OPTRAGEN 2.0

A MATLAB Toolbox for Optimal Trajectory Generation.

Raktim Bhattacharya

raktim@aero.tamu.edu

Aerospace Engineering Department,
Texas A&M University,
College Station, TX 77843-3141.

October 5, 2007

1 Introduction

OPTRAGEN is not a medicine :-), but a MATLAB toolbox that numerically solves optimal control problems. OPTRAGEN translates optimal control problems of the form

$$\min_{x(t),u(t)} \Phi_i(x(t_0),u(t_0),t_0) + \int_{t_0}^{t_f} \Phi_t(x(t),u(t),t)d\tau + \Phi_f(x(t_f),u(t_f),t_f)$$

subject to dynamics

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

and constraints

$$\begin{array}{ccccccc} l_i & \leq & \Psi_i(x(t_0),u(t_0),t_0) & \leq & u_i & & (\textit{Initial Constraint}) \\ l_t & \leq & \Psi_t(x(t),u(t),t) & \leq & u_t & & (\textit{Trajectory Constraint}) \\ l_f & \leq & \Psi_f(x(t_f),u(t_f),t_f) & \leq & u_f & & (\textit{Final Constraint}), \end{array}$$

to nonlinear programming problems of the form

$$\min_{\rho} F(\rho)$$

subject to

$$L \quad \leq \quad \left\{ \begin{array}{c} \rho \\ A\rho \\ G(\rho) \end{array} \right\} \quad \leq \quad U.$$

The transcription of optimal control problem (OCP) to nonlinear programming (NLP) problem is done by parameterising trajectories as Splines. The output of the transcription is a cost function and a constraint function that can be interfaced with any commercially available nonlinear programming solver. OPTRAGEN can be considered to be a parser that translates optimal control problems to nonlinear programming problems, and is *not* dependent on any nonlinear programming solver.

2 Dependencies

OPTRAGEN requires the following:

- MATLAB Barebones
- MATLAB Symbolic Computing Toolbox
- MATLAB Spline Toolbox

To solve the transcribed nonlinear programming problem, a nonlinear programming solver is needed. Recommended solver is SNOPT, since the resulting nonlinear programming problem is *sparse*. Examples provided with this toolbox demonstrates interface with SNOPT. A free, student version of SNOPT is available from Philip Gill's website at Univ. California, San Diego.

3 Installation

Step 1: Unzip it in a desired directory. Avoid unzipping under directories with space in the name, e.g. `C:/Program Files/etc`. It will create directory `optragen/` under the present working directory. Under `optragen/`, there will be subdirectories:

1. `doc/` : contains documentation
2. `src/` : contains source files for OPTRAGEN
3. `examples/` : contains examples that demonstrate use of this toolbox

In the `examples/` directory, there will be directories:

1. `vanderpol/` : fixed final time optimal control problem
2. `brachistochrone/` : open final time optimal control problem
3. `pathplanning/` : obstacle avoidance in 2D

Step 2: Put `optragen/src/` in the MATLAB path.

Step 3: Put SNOPT in MATLAB path, if you are using SNOPT.

4 Changes in Version 2

The two main changes in this version are:

1. Complete vectorization of code. This has reduced the computational time considerably.
2. Dynamical constraints can now be imposed via Galerkin projections. See vanderpol example under `optragen/examples/vanderpol/vanderpol_galerkin.m`

5 Parameterisation of Trajectory Space

Trajectories in OPTRAGEN are parameterised as Splines. A Spline is defined by a set of polynomials that are stitched at predefined break points, satisfying a given degree of smoothness. Splines are approximating functions of chioce for $f(t)$, when approximation is desired over a large interval.

The large interval $[t_0, t_f]$ is subdivided into sufficiently small intervals $[\xi_i, \xi_{i+1}]$, with $t_0 = \xi_1 < \cdots < \xi_N = t_f$. The points ξ_i are called *break points* or *knot points*. On each such interval, polynomial of relatively low degree can provide a good approximation to $f(t)$. Approximation over smaller intervals offers *local support*.

The different polynomial pieces can be constructed such that the resulting composite function has several continuous derivatives over the interval boundaries or the break points. The number of continuous derivatives across the breakpoints defines the *smoothness* condition of the spline.

Remark 1: The continuity of the resulting curve *at the break points* is determined by the smoothness condition.

Remark 2: The continuity of the resulting curve *in between the break points* is determined by the order of the polynomial pieces.

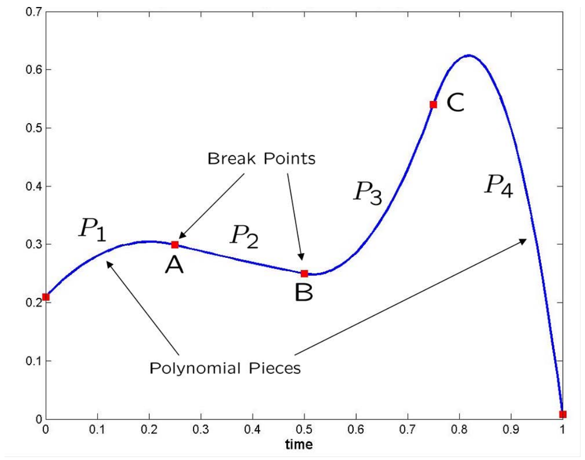

Figure 1: Splines and polynomial pieces.

Figure 1 shows a spline curve composed of four polynomial pieces P_1, P_2, P_3 and P_4. They are joined at break points A,B and C. Splines can be represented in a computationally efficient manner using normalised B-Splines. For a more information on Splines, the reader is directed to:

1. *The MATLAB Spline Toolbox Manual*
2. *Practical Guide to Splines*, by Carl de Boor, Springer Verlag Publication.

Next we present trajectory parameterisation using B-Splines.

6 B-Splines

A B-Spline curve is defined as

$$f(t) = \sum_{k=1}^{N_c} \alpha_k B_{k,r}(t)$$

where

- α_k are the free parameters or the degrees of freedom
- N_c is the number of free parameters defined by $N_c = N \times (r - s) + s$
- N is the number of intervals or the number of polynomial pieces

- r is the degree of the polynomial pieces
- s is the smoothness condition at the breakpoints
- $B_{k,r}(t)$ are the basis functions, see MATLAB Spline Toolbox manual for definition

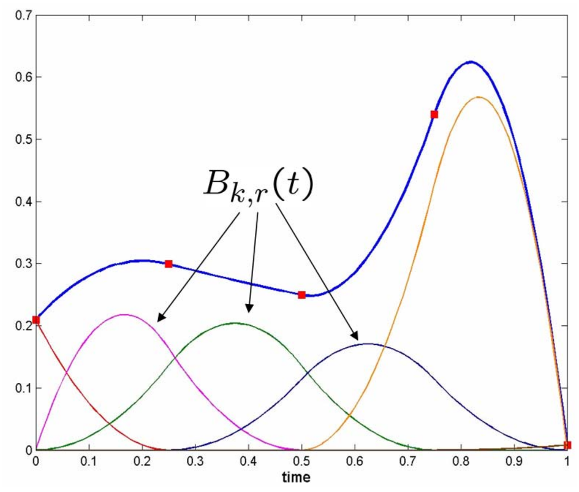

Figure 2: A Spline curve as a composite of B-Splines.

Figure 2 shows the same spline as in figure 1 as a composite of B-Splines.

7 Transcription of Optimal Control Problem to Nonlinear Programming Problem

The transcription is achieved in the following manner:

- The unknown trajectories are parameterised as B-Splines
- The optimisation problem is then written in terms of the B-Spline coefficients α_k.
- Costs and constraints are evaluated and enforced at the *collocation points*.
- The collocation points are *different* from the break points. In general, the collocation points are more dense than the break points. The optimal set of collocation points can be determined from the order of the spline using the *Gaussian Quadrature* formula. See MATLAB Spline Toolbox manual, pg 2-45 (example on solving a nonlinear ODE using Splines).

Figure 3 illustrates the transcription process.

8 Collocation

8.1 Gaussian Quadrature

The Gaussian quadrature formula seeks to obtain the best numerical estimate of an integral by picking the optimal abscissae x_i at which to evaluate $f(x)$. The fundamental theorem of Gaussian quadrature states that the optimal abscissae of the m-point Gaussian quadrature formulas are precisely the roots of the orthogonal polynomial for the same interval and weighting function. Gaussian quadrature is optimal because it fits

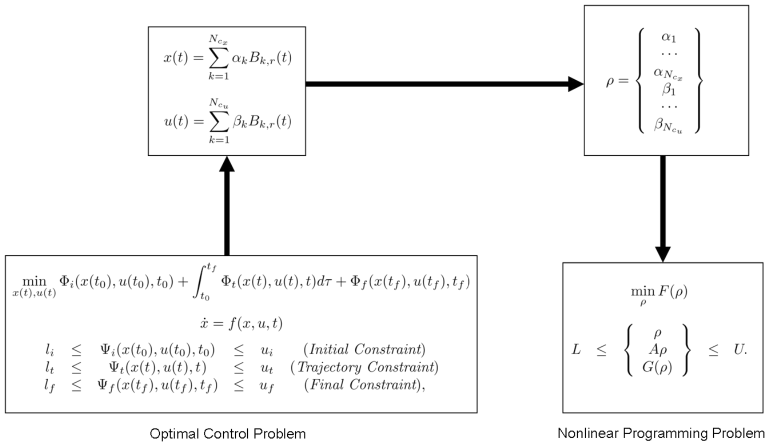
$x(t) = \sum_{k=1}^{N_{c_x}} \alpha_k B_{k,r}(t)$
$u(t) = \sum_{k=1}^{N_{c_u}} \beta_k B_{k,r}(t)$
$\rho = \left\{ \alpha_1, \dots, \alpha_{N_{c_x}}, \beta_1, \dots, \beta_{N_{c_u}} \right\}$
$\min_{x(t),u(t)} \Phi_i(x(t_0),u(t_0),t_0) + \int_{t_0}^{t_f} \Phi_t(x(t),u(t),t)d\tau + \Phi_f(x(t_f),u(t_f),t_f)$
$\dot{x} = f(x,u,t)$
$l_i \le \Psi_i(x(t_0),u(t_0),t_0) \le u_i$ (Initial Constraint)
$l_t \le \Psi_t(x(t),u(t),t) \le u_t$ (Trajectory Constraint)
$l_f \le \Psi_f(x(t_f),u(t_f),t_f) \le u_f$ (Final Constraint),
$\min_\rho F(\rho)$
$L \le \left\{ \rho, A\rho, G(\rho) \right\} \le U.$
Optimal Control Problem
Nonlinear Programming Problem

Figure 3: Transcription of OCP to NLP.

all polynomials upto degree $2m$ exactly. Slightly less optimal fits are obtained from Radau quadrature and Laguerre quadrature.

If the weighting function is chosen to be 1 and the interval $[-1, 1]$, then Legendre polynomials are the orthogonal functions whose roots are the Gaussian sites. A Legendre polynomials of order r is represented by $P_r(x)$. The first few Legendre polynomials are:

$$\begin{aligned}
P_0(x) &= 1 \\
P_1(x) &= x \\
P_2(x) &= \tfrac{1}{2}(3x^2 - 1) \\
P_3(x) &= \tfrac{1}{2}(5x^3 - 3x) \\
P_4(x) &= \tfrac{1}{8}(35x^4 - 30x^2 + 3) \\
P_5(x) &= \tfrac{1}{8}(63x^5 - 70x^3 + 15x) \\
P_6(x) &= \tfrac{1}{16}(231x^6 - 315x^4 + 105x^2 - 5)
\end{aligned}$$

8.2 Chebyshev-Gaussian-Labatto Points

Chebyshev-Gaussian-Labato (CGL) points can also be used to determine the collocation points. CGL points in the interval $[-1, 1]$ are

$$t_k = \cos(\pi k/N), k = 0, \dots, N.$$

These points are the extreme of the N^{th} order Chebyshev polynomial. The i^{th} order Chebyshev polynomial is expressed by

$$T_i(t) = \cos(i \cos^{-1} t).$$

9 Examples

9.1 Vanderpol Oscillator : Fixed Final Time

Consider the following optimisation problem based on the Vanderpol oscillator.

$$\min \int_0^5 (x_1^2 + x_2^2 + u^2)dt$$

subject to the dynamics

$$\begin{aligned}\dot{x}_1 &= x_2\\ \dot{x}_2 &= -x_1 + (1-x_1)^2 x_2 + u\end{aligned}$$

and constraints

$$\begin{aligned}x_1(0) &= 1\\ x_2(0) &= 0\\ -x_1(5) + x_2(5) &= 1\end{aligned}$$

The optimisation problem has

- one trajectory or integral cost function
- two initial linear constraints
- one final linear constraint
- one linear trajectory or path constraint
- one nonlinear trajectory or path constraint

The problem can be formulated as a numerical optimisation problem in more than one way, depending on the trajectories that are parameterised.

9.1.1 Over Parameterisation

If we choose to over parameterise the problem by parameterising $x_1(t), y(t)$ and $u(t)$ as B-Splines, then the dynamics of the system should constrain the behaviour of these trajectories. Since the constraints are satisfied at the collocation points, a sparse distribution of collocation points will result in trajectories that does not approximate the Vanderpol oscillator system satisfactorily. The optimal distribution is given by the Gaussian quadrature formula. Note that this approach will lead to a large number of equality constraints. Therefore, the optimisation problem has to have a large number of free parameters for feasibility.

9.1.2 Differential Flatness

The Vanderpol problem can be formulated as an optimisation problem on a single trajectory as compared to three trajectory variables as shown previously. The Vanderpol system exhibits a property called *differential flatness*. Choosing $z(t) \equiv x(t)$ as the flat output we can rewrite the optimisation problem as follows:

$$\min_{z(t)} \int_0^5 [z^2 + \dot{z}^2 + \{\ddot{z} + z - (1-z^2)\dot{z}\}^2] dt$$

subject to constraints

$$\begin{aligned}z(0) &= 1\\ \dot{z}(0) &= 0\\ -z(5) + \dot{z}(5) &= 1\end{aligned}$$

In this formulation, the dynamics of the system is implicitly enforced. It gets rid of the unnecessary equality path constraints which are hard to satisfy and may lead to infeasibility for a small number of B-Spline coefficients. Therefore, exploitation of differential flatness helps in reducing the problem complexity in terms of the number of constraints due to system dynamics and the number of trajectories required to define the problem.

Having said so, it is also important to note that formulation with flat outputs may lead to numerically sensitive problems. Typically, when flatness is exploited the state and conrol variables of the system are complicated algebraic functions of the flat outputs. Numerical optimisation with such functions may lead to singularity problems for naively imposed state and control constraints.

Clearly, there is a tradeoff in the choice of formulation. In some cases, it may not be a good idea to exploit flatness even if possible. It may be computationally feasible to over parameterise the system in the formulation. The choice of formulation is problem dependent and it is not possible to rank them in general in terms of computational efficiency, numerical conditioning, etc.

9.1.3 Construction of the Trajectory Space

The trajectory space is defined by the number of intervals n, the smoothness condition s and the order of the polynomial r. It is important to select these parameters carefully. For the differentially flat system we observe that the highest derivative of the flat output that appears in the formulation is two, i.e. $\ddot{z}$. If it is desired that $\ddot{z}$ is atleast twice differentiable, then the smoothness condition for $z(t)$ should be $s = 5$, resulting in $z(t) \in C^{s-1}$. The order of the spline r has to be atleast s. The number of intervals n and the order r can be tuned to make sure that the problem has sufficient degrees of freedom. The order of the spline also determines the number of collocation points necessary. This is determined from the Gaussian quadrature formula.

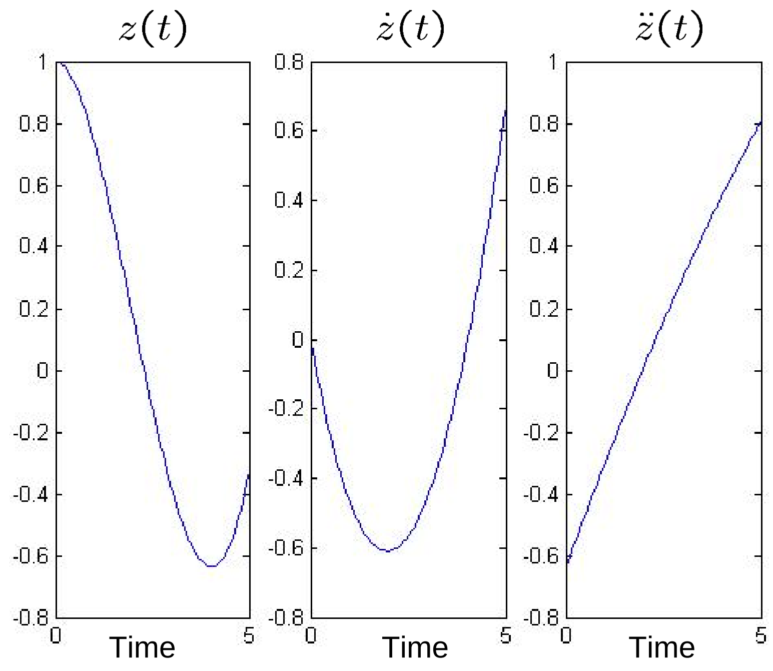

Figure 4: Fixed final time optimal control problem.

Figure 4 shows the optimal solution for $n = 1, s = 4$ and $r = 5$. The optimal cost is 2.4751 and it took 0.172 seconds to run in a Pentium 4, 2Ghz machine running Windows XP. The optimisation environment is MATLAB based. Computational time is expected to improve of the same problem is implemented using ANSI C programming language.

The solution shown in figure 4 is really the suboptimal solution, since optimality has been determined in the subspace of $z(t)$ defined by the choices of n, s and r. Changing n, s and r will change the suboptimal solution.

OPTRAGEN Implementation
The vanderpol example is at `optragen/examples/vanderpol/vanderpol.m`

9.2 The Classical Brachistochrone Problem : Open Final Time

The problem is defined as follows. A bead slides on a frictionless wire between two points $A(x_0, y_0)$ and $B(x_f, y_f)$ in a constant-gravity field. The bead has an initial velocity V_0 at point A. What is the shape of the wire that will produce a minimum-time path between the two points? For formulation details please refer to *Applied Optimal Control: Optimization, Estimation and Control* by Arthur E. Bryson, Jr. and Yu-Chi Ho.

The problem is defined by the following optimisation problem

$$\min t_f$$

such that

$$\begin{aligned}
\dot{x} &= V(y)\cos(\theta) \\
\dot{y} &= V(y)\sin(\theta) \\
V(y) &= \sqrt{V_0^2 + 2gy} \\
x(0) &= 0 \\
y(0) &= 0 \\
x(t_f) &= x_f \\
y(t_f) &= y_f
\end{aligned}$$

Since the final time is unknown, it has to be a parameter of optimisation. Constants are parameterised as splines with $n = 1, s = 0, r = 1$. The problem is then normalised with respect to time. For this we define the normalised time to be $\tau = t/t_f$, where t_f is the unknown constant. Time derivatives are scaled appropriately as

$$\frac{d}{dt} = \frac{1}{t_f}\frac{d}{d\tau}$$

The problem with normalised time is therefore

$$\min t_f$$

subject to

$$\begin{aligned}
\tfrac{dx}{d\tau} &= t_f V(y)\cos(\theta) \\
\tfrac{dy}{d\tau} &= t_f V(y)\sin(\theta) \\
V(y) &= \sqrt{V_0^2 + 2gy} \\
x(0) &= 0 \\
y(0) &= 0 \\
x(1) &= x_f \\
y(1) &= y_f
\end{aligned}$$

The problem can be solved by parametersing $\theta(\tau)$, but that will require numerical integration of the dynamics to determine $x(\tau), y(\tau)$. Numerical integration in optimisation problems tend to be time consuming and inaccurate, especially when constraints are defined on the integrated function.

For this problem we over parameterise the system and chose $x(\tau), y(\tau)$ to be the variables of optimisation. The trajectories are constrained by the dynamics. Since derivatives of splines are easily computable, derivative constraints can be easily implemented.
Figure 5 shows the suboptimal solution of the Brachistochrone problem. This is obtained using $n = 3, s = 2, r = 4$ for $x(\tau), y(\tau)$ and $n = 1, s = 0, r = 1$ for t_f. The minimum time is 0.3270 and it took 2.266 seconds to compute in a MATLAB environment.

This problem has an analytical solution. It turns out that the minimum time trajectories are cycloids where $\theta(t)$ is a linear function of time. Figure 5 shows almost linear variation for suboptimal $\theta(t)$. Deviation is due to the errors in imposing the dynamics as constraints. Increasing collocation points and number of intervals

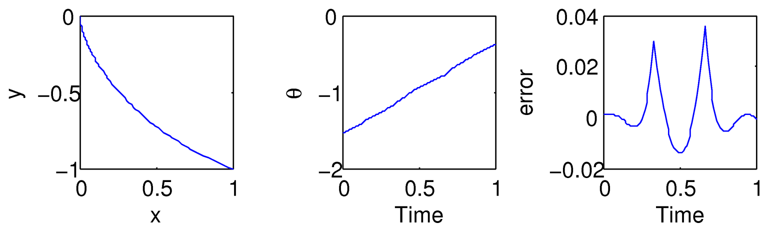

Figure 5: Open final time optimal control problem.

is expected to bring this error down, at the cost of increased computational time.

OPTRAGEN Implementation
The Brachistochrone example is at `optragen/examples/brachistochrone/brachi.m`

9.3 Robot Motion Planning : Obstacle Avoidance

In this example we consider motion planning for a robot with obstacles in 2D space. Let us assume that it is desired that the robost goes from point $A(x_0, y_0)$ to $B(x_f, y_f)$ avoiding circular obstacles in the way and using minimum energy. This is cast as the following optimisation problem.

$$\min \int_0^1 (\dot{x}^2 + \dot{y}^2) dt$$

subject to

$$\begin{aligned} \dot{x} &= V\cos\theta \\ \dot{y} &= V\sin\theta \\ x(0) &= 0 \\ y(0) &= 0 \\ x(1) &= 1.2 \\ y(1) &= 1.6 \\ (x(t)-0.4)^2 + (y(t)-0.5)^2 &\geq 0.1 \\ (x(t)-0.8)^2 + (y(t)-1.5)^2 &\geq 0.1 \end{aligned}$$

The last two constraints model circular obstacles that must be avoided.
Figure 6 shows the optimal trajectory generated with the obstacles at $(0.4, 0.5)$ and $(0.8, 1.5)$ and radius 0.3162. $n = 2, s = 2, r = 3$. The minimum cost is 5.0232 and it took 0.8590 seconds to solve the problem in a MATLAB environment.

OPTRAGEN Implementation
The path planning example is at `optragen/examples/path planning/obstacleAvoidance.m`

10 User Manual

10.1 The `nlp` Data Structure

The `nlp` data structure contains information necessary to transcribe the optimal control problem to nonlinear programming problem. It has information that is used by the cost and constraint function. It should be declared global.

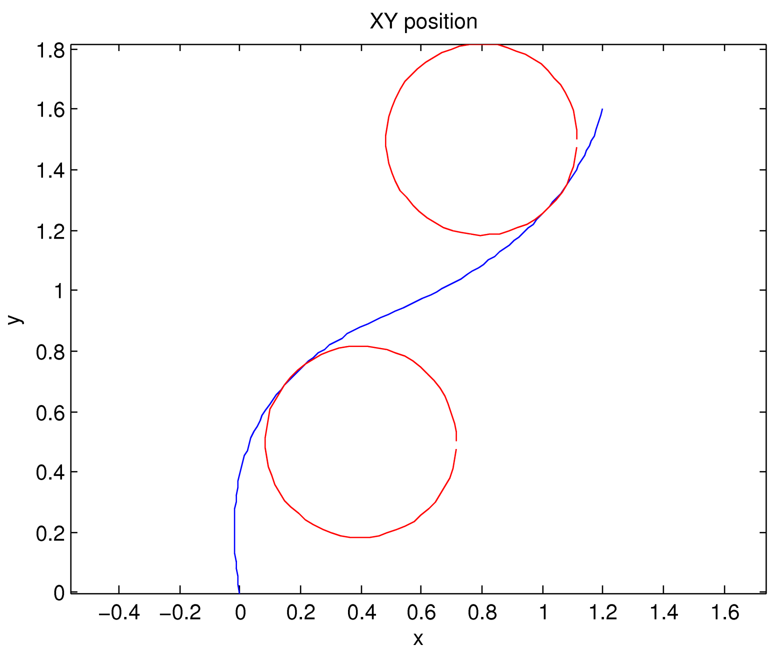

Figure 6: Suboptimal minimum energy trajectory, with obstacles.

```
EXAMPLE clear;
clc;
global nlp;
```

10.2 Defining Trajectories: `traj(n,s,r)`

```
PURPOSE
Creates a trajectory object.

SYNTAX
T = traj(n,s,r);

T = trajectory object returned.
n = (MATLAB integer) number of intervals.
s = (MATLAB integer) smoothness condition at the break points.
r = (MATLAB integer) order of the polynomial.

EXAMPLE
T = traj(5,2,3);
```

The above command creates a trajectory object with 5 polynomials, smoothness condition of 2 and order 3.

```
SEE ALSO
Vanderpol example.
```

10.3 Derivatives of Trajectories: `deriv(z)`

```
PURPOSE
Creates a trajectory object that is the derivative of another trajectory object.

SYNTAX
zd = deriv(z);

zd = trajectory object containing the derivative of the trajectory object z.

z = a trajectory object.

EXAMPLE
z = traj(5,2,3);
zd = deriv(z);
```

The above commands create a trajectory object with 5 polynomial pieces, each of order 3 and smoothness condition 2. Then it creates its derivative `zd`, which is another object with 5 polynomials, of order 2 and smoothness condition 1.

```
SEE ALSO
Vanderpol example.
```

10.4 Constraint Function:`constraint(lb,func,ub,type)`

```
PURPOSE
Defines a constraint object.

SYNTAX
C1 = constraint(lb,func,ub,type);

C1 = constraint object returned.
lb = lower bound, must be scalar.

func = string expression defining the constraint function in terms of trajectory objects and
their derivatives.

ub = upper bound, must be scalar.
type = one of the following strings :  'initial', 'trajectory', 'final'.

EXAMPLE
x = traj(3,4,5); xd = deriv(x);
y = traj(3,4,5); yd = deriv(y);
C1 = constraint(0,'xd',0,'initial'); % Initial constraint

C2 = constraint(1,'x^2+y^2',Inf,'trajectory'); % Path constraint
C = C1+C2; % Combine both constraints into a single variable

SEE ALSO
Brachistochrone example.
```

10.5 Cost Function: `cost(func,type)`

```
PURPOSE
Defines a cost object.

SYNTAX
C = cost(func,type);

C = constraint object returned.

func = string expression defining the cost function in terms of trajectory objects and their
derivatives.

type = one of the following strings :  'initial', 'trajectory', 'final'.


EXAMPLE
z = traj(3,4,5); zd = deriv(z); zdd = deriv(zd);
C1 = cost('z^2+zd^2+zdd^2','trajectory'); % Integral cost
C2 = cost('zdd+zd','final'); % Terminal cost
C = C1 + C2; % Combine two costs into a single variable
```

In a given problem, there can be only one cost of a given kind. That is, there can be only one `'initial'` cost, only one `'trajectory'` cost and only one `'final'` cost. These can be added together. Cannot add two cost variables of the same type.

```
SEE ALSO
Brachistochrone example.
```

10.6 Trajectory List: `trajList(T1,...,Tn)`

```
PURPOSE
Defines a list of trajectories.  It is used to specify the trajectories involved in a given
optimisation problem.

SYNTAX
TL = trajList(T1,...,Tn);

EXAMPLE
TL = trajList(x,xd,xdd,y,yd,z,zd);

SEE ALSO
Vanderpol example.);
```

10.7 Parameter List

```
PURPOSE
Defines a MATLAB cell array containing the variable names of parameters such as mass, length,
etc, that are used in the cost and constraint functions.

SYNTAX
P1 = {varname1,...,varnameN}
```

```
varname1,..., varnameN = MATLAB strings.

EXAMPLE
global V0 g;
...
V0 = 1; g = 9.806;
ParamList = {'V0','g'}

The variables used in the parameter list must be declared global and assigned a value.

SEE ALSO
Brachistochrone example.
```

10.8 Transription of OCP to NLP: ocp2nlp(TrajList, Cost,Constr, HL,ParamList,pathName,probName)

```
PURPOSE
Transcribe the optimal control problem to a nonlinear programming problem.

SYNTAX
nlp = ocp2nlp(TrajList, Cost, Constr, HL, ParamList, pathName, probName);

nlp = data structure containing definition of the resulting NLP.
TrajList = MATLAB object returned by trajList(...).
Cost = MATLAB object that defines the cost functions.
Constraint = MATLAB object that defines the constriant functions.
HL = Horizon length, vector containing the collocation points.Final time is taken to be HL(end).
ParamList = Parameter list, cell array of parameter variable names.
pathName = MATLAB string containing location where working files are created.
probname = MATLAB string used to prefix the names of the working files.
```

10.9 Providing Initial Guess

The nonlinear programming problem is sensitive to the initial guess provided. The solution returned is the locally optimal solution. Computational time also greatly depends on the initial guess provided. Some of the choices are,

```
init = zeros(ncoeff,1); % All zero
init = ones(ncoeff,1);  % All unity
init = rand(ncoeff,1);  % Random numbers
```

It is possible to construct initial coefficients from a trajectory. This is shown in the robotic path planning example. In that problem, the initial trajectory is assumed to be a straight line from (x_0, y_0) to (x_f, y_f), i.e.

```
Time = linspace(0,1,100);
xval = linspace(x0,xf,100);
yval = linspace(y0,yf,100);
```

We then fit splines to the data points with the number of intervals, smoothness and order that was used to parameterise $x(t)$ and $y(t)$, i.e.

```
xsp = createGuess(x,Time,xval); % x = trajectory object used to parameterise x(t)
ysp = createGuess(y,Time,yval); % y = trajectory object used to parameterise y(t)
```

Variables `xsp,ysp` are spline objects that fit the data points `(t,xval)` and `(t,yval)`. Initial guess is constructed using the command,
`init = [xsp.coefs ysp.coefs]';`

The initial guess must be a column vector. The sequence of coefficients corresponds to the trajectories in the trajectory list created using `trajList(...)`.

10.10 Calling SNOPT

The user is encouraged to read the SNOPT user manual to tweak the solver. The OPTRAGEN examples demonstrate interface with SNOPT. An example is as follows.

```
EXAMPLE
x = solution of the optimisation problem.  These are the coefficients of the B-Splines.

F = Vector containing the cost and constraint functions.  F(1) is the value of the objective.
F(2:end) are the values of the constraint functions.

inform = returns the status of the optimisation problem.  See SNOPT manual for more information.

init = initial guess for the problem.
xlow = lower bound on the parameters.
xupp = upper bound on the parameters.

LB = lower bound of the cost and constraint functions.  LB(1) = lower bound of the cost function.
LB(2:end) = lower bound of the constraint functions.

UB = upper bound of the cost and constraint functions.  UB(1) = upper bound of the cost function.
UB(2:end) = upper bound of the constraint functions.

'ocp2nlp_cost_and_constraint' = MATLAB function that is called by the nonlinear solver.  This
function in turn calls the cost and constraint functions defined by the user.  The user should
not edit this file.
```

10.11 Plotting Results

```
PURPOSE
Construct trajectory from B-Spline coefficients.

SYNTAX
sp = getTrajSplines(nlp,x);

sp = cell array of spline objects.
nlp = MATLAB object returned by ocp2nlp(...).
x = solution returned by SNOPT or any other solver.

Sequence of trajectories in sp is the sequence of non-derivative trajectories in the trajList(...).

EXAMPLE
TL = trajList(x,xd,y,yd,ydd,z);
....
% Call ocp2nlp(...)
% Call SNOPT(...)

sp = getTrajSplines(nlp,x); X = sp1; % x trajectory
Y = sp2; % y trajectory
Z = sp3; % z trajectory

Derivatives of splines are obtained as
Xd = fnder(X); Yd = fnder(Y); Zd = fnder(Z);
Xd,Yd and Zd are spline objects.

To plot spline objects use command
fnplt(X); fnplt(Y); fnplt(Z);

To get numerical values at sites TGrid,
Xval = fnval(X,TGrid);

SEE ALSO
MATLAB Spline Toolbox, Vanderpol example.
```