MAT-63506 Programming projects 2019 Fall

Choose one of the assignments 1–3. The programming project should be done with Matlab's App Designer. If you have older version of Matlab which doesn't properly support App Designer and can't for some reason update, you can use the old Guide or do the whole app programmatically. The project should be submitted to moodle in a single zip-archive which contains all program files and documentation. The name of the zip-file must be of the form `Px_studentnumbers.zip`, where x is 1, 2, or 3 is the project number. The application name can be anything you like.

The deadline for the project is on **27.10.2019 at 23:55**. Late submissions will not be graded. The project should be done in groups of 2–4 students. The group size does not affect the grade.

All source files should be provided. In particular, if you use Guide remember to return both m- and fig-files. The code should be readable and commented. If you use Guide, the unnecessary comments generated by it *must* be removed or replaced with more relevant comments.

The documentation in pdf-format should include a brief user's guide (you can use annotated screen caps) and a description of any nonobvious aspects of your code. Also mention any limitations your app has. The documentation may be written in English or Finnish.

The grading is approximately as follows: Minimum requirements decently executed gives you a 3, so if you want a better grade you should do at least some of the extra features, described with each project. If the minimum requirements are not satisfied or are implemented poorly, the grade will be less than 3. You can also add your own properties and functionality.

The final grade is the average of the weekly exercises and the project, so getting a 4 from one and a 5 from the other gives you 5. Note that the grades from the weekly exercises can be nonintegers (like 4.25 for example) and the same holds for the project. If you for example do more than the minimum for 3 but not quite enough for 4, you will get a grade between 3 and 4. Quality also counts, the following are taken into account in grading.

- The program works as required. It should give a warning or error message for invalid user input and should perform correctly for valid user input. Don't print warnings and/or errors into the command window, use a dialog box.
- The design and usability of the GUI. Use tab groups to avoid cluttering the GUI with too many UI elements.
- The commenting of the code. Functions should have H1-line(s), i.e., comment line(s) describing the function's purpose. Nonobvious parts of your code should also be commented.
- The readability of the code (variable names, function names). Use temporary variables and spaces liberally, don't write monstrous, deeply nested expressions that are difficult to understand.
- Efficiency of the code (vectorization, preallocation etc.). It is not necessary to optimize the code to within an inch of its life, but there also shouldn't be obvious inefficiencies.
- The quality of the documentation.

Project 1: Basins of Attraction

We can find the complex roots of a polynomial $p(z)$ with Newton's method as follows: Choose an initial point $z_0 \in \mathbb{C}$ and compute

$$z_{n+1} = z_n - \frac{p(z_n)}{p'(z_n)}, \qquad n \geq 0. \tag{1}$$

If the initial point z_0 is close enough to a root z^* of $p(z)$, the sequence (z_n) converges to z^* quadratically. The set of initial points $A(z^*) \subset \mathbb{C}$ for which the iteration (1) converges to z^* is called the *basin of attraction* of z^*. The basins of attraction can be very complicated, they are often fractals. Figure 1 shows the basins of attraction of the three roots of the polynomial $z^3 - 1$.

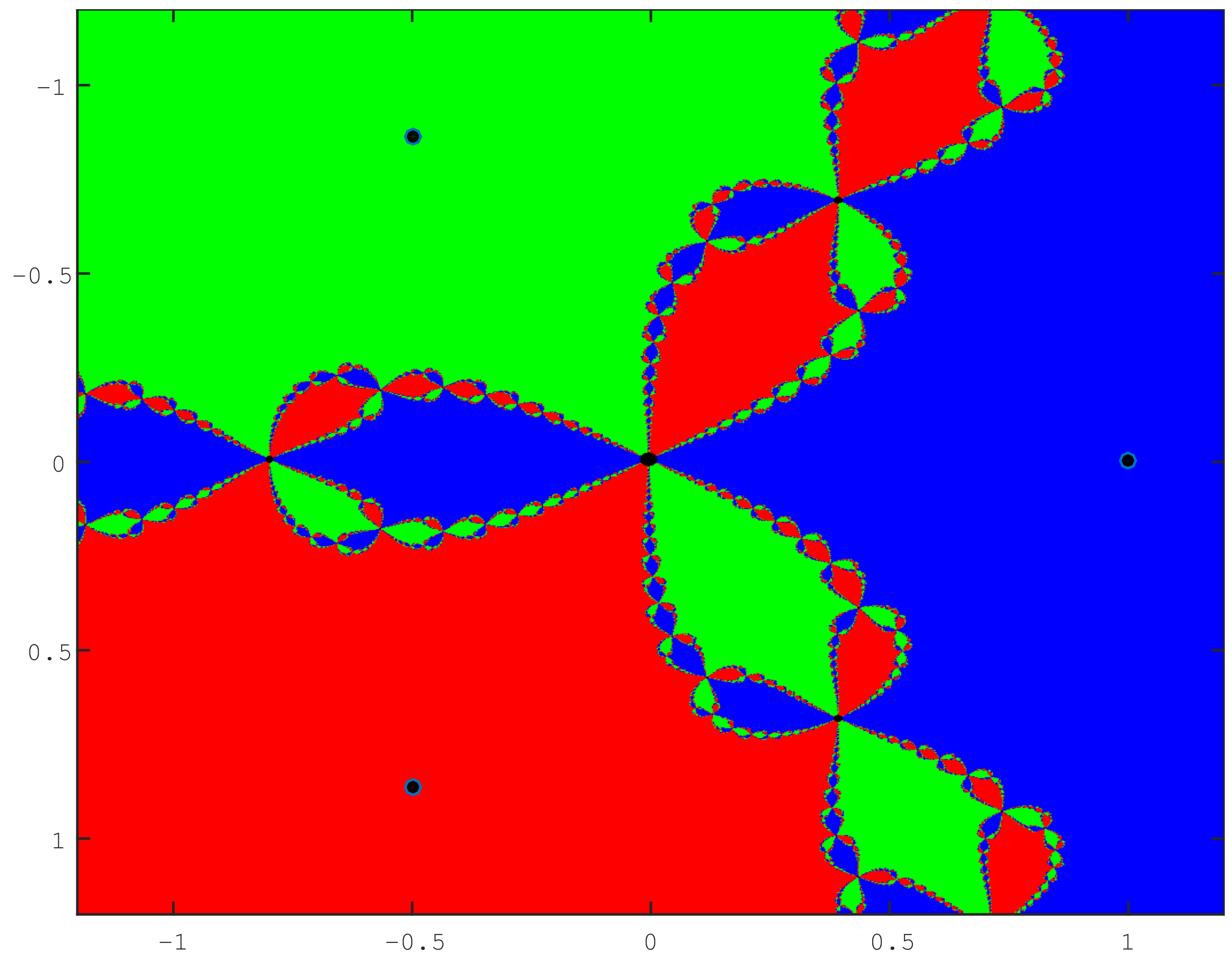

Figure 1: Basins of attraction of the roots of $z^3 - 1$.

Make a GUI for computing and coloring the basins of attraction of a polynomial. The point is not to find the roots with Newton's method, but to determine which points converge to which roots. So the roots are assumed to be known and you can find them with `roots`.

- The minimum requirements for the GUI are: Edit field or `uitable` for the coefficients of the polynomial $p(z)$, given as a vector in descending order, the grid $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$ of initial values, the number of grid points m and n in the x- and y-directions, the tolerance for checking convergence to a root, and a bound M for deciding if the iteration diverges. A plot button that starts the plotting. Your GUI should work at least for polynomials of degree 3 (degree 2 isn't interesting) using red, green, and blue colors. You can plot the diverging initial values as black.

- To get a 4 additionally do the following: Handle polynomials up to degree 6. Allow the user to choose the coloring of the roots from the basic 6 colors (excluding black and white). Check that the same color is not used for more than one root. A stop button in case the plotting takes too long. Add the roots to the plot. A check box for turning the grid on/off. Edit field for the maximum number of iterations, then the initial points that do not converge to a root or diverge to infinity should be colored white.

- For a 5 do the following: Handle polynomials of higher degree than 6. Allow the user to select the colors from a palette. To spare the user from selecting the colors for a high degree polynomial there should also be the option to generate them automatically (perhaps randomly). Show the numerical values of the roots with the corresponding colors. Animate the plotting or alternatively color according to the speed of convergence (for example use a colormap with shades of blue, red and green for three roots). You don't have to do both animation and the speed of convergence coloring, one of them is enough.

  There are various possibilities for animation. You could show the image after each iteration of the inner loop with a user selectable amount of pause, or start from a coarse image and then refine it, or come up with your own way to animate. In any case you should indicate some way when the computation is done, for example use the lamp.

  The speed of convergence coloring doesn't have to work with arbitrarily high degree polynomials, selecting the shading might be too much work in this case. Hence you can choose some convenient upper limit for the degree.

The commands `polyval` and `polyder` are useful, also see the Mandelbrot example in the file "Graphics3D.mlx". There is also the example file "NewtonBasin.mlx" at Moodle, which you can use as a starting point.

Project 2: Plotting the Pseudospectrum of a Matrix

The ε-pseudospectrum of the matrix $A \in \mathbb{C}^{n\times n}$ is defined as

$$\sigma_\varepsilon(A) = \left\{ z \in \mathbb{C} \;\middle|\; \|(zI - A)^{-1}\| > 1/\varepsilon \right\}. \tag{2}$$

If we use the 2-norm (induced by the Euclidean vector norm) as the matrix norm we have $\|A^{-1}\|_2 = 1/\sigma_{\min}(A)$, where $\sigma_{\min}(A)$ is the minimum singular value of A. This gives the following equivalent definition of $\sigma_\varepsilon(A)$

$$\sigma_\varepsilon(A) = \{ z \in \mathbb{C} \mid \sigma_{\min}(zI - A) < \varepsilon \}. \tag{3}$$

A third equivalent definition of $\sigma_\varepsilon(A)$ is

$$\sigma_\varepsilon(A) = \bigcup_{\|E\|<\varepsilon} \sigma(A + E), \tag{4}$$

where $\sigma(A)$ is the spectrum (the set of eigenvalues) of A.

The following example code makes a contour plot of the boundaries of the pseudospectrum of A on an $m \times n$ grid (`N` is the dimension of A).

```
S = zeros(n, m);
for j = 1:m
    for k = 1:n
        z = x(j) + 1i*y(k);
        S(k, j) = svds(z*eye(N) - A, 1, 'smallest');
    end
end
contour(x, y, S, levels);
```

Make a GUI for plotting the pseudospectrum of a matrix using definition (3) above.

- The minimum requirements for the GUI follow. Inputting the matrix A in at least one of the following ways: as an `uitable`, as a variable in the workspace (use `evalin`), or read from a file (m-file or mat-file). A plot button and edit boxes for the grid $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$, the number of grid points m and n, and an `uitable` for the ε-values of the contour levels.

- To get a 4 do additionally the following. All of the methods of inputting A given above, and also a random A of user given size which can optionally be written to a file or to variable in the workspace. A check box for turning the grid on/off. A stop button (if the computation takes too long). Add the eigenvalues to the plot with the possibility to set the marker and its size and color. Show the minimum and maximum values of `S` and let the user choose the number of levels to use between them. The same as previous but with user selected minimum and maximum levels. Add contour line labels.

- For a 5 do the following. Make a surface plot of `S` or `log(S)` (the user decides which) with the possibility of setting at least the shading, coloring and lighting. Compute the pseudospectrum (for a single value of ε with plot the coressponding contour) using definition (4). Do this as follows. Generate a user given number of the E matrices randomly and plot the eigenvalues of $A+E$. To speed up plotting, compute the eigenvalues for N (which the user sets) matrices at a time and then show them all at once. Use `animatedline` for this. You should also indicate when the computation is done, for example use the lamp.

Here are some examples. Figure 2 shows the boundaries of $\sigma_\varepsilon(A)$ for

$$A = \begin{bmatrix} i & 0 & 1+i \\ 1-1i & 0.5 & 0 \\ 1.1i & 0.5 & 2 \end{bmatrix} \tag{5}$$

and $\varepsilon = 1$, $1/2$, $1/3$, $1/4$, $1/5$, $1/10$, and $1/20$. The eigenvalues of A are shown as red circles.

Figure 3 shows the boundaries of the pseudosspectrum (and eigenvalues) of a 50×50 random matrix with elements from standard normal distribution.

Finally, Figure 4 shows $\sigma_\varepsilon(A)$ for the matrix (5) with $\varepsilon = 1/2$ using definition (4) with one million normally distributed random matrices E.

Project 3: Random Walk

Make a GUI for the random walk of Exercise Set 4.

- The minimum requirements are: A Start/Stop button for starting and stopping the animation (use the Start/Stop function given in the solutions to Exercise Set 4 as a starting point). Static text fields for printing the x- and y-coordinates of the current location.

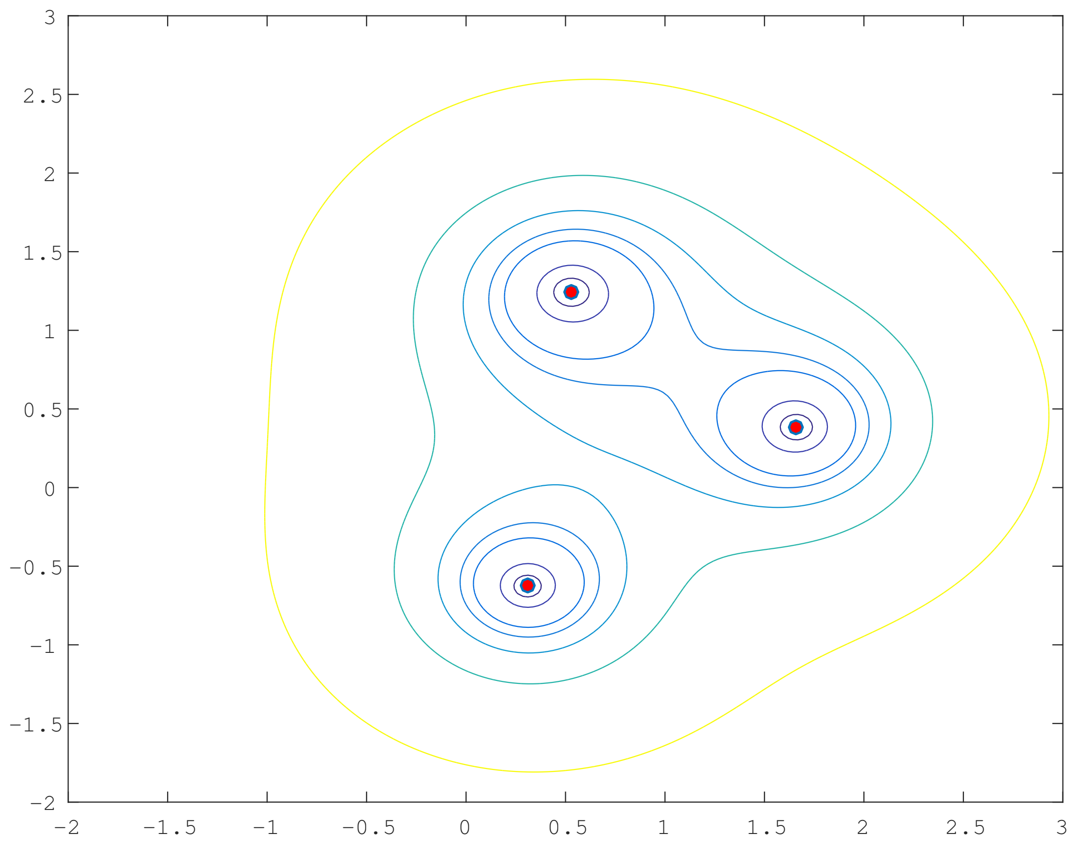

Figure 2: The boundaries of the pseudospectrum of the matrix (5).

A drop down menu for selecting the marker. A spinner for setting the marker size. A button for selecting the marker face (default red) and edge (default black) colors from a palette. A way of setting the initial axes limits and the amount to change them when the marker goes outside the current limits. A numeric edit field for setting the step size.

- To get a 4 do additionally the following. An option for leaving a trail of previous points visible. Use `animatedline` for this. In this case the user should be able to choose the line style (including `'none'`), color and width. It should also be possible to set the marker to `'none'`. Of course the marker and line style shouldn't be set to `'none'` at the same time, so you should give an error message if the user tries to do this.

  Usually the angle θ is uniformly distributed in the interval $[0.2\pi)$. Add the option of θ having a uniform discrete distribution of values $2\pi k/n$ for $k = 0, \ldots, n-1$ with equal probability. Here we must have $n \geq 3$.

  The case $n = 4$ is the widely used rectangular discrete walk where the walk moves to right, left, down, or up with equal probability.

- For a 5 do the following. Add an option to change the boundary behaviour so that if x goes outside the axis on the right (left) it comes back from the left (right) and similarly for y. This means that we have glued the line $x = x_{\text{min}}$ to the line $x = x_{\text{max}}$ and the line $y = y_{\text{min}}$ to the line $y = y_{\text{max}}$. This makes the square into a torus.

  An option for making the random walk in 3 dimensions. Then you must add the z coordinate to all relevant places, such as axis limits. You also need two angles, latitude $\theta \in [0.\pi]$ and longitude $\phi \in [0.2\pi)$. When computing the x, y, and z values using spherical

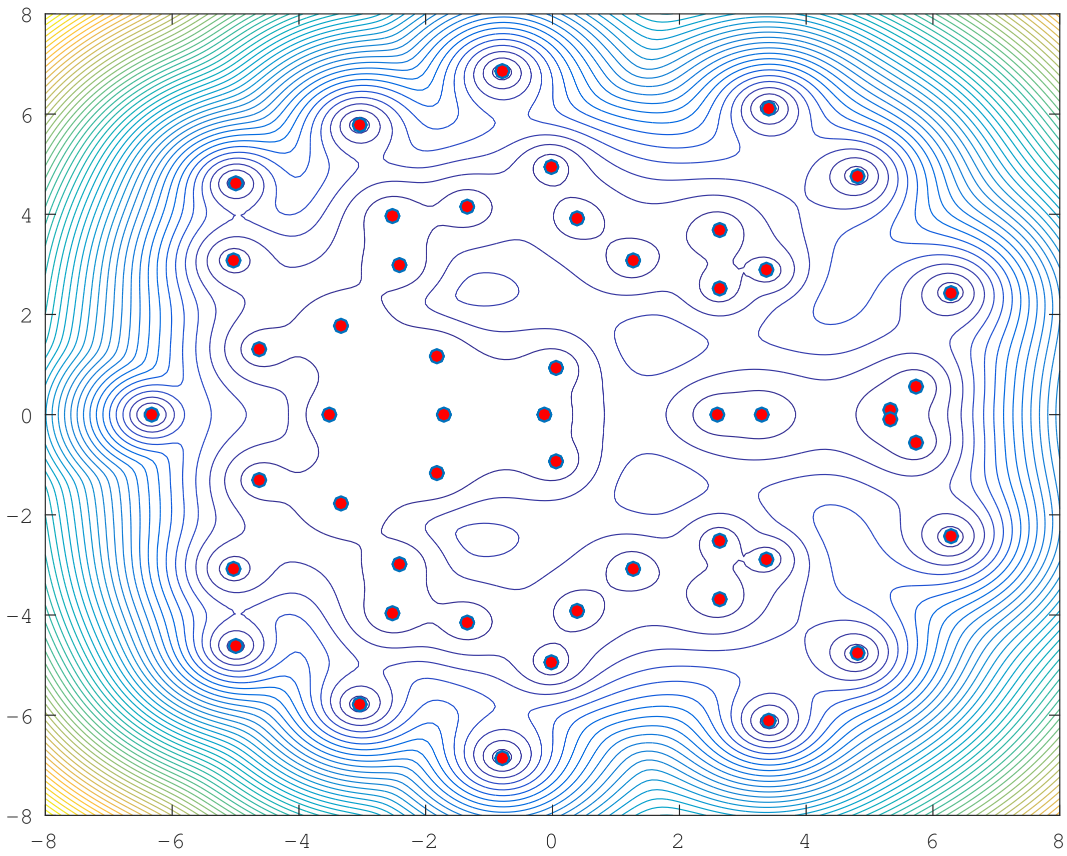

Figure 3: The boundaries of the pseudospectrum of a random 50×50 matrix.

coordinates

$$\begin{aligned} x_{n+1} &= x_n + r\sin\theta\cos\phi, \\ y_{n+1} &= y_n + r\sin\theta\sin\phi, \\ z_{n+1} &= z_n + r\cos\theta, \end{aligned}$$

we cannot assume that θ and ϕ are uniformly distributed in their respective intervals, because then the x, y, and z values are not uniformly distributed on the sphere. Instead we proceed as follows. Generate uniformly distributed $u, v \in (0, 1)$ and then compute $\phi = 2\pi u$ and $\theta = \arccos(1 - 2v)$.

The rectangular discrete walk is easiest to make by generating a random integer in $\{1, 2, \ldots, 6\}$ and using that to select the xyz direction.

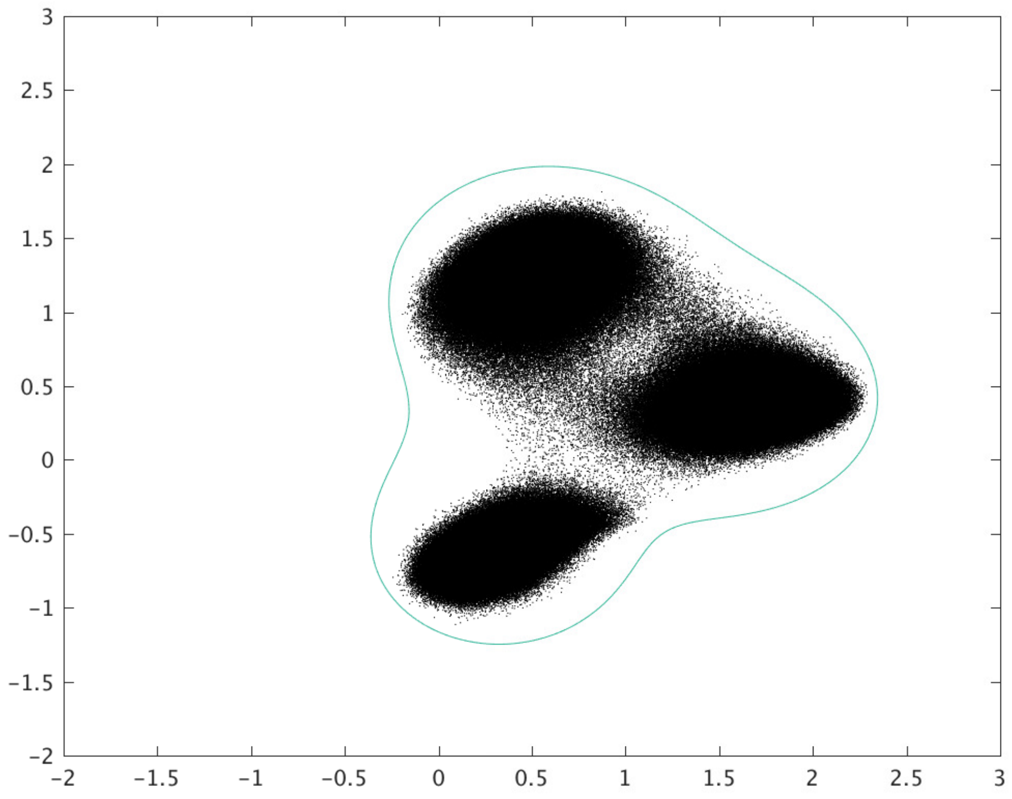

Figure 4: The pseudospectrum of (5) using definition (4).