Computational Linear Algebra Project 2

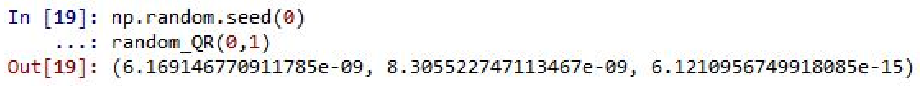
Jonathan Cheung
CID: 01063446
Lecturer: Professor Colin Cotter
The contexts of this report and associated codes are my own work unless otherwise stated and references to sources must be provided if they are used.

1st December 2019

1 Conditioning and stability

The function 'random_QR' will take as its input a mean, standard deviation and dimension and returns the residuals when the built in numpy.linalg.qr function is used to factorise matrices. More specifically, the function generates two matrices with random entries from the normal distribution with mean and standard deviation specified in the input. The dimension of the matrix can also be specified, but it is set to 20x20 by default. These matrices are then factorised to generate a Q matrix and a R matrix, which have been generated independently from each other. Q and R are multiplied to generate A, so A has a known QR factorisation. A is then factorised using np.linalg.qr and 'random_QR' will return the following 2-norms $\|Q_2 - Q|$, $\|R_2 - R\|, \|Q_2R_2 - A\|$.

Running the line of code 'random_QR(0,1)' will generate an array of three values like in figure 1.



```
In [19]: np.random.seed(0)
    ...: random_QR(0,1)
Out[19]: (6.169146770911785e-09, 8.305522747113467e-09, 6.1210956749918085e-15)
```

Figure 1: An example output from random_QR

We can run the function 'random_QR_test' by inputting a number of iterations, mean, and standard deviation. This function will run the previously described 'random_QR' function for the specified number of iterations with the specified mean and standard deviation. It then will output an array of these residuals; each row corresponds to one iteration. Finally, the function will plot a graph with lines representing these different errors for Q,R, and QR. An example output of this function with the seed set to 0 will return figure 2. The residuals have been taken to log base 10 so that the graph is easier to read. In this graph we see that the Q and R residual fluctuate between order -7 and -13. However the order of the QR residual is a lot more constant around the -14 mark.

This is due to the fact that the householder algorithm is backwards stable. The calculated Q_2 and R_2 values can be different to the true Q and R values, whilst the calculated Q_2 R_2 will is of insignificant difference to the true A value. To further demonstrate this, run the function 'accuracy_check' by running lines of code 70-71 in CLA_P2_Q1. This function will carry out the similar steps to 'random_QR_test' but will then add some random values of order 1e-10 so that the Q3 and R3 errors are of the same magnitude as Q2 and R2, but you will then see that the difference between Q_3 R_3 and A is approximately 14, a significant magnitude. This demonstrates how effective the householder algorithm is at recovering errors.

The function random_SVD takes as its input a mean (default value 0), standard deviation (default value 1), and dimension (default value 20). This function is similar to random_QR; its first three outputs are the residuals of U_2, σ_2, V_2,and $U_2 \Sigma V_2$. Setting the seed to 0 and running the function with mean 0 and standard deviation 1 gives the 5 residuals. The errors for U_2 and V_2 are large. However, upon further inspection we notice that the absolute value of the calculated U_2 and V_2 is very close to the true U and V. So we have generated a U_3 and V_3 which have had the signs adjusted to match those of U and V. The fourth and fifth outputs of the function are the residuals for U_3 and V_3 and these are of insignificant size; they have order of magnitude 1e-13. The third output is the residual for $U_2 \Sigma V_2$. This has order of magnitude 1e-14. We can see that the SVD algorithm is very numerically stable. We can perturbate U, V, and sigma like we did for the calculated QR so that they are in the same magnitude of error and find out that the final residual is of a much bigger error. Again, SVD is good at recovering errors

2 LU factorisation of banded matrices

The function 'bandedLU' in CLA_P2_Q2 takes a csr matrix, a lower and upper bandwidth and then returns the LU factorisation of this matrix.

Running lines 61-69 of code will do a quick check on a complex matrix and show that this function returns two matrices that multiply back to get the original B matrix, and that the L and U are upper and lower triangular. Lines 75 to 102 defines a function 'bandedLUtest' that will generate random matrices and test that the previous function works.

3 Exponential integrators

1) We can find a matrix that represents the second derivative with respect to time on the vector v. From equation (2), this is the same as finding the matrix that represents the following equation.

$Kv = -\frac{u_k - 2u_k + u_{k+1}}{\Delta x^2}$
This matrix is as follows:

$$K = \begin{bmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & -1 \\ 0 & \dots & 0 & -1 & 2 \end{bmatrix}$$

2) We can introduce w = dv/dt so that

$$LU = \begin{pmatrix} 0 & I \\ -K & 0 \end{pmatrix} U = \begin{pmatrix} 0 & I \\ -K & 0 \end{pmatrix} \begin{pmatrix} v \\ w \end{pmatrix} = \begin{pmatrix} w \\ -Kv \end{pmatrix} = \begin{pmatrix} \frac{dv}{dt} \\ \\ \frac{d^2v}{dt^2} \end{pmatrix} = \frac{dU}{dt}$$

3) λ is an eigenvalue of L if it satisfies the equation $LV = \lambda V$ where v is the eigenvector $V = \frac{v_1}{v_2}$, so

$$LV = \begin{pmatrix} v_2 \\ -kv_1 \end{pmatrix} = \begin{pmatrix} \lambda v_1 \\ v_2 \end{pmatrix}$$

From the top component of this system of linear equations we conclude that the eigenvector is of the form $v = \begin{pmatrix} v_1 \\ \lambda v_1 \end{pmatrix}$

$$L \begin{pmatrix} v_1 \\ \lambda v_1 \end{pmatrix} = \lambda \begin{pmatrix} v_1 \\ \lambda v_1 \end{pmatrix}$$
$$\begin{pmatrix} \lambda v_1 \\ -Kv_1 \end{pmatrix} = \begin{pmatrix} \lambda v_1 \\ \lambda^2 v_1 \end{pmatrix}$$

Focusing on the bottom half of the equation, we multiply both sides by v_1^T to get $-v_1^T K v_1 = v_1^T \lambda^2 v_1$ K is positive definite, so the left side of of the equation is less than zero. Thus, λ must be imaginary. Otherwise, if lambda was not, the right side of the equation would be greater than zero and contradict the left side.

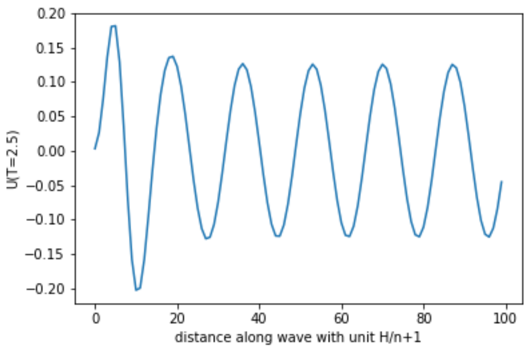
4) The exact solution to $\dfrac{dU}{dt} = LU$ satisfies $U(T) = exp(TL)U(0)$
, where $U(T) \approx \sum_{j=1}^{J} \beta_j U_j$ and $(\alpha_j I + TL)U_j = U(0)$ where α_j, β_j are coefficients from the RexiCoefficients function for j=1,...,J.

We now show that we can eliminate w_j from the above equation to form a banded matrix system for v_j.
$(\alpha_j + TL)\begin{pmatrix} v_j \\ w_j \end{pmatrix} = U(0) = \begin{pmatrix} v(0) \\ w(0) \end{pmatrix}$
$\begin{pmatrix} \alpha_j v_j + T w_j \\ \alpha_j w_j - kT v_j \end{pmatrix} = \begin{pmatrix} v(0) \\ w(0) \end{pmatrix}$

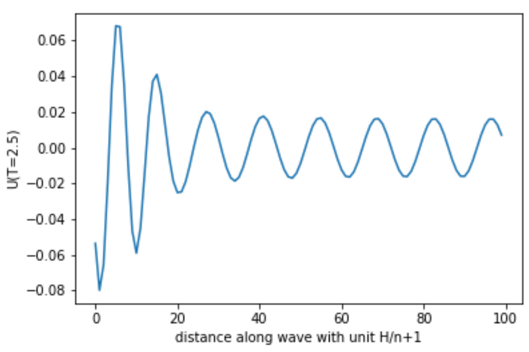
Scaling the separate equations by α_j and T
$\alpha_j^2 v_j + T\alpha_j w_j = \alpha_j v(0)$
$T\alpha_j w_j - T^2 K v_j = Tw(0)$
We can now eliminate w_j
$\alpha_j^2 v_j + T^2 K v_j = \alpha_j v(0) - Tw(0)$
$(\alpha_j^2 I + T^2 K)v_j = \alpha_j v(0) - Tw(0)$

where the matrix $\alpha_j^2 + T^2 K$ has lower and upper bandwidth 1. This comes from the K matrix having these bandwidths.

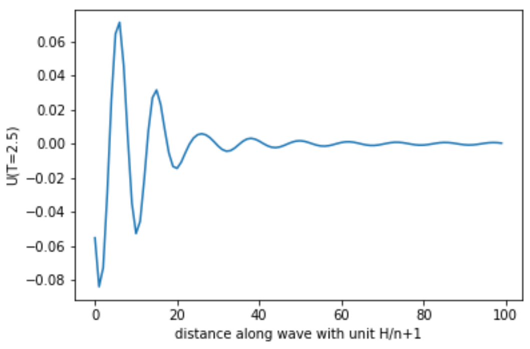
4) in file 'CLA_P2_Q3' there is a function 'wave_solve'. which will take as its input an M, n, T, and H. T and H have been set as default to the initial conditions given in the question. This function then calculates the maximum eigenvalue of the matrix L and uses this to calculate an h such that the condition for hM is satisfied. These h and M values are inputted into the RexiCoefficients function which returns a column of alpha coefficients and a column of beta coefficients. The alpha values are used to generate banded systems of equations. The solution of which is multiplied with the respective beta coefficient to find our solution. So running



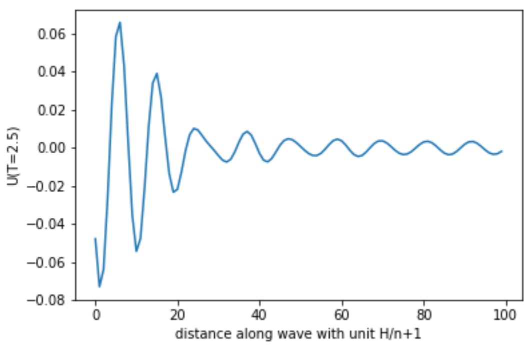
(a) m=5



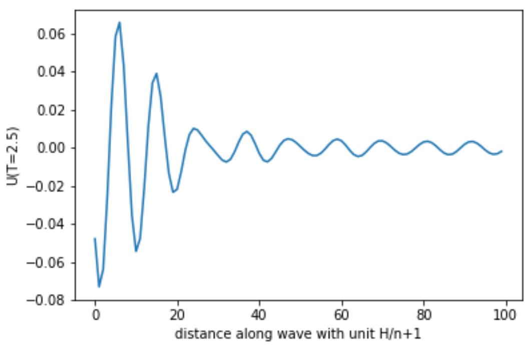
(b) m=6



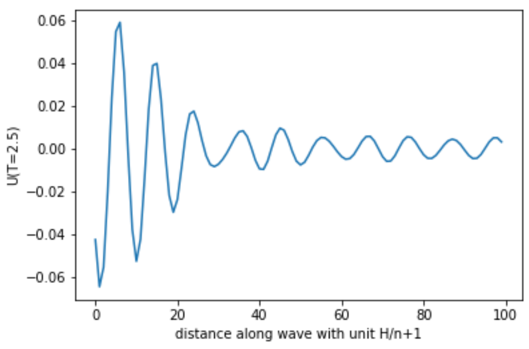
(c) M=8



(d) m=10



(e) m=20



(f) m=30

Figure 2: Waves for increasing m, with N set as 100, T=2.5, H = 10

We can see that as M is increasing, the wave is converging to some waveform. This agrees with the theory that the solution converges as M converges to infinity. Though, we do not need particularly large M to see what the shape of the wave should be. In fact, for M=30, the wave has already mostly settled down.

The function 'rungekutta' uses a timestepping method to calculate a solution to the wave equation.

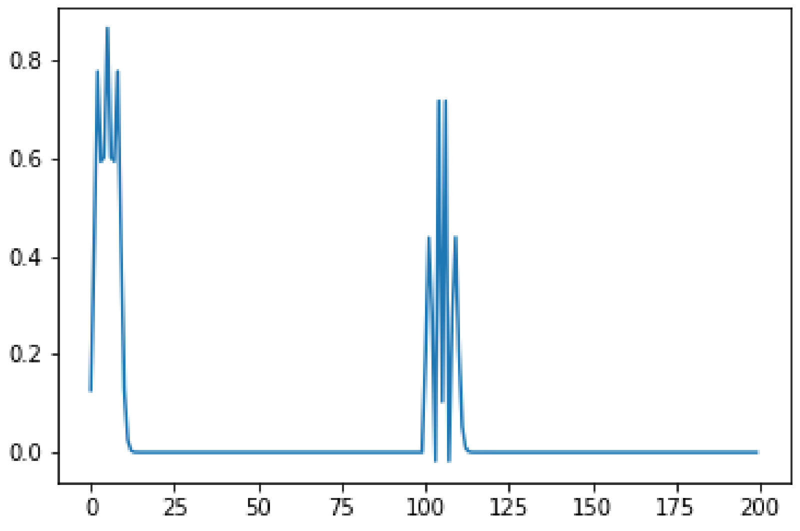


Figure 3: The solution from the Runge-Kutta method, m=30, n=100, t=0.1

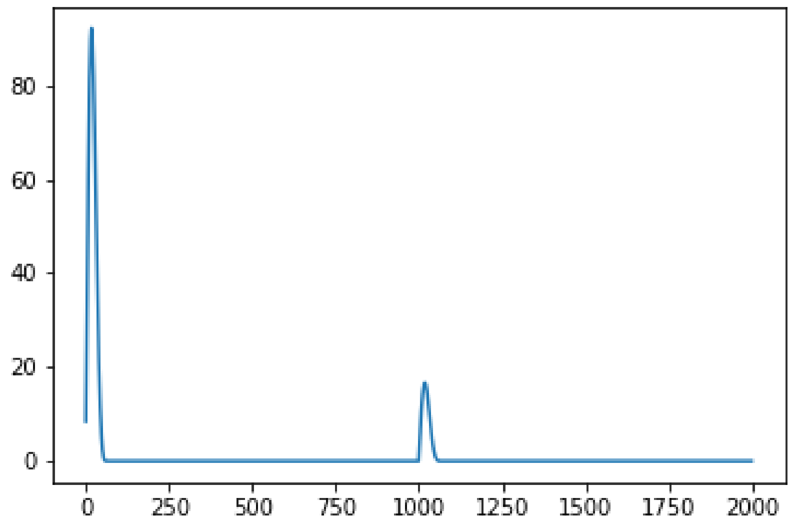


Figure 4: The solution from the Runge-Kutta method, m=1000, n=1000, t=0.1