

# ps1

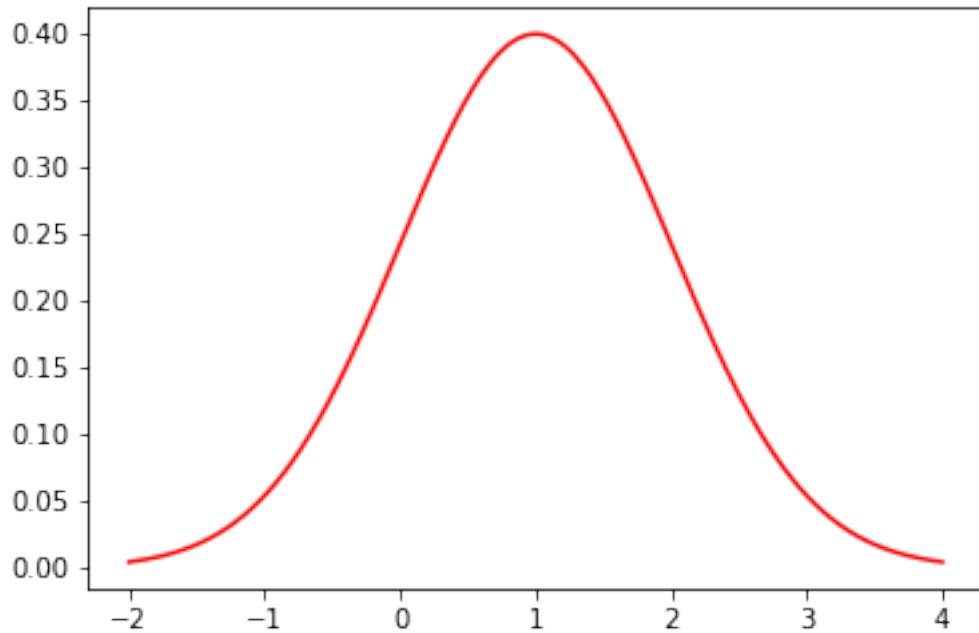
November 11, 2018

```
In [2]: from scipy.stats import norm
import numpy as np
import matplotlib.pyplot as plt
from math import *
```

## 0.0.1 Task 1.A

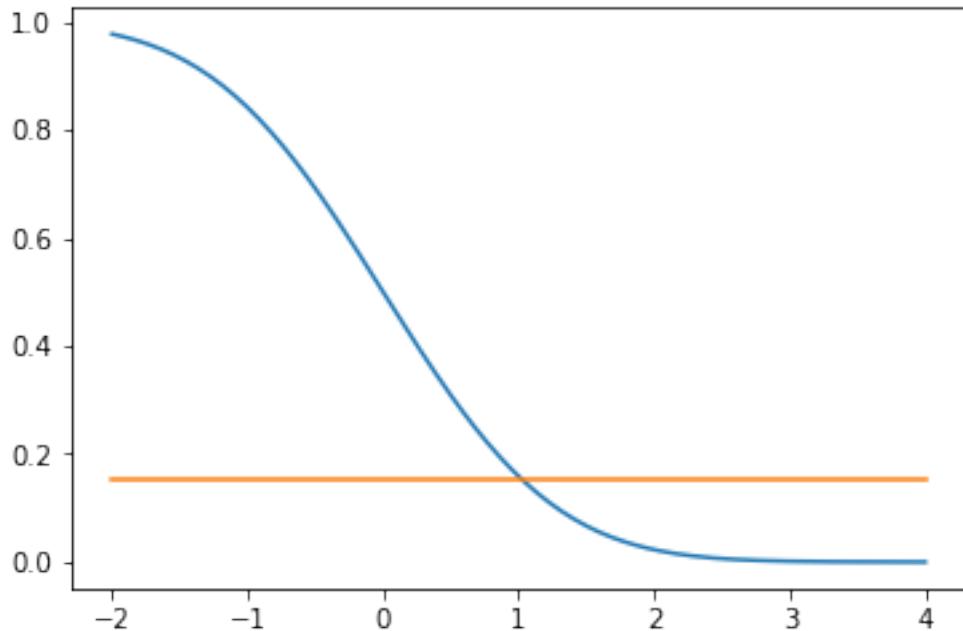
```
In [2]: fig, ax = plt.subplots(1, 1)
mx = 1; sx = 1
x = np.linspace(-2,4,200)
p = norm.pdf(x, mx, sx)
plt.plot(x, p, 'r-', label='norm pdf')
```

```
Out[2]: [<matplotlib.lines.Line2D at 0x7f49d68e8c88>]
```



### 0.0.2 Task 1.B

```
In [3]: fig, ax = plt.subplots(1, 1)
i1 = list(x).index(1.0150753768844218)
F = 1-norm.cdf(x)
ax.plot(x, F )
ax.plot(x, F[i1]*np.ones_like(x) )
plt.show()
print("F(x>1)="+str(F[i1]))
```



F(x>1)=0.15503494900490788

### 0.0.3 Task 1.C

$$p(x|z=0.75) = p(0.75|x) * p(x) / p(0.75)$$

```
In [10]: def pdf(x, mx, sigma):
    p = exp(-(x-mx)**2/2)/sqrt(2*pi*sigma**2)
    return p
```

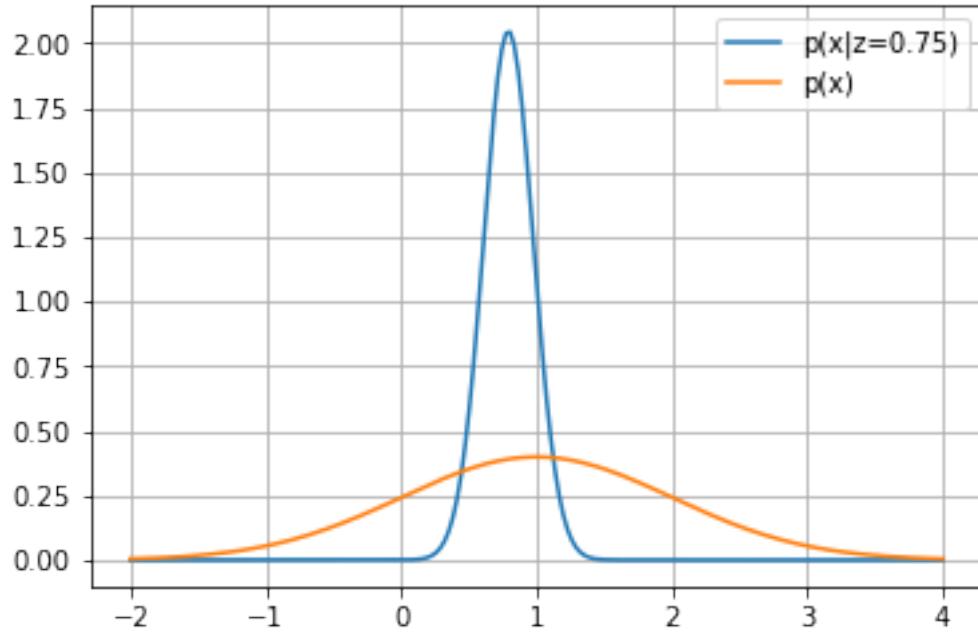
```
In [4]: Z = 0.75
        sigma = sqrt(0.2)
        N = norm.pdf(0.75, x, sigma**2)
        p_x_given_z = N * norm.pdf(x,mx,sigma) / norm.pdf(0.75,mx,sigma)
        plt.plot(x, p_x_given_z)
```

```

plt.plot(x, p)
plt.grid()
plt.legend(['p(x|z=0.75)', 'p(x)'])

```

Out [4]: <matplotlib.legend.Legend at 0x7f49d64bfbe0>



#### 0.0.4 Task 1.D

```

In [5]: E = round( x[ list(p_x_given_z).index(max(p_x_given_z)) ],3)
print("Estimate the expected value of the posterior distance to the wall"+"\n"+"E(x|0.75")

```

Estimate the expected value of the posterior distance to the wall  
 $E(x|0.75)=0.804$

#### 0.0.5 Task 1.E

Product rule:  $p(x,z) = p(x|z) * p(z)$

```

In [6]: def condprob(x,z=0.75,mx=1,sigma=sqrt(0.2)):
    N = norm.pdf(z, x, sigma**2)
    p_x_given_z = N * norm.pdf(x,mx,sigma) / norm.pdf(z,mx,sigma)
    return p_x_given_z

```

```

In [7]: z = x
p_joint = np.zeros((len(x), len(z)))

```

```

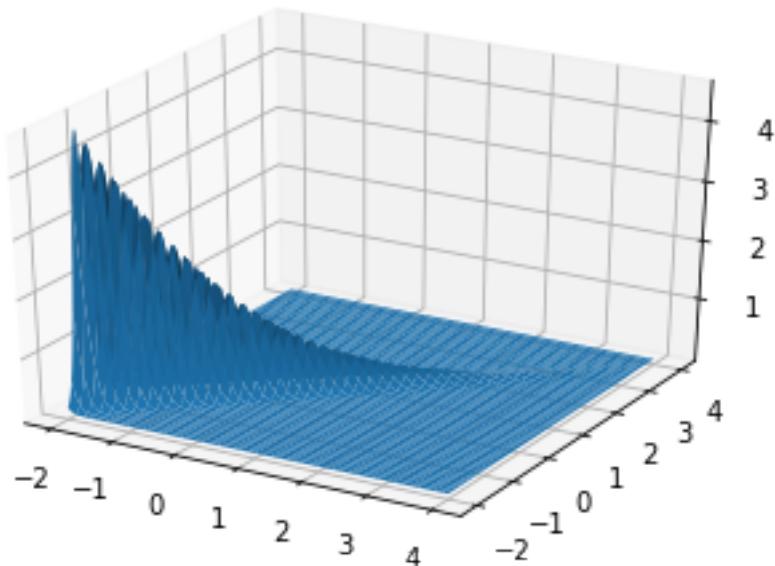
for X in range(len(x)):
    for Z in range(len(z)):
        p_joint[X,Z] = condprob(x[X],z[Z]) * norm.pdf(z[Z])

In [8]: X = np.zeros((len(x),len(z)))
Z = np.zeros((len(x), len(z)))
for i in range(len(x)):
    X[i,:] = x
    Z[i,:] = z

In [9]: from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.gca(projection='3d')
X, Z = np.meshgrid(x, z)

# Plot the surface.
surf = ax.plot_surface(X, Z, p_joint)
plt.show()

```



## 0.0.6 Task 2.A

```

In [10]: from scipy.linalg import cholesky

def plot2dcov(mean, sigma2D, k=1, n_samples=100):
    sigma2D = np.array(sigma2D)

```

```

L = cholesky(sigma2D).T
t = np.linspace(0,2*pi,n_samples)
xe = np.array([]); ye = np.array([])
for T in t:
    [Xe, Ye] = np.dot(L, k*np.array([cos(T),sin(T)]))
    xe=np.append(xe,Xe); ye=np.append(ye,Ye)
xe += mean[0]; ye += mean[1]

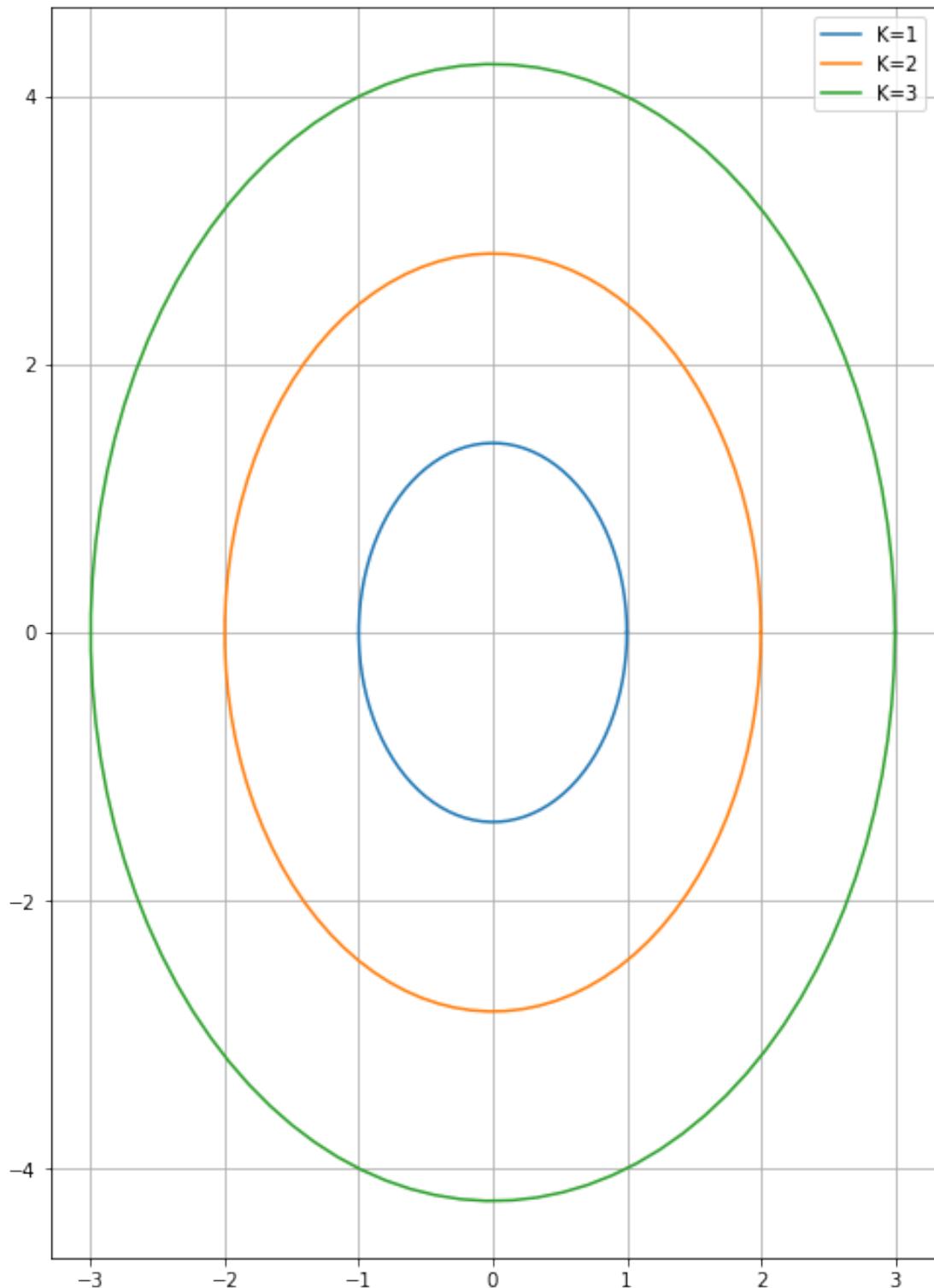
return xe, ye

In [11]: M1 = [0,0]; E1 = np.array([[1,0], [0,2]])
M2 = [5,0]; E2 = np.array([[3,-0.4], [-0.4,2]])
M3 = [2,2]; E3 = np.array([[9.1,6], [6,4]])

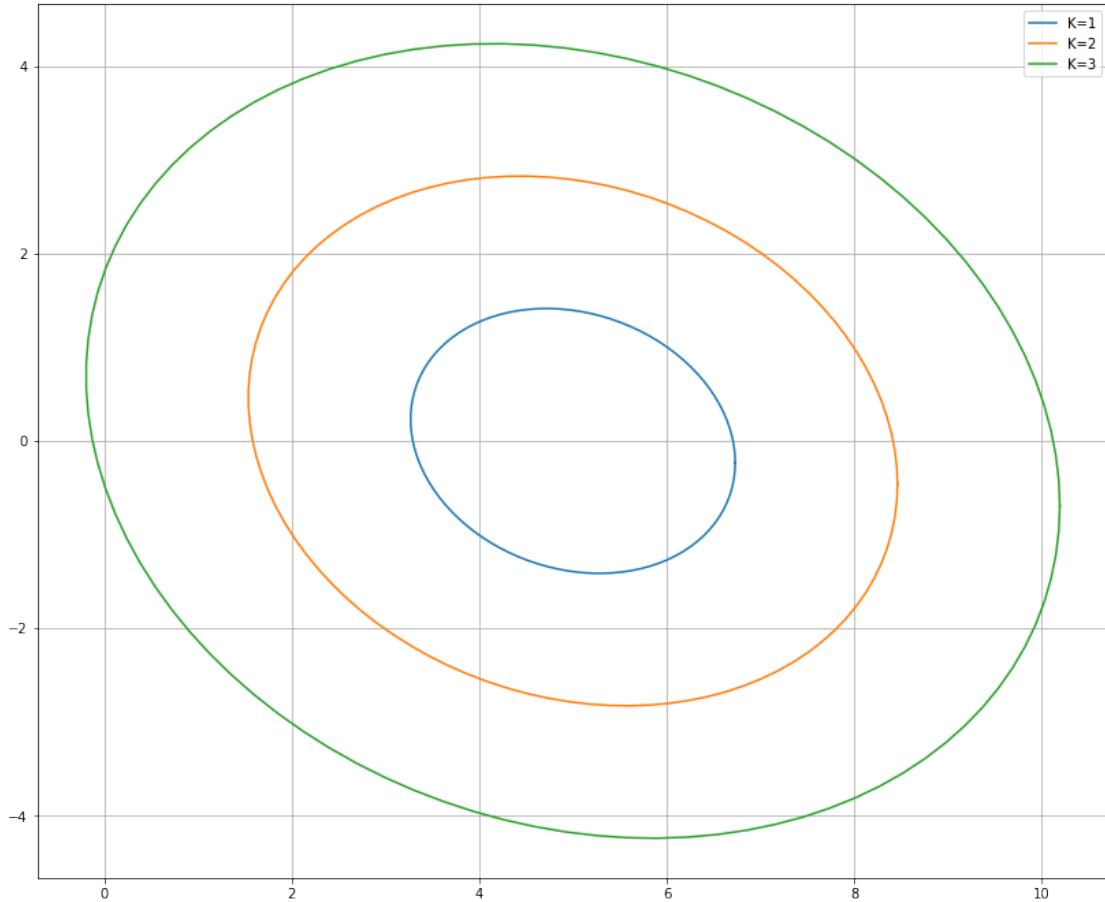
In [12]: def plotellipse(M,E, K=[1]):
    fig = plt.figure()
    fig.set_size_inches(40, 40)
    ax = fig.add_subplot(311, aspect='equal')
    for i in range(len(K)):
        xe, ye = plot2dcov(mean=M, sigma2D=E, k=K[i])
        plt.plot(xe,ye)
    plt.grid()
    plt.legend(['K=1','K=2', 'K=3'])
    plt.show()

In [13]: plotellipse(M1,E1, K=[1,2,3])

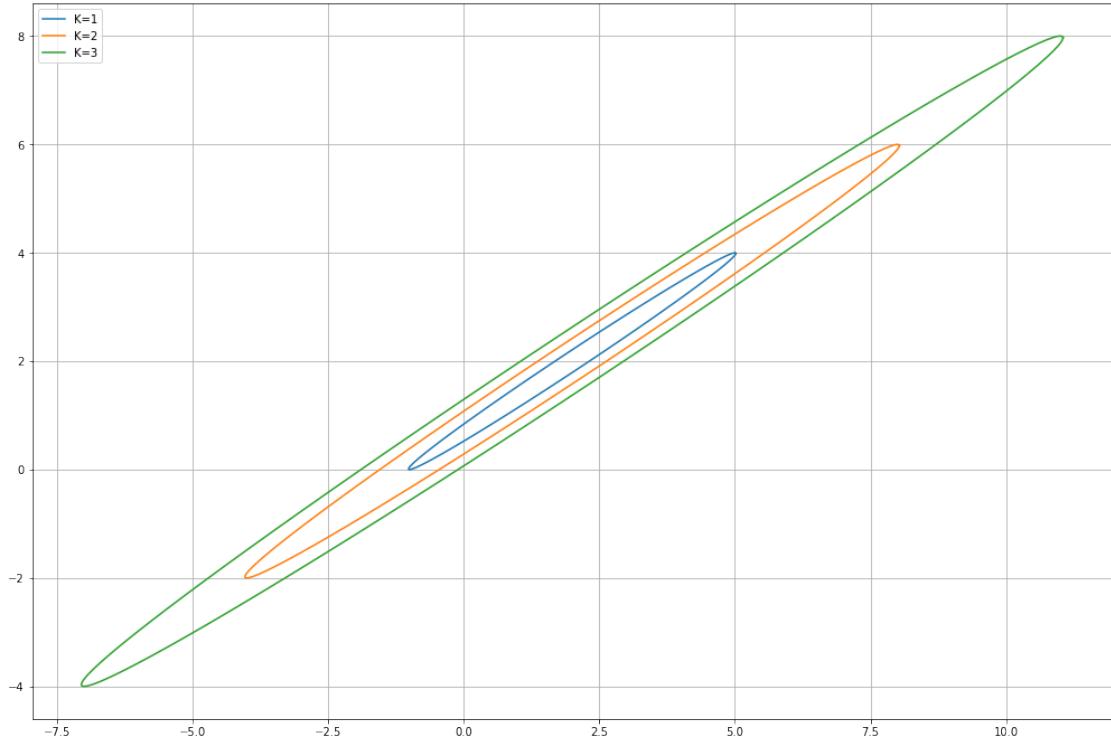
```



```
In [14]: plotellipse(M2,E2, K=[1,2,3])
```



```
In [15]: plotellipse(M3,E3, K=[1,2,3])
```



### 0.0.7 Task 2.B

Sample mean:

$$\bar{x} = \frac{1}{N} \sum_{i=0}^N \bar{x}_i$$

```
In [16]: def sample_mean(samples):
    sample_mean = np.zeros((len(samples), 1))
    for i in range(len(samples)):
        sample_mean[i] = np.mean(samples[i])
    return sample_mean
```

Sample covariance matrix:

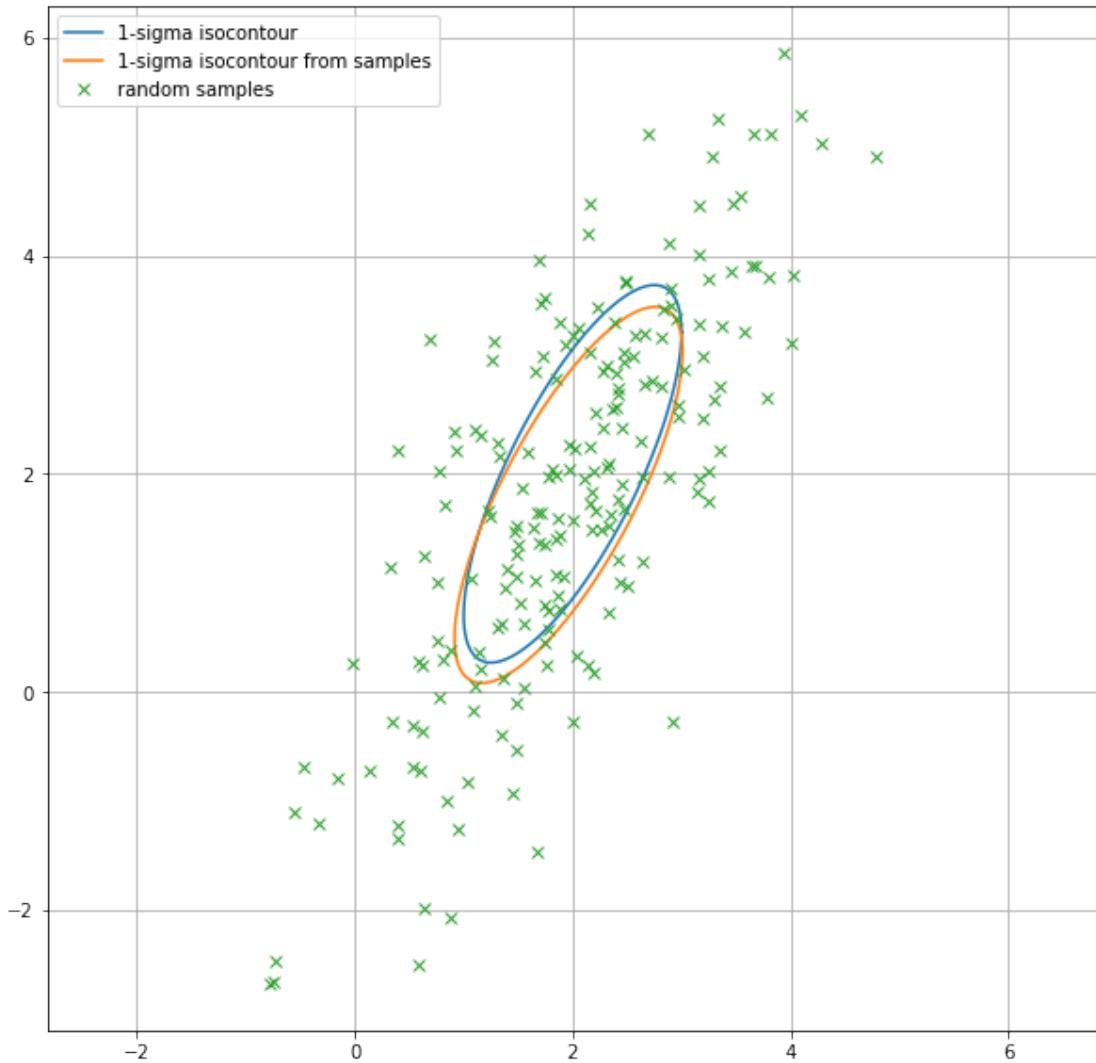
$$Q = \frac{1}{N-1} \sum_{i=1}^N (\bar{x}_i - \bar{x})(\bar{x}_i - \bar{x})^T$$

```
In [19]: def sample_cov(samples):
    Q = np.cov(samples)
    return Q
```

## 0.0.8 Task 2.C

```
In [20]: mean = np.array([2,2]).T
cov = np.array([[1, 1.3], [1.3, 3]])

fig = plt.figure()
fig.set_size_inches(10, 10)
xe, ye = plot2dcov(mean=mean, sigma2D=cov, k=[1])
plt.plot(xe,ye)
n_samples = 200
x, y = np.random.multivariate_normal(mean, cov, n_samples).T
M = sample_mean([x,y])
Q = sample_cov([x,y])
xs, ys = plot2dcov(mean=M, sigma2D=Q, k=[1])
plt.plot(xs,ys)
plt.plot(x, y, 'x')
plt.axis('equal')
plt.grid()
plt.legend(["1-sigma isocontour", "1-sigma isocontour from samples", "random samples"])
plt.show()
print("mean = "+str(np.round(M,2)))
print("covariance matrix:\nQ = "+str(np.round(Q,2)))
```



```

mean = [[1.96]
[1.81]]
covariance matrix:
Q = [[1.1  1.38]
[1.38  2.98]]
```

$N_{samples}$	100	200	300	400	500	1000	10000
Mean	1.93	1.93	2.0	2.05	2.0	2.03	2.02
	1.73	1.87	2.06	2.27	2.01	2.1	2.05
Cov	1.16	1.44	1.13	1.42	0.93	1.16	0.88
	1.44	2.98	1.42	3.09	1.16	3.02	1.07

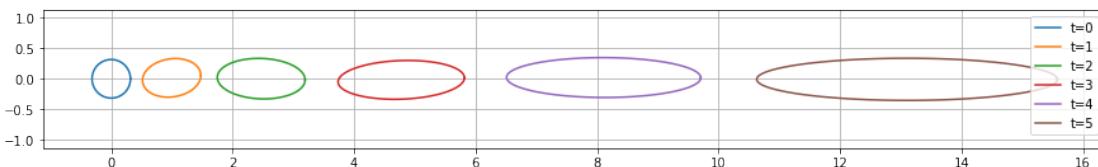
### 0.0.9 Task 3.A

$$\begin{bmatrix} x \\ y \end{bmatrix}_t = \begin{bmatrix} 1 + \delta t & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}_{t-1} + \begin{bmatrix} 2\delta t \\ 0 \end{bmatrix}_t$$

```
In [21]: s0 = np.array([0,0]).T
n_samples = 500
mean0 = s0; cov0 = np.array([[0.1, 0], [0, 0.1]])
dt = 0.5
A = np.array([[1+dt,0],[0,1]]); b = np.array([2*dt, 0]).T
xs, ys = plot2dcov(mean=mean0, sigma2D=cov0, k=[1])
fig = plt.figure()
fig.set_size_inches(15, 2)
plt.plot(xs, ys)
legend_list=[ "t=0"]
for i in range(5):
    s = np.dot(A, s0) + b
    s0 = s
    mean = np.dot(A, mean0) + b
    mean0 = mean
    cov = np.dot(np.dot(A, cov0), A.T )
    cov0 = cov
```

```
x, y = np.random.multivariate_normal(mean, cov, n_samples).T
M = sample_mean([x,y])
Q = sample_cov([x,y])
xs, ys = plot2dcov(mean=M, sigma2D=Q, k=[1])
legend_list.append("t="+str(i+1))
plt.plot(xs, ys)
plt.legend(legend_list)
plt.axis('equal')
plt.grid()
print(mean[0])
```

1.0  
2.5  
4.75  
8.125  
13.1875

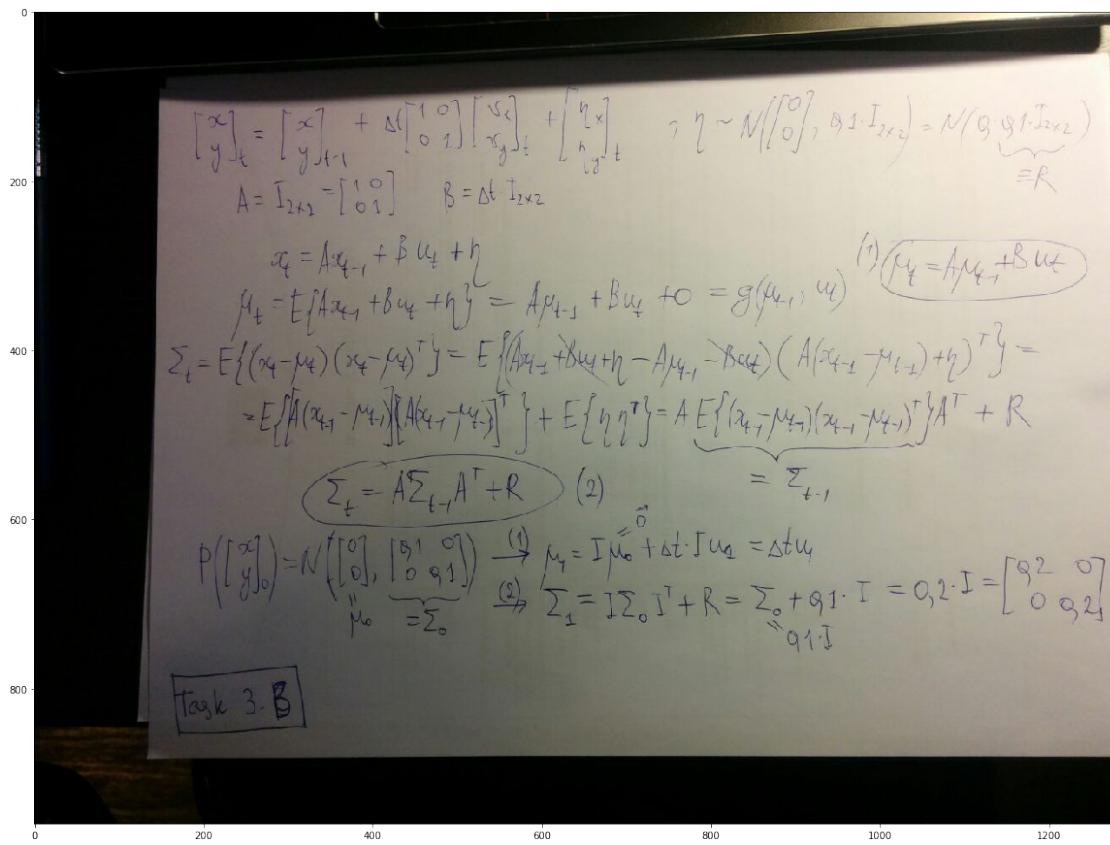


### 0.0.10 Task 3.B

In [8]: `import matplotlib.image as mpimg`

```
In [6]: def show_image(name):
    img=mpimg.imread(name)
    fig = plt.figure(figsize=(20,20))
    imgplot = plt.imshow(img)
    plt.show()
```

In [205]: `show_image('task3B.jpg')`



### 0.0.11 Task 3.C

```
In [85]: u = np.array([np.array([[3],[0]]),
                  np.array([[0],[3]]),
                  np.array([[3],[0]]),
                  np.array([[0],[-3]]),
                  np.array([[3],[0]]))
```

A = np.eye(2)

B = dt \* np.eye(2)

```

R = 0.1 * np.eye(2)

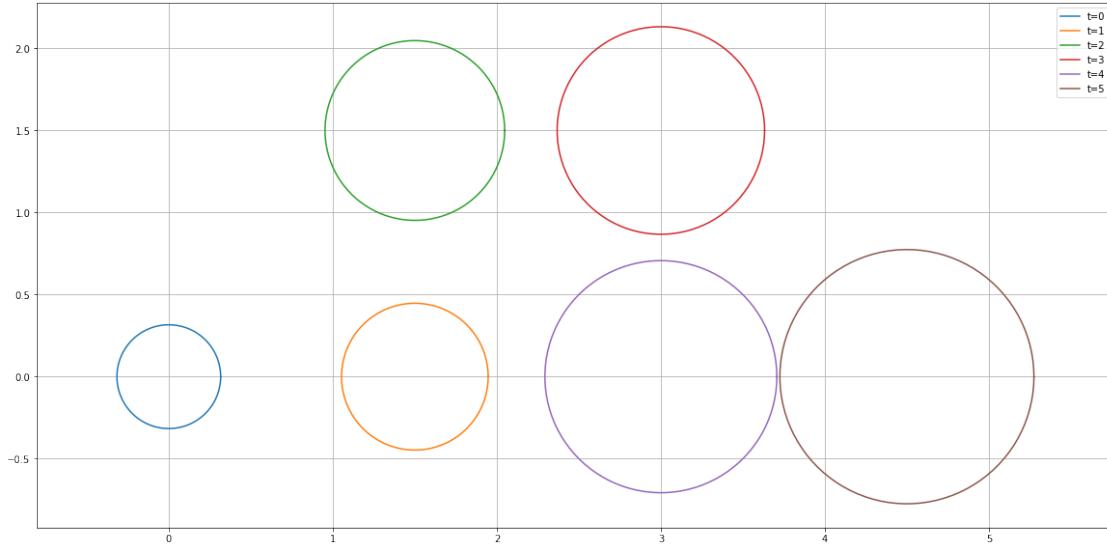
mean0 = np.zeros((2,1))
cov0 = 0.1 * np.eye(2)

plt.figure(figsize=(20,10))
legend_list=[]
xs, ys = plot2dcov(mean0, cov0, k=[1])
legend_list.append("t="+str(0))
plt.plot(xs, ys)
plt.legend(legend_list)
plt.axis('equal')
plt.grid()
for i in range(len(u)):
    mean_u = u[i]
    mean = np.dot( A, mean0 ) + np.dot( B, mean_u )
    mean0 = mean

    cov = cov0 + R
    cov0 = cov

    xs, ys = plot2dcov(mean, cov, k=[1])
    legend_list.append("t="+str(i+1))
    plt.plot(xs, ys)
    plt.legend(legend_list)
    plt.axis('equal')

```



### 0.0.12 Task 3.D

In [206]: `show_image('task3D.jpg')`

Task 3.D

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_t = I_{3 \times 3} \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_{t-1} + \begin{bmatrix} \Delta t \cos \theta & 0 & 0 \\ \Delta t \sin \theta & 0 & 0 \\ 0 & \Delta t & 0 \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix}_t + \vec{\eta}, \vec{\eta} \sim N(\vec{0}, \begin{bmatrix} \sigma^2 & 0 & 0 \\ 0 & \sigma^2 & 0 \\ 0 & 0 & \sigma^2 \end{bmatrix})$$

$$\vec{x}_0 \sim N\left(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.5 \end{bmatrix}\right) \quad u_t = \begin{bmatrix} 3 \\ 2 \end{bmatrix} \quad t = 0.5 \quad \vec{x}_t = \begin{bmatrix} x_{t-1} + \Delta t v \cos \theta_{t-1} \\ y_{t-1} + \Delta t v \sin \theta_{t-1} \\ \theta_{t-1} + \Delta t w_t \end{bmatrix}$$

$$x_t = A x_{t-1} + B(\theta) u_t + \eta = x_{t-1} + B(\theta) u_t + \eta, \eta \sim N(\vec{0}, R)$$

$$I_{3 \times 3} \quad = g(x_{t-1}, u_t, \eta) \approx g(u_{t-1}, \vec{x}_{t-1}) + G_t \begin{pmatrix} (x_t - \mu_{t-1}) \\ (u_t - \mu_{t-1}) \end{pmatrix} \oplus$$

$$\Theta \left( \mu_{t-1} + B(\theta) u_t + \eta + \frac{\partial g(u_{t-1}, u_t)}{\partial x_{t-1}} \right) \begin{pmatrix} \mu_{t-1} \\ (x_t - \mu_{t-1}) \end{pmatrix} = g(u_{t-1}, u_t) + \eta \quad \begin{pmatrix} \mu_{t-1} \\ (x_t - \mu_{t-1}) \end{pmatrix}$$

$$\mu_t = E(x_t) = \mu_{t-1} + E(B(\theta) u_t) + \vec{o} = \mu_{t-1} + E(B(\theta) u_t) u_t = \mu_{t-1} + B(\theta_{t-1}) u_t$$

$$\Sigma_t = G_t \Sigma_{t-1} G_t^T + R, \quad G_t = \frac{\partial g}{\partial x_{t-1}} \begin{pmatrix} 1 & 0 & -\Delta t v \sin \theta_{t-1} \\ 0 & 1 & \Delta t v \cos \theta_{t-1} \\ 0 & 0 & 1 \end{pmatrix}$$

marginalization:  $N\left(\begin{bmatrix} \mu_x \\ \mu_y \\ \mu_\theta \end{bmatrix}_{1 \times 1}, \begin{bmatrix} \sum_{xy} & \sum_{x\theta} & \sum_{\theta\theta} \\ \sum_{y\theta} & \sum_{\theta\theta} & \sum_{\theta\theta} \end{bmatrix}\right)$

In [89]: `def control_matrix(dt, theta):  
 return np.array([[dt*cos(theta), 0], [dt*sin(theta), 0], [0, dt]])`

In [97]: `def jacobian(dt, v, theta):  
 return np.array([[1, 0, -dt*v*sin(theta)],  
 [0, 1, dt*v*cos(theta)],  
 [0, 0, 1]])`

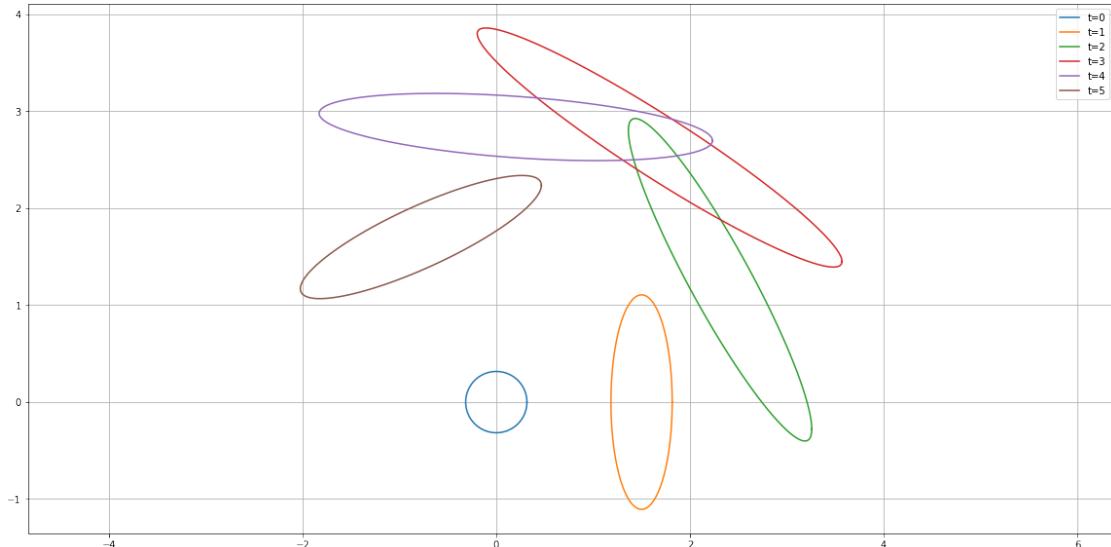
In [108]: `A = np.eye(3)  
theta0 = 0  
B = control_matrix(dt, theta0)  
mean0 = np.zeros((3,1))  
cov0 = np.diag([0.1, 0.1, 0.5])  
R = np.diag([0.2, 0.2, 0.1])  
u = np.array([[3], [2]])  
  
legend_list = []`

```

figure = plt.figure(figsize=(20,10))
x, y = plot2dcov(mean0[:2], cov0[:2,:2], k=[1])
legend_list.append("t="+str(0))
plt.plot(x, y)
plt.legend(legend_list)
plt.axis('equal')
for i in range(5):
    mean = mean0 + np.dot( control_matrix(dt, mean0[2]), u)
    G = jacobian(dt, u[0], mean0[2])
    cov = np.dot( np.dot( G, cov0 ), G.T )
    mean0 = mean
    cov0 = cov

    x, y = plot2dcov(mean[:2], cov[:2,:2], k=[1])
    legend_list.append("t="+str(i+1))
    plt.plot(x, y)
    plt.legend(legend_list)
    plt.axis('equal')
plt.grid()

```



### 0.0.13 Task 3.E

In [207]: `show_image('task3E.jpg')`

Task 3.E

$$x_t = g(x_{t-1}, u_t + \eta_t) \quad \text{where } \eta \sim N(0, \begin{bmatrix} 0 & 0 \\ 0 & 0.1 \end{bmatrix}), \eta = \begin{bmatrix} \eta_1 \\ \eta_2 \end{bmatrix}$$

$$\alpha_t = x_{t-1} + B(\theta_{t-1})(\bar{u}_t + \eta_t), \quad \bar{\eta} \sim N(0, \begin{bmatrix} 0 & 0 \\ 0 & 0.1 \end{bmatrix}), \bar{\eta} = \begin{bmatrix} \bar{\eta}_1 \\ \bar{\eta}_2 \end{bmatrix}$$

$$E\{x_t\} = \mu_t + E\{B(\theta_{t-1})\bar{u}_t\} + E\{B(\theta_{t-1})\bar{\eta}_t\}$$

$$x_t = g(x_{t-1}, u_t + \eta_t) \approx g(u_{t-1}, \bar{u}_t + \eta_t) + \frac{\partial g}{\partial x_{t-1}}(x_{t-1}, \mu_{t-1}) + \frac{\partial g}{\partial u_t}(u_t - \bar{u}_t)$$

$$\Theta \mu_t + B(\theta_{t-1})(u_t + \eta_t) + G_t(x_{t-1} - \mu_{t-1}) + V_t(u_t - \bar{u}_t)$$

$$\Rightarrow \mu_t - \mu_{t-1} = B(\theta_{t-1})\bar{u}_t + G_t(x_{t-1} - \mu_{t-1}) + V_t(u_t - \bar{u}_t)$$

$$\Sigma_t = E\{(B(\theta_{t-1})\bar{u}_t + G_t(x_{t-1} - \mu_{t-1}) + V_t(u_t - \bar{u}_t))(B(\theta_{t-1})\bar{u}_t + G_t(x_{t-1} - \mu_{t-1}) + V_t(u_t - \bar{u}_t))^T\}$$

$$= \begin{cases} \bar{u}_t, \bar{u}_t - \text{uncorrel.} \\ \bar{u}_t, u_t - \text{uncorrel.} \\ u_t, u_t - \text{uncorrel.} \end{cases} = B(\theta_{t-1})R(B(\theta_{t-1}))^T + (G_t \cdot \Sigma_{t-1} G_t^T + V_t \cdot R \cdot V_t^T) = \Sigma_t \quad (2)$$

$$V_t = \frac{\partial}{\partial u_t} \begin{bmatrix} x_{t-1} + \Delta t V_t \cos \theta_{t-1} \\ y_{t-1} + \Delta t V_t \sin \theta_{t-1} \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} \Delta t \cos \theta_{t-1} & 0 \\ \Delta t \sin \theta_{t-1} & 0 \\ 0 & \Delta t \\ 0 & 0 \end{bmatrix}$$

```
In [111]: def jacobianV(dt, theta):
```

```
    return np.array([[dt*cos(theta), 0],
                    [dt*sin(theta), 0],
                    [0, dt]])
```

```
In [115]: A = np.eye(3)
```

```
theta0 = 0
B = control_matrix(dt, theta0)
mean0 = np.zeros((3,1))
cov0 = np.diag([0.1, 0.1, 0.5])
R = np.diag([2, 0.1])
u = np.array([[3], [2]])
```

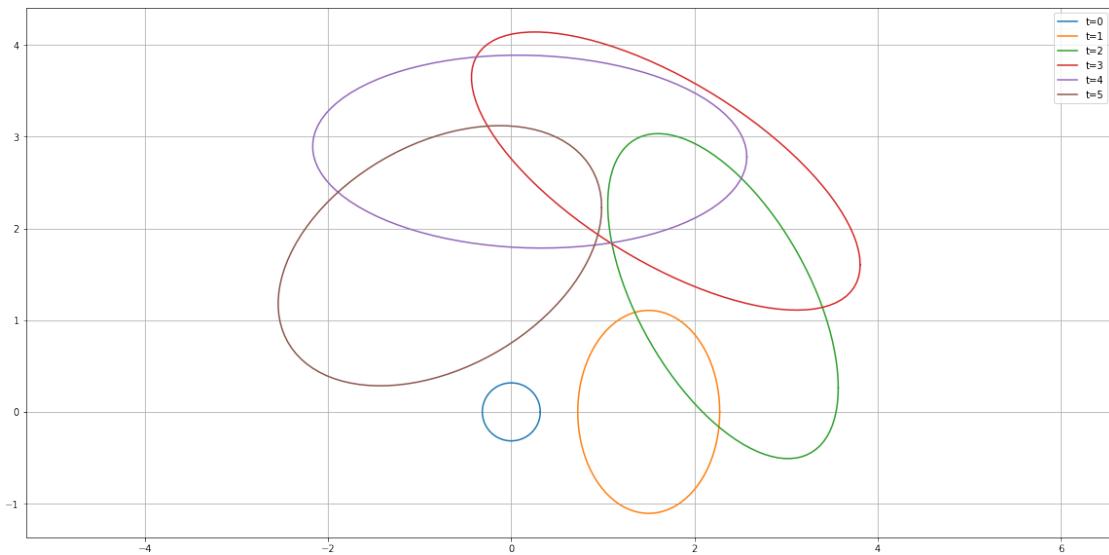
```
legend_list = []
figure = plt.figure(figsize=(20,10))
x, y = plot2dcov(mean0[:2], cov0[2:2], k=[1])
legend_list.append("t=" + str(0))
plt.plot(x, y)
plt.legend(legend_list)
plt.axis('equal')
for i in range(5):
    mean = mean0 + np.dot(control_matrix(dt, mean0[2]), u)
    G = jacobian(dt, u[0], mean0[2])
    V = jacobianV(dt, mean0[2])
    cov = np.dot(np.dot(G, cov0), G.T) + np.dot(np.dot(V, R), V.T)
    mean0 = mean
```

```

cov0 = cov

x, y = plot2dcov(mean[:2], cov[:2,:2], k=[1])
legend_list.append("t="+str(i+1))
plt.plot(x, y)
plt.legend(legend_list)
plt.axis('equal')
plt.grid()

```



#### 0.0.14 Task 4.A

```
In [3]: data = np.load('t4')
```

```

In [4]: dt = float(data['dt'])
N = int(data['N'])
P_0 = float(data['P_0'])
Q = np.float(data['Q'])
R = np.float(data['R_action'])
t = np.array(data['t'])
u = np.array(data['u'])
x_0 = float(data['x_0'])
x_real = np.array(data['x_real'])
z = np.array(data['z'])

```

```
In [9]: show_image('task4A.jpg')
```

Task 4.A

$$x_t = x_{t-1} + \Delta t \cdot (u_t + \varepsilon_t), \quad \varepsilon_t \sim N(0, 4)$$

$$\tilde{x}_t \sim N(0, 0) \quad \Delta t = 0, 1$$

$$\mu_{\tilde{x}_t} \quad \Sigma_{\tilde{x}_t}$$

$$\tilde{\mu}_t = E\{x_t\} = \mu_{t-1} + \Delta t (\tilde{\mu}_{t-1} + 0) = \mu_{t-1} + \Delta t \cdot \tilde{\mu}_t = \mu_t \quad (1)$$

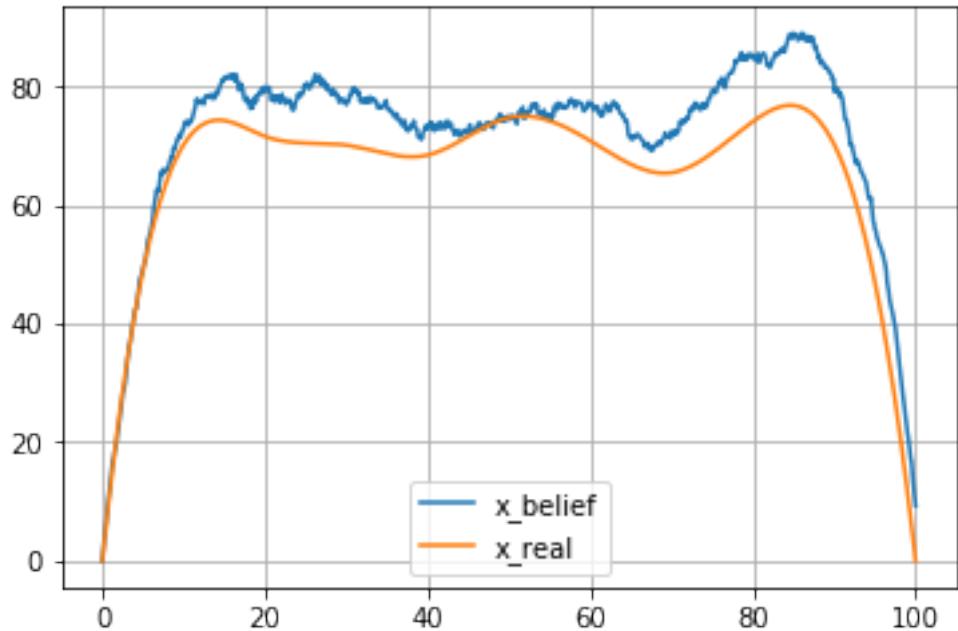
$$\begin{aligned} \tilde{\Sigma}_t &= E[(x_t - \mu_t)(x_t - \mu_t)^T] = E[(x_{t-1} + \Delta t(u_t + \varepsilon_t) - \mu_{t-1} - \Delta t \mu_t)(x_{t-1} + \Delta t(u_t + \varepsilon_t) - \mu_{t-1} - \Delta t \mu_t)^T] = \\ &= \left( \sum_{t-1} + \Delta t^2 \cdot R = \tilde{\Sigma}_t \right) \quad (2) \end{aligned}$$

uncertainty  $\rightarrow$  increasing over time

```
In [10]: me = 0; cove = 4
x = [x_0]
x_prev = x[0]
for i in range(N-1):
    e = np.random.normal(me, cove)
    x.append(x_prev + dt*(u[i] + e))
    x_prev = x[-1]

x = np.array(x)
```

```
In [11]: plt.plot(t, x)
plt.plot(t, x_real)
plt.grid()
plt.legend(['x_belief', 'x_real'])
plt.show()
```

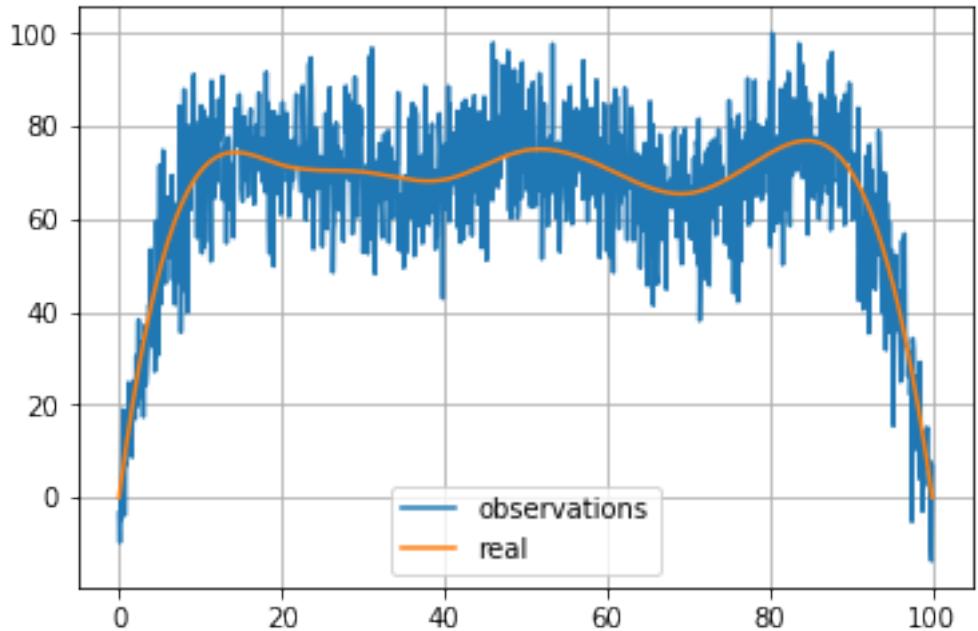


There is a shift between the 2 graphs: real state and belief based only on actions information without measurements.

### 0.0.15 Task 4.B

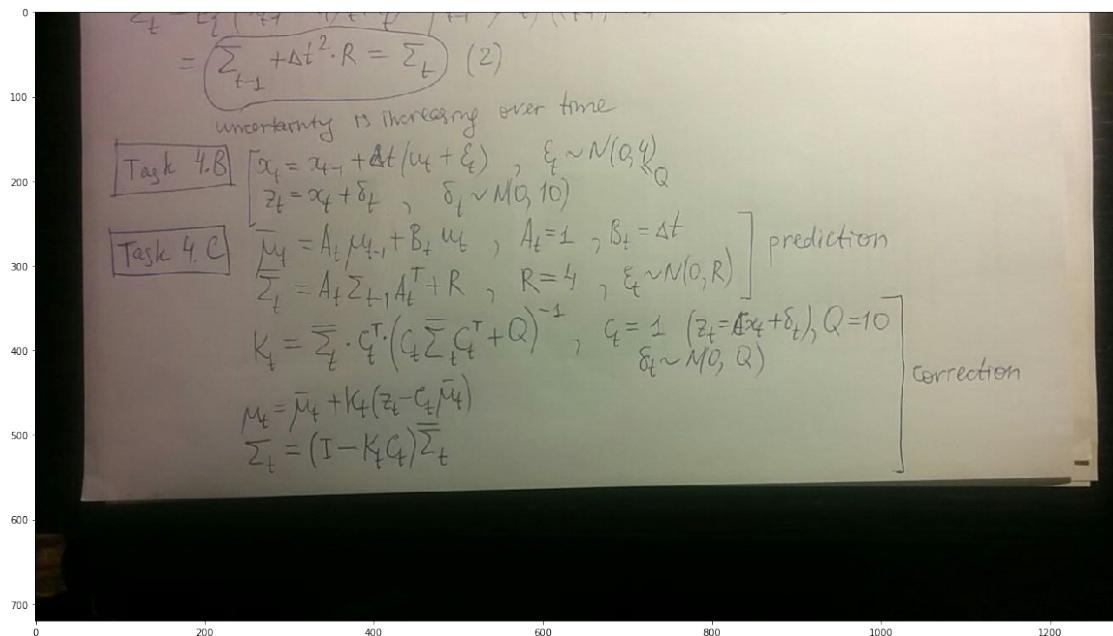
```
In [33]: md = 0; covd = Q
z = np.zeros_like(x)
for i in range(N):
    d = np.random.normal(md, covd)
    z[i] = x_real[i] + d

plt.plot(t,z)
plt.plot(t,x_real)
plt.grid()
plt.legend(['observations','real'])
plt.show()
```



### 0.0.16 Task 4.C

In [213]: `show_image('task4C.jpg')`



```
In [34]: A = np.array(1)
        B = np.array(dt)
        C = np.array(1)
```

### 0.0.17 Task 4.D

```
In [1]: def kalman1D(R,Q, u,z, mu0=10,cov0=100):
    N = len(z)
    mu = mu0; cov = cov0
    x_pred = np.array([mu0])
    x_cor = np.array([mu0])
    sigma_pred = np.array([mu0])
    sigma_cor = np.array([mu0])
    for i in range(N-1):
        # prediction:
        mu_pred = mu + 0.1*u[i]
        cov_pred = cov + R

        #correction:
        K = cov_pred / (cov_pred + Q)
        mu = mu_pred + K * (z[i]-mu_pred)
        cov = (1-K) * cov_pred

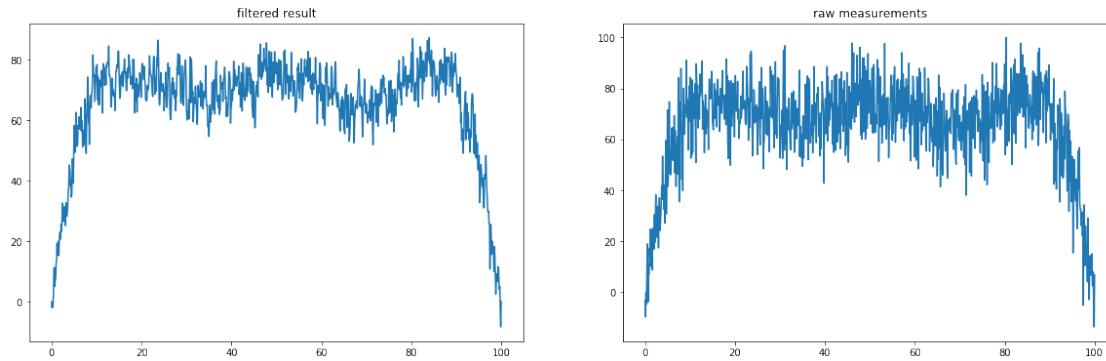
        x_pred = np.append(x_pred, mu_pred)
        x_cor = np.append(x_cor, mu)
        sigma_pred = np.append(sigma_pred, cov_pred)
        sigma_cor = np.append(sigma_cor, cov)

    return x_cor, sigma_cor, x_pred, sigma_pred
```

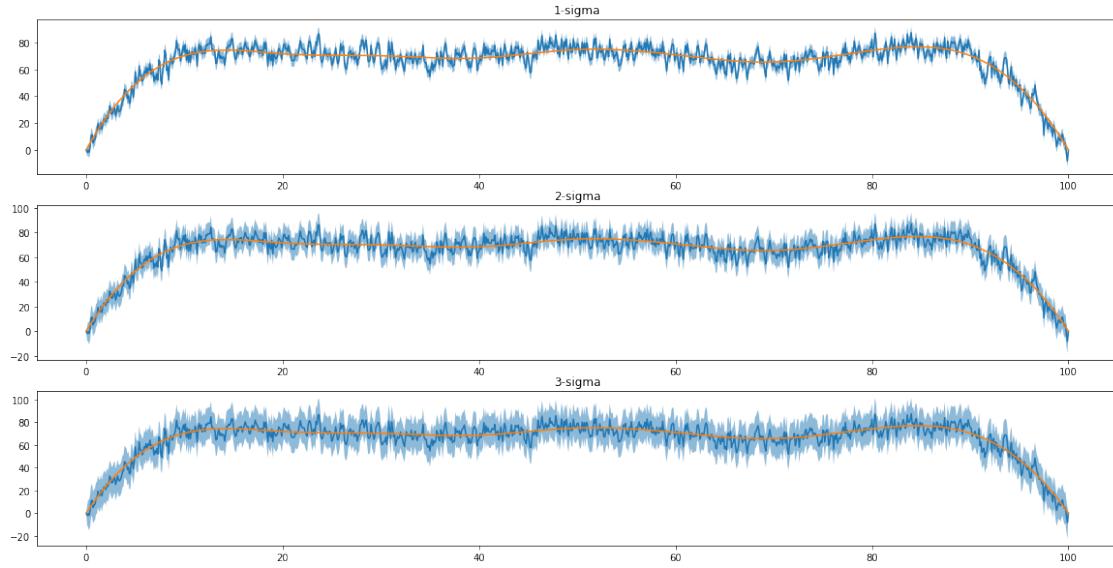
```
In [35]: def kalman(A,B,C,R,Q, u,z, mu0,cov0):
    N = len(z)
    mu_pred = np.zeros_like(z)
    cov_pred = np.zeros_like(z)
    mu = np.zeros_like(z)
    cov = np.zeros_like(z)
    mu[0] = mu0; cov[0] = cov0
    for i in range(1,N-1):
        # prediction:
        mu_pred[i] = np.dot(A, mu[i-1]) + np.dot(B, u[i])
        cov_pred[i] = np.dot( np.dot(A,cov[i-1]), A.T ) + R
        # correction:
        if len([mu[i]])<2:
            K = np.dot( np.dot(cov_pred[i], C.T), 1.0/(np.dot( np.dot(C,cov_pred[i]), C.T)))
        else:
            K = np.dot( np.dot(cov_pred[i], C.T), np.linalg.inv(np.dot( np.dot(C,cov_pred[i]), C.T)))
        mu[i] = mu_pred[i] + np.dot( K, (z[i]-np.dot(C,mu_pred[i])) )
        cov[i] = np.dot( (np.eye(len([mu[i]]))-np.dot(K,C)), cov_pred[i] )
```

```
    return mu,cov, mu_pred,cov_pred
```

```
In [36]: x_cor, cov_cor, x_pred, cov_pred=kalman(A,B,C,R,Q, u,z, mu0=0,cov0=0)
fig = plt.figure(figsize=(20,6))
fig.add_subplot(121)
plt.plot(t,x_cor)
plt.title('filtered result')
fig.add_subplot(122)
plt.plot(t,z)
plt.title('raw measurements')
plt.show()
```

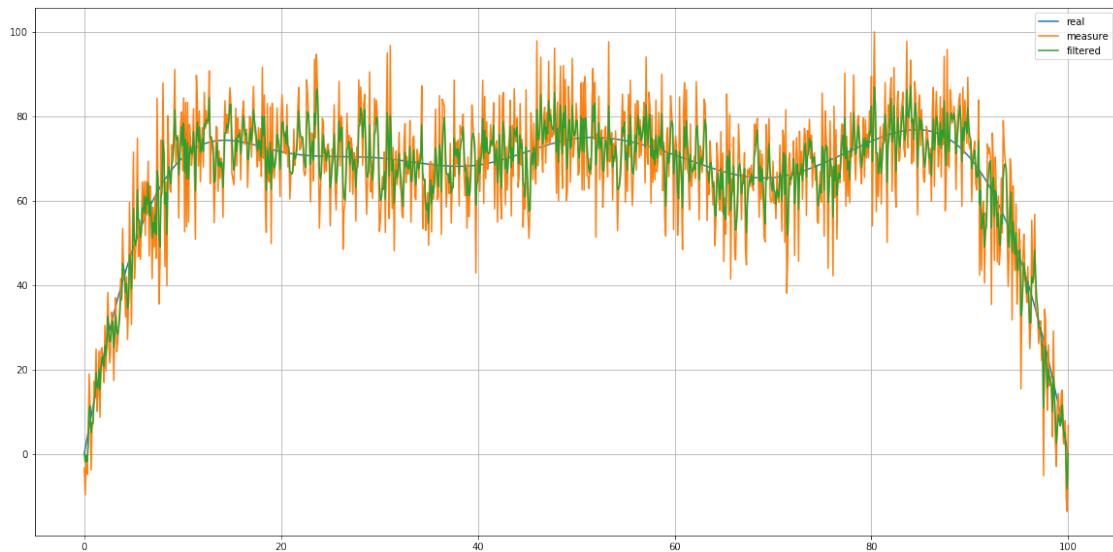


```
In [37]: from ciplot import *
fig = plt.figure(figsize=(20,10))
fig.add_subplot(311)
ciplot(t, mu=x_cor, minus_sigma=x_cor-cov_cor, plus_sigma=x_cor+cov_cor, x_real=x_real,
plt.title('1-sigma')
fig.add_subplot(312)
ciplot(t, mu=x_cor, minus_sigma=x_cor-2*cov_cor, plus_sigma=x_cor+2*cov_cor, x_real=x_r
plt.title('2-sigma')
fig.add_subplot(313)
ciplot(t, mu=x_cor, minus_sigma=x_cor-3*cov_cor, plus_sigma=x_cor+3*cov_cor, x_real=x_r
plt.title('3-sigma')
plt.show()
```



3-sigma interval contains real data completely, while more narrow intervals (1- and 2-sigma) include approximately 68% and 96% of real data.

```
In [41]: fog = plt.figure(figsize=(20,10))
plt.plot(t,x_real)
plt.plot(t,z)
plt.plot(t,x_cor)
plt.legend(['real','measure','filtered'])
plt.grid()
plt.show()
```



In comparison to part 4A there is no shift now between filtered data and real state. That is because measurements are now included in the correction step of the Kalman filer.