Control Theory and Splines, Applied to Signature Storage

by

Per Enqvist

September 12, 1994

Abstract

In this report the problem we are going to study, is the interpolation of a set of points in the plane with the use of control theory. We will discover how different systems generate different kinds of splines, cubic and exponential, and investigate the effect that the different systems have on the tracking problem. Actually we will see that the important parameters will be the two eigenvalues of the control matrix.

Contents

List of Figures

List of Tables

1 Introduction

I would like to begin by thanking my advisor Pr. Anders Lindquist for initiating contact with Texas Tech. I would also like to thank Pr. Clyde F. Martin for being my advisor at Texas Tech.

In this report we will look at a way to store signatures. We want to do this by storing only a minimal amount of points on the signature curve, and still be able to reconstruct the curve by interpolating these points. The interpolation will be performed by splines, and we will look at the common splines-problem from the control theory point of view. We can construct a trajectory of a system that passes through a specified set of points, and thus interpolate the points.

Two questions that need to be answered arise. First, when is it possible for the system to pass through the points? Second, when there are many ways to accomplish that, what sort of conditions should we demand that the system fulfill in order that we get a unique solution.

The question of when it is possible to interpolate the points will be answered in the general case in section 2 *Reachability* and for our particular system in section 3 *The System.*

An algorithm to find the solution is developed in section 4 *Derivations.*

The choice of boundary conditions is discussed in section 5 *Boundary conditions.*

In section 6 *Results* the results of tests done upon parametric curves are displayed and discussed.

A summary in Swedish is provided in section 7 *Resumé - Summary in Swedish.*

The programs I have been using are included in section 8 *Programs.* Included among the Matlab programs is an altered version of the original Matlab program quad8.

When we have answered the two questions, we have to decide what kind of system we will use for the interpolation. We can easily imagine that we would get to completely different curves if we asked a pedestrian to walk through a set of points and if we asked a cyclist to ride his bike through the same points. In the first case we would get (if we suppose that man is lazy), linear interpolation, and in the second case we would get a smooth rounded curve. This is the same as in our case where we have exponential and cubic parametric splines.

2 Reachability

In this section we will determine under what circumstances it is possible to take a time invariant linear system from a point $\mathbf{x}_0$ at time t_0 to a point $\mathbf{x}_1$ at time t_1. It is a vital property to us, because, in order to interpolate we have to be able to pass through the points. We will call a system completely reachable if it has the property that this can be done in any positive time for any two points.

This is a classical control theory question, and it is answered by the following well known theorem, which was at least implicitly discovered by Kalman.

Theorem 2.1 *Suppose that the system below is given,*

$$\begin{cases} \dot{\mathbf{x}} &= A\mathbf{x} + B\mathbf{u} \\ \mathbf{y} &= C\mathbf{x} \end{cases} \tag{2.1}$$

where A is $n \times n$ and B is $n \times k$. Then it is completely reachable iff $\Gamma \triangleq [B, AB, A^2B, \ldots, A^{n-1}B]$, is full rank.

In order to prove this and understand how the reachability concept can be characterized by the matrix Γ, we will have to look at the general solution of equation 2.1.

$$\mathbf{x}(t) = e^{A(t-t_0)}\mathbf{x}_0 + \int_{t_0}^{t} e^{A(t-s)}B\mathbf{u}_k(s)ds \tag{2.2}$$

In order to have the desired state $\mathbf{x}_1$ at time t_1 the following equality must be satisfied.

$$\mathbf{x}_1 = e^{A(t_1-t_0)}\mathbf{x}_0 + \int_{t_0}^{t_1} e^{A(t_1-s)}B\mathbf{u}_k(s)ds \tag{2.3}$$

The question of reachability is now easily seen to be the question of whether there are any solutions to the mapping $L : \mathcal{U} \mapsto \mathbb{R}^n$ such that

$$L\mathbf{u} \triangleq \int_{t_0}^{t_1} e^{A(t_1-s)}B\mathbf{u}(s)ds = \mathbf{x}_1 - e^{A(t_1-t_0)}\mathbf{x}_0 \triangleq \mathbf{d} \tag{2.4}$$

Since we recognize L as a linear operator, it is as always very fruitful to use a theorem from the general theory of functional analysis [4, p.250].

Theorem 2.2 *If X, Y are complete inner-product spaces and A:X $\mapsto$ Y is a linear continuous operator then*

$$\mathfrak{Im}A = \mathfrak{Im}AA^*$$

We know that $\mathbb{R}^n$ is a Hilbert space, but we have to look at what kind of space $\mathcal{U}$ is. We choose to introduce the following inner product for $\mathcal{U}$

$$(\mathrm{u}, \mathrm{v})_{\mathcal{U}} \triangleq \int_{t_0}^{t_1} \mathrm{u}(t)'\mathrm{v}(t)dt$$

and it can be checked that U becomes a Hilbert space. We know that there is a adjoint operator $L^* : \mathbb{R}^n \mapsto \mathcal{U}$ such that

$$(\mathrm{d}, L\mathrm{u})_{\mathbf{R}^n} = (L^*\mathrm{d}, \mathrm{u})_{\mathcal{U}}$$

and we get the adjoint L^* of the mapping L through this equation

$$\begin{aligned}(\mathrm{d}, L\mathrm{u})_{\mathbf{R}^n} &= \mathrm{d}' \int_{t_0}^{t_1} e^{A(t_1-s)}B\mathrm{u}(s)ds \\ &= \int_{t_0}^{t_1} \left(B'e^{A'(t_1-s)}\mathrm{d}\right)' \mathrm{u}(s)ds = (L^*\mathrm{d}, \mathrm{u})_{\mathcal{U}}\end{aligned}$$

We thus have a linear mapping $W \triangleq LL^* : \mathbb{R}^n \mapsto \mathbb{R}^n$, that is, it is actually only a matrix operator.

$$W \triangleq LL^* = \int_{t_0}^{t_1} e^{A(t_1-s)}BB'e^{A'(t_1-s)}ds$$

With only the basic knowledge about matrices we will now be able to prove the following lemma

Lemma 2.1 *Let A be $n \times n$ and B be $n \times k$. Then, for all t_0, t_1 such that $t_0 < t_1$ we have*

$$\mathfrak{Im}W(t_0, t_1) = \mathfrak{Im}\left[B, AB, A^2B, \ldots, A^{n-1}B\right]$$

Proof: We will do this by showing that $\mathfrak{Im}\Gamma \subseteq \mathfrak{Im}W$ and $\mathfrak{Im}W \subseteq \mathfrak{Im}\Gamma$.

[$\mathfrak{Im}\Gamma \subseteq \mathfrak{Im}W$]

Let $\mathbf{a} \in \mathfrak{Ker}W$, which implies that $0 = \mathbf{a}'W\mathbf{a} = \int_{t_0}^{t_1} \mathbf{a}'e^{A(t_1-s)}BB'e^{A'(t_1-s)}\mathbf{a}ds$, i.e.

$$\int_{t_0}^{t_1} \left| B'e^{A'(t_1-s)}\mathbf{a} \right| ds = 0$$

from which it follows that

$$B'e^{A'(t_1-s)}\mathbf{a} \equiv 0, \qquad \forall s \in [t_0, t_1],$$

i.e,

$$\sum_{j=0}^{\infty} \frac{1}{j!}(t_1 - s)^j B'(A')^j \mathbf{a} \equiv 0.$$

This implies that $[B, AB, A^2B, \ldots, A^{n-1}B]'\,\mathbf{a} = 0$. That is, for an arbitrary $\mathbf{a} \in \mathfrak{Ker}W$ we have $\mathbf{a} \in \mathfrak{Ker}\Gamma'$ which implies that $\mathfrak{Ker}W \subseteq \mathfrak{Ker}\Gamma'$ and by a theorem from fundamental algebra this equals $\mathfrak{Im}\Gamma \subseteq \mathfrak{Im}W$.

[$\mathfrak{Im}W \subseteq \mathfrak{Im}\Gamma$]
Suppose $\mathbf{a} \in \mathfrak{Im}W$. Then there exists a $\mathbf{x} \in \mathbb{R}^n$ such that $\mathbf{a} = W\mathbf{x}$, and hence

$$\mathbf{a} = \sum_{j=0}^{\infty} A^j B \int_{t_0}^{t_1} \frac{1}{j!}(t_1 - s)^j B'e^{A'(t_1-s)}\mathbf{x}ds$$

from which it is obvious that $\mathbf{a} \in \mathfrak{Im}[B, AB, A^2B, \ldots]$ and by Cayley-Hamiltons theorem that $\mathbf{a} \in \mathfrak{Im}\Gamma$, which concludes the proof. □

The main theorem of this section will follow as a direct consequence of the lemma.

Proof:[Theorem 2.1] By lemma 2.1, $\mathbf{x}_1 - e^{A(t_1-t_0)}\mathbf{x}_0 \triangleq \mathbf{d} \in \mathbb{R}^n$ and $\mathfrak{Im}\Gamma = \mathbb{R}^n$ implies complete reachability.□

3 The System

We will consider the system with the state and dynamics given by

$$\mathbf{x} = \begin{bmatrix} x \\ \dot{x} \\ y \\ \dot{y} \end{bmatrix}, \qquad \begin{cases} \dot{\mathbf{x}} &= A\mathbf{x} + B\mathbf{u} \\ \mathbf{y} &= C\mathbf{x} \end{cases} \tag{3.5}$$

where

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & \lambda_1 & \alpha_1 & \alpha_2 \\ 0 & 0 & 0 & 1 \\ \beta_1 & \beta_2 & 0 & \lambda_2 \end{pmatrix}, B = \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix}, C = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{3.6}$$

This gives us the property

$$\mathbf{y} = C\mathbf{x} = \begin{pmatrix} x \\ y \end{pmatrix}$$

and the system dynamics will look like

$$\begin{cases} \ddot{x} &= \lambda_1 \dot{x} + \alpha_1 y + \alpha_2 \dot{y} + u_1 \\ \ddot{y} &= \lambda_2 \dot{y} + \beta_1 x + \beta_2 \dot{x} + u_2 \end{cases} \tag{3.7}$$

And it gives us the following Γ

$$\Gamma = \begin{bmatrix} 0 & 0 & 1 & 0 & \lambda_1 & \alpha_2 & \Gamma_{1,7} & \Gamma_{1,8} \\ 1 & 0 & \lambda_1 & \alpha_2 & \alpha_2\beta_2 + \lambda_1^2 & \alpha_1 + \alpha_2(\lambda_1 + \lambda_2) & \Gamma_{2,7} & \Gamma_{2,8} \\ 0 & 0 & 0 & 1 & \beta_2 & \lambda_2 & \Gamma_{3,7} & \Gamma_{3,8} \\ 0 & 1 & \beta_2 & \lambda_2 & \beta_1 + \beta_2(\lambda_1 + \lambda_2) & \alpha_2\beta_2 + \lambda_2^2 & \Gamma_{4,7} & \Gamma_{4,8} \end{bmatrix}$$

where

$$\begin{cases} \Gamma_{1,7} &= \alpha_2\beta_2 + \lambda_1^2 \\ \Gamma_{1,8} &= \alpha_1 + \alpha_2(\lambda_1 + \lambda_2) \\ \Gamma_{2,7} &= \alpha_1\beta_2 + \alpha_2\beta_1 + \alpha_2\beta_2(2\lambda_1 + \lambda_2) + \lambda_1^3 \\ \Gamma_{2,8} &= \alpha_2^2\beta_2 + \alpha_1(\lambda_1 + \lambda_2) + \alpha_2\lambda_1^2 + \alpha_2\lambda_1\lambda_2 + \alpha_2\lambda_2^2 \\ \Gamma_{3,7} &= \beta_1 + \beta_2(\lambda_1 + \lambda_2) \\ \Gamma_{3,8} &= \alpha_2\beta_2 + \lambda_2^2 \\ \Gamma_{4,7} &= \alpha_2\beta_2^2 + \beta_1(\lambda_1 + \lambda_2) + \beta_2\lambda_1^2 + \beta_2\lambda_1\lambda_2 + \beta_2\lambda_2^2 \\ \Gamma_{4,8} &= \alpha_1\beta_2 + \alpha_2\beta_1 + \alpha_2\beta_2(\lambda_1 + 2\lambda_2) + \lambda_2^3 \end{cases}$$

As Γ is easily seen to have full row rank, by simply looking at the first four columns. The class of systems we are going to consider in this article will all have the desired property of complete reachability, by Theorem 2.1.

4 Derivations

Given a set of points in the plane $\{(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)\}$ and the corresponding time points $\{t_0, t_1, \ldots, t_n\}$ we would like to find the control functions $\{\mathrm{u}_0, \mathrm{u}_1, \ldots, \mathrm{u}_{n-1}\}$ that takes the system through the points at the specified times.

Let's study the control $\mathrm{u}_k : \begin{pmatrix} \mathrm{x}_k \\ t_k \end{pmatrix} \mapsto \begin{pmatrix} \mathrm{x}_{k+1} \\ t_{k+1} \end{pmatrix}$

As $t \in [t_k, t_{k+1}]$ the state of the system will be

$$\mathrm{x}(t) = e^{A(t-t_k)}\mathrm{x}_k + \int_{t_k}^{t} e^{A(t-s)}B\mathrm{u}_k(s)ds \tag{4.8}$$

and as we want the state of the system to be x_{k+1} at time t_{k+1} we get the following condition.

$$\mathrm{x}_{k+1} = e^{A(t_{k+1}-t_k)}\mathrm{x}_k + \int_{t_k}^{t_{k+1}} e^{A(t_{k+1}-s)}B\mathrm{u}_k(s)ds \tag{4.9}$$

The solution $\mathbf{u}_k$ to equation 4.9 that minimizes the norm of the control signal is then given by

$$\mathbf{u}_k(t) = B'e^{-A't}\left(\int_{t_k}^{t_{k+1}} e^{-As}BB'e^{-A's}ds\right)^{-1}\left(e^{-At_{k+1}}\mathbf{x}_{k+1} - e^{-At_k}\mathbf{x}_k\right) \quad (4.10)$$

The control would be specified completely by equation 4.10 if we knew the whole state-vector at each interpolation point. We know all (x_k, y_k) but we do not know the $(\dot{x}_k, \dot{y}_k)$. To determine the $2(n+1)$ unknowns we have to apply some conditions on the solution, and our first choice will be to require that the control is continuous,

Assumption 4.1

$$\mathbf{u}_k(t_{k+1}) = \mathbf{u}_{k+1}(t_{k+1}), \qquad k = 0 \ldots n-2$$

This will give us $2(n-1)$ conditions and will leave only four unknowns. We will apply the additional conditions on the boundary and we will have to come back to this in section 5 *Boundary Conditions*.

In many applications it is just the shape of the trajectory that matters, and not the velocity that the system tracks the trajectory with. In these cases it makes it much easier to assume that the time between each point is a constant.

Assumption 4.2 *Let* $t_{k+1} - t_k = h$.

Assumption 4.2 can be used to simplify the integral in equation 4.10.

$$\int_{t_k}^{t_{k+1}} e^{-As}BB'e^{-A's}ds = \{\tau = s - t_k\} =$$

$$e^{-At_k}\underbrace{\int_0^h e^{-A\tau}BB'e^{-A'\tau}d\tau}_{matrix constant}e^{-A't_k}$$

Definition 4.1

$$M \triangleq \int_0^h e^{-A\tau}BB'e^{-A'\tau}d\tau$$

We can now rewrite expression 4.10 as

$$\mathbf{u}_k(t) = B' e^{-A'(t-t_k)} M^{-1} (e^{-Ah}\mathbf{x}_{k+1} - \mathbf{x}_k) \tag{4.11}$$

Using this expression we will now investigate how the continuity condition in assumption 4.1 looks.

$$\mathbf{u}_k(t_{k+1}) = B' e^{-A'h} M^{-1} (e^{-Ah}\mathbf{x}_{k+1} - \mathbf{x}_k) \;=$$

$$B' M^{-1} (e^{-Ah}\mathbf{x}_{k+2} - \mathbf{x}_{k+1}) \;=\; \mathbf{u}_{k+1}(t_{k+1})$$

We can rewrite this condition using

Definition 4.2

$$Z \triangleq M^{-1} e^{-Ah}$$

$$W \triangleq e^{-A'h} M^{-1} e^{-Ah} + M^{-1}$$

giving the equation the simple form

$$B' (Z'\mathbf{x}_k - W\mathbf{x}_{k+1} + Z\mathbf{x}_{k+2}) = 0, \qquad k = 0 \ldots n-2 \tag{4.12}$$

In block diagonal form

$$B' \begin{bmatrix} Z' & -W & Z & \ldots & 0 & 0 & 0 \\ 0 & Z' & -W & \ldots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & -W & Z & 0 \\ 0 & 0 & 0 & \ldots & Z' & -W & Z \end{bmatrix} \begin{bmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_{n-2} \\ \mathbf{x}_{n-1} \\ \mathbf{x}_n \end{bmatrix} = 0 \tag{4.13}$$

Now, we have to look at what the unknowns are. The vector in equation 4.13 is made up of subvectors

$$\mathbf{x}_k = \begin{bmatrix} x_k \\ \dot{x}_k \\ y_k \\ \dot{y}_k \end{bmatrix}$$

consisting of two knowns and two unknowns. By partitioning the submatrixes W, Z as

$$W = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ w_{\cdot 1} & w_{\cdot 2} & w_{\cdot 3} & w_{\cdot 4} \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}, Z = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ z_{\cdot 1} & z_{\cdot 2} & z_{\cdot 3} & z_{\cdot 4} \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} = \begin{bmatrix} \cdots & z_{1\cdot} & \cdots \\ \cdots & z_{2\cdot} & \cdots \\ \cdots & z_{3\cdot} & \cdots \\ \cdots & z_{4\cdot} & \cdots \end{bmatrix}$$

and using the notations given in the following the definition, we can keep the unknowns on the left side and move the position coordinates over to the right hand side.

Definition 4.3

$$\begin{aligned}
\mathbf{x}^{pos} &= \begin{bmatrix} x \\ y \end{bmatrix} \\
\mathbf{x}^{vel} &= \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} \\
W_l^B &= B' \begin{bmatrix} w_{\cdot 2} & w_{\cdot 4} \end{bmatrix} = \begin{bmatrix} w_{22} & w_{24} \\ w_{42} & w_{44} \end{bmatrix} \\
W_r^B &= B' \begin{bmatrix} w_{\cdot 1} & w_{\cdot 3} \end{bmatrix} = \begin{bmatrix} w_{21} & w_{23} \\ w_{41} & w_{43} \end{bmatrix} \\
Z_{lu}^B &= B' \begin{bmatrix} z_{\cdot 2} & z_{\cdot 4} \end{bmatrix} = \begin{bmatrix} z_{22} & z_{24} \\ z_{42} & z_{44} \end{bmatrix} \\
Z_{ru}^B &= B' \begin{bmatrix} z_{\cdot 1} & z_{\cdot 3} \end{bmatrix} = \begin{bmatrix} z_{21} & z_{23} \\ z_{41} & z_{43} \end{bmatrix} \\
Z_{ll}^B &= B' \begin{bmatrix} z_{2\cdot} \\ z_{4\cdot} \end{bmatrix}' = \begin{bmatrix} z_{22} & z_{42} \\ z_{24} & z_{44} \end{bmatrix} \\
Z_{rl}^B &= B' \begin{bmatrix} z_{1\cdot} \\ z_{3\cdot} \end{bmatrix}' = \begin{bmatrix} z_{12} & z_{32} \\ z_{14} & z_{34} \end{bmatrix}
\end{aligned}$$

And we get the system

$$\begin{bmatrix} Z_{ll}^B & -W_l^B & Z_{lu}^B & \dots & 0 & 0 & 0 \\ 0 & Z_{ll}^B & -W_l^B & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & -W_l^B & Z_{lu}^B & 0 \\ 0 & 0 & 0 & \dots & Z_{ll}^B & -W_l^B & Z_{lu}^B \end{bmatrix} \begin{bmatrix} x_0^{vel} \\ x_1^{vel} \\ x_2^{vel} \\ \vdots \\ x_{n-2}^{vel} \\ x_{n-1}^{vel} \\ x_n^{vel} \end{bmatrix} =$$

$$- \begin{bmatrix} Z_{rl}^B & -W_r^B & Z_{ru}^B & \dots & 0 & 0 & 0 \\ 0 & Z_{rl}^B & -W_r^B & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & -W_r^B & Z_{ru}^B & 0 \\ 0 & 0 & 0 & \dots & Z_{rl}^B & -W_r^B & Z_{ru}^B \end{bmatrix} \begin{bmatrix} x_0^{pos} \\ x_1^{pos} \\ x_2^{pos} \\ \vdots \\ x_{n-2}^{pos} \\ x_{n-1}^{pos} \\ x_n^{pos} \end{bmatrix}$$

As we evaluate the right hand side, we get a constant vector. Depending on what kind of boundary conditions we choose to use, the derivations differ from here. We will deal with the most common cases, each case in turn, beginning in the next chapter.

5 Boundary Conditions

In order to get a unique solution to our problem, we had to apply the continuity condition and the boundary conditions. The continuity condition is a rather natural condition, but the boundary conditions have to be studied more extensively. The four most common choices of boundary conditions in the one dimensional case according to [2] are

1. Zero velocity at the first and at the last point.
2. Specified starting and ending direction.
3. Natural boundary conditions, $y'' = 0$.
4. Periodical conditions.

We will look at how the two dimensional equivalent of choice number 1 and 2 affect the curves, and for which the same derivation is valid.

Item 3:s two dimensional equivalent, $\tilde{x} = \tilde{y} = 0$, requires a derivation and will be studied in subsection 5.2.

We will avoid dealing with condition 4 as it only complicates the calculations, and would only be natural and interesting if we were to write the same word twice, connected with itself as RogerRoger.

Because the effect of the boundary conditions are similar at both boundaries we will restrict ourselves to only talking about the starting point.

5.1 Known velocities at boundary

We have assumed that we also know $\mathbf{x}_0^{vel}$ and $\mathbf{x}_n^{vel}$. Moving these over to the right hand side, we get a block diagonal matrix system to solve.

$$\begin{bmatrix} -W_l^B & Z_{lu}^B & \dots & 0 & 0 \\ Z_{ll}^B & -W_l^B & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & -W_l^B & Z_{lu}^B \\ 0 & 0 & \dots & Z_{ll}^B & -W_l^B \end{bmatrix} \begin{bmatrix} \mathbf{x}_1^{vel} \\ \mathbf{x}_2^{vel} \\ \vdots \\ \mathbf{x}_{n-2}^{vel} \\ \mathbf{x}_{n-1}^{vel} \end{bmatrix} = \begin{bmatrix} \Omega_1 \\ \Omega_2 \\ \vdots \\ \Omega_{n-2} \\ \Omega_{n-1} \end{bmatrix} \tag{5.14}$$

Where

$$\begin{aligned} \Omega_1 &= -Z_{rl}^B \mathbf{x}_0^{pos} + W_r^B \mathbf{x}_1^{pos} - Z_{ru}^B \mathbf{x}_2^{pos} - Z_{ll}^B \mathbf{x}_0^{vel} \\ \Omega_k &= -Z_{rl}^B \mathbf{x}_{k-1}^{pos} + W_r^B \mathbf{x}_k^{pos} - Z_{ru}^B \mathbf{x}_{k+1}^{pos} \\ \Omega_{n-1} &= -Z_{rl}^B \mathbf{x}_{n-2}^{pos} + W_r^B \mathbf{x}_{n-1}^{pos} - Z_{ru}^B \mathbf{x}_n^{pos} - Z_{lu}^B \mathbf{x}_n^{vel} \end{aligned}$$

This can easily be solved, and with a linear increase of time. Having solved the system above we now know all the states of the system in the interpolation points. We will now use equation 4.11 for the control, and insert it into equation 4.8 to get the trajectory.

$$\begin{aligned} \mathbf{x}(t) &= e^{A(t-t_k)} \left(\mathbf{x}_k + \int_0^{t-t_k} e^{-A\tau} B B' e^{-A'\tau} d\tau \, M^{-1} (e^{-Ah} \mathbf{x}_{k+1} - \mathbf{x}_k) \right) \\ \text{for } k &= 0 \dots n-1 \\ t &\in [t_k, t_{k+1}] \end{aligned} \tag{5.15}$$

As we can see from equation 5.15 the fundamental matrix and the integral is the same for all k and only has to be evaluated for $t - t_k$ between 0 and h, once and for all.

5.1.1 Zero initial velocity

We can choose to set

Assumption 5.1

$$\begin{aligned}(\dot{x}_0, \dot{y}_0) &= 0 \\ (\dot{x}_n, \dot{y}_n) &= 0\end{aligned}$$

With this condition, we will let the system start up in whatever direction that minimizes the energy norm of the control signal and takes the system to the second point.

As we know from one dimensional control theory, a system with a transfer function with zeros in the numerator will start off in the opposite direction to where it is going. Such undesired properties should certainly be avoided in our tracking problem. In the case where $\alpha_1 = \alpha_2 = \beta_1 = \beta_2 = 0$, the states x and y are independent, yielding two one dimensional transfer functions. It is easily seen to have no zeros, which is good.

Otherwise, we will have to look at the two dimensional transfer function given below:

$$\begin{aligned}T(s) &= \frac{1}{s^4 - (\lambda_1 + \lambda_2)s^3 + (\lambda_1\lambda_2 - \alpha_2\beta_2)s^2 - (\alpha_1\beta_2 + \alpha_2\beta_1)s - \alpha_1\beta_1} \times \\ &\times \begin{bmatrix} s(s - \lambda_2) & \alpha_1 + \alpha_2 s \\ \beta_1 + \beta_2 s & s(s - \lambda_1) \end{bmatrix} \qquad (5.16)\end{aligned}$$

This is a bit more tricky, and we will have to find the Q and D of least degree that is a solution to the equation

$$T(s) = Q(s)D(s)^{-1} \qquad (5.17)$$

and satisfies

$$X(s)Q(s) + Y(s)D(s) = I \qquad (5.18)$$

It is easy to verify that the choice

$$Q(s) = I, \quad D(s) = \begin{bmatrix} s(s-\lambda_1) & -(\alpha_1 + \alpha_2 s) \\ -(\beta_1 + \beta_2 s) & s(s-\lambda_2) \end{bmatrix}.$$

is a solution to equation 5.17, and there exists $X(s)$ and $Y(s)$ so that equation 5.18 is satisfied. From $Q(s)$ we can see that the system has no zeros. The zero initial conditions should thus not give us any problems.

5.1.2 Derivative approximation

Instead of setting the velocity equal to zero, another alternative is of course to specify a starting velocity. However, this requires that we make a good choice, to avoid situations as exemplified below. Using a bad direction and a high velocity boundary condition on $y = x^2$, we get the graph of figure 1. Even as we are using n=40 to reproduce the curve, the bad boundary conditions are still ruining the tracking.

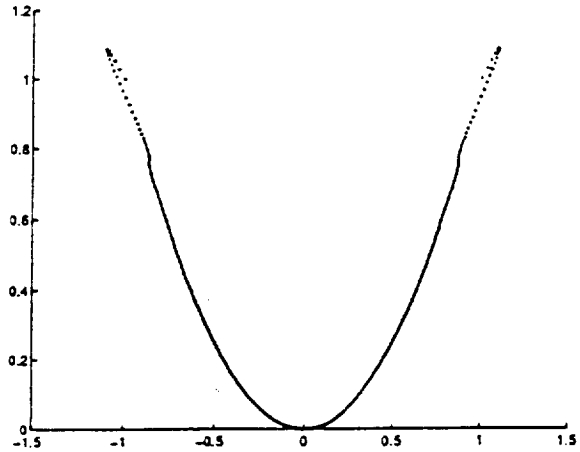


Figure 1: Trajectory of system tracking $y = x^2$, with badly specified starting direction

As discussed in [3, p.86], the fact is that when we set the boundary conditions in the parametric case, we do not only specify a direction, but also the speed in that direction. The greater the speed, the greater impact

the boundary conditions will have on the solution. We are thus forced to make a good choice, and we would like the choice to get better the more interpolation points we are using. A simple choice that satisfies this is

Assumption 5.2

$$\begin{aligned}(\dot{x}_0, \dot{y}_0) &= \frac{x_1 - x_0}{h} \\ (\dot{x}_n, \dot{y}_n) &= \frac{x_n - x_{n-1}}{h}\end{aligned}$$

which imply that we will set off from the first point in the direction of the second point, and arrive at the last point in the direction from the next last point.

Another way of deciding the initial velocity would be to use the same technic as in Bezier curves and choose the settings graphically. This would probably be the best way to get the desired properties of the signatures. As discussed in [3] the choice of boundary conditions will affect the whole curve, and the solving of the blockdiagonal system must be done over from scratch, making this method a bit slow. If we are going to do this only once to store a signature it does not matter. What matters with this method is that it adds four more parameters to be stored, and we could probably get equally good results just by increasing the number of interpolation points by two.

5.2 Constant velocity

Suppose we want to use the boundary conditions

Assumption 5.3

$$\begin{aligned}(\ddot{x}_0, \ddot{y}_0) &= 0 \\ (\ddot{x}_n, \ddot{y}_n) &= 0\end{aligned}$$

This will let the initial direction and constant velocity of the system be decided so that the control energy is minimized. Using the system dynamics equations 3.7, we get the equation system below.

$$\begin{bmatrix} \lambda_1 & \alpha_2 \\ \beta_2 & \lambda_2 \end{bmatrix} \begin{bmatrix} \dot{x}_0 \\ \dot{y}_0 \end{bmatrix} + \mathrm{u}_0(0) = - \begin{bmatrix} \alpha_1 y_0 \\ \beta_1 x_0 \end{bmatrix} \tag{5.19}$$

where

$$\begin{aligned} \mathrm{u}_0(0) &= B' M^{-1}(e^{-Ah}\mathrm{x}_1 - \mathrm{x}_0) && (5.20) \\ &= B' Z \mathrm{x}_1 - B' M^{-1} \mathrm{x}_0 && (5.21) \end{aligned}$$

Now, by using partitioned matrixes as in section 4 and the following definition,

Definition 5.1 $$U_{lu} \triangleq \begin{bmatrix} m_{22}^{-1} & m_{24}^{-1} \\ m_{42}^{-1} & m_{44}^{-1} \end{bmatrix}, \qquad U_{ru} \triangleq \begin{bmatrix} m_{21}^{-1} & m_{41}^{-1} \\ m_{23}^{-1} & m_{43}^{-1} \end{bmatrix}$$

we get the system

$$\begin{aligned} &\left(\begin{bmatrix} \lambda_1 & \alpha_2 \\ \beta_2 & \lambda_2 \end{bmatrix} - U_{lu} \right) \mathrm{x}_0^{vel} + Z_{lu}^B \mathrm{x}_1^{vel} = && (5.22) \\ &= \left(U_{ru} - \begin{bmatrix} 0 & \alpha_1 \\ \beta_1 & 0 \end{bmatrix} \right) \mathrm{x}_0^{pos} - Z_{ru}^B \mathrm{x}_1^{pos} \end{aligned}$$

In a similar way we get the equation at the other boundary.

$$\begin{aligned} &\left(\begin{bmatrix} \lambda_1 & \alpha_2 \\ \beta_2 & \lambda_2 \end{bmatrix} + W_l^B - U_{lu} \right) \mathrm{x}_n^{vel} - Z_{ll}^B \mathrm{x}_{n-1}^{vel} = && (5.23) \\ &= \left(U_{ru} - W_r^B - \begin{bmatrix} 0 & \alpha_1 \\ \beta_1 & 0 \end{bmatrix} \right) \mathrm{x}_n^{pos} + Z_{rl}^B \mathrm{x}_{n-1}^{pos} \end{aligned}$$

Adding these two equations to equation 4.14 yields a blockdiagonal system to solve. This system is two blocks bigger than the one in section 5.1, but it can also be solved with a linear increase of time. And once it is solved, we can still use equation 5.15.

A comparison between the three boundary conditions, BC=1 zero initial velocity, BC=2 derivative approximation, and BC=3 zero initial acceleration is made in section 6.

6 Results

The first tests with the algorithm were done with matrices A with both eigenvalues equal to zero, $\lambda_1 = 0, \lambda_2 = 0$, and $\alpha_1 = \alpha_2 = \beta_1 = \beta_2 = 0$. This produces cubic parametric splines, and makes the calculation of the fundamental matrix easy. The cubic splines produce smooth curves, but were also able to reconstruct a cusp much better than you would have guessed at first, as shown in figure 2.

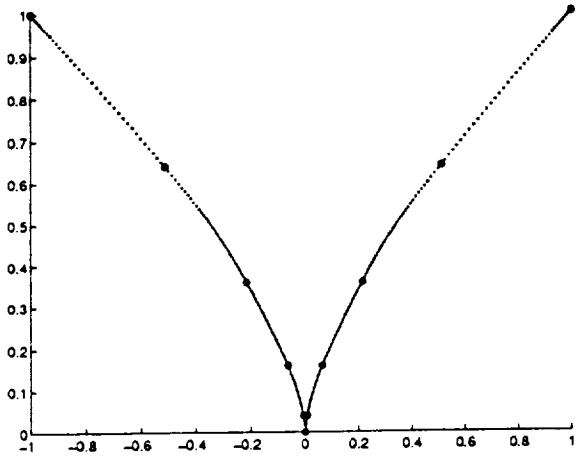


Figure 2: Reconstruction of $y = x^{\frac{2}{3}}$ with cubic parametric splines where n=10.

If we look at the function $y = x^{\frac{2}{3}}$ we know that this function describes a cusp. But if we parameterized it like $x = t^3, y = t^2$ we see that both x and y are smooth cubic functions of t, so it is not very remarkable that it can be reconstructed well.

When we used A-matrices with nonzero eigenvalues, and decoupled x and y coordinates, i.e. $\alpha_1 = \alpha_2 = \beta_1 = \beta_2 = 0$, we were able to generate exponential parametric splines with the basis functions $1, \lambda_1 t, e^{\lambda_1 t}, e^{-\lambda_1 t}$ and $1, \lambda_2 t, e^{\lambda_2 t}, e^{-\lambda_2 t}$ for the x and y coordinates.

The result of taking big eigenvalues is almost linear interpolation, which can be good for certain applications, but not if it is to be used for storing signatures. It is quite obvious that the roundness of a persons signature is one important factor of it's characteristics. Therefore, it's vital that one of

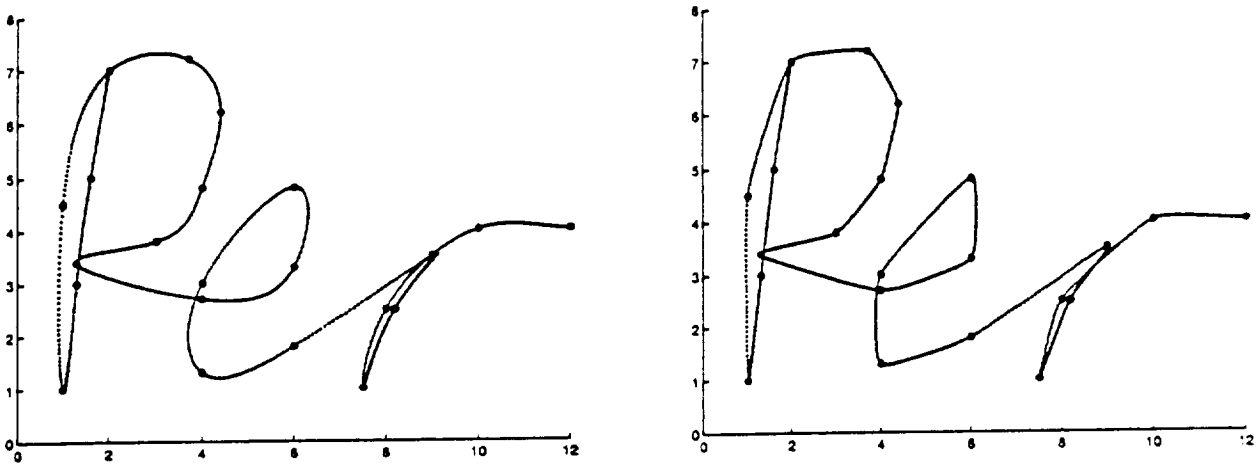


Figure 3: Graph of signature Per, reproduced with $\lambda_1 = 1, \lambda_2 = 1$ in the left figure, and $\lambda_1 = 100, \lambda_2 = 100$ in the right

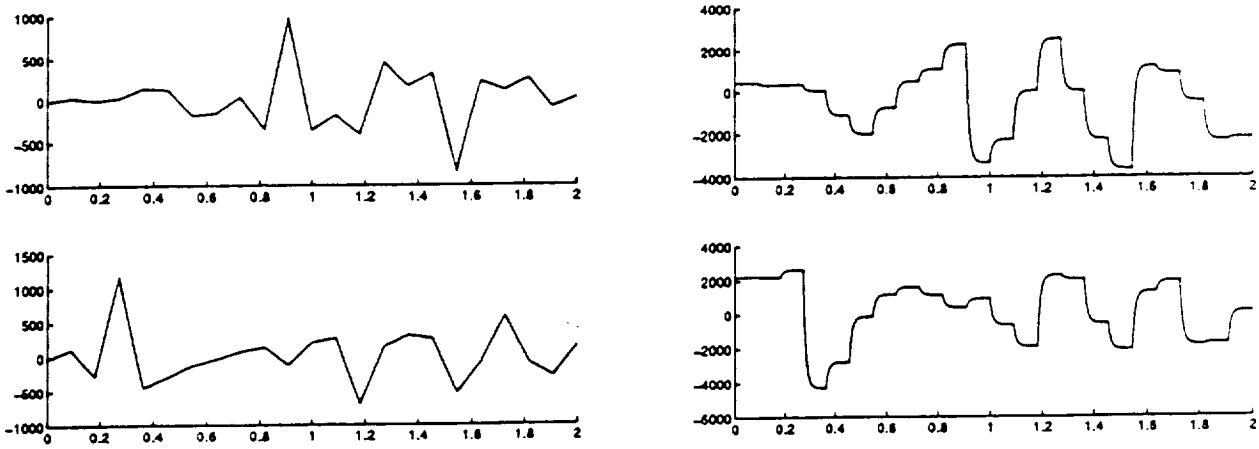


Figure 4: Graph of control signals in figure above.

data stored on the signature is the eigenvalues of the A-matrix, as can be shown in figure 3.

Figure 4 shows the corresponding control signals. In the $\lambda_1 = 1, \lambda_2 = 1$ case the linear part of the control is dominating, and in the $\lambda_1 = 100, \lambda_2 = 100$ case the exponential part is dominating. It is also evident that the magnitude of the control signals is greater in the case of larger eigenvalues, but that is not a problem for our fictitious system.

In the case of nonzero $\alpha_1, \alpha_2, \beta_1, \beta_2$ we get a coupling between the x and y coordinates. This will give us very complicated basis functions, like polynomials times exponentials times sine- and cosine-functions.

As with all approximation methods one should always investigate how big the errors are. To do this we had to somehow determine the distance between the original curve and the interpolating one. The tests were performed on known parametric curves, so we had an explicit expression for the points on the original curve. We had to try to find the nearest point on the original curve to the point on the trajectory. This was solved numerically with the "Golden Intersection algorithm" . For the method to work we have to assume two things :

1. The section of the original curve between the two interpolation points nearest in time is convex.

2. The point on the original curve nearest the point on the interpolating curve lies on the section in item 1.

As an error estimate I have calculated the distance between a number of points on the reproduced curve and the original curve and divided with the number of points. We have applied this error estimate method on four different curves and with different number of interpolation points and 40 points between each of these.

points	$BC = 1$	$BC = 2$	$BC = 3$
$n = 10$	6671.1	5821.1	4515.9
$n = 20$	1028.4	765.6	502.1
$n = 100$	8.8	5.9	3.5

Table 1: μ units of error for unit circle.

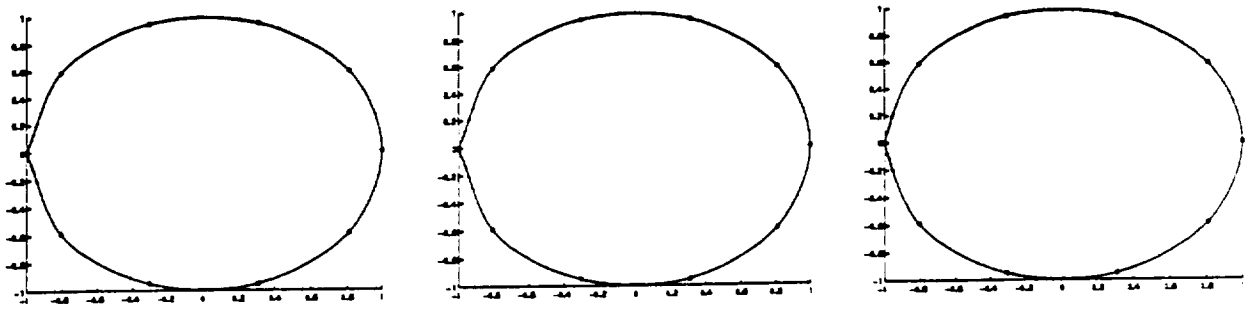


Figure 5: Graph of circle, reproduced with n=10 and $\lambda_1 = 10, \lambda_2 = 10$, BC=1,2,3

6.1 Four test curves

The first curve we tested was the unit circle. This very round curve was tracked best by the cubic splines, but the exponential splines did a good job too, as can be seen in figure 5 and table 1. We can also see that the error was smallest in the case of zero initial acceleration boundary conditions.

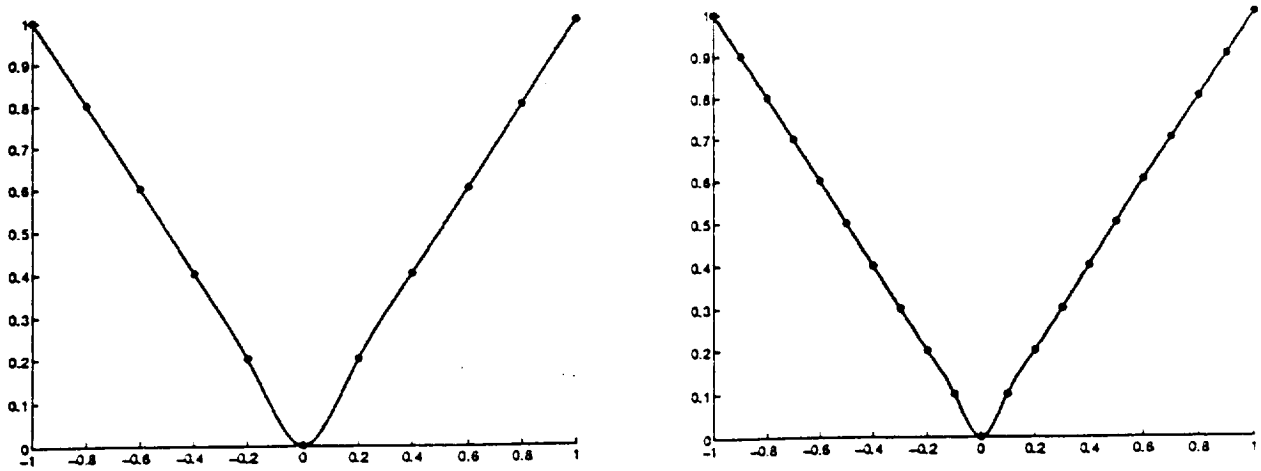


Figure 6: Graph of $y = |x|$, reproduced with n=10, n=20 and $\lambda_1 = 10, \lambda_2 = 10$, BC=2

Next, we looked at a curve with the opposite properties, linear and non-differentiable, $y = |x|$. The error is mainly located between the two points

points	$BC = 1$	$BC = 2$	$BC = 3$
$n = 10$	3450.3	3446.8	3450.3
$n = 20$	972.6	972.6	972.6
$n = 100$	40.7	40.7	40.7

Table 2: μ units of error for $y = |x|$.

next to the non-differentiable point. As could be guessed the tracking of the curve $y = |x|$ got better the bigger the eigenvalues of the A matrix were, the error was reduced to 142 μ units for $\lambda_1 = \lambda_2 = 500$ in the case of $BC = 1$ and $n = 10$, but for bigger values the numerical calculations failed. Using negative eigenvalues gives the same error as the positive, which can be expected since we have a symmetric curve and time interval and by looking at the basis functions. The results with different boundary conditions were very much alike, as seen in table 2.
This was the only case were BC 3 did not give us the smallest error.

Can we get a smaller error with any choice of the coupling parameters? Yes, for example by choosing $\alpha_1 = -\alpha_2 = \beta_1 = -\beta_2 = 10$, we get the error 3249 μ units for n=10. Choosing these parameters could thus be a way to reduce the error, but by using $n = 12$ instead, we got the error 2506 μ units. So we do not get a more efficient way to store it, unless we can find parameters so we get below 2506 μ units.

points	$BC = 1$	$BC = 2$	$BC = 3$
$n = 10$	6506.6	5313.7	3842.9
$n = 20$	1014.4	725.4	460.9
$n = 100$	8.8	5.8	3.4

Table 3: μ units of error for cycloid.

points	$BC = 1$	$BC = 2$	$BC = 3$
$n = 10$	13897.6	11664.8	8867.6
$n = 20$	2100.5	1531.0	1001.7
$n = 100$	17.7	11.8	7.0

Table 4: μ units of error for prolate cycloid.

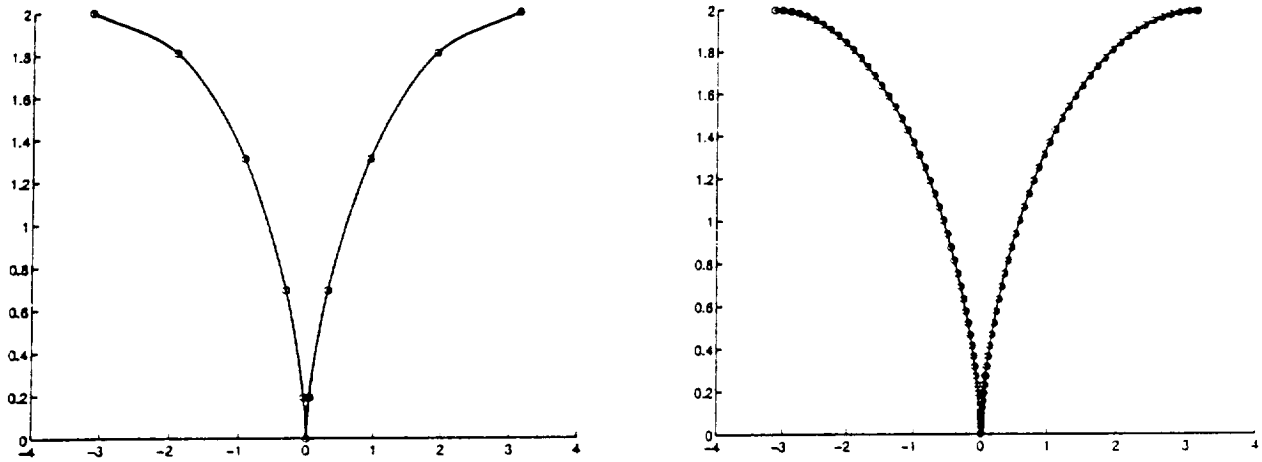


Figure 7: Graph of cycloid, BC= 2, reproduced with n=10, n=100 and $\lambda_1 = 10, \lambda_2 = 10$

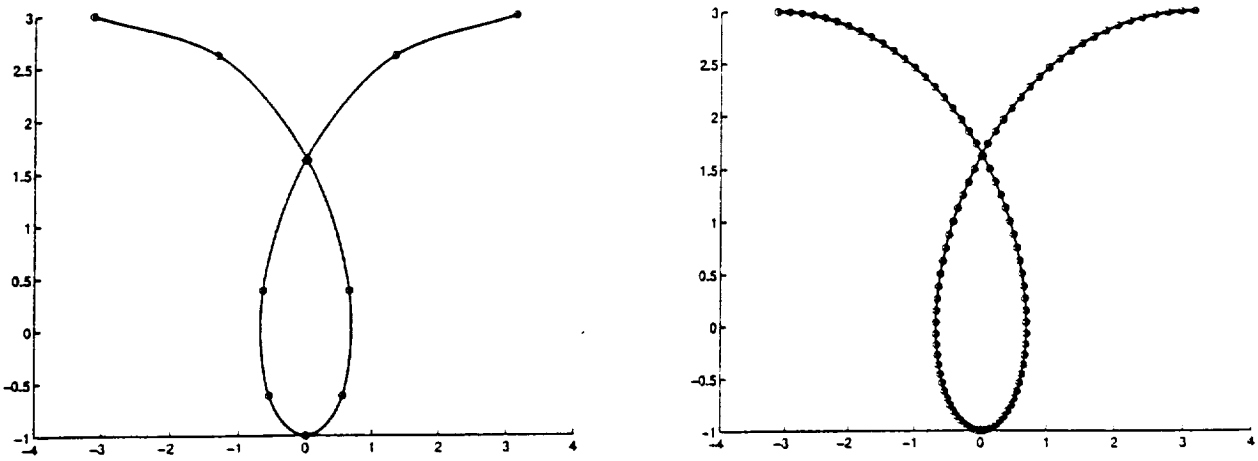


Figure 8: Graph of prolate cycloid, BC= 2, reproduced with n=10, n=100 and $\lambda_1 = 10, \lambda_2 = 10$

Finally we look at a cycloid.

$$\begin{cases} x &= \pi t - \sin \pi t \\ y &= 1 - \cos \pi t \end{cases}$$

and a prolate cycloid.

$$\begin{cases} x &= \pi t - 2 \sin \pi t \\ y &= 1 - 2 \cos \pi t \end{cases}$$

It is evident that the cusp and the crossover do not cause any problem, as could be expected since we are using a parameterized interpolant.

We can compare the difference of the graphs in figure 7 and figure 8, where we are using one crude approximation with n=10 and one extensive approximation with n=100. This time it is evident that the error is mainly located at the boundaries. In table 3 and table 4 we can see that the best results came from using constant velocity boundary conditions.

Looking at table 3 and table 4 again, we see that the error decreases at an approximately cubic rate as the number of interpolation points increase, which is much better than the quadratic decrease that can be seen in table 2.

My guess is that this behavior comes from the fact that the curve $y = |x|$ does not have a differentiable parameterization.

6.2 Applied on a signature

Included as figure 9 is a scanned picture of my own signature. I have tried to pick some roughly equidistant points on the signature, (According to the scale indicated on the axis.) and used the interpolation algorithm we have been studying to reproduce it. The reproduction is made with $n = 74$, $\lambda_1 = \lambda_2 = 10$ and no coupling between the two coordinates. For boundary conditions I have chosen to use constant velocity, since it has been the most successful condition.

As can be seen we get a very close resemblance between the original and the reproduction. How close is hard to say because we do not have the signature given as a parameterized function, therefore we are not able to calculate the error as before.

The things characterizing the signatures, are also the things that are hard to recover with the interpolation. Such as the turnover in the connection from the "P", and the cusps in the "r". To get a good reproduction, an equidistant

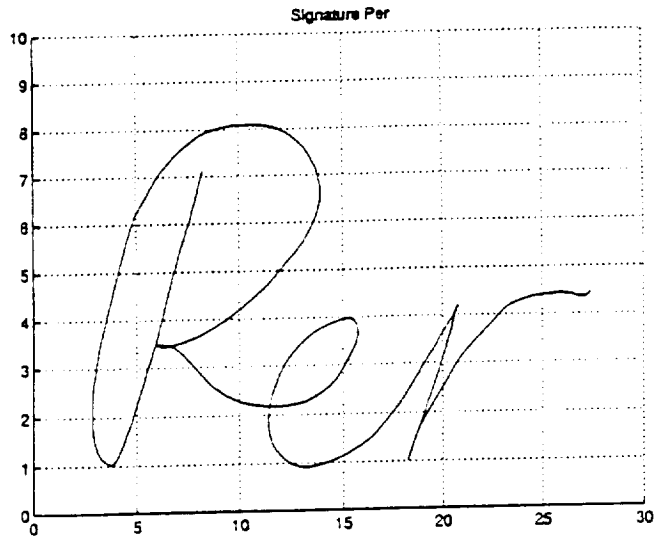


Figure 9: The scanned signature

Figure 10: The reproduced signature

distribution of the interpolation points is not enough, more points has to be concentrated around the characterizing areas.

7 Resumé - Summary in Swedish

Lagring av signaturer kan göras på många sätt. Vi har valt att lagra ett antal punkter på signaturen, och sedan reproducera denna genom att interpolera punkterna med splines.

Genom att använda välkända resultat inom systemteori så kan man generera olika sorters splines genom att ändra på några parametrar. Jag har nyttjat denna metod för att generera parametriserade splines i planet. Man inser snart att man måste införa randvillkor på lösningen, och valet av dessa får inte ske hur som helst eftersom de påverkar hela lösningen. Därför har jag lagt ner en hel del arbete just på denna punkt. De bästa resulteten har jag erhållit genom valet att ha konstant hastighet vid ändpunkterna.

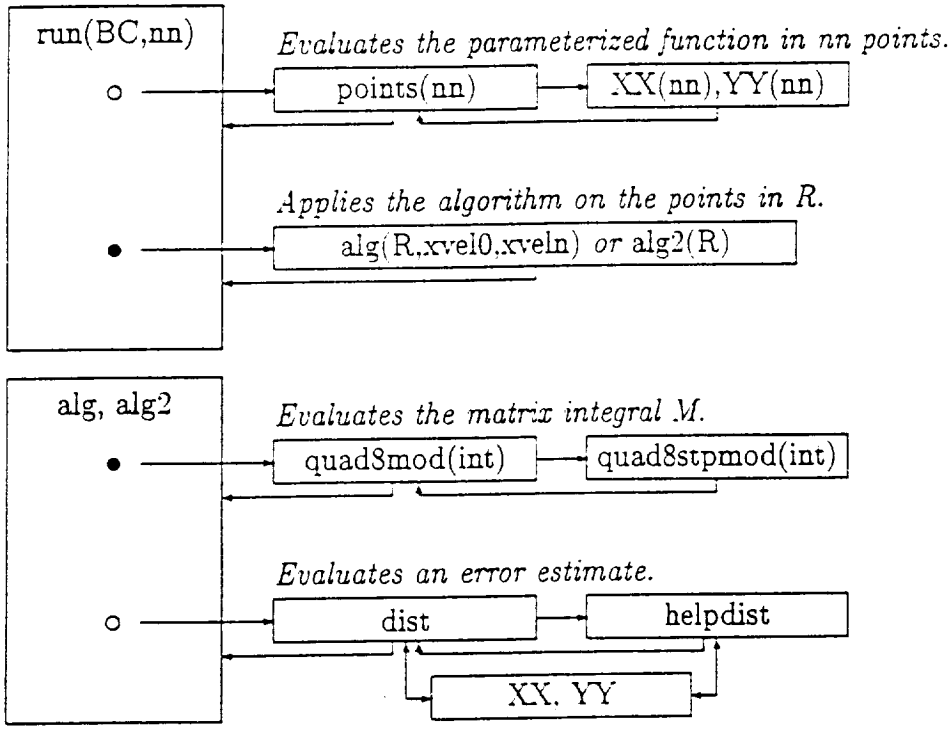
En av de saker som karaktäriserar handstilar är dess rundhet. Denna kan ges en direkt översättning i egenvärdena till systemmatrisen, och vi kommer alltså välja att lagra dessa utöver punkterna på signaturen.

8 Programs

The programs in Matlab and Maple that were used to implement the algorithm developed in this report follow.

8.1 Matlab Program

To make it easier to understand the structure of the program, the following flow charts describe how the programs are traversed.



The loops marked with an unfilled circle is only available when the interpolated points are given by the parameterized function (XX(t),YY(t)).

```
function error=run(BC,nn);
%BC              % Type of boundary conditions
                 % 1 = zero initial and final velocity.
                 % 2 = heading for first point, last.
                 % 3 = zero initial and final acceleration.
% If nn is specified, runs alg with nn points given by XX,YY.
% Otherwise runs alg with points given by ginput.
global n h alpha1 alpha2 beta1 beta2 lambda1 lambda2
global errorcalc ctrlsignal
clg

%% Setting of parameters %%
alpha1=0;
alpha2=0;
beta1=0;
beta2=0;
lambda1=10;
lambda2=10;

% Decides what steps are going to be made
errorcalc=1;        % error estimation
ctrlsignal=0;       % plotting of control signals


if nargin == 1
  [x,y]=ginput;
  R(1,:)=x'; R(2,:)=y';
else
  R=points(nn);
end;

n=length(R)-1;    % Number of interpolationpoints -1.
h=2/n;            % Time inbetween points

%% Plots a circle at all the points thats interpolated %%
hold on
for i=1:n+1
```

```
  plot(R(1,i),R(2,i),'o')
end;

%% Calling alg with the prepared data %%
if BC == 3
   error=alg2(R);
else
  if BC == 2, xvel0=(R(:,2)-R(:,1))/h;
    else xvel0=zeros(2,1);
  end;
  if BC == 2, xveln=(R(:,n+1)-R(:,n))/h;
    else xveln=zeros(2,1);
  end;
  error=alg(R,xvel0, xveln);
end;

%% Loop to allow graphic alteration of BC. %%

b=input('"1" for graphic mod of BC, "0" to quit  ');
while b == 1,
  xvel0=10*(ginput(1)'-R(:,1));
  xveln=-10*(ginput(1)'-R(:,n+1));
  clg
  hold on
  for i=1:n+1
    plot(R(1,i),R(2,i),'o')
  end;
  error=alg(R,xvel0,xveln);
  b=input('"1" for graphic mod of BC, "0" to quit ');
end;

end;
```

```
function R=points(nn);
% Forms R  with the help of XX, YY.
```

```
% R = 2*(nn+1)-matrix.
global a b h

a=1; b=a;
h=2/nn;

for i= 0:nn
  R(:,i+1)=[XX(-1+i*h); YY(-1+i*h)];
end
end;
```

```
function res=XX(t);
global a b
  res= a*t*pi-b*sin(t*pi);
end;
```

```
function res=YY(t)
global a b
  res= a-b*cos(t*pi);
end;
```

```
function error=alg(R,xvel0,xveln);
% R = Matrix of interpolationpoints,  x0 ... xn.
%    first row = x-coordinates, second row = y-coordinates.
% xvel0, xveln = Boundary conditions

global a b A B C n h alpha1 alpha2 beta1 beta2 lambda1 lambda2
global errorcalc ctrlsignal

%% The System %%

A=[[ 0  1  0  0]
   [ 0  lambda1  alpha1  alpha2]
```

```
   [ 0   0   0 1]
   [ beta1 beta2 0 lambda2]];
B=[[0 0]
   [1 0]
   [0 0]
   [0 1]];
C=[[1 0 0 0]
   [0 0 1 0]];

%% Calculation of the integral from 0 to h in m steps %%
m=40;              % number of points between interpolationpoints
tol=1e-08;         % the numeric error tolerance

Mtau(:,1:4)=zeros(4);
tau=0;
for j=1:m
  oldtau=tau;
  tau=oldtau+h/m;
  Mtau(:,4*j+1:4*j+4)= quad8mod('int',oldtau,tau,tol)
                       + Mtau(:,4*j-3:4*j);
end;
M=Mtau(:,4*m+1:4*m+4);

%% Forming of the Matrixes for the Blockdiagonal system %%

e_Ah=expm(-A*h);
Minv=inv(M);
ZZ=Minv*e_Ah;
WW=e_Ah'*ZZ+Minv;

WL=[WW(2,2) WW(2,4); WW(4,2) WW(4,4)];  % Partitioning matrixes
ZLU=[ZZ(2,2) ZZ(2,4); ZZ(4,2) ZZ(4,4)];
ZLL=[ZZ(2,2) ZZ(4,2); ZZ(2,4) ZZ(4,4)];

WR=[WW(2,1) WW(2,3); WW(4,1) WW(4,3)];
ZRU=[ZZ(2,1) ZZ(2,3); ZZ(4,1) ZZ(4,3)];
ZRL=[ZZ(1,2) ZZ(3,2); ZZ(1,4) ZZ(3,4)];
```

```
%% The boundary conditions  %%

xvel(:,1)=xvel0;
xvel(:,n+1)=xveln;

%%  Forming of the right side of the Blockdiagonal system  %%

for i=2:n
  Omega(:,i)=ZRL*R(:,i-1)-WR*R(:,i)+ZRU*R(:,i+1);
end;
Omega(:,2)=Omega(:,2)+ZLL*xvel(:,1);
Omega(:,n)=Omega(:,n)+ZLU*xvel(:,n+1);

%%  Gausselimination to produce upper triangular system  %%

DD(:,3:4)=WL;
for i=3:n
  zd=ZLL*inv(DD(:,2*i-3:2*i-2));
  DD(:,2*i-1:2*i)=WL-zd*ZLU;
  Omega(:,i)=Omega(:,i)+zd*Omega(:,i-1);
end

%%  Backsubstitution to solve for the xvel  %%

xvel(:,n)=DD(:,2*n-1:2*n)\Omega(:,n);
for i=n-1:-1:2
  xvel(:,i)=DD(:,2*i-1:2*i)\(ZLU*xvel(:,i+1)+Omega(:,i));
end;

%% Making of state vectors %%

for i=0:n
  x(:,i+1)=[[R(1,i+1)]
            [xvel(1,i+1)]
            [R(2,i+1)]
            [xvel(2,i+1)]];
```

```
function error=alg2(R);
% R = Matrix of interpolationpoints,  x0 ... xn.
%    first row = x-coordinates, second row = y-coordinates.
% BC = acceleration in x and y direction are both = 0.

global a b A B C n h alpha1 alpha2 beta1 beta2 lambda1 lambda2
global errorcalc ctrlsignal

%% The System %%

A=[[ 0  1  0  0]
   [ 0  lambda1  alpha1  alpha2]
   [ 0  0  0 1]
   [ beta1 beta2 0 lambda2]];
B=[[0 0]
   [1 0]
   [0 0]
   [0 1]];
C=[[1 0 0 0]
   [0 0 1 0]];

%% Calculation of the integral from 0 to h in m steps %%
m=40;              % number of points between interpolationpoints
tol=1e-08;         % the numeric error tolerance

Mtau(:,1:4)=zeros(4);
tau=0;
for j=1:m
  oldtau=tau;
  tau=oldtau+h/m;
  Mtau(:,4*j+1:4*j+4)= quad8mod('int',oldtau,tau,tol)
                       + Mtau(:,4*j-3:4*j);
end;
M=Mtau(:,4*m+1:4*m+4);
```

```
%%  Forming of the Matrixes for the Blockdiagonal system  %%

e_Ah=expm(-A*h);
Minv=inv(M);
ZZ=Minv*e_Ah;
WW=e_Ah'*ZZ+Minv;

WL=[WW(2,2) WW(2,4); WW(4,2) WW(4,4)];  % Partitioning matrixes
ZLU=[ZZ(2,2) ZZ(2,4); ZZ(4,2) ZZ(4,4)];
ZLL=[ZZ(2,2) ZZ(4,2); ZZ(2,4) ZZ(4,4)];

WR=[WW(2,1) WW(2,3); WW(4,1) WW(4,3)];
ZRU=[ZZ(2,1) ZZ(2,3); ZZ(4,1) ZZ(4,3)];
ZRL=[ZZ(1,2) ZZ(3,2); ZZ(1,4) ZZ(3,4)];

Ulu=[Minv(2,2) Minv(2,4);Minv(4,2) Minv(4,4)];
Uru=[Minv(2,1) Minv(4,1);Minv(2,3) Minv(4,3)];
Vl=[lambda1 alpha2; beta2 lambda2];
Vr=[0 alpha1; beta1 0];

%%  Forming of the right side of the Blockdiagonal system  %%

for i=2:n
  Omega(:,i)=ZRL*R(:,i-1)-WR*R(:,i)+ZRU*R(:,i+1);
end;
Omega(:,1)=(Vr-Uru)*R(:,1) + ZRU*R(:,2);
Omega(:,n+1)=ZRL*R(:,n) - (WR-Uru+Vr)*R(:,n+1);

%%  Gausselimination to produce upper triangular system  %%

DD(:,1:2)=Ulu-Vl;
for i=2:n+1
  zd=ZLL*inv(DD(:,2*i-3:2*i-2));
  DD(:,2*i-1:2*i)=WL-zd*ZLU;
  Omega(:,i)=Omega(:,i)+zd*Omega(:,i-1);
end
DD(:,2*n+1:2*n+2)= (WL-Ulu+Vl) - zd*ZLU;
```

```
%%  Backsubstitution to solve for the xvel  %%

xvel(:,n+1)=DD(:,2*n+1:2*n+2)\Omega(:,n+1);
for i=n:-1:1
  xvel(:,i)=DD(:,2*i-1:2*i)\(ZLU*xvel(:,i+1)+Omega(:,i));
end;

%% Making of state vectors %%

for i=0:n
  x(:,i+1)=[[R(1,i+1)]
            [xvel(1,i+1)]
            [R(2,i+1)]
            [xvel(2,i+1)]];
end;

%%  Plotting of trajectory
%%  and error estimate calculation

sumnorm=0;
hold on
for j=0:m
  eAtau=expm(A*j*h/m);
  for i=0:n-1
    entry=eAtau*(x(:,i+1)+Mtau(:,4*j+1:4*j+4)*Minv*
                          (e_Ah*x(:,i+2)-x(:,i+1)));
    plot(entry(1),entry(3),'.')
    if errorcalc
      sumnorm = sumnorm + dist(entry(1),entry(3),i,h);
    end;
  end;
end;

%% Plotting of the control signals %%

if ctrlsignal
```

```
  pause, clg
  subplot(2,1,1),hold on
  subplot(2,1,2),hold on
  for i=0:n-1
    for j=0:m
      csignvec(:,j+1)=B'*expm(-A'*j*h/40)*Minv*
                      (e_Ah*x(:,i+2)-x(:,i+1));
    end;
    subplot(2,1,1),plot(i*h:h/m:(i+1)*h,csignvec(1,:))
    subplot(2,1,2),plot(i*h:h/m:(i+1)*h,csignvec(2,:))
  end;
end;

error=sumnorm/n/m;
end;
```

```
function [Q,cnt] = quad8mod(F,a,b,tol)
%Alteration of the original matlab toolbox program. QUAD8
% Numerical evaluation of an integral, higher order method.
% Q = QUAD8('F',A,B,TOL) approximates the integral of F(X)
% from A to B to within a relative error of TOL.
% 'F' is a string containing the name of the function.
% The function must return a 4*4-matrix output value if
% given an input value.
% Q = Inf is returned if an excessive recursion level is
% reached, indicating a possibly singular integral.
% QUAD8 uses an adaptive recursive Newton Cotes 8 panel rule.
% Cleve Moler, 5-08-88.
% Copyright (c) 1984-94 by The MathWorks, Inc.
% [Q,cnt] = quad8(F,a,b,tol) also returns a function
%           evaluation count.

% Top level initialization, Newton-Cotes weights
w=[3956 23552 -3712 41984 -18160 41984 -3712 23552 3956]/14175;
x=a + (0:8)*(b-a)/8;
```

```
% set up function call
for i=x
  y = [y feval(F,i)];
end;

% Adaptive, recursive Newton-Cotes 8 panel quadrature
Q0 = zeros(4);
[Q,cnt] = quad8stpmod(F,a,b,tol,0,w,x,y,Q0);
cnt = cnt + 9;
end;
```

```
function [Q,cnt] = quad8stpmod(F,a,b,tol,lev,w,x0,f0,Q0)
%Alteration of the original matlab toolbox program.QUAD8STP
% Recursive function used by QUAD8.
% [Q,cnt] = quad8stp(F,a,b,tol,lev,w,f,Q0) tries to approximate
% the integral of f(x) from a to b to within a relative error
% of tol.
% F is a string containing the name of f. The remaining
% arguments are generated by quad8mod or by the recursion.
% lev is the recursion level.
% w is the weights in the 8 panel Newton Cotes formula.
% x0 is a vector of 9 equally spaced abscissa is the interval.
% f0 is a matrix of the 9 function values at x.
% Q0 is an approximate value of the integral.
% Cleve Moler, 5-08-88.
% Copyright (c) 1984-94 by The MathWorks, Inc.

LEVMAX = 10;
% Evaluate function at midpoints
% of left and right half intervals.
x = zeros(1,17);
x(1:2:17) = x0;
x(2:2:16) = (x0(1:8) + x0(2:9))/2;

f(:,1:4)= f0(:,1:4);
for i=1:8
```

```
  f(:,8*i-3:8*i) = feval(F,x(2*i));
  f(:,8*i+1:8*i+4) = f0(:,4*i+1:4*i+4);
end;

% Integrate over half intervals.
h = (b-a)/16;
Q1=0;Q2=0;
for i=1:9
  Q1 = Q1 + h*w(i)*f(:,4*i-3:4*i);
  Q2 = Q2 + h*w(10-i)*f(:,69-i*4:72-i*4);
end;
Q = Q1 + Q2;

% Recursively refine approximations.
if norm(Q - Q0) > tol*norm(Q) & lev <= LEVMAX
   c = (a+b)/2;
   [Q1,cnt1] =
      quad8stpmod(F,a,c,tol/2,lev+1,w,x(1:9),f(:,1:36),Q1);
   [Q2,cnt2] =
      quad8stpmod(F,c,b,tol/2,lev+1,w,x(9:17),f(:,33:68),Q2);
   Q = Q1 + Q2;
   cnt = cnt + cnt1 + cnt2;
end
end;
```

```
function res = integrand(v)
global A B C

e_AvB=expm(-A*v)*B;
res = e_AvB*e_AvB';
end;
```

```
function [d]=dist(xx,yy,i,h);
% Initiating search algorithm.
```

8.2 Maple Program

```
with(linalg);
with(student);

alpha1:=1;
alpha2:=1;
beta1:=0;
beta2:=0;
lambda1:=100;
lambda2:=100;

a:=1;
b:=a;
h:=0.2;
n:=22;
m:=15;
R:=vector(n+1);.
XX:=t->a*t*Pi-b*sin(t*Pi);
YY:=t->a-b*cos(t*Pi);
for i from 0 to n
do
  R[i+1]:=matrix([[XX(-1+i*h)],
                  [YY(-1+i*h)]]);
od;

A:=matrix([[ 0, 1, 0, 0],
           [ 0, lambda1, alpha1, alpha2],
           [ 0, 0, 0, 1],
           [ beta1, beta2, 0, lambda2]]);
B:=matrix([[0,0],[1,0],[0,0],[0,1]]);
C:=matrix([[1,0,0,0],[0,0,1,0]]);

alias(Id = &*())
Aprim:=transpose(A);
Bprim:=transpose(B);
e_At:=t->exponential(-A*t);
```

```
e_Aprimt:=t->exponential(-Aprim*t);
eAt:=t->exponential(A*t);
e_Ah:=e_At(h);

integranden:= proc (v)
    evalm(e_At(v) &* B &* Bprim &* e_Aprimt(v));
end;

integrera:= proc (funk)
   global v;
   int(funk,v);
end;

map(integrera,integranden(v));
integralen:=map(simplify,");

evaluera:=proc (funk)
 global tau;
 subs(v=tau,funk);
end;

Mtau:=vector(m+1);
Mtau[1]:=matrix([[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]);
tau:=0;
M0:=evalm(map(evaluera,integralen));

for j from 1 to m
do
  tau:=j*h/m;
  Mtau[j+1]:=evalm(map(evaluera,integralen)-M0);
od;
M:=Mtau[m+1];

Minv:=evalm(inverse(M));
ZZ:=evalm(Minv&*e_Ah);
WW:=evalm(transpose(e_Ah)&*ZZ+Minv);
```

```
WL:=submatrix(WW,[2,4],[2,4]);
ZLU:=submatrix(ZZ,[2,4],[2,4]);
ZLL:=transpose(submatrix(ZZ,[2,4],[2,4]));

WR:=submatrix(WW,[2,4],[1,3]);
ZRU:=submatrix(ZZ,[2,4],[1,3]);
ZRL:=transpose(submatrix(ZZ,[1,3],[2,4]));

xvel:=vector(n+1);
xvel[1]:=evalm((R[2]-R[1])/h);
xvel[n+1]:=evalm((R[n+1]-R[n])/h);

Omega:=vector(n+1);
for i from 2 to n
do
  Omega[i]:=ZRL&*R[i-1]-WR&*R[i]+ZRU&*R[i+1];
od;
Omega[2]:=Omega[2]+ZLL&*xvel[1];
Omega[n]:=Omega[n]+ZLU&*xvel[n+1];

DD:=vector(n+1);
DD[2]:=WL;
for i from 3 to n
do
  zd:=evalm(ZLL&*inverse(DD[i-1]));
  DD[i]:=WL-zd&*ZLU;
  Omega[i]:=Omega[i]+zd&*Omega[i-1];
od;

xvel[n]:=linsolve(DD[n],Omega[n]);
for i from n-1 by -1 to 2
do
  xvel[i]:=linsolve(DD[i],ZLU&*xvel[i+1]+Omega[i]);
od;

x:=vector(n+1);
for i from 0 to n
```

```
do
  x[i+1]:=matrix([[R[i+1][1,1]],
                  [xvel[i+1][1,1]],
                  [R[i+1][2,1]],
                  [xvel[i+1][2,1]]]);
od;

plotlist:=[];
for j from 1 to m
do
  tau:=j*h/m;
  eAtau:=evalm(eAt(tau));

  for i from 0 to n-1
  do
    entry:=evalm(eAtau&*(x[i+1]+Mtau[j+1]&*Minv
                                  &*(e_Ah&*x[i+2]-x[i+1])));
    plotlist:={[entry[1,1],entry[3,1]],op(plotlist)};
  od;
od;

plot(plotlist,style=point);
```

References

[1] Hildebrand, F. B. Introduction to Numerical Analysis. *New York: Dover Publications, Inc.* (1987).

[2] Kahaner, D., Moler, C., Nash, S. Numerical Methods and Software. *Eaglewood Cliffs: Prentice-Hall International, Inc.* (1989).

[3] Faires J.D., Burden L.R. Numerical Methods. *Boston: PWS-KENT Publishing Company* (1993).

[4] Taylor A.E. Introduction to Functional Analysis.*New York: Wiley* (1958).

[5] Selby S.M. CRC Standard Mathematical Tables. *Cleveland: The Chemical Rubber Co.* (1972)