	NTNU Norges teknisk-naturvitenskapelige universitet Fakultet for ingeniørvitenskap Institutt for marin teknikk	**EXERCISE 3** **GRADED** Chp. 2.3, 3.1, 3.2, 3.5

TMR4182 MARINE DYNAMICS

NUMERICAL LAB: TIME INTEGRATION OF DYNAMIC RESPONSE

The purpose of this exercise is to carry out numerical integration of the responses of a single degree-of-freedom system and to understand the effects of different inputs. This exercise shall be carried out in **groups** of (maximum) 4 students. One set of codes and one report should be submitted by each group.

In this exercise, we consider the mass-spring-dashpot system in Fig. with different applied loads, $P(t)$.

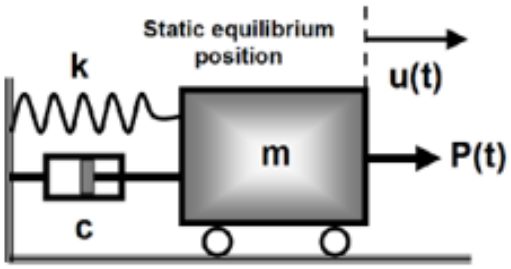

Figure 1: SDOF system

1 Implementation (30%)

In either Matlab or Python, the group should implement three generic approaches to estimating the response u to a load $P(t)$. One approach is the constant average acceleration method. The second approach is the 4th order Runge-Kutta algorithm. The third is the discrete time solution using the state-space form with piecewise constant input.

You should **submit your code** for the functions which carry out the integration. These functions should be in the form:

- Matlab:

```
u = const_avg_acc(m,k,c,P,h,u0,udot0)
```

```
u = runge_kutta4(m,k,c,P,h,u0,udot0)
```

```
u = dis_state_space(m,k,c,P,h,u0,udot0)
```

- Python:

```
def const_avg_acc(m,k,c,P,h,u0,udot0):
    # your code to define u here
    return u
```

```
def runge_kutta4(m,k,c,P,h,u0,udot0):
    # your code to define u here
    return u
```

```
def dis_state_space(m,k,c,P,h,u0,udot0):
    # your code to define u here
    return u
```

Furthermore, each function should each be saved in a separate file with the appropriate naming convention: const_avg_acc.m, runge_kutta4.m, and dis_state_space.m, or const_avg_acc.py, dis_state_space.py, and runge_kutta4.py. Submit all three scripts as a single zipped file in Blackboard. This makes it easier for us to carry out automated testing of the code. This is also good practice for ensuring that your code does not rely on any unpassed variables from the calling code.

The inputs to these functions are:

variable	description	units	size
m	mass of the SDOF system	kg	1x1
k	stiffness coefficient of the SDOF system	N/m	1x1
c	damping coefficient of the SDOF system	Ns/m	1x1
P	load time series with time step h	N	nx1
h	time step for the load time series P	s	1x1
u0	initial position of the SDOF system	m	1x1
udot0	initial velocity of the SDOF system	m/s	1x1

The output from each function is a vector of the position of the system:

variable	description	units	size
u	displacement	m	nx1

The time vector for the response should be identical to the time vector of the load.

An example of the calling code for these codes is given at the end of the assignment. **Your code will be tested in an automated manner using m, c, k, P, initial conditions, and time duration/step which differ from the calling code at the end of the assignment. The file name, function name, input and output must therefore be identical to the specification above.** As you are soon-to-be professionals, I expect you to write code in a professional manner (use comments as needed, good variable names, break lines as needed). Someone else should be able to read and understand your work!

2 Application (70%)

We will investigate the use of numerical tools for different types of loads (Fig. 2). Your response to this part of the assignment should be in the form of a short report on your investigations. You should carry out **any two investigations of your choosing** out of the options labelled below (A, B, C,...).

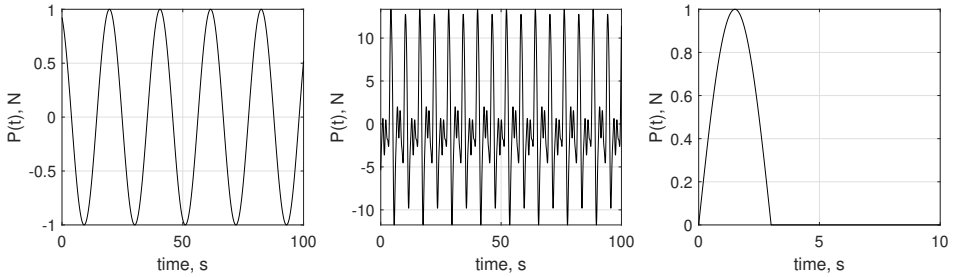

Figure 2: Load types. From left to right: harmonic load, general periodic load, and short impulsive load.

Figures and discussion are expected - and all graphs should have axis labels and correct units. Inputs (such as mass, stiffness, damping, etc) should be clearly documented for all results that are shown.

(A) **Simple harmonic load, analytical results and damping**

The load takes the form of a simple harmonic load, for example:

$$P(t) = P_0 \cos(\omega t + \epsilon). \tag{1}$$

Required investigations:

- Compare your results to the analytical solution for an underdamped (subcritically damped) system with $P = P_0 \cos(\omega t + \epsilon)$:

$$u(t) = \underbrace{e^{-\zeta\omega_0 t} R \cos(\omega_d t - \theta)}_{\text{homogeneous (transient) solution}} + \underbrace{U \cos(\omega t + \epsilon + \phi)}_{\text{particular (steady-state) solution}} \tag{2}$$

$$U = \frac{P_0}{k} \frac{1}{\sqrt{(1-\beta^2)^2 + (2\zeta\beta)^2}} \tag{3}$$

$$\phi = \tan^{-1}\left(\frac{-2\zeta\beta}{1-\beta^2}\right) \tag{4}$$

$$\theta = \tan^{-1}\left(\frac{\dot{u}_0 + \zeta\omega_0 (u_0 - U\cos(\epsilon+\phi)) + \omega U \sin(\epsilon+\phi)}{\omega_d (u_0 - U\cos(\epsilon+\phi))}\right) \tag{5}$$

$$R = \frac{u_0 - U\cos(\epsilon+\phi)}{\cos\theta} \tag{6}$$

(where we use the usual definitions for the damping ratio ζ and frequency ratio β).

- Determine a criterion to identify the duration of the transient solution. Examine the effect of the critical damping ratio on 1) the duration of the transient and 2) the maximum steady-state response. Your investigation should consider at least three sets of inputs so that you can see if the conclusions differ for:
  - a system at resonance
  - a system experiencing stiffness-dominated response
  - a system experiencing inertia-dominated response

(B) **Simple harmonic load, physics-based vs. machine learning models**

Machine learning models are a popular method for trying to use extensive measurements to make predictions. In general, these types of models are used when modelling the physics directly is infeasible (either due to a lack of information about the system itself, or a lack of knowledge of the important physical phenomena). Clearly, for a simple spring-mass-dashpot system, we don't actually need a machine learning model. But, it can still be fun to see how such a model performs.

In this investigation, you will generate a set of training "measurements" by running one of your numerical simulation tools with fixed inputs m, c, k, $u(0) = 0$ and $\dot{u} = 0$ and variable simple harmonic inputs $P_0 \cos(\omega t + \alpha)$. Then, you should train a recurrent neural network (RNN) or other model of your choice to predict the system's response to an new simple harmonic load over time. In addition to initial conditions at rest, the fixed inputs are given as in the table below.

m	birth month of group member 1 (range 1-12)	kg
k	date in the month of birthday of group member 2 (range 1-31)	N/m
c	selected randomly so that $0.01 \leq \zeta \leq 0.1$	Ns/m

- First, you have to generate an extensive set of training data for your model. It is up to you to decide how large (i.e. how many combinations of input sequences and output sequences) you you want this training set to be. To have some bounds for the training data, a suggestion is given below.

P_0	load amplitude	$1.0 \leq P_0 \leq 30$ N
ω	load frequency	$0.1 \leq \omega \leq 6$ rad/s
α	load phase	$0 \leq \alpha < 2\pi$ rad
h	time step	0.05 s
t_{max}	load and response sequence duration	100 s

- Next, you have to train the model. In our model, we have a sequence (load time history) as input and a sequence (response time history) as output, so we can use what is called a sequence-to-sequence RNN.

  In matlab, it is suggested to make use of the Deep Learning Toolbox for training the RNN. `https://se.mathworks.com/help/deeplearning/ug/sequence-to-sequence-regressio` `html`. Some example code for training the RNN is provided, but you have to write your own code for generating inputs and testing.

  In python, it is suggested to make use of Keras (Tensorflow). An example is again provided, although very little tuning of the RNN model setup has been done here. (You are likely to get better results if using matlab for this particular model).

- Finally, you should test your trained model by comparing its outputs for some new input loads which were not used during training. Discuss the performance of the model. How and to what extent you quantify this performance is up to you, but you should at least test some inputs that give stiffness-dominated, resonant, and inertia-dominated responses. Discuss both the amplitudes and phase predictions and transient vs. steady-state responses.

(C) **General periodic load**

A general periodic load consists of many harmonic components (Chapter 3.1) and can be related to how we will (eventually) model irregular sea states.

Investigations:

- Consider the rectangular periodic load from Chapter 3.1:

$$P(t) = a_0 + \sum_{n=1}^{\infty}(b_n \sin \omega_n t) \tag{7}$$

$$a_0 = \quad P_0/2 \tag{8}$$

$$b_n = \begin{cases} \frac{2P_0}{n\pi} & \text{for } n = 1, 3, 5... \\ 0 & \text{for } n = 2, 4, 6... \end{cases} \tag{9}$$

  Compare your results for an undamped system with $\omega_0 = 4\omega_p$ to the results from the compendium (see also lecture slides).

- Consider the same rectangular periodic load, let $\omega_p = 2\pi/10$ rad/s and $P_0 = 10$ N. Determine a metric to identify when the response has reached a steady-state. For a mass-spring-dashpot system with $m = 2$ kg, which combinations of $c \geq 0.1$ Nm/s and $k \geq 1$ N/m result in a steady-state response within $-1 < u(t) < 1$ m? (These minimum values for stiffness and damping are intended to reduce the computational demand for reaching steady-state).

(D) **Short impulsive load**

Here, we consider a load of short duration (remember what the criteria for "short" are!). You can consider a half sine wave for the load form.

- Document the effect of the duration of the load pulse on maximum response.
- What is the influence of damping on maximum response.
- Compare the results between simplified theory (compendium, chapter 3.2.2) and time integration.
- Find a set of inputs such that (at least) one of the three numerical integration methods gives a very different response from the others and explain why.

3 Example calling code

3.1 matlab

Listing 1: Sample calling code from Matlab

```
close all
clear all
clc

%% Input
% Input dynamic system characteristics
m = 1; %mass, kg
k = 1; %stiffness, N/m
c = 0.5; %damping, N/s
u0 = 0.2; % initial position, m
udot0 = 0.1; % initial velocity, m/s
loadtype = 3; % see if statements below for definition

h = 0.1; %time increment, s
tmax = 400; % time duration
t = (0:h:tmax).'; % time vector

%% formulate load vector
% load time series (just examples. Your code will be tested using
% different load time series);

if loadtype == 1 % harmonic load
    omega = 0.4; % frequency
    amp = 1; % amplitude
    eps1 =pi; % phase angle
    P = amp*cos(omega*t+eps1);  % load vector
%
elseif loadtype == 2 % general periodic load
    ncomponents = 3; % number of load components
    omegas = [0.9 1 1.1 1/4 1/6]; % load frequencies
    amps = [1 1 1 1 4]; % load amplitudes
    epss = [0 pi/4 0 pi 3*pi/4];
    P = zeros(length(t),1);
    for ii = 1:ncomponents
        P = P + amps(ii)*cos(omegas(ii)*t+epss(ii));
    end

elseif loadtype == 3 % short duration
    Pmax = 10; % maximum load, N
    t1 = 1; % time duration of load, s
    omega = 2*pi/(2*t1); % frequency for sine shape
```

```
    P = zeros(length(t),1); % load vector
    indt1 = sum(t<=t1);
    P(1:indt1) = Pmax*sin(omega*t(1:indt1));
end

%% student code called here!
urk = runge_kutta4(m,k,c,P,h,u0,udot0);
u = const_avg_acc(m,k,c,P,h,u0,udot0);
uss = dis_state_space(m,k,c,P,h,u0,udot0);


%% plot
figure(1)
%% plot load
subplot(2,1,1)
hold on
plot(t,P,'k')
ylabel('P(t), N')
xlabel('time, s')
xlim([0 max(t)])
grid on

subplot(2,1,2)
hold on
plot(t,u)
plot(t,urk)
plot(t,uss)
legend('CAA','RK4','DSS')
xlabel('time, s')
ylabel('u, m')
xlim([0 max(t)])
grid on
```

3.2 python

Listing 2: Sample calling code from python

```
# -*- coding: utf-8 -*-
#Python code for TMR 4182 exercise 3
#@author: bachynsk

import numpy as np
import matplotlib.pyplot as plt
from const_avg_acc import const_avg_acc
from runge_kutta4 import runge_kutta4
from dis_state_space import dis_state_space

```

```
##Input
# Input dynamic system characteristics
m = 1 #mass, kg
k = 1 #stiffness, N/m
c = 0.5 #damping, N/s
u0 = 0.2 # initial position, m
udot0 = 0.1 # initial velocity, m/s
loadtype = 2 # see if statements below for definition


h = 0.1 #time increment, s
tmax = 400 # time duration, s
t = (np.arange(0, tmax,h))
P = np.zeros(len(t)) # initialize load vector to zer

# formulate load vector
if (loadtype == 0): # harmonic load
    omega = 0.4 # frequency
    amp = 1 # amplitude
    eps1 = np.pi #phase angle
    P = amp*np.cos(omega*t+eps1);  # load vector

elif (loadtype == 1): # general periodic load
    ncomponents = 3 #number of load components
    omegas = np.array([0.9,1,1.1,0.25,1/6]) # load frequencies
    amps = [1, 1, 1, 1, 4] #load amplitudes
    epss = np.array([0, np.pi/4, 0, np.pi, 3*np.pi/4])

    for ii in range(0,ncomponents):
        P = P + amps[ii]*np.cos(omegas[ii]*t+epss[ii]);


else: # (loadtype == 3) short duration
    Pmax = 10 # maximum load, N
    t1 = 1 # time duration of load, s
    omega = 2*np.pi/(2*t1) # frequency for sine shape
    indt1 = sum(t<=t1) # find end of the impulse
    P[0:indt1] = Pmax*np.sin(omega*t[0:indt1])



## Student code called here!
u = const_avg_acc(m, k, c, P, h, u0, udot0)

urk = runge_kutta4(m,k,c,P,h,u0,udot0)

uss = dis_state_space(m,k,c,P,h,u0,udot0)
```

```
58
59 #
60 # plot
61 plt.figure(1)
62 plt.subplot(211)
63 plt.plot(t,P)
64 plt.grid(True)
65 plt.xlabel('time, s')
66 plt.ylabel('P, N')
67 plt.xlim((0,tmax))
68
69 plt.subplot(212)
70 plt.plot(t,u,'k', label='CAA')
71 plt.plot(t,urk,'b--', label='RK')
72 plt.plot(t,uss,'r--', label='DSS')
73 plt.legend()
74 plt.grid(True)
75 plt.xlabel('time, s')
76 plt.ylabel('u, m')
```