MAT-63506 Programming projects 2018 Spring

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 DESIGNERt and can't for some reason update, you can use the old GUIDE. The whole 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.

The deadline for the project is on **18.05.2018 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. The code should be readable and commented. The unnecessary comments generated by GUIDE *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. The documentation may be written in English or Finnish. Remember to return both m- and fig-files if using the GUIDE.

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, no need to do all of them. 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. 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.
- The commenting of the code. Functions should have H1-line(s), i.e., comment line(s) describing the function's purpose.
- 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 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 (**??**) converges to z^*

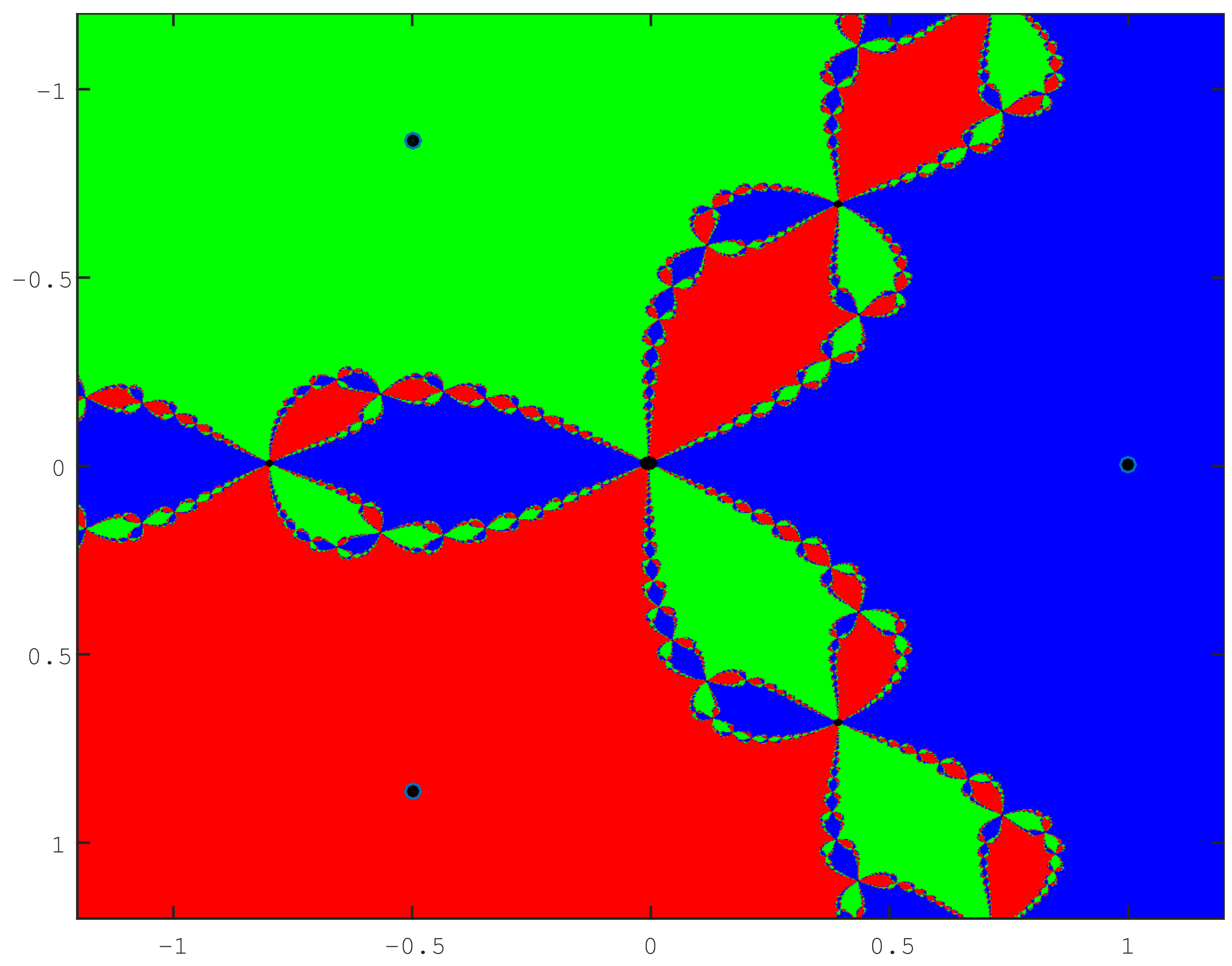


Figure 1: Basins of attraction of the roots of $z^3 - 1$.

is called the *basin of attraction* of z^*. The basins of attraction can be very complicated. Figure **??** shows the basins of attraction of the three roots of the polynomial $z^3 - 1$.

Make a GUI for coloring the the basins of attraction of a polynomial. The GUI should have at least the following inputs: Edit boxes for the polynomial $p(z)$ (given as a vector of coefficients 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, 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 using red, green, and blue colors. You can plot the diverging initial values as black or white.

For a better grade consider some of the following: A check box for turning the grid on/off, allow the user to choose the coloring of the roots, a stop button in case the plotting takes too long, save the plot to a file, add the roots to the plot, handle polynomials of degree greater than three, allow the user to select the colors from a palette, color according to the speed of convergence (for example use a colormap with shades of blue, red and green for three roots). Anything else that comes to mind.

The commands `polyval` and `polyder` are useful, also see the Mandelbrot example in the file "Graphics3D.mlx".

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 (**??**) above. The GUI should have at least the following inputs: A way for the user to input the matrix A, either as an expression, as a variable in the workspace (use `evalin`), or read from a file (or all). 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 the ε-values for the contour levels.

For better grade consider some of the following. A stop button (if the computation takes too long), adding the eigenvalues to the plot, choosing the number of levels to use (possibly between the smallest and largest values of `S`), saving the plot to a file, adding the contour line labels, animating the pseudospectrum, using definition (**??**) to compute the pseudospectrum (for a single value of ε), making a surface plot of `S` (or `log(S)`). Anything else that comes to mind.

Here are some examples. Figure **??** 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 **??** shows the boundaries of the pseudosspectrum (and eigenvalues) of a 50×50 random matrix with elements from standard normal distribution.

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

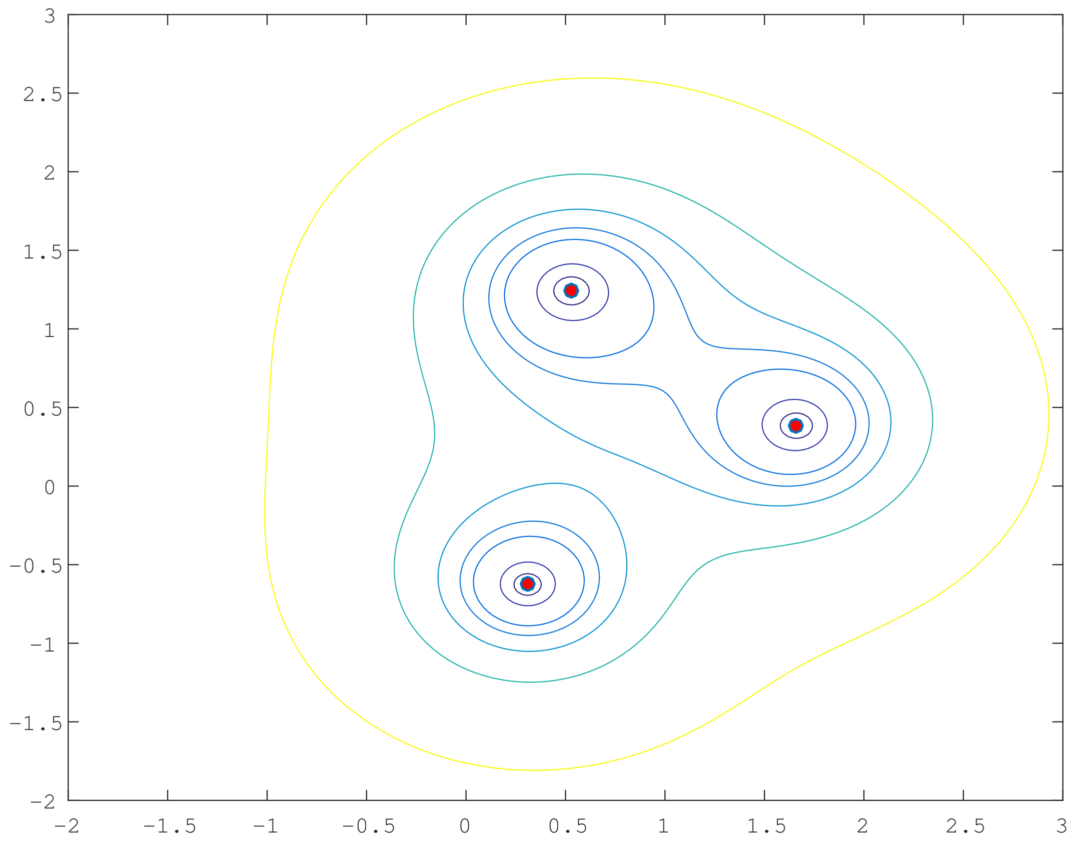


Figure 2: The boundaries of the pseudospectrum of the matrix (**??**).

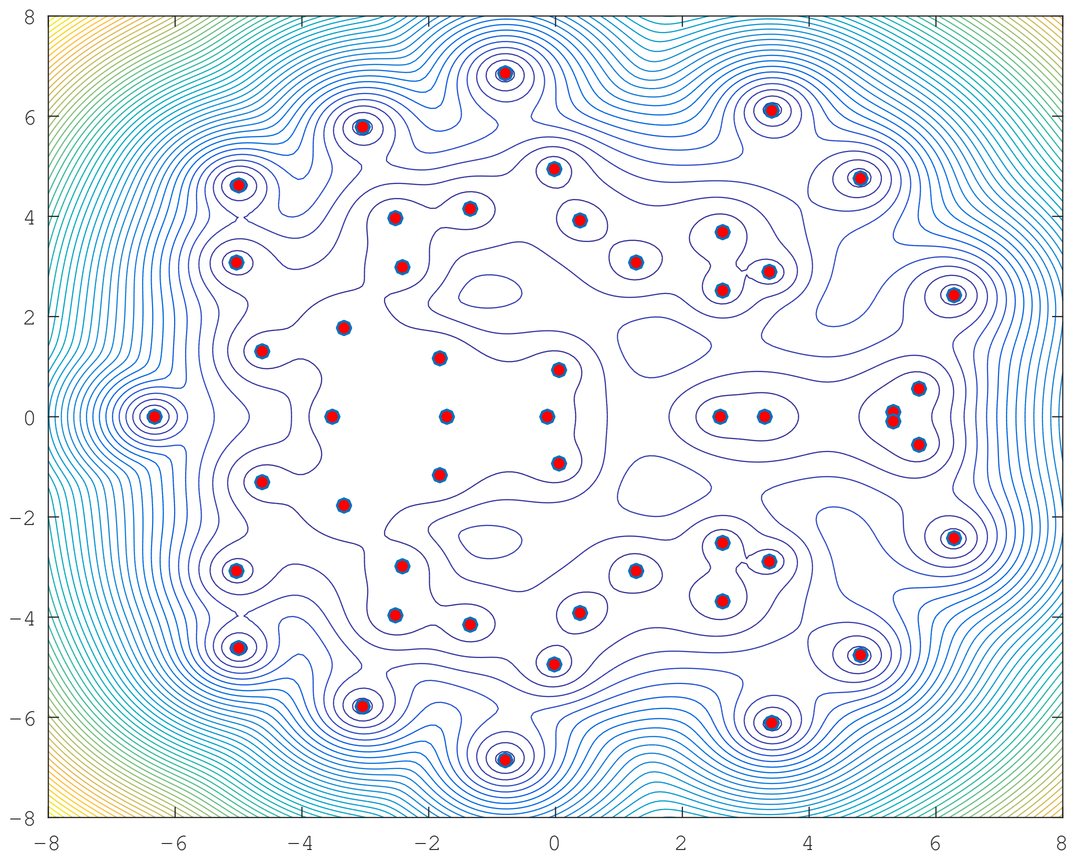


Figure 3: The boundaries of the pseudospectrum of a random 50×50 matrix.

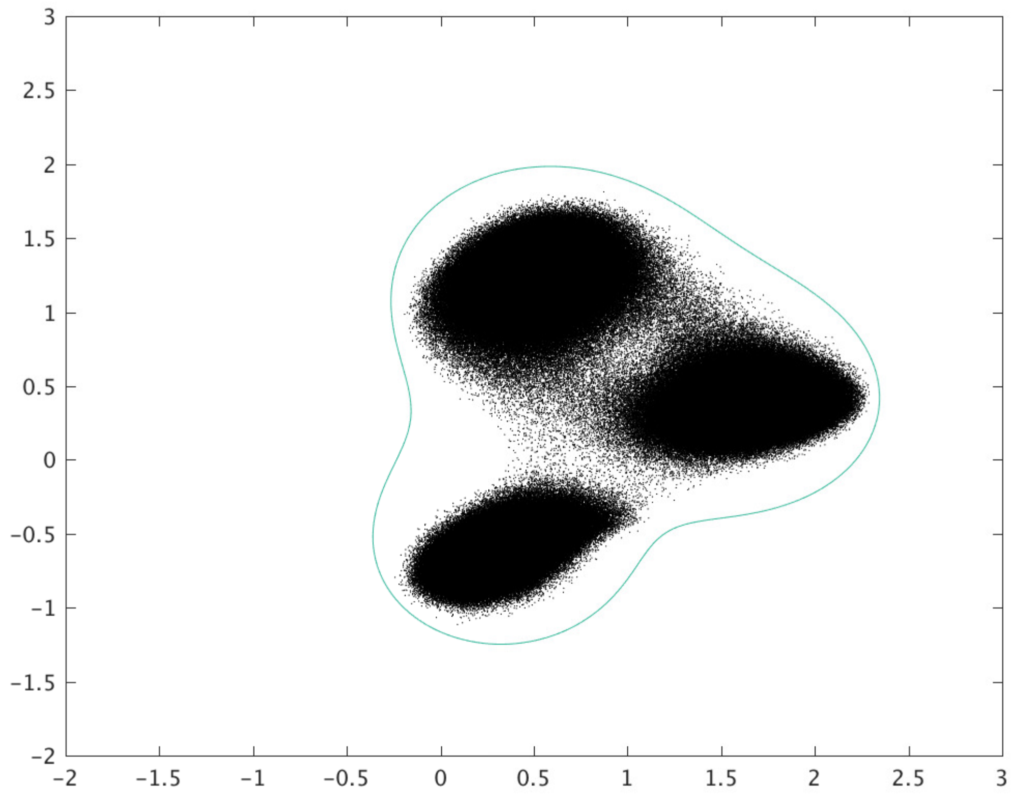


Figure 4: The pseudospectrum of (**??**) using definition (**??**).

Project 3: Kalman filter GUI-topic

The Kalman filter solves the filtering problem where the state model and the measurement model (measurement equations) are linear. See more info [1] and webpage:

Wikipedia: Kalman filter

Kalman filter algorithm is [1, p. 46]

Algorithm 3 The Kalman filter

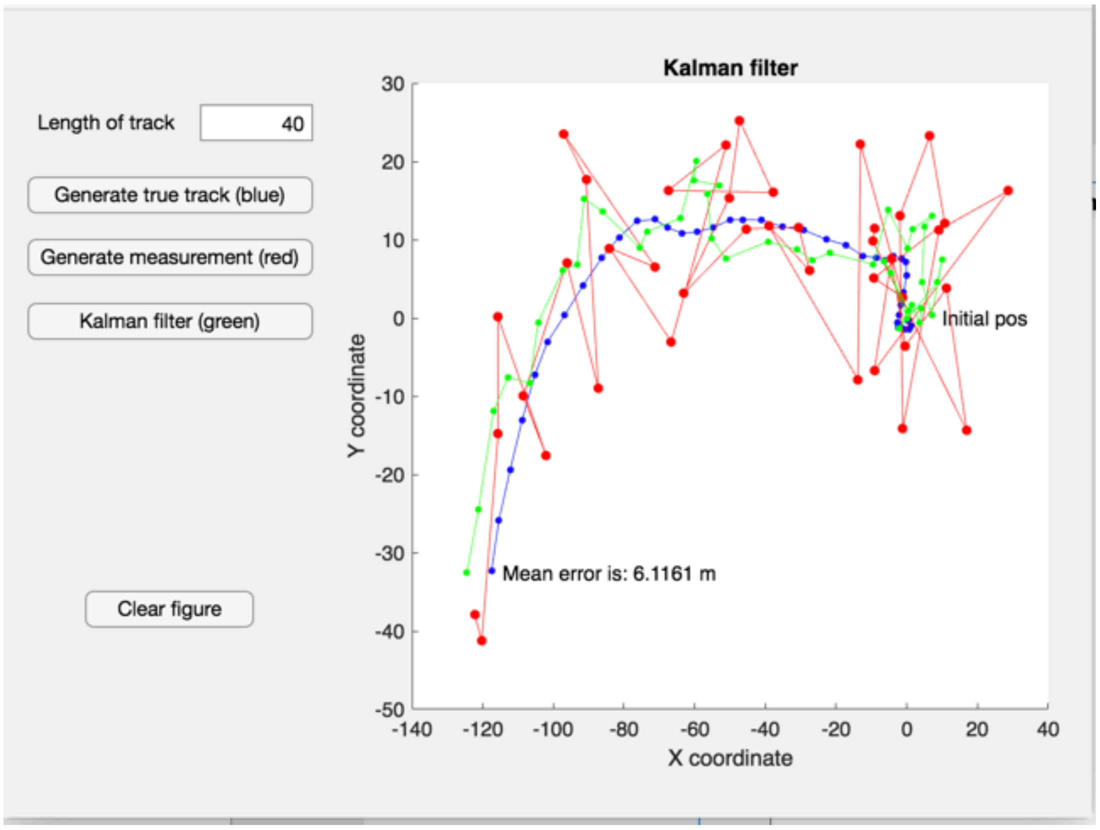
```
• The state model:            $\mathbf{x}_k = \Phi_{k-1}\mathbf{x}_{k-1} + \mathbf{w}_{k-1}$,   $V(\mathbf{w}_{k-1}) = Q_{k-1}$
• The measurement model:      $\mathbf{y}_k = H_k\mathbf{x}_k + \mathbf{v}_k$,   $V(\mathbf{v}_k) = R_k$
• The measurements:           $y_{1:m} = \{y_1, y_2, \ldots, y_m\}$
• The initial state estimate and its MSE:  $\hat{x}_0$ and $P_0$
1. Set $k = 1$.
2.  • The prior estimate:      $\hat{x}_k^- = \Phi_{k-1}\hat{x}_{k-1}$
    • The prior MSE:           $P_k^- = \Phi_{k-1}P_{k-1}\Phi_{k-1}^T + Q_{k-1}$
    • The Kalman gain:         $K_k = P_k^- H_k^T (H_k P_k^- H_k^T + R_k)^{-1}$
    • The posterior estimate:  $\hat{x}_k = \hat{x}_k^- + K_k(y_k - H_k\hat{x}_k^-)$
    • The posterior MSE:       $P_k = (I - K_k H_k)P_k^-$
3. Stop if $k = m$, otherwise set $k = k + 1$ and get back to step 2.
```

Mimimum requirements (see figure):

- Length of track
- Generate track-button using constant velocity model
- Generate measuremnt-button using position measurements
- Compute Kalman filter
- Plot results
- Clear figure

Options:

- Compute different errors (positioning, velocity: mean, max, median)
- Plot covariance ellipse (area where final position is with 68% probability)
- Use also nonlinear range measurement and Extended Kalman filter and/or Particle Filter (if your aim is grade 5).
- Use different parameters and motion models



There are some examples for the first stage.

Generate track:

-Use "Constant velocity model" where delta t is one [1, p. 43] :

State $x = \begin{bmatrix} r_x : x \text{ position} \\ r_y : y \text{ position} \\ v_x : x \text{ velocity} \\ v_y : y \text{ velocity} \end{bmatrix}$.

Our state model is $r_{k+1} = r_k + \Delta t\, v_k + \text{error}$ and $v_{k+1} = v_k + \text{error}$ ("constant velocity"), so state transition matrix is $(\Delta t = 1)$ $\Phi = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$.

The error $w \sim N(0, Q)$, where covariance matrix $Q = \begin{bmatrix} \frac{1}{3} & 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{3} & 0 & \frac{1}{2} \\ \frac{1}{2} & 0 & 1 & 0 \\ 0 & \frac{1}{2} & 0 & 1 \end{bmatrix}$. We can use function "mvnrnd" (help mvnrnd).

So we can generate track using script:

```
x0 = zeros(4,1); %Mean of initial state
P0 = diag([100 100 1 1]);%Covariance matrix of initial state
Phi = [eye(2) eye(2);zeros(2,2) eye(2)];% State transition matrix
Q = [1/3*eye(2),1/2*eye(2);1/2*eye(2),eye(2)]; % Covariance matrix of state model
n = 40; % length of track
dim = 2+2;% Dimension 2D position and 2D velocity
X=nan(dim,n+1); %Track t=0...t=n
X(:,1)=x0;
for t=1:n
    X(:,t+1)=Phi*X(:,t)+mvnrnd(zeros(1,4),Q)';
end

figure;
plot(X(1,:),X(2,:),'b.-');
hold on
xlabel('x coordinate')
ylabel('y coordinate')
axis equal
title('Track')

% Generate measurement you can use
% y(:,i)=H*x(:,i)+mvnrnd(zeros(1,2),R)';
```

Generate measurement:

At the first stage use measurement model $y = \begin{bmatrix} r_x \\ r_y \end{bmatrix} + \text{error} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} x + v$, where $v \sim N(0, R)$ and the covariance matrix of measurement error $R = \begin{bmatrix} \sigma_x{}^2 & 0 \\ 0 & \sigma_y{}^2 \end{bmatrix}$.

```
% Generate measurement you can use
H = [eye(2) zeros(2,2)];
sigmax2 = 4^2;
sigmay2 = 2^2;
R = diag([sigmax2,sigmay2]);
Y = nan(2,n);
for t=1:n
    Y(:,t)=H*X(:,t+1)+mvnrnd(zeros(1,2),R)';
end
plot(Y(1,:),Y(2,:),'r.-')
title('Track and measurements')
```

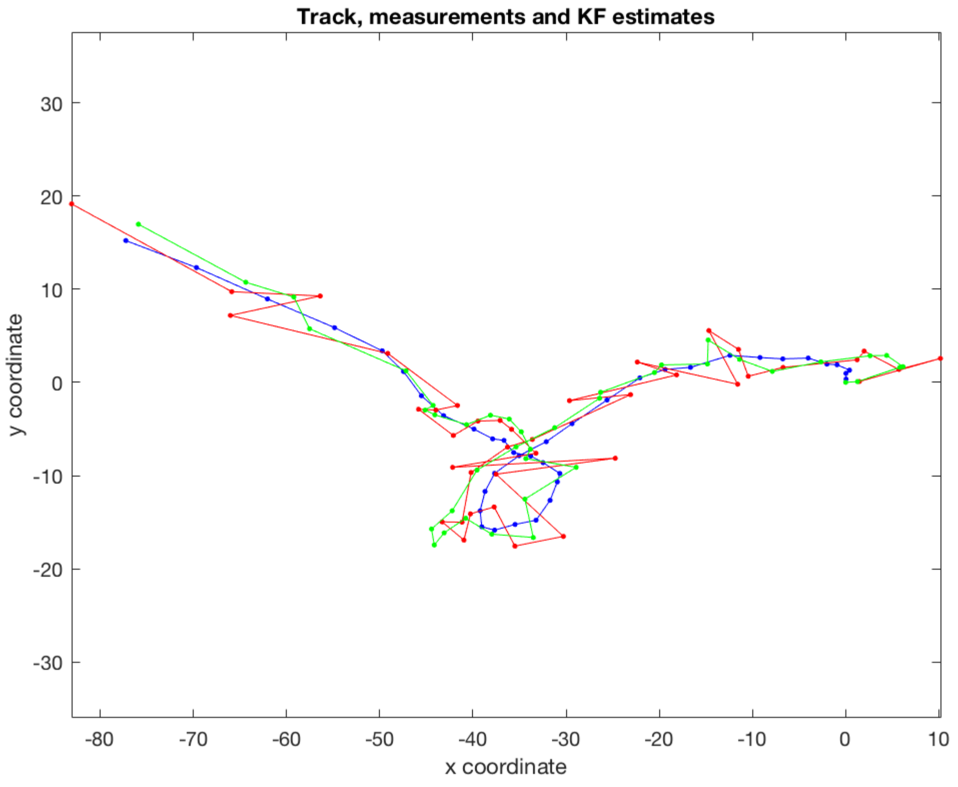
Kalman filter:

```
Estimate = nan(size(X));
```

```
Estimate(:,1)=x0;
P = nan(dim,dim,n+1);
P(:,:,1) = P0;

for t=1:n
    Estimate(:,t+1) = Phi*Estimate(:,t); % Prior estimate
    P(:,:,t+1) = Phi*P(:,:,t)*Phi'+Q; % Prior covariance
    K = P(:,:,t+1)*H'/(H*P(:,:,t+1)*H'+R); % Kalman gain
    Estimate(:,t+1) = Estimate(:,t+1)+K*(Y(:,t)-H*Estimate(:,t+1));% Posterior estimat
    P(:,:,t+1) = (eye(dim)-K*H)*P(:,:,t+1);% Posterior covariance
end

plot(Estimate(1,:),Estimate(2,:),'g.-')
title('Track, measurements and KF estimates')
```



Help:

```
help mvnrnd
```

```
mvnrnd Random vectors from the multivariate normal distribution.
   R = mvnrnd(MU,SIGMA) returns an N-by-D matrix R of random vectors
   chosen from the multivariate normal distribution with mean vector MU,
   and covariance matrix SIGMA.  MU is an N-by-D matrix, and mvnrnd
   generates each row of R using the corresponding row of MU.  SIGMA is a
   D-by-D symmetric positive semi-definite matrix, or a D-by-D-by-N array.
   If SIGMA is an array, mvnrnd generates each row of R using the
   corresponding page of SIGMA, i.e., mvnrnd computes R(I,:) using MU(I,:)
```

```
and SIGMA(:,:,I).  If the covariance matrix is diagonal, containing
variances along the diagonal and zero covariances off the diagonal,
SIGMA may also be specified as a 1-by-D matrix or a 1-by-D-by-N array,
containing just the diagonal. If MU is a 1-by-D vector, mvnrnd
replicates it to match the trailing dimension of SIGMA.

R = mvnrnd(MU,SIGMA,N) returns a N-by-D matrix R of random vectors
chosen from the multivariate normal distribution with 1-by-D mean
vector MU, and D-by-D covariance matrix SIGMA.

Example:

   mu = [1 -1]; Sigma = [.9 .4; .4 .3];
   r = mvnrnd(mu, Sigma, 500);
   plot(r(:,1),r(:,2),'.');

See also mvtrnd, mvnpdf, mvncdf, normrnd.

Reference page for mvnrnd
```

References:

[1] **MAT-45806 Mathematics for positioning & MAT-45807 Mathematics for positioning. / Ali-Löytty, Simo; Collin, Jussi; Sirola, Niilo.**

TUT Tampere : Tampere University of Technology, 2010. 63 p. (Tampere University of Technology. Department of Mathematics).

pdf