RIKEN
Center for
Computational Science

13 Mar. 2024

RIKEN International HPC Spring School 2024 $\sim$ Toward Society 5.0

implementation of Krylov subspace method with sparse matrix

Atsushi Suzuki[1]

[1]R-CCS, Large-scale Parallel Numerical Computing Technology Research Team
atsushi.suzuki.aj@a.riken.jp

sparse matrix format : 1/3

n : # of rows

nnz : # of nonzeros

$[A]_{i\,j}$: nonzero entries at $(i,\ j)$

- COO (Coordinate) format `MUMPS`

```
structure COOformat {
  int n, nnz;
  int irow[nnz];
  int jcol[nnz];
  double coef[nnz];
};
```

- CSR (Compressed Sparse Row) /
  CRS (Compressed Row Storage) format `Pardiso`

```
structure CSRformat {
  int n, nnz;
  int ptrow[n+1];
  int indcol[nnz];
  double coef[nnz];
}
```

$[A]_{i\,j}$=`coef`[k]
  j=`indcol`[k], `ptrow`[i]$\leq k <$ `ptrow`[$i+1$]

sparse matrix format, zero-based index : 2/3

an example, 5×5 unsymmetric matrix, $n = 5$, $nnz = 15$.

1.1	1.2		1.4	
2.1	2.2	2.3		2.5
	3.2	3.3		
4.1			0.0	4.5
	5.2		5.4	5.5

	0	1	2	3	4
0	0	1		2	
1	3	4	5		6
2		7	8		
3	9			10	11
4		12		13	14

i	0			1				2		3			4			5
`ptrow[`i`]`	0			3				7		9			12			15
`indcol[`k`]`	0	1	3	0	1	2	4	1	2	0	3	4	1	3	4	
`coef[`k`]`	1.1	1.2	1.4	2.1	2.2	2.3	2.5	3.2	3.3	4.1	0.0	4.5	5.2	5.4	5.5	

- diagonal entry should exist even if the value is 0
- `indcol[]` should be in ascending order in each row

sparse matrix format, zero-based index : 3/3

5×5 symmetric matrix, upper triangular, $n = 5$, $nnz = 10$.

1.1	1.2		1.4	
	2.2	2.3		2.5
		3.3		
			0.0	4.5
				5.5

	0	1	2	3	4
0	0	1		2	
1		3	4		5
2			6		
3				7	8
4					9

i	0			1			2	3		4	5
`ptrow[`i`]`	0			3			6	7		9	10
`indcol[`k`]`	0	1	3	1	2	4	2	3	4	4	
`coef[`k`]`	1.1	1.2	1.4	2.2	2.3	2.5	3.3	0.0	4.5	5.5	

- diagonal entry should exist even if the value is 0
- `indcol[]` should be in ascending order in each row
- upper triangular matrix is accepted by `Pardiso`

SpMV : sparse matrix vector multiplication : 1/3

A : CSRformat { ptrow, indcol, coef }, $\vec{x}, \vec{y} \in \mathbb{R}^N$
to calculate $\vec{y} = A\,\vec{x}$

$$
\begin{aligned}
[A\vec{x}]_i &= \sum_j [A]_{i\,j} [\vec{x}]_j \\
&= \sum_{j \in \{j\,;\,[A]_{i\,j} \neq 0\}} [A]_{i\,j} [\vec{x}]_j \\
&= \sum_{\mathtt{ptrow}[i] \leq k < \mathtt{ptrow}[i+1]} \mathtt{coef}[k] \times [\vec{x}]_{\mathtt{indcol}[k]}
\end{aligned}
$$

```
for (i = 0; i < n; i++) {
  y[i] = 0.0;
  for (k = ptrow[i]; k < ptrow[i + 1]; k++) {
    int j = indcol[k];
    y[i] += coef[k] * x[j];
  }
}
```

SpMV : sparse matrix vector multiplication : 2/3

A : CSRformat { ptrow, indcol, coef }, upper part is stored, $\vec{x}, \vec{y} \in \mathbb{R}^N$
to calculate $\vec{y} = A\,\vec{x}$
assumption

- coefficient of diagonal value of A is stored
- the first entry of the `i`-th row `indcol[ptrow[i]] == i`

```
    for (i = 0; i < n; i++) {
      y[i] = 0.0;
    }
    for (i = 0; i < n; i++) {
      for (k = ptrow[i] + 1; k < ptrow[i + 1]; k++) {
        int j = indcol[k];
        y[i] += coef[k] * x[j];
        y[j] += coef[k] * x[i];
      }
      int k = ptrow[i];
//      int j = indcol[k] ( == i )
      y[i] += coef[k] * x[i];
    }
```

SpMV : sparse matrix vector multiplication : 3/3

for calculation of $\vec{y} = \alpha A\vec{x} + \beta\vec{y}$ with general sparse matrix A stored in `CSRformat`

```
void SparseGEMV(struct CSRformat &A,
                const double &alpha, std::vector<double> &x,
                const double &beta, std::vector<double> &y)
{
  int nrow = A.n;
  for (int i = 0; i < nrow; i++) {
    y[i] *= beta;
  }
  double tmp;
  for (int i = 0; i < nrow; i++) {
    tmp = 0.0;
    for (k = ptrow[i]; k < ptrow[i + 1]; k++) {
      int j = indcol[k];
      tmp += coef[k] * x[j];
    }
  }
  y[i] += alpha * tmp;
}
```

inner product

two vectors : $\vec{x},\ \vec{y} \in \mathbb{R}^N$

$$(\vec{x}, \vec{y}) = \sum_{1 \leq i \leq N} [\vec{x}]_i [\vec{y}]_i$$

```
double tmp = 0.0;
for (i = 0; i < n; i++) {
  tmp += x[i] * y[i];
}
```

BLAS level 1 subroutine `ddot`

```
double cblas_ddot(const int n,
                  const double *x, const int incx,
                  const double *y, const int incy);

tmp = cblas_ddot(n, &x[0], 1, &y[0], 1);
```

C++ STL `std::vector< >` class

```
#include <vector>
std::vector<double> x(100);  // allocation with size = 100
x.resize(200); // enlarging array with keeping 100 entries
x.clean();     // deallocation
&x[0];         // double pointer to the first entry
x.data();      //
```

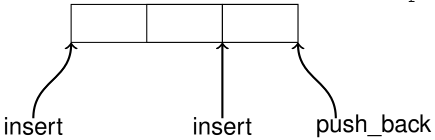
conversion of Sparse matrix format COO to CSR : 1/2

an example, 5×5 unsymmetric matrix, $n = 5$, $nnz = 15$.

1.1	1.2		1.4	
2.1	2.2	2.3		2.5
	3.2	3.3		
4.1			0.0	4.5
	5.2		5.4	5.5

	pcol				pcoef			
0	0	1	2		1.1	1.2	1.4	
1	0	1	2	4	2.1	2.2	2.3	2.5
2	1	2			3.2	3.3		
3	0	3	4		4.1	0.0	4.5	
4	1	3	4		5.2	5.4	5.5	

column index is stored in list<int> pcol[i] data i-th each row



- array of list for each row are stored in vector<list<int> >
- insertion using std::list iterator sotres data in ascending order
- one dimensional array data int indcol[k] is copied from pcol

list data structure will manage dynamic memory allocation but is not efficient for HPC application

conversion of Sparse matrix format COO to CSR : 2/2

```
std::vector<std::list<int> > pcol(nrow);
std::vector<std::list<double> > pcoef(nrow);
for (int k = 0; k < nnz; k++) {
  int i = Acoo.irow[k], j = Acoo.jcol[k];
  double coef = Acco.coef[k];
  if (pcol[i].empty())
    pcol[i].push_back(j); pcoef[i].push_back(coef);
  else {
    if (pcol[i].back() < j)
      pcol[i].push_back(j); pcoef[i].push_back(coef);
    else {
      std::list<double>::iterator iv = pcoef[i].begin();
      std::list<int>::iterator it = pcol[i].begin();
      for (; it != pcol[i].end(); ++it, ++iv) {
        if ((*it) > j)
          pcol[i].insert(it, j); pcoef[i].insert(iv, coef);
      }
    }
  }
}
Acsr.ptrow[0] = 0;
int k = 0;
for (int i = 0; i < n; i++) {
  ptrow[i + 1] = ptrow[i] + pcol[i].size();
  std::list<double>::iterator iv = pcoef[i].begin();
  std::list<int>::iterator it = pcol[i].begin();
  for (; it != pcol[i].end(); ++it, ++iv, k++) {
    Acsr.pcol[k] = (*it); Acsr.coef[k] = (*iv); } }
```

conversion of Sparse matrix format CSR to COO

```
int nrow = Acsr.n;
for (int i = 0; i < n; i++) {
  for (int k = Acsr.ptrow[i]; k < Acsr.ptrow[i + 1]; k++) {
    Acoo.irow[k] = i;
    Acoo.jcol[k] = Acsr.indcol[k];
    Acoo.coef[k] = Acsr.coef[k];
  }
}
```

exercise : implement GMRES/CG using IML++ : 1/2

GMRES is written by `C++` template in `IML++`,
`https://math.nist.gov/iml++/gmres.h.txt`
first version without preconditoner `const Preconditioner &M`
replacing Operator, Vector, and Matrix classes by simpler ones

- `Real` → `double`
- `Vector` → `std::vector<double>`
- `Operator` → `structure CSRformat`
- `Matrix` → `std::vector<double>` as one-dimensionalized array

```
std::vector<double> HH((max_iter + 1) * (max_iter + 1));
#define H(i, j) HH((i) + (j) * (max_iter + 1))
```

matrix vector product to compute the residual as

```
r = b - A * x;
```

will be replaced by

```
std::vector<double> r(b);
SparseGEMV(A, (-1.0), x, 1.0, r);
```

CG in `IML++`, `https://math.nist.gov/iml++/cg.h.txt`

- CSRformat for symmetric matirx with storing uppper part
- `cblas_daxy` for $\vec{x}$ += $\alpha \times \vec{p}$

```
double cblas_daxpy(const int n, const double alpha,
                   const double *x, const int incx,
                   double *y, const int incy);
```

exercise : implement GMRES/CG using IML++ : 2/2

```
template < class Operator, class Vector, class Preconditioner,
           class Matrix, class Real >
int
GMRES(const Operator &A, Vector &x, const Vector &b,
      const Preconditioner &M, Matrix &H, int &m, int &max_iter,
      Real &tol)
{
  Real resid;
  int i, j = 1, k;
  Vector s(m+1), cs(m+1), sn(m+1), w;

  Real normb = norm(M.solve(b));
  Vector r = M.solve(b - A * x);
```

→ without template

```
int
GMRES(csr_matrix &A, std::vector<double> &x, std::vector<double> &b,
      int &m, int &max_iter,
      double &tol)
{
  double resid;
  int nrow = x.size();
  int i, j = 1, k;
  std::vector<double> s(m+1, 0.0), cs(m+1), sn(m+1), w(nrow);
  std::vector<double> HH((max_iter + 1) * (max_iter + 1));

  double normb = norm(b);
// r = Precond(A, b, x);
```