E.2　ライブラリ使用の有無における計算速度の比較

ここでは，Python 用の代表的な数値計算ライブラリである **NumPy** を使用することで大きな計算速度が得られる例を示す．次に示すプログラム matmult01.py は 800×800 の行列同士の積

$$\begin{pmatrix} c_{0,0} & c_{0,1} & \cdots & c_{0,799} \\ c_{1,0} & c_{1,1} & \cdots & c_{1,799} \\ \vdots & \vdots & \ddots & \vdots \\ c_{799,0} & c_{799,1} & \cdots & c_{799,799} \end{pmatrix} = \begin{pmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,799} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,799} \\ \vdots & \vdots & \ddots & \vdots \\ a_{799,0} & a_{799,1} & \cdots & a_{799,799} \end{pmatrix} \begin{pmatrix} b_{0,0} & b_{0,1} & \cdots & b_{0,799} \\ b_{1,0} & b_{1,1} & \cdots & b_{1,799} \\ \vdots & \vdots & \ddots & \vdots \\ b_{799,0} & b_{799,1} & \cdots & b_{799,799} \end{pmatrix}$$

を求めるものである．

プログラム：matmult01.py

```
1  # coding: utf-8
2  import time
3
4  W = 800 # 配列のサイズ
5
6  M1 = [];    M2 = [];    M3 = []
7  # サンプル行列の作成
8  for i in range(W):
9      L1 = [];    L2 = [];    L3 = []
10     for j in range(W):
11         L1.append(float(i+j))
12         L2.append(float(i-j))
13         L3.append(float(0))
14     M1.append(L1);  M2.append(L2);  M3.append(L3)
15
16 # 行列の積の計算
17 t1 = time.time()
18 print('start')
19 for i in range(W):
20     for j in range(W):
21         for k in range(W):
22             M3[i][j] += M1[i][k] * M2[k][j]
23 t2 = time.time()
24 print('stop')
25 print('time(sec):',t2-t1)
26
27 # 先頭部分の表示
28 for i in range(5):
29     for j in range(5):
30         print(f'{M3[i][j]:.1f}, ',end='')
31     print('')
```

このプログラムではリスト M1，M2，M3 で行列を表現している．はじめに M1，M2 に値を設定（6〜14 行目）し，それらの積を M3 に得る（19〜22 行目）．

このプログラムを実行した様子を次に示す．

```
start          ←行列の積の計算開始
stop            ←計算終了
time(sec):  168.72231149673462     ←計算にかかった時間
170346800.0, 170027200.0, 169707600.0, 169388000.0, 169068400.0,   ←計算結果の
170666400.0, 170346000.0, 170025600.0, 169705200.0, 169384800.0,   ←最初の 5 × 5 の
170986000.0, 170664800.0, 170343600.0, 170022400.0, 169701200.0,   ←部分
171305600.0, 170983600.0, 170661600.0, 170339600.0, 170017600.0,
171625200.0, 171302400.0, 170979600.0, 170656800.0, 170334000.0,
```

このプログラムを CPU Intel Core i7-6770HQ 2.6GHz, RAM 16GB, Windows 10 Pro, Python 3.7.3 の環境で 3 回実行して得られた計算時間の平均値は 166.1089682 秒であった．

次に，同様の計算を行うプログラムを NumPy を用いて実装した例が次に示す matmult01_np.py である．

プログラム：matmult01_np.py

```
1  # coding: utf-8
2  import time
3  import numpy as np
4
5  W = 800 # 配列のサイズ
6
7  M1 = [];    M2 = [];    M3 = []
8  # サンプル行列の作成
9  for i in range(W):
10     L1 = [];    L2 = [];    L3 = []
11     for j in range(W):
12         L1.append(float(i+j))
13         L2.append(float(i-j))
14         L3.append(float(0))
15     M1.append(L1);  M2.append(L2);  M3.append(L3)
16 N1 = np.array(M1);  N2 = np.array(M2)
17
18 # 行列の積の計算
19 t1 = time.time()
20 print('start')
21 N3 = np.dot(N1,N2)
22 t2 = time.time()
23 print('stop')
24 print('time(sec):',t2-t1)
25
26 # 先頭部分の表示
27 for i in range(5):
28     for j in range(5):
29         print(f'{N3[i][j]:.1f}, ',end='')
30     print('')
```

このプログラムの前半部（M1，M2 の作成）は先の matmult01.py と同じであるが，それらを NumPy 独自の配列オブジェクト N1，N2 に変換し，積を N3 に得ている．また，行列の積を求める部分には NumPy の dot 関数を用いている（21 行目）．
（NumPy に関する詳細は公式インターネットサイトをはじめとする他の資料[105] を参照のこと）

このプログラムを実行した様子を次に示す．

```
start          ←行列の積の計算開始
stop           ←計算終了
time(sec):  0.01336050033569336     ←計算にかかった時間
170346800.0, 170027200.0, 169707600.0, 169388000.0, 169068400.0,   ←計算結果の
170666400.0, 170346000.0, 170025600.0, 169705200.0, 169384800.0,   ←最初の 5 × 5 の
170986000.0, 170664800.0, 170343600.0, 170022400.0, 169701200.0,   ←部分
171305600.0, 170983600.0, 170661600.0, 170339600.0, 170017600.0,
171625200.0, 171302400.0, 170979600.0, 170656800.0, 170334000.0,
```

このプログラムを先と同じ環境で 3 回実行して得られた計算時間の平均値は 0.01311938 秒であった．この値は先のプログラムと比較すると約 12661.3 倍であり，1 万 2 千倍以上の速度が得られていることになる．

参考までに，同じ処理を行うプログラムを C 言語で実装（matmult01.c）して実行した例を次に示す．

プログラム：matmult01.c

```
1  #include <stdio.h>
2  #include <time.h>
3
4  #define W 800
5
```

[105] 拙書「Python3 ライブラリブック - 各種モジュールの基本的な使用方法」でも解説しています．

```c
int main() {
    int i, j, k;
    static double   M1[W][W], M2[W][W], M3[W][W];
    clock_t t1, t2;

    /* サンプル行列の作成 */
    for( i=0; i<W; i++ ) {
        for ( j=0; j<W; j++ ) {
            M1[i][j] = (double)i + (double)j;
            M2[i][j] = (double)i - (double)j;
        }
    }

    /* 行列の積の計算 */
    printf("start\n");
    fflush(stdout);
    t1 = clock();
    for( i=0; i<W; i++ ) {
        for ( j=0; j<W; j++ ) {
            M3[i][j] = 0.0;
            for ( k=0; k<W; k++ ) {
                M3[i][j] += M1[i][k] * M2[k][j];
            }
        }
    }
    t2 = clock();
    printf("stop\n");
    fflush(stdout);

    printf("time(sec): %8.4f\n", (double)(t2-t1) / CLOCKS_PER_SEC );

    /* 先頭部分の表示 */
    for ( i=0; i<5; i++ ) {
        for ( j=0; j<5; j++ ) {
            printf("%.1f, ",M3[i][j]);
        }
        printf("\n");
    }

    return(0);
}
```

このプログラムを先と同じ環境の MinGW64 下でコンパイル（gcc 8.3.0，オプション -O3）して実行した様子を次に示す．

```
start          ←行列の積の計算開始
stop           ←計算終了
time(sec):    0.4400     ←計算にかかった時間
170346800.0, 170027200.0, 169707600.0, 169388000.0, 169068400.0,   ←計算結果の
170666400.0, 170346000.0, 170025600.0, 169705200.0, 169384800.0,   ←最初の 5 × 5 の
170986000.0, 170664800.0, 170343600.0, 170022400.0, 169701200.0,   ←部分
171305600.0, 170983600.0, 170661600.0, 170339600.0, 170017600.0,
171625200.0, 171302400.0, 170979600.0, 170656800.0, 170334000.0,
```

3 回実行して得られた計算時間の平均値は 0.462 秒であった．これは最初のプログラム matmult01.py と比較すると約 359 倍の実行速度である．ここに挙げた 3 つのプログラムの実行時間などを表 34 にまとめる．

表 34: 実験結果の比較

プログラム	特徴	実行時間（秒）	比率（倍）
matmult01.py	行列をリストで表現	166.1089682	（基準）1
matmult01_np.py	NumPy を用いて計算	0.01311938	12661.3
matmult01.c	C 言語による実装	0.462	359.3