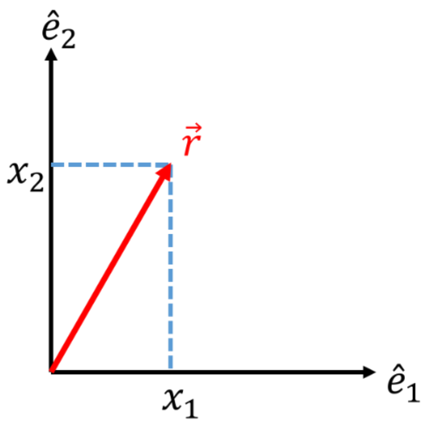
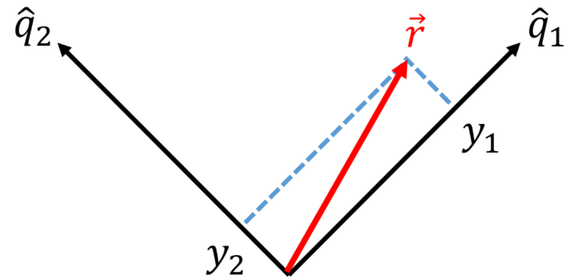
Ellipse and
Gaussian Distribution

Industrial AI Lab.

Coordinates with Basis

- Basis $\{\hat{e}_1 \ \hat{e}_2\}$ or basis $\{\hat{q}_1 \ \hat{q}_2\}$

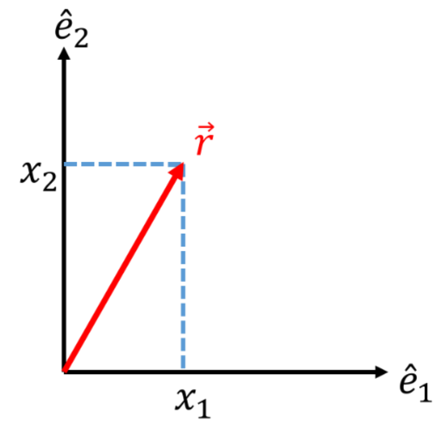


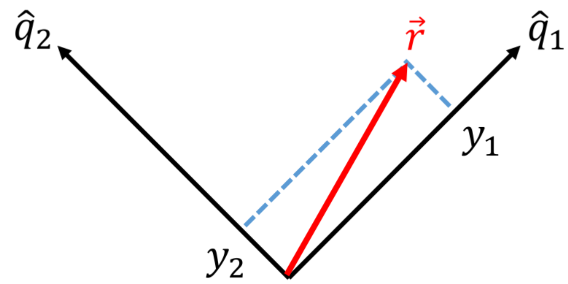


$$\vec{r}_I = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} : \text{coordinate of } \vec{r} \text{ in basis } \{\hat{e}_1 \ \hat{e}_2\} \ (= I)$$

$$\vec{r}_Q = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} : \text{coordinate of } \vec{r} \text{ in basis } \{\hat{q}_1 \ \hat{q}_2\} \ (= Q)$$

Coordinate Transformation





$$
\begin{aligned}
\vec{r} &= x_1\hat{e}_1 + x_2\hat{e}_2 = y_1\hat{q}_1 + y_2\hat{q}_2 \\
&= \begin{bmatrix} \hat{e}_1 & \hat{e}_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \hat{q}_1 & \hat{q}_2 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}
\end{aligned}
$$

$$
\begin{aligned}
&\Longrightarrow \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = Q \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \\
&\Longrightarrow \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = Q^{-1} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = Q^T \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}
\end{aligned}
$$

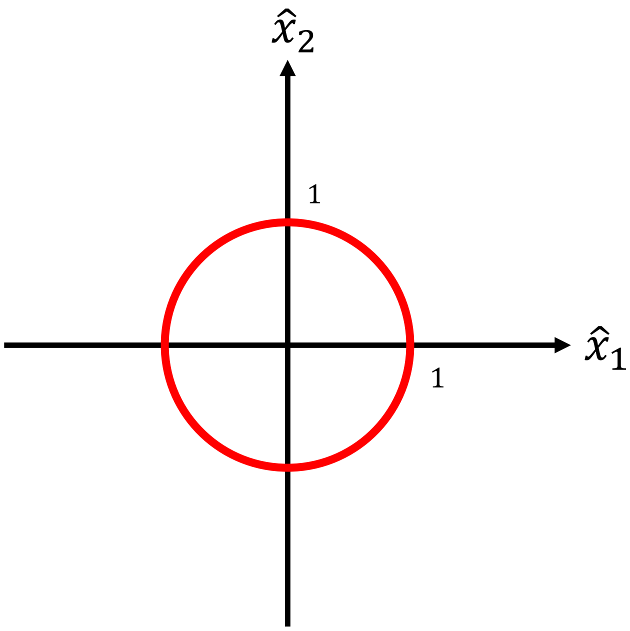
Coordinate Transformation

- Coordinate change to basis of $\{\hat{q}_1\ \hat{q}_2\}$

$$\underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{\text{coordinate in } I} \quad \xrightarrow{Q^T} \quad \underbrace{\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}}_{\text{coordinate in } Q}$$

Equation of an Ellipse

- Unit circle

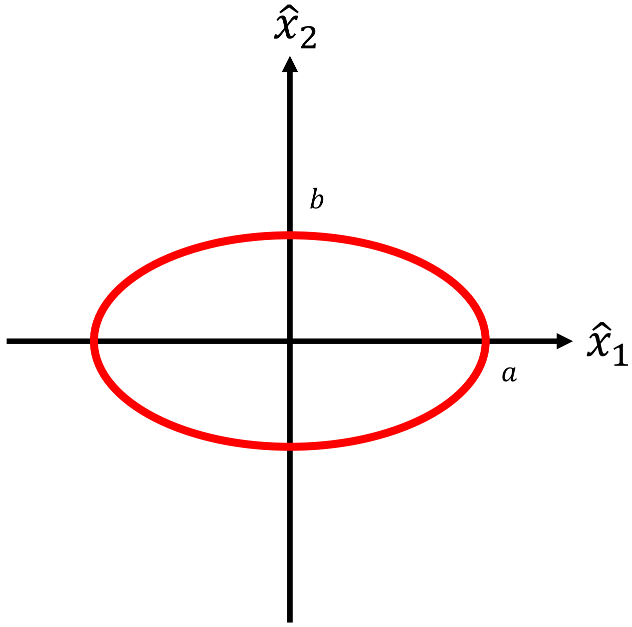


$$x_1^2 + x_2^2 = 1 \implies$$

$$\begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 1$$

Equation of an Ellipse

- Independent ellipse



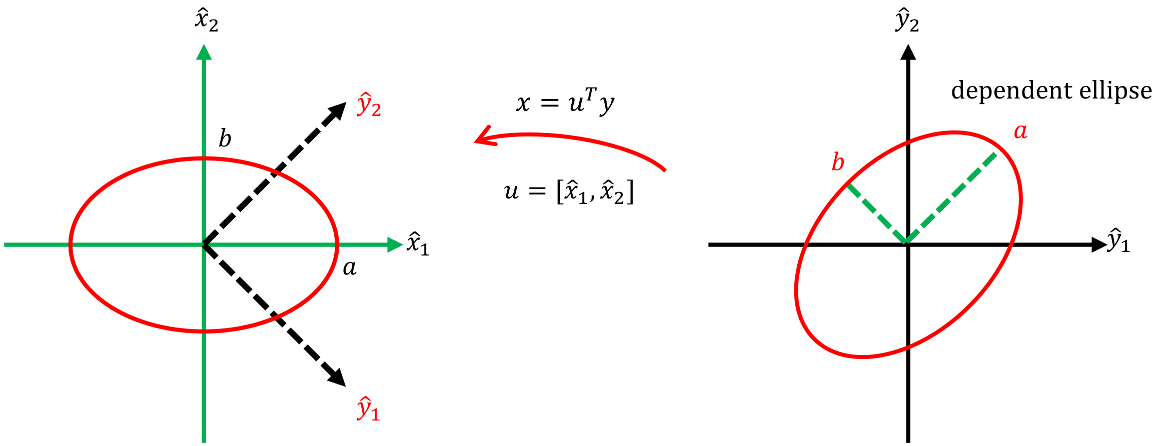
$$\frac{x_1^2}{a^2} + \frac{x_2^2}{b^2} = 1 \implies \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} \frac{1}{a^2} & 0 \\ 0 & \frac{1}{b^2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 1$$

$$\implies \begin{bmatrix} x_1 & x_2 \end{bmatrix} \Sigma_x^{-1} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 1$$

$$\text{where } \Sigma_x^{-1} = \begin{bmatrix} \frac{1}{a^2} & 0 \\ 0 & \frac{1}{b^2} \end{bmatrix}, \ \Sigma_x = \begin{bmatrix} a^2 & 0 \\ 0 & b^2 \end{bmatrix}$$

Equation of an Ellipse

- Dependent ellipse (Rotated ellipse)
  - Coordinate changes



$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = u^T \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}, \qquad \begin{aligned} x &= u^T y \\ ux &= y \end{aligned}$$

- Now we know in basis $\{\hat{x}_1, \hat{x}_2\} = I$

$$x^T \Sigma_x^{-1} x = 1 \quad \text{and} \quad \Sigma_x = \begin{bmatrix} a^2 & 0 \\ 0 & b^2 \end{bmatrix}$$

Equation of an Ellipse

- Then, we can find Σ_y such that

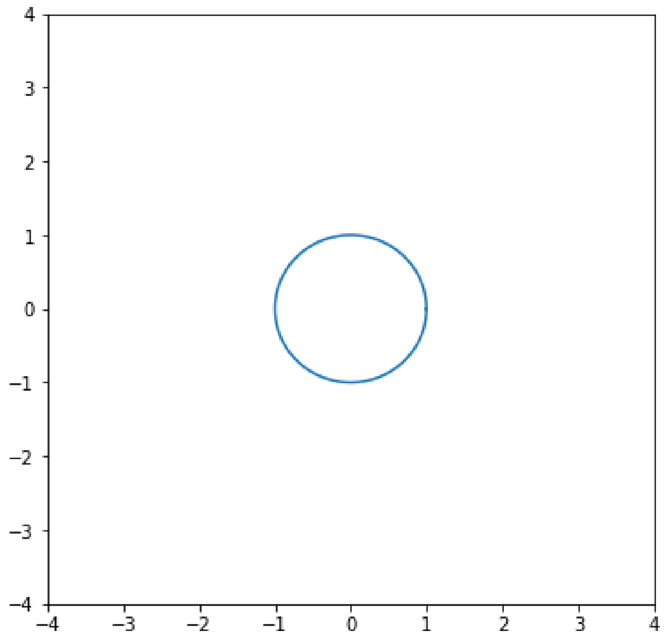
$$y^T \Sigma_y^{-1} y = 1 \quad \text{and} \quad \Sigma_y = ?$$
$$\Longrightarrow \; x^T \Sigma_x^{-1} x = y^T u \Sigma_x^{-1} u^T y = 1 \quad (\Sigma_y^{-1} : \text{similar matrix to } \Sigma_x^{-1})$$

$$\therefore \; \Sigma_y^{-1} = u \Sigma_x^{-1} u^T \quad \text{or}$$
$$\Sigma_y = u \Sigma_x u^T$$

Equation of an Ellipse (Python)

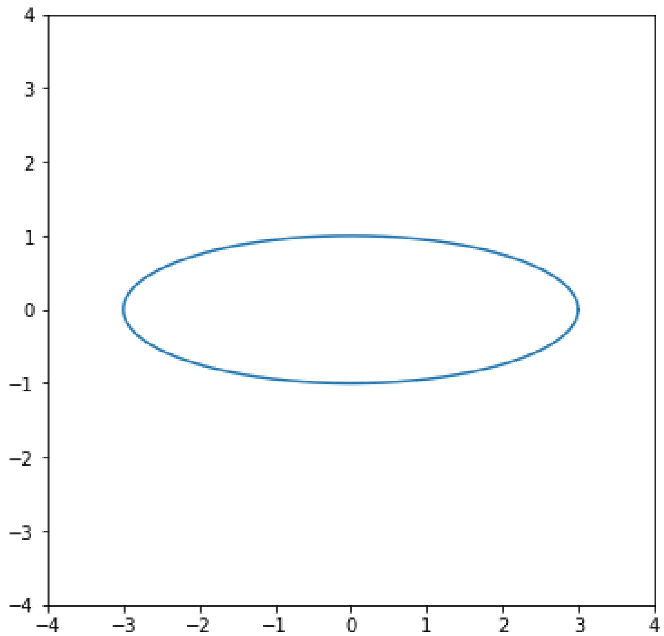
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

```
theta = np.arange(0,2*np.pi,0.01)
x1 = np.cos(theta)
x2 = np.sin(theta)

plt.figure(figsize=(6,6))
plt.plot(x1, x2)
plt.xlim([-4,4])
plt.ylim([-4,4])
plt.show()
```



Equation of an Ellipse (Python)

```
x1 = 3*np.cos(theta);
x2 = np.sin(theta);

plt.figure(figsize=(6,6))
plt.plot(x1, x2)
plt.xlim([-4,4])
plt.ylim([-4,4])
plt.show()
```

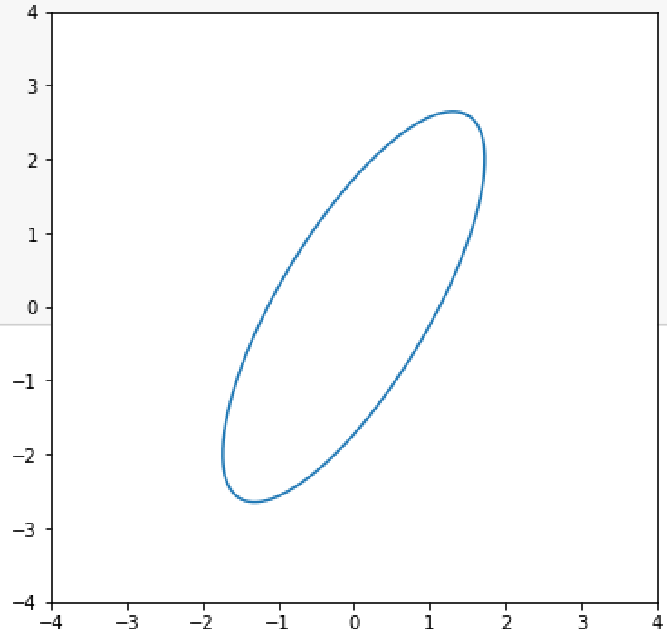


Equation of an Ellipse (Python)

$$u = [\hat{x}_1 \; \hat{x}_2] = \begin{bmatrix} \frac{1}{2} & -\frac{\sqrt{3}}{2} \\ \frac{\sqrt{3}}{2} & \frac{1}{2} \end{bmatrix}$$

```
theta = np.arange(0,2*np.pi,0.01)

u = np.array([[1/2, -np.sqrt(3)/2], [np.sqrt(3)/2, 1/2]])
X = np.array([x1, x2])

u = np.asmatrix(u)
X = np.asmatrix(X)
y = u*X

print(u)

plt.figure(figsize=(6,6))
plt.plot(y[0,:].T, y[1,:].T)
plt.xlim([-4,4])
plt.ylim([-4,4])
plt.show()
```

```
[[ 0.5        -0.8660254]
 [ 0.8660254  0.5       ]]
```



Equation of an Ellipse (Python)

```
Sx = np.array([[9, 0],[0, 1]])
Sx = np.asmatrix(Sx)

Sy = u*Sx*u.T

print(Sx)
print(Sy)
```

```
[[9 0]
 [0 1]]
[[ 3.          3.46410162]
 [ 3.46410162  7.        ]]
```

Question (Reverse Problem)

- Given Σ_y^{-1} (or Σ_y), how to find a (major axis) and b (minor axis) or
- How to find the proper matrix u

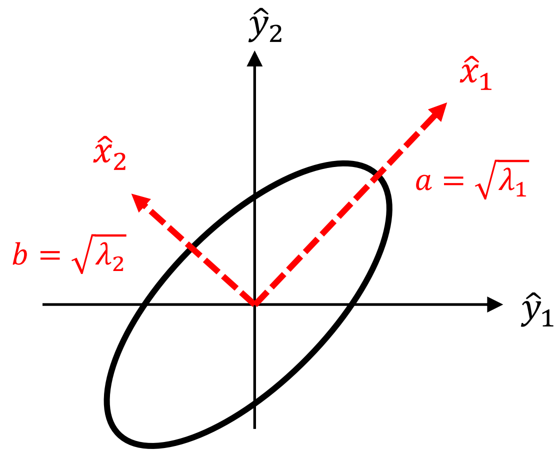
- Eigenvectors of Σ

$$A = S\Lambda S^T \qquad \text{where } S = [v_1 \ v_2] \text{ eigenvector of } A, \text{ and } \Lambda = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$$

$$\text{here, } \Sigma_y = u\Sigma_x u^T = u\Lambda u^T$$

$$\text{where } u = \begin{bmatrix} \hat{x}_1 & \hat{x}_2 \end{bmatrix} \text{ eigenvector of } \Sigma_y, \text{ and } \Lambda = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} = \begin{bmatrix} a^2 & 0 \\ 0 & b^2 \end{bmatrix}$$

Question (Reverse Problem)



$$\text{eigen-analysis} \begin{cases} \Sigma_y \hat{x}_1 = \lambda_1 \hat{x}_1 \\ \Sigma_y \hat{x}_2 = \lambda_2 \hat{x}_2 \end{cases} \implies \Sigma_y \underbrace{[\,\hat{x}_1 \quad \hat{x}_2\,]}_{u} = \underbrace{[\,\hat{x}_1 \quad \hat{x}_2\,]}_{u} \underbrace{\begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}}_{\Lambda}$$

$$\Sigma_y u = u\Lambda$$
$$\Sigma_y = u\Lambda u^T = u\Sigma_x u^T$$

Therefore

$$x = u^T y$$
$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = u^T \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

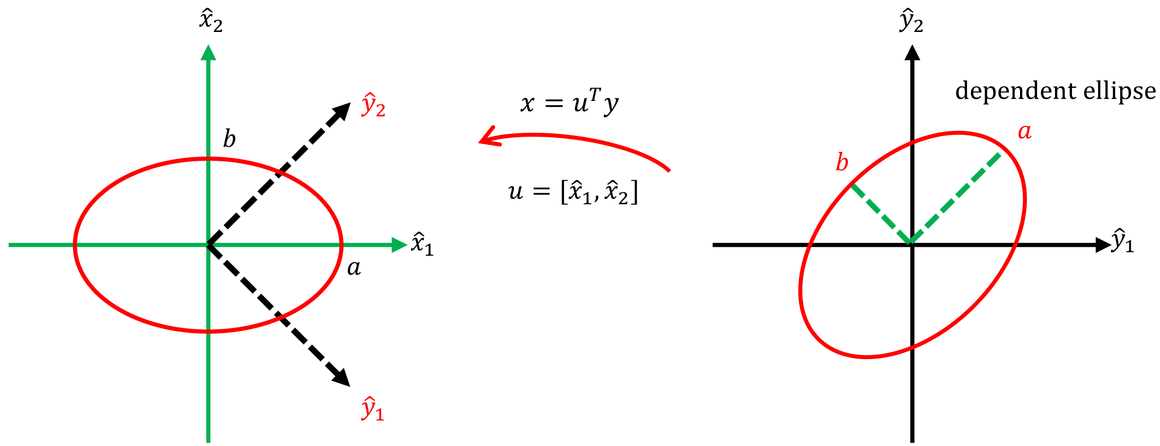
$$a = \sqrt{\lambda_1}$$
$$b = \sqrt{\lambda_2}$$
$$\text{major axis} = \hat{x}_1$$
$$\text{minor axis} = \hat{x}_2$$

Question (Reverse Problem)

```
D, U = np.linalg.eig(Sy)

idx = np.argsort(-D)
D = D[idx]
U = U[:,idx]

print(D)
print(np.diag(D))
print(U)
```

```
[ 9.  1.]
[[ 9.  0.]
 [ 0.  1.]]
[[-0.5        -0.8660254]
 [-0.8660254  0.5       ]]
```

Summary



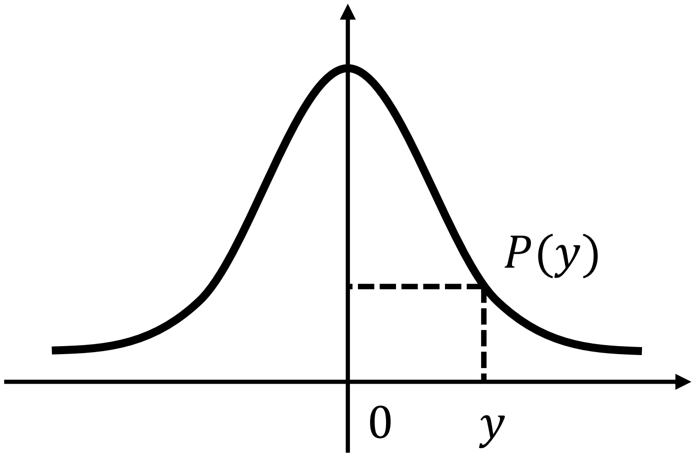
$$x = u^T y$$
$$u = \begin{bmatrix} \hat{x}_1 & \hat{x}_2 \end{bmatrix}$$

- Independent ellipse in $\{\hat{x}_1, \hat{x}_2\}$
- Dependent ellipse in $\{\hat{y}_1, \hat{y}_2\}$
- Decouple
  - diagonalize
  - eigen-analysis

Standard Univariate Normal Distribution

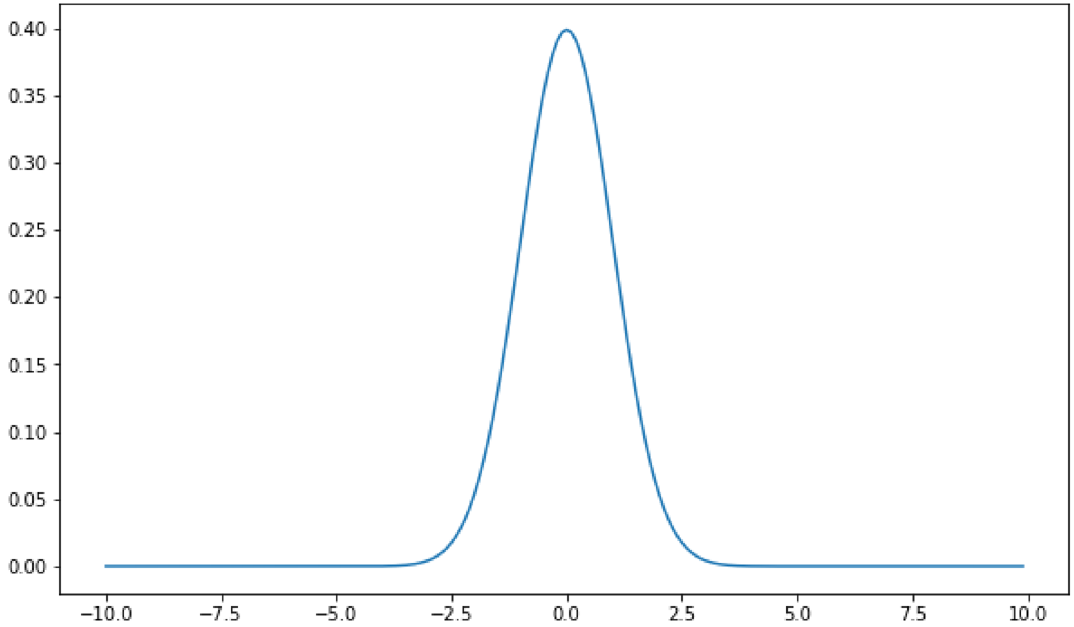
$$P_Y\left(Y=y\right)=\frac{1}{\sqrt{2\pi}}\exp\left(-\frac{1}{2}y^2\right)$$

$$\frac{1}{2}y^2=\text{const}\implies\text{prob. contour}$$



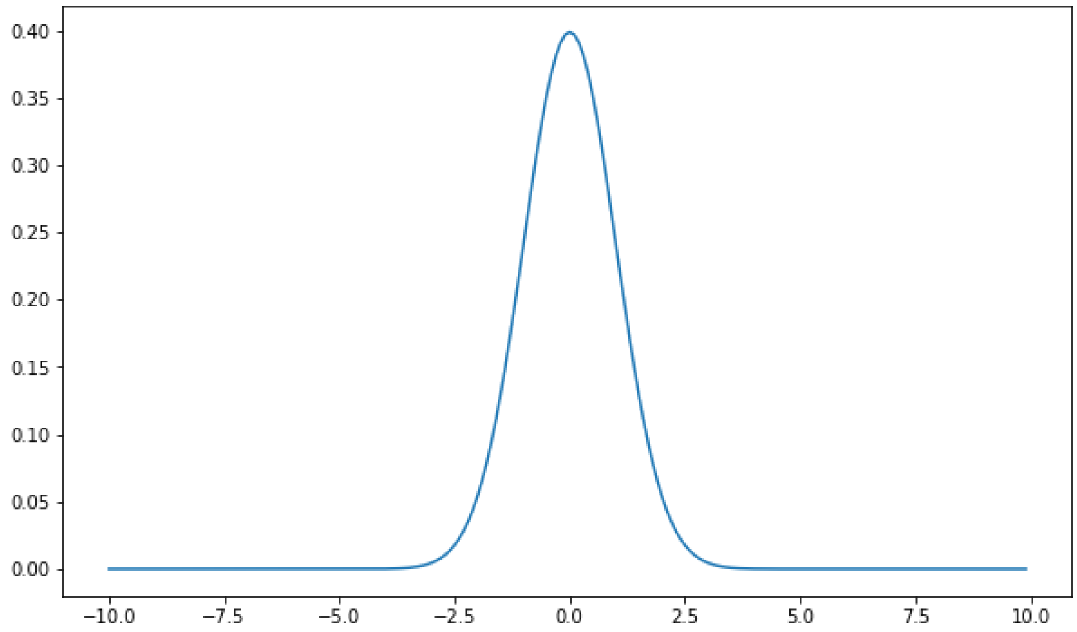
Standard Univariate Normal Distribution

```
y = np.arange(-10,10,0.1)

ProbG = 1/np.sqrt(2*np.pi)*np.exp(-1/2*y**2)

plt.figure(figsize=(10,6))
plt.plot(y, ProbG)
plt.show()
```



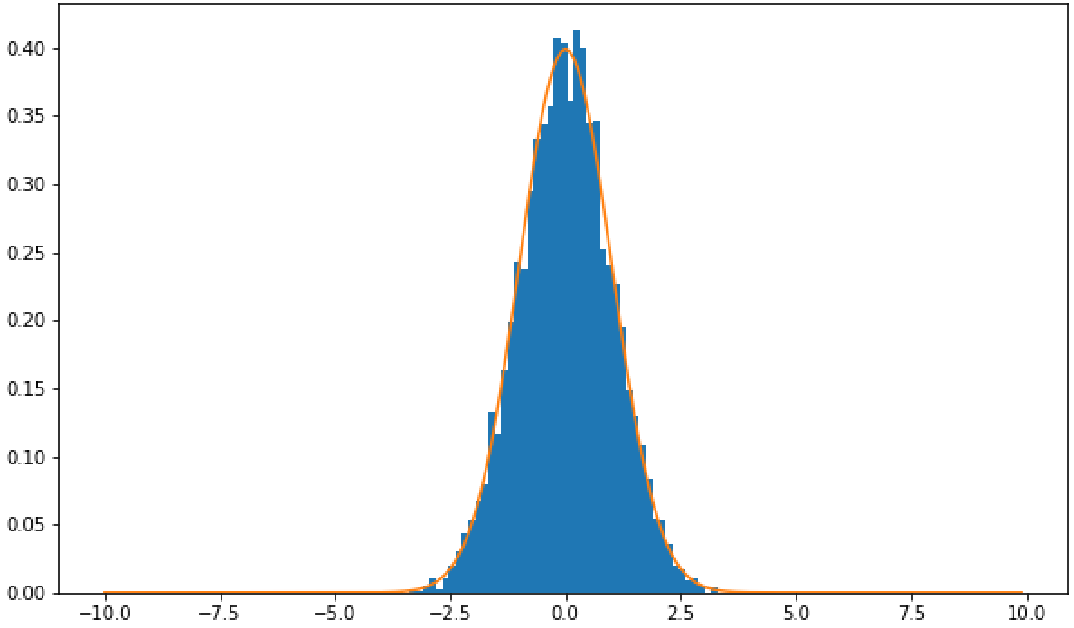
Standard Univariate Normal Distribution

```
from scipy.stats import norm

ProbG2 = norm.pdf(y)

plt.figure(figsize=(10,6))
plt.plot(y, ProbG2)
plt.show()
```



Standard Univariate Normal Distribution

```
x = np.random.randn(5000,1)

plt.figure(figsize=(10,6))
plt.hist(x, bins=51, normed=True)
plt.plot(y, ProbG2, label='G2')
plt.show()
```



Univariate Normal distribution

- Gaussian or normal distribution, 1D (mean μ, variance σ^2)

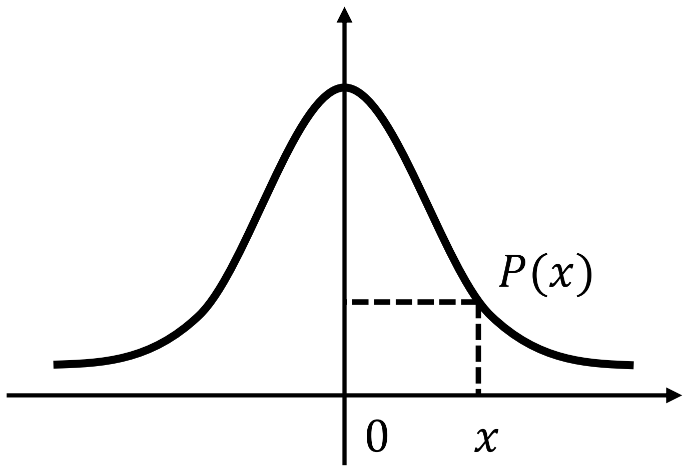
$$N(x;\ \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right)$$

$$x \sim N\left(\mu, \sigma^2\right)$$

$$\Longrightarrow \quad P_Y(y) = P_X(x), \quad y = \frac{x-\mu}{\sigma}$$

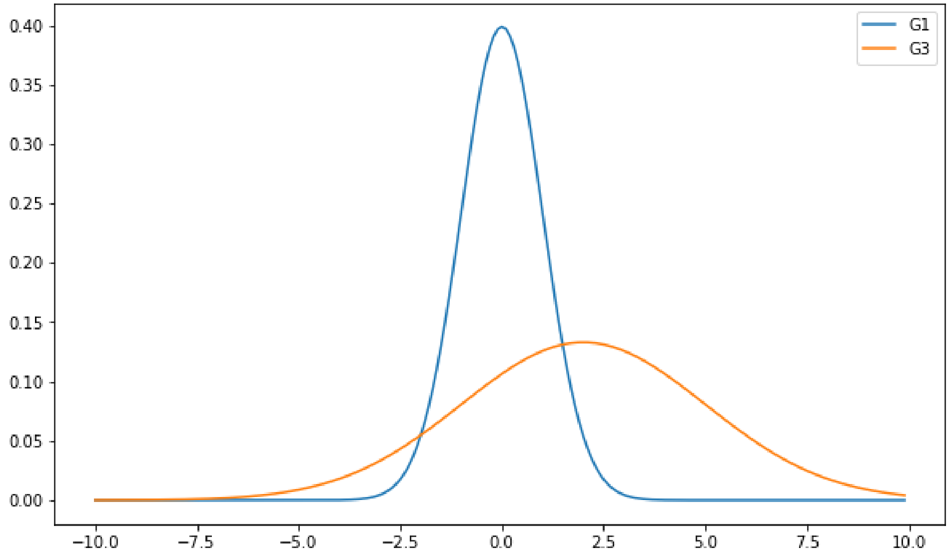
$$P_X(X = x) \sim \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$$

$$= \exp\left(-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right)$$



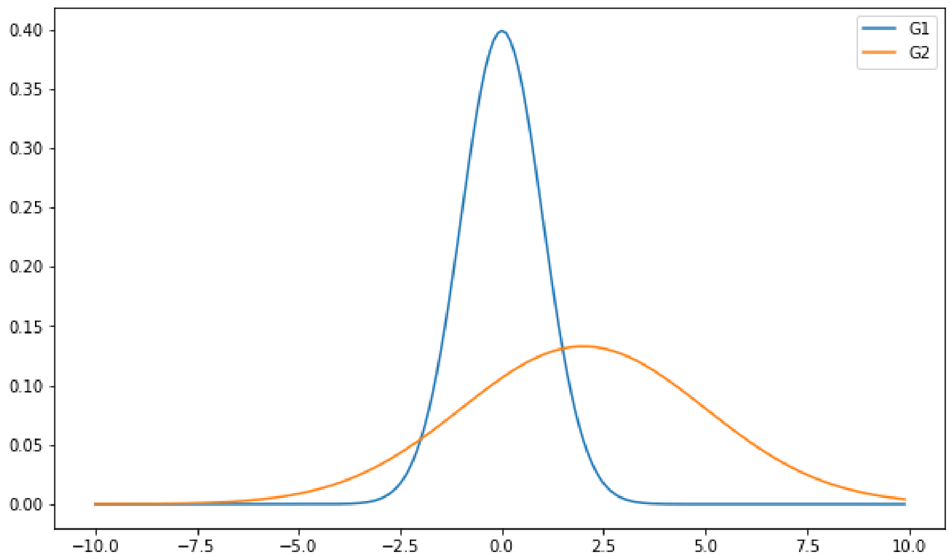
Univariate Normal distribution

```
mu = 2
sigma = 3

x = np.arange(-10, 10, 0.1)

ProbG3 = 1/(np.sqrt(2*np.pi)*sigma) * np.exp(-1/2*(x-mu)**2/(sigma**2))

plt.figure(figsize=(10,6))
plt.plot(y,ProbG, label='G1')
plt.plot(x,ProbG3, label='G3')
plt.legend()
plt.show()
```



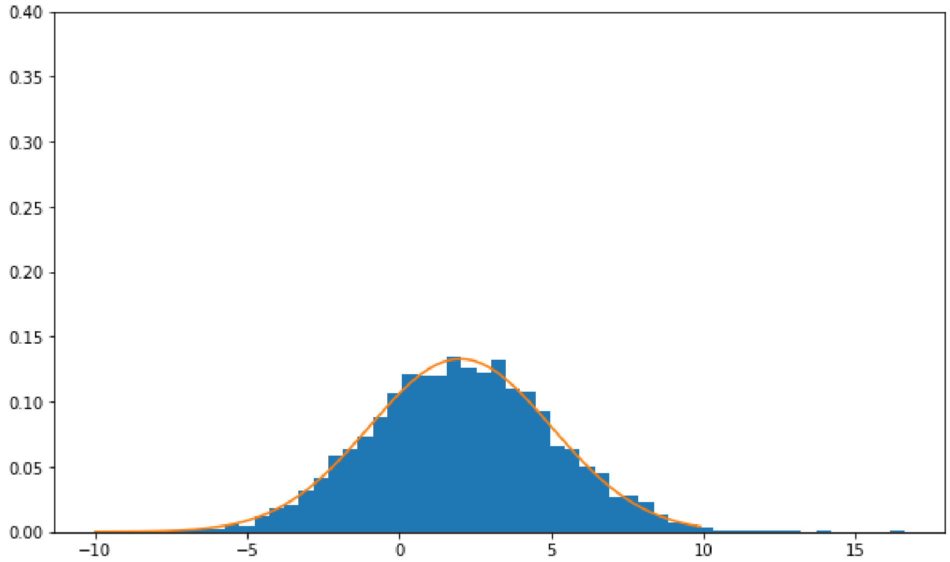
Univariate Normal distribution

```
ProbG2 = norm.pdf(x, 2, 3)

plt.figure(figsize=(10,6))
plt.plot(y,ProbG, label='G1')
plt.plot(x,ProbG2, label='G2')
plt.legend()
plt.show()
```



Univariate Normal distribution

```
x = mu + sigma*np.random.randn(5000,1)

plt.figure(figsize=(10,6))
plt.hist(x, bins=51, normed=True)
plt.plot(y,ProbG2, label='G2')
plt.ylim([0,0.4])
plt.show()
```



Multivariate Gaussian Models

- Similar to a univariate case, but in a matrix form

$$N(x;\,\mu,\Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}}\,|\Sigma|^{\frac{1}{2}}}\exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}\,(x-\mu)\right)$$

$$\begin{aligned}
\mu &= \text{length } n \text{ column vector} \\
\Sigma &= n \times n \text{ matrix (covariance matrix)} \\
|\Sigma| &= \text{matrix determinant}
\end{aligned}$$

- Multivariate Gaussian models and ellipse
  - Ellipse shows constant Δ^2 value…

$$\Delta^2 = (x-\mu)^T\Sigma^{-1}(x-\mu)$$

Two Independent Variables

$$P(X_1 = x_1, X_2 = x_2) = P_{X_1}(x_1)\, P_{X_2}(x_2)$$

$$\sim \exp\left(-\frac{1}{2}\frac{\left(x_1 - \mu_{x_1}\right)^2}{\sigma_{x_1}^2}\right) \cdot \exp\left(-\frac{1}{2}\frac{\left(x_2 - \mu_{x_2}\right)^2}{\sigma_{x_2}^2}\right)$$

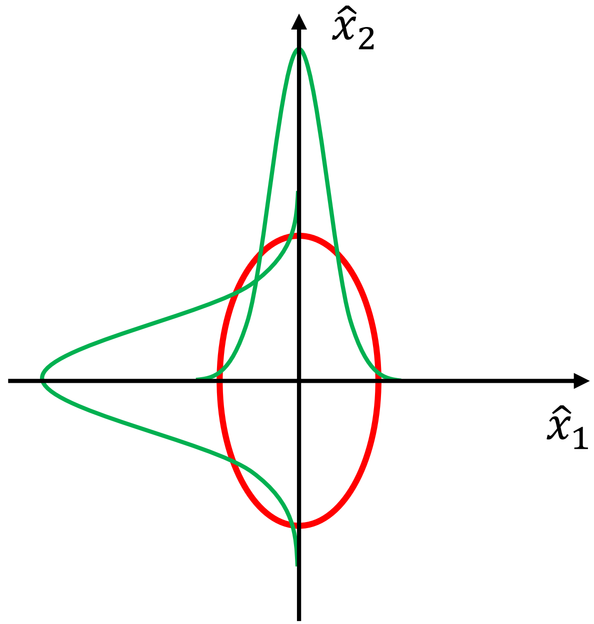
$$\sim \exp\left(-\frac{1}{2}\left(\frac{x_1^2}{\sigma_{x_1}^2} + \frac{x_2^2}{\sigma_{x_2}^2}\right)\right)$$

- In a matrix form

$$P(x_1) \cdot P(x_2) = \frac{1}{Z_1 Z_2} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)$$

$$\left(x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad \mu = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \quad \Sigma = \begin{bmatrix} \sigma_{x_1}^2 & 0 \\ 0 & \sigma_{x_2}^2 \end{bmatrix}\right)$$

Two Independent Variables



$$\frac{x_1^2}{\sigma_{x_1}^2} + \frac{x_2^2}{\sigma_{x_2}^2} = c \quad \text{(ellipse)}$$

$$\begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} \frac{1}{\sigma_{x_1}^2} & 0 \\ 0 & \frac{1}{\sigma_{x_2}^2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = c \qquad (\sigma_{x_1} < \sigma_{x_2})$$

- Summary in a matrix form

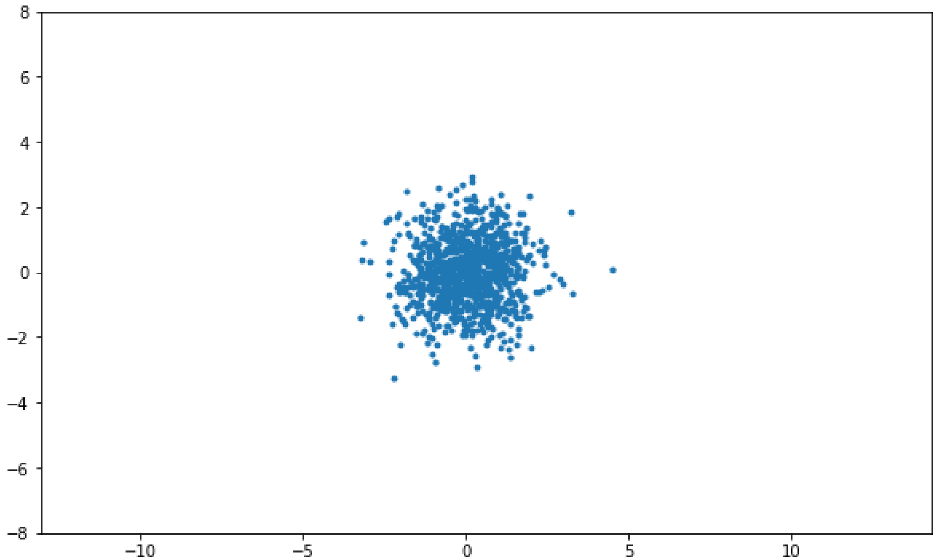
$$N(0, \Sigma_x) \sim \exp\left(-\frac{1}{2} x^T \Sigma_x^{-1} x\right)$$

$$N(\mu_x, \Sigma_x) \sim \exp\left(-\frac{1}{2} (x - \mu_x)^T \Sigma_x^{-1} (x - \mu_x)\right)$$

Two Independent Variables

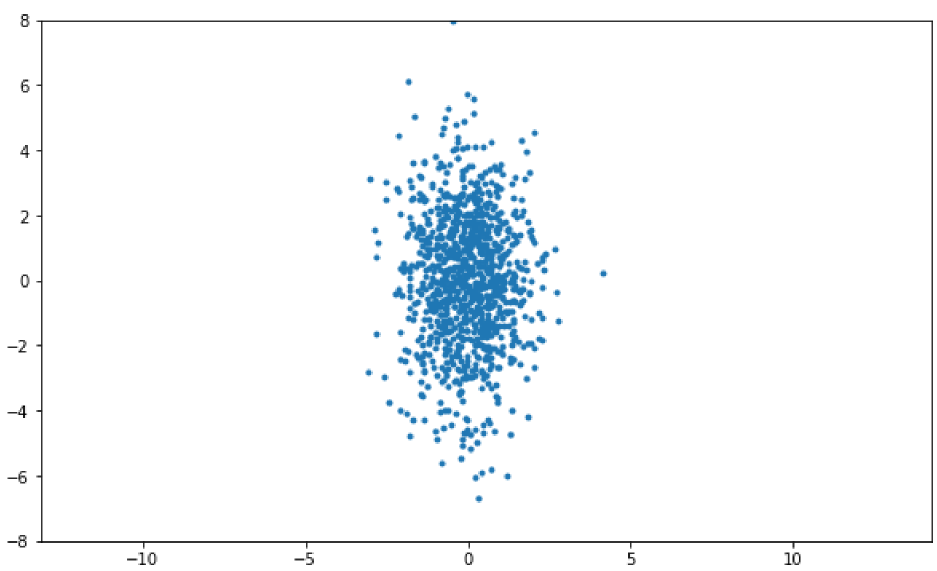
```
mu = np.array([0, 0])
sigma = np.eye(2)

m = 1000
x = np.random.multivariate_normal(mu, sigma, m)
print(x.shape)

plt.figure(figsize=(10,6))
plt.plot(x[:,0], x[:,1], '.')
plt.axis('equal')
plt.ylim([-8,8])
plt.show()
```

```
(1000, 2)
```

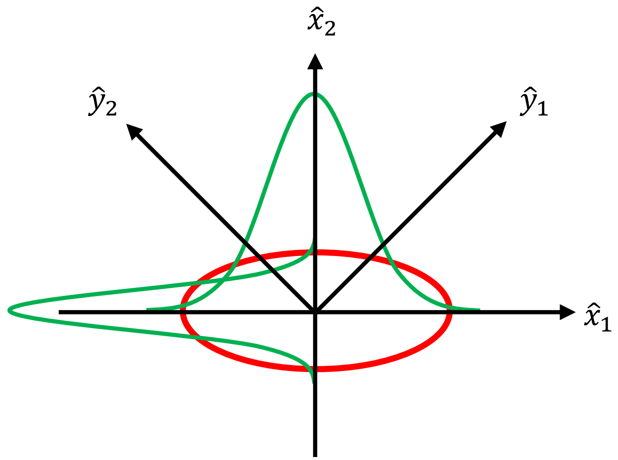


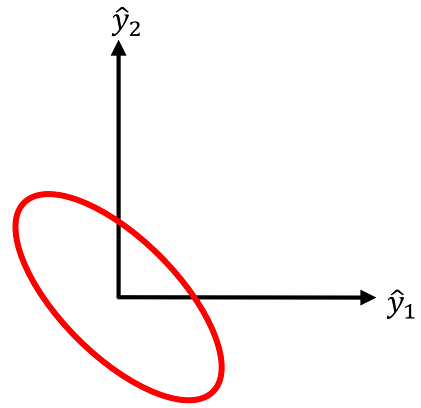
Two Independent Variables

```
mu = np.array([0, 0])
sigma = np.array([[1, 0], [0, 4]])

m = 1000
x = np.random.multivariate_normal(mu, sigma, m)

plt.figure(figsize=(10,6))
plt.plot(x[:,0], x[:,1], '.')
plt.axis('equal')
plt.ylim([-8,8])
plt.show()
```



Two Dependent Variables in $\{y_1, y_2\}$





- Compute $P_Y(y)$ from $P_X(x)$

$$P_X(x) = P_Y(y) \;\; \text{where} \;\; x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

- Relationship between y and x

$$x = \begin{bmatrix} \hat{x}_1 & \hat{x}_2 \end{bmatrix}^T y = u^T y$$

Two Dependent Variables in $\{y_1, y_2\}$

$$x^T \Sigma_x^{-1} x = y^T u \Sigma_x^{-1} u^T y = y^T \Sigma_y^{-1} y$$

$$\therefore \quad \Sigma_y^{-1} = u \Sigma_x^{-1} u^T$$

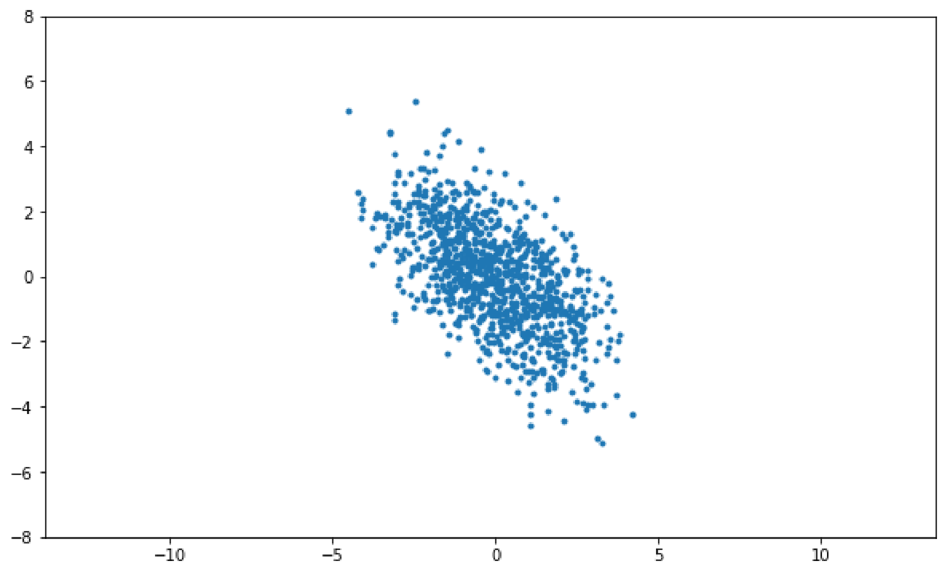
$$\rightarrow \Sigma_y = u \Sigma_x u^T$$

- Σ_x : covariance matrix of x
- Σ_y : covariance matrix of y

- If u is an eigenvector matrix of Σ_y, then Σ_x is a diagonal matrix

Two Dependent Variables in $\{y_1, y_2\}$

```
mu = np.array([0, 0])
sigma = 1./2.*np.array([[5, -3], [-3, 5]])

m = 1000
x = np.random.multivariate_normal(mu, sigma, m)

plt.figure(figsize=(10,6))
plt.plot(x[:,0], x[:,1], '.')
plt.axis('equal')
plt.ylim([-8,8])
plt.show()
```



Two Dependent Variables in $\{y_1, y_2\}$

- Remark

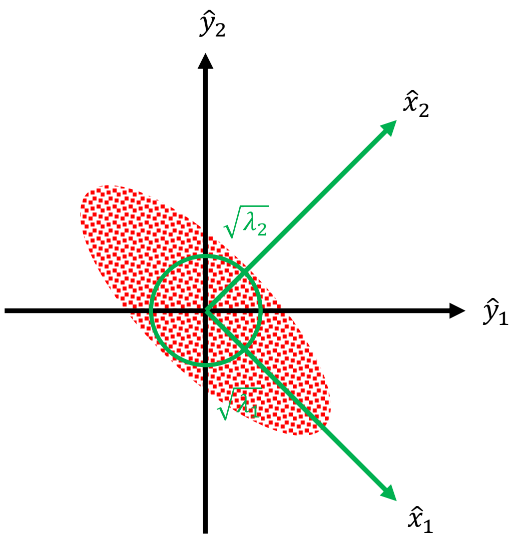
$$x \sim N(\mu_x, \Sigma_x) \text{ and } y = Ax + b \text{ affine transformation}$$

$$\implies y \sim N(\mu_y, \Sigma_y) = N(A\mu_x + b, A\Sigma_x A^T)$$

$$\implies y \text{ is also Gaussian with } \mu_y = Ax + b, \quad \Sigma_y = A\Sigma_x A^T$$

Decouple using Covariance Matrix

- Given data, how to find Σ_y and major (or minor) axis (assume $\mu_y = 0$)



$$\Sigma_y = \begin{bmatrix} \mathrm{var}(y_1) & \mathrm{cov}(y_1, y_2) \\ \mathrm{cov}(y_2, y_1) & \mathrm{var}(y_2) \end{bmatrix}$$

eigen-analysis

$$\Sigma_y \hat{x}_1 = \lambda_1 \hat{x}_1$$
$$\Sigma_y \hat{x}_2 = \lambda_2 \hat{x}_2$$

$$\Sigma_x^{-1} = \begin{bmatrix} \frac{1}{\sqrt{\lambda_1}^2} & 0 \\ 0 & \frac{1}{\sqrt{\lambda_2}^2} \end{bmatrix}$$

$$\Sigma_x = \begin{bmatrix} \sqrt{\lambda_1}^2 & 0 \\ 0 & \sqrt{\lambda_2}^2 \end{bmatrix}$$

$$\Sigma_y \begin{bmatrix} \hat{x}_1 & \hat{x}_2 \end{bmatrix} = \begin{bmatrix} \hat{x}_1 & \hat{x}_2 \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$$
$$= \begin{bmatrix} \hat{x}_1 & \hat{x}_2 \end{bmatrix} \Sigma_x$$

$$y = ux \implies u^T y = x$$
$$\begin{bmatrix} \hat{x}_1 & \hat{x}_2 \end{bmatrix} = u$$

$$\Sigma_y = u \Sigma_x u^T$$

Decouple using Covariance Matrix

```
x.shape

S = np.cov(x.T)
print ("S = \n", S)
```

```
S =
 [[ 2.59216411 -1.54924881]
 [-1.54924881  2.54567035]]
```

```
D, U = np.linalg.eig(S)

idx = np.argsort(-D)
D = D[idx]
U = U[:,idx]

print ("U = \n", U)
print ("D = \n", D)
```

```
U =
 [[ 0.7123916   0.70178217]
 [-0.70178217  0.7123916 ]]
D =
 [ 4.11834045  1.01949402]
```

Decouple using Covariance Matrix

```
xp = np.arange(-10, 10)

plt.figure(figsize=(10,6))
plt.plot(x[:,0],x[:,1],'.')
plt.plot(xp, U[1,0]/U[0,0]*xp, label='u1')
plt.plot(xp, U[1,1]/U[0,1]*xp, label='u2')
plt.axis('equal')
plt.ylim([-8, 8])
plt.legend()
plt.show()
```