Introduction to computers and Matlab at CSC

Goal
Participants students and lecturer
Lessions and labs
Timetable
Outline
Resources

Goal

To prepare students to use the Unix-system at CSC for problem solving with Matlab. In the next course next week you are supposed to solve numerical problems in Matlab on the computer. The course is non-compulsory and without examination.

Lecturer

Staffan Romberger, srom@csc.kth.se, room 1615, Osquars backe 2, floor 6, phone 790 8164.

Students

- Can you program?
- Can you program in Matlab?
- Do you know fundamentals of Unix?
- Do you have access to a computer outside KTH?

Lessions and labs

We have 4 2-hour lessions for presentation of new material, advices about study, demonstations, questions.
We have 4 2-hour lab sessions in a computer room for work with the computer, with access to an assisting teacher. You are recommended to work in groups of two students with almost equal skill and experience.
It is adviceable to work also outside these hours at home or in the comuter room.

Timetable

day	lesson in room 4523	lab in room Brun
Tuesday 26 August	13–15	15–17
Wednesday 27 August	10–12	13–15
Thursday 28 August	10–12	13–15
Friday 29 August	10–12	13–15

Outline

Lesson 1: Introduction, Unix, terminal, e-mail, web, printing, access from outside KTH, command window, formulae, display of numbers, priority of operators, built-in functions, variables, workspace, arrays
Lab 1: Ch. 1, ch. 2 (esp. 2.1, 2.6, 2.10, 2.14).
Lesson 2: Polynomials, data types, editor window, command file, directory, printing, conditional commands, if, logical values, else, elseif, switch, try-catch, loops, for, while, break, continue.
Lab 2: Ch. 3, ch. 4 (esp. 3.3, 3.7, 4.7a, 4.9a, 4.19).

Lesson 3: Functions, function file, input parameter, output parameter, formal and actual parameter, local, global, persistent variable, return, local function, variable number of parameters, debugging.
Lab 3: Ch. 5, ch. 6 (esp. 5.2, 5.9, 5.17, 5.24).
Lesson 4: Functions as parameters. Other types of data. Other parts of Matlab as cell array, structure array, symbolic algebra, file handling, objects, graphical user interfaces (GUI).
Lab 4: Ch. 7, ch. 8, (ch. 8, ch. 9, ch. 10) (esp. 6.11, 6.12, 6.22, 7.3, 8.8, 8.11).

Resources

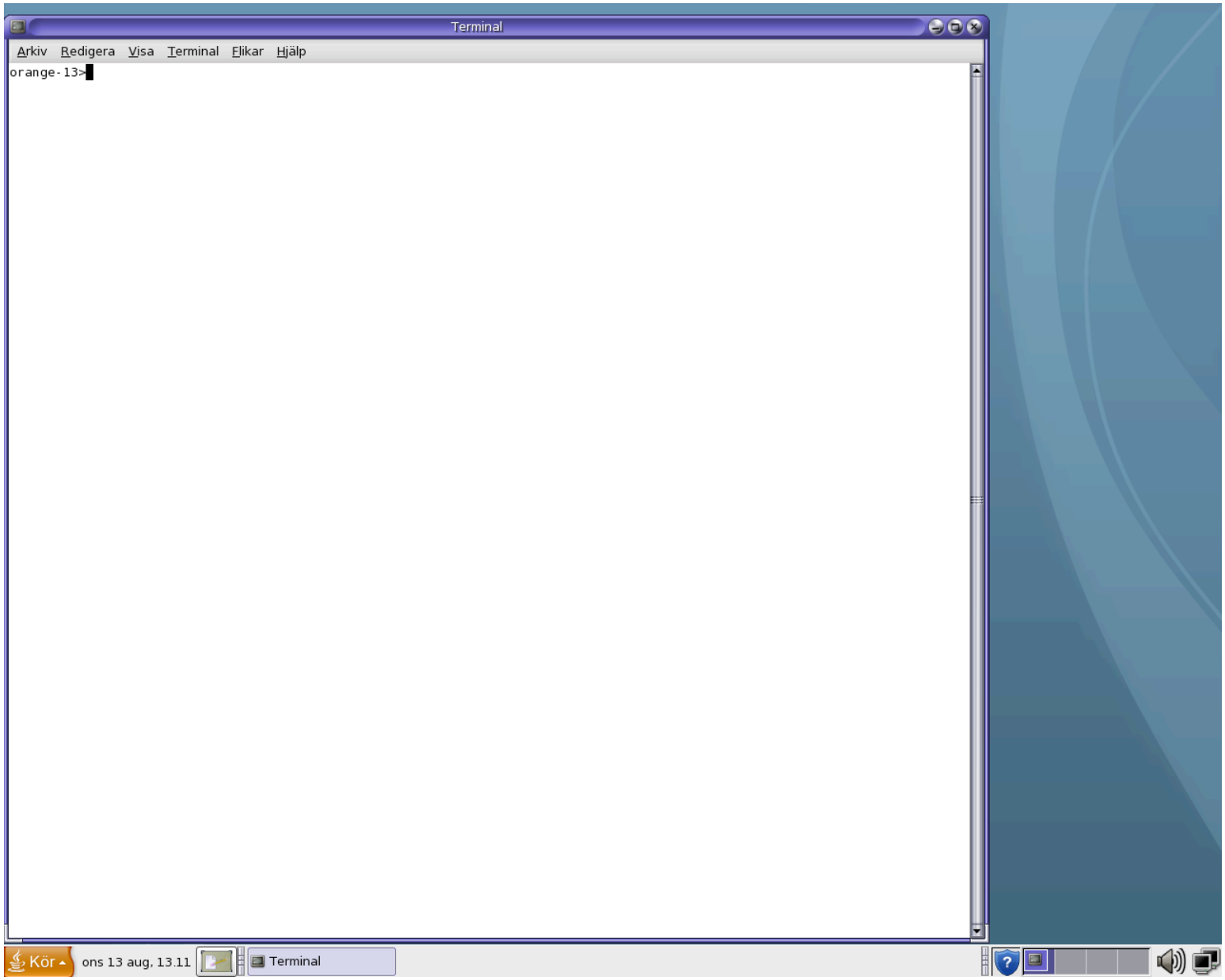
Computer rooms on floor 4, west wing. See http://lokal2.timeedit.se/kth.
Delphi help desk Osquars backe 2, floor 2, see opening hours. Web site: http://www.sgr.nada.kth.se/index.html.en.
Stephen J Chapman: Matlab programming for engineers, latest edition 4th (THS bookshop).

David F Griffiths: An introduction to Matlab. University of Dundee (www.maths.dundee.ac.uk/~ftp/na-reports/matlabNotes.pdf, Students office, Osquars backe 2, floor 2).
Refresher: www.nada.kth.se/~katarina/refresher.pdf.
Course web site: http://www.nada.kth.se/~katarina/master08.html.

Computer room, Sun computers

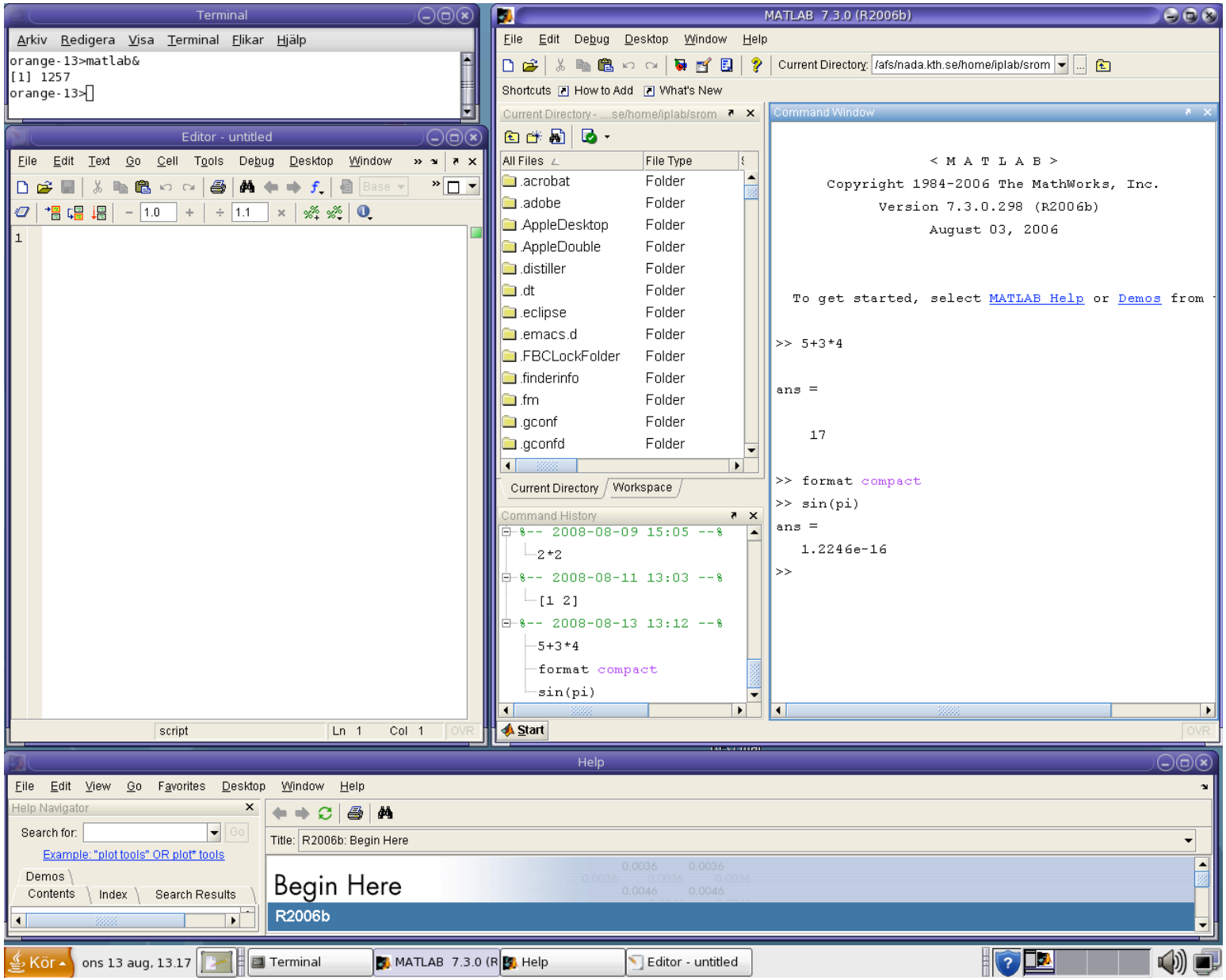
Login with your username and password.

You will get a graphical interface, JDS (Java desktop system), with a desktop with a window for terminal that accepts Unix commands, the panel and possibly other windows.


Terminal
Arkiv Redigera Visa Terminal Flikar Hjälp
orange-13>
Kör
ons 13 aug, 13.11
Terminal

You can start Matlab with `>matlab&` i.e. type `matlab&` at the prompt. The & signifies that Matlab is run i a process of its own and that control returns to terminal when Matlab is started.

After some typing the screen looks like this:


Terminal
Arkiv Redigera Visa Terminal Flikar Hjälp
orange-13>matlab&
[1] 1257
orange-13>
Editor - untitled
File Edit Text Go Cell Tools Debug Desktop Window
script Ln 1 Col 1 OVR
MATLAB 7.3.0 (R2006b)
File Edit Debug Desktop Window Help
Current Directory: /afs/nada.kth.se/home/iplab/srom
Shortcuts How to Add What's New
Current Directory - ....se/home/iplab/srom
All Files File Type
.acrobat Folder
.adobe Folder
.AppleDesktop Folder
.AppleDouble Folder
.distiller Folder
.dt Folder
.eclipse Folder
.emacs.d Folder
.FBCLockFolder Folder
.finderinfo Folder
.fm Folder
.gconf Folder
.gconfd Folder
Current Directory Workspace
Command History
%-- 2008-08-09 15:05 --%
2*2
%-- 2008-08-11 13:03 --%
[1 2]
%-- 2008-08-13 13:12 --%
5+3*4
format compact
sin(pi)
Command Window
< M A T L A B >
Copyright 1984-2006 The MathWorks, Inc.
Version 7.3.0.298 (R2006b)
August 03, 2006
To get started, select MATLAB Help or Demos from
>> 5+3*4
ans =
17
>> format compact
>> sin(pi)
ans =
1.2246e-16
>>
Start
OVR
Help
File Edit View Go Favorites Desktop Window Help
Help Navigator
Search for:
Example: "plot tools" OR plot* tools
Demos
Contents Index Search Results
Title: R2006b: Begin Here
Begin Here
R2006b
Kör
ons 13 aug, 13.17
Terminal
MATLAB 7.3.0 (R
Help
Editor - untitled

To log out choose Quit Matlab in Matlab's File menu and Logout in the launch menu of the panel.

I will henceforth run Matlab on the Macintosh. It works practically the same.

It is possible to rent Matlab for use at home, see http://progdist.ug.kth.se/public/index.shtml. You can access your files at KTH from home by ftp with Fugu on Macintosh or Putty on Windows or you can move files on USB-memory, see http://www.sgr.nada.kth.se/delfi/docs/USB-minne.html.

Matlab

Formulas

You can use the command window as a calculator. The value of a formula is displayed. The value is also assigned to the variable `ans`. You can change the format of the displayed value with `format`, e.g. `format compact`.

```
2 * 2
2 – 3*5
pi
e
i
```

Formulas are interpreted as expected. Matlab evaluates a formula with the following operator priority:

subformulas within `( )`

unary `+, –`

`^`

`*, /`

binary `+, –`

Adjacent operators with the same priority are evaluated from left to right.

Matlab has many built-in functions:

```
exp(1)
format long
sin(pi/4)
1/sqrt(2)
sind(45)
log(exp(1))
sqrt(-2)
```

You can use variables to store values. Variables have names containing letters and digits and letter case is significant. With `who`, `whos` and `clear` you can manipulate variables.

```
e = exp(1);
log(e)
```

```
r = 2;
area = pi*r^2
```

The display of the value of a command is suppressed when the kommand end with a semicolon ";". You may type several commands om a line. Commands ended with comma "," display their value. A command on a line ended with "…" continues with the next line.

```
a = 1, b = 2; c = 3
s = B/(w0^2*sqrt((1-w^2/w0^2)^2+...
    (w/Q*w0)^2))
```

How do you find information in Matlab? `help` *functionname* gives information about the function. `lookfor` *keyword* gives information about functions that contain the keyword in their description. The help browser (`doc`) contains extensive documentation of Matlab.

```
help log
lookfor logarithm
doc startup
edit startup
```

Arrays

Matlab (matrix laboratory) was invented by Clive Mohler to describe and carry out by computer, common matrix (array) calculations.
Most values in Matlab are arrays with elements. By now we have only seen 1×1-arrays of numbers, scalars. We will later see different kinds of numbers, logical values (false and true), characters and symbolic algebra entities, and other kinds of arrays such as cell arrays, and structure arrays.

```
A = [1 2 3; 4 5 6; 7 8 9]
b = [10 11 12]
c = [13; 14; 15]
size(A), size(b), size(c)
```

Arrays can be constructed by "`[ ]`" where space or "`,`" denotes new column and "`;`" or return denotes new row. Row and column vectors are considered 1×*n* and *n*×1 arrays respectively.
Single elements can be accessed with indices:

```
A(2, 3)
b(2) = 4
b(1, 2)
```

Any array can be treated as a kolumn vector (with just one index):

```
A(4)
A(8) = 10
```

In Matlab you can also denote subarrays:

```
A([1 2],[2 3])
A(end,2)
A(:,2) = [7 7 7]
A(:,2) = 6
A(1:4:9) = 1;
A([3 5 7]) = [1 2 3]
A = [1 4 3
     2 2 8
     1 6 1]
A([1 3],[1 3]) = 0
```

```
A = [0 4 0
     2 2 8
     0 6 0]
```

You kan use arrays in construction of new arrays. Create `xnew` as `[9 10 0 15]`:

```
A = [1 2;3 4]; B = [5 6;7 8]; x = [9 10];
y = [11;12]; z = [13 14];
xnew = x; xnew(3) = 0; xnew(4) = 15;
xnew = [x 0 15];
temp = [0 15]; xnew = [x temp];
Anew = [A;z];
Anew = [A;[13 14]];
Anew = [A y];
Anew = [A [11;12]];
```

There are many ways to create certain arrays:

```
a = 2:5
b = 1:3:10
```

```
c = []
d = 5:2:4
e = 10:-1:1
linspace(0, 12, 6)
linspace(0, 12)
ones(2)
ones(2, 3)
zeros(2)
zeros(2, 3)
eye(2)
eye(2, 3)
```

Some operators (`+, -`) require array operands of equal size or that one operand is scalar (elementwise operators):

```
[1 2; 3 4]+[5 5; 6 6]
[1 2 3]-1
-[3; 4]
```

`.*`, `./`, `.\` and `.^` are also elementwise operators:

```
[1 2; 3 4].*[2 2; 3 3]
[1 2; 3 4]./[2 2; 3 3]
[1 2; 3 4].\[2 2; 3 3]
[1 2; 3 4].^[2 2; 3 3]
```

Transpose an array with `.'` , or `'` , the latter creates conjugate transpose.

`a*b` denotes matrix multiplication, i.e. last dimension of `a` must have the same size as first dimension of `b`.

`x = A\b` gives the solution to `A*x = b` or `x = inv(A)*b` and `A/B` is `A*inv(B)`. `A^n` denotes `A` multiplied with itself `n` times where `A` has to be square.

Diagrams

To plot $y = \sin(x)$ for $-2\pi \leq x \leq 2\pi$:

```
x = -2*pi:0.1:2*pi
y = sin(x)
plot(x,y);
```

To draw a bar diagram of a vector:

```
val = [15 17 10 2 13 19]:
bar(val);
```

At the computer

Log in and start Matlab. Try selected examples from this lession.
Try exercises from the book, especially those also on the handout.
Ask your mates or the teacher when you need help.

Lession 2

Questions, comments about yesterday

Polynomials

Matlab stores polynomials as a coefficient vector with the coefficient of highest degree first. With $p(x)=x^4+5x^3-2x^2+7x-11$ can we calculate:

```
p = [1 5 -2 7 -11]; polyval(p,1)
val = polyval(p,[2 3 5 7 9])
```

Data types

A data type is one way to store data/interpret the content of computer memory as a value. In some programming languages you have to give the data type of a variable when you create it. In Matlab a variable can have different data type at different times. Values are stored as bit sequences in the computer memory. There are infinitely many numbers. Therefore infinite amout of memory should be given to each variable. Instead you

have different data types with different memory requirements for different purposes.

`int8`	integer	8 bits	–128..127
`int16`	integer	16 bits	–32 768..32 767
`int32`	integer	32 bits	–2 147 483 648..2 147 483 647
`single`	float	32 bits	approx. 3,4E–38..3,4E38 with 7 digits
`double`	float	64 bits	approx. 1,7E–308..1,7E308 with 15 digits
`uint8`	unsigned	8 bits	0..255
`uint16`	unsigned	16 bits	0..65 535
`uint32`	unsigned	32 bits	0..4 294 967 295
`logical`	not a data type		

Numbers are usually stored as `double`, but you can choose another data type.

```
d1 = 32; d2 = uint8(32); whos d1 d2
Name Size Bytes Class
d1    1x1     8 double array
```

```
d2     1x1      1 uint8 array
```

Command file

You can store a series of commands in a file with a name which ends in "`.m`". These commands are performed when the name is executed in the command window.

Conditional commands, if, switch

Often you want some commands to be performed only if som condition is true.

```
if condition
   commands
end
```

Instead you can make different kommands to be performed if the condition is true or false respectively.

```
if condition
   commands1
else
```

```
    commands2
end
```

One **if**-command can contain another.

```
if condition1
    commands1
    if condition2
        commands2
    end
else
    if condition3
        commands3
    else
        commands4
    end
    commands5
end
```

There shall be exactly one `end` for each `if`.

When a series of conditions are to be examined you can use `elseif`.

```
if condition1
   commands1
elseif condition2
   commands2
else
   commands3
end
```

```
if condition1
   commands1
else
   if condition2
      commands2
   else
      commands3
   end
end
```

There should not be an `end` to an `elseif`. You may omit the `else`-part:

```
if condition1
   commands1
elseif condition2
   commands2
end
```

You may continue a line with `else`, `elseif` or `end` preceeded by »,» or »;».

Logical values, logical expressions

Any numerical expression can be used as a condition. 0 is false, nonzero is true. True is stored as 1. An array is true if real part of every element is nonzero.

The relational operators `<`, `<=`, `>`, `>=`, `==` and `~=` give a logical value. You can compose logical values with the logical operators `~`, `&`, `|`, `&&` and `||` and the logical function `xor`. Many built-in functions return a logical value.

```
~a      a&b          a|b          xor(a,b)
a       a b 0 1      a b 0 1      a b 0 1
0 1     0   0 0      0   0 1      0   0 1
1 0     1   0 1      1   1 1      1   1 0
```

That the value of *k* is between 5 and 17 can be written 5<*k*<17 in mathematics. In Matlab it is written `5<k & k<17`. In Matlab `5<k<17` means something else:

```
k = 1; 5<k<17
ans = 1
```

First `5<k` is calculated, i.e. `5<1` which is false i.e. 0. Then `0<17` which is true, i.e. 1.

It is easy to mix up = (assign) and == (are equal).

Operands are usually evaluated before the operator. The short cut operators `&&` and `||` work differently.

```
% c = a && b
if ~a c = 0
else c = b
% c = a||b
if a c = 1
else c = b
```

Leapyear

In the Julian (Roman) calendar a year is a leap year if it is a multiple of 4. In the Gregorian calendar (used i Sweden from 1753) of years ending in "00" only those that are multiples of 400 are leap years.

Let `y` be a year. Set `leapjul` and `leapgreg` to `true` when `y` is a leap year according to Julian and Gregorian calendar respectively. A year is about 365.2422 days and the Gregorian calendar gives 365.2425 i.e. error of one day in 3 236 years.

```
leapjul = mod(y,4)==0;
leapgreg = leapjul & ~(mod(y,100)==0 & ...
mod(y,400)~=0);
% alternatively
leapgreg = leapjul & mod(y,100)~=0 | ...
mod(y,400)==0;
```

To test the program:

```
y = [1996 2000 2001 2004 2096 2100 2200 2300 2400 2500];
leapjul = mod(y,4)==0;
```

```
disp('leapjul = mod(y,4)==0'); disp([y;leapjul]);
leapgreg = leapjul & ~(mod(y,100)==0 & mod(y,400)~=0);
disp('leapgreg = leapjul & ~(mod(y,100)==0 & mod(y,400)~=0)');
% årtal delbara med 4 men inte sekelår som inte är delbara med 400
disp([y;leapgreg]); disp('mod(y,100)==0');
disp([y;mod(y,100)==0]); disp('mod(y,400)~=0');
disp([y;mod(y,400)~=0]);
disp('~(mod(y,100)==0 & mod(y,400)~=0)');
disp([y;~(mod(y,100)==0 & mod(y,400)~=0)]);
leapgreg = leapjul & mod(y,100)~=0 | mod(y,400)==0;
disp('leapgreg = leapjul & mod(y,100)~=0 | mod(y,400)==0');
% years multiples of 4, not multiples of 100 but of 400
disp([y;leapgreg]);
disp('mod(y,100)~=0'); disp([y;mod(y,100)~=0]);
disp('mod(y,400)==0'); disp([y;mod(y,400)==0]);
disp('leapjul & mod(y,100)~=0');
disp([y;leapjul & mod(y,100)~=0]);
```

Triangle analysis

Determine if a triangle with the sides a, b and c is equilateral, isosceles, scalene or non-existent.

```
% Analyses a triangle with sides a, b and c.
if a<=0 | b<=0 | c<=0 'non-existent'
```

```
  elseif a>=b+c | b>=c+a | c>=a+b 'non-existent'
  elseif a==b & b==c & c==a 'equilateral'
  elseif a==b | b==c | c==a 'isosceles'
  else 'scalene'
  end
```

Let's put the commands in the file tri.m. Then we can use it as a command file.

```
a = 0; b = 1; c = 2; tri % 0 1 2
ans = non-existent
a = 3; tri               % 3 1 2
ans = non-existent
a = 1.5; tri             % 1.5 1 2
ans = scalene
a = 2; tri               % 2 1 2
ans = isosceles
b = 2; tri               % 2 2 2
ans = equilateral
```

Switch

When you want to perform different command for different values of an expression you can use `switch` instead of `if`.

```
switch expression
case value1
   commands1
case value2
   commands2
...
otherwise
   commands
end
```

Values are expressions or sequences of values within “{}”, i.e. a cell array. If the value of the expression is not among the values the kommands after `otherwise` are performed. The `otherwise` part may be omitted.

Let `t` be the value of a throw of a dice:

```
switch t
case {1,3,5}
   'Odd number of dots'
case {2,4,6}
   'Even number of dots'
otherwise
   'Funny dice!'
end
```

Error handling, the try command

Some errors make Matlab display an error message.

```
a = [1]; a(3)
??? Index exceeds matrix dimensions.
```

You can catch the error, display another error message or try to correct the error.

```
a = [1];
try
```

```
    a(3)
  catch
    '??? Index error'
  end
  ??? Index error
```

You can create an error message with `error(`*error message*`)`. Matlab shows where the error occured and displays the message. If such an error occurs in a m-file that is called between `try` and `catch` the error message is not displayed but the command between `catch` and `end` are performed.

Some calculations doesn't display an error message, possibly a warning message or return a special error value as result. `3/0` and `2*realmax` both return the result `Inf` (∞) and 0/0 returns `NaN` (not a number). When you convert a value to a data type that doesn't contain that value the closest possible value is stored.

```
  int8(1000)
  ans = 127
```

Loops

To repeat the performance of some commands is common. When it is done for a predetermined sequence of values of a variable you use `for` and when you want to repeat while a contition is true you use `while`.

```
for variable = expression
   commands
end
```

The variable is called the control variabl of the loop. It is assigned the columns of the expression in order. The expression is evaluated once, when the loop is started. It usually is a colon expression, `b:s:e`.

Gear box

Let's design a gear box with a ratio as close to a given value, *gear*, as possible. You have a set of gear-wheel with from *min* to *max* cogs. Read *gear*, *min* and *max*.

- Read *min*, *max* and *gear*.
- Check that *min*, *max* and *gear* are greater than 0.

- Let *m* vary from *min* to *max*.
- Test the two *n* on each side of *m*/*gear*.
- Note the current best *m* and *n* i.e. those that gave the least value of abs(*m*/*n*-*gear*).
- Display the best values.

```
min = input('Smallest number of cogs: ')
max = input('Largest number of cogs: ')
gear = input('Requested gear: ')
bestdiff = realmax;
if gear>0 & max>0 & min>0
  for m = min:max
    nprel = floor(m/gear);
    for n = nprel:nprel+1
      if n<min n=min; end; if n>max n=max; end;
      diff = abs(m/n-gear);
      if diff<bestdiff, bestdiff = diff;
         bestm = m; bestn = n; end % if
```

```
    end % for n
  end % for m
  disp(['Best m, n and m/n are ' int2str(bestm)...
  ', ' int2str(bestn)' and ' ...
  num2str(bestm/bestn)]); end % if
```

Suppose the comands are in file gearbox.m

```
gearbox
Smallest number of cogs: 5
Largest number of cogs: 50
Requested gear: 3.14159
Best m, n and m/n are 22, 7 and 3.1429
```

The function `disp(`*string*`)` displays its argument without string delimiters (`'`). We can with the same result write:

```
  ['Best m, n and m/n are ' int2str(bestm) ...
  ', ' int2str(bestn) ' and ' ...
  num2str(bestm/bestn)]
```

While

`while` *condition*
commands
`end`

The condition is evaluated when the while command is reached and after each lap. The loop is left when the condition is false.

Loops and conditional command may contain other loops and conditional commands.

The command `break` causes closest surrounding loop to be finished and execution continues after its `end`. The command `continue` causes the current lap in the closest surrounding loop to finish and the next lap, if there is some, to start.

eps

A measure of the precision of a numerical data type is the smallest value *systeps*, for which 1+*systeps*≠1.

```
systeps = 1;
```

```
for i = 1:1000
   systeps = systeps/2;
   if systeps+1==1
      break
   end
end
systeps = systeps*2
systeps = 2.2204e-16
```

Mathlab has a built-in variable `eps` with this meaning.

```
eps = 2.2204e-16
```

Testing—is a program correct?

To determine if a program is correct is difficult.

- How do you describe (specify) what the program is supposed to do?
  How do you find out the clients or customers future wishes?

- You can systematically test different cases/inpu data. It is seldom feasible to test every case. However when you find an error you can correct it.
- You can reason systematically about a program and compare the reasoning with the specification.
- An *algorithm* (from Abu Ja‘far Mohammaed ibn Mûsâ al-Khowârizmî who in 825 published an algebra book *Kitab al jabr w’al-muqabala*) is a detailed description of how a problem is solved in i finite number of steps.

File io

Read numbers from the file in.dat until 0 is found or the end of the file is reached and print the positive numbers on the file out.dat.

You can read all values from a file to a variable with `load`. Vi want to read a number at a time. Open the file with `fopen` and if that is successful we will get a file handle, if not –1 is returned. After that we can read with `fscanf` and print with `fprintf`.

file handle = `fopen(`*filename*, *access*`)`
returns a file handle. Access can be omitted or given with `'r'` for reading, with `'w'` for writing from the beginning of the file or with `'a'` for wrining after the files former end (append).

```
frin = fopen('in.dat');
frout = fopen('out.dat','w');
% Preferrably test that fopen not returned -1
[x,count] = fscanf(frin,'%d',1);
if count>0
  while x~=0
    if x>0
       fprintf(frout,'%d\n',x);
    end
    [x,count] = fscanf(frin,'%d',1);
    if count==0 break, end
  end % while
end % if
```

```
  fclose(frin);
  fclose(frout);
```

Suppose the commands are in file io.m.

```
type in.dat
2 3 5 -7 11
13 -17 0 19 -23 29
io
type out.dat
2
3
5
11
13
```

Change in.dat.

```
type in.dat
2 3 5 -7 11
13 -17 19 -23 29
io
type out.dat
2
3
5
11
13
19
29
```

`[`*var*`,`*no_of_elements*`] = fscanf(`*file_handle*`,` *format_string*`,` *requested_no_of_elements*`)`
returns a variable vith values from the file and number of elements read. Parameter *format_string* bescribe how data on the file shall be interpreted (`%d` denotes integer) and *requested_no_of_elements* denotes how many values that are requested.

`fprintf(`*filr_handle*`,`*format_string*`,` *val1*`, ...)`
prints the values *val1*, … in the file according to *format_string* (`%d\n` denotes as integers with new line after each number).

Lession 3, functions

Funktionsfil, funktionsanrop
in- och utparametrar, anropsin- och anropsutparametrar
lokala, globala och persistenta variabler
lokala funktioner
return
variabelt antal parameterar, skönsvärde
slumptal
avlusning, felsökning (debugging), programprofil

A function is a sequence of commands in a m-file that begins with a function-head:

`function [`*outparameter1*`, …] = `*functionname*`(`*inparameter1*`, …)`

or

`function` *functionname*`(`*inparameter1*`, …)`

Directly after the function head you write documentation comments according to the rules for `lookfor` and `help`. After that are commands that are performed when the function is called. The function is called with:

functionname`(`*call_inparameter1*`, ...)`

eller

`[`*call_outparameter1*`, ...] = `*functionname*`(`*call_inparameter1*`, ...)`

When the commands in the function are finished execution continues after the call. A function usually is put in a file with extension ".m" and the name of the function.

Parameters

Parameters and other variables in a function are local and have nothing to do vith variables with the same names in other functions, commandfiles or the command window. Exceptions are *global* and *persistent* variables. You can write a function so that it can be called with different number of callparameter different times.

An m-file can contain more than one function. The functions after the first are called *local functions* and can be called only by functions in the same file.

Let's change the kommandfile gearbox.m to a function:

```
function [m n finalgear] = findgear(gear,min,max)
% Finds the pair of cogweels that best approximate
a given gear ratio.
% [m n finalgear] = findgear(gear,min,max)
% finds the best approximation m/n of gear
% for m and n between min and max for positive
% min, max and gear.
if gear>0 & max>0 & min>0
  bestdiff = realmax;
  for m = min:max
    nprel = floor(m/gear);
    for n = nprel:nprel+1
      if n<min n=min; end; if n>max n=max; end;
```

```
      diff = abs(m/n-gear);
      if diff<bestdiff
         bestdiff = diff; bestm = m; bestn = n;
      end % if
    end % for n
  end % for m
  m = bestm; n = bestn; finalgear = m/n;
else error('Indata should be positive.');
end
```

Let's test.:

```
findgear(3.14159,5,50)
ans = 22
[m n g] = findgear(3.14159,5,50)
m = 22
n = 7
g = 3.1429
findgear(3.14159,5,-50)
```

```
??? Error using ==> findgear
Indata should be positive.
```

Call inparameters are expressions. At call the values of these expressions are assigned to the corresponding inparameters. When the function finishes the values of the outparameters are assigned to the call outparameters. Call autparameter may be fewer than the outparameters. The function finishes after its last command or the last command before the next function has executed or when the command `return` is executed.

Let's change the function so that default values for *min* and *max* is 5 and 100 respectively. For handling of variable number of parameters we can use `nargin`, `nargout`, `nargcheck`, `error`, `warning` and `inputname`.

```
function [m n finalgear] = findgear(gear,min,max)
% Finds the pair of cogwheels that best
approximates a given gear ratio.
% [m n finalgear] = findgear(gear,min,max)
% finds the best approximation m/n of gear
```

```
% for m and m between min and max for positive
% min, max och gear. Default value for min and max
%  are 5 and 100 respectively.
msg = nargchk(1,3,nargin);
error(msg);
if nargin<3 max = 100; end
if nargin<2 min = 5; end
```

Let' test:

```
[m n g] = findgear(pi,25)
m = 88
n = 28
g = 3.1429
[m n g] = findgear(3.14159)
m = 22
n = 7
g = 3.1429
[m n g] = findgear
```

```
??? Error using ==> findgear
Not enough input arguments.
[m n g] = findgear(17,10,50,50)
??? Error using ==> findgear
Too many input arguments.
```

You can in similar ways make the function handle different numbers of call outparameters.

```
m = findgear(3.14159)
m = 22
[m n] = findgear(3.14159)
m = 22
n = 7
```

Global variable

Program a counter that can be incremented (with `count`) and inspected (with `getcount`). The counter variable has to be accessible from both functions. Ordinary variables are local. We will create a global variable

COUNTER. It is common practice to give global variables upper case names. In the file count.m:

```
function count
% Increments COUNTER
global COUNTER
if isempty(COUNTER) COUNTER = 0; end
COUNTER = COUNTER+1;
```

In the file getcout.m:

```
function c = getcount
% Returns the value of COUNTER
global COUNTER
c = COUNTER;
```

Let's test:

```
getcount
ans = []
count; count; getcount
ans = 2
```

```
  COUNTER
  ??? Undefined function or variable 'COUNTER'
```

A variable you want to save between calls but only needs to be accessed in one function can be made `persistent`.

Random walk

Program Brownian movement with equal probability for any direction, movement the distance 1 per stp and start at the origin. Let brown.m be:

```
function pos = brown(steps,runs)
% Simulates Brownian movement.
% pos = brown(steps) draws a walk of steps steps.
% pos = brown(steps,runs) draws the final points
% of runs walks with steps steps each.
msg = nargchk(1,2,nargin);
error(msg);
if nargin==1
  pos = b(steps,1);
```

```
else
  pos = [];
  for run = 1:runs
    pos = [pos b(steps,0)];
  end
  plot(pos(1,:),pos(2,:),'ko');
  pos = pos(:,end);
end
```

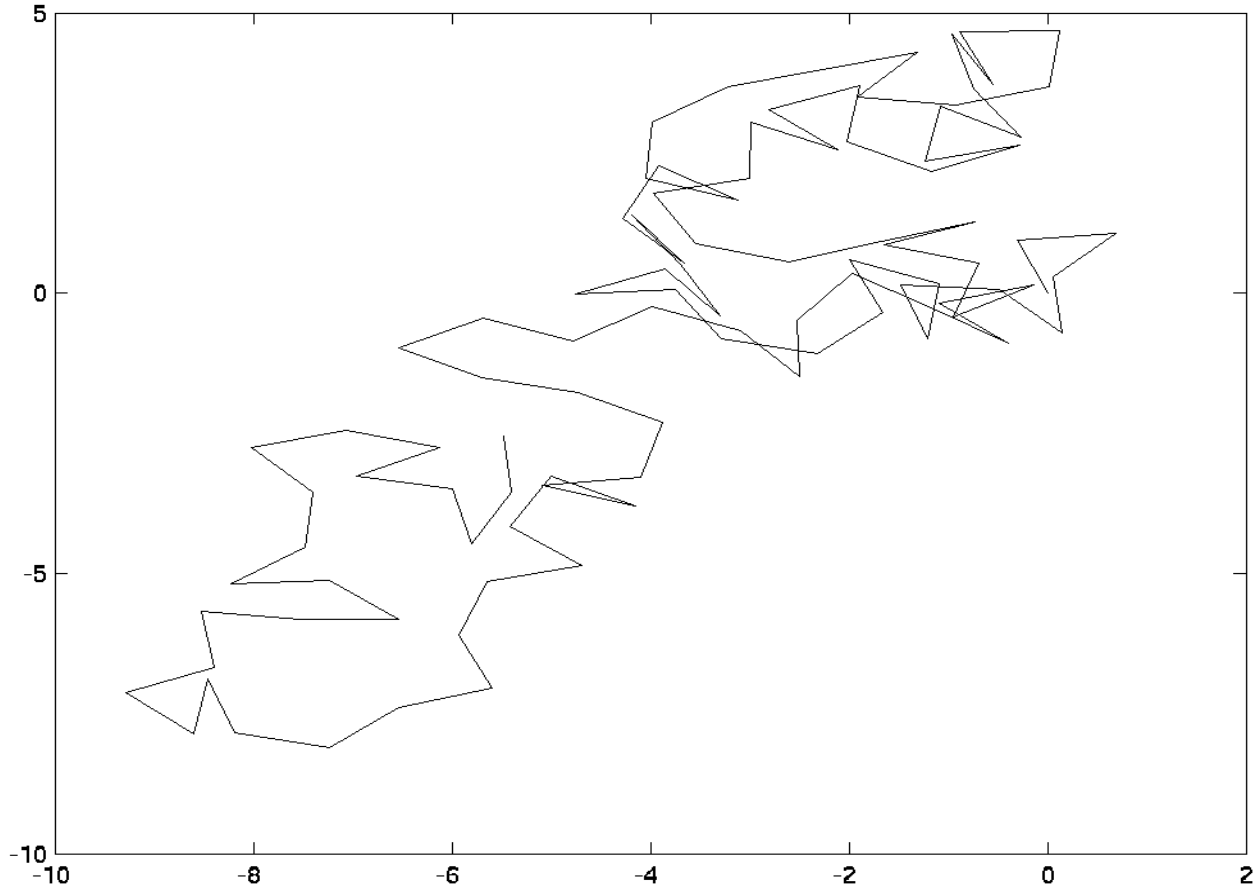
After brown in the same file, as a local function we insert:

```
function pos = b(steps,show)
% Simulates brownian movement and
% plots the walk when show is true.
pos = [0;0]; x = pos;
for i = 1:steps
  a = rand*2*pi;
  pos = x(:,end)+[cos(a);sin(a)];
  x = [x pos];
```

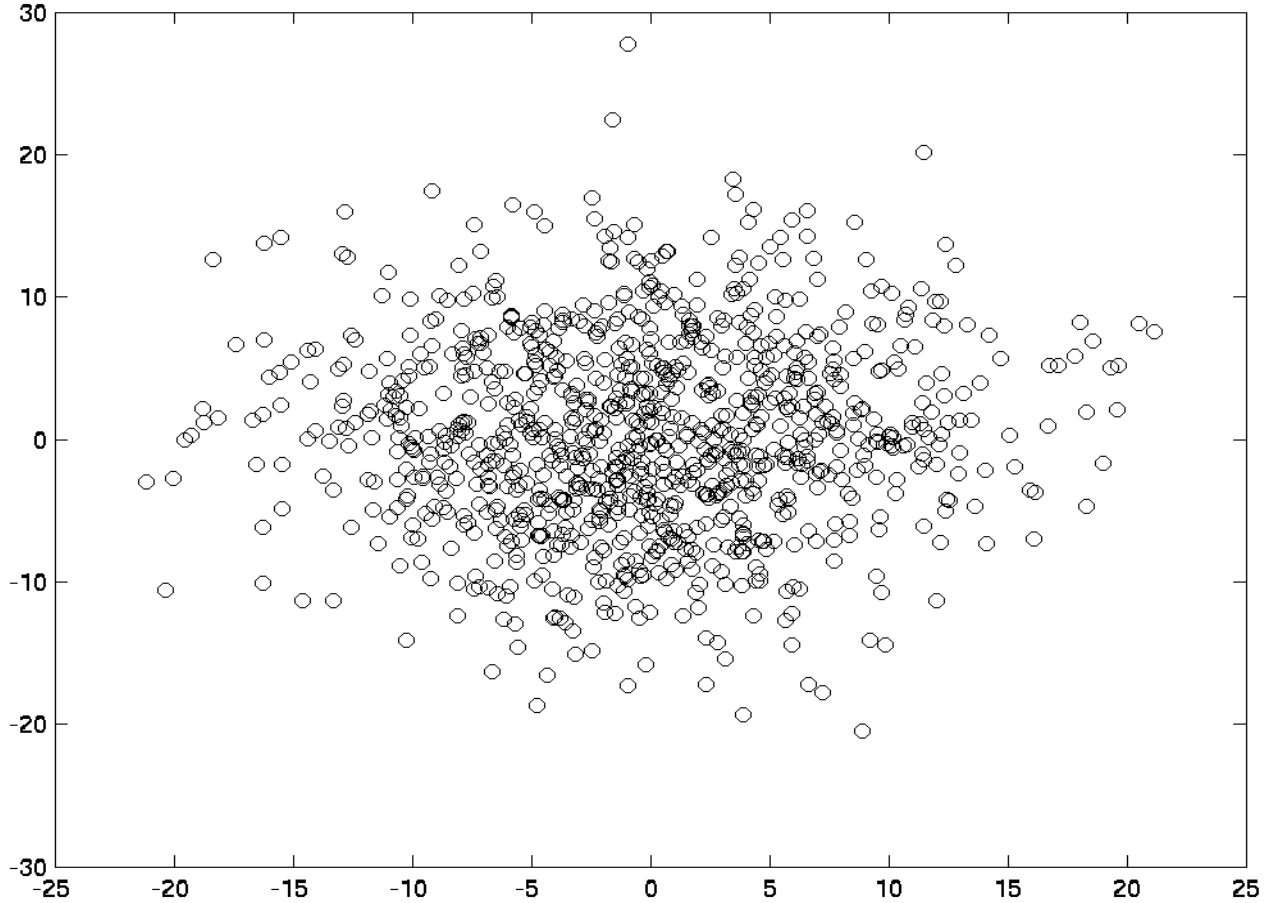
```
end
if show
  plot(x(1,:),x(2,:),'k-');
end
```

Let's test:

`brown(100); draws the following diagram.`



and `brown(100,1000);` draws:



The area with endpoints after n steps has a radius of about $\sqrt{n}$. Command files may not have local functions. There is something called private functions.

The function rand gives a value greater than 0 and less than 1.

Debugging

When your program doesn't work as expected:

- Read the code.
- If you find a mistake correct and test again,
- else add trace prints to se in what order commands are performed and the intermediate values of some variables (remove `;`, add `disp` or `keyboard`).

In the editor window you can set breakpoints and step.

The command `tic` starts a timer and `toc` returnes time since last `tic`.

Lesson 4

Functions as parameters

Functions are descriptions of solutions to problems where some details are variable. Sometimes this variation can be specified as different functions. Therefore we would like to have functions as parameters.

Integration can be seen as a function of a function $f(x)$, integration bounds, a and b, some requirement about steplength or precision, say n— the number of points and an integration method, *method*.

```
% An example of functions as parameters
% Use different methods for integration with
% rectangle, trapezoid och Simpson's method
% Integrate exp(x) from 0 to 4
function integtest
a = 0; b = 4;
exact = exp(b)-exp(a)
```

```
'Rektangle'
area = integrera(@rektangle,@f,a,b,20)
err = exact-area
'Trapezoid'
area = integrera(@trapezoid,@f,a,b,20)
err = exact-area
'Simpson'' method'
area = integrera(@simpson,@f,a,b,20)
err = exact-area
%
function area = integrate(method,f,a,b,n)
area = method(f,a,b,n);
%
function area = rektangle(f,a,b,n)
int = 0;
h = (b-a)/n;
for x =a+h/2:h:b
```

```
    int = int + f(x);
end
area = int*h;
%
function area = trapezoid(f,a,b,n)
h = (b-a)/n;
int = f(a)+f(b);
for j = 1:n-1
    x = a+j*h;
    int = int + 2*f(x);
end
area = int*h/2;
%
function area = simpson(f,a,b,n)
n = ceil(n/2)*2;
h = (b-a)/n;
int = f(a)+f(b);
```

```
for j = 1:2:n-1
    x = a+j*h;
    int = int + 4*f(x);
end
for j = 2:2:n-1
    x = a+j*h;
    int = int + 2*f(x);
end
area = int*h/3;
%
function y = f(x)
y = exp(x);
```

Some of the things not covered are:

- symbolic algebra
- sparse array
- cell array
- structure array

- more on io
- objects
- handle graphics
- graphical user interfaces (GUI)
- content and use of toolboxes

Choose from:

- example of symbolic calculations
- a line breaking program
- a bank accout program
- polynomial approximation with GUI

Symbolic calculation

Let's try to find the quadrature coefficients that would be exact for polynomials of degree 2, Simpson's method.

```
% Quadrature coefficients exact for
% polynimial of degree 2
syms A B C x x0 h y0 y1 y2
```

```
p = sym(A*x^2+B*x+C)
s = solve(subs(p,x,x0)-y0, subs(p,x,x0+h)-y1,...
subs(p,x,x0+2*h)-y2, A, B, C)
q = subs(int(p,x),x,x0+2*h)-subs(int(p,x),x,x0)
simplify(subs(q,{A B C},{s.A s.B s.C}))
%
p =
A*x^2+B*x+C
s =
    A: [1x1 sym]
    B: [1x1 sym]
    C: [1x1 sym]
q =
1/3*A*(x0+2*h)^3+1/2*B*(x0+2*h)^2+C*(x0+2*h)-
1/3*A*x0^3-1/2*B*x0^2-C*x0
ans =
1/3*h*(y0+y2+4*y1)
```

Counting letters

Let's make a program that counts the different letters in a text and print the alphabet sorted after number of occurences. Programs without loops are usually fastest.

```
% count.m
n = 10000;
% Create a random string with n letters
text = char('A'+floor(rand(1, n)*('Z'-'A')));
alpha = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'.';
% With array operators
% Create an 26*n array with alphabet as columns
tic;
alpham = repmat(alpha, 1, n);
% Create an 26*n arrat with text as rows
textm = repmat(text, 26, 1);
% Compare arrays and count matches on each row
charcount = sum(alpham==textm,2);
```

```
% Sort count
[c index] = sort(charcount, 'descend');
% Display alphabet in frequency order
sortalpha = alpha(index)';
t = toc;
fprintf('Time with array operations: %f\n', t);
sortalpha
% With loop
tic;
charcount = zeros(1, 26);
for index = 1:n
    charno = text(index)-64;
    charcount(charno) = charcount(charno)+1;
end
[c index] = sort(charcount, 'descend');
% Display alphabet in frequency order
sortalpha = alpha(index)';
t = toc;
```

```
fprintf('Time with loop operations: %f\n', t);
sortalpha
% Still another version
tic;
charcount = zeros(1, 26);
for textindex = 1:n
  ch = text(textindex);
  for alphaindex = 1:26
    if ch==alpha(alphaindex)
      charcount(alphaindex) =...
      charcount(alphaindex)+1;
      break;
    end % ==
  end % for alphaindex
end % for textindex
[c index] = sort(charcount, 'descend');
% Display alphabet in frequency order
```

```
sortalpha = alpha(index)';
t = toc;
fprintf('Time with double loop operations: %f\n',
t);
sortalpha
```