// Illustration of concepts from Linear Algebra using the NEWMAT library

// Written by Prof. R.S. Sreenivas for IE523: Financial Computation

// We want to check if there is (or, if there is no) solution to Ax = y

// Notice, in my example A is singular, and therefore has no inverse.

#include "stdafx.h"

#include <iostream>

#include <iomanip>

#include <cmath>

#include "windows.h"

#include "newmatap.h"

#include "newmat.h"

#include "newmatio.h"

#define EPSILON 0.0000001

// This routine computes the rank of a matrix by counting the number of

// non-zero singular-values of the matrix.

// See http://en.wikipedia.org/wiki/Singular\_value\_decomposition for details

// regarding the singular-value-decomposition of a matrix

int Rank(Matrix A)

{

// The SVD routine in NEWMAT assume #rows >= #cols in A

// if #rows < #cols, then we compute the SVD of the transpose of A

if (A.nrows() >= A.ncols())

{

DiagonalMatrix D(A.ncols());

SVD(A, D);

int rank = 0;

for (int i = 1; i <= A.ncols(); i++)

if (abs(D(i)) > EPSILON)

rank++;

return (rank);

}

else {

DiagonalMatrix D(A.nrows());

SVD(A.t(), D);

int rank = 0;

for (int i = 1; i <= A.nrows(); i++)

if (abs(D(i)) > EPSILON)

rank++;

return (rank);

}

}

ColumnVector Find\_Solution(Matrix A, Matrix y)

{

// A should be of full rank and y is a column vector

if (Rank(A) == min(A.ncols(), A.nrows()))

{

return ((A.t()\*A).i())\*A.t()\*y;

}

else {

cout << "Matrix: " << endl;

cout << setw(9) << setprecision(3) << A;

cout << "is not of full-rank... exiting..." << endl;

exit(0);

}

}

int main(int argc, char\* argv[])

{

Matrix A(8,5);

A(1, 1) = 2; A(1, 2) = 0; A(1, 3) = 0; A(1, 4) = 6; A(1, 5) = 2;

A(2, 1) = 1; A(2, 2) = -1; A(2, 3) = -1; A(2, 4) = 4; A(2, 5) = 0;

A(3, 1) = 2; A(3, 2) = -2; A(3, 3) = 2; A(3, 4) = 4; A(3, 5) = 0;

A(4, 1) = 2; A(4, 2) = -2; A(4, 3) = 0; A(4, 4) = 6; A(4, 5) = 0;

A(5, 1) = -4; A(5, 2) = 4; A(5, 3) = -8; A(5, 4) = -4; A(5, 5) = 0;

A(6, 1) = 4; A(6, 2) = 0; A(6, 3) = -2; A(6, 4) = 14; A(6, 5) = 4;

A(7, 1) = 0; A(7, 2) = 0; A(7, 3) = 0; A(7, 4) = 7; A(7, 5) = 1;

A(8, 1) = 0; A(8, 2) = 1; A(8, 3)= 0; A(8, 4) = 0; A(8, 5) = 2;

Matrix y(8, 1);

y(1, 1) = 2; y(2, 1) = 7; y(3, 1) = 18; y(4, 1) = 16; y(5, 1) = -40; y(6, 1) = 2; y(7, 1) = 7; y(8, 1) = 7;

cout << "Finding the general solution to Ax = b in Lesson 2" << endl;

cout << "--------------------------------------------------" << endl;

cout << "Rank of A = " << Rank(A) << " (which has " << A.nrows();

cout << " rows and " << A.ncols() << " cols)" << endl;

cout << "So... we can drop two columns of A preserve rank" << endl;

// select first three cols of A

Matrix B1(6, 3), B2(6, 3), B3(6, 3), B4(6, 3), B5(6, 3), B6(6, 3), B7(6, 3), B8(6, 3), B9(6, 3), B10(6, 3);

B1 = (A.column(1) | A.column(2) | A.column(3));

B2 = (A.column(1) | A.column(2) | A.column(4));

B3 = (A.column(1) | A.column(2) | A.column(5));

B4 = (A.column(1) | A.column(3) | A.column(4));

B5 = (A.column(1) | A.column(3) | A.column(5));

B6 = (A.column(2) | A.column(3) | A.column(4));

B7 = (A.column(2) | A.column(3) | A.column(5));

B8 = (A.column(3) | A.column(4) | A.column(5));

B9 = (A.column(1) | A.column(4) | A.column(5));

B10 = (A.column(2) | A.column(4) | A.column(5));

cout << "Ranks of:" << endl;

cout << "(c1 c2 c3) = " << Rank(B1) << "; (c1 c2 c4) = " << Rank(B2);

cout << "; (c1 c2 c5) = " << Rank(B3) << "; (c1 c3 c4) = " << Rank(B4);

cout << "; (c1 c3 c5) = " << Rank(B5) << "; (c2 c3 c4) = " << Rank(B6);

cout << "; (c2 c3 c5) = " << Rank(B7) << "; (c3 c4 c5) = " << Rank(B8);

cout << "; (c1 c4 c5) = " << Rank(B9) << "; (c2 c4 c5) = " << Rank(B10) << endl;

cout << "Apply Equation 1 of Lesson 2 to get eight Basic Feasible Solutions" << endl << endl;

ColumnVector s1(3), s2(3), s5(3), s6(3), s7(3), s8(3), s9(3), s10(3);

s1 = Find\_Solution(B1, y); s2 = Find\_Solution(B2, y);

s5 = Find\_Solution(B5, y); s6 = Find\_Solution(B6, y);

s7 = Find\_Solution(B7, y); s8 = Find\_Solution(B8, y);

s9 = Find\_Solution(B9, y); s10 = Find\_Solution(B10, y);

ColumnVector soln1(5), soln2(5), soln5(5), soln6(5), soln7(5), soln8(5), soln9(5), soln10(5);

soln1(1) = s1(1); soln1(2) = s1(2); soln1(3) = s1(3); soln1(4) = 0; soln1(5) = 0;

soln2(1) = s2(1); soln2(2) = s2(2); soln2(3) = 0; soln2(4) = s2(3); soln2(5) = 0;

soln5(1) = s5(1); soln5(2) = 0; soln5(3) = s5(2); soln5(4) = 0; soln5(5) = s5(3);

soln6(1) = 0; soln6(2) = s6(1); soln6(3) = s6(2); soln6(4) = s6(3); soln6(5) = 0;

soln7(1) = 0; soln7(2) = s7(1); soln7(3) = s7(2); soln7(4) = 0; soln7(5) = s7(3);

soln8(1) = 0; soln8(2) = 0; soln8(3) = s8(1); soln8(4) = s8(2); soln8(5) = s8(3);

soln9(1) = s9(1); soln9(2) = 0; soln9(3) = 0; soln9(4) = s9(2); soln9(5) = s9(3);

soln10(1) = 0; soln10(2) = s10(1); soln10(3) = 0; soln10(4) = s10(2); soln10(5) = s10(3);

Matrix Collection\_of\_BFS(5, 8);

Collection\_of\_BFS = (soln1 | soln2 | soln5 | soln6 | soln7 | soln8 | soln9 | soln10);

cout << "The Eight Basic-Feasible-Solutions are:" << endl;

cout << setw(9) << setprecision(3) << Collection\_of\_BFS << endl;

cout << "Verification:" << endl;

cout << setw(9) << setprecision(3) << A \* Collection\_of\_BFS << endl;

cout << "Rank of [soln1 | soln2 | soln5 | soln6 | soln7 | soln8 | soln9 | soln10] is: " << Rank(Collection\_of\_BFS);

cout << ". Hence, we need three of the eight solutions." << endl;

Matrix BFS\_subset(5, 3);

BFS\_subset = (soln1|soln2|soln7);

cout << "Rank of [soln1 | soln2 | soln 7] is: " << Rank(BFS\_subset) << endl;

//cout << ". So, soln1 and soln2 are sufficient." << endl;

//cout << "--------------------------------------------------" << endl;

// express soln3 in terms of soln1 and soln2

//ColumnVector x(2);

//x = Find\_Solution(BFS\_subset, soln3);

//cout << "soln3 = (" << x(1) << " \* soln1) + (" << x(2) << " \* soln2)" << endl;

//x = Find\_Solution(BFS\_subset, soln4);

//cout << "soln4 = (" << x(1) << " \* soln1) + (" << x(2) << " \* soln2)" << endl;

}