Smart Products Lab 1 & 2

Tyler Morrison

February 2, 2019

Section 1.

a) Data Structures Vectors are formed as vectors of floats using the STL container `vector<int>`. Matrices are vectors of vectors (i.e: `vector<vector<int>>`).

b) Extra Functions

- `float dotv(const VecF_t, const VecF_t)`: Dot product of two vectors.
- `MatF_t initIdent(const unsigned int)`: Initialize an identity matrix.
- `void swapRows(MatF_t&, const int, const int)`: Swap two rows of a matrix.
- `float detm(const MatF_t)`: Compute determinant of a matrix. (Gauss-based)
- `void fwdElim(MatF_t&)`: Conduct forward elimination on matrix. (Gauss-based)
- `void bckWlim(MatF_t&)`: Conduct backward elimination on matrix. (Gauss-based)
- `void rref(MatF_t&)`: Get reduced row-echelon form of matrix. (Gauss-based)
- `int rankm(const MatF_t)`: Get rank of a matrix. (Gauss-based)

c) Implementation of SPMath functions in SPmatrix The matrix class in `SPmatrix` calls functions from `SPmath` whenever it can so that functions are not redefined. For example:

```
1 MatF MatF::inv() {
2         MatF B(invertm(mat_data_));
3         return B;
4 }
```

Section 2.

a) Matrix inversion algorithm To do matrix inversion, I used the forward elimination algorithm from Wikipedia: Gaussian elimination, and completed the backward elimination myself.

b) Technique name An algorithm for Gauss-Jordan elimination with pivots selected by numerical magnitude was used because it has relatively low complexity compared to other methods ($\mathcal{O}(n^3)$) and good numerical stability for poorly conditioned matrices.

c) Invertible check method A check was added to invertm() to determine if, after forward elimination, the left hand side of the composite matrix has any zeros in the diagonal.

This is, of course, a tad inefficient, as it adds overhead to cases where the matrix is non invertible, but in those cases, performance is usually not the limiting issue. IMHO it is better to limit the peformance of the non-invertible cases with a post-hoc test than hurt the invertible cases with a costly check before calling or increase complexity by restructuring the library to include a test partway through inversion (which would be optimal.) `Commit# 499b9f5e1`

Section 3.

a) What info do you think would have helped you to better complete the lab? Nothing that I can think of right now.

b) Output from labone.cpp and labtwo.cpp Included below are the outputs of the compiled programs.

Program 1: Output of labone.cpp

```
Adding Vectors (C=A+B): C =
5
5
5
5
5
5
5
5
5
5
error!: addv(): Cannot add vectors of different dimensions!

Fat Matrix: Am =
 2.000       2.000       2.000       2.000       2.000       2.000       2.000       2.000
2.000       2.000
 3.000       3.000       3.000       3.000       3.000       3.000       3.000       3.000
3.000       3.000
 5.000       5.000       5.000       5.000       5.000       5.000       5.000       5.000
5.000       5.000

transposed: Am =
 2.000       3.000       5.000
 2.000       3.000       5.000
 2.000       3.000       5.000
 2.000       3.000       5.000
 2.000       3.000       5.000
 2.000       3.000       5.000
 2.000       3.000       5.000
 2.000       3.000       5.000
 2.000       3.000       5.000
 2.000       3.000       5.000

Skinny Matrix: Bm =
 1.000       1.000       1.000
 2.000       2.000       2.000
 3.000       3.000       3.000
 4.000       4.000       4.000
 5.000       5.000       5.000

```

```
transposed: Bm =
 1.000       2.000       3.000       4.000       5.000
 1.000       2.000       3.000       4.000       5.000
 1.000       2.000       3.000       4.000       5.000

Square Matrix: Cm =
 1.000       1.000       1.000
 2.000       2.000       2.000
 3.000       3.000       3.000

transposed: Cm =
 1.000       2.000       3.000
 1.000       2.000       3.000
 1.000       2.000       3.000

Cm resized with pointer
 1.000       2.000       3.000
 1.000       2.000       3.000
 1.000       2.000       3.000
 0.0000      0.0000      0.0000
 0.0000      0.0000      0.0000
```

Program 2: Output of labtwo.cpp

```
Matrix A =
         5        3.3       2.34
       6.7       7.32        9.3
     8.123      7.345      22.34

Matrix Ainv = inverse(A)
    0.4589    -0.2725    0.06536
   -0.3573     0.4468    -0.1486
   -0.0494   -0.04781    0.06984

Matrix B = A * Ainv
         1 -7.451e-08 7.451e-08
-8.941e-08          1  5.96e-08
         0 1.192e-07          1

Matrix C = inverse((transpose(A) * Ainv + A - B) * A)
     2.089     -1.786      0.524
    -2.216       1.94    -0.5782
 -0.01374 -0.007176   0.007603
```