Matlab and Programming 2

Advanced programming techniques

Ivo Roghair, Martin van Sint Annaland

Chemical Process Intensification
Eindhoven University of Technology

Today's outline

1. Coding in style
2. Error management
3. Visualisation
4. Functions: the sequel
5. Excel
6. Algorithms
7. Conclusions

Today's outline

1. Coding in style
2. Error management
3. Visualisation
4. Functions: the sequel
5. Excel
6. Algorithms
7. Conclusions

If anything sticks today, let it be this

Your code will not be understood by anyone

If anything sticks today, let it be this

Your code will not be understood by anyone

That includes future-you

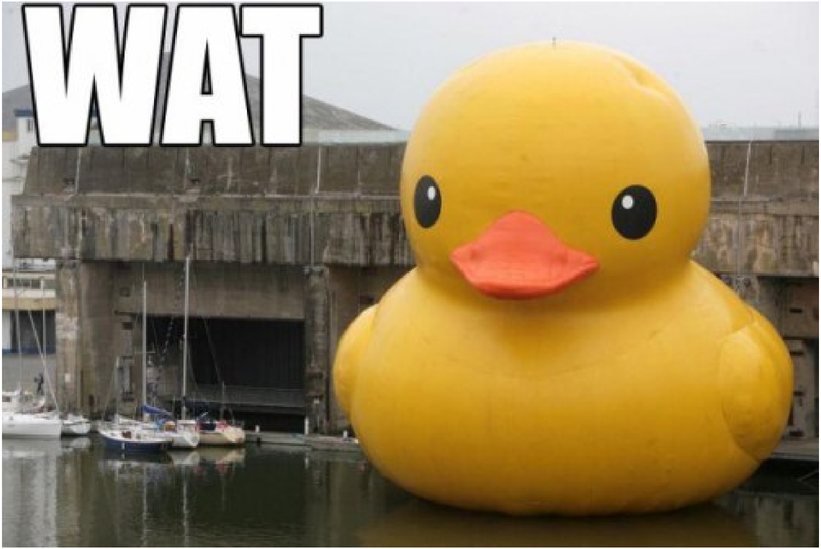
Code organization

- Optimization of a code is time-consuming and complicated
- The more you optimize your code, the less readable it becomes
- But... You can write it in a such way that it will be flexible and easy to maintain
- Especially important in team work
- Any person has its own handwriting. Any programmer has its own coding style.

The coding style $\equiv$ handwriting...

Interpret the following code

```
s=checksc();
if(s==true)
a=cb();
b=cfrsp();
if(a<5)
if(b>5)
a=gtbs();
end
if(a>b)
ubx();
end
end
else
brn();
gtbs();
end
```


WAT

Let's change that a bit... Indentation

Let's change that a bit... Indentation

Shown here with 2 spaces of indentation, Matlab uses 4 by default!

```
s=checksc();
if(s==true)
a=cb();
b=cfrsp();
if(a<5)
if(b>5)
a=gtbs();
end
if(a>b)
ubx();
end
end
else
brn();
gtbs();
end
```

```
s = checksc();
if (s == true)
  a = cb();
  b = cfrsp();
  if (a < 5)
    if (b > 5)
      a = gtbs();
    end
    if (a > b)
      ubx();
    end
  end
else
  brn();
  gtbs();
end
```

Readable variables and function names

```
s = checksc();
if (s == true)
  a = cb();
  b = cfrsp();
  if (a < 5)
    if (b > 5)
      a = gtbs();
    end
    if (a > b)
      ubx();
    end
  end
else
    brn();
    gtbs();
end
```

```
IAmFree = checkSchedule();
if (IAmFree == true)
  books = countBooks();
  shelfSize =
       countFreeSpaceShelf();
  if (books < 5)
    if (shelfSize > 5)
      books = goToBookStore();
    end
    if (books > shelfSize)
      useBox();
    end
  end
else
  burnBooks();
  goToBookStore();
end
```

Get rid of obscure constants in the code

```
IAmFree = checkSchedule();
if (IAmFree == true)
  books = countBooks();
  shelfSize =
      countFreeSpaceShelf();
  if (books < 5)
    if (shelfSize > 5)
      books = goToBookStore();
    end
    if (books > shelfSize)
      useBox();
    end
  end
else
  burnBooks();
  goToBookStore();
end
```

```
IAmFree = checkSchedule();
if (IAmFree == true)
  books = countBooks();
  shelfSize =
      countFreeSpaceShelf();
  if (books < maxShelfSize)
    if (shelfSize >
        minBooksNeeded)
      books = goToBookStore();
    end
    if (books > shelfSize)
      useBox();
    end
  end
else
  burnBooks();
  goToBookStore();
end
```

That's more like it!

```
s=checksc();
if(s==true)
a=cb();
b=cfrsp();
if(a<5)
if(b>5)
a=gtbs();
end
if(a>b)
ubx();
end
end
else
brn();
gtbs();
end
```

```
IAmFree = checkSchedule();
if (IAmFree == true)
  books = countBooks();
  shelfSize = countFreeSpaceShelf();
  if (books < maxShelfSize)
    if (shelfSize > minBooksNeeded)
      books = goToBookStore();
    end
    if (books > shelfSize)
      useBox();
    end
  end
else
  burnBooks();
  goToBookStore();
end
```

Writing readable code

Good code reads like a book.

Writing readable code

Good code reads like a book.

- When it doesn't, make sure to use comments. In Matlab, everything following `% is a comment`
- Prevent "smart constructions" in the code
- Re-use working code (i.e. create functions for well-defined tasks).
- Documentation is also useful, but hard to maintain.

Writing readable code

Good code reads like a book.

- When it doesn't, make sure to use comments. In Matlab, everything following `% is a comment`
- Prevent "smart constructions" in the code
- Re-use working code (i.e. create functions for well-defined tasks).
- Documentation is also useful, but hard to maintain. (Matlab comes with a function that generates reports from comments)

How not to comment

- Useless:

```
% Start program
```

How not to comment

- Useless:

```
% Start program
```

- Obvious:

```
if (a > 5)    % Check if a is greater than 5
    ...
end
else          % else add 1 to b
    b = b + 1;
end
```

How not to comment

- Useless:

```
% Start program
```

- Obvious:

```
if (a > 5)    % Check if a is greater than 5
    ...
end
else          % else add 1 to b
    b = b + 1;
end
```

- Too much about the life:

```
% Well... I do not know how to explain what is going on
% in the snippet below. I tried to code in the night
% with some booze and it worked then, but now I have a
% strong hangover and some parameters still need to be
% worked out...
```

Adding comments to our program

```
IAmFree = checkSchedule();
if (IAmFree == true)
% Count books and amount of free space on a shelf.
% If minimum number of books I need is less than a
% shelf capacity, go shopping and buy additional
% literature. If the amount of books after the
% shopping is too big, use boxes to store them.
  books = countBooks();
  shelfSize = countFreeSpaceShelf();

  ...

else
  burnBooks();
  goToBookStore();
end
```

If anything sticks today, let it be this

Your code will not be understood by anyone

If anything sticks today, let it be this

Your code will not be understood by anyone

That includes future-you

If anything sticks today, let it be this

Your code will not be understood by anyone

That includes future-you

> Use comments and code to document design and purpose (functionality), not mechanics (implementation).

If anything sticks today, let it be this

Your code will not be understood by anyone

That includes future-you

Use comments and code to document design and purpose (functionality), not mechanics (implementation).

Use consistent and sensible naming of functions and variables.

Today's outline

Errors in computer programs

Computer programs often contain errors (bugs): buildings collapse, governments fall, kittens will die.

Errors in computer programs

The following symptoms can be distinguished:

- Unable to execute the program
- Program crashes, warnings or error messages
- Never-ending loops
- Wrong (unexpected) result

Errors in computer programs

The following symptoms can be distinguished:

- Unable to execute the program
- Program crashes, warnings or error messages
- Never-ending loops
- Wrong (unexpected) result

Three error categories:

Syntax errors You did not obey the language rules. These errors prevent running or compilation of the program.

Runtime errors Something goes wrong during the execution of the program resulting in an error message (problem with input, division by zero, loading of non-existent files, memory problems, etc.)

Semantic errors The program does not do what you expect, but does what have told it to do.

Errors in computer programs

The following symptoms can be distinguished:

- Unable to execute the program
- Program crashes, warnings or error messages
- Never-ending loops
- Wrong (unexpected) result

Three error categories:

Syntax errors You did not obey the language rules. These errors prevent running or compilation of the program.

Runtime errors Something goes wrong during the execution of the program resulting in an error message (problem with input, division by zero, loading of non-existent files, memory problems, etc.)

Semantic errors The program does not do what you expect, but does what have told it to do.

Errors in computer programs

The following symptoms can be distinguished:

- Unable to execute the program
- Program crashes, warnings or error messages
- Never-ending loops
- Wrong (unexpected) result

Three error categories:

Syntax errors You did not obey the language rules. These errors prevent running or compilation of the program.

Runtime errors Something goes wrong during the execution of the program resulting in an error message (problem with input, division by zero, loading of non-existent files, memory problems, etc.)

Semantic errors The program does not do what you expect, but does what have told it to do.

A convenient tool: the debugger

- No-one can write a 1000-line code without making errors
  - If you can, please come work for us
- One of the most important skills you will acquire is debugging.
- Although it can be frustrating, debugging is one of the most intellectually rich, challenging, and interesting parts of programming.
- In some ways, debugging is like detective work. You are confronted with clues, and you have to infer the processes and events that led to the results you see.

A convenient tool: the debugger

- No-one can write a 1000-line code without making errors
  - If you can, please come work for us
- One of the most important skills you will acquire is debugging.
- Although it can be frustrating, debugging is one of the most intellectually rich, challenging, and interesting parts of programming.
- In some ways, debugging is like detective work. You are confronted with clues, and you have to infer the processes and events that led to the results you see.

A convenient tool: the debugger

- No-one can write a 1000-line code without making errors
  - If you can, please come work for us
- One of the most important skills you will acquire is debugging.
- Although it can be frustrating, debugging is one of the most intellectually rich, challenging, and interesting parts of programming.
- In some ways, debugging is like detective work. You are confronted with clues, and you have to infer the processes and events that led to the results you see.

A convenient tool: the debugger

- No-one can write a 1000-line code without making errors
  - If you can, please come work for us
- One of the most important skills you will acquire is debugging.
- Although it can be frustrating, debugging is one of the most intellectually rich, challenging, and interesting parts of programming.
- In some ways, debugging is like detective work. You are confronted with clues, and you have to infer the processes and events that led to the results you see.

"When you have eliminated the impossible, whatever remains, however improbable, must be the truth."
— A. Conan Doyle, The Sign of Four

A convenient tool: the debugger

The debugger can help you to:

- Pause a program at a certain line: set a *breakpoint*
- Check the values of variables during the program
- Controlled execution of the program:
  - One line at a time
  - Run until a certain line
  - Run until a certain condition is met (conditional breakpoint)
  - Run until the current function exits
- Note: You may end up in the source code of Matlab functions!

A convenient tool: the debugger

The debugger can help you to:

- Pause a program at a certain line: set a *breakpoint*
- Check the values of variables during the program
- Controlled execution of the program:
  - One line at a time
  - Run until a certain line
  - Run until a certain condition is met (conditional breakpoint)
  - Run until the current function exits
- Note: You may end up in the source code of Matlab functions!
- Check Canvas (Matlab Crash Course section) for a movie that demonstrates the debugger.

About testcases (validation)

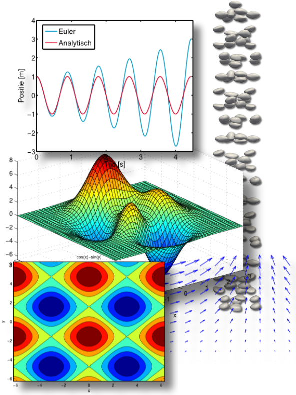
- Testcases: run the program with parameters such that a known result is (should be) produced.
- Testcases: what happens when unforeseen input is encountered?
  - More or fewer arguments than anticipated? (Matlab uses `varargin` and `nargin` to create a varying number of input arguments, and to check the number of given input arguments
  - Other data types than anticipated? How does the program handle this? Warnings, error messages (crash), NaN or worse (a continuing program)?
- For physical modeling, we typically look for analytical solutions
  - Sometimes somewhat stylized cases
  - Possible solutions include Fourier-series
  - Experimental data

Today's outline

Data visualisation

Modeling can lead to very large data sets, that require appropriate visualisation to convey your results.

- 1D, 2D, 3D visualisation
- Multiple variables at the same time (temperature, concentration, direction of flow)
- Use of colors, contour lines
- Use of stream lines or vector plots
- Animations



Plotting

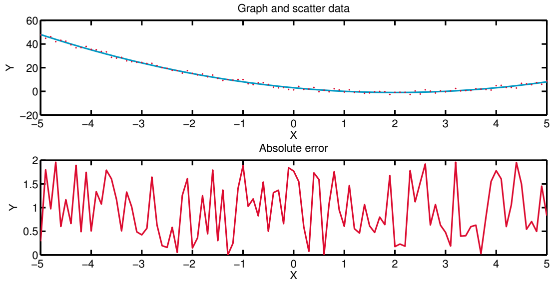
```
x = -5:0.1:5;
y = x.^2-4*x+3;
y2 = y + (2-4*rand(size(y)));
```

Plotting

```
x = -5:0.1:5;
y = x.^2-4*x+3;
y2 = y + (2-4*rand(size(y)));
subplot(2,1,1); plot(x,y,'-',x,y2,'r.');
xlabel('X'); ylabel('Y'); title('Graph and Scatter');
```

Plotting

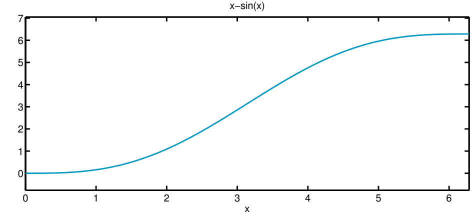
```
x = -5:0.1:5;
y = x.^2-4*x+3;
y2 = y + (2-4*rand(size(y)));
subplot(2,1,1); plot(x,y,'-',x,y2,'r.');
xlabel('X'); ylabel('Y'); title('Graph and Scatter');
subplot(2,1,2); plot(x,abs(y-y2),'r-');
xlabel('X'); ylabel('Y'); title('Absolute error');
```



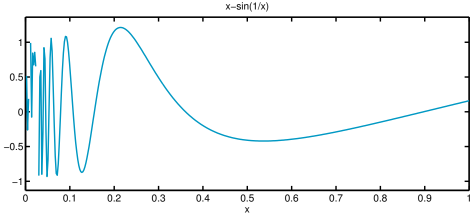
Plotting (2)

Easy plotting of functions can be done using the ezplot function:

`ezplot('x-sin(x)', [0 2*pi]):`

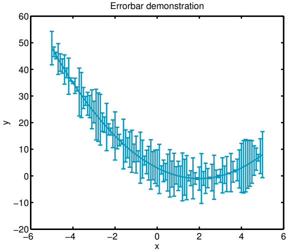


Be careful with steep gradients: `ezplot('x-sin(1/x)', [0 1])`



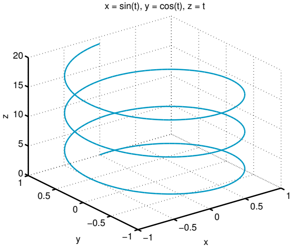
Other plotting tools

- Errorbars: `errorbar(x,y,err)`



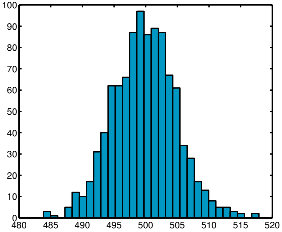
Other plotting tools

- Errorbars: `errorbar(x,y,err)`
- 3D-plots: `plot3(x,y,z)`



Other plotting tools

- Errorbars: `errorbar(x,y,err)`
- 3D-plots: `plot3(x,y,z)`
- Histograms: `histogram(x,20)`



Multi-dimensional data

Matlab typically requires the definition of rectangular grid coordinates using `meshgrid`:

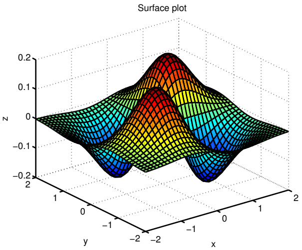
```
[x y] = meshgrid(-2:0.1:2,
    -2:0.1:2);
z = x .* y .* exp(-x.^2 - y.^2);
```

Multi-dimensional data

Matlab typically requires the definition of rectangular grid coordinates using `meshgrid`:

```
[x y] = meshgrid(-2:0.1:2,
    -2:0.1:2);
z = x .* y .* exp(-x.^2 - y.^2);
```

- Surface plot



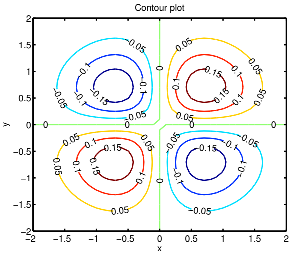

```
surf(x,y,z);
```

Multi-dimensional data

Matlab typically requires the definition of rectangular grid coordinates using `meshgrid`:

```
[x y] = meshgrid(-2:0.1:2,
    -2:0.1:2);
z = x .* y .* exp(-x.^2 - y.^2);
```

- Surface plot
- Contour plot



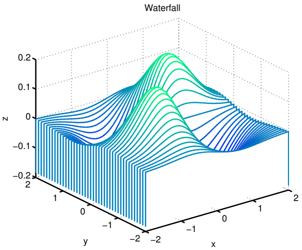

```
v=-0.5:0.05:0.5;
contour(x,y,z,v,'ShowText'
, 'on');
```

Multi-dimensional data

Matlab typically requires the definition of rectangular grid coordinates using `meshgrid`:

```
[x y] = meshgrid(-2:0.1:2,
    -2:0.1:2);
z = x .* y .* exp(-x.^2 - y.^2);
```

- Surface plot
- Contour plot
- Waterfall



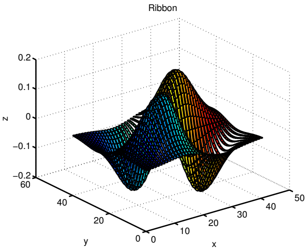

```
waterfall(x,y,z);
colormap(winter);
```

Multi-dimensional data

Matlab typically requires the definition of rectangular grid coordinates using `meshgrid`:

```
[x y] = meshgrid(-2:0.1:2,
    -2:0.1:2);
z = x .* y .* exp(-x.^2 - y.^2);
```

- Surface plot
- Contour plot
- Waterfall
- Ribbons



`ribbon(z);`

Vector data

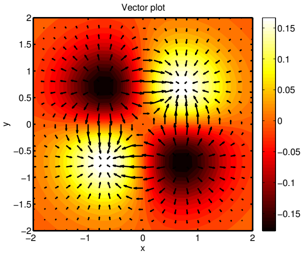
The gradient operator, as expected, is used to obtain the gradient of a scalar field. Colors can be used in the background to simultaneously plot field data:

```
[x y] = meshgrid(-2:0.2:2,
    -2:0.2:2);
z = x .* y .* exp(-x.^2 - y.^2)
[dx dy] = gradient(z,8,8)

% Background
contourf(x,y,z,30,'LineColor','
    none');
colormap(hot); colorbar;

axis tight; hold on;

% Vectors
quiver(x,y,dx,dy,'k');
```



Today's outline

Functions: revisited

In MATLAB you can define your own functions to re-use certain functionalities. We now define the mathematical function $f = x^2 + e^x$:

```
function y = f(x)
y = x.^2 + exp(x);
```

Functions: revisited

In MATLAB you can define your own functions to re-use certain functionalities. We now define the mathematical function $f = x^2 + e^x$:

```
function y = f(x)
y = x.^2 + exp(x);
```

Note:

- The first line of the file has to contain the `function` keyword
- The variables used are *local*. They will not be available in your Workspace
- The file needs to be saved with the same name as the function, i.e. “f.m”
- The semi-colon prevents that at each function evaluation output appears on the screen
- If `x` is an array, then `y` becomes an array of function values.

Anonymous functions

If you do not want to create a file, you can create an *anonymous function*:

```
>> f = @(x) (x.^2+exp(x))
```

Anonymous functions

If you do not want to create a file, you can create an *anonymous function*:

```
>> f = @(x) (x.^2+exp(x))
```

- `f`: the name of the function
- `@`: the function handle
- `x`: the input argument
- `x.^2+exp(x)`: the actual function

Anonymous functions

If you do not want to create a file, you can create an *anonymous function*:

```
>> f = @(x) (x.^2+exp(x))
```

- `f`: the name of the function
- `@`: the function handle
- `x`: the input argument
- `x.^2+exp(x)`: the actual function

```
>> f(0:0.1:1)
```

Using function handles

A function handle points to a function. It behaves as a variable

```
>> myFunctionHandle = @exp
>> myFunctionHandle(1)
```

Using function handles

A function handle points to a function. It behaves as a variable

```
>> myFunctionHandle = @exp
>> myFunctionHandle(1)
```

Used a.o. for passing a function to another function, for instance for optimization functions.

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

Using function handles

A function handle points to a function. It behaves as a variable

```
>> myFunctionHandle = @exp
>> myFunctionHandle(1)
```

Used a.o. for passing a function to another function, for instance for optimization functions.

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

Matlab offers a function `fzero` that can find the roots of a function in a certain range:

```
>> f = @(x) x.^3 - x.^2 - 3*atan(x) + 1;
>> fzero(f,[-2 2])
>> ezplot(f)
>> f(ans)
```

Practice function handles

Consider the function

$$f(x) = -x^2 - 3x + 3 + e^{x^2}$$

The built-in Matlab function `fminbnd` allows to find the minimum of a function in a certain range. Find the minimum of $f(x)$ on $-2 \leq x \leq 2$. Example usage:

```
x = fminbnd(fun,x1,x2)
```

Practice function handles

Consider the function

$$f(x) = -x^2 - 3x + 3 + e^{x^2}$$

The built-in Matlab function `fminbnd` allows to find the minimum of a function in a certain range. Find the minimum of $f(x)$ on $-2 \leq x \leq 2$. Example usage:

```
x = fminbnd(fun,x1,x2)
```

Answer using an anonymous function:

```
>> f = @(x) -x.^2 - 3*x + 3 + exp(x.^2)
>> ezplot(f,[-2 2])
>> fminbnd(f,-2,2)
>> f(ans)
```

Practice function handles

Consider the function

$$f(x) = -x^2 - 3x + 3 + e^{x^2}$$

The built-in Matlab function `fminbnd` allows to find the minimum of a function in a certain range. Find the minimum of $f(x)$ on $-2 \leq x \leq 2$. Example usage:

```
x = fminbnd(fun,x1,x2)
```

Answer using an anonymous function:

```
>> f = @(x) -x.^2 - 3*x + 3 + exp(x.^2)
>> ezplot(f,[-2 2])
>> fminbnd(f,-2,2)
>> f(ans)
```

Various related functions are `fzero`, `feval`, `fsolve`, `fminsearch`. They will be discussed later in the course.

Today's outline

Solver and goal-seek

Excel comes with a goal-seek and solver function. For Excel 2010:

- Install via Excel ⇒ File ⇒ Options ⇒ Add-Ins ⇒ Go (at the bottom) ⇒ Select solver add-in. You can now call the solver screen on the 'data' menu ('Oplosser' in Dutch)
- Select the goal-cell, and whether you want to minimize, maximize or set a certain value
- Enter the variable cells; Excel is going to change the values in these cells to get to the desired solution
- Specify the boundary conditions (e.g. to keep certain cells above zero)
- Click 'solve' (possibly after setting the advanced options).

Goal-seek: a simple example

Goal-Seek can be used to make the goal-cell to a specified value by changing another cell:

- Open Excel and type the following:

	A	B
1	x	3
2	f(x)	=–3*B1^2–5*B1+2
3		

- Go to Data ⇒ What-If Analysis ⇒ Goal Seek...
  - Set cell: B2
  - To value: 0
  - By changing cell: B1
- OK. You find a solution of 0.333 . . . .

Goal-seek: a simple example

Goal-Seek can be used to make the goal-cell to a specified value by changing another cell:

- Open Excel and type the following:

	A	B
1	x	3
2	f(x)	=−3*B1^2−5*B1+2
3		

- Go to Data ⇒ What-If Analysis ⇒ Goal Seek...
  - Set cell: B2
  - To value: 0
  - By changing cell: B1
- OK. You find a solution of 0.333 . . ..

Goal-seek: a simple example

Goal-Seek can be used to make the goal-cell to a specified value by changing another cell:

- Open Excel and type the following:

	A	B
1	x	3
2	f(x)	=–3*B1^2–5*B1+2
3		

- Go to Data ⇒ What-If Analysis ⇒ Goal Seek...
  - Set cell: B2
  - To value: 0
  - By changing cell: B1
- OK. You find a solution of 0.333 . . . .

Goal-seek: a simple example

Goal-Seek can be used to make the goal-cell to a specified value by changing another cell:

- Open Excel and type the following:

	A	B
1	x	3
2	f(x)	=–3*B1^2–5*B1+2
3		

- Go to Data ⇒ What-If Analysis ⇒ Goal Seek...
  - Set cell: B2
  - To value: 0
  - By changing cell: B1
- OK. You find a solution of 0.333 . . ..

Solver: a simple example

The solver is used to change the value in a goal-cell, by changing the values in 1 or more other cells while keeping boundary conditions:

- Use the following sheet:

	A	B	C
1		x	f(x)
2	x1	3	=2*B2*B3–B3+2
3	x2	4	=2*B3–4*B2-4

- Go to Data ⇒ Solver
    - Goalfunction: C2 (value of: 0)
    - Add boundary condition: C3 = 0
    - By changing cells: $B$2:$B$3 (you can just select the cells)
- Solve. You will find B2=0 and B3=2.

Solver: a simple example

The solver is used to change the value in a goal-cell, by changing the values in 1 or more other cells while keeping boundary conditions:

- Use the following sheet:

	A	B	C
1		x	f(x)
2	x1	3	=2*B2*B3–B3+2
3	x2	4	=2*B3–4*B2-4

- Go to Data ⇒ Solver
  - Goalfunction: C2 (value of: 0)
  - Add boundary condition: C3 = 0
  - By changing cells: $B$2:$B$3 (you can just select the cells)
- Solve. You will find B2=0 and B3=2.

Solver: a simple example

The solver is used to change the value in a goal-cell, by changing the values in 1 or more other cells while keeping boundary conditions:

- Use the following sheet:

	A	B	C
1		x	f(x)
2	x1	3	=2*B2*B3–B3+2
3	x2	4	=2*B3–4*B2-4

- Go to Data ⇒ Solver
  - Goalfunction: C2 (value of: 0)
  - Add boundary condition: C3 = 0
  - By changing cells: $B$2:$B$3 (you can just select the cells)
- Solve. You will find B2=0 and B3=2.

Solver: a simple example

The solver is used to change the value in a goal-cell, by changing the values in 1 or more other cells while keeping boundary conditions:

- Use the following sheet:

	A	B	C
1		x	f(x)
2	x1	3	=2*B2*B3–B3+2
3	x2	4	=2*B3–4*B2-4

- Go to Data ⇒ Solver
  - Goalfunction: C2 (value of: 0)
  - Add boundary condition: C3 = 0
  - By changing cells: $B$2:$B$3 (you can just select the cells)
- Solve. You will find B2=0 and B3=2.

Exercise

Use Excel functions to obtain the Antoine coefficients A, B and C for diethyl ether following the equation:

$$\ln P = A - \frac{B}{T + C}$$

P in kPa, T in °C. Experimental data is given (see Canvas for the xls):

P [mmHg]	T [K]
15.6	230.0
29.1	239.3
52.5	248.9
91.9	258.9
156.0	269.3
257.6	280.1
414.6	291.4
651.1	303.1
999.2	315.3
1501.0	328.0

Exercise

Use Excel functions to obtain the Antoine coefficients A, B and C for diethyl ether following the equation:

$$\ln P = A - \frac{B}{T + C}$$

P in kPa, T in °C. Experimental data is given (see Canvas for the xls):

P [mmHg]	T [K]
15.6	230.0
29.1	239.3
52.5	248.9
91.9	258.9
156.0	269.3
257.6	280.1
414.6	291.4
651.1	303.1
999.2	315.3
1501.0	328.0

1. Dedicate three separate cells for A, B and C. Give an initial guess

Exercise

Use Excel functions to obtain the Antoine coefficients A, B and C for diethyl ether following the equation:

$$\ln P = A - \frac{B}{T + C}$$

P in kPa, T in °C. Experimental data is given (see Canvas for the xls):

P [mmHg]	T [K]
15.6	230.0
29.1	239.3
52.5	248.9
91.9	258.9
156.0	269.3
257.6	280.1
414.6	291.4
651.1	303.1
999.2	315.3
1501.0	328.0

1. Dedicate three separate cells for A, B and C. Give an initial guess
2. Convert all values to proper units (hint: use e.g. =CONVERT(A2,“mmHg”,“Pa”))

Exercise

Use Excel functions to obtain the Antoine coefficients A, B and C for diethyl ether following the equation:

$$\ln P = A - \frac{B}{T + C}$$

P in kPa, T in °C. Experimental data is given (see Canvas for the xls):

P [mmHg]	T [K]
15.6	230.0
29.1	239.3
52.5	248.9
91.9	258.9
156.0	269.3
257.6	280.1
414.6	291.4
651.1	303.1
999.2	315.3
1501.0	328.0

1. Dedicate three separate cells for A, B and C. Give an initial guess
2. Convert all values to proper units (hint: use e.g. =CONVERT(A2,“mmHg”,“Pa”))
3. Compute $\ln P_{\text{exp}}$ and $\ln P_{\text{corr}}$

Exercise

Use Excel functions to obtain the Antoine coefficients A, B and C for diethyl ether following the equation:

$$\ln P = A - \frac{B}{T + C}$$

P in kPa, T in °C. Experimental data is given (see Canvas for the xls):

P [mmHg]	T [K]
15.6	230.0
29.1	239.3
52.5	248.9
91.9	258.9
156.0	269.3
257.6	280.1
414.6	291.4
651.1	303.1
999.2	315.3
1501.0	328.0

1. Dedicate three separate cells for A, B and C. Give an initial guess
2. Convert all values to proper units (hint: use e.g. =CONVERT(A2,"mmHg","Pa"))
3. Compute $\ln P_{\mathrm{exp}}$ and $\ln P_{\mathrm{corr}}$
4. Compute $(\ln P_{\mathrm{exp}} - \ln P_{\mathrm{corr}})^2$, and sum this column

Exercise

Use Excel functions to obtain the Antoine coefficients A, B and C for diethyl ether following the equation:

$$\ln P = A - \frac{B}{T + C}$$

P in kPa, T in °C. Experimental data is given (see Canvas for the xls):

P [mmHg]	T [K]
15.6	230.0
29.1	239.3
52.5	248.9
91.9	258.9
156.0	269.3
257.6	280.1
414.6	291.4
651.1	303.1
999.2	315.3
1501.0	328.0

1. Dedicate three separate cells for A, B and C. Give an initial guess
2. Convert all values to proper units (hint: use e.g. =CONVERT(A2,"mmHg","Pa"))
3. Compute $\ln P_{exp}$ and $\ln P_{corr}$
4. Compute $(\ln P_{exp} - \ln P_{corr})^2$, and sum this column
5. Start the solver, and minimize the sum by changing cells for A, B and C.

Today's outline

1. Coding in style
2. Error management
3. Visualisation
4. Functions: the sequel
5. Excel
6. Algorithms
7. Conclusions

Algorithm design

1. *Problem analysis*
   Contextual understanding of the nature of the problem to be solved
2. *Problem statement*
   Develop a detailed statement of the mathematical problem to be solved with the program
3. *Processing scheme*
   Define the inputs and outputs of the program
4. *Algorithm*
   A step-by-step procedure of all actions to be taken by the program (*pseudo-code*)
5. *Program the algorithm*
   Convert the algorithm into a computer language, and debug until it runs
6. *Evaluation*
   Test all of the options and conduct a validation study

Algorithm design

1. *Problem analysis*
   Contextual understanding of the nature of the problem to be solved
2. *Problem statement*
   Develop a detailed statement of the mathematical problem to be solved with the program
3. *Processing scheme*
   Define the inputs and outputs of the program
4. *Algorithm*
   A step-by-step procedure of all actions to be taken by the program (*pseudo-code*)
5. *Program the algorithm*
   Convert the algorithm into a computer language, and debug until it runs
6. *Evaluation*
   Test all of the options and conduct a validation study

Algorithm design

1. *Problem analysis*
   Contextual understanding of the nature of the problem to be solved
2. *Problem statement*
   Develop a detailed statement of the mathematical problem to be solved with the program
3. *Processing scheme*
   Define the inputs and outputs of the program
4. *Algorithm*
   A step-by-step procedure of all actions to be taken by the program (*pseudo-code*)
5. *Program the algorithm*
   Convert the algorithm into a computer language, and debug until it runs
6. *Evaluation*
   Test all of the options and conduct a validation study

Algorithm design

1. *Problem analysis*
   Contextual understanding of the nature of the problem to be solved
2. *Problem statement*
   Develop a detailed statement of the mathematical problem to be solved with the program
3. *Processing scheme*
   Define the inputs and outputs of the program
4. *Algorithm*
   A step-by-step procedure of all actions to be taken by the program (*pseudo-code*)
5. *Program the algorithm*
   Convert the algorithm into a computer language, and debug until it runs
6. *Evaluation*
   Test all of the options and conduct a validation study

Algorithm design

1. *Problem analysis*
   Contextual understanding of the nature of the problem to be solved
2. *Problem statement*
   Develop a detailed statement of the mathematical problem to be solved with the program
3. *Processing scheme*
   Define the inputs and outputs of the program
4. *Algorithm*
   A step-by-step procedure of all actions to be taken by the program (*pseudo-code*)
5. *Program the algorithm*
   Convert the algorithm into a computer language, and debug until it runs
6. *Evaluation*
   Test all of the options and conduct a validation study

Algorithm design

1. *Problem analysis*
   Contextual understanding of the nature of the problem to be solved
2. *Problem statement*
   Develop a detailed statement of the mathematical problem to be solved with the program
3. *Processing scheme*
   Define the inputs and outputs of the program
4. *Algorithm*
   A step-by-step procedure of all actions to be taken by the program (*pseudo-code*)
5. *Program the algorithm*
   Convert the algorithm into a computer language, and debug until it runs
6. *Evaluation*
   Test all of the options and conduct a validation study

Example: finding the roots of a parabola

We are writing a program that finds for us the roots of a parabola. We use the form

$$y = ax^2 + bx + c$$

What is our program in pseudo-code?

Example: finding the roots of a parabola

We are writing a program that finds for us the roots of a parabola. We use the form

$$y = ax^2 + bx + c$$

What is our program in pseudo-code?

1. Input data (a, b and c)

Example: finding the roots of a parabola

We are writing a program that finds for us the roots of a parabola. We use the form

$$y = ax^2 + bx + c$$

What is our program in pseudo-code?

1. Input data (a, b and c)
2. Identify special cases ($a = b = c = 0$, $a = 0$)

Example: finding the roots of a parabola

We are writing a program that finds for us the roots of a parabola. We use the form

$$y = ax^2 + bx + c$$

What is our program in pseudo-code?

1. Input data (a, b and c)
2. Identify special cases ($a = b = c = 0$, $a = 0$)

   $a = b = c = 0$ Solution indeterminate

Example: finding the roots of a parabola

We are writing a program that finds for us the roots of a parabola. We use the form

$$y = ax^2 + bx + c$$

What is our program in pseudo-code?

1. Input data (a, b and c)
2. Identify special cases ($a = b = c = 0$, $a = 0$)

   $a = b = c = 0$ Solution indeterminate

   $a = 0$ Solution: $x = -\frac{c}{b}$

Example: finding the roots of a parabola

We are writing a program that finds for us the roots of a parabola. We use the form

$$y = ax^2 + bx + c$$

What is our program in pseudo-code?

1. Input data (a, b and c)
2. Identify special cases ($a = b = c = 0$, $a = 0$)

   $a = b = c = 0$ Solution indeterminate

   $a = 0$ Solution: $x = -\frac{c}{b}$
3. Find $D = b^2 - 4ac$

Example: finding the roots of a parabola

We are writing a program that finds for us the roots of a parabola. We use the form

$$y = ax^2 + bx + c$$

What is our program in pseudo-code?

1. Input data (a, b and c)
2. Identify special cases ($a = b = c = 0$, $a = 0$)

   $a = b = c = 0$ Solution indeterminate

   $a = 0$ Solution: $x = -\frac{c}{b}$
3. Find $D = b^2 - 4ac$
4. Decide, based on D:

Example: finding the roots of a parabola

We are writing a program that finds for us the roots of a parabola. We use the form

$$y = ax^2 + bx + c$$

What is our program in pseudo-code?

1. Input data (a, b and c)
2. Identify special cases ($a = b = c = 0$, $a = 0$)

   $a = b = c = 0$ Solution indeterminate

   $a = 0$ Solution: $x = -\frac{c}{b}$
3. Find $D = b^2 - 4ac$
4. Decide, based on D:

   $D < 0$ Display message: complex roots

Example: finding the roots of a parabola

We are writing a program that finds for us the roots of a parabola. We use the form

$$y = ax^2 + bx + c$$

What is our program in pseudo-code?

1. Input data (a, b and c)
2. Identify special cases ($a = b = c = 0$, $a = 0$)
   - $a = b = c = 0$ Solution indeterminate
   - $a = 0$ Solution: $x = -\frac{c}{b}$
3. Find $D = b^2 - 4ac$
4. Decide, based on D:
   - $D < 0$ Display message: complex roots
   - $D = 0$ Display 1 root value

Example: finding the roots of a parabola

We are writing a program that finds for us the roots of a parabola. We use the form

$$y = ax^2 + bx + c$$

What is our program in pseudo-code?

1. Input data (a, b and c)
2. Identify special cases ($a = b = c = 0$, $a = 0$)

   $a = b = c = 0$ Solution indeterminate

   $a = 0$ Solution: $x = -\frac{c}{b}$
3. Find $D = b^2 - 4ac$
4. Decide, based on D:

   $D < 0$ Display message: complex roots

   $D = 0$ Display 1 root value

   $D > 0$ Display 2 root values

Example: finding the roots of a parabola

```
function x = parabola(a,b,c)
% Catch exception cases
if (a==0)
    if(b==0)
        if(c==0)
            disp('Solution indeterminate'); return;
        end
        disp('There is no solution');
    end
    x = -c/b;
end

D = b^2 - 4*a*c;
if (D<0)
    disp('Complex roots'); return;
    else if (D==0)
        x = -b/(2*a);
        else if (D>0)
                x(1) = (-b + sqrt(D))/(2*a);
                x(2) = (-b - sqrt(D))/(2*a);
                x = sort(x);
        end
    end
end
```

Example: finding the roots of a parabola

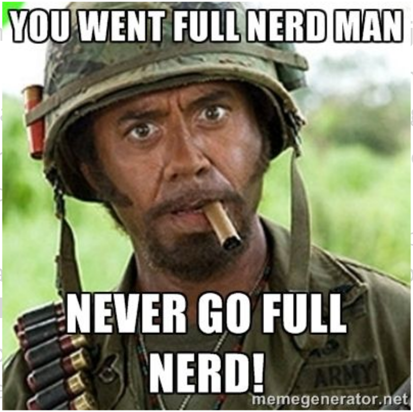
```
>> roots([1 -4 -3])
ans =
     4.6458
    -0.6458
```

Advanced concepts

- Object oriented programming: classes and objects
- Memory management: some programming languages require you to allocate computer memory yourself (e.g. for arrays)
- External libraries: in many cases, someone already built the general functionality you are looking for
- Compiling and scripting ("interpreted"); compiling means converting a program to computer-language before execution. Interpreted languages do this on the fly.
- Profiling, optimization, parallellization: Checking where your program spends the most of its time, optimizing (or parallellizing) that part.

Advanced concepts

- Object orien jects
- Memory ma nguages require you to alloc g. for arrays)
- External libr eady built the general func
- Compiling a piling means converting a before execution. Interpreted l
- Profiling, op ing where your program spe ng (or parallellizing) that part.



In conclusion...

- Algorithm design: define your problem, think ahead, make a scheme, sketch the interplay between variables and functions, then start programming
- Programming basics: variables, operators and functions, locality of variables, recursive operations
- Dealing with complex programs, verification of your algorithms, use of the debugger
- Visualisation: how to make 1D and 2D/3D plots, create a sensible and intuitive presentation of your data.
- Examples: a few practice cases

In conclusion...

- Algorithm design: define your problem, think ahead, make a scheme, sketch the interplay between variables and functions, then start programming
- Programming basics: variables, operators and functions, locality of variables, recursive operations
- Dealing with complex programs, verification of your algorithms, use of the debugger
- Visualisation: how to make 1D and 2D/3D plots, create a sensible and intuitive presentation of your data.
- Examples: a few practice cases

In conclusion...

- Algorithm design: define your problem, think ahead, make a scheme, sketch the interplay between variables and functions, then start programming
- Programming basics: variables, operators and functions, locality of variables, recursive operations
- Dealing with complex programs, verification of your algorithms, use of the debugger
- Visualisation: how to make 1D and 2D/3D plots, create a sensible and intuitive presentation of your data.
- Examples: a few practice cases

In conclusion...

- Algorithm design: define your problem, think ahead, make a scheme, sketch the interplay between variables and functions, then start programming
- Programming basics: variables, operators and functions, locality of variables, recursive operations
- Dealing with complex programs, verification of your algorithms, use of the debugger
- Visualisation: how to make 1D and 2D/3D plots, create a sensible and intuitive presentation of your data.
- Examples: a few practice cases

In conclusion...

- Algorithm design: define your problem, think ahead, make a scheme, sketch the interplay between variables and functions, then start programming
- Programming basics: variables, operators and functions, locality of variables, recursive operations
- Dealing with complex programs, verification of your algorithms, use of the debugger
- Visualisation: how to make 1D and 2D/3D plots, create a sensible and intuitive presentation of your data.
- Examples: a few practice cases