## CP.2.3.1, Sauer3

For the $n$-by-$n$ matrix with entries $a_{ij} = 5/(i+2j-1)$, set $x = (1,\dots,1)^T$ and $b = Ax$. Use the Use the python program from CP.2.2.2 or Numpy's `numpy.linalg.solve` command to compute `x_c`, the double precision computed solution. Find the infinity norm of the forward error and the error magnification factor of the problem $Ax = b$,and compare it with the condition number of $A$: (a) $n = 6$, (b) $n = 10$.

Hint: Be careful with Python's indexing that starts at 0, and Sauer's and Matlab's indexing that starts at 1. With zero-based indexing the formula for $a_{ij}$ is $a_{ij} = 5/(i+2j+2)$. Here is a code snippet to generate $A$ with $n = 5$.

```
n = 5
A = np.zeros( [ n, n ], dtype=float )
for i in range(0,n):
  for j in range(0,n):
    A[i,j] = 5 / ( i + 2*j + 2.)
print(A)
```

```
[[2.66666667 2.         1.71428571 1.55555556 1.45454545]
 [2.25       1.83333333 1.625      1.5        1.41666667]
 [2.         1.71428571 1.55555556 1.45454545 1.38461538]
 [1.83333333 1.625      1.5        1.41666667 1.35714286]
 [1.71428571 1.55555556 1.45454545 1.38461538 1.33333333]]
```

## CP.2.3.1, Sauer3, solution, Langou

Colab: `https://colab.research.google.com/drive/1YjBvnQZNZYyeWpx5IR5pOM9AsOesFe4j`

We see that both our python code and `numpy.linalg.solve` are backward stable for this example. That the backward error returned is less than $2 \times 10^{-16}$ which is excellent. We do expect `numpy.linalg.solve` to be backward stable. That is, for any input $A$ and $b$, we expect the backward error to be of the order of machine precision. For our python code, we are using LU without pivoting and, while for this type of matrix, this seems to work, LU without pivoting is not a backward stable algorithm. So good backward error.
Now the condition number is about $10^8$ for $n = 6$ and about $10^{14}$ for $n = 10$.
We see that the forward error is about

$$\texttt{forward\_error} \approx \texttt{backward\_error} \times \texttt{condition\_number}$$

So despite having low backward error, the forward error is quite high. For $n = 6$, the forward error is of the order $10^{-9}$ (which is about $(10^8)\cdot(10^{-16})$.) For $n = 10$, the forward error is of the order $10^{-4}$ (which is about $(10^{14})\cdot(10^{-16})$.)
The relation

$$\texttt{forward\_error} \approx \texttt{backward\_error} \times \texttt{condition\_number}$$

is related to the fact that

$$\texttt{error\_magnification\_factor} = \texttt{forward\_error} \;/\; \texttt{backward\_error} \approx \texttt{condition\_number}$$

Note: we can prove that

$$\text{error\_magnification\_factor} = \text{forward\_error} \, / \, \text{backward\_error} \leq \text{condition\_number}$$

```
for n in [ 6, 10 ]:
  A = np.zeros( [ n, n ], dtype=float )
  for i in range(0,n):
    for j in range(0,n):
      A[i,j] = 5 / ( i + 2*j + 2. )
  x = np.ones( [ n, 1 ], dtype=float )
  b = A @ x

  xc = np.linalg.solve( A, b )

  relative_forward_error = np.linalg.norm( x - xc, np.inf )\
     / np.linalg.norm( x, np.inf )
  relative_backward_error = np.linalg.norm( b - A@xc, np.inf )\
    / np.linalg.norm( b, np.inf )
  error_magnification_factor = relative_forward_error\
    / relative_backward_error
  print( "n = ",f"{n:4d}" )
  print( "relative forward error                      = ",\
    f"{relative_forward_error:8.2e}" )
  print( "relative backward error                     = ",\
    f"{relative_backward_error:8.2e}" )
  print( "error magnification factor                  = ",\
   f"{error_magnification_factor:8.2e}" )
  print( "kappa( A ) = || A ||_oo * || A^{-1} ||_oo    = ",\
   f"{np.linalg.norm(A,np.infty)\
      * np.linalg.norm(np.linalg.inv(A),np.infty):8.2e}")
  print("\n")
```

```
n =     6
relative forward error                       =  6.48e-10
relative backward error                      =  1.45e-16
error magnification factor                   =  4.47e+06
kappa( A ) = || A ||_oo * || A^{-1} ||_oo    =  7.03e+07


n =    10
relative forward error                       =  2.51e-04
relative backward error                      =  1.21e-16
error magnification factor                   =  2.07e+12
kappa( A ) = || A ||_oo * || A^{-1} ||_oo    =  1.31e+14
```

```
for n in [ 6, 10 ]:
  A = np.zeros( [ n, n ], dtype=float )
  for i in range(0,n):
    for j in range(0,n):
      A[i,j] = 5 / ( i + 2*j + 2. )
  x = np.ones( [ n, 1 ], dtype=float )
  b = A @ x
  L, U = lu_no_pivoting( A )
  y = forward__substitution( L, b )
  xc = backward_substitution( U, y )

  relative_forward_error = np.linalg.norm( x - xc, np.inf )\
     / np.linalg.norm( x, np.inf )
  relative_backward_error = np.linalg.norm( b - A@xc, np.inf )\
    / np.linalg.norm( b, np.inf )
  error_magnification_factor = relative_forward_error\
    / relative_backward_error
  print( "n = ",f"{n:4d}" )
  print( "relative forward error                      = ",\
    f"{relative_forward_error:8.2e}" )
  print( "relative backward error                     = ",\
    f"{relative_backward_error:8.2e}" )
  print( "error magnification factor                  = ",\
   f"{error_magnification_factor:8.2e}" )
  print( "kappa( A ) = || A ||_oo * || A^{-1} ||_oo    = ",\
   f"{np.linalg.norm(A,np.infty)\
    * np.linalg.norm(np.linalg.inv(A),np.infty):8.2e}")
  print("\n")
```

```
n =     6
relative forward error                      =  6.25e-10
relative backward error                     =  1.45e-16
error magnification factor                  =  4.31e+06
kappa( A ) = || A ||_oo * || A^{-1} ||_oo    =  7.03e+07


n =    10
relative forward error                      =  7.17e-04
relative backward error                     =  1.21e-16
error magnification factor                  =  5.91e+12
kappa( A ) = || A ||_oo * || A^{-1} ||_oo    =  1.31e+14
```