Assignment 3 - Linear Algebra, 2019

Alex Hiller

April 12, 2019

Question 1

Jacobi's method for iteration is given by:

$$x_i^{k+1} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1,j\neq i}^{n} a_{ij}x_j\right) \tag{1}$$

Using an initial guess of $\mathbf{x}^{(0)} = \mathbf{0}$, the code in the appendix produces the following output for four iterations:

```
---------------------------------------------------------------------------------
x_(1):
-4.000000                       2.000000                        6.000000

relative precision w/ supremum norm is: 1.000000


x_(2):
2.000000                        7.000000                        9.333334

relative precision w/ supremum norm is: 0.642857


x_(3):
5.333334                        5.666667                        7.000000

relative precision w/ supremum norm is: 0.476191


x_(4):
3.000000                        2.833333                        4.333333

relative precision w/ supremum norm is: 0.538462
---------------------------------------------------------------------------------
```

Question 2

The Gauss-Siedel method for iteration is given by:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right) \tag{2}$$

Using an initial guess of $\mathbf{x}^{(0)} = \mathbf{0}$, the code in the appendix produces the following output for three iterations:

```
--------------------------------------------------------------------------------
x_(1):
-4.000000                        4.000000                        10.000000

relative precision w/ supremum norm is: 1.000000


x_(2):
6.000000                         4.000000                        3.333333

relative precision w/ supremum norm is: 1.666667


x_(3):
-0.666667                        4.000000                        7.777778

relative precision w/ supremum norm is: 0.571429
--------------------------------------------------------------------------------
```

Question 3

The relaxation parameter method uses the value of ω to determine how much to weight the previous value of $x^{(k)}$ and is given by the formula:

$$x_i^{(k+1)} = x_i^{(k)} + \frac{\omega}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i}^{n} a_{ij} x_j^{(k)} \right) \tag{3}$$

Setting $\omega = 0.5$ and using an initial guess of $\mathbf{x}^{(0)} = \mathbf{0}$, the code in the appendix produces the following output for one iteration:

```
-------------------------------------------------------------------------------
x_(1):
-3.600000                       3.420000                        8.585999

relative precision w/ supremum norm is: 1.000000
-------------------------------------------------------------------------------
```

Question 4

If the matrix is diagonally dominant, then it is known that Gauss-Siedel and Jacobi methods will converge.

For our matrix, $\mathbf{A}$:

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 2 & -1 \\ 2 & -1 & 3 \end{bmatrix}$$

We can see that it is diagonally dominant (but not strictly diagonally dominant).

And so it should converge for the Jacobi and Gauss-Siedel methods, which it does, as we've seen in previous questions.

Question 5

Part (a)

For the system $\mathbf{Ax} = \mathbf{b}$, the relative number of iterations amongst the three methods are summarised below.

Method	**Number of Iterations**
Jacobi	23
Gauss-Siedel	25
Under Relaxation ($\omega = 0.5$)	17

Part (b)

Here with $\mathbf{x}^{(0)} = \mathbf{0}$, the relative precision being 10^{-4} with the supremum norm and varying $w \in \{0.1, 0.2, \dots, 0.9\}$ we can see the optimal value for under- relaxation is $\omega = 0.8$.

ω	**Number of Iterations**
0.1	45
0.2	31
0.3	27
0.4	21
0.5	17
0.6	13
0.7	10
0.8	8
0.9	9

Appendix

```
/*
 * Author: Alex Hiller
 * Year: 2019
 *
 * Program Description:
 *    Implementation of the Jacobi, Gauss-Siedel and relaxation parameter
 *    algorithm for approaching solutions for x of Ax=b     x = A^-1 b
 *
 */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>


enum {JACOBI, GAUSS_SIEDEL, PARAMETER};
enum {CONTINUE, STOP};

/* Params for tuning the approximation */
#define INITIAL_GUESS   {0,0,0}     /* Initial guess for all methods for x0 */
#define ITERS           1000        /* Number of maximum iterations */
#define OMEGA           0.5         /* Value of relaxation parameter */
#define EPSILON         0.0001      /* Precision to stop at */
#define METHOD          PARAMETER   /* Changed according to method desired */


float* copyMat(float* val, float* store);
void printMat1d(float* mat);
float supNorm(float* arr, int arrLen);
int stopCriterion(float* current, float* prev, int arrlen);


int
main (int argc, char *argv[]) {
    float x_k[3]   = INITIAL_GUESS;
    float x_km1[3] = INITIAL_GUESS;

    float A[3][3] = {{1,0,-1},{1,2,-1},{2,-1,3}};
    float b[3] = {-4, 4, 18};

    int iter; /* Iteration number */
    int i;
    int j;


    switch(METHOD) {
        /* Method 1: Relaxation parameter method */
        case(PARAMETER):
            for (iter=0; iter<ITERS; iter++) {
                for (i = 0; i<3; i++) {
                    float sum1=0, sum2=0;
                    for (j=0; j<i; j++) {
                        sum1 += A[i][j]*x_k[j];
                    }
                    for (j=i; j<4; j++) {
                        sum2 += A[i][j]*x_k[j];
                    }
                    x_k[i] += (OMEGA/(A[i][i]))*(b[i] - sum1 - sum2);
                }
                printf("x_(%i):\n", iter+1);
                printMat1d(x_k);
                printf("\n");
                if (stopCriterion(x_k, x_km1, 3))
                    break;
                copyMat(x_k, x_km1);
```

```c
            }
            break;


        /* Method 2: Jacobi iteration */
        case(JACOBI):
            for (iter=0; iter<ITERS; iter++) {
                for (i = 0; i<3; i++) {
                    float sum = 0;
                    for (j=0; j<3; j++) {
                        if (j == i)
                            continue;
                        sum += A[i][j]*x_km1[j];
                    }
                    x_k[i] = (1/(A[i][i]))*(b[i] - sum);
                }
                printf("x_(%i):\n", iter+1);
                printMat1d(x_k);
                printf("\n");
                if (stopCriterion(x_k, x_km1, 3))
                    break;
                /* Copy over results from this iteration */
                copyMat(x_k, x_km1);
            }
            break;

        /* Method 3: Gauss-Siedel */
        case(GAUSS_SIEDEL):
            for (iter=0; iter<ITERS; iter++) {
                for (i = 0; i<3; i++) {
                    float sum1=0, sum2=0;
                    for (j=0; j<i; j++) {
                        sum1 += A[i][j]*x_k[j];
                    }
                    for (j=i+1; j<3; j++) {
                        sum2 += A[i][j]*x_k[j];
                    }
                    x_k[i] = (1/(A[i][i]))*(b[i] - sum1 - sum2);
                }
                printf("x_(%i):\n", iter+1);
                printMat1d(x_k);
                printf("\n");
                if (stopCriterion(x_k, x_km1, 3))
                    break;
                /* Copy over results from this iteration */
                copyMat(x_k, x_km1);
            }
            break;
    }


    return 0;
}


/* Return true if algorithm should stop. */
int
stopCriterion(float* current, float* prev, int arrlen) {
    float difference[arrlen];
    int i;
    for (i=0; i<arrlen; i++)
        difference[i] = current[i]-prev[i];

    float num = supNorm(difference, 3);
    float den = supNorm(current, 3);
    float calc = fabs(num)/fabs(den);
    printf("relative precision w/ supremum norm is: %f\n\n\n", calc);
    if (calc < EPSILON) {
        printf("[STOPPING]\nRelative precision of %f has been reached.\n",
                EPSILON);
```

```
        return STOP;
    }
    else
        return CONTINUE;

    return 2;
}

float
supNorm(float* arr, int arrlen) {
    int i;
    float max = (arr[0]);
    for (i=0; i<arrlen; i++) {
        if (max < (arr[i]))
            max = (arr[i]);
    }
    return (max);
}

void
printMat1d(float* mat) {
    printf("%.6f\t\t\t%.6f\t\t\t%.6f\n", mat[0], mat[1], mat[2]);
}

float*
copyMat(float* val, float* store) {
    int i;
    for (i=0; i<3; i++)
        store[i] = val[i];
    return store;
}
```