UNIVERSITY OF SUSSEX
Scientific Computing
Tutor: Dr. Ilian Iliev, Office: Pev 3 4A5

Problem Sheet 2 (Problem 2 will be assessed)
Deadline: 12pm on Wednesday, October 30th, 2013.
Penalties will be imposed for submissions beyond this date.
Final submission date: Thursday, October 31st, 2013.
No submissions will be accepted beyond this date.

1. Consider the n simultaneous equations $\mathbf{A}\mathbf{x} = \mathbf{b}$, where

$$A_{ij} = (i+j)^2, \;\; b_i = \sum_{j=0}^{n-1} A_{ij}, \;\; i = 0, 1, ..., n-1, \;\; j = 0, 1, ..., n-1$$

Clearly, the solution is $\mathbf{x} = (1, 1, \ldots, 1)^T$. Write a program that solves these equations for any given n using both the provided code `GaussPivot` and the internal `solve` routine. Run your program with $n = 2, 3$, and 4 and comment on the results.

Solution: We first import the required modules and start the main `for` loop over the matrix sizes to be done:

```
from numpy import *
from numpy.linalg import *
from gaussPivot import *
import pylab

for n in range(2,5):
```

Next,we have to setup the matrix A and right hand side vector b, e.g. as follows:

```
    A=zeros((n,n))

    for i in range(n):
        for j in range(n):
            A[i][j]=(1.0*i+j)**2
    print h
    x0=ones(n)
    b=dot(h,x0)
```

Important: make sure arrays are filled with floats, not integers!

Then, the solution is given simply by

```
x1=solve(A,b)

x2=gaussPivot(A,b)

error1[n-2]=max(x1-x0)
error2[n-2]=max(x2-x0)
print(n,error1[n-2], error2[n-2])
```

The result is:

```
[[ 0.  1.]
 [ 1.  4.]]
(2, 0.0, 0.0)
[[  0.   1.   4.]
 [  1.   4.   9.]
 [  4.   9.  16.]]
(3, 6.6613381477509392e-16, 0.0)
[[  0.   1.   4.   9.]
 [  1.   4.   9.  16.]
 [  4.   9.  16.  25.]
 [  9.  16.  25.  36.]]
Matrix is singular
```

Clearly, the errors for $n = 2$ and 3 are quite small, but for $n = 4$ the matrix becomes singular, i.e. the 4 equations are not all independent.

2. Use the built-in NumPy linalg function `vander(v)` to create the $n \times n$ Vandermonde matrix:

$$V_{m,n} = \begin{pmatrix} v_0^n & v_0^{n-1} & \cdots & v_0 & 1 \\ v_1^n & v_1^{n-1} & \cdots & v_1 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ v_n^n & v_n^{n-1} & \cdots & v_n & 1 \end{pmatrix} \tag{1}$$

where $v = (v_0, v_1, \ldots, v_n)$ is a vector.

(a) Use this to create an $n \times n$ matrix A with $v = (1, 2, 3, ..., n)$ for n=5 to 27.

(b) For this matrix A construct a vector b so that the solution to $Ax = b$ is $x = (1, 1, \ldots, 1)^T$.

(c) Test your construction by applying the internal Numpy function `solve(A,b)` to compute x. What is the largest element-by-element error for each n? Plot the decimal log of the max element-by-element error vs. n. Comment on the results. Why do you think this happens?

[40]

Solution:

We first import the required modules, `numpy`, and `numpy.linalg`:

```
from numpy import *
from numpy.linalg import *
import pylab

xx=zeros(23) # we will store here the errors for plotting
error=zeros(23)
```

Next, we setup a `for` loop for the required range of values and calculate the Vandermonde matrix, the b vector which corresponds to a solution of ones and then call the solution routines:

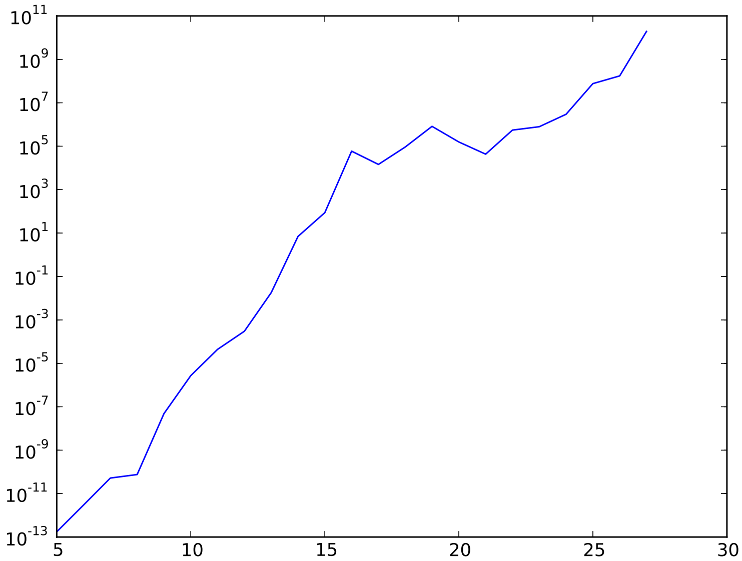
```
for n in range(5,28):

    v=arange(1,n+1)
    a=vander(v)

    x0=ones(n)
    x=zeros(n)
    b=dot(h,x0)

    x=linalg.solve(v,b) # solution using internal routine

    print n,max(x-x0) # prints the result on screen

    xx[n-5]=n                #save errors for plotting
    error[n-5]=max(x-x0)
```

```
pylab.semilogy(xx,error)    #plot errors
pylab.show()
```

Result is

```
5 1.73860925656e-13
6 2.97051272469e-12
7 5.20652410074e-11
8 7.55688844833e-11
9 4.83710294041e-08
10 2.71211799774e-06
11 4.40419572507e-05
12 0.000300584294233
13 0.0179439816366
14 6.98573835649
15 86.7835673439
16 59212.9703732
17 14488.7032158
18 91767.8664893
19 821035.517948
20 156770.4008
21 43018.0330413
22 552355.387253
23 790748.359786
24 2957109.1522
25 76330146.2547
26 173012589.13
27 19356424864.0
```

plotted in the Figure below. Errors rise very fast with n and for $n > 13$ become larger than the solution itself, i.e. results become meaningless.



10^11
10^9
10^7
10^5
10^3
10^1
10^-1
10^-3
10^-5
10^-7
10^-9
10^-11
10^-13
5
10
15
20
25
30