

Scientific Computing Sheet 4

Brandon Tang

1.

```
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     Vec x2 = conjugateGradientPreconditionedDiagonal(A, b, 1000, 1e-16);
36     std::cout << "Solution to Ax = b using Preconditioned Conjugate Gradient: " <<
x2 << std::endl;
37     std::cout << "Residual: " << (A * x2 - b).norm() << std::endl;
38     return 0;
39 }
```

```
1  Vec conjugateGradientPreconditioned(const Matrix& A, const Vec& b, int maxIter,
double tol, std::function<Vec(Vec)> multMInv) {
2      // Precondition: multMInv(v) = M_inv * v where M is a positive definite
preconditioning matrix.
3      int n = b.N;
4      Vec x(n, 0.0); // Initial guess is a zero vector
5      Vec rh = b;
6      Vec ph = rh;
7      double rhMinvrh = rh.dot(multMInv(rh)); // r_T M_-1 r
8
9      for (int i = 0; i < maxIter && rh.norm() > tol; ++i) {
10         Vec Aph = A * ph;
11         double alpha = rhMinvrh / (ph.dot(Aph));
12         x = x + alpha * ph;
13         rh = rh - alpha * Aph;
```

```

14         double rhMinvrh_new = rh.dot(multMInv(rh));
15         double beta = rhMinvrh_new / rhMinvrh;
16         ph = multMInv(rh) + beta * ph;
17         rhMinvrh = rhMinvrh_new;
18     }
19
20     return x;
21 }
22
23 Vec conjugateGradientPreconditionedDiagonal(const Matrix& A, const Vec& b, int
24 maxIter, double tol){
25     std::vector<double> diagA = A.diag();
26     auto multMInv = [&diagA](Vec v) {
27         Vec result(v.N);
28         for (int i = 0; i < v.N; i++) {
29             result[i] = v[i] / diagA[i];
30         }
31         return result;
32     };
33     Vec v(b.N, 1.0);
34     return conjugateGradientPreconditioned(A, b, maxIter, tol, multMInv);
35 }

```

Solution to $Ax = b$ using Conjugate Gradient: Vector of size 10:

-0.0564121 -0.00548684 0.075781 0.030698 0.0249482 0.0140346 0.00468948 0.0114935
-0.0041178 0.00363854

Residual: 2.28885e-16

Solution to $Ax = b$ using Preconditioned Conjugate Gradient: Vector of size 10:

-0.0564121 -0.00548684 0.075781 0.030698 0.0249482 0.0140346 0.00468948 0.0114935
-0.0041178 0.00363854

Residual: 1.13624e-09