CAD practical – programming part 2: Splines

Compile the source code "`gnurbsTP2`" by using CMake. Select "**curve-static**" as build target. You will see a first half-circle (which is built using Lagrangian polynomials as seen during the last session) and three sets of points under it. These three sets will be used to draw three identical half-circles but with different spline definitions. The main code is not supposed to be modified (except when notified), as you only need to implement functions in "`api/nspline.cc`".

1. Edit "`api/nspline.cc`" and implement the following functions. The functions computing the first derivatives must give them to `der`, which is a `std::vector<npoint>` inside the class. This attribute will be read later inside the function `P` (that you will implement) to interpolate points.
   - `void nspline::compute_FD(void)`
     This function computes the first derivatives (slopes) using the finite difference. See page 40 of "Course 2" for the theory.
   - `void nspline::P(double u_, npoint& ret) const`
     Implement this function to draw the curve in order to test the previous function. Pay attention that the polynomial coefficients depend on the spline, thus there must be a small computation to find which spline is concerned by the given `u_`. See page 36 of "Course 2" for the theory. Compile and run to test your two functions.
   - `void nspline::compute_cardinal(double c)`
     This function computes the first derivatives using the cardinal formulation. See page 40 of "Course 2" for the theory. Compile and run to test.
   - `void nspline::compute_natural(void)`
     This function computes the first derivatives by solving the linear equations derived from imposing the constraint of continuous second derivatives between splines patches. See page 52 of "Course 2" for the theory.
     The systems of equations (there are four since it is in the homogenous space) can be solve using the library available within "`gnurbsTP3/nutil/linear_algebra.h`" : create the matrix as a `Square_Matrix`, convert it as a `LU_Matrix`, create `Vector` for the right hand member and call `Solve_Linear_System`. Compile and run to test.
2. When the four functions are correct (the three curves previously missing are now correctly drawn), look at the impact of the choice of the parameter u. In order to do so, modify the line `65` in `main.cc` : `curve.u(i)=i*20` (for instance) and compile to see the impact on the curves.
   Compensate this problem by modifying the function implemented. See pages 80 and 84 of "Course 2" for the theory.