Project n°5

Interpolation and Integration Methods / Cubic Splines and Surface Interpolation

Group n°2 Team n°1

Manager : fmonjalet
Secretary : tsanchez
Programmers : rrasoldier, ylevif

Abstract : the goal of this project was to compute a basic model to represent the airflow around an airfoil. With that airflow model, it's be possible to obtain the pressure map of the air above and under the wing, in order to approximate the wing's lift (it's ability to keep the plane in the air). This is to be done in two steps : refining the airfoil into a sufficiently smooth curve, and then computing the pressure map using an integration method.

1 Airfoil Refinement Using Cubic Splines

The only airfoil representation available so far is a .dat file describing a cloud of points. So this file has to be converted into python exploitable data.

In order to be able to work on it using the python programming language, the .dat file information[1] has to be converted into a smooth curve. To obtain that curve, the points describing the upper part (named extrados) and the lower part (named intrados) of the airfoil will be interpolated using cubic splines.

1.1 How to Interpolate a Cloud of Points with Cubic Splines

To do this interpolation the best way possible, the method and formulas given by the *Numerical Recipes in C* were used. We assume that a cloud of points (x_i, y_i), $i \in [0, N-1]$ is provided. There are two main parts of pre-calculus to get the interpolation on any $x \in [x_0, x_{N-1}]$ point, given by the equation 1:

$$interp(x) = A(x) * y_i + B(x) * y_{i+1} + C(x) * y_i'' + D(x) * y_{i+1}'' \tag{1}$$

The first task to carry out is to find the second derivatives of the curve at the points corresponding to x_i, $i \in [0, N-1]$. It can be done by solving an equation system. Actually, all the y_i, $i \in [1, N-2]$ are linked by the following equation :

$$\frac{x_i - x_{i-1}}{6} * y_{i-1}'' + \frac{x_{i+1} - x_{i-1}}{3} * y_i'' + \frac{x_{i+1} - x_i}{6} * y_{i+1}'' = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} - \frac{y_i - y_{i-1}}{x_i - x_{i-1}} \tag{2}$$

There are multiple options for the values given to y_0 and y_{N-1}. We have chosen to set them at 0. Once the values were set up, the system was solved. The whole process is summed up by algorithm 1.

Algorithm 1 Pre-calculus of the second derivatives for the cubic spline interpolation

```
1:  function cubic_spline_interpolation(X : abscissaofthepoints, Y : ordinatesofthepoints)
2:      N ← size(X);   Y'' ← null_vector(N)
3:      Mat_coeff ← null_matrix(N, N)
4:      Mat_coeff[0,0] ← 1;   Mat_coeff[N−1, N−1] ← 1
5:      for i = 0 → N − 1 do
6:          Mat_coeff[i, i−1] ← (X[i] − X[i−1])/6
7:          Mat_coeff[i, i] ← (X[i+1] − X[i−1])/3
8:          Mat_coeff[i, i+1] ← (X[i+1] − X[i])/6
9:          Images[i] ← (Y[i+1] − Y[i])/(X[i+1] − X[i]) − (Y[i] − Y[i−1])/(X[i] − X[i−1])
10:     end for
11:     Y'' ← solve_linear_system(Mat_coeff, Images)
12:     return Y''
13: end function
```

Once this system was solved, the values of $A(x)$, $B(x)$, $C(x)$ and $S(x)$ were computed, with the formulas given by the *Numerical Recipes in C*, and $interp(x)$[2] was calculated.

[1] The point cloud positions

[2] The value of the cubic splines interpolation of the cloud of points in x.

The result was right, but one problem remained : the $interp(x)$ evaluation complexity was $\mathcal{O}(N^3)$ (N being the number of points), because the equation system had to be solved at each evaluation.

1.2 Optimisation

Once everything was computed to calculate $interp(x)$, it was noticed that without optimisation, the equation system to find all the second derivatives had to be solved every time $interp(x)$ was evaluated, and a lot of time was lost. To avoid this problem, it was chosen to use functional programming, as shown in algorithm 2.

Algorithm 2 Computation of the cubic spline interpolation function

1: **function** $cubic\_splines\_interpolation\_precalc(X : abscissaofthepoints, Y : ordinatesofthepoints)$
2: $N \leftarrow size(X)$
3: $Y'' \leftarrow cubic\_splines\_interpolation\_precalc(X, Y)$
4: **return** $lambda(x) : cubic\_splines\_interpolation\_calc(X, Y, Y'', x)$
5: **end function**

The cubic_splines_interpolation_precalc function pre computed the values of the second derivatives, and the cubic_splines_interpolation_calc function that was returned only calculated the A, B, C and D factors corresponding to the x given in parameter, in order to return the interpolation value in x. The function returned memorized the values of the second derivatives.

The final complexity of this process was : $\mathcal{O}(N^3)$ to obtain the interpolating function, and $\mathcal{O}(N)$ to evaluate it. It was linear and not constant, because the right value of i had to be found so that $x_i < x \leq x_{i+1}$.

1.3 Computing the Derivative

For the second part of this project, a cubic spline derivative function had to be computed. The process will not be detailed here because it was mainly the same as the computation of the cubic spline function. The only thing that changed was when A, B, C and D were computed, the derivative of these coefficients with respect to x were used.

To sum up, all that had to be done was to compute the formal derivative of A, and all the other coefficients were easily calculated by using it. This algorithm complexity was exactly the same as the previous one: $\mathcal{O}(N^3)$ to obtain the interpolating function, and $\mathcal{O}(N)$ to evaluate it.

1.4 Results

Figure 1 shows the interpolation of the airfoil used with the original points provided. Figure 2 shows the airfoil interpolation and its derivative.

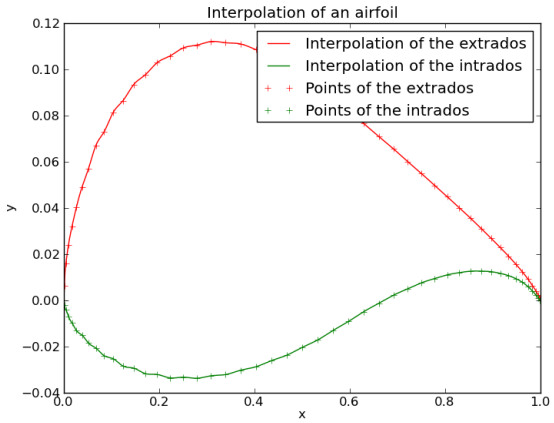

Figure 1: The airfoil interpolation with the original points provided

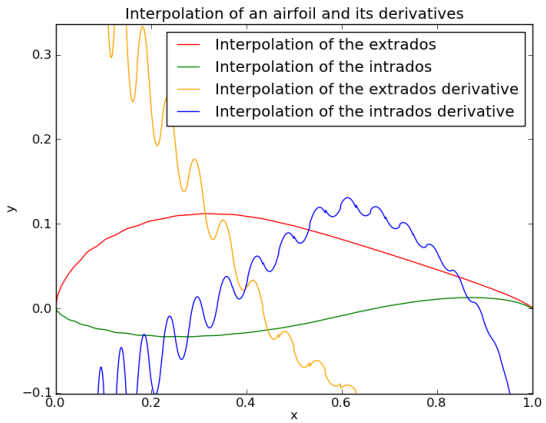

Figure 2: The airfoil interpolation and its derivative

2 Computing the Length of Plane Curves with Various Integration Methods

Once a curve representing the airfoil was obtained, finding a way to compute the airfoil's length was the next step in the pressure map creation. To do that, equation 3 had to be computed with our values.

$$L([0;T]) = \int_0^T \sqrt{1 + f'(x)^2} dx \tag{3}$$

Several integration methods were used and compared.

Every method listed bellow follows the same steps to compute the integral value of a function on an interval and with a given number of intervals :

- the approximated integral value on the first interval was computed.
- the integral on the next interval was computed and added to the previous,
- when the integral on the last interval had been calculated, the process was stopped.

The only thing that changed was the method used to compute the integral value, and with it the convergence speed.

2.1 Left Rectangle Integration Method

The principle was the following:

- the value of $f(x)$ was computed (x was the left bound of the interval),
- the surface of the rectangle having $f(x)$ as height and the interval's size as width was computed.

Finally, the formula used was described by the equation 4, for a function f, into the interval $[a, b]$ subdivided in n intervals.

$$I(f, a, b, n) = h * \sum_{k=0}^{N-2} f(a + kh) \quad \text{with} \quad h = \frac{b-a}{n} \tag{4}$$

2.2 Right Rectangle Integration Method

This method is very similar to the left rectangle one, except that the $f(x)$ value was the right bound of the interval. The formula is given by the equation 5.

$$I(f, a, b, n) = h * \sum_{k=1}^{N-1} f(a + kh) \quad \text{with} \quad h = \frac{b-a}{n} \tag{5}$$

2.3 Middle Rectangle Integration Method

This method is very similar to the left and right rectangle ones except that the $f(x)$ used was the middle of the interval. See equation 6.

$$I(f, a, b, n) = h * \sum_{k=0}^{N-2} f(a + kh + \frac{h}{2}) \quad \text{with} \quad h = \frac{b-a}{n} \tag{6}$$

2.4 Trapezium Integration Method

This integration method was a better alternative to the previous method, even though the only difference was that both images of the left and right bounds were computed to construct a trapezium and to calculate its surface. A good thing to notice is that this method can be deduced from the previous ones using equation 7.

$$I(f,a,b,n) = I_{left\_rect} - \frac{h}{2}(f(a) - f(b)) \quad \text{with} \quad h = \frac{b-a}{n} \tag{7}$$

Processing this way, f was only evaluated once, instead of twice.

2.5 Simpson Integration Method

The problem with the previous methods was that they just gave an average value of the function and did not take the possible variations inside the interval into account.

The Simpson method approximates the function by a polynomial on each interval. This way, the approximation follows the variations of the curve.

Thankfully, the real polynomial did not have to be calculated, and the formula given by equation 8 directly calculates the integral of the second degree polynomial interpolating the curve.

$$I(f,a,b,n) = h * \sum_{k=1}^{N-1} \frac{1/6}{f}(a + (k-1)h) + \frac{2/3}{f}(a + (k-0.5)h) + \frac{1/6}{f}(a + kh) \quad \text{with} \quad h = \frac{b-a}{n} \tag{8}$$

2.6 Convergence Speed Comparison

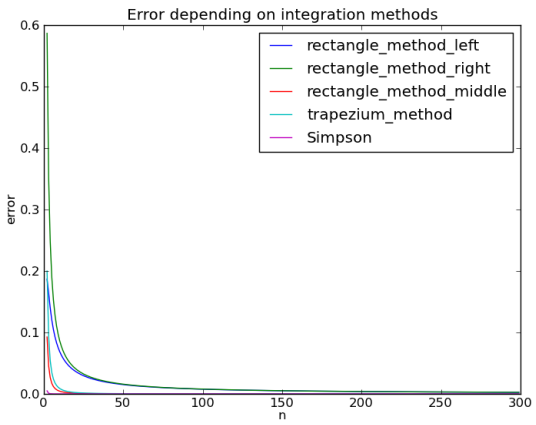

Figure 3: Error of the integration methods with respect to the number of points taken

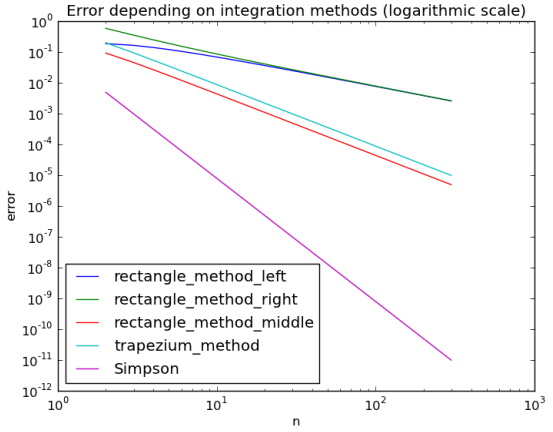

Figure 4: Error of the integration methods with respect to the number of points taken (logarithmic scales)

As it can be seen in figure 3 and even more in figure 4 the Simpson method converges faster than the other ones.

With the logarithmic errors, the convergence speed of the various methods can be determined more precisely. In fact, the slope of the straight line provides the convergence speed, because powers become multiplicative factors in logarithmic scales. What can be seen is that the left and right rectangle method converge with a $\mathcal{O}(n)$ speed (n being the number of iterations). The middle rectangle and trapezium method converge with a $\mathcal{O}(n^2)$ speed, and the Simpson method with a $\mathcal{O}(n^3)$ speed.

On these graphs, the function chosen was representative of the global results found, but for the function $f(x) = \sqrt{1-x^2}$ on the interval $[-1,1]$ all the methods converge with the same speed. We think that it's linked to the fact that this function is not differentiable on -1 and 1, but we did not have the time to develop it.

In every case seen, the Simpson method clearly seemed to be the best.

The complexity of each method is $\mathcal{O}(n)$ in terms of f calls (n being the number of intervals).

3 Computing the Pressure Map

The airfoil length is required to compute the pressure of the air around the airfoil. In order to compute this pressure, the airflow had to be modeled.

3.1 Modeling the Laminar Airflow

Even if the real airflow around an airfoil is not laminar, this model have been chosen to make the computation easier. The laminar airflow around the airfoil can be modeled by using the formula below, each curve obtained represents the path followed by an air particle (see figure 3.2).

Given the airfoil points interpolation function, each curve was computed for the extrados or the intrados with this formula:

$$y = (1 - \lambda)f(x) + \lambda \times 3h \; \forall \lambda \in [0; 1]$$

where f was the result of the interpolation. Each curve corresponds to a value of λ.

3.2 Computing the Pressure Variation Map

The pressure around the airfoil depends on the length of the path followed by the air particles. It is assumed that every air particle takes the same time to go from the beginning to the end of the airfoil, the more curved the path is, the faster the particle will have to travel, and the greater the dynamic pressure will be.

To plot this graph, a lot of airflow paths were traced to reduce the gap between each curve, and each one was colored with a color corresponding to the pressure value on it with an arbitrary formula. The density of the curves and the gradient of color chosen provide a good representation of the pressure map (see figure 6).

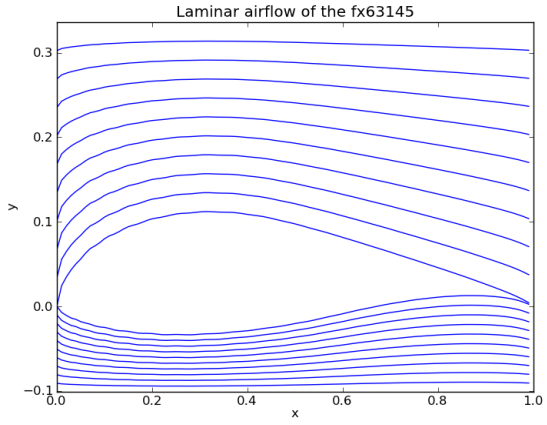

Figure 5: The airflow shape around the airfoil

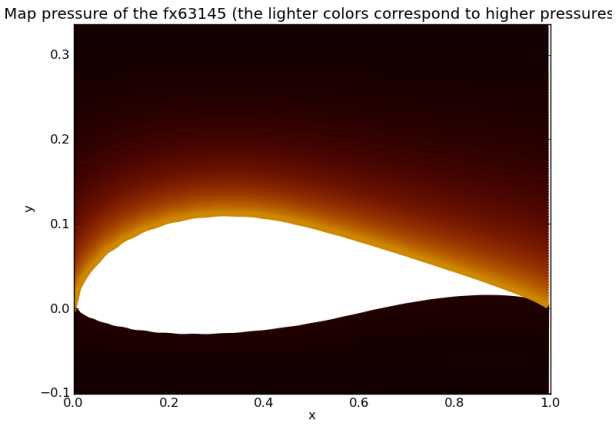

Figure 6: Pressure map around the airfoil (the Simpson method was used)

4 Conclusion

This project was an opportunity to understand the influence of every module involved in a modelling program. In fact, the integral computation used here may seem to be a small part of this project, but seeing the convergence speed comparison results, we realized that it is one of the most important parts in term of performance and precision. We also saw the necessity to have a smooth derivable interpolation of the airfoil to obtain good results, which was possible thanks to cubic splines. Moreover, it's always interesting to apply mathematical concepts to a very concrete purpose.