W8_Keshaw_EE21B069

April 17, 2023

0.1 Keshaw Choudhary EE21B069

1 Assignment

2 Week 8

We first measure the time taken for a simple function. Then we can look at optimizing this using Cython.

2.1 Week 2 Code

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

[2]:
```
def linear_equation_solver(A, b):
    m = len(A)
    n = len(A[0])
    if m != n:
        return "Error: Matrix A must be square"
    if m != len(b):
        return "Error: Number of rows in A and b must match"

    Ab = []
    for i in range(m):
        row = []
        for j in range(n):
            row.append(A[i][j])
        row.append(b[i])
        Ab.append(row)

    for i in range(n):
        pivot = i
        for j in range(i+1, n):
            if abs(Ab[j][i]) > abs(Ab[pivot][i]):
                pivot = j
        if pivot != i:
            Ab[i], Ab[pivot] = Ab[pivot], Ab[i]
        if Ab[i][i] == 0:
            return "Error: Singular matrix - system has no unique solution"
```

```
        for j in range(i+1, n-1):
            factor = Ab[j][i] / Ab[i][i]
            for k in range(i, n+1):
                Ab[j][k] = Ab[j][k] - factor * Ab[i][k]

    x = [0] * n
    for i in range(n-1, -1, -1):
        for j in range(i+1, n):
            Ab[i][n] -= x[j] * Ab[i][j]
        x[i] = Ab[i][n] / Ab[i][i]
    return x


N=int(input("Enter a number of Row/Column"))
A = np.random.rand(N,N)
b = np.random.rand(N)
x = linear_equation_solver(A, b)
print("Solution:", x)
```

```
Enter a number of Row/Column 10
```

```
Solution: [-1.7412225904892265, 29.892377620460547, 30.286185333978224,
-30.910348548847114, -25.87982972040304, 9.899264851416484, 2.918263066899024,
-3.4440925123805233, -13.043798011516094, -2.4112455819450074]
```

This code creates the function "linear equation solver," which solves a system of linear equations represented by a square matrix "A" and a vector "b" of length equal to the number of rows in "A". The function first determines if the matrix "A" is square and whether the number of rows in "A" and "b" match, and then produces an error message if either condition is not fulfilled. The function then generates an augmented matrix "Ab" by attaching the vector "b" to the right of "A". The matrix "Ab" is then reduced to row echelon form using Gaussian elimination. If throughout the elimination process, any pivot is determined to be zero, the method generates an error message indicating that the system has no unique solution. Finally, the function use back substitution to solve the system of linear equations and provides the result as a vector. The answer is validated by constructing a "NxN" matrix "A" and a vector "b" of length "N" with the "numpy" library and feeding them into the function as inputs. After then, the solution is printed.

2.2 Speed Test

[3]:
```
%timeit linear_equation_solver(A,b)
```

```
101 µs ± 2.81 µs per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

3 Cython

For the above "linear equation solver", Lets write Cython code to make more fast

[5]:
```
%load_ext Cython
```

```
The Cython extension is already loaded. To reload it, use:
  %reload_ext Cython
```

[10]:
```
%%cython --annotate
import cython
import numpy as np
cimport numpy as np


cpdef clinear_equation_solver(np.ndarray[double, ndim=2] A, np.ndarray[double,␣
 ↪ndim=1] b):
    cdef int m = len(A)
    cdef int n = len(A[0])
    cdef int i, j, k, pivot
    cdef double factor
    cdef np.ndarray[double, ndim=2] Ab


    if m != n:
        return "Error: Matrix A must be square"
    if m != len(b):
        return "Error: Number of rows in A and b must match"

    Ab = np.empty((m, n+1), dtype=np.float64)

    for i in range(m):
        for j in range(n):
            Ab[i][j] = A[i][j]
        Ab[i][n] = b[i]

    for i in range(n):
        pivot = i
        for j in range(i+1, n):
            if abs(Ab[j][i]) > abs(Ab[pivot][i]):
                pivot = j
        if pivot != i:
            Ab[i], Ab[pivot] = Ab[pivot], Ab[i]
        if Ab[i][i] == 0:
            return "Error: Singular matrix - system has no unique solution"
        for j in range(i+1, n-1):
            factor = Ab[j][i] / Ab[i][i]
            for k in range(i, n+1):
                Ab[j][k] = Ab[j][k] - factor * Ab[i][k]


    x = np.empty(n, dtype=np.float64)
    for i in range(n-1, -1, -1):
```

```
        for j in range(i+1, n):
            Ab[i][n] -= x[j] * Ab[i][j]
        x[i] = Ab[i][n] / Ab[i][i]
    return x

N = int(input("Enter a number of Row/Column: "))
A = np.random.rand(N, N)
b = np.random.rand(N)
x = clinear_equation_solver(A, b)

print("Solution:", x)
```

```
In file included from /usr/local/lib/python3.9/dist-
packages/numpy/core/include/numpy/ndarraytypes.h:1940,
                 from /usr/local/lib/python3.9/dist-
packages/numpy/core/include/numpy/ndarrayobject.h:12,
                 from /usr/local/lib/python3.9/dist-
packages/numpy/core/include/numpy/arrayobject.h:5,
                 from /home/btech/ee21b069/.cache/ipython/cython/_cython_magic_e
4ac8087a9b6cc7222c4acdc9248e999.c:769:
/usr/local/lib/python3.9/dist-
packages/numpy/core/include/numpy/npy_1_7_deprecated_api.h:17:2: warning:
#warning "Using deprecated NumPy API, disable it with " "#define
NPY_NO_DEPRECATED_API NPY_1_7_API_VERSION" [-Wcpp]
   17 | #warning "Using deprecated NumPy API, disable it with " \
      |  ^~~~~~~

Enter a number of Row/Column:  10

Solution: [-0.1569148   0.56291959 -0.5265582   0.67266973 -0.14975739
0.25450039
  0.          0.          0.          0.04258828]
```

[10]: <IPython.core.display.HTML object>

The first section of the code defines the function clinear_equation_solver, which requires two arguments: a 1-dimensional NumPy array b and a 1-dimensional NumPy array A, which together represent the constants on the right-hand side of the equations and the matrix of coefficients, respectively. As a 1-dimensional NumPy array, the method returns the answer x.

The function first tests that the matrix A is square and that the number of rows in A corresponds to the length of b. The function gives an error message if one of these requirements is not satisfied.

The function then connects the matrix A and the array b along the last dimension to create a new 2-dimensional NumPy array called Ab. As a result, a matrix with the form [A | b] is enhanced.

After that, the function applies Gaussian elimination on the enhanced matrix Ab. The function chooses a pivot element and swaps rows for the pivot to be in the right place throughout each iteration of the outer loop. A message is returned by the function if the pivot is zero. The function subtracts multiples of the pivot row from the rows below it in the inner loop to produce zeros below the pivot.

After the completion of the Gaussian elimination, the function back-substitutes to find the answer x. By subtracting the previously obtained solutions from the corresponding constant and dividing by the diagonal element of the augmented matrix, the function, starting from the last row, finds a solution for each element of x.

The second section of the code asks the user to provide N rows and columns, initialises A random matrix and b random vector, then invokes clinear_equation_solver function to solve the equation system. The answer X is subsequently printed to the console.

3.1 Speed Test

[7]:
```
%timeit clinear_equation_solver(A,b)
```

```
10.8 µs ± 275 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
```

[ ]:
```

```