Aalto-yliopisto

Credit Exercises

Updated 9.11.2018

If you wish to obtain the credits for the course, choose 4 of the problems below and prepare a published .pdf-file. Send the pdf-file and the source code by email to Heikki and Juha (heikki.apiola 'at' aalto.fi,juha.kuortti 'at'aalto.fi).
Please: Use **semicolons(;)** to avoid long outputs!
Deadline for submission is **Wed 14.11.18** (at 23.59.59)

Exercise 1: Write functions F2C and C2F that convert Fahrenheit scale to Celsius, and Celsius to Fahrenheit. Be sure to have them work for vectors as well as scalars.

$$T_C = \frac{5}{9}(T_F - 32).$$

(F: Fahrenheit, C: Celsius)
Test your functions, especially: $(-40, -40)$ and $(0°C, 32°F)$.
Make a table of values and plot (for instance for $F = -40, -39, \ldots, 99, 100$). Make grid, title, and axis labels for your plot.

Exercise 2: (a) Evaluate $y = x^2$ for `x=-4:0.1:4`.

(b) Add random noise to these samples. Use `randn`. Plot the noisy signal for instance with markers `('.:')`.

(c) Fit a 2^{nd} degree polynomial to the noisy data.

(d) Plot the fitted polynomial on the same plot, using same x-values and a red line.

Hint: The function `polyfit` returns coefficients of fitted polynomial.
To evaluate, use `polyval`.

Exercise 3: The height $h(t)$ and horizontal distance $x(t)$ traveled by a ball thrown at an angle α with a speed v are given by

$$\begin{cases} h(t) = (v \sin \alpha)\, t - \frac{1}{2} g\, t^2 \\ x(t) = (v \cos \alpha)\, t \end{cases}$$

$g = 9.81 m/s^2$.

(a) Suppose the ball is thrown with a velocity $v = 10m/s$ at an angle of $35°$. Use MATLAB to compute how high the ball will go, how far it will go, and how long it will take to hit the ground.

Suggestions, help:

- For your convenience you can use degrees with functions `cosd, sind`.
- Start by plotting h vs. t (as suggested on part (b)). For that you need a sufficienlly large T_{max} (just experiment). Then form a vector `[0,...,Tmax]` with sufficiently many points to make the plot. (Adjust `Tmax`).
- To find the smallest `Tmax`, use the function `find` in the form: `find(<condition on h>,1)`, where 1 means "first" (`help find`).
- For max height you can use `find`, or even one step shorter: using `max` in the form with two output arguments. See links of Session 3 on web-page, "minmaxexa.m"

(b) Use the values of v and α and T_{max} to plot the ball's *trajectory*, that is the parametric curve $(x(t), h(t)), t = 0, \ldots, T_{max}$.

(c) Find T_{max} for $\alpha = 45$ repeating what you did in (a). Plot the trajectories for $v = 10m/s$ and the values of $\alpha = 20, 30, 45, 60, 70$ in the same picture. Use the above T_{max} for $\alpha = 45$.

(d) (This is "voluntary", no difficulties, but the extra work not necessary this time.) Plot the trajectories for $\alpha = 45$ and values of $v = 10, 12, 14, 16, 18$.

Exercise 4: This assignment focuses on the connection of matrices, images and singular values.
The *singular value decomposition* of a matrix is

$$\mathbf{A} = \mathbf{USV}^T,$$

where matrix S is a diagonal, and U and V are orthogonal square matrices; if this all means nothing to you, don't worry – this assignment requires essentially no mathematical framework.
The information contained in matrix be compressed (in a sense) by dropping off parts of the singular value decomposition, and conveniently the size of singular value tells the importance of the associated singular vectors; even more conveniently they are already sorted inside the S; or in MATLABese, `U(:,1:k)*S(1:k,1:k)*V(:,1:k)'` is the best possible approximation of A using $2k$ vectors.
Any image can be thought of as an $m \times n$ matrix, where element i, j describes the color value of the pixel at the location. We will now see, how singular values can be used to compress images.

Read in any image you wish using MATLAB's `imread`-command. It will (usually, but little depending on the image format). $m \times n \times 3$ array. Each of the three layers of the array corresponds to color intensities of red, green and blue respectively. First, convert the image to grayscale with command `rgb2gray`, or pick one of the layers: `im = A(:,:,1)`. After this, do the singular value decomposition with command `[u,s,v] = svd(im)`.
Now you can test with which k the commands

```
>> M = u(:,1:k)*s(1:k,1:k)*v(:,1:k)';
>> image(M)
```

produces results that are distinguishable. The large components of the image should also start appearing first, which makes singular value decomposition a useful tool in pattern recognition.
In order to do this properly for colored images, you should do the above process separately for each of the three layers of the array; also, for fun, you could try different values of k for each layer, suppressing certain colors more than others.

Exercise 5: Here are 25 observations y_k, taken at equally spaced values of t.

```
t = 1:25
y =[5.0291    6.5099    5.3666    4.1272    4.2948
    6.1261   12.5140   10.0502    9.1614    7.5677
    7.2920   10.0357   11.0708   13.4045   12.8415
   11.9666   11.0765   11.7774   14.5701   17.0440
   17.0398   15.9069   15.4850   15.5112   17.6572];
y = y';
y = y(:)
```

a) Fit the data with a straight line $y(t) = \beta_1 + \beta_2 t$, and plot the residuals, $y(t_k) - y_k$. You should observe that one of the data points has a much larger (in absolute value) residual than the others. This is probably an *outlier*.

b) Discard the outlier, and fit the data again by a straight line. Plot the residuals again. Do you see any pattern in the residuals?

c) Fit the data, with the outlier excluded, by a model of the form

$$y(t) = \beta_1 + \beta_2 t + \beta_3 \sin(t)$$

d) Evaluate the third fit on a finer grid over the interval `[0,26]`. Plot the fitted curve, using line style `'-'` , together with the data, using line style `'o'` . Include the outlier, using a different marker, `'*'` .

Exercise 6: (a) Plot the unit square : $-1 \leq x \leq 1, -1 \leq y \leq 1$ and the unit circle inside it. (i.e. a circle with radius one)

(b) Generate random points, uniformly distributed on the unit square $-1 < x < 1, -1 < y < 1$. Find an approximation for π by counting the ratio of the number of points inside the circle with the total number of points generated.

(c) Write a function that approximates the area of the ellipse:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

```
[Aappr,Atrue]=function Aellipse(a,b,N)
```

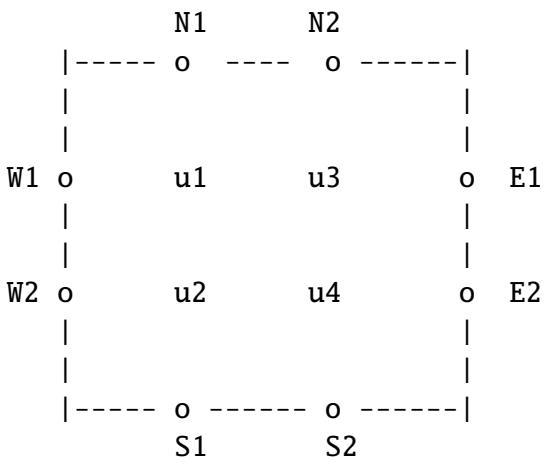
(d) Make plots: color points inside and points outside differently. If you include a plot in your function, test for N to avoid plotting for too large value.

Hint: Generate two random vectors x and y. Find a suitable condition for logical indexing to pick the "inside-points".
Note: You can use arithmetic (sum) to a logical vector. See also links to Session 3, where you can find a good start. Loops are not forbidden here, but avoiding them makes life a lot easier.

Exercise 7: The solution of Laplace's (partial) differential equation

$$\Delta u = 0$$

can be numerically approximated on a square by the *Finite Difference (FD)* method. The figure illustrates the simplest case, the boundary values $S_1, S_2, E_1, E_2, \ldots$ are given and the u_k:s are the unknown solutions to be found.



The solution of Laplace's equation models f.eg. the steady state temperature inside a metal plate whose boundary temperatures are held at the given temperature values. Replacing the 2^{nd} derivatives by their difference approximations leads to the simple rule:

. **Each u_k is the average of its 4 neighbours**

Starting with u_1 we get the linear system of 4 equations:
$4u_1 = u_2 + u_3 + N_1 + W_1$, and the remaining 3 similarly.

(a) Write the system of 4 equations in the form $A\,u = b$.

(b) Let the square area be the unit square, $n = 4$, nr. of interior nodes in x- and y-directions = $n - 2$.
Boundary conditions:

$$u(x,0) = 0, u(x,1) = \frac{1}{(1+x^2)+1}, u(0,y) = \frac{y}{1+y^2}, u(1,y) = \frac{y}{4+y^2}.$$

Exact solution:

$$u(x,y) = \frac{y}{(1+x^2)+y^2}.$$

(c) Find the FD-solutions u_k at the inner nodes and place them into a 2×2 matrix U.

(d) Embed your U into a 4×4 matrix W, with boundary values. (Remember lecture task after "perimeter of matrix".)

(e) Visualize with `surf`,`surfc`, `contourf`, etc. To get the axes right use the form `surf(x,y,U)`, where x an y are the obviuos (identical) vectors.
(No need for `meshgrid` as the U-matrix is there already.) **Note:** There may be a little "mental flip" in distinguishing the matrix and xy-plane directions. (Maybe some hints follow, but it suffices to have the solution right even with misleading orientation.)