Mascherpa Audric
Mulac Nino
Vermot-Desroches Matthias

2023-2024
ROBO4

LAB03 Report

I. Lab Questions

1) Explain briefly how you can apply the above equation to reduce the size of the matrix to $3N \times 3N$.

If we change the variable X to x-xi, the value of the function at xi is equal to di because all other terms simplify to zero.
So yi = di, it is no more a variable.

2) What must be added to the compile statement to avoid such error messages?

We must include the path leading to where we install the Eigen Library in the command. To do this, we add -I path/to/eigen/ to the command.
Here is the full command :

```
g++ -I /path/to/eigen/ Spline.cpp -o Spline.exe
```

In our case, eigen is in the same folder as the program. So we just put `eigen` in our Makefile. If you installed eigen in `/bin` for example, edit the Makefile accordingly.

3) Explain what is the meaning of the red const key-word in my example above?

The *const* keyword is used to explicitly say to the program that the method get_value won't modify the fields of its object.
Practically, it allows the method to be called on *const* objects.

4) What are the pre-requisites for the vectors xs and ys, if any?

The vectors xs and ys need to be sorted and have a finite size. Also, xs must be bounded between 0 and 1 and initialized at 0. We must define them as vectors that can only contain double value in order to compile the program.

5) Explain why these preprocessor statements must be added in each include file. (include une classe ifdef, ifndef ou pragma)

These statements must be added in each include file to avoid header files from being executed multiple times.
Otherwise, if the file got imported multiple times by programs needing it, the processor wouldn't know which one to assign to the declaration.

Mascherpa Audric
Mulac Nino
Vermot-Desroches Matthias

II. The construction of the matrix M and the vector u

Here's a step-by-step explanation of how the matrix M and vector U are constructed :

Initialization :

`Eigen::MatrixXd m(3 * dim_, 3 * dim_);` creates a square matrix m with a size of 3 times the number of data points named `dim_`.

`Eigen::VectorXd v(3 * dim_);` creates a vector u with a size of 3 times `dim_`.

Zero Filling :

`m.setZero();` sets all elements of matrix m to zero.

`u.setZero();` sets all elements of vector u to zero.

Value at x = 1.0 :

An extra data point is added at x = 1.0 and its corresponding y-value is set to match the first data point (`ds_[0]`). This is done to initialize the first and last value for the calculations that will follow.

Filling U and M :

A loop iterates 'for' through the data points (dim_ - 1 times, as the last point is treated separately).

Inside the loop, the differences in abscissa (dx) between adjacent data points are calculated.

Blocks of values are assigned to m and v using Eigen's block operations. These blocks correspond to specific coefficients in the spline interpolation formula.

`m.block<1, 3>(3 * i, 3 * i) << pow(dx, 3), pow(dx, 2), dx;` This line sets a block of values in the matrix m. The block is a 1x3 submatrix starting at row 3 * i and column 3 * i. It represents the coefficients for the cubic spline interpolation for the current data point. The values are dx^3, dx^2, and dx since these are the coefficient respectively multiplied by a,b, and c in the third degree equation:

$$y_i = dx^3 * a_i + dx^2 * b_i + dx * c_i + d_i$$

`m.block<1, 6>(3 * i + 1, 3 * i) << 3 * pow(dx, 2), 2 * dx, 1, 0, 0, -1;` . This line sets another bloc of values in the matrix m. The bloc is a 1x6 submatrix starting at row 3 * 1 + 1 and column 3 * i. It represents the coefficients for the continuity of the spline between the current and next data points. The values are obtained from the first derivative of the cubic spline:

$$y'_i = 3 * dx^2 * a_i + 2 * dx * b_i + c_i - y'_{i-1} = 3 * dx^2 * a_i + 2 * dx * b_i + c_i - c_{i-1}$$

`m.block<1, 5>(3 * i + 2, 3 * i);` Similar to the previous line, this sets a 1x5 submatrix in m. It represents the coefficients for the continuity of the spline's second derivative between the current and next data points:

$$y''_i = 6 * dx^2 * a_i + 2 * b_i - y''_{i-1} = 6 * dx^2 * a_i + 2 * b_i - b_{i-1}$$

`v.block<1, 1>(3 * i, 0);` This line sets a block in the vector v. The block is a single value at position 3 * i. It represents the difference in y-values (`ds_`) between the next and current data points, which contributes to the right-hand side of the linear system being solved.

Last data point :

After the loop, these following lines together are responsible for setting up the necessary coefficients and values in the matrix M and the vector V for the last data point in the cubic spline interpolation algorithm.

The third line from the end is generated the same way as the other points.

The second line from the end and the last one are used to explain the condition that the derivatives of both sides of the spline are null tangents:

$y'_0 = b = 0$ and $y'_1 = 3 * dx^2 * a_n + 2 * dx * b_n + c_n$

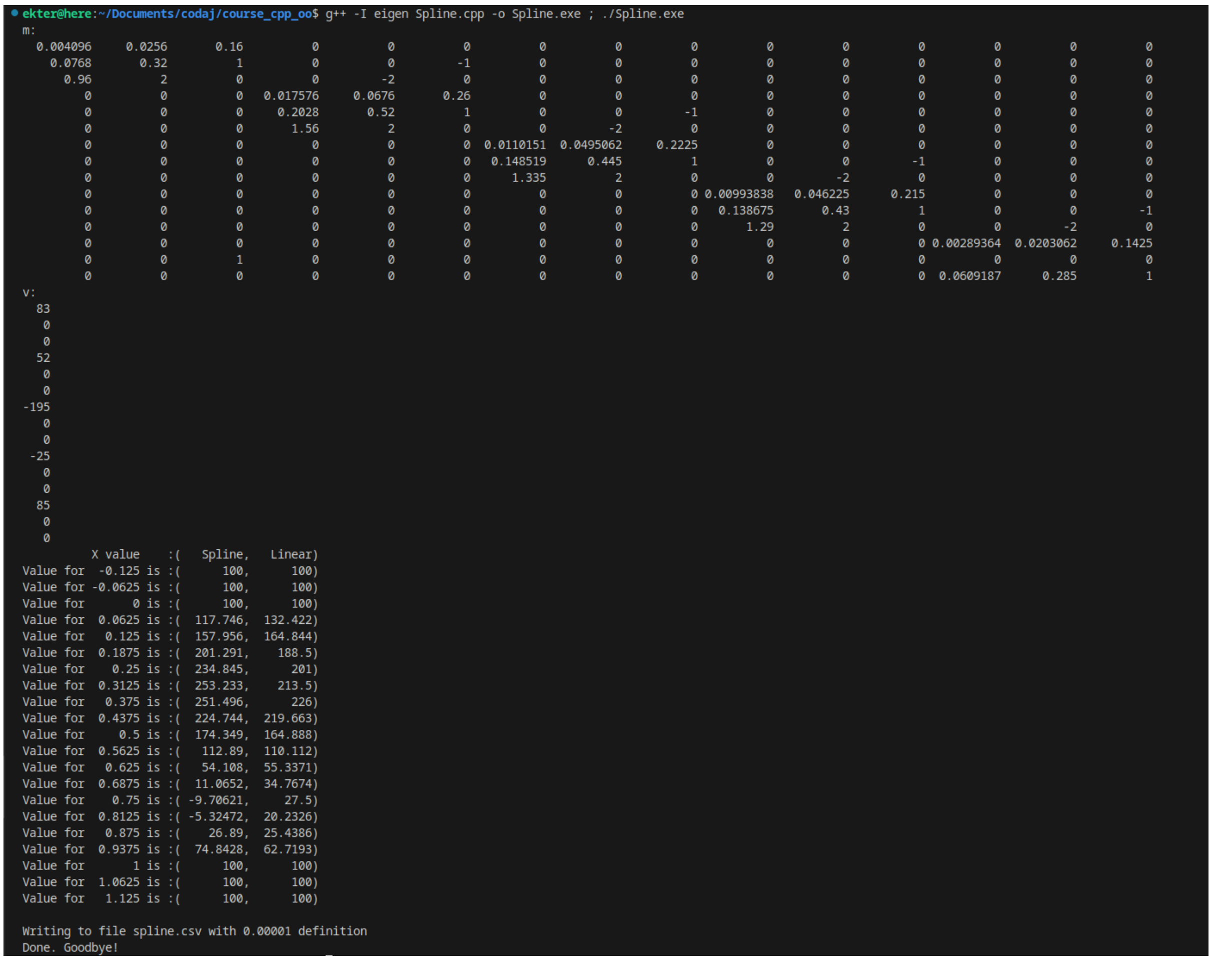
Solving the Linear System M.U = V :

`Eigen::VectorXd u = m.fullPivHouseholderQr().solve(v);` The program solves the linear system M.U = V, where U is the vector of coefficients for the piecewise polynomials.

The solution U is calculated using the fullPivHouseholderQr().solve(v) method, which efficiently solves the linear system.

III. The execution of the program

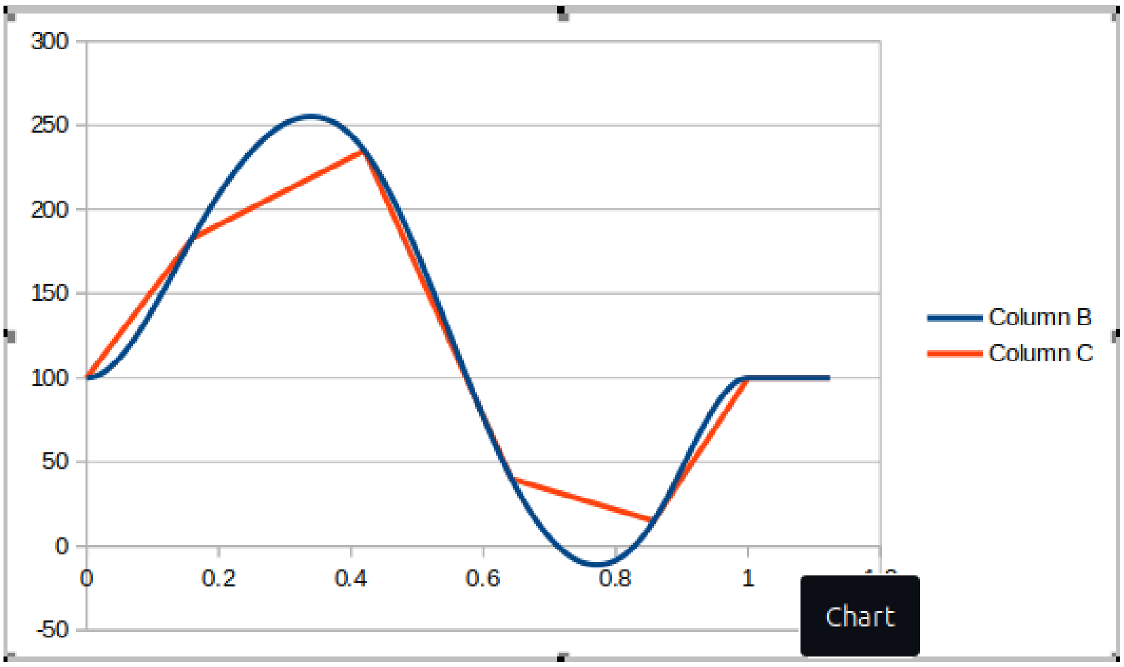
Here is the output of our program :



```
● ekter@here:~/Documents/codaj/course_cpp_oo$ g++ -I eigen Spline.cpp -o Spline.exe ; ./Spline.exe
m:
  0.004096     0.0256       0.16          0          0          0          0          0          0          0          0          0          0          0          0
    0.0768       0.32          1          0          0         -1          0          0          0          0          0          0          0          0          0
      0.96          2          0          0         -2          0          0          0          0          0          0          0          0          0          0
         0          0          0   0.017576     0.0676       0.26          0          0          0          0          0          0          0          0          0
         0          0          0     0.2028       0.52          1          0          0          0         -1          0          0          0          0          0
         0          0          0       1.56          2          0          0         -2          0          0          0          0          0          0          0
         0          0          0          0          0          0  0.0110151  0.0495062     0.2225          0          0          0          0          0          0
         0          0          0          0          0          0   0.148519      0.445          1          0          0         -1          0          0          0
         0          0          0          0          0          0      1.335          2          0          0         -2          0          0          0          0
         0          0          0          0          0          0          0          0          0 0.00993838   0.046225      0.215          0          0          0
         0          0          0          0          0          0          0          0          0   0.138675       0.43          1          0          0         -1
         0          0          0          0          0          0          0          0          0       1.29          2          0          0         -2          0
         0          0          0          0          0          0          0          0          0          0          0          0 0.00289364  0.0203062     0.1425
         0          0          1          0          0          0          0          0          0          0          0          0          0          0          0
         0          0          0          0          0          0          0          0          0          0          0          0  0.0609187      0.285          1
v:
  83
   0
   0
  52
   0
   0
-195
   0
   0
 -25
   0
   0
  85
   0
   0
          X value    :(   Spline,   Linear)
Value for  -0.125 is :(      100,      100)
Value for -0.0625 is :(      100,      100)
Value for       0 is :(      100,      100)
Value for  0.0625 is :(  117.746,  132.422)
Value for   0.125 is :(  157.956,  164.844)
Value for  0.1875 is :(  201.291,    188.5)
Value for    0.25 is :(  234.845,      201)
Value for  0.3125 is :(  253.233,    213.5)
Value for   0.375 is :(  251.496,      226)
Value for  0.4375 is :(  224.744,  219.663)
Value for     0.5 is :(  174.349,  164.888)
Value for  0.5625 is :(   112.89,  110.112)
Value for   0.625 is :(   54.108,  55.3371)
Value for  0.6875 is :(  11.0652,  34.7674)
Value for    0.75 is :( -9.70621,     27.5)
Value for  0.8125 is :( -5.32472,  20.2326)
Value for   0.875 is :(    26.89,  25.4386)
Value for  0.9375 is :(  74.8428,  62.7193)
Value for       1 is :(      100,      100)
Value for  1.0625 is :(      100,      100)
Value for   1.125 is :(      100,      100)

Writing to file spline.csv with 0.00001 definition
Done. Goodbye!
```

The data is written to a CSV file, and it can be imported with a spreadsheet to draw the following curve :



IV. The construction of the Spline class

Both the `Spline` and the `Linear` class are inheritance descendants from the class `FunctionInterpolator`.
The class `FunctionInterpolator` is an abstract class with a public virtual method, `get_value`, and an implemented method, `operator()`, to be able to get the values by a call on the object.
Making this class abstract allowed us to factorize code, since the *operator()* method was present on both classes.

The Spline class has 6 private components : `dim_, as_, bs_, cs_, ds_` and `xs_.` `As_, bs_, cs_` and `ds_` are the vectors of the coefficients of the spline's equation. `Dim_` corresponds to the dimension of the matrix.

In the constructor, we start by filling the matrix M and the vector u to calculate the vector v like we explained in part II.
Then, we extract the ai, bi and ci coefficients from the vector v.

The destructor is set to the default one.

The method `get_value`, implemented from the virtual parent class, takes the abscissa of the point we want to interpolate.
If it is outside the boundaries `[0,1]`, the return value is the value of the first point of the spline.
If the abscissa is correct, the *xs_* vector is iterated over to find the correct spline segment.
Then the variable change is made in order to compute the exact third grade equation.