Solving $Ax = b$ Using Iterative Methods

Eric Rice
ericjrice95@gmail.com

Source code for all computations is given at the end of this submission.

 # 1. *Consider the matrix $A \in \mathbb{R}^{4\times 4}$ and the vector $b \in \mathbb{R}^4$ given by*

$$A = \begin{bmatrix} 4 & 1 & 1 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & 1 & -2 & 0 \\ 2 & 0 & 0 & 4 \end{bmatrix}, \quad b = \begin{bmatrix} -4 \\ 1 \\ -5 \\ -8 \end{bmatrix}.$$

Use the starting vector $x^{(0)} = 0$ and conduct five steps each of the Jacobi iteration, the forward Gauss-Seidel iteration, and compute the Euclidean norms of the errors.

Let $x^* = (-2, 1, 3, -1)$. It is easily verified that x^* is an exact solution to $Ax = b$. We use this x^* in our calculation of the errors at each step of the Jacobi and Gauss-Seidel iterations.

```
Jacobi Calculation for (1):

x^(1) = { −1, 0.5, 2.5, −2 }
    with error ||x* − x^(1)|| = 1.58114

x^(2) = { −1.75, 1.25, 2.75, −1.5 }
    with error ||x* − x^(2)|| = 0.661438

x^(3) = { −2, 1, 3.125, −1.125 }
    with error ||x* − x^(3)|| = 0.176777

x^(4) = { −2.03125, 1.0625, 3, −1 }
    with error ||x* − x^(4)|| = 0.0698771

x^(5) = { −2.01563, 0.984375, 3.03125, −0.984375 }
    with error ||x* − x^(5)|| = 0.0413399


Gauss−Seidel Calculation for (1):

x^(1) = { −1, 0, 2.5, −1.5 }
    with error ||x* − x^(1)|| = 1.58114
```

```
x^(2) = { −1.625, 0.9375, 2.96875, −1.1875 }
    with error ||x* − x^(2)|| = 0.425046

x^(3) = { −1.97656, 0.996094, 2.99805, −1.01172 }
    with error ||x* − x^(3)|| = 0.0265654

x^(4) = { −1.99854, 0.999756, 2.99988, −1.00073 }
    with error ||x* − x^(4)|| = 0.00166034

x^(5) = { −1.99991, 0.999985, 2.99999, −1.00005 }
    with error ||x* − x^(5)|| = 0.000103771
```

 # 2. *Consider the symmetric positive definite matrix* $A \in \mathbb{R}^{4\times 4}$ *and the vector* $b \in \mathbb{R}^4$ *given by*

$$A = \begin{bmatrix} 1 & -1 & -1 & -1 \\ -1 & 2 & 2 & 2 \\ -1 & 2 & 3 & 1 \\ -1 & 2 & 1 & 4 \end{bmatrix}, \quad b = \begin{bmatrix} -1 \\ 1 \\ 6 \\ -7 \end{bmatrix}.$$

Use the starting vector $x^{(0)} = 0$ *and conduct two steps of the steepest descent algorithm. Compute the errors for each approximate solution.*

Let $x^* = (-1, 1, 2, -3)$. It is easily verified that x^* is the exact solution to $Ax = b$. We use this x^* in our calculation of the errors at each step of the steepest descent iteration.

```
Steepest Descent Calculation for (2):

x^(1) = { −0.39726, 0.39726, 2.38356, −2.78082 }
    with error ||x* − x^(1)|| = 0.960078

x^(2) = { −0.493414, 0.493414, 2.45349, −2.69341 }
    with error ||x* − x^(2)|| = 0.901616
```

 # 3. *Consider the symmetric positive definite matrix* $A \in \mathbb{R}^{3\times 3}$ *and the vector* $b \in \mathbb{R}^3$ *given by*

$$A = \begin{bmatrix} 4 & 1 & 2 \\ 1 & 9 & 1 \\ 2 & 1 & 16 \end{bmatrix}, \quad b = \begin{bmatrix} 0 \\ 18 \\ 16 \end{bmatrix}.$$

Use the starting vector $x^{(0)} = 0$ *and conduct three steps of the steepest descent algorithm with and without Jacobi preconditioning. Compute the errors for each approximate solution.*

Let $x^* = (-1, 2, 1)$. It is easily verified that x^* is the exact solution to $Ax = b$. We use this x^* in our calculation of the errors at each step of the steepest descent iteration (preconditioned and otherwise).

```
Steepest Descent Calculation for (3):

x^(1) = { 0, 1.37586, 1.22298 }
    with error ||x* - x^(1)|| = 1.1997

x^(2) = { -0.358687, 1.78827, 0.759016 }
    with error ||x* - x^(2)|| = 0.717066

x^(3) = { -0.538975, 1.93327, 1.02728 }
    with error ||x* - x^(3)|| = 0.466627


(Jacobi) Preconditioned Steepest Descent Calculation for (3):

x^(1) = { 0, 1.85714, 0.928571 }
    with error ||x* - x^(1)|| = 1.01267

x^(2) = { -0.90562, 1.89584, 0.885032 }
    with error ||x* - x^(2)|| = 0.181586

x^(3) = { -0.915909, 1.99672, 0.988976 }
    with error ||x* - x^(3)|| = 0.0848735
```

```
1  // SomeIterativeMethods.h
2  // Eric Rice
3
4  #ifndef SomeIterativeMethods_h
5  #define SomeIterativeMethods_h
6
7  #include <vector>
8  #include <iostream>
9  using namespace std;
10
11 void print(vector<double> const &input);
12
13 double dot(vector<double> u, vector<double> v);
14
15 vector<double> vecAdd(vector<double> u, vector<double> v);
16
17 vector<double> scalMult(double c, vector<double> v);
18
19 vector<double> matMul(vector<vector<double> > A, vector<double> x);
20
21 void jacobi(vector<vector<double> > A, vector<double> b, vector<double> &x);
22
23 void gaussSeidel(vector<vector<double> > A, vector<double> b, vector<double> &x);
24
25 void steepestDescent(vector<vector<double> > A, vector<double> b, vector<double>
     &x);
26
27 void preconSteepDesc(vector<vector<double> > A, vector<double> b,
28     vector<double> &x, vector<vector<double> > Pinv);
29
30 #endif
```

```cpp
// SomeIterativeMethods_funcs.cpp
// Eric Rice

#include "SomeIterativeMethods.h"

void print(vector<double> const &input)
{
    cout << "{ ";
    for (int i = 0; i < input.size() - 1; i++)
    {
        cout << input.at(i) << ", ";
    }
    cout << input.back() << " }";
}

double dot(vector<double> u, vector<double> v)
{
    // Standard Euclidean dot product

    double summ = 0.0;

    for (int i = 0; i < u.size(); i++)
    {
        summ += u.at(i) * v.at(i);
    }
    return summ;
}

vector<double> vecAdd(vector<double> u, vector<double> v)
{
    // Standard vector addition

    vector<double> result(u.size(), 0.0);

    for (int i = 0; i < u.size(); i++)
    {
        result.at(i) = u.at(i) + v.at(i);
    }
    return result;
}

vector<double> scalMult(double c, vector<double> v)
{
    // Standard scalar multiplication

    vector<double> result(v.size(), 0.0);

    for (int i = 0; i < v.size(); i++)
    {
        result.at(i) = c * v.at(i);
    }
    return result;
```

```cpp
}

vector<double> matMul(vector<vector<double> > A, vector<double> x)
{
    // Computes the matrix product Ax

    vector<double> result(x.size(), 0.0);

    for (int i = 0; i < x.size(); i++)
    {
        result.at(i) = dot(A.at(i), x);
    }
    return result;
}

void jacobi(vector<vector<double> > A, vector<double> b, vector<double> &x)
{
    /* Computes one iteration of the Jacobi method for the system Ax = b, with
    the initial guess x.*/

    vector<double> y(b.size(), 0.0);

    for (int i = 0; i < b.size(); i++)
    {
        double sum_below_i = 0.0;
        double sum_above_i = 0.0;

        for (int j = 0; j < i; j++)
        {
            sum_below_i += A.at(i).at(j) * x.at(j);
        }
        for (int j = i + 1; j < b.size(); j++)
        {
            sum_above_i += A.at(i).at(j) * x.at(j);
        }
        y.at(i) = b.at(i) - sum_below_i - sum_above_i;
    }
    for (int i = 0; i < b.size(); i++)
    {
        x.at(i) = (1 / A.at(i).at(i)) * y.at(i);
    }
}

void gaussSeidel(vector<vector<double> > A, vector<double> b, vector<double> &x)
{
    /* Computes one iteration of the Gauss-Seidel method for the system Ax = b,
      with
    the initial guess x.*/

    vector<double> y(b.size(), 0.0);

    for (int i = 0; i < b.size(); i++)
```

```
104     {
105         double sum_above_i = 0.0;
106 
107         for (int j = i + 1; j < b.size(); j++)
108         {
109             sum_above_i += A.at(i).at(j) * x.at(j);
110         }
111         y.at(i) = b.at(i) - sum_above_i;
112     }
113     for (int i = 0; i < b.size(); i++)
114     {
115         double sum_below_i = 0.0;
116 
117         for (int j = 0; j < i; j++)
118         {
119             sum_below_i += A.at(i).at(j) * x.at(j);
120         }
121         x.at(i) = (1 / A.at(i).at(i)) * (y.at(i) - sum_below_i);
122     }
123 }
124 
125 void steepestDescent(vector<vector<double> > A, vector<double> b, vector<double>
      &x)
126 {
127     /* Computes one iteration of the steepest descent method for the system Ax =
        b,
128     with the initial guess x.*/
129 
130     vector<double> r = vecAdd(b, scalMult(-1, matMul(A, x)));
131     x = vecAdd(x, scalMult(dot(r, r) / dot(matMul(A, r), r), r));
132 }
133 
134 void preconSteepDesc(vector<vector<double> > A, vector<double> b,
135     vector<double> &x, vector<vector<double> > Pinv)
136 {
137     /* Computes one iteration of the preconditioned steepest descent method for
        the system Ax = b,
138     with the initial guess x and inverse of preconditioner Pinv.*/
139 
140     vector<double> r = vecAdd(b, scalMult(-1, matMul(A, x)));
141     vector<double> z = matMul(Pinv, r);
142     x = vecAdd(x, scalMult(dot(r, z) / dot(matMul(A, z), z), z));
143 }
```

```cpp
// main.cpp
// Eric Rice

#include "SomeIterativeMethods.h"

vector<vector<double> > A;
vector<double> b;
vector<double> x;
vector<double> xstar;
vector<double> error;
vector<vector<double> > Pinv;

int main()
{
    A = { {  4.0,  1.0,  1.0,  0.0 },
          { -1.0,  2.0, -1.0,  0.0 },
          {  0.0,  1.0, -2.0,  0.0 },
          {  2.0,  0.0,  0.0,  4.0 } };
    b = { -4.0, 1.0, -5.0, -8.0 };
    xstar = { -2.0, 1.0, 3.0, -1.0 };

    x = { 0.0, 0.0, 0.0, 0.0 };
    cout << "Jacobi Calculation for (1):\n" << endl;
    for (int i = 1; i <= 5; i++)
    {
        jacobi(A, b, x);
        cout << "x^(" << i << ") = ";
        print(x);
        error = vecAdd(xstar, scalMult(-1, x));
        cout << "\n    with error ||x* - x^(" << i << ")|| = "
             << sqrt(dot(error, error)) << "\n\n";
    }

    x = { 0.0, 0.0, 0.0, 0.0 };
    cout << "\nGauss-Seidel Calculation for (1):\n" << endl;
    for (int i = 1; i <= 5; i++)
    {
        gaussSeidel(A, b, x);
        cout << "x^(" << i << ") = ";
        print(x);
        error = vecAdd(xstar, scalMult(-1, x));
        cout << "\n    with error ||x* - x^(" << i << ")|| = "
             << sqrt(dot(error, error)) << "\n\n";
    }

    A = { {  1.0, -1.0, -1.0, -1.0 },
          { -1.0,  2.0,  2.0,  2.0 },
          { -1.0,  2.0,  3.0,  1.0 },
          { -1.0,  2.0,  1.0,  4.0 } };
    b = { -1.0, 1.0, 6.0, -7.0 };
    xstar = { -1.0, 1.0, 2.0, -3.0 };

```

```
53      x = { 0.0, 0.0, 0.0, 0.0 };
54      cout << "\nSteepest Descent Calculation for (2):\n" << endl;
55      for (int i = 1; i <= 2; i++)
56      {
57          steepestDescent(A, b, x);
58          cout << "x^(" << i << ") = ";
59          print(x);
60          error = vecAdd(xstar, scalMult(-1, x));
61          cout << "\n    with error ||x* - x^(" << i << ")|| = "
62               << sqrt(dot(error, error)) << "\n\n";
63      }
64
65      A = { { 4.0,  1.0,  2.0 },
66            { 1.0,  9.0,  1.0 },
67            { 2.0,  1.0, 16.0 } };
68      b = { 0.0, 18.0, 16.0 };
69      xstar = { -1.0, 2.0, 1.0 };
70
71      x = { 0.0, 0.0, 0.0 };
72      cout << "\nSteepest Descent Calculation for (3):\n" << endl;
73      for (int i = 1; i <= 3; i++)
74      {
75          steepestDescent(A, b, x);
76          cout << "x^(" << i << ") = ";
77          print(x);
78          error = vecAdd(xstar, scalMult(-1, x));
79          cout << "\n    with error ||x* - x^(" << i << ")|| = "
80               << sqrt(dot(error, error)) << "\n\n";
81      }
82
83      Pinv = { { 1.0 / 4.0,       0.0,        0.0 },
84               {       0.0, 1.0 / 9.0,        0.0 },
85               {       0.0,       0.0, 1.0 / 16.0 } };
86
87      x = { 0.0, 0.0, 0.0 };
88      cout << "\n(Jacobi) Preconditioned Steepest Descent Calculation for (3):\n"
          << endl;
89      for (int i = 1; i <= 3; i++)
90      {
91          preconSteepDesc(A, b, x, Pinv);
92          cout << "x^(" << i << ") = ";
93          print(x);
94          error = vecAdd(xstar, scalMult(-1, x));
95          cout << "\n    with error ||x* - x^(" << i << ")|| = "
96              << sqrt(dot(error, error)) << "\n\n";
97      }
98
99      cin.get();
100     return 0;
101 }
```