Use of delay differential equations to approximate some decaying functions using exponentials

Anindya Chatterjee
Mechanical Engineering
Indian Institute of Science
Bangalore, India
anindya100@gmail.com

This draft: August 24, 2009

1 A dynamic model

We wish to approximate a somewhat arbitrary function $x(t)$ (for $t > 0$) that eventually decays to zero as t increases. We wish to approximate it using a linear combination of decaying exponentials. We further wish to use the infinity-norm in our approximation.

For demonstration of ideas, we use the hockey stick function, defined as $x(t) = 1-t$ for $0 \leq t \leq 1$, and $x(t) \equiv 0$ otherwise.

In principle, every linear combination of exponentials is the solution to some linear constant coefficient dynamic system. A fairly general form of the same may be expressed using a convolution integral as follows:

$$\dot{x}(t) = \int_0^t f(\tau)x(t-\tau)\,d\tau. \tag{1}$$

The first question to answer is, what choice of f will be consistent with the above x? If we ignore possible difficulties that might arise with highly irregular choices of f, we can discretize the above equation and find f using a system of simultaneous linear equations. Here is Matlab code that does it (Matlab has a convolution command, `conv`). It is assumed the input t points are equally spaced.

```
function f=findf(t,x)
t=t(:); x=x(:); dt=t(2)-t(1);
xd=[x(2:end)-x(1:end-1);0]/dt;
n=length(x);
A=zeros(n);
for k=1:n
    f=zeros(n,1); f(k)=1; c=conv(f,x); A(:,k)=c(1:n)*dt;
end
f=A\xd;
```

For the hockey stick function, using 2000 points uniformly spaced on [0,6], we obtain f using the above Matlab code. We do the same for 1000 points as well. Results are shown in figure 1.

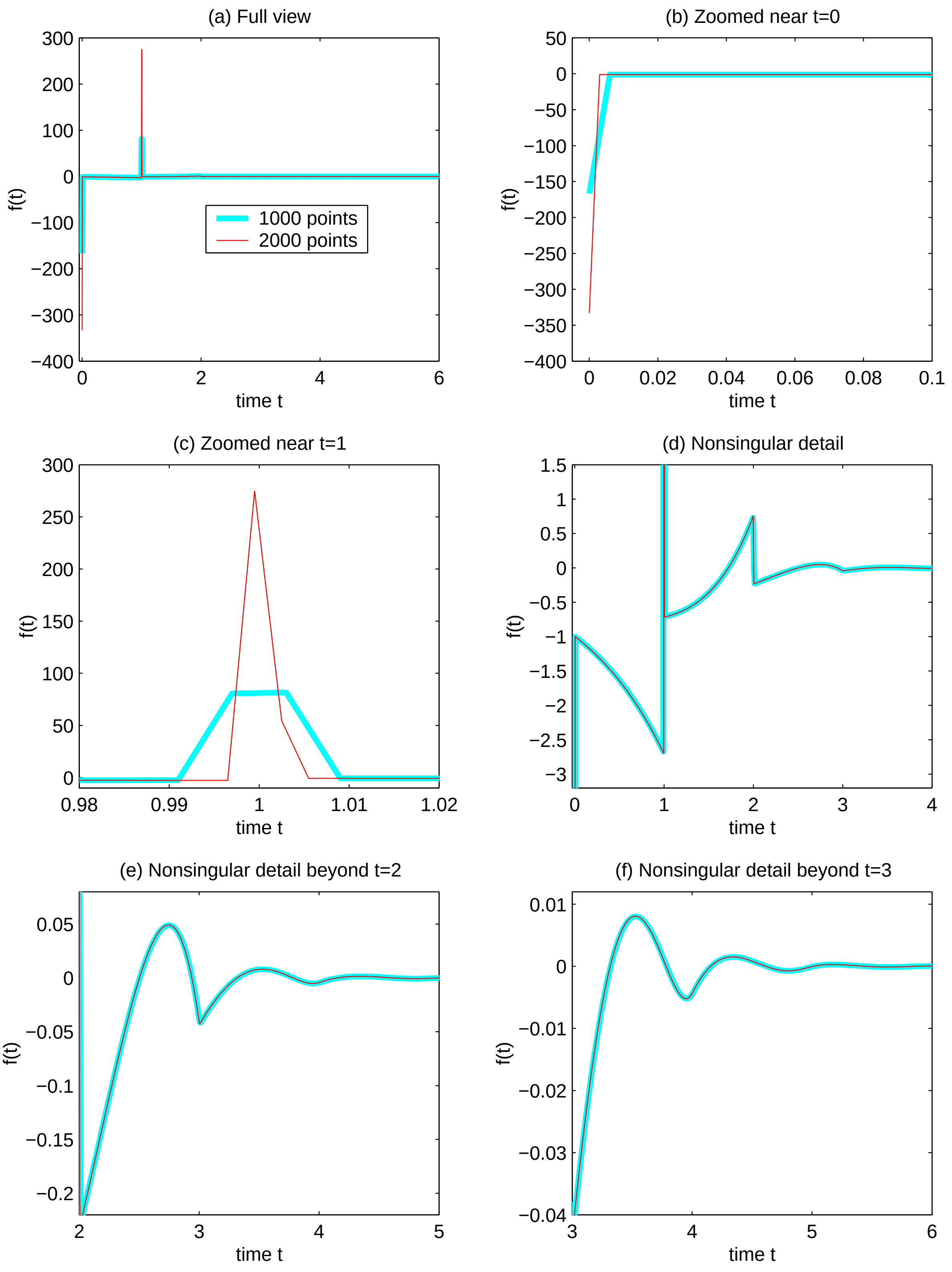

Figure 1: First view of $f(t)$.

2 Characterization of f

It is clear from figure 1 that there are Dirac delta functions at $t = 0$ and $t = 1$. Numerical estimates of their strengths suggest both are of unit magnitude. This unit magnitude can be verified analytically, but we accept the unit magnitude here and move on. For the benefit of the student, the Matlab code used to draw the figure is given in appendix A.

The rest of $f(t)$ can be well approximated by piecewise polynomials, on the intervals $(0,1)$, $(1,2)$, $(2,3)$ and so on. We choose to ignore the nonzero values of f beyond some large enough t. Six such polynomial plots (each of fifth order) are shown in figure 2 (thick cyan: actual value; thin blue: polynomial fit). The Matlab code for generating the plots is given in appendix B. Matlab

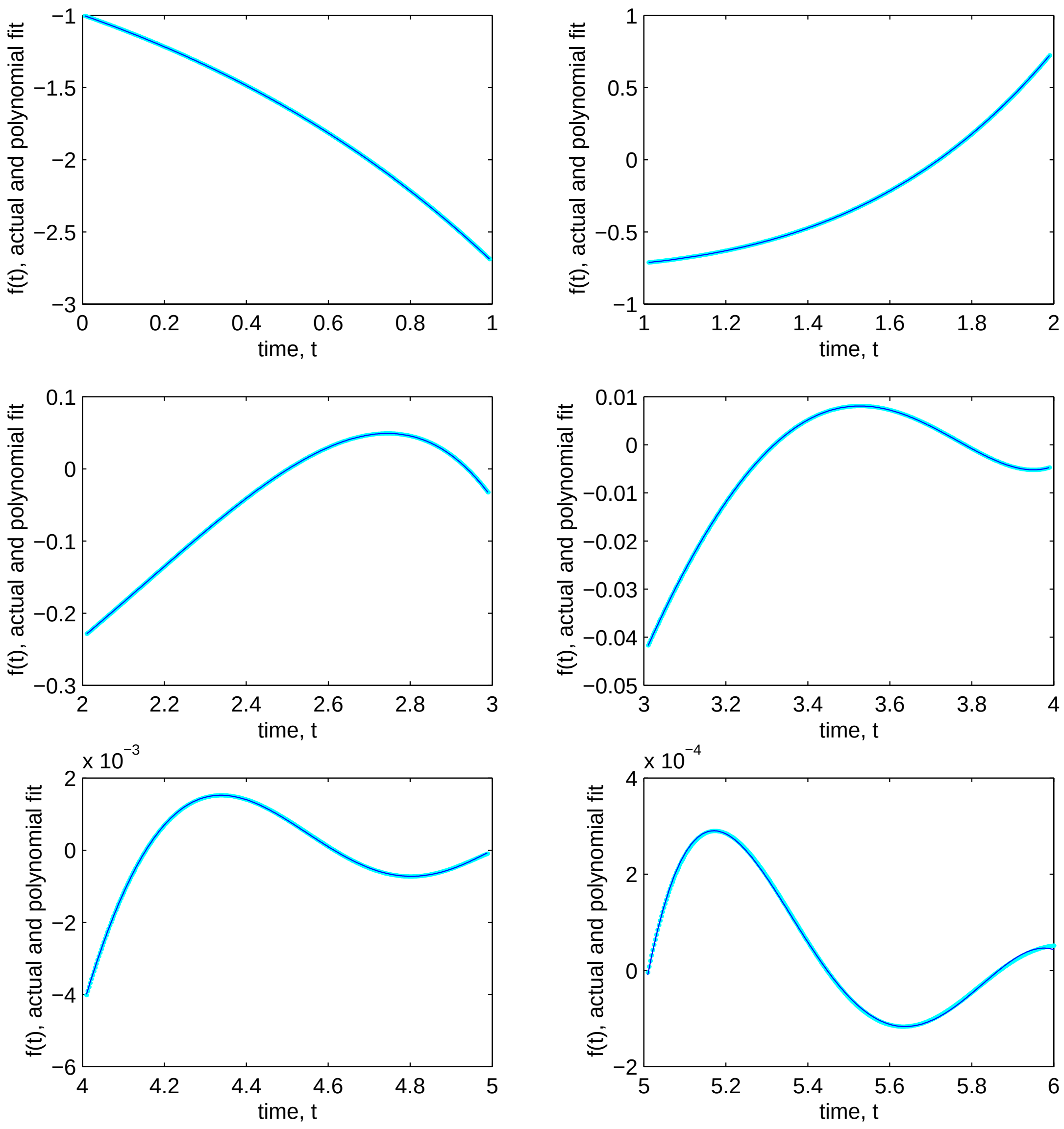

Figure 2: Piecewise polynomial approximations to $f(t)$.

complains of numerical ill-conditioing for the last fit, and suggests shifting the time variable before fitting, but I am going ahead with the fit obtained. The coefficients of the six polynomials fitted (in Matlab's sequence of highest order coefficient first, constant term last) are given here to 4 significant digits (though I used more digits in the actual calculations). Each row below contains the coefficients of one polynomial; row 1 for (0,1), row 2 for (1,2), and so on.

```
-0.0137   -0.0345   -0.1692   -0.4960   -0.9956   -0.9970
 0.0658   -0.2321    0.6696   -0.6434    0.1584   -0.7333
-0.1230    1.1804   -4.7502   10.1287  -10.9628    4.2296
 0.1091   -1.7158   10.7465  -33.6882   53.2785  -34.3780
-0.0412    0.9092   -7.9743   34.6856  -74.7490   63.7710
 0.0011   -0.0407    0.5696   -3.8204   12.4353  -15.8333
```

From the above, the polynomial valid over the interval (0,1) (let us call it $p_1(t)$) is

$$p_1(t) = -0.0137t^5 - 0.0345t^4 - 0.1692t^3 - 0.4960t^2 - 0.9956t - 0.9970.$$

Similarly for the other polynomials, here called $p_2(t)$, $p_3(t)$, etc.

We now have our delay differential equation, or approximation thereto.

3 Delay differential equation (DDE)

Our system is (the Dirac delta functions lead to discrete delayed feedback, while the rest of f leads to distributed delayed feedback through integrals)

$$\dot{x}(t) = -x(t) + x(t-1) + \int_0^1 p_1(\tau)x(t-\tau)d\tau + \int_1^2 p_2(\tau)x(t-\tau)d\tau + \cdots + \int_5^6 p_6(\tau)x(t-\tau)d\tau\,.$$

The characteristic roots of the above DDE give, in my opinion, a good choice of exponential rates to use for the problem of approximating the original function $x(t)$.

Moving on, we now insert $x(t) = e^{\lambda t}$ into the DDE, carry out the integrations, and obtain the chanracteristic equation. Maple is better at this than Matlab. The final, somewhat long, characteristic equation obtained is (I have not tried to clean it up, because this draft is just to explain the idea)

$$-4.7950537951662\,\frac{1}{\lambda^5 e^\lambda} - 5.0589548128294\,\frac{1}{\lambda^4 e^\lambda} - 3.9562950825100\,\frac{1}{\lambda^3 e^\lambda} - 2.9825867606398\,\frac{1}{\lambda^2 e^\lambda}$$

$$-1.9910129728492\,\frac{1}{\lambda\, e^\lambda} + 0.99700479432257\,\lambda^{-1} - 9.5375967400964\,\frac{1}{\lambda^6 e^\lambda} + 0.0030532696611102\,\frac{1}{\lambda\,(e^\lambda)^3}$$

$$+11.409743137697\,\frac{1}{\lambda^5\,(e^\lambda)^2} + 10.028571451764\,\frac{1}{\lambda^4\,(e^\lambda)^2} + 5.9001081828073\,\frac{1}{\lambda^3\,(e^\lambda)^2}$$

$$+2.9739253921687\,\frac{1}{\lambda^2\,(e^\lambda)^2} + 0.99099655847756\,\frac{1}{\lambda\,(e^\lambda)^2} + 22.654914390991\,\frac{1}{\lambda^6\,(e^\lambda)^2} - 1.0\,\left(e^\lambda\right)^{-1}$$

$$+0.99561680551259\,\lambda^{-2} + 0.99205746698768\,\lambda^{-3} + 1.0152807572428\,\lambda^{-4} + 0.82833187709857\,\lambda^{-5}$$

$$-27.856644059459\,\frac{1}{\lambda^6\,(e^\lambda)^3} - 9.8106774616557\,\frac{1}{\lambda^4\,(e^\lambda)^3} - 3.8760024845469\,\frac{1}{\lambda^3\,(e^\lambda)^3}$$

$$-0.98222980068417\,\frac{1}{\lambda^2\,(e^\lambda)^3} - 14.061606796021\,\frac{1}{\lambda^5\,(e^\lambda)^3} + 0.90950349926388\,\frac{1}{\lambda^3\,(e^\lambda)^4}$$

$$-0.0054627983340090\,\frac{1}{\lambda^2\left(e^\lambda\right)^4}-0.000077117586156959\,\frac{1}{\lambda\left(e^\lambda\right)^4}+9.1339075490278\,\frac{1}{\lambda^5\left(e^\lambda\right)^4}$$
$$+18.033507687771\,\frac{1}{\lambda^6\left(e^\lambda\right)^4}+4.5927323865473\,\frac{1}{\lambda^4\left(e^\lambda\right)^4}-2.5531390341680\,\frac{1}{\lambda^5\left(e^\lambda\right)^5}$$
$$-5.0702397360047\,\frac{1}{\lambda^6\left(e^\lambda\right)^5}-0.65140482215439\,\frac{1}{\lambda^4\left(e^\lambda\right)^5}+0.039156568431428\,\frac{1}{\lambda^3\left(e^\lambda\right)^5}$$
$$+0.0011317472998933\,\frac{1}{\lambda^2\left(e^\lambda\right)^5}+0.000047772601514467\,\frac{1}{\lambda\left(e^\lambda\right)^5}+1.6452759013776\,\lambda^{-6}$$
$$+0.000043472323367708\,\frac{1}{\lambda\left(e^\lambda\right)^6}-0.090417353781163\,\frac{1}{\lambda^4\left(e^\lambda\right)^6}-0.012672679705313\,\frac{1}{\lambda^3\left(e^\lambda\right)^6}$$
$$-0.00027515098653883\,\frac{1}{\lambda^2\left(e^\lambda\right)^6}-0.19235567137657\,\frac{1}{\lambda^5\left(e^\lambda\right)^6}+0.13078255542000\,\frac{1}{\lambda^6\left(e^\lambda\right)^6}+1+\lambda=0.$$

The above equation must be solved for λ. It is a transcendental equation with infinitely many roots. But the larger roots of such equations usually follow some discernible pattern, so numerically finding several of them is not really difficult. (In particular, eventually the real parts change slowly while the imaginary parts are incremented by near-constant amounts.)

The first 66 numerically determined roots are shown in figure 3.

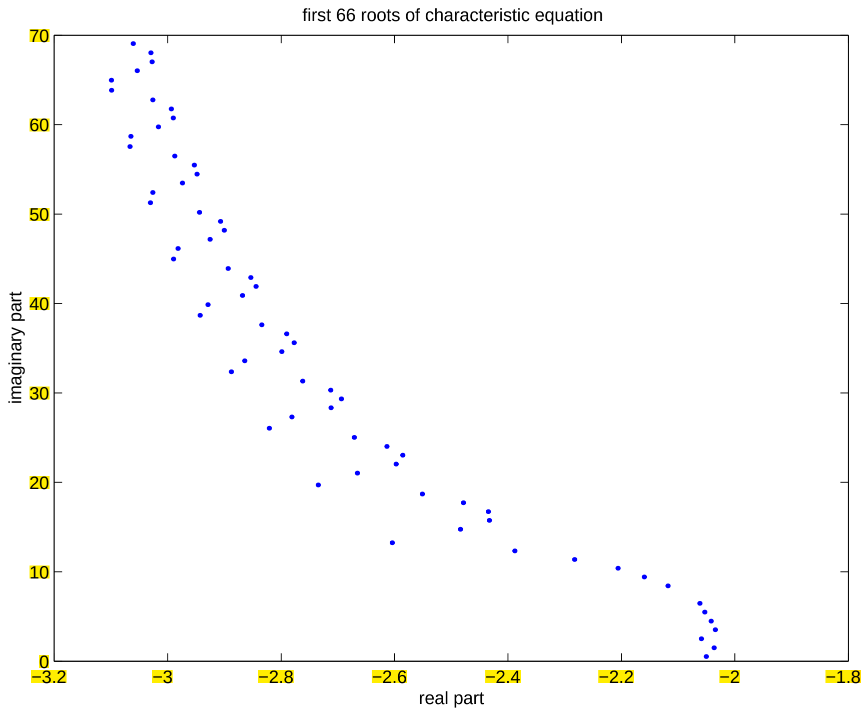

Figure 3: First several roots of the characteristic equation.

4 Least squares and least infinity-norm solution

If we have an overdetermined system $Ax = b$, then the least squares (or minimum-error in 2-norm) solution is easy, and given by Matlab in response to simply $A/\ b$. It is less easy, but still standard, to find the solution x that minimizes $\|Ax - b\|_\infty$. I claim no originality here at all, but here is a small Matlab program that uses Matlab's `linprog` to do this.

```
function x=inf_norm_sol(A,b)
% solves Ax=b approximately such that Ax-b is minimized in the
% infinite-norm as opposed to x=A\b, which minimizes Ax-b in the 2-norm.
% It is assumed that A has n rows and m columns, with n >= m,
% and that b has n rows and ONE column

[n,m]=size(A);
f=[1;zeros(n+m,1)];
% op=optimset('maxiter',2000);     % this is an optional command

AA=[zeros(n,1),-eye(n),A];
B=zeros(2*n+1,n+m+1);
B(:,1)=-ones(2*n+1,1);
B(2:n+1,2:n+1)=eye(n);
B(n+2:2*n+1,2:n+1)=-eye(n);
z=zeros(2*n+1,1);

% x=linprog(f,B,z,AA,b,[],[],[],op);   % use this, in place of the
                                       % following line, if options are set above
x=linprog(f,B,z,AA,b);
x=x(n+2:end);
```

Using the above infinity-norm minimizing solver, results obtained using 5 roots (i.e., 10 including complex conjugates) are plotted in figure 4. The results obtained using 8 roots are plotted in figure 5. However, attempts to use the above Matlab code with more roots leads to convergence problems or some similar problem that I do not understand. It seems to me that someone who understands linear programming (which I do not, really) might be able to use a better algorithm here than's Matlab's default.

A direct least-squares solution can also be used here, though it is somewhat unsatisfactory if the goal is to minimize the error in the infinity-norm. Results obtained with 62 roots are shown in figure 6.

5 Questions

There are many questions related to this approach, to which I do not know the answers.

For what class of functions x will this approach work? When will a somewhat regular solution for f exist? Can an analytical solution for f be found for the hockey stick function? What other forms can be used in place of Eq. 1, and what consequences does that hold for the roots of the characteristic equation? What is the clearest explanation for why the method works at all? How to get the infinity-norm solution robustly when many roots are used?

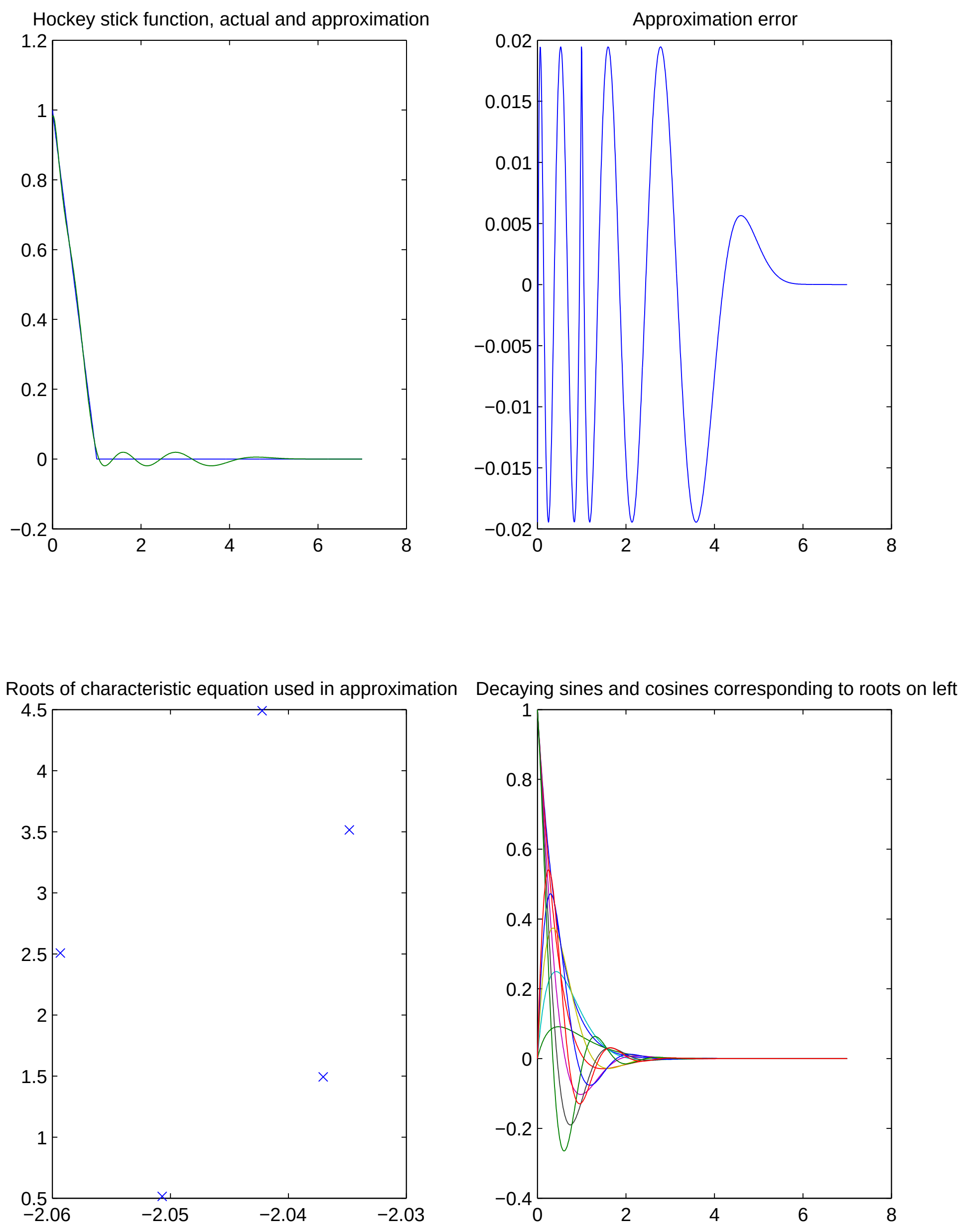

Figure 4: Approximation obtained using 5×2 roots.

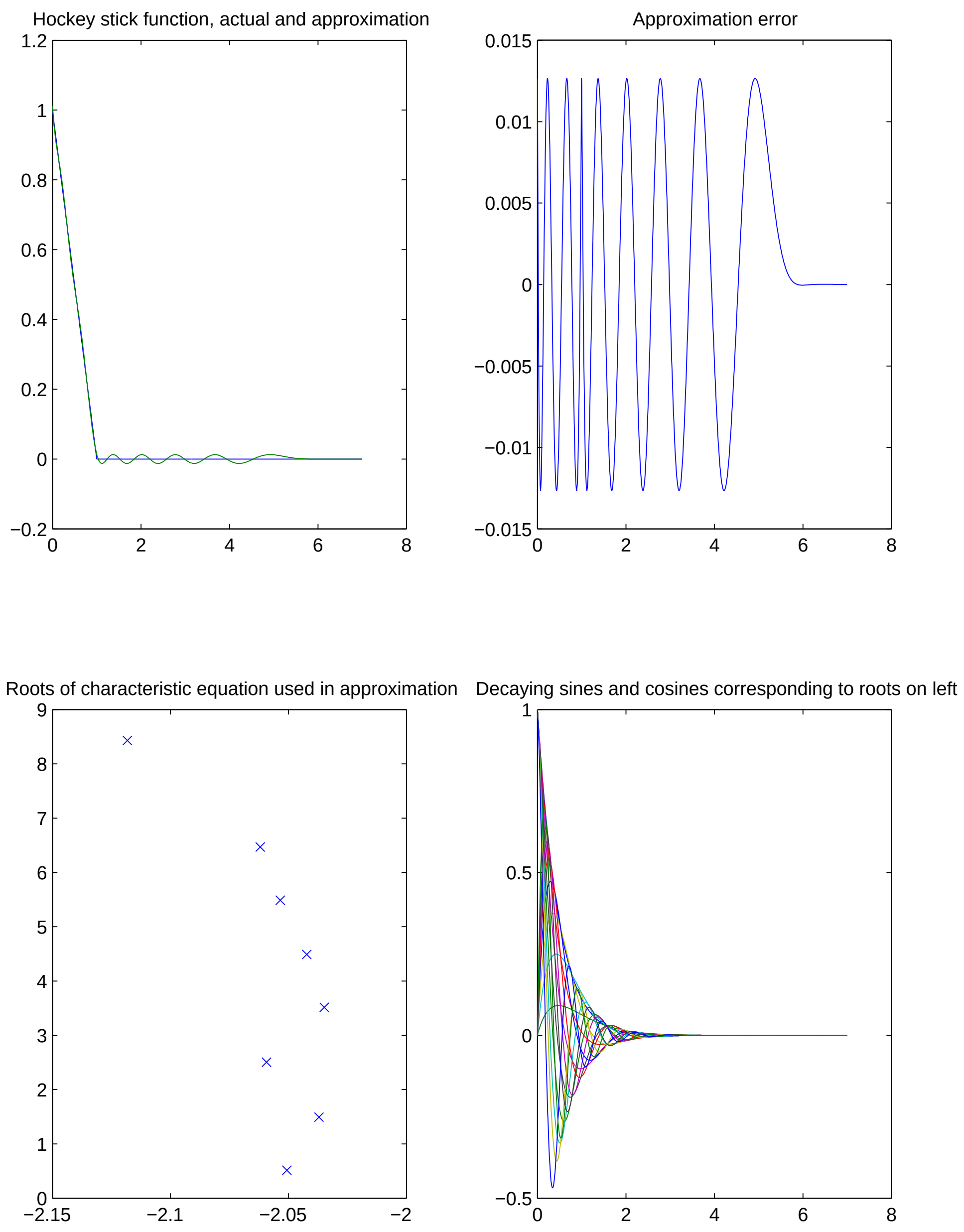

Figure 5: Approximation obtained using 8×2 roots.

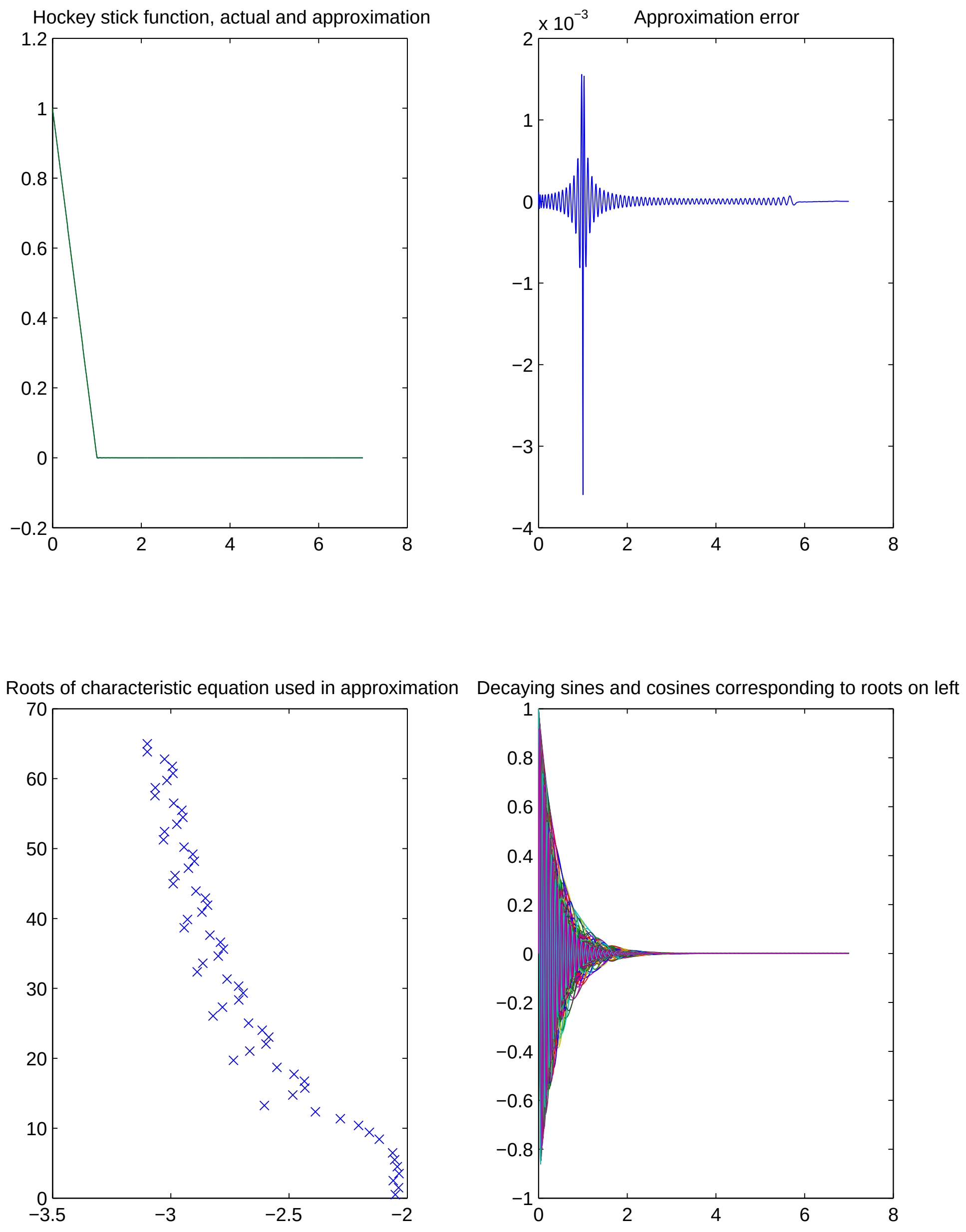

Figure 6: Approximation obtained using 62×2 roots and *least squares*.

I have some speculations about this method as well, related to things about DDEs that I believe but have nor formally studied. We can discuss these later.

A Matlab code for figure 1

```
close all; clear all
% t and f for 2000 and 1000 steps have been found and saved
% in 2 files called tf2000 and tf1000

load tf2000; t1=t; f1=f;
load tf1000
subplot(3,2,1)
plot(t,f,'c','linewidth',3); hold on; plot(t1,f1,'r');
legend('1000 points','2000 points')
axis([-.05,6,-400,300])
title('(a) Full view'), xlabel('time t'), ylabel('f(t)')

subplot(3,2,2)
plot(t,f,'c','linewidth',3); hold on; plot(t1,f1,'r');
axis([-.005,0.1,-400,50])
title('(b) Zoomed near t=0'), xlabel('time t'), ylabel('f(t)')

subplot(3,2,3)
plot(t,f,'c','linewidth',3); hold on; plot(t1,f1,'r');
axis([0.98,1.02,-10,300])
title('(c) Zoomed near t=1'), xlabel('time t'), ylabel('f(t)')

subplot(3,2,4)
plot(t,f,'c','linewidth',3); hold on; plot(t1,f1,'r');
axis([-0.03,4,-3.2,1.5])
title('(d) Nonsingular detail'), xlabel('time t'), ylabel('f(t)')

subplot(3,2,5)
plot(t,f,'c','linewidth',3); hold on; plot(t1,f1,'r');
axis([2,5,-.22,.08])
title('(e) Nonsingular detail beyond t=2'), xlabel('time t'), ylabel('f(t)')

subplot(3,2,6)
plot(t,f,'c','linewidth',3); hold on; plot(t1,f1,'r');
axis([3,6,-.04,.012])
title('(f) Nonsingular detail beyond t=3'), xlabel('time t'), ylabel('f(t)')
```

B Matlab code for generating figure 2

```
close all; clear all;
load tf2000
```

```
subplot(3,2,1)
n1=sum(t<0.995); n=3:n1;
t1=t(n); f1=f(n);
c1=polyfit(t1,f1,5);
plot(t1,f1,'c.',t1,polyval(c1,t1))
xlabel('time, t')
ylabel('f(t), actual and polynomial fit')

subplot(3,2,2)
n1=sum(t<1.01); n2=sum(t<1.99); n=n1+1:n2;
t1=t(n); f1=f(n);
c2=polyfit(t1,f1,5);
plot(t1,f1,'c.',t1,polyval(c2,t1))
xlabel('time, t')
ylabel('f(t), actual and polynomial fit')

subplot(3,2,3)
n1=sum(t<2.01); n2=sum(t<2.99); n=n1+1:n2;
t1=t(n); f1=f(n);
c3=polyfit(t1,f1,5);
plot(t1,f1,'c.',t1,polyval(c3,t1))
xlabel('time, t')
ylabel('f(t), actual and polynomial fit')

subplot(3,2,4)
n1=sum(t<3.01); n2=sum(t<3.99); n=n1+1:n2;
t1=t(n); f1=f(n);
c4=polyfit(t1,f1,5);
plot(t1,f1,'c.',t1,polyval(c4,t1))
xlabel('time, t')
ylabel('f(t), actual and polynomial fit')

subplot(3,2,5)
n1=sum(t<4.01); n2=sum(t<4.99); n=n1+1:n2;
t1=t(n); f1=f(n);
c5=polyfit(t1,f1,5);
plot(t1,f1,'c.',t1,polyval(c5,t1))
xlabel('time, t')
ylabel('f(t), actual and polynomial fit')

subplot(3,2,6)
n1=sum(t<5.01); n=n1:2000;
t1=t(n); f1=f(n);
c6=polyfit(t1,f1,5);
plot(t1,f1,'c.',t1,polyval(c6,t1))
xlabel('time, t')
ylabel('f(t), actual and polynomial fit')
```