Tools for robotic trajectory planning using cubic splines and semi-infinite programming

A. Ismael F. Vaz and Edite M.G.P. Fernandes

Production and Systems Department - Engineering School

Minho University - Braga - Portugal

{aivaz,emgpf}@dps.uminho.pt

FGS 2004 - Avignon - France

19-24 September 2004

Outline

- Semi-Infinite Programming (SIP)
- Robotics terminology
- The trajectory planning problem
- Cubic splines
- Problems coded
- Numerical results
- Conclusions

Semi-Infinite Programming (SIP)

Standard vs Generalized

$$\min_{x \in R^n} f(x)$$

$$s.t.\ \ g_i(x,t) \leq 0,\ i = 1, ..., m$$

$$h_i(x) \leq 0,\ i = 1, ..., o$$

$$h_i(x) = 0,\ i = o + 1, ..., q$$

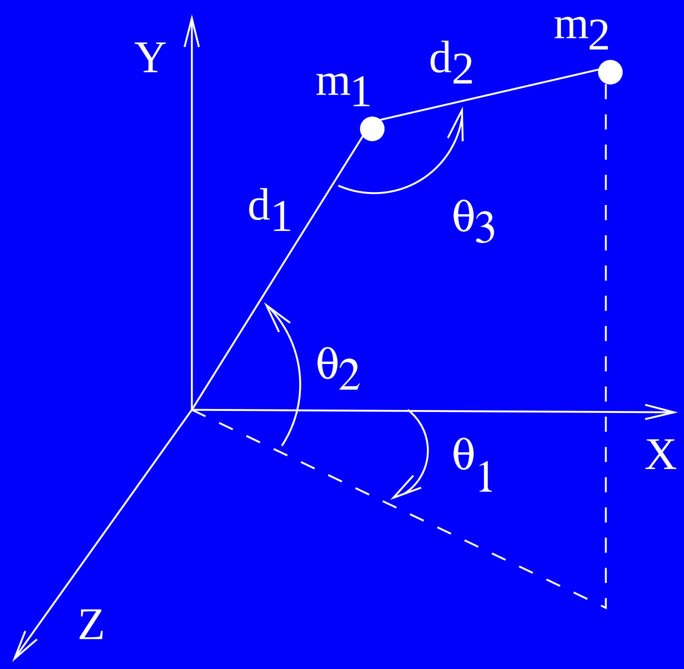
$$\forall t \in T \subset R^p$$

$f(x)$ is the objective function, $h_i(x)$ are the finite constraint functions, $g_i(x,t)$ are the infinite constraint functions and T is, usually, a cartesian product of intervals ($[\alpha_1, \beta_1] \times [\alpha_2, \beta_2] \times \cdots \times [\alpha_p, \beta_p]$)

If T depends on x ($T(x)$) then problem is called generalized SIP, otherwise it is called a standard SIP problem.

Robotics terminology

Each link has a length and mass associated (d_1, d_2, m_1 and m_2 for the example with 2 links). The ability that the robot has to position the links is called the *degrees of freedom* (d.o.f.) of the robot (3 d.o.f. in the example).



Trajectory definition

In robot joint space the specification of the values for the d.o.f. are enough to define the robot position in space. Since the robot position (d.o.f. values) varies, we can define the path as a curve

$$\theta(\tau) = [\theta_1(\tau), \theta_2(\tau), \ldots, \theta_l(\tau)]^T \quad \tau \in [0, \tau_f]$$

parametrized by τ, where l is the number of d.o.f..

Natural constraints

Possible constraints applied to the parametric curve are:

- to impose a given inicial/final velocity,

$$\frac{d\theta}{d\tau}(0) = v_i \quad \text{and} \quad \frac{d\theta}{d\tau}(\tau_f) = v_f$$

- to impose a given inicial/final acceleration/deacceleration,

$$\frac{d^2\theta}{d\tau^2}(0) = a_i \quad \text{and} \quad \frac{d^2\theta}{d\tau^2}(\tau_f) = a_f.$$

Optimal cubic polynomial joint trajectories

Assume that $[\theta_1(\tau_0), \ldots, \theta_1(\tau_n)]$, $[\theta_2(\tau_0), \ldots, \theta_2(\tau_n)]$, $\ldots$, $[\theta_l(\tau_0), \ldots, \theta_l(\tau_n)]$ are the vectors of points (knots) where the joint trajectory passes through.

The optimization consists of finding the optimum total displacements time that fits the joint trajectory by using cubic splines constrained to velocity, acceleration, jerk and torque bounds.

Some more notation

Let

- $t_0 < t_1 < \cdots < t_n$ be a time sequence where t_i is the time where the robot is in the joint position $[\theta_1(\tau_i), \ldots, \theta_l(\tau_i)]$
- $h_1 = t_1 - t_0$, $h_2 = t_2 - t_1$, ..., $h_n = t_n - t_{n-1}$ be the time displacements
- $Q_{ij}(t)$ be the cubic spline for joint i in $[t_{j-1}, t_j]$ and $Q_i(t)$ be the cubic spline for joint i.

We will use the notation $Q'(t) = \frac{dQ(t)}{dt}$ for the derivative.

Generalized SIP

The SIP problem can be formulated in the following mathematical form:

$$\begin{aligned}
&\min \sum_{j=1}^{n} h_j \equiv t_n - t_0 \\
s.t.\quad &|Q_i'(t)| \leq C_{i,1} \\
&|Q_i''(t)| \leq C_{i,2} \\
&|Q_i'''(t)| \leq C_{i,3} \\
&|F_i(t)| \leq C_i, \quad i = 1, ..., l \\
&h_j > 0 \;\; j = 1, ..., n; \\
&\forall t \in [t_0, t_n]
\end{aligned}$$

where $C_{i,1}$, $C_{i,2}$, $C_{i,3}$ and C_i are the bounds for the velocity, acceleration, jerk and torque, respectively, on joint i.

Torque expression

The expression for the manipulator's torque is

$$F_i(t) = J_i n_i Q_i''(t) + B_i n_i Q_i'(t) + \frac{1}{n_i}\left(\sum_{j=1}^{l} I_{ij}(Q(t))Q_j''(t) + \sum_{j=1}^{l}\sum_{k=1}^{l} C_{ijk}(Q(t))Q_j'(t)Q_k'(t) + d_i(Q(t))\right)$$

where for the ith robot joint

Torque expression (cont.)

J_i=motor inertia $(J_i > 0,\ i = 1, \ldots, l)$;

n_i=gear ratio;

B_i=viscous damping coefficient $(B_i > 0,\ i = 1, \ldots, l)$;

$(I_{ij}(Q(t)))_{i,j=1,\ldots,l}$=inertia matrix (positive definite);

$(C_{ijk}(Q(t)))_{i,j,k=1,\ldots,l}$=Coriolis tensor;

$d_i(Q(t))$=gravitational torque.

Reformulation as standard SIP

$$\min \sum_{j=1}^{n} h_j$$

$$s.t. \quad \left| Q_i'\left(\tau \sum_{k=1}^{n} h_k + t_0\right)\right| \leq C_{i,1}$$

$$\left| Q_i''\left(\tau \sum_{k=1}^{n} h_k + t_0\right)\right| \leq C_{i,2}$$

$$\left| Q_i'''\left(\tau \sum_{k=1}^{n} h_k + t_0\right)\right| \leq C_{i,3}$$

$$\left| F_i\left(\tau \sum_{k=1}^{n} h_k + t_0\right)\right| \leq C_i, \quad i = 1, ..., l$$

$$h_j > 0, \quad j = 1, ..., n, \quad \forall \tau \in [0, 1].$$

using the linear transformation $t = \tau \sum_{k=1}^{n} h_k + t_0$.

Cubic splines

Assume, for clarity of notation, only one joint. $Q(t)$ is the function that approximates the joint trajectory $f(t)$ and $Q_i(t)$, $i = 1, \ldots, n$, are the cubic spline segments that approximate the function in $[t_{i-1}, t_i]$.

Given a finite number of data points, $f_0 = f(t_0)$, . . . , $f_n = f(t_n)$, a C-Spline is formed by n cubic polynomials ($Q_i(t)$, $i = 1, \ldots, n$) that interpolate the given data points. The set of the $Q_i(t)$, $i = 1, \ldots, n$ will provide a cubic approximation to the function $f(t)$. Since $Q_i(t)$ is a cubic polynomial, the second derivatives w.r.t. t can be expressed as

$$Q_i''(t) = \frac{t_i - t}{t_i - t_{i-1}} M_{i-1} + \frac{t - t_{i-1}}{t_i - t_{i-1}} M_i, \quad i = 1, \ldots, n$$

where M_i is the second derivative of $f(t)$ at t_i.

Cubic splines (cont.)

Integrating $Q_i''(t)$ twice and imposing the (continuity) conditions $Q_i(t_{i-1}) = f_{i-1}$ and $Q_i(t_i) = f_i$ results in the following interpolating functions:

$$\begin{aligned} Q_i(t) =& \frac{M_{i-1}}{6h_i}(t_i - t)^3 + \frac{M_i}{6h_i}(t - t_{i-1})^3 \\ &+ \left(\frac{f_{i-1}}{h_i} - \frac{h_i M_{i-1}}{6}\right)(t_i - t) \\ &+ \left(\frac{f_i}{h_i} - \frac{h_i M_i}{6}\right)(t - t_{i-1}), \quad i = 1, \dots, n, \end{aligned}$$

where $h_i = t_i - t_{i-1}$.

The C-Spline is completely defined if the M_i, $i = 0, \dots, n$, are known.

Cubic splines (cont.)

Imposing the continuity of the first derivative, $Q'_i(t_i) = Q'_{i+1}(t_i)$, $i = 1, \ldots, n-1$, results in a tridiagonal system from where the M_i, $i = 1, \ldots, n-1$ can be obtained.

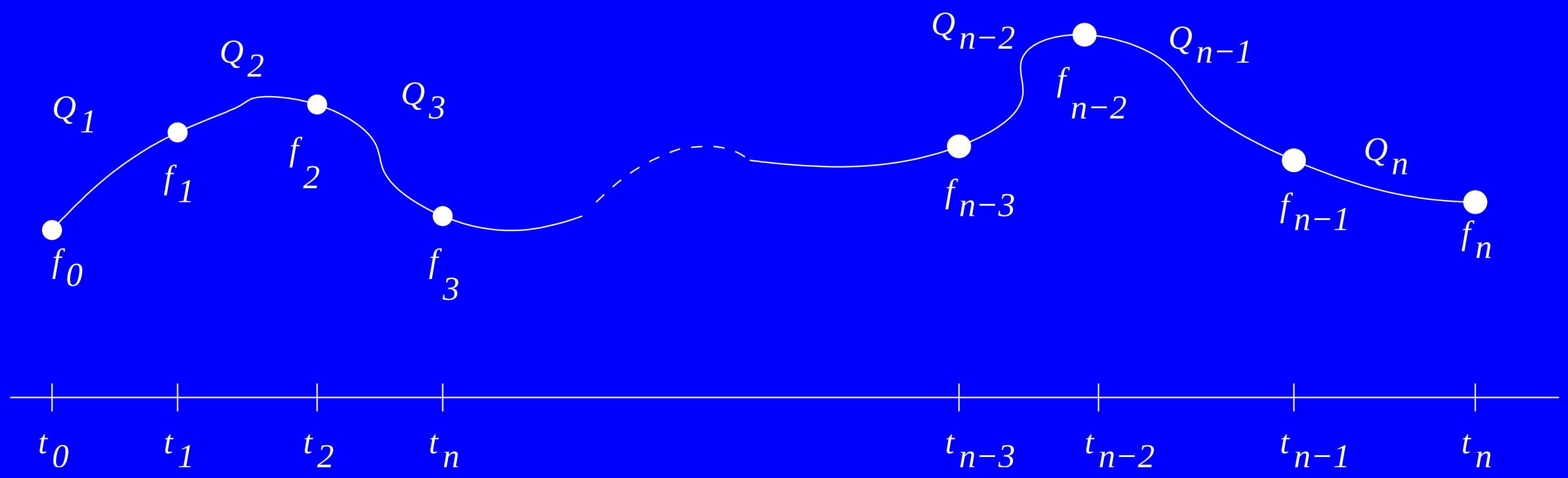
If we assume $M_0 = M_n = 0$ we have a natural cubic spline.

If we assume $Q'_1(t_0) = f'_0$ and $Q'_n(t_n) = f'_n$ we have a complete cubic spline.

Problems to approximate trajectories need to specify the first and second derivatives at the extreme of the spline (initial/final velocity and acceleration).

Cubic splines (cont.)

Two more degrees of freedom are necessary and the goal is achieved by considering two extra knots where the f values are not specified.



Cubic splines (cont.)

Consider, without loss of generality, that t_1 and t_{n-1} are the extra knots.

Solving

$$Q_1'(t_0) = v_i$$
$$Q_n'(t_n) = v_f$$

for the two unknowns f_1 and f_{n-1} results in

$$f_1 = f_0 + h_1 v_i + \frac{h_1^2 M_0}{3} + \frac{h_1^2 M_1}{6}$$

$$f_{n-1} = f_n - v_f h_n + \frac{h_n^2 M_n}{3} + \frac{h_n^2 M_{n-1}}{6}.$$

Cubic splines (cont.)

Replacing f_1 and f_{n-1} in the natural C-Spline tridiagonal system results in a new tridiagonal system also to be solved for the M_i, $i = 1, \ldots, n-1$, unknowns.

AMPL

AMPL (Algebraic Modeling Programming Language) is a modeling language for mathematical programming.

AMPL

- does not allow semi-infinite programming problems to be coded;
- is unable to solve a linear system.

AMPL is commercial software but a student edition is available for evaluation (http://www.ampl.com).

Splines dynamic library for AMPL

An external dynamic splines library was built for AMPL, providing B (out of the talk) and C-Splines.

The function syntax is

$$\texttt{cspline}\ (t, d, n, h_1, h_2, \ldots, h_n, f_1, f_2, \ldots, f_{n-1}, v_i, v_f, a_i, a_f)$$

where t and $h_1, \ldots, h_n$ are AMPL variables. d, n, $f_1, \ldots, f_{n-1}$, v_i, v_f, a_i and a_f are constants, where d is the derivative order (0 for the C-Spline, 1 for the first, 2 for the second and 3 for the third derivatives w.r.t. t).

SIPAMPL

SIPAMPL is an extension for AMPL, allowing the codification (modeling) of SIP problems. SIPAMPL today provides:

- a database with more than 160 SIP problems coded
- a dynamic B and C-Splines library (robotics problems)
- interface routines between AMPL and any SIP solver (NSIPS) - SIPAMPL routines
- interface routines between MATLAB and SIPAMPL routines
- a *Select* tool

SIPAMPL is publicly available in (http://www.norg.uminho.pt/aivaz/)

A robotics example

The total travel time is to be minimized while the velocity is to be bounded by a constant:

$$\min_{h \in R^5} \sum_{i=1}^{5} h_i$$

$$s.t. \qquad -100 \leq Q'\left(\tau \sum_{i=1}^{5} h_i\right) \leq 100$$

$$h_i \geq 0, \quad \forall \tau \in [0,1]\,.$$

A possible codification of this problem in (SIP)AMPL format is the following:

```
# declare used external funtions
function cspline;

# number of coefficients
param n:=5;
# number of knots (nk = n-1)
param nk:=4;
# knots vector
param f{1..nk};

# initial guess for spaces
param hinit{1..n};

# coefficients
var h{i in 1..n}:=hinit[i];
# parameter
var tau:=0;

# initial and final velocity
               and acceleration
param vi:=0;
param vf:=0;
param ai:=0;
param af:=0;
```

```
# declare used external funtions
function cspline;

# number of coefficients
param n:=5;
# number of knots (nk = n-1)
param nk:=4;
# knots vector
param f{1..nk};

# initial guess for spaces
param hinit{1..n};

# coefficients
var h{i in 1..n}:=hinit[i];
# parameter
var tau:=0;

# initial and final velocity
               and acceleration
param vi:=0;
param vf:=0;
param ai:=0;
param af:=0;
```

```
# to save some space we define a variable
#  which is the C-Spline
var g=cspline(tau*(sum {i in 1..n} (h[i])),1,n,
      {i in 1..n}h[i], {i in 1..nk}f[i],vi,vf,ai,af);


minimize obj:
  (sum {i in 1..n} (h[i]));

subject to tcons:
  -100 <= g <= 100;

subject to bounds {i in 1..n}:
  h[i] >= 0;

subject to tbounds:
  0 <= tau <= 1;
```

```
# declare used external funtions
function cspline;

# number of coefficients
param n:=5;
# number of knots (nk = n-1)
param nk:=4;
# knots vector
param f{1..nk};

# initial guess for spaces
param hinit{1..n};

# coefficients
var h{i in 1..n}:=hinit[i];
# parameter
var tau:=0;

# initial and final velocity
               and acceleration
param vi:=0;
param vf:=0;
param ai:=0;
param af:=0;
```

```
# to save some space we define a variable
#  which is the C-Spline
var g=cspline(tau*(sum {i in 1..n} (h[i])),1,n,
      {i in 1..n}h[i], {i in 1..nk}f[i],vi,vf,ai,af);


minimize obj:
  (sum {i in 1..n} (h[i]));

subject to tcons:
  -100 <= g <= 100;

subject to bounds {i in 1..n}:
  h[i] >= 0;

subject to tbounds:
  0 <= tau <= 1;
```

```
data;

# knots
param f :=
  1   1.5
  2   2
  3   1.75
  4   1.8;
```

```
# declare used external funtions
function cspline;

# number of coefficients
param n:=5;
# number of knots (nk = n-1)
param nk:=4;
# knots vector
param f{1..nk};

# initial guess for spaces
param hinit{1..n};

# coefficients
var h{i in 1..n}:=hinit[i];
# parameter
var tau:=0;

# initial and final velocity
               and acceleration
param vi:=0;
param vf:=0;
param ai:=0;
param af:=0;
```

```
# to save some space we define a variable
#  which is the C-Spline
var g=cspline(tau*(sum {i in 1..n} (h[i])),1,n,
      {i in 1..n}h[i], {i in 1..nk}f[i],vi,vf,ai,af);


minimize obj:
  (sum {i in 1..n} (h[i]));

subject to tcons:
  -100 <= g <= 100;

subject to bounds {i in 1..n}:
  h[i] >= 0;

subject to tbounds:
  0 <= tau <= 1;
```

```
data;

# knots
param f :=
  1   1.5
  2   2
  3   1.75
  4   1.8;
```

```
# initial guess

param hinit :=
  1   0.5
  2   0.25
  3   0.75
  4   0.5
  5   0.25;
```

Problems coded (`lin2.mod`)

- Unimate PUMA 560 type robot with 6 revolute joints
- minimum time trajectory planning with simple velocity, acceleration and jerk constraints
- $v_i = v_f = a_i = a_f = 0$
- $h^0 = [3.607, 3.607, 2.878, 4.275, 5.612, 2.915, 5.879, 1.336, 1.336]$, giving a total time of 31.445 seconds ($t_0 = 0$, $t_n = 31.445$)

knot	Joint 1	Joint 2	Joint 3	Joint 4	Joint 5	Joint 6
			position in *degrees*			
1	10	15	45	5	10	6
2	60	25	180	20	30	40
3	75	30	200	60	-40	80
4	130	-45	120	110	-60	70
5	110	-55	15	20	10	-10
6	100	-70	-10	60	50	10
7	-10	-10	100	-100	-40	30
8	-50	10	50	-30	10	20

Bounds	Joint 1	Joint 2	Joint 3	Joint 4	Joint 5	Joint 6
Velocity $(degrees/sec)$	100	95	100	150	130	110
Acceleration $(degrees/sec^2)$	45	40	75	70	90	80
Jerk $(degrees/sec^3)$	60	60	55	70	75	70

Problems coded (`delucas1.mod` and `delucas2.mod`)

`deluca1.mod` is a light robot with 2 joints and `deluca2.mod` is a planar motion of an IBM 7535 robot with 2 joints.

	l_1 (m)	l_2 (m)	d_2 (m)	m_2 (kg)	m_p (kg)	J_1 $(kg\ m^2)$	J_2 $(kg\ m^2)$	J_p $(kg\ m^2)$
`deluca1`	0.5	0.5	0.25	1	0	0.084	0.084	0
`deluca2`	0.4	0.25	0.125	15	6	1.6	0.34	0.01

where l_i and J_i $(i = 1, 2)$ are the length and moment of inertia, w.r.t. the axis of the driving joint for link i, m_2 is the mass of link 2, while m_p and J_p are the mass and centroidal inertia of the payload. d_2 is the distance between the axis of the second link joint and the center of mass of the second link.

These problems contain velocity and torque constraints. The velocity limit was $2\ rad/sec$ for both joints and $7\ Nm$ and $2\ Nm$ were the torque limits in joint 1 and joint 2, respectively.

Problem	Joint	$v_i\ (rad/sec)$	$v_f\ (rad/sec)$	$a_i\ (rad/sec^2)$	$a_f\ (rad/sec^2)$
`deluca1`	1	0	0	13.880794	-0.415203
	2	0	0	-11.067942	-4.186542
`deluca2`	1	0	0	2.5207742	-2.1966904
	2	0	0	2.5207742	-2.1966904

The initial time intervals, in seconds, considered were
$h^0 = [1, 1, 0.5, 0.5, 0.5, 0.5, 0.5]$ and
$h^0 = [0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3]$ for `deluca1` and `deluca2`, respectively.

Problem	Joint	Position in $radians$ (knots)					
		1	2	3	4	5	6
`deluca1`	1	0	0.5	0.75	1	1.25	1.5
	2	0	-0.5	-1	-1.5	-1	0.5
`deluca2`	1 and 2	0.1	0.2	0.25	0.3	0.35	0.4

Problems coded (`lobianco1.mod`)

The robot arm (with two joints) is considered in initial and final rest position ($v_i = v_f = a_i = a_f = 0$). The problem considers torque, linear and angular velocity limits of 260 Nm, 50 Nm, 0.7 m/sec and 1.5 rad/sec, respectively.

l_1 (m)	l_2 (m)	m_1 (kg)	m_2 (kg)
1.0	0.5	15.0	7.0

where l_i and m_i, $i = 1, 2$, are the link lengths and the link masses.

$h^0 = [0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]$ was used as a initial guess which gives a total time travel of 5.5 sec.

Knot	Joint 1 (rad)	Joint 2 (rad)
1	0.0000	-1.5708
2	0.1253	-1.6804
3	0.2517	-1.7594
4	0.3789	-1.8074
5	0.5054	-1.8235
6	0.5837	-1.7087
7	0.6119	-1.4581
8	0.4263	-1.1040
9	0.3903	-1.1124
10	0.3526	-1.1152

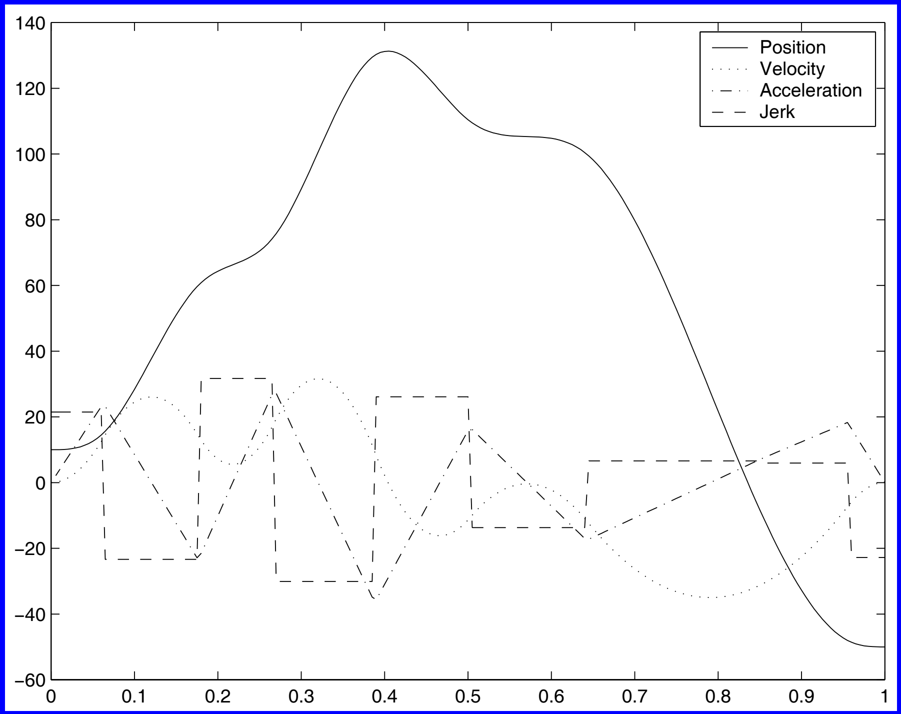
Numerical results

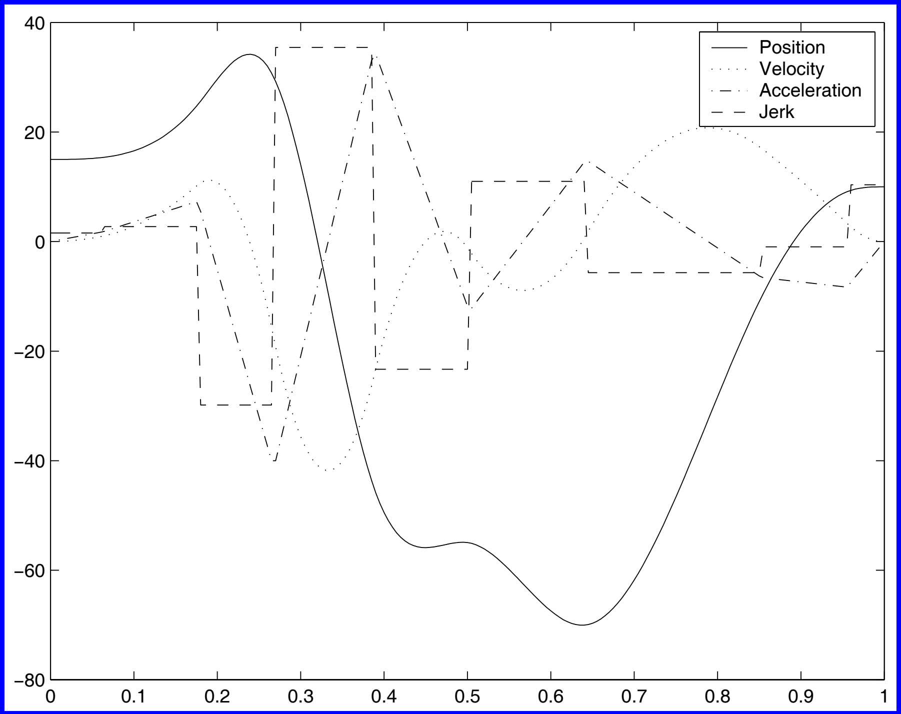
	lin2		lobianco1	
	NSIPS	Prev.	NSIPS	Prev.
h_1	1.125150	1.131000	0.010000	0.020000
h_2	2.039520	2.004000	0.348599	0.364290
h_3	1.635940	2.068000	0.156699	0.184190
h_4	2.158020	2.016000	0.150559	0.183860
h_5	2.046600	2.714000	0.154683	0.184230
h_6	2.510830	1.973000	0.138140	0.167350
h_7	3.781200	3.807000	0.191483	0.223100
h_8	1.831450	1.971000	0.391619	0.365390
h_9	0.803105	0.767000	0.106903	0.099450
h_{10}			0.022616	0.238180
h_{11}			0.385888	0.020050
Total	17.931800	18.45100	2.057190	2.050090

Numerical results (cont.)

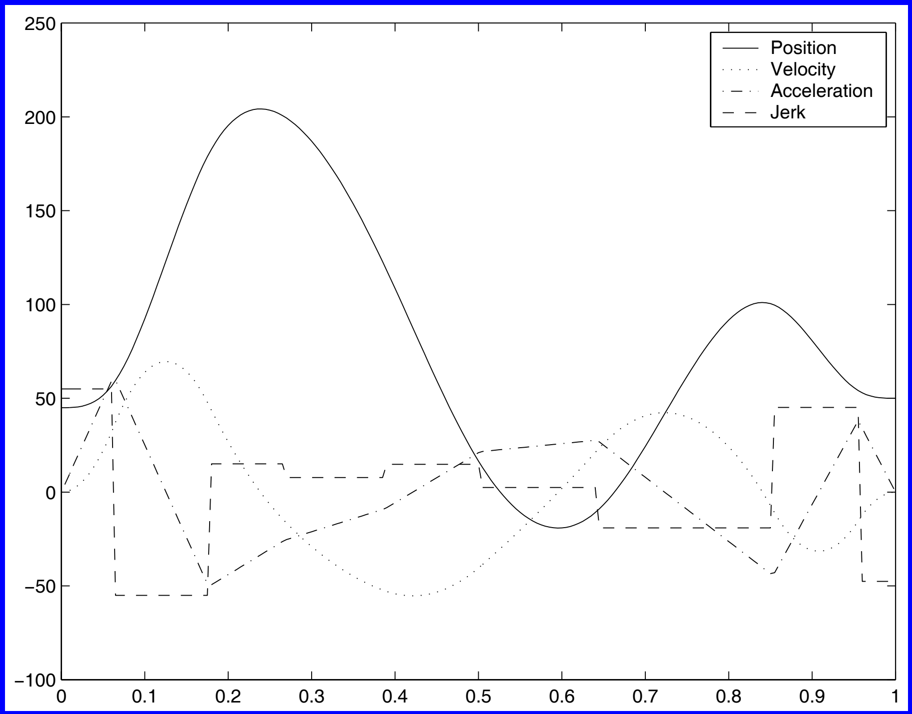
	deluca1		deluca2	
	NSIPS	Prev.	NSIPS	Prev.
h_1	0.010000	0.370000	0.010000	0.290000
h_2	0.348255	Extra knot	0.134696	Extra knot
h_3	0.260631	0.250000	0.053838	0.070000
h_4	0.361528	0.340000	0.050615	0.070000
h_5	0.351404	0.430000	0.051988	0.080000
h_6	0.010000	Extra knot	0.066819	Extra knot
h_7	1.061350	1.070000	0.010000	0.200000
Total	2.403170	2.460000	0.377956	0.710000

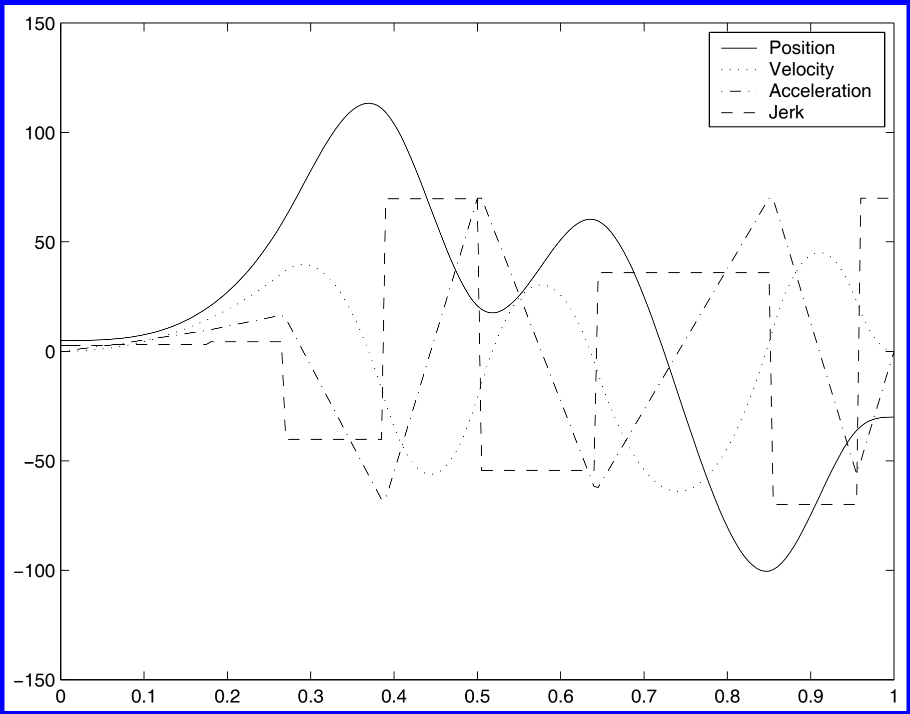
Plots - Joint 1 and 2 of `lin2`



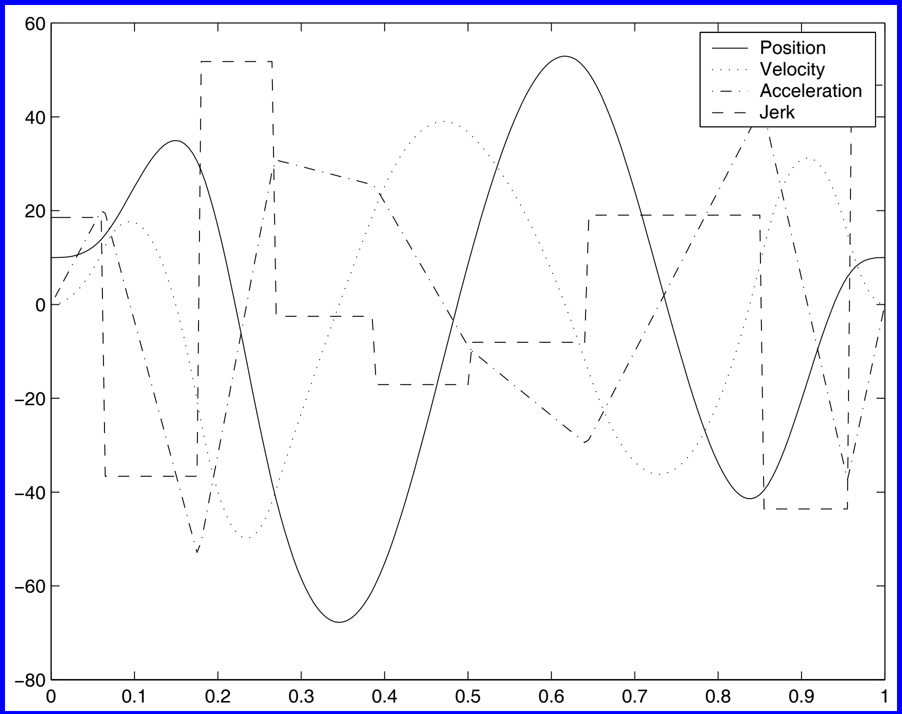


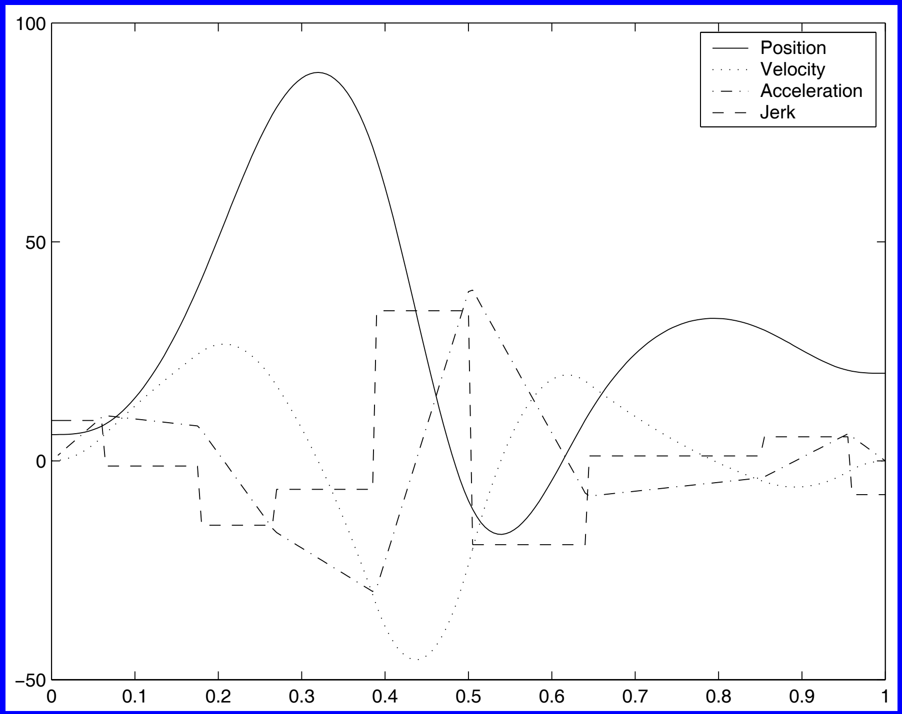
Plots - Joint 3 and 4 of `lin2`



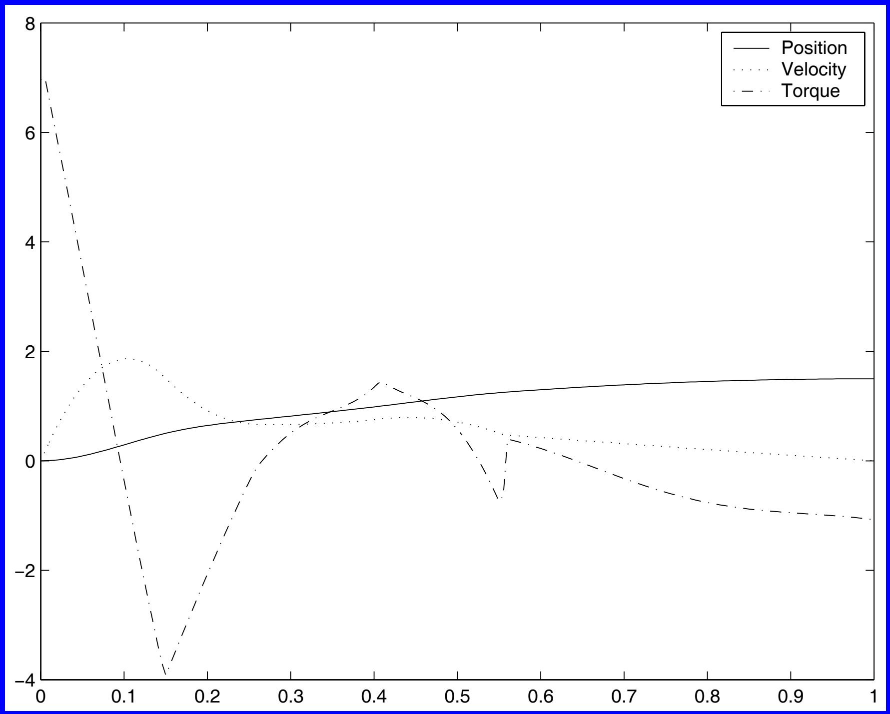


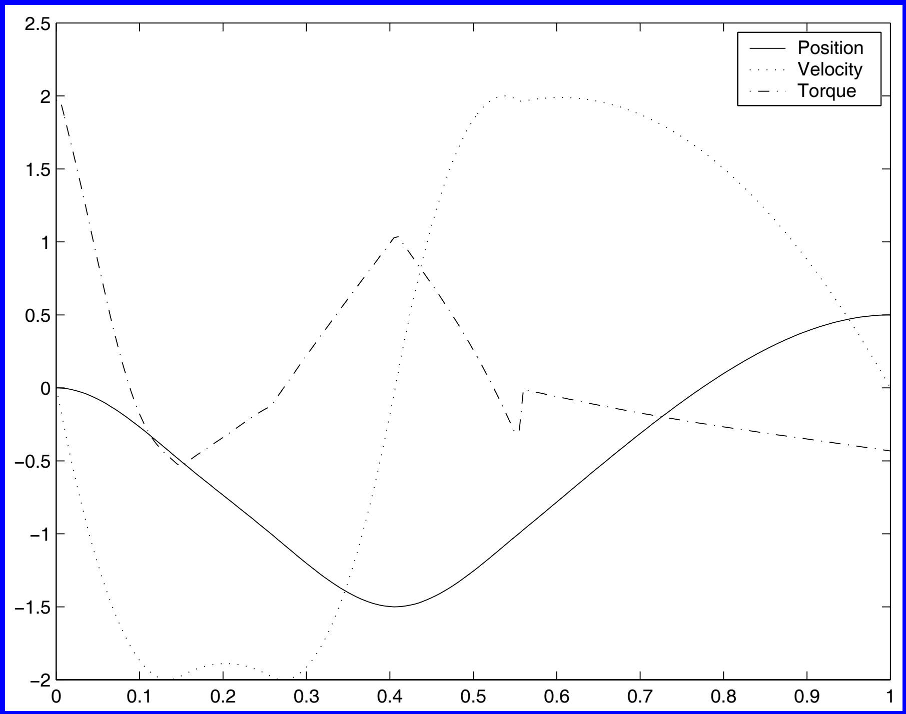
Plots - Joint 5 and 6 of `lin2`



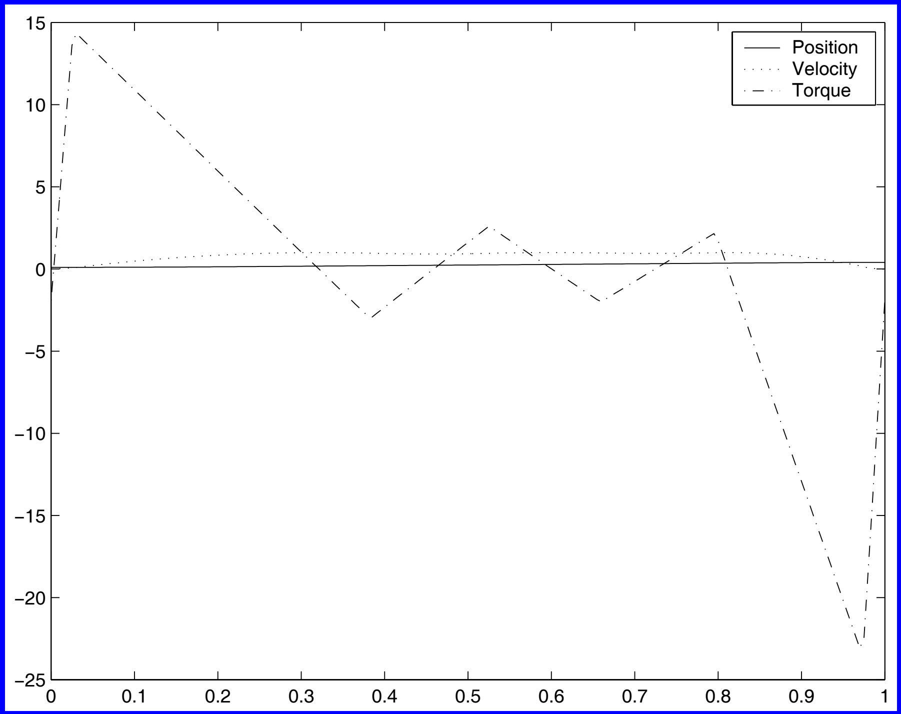


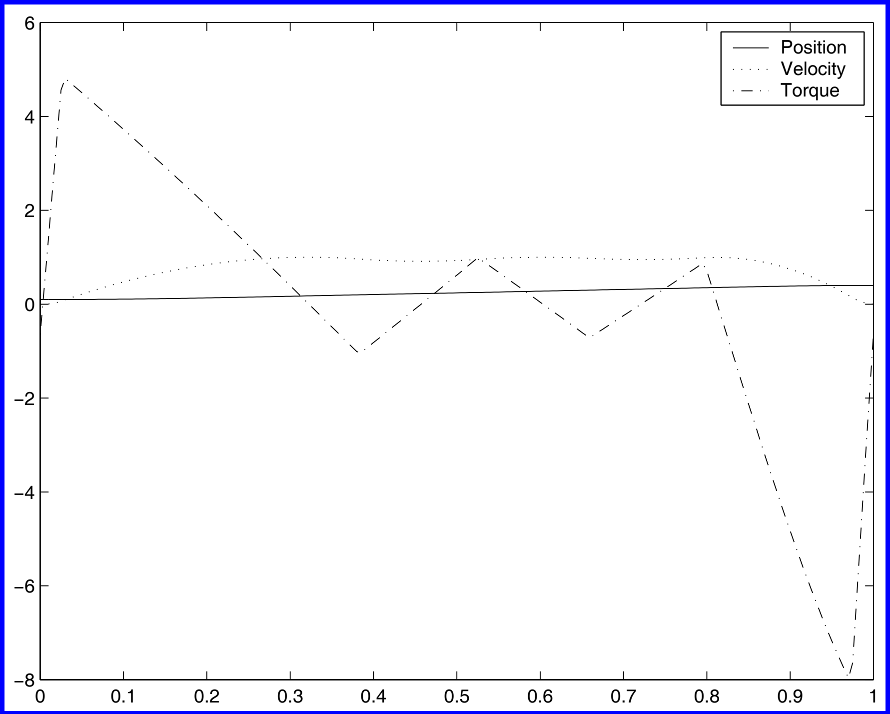
Plots - Joint 1 and 2 of `deluca1`



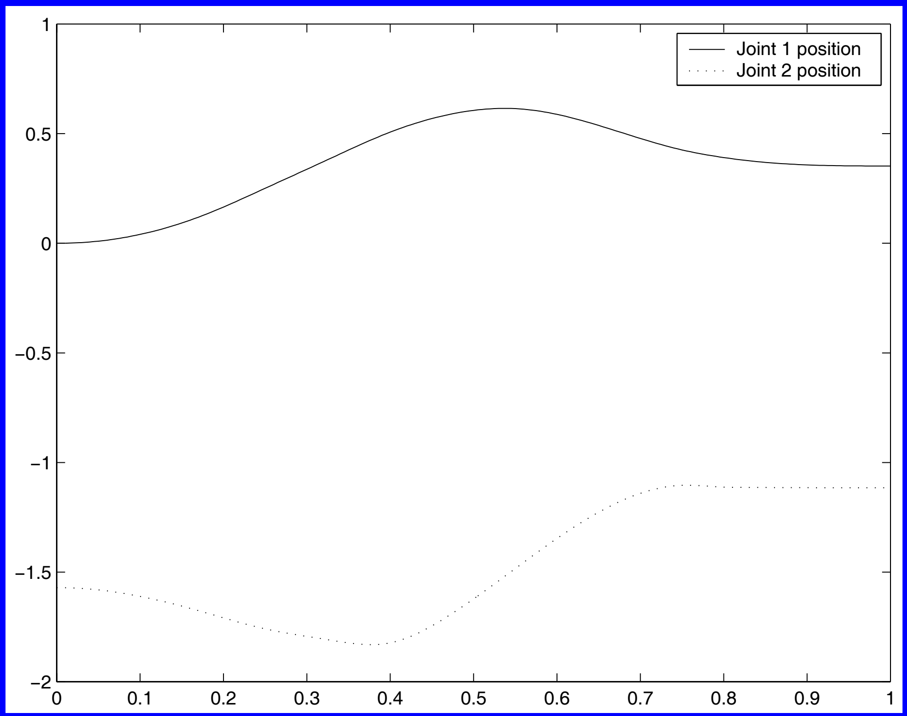


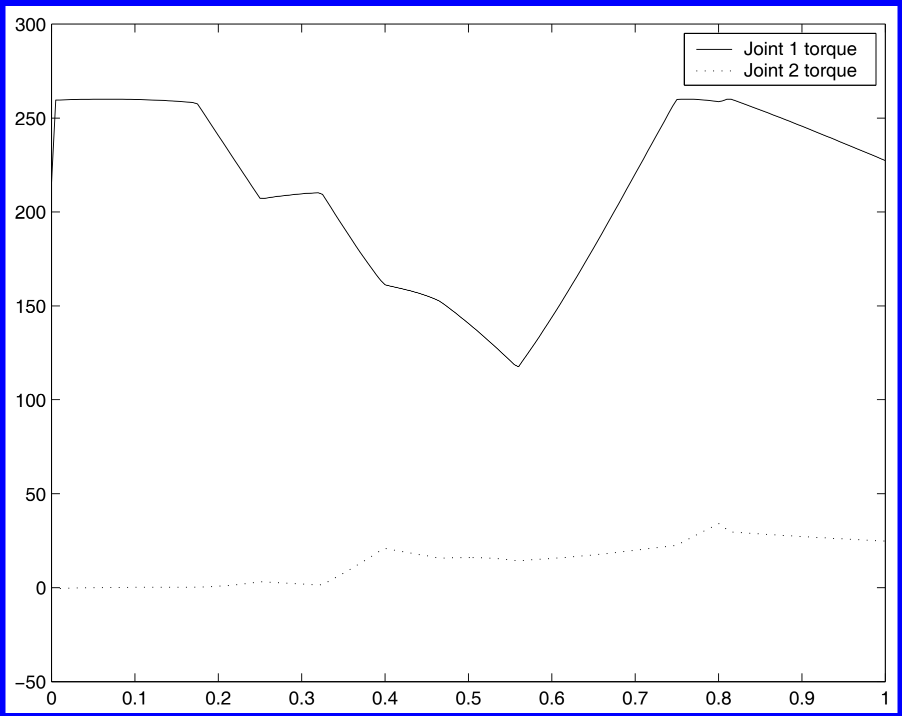
Plots - Joint 1 and 2 of `deluca2`



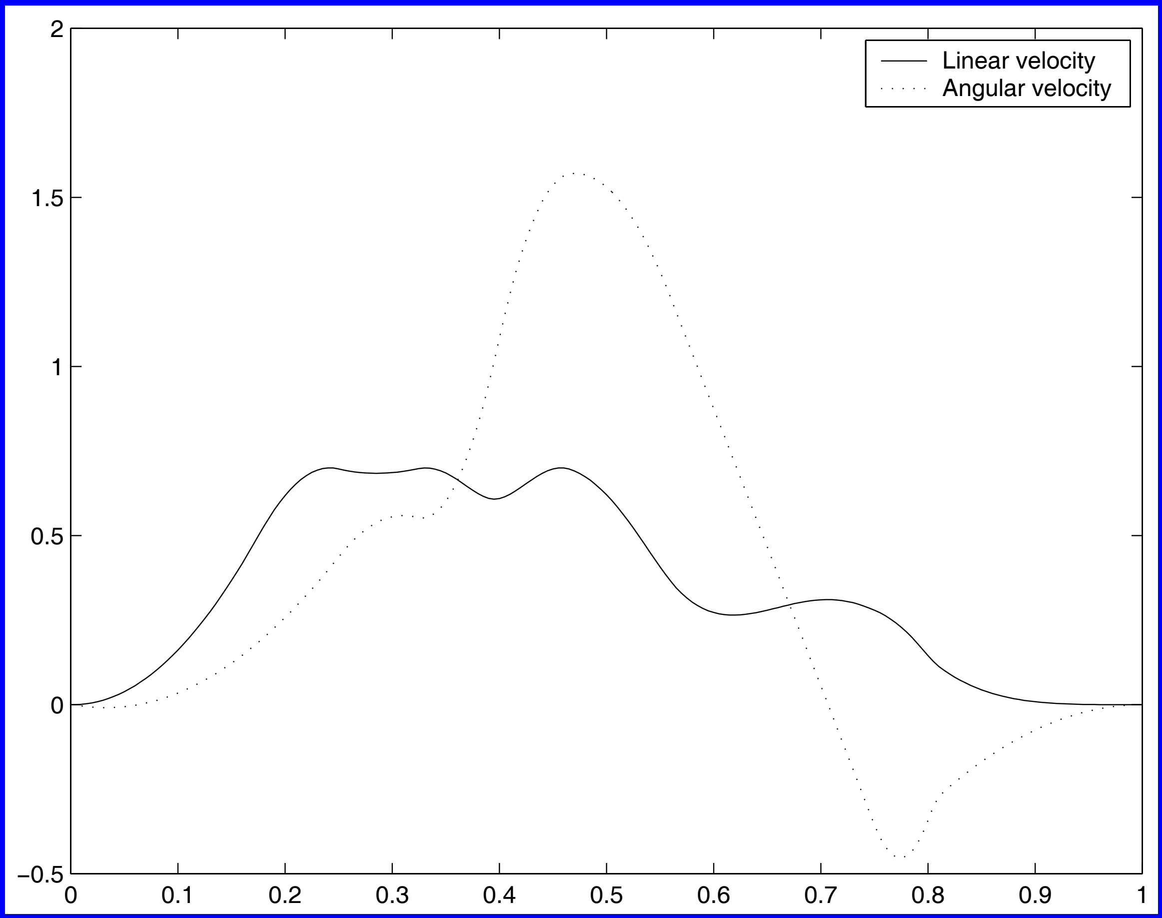


Plots - Position and Torque of `lobianco1`





Plots - Linear and angular velocity of `lobianco1`



Conclusions

- Robot trajectory planning can be posed as SIP problems;

Conclusions

- Robot trajectory planning can be posed as SIP problems;
- C-Splines dynamic library for AMPL is provided;

  `http://www.norg.uminho.pt/aivaz/`

Conclusions

- Robot trajectory planning can be posed as SIP problems;
- C-Splines dynamic library for AMPL is provided;

  `http://www.norg.uminho.pt/aivaz/`
- Four (more) test problems coded in SIPAMPL;

Conclusions

- Robot trajectory planning can be posed as SIP problems;
- C-Splines dynamic library for AMPL is provided;

  `http://www.norg.uminho.pt/aivaz/`
- Four (more) test problems coded in SIPAMPL;
- Numerical results with the NSIPS solver (discretization method);

The End

email: aivaz@dps.uminho.pt
emgpf@dps.uminho.pt
Web http://www.norg.uminho.pt/aivaz/
http://www.norg.uminho.pt/emgpf/

First Page

Coefficients matrix

$$\begin{pmatrix}
3h_1 + 2h_2 + \frac{h_1^2}{h_2} & h_2 & & & & & \mathbf{0} \\
h_2 - \frac{h_1^2}{h_2} & 2(h_2 + h_3) & h_3 & & & & \\
& & \vdots & & & & \\
& h_i & 2(h_i + h_{i+1}) & h_{i+1} & & i = 3, \ldots, n-3 & \\
& & \vdots & & & & \\
& & & & h_{n-2} & 2(h_{n-2} + h_{n-1}) & h_{n-1} - \frac{h_n^2}{h_{n-1}} \\
\mathbf{0} & & & & & h_{n-1} & 3h_n + 2h_{n-1} + \frac{h_n^2}{h_{n-1}}
\end{pmatrix}$$

Independent term

$$\begin{pmatrix} 6\left(\frac{f_2}{h_2}+\frac{f_0}{h_1}\right)-6\left(\frac{1}{h_1}+\frac{1}{h_2}\right)\left(f_0+h_1v_i+\frac{h_1^2M_0}{3}\right)-h_1M_0 \\ \frac{6}{h_2}\left(f_0+h_1v_i+\frac{h_1^2M_0}{3}\right)+\frac{6f_3}{h_3}-6\left(\frac{1}{h_2}+\frac{1}{h_3}\right)f_2 \\ \cdots \\ 6\left(\frac{f_{i+1}-f_i}{h_{i+1}}-\frac{f_i-f_{i-1}}{h_i}\right), \quad i=3,\ldots,n-3 \\ \cdots \\ \frac{6}{h_{n-1}}\left(f_n-f_{n-2}-v_fh_n+\frac{h_n^2M_n}{3}\right)-6\left(\frac{f_{n-2}-f_{n-3}}{h_{n-2}}\right) \\ -6\left(\frac{f_n-f_{n-2}-v_fh_n}{h_{n-1}}\right)+6\left(v_f-\frac{h_nM_n}{3}-\frac{h_n^2M_n}{3h_{n-1}}\right)-h_nM_n \end{pmatrix}$$