MCSC 6020G - Numerical Analysis
Assignment 1

Parikshit Bajpai

Question 1

(a) Skew-symmetric matrix

A square matrix A is called skew-symmetric if $A^T = -A$ i.e. $a_{ij} = -aji$. A general example of such a matrix is:

$$A = \begin{bmatrix} 0 & \lambda_{11} & \dots & \lambda_{1n} \\ -\lambda_{11} & 0 & \dots & \lambda_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -\lambda_{1n} & -\lambda_{2n} & \dots & 0 \end{bmatrix}$$

where, $\lambda_{ij} \in \mathbb{R}$.

A specific example of a skew-symmetric matrix is as follows:

$$A = \begin{bmatrix} 0 & \pi & \sqrt{2} & \sqrt{5} \\ -\pi & 0 & -\sqrt{3} & -e \\ -\sqrt{2} & \sqrt{3} & 0 & \sqrt{7} \\ -\sqrt{5} & e & -\sqrt{7} & 0 \end{bmatrix}$$

(b) Orthogonality

To prove. *If B is a skew-symmetric matrix, then $A = (\mathbb{I} + B)(\mathbb{I} - B)^{-1}$ is orthogonal, where $\mathbb{I}$ is an identity matrix.*

Proof. For a matrix A to be orthogonal, $A^T A = 1$, i.e., $A^T = A^{-1}$. For a skew-symmetric matrix B, we can define $A = (\mathbb{I} + B)(\mathbb{I} - B)^{-1}$. Then,

$$\begin{aligned}
A^T &= \left((\mathbb{I} + B)\,(\mathbb{I} - B)^{-1}\right)^T \\
&= \left((\mathbb{I} - B)^{-1}\right)^T (\mathbb{I} + B)^T && \left(\because (XY)^T = X^T Y^T\right) \\
&= \left((\mathbb{I} - B)^T\right)^{-1} (\mathbb{I} + B)^T && \left(\because (X^{-1})^T = (X^T)^{-1}\right) \\
&= \left(\mathbb{I}^T - B^T\right)^{-1} \left(\mathbb{I}^T + B^T\right) && \text{(Distributivity)} \\
&= (\mathbb{I} + B)^{-1} (\mathbb{I} - B) && \left(\because B^T = -B\right) \\
&= (\mathbb{I} - B)\,(\mathbb{I} + B)^{-1} && \left(\text{Commutativity}^1\right) \\
&= A^{-1}
\end{aligned}$$

■

[1] $(I+B)^{-1}$ and $(I-B)$ are simultaneously diagonalisable matrices and, in such cases, matrix multiplication is commutative.

Question 2

To prove. *For a complex-valued vector* $v \in \mathbb{C}^n$, $\frac{1}{n}\|v\|_1 \leq \|v\|_\infty \leq \|v\|_2$

Proof. Let $\|v\|_\infty = \max_i |v_i|$ and $\|v\|_p = \left(\sum_i |v_i|^p\right)^{\frac{1}{p}}$

$$\begin{aligned} subsectionname\|v\|_p &= \|v\|_\infty \frac{(\sum_i |v_i|^p)^{\frac{1}{p}}}{\|v\|_\infty} \\ &= \|v\|_\infty \left(\sum_i \frac{|v_i|^p}{\|v\|_\infty^p}\right)^{\frac{1}{p}} \\ &= \|v\|_\infty \left(\sum_i \left(\frac{|v_i|}{\|v\|_\infty}\right)^p\right)^{\frac{1}{p}} \\ &\leq \|v\|_\infty n^{\frac{1}{p}} \qquad \left(\because \left(\frac{|v_i|}{\|v\|_\infty}\right)^p \leq 1, \forall i\right) \end{aligned}$$

Thus we have[2]

$$\|v\|_\infty \leq \|v\|_p \leq \|v\|_\infty n^{\frac{1}{p}}$$

So, taking $p = 1$ and $p = 2$, we get

$$\begin{aligned} \|v\|_\infty \leq \|v\|_1 \leq \|v\|_\infty n \qquad & (p = 1) \\ \|v\|_\infty \leq \|v\|_p \leq \|v\|_\infty \sqrt{n} \qquad & (p = 2) \end{aligned}$$

Therefore, from the above two inequalities,

$$\frac{1}{n}\|v\|_1 \leq \|v\|_\infty \leq \|v\|_2 \qquad \blacksquare$$

Example of a vector satisfying the left equality:

$$V = \begin{bmatrix} 3+4i & 4-3i & 5 \end{bmatrix}$$

$$V = \begin{bmatrix} 1-7i & 5+5i & -\sqrt{50} & \sqrt{50}i \end{bmatrix}$$

Example of a vector satisfying the right equality:

$$V = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

$$V = \begin{bmatrix} i & 0 & 0 & 0 \end{bmatrix}$$

Example of a vector simultaneously satisfying both the equalities:

$$V = \begin{bmatrix} C \end{bmatrix} \ \forall C \in \mathbb{C}$$

[2] $\|v\|_\infty = \max_i |v_i|$ while the other norms can be expressed as $\|v\|_p = \max_i |v_i| + C$ where $C \in \mathbb{R}_{\geq 0}$. Therefore, $|v\|_\infty \leq \|v\|_p$.

Question 3

(a) Determinant of a matrix using LU factorisation

LU factorisation results in two triangular matrices. Since determinant respects matrix multiplication, LUP factorizarion can be used to find the determinant of a matrix as shown:

$$\begin{aligned} PA &= LU \\ \det(PA) &= \det(LU) \\ \det(P)\det(A) &= \det(L)\det(U) \\ \det(A) &= \begin{cases} -\det(L)\det(U) & \text{Odd number of row exchanges}[3] \\ \det(L)\det(U) & \text{Even number of row exchanges} \end{cases} \\ &= \begin{cases} -\prod_{i=1}^{n} l_{ii} \prod_{j=1}^{n} u_{jj} & \text{Odd number of row exchanges}[4] \\ \prod_{i=1}^{n} l_{ii} \prod_{j=1}^{n} u_{jj} & \text{Even number of row exchanges} \end{cases} \end{aligned}$$

The pseudo-code for the implementation of the above method has been shown in algo. 1.

Complexity. *The order of complexity of determinant computation using LUP decomposition is $\mathcal{O}(N^3)$, where N is the dimension of matrix.*

Proof. The order of complexity of LUP factorisation is well known as $\mathcal{O}(N^3)$. The determinant of a triangular matrix is the product of its diagonal elements resulting in $(n-1)$ FLOPS. Therefore, after the LUP factorisation, the number of FLOPS for computing the determinant equals $[2(n-1)+2]$ (computing two determinant of triangular matrix, multiplying them and multiplying by parity).

The highest order of complexity of the complete algorithm results from the LUP factorisation step and thus the order of complexity of the algorithm itself is $\mathcal{O}(N^3)$. ■

(b) Determinant of a matrix using Cofactor expansion

If A is a $n \times n$ matrix with elements $a_{ij}, i, j \in \{1, 2, ..., n\}$, its determinant is a weighted sum of the determinants of n sub-matrices (or minors) of A, each of size $(n-1) \times (n-1)$.

$$\det(A) = \sum_{i=1}^{n} (-1)^{i+j} a_{ij} M_{i,j} = \sum_{j=1}^{n} (-1)^{i+j} a_{ij} M_{i,j}$$

where, $M_{i,j}$ represents the minor matrix formed by eliminating row i and column j from matrix A. The pseudo-code for the implementation of the cofactor expansion method has been shown in algo. 2.

Complexity. *The order of complexity of determinant computation using cofactor expansion is $\mathcal{O}(N!)$, where N is the dimension of matrix.*

Proof. The recurrence relation for the above algorithm can be written as $T(n) = nT(n-1) + (2n-1)$. Using recursion, we can show the following

$$\begin{aligned} T(n) &= n(n-1)T(n-1) + n(n-1) \\ &= n(n-1)(n-2)T(n-3) + o(n^3) \\ &\vdots \\ &= Cn! + o(n!) \end{aligned}$$

Therefore, the order of complexity of cofactor expansion is $\mathcal{O}(N!)$. ■

[3] Determinant of the permutation matrix is -1 if there's an odd number of row exchanges and 1 if there's an even number of row exchanges.

[4] Determinant of a triangular matrix is equal to the product of its diagonal elements.

Algorithm 1: Determinant of a matrix using LUP factorisation

Data: Matrix, A
Result: Determinant of matrix, $\det(A)$

Function `determinant_LUP(`*Matrix* A`):` `/* Computes determinant */`

Input: Matrix (A)
Output: Determinant ($\det A$)

Data: Lower triangular matrix (L), Upper triangular matrix (U), Permutation matrix (P), Parity (par)

`/*` *par* `accounts for the number of row exchanges during LUP factorisation and is equal to 1 for even no. of exchanges and -1 for odd no. of exchanges. */`

L, U, P, $par \leftarrow$ `LUP_factorisation(`A`)` `/* Call function for LUP factorisation */`
$determinant \leftarrow par \times$ `determinant_triangular(`L`)` $\times$ `determinant_triangular(`U`)`

return *determinant*

Function `determinant_triangular(`*Matrix* A`):` `/* Computes det of triangular matrix */`

Input: Triangular matrix (A)
Output: Determinant ($\det A$)

Initialize:
$n \leftarrow$ Size of A `/* Size of matrix computed using numpy */`
$determinant \leftarrow 1$ `/* Stores determinant */`
r_i & $r_j \leftarrow 0$ `/* Indices for minor matrix */`

for $i = 1$ **to** n **do** `/* Loop over rows of` A `*/`
  $determinant \leftarrow determinant * a_{ii}$ `/* Product of diagonal elements */`

return *determinant*

Function `LUP_factorisation(`*Matrix* A`):` `/* Performs LUP factorisation */`

Input: Triangular matrix (A)
Output: Lower triangular matrix (L), Upper triangular matrix (U), Permutation matrix (P), Parity (par)

`/*` *par* `accounts for the number of row exchanges during LUP factorisation and is equal to 1 for even no. of exchanges and -1 for odd no. of exchanges. */`

`/* The details of LUP factorization have been skipped for brevity. */`

return *determinant*

Algorithm 2: Determinant of a matrix using cofactor expansion

Data: Matrix, A
Result: Determinant of matrix, $\det(A)$

Function `determinant_cofactor(`*Matrix A*`):` `/* Recursively computes determinant */`

Input: Matrix (A)
Output: Determinant ($\det A$)

Initialize:
$n \leftarrow$ Size of A `/* Size of matrix computed using numpy */`
$det\_2 \leftarrow 0$ `/* Stores determinant of 2×2 matrix */`
$determinant \leftarrow 0$ `/* Stores determinant of n×n matrix */`

if $n = 2$ **then**
  $det\_2 \leftarrow a_{11}a_{22} - a_{21}a_{12}$
  return det_2
else
  for $i = 1$ **to** n **do** `/* Cofactor expansion about first row */`

$$determinant \leftarrow determinant + (-1)^{i+1} \times a_{1i} \times \texttt{determinant\_cofactor(comp\_minor(}A,n,1,i\texttt{))}$$

  return *determinant*

Function `comp_minor(`*Matrix A, Size n, Row r, Column c*`):` `/* Computes minor matrix */`

Input: Matrix (A), Size of A (n), Row (r), Column (c)
Output: Minor matrix ($M_{r,j}$)

Initialize:
$minor \leftarrow$ Zero matrix of size $(n-1) \times (n-1)$ `/* Stores minor matrix */`
r_i & $r_j \leftarrow 0$ `/* Indices for minor matrix */`

for $i = 1$ **to** n **do** `/* Loop over rows of A */`
  $r_j \leftarrow 0$ `/* Reset column index of minor at start of each row */`
  if $i = r$ **then**
    Continue to next iteration `/* Omitting the row r */`
  else
    for $i = 1$ **to** n **do** `/* Loop over columns of A */`
      if $j = c$ **then**
        Continue to next iteration `/* Omitting the column c */`
      else
        $minor_{r_i r_j} \leftarrow A_{ij}$ `/* Assigning elements to minor matrix */`
        $r_j \leftarrow r_j + 1$
    $r_i \leftarrow r_i + 1$

return *minor*

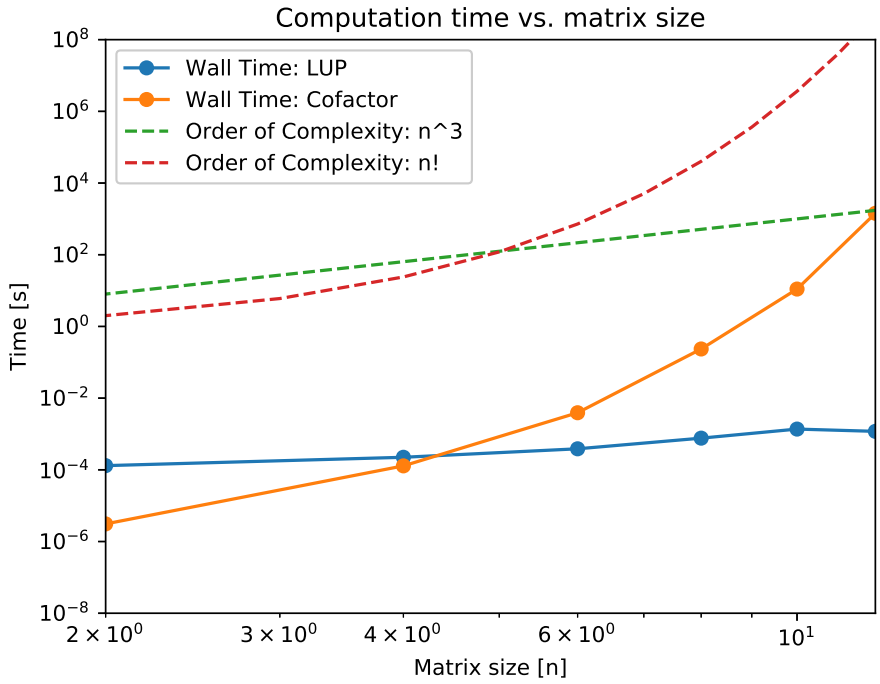

Figure 1: Wall times vs. matrix size for the Cofactor expansion and LUP factorisation methods.

(c) Computations

The cofactor expansion and LU factorisation algorithms discussed above were tested and compared for multiple matrix sizes and the obtained results are discussed hereon. As shown in and fig. 1, the cofactor expansion method is quick for very small matrices but becomes very slow as the matrix increases. The LUP factorisation method on the other hand is extremely fast. In fact, LUP marginally outperforms cofactor expansion for a 4×4 matrix and observed to be slower than cofactor method only for 2×2 matrix where the cost of performing the LUP factorisation is significantly higher than the couple of FLOPS required for finding determinant by cofactor expansion. This behaviour is as expected from the order of complexity calculation shown above and the trend can be observed in fig. 1 where the computation times of each method follow the orders of complexity trend shown by the dashed lines[5]. Furthermore, the computation of determinant of large matrices using the cofactor method takes insanely long limiting the feasibility of the method to matrices of upto 12×12 on the machine used for computations.

For the purpose of analysis, the determinant values obtained through the two algorithms were compared with the results computed using `numpy` which uses the `LAPACK` subroutine `dgetrf`. The absolute error was then defined as

$$e_{abs} = |\det(A)_{method} - \det(A)_{numpy}|$$

As shown in fig. 2, the absolute errors obtained through LUP factorisation were around 2 orders higher than the errors obtained through cofactor expansion. This can be attributed to the rounding errors during the LUP factorisation step. Since, cofactor matrix does not involve factorisation step, the errors due to factorisation due not arise and the errors in determinants are lower compared to LUP factorisation. However, the errors in both the cases were less than the order of precision of floating point numbers. The

[5]The computation time for LUP factorisation of 12×12 matrix seems to be slightly lower than the 10×10 matrix. This anomalous result might be attributed to the randomness of the matrix but the behaviour was present over multiple runs and is still unexplained.

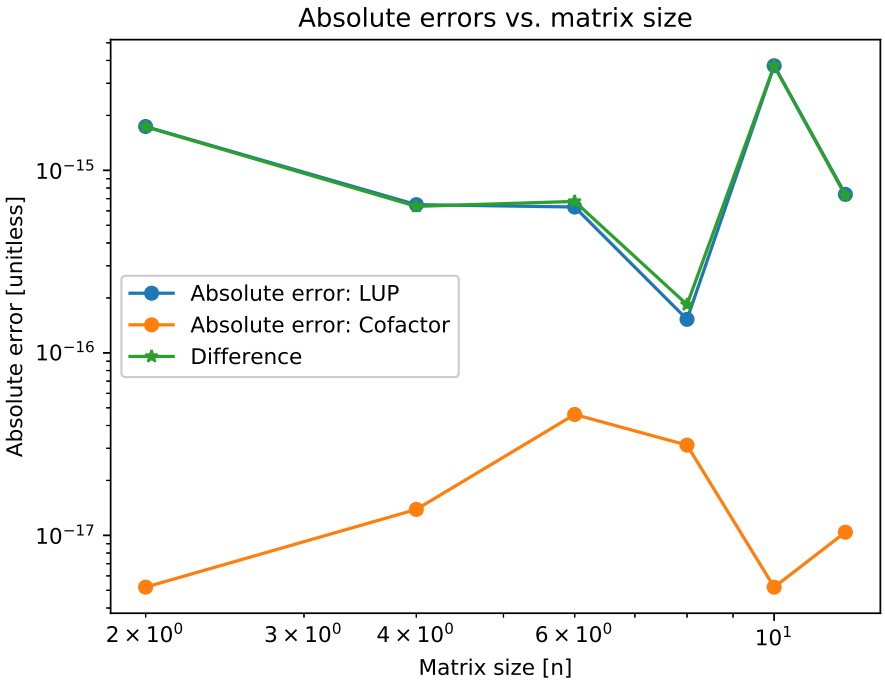

Figure 2: Absolute error (w.r.t. `numpy`) vs. matrix size for the Cofactor expansion and LUP factorisation methods. The difference between the results from the two methods is also shown.

same arguments hold for the relative error which is defined as

$$e_{rel} = \frac{|\det(A)_{method} - \det(A)_{numpy}|}{|\det(A)_{numpy}|}$$

The relative errors have been shown in fig. 3. It must be mentioned that due to the relatively small number of feasible computations, a notable error trend wasn't observed.

The LUP factorisation method does not face the computational limitations observed in the cofactor expansion method and therefore the computations were performed for matrices of sizes upto 256×256. The results are shown in figs.4–6. The computation time follows the $\mathcal{O}(N^3)$ trend and the absolute error grows exponentially with matrix size.

Summary of results

The order of complexity for the cofactor expansion method was demonstrated to be $\mathcal{O}(N!)$ while that for LUP factorisation was demonstrated to be $\mathcal{O}(N^3)$. As a result, LUP factorisation results in much faster calculation of the determinant of a matrix albeit at the cost of accuracy. However, the errors are well below the precision of floating point numbers which makes LUP factorisation computationally more efficient. The theoretically foreseen behaviour was observed in the computational results thus *a-posteriori* verifying the theoretical proofs.

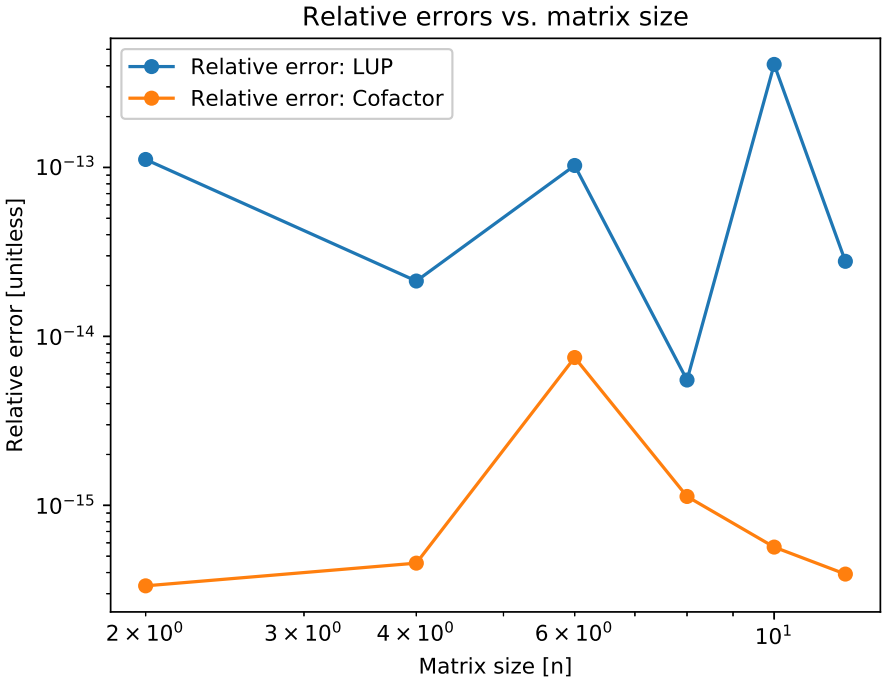

Figure 3: Relative error (w.r.t. `numpy`) vs. matrix size for the Cofactor expansion and LUP factorisation methods.

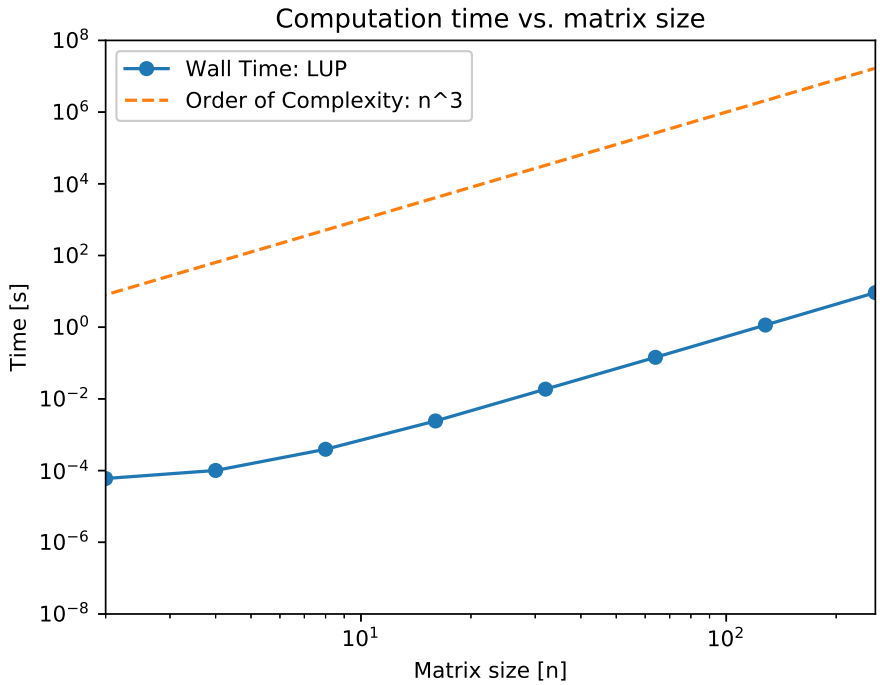

Figure 4: Wall times vs. matrix size for the LUP factorisation method.

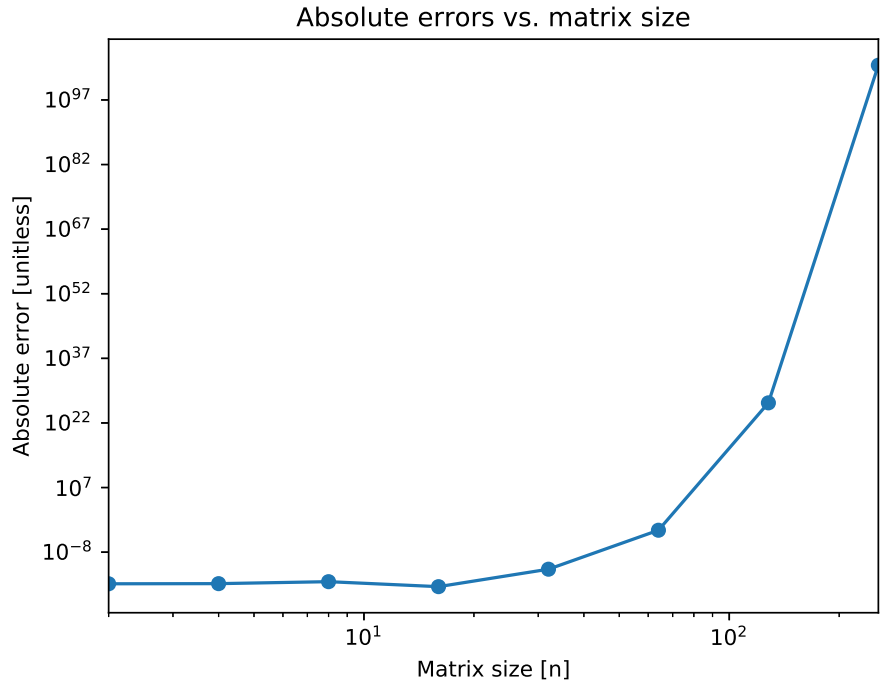

Figure 5: Absolute error (w.r.t. `numpy`) vs. matrix size for the LUP factorisation method.

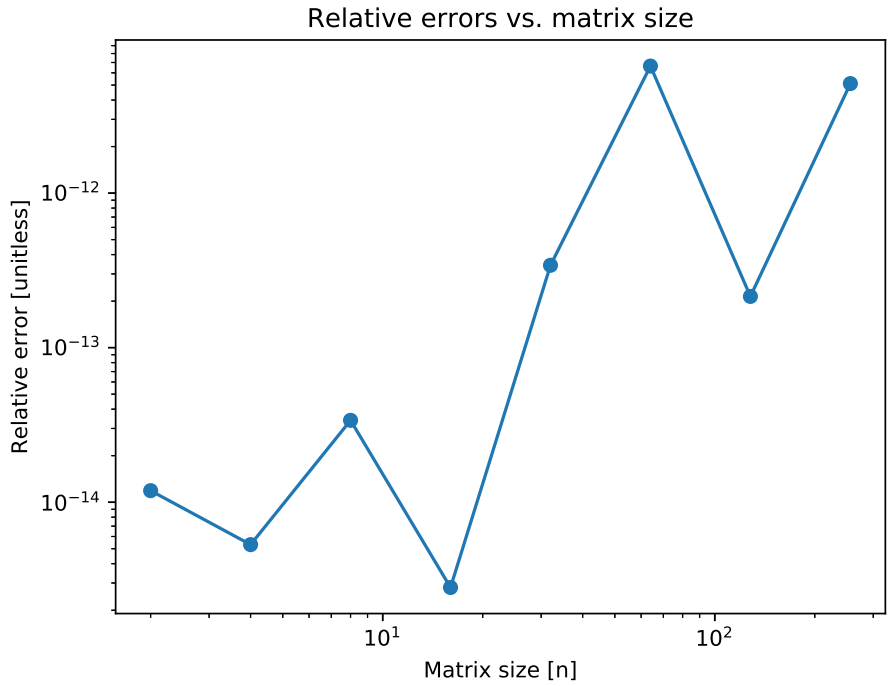

Figure 6: Relative error (w.r.t. `numpy`) vs. matrix size for the LUP factorisation method.