MTH 373/573:
Scientific Computing
Monsoon 2018

Homework 3

IIIT-Delhi
Kaushik Kalyanaraman

Due: October 23, 2018 by 11.55 pm **Total: 450 points.**

Before you start, please read General Submission Guidelines on Page 7.

Problem 1: Theoretical problems

(5 + 10 + 10 = 25 points)

(a) Give an example of a 2×2 matrix A and a nonzero starting vector x_0 such that power iteration fails to converge to the eigenvector corresponding to the dominant eigenvalue of A.

(b) Implement power iteration to compute the dominant eigenvalue and a corresponding normalized eigenvector of the matrix

$$A = \begin{bmatrix} 2 & 3 & 2 \\ 10 & 3 & 4 \\ 3 & 6 & 1 \end{bmatrix}$$

As a starting vector, take $x_0 = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$.

(c) Using any of the method for deflation given in Section 4.5.4 of the textbook (Heath, second edition), deflate out the eigenvalue found in the previous part and apply power iteration again to compute the second largest eigenvalue of the same matrix.

Problem 2: Least squares fitting with Gram-Schmidt and QR

(30 + 30 + 40 = 100 points)

(a) Write a program that implements QR factorization using the Gram-Schmidt procedure.

(b) Compute the QR factorization for three random matrices (of sizes 5×5, 10×10, and 100×80) using both the algorithms.

For each test case, print

- the matrix shape,
- the relative error $\|\mathbf{QR} - \mathbf{A}\|_2 / \|\mathbf{A}\|_2$, where $\mathbf{Q}$ and $\mathbf{R}$ are your computed QR factors, and
- the condition number of the matrix, as found by `numpy.linalg.cond`.

(c) Use the above algorithm and `numpy.linalg.lstsq()` to solve a least squares data fitting problem.

The file `petrol_price_delhi.txt` contains prices of gasoline from Jan. 1, 2018 to Sep. 30, 2018 (source). Construct least squares problems to fit the data to polynomials of degree 1 to 5.

For each polynomial degree, and for each method, provide the following, clearly identifying the degree and the method used:

- the approximate polynomial, for example, in the format as shown below:

```
a * x^2 + b *x + c
a = 0.324234
b = 1.34234
c = 3.2374
```

- the relative residual $\|Ax - b\|_2/\|b\|_2$ of the least squares problem solved.

For each of the two methods above, also provide a plot of the data and the five fitted polynomials (one plot per method).

Provide an evaluation:

- Do the methods differ in the relative error obtained? If so, which one is more accurate?
- Do the polynomial degrees differ in the relative error obtained? If so, which one provides the best approximant? Why?

Hint: You may use the following code to read the data into an array.

```
import numpy as np
v = np.loadtxt("petrol_price_delhi.txt", delimiter="\n")
```

Problem 3: Eigenvalue finding

(20 + 20 + 10 + 10 + 15 = 75 points)

In each of the following, consider the iteration to be converged if the 2-norm of the relative difference of subsequent iterates changes by 10^{-12} or less.

(a) Implement inverse iteration with a shift to compute the eigenvalue nearest to 2, and a corresponding 2-norm normalized eigenvector, of the matrix

$$A = \begin{bmatrix} 6 & 2 & 1 \\ 2 & 3 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

Run your program using random starting vectors with 10 different (but deterministic) random seeds. Print the final estimate of the eigenvalue, the eigenvector, and the number of iterations.

(b) Write a program to implement Rayleigh quotient iteration for computing an eigenvalue and corresponding 2-norm normalized eigenvector of a matrix. Test your program on the matrix given in part (a) using the same random starting vectors as in that part.

Print the final estimate of the eigenvalue, the eigenvector, and the number of iterations.

(c) In comparing the eigenvalue estimates produced by the two previous parts, what do you observe?

(d) Use `scipy.linalg.eig()` to compute all of the eigenvalues and eigenvectors of the matrix, and compare the results with those obtained in parts (a) and (b).

Modify your code for (a) and (b) so that it finds the eigenvalue output by `eig` which is closest to your approximate value, and using the value from `eig` as the 'true' value, outputs the relative error in the eigenvalue and the relative error of the eigenvector in the 2-norm.

(e) Using $\begin{bmatrix}1 & 4 & 2\end{bmatrix}^T$ as the starting vector, print and compare the computed eigenvalue and the computed eigenvector of inverse iteration and Rayleigh quotient iteration. You may assume either value to be the true value. Also print and compare the number of iterations each method takes for convergence. Which one converges more rapidly?

Problem 4: Principal component analysis

(10 + 40 + 25 + 25 = 100 points)

Principal component analysis is a statistical procedure that can be used to find the 'main axes' of a data set. Here, data set refers to a set of points in n-dimensional space, where the idea is that most of the data lies 'around' these main axes. See the accompanying figure 1 for an intuitive idea of what this means.

In the language of statistics, this helps identify variables that vary together, or, literally, are *correlated*. It also provides so-called 'principal components', which are orthogonal to one another and uncorrelated.

For a data set $(x_i)_{i=1}^N \subset \mathbb{R}^n$, PCA can be performed as follows:

(1) Compute the empirical mean:

$$\mathbf{u} := \frac{1}{N}\sum_{i=1}^{N} x_i.$$

(2) Remove the mean:

$$\widetilde{x}_i := x_i - u, \qquad i = 1, \dots, N.$$

(3) Assemble the 'data matrix':

$$Y := \frac{1}{\sqrt{N-1}} \begin{bmatrix}\widetilde{x}_1 & \widetilde{x}_2 & \cdots & \widetilde{x}_N\end{bmatrix}$$

(4) Compute the SVD (using `numpy.linalg.svd`) of Y:

$$U\Sigma V^T = Y.$$

The principal components as shown in figure (1) can now be computed as the columns of $U\Sigma$.

See this Overview and the Nature Methods article linked at start for more on PCA.

Questions:

(a) Plot the 2D data set generated by this function:

```python
def make_data(dims=2, npts=3000):
    np.random.seed(13)

    mix_mat = np.random.randn(dims, dims)
    mean = np.random.randn(dims)

    return np.dot(
           mix_mat,
           np.random.randn(dims, npts)) + mean[:, np.newaxis]
```

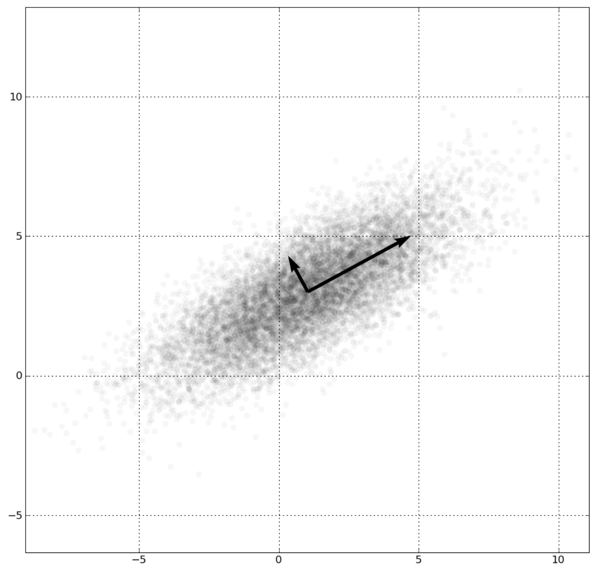

Figure 1: Principal components of a two-dimensional data set.

(b) Compute the principal components of this same data set and draw them into the same plot as the result from (a), centered at the mean. The figure you obtain should look similar to Figure 1.

Hints:

- Use `matplotlib.pyplot.gca().set_aspect("equal")` to make sure Matplotlib does not distort angles in your figure.
- Use `matplotlib.pyplot.arrow()` to draw arrows into the figure. Draw one for each principal component.

(c) From U, Σ, and V^T as returned by `numpy`, reconstruct the matrix $\widetilde{Y}$:

$$\widetilde{Y} = U\Sigma V^T.$$

Print the relative error $\|\widetilde{Y} - Y\|_2/\|Y\|_2$. (This should come out to about machine precision.)

(d) Now set the bottom right nonzero entry of Σ to zero. Call the resulting matrix as Σ' and reconstruct another matrix Y' as:

$$Y' = U\Sigma' V^T.$$

Next, undo the transformations done to Y' to get back to plain data (undo scaling by $1/\sqrt{N-1}$, re-add mean) and make a new plot with the principal components and the data set obtained from Y'.

What have you just accomplished? How could this be useful? (*Hint:* Consider a scenario where some component of the data comes from noise. Also consider how much memory is required to represent the 'reduced' data set.)

Problem 5: QR iteration with shifts

(30 + 10 + 10 = 50 points)

Write a Python function to implement the following version of QR iteration with shifts for computing the eigenvalues of a general real matrix A.

```
for $i$ in $n, \dots, 2$ do
    repeat
        $\sigma = a_{i,i}$ (use corner entry as shift)
        Compute QR factorization $\boldsymbol{QR} = \boldsymbol{A} - \sigma \boldsymbol{I}$
        $\boldsymbol{A} = \boldsymbol{RQ} + \sigma \boldsymbol{I}$
    until $\|A[i-1, : i-1]\| <$ tol
end for
```

You may use `np.linalg.qr` for computing the QR factorization. Test your function for the following two matrices:

$$A_1 = \begin{bmatrix} 2 & 3 & 2 \\ 10 & 3 & 4 \\ 3 & 6 & 1 \end{bmatrix}, \quad A_2 = \begin{bmatrix} 6 & 2 & 1 \\ 2 & 3 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

and use `tol` $= 10^{-16}$. Compare the computed eigenvalues with ones computed using `numpy.linalg.eigvals`.

Note: You should incorporate the following code snippet in your program to print your computed and NumPy computed eigenvalues.

```
def qr_iteration(A, tol):
# Your implementation goes here

A_1 = # Matrix 1
eigenvalues_1 = qr_iteration(A_1.copy(), tol)
print "Computed eigenvalues: ", eigenvalues_1
print "Actual eigenvalues: ", np.linalg.eigvals(A_1)
```

Problem 6: Lanczos iteration and convergence of Ritz values

(25 + 25 + 50 = 100 points)

(a) Prove that, in the case of A symmetric and real-valued, Arnoldi iteration (Algorithm 4.9 in the book) reduces to Lanczos iteration (Algorithm 4.10 in the book).

Specifically, show that the Hessenberg matrix H generated by Arnoldi iteration becomes symmetric and tridiagonal, with entries

$$H = Q^T A Q = \begin{bmatrix} \alpha_1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \beta_2 & \alpha_3 & \ddots & \\ & & \ddots & \ddots & \beta_{n-1} \\ & & & \beta_{n-1} & \alpha_n \end{bmatrix}.$$

Specifically, show that:

(1) H is symmetric.

(2) H is zero above the second super-diagonal.

(3) Deduce that only two iterations (not necessarily the first two) of the orthogonalization loop in Algorithm 4.9 need to be carried out.

Hint: You may use the fact that $\boldsymbol{H}$ is upper Hessenberg, as we showed in the lecture.

(b) Write a Python function to implement Lanczos iteration. Your function should take as input a matrix `A` and return the orthogonal matrix `Q` that is a basis for the Krylov subspace and the tridiagonal matrix `H`.

To test your function, generate a random symmetric matrix of size 100×100 as $B + B^T$ for another random matrix B. Print the following norms derived from the output of your routine:

- $\|QQ^T - I\|_2$,
- $\|Q^T AQ - H\|_2 / \|A\|_2$.

Both should be small.

(c) In this part, again let $n = 100$. Set up a semi-random real symmetric matrix $A \in \mathbb{R}^{n\times n}$ with eigenvalues $\lambda = 1, 2, \ldots, n$. You can do this as follows:

1. Generate a random matrix $B \in \mathbb{R}^{n\times n}$ using `np.random.rand`.
2. Compute its QR factorization: $B = QR$.
3. Set up a diagonal matrix D with diagonal entries $1, 2, \ldots, n$ using `np.diag`.
4. Set $A = QDQ^T$.

Perform Lanczos iteration on A and obtain the Ritz values γ_i for each iteration $i = 1, \ldots, n$. You may use `np.linalg.eigvals` to compute the Ritz values as the eigenvalues of the matrix T_i.

To see graphically how the Ritz values behave as iterations proceed, construct a plot with the iteration number on the vertical axis and the Ritz values at each iteration on the horizontal axis. Plot each pair (γ_i, k), $i = 1, \ldots, k$, as a discrete point on each iteration k. As iterations proceed and the number of the Ritz values grows correspondingly, you should see vertical "trails" of Ritz values converging on the true eigenvalues.

General Submission Guidelines

- **This assignment should be answered individually.**
- **You will be penalized for copying or any plagiarism with an automatic zero.**
- **Start working on your solutions early and don't wait until close to due date.**
- The points for each problem is roughly indicative of effort needed for answering that problem. For instance, 50 points can be taken to mean about 50 minutes of *focussed work*. This can include reading and thinking if the problem is theoretical. For machine problems, this will include time spent in coding, debugging and producing output assuming some Python competency. Your mileage may vary!
- IIIT-Delhi academic policies on honesty and integrity apply to all HWs. This includes not copying from one another, from the internet, a book, or any other online or offline source. A repeat offense will be reported to academic administration.
- You can ask questions about concepts, ideas and basics of coding on the Piazza discussion group for the class. You can also ask the instructor questions during office hours. You should however provide your own solutions.
- Any technical question pertaining to the HW emailed to me will be ignored.
- If you discuss or read secondary sources (other than class notes), please list all your discussion partners and/or secondary sources in your writeup. Failure to do so will constitute violation of honor code.
- Each problem should have a clear and concise write-up.
- Please clearly show all steps involved in theoretical problems.
- The answer to all theoretical problems, and output, figures and analyses for computer problems should be submitted in a PDF file or turned in handwritten.
- Handwritten submissions can be turned in during class, in person in my office or under my office door before the submission deadline. I will be present in my office during and for 5 minutes after every submission deadline to collect submissions.
- No late submission will be allowed unless I explicitly permit it.
- All Python files for computer problems should be submitted through Backpack.
- However, if your Python code generates an output figure or table, please provide all such results in a single PDF file along with your code submission.
- You will need to write a separate Python code for each problem and sometimes for each subproblem as well. You should name each such file as `problem_`n`.py` where n is the problem number. For example, your files could be named `problem_3a.py`, `problem_3b.py` and `problem_4.py`.
- You should import Python modules as follows:

```python
from __future__ import division
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
import numpy.linalg as npla
import scipy.linalg as spla
```

Every code you write will have one or more of these import statements.