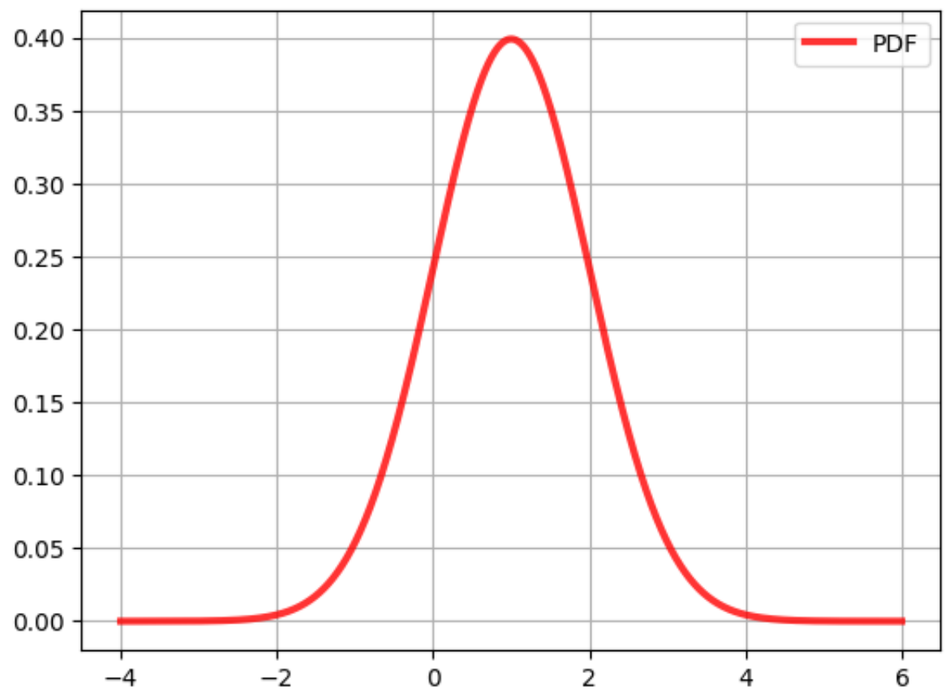
Task 1: Probability (25 points)

A. (5 pts) Plot the probability density function p(x) of a one dimensional Gaussian distribution N(x; 1; 1).

Hint: you might want to look at the library scipy.stats and use the function norm.pdf().

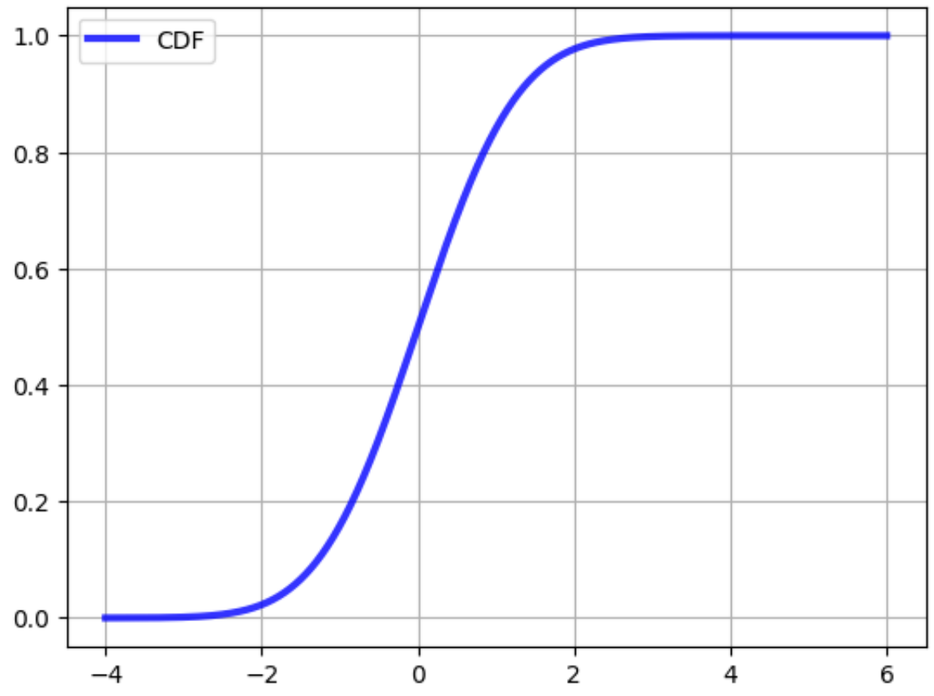
In [ ]:
```
import numpy as np
from scipy.stats import norm
from scipy.stats import multivariate_normal
import matplotlib.pyplot as plt
%matplotlib inline
mu_x = 1
sigma_x = 1
x = np.arange(mu_x - 5 * sigma_x, mu_x + 5 * sigma_x, 0.001)
p_x = norm.pdf(x, loc = mu_x, scale = sigma_x)
plt.plot(x, p_x, 'r-', lw = 3, alpha = 0.8, label = "PDF")
plt.legend()
plt.grid()
```



B. (5 pts) Calculate the probability R mass that the random variable X is less than 0

In [ ]:
```
plt.plot(x, norm.cdf(x), 'b-', lw = 3, alpha = 0.8, label = 'CDF')
plt.legend()
plt.grid()
print("CDF for x from -inf to 0 = ", norm.cdf(0, loc=mu_x, scale=sigma_x))
```

```
CDF for x from -inf to 0 =  0.15865525393145707
```



C. (15 pts) Consider the new observation variable z, it gives information about the variable x by the likelihood function p(z|x) = N(z; x; σ^2), with variance σ^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.

Hint: There are different ways in which the normalization factor for the posterior can be calculated (e.g. or direct derivation, numerical normalization, etc.). Choose one method.

In [ ]:
```
z = 0.75
sigma_z = 0.2 ** 0.5
p_z_x = norm.pdf(z, loc = x, scale = sigma_z)
plt.plot(x, p_x, 'b-', lw = 3, alpha = 0.8, label = 'PDF p(x)')
p_x_z = p_z_x * p_x / (np.trapz(p_z_x * p_x, x = x))
plt.plot(x, p_x_z, 'r-', lw = 3, alpha = 0.8, label = 'PDF p(x|z)')
plt.grid()
plt.legend();
```

Task 2: Multivariate Gaussian (35 points)

A. (15 pts) 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.

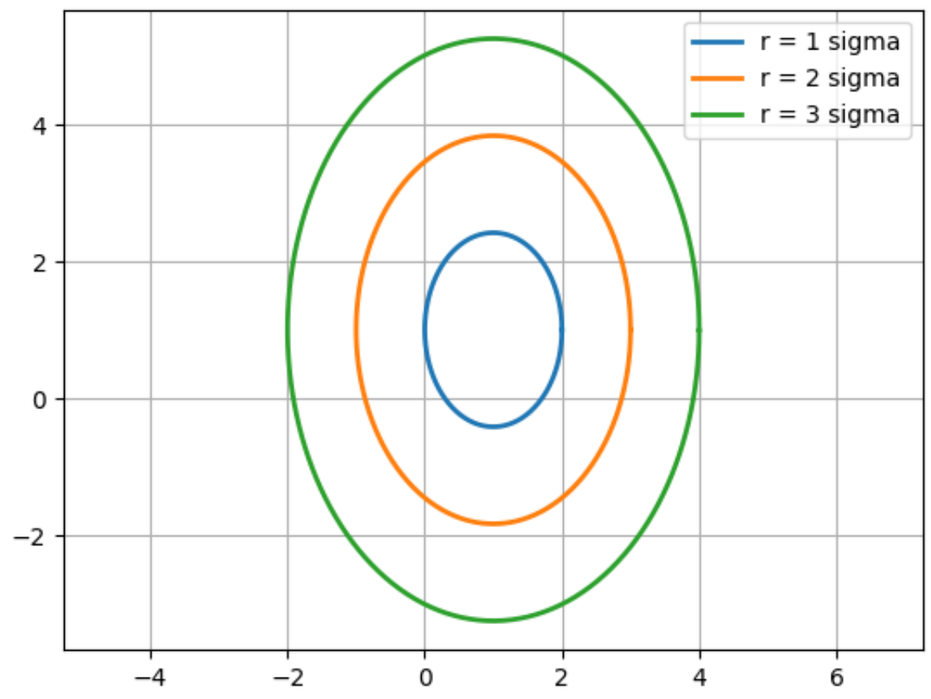
Hint: Make use of the Cholesky decomposition `scipy.linalg.cholesky` or SVD and project a circumference with radius k, as explained in class. Use, for instance, 30 points.

Hint: the Cholesky decomposition routine, could solve for the upper triangular matrix, $A^\top \cdot A = \Sigma$ which is not what you may want. Read the function help and make sure the decomposition maps back to the original covariance in the form $A \cdot A^\top = \Sigma$.

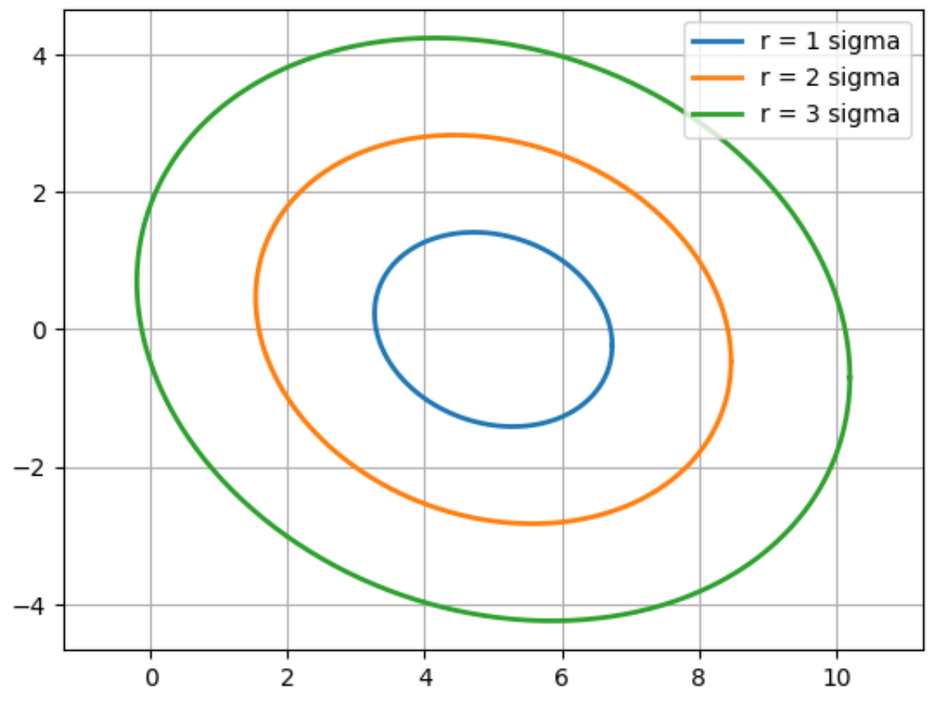
Then, use `plot2dcov` to draw the iso-contours corresponding to 1,2,3-sigma of the following Gaussian distributions: $\mathcal{N}\left(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}1 & 0\\0 & 2\end{bmatrix}\right)$, $\mathcal{N}\left(\begin{bmatrix}5\\0\end{bmatrix}, \begin{bmatrix}3 & -0.4\\-0.4 & 2\end{bmatrix}\right)$ and $\mathcal{N}\left(\begin{bmatrix}2\\2\end{bmatrix}, \begin{bmatrix}9.1 & 6\\6 & 4\end{bmatrix}\right)$. Use the `set_aspect('equal')` command and comment on them.

In [ ]:
```
def plot2dcov (mu, cov, k, label):
    L = np.linalg.cholesky(cov)
    x = []
    y = []
    for ph in np.arange(0, 2 * np.pi, 0.01):
        x.append((L @ np.array([[np.cos(ph)], [np.sin(ph)]]))[0] * k + mu[0, 0])
        y.append((L @ np.array([[np.cos(ph)], [np.sin(ph)]]))[1] * k + mu[1, 0])
    plt.plot(x ,y, label = label, lw = 2)
    plt.axis('equal')
    plt.grid(True)
    plt.legend()
```

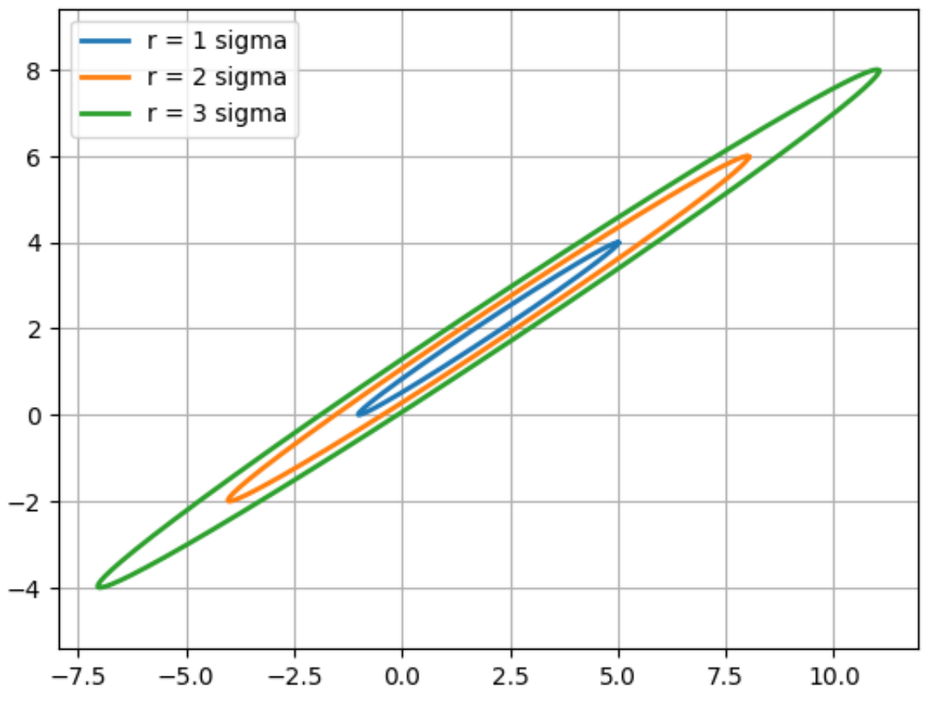
In [ ]:
```
cov = np.array([[1, 0], [0, 2]])
mu = np.array([[1], [1]])

plot2dcov(mu, cov, 1, "r = 1 sigma")
plot2dcov(mu, cov, 2, "r = 2 sigma")
plot2dcov(mu, cov, 3, "r = 3 sigma")
```



In [ ]:
```
cov = np.array([[3, -0.4], [-0.4, 2]])
mu = np.array([[5], [0]])

plot2dcov(mu, cov, 1, "r = 1 sigma")
plot2dcov(mu, cov, 2, "r = 2 sigma")
plot2dcov(mu, cov, 3, "r = 3 sigma")
```



In [ ]:
```
cov = np.array([[9.1, 6], [6, 4]])
mu = np.array([[2], [2]])

plot2dcov(mu, cov, 1, "r = 1 sigma")
plot2dcov(mu, cov, 2, "r = 2 sigma")
plot2dcov(mu, cov, 3, "r = 3 sigma")
```



B. (5 pts) 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. You can provide your solution by using Markdown, latex, by hand, etc.

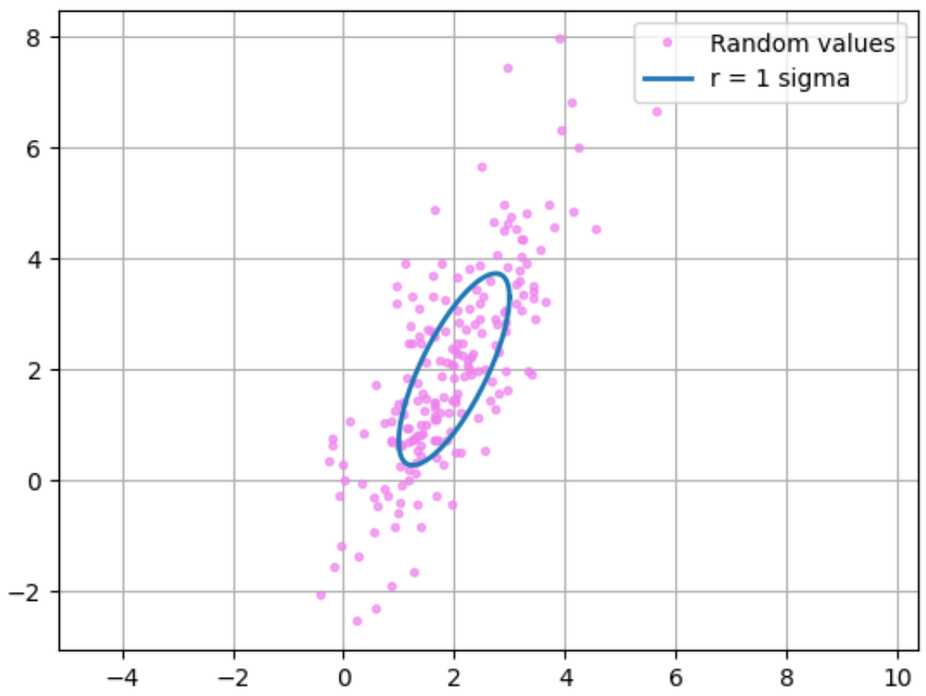
Sample mean: $\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$

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

C. (15 pts) 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.

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

X = np.array([[x], [x]])
```

In [ ]:
```
x, y = np.random.multivariate_normal(mu, cov, 200).T
plt.plot(x, y, '.', alpha = 0.7, c = "violet", label = "Random values")
plot2dcov (mu.reshape((2, 1)), cov, 1, "r = 1 sigma")
```



In [ ]:
```
X = np.stack((x, y), axis=0)
print("Covariance matrix:\n", np.cov(X))
```

```
Covariance matrix:
 [[1.0628067  1.42758647]
 [1.42758647 3.30557331]]
```

In [ ]:
```
print('Mean x = ',np.mean(x))
print('Mean y = ',np.mean(y))
```

```
Mean x =  1.9173331879621116
Mean y =  1.9742120784269388
```

In [ ]:
```
plt.plot(x, y, '.', label = "Random values", alpha = 0.7, c = "violet")
plt.grid()
plot2dcov (mu.reshape((2, 1)), cov, 1, "r = 1 sigma")
plot2dcov(np.array([[np.mean(x)], [np.mean(y)]]), np.cov(X), 1, "r = 1 new sigma")
```

The factic covariance and the mean value depend on the number of samples: more samples - real parameters are more closer to the original ones. The calculated parameters will coincide with the original ones only when N tends to infinity

Task 3: Covariance Propagation (40 points)

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 $u = [v_x, v_y]^\top$ 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. (5 pts) Write the equations corresponding to the mean and covariance after a single propagation of the holonomic platform.

$$\mu_1 = E\{A \cdot x_0 + B \cdot u_1 + \Sigma_1\} = A \cdot \mu_0 + B \cdot u_1$$

$$\Sigma_1 = E\{(A x_0 + B u_1 + \Sigma_1 - A \cdot \mu_0 - B \cdot u_1)(A x_0 + B \cdot u_1 + \Sigma_1 - A \cdot \mu_0 - B u_1)^T\}$$

$$= E\{(A(x_0 - \mu_0) + \Sigma_1)(A(x_0 - \mu_0) + \Sigma_1)^T\} =$$

$$= E\{A(x_0 - \mu_0)(x_0 - \mu_0)^T A^T + A(x_0 - \mu_0)\Sigma_1^T + \Sigma_1 (x_0 - \mu_0)^T A^T +$$

$$+ \Sigma_1 \cdot \Sigma_1^T\} = \left| \begin{array}{l} \Sigma, x\text{-uncorrelated} \\ E\{\Sigma\} = 0 \end{array} \right| = A \Sigma_0 A^T + R$$

B. (5 pts) How can we use this result iteratively?

$$\mu_t = E\{A_t x_{t-1} + B_t u_t + \Sigma_t\} = A_t \mu_{t-1} + B_t u_t$$

$$\Sigma_t = E\{(A_t x_{t-1} + B_t u_t + \Sigma_t - A_t \mu_{t-1} - B_t u_t)(A_t x_{t-1} + B_t u_t + \Sigma_t - A_t \mu_{t-1} - B_t u_t)^T\}$$

$$= E\{(A_t(x_{t-1} - \mu_{t-1}) + \Sigma_t)(A_t(x_{t-1} - \mu_{t-1}) + \Sigma_t)^T\} =$$

$$= E\{A_t(x_{t-1} - \mu_{t-1})(x_{t-1} - \mu_{t-1})^T A_t^T + A_t(x_{t-1} - \mu_{t-1})\Sigma_t^T + \Sigma_t(x_{t-1} - \mu_{t-1})^T A_t^T +$$

$$+ \Sigma_t \cdot \Sigma_t^T\} = \left| \begin{array}{l} \Sigma, x\text{-uncorrelated} \\ E\{\Sigma\} = 0 \end{array} \right| = A_t \Sigma_{t-1} A_t^T + R$$

C. (7 pts) Draw the propagation state PDF (1-sigma iso-contour) for times indexes $t = 0, \ldots, 5$ and the control sequence $u_t = [3, 0]^\top$ for all times t. The PDF for the initial state is $\begin{bmatrix} x \\ y \end{bmatrix}_0 \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}\right)$.

In [ ]:
```
dt = 0.5
A = np.eye(2)
B = np.eye(2) * dt
R = np.eye(2) * 0.1
```

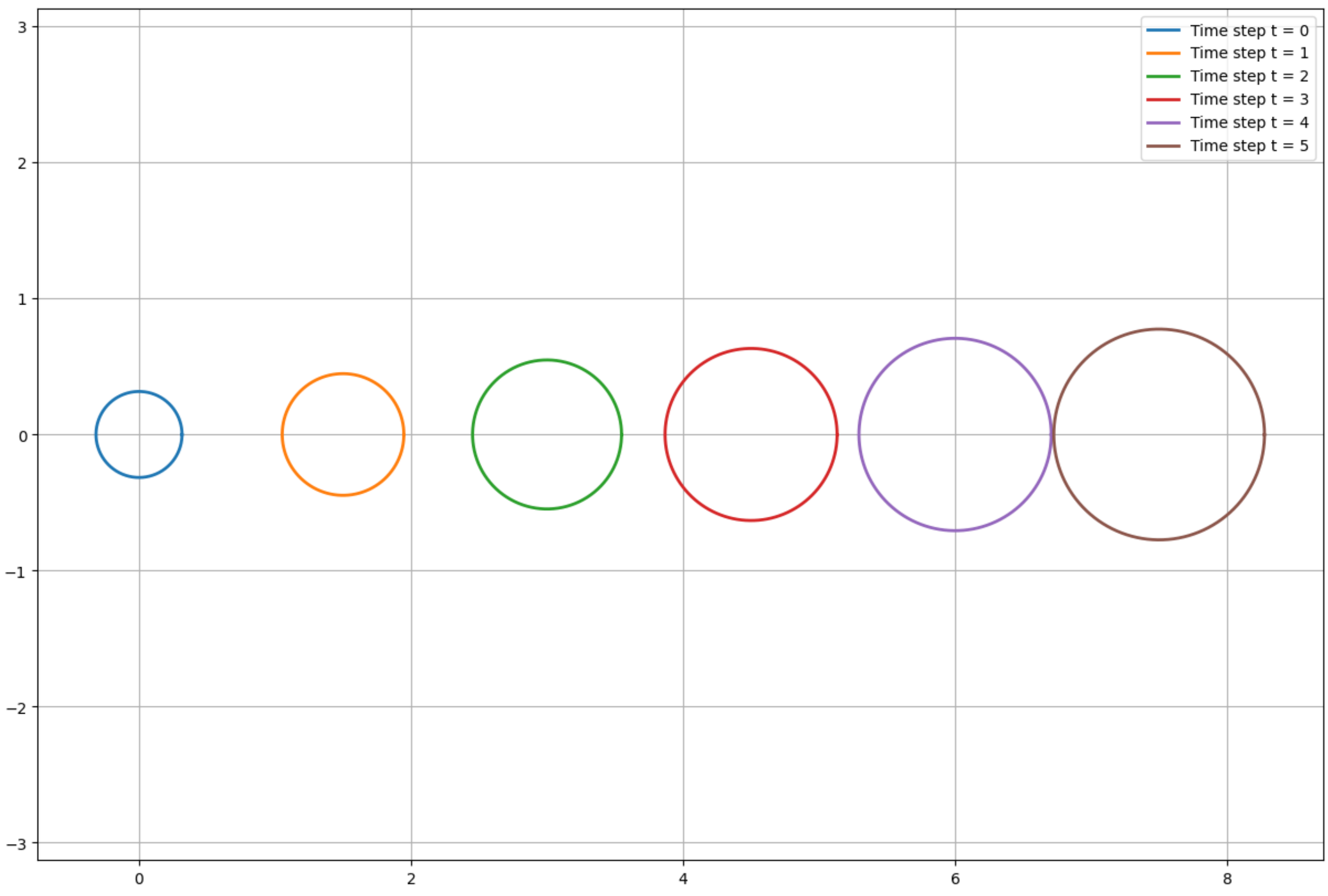
In [ ]:
```
mu = np.zeros([2, 1, 6])
cov = np.zeros([2, 2, 6])
ut = np.array([[3], [0]])
```

In [ ]:
```
mu[:, :, 0] = np.array([[0], [0]])
```

In [ ]:
```
cov[:,:,0] = np.array([[0.1, 0], [0, 0.1]])
```

In [ ]:
```
for i in range(1, 6):
    mu[:, :, i] = A @ mu[:, :, i - 1] + B @ ut
    cov[:, :, i] = A @ cov[:, :, i - 1] @ A.T + R
```

In [ ]:
```
plt.figure(figsize = [15, 10])
for i in range(6):
    plot2dcov(mu[:, :, i], cov[:, :, i], 1, "Time step t = " + str(i))
```

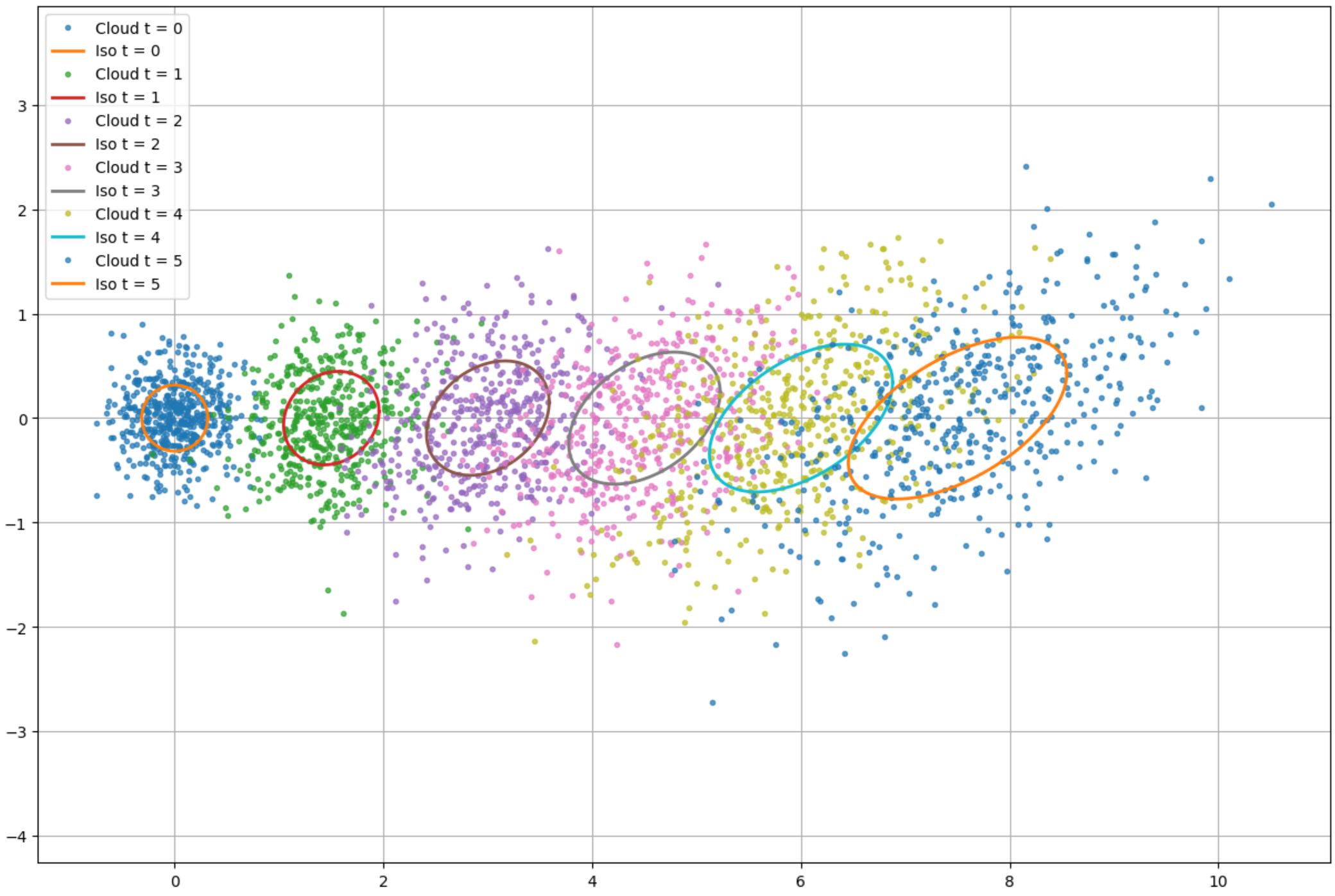


D. (8 pts) Somehow, the platform is malfunctioning; thus, it is moving strangely and its propagation model has changed: $\begin{bmatrix} x \\ y \end{bmatrix}_t = \begin{bmatrix} 1 & 0.3 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}_{t-1} + \begin{bmatrix} \Delta t & 0 \\ 0 & \Delta t \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix}_t + \begin{bmatrix} \eta_x \\ \eta_y \end{bmatrix}_t$. 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 $t = 0, \ldots, 5$,

In [ ]:
```
A = np.array([[1, 0.3], [0, 1]])
for i in range(1, 6):
    mu[:, :, i] = A @ mu[:, :, i - 1] + B @ ut
    cov[:, :, i] = A @ cov[:, :, i - 1] @ A.T + R
```

In [ ]:
```
plt.figure(figsize = [15, 10])
for i in range(6):
    x, y = np.random.multivariate_normal(mu[:, :, i].reshape(2, 1).squeeze(), cov[:, :, i], 500).T
    plt.plot(x, y, '.', alpha = 0.7, label = "Cloud t = " + str(i))
    plot2dcov(mu[:, :, i], cov[:, :, i], 1, "Iso t = " + str(i))
```



E. (7 pts) Now, suppose that the robotic platform is non-holonomic, and the corresponding propagation model is: $\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_t = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_{t-1} + \begin{bmatrix} cos(\theta)\Delta t & 0 \\ sin(\theta)\Delta t & 0 \\ 0 & \Delta t \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix}_t + \begin{bmatrix} \eta_x \\ \eta_y \\ \eta_\theta \end{bmatrix}_t$, being $\begin{bmatrix} \eta_x \\ \eta_y \\ \eta_\theta \end{bmatrix}_t \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0.2 & 0 & 0 \\ 0 & 0.2 & 0 \\ 0 & 0 & 0.1 \end{bmatrix} \right)$ and the PDF for the initial state $\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_0 \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.5 \end{bmatrix} \right)$.

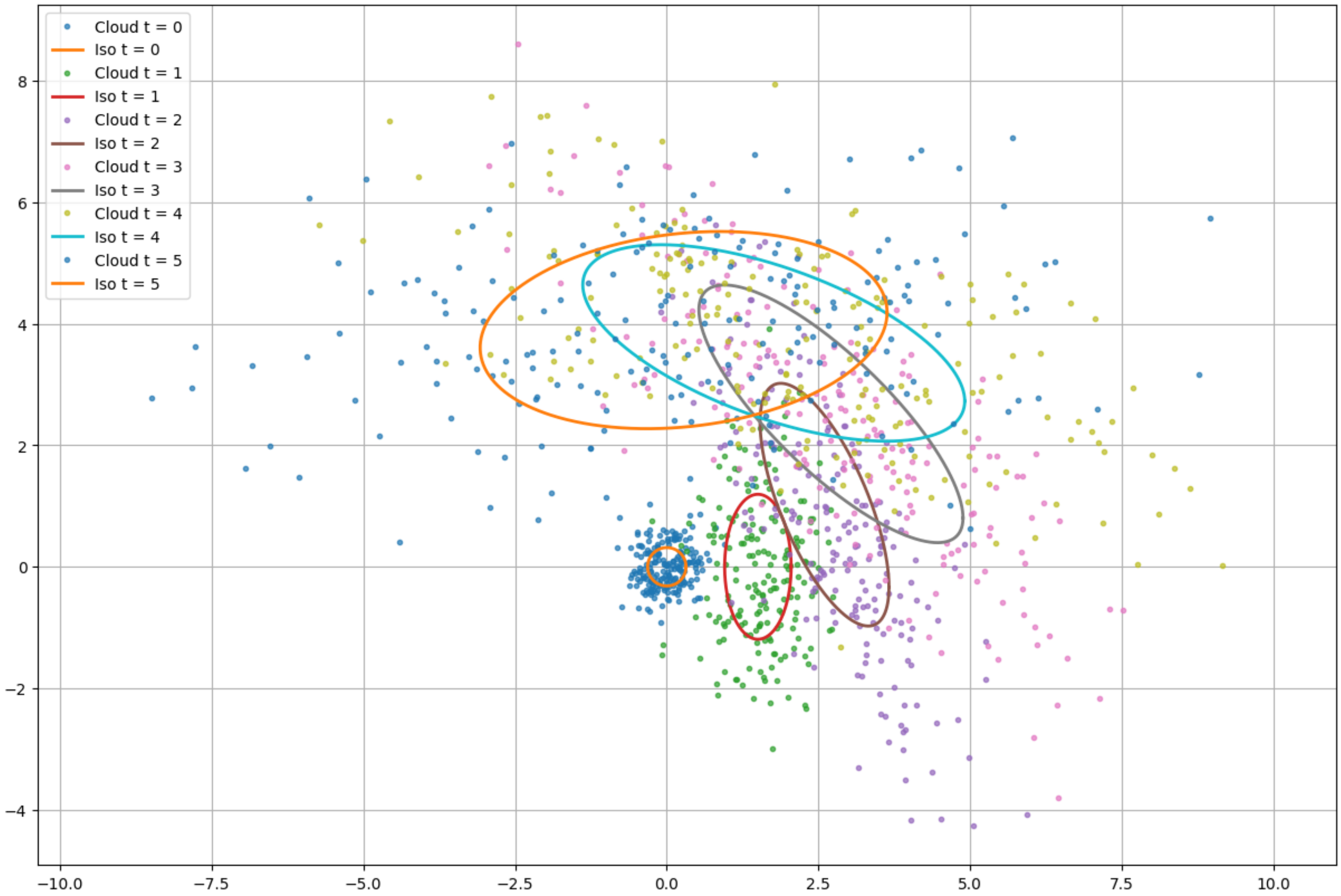
Propagate, as explained in class (linearize plus covariance propagation), for five time intervals, using the control $u_t = [3, 1.5]^\top$ showing the propagated Gaussian by plotting the 1-sigma iso-contour. Angles are in radians. *Hint:* you can marginalize out θ and plot the corresponding $\Sigma_{(xy)}$ as explained in class.

In [ ]:
```
A = np.eye(3)
R = np.array([[0.2, 0, 0], [0, 0.2, 0], [0, 0, 0.1]])
G = np.zeros([3, 3, 6])
V = np.zeros([3, 2, 6])
mu = np.zeros([3, 1, 6])
cov = np.zeros([3, 3, 6])
ut = np.array([[3], [1.5]])
```

In [ ]:
```
G[:, :, 0] = np.array([[1, 0, -np.sin(mu[2, 0, 0]) * dt * ut[0, 0]],
                       [0, 1,  np.cos(mu[2, 0, 0]) * dt * ut[0, 0]],
                       [0, 0,             1     ]])

V[:, :, 0] = np.array([[np.cos(mu[2, 0, 0]) * dt, 0], [np.sin(mu[2, 0, 0]) * dt, 0], [0, dt]])

mu[:, :, 0] = np.array([[0], [0], [0]])
cov[:, :, 0] = np.array([[0.1, 0, 0], [0, 0.1, 0], [0, 0, 0.5]])
```

In [ ]:
```
for i in range(5):
    mu[:, :, i + 1] = A @ mu[:, :, i] + V[:, :, i] @ ut
    cov[:, :, i + 1] = G[:, :, i] @ cov[:, :, i] @ G[:, :, i].T + R
    G[:, :, i + 1] = np.array([[1, 0, -np.sin(mu[2, 0, i + 1]) * dt * ut[0, 0]],
                               [0, 1,  np.cos(mu[2, 0, i + 1]) * dt * ut[0, 0]],
                               [0, 0,                                      1]])

    V[:, :, i + 1] = np.array([[np.cos(mu[2, 0, i + 1]) * dt, 0], [np.sin(mu[2, 0, i + 1]) * dt, 0], [0, dt]])
```

In [ ]:
```
plt.figure(figsize = [15, 10])
for i in range(6):
    x, y = np.random.multivariate_normal(mu[:2, 0, i], cov[:2, :2, i], 200).T
    plt.plot(x, y, '.', alpha = 0.7, label = "Cloud t = " + str(i))
    plot2dcov(mu[:2, :2, i], cov[:2, :2, i], 1, "Iso t = " + str(i))
```



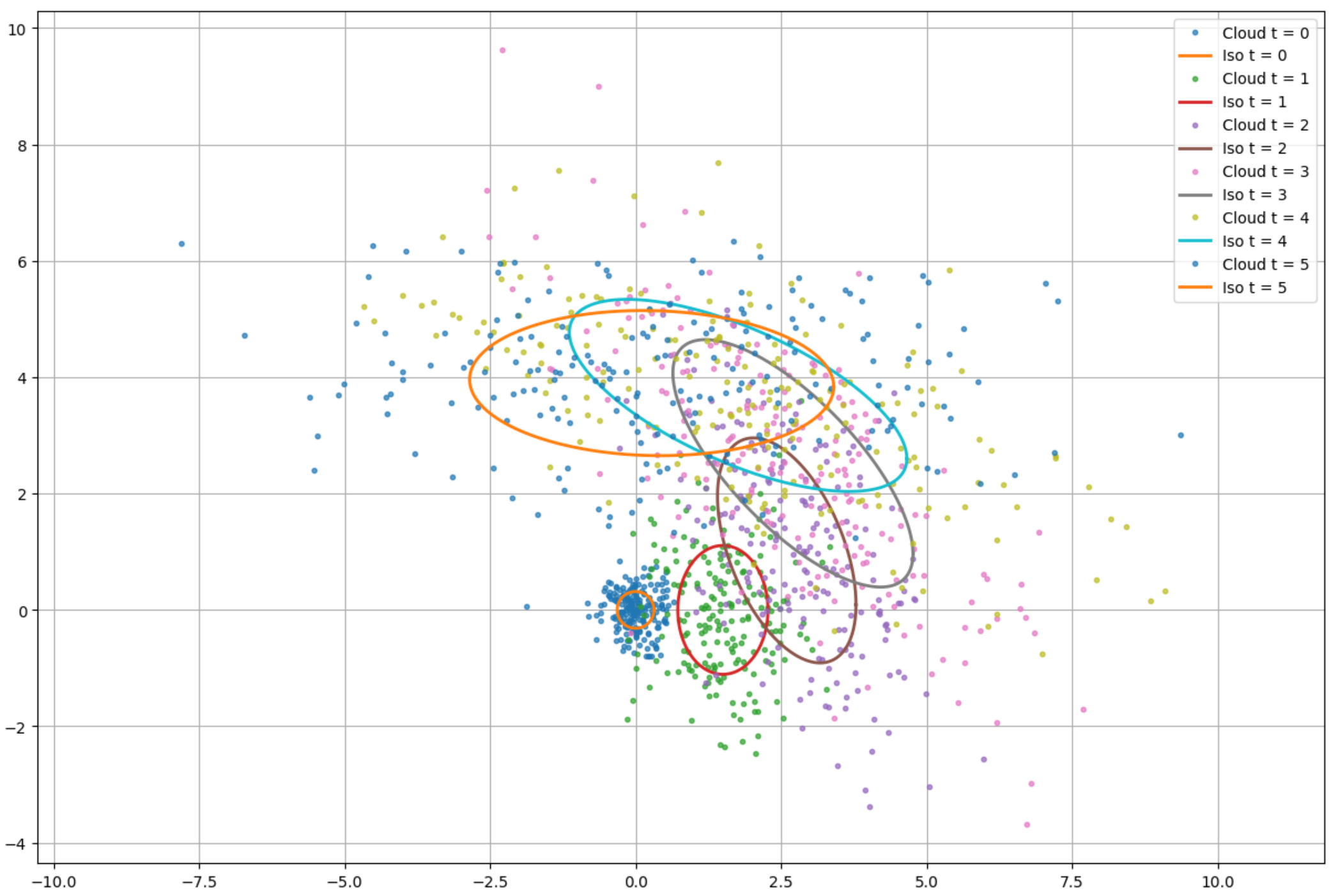
F. (8 pts) Repeat the same experiment as above, using the same control input u_t and initial state estimate, now considering that noise is expressed in the action space instead of state space: $\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_t = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_{t-1} + \begin{bmatrix} cos(\theta)\Delta t & 0 \\ sin(\theta)\Delta t & 0 \\ 0 & \Delta t \end{bmatrix} \begin{bmatrix} v + \eta_v \\ w + \eta_w \end{bmatrix}_t$, being $\begin{bmatrix} \eta_v \\ \eta_w \end{bmatrix}_t \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 2 & 0 \\ 0 & 0.1 \end{bmatrix}\right)$. Comment on the results.

In [ ]:
```
A = np.eye(3)
R = np.array([[2, 0], [0, 0.1]])
G = np.zeros([3, 3, 6])
V = np.zeros([3, 2, 6])
mu = np.zeros([3, 1, 6])
cov = np.zeros([3, 3, 6])
ut = np.array([[3], [1.5]])
```

In [ ]:
```
G[:, :, 0] = np.array([[1, 0, -np.sin(mu[2, 0, 0]) * dt * ut[0, 0]],
                       [0, 1,  np.cos(mu[2, 0, 0]) * dt * ut[0, 0]],
                       [0, 0,             1     ]])

V[:, :, 0] = np.array([[np.cos(mu[2, 0, 0]) * dt, 0], [np.sin(mu[2, 0, 0]) * dt, 0], [0, dt]])

mu[:, :, 0] = np.array([[0], [0], [0]])
cov[:, :, 0] = np.array([[0.1, 0, 0], [0, 0.1, 0], [0, 0, 0.5]])
```

In [ ]:
```
for i in range(5):
    mu[:, :, i + 1] = A @ mu[:, :, i] + V[:, :, i] @ ut
    cov[:, :, i + 1] = G[:, :, i] @ cov[:, :, i] @ G[:, :, i].T + V[:, :, i] @ R @ V[:, :, i].T
    G[:, :, i + 1] = np.array([[1, 0, -np.sin(mu[2, 0, i + 1]) * dt * ut[0, 0]],
                               [0, 1,  np.cos(mu[2, 0, i + 1]) * dt * ut[0, 0]],
                               [0, 0,                                    1]])

    V[:, :, i + 1] = np.array([[np.cos(mu[2, 0, i + 1]) * dt, 0], [np.sin(mu[2, 0, i + 1]) * dt, 0], [0, dt]])
```

In [ ]:
```
plt.figure(figsize = [15, 10])
for i in range(6):
    x, y = np.random.multivariate_normal(mu[:2, 0, i], cov[:2, :2, i], 200).T
    plt.plot(x, y, '.', alpha = 0.7, label = "Cloud t = " + str(i))
    plot2dcov(mu[:2, :2, i], cov[:2, :2, i], 1, "Iso t = " + str(i))
```



Because of uncertainty in control impact appared inaccuracy in estimating of robot position: parameters x and y became correlated (it can be seen due to rotation angle changes) and their variance increased on each step ((it can be due to ellipse radiouses changes)). The noise impact was less than in previous task due to the noise causes: centers of ellipses are approximately at the same places, but the covariance became less.