Scientific Computing Sheet 4

Brandon Tang

1.

```
#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;

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

```
Vec conjugateGradientPreconditioned(const Matrix& A, const Vec& b, int maxIter, double tol, std::function<Vec(Vec)> multMInv) {
    // Precondition: multMInv(v) = M_inv * v where M is a positive definite preconditioning matrix.
    int n = b.N;
    Vec x(n, 0.0);  // Initial guess is a zero vector
    Vec rh = b;
    Vec ph = rh;
    double rhMinvrh = rh.dot(multMInv(rh)); // r_T M_-1 r

    for (int i = 0; i < maxIter && rh.norm() > tol; ++i) {
        Vec Aph = A * ph;
        double alpha = rhMinvrh / (ph.dot(Aph));
        x = x + alpha * ph;
        rh = rh - alpha * Aph;
```

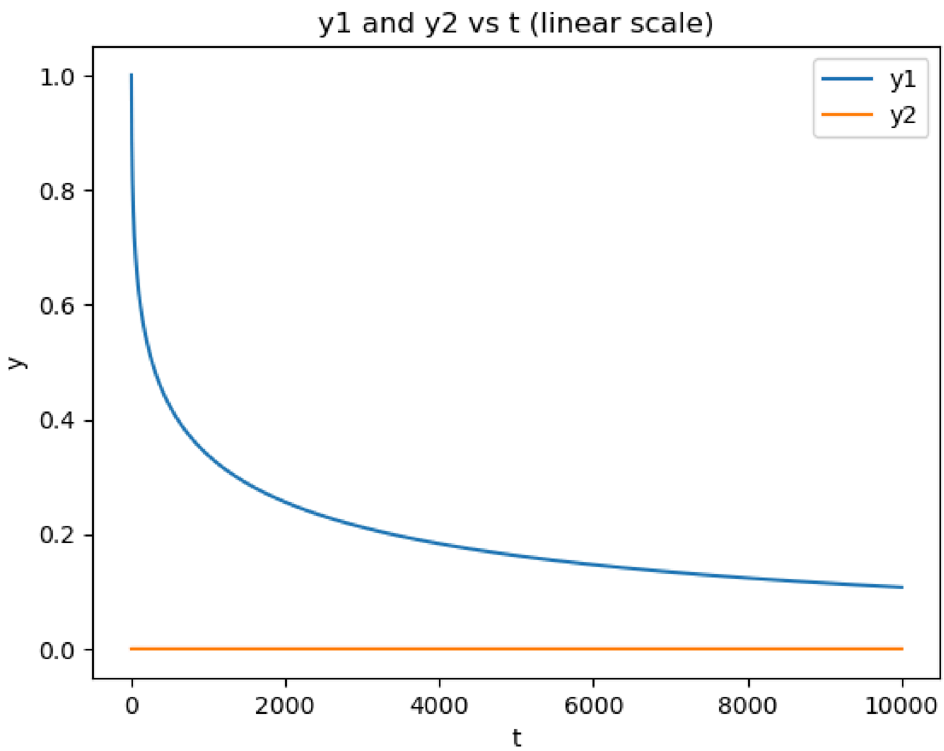
```cpp
        double rhMinvrh_new = rh.dot(multMInv(rh));
        double beta = rhMinvrh_new / rhMinvrh;
        ph = multMInv(rh) + beta * ph;
        rhMinvrh = rhMinvrh_new;
    }

    return x;
}

Vec conjugateGradientPreconditionedDiagonal(const Matrix& A, const Vec& b, int maxIter, double tol){
    std::vector<double> diagA = A.diag();
    auto multMInv = [&diagA](Vec v) {
        Vec result(v.N);
        for (int i = 0; i < v.N; i++) {
            result[i] = v[i] / diagA[i];
        }
        return result;
    };

    Vec v(b.N, 1.0);
    return conjugateGradientPreconditioned(A, b, maxIter, tol, multMInv);
}
```

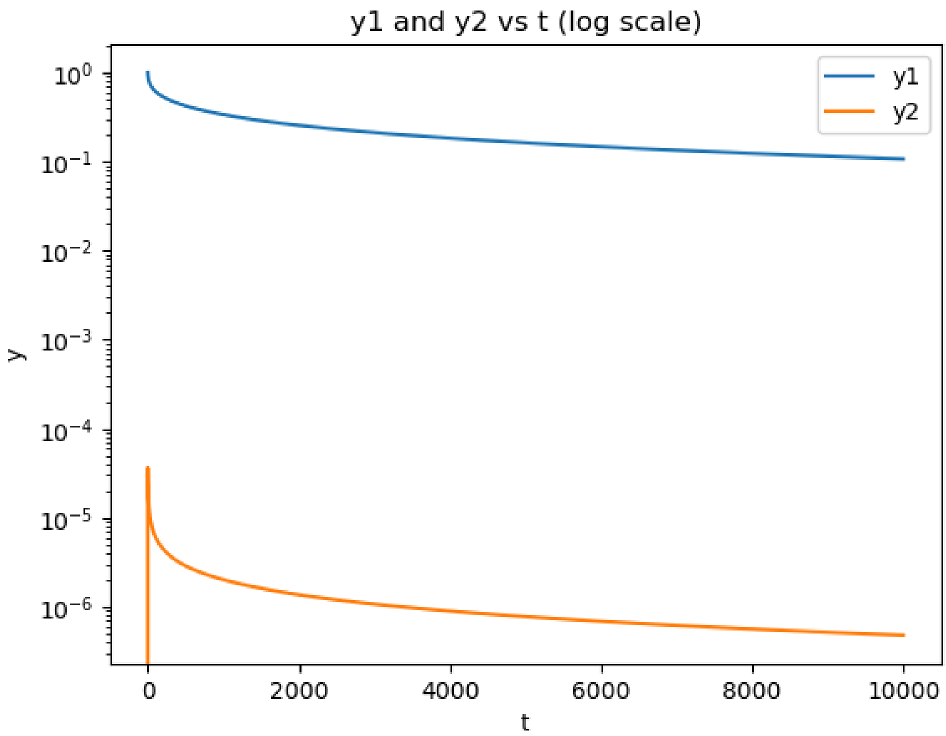
```
Solution to Ax = b using Conjugate Gradient: Vector of size 10:
-0.0564121 -0.00548684 0.075781 0.030698 0.0249482 0.0140346 0.00468948 0.0114935
-0.0041178 0.00363854

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
```

6.

```python
# %%
import numpy as np
import matplotlib.pyplot as plt
from tqdm.notebook import tqdm
def iterate(y_prev, h):
    y1 = y_prev[0]
    y2 = y_prev[1]
    z = -0.04 * y1 + 1e4 * y2 * (1 - y1 - y2)
    y1_new = y1 + h * z
    y2_new = y2 + h * (-z - 3e7 * y2**2)

    return y1_new, y2_new

# %%
y1 = 1
y2 = 0

T = 10000
h = 0.0001
ts = np.arange(0, T, h)

```

```
y1s = np.zeros_like(ts)
y2s = np.zeros_like(ts)

y1s[0] = y1
y2s[0] = y2

for i in tqdm(range(1, len(ts))):
    y1, y2 = iterate((y1, y2), h)
    y1s[i] = y1
    y2s[i] = y2

# %%
sampledy1s = y1s[::100]
sampledy2s = y2s[::100]
sampledts = ts[::100]

plt.plot(sampledts, sampledy1s, label='y1')
plt.plot(sampledts, sampledy2s, label='y2')
plt.legend()
plt.xlabel('t')
plt.ylabel('y')
plt.show()

# %%
plt.plot(sampledts, sampledy1s, label='y1')
plt.plot(sampledts, sampledy2s, label='y2')
plt.legend()
plt.xlabel('t')
plt.ylabel('y')
plt.yscale('log')
plt.show()
```




y1 and y2 vs t (log scale)
y1
y2
10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
y
0
2000
4000
6000
8000
10000
t