saxpy

March 10, 2019

1 Ejercicio clase 06 de Marzo, 2019

Equipo 9

```
In [1]: import numpy as np
```

1.1 Saxpy

```
In [2]: L = np.array([
            [3,0,0,0],
            [2,1,0,0],
            [1,5,1,0],
            [7,9,8,4]])
        B = np.array([ -9, 6, 2, 9 ])
        print(L,B)
```

```
[[3 0 0 0]
 [2 1 0 0]
 [1 5 1 0]
 [7 9 8 4]] [-9  6  2  9]
```

Resultado de Saxpy

```
In [3]: def resolver_saxpy(L, B, bs=1):
            N = B.shape[0]
            if N != L.shape[1]:
                raise Exception("Error: no es una matriz cuadrada")
            if not np.allclose(L, np.tril(L)):
                raise Exception("Error: no es una matriz triangular inferior")
            if (N%bs) != 0:
                raise Exception("Error: bloque tamaño %d no es factorizable entre el tamaño de l
            if bs == 1 and len(B.shape) > 1:
                raise Exception("Error: bloque tamaño %d mayor a resolución univariada"%(len(B.s
            elif bs > 1 and bs > B.shape[1]:
                raise Exception("Error: bloque tamaño %d mayor al tamaño de B %d"%(len(B.shape),
```

```
            X = np.zeros((N,bs))
            B_ = B.copy()
            L_ = L.copy()
            N = N
            if bs < 2: #detectar si es directa (sin bloques)
                for i in range(0,N):
                    X[i] = (B_[i]/L[i,i])
                    for j in range((i+1), N):
                        B_[j] = B_[j] - L[j,i]*X[i]
            if bs >= 2: #detectar si es por bloques
                for i in range(0, N, bs):
                    X[i:i+bs, :] = np.linalg.solve(L_[i:i+bs, i:i+bs], B_[i:i+bs, :])
                    for j in range(i+bs, N, bs):
                        B_[j:j+bs, :] = B_[j:j+bs, :] - L_[j:j+bs, i:i+bs]@X[i:i+bs]
            return X
        resolver_saxpy(L, B)
```

```
Out[3]: array([[ -3. ],
               [ 12. ],
               [-55. ],
               [ 90.5]])
```

Resolución por Numpy

```
In [4]: np.linalg.solve(L, B)
```

```
Out[4]: array([ -3. ,  12. , -55. ,  90.5])
```

1.1.1 Ejercicio por Bloques

```
In [5]: L = np.array([
            [3,0,0,0],
            [2,1,0,0],
            [1,5,1,0],
            [7,9,8,4]
        ])
        B = np.array([
            [-9,12],
            [6,-1],
            [2,0],
            [5,1]
        ])
        L, B
```

```
Out[5]: (array([[3, 0, 0, 0],
                [2, 1, 0, 0],
                [1, 5, 1, 0],
                [7, 9, 8, 4]]), array([[-9, 12],
                [ 6, -1],
```

```
               [ 2,  0],
               [ 5,  1]]))
```

Resultado de solver de Numpy

```
In [6]: np.linalg.solve(L,B)

Out[6]: array([[ -3. ,   4. ],
               [ 12. ,  -9. ],
               [-55. ,  41. ],
               [ 89.5, -68.5]])
```

Resolución por Saxpy Nota: el algoritmo está en la función declarada al inicio. Utilizamos el argumento bs para marcar el tamaño de bloque

```
In [7]: resolver_saxpy(L, B, bs = 2)

Out[7]: array([[ -3. ,   4. ],
               [ 12. ,  -9. ],
               [-55. ,  41. ],
               [ 89.5, -68.5]])
```

1.2 Eliminación Gaussiana Simple

```
In [8]: L = np.array([
            [1,2,1],
            [2,2,3],
            [-1,-3,0]
        ])
        B = np.array([0,3,2])
        print("L = \n", L, "\n B =", B)

L =
 [[ 1  2  1]
 [ 2  2  3]
 [-1 -3  0]]
 B = [0 3 2]
```

Resolución por Numpy

```
In [9]: np.linalg.solve(L, B)

Out[9]: array([ 1., -1.,  1.])
```

Resolución por Algoritmo Nota: No pudimos usar una multiplicación directa, por lo cual consideramos que la lista comprehensiva al ser ejecutada en paralelo es más ágil que un for tradicional.

Nuestra intención era ejecutar el siguiente código:

```
    L_[i, i] = L_[i, i] - l_k*L_[k,i] #BLAS nivel 2
```

```
In [10]: def resolver_gauss(L, B):
             N = B.shape[0]
             if N != L.shape[1]:
                 raise Exception("Error: no es una matriz cuadrada")

             X = np.zeros((N))
             B_ = B.astype(np.double)
             L_ = L.astype(np.float)
             N = N

             for k in range(0, N-1):
                 i = [m for m in range(k+1, N)]
                 l_k = L_[k, k] / L_[i,k]
                 # L_[i, i] = L_[i, i] - l_k*L_[k,i] #BLAS nivel 2
                 # no pudimos usar una multiplicación directa, por lo
                 # cual consideramos que la lista comprehensiva
                 # al ser ejecutada en paralelo es más ágil que un for tradicional.
                 L_[i,] = [L_[k, :] - l_k[j]*L_[k+j+1,:] for j in range(0, l_k.shape[0])]
         #         print("L =\n", L_)
                 B_[i] =  B_[k] - B_[i]*l_k
         #         print("B =", B_, "\n")
                 X = np.linalg.solve(L_, B_)
             return X

         resolver_gauss(L, B)

Out[10]: array([ 1., -1.,  1.])
```