1 Algorithme

```
#!/usr/bin/env python

from copy import deepcopy
from numpy import array, identity


def dilatation(A, i, ):
    """
    >>> mat = array([[1, 2, 3], [3, 4, 6], [1, 0, -1]], dtype=float)
    >>> dilatation(mat, 1, 4)
    array([[ 1.,  2.,  3.],
           [12., 16., 24.],
           [ 1.,  0., -1.]])
    """
    A[i] *=
    return A


def permutation(A, i, j):
    """
    >>> mat = array([[1, 2, 3], [3, 4, 6], [1, 0, -1]], dtype=float)
    >>> permutation(mat, 1, 2)
    array([[ 1.,  2.,  3.],
           [ 1.,  0., -1.],
           [ 3.,  4.,  6.]])
    """
    A[[i, j]] = A[[j, i]]
    return A


def transvection(A, i, j, ):
    """
    >>> mat = array([[1, 2, 3], [3, 4, 6], [1, 0, -1]], dtype=float)
    >>> transvection(mat, 0, 1, 2)
    array([[ 7., 10., 15.],
           [ 3.,  4.,  6.],
           [ 1.,  0., -1.]])
    """
    A[i] +=  * A[j]
    return A


def inversion_mat():
    transvection(mat, 0, -3)
    transvection(mat, 0, -1)
    transvection(mat, 1, -1)


def recherche_pivot(A, b, j):
    pivot = j
    for i in range(j, A.shape[0]):
        if abs(A[pivot, j]) < abs(A[i, j]):
```

```
            pivot = i
    if pivot != j:
        permutation(A, pivot, j)
        permutation(b, pivot, j)


def elimination_bas(A, b, j):
    for i in range(j + 1, A.shape[0]):
        c = -A[i, j]/A[j, j]
        A[i] += A[j] * c
        b[i] += b[j] * c


def descente(A, b):
    for j in range(A.shape[1] - 1):
        recherche_pivot(A, b, j)
        elimination_bas(A, b, j)


def elimination_haut(A, b, j):
    for i in range(j + 1, A.shape[0]):
        c = -A[i, j]/A[j, j]
        A[i] += A[j] * c
        b[i] += b[j] * c


def remontee(A, b):
    for j in range(A.shape[0] - 1, 0, -1):
        elimination_haut(A, b, j)


def solve_diagonal(A, b):
    for j in range(A.shape[0]):
        b[j] /= A[j, j]
        A[j, j] = 1
    return b


def gauss(A, b):
    descente(A, b)
    remontee(A, b)
    return solve_diagonal(A, b)


def inversion(A):
    """
    >>> mat = array([[1, 2, 3], [3, 4, 6], [1, 0, -1]], dtype=float)
    >>> inversion(mat)
    array([[ 0.        ,  0.33333333,  0.        ],
           [-0.        , -0.        , -0.75      ],
           [-2.        , -0.        , -0.        ]])
    """
    A_1 = deepcopy(A)
    return gauss(A_1, identity(A_1.shape[0]))
```

2 Complexité de la méthode

2.1 Coût de `descente`

- On itère sur chaque colonne (n)
  - `recherche_pivot`: On itère sur chaque ligne (n)
  - `elimination_bas`: On itère sur chaque ligne (n)
    - On transvecte sur chaque ligne (n)

D'où $O(n^3)$

2.2 Coût de `remontee`

- On itère sur chaque colonne (n)
  - `elimination_haut`: On itère sur chaque ligne (n)

D'où $O(n^2)$

2.3 Coût de `pivot_partiel`

oui (on l'a pas fait)

3 Peut-on faire mieux que $O(n^3)$?