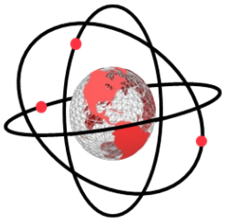

HamPath: On solving optimal control problems by indirect and path following methods.

HamPath 3.0 - User guide

Jean-Baptiste Caillau, Olivier Cots, Joseph Gergaud

Jean Matthieu Khoury

June 16, 2016

Contents

1 Introduction

The `HamPath` package [8] is a an open-source software developed to solve optimal control problems via indirect methods but also to study Hamiltonian systems. `HamPath` is developed since 2009 by members of the APO (Algorithmes Parallèles et Optimisation) team from Institut de Recherche en Informatique de Toulouse, jointly with colleagues from the Université de Bourgogne. `HamPath` is distributed under the EPL license, and is free for both academic and industrial use.

The main use of `HamPath` is to study and solve optimal control problems of the general form:

$$\text{(OCP)} \quad \begin{cases} J(x(\cdot),u(\cdot),t_0,t_f) = g(t_0,x(t_0),t_f,x(t_f),\Lambda) + \displaystyle\int_{t_0}^{t_f} f^0(t,x(t),u(t),\Lambda)\ \mathrm{d}t \longrightarrow \min \\ \dot{x}(t) = f(t,x(t),u(t),\Lambda), \quad u(t)\in U, \quad t\in[t_0\,,t_f] \text{ a.e.}, \\ (t_0,x(t_0),t_f,x(t_f))\in M_b, \\ x(t)\in X_c\subset X, \quad t\in[t_0\,,t_f], \end{cases}$$

where Λ is a set of parameters, X is the state space of dimension n, U is the control domain, M_b is a manifold describing the boundary conditions and where X_c is a submanifold of X with boundary and which defines the pure state constraints. Note that all the sets X, U, M_b and X_c may depends on Λ, and we assume that the functions f, f^0 and g have enough regularity (for instance at least C^1). Besides, problem (OCP) may have mixed state and control constraints (which is not presented for readibility). We are looking for solutions $(x(\cdot),u(\cdot),t_0,t_f)$, for which the optimal control $u(\cdot)$ leaves in $\mathcal{U}\subset \mathrm{L}^\infty([t_0\,,t_f],U)$, where $\mathcal{U}$ is the set of *admissible controls*[1], and where the trajectory $x(\cdot)$ is absolutely continuous.

Here is a non-exhaustive list of possibilities:

- The problem (OCP) may be in Bolza form as presented or in Mayer form, *i.e.* $f^0\equiv 0$, or in Lagrange form, *i.e.* $g\equiv 0$.
- The system may be autonomous, *i.e.* it has no explicit dependency on the time t and so we have $f(x,u,\Lambda)$ and $f^0(x,u,\Lambda)$.
- The control domain U may be compact, open (*i.e.* no control constraints)...
- The set of state constraints X_c may also be open so we do not have any pure state constraints. It may also be defined by an equation of the form $c(x(t))\leqslant 0$, $t\in[t_0\,,t_f]$.
- The initial time t_0 and the final time t_f may be free or fixed. If they are fixed, then the cost becomes $J(x(\cdot),u(\cdot))$. Besides, if the initial point $x(t_0)$ is fixed to the value $x_0\in X$ for instance, then the control $u(\cdot)$ determines uniquely the trajectory $x(\cdot)$ (by Cauchy-Lipschitz theorem) so the cost is simply $J(u(\cdot))$.

The optimal solution can be found as an extremal solution of the maximum principle (with or without state constraints), see [1, 4, 18, 19], and analyzed with the recent advanced techniques

[1]The set of admissible controls is the set of L^∞-mappings on $[t_0\,,t_f]$ taking their values in U such that the associated trajectory $x(\cdot)$ is globally defined on $[t_0\,,t_f]$.

of geometric optimal control. This analysis has to be performed to reduce the set of candidates as minimizers. These candidates, given by the maximum principle, are the concatenation of extremals solution of different Hamiltonian systems. Hence, a key point is to define all the Hamiltonian systems and to determine the sequence of the arcs. The optimal sequence (or the optimal structure) may involve bang arcs, where the optimal control norm is constant and maximum everywhere, or singular arcs with intermediate values on the norm of the control, or in the case of pure state constraints, we may have more complex structure with boundary ($x(t) \in \partial X_x$) and interior ($x(t) \in \mathring{X}_c$) arcs. When the optimal structure is determined, then we can define a Boundary Value Problem (BVP) which will be solved using the `HamPath` software.

Remark 1.1. The (BVP) problem is written as a set of non linear equations we have to solve, depending on the vector of parameters Λ. The `HamPath` code is made to solve these non linear equations (*i.e.* for a fixed value of Λ) but also to solve a family of optimal control problems, for Λ taking a range of values.

Remark 1.2. The maximum principle gives necessary conditions of optimality. As in the finite dimension case (*i.e.* in optimization in finite dimension), there exists necessary and sufficient conditions of higher order. In [1], the authors give necessary and sufficient conditions for problem (OCP) without state, neither control constraints, in the "regular case" for which the optimal control is smooth. These conditions arise from Jacobi equations and the theory of fields of extremals, which turns out to be checked by computing conjugate points, see [6]. This may be done using `HamPath`.

Here is a list of different studies using `HamPath`:

- In [11], the `HamPath` software is explained in details and some examples from quantum control or space mechanics can be found.
- In [9], the `HamPath` code is presented to solve regular optimal control problems where the optimal control is smooth. In this case, we give details on how we can check the second order conditions of optimality and how we can use differential path following methods to solve a one-parameter family of optimal control problems.
- In [7], the authors study the contrast problem in medical imaging by nuclear magnetic resonance. The optimal solution is the concatenation of bang and singular arcs. In this article, some comparisons with others methods are presented: direct and global techniques. We may also find some tests of second order of optimality and a study on the influence of the final time.
- In [10], the author presents an approach which combines geometric analysis and numerical methods to solve optimal control problems with pure state constraints, using the `HamPath` software. Multiple shooting and homotopy techniques are used to build a synthesis with respect to the bounds (2 parameters) of the boundary sets.

From the user sight. Applying the maximum principle leads to define a set of Hamiltonians and a Boundary Value Problem, which is described by a set of non linear equations, that can be grouped together in what we call the shooting equations. `HamPath` compiles the FORTRAN codes of the (maximized) Hamiltonians and the shooting function (defined by the shooting equations) and produces a collection of MATLAB, OCTAVE, FORTRAN or PYTHON functions (depending on

the chosen user interface) which allows first of all to compute the solutions of the Hamiltonian systems and to solve the implemented shooting equations.

However, it is well known that the main difficulty to solve such problems – with indirect methods based on Newton algorithms – is to find a good initial guess. So a differential path following method has been implemented which makes `HamPath` the natural extension of the `cotcot` package [5]. It is also possible to compute Jacobi fields of the Hamiltonian systems to check order two conditions of optimality and look for conjugate points, as `cotcot` does.

Organization of the user-guide. Subection 2.1 starts with a presentation of `HamPath` possibilities, through the study of a simple optimal control problem. This subsection ends up with a more detailed explanation of the main tool of the code: the differential path following (or homotopy) method. Then, in subsection 2.2, a schematic view of `HamPath` is presented in part 2.2.1 which leads to explain in more details the inputs and outputs of the code in part 2.2.2, and then the thirdparties used to build the core of the code, see 2.2.3. Section 3 is devoted to the resolution of some examples. A simple shooting example with only one single arc is presented in subsection 3.1 (the optimal control is smooth) while in subsection 3.2, the solution of the same problem but with a different criterium gives a Bang-Bang solution with two bang arcs (*i.e.* the optimal control has one discontinuity). Subsection 4 presents how to use efficiently `HamPath` on a non trivial example: the Goddard problem. In this case, we first solve a Bang-Bang (BB) problem, then a Bang-Singular-Bang (BSB) problem. The structure depends on the value of the final time, thus we finish showing how to detect this change of structure using the homotopy method. Finally, section 5 gives the instructions to install the `HamPath` code.

Keywords. Geometric optimal control; Simple and multiple shooting methods; Homotopy (or differential path following); Second order conditions of optimality (conjugate points); Goddard problem (Bang-Bang and Bang-Singular-Bang solutions).

2 `HamPath` overview

2.1 Presentation of the overall strategic and algorithmic approach

Let consider a simple optimal control problem with $q := (x, v)$ the state and with λ a parameter:

$$(P_\lambda) \qquad \begin{cases} J(u(\cdot)) = \dfrac{1}{2}\displaystyle\int_0^1 u(t)^2\,\mathrm{d}t \longrightarrow \min \\ \dot{x}(t) = v(t), \\ \dot{v}(t) = -\lambda v(t)^2 + u(t), \quad u(t) \in \mathbb{R}, \quad t \in [0\,,1] \text{ a.e.}, \\ x(0) = -1, \quad x(1) = 0, \\ v(0) = \ \ 0, \quad v(1) = 0, \end{cases}$$

where the initial and final times are fixed ($t_0 = 0$ and $t_f = 1$) and the boundaries are fixed to $q_0 := q(0) = (-1, 0)$ and $q_f := q(1) = (0, 0)$. Define the *pseudo-Hamiltonian* depending on λ:

$$\begin{aligned} H_\lambda \colon \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R} &\longrightarrow \mathbb{R} \\ (q, p, u) &\longmapsto H_\lambda(q, p, u) := p_x v + p_v(-\lambda v^2 + u) + \frac{1}{2}p^0 u^2, \end{aligned}$$

with $n = 2$ the state dimension, $p := (p_x, p_v)$ and $p^0 = -1$, assuming we are in the normal case. The application of the *Pontryagin Maximum Principle* (PMP) tells us that the minimizing trajectories $q(\cdot)$ are the projection of absolutely continuous *extremals* $z(\cdot) := (q(\cdot), p(\cdot))$, $z(\cdot)\colon t \mapsto z(t) \in \mathbb{R}^{2n}$, satisfying a.e.

$$\dot{z}(t) = \overrightarrow{h_\lambda}(z(t)) \tag{1}$$

with

Definition 2.1 (Maximized Hamiltonian)**.**

$$h_\lambda(z) := H_\lambda(z, \bar{u}(z)) = p_x v + p_v(-\lambda v^2 + \bar{u}(z)) - \frac{1}{2}\bar{u}^2(z), \tag{2}$$

the *maximized* (or true) and *smooth Hamiltonian*, where the optimal control is

$$\bar{u}(z) = p_v = \arg\max_{u\in\mathbb{R}} H_\lambda(z, u)$$

and where the *Hamiltonian system* is given by

$$\overrightarrow{h_\lambda}(z) = \left(\frac{\partial h_\lambda}{\partial p}(z), -\frac{\partial h_\lambda}{\partial q}(z)\right).$$

Definition 2.2 (Exponential mapping)**.** For fixed $\bar{z}_0 \in \mathbb{R}^{2n}$ and $\bar{t} \geqslant 0$, we define in a neighborhood of $(\bar{t}, \bar{z}_0)$ (if possible), the following *exponential mapping* $(t, z_0) \mapsto \exp(t\overrightarrow{h_\lambda})(z_0)$ as the trajectory $z(\cdot)$ at time t satisfying (1) for every $s \in [0\,, t]$, with $z(0) = z_0$.

Back to problem (OCP), the minimizing curves $q(\cdot)$ must satisfy the *boundary conditions*. As a consequence, they are the projection of what we call *BC-extremals*, *i.e.* extremals which satisfy the boundary conditions, and we can define the following *shooting function*:

Definition 2.3 (Shooting function)**.**

$$\begin{array}{rccl} S_\lambda\colon \mathbb{R}^n & \longrightarrow & \mathbb{R}^n \\ y & \longmapsto & S_\lambda(y) := \Pi_q(\exp((t_f - t_0)\,\overrightarrow{h_\lambda})(q_0, y)) - q_f, \end{array} \tag{3}$$

where Π_q is the *canonical q-projection*, *i.e.* $\Pi_q(z) = q$.

The *simple shooting method* consists in finding a zero of the simple shooting function S_λ, *i.e.* in solving $S_\lambda(y) = 0$. This is done by Newton type methods. A zero of the simple shooting function satisfies the necessary conditions of optimality given by the PMP.

Remark 2.4. S_λ depends on λ, and we write $S(y, \lambda) := S_\lambda(y)$ the *homotopic function* (instead of shooting function) when we consider the parameter λ as an *independent variable*. With `HamPath`, it is possible to solve $S(y, \lambda) = 0$ for $\lambda \in [0\,, 1]$ for instance, using differential path following methods. In this case, we say that λ is a homotopic parameter.

If we note $q(t, q_0, p_0) := \Pi_q(\exp(t\,\overrightarrow{h_\lambda})(q_0, p_0))$, then the trajectory $q(\cdot, q_0, p_0)$ ceases to be optimal after the time t_c if p_0 is a critical point of the mapping $q(t_c, q_0, \cdot)$. In this case, we name t_c a conjugate time and $q(t_c, q_0, p_0)$ the associated conjugate point. Let give the following definition.

Definition 2.5 (Jacobi field)**.** Let $\overrightarrow{h}$ be a Hamiltonian system, and let $z(\cdot)$ be a trajectory of $\overrightarrow{h}$ defined on $[0\,, t_f]$. The differential equation on $[0\,, t_f]$

$$\dot{\delta z}(t) = \mathrm{d}\overrightarrow{h}(z(t)) \cdot \delta z(t) \tag{4}$$

is called a *Jacobi equation*, or *variational system*, along $z(\cdot)$. Let $\delta z(\cdot)$ be a solution of (4), then we name $\delta z(\cdot)$ a *Jacobi field* and we write $\delta z(t) =: \exp(t\ \mathrm{d}\overrightarrow{h}|_{z(\cdot)})(\delta z(0))$.

As a conclusion, it comes that if t_c is a conjugate time then

$$\frac{\partial q}{\partial p_0}(t_c, q_0, p_0) = \Pi_q \circ \exp(t_c\ \mathrm{d}\overrightarrow{h_\lambda}|_{z(\cdot,q_0,p_0)})(\delta z_0), \quad \delta z_0 = \begin{bmatrix} 0_n \\ I_n \end{bmatrix},$$

is not of full rank n.

Summary of `HamPath` possibilities. The idea of `HamPath` is to produce a collection of numerical functions in order to solve problems of the form (OCP). The user must only implement the maximized Hamiltonian, see 2.1, and the shooting function, see 2.3. The different numerical functions can be used to:

- compute the solutions of the *exponential mapping*, see definition 2.2;
- solve the *shooting equations*, see definition 2.3;
- compute the *set of zeros* of a *homotopic function* given by a *family of shooting equations* depending on parameters, see remark 2.4;
- compute the *Jacobi fields*, see definition 2.5, and check if there exists any conjugate points.

Details on differential path following (or homotopy) methods. The homotopy method is used to solve a one-parameter family of optimal control problems. This approach is well known and widely used: see for example [2] for theoretical and numerical details. Here, we present some general facts related to homotopy and then we summarize how `HamPath` implements the differential path following method.

Suppose we want to solve the nonlinear equations $F(y) = 0$, $F\colon \mathbb{R}^N \to \mathbb{R}^N$ sufficiently smooth. If we know a good approximation of a zero point of F then it is advisable to calculate the zero point via a Newton-type algorithm. If it is not the case and no convergence is obtained, to remedy to this problem one can define a homotopy function $\Phi\colon \mathbb{R}^N \times \mathbb{R} \to \mathbb{R}$ such that $\Phi(y, 0) = G(y)$ and $\Phi(y, 1) = F(y)$ where $G\colon \mathbb{R}^N \to \mathbb{R}^N$ is a smooth mapping having known zeros. For example we can choose a convex homotopy $\Phi(y, \lambda) := \lambda F(y) + (1 - \lambda)G(y)$ and try to trace an implicitely defined curve contained in $\Phi^{-1}(0)$ from a starting point $(y_0, 0)$ to a solution point $(y_1, 1)$. Another possible deformation is the *global homotopy* (related to the concept of global Newton method): $\Phi(y, \lambda) = F(y) - (1 - \lambda)F(y_0)$. Another possibility making the homotopic method interesting is when the problem to solve has some parameters. Then the deformation of the solutions with respect to these parameters can be studied.

The classical difficulties about homotopic methods consist in assuring that a curve in $\Phi^{-1}(0)$ exists, is sufficiently smooth and will intersect the target homotopic level $\lambda = 1$ in a finite length.

Suppose first that Φ is continuously differentiable and that we know y_0 such as $\Phi(y_0, 0) = 0$. To ensure that a curve exists, we suppose:

$$\operatorname{rank}\left(\frac{\partial \Phi}{\partial y}(y_0, 0)\right) = N$$

and we suppose that 0 is a regular value of Φ, *i.e.* for every $r := (y, \lambda) \in \Phi^{-1}(0)$, r is a regular point:

$$\operatorname{rank}\left(\Phi'(r)\right) = N.$$

We do not discuss about bifurcation points or others singularities. In the regular case, a continuously differentiable curve starting from $r_0 := (y_0, 0)$ exists and is either diffeomorphic to a circle or the real line. The only possibilities that prevent to intersect the final target $\lambda = 1$ is that the curve returns back to the level $\lambda = 0$, goes to infinity (*i.e.* $|y| \to \infty$), or hits the boundary of the domain of Φ if any boundaries exist. Sufficient conditions which ensure that the curve will connect the target level $\lambda = 1$ can be linked with topological degree theory. See also Smale theorem from [2] which gives some sufficient boundary conditions for the global homotopy. Note that in the regular case, the curves in $\Phi^{-1}(0)$ are disjoints and we call each branch of $\Phi^{-1}(0)$ a path of zeros.

`HamPath` code. Since 0 is a regular value of Φ, then for any $r \in \Phi^{-1}(0)$, $\dim \ker \Phi'(r) = 1$ and one can define the (tangent) vector $T(r)$ as being the unique –up to orientation– unit vector in the kernel. The orientation is chosen so that the nonvanishing determinant

$$\det \begin{bmatrix} \Phi'(r) \\ T(r)^T \end{bmatrix} \tag{5}$$

has constant sign on each connected component of $\Phi^{-1}(0)$ which are then parameterized by arc length and are computed integrating the following differential equation (with $' = \mathrm{d}/\mathrm{d}s$):

$$r'(s) = T(r(s)), \quad r(0) = r_0 \in \Phi^{-1}(0),$$

with $r_0 := (y_0, 0)$ obtained by a first shooting. The path of zeros is computed by integrating the differential system with a high order Runge-Kutta scheme with step size control, combined with few steps of a Newton's method as corrector, see figure 1. See [2] for more details on prediction-correction methods.

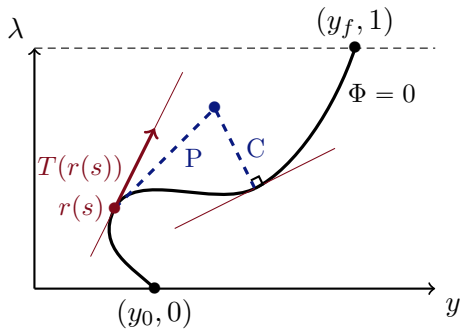

Figure 1: Predictor-Corrector (PC) method. Here the prediction is better than Euler step.

2.2 Features

2.2.1 Schematic view of `HamPath`

On figure 2, you can see the links between the different user's inputs and the outputs produced by `HamPath`. These inputs and outputs are detailed in the following subsection 2.2.2. The top part of the schema represents a piece of the user's input, which have to be coded in FORTRAN 90. The part of this picture between the doted lines is the core of `HamPath` that the user can't access. The libraries used by `HamPath` in its core are detailed in the thirdparty part (see 2.2.3). The part below the doted lines is a fragment of the outputs of `HamPath` which is in the language chosen during the installation: it may be chosen among FORTRAN, PYTHON, MATLAB (only) or both MATLAB and OCTAVE.

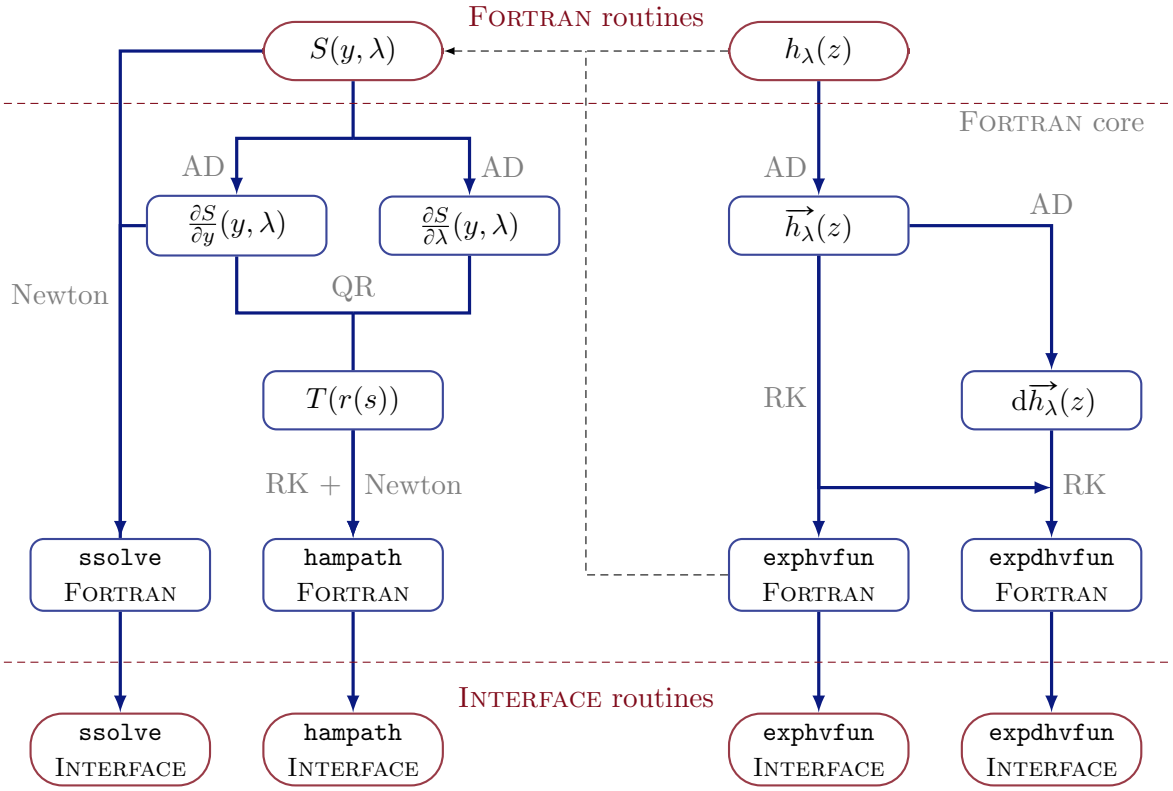

Figure 2: Global diagram of `HamPath` code. AD stands for Automatic Differentiation, RK for Runge-Kutta integrators used to solve ordinary differential equations, Newton for Newton-type methods to solve non-linear equations and QR for QR factorization.

2.2.2 Interface

The user has first to implement some FORTRAN 90 subroutines. There are five possible entries, at least one out of the first two is required for `HamPath` to work:

- `sfun.f90` contains the **`sfun`** subroutine wich codes the shooting function $S_\lambda(y)$ or equivalently the homotopic function $S(y,\lambda) = S_\lambda(y)$.

- `hfun.f90` contains the **hfun** subroutine which codes the maximized Hamiltonian $h_\lambda(z)$.
- `afun.f90` groups together a set of auxiliary subroutines that can be called from **sfun** and **hfun**. For instance the control is classicaly implemented there;
- `mfun.f90` may be used to do monitoring along the differential path following. For instance, see section 4.3.3, it can be used to detect structural changes of the solutions during the homotopy;
- `pfun.f90`: the problem (OCP) may depend on a vector parameter $\Lambda \in \mathbb{R}^k$, $k \geqslant 1$. A homotopy on several parameters (from Λ_0 to Λ_f for instance) can be computed using a scalar reparameterization. This can be done by adding a scalar parameter λ such as $\Lambda = \Lambda_0$ when $\lambda = 0$ and $\Lambda = \Lambda_f$ when $\lambda = 1$. The affine homotopic function $\Lambda(\lambda) = (1-\lambda)\Lambda_0 + \lambda\Lambda_f$ is a simple example of such a parameterization. This is the default behavior of **HamPath** code when the user ask to perform a homotopy on several parameters, unless the user implements its own function $\Lambda(\lambda)$ into the file `pfun.f90`.

The outputs of **HamPath** are (see paragraph "Manuals of **HamPath** functions" from section 2.2.4 for help):

- **hfun**, **hvfun**, **dhvfun**: these functions are created from a `hfun.f90` file and implement respectively the maximized Hamiltonian h_λ, the Hamiltonian system $\overrightarrow{h_\lambda}$ and its differential $\mathrm{d}\overrightarrow{h_\lambda}$;
- **exphvfun**, **expdhvfun**: these functions really flow from the previous ones and may be used to compute the solutions of $\dot{z}(t) = \overrightarrow{h}_\lambda(z(t))$ (see the definition of the exponential mapping 2.2), or the solutions of $\dot{\delta z}(t) = \mathrm{d}\overrightarrow{h_\lambda}(z(t)) \cdot \delta z(t)$ (see the definition of a Jacobi field 2.5);
- **sfun**, **sjac**: the first function makes just a call to the FORTRAN subroutine **sfun** implemented by the user in the file `sfun.f90`, while the function **sjac** computes the Jacobian of the shooting function.
- **ssolve**, **hampath**: these are really important outputs. The function **ssolve** solves the shooting equations by shooting methods, while the command **hampath** may be used to solve a family of optimal control problems, computing a path of zeros of the associated homotopic function.
- **hampathOptions**: this is the class used to manipulate the options of the previous functions, only under the PYTHON interface;
- **hampathset**, **hampathget**: these files group together the methods to set and get options of the previous functions, only with the MATLAB and/or OCTAVE interface;
- each subroutine of `afun.f90` whose signature is identical to `hfun` in `hfun.f90` will have automatically an implementation in the interface.

2.2.3 Core of HamPath and thirdparty

The Fortran hybrid Newton method **hybrj** (from the MINPACK library [21]) is used to solve the nonlinear system $S_\lambda(y) = 0$. Providing h_λ and S_λ to **HamPath**, the code generates automatically the Jacobian of the shooting function which is given to the solver. To make the implementation of S_λ easier, **HamPath** supplies the exponential mapping. Automatic Differentiation (TAPENADE software [17]) is used to produce $\overrightarrow{h_\lambda}$ and is combined with Runge-Kutta integrators, see [15, 16], to assemble the exponential mapping. Here is the list of the available Runge-Kutta methods:

- explicit with fixed step: **Euler_Explicite**, **Runge**, **Heun**, **rk4** (5);
- explicit with step size control: **Dopri5** (default), **Dop853**;
- implicit with fixed step: **Euler**, **MidPoint**, **Gauss4** (6, 8), **Radau IA1** (3, 5), **Radau IIA1** (3, 5), **RadauS**, **Lobatto4** (6), **LobattoIIIA2** (4, 6), **LobattoIIIB2** (4, 6), **LobattoIIIC2** (4, 6), **SDIRK3**, **SDIRK4L**, **SDIRK4A**, **DIRK5**;
- implicit with step size control: **Radau5** (9, 13) and a **Radau** with adaptative order.

Besides, to compute the tangent vector $T(r(s))$ along the path of zeros, **HamPath** calls LAPACK library [3] for QR factorization, and actually every matrix-matrix or matrix-vector operations is performed by BLAS [12, 13] subroutines.

2.2.4 Additional information on HamPath code: get examples, helps and options.

Get examples. You can copy the files describing the simple shooting problem into your current directory by typing in your terminal:

- `hampath -example simple_shooting`.

You can check the list of the available examples by typing:

- `hampath -example`.

Helps. There are manuals for every function of **HamPath**, you can access them by typing:

- `hampath -help function` in your **terminal**.
- `help function` in MATLAB for your MATLAB/OCTAVE INTERFACE.
- `help(function)` in the PYTHON interpreter.

Options. There are options for many functions of **HamPath**, you can get details by typing:

- `hampath -help options` in your **terminal**.
- `help hampathset` in MATLAB or OCTAVE under the MATLAB/OCTAVE INTERFACE.
- `help(HampathOptions)` in the PYTHON interpreter.

Outputs of `hampath` function. The `hampath` command is quite verbose. After the compilation, when the interface is loaded, you can get something like this:

```
 Homotopic param.   Arclength s     det(s)          |S(y)|          Inner product   ...

-0.15793129590E-16  0.00000000E+00 -0.83336587E-01  0.31577991E-14  0.00000000E+00  ...
 0.58238810394E-06  0.58239015E-06 -0.83333344E-01  0.69388939E-14  0.99996531E+00  ...
 0.94887097171E-06  0.94887309E-06 -0.83333347E-01  0.12880501E-13  0.10000000E+01  ...
 0.24319249308E-05  0.24319271E-05 -0.83333340E-01  0.86870303E-14  0.99999998E+00  ...
 0.44863073286E-05  0.44863095E-05 -0.83333335E-01  0.24588761E-14  0.99999999E+00  ...
 0.11416414981E-04  0.11416417E-04 -0.83333336E-01  0.85602443E-14  0.10000000E+01  ...
 0.30991940957E-04  0.30991943E-04 -0.83333337E-01  0.58894820E-15  0.10000000E+01  ...
 0.91219417098E-04  0.91219428E-04 -0.83333359E-01  0.28527632E-15  0.99999989E+00  ...
 0.21626296893E-03  0.21626309E-03 -0.83333471E-01  0.17737528E-14  0.99999948E+00  ...
 0.61298250969E-03  0.61298515E-03 -0.83334428E-01  0.45459024E-15  0.99999461E+00  ...

   ...

 0.52007251880E+00  0.12689392E+01 -0.41334984E+00  0.32922624E-15  0.99981773E+00  ...
 0.56415947311E+00  0.14669763E+01 -0.45550566E+00  0.12570776E-14  0.99985895E+00  ...
 0.60972561305E+00  0.16868735E+01 -0.50157364E+00  0.51763654E-14  0.99988966E+00  ...
 0.65678541917E+00  0.19300563E+01 -0.55207738E+00  0.81052455E-15  0.99991286E+00  ...
 0.70534162343E+00  0.21979006E+01 -0.60761829E+00  0.54793088E-14  0.99993060E+00  ...
 0.75538497947E+00  0.24917071E+01 -0.66888567E+00  0.71239377E-15  0.99994431E+00  ...
 0.80689378431E+00  0.28126725E+01 -0.73666771E+00  0.24432825E-14  0.99995502E+00  ...
 0.85983368675E+00  0.31618618E+01 -0.81186362E+00  0.29758909E-15  0.99996345E+00  ...
 0.91415922277E+00  0.35401921E+01 -0.89549939E+00  0.20051516E-14  0.99997013E+00  ...
 0.96981434231E+00  0.39484123E+01 -0.98874381E+00  0.34271409E-14  0.99997545E+00  ...
 0.99999999660E+00  0.41784127E+01 -0.10428214E+01  0.74373845E-14  0.99999398E+00  ...

Results of the homotopy:

Homotopy successfully completed !

steps               = 44
flag                = 1
sf                  =  0.417841271645901E+01
lambdaf             =  0.999999996602098E+00
|S(y_final)|        =  0.743738448589125E-14


y_final = [  0.159471278099168E+02  0.648065830023709E+01]'

par_final = [  0.000000000000000E+00  ...  0.999999996602098E+00]'
```

There is a chart of values and some columns that have to be detailed:

- `Arclength s`: it is the arclength of the path of zeros at each iteration of the algorithm.
- `det(s)`: it is the value of the following *determinant* (cf. eq. (5)) at each iteration of the algorithm
$$\det \begin{bmatrix} S'(r(s)) \\ r'(s)^T \end{bmatrix},$$
with $r := (y, \lambda)$.
- `Inner product`: it is the *scalar product* of two consecutive *tangent vectors* (cf. definition 2.1) at each iteration of the algorithm, this product should be close to 1.

3 Simple examples

3.1 A simple shooting problem

Let consider the following optimal control problem which will be solved by simple shooting:

$$(P_\lambda) \qquad \begin{cases} J(u(\cdot)) = \dfrac{1}{2}\displaystyle\int_{t_0}^{t_f} u(t)^2\,\mathrm{d}t \longrightarrow \min \\ \dot{x}(t) = v(t), \\ \dot{v}(t) = -\lambda v(t)^2 + u(t), \quad u(t) \in \mathbb{R}, \quad t \in [t_0\,, t_f] \text{ a.e.}, \\ \\ x(t_0) = x_0, \quad x(t_f) = x_f, \\ v(t_0) = v_0, \quad v(t_f) = v_f, \end{cases}$$

with $t_0 = 0$, $t_f = 1$, $q_0 := (x_0, v_0) = (-1, 0)$ and $q_f := (x_f, v_f) = (0, 0)$. We define the following vector of parameters:

$$\Lambda(\lambda) := (t_0, t_f, x_0, v_0, x_f, v_f, \lambda). \tag{6}$$

The vector $\Lambda(\lambda)$ is the par vector in the headers of the FORTRAN routines and in the `main` file.

Main goals. For this simple problem we want to

- solve P_λ for $\lambda \in [0\,, 1]$;
- check the optimality of the solution for $\lambda = 1$;
- display the path of zeros for $\lambda \in [0\,, 1]$ and the solution for $\lambda = 1$.

We present now the files the user has to implement to solve this problem. You can copy the files into your current directory, see paragraph "Get examples" from section 2.2.4.

3.1.1 User implementation of `hfun` Fortran routines.

Remark 3.1. See section 3.2.1 to get an idea of the role of iarc variable!

The PMP gives us the *optimal control* (cf. definition 2.1 and listing 1) $\bar{u}(z) := p_v$ and we can define the *maximized Hamiltonian* (cf. definition 2.1 and listing 2)

$$h_\lambda(z) := p_x v + p_v(-\lambda v^2 + \bar{u}(z)) + \frac{1}{2}p^0\,\bar{u}^2(z), \quad p^0 = -1 \text{ (normal case)}.$$

```
Subroutine control(t,n,z,iarc,npar,par,u)
    implicit none
    integer,                          intent(in)  :: n,npar,iarc
    double precision,                 intent(in)  :: t
    double precision, dimension(2*n), intent(in)  :: z
    double precision, dimension(npar),intent(in)  :: par
    double precision,                 intent(out) :: u

    u = z(n+2) ! u = pv

end subroutine control
```

Listing 1: `afun.f90`

```
Subroutine hfun(t,n,z,iarc,npar,par,h)
    implicit none
    integer,                              intent(in)  :: n,npar,iarc
    double precision,                     intent(in)  :: t
    double precision, dimension(2*n),     intent(in)  :: z
    double precision, dimension(npar),    intent(in)  :: par
    double precision,                     intent(out) :: h

    ! Local declarations
    double precision :: x,v,px,pv,lambda,u

    lambda  = par(7)
    x       = z(  1);    v    = z(  2)
    px      = z(n+1);    pv   = z(n+2)

    call control(t,n,z,iarc,npar,par,u)

    h = px*v + pv*(-lambda*v**2 + u) - 0.5d0*u**2

end subroutine hfun
```

Listing 2: `hfun.f90`

3.1.2 User implementation of `sfun` Fortran routines.

Then, we define the *Shooting function* (cf. definition 2.3 and listing 3)

$$\begin{aligned} S_\lambda : \mathbb{R}^2 &\longrightarrow \mathbb{R}^2 \\ y &\longmapsto S_\lambda(y) := \Pi_q(\exp((t_f - t_0)\,\overrightarrow{h_\lambda})(q_0, y)) - q_f \end{aligned}$$

```
Subroutine sfun(ny,y,npar,par,s)
    use mod_exphv4sfun
    implicit none
    integer,                                intent(in)  :: ny
    integer,                                intent(in)  :: npar
    double precision,  dimension(ny),       intent(in)  :: y
    double precision,  dimension(npar),     intent(in)  :: par
    double precision,  dimension(ny),       intent(out) :: s

    !local variables
    double precision :: z0(4), zf(4), tspan(2)
    integer          :: n, iarc

    n       = 2                 ! Dimension of the state

    tspan(1) = par(1)           ! t0
    tspan(2) = par(2)           ! tf

    z0(  1)  = par(3)           ! x0
    z0(  2)  = par(4)           ! v0
    z0(n+1)  = y(1)             ! px0
```

```
    z0(n+2)  = y(2)                ! pv0

    iarc     = 1                   ! There is only one arc:
                                   !   compare with Bang–Bang case

    call exphv(tspan,n,z0,iarc,npar,par,zf)

    s(1)     = zf(1) - par(5)      ! x(tf) - xf
    s(2)     = zf(2) - par(6)      ! v(tf) - vf

end subroutine sfun
```

Listing 3: Shooting function in `sfun.f90`

3.1.3 User implementation of the `main` file.

We present the most important parts of the main file `main.m` (callable from the MATLAB INTERFACE). See paragraph "Get examples" from section 2.2.4 to get the complete file.

1. We first use the `ssolve` command to find a y_0 such as $S_\lambda(y_0) = 0$ with $\lambda = 0$:

   ```
   [y0,ssol,nfev,njev,flag] = ssolve(yGuess,options,par0);
   ```

   where

   - yGuess is a wisely chosen initial guess.
   - options are the default options (cf. the options paragraph section 2.2.4).
   - par0 is the vector of parameters $\Lambda(0)$, *i.e.* with $\lambda = 0$ (cf. eq. (6)).

2. Then we compute the path of zeros of $S(y, \lambda) = 0$ for $\lambda \in [0, 1]$ (we consider here λ as a *homotopic variable* of S and not as a *parameter*) with the `hampath` command:

   ```
   [parout,yout,~,~,~,~,flag] = hampath(parspan,y0,options);
   ```

   where

   - parspan is the couple of vector of parameters (Λ_0, Λ_f) with $\Lambda_0 = \Lambda(0)$ and $\Lambda_f = \Lambda(1)$.
   - y0 is the solution at Λ_0.
   - options are still the default options.

   That gives us:

3. We get p0f the solution at Λ_f and compute the extremal $z(\cdot)$ and the control $u(\cdot)$ at Λ_f with the `exphvfun` and `control` commands (see figures 4 and 5):

   ```
   p0f      = yout(:,end);
   [tout,z,flag] = exphvfun([t0 tf],[q0;p0f],options,parf);
   u        = control(tout,z,parf);
   ```

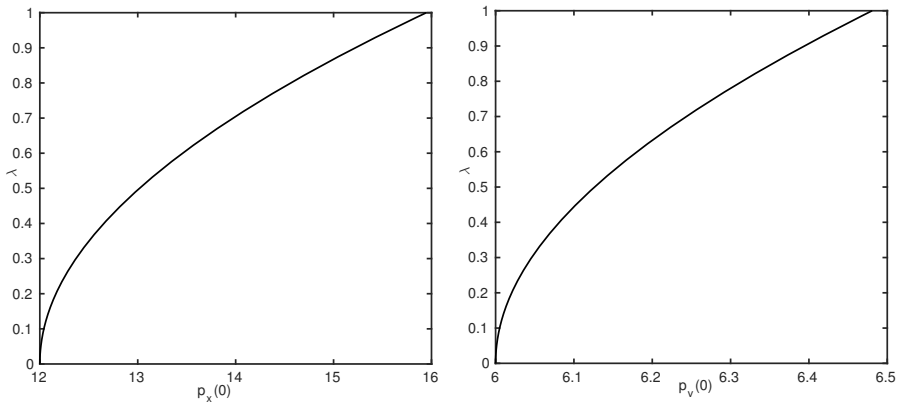

Figure 3: path of zeros for $\lambda \in [0\,,1]$

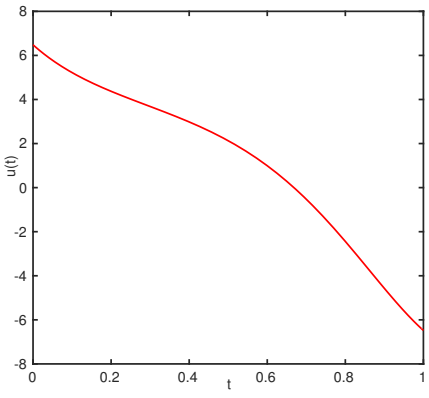

Figure 4: Optimal control at Λ_f.

4. At Λ_f, we compute the *Jacobi fields* $\delta z(\cdot)$ along $z(\cdot)$ (cf. definition 2.5) for $t \in [t_0\,,t_f]$ with $z(t_0) = (q_0, p_{0,f})$ and $\delta z_0 := \delta z(t_0) = \begin{bmatrix} 0_n \\ I_n \end{bmatrix}$:

```
z0   = [q0;p0f];  dz0 = [zeros(n);eye(n)];
[tout,z,dz,flag] = expdhvfun([t0  tf],z0,dz0,options,parf);
```

And check the optimality of the solution by looking at:

- the *determinant* of $\Pi_q \circ \exp((t-t_0)\,\mathrm{d}\overrightarrow{h_\lambda}|_{z(\cdot)})(\delta z_0)$ for $\lambda = 1$ which must be of a constant sign for $t \in [t_0\,,t_f]$.
- the smallest *singular value* which must not vanish.

```
sv=[];  de=[];
for  j = 1:length(tout)
     dq     = dz(1:n,1+(j-1)*k:j*k);
     sv(j) = min(svd(dq));              % Get smallest singular value
     de(j) = det(dq);                   % Get determinant
end;
```

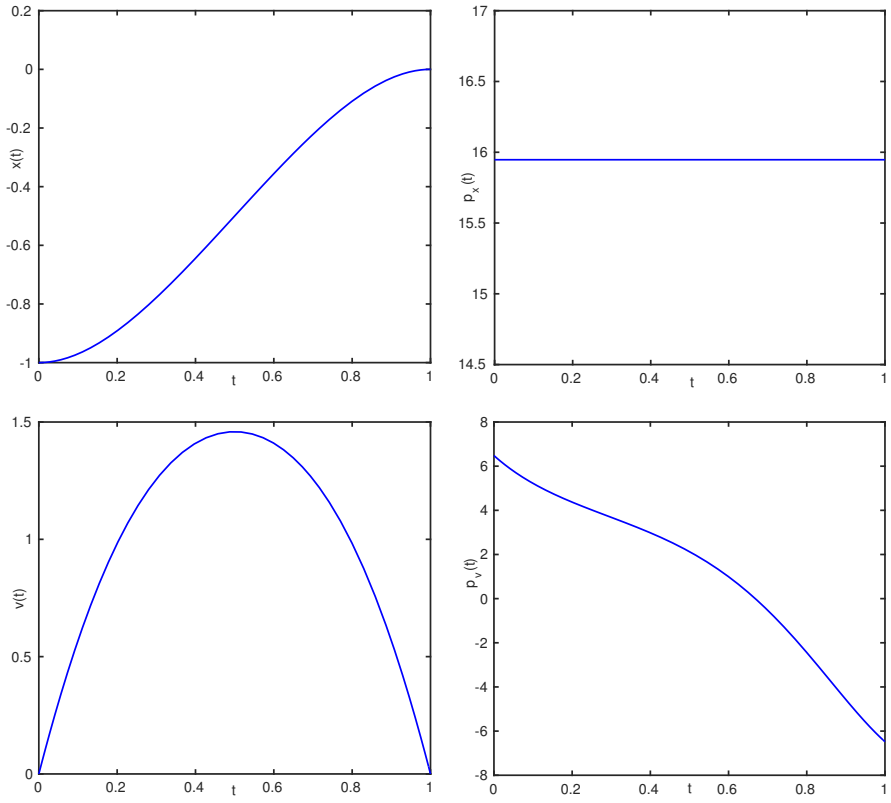

Figure 5: State and co-state solution at Λ_f.

This gives

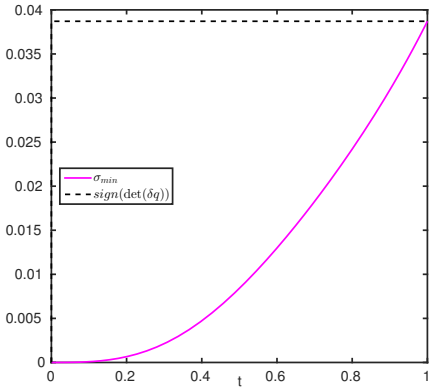

Figure 6: Conjugate points at Λ_f.

3.2 A multiple shooting problem

Let's look at another problem P_λ:

$$(P_\lambda) \qquad \begin{cases} J(t_f, u(\cdot)) = t_f \longrightarrow \min \\ \\ \dot{x}(t) = v(t), \\ \dot{v}(t) = -\lambda v(t)^2 + u(t), \quad u(t) \in [-1\,,1], \quad t \in [t_0\,, t_f] \text{ a.e.}, \\ \\ x(t_0) = x_0, \quad x(t_f) = x_f, \\ v(t_0) = v_0, \quad v(t_f) = v_f, \end{cases}$$

with $t_0 = 0$, $q_0 := (x_0, v_0) = (-1, 0)$ and $q_f := (x_f, v_f) = (0, 0)$. We define the following vector of parameters:

$$\Lambda(\lambda) := (t_0, x_0, v_0, x_f, v_f, \lambda). \tag{7}$$

The vector $\Lambda(\lambda)$ is the par vector in the headers of the FORTRAN routines and in the `main` file.

Main goal. For this problem we want to

- solve P_λ for $\lambda \in [0\,,1]$;
- compute and display the path of zeros for $\lambda \in [0\,,1]$ and the solution for $\lambda = 1$.

We present now some parts of the files the user has to implement to solve this problem. You can copy the files into your current directory, see paragraph "Get examples" from section 2.2.4.

3.2.1 User implementation of `hfun` Fortran routines

The *pseudo-Hamiltonian* of this problem is:

$$H_\lambda(q, p, u) = p_x v + p_v(-\lambda v^2 + u),$$

with $q := (x, v)$ and $p := (p_x, p_v)$. The optimal control is given by:

$$\bar{u}(z) = \begin{cases} +1 & \text{if} \quad p_v > 0 \\ u_s(z) & \text{if} \quad p_v = 0 \\ -1 & \text{if} \quad p_v < 0 \end{cases}$$

with $z := (q, p)$ and where $u_s(z) \in [-1\,,1]$ is the *singular control*. So, it is giving us this set of *Hamiltonians*:

$$h_\lambda(z) = \begin{cases} h_+(z) := H_\lambda(z, +1) & \text{when} \quad p_v > 0 \\ h_s(z) := H_\lambda(z, u_s(z)) & \text{when} \quad p_v = 0 \\ h_-(z) := H_\lambda(z, -1) & \text{when} \quad p_v < 0 \end{cases}.$$

The PMP gives us the structure of the solution: Bang-Bang with two arcs associated to h_+ then h_-. The role of the iarc variable, in `control` and `hfun` subroutines, is to choose the control that fit to the current arc of study, it is the index of the arc.

```
if(iarc.eq.1)then
  u = 1d0
else ! iarc = 2
  u = -1d0
end if
```

control in `afun.f90`

```
lambda  = par(6)
x       = z(   1); v       = z(   2)
px      = z(n+1); pv       = z(n+2)

call control(t,n,z,iarc,npar,par,u)

h = px * v + pv * (-lambda*v**2 + u)
```

Hamiltonian in `hfun.f90`

3.2.2 User implementation of `sfun` Fortran routines.

The *shooting function* is a bit more tricky than the previous one, because we have 8 unknown variables: the initial adjoint vector, the final time t_f, the switching time t_1 and the state and co-state z_1 at t_1. The shooting function becomes

$$S_\lambda: \quad \mathbb{R}^8 \longrightarrow \mathbb{R}^8$$
$$y := \begin{bmatrix} t_1 \\ t_f \\ p_{x,0} \\ p_{v,0} \\ z_1 \end{bmatrix} \longmapsto S_\lambda(y) := \begin{bmatrix} x(t_f) - x_f \\ v(t_f) - v_f \\ z_1 - z(t_1) \\ p_v(t_1) \\ h_-(z(t_f)) - 1 \end{bmatrix}$$

with

$$z(t_1) := \exp((t_1 - t_0)\overrightarrow{h_+})(z_0), \quad z(t_f) := \exp((t_f - t_1)\overrightarrow{h_-})(z_1)$$

and $z_0 := (x_0, v_0, p_{x,0}, p_{v,0})$.

```
n       = 2
t0      = par(1); t1 = y(1); tf = y(2)
z0(   1) = par(2); z0(   2) = par(3)   ! x0, v0
z0(n+1) = y(3)   ; z0(n+2) = y(4)      ! px0, pv0
z1      = y(5:8)

! Integration on the first arc: u = +1
iarc    = 1; tspan   = (/t0, t1/)
call exphv(tspan,n,z0,iarc,npar,par,expz0)

! Integration on the second arc: u = -1
iarc    = 2; tspan   = (/t1, tf/)
call exphv(tspan,n,z1,iarc,npar,par,expz1)

call hfun(tf,n,expz1,iarc,npar,par,hf)

s(1)    = expz1(1) - par(4) ! Final condition on xf
s(2)    = expz1(2) - par(5) ! Final condition on vf
s(3:6)  = z1 - expz0        ! Matching condition z1 = z(t1)
s(7)    = expz0(n+2)        ! Switching condition pv(t1) = 0
s(8)    = hf - 1d0          ! Final Hamiltonian condition
```

Shooting function in `sfun.f90`

3.2.3 User implementation of the `main` file

We present the most important parts of the main file `main.m` (callable from the MATLAB INTERFACE). See paragraph "Get examples" from section 2.2.4 to get the complete file.

1. We first use the `ssolve` command to find a y_0 such as $S_\lambda(y_0) = 0$ with $\lambda = 0$:

```
[y0,ssol,nfev,njev,flag] = ssolve(yGuess,options,par0);
```

   where

   - yGuess is a wisely chosen initial guess.
   - options are the default options (cf. the options paragraph section 2.2.4).
   - par0 is the vector of parameters $\Lambda(0)$, *i.e.* with $\lambda = 0$ (cf. eq. (7)).

2. Then we compute the path of zeros of $S(y, \lambda) = 0$ for $\lambda \in [0, 1]$ with the `hampath` command, see figure 2:

```
[parout,yout,~,~,~,~,flag] = hampath(parspan,y0,options);
```

   where

   - parspan is the couple of vector of parameters (Λ_0, Λ_f) with $\Lambda_0 = \Lambda(0)$ and $\Lambda_f = \Lambda(1)$.
   - y0 is the solution at Λ_0.
   - options are still the default options.

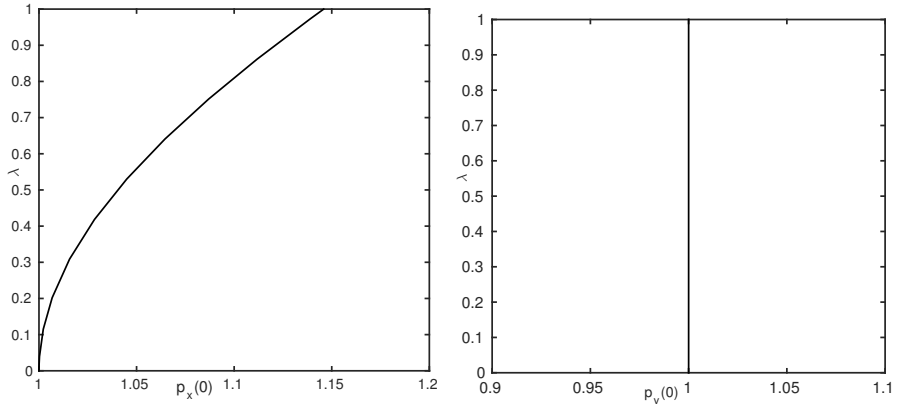

Figure 7: Path of zeros

3. We want to display the solution for $\lambda = 1$. To do so, we get p0f the co-vector at Λ_f and compute the extremal $z(\cdot)$ and the control $u(\cdot)$ at Λ_f with the `exphvfun` and `control` commands (see figures 8 and 9):

```
p0f  = yout(3:4,end);
ti   = [t0 t1 tf];  % ti is required since there are two arcs!
[tout,z,flag] = exphvfun([t0 tf],[q0;p0f],ti,options,parf);
u    = control(tout,z,ti,parf);
```

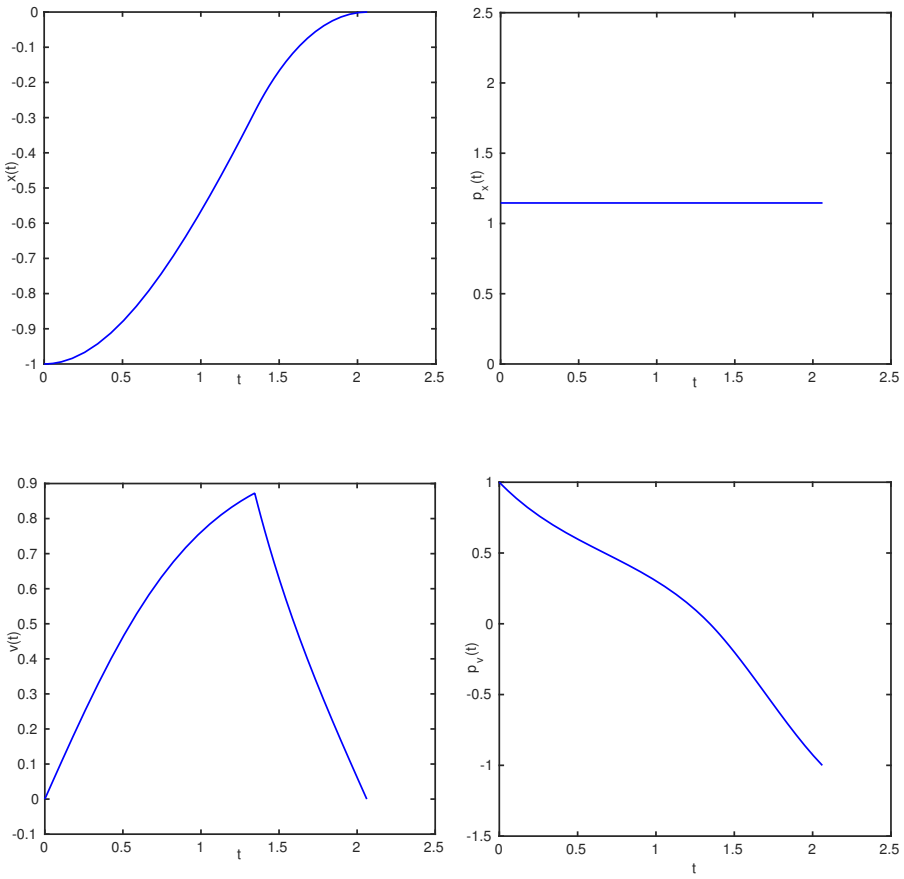

Figure 8: State and co-state solution at Λ_f.

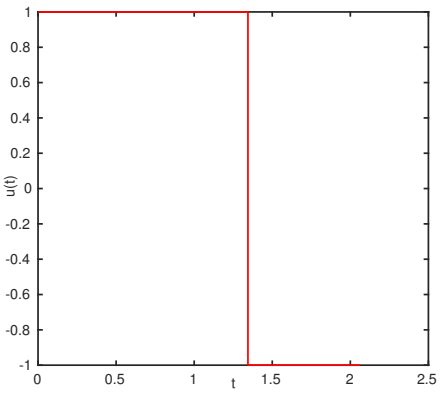

Figure 9: Optimal control at Λ_f.

4 Goddard problem

We will now focus on a one hundred years old problem: the **Goddard Problem**. This problem has already been treated many times (see [14], [20] and [22]), wether theoretically and numerically. Here, we are going to focus on some particular cases of this problem.

The goal is to maximize the final altitude of a rocket which is flying verticaly (a one-dimension flight) with a fixed final time. Here are the equations of the problem:

$$(P_{t_f}) \qquad \begin{cases} \max h(t_f) \\ \dot{h}(t) = v(t), \\ \dot{v}(t) = \dfrac{1}{m(t)}\left(c\,u(t) - D(v(t), h(t))\right) - g_0, \\ \dot{m}(t) = -u(t), \\ 0 \leqslant u(t) \leqslant u_{\max}, \\ h(t_0) = h_0, \quad v(t_0) = v_0, \quad m(t_0) = m_0, \\ m(t_f) = m_f, \end{cases}$$

with $t_0 = 0$, $q_0 := (h_0, v_0, m_0) = (0, 0, m_0)$, and where $h(t)$ is the altitude at the time t, $v(t)$ the speed, $m(t)$ the mass of the rocket, $D(h, v)$ the drag, g_0 the constant of gravity, c the fixed specific impulse and $u(t)$ the control or *thrust*. The optimal structure depends on the value of the fixed final time t_f. We define a new vector of parameters:

$$\Lambda(t_f) := (t_f, \alpha, \beta, g_0, u_{\max}, c, t_0, h_0, v_0, m_0, m_f) \tag{8}$$

The *pseudo-Hamiltonian* of this system is

$$H(q, p, u) = p_h v + p_v \left(\frac{c\,u - D(h, v)}{m} - g_0\right) - p_m u = H_0(q, p) + u\,H_1(q, p)$$

with $H_0(q, p) := p_h v - p_v(\frac{D(h,v)}{m} - g_0)$ and $H_1(q, p) := \dfrac{p_v\,c}{m} - p_m$. The control law is defined by:

$$\bar{u}(z) = \begin{cases} u_{\max} & \text{if} \quad H_1(z) > 0 \\ u_s(z) & \text{if} \quad H_1(z) = 0 \\ 0 & \text{if} \quad H_1(z) < 0 \end{cases} \tag{9}$$

So, it is giving us this set of *Hamiltonians*:

$$h(z) = \begin{cases} h_+(z) := H_0(z) + u_{\max}\,H_1(z) & \text{when} \quad H_1(z) > 0 \\ h_s(z) := H_0(z) + u_s(z)\,H_1(z) & \text{when} \quad H_1(z) = 0 \\ h_0(z) := H_0(z) & \text{when} \quad H_1(z) < 0 \end{cases}. \tag{10}$$

with $z := (q, p)$ and u_s the singular control. Let compute the singular control. Let assume $H_1(z(t)) = 0$ for all $t \in I$, $I \subset [0\,, t_f]$ an interval with non empty interior. Then for all $t \in I$ we have:

$$\frac{\mathrm{d}}{\mathrm{d}t}(H_1 \circ z)(t) = \{H, H_1\}(z(t)) = \{H_0, H_1\}(z(t)) =: H_{01}(z(t)) = 0$$

where $\{H_0, H_1\}$ is the *Poisson bracket* between H_0 and H_1. Differentiating twice, we obtain:

$$\frac{\mathrm{d}}{\mathrm{d}t}(H_{01} \circ z)(t) = \{H, H_{01}\}(z(t)) = H_{001}(z(t)) + u(t)\,H_{101}(z(t)) = 0$$

Then, if $H_{101}(z) \neq 0$ we have:

$$u_s(z) = -\frac{H_{001}(z)}{H_{101}(z)} = \frac{D}{c} + m\frac{(c-v)\,D_h + g\,D_v + c\,g\,D_{vv} - c\,v\,D_{vh}}{D + 2\,c\,D_v + c^2 D_{vv}} \tag{11}$$

with $D := D(h,v)$, $D_h := \frac{\partial D}{\partial h}(h,v)$, $D_v := \frac{\partial D}{\partial v}(h,v)$, $D_{vh} := \frac{\partial^2 D}{\partial v \partial h}(h,v)$ and $D_{vv} := \frac{\partial^2 D}{\partial v^2}(h,v)$.

We are now going to treat a few different solutions of this problem depending on the fixed chosen final time t_f.

4.1 A Bang-Bang solution: $t_f = 20$.

We know (see [20]) that for $t_f = 20$, the structure of the solution is Bang-Bang with two arcs, first $u(\cdot) \equiv u_{max}$ and then $u(\cdot) \equiv 0$.

Main goal.

- Solve (P_{t_f}) for $t_f = 20$.
- Show the state and co-state solution, the optimal control and the graph of $t \mapsto H_1(z(t))$.

4.1.1 User implementation of `hfun` Fortran routine

Before coding the *maximized Hamiltonian* in the `hfun.f90` file, we detail two useful auxiliary functions, `geth1` and `control`, we implement in the `afun.f90` file. `geth1` will simply computes $H_1(z)$, see equation (10), and the `control` will code the control law defined in equation (9).

geth1 subroutine in afun.f90

This routine has the same head as the `hfun` subroutine which allows `HamPath` to create an interfaced function of `geth1`!

```
m  = z(3); pv = z(2+n); pm = z(3+n)
c  = par(6)
H1 = pv*c/m-pm
```

Listing 4: H_1 in `afun.f90`

The same way, we implemented a `geth0` subroutine in `afun.f90` to compute H_0.

control subroutine in afun.f90

The PMP gives us the structure of the solution: Bang-Bang with two arcs associated to h_+ and then h_0. The role of the iarc variable is to choose the control that fit to the current arc of study, it is the index of the arc. But, unlike in section 3.2, it is used in a more generic way: the structure of the control is stocked in the par vector of parameters. In this case, we have:

$$par = [\Lambda(t_f) \quad 1 \quad 0] \tag{12}$$

where 1 means that the first arc is $u(\cdot) \equiv u_{max}$ and 0 that the second arc is $u(\cdot) \equiv 0$. So if we note nparmin (= 11) the dimension of Λ, then the nparmin + iarc parameter of the par vector is the structure of the control on the current arc.

```
    umax       = par(5)
    labelArc   = nint(par(nparmin+iarc))
    select case(labelArc)
        case (0)          ! The second Bang arc is u = 0
            u = 0d0
        case (1)          ! The first Bang arc is u = umax
            u = umax
    end select
```

control in `afun.f90`

hfun subroutine in hfun.f90

We normalize the time by the simple change of variable $t = (t_f - t_0)\, s + t_0$, $s \in [0\,,1]$, and the Hamiltonian becomes:

$$H(q,p,u) = (t_f - t_0)(H_0(q,p) + u\, H_1(q,p))$$

```
 t0 = par(7); tf = par(1)
 call    geth0(t,n,z,iarc,npar,par,H0)
 call    geth1(t,n,z,iarc,npar,par,H1)
 call control(t,n,z,iarc,npar,par,u)
 H  = (tf-t0)*(H0+u*H1)
```

Listing 5: Hamiltonian in `hfun.f90`

4.1.2 User implementation of `sfun` Fortran routine

The *shooting function* is a bit more tricky than the previous one, because we have 10 unknown variables: the initial adjoint vector, the final time t_f, the switching time t_1 and the state and co-state z_1 at t_1. The shooting function becomes

$$\begin{array}{rccc} S_{t_f}: & \mathbb{R}^{10} & \longrightarrow & \mathbb{R}^{10} \\ & y := \begin{bmatrix} p_{h,0} \\ p_{v,0} \\ p_{m,0} \\ t_1 \\ z_1 \end{bmatrix} & \longmapsto & S_{t_f}(y) := \begin{bmatrix} z(t_1) - z_1 \\ h_1(z_1) \\ p_h(t_f) + p^0 \\ p_v(t_f) \\ m(t_f) - m_f \end{bmatrix} \end{array} \tag{13}$$

with

$$z(t_1) := \exp\left(\frac{t_1 - t_0}{t_f - t_0}\, \vec{h}_+\right)(z_0),$$

$$(h, v, m, p_h, p_v, p_m)(t_f) := \exp\left(\frac{t_f - t_1}{t_f - t_0} \vec{h}_0\right)(z_1)$$

and $p^0 = -1$, $z_0 := (q_0, p_{h,0}, p_{v,0}, p_{m,0})$.

```
n       = 3
p0      = y(1:3); t1          = y(4)   ; z1 = y(5:10)
tf      = par(1); t0          = par(7); q0 = par(8:10); mf = par(11)
z0(1:3) = q0     ; z0(4:6)    = p0
t0norm  = 0d0    ; t1norm     = (t1-t0)/(tf-t0); tfnorm = 1d0

! Integration on the first arc
iarc    = 1; tspan   = (/t0norm, t1norm/)
call exphv(tspan,n,z0,iarc,npar,par,expz0)
call geth1(t1norm,n,z1,iarc,npar,par,H1)

! Integration on the second arc
iarc    = 2; tspan   = (/t1norm, tfnorm/)
call exphv(tspan,n,z1,iarc,npar,par,expz1)

s(1:6)  = expz0       - z1    ! Matching condition
s(7)    = H1                  ! Switching condition
s(8)    = expz1(n+1) - 1d0    ! Transversality condition on ph
s(9)    = expz1(n+2)          ! Transversality condition on pv
s(10)   = expz1(3)    - mf    ! Final condition on m(tf)
```

Shooting function in `sfun.f90`

4.1.3 User implementation of the `main` file

We present the most important parts of the main file `main.m` (callable from the MATLAB INTERFACE). See paragraph "Get examples" from section 2.2.4 to get all the files.

1. We first use the `ssolve` command to find a y_0 such as $S_{t_f}(y_0) = 0$ with $t_f = 20$:

   ```
   [y0,ssol,nfev,njev,flag] = ssolve(yGuess,options,par);
   ```

   where

   - yGuess is a wisely chosen initial guess.
   - options are the default options (cf. the options paragraph section 2.2.4).
   - par is the vector of parameters of size nparmin + *the number of arcs*, see its initialization:

     ```
     par = [tf 0.01227 0.000145 9.81 9.52551 2060 t0 q0(1) q0(2) q0(3)
        67.983310 1 0]';  % tf alpha beta g0 umax c t0 q0 mf
     ```

2. To display the solution for $t_f = 20$, we compute the extremal $z(\cdot)$, the control $u(\cdot)$ and $H_1(z(\cdot))$ (see figure 10 and 11). We need the vector t_i of initial, switching and final times.

   ```
   ti   = [t0norm t1norm tfnorm];
   [tout,z,flag] = exphvfun([t0norm tfnorm],z0,ti,options,par);
   u    = control(tout,z,ti,par);
   H1   = geth1(tout,z,ti,par);
   ```

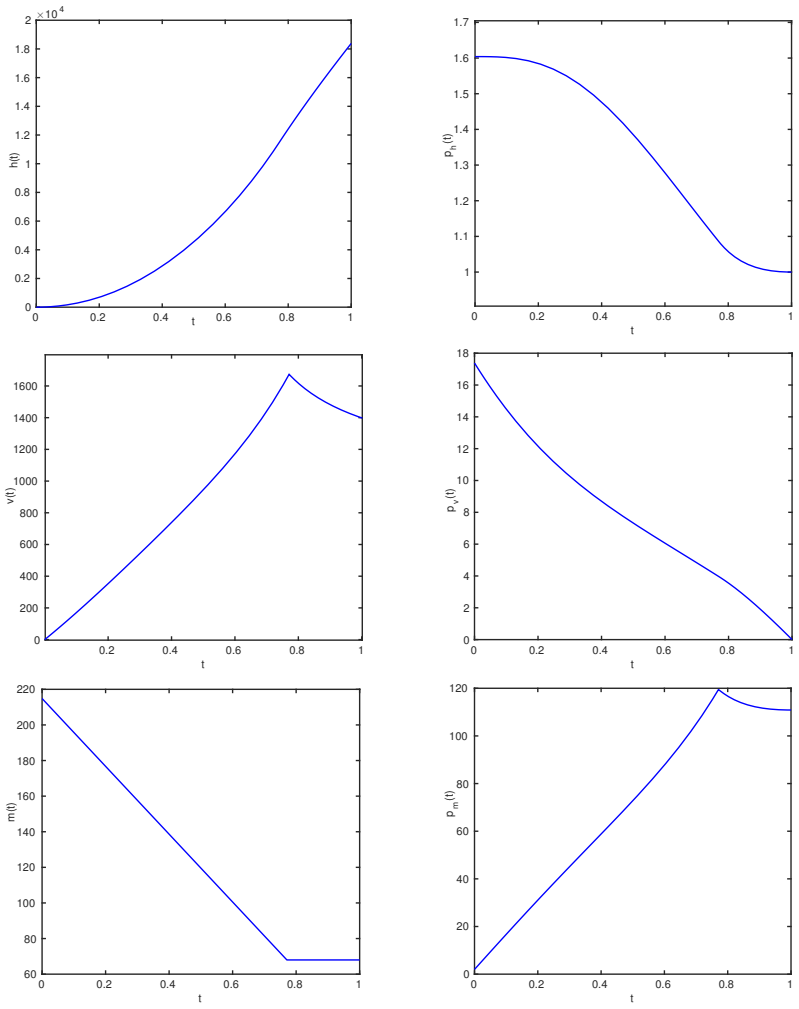

Figure 10: State (left) and co-state (right) solution

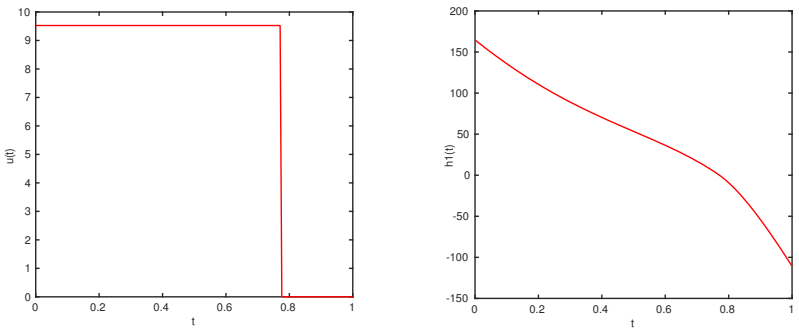

Figure 11: Optimal control and $H_1(z(\cdot))$

4.2 A Bang-Singular-Bang solution: $t_f \approx 206$.

We know (see [20]) that for a fixed final time $t_f \approx 206$, the structure of the solution is Bang-Singular-Bang with three arcs, first $u(\cdot) \equiv u_{max}$, then $u(\cdot) \equiv u_s(z(\cdot))$ and finally $u(\cdot) \equiv 0$. We also have the same vector of parameters as before (cf. equation (8)).

Main goal.

- Solve (P_{t_f}) for $t_f = 206.661$.
- Show the state and co-state solution, the optimal control, the graphs of $t \mapsto H_1(z(t))$ and $t \mapsto H_{01}(z(t))$.

4.2.1 User implementation of `hfun` Fortran routines.

Compare to the Bang-Bang case, we need now the singular control, see equation (9). The maximized Hamiltonian, H_1 and H_0 are the same as in the Bang-Bang case (cf. section 4.1.1), see listings 4 and 5.

```
ht = z(  1); v  = z(  2); m  = z(  3)
ph = z(n+1); pv = z(n+2); pm = z(n+3)
tf = par(1); alpha = par(2); beta = par(3);
g0 = par(4); umax  = par(5); c    = par(6)

D   = (alpha*v**2)       *exp(-beta*ht)  ! Drag function
Dh  = (-beta*alpha*v**2)*exp(-beta*ht)  ! d/dh (drag function)
Dv  = (2*alpha*v)        *exp(-beta*ht)  ! d/dv (drag function)
Dvv = (2*alpha)          *exp(-beta*ht)  ! d2/dv2 (drag function)
Dvh = (-beta*2*alpha*v) *exp(-beta*ht)  ! d2/dhdv (drag function)

labelArc = nint(par(nparmin+iarc))
select case(labelArc)
    case (0)          ! The third Bang arc is u = 0
        u = 0d0
    case (1)          ! The first Bang arc is u = umax
        u = umax
    case (2)          ! The second Bang arc is u = us
        u = D/c+m*((c-v)*Dh+g0*Dv+c*g0*Dvv-c*v*Dvh)/(D+2*c*Dv+Dvv*c**2)
end select
```

Listing 6: Control in `afun.f90`

4.2.2 User implementation of `sfun` Fortran routines.

The shooting function is:

$$
\begin{array}{rccc}
S_{t_f}: & \mathbb{R}^{17} & \longrightarrow & \mathbb{R}^{17} \\
& y := \begin{bmatrix} p_{h,0} \\ p_{v,0} \\ p_{m,0} \\ t_1 \\ z_1 \\ t_2 \\ z_2 \end{bmatrix} & \longmapsto & S_{t_f}(y) := \begin{bmatrix} z(t_1) - z_1 \\ z(t_2) - z_2 \\ h_1(z_1) \\ h_{01}(z_1) \\ p_h(t_f) + p^0 \\ p_v(t_f) \\ m(t_f) - m_f \end{bmatrix}
\end{array}
\tag{14}
$$

with

$$
z(t_1) := \exp(\frac{t_1 - t_0}{t_f - t_0}\,\vec{h}_+)(z_0), \quad z(t_2) := \exp(\frac{t_2 - t_1}{t_f - t_0}\,\vec{h}_s)(z_1),
$$

$$
(h, v, m, p_h, p_v, p_m)(t_f) := \exp(\frac{t_f - t_2}{t_f - t_0}\vec{h}_0)(z_2)
$$

and $p^0 = -1$, $z_0 := (q_0, p_{h,0}, p_{v,0}, p_{m,0})$. We see that we need a function that compute H_{01}:

```
ht  = z(  1); v  = z(  2); m  = z(  3)
ph  = z(n+1); pv = z(n+2); pm = z(n+3)
tf  = par(1); alpha = par(2); beta = par(3)
g0  = par(4); umax  = par(5); c    = par(6)
D   = (alpha*v**2)*exp(-beta*ht)
Dv  = (2*alpha*v) *exp(-beta*ht)
H01 = (1/m**2)*(pv*(D+c*Dv)-ph*c*m)
```

H_{01} in `afun.f90`

```
iarc    = 1; tspan = (/t0norm, t1norm/)
call exphv(tspan,n,z0,iarc,npar,par,expz0)
call geth1(t1norm,n,z1,iarc,npar,par,H1)
call geth01(t1norm,n,z1,iarc,npar,par,H01)

!Integration on the second arc
iarc    = 2; tspan = (/t1norm, t2norm/)
call exphv(tspan,n,z1,iarc,npar,par,expz1)

!Integration on the third arc
iarc    = 3; tspan = (/t2norm, tfnorm/)
call exphv(tspan,n,z2,iarc,npar,par,expz2)

s(1:6)  = z1          - expz0 ! Matching condition
s(7:12) = z2          - expz1 ! Matching condition
s(13)   = H1                  ! Contact with the switching surface
s(14)   = H01                 ! Contact of order 2
s(15)   = expz2(n+1) - 1d0    ! Transervsality condition on ph
s(16)   = expz2(n+2)          ! Transervsality condition on pv
s(17)   = expz2(n)   - mf     ! Final condition on m(tf)
```

Shooting function in `sfun.f90`

4.2.3 User implementation of the `main` file.

This part is very similar to the one in section 4.1.3. Just note that you can get all the files from this example typing in your terminal: `hampath -example goddardBSB`. See paragraph "Get examples" from section 2.2.4. Just note that the par vector is now:

$$par = [\Lambda(t_f) \quad 1 \quad 2 \quad 0].$$

We get the following results (see figures 12 and 13).

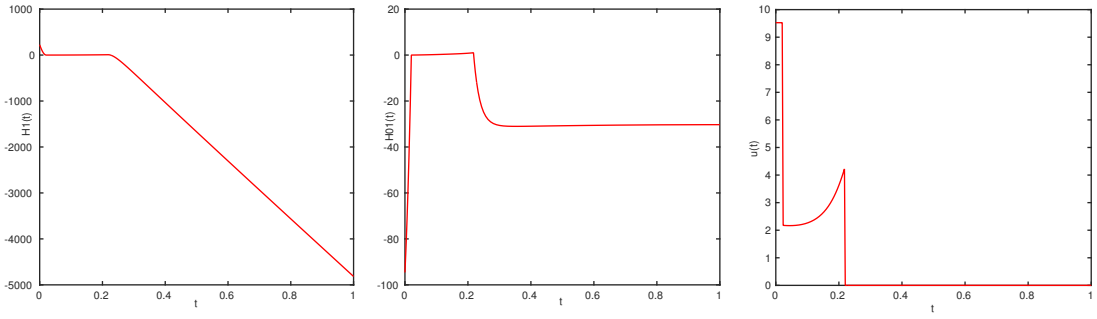

Figure 12: Optimal control, $H_1(z(\cdot))$, and $H_{01}(z(\cdot))$.

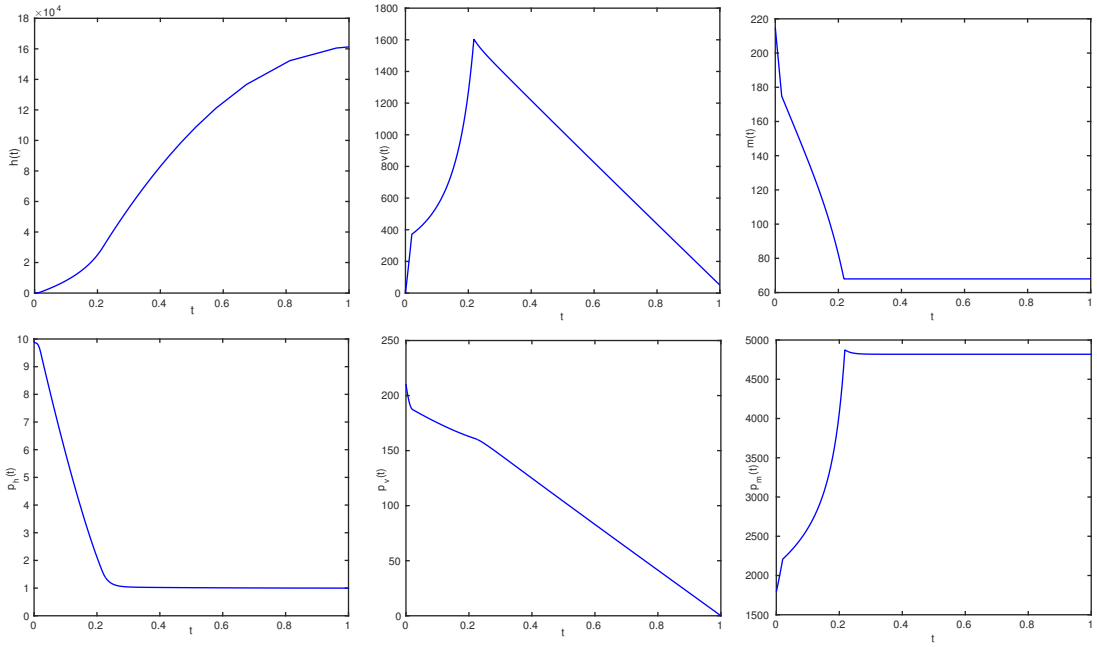

Figure 13: State (top) and co-state (bottom) solution.

4.3 An example of changing structure: an homotopy on t_f.

Our goal, in this section, is to show how `HamPath` can handle changing structures in the resolution of problems like this one. To do so, we will use t_f as our homotopic parameter, the same way we used λ in the simple shooting problem (cf. page 14). We will use the same vector of parameters as in the Bang-Singular-Bang case (cf. section 4.2).

Main goal.

- Solve (P_{t_f}) for $t_f \leqslant 70$.
- Show the switching times as functions of t_f. This gives the evolution of the structure with respect to the homotopic parameter.

For $t_f = 70$, the solution is Bang-Singular-Bang as for $t_f \approx 206$ (see section 4.2). We start by solving the problem for $t_f = 70$ and then we decrease t_f. We will see that the structure from Bang-Singular-Bang becomes Bang-Bang and then is only Bang. To capture the information that the structure has to change when t_f get smaller, we use `mfun.f90` file which gives the possibility to check some conditions during the homotopy process. For instance, if $t_1 < t_2$ denote the switching times then we can check if the condition $t_1 < t_2$ is satisfied along the path of zeros.

To get this example, type in your terminal: `hampath -example goddardBB-BSB`.

4.3.1 User implementation of `hfun` Fortran routines.

This implementation is exactly the one developed in the treatment of the Bang-Singular-Bang solution (cf. section 4.2.1)

4.3.2 User implementation of `sfun` Fortran routines.

We need here four shooting functions. Two for the Bang-Bang (BB) and Bang-Singular-Bang (BSB) cases presented in sections 4.1 and 4.2. We need also two more shooting functions, one for the intermediate case between BB and BSB solutions and another one at the limit of the BB solutions when they become simply a Bang arc. This last strategy happens when the final time t_f is exactly the time needed to reached the final mass m_f with a maximal thrust. The two first shooting functions are already given, see equations (13) and (14), and the last two are clear, see the fortran file `sfun.f90` from this example.

afun.f90

We are using the same `afun.f90` file as in the section 4.2.

sfun subroutine in sfun.f90

The `sfun.f90` file used in this case is a concatenation of previous `sfun.f90` files, see section 4.1.2 and section 4.2.2, with the two other shooting functions. See this file to get details on how the shooting functions are implemented. We present in the following listing only how one can deal with several shooting functions using the number of variables, *i.e.* the length ny of shooting variables, and the vector par which encodes the structure, see section 4.1.1. This is an example since here we only need ny.

```
    structure  = 0
    do i=1,npar-nparmin
        structure = structure + nint(abs(par(nparmin+i)))
    end do

    !Limit case
    !Bang with one contact of order 1 with the switching manifold
    if(structure.eq.1 .and. ny.eq.4) then

    !Case: 1 0
    !Bang-Bang
    else if(structure.eq.1 .and. ny.eq.10) then

    !Limit case: 1 0
    !Bang-Bang with one contact of order 2 with the switching manifold
    else if(structure.eq.1 .and. ny.eq.11) then

    !Case: 1 2 0
    !Bang-Singular-Bang
    else if(structure.eq.3 .and. ny.eq.17) then

    end if
```

Shooting function in `sfun.f90`

4.3.3 User implementation of the `mfun.f90` file.

In order to stop the homotopy when the structure changes, a monitoring interface can be coded in an optional `mfun.f90` file. Here, we know that the problem is first Bang-Singular-Bang. Let denote by $t_1(t_f) < t_2(t_f)$ the switching times. We want to stop the homotopy if for a certain value of t_f we have $t_1(t_f) > t_2(t_f)$. For smaller values of t_f we know that the structure is Bang-Bang with one switching $t_1(t_f)$. Again, we stop the homotopy if $t_1(t_f) > t_f$. Check the `mfun.f90` file for details on **`mfun`** command. We can see in the following listing that the homotopy will stop with flag = -11 or -12 if a change occurs. To make the homotopy stop one has to give the flag a value less than -10.

```
np = 1 ! Get the last point from the path of zeros, ie y and par
call getPointsFromPath(ny,npar,np,arclengths,ys,pars)

t1 = ys(4,1) ! The first switching time

!Bang-Singular-Bang case
if(ny.eq.17) then

    t2 = ys(11,1) ! The second switching times

    !if t1>t2, it means the structure has changed from BSB to BB
    if(t1-t2.ge.0d0)then
        flag = -11 ! hampath will exit with flag = -11
    end if
```

```
!Bang-Bang case
else if(ny.eq.10) then

    tf = pars(1,1) ! The final time

    !if t1>tf, it means the structure has changed from BB to B
    if(t1-tf.ge.0d0)then
        flag = -12 ! hampath will exit with flag = -12
    end if

end if
```

Monitoring function during homotopy in `mfun.f90`

4.3.4 User implementation of the `main` file.

Here we give some parts of the main file. See the `main.m` file (or `main.py` or `main.f90`) to get more details. The strategy with respect to t_f is given figure 14.

```
% Initial guess
n       = 3;                    % Dimension state
t0      = 0.0;                  % Initial time
t0norm  = 0.0;                  % Normalized initial time
tf      = 70;                   % Final time
tfnorm  = 1.0;                  % Normalized final time
t1      = 5.0;                  % First guessed switching time
t1norm  = (t1-t0)/(tf-t0);      % Normalized first guessed switching time
t2      = 25.0 ;                % Second guessed switching time
t2norm  = (t2-t0)/(tf-t0);      % Normalized second guessed switching time
q0      = [0.0 0.0 214.839];    % Initial state h_0 v_0 m_0
p0      = [5.0 100.0 500.0];
% par = [ tf alpha beta g0 umax c t0 q0 mf 1 2 0]
par     = [tf 0.01227 0.000145 9.81 9.52551 2060 t0 q0(1) q0(2) q0(3)
    67.9833 1 2 0]';
options = hampathset            % Hampath options
par0=par; parf=par; indtf=1; parf(indtf)=0.0; % Homotopy from tf = 70 to 0
nparmin = 11;

% Initial guess
[ tout, z, flag] = exphvfun([t0norm t1norm t2norm], [q0 p0]', [t0norm
    t1norm t2norm tfnorm], options, par);
z1      = z(:,2);               % z1 = z(t1, z0)
z2      = z(:,3);               % z2 = z(t2, z(t1, z0))
yGuess  = [p0 t1 z1' t2 z2'];   % yGuess = [p0, t1, z1, t2, z2]

% First shooting in the Bang-Singular-Bang case
[y0,ssol,nfev,njev,flag] = ssolve(yGuess,options,par);

% First homotopy with Bang-Singular-Bang structure
parspan = [par0 parf];
[parout,yout,sout,~,~,~,~,flag] = hampath(parspan,y0,options);
```

```
if(flag==-11) % Change in the structure detected: from BSB to BB

    % Shooting to get the exact value of tf at the change of structure
    % The limit structure is Bang-Bang with H1(tf) = H01(tf) = 0
    par          = [parout(1:nparmin,end); 1; 0];
    yGuess       = yout(1:10,end); yGuess(11)  = par(indtf);
    [y0,ssol,nfev,njev,flag] = ssolve(yGuess,options,par);
    t1BSB = y0(4); tfBSB = y0(11);

    % Second homotopy with Bang-Bang structure
    par(indtf)  = tfBSB % Value of tf when the structure changes
    par0 = par; parf = par0; parf(indtf) = 0.0; parspan = [par0 parf];
    [parout,yout,sout,~,~,~,flag] = hampath(parspan,y0(1:10),options);

    if(flag==-12) % Change in the structure detected: from BB to B

        % Shooting to get the exact value of tf at the change of structure
        % The limit structure is Bang with H1(tf) = 0
        par          = [parout(1:nparmin,end); 1];
        yGuess       = yout(1:3,end); yGuess(4)   = par(indtf);
        [y0,ssol,nfev,njev,flag] = ssolve(yGuess,options,par); tfBB=y0(4);

    end;

end;
```

main file

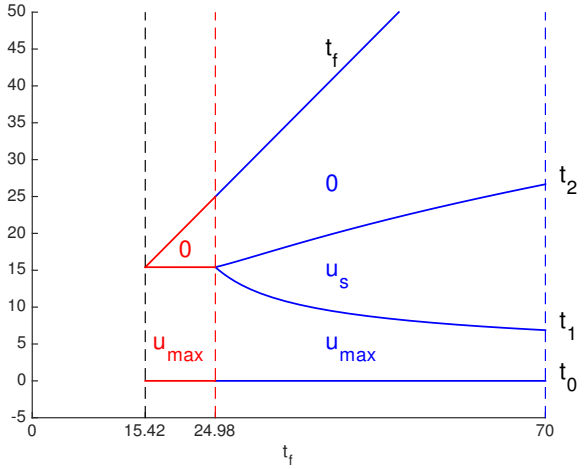

Figure 14: Homotopy on t_f. The structure is Bang-Bang for $t_f \in (15.42, 24.98]$ and Bang-Singular-Bang for $t_f \in [24.98, 70]$. The time $t_f = 15.4170957775489$ (to be accurate) is the exact time to reach the final mass m_f with maximal thrust. Hence, for smaller t_f the final condition $m(t_f) = m_f$ cannot be satisfied.

5 Install file

```
========================================================================
           How to install HamPath on Linux or Mac OS X
========================================================================

========================================================================
                          0 - Requirements
========================================================================

You must read the user guide before proceeding with this installation

Before installing Hampath, you must have an up-to-date
Java Runtime Environment (JRE), the Tapenade software (you can get it
here: http://www-sop.inria.fr/tropics/), a FORTRAN Compiler and
(you don't need to have both):

    - a version (3.X at least) of Python with these librairies:
      numpy, scipy and matlplotlib; and the extension tool
      F2PY (https://sysbio.ioc.ee/projects/f2py2e/)
    - a working Matlab application and well-configured mex-files

========================================================================
                          1 - Installation
========================================================================

    - Unpack the hampath300 archive in your installation directory
      (~/install_dir for example)

    - Go into the newly extracted folder (~/install_dir/hampath300)

    - Launch the setup.sh in a terminal

    - Enter the absolute path of your Tapenade installation folder
      if asked (~/install_dir/tapenade)

    - Choose how you want to install HamPath:
      0) Fortran stand-alone: install Hampath without interface
      1) Matlab, via mex-files: install Hampath with a Matlab
          interface (make sure your mex-files are well-configured)
      2) Matlab or Octave, via text files: install HamPath
          with a Matlab or Octave interface with no mex command
      3) Python, via f2py tool: install HamPath with a Python
          interface which needs numpy, scipy and matplotlib
          librairies and f2py extension tool

    - HamPath will detect your configuration and allow you to change it
      if you want to (for example, you can change the detected f2py
      command if it works with a 2.X version of Python to a f2py
      working with a 3.X version of python)

    - HamPath is installed

========================================================================
                          2 - Documentation
========================================================================
```

```
First thing to read before this document: user_guide.pdf
Licence: LICENSE.txt

================================================================================
```

References

[1] A. A. Agrachev & Y. L. Sachkov, *Control theory from the geometric viewpoint*, vol **87** of *Encyclopaedia of Mathematical Sciences*, Springer-Verlag, Berlin (2004), xiv+412.

[2] 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), xxvi+388.

[3] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney & D. Sorensen, *LAPACK Users' Guide*, Soc. for Industrial and Applied Math., Philadelphia, PA, USA, third edn (1999)

[4] V. G. Boltyanskiĭ, R. V. Gamkrelidze, E. F. Mishchenko, & L. S. Pontryagin, *The mathematical theory of optimal processes*. Classics of Soviet Mathematics. Gordon & Breach Science Publishers, New York, (1986), xxiv+360.

[5] B. Bonnard, J.-B. Caillau & E. Trélat, *Cotcot: short-reference manual.* `http://apo.enseeiht.fr/cotcot`, Rapport de recherche RT/APO/05/1, Institut National Polytechnique de Toulouse, Toulouse, France (2005).

[6] B. Bonnard, J.-B. Caillau & E. Trélat, *Second order optimality conditions in the smooth case and applications in optimal control*, ESAIM Control Optim. Calc. Var., **13** (2007), no 2, 207–236.

[7] 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** (2014), no. 1, 5–45. Pdf.

[8] J.-B. Caillau, O. Cots & J. Gergaud *HamPath: on solving optimal control problems by indirect and path following methods.* `http://hampath.org`

[9] J.-B. Caillau, O. Cots & J. Gergaud, *Differential continuation for regular optimal control problems*, Optim. Methods Softw., **27** (2012), no 2, 177–196. Pdf.

[10] O. Cots, *Geometric and numerical methods for a state constrained minimum time control problem of an electric vehicle*, 35 pages, submitted.

[11] O. Cots, *Contrôle optimal géométrique : méthodes homotopiques et applications*. Thèse de doctorat, Institut Mathématiques de Bourgogne, Dijon, France, septembre 2012. Pdf.

[12] J. J. Dongarra, J. Du Croz, S. Hammarling & I. S. Duff, *A set of level 3 basic linear algebra subprograms*, ACM Trans. Math. Softw., **16** (1990), no 1, 1–17.

[13] J. J. Dongarra, J. Du Croz, S. Hammarling & R. J. Hanson, *An extended set of fortran basic linear algebra subprograms*, ACM Trans. Math. Softw., **14** (1988), no 1, 1–17.

[14] R. H. Goddard, *A Method of Reaching Extreme Altitudes, volume 71(2)*, Smithsoian Miscellaneous Collections. Smithsonian institution, City of Washington, (1919).

[15] 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).

[16] 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).

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

[18] D. H. Jacobson, M. M. Lele & J. L. Speyer, *New Necessary Conditions of Optimality for Control Problems with State-Variable Inequality Constraints*, J. Math. Anal. Appl., **35** (1971), 255–284.

[19] H. Maurer, *On optimal control problems with bounded state variables and control appearing linearly*, SIAM J. Control Optim., **15** (1971), no. 3, 345–362.

[20] H. Maurer, *Numerical solution of singular control problems using multiple shooting techniques*, Journal of optimization theory and applications, Vol.18, No.2, (1976).

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

[22] H. Seywald and E.M. Cliff., Goddard problem in presence of a dynamic pressure limit. *Journal of Guidance, Control, and Dynamics*, (1993).