Cotcot: short reference manual

Bernard Bonnard, Jean-Baptiste Caillau, Emmanuel Trélat

To cite this version:

Bernard Bonnard, Jean-Baptiste Caillau, Emmanuel Trélat. Cotcot: short reference manual. 2005. hal-00086640

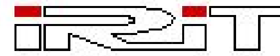
HAL Id: hal-00086640

https://hal.science/hal-00086640v1

Preprint submitted on 19 Jul 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Ecole **N**ationale **S**upérieure
d'**E**lectronique, d'**E**lectrotechnique
d'**I**nformatique, d'**H**ydraulique et de **T**élécom

Institut de **R**echerche en **I**nformatique de **T**oulouse

Cotcot: short reference manual

B. Bonnard, J.-B. Caillau and E. Trélat

Parallel Algorithms and Optimization Team
ENSEEIHT–IRIT (UMR CNRS 5505)
2 rue Camichel, F-31071 Toulouse

www.n7.fr/apo

Technical Report **RT/APO/05/1**

Cotcot: short reference manual*

B. Bonnard[†], J.-B. Caillau[‡] and E. Trélat[§]

March 2005

Abstract

This reference introduces the *Matlab* package `cotcot` designed to compute extremals in the case of smooth Hamiltonian systems, and to obtain the associated conjugate points with respect to the index performance of the underlying optimal control problem.

Keywords. Smooth Hamiltonian systems, optimal control, shooting method, conjugate points.

Classification AMS. 49-04.

1 Introduction

Consider the minimum time control of the system

$$\begin{aligned} \dot{x}_1 &= u \\ \dot{x}_2 &= 1 - u^2 + x_1^2 \end{aligned}$$

where the extremities are fixed, $x(0) = x_0$, $x(T) = x_1$, and where u is in $\mathbf{R}$. A standard application of the maximum principle tells us that the so-called *regular* minimizing curves [2] are the projection of extremals $z = (x, p)$ such that

$$\dot{z} = \overrightarrow{H}(z) \tag{1}$$

where $H(z) = p_1^2/4p_2 + p_2(1 + x_1^2)$ is the smooth *regular* Hamiltonian defined on the open subset $\Sigma = \{p_2 \neq 0\}$, and where $\overrightarrow{H} = (\partial H/\partial p, -\partial H/\partial x)$. Since we have boundary conditions, the extremals we are interested in are *BC-extremals*. They are zeros of the *shooting mapping* defined by

$$S \ : \ (T, p_0) \mapsto \Pi(\exp_T(x_0, p_0)) - x_1 \tag{2}$$

*Work supported in part by the French Space Agency through contract 02/CNES/0257/00-DPI 500.

†Institut de Mathématiques, Université de Bourgogne, BP 47870, F-21078 Dijon (`Bernard.Bonnard@u-bourgogne.fr`).

‡ENSEEIHT-IRIT (UMR CNRS 5505), 2 rue Camichel, F-31071 Toulouse (`caillau@n7.fr`).

§Laboratoire d'Analyse Numérique et EDP, Université de Paris-Sud, F-91405 Orsay (`emmanuel.trelat@math.u-psud.fr`).

with $\exp_t(z_0) = z(t, z_0)$ the solution of (1) for the initial condition z_0, $\Pi : (x, p) \mapsto x$ the canonical projection, and $p_0 \in \mathbf{R}^n$ defined up to a constant by homogeneity. Moreover, the (local) optimality of such extremals is checked by a rank test on the subspaces spanned by the *Jacobi fields* along the trajectory [2]. These fields are solutions of the variational equation

$$\delta\dot{z} = d\overrightarrow{H}(z(t))\delta z \tag{3}$$

with suitable initial conditions. The aim of the code `cotcot`, which stands for *Conditions of Order Two, COnjugate Times*, is to provide the numerical tools

1. to integrate smooth Hamiltonian systems such as (1)
2. to solve the associated shooting equation defined by (2)
3. to compute the corresponding Jacobi fields along the extremal
4. to evaluate the resulting conjugate points, if any.

We first review the installation procedure of the software in §2. Then, we illustrate in §3 the way it works on the previous example. Elementary *Matlab* code is discussed. The synopsis of the `M-files` provided are given in the appendix.

2 Installation

The package is intended for a standard *Unix* system with

– *Matlab* (version 6 or higher)

– *Adifor* (version 2.0 or higher)

– a *Fortran* compiler known as `f77`.

The automatic differentiation software *Adifor* [1] (version 2.0 or higher) is required. It is downloaded at

`www-unix.mcs.anl.gov/autodiff/ADIFOR`

The `cotcot` installation procedure is performed in three steps.

Step 1. Retrieve and uncompress the `cotcot` archive at the following URL:

`www.n7.fr/apo/cotcot.zip`

Step 2. From the parent directory `cotcot/` simply run the command `make` so as to generate the code and compile it. Basically, the *Fortran* code defining the Hamiltonian is automatically differentiated twice and the associated MEX-files for *Matlab* are generated.

Step 3. Go into the folder `main/`, launch *Matlab*, and try the command `main`. Among other printing, you should get Fig. 1 as well as the following final result (check):

```
tcs =

   3.14159265358980    6.28318530717959    9.42477796076938 12.56637061435917
```

The computation performed is analyzed in the next section.

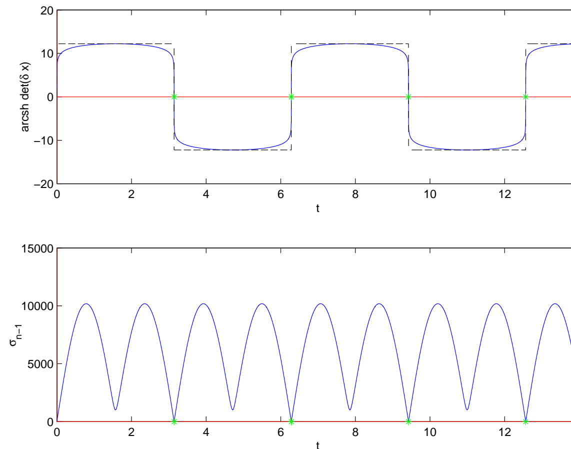


Figure 1: Result of the command `main`.

3 Tutorial example

We go back to the initial example provided in §1 and proceed in five steps.

Defining the Hamiltonian. The only user provided code is the *Fortran* subroutine defining the Hamiltonian H of the system (1). The subroutine must be stored in the file `f77/hfun.F`, and its signature must be

```
SUBROUTINE HFUN(T, N, Z, LPAR, PAR, H)
```

Obviously, the Hamiltonian may be time dependent. Moreover, additional parameters may be used (see remark 3.3). In our case, the code essentially amounts to:

```
X1 = Z(1)
X2 = Z(2)
P1 = Z(3)
P2 = Z(4)

H  = P1**2/(4.0D0*P2) + P2*(1.0D0+X1**2)
```

Note that, for the sake of robustness, dimensions are checked (`MEXERRMSGTXT` calls). Go back to the parent directory and run the command `make`. The Hamiltonian equation (1) and the variational system (3) are generated by automatic differentiation, and compiled to produce MEX-files callable from *Matlab*.

Remark 3.1. The dimension n must be lower or equal to the half of the constant `N2MAX` (maximum value of $2n$) defined in `include/constants.h`. An error during the *Matlab* run is generated otherwise. In this case, just update the value of the constant properly and generate the code again.

Computing extremals. Go to the `matlab/` subfolder and launch *Matlab*. The mapping $\exp_t$ is computed by `exphvfun`. Try

```
T  = 10
x0 = [  0 0 ]'
p0 = [  1 1 ]'
z0 = [ x0; p0 ]
odeopt = rkf45set
z  = exphvfun([ 0 T ], z0, odeopt)
```

Remark 3.2. The underlying algorithm is *Netlib* Runge-Kutta one-step ODE integrator `RKF45` [3] whose parameters are managed thanks to `rkf45get` and `rkf45set`, and then passed to `exphvfun`.

Computing BC-extremals. We assume that we are in the *normal case* [2] and normalize the adjoint covector by prescribing the Hamiltonian level to $H = 1$. As a result, the shooting mapping (2) is evaluated according to

$$S(T, p_0) = (\Pi(\exp_T(x_0, p_0)) - x_1, -1 + H(x_0, p_0))\,.$$

The Hamiltonian is computed by `exphvfun`. Try

```
h = hfun(0, z0)
```

Accordingly, denoting $\xi = (T, p_0)$, the shooting function is defined by (see `main/sfun.m`):

```
T  = xi(1);
p0 = xi(2:end);

z0 = [ x0; p0 ];
[ z, iflag ] = exphvfun([0 T], z0, options);
z1 = z(:, end);
s  = z1(1:2) - x1;
h  = hfun(0, z0);
s  = [ s; -1+h ];
```

Remark 3.3. Any number of additional parameters can be passed to `hfun`. All of them must be real, or real matrices. They are vectorized to form one row vector which is the `PAR` argument of the *Fortran* subroutine `HFUN`. Furthermore, all *Matlab* commands provided in the package (`exphvfun, expdhvfun...`), accept such additional parameters that will be passed to the Hamiltonian.
Try

```
xi = [ T; p0 ]
x1 = [ 10 0 ]'
s  = sfun(xi, odeopt)
```

Zeros of the shooting mapping can be computed by any available solver, *e.g.* `fsolve`, or the faster and more robust function `hybrd` which is a *Matlab* port of *Netlib* `HYBRD` Newton solver provided with the `cotcot` package. On our example, convergence is obtained with the initial guess $T = 10$ and $p_0 = (1, 1)$:

```
nleopt = hybrdset
xii = [ 10 1 1 ]'
xi  = hybrd('sfun', xii, nleopt, odeopt)
T   = xi(1)
p0  = xi(2:end)
```

Remark 3.4. As for the ODE integrator (see remark 3.2), `HYBRD` parameters are managed with `hybrdget` and `hybrdset`.

Computing Jacobi fields. First define the initial value of the Jacobi field, for instance according to

```
[ dummy, dp0 ] = gram(p0)
dz0 = [ 0; 0; dp0 ]
```

so that $\delta p(0)$ belongs to the tangent space $T_{p(0)}\mathbf{S}^{n-1}$ (Gram-Schmidt orthonormalization, `help gram`) because of the normalization of the initial covector (equivalent to prescribing $|p(0)|$ since we are in the normal case, see [2]). Since we must integrate the variational system *along* the previous extremal, the standard trick is to integrate both systems, Hamiltonian and variational, with the relevant initial conditions. Therefore, we extend the system and *left-concatenate* the extremal to the Jacobi field:

```
z0  = [ x0; p0 ]
dz0 = [ z0 dz0 ]
```

The sibling of `exphvfun` for the (extended) variational system is `expdhvfun`. Try

```
dz  = expdhvfun([ 0 T ], dz0, odeopt)
z1  = dz(:, 3)
dz1 = dz(:, 4)
```

More generally, a full basis of Jacobi fields is computed exactly the same way by providing a matrix instead of a single vector. At each point, the image of the upper half matrix is the subspace whose rank must be tested for conjugate points. The command `expdhvfun` then returns the concatenation of these matrices (each of them been extended by the extremal, concatenated *to the left*) at each point of the time array `t` (standard vectorized input/output).

Computing conjugate points. The conjugate point test consists in checking a rank condition. To this end, a singular value decomposition is performed, see function `main/draw.m`. For minimum time problems, in the regular case it is equivalent to find a zero of the determinant of the projections of Jacobi fields on the x-space with the dynamics [2]. Here the test is *e.g.*, at the final point,

```
dx = dz1(1:2, :)
hv = hvfun(T, z1)
det([ dx hv(1:2, :) ])
```

where `hvfun` computes $\overrightarrow{H}$. The function `dfun` evaluates this determinant at an arbitrary time t for given initial conditions (see `main/dfun.m`). Hence, conjugate points are computed by finding its roots. As before, we use the `hybrd` solver and finally get (with an initial guess of 3.0 for t_c):

```
tci = 3.0           % initialization
tc  = hybrd('dfun', tci, nleopt, 0, dz0, odeopt)
```

Indeed, the first conjugate time of our system is $t_{c,1} = \pi$. The code `main/main.m` computes several conjugate points in this way.

4 Credits

The authors are grateful to *Adifor* and *Netlib* people for making their codes available.

`www-unix.mcs.anl.gov/autodiff/ADIFOR`
`netlib.enseeiht.fr`

A Synopsis

The following `M-files` are provided with the package:

- `hfun.m`
- `hvfun.m`
- `exphvfun.m`
- `expdhvfun.m`
- `hybrd.m`
- `hybrdset.m`
- `hybrdget.m`
- `hybrd.m`
- `rkf45set.m`
- `rkf45get.m`

hfun

```
function h = hfun(t, z, varargin)
% hfun -- Hamiltonian.
%
%  Usage
%    h = hfun(t, z, p1, ...)
%
%  Inputs
%    t      real, time
%    z      real vector, state and costate
%    p1     any, optional argument
%    ...
%
%  Outputs
%    h      real, Hamiltonian at time t
%
%  Description
%    Computes the Hamiltonian.
%
```

hvfun

```
function hv = hvfun(t, z, varargin)
% hvfun -- Vector field associated to H.
%
%  Usage
%    hv = hvfun(t, z, p1, ...)
%
%  Inputs
%    t      real, time
%    z      real vector, state and costate
%    p1     any, optional argument
%    ...
%
%  Outputs
%    hv     real matrix, vector H at time t
%
%  Description
%    Computes the Hamiltonian vector field associated to H.
%
```

exphvfun

```
function [ exphv, iflag ] = exphvfun(t, z0, options, varargin)
% exphvfun -- Exponential of hv
%
%  Usage
%    [ exphv, iflag ] = exphvfun(t, z0, options, p1, ...)
%
%  Inputs
%    t       real, time
%    z0      real vector, initial flow
%    options struct vector, options
%    p1      any, optional arguments passed to hvfun
%    ...
%
%  Outputs
%    exphv  real matrix, flow at time t
%    iflag  integer, ODE solver output (should be 2)
%
%  Description
%    Computes the exponential of the Hamiltonian vector field hv
%    defined by h.
%
%  See also
%    expdhvfun, rkf45set, rkf45get
%
```

expdhvfun

```
function [ expdhv, iflag ] = expdhvfun(t, dz0, options, varargin)
% exphvfun -- Exponential of dhv/dz
%
%  Usage
%    [ expdhv, iflag ] = expdhvfun(t, dz0, options, p1, ...)
%
%  Inputs
%    t       real, time
%    dz0     real matrix, initial flow
%    options struct vector, options
%    p1      any, optional arguments passed to dhvfun
%    ...
%
%  Outputs
%    expdhv real matrix, flow at time t
%    iflag  integer, ODE solver output (should be 2)
%
%  Description
%    Computes the exponential of the variational system associated to hv.
%
%  See also
%    exphvfun, rkf45set, rkf45get
%
```

hybrd

```
function [ x, y, iflag, nfev ] = hybrd(nlefun, x0, options, varargin)
% hybrd -- Hybrid Powell method.
%
%  Usage
%    [ x, y, iflag, nfev ] = hybrd(nlefun, x0, options, p1, ...)
%
%  Inputs
%    nlefun  string, function y = nlefun(x, p1, ...)
%    x0      real vector, initial guess
%    options struct vector, options
%    p1      any, optional argument passed to nlefun
%    ...
%
%  Outputs
%    x       real vector, zero
%    y       real vector, value of nlefun at x
%    iflag   integer, hybrd solver output (should be 1)
%
%  Description
%    Matlab interface of Fortran hybrd. Function nlefun must return
%    a column vector.
%
%  See also
%    hybrdset, hybrdget
%
```

hybrdget

```
function value = hybrdget(options, name)
% hybrdget -- Gets hybrd options.
%
%  Usage
%    value = hybrdget(options, name)
%
%  Inputs
%    options struct, options
%    name    string, option name
%
%  Outputs
%    value   any, option value
%
%  Description
%    Options are:
%      xTol   - Relative tolerance between iterates      [        1e-8 ]
%      MaxFev - Max number of function evaluations       [ 800 x (n+1) ]
%      ml     - Number of banded Jacobian subdiagonals   [         n-1 ]
%      mu     - Number of banded Jacobian superdiagonals [         n-1 ]
%      EpsFcn - Used for FD step length computation      [           0 ]
%      diag   - Used for scaling if mode = 2             [  [1 ... 1]' ]
%      mode   - Automatic scaling if 1, manual if 2      [           1 ]
%      factor - Used for initial step bound              [         1e2 ]
%
%  See also
%    hybrd, hybrdset
%
```

hybrdset

```
function options = hybrdset(varargin)
% hybrdset -- Sets hybrd options.
%
%  Usage
%    options = hybrdset(name1, value1, ...)
%
%  Inputs
%    name1   string, option name
%    value1  any, option value
%    ...
%
%  Outputs
%    options struct, options
%
%  Description
%    Options are:
%      xTol   - Relative tolerance between iterates      [        1e-8 ]
%      MaxFev - Max number of function evaluations       [ 800 x (n+1) ]
%      ml     - Number of banded Jacobian subdiagonals   [         n-1 ]
%      mu     - Number of banded Jacobian superdiagonals [         n-1 ]
%      EpsFcn - Used for FD step length computation      [           0 ]
%      diag   - Used for scaling if mode = 2             [  [1 ... 1]' ]
%      mode   - Automatic scaling if 1, manual if 2      [           1 ]
%      factor - Used for initial step bound              [         1e2 ]
%
%  See also
%    hybrd, hybrdget
%
```

rkf45get

```
function value = rkf45get(options, name)
% rkf45get -- Gets rkf45 options.
%
%  Usage
%    value = rkf45get(options, name)
%
%  Inputs
%    options struct, options
%    name    string, option name
%
%  Outputs
%    value   any, option value
%
%  Description
%    Options are:
%      AbsTol - Absolute error tolerance [ 1e-14 ]
%      RelTol - Relative error tolerance [  1e-8 ]
%
%  See also
%    rkf45, rkf45set
%
```

rkf45set

```
function options = rkf45set(varargin)
% rkf45set -- Sets rkf45 options.
%
%  Usage
%    options = rkf45set(name1, value1, ...)
%
%  Inputs
%    name1   string, option name
%    value1  any, option value
%    ...
%
%  Outputs
%    options struct, options
%
%  Description
%    Options are:
%      AbsTol - Absolute error tolerance [ 1e-14 ]
%      RelTol - Relative error tolerance [  1e-8 ]
%
%  See also
%    rkf45, rkf45get
%
```

References

[1] C. Bischof, A. Carle, P. Kladem, and A. Mauer. Adifor 2.0: Automatic Differentiation of Fortran 77 Programs. *IEEE Computational Science and Engineering*, 3(3):18–32, 1996.

[2] B. Bonnard, J.-B. Caillau, and E. Trélat. Second order optimality conditions and applications in optimal control. *in preparation*, 2005.

[3] L. F. Shampine, H. A. Watts, and S. Davenport. Solving non–stiff ordinary differential equations—the state of the art. Technical Report sand75-0182, Sandia Laboratories, Albuquerque, New Mexico, 1975.