▷Project 8. Fundamental Theorem of Surfacess

This final Matlab project asks you to implement the Fundamental Theorem of Surface Theory as a Matlab function. It is a complicated problem and to manage the complexity successfully you will have to organize your work carefully and work slowly and deliberately. When you have completed this project, I think you will have adequate excuse to feel proud of both your programming skill and your comprehension of the mathematics involved.

Review of Notation and Definitions

Before stating the project we need to recall some definitions and notational conventions that will be used. $\mathcal{O}$ will be the rectangle $[a1, b1] \times [a2, b2]$ in $\mathbf{R}^2$. A point in $\mathcal{O}$ will be denoted by p or (t_1, t_2) in a mathematical context or `(t1,t2)` in a Matlab context. We have two 2×2 symmetric matrix-valued functions, g and ℓ defined in $\mathcal{O}$: $g = (g_{ij})$ and $\ell = (\ell_{ij})$. The matrix g should be positive definite, so in particular it is invertible and we denote its inverse by $g^{-1} = (g^{ij})$. By Cramer's Rule:

$$g^{-1} = \frac{1}{\det(g)} \begin{pmatrix} g_{22} & -g_{12} \\ -g_{12} & g_{11} \end{pmatrix},$$

i.e., $g^{11} = g_{22}/\det(g)$, $g^{22} = g_{11}/\det(g)$, and $g^{12} = g^{21} = -g_{12}/\det(g)$, where $\det(g)$ is the determinant of g, given by $\det(g) := g_{11}g_{22} - g_{12}^2$. We also have corresponding positive definite 3×3 symmetric matrices G and G^{-1} defined in $\mathcal{O}$ by:

$$G = \begin{pmatrix} g_{11} & g_{12} & 0 \\ g_{12} & g_{22} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

and hence

$$G^{-1} = \begin{pmatrix} g^{11} & g^{12} & 0 \\ g^{12} & g^{22} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

We also have two 3×3 matrix-valued functions A^1 and A^2 defined in $\mathcal{O}$ by:

$$A^1 = \begin{pmatrix} \frac{1}{2}(g_{11})_{t_1} & \frac{1}{2}(g_{11})_{t_2} & -\ell_{11} \\ (g_{12})_{t_1} - \frac{1}{2}(g_{11})_{t_2} & \frac{1}{2}(g_{22})_{t_1} & -\ell_{12} \\ \ell_{11} & \ell_{12} & 0 \end{pmatrix}$$

$$A^2 = \begin{pmatrix} \frac{1}{2}(g_{11})_{t_2} & (g_{12})_{t_2} - \frac{1}{2}(g_{22})_{t_1} & -\ell_{12} \\ \frac{1}{2}(g_{22})_{t_1} & \frac{1}{2}(g_{22})_{t_2} & -\ell_{22} \\ \ell_{12} & \ell_{22} & 0 \end{pmatrix}$$

and finally, there are two further 3×3 matrix-valued functions in $\mathcal{O}$, $P^k := G^{-1}A^k$.

The Gauss-Codazzi Equations

If g_{ij} and ℓ_{ij} are the coefficients of the First and Second Fundamental Forms of a surface $\mathcal{F} : \mathcal{O} \to \mathbf{R}^3$, then the matrix-valued functions P^1 and P^2 defined in $\mathcal{O}$ as above satisfy the matrix identity

$$P^1_{t_2} - P^2_{t_1} = P^1 P^2 - P^2 P^1$$

called the Gauss-Codazzi Equations.

Statement of the Project

The primary Matlab M-File should be called SurfaceFT.m and should start out:

```
function   F = SurfaceFT(g11,g12,g22,l11,l12,l22,a1,b1,a2,b2,T1Res,T2Res)
```

where $I := \sum_{ij}$ `gij(t1,t2) dti dtj` and $II := \sum_{ij}$ `lij(t1,t2) dti dtj` are quadratic forms in $\mathcal{O}$, and the function $F : \mathcal{O} \to \mathbf{R}^3$ returned is supposed to be the surface having I and II as its First and Second Fundamental Forms. For this surface to exist, we know from the Fundamental Theorem that it is necessary and sufficient that I be positive definite and that the Gauss-Codazzi equations be satisfied.

Of course the heavy lifting of the SurfaceFT will be done by AlgorithmF (i.e., the solution of the Frobenius Problem) which you will apply to integrate the frame equations in order to get the frame field $\mathfrak{f}$—after which you must apply AlgorithmF a second time to get the surface $\mathcal{F}$ from $\mathfrak{f}_1$ and $\mathfrak{f}_2$.

But to carry out the first application of AlgorithmF, you must first compute the matrices $P^1 = G^{-1}A^1$ and $P^2 = G^{-1}A^2$ that define the right hand sides of the two frame equations. Recall that G^{-1} is the inverse of the 3×3 matrix

$$G = \begin{pmatrix} \texttt{g11(t1,t2)} & \texttt{g12(t1,t2)} & 0 \\ \texttt{g12(t1,t2)} & \texttt{g22(t1,t2)} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

and so it can be easily computed from the `gij(t1,t2)` by using Cramer's Rule, while the two 3×3 matrices A^1 and A^2 are given explicitly (see above) in terms of the `g11(t1,t2)` **and their partial derivatives with respect to the** t_i . (Of course, once you have both G^{-1} and A^i, you get P^i as their matrix product.)

In order to be able to compute the A^i, you will first need to define some auxilliary functions. Most of them can probably be subfunctions defined in the same file, though some could be separate M-Files. For example you will want to have a function called `g(t1,t2)` that returns a 2×2 matrix $\begin{pmatrix} \texttt{g11(t1,t2)} & \texttt{g12(t1,t2)} \\ \texttt{g12(t1,t2)} & \texttt{g22(t1,t2)} \end{pmatrix}$ and another called `l(t1,t2)` that returns a 2×2 matrix $\begin{pmatrix} \texttt{l11(t1,t2)} & \texttt{l12(t1,t2)} \\ \texttt{l12(t1,t2)} & \texttt{l22(t1,t2)} \end{pmatrix}$. You will then want to create the functions `G(t1,t2)` and `invG(t2,t2)` that return the 3×3 matrices G and G^{-1}.

There is another complication before you can define the Matlab functions `A1` and `A2` that represent A^1 and A^2. You not only need the functions `gij(t1,t2)` but also their first partial derivatives with respect to the variables `t1` and `t2`. I recommend that along with

the function `g(t1,t2)` you also define two more functions `g_t1(t1,t2)` and `g_t2(t1,t2)` that return 2×2 matrices whose entries are the partial derivatives of the `gij(t1,t2)` with respect to `t1` and `t2` respectively. As usual you can compute these partial derivatives using symmetric differencing—you don't need to do it symbolically.

You should define a Matlab function GaussCodazziCheck that will check whether or not the Gauss-Codazzi Equations are satisfied. Once you have defined the two 3×3 matrix-valued functions P^1 and P^2, it will be easy to write GaussCodazziCheck, since the Gauss-Codazzi equations are just $P^1_{t_2} - P^2_{t_1} = P^1P^2 - P^2P^1$. The idea is to check the identities numerically, matrix element by matrix element, at a sufficiently dense set of points, again using symmetric differencing to compute the derivatives.

Of course, when you are all done you will need some good test cases on which to try out you algorithm. We will discuss this elsewhere.

GOOD LUCK, AND HAVE FUN!