DD2370 Computational Methods for Electromagnetics *Extracting Eigenfrequencies in Time-Domain Simulations*

Stefano Markidis, KTH Royal Institute of Technology

Objective

We seek the spectrum –omega^2 of the operator L = $\partial^2/\partial x^2$ on the interval 0 < x < a with the boundary conditions f(0) = f(a) = 0. The true eigenfrequencies are

$$\boxed{\omega_m = \frac{m\pi}{a}, \quad m = 1, 2, \ldots .}$$

Find Spectrum with FD in Time-Domain

- The spectrum of L can be found by solving the wave equation

$$\frac{\partial^2 f}{\partial t^2} = \frac{\partial^2 f}{\partial x^2}, \quad 0 < x < a, \quad f(0,t) = f(a,t) = 0.$$

- We use the simplest finite difference scheme:

$$f_i^{(n+1)} = 2f_i^{(n)} - f_i^{(n-1)} + \left(\frac{\Delta t}{\Delta x}\right)^2 \left(f_{i+1}^{(n)} + f_{i-1}^{(n)} - 2f_i^{(n)}\right)$$

Matlab – Solving FD in Time and use Detectors

MATLAB function that records two signals at two locations and stores them in arrays (keeping the history of Ex) to be analyzed afterwards:

- the midpoint
- and a point close to the left boundary

More than one signal is recorded because some eigenmodes can be undetected if the eigenfunction **f has a node (i.e., zero amplitude) at the “detector” location**.

An eigenmode may also be undetected if the initial condition does not excite it at sufficient amplitude (fields are initialized randomly)

The Eigenfrequency **are calculated using FFT**.

Matlab - Wave1D.m

```
% ----------------------------------------------------------------
% Time step 1D wave equation using two time-levels f0 & f1
% ----------------------------------------------------------------
function [omega, s1, s2] = Wave1D(a, time, nx)

% Arguments:
%    a     = the length of the interval
%    time  = the total time interval for the simulation
%    nx    = the number of subintervals in the domain (0,a)
% Returns:
%    omega = the angular frequencies
%    s1    = the complex Fourier transform of data at x = a/5
%    s2    = the complex Fourier transform of data at x = a/2

f0           = randn(nx+1, 1); % Initialize with random numbers
f0(1,1)      = 0;              % Boundary condition at x = 0
f0(nx+1,1) = 0;                % Boundary condition at x = a

f1           = randn(nx+1, 1); % Initialize with random numbers
f1(1,1)      = 0;              % Boundary condition at x = 0
f1(nx+1,1) = 0;                % Boundary condition at x = a

dx           = a/nx;           % The cell size
d2tmax       = 1.9*dx;         % The time step must satisfy
                               % 2*dt < 2*dx for stability

ntime = round(time/d2tmax + 1);  % The number of time steps
dt = time/(2*ntime);             % The time step

% Initialize the coefficient matrix for updating the solution f
A = spalloc(nx+1,nx+1,3*(nx+1)); % Sparse empty matrix with
                                 % 3*(nx+1) nonzero entries
for i = 2:nx
  A(i,i)   = 2*(1-(dt/dx)^2);    % Diagonal entries
  A(i,i+1) = (dt/dx)^2;          % Upper diagonal entries
  A(i,i-1) = (dt/dx)^2;          % Lower diagonal entries
end

% Time step and sample the solution
% Sample location #1 is close to the left boundary
% Sample location #2 is at the midpoint of the domain
for itime = 1:ntime % Every 'itime' means two time steps 'dt'

  f0                = A*f1 - f0;          % Update
  sign1(2*itime-1) = f0(round(1+nx/5));   % Sample at location #1
  sign2(2*itime-1) = f0(round(1+nx/2));   % Sample at location #2

  f1                = A*f0 - f1;          % Update
  sign1(2*itime)    = f1(round(1+nx/5));  % Sample at location #1
  sign2(2*itime)    = f1(round(1+nx/2));  % Sample at location #2

end

% Compute the discrete Fourier transform of
% the time-domain signals
spectr1      = fft(sign1);
spectr2      = fft(sign2);

% In the MATLAB implementation of the function fft(),
% the first half of the output corresponds to positive frequency
s1(1:ntime) = spectr1(1:ntime);
s2(1:ntime) = spectr2(1:ntime);

% Frequency vector for use with 's1' and 's2'
omega        = (2*pi/time)*linspace(0, ntime-1, ntime);
```

Maxwell's Solver – Recording in Two Points

```
% ------------------------------------------------------------------
% Time step 1D wave equation using two time-levels f0 & f1
% ------------------------------------------------------------------
function [omega, s1, s2] = Wave1D(a, time, nx)

% Arguments:
%    a     = the length of the interval
%    time  = the total time interval for the simulation
%    nx    = the number of subintervals in the domain (0,a)
% Returns:
```

$$f_i^{(n+1)} = 2f_i^{(n)} - f_i^{(n-1)} + \left(\frac{\Delta t}{\Delta x}\right)^2 \left(f_{i+1}^{(n)} + f_{i-1}^{(n)} - 2f_i^{(n)}\right)$$

```
a/5
a/2

f0          = randn(nx+1, 1); % Initialize with random numbers
f0(1,1)     = 0;              % Boundary condition at x = 0
f0(nx+1,1)  = 0;              % Boundary condition at x = a

f1          = randn(nx+1, 1); % Initialize with random numbers
f1(1,1)     = 0;              % Boundary condition at x = 0
f1(nx+1,1)  = 0;              % Boundary condition at x = a

dx          = a/nx;           % The cell size
d2tmax      = 1.9*dx;         % The time step must satisfy
                              % 2*dt < 2*dx for stability

ntime = round(time/d2tmax + 1);  % The number of time steps
dt = time/(2*ntime);             % The time step

% Initialize the coefficient matrix for updating the solution f
A = spalloc(nx+1,nx+1,3*(nx+1)); % Sparse empty matrix with
                                 % 3*(nx+1) nonzero entries
```

```
  A(i,i-1) = (dt/dx)^2;          % Lower diagonal entries
end

% Time step and sample the solution
% Sample location #1 is close to the left boundary
% Sample location #2 is at the midpoint of the domain
for itime = 1:ntime % Every 'itime' means two time steps 'dt'

  f0                 = A*f1 - f0;              % Update
  sign1(2*itime-1)   = f0(round(1+nx/5));     % Sample at location #1
  sign2(2*itime-1)   = f0(round(1+nx/2));     % Sample at location #2

  f1                 = A*f0 - f1;              % Update
  sign1(2*itime)     = f1(round(1+nx/5));     % Sample at location #1
  sign2(2*itime)     = f1(round(1+nx/2));     % Sample at location #2

end

% Compute the discrete Fourier transform of
% the time-domain signals
spectr1      = fft(sign1);
spectr2      = fft(sign2);

% In the MATLAB implementation of the function fft(),
% the first half of the output corresponds to positive frequency
s1(1:ntime) = spectr1(1:ntime);
s2(1:ntime) = spectr2(1:ntime);

% Frequency vector for use with 's1' and 's2'
omega       = (2*pi/time)*linspace(0, ntime-1, ntime);
```

Extracting Eigenfrequencies with FFT

```
% ------------------------------------------------------------
% Time step 1D wave equation using two time-levels f0 & f1
% ------------------------------------------------------------
function [omega, s1, s2] = Wave1D(a, time, nx)

% Arguments:
%    a     = the length of the interval
%    time  = the total time interval for the simulation
%    nx    = the number of subintervals in the domain (0,a)
% Returns:
%    omega = the angular frequencies
%    s1    = the complex Fourier transform of data at x = a/5
%    s2    = the complex Fourier transform of data at x = a/2

f0          = randn(nx+1, 1); % Initialize with random numbers
f0(1,1)     = 0;              % Boundary condition at x = 0
f0(nx+1,1)  = 0;              % Boundary condition at x = a

f1          = randn(nx+1, 1); % Initialize with random numbers
f1(1,1)     = 0;              % Boundary condition at x = 0
f1(nx+1,1)  = 0;              % Boundary condition at x = a

dx          = a/nx;           % The cell size
d2tmax      = 1.9*dx;         % The time step must satisfy
                              % 2*dt < 2*dx for stability

ntime = round(time/d2tmax + 1);  % The number of time steps
dt = time/(2*ntime);             % The time step

% Initialize the coefficient matrix for updating the solution f
A = spalloc(nx+1,nx+1,3*(nx+1)); % Sparse empty matrix with
                                 % 3*(nx+1) nonzero entries
```

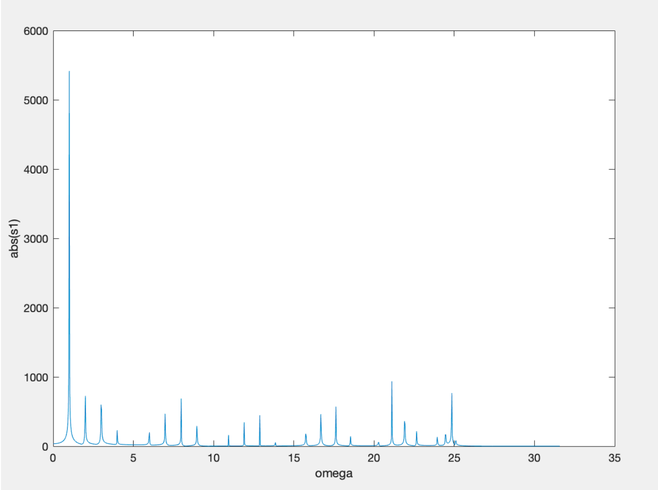
```
    A(i,i-1) = (dt/dx)^2;          % Lower diagonal entries
end

% Time step and sample the solution
% Sample location #1 is close to the left boundary
% Sample location #2 is at the midpoint of the domain
for itime = 1:ntime % Every 'itime' means two time steps 'dt'

    f0                = A*f1 - f0;          % Update
    sign1(2*itime-1)  = f0(round(1+nx/5));  % Sample at location #1
    sign2(2*itime-1)  = f0(round(1+nx/2));  % Sample at location #2

    f1                = A*f0 - f1;          % Update
    sign1(2*itime)    = f1(round(1+nx/5));  % Sample at location #1
    sign2(2*itime)    = f1(round(1+nx/2));  % Sample at location #2

end

% Compute the discrete Fourier transform of
% the time-domain signals
spectr1     = fft(sign1);
spectr2     = fft(sign2);

% In the MATLAB implementation of the function fft(),
% the first half of the output corresponds to positive frequency
s1(1:ntime) = spectr1(1:ntime);
s2(1:ntime) = spectr2(1:ntime);

% Frequency vector for use with 's1' and 's2'
omega       = (2*pi/time)*linspace(0, ntime-1, ntime);
```

Extracting the Eigenfrequencies

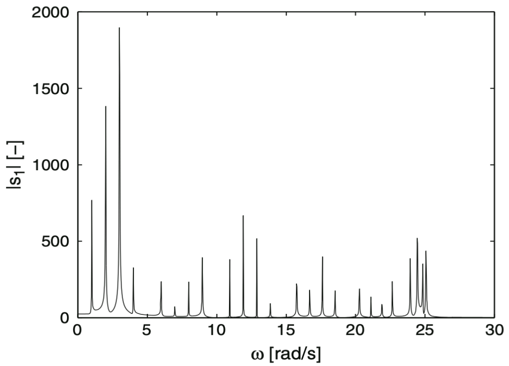
- We call the routine by to compute the spectrum of the second derivative on the interval [0, pi].

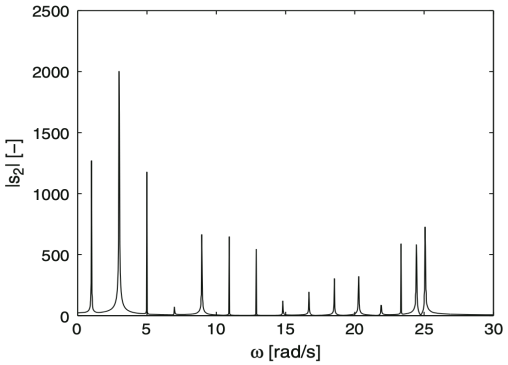
time nx

```
[omega, s1, s2] = Wave1D(pi, 200, 30)
plot(omega,abs(s1))
```



- The spectral peaks fall very close to integers, as they should. Because of the spatial locations of the observation points, **the even peaks are absent in s2 and those divisible by 5 in s1**.
    - These are the eigenmodes that have zero amplitudes (nodes) at the respective observation points.
- A significant advantage of such a time-domain calculation is that we can find the whole spectrum (given several sensors) from a single simulation.





Considerations on Extract Eigenfrequencies

- The longer the simulation is run, the sharper the spectral peaks become, and the better the eigenfrequencies are determined, but the convergence of the estimated frequencies is slow.
- When there is no damping, the estimates are sensitive to how close the various frequency components are to **making an integer number of oscillations during the simulation**.
  - This is because the fast Fourier transform (FFT) treats the signal as if it were periodic with a period equal to the simulated time.
    - If the time interval is not an integer number of wave periods, either the signal or its time derivative will have a jump at the end of the time window, and this broadens the Fourier spectrum of a sinusoidal signal.

Assignment – Compare Spectrum

- Compare the spectrum obtained by calling the time-stepping routine by Wave1D(pi,20*pi,30)
  - it gives 10 (analytical) oscillation periods for the first mode, and where all the low-order modes make approximately an integer number of oscillations
- Wave1D(pi,21*pi,30), where the first mode has 10.5 oscillation periods.