ORF307_HW1

January 25, 2022

ORF307 Homework 1

Due: Friday, Feburary 4, 2021 9:00 pm ET

- The jupyter file is available at https://github.com/ORF307/companion
- Please export your code with output as `pdf`.
- If there is any additional answers, please combine them as **ONE** pdf file before submitting to the Gradescope.

Q1 Interpolation of rational functions

A rational function of degree two has the form

$$f(t) = \frac{c_1 + c_2 t + c_3 t^2}{1 + d_1 t + d_2 t^2}$$

where c_1, c_2, c_3, d_1, d_2 are coefficients. ('Rational' refers to the fact that f is a ratio of polynomials. Another name for f is bi-quadratic.) Consider the interpolation conditions

$$f(t_i) = y_i \quad i = 1, ..., K$$

where t_i and y_i are given numbers. Express the interpolation conditions as a set of linear equations in the vector of coefficients $\theta = (c_1, c_2, c_3, d_1, d_2)$, as $A\theta = b$. Give A and b, and their dimensions.

Q2 Python timing test for linear equations

(a) Determine how long it takes for your computer to solve a system $Ax = b$ of $n = 2000$ linear equations in $n = 2000$ variables (with invertible coefficient matrix) using the Python's inverse function. You may use the following code to get you started.

[ ]:
```
import numpy as np
from time import time
N = 3000
A = 1 + np.random.normal(size=(N,N))
b = np.ones(N)

t_start = time()
```

```
Ainv = np.linalg.inv(A) # Compute inverse
# some other operation to solve the system ...
elapsed_time = time() - t_start
print('time to solve (a):', elapsed_time, 'sec')
```

(b) Python is rather clever about how it solves linear systems. Use the following function to solve the same linear system and determine how long it takes your computer to solve it. Verify that you get the same solution as from part (a).

[ ]:
```
x = np.linalg.solve(A,b)
```

(c) Now we will use LU factorization to solve the linear system. Use the following lines to first factorize the matrix A as $A = PLU$ where P is a permutation matrix, L is a lower triangular matrix and U is a upper triangular matrix. The idea is that we have $Ax = PLUx = b$. For convenience we have coded the functions `forward_substitution` and `backward_substitution` for you. A property of any permutation matrix P is that $P^TP = I$. Use this property to solve the same linear system as in the previous parts and determine how long your computer takes. Verify that you get the same solution as from parts (a) and (b).

Note Your implementation is not going to be the fastest one, but it will work well. If written in a lower level language such as C or C++, it can sometimes be faster. That's part of the tricks inside numpy's `np.linalg.solve` function.

[ ]:
```
def forward_substitution(L, b):
    n = L.shape[0]
    x = np.zeros(n)
    for i in range(n):
        x[i] = (b[i] - L[i,:i] @ x[:i])/L[i, i]
    return x

def backward_substitution(U, b):
    n = U.shape[0]
    x = np.zeros(n)
    for i in reversed(range(n)):
        x[i] = (b[i] - U[i,i+1:] @ x[i+1:])/U[i, i]
    return x
```

[ ]:
```
def solve_via_lu(P, L, U, b):
    '''
    Solve linear system Ax = b where
        A = PLU
    '''

    # Complete ...

    return x
```

[ ]:
```
from scipy.linalg import lu
# ...
P, L, U = lu(A)  # Factor matrix A as A = PLU
x_lu = solve_via_lu(P, L, U, b)  # Solve via LU decomposition
# ...
```

(d) Can you explain why the times differ by so much between parts (a) and (b)? How does the time in (c) compare?

Q3

Suppse x and y are Boolean feature vectors, *i.e.*, each entry is either 0 or 1, that encode the presence of symptoms in patients Alice and Bob. Which of the following are true statements?

(a) $x^T y$ is number of symptoms Alice and Bob have in common

(b) $\|x\|^2$ is the number of symptoms Alice has

(c) $\mathbf{1}^T y$ is number of symptoms Bob has

(d) $\|x - y\|^2$ is number of symptoms Alice has but Bob does not

(e) $\mathbf{1}^T(x - y)$ is the number of symptoms Alice has but Bob does not

(f) $x^T y = 0$ means that Alice and Bob do not share any symptoms