Installation des librairies

In [1]:

```
pip install numpy
```

```
Requirement already satisfied: numpy in c:\users\bette\anaconda3\lib\site-packages (1.26.
4)
Note: you may need to restart the kernel to use updated packages.
```

In [2]:

```
pip install -U matplotlib
```

```
Requirement already satisfied: matplotlib in c:\users\bette\anaconda3\lib\site-packages (
3.9.2)
Collecting matplotlib
  Downloading matplotlib-3.10.1-cp312-cp312-win_amd64.whl.metadata (11 kB)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\bette\anaconda3\lib\site-pack
ages (from matplotlib) (1.2.0)
Requirement already satisfied: cycler>=0.10 in c:\users\bette\anaconda3\lib\site-packages
(from matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\bette\anaconda3\lib\site-pac
kages (from matplotlib) (4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\bette\anaconda3\lib\site-pac
kages (from matplotlib) (1.4.4)
Requirement already satisfied: numpy>=1.23 in c:\users\bette\anaconda3\lib\site-packages
(from matplotlib) (1.26.4)
Requirement already satisfied: packaging>=20.0 in c:\users\bette\anaconda3\lib\site-packa
ges (from matplotlib) (24.1)
Requirement already satisfied: pillow>=8 in c:\users\bette\anaconda3\lib\site-packages (f
rom matplotlib) (10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\bette\anaconda3\lib\site-pack
ages (from matplotlib) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\bette\anaconda3\lib\site-
packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in c:\users\bette\anaconda3\lib\site-packages (fr
om python-dateutil>=2.7->matplotlib) (1.16.0)
Downloading matplotlib-3.10.1-cp312-cp312-win_amd64.whl (8.1 MB)
   ---------------------------------------- 0.0/8.1 MB ? eta -:--:--
   ----------- ---------------------------- 2.4/8.1 MB 12.2 MB/s eta 0:00:01
   ---------------------------- ----------- 5.8/8.1 MB 14.1 MB/s eta 0:00:01
   ---------------------------------------- 8.1/8.1 MB 13.1 MB/s eta 0:00:00
Installing collected packages: matplotlib
  Attempting uninstall: matplotlib
    Found existing installation: matplotlib 3.9.2
    Uninstalling matplotlib-3.9.2:
      Successfully uninstalled matplotlib-3.9.2
Successfully installed matplotlib-3.10.1
Note: you may need to restart the kernel to use updated packages.
```

In [3]:

```
pip install scikit-learn
```

```
Requirement already satisfied: scikit-learn in c:\users\bette\anaconda3\lib\site-packages
(1.5.1)
Requirement already satisfied: numpy>=1.19.5 in c:\users\bette\anaconda3\lib\site-package
s (from scikit-learn) (1.26.4)
Requirement already satisfied: scipy>=1.6.0 in c:\users\bette\anaconda3\lib\site-packages
(from scikit-learn) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in c:\users\bette\anaconda3\lib\site-package
s (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\bette\anaconda3\lib\site-
packages (from scikit-learn) (3.5.0)
Note: you may need to restart the kernel to use updated packages.
```

In [29]:

In [29]:

```
pip install pandoc
```

```
Collecting pandoc
  Downloading pandoc-2.4.tar.gz (34 kB)
  Preparing metadata (setup.py): started
  Preparing metadata (setup.py): finished with status 'done'
Collecting plumbum (from pandoc)
  Downloading plumbum-1.9.0-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: ply in c:\users\bette\anaconda3\lib\site-packages (from pa
ndoc) (3.11)
Requirement already satisfied: pywin32 in c:\users\bette\anaconda3\lib\site-packages (fro
m plumbum->pandoc) (305.1)
Downloading plumbum-1.9.0-py3-none-any.whl (127 kB)
Building wheels for collected packages: pandoc
  Building wheel for pandoc (setup.py): started
  Building wheel for pandoc (setup.py): finished with status 'done'
  Created wheel for pandoc: filename=pandoc-2.4-py3-none-any.whl size=34821 sha256=d0b9a8
c70710b0e9bd485d50e31f30a4bcd1d60346581baf19dc64a6f7b4a64f
  Stored in directory: c:\users\bette\appdata\local\pip\cache\wheels\9c\2f\9f\b1aac8c3e74
b4ee327dc8c6eac5128996f9eadf586e2c0ba67
Successfully built pandoc
Installing collected packages: plumbum, pandoc
Successfully installed pandoc-2.4 plumbum-1.9.0
Note: you may need to restart the kernel to use updated packages.
```

Chargement des librairies

In [1]:

```python
import numpy as np
import random
import math as m
import matplotlib.pyplot as plt
from sklearn.metrics import precision_score, recall_score
```

Modèle de régression cible

In [2]:

```python
#conception of the target model
Nt=20
D=4
sigma_t=0.5
betat0=-1
betat1=-1.8
betat2=1.2
betat3=1
X_t=np.ones((Nt,D))
x_t=np.ones((Nt,1))
random.seed(a=256, version=2)
for i in range(0,Nt):
    x_t[i,0]=-4*random.random()+1
for j in range(1,D):
    for i in range(0,Nt):
        if j==1:
            X_t[i,j]=x_t[i,0]
        elif j==2:
            X_t[i,j]=(x_t[i,0])**2
        else:
            X_t[i,j]=(x_t[i,0])**3
big_sigma_t=np.matmul(np.transpose(X_t),X_t)
random.seed(a=255, version=2)
epsilon_t=np.ones((Nt,1))
for i in range(Nt):
    epsilon_t[i,0]=random.normalvariate(mu=0.0, sigma=m.sqrt(sigma_t))
beta_t=np.array([[betat0],[betat1],[betat2],[betat3]])
Y_t=X_t@beta_t+epsilon_t
```

Modèle de régression source

In [3]:

```
#conception of the source model
Ns=100
D=4
sigma_s=0.5
X_s=np.ones((Ns,D))
x_s=np.ones((Ns,1))
random.seed(a=254, version=2)
for i in range(0,Ns):
    x_s[i,0]=3*random.random()
for j in range(1,D):
    for i in range(0,Ns):
        if j==1:
            X_s[i,j]=x_s[i,0]
        elif j==2:
            X_s[i,j]=(x_s[i,0])**2
        else:
            X_s[i,j]=(x_s[i,0])**3
big_sigma_s=np.matmul(np.transpose(X_s),X_s)
random.seed(a=253, version=2)
epsilon_s=np.ones((Ns,1))
for i in range(Ns):
    epsilon_s[i,0]=random.normalvariate(mu=0.0, sigma=m.sqrt(sigma_s))

random.seed(a=252, version=2)
beta_s=np.array([[betat0 + random.normalvariate(mu=0.0, sigma=0.3)],[betat1 +random.norm
alvariate(mu=0.0, sigma=0.3) ],[betat2 + random.normalvariate(mu=0.0, sigma=0.3) ],[beta
t3 + random.normalvariate(mu=0.0,sigma=0.3) ]])
Y_s=X_s@beta_s+epsilon_s
```

Choix de k par maximisation de $\bar{U}_k$

In [ ]:

```
#separate the data betwenn , train set and test set
X_t_train=X_t[0:10,:]
Y_t_train=Y_t[0:10]
Nt_train=np.shape(X_t_train)[0]
X_t_test=X_t[10:21,:]
Y_t_test=Y_t[10:21]
Nt_test=np.shape(X_t_test)[0]
big_sigma_t_train=np.matmul(np.transpose(X_t_train),X_t_train)
##choice of alpha
alpha_etoile=2/( np.max(np.linalg.eigvals(big_sigma_t_train)) + np.min(np.linalg.eigvals
(big_sigma_t_train)) )
alpha=alpha_etoile/2
#maximization of Uk_barre
L=[]
Uk=np.ones(Ns+Nt_train)
A=np.eye(D) - alpha*big_sigma_t_train
def matrix_power_diagonalization_stable(A, k):
    #Obtenir les valeurs propres et les vecteurs propres de A
    eigenvalues, eigenvectors = np.linalg.eig(A)
    #Calculer la matrice diagonale des valeurs propres élevées à la puissance k
    D_k = np.diag(np.exp(k*np.log(eigenvalues)))
    # Calculer A^k = P D^k P^-1
    A_k = eigenvectors @ D_k @ np.linalg.inv(eigenvectors)
    return A_k
for k in range(1500):
    A_k=matrix_power_diagonalization_stable(A,k)
    omegak=(1/alpha)*np.linalg.inv(big_sigma_t_train)@(np.eye(D) - A_k)
    Vk=(sigma_s**2)*(A_k)@(np.linalg.inv(big_sigma_s))@(A_k) +(sigma_t**2)*(alpha**2)*(o
megak)@(big_sigma_t_train)@(omegak)
    for i in range(Ns):
        Uk[i]=(sigma_t**2)*np.transpose(X_s[i,:])@np.linalg.inv(big_sigma_t_train)@X_s[i
,:]*m.log( (np.transpose(X_s[i,:])@Vk@X_s[i,:])/ ( (sigma_t**2)*np.transpose(X_s[i,:])@n
```

```
p.linalg.inv(big_sigma_t_train)@X_s[i,:] ))
    for i in range(Ns,Ns+Nt_train):
        Uk[i]=(sigma_t**2)*np.transpose(X_t_train[i-Ns,:])@np.linalg.inv(big_sigma_t_tra
in)@X_t_train[i-Ns,:]*m.log( (np.transpose(X_t_train[i-Ns,:])@Vk@X_t_train[i-Ns,:]) / (
(sigma_t**2)*np.transpose(X_t_train[i-Ns,:])@np.linalg.inv(big_sigma_t_train)@X_t_train[
i-Ns,:]) )
    Uk_barre=0
    for i in range(len(Uk)):
        Uk_barre+=(1/(Ns+Nt_train))*Uk[i]
    L.append(Uk_barre)
        #Uk_barre=np.mean(Uk)
        #Uk_barre=np.mean(Uk)
        #L.append(Uk_barre)
print(L.index(max(L)))

L.pop(0)
plt.plot(L,label='Uk_barre')
#plt.axvline(x=1000,color='r',label='x = 200',linestyle='--')
#plt.axvline(x=180,color='r',label='x = 200',linestyle='--')
#plt.title('Uk_barre')
plt.xlabel('iterations de Gradient k')
plt.ylabel('Uk')
plt.legend()
```

Choix de k par maximisation de l'erreur LOOCV (leave-one-out cross validation)

In [12]:

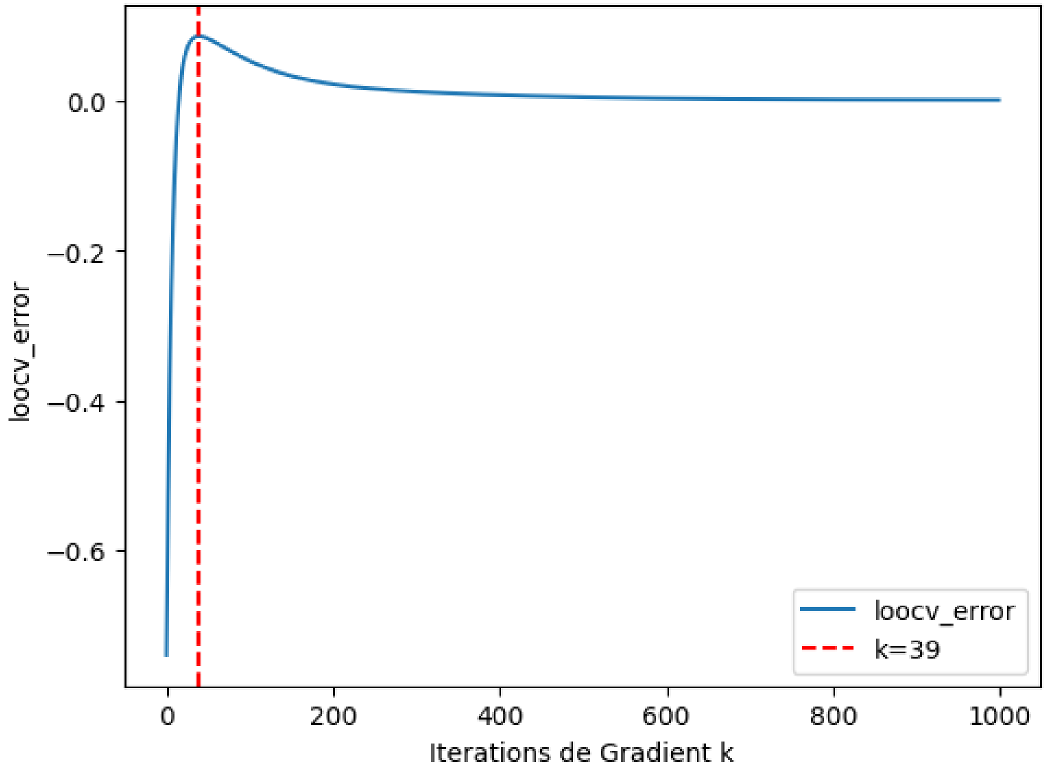
```
#maximizing of the LOOCV
Beta_t_chap_remove=np.ones((Nt_train,D))
for i in range(Nt_train):
    beta_t_chap_remove=np.ones(D)
    X_t_train_remove=np.delete(X_t_train, (i), axis=0)
    Y_t_train_remove=np.delete(Y_t_train, (i), axis=0)
    beta_t_chap_remove=np.linalg.inv( np.transpose(X_t_train_remove)@X_t_train_remove )@
( np.transpose(X_t_train_remove)@Y_t_train_remove )
    Beta_t_chap_remove[i,:]=np.transpose(beta_t_chap_remove) # ou alors onutilise.squeeze
()

Y_t_chap_cross_val=np.ones(Nt_train)
for i in range(Nt_train):
    Y_t_chap_cross_val[i]=np.transpose(X_t_train[i,:])@Beta_t_chap_remove[i,:]

Beta_s_chap_remove=np.ones((Nt_train,D))
for i in range(Nt_train):
    beta_s_chap_remove=np.ones((D))
    X_s_remove=np.delete(X_s, (i), axis=0)
    Y_s_remove=np.delete(Y_s, (i), axis=0)
    beta_s_chap_remove=np.linalg.inv( np.transpose(X_s_remove)@X_s_remove )@(np.transpos
e(X_s_remove)@Y_s_remove )
    Beta_s_chap_remove[i,:]=np.transpose(beta_s_chap_remove)

P=[]
S=[]
for k in range(1000):
    A_k=matrix_power_diagonalization_stable(A,k)
    Beta_k_chap_remove=np.ones((Nt_train,D))
    Y_k_chap_cross_val=np.ones(Nt_train)
    for i in range(Nt_train):
        beta_k_chap_remove=(A_k)@Beta_s_chap_remove[i,:] + ( np.eye(D)- A_k)@Beta_t_chap
_remove[i,:]
        Beta_k_chap_remove[i,:]=beta_k_chap_remove
        Y_k_chap_cross_val[i]=np.transpose( X_t_train[i,:])@Beta_k_chap_remove[i,:]
    loocv_error_k=0
    for i in range(Nt_train):
        original_error=(Y_t_train[i]- Y_t_chap_cross_val[i])**2
        adjusted_error=(Y_t_train[i]-Y_k_chap_cross_val[i])**2
        loocv_error_k+=(1/Nt_train)*(original_error-adjusted_error)
    #print(loocv_error_k[0])
    P.append(loocv_error_k[0])
```

```python
P.pop(0)
print(P.index(max(P)))
plt.plot(P,label='loocv_error')
plt.axvline(x=39,color='r',label='k=39',linestyle='--')
#plt.title('LOOCV_ERROR')
plt.xlabel('Iterations de Gradient k')
plt.ylabel('loocv_error')
plt.legend()
```

39

Out[12]:

<matplotlib.legend.Legend at 0x2110e6b0440>



Tracé du polynome cible P_t et ses estimations

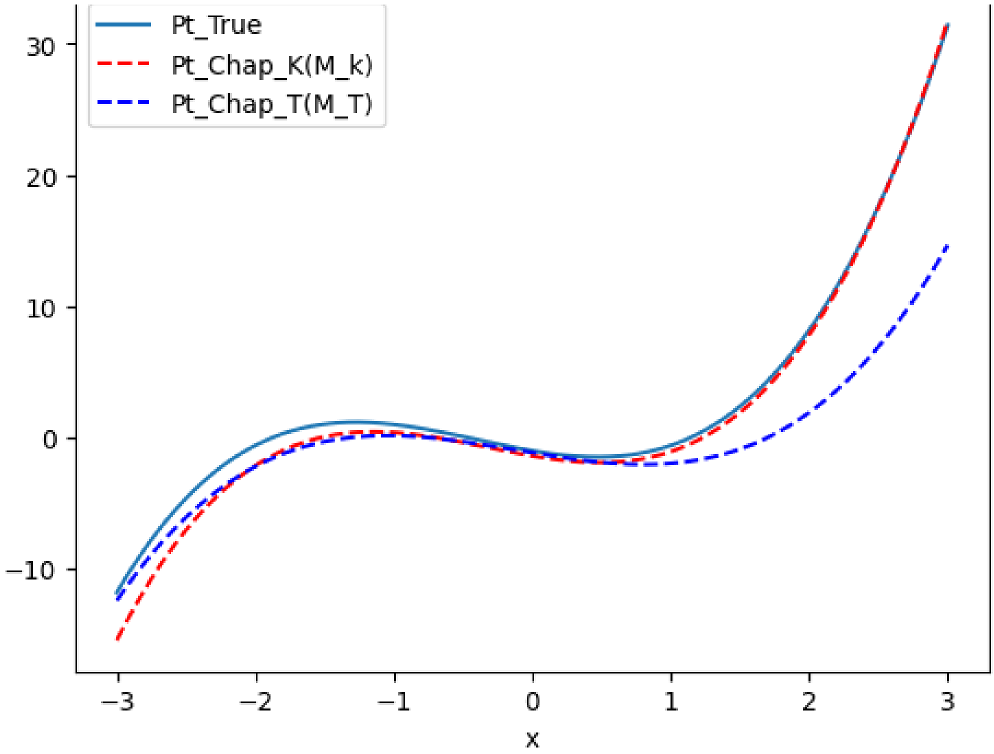
In [16]:

```python
#Plot of the true polynomial Pt and its estimations P_t_chap and P_k_chap
k=P.index(max(P))
A_k = matrix_power_diagonalization_stable(A, k)
beta_t_chap=np.linalg.inv( np.transpose(X_t_train)@X_t_train )@( np.transpose(X_t_train)
@Y_t_train )
beta_s_chap=np.linalg.inv( np.transpose(X_s)@X_s )@( np.transpose(X_s)@Y_s )
beta_k_chap=A_k@beta_s_chap + (np.eye(D) - A_k)@beta_t_chap
x = np.linspace(-3, 3,800 )
Pt_true = x**3 + (1.2)*x**2 + (-1.8)*x -1
Pt_chap=(beta_t_chap[3])*x**3 + (beta_t_chap[2])*x**2 + (beta_t_chap[1])*x +beta_t_chap[
0]
Pt_chap_k=(beta_k_chap[3])*x**3 + (beta_k_chap[2])*x**2 + (beta_k_chap[1])*x +beta_k_cha
p[0]
plt.title('Figure 1 : Polynome_cible_réel_PT_et_ses_estimations_M_T_&_M_k')
plt.plot(x, Pt_true, label='Pt_True')
plt.plot(x, Pt_chap_k, color='red', linestyle='--', label='Pt_Chap_K(M_k)')
plt.plot(x, Pt_chap, color='blue', linestyle='--', label='Pt_Chap_T(M_T)')
plt.xlabel('x')
plt.legend()
```

Out[16]:

<matplotlib.legend.Legend at 0x2110dd25dc0>

Figure 1 : Polynome_cible_réel_PT_et_ses_estimations_M_T_&_M_k



Test de positivité du transfert et tracé des p-valeurs du test

In [20]:

```
#New test set, take to be gigger than the previous one (for plot the p-value)
Nz=100
z_t=np.ones((Nz,D))
z_t[:,1]=np.linspace(-3,3,Nz)
z_t[:,2]=np.linspace(-3,3,Nz)**2
z_t[:,3]=np.linspace(-3,3,Nz)**3
random.seed(a=255, version=2)
epsilon_t=np.ones((len(z_t),1))
for i in range(Nz):
    epsilon_t[i,0]=random.normalvariate(mu=0.0, sigma=m.sqrt(sigma_t))
Y_z=z_t@beta_t + epsilon_t
```

In [21]:

```
from scipy.stats import f
#Test for the positiveness of the transfer( and plot the pk_x)
k=39
A_k = matrix_power_diagonalization_stable(A, k)
omegak=(1/alpha)*np.linalg.inv(big_sigma_t_train)@(np.eye(D) - A_k)
p=np.median(np.linspace(10**(-5),1))
beta_t_chap=np.linalg.inv( np.transpose(X_t_train)@X_t_train )@( np.transpose(X_t_train)
@Y_t_train )
beta_s_chap=np.linalg.inv( np.transpose(X_s)@X_s )@( np.transpose(X_s)@Y_s )
beta_k_chap=A_k@beta_s_chap + (np.eye(D) - A_k)@beta_t_chap
sigma_t_chap_carre=((np.linalg.norm(Y_t_train - X_t_train@beta_t_chap ,ord=2))**2) / (Nt
_train-D)
sigma_s_chap_carre=((np.linalg.norm(Y_s - X_s@beta_s_chap , ord=2))**2) / (Ns-D)
Y_t_test_chap=np.ones(len(z_t))
s=0
y=np.ones(len(z_t))
for i in range(len(z_t)):
    phi_kx=(sigma_t_chap_carre)*( z_t[i,:]@(np.linalg.inv(big_sigma_t_train) -(alpha**2)
*omegak@big_sigma_t_train@omegak)@np.transpose(z_t[i,:]) - (p*np.linalg.norm(A_k@np.trans
pose(z_t[i,:]), ord=2))**2 ) / ((sigma_s_chap_carre)*z_t[i,:]@A_k@np.linalg.inv(big_sigm
a_s)@A_k@np.transpose(z_t[i,:]) )
    p_kx=f.sf(phi_kx, Nt_train-D, Ns-D)
    y[i]=p_kx
    if p_kx<0.05:
        Y_t_test_chap[i]=(z_t[i,:]@beta_k_chap).item()
        s=s+1
```

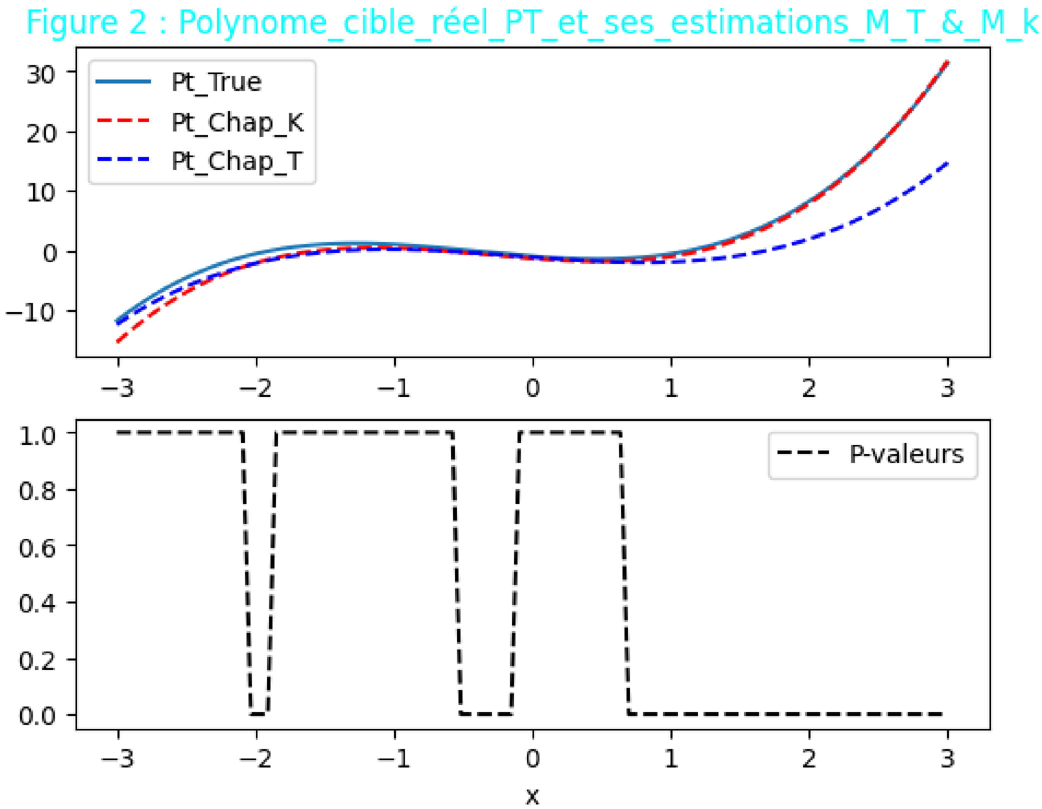
```
    else:
        Y_t_test_chap[i]=(z_t[i,:]@beta_t_chap).item()
    #print(s)
for i in range(len(y)):
    if y[i]>0.05:
        y[i]=1
    else:
        y[i]=0
```

In [22]:

```
#plots the p-values of test
x = np.linspace(-3, 3, 800)
plt.subplot(211)
plt.title('Figure 2 : Polynome_cible_réel_PT_et_ses_estimations_M_T_&_M_k',color='cyan')
plt.plot(x, Pt_true, label='Pt_True')
plt.plot(x, Pt_chap_k, color='red', linestyle='--', label='Pt_Chap_K')
plt.plot(x, Pt_chap, color='blue', linestyle='--', label='Pt_Chap_T')
plt.legend()
x = np.linspace(-3, 3, 100)
plt.subplot(212)
plt.plot(x,y,color='black', linestyle='--', label='P-valeurs')
plt.xlabel('x')
plt.legend()
```

Out[22]:

<matplotlib.legend.Legend at 0x2110f92a870>



Tracé des valeurs de $Y_{prédit}$ du test en fonction des valeurs observées

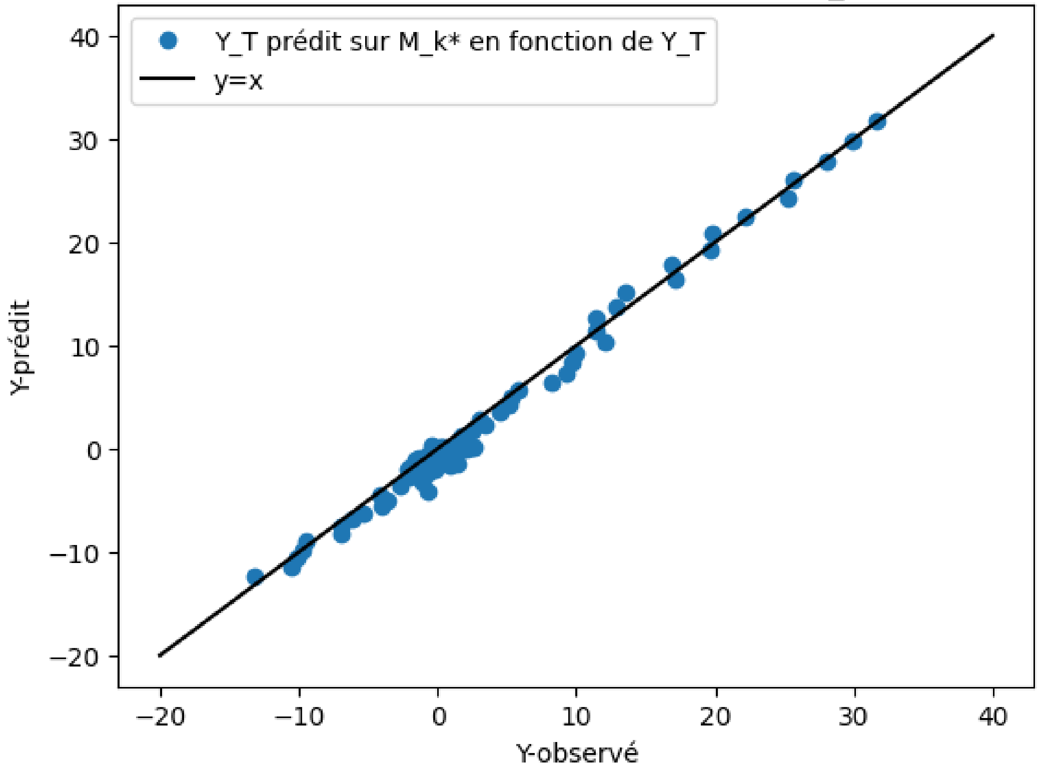
In [28]:

```
plt.title('Figure 3 : Valeurs prédites du vecteur réponse cible test Y_T en fonction des vraies')
plt.plot(Y_z,Y_t_test_chap,"o",label='Y_T prédit sur M_k* en fonction de Y_T')
plt.xlabel('Y-observé')
plt.ylabel('Y-prédit')
x = np.linspace(-20, 40, 100)
plt.plot(x,x,color='black',label='y=x')
plt.legend()
```

Out[28]:

```
<matplotlib.legend.Legend at 0x2110f89e7b0>
```

Figure 3 : Valeurs prédites du vecteur réponse cible test Y_T en fonction des vraies



In [ ]:

In [ ]: