



Horizon 2020
European Union funding
for Research & Innovation

E-Infrastructures H2020-
EINFRA-2015-1

EINFRA-5-2015: Centres of Excellence for computing applications

# EoCoE

Energy oriented Center of Excellence for computing applications

Grant Agreement Number: EINFRA-676629

D5.2 M36

Tokamesh: a software for mesh generation in tokamaks

## Project and Deliverable Information Sheet

| | | |
|---|---|---|
| EoCoE | Project Ref: | EINFRA-676629 |
| | Project Title: | Energy oriented Centre of Excellence |
| | Project Web Site: | http://www.eocoe.eu |
| | Deliverable ID: | D5.2 M36 |
| | Lead Beneficiary: | CEA |
| | Contact: | Edouard Audit |
| | Contact's e-mail: | edouard.audit@cea.fr |
| | Deliverable Nature: | Report |
| | Dissemination Level: | PU* |
| | Contractual Date of Delivery: | M36 30/09/2018 |
| | Actual Date of Delivery: | M36 30/09/2018 |
| | EC Project Officer: | Carlos Morais-Pires |

* - The dissemination level are indicated as follows: PU – Public, CO – Confidential, only for members of the consortium (including the Commission Services) CL – Classified, as referred to in Commission Decision 2991/844/EC.

## Document Control Sheet

| | | |
|---|---|---|
| Document | Title : | D5.2 Tokamesh: a software for mesh generation in Tokamaks |
| | ID : | D5.2 M36 |
| | Available at: | http://www.eocoe.eu |
| | Software tool: | Microsoft Word |
| Authorship | Written by: | Hervé Guillard |
| | Contributors: | J. Lakhlili, A. Loseille, B. Nkonga, A.Ratnani, Ali-aboudou Elarif |
| | Reviewed by: | E. Audit, N. Girard, PEC members |

# Tokamesh : A software for mesh generation in Tokamaks

Hervé Guillard[1] Jalal Lakhlili[2] Adrien Loseille[3] Alexis Loyer[4]
Boniface Nkonga[5] Ahmed Ratnani[6] Ali Elarif[7]

September 4, 2018

[1]herve.guillard@inria.fr
[2]jalal.lakhlili@ipp.mpg.de
[3]adrien.loseille@inria.fr
[4]alexis.loyer@inria.fr
[5]boniface.nkonga@inria.fr
[6]ahmed.ratnani@ipp.mpg.de
[7]ali-aboudou.elarif@inria.fr

# Contents

# Part I

# Introduction

### 0.0.1 Generalities

Small scale turbulence as well as large scale MHD instabilities in Tokamak plasmas are believed to be quasi bi-dimensional : the fluctuations scales in the parallel direction to the magnetic field are typically 3 to 4 orders of magnitude larger than those in the transverse plane. This property is used in many tokamak plasma simulation codes to have a different spatial resolution in the parallel and perpendicular direction. The necessity to consider a sufficiently fine spatial resolution in the transverse direction (that can be of the order of the Larmor radius) make particularly attractive the use of grids/meshes which are adapted to the magnetic flux surfaces in each 2D poloidal plane while the third periodic toroidal direction can be discretized with great accuracy with Fourier series. The shape of the poloidal cross-section of the averaged flux surfaces in the poloidal planes is governed by magnetic equilibrium. These magnetic equilibria are themselves computed by dedicated equilibrium codes. Having as input a prescribed magnetic field equilibrium, the purpose of the WP5.1 task of the EoCoE project is to develop a generic software to construct 2D triangular/quadrangular meshes of the poloidal plane of a tokamak, which will be aligned to magnetic flux-surfaces.

### 0.0.2 Design of the software and contents of the report

Tokamesh has been designed based on minimal assumptions about the user's input. To construct flux surface aligned meshes of the poloidal plane, the minimal inputs that the meshing software must require are the boundaries of the computational domain and the definition of a flux function inside this computational domain. Although, there is a growing interest for tokamak simulation to include the walls into the computational domain, the domains considered in tokamesh are restricted to the vacuum vessel of the Tokamaks or to a part of it. Specifically, edge plasma codes (SOLPS,FBGKI, TOKAM3X, SOLEDGE) as well as GYSELA exclude a central part of the core plasma around the magnetic axis (O-point). The boundary of this central part will be considered as a constant flux surface. We note also that in some machines, (for instance MAST) poloidal coils are located inside the vacuum vessel. In in this case, for the tokamesh software, the computational domain must exclude these parts of the machine. Thus instead of being defined by the walls of the vacuum vessel, the boundaries of the computational domain is rather defined by a (part of) constant flux surface. The definition of these constant flux surface boundaries cannot be automated as it depends on the user's interest.
There are also a priori no reasons for the computational grid where the flux function is computed to contain the boundary of the vacuum vessel or the in-vessel flux surfaces that the user wishes to be a boundary of the computational domain. Therefore in tokamesh, the definition of the vacuum vessel is independent of the definition of the grid where the flux function is defined.
Given these two inputs : a mesh where the magnetic flux function is defined and a definition of the vacuum vessel boundary, the tokamesh software will follow the following steps :

- Pre-processing of the data
- Domain decomposition in logical squares
- Meshing and gluing of the different subdomains.

**Pre-processing of the data** Pre-processing the data have been found necessary in the first stage of this work. This was not foreseen at the beginning of the project but the interactions with potential users has showed that the inputs that they were providing were not appropriate

to generate flux aligned grids. Specifically, the equilibrium codes like EFIT [8] or cedres++ [16] use low-order approximations and coarse grids to solve the Grad-Shafranov equation with the results that the inputs (mesh and flux solution) lack a sufficient quality and resolution to generate "good" flux aligned meshes. Therefore, we have developed in tokamesh, some tools to improve the quality of the data generated by these equilibrium solvers. This is described in section 0.2.1. In addition, the software contains an independent free equilibrium Grad-Shafranov solver working on unstructured meshes that allows to get a $\mathscr{C}^1$ equilibrium solution provided that the user can give the domain boundary conditions and the functional form with respect to the magnetic flux of the pressure and toroidal current (see section 0.2.2).

**Domain decomposition** Once, the data is judged to be of sufficient quality, a fundamental task in tokamesh consists to split the computational domain into physical subdomains that can be mapped onto computational square grids. This is a common task with existing tokamak mesh generator e.g [24]. However, the approach used in tokamesh extends these previous approaches and does not rely on the physical interpretation of the subdomain decomposition. Instead we use the Morse theory for $\mathscr{C}^2$ functions [25] [19] as well as the companion concept of Reeb graph [28] to show rigorously that the computational domain enclosed by a level set of the magnetic flux can be decomposed into subdomains that are either homeomorphic to a disk or a ring. The notion of Reeb graph allows to organize the mappings of these different subdomains and to construct a global mesh of the collection of these subdomains. These constructions are given in section 0.4.2.

**Meshing and gluing of the different subdomains** Up to now, the tasks in tokamesh are independent of the discretization techniques that are used in the simulation codes. From this points however, one must distinguish between the numerical techniques that use unstructured meshes and those that rely on block-structured meshes. We recall that an unstructured mesh is a tessellation of a domain of the plane by simple polygonal shapes in an irregular pattern. In practice, the polygons are very often triangles or quadrangles or a mixture of the two. On the opposite, block-structured meshes relies on the partition of the computational domain into a small number of sub-domains where each sub-domain is discretized by a structured grid. Unstructured meshes can easily handle very complex geometrical shapes but must maintain a list storing neighboring relations between the mesh elements. Parallelism are vectorization are less efficient with these type of meshes than with block-structured ones. On the opposite, the discretization of complex geometrical shape is extremely difficult with block-structured meshes. In tokamaks, since the grids must be flux aligned, if the vacuum vessel boundaries do not coincide with a flux surface, it is not possible to create block-structured meshes that are both aligned with the flux surfaces and that respect the vacuum chamber boundaries. Therefore, following a common practice in this domain, the computational domain is restricted to the interior of a flux surface close to the vacuum chamber boundary.

**Unstructured meshes** Once the domain decomposition has been generated, for the simulation codes using unstructured triangulation, the strategy used in tokamesh consist to generate first a cloud of node lying on the level sets of the magnetic flux function in each subdomain. The generation of these sets of points is done using two different techniques, one of them gives orthogonal grids but will cluster the grid points in regions of high curvature, in contrast the other technique will give a uniform distribution of the nodes along the isolines but the mesh will not be orthogonal. These two techniques are described in section 0.3.2. In practice, the actual

distribution of the nodes can be a weighted average of the two methods. Once the cloud of node has been generated, a second step consists to eliminate all the nodes that are not inside the vacuum vessel and the final step creates the mesh using a constrained and anisotropic Delaunay technique as described in section 0.3.2

**Block-structured meshes** For the construction of block-structured meshes, the first steps of the strategy is almost identical to the previous case except that instead of generating in each subdomain a cloud of nodes, one generates a mapping between the physical domain and a logical square $[0,1] \times [0,1]$. The construction of these mappings uses approximation of the level sets of the flux function by cubic b-splines and construction of the mapping by spline tensor product see section 0.4.4 and 0.4.4. The construction of these mapping uses heavily the fact that each subdomain is homeomorphic to a disk or annulus. Once done, these mapping have to be combined in order to have a global mesh of the computational domain. A global mapping involving only $\mathscr{C}^0$ continuity only requires that the curves defining the boundaries of the subdomains are identical and this imposes some compatibility conditions between the discretization of the boundaries of the subdomain that are described in section 0.4.3. However, in addition to the previous requirement, the construction of a global mapping with $\mathscr{C}^1$ continuity imposes also that the derivatives of the mapping are continuous when passing through a subdomain boundary. This is described in section 0.4.4.

The remaining of this report is as follows : In part I, we give the necessary theoretical concepts used in the algorithms developed in the software. This part is split in 4 sections developing successively the general tools used through all the algorithms, the pre-processing of the data provided by the users, the algorithms for the generation of unstructured grids and finally the ones developed for the creation of block-structured meshes. Part II then presents the installation of the software and how to use it. For the potential users of tokamesh, this last part have to be complemented by the html documentation present in the sphinx folder of the documentation and presenting the individual functions.

# Part II

# Theoretical part

# 0.1 General tools

## 0.1.1 Clough-Tocher Interpolation

The construction of a flux aligned mesh in tokamesh requires the knowledge of the flux function denoted $\psi(\boldsymbol{x})$ in the sequel. For all modern tokamak configurations, this function cannot be specified analytically and is known as the numerical solution of the Grad-Shafranov (GS) equation [30]. Solving this equation requires to define a mesh covering the domain of interest as well as a numerical approximation procedure. Although the methods to solve the GS equations are numerous, many of them, specially when coupled to real-time or near real-time identification methods use low -order numerical methods. In tokamesh, we have decided to use the Cedre++ software [16]. This choice is guided by the fact that Cedre++ using a P1 triangular finite element method is very general and is not limited by the geometry in contrast to other GS solvers relying on more accurate methods. Tokamesh also offers an interface with the output of the EFIT software [8] that appears to be very often used in the fusion community.
However, the experiments that we have done on several input provided by potential users of tokamesh show that the output produced by these software very often uses a too coarse spatial resolution to produce high quality and smooth outputs. As the construction of flux aligned meshes depends on the smoothness of the iso-contours computed by the GS equilibrium codes, we had to develop some generic tools to improve the quality of the fields provided by the GS solvers. For the sake of generality, since it is always possible to convert a mesh into an equivalent triangular one, these tools have to be effective on unstructured triangular meshes. In a first version of the software, we have used for this task Powell-Sabin finite element but the present version use instead the reduced Clough Tocher or (Heish Clough Tocher) interpolation [5] as this family of finite element is simpler to handle than Powell-Sabin and seems in practice to have the same accuracy. The following section is a brief description of Clough Tocher interpolation. More detailed and accurate description can be found in [2], [1] and for a general presentation of splines defined on triangulation in [21].

#### Definition of Heish Clough Tocher (HCT) finite element

The HCT method is a finite element method that uses as basis function $\mathscr{C}^1$ functions. In the sequel, we use the same notation as in [2].

A reduced HCT finite element is the triple $(K, P_K, \Sigma_K)$ where the data $K$, $P_K$ and $\Sigma_K$ are defined as follows :

- The set $K$ is a triangle with vertices $a_1, a_2, a_3$ and $a$ the barycenter of $K$. Denoting by $K_i$ (cf. Figure 1 ) the triangles with vertices $a, a_{i+1}, a_{i+2}$, $i \in \{1, 2, 3[\equiv 3]\}$.
- The space $P_K$ is given by :

$$P_K = \left\{ P \in C^1; P_{|K_i} \in P_3(K_i), i = 1, 2, 3 \quad \text{and} \quad \frac{\partial P}{\partial n} \in P_1(K_i') \quad \forall \quad K' \subset K \right\}$$

  where $\frac{\partial P}{\partial n}$ is the outer normal derivative of the function $P_{|K_i'}$ and $K_i'$ the side opposite to the vertex $a_i$.
- $\Sigma_K$ is the set of degrees of freedom defined by :

$$\Sigma_K = \{P(a_i), DP(a_i)(a_{i+1} - a_i), DP(a_i)(a_{i+2} - a_i), i = 1, 2, 3[\equiv 3]\}$$

  where $DP(a_i)(a_{i+1} - a_i)$ is the directional derivatives across the side $[a_i a_{i+1}]$.

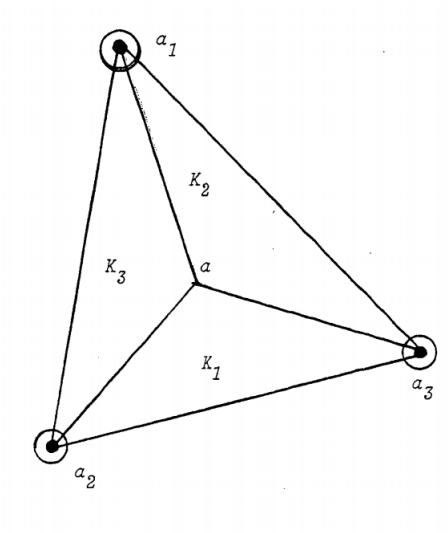



Figure 1: Triangle HCT réduit

Note that in contrast to other finite element methods, the definition of the finite element space can be done on the triangles in the physical space without the use of a reference element and a mapping[2]. This is one of the advantage of this method.

**Passage from local to global degrees of freedom**

Let $v$ a fonction of class $\mathscr{C}^1$ on the domain $\Omega$. Denoting by $DL(v)$ and $DG(v)$ respectively the vector of local and global degrees of freedom, which are defined by :

$$\begin{aligned} DL(v) = [&v(a_1), v(a_2), v(a_3), Dv(a_1)(a_3 - a_1), Dv(a_1)(a_2 - a_1), \\ &Dv(a_2)(a_1 - a_2), Dv(a_2)(a_3 - a_2), Dv(a_3)(a_2 - a_3), Dv(a_3)(a_1 - a_3)] \\ DG(v) = [&v(a_1), v(a_2), v(a_3), \partial_x v(a_1), \partial_y v(a_1), \partial_x v(a_2), \partial_y v(a_2), \partial_x v(a_3), \partial_y v(a_3)] \end{aligned}$$

The directional derivatives and the global derivatives are related by :

$$\begin{bmatrix} Dv(a_i)(a_{i-1} - a_i) \\ Dv(a_i)(a_{i+1} - a_i) \end{bmatrix} = \begin{bmatrix} x_{i-1} - x_i & y_{i-1} - y_i \\ x_{i+1} - x_i & y_{i+1} - y_i \end{bmatrix} \begin{bmatrix} \partial_x v(a_i) \\ \partial_y v(a_i) \end{bmatrix}$$

So that, $DL(v)$ and $DG(v)$ are linked by :

$$DL(v) = DG(v)D$$

where $D$ is a block diagonal matrix of size $9 \times 9$ given by :

$$D = \begin{bmatrix} 1 & & & & & & & & 0 \\ & 1 & & & & & & & \\ & & 1 & & & & & & \\ & & & d_1 & & & & & \\ & & & & & d_2 & & & \\ 0 & & & & & & & & d_3 \end{bmatrix}$$

where we denote $d_i$ the matrix defined by :

$$d_i = \begin{bmatrix} x_{i-1} - x_i & y_{i-1} - y_i \\ x_{i+1} - x_i & y_{i+1} - y_i \end{bmatrix}, \text{for} i \in \{1, 2, 3 \mod 3\}$$

**Expression of basis functions**

The basis functions value can be evaluated with the use of the barycentric coordinates of the points in $K$.
Denoting by $[R_i]$, the vector of 9 basis functions associated with the subtriangle $K_i$ ($i \in \{1, 2, 3 \mod 3\}$). From [2], we have that :

$$[R_i] = [A_i][\lambda] \tag{1}$$

where $[\lambda]$ is the vector of Bernstein polynomials given by :

$$\lambda = \begin{bmatrix} \lambda_i^3 & \lambda_{i+1}^3 & \lambda_{i+2}^3 & \lambda_i^2\lambda_{i+2} & \lambda_i^2\lambda_{i+1} & \lambda_{i+1}^2\lambda_i & \lambda_{i+1}^2\lambda_{i+2} & \lambda_{i+2}^2\lambda_{i+1} & \lambda_{i+2}^2\lambda_i & \lambda_i\lambda_{i+1}\lambda_{i+2} \end{bmatrix}$$

and $[A_i]$ is a $9 \times 10$ matrix defined according to the excentricity parameters [2].

The derivatives of these basis functions are therefore defined by :

$$\begin{cases} [\partial_j R_i] = [A_i][\partial_j \lambda] & j = 1, 2, 3 \\ [\partial_{jk}^2 R_i] = [A_i][\partial_{jk}^2 \lambda] & j, k = 1, 2 \end{cases} \tag{2}$$

with $\partial_j R = \frac{\partial R}{\partial \lambda_j}$ and $\partial_{jk}^2 R = \frac{\partial^2 R}{\partial \lambda_j \partial \lambda_k}$

**HCT interpolation of function and derivatives of interpolate**

Let $(K, P_K, \Sigma_K)$ a given reduced HCT finite element. Given a function $v$ defined over the set $K$, the **HCT-interpolant** $\Pi_K(v)$ of the function $v$ is defined through the relations :

$$\Pi_K(v) \in P_K \quad \text{and} \quad R_i(\Pi_{K_i}(v)) = R_i(v)$$

with $R_i$ the basis functions vector associated to $K_i$.

From this definition, the interpolation of a function $v \in C^1(\Omega)$ on the subtriangle $K_i$ of $K \subset \Omega$ is equivalent to :

$$\Pi_{K_i}(v) = [DL(v)][R_i]$$

with $[R_i]$ and $DL(v)$ defined in (1).

Then, from (2) and this equivalence, the derivatives of the interpolating polynomials $\Pi_{K_i}(v)$ are given by the follows expressions :

$$\begin{cases} \partial_x \Pi_{K_i} v = & DL(v)[A_i][\partial_x \lambda] \\ \partial_y \Pi_{K_i} v = & DL(v)[A_i][\partial_y \lambda] \\ \partial_x x \Pi_{K_i} v = & DL(v)[A_i][\partial_x x \lambda] \\ \partial_x y \Pi_{K_i} v = & DL(v)[A_i][\partial_x y \lambda] \\ \partial_y y \Pi_{K_i} v = & DL(v)[A_i][\partial_y y \lambda] \end{cases} \tag{3}$$

with $DL$, $[A_i]$ and $\lambda$ defined previously.

**Regular family of reduced HCT**
Let $\bar{K}$ a triangle of reference and $\bar{B}$ a compact in $\mathring{\bar{K}}$. Let $(K)$ a family of reduced HCT. We denote by :

$$h = diamK \quad \text{and} \quad \rho = diam(\text{registred cercle in } K)$$

The family $(K)$ is regular if :

1. There is a constant $\alpha > 0$ such that $\alpha \leq \frac{\rho}{h}$ for all element of family $(K)$.
2. Let $F$ denote the affine transformation of the plane that verifies $F(\bar{a}_i) = a_i$, $1 \leq i \leq 3$. Then $\bar{a} = F^{-1}(a)$ is in the compact $\bar{B}$. with $\bar{a}_i$ and $a_i$ respectively the vertices of $\bar{K}$ and $K$ and $\bar{a}$, $a$ some points of $\bar{K}$ and $K$ respectively.

**Definition 0.1.1. Interpolation error**
Given a regular family of reduced HCT triangles, there exists a constant $C$ such that

$$|v - \Pi_K(v)|_{m,K} \leq C h_K^{3-m} |v|_{3,K} \quad \forall v \in H^3(K) \quad \text{and} \quad m = 0, 1, 2 \tag{4}$$

where,

$$|v|_{m,K} = \left( \int_K \sum_{|\alpha|=m} |\partial^\alpha v|^2 \right)^{\frac{1}{2}}$$

and $h_K$ is the diameter of $K$

**Proof :** The proof of this result is given in [4]
The figure 2 illustrates this result. On this figure, the function $v(x, y) = x^4 + y^4$ is interpolated on meshes successively refined and the $L^2$ discrete norm of the interpolation error $|v - \Pi_K(v)|$ versus the mesh parameter is plotted. On can check on this figure that in agreement with (4) the error is of order 3 on the function and 2 on its derivatives.

### 0.1.2 Stream-line integration

Constructing a structured grid on a domain $\Omega$ is equivalent to identify a mapping

$$(s, t) \in [0, 1] \times [0, 1] \to \boldsymbol{x}(s, t) \in \Omega \tag{5}$$

Since we want to construct flux aligned grids, the isolines of the flux function correspond to one of the coordinate lines, say $t =$ for instance. Therefore the streamlines of the ode

$$\begin{cases} \dfrac{d\boldsymbol{x}}{ds} = v(s) \nabla \psi \\ \boldsymbol{x}\,(0) = \boldsymbol{x}_0 \end{cases} \tag{6}$$

where $v(s)$ is any scalar velocity will cross the isolines. In tokamesh, this result is used many times to identify seed points to generate isolines. More precisely, in a connected subdomain corresponding to an edge of the Reeb graph (see section 0.4.2 for these notions), the isolines are closed and unique. One way, to generate a particular isoline $\psi(\boldsymbol{x}) = \psi_0$ is thus to identify a seed point $\boldsymbol{x}_0$ where $\psi(\boldsymbol{x}_0) = \psi_0$ and then to follow the isoline $\psi_0$ by computing the intersection of this isoline with the elements. If the mesh where the function $\psi(\boldsymbol{x})$ is defined, is a $P1$ triangular mesh, this procedure is not sensible to round-off error and the generated isoline is closed and will come back exactly on the seed point $\boldsymbol{x}_0$. To identify the seed point, we will use streamline

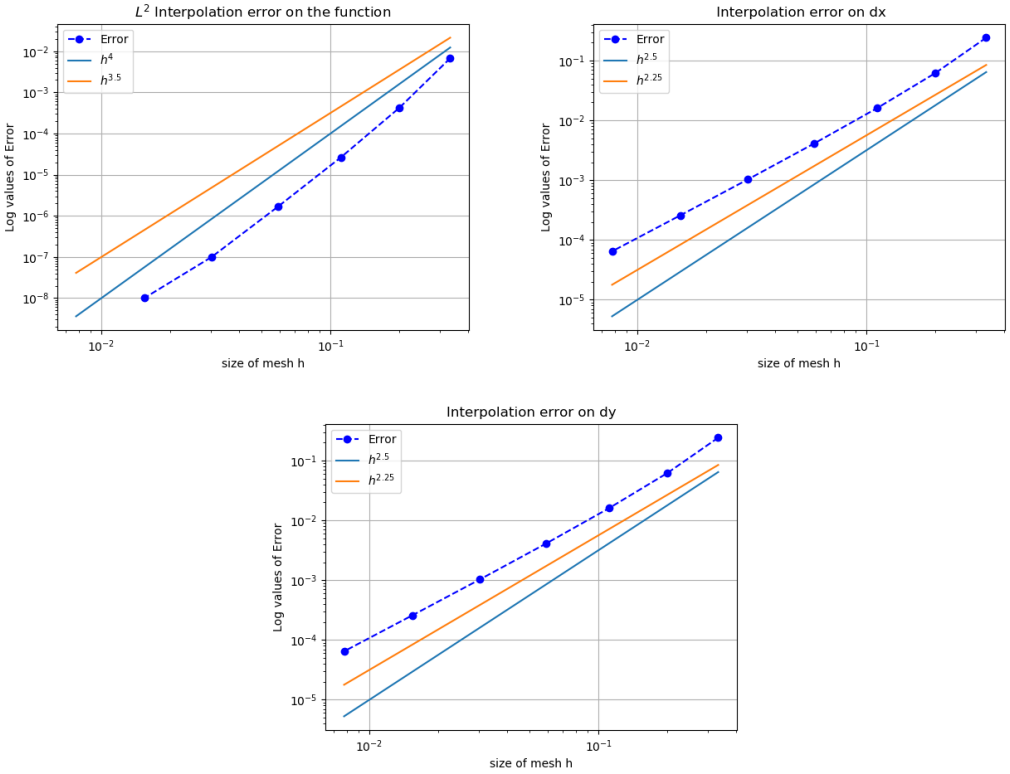



Figure 2: Interpolation for the function $v(x, y) = x^4 + y^4$

integration of the ode (6). More precisely, we used the following result that allow to control accurately the length of the integration step :

**Lemma :** Let $\boldsymbol{x}_0 \in \Omega$, we set $\psi_0 = \psi(\boldsymbol{x}_0)$ then on the curve $\mathscr{S}(s)$ defined by the solution of the ordinary differential system :

$$\begin{cases} \dfrac{d\boldsymbol{x}}{ds} = \dfrac{\psi_M - \psi_0}{\|\nabla\psi\|^2}\nabla\psi \\ \boldsymbol{x}\,(0) = \boldsymbol{x}_0 \end{cases} \tag{7}$$

the function $\Psi(s) = \psi(\boldsymbol{x}(s))$ is linear and $\Psi(s) = \psi_M s + (1-s)\psi_0$.
Proof : We have $d\psi(\boldsymbol{x}(s))/ds = (\psi_M - \psi_0)\nabla\psi.d\boldsymbol{x}(s)/ds = \psi_M - \psi_0 = const$, thus $\psi(s) = \psi_0 + (\psi_M - \psi_0)s$.

This result therefore allows to find easily a point where a specific isoline passes through : Choose a random point in a connected component of $\Omega$ and then integrate the ode (7) until $s = 1$. The solution $\boldsymbol{x}_M = \boldsymbol{x}(s = 1)$ will verify $\psi(\boldsymbol{x}_M) = \psi_M$. This procedure is illustrated on figure 0.1.2 that shows the integration from several points on the same isoline (green curve) toward another isoline (green dashed curve). The blue curves represent an spline interpolation of the curves generated through the integration of (6). All the curves end on the same isoline.
Although any ode solver can used to integrate (6), we have found that this task requires accu-

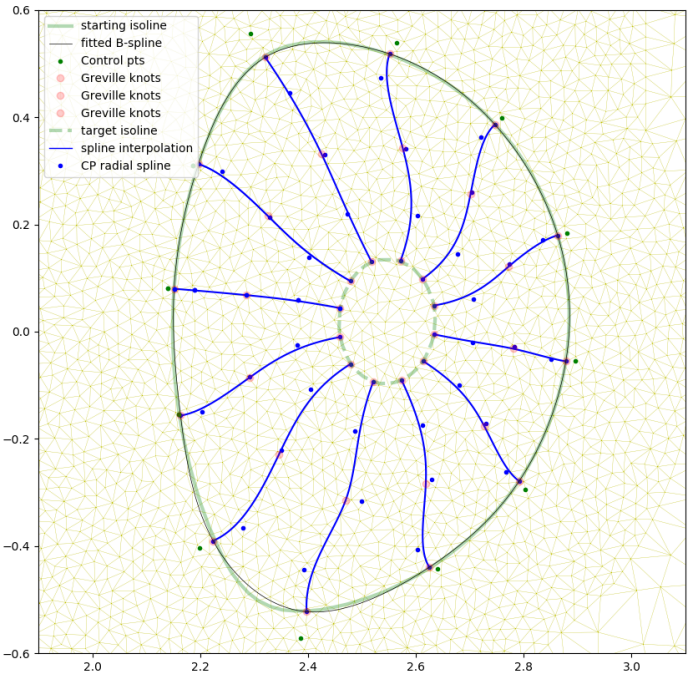



Figure 3: Streamline integration from an isoline

rate solver. In practice, the evaluation of the gradient $\nabla\psi$ is done with the Clough-Tocher $\mathscr{C}^1$ interpolation described in section 0.1.1 and we use dopri5, [15], an explicit Runge-Kutta method of order (4)5 due to Dormand and Prince that use step size control to monitor the accuracy and is available in the scipy library [15]. This procedure is quite costly and can be simplified for many cases. However, it has been found necessary for some difficult cases and it is the default procedure used in tokamesh[1].

Note also that the result (7) allows for a simple construction of a one-to-one mapping from $[0,1]\times[0,1]$ on $\Omega$ such that the curves $t = \ constant$ are isolines of $\psi$ : Let $t \in [0,1]$ and let $\mathscr{C}_0(t)$ be a parametric representation of the isoline $\psi = \psi_0$, then $\boldsymbol{x}(s,t)$ is defined as the solution at $s \in [0,1]$ of the ode 7 with initial condition $\boldsymbol{x}(0) = \mathscr{C}_0(t)$. This mapping is obviously one-to-one since being streamlines of the gradient of $\psi$, for $s \neq s'$, the curves $\mathscr{S}(s)$ and $\mathscr{S}(s')$ cannot cross. Moreover for any $s_0 \in [0,1]$, the curve $\boldsymbol{x}(s_0,t)$ is the isoline $\psi_1 s_0 + (1-s_0)\psi_0$. Such a mapping is orthogonal since isolines and gradient streamlines are orthogonal by definition.

[1]Actually, the use of accurate ode solver is required when one need to reach an isoline close to the magnetic axis. Since in this region, $\nabla\psi$ becomes small, the integration of (6) requires a great accuracy. In practice, for all the other cases the integration of (6) can be done with cheaper methods

While orthogonality is usually a desirable feature for a grid, it can also create extreme clustering of the grid nodes in the regions where the isolines are convex and possess a high curvature. Thus this procedure cannot be used alone but must be mixed with the methods described in section (0.4.4) if the mesh extends in the plasma core. However, the use of this mapping can be an option for edge simulation codes.

### 0.1.3 Spline approximation

**Generalities on splines**

This section recalls some basic properties of B-splines curves and surfaces. We also recall some fundamental algorithms (knot insertion and degree elevation). For a basic introduction to the subject, we refer to the book [20].
A B-Splines family, $(N_i)_{1\leqslant i\leqslant n}$ of order $k$, can be generated using a non-decreasing sequence of knots $T = (t_i)_{1\leqslant i\leqslant n+k}$.

**Definition 0.1.1** (B-Splines series)**.** *The j-th B-Spline of order $k$ is defined by the recurrence relation:*

$$N_j^k = w_j^k N_j^{k-1} + (1 - w_{j+1}^k) N_{j+1}^{k-1}$$

*where,*

$$w_j^k(x) = \frac{x - t_j}{t_{j+k-1} - t_j} \qquad\qquad N_j^1(x) = \chi_{[t_j, t_{j+1}[}(x)$$

*for $k \geq 1$ and $1 \leq j \leq n$.*

We note some important properties of a B-splines basis:

- B-splines are piecewise polynomial of degree $p = k - 1$,
- Compact support; the support of $N_j^k$ is contained in $[t_j, t_{j+k}]$,
- If $x \in\ ]t_j, t_{j+1}[$, then only the *B-splines* $\{N_{j-k+1}^k, \cdots, N_j^k\}$ are non vanishing at $x$,
- Positivity: $\forall j \in \{1, \cdots, n\}\ \ N_j(x) > 0,\ \ \forall x \in ]t_j, t_{j+k}[$,
- Partition of unity : $\sum_{i=1}^n N_i^k(x) = 1, \forall x \in \mathbb{R}$,
- Local linear independence,
- If a knot $t_i$ has a multiplicity $m_i$ then the B-spline is $\mathcal{C}^{(p-m_i)}$ at $t_i$.

The vectorial space spanned by these B-splines, which we denote $\mathcal{S}_k(T, I)$, where $I$ denotes the interval $[t_1, t_{n+1}]$, is called the Schoenberg space.

**Definition 0.1.2** (B-Spline curve)**.** *The B-spline curve in $\mathbb{R}^d$ associated to knot vector $T = (t_i)_{1\leqslant i\leqslant n+k}$ and the control polygon $(\mathbf{P}_i)_{1\leqslant i\leqslant n}$ is defined by :*

$$\mathcal{C}(t) = \sum_{i=1}^{n} N_i^k(t) \boldsymbol{P}_i$$

We have the following properties for a *B-spline* curve:

- If $n = k$, then $\mathcal{C}$ is just a Bézier-curve,

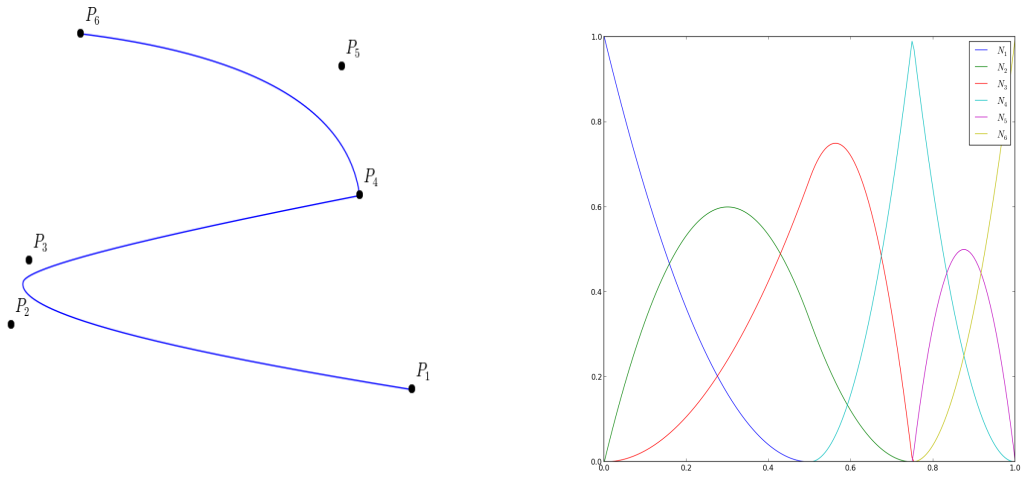



Figure 4: (left) A quadratic B-Spline curve and its control points (the knot vector is $\{000\ \frac{1}{2}\ \frac{3}{4}\frac{3}{4}\ 111\}$), (right) the corresponding B-Splines.

- $\mathcal{C}$ is a piecewise polynomial curve,
- The curve interpolates its extremas if the associated multiplicity of the first and the last knot are maximum (*i.e.* equal to $k$), *i.e.* open knot vector,
- Invariance with respect to affine transformations,
- Strong convex-hull property:

  if $t_i \leq t \leq t_{i+1}$, then $\mathcal{C}(t)$ is inside the convex-hull associated to the control points $\mathbf{P}_{i-p}, \cdots, \mathbf{P}_i$,
- Local modification : moving the $i^{th}$ control point $\mathbf{P}_i$ affects $\mathcal{C}(t)$, only in the interval $[t_i, t_{i+k}]$,
- The control polygon approaches the behavior of the curve.

**Remark 0.1.3.** *In order to model a singular curve, we can use multiple knots or control points, i.e.* $\mathbf{P}_i = \mathbf{P}_{i+1}$.

**Generation of the B-spline functions $N_j^k$**

In order to generate the B-spline functions $N_j^k$, we need to give the appropriate knot vector:

$$T = \{t_1 = t_2 = \cdots = t_k = 0 < \cdots < t_{n+1} = t_{n+2} = \cdots = t_{n+k} = 1\}$$

Classical choices are:

- Equally spaced knots: $t_i = \frac{i-k}{n+1-k}, \quad \forall j \in \{k+1, \ldots, n\}$,
- Average knots: $t_i = \frac{\sum_{i-k}^{i-1} \bar{t}_j}{p}, \quad \forall j \in \{k+1, \ldots, n\}$.

**Choice of the parameterization $\bar{t}_j$**

Usually, three methods are used to find the values of $\bar{t}_j$:

- Equally spaced parametrization: $\bar{t}_j = \frac{j-1}{n-1}, \quad \forall j \in \{1, \ldots, n\}$.
- Chordal parametrization: $\bar{t}_j = \bar{t}_{j-1} + \frac{\|\mathbf{P}_j - \mathbf{P}_{j-1}\|}{d}, \quad \forall j \in \{3, \ldots, n\}$.
  Where we defined:
  $$d = \sum_{j=2}^{n} \|\mathbf{P}_j - \mathbf{P}_{j-1}\|, \quad \text{and} \quad \bar{t}_0 = 0$$
- Centripetal parametrization: $\bar{t}_j = \bar{t}_{j-1} + \frac{\sqrt{\|\mathbf{P}_j - \mathbf{P}_{j-1}\|}}{d}, \quad \forall j \in \{3, \ldots, n\}$.
  Where we defined:
  $$d = \sum_{j=2}^{n} \sqrt{\|\mathbf{P}_j - \mathbf{P}_{j-1}\|}, \quad \text{and} \quad \bar{t}_0 = 0$$

In the case of tokamaks, for the core region where isolines are closed, an alternate parametrization that have been found useful is to parametrize the data point $\mathbf{P}_j$ by the angular value of the segment $\mathbf{OP}_j$ where $\mathbf{O}$ denotes the magnetic axis :

$$t_j = \frac{\text{angle}(\mathbf{OP}_0\mathbf{OP}_j)}{2\pi}$$

where $\mathbf{P}_0$ is an arbitrary point on the curve.

**Fundamental geometric operations**

By inserting new knots into the knot vector, we add new control points without changing the shape of the B-Spline curve. This can be done using the DeBoor algorithm [7]. We can also increase the degree of the B-Spline family and keep unchanged the curve [17]. In (Fig. 5), we apply these algorithms to a quadratic B-Spline curve and we show the position of the new control points.

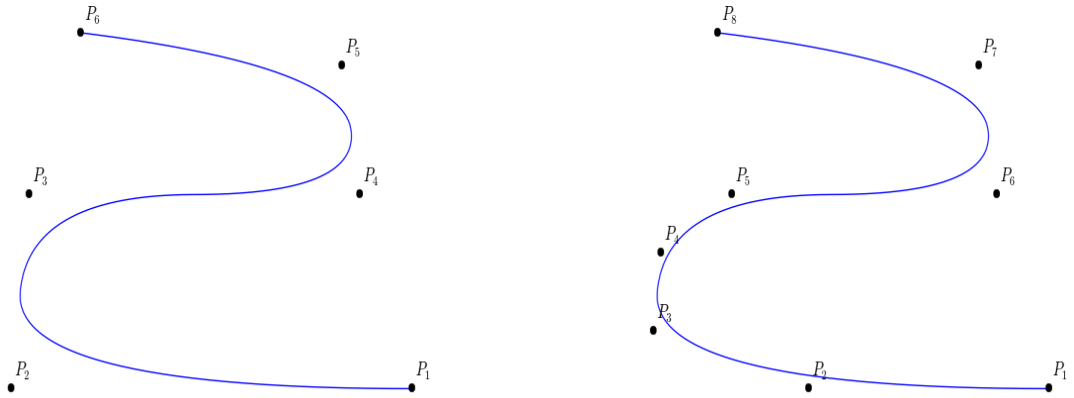
**Deriving a B-spline curve**

The derivative of a B-spline curve is obtained as:

$$\mathcal{C}'(t) = \sum_{i=1}^{n} N_i^{k\prime}(t)\mathbf{P}_i = \sum_{i=1}^{n} \left( \frac{p}{t_{i+p} - t_i} N_i^{k-1}(t)\mathbf{P}_i - \frac{p}{t_{i+1+p} - t_{i+1}} N_{i+1}^{k-1}(t)\mathbf{P}_i \right) = \sum_{i=1}^{n-1} N_i^{k-1\,*}(t)\mathbf{Q}_i \tag{8}$$

where $\mathbf{Q}_i = p\frac{\mathbf{P}_{i+1} - \mathbf{P}_i}{t_{i+1+p} - t_{i+1}}$, and $\{N_i^{k-1\,*}, \ 1 \leq i \leq n-1\}$ are generated using the knot vector $T^*$ which is obtained from $T$ by reducing by one the multiplicity of the first and the last knot (in the case of open knot vector), *i.e.* by removing the first and the last knot.

More generally, by introducing the B-splines family $\{N_i^{k-j\,*}, 1 \leq i \leq n-j\}$ generated by the knot vector $T^{j^*}$ obtained from $T$ by removing the first and the last knot $j$ times, we have the following result:




(a) (b)

Figure 5: (a) A quadratic B-spline curve and its control points. The knot vector is $T = \{000, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 111\}$. (b) The curve after a h-refinement by inserting the knots $\{0.15, 0.35\}$ while the degree is kept equal to 2.

**Proposition 0.1.4.** *The $j^{th}$ derivative of the curve $\mathcal{C}$ is given by*

$$\mathcal{C}^{(j)}(t) = \sum_{i=1}^{n-j} N_i^{k-j^*}(t)\mathbf{P}_i^{(j)}$$

*where, for $j > 0$,*

$$\mathbf{P}_i^{(j)} = \frac{p-j+1}{t_{i+p+1} - t_{i+j}} \left(\mathbf{P}_{i+1}^{(j-1)} - \mathbf{P}_i^{(j-1)}\right)$$

*and $\mathbf{P}_i^{(0)} = \mathbf{P}_i$.*

By denoting $\mathcal{C}'$ and $\mathcal{C}''$ the first and second derivative of the B-spline curve $\mathcal{C}$, it is easy to show that:

**Proposition 0.1.5.** *We have,*

- $\mathcal{C}'(0) = \frac{p}{t_{p+2}} (\mathbf{P}_2 - \mathbf{P}_1)$,
- $\mathcal{C}'(1) = \frac{p}{1-t_n} (\mathbf{P}_n - \mathbf{P}_{n-1})$,
- $\mathcal{C}''(0) = \frac{p(p-1)}{t_{p+2}} \left(\frac{1}{t_{p+2}}\mathbf{P}_1 - \{\frac{1}{t_{p+2}} + \frac{1}{t_{p+3}}\}\mathbf{P}_2 + \frac{1}{t_{p+3}}\mathbf{P}_3\right)$,
- $\mathcal{C}''(1) = \frac{p(p-1)}{1-t_n} \left(\frac{1}{1-t_n}\mathbf{P}_n - \{\frac{1}{1-t_n} + \frac{1}{1-t_{n-1}}\}\mathbf{P}_{n-1} + \frac{1}{1-t_{n-1}}\mathbf{P}_{n-2}\right)$.

**Example:** Let us consider the quadratic B-spline curve associated to the knot vector $T = \{000\ \frac{2}{5}\ \frac{3}{5}\ 111\}$ and the control points $\{P_i, 1 \leq i \leq 5\}$:

$$\mathcal{C}(t) = \sum_{i=1}^{5} N_i^{3'}(t)\mathbf{P}_i$$

we have,

$$\mathcal{C}'(t) = \sum_{i=1}^{4} N_i^{2^*}(t)\mathbf{Q}_i$$

where

$$\mathbf{Q}_1 = 5\{\mathbf{P}_2 - \mathbf{P}_1\}, \quad \mathbf{Q}_2 = \frac{10}{3}\{\mathbf{P}_3 - \mathbf{P}_2\},$$

$$\mathbf{Q}_3 = \frac{10}{3}\{\mathbf{P}_4 - \mathbf{P}_3\}, \quad \mathbf{Q}_4 = 5\{\mathbf{P}_5 - \mathbf{P}_4\}.$$

The *B-splines* $\{N_i^{2^*},\ 1 \leq i \leq 4\}$ are associated to the knot vector $T^* = \{00\ \frac{2}{5}\ \frac{3}{5}\ 11\}$.

**Intersection of two spline curves**

The problem of finding the intersections between two parametric curves is fundamental in Geometric Modeling. In this work, we need to compute the intersection of the wall with a given magnetic surface. Among the available algorithms, we present in the sequel, the one developped by Mørken et al.[26], which is based on knot insertion. The underlying idea is based on the fact that we can approach the *B-splines* coefficients, using the value at some knots.
Let us consider the Greville abscissa, which are successive averages of knots, as

$$\bar{t}_j = \frac{t_{j+1} + \cdots + t_{j+k-1}}{k-1}, \qquad \forall j \in \{1, \cdots, n+1\}. \tag{9}$$

We have the following result

**Proposition 0.1.6.** *For each $S = \sum_{j=1}^{n} s_j N_j^k \in \mathcal{S}_k(T, I) \cap \mathcal{C}^1$ we have,*

$$|S(\bar{t}_j) - s_j| \leqslant C(k)h^2 \|D^2 S\|_{[t_{j+1}, t_{j+k-1}]} \tag{10}$$

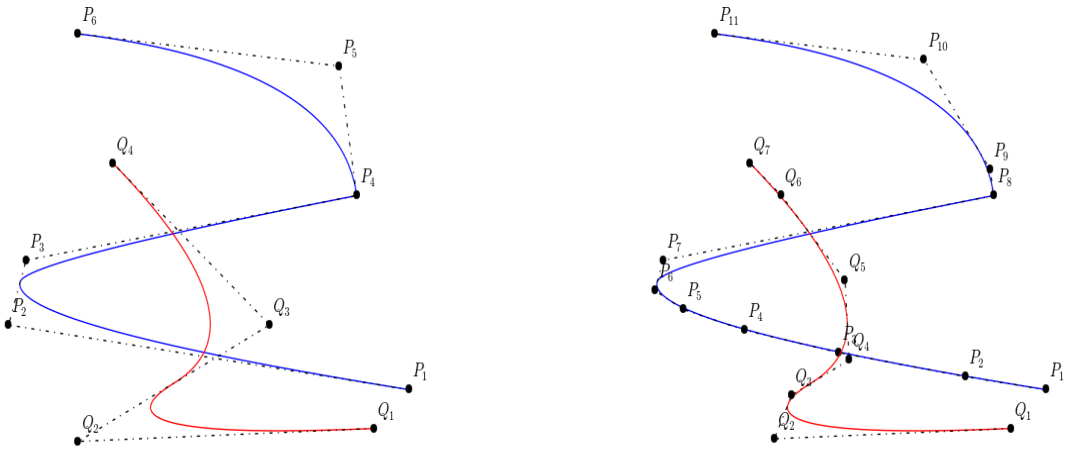
*where $h := \max_{1 \leq i \leq n}(t_{i+1} - t_i)$.*
*Moreoer, if we define $VS = \sum_{i=1}^{n} S(\bar{t}_i)N_i$, we have*

$$\|S - VS\| \lesssim h^2, \quad \forall S \in \mathcal{S}_k(T, I) \cap \mathcal{C}^m,\ \ m \geqslant 2 \tag{11}$$

The simplest version of the Intersection Algorithm may be stated as the following

- Refine the two B-splines curves, for a given mesh resolution,
- Compute the intersection of their control polygons,
- Use the (Eq. 10) to compute knots approximation for all intersections of the two curves,

In (Fig. 6), we show the intersection of two quadratic B-splines curves. By inserting more knots, the intersection of the two control polygons converge to the desired points. Once the intersections points, and their corresponding knots are computed, we can split the two curves with respect to these knots as shown in (Fig. 6(d)).




(a) (b)

Figure 6: (a) Two quadratic B-spline curves and their control polygons. The knot vector for the blue curve is $\{000, \frac{1}{2}, \frac{3}{4}, \frac{3}{4}, 111\}$ while for the red curve it is $\{000, \frac{1}{2}, 111\}$. (b) Curves and their control polygons after inserting the knots $\{0.1, 0.2, 0.3, 0.4, 0.8\}$ in the blue curve and $\{0.4, 0.7, 0.9\}$ in the red curve. Intersections of the two control polygons approximate the the intersection of the two curves.

**Multivariate tensor product splines**

Let us consider $d$ knot vectors $\mathcal{T} = \{T^1, T^2, \cdots, T^d\}$. For simplicity, we consider that these knot vectors are open, which means that $k$ knots on each side are duplicated so that the spline is interpolating on the boundary, and of bounds 0 and 1. In the sequel we will use the notation $I = [0, 1]$. Each knot vector $T^i$, will generate a basis for a Schoenberg space, $\mathcal{S}_{k_i}(T^i, I)$. The tensor product of all these spaces is also a Schoenberg space, namely $\mathcal{S}_{\mathbf{k}}(\mathcal{T})$, where $\mathbf{k} = \{k_1, \cdots, k_d\}$. The cube $\mathcal{P} = I^d = [0, 1]^d$, will be referred to as a patch.
The basis for $\mathcal{S}_{\mathbf{k}}(\mathcal{T})$ is defined by a tensor product :

$$N_{\mathbf{i}}^{\mathbf{k}} := N_{i_1}^{k_1} \otimes N_{i_2}^{k_2} \otimes \cdots \otimes N_{i_d}^{k_d}$$

where, $\mathbf{i} = \{i_1, \cdots, i_d\}$.
A typical cell from $\mathcal{P}$ is a cube of the form : $Q_{\mathbf{i}} = [\xi_{i_1}, \xi_{i_1+1}] \otimes \cdots \otimes [\xi_{i_d}, \xi_{i_d+1}]$.

We are looking for a *B-spline* curve of order $k$ (with a degree $p = k - 1$), such that it interpolates the points $\mathbf{P}_i$ :

$$\sum_{j=1}^{n} N_j^k(\bar{t}_i)\mathbf{c}_j = \mathbf{P}_i, \quad \forall i \in \{1, \ldots, n\} \tag{12}$$

As noticed, in order to solve this problem, we need to know how to generate the B-spline func-

tions $N_j^k$ and the parameters $\bar{t}_i$, then we will have to solve the following linear system:

$$MC = P \tag{13}$$

$$M_{i,j} = N_j^k(\bar{t}_i), \quad C_i = \mathbf{c_i}, \text{ and } \quad P_i = \mathbf{P_i} \tag{14}$$

**Remark 0.1.7.** *Here, we have considered the case of interpolation without end conditions. Additional conditions can be added on the two extremeties of the curve.*

### Approximation of iso-lines by BSplines

Generally in *CAD/CAM* systems, one has to reconstruct smooth surfaces from discrete data. Let us consider a set of $n$ points $\mathbf{V}_i,\ i \in \{1, \cdots, n\}$. In order to have a *well-posed problem*, it is necessary to have an additional information about the *topology* of the points set. This can be done by defining the set of neighbors for each entry point. The approximation problem writes:

*Find a smooth parameterized surface* $\mathbf{F}$ *such that*

$$\text{minimize} \sum_{i=1,n} \text{dist}(\mathbf{F}(\mathbf{u}_i) - \mathbf{V}_i) \tag{15}$$

where $dist$ is a distance function to be defined, as well as the $n$ parameters $\mathbf{u}_i$. Many distance functions can be used. In this work, we are interested in *least square distance*. Moreover, we consider the *parametric distance*, which means that the parameter $\mathbf{u}_i$ has been defined previously. Another approach, is to consider the *geometry distance* which is defined as $\inf_{\mathbf{u}} \|\mathbf{F}(\mathbf{u}) - \mathbf{V}_i\|$.

### Approximation by a B-spline curve

The approximation problem can be formulated as the following:
Given $n$ points $(\mathbf{V}_k)_{1\leq k\leq n}$ in the plane and a parametrization $0 = u_1 \leq u_2 \leq \cdots \leq u_k \leq u_n = 1,\ k \in \{1, \ldots, n\}$, find a B-spline curve $\mathcal{C}(t) = \sum_{j=1}^{n} N_j^k(t)\mathbf{P}_j$ of order $k$ (with a degree $p = k-1$), such that

- it approximates the points $\mathbf{V}_k$, in the least squares sense:

$$\mathcal{F}(\mathbf{c}) = \sum_{k=2}^{n-1} \|\mathcal{C}(u_k) - \mathbf{V}_k\|^2 \to \min \tag{16}$$

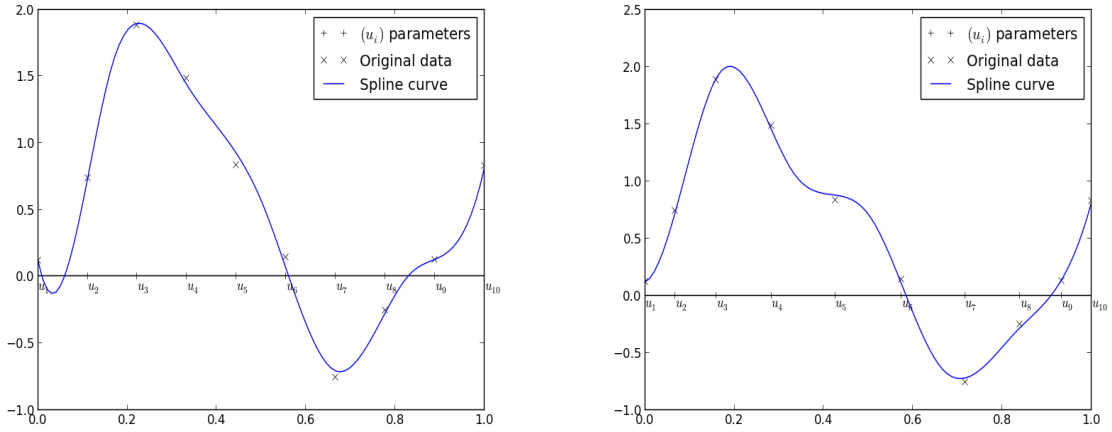
- it interpolates the start and end points (*i.e* $V_1 = \mathcal{C}(0)$ and $V_m = \mathcal{C}(1)$).

We can easily show that the minimum is achieved if and only if

$$D^T D\mathbf{P}^\delta = D^T\mathbf{V}^\delta, \quad \delta \in \{x, y, z\} \tag{17}$$

$$D_{i,j} = N_i^k(\bar{t}_j) \tag{18}$$

Where $\mathbf{V}^\delta$ contains the $\delta \in \{x, y, z\}$ coordinate of the points $\mathbf{V}_k$, $(\bar{t}_j, 1 \leq j \leq n)$ are the Greville knots, while the matrix $D$ will be defined in $2D$ in the next section. More details can be found in [20]. It is interesting to see the impact of the parametrization on the constructed curve (c.f. Fig 7). As we can see, this may lead to high variations of the first derivative. In [27], The authors show how to avoid the parametrization problem but the construction is much more complicated.




(a) (b)

Figure 7: Impact of the parametrization: the discrete data, the approximated curve depending on the parameters $(u_k)$ - (a): using uniform distribution, (b): using Gauss-Legendre points.

**Approximation by a B-spline surface**

Let $(\mathbf{V}_k)_{1\leq k\leq n}$ be a sequence of points in the plane. In this section, we will construct a mapping $\mathbf{F}$, as a B-spline surface, that maps the unit square (*i.e.* patch) onto an approximation of the given set of points.
The *pure approximation* problem can be sketched as follows:

- For each point $(\mathbf{V}_k)_{1\leq k\leq n}$ we associate a parameter (site) $\mathbf{u}_k$. This means also that we have endowed the patch with a specific *topology*. Hence the mapping $\mathbf{F}$ will be such that $\mathbf{F}(\mathbf{u}_k) \simeq \mathbf{V}_k$.
- Given 2 knot vectors, which will define the *B-splines* family that we will use later.

  *Construct the mapping* $\mathbf{F}$ *by minimizing the following functional*:

$$\mathcal{J}_0[\mathbf{F}] = \sum_{k=1}^{n} \|\mathbf{F}(\mathbf{u}_k) - \mathbf{V}_k\|^2 \tag{19}$$

Here again, the unknowns are the B-splines coefficients. In general, the constructed mapping given by (Eq. 19), may be not regular. One needs to add a constraint on the curvature. In this case, the general approximation problem writes:

$$\textit{minimize} \quad \mathcal{J}_0[\mathbf{F}] + \mu\, \mathcal{J}_1[\mathbf{F}] \tag{20}$$

and is called *approximation with fairing*. The choice of the *fairness functional* $\mathcal{J}_1$ must be a compromise between low numerical complexity and high quality of the resulting surfaces [14]. This can be adjusted using the constant $\mu$ which measures the *fairing* weight with respect to fitting. High quality can be achieved by using better approximation of the curvature functionals which can be given by the *exact thin plate energy*. For low numerical complexity, the *simple thin plate*

*energy* can be used

$$\mathcal{J}_1[\mathbf{F}] = \int_{\mathcal{P}} \|\mathbf{F}_{\xi\xi}\|^2 + 2\|\mathbf{F}_{\xi\eta}\|^2 + \|\mathbf{F}_{\eta\eta}\|^2 \, d\mathcal{P} \tag{21}$$

A good compromise between the *simple* and *exact* thin energies is the *data dependent thin plate energy* [13]. More details can be found in [13, 14].

**Example**

Let us consider for the moment the approximation problem using one patch. In order to simplify the text, we will only consider one coordinate ($x$). We will also use $c_{ij}$ instead of $\mathbf{P}^x_{ij}$. The patch coordinates will be denoted by $\boldsymbol{\xi} = (\xi, \eta)$. A parameter $\mathbf{u}$ will the two components $(u_{1,k}, u_{2,k})$. Let us define,

$$\mathcal{F}[\mathbf{F}] = \sum_{k=1}^{n} \|\mathbf{F}(\mathbf{u}_k) - \mathbf{V}_k\|^2 + \mu \mathcal{J}_1[\mathbf{F}] \tag{22}$$

with,

$$\mathbf{F}(\boldsymbol{\xi}) = \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} N_i^{(1)}(\xi) N_j^{(2)}(\eta) c_{ij} \tag{23}$$

and $\mathcal{J}_1$ the *simple thin plate energy*.

**Remark 0.1.8.** *In the general case, we shall write $\mathbf{c}_{ij}$ instead of $c_{ij}$. We will consider the vector representation of the matrix $c$. The corresponding $1D$ index for $(i_b, j_b)$ will be denoted by $b$.*

Simple computations lead to the following expression of $\mathcal{J}_1[\mathbf{F}]$:

$$\mathcal{J}_1[c] = c^T E c \tag{24}$$

where the matrix $E$ can be written as

$$E = A + 2B + C, \tag{25}$$

with the matrices $A, B$ and $C$ are defined as

$$A_{b,b'} = \left( \int_0^1 N''_{i_b} N''_{i_{b'}} \right) \left( \int_0^1 N_{j_b} N_{j_{b'}} \right) \tag{26}$$

$$B_{b,b'} = \left( \int_0^1 N'_{i_b} N'_{i_{b'}} \right) \left( \int_0^1 N'_{j_b} N'_{j_{b'}} \right) \tag{27}$$

$$C_{b,b'} = \left( \int_0^1 N_{i_b} N_{i_{b'}} \right) \left( \int_0^1 N''_{j_b} N''_{j_{b'}} \right) \tag{28}$$

The minimum is achieved if,

$$(D^T D + \mu E) c = D^T x \tag{29}$$

where the matrix $D$ is defined as

$$D_{b,k} = N_{i_b}^{(1)}(u_{1,k}) N_{j_b}^{(2)}(u_{2,k}) \tag{30}$$

the matrix $D^T D$ can be easily computed

$$M_{b,b'} = \left(D^T D\right)_{b,b'} = \sum_{k=1}^{n} N_{i_b}^{(1)}(u_{1,k}) N_{j_b}^{(2)}(u_{2,k}) N_{i_{b'}}^{(1)}(u_{1,k}) N_{j_{b'}}^{(2)}(u_{2,k})$$

which can be seen as a mass matrix with weights equal to one (but not evaluated on quadrature points).

**Global regularity**

For our applications, we need to use multi-patch description in order to take into account the topology of the magnetic field. Hence, we will have to stick these patches together ensuring a $\mathscr{C}^0$ or $\mathscr{C}^1$ regularity. This can be done easily with *B-splines* surfaces (*c.f.* subsection 0.1.3).
As an example, let us consider the case of sticking two patches, namely $\mathcal{P}_m$ and $\mathcal{P}_s$, on the faces $f_m$ and $f_s$ respectively, ensuring the $\mathcal{C}^1$ continuity condition. This can be written as follows:

$$\mathbf{c}_{:,f_m} = \mathbf{c}_{:,f_s} \tag{31}$$

and,

$$\alpha_m \left(\mathbf{c}_{:,f_m} - \mathbf{c}_{:,\delta f_m}\right) = \beta_s \left(\mathbf{c}_{:,f_s} - \mathbf{c}_{:,\delta f_s}\right) \tag{32}$$

where the coefficients $\alpha_m$ and $\beta_s$ are computed using the knot vectors. The first condition enforces the $\mathscr{C}^0$ continuity be requiring that the curves separating the two patches are identical while (32) ensures the $\mathcal{C}^1$ continuity. These conditions can be enforced explicitly or can be added when solving the problem (Eq. 29).

**Spline approximation and singular points**

In section 0.4.4, we will construct $\mathscr{C}^1$ meshes by gluing together spline tensor-product patches. A minimal requirement is therefore that the spline curves are well-behaved and do not contain loops or cusps. Moreover, in the process of gluing together two patches, we will impose that the end derivatives of the curves are colinear. We therefore have to take care that this process do not introduce singular points on the spline curve. Thus, in this section, we consider a cubic Bezier curve defined by the control points $\boldsymbol{x}_k, k = 0, \cdots, 3$ and the parametric expression :

$$\mathscr{C}_k(s) = \sum_{k=0}^{3} \boldsymbol{x}_k N_k(s) \tag{33}$$

the basis functions $N_k(s)$ are displayed in figure 0.1.3. The expression of these functions and of their derivatives are given below :

$$\begin{array}{lll}
N_1^3(s) = (1-s)^3 & (N_1^3)'(s) = -3(1-s)^2 & (N_1^3)''(s) = 6(1-s) \\
N_2^3(s) = 3s(1-s)^2 & (N_2^3)'(s) = 3(1-s)(1-3s) & (N_2^3)''(s) = -6(2-3s) \\
N_3^3(s) = 3s^2(1-s) & (N_3^3)'(s) = 3s(2-3s) & (N_3^3)''(s) = 6(1-3s) \\
N_4^3(s) = s^3 & (N_3^3)'(s) = 3s^2 & (N_3^3)''(s) = 6s
\end{array} \tag{34}$$

Instead of the 4 control points, $\boldsymbol{x}_k$, the curve (33) is usually defined by their end points $\boldsymbol{x}_0$ and $\boldsymbol{x}_3$ and its derivatives $\boldsymbol{d}_0$ and $\boldsymbol{d}_1$. Since we are interested by $G-1$ continuity, the norm

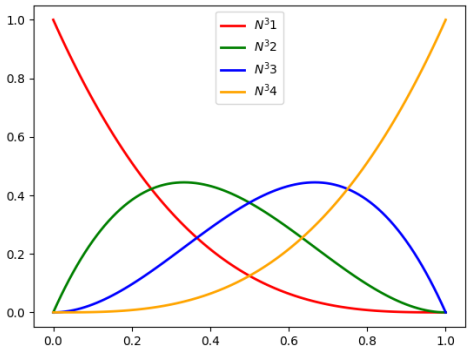



Figure 8: b-spline basis function associated to the knot vector $S = [0, 0, 0, 0, 1, 1, 1, 1]$

of the end derivatives is not relevant and we can assume that $\boldsymbol{d}_0$ and $\boldsymbol{d}_1$ are unit vectors. The control points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are then given by [2]:

$$\boldsymbol{x}_1 = \boldsymbol{x}_0 + \alpha \boldsymbol{d}_0/3 \quad \boldsymbol{x}_2 = \boldsymbol{x}_3 - \alpha \boldsymbol{d}_1/3 \tag{35}$$

where $\alpha$ is a positive parameter whose expression has to be found. The curve then has the following expression :

$$\mathscr{C}_k(s) = \boldsymbol{x}_0 \ (1-s)^3 + 3 \ (\alpha \, \boldsymbol{d}_0 + \boldsymbol{x}_0) \, s \, (1-s)^2 + 3 \ (-\alpha \, \boldsymbol{d}_1 + \boldsymbol{x}_3) \, s^2 \, (1-s) + \boldsymbol{x}_3 \, s^3 \tag{36}$$

Introducing the notation $u = 1 - s$ it will be useful to write it as :

$$\mathscr{C}_k(s) = \boldsymbol{x}_0 + (s^3 + 3us^2)(\boldsymbol{x}_3 - \boldsymbol{x}_0) + \alpha u^2 s \boldsymbol{d}_0 - \alpha u s^2 \boldsymbol{d}_1 \tag{37}$$

Its derivative is given by :

$$\mathscr{C}'_k(s) = \alpha[(u^2 - 2us)\boldsymbol{d}_0 + (s^2 - 2us)\boldsymbol{d}_1] + 6su(\boldsymbol{x}_3 - \boldsymbol{x}_0) \tag{38}$$

while its second derivative is

$$\mathscr{C}''_k(s) = \alpha[(2s - 4u)\boldsymbol{d}_0 + (4s - 2u)\boldsymbol{d}_1] + 6(u - s)(\boldsymbol{x}_3 - \boldsymbol{x}_0) \tag{39}$$

We will examine under what conditions the curve (33) has cusps or loops. Figure 0.1.3 shows a Bezier curve defined by 4 control point with a loop. The red and blue points are the end points of the curve while the green and yellow control the end derivatives. We want to give sufficient conditions on the position of these points in order to avoid loops or cusps on the curve.
The curve $\mathscr{C}_k(s)$ is given by 4 vectors among which only two of them are independent. Our discussion on the existence of singular points for the curve will thus depends on which of them are independent.

[2]For the sake of simplicity, we have considered here the case where the multiplicative coefficients of the derivatives are equal but in practice they can be different if needed

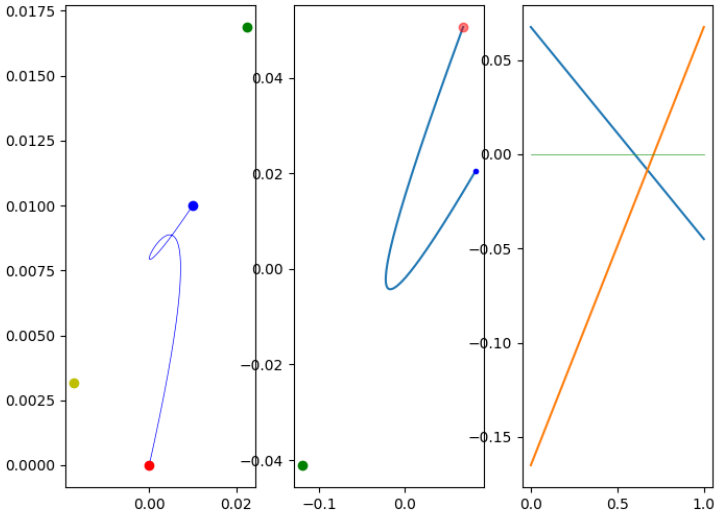



Figure 9: Bezier curve with a loop.Left : curve, middle : derivative, right : the curves 50 for large $\alpha$ showing that 50 have roots in $[0, 1]$.

**The case $\boldsymbol{d}_0 = \boldsymbol{d}_1 = \boldsymbol{d}$** We first examine the case where the two end tangents are identical. In this case we have :

$$\mathscr{C}_k(s) = \boldsymbol{x}_0 + (s^3 + 3us^2)(\boldsymbol{x}_3 - \boldsymbol{x}_0) + \alpha(u^2 s - us^2)\boldsymbol{d} \tag{40}$$

$$\mathscr{C}'_k(s) = \left(-6\,s^2 + 6\,s\right)(\boldsymbol{x}_3 - \boldsymbol{x}_0) + \alpha\left(6\,s^2 - 6\,s + 1\right)\boldsymbol{d} \tag{41}$$

and

$$\mathscr{C}''_k(s) = 6(1 - 2s)((\boldsymbol{x}_3 - \boldsymbol{x}_0) - \alpha\boldsymbol{d}) \tag{42}$$

Let us first study when the curve can have a cusp where $\mathscr{C}'_k(s) = 0$ or a loop. We have two cases:

**$(\boldsymbol{x}_3 - \boldsymbol{x}_0)$ and $\boldsymbol{d}$ are not colinear**

A loop will be possible if there exits two distinct reals $a, b \in ]0, 1[$ such that $\mathscr{C}_k(a) = \mathscr{C}_k(b)$. Since $(\boldsymbol{x}_3 - \boldsymbol{x}_0)$ and $\boldsymbol{d}$ are independent this implies that we must have :

$$\begin{aligned} a^2(3 - 2a) &= b^2(3 - 2b) \\ a(1 - a)(1 - 2a) &= b(1 - b)(1 - 2b) \end{aligned} \tag{43}$$

and this is not possible since in $[0, 1]$, the function $x^2(3 - 2x)$ is monotone. For the same reason the existence of a cusp where $\mathscr{C}'_k(s) = 0$ is not possible.

**$(\boldsymbol{x}_3 - \boldsymbol{x}_0)$ and $\boldsymbol{d}$ are colinear**

In this case, we note $\lambda = |\boldsymbol{x}_3 - \boldsymbol{x}_0|$ and a necessary and sufficient condition for $\mathscr{C}'_k(s) = 0$ is that the equation

$$f(s) = \left(6\,s^2 - 6\,s + 1\right)\alpha - 6\,\lambda\,s^2 + 6\,\lambda\,s = 0 \tag{44}$$

has a solution for $s \in ]0,1[$. But equation (42) that writes now :

$$\mathscr{C}_k''(s) = 6(1-2s)(\lambda-\alpha)\boldsymbol{d} \tag{45}$$

shows that $s = 1/2$ is always an extremum for the quadratic curve $f(s)$. On $s = 1/2$, the value of $f(s)$ on the extremum is

$$f(1/2) = \frac{3\lambda - \alpha}{2}$$

Since in any case $f(0) = f(1) = \alpha$, we can conclude that $f(s)$ is always in $[\min(\alpha, f(1/2)), \max(\alpha, f(1/2)]$. Thus, if $\alpha \leq \lambda$, $f(s) > \alpha$ and is strictly positive while for $\lambda \leq \alpha < 3\lambda$, $f(s) > f(1/2) = \frac{3\lambda-\alpha}{2}$ and is again always positive. Therefore for $\alpha < 3\lambda$, the derivative is always strictly positive and the curve cannot have a cusp. In the same way, it cannot have a loop since Rolle's theorem will imply in this case the existence of a solution in $]0,1[$ of the equation $\mathscr{C}_k'(s) = 0$. For $\alpha > 3\lambda$, the curve admits a cusp.

We summarize these results : if the two end derivatives are colinear, $\boldsymbol{d}_0 = \boldsymbol{d}_1$ for $\alpha \leq 3|\boldsymbol{x}_3 - \boldsymbol{x}_0|$, the curve $\mathscr{C}_k(s)$ has no singular point.

**$\boldsymbol{d}_0$ and $\boldsymbol{d}_1$ are independent** Thus we can develop $(\boldsymbol{x}_3 - \boldsymbol{x}_0)$ on the basis formed by these two vectors :

$$(\boldsymbol{x}_3 - \boldsymbol{x}_0) = \lambda \boldsymbol{d}_0 + \mu \boldsymbol{d}_1 \tag{46}$$

where

$$\lambda = \frac{(\boldsymbol{x}_3 - \boldsymbol{x}_0) \times \boldsymbol{d}_1}{\boldsymbol{d}_0 \times \boldsymbol{d}_1} \qquad \mu = \frac{(\boldsymbol{x}_3 - \boldsymbol{x}_0) \times \boldsymbol{d}_0}{\boldsymbol{d}_1 \times \boldsymbol{d}_0} \tag{47}$$

and the expression of the derivative of the curve becomes :

$$\mathscr{C}_k'(s) = [6su\lambda + \alpha(u^2 - 2us)]\boldsymbol{d}_0 + [6su\mu + \alpha(s^2 - 2us)]\boldsymbol{d}_1 \tag{48}$$

The existence of a cusp or a loop[3] therefore means that it exists $(s,u)$ in $]0,1[\times]0,1[$ such that

$$\begin{aligned} 6su\lambda + \alpha(u^2 - 2us) &= 0 \\ 6su\mu + \alpha(s^2 - 2us) &= 0 \end{aligned} \tag{49}$$

i.e :

$$\begin{aligned} 6s\lambda + \alpha(u - 2s) &= 0 \\ 6u\mu + \alpha(s - 2u) &= 0 \end{aligned} \tag{50}$$

whose solutions are

$$\begin{aligned} s &= \frac{\alpha/\lambda}{3(\alpha/\lambda - 2)} \\ u &= \frac{\alpha/\mu}{3(\alpha/\mu - 2)} \end{aligned} \tag{51}$$

Figure 0.1.3 shows the function $f(x) = \dfrac{x}{3(x-2)}$. It can be seen that the region of the plane $(x,y)$ with $x = \alpha/\lambda, y = \alpha/\mu$ where $s$ or $u$ solutions of (50) do not belong to $]0,1[$ is $x \in [0,3]$ or $y \in [0,3]$.
We therefore have :

[3] for a cusp, we must have $u = 1 - s$ while a loop can correspond to different abscissa

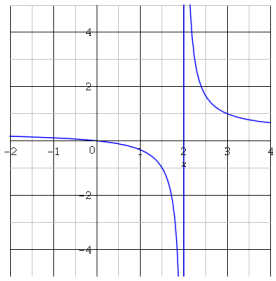



Figure 10: $f(x) = \dfrac{x}{3(x-2)}$

If $\lambda > 0$ and $\alpha < 3\lambda$ then the curve $\mathscr{C}'_k(s)$ has no loop or singular points.

and in a symmetric manner :

If $\mu > 0$ and $\alpha < 3\mu$ then the curve $\mathscr{C}'_k(s)$ has no loop or singular points.
To summarize the results of this section :

| |
|---|
| Let $\lambda > 0$ or $\mu > 0$ be the coordinates of $(\boldsymbol{x}_3 - \boldsymbol{x}_0)$ in the basis $\boldsymbol{d}_0 = \boldsymbol{d}_1$ if these vectors are independent, then if $\alpha < 3\min(\mu, \lambda, \lvert\boldsymbol{x}_3 - \boldsymbol{x}_0\rvert$, the curve (36) has no cusps or loops. |

## 0.2 Pre-processing of the data

The construction of flux aligned meshes depends on the smoothness of the iso-contours computed by the equilibrium codes. In particular, the inputs (mesh and flux solution) must have a sufficient quality and resolution to generate "good" flux aligned meshes. The problem is particularly important near the X point since in many cases, the iso-contours of the flux function in this region are not very regular. Figure 11 shows the level set passing through the x-point as computed by Cedre++. It can be seen that the curve oscillates in this region. This characteristic

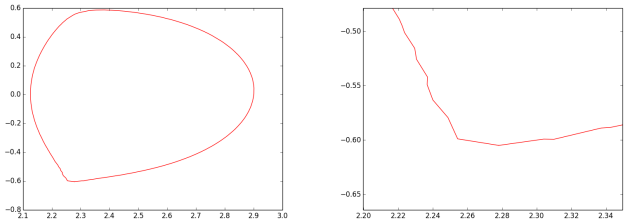


Figure 11: Level set passing through the x-point for GS equilibrium computed by Cedre++

of the solution is not specific to the Cedre++ software. Figure 12 shows the same curve for an equilibrium computed in the jet tokamak with the EFIT software where the same problem can be noticed.

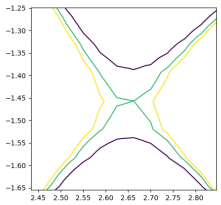


Figure 12: Level set passing through the x-point for GS equilibrium computed by EFIT

### 0.2.1 Data regularization

This absence of smoothness of the equilibrium computed by the equilibrium solvers leads us to develop some tools to regularize the data produced by these codes. The first one is a local refinement tool that selectively allows a mesh refinement in the vicinity of the x-points. Note that a global refinement tool is also available in tokamesh. However, it has been found that an aggressive refinement dividing the spatial step size by factor of order $2^4$ or $2^5$ is necessary to yield smooth contours near the x-point and a global refinement by these factors is too costly in practice and produce too large meshes (of the order of several M points). Figure 13 shows an example of this refinement process for a double disconnect null equilibrium in the West tokamak.

However, this refinement process does not change by itself the data, we have therefore complemented it by a smoothing using cubic interpolation on this refined mesh. More precisely, given the initial mesh where the equilibrium is computed and the flux function defined on this

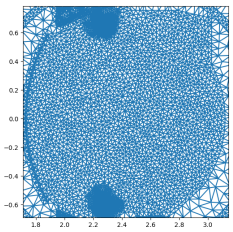


Figure 13: Local refinement near x-points

mesh, we replace this function by its Clough Tocher interpolant (see section 0.1.1) defined by the value of the function on the node and derivatives at the node defined as an average value of the gradient on the triangles surrounding this node. This CT-interpolant is then evaluated on the refined triangulation and use to produce smooth iso-contours. Although there is no theory on this technique, it has been found quite effective to regularize the data in such a way that it can be used for the computation of smooth iso-contours. Figure 14 shows the result of this technique applied to the data of figure 12. Note that in general this technique produces a slight displacement of the x-points. This is usually, barely noticeable except in the case of a connected double null (CDN) where in agreement to the result given [23] (see section 0.4.2), these slight displacement of the x-points produce two disconnected x-points lying on two distinct but extremely close isolines. Thus for this specific case, we had to implement a function to connect again the x-points.

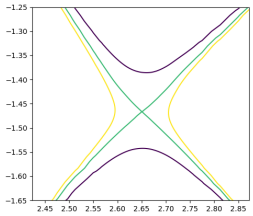


Figure 14: Local refinement near x-points

### 0.2.2 Re-computing a Grad-Shafranov equilibrium

One of the drawback of the previous technique is that it gives a regularization of a GS equilibrium computed with a low-order method. Therefore even if the results are smooth, their accuracy is not greater than the accuracy of the original data. For this reason, it can be interesting to provide in tokamesh a GS solver allowing the use of high-order $\mathscr{C}^1$ method to compute the solution of the equilibrium problem expecting that the use of high order $\mathscr{C}^1$ method will provide solutions that will be both smoother and of greater accuracy than the original input. Note however that it is not guaranteed that the use of high-order elements alone can yield an improved regularity as remarked in [16] : "We are solving here a nonlinear elliptic problem with discontinuous coefficients(in the case of iron-transformer tokamaks) and discontinuous right-hand side. The standard convergence theory for finite elements and elliptic regularity theory

does not yield improved approximation results for polynomials of degree higher than 1".

Nevertheless we have developed a free boundary Grad-Shafranov solver using Clough-Tocher reduced cubic finite element (see section 0.1.1). The interest of this finite element family is that it can use the same mesh that the one where the initial data has been computed. Moreover, since the Grad-Shafranov equation is a non-linear one that must be solved with an iterative procedure, the initial data provides a convenient initial guess for this procedure. Non-homogeneous Dirichlet boundary conditions coming from the equilibrium data computed on the initial mesh are used and the non-linear system is solved by Picard iterations. Note that to take into account non-homogeneous Dirichlet boundary conditions, it has been necessary to develop a penalization method of Nischte type. Figure 15 compare the solution obtained for the computation an equilibrium in the JET tokamak with three different numerical methods : The original equilibrium computed with EFIT, a solution computed with a P1 finite element solver and the one computed with the $\mathscr{C}^1$ Clough-Tocher finite element method. This Grad-Shafranov

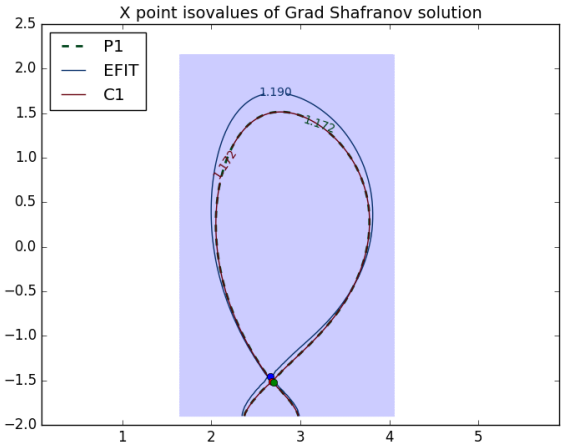



Figure 15: Comparison of magnetic equilibrium computed by EFIT, a P1 method and a $\mathscr{C}^1$ method using Clough-Tocher finite elements

solver can only be used if the user possesses information about the non-linear functions defining the right-hand side of the GS equation, namely the functional form of the pressure and toroidal current with respect to the magnetic flux. In case where these information are missing, the only possibility will be to use the data smoothing procedure described in the previous section 0.2.1.

## 0.3 Unstructured meshes

### 0.3.1 Introduction

The use of unstructured meshes is the method of choice when one has to deal with complex geometries. In the tokamak fusion community, the number of codes using this technology is at present limited. A preeminent example is EIRENE [29] devoted to the interactions of the plasma and the wall that require an accurate description of the wall geometry, description that can only be addressed easily with non-structured meshes. Another examples for MHD studies are the

the Nimrod [33] and M3DC1 [18] codes that uses triangular high-order finite elements. However, the use of triangular unstructured meshes is actually growing since a realistic description of the interactions between the plasma and the vacuum chamber walls becomes mandatory. Some new edge simulation codes are using this approach [10]. In consequence mesh generation software dedicated to tokamak simulation are currently being developed [34]. The previous examples, however does not seem to fully exploit the use of adaptive meshes that can easily be generated with the modern simplicial mesh generation techniques. We describe in the following the techniques and algorithms that have been developed in Tokamesh for this purpose.

### 0.3.2 Triangulation

A triangular mesh consists of a set of nodes and a set of simplicial elements (triangle in 2D, tetrahedron in 3D) whose edges connect the nodes. To define a triangular mesh, it is then necessary to define these two sets.

#### Choice of the nodes

There are at least two ways to define the set of nodes required to built a triangular mesh : One can start from a small set of elements, and introduce the nodes in an incremental way, one by one maintaining at each step of the procedure, a triangulation of the set of nodes. In this way, the decision to add a node and where to add it, is governed by the overall quality of the triangulation at the previous step. On the other hand, one can also generate once for all, a cloud of unconnected nodes and then built the triangulation of this set of nodes. In tokamesh, we have chosen the second way and we first generate a cloud of nodes. The reason for this choice is that we want to have a flux aligned mesh where the elements have at least one edge on an isoline of the magnetic flux function. It is therefore natural to create a set of nodes that consists of parallel collections of nodes sampling the iso-lines. In addition since orthogonality is a desirable property, it can be interesting to sample the iso-lines to obtain an "orthogonal" mesh where the triangles are close to right-angled triangles. However, in case of a strong curvature of the iso-lines, orthogonal meshes produces a non uniform distribution of the nodes along the iso-lines with very small steps near the magnetic axis. The most satisfactory strategy than we have found therefore consists to use the domain segmentation that will be described in section 0.4.2. Then in the regions corresponding to leaves of the Reeb graph (see section 0.4.2 or in more physical terms in the plasma core and private flux domain, a set of node based on spline approximation of level sets is generated. Figure 0.3.2 shows this distribution of nodes for an equilibrium in jet where 31 isovalues have been generated in this region. The red lines indicates the P1 isovalues computed on the background mesh while the green dots show the distribution of nodes on the spline approximation of these isovalues. On the opposite, in the region corresponding to edges of the Reeb graph (the SOL region), we generate the set of nodes based on streamline integration of the ode

$$\begin{cases} \dfrac{d\boldsymbol{x}}{ds} = v(s)\dfrac{\nabla\psi}{\|\nabla\psi\|} \\ \boldsymbol{x}\,(0) = \boldsymbol{x}_0^i \end{cases} \tag{52}$$

where $v(s)$ is a scalar velocity and where $\boldsymbol{x}_0^i$ are a set of nodes lying on the separatrix. In this way, we generate a mesh that will be locally orthogonal near the separatrix while being characterized by a uniform distribution near the magnetic axis.
In the SOL region, this generation of nodes, is done independently of the existence of the vacuum chamber walls, the nodes exterior to the vacuum chamber are then eliminated in a second

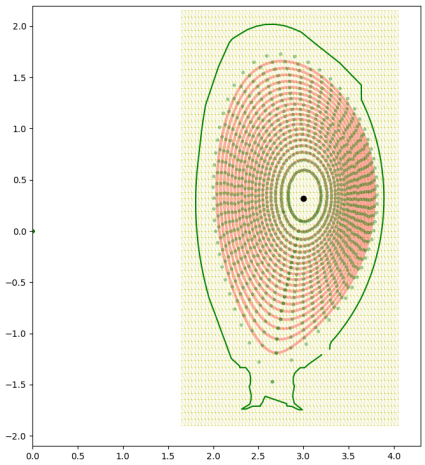



Figure 16: Distribution of nodes in the core region for a jet equilibrium

step. From a practical point, it is much simpler to proceed in this way than to stop the integration when the vacuum chamber walls are crossed by a streamline as the algorithm to decide if a 2D point is inside or outside a closed curve relies only on simple operations. The triangulation algorithm will then use the nodes defining the vacuum chamber and the interior nodes generated in the SOL to construct a triangulation of the vacuum chamber. Figure 0.3.2 illustrates the procedure for an equilibrium in JET where the integration of equation 52 has been stopped at seven different values of the flux function.

**Triangulation of a set of nodes**

An overview of the modern techniques for non-structured mesh generation can be found in [22]. We give below the principal concepts used in this field and applied in the algorithms used in tokamesh.
The construction of an adapted triangular mesh relies on the notion of the "quality" of an element. In 2d, the "best" possible element is the equilateral triangle where the three edges of the triangle are of equal length. An optimal triangulation is therefore a triangulation where all triangles are equilateral. As detailed for instance in [9], one way to adapt this notion of optimal triangulation to adaptive meshing is to define a metric $\mathscr{M}(\boldsymbol{x})$ represented as a symmetric positive definite matrix allowing to define the distance between two points $\boldsymbol{a}$ and $\boldsymbol{b}$ as :

$$d(\boldsymbol{a},\boldsymbol{b}) = \sqrt{(\boldsymbol{b}-\boldsymbol{a}).\mathscr{M}(\boldsymbol{x})(\boldsymbol{b}-\boldsymbol{a})}$$

The notion of equilateral triangles then refers to lengths evaluated with this metric.
In two dimensions, a positive definite symmetric matrix can be written as the product

$$\mathscr{M} = R\Lambda R^t$$

where $R$ is an orthogonal matrix that contains the eigenvectors $\boldsymbol{n}_1$ and $\boldsymbol{n}_2$ while $\Lambda$ is the diagonal matrix of the positive eigenvalues. Therefore a useful geometrical interpretation of the

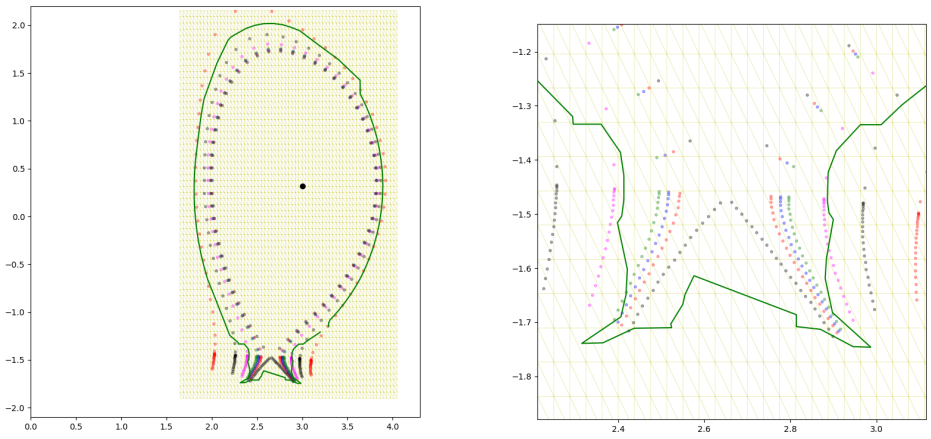


Figure 17: left : Distribution of nodes in the SOL region for a jet equilibrium, right : zoom on the lower region.

Riemanian structure defined on the domain, is to consider that on each point of the domain, the unit ball is transformed into an ellipsis whose axis are given by the two directions $\boldsymbol{n}_1$ and $\boldsymbol{n}_2$ and whose length $h_1$ and $h_2$ on the axis $\boldsymbol{n}_1$ and $\boldsymbol{n}_2$ are given by the inverse of the square root of the eigenvalues of $\Lambda$.
The metric $\mathscr{M}(\boldsymbol{x})$ depends on the point $\boldsymbol{x}$ of the domain. Since in general, it is not possible to define analytically the relation

$$\boldsymbol{x} \rightarrow \mathscr{M}(\boldsymbol{x})$$

It is then necessary to find a way to bring this information to the mesh generator. Tokamesh therefore uses the notion of background mesh to carry this information. In tokamesh, the necessary information to construct an adapted mesh is contained in three files : a background mesh in gmf format, that extends on a spatial domain containing the vacuum chamber, a solution file that contains on the nodes of the background mesh the solution of a Grad-Shafranov equilibrium and a third file containing a polygonal description of the vacuum chamber geometry as a closed list of 2D points (see section 4.2 and 5.1). The Pyamg software will use the information contained in these files to construct the desired metric and triangulation.

**Delaunay and constrained Delaunay triangulation** Once the generation of iso-lines mesh is done (iso-line extractions and distribution of points along the iso-lines), a surface mesh is generated to fill the domain within the edge of the Reeb graph. The iso-line mesh then becomes an input but also a constraint as all the input edges have to match an edge of the triangular mesh. Two different approches have emerged and have proved to be robust to the complexity of the geometry: the frontal and the Delaunay methods. We focus in this report on the Delaunay approach.

The constrained Delaunay approach starts from an initial simple mesh of a box surrounding the surface mesh (composed of six tetrahedra). We then have the following steps:

1. Insert the points of composing the iso-line mesh in the current mesh;

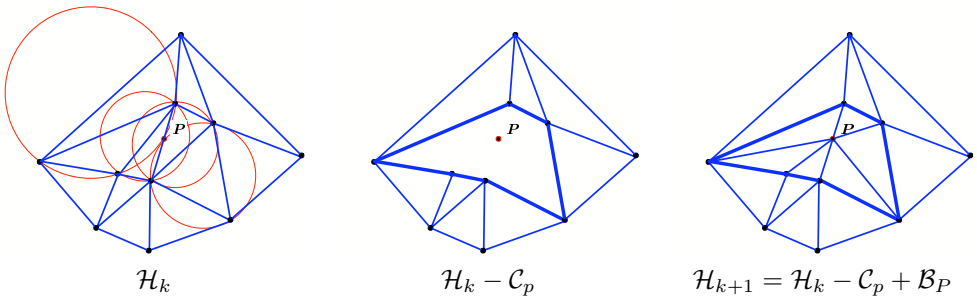



Figure 18: Illustration of the incremental Delaunay insertion of a point in a mesh.

2. Recover the boundary corresponding to the initial edge mesh (list of edges );

3. Fill the interior of the domain by inserting internal points;

4. Optimize the mesh with the smoothing of points and the swap of edges and faces.

Contrary to the frontal approach, a valid 2D mesh is always kept through the entire process. This is due to the insertion procedure based on a iterative process, see Figure 18. Once Step (i) is completed, some faces or edges of the initial mesh may not be present in the current mesh, a boundary recovery is used. It is generally used in enforcing these entities by applying successively or randomly standard optimization operators as the swap of edges [12]. In addition, some theoretical and constructive proofs exist to show that this procedure can succeed to generate a mesh, see [11, 32]. The most critical step is the second one. However, if we accept to modify the initial surface mesh, this procedure can always succeed to output a surface mesh with a (slightly) modified surface mesh. Consequently, this approach is more robust than the frontal approach. Note that Step 3. and 4. are not performed in this specific setting as only the node composing the iso-lines are kept in the final mesh.

Note that a lot of hybrid approaches are a combinaison of both. The frontal creation of points can be used with Delaunay insertion, or the closure of the front can used a complete constrained Delaunay approach.

**Pyamg software** Pyamg relies on the Feflo.a and AMG-Lib INRIA peaces of software. Feflo.a is an robust anisotropic local remeshing software. Surface and volume mesh adaptation are handled in a coupled-way. It also includes boundary layers mesh generation for RANS simulations. It includes:

- CAD re-projection and discrete surface remeshing
- Hybrid mesh generation for boundary-layers
- High-quality quasi-structured grids for complex geometries and complex corners: multi-normals, normals deactivation, ...
- Highly anisotropic mesh adaptation, ratios: O(1:1000)
- Anisotropic/Boundary-layer coupling for supersonic shock/boundary layer interaction

AMG-Lib is the Adaptive Mesh Generation Library based on the Feflo.a technology. Its intent is to ease coupling with external flow solvers. In this project, Pyamg is used. It is a Python interface to the FEFLO.A/AMG-Library. However, only the 2D constrained mesh generation functionality is used. Pyamg uses Python (2.7.x or 3.x) dictionary to store mesh, solution, metric and remeshing options. For instance, the following script defines a simple 2D mesh and performs a uniform refinement:

```
mesh = {}
mesh['xy']        = [ [0,0], [1,0], [1,1], [0,1] ]
mesh['Triangles'] = [ [1,2,3,0],[3,4,1,0]]
```

Then, to refine this mesh with a constant size of 0.1, we have to declare a dictionary of options:

```
remesh_options = {}
remesh_options['hmax'] = 0.1
remesh_options['logfile'] = "remesh-unif.log"
```

To perform the remeshing, just do:

```
mesh_adap = pyamg.adapt_mesh(mesh, remesh_options)
```

For the remeshing options, a generic key exists `options`. It is then possible to pass a list of options directly. For instance, the previous options can be declared as:

```
remesh_opt = {}
remesh_opt["options"] = " -hmax 0.1 -logfile remesh-unif.log  "
```

**Example** We illustrate in this paragraph the different steps involved in the generation of the mesh of a poloidal section of a tokamak, namely the JET tokamak. The starting point is the segmentation of the domain that create a collection of subdomains. Figure 19 illustrates the construction of these subdomains (see section 0.7.3 in part II of this report). Inside each of the

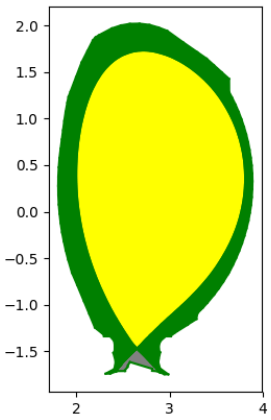



Figure 19: Segmentation of the vacuum chamber of the JET tokamak

subdomains indicated on figure 19 by a different color, the level sets of the flux function are

unique.
The second step of the algorithm then consists to choose inside each subdomain a collection of isolines and to discretize each isolines with a constant number of mesh points (per subdomain) as explained in section 0.3.2. For each subdomain, the segments connecting the points belonging to the same isoline are constrained to belong to the triangulation. Once all the nodes and all the constrained edges have been generated, the following Python script is used to generate the mesh for this example where cloud_mesh denote the set of generated nodes and edges.

```
import pyamg
remesh_options = {}
remesh_options['logfile'] = "gen2d.log"
remesh_options["gen2d"] = []
remesh_options["MeshAllSubDomains"] = []

pymsh = pyamg.adapt_mesh(cloud_mesh,remesh_options)
```

Figure 20 shows the final mesh generated by this procedure containing a total of 801 nodes. In practice, the meshes used for simulations will be much more refined. This example presents intentionally a coarse mesh in order to be able to distinguish its different features.

## 0.4 Block-structured meshes

### 0.4.1 Domain segmentation and Morse function

The generation of block structured grids can be divided into two different tasks :

- First, one have to identify a partition of the computational domain into sub-domains such that the number of singular connection between the sub-domains is minimal and such that each sub-domain can be mapped to the square [0,1]x[0,1]. We will call this part, the domain segmentation problem.
- Second, one has to mesh in a structured way, each sub-domain ensuring some compatibility between the sub-domain (continuity of the finite element)

In the general case, the first part is by far, the most difficult part of the problem. It usually requires some manual input from the users relying on the knowledge that they have from the physical problem at hand. In the tokamak plasma community this is also the case for the simulation codes and meshing tools in use. The meshing tool CARRE [24] relies on a a priori segmentation of the domain into several sub-domain corresponding to the known expected magnetic configurations (single null geometry, double null geometry, disconnected double null). This is also the case in the Jorek code [6], where the meshing tool contains different subroutines corresponding to known configurations : grid_xpoint.f90, grid_double_xpoint.f90, etc or for the SOLEDGE code [3] where a graphical user interface has been designed to help the user to define the sub-domains.

While in the general case, the segmentation problem appears as a very difficult one, the construction of flux surface aligned block-structured grids can benefit from some peculiarities of the problem. Actually, it appears that all the partition strategies of the computational domain used for the generation of 2D meshes in tokamaks plasma modeling rely on some assumptions on the flux function and that it is possible to unify these approaches by a preliminary analysis of the equilibrium fields without having to use some pre-defined configurations. This analysis uses some notion of topology that we recall below.

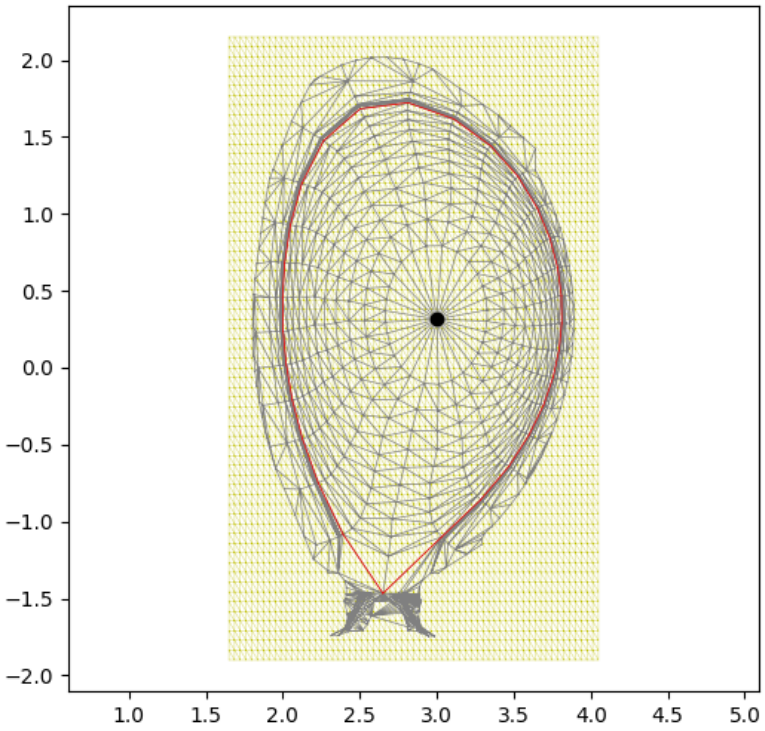



Figure 20: Illustration of the constrained Delaunay mesh generation from an initial set of iso-line mesh.

### 0.4.2 Morse functions

Morse theory relates the study of the critical points of a function $f$ defined on a smooth manifold $\Omega$ to the topology of this space. Introduced by M.Morse in the twenties, this theory etc
In this work, we will restrict ourselves to 2D smooth manifolds and consider that $\Omega$ is a 2-D compact manifold diffeomorphic to a submanifold of the unit sphere $S^2$. In order to avoid technical difficulties, we will assume that the boundary of $\Omega$ is an iso-contour of $f$ and considering the embedding of $S^2$ into $I\!R^3$ we introduce a virtual point located

We begin to state some definitions and classical results.

#### Some definitions and results

**Critical points**
Let $C^r$ be the space of $r$ differentiable scalar field defined on $\Omega$. For $r \geq 2$, a point $\mathbf{p} \in \Omega$ is a critical or singular point of $f$ if $\nabla f = 0$. Considering a (possibly local) coordinate system around $\mathbf{p}$, we therefore have :

$$\frac{\partial f}{\partial x_i}(\mathbf{p}) = 0 \text{ for i} = 1, 2.$$

A critical point is regular (or non-degenerate) if the Hessian $H_f = \left[\frac{\partial^2 f}{\partial x_i \partial x_j}(\mathbf{p})\right]$ of $f$ at $\mathbf{p}$ is invertible.

A function $f$ is a Morse function if all its critical points are regular.

The index $\lambda(p)$ of a regular critical point $\mathbf{p}$ is the number of negative eigenvalues of the Hessian $H_f$.
A Morse function can only have isolated critical points, this is a consequence of the Morse lemma ([19] p 6 ) that states that it exists a local coordinate system $y^1, \cdot, y^n$ with $y^i(\mathbf{p}) = 0$ such that in the neighborhood of $\mathbf{p}$, $f = f(\mathbf{p}) - (y^1)^2 - \cdots - (y^\lambda)^2 + (y^{\lambda+1})^2 + \cdots + (y^n)^2$.
For $n = 2$, if we note $\mu_1, \mu_2$ the two eigenvalues of $H_f$ the only possible critical points of a Morse function are therefore :

| | | | |
|---|---|---|---|
| Maxima | index = 2 | $\mu_1 < 0$ | $\mu_2 < 0$ |
| Saddles | index = 1 | $\mu_1 < 0$ | $\mu_2 > 0$ |
| Minima | index = 0 | $\mu_1 > 0$ | $\mu_2 > 0$ |

The following result allows to check that the identification of critical point is correct :

**Theorem 0.4.1.** *Let $f$ be a Morse function defined on $\Omega$, a region defined by a closed iso-contour, then the number of critical points (counting the virtual minimum) verify the relation :*

$$C_M - C_S + C_m = 2$$

Now let $\boldsymbol{v}$ be a vector field defined on $\Omega$, and $\boldsymbol{x}_0 \in \Omega$ to this vector field can be associated the ordinary differential equation

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{v}(\boldsymbol{x}) \qquad \boldsymbol{x}(0) = \boldsymbol{x}_0$$

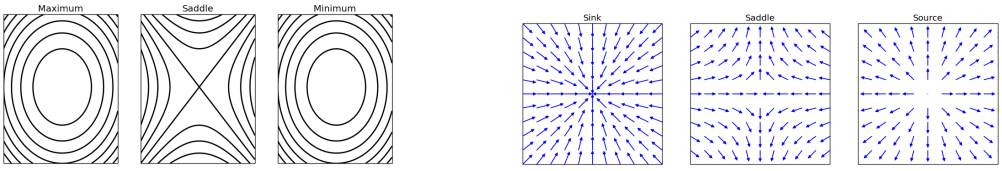
an orbit, streamline or trajectory (these notions are identical for autonomous odes) is a curve

$$\boldsymbol{x}(t) = \boldsymbol{x}_0 + \int_0^t \boldsymbol{v}(u)du$$

These curves are everywhere tangent to the vector $\boldsymbol{v}$.
If we associate to the scalar field $f$ the vector field $\boldsymbol{v} = \nabla f$ or the rotated vector field $\boldsymbol{v}^\perp = \boldsymbol{k} \times \nabla f$ where $\boldsymbol{k}$ is a unit vector field of $\mathbb{R}^3$ orthogonal to $\Omega$, we see that the iso-contours $\gamma(s)$ of $f$ defined by $f(\gamma(s)) = C$ where $C$ is constant are the orbits of the vector field $\boldsymbol{v}^\perp$. Conversely, the orbits of $\nabla f$ are orthogonal to the iso-contour of $f$.

The critical points of $f$ are also the critical points of the vector fields $\boldsymbol{v} = \nabla f$ and $\boldsymbol{v}^\perp = \boldsymbol{k} \times \nabla f$. The vector field $\boldsymbol{v}$ is curl free while $\boldsymbol{v}^\perp$ is divergence free and we have the following correspondence between the critical points

| f | maximum | saddle | minimum |
|---|---|---|---|
| $\boldsymbol{v}$ | source | saddle | sink |
| $\boldsymbol{v}^{\perp}$ | center | saddle | center |




Since the orbits of the divergence free vector field $\boldsymbol{v}^{\perp}$ are the iso-contours of $f$, we use the results given in [23] to give a structural classification of the iso-contours of $f$.

**Theorem 0.4.1.** *Let $f$ be a Morse function defined on $\Omega$. Then the topological set of the iso-contours of $f$ consists of finite connected components that are either*

- *Circle cells which are homeomorphic to open disks*
- *Circle bands which are homeomorphic to open annulus*
- *Saddle connections*

*Proof.* see [23] □

The following result also proven in [23] gives a refined result on the possible saddle connections.

**Theorem 0.4.2.** *Let $f$ be a Morse function defined on $\Omega$. This field is structurally stable if and only if all saddle points are self-connected.*

This last result establishes that the situation depicted in figure 0.4.2 is not structurally stable. This is intuitively easy to understand : An arbitrary small perturbation near any of the two connected saddles will change the value of $f$ at this point and break the saddle connection. In the plasma physics application where the function $f$ is the magnetic flux, the situation depicted in figure 0.4.2 is known as a connected double null (CDN) while the one of figure 0.4.2 is called a disconnected double null (DDN).

As remarked for instance in [24], the CDN is only an idealization and in real experiments, the two saddles are never on exactly the same iso-contour. For the meshing algorithm, we are considering in this work, there is no necessity to consider the CDN pattern and it will be sufficient to consider the figure 0.4.2 pattern as the generic one : if the values of the magnetic flux in the two saddles points are too close, we will simply fuse the two iso-values and there will be no need to mesh the inter-region between the saddle iso-contours since its surface will be zero.

To sum up the results of this section in a concrete way, we have established that $\Omega$ consists of connected regions that contains only closed orbits. These regions are either circle cells containing an extremum critical point or circle bands separated from the other connected regions by self connected saddles. In the next section, we will give a concrete way to construct these regions and to organize the neighboring relationship between them.

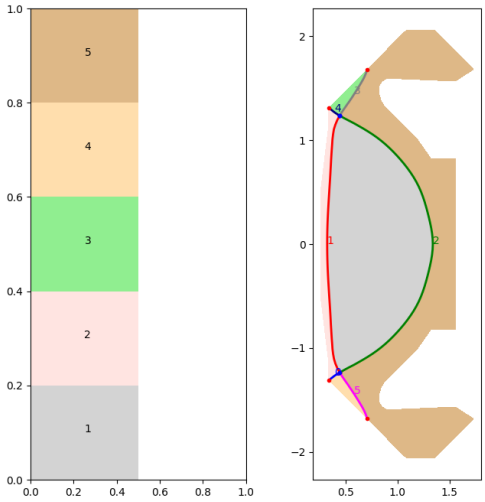



Figure 21: A connected double null in the MAST tokamak

**Reeb graph**

In the previous section, we have seen that it is possible to split the domain $\Omega$ into connected component that are either homeomorphic to a disk or to an annulus. The Reeb graph [28] gives a concrete way to construct and store this decomposition of the domain. To define formally the Reeb graph, we first define an equivalence relation between points of $\Omega$.

**Definition 0.4.1.** Given a topological space $\Omega$ and a continuous function $f : \Omega \to I\!R$, two points $p$ and $q$ are equivalent $p \sim q$ if they belong to the same connected component of a single iso-contour $f^{-1}(c)$ for some $c \in I\!R$.

The formal definition of the Reeb graph is then :

**Definition 0.4.2. Reeb Graph :** The Reeb graph is the quotient space $\Omega/\sim$ endowed with the quotient topology.

Loosely speaking, the Reeb graph concatenates all the points belonging to the same connected component of a level set into a single representative.
In the case of a Morse function that have only isolated critical points, the construction of the Reeb graph can be visualized in the following way : Let us consider the graph $\mathcal{F}$ of the function $f$ defined by $\boldsymbol{x} = (x_1, x_2, x_3) = (\mathbf{p}, f(\mathbf{p})) \in I\!R^3$ for $\mathbf{p} \in \Omega$. We "slice" $\mathcal{F}$ by the plane $x_3 = C$ and evolve $C$ in the positive direction starting from from $-\infty$. Each intersection of $\mathcal{F}$ with the plane $x_3 = C$ corresponds to a set of connected iso-contour $f^{-1} = C$. The change in topology of this set occurs at critical points of $f$. As $C$ goes from $-\infty$ (the virtual pit) to the global maximum of $f$, a new connected level set will appear when $C$ passes trough a minimum. Conversely a connected iso-contour will disappear when $C$ goes beyond a maximum. When $C$ passes through a saddle, two components of the iso-contour $f^{-1} = C$ will merge giving birth to two new connected components as depicted in Figure 0.4.2.

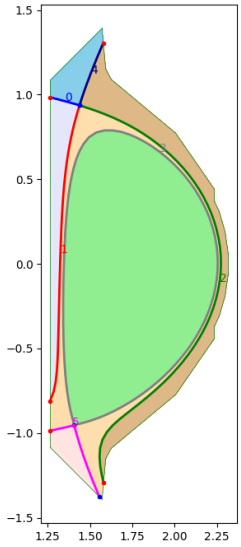



Figure 22: A disconnected double null in the DIIID tokamak

The nodes of the Reeb graph then corresponds to critical points of $f$ and each edge corresponds to a change in the number of connected components that occur when the slicing plane passes through this node. Consequently an edge originates (resp. terminates) at the nodes corresponding to a minimum (resp. maximum) and these nodes have degree 1. If the node corresponds to a saddle, two components of the level set merge and two new closed component appear. The corresponding node of the Reeb graph is therefore of degree 3 and looks like the letter "Y".
For Morse functions defined on a orientable surface of genus 0, the Reeb Graph contains no loop and on a flat $\Omega$, it is a tree. Reeb Graph are therefore also called sometimes contour tree. Due to their numerous applications in computational geometry and computer graphics, the efficient construction of Reeb graphs has been well studied. In this work, we have used the algorithm described in [31]. Several efficient alternative exists but this one has been found sufficient for our purpose.

### 0.4.3 Meshing algorithm for $\mathscr{C}^0$ meshes

We are now in position to describe our meshing algorithm. Given a domain $\Omega$ and a Morse function $f : \Omega \to I\!R$ we first identify the critical points of $f$ and construct its Reeb Graph. Each edge of the Reeb Graph corresponds to a connected subdomain of $\Omega$ that contains only closed level sets of $f$. Thus let $e$ be an edge of the Reeb graph and $e_1, e_2$ be its two extremities. Let $\Omega_e$ be the subdomain defined by $e$. In $\Omega_e$, the value of $f$ evolves monotically from $f_m = \min(f(e_1), f(e_2))$ to $f_M = \max(f(e_1), f(e_2))$. Moreover, the domains $\Omega_e$ are of only two different types : either the vertices $e_1$ and $e_2$ are two saddle points or one and only one of these two vertices, say $e_1$ is an extremum of $f$. We therefore have the following result :

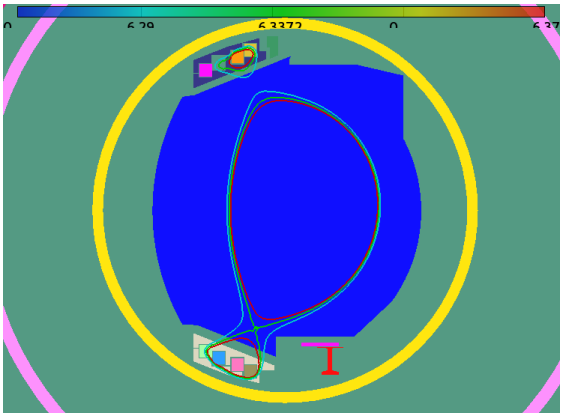


Figure 23: Behavior of the level sets when the plane $x_3 = C$ passes through a saddle : For $C <$ saddle value, the level set is the single blue line, at $C =$ saddle value, the two components merge (green line) giving rise to two connected components for $C >$ saddle value (red line)

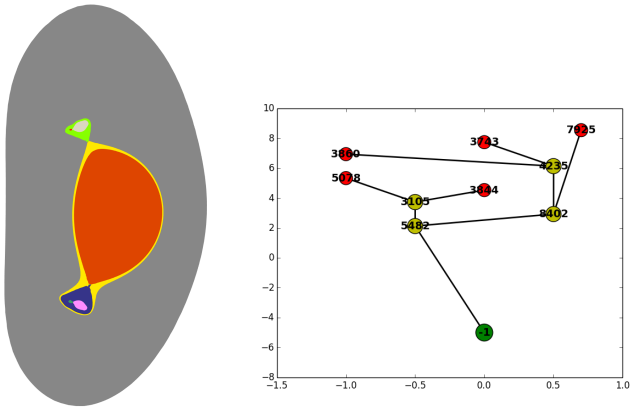



Figure 24: Example of domain decomposition (left) encoded in a Reeb graph (right)

**Theorem 0.4.1.** *Let $e$ be an edge of the Reeb graph and $\Omega_e$ be the subdomain of $\Omega$ associated to $e$. Let $f_m^e, f_M^e$ with $f_m^e < f_M^e$ be the value of $f$ associated to the extremities of $e$ then there exists a one to one mapping $G^e : (f, s) \in [f_m^e, f_M^e] \times [0, 1] \to \mathbf{p} \in \Omega_e$.*

*Proof.* We can first remark that the last paragraph of section 0.1.2 provides a concrete way to construct this mapping and thus establish this result. We provide below an alternate proof. Let $c \in [f_m^e, f_M^e]$ then $c$ identifies in a unique way a single level set and $f^{-1}(c)$ is a closed 1-D curve $\subset \Omega_e$. It therefore suffices to give a one-to-one mapping from $[0, 1] \to f^{-1}(c)$ to prove the result. But $\Omega_e$ can only be of two different types :

1. $e$ is an extremal edge of the graph and one of its extemities, say $e_1$ is an extremum of $f$ then $\Omega_e$ is homeomorphic to a disk and we can associate to any $s \in [0, 1]$ the unique $\mathbf{p} = f^{-1}(c) \cap \mathcal{D}$ where $\mathcal{D}$ is the straight line passing through $e_1$ and making an angle $2\pi \times s$ with the x-axis.

2. $e$ is internal edge of the graph and its two extremities are saddle points of $f$ then $\Omega_e$ is

a circle band and we can identify a point of $f^{-1}(c)$ by its scaled curvilinear coordinate $\int_0^s |dl|/L$ where $dl$ is a differential displacement vector along the curve $f^{-1}(c)$ and $L$ its total length.

□

**Remark :** In the first case, the mapping $G$ is singular since on the extremum the contour line is reduced to a single point. On $\Omega_e$, the generated mesh is a polar mesh that contains a singular point at its center. In practical applications (see the example section) we either assume that the numerical method is able to handle the degenerate case of the center of a polar grid or we will modify locally the mesh in the vicinity of the center to avoid this situation.

The previous result shows that it is possible to construct a block structured grid $\Omega_e^h$, aligned with the iso-contours of $f$ into each of the subdomain $\Omega_e$. We now show that it is possible to glue these individual grids to obtain a single mesh covering all the domain. In the following, we will use a lagrange type discretization for concretness, but other type of discretization can be used. Since inside each $\Omega_e$ the contour lines are closed and their values vary monotically in $[f_m^e, f_M^e]$, let us choose $\{f_1^e, f_2^e, \cdots, f_m^e\}$ an increasing sequence of values $\in [f_m^e, f_M^e]$. This set can be choosen in a totally arbitrary manner with no influence on the discretization of the other subdomains $\Omega_{e'}$ with $e' \neq e$ and therefore it is sufficient to examine the discretization in the $s$ coordinate. Let $\{s_j^e : i = j, \cdots, n_e\}$ be a sequence of nodes choosen $\in [0, 1]$. Using the mapping $G^e$, it is clear that the set $\Omega_e^h = G^e(f_i^e, s_j^e)$ defines a structured grid of $\Omega_e$.

Then let us consider the Reeb graph, and assume that this graph contains $K$ extremal edge or alternatively $K$ vertices of degree 1 (the leaves). We now construct the $K$ paths $P_k$ in the Reeb graph connecting the $K$ extremal vertices to the virtual pit. These $K$ paths are composed of a unique extremal edge and a sequence of edges whose vertices are saddle points. We will call $\{[m^k, S_1^k], [S_1^k, S_2^k] \cdots, [S_m^k, p]\}$ the sequence of edges composing the $k^{th}$ path with $m^k$ the extremal vertex and $p$ the virtual pit. Note that this implies that the edges are ordered and that in this way we have an order relation between edges of the same path. We adopt the notation $\Omega_{[v_1,v_2]}$ to designate the subdomain corresponding to the edge $[v_1, v_2]$ in the Reeb graph. To initialize the grid generation algorithm, we begin by choosing $K$ sequences $\{s_i^k : i = 1, \cdots, n_k\}$ of nodes $\in [0, 1]$ and construct the $K$ grids $\Omega^h_{[m^k,S_1^k]} = G^{[m^k,S_1^k]}(f_i^k, s_j^k)$ corresponding to the leaves of the Reeb graph. In this construction, the only requirement imposed to these $K$ grids will be that the saddle $S_1^k$ belongs to the grid $\Omega^h_{[m^k,S_1^k]}$.

Then the construction of the grids $\Omega^h_{[S_j,S_{j+1}]}$ will be done according to the following rules :

1. if two extremal paths $P_k$ and $P_{k'}$ meet at the saddle $S_j$, let $[S_{j-1}^k, S_j]$ and $[S_{j-1}^{k'}, S_j]$ be the two edges of the Reeb graph distinct of $[S_j, S_{j+1}]$ that contain $S_j$, then construct first the grids $\Omega^h_{[S_{j-1}^k,S-j]}$ and $\Omega^h_{[S_{j-1}^{k'},S-j]}$

2. The saddles $S_j$ and $S_{j+1}$ belong to the grid $\Omega^h_{[S_j,S_{j+1}]}$.

3. the number of nodes $n_{[S_j,S_{j+1}]}$ used to discretize the $s$- coordinate in $\Omega^h_{[S_j,S_{j+1}]}$ verifies :

$$n_{[S_j,S_{j+1}]} = n_{[S_{j-1}^k,S_j]} + n_{[S_{j-1}^{k'},S_j]}$$

### 0.4.4 $\mathscr{C}^1$ meshes

Consider a spatial domain composed of a collection of patches or subdomains corresponding to edges of a Reeb graph. The boundary of each patch is therefore a closed isoline (or a part of it) and the generic case is that a closed isoline is the boundary of two different patches. One can construct a grid or mapping of each subdomain in an independent way and "glue" the different patches together. If $\mathscr{S}^i(s,t)$ is the mapping defining the patch $i$, a $\mathscr{C}^0$ gluing only requires that the two curves defining the boundaries of the patches are identical, for instance :

$$\mathscr{S}^j(0,t) = \mathscr{S}^i(0,t) = \mathscr{C}(t) \tag{53}$$

if $\mathscr{C}(t)$ is the curve defining the boundaries of the patches $i$ and $j$. On the contrary a $\mathscr{C}^1$ gluing will require not only that the relation (53) is verified but also that the $s-$derivatives to this curve are proportional.

$$\frac{\partial \mathscr{S}^j}{\partial s}(0,t) = \alpha(t)\frac{\partial \mathscr{S}^i}{\partial s}(0,t) \tag{54}$$

where $\alpha$ is a scalar factor. We emphasize that (54) is sufficient for $\mathscr{C}^1$ continuity between the patches when only two patches are involved. If more than two patches meet at one point, additional constraints are required. However, this is the general situation for our problem, except on the x-points where more than two patches are present. Thus in the next section, we will describe how the mapping of the patches and the gluing of two patches is done when the curves separating the patches are defined by splines functions.

#### Grid construction of a patch enclosed by isolines

We consider a sub-domain bounded by two level sets $\psi(\boldsymbol{x}) = \psi_m$ and $\psi(\boldsymbol{x}) = \psi_M$. This section explains the construction of the mapping of this region using a spline tensor product. For simplicity, we consider only two elements in the radial (orthogonal to the level sets) direction but the procedure can be easily generalized to an arbitrary number of elements. Thus, let $S = [s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9] = [0, 0, 0, 0, 0.5, 1, 1, 1, 1]$ be an open-knot vector that defines a spline vector space $\mathbb{S}^3$ of cubic splines with two elements. The b-spline basis functions $N_i(s), i = 1, \cdots, 5$ generating this space are represented in figure 0.4.4.

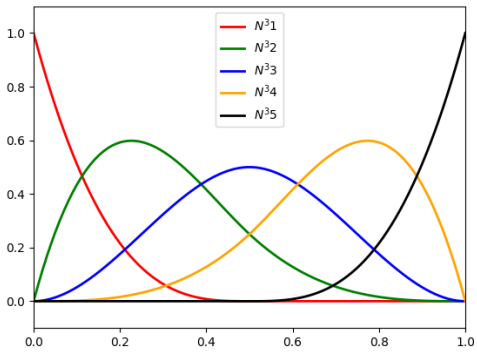



Figure 25: b-spline basis function associated to the knot vector $S = [0, 0, 0, 0, 0.5, 1, 1, 1, 1]$

Define $n$ as the number of "elements" and let $T = [\frac{-3}{n}, \frac{-2}{n}\frac{-1}{n}, 0, \frac{1}{n}, \cdots, \frac{n-1}{n}, 1, 1+\frac{1}{n}, 1+\frac{2}{n}, 1+\frac{3}{n}]$ be a periodic knot vector that defines a periodic spline vector space $\mathbb{S}^3_{per}$ of dimension $n$ with generating basis function $N_j(t), j = 1, \cdots, n$. Taking the tensor product of the two spaces $\mathbb{S}^3 \otimes \mathbb{S}^3_{per}$, we can construct bivariate functions by :

$$\mathscr{S}(s,t) = \sum_{i=1}^{5}\sum_{j=1}^{n} \boldsymbol{C}_{i,j} N_i(s) N_j(t) \tag{55}$$

Now, let $\mathscr{C}_k(t), k = 1, \cdots, 5$ be 5 curves defined by

$$\mathscr{C}_k(t) = \sum_{j=1}^{n} \boldsymbol{x}_{k,j} N_j(t) \quad k = 1, \cdots, 5 \tag{56}$$

We ask under what conditions the 5 curves $\mathscr{S}(s_k, t)$ for $s_k \in \{0, 0.25, 0.5, 0.75, 1\}$ can be identical to the curves $\mathscr{C}_k(t), k = 1, \cdots, 5$. That is we want to solve for the $5 \times n$ unknowns $\boldsymbol{C}_{i,j}$ the system:

$$\sum_{i=1}^{5}\sum_{j=1}^{n} \boldsymbol{C}_{i,j} N_i(s_k) N_j(t) = \sum_{j=1}^{n} \boldsymbol{x}_{k,j} N_j(t) \tag{57}$$

since the $N_j(t)$ form a basis, this gives for each $j = 1, \cdots, n$, the $5 \times 5$ system :

$$\sum_{i=1}^{5} \boldsymbol{C}_{i,j} N_i(s_k) = \boldsymbol{x}_{k,j} \text{ for } s_k \in \{0, 0.25, 0.5, 0.75, 1\} \tag{58}$$

Actually, (see figure 0.4.4) since $N_i(0) = 0$ except for $i = 1$ (resp. $N_i(1) = 0$ except for $i = 5$), we get $\boldsymbol{C}_{1,j} = \boldsymbol{x}_{1,j}$ and $\boldsymbol{C}_{5,j} = \boldsymbol{x}_{5,j}$ and we just have to solve the 3 equations :

$$\sum_{i=1}^{5} \boldsymbol{C}_{i,j} N_i(s_k) = \boldsymbol{x}_{k,j} \text{ for } s_k \in \{0.25, 0.5, 0.75\} \tag{59}$$

Since the basis functions are of compact support, these equations gives a $3 \times 3$ system that writes explicitly (see figure 0.4.4 ) :

$$\begin{pmatrix} 0.59375 & 0.25 & 0.03125 \\ 0.25 & 0.5 & 0.25 \\ 0.03125 & 0.25 & 0.59375 \end{pmatrix} \begin{pmatrix} \boldsymbol{C}_{2,j} \\ \boldsymbol{C}_{3,j} \\ \boldsymbol{C}_{4,j} \end{pmatrix} = \begin{pmatrix} \boldsymbol{x}_{2,j} - 0.125\boldsymbol{x}_{1,j} \\ \boldsymbol{x}_{3,j} \\ \boldsymbol{x}_{4,j} - 0.125\boldsymbol{x}_{5,j} \end{pmatrix} \tag{60}$$

However, this procedure does not give any means to control the curves $\mathscr{S}(s, t^*)$ with $t^*$ fixed. Obviously, one can reverse the role of $s$ and $t$ and instead of the 5 curves $\mathscr{C}_i(t), i = 1, \cdots, 5$ consider $n$ curves $\mathscr{T}_j(s), j = 1, \cdots, n$ in the $s$ direction. However this time, the procedure will not control the curves in the $t$ direction.

Thus instead, we propose a procedure where one control *some* of the $s = constant$ curves and *some* of the $t = constant$ curves. More precisely, we will impose that the three curves $\mathscr{S}(s_k, t)$ for $s_k \in \{0, 0.5, 1\}$ are identical to the three curves $\mathscr{C}_k(t)$ for $k = 1, 3, 5$. As seen previously, this requirement sets the control points $\boldsymbol{C}_{1,j}$ and $\boldsymbol{C}_{5,j}$ and we are left with the system (60) for $s_k = 0.5$ that writes :

$$\frac{\boldsymbol{C}_{2,j}}{2} + \boldsymbol{C}_{3,j} + \frac{\boldsymbol{C}_{4,j}}{2} = 2\,\boldsymbol{x}_{3,j} \quad \forall j = 1, \cdots, n \tag{61}$$

This system gives explicitly $\boldsymbol{C}_{3,j}$ if $\boldsymbol{C}_{2,j}$ and $\boldsymbol{C}_{4,j}$ are known. To compute these last control points, we will require that for each $t$ the derivatives at $s = 0$ and $s = 1$ of the curves $\mathscr{S}(s,t)$ have known values, say $\boldsymbol{x}'(0,t)$ and $\boldsymbol{x}'(1,t)$. The rationale of such a requirement is to insure the $\mathscr{C}^1$ continuity of the curves $t = constant$ when passing from one patch to a neighboring one. The system that we have now to solve is thus

$$\begin{array}{l} \partial_s(\mathscr{S}(s,t))(0,t) = \boldsymbol{x}'(0,t) \quad \forall t \\ \partial_s(\mathscr{S}(s,t))(1,t) = \boldsymbol{x}'(1,t) \quad \forall t \end{array} \tag{62}$$

Computing the derivatives of the basis functions $N_i(s)$ in 0 and 1 gives explicitly :

$$\begin{array}{l} \sum_{j=1}^{n} 6(\boldsymbol{C}_{2,j} - \boldsymbol{C}_{1,j}) N_j(t) = \boldsymbol{x}'(0,t) \quad \forall t \\ \sum_{j=1}^{n} 6(\boldsymbol{C}_{5,j} - \boldsymbol{C}_{4,j}) N_j(t) = \boldsymbol{x}'(1,t) \quad \forall t \end{array} \tag{63}$$

To solve (63)we will develop $\boldsymbol{x}'(0,t)$ and $\boldsymbol{x}'(1,t)$ on the $N_j(t)$ basis. Thus, let $\boldsymbol{x}'_{0,j}$ and $\boldsymbol{x}'_{1,j}$ be the control points defining the functions $\boldsymbol{x}'(0,t)$ and $\boldsymbol{x}'(1,t)$ :

$$\begin{array}{l} \boldsymbol{x}'(0,t) = \sum_{j=1}^{n} \boldsymbol{x}'_{0,j} N_j(t) \quad \forall t \\ \boldsymbol{x}'(1,t) = \sum_{j=1}^{n} \boldsymbol{x}'_{1,j} N_j(t) \quad \forall t \end{array} \tag{64}$$

The equation (62) gives :

$$\begin{array}{l} 6(\boldsymbol{C}_{2,j} - \boldsymbol{C}_{1,j}) = \boldsymbol{x}'_{0,j} \quad \forall j \\ 6(\boldsymbol{C}_{5,j} - \boldsymbol{C}_{4,j}) = \boldsymbol{x}'_{1,j} \quad \forall j \end{array} \tag{65}$$

and can be solved for $\boldsymbol{C}_{2,j}, \boldsymbol{C}_{4,j}$. Now solving for $\boldsymbol{C}_{3,j}$ thanks to (61) defines completely the spline surface $\mathscr{S}(s,t)$. On this surface, the curves $\mathscr{S}(s_k,t)), s_k \in \{0, 0.5, 1\}$ exactly coincide with the three curves $\mathscr{C}_k(t), k \in \{1,3,5\}$ and the definition of the set of control points $\boldsymbol{C}_{2,j}, \boldsymbol{C}_{4,j}$ ensures that the derivatives of the curves $\mathscr{S}(s,t))$ in $s = 0$ and $s = 1$ are controlled. This procedure is illustrated in figure 0.4.4. The black lines represents the curves $\mathscr{S}(s_*,t)$ for $s_*$ fixed while the blue ones are the curves $\mathscr{S}(s,t_*)$ for $t_*$ fixed. Observe that the three controlling curves $\mathscr{C}_k(t)$ (in red) are exactly superposed with the curves $\mathscr{S}(s,t))$ (in black) for $s = 0, 0.5, 1$ defined on the surface. Here the derivatives at $s = 0$ and $s = 1$ have been chosen to be proportional to the gradient of the function $\psi$. This makes the grid locally orthogonal in the vicinity of the two level sets $\mathscr{S}(0,t))\ \mathscr{S}(1,t))$ as can be seen on figure 0.4.4. As already mentioned, this procedure can be easily generalized to an arbitrary number of elements in the $s$ direction. The only change is that instead of the equation (61), we will have to solve a tridiagonal linear system. In this way, one can obtain a grid whose $\mathscr{S}(s_*,t))$ curves are exactly aligned on an arbitrary number of level sets.

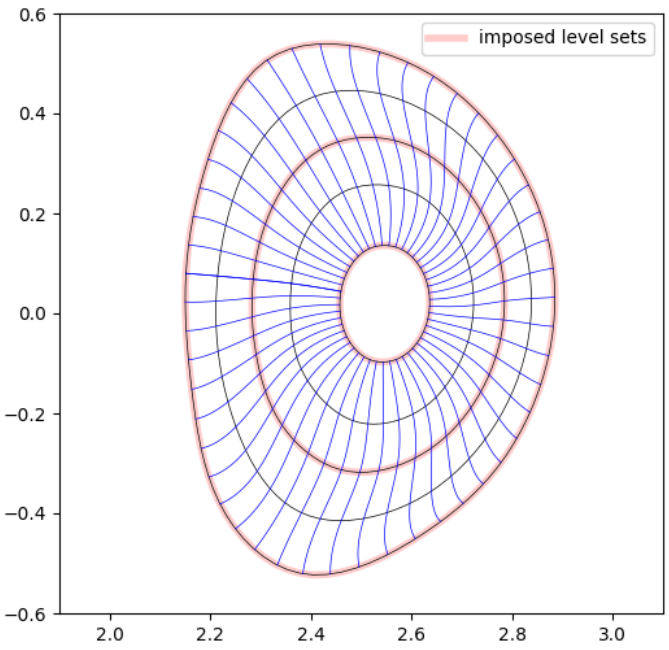



Figure 26: Tensor product surface of a patch enclosed by two level sets.

**Grid construction of a patch enclosed by non $\mathscr{C}^1$ isolines**

The above procedure is appropriate for the case where the two isolines $\psi(\boldsymbol{x}) = \psi_m$ and $\psi(\boldsymbol{x}) = \psi_M$ have $\mathscr{C}^1$ regularity. However when one of the two isolines enclosing the patch passes through an x-point, the previous strategy have to be adapted since the curve $\psi(\boldsymbol{x}) = \psi_m$ is no more $\mathscr{C}^1$ and cannot be expanded on a basis of periodic splines. Moreover, the normal to $\psi(\boldsymbol{x}) = \psi_m$ is not defined on the x-point and has a discontinuity on this point making the spline approximation of the gradient defined in 64 problematic. We therefore have to modify the previous algorithm to cope with this case.

A second problem, is that we would like the loss of $\mathscr{C}^1$ regularity to be as localized as possible. Ideally, we would like this loss of regularity to be confined only in the elements surrounding the x-point. Thus, we will in this section consider the construction of a surface composed by a *unique* element in the $s$ direction enclosing a region defined by 2 isolines $\psi(\boldsymbol{x}) = \psi_m$ and $\psi(\boldsymbol{x}) = \psi_M$ where one of them ,say for instance $\psi(\boldsymbol{x}) = \psi_m$ ) is not $\mathscr{C}^1$. The vector space of cubic splines defined by a unique element and the open-knot vector$[0, 0, 0, 0, 0, 1, 1, 1, 1]$ is generated by the b-spline basis functions $N_i(s), i = 1, \cdots, 4$ represented in figure 0.4.4. For future

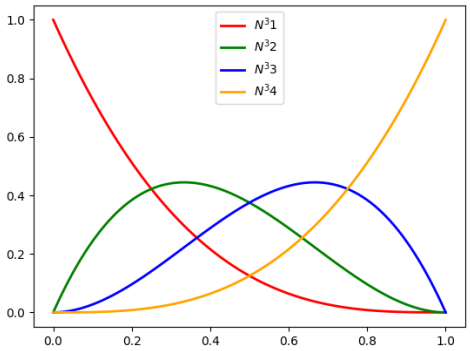



Figure 27: b-spline basis function associated to the knot vector $S = [0, 0, 0, 0, 1, 1, 1, 1]$

reference, the expression of these functions and of their derivatives are given below :

$$\begin{array}{lll} N_1^3(s) = (1-s)^3 & (N_1^3)'(s) = -3(1-s)^2 & (N_1^3)''(s) = 6(1-s) \\ N_2^3(s) = 3s(1-s)^2 & (N_2^3)'(s) = 3(1-s)(1-3s) & (N_2^3)''(s) = -6(2-3s) \\ N_3^3(s) = 3s^2(1-s) & (N_3^3)'(s) = 3s(2-3s) & (N_3^3)''(s) = 6(1-3s) \\ N_4^3(s) = s^3 & (N_3^3)'(s) = 3s^2 & (N_3^3)''(s) = 6s \end{array} \tag{66}$$

In the $t$ direction, we cannot use periodic splines, thus we use clamped splines defined on the open-knot vector $T = [0, 0, 0, 0, \frac{1}{n}, \cdots, \frac{n-1}{n}, 1, 1, 1]$ where $n$ is the number of elements. As in the previous section, we consider the surface

$$\mathscr{S}(s,t) = \sum_{i=1}^{4}\sum_{j=1}^{n} \boldsymbol{C}_{i,j} N_i(s) N_j(t) \tag{67}$$

and given the two splines approximation $\mathscr{C}_k(t), k = 1, 2$ of the two isolines $\psi(\boldsymbol{x}) = \psi_m$ and $\psi(\boldsymbol{x}) = \psi_M$ :

$$\mathscr{C}_k(t) = \sum_{j=1}^{n} \boldsymbol{x}_{k,j} N_j(t) \quad k = 1, 2 \tag{68}$$

we require the 2 curves $\mathscr{S}(0, t)$ and $\mathscr{S}(1, t)$ to be identical to the curves $\mathscr{C}_k(t), k = 1, 2$. This immediately gives :

$$C_{1,j} = \boldsymbol{x}_{1,j} \quad C_{4,j} = \boldsymbol{x}_{2,j} \quad \forall j$$

To defines the surface, again as in the previous section, we will require that the derivatives in $s = 0$ and $s = 1$ of the curves $\mathscr{S}(s, t)$ match two prescribed directions $\boldsymbol{d}^1(t)$ and $\boldsymbol{d}^2(t)$ defined by two sets of control points $\boldsymbol{d}_{1,j}$ and $\boldsymbol{d}_{2,j}$ defined on the basis $N_j(t)$. Given the expression of the derivatives of the basis functions given in 66, we obtain :

$$C_{2,j} = C_{1,j} + \boldsymbol{d}_{1,j}/3 \quad C_{3,j} = C_{4,j} - \boldsymbol{d}_{2,j}/3 \quad \forall j \tag{69}$$

However, in section 0.4.4, the direction fields $\boldsymbol{d}_{1,j}, \boldsymbol{d}_{2,j}$ can be defined by the gradient of the magnetic flux field. This is no more possible on the $\mathscr{C}_1(t)$ curve since the gradient is not defined on the x-point and is a highly discontinuous quantity. We have therefore found convenient to define the $\boldsymbol{d}_{1,j}$ control points as an average between the normal vectors of the two curves $\mathscr{C}_k(t), k = 1, 2$ :

$$\boldsymbol{d}_{1,j} \leftarrow \alpha \boldsymbol{d}_{1,j} + (1 - \alpha) \boldsymbol{d}_{2,j} \tag{70}$$

where $\boldsymbol{d}_{1,j}$ and $\boldsymbol{d}_{2,j}$ are the control points of the gradient field along the curves $\mathscr{C}_k(t), k = 1, 2$ and $\alpha$ is a weighting factor null on the x-point and close to 1 far away from this point. In this way, we smooth out the discontinuity on the x-point and construct a surface where only the elements directly touching the x-point are not $\mathscr{C}^1$. This can be verified on figure 0.4.4 that display the grid between the isoline $\psi = -1$ and $\psi = -0.95$[4].

[4]The flux function is normalized between -1 on the x-point and 0 on the magnetic axis

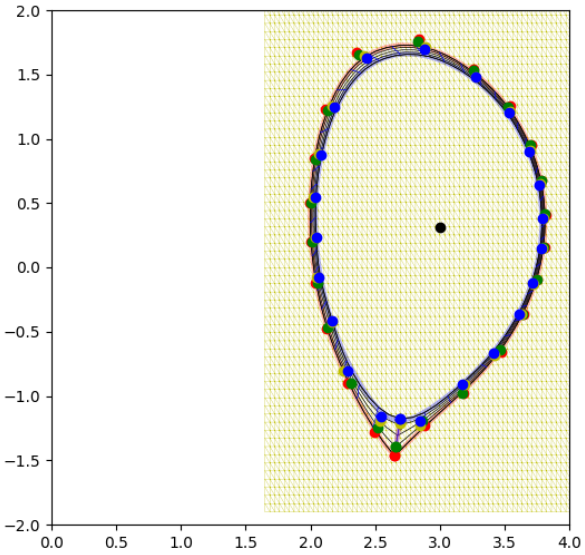



Figure 28: One-element surface containing an-x-point

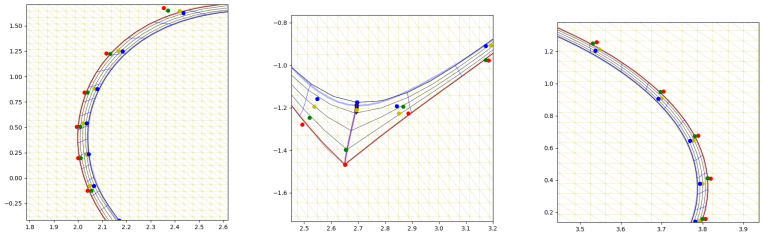


Figure 29: Zoom on different parts of the one-element surface

The element is quite narrow in order to confine the width of the discontinuity to a very small extent in the $s$ direction. As can be seen on the enlarged view of figure 0.4.4, in the $t$ direction, the derivative discontinuity is confined to the first element near the x-point.

### $\mathscr{C}^1$ gluing of two patches

We consider in this section the problem of constructing a global $\mathscr{C}^1$ mapping of a domain described by a collection of patches. This is in general a difficult problem. However for domains described by a Reeb graph, except on saddle points, we need only to stick two patches since a patch has only one direct neighbor. This sticking of two patches can be realized using parametric continuity ($\mathscr{C}^1$) or a geometric continuity ($\mathscr{G}^1$). For the $\mathscr{C}^1$ condition, the derivative vectors

are in opposite directions with the same magnitude, while for the $\mathscr{G}^1$ the derivative vectors are in opposite directions but can be of different magnitude. As explained in section 0.1.3, these conditions translate on geometric conditions on the control points requiring the alignment of three control points. When only two patches have to be glued, this condition is therefore easy to satisfy.

As a first example, consider the construction of a regular grid inside a region limited by the separatrix. As explained in the previous sections (0.4.4 and 0.4.4) since we want the derivative discontinuity generated by the x-point to extend in the smallest region possible, we will generate two patches : $\mathscr{S}^c(s,t)$ an inner surface for the core region with $\mathscr{C}^1$ regularity and $\mathscr{S}^e(s,t)$ a one-element surface containing the x-point where the derivative discontinuity is limited to the 2 elements containing the x-points. The common boundary of the two surfaces is a spline curve $\mathscr{C}^(t)$ defined by the control points $\boldsymbol{P}_{0,j}$ (purple curve on figure 0.4.4). Given for instance, the control points $\boldsymbol{P}^c_{1,j}$ that define the $s$-derivative on the boundary of $\mathscr{S}^c(s,t)$, it is easy to define by the equation (32) the $\boldsymbol{P}^e_{1,j}$ control points in the patch $\mathscr{S}^e(s,t)$ using for instance a $\mathscr{G}^1$ condition.

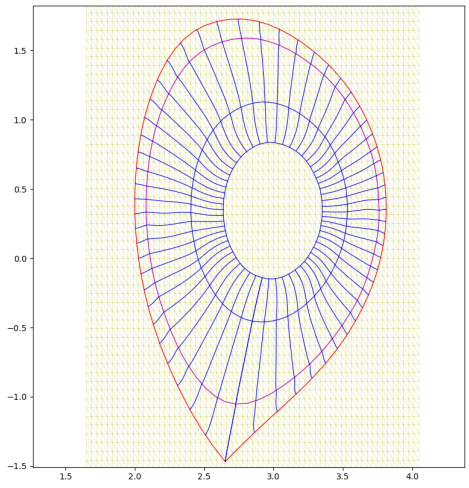



Figure 30: $\mathscr{G}^1$ regular grid inside the Separatix.

Our second example is more complex and involves sticking 5 different patches around the x-point. We start by constructing a grid for the SOL region by sticking 3 different patches (figure 0.4.4). Since the separatix crosses itself on the x-point, in a first step shown in figure 0.4.4, we approximate the isolines $C_1$ and $C_2$ passing through the x-point with specified end derivatives in order to have $\mathscr{C}^1$ continuity with the two branches of the isoline $C_0$. Once this is done, we construct a grid for the SOL region by sticking together 3 different patches (figure 0.4.4).

As in the first example, the next step is the construction of one-element surface inside the vaccum vessel and a regular surface for the core region. The result of this procedure is shown in figure 0.4.4

In the last step, we proceed to join the CORE/Edge and SOL grids. The boundary of the two

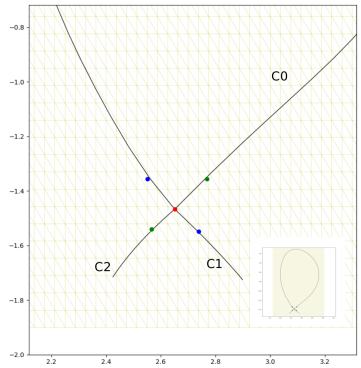



Figure 31: $\mathscr{C}^1$ continuity of splines curves across the X-point.

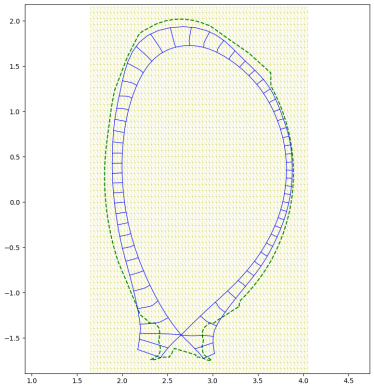


Figure 32: A regular grid of the SOL region.

patches is the curve approximating the separatix (cf. the 1$^{\text{st}}$ step).

Note that here, to define the normal derivative to the separatrix, we have chosen to use the gradient of the flux function. This create a local orthogonal grid in this region.

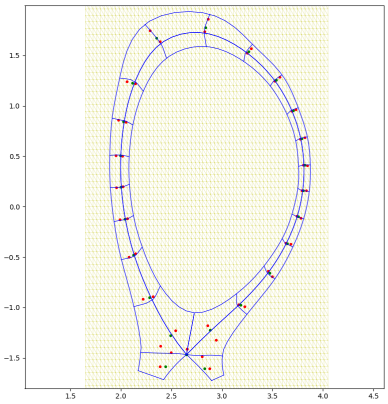
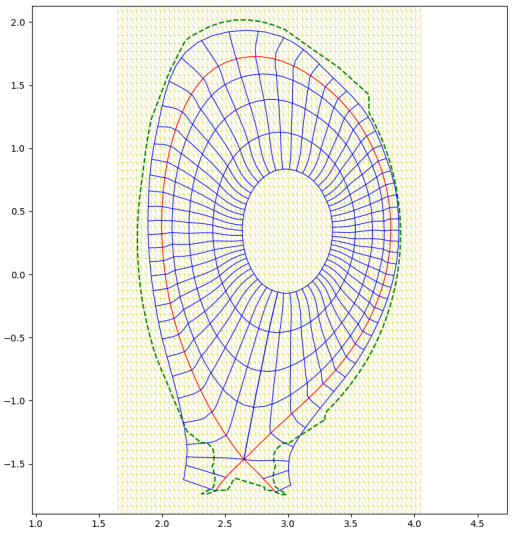
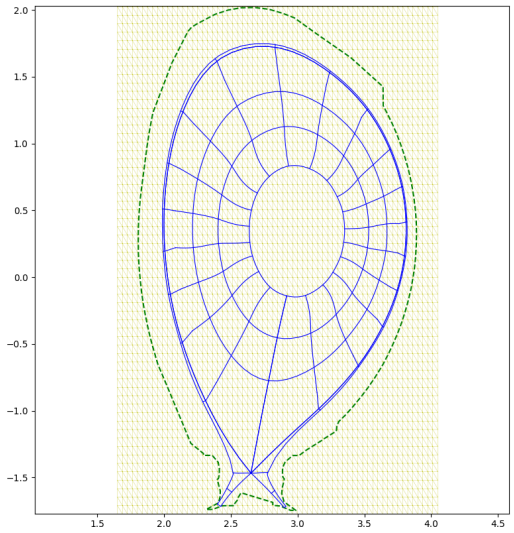


Figure 33: $\mathscr{C}^1$ stick of CORE/Edge and SOL grids.



(a) Partially inside vaccum vessel.



(b) Fully inside vaccum vessel.

Figure 34: Full mesh of CORE/Edge/SOL regions.

**Conversion to Hermite-Bézier mesh**

The previous construction uses spline tensor patches defining a global surface. Once this is done, one can convert these spline surfaces into an Hermite-Bézier mesh using an arbitrary number of elements defined by the user. This transformation does not change the surface and therefore the $\mathscr{G}^1$ character of the global ssurface is maintained. In particular, this construction implies that the control points corresponding to mixed derivatives are aligned across the elements corners as seen in figure 0.4.4

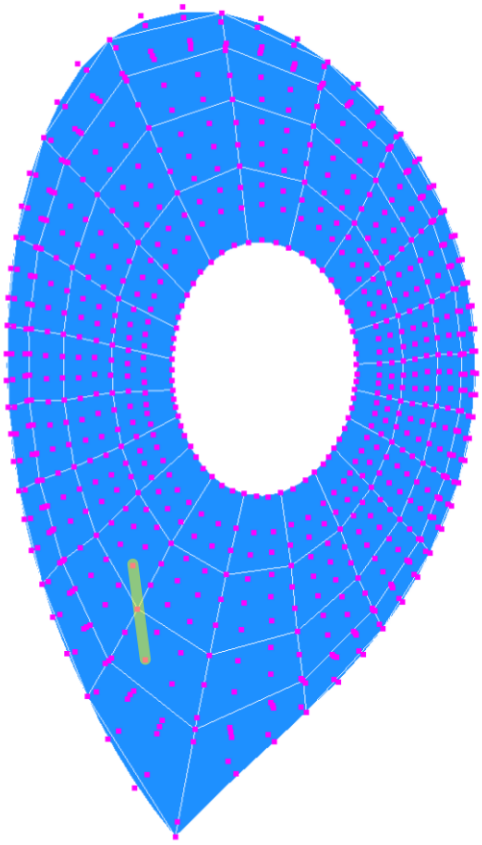


Figure 35: Hermite-Bézier mesh (for CORE region) with control points of mixed derivarives.

# Part III

# User's manual

## 0.5 Installation

The tokamesh library uses a few external packages written in C and Fortran as well as a large number of modules written in Python. To use the tokamesh Library, you first need to install Python on your machine, and these few external packages. The following describes steps by steps this installation.

### 0.5.1 Installing Python

Python is installed by default on all major distributions. If this is not the case on your machine proceed as follows, otherwise go to step 0.5.2

- On Linux

  ```
  sudo apt-get install python
  ```

- On Fedora

  ```
  sudo dnf install python
  ```

- On OSX
  First you have to install xcode and homebrew:

  ```
  xcode-select --install
  /usr/bin/ruby -e "$(curl -fsSL
  https://raw.githubusercontent.com/Homebrew/install/master/install)"
  export PATH="/usr/local/bin:/usr/local/sbin:$PATH"
  ```

  Then you can use homebrew to install python and the package easily

  ```
  brew install python@2
  export PATH="/usr/local/opt/python@2/libexec/bin:$PATH"
  ```

### 0.5.2 Package installation

Installing python also install automatically pip, the python package manager. Except the last one (IgaKit) the following packages are usually available by default. If they are not present on your machine use pip to install them.

- **Numpy :** the fundamental package for scientific computing with Python.

  ```
  pip install Numpy
  ```

- **Scipy :** a Python-based ecosystem of open-source software for mathematics, science, and engineering.

  ```
  pip install Scipy
  ```

- **Matplotlib :** a python 2D plotting library

```
pip install Matplotlib
```

- **IgaKit :** a toolkit for IsoGeometric Analysis that implements many of the NURBS routines from Piegl's book using Fortran and Python.

  ```
  pip install https://bitbucket.org/dalcinl/igakit/get/default.tar.gz
  ```

### 0.5.3 Installing the library

**Getting the source code**

The source code is located on the Inria GitLab platform. Thus first create an Inria Gitlab account as an external user. Then search for the EoCoE/tokamesh project and as External user you will be allowed to download the project, have access to the documentation and the error tracking information. You can clone the tokamesh project using:

```
git clone git@gitlab.inria.fr:EoCoE_Mesh/tokamesh.git
cd tokamesh
```

Or download it form the project page on the git platform : `https://gitlab.inria.fr/users/sign_in`.

**Customization**

Next, define a custom Path Variable enabling your system to locate the the source code :

```
Add the following line to your *.bashrc*, *.bash_profile* or *.profile* file:
export TOKAMESH_DIR=MY_SOURCE_CODE_PATH
where MY_SOURCE_CODE_PATH is the path where you have clone the project.
```

**Pyamg**

After downloading the tokamesh library, you will find in the tokamesh folder an 'external' folder, which contains the archive of pyamg an external package for mesh generation. To install pyamg : Select the compressed folder according to your Operating System, unzip it and install pyamg :

- For Linux:

  ```
  cp pyamg-inria.linux64bits.tgz $MY_INSTALL_PATH

  cd $MY_INSTALL_PATH

  tar zxvf  pyamg-inria.linux64bits.tgz
  ```

- For macOS:

  ```
  cp pyamg-inria.macosx.tgz $MY_INSTALL_PATH

  cd $MY_INSTALL_PATH

  tar zxvf  pyamg-inria.macosx.tgz
  ```

where MY_INSTALL_PATH is the path where you want the PyAMG package to be installed. You can use for example: " $TOKAMESH_DIR/usr " (after creating usr folder). Finally, add the following line to your *.bashrc*, *.bash_profile* or *.profile* file:

```
export PYTHONPATH=$PYTHONPATH:MY_INSTALL_PATH

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:MY_INSTALL_PATH/pyamg/pyamg/dep
```

**Installation:**

The last step installs the library : open a terminal, go to the main folder of the library , and type in one of the following command line:

- Standard mode :

  ```
  python -m pip install .
  ```

- Development mode :

  ```
  python -m pip install --user -e .
  ```

**Uninstall:**

To uninstall the library, in a terminal, go to the main folder of the library, and type in :

```
python -m pip uninstall tokamesh
```

## 0.6 Organization of the software

tokamesh relies mainly on two different tools. The first one is Pyamg, a generic software developed by Adrien Loseille at Inria and built upon Feflo.a. Feflo.a is an robust anisotropic local remeshing software written in C that can handle highly anisotropic mesh adaptation (anisotropy ratio of more that $\mathcal{O}(1:1000)$. Pyamg provides a python interface with Feflo.a.
The second is IgaKit written by Lisandro Dalcin at KAUST that includes many algorithms to create and handle spline and NURBS curves and surfaces. Beside these basic tools, tokamesh is composed of a collection of python modules representing approximately 12k lines of codes. In order to ease the construction of meshes and grids, tokamesh have been organized in different subsets contained in distinct directories.

The following sections give a presentation of the organization of the library. Its main parts are:

- doc
- data
- examples
- externals
- tokamesh


Figure 36: Tokamesh library main folders

### 0.6.1 Doc

This folders contains the documentation about the library and includes the present report. The auto-documentation of the code uses sphinx (http://www.sphinx-doc.org/en/master/). The subfolder sphinx present in the doc folder contains the html code documentation generated by sphinx using the doc_string of individual python modules and functions.

### 0.6.2 Data

In tokamesh, the inputs to the python scripts are gathered in a folder data/inputs that contains subfolders. Each subfolder corresponds to the definition of a different case. As can be seen in figure 37, data/inputs already contains several examples corresponding to different tokamaks and cases. To treat a new problem, the user then must create a new folder in data/inputs.

Then loading a problem can be done with the single instruction line :

```
filename="data/inputs/*/.../name_of_case"
data=load_data(filename)
```

provided the last folder contains three files respectively called name_of_case.mesh, name_of_case.sol and name_of_case.vac whose definition is as follows :

- a .mesh file containing the list of points ( by coordinates), edges (doublet of points) and elements of the mesh
- a .sol file containing the list of,value of a function Z(x,y) on each vertices of the mesh
- a .vac file containing the list of coordinates of the vertices describing the vacuum chamber boundary. In this file, each point must be unique.

The .mesh and .sol files are in Gamma Mesh Format (GMF) (see https://team.inria.fr/gamma3/gamma-software/) with the following header format:

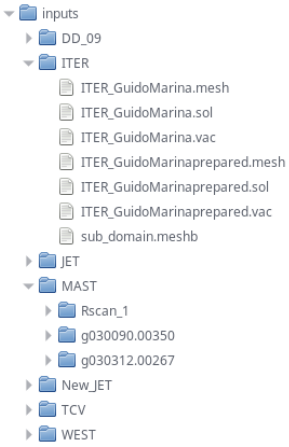



Figure 37: data folder

```
MeshVersionFormatted
2
Dimension
2
```

and the 'End' keyword at the bottom end of the file. Note that to use the tokamesh library, you do not need to download the LibMesh library, just to adhere for the definition of the .mesh and .sol files to the convention used in the LibMesh library.
The . vac is specific to this library. Here is an example of .vac file:

```
        24
  1.26500000   0.00000000        0
  1.26500000   1.08500000        0
  1.57150000   1.39240000        0
  1.57800000   1.32000000        0
  1.59300000   1.15300000        0
  1.62600000   1.09000000        0
  2.00600000   0.77300000        0
  2.23300000   0.44400000        0
  2.23500000   0.36900000        0
  2.26300000   0.31000000        0
  2.29800000   0.18900000        0
  2.31600000   0.06200000        0
  2.31600000  -0.06200000        0
  2.29800000  -0.18900000        0
  2.26300000  -0.31000000        0
  2.23500000  -0.36900000        0
  2.23300000  -0.44400000        0
  2.00600000  -0.77300000        0
  1.62600000  -1.09000000        0
  1.59300000  -1.15300000        0
  1.57800000  -1.32000000        0
```

```
   1.57150000  -1.39240000        0
   1.26500000  -1.08500000        0
   1.26500000   0.00000000        0
 End
```

The first line indicate the number of vertices defining the boundary of the vacuum chamber. The following lines gives the coordinates (two reals) of the vertices defining the boundary plus an additional integer flag that the user can use to distinguish different parts of the boundary if needed.
The list of coordinates present in a .vac have to be organized in such a way the vertices are consecutive : the segment $a_i$ and $a_{i+1}$ is assumed to be a part of the vacuum chamber boundary as well as segment $a_n$ and $a_1$ (closed curve).
It is assumed that the coordinates in a .vac file are unique and that there is no repeated nodes. In order to ensure that this is the case, the vacuum_utils module contains a python script : check_vac_bdry_double allowing to verify this point.

Note also that if the data corresponding to a case is in .eqdsk format, the EQDSKtoMesh Fortran tool allows you to generate the .mesh, .sol and .vac files from it. This tool can be found in the *topology* folder.

### 0.6.3 External

In this folder are stored the binaries of the external tools used by the tokamesh library : Pyamg, EqdskToMesh and transmesh. For more information about these tools, the first two have been previously described, the last one is described in section 0.7.4

### 0.6.4 tokamesh

The tokamesh folder contains the core of the library, the different subfolders contained in *tokamesh* are :

- preprocess : tools for initial data analysis (meshes and solutions),
- bsplines : functions for b-splines generation
- meshgen : functions for mesh generation
- hbezier : functions for Hermite-Bézier splines generation
- utilities : various functions used by other tools

### 0.6.5 Examples

A small set of examples showing how to use the different part of the library. For instance, the analysis and segmentation of a problem named “pb” is demonstrated by running the python script

```
python nameoffile.py
```

in the corresponding folder “pb”. It is advisable to do not erase the folders present in Examples as they are used to provide reference results.

## 0.7 Pre-processing tools

Various preprocessing tools are included in the library, which you can find in the tokamesh/preprocess folder of it. They can be divided in two parts: data analysis and preparation, tools related to the definition of the vacuum chamber as well as function for domain decomposition. We present here the main functions, which use several smaller routine , for more information you can check the documentation generated with sphinx in the doc folder .

### 0.7.1 Data analysis

**load_data**

The first par of an usual Tokamesh routine is to check if the given input data are conform. The *load_data(filename)* routine contained in the *load_data.py* file, read the input data and return them in a data format. If they are not conform to the format it return an error.
Example:

```
filename="/data/inputs/WEST/west_gmf"
data=load_data(filename)
mesh=data[0]
z=data[1]
vac_nodes= data[2]
```

Here we load the data contained in west_gmf.mesh, west_gmf.sol and west_gmf.vac and stack them in *data* variable.

**data_preparation**

The data can be refined using the *data_preparation* function ( located in the *preparation.py* module. This function works as follow: First the critical points of the magnetic flux function are identified, then the x-points that are present inside the vacuum chamber are detected. This step take into account that some data sets use meshes that cover not only the vacuum vessel but also a large part of the tokamak. At present, since tokamesh have been designed for x-point configurations, if there are no saddle point inside the vacuum chamber, the function returns an error and stops. Then the mesh around the saddle points which are inside the vacuum chamber is refined, and finally the solution is interpolated on the new refined mesh using Clough-Tocher $\mathscr{C}^1$ interpolation. This phase of preparation of the data has been found necessary in many cases to obtain better and cleaner data around the saddle points.

Here is an example of call:

```
mesh_r,z_r=data_preparation(mesh,z,vac_nodes)
""" writing the refined . mesh and .sol """
filename_r=filename+'prepared'
write_mesh_sol_from_tri_mesh(filename_r,mesh_r,z_r)
WriteVacuum(filename_r,vac_nodes)
```

It return a new mesh, and new solution. Here we write down the new mesh, solution and vacuum border in new files, by using two routines: *write_mesh_sol_from_tri_mesh* and *WriteVacuum*

### 0.7.2 Vacuum chamber tools

Being able to describe the vacuum chamber and to know whether we are inside or outside of it is an important part of the Tokamesh library functions. You will find described here some of the useful routines for it.

#### is_in_vaccum_pt

For a random point of the 2d space, says if it is inside the vacuum chamber
Arguments :

- pnt : Pt2D a 2D set of coordinates (x,y)
- bdry_list : an ordered list of 2D point defining the vacuum vessel.

Return : true if i is inside the vacuum, false otherwise.

#### interpolate_on_mesh

Read a function defined on a triangulation and return the value of the function on a list of points inside the triangulation . It can be used to define the value of the function on the points defining the vacuum chamber border if they aren't triangulation nodes, or to calculate the value at newly inserted points. Arguments:

- pt : list of mesh nodes
- el : list of mesh triangles (triplets of int)
- z : values of a function on the point pt
- pt_int : list of points where we want to interpolate the function
- kind : interpolation method possible values : linear or cubic

Return: the list of value of z on pt_int points given by interpolation

#### curve_refinement

In some data sets, the vacuum chamber is defined by a very coarse discretization containing very large as well as very small edges. This situation might be problematic for the generation of a new mesh using the vacuum chamber as external boundary. To solve this problem, the *curve_refinement* function compute a new distribution of nodes on the vacuum vessel boundary. Given an ordered list of points coordinates (in the Pt2D format), this function returns a new list of points in the same format, containing the old list plus new points inserted so that the edges of the boundary are of a regular size. Figure 38 show an example on the JET vacuum chamber .

### 0.7.3 Segmentation

The function segmentation present in the module *preprocess/segmentation.py* create a new mesh containing only the vacuum chamber boundary and the isolines passing through the saddle points present in the vacuum vessel. In addition, these isolines are sampled uniformly and the function also returns a list of critical points present in the vacuum vessel. This is necessary to handle the case where the data extends far beyond the vacuum vessel and contains also critical

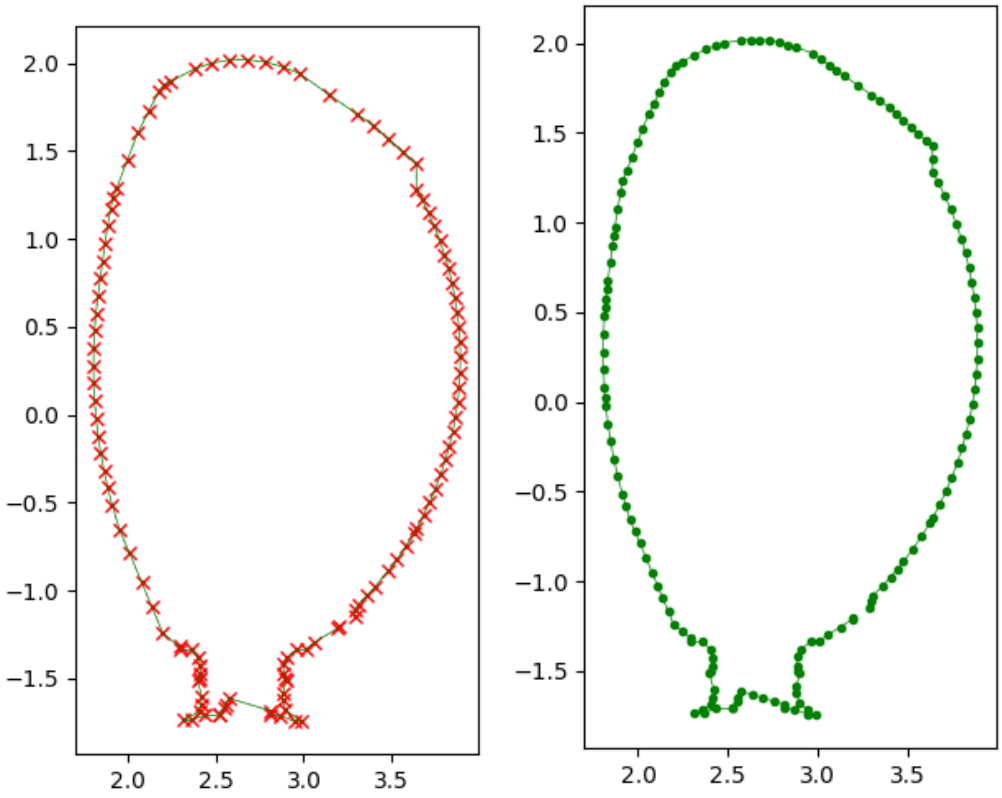



Figure 38: left : initial JET border, right : refined JET border.

points related to the presence of external coils. This function has to be called to generate the different subdomains that will be meshed by the different meshing functions. Here is an example of call:

```
sampling_coeff = 0.1
min_pt = 8

""" compute the domain segmentation """
seg_mesh, seg_sol,x_pt_i,o_pt = Segmentation(mesh,z,vac_nodes,\\
sampling_coeff , min_pt )
```

Where sampling_coeff and min_pt are parameters used for an uniform sampling of the isolines.

First it detect the critical points inside the vacuum chamber, and store those who are inside the vacuum chamber in two list : one for the min/max and one for the saddle points :

```
""" calculate critical points in mesh"""
cp= CriticalPoints(pt,el,z)

""" get the x-points and extremal points inside the vaccum chamber """
x_pt_i=[] # list of x points
```

```
M_pt_i =[] # list of extrema
x_index_i=[]

for c in cp:
pnt = pt[c.index]
    test = is_in_vaccum_pt(pnt,vac_nodes,verbose=0)
    if  test == True:
        if (c.kind == 's'):
         x_pt_i.append(c)
            x_index_i.append(c.index)
            #print('pt ', c.index,'is in vaccum')
        else :
            M_pt_i.append(c)
```

The calculation of the critical points is done using the *CriticalPoints* routine, which can be found in the preprocess/morse_lib.py file . To know if one of critical points is inside the vacuum chamber, we use the *is_in_vaccum_pt* routine which given a contour described by a series of ordered points can return if a random point is inside it or not . This routine can be found in the preprocess/vaccum_utils file.

Then for each saddle point, it generate the isoline going through it, restricted to the vacuum chamber border. For this we used the routine *Gen_iso_from_ver* from the file preprocess/ret_iso. From a node of the mesh, this routine generate the isoline corresponding to the value of the solution field at this point. An isoline is defined by two list:

- isoline containing the coordinates of the points describing it
- isolineedge containing doublet of index (called "edges") , which describe how those points are connected to each other. If the isoline is split in several part by the border of the vacuum chamber, the edges are stored in sublist inside isolineedge corresponding to each branch.

Example: in the figure 39, each color correspond to a different "branch" of an isoline .

If two saddle point have a value different from less than 0.00001 of the amplitude of the flux function, we decide that they form a double null configuration, and fuse their isoline, using the *glue_isolines* routine.

```
center_pt = M_pt_i[0] # the magnetic axis
""" construct the isolines passing trough the x-points """
list_iso_saddle = []
list_iso_sad_edge = []
for idx in range (len(x_pt_i)):
node = x_pt_i[idx].index
    iso_val = z[node]
    isoline, isolineedge, l_newbdry = Gen_iso_from_ver(node,mesh, z,\\
    iso_val, new_bdry_pt)
    new_bdry_pt = l_newbdry
    list_iso_saddle.append(isoline)
    list_iso_sad_edge.append(isolineedge)
    print("the isoline passing through x point of index",node)
    print("has", len(isolineedge), "separated branches")
```

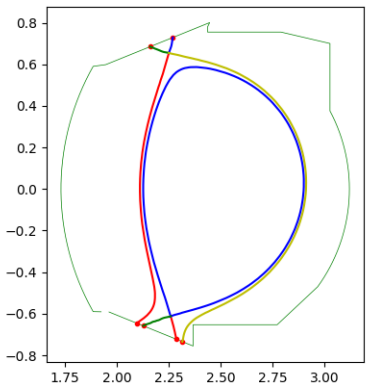



Figure 39: Isolines branches as generated by Segmentation

```
    print("-------------------------------------------------------------")
z_min=np.amin(z)
z_max=np.amax(z)
delta_z=z_max-z_min
x_value=[c.valeur for c in x_pt_i]
if len(x_value) >1 :
    if abs(x_value[0]-x_value[1]) < 1.e-5*delta_z :
        list_iso_saddle,list_iso_sad_edge=glue_isolines(list_iso_saddle,\\
         list_iso_sad_edge)
```

Using those isolines, the border and the center point ( corresponding to the absolute min/max in the vacuum chamber), we call the routine *get_void_mesh* to generate a triangular mesh, with several ref corresponding to the different branch of the Reeb graph of our problem, as in figure 40

```
"""creating a mesh structured around the vaccum border + the saddle isolines"""

seg_mesh, seg_sol = get_void_mesh(l_nodes,l_edges,center_pt)

flag_t=[t[3] for t in seg_mesh['triangles']]
flag_t=list(set(flag_t))
#    print (len(flag_t), flag_t)
"""renumerotation: we want sub domain with ref who are contiguous"""
for t in seg_mesh['triangles']:
for i in range(len(flag_t)):
     if (t[3] == flag_t[i] ):
           t[3] = i+1
new_flag_t=[t[3] for t in seg_mesh['triangles']]
new_flag_t=list(set(new_flag_t))
```

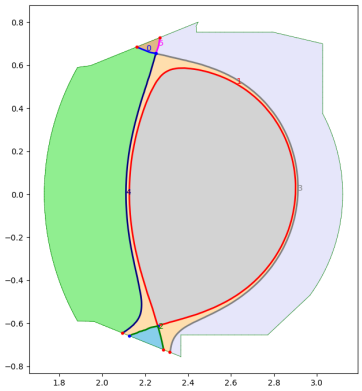



Figure 40: Basic mesh generated by Segmentation

### 0.7.4 External tools

**Pyamg**

Pyamg is a python tool developed by Adrien Loseille. It is used to generate mesh from a list of points and edges.

**pyamg.adapt_mesh :** This function generate a mesh in the form of a python dictionary which contains its vertices, edges, triangles and corners. That mesh is in a compatible format with the libmesh data format.
Arguments :

- mesh : a python dictionary consisting of :
  - "xy" , a list of doublets corresponding to the 2D coordinates of the mesh vertices
  - "edges", a list of doublets corresponding to the edges between vertices we want the generated mesh to have : "edges" = [..., [ 3, 5],... ] force the pyamg.adapt_mesh routine to generate a mesh where vertices 3 and 5 are linked by an edge.
  - "corner", a list of vertices index which indicate the mandatory vertices of the mesh. Which mean that any mesh optimization or adaptation routines will be forced to keep those points.
- remesh_options

Output:
pymsh : a mesh in pyamg mesh data storage format

**pyamg.write_mesh:** This function write down a mesh in a .mesh(b) file format .
Arguments :

- mshtab : a mesh in pyamg mesh data storage format.
- filename : the name of the file where it is going to be written, including the filetype (.mesh or .meshb)

#### Transmesh

This tool developed by Loic Maréchal allows you to transform libmesh binary format to ASCII format and the other way round. it works as follow :

```
./transmesh source_name destination_name
```

#### EqdskToMesh

This Fortran program transforms .eqdsk files into mesh files. From one Eqdsk file it produces three files:

- a .mesh file containing the list of points ( by coordinates), edges and elements of the mesh
- a .sol file containing the list of,value of a function Z(x,y) on each vertices of the mesh
- a .vac file containing the list of coordinates of the vertices describing the vacuum chamber boundary.

To compile it use the following line :

```
gfortran EqdskToMesh.f90 -o EqdskToMesh
```

To use it on a case, named **mycase**.
In the same repository where the EqdskToMesh programm is, create a folder named "mycase" :

```
mkdir mycase
```

run the program on your case :

```
./EqdskToMesh mycase
```

### 0.7.5 Mesh generation

#### patch_connectivity

The function patch_connectivity present in the module meshgen/patch_definition takes as argument the mesh resulting from running the segmentation function. It returns the connectivity between the different subdomains (or patches) and the definition of the curves separating the different patches either as a list of points or as cubic spline curves.

#### Triangular unstructured meshes

An example script showing the generation of the triangular mesh shown in figure 20 can be found in the folder Examples.

#### Block-structured $\mathscr{C}^1$ meshes

An example script showing the generation of the meshes shown in figure 34 can be found in the folder Examples.

**Acknowledgments**

This work has been realized in the framework of the EoCoE project, Energy Oriented Center of Excellence - http://www.eocoe.eu/. In particular Alexis Loyer and Jalal Lakhlili have been directly supported by this action. Ali Elarif has been supported by a PhD grant from Inria. The authors also acknowledge the support of Inria through the Inria IPL program FRATRES https://team.inria.fr/ipl-fratres/

# Bibliography


[1] Peter Alfeld and Larry L. Schumaker. Smooth macro-elements based on clough-tocher triangle splits. *Numerische Mathematik*, 90(4):597–616, Feb 2002.

[2] J.M. Boisserie, K. Hassan, and M. Bernadou. Sur l'implementation des éléments finis de hseieh-clough-tocher complet et reduit. *HAL*, May 2006.

[3] H. Bufferand. *Development of a fluid code for tokamak edge plasma simulation. Investigation on non-local transport*. PhD thesis, Thesis University Aix-Marseille, 2012.

[4] P.G. Ciarlet. Interpolation error estimates for the reduced hsieh-clough-tocher triangle. *Math. Comp.*, 32:335–344, 04 1978.

[5] J.L. Clough, R.W.and Tocher. Finite element stiffness matrices for analysis of plates in bending. In *Int. Conf. on Matrix Methods in Structural Mechanics*, pages 515–545, Wright Patterson Air Force Base, Ohio, 1965.

[6] Olivier Czarny and Guido Huysmans. Bézier surfaces and finite elements for mhd simulations. *Journal of Computational Physics*, 227(16):7423 – 7445, 2008.

[7] C. DeBoor. *A practical guide to splines*. Springer-Verlag, New York, applied mathematical sciences 27 edition, 2001.

[8] M. Fitzgerald, L.C. Appel, and M.J. Hole. Efit tokamak equilibria with toroidal flow and anisotropic pressure using the two-temperature guiding-centre plasma. *Nuclear Fusion*, 53(11):113040, 2013.

[9] P. Frey and P-L. George. *Mesh Generation, 2nd Edition*. Wiley-ISTE, 2008. 848 pages.

[10] Giorgiani G., Camminady T.and Bufferand H., Ciraolo G., Ghendrih P., Guillard, Heumann H. H., Nkonga B., Schwander F., Serre E., , and Tamain P. A new high-order fluid solver for tokamak edge plasma transport simulations based on a magnetic-field independent discretization. *Contributions to Plasma Physics*, 2017.

[11] P. L. George, H. Borouchaki, and E. Saltel. 'ultimate' robustness in meshing an arbitrary polyhedron. *International Journal for Numerical Methods in Engineering*, 58(7):1061–1089, 2003.

[12] P.L. George and H. Borouchaki. Back to edge flips in 3 dimensions. In *International Meshing Roundtable 12*, pages 393–402, Santa Fe, NM, USA, 2003.

[13] G. Greiner, J. Loos, and W. Wesselink. Data dependent thin plate energy and its use in interactive surface modeling. *Computer Graphics Forum*, 15(3):175–185, 1996.

[14] Günther Greiner and Kai Hormann. Interpolating and approximating scattered 3D-data with hierarchical tensor product B-splines. In A. Le Méhauté, C. Rabut, and L. L. Schumaker, editors, *Surface Fitting and Multiresolution Methods*, Innovations in Applied Mathematics, pages 163–172. Vanderbilt University Press, Nashville, TN, 1997.

[15] Ernst Hairer, Syvert P. Norsett, and Gerhard Wanner. *Solving Ordinary Differential Equations I. Nonstiff Problems*. Springer, Berlin, 2nd rev. ed. 1993. corr. 3rd printing edition, 1993.

[16] H. Heumann, J. Blum, C. Boulbe, B. Faugeras, G. Selig, J.-M. Ané, S. Brémond, V. Grandgirard, P. Hertout, and E. Nardon. Quasi-static free-boundary equilibrium of toroidal plasma with cedres++. *Journal of Plasma Physics*, 2015.

[17] Qi-Xing Huang, Shi-Min Hu, and Ralph R. Martin. Fast degree elevation and knot insertion for b-spline curves. *Computer Aided Geometric Design*, 22(2):183 – 197, 2005.

[18] S.C. Jardin. A triangular finite element with first-derivative continuity applied to fusion mhd applications. *Journal of Computational Physics*, 200(1):133 – 152, 2004.

[19] Milnor John. *Morse Theory*. Princeton University Press, 1963.

[20] W. Tiller L. Piegl. *The NURBS Book*. Springer-Verlag, Berlin, Heidelberg, 1995. second ed.

[21] M.J. Lai and L.L. Schumaker. *Spline Functions on Triangulations*. Cambridge University Press, 2007.

[22] A. Loseille. Chapter 10 - unstructured mesh generation and adaptation. In Rémi Abgrall and Chi-Wang Shu, editors, *Handbook of Numerical Methods for Hyperbolic Problems*, volume 18 of *Handbook of Numerical Analysis*, pages 263 – 302. Elsevier, 2017.

[23] Tian Ma and Shouhong Wang. Structural classification and stability of divergence-free vector field. *Physica D: Nonlinear Phenomena*, pages 107– 126, 2002.

[24] R. Marchand and M. Dumberry. Carre: a quasi-orthogonal mesh generator for 2d edge plasma modelling. *Computer Physics Communications*, pages 232 –246, 1996.

[25] Morse. Marston. *The Calculus of Variations in the Large*. American Mathematical Society Colloquium Publication, 1934.

[26] Knut Mørken, Martin Reimers, and Christian Schulz. Computing intersections of planar spline curves using knot insertion. *Computer Aided Geometric Design*, 26(3):351 – 366, 2009.

[27] Helmut Pottmann and Stefan Leopoldseder. A concept for parametric surface fitting which avoids the parametrization problem. *Computer Aided Geometric Design*, 20(6):343 – 362, 2003.

[28] Georges Reeb. Sur les points singuliers d'une forme de pfaff complètement intégrable ou d'une fonction numérique. *C. R. Acad. Sci. Paris*, pages 847 – 849, 1946.

[29] D. Reiter, M. Baelmans, and P. Börner. The eirene and b2-eirene codes. *Fusion Science and Technology*, pages 172–186, 2005.

[30] V.D Shafranov. Plasma equilibrium in a magnetic field. *Reviews of Plasma Physics*, 1966.

[31] Takahashi Shigeo, Ikeda Tetsuya, Shinagawa Yoshihisa, Kunii Tosiyasu L., and Ueda Minoru. Algorithms for extracting correct critical points and constructing topological graphs from discrete geographical elevation data. *Computer Graphics Forum*, 1995.

[32] Hang Si. Tetgen, a delaunay-based quality tetrahedral mesh generator. *ACM Trans. Math. Softw.*, 41(2):11:1–11:36, February 2015.

[33] C.R. Sovinec, A.H. Glasser, T.A. Gianakon, D.C. Barnes, R.A. Nebel, S.E. Kruger, S.J. Plimpton, A. Tarditi, M.S. Chu, and the NIMROD Team. Nonlinear magnetohydrodynamics with high-order finite elements. *J. Comp. Phys.*, 195:355, 2004.

[34] Fan Zhang, Robert Hager, Seung-Hoe Ku, Choong-Seock Chang, Stephen C. Jardin, Nathaniel M. Ferraro, E. Seegyoung Seol, Eisung Yoon, and Mark S. Shephard. Mesh generation for confined fusion plasma simulation. *Eng. with Comput.*, 32(2):285–293, April 2016.