{2!}(-h)^{\cancel{2}} + \dots\right) - \cancel{f(a)}}{\cancel{-h}}$$

$$= \overbrace{f'(a)}^{\text{wanted}} + \underbrace{\overbrace{\frac{f''(a)}{2!}(-h)}^{\text{dominating term}} + \frac{f'''(a)}{3!}(-h)^2}_{\text{truncation error depending on step size h}} + \dots$$

Let's assume that the neighboring points are equidistant.



Then, averaging the forward and backward distance leads to:

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

Centered Difference

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

This is the slope of the linear interpolation of the two neighboring points.

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

$$=\frac{\left(\cancel{f(a)}+f'(a)\cdot h+\cancel{\frac{f''(a)}{2!}h^2}+\frac{f'''(a)}{3!}h^3+\ldots\right)-\left(\cancel{f(a)}+f'(a)(-h)+\cancel{\frac{f''(a)}{2!}(-h)^2}+\ldots\right)}{2h}$$

$$=\frac{1}{2h}\left(2f'(a)\cdot h+\frac{f'''(a)}{3!}h^3+\ldots\right)=\underbrace{f'(a)}_{\text{wanted}}+\underbrace{\overbrace{\frac{f'''(a)}{3!}h^2}^{\text{Dominating term now dependent upon } h^2}+\ldots}_{\text{truncation error}}$$

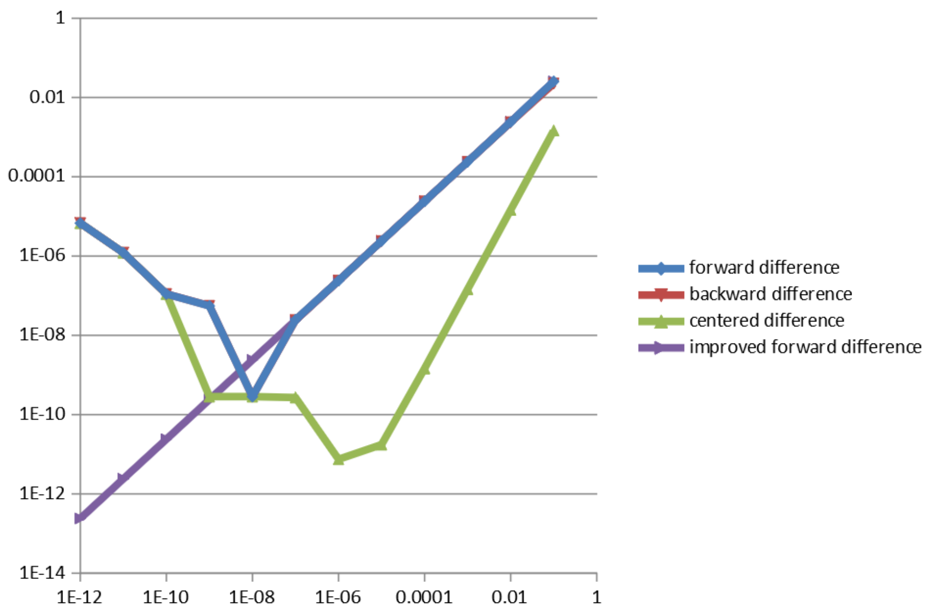


Figure 3.1: Round-off error vs stepsize h for $f(x)=\sin x$

For the centered difference formula, the truncation error is dominated by $constant \cdot h^2$. Reducing the step size by a factor of 10 reduces error by a factor of 100. *This is a second-order approximation method.*

<u>*Side Remark*</u>: The Excel experiments showed round-off errors for small step sizes because the difference formulas involve a subtraction of nearly equal numbers.

Example: Forward Difference: $\frac{\sin(a+h)-\sin a}{h}$

How can we avoid this?

$$\sin(x+y)-\sin(x-y) = \sin x\cos y + \sin y\cos x - \big(\sin x\cos(-y)+\sin(-y)\cos x\big)$$
$$= \cancel{\sin x\cos y} + \sin y\cos x - \big(\cancel{\sin x\cos y} - \sin y\cos x\big)$$
$$= 2\sin y\cdot\cos x$$

We need:
$$x+y = a+h$$
$$x-y = a$$

Adding:
$$2x = 2a+h \Leftrightarrow x = a+\tfrac{h}{2}$$

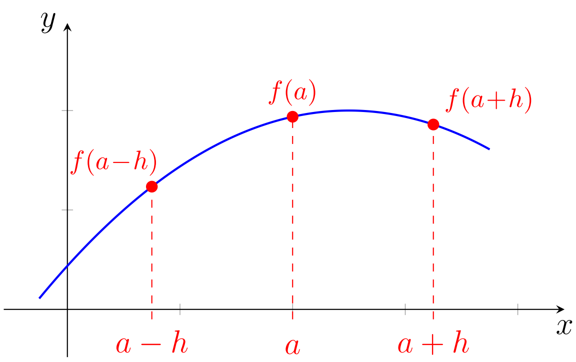
Subtracting:
$$2y = h \Leftrightarrow y = \tfrac{h}{2}$$

We find:
$$\sin(a+h) - sin(a) = 2\sin\tfrac{h}{2}\cdot\big(a+\tfrac{h}{2}\big)$$

<u>Excel Results</u>: We no longer see *any* round-off error, using the product formula instead of subtraction.

How can we find **finite difference** formulas that fit the given problem?

Use quadratic interpolation for three points.



<u>Interpolating parabola</u>:

$$p(x) = f(a-h)\frac{\big(x-a\big)\big(x-(a+h)\big)}{\big(a-h-a\big)\big(a-h-(a+h)\big)}$$
$$+f(a)\frac{\big(x-(a-h)\big)\big(x-(a+h)\big)}{a-\big(a-h\big)\big(a-(a+h)\big)}$$
$$+f(a+h)\frac{\big(x-(a-h)\big)\big(x-a\big)}{\big(a+h-(a-h)\big)\big((a+h)-a\big)}$$

$$= f(f-h)\ \frac{x^2-(2a+h)x+a(a+h)}{-h(-2h)}\ \ f(a)\ \frac{(x-a)^2-h^2}{h\cdot(-h)}\ +f(a+h)\frac{x^2-(2a-h)x+a(a-h)}{2hh}$$

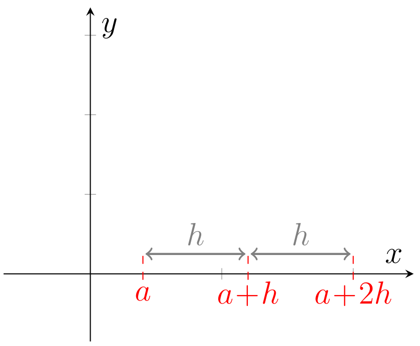
Then $p'(x) =$
$$f(a-h)\frac{2x-(2a+h)}{2h^2} + f(a)\frac{2x-2a}{-h^2} + f(a+h)\frac{2x-(2a-h)}{2h^2}$$

What is happening at a?
$$p'(a) = f(a-h)\frac{-\cancel{h}}{2h^{\cancel{2}}} + \cancel{f(a)\cdot 0} + f(a+h)\frac{\cancel{h}}{2h^{\cancel{2}}} = \underbrace{\frac{f(a+h)-f(a-h)}{2h}}_{\text{Centered Difference Formula}}$$

<u>We learn</u>: To find a second order finite difference approximation use an interpolating parabola.

3.1 Another Way of Creating Formulas:

Use Taylor Series



Find a second order formula for the derivative at a when $\big(a, f(a)\big), \big(a+h, f(a+h)\big), \big(a+2h, f(a+2h)\big)$ are given. Derivative at left end point.

$$f(a+h) = f(a) + f'(a)h + \frac{f''(a)}{2!}h^2 + \frac{f'''(a)}{3!}h^3 + \dots$$

$$f(a+2h) = f(a) + f' \cdot 2h + \frac{f''(a)}{2!}4h^2 + \frac{f'''(a)}{3!}8h^3 + \dots$$

Structure of finite difference formula:

$$f'(a) \approx A \cdot f(a) + B \cdot f(a+h) + C \cdot f(a+2h)$$

$$f'(a) \approx A \cdot f(a) + B \cdot f(a) + B \cdot f'(a)h + B\frac{f''(a)}{2}\ h^2 + B\ \frac{f'''(a)}{3!}\ h^3$$
$$+C \cdot f(a) + C \cdot f'(a)2h + C\frac{f''(a)}{2}\ 4h^2 + C\ \frac{f'''(a)}{3!}8h^3$$

What do we want?

$A + B + C = 0$ to get rid of $f(a)$

$hB + 2hC = 1$ to get $f'(a)$

$\frac{B}{2}h^2 + \frac{C}{2}4h^2 = 0$ to get rid of $f''(a)$ ← *truncation error*

A	B	C	$r.h.s.$
1	1	1	0 I
0	1	2	$\frac{1}{h}$ II
0	1	4	0 III - II
1	1	1	0
0	1	2	$\frac{1}{h}$
0	0	2	$\frac{-1}{h}$

Back Substitute
$C = \frac{-1}{2h}$

Plug into II:
$B = \frac{1}{h} - 2\big(\frac{-1}{2h}\big) = \frac{2}{h}$

Plug into I:
$A = -B - C = \frac{-2}{h} + \frac{1}{2h} = \frac{-4}{2h} + \frac{1}{2h} = \frac{-3}{2h}$

We obtain:

$$f'(a) \approx \frac{-3}{2h}f(a) + \frac{4}{2h}f(a+h) - \frac{1}{2h}f(a+2h)$$

$$= \frac{-3f(a)+4f(a+h)-f(a+2h)}{2h}$$

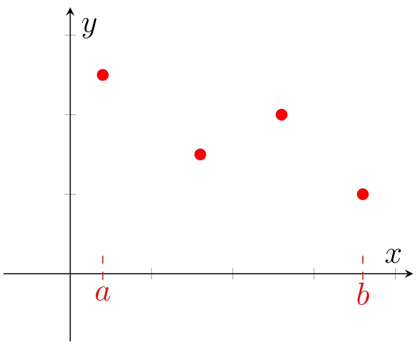
Chapter 4

Numerical Integration

> **<u>Fundamental Theorem of Calculus</u>:**
> If f is continuous on $[a, b]$ then
> $$\int_a^b f(x)dx = F(b) - F(a)$$
> where F is any anti-derivative of f.

<u>Problem</u>: It may be difficult to find an anti-derivative or even impossible to write as a finite combination of "basic functions". (Power series may help using infinite combinations but not all functions possess power series representation.)

Another motivation: How do we integrate when the function is only known by some sample values?

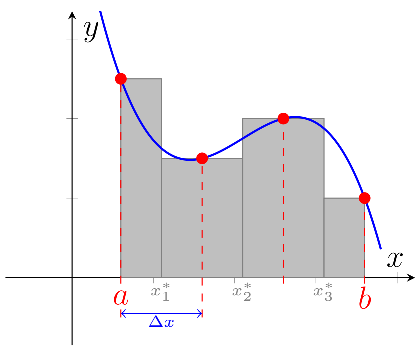


$$\int_a^b f(x)dx = ?$$

<u>Already expected</u>: Use polynomial interpolation or piecewise polynomial interpolation (to avoid large oscillations).

First let's consider the definition of the definite integral as limit of the Riemann Sums:

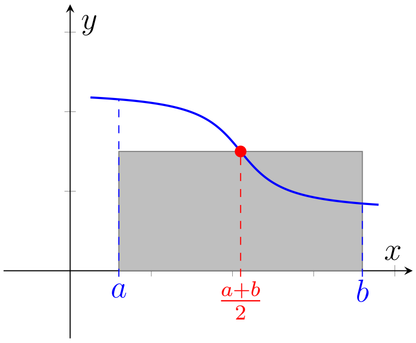
> **<u>Riemann Sum</u>**
> $$\int_a^b f(x)dx = \lim_{n\to\infty} \sum_{i=1}^{n} f(x_i^*) \cdot \frac{b-a}{n}$$



We obtain an approximation of the definite integral using a Riemann Sum:

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} f(x_i^*) \cdot \frac{b-a}{n}$$

4.1 Midpoint Rule

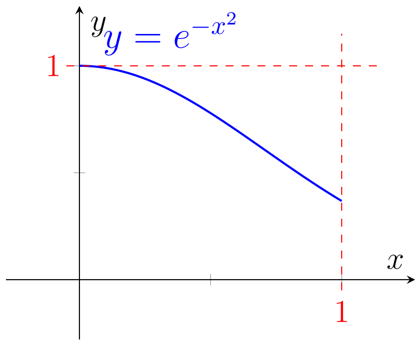


$$\int_a^b f(x)dx \approx \underbrace{\underbrace{f\Big(\frac{a+b}{2}\Big)}_{\text{height}} \cdot \underbrace{(b-a)}_{\text{width}}}_{\text{area of rectangle}}$$

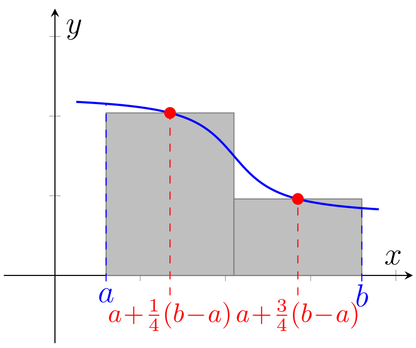
Example:

$$\int_0^1 e^{-x^2} dx \quad \approx \quad e^{-(\frac{1}{2})^2} \cdot (1-0) \quad = \quad 0.77...$$

Is this approximation any good?



For more accuracy use the midpoint rule repeatedly.



$$M_2 = f\big(a+\tfrac{1}{4}(b-a)\big)\tfrac{b-a}{2} + f\big(a+\tfrac{3}{4}(b-a)\big)\tfrac{b-a}{2}$$

$$= \tfrac{b-a}{2}\Big[f\big(a+\tfrac{1}{4}(b-a)\big) + f\big(a+\tfrac{3}{4}(b-a)\big)\Big]$$

Back to our example:

$$M_2 = \tfrac{1}{2}\left(e^{-(\frac{1}{4})^2} + e^{-(\frac{3}{4})^2}\right) \approx 0.7545$$

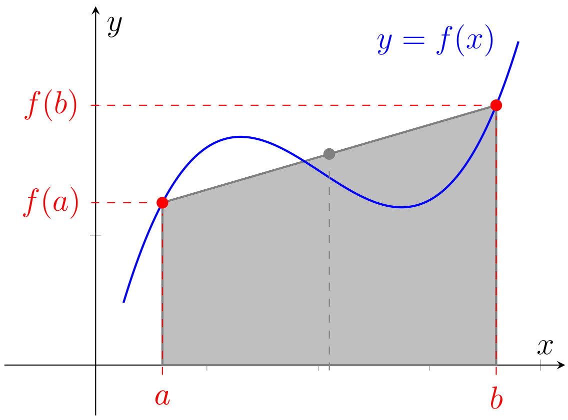
<u>*Simple approach:*</u>

Compute $M_1, M_2, M_4, M_8, \ldots$ until $M_n \approx M_{2n}$ within our desired tolerance. Then:

$$\int_a^b f(x)dx \approx M_{2n}$$

<u>*Be aware*</u>*:* With an increasing number of intervals, the summands become smaller and more susceptible to round-off errors. They may add up to create a huge uncertainty.

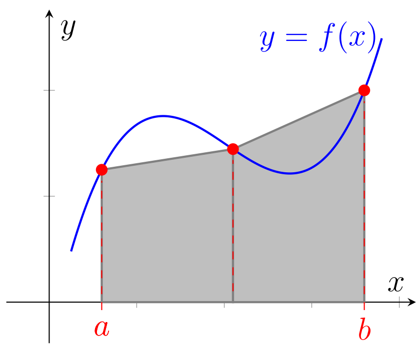
4.2 Trapezoid Rule



Use linear interpolation between the end points:

$$T_1 = \tfrac{f(a)+f(b)}{2}(b-a)$$

<u>Piecewise Linear Interpolation</u>



$$\int_a^b f(x)dx \approx T_2 = \frac{f(a) - f(\frac{a+b}{2})}{2} \cdot \frac{b-a}{2}$$

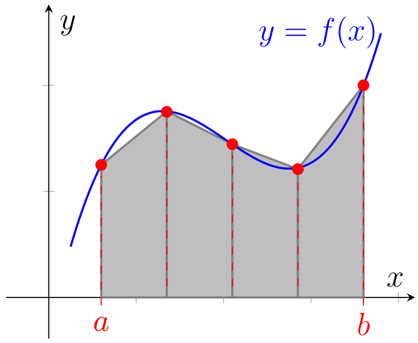
$$+ \frac{f(\frac{a+b}{2}) + f(b)}{2} \cdot \frac{b-a}{2}$$

$$= \left(\frac{f(a)}{2} + f\left(\frac{a+b}{2}\right) + \frac{f(b)}{2}\right) \cdot \frac{b-a}{2} \quad = \quad \frac{T_1 + M_1}{2}$$

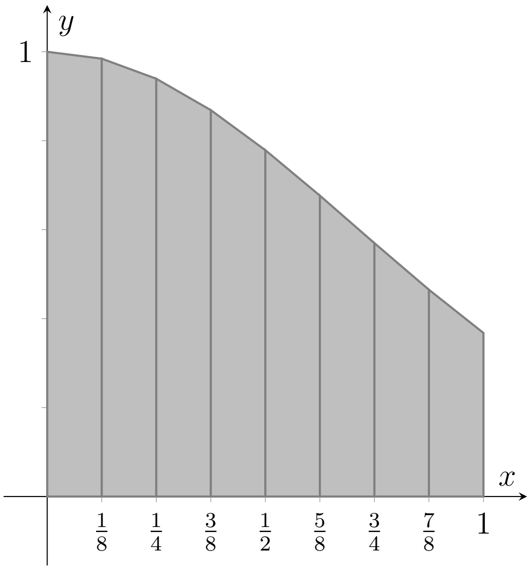
This pattern continues:

$$T_4 = \frac{T_2 + M_2}{2}$$

$$T_8 = \frac{T_4 + M_4}{2}$$



Example: $\int_0^1 e^{-x^2}dx$



$$T_1 = \frac{(e^{-0} + e^{-1}) \cdot (1-0)}{2} = 0.6839...$$

$$M_1 = e^{-(\frac{1}{2})^2} \cdot (1-0) = 0.7788...$$

$$T_2 = \frac{T_1 + M_1}{2} = 0.7313...$$

$$M_2 = \left(e^{-(\frac{1}{4})^2} + e^{-(\frac{3}{4})^2}\right) \cdot \frac{1}{2} = 0.7545...$$

$$T_4 = \frac{T_2 + M_2}{2} = 0.7429...$$

$$M_4 = \left[e^{-(\frac{1}{8})^2} + e^{-(\frac{3}{8})^2} + e^{-(\frac{5}{8})^2} + e^{-(\frac{7}{8})^2}\right] \cdot \frac{1}{4}$$

$$T_8 = \frac{T_4 + M_4}{2}$$

Note: With this procedure, each function value is only computed once. We can accelerate this process using the **Romberg scheme**. In our example, we can begin the Romberg scheme by arranging our first four T-values in a column. Beginning with T_2, we multiply our $T - value$ by 4, then subtract the preceding T-value. We then divide our result by 3.

$$T_1 = 0.6839\;3972\;06$$

$$\xrightarrow{\times(-1)} \quad \xrightarrow{\times 4} \; : 3 = 0.7471\;8042\;9$$

$$T_2 = 0.7313\;7025\;19$$

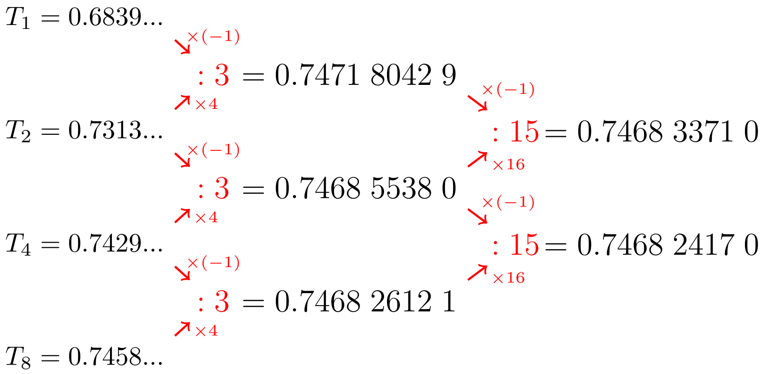
$$\xrightarrow{\times(-1)} \quad \xrightarrow{\times 4} \; : 3 = 0.7468\;5538\;0$$

$$T_4 = 0.7429\;8409\;80$$

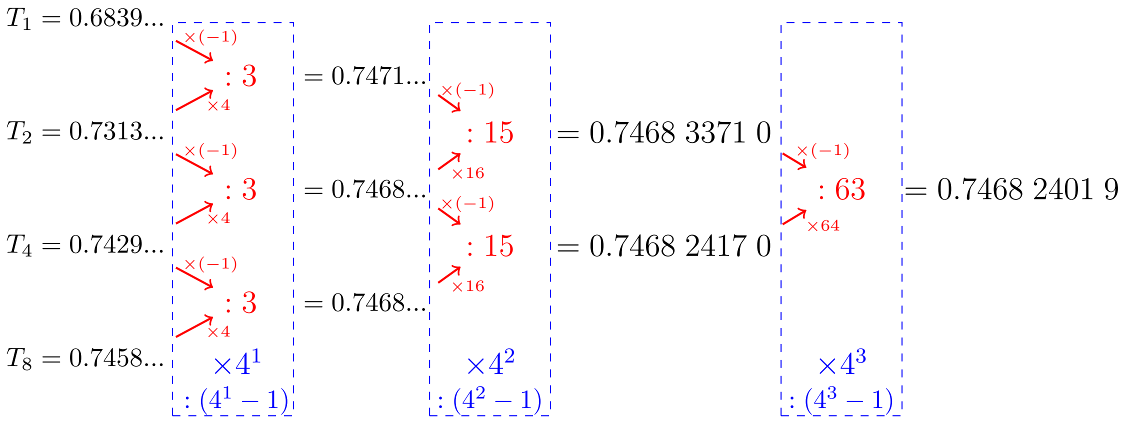
$$\xrightarrow{\times(-1)} \quad \xrightarrow{\times 4} \; : 3 = 0.7468\;2612\;1$$

$$T_8 = 0.7458\;6561\;50$$

We continue this scheme by repeating the process on our results from the second column, only this time, instead of multiplying each new term by 4 and later dividing by 3, we multiply by 16 and divide by 15. The preceding term still has a coefficient of -1.



In the third iteration of our example, we multiply our lower column value by 64, subtract the upper value, then divide the result by 63. At this point we notice the pattern that each subsequent column is multiplied by a progressively higher power of 4. In the first column we multiplied our terms by 4^1 and then divided by 3 (i.e. $4^1 - 1$). In the next column we multiply by 4^2 and divide by $4^2 - 1$.



Where do the powers of 4 come from? Using Taylor series, it can be shown that:

$$T(h) = \int_a^b f(x)dx - \underbrace{C_1 h^2}_{Dominating} - C_2 h^4 - C_3 h^6$$

$$T(\frac{1}{2}) = \int_a^b f(x)dx - \tilde{C}_1 \frac{h^2}{4} - \tilde{C}_2 \frac{h^4}{16} - \tilde{C}_3 \frac{h^6}{64} - ...$$

The smaller the h, the closer $C_1 \approx \tilde{C}_1$, $C_2 \approx \tilde{C}_2$...

$$\frac{4T(\frac{h}{2}) - T(h)}{3} = \int_a^b f(x)dx \underbrace{\qquad\qquad}_{\text{eliminates dominating term}} - \underbrace{\frac{\tilde{C}_2 \frac{h^4}{4} + C_2 h^4}{3}}_{C_2 \frac{5}{12} h^4}$$

Also worth noting is that because neighboring values are multiplied by -1 and a power of 4, we do not need to worry about round-off errors from subtracting similar values.

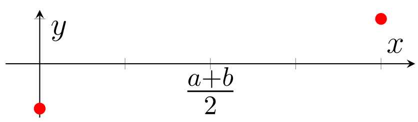
Chapter 5

Solving Non-Linear Equations

Previously:
<u>Bisection Method</u>: We want to find zeros of a function, $f(x) = 0$. We need an interval with a sign change:

$$f(a) \cdot f(b) < 0$$

f must be continuous (Intermediate Value Theorem)



Compute function values of midpoint.

$$f\left(\frac{a+b}{2}\right) = 0 \Rightarrow \text{done}$$

$$f\left(\frac{a+b}{2}\right) > 0 \Rightarrow \text{keep going}$$

$$f\left(\frac{a+b}{2}\right) < 0 \Rightarrow \text{keep going}$$

In the case that the midpoint value is greater than or less than zero, repeat the process with the sub-interval with the sign change, and repeat until the required accuracy is reached.

$$10 \text{ steps} \longrightarrow 2^{10} = 1024 \longrightarrow 3 \text{ digits}$$

Problems can occur in machine calculations, because of the gaps in the number system and because of roundoff errors in function evaluations. When "$f(x) =$ close to zero", we do not know if this is a roundoff error or really a value different from zero.

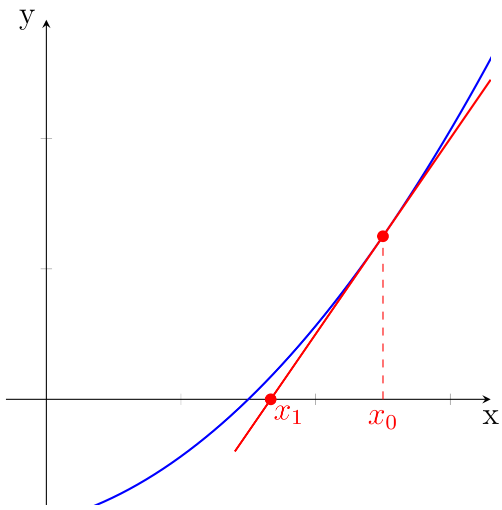
5.1 Newton's Method

$f(x) = 0$
f is differentiable
x_0 first guess

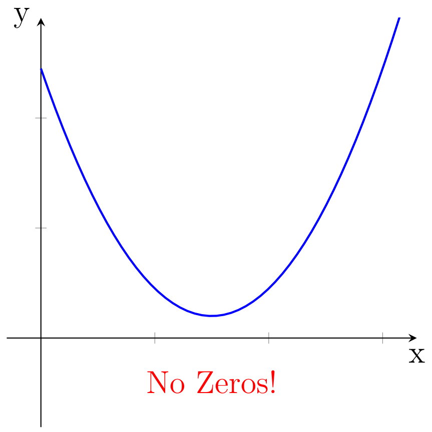
Improved guess:

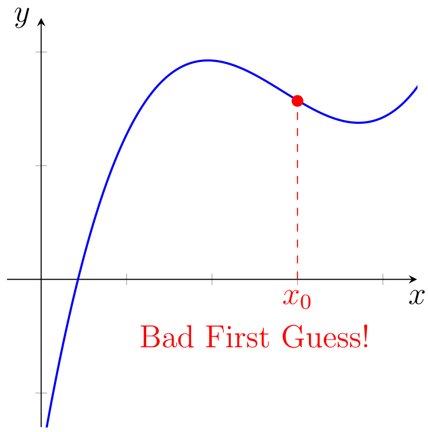
$$y = f(x_0) + f'(x_2) \cdot (x - x_0)$$

Continue until the desired accuracy is reached. (Look for some digits in successive guesses), or until it becomes obvious that the method fails.



This method can go wrong. Look at the function first!





5.2 Solving Fixed Point Equations

Another common type of equation is

$$x = g(x)$$

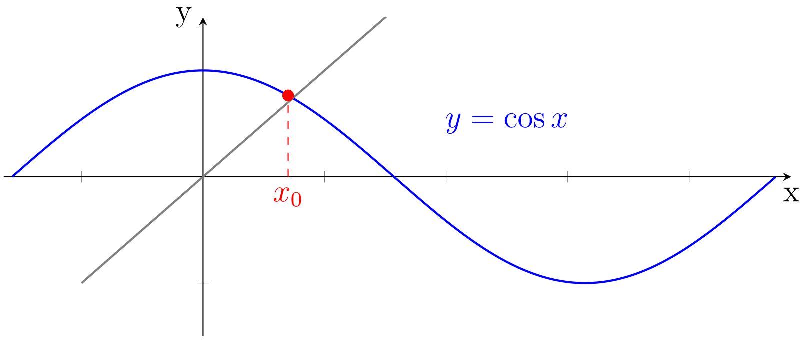
This is a fixed point equation.

$$0 = g(x) - x \;\text{ when }\; x \neq 0 \quad \Rightarrow \quad 1 = \frac{g(x)}{x} \Rightarrow 0 = \frac{g(x)}{x} - 1$$

Reflecting on Newton's Method:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad \text{when} \quad \lim_{n\to\infty} x_* \quad \text{then} \quad \underbrace{x_* = x_* - \frac{f(x_*)}{f'(x_*)}}_{g(x_*)}$$

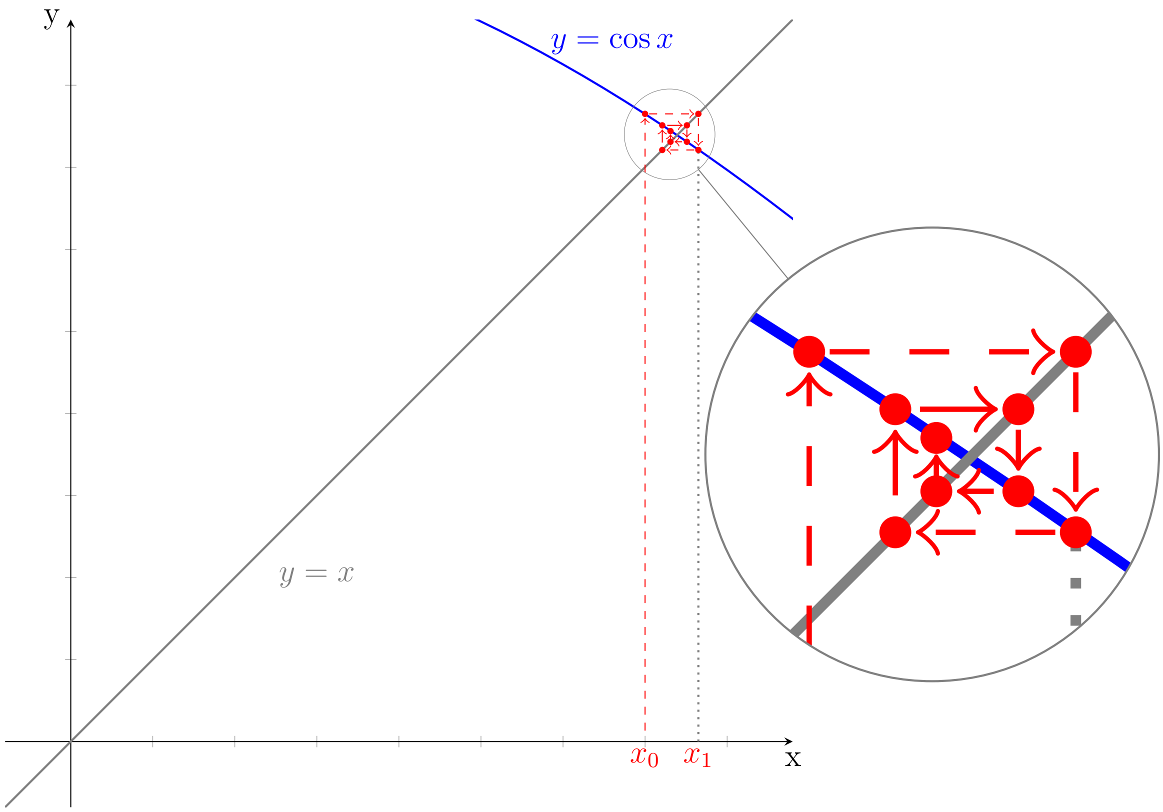
Example: $x = \cos x$



First guess: $x_0 = 0.7$. Apply Fixed Point Iteration:

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

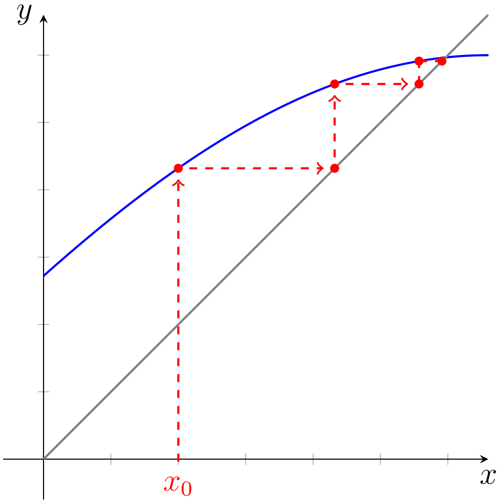
$$\cos(0.7) = 0.76484219...$$



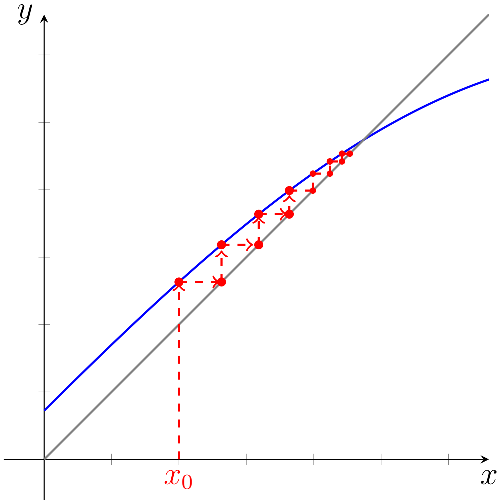
$y_1 = \cos x_0 \qquad x_1 = y_1 \qquad y_2 = \cos x_1 \qquad x_2 = y_2 \qquad y_3 = \cos x_2 \qquad x_3 = y_3$

> <u>*Fixed Point Iteration:*</u>
>
> To solve $x = g(x)$, try x_0 (first guess)
>
> $$x_{n+1} = g(x_n), n = 0, 1, 2, ...$$
>
> until the method converges or shows that it does not work.

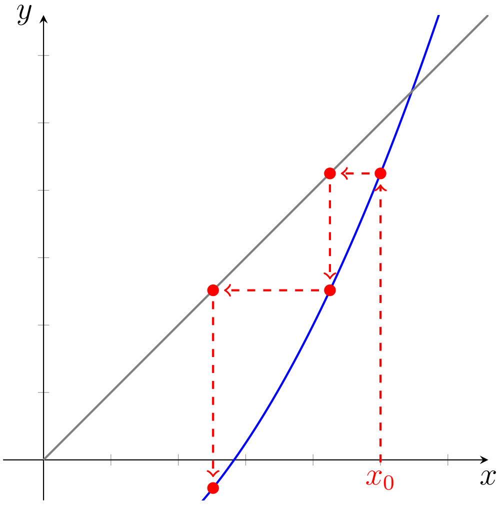
When does it work? In case it works, when is it fast or slow?



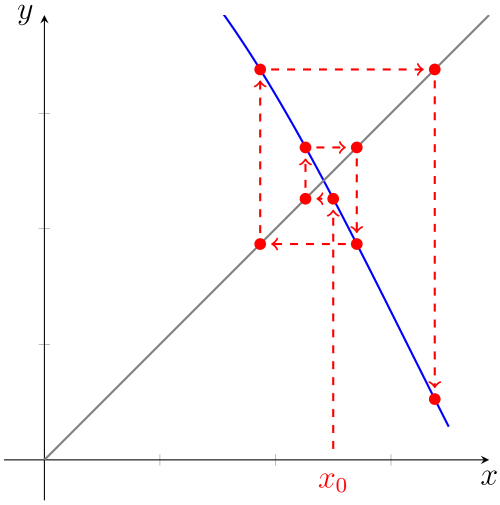
Converging "Staircase"



Slow Converging Staircase



Diverging Staircase



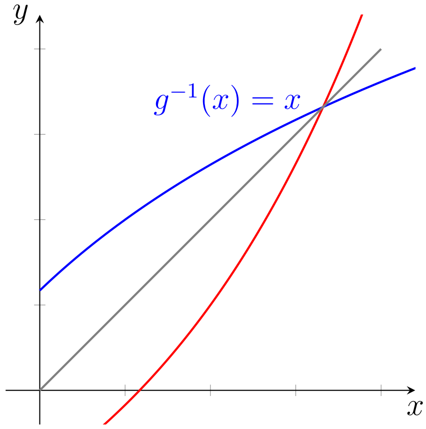
Diverging Spiral

Slope between $-1 < s < 1 \Rightarrow$ convergent

If g is differentiable, the fixed point iteration works when $|g'| < 1$ for all x in that neighborhood and if the initial guess is also in that neighborhood.

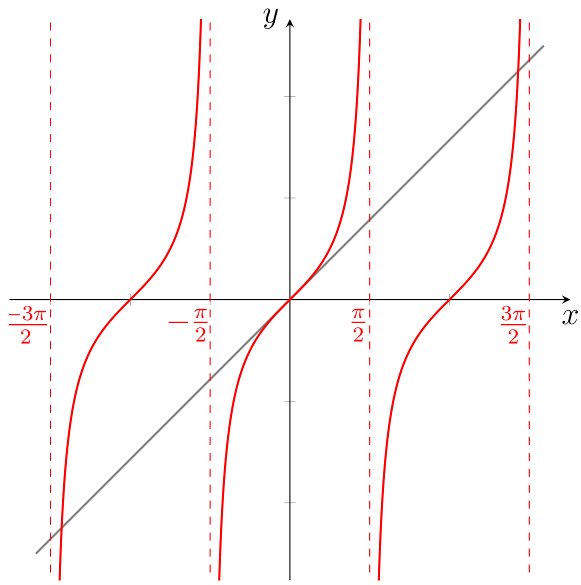
$$g' > 0 \Rightarrow \text{“Staircase”}$$
$$g' < 0 \Rightarrow \text{“Spiral”}$$

The closer $|g'|$ is to 1, the slower the convergence. In the spiral case we can use averaging to accelerate the convergence.



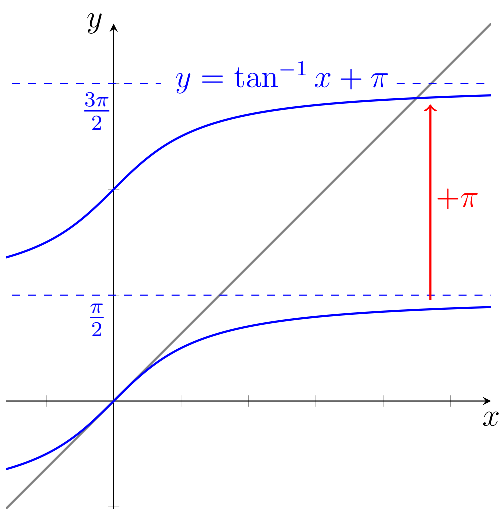
Take the inverse function.

Example: $x = \tan x$



We have infinitely many solutions. One of them is $x = 0$. The further we get from 0, the closer the value will get to the vertical asymptotes.

Let's find a solution close to $x = \frac{3\pi}{2}$. We need the inverse of the function to make fixed point iteration work.



This problem works extremely well with fixed point iteration since the slope is almost horizontal.

$$4.493 = \tan^{-1} 4.493 + \pi$$

$\longrightarrow$ applying inverse function:

$$\tan(4.493... = \tan\left(\tan^{-1}(4.493) + \cancel{\pi}\right) \quad \pi - \text{periodic}$$

$$= \tan\left(\tan^{-1}(4.493)\right)$$

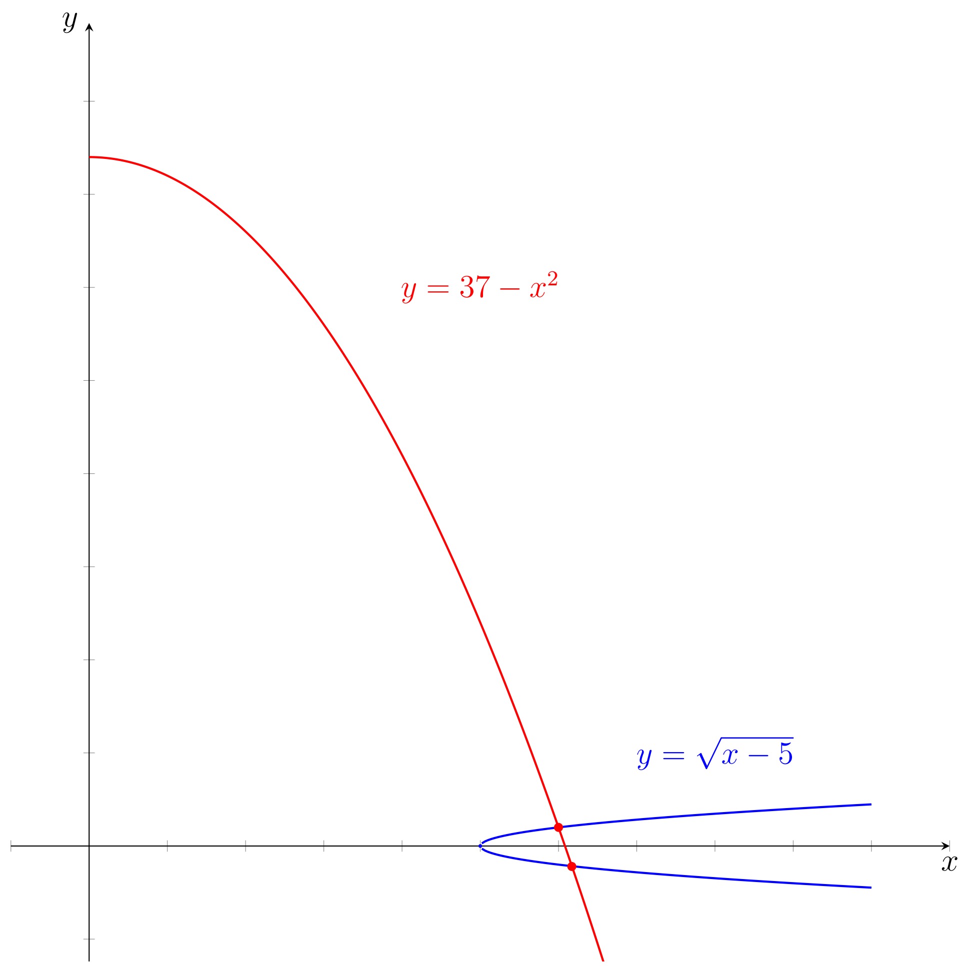
No need to go back to the original problem.

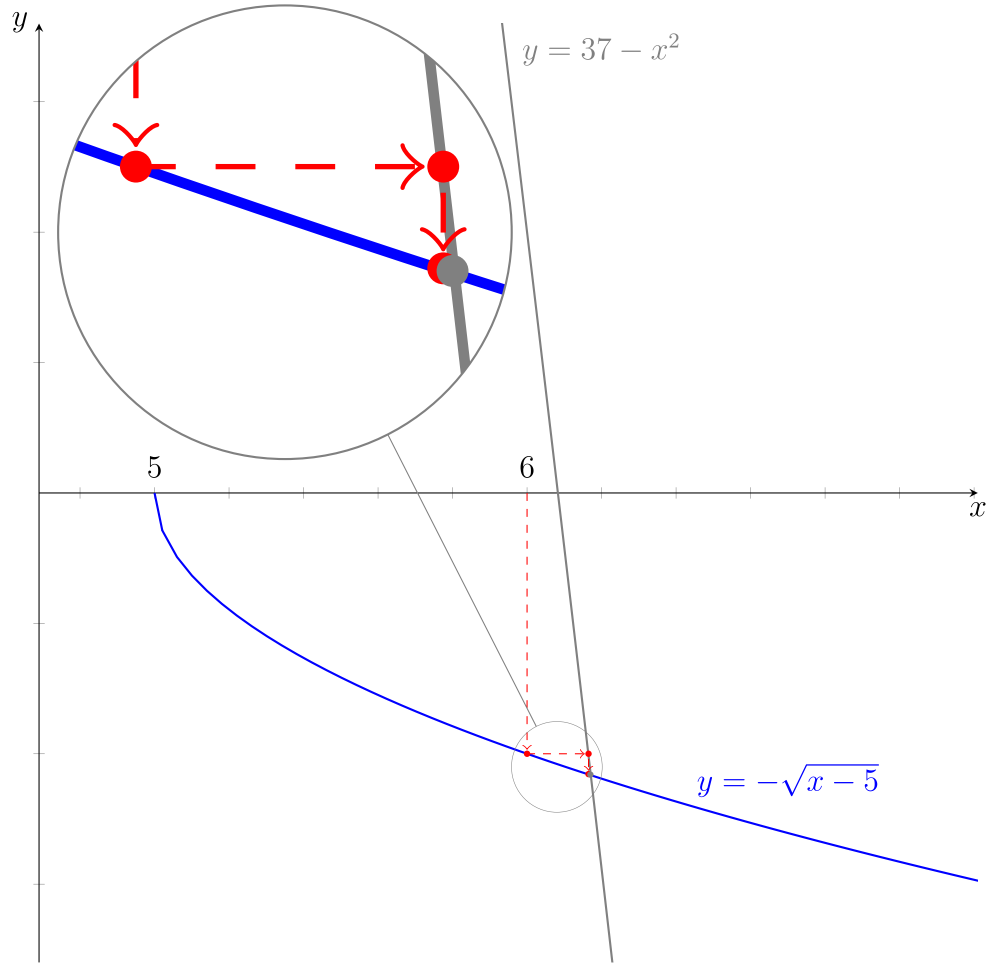
Example:

$$x^2 + y - 37 = 0$$
$$x - y^2 - 5 = 0$$
$$x + y + z - 3 = 0$$

Only the third equation involves z, so use the first and second to solve (x, y). Then plug the solutions in the corresponding z.
One possibility: We can plug $y = 37 - x^2$ into the second equation and solve

$$x - (37 - x^2)^2 = 5 \longleftarrow \text{(4th order equation)}$$





Now, develop the method:

$$-1 = y_0 = 37 - x^2$$

We need the positive solution.

$$x^2 = 37 - y_0$$

$x_1 = \sqrt{37 - y_0}$

Horizontal Step

$\Rightarrow$ Plug In $\Rightarrow$

$y_1 = -\sqrt{x_1 - 5}$

Vertical Step

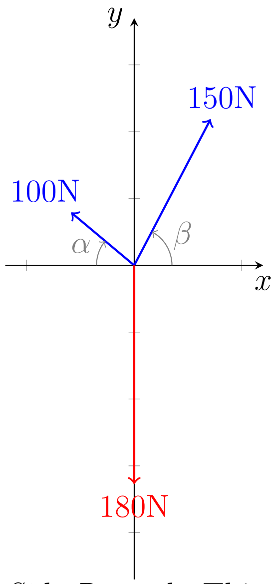
5.3 What we've learned:

- Whenever possible, *draw a picture*.
- When we draw a picture, look for simple moves from a known point, toward the solution. In particular, vertical or horizontal moves.
- Find the corresponding function evaluations that accomplish these moves.

Chapter 6

Iterative Matrix Methods

6.1 A familiar Statics problem:



What angles lead to equilibrium of the forces?

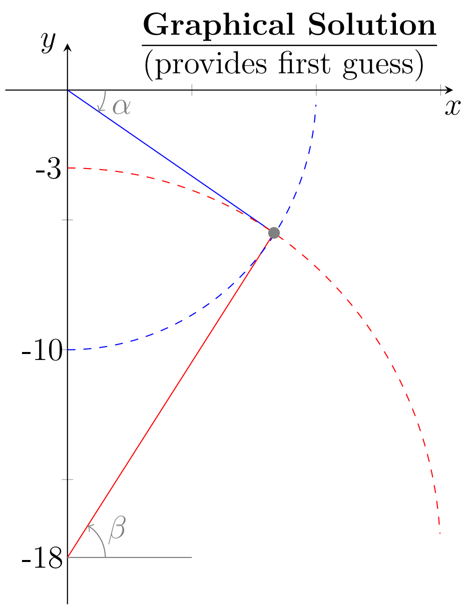
Conditions of equilibrium:

$$100\begin{pmatrix}-\cos\alpha\\ \sin\alpha\end{pmatrix}+150\begin{pmatrix}\cos\beta\\ \sin\beta\end{pmatrix}+180\begin{pmatrix}0\\ -1\end{pmatrix}=\begin{pmatrix}0\\ 0\end{pmatrix}$$

When we reduce the coefficients we get:

$$-2\cos\alpha+3\cos\beta=0$$
$$10\sin\alpha+15\sin\beta-18=0$$

Side Remark: This problem can easily be solved with the Law of Cosines, but for our purposes, we need to explore other methods.



Structure of the problem

$$\begin{cases} f_1(\alpha,\beta)=0\\ f_2(\alpha,\beta)=0\end{cases}$$

or as a vector-valued function:

$$\vec{F}(\alpha,\beta)=\vec{0}$$

where

$$\vec{F}(\alpha,\beta)=\begin{pmatrix}f_1(\alpha,\beta)\\ f_2(\alpha,\beta)\end{pmatrix}$$

This is an equation $function = zero$ as used in connection with Newton's Method.

Newton's Method for one variable:

$$x_{n+1} = x_n - \frac{f(x)_n}{f'(x)_n} \qquad\qquad n = 0, 1, 2...$$

starting from x_0 (for example, from a drawing)

Multidimensional:

$$\begin{pmatrix} \alpha_{n+1} \\ \beta_{n+1} \end{pmatrix} = \begin{pmatrix} \alpha_n \\ \beta_n \end{pmatrix} - (DF)^{-1} \begin{pmatrix} f_1(\alpha_n, \beta_n) \\ f_2(\alpha_n, \beta_n) \end{pmatrix}$$

What is the derivative of a vector-valued function, and what is its inverse? The derivative of $f_1(\alpha, \beta)$ a function of two arguments is containe in its gradient.

$$\nabla f_1(\alpha, \beta) = \underbrace{\begin{pmatrix} \frac{\partial f_1}{\partial \alpha}(\alpha, \beta) & \frac{\partial f_1}{\partial \beta}(\alpha, \beta) \end{pmatrix}}_{\text{Row Vector}}$$

$$\nabla f_2(\alpha, \beta) = \begin{pmatrix} \frac{\partial f_2}{\partial \alpha}(\alpha, \beta) & \frac{\partial f_2}{\partial \beta}(\alpha, \beta) \end{pmatrix}$$

Combine these gradients in a matrix:

$$\nabla \vec{F}(\alpha, \beta) = \begin{pmatrix} \frac{\partial f_1}{\partial \alpha}(\alpha, \beta) & \frac{\partial f_1}{\partial \beta}(\alpha, \beta) \\ \\ \frac{\partial f_2}{\partial \alpha}(\alpha, \beta) & \frac{\partial f_2}{\partial \beta}(\alpha, \beta) \end{pmatrix}$$

This can also be denoted as $D\vec{F}$, the **Jacobian Matrix**.

CAUTION: Some people use "Jacobian" to denote this Matrix, otheres use it to denote the determinant of this matrix.

Newton's Method for several variables:

$$\begin{cases} f_1(\alpha_1, ..., \alpha_n) = 0 \\ \vdots \\ f_n(\alpha_n, ..., \alpha_n) = 0 \end{cases}$$

From a first guess $(\alpha_1^{(0)}, ..., \alpha_n^{(0)})$ compute

$$\begin{pmatrix} \alpha_1^{(k+1)} \\ \vdots \\ \alpha_n^{(k+1)} \end{pmatrix} = \begin{pmatrix} \alpha_1^{(k)} \\ \vdots \\ \alpha_n^{(k)} \end{pmatrix} - \begin{pmatrix} \frac{\partial f_1}{\partial \alpha_1} & \cdots & \frac{\partial f_1}{\partial \alpha_n} \\ & \vdots & \vdots \\ \frac{\partial f_n}{\partial \alpha_1} & \cdots & \frac{\partial f_n}{\partial \alpha_n} \end{pmatrix}^{-1} \begin{pmatrix} f_1(\alpha_1^{(k)}, ..., \alpha_n^{(k)}) \\ \vdots \quad \vdots \\ f_n(\alpha_1^{(k)}, ..., \alpha_n^{(k)}) \end{pmatrix}$$

The one-dimensional Newton's Method is often distinguished from higher dimensional which are themselves referred to as *Newton - Raphson Method*.

Instead of computing the inverse matrix, it is often easier to use the equation:

$$\begin{pmatrix} \frac{\partial f_1}{\partial \alpha_1} & \cdots & \frac{\partial f_1}{\partial \alpha_n} \\ \vdots & & \vdots \\ \frac{\partial f_n}{\partial \alpha_1} & \cdots & \frac{\partial f_n}{\partial \alpha_n} \end{pmatrix} \underbrace{\begin{pmatrix} \alpha_1^{(k+1)} \\ \vdots \\ \alpha_n^{(k+1)} \end{pmatrix} - \begin{pmatrix} \alpha_1^{(k+1)} \\ \vdots \\ \alpha_n^{(k+1)} \end{pmatrix}}_{\begin{pmatrix} \delta_1^{(k)} \\ \vdots \\ \delta_n^{(k)} \end{pmatrix}} = -\begin{pmatrix} \frac{\partial f_1}{\partial \alpha_1} & \cdots & \frac{\partial f_1}{\partial \alpha_n} \\ \vdots & & \vdots \\ \frac{\partial f_n}{\partial \alpha_1} & \cdots & \frac{\partial f_n}{\partial \alpha_n} \end{pmatrix}$$

Solve for "change" using Gaussian Elimination or other method.

Then:

$$\begin{pmatrix} \alpha_1^{(k+1)} \\ \vdots \\ \alpha_n^{(k+1)} \end{pmatrix} = \begin{pmatrix} \alpha_1^{(k)} \\ \vdots \\ \alpha_n^{(k)} \end{pmatrix} + \begin{pmatrix} \delta_1^{(k)} \\ \vdots \\ \delta_n^{(k)} \end{pmatrix}$$

To reduce the computational cost of computing the Jacobian Matrix (and its inverse) in each step, one can use the same matrix for two or three steps and then update it. *This only makes sense when changes are small!*

Note: A major advantage of Newton's Method and other iterative methods is that *each step can be considered independent of all previous steps* There's no accumulation of round-off errors.

Idea: Gaussian Elimination can lead to accumulated round-off errors in the result. Use an *iterative method* to improve the round-off errors, after having performed Gaussian Elimination.

Our Example

$$f_1(\alpha, \beta) = -2\cos\alpha + 3\cos\beta$$
$$f_2(\alpha, \beta) = 10\sin\alpha + 15\sin\beta - 18$$
$$\frac{\partial f_1}{\partial \alpha} = 2\sin\alpha \qquad \frac{\partial f_1}{\partial \beta} = -\sin\beta$$
$$\frac{\partial f_2}{\partial \alpha} = 10\cos\alpha \qquad \frac{\partial f_2}{\partial \beta} = 15\cos\beta$$

Inverse of a 2×2 matrix:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} = \frac{1}{ad-bc}\begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

Hence:

$$\begin{pmatrix} 2\sin\alpha & -3\sin\beta \\ 10\cos\alpha & 15\cos\beta \end{pmatrix}^{-1} = \frac{1}{\underbrace{30\sin\alpha\cos\beta + 30\sin\beta\cos\alpha}_{=30\sin(\alpha+\beta)}}\begin{pmatrix} 15\cos\beta & 3\sin\beta \\ -10\cos\alpha & 2\sin\alpha \end{pmatrix}$$

Netwon's Method (right hand side):

$$\begin{pmatrix}\alpha\\ \beta\end{pmatrix} - \frac{1}{30\sin(\alpha+\beta)}\begin{pmatrix}15\cos\beta & 3\sin\beta\\ -10\cos\alpha & 2\sin\alpha\end{pmatrix}\begin{pmatrix}-2\cos\alpha+3\cos\beta\\ 10\sin\alpha+15\sin\beta-18\end{pmatrix}$$

$$= \begin{pmatrix}\alpha\\ \beta\end{pmatrix} - \frac{1}{30\sin(\alpha+\beta)}\begin{pmatrix}-30\cos\alpha\cos\beta+45\cos^2\beta+30\sin\alpha\sin\beta+45\sin^2\beta-54\sin\beta\\ 20\cos^2\alpha-30\cos\alpha\cos\beta+20\sin^2\alpha+30\sin\alpha\sin\beta-36\sin\alpha\end{pmatrix}$$

$$= \begin{pmatrix}\alpha\\ \beta\end{pmatrix} - \frac{1}{30\sin(\alpha+\beta)}\begin{pmatrix}-30\cos(\alpha+\beta)+45-54\sin\beta\\ 20-30\cos(\alpha+\beta)-36\sin\alpha\end{pmatrix}$$

n	α_n	β_n	$\alpha_n+\beta_n$	$\frac{45-30\cos(\alpha_n+\beta_n)-54\sin\beta_n}{30\sin(\alpha_n+\beta_n)}$	$\frac{20-30\cos(\alpha_n+\beta_n)-36\sin\alpha_n}{30\sin(\alpha_n+\beta_n)}$
0	0.59	0.99	1.58	0.004356966321	0.00823733132
1	0.5856430337	0.9817626687	1.567405702	$-5.088503697\times10^{-5}$	$-6.399002949\times10^{-6}$
2	0.5856939187	0.9817690677	1.567462986	0	$-8.630320147\times10^{-10}$
3	0.5856939187	0.9817690685	1.567462987	0	0
	$\Uparrow$ subtract	$\Uparrow$ subtract			

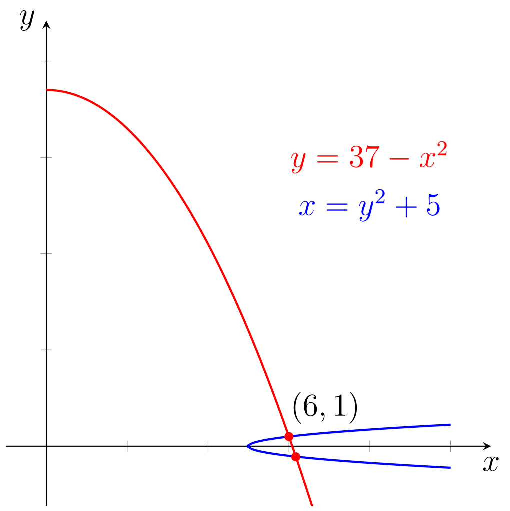
6.2 Example:

Solve

$$\begin{aligned} \textbf{I.}\quad & x^2+y-37=0\\ \textbf{II.}\quad & x-y^2-5=0\\ \textbf{III.}\quad & x+y+z-3=0\end{aligned}$$

Equation **III**: Determine z if x and y are known. Take equations **I** and **II** and solve with Newton's Method.

Step 1: Sketch all pairs (x,y) that solve equations **I** and **II**.



Starting Point for second solution

$$(x_0,y_0)=(6,-1)$$

One dimensional Newton:

$$x_{k+1}=x_k-\frac{f(x_k)}{f'(x_k)}$$

Multidimensional Newton:

$$\vec{x}_{k+1}=\vec{x}_k-DF^{-1}\cdot F(\vec{x}_n)$$

Here:

$$F(x,y)=(\underbrace{x^2+y-37}_{f_1(x,y)}\ ;\ \underbrace{x-y^2-5}_{f_2(x,y)})$$

$$DF(x,y)=\begin{pmatrix}\text{grad } f_1\\ \text{grad } f_2\end{pmatrix}=\begin{pmatrix}2x & 1\\ 1 & -2y\end{pmatrix}$$

We need

$$DF^{-1} \cdot F(x_0, y_0) = \begin{pmatrix} a_0 \\ b_0 \end{pmatrix}$$

In general, it is better to solve

$$DF \cdot \begin{pmatrix} a_0 \\ b_0 \end{pmatrix} = F(x_0, y_0)$$

Here it is practical to use the inverse, because it's a 2×2 matrix.

$$DF^{-1} = \tfrac{1}{-4xy-1} \begin{pmatrix} -2y & -1 \\ -1 & 2x \end{pmatrix}$$

So

$$DF^{-1} \cdot F(x,y) = \frac{1}{-4xy-1} \begin{pmatrix} -2y & -1 \\ -1 & 2x \end{pmatrix} \begin{pmatrix} x^2 + y - 37 \\ x - y^2 - 5 \end{pmatrix}$$

$$= \frac{1}{-4xy-1} \begin{pmatrix} -2y(x^2 + y - 37) - (x - y^2 - 5) \\ -(x^2 + y - 37) + 2x(x - y^2 - 5) \end{pmatrix}$$

$$= \frac{1}{-4xy-1} \begin{pmatrix} -2x^2y - y^2 + 74y - x + y \\ x^2 - y + 37 - 2xy^2 - 10x \end{pmatrix}$$

Step 2: Iteration

n	x_n	y_n	$\frac{-2x^2y-y^2+74y-x+5}{-4xy-1}$	$\frac{x^2-y+37-2xy^2-10x}{-4xy-1}$
0	6	-1	-0.1739130435	0.08695652174
1	6.173913043	-1.086956522	0.002836852366	-0.004783212958
2	6.171076191	-1.082173309	$1.567210481 \times 10^{-6}$	$-1.129501924 \times 10^{-5}$
3	6.171074624	-1.082162014	0	0

6.3 The Jacobian Matrix and the Chain Rule

Some connections with the Chain Rule, a special case:

$$z = f(x, y) \quad x = g(t) \quad y = (t)$$

Then:

$$z = f\big(g(t),\, h(t)\big)$$

with derivative:

$$\frac{\mathrm{d}z}{\mathrm{d}t} = \frac{\mathrm{d}}{\mathrm{d}t}\Big(f\big(g(t),\, h(t)\big)\Big) = \frac{\partial f}{\partial x}\big(g(t),\, h(t)\big) \cdot \frac{\mathrm{d}}{\mathrm{d}t} g(t) + \frac{\partial f}{\partial y}\big(g(t),\, h(t)\big) \cdot \frac{\mathrm{d}}{\mathrm{d}t} h(t)$$

with matrices:

$$Df\big(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\big) \qquad \Leftarrow \text{Gradient is a row vector!}$$

$$G(t) = \begin{pmatrix} g(t) \\ h(t) \end{pmatrix} \qquad \Leftarrow \text{vector valued function}$$

$$DG = \begin{pmatrix} g'(t) \\ h'(t) \end{pmatrix} \quad \Leftarrow \text{component-wise differentiation}$$

$$f\big(g(t),\, h(t)\big) = \big(f \circ G\big)(t) \Leftarrow \text{composition of functions}$$

$$D\big(f \circ G\big) = Df \cdot DG = \left(\tfrac{\partial f}{\partial x},\, \tfrac{\partial f}{\partial y}\right) \cdot \begin{pmatrix} g'(t) \\ h'(t) \end{pmatrix} = \tfrac{\partial f}{\partial x} \cdot g'(t) + \tfrac{\partial f}{\partial y} \cdot h'(t)$$

Generalized Chain Rule:

$$\begin{pmatrix} f_1(x_1, ..., x_n) \\ \vdots \\ f_k(x_1, ..., x_n) \end{pmatrix} = F(x_1, ..., x_n)$$

$$G(y_1, ..., y_r) = \begin{pmatrix} g_1(y_1, ..., y_r) \\ \vdots \\ g_n(y_1, ..., y_r) \end{pmatrix}$$

Composition:

$$F \circ G(y_1, ..., y_r) = \begin{pmatrix} f_1\big(g_1(y_1, ..., y_r) ... g_n(y_1, ..., y_r)\big) \\ \vdots \\ f_k\big(g_1(y_1, ..., y_r) ... g_n(y_1, ..., y_r)\big) \end{pmatrix}$$

$$D\big(F \circ G\big) = \underbrace{\begin{pmatrix} \frac{\partial f_1}{\partial y_1} \cdots \frac{\partial f_1}{\partial y_r} \\ \vdots \quad\quad \vdots \\ \frac{\partial f_k}{\partial y_1} \cdots \frac{\partial f_k}{\partial y_r} \end{pmatrix}}_{k \times r}$$

or via chain rule...

$$DF \cdot DG = \underbrace{\begin{pmatrix} \frac{\partial f_1}{\partial x_1} \cdots \frac{\partial f_1}{\partial x_n} \\ \vdots \quad\quad \vdots \\ \frac{\partial f_k}{\partial x_1} \cdots \frac{\partial f_k}{\partial x_n} \end{pmatrix}}_{k \times n} \underbrace{\begin{pmatrix} \frac{\partial g_1}{\partial y_1} \cdots \frac{\partial g_1}{\partial y_r} \\ \vdots \quad\quad \vdots \\ \frac{\partial g_n}{\partial y_1} \cdots \frac{\partial g_n}{\partial y_r} \end{pmatrix}}_{n \times r}$$

$$\boxed{\frac{\partial f_i}{\partial y_j} = \frac{\partial f_i}{\partial x_1} \cdot \frac{\partial g_1}{\partial y_j} + \frac{\partial f_i}{\partial x_2} \cdot \frac{\partial g_2}{\partial y_j} + ... + \frac{\partial f_i}{\partial x_n} \cdot \frac{\partial g_n}{\partial y_j}}$$

Chapter 7

Numerical Solution to an Initial Value Problem

Explicit First-Order Ordinary Differential Equation:

$$\left.\begin{array}{l} y' = f(x, y) \\ y(x_0) = y_0 \end{array}\right\} \text{Initial Value Problem (IVP)}$$

Example:

$$y' = x + y \qquad y(0) = 0 \qquad \Leftrightarrow \qquad y' - y = x$$

Linear differential equation, first order, with constant coefficients.

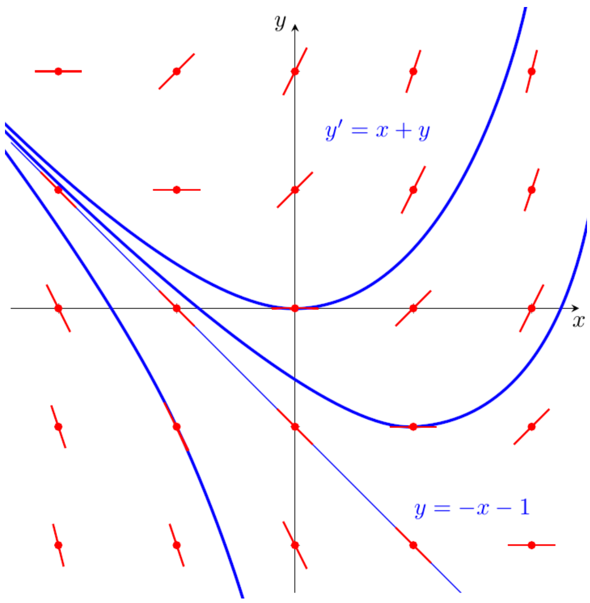
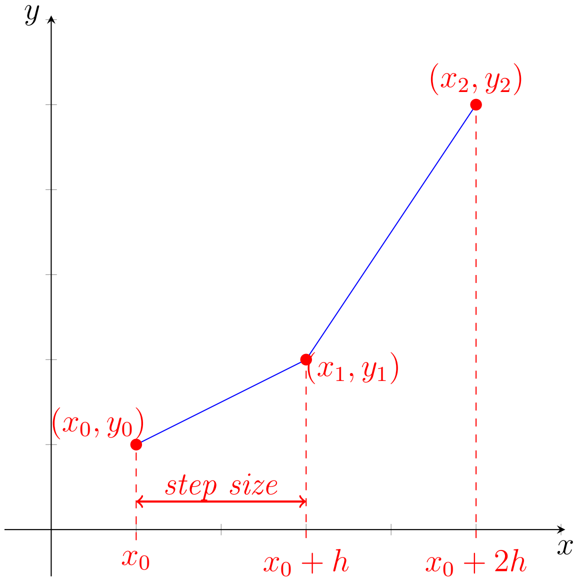


Figure 7.1: Visualization: Direction Field

A very important and simple method for practical purposes is to draw a direction field and "look" for solutions. How can we translate this into a computational method?

7.1 Euler's Method

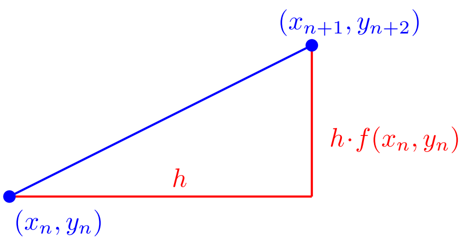


Euler's Method
(Forward Euler), (Explicit Euler)

Construct $(x_0, y_0)(x_1, y_1)(x_2, y_2)$... as approximations to the wanted function (x, y(x)).
Start at initial point (x_0, y_0). Use $y' = f(x, y)$ to compute the slope at this point and follow the tangent line for a horizontal step size h.

$$x_{n+1} = x_n + h$$

$$y_{n+1} = y_n + h \cdot f(x_n, y_n)$$



Back to our example:

$$y' = x + y \qquad y(0) = 0$$

Find solution on interval [0,2]

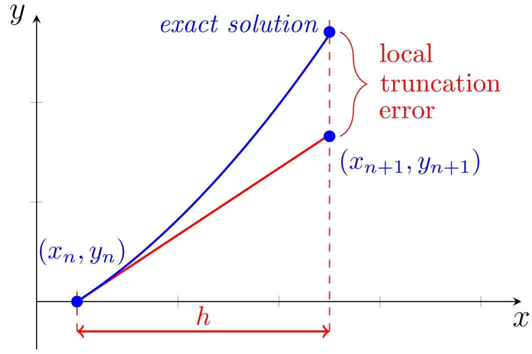
Naive method: Halve the step size until two successive step sizes agree within the desired tolerance.

Possible problems: Smaller step sizes may lead to more severe round-off errors, especially since more steps are needed.

How accurate is Euler's Method? First consider its accuracy for one step:

With $y(x_n) = y_n$

Assumption: We start at a correct point.



$$\text{error} \quad = \quad y(x_{n+1}) - y_{n+1}$$

$$= y(x_n) + y'(x_n)\big(x_{n+1} - x_n\big) + \frac{y''}{2}\big(x_{n+1} - x_n\big)^2 + \ldots - y_{n+1}$$

$$= \underbrace{y_n + f(x_n, y_n) \cdot h}_{y_{n+1}} + \frac{y''}{2}\big(x_{n+1} - x_n\big)^2$$

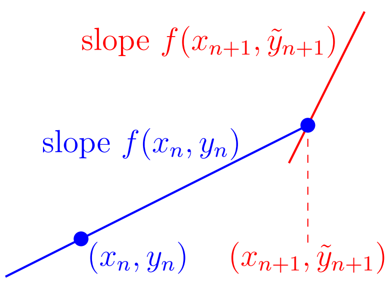
$$= \frac{y''(x_n)}{2} h^2 + \text{...higher order terms}$$

What is the error when we have to take several steps? (interval [a,b]) Step size $h = \frac{b-a}{n}$, n = number of steps. A rough estimate of the total error at b is:

$$C_1 h^2 + C_2 h^2 + \ldots C_n H^2 \approx n \cdot C h^2 = \frac{b-a}{h} \cdot C \cdot h^2 = (b-a) C \cdot h$$

Euler's method is a *first-order* method when we have the step-size we expect to halve the truncation error (until round-off errors take over). When round-off errors take over too soon, we need a higher order method. For example:

7.2 Heun's Method



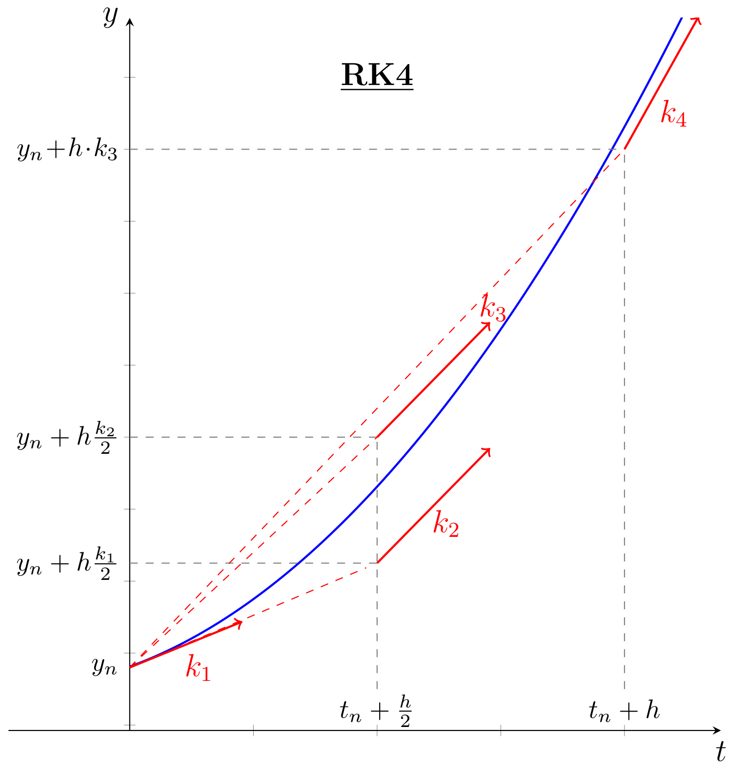
average slope:

$$x_{n+1} = x + h$$

$$\tilde{y}_{n+1} = y_n + h\big(f(x_n, y_n)\big)$$

$$y_{n+1} = y_n + h \frac{f(x_n, y_n) + f(x_{n+1}, \tilde{y}_{n+1})}{2}$$

This is a second order method. Among those single step methods, the most popular for engineers is the classical **Runge-Kutta** method.



$$k_1 = hf(t_n, y_n) \qquad k_2 = hf(t_n + \frac{h}{2}, y_n\frac{k_1}{2})$$

$$k_3 = hf(t_n + \frac{h}{2}, y_n\frac{k_2}{2} \qquad k_4 = hf(t_n + h, y_n + k_3)$$

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 3k_3 + k_4)$$

7.3 Limitations of Single-Step Methods

So far, we have seen so-called "single-step methods" to compute approximate solutions for differential equations using the information $y' = f(x, y)$ and (x_n, y_n), these methods compute (x_{n+1}, y_{n+1}). There are some problems.

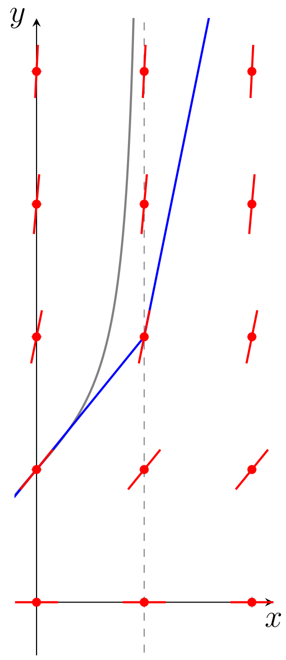
Example:

$$y' = y^2 \qquad\qquad y(0) = 1$$

Use separation of variables to obtain the exact solution:

$$y(x) = \frac{-1}{x-1} = \frac{1}{1-x}$$

There is a vertical asymptote at $x = 1$.

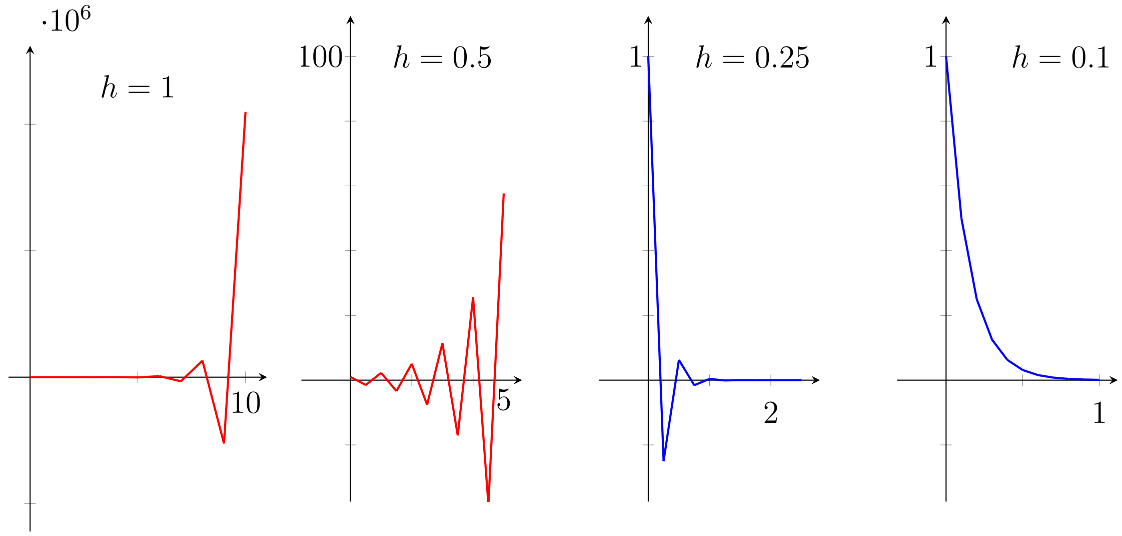


Euler's Method: See picture.
Single-step methods cannot detect vertical asymptotes. Asymptotes are *very* important in engineering. We don't have a solution for this quite yet.

7.4 Stability Analysis

Example: $y' = -5y \qquad y(0) = 1$
Using forward Euler we get:

	$h = 1$		$h = 0.5$		$h = 0.25$		$h = 0.1$	
n	x_n	y_n	x_n	y_n	x_n	y_n	x_n	y_n
0	0	1	0	1	0	1	0	1
1	1	-4	0.5	-1.5	0.25	-0.25	0.1	0.5
2	2	16	1	2.25	0.5	0.0625	0.2	0.25
3	3	-64	1.5	-3.375	0.75	-0.015625	0.3	0.125
4	4	256	2	5.0625	1	0.00390625	0.4	0.0625
5	5	-1024	2.5	-7.59375	1.25	-0.0009766	0.5	0.03125
6	6	4096	3	11.390625	1.5	0.00024414	0.6	0.015625
7	7	-16384	3.5	-17.085938	1.75	-6.1035×10^{-5}	0.7	0.0078125
8	8	65536	4	25.6289063	2	1.5259×10^{-5}	0.8	0.00390625
9	9	-262144	4.5	-38.443359	2.25	-3.8147×10^{-6}	0.9	0.00195313
10	10	1048576	5	57.6650391	2.5	9.5367×10^{-7}	1	0.00097656



<u>Exact Solution:</u> $\quad y(x) = C \cdot e^{-5x} \qquad y(x) = e^{-5x}$ for IVP.

We observe that the shape of the numerical solution using Euler's method varies extremely depending on the step size.

$h = 1$: oscillating behavior with fast increasing amplitude. (error becomes infinitely large)

$h = 0.5$: Same only slower increase

$h = 0.25$: Still oscillating, but decreasing amplitude. (Error stays bounded)

$h = 0.1$: Same shape as the actual solution.

It is possible that the true solution is "stable" (i.e. bounded from above and below) but the numerical solution is "unstable" (i.e. unbounded). Then the error gets arbitrarily large.

Euler's method is conditionally stable: i.e. the numerical solution is stable for sufficiently small step sizes but may be unstable for larger step sizes.

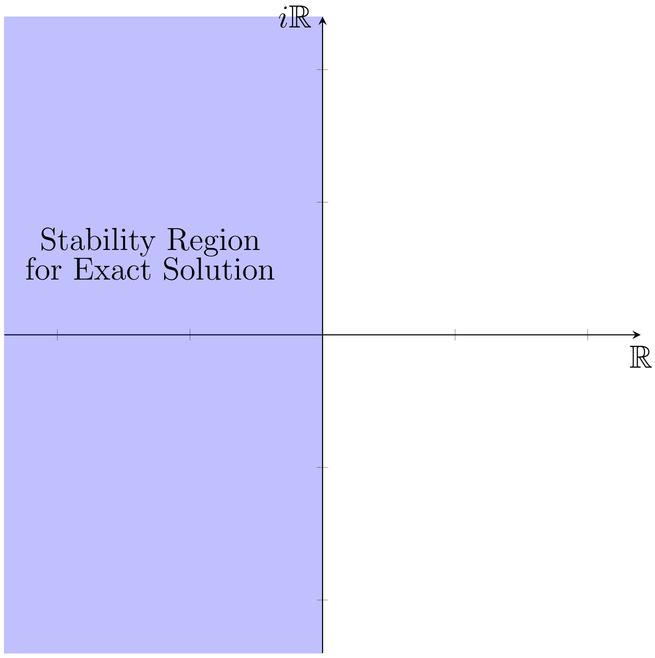
How can we determine a suitable step size? We apply the method under investigation to the model equation:

$$y' = \lambda y$$

λ parameter may be complex (i.e. $\lambda = a + ib$).

Example:

$$\begin{aligned} y(x) &= e^{\lambda x} = e^{(a+ib)x} = e^{ax} \cdot e^{ibx} \\ &= \overbrace{e^{ax}}^{\text{bounded when } a \le 0} \cdot \underbrace{\left(\cos bx + i \sin bx\right)}_{\text{bounded}} \end{aligned}$$



Now apply Euler's Method:

$$\begin{aligned} y_{n+1} &= y_n + h \cdot f(x_n, y_n) \\ &= y_n + h \cdot \lambda y_n \\ &= \left(1 + \lambda h\right) y_n \\ &= \underbrace{\left(1 + h \cdot \lambda\right)^{n+1}}_{\text{Amplification Factor}} \cdot y_0 \end{aligned}$$

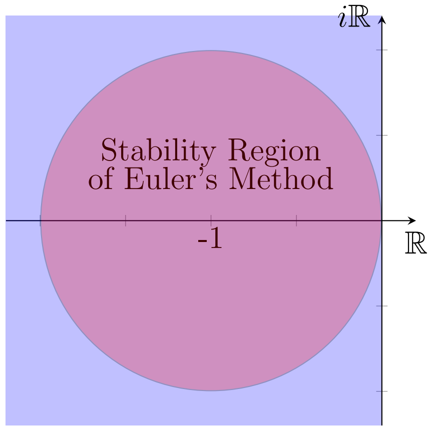
This would be stable when

$$|1 + h\lambda| \le 1$$

$$|1 + ha + hib| \leq 1$$

$$\Rightarrow \sqrt{(1+ha)^2 + (hb)^2} \leq 1$$

$$\Rightarrow (1+ha)^2 + (hb)^2 \leq 1^2$$



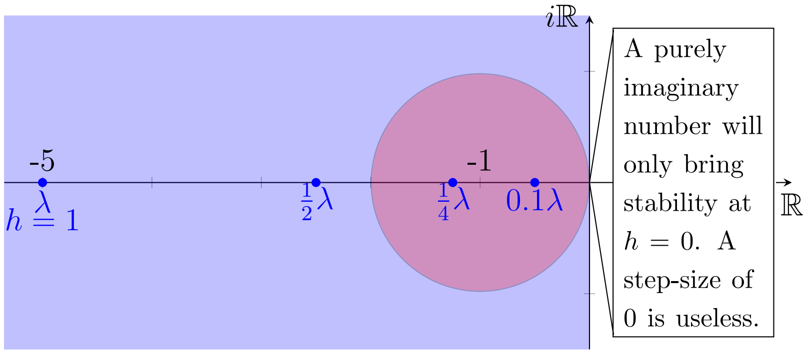
Describes a circle (with interior i.e. "disk")

In our example:

$$y' = -5 \qquad\qquad y(0) = 1$$

The exact solution is stable when

$$\text{Real}(\lambda) = -5 \leq 0$$



Amplification Factor:

$$|1 + h\lambda| \leq 1 \qquad \Rightarrow \qquad |1 - 5h| \leq 1$$

$$\Rightarrow \big(1 - 5h\big)^2 \leq 1$$

$$\Rightarrow 1 - 10h + 25h^2 \leq 1$$

$$\Rightarrow 25h^2 - 10h \leq 0$$

$$\Rightarrow 5h\big(5h - 2\big) \leq 0$$

	$5h$	$5h-2$	$5h(5h-2)$
$-\infty, 0$	-	-	$+$
$0, \frac{2}{5}$	$+$	-	-
$\frac{2}{5}, \infty$	$+$	$+$	$+$

Inequality is satisfied when:

$$h \in [0, \frac{2}{5}]$$

Note that for $\lambda = ib$ (purely imaginary parameter) we cannot find any step size for which Euler's Method is stable. For Real$(\lambda)a < 0$, we can find sufficiently small step-size for Euler's Method to remain stable. A step-size $h = 0$ doesn't make sense, though.

Other methods have other stability regions. Let's consider Euler's Method from another perspective:

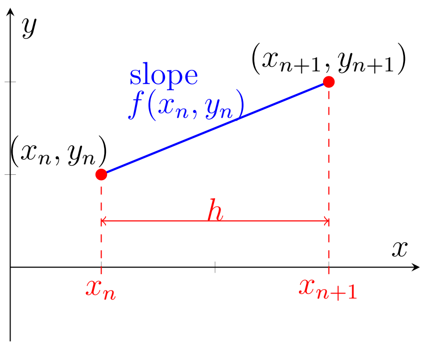
$$y_{n+1} = y_n + h \cdot f(x_n, y_n)$$

$$\Rightarrow \underbrace{\frac{y_{n+1} - y_n}{h}}_{\text{Forward Difference}} = f(x_n, y_n)$$

Compare with:

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

Here the derivative is replaced by the **forward difference formula**:



Let's use the backward difference instead:

$$\frac{y_{n+1} - y_n}{h} = \underbrace{f(x_{n+1}, y_{n+1})}_{\text{The change is here.}}$$

$$\Rightarrow \underbrace{y_{n+1} = y_n + hf(x_{n+1}, y_{n+1})}_{\text{Implicit equation for unknown } y_{n+1}}$$

This is referred to as the **Implicit Euler** or **Backward Euler** method. Conversely, the method that we've previously studied is referred to as the *Explicit Euler* or *Forward Euler* method.

The Implicit Euler step is an equation in fixed point form. We can apply fixed point iteration to solve it. For sufficiently small step size h, the necessary condition for fixed point iteration is satisfied. Use Explicit Euler to obtain a first guess. If it deviates a lot from the result of Implicit Euler, use a smaller step size.

Example:

$$y' = \sin y \qquad\qquad y(0) = 1$$

7.5 Stability Analysis for Implicit Euler

The example shows that each step requires significantly more work. So what's the advantage?

Model Problem:

$$y' = \lambda \cdot y \qquad \lambda = a + ib$$

The exact solution is stable when $a \leq 0$. Apply Implicit Euler:

$$y_n + 1 = y_n + h \cdot f(x_{n+1}, y_{n+1}) = y_n + h \cdot \lambda \cdot y_{n+1}$$

$$\Rightarrow y_{n+1} - h \cdot \lambda \cdot y_{n+1} = y_n$$

$$\Rightarrow y_{n+1}\big(1 - h\lambda\big) = y_n$$

$$\Rightarrow y_{n+1} = \underbrace{\frac{1}{1 - h\lambda}}_{\text{Amplification factor}} y_n$$

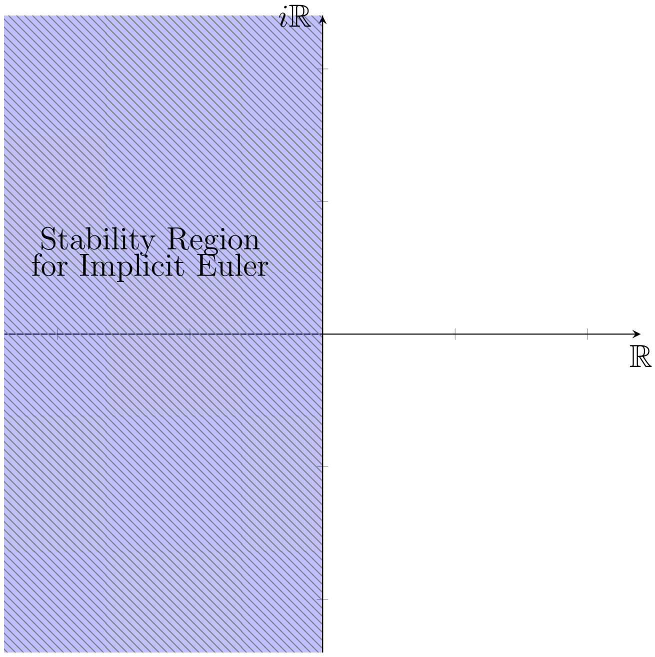
For stability:

$$|\frac{1}{1 - h\lambda}| \leq 1$$

$$\Rightarrow 1 \leq |1 - h\lambda|$$

$$\Rightarrow 1 \leq |1 - h \cdot a - i \cdot h \cdot b|$$

$$\Rightarrow 1 \leq \underbrace{\sqrt{\big(1 - ha\big)^2 + \big(hb\big)^2}}_{>1 \text{ when } \lambda \neq 0}$$



The extra computational effort affords us nice stability behavior.

Example: (revisited)

$$y' = -5y \qquad\qquad y(0) = 1$$

Implicit Euler: $h = 1$

$$y_1 = y_0 + h \cdot f(x_1, y_1)$$

$$\Rightarrow y_1 = 1 + 1 \cdot (-5)y_1$$

$$\Rightarrow 6y_1 = 1 \Rightarrow y_1 = \frac{1}{6}$$

$$y_2 = y_1 + h \cdot f(x_2, y_2)$$

$$\Rightarrow y_2 = \frac{1}{6} + 1(-5)y_2$$

$$\Rightarrow 6y_2 = \frac{1}{6}$$

$$\Rightarrow y_2 = \frac{1}{36}$$

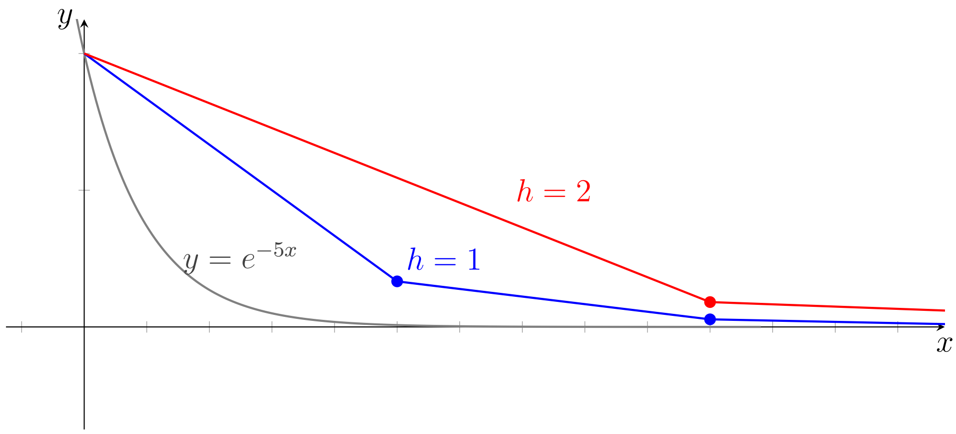
Even a larger step-size works: $h = 2$

$$y_1 = y_0 + 2(-5)y_1$$

$$\Rightarrow 11y_1 = 1$$

$$\Rightarrow y_1 = \frac{1}{11}$$

This step size isn't very accurate, but is ultimately still stable.

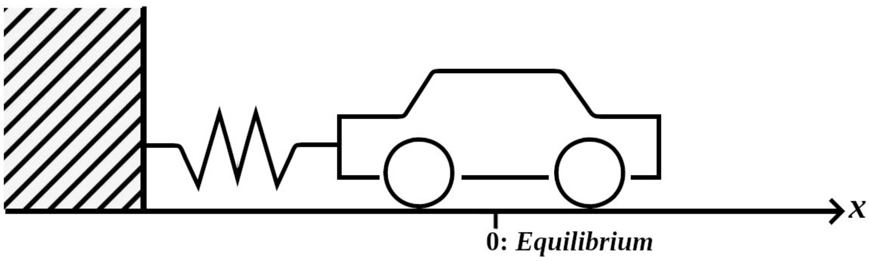


Observation:
Explicit Euler always uses the same slope regardless of the step-size. Implicit Euler adjusts the slope to the step-size because the slope depends on the final point.

Chapter 8

Numerical Solutions to Differential Equations

Example:



Point Mass:
$x(t)$ position of particle at time t.

Restoring Force:
Here visualized by a spring.

Hooke's Law:

$$F_s = -k \cdot x$$

$k > 0$ Spring Constant

Newton's Law:

$$F = m \cdot a = m\ddot{x}$$

Thus:

Equation of Motion:

$$m\ddot{x} = -kx$$

$$\Rightarrow m\ddot{x} + kx = 0$$

Second-order Linear Differential Equation with constant coefficient, homogeneous. Choose units so that $m = 1, \ k = 1$

$$\Rightarrow \ddot{x} + x = 0$$

Initial conditions:

$$x(0) = 1 \qquad\qquad \dot{x}(0) = 0$$

Exact solution:

$$x(t) = \cos(t) \qquad \text{(Review Applied Math)}$$

So far, our numerical methods considered a first-order ODE:

$$y' f(x, y)$$

How do we adapt them to the second-order setting?

Introduce the first derivative of the wanted function as a new component function:

$$\begin{array}{rcccl} v & = & \dot{x} & & \\ \dot{v} & = & \ddot{x} & = & -x \end{array}$$

We obtain a system of first-order ordinary differential equations:

$$\begin{cases} \dot{x} = v \\ \dot{v} = -x \end{cases}$$

$$\frac{\mathrm{d}}{\mathrm{d}t} \begin{pmatrix} x \\ v \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \begin{pmatrix} x \\ v \end{pmatrix}$$

This particular system is even linear. (Matrix-vector Product)
Compare:

$$\begin{array}{rcl} y\prime & = & f(x, y) \\ y_{n+1} & = & y_n + h \cdot f(x_n, y_n) \end{array}$$

Step size: h

Just use one computation step for each component of the system.

Explicit Euler:

$$t_{n+1} = t_n + h$$

$$x_{n+1} = x_n + h \cdot v_n$$

$$v_{n+1} = v_n + h \cdot (-x_n)$$

$$\frac{x_{n+1} - x_n}{h} = v_n \qquad\qquad \frac{v_{n+1} - v_n}{h} = -x_n$$

8.1 Energy Example

Some Physics:

$$\begin{aligned}\text{total energy} &= \text{potential energy} + \text{kinetic energy}\\ &= \frac{1}{2}kx^2 + \frac{1}{2}mv^2\end{aligned}$$

With our units:

$$2 \cdot \text{energy} = x^2 + v^2$$

This quantity is constant in our example.

Exact solution:

$$x^2 + v^2 = \cos^2 t + \sin^2 t = 1$$

Yes, constant energy.

Numerical Solution:

$$\begin{aligned}&{x_{n+1}}^2 + {v_{n+1}}^2\\ &= (x_n + hv_n)^2 + (v_n - hx_n)^2\\ &= {x_n}^2 + \cancel{2hx_nv_n} + h^2{v_n}^2 + {v_n}^2 - \cancel{2hx_nv_n} + h^2{x_n}^2\\ &= (1+h^2){x_n}^2 + (1+h^2){v_n}^2\\ &(1+h^2)({x_n}^2 + {v_n}^2)\end{aligned}$$

"Energy" is increasing. The numerical method is unstable in this case. To avoid an increase of the energy, try implicit Euler. Yes it is stable, but the energy will decrease for implicit Euler and tend to 0.

There are more advanced methods that "keep" the energy at a better level. Sadly, we won't be covering them here.

Example:
Some motion in the xy plane is described by:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{pmatrix} -50 & 49 \\ 49 & -50 \end{pmatrix}\begin{pmatrix} x(t) \\ v(t) \end{pmatrix}$$

With initial conditions:

$$\begin{pmatrix} x(0) \\ y(0) \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \end{pmatrix}$$

How do we solve such a system exactly?

1.) Compute the eigenvalues of the matrix:

$$\det\begin{pmatrix} -50-\lambda & 49 \\ 49 & -50-\lambda \end{pmatrix} = 0$$

$$\Rightarrow (-50 - \lambda)^2 - 49^2 = 0$$
$$\Rightarrow \lambda^2 + 100\lambda + 50^2 - 49^2 = 0$$
$$\Rightarrow \lambda^2 + 100\lambda + (50 - 49)(50 + 49) = 0$$
$$\Rightarrow \lambda^2 + 100\lambda + 99 = 0$$

Quadratic Formula / or Simpler

$$(\lambda + 99)(\lambda + 1) = 0$$

Eigenvalues are $\lambda_1 = 1$ and $\lambda_2 = -99$

2.) Compute Eigenvectors:

$$\lambda = -1$$

x	y	r.h.s.
-49	49	0
49	-49	0
-49	49	0
0	0	0

$$\lambda = -99$$