Sparse data structures

Praveen. C
praveen@math.tifrbng.res.in

Tata Institute of Fundamental Research
Center for Applicable Mathematics
Bangalore 560065
http://math.tifrbng.res.in

March 27, 2023

Sparse matrix: Compressed Row Storage (CRS)

We assume C/C++ style indexing, starting from 0.

`nrow` $\times$ `ncol` matrix $\qquad A = \begin{bmatrix} 10 & 0 & 0 & 7 \\ 0 & 1 & 2 & 0 \\ 3 & 0 & 5 & 9 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

	0	1	2	3	-
`row_ptr`	0	2	4	7	8

	0	1	2	3	4	5	6	7
`col_ind`	0	3	1	2	0	2	3	3
`val`	10	7	1	2	3	5	9	1

Integer arrays: `row_ptr` and `col_ind`

```
length(row_ptr) = nrow + 1
```

```
length(col_ind) = length(val) = row_ptr[nrow]
```

Modified CSR format

`row_ptr[i]` is starting index of i'th row and the values

$$\texttt{val[c]}, \quad c = \texttt{row\_ptr[i]}, ..., \texttt{row\_ptr}[\texttt{i}+\texttt{1}] - \texttt{1}$$

are the non-zero values in the i'th row. Their corresponding column indices are

$$\texttt{col\_ind[c]}, \quad c = \texttt{row\_ptr[i]}, ..., \texttt{row\_ptr}[\texttt{i}+\texttt{1}] - \texttt{1}$$

In each row, the columns need not be stored in any particular order. This allows us to store the diagonal element as the first element in each row. In modified CSR format

$$\mathrm{A(i,i)} = \mathrm{val[c]}, \quad c = \texttt{row\_ptr[i]}$$

	0	1	2	3	4	5	6	7
`col_ind`	0	3	1	2	2	0	3	3
`val`	10	7	1	2	5	3	9	1

Very useful for some iterative algorithms, since it allows us to access diagonal element without searching for the column.

Example of SparseMatrix Class

```
class SparseMatrix
{
   public:
      SparseMatrix (); // constructor
      ~SparseMatrix(); // destructor
      void multiply(const Vector& x, Vector& y) const;
      double operator()(int i, int j) const;

   private:
      int nrow;
      int *row_ptr, *col_ind;
      double *val;
}
```

`class Vector` allows us to store an array of doubles. We will see more in the examples.

Multiply sparse matrix with vector

$$y = Ax$$

```
void SparseMatrix::multiply(const Vector& x,
                            Vector& y) const
{
   assert (x.size() == nrow);
   assert (x.size() == y.size());
   for(int i=0; i<nrow; ++i)
   {
      y(i) = 0;
      int row_beg = row_ptr[i];
      int row_end = row_ptr[i+1];
      for(int j=row_beg; j<row_end; ++j)
         y(i) += val[j] * x(col_ind[j]);
   }
}
```

Example usage

```
int main()
{
   SparseMatrix A;
   Vector x, y;
   // Fill the matrix A and vector x
   A.multiply(x, y); // y = A*x
}
```

Access element of SparseMatrix

```
double operator()(int i, int j) const
{
   int row_beg = row_ptr[i];
   int row_end = row_ptr[i+1];
   for(int d=row_beg; d<row_end; ++d)
      if(col_ind[d] == j)
         return val[d];
   return 0.0;
}
```

Usage:

```
   SparseMatrix A;
   // Fill the matrix A
   cout << A(2,3) << endl;
```

Destructor

```
SparseMatrix::~SparseMatrix()
{
   // WARNING: Check that these data have been allocated
   if(nrow > 0)
   {
      delete[] row_ptr;
      delete[] col_ind;
      delete[] val;
   }
}
```

Practical example

SparseMatrix Class: `sparse_matrix.h`

Usually, *header* files contain function or class *declarations*.

```
template<class T>
class SparseMatrix
{
   public:
      SparseMatrix (std::vector<unsigned int>& row_ptr,
                    std::vector<unsigned int>& col_ind,
                                       std::vector<T>& val);
      ~SparseMatrix() {};
      void multiply(const Vector<T>& x, Vector<T>& y) const;
      T operator()(unsigned int i,
                   unsigned int j) const;

   private:
      unsigned int nrow;
      std::vector<unsigned int> row_ptr, col_ind;
      std::vector<T> val;
}
```

`class vector` is part of `standard` *namespace* in C++.

Constructor: `sparse_matrix.cc`

The actual function or class *definition* is usually put in a `*.cc` file.

```
template<class T>
SparseMatrix<T>::SparseMatrix
    (std::vector<unsigned int>& row_ptr,
     std::vector<unsigned int>& col_ind,
                 std::vector<T>& val)
   :
   nrow (row_ptr.size()-1),
   row_ptr (row_ptr),
   col_ind (col_ind),
   val (val)
{
   assert (row_ptr.size() >= 2);
   assert (col_ind.size() > 0);
   assert (col_ind.size() == val.size());
}
```

`row_ptr (row_ptr)` calls the *copy constructor* of `std::vector`.

Main program

```
#include "sparse_matrix.h"
#include "Vector.h"

using namespace std;

int main ()
{
   unsigned int nrow=4, nval=8;
   vector<unsigned int> row_ptr(nrow+1), col_ind(nval);
   vector<double> val(nval);
   row_ptr[0] = 0; row_ptr[1] = 2; etc...
   SparseMatrix<double> A(row_ptr, col_ind, val);
   cout << A;
   cout << A(2,3) << endl;

   Vector<double> x(nrow), y(nrow);
   x = 1.0;
   A.multiply(x, y);
   cout << y;
   return 0;
}
```

Now we program

- `Vector.h`, `Vector.cc`
- `sparse_matrix.h`, `sparse_matrix.cc`
- `sparse_test.cc`
- `makefile`

Assignment

- What happens if a row does not have any non-zero entries ? Try with an example.
- Create sparse matrix by directly entering values: This should automatically create `row_ptr`, `col_ind` and `val` data. Assume the values are entered *row-wise only*.

```
SparseMatrix<double> A(4); // 4 x 4 sparse matrix
A.set(0,0,10);
A.set(0,3,7);
A.set(1,1,1);
etc..
A.set(3,3,1);
A.close();
```

  After this point, we cannot change the sparsity structure of the matrix.
- Read up about conjugate gradient method.
- Write a function that implements $y = A^\top x$ given A in CSR format.

COO format