1. Calculate the number of flops required for computing a matrix inverse as a function of n where $n \times n$ is the size of the matrix. Consider two algorithms.

   (a) **Algorithm 1.** Define an $n \times 2n$ matrix $M := (A, I)$ where I is the $n \times n$ identity matrix. Subject M to row operations to transform it into a matrix of the form (I, B). Then $B = A^{-1}$. Write this algorithm as a pseudocode. For simplicity assume that pivoting is not needed (anyway, row swaps do not involve flops). Calculate the number of flops. An answer of the form $W(n) = Cn^p + O(n^{p-1})$ is good enough. You need to determine the constants C and p.

Solution: We give the following implementation here.

```
1  def inv(A):
2    n = len(A)
3    A_aug = np.concatenate((A, np.identity(n)), axis=1)
4
5    for i in range(n):
6      scale = A_aug[i][i]
7      for j in range(2 * n):
8        A_aug[i][j] /= scale
9      for k in range(n):
10       scale2 = A_aug[k][i]
11       if k != i:
12         for j in range(2 * n):
13           A_aug[k][j] -= scale2 * A_aug[i][j]
14   return A_aug[:, n:]
```

Observe that line 13 contributes 2 flops, and runs $2n$ times. Hence, the line 9 loop contributes $4n^2$ flops. Further, since line 8 contributes 1 flops, and also runs $2n$ times. Then the line 5 loop requires $4n^2 + 2n$ flops per iteration and hence $4n^3 + 2n^2$ flops in total. Hence, we find that $C = 4$ and $p = 3$.

   (b) **Algorithm 2.** Decompose A to $A = LU$. The cost of this is $W_1(n) = \frac{2}{3}n^3 + O(n^2)$. Compute L^{-1} and U^{-1} and calculate $A^{-1} = U^{-1}L^{-1}$. Write this algorithm as a pseudocode. For simplicity assume that pivoting is not needed. Start it with calling the LU algorithm (you do not need to write a pseudocode for LU, just add its cost to your result). Calculate the number of flops. An answer should be of the form $W(n) = Cn^p + O(n^{p-1})$. You need to determine the constants C and p.

Solution: We give the following implementation here.

```
def tril_inv(L):
  n = len(L)
  L_inv = np.zeros((n, n))

  for i in range(n):
    L_inv[i][i] = 1 / L[i][i]
    for j in range(i):
      dot = 0
      for k in range(j, i):
        dot -= L[i][k] * L_inv[k][j]
      dot /= L[i][i]
      L_inv[i][j] = dot
```

```
13     return L_inv
14
15 def triu_inv(U):
16     return tril_inv(U.T).T
17
18 def tri_matmul(U, L):
19     n = len(U)
20     A = np.zeros((n, n))
21
22     for i in range(n):
23         for j in range(n):
24             dot = 0
25             for k in range(max(i, j), n):
26                 dot += U[i][k] * L[k][j]
27             A[i][j] = dot
28     return A
29
30 def inv_lu(A):
31     _, L, U = scipy.linalg.lu(A)
32     return tri_matmul(triu_inv(U), tril_inv(L))
```

We first compute the number of flops required by `tril_inv`. Line 10 contributes 2 flops and line 11 contributes 1; hence, the line 7 loop contributes $2(i-j)+1$ flops per iteration and hence

$$\sum_{j=1}^{i} 2(i-j)+1 = 2i^2 + i - 2\sum_{j=1}^{i} j = 2i^2 + i - i(i+1) = i^2$$

flops in total, so `tril_inv` on the whole requires

$$\sum_{i=1}^{n} 1 + i^2 = n + \frac{1}{6}n(n+1)(2n+1) = \frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{7}{6}n$$

flops. It is clear that `triu_inv` will require the same number. It remains to compute the number of flops for `tri_matmul`, that is,

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=\max(i,j)}^{n} 2 = 2\sum_{i=1}^{n}\sum_{j=1}^{n}(n-\max(i,j)+1) = 2n^3 + 2n^2 - 2\sum_{i=1}^{n}\sum_{j=1}^{n}\max(i,j)$$

noting that by properties of triangle numbers,

$$\begin{aligned} 2\sum_{i=1}^{n}\sum_{j=1}^{n}\max(i,j) &= 2\sum_{i=1}^{n}\left(\sum_{j=1}^{i} i + \sum_{j=i+1}^{n} j\right) = 2\sum_{i=1}^{n}\left(i^2 + \sum_{j=1}^{n} j - \sum_{j=1}^{i} j\right) \\ &= \cdots = \frac{1}{3}n(n+1)(4n-1) = \frac{4}{3}n^3 + n^2 - \frac{1}{3}n \end{aligned}$$

and hence `tri_matmul` requires

$$2n^3 + 2n^2 - \left(\frac{4}{3}n^3 + n^2 - \frac{1}{3}n\right) = \frac{2}{3}n^3 + n^2 + \frac{1}{3}n$$

flops, and so the whole procedure to decompose $A = LU$, find L^{-1} and U^{-1}, and multiply $U^{-1}L^{-1}$ takes a total of

$$\frac{2}{3}n^3 + 2\left(\frac{1}{3}n^3\right) + \frac{2}{3}n^3 + O(n^2) = 2n^3 + O(n^2)$$

flops. Hence, we find that $C = 2$ and $p = 3$ by this method.

2. (a) Consider the set $\mathcal{L}$ of all $n \times n$ lower-triangular matrices with positive diagonal entries.

   i. Prove that the product of any two matrices in $\mathcal{L}$ is also in $\mathcal{L}$.

   ii. Prove that the inverse of any matrix in $\mathcal{L}$ is also in $\mathcal{L}$.

This means that the set of all $n \times n$ lower-triangular matrices with positive diagonal entries forms a group with respect to matrix multiplication.

Solution: First let $A, B \in \mathcal{L}$ and consider

$$(AB)_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj}.$$

But since A is lower-triangular, then $a_{ik} = 0$ when $k > i$ and similarly $b_{kj} = 0$ when $j > k$, and so

$$(AB)_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj} = \sum_{k=j}^{i} a_{ik}b_{kj}. \tag{1}$$

Hence, we see that if $j > i$, this sum is empty, and so $(AB)_{ij} = 0$, meaning AB is lower-triangular. It remains to show the diagonal entries of AB are positive; taking $j = i$ in (1), we see that

$$(AB)_{ii} = \sum_{k=i}^{i} a_{ik}b_{ki} = a_{ii}b_{ii}$$

and so since $a_{ii}, b_{ii} > 0$ since $A, B \in \mathcal{L}$ it follows that $(AB)_{ii} > 0$, and hence $AB \in \mathcal{L}$.

It remains to show that $\mathcal{L}$ is closed under inverses for all $n \in \mathbb{N}$. We proceed by induction on n. The case for $n = 1$ is clear, since $\begin{bmatrix} a \end{bmatrix}^{-1} = \begin{bmatrix} a^{-1} \end{bmatrix}$, which is trivially lower-triangular, and $a^{-1} > 0$ if $a > 0$. Now suppose the hypothesis holds for some $n \in \mathbb{N}$, and let A be an $(n+1) \times (n+1)$ lower-triangular matrix with positive diagonal entries.

We can then write A in block form like so

$$A = \begin{bmatrix} \tilde{A} & \mathbf{0} \\ \mathbf{b}^\top & c \end{bmatrix}$$

and since A is lower-triangular with positive diagonal entries, $\det(A) \neq 0$, so A is invertible. Let us write A^{-1} in block form as well

$$A^{-1} = \begin{bmatrix} \tilde{D} & \mathbf{e} \\ \mathbf{f}^\top & g \end{bmatrix}.$$

It now suffices to show that $\tilde{D}$ is lower-triangular and $\mathbf{e} = \mathbf{0}$; indeed

$$\begin{bmatrix} I_n & \mathbf{0} \\ \mathbf{0}^\top & 1 \end{bmatrix} = I_{n+1} = AA^{-1} = \begin{bmatrix} \tilde{A} & \mathbf{0} \\ \mathbf{b}^\top & c \end{bmatrix} \begin{bmatrix} \tilde{D} & \mathbf{e} \\ \mathbf{f}^\top & g \end{bmatrix} = \begin{bmatrix} \tilde{A}\tilde{D} & \tilde{A}\mathbf{e} \\ \mathbf{b}^\top \tilde{D} + c\mathbf{f}^\top & \mathbf{b}^\top \mathbf{e} + cg \end{bmatrix}$$

so in particular $\tilde{D} = \tilde{A}^{-1}$ and hence lower-triangular with diagonal entries by the inductive hypothesis. Further, we have that $\tilde{A}\mathbf{e} = \mathbf{0}$; Since $\tilde{A}$ is invertible, then it must be that $\mathbf{e} = \mathbf{0}$, and hence A^{-1} is lower-triangular.

Finally, to conclude the diagonal entries of A^{-1} are positive, we need only show that $g > 0$. And indeed, since $\mathbf{e} = \mathbf{0}$ and $c > 0$ by assumption,

$$g = \frac{1 - \mathbf{b}^\top \mathbf{e}}{c} = \frac{1}{c} > 0$$

and hence A^{-1} is lower-triangular with positive diagonal entries. By the principle of induction, $\mathcal{L}$ is closed under inverses for any $n \in \mathbb{N}$.

(b) Prove that the Cholesky decomposition for any $n \times n$ symmetric positive definite matrix is unique. *Hint. Proceed from converse. Assume that there are two Cholesky decompositions $A = LL^\top$ and $A = MM^\top$. Show that then $M^{-1}LL^\top M^{-\top} = I$. Conclude that $M^{-1}L$ must be orthogonal. Then use item (a) of this problem to complete the argument.*

Solution: Toward a contradiction, suppose for some $n \times n$ symmetric positive definite matrix A we have two Cholesky decompositions $A = LL^\top$ and $A = MM^\top$.

Then clearly $LL^\top = MM^\top$, and so by multiplying by M^{-1} on the left and by $M^{-\top}$ on the right, we obtain

$$M^{-1}LL^\top M^{-\top} = I.$$

But now observe that

$$M^{-1}LL^\top M^{-\top} = M^{-1}L(M^{-1}L)^\top$$

and hence $M^{-1}L(M^{-1}L)^\top = I$, meaning $M^{-1}L$ is orthogonal.

By item (a), we also have that $M^{-1}L$ is lower-triangular with positive diagonal entries (since L and M are each as such). Further, its inverse $(M^{-1}L)^{-1} = (M^{-1}L)^\top$ is also lower-triangular with positive diagonal entries. Then since both $M^{-1}L$ and $(M^{-1}L)^\top$ are lower-triangular, $M^{-1}L$ must be diagonal.

Finally, since it is orthogonal, its diagonal entries must satisfy $a_{jj}^2 = 1$. Since we know its diagonal entries are positive, this implies $a_{jj} = 1$, so $M^{-1}L = I$ and multiplying on the left by M, we find $L = M$, hence the Cholesky decomposition is unique.

3. The Cholesky algorithm is the cheapest way to check if a symmetric matrix is positive definite.

(a) Program the Cholesky algorithm. If any L_{jj} turns out to be either complex or zero, make it terminate with a message: "The matrix is not positive definite".

Solution: We give the following implementation of the Cholesky algorithm here.

```
def cholesky(A):
  n = len(A)
  L = np.zeros((n, n))

  for j in range(n):
    L[j][j] = A[j][j]
    for k in range(j):
      L[j][j] -= L[j][k] ** 2
    if L[j][j] <= 0:
      print("The matrix is not positive definite")
      return None
    L[j][j] = np.sqrt(L[j][j])
    for i in range(j + 1, n):
      L[i][j] = A[i][j]
      for k in range(j):
        L[i][j] -= L[i][k] * L[j][k]
      L[i][j] /= L[j][j]
  return L
```

(b) Generate a symmetric 100×100 matrix as follows: generate a matrix $\tilde{A}$ with entries being random numbers uniformly distributed in $(0, 1)$ and define $A := \tilde{A} + \tilde{A}^\top$. Use the Cholesky algorithm to check if A is symmetric positive definite. Compute the eigenvalues of A using a standard command (e.g. `eig` in MATLAB), find minimal eigenvalue, and check if the conclusion of your Cholesky-based test for positive definiteness is correct. If A is positive definite, compute its Cholesky factor using a standard command (e.g. see this help page for MATLAB) and print the norm of the difference of the Cholesky factors computed by your routine and by the standard one.

Solution: Our algorithm from item (a) returns "The matrix is not positive definite" and so we expect that the minimal eigenvalue $\lambda_{\min}$ of A should be such that $\lambda_{\min} \leq 0$. Indeed, we find that $\lambda_{\min} \approx -7.764$.

(c) Repeat item (b) with A defined by $A = \tilde{A}^\top \tilde{A}$. The point of this task is to check that your Cholesky routine works correctly.

Solution: Our algorithm from item (a) returns a Cholesky factor L suggesting A is symmetric positive definite. To verify our algorithm is correct, we also compute L_{std} via `np.linalg.cholesky` and find that $\|L - L_{\text{std}}\| \approx 1.128 \cdot 10^{-12}$.