Available online at www.sciencedirect.com

ScienceDirect

IFAC PapersOnLine 55-16 (2022) 13–18

# ct: control toolbox – Numerical tools and examples in optimal control [★]

Jean-Baptiste Caillau [*] Olivier Cots [**] Pierre Martinon [***]

[*] *Université Côte d'Azur, CNRS, Inria, LJAD (e-mail:* `jean-baptiste.caillau@univ-cotedazur.fr`*).*
[**] *Université Toulouse, CNRS, ENSEEIHT-IRIT (e-mail:* `olivier.cots@irit.fr`*)*
[***] *CAGE team, Inria Paris (e-mail:* `pierre.martinon@inria.fr`*)*

**Abstract:** Combining direct and indirect methods to have the best of both worlds is an efficient method to solve numerically optimal control problems. A direct solver will typically provide information on the structure of the optimal control, allowing an educated guess for indirect shooting. The control toolbox `ct` offers such possibilities and is presented on two examples. The first example has a bang-singular solution and is solved by chaining direct and indirect solvers. The second one consists in computing conjugate and cut loci on an ellipsoid of revolution, which is performed using a more advanced combination of indirect methods with differential continuation.

*Keywords:* direct transcription method, shooting, bocop, nutopy, differential continuation, automatic differentiation

## 1. INTRODUCTION

Methods for solving optimal control problems with ordinary differential equations essentially fall into three main categories. Dynamic Programming (or Hamilton-Jacobi-Bellman method; see, *e.g.*, ROC-HJ [1] for an implementation) computes the global optimum but may suffer from high computational costs, the so-called *curse of dimensionality*. Indirect methods based on Pontrjagin Maximum Principle are extremely fast and accurate but often require more work to be applied, in terms of mathematical analysis and *a priori* knowledge of the structure of the solution. Direct transcription methods offer a good tradeoff between robustness and accuracy and are widely used for industrial applications. There are excellent available direct solvers, usually making an intensive use of automatic differentiation (see, *e.g.*, CasADI [2]). For challenging problems, an effective strategy is to start with a direct method to find a first rough solution, then refine it through an indirect method. Such a combined approach is very commonly used; see, *e.g.*, this paper Bonnard et al. (2015) for the optimization of contrast in medical imaging (MRI). This combination of direct and indirect methods has a lot of interest to solve optimal control problems that contain state or control constraints. In the example mentioned above, the interfacing between the two solvers Bocop [3] and `HamPath` [4] were done manually by *ad hoc* `python` or `matlab` layers. The aim of the current paper is to present a common project between several Inria and CNRS research teams from Sophia, Paris and Toulouse whose goal is to interoperate these solvers using a high level common interface. This project, coined `ct: control toolbox`, [5] benefits from the support of Inria through a so-called *Action de Développement Technologique*. It is open source and welcomes external contributions.

Two examples are presented. First, a regulator example with constraints on the control; though very simple, the solution has a bang-singular structure that illustrates in an elementary fashion the interplay between direct (to capture the structure) and indirect (to compute efficiently and precisely) methods. It is solved using the new `python` interface of Bocop3 [6] and the `python` package `nutopy`. [7] The second example is taken from Riemannian geometry and more sophisticated. It leverages the use of differential continuation (homotopy methods) to compute conjugate and cut loci on an ellipsoid of revolution. In both examples, automatic differentiation is crucial (and used up to order three in the second one). The presented results are entirely reproducible online [8] without any installation, using the Binder [9] technology.

## 2. FIRST EXAMPLE: A SIMPLE REGULATOR PROBLEM

### 2.1 A simple regulator problem

We illustrate here the coupling of direct and indirect methods in optimal control, on a simple case of Linear Quadratic Regulator in dimension 2. More precisely, we

[★] Supported by InriaHub / ADT ct from Inria Sophia Antipolis-Méditerranée
[1] `itn-sadco.inria.fr/itn-sadco.inria.fr/software/ROC-HJ.html`
[2] `casadi.org`
[3] `bocop.org`
[4] `hampath.org`
[5] `ct.gitlabpages.inria.fr/gallery`
[6] `ct.gitlabpages.inria.fr/bocop3`
[7] `ct.gitlabpages.inria.fr/nutopy`
[8] See the regulator and ellipsoid examples on the `ct` project gallery.
[9] `mybinder.org`

10.1016/j.ifacol.2022.08.074

show how the direct method can provide the relevant information to initialize the indirect method accurately, namely with the correct control structure and with an estimate of junction times as well as state and costate values at the relevant times. We study the following problem, for which the optimal control structure consists in a bang arc followed by a singular arc.

$$\begin{cases} \min \ \dfrac{1}{2}\displaystyle\int_0^5 (x_1^2(t) + x_2^2(t))\,\mathrm{d}t \\ \dot{x}_1(t) = x_2(t),\ \dot{x}_2(t) = u(t),\ u(t) \in [-1, 1] \\ x(0) = (0, 1) \end{cases}$$

### 2.2 Direct method

We first solve the problem with the so-called direct transcription approach, that basically consists in solving the finite-dimensional nonlinear programming problem (NLP) resulting from applying a time discretization to the original optimal control problem (OCP). We reformulate the Lagrange cost as a Mayer cost by introducing a new state variable

$$\dot{x}_3(t) = (x_1^2(t) + x_2^2(t))/2, \quad x_3(5) \to \min$$

and use the software BOCOP for the numerical simulations. The definition of the problem consists in one `C++` file for the problem functions, and one text file for all other parameters and settings. The file `problem.cpp` defines the final cost, dynamics and boundary conditions for the OCP.

```
Variable x1 = state[0];
Variable x2 = state[1];
Variable u = control[0];
state_dynamics[0] = x2;
state_dynamics[1] = u;
state_dynamics[2] = 0.5*(x1*x1+x2*x2);
```

A separared definition file contains all other relevant information, such as problem dimensions, time discretization choices, bounds for the different variables and constraints, initial guess and numerical settings for the NLP solver `Ipopt`, *etc.*

We build the problem executable and solve the problem by launching the optimization.

```python
import bocop
problem_path = "."
bocop.build(problem_path)
bocop.run(problem_path)
```

### 2.3 Solution analysis

In addition to the state and control from the optimal trajectory, we can plot the Lagrange multipliers associated to the constraints for the discretized dynamics, that correspond to the costate variables from Pontrjagin Maximum Principle. Note that for the case of basic initial conditions $x_i(0) = x_i^0$, the multipliers for these constraints will also match the costate variables at the initial time for the corresponding state variables. For this problem the third costate is constant and equal to $-1$, which is normal since the third state corresponds to the integral cost to be minimized. The solution exhibits a bang-singular structure, as can be clearly seen from Fig. 1 on the control variable in purple. The oscillations of the control over the singular arc are often encountered when using discrete transcription, however the averaged control usually corresponds to the correct singular control, meaning that the state dynamics should be close to the optimal trajectory.

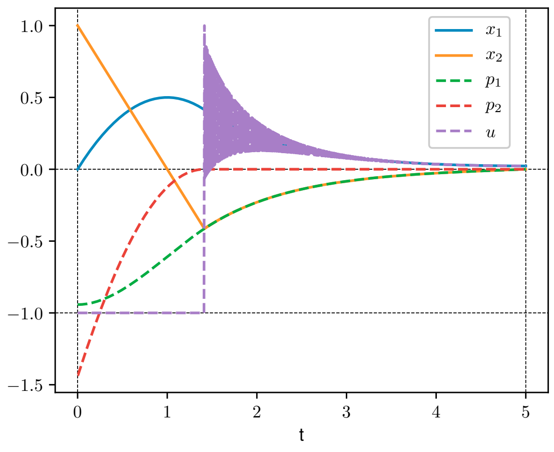

Fig. 1. States, costates and control computed by BOCOP. The bang-singular structure of the control is clearly observed.

Here is the output of BOCOP:

```
Bocop returns status 0 with objective 0.377
and constraint violation 1.221e-12

Multipliers for initial conditions:
[-0.94216793 -1.44190126 -1. ]
```

### 2.4 Indirect method

Using the information gathered on the solution from the direct method, we can now solve the problem accurately with an indirect method, with the knowledge of both the optimal control structure and an estimate of the junction time and state-costate values at this time. We use the package `nutopy` for the indirect shooting method. Applying Pontrjagin maximum principle, one has to consider the following Hamiltonian (in normal form since there are no terminal constraints):

$$H(x_1, x_2, p_1, p_2, u) = -(x_1^2 + x_2^2)/2 + p_1 x_2 + p_2 u.$$

One checks that the control is either bang (equal to $\pm 1$) or singular. Singular arcs are actually of order one, given by the condition $p_2 = 0$ (singular arcs actually leave on the codimension two submanifold $\{p_2 = x_2 - p_1 = 0\}$ of the cotangent bundle $T\mathbf{R}^2 \simeq \mathbf{R}^2 \times \mathbf{R}^2$, and the singular control (in feedback form) is $u = x_1$. So we have a competition between three Hamiltonian flows, respectively associated with

$$H_\pm := H(x, p, \pm 1),\ H_s := H(x, p, x_1).$$

We define these three Hamiltonians and the corresponding flows in order to set up our shooting algorithm. (For this particular example, in view of the BOCOP run, we restrict to just $-1$ bang arcs.)

```
import nutopy as nt
import numpy as np
t0   = 0.
tf   = 5.
x0   = np.array([0., 1.])
```

First, we define the (OCP) setting the *running cost*, that is the integrand of the Lagrange cost, and the dynamics.

```
# Running cost
def df0fun(t, x, dx, u, du):
    df0 = x[0]*dx[0] + x[1]*dx[1]
    return df0

def d2f0fun(t, x, dx, d2x, u, du, d2u):
    d2f0 = dx[0]*d2x[0]+dx[1]*d2x[1]
    return d2f0

@nt.tools.
↪tensorize(df0fun,d2f0fun,tvars=(2,3))
def f0fun(t, x, u):
    f0 = 0.5 * (x[0]**2 + x[1]**2)
    return f0

# Dynamics
def dffun(t, x, dx, u, du):
    df = np.zeros(2)
    df[0] = dx[1]
    df[1] = du
    return df

def d2ffun(t, x, dx, d2x, u, du, d2u):
    d2f = np.zeros(2)
    return d2f

@nt.tools.tensorize(dffun,d2ffun,tvars=(2,3))
def ffun(t, x, u):
    f = np.zeros(2)
    f[0] = x[1]
    f[1] = u
    return f

o = nt.ocp.OCP(f0fun, ffun) # OCP
```

Then, we define the Hamiltonians and flows. For the control laws we do not present the part of the code defining the derivatives.

```
def ubang(t, x, p): # Bang control
    return -1.

def using(t, x, p): # Singular control
    return x[0]

# Hamiltonians and Flows
hbang = nt.ocp.Hamiltonian.fromOCP(o, ubang)
hsing = nt.ocp.Hamiltonian.fromOCP(o, using)

fbang = nt.ocp.Flow(hbang)
fsing = nt.ocp.Flow(hsing)
```

Assuming a bang-singular structure, it is then easy to define a shooting function whose unknowns are the value $p_0$ of the costate at initial time, $(x_1, p_1)$ the state and costate values at the junction point between the $(-1)$ bang arc and the singular one, and the time $t_1$ where this junction occurs. A standard solver, properly initialised thanks to the previous solution provided by BOCOP, is eventually called to solve the problem. As is clear from the obtained numerical results, a very precise solution is computed.

```
# Shooting function
def shoot(z):

    p0 = z[0:2]
    t1 = z[2]
    x1_in = z[3:5]
    p1_in = z[5:7]
    s = np.ones(7)

    # first bang arc
    x1, p1 = fbang(t0, x0, p0, t1)
    s[3:5] = x1 - x1_in # matching conditions
    s[5:7] = p1 - p1_in # matching conditions

    # singular arc
    xf, pf = fsing(t1, x1_in, p1_in, tf)
    s[0] = p1[1]          # p2(t1)
    s[1] = x1[1]-p1[0]  # x2(t1)-p1(t1)
    s[2] = pf[0]          # p1(tf)

    return s
```

```
# Find a zero of the shooting function
# z0 is given by BOCOP
sol = nt.nle.solve(shoot, z0)
```

```
Results of the nle solver method:

zsol =  [-9.42173346e-01 -1.44191018e+00
          1.41376409e+00  4.14399640e-01
         -4.13764088e-01 -4.13764088e-01
                         -1.26506733e-28]

norm(shoot(zsol)) = 5.400314267047623e-16
```

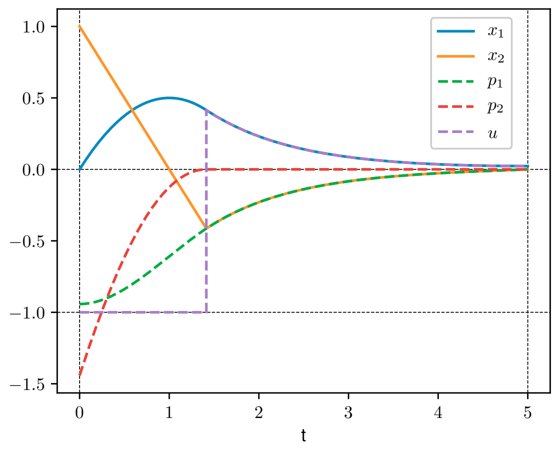

Fig. 2. States, costates and control computed by shooting (nutopy).

## 3. SECOND EXAMPLE: CUT AND CONJUGATE LOCI OF AN ELLIPSOID OF REVOLUTION

### *3.1 Preliminaries*

We recall standard notions of Riemannian geometry that can be found in Berger (2003); Do Carmo (1988). On a connected Riemannian manifold $(M, g)$ of dimension $n$, the *metric* $g$ induces a *distance* function $d(q_0, q_1)$ for any pair $(q_0, q_1) \in M^2$. The distance $d(q_0, q_1)$ being the infimum of lengths of continuously differentiable curves $q$ joining the points $q_0$ and $q_1$. If we denote by $l(q)$ the length of a curve $q$, then on a connected and complete Riemannian manifold, for any pair $(q_0, q_1) \in M^2$, there exists a continuously differentiable curve $q$ such that $d(q_0, q_1) = l(q)$. In this case, the solution curve $q$ being necessarily chosen among the set of the so-called *geodesic curves* (or *geodesics*). The geodesics are so candidates as minimizers and we know that every geodesic is a minimizing curve at least on short distances. Our main goal is to compute for a fixed $q_0 \in M$, the partial distance function $d(q_0, \cdot)$ together with the set of minimizing geodesics, that is we want to find for any point $q_1 \in M$, the distance $d(q_0, q_1)$ and the geodesic $q$ such that $d(q_0, q_1) = l(q)$. To do so, we compute the geodesics, which are projections of *extremals* given by the flow of the *Hamiltonian vector field* $\overrightarrow{H}$ associated to the *Hamiltonian* $H$ given by the *Legendre transform* of the metric $g$. The *cut point* of a geodesic is the point where it ceases to be minimizing. Fixing $q_0 \in M$, the set of cut points of all the geodesics starting from $q_0$ is the *cut locus* denoted $\mathrm{Cut}(q_0)$. Our main goal is thus equivalent to compute $\mathrm{Cut}(q_0)$.

We consider the academic case of an oblate ellipsoid of revolution to illustrate the numerical tools. The theoretical results related to this case may be found in Itoh and Kiyohara (2004). In the Riemannian frame, the cut points may be of two kinds, either *conjugate points* or *isochronous separating points*. A conjugate point is a point where the geodesic $q$ ceases to be optimal among the geodesics $C^1$-close to $q$. The conjugate points are fold points of the geodesic flow and are due to the intrinsic curvature of the Riemannian manifold. This points may be computed by linearization of the flow of $\overrightarrow{H}$, that is computing *Jacobi fields*. An isochronous separating point is a point where two distincts minimizing geodesics intersect, that is it is a self-intersection of a *wavefront* of the geodesic flow. The set of separating points is called the *separating locus*. The rest of the section illustrates how to use the `nutopy` package to compute the cut locus and related objects. See Bonnard et al. (2014); Facca et al. (2021); Itoh and Sinclair (2004); Sinclair and Tanaka (2002) for others methods to compute cut loci in Riemannian geometry.

### *3.2 The Hamiltonian and flow for the oblate ellipsoid of revolution*

The Cartesian equation of the normalized 2d ellipsoid of revolution may be written as

$$x^2 + y^2 + \frac{z^2}{\varepsilon^2} = 1$$

The associated ellipsoid of revolution may be generated by a rotation of axis $Oz$ of the parameterized curve $y = \cos u$, $z = \varepsilon \sin u$, with $u \in [0, 2\pi]$ and where $0 < \varepsilon < 1$ corresponds to the oblate (flattened) case while $\varepsilon > 1$ is the prolate (elongated) case. We parameterize the Cartesian coordinates by the azimuth $\theta \in [-\pi, \pi]$ and the latitude $\varphi \in [-\pi/2, \pi/2]$, by the relations $x = \cos\varphi \cos\theta$, $y = \cos\varphi \sin\theta$ and $z = \varepsilon \sin\varphi$. The restriction of the Euclidean metric of the ambient space $\mathbf{R}^3$ gives the following induced metric on the ellipsoid:

$$g = g_1(\varphi)\, \mathrm{d}\theta^2 + g_2(\varphi)\, \mathrm{d}\varphi^2$$

where $g_1(\varphi) = \cos^2\varphi$ and $g_2(\varphi) = \sin^2\varphi + \varepsilon^2 \cos^2\varphi$. In the following, we consider the oblate case and we fix $\varepsilon = 0.75$ and $q_0 = (0, 0)$. The associated Hamiltonian is

$$H = \frac{1}{2}\left(\frac{p_\theta^2}{g_1(\varphi)} + \frac{p_\varphi^2}{g_2(\varphi)}\right)$$

that we implement in `Fortran`:

```
subroutine hfun(x, p, e, h)

 double precision, intent(in) :: x(2), p(2), e
 double precision, intent(out) :: h

 ! local variables
 double precision :: theta, phi, ptheta, pphi
 double precision ::g1, g2

 theta   = x(1)
 phi     = x(2)
 ptheta  = p(1)
 pphi    = p(2)

 g1 = cos(phi)**2
 g2 = sin(phi)**2+e**2*cos(phi)**2

 h = 0.5d0 * (ptheta**2 / g1 + pphi**2 / g2)

end subroutine hfun
```

From the Hamiltonian $H$, we define the *Hamiltonian vector field* defined on the cotangent bundle $T^*M$ ($M$ being the ellipsoid)

$$\overrightarrow{H}(q,p) \overset{\text{def}}{=} \left(\frac{\partial H}{\partial p}(q,p), -\frac{\partial H}{\partial q}(q,p)\right).$$

We finally define the *Hamiltonian exponential map*, $e^{t\overrightarrow{H}} : T^*M \to T^*M$, by

$$e^{t\overrightarrow{H}}(q_0, p_0) \overset{\text{def}}{=} (q(t, q_0, p_0), p(t, q_0, p_0)),$$

where the *extremal* $z(\cdot, q_0, p_0) = (q(\cdot, q_0, p_0), p(\cdot, q_0, p_0))$ is defined as the maximal solution of the Cauchy problem $\dot z(t) = \overrightarrow{H}(z(t))$, $z(0) = (q_0, p_0)$. Thus, the Hamiltonian exponential map gives us the flow of extremals, associated to the Hamiltonian system.

```
# Hamiltonian
h = nt.ocp.Hamiltonian(hfun)

# Hamiltonian exponential map
extremal = nt.ocp.Flow(h)
```

### *3.3 Geodesics*

By homogeneity, the extremals may be parameterized fixing $H = 1/2$. This amounts to parameterize the geodesics by the arc length. Thus, given an initial point $q_0 \in M$, the initial covector $p_0$ must satisfy $H(q_0,p_0) = 1/2$. One can write that $p_0 \in S^1 = \{p \mid ||p|| = 1\}$, defining the norm $||p||^2 = 2H(q_0,p)$ on $T^*_{q_0}M$. Hence, the initial covector may be parameterized by its angle $\alpha_0 \in [0,2\pi)$. We write

$$p_0 = p(\alpha_0) \overset{\text{def}}{=} \left(\cos\alpha_0\sqrt{g_1(\varphi_0)}, \sin\alpha_0\sqrt{g_2(\varphi_0)}\right)$$

this parameterization, with $p(\alpha_0) \in T^*_{q_0}M$. Finally, a geodesic starting from $q_0$ is the projection of an extremal parameterized by its initial angle $\alpha_0$ and is given by the following classical exponential map:

$$\exp_{q_0}(t,\alpha_0) \overset{\text{def}}{=} \pi_q\left(e^{t\vec{H}}(q_0,p(\alpha_0))\right),$$

where $\pi_q(q,p) = q$ is the canonical projection on the state space.

```
# Initial covector
def covector(q, alpha):
    g1, g2 = metric(q)
    p0 = [np.cos(alpha)*np.sqrt(g1),
          np.sin(alpha)*np.sqrt(g2)]
    return p0

# Geodesic
def geodesic(t, alpha0):
    p0 = covector(q0, alpha0)
    q, p = extremal(t0, q0, p0, t)
    return q
```

## 3.4 Conjugate locus

A point $q(t_c)$ on an extremal $z = (q,p)$ is conjugate to $q_0 = q(0)$ if there exists a *Jacobi field* $\delta z = (\delta q, \delta p)$, solution of the linearized system along the extremal,

$$\delta\dot{z}(t) = \frac{\partial}{\partial z}\vec{H}(z(t)) \cdot \delta z(t),$$

which is non-trivial ($\delta q \not\equiv 0$) and vertical at $t = 0$ and $t_c > 0$ (called the *conjugate time*), that is $\delta q(0) = \delta q(t_c) = 0$. The *conjugate locus* is the set of such first points on extremals departing from $q_0$. Conjugacy is classicaly related to local optimality of extremals in the relevant topologies. Let $q\colon t \mapsto \exp_{q_0}(t,\alpha_0)$ being a reference geodesic and introduce $F_{\text{conj}}\colon \mathbf{R}^*_+ \times [0,2\pi) \to \mathbf{R}$ as

$$F_{\text{conj}}(t,\alpha) \overset{\text{def}}{=} \det\left(\exp'_{q_0}(t,\alpha)\right).$$

Then, in the Riemannian setting, $q(t_c)$ is conjugate to $q_0$ if and only if $F_{\text{conj}}(t_c,\alpha_0) = 0$. The conjugate locus from $q_0$ is thus given by

$$\text{Conj}(q_0) = \bigcup_{\alpha_0 \in [0,2\pi)} \{\exp_{q_0}(t_{1c},\alpha_0) \\ \text{s.t. } F_{\text{conj}}(t_{1c},\alpha_0) = 0\},$$

where $t_{1c}$ has to be understood in the sense that it is the **first** conjugate time. To compute the conjugate locus, we need to compute the first conjugate times, that is to compute

$$\bigcup_{\alpha_0 \in [0,2\pi)} \{(t_{1c},\alpha_0) \mid F_{\text{conj}}(t_{1c},\alpha_0) = 0\}.$$

**Algorithm.** We have to compute a subset of $F^{-1}_{\text{conj}}(\{0\})$. We use the numerical continuation method (or homotopy method) from Allgower and Georg (2003) implemented in the **nutopy** package. Under some regularity assumptions, the set $F^{-1}_{\text{conj}}(\{0\})$ is a disjoint union of differential curves, each curve being called a *path of zeros*. To compute a path of zeros, we search a first point on the curve by fixing the homotopic parameter (here it will be $\alpha_0$) to a certain value $\alpha_0^*$ and then calling a Newton method to solve $F_{\text{conj}}(\cdot,\alpha_0^*) = 0$. The Newton solver in the **nutopy** package is the hybrj code from the minpack library, see Moré et al. (1980). When we have our initial point on the path of zeros, we use a Predictor-Corrector (PC) method with arc length parameterization to compute the differential curve. The **nutopy** package implements a PC method with a high-order Runge-Kutta scheme with adaptive step size for the prediction from Hairer et al. (1993).[10] The correction step is performed by a simplified Newton algorithm from (Hairer and Wanner, 1996, p. 119) with few iterations and where the Jacobian of the associated system to solve is not updated along the iterations. For the prediction and the correction steps, we need to compute the Jacobian of $F_{\text{conj}}$. It is computed by a combination of automatic differentiation (to get $\vec{H}$ and its derivatives up to order 2) performed with the **tapenade** software, see Hascoët and Pascual (2012), and variational equations to compute Jacobi fields and their derivatives with respect to the initial angle $\alpha_0$.

```
# Jacobi field
@nt.tools.vectorize(vvars=(1,))
def jacobi(t, alpha0):
    p0, dp0 = covector(q0, (alpha0,1.))
    (q,dq),(p,dp) = extremal(t0,q0,(p0,dp0),t)
    return (q, dq), (p, dp)
```

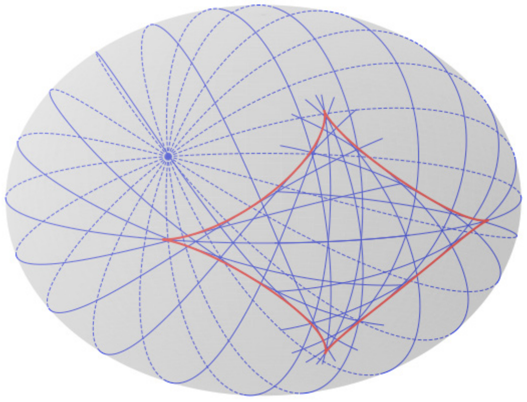
Fig. 3. Geodesics and their envelope when the flow is folding, *i.e.* the conjugate locus (in red).

## 3.5 Cut locus

The cut locus Cut$(q_0)$ is the union of the cut points of the geodesics departing from $q_0$. A cut point is either a conjugate point or an *isochronous separating point*, see (Do Carmo, 1988, Proposition 2.2 p. 267), that is a point where two minimizing geodesics intersect with the same time (or length). A separating point (or *splitting* point) corresponds

[10] The prediction part is the same as in the **HamPath** software, see Caillau et al. (2011).

to a self-intersection of a wavefront. To compute the cut locus we need at this stage, to compute the *separating locus* (or *splitting locus*) and then compare for each geodesic its first conjugate time with its *separating time*. We introduce the following mapping:

$$F_{\text{split}}(t, \alpha_1, q, \alpha_2) \overset{\text{def}}{=} \left(q - \exp_{q_0}(t, \alpha_1),\ q - \exp_{q_0}(t, \alpha_2)\right).$$

The splitting locus is then given by solving $F_{\text{split}} = 0$ since we have

$$\text{Split}(q_0) \subset \{q \in M \mid \exists (t, \alpha_1, \alpha_2) \text{ s.t. } \alpha_1 \neq \alpha_2 \text{ and } F_{\text{split}}(t, \alpha_1, q, \alpha_2) = 0\}.$$

The splitting locus is also computed by homotopy.

```
# Equations to compute Split(q0)
def Fsplit(y, alpha2):

    t      = y[0]
    alpha1 = y[1]
    q      = y[2:4]

    q1, _ = extremal(t0, q0, covector(q0,␣
↪alpha1), t)
    q2, _ = extremal(t0, q0, covector(q0,␣
↪alpha2), t)

    eq = np.zeros(4)
    eq[0:2] = q-q1
    eq[2:4] = q-q2

    return eq
```

At the end, the cut locus of the oblate ellipsoid of revolution is the union of the splitting locus with the two conjugate points at the extremity of Split$(q_0)$, that is

$$\text{Cut}(q_0) = \text{Split}(q_0) \cup \{\exp_{q_0}(t^*, 0), \exp_{q_0}(t^*, \pi)\},$$

where $t^*$ is the *injectivity radius*, the first positive time solution of $F_{\text{conj}}(t^*, 0) = 0$, see Fig. 4 for a view of the cut locus and see Itoh and Kiyohara (2004) for theoretical details.

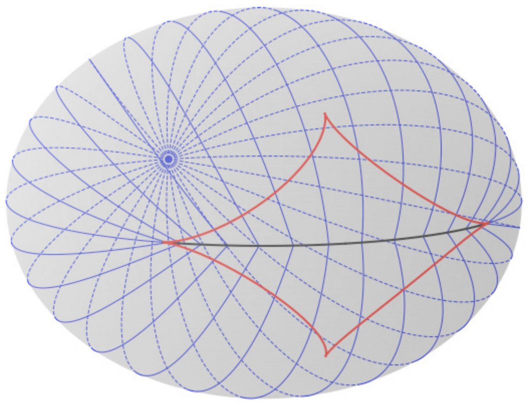

Fig. 4. Geodesics until cut point. Conjugate locus in red. Cut locus in black. It is the segment of the *equator* contained in the conjugate locus. The extremities of the cut locus are cusps of the conjugate locus while interior points are separating points.

ACKNOWLEDGEMENTS

The authors thank the SED[11] team of Inria Sophia for their help and commitment on this project.

REFERENCES

E. Allgower & K. Georg, *Introduction to numerical continuation methods*, vol. 45 of Classics in Applied Mathematics, Soc. for Industrial and Applied Math., Philadelphia, PA, USA, (2003), 388 pages.

M. Berger, *A panoramic view of Riemannian geometry*, Springer, 2003.

B. Bonnard, M. Claeys, O. Cots & P. Martinon, *Geometric and numerical methods in the contrast imaging problem in nuclear magnetic resonance*, Acta Appl. Math., 135 (2015), no. 1, 5-45.

B. Bonnard, O. Cots & L. Jassionnesse, *Geometric and numerical techniques to compute conjugate and cut loci on Riemannian surfaces*, in INDAM Series vol. 5, Geometric Control and sub-Riemannian Geometry, (2014).

J.-B. Caillau, O. Cots & J. Gergaud, *Differential continuation for regular optimal control problems*, Optim. Methods Softw., **27** (2011), no. 2, pp. 177–196.

M. P. Do Carmo, *Riemannian geometry*, Birkhäuser, Mathematics: Theory & applications, second edn 1988.

E. Facca, L. Berti, F. Fassó & M. Putti, *Computing the Cut Locus of a Riemannian Manifold via Optimal Transport*, 2021. ⟨hal-03467888⟩

E. Hairer, S. P. Nørsett & G. Wanner, *Solving Ordinary Differential Equations I, Nonstiff Problems*, vol 8 of Springer Serie in Computational Mathematics, Springer-Verlag, second edn (1993).

E. Hairer & G. Wanner, *Solving Ordinary Differential Equations II, Stiff and Differential-Algebraic Problems*, vol 14 of Springer Serie in Computational Mathematics, Springer-Verlag, second edn (1996).

L. Hascoët & V. Pascual, *The Tapenade Automatic Differentiation tool: principles, model, and specification*, Rapport de recherche RR-7957, INRIA (2012).

J. Itoh & K. Kiyohara, *The cut loci and the conjugate loci on ellipsoids*, Manuscripta math., **114** (2004), no. 2, pp. 247-264.

J. Itoh & R. Sinclair, *Thaw: A Tool for Approximating Cut Loci on a Triangulation of a Surface*, Experiment. Math. **13** (2004), no. 3, 309-325.

J. J. Moré, B. S. Garbow & K. E. Hillstrom, *User Guide for MINPACK-1*, ANL-80-74, Argonne National Laboratory, (1980).

R. Sinclair & M. Tanaka, *Loki: Software for Computing Cut Loci,* Exper. Math. **11** (2002), no. 1, 1–25.

[11] iww.inria.fr/sed-sophia