INSTITUT NATIONAL POLYTECHNIQUE DE TOULOUSE

Ecole Nationale Supérieure d'Electrotechnique, d'Electronique, d'Informatique, d'Hydraulique et de Télécommunications

TfMin: Short Reference Manual

J. B. Caillau, J. Gergaud and J. Noailles

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

Technical Report **RT/APO/01/3**

TFMIN – SHORT REFERENCE MANUAL*

J. B. CAILLAU†, J. GERGAUD† AND J. NOAILLES†

Abstract. This is a short guide to use the `Fortran 77` and *Matlab* package `TfMin` designed for the numerical solution of continuous 3D minimum-time orbit transfer around the Earth (with free final longitude), especially for low thrust engines. The underlying method is single shooting. The *Matlab* interface with the solver allows the user to define the problem and to draw the min-time orbits computed.

Key words. continuous minimum-time orbit transfer, low-thrust transfer, single shooting, continuation

AMS subject classifications. 49-04, 70Q05

1. Introduction. The 3D minimum time transfer of a satellite around the Earth is considered. The optimal control model given by the French Space Agency is the following (we refer to [3] for details[1]):

$$
\begin{array}{l}
t_f \to \min \\
(x, m) \in \mathrm{W}^{1,\infty}_7([0, t_f]),\ u \in \mathrm{L}^{\infty}_3([0, t_f]) \\
\dot{x} = f_0(x) + 1/m\ \sum_{i=1}^{3} u_i f_i(x),\ t \in [0, t_f] \\
\dot{m} = -\beta|u| \\
x(0) = x^0,\ m(0) = m^0,\ h(x(t_f)) = 0 \\
|u| \leq T_{max}
\end{array}
$$

The norms $|.|$ above are Euclidean so that, for instance, the control constraint can be rewritten

$$|u_1|^2 + |u_2|^2 + |u_3|^2 \leq T_{max}^2$$

where T_{max} is the maximum modulus of the thrust. The state of the satellite is described by the geometry of the ellipse osculating to the trajectory and its position on it, $x = (P, e_x, e_y, h_x, h_y, L)$, and by its mass m. The four vector fields that define the dynamics are:

$$f_0 = \sqrt{\mu^0/P} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ W^2/P \end{bmatrix}$$

*This work was supported in part by the French Ministry for Education, Higher Education and Research (grant 20INP96) and by the French Space Agency (contracts 871/94/CNES/1454 and 86/776/98/CNES/7462).

†ENSEEIHT–IRIT, UMR CNRS 5505, 2 rue Camichel, F-31071 Toulouse (`caillau@enseeiht.fr`, `gergaud@enseeiht.fr`, `jnoaille@enseeiht.fr`).

[1]The paper is available at the following Web address: `www.enseeiht.fr/apo/tfmin`

$$f_1 = \sqrt{P/\mu^0} \begin{bmatrix} 0 \\ \sin L \\ -\cos L \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$f_2 = \sqrt{P/\mu^0} \begin{bmatrix} 2P/W \\ \cos L + (e_x + \cos L)/W \\ \sin L + (e_y + \sin L)/W \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$f_3 = 1/W \ \sqrt{P/\mu^0} \begin{bmatrix} 0 \\ -Ze_y \\ Ze_x \\ C/2 \ \cos L \\ C/2 \ \sin L \\ Z \end{bmatrix}$$

with

$$\begin{aligned} W &= 1 + e_x \cos L + e_y \sin L \\ Z &= h_x \sin L - h_y \cos L \\ C &= 1 + h_x^2 + h_y^2. \end{aligned}$$

The boundary conditions are given by:

$$\begin{aligned} x^0 &= (P^0, e_x^0, e_y^0, h_x^0, h_y^0, L^0) \in \mathbf{R}^6 \\ h(x) &= (P - P^f, e_x - e_x^f, e_y - e_y^f, h_x - h_x^f, h_y - h_y^f) \in \mathbf{R}^5 \end{aligned}$$

and

P^0	=	11625 km	P^f	=	42165 km
e_x^0	=	0.75	e_x^f	=	0
e_y^0	=	0	e_y^f	=	0
h_x^0	=	0.0612	h_x^f	=	0
h_y^0	=	0	h_y^f	=	0
L^0	=	π	L^f	free	
m^0	=	1500 kg	m^f	free	

The two constants β (related to the specific impulsion of the engine) and μ^0 (gravitation constant of the Earth) are respectively taken equal to:

$$\begin{aligned} \beta &= 1.42.10^{-5} \text{ km}^{-1}.\text{h} \\ \mu^0 &= 398600.47 \text{ km}^3.\text{s}^{-2} \end{aligned}$$

The `Fortran 77` kernel (see §4) of the software uses two public libraries:

- *Minpack*: the ODE solver `RKF45` by H. A. Watts and L. F. Shampine, and the NLE solver `HYBRD` by B. S. Garbow, K. E. Hillstrom and J. J. More.

- *Adifor* [1]: automatic differentiation is called to generate the right hand side of the boundary value problem.

We are grateful to the authors for making their software available.

The document is organised in the following way: the installation procedure of the package is described in §2; the usage of the *Matlab* package `TfMin` and of the underlying `Fortran 77` solver `shoot` are detailed in §3 and §4, respectively. Several examples are provided. The synopses of the `Fortran 77` files is finally given in appendix A.

2. Installation. The installation procedure is performed in five steps:

1. Retrieve the compressed archive `tfmin-v1.zip` at the following Web address:
   `www.enseeiht.fr/apo/tfmin`
2. Uncompress and unarchive it (`unzip tfmin.zip`). Check with the `readme` file that the contents is correct.
3. From the parent directory `tfmin-v1`, go into the subdirectory src, edit the appropriate makefile (e.g.: `makefile.sol` if you are working on a *Solaris* workstation), and set the variable `LADI` properly (for instance by setting the environment variables `AD_LIB` and `AD_OS`). If you do not have *Adifor* (version 2.0 or higher), install it first. The package can be downloaded at:
   `www-unix.mcs.anl.gov/autodiff/ADIFOR`
4. Go back to the parent directory, and do a `make`, precising the architecture (e.g.: `make sol` if you are on a *Solaris* workstation; `make` alone will indicate the architectures currently supported).
5. Go into the `matlab` subdirectory, launch *Matlab* (version 4.0 or higher) and try the command `tfmin`. A menu should prompt as below.

```
>> tfmin

            tfmin - Minimum time orbit transfer

            1. Create initial data
            2. Call the solver
            3. Make graphs and prints
            4. Demo
            5. Help

            0. Quit

Choice ?
```

3. Usage of the Matlab package TfMin. The *Matlab* package `TfMin` is an interface for the `Fortran 77` code `shoot`. Though `shoot` is designed to handle general boundary value problems by single shooting (see §4), the interface is aimed at solving the specific minimum time transfer problem of §1. Roughly, it allows the user to pre and postprocess the data for the `Fortran 77` code.

Here is a brief review of `TfMin`'s entries:

1. `Create initial data`. Generates input files for the solver. The user can specify:
   - the main physical values, `Tmax` to `hyf`.
   - the initial guesses for the shooting unknowns, `zi1` to `zi7`:
     - `zi1` is t_f (the unknown transfer time)

  - `zi2` is p_P^0 (value at $t = 0$ of the costate adjoint to P)
  - `zi3` is $p_{e_x}^0$ (value at $t = 0$ of the costate adjoint to e_x)
  - `zi4` is $p_{e_y}^0$ (value at $t = 0$ of the costate adjoint to e_y)
  - `zi5` is $p_{h_x}^0$ (value at $t = 0$ of the costate adjoint to h_x)
  - `zi6` is $p_{h_y}^0$ (value at $t = 0$ of the costate adjoint to h_y)
  - `zi7` is p_L^0 (value at $t = 0$ of the costate adjoint to L)
- the scalings for each component of the state-costate vector $y = (x, p)$ (`scalei` is the scaling for y_i).
- the number `nint` of integration steps for the solution.
- the name of the generated input file.

The values in brackets are default values. They correspond to values suited for the resolution of the transfer problem with a maximum thrust T_{max} of 60 Newtons and boundary conditions exactly as in §1. The default name for the generated file is `in.dat`.

2. `Call the solver`. Calls the solver using the indicated files as input/output files.
3. `Make graphs and prints`. Reads output files to draw *Matlab* graphs (3 types: states and controls in Fig. 1, orbit in Fig. 2, costates and switching function (see [3]) in Fig. 3; see Figs. 3.1 to 3.3).
4. `Demo`. Solves the problem with physical values as in §1 for the following thrusts (continuation technique, see [3]):

$$60, 24, 12, 9, 6, 3, 2, 1.4, 1, 0.7, 0.5, 0.3, 0.2, 0.14$$

   The result files are written into the subdirectory `matlab/out`.
5. `Help`. This document (allows you to configure your PDF browser).

Here are four examples:

1. Generate the default file `in.dat` using defaults values of entry 1. of the main menu. Call the solver (entry 2.), save the results in the default file `out.dat`, and draw the results. Compare with figures 3.1, 3.2 and 3.3. The transfer time for $T_{max} = 60$ Newtons sould be $t_f \simeq 14.800$ hours.
2. Generate a new input file using the defaults values (`Tmax` equal to 60 Newtons, etc.), except for `hx0`: do `hx0 (6.120000000000000e-02) ? tan(0.5 * 50 * pi/180)` (initial inclination of 50 degrees instead of 7). Call the solver and draw the graphs as before. You should get an optimal transfer time around 21.331 hours.
3. Call the solver (entry 2.) with input file `in/in1.dat` (initial data for 1 Newton computation). Save the result in the default `out.dat`, and edit this file. Generate a new input file (entry 1.) with `Tmax` equal to 0.8 Newton, default values anywhere else except for the `zi`'s: use the heuristic described in [3] for the initial guess of t_f (`zi1`), together with the values of the solutions for 1 Newton read on the line `# Z =...` of the file `out.dat` as below (continuation technique).

```
zi1 (1.520550000000000e+01) ? 853.310792 * (1 / 0.8)
zi2 (3.612660000000000e-01) ? -22.153197
zi3 (2.224120000000000e+01) ? -43.4710948
zi4 (7.877360000000000e+00) ? 0.961318881
zi5 (0.000000000000000e+00) ? 318.180099
zi6 (0.000000000000000e+00) ? -2.30723609
zi7 (-5.908020000000000e+00) ? -0.579786311
```

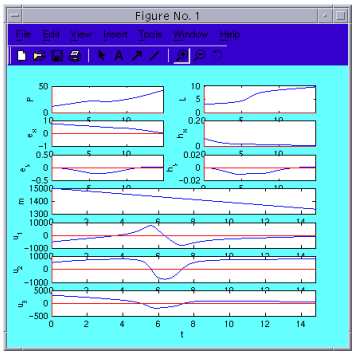


FIG. 3.1. *Optimal states and controls for* 60 *Newtons.*

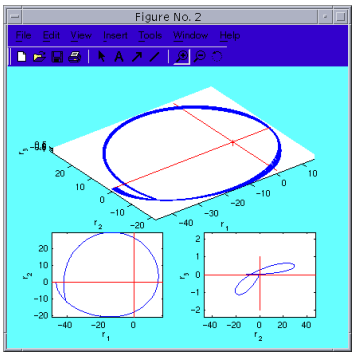


FIG. 3.2. *Minimum time orbit for* 60 *Newtons.*

You should get an optimal transfer time around 1068.8 hours.

4. Finally, run the demo (entry 4.) to generate all the results for the standard minimum time problem and thrusts from 60 Newtons down to 0.14 Newton. Check for files in the subdirectory `matlab/out`. Results obtained on various architectures are detailed in table 3.1.

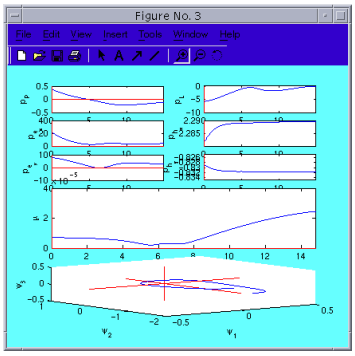


FIG. 3.3. *Costates and switching function for* 60 *Newtons.*

TABLE 3.1
Results on various architectures (HP PA-C160, Ultra Sparc 5 under Solaris, Pentium III 933 Mhz under Linux86). Transfer times are in hours, execution times in seconds. The norm of the shooting function $|S|$ *at the solution is given.*

	t_f			$\|S\|$			Exec		
T_{max}	HP	Solaris	Linux86	HP	Solaris	Linux86	HP	Solaris	Linux86
60	14.800	14.800	14.800	1e-8	1e-8	1e-8	2	2	1
24	34.716	34.716	34.716	3e-11	1e-10	1e-10	5	6	3
12	70.249	70.249	70.249	9e-9	9e-9	9e-9	6	7	3
9	93.272	93.272	93.272	4e-10	4e-10	4e-10	14	15	7
6	141.22	141.22	141.22	1e-9	1e-9	1e-9	10	12	6
3	285.77	285.77	285.77	4e-10	4e-10	4e-10	29	31	16
2	425.61	425.61	425.61	7e-12	4e-12	1e-12	40	46	23
1.4	606.13	606.13	606.13	1e-10	4e-13	9e-13	86	70	31
1	853.31	853.31	853.31	2e-12	7e-12	9e-13	79	91	47
0.7	1214.5	1214.5	1214.5	2e-9	2e-9	2e-9	115	133	67
0.5	1700.9	1700.9	1700.9	1e-8	6e-8	1e-8	167	188	101
0.3	2878.8	2878.8	2874.5	4e-9	1e-10	3e-8	292	339	167
0.2	4260.6	4269.0	4260.6	6e-8	9e-8	1e-8	425	594	249
0.14	6079.5	6079.5	6079.5	4e-8	1e-7	7e-9	815	591	279

4. Usage of the Fortran package shoot. The solver is aimed at solving general boundary value problems of the following type:

$$\dot{y} = \xi(t, y),\ t \in [t_0, t_f]$$
$$y_i(t_0) = y_i^0,\ i \in I^0,\ b_2(y(t_f)) = 0$$

where $y = (x, p)$, $I^0 \subset \{1, \ldots, 2n\}$ $(\mathrm{Card}(I^0) = n)$ being the set of fixed initial components, and where $b_2 : \mathbf{R}^{2n} \to \mathbf{R}^n$ defines the terminal boundary condition. This structure for boundary conditions is typical of a large class of BVP's arising in optimal control that can be handled by single shooting. This is indeed the method implemented here which has proved to be very efficient in the orbit transfer case when coupled with a continuation technique [3]. In order to solve more general problems with multiple shooting, the celebrated code BNDSCO [4] is available.

One of the specificity of the code is the use of automatic differentiation to generate the right hand side of the BVP. The user has only to provide the dynamics

$$\dot{x} = f(t, x, u)$$

of the underlying optimal control together with the mapping $u(t, x, p)$ giving the control as a function of the state and the costate. Then, *Adifor* computes ξ such that

$$\xi(t, y) = (f(t, x, u(t, x, p)), -\nabla_x H(t, x, u(t, x, p), p))$$

(where $H(t, x, u, p)$ is the canonical Hamiltonian). This approach is particularly time and error saving in the transfer case where the state has seven components[2]. For a similar approach based on *Maple* and *Scilab*, see [2].

The code reads a file named `in.dat` and produces two files called `out.dat` and `next.dat`, structured as below:

- `bin/in.dat`:
  - `ZI`: n values (double precision) as initial guesses for the shooting unknowns.
  - `FREE0`: n indexes (integer) defining the *free* components of y at t_0 (complement of I^0).
  - `Y0`: the $2n$ components of y^0 (the components not in I^0–that is components in `FREE0`–are unused and can be set set to any value).
  - `T0`: the initial time t_0 (double precision).
  - `TF`: the final time t_f (double precision).
  - `PAR`: `LPAR` parameters (double precision) needed by the user to define the dynamics (T_{max}, β... in the transfer case; see description of file `pbdef.h` below).
  - `NIT`: number (integer) of reintegration step to generate the result file `out.dat` (see also §3).
  - `SCALE`: $2n$ values (double precision) used to scale each component of y (see also §3).
- `bin/out.dat`:
  - summary of values read in `in.dat`: `ZI, FREE0, Y0, T0, TF, PAR, NIT, SCALE`
  - values from the computation:
    - `Z`: n shooting values (double precision) at solution.
    - `S`: n components (double precision) of the shooting function at solution.
    - `|S|`: 2-norm (double precision) of the shooting function at solution.
    - `TEXEC`: execution time in seconds (double precision).
    - `Nle Solver`: nonlinear equation solver (default: *Minpack* `HYBRD`).
    - `Ode Solver`: ordinary differential equation solver (default: *Minpack* `RKF45`).
  - reintegration of the solution: `NIT+1` values of (t, x, p, u) between t_0 and t_f (uniform discretization).
- `bin/next.dat`: same as `in.dat`, `ZI` being remplaced by `Z` (n shooting values at solution); the file is intended to serve as a new input file in a continuation process.

[2]Though the variation of the mass can be reduced to $m(t) = m^0 - \beta T_{max} t$ [3], there are still seven components since the unknown transfer is treated as a new state variable $\dot{t}_f = 0$.

In order to define a new problem, the user has only to modify 5 files:

- `include/pbdef.h`: defines the constants of the problem such as dimensions:
  - `N`: the state dimension n.
  - `M`: the control dimension m.
  - `LPAR`: the number of auxiliary parameters passed, for instance, to the dynamics (array `PAR`); must be at least 1.
- `bin/in.dat`: gives the appropriate values for the parameters according to the format described before.
- `src/b2fun.F`: gives the `Fortran 77` code of the function $b_2 : \mathbf{R}^{2n} \to \mathbf{R}^n$ defining the terminal boundary condition.
- `src/ffun.F`: defines the function f in $\dot{x} = f(t, x, u)$; if necessary, uses auxiliary parameters (double precision array `PAR`).
- `src/ufun.F`: defines the control as a function of t, x and p.

Finally, the user can also specify its own solvers, for ordinary differential equations and nonlinear equations, by changing the four files below:

- `include/algodef.h`: defines the constants used by the solvers (such as work array lengths for the integrator, tolerance, and so on).
- `src/pfun.F`: calls `RKF45` of *Minpack* by default; any other integrator can be used.
- `src/ssolve.F`: calls `HYBRD` of *Minpack* by default; any other solver can be used.
- `src/term.F`: the user has to update in an obvious way the lines that print the names of the solvers used (C preprocessor style is prefered).

Appendix A. Synopses of Fortran files.

List of `Fortran 77` subroutines:

```
SUBROUTINE B2FUN(Y,B2)
SUBROUTINE ELTIME(T)
SUBROUTINE FFUN(T,X,U,F)
SUBROUTINE HFUN(T,X,P,U,H)
SUBROUTINE INIT(ZI)
SUBROUTINE MAIN(ZI,Z,TEXEC)
SUBROUTINE PFUN(Z,Y)
SUBROUTINE SFUN(Z,S)
SUBROUTINE SFUN_I(NEQ,Z,S,IFLAG)
SUBROUTINE SSOLVE(ZI,Z)
SUBROUTINE TERM(ZI,Z,TEXEC)
SUBROUTINE UFUN(T,X,P,U)
SUBROUTINE XIFUN(T,Y,XI)
```

b2fun

```
      SUBROUTINE B2FUN(Y,B2)
      IMPLICIT NONE
C     Written on Fri Apr  6 17:30:30 MET DST 2001
C     by Jean-Baptiste Caillau - ENSEEIHT-IRIT, UMR CNRS 5505
C
C     Function b2 = b2fun(y). Computes the boundary condition (at
C     final instant) of the BVP. **** Must be set by the user****.
C     See also FFUN and UFUN.

C     .. Global ..
#include "pbdef.h"

C     .. Arguments ..
C     Y                (input) DOUBLE PRECISION array, dimension 2*N
C                      Flow of BVP at final instant
C     B2               (output) DOUBLE PRECISION array, dimension N
C                      Boundary condition
```

eltime

```
      SUBROUTINE ELTIME(T)
      IMPLICIT NONE
C     Written on Mon Jul  2 11:52:16 MET DST 2001
C     by Jean-Baptiste Caillau - ENSEEIHT-IRIT, UMR CNRS 5505
C
C     Function t = eltime(). Returns the elapsed time.

C     .. Global ..

C     .. Arguments ..
C     T                (output) DOUBLE PRECISION
C                      Elapsed time
```

ffun

```
      SUBROUTINE FFUN(T,X,U,F)
      IMPLICIT NONE
C     Written on Fri Apr  6 17:30:30 MET DST 2001
C     by Jean-Baptiste Caillau - ENSEEIHT-IRIT, UMR CNRS 5505
C
C     Function y = ffun(t,x,u). Computes the dynamics. **** Must be
set
C     by the user ****. See also UFUN and B2FUN.

C     .. Global ..
#include "pbdef.h"

C     .. Arguments ..
C     T                (input) DOUBLE PRECISION
C                      Time
C     X                (input) DOUBLE PRECISION array, dimension N
C                      State
C     U                (input) DOUBLE PRECISION array, dimension M
C                      Control
C     F                (output) DOUBLE PRECISION array, dimension N
C                      Second member of the dynamics
```

hfun

```
      SUBROUTINE HFUN(T,X,P,U,H)
      IMPLICIT NONE
C     Written on Fri Apr  6 17:30:30 MET DST 2001
C     by Jean-Baptiste Caillau - ENSEEIHT-IRIT, UMR CNRS 5505
C
C     Function h = hfun(t,x,p,u). Hamiltonian of the problem.

C     .. Global ..
#include "pbdef.h"

C     .. Arguments ..
C     T                (input) DOUBLE PRECISION
C                      Time
C     X                (input) DOUBLE PRECISION array, dimension N
C                      State
C     P                (input) DOUBLE PRECISION array, dimension N
C                      Adjoint state
C     U                (input) DOUBLE PRECISION array, dimension M
C                      Control
C     H                (output) DOUBLE PRECISION
C                      Hamiltonian
```

init

```
      SUBROUTINE INIT(ZI)
      IMPLICIT NONE
C     Written on Fri Apr  6 17:30:30 MET DST 2001
C     by Jean-Baptiste Caillau - ENSEEIHT-IRIT, UMR CNRS 5505
C
C     Function zi = init(). Initializes zi and the global variables
C     (see pbdef.h and algodef.h). Reads file ./in.dat with the
C     following format:
C     ZI FREE0 Y0 T0 TF PAR NIT SCALE
```

main

```
      SUBROUTINE MAIN(ZI,Z,TEXEC)
      IMPLICIT NONE
C     Written on Fri Apr  6 17:30:30 MET DST 2001
C     by Jean-Baptiste Caillau - ENSEEIHT-IRIT, UMR CNRS 5505
C
C     Function [z texec] = main(zi). Returns the solution of the
C     shooting problem together with the elapsed time for its
C     computation.

C     .. Global ..
#include "pbdef.h"

C     .. Arguments ..
C     ZI               (input) DOUBLE PRECISION array, dimension N
C                      Initial guess for the shooting unknown
C     Z                (output) DOUBLE PRECISION array, dimension N
C                      Shooting solution
C     TEXEC            (output) DOUBLE PRECISION
C                      Execution time
```

pfun

```
      SUBROUTINE PFUN(Z,Y)
      IMPLICIT NONE
C     Written on Fri Apr  6 17:30:30 MET DST 2001
C     by Jean-Baptiste Caillau - ENSEEIHT-IRIT, UMR CNRS 5505
C
C     Function y = pfun(z). Computes the evaluation at the final
C     instant of the maximal flow associated with the BVP for the
C     initial condition defined by z. See also XIFUN.

C     .. Global ..
#include "pbdef.h"
#include "algodef.h"

C     .. Arguments ..
C     Z                (input) DOUBLE PRECISION array, dimension N
C                      Shooting unknown
C     Y                (output) DOUBLE PRECISION array, dimension 2*N
C                      Flow at final instant
```

sfun

```
      SUBROUTINE SFUN(Z,S)
      IMPLICIT NONE
C     Written on Fri Apr  6 17:30:30 MET DST 2001
C     by Jean-Baptiste Caillau - ENSEEIHT-IRIT, UMR CNRS 5505
C
C     Function s = sfun(z). Computes the shooting function for the
C     initial condition defined by z. See also PFUN, B2FUN.

C     .. Global ..
#include "pbdef.h"
#include "algodef.h"

C     .. Arguments ..
C     Z                (input) DOUBLE PRECISION array, dimension N
C                      Shooting unknown
C     S                (output) DOUBLE PRECISION array, dimension N
C                      Shooting value
```

sfun_i

```
      SUBROUTINE SFUN_I(NEQ,Z,S,IFLAG)
      IMPLICIT NONE
C     Written on Fri Apr  6 17:30:30 MET DST 2001
C     by Jean-Baptiste Caillau - ENSEEIHT-IRIT, UMR CNRS 5505
C
C     Function [s iflag] = sfun_i(neq,z,iflag). Interface for SFUN
C     used by the NLE solver. See also SFUN.

C     .. Global ..
#include "pbdef.h"

C     .. Arguments ..
C     NEQ               (input) INTEGER
C                       Number of equations
C     Z                 (input) DOUBLE PRECISION array, dimension N
C                       Shooting unknown
C     S                 (output) DOUBLE PRECISION array, dimension N
C                       Shooting value
C     IFLAG             (input/output) INTEGER
C                       Flag from the NLE solver
```

ssolve

```
      SUBROUTINE SSOLVE(ZI,Z)
      IMPLICIT NONE
C     Written on Fri Apr  6 17:30:30 MET DST 2001
C     by Jean-Baptiste Caillau - ENSEEIHT-IRIT, UMR CNRS 5505
C
C     Function z = ssolve(z_i). Solves the shooting equation
C     S(z) = 0 with initial guess z_i.

C     .. Global ..
#include "pbdef.h"
#include "algodef.h"

C     .. Arguments ..
C     ZI               (input) DOUBLE PRECISION array, dimension N
C                      Initial guess for z
C     Z                (output) DOUBLE PRECISION array, dimension N
C                      Shooting solution
```

term

```
      SUBROUTINE TERM(ZI,Z,TEXEC)
      IMPLICIT NONE
C     Written on Fri Apr  6 17:30:30 MET DST 2001
C     by Jean-Baptiste Caillau - ENSEEIHT-IRIT, UMR CNRS 5505
C
C     Function term(zi,z,texec). Writes the result into the files
C     ./out.dat and ./next.dat (same format as in.dat). ./out.dat
C     contains all the information required to reproduce the
C     computation plus an evaluation of T, Y, U (solution) at NIT
C     steps.

C     .. Global ..
#include "pbdef.h"
#include "algodef.h"

C     .. Arguments ..
C     ZI               (input) DOUBLE PRECISION array, dimension N
C                      Initial guess for the shooting unknown
C     Z                (input) DOUBLE PRECISION array, dimension N
C                      Shooting solution
C     TEXEC            (input) DOUBLE PRECISION
C                      Execution time
```

ufun

```
      SUBROUTINE UFUN(T,X,P,U)
      IMPLICIT NONE
C     Written on Fri Apr  6 17:30:30 MET DST 2001
C     by Jean-Baptiste Caillau - ENSEEIHT-IRIT, UMR CNRS 5505
C
C     Function u = u(t,x,p). Computes the control as a function of t,x
C     and p (maximum principle). **** Must be set by the user ****.
C     See also FFUN, B2FUN.

C     .. Global ..
#include "pbdef.h"

C     .. Arguments ..
C     T                (input) DOUBLE PRECISION
C                      Time
C     X                (input) DOUBLE PRECISION array, dimension N
C                      State
C     P                (input) DOUBLE PRECISION array, dimension N
C                      Adjoint state
C     U                (output) DOUBLE PRECISION array, dimension M
C                      Control
```

xifun

```
      SUBROUTINE XIFUN(T,Y,XI)
      IMPLICIT NONE
C     Written on Fri Apr  6 17:30:30 MET DST 2001
C     by Jean-Baptiste Caillau - ENSEEIHT-IRIT, UMR CNRS 5505
C
C     Function xi = xifun(t,y). Defines the second member \xi(t,y) of
C     the BVP by automatic differentiation. See also FFUN, UFUN and
C     HFUN.

C     .. Global ..
#include "pbdef.h"
#include "algodef.h"

C     .. Arguments ..
C     T                 (input) DOUBLE PRECISION
C                       Time
C     Y                 (input) DOUBLE PRECISION array, dimension 2*N
C                       State and adjoint state
C     XI                (output) DOUBLE PRECISION array, dimension 2*N
C                       Second member of the BVP
```

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 (1996), pp. 18–32.

[2] F. Bonnans and S. Maurin, *An implementation of the shooting algorithm for solving optimal control problems*, report 0240, INRIA, Rocquencourt, France, May 2000.

[3] J. B. Caillau, J. Gergaud, and J. Noailles, *3D Geosynchronous Transfer of a Satellite: Continuation on the Thrust*, Submitted to Journal of Optimization Theory and Applications, (2001).

[4] H. J. Oberle and W. Grimm, *BNDSCO: A Program for the Numerical Solution of Optimal Control Problems*, report 515, Institute for Flight Systems Dynamics, German Aerospace Research Establishment DLR, Oberpfaffenhofen, Germany, 1989.