Picard iteration convergence for solving non-linear state space system

Nasser M. Abbasi

January 7, 2015 Compiled on September 9, 2023 at 9:27pm

1 introduction

Picard iteration for the solution of non-linear system $x'(t) = f(x(t), t)$ is given by

$$x^{k+1}(t) = x^0 + \int_0^t f\left(x^k(\tau), \tau\right)\, d\tau$$

The above iteration was implemented numerically for a two state system with the forcing function $f = \begin{pmatrix} \cos x_1 \\ t x_1 + e^{-t} x_2 \end{pmatrix}$

The initial guess used is the same as the initial conditions which is given by $x^0 = \begin{pmatrix} 2 \\ -1 \end{pmatrix}$. Matlab was used for the implementation. The source code is in one m file and can be downloaded from the link above. A movie showing the convergence is given below as well.

2 Observation

Numerical solution was required since there is no closed form solution using symbolic integration after the second iteration for the first state. The Picard solution is compared to the final numerical solution obtained from ODE45. Time span of 30 seconds was used. For numerical integration, different sampling times were tried. The results shows that the smaller the time span, the less Picard iterations are needed to converge.

Reference: Lecture notes, ECE 717, Fall 2014, University of Wisconsin, Madison, by Professor B. Ross Barmish

2.1 Source code

```
function   nma_picard()
%version oct 17, 2014
%study of picard iteration method for non-linear state space system
%Matlab 2013a
%The following parameters can be changed: max simulation time,
%time spacing between each sample, initial conditions, and
%initial guess.
%by Nasser M. Abbasi

close all

number_of_iterations = 100;    %How many Picard iterations to do?
initial_conditions   = [2 -1]; %initial conditions for x_1 and x_2
initial_guess        = [2 -1]; %initial guess
max_time             = 50;     %simulation time in seconds
delT                 = 0.02;   %time spacing for sampling, numerical integration

nSamples             = round(max_time/delT);
first_K              = bsxfun(@times, initial_guess,ones(nSamples,2));
next_K               = zeros(nSamples,2);
t                    = (0:delT:max_time-delT)';

%obtain numerical ODE45 solution to use to compare with
odeTime                     = 0:delT:max_time-delT;
[odeTime,ODE45_solution]    = ode45(@rhs,odeTime,initial_conditions);

for n = 1:number_of_iterations
    for i = 1:nSamples %numerical integration as time is increased
                       %t vector above hold incremental time values.
        next_K(i,1) = initial_conditions(1) + delT*trapz(cos(first_K(1:i,1)));
        z = t(1:i).*first_K(1:i,1)+exp(-t(1:i)).*first_K(1:i,2);
        next_K(i,2) = initial_conditions(2) + delT*trapz(z);
    end

    makePlot(first_K,t,n,delT,initial_guess,max_time,ODE45_solution);
    first_K  = next_K;
end
    function dxdt=rhs(t,x)
        dxdt = [cos(x(1));t*x(1)+exp(-t)*x(2)];
    end
end

%-------------------------------
function makePlot(x,t,n,delT,guess,max_time,ODE45_solution)
if n==1
```

```
    scrsz = get(groot,'ScreenSize');
    figure('Position',[.25*scrsz(3) .35*scrsz(4) .5*scrsz(3) .5*scrsz(4)]);
    set(0,'DefaultAxesFontName', 'Times New Roman');
    set(0,'DefaultAxesFontSize',10);
    set(0,'DefaultTextFontname', 'Times New Roman');
    set(0,'DefaultTextFontSize', 12);
end

%fix the Y scale, to make the plots easier to see as simulation runs
minY1   = -1;
maxY1   = 2.5;
minY2   = -10;
maxY2   = 1000;

subplot(1,2,1);
hold off;
plot(t,x(:,1));
hold on;
plot(t,ODE45_solution(:,1),'r:');
title(sprintf('$$x_1^{%d}, \\Delta t=%3.3f, guess=[%d,%d]$$',n,delT,guess(1),guess(2)),'FontS
xlabel('time(sec)');
ylim([minY1,maxY1]);
xlim([0,max_time]);
subplot(1,2,2);
hold off;
plot(t,x(:,2));
hold on;
plot(t,ODE45_solution(:,2),'r:');
title(sprintf('$$x_2^{%d}$$',n),'FontSize', 14,'interpreter','latex');
xlabel('time(sec)');
ylim([minY2,maxY2]);
xlim([0,max_time]);
drawnow
end
```