**CMDA 3605 Jung H Choi**

# Term Project

SVD and Its Application to Dynamical Systems

## Abstract

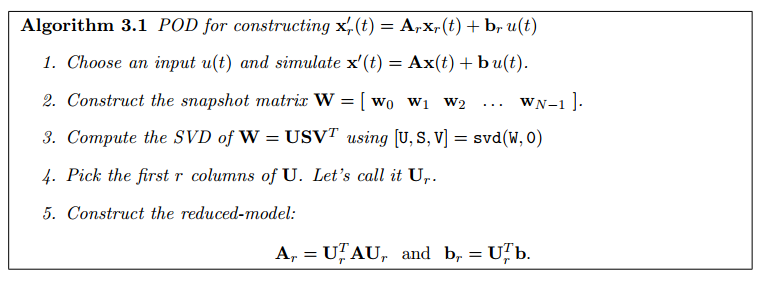
The goal of this project is to employ your Difference and Differential Equations, Linear Algebra and Matlab skills for applications in Dynamical Systems. At the end, these skills will be combined to construct dynamical systems directly from data (without access to original mathematical equations) and to reduce the number of equations in a dynamical system. The main tool in achieving this goal would be an important matrix decomposition, called the Singular Value Decomposition (SVD).

# Some instructions on the structure of the term project

Discussion below explains the required concepts in detail. Throughout the text, several problems are listed that you need to answer. However, your term project, once completed, should be a coherent report. Therefore do **NOT** simply answer the problems. Follow the pattern of the document below. You may use the same language or use other resources; but in both cases cite the appropriate source. Then, when you reach a certain problem, such as Problem 2.1, 2.2 etc, include its answer in the corresponding location and continue your discussion. Your criterion should be that your term project needs to be a coherent, well-written report; rather than a collection of solutions to specific problems. You are required to type your term project using your favorite editing software. Term projects that do not offer a coherent discussion (i.e. simply list solutions to the specific problems like a homework) and/or that are hand written will lose at least 20%. A printed hardcopy should be turned in, in addition to uploading it into Canvas. For the numerical problems, in addition to including your results and/or figures, attach your Matlab code as an Appendix. Also, as in the homework assignments, the code should also be uploaded into Canvas.

## 3.1 Proper Orthogonal Decomposition (POD) for Model Reduction

POD is combination of both Numerical simulation of dynamical systems and SVD. We will apply previous material from SVD to construct a reduced model.



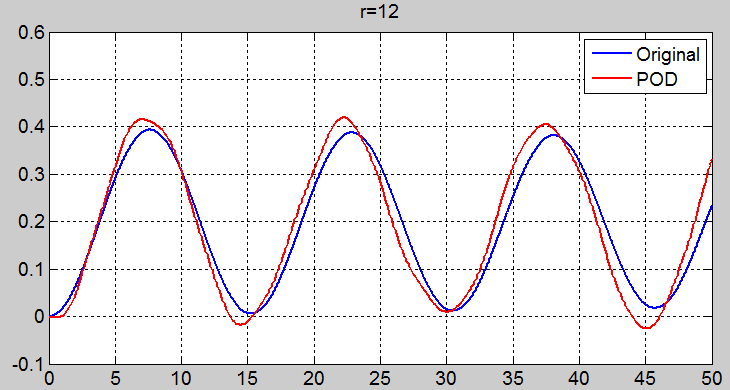
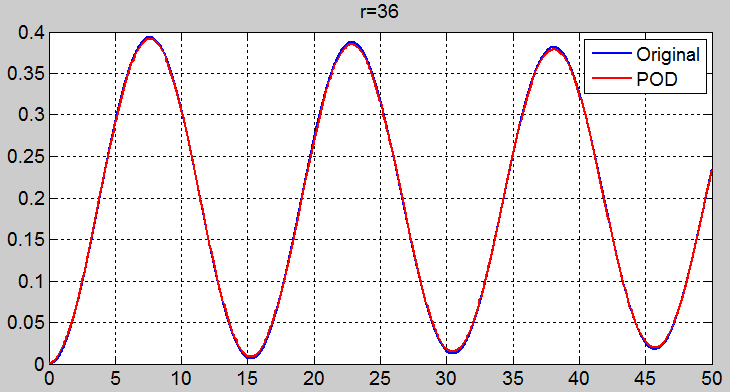
**Problem 3.1 Approximation of International Space Station 12A Module:**

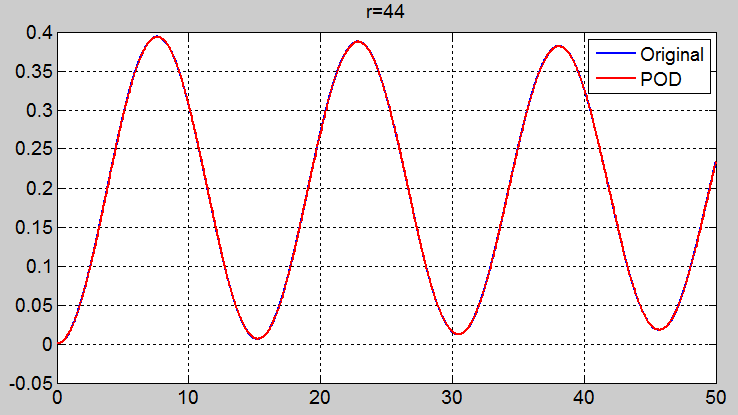
Using POD, we will approximate the behavior of ISS 12A module which contains **A** ∈ **R1412×1412****and b** ∈ **R1412**. We have constant input, u(t) =1 with x(0) =0. We will be using **ode45** to approximate the **x**’(*t*) = **Ax**(*t*) + **b***u*(*t*) with time samples *t = linspace(0,50,501).*

*Compute the SVD of W using [U,S,V] = svd(X, 0). We will apply the relative error from the SVD section.*

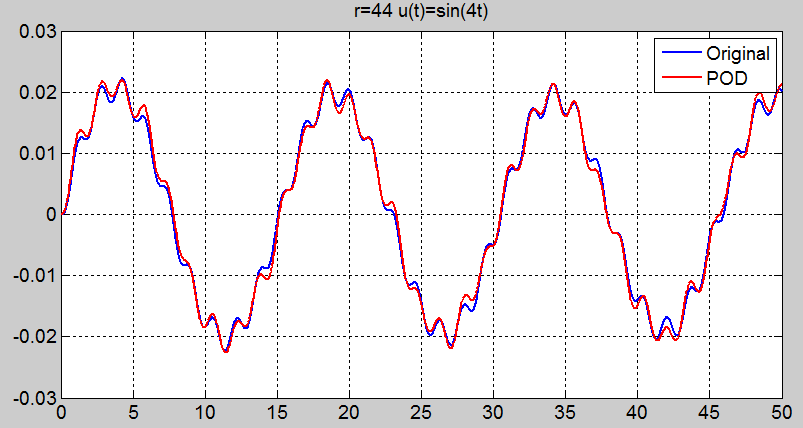


We will be using three relative errors: **10−1, 10−2** and **5 × 10−3**. The **r** values should be **12, 36**, and **44** respectively. Following graphs represent the POD approximations using the ode45 function.



# As you can see, the higher the r is, the more accurate the graphs are to the original function. This reduced model is very accurate.

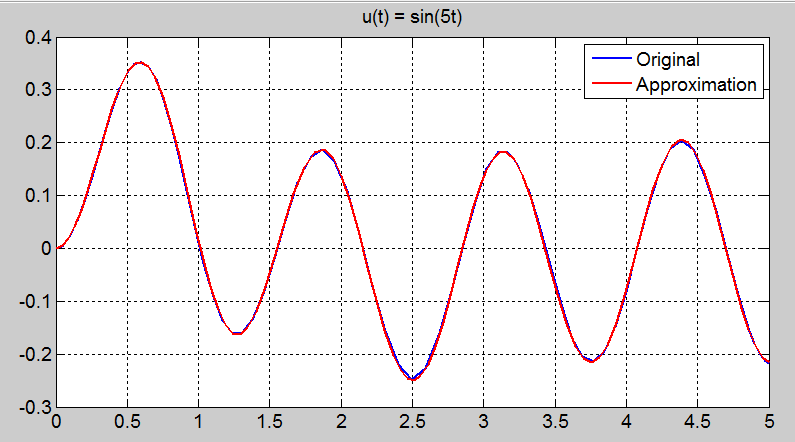


In this model, we have changed the input, **u(t) = sin(4t)** and kept **r** at **44**. There is a non-constant input to the model, but the result still shows very accurate approximation. Thus, this is the power of model reduction. Once we have a good reduced-model, we can use it over and over again for different inputs to approximate the original model.

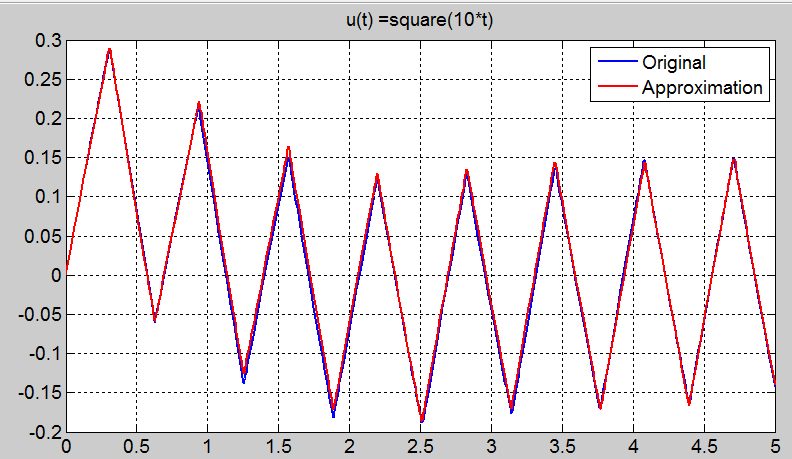
**Problem 3.2 Model Reduction of a Large-Scale Mass-Spring-Damper System:**

Now consider 10000 of such mass-spring-damper systems connected to each other. This results in a dynamical system of the form (3.2) with n = 20000. Hence **A** ∈ R 20000×20000 and **b** ∈ R 20000. Download the file **msd20000.mat** into your workspace. This file contains **A** and **b** for this model.[4]





For this plot, we choose **u(t) = sin(5t)** with error ratio less **5 x 10-3**. As you can see above, the result is even more accurate compare to the previous problem with **u(t)=sin(4t)**.



Same goes for this case, the original and the reduced-model graphs are very accurate and identical to each other. This is the power of model reduction. We can say that the POD-based model reduction provides well behavior of the Large-Scale Mass-Spring-Damper System.

# EC Implicit Solvers for Differential Equations

Implicit solver is one of the techniques we learned in class for dynamical systems of differential equations.

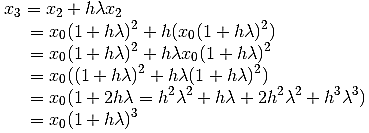
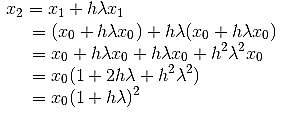
**x**’(*t*) = **A***x*(*t*) + **b***u*(*t*)*, x*(*t*0) = *x*0 (1)

We will be using Euler’s method to solver for the differential equation in this section. Euler’s method from above equation proceeds as

*wk*+1 = *wk* + *h*(*Awk* + *buk*)*, for k* = 0*,*1*,*2*,...* (2)

***Problem 4.1*** *Show that, if we apply the Eulers Method to (4.3), the resultingiterate wk satisfies*

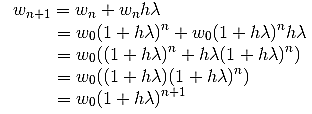
 *wk* = (1 + *hλ*)*kw*0*, for k* = 1*,*2*,...* (3)



Iterating through a scaler equation on the above with **b** = 0, **A**= λ (Since *n* = 1, **A** is a 1 × 1 matrix), we can see the **xk** comes out to be **xk= x0(1 + hλ)k** or

(4)

Assume that the equation for xn is **wn= w0(1 + hλ)n** and its true, then we can also assume for **wn+1**

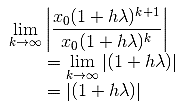


We can say the equation 4 is indeed for all k>= 1.

|  |  |  |
| --- | --- | --- |
| *Moreover, verify that* |  |  |
| *wk* → 0 if and only if | 0 *< h* | *λ* |*<* 2 | (5) |

*Explain what happens to wk as k* → ∞ *if h* | *λ* |*>* 2

The exact solution **x(t) = x0eλt** decays to zero as **t** goes to infinity and **λ<0**. As in the scalar case, all the eigenvalues will have negative real number**, x(t)** goes to 0 as **t** goes to infinity. We can apply Euler’s method to ensure that the numerical solution we have proved decays to zero.



So we have **xk** goes to **0** as **k** goes to infinity then,

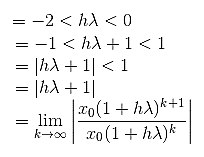
**|1+h λ |<1**

**-1|<1+h λ |<1**

**-2|<h λ|<0**

We need to ensure that the step sizeh satisfies

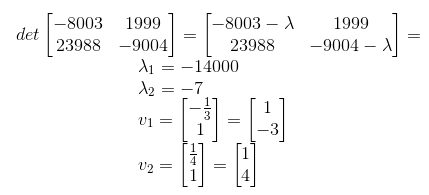
0 *< h* | *λi* |*<* 2 for *i* = 1*,*2*,...,n.* (6)



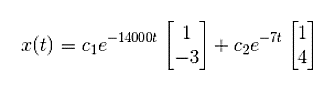
Thus, this solution decays to zero as **t** goes to infinity and **λ<0** which it converges. This condition puts a huge restriction on **h**. With Euler’s method applied to the original function makes it more desirable. If **h| λ |>2,** then the solution would diverge which would make Euler’s method no desirable.

**Problem 4.2** *(Example from Suli and Mayers [8]) Consider the system of differential equations*

 (7)

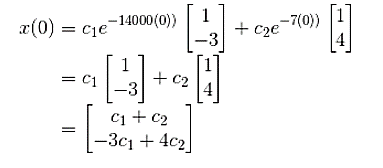


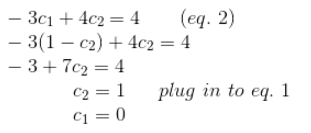
We have the general solution for **A**

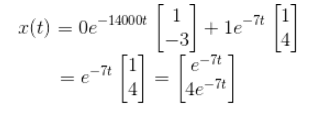


Applying the initial condition give for



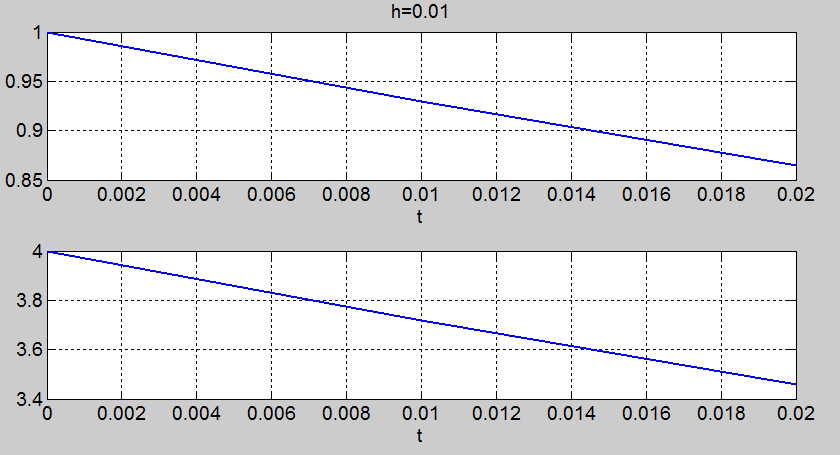


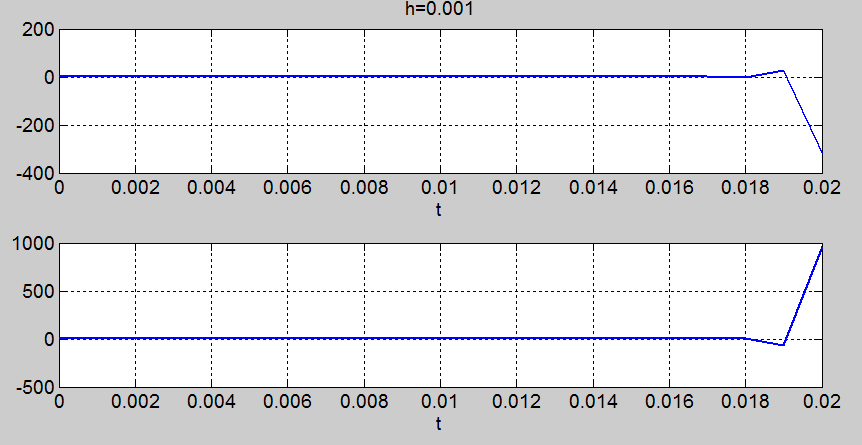


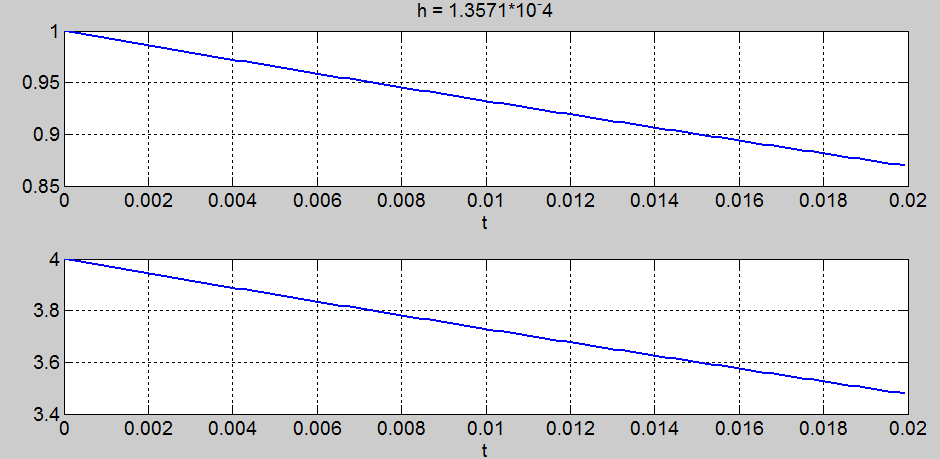


We verified that the exact solution of this initial value problem is **x(t) = [e-7t, 4e-7t]T**. Since both elements are in negative, both will approach **zero** as **t** goes to infinity.

Let’s apply Euler’s method to this problem with h=0.001 for the interval *t* ∈ [0*,*0*.*02]







As **t** gets bigger for h=0.001, x is decaying since it goes to [-300, 1000]T . We already know that this would decay because we proved in the last problem that the **h| λ |>2** would diverge. **0.001| -14000 |>2 => 14>2.** However, if we use **h = 1.3571 × 10−4** , then **h| λ |<2 => (1.3571 × 10−4| -14000 | = 1.899 < 2 and 1.3571 × 10−4| -7 | =0.00094 <2).** As you can see on the above, both graphs converges to 0 which shows better approximation.

## 4.1 Trapezoidal Rule Method

Euler’s method for the formula (1) proceeds as

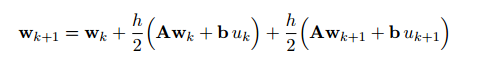


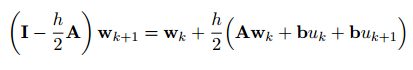


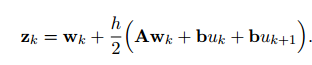
Trapezoidal Rule Method:



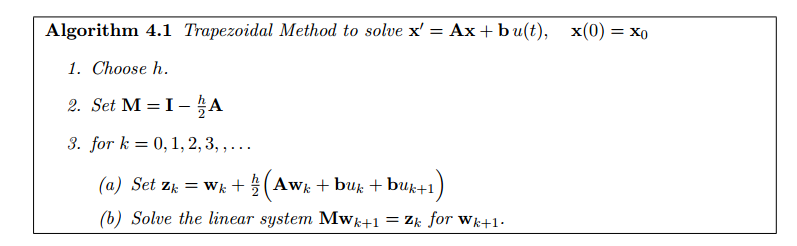
F(x,t) = Ax + bu(t). Plug it in to Trapezoidal Rule Method

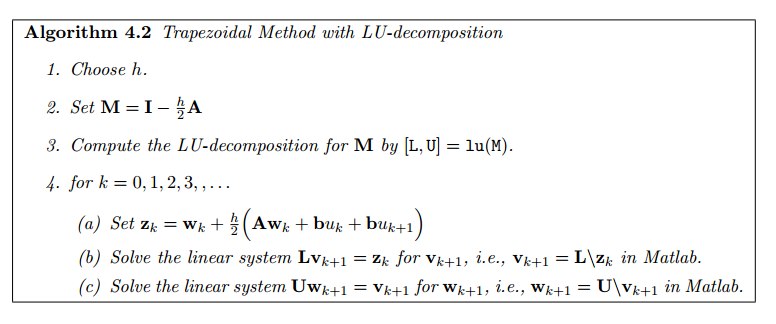




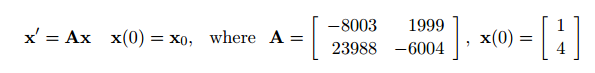
We know everything on the right hand side so we write it as



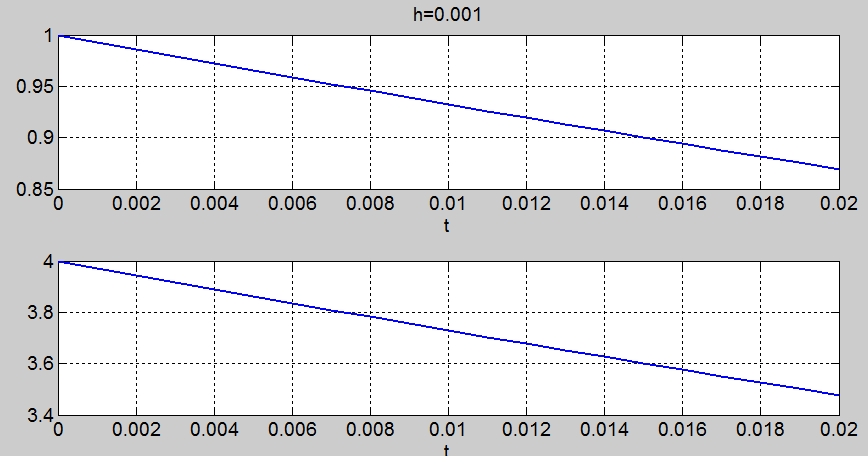


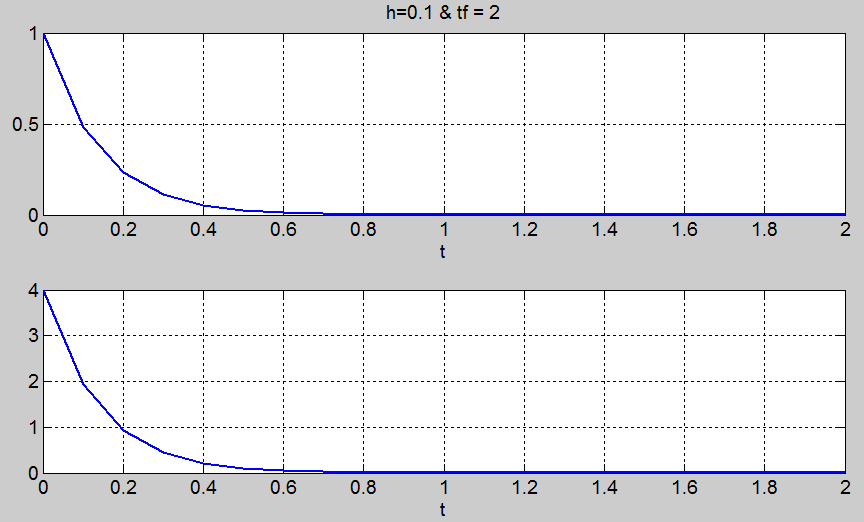


**Problem 4.3** *Write a Matlab code to implement Algorithm 4.2 and solve the differential equation in Problem 4.2 using the Trapezoidal Rule for the case h* = 0*.*001*.*



Check the Matlab code in the Appendix. We will approximate the differential equation in problem 4.2 using the Trapezoidal Method for h =0.001 and t ∈ [0, 0.02]





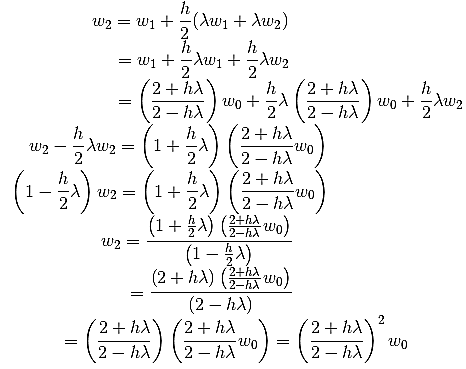
Trapezoidal Rule for the case h=0.001 is indeed very close to the original solution, however it still diverges for h=0.001.

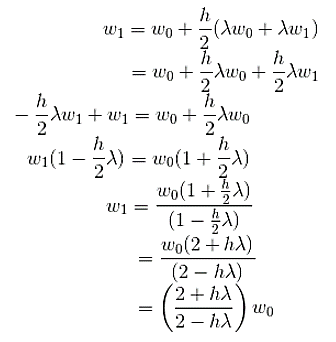
Using a bigger **h** and simulating the differential equation for **t** ∈  **[0, 2]** gives much better output. As you can see above, Trapezoidal Method with a larger step size **converges**. Euler’s Method has diverged for h=0.001, but here it’s much better than Euler’s method which shows accurate result for larger intervals.

**Problem 4.4** *Once again consider the scalar equation*

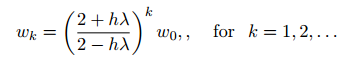
Apply the Trapezoidal Method to the scalar equation on the above.



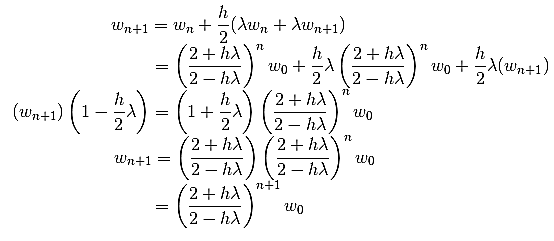


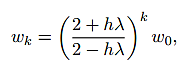


Just like from the problem 4.1, the result seems to have a pattern that leads to



We can check the iteration above to check for the base case, **w1**. Now, to verify that the **wk** → **0** for any **h>0**, we can check to see if **wn+1**holds the case.

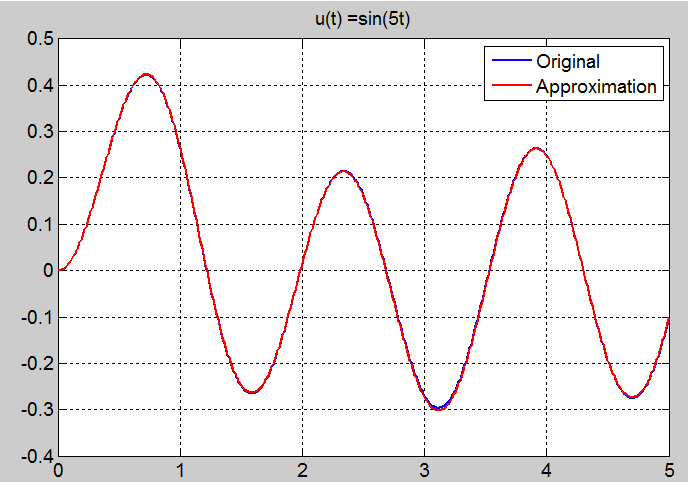




We have shown the equation indeed hold the case for wn+1 as well as

for all **k >0**. Since **λ < 0** and **wk → 0** for any **h > 0**, the fraction results to be less than **1**. Thus, any number of the fraction will result **0** as **k → ∞** which proves the convergence for the **Trapezoidal Method**.

**Problem 4.5 Simulating the Large-Scale Mass-Spring-Damper System via Trapezoidal Method:** *Let’s revisit the large-scale Mass-Spring-Damper System of Problem 3.2. Use the input* ***u*(*t*) = sin(5*t*)*, h* = 0*.*01** *and simulate* **x**’ = **Ax** + **b***u*(*t*) *with* **x**(0) = **0** *using the Trapezoidal Rule Method for t* ∈ [0*,*5]*. Then, plot the evolution of the second component of* **x(*t*)***.*



This is the Mass-Spring Damper System model from problem 3.2 with Trapezoidal Method applied instead of ode45 function. The approximation compare to the original graph is nearly identical and almost indistinguishable. Trapezoidal Method provides much better accuracy compare to the ode45 function.

**Appendix POD 1**

clear all

clf

set(0, 'defaultaxesfontsize',14,'defaultaxeslinewidth',1.0,...

'defaultlinelinewidth',2.0,'defaultpatchlinewidth',1.0,...

'defaulttextfontsize',18,'DefaultLineMarkerSize',14)

load 'iss12a'

t0 = 0; %given

tf = 50; %given

t = linspace(0, 50, 501);

x0 = zeros(length(A),1);

x= zeros(length(x0),length(t));

x(:,1) = x0;

f = @(t,x) A\*x+b\*(sin(4\*t)); %u(t) =1 or u(t) =1 or sin(4\*t)

[t,X] = ode45(f,[t0 tf], x0);

X = X';

[U S V] = svd(X,0);

plot(t, X(1,:),'b')

title('r=44')

hold on

i = 1;

err = 5\*10^-3;

while S(i,i)/S(1,1) > err

r=i;

i = i+1;

end

%r =12 36 and 44

Ur = U(:,1:r); %1412xr

Ar = Ur'\*A\*Ur; %rxr

br = Ur'\*b; %rx1

% Xr = x(1:r,1); %r x1

x0 = zeros(length(Ar),1);

x= zeros(length(x0),length(t));

x(:,1) = x0;

f2 = @(t,x) Ar\*x+br\*(sin(4\*t)); % u(t) =1 or u(t) =1 or sin(4\*t)

[t,Xhat] = ode45(f2,[t0,tf], x0);

Xhat = Xhat';

Xhat = Ur\*Xhat;

plot(t, Xhat(1,:),'r')

grid on

**Appendix POD 2**

clear all

clf

set(0, 'defaultaxesfontsize',14,'defaultaxeslinewidth',1.0,...

'defaultlinelinewidth',2.0,'defaultpatchlinewidth',1.0,...

'defaulttextfontsize',18,'DefaultLineMarkerSize',14)

load 'msd20000'

t0 = 0; %given

tf = 5; %given

t = linspace(0, 5, 501);

x0 = zeros(length(A),1);

x= zeros(length(x0),length(t));

x(:,1) = x0;

f = @(t,x) A\*x+b\*square(10\*t);

[t,X] = ode45(f,[t0 tf], x0);

X = X';

[U S V] = svd(X,0);

plot(t, X(2,:),'b')

hold on

% Approximate----------------------

i = 1;

err = 5\*10^-3;

while S(i,i)/S(1,1) > err

r=i;

i = i+1;

end

%r =12 36 and 44

Ur = U(:,1:r); %1412xr

Ar = Ur'\*A\*Ur; %rxr

br = Ur'\*b; %rx1

% ---------------------------------

%

%

% x0 = zeros(length(A),1);

% x= zeros(length(x0),length(t));

% x(:,1) = x0;

%

% f1 = @(t,x) A\*x+b\*square(10\*t);

% [t,Xhat0] = ode45(f1,[t0 tf], x0);

%

% Xhat0 = Xhat0';

% [U S V] = svd(X,0);

% plot(t, Xhat0(2,:),'b')

% hold on

x0 = zeros(length(Ar),1);

x= zeros(length(x0),length(t));

x(:,1) = x0;

f2 = @(t,x) Ar\*x+br\*square(10\*t);

[t,Xhat1] = ode45(f2,[t0,tf], x0);

Xhat1 = Xhat1';

Xhat1 = Ur\*Xhat1;

plot(t, Xhat1(2,:),'r')

title('u(t) =square(10\*t)')

legend('Original','Approximation')

grid on

**Appendix Euler**

clear all;

f = @(t,x) [-8003 1999; 23988 -6004]\*[x(1);x(2)];

A= [-8003 1999; 23988 -6004];

% h=0.1;

h=0.001;

% h =1.3571\*10^-4;

t0 =0;

tf =0.02;

% tf =2;

x0 = [1; 4];

% [t,x] = euler(t0,tf,h,f,x0);

[t,x] = Trapezoidal(t0,tf,h,A,x0,0,0);

subplot(2,1,1);

plot(t,x(1,:));

title('h=0.001')

xlabel('t') % x-axis label

grid on

subplot(2,1,2);

plot(t,x(2,:));

xlabel('t') % x-axis label

grid on

function [t,x] = euler(t0,tf,h,f,x0)

% t0 = initial time

% tf = final time

% h = step size

% f = the function handle defning the nonlinearity of your problem

% x0 = initial condition

% t = vector of time steps, i.e, t = [t0; t1; : : : ; tf]

% x = stores the approximate numerical solution; i.e, the kth column of x is xk.

% h = (tf-t0)/10

t= t0:h:tf;

x= zeros(length(x0),length(t));

x(:,1)=x0;

for i=1:length(t)-1

x(:,i+1)= x(:,i)+h\*f(t(i),x(:,i));

end

# Appendix Trapezoidal 1

function [t,x] =Trapezoidal(t0,tf,h,A,x,b,uk)

% u=0

% b=0;

b=0;

u= @(t) 0;

% u = @(t) uk % When u=0

% u = @(t) uk(t);

[m,n] = size(x);

if m==1 && n==1 && x==0

x0 = zeros(length(A),1);

elseif m~=1 && n==1

x0 = x;

elseif m==1 && n~=1

x0 = x';

else

error('error');

end

M = speye(length(A)) -h/2\*(A);

[L,U] = lu(M);

t = t0:h:tf;

x= zeros(length(x0),length(t));

x(:,1) = x0;

for i =1:length(t)-1

z(:,i)=x(:,i) + h/2.\*(A\*x(:,i)+b.\*u(t(i))+b.\*u(t(i+1)));

v(:,i+1) = L\z(:,i);

x(:,i+1) = U\v(:,i+1);

end

end

# Appendix Trapezoidal 2

function [t,x] =Trapezoidal(t0,tf,h,A,x,b,uk)

% u=0

% b=0;

% When u=0

% u = @(t) uk

u = @(t) uk(t);

[m,n] = size(x);

x0 = zeros(length(A),1);

M = speye(length(A)) -h/2\*(A);

[L,U] = lu(M);

t = t0:h:tf;

x= zeros(length(x0),length(t));

x(:,1) = x0;

for i =1:length(t)-1

z(:,i)=x(:,i) + h/2.\*(A\*x(:,i)+b.\*u(t(i))+b.\*u(t(i+1)));

v(:,i+1) = L\z(:,i);

x(:,i+1) = U\v(:,i+1);

end

end

# Appendix EC Trapezoidal

clear all

clf

set(0, 'defaultaxesfontsize',14,'defaultaxeslinewidth',1.0,...

'defaultlinelinewidth',2.0,'defaultpatchlinewidth',1.0,...

'defaulttextfontsize',18,'DefaultLineMarkerSize',14)

load 'msd20000'

t0 = 0; %given

tf = 5; %given

h = 0.01; %given

u =@(t) sin(4\*t);

[t,X] = Trapezoidal(t0,tf,h,A,0,b,u);

[U S V] = svd(X,0);

% Approximate----------------------

i = 1;

err = 5\*10^-3;

while S(i,i) / S(1,1) > err

r = i;

i = i+1;

end

%-----------------------------------

%r =12 36 and 44

Ur = U(:,1:r); %1412xr

Ar = Ur'\*A\*Ur; %rxr

br = Ur'\*b; %rx1

% ---------------------------------

% plot the evolution of the second component of x(t).

[t,Xhat] = Trapezoidal(t0,tf,h,Ar,0,br,u);

Xhat = Ur\*Xhat;

plot(t, X(2,:),'b')

hold on

plot(t, Xhat(2,:),'r')

title('u(t) =sin(5t)')

legend('Original','Approximation')

grid on

**References**

[1] G. Golub and C. van Loan, *Matrix Comptutations*, 3rd Edition, JHU Press, 2012.

[2] N. Muller, L. Magaia, and B.M. Herbst, *Singular value decomposition*, *eigen- faces, and 3D reconstructions*, SIAM review, Vol. 46, No. 3, pp: 518{545, 2004. http://mathworld.wolfram.com/SingularValueDecomposition.html

[3] wolfram, SingularValueDecomposition.html mathworld.wolfram.com/SingularValueDecomposition.html.

[4] S. Gugercin, *Term Project SVD and Its Application to Dynamical Systems*, CMDA 3605, 2017.