STL and templates

Paolo Joseph Baioni

March 27, 2025

Exercise 1 - What is a sparse matrix?

- A $N \times N$ sparse matrix is a matrix whose number of non-zero elements N_{nz} is $O(N)$.
- The average number m of non-zero elements per row (or column) is constant w.r.t. the matrix size.
- If the majority of matrix entries is 0, *i.e.* if $m \ll N$ it is convenient to store only the non-zero elements.
- The matrix-vector product (which is the basic ingredient of Krylov solvers) is $O(N)$ rather than $O(N^2)$.

Exercise 1 - Sparse matrix formats

Some (slightly revisited) classical data structures for sparse matrices

$$A = \begin{bmatrix} 4 & -1 & 0 & 0 \\ -1 & 4 & -1 & 0 \\ 0 & -1 & 4 & -1 \\ 0 & 0 & -1 & 4 \end{bmatrix}$$

COO (coordinates) or AIJ:

```
std::vector<double> A{4, -1, -1, ...};
std::vector<int>    I{0,  0,  1, ...};
std::vector<int>    J{0,  1,  0, ...};
```

triplet vector:

```
std::vector<std::tuple<int, int, double>>
t{{0, 0, 4}, {0, 1, -1}, {1, 0, -1}, ...};
```

CSR (Compressed Sparse Row) or CRS (Compr. Row Storage) or Yale:

```
std::vector<double> val{4, -1, -1, 4, -1, -1, 4, -1, -1, 4};
std::vector<int>    col_idx{0, 1, 0, 1, 2, 1, 2, 3, 2, 3};
std::vector<int>    row_ptr{0, 2, 5, 8, 10}; // One entry per row.
```

Exercise 1 - Typical operations with sparse matrices

- Insertion:
  `A[i][j] = x;`
- Deletion:
  `A[i].erase(j);`
- Random access:
  `x = A[i][j]; A[i][j] += y;`
- Sequential traversing:

```
for (row : A)
  {
    for (column : row)
          std::cout << column.value << " ";
    std::cout << std::endl;
  }
```

- Matrix-vector multiplication:
  `std::vector<double> y = A * x;`

Exercise 1 - Inheriting from STL containers

- The C++ standard leaves a lot of freedom for implementation of containers.
- STL containers have non-virtual destructors.
- C.35: A base class destructor should be either public and virtual, or protected and non-virtual.

```
class Base {
public:
 ~Base { do_something(); }; // Non-virtual.
}

class Derived : public Base {
public:
 MyComplexType member;
 ~Derived { member.clear(); ... }
}

Base *var = new Derived;
delete var; // Calls var::~Base() but not var::~Derived()!
```

Exercise 1 - Sparse matrix class

- Implement a `C++` class to represent a sparse matrix based on STL containers.
- Simplify random access, allocation, entry increment, sequential traversing.

Exercise 1 - Sparse matrix class

Implement the following methods:

```
/// Convert row-oriented sparse matrix to AIJ format.
void
aij(std::vector<double> &a,
    std::vector<int> &    i,
    std::vector<int> &    j);

/// Convert row-oriented sparse matrix to CSR format.
void csr(std::vector<double> &a,
         std::vector<int> &    col_ind,
         std::vector<int> &    row_ptr);

/// Stream operator.
friend std::ostream &
operator<<(std::ostream &stream,
           sparse_matrix &M);

/// Sparse matrix increment.
void sparse_matrix::operator+=(sparse_matrix &other);

/// Compute matrix-vector product.
friend std::vector<double>
operator*(sparse_matrix &M,
          const std::vector<double> &x);
```

Bonus 2 - Finite differences

Implement a `C++` template class to compute derivatives of any order of a given function (callable object) using recursive backward/forward first-order finite difference formulas.