Scientific Computing Sheet 2

Brandon Tang

```cpp
#include "ConjugateGradient.h"
#include "Vec.h"
#include "Matrix.h"
#include <iostream>

Vec conjugateGradient(const Matrix& A, const Vec& b, int maxIter, double tol){
    int n = b.N;
    Vec x(n, 0.0); // Initial guess is a zero vector
    Vec r = b - A * x;
    Vec p = r;

    for (int i = 0; i < maxIter && r.norm() > tol; ++i) {
        Vec Ap = A * p;
        double alpha = r.dot(r) / p.dot(Ap);
        x = x + alpha * p;
        Vec r_new = r - alpha * Ap;
        double beta = r_new.dot(r_new) / r.dot(r);
        p = r_new + beta * p;
        r = r_new;
    }

    return x;
}
```

```cpp
#include <iostream>
#include <random>

#include "ConjugateGradient.h"
#include "Matrix.h"
#include "Vec.h"

int main() {

    std::random_device rd;   // obtain a random number from hardware
    std::mt19937 gen(rd());  // seed the generator

    // Create a uniform real distribution between 0 and 1
    std::uniform_real_distribution<> dis(0.0, 1.0);

    int N = 10;
    Matrix L = Matrix(N, N);  // a lower triangular matrix
    for (int i = 0; i < N; i++) {
        L.data[i][i] = static_cast<double>(i + 1);
        for (int j = 0; j < i; j++) {
            L.data[i][j] = dis(gen);
        }
    }

    Matrix A = L * transpose(L);  // A is a symmetric positive definite matrix
    Vec b = Vec(N, 0);
    for (int i = 0; i < N; i++) {
        b[i] = dis(gen);
    }

    Vec x = conjugateGradient(A, b, 1000, 1e-16);
    std::cout << "Solution to Ax = b using Conjugate Gradient: " << x << std::endl;
    std::cout << "Residual: " << (A * x - b).norm() << std::endl;

    return 0;
}
```

```
Solution to Ax = b using Conjugate Gradient: −0.147829 0.155398 0.0584987 −0.025608
0.00839462 0.0177763 0.0018769 −0.00137001 0.00396464 0.00210073

Residual: 3.82899e-16
```