Scientific Computing

Maastricht Science Program

Week 1

Frans Oliehoek
<frans.oliehoek@maastrichtuniversity.nl>

Good Choice!

- Let me start: Congratulations!
- There is virtually no branch of science that can do without scientific computations...
- Exact science require a way of thinking that is closely linked with math and programming

- But also: bear with me!
  - There is a **lot** to be learned.
  - Different backgrounds.
- But don't worry: programming is not difficult.

Practicalities

```
      Name: Frans Oliehoek
Department: DKE (RAI group)
  Location: SSK 39, room 2.001
      Tel.: +31 43 3883485
     Email: frans.oliehoek@maastrichtuniversity.nl
       WWW: http://people.csail.mit.edu/fao/
```

- About me
  - Computer Science / AI
  - First time I give this course

    → let me know if things are unclear!

- Book “QSG”:
  - *Scientific Computing with MATLAB and Octave*. Alfio Quarteroni, Fausto Saleri & Paola Gervasio. 3rd edition.
- Course manual on Eleum and my website.
- All information will be posted on my website under 'teaching':

  http://people.csail.mit.edu/fao/

Practicalities


```
      Name: Frans Oliehoek
Department: DKE (RAI group)
  Location: SSK 39, room 2.001
     Tel.: +31 43 3883485
    Email: frans.oliehoek@maastrichtuniversity.nl
      WWW: http://people.csail.mit.edu/fao/
```


- Examination etc.
- Attendance
- Grades based on:
  - A small report at the end of each lab
  - A (short) closed book test during the last session
- Work in pairs
  - linear algebra students not together

More Practicalities

- Schedule:

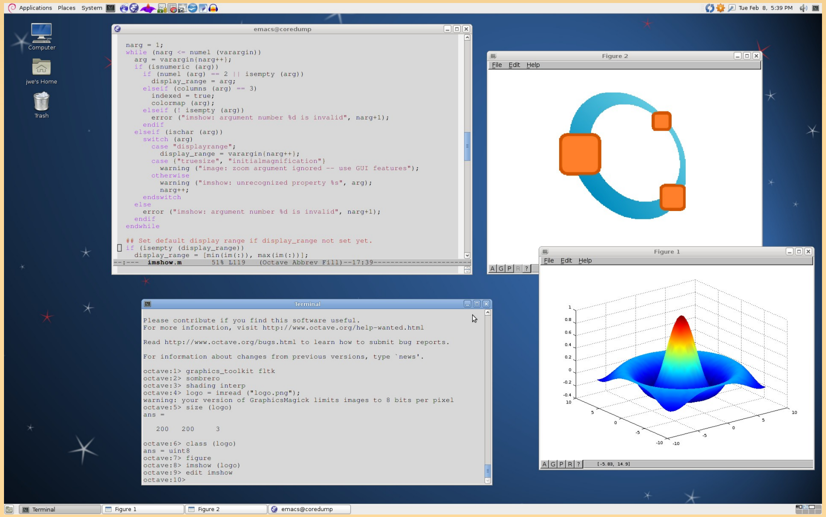
Session	Date	hours / location
1	2012-04-13 (Fri)	-MSC Lecture Hall **1.028** 0900-1100 -DKE computer room 1.001 1100-1600
2	2012-04-18 (Wed)	-MSC Lecture Hall **1.009** 0900-1100 -DKE computer room 1.001 1100-1600
3	2012-04-25 (Wed)	-MSC Lecture Hall **1.028** **1100-1300** -DKE computer room 1.001 **1400-1800**
4	2012-05-04 (Fri)	-MSC Lecture Hall **1.009** 0900-1100 -DKE computer room 1.001 1100-1600
5	2012-05-11 (Fri)	-MSC Lecture Hall **1.009** 0900-1100 -DKE computer room 1.001 1100-1600
6	2012-05-16 (wed)	-MSC Lecture Hall **1.001** 0900-1100 -DKE computer room 1.001 1100-1600

Scientific Computing - Goals

- Goals
  - familiar with the concepts of programming
  - get accustomed with high-level languages like Matlab and Mathematica.
  - Provide an overview of some of the issues and problems that arise in scientific computation:
    - (non-)linear systems, numerical and symbolic integration, differential equations and simulation.

Scientific Computing: What is it about?

- **Computing**: we will learn to 'program'
  - Really: make the computer do what you want.
  - In this course we will work with
    - Matlab, or
    - (free software) Octave.



- **Scientific**:
  - We will deal with scientific problems.
  - Mostly based on calculus and linear algebra.

Scientific Computing – Quiz

- Pop quiz:
  - Who has programming experience?
  - Who has experience with Matlab or Octave? Who with Mathematica?
  - Who knows what a matrix is?
  - Who knows what a matrix inverse is?
  - Who knows how to solve a system of linear equations?

Recommended further reading

- Recommended reading.

- MATLAB
  - Introduction to MATLAB. Delores M. Etter. 2nd ed.
- Linear Algebra
  - Linear Algebra and Its Applications. David C. Lay. 4th ed.
  - Linear Algebra. Gilbert Strang
- Further exploring numerical methods
  - Numerical Methods. An introduction to Scientific Computing Using MATLAB. Peter Linz, Richard L.C. Wang.

Why Scientific Computing?

- Why use computers?
- Why program yourself?

Why Scientific Computing?

- Why use computers?
  - Only very simple models can be solved by hand.
  - Usually: there is no closed form solution.
    - E.g., solving a polynomial equation of degree > 4
  - But can get numerical approximations!
- Why program yourself?
  - Science: if somebody programmed it, it has already been done!
  - Industry:
    - to use it, need to **understand** what a program does and how,
    - somebody needs to develop these programs (often internally)!

Alright, so what is programming?

- Programming is about making a machine (computer) do what you want it to.
  - difference with a oven or other machines?

Alright, so what is programming?

- Programming is about making a machine (computer) do what you want it to.
  - difference with a oven or other machines?
  - → a computer can do many tasks

    and programming let's you do that!
- We focus on scientific computations.

- Example: how many km is 1 light year?

How many km in a light year?

- 299792458 * 365 * 24 * 60 * 60 / 1000 = 9.4543e+12
- These computations become difficult to interpret!
  - How about if we could name parts of this computation?

How many km in a light year?

- 299792458 * 365 * 24 * 60 * 60 / 1000 = 9.4543e+12

- These computations become difficult to interpret!
  - How about if we could name parts of this computation?

```
speed_of_light = 299792458
secs_per_year = 365 * 24 * 60 * 60
m_per_lyear = speed_of_light * secs_per_year
km_per_lyear = m_per_year / 1000
```

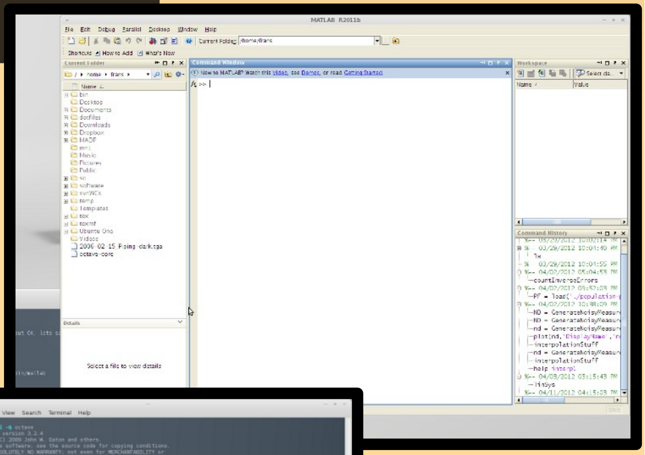
  - meaning of '='
  - the names are called 'variables'

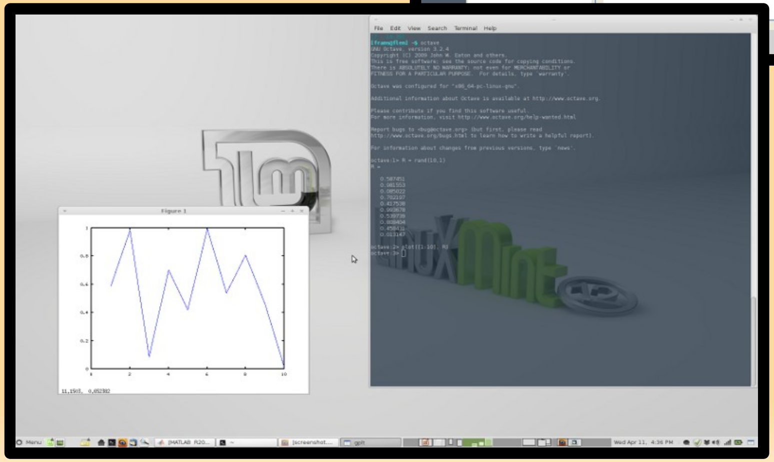
Our first Matlab/Octave code!

- This is our first Matlab code!

```
speed_of_light = 299792458
secs_per_year = 365 * 24 * 60 * 60
m_per_lyear = speed_of_light * secs_per_year
km_per_lyear = m_per_year / 1000
```

- (Demonstration)
- Matlab (Octave) is like a convenient calculator.



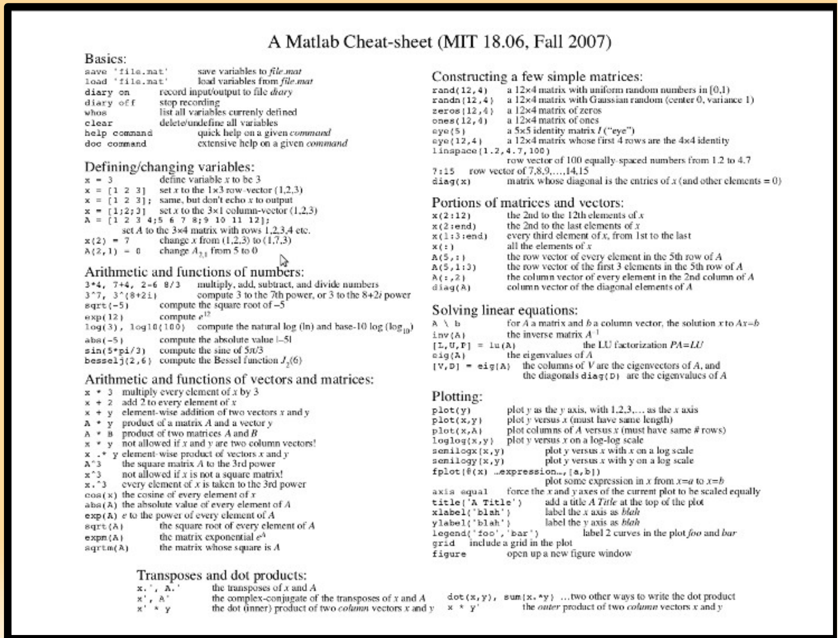


Operators

- Arithmetic:
  - +, -		addition, subtraction
  - *, /		multiplication, division
  - ^		power
  - sqrt		square root
  - log, log10	logarithms
  - mod		modulo
- E.g.:

```
octave:4> 1982980 / 2^8
ans =  7746.0
octave:5> mod(5,4)
ans =  1
```

→ all this is summarized in QSG
→ Google: 'matlab cheat sheet'



A Matlab Cheat-sheet (MIT 18.06, Fall 2007)

Basics:
save 'file.mat' — save variables to file.mat
load 'file.mat' — load variables from file.mat
diary on — record input/output to file diary
diary off — stop recording
whos — list all variables currenly defined
clear — delete/undefine all variables
help command — quick help on a given command
doc command — extensive help on a given command

Defining/changing variables:
x = 3 — define variable x to be 3
x = [1 2 3] — set x to the 1×3 row-vector (1,2,3)
x = [1 2 3]; — same, but don't echo x to output
x = [1;2;3] — set x to the 3×1 column-vector (1,2,3)
A = [1 2 3 4;5 6 7 8;9 10 11 12]; — set A to the 3×4 matrix with rows 1,2,3,4 etc.
x(2) = 7 — change x from (1,2,3) to (1,7,3)
A(2,1) = 0 — change $A_{2,1}$ from 5 to 0

Arithmetic and functions of numbers:
3*4, 7+4, 2-6 8/3 — multiply, add, subtract, and divide numbers
3^7, 3^(8+2i) — compute 3 to the 7th power, or 3 to the 8+2i power
sqrt(-5) — compute the square root of –5
exp(12) — compute e^{12}
log(3), log10(100) — compute the natural log (ln) and base-10 log ($\log_{10}$)
abs(-5) — compute the absolute value |–5|
sin(5*pi/3) — compute the sine of 5π/3
besselj(2,6) — compute the Bessel function $J_2(6)$

Arithmetic and functions of vectors and matrices:
x * 3 — multiply every element of x by 3
x + 2 — add 2 to every element of x
x + y — element-wise addition of two vectors x and y
A * y — product of a matrix A and a vector y
A * B — product of two matrices A and B
x * y — not allowed if x and y are two column vectors!
x .* y — element-wise product of vectors x and y
A^3 — the square matrix A to the 3rd power
x^3 — not allowed if x is not a square matrix!
x.^3 — every element of x is taken to the 3rd power
cos(x) — the cosine of every element of x
abs(A) — the absolute value of every element of A
exp(A) — e to the power of every element of A
sqrt(A) — the square root of every element of A
expm(A) — the matrix exponential e^A
sqrtm(A) — the matrix whose square is A

Constructing a few simple matrices:
rand(12,4) — a 12×4 matrix with uniform random numbers in [0,1)
randn(12,4) — a 12×4 matrix with Gaussian random (center 0, variance 1)
zeros(12,4) — a 12×4 matrix of zeros
ones(12,4) — a 12×4 matrix of ones
eye(5) — a 5×5 identity matrix I ("eye")
eye(12,4) — a 12×4 matrix whose first 4 rows are the 4×4 identity
linspace(1.2,4.7,100) — row vector of 100 equally-spaced numbers from 1.2 to 4.7
7:15 — row vector of 7,8,9,...,14,15
diag(x) — matrix whose diagonal is the entries of x (and other elements = 0)

Portions of matrices and vectors:
x(2:12) — the 2nd to the 12th elements of x
x(2:end) — the 2nd to the last elements of x
x(1:3:end) — every third element of x, from 1st to the last
x(:) — all the elements of x
A(5,:) — the row vector of every element in the 5th row of A
A(5,1:3) — the row vector of the first 3 elements in the 5th row of A
A(:,2) — the column vector of every element in the 2nd column of A
diag(A) — column vector of the diagonal elements of A

Solving linear equations:
A \ b — for A a matrix and b a column vector, the solution x to Ax=b
inv(A) — the inverse matrix A^{-1}
[L,U,P] = lu(A) — the LU factorization PA=LU
eig(A) — the eigenvalues of A
[V,D] = eig(A) — the columns of V are the eigenvectors of A, and the diagonals diag(D) are the eigenvalues of A

Plotting:
plot(y) — plot y as the y axis, with 1,2,3,... as the x axis
plot(x,y) — plot y versus x (must have same length)
plot(x,A) — plot columns of A versus x (must have same # rows)
loglog(x,y) — plot y versus x on a log-log scale
semilogx(x,y) — plot y versus x with x on a log scale
semilogy(x,y) — plot y versus x with y on a log scale
fplot(@(x) ...expression...,[a,b]) — plot some expression in x from x=a to x=b
axis equal — force the x and y axes of the current plot to be scaled equally
title('A Title') — add a title A Title at the top of the plot
xlabel('blah') — label the x axis as blah
ylabel('blah') — label the y axis as blah
legend('foo','bar') — label 2 curves in the plot foo and bar
grid — include a grid in the plot
figure — open up a new figure window

Transposes and dot products:
x.', A.' — the transposes of x and A
x', A' — the complex-conjugate of the transposes of x and A
x' * y — the dot (inner) product of two column vectors x and y
dot(x,y), sum(x.*y) — ...two other ways to write the dot product
x * y' — the outer product of two column vectors x and y

Scripts

- You may want to repeat a list of instructions.
- Just create a plain text file with .m extension

```
% a_script.m
% A first matlab script
%
% <- note that these percentages
% indicate comments

radius = 2.4

% Note 'pi'
circum = radius * 2 * pi

height = volume / circum
```

→ What is the output?

Scripts

- You may want to repeat a list of instructions.
- Just create a plain text file with .m extension

```
% fixed_script.m

radius = 2.4
volume = 48

% Note 'pi'
circum = radius * 2 * pi

height = volume / circum
```

→ Volume was not defined!

→ Alternative: set volume before calling the script.

So, perform:
> volume = 48
> a_script

Matlab Path

- A script will only run when it is in a place where matlab can find it.
- Matlab looks in a list of directories called 'path'
  - path
  - see “help path”

- Normally: the current working directory is in the path
  - pwd
  - cd

Suppressing/Showing Output

- we may not want to show all intermediate results
  - use ';'
  - show some particular things using 'disp'

```
% fixed_script.m

radius = 2.4; %<- surpress output!
circum = radius * 2 * pi;
volume = 48;
height = volume / circum;
disp('height is');
disp(height);
```

Conditions: If

- Sometimes you want to do things only is some cases.
- Called '**branching**' and is a very important capability.

```
% longest_side.m
% ---------
% this script determines the longest
% side of a rectangle. It expects 2
% variables 'length_x' and 'length_y'
% to be defined.

% assume y is longest side:
longest_side = length_y;

if length_x > length_y
    longest_side = length_x;
end

disp(longest_side);
```

If...else...

- The previous way of writing is not the most intuitive...
  - the default assumption is awkward
  - use “else”

```
% longest_side_else.m
% ---------
% this script determines the longest
% side of a rectangle. It expects 2
% variables 'length_x' and 'length_y'
% to be defined.

if length_x > length_y
    longest_side = length_x;
else
    longest_side = length_y;
end
disp(longest_side);
```

If...elseif...else...

- More generally, we test multiple conditions

```
if CONDITION1
    …
elseif CONDITION2
    …
elseif CONDITION3
    …
else
    …
end
```

Conditions

- So exactly what are the CONDITIONs?
  - expressions that evaluate to `true' or 'false'
  - 'false' defined as '0'
  - 'true' is any non-zero value

```
truthvalue = 0
if truthvalue
    disp('true')
else
    disp('false')
end
```

- This code can be used to test any truth value expression.

Conditions - 2

- Can make more complex expressions by 'operators'

Relational operators:

- A < B,
- A > B
- A <= B
- A >= B
- A == B
- A ~= B

Logical operators:

- ~A
- A | B,
- A & B

'short-circuit'

- A || B
- A && B

```
octave> ~1
ans = 0
octave> 1 & 0
ans = 0
octave> -1 | 0
ans =  1
octave> 0 | 0
ans = 0
```

Do it again: loops

- Another important capability: repeating instructions.
  - i.e., performing 'loops'.

- Matlab has 2 types of loops:
  - 'for' when you know how often you need to loop in advance.
  - 'while' when you don't, but only have a stopping criteria.

For loop

- For loops: used when you know how often you need to loop.

```
%count to 10
for i = [1:10]
    disp(i)
end

%count down:
for i = [10:1]
    disp(i)
end
```

For loop

- For loops: used when you know how often you need to loop.

```
%count to 10
for i = [1:10]
    disp(i)
end

%count down:
start = 10
for i = [start:1]
    disp(i)
end
```

```
octave:12> [1:10]
ans =

    1    2    3    4    5    6    7    8    9   10
```

- (almost) everything in matlab is an array or matrix!

While loop

- Sometimes it is hard to know how often we loop

  → use 'while'

```
% strange count down
n = 14209

i = 1;
while(n > 1)
    disp(i)
    if n % 2 == 0
        n = n / 2
    else
        n = n + 1
    end
    i = i + 1;
end
```

While loop

- Sometimes it is hard to know how often we loop

  → use 'while'

```
% strange count down
n = 14209

i = 1;
while(n > 1)
    disp(i)
    if n % 2 == 0
        n = n / 2
    else
        n = n + 1
    end
    i = i + 1;
end
```

```
n =  14209
    1
n =  14210
    2
n =  7105
    3
n =  7106
    4
n =  3553
    5
n =  3554
    6
n =  1777
    7
n =  1778
    8
n =  889
    9
n =  890
   10
n =  445
   11
n =  446
   12
n =  223
   13
n =  224
   14
n =  112
   15
n =  56
   16
n =  28
   17
n =  14
   18
n =  7
   19
n =  8
   20
n =  4
   21
n =  2
   22
n =  1
```

Reusing code

- A very important concept: code reuse
- All these scripts are nice, but...
  - writing scripts for complex tasks is a lot of work.
  - often there is functionality we want to reuse!
- This is where 'functions' come in...
  - a piece of code that performs a specific task
  - has input and output.

Using Matlab/Octace Functions

- Matlab has many built in functions.
  - We already saw a few: 'mod', 'sqrt'

- Calling a function: FUNCTIONNAME(…, …, ...)
  - 'mod(3,2)'
  - 'pi()' or just 'pi'
  - [m, index] = max([4, 2, 6, 3])

Writing your own Functions

- You can write your own function very simply

```
function output = FunctionName(input1, input2)

…
…
output = …
…
```

- Need to name the file 'FunctionName.m'

Writing your own Functions

- You can write your own function very simply

```
function longest = LongestSide(length_x, length_y)

if length_x > length_y
    longest = length_x;
else
    longest = length_y;
end
```

- Need to name the file 'LongestSide.m'
- Capitalization of 'LongestSide' is a convention
  - (no rule)

Writing your own Functions

- You can write your own function very simply

```
function longest = LongestSide(length_x, length_y)

if length_x > length_y
    longest = length_x;
else
    longest = length_y;
end
```

- Need to name the file 'LongestSide.m'
- Capitalization of 'LongestSide' is a convention
  - (no rule)

```
octave:33> LongestSide(3, 5)
ans =  5
```

Writing your own Functions

- Document your functions!

```
function longest = LongestSide(length_x, length_y)
%function longest = LongestSide(length_x, length_y)
%
% this is a special comment block: it is shown when
% calling 'help LongestSide'

if length_x > length_y
    longest = length_x;
else
    longest = length_y;
end
```

- For yourself **and** others.

Anonymous Functions

- Small functions can also be defined in the matlab environment.
  - in lab
  - even more ways in book

```
octave:35> MyAddFunction = @(x,y) x+y
MyAddFunction =

@(x, y) x + y

octave:36> MyAddFunction(2,4)
ans =  6
```

Recap Programming

- Congrats: Now you know the most important constructs of programming!
- Let's summarize:
  - → Advanced calculator
  - Variables: names for intermediate parts of computation.
  - Arithmetic operators
  - Scripts
  - Branching: if … else …, conditions
  - Loops: for, while
  - Functions
  - ← full blown programming.

A First Bit of Scientific Programming

- Now that you know the most important constructs of programming...

  ...we can start!

- with a **scientific** programming problem:

  Solving non-linear equations

What are (non-)linear equations?

- linear equations?

What are (non-)linear equations?

- linear equations

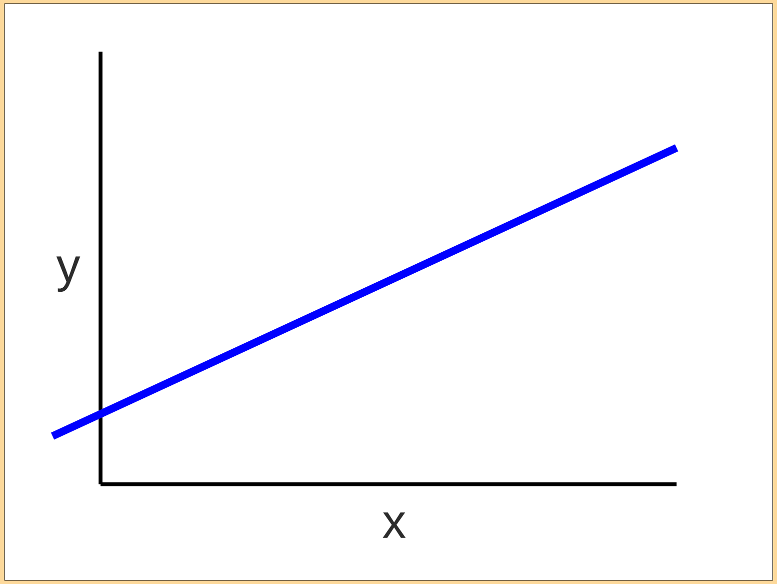
$$3x+7y=4$$

$$x-3y=4+z$$

$$(x-3y)/z=2$$

'General Form'

$$a_0+a_1x_1+a_2x_2+\ldots=0$$



Straight line
(for 2 variables)

What are (non-)linear equations?

- linear equations

$$3x + 7y = 4$$

$$x - 3y = 4 + z$$

$$(x - 3y)/z = 2$$

'General Form'

$$a_0 + a_1 x_1 + a_2 x_2 + \ldots = 0$$

- non-linear equations?

What are (non-)linear equations?

- linear equations

$$3x+7y=4$$

$$x-3y=4+z$$

$$(x-3y)/z=2$$

'General Form'

$$a_0+a_1x_1+a_2x_2+...=0$$

- non-linear equations:

All equations that are **not** linear!

$$x^2=4$$

$$xy=2$$

$$y=\sqrt{x}$$

Finding the 'roots'

- Many problems can be reformulated as finding the 'roots' or 'zeros' of a function.

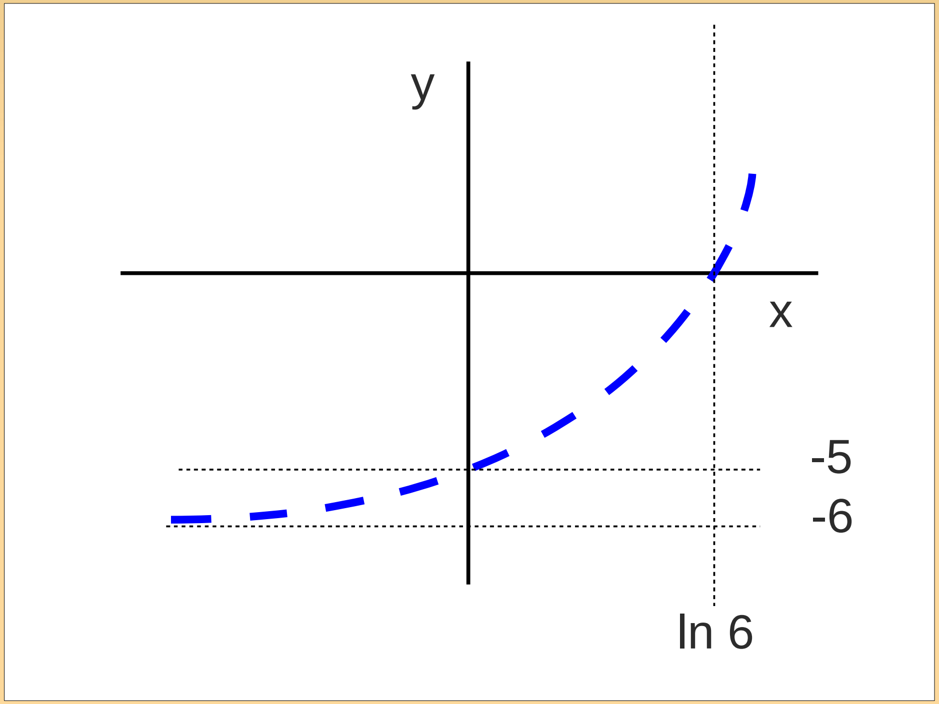
- What is ln 6 ?

Finding the 'roots'

- Many problems can be reformulated as finding the 'roots' or 'zeros' of a function.

- What is ln 6 ?

$$e^x = 6$$
$$e^x - 6 = 0$$

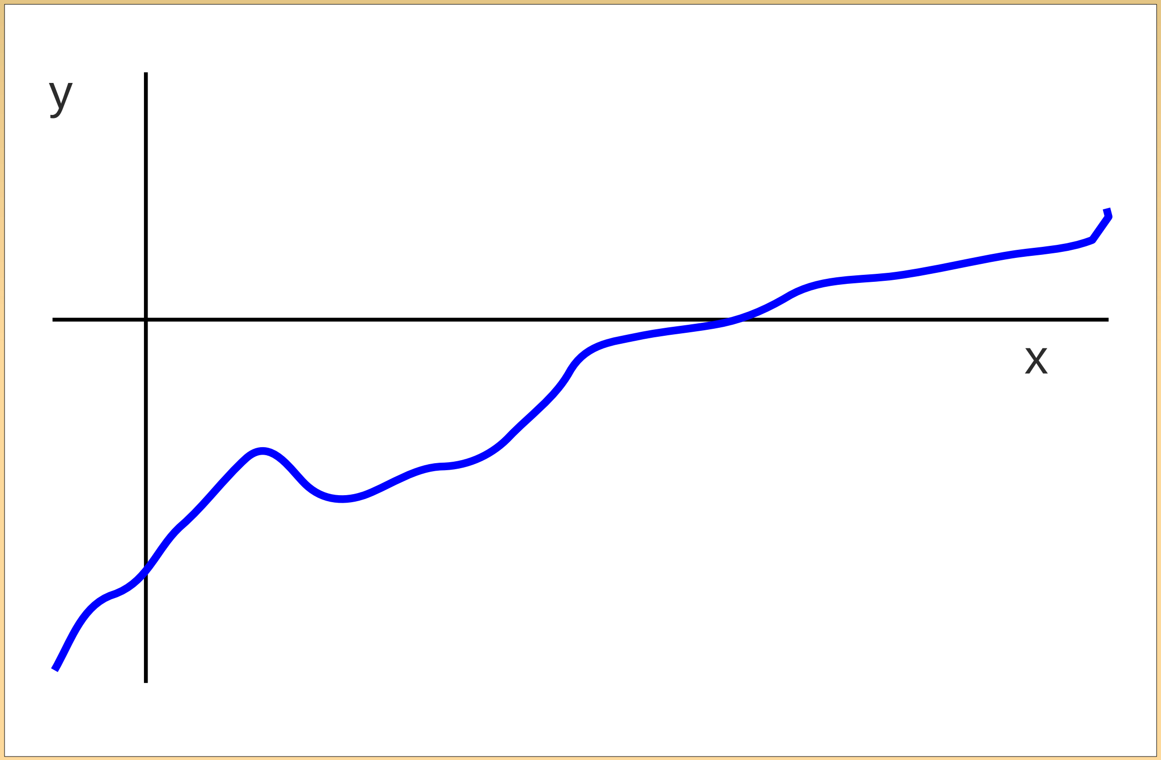


Numerical Algorithms

- To solve this problem we will now discuss our first numerical method, or numerical **algorithm**.
- Roughly:
  - algorithm = cook-book recipe
  - an algorithm can be **implemented** (converted to code in a programming language).

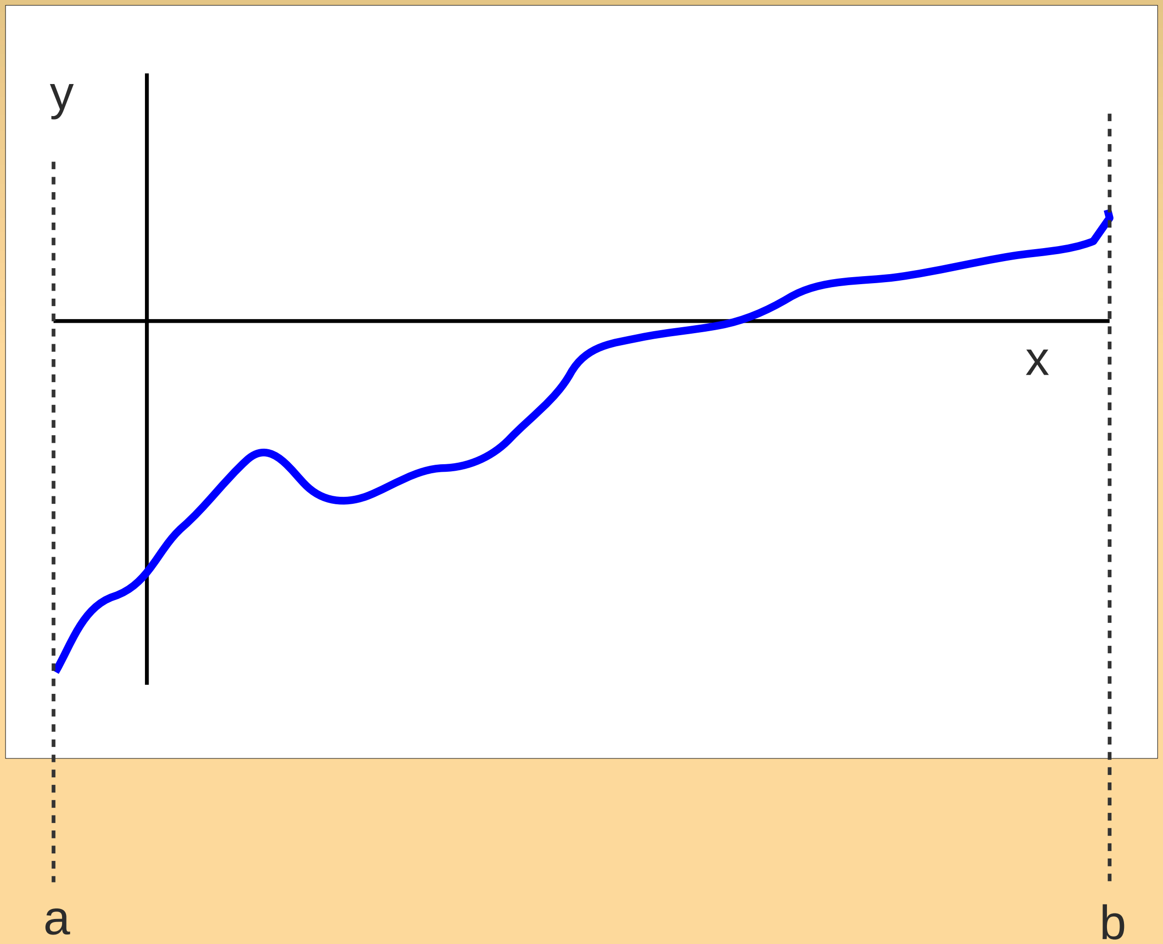
The Bisection Method

- Suppose we want to find the roots of this function?



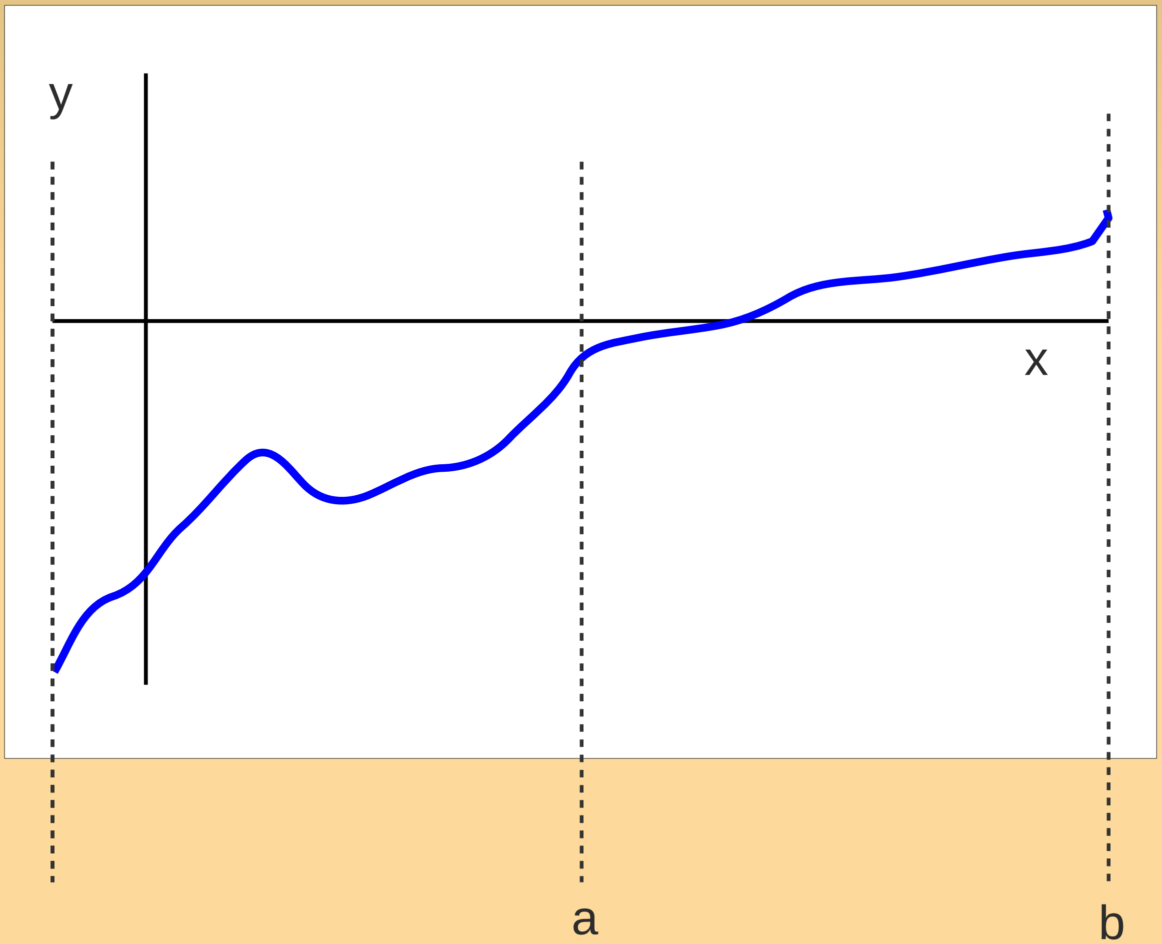
The Bisection Method

- Search the interval [a,b] for the crossing point!



The Bisection Method

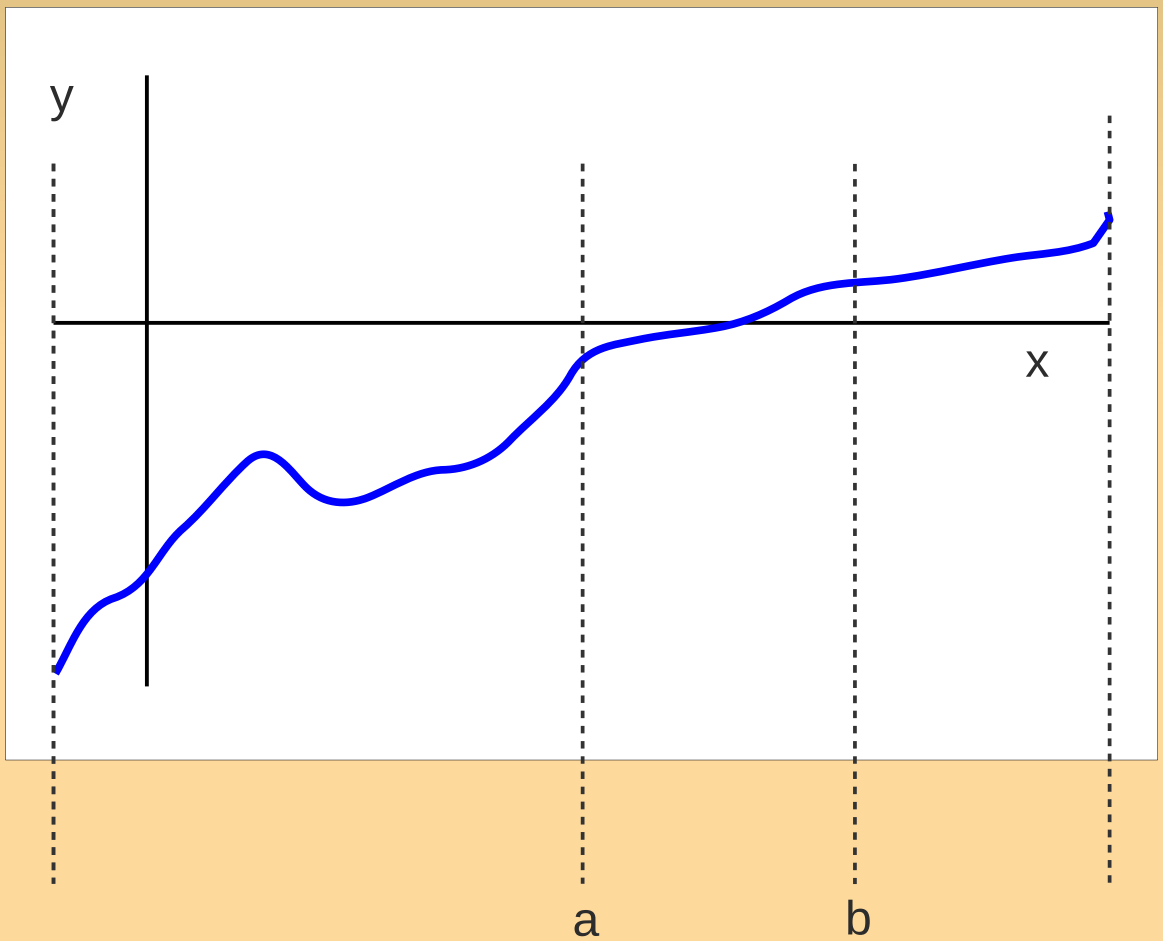
- Halve the interval



- Then select the interval where the crossing occurs

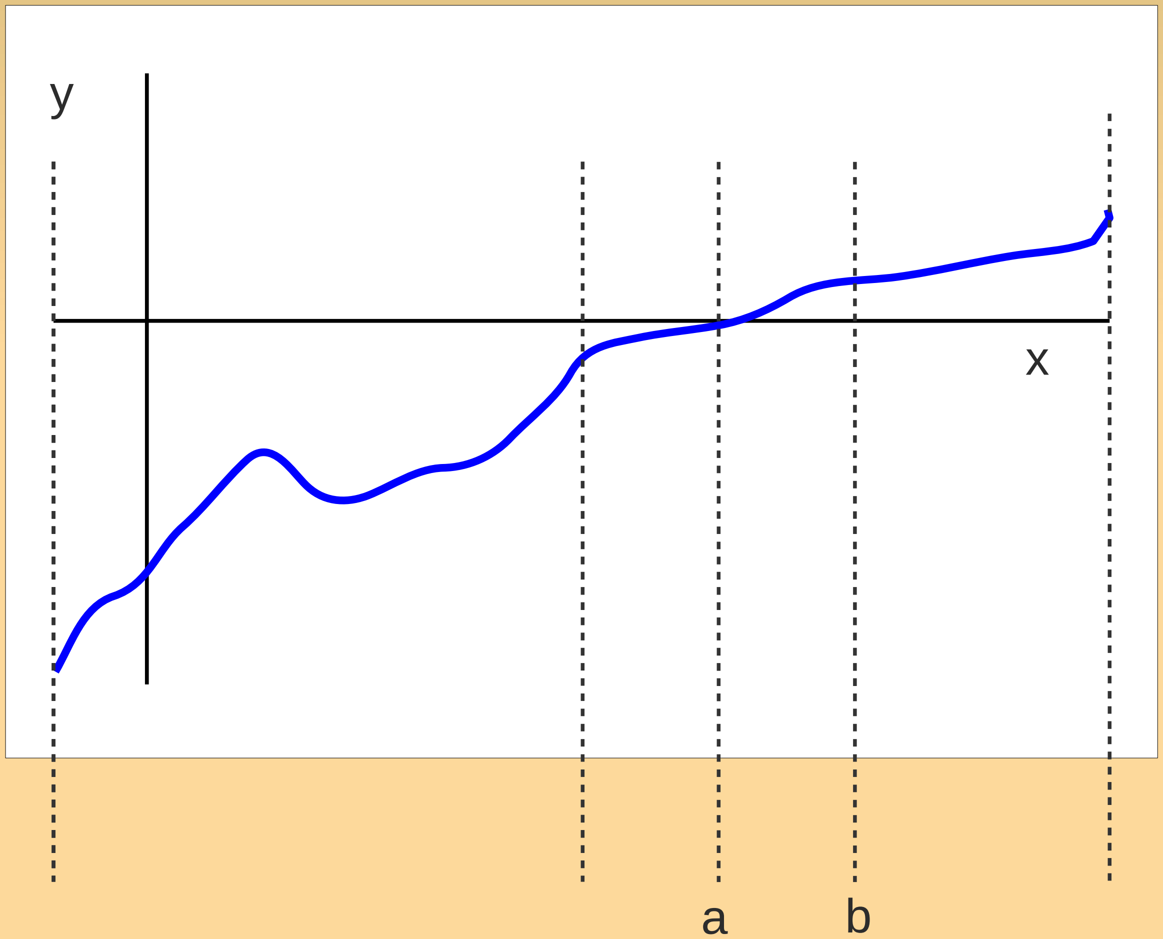
The Bisection Method

- Repeat, until the interval is small enough



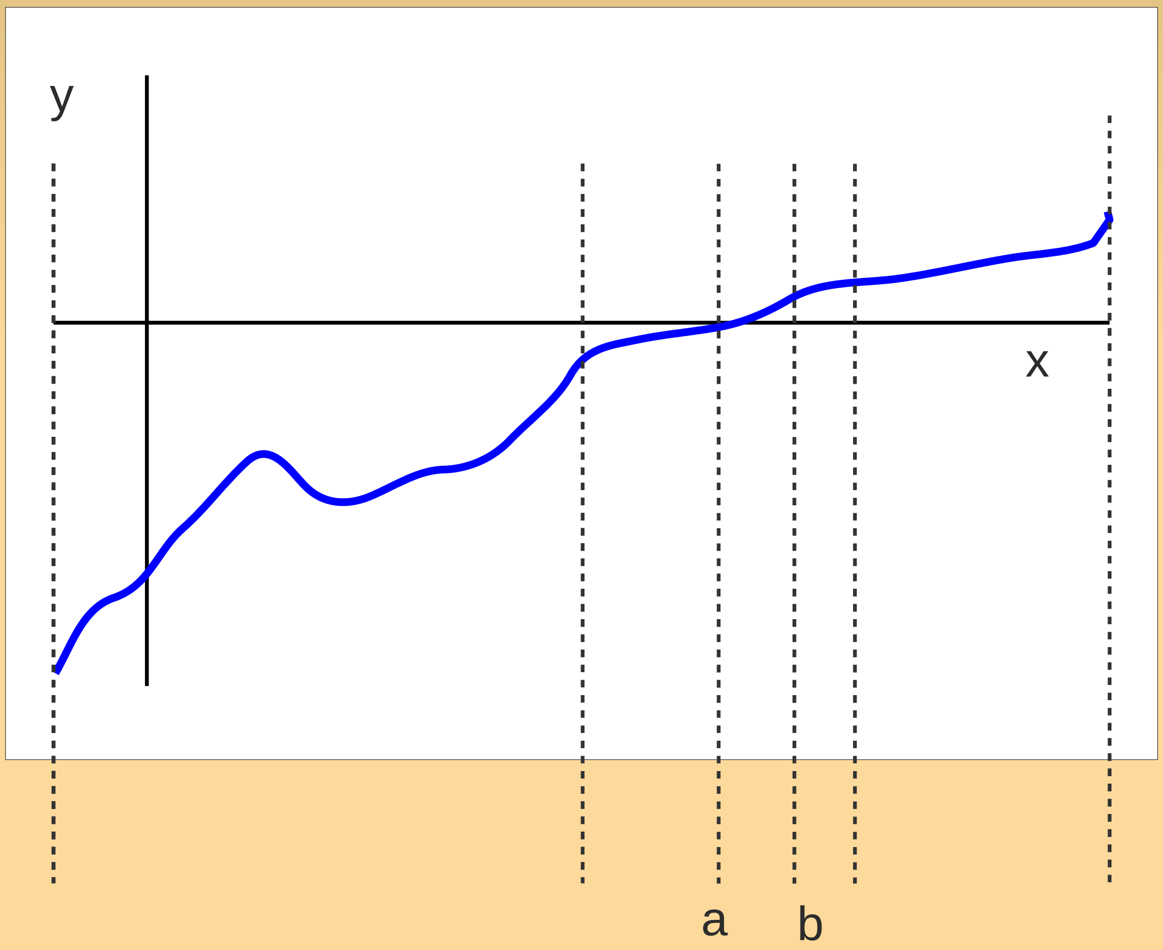
The Bisection Method

- Repeat, until the interval is small enough



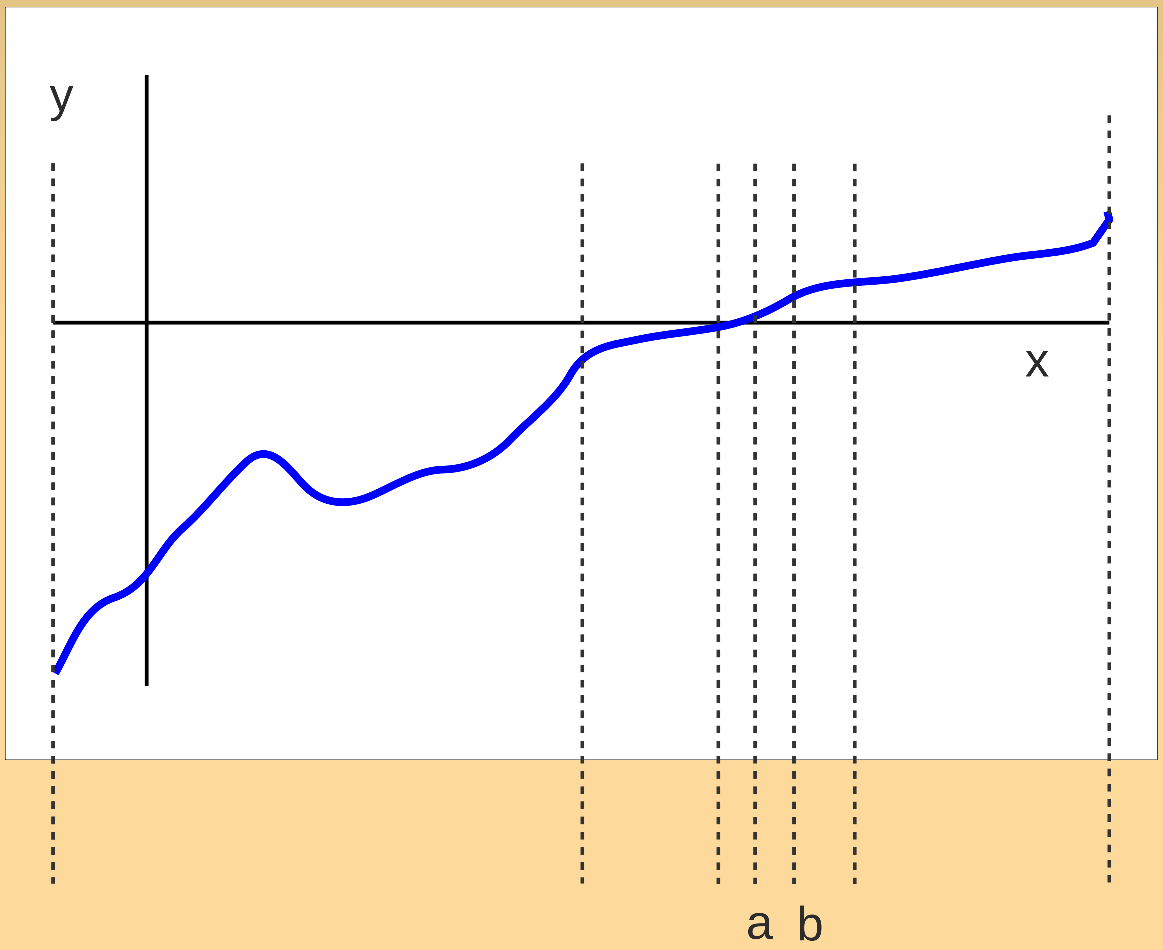
The Bisection Method

- Repeat, until the interval is small enough



The Bisection Method

- Repeat, until the interval is small enough



The Bisection Method

- Conditions to apply the Bisection Method:
  - f is continuous
  - interval [a,b]
    - f(a) is positive and f(b) is negative or vice versa

      → contains an a zero
      ('theorem of zeros of continuous functions')
    - check with $f(a)f(b) < 0$

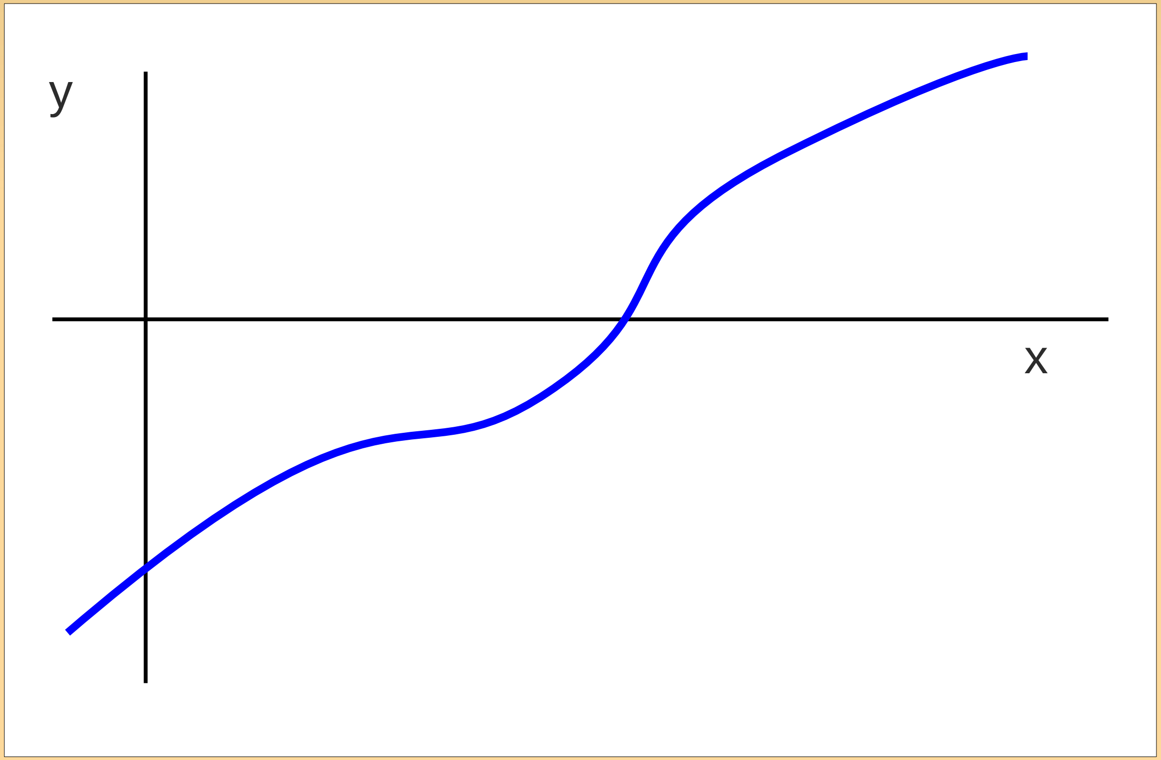
- To find a good initial interval: e.g., plot the function

The Bisection Method

- Pros
  - Simple conceptually
  - Only need information of sign of the function
    - Works in many settings
- Cons
  - Even needs many iterations on a linear function!

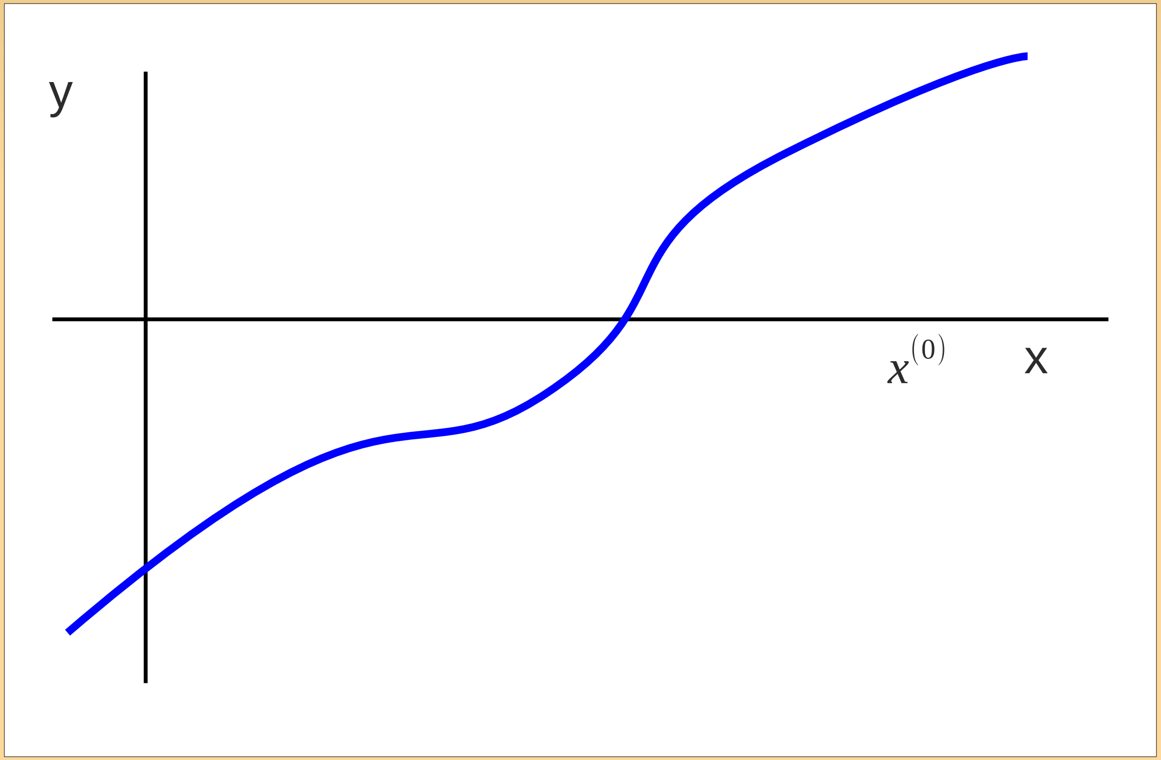
Newton's Method

- Newton's method is a different approach
  - overcomes some problems (but has its own)



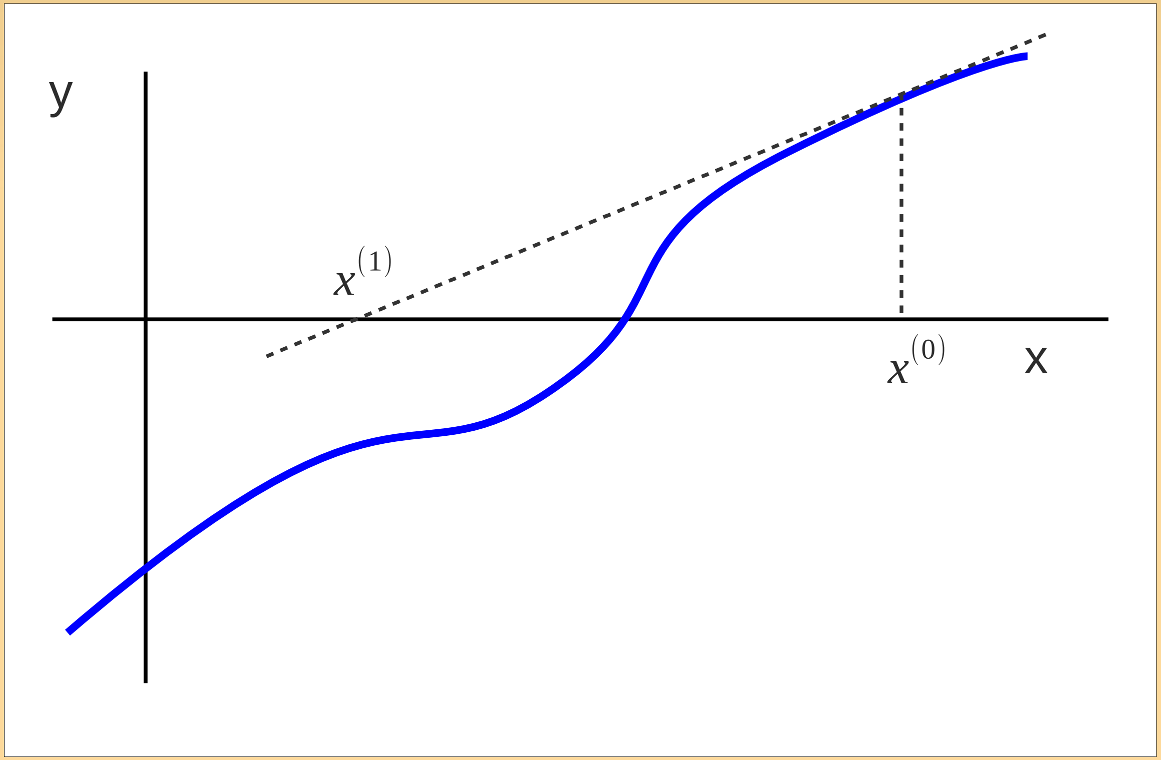
Newton's Method

- Start with an arbitrary point.



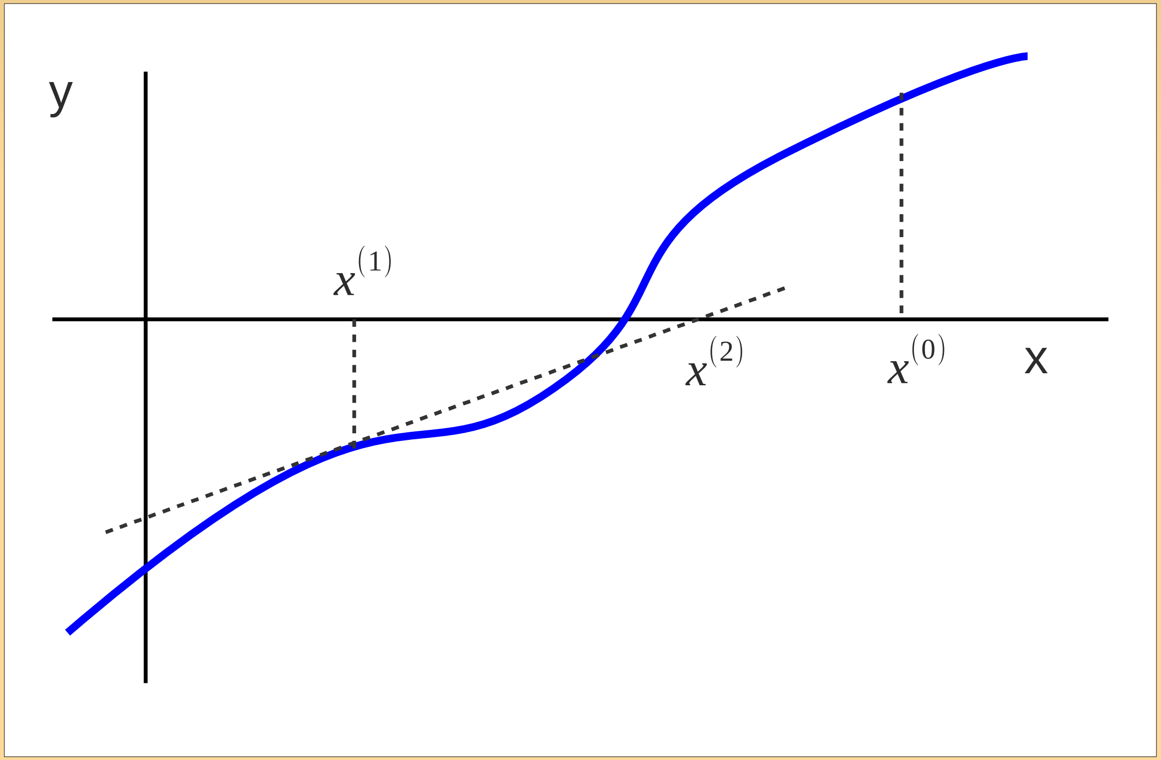
Newton's Method

- Compute next point via the derivative f'



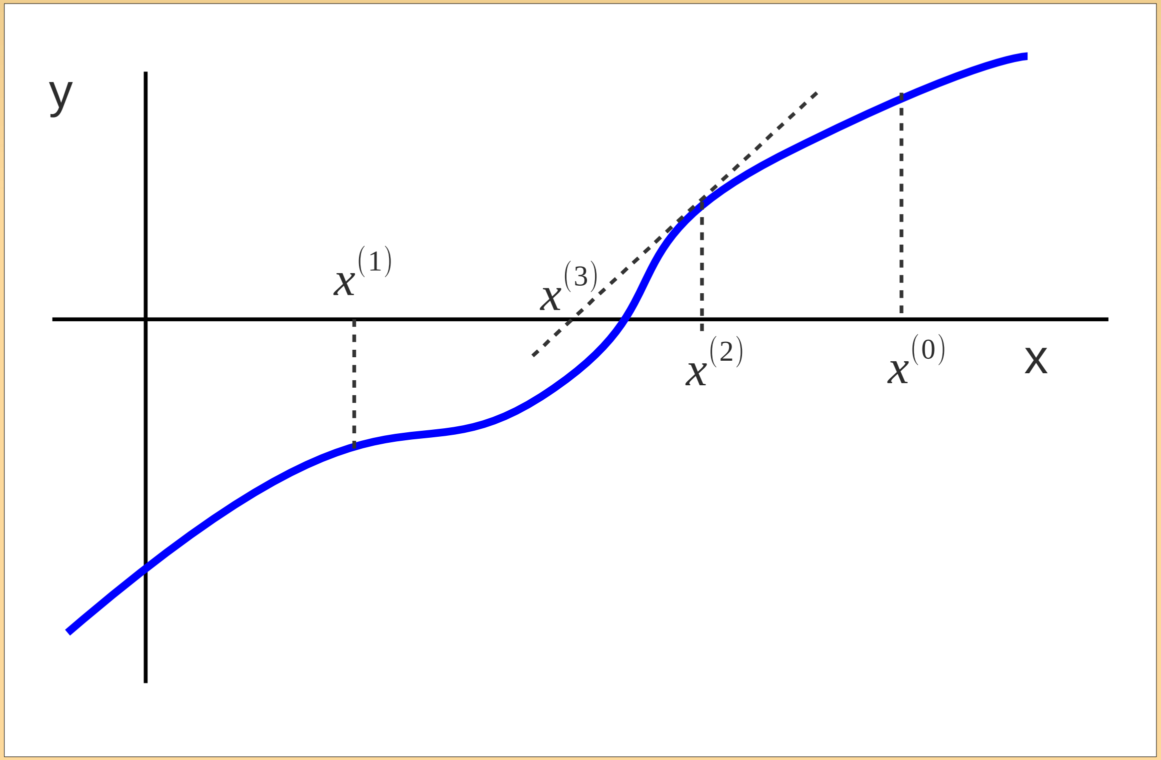
Newton's Method

- etc.



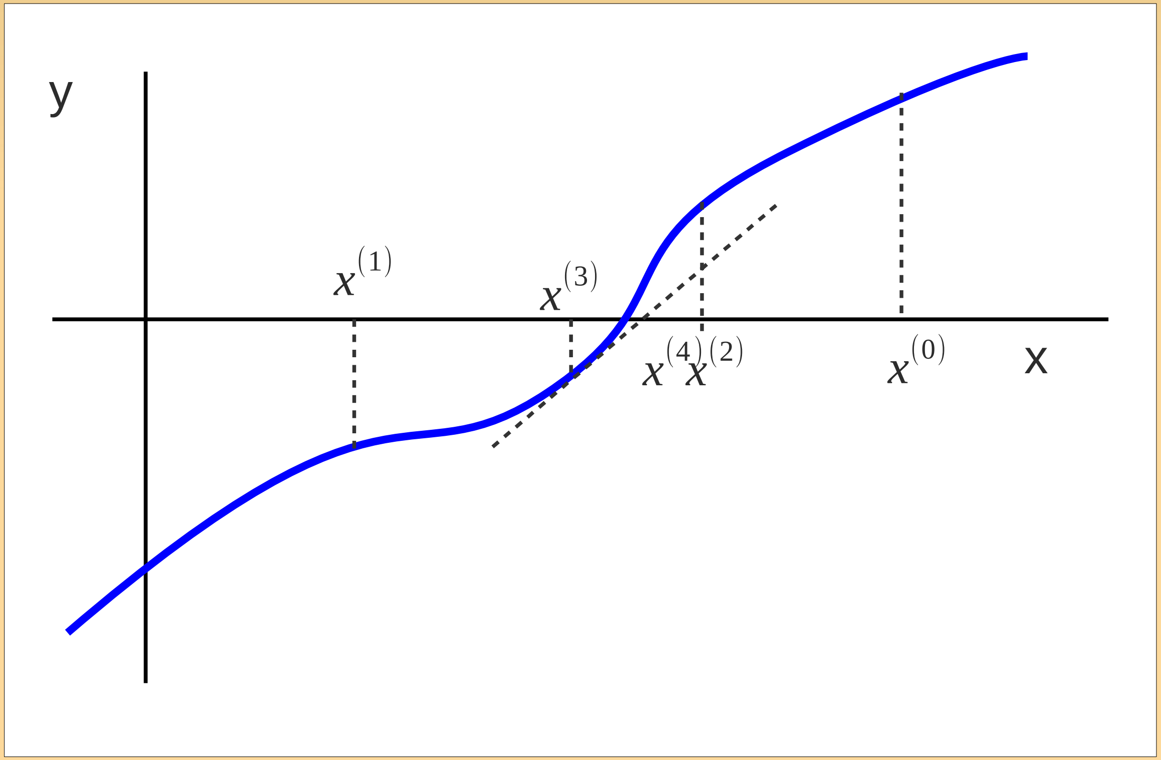
Newton's Method

- etc.



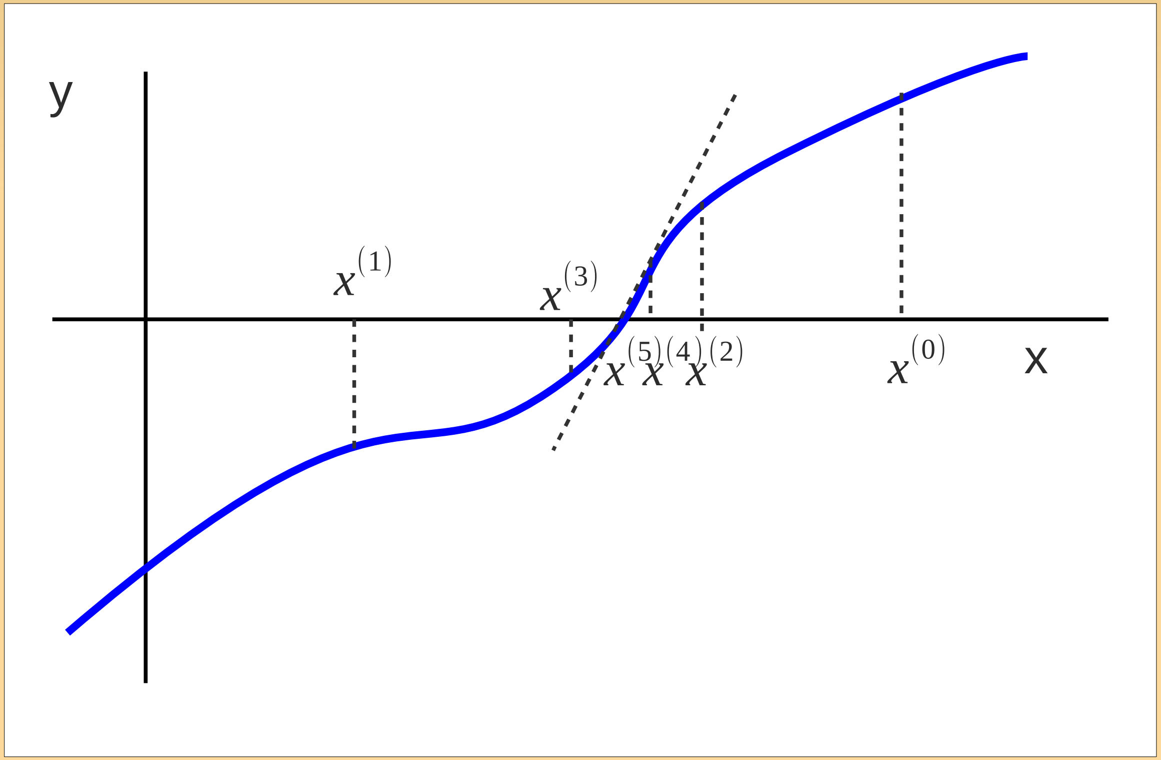
Newton's Method

- etc.



Newton's Method

- until difference with previous point small enough.



Newton's Method

- Algorithm:
  - Start with an arbitrary point $x^{(0)}$
  - Compute the next point $x^{(k+1)} = x^{(k)} - \dfrac{f(x^{(k)})}{f'(x^{(k)})}$
  - repeat while $|x^{(k+1)} - x^{(k)}| < \epsilon$

Newton's Method

- Pros
  - From some point on, it is **fast**!
    - converges 'quadratically'
    - error of next error is square of previous one.
- Cons
  - Need more information: function derivative
  - Needs to be initialized sufficiently close to 0
  - Problem when $f'\left(x^{(k)}\right)=0$

Homework Reading

- Recap:
  - H1: 1.1, 1.5-1.5.1, 1.7,
  - H2: p. 41--48 (that is including 48).
- Preparation for next time:
  - H1: 1.2, 1.5.2, 1.6.
  - H3: p. 75--81, 93--103 (sec. 3.5 is optional)

- Read chap.1 sequentially
  - skipping 1.3 and 1.5.3.
- When reading for preparation:
  - skip things that are not clear!

    → Ask them in class.

Let's get started

- Lab assignments are posted on my website.

  http://people.csail.mit.edu/fao

- Reminder: bring the head phones!