# homework3

November 21, 2021

[2]:
```
import math
import numpy as np
import scipy.linalg as la
import matplotlib.pyplot as plt
```

---

# 1 Exercise 1

# 2 (a)

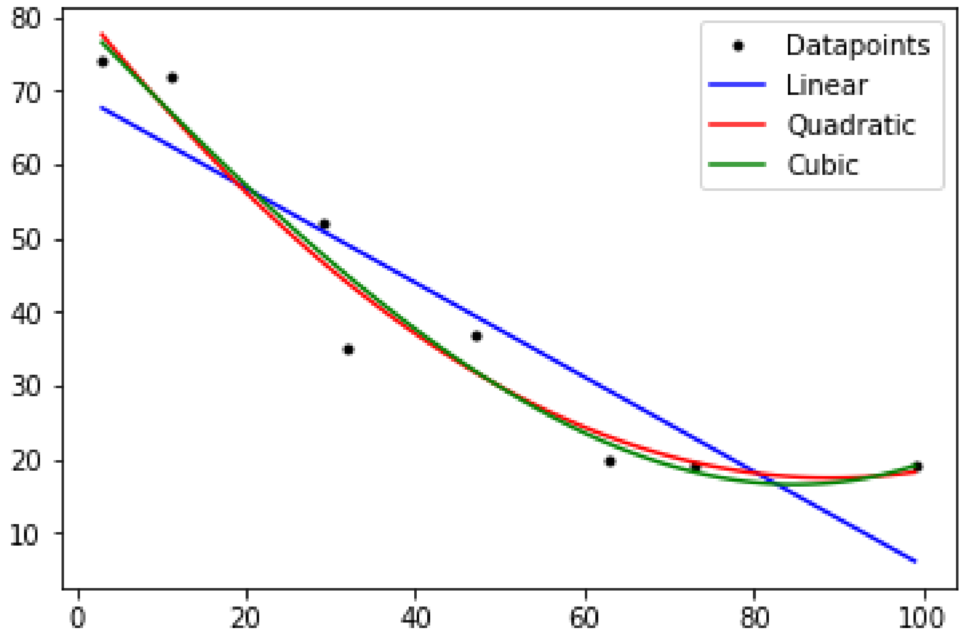
We have the table of data in the next cell.

Using `numpy.linalg.lstsq()`, fit a straight line, a quadratic function, and a cubic function to these data. Plot the data and your fitted functions in a graph. It is **not** allowed to use numpy.polyfit, but you may have a look at the documentation to see some examples.

[3]:
```
t = np.array([3, 11, 29, 32, 47, 63, 73, 99])
w = np.array([74, 72, 52, 35, 37, 20, 19, 19])
```

[4]:
```
M = np.vstack([t, np.ones(len(t))]).T
A = np.vstack([t**2,t,np.ones(len(t))]).T
Q = np.vstack([t**3,t**2,t,np.ones(len(t))]).T

a, b = np.linalg.lstsq(M, w, rcond=None)[0]
c,d,e = np.linalg.lstsq(A,w,rcond=None)[0]
f,g,h,i = np.linalg.lstsq(Q,w,rcond=None)[0]

smaller_t = np.linspace(t[0],t[-1],100)

plt.plot(t,w,'ko',markersize=3,label='Datapoints')
plt.plot(smaller_t,a*smaller_t+b,'b-',label='Linear')
plt.plot(smaller_t,c*smaller_t**2+d*smaller_t+e,'r-',label='Quadratic')
plt.plot(smaller_t,f*smaller_t**3+g*smaller_t**2+h*smaller_t+i,'g-',␣
 ↪label='Cubic')
plt.legend()
plt.show()
```

```
print(f"A residual of {round(np.sum((w-(a*t + b))**2),3)} was found with the␣
 ↪straight line fit.")
print(f"A residual of {round(np.sum((w-(c*t**2+d*t+e))**2),3)} was found with␣
 ↪the quadratic function fit.")
print(f"A residual of {round(np.sum((w-(f*t**3+g*t**2+h*t+i))**2),3)} was found␣
 ↪with the cubic function fit.")
```




```
A residual of 599.666 was found with the straight line fit.
A residual of 181.77 was found with the quadratic function fit.
A residual of 174.967 was found with the cubic function fit.
```

---

# 3 Exercise 2

We want to reconstruct a function $s(t)$ (also called the signal in this exercise), $t \in [0, 1]$, from data given by

$$d(t) = \int_0^1 s(t)\, dt + \text{noise}.$$

We assume the data is given at $n$ equally space time points $t_j = jh$, $h = \frac{1}{n}$, $j = 1, 2, \ldots, n$. The data is therefore a vector $d = [d_1, \ldots, d_n]$, where $d_j$ denotes the value at $t_j$. The signal $s$ is to be reconstructed at time points $t_{j-1/2} = (j - 1/2)h$ for $j = 1, 2, \ldots, n$. It is described by a vector $s = [s_1, \ldots, s_n]$ with $s_j$ the value at $t_{j-1/2}$. Numerical integration is described in Chapter 8 of the book by Heath. Using the composite midpoint rule, the vectors $s$ and $d$ are related by

$$d = A \cdot s + \text{noise}$$

where

$$A = \begin{bmatrix} h & 0 & 0 & \dots & 0 \\ h & h & 0 & \dots & 0 \\ h & h & h & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ h & h & \dots & h & h \end{bmatrix}.$$

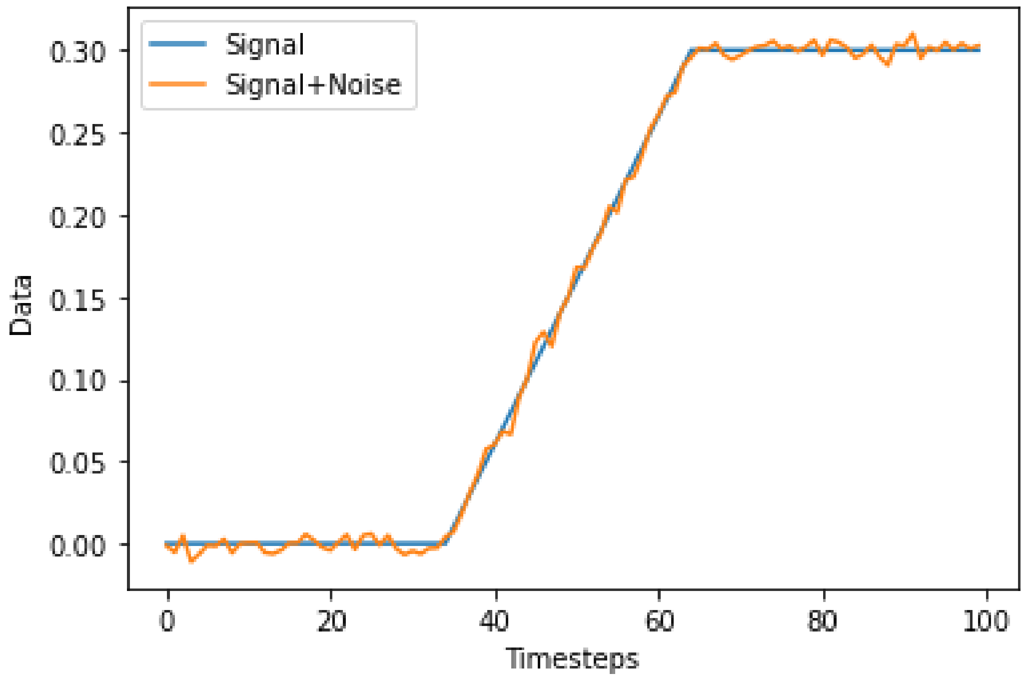
### 3.1 (a)

As a test signal we take

$$s_{\text{true}}(t) = \begin{cases} 1 & \text{if } |t - 1/2| < 0.15 \\ 0 & \text{otherwise} \end{cases}.$$

Generate data $d_0$ without noise and data $d_\epsilon$ with noise, where the noise is normally distributed, with mean zero and standard deviation $\epsilon = 0.005$. Take for example $n = 100$. Plot the data.

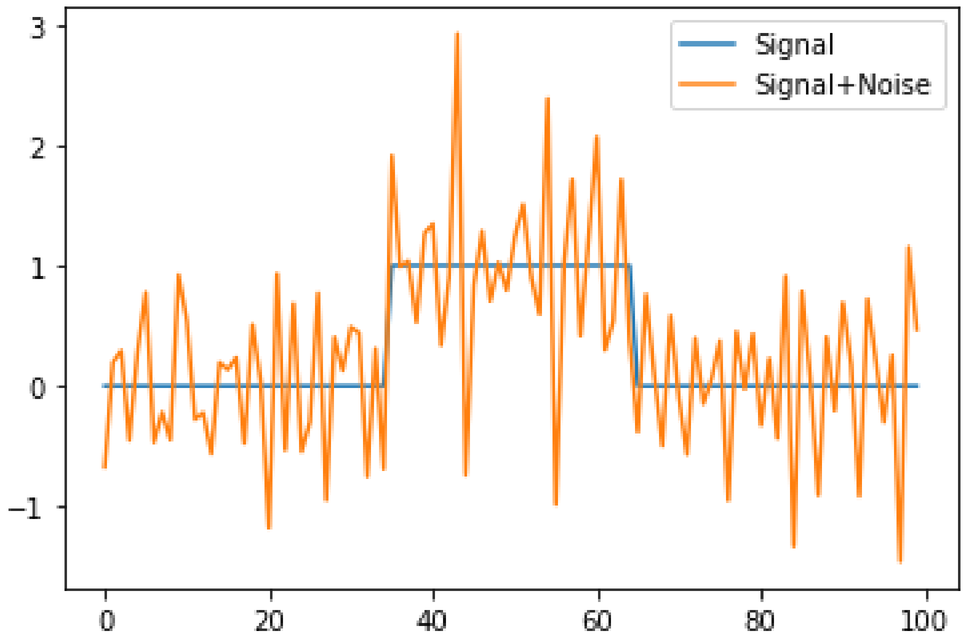
[24]:
```
def signal(t):
    return np.where(np.abs(t - 1/2)<0.15, 1, 0)

def data(s, n, noise_on=True):
    h = 1/n
    A = np.tril(h*np.ones((n,n)))

    noise = np.zeros(n)
    if noise_on:
        noise = np.random.normal(0, 0.005, n)

    return A.dot(s) + noise

n = 100
t = np.arange(0,1,1/n)

s = signal(t)

d_0 = data(s,n,False)
d_e = data(s,n)

plt.figure()
plt.plot(d_0,label='Signal')
plt.plot(d_e,label='Signal+Noise')
plt.xlabel("Timesteps")
plt.ylabel("Data")
plt.legend()
plt.show()
```




## 3.2 (b)

Try to determine $s$ from $d_0$ by inverting the matrix $A$, ignoring the noise term. Do the same with $d_\epsilon$ instead of $d_0$. Plot the results. What do you observe about the errors in the inversion?

[21]:
```
h = 1/n
A = np.tril(h*np.ones((n,n)))
A_inv = np.linalg.inv(A)
noise = np.random.normal(0, 0.005, n)

s_0 = A_inv.dot(d_0)
s_e = A_inv.dot(d_e)
plt.plot(s_0,label='Signal')
plt.plot(s_e,label='Signal+Noise')
plt.legend()
plt.show()
```




### 3.3 (c)

Create a function to solve the linear system $As = d$ using the singular value decomposition (SVD) $A = U\Sigma V^T$ and run a test to verify that your function is correct. `numpy` and `scipy` contain functions to compute the SVD. These may be used.

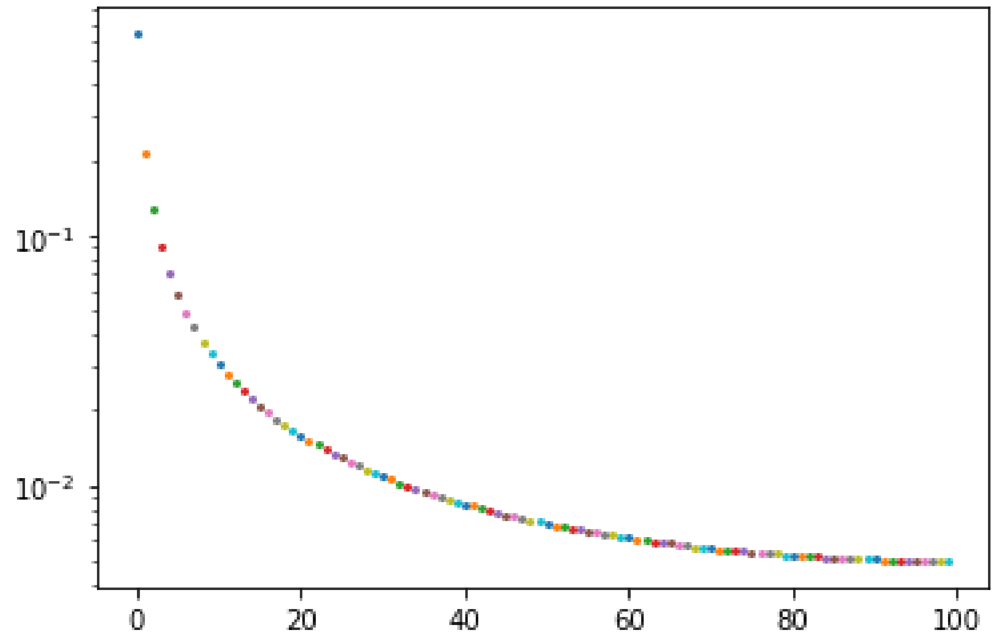
Then plot the singular values of the matrix $A$ and explain the behavior found in (b) using the SVD.

[20]:
```
def solve(d, n):
    h = 1/n
    A = np.tril(h*np.ones((n,n)))
    u,singulars,vt = np.linalg.svd(A, full_matrices=True)
    sigma = np.diag(singulars)

    u_inv = np.linalg.inv(u)
    sigma_inv = np.linalg.inv(sigma)
    vt_inv = np.linalg.inv(vt)
    s = vt_inv.dot(sigma_inv.dot(u_inv)).dot(d)
    return s

s_0_estimate = solve(d_0,n)
s_e_estimate = solve(d_e,n)

print(s_0_estimate - s_0)
print(s_e_estimate - s_e)
```

```
sigma = np.diag(np.linalg.svd(A, full_matrices=True)[1])
plt.plot(sigma, 'o',markersize=2)
plt.yscale('log')
plt.show()
```

```
[ 6.45796177e-14 -4.22571960e-14  4.78760261e-14 -1.85984908e-15
 -1.88435378e-14 -8.78255801e-16  1.42750395e-14 -4.12270912e-15
  9.27702359e-15 -1.74277953e-14  6.47544518e-15  2.54149479e-14
  3.76781245e-14 -5.08523085e-14 -2.58220528e-14  3.18564619e-14
  4.07283235e-14 -2.68465805e-15 -6.63774591e-15  4.05113443e-14
  1.44791124e-14  6.38183950e-15 -3.91642274e-14 -2.16930640e-14
 -2.96151992e-15  3.95183886e-15  3.03569669e-14  5.67060288e-15
 -2.81678847e-14 -3.53757301e-14  1.29063427e-16 -1.50018886e-15
 -2.49662790e-14  4.47079873e-14  1.35155775e-14 -1.50990331e-14
  1.77635684e-15  2.57571742e-14  5.77315973e-15 -7.99360578e-15
 -1.42108547e-14  3.55271368e-14  4.52970994e-14 -4.44089210e-14
  3.55271368e-15  2.66453526e-14 -1.24344979e-14 -1.77635684e-14
  0.00000000e+00  3.55271368e-15  3.55271368e-15  2.30926389e-14
  1.77635684e-14 -8.17124146e-14 -7.10542736e-15  4.61852778e-14
 -5.32907052e-14  3.55271368e-15  1.77635684e-14  2.13162821e-14
 -3.90798505e-14 -2.48689958e-14 -3.90798505e-14  4.97379915e-14
 -7.10542736e-15  2.13162821e-14  1.77635684e-14 -6.39488462e-14
 -2.48689958e-14  1.42108547e-14  1.42108547e-14  1.06581410e-14
 -6.03961325e-14  4.97379915e-14  2.13162821e-14 -7.10542736e-15
  3.90798505e-14 -7.46069873e-14 -7.10542736e-15  6.03961325e-14
 -1.17239551e-13 -2.48689958e-14 -1.06581410e-14 -1.42108547e-14
 -3.55271368e-14  2.84217094e-14  3.90798505e-14 -3.19744231e-14
  1.77635684e-14  6.75015599e-14 -6.03961325e-14 -3.90798505e-14
 -2.48689958e-14  3.55271368e-15 -7.10542736e-15  0.00000000e+00
 -7.10542736e-15  6.39488462e-14  1.42108547e-14 -3.55271368e-14]
[ 6.45039577e-14 -4.04676292e-14  4.79616347e-14 -7.43849426e-15
 -1.33781874e-14 -2.10942375e-15  1.70974346e-14 -8.68749517e-15
  8.49320614e-15 -1.34336986e-14  5.88418203e-15  2.85882429e-14
  3.10862447e-14 -4.26325641e-14 -2.60347299e-14  2.81719092e-14
  4.39370762e-14 -5.44009282e-15 -9.43689571e-15  4.30211422e-14
  1.37667655e-14  2.77555756e-15 -3.60822483e-14 -1.75415238e-14
 -5.55111512e-15  2.83106871e-15  3.30846461e-14  3.33066907e-15
 -2.93098879e-14 -3.07809334e-14  2.10942375e-15 -4.55191440e-15
 -2.77555756e-14  4.52970994e-14  1.66533454e-14 -1.68753900e-14
  2.22044605e-16  2.53130850e-14  7.54951657e-15 -1.24344979e-14
 -1.59872116e-14  3.19744231e-14  5.32907052e-14 -4.97379915e-14
  5.32907052e-15  1.95399252e-14 -1.42108547e-14 -2.30926389e-14
  3.55271368e-15  1.77635684e-15  8.88178420e-15  1.42108547e-14
  1.42108547e-14 -6.75015599e-14 -1.42108547e-14  4.61852778e-14
 -6.03961325e-14  1.06581410e-14  1.42108547e-14  2.13162821e-14
 -3.55271368e-14 -2.48689958e-14 -4.26325641e-14  4.97379915e-14
  3.55271368e-15  2.48689958e-14  1.06581410e-14 -7.46069873e-14
 -2.13162821e-14  1.42108547e-14  1.77635684e-14  7.10542736e-15
```

```
-5.68434189e-14  3.90798505e-14  2.48689958e-14 -3.55271368e-15
 4.26325641e-14 -7.81597009e-14 -1.06581410e-14  6.39488462e-14
-1.03028697e-13 -2.84217094e-14 -7.10542736e-15 -1.42108547e-14
-2.84217094e-14  2.84217094e-14  3.90798505e-14 -3.19744231e-14
 2.48689958e-14  6.75015599e-14 -6.03961325e-14 -3.90798505e-14
-2.13162821e-14  7.10542736e-15 -7.10542736e-15  0.00000000e+00
-1.06581410e-14  7.10542736e-14  2.13162821e-14 -3.55271368e-14]
```




The solve function has errors of $10^{-13}$ or smaller, which means the error is quite substancial.

The singular values in the plot show that most singular values are very small, but there are a few relatively big ones. This explains the error seen in exercise (b).

### 3.4 (d)

When solving the system using the SVD, the matrix $\Sigma^{-1} = \mathrm{diag}(\sigma_1^{-1}, \dots, \sigma_n^{-1})$ is used. To **regularize** the problem the matrix $\Sigma^{-1}$ can be replaced by a matrix

$$T = \mathrm{diag}(\sigma_1^{-1}, \dots, \sigma_k^{-1}, 0, \dots 0),$$

where $\sigma_j^{-1}$ is replaced by 0 if $\sigma_j$ is smaller than a threshold $\alpha$.

Implement a function `TruncatedSVDSolve(A, b, alpha)` that performs this procedure. Find a value of $\alpha$ such that the test signal $s$ is reconstructed reasonably well from the noisy data. Plot the result.

Explain that in the presence of noise, the result of `TruncatedSVDSolve(A, b, alpha)` can be more accurate than exact inversion.