HW5 - Math5610

Andrew Sheridan

October 27, 2016

Problem 1

LU Factorization with Scaled Partial Pivoting.

The code written below contains calls to other functions contained in the Matrix.h header file. The function **FindArrayMax** finds the entry in vector V of size n with largest magnitude, starting with entry k. **FindMaxIndex** finds the index of the entry with largest magnitude in vector V of size n. When the algorithm completes, for each size i, a vector of ones, size i, is printed. I have also verified that the pivoting is being run by both printing the full matrix at the lower size counts, and printing every time a pivot is executed.

```
//Matrix.h
/// Finds the LU factorization of matrix A. A becomes the upper ↩
    triangular matrix U, and the lower triangular matrix L is returned↩
    .
// A: The nxn coefficient matrix
// b: Right-Hand-Side
// n: The size of the matrices
double** ScaledLUFactorization(double** A, double* b, unsigned n) {
  double** L = CreateIdentityMatrix(n);

  for (int k = 0; k < n; k++) {
    double* ratios = new double[n - k]; // New vector of size n - k to↩
         store the ratios
    for (int i = k; i < n; i++) {
      double rowMax = FindArrayMax(A[i], k, n);
      ratios[i - k] = rowMax / A[i][k];
    }
    int newPivot = FindMaxIndex(ratios, n - k) + k; //Find the best ↩
        row for this iteration

    double* temp = A[k]; //
    A[k] = A[newPivot];  // Switch the current row with the best row ↩
        for this iteration
    A[newPivot] = temp;  //
    double tempEntry = b[k];
    b[k] = b[newPivot];
    b[newPivot] = tempEntry;

    for (int i = k + 1; i < n; i++) {
      double factor = A[i][k] / A[k][k];
      L[i][k] = factor;
      for (int j = k + 1; j < n; j++) {
        A[i][j] = A[i][j] - factor*A[k][j];
      }
      A[i][k] = 0;
      b[i] = b[i] - factor*b[k];
    }
  }

  return L;
}
```

```
//Andrew Sheridan
//Math 5610
//Written in C++

//main.cpp
#include "../../HW4/BackSubstitution/BackSubstitution/Matrix.h"
#include <iostream>

int main(void) {
  //Problem 1
  double** matrix;
  double* vector;
  double* onesVector = CreateOnesVector(size);
  double** matrixCopy;
  double** resultMatrix;
  double* resultVector;

  for (int i = 10; i <= 160;  i *= 2) {
    matrix = CreateDiagonallyDominantMatrix(i);
    vector = CreateOnesVector(i);
    vector = VectorMatrixMultiply(matrix, vector, i);
    matrixCopy = CopyMatrix(matrix, i);
    ScaledLUFactorization(matrixCopy, vector, i);
    resultVector = BackSubstitution(matrixCopy, vector, i);

    std::cout << "Result of LU Factorization and Back Substitution ↩
        with size " << i <<  std::endl;
    PrintVector(resultVector, i);
  }

  return 0;
}
```

Problem 2

Gaussian Elimination with Scaled Partial Pivoting and Back Substitution

Below is my code for Gaussian Elimination with scaled partial pivoting. Other routines, such as *FindArrayMax* and *FindMaxIndex* are found elsewhere in the Matrix.h file. **FindArrayMax** finds the entry in vector V of size n with largest magnitude, starting with entry k. **FindMaxIndex** finds the index of the entry with largest magnitude in vector V of size n. For each iteration of our testing method in *main.cpp*, we obtain a vector of ones, as hoped. I have also ensured that pivoting is working successfully by printing to the console which rows were switched every time a pivot occurs.

```cpp
//Matrix.h

/// Reduces an nxn matrix A and right-hand-side b to upper-triangular form using Gaussian Elimination
// A: The nxn coefficient matrix
// b: Right-Hand-Side
// n: The size of the matrices
double** GaussianEliminationWithScaledPivoting(double** A, double* b, unsigned n) {
  try {
    for (int k = 0; k < n; k++) {
      double* ratios = new double[n - k]; // New vector of size n - k to store the ratios
      for (int i = k; i < n; i++) {
        double rowMax = FindArrayMax(A[i], k, n);
        ratios[i - k] = rowMax / A[i][k];
      }
      int newPivot = FindMaxIndex(ratios, n - k) + k; //Find the best row for this iteration

      double* temp = A[k]; //
      A[k] = A[newPivot];  // Switch the current row with the best row for this iteration
      A[newPivot] = temp;  //

      double tempEntry = b[k];
      b[k] = b[newPivot];
      b[newPivot] = tempEntry;

      for (int i = k + 1; i < n; i++) {
        double factor = A[i][k] / A[k][k];
        for (int j = 0; j < n; j++) {
          A[i][j] = A[i][j] - factor*A[k][j];
        }
        b[i] = b[i] - factor*b[k];
      }
    }
  }
  catch (std::exception& e)
  {
    std::cout << "These matrices are not the correct size." << std::endl;
  }

  return A;
}
```

```cpp
//Andrew Sheridan
//Math 5610
//Written in C++
#include "../../HW4/BackSubstitution/BackSubstitution/Matrix.h"
#include <iostream>

int main(void) {
  double** matrix;
  double* vector;
  double* onesVector = CreateOnesVector(size);
  double** matrixCopy;
  double** resultMatrix;
  double* resultVector;

  //Problem 2
  for (int i = 10; i <= 160; i *= 2) {
    matrix = CreateDiagonallyDominantMatrix(i);
    vector = CreateOnesVector(i);
    vector = VectorMatrixMultiply(matrix, vector, i);
    matrixCopy = CopyMatrix(matrix, i);
    matrixCopy = GaussianEliminationWithScaledPivoting(matrixCopy, vector, i);
    resultVector = BackSubstitution(matrixCopy, vector, i);

    std::cout << "Result of GE with Scaled Pivoting and Back Substitution with size " << i << std::endl;
    PrintVector(resultVector, i);
  }

  return 0;
}
```

Problem 3

Gaussian Elimination with Scaled Partial Pivoting and Back Substitution

The code for my Gaussian Elimination routine (with pivoting) is found in the previous problem. This problem is nearly identical to Problem 2, except that instead of using a starting matrix which is diagonally dominant, we have used a symmetric matrix which is not diagonally dominant. I have confirmed that the pivoting is successful by printing which rows were switched in the Gaussian Elimination routine, and our final result, for each size, is a vector full of ones.

```cpp
//Andrew Sheridan
//Math 5610
//Written in C++
#include "../../HW4/BackSubstitution/BackSubstitution/Matrix.h"
#include <iostream>

int main(void) {
  double** matrix;
  double* vector;
  double* onesVector = CreateOnesVector(size);
  double** matrixCopy;
  double** resultMatrix;
  double* resultVector;

  //Problem 3
  for (int i = 10; i <= 160; i *= 2) {
    matrix = CreateSymmetricMatrix(i);
    vector = CreateOnesVector(i);
    vector = VectorMatrixMultiply(matrix, vector, i);
    matrixCopy = CopyMatrix(matrix, i);
    GaussianElimination(matrixCopy, vector, i);
    resultVector = BackSubstitution(matrixCopy, vector, i);

    std::cout << "Result of Gaussian Elimination and Back Substitution↩
        with size " << i << std::endl;
    PrintVector(resultVector, i);
  }

  return 0;
}
```

Problem 4

5.5

(a) Out goal is to decompose the following matrix into $PA = LU$.

$$\begin{pmatrix} 5 & 6 & 7 & 8 \\ 0 & 4 & 3 & 2 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & -2 \end{pmatrix}$$

This matrix has a zero on its diagonal, and so we can use a permutation matrix to rearrange the third and fourth rows.

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$$PA = \begin{pmatrix} 5 & 6 & 7 & 8 \\ 0 & 4 & 3 & 2 \\ 0 & 0 & -1 & -2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

This resulting matrix is now the upper triangular matrix U, and we can represent the lower triangular matrix L as the identity matrix I. We now have all the desired matrices.

$$U = \begin{pmatrix} 5 & 6 & 7 & 8 \\ 0 & 4 & 3 & 2 \\ 0 & 0 & -1 & -2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

(b) We are given the right hand side $b = (26, 9, 1, -3)^T$, and asked to find x such that $Ax = b$. Page 107 of the text book shows us that $PA = LU$, which is the same as $A = (P^T L)U$. This can be simplified further to $A = (PL)U$, because $P^T = P$. We can then use the set of equations $Ly = b$ and $Ux = y$ to find x. We must first include the permutation matrix in this set of equations, so now $PLy = b$. L is simply the identity matrix, so $Py = b$. We can multiply both sides by the inverse of P, P^-1, to get $y = P^{-1}b$. The inverse of P is equal to P, because $PP = I$. Therefore our new equation is simply $y = Pb$, which gives us the result $y = (26, 9, -3, 1)^T$.

We can now solve for x in $Ux = y$

$$\begin{pmatrix} 5 & 6 & 7 & 8 \\ 0 & 4 & 3 & 2 \\ 0 & 0 & -1 & -2 \\ 0 & 0 & 0 & 1 \end{pmatrix} x = \begin{pmatrix} 26 \\ 9 \\ -3 \\ 1 \end{pmatrix}$$

It is easy to see that each row in U sums to the corresponding entry in y, therefore $x = \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix}^T$.

Problem 5

5.6

If we have matrices A and B, where $A = B - B^T$, we cannot do LU decomposition on A unless we do some pivoting first. This is because the diagonal entries of B and B^T are equal, and when we subtract B^T from B, every diagonal entry in the resulting matrix A will be 0, therefore we cannot carry out LU decomposition.