Basic Introduction to MATLAB
Exercises

Part 1.....Elementary matrix operations

i) The most important MATLAB-command is `help`.
Try `help`, `help` <*directory*> and `help` <*function*>.

ii) Create an arbitrary matrix `a` of size 3×3. The central element of `a` is `a(2,2)`. The third row is `a(3,1:3)` or in shorter notation `a(3,:)`. Try these types of indexing. Change the values of specific elements, e.g., `a(3,1)=9`. Also try changing the column order using `a=a(:,[2 3 1])`. Can you explain, how these matrix manipulations work?

iii) Separate your matrix `a` into three column vectors `a1, a2, a3` by suitable indexing. Also try a separation into three row vectors `b1, b2, b3`. Combine your row or column vectors to recreate the original matrix `a`.

iv) MATLAB uses the values `Inf` and `NaN` to deal with infinity and indefiniteness. Create variables of such values by computing $1/0$ and $0/0$, respectively. Assign these values to any of the elements of `a`.

v) By default, MATLAB uses the variables `i` or `j` for the imaginary unit of a complex number. Create a complex vector `x=[6+4i 3+17i 4]`.
Watch out: You can assign arbitrary values to the variables `i` and `j`, e.g. by writing `i=3`. However, this might cause some confusion. Therefore, avoid the variable names `i` and `j` in your computations!

vi) Choose one of the commands `magic, randn, zeros, ones, eye`

- to create a matrix `d1` of size 4×4 containing zero-elements only,
- create a matrix `d2` of size 7×3 with all elements being 1,
- create an identity matrix `d3` of size 5×5,
- create a matrix `d4` of size 2×3 with random numbers,
- create a matrix `d5` of size 5×5, where the sum along rows or columns is equal for all rows and columns of the matrix. Test, if this condition really

holds, using `sum(d5)`, `sum(d5')` or `sum(d5,2)`. Can you explain, how these commands act? What is the sum of all elements of `d5`?

vii) List all variables using the command `whos`. Check the size of the individual variables using `size`. What is the difference between the commands `size` and `length`?

viii) Variables can be deleted using `clear <`*name*`>`. Try `clear x` and check the effect in the workspace (watch the workspace window or type `whos` again). Clear the whole workspace using `clear all`.

ix) Create the following matrices `a`, `b` and `c`:
`a = magic(3)`,
`b = ones(3,1)*(2:4)`
`c = [cos(x) sin(x) 0; -sin(x) cos(x) 0; 0 0 1]`, with `x=pi/6`.
Try the operations `a.*b, b./a, a./c, a.^2`. What happens, if you leave out the period in front of the operators, i.e. `a*b, b/a, a/c, a^2`?

x) Choose one of the commands `fliplr, inv, diag, trace, det, mean, std, max, min, flipud` to

- compute the inverse of `c`,
- exchange the left and right column of `c`,
- compute the trace and the determinant of `c`,
- compute the mean and the standard deviation of `c` and of its inverse,
- exchange the first and the last row of `a`,
- determine the maximum and minimum values of `c`,
- determine the diagonal of `a` and compute the sum of its elements. Compare the result with the trace of `a`.

xi) $\mathbf{ax} = \mathbf{y}$ represents a system of linear equations, where $\mathbf{a}$ is defined as above and $\mathbf{y} = [28\ 34\ 28]'$. Solve the system by typing $\mathbf{x} = \mathbf{a}\backslash\mathbf{y}$ and proof the result. Also try $\mathbf{x} = \mathbf{inv}(\mathbf{a}) * \mathbf{y}$.

xii) The command `randn` creates a matrix of normally distributed elements with mean value $\mu = 0$ and standard deviation $\sigma = 1$.
Create the following matrix `a` of normally distributed column vectors:
`a = [randn(100,1) randn(100,1)*3 randn(100,1)+4]`. Compute the mean and standard deviation along the columns of `a`. Does the result meet your expectations?

xiii) Check the following properties:

$$\begin{aligned} A+B &= B+A & \qquad A+(B+C) &= (A+B)+C \\ (A+B)' &= A'+B' & \qquad \mathrm{tr}(A+B) &= \mathrm{tr}A+\mathrm{tr}B \\ A(BC) &= (AB)C & \qquad A(B+C) &= AB+AC \\ (AB)' &= B'A' & \qquad \det(AB)=\det(BA) &= \det A \det B \\ \mathrm{tr}(AB) &= \mathrm{tr}(BA) & \qquad \mathrm{rk}(AB) &\le \min(\mathrm{rk}A, \mathrm{rk}B) \\ A^{-1}A &= I & \qquad (AC)^{-1} &= C^{-1}A^{-1} \\ (A')^{-1} &= (A^{-1})' & \qquad \det A^{-1} &= 1/\det A \\ C'C = CC' &= I & \qquad C^{-1} &= C' \end{aligned}$$

Remarks:

- The last two properties hold only for orthonormal matrices.
- tr is the trace of a matrix, rk its rank.
- The inverse A^{-1} can be computed in four different ways: `inv(a)`, `a^(-1)`, `a\eye(3)`, or `eye(3)/a`.

xiv) Multiplication of a matrix with a diagonal matrix is inefficient due to the large amount of zero-products. Proof this by typing the following lines:

```
>> d = 1:200;
>> a = randn(200,200);
>> tic, b1 = a * diag(d); toc
>> tic, b2 = a .* d(ones(200,1),:); toc
```

Which method is faster? Can you explain, what happens analytically? Check the matrices for equality without comparing each element separately, e.g., using `max(max(abs(b1-b2)))`.

xv) Define a matrix `a = magic(10)`. Use the command `find` to get the index of those elements $a_{ij} > 75$. Also determine the number of those elements. Set them to 0. Also try

```
>> m = a<75;
>> b=a.*m;
```

Part 2.....Graphics

i) The following function is a representation of a satellite orbit:

$$\mathbf{F}(t) = 5\sin t * \mathbf{e_1} + 5\cos t * \mathbf{e_2} + 4\sin 4t * \mathbf{e_3}$$

Visualize the following elements in 4 subplots using the commands in brackets. Always use `t=0:0.01:2*pi`.

- $4\sin 4t$ `<plot>`
- $5\sin t, 5\cos t$ `<plot,hold>`
- $5\sin t * \mathbf{e_1} + 5\cos t * \mathbf{e_2}$ `<plot>`
- $5\sin t * \mathbf{e_1} + 5\cos t * \mathbf{e_2} + 4\sin 4t * \mathbf{e_3}$ `<plot3>`

Add title, axes labels, legends etc. to the subplots. Add the following lines to the fourth subplot:

```
>> hold on;
>> sphere(20);
>> colormap(white);
>> axis equal;
```

Are you aware, how the command `sphere` acts?
Try the rotation-button in the figure window to change the perspective interactively. Also try out the command `view`.

ii) The difference between two graphs may appear quite different, depending on the scale of the axis. To see the effect, try different logarithmic scalings.
Define a vector `t=(1:1000)` and generate both `a=t` and `b=t.^2`. Then display `a` and `b` as functions of `t`. Create 4 subplots, each with one of the commands `plot, semilogy, semilogx, loglog`. In all cases use `hold on` to display both curves in the same axes.

iii) Display one of the above curves in a separate figure window. This time assign a graphic handle to the curves, e.g. `h = semilogy(t,a)`. Try the commands `get` and `set` to get a list of properties and change their values, e.g., try to change the width, color and style (dashed, dotted, etc.) of the lines.
Try the same interactively, using the options in the figure window.

iv) Try the following graphics:

```
>> a=0:0.01:2*pi;
>> polar(a,cos(a));
>> polar(a,cos(3*a).^2);
>> polar(sin(a),cos(a));
```

v) The file `geoidbsp.mat` contains the variable `Nges`, which is a matrix of geoid height values for Bavaria. Load the file using `load geoidbsp` – also check, if `Nges` really exists (`whos`). Again use 4 subplots to visualize the geoid surface:

- Use an elementary 3-dimensional surface representation like `mesh(Nges)` or `surf(Nges)`.
- Repeat the same representation, but add a coordinate matrix to the graph (geo-referencing). Use the following coordinate-vectors:
  ```
  >> x = (9+5/60):(10/60):(14-5/60)
  >> y = (47+3/60):(6/60):(52-3/60)
  ```
- Use `contour` to create a contour line representation. Plot contour lines in an 0.5 m interval, but label only the integer valued lines.
- Create a `pcolor`-representation and overlay black contour lines.
  Get familiar with Bavaria by adding the following cities: Hof (11.92, 50.33), München (11.58, 48.17), Nürnberg (11.08, 49.45) and Passau (13.47, 48.57)
  Example:
  ```
  >> plot(13.47,48.57,'ok');
  >> gtext('Passau');
  ```

Again, add axis labels, titles, colorbars, etc. to the subplots.

vi) The following code lines generate a vector field in 2 dimensions. Check the syntax for errors, correct them and plot the vector field.

```
>> [x,y,p]=peaks;
>> [dx,dy]=gradeint(p);
>> contour(x,y,p,20,'r');
>> grid; axis square; hold off;
>> quiver(x,y,dy,dy);
```

vii) Let `x=-100:100` and `y=-100:100` define a rectangular area. Compute and display the distance of each point of the area to the origin. Try

```
>> [X,Y]=meshgrid(x,y);
>> Z = sqrt(X.^2 + Y.^2);
>> surf(X,Y,Z);
```

viii) Let the matrix `x` in exercise 1.*xii* represent a time series of 3D-coordinates x, y, z. The first column contains $x(t)$, the second $y(t)$ and third $z(t)$. Accordingly each row contains point coordinates for a certain epoch. Compute the time series of distances from the origin and plot the result. Type `help sum` to see, how you can apply the function `sum` to solve this problem.

Part 3 ..... MATLAB as programming language

i) Write a MATLAB script file `gauss.m`, which computes and plots the normal distribution function

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

Run the script with different values for the parameters σ and μ (for comparison display the results in the same axes using `hold on`).

ii) Change the script into a function file. The first line should read:
`function f = gauss(x,mu,sigma)`,
where the input argument `x` is a vector, e.g., `x = -10:0.2:10`, while `mu` and `sigma` are scalars. The output argument `f` should have the same dimension as `x`. *Programming style:* do not include graphic commands in your function file – only computations!

iii) Add a suitable help text to your function. This should describe at least all input and output arguments and the function call. Make your function monkey-proof by checking the input arguments for proper dimensions, etc. Also add reasonable default values, e.g., in case `x` is the only input argument (use `nargin`. Proper default values are `mu=0` and `sigma=1`.

iv) A famous equation in orbit mechanics is Kepler's problem: $M = E - e \sin E$. There M and E are mean and eccentric anomalies, respectively, and e is the eccentricity ($0 \leq e < 1$). Both M and E are in radians. To solve the equation for E, one uses the following iteration:

$$E_{i+1} = M + e \sin E_i \quad \text{with initial value } E_0 = M$$

Write a function file to compute E from M and e. The first line may read
`function [E,iter] = kepler(M,e)`.
The iteration shall last, while $(E_{i+1} - E_i)$ is larger than 10^{-6}. The output argument `iter` describes the number of iteration steps performed.

v) Write a function file, that computes rotation matrices. The first line should read:
`function r = rotmtx(alpha,i)`,
where `alpha` is the rotation angle (units?) and `i` the rotation axis (possible values are 1, 2 or 3). Both input arguments should be scalars. Output argument should be a rotation matrix $R_i(\alpha)$ of size 3×3. Proof your function by testing the relation $R_i(\alpha)' = R_i(\alpha)^{-1} = R_i(-\alpha)$.

vi) The scalar product of two column vectors can be computed in different ways:

$$\mathbf{a} \cdot \mathbf{b} = \mathbf{a^T b} = \sum_i a_i b_i$$

Create two arbitrary column vectors `a` and `b` of equal length and try `dot(a,b)`, `a'*b`, `sum(a.*b)` as well as a `for`-loop. Compare the results and the necessary

computation times (you might have to use long vectors in order to see a reasonable effect).

vii) Write two function files `dms2deg` and `deg2dms` to transform an angle from a degree-minutes-seconds representation to float degrees and vice versa, e.g., from $5°30'00''$ to $5.5°$. Use the command `fix`.

Remarks

- Check `http://www.mathworks.com` for more information on the MATLAB software package.
- Use the *primer* as reference documentation. Also try the examples there. It also contains a detailed list of MATLAB commands and in-built functions (of MATLAB version 4).
- Also check the MATLAB-demo routines (type `demo` or `expo`) and MATLAB's own introduction (`intro`).
- The most important command is `help`! Use it extensively!

Commands .....Part 1

Commands .....Part 2

Commands .....Part 3