# Assignment 1—Group:

Chen Chen (123123123), Dania(123123123), Joyce(123123123)

2nd September 2018

**Abstract**

*Abstract text goes here, justified and in italics. The abstract would normally be one paragraph long.*

## Contents



## 1 Introduction

This template should be used as a starting point for your report.

# 2 Methods

12312323 Lee and Seung [2001] Guan et al. [2012]

# 3 Experiments

123

# 4 Conclusion

Your conclusion goes at the end, followed by References, which must follow the Vancouver Style (see: www.icmje.org/index.html). References begin below with a header that is centered. Only the first word of an article title is capitalized in the References.

# A Appendix

Algorithm 1: The Levenberg–Marquardt algorithm iteratively finds optimal macroscale Robin boundary conditions.

```
% solution of two layer periodic wave equation where both spring
    constant
% and density are periodic. Assume dirichlet boundary condition
% u(0,t)=u(l,t)=0. Assume initial condition
% u(x,0)=sin(pi*x(:))/10 and u_t(x,0)=sin(pi*x(:)).
% use explicitly central difference method solve for 0<x<l and
% for 0<t<T using n+1 space points and m+1 time points
% x(1)..x(n+1) and t(1)..t(m+1)
% chen chen 31/03/2014
% modified on 9/05/2015 for full plot
```

```
clear
%close all
global inf_d
inf_d=inf; %if d>=inf_d d=inf
global nvs % used in many places so parametrise and set just once
global nvs2 % used in many places so parametrise and set just once
 % compare these eigenvectors
global l num_of_node_in_a_period_perlayer;
global discrete h kamplitude kcamplitude rhoamplitude kphase kcphase rhophase numberoflayer;
numberoflayer=2;
%rng(449)%77

kamplitude=rand(numberoflayer,1);
kphase=rand(numberoflayer,1)*pi*2-pi;
kphase=rand(numberoflayer,1)*pi*2*0+0.1;
kcamplitude=rand(numberoflayer);
kcamplitude=triu(kcamplitude,1)+triu(kcamplitude,1)';
kcphase=rand(numberoflayer)*pi*2-pi;
kcphase=triu(kcphase,1)+(triu(kcphase,1))';
rhoamplitude=rand(numberoflayer,1);
rhophase=rand(numberoflayer,1)*pi*2-pi;

rhophase=rand(numberoflayer,1)*pi*0+0.2;
N=15% number of micro grid intervals in space
L=pi% spatial domain
h=L/N;
x=(0:h:L)';
midportion=1;
discrete=0;% if discrete, above phase and amplitude will not be used. Instead, specify the density and elasticities in files macro_rho, macro_kc and macro_k

%% This trunk of code allow me to specify any micro function e.g.
%non-trig functions
%microscale
num_of_node_in_a_period_perlayer=2%floor(N/num_of_ce7ylls);
```

```
num_of_cells=floor((N+1)/num_of_node_in_a_period_perlayer);
num_of_node_in_a_cell=num_of_node_in_a_period_perlayer*
    numberoflayer;
nvs=1:1%floor(num_of_cells/2);
nvs2=1%:floor(num_of_cells/2);% to optimize these
l=h*num_of_node_in_a_period_perlayer % period for rho and k
nop=L/l%number of period
%% construct B matrix
rhot=[];
for j=1:numberoflayer
    rhot=[rhot feval(@macro_rho,j,x(1:end))];
end
rhot([1 end],:)=nan;
rhotemp=rhot';
rho=rhotemp(:);
%% construct A matrix
K=zeros(numberoflayer,numberoflayer,
    num_of_node_in_a_period_perlayer);
KC=K;
K(:,:,num_of_node_in_a_period_perlayer)=diag(macro_k(1:
    numberoflayer,...
    (x(num_of_node_in_a_period_perlayer)...
    +x(num_of_node_in_a_period_perlayer+1))/2));%when starting
        the loop we need information about the lateral end spring
for i=[1:num_of_node_in_a_period_perlayer]
    K(:,:,i)=diag(macro_k(1:numberoflayer,(x(i)+x(i+1))/2));
    for j=1:numberoflayer
        for jj=1:numberoflayer
            KC(j,jj,i)=macro_kc(j,jj,x(i));
        end
    end
    KC(:,:,i)=KC(:,:,i)-diag(sum(KC(:,:,i),2))-(...
        K(:,:,mod(i-2,num_of_node_in_a_period_perlayer)+1)...
        +K(:,:,i));
end


%% write rho and k into txt file for reduce
fid = fopen('parameters.txt', 'w');
for i=[1:num_of_node_in_a_period_perlayer]-1
```

```
    for j=0:numberoflayer-1
        fprintf(fid,'k%d%d:=%6.4f;\n',i,j,K(j+1,j+1,i+1));
        fprintf(fid,'rho%d%d:=%6.4f;\n',i,j,feval(@macro_rho,j
            +1,x(i+1)));
    end
end
for i=[1:num_of_node_in_a_period_perlayer]-1
    for jj=1:numberoflayer-1
        for j=0:jj-1
            fprintf(fid,'kc%d%d%d:=%6.4f;\n',i,j,jj,KC(j+1,jj+1,
                i+1));
        end
    end
end
fprintf(fid,'nos:=%d;\n',num_of_node_in_a_period_perlayer);
fprintf(fid,'nol:=%d;\n',numberoflayer);
fprintf(fid,';end;\n',numberoflayer);
fclose(fid);
if isunix
    setenv('PATH', [fileread('/etc/paths')]);
    unix('unset␣DYLD_LIBRARY_PATH;reduce<wstrand_Nstep.red');
elseif ispc
    dos('d:\reduce-windows32-20110414\reduce<wstrand_Nstep.red')
end

%% solve the microscale eigen problem
%initialise A and u
%tri diagonal matrix
A=zeros(numberoflayer*(N-1),numberoflayer*(N+1));
count=0;
for i=1:numberoflayer:numberoflayer*(N-1)
    count=count+1;
    A(i:i+numberoflayer-1,i:i+3*numberoflayer-1)=...
        [K(:,:,mod(count-1,num_of_node_in_a_period_perlayer)+1)
            ...
        ,KC(:,:,mod(count,num_of_node_in_a_period_perlayer)+1)
            ...
        ,K(:,:,mod(count,num_of_node_in_a_period_perlayer)+1)];
end;
A=A/h^2;
```

```
AA=[eye(numberoflayer) zeros(numberoflayer,numberoflayer*N)
    A
    zeros(numberoflayer,numberoflayer*N),eye(numberoflayer)]; %
       with Dirichlet boundary condition

BB=diag(rho);
BB(1:numberoflayer,1:numberoflayer)=0;
BB(end-numberoflayer+1:end,end-numberoflayer+1:end)=0;

[v_ori,d_ori]=eig(AA,BB);
[d_ori1,j]=sort(-diag(d_ori));
v_ori=v_ori(:,j);

v_ori=v_ori(:,abs(d_ori1)<inf_d);
d_ori=d_ori1(abs(d_ori1)<inf_d); % get rid of inf
v_ori=v_ori*diag(2./sqrt(sum(v_ori.^2)))*diag(sign(sum(v_ori)));
   %normalise
avg_matrix=[];

v_ori_avg=v_ori(1:numberoflayer:end,:);
for ii=2:numberoflayer
    v_ori_avg=v_ori_avg+v_ori(ii:numberoflayer:end,:);
end
v_ori_avg=v_ori_avg/numberoflayer;

v_ori2=v_ori;
v_ori_avg2=v_ori_avg;
v_ori_avg2=v_ori_avg2*diag(1./sqrt(sum(v_ori_avg2.^2)));%
   normalise


%% Discretise the homogenized eigenproblem in the domain
   interior
% macroscale equation
konrho=load('macro');
i_homo=2:N;

A_homo_temp=full(-sparse(i_homo-1,i_homo,(2*konrho)/h^2,N-1,N+1)
    ...
    +sparse(i_homo-1,i_homo-1,konrho/h^2,N-1,N+1) ...
```

```
    +sparse(i_homo-1,i_homo+1,konrho/h^2,N-1,N+1));

B=eye(size(A_homo_temp)+[2 0]);
B(1,1)=0;
B(end,end)=0;

[dirichlet_error,v0,d0]=eigenvector_wave( ...
    [0,0],v_ori_avg(:,nvs2),A_homo_temp,i_homo,B,midportion,
        num_of_node_in_a_period_perlayer);


%% Apply Levenberg--Marquardt algorithm
options = optimset('Algorithm','levenberg-marquardt');
% best(1) is the coefficient of dudx on x=0 and best(2) is that
    on x=L
[bestt,df_v_best] = lsqnonlin(@(xx)...
    eigenvector_wave(xx,v_ori_avg2(:,nvs2),A_homo_temp,i_homo,B,
        midportion,num_of_node_in_a_period_perlayer) ...
    ,[0 0]',[],[],options);
[besterror,vbest,dbest]=eigenvector_wave(bestt,v_ori_avg(:,nvs2)
    ,A_homo_temp,i_homo,B,midportion,
    num_of_node_in_a_period_perlayer);
```

# References

Daniel D. Lee and H. Sebastian Seung. Algorithms for non-negative matrix factorization. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 556–562. MIT Press, 2001. URL http://papers.nips.cc/paper/1861-algorithms-for-non-negative-matrix-factorization.pdf.

N. Guan, D. Tao, Z. Luo, and B. Yuan. Nenmf: An optimal gradient method for nonnegative matrix factorization. *IEEE Transactions on Signal Processing*, 60(6):2882–2898, June 2012. ISSN 1053-587X. doi:10.1109/TSP.2012.2190406.