Homework 3

September 21, 2017

Ramona Bendias, Eda Dagasan

1 Task 1

We implemented the unknown a_i with help of the three different ways. The a looks exactly the same with the normal equation, the qr factorization and with the svd decomposition: a= [3.99977426e+00, -1.53545576e-03 , 3.00267362e+00, 1.99819673e+00, 1.00040411e+00]. If we plot the discrete values and the function it fits really well.

To compare the execution time we can firstly note that we have to build A (the matrix with the function in it) for all methods. So this time doesn't matter. For the normal equation solution we need to transpose A, built two products of two matrices and solve one linear equation system. (If we restart the kernel, it takes about 378ms for the execution) For the qr factorization we need the qr factorization command or compute it and we need also one transpose, one product and one solution of a linear equation system. (about 344ms) For the last solution, the svd decomposition, we need the svd decomposition, two inverses, three products and build the diagonal matrix with the diagonal entries from the svd decomposition. (123ms) It seems that the svd takes the most time because there are the most calculations. But even when executing the program with the existing A and the stored solution the normal equation takes the most time (about 20ms). We think the execution time of the last method is so good, because we don't have to use the solve command. We can built the inverse of the diagonal matrix very easy so that we can use $x = A^{-1} * b$ to solve the linear equation.

You can find the code in the attachment also with the plot.

2 Task 2

For the problem $A \mapsto A^{-1}b$, with a given matrix A and b as input, we have that the condition number

$$\frac{||\delta x||}{||x||}\frac{||b||}{||\delta b||} \leq \kappa(A) = ||A||||A^{-1}|| = \frac{\sigma_n}{\sigma_1}$$

when the norm is the 2-norm. Then, if we choose $b = u_1$, where u_1 is the left singular vector corresponding to the greatest singular value σ_1 in te SVD-decomposition of A,

$$x = A^{-1}b = V\Sigma^{-1}U^T u_1 = V\Sigma^{-1}\begin{bmatrix} u_1^T \\ \vdots \\ u_n^T \end{bmatrix} u_1 = V \begin{bmatrix} \frac{1}{\sigma_1} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \frac{1}{\sigma_n} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} v_1 & \dots & v_n \end{bmatrix} \begin{bmatrix} \frac{1}{\sigma_1} \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \frac{v_1}{\sigma_1},$$

where v_1 is the corresponding right singular vector. Then

$$\frac{||b||}{||x||} = \frac{||u_1||}{||\frac{1}{\sigma_1}v_1||} = \sigma_1.$$

Similarily, with the choice $\delta b = u_n$ – the left singular vector associated with the smallest singular value σ_n,

$$\frac{||\delta b||}{||\delta x||} = \frac{||\frac{v_1}{\sigma_n}||}{||u_n||} = \frac{1}{\sigma_n}.$$

Finally

$$\frac{||\delta x||}{||x||}\frac{||b||}{||\delta b||} = \frac{\sigma_1}{\sigma_n} = ||A||||A^{-1}||,$$

and the equality holds.

3 Task 3

We constructed the Hilbert matrix H of dimension 50 and its inverse in python. The condition number of the matrix we determined to

$$\kappa(H) = ||H||||H^{-1}|| = 1.42 \cdot 10^{74}.$$

However when calculating the condition number using the singular values (which theoretically is the same as $||H||_2||H^{-1}||_2 = \sigma_1/\sigma_n$) the value of κ is $1.39 \cdot 10^{16}$.

We compute (using the result of task 2)

$$\frac{||\delta x||}{||x||}\frac{||b||}{||\delta b||}, \tag{1}$$

with the left singular vectors described in task 2. We can get these by SVD-decomposing H,

$$H = U\Sigma V^T,$$

and taking u_1, u_n from the matrix U. This works well for small dimensions (up to 10), so that the equality from task 2 holds, no matter which way of calculating the condition number of H we choose. However for larger dimensions (and specifically for n=50), $||H||||H^{-1}||$ is not equal to $\frac{\sigma_1}{\sigma_n}$. The fraction is for n=50:

$$\frac{||\delta x||}{||x||}\frac{||b||}{||\delta b||} = 1.74 \cdot 10^{16}.$$

This number corresponds much more to the second way of determinating the condition number of H. We suspect it can have to do with the properties of H (since it is very ill conditioned small numerical perturbations affects gravely the solution?). In particular since it works for lower dimensions we dont suspect an error in our implementation.

We tried to put in other vectors for $b, \delta b$ and saw that we never suceeded to get a larger condition number for this problem, which motivates that the choice was the good one.

4 Attachements

4.1 Task 1 – Code and plot

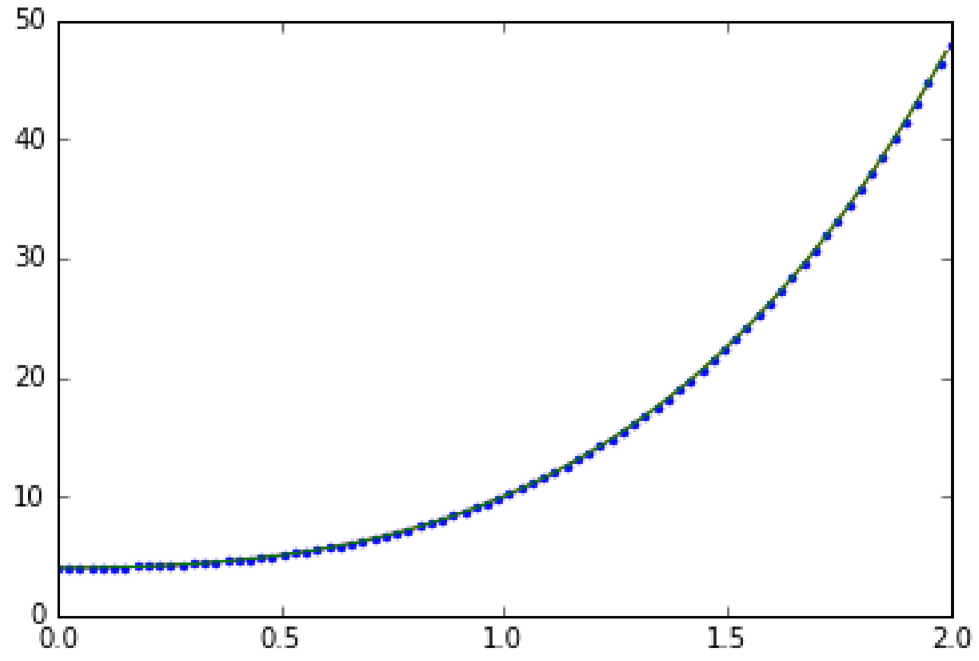
```
In [2]: import numpy as np
        import scipy.linalg as sl
        import csv
        import matplotlib.pyplot as plt
        % matplotlib inline

        # import the text file and create two vectors v and t
        import os

        try:
                datfilestr = input('Which file name?\n---->  ')
                datfile=open(datfilestr,'r')
        except FileNotFoundError:
                txt="""File {} not found in catalogue {}.
                    Make sure you downloaded the file from the course homepage
                    and you run this script in its catalogue.
                    """
                print(txt.format(datfilestr,os.path.abspath(os.curdir)))
        else:
                line = [ll.strip('\n').split(',') for ll in datfile.readlines()]
                t,v=list(zip(*[(float(l[0]),float(l[1])) for l in line]))
                t=np.array(t)
                v=np.array(v)
                print('You get now two shape {sh} arrays "t" and "v" with data.'.format(sh=t.shape))
```

```
Which file name?
---->  homework3.txt
You get now two shape (80,) arrays "t" and "v" with data.
```

```
In [2]: # blue points are the given diskret values
        # implement a code which uses the least square problem with 3 different
        class Least_square_methode():

            def __init__(self, v, t):
                self.v = v
                self.t = t
                self.A = np.empty((0,5), int)
                for t_i in self.t:
                    self.A = np.vstack([self.A,[1, t_i, t_i**2, t_i**3, t_i**4]])

            def normal_equation(self):
                AT = np.transpose(self.A)
                normal = sl.solve(np.dot(AT, self.A), np.dot(AT,v))
                return normal
            def qr_factorisation(self):
                q,r = np.linalg.qr(self.A) # compute the reduced QR factorization
                qT = np.transpose(q) # build the transpose because q isn't square
                qv = np.dot(qT,v)
                beta = sl.solve(r,qv) # solve the upper-triangular system R*x = QT*v  beta is the x vec
                return beta
            def svd_decomposition(self):
                u,s,d = np.linalg.svd(self.A) # computes the svd
                u = np.linalg.inv(u)
                uv = np.dot(u,self.v)
                S = np.zeros((5, 80))
                for i in range(5):
                    S[i,i] = 1/s[i]
                w = np.dot(S,uv)
                gamma = np.dot(np.linalg.inv(d),w)
                return gamma

        A = Least_square_methode(v,t)
        normal = A.normal_equation()
        print(normal)
        qr = A.qr_factorisation()
        print(qr)
        svd = A.svd_decomposition()
        print(svd)
        g = np.arange(0,2,0.01)
        plt.plot(g, ((svd[0]+svd[1]*g+svd[2]*g**2+svd[3]*g**3+svd[4]*g**4)),'r-', t,v,'.', g, \
                 ((qr[0]+qr[1]*g+qr[2]*g**2+qr[3]*g**3+qr[4]*g**4)),'g-')
```

```
[  3.99977426e+00  -1.53545580e-03   3.00267362e+00   1.99819673e+00
   1.00040411e+00]
[  3.99977426e+00  -1.53545576e-03   3.00267362e+00   1.99819673e+00
   1.00040411e+00]
[  3.99977426e+00  -1.53545576e-03   3.00267362e+00   1.99819673e+00
   1.00040411e+00]
```

```
Out[2]: [<matplotlib.lines.Line2D at 0x11f4beac8>,
         <matplotlib.lines.Line2D at 0x11f4be400>,
         <matplotlib.lines.Line2D at 0x11f4b5c50>]
```



4.2 Task 3 – Code

```
In [6]: H = sl.hilbert(50)
        H_inv = sl.invhilbert(50)

        U,s,V = sl.svd(H)    #U unitary matrix having left sing vectors as columns
                             #s vector with sing values ordered (greatest first)
        S = np.diag(s)       #constructing a diag matrix
        b = U[:,0] #first column
        delta_b = U[:,len(s)-1] #last column

        x = H_inv@b
        delta_x = H_inv@delta_b


        K = (sl.norm(delta_x, ord=2)/sl.norm(x, ord=2))/(sl.norm(delta_b, ord=2)/sl.norm(b, ord=2))

        K_A = sl.norm(H, ord=2)*sl.norm(H_inv, ord=2)
        K2_A = s[0]/s[len(s)-1]

        print('K',K,'\nK(A)', K_A, '\nsigma_1/sigma_n', K2_A)
```

```
K 1.7423272911991916e+16
K(A) 1.42294184155e+74
sigma_1/sigma_n 1.39472330019e+16
```