Introduction to Matlab

Helmut Harbrecht

University of Basel

SS 2024

Overview

Basics

Launch Matlab:

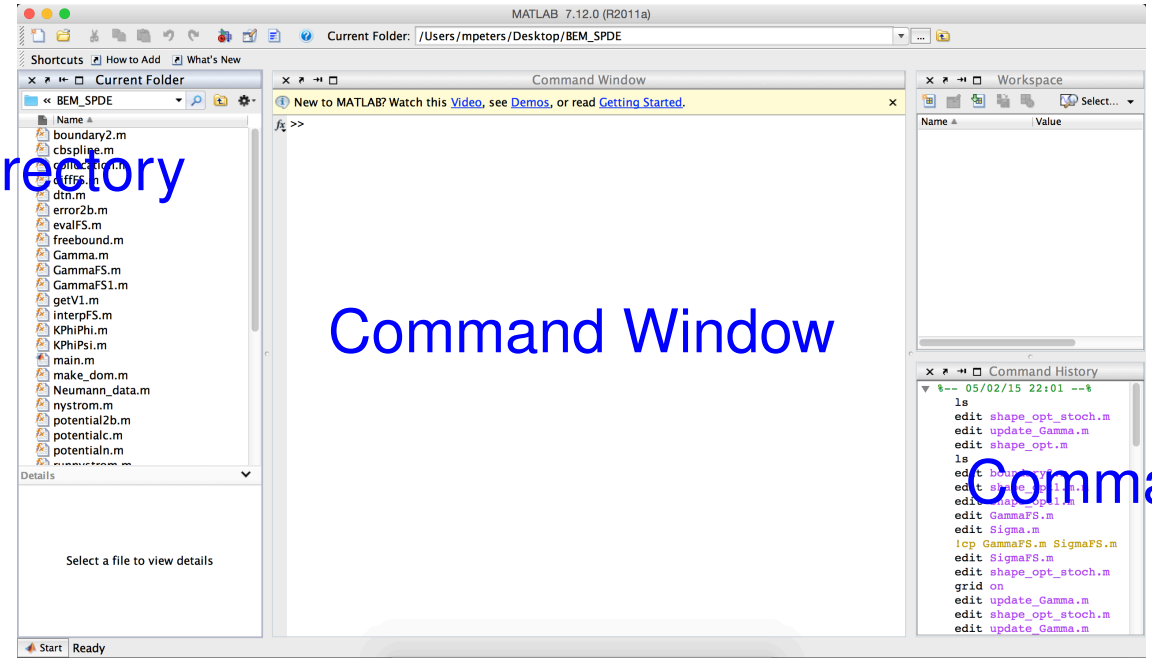
1. Open console;
2. `matlab &` and enter return;
3. Keep the console permanently open.

or with Desktop/Menu-Icon

Exit Matlab:

```
exit
```

Basics



Current Directory

Command Window

Command History

Matlab Features:

1. Command Window for direct execution of Matlab commands
2. Command History list of executed commands. Click to execute again
3. Current Directory search for self-programmed commands here.
4. Editor for writing your own commands

Always terminate commands with Return.

Abort: `ctrl+C`.

Matlab as a calculator

Matrix operations

Creating matrices:

`[1 2 3 ; 4 5 6]` `zeros(2,3)` `eye(5)`

Creation of special row vectors:

`1:3` gives the vector

$$\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}.$$

`1:0.2:2` gives

$$\begin{pmatrix} 1.0 & 1.2 & 1.4 & 1.6 & 1.8 & 2.0 \end{pmatrix}.$$

Assignment (to store values in a variable):

```
c1 = 6      A = [1 2 3 ; 4 5 6]      v = 1:10
```

Matlab as a calculator

Matrix operations

Matrix concatenating:

`[A, zeros(2,2)]` gives

$$\begin{pmatrix} 1 & 2 & 3 & 0 & 0 \\ 4 & 5 & 6 & 0 & 0 \end{pmatrix}$$

and `B = [A, zeros(2,2); eye(2), A]` stores the matrix

$$\begin{pmatrix} 1 & 2 & 3 & 0 & 0 \\ 4 & 5 & 6 & 0 & 0 \\ 1 & 0 & 1 & 2 & 3 \\ 0 & 1 & 4 & 5 & 6 \end{pmatrix}.$$

in the variable *B*.

Matlab as a calculator

Matrix operations

Items selecting:

```
A(1,2) A(1:2,2) A(1:2,1:3) A(:,[1 3])
```

Element-wiset *arithmetic* operations:

```
+   -   .*   ./   .^   .'
```

Element-wise functions:

```
abs   sin   cos   exp   sqrt   min   max ...
```

Element-wise *boolean* operations:

The result is 0 (`false`) or 1 (`true`).

```
 ==   ~=   <   >   <=   >=   &   |   ~
```

Matlab as a calculator

Arithmetical operations of linear algebra

Basic arithmetic operations of linear algebra:

`+` `-` `*` `^`

Further operations:

`A/B`	$A \cdot B^{-1}$
`A\B`	$A^{-1} \cdot B$
`A'`	A^*
`det(A)`	$\det(A)$
`inv(A)`	A^{-1}
`rank(A)`	$\operatorname{rang}(A)$

Matlab as a calculator

Solving linear equation systems

Input:

```
A = [ 1 2 ; 3 4 ]
```

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

```
b = [ 3 ; 5 ]
```

$$b = \begin{pmatrix} 3 \\ 5 \end{pmatrix}$$

```
A\b
```

$$A^{-1}b$$

The output is the solution of $Ax = b$:

```
ans =

    -1

     2
```

Magic squares

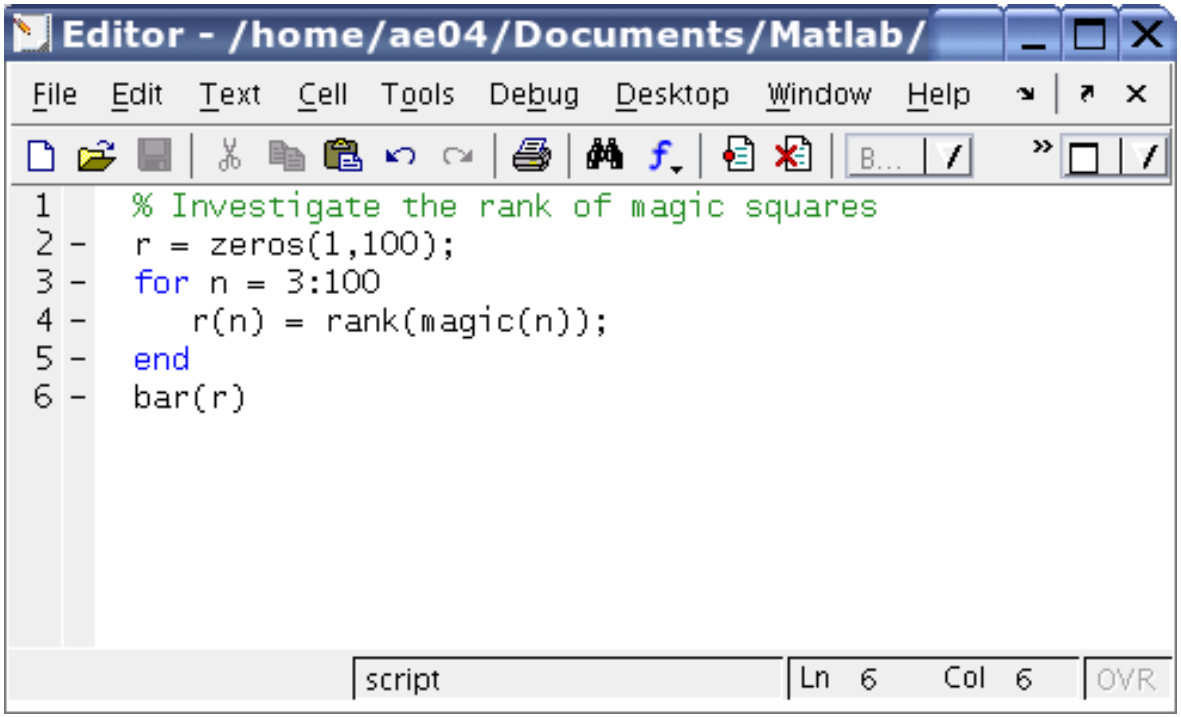
Matrix A $(n \times n)$ is a magic square if the row or column sums are constant, e.g.

$$A = \begin{pmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{pmatrix}$$

`magic(n)` returns $(n \times n)$ magic square.

Scripts

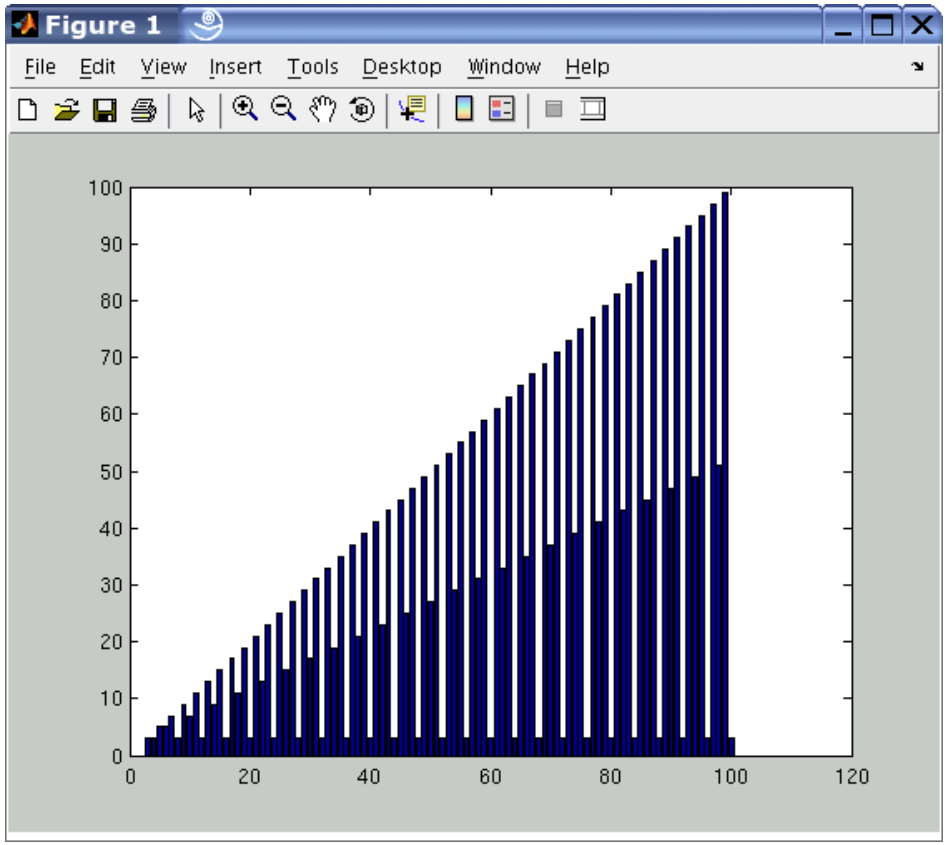
Magic squares



Entering `magicrank` executes `magicrank.m`.
Make sure the directory is active!

Scripts

Magic squares: Output



Scripts

Remarks

- Any file containing a sequence of commands is called script.
- Any file extension must be `.m`.
- Calling the script (without file extension) executes the script sequentially.
- Semicolon blocks the output.
- Comments begin with a percent sign.

Scripts

Restrictions

Restrictions for names of variables and files

- All names must be different.
- Upper and lower case letters are separated.
- Names must begin with a letter.
- Names must not contain any special characters.

Scripts and functions

Restrictionsn

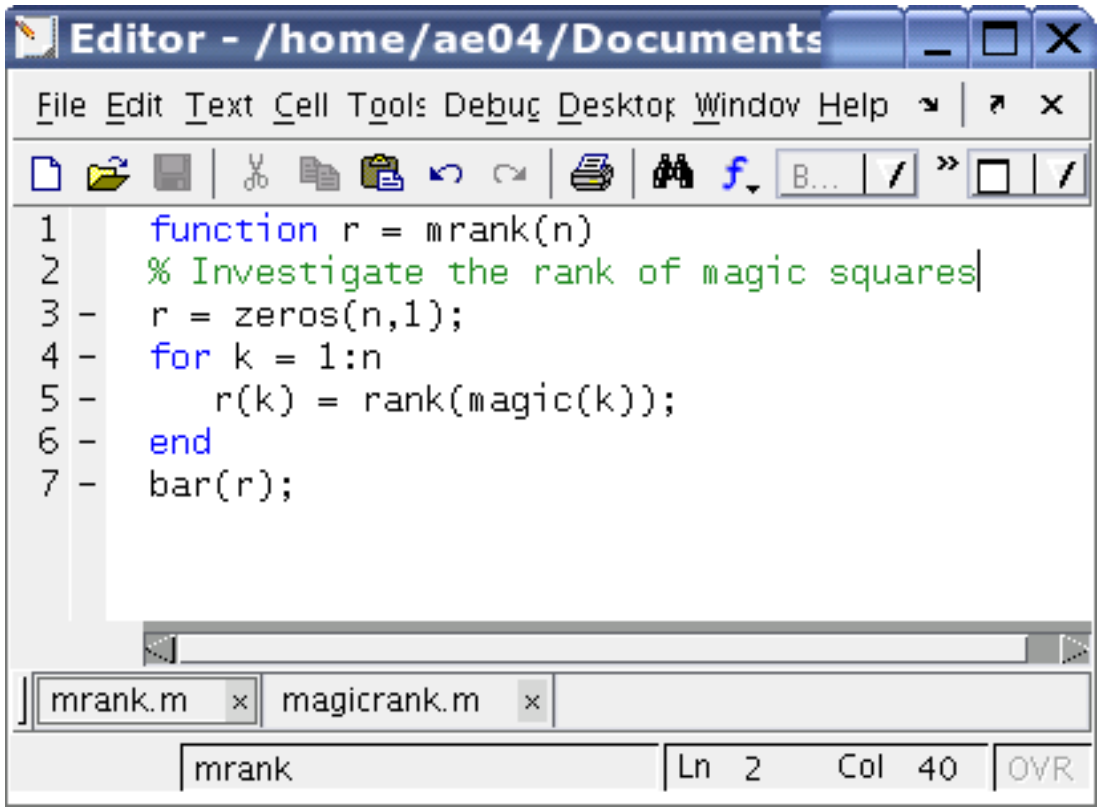
Scripts do not provide:

- Transfer of parameters.
- Return of values and results.

Solution: functions.

Additional distinction: *all* parameters *must* be transferred!

Functions



Entering `mrank(100)` returns the same result.

Functions

Remarks

- File name and function name must match!
- The file extension must be `.m`.
- All variables in a function are local.
- The function `function r = mrank(n)` requires an input parameter and returns a variable as the result.
- Return of several variables:

```
function [r,k] = mrank(n)
```

Call: `[z,m] = mrank(100);`

Functions

Call:

1. `mrank(100)` executes the function and returns the return value.
2. `mrank(100);` executes the function and does not return the return value.
3. `z = mrank(100);` executes the function and saves the return value in z.
4. `z = mrank(100)` executes the function, saves the return value in z and displays it.

The for-loop

```
for k = 4:n
      Commands
end
```

1. Initially, $k = 4$ and all commands between `for` and `end` are executed with the value $k = 4$.
2. Afterwards, $k = 5$ is set and all commands between `for` and `end` are executed with the value $k = 5$, etc.
3. All values from k are run through, up to and including $k = n$.

More precisely: `for k =` *vector*
k runs through the *vector* from start to finish.

The while-loop

```
while t > 0
      Commands
end
```

1. If $t > 0$ applies, all commands between `while` and `end` are executed.
2. It is checked again whether $t > 0$ applies. If this is the case, all commands between `while` and `end` are executed again.
3. This is repeated until $t > 0$ is not failed.

The while-loop

Remarks:

- If $t > 0$ does not apply at the beginning, all commands are skipped.
- You must set a value to t before `while t > 0`.
- If $t > 0$ always applies, the program sequence never aborts.Therefore, the value of t is normally changed within the loop.
- Also permitted e.g. `while (t>0) & (t<1)`.
- More precisely: $t > 0$ returns either 0 (`false`) or 1 (`true`). The loop is run through until the value is 0 (`false`).

Branch: if-elseif-else-end

```
if n == 1
     Commands 1
elseif n == 2
     Commands 2
elseif n == 3
     Commands 3
     ⋮
elseif n == N
     Commands N
else
     CommandsAlt
end
```

elseif and else with the following commands are optional.

Branch: if-elseif-else-end

```
if n == 1
      Commands
end
```

If $n = 1$ applies, *Commands* is executed, otherwise none.

```
if n == 1
      Commands
else
     CommandsAlt
end
```

If $n = 1$, *Commands* is executed, otherwise *CommandsAlt*.

Branch: if-elseif-else-end

```
if n == 1
        Commands
1
elseif n == 2
        Commands
2
elseif n == N
      Commands N
else
     CommandsAlt
end
```

- If $n = 1$, *Commands 1* is executed.
- If $n = 1$ is false and $n = 2$ applies, *Commands 2* is executed.
- If both $n = 1$ and $n = 2$ are false and $n = N$ applies, *Commands N* is executed.
- In all other cases, *CommandsAlt* is executed.

Game of life

Given: Area of size $N \times N$ consisting of 1×1 cells.

- Every cell is in the state *alive* or *dead*.
- At time $T = 0$ there is an initial state, e.g. a living 4×4 block in the middle.
- Generation transfer at time $T = t$ to a new generation at time $T = t + 1$::
  1. Too many or too few living neighbors: cell is dead.
     e.g. 0, 1, 7, 8 neighbors.
  2. Optimal number of neighbors: Cell lives.
     e.g. 3, 4, 5 neighbors.
  3. Otherwise: no change
     e.g. 2, 6 neighbors.

Main program as script

main.m

```
% Main program

parameter;                              % Set parameters
M = init(N);                            % Initialization

for t = 1:Tmax
    display(['Schritt ',num2str(t)]);
    spy(M);                             % Visualization
    pause;
    M = neue_generation(M,tot,gleich,leben);
end

spy(M);                                 % Visualization
```

Parameter setting as script

parameter.m

```
% Parameter

N       = 50;              % Dimension
Tmax    = 100;             % Number of time steps

tot     = [0 1 7 8];       % #Neighbors -> die
gleich  = [2 6];           % #Neighbors -> unchanged
leben   = [3 4 5];         % #Neighbors -> live
```

Initialization as a function

init.m

```
function A = init(N)

% Generate population matrix
% with initial population

A = zeros(N,N);
k = floor(N/2);
A(k-1:k+2,k-1:k+2) = ones(4,4);
```

Calculation of the next generation as a function

```
function A = neue_generation(A,tot,gleich,leben);
% calculate the new generation
B = A;                  % Create a copy of A
[M,N] = size(A);
for j = 1:N
    for k = 1:N
        anz_nachbarn = nachbarzahl(B,j,k);
        if length(find(tot == anz_nachbarn))
            A(j,k) = 0;
        elseif length(find(leben == anz_nachbarn))
            A(j,k) = 1;
        end             % otherwise: nothing changes
    end
end
```

Calculation of the neighboring number as a function

```
function n = nachbarzahl(A,j,k);
% calculate the number of neighbors of cell (j,k)
[M,N] = size(A);
if (j == 1) & (k == 1)
      n = A(1,2) + A(2,2) + A(2,1);
elseif (j == N) & (k == 1)
      n = A(N,2) + A(N-1,1) + A(N-1,2);
elseif (j == 1) & (k == N)
      n = A(2,N) + A(1,N-1) + A(2,N-1);
elseif (j == N) & (k == N)
      n = A(N,N-1) + A(N-1,N) + A(N-1,N-1);
```

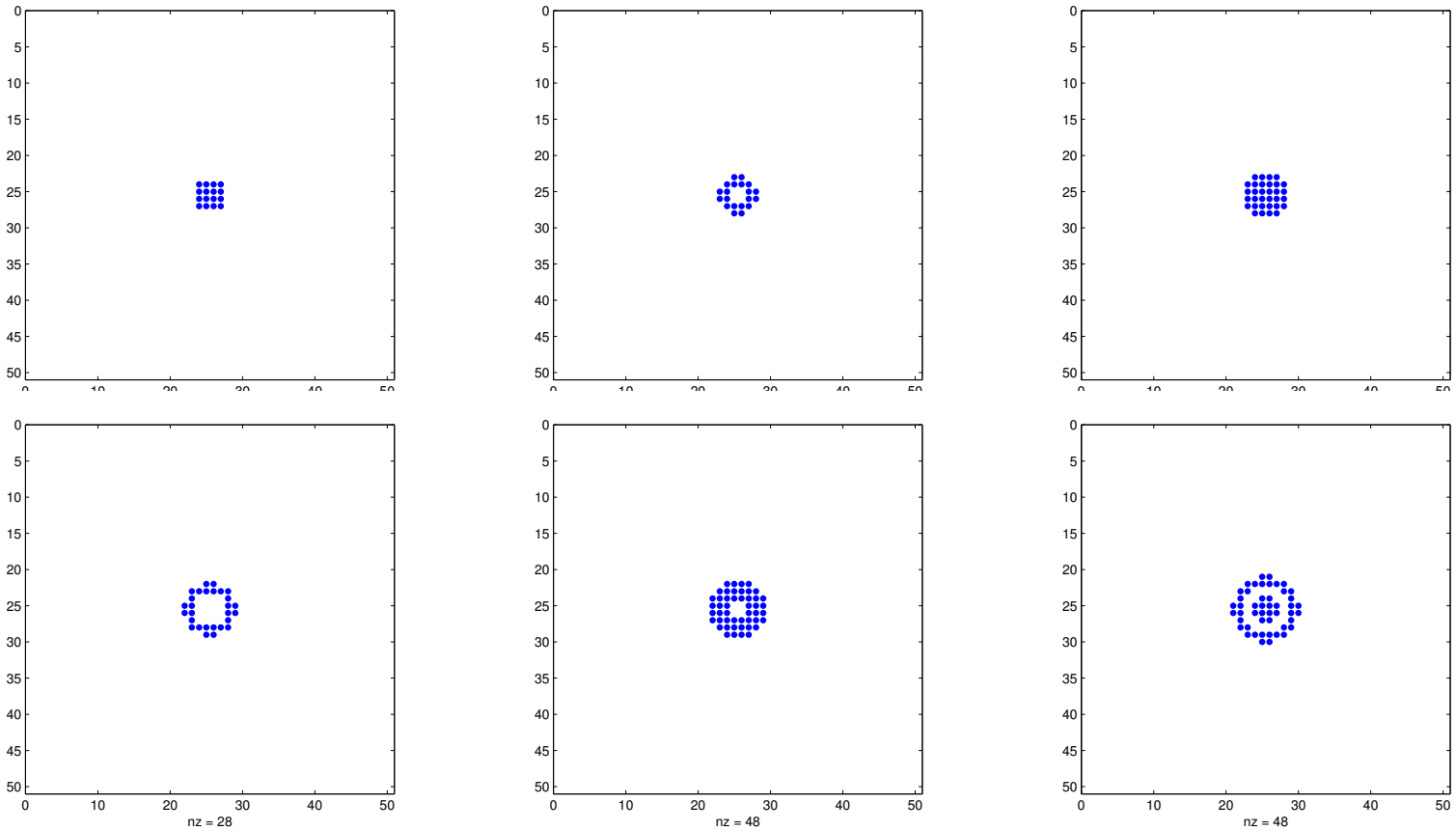
Calculation of the neighboring number as a function

```
elseif j == 1
    n = A(2,k-1) + A(2,k) + A(2,k+1) + ...
        A(1,k-1) + A(1,k+1);
elseif j == N
    n = A(N-1,k-1) + A(N-1,k) + A(N-1,k+1) + ...
        A(N,k-1) + A(N,k+1);
elseif k == 1
    n = A(j-1,2) + A(j,2) + A(j+1,2) + ...
        A(j-1,1) + A(j+1,1);
elseif k == N
    n = A(j-1,N-1) + A(j,N-1) + A(j+1,N-1) + ...
        A(j-1,N) + A(j+1,N);
```

Calculation of the neighboring number as a function

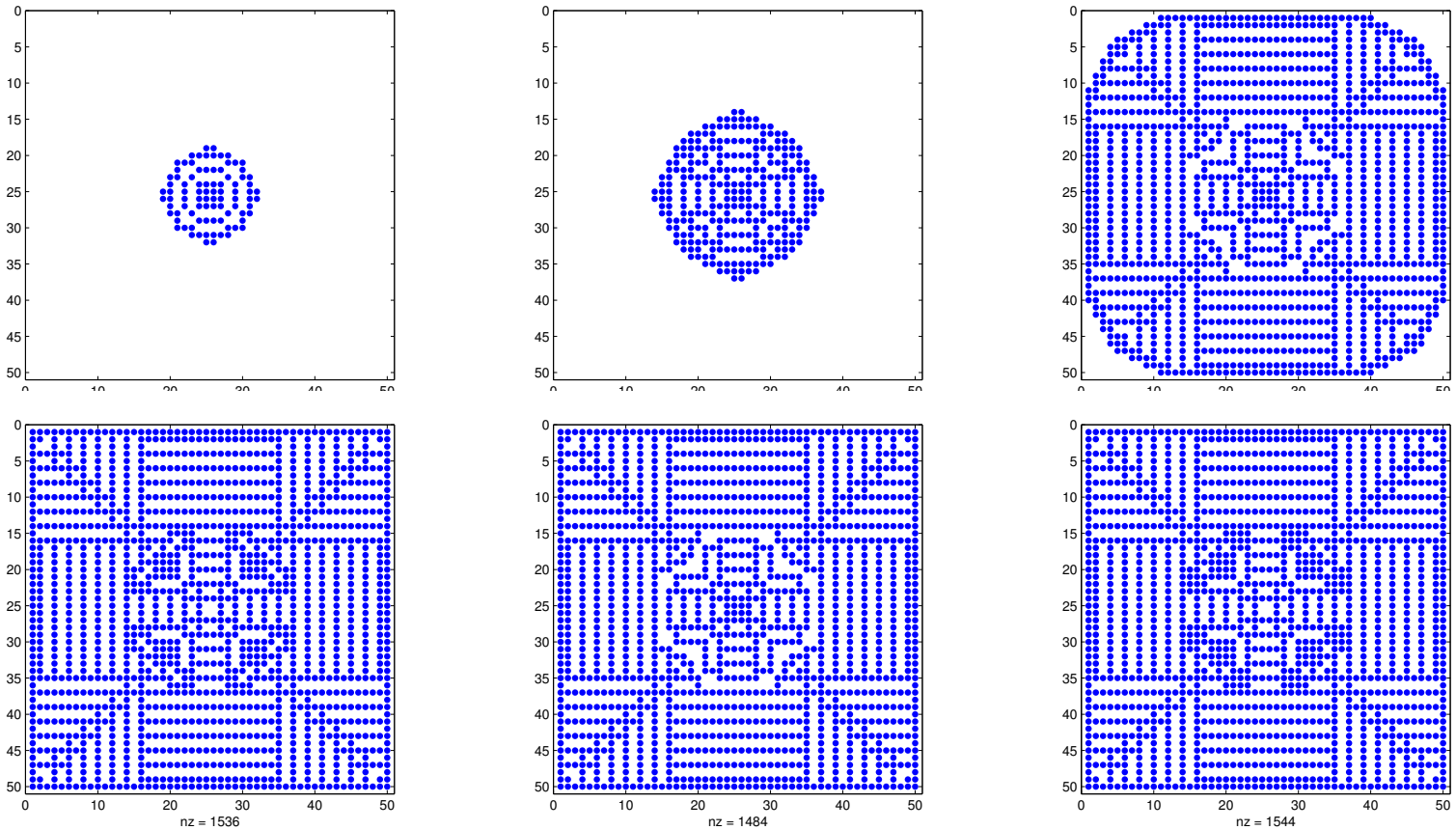
```
else
    n = sum(A(j-1:j+1,k-1)) + ...
        sum(A(j-1:j+1,k+1)) + A(j-1,k) + A(j+1,k);
end
```

Output



The first 6 generations

Output



The generations 10, 20, 60, 75, 76, 77.

Matlab demo for the game of life

The command

```
life
```

opens a game of life app.

Global variables

- Variables that are defined directly in the „Command Window“ or by a command sequence in a script are global.

  These variables can be accessed by the „Command Window“ and by any script.

  The variables and their values exist until they are deleted or changed.

Local variables

- Variables in functions are local.

  They are only recognized inside this function.

  As soon as the function call ends, the variables and their values are deleted.

- Functions do not know global variables, therefore:
  - functions cannot modify global variables,
  - all required values must be passed as parameter,
  - all values still required after the end of the function call must be returned as return values.

Functions as parameters

A method often depends on a function f that varies, e.g.

- Method for determining a zero x^* with $f(x^*) = 0$
- Method for calculating an integral

$$\int_a^b f(x)\,\mathrm{d}x$$

- Special graphical methods for the visualization of f

Problem: Only variables can be passed to functions, not functions themselves!

Functions as parameters

Solution: turn the function into a variable:

If `f` is a Matlab function, `@f` is the „numerical value“ of the function. It is called handle of the function `f`

This handle can be stored in a variable and/or passed as a value to another function.

Functions as parameters

```
function y=g(x)
y = 1./(x.^2);
```

```
function S = p_summe(f,n,N)
S=0;
for i = n:N
     S = S + f(i);
end
```

The call `p_sum(@g,1,100)` calculates

$$\sum_{i=1}^{100} \frac{1}{i^2}.$$

Functions as parameters

- Handles can be stored in variables:

  `s = @sin` oder `handle_g = @g`

- Handles can also be defined and saved directly:

  ```
  s = @(x) 2*x+1;
  ```

  saves the function $x \mapsto 2x + 1$ in the variable `s`. The call `s(3)` results in 7.

Memory management

- If you want to access a matrix entry, for example by `A(2,3:6)`, the matrix must have at least size 2×6 and be defined accordingly **previously**, for example as

  `A = zeros(2,10).`

  Remark. If you access `A(2,3:6)` without defining the matrix beforehand or if it is too small, it is automatically created as a zero matrix of the required size or resized accordingly.

  Nevertheless, you should define the matrix yourself beforehand because:
  - the size adjustment costs a lot of running time
  - defining the matrix sizes at the beginning is good programming style
- The memory of variables that are no longer required clears by

  `clear` or `clear`+Variablename

Vectorization

There are (at least) two ways to calculate the vector:

$$v = \begin{pmatrix} 1 & -2 & 3 & -4 & \cdots & \cdots & 999 & -1000 \end{pmatrix}.$$

```
v = 1:1000;
for i = 1:500
     v(2*i) = -v(2*i);
end
```

```
v = 1:1000;
v(2:2:1000) = -v(2:2:1000);
```

The second option uses vector calculation instead of loops. That's much faster!

As a rule, the use of `for` loops should be avoided

Visualization of vector data

1. `bar`: bar chart
2. `pie`: pie chart
3. `plot`: polygon curve
4. `area`: curve or surface colored in below

Visualization of matrix data

1. `spy`: graphic of non-zero elements
2. `contour`: isolines of matrix
3. `pcolor`: pseudocolor plot using the values of matrix

   Achtung: With `pcolor` the last row and column are not displayed!

Functions plot

Matlab does not plot functions, but tables of values as curves!

1. Define column vector `x` of x-values, e.g.
   ```
   x = (0 : pi/100 : pi)';
   ```
2. Define column vector `y` of y-values of a function, e.g.
   ```
   y = sin(x);
   ```
3. Plot value table
   ```
   plot(x,y);
   ```
4. Alternative:
   ```
   plot(x,sin(x));
   ```

Remark: The function to be plotted must be applicable elementwise to `x`!

Functions plot

To plot multiple functions, the *y* values column by column must be placed in a matrix `y`:

1. Define column vector `x` of *x*-values, e.g.
   ```
   x = (0 : pi/100 : pi)';
   ```
2. Define column vector `y` of *y*-values of functions, e.g.
   ```
   y = [sin(x),cos(x)];
   ```
3. Plot value table
   ```
   plot(x,y);
   ```
4. Alternative:
   ```
   plot(x,[sin(x),cos(x)]);
   ```

Plot parameterized curves

A (continuous) function $[a, b] \longrightarrow \mathbb{R}^2$, $t \mapsto (u(t), v(t))$ is called plane curve in parameter form and

$$K = \{(x, y) \in \mathbb{R}^2 \mid x = u(t),\ y = v(t),\ t \in [a, b]\}$$

is the graph of the curve. For example

$$K = \{(x, y) \in \mathbb{R}^2 \mid x = \sin t,\ y = \cos t,\ t \in [0, 2\pi]\}$$

a circle with radius 1.

Plot parameterized curves

Plot K:

1. Define column vector `t` of t values, e.g.
   ```
   t = 2*(0 : pi/100 : pi)';
   ```
2. Define column vector `x` of x-values, e.g.
   ```
   x = sin(t);
   ```
3. Define column vector `y` of y values, e.g.
   ```
   y = cos(t);
   ```
4. Plot value table
   ```
   plot(x,y);
   ```
5. Alternative:
   ```
   plot(sin(t),cos(t));
   ```

Mathematical functions definition

To plot a function f, you have to apply the Matlab implementation `f` to the vector `x` of the x-values to get the y-values!

For curves in parameter form, u and v must be implemented as above.

So `f` (or `u` and `v`) must be implemented in such a way that `x` (or `t`) can also be used!

Mathematical functions definition

Example: The function

$$f(x) = \frac{1}{1 + 5x^2}, \qquad x \in \mathbb{R}$$

is to be implemented as the Matlab function `f`:

`f.m`

Mathematical functions definition

Example: The function

$$f(x) = \frac{1}{1 + 5x^2}, \qquad x \in \mathbb{R}$$

is to be implemented as the Matlab function `f`:

`f.m`

```
function y = f(x)

eins = ones(size(x));
y = eins ./ (eins + 5 * x.^2);
```

Mathematical functions definition

Example: The function

$$f(x) = \frac{1}{1 + 5x^2}, \qquad x \in \mathbb{R}$$

is to be implemented as the Matlab function `f`:

`f.m`

Alternative:

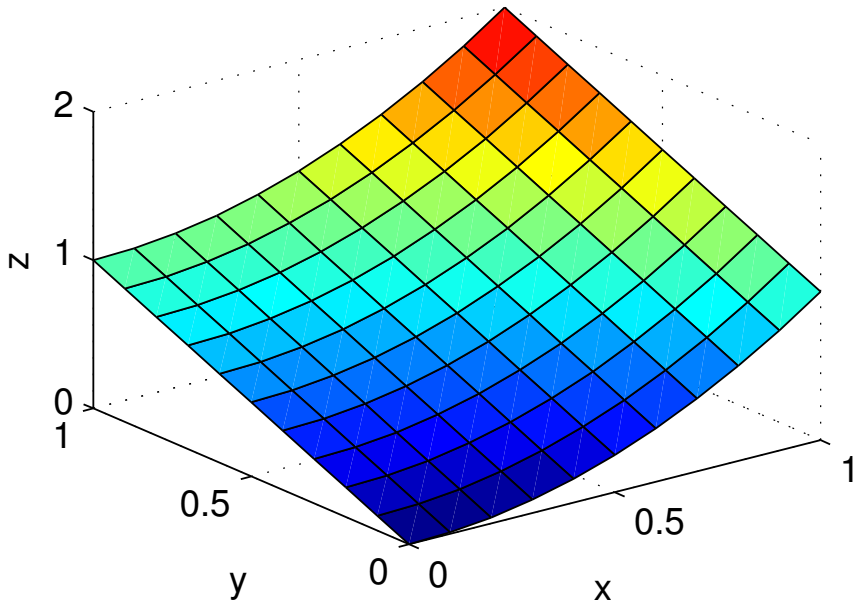
```
function y = f(x)
y = 1 ./ (1 + 5 * x.^2);
```

Plot functions of two variables

Basics

Example: Plot

$$z = f(x, y) = x^2 + y, \qquad x \in [0, 1], \quad y \in [0, 1].$$



Plot functions of two variables

Basics

Matrices `X` and `Y` are required so that the element-by-element evaluation of `X` and `Y` provides a matrix `Z` with the function values, e.g.

$$X = \begin{pmatrix} 0 & 0.5 & 1 \\ 0 & 0.5 & 1 \\ 0 & 0.5 & 1 \end{pmatrix} \qquad Y = \begin{pmatrix} 0 & 0 & 0 \\ 0.5 & 0.5 & 0.5 \\ 1 & 1 & 1 \end{pmatrix}$$

results for $z = f(x, y) = x^2 + y$

$$Z = \begin{pmatrix} 0 & 0.25 & 1 \\ 0.5 & 0.75 & 1.5 \\ 1 & 1.25 & 2 \end{pmatrix}$$

Plot functions of two variables

Procedure

1. Create matrices X and Y, e.g.

   ```
   [X,Y] = meshgrid(0:0.1:1, 0:0.1:1);
   ```

   More precisely: if `x` and `y` vectors of the x and y values at which the function f is to be evaluated, the required matrices `X` and `Y` are generated by:

   ```
   [X,Y] = meshgrid(x,y);
   ```

2. Create matrices Z of the function values, e.g.

   ```
   Z = X.^2+Y;
   ```

3. Plot function

   ```
   surf(X,Y,Z);
   ```

Plot options and modification of graphs

1. Modification of the shade:

   ```
   shading faceted
   shading flat
   shading interp
   ```

2. Modification of the axes:

   ```
   axis([xmin xmax ymin ymax])
   axis equal
   ```
   und viele mehr

3. Labeling:

   ```
   title('Überschrift')
   xlabel('x-Achse')
   ```

4. Further modifications possible directly on the graph!

Plot options and modification of graphs

Further possibilities:

1. Surface without grid:
   ```
   surf(X,Y,Z, 'EdgeColor', 'none');
   ```
2. „illuminated “ surface:
   ```
   surfl(X,Y,Z);
   ```
3. Grid only:
   ```
   mesh(X,Y,Z);
   ```
4. Vertical lines:
   ```
   contour(X,Y,Z);
   ```
5. Color map:
   ```
   pcolor(X,Y,Z);
   ```

Plot options and modification of graphs

Further possibilities:

1. Plot (parameterized) curves in 3d-space

   ```
   plot3(X,Y,Z);
   ```

2. Plot 2D vector fields, i.e.

   of functions $D \subseteq \mathbb{R}^2 \longrightarrow \mathbb{R}^2$ with $(x, y) \mapsto (u, v)$

   ```
   quiver(x,y,u,v)
   ```

3. Plot 3D vector fields, i.e.

   of functions $D \subseteq \mathbb{R}^3 \longrightarrow \mathbb{R}^3$ with $(x, y, z) \mapsto (u, v, w)$

   ```
   quiver3(x,y,z,u,v,w)
   ```

Mathematical operators and functions

+	-	.*	./	.^	*	/	\	^
:	[]	,	;	()				
==	~=	<	>	<=	>=	&	|	~

Mathematical operators and functions

```
zeros   ones    eye     size    length
det     inv     rank    trace
sum     find

exp     log     sqrt    power
abs     floor   ceil    min     max

sin     cos     tan     cot     sec     csc
asin    acos    atan    acot    asec    acsc
sinh    cosh    tanh    coth    sech    csch
asinh   acosh   atanh   acoth   asech   acsch
```

Featured commands

```
%

function

if   elseif   else  end

for  end

while  end

pause    ...

display  ' '

bar      spy

magic    life
```

Matlab use

Due to the large number of commands and even more options, it is impossible to memorize the full functionality and syntax. That is why there is help:

1. `help` + shows the help
2. matlab help menu: Extensive search options
3. character sequence + tabulator: automatic command extension
4. up arrow: command history

Help and demo programs

- Extensive Matlab help files can be called up with F1. These include:
  - information on the individual commands;
  - Demo programs.
- Literature about Matlab:
  - Good introduction in German:
    `www.hs-ulm.de/users/gramlich/EinfMATLAB.pdf`
  - Good overview: Matlab primer
    `www.math.toronto.edu/mpugh/primer.pdf`
    is a free, old edition. New edition at `books.google.com`