Math 4610 – HW 9

Jake Daniels A02307117

November 29, 2022

Task 1:

For the inverse power method I wrote the following code:

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

double inv_pm(int n, double A[n][n], double x[n], double lambda, double tol, int max_iter);
double vector_norm(int n, double x[n]);
double dot_product(int n, double x[n], double y[n]);
void vecScalar(int n, double a, double x[n], double y[n]);
void jacobi_iter(int n, double A[n][n], double x[n], double b[n], double tol, int max_iter);
void matrix_vector_mult(int n, double A[n][n], double x[n], double y[n]);

double inv_pm(int n, double A[n][n], double x[n], double lambda, double tol, int max_iter)
{
    double error = 10.0 * tol;
    double y[n], w[n], z[n];
    int iter = 0;
    jacobi_iter(n, A, y, x, tol, max_iter);
    while (error > tol && iter < max_iter)
    {
        vecScalar(n, 1 / vector_norm(n,y), y, z);
        jacobi_iter(n, A, w, z, tol, max_iter);
        double lambda_new = dot_product(n, z, w);
        error = fabs(lambda_new - lambda);
        lambda = lambda_new;
        vecScalar(n, 1 / vector_norm(n, w), w, y);
        iter++;
    }
    return 1 / lambda;
```

As well as the following code to test it:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <omp.h>

double inv_pm(int n, double A[n][n], double x[n], double lambda, double tol, int max_iter);
double vector_norm(int n, double x[n]);
double dot_product(int n, double x[n], double y[n]);
void vecScalar(int n, double a, double x[n], double y[n]);
void jacobi_iter(int n, double A[n][n], double x[n], double b[n], double tol, int max_iter);
void matrix_vector_mult(int n, double A[n][n], double x[n], double y[n]);

void main()
{;
    int n = 3;
    double A[n][n], x[n];
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (i == j)
            {
                A[i][j] = i + 4.0;
            }
```

```
            else
            {
                A[i][j] = 1;
            }
        }
        x[i] = 1;
    }
    int max_iter = 100;
    double tol = 1e-6;
    double lambda = 0.0;
    double lambda_new = inv_pm(n, A, x, lambda, tol, max_iter);
    printf("lambda = %f", lambda_new);
}
```

Here is a screenshot of me compiling and running my code:

```
jdsoc@Jakes-Laptop ~
$ cd /cygdrive/c/Users/jdsoc/Documents/math4610/Linear_Algebra_Code/C\ Code/

jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/Documents/math4610/Linear_Algebra_Code/C Code
$ gcc -o task91 ts9.c inv_pm.c jacobi_iter.c vecScalar.c dot.c norm.c mv_mult.c

jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/Documents/math4610/Linear_Algebra_Code/C Code
$ ./task91.exe
lambda = 3.324943
jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/Documents/math4610/Linear_Algebra_Code/C Code
$
```

According to Wolfram Alpha this is in fact the smallest eigenvalue so my code works correctly!

Task 2:

For the shifted inverse theorem I wrote the following code:

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

double shifted_inv_pm(int n, double A[n][n], double x[n], double lambda, double tol, int max_iter);
double vector_norm(int n, double x[n]);
double dot_product(int n, double x[n], double y[n]);
void vecScalar(int n, double a, double x[n], double y[n]);
void jacobi_iter(int n, double A[n][n], double x[n], double b[n], double tol, int max_iter);
void matrix_vector_mult(int n, double A[n][n], double x[n], double y[n]);

double shifted_inv_pm(int n, double A[n][n], double x[n], double lambda, double tol, int max_iter)
{
    double error = 10.0 * tol;
    double y[n], w[n], z[n];
    int iter = 0;
    double shift = lambda;
    for (int i = 0; i < n; i++)
    {
        A[i][i] = A[i][i] - lambda;
    }
    jacobi_iter(n, A, y, x, tol, max_iter);
    while (error > tol && iter < max_iter)
```

```
    {
        vecScalar(n, 1 / vector_norm(n,y), y, z);
        jacobi_iter(n, A, w, z, tol, max_iter);
        double lambda_new = dot_product(n, z, w);
        error = fabs(lambda_new - lambda);
        lambda = lambda_new;
        vecScalar(n, 1 / vector_norm(n, w), w, y);
        iter++;
    }
    return 1 / lambda + shift;
}
```

I wrote the following code to test the above code:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <omp.h>

double vector_norm(int n, double x[n]);
double dot_product(int n, double x[n], double y[n]);
void vecScalar(int n, double a, double x[n], double y[n]);
void jacobi_iter(int n, double A[n][n], double x[n], double b[n], double tol, int max_iter);
void matrix_vector_mult(int n, double A[n][n], double x[n], double y[n]);
double shifted_inv_pm(int n, double A[n][n], double x[n], double lambda, double tol, int max_iter);

void main()
{;
    int n = 3;
    double A[n][n], x[n];
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (i == j)
            {
                A[i][j] = i + 4.0;
            }
            else
            {
                A[i][j] = 0.1;
            }
        }
        x[i] = 1;
    }
    int max_iter = 100;
    double tol = 1e-6;
    double lambda = 4.8;
    double lambda_new = shifted_inv_pm(n, A, x, lambda, tol, max_iter);
    printf("lambda = %f", lambda_new);
}
```

Here is a screenshot of me compiling and running the above code:

```
jdsoc@Jakes-Laptop ~
$ cd /cygdrive/c/Users/jdsoc/Documents/math4610/Linear_Algebra_Code/C\ Code/

jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/Documents/math4610/Linear_Algebra_Code/C Code
$ gcc -o task92 ts9.c shifted_inv_pm.c jacobi_iter.c vecScalar.c dot.c norm.c mv_mult.c

jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/Documents/math4610/Linear_Algebra_Code/C Code
$ ./task92.exe
lambda = 4.998086
jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/Documents/math4610/Linear_Algebra_Code/C Code
$
```

Which according to Wolfram Alpha is the correct middle eigenvalue.

Task 3:

For this section I wrote the following code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

void e_val(int n, int part, double lambdas[part], double A[n][n], double x[n], double lambda,
    double tol, int max_iter);
double shifted_inv_pm(int n, double A[n][n], double x[n], double lambda, double tol, int max_iter);
double inv_pm(int n, double A[n][n], double x[n], double lambda, double tol, int max_iter);
double power_method(int n, double A[n][n], double x[n], double lambda, double tol, int max_iter);
double vector_norm(int n, double x[n]);
double dot_product(int n, double x[n], double y[n]);
void vecScalar(int n, double a, double x[n], double y[n]);
void jacobi_iter(int n, double A[n][n], double x[n], double b[n], double tol, int max_iter);
void matrix_vector_mult(int n, double A[n][n], double x[n], double y[n]);

void e_val(int n, int part, double lambdas[part + 2], double A[n][n], double x[n], double lambda,
    double tol, int max_iter)
{
    double ones[n], reset[n][n];
    reset[n][n] = A[n][n];
    for (int i = 0; i < n; i++)
    {
        ones[i] = 1.0;
    }
    double lambda_max = power_method(n, A, x, lambda, tol, max_iter);
    x[n] = ones[n];
    double lambda_min = inv_pm(n, A, x, lambda, tol, max_iter);
    x[n] = ones[n];
    lambdas[0] = lambda_max;
    lambdas[part + 1] = lambda_min;
    double partition = (lambda_max - lambda_min) / part;
    lambda = lambda_min + partition;
    for (int i = 1; i < part + 1; i++)
    {
        lambdas[i] = shifted_inv_pm(n, A, x, lambda, tol, max_iter);
        lambda = lambda_min + partition * (i + 1);
        x[n] = ones[n];
        A[n][n] = reset[n][n];
```

```
    }
}
```

As well as the following code to test it:

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

void e_val(int n, int part, double lambdas[part], double A[n][n], double x[n], double lambda,
    double tol, int max_iter);
double shifted_inv_pm(int n, double A[n][n], double x[n], double lambda, double tol, int max_iter);
double inv_pm(int n, double A[n][n], double x[n], double lambda, double tol, int max_iter);
double power_method(int n, double A[n][n], double x[n], double lambda, double tol, int max_iter);
double vector_norm(int n, double x[n]);
double dot_product(int n, double x[n], double y[n]);
void vecScalar(int n, double a, double x[n], double y[n]);
void jacobi_iter(int n, double A[n][n], double x[n], double b[n], double tol, int max_iter);
void matrix_vector_mult(int n, double A[n][n], double x[n], double y[n]);

void main()
{
    int n = 3;
    double A[n][n], x[n];
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (i == j)
            {
                A[i][j] = i + 4.0;
            }
            else
            {
                A[i][j] = 0.1;
            }
        }
        x[i] = 1;
    }
    int max_iter = 1000;
    double tol = 1e-6;
    double lambda = 0.0;
    double max, min;
    int part = n - 1;
    double lambdas[part + 2];
    e_val(n, part, lambdas, A, x, lambda, tol, max_iter);
    printf("\nlambda max: %f", lambdas[0]);
    for (int i = 1; i < part + 1; i++)
    {
        if (lambdas[i] > lambdas[part + 1] && lambdas[i] < lambdas[0])
        {
            printf("\nlambda: %f", lambdas[i]);
        } else {
            printf("");
        }
    }
    printf("\nlambda min: %f", lambdas[part + 1]);
}
```

Here is a screenshot of me compiling and running the code:

```
jdsoc@Jakes-Laptop ~
$ cd /cygdrive/c/Users/jdsoc/Documents/math4610/Linear_Algebra_Code/C\ Code/

jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/Documents/math4610/Linear_Algebra_Code/C Code
$ gcc -o task93 ts9.c shifted_inv_pm.c inv_pm.c power_method.c jacobi_iter.c vecScalar.c dot.c norm.c mv_mult.c e_val.c

jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/Documents/math4610/Linear_Algebra_Code/C Code
$ ./task93.exe

lambda max: 6.093022
lambda: 4.998139
lambda min: 3.986161
jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/Documents/math4610/Linear_Algebra_Code/C Code
$
```

Which matches with what we know the middle eigenvalue should be above, as well as what Wolfram Alpha says the largest and smallest eigenvalues should be. So therefore we have found all the eigenvalues.

Task 4:

For this task I wrote the following code in parallel:

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <omp.h>
#define NUM_THREADS 8

void e_val_par(int n, int part, double lambdas[part], double A[n][n], double x[n], double lambda,
    double tol, int max_iter);
double shifted_inv_pm(int n, double A[n][n], double x[n], double lambda, double tol, int max_iter);
double inv_pm(int n, double A[n][n], double x[n], double lambda, double tol, int max_iter);
double power_method(int n, double A[n][n], double x[n], double lambda, double tol, int max_iter);
double vector_norm(int n, double x[n]);
double dot_product(int n, double x[n], double y[n]);
void vecScalar(int n, double a, double x[n], double y[n]);
void jacobi_iter(int n, double A[n][n], double x[n], double b[n], double tol, int max_iter);
void matrix_vector_mult(int n, double A[n][n], double x[n], double y[n]);

void e_val_par(int n, int part, double lambdas[part + 2], double A[n][n], double x[n], double lambda,
    double tol, int max_iter)
{
    double ones[n], reset[n][n];
    reset[n][n] = A[n][n];
    for (int i = 0; i < n; i++)
    {
        ones[i] = 1.0;
    }
    double lambda_max = power_method(n, A, x, lambda, tol, max_iter);
    x[n] = ones[n];
    double lambda_min = inv_pm(n, A, x, lambda, tol, max_iter);
    x[n] = ones[n];
    lambdas[0] = lambda_max;
    lambdas[part + 1] = lambda_min;
    #pragma omp parallel
    {
        double partition = (lambda_max - lambda_min) / part;
        int id, nthreads;
        id = omp_get_thread_num();
```

```
        nthreads = omp_get_num_threads();
        if (id > part)
        {
            lambda = lambda;
        } else if (id ==0) {
            lambda = lambda;
        } else {
            lambda = lambda_min + partition * id;
            lambdas[id] = shifted_inv_pm(n, A, x, lambda, tol, max_iter);
            x[n] = ones[n];
            A[n][n] = reset[n][n];
        }
    }
}
```

I also wrote the following code to test it:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <omp.h>

void e_val_par(int n, int part, double lambdas[part], double A[n][n], double x[n], double lambda,
    double tol, int max_iter);
double shifted_inv_pm(int n, double A[n][n], double x[n], double lambda, double tol, int max_iter);
double inv_pm(int n, double A[n][n], double x[n], double lambda, double tol, int max_iter);
double power_method(int n, double A[n][n], double x[n], double lambda, double tol, int max_iter);
double vector_norm(int n, double x[n]);
double dot_product(int n, double x[n], double y[n]);
void vecScalar(int n, double a, double x[n], double y[n]);
void jacobi_iter(int n, double A[n][n], double x[n], double b[n], double tol, int max_iter);
void matrix_vector_mult(int n, double A[n][n], double x[n], double y[n]);

void main()
{
    int n = 3;
    double A[n][n], x[n];
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (i == j)
            {
                A[i][j] = i + 4.0;
            }
            else
            {
                A[i][j] = 0.1;
            }
        }
        x[i] = 1;
    }
    int max_iter = 1000;
    double tol = 1e-6;
    double lambda = 0.0;
    double max, min;
    int part = n - 1;
    double lambdas[part + 2];
```

```
    e_val_par(n, part, lambdas, A, x, lambda, tol, max_iter);
    printf("lambda max: %f", lambdas[0]);
    for (int i = 1; i < part + 1; i++)
    {
        if (lambdas[i] > lambdas[part + 1] && lambdas[i] < lambdas[0])
        {
            printf("\nlambda: %f", lambdas[i]);
        } else {
            printf("");
        }
    }
    printf("\nlambda min: %f", lambdas[part + 1]);
}
```

Here is a screenshot of me compiling and running the code:

```
jdsoc@Jakes-Laptop ~
$ cd /cygdrive/c/Users/jdsoc/Documents/math4610/Linear_Algebra_Code/C\ Code/

jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/Documents/math4610/Linear_Algebra_Code/C Code
$ gcc -o task94 ts9.c shifted_inv_pm.c inv_pm.c power_method.c jacobi_iter.c vecScalar.c dot.c norm.c mv_mult.c shifted_inv_pm_par.c -fopenmp

jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/Documents/math4610/Linear_Algebra_Code/C Code
$ ./task94.exe
lambda max: 6.093022
lambda: 4.998139
lambda min: 3.986161
jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/Documents/math4610/Linear_Algebra_Code/C Code
$
```

Which is the same correct output as above!

Task 5:

To start I wrote the following code for the Gauss-Seidel method:

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

void gauss_seidel(int n, double A[n][n], double x[n], double b[n], double tol, int max_iter);

void gauss_seidel(int n, double A[n][n], double x[n], double b[n], double tol, int max_iter)
{
    int iter = 0;
    double error = 10.0 * tol;
    double c[n], y[n], r[n];
    while (error > tol && iter < max_iter)
    {
        for (int i = 0; i < n; i++)
        {
            y[i] = b[i];
            for (int j = 0; j < n; j++)
            {
                if (j != i)
                {
                    y[i] -= A[i][j] * x[j];
                }
            }
            y[i] /= A[i][i];
        }
        error = 0.0;
        for (int i = 0; i < n; i++)
```

```
        {
            error += (y[i] - x[i]) * (y[i] - x[i]);
        }
        error = sqrt(error);
        if (error < tol)
        {
            break;
        }
        for (int i = 0; i < n; i++)
        {
            x[i] = y[i];
        }
        iter++;
    }
}
```

Before I put this code into the Shifted Inverse Method, I wanted to test to make sure it works so I wrote this:

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

void gauss_seidel(int n, double A[n][n], double x[n], double b[n], double tol, int max_iter);

void main()
{
    int n = 3;
    double A[n][n], x[n], b[n];
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (i == j)
            {
                A[i][j] = i + 4.0;
            }
            else
            {
                A[i][j] = 0.1;
            }
        }
        x[i] = 1;
        b[i] = 1;
    }
    int max_iter = 1000;
    double tol = 1e-6;
    gauss_seidel(n, A, x, b, tol, max_iter);
    printf("x = %f, %f, %f", x[0], x[1], x[2]);
}
```

Which when compiled and ran spit out this:

```
jdsoc@Jakes-Laptop ~
$ cd /cygdrive/c/Users/jdsoc/Documents/math4610/Linear_Algebra_Code/C\ Code/

jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/Documents/math4610/Linear_Algebra_Code/C Code
$ gcc -o task95a gauss_seidel.c ts9.c

jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/Documents/math4610/Linear_Algebra_Code/C Code
$ ./task95a.exe
x = 0.241214, 0.191987, 0.159447
jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/Documents/math4610/Linear_Algebra_Code/C Code
$
```

Which when checked is the correct solution! Now that I know it works I wrote the following code, changing task 2 so that everywhere we used Jacobi iteration now we use GS:

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

double shifted_inv_pm_GS(int n, double A[n][n], double x[n], double lambda, double tol, int max_iter);
double vector_norm(int n, double x[n]);
double dot_product(int n, double x[n], double y[n]);
void vecScalar(int n, double a, double x[n], double y[n]);
void gauss_seidel(int n, double A[n][n], double x[n], double b[n], double tol, int max_iter);
void matrix_vector_mult(int n, double A[n][n], double x[n], double y[n]);

double shifted_inv_pm_GS(int n, double A[n][n], double x[n], double lambda, double tol, int max_iter)
{
    double error = 10.0 * tol;
    double y[n], w[n], z[n];
    int iter = 0;
    double shift = lambda;
    for (int i = 0; i < n; i++)
    {
        A[i][i] = A[i][i] - lambda;
    }
    gauss_seidel(n, A, y, x, tol, max_iter);
    while (error > tol && iter < max_iter)
    {
        vecScalar(n, 1 / vector_norm(n,y), y, z);
        gauss_seidel(n, A, w, z, tol, max_iter);
        double lambda_new = dot_product(n, z, w);
        error = fabs(lambda_new - lambda);
        lambda = lambda_new;
        vecScalar(n, 1 / vector_norm(n, w), w, y);
        iter++;
    }
    return 1 / lambda + shift;
}
```

As well as the following code to test it:

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

double shifted_inv_pm_GS(int n, double A[n][n], double x[n], double lambda, double tol, int max_iter);
double vector_norm(int n, double x[n]);
```

```c
double dot_product(int n, double x[n], double y[n]);
void vecScalar(int n, double a, double x[n], double y[n]);
void gauss_seidel(int n, double A[n][n], double x[n], double b[n], double tol, int max_iter);
void matrix_vector_mult(int n, double A[n][n], double x[n], double y[n]);

void main()
{
  int n = 3;
    double A[n][n], x[n];
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (i == j)
            {
                A[i][j] = i + 4.0;
            }
            else
            {
                A[i][j] = 0.1;
            }
        }
        x[i] = 1;
    }
    int max_iter = 100;
    double tol = 1e-6;
    double lambda = 4.8;
    double lambda_new = shifted_inv_pm_GS(n, A, x, lambda, tol, max_iter);
    printf("lambda = %f", lambda_new);
}
```

Here is a screenshot of me compiling and running the code:

```
jdsoc@Jakes-Laptop ~
$ cd /cygdrive/c/Users/jdsoc/Documents/math4610/Linear_Algebra_Code/C\ Code/

jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/Documents/math4610/Linear_Algebra_Code/C Code
$ gcc -o task95b shifted_inv_pm_GS.c gauss_seidel.c ts9.c vecScalar.c dot.c norm.c mv_mult.c

jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/Documents/math4610/Linear_Algebra_Code/C Code
$ ./task95b.exe
lambda = 4.998058
jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/Documents/math4610/Linear_Algebra_Code/C Code
$
```

Which is the same as the output from task 2 so again we were able to calculate the correct eigenvalue wanted! If I were to rewrite the code from Task 4 using Gauss Seidel method meaning I would just switch the shifted inverse function that I called, I don't see any issue running it in parallel.