In [1]:

```
from mpl_toolkits import mplot3d
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
plt.rc('font',family='serif')
```

In [2]:

```
d=1000
n=2000
A = np.random.normal(size=(n,d)) / np.sqrt(n)
y = np.random.normal(size=n)
lambd= 1
```

We consider the Ridge cost:

$$f(x) = \frac{1}{2}\|Ax - y\|^2 + \frac{\lambda}{2}\|x\|^2,$$

where $\lambda > 0$ is some regularization parameter.

(a) Show that f is can be written in the format the function f of Problem 12.2, for some $M \in \mathbb{R}^{d\times d}$, $b \in \mathbb{R}^d$ and $c \in \mathbb{R}$. Compute numerically the values of L and μ. Plot the eigenvalues of $H_f(x)$ using an histogram.

We have

$$f(x) = \frac{1}{2}\|Ax - y\|^2 + \frac{\lambda}{2}\|x\|^2 = \frac{1}{2}x^T A^T Ax - \langle x, A^T y\rangle + \frac{1}{2}\|y\|^2 + \frac{\lambda}{2}\|x\|^2 = \frac{1}{2}x^T (A^T A + \lambda Id)x -$$

Hence f can be put in the format of Problem 12.2 with $M = A^T A + \lambda Id$, $b = A^T y$ and $c = \frac{1}{2}\|y\|^2$.

In [32]:

```
M = A.T @ A + lambd * np.identity(d)
w,v = np.linalg.eigh(M)
L=np.max(w)
mu=np.min(w)
print('L is equal to ',L ,'and mu is equal to ',mu)
```

```
L is equal to  3.9042013011704997 and mu is equal to  1.0850698634375
893
```

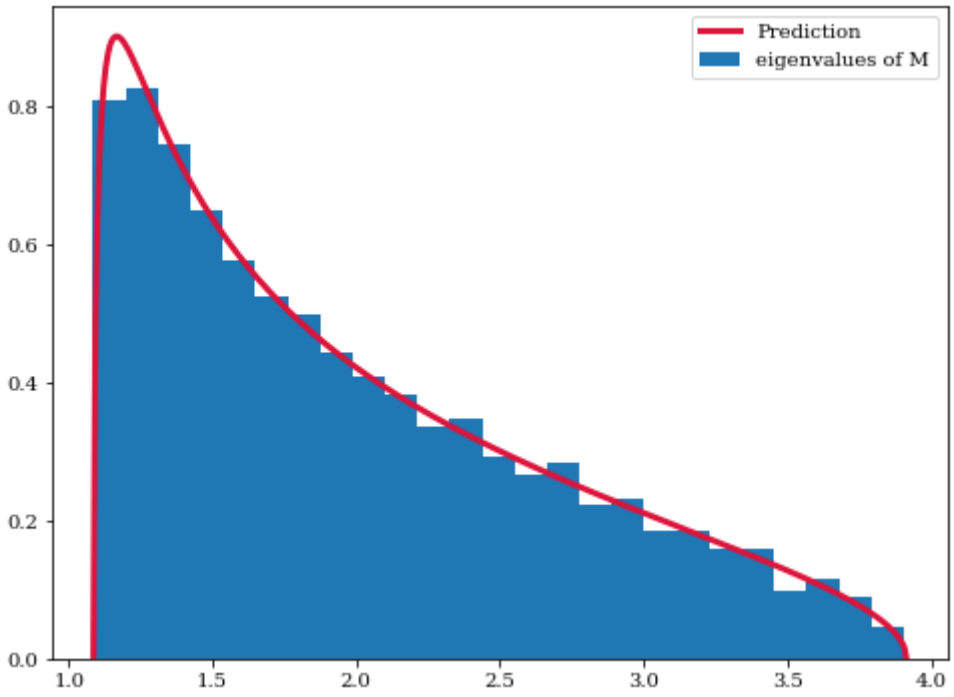
In [31]:

```
# Optional: the limiting shape of the histogram of the eigenvalues of M
# is known as the "Marcenko-Pastur" distribution. We plot is below

lambdaMin = (1-np.sqrt(d/n))**2
lambdaMax = (1+np.sqrt(d/n))**2
print(lambdaMin+lambd,lambdaMax+lambd)

t=np.linspace(lambdaMin,lambdaMax,400)
MP = (1/(2*np.pi))*(n/d)*np.sqrt((lambdaMax-t)*(t-lambdaMin))/t

plt.figure(figsize=(8,6))
plt.hist(w,bins=25,density=True, label='eigenvalues of M')
plt.plot(t+lambd,MP,color='crimson',linewidth=3, label='Prediction')
plt.legend()
```

```
1.0857864376269049 3.914213562373095
```

Out[31]:

```
<matplotlib.legend.Legend at 0x7f343ce14400>
```



(b) Implement gradient descent with constant step-size $\beta = 1/L$ (as in Problem 12.2), with random initial

position x_0. Plot the log-error $\log(\|x_t - x_*\|)$ as a function of t.

In [33]:

```
T=30
B = np.identity(d) - M/L
b = A.T @ y
x=np.random.normal(size=(d,T))
for t in range(T-1):
    x[:,t+1] = B @ x[:,t] + b /L
```

In [34]:

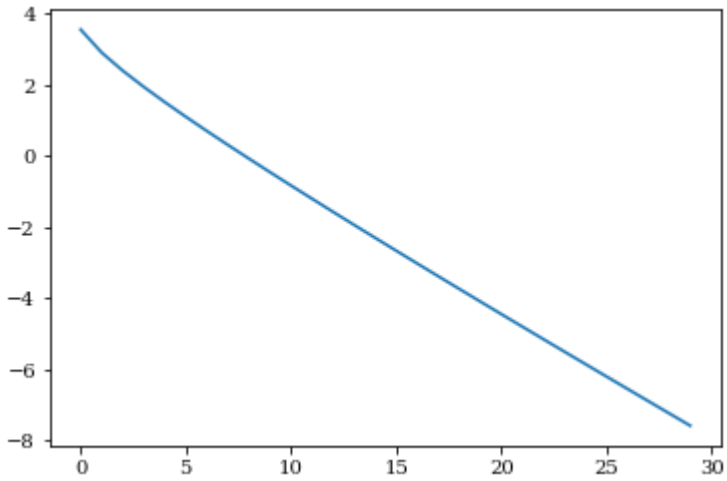
```
x_star= np.linalg.inv(M) @ A.T @ y
```

In [35]:

```
error = [np.sqrt(np.sum(np.square(x[:,t]-x_star))) for t in range(T)]
plt.plot(np.arange(T),np.log(error))
```

Out[35]:

```
[<matplotlib.lines.Line2D at 0x7f3440d7b280>]
```



(c) Implement gradient descent with momentum, with the same parameters as in Problem 12.4. Plot the log-error $\log(\|x_t - x_*\|)$ as a function of t, on the same plot than the log-error of gradient descent without momentum.

In [36]:

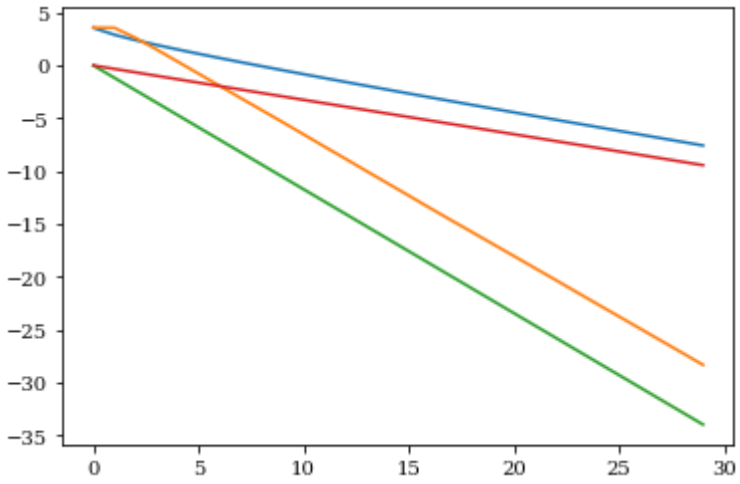
```
T=30
gamma = ((np.sqrt(L)-np.sqrt(mu))/(np.sqrt(L)+np.sqrt(mu)))**2
beta= 4 / (np.sqrt(mu)+np.sqrt(L))**2
B = np.identity(d) - beta*M
b = A.T @ y
x=np.random.normal(size=(d,T))
for t in range(T-2):
    x[:,t+2] = B @ x[:,t+1] + beta*b + gamma*(x[:,t+1]-x[:,t])
```

In [37]:

```
error2 = [np.sqrt(np.sum(np.square(x[:,t]-x_star))) for t in range(T)]
plt.plot(np.arange(T),np.log(error))
plt.plot(np.arange(T),np.log(error2))
t=np.arange(T)
plt.plot(t*np.log(gamma)/2)
plt.plot(t*np.log(1- mu/L))
```

Out[37]:

```
[<matplotlib.lines.Line2D at 0x7f3440d47730>]
```



In [ ]:

In [ ]:

In [ ]: