計数工学プログラミング演習 最終レポート

所属: 数理情報工学コース 3 年
学生証番号: XX-XXXXXX 氏名: 佐藤 瞭
提出日: 2019/06/09

1. 課題内容

行列の 2 乗を C 言語で計算し、データ構造やアルゴリズムを変えながら実行速度を計測する。入力と出力の書式は疎行列表現 (値が 0 でない要素のみ保持) を用いる。

入力:

- 標準入力から与えられる。
- 1 行目: 行列の行数と列数。
- 2 行目以降: 各行の要素。(列番号) (値) (列番号) (値) … と続き、(列番号)の位置に-1 がある時点でその行は終わり。
- 正方行列が与えられる。すなわち、1 行目で与えられる行列の行数と列数は同じ値になる。

出力:

- 標準出力に表示する。
- 入力された行列を (行列-行列積で) 2 乗したものを、入力と同じ書式で出力する。

以下に疎行列表現の例を記す。

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 3 \\ 1 & 0 & 2.3 & 0 \end{bmatrix}$$

の疎行列表現は、

```
4 4
2 1.0 -1
-1
1 1.0 4 3.0 -1
1 1.0 3 2.3 -1
```

2. 手法

データ構造とアルゴリズムは以下の 3 通りの組み合わせで実験した。
①疎行列、行ベクトルと列ベクトルの内積の全ての項を計算 (スクリプト: sparse_element.c)
②疎行列、行ベクトルと列ベクトルの内積で、0 でない項のみ計算 (スクリプト: sparse_union.c)
③密行列、行ベクトルと列ベクトルの内積の全ての項を計算 (スクリプト: dense_element.c)

ここで、「疎行列」のデータ構造は、「課題内容」の章で説明したような、値が 0 でない要素のみ保持するようなデータ構造のことを指し、「密行列」のデータ構造は、配列の配列を用いて、全ての要素の値を保持するようなデータ構造のことを指す。

②の「内積で 0 でない項のみ計算」のアルゴリズムを説明する。ベクトルの内積は、(n×l) 行列と(i×m) の場合の定義式 $c_{ij}=\sum_{k=1}^{l} a_{ik} b_{kj}$ $(i=1,2,\ldots,n,\ j=1,2,\ldots,m)$ からわかるように、要素同士の積を足したものである。ここで、a_{ik} または b_{kj} の値が 0 の場合は、$a_{ik} b_{kj}$ の項の値が 0 になる。そのため、a_{ik} も b_{kj} も 0 でない部分だけ要素同士の積を計算して、a_{ik} または b_{kj} の値が 0 の場合は何もせずに次のインデックスに移る、という処理にすれば効率よく計算することができる。a_{ik} も b_{kj} も 0 でない部分をみつける処理は、入力された行列の行ベクトルと、入力された行列を転置したものの行ベクトルをそれぞれリストで表現して、2 つのリストに共通して現れる列番号をとりだすことで実現した。

入力は Matrix Market (https://math.nist.gov/MatrixMarket/matrices.html) のデータを用いた。
実行時間は、bash の time コマンドで測定し、real の値を用いた。
以下にマシンのスペックを示す。学科で貸与されたラップトップの Ubnutu の環境で行った。
OS: Ubuntu 18.04.2 LTS 64bit, Memory: 15.6 GiB, Processor: Intel Core i7-6820HQ CPU @ 2.70GHz × 8

3. 実験結果

手法ごとの実行時間を表 1 に示す。

表 1 手法ごとの行列の 2 乗の計算の実行時間

データの番号	手法①での実行時間	手法②での実行時間	手法③での実行時間
bcspwr01	0m0.002s	0m0.001s	0m0.001s
bcspwr02	0m0.003s	0m0.001s	0m0.002s
bcspwr03	0m0.026s	0m0.003s	0m0.005s
bcspwr04	0m0.327s	0m0.009s	0m0.042s
bcspwr05	0m1.143s	0m0.010s	0m0.136s
bcspwr06	1m1.811s	0m0.042s	0m7.941s
bcspwr07	1m25.543s	0m0.064s	0m17.853s
bcspwr08	1m29.979s	0m0.065s	0m20.215s
bcspwr09	1m15.619s	0m0.094s	0m22.732s
bcspwr10	35m56.509s	0m1.168s	21m0.924s

4. 考察

表 1 をみると、今回のデータセットには手法②が非常に有効であることがわかる。実際、行列の非零要素数はデータの番号順に 131, 167, 476, 1612, 1623, 5300, 5824, 6050, 6511, 21842 であり、行列の全ての要素数の 1/10 から 1/1000 程度の数である。「ベクトルの内積をとるときに、 $a_{ik}b_{kj}$ の項の値が 0 にならない部分だけ計算すればよい」という方針により、大量の $0 \cdot b_{kj}$ または $a_{ik} \cdot 0$ の形の項の計算を省略でき、計算時間の短縮に繋がったと考えられる。

次に、手法ごとの計算量を見積もることを考える。行列のサイズを n とする。

まず、手法①および手法③では、 $c_{ij}=\sum_{k=1}^{n} a_{ik}b_{kj}$ $(i=1,2,\ldots,n,\ j=1,2,\ldots,n)$ の定義式に従ってすべての $a_{ik}b_{kj}$ の項を計算しているため、for 文が i, j, k の 3 重ループになり、計算量は $O(n^3)$ となることが予想される。横軸をサイズの 3 乗とした、それぞれの手法での計算時間のグラフを図 1、2 に示す。

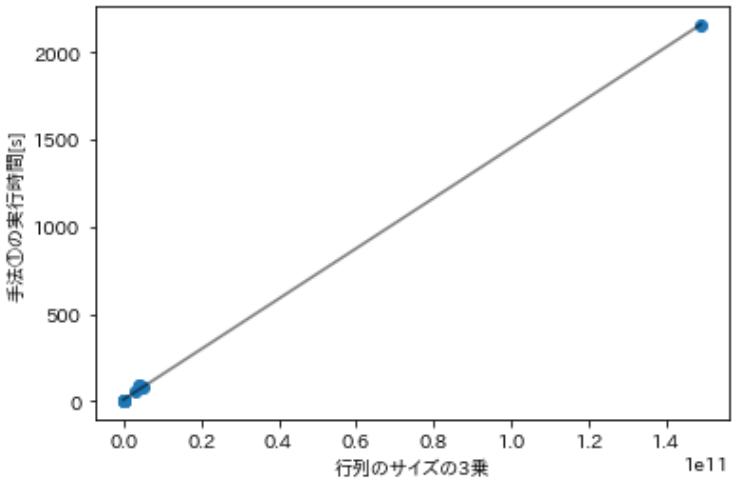

図 1 行列のサイズの 3 乗と手法①での計算時間

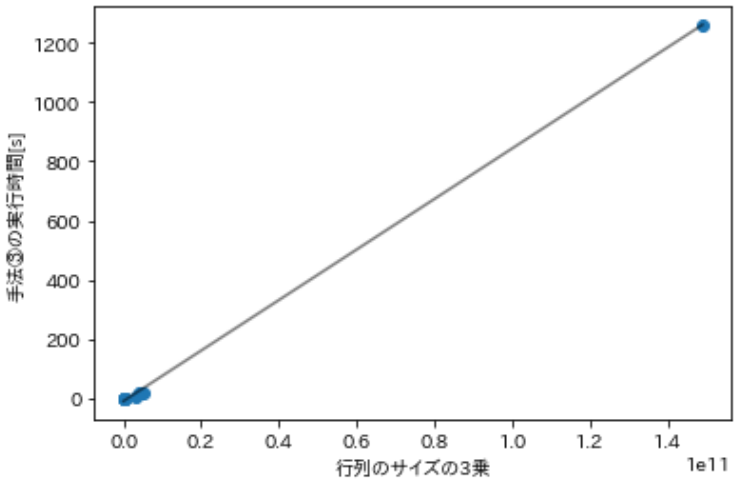

図 2 行列のサイズの 3 乗と手法③での計算時間

計算量が本当に $O(n^3)$ になっているならグラフから直線関係が読み取れるはずである。関係を見やすくするために、サイズが最大のデータを取り除いたグラフを図 3、4 に示す。回帰直線のパラメータは取り除く前と同じ値にしてある。

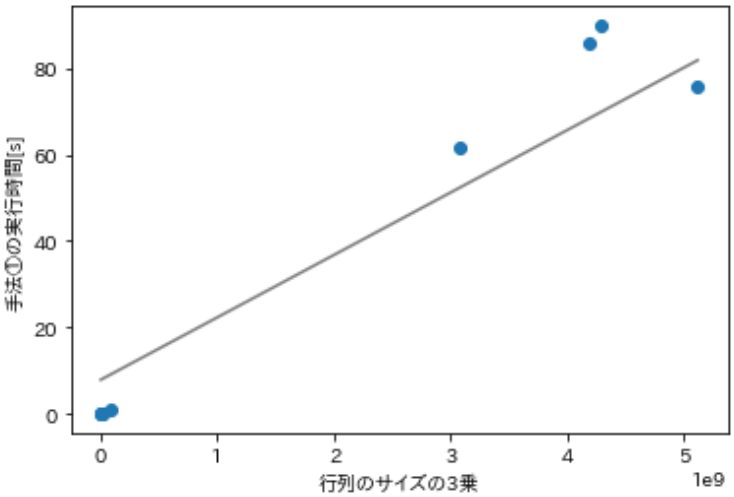

図 3 行列のサイズの 3 乗と手法①での計算時間
（サイズが最大のデータ以外）

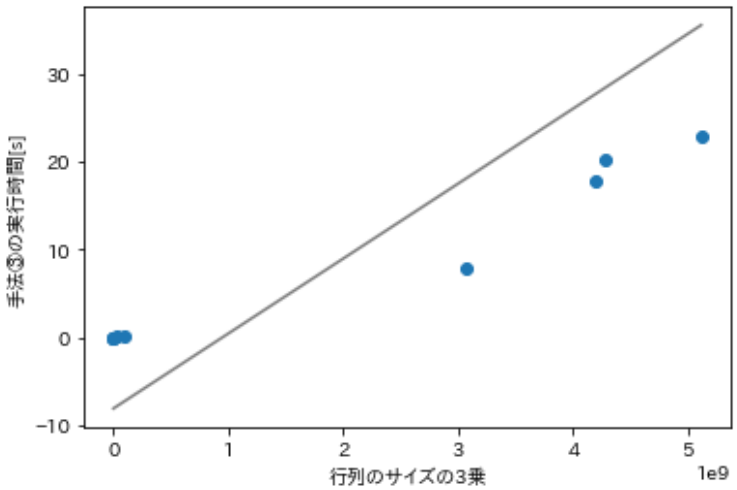

図 4 行列のサイズの 3 乗と手法③での計算時間
（サイズが最大のデータ以外）

図 3 はおおよそ直線的な関係にあるように見えるが、図 4 は、直線がサイズが最大のデータに向かって伸びており、計算量のオーダーは $O(n^3)$ よりも大きいことが伺える。行列の作成及び読み込み、積の計算、出力の中で最も計算量が大きいのは積の計算と考えられるので（他は $O(n^2)$ でできると思われるため）、要素のアクセスなどに、行列のサイズに依存する時間がかかっていると思われる。しかし、そのように考えると、手法①でも要素のアクセスにかかる時間は行の非零要素数に依存しており、これは行列のサイズとともに大きくなる関係にあることが予想されるため、手法①も $O(n^3)$ 以上の計算量であることが予想される。図 1～図 4 のデータが直線関係にあるかどうかをより確信をもって確かめるには、今回のデータセットに含まれなかった、サイズが 3000 前後の行列のデータを追加することで、グラフの回帰直線の妥当性を評価しやすくなるだろう。

つぎに、手法②の計算量を考える。行列の非零要素数を m とする。スクリプト sparse_union.c の、smatrix_product をみると、n に関する for 文の 2 重ループの中に、行ベクトルと列ベクトルの要素がどちらも 0 でない部分だけ内積をとるループ（slist_join_product）が入っている。slist_join_product のループ回数はおおよそ行あたりの非零要素数であると考え、計算量を $O(n^2 \cdot \frac{m}{n}) = O(nm)$ と予想してグラフを描画した。先程と同様に、手法②での計算時間のグラフを示す。すべてのデータのグラフを図 5 に、行列のサイズが最大のデータのみ除いたグラフを図 6 に示す。回帰直線のパラメータは、図 5 で計算したものを図 6 でも用いている。

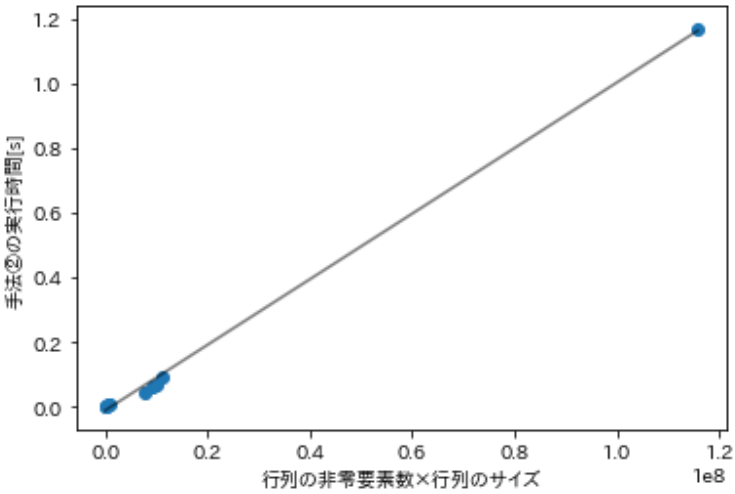

図 5 行列の非零要素数×行列のサイズと手法②での計算時間

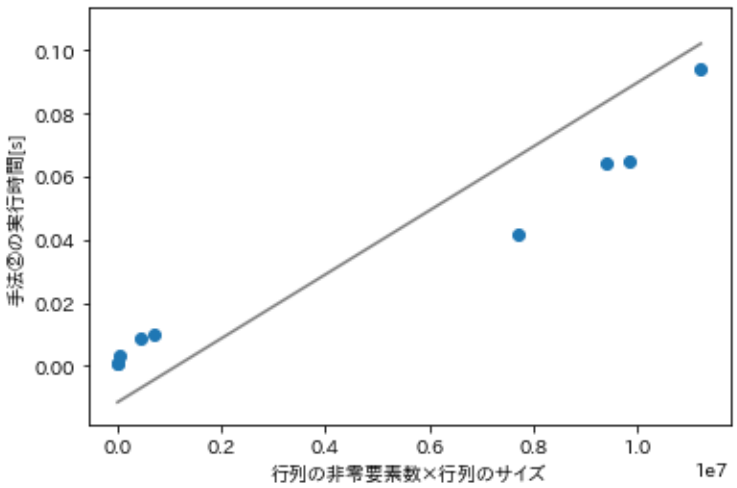

図 6 行列の非零要素数×行列のサイズと手法②での計算時間
（サイズが最大のデータ以外）

図 6 をみると、データは直線的な関係にあるように見える。そのため、手法②の計算量は $O(nm)$ である、という仮説を立てることができる。また、この仮説が正しいとすれば、今回の実験では非零要素数 m の値は、行列のサイズ n と比べて非常に小さかったため、手法②がほかの手法 2 つを比べて非常に高速だったことを、計算量のオーダーから説明することができる。また、計算量が m に比例するため、より疎な行列ほど、手法②は他の手法と比べて高速であるという説明にもつなげることができる。

今後の実験案としては、サイズが 3000 前後の行列の計算時間を計測し、手法②の計算量は $O(nm)$ であるという仮説の検証や、手法③の計算量が $O(n^3)$ なのかどうかの確認を行う、といったことが考えられる。