nnaglov_ps1

February 9, 2023

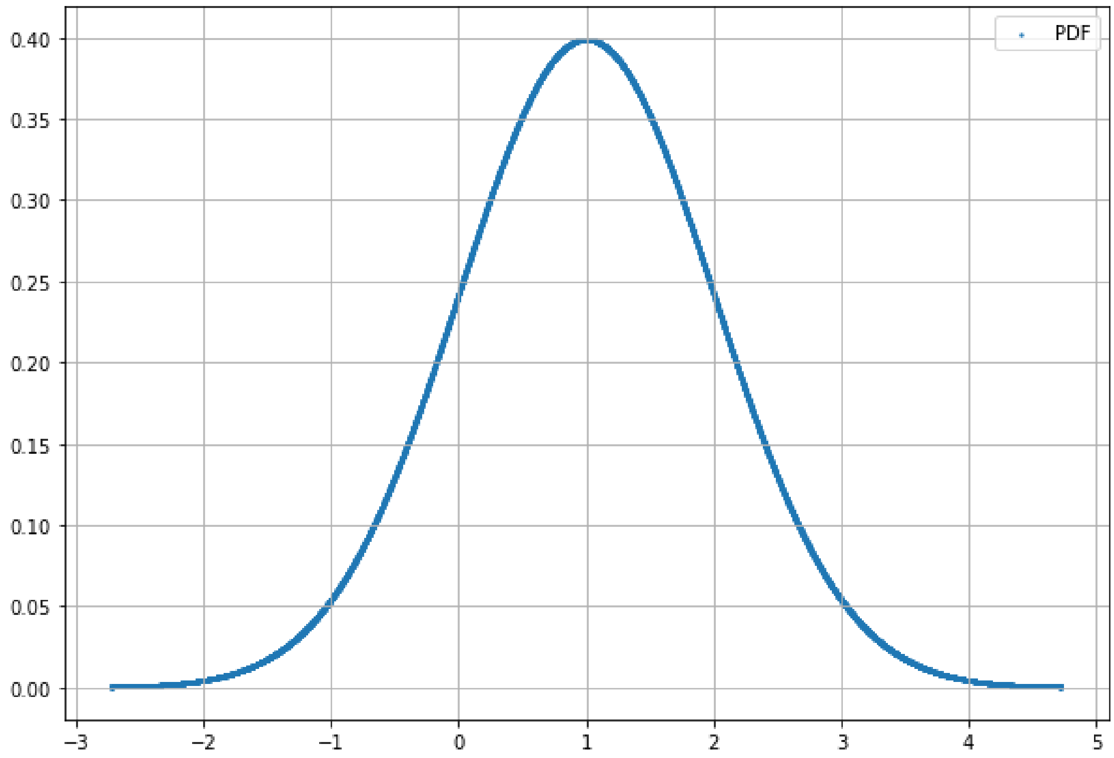
[ ]:
```python
# Import libraries
import numpy as np
import matplotlib.pyplot as plt
from typing import List
from scipy.stats import norm
from scipy.linalg import cholesky
```

0.1 Task 1: Probability

A. Plot the probability density function $p(x)$ of a one dimensional Gaussian distribution $\mathcal{N}(x; 1; 1)$

[ ]:
```python
n = 10000
mu = 1
sigma = np.sqrt(1)
start = norm.ppf(0.0001, loc = mu, scale = sigma)
end = norm.ppf(0.9999, loc = mu, scale = sigma)
x = np.linspace(start, end, n)
p_x = norm.pdf(x, loc = mu, scale = sigma)
```

[ ]:
```python
plt.figure(figsize=(10,7))
plt.scatter(x, p_x, label='PDF', s=2)
plt.legend()
plt.grid()
```

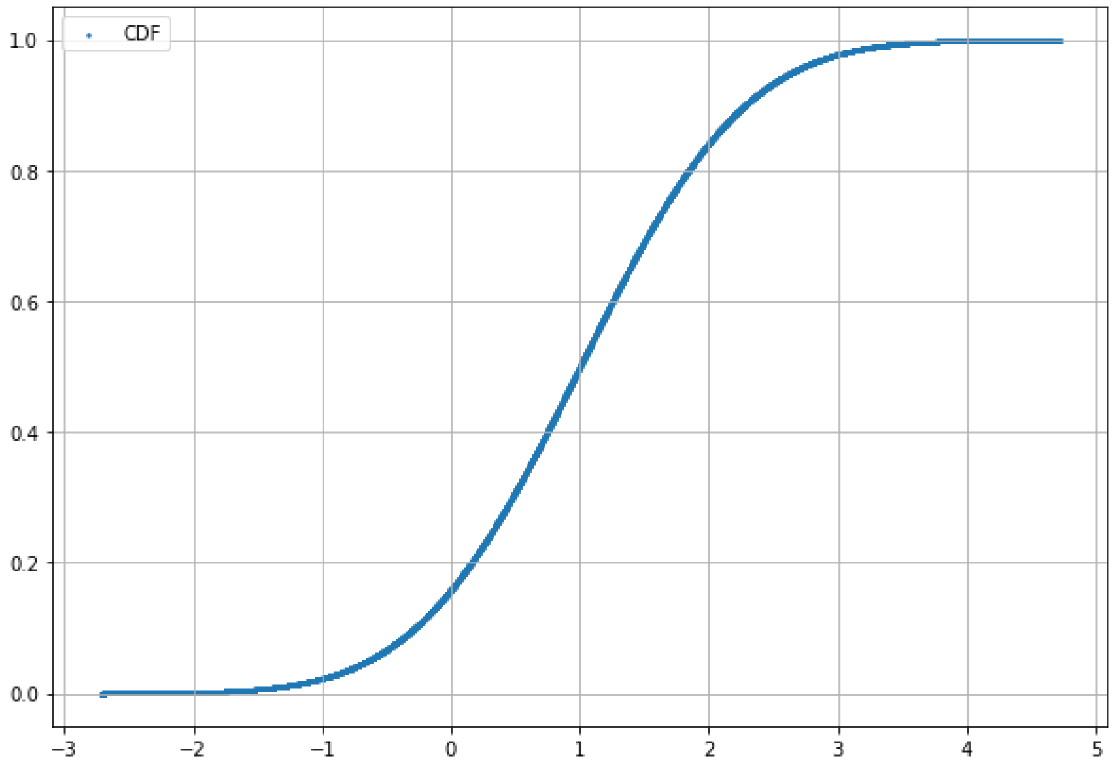


B. Calculate the probability mass that the random variable X is less than 0, that is, $Pr\{X <= 0\} = \int_{-\infty}^{0} p(x)dx$

[ ]:
```
cdf = norm.cdf(x, loc=mu, scale=sigma)
p_mass = norm.cdf(0, loc=mu, scale=sigma)
print(f"P(X <= 0) = {p_mass:.4f}")
```

```
P(X <= 0) = 0.1587
```

[ ]:
```
plt.figure(figsize=(10,7))
plt.scatter(x, cdf, label='CDF', s=2)
plt.legend()
plt.grid()
```

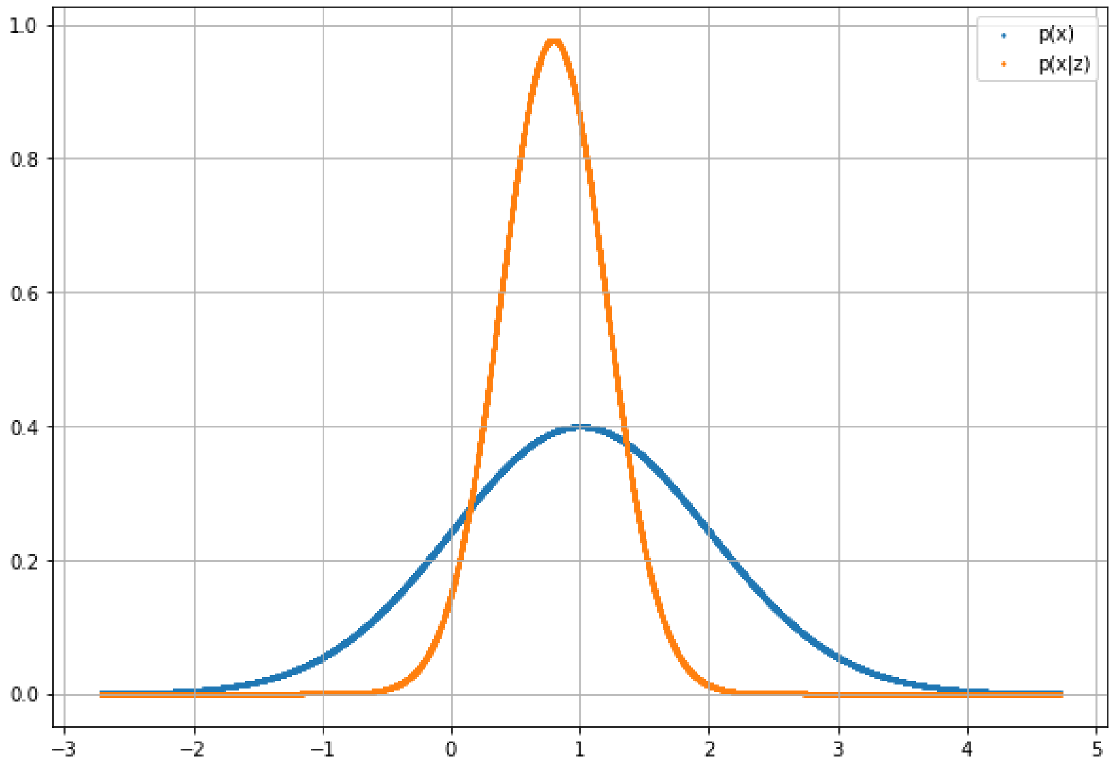


C. Consider the new observation variable z, it gives information about the variable x by the likelihood function $p(z|x) = \mathcal{N}(z; x; \sigma^2)$, with variance $\sigma^2 = 0.2$. Apply the Bayes' theorem to derive the posterior distribution, $p(x|z)$, given an observation $z = 0.75$ and plot it. For a better comparison, plot the prior distribution, $p(x)$, too.

[ ]:
```
z = 0.75
sigma_z = np.sqrt(0.2)
p_z_x = norm.pdf(z, loc=x, scale=sigma_z)
```

[ ]:
```
p_z = np.trapz(p_z_x * p_x, x = x)
p_x_z = p_z_x * p_x / p_z
```

[ ]:
```
plt.figure(figsize=(10,7))
plt.scatter(x, p_x, label='p(x)', s=2)
plt.scatter(x, p_x_z, label='p(x|z)', s=2)
plt.legend()
plt.grid()
```



0.2 Task 2: Multivariate Gaussian

A. Write the function *plot2dcov* which plots the 2d contour given three core parameters: mean, covariance, and the iso-contour value k. You may add any other parameter such as color, number of points, etc. Then, use *plot2dcov* to draw the iso-contours corresponding to 1,2,3-sigma of the following Gaussian distributions:

$$\mathcal{N}\left(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}1 & 0\\0 & 2\end{bmatrix}\right)$$

$$\mathcal{N}\left(\begin{bmatrix}5\\0\end{bmatrix}, \begin{bmatrix}3 & -0.4\\-0.4 & 2\end{bmatrix}\right)$$

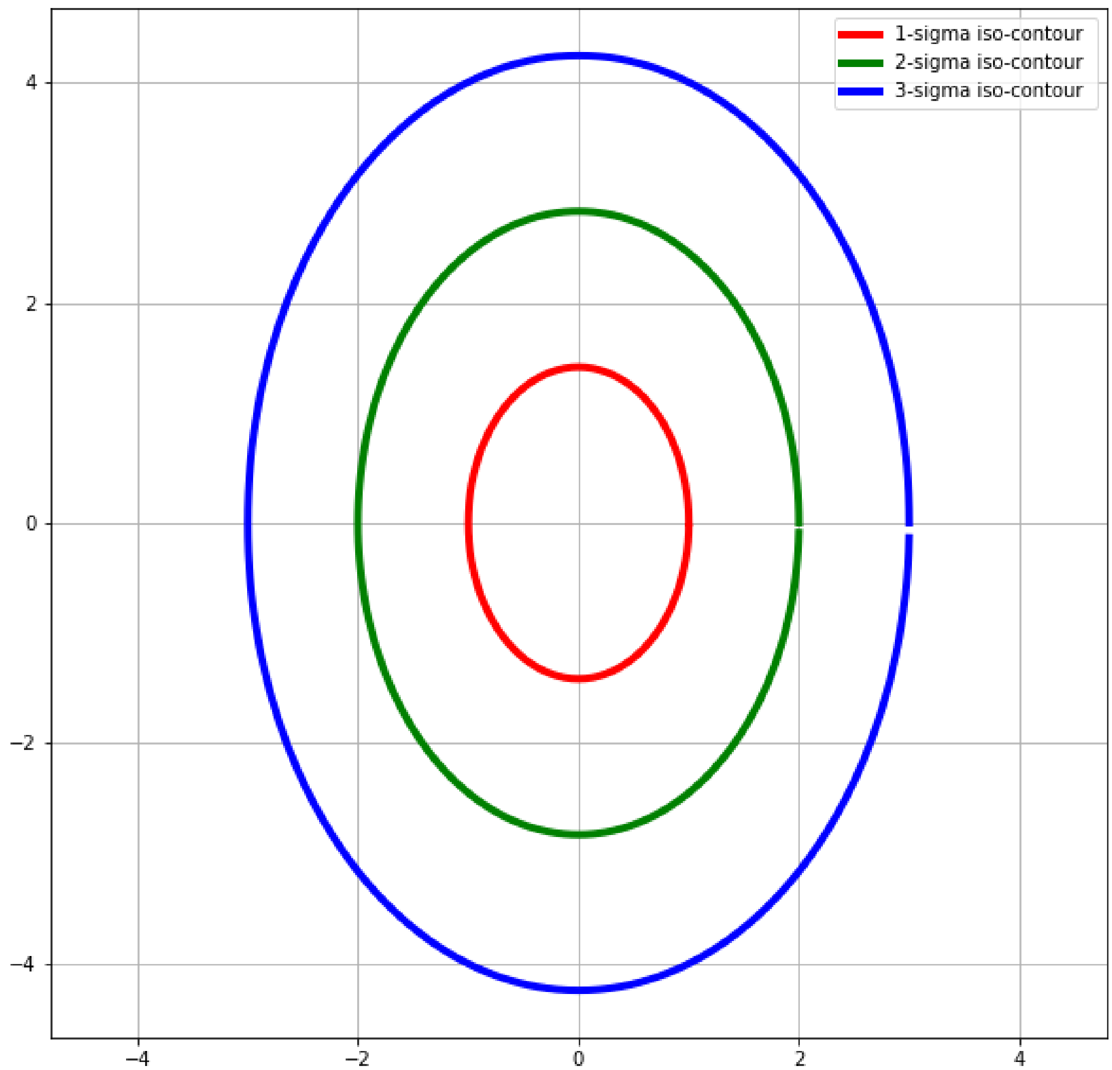
$$\mathcal{N}\left(\begin{bmatrix}2\\2\end{bmatrix}, \begin{bmatrix}9.1 & 6\\6 & 4\end{bmatrix}\right)$$

Use the *set_aspect('equal')* command and comment on them.

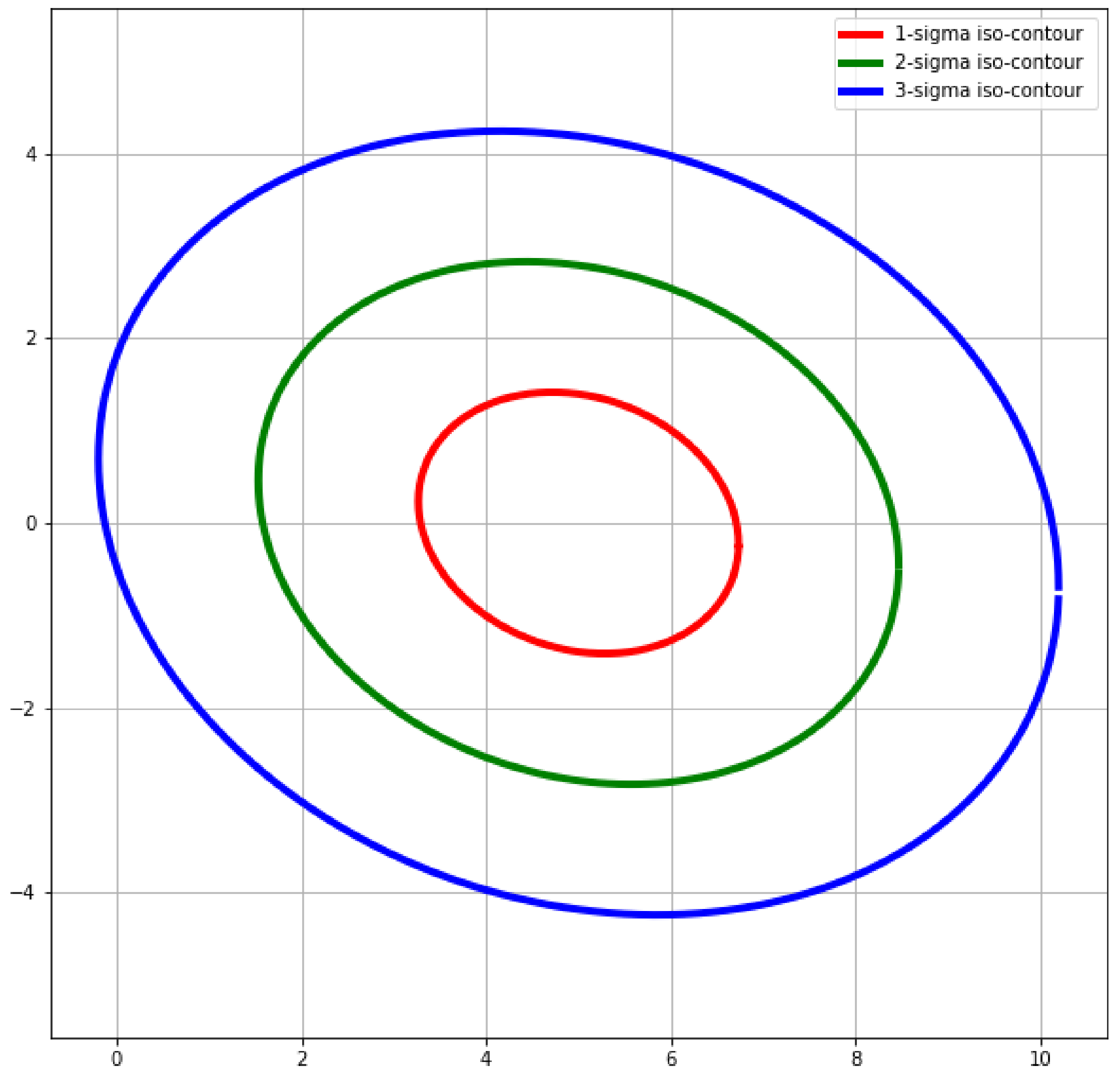
[ ]:
```
def plot2dcov(mean: np.ndarray, cov: np.ndarray, k: List, colors: List = ["r",
    "g", "b"], new_figure: bool = False, extra_label: str = "", legend: bool =
    True) -> None:
    L = cholesky(cov, lower=True)
    mean = mean.reshape((mean.size, 1))
```

```
    size = 200
    step = 2 * np.pi / size
    points = np.zeros((2, size))
    if new_figure:
        plt.figure(figsize=(10, 10))
    for j in range(k):
        for i in range(size):
            point = np.array([
                [(j + 1) * np.cos(i * step)],
                [(j + 1) * np.sin(i * step)],
            ])
            points[:, i] = (L[:2, :2] @ point + mean[:2]).flatten()
        plt.plot(points[0, :], points[1, :], color=colors[j], linewidth=4,␣
 ↪label=f"{j + 1}-sigma iso-contour {extra_label}")
    plt.axis('equal')
    if legend:
        plt.legend()
    plt.grid()
    return
```

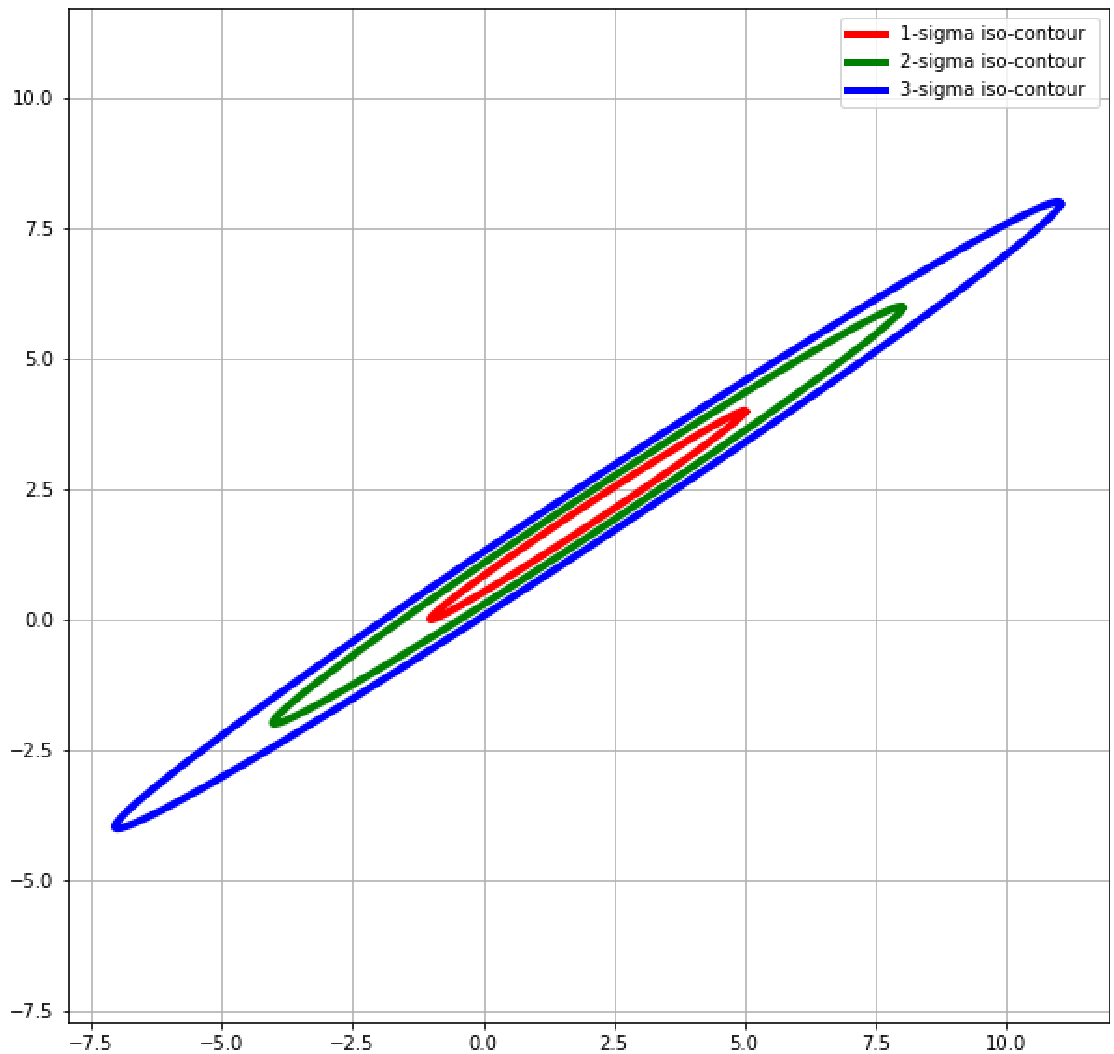
[ ]:
```
mu = np.array([[0], [0]])
sigma = np.array([
    [1, 0],
    [0, 2]
])
plot2dcov(mu, sigma, 3, new_figure=True)
```



[ ]:
```
mu = np.array([[5], [0]])
sigma = np.array([
    [3, -0.4],
    [-0.4, 2]
])
plot2dcov(mu, sigma, 3, new_figure=True)
```



[ ]:
```
mu = np.array([[2], [2]])
sigma = np.array([
    [9.1, 6],
    [6, 4]
])
plot2dcov(mu, sigma, 3, new_figure=True)
```



B. Write the equation of sample mean and sample covariance of a set of points $\{x_i\}$, in vector form as was shown during the lecture.

Sample mean: $\overline{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$

Sample covariance: $\overline{\Sigma}_x = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})(x_i - \overline{x})^T$

C. Draw random samples from a multivariate normal distribution. You can use the python function that draws samples from the univariate normal distribution $\mathcal{N}(0, 1)$.

In particular, draw and plot 200 samples from:

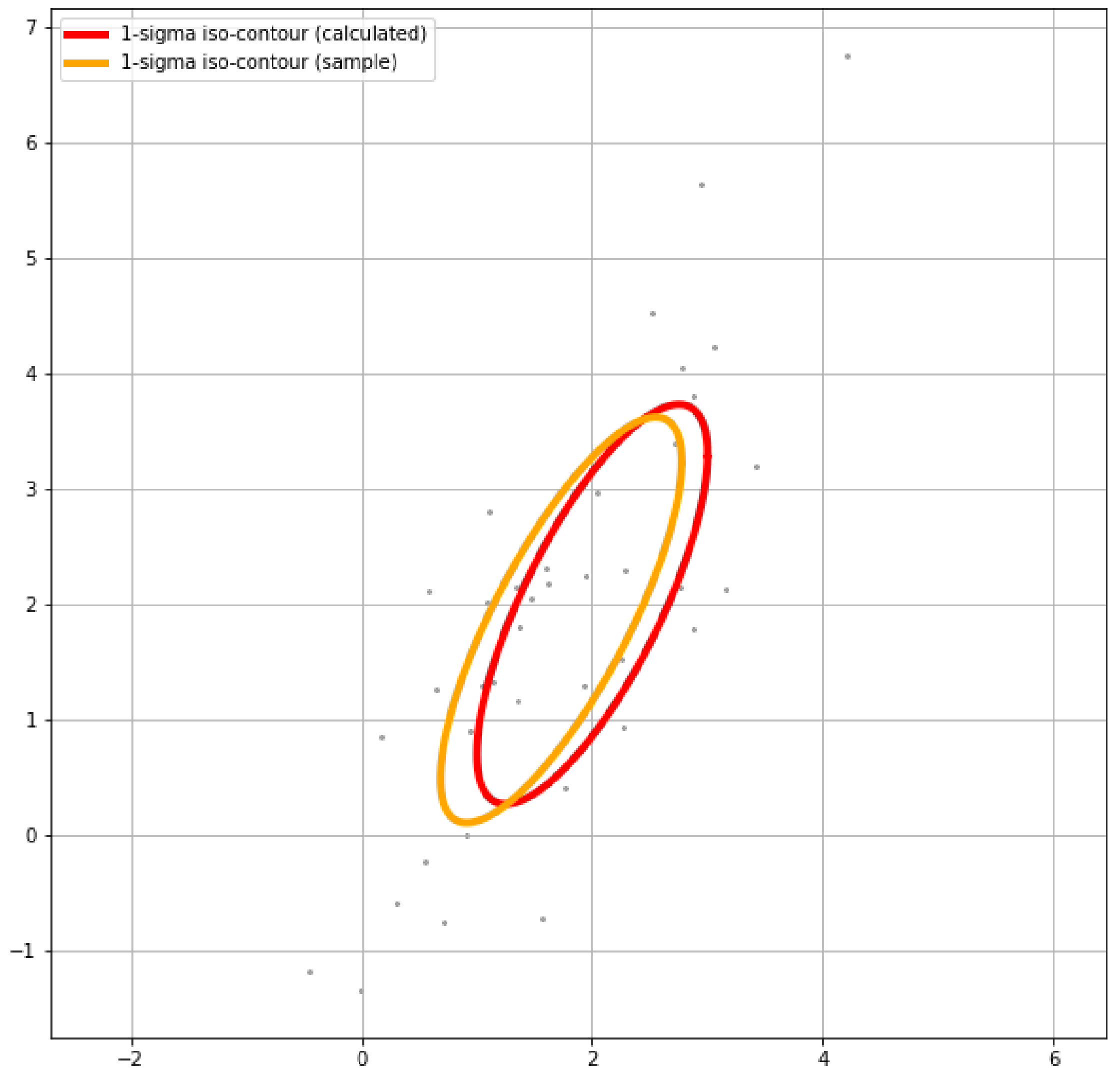
$$\mathcal{N}\left(\begin{bmatrix}2\\2\end{bmatrix}, \begin{bmatrix}1 & 1.3\\1.3 & 3\end{bmatrix}\right)$$

Also plot their corresponding 1-sigma iso-contour. Then calculate the sample mean and covariance in vector form and plot again the 1-sigma iso-contour for the estimated Gaussian parameters. Run the experiment multiple times and try different number of samples. Comment on the results.

[ ]:
```
def plot2dcov_with_cloud(cloud: np.ndarray, mean: np.ndarray, cov: np.ndarray,␣
↪k: List, iso_colors: List = ["red", "green", "darkblue"], cloud_color: str =␣
↪"black", new_figure: bool = True, legend: bool = True, sample_iso: bool =␣
↪True) -> None:
    plot2dcov(mean, cov, k, colors=iso_colors, new_figure=new_figure,␣
↪extra_label="(calculated)", legend=legend)
    if sample_iso:
        cloud_mu = cloud.mean(axis=0)
        cloud_sigma = np.cov(cloud.T)
        with np.printoptions(precision=4, suppress=True):
            print(f"Sample mean:\n {cloud_mu}")
            print(f"Sample covariance:\n {cloud_sigma}")
        plot2dcov(cloud_mu, cloud_sigma, 1, colors=["orange", "lime", "blue"],␣
↪extra_label="(sample)", legend=legend)
    plt.scatter(cloud[:,0], cloud[:,1], color=cloud_color, label='Sample␣
↪cloud', s=4, alpha=0.3)
    plt.axis('equal')
    plt.grid(visible=True)
    return
```

[ ]:
```
k = 40
calc_mu = np.array([2, 2])
calc_sigma = np.array([
    [1, 1.3],
    [1.3, 3]
])

cloud = np.random.multivariate_normal(calc_mu, calc_sigma, k)
plot2dcov_with_cloud(cloud, calc_mu, calc_sigma, 1)
```

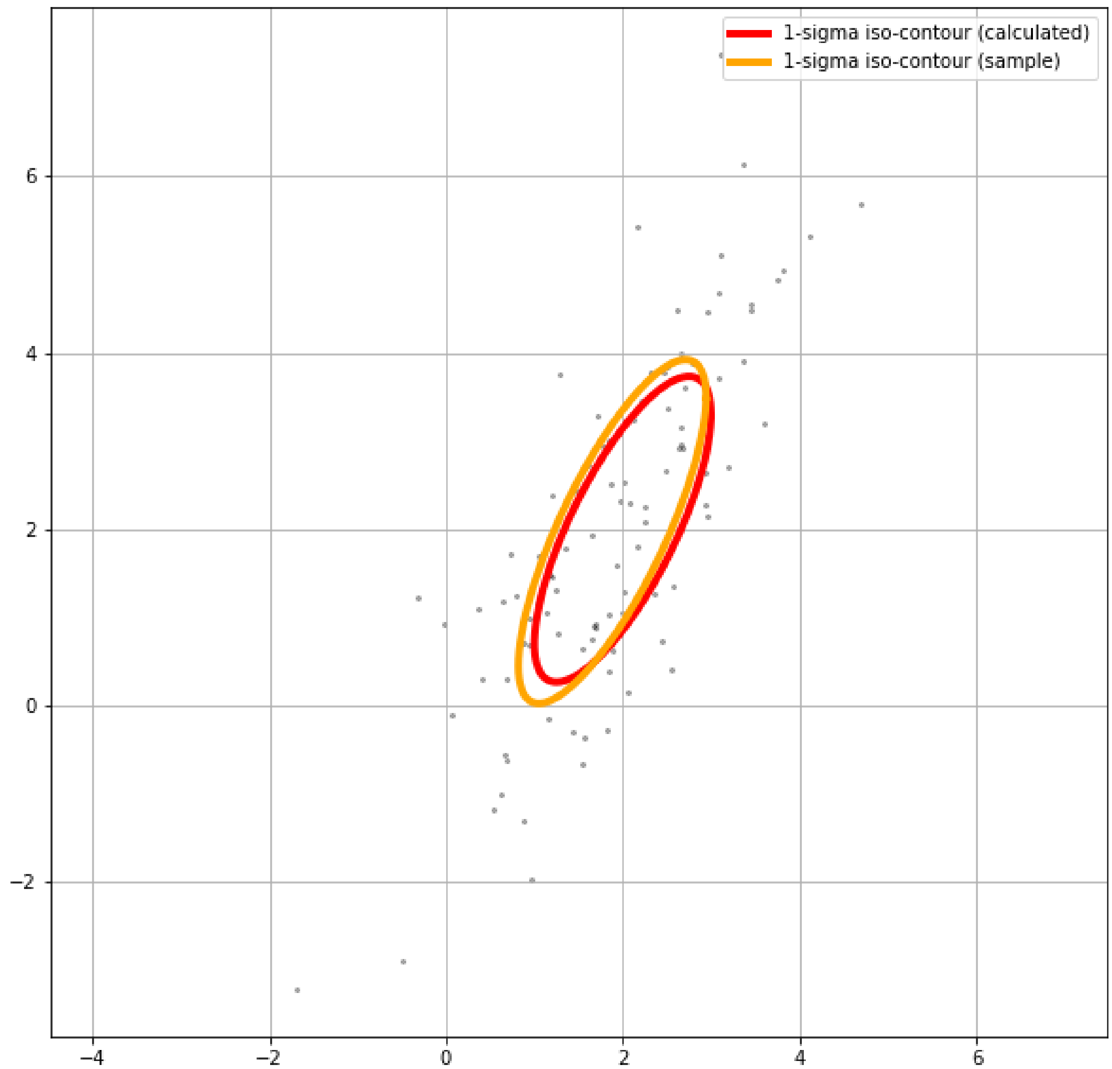
```
Sample mean:
 [1.7305 1.863 ]
Sample covariance:
 [[1.0953 1.4394]
 [1.4394 3.0892]]
```



[ ]:
```
k = 100
calc_mu = np.array([2, 2])
calc_sigma = np.array([
    [1, 1.3],
    [1.3, 3]
])

cloud = np.random.multivariate_normal(calc_mu, calc_sigma, k)
plot2dcov_with_cloud(cloud, calc_mu, calc_sigma, 1)
```

```
Sample mean:
 [1.8806 1.9749]
Sample covariance:
 [[1.1284 1.6181]
```

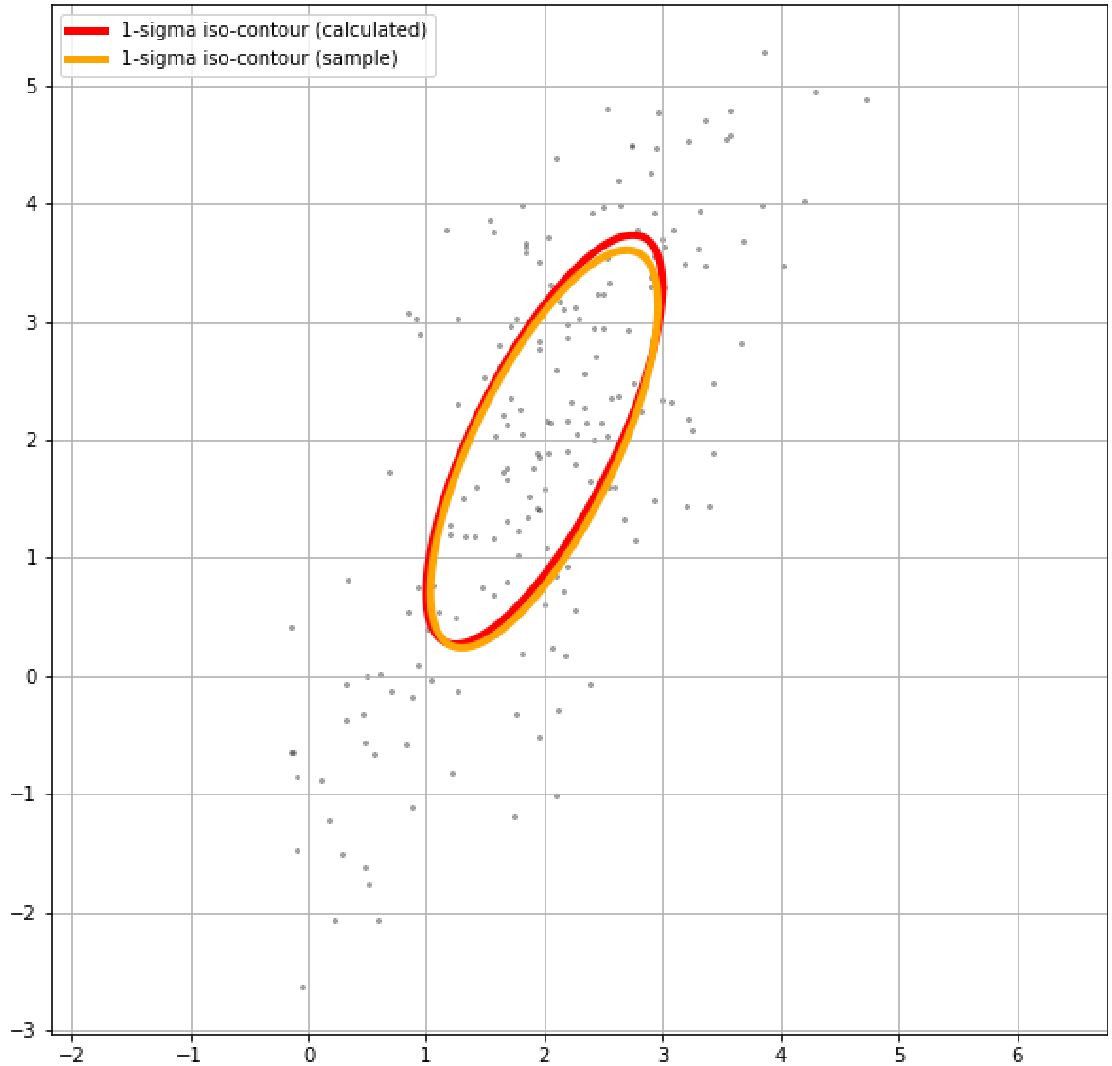
[1.6181 3.7935]]




```
[ ]: k = 200
     calc_mu = np.array([2, 2])
     calc_sigma = np.array([
         [1, 1.3],
         [1.3, 3]
     ])

     cloud = np.random.multivariate_normal(calc_mu, calc_sigma, k)
     plot2dcov_with_cloud(cloud, calc_mu, calc_sigma, 1)
```

```
Sample mean:
 [1.9998 1.9207]
Sample covariance:
```

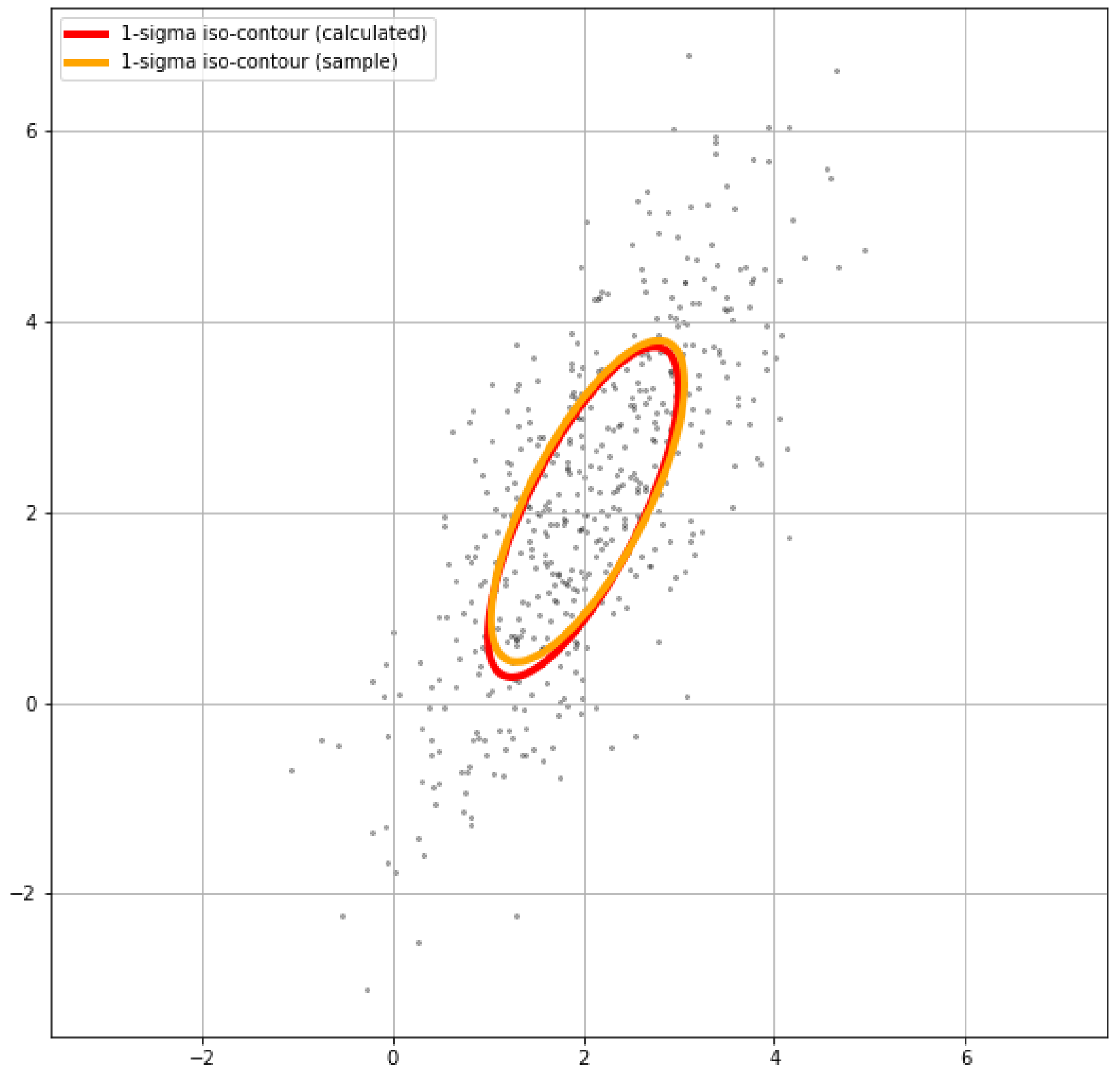
```
[[0.9299 1.1752]
 [1.1752 2.8336]]
```



[ ]:
```python
k = 500
calc_mu = np.array([2, 2])
calc_sigma = np.array([
    [1, 1.3],
    [1.3, 3]
])

cloud = np.random.multivariate_normal(calc_mu, calc_sigma, k)
plot2dcov_with_cloud(cloud, calc_mu, calc_sigma, 1)
```

```
Sample mean:
 [2.0468 2.1151]
```

```
Sample covariance:
 [[1.0256 1.255 ]
 [1.255  2.8402]]
```



Result is the following: the bigger sample size, the closer sample iso-contour is to the calculated one.

0.3 Task 3: Covariance Propagation

For this task, we will model an omni-directional robotic platform, i.e., a holonomic platform moving as a free point without restrictions. The propagation model is the following:

$$\begin{bmatrix} x \\ y \end{bmatrix}_t = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}_{t-1} + \begin{bmatrix} \Delta t & 0 \\ 0 & \Delta t \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix}_t + \begin{bmatrix} \eta_x \\ \eta_y \end{bmatrix}_t$$

where the controls $u = \begin{bmatrix} v_x & v_y \end{bmatrix}^T$ are the velocities which are commanded to the robot. Unfortunately, there exists some uncertainty on command execution:

$$\begin{bmatrix} \eta_x \\ \eta_y \end{bmatrix}_t \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix} \right)$$

We will consider a time step of $\Delta t = 0.5$.

A. Write the equations corresponding to the mean and covariance after a single propagation of the holonomic platform.

Mean after single propagation:

$$\begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix}_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^T \begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix}_0 + \begin{bmatrix} \Delta t & 0 \\ 0 & \Delta t \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix}_1 + \begin{bmatrix} \mu_{\eta_x} \\ \mu_{\eta_y} \end{bmatrix}_1 = \begin{bmatrix} x \\ y \end{bmatrix}_0 + \begin{bmatrix} 0.5v_x \\ 0.5v_y \end{bmatrix}_1$$

Covariance after single propagation:

$$\Sigma_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \Sigma_0 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^T + \Sigma_{\eta_0} = \begin{bmatrix} 0.2 & 0 \\ 0 & 0.2 \end{bmatrix}$$

B. How can we use this result iteratively?

$$\begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix}_t = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^T \begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix}_{t-1} + \begin{bmatrix} \Delta t & 0 \\ 0 & \Delta t \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix}_t + \begin{bmatrix} \mu_{\eta_x} \\ \mu_{\eta_y} \end{bmatrix}_t = \begin{bmatrix} x + v_x \Delta t + \eta_x \\ y + v_y \Delta t + \eta_y \end{bmatrix}_{t-1}$$

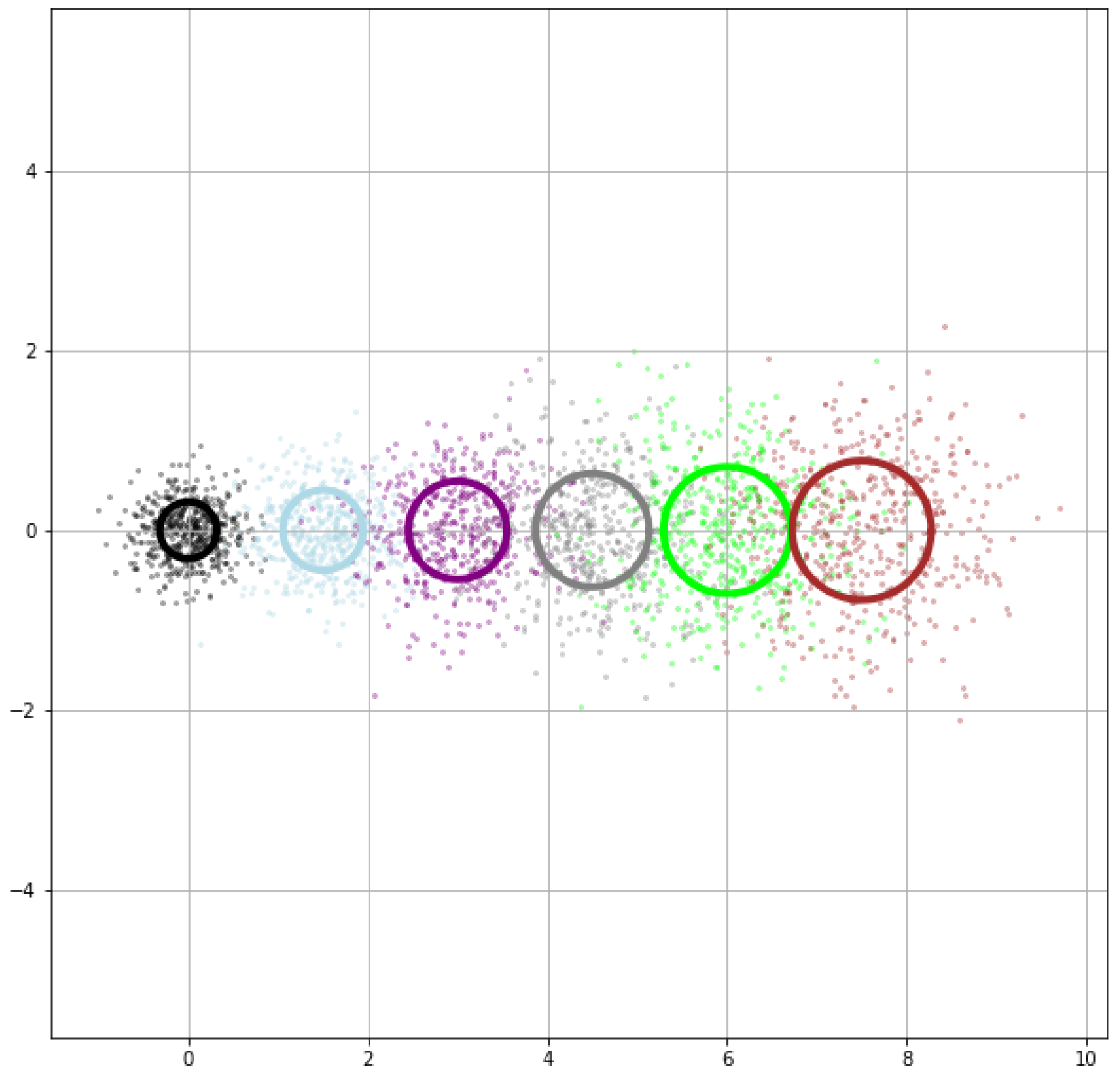
$$\Sigma_t = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \Sigma_{t-1} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^T + \Sigma_{\eta_t}$$

C. Draw the propagation state PDF (1-sigma iso-contour) for times indexes $t = 0...5$ and the control sequence $u_t = [3, 0]^T$ for all times t. The PDF for the initial state is:

$$\begin{bmatrix} x \\ y \end{bmatrix}_0 \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix} \right)$$

[ ]:
```
dt = 0.5
u = np.array([[3], [0]])
A = np.array([
    [1, 0],
    [0, 1]
])
mu = np.array([[0], [0]])
cov = np.array([
    [0.1, 0],
    [0, 0.1]
```

```
])
n_mu = np.array([[0], [0]])
n_cov = np.array([
    [0.1, 0],
    [0, 0.1]
])
k = 500
new_figure = True
cloud_colors = ["black", "lightblue", "purple", "gray", "lime", "brown"]
iso_colors = cloud_colors
for i in range(6):
    cloud = np.random.multivariate_normal(mu.reshape(2, 1).squeeze(), cov, k)
    plot2dcov_with_cloud(cloud, mu, cov, 1, new_figure=new_figure,␣
 ↪legend=False, sample_iso=False, cloud_color=cloud_colors[i %␣
 ↪len(cloud_colors)], iso_colors=[iso_colors[i % len(iso_colors)]])
    mu = A @ mu + u * dt + n_mu
    cov = A @ cov @ A.T + n_cov
    new_figure = False
```



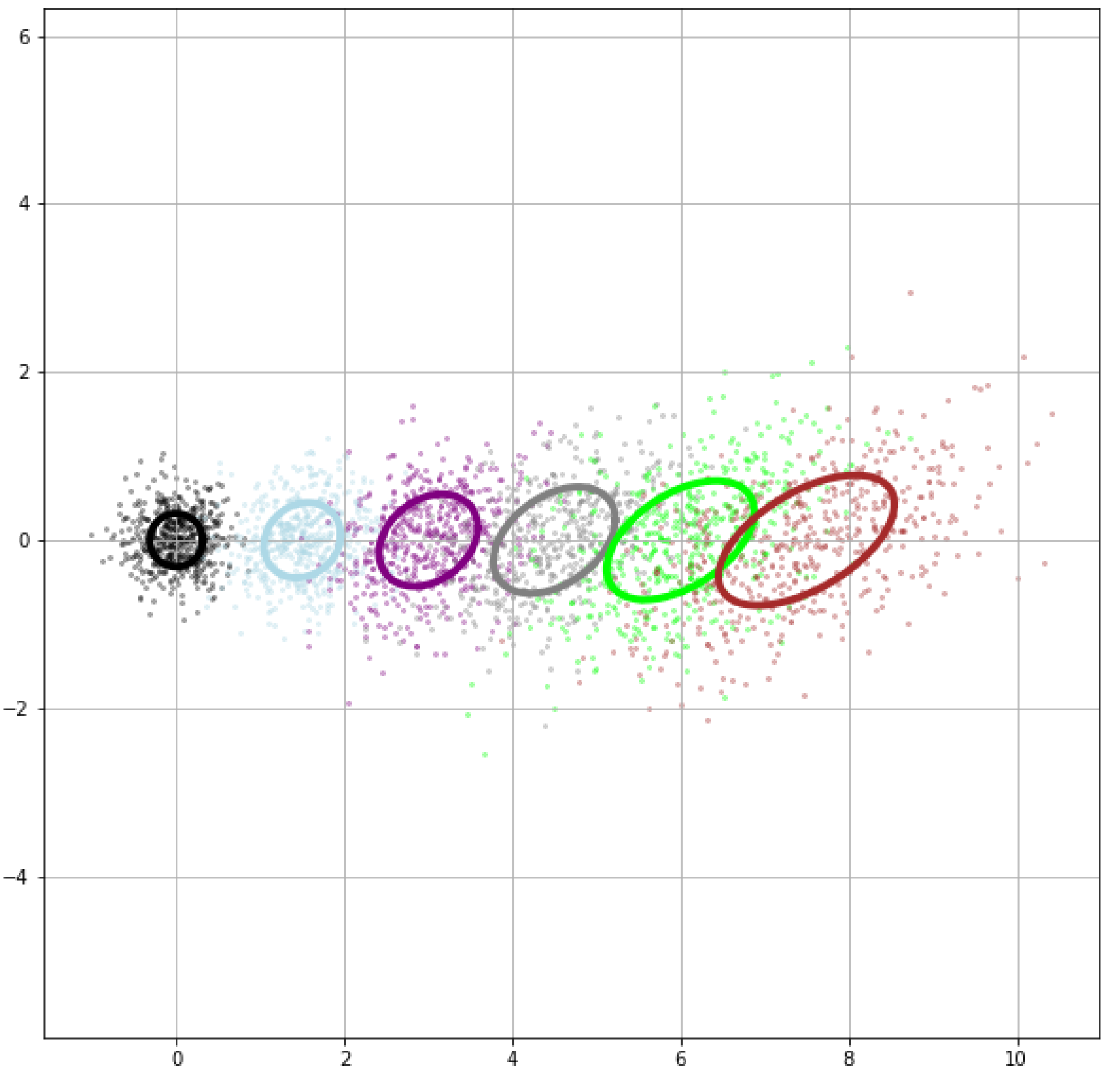
D. Somehow, the platform is malfunctioning; thus, it is moving strangely and its propagation model has changed:

$$\begin{bmatrix} x \\ y \end{bmatrix}_t = \begin{bmatrix} 1 & 0.3 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}_{t-1} + \begin{bmatrix} \Delta t & 0 \\ 0 & \Delta t \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix}_{t-1} + \begin{bmatrix} \eta_x \\ \eta_y \end{bmatrix}_t$$

All the other parameters and controls are the same as defined earlier. Draw the propagation state PDF (1-sigma iso-contour and 500 particles) for times indexes $t = 0 \ldots 5$

[ ]:
```
dt = 0.5
u = np.array([[3], [0]])
A = np.array([
    [1, 0.3],
    [0, 1]
```

```
])
mu = np.array([[0], [0]])
cov = np.array([
    [0.1, 0],
    [0, 0.1]
])
n_mu = np.array([[0], [0]])
n_cov = np.array([
    [0.1, 0],
    [0, 0.1]
])
k = 500
new_figure = True
for i in range(6):
    cloud = np.random.multivariate_normal(mu.reshape(2, 1).squeeze(), cov, k)
    plot2dcov_with_cloud(cloud, mu, cov, 1, new_figure=new_figure,␣
 ↪legend=False, sample_iso=False, cloud_color=cloud_colors[i %␣
 ↪len(cloud_colors)], iso_colors=[iso_colors[i % len(iso_colors)]])
    mu = A @ mu + u * dt + n_mu
    cov = A @ cov @ A.T + n_cov
    new_figure = False
```



E. Now, suppose that the robotic platform is non-holonomic, and the corresponding propagation model is:

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_t = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_{t-1} + \begin{bmatrix} cos(\theta)\Delta t & 0 \\ sin(\theta)\Delta t & 0 \\ 0 & \Delta t \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}_{t-1} + \begin{bmatrix} \eta_x \\ \eta_y \\ \eta_\theta \end{bmatrix}_t$$

and the PDF for the initial state

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_0 \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.5 \end{bmatrix} \right)$$

Propagate, as explained in class (linearize plus covariance propagation), for five time intervals, using the control $u_t = [3, 1.5]^T$ showing the propagated Gaussian by plotting

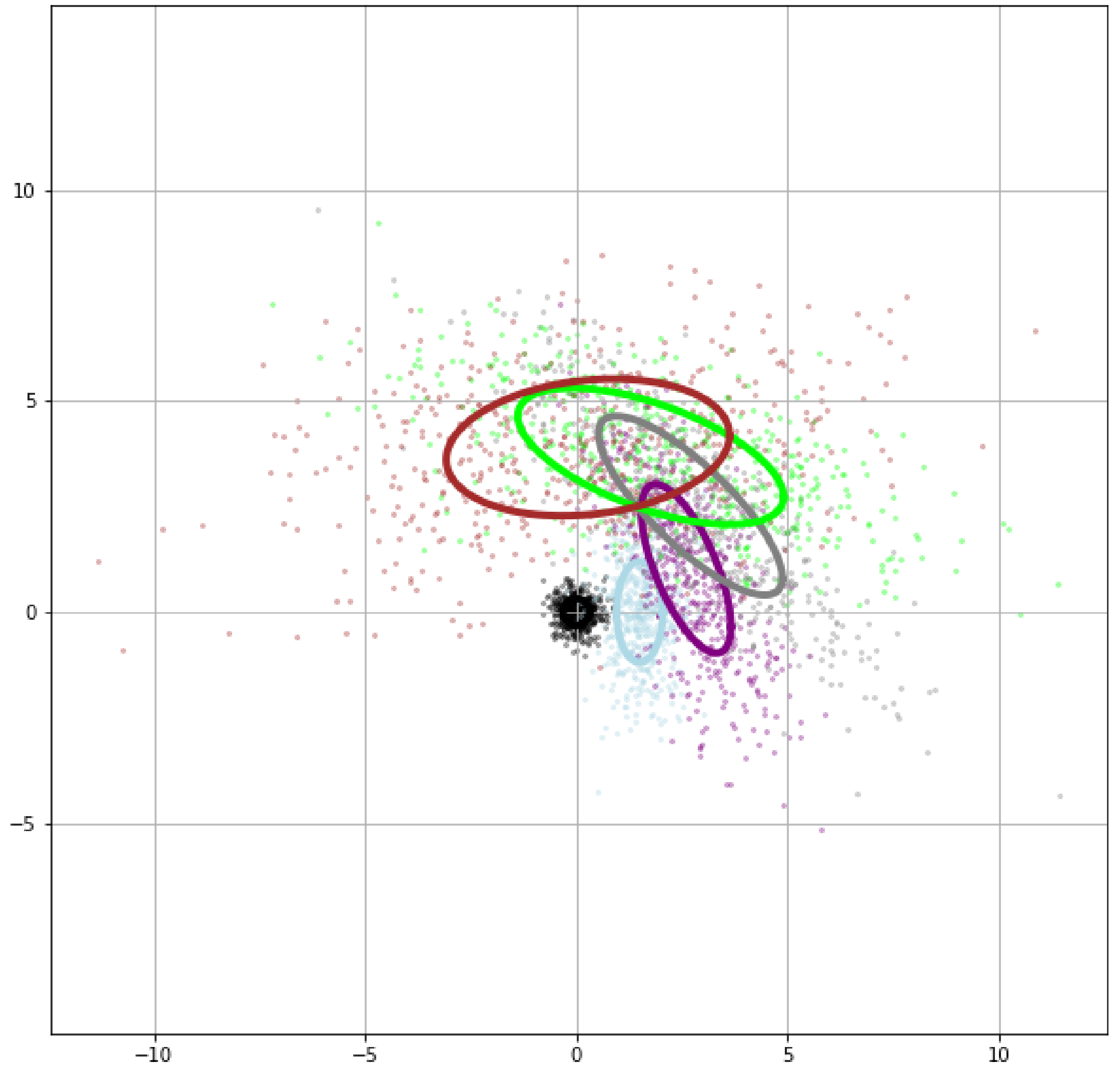
the 1-sigma iso-contour. Angles are in radians. Hint: you can marginalize out θ and plot the corresponding $\Sigma(xy)$ as explained in class

[ ]:
```
dt = 0.5
u = np.array([[3], [1.5]])
A = np.array([
    [1, 0, 0],
    [0, 1, 0],
    [0, 0, 1]
])
mu = np.array([[0], [0], [0]])
cov = np.array([
    [0.1, 0, 0],
    [0, 0.1, 0],
    [0, 0, 0.5]
])

def V(mu: np.ndarray) -> np.ndarray:
    return np.array([
        [np.cos(mu[2, 0]), 0],
        [np.sin(mu[2, 0]), 0],
        [0, 1],
    ]) * dt

def G(mu: np.ndarray) -> np.ndarray:
    return np.array([
        [1, 0, -np.sin(mu[2, 0]) * dt * u[0, 0]],
        [0, 1, np.cos(mu[2, 0]) * dt * u[0, 0]],
        [0, 0, 1],
    ])

n_mu = np.array([[0], [0], [0]])
n_cov = np.array([
    [0.2, 0, 0],
    [0, 0.2, 0],
    [0, 0, 0.1]
])
k = 500
new_figure = True
for i in range(6):
    with np.printoptions(precision=4, suppress=True):
        print(f"Mean on iteration {i}:\n {mu}")
        print(f"Covariance on iteration {i}:\n {cov}")
    cloud = np.random.multivariate_normal(mu.reshape(3, 1).squeeze(), cov, k)
    plot2dcov_with_cloud(cloud, mu, cov, 1, new_figure=new_figure,␣
↪legend=False, sample_iso=False, cloud_color=cloud_colors[i %␣
↪len(cloud_colors)], iso_colors=[iso_colors[i % len(iso_colors)]])
    cov = G(mu) @ cov @ G(mu).T + n_cov
```

```python
    mu = A @ mu + V(mu) @ u + n_mu
    new_figure = False
```

```
Mean on iteration 0:
 [[0]
 [0]
 [0]]
Covariance on iteration 0:
 [[0.1 0.  0. ]
 [0.  0.1 0. ]
 [0.  0.  0.5]]
Mean on iteration 1:
 [[1.5 ]
 [0.  ]
 [0.75]]
Covariance on iteration 1:
 [[0.3   0.    0.   ]
 [0.    1.425 0.75 ]
 [0.    0.75  0.6  ]]
Mean on iteration 2:
 [[2.5975]
 [1.0225]
 [1.5   ]]
Covariance on iteration 2:
 [[ 1.1273 -1.4402 -0.6135]
 [-1.4402  3.994   1.4085]
 [-0.6135  1.4085  0.7   ]]
Mean on iteration 3:
 [[2.7036]
 [2.5187]
 [2.25  ]]
Covariance on iteration 3:
 [[ 4.7302 -3.7239 -1.6608]
 [-3.7239  4.5008  1.4828]
 [-1.6608  1.4828  0.8   ]]
Mean on iteration 4:
 [[1.7614]
 [3.6858]
 [3.    ]]
Covariance on iteration 4:
 [[ 9.8967 -3.0097 -2.5945]
 [-3.0097  2.6168  0.729 ]
 [-2.5945  0.729   0.9   ]]
Mean on iteration 5:
 [[0.2764]
 [3.8975]
 [3.75  ]]
```

```
Covariance on iteration 5:
 [[11.2354  0.9717 -2.785 ]
 [ 0.9717  2.6364 -0.6075]
 [-2.785  -0.6075  1.    ]]
```



F. Repeat the same experiment as above, using the same control input ut and initial state estimate, now considering that noise is expressed in the action space instead of state space:

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_t = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_{t-1} + \begin{bmatrix} cos(\theta)\Delta t & 0 \\ sin(\theta)\Delta t & 0 \\ 0 & \Delta t \end{bmatrix} \begin{bmatrix} v + \eta_x \\ \omega + \eta_\omega \end{bmatrix}_t$$

being

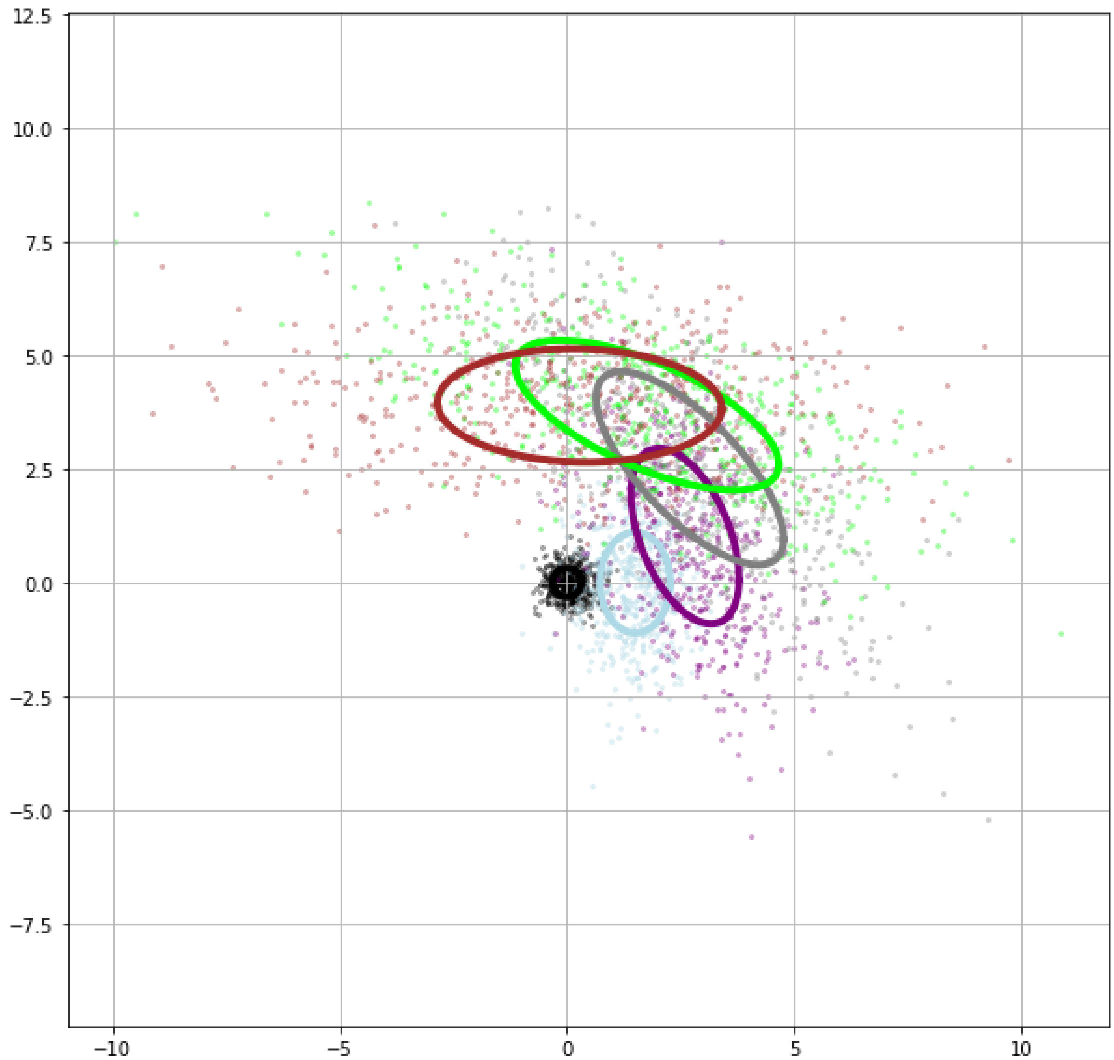
$$\begin{bmatrix} \eta_v \\ \eta_\omega \end{bmatrix}_t \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 2 & 0 \\ 0 & 0.1 \end{bmatrix} \right)$$

Comment on the results.

[ ]:
```
dt = 0.5
u = np.array([[3], [1.5]])
A = np.array([
    [1, 0, 0],
    [0, 1, 0],
    [0, 0, 1]
])
mu = np.array([[0], [0], [0]])
cov = np.array([
    [0.1, 0, 0],
    [0, 0.1, 0],
    [0, 0, 0.5]
])

n_mu = np.array([[0], [0]])
n_cov = np.array([
    [2, 0],
    [0, 0.1],
])
k = 500
new_figure = True
for i in range(6):
    with np.printoptions(precision=4, suppress=True):
        print(f"Mean on iteration {i}:\n {mu}")
        print(f"Covariance on iteration {i}:\n {cov}")
    cloud = np.random.multivariate_normal(mu.reshape(3, 1).squeeze(), cov, k)
    plot2dcov_with_cloud(cloud, mu, cov, 1, new_figure=new_figure,␣
 ↪legend=False, sample_iso=False, cloud_color=cloud_colors[i %␣
 ↪len(cloud_colors)], iso_colors=[cloud_colors[i % len(cloud_colors)]])
    cov = G(mu) @ cov @ G(mu).T + V(mu) @ n_cov @ V(mu).T
    mu = A @ mu + V(mu) @ (u + n_mu)
    new_figure = False
```

```
Mean on iteration 0:
 [[0]
 [0]
 [0]]
Covariance on iteration 0:
 [[0.1 0.  0. ]
 [0.  0.1 0. ]
 [0.  0.  0.5]]
Mean on iteration 1:
```

```
 [[1.5 ]
 [0.  ]
 [0.75]]
Covariance on iteration 1:
 [[0.6   0.    0.   ]
 [0.    1.225 0.75 ]
 [0.    0.75  0.525]]
Mean on iteration 2:
 [[2.5975]
 [1.0225]
 [1.5   ]]
Covariance on iteration 2:
 [[ 1.4165 -1.1066 -0.5368]
 [-1.1066  3.736   1.3262]
 [-0.5368  1.3262  0.55  ]]
Mean on iteration 3:
 [[2.7036]
 [2.5187]
 [2.25  ]]
Covariance on iteration 3:
 [[ 4.2567 -3.1999 -1.3597]
 [-3.1999  4.5211  1.3846]
 [-1.3597  1.3846  0.575 ]]
Mean on iteration 4:
 [[1.7614]
 [3.6858]
 [3.    ]]
Covariance on iteration 4:
 [[ 8.4111 -3.1467 -2.0308]
 [-3.1467  2.7251  0.8428]
 [-2.0308  0.8428  0.6   ]]
Mean on iteration 5:
 [[0.2764]
 [3.8975]
 [3.75  ]]
Covariance on iteration 5:
 [[ 9.7878 -0.1906 -2.1578]
 [-0.1906  1.5552 -0.0482]
 [-2.1578 -0.0482  0.625 ]]
```


12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10
−5
0
5
10