C 言語による行列式アルゴリズム

Takahiro HASEGAWA

2019 年 5 月 30 日

1 研究内容

疎行列の 2 乗を計算し、実行時間を測定する。
行列のデータ構造や乗算アルゴリズムを複数用いて、それらの違いを調べる。

2 手法

2.1 データ構造

データ構造として、2 次元配列と、列数と値を要素とする各行のリストの 2 つを扱う。以下に例を与える。

$$A = \begin{pmatrix} 1.0 & 2.0 & 0 & 0 & 0 \\ 2.0 & 1.0 & 2.0 & 0 & 0 \\ 0 & 2.0 & 1.0 & 0 & 0 \\ 0 & 0 & 0 & 1.0 & 0 \\ 0 & 0 & 0 & 0 & 1.0 \end{pmatrix} \tag{1}$$

2.1.1 密行列形式

行列 A のデータ構造として、2 次元配列を与えると、以下のデータ構造となる。

[[1.0, 2.0, 0, 0, 0], [2.0, 1.0, 2.0, 0, 0], [0, 2.0, 1.0, 0, 0], [0, 0, 0, 1.0, 0], [0, 0, 0, 0, 1.0]]

2.1.2 疎行列形式

行列 A のデータ構造として、列数と値を要素とする各行のリストを与えると、以下のデータ構造となる。(各行のリストの末尾に-1 を置き、その行が終了したことを明示する。)

A の行数 = m
A の列数 = n
[[1, 1.0, 2, 2.0, -1], [1, 2.0, 2, 1.0, 3, 2.0, -1], [2, 2.0, 3, 1.0, -1], [4, 1.0, -1], [5, 1.0, -1]]

2.2 アルゴリズム

2.2.1 2 次元配列・3 重ループ

1. $n \times n$ 行列 A に対し、新たな $n \times n$ 行列 B を作成する。(値は NULL である。)
2. $B_{i,j} = \sum_{k=1}^{n} A_{i,k} A_{k,j}$

ソースコード 1: 行列積を求める 3 重ループ

```
for (i=0 ; i<An ; i++){
    for (j=0 ; j<Bm ; j++){
        for (k=0; k<Am; k++){
            C[i][j] += A[i][k] * B[k][j];
        }
    }
}
```

2.2.2 2 次元配列・3 重ループ (転置行列作成)

2.2.1 の 2 の行列積計算では、最も内側のループ k が列方向に参照している。行列の 2 次元配列は第 i 行目の要素の配列の配列として記憶されているため、列方向に参照していくとキャッシュから参照できる要素が少なくなり、主記憶へのアクセス回数が増加するため、計算時間が増加する。そこで、転置行列を予め作成し、列方向へのアクセスを短時間に連続して行わないように変更する。

1. $n \times n$ 行列 A の転置行列 A^{T} を 2 次元配列のデータ構造で作成する。
2. 新たな $n \times n$ 行列 B を作成する。(値は NULL である。)
3. $B_{i,j} = \sum_{k=1}^{n} A_{i,k} A^{\mathrm{T}}_{j,k}$

2.2.3 2 次元配列・3 重ループ (jk 交換)

2.2.2 同様に、列方向への参照頻度を下げるため、2.2.1 の 2 におけるループの順番を変更する。具体的には、列方向に参照するループ文字 i,k を外側にしたループを作る。k は行方向にも参照するため、i よりも内側に置き、j, k, i の順にループさせる。

1. $n \times n$ 行列 A の転置行列 A^{T} を 2 次元配列のデータ構造で作成する。
2. 新たな $n \times n$ 行列 B を作成する。(値は NULL である。)
3. $B_{i,j} = \sum_{k=1}^{n} A_{i,k} A_{k,j}$

ソースコード 2: jk 交換 3 重ループ

```
for (i=0 ; i<An ; i++){
    for (k=0; k<Am; j++){
        for (j=0 ; j<Bm ; k++){
            C[i][j] += A[i][k] * B[k][j];
        }
    }
}
```

2.2.4 列数と値を要素とする各行のリスト

1. $n \times n$ 行列 A の転置行列 A^{T} を列数と値を要素とする各行のリストのデータ構造で作成する。
2. n 行をもつ新たな行列 B を作成する。(値は NULL である。)
3. $A[i]$ と共通する列番号を持つ $A^{\mathrm{T}}[j]$ の要素のみに対して内積をとり、k とする。
4. $B[i]$ の末尾に要素 j, k を追加する。
5. 各 $B[i]$ の末尾に要素 -1 を追加する。

3　実験結果

3.1　計算時間グラフ

実験結果を両対数グラフで示す。

赤点で計算結果をプロットし、青実線で最小二乗法による回帰直線を示す。凡例に回帰直線の方程式を記載している。

密行列形式では、全要素にアクセスするため、横軸は行列サイズの対数とし、疎行列形式では、非ゼロ要素にのみアクセスするため、横軸は非ゼロ要素数の対数とした。

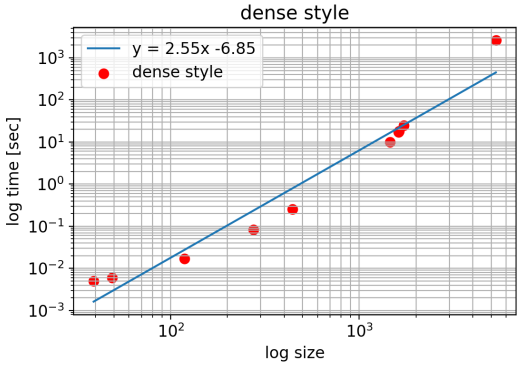

図 1: 密行列形式：3 重ループ

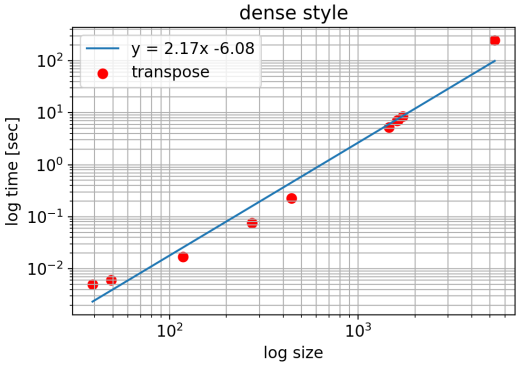

図 2: 密行列形式：転置行列作成

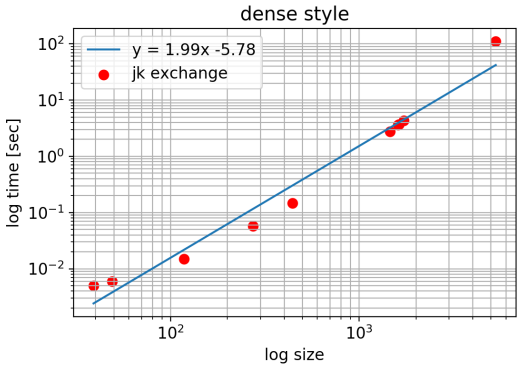

図 3: 密行列形式：jk 交換

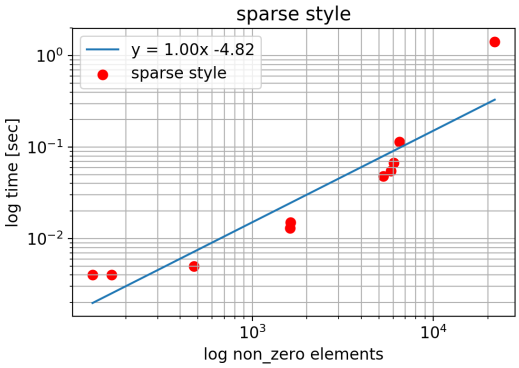

図 4: 疎行列形式

4 考察

4.1 密行列形式

- 行列積の計算に $\mathcal{O}\left(n^3\right)$ の計算量を必要とする。
  しかし、図 1 の実験結果では、単純な 3 重ループでも $n^{2.6}$ に比例する計算時間となった。これは、行列が大きくなるにつれ、キャッシュミスによる主記憶の読み出し回数が増えるためである。実際、両対数グラフにプロットした点を結ぶと下に凸となり、行列が大規模になるにつれ、計算時間が指数関数より急激に増加している。
  大規模行列での計算時間に関して、行列サイズが 100×100 以上の行列に対して、最小二乗法による単回帰を実行すると、計算時間は $n^{3.1}$ に比例する。

- 転置行列作成では、$n^{2.2}$ に比例する計算時間となった。
  これも、図 2 の両対数グラフにプロットした点を結ぶと下に凸となるが、図 1 よりも緩やかである。
  行列サイズが 100×100 以上の行列に対して、最小二乗法による単回帰を実行すると、計算時間は $n^{2.5}$ に比例する。

- jk 交換では、n^2 に比例する計算時間となった。
  これも、図 3 の両対数グラフにプロットした点を結ぶと下に凸となるが、図 1 よりも緩やかである。
  行列サイズが 100×100 以上の行列に対して、最小二乗法による単回帰を実行すると、計算時間は $n^{2.3}$ に比例する。

4.2 疎行列形式

- 行列の非ゼロ要素の個数を k とする。行列積の計算に $\mathcal{O}\left(k^2\right)$ の計算量を必要とする。
  しかし、図 4 の実験結果では、両対数グラフにプロットした点を結ぶと下に凸となり、行列が大規模になるにつれ、計算時間が指数関数より急激に増加する。
  これは、行列の大きくなるにつれ、キャッシュミスによる主記憶の読み出し回数が増えるためである。
  実際、両対数グラフの一番右上の点（非ゼロ要素数：21842）が、回帰直線から大きく外れている。