M3/4/5N9 Computational Linear Algebra
Project 2 (20% of the final mark)
Due December 3rd 2019 (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 Conditioning and stability

1. Randomly generate an orthogonal matrix Q and and a triangular matrix R by:

   (a) Randomly generating a 20×20 matrix B with independent normally distributed entries, computing the QR factorisation of B and keeping the Q matrix (throw the R matrix away).

   (b) Independently generating another 20×20 matrix C with independent normally distributed entries, and extracting the upper triangular part as R.

   Now form $A = QR$. We now have a matrix A for which we know the exact QR factorisation (up to small rounding errors in computing the product QR. Use the built in `numpy.linalg.qr` function to compute the QR factorisation of A using the Householder algorithm, returning Q_2, R_2.
   Compute the following quantities:

   (a) $\|Q_2 - Q\|$,

   (b) $\|R_2 - R\|$,

   (c) $\|Q_2 R_2 - A\|$.

   Explain your results using what you know about the backward stability of the Householder algorithm.

   [15 marks]

2. Repeat this exercise for the Singular Value Decomposition, which writes $A = U\Sigma V^T$ where U and V are orthogonal matrices and Σ is diagonal with non-negative entries. In other words, randomly generate U, V and Σ with these properties, form the corresponding A, and then use the built in `numpy.linalg.svd` function to compute the QR factorisation, returning U_2, V_2, Σ_2. Making similar comparisons to the above, what do your results suggest?

   [15 marks]

Hints: the functions `numpy.triu`, `numpy.linalg.norm`, `numpy.random.randn` will all be useful for this section.

2 LU factorisation of banded matrices

Banded matrices are an example of sparse matrices, which contain many zeros. The Compressed Sparse Row (CSR) format is a method of storing sparse matrices that only stores the non-zero entries (plus their locations). This is implemented by `scipy.sparse.csr_matrix` from the `scipy` Python module.

1. Using what you have learned about banded matrices from lectures, write a function `L, U = bandedLU(M, ml, mu)` that takes a CSR matrix M, plus the lower and upper bandwidths m_l and m_u, and returns the LU factorisation as CSR matrices, computed from your implementation of Gaussian elimination. Your implementation should:

   - Make use of `numpy` vector array operations where possible,
   - Avoid unnecessarily computing zeros,
   - Work for complex data types (we'll be using them in the project).

   It is not necessary to implement pivoting.

   [15 marks]

2. Provide some tests that demonstrate that the LU factorisation has worked.

   [15 marks]

3 Exponential integrators

In this section we will focus on the wave equation

$$u_{tt} - u_{xx} = 0, \quad u(0) = u(H) = 0, \tag{1}$$

in the domain $[0, H]$. We approximate this equation with a finite difference approximation,

$$\frac{\mathrm{d}^2}{\mathrm{d}t^2}u_k - \frac{u_{k-1} - 2u_k + u_{k+1}}{\Delta x^2} = 0,\ k = 1, \ldots, N, \quad u_0 = u_{N+1} = 0, \tag{2}$$

where $\Delta x = H/(N+1)$, and u_k is our approximation to $u(k\Delta x)$.

1. Writing $v = (u_1, \ldots, u_N)^T \in \mathbb{R}^N$, obtain the matrix K such that

   $$\frac{\mathrm{d}^2}{\mathrm{d}t^2}v + Kv = 0. \tag{3}$$

   [2 marks]

2. Introducing $w = \frac{\mathrm{d}v}{\mathrm{d}t}$, show that

   $$\frac{\mathrm{d}U}{\mathrm{d}t} = LU, \tag{4}$$

   where $U = (v^T, w^T)^T$, and

   $$L = \begin{pmatrix} 0 & I \\ -K & 0 \end{pmatrix}. \tag{5}$$

   [2 marks]

3. It is known that K is a positive-definite matrix. Show that all of the eigenvalues of L are imaginary.

   [2 marks]

4. The exact solution to Equation (4) satisfies

$$U(T) = \exp(TL)U(0), \tag{6}$$

where $\exp(TL)$ is the matrix exponential of TL. For matrices L with imaginary eigenvalues $\lambda_k = i\mu_k$, $k = 1,\ldots,N$, the action $\exp(TL)U(0)$ of the matrix exponential $\exp(TL)$ on $U(0)$ can be computed from

$$U(T) \approx \textstyle\sum_{j=1}^{J} \beta_j U_j, \tag{7}$$

where U_j is the solution of

$$(\alpha_j I + TL)U_j = U(0), \tag{8}$$

and where α_j and β_j are some known complex coefficients for $j = 1,\ldots,J$, provided by the function `rexi_coefficients.RexiCoefficients` in the supplied file. This function returns arrays of α and β coefficients given $h \in \mathbb{R}^+$, $M \in \mathbb{N}^+$, where $hM = 1.1T|\mu_{\max}|$ and $\mu_{\max}$ is the eigenvalue of L with maximum magnitude. The approximation converges as M goes to infinity with hM fixed.

Remark: the advantage of this formulation is that the Equation (8) *can be solved in parallel over all the values of* $j = 1,\ldots,J$. *This means that parallel computation can be used to take a much larger* T *in a shorter time than is possible with standard timestepping methods. However, we shall not explore any parallel aspects in this project.*

Writing $U_j = (v_j^T, w_j^T)^T$, show that w_j can be eliminated from Equation (8) to produce a banded matrix system for v_j, and give the upper and lower bandwidths.

[5 marks]

5. Given the initial condition $u(x,0) = \exp(-(x-5)^2/0.2) - \exp(-125)$, $u_t(x,0) = 0$, and taking $H = 10$, use this method to compute the solution at time $T = 2.5$, using your banded solver to obtain the values of u on the grid.

[20 marks]

6. Verify that your code works by comparing the solution with standard explicit second-order Runge Kutta timestepping with several steps sufficiently small Δt, exploring the dependence on h and M.

[9 marks]