University of Antwerp

Scientific Programming

Systems of Linear Equations
Exercise 1

Armin Halilovic - s0122210

November 6, 2015

Contents

1 Problem

2 Using the code

All of the C++ code can be found in the "main.cpp" file and in appendix A of this document. main.cpp comes accompanied by createImages.sh, which contains all of the neccessary UNIX commands to generate the graph images. This file relies on the graph program in the GNU plotutils package to plot graphs, so make sure that it is installed.

To compile and run the program, execute the following commands in the build/ directory:

```
cmake ..
make
chmod +x ./createImages.sh
./data_fitting
```

All of the graphs should be present in the build/images/ directory. If they are not there, make createImages.sh executable with "chmod +x" and run it to create them.

3 Solutions

3.1 Polynomial through non equidistant points

3.2 Natural cubic spline

3.3 Cubic spline with modified conditions

4 Conclusion

After conducting the interpolations, we can conclude that using a cubic spline is the best way to approximate the Runge function, if we had to choose between polynomial and spline interpolation. This conforms to what we learned during the lecture on data fitting.

Appendices

A main.cpp

```
#include <stdio.h>
#include <iostream>
#include <gsl/gsl_linalg.h>
#include <gsl/gsl_pow_int.h>

/*
 * Conditioning: kappa(A)
 * LU decomposition / maybe show multipliers
 * Gaussian elimination with and without pivoting
 * Residual/Error vectors
 * Theorem 14.1
 */

int main (void) {
    int size_i = 3, size_j = 3;
    double a[ size_i ][ size_j ] = {{3.021,  2.174,  6.913},
                                    {1.031, -4.273, 1.121},
                                    {5.084, -5.832, 9.155}};
    double y[ size_j ] = {12.648,
                          -2.121,
                          8.407};

    // A*X = Y
    gsl_matrix *A = gsl_matrix_alloc( size_i , size_j );
    gsl_vector *Y = gsl_vector_alloc( size_j );
    gsl_vector *X = gsl_vector_alloc( size_j );
    gsl_vector *r = gsl_vector_alloc ( size_j );

    for (int i = 0; i < size_i; i++)
        for (int j = 0; j < size_j; j++) {
            gsl_vector_set (Y, j, y[j]);
            gsl_matrix_set (A, i, j, a[i][j]);
        }

    gsl_matrix *LU = gsl_matrix_alloc( size_i , size_j );
    gsl_matrix_memcpy(LU, A);

    //The algorithm used in the decomposition is Gaussian Elimination with partial pivoting
    gsl_permutation *p = gsl_permutation_alloc( size_i );
    int pInt = static_cast<int>(gsl_pow_int(-1, size_i));
    gsl_linalg_LU_decomp(LU, p, &pInt);
    gsl_linalg_LU_solve (LU, p, Y, X);

    gsl_linalg_LU_refine (A, LU, p, Y, X, r);

    FILE *output = fopen("output.txt", "w");
    gsl_matrix_fprintf (output, A, "%g");
    fprintf (output,"======\n");
    gsl_matrix_fprintf (output, LU, "%g");
    fprintf (output,"======\n");
    gsl_vector_fprintf (output, X, "%g");;
    fprintf (output,"======\n");
    gsl_vector_fprintf (output, r, "%g");
```

```
    gsl_matrix_set (A, 1, 1, -4.275);
    gsl_matrix_memcpy(LU, A);
    gsl_linalg_LU_decomp(LU, p, &pInt);
    gsl_linalg_LU_solve (LU, p, Y, X);
    gsl_linalg_LU_refine (A, LU, p, Y, X, r);

    fprintf (output,"======\n\n\n");
    gsl_matrix_fprintf (output, A, "%g");
    fprintf (output,"======\n");
    gsl_matrix_fprintf (output, LU, "%g");
    fprintf (output,"======\n");
    gsl_vector_fprintf (output, X, "%g");
    fprintf (output,"======\n");
    gsl_vector_fprintf (output, r, "%g");

    fcloseall ();
    gsl_matrix_free (A);
    gsl_matrix_free (LU);
    gsl_vector_free (X);
    gsl_vector_free (Y);
    gsl_permutation_free(p);
    return 0;
}
//gsl_matrix_swap_rows();
```