

Scientific Computing Sheet 2

Brandon Tang

```
1  #include "ConjugateGradient.h"
2  #include "Vec.h"
3  #include "Matrix.h"
4  #include <iostream>
5
6  Vec conjugateGradient(const Matrix& A, const Vec& b, int maxIter, double tol){
7      int n = b.N;
8      Vec x(n, 0.0); // Initial guess is a zero vector
9      Vec r = b - A * x;
10     Vec p = r;
11
12     for (int i = 0; i < maxIter && r.norm() > tol; ++i) {
13         Vec Ap = A * p;
14         double alpha = r.dot(r) / p.dot(Ap);
15         x = x + alpha * p;
16         Vec r_new = r - alpha * Ap;
17         double beta = r_new.dot(r_new) / r.dot(r);
18         p = r_new + beta * p;
19         r = r_new;
20     }
21
22     return x;
23 }
```

```
1  #include <iostream>
2  #include <random>
3
4  #include "ConjugateGradient.h"
5  #include "Matrix.h"
6  #include "Vec.h"
7
8  int main() {
9
10     std::random_device rd; // obtain a random number from hardware
11     std::mt19937 gen(rd()); // seed the generator
12
13     // Create a uniform real distribution between 0 and 1
14     std::uniform_real_distribution<> dis(0.0, 1.0);
15
16     int N = 10;
17     Matrix L = Matrix(N, N); // a lower triangular matrix
18     for (int i = 0; i < N; i++) {
19         L.data[i][i] = static_cast<double>(i + 1);
20         for (int j = 0; j < i; j++) {
21             L.data[i][j] = dis(gen);
22         }
23     }
24
25     Matrix A = L * transpose(L); // A is a symmetric positive definite matrix
26     Vec b = Vec(N, 0);
27     for (int i = 0; i < N; i++) {
28         b[i] = dis(gen);
29     }
30
31     Vec x = conjugateGradient(A, b, 1000, 1e-16);
32     std::cout << "Solution to Ax = b using Conjugate Gradient: " << x << std::endl;
33     std::cout << "Residual: " << (A * x - b).norm() << std::endl;
34
35     return 0;
36 }
```

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