M3/4/5N9 Computational Linear Algebra
Project 3 (60% of the final mark [excluding mastery component for 4th years/MSc)
Due January 12th 2020 (must submit on Blackboard)

Prof Colin Cotter

Autumn Term 2019

Reminders

1. Be sure to follow all project guidelines (separate document on Blackboard).
2. Write your code in Python. You may use the `numpy`, `scipy` and `matplotlib` Python modules to complete the tasks unless otherwise indicated.
3. In terms of code, marks will be given for: clear, accessible, readable, re-useable code that makes sensible use of `numpy` array operations, and is well-organised into suitable files.
4. In terms of reporting, marks will be given for: clear, appropriate description that demonstrates understanding of the material.

1 Eigenvalue algorithms

In this section we will do some experiments with the tridiagonal matrix

$$A = \begin{pmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & \ddots & & \ddots & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{pmatrix} \tag{1}$$

1. Construct this matrix in CSR matrix format using
```
import numpy as np
import scipy.sparse as sp diagonals = [2*np.ones(n), -1*np.ones(n-1), -1*np.ones(n-1)]
A = sp.csr_matrix(sp.diags(diagonals, [0, -1, 1]))
```
   Write a function to implement Rayleigh Quotient Iteration for A. You may use the function `scipy.sparse.linalg.spsolve` to solve sparse matrix systems.
2. Using the FLOPS model given in lectures, derive the approximate cost of this algorithm.
3. Investigate the behaviour of this algorithm for different choices of initial vector. What do you notice about the accuracy of the iterative solution - can you explain it from lectures?
4. What determines which eigenvalue of A is computed? Can you design an experiment to verify this? Provide the design and the results.
5. Compute the QR factorisation of the matrix A by transforming it to a dense matrix and using `numpy.linalg.qr` (which uses Householder reflections). What do you notice about the structure of the Q and R matrices? Explain this using your knowledge of how Householder reflections work. Comment on the efficiencies available (or lack thereof) of using sparse matrices in the QR algorithm when the QR factorisation is computed using Householder reflections.

6. **For the rest of this section, we will use dense matrices.**
   Implement the pure QR algorithm for A, keeping track of the diagonal values of A at each iteration. Describe what you observe, relating to what you have learned from lectures.

7. Implement the QR algorithm with constant shift μ. Namely, at each iteration, compute the QR factorisation of $A - \mu I$. Then, set $A = RQ + \mu I$, before repeating. As you did in the previous question, keep track of the diagonal values of A at each iteration. Describe what you observe, relating to what you have learned from lectures.

8. Construct a random $n \times n$ matrix A with independent normally distributed entries, and then multiply by A^T to obtain a random symmetric matrix S. Apply the pure QR algorithm to S, keeping track of the diagonal entries as before. Now reduce S to a tridiagonal matrix S_3 by zeroing all the values outside the main three diagonals. Apply the pure QR algorithm to S_3, keeping track of the diagonal entries as before. What do you observe? What do you observe about the magnitude of the entries away from the main diagonal, and how does this relate to what you observe?

2 GMRES and preconditioning

This section is about fast iterative methods for solving the linear system arising from numerical discretisation of the partial differential equation

$$-\frac{\partial^2 p}{\partial x^2} - \frac{\partial^2 p}{\partial y^2} = f, \quad p(0,y) = p(1,y) = p(x,0) = p(x,1) = 0. \tag{2}$$

We approximate the equation by first writing $p_{i,j} = p(i\Delta x, j\Delta x)$, $i,j = 1,\ldots,N$, with $\Delta x = 1/(N+1)$ (and similar for $f_{i,j}$). Then the centred difference approximation to Equation (2) is

$$4p_{i,j} - p_{i-1,j} - p_{i+1,j} - p_{i,j-1} - p_{i,j+1} = \Delta x^2 f_{i,j}, \quad i,j = 1,\ldots,N, \tag{3}$$

where we take $p_{0,j} = p_{N+1,j} = p_{i,0} = p_{i,N+1} = 0$.

1. Equation (3) is a linear system for the $p_{i,j}$ values and hence can be expressed as a matrix-vector system

$$Ax = b, \tag{4}$$

   where $x = (p_{1,1}, p_{2,1}, \ldots, p_{N,1}, p_{1,2}, p_{2,2}, \ldots, , p_{N,2}, \ldots, p_{1,N}, \ldots, p_{N,N})^T$. Describe the structure and values of A and b.

2. Write code to construct matrix A in CSR format, using `scipy.sparse.csr_matrix`. For randomly generated b, use `scipy.sparse.gmres` to solve the equation using GMRES (without preconditioner). Study the convergence properties for different values of N, explaining the results using what you have learned from lectures.

3. We will now use an alternative iterative method to solve this system. The idea is to split the matrix into $A = M + N$, with M arising from the linear system

$$2p_{i,j} - p_{i-1,j} - p_{i+1,j} = \Delta x^2 f_{i,j}, \quad i,j = 1,\ldots,N, \tag{5}$$

   whilst N arises from the linear system

$$2p_{i,j} - p_{i,j-1} - p_{i,j+1} = \Delta x^2 f_{i,j}, \quad i,j = 1,\ldots,N. \tag{6}$$

   We then take an initial guess $x = x^0$, and iterate according to

$$(\gamma I + M)x^{n+1/2} = (\gamma I - N)x^n + b, \tag{7}$$
$$(\gamma I + N)x^{n+1} = (\gamma I - M)x^{n+1/2} + b, \tag{8}$$

   where $\gamma > 0$ is a chosen real parameter.
   Show that if the algorithm converges, then it converges to the solution of $Ax = b$.

4. Show that each of the two equations in this iterative method can be solved by solving N independent tridiagonal matrix systems of size $N \times N$. Hence derive the approximate cost in FLOPS of one iteration.

5. Implement this algorithm as code, using `csr_matrix` data types for all matrices. You may use `scipy.sparse.spsolve` for solving sparse tridiagonal systems.

6. It can be shown that the optimal convergence rate (optimising an upper bound, not the actual rate) for this algorithm is obtained when $\gamma = \sqrt{\mu_{\min}\mu_{\max}}$, where $\mu_{\max}$ is the maximum over magnitudes of eigenvalues for both M amd N, and $\mu_{\min}$ is the minimum. Investigate this using your code.

7. Now we use the algorithm ((7)-(8)) as a preconditioner for GMRES. The idea of the preconditioner is that during each iteration GMRES computes a vector r_k, and the preconditioner computes z_k by solving $\hat{A}z_k = r_k$ where $\hat{A}$ is some approximation of A. In this project, we will relate r_k to z_k by solving

$$(\gamma I + M)y = r_k, \tag{9}$$
$$(\gamma I + N)z_k = (\gamma I - M)y + r_k, \tag{10}$$

which amounts to one iteration of ((7)-(8)) with $b = r_k$ and $x_0 = 0$.
Write a function `iterate(r_k)` that takes in r_k and returns z_k according this procedure. We can pass this preconditioner to the scipy implementation of GMRES in `scipy.sparse.linalg.gmres` by
`M=scipy.sparse.linalg.LinearOperator((n,n), matvec=iterate)`
where n is the size of the vector x (equal to N^2). We can now call GMRES via
`x = scipy.sparse.linalg.gmres(A, b, M=M)`
Investigate the convergence behaviour of GMRES with varying N as a result of using this preconditioner, and comment on the results.

3 Image denoising

This section builds upon the previous section.
We now consider 2D bitmap greyscale images, represented as an $N \times N$ grid of boxes, with $p_{i,j}$ is the intensity of the box i points from the left, j points from the bottom. A value of 1 indicates white, 0 indicates black, and intermediate values indicate grey.

1. Write code to set $p_{i,j}$ values for the function

$$p(x,y) = \begin{cases} 1 & \text{if } |x - 0.5| < 0.25 \text{ and } |y - 0.5| < 0.25 \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

Note that the denoising algorithms in this section all assume that the image is black on the boundary. Add normally distributed independent random perturbations with variance $\sigma^2 = 0.1^2$ and visualise the output using `matplotlib.pyplot.pcolor` with the `gray` colour map.
Our goal is to remove this noise.

2. One approach to denoising is to minimise the following function,

$$\sum_{i=0}^{N}\sum_{j=1}^{N}\left(\frac{p_{i+1,j} - p_{i,j}}{\Delta x}\right)^2 + \sum_{i=1}^{N}\sum_{j=0}^{N}\left(\frac{p_{i,j+1} - p_{i,j}}{\Delta x}\right)^2 + \mu\sum_{i=1}^{N}\sum_{j=1}^{N}\left(p_{i,j} - \hat{p}_{i,j}\right)^2, \tag{12}$$

over $p_{i,j}$, $i,j = 1,\ldots,N$, where $\hat{p}_{i,j}$ are the noisy image values, and where $\mu > 0$ is a penalty parameter. This minimisation produces a trade off between being close to the noisy image values, and being smooth. It can be shown that the minimum as attained at the solution of

$$(4 + \mu\Delta x^2)p_{i,j} - p_{i-1,j} - p_{i+1,j} - p_{i,j-1} - p_{i,j+1} = \mu\Delta x^2\hat{p}_{i,j}, \quad i,j = 1,\ldots,N. \tag{13}$$

Adapt your GMRES solver from the previous section to solve this equation, and visualise the results for different values of μ, explaining what you see, and commenting on the number of iterations required for GMRES convergence and how that relates to μ.

3. If the image is piecewise constant (like our example image), a better method is to minimise the following function,

$$\sum_{i=0}^{N}\sum_{j=1}^{N}\left|\frac{p_{i+1,j}-p_{i,j}}{\Delta x}\right| + \sum_{i=1}^{N}\sum_{j=0}^{N}\left|\frac{p_{i,j+1}-p_{i,j}}{\Delta x}\right| + \mu\sum_{i=1}^{N}\sum_{j=1}^{N}\left(p_{i,j}-\hat{p}_{i,j}\right)^2. \tag{14}$$

This is no longer a quadratic function, so the minimum can no longer be found by solving a linear system. A popular iterative algorithm for finding the minimum starts by introducing extra variables $d^h_{i,j}$ for $i = 0,\dots,N$, $j = 1,\dots,N$, and $d^v_{i,j}$ for $i = 1,\dots,N$, $j = 0,\dots,N$, and seeking the minimum over the p, d^h and d^v variables of the function

$$\sum_{i=0}^{N}\sum_{j=1}^{N}\left|d^h_{i,j}\right| + \sum_{i=1}^{N}\sum_{j=0}^{N}\left|d^v_{i,j}\right| + \mu\sum_{i=1}^{N}\sum_{j=1}^{N}\left(p_{i,j}-\hat{p}_{i,j}\right)^2, \tag{15}$$

subject to the constraints

$$\Delta x d^h_{i,j} = p_{i+1,j}-p_{i,j}, \quad \Delta x d^v_{i,j} = p_{i,j+1}-p_{i,j}. \tag{16}$$

To do this, we introduce further variables $b^h_{i,j}$ for $i = 0,\dots,N$, $j = 1,\dots,N$, and $b^v_{i,j}$ for $i = 1,\dots,N$, $j = 0,\dots,N$. We also select a parameter $\lambda > 0$. Then, starting from $p = 0$, $b^h = d^h = 0$, $b^v = d^v = 0$, we iterate through the following steps:

(a) Solve

$$\begin{aligned}(4\lambda+\mu\Delta x^2)p_{i,j} - \lambda p_{i-1,j} - \lambda p_{i+1,j} - \lambda p_{i,j-1} - \lambda p_{i,j+1} = {} & \mu\Delta x^2\hat{p}_{i,j} \\ & - \lambda\Delta x(d^h_{i,j} - d^h_{i-1,j} - b^h_{i,j} + b^h_{i-1,j}) \\ & - \lambda\Delta x(d^v_{i,j} - d^v_{i,j-1} - b^v_{i,j} + b^v_{i,j-1}),\end{aligned} \tag{17}$$

for $i, j = 1,\dots,N$.

(b) Set

$$d^h_{i,j} = \operatorname{shrink}\left(b^h_{i,j} + \frac{p_{i+1,j}-p_{i,j}}{\Delta x}, 1/\lambda\right), \quad i = 0,\dots,N, j = 1,\dots,N, \tag{18}$$

$$d^v_{i,j} = \operatorname{shrink}\left(b^v_{i,j} + \frac{p_{i,j+1}-p_{i,j}}{\Delta x}, 1/\lambda\right), \quad i = 1,\dots,N, j = 0,\dots,N, \tag{19}$$

where $\operatorname{shrink}(x,\gamma) = \frac{x}{|x|}\max(|x|-\gamma, 0)$.

(c) Set

$$b^h_{i,j} = b^h_{i,j} + \left(-d^h_{i,j} + \frac{p_{i+1,j}-p_{i,j}}{\Delta x}\right), \quad i = 0,\dots,N, j = 1,\dots,N, \tag{20}$$

$$b^v_{i,j} = b^v_{i,j} + \left(-d^v_{i,j} + \frac{p_{i,j+1}-p_{i,j}}{\Delta x}\right), \quad i = 1,\dots,N, j = 0,\dots,N, \tag{21}$$

and then repeat these steps until convergence. Note that λ is an algorithm parameter, and affects convergence rates but not the end result if convergence is achieved.

Implement this algorithm and investigate the roles of λ and μ. For an advanced level investigation, you could include other images.