疎行列フォーマット

目次

1. 疎行列とは
2. DNS形式
3. COO形式
4. CSR形式
5. まとめ

1. 疎行列とは
2. DNS形式
3. COO形式
4. CSR形式
5. まとめ

疎行列(sparse matrix)とは

- 行列要素のうち、ほとんどが0のもの
- 「行列要素のxx%がNon zeroなら疎行列である」といった定量的な基準はない
- 用途に応じてユーザーが判断する
- ベクトルとの演算やメモリ消費の観点からいくつかの便利な格納形式が存在する。

疎行列・ベクトル積の演算

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & b & c & 0 \\ 0 & c & d & 0 \\ 0 & 0 & 0 & e \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

$$y_0 = a * x_0 + \boxed{0 * x_1} + \boxed{0 * x_2} + \boxed{0 * x_3}$$

結局0なので計算するだけムダ

$$= a * x_0$$

ゼロ要素が多い場合、必要のない計算が多数発生する

疎行列とメモリ消費

- 行列の0要素は計算に影響しないので保存しなくとも良い
- 1000×1000行列のうち、1行に5つずつしかNon-zero要素がない場合を考える
  - いつものように配列を倍精度で確保すると
    8 * 1000 * 1000 = 8,000,000 Byte = 8 MB
    のメモリが必要
  - Non zero要素だけ倍精度で確保できたとすると、単純には
    8 * 5 * 1000 = 40,000 Byte = 40 KB
    で済む(もちろん実際にはもう少しメモリ消費がある)

数値計算における疎行列の扱いのポイント

ゼロ要素を多く含む行列では、ベクトル積での演算やメモリ消費において無駄が生じる。そこで

- Non zero要素を排除したデータ形式
- 上記のデータ形式のもとでの行列ベクトル積

を考えることで計算を効率的に行う。

1. 疎行列とは
2. DNS形式
3. COO形式
4. CSR形式
5. まとめ

DNS(=Dense)形式

- 密行列(=普段扱う行列)の格納形式
- N次正方行列AのメモリをQ確保するならA[N*N]などのように確保する
- 行列ベクトル積($\boldsymbol{y} = A\boldsymbol{x}$)

```
for(int i=0; i<N; i++)
{
    y[i] = 0.0;
    for(int j=0; j<N; j++){
        y[i] += A[i*N+j] * x[j];
    }
}
```

- メモリ消費 : $8N^2$ Byte
- 計算量 : N^2

1. 疎行列とは
2. DNS形式
3. COO形式
4. CSR形式
5. まとめ

COO形式とは

- 疎行列の格納形式の1つ
- Coordinate形式の略称
- 他の形式と比較すると簡単で理解しやすいフォーマット

COO形式とは

- 1つのN次正方行列に対して以下のような3つの配列を用意する

A.row_index : nnz(=number of non zero)個の要素を持つ、行番号を示す整数型インデックス配列

A.col_index : nnz個の要素を持つ、列番号を示す整数型インデックス配列

A.val : nnz個の要素を持つ、行列要素を保持する倍精度実数配列

$$A = \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & b & c & 0 \\ 0 & c & d & 0 \\ 0 & 0 & 0 & e \end{pmatrix}$$

A.row_index : nnz(=number of non zero)個の要素を持つ、行番号を示す整数型インデックス配列

A.col_index : nnz個の要素を持つ、列番号を示す整数型インデックス配列

A.val : nnz個の要素を持つ、行列要素を保持する倍精度実数配列

A.val = [a , b , c , c , d , e]

A.row_index = [0 , 1 , 1 , 2 , 2 , 3]

A.col_index = [0 , 1 , 2 , 1 , 2 , 3]

COOでの疎行列ベクトル積

$$Ax = \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & b & c & 0 \\ 0 & c & d & 0 \\ 0 & 0 & 0 & e \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

A.val = [a , b , c , c , d , e]

A.row_index = [0 , 1 , 1 , 2 , 2 , 3]

A.col_index = [0 , 1 , 2 , 1 , 2 , 3]

```
for(int i=0; i<nnz; i++){
    y[ row_index[i] ] += val[i] *x[ col_index[i] ];
}
```

COOでのメモリ消費と疎行列ベクトル積の計算量

■ メモリ消費

A.val(double型)　　　: 8 Byte

A.row_index(int型)　: 4 Byte

A.col_index(int型)　: 4 Byte

2^31を超える行列サイズになると4 byteでは済まなくなる

→ 1つの行列要素を保持するのに消費するメモリは

DNS : 8 Byte

COO : 16 Byte (= 8 + 4 + 4)

DNSとの比較

■ nnz：N次正方行列AのNon-zero要素数

COO形式で消費するメモリ < DNS形式で消費するメモリ

➡ $$16nnz < 8N^2$$

➡ $$nnz < \frac{N^2}{2}$$

行列の半分以上をNon-zero要素が占める場合は、DNS形式を用いても構わない。

1. 疎行列とは
2. DNS形式
3. COO形式
4. CSR形式
5. まとめ

CSR形式とは

- 疎行列をメモリ上に保持するための格納形式の1種
- CRS(Compressed Row Storage)と呼ばれることもある
- CSR = Compressed Sparse Rowの略。
  (IntelやNVIDIAのライブラリではこちらで表記されており、
  普及率はこちらのほうが高い模様)
- CSCもある。(CSRの列ver.)
  以下では、CSRの名称を用いることにする。

CSR形式とは

- COOと同じく、3つの配列を用意する。

A.row_ptr : N+1個の要素を持つ配列。非ゼロ要素の開始位置を示す整数値インデックス配列。
N+1番目は非ゼロ要素数を表す。

A.col_index : nnz(=number of non zero)個の要素を持つ、列番号を示す整数型インデックス配列

A.val : nnz個の要素を持つ、行列要素を保持する倍精度実数配列

$$A = \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & b & c & 0 \\ 0 & c & d & 0 \\ 0 & 0 & 0 & e \end{pmatrix}$$

- COOと同じく、3つの配列を用意する。
  A.row_ptr : N+1個の要素を持つ配列。非ゼロ要素の開始位置を示す整数値インデックス配列。
  N+1番目は非ゼロ要素数を表す。

  A.col_index : nnz(=number of non zero)個の要素を持つ、列番号を示す整数型
  インデックス配列

  A.val : nnz個の要素を持つ、行列要素を保持する倍精度実数配列

A.val = [a , b , c , c , d , e]

A.row_ptr = [0 , 1 , 3 , 5 , $\boxed{6}$] ← 全要素数

A.col_index = [0 , 1 , 2 , 1 , 2 , 3]

CSRでの疎行列ベクトル積

$$Ax = \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & b & c & 0 \\ 0 & c & d & 0 \\ 0 & 0 & 0 & e \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

A.val　　　= [a , b , c , c , d , e]

A.row_ptr = [0 , 1 , 3 , 5 , 6]

A.col_index= [0 , 1 , 2 , 1 , 2 , 3]

Aの1行目とベクトルの積を考える。

1行目の走査　: j = A.row_ptr[2]　< A.row_ptr[3]

$bx_1 + cx_2$　　: jについてのループ内で

A.val[j] * x[col_index[j]]

を足し合わせる。

CSRでの疎行列ベクトル積

$$Ax = \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & b & c & 0 \\ 0 & c & d & 0 \\ 0 & 0 & 0 & e \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

A.val = [a , b , c , c , d , e]

A.row_ptr = [0 , 1 , 3 , 5 , 6]

A.col_index= [0 , 1 , 2 , 1 , 2 , 3]

Aのk行目とベクトルの積を考える。

1行目の走査 : j = A.row_ptr[k] < A.row_ptr[k+1]

$bx_1 + cx_2$: jについてのループ内で

A.val[j] * x[col_index[j]]

を足し合わせる。

CSRでの疎行列ベクトル積

$$Ax = \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & b & c & 0 \\ 0 & c & d & 0 \\ 0 & 0 & 0 & e \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

A.val = [a , b , c , c , d , e]

A.row_ptr = [0 , 1 , 3 , 5 , 6]

A.col_index= [0 , 1 , 2 , 1 , 2 , 3]

```
for(int i=0;i<N;i++) //行の走査
{
   y[i] = 0.;
   for(int j = row_ptr[i];j < row_ptr[i+1]; j++) //列の走査
   {
      y[i] += val[j] * x[col_index[j]];
   }
}
```

DNSとの比較

- nnz：N次正方行列AのNon-zero要素数

CSR形式で消費するメモリ < DNS形式で消費するメモリ

➡ $$8nnz + 4nnz + 4(N+1) < 8N^2$$

➡ $$nnz < \frac{2N^2 - N - 1}{3}$$

行列のサイズが大きくなるほど、CSRのほうがメモリ消費の観点から有利

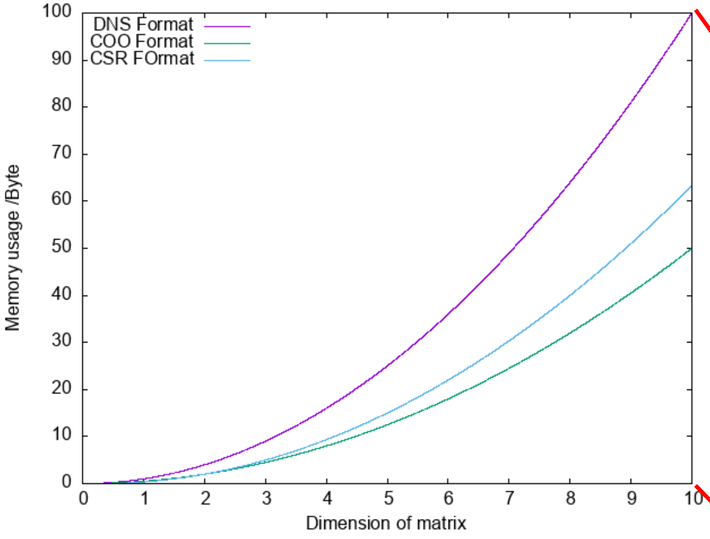
1. 疎行列とは
2. DNS形式
3. COO形式
4. CSR形式
5. まとめ

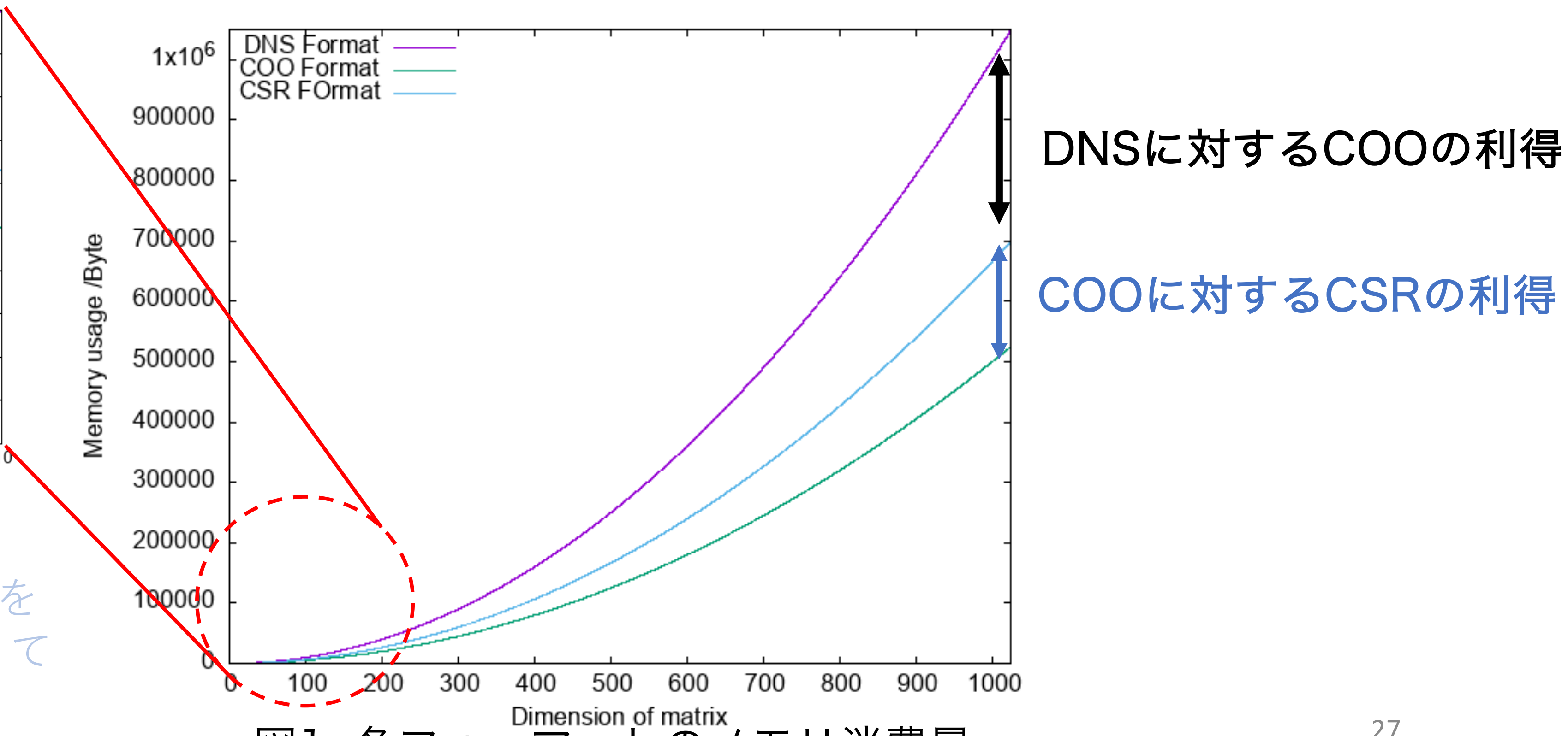
まとめ

- 行列要素のほとんどが0を占める疎行列に対して行列-ベクトル積を実行する際には(1)計算量(2)メモリ消費の観点から都合のよいフォーマットが存在する。
  - COO
  - CSR
  - ELL、JDS、BCSR、etc…
- 行列にベクトルを掛け反復計算を行うLanczos法などで有用
  ←行列要素の書き換えが生じないところがポイント

まとめ

■ 行列サイズを横軸、消費メモリ量を縦軸に取ったとき、各フォーマットごとのメモリ消費量は以下のようになる。





注)簡単のため、行列の次元を連続変数としてplotしてまっている

図1. 各フォーマットのメモリ消費量

参考文献

- [Zenn]疎行列とベクトルを掛けたい貴方に https://zenn.dev/hishinuma_t/books/sparse-matrix-and-vector-product
- [森北出版]LAPACK/BLAS入門(幸谷智紀 著)