```
clc; close all; clear;
```

Rigoni Luca 247451

Advanced Mechanical Systems - Assigment 2 - Model of an Electromagnetic Energy Harvester

Table of Contents

System:

- voice coil: magnetic circuit with fixed coil and magnet connected to the circuit through elastic element

- voice coil stator attched to machine frame, while the magnet oscillate in respect of the stator, inducing a current in the coil

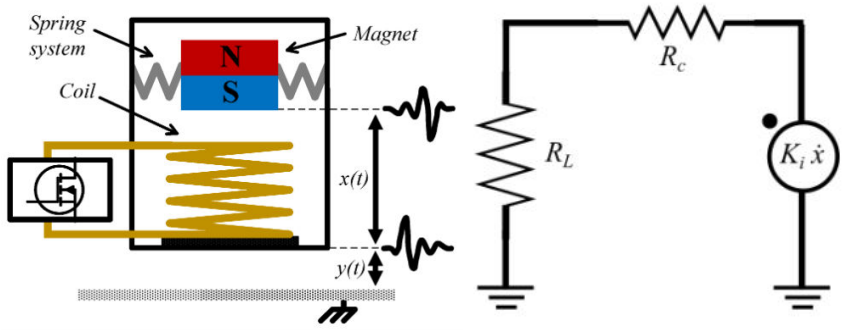
- the equivalent circuit consists of an emf generator and two resistances

Assumptions of the system:

- small inertia harvester do not affect the vibration acceleration $\ddot{y}(t)$

Data:

```
m = 93e-3;      % [kg] magnet mass
c = 1.52;       % [Ns/m] mechanical magnet damping
ks = 35.6e3;    % [N/m] supension stiffness
Ki = 32.6;      % [N/A] force/emf constant
Rc = 300;       % [Ohm] coil resistance
```



1) Analytical analysis

a) Harmonic acceleration

The system behaves as a spring mass damper system, where:

- mass = magnet mass (neglecting springs inertia)
- damper = machanical magnet damper (ideal springs have no damping) --> highly underdamped system
- spring = stiffness of ideal spring

Note the cosidered system has single linear spring along the Degree of Freedom direction.

Appropriate models are choosen:

- the single DoF model, on top of which the Newton law can be written
- the electrical circuit model, R L equivalent circuit

$$\begin{cases} m\ddot{x} + c\dot{x} + k_s x - k_i i = -m\ddot{y} \\ L\dfrac{di}{dt} + Ri + K_i\dot{x} = v \end{cases}$$

Assumptions have to be made:

- L = 0 -> negligible inductance due to low current
- v = 0 -> no external energy injected in the system
- c -> daping only due to magnet

The system becomes:

$$\begin{cases} m\ddot{x} + c\dot{x} + k_s x - k_i i = -m\ddot{y} \\ Ri + K_i\dot{x} = 0 \end{cases}$$

$$i = -\frac{K_i}{R}\,\dot{x}$$

Combining the 2 equations we obtain:

$$m\ddot{x} + \left(\frac{k_i^2}{R} + c\right)\dot{x} + k_s x = -m\ddot{y}$$

term $\frac{k_i^2}{R}$ represent the additional damping

- **Compute the transfer function** $H(\omega)$

Since we are interested in the frequency response, the Laplace trasform is performed assuming $\ddot{y} = a$ --> $\frac{X(\omega)}{A(\omega)}$

```
syms R_L R_c K_i k_s C M w A_0 s real
assume([R_L R_c K_i k_s C M w A_0] >= 0);

% Define equivalent Electric Resistance and Mechanical Damping
R_eq = R_c + R_L;
c_eq = K_i^2/R_eq + C;

num_sys = [0 0 -M];
den_sys = [M c_eq k_s];

num_sys_a = -M;
den_sys_a = M * s^2 + c_eq * s + k_s;

num_sys_a_freq = subs(num_sys_a, s, 1i*w);
den_sys_a_freq = subs(den_sys_a, s, 1i*w);
num_sys_a_freq_re = real(num_sys_a_freq);
num_sys_a_freq_im = imag(num_sys_a_freq);
den_sys_a_freq_re = real(den_sys_a_freq);
den_sys_a_freq_im = imag(den_sys_a_freq);
```

$$H(\omega) = \frac{X(\omega)}{\ddot{Y}(\omega)} = \frac{X}{A}$$

We obtain the Transfer Function X(w)/A(w)

input: machine frame accelerations

output: magnet displacement in respect of the stator coil

```
H_sys_a = num_sys_a_freq / den_sys_a_freq
```

```
H_sys_a =
```

$$-\frac{M}{k_s - M\,w^2 + w\left(C + \frac{K_i^2}{R_L + R_c}\right)\mathrm{i}}$$

```
% Transfer Function Magnitude
H_sys_a_module = simplify(simplify(sqrt(num_sys_a_freq_re^2 +
num_sys_a_freq_im^2)) / simplify(sqrt(den_sys_a_freq_re^2 + den_sys_a_freq_im^2)))
```

H_sys_a_module =

$$\frac{M}{\sqrt{w^2\left(C + \frac{K_i^2}{R_L + R_c}\right)^2 + \left(k_s - M\,w^2\right)^2}}$$

- **Compute the relative displacement** $X(\omega)$

```
X_a_module = H_sys_a_module * A_0
```

X_a_module =

$$\frac{A_0\,M}{\sqrt{w^2\left(C + \frac{K_i^2}{R_L + R_c}\right)^2 + \left(k_s - M\,w^2\right)^2}}$$

- **Compute the voltage** $V(\omega)$

$$i = -\frac{K_i}{R}\,\dot{x}$$

```
i_a_module = abs(- K_i / R_eq * 1i*w * X_a_module)
```

i_a_module =

$$\frac{A_0\,K_i\,M\,w}{(R_L + R_c)\sqrt{w^2\left(C + \frac{K_i^2}{R_L + R_c}\right)^2 + \left(k_s - M\,w^2\right)^2}}$$

```
% i_a_complex = - K_i / R_eq * 1i*w * X_a_module

V_a_module = R_L * i_a_module
```

V_a_module =

$$\frac{A_0\,K_i\,M\,R_L\,w}{(R_L + R_c)\sqrt{w^2\left(C + \frac{K_i^2}{R_L + R_c}\right)^2 + \left(k_s - M\,w^2\right)^2}}$$

```
% V_a_complex = R_L * i_a_complex
```

- **Compute the power** $P(\omega)$

This is the average power harvested in a period as finction of the frequency:

```
P_a_module = 1/2 * R_L * i_a_module^2
```

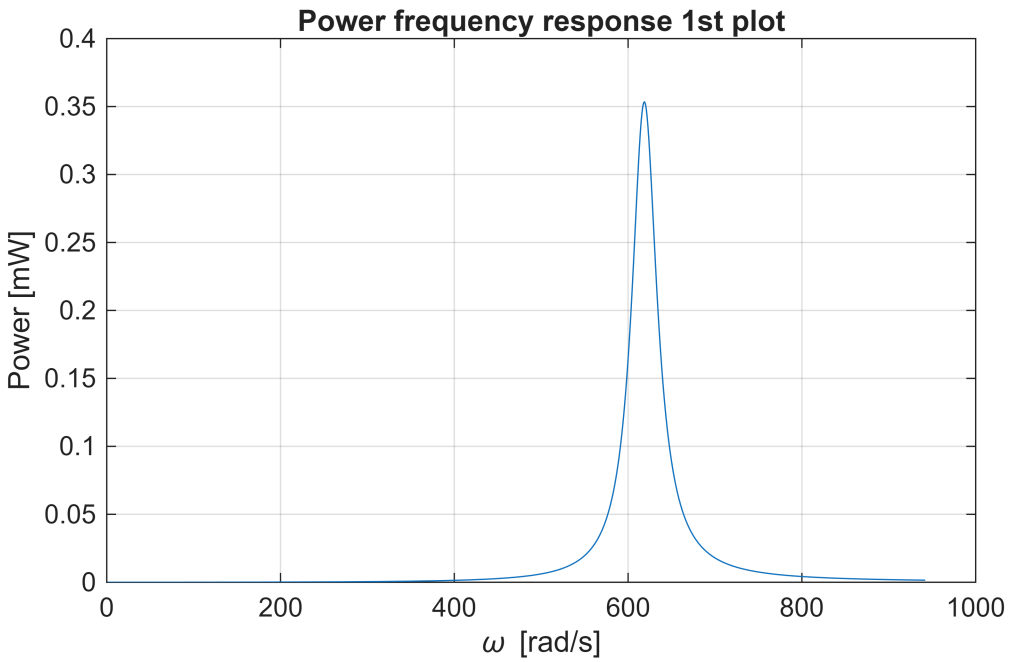
P_a_module =

$$\frac{A_0^2\, K_i^2\, M^2\, R_L\, w^2}{2\,(R_L+R_c)^2\left(w^2\left(C+\frac{K_i^2}{R_L+R_c}\right)^2+\left(k_s-M\,w^2\right)^2\right)}$$

```
% P_a_complex_a = 1/2 * V_a_module * conj(i_a_module) % 1/2 V I* (complex conj of I)
% P_a_complex_b = 1/2 * V_a_complex * conj(i_a_module)

A0 = 1; % [m/s^2] typical train suspension acceleration
Rl = 300; % [Ohm]

% P_a_module_num = eval(subs(P_a_module,{R_c K_i k_s C M},{Rc Ki ks c m}))
% find w_a A_0 plot for different R_L or maybe better find optimal R_L
% (analytic expression)
w_max = 150*2*pi;
w_sweep = (1:1:w_max)';
P_a_module_num = eval(subs(P_a_module,{R_c K_i k_s C M R_L A_0 w},{Rc Ki ks c m Rl
A0 w_sweep}));
```

```
Warning: The function sym/eval is deprecated and will be removed in a future release. Depending on the
usage, use subs or double instead.
```

```
plot(P_a_module_num*1e3)
xlabel('\omega [rad/s]')
ylabel('Power [mW]')
title('Power frequency response 1st plot')
grid on
```




```
% Find resonance frequency
[pks, ipks] = findpeaks(P_a_module_num);
w_n = w_sweep(ipks);
f_n = w_n/(2*pi);
disp('From graph computation:')
```

```
From graph computation:
```

```
disp(['Resonance Frequency w_n: ', num2str(w_n), ' rad/s'])
```

```
Resonance Frequency w_n: 619 rad/s
```

```
disp(['Resonance Frequency f_n: ', num2str(f_n), ' Hz'])
```

```
Resonance Frequency f_n: 98.5169 Hz
```

The theoretical natural frequency is consistent to the one extracted from the frequency repsonse.

```
% Define Excitation Frequency:
w_e = sqrt(ks/m); % [rad/s]
f_e = w_e / (2*pi); % [Hz]
disp('From theory formula:')
```

```
From theory formula:
```

```
disp(['Resonance Frequency w_e: ', num2str(w_e), ' rad/s'])
```

```
Resonance Frequency w_e: 618.7049 rad/s
```

```
disp(['Resonance Frequency f_e: ', num2str(f_e), ' Hz'])
```

```
Resonance Frequency f_e: 98.4699 Hz
```

Proof / Cross check:

the computed exitation frequency equals the one extracted from transfer function.

b) Optimal value of the load resistance R_L to extract maxinum power

- Find the optimal value for Load Resistance

```
P_a_module_wn = simplify(subs(P_a_module,[w], [sqrt(k_s/M)]));
d__P_a_module_wn__dRL = diff(P_a_module_wn, R_L);
R_L_sol = solve(d__P_a_module_wn__dRL==0, R_L)
```

```
R_L_sol =
```

$$\frac{K_i^2 + C\,R_c}{C}$$

```
R_L_opt = double(subs(R_L_sol, [R_c K_i C], [Rc Ki c])); % [Ohm]
```

- Plot to verify the existance of only one value:

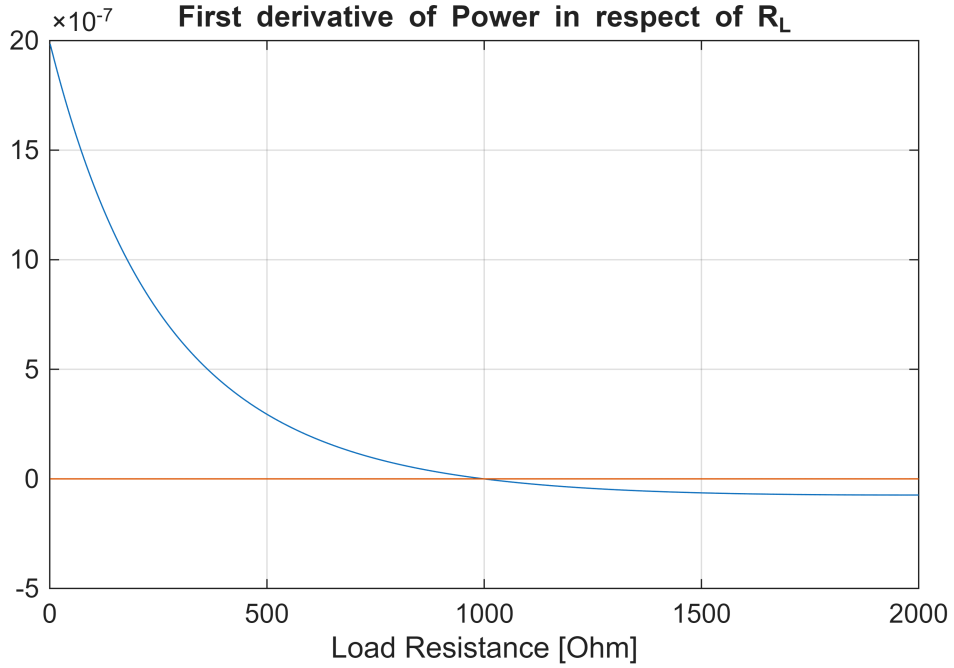
```
d__P_a_module__dRL = diff(P_a_module, R_L);
d__P_a_module__dRL_num = eval(subs(d__P_a_module__dRL,{R_c K_i k_s C M A_0 w},{Rc
Ki ks c m A0 w_e}));
```

```
Warning: The function sym/eval is deprecated and will be removed in a future release. Depending on the
usage, use subs or double instead.
```

```
delta_R_L = 1;
R_L_sweep = (delta_R_L:delta_R_L:2000)';
d__P_a_module__dRL_num_w_n = eval(subs(d__P_a_module__dRL,{R_c K_i k_s C M R_L A_0
w},{Rc Ki ks c m R_L_sweep A0 w_e}));
```

```
Warning: The function sym/eval is deprecated and will be removed in a future release. Depending on the
usage, use subs or double instead.
```

```
figure();
plot(d__P_a_module__dRL_num_w_n);
hold on;
plot(zeros(size(R_L_sweep)))
title('First derivative of Power in respect of R_L')
xlabel('Load Resistance [Ohm]')
grid on
hold off;
```




```
disp(['Optimal load resistance R_L: ', num2str(R_L_opt), ' Ohm'])
```

```
Optimal load resistance R_L: 999.1842 Ohm
```

```
Req = Rc + R_L_opt;
```

c) Plot of the frequency responses with accelerations 'g' and '2g'

Plot the Position, the Voltage and the Power frequency responses

c1) Position frequency response $X(\omega)$

```
g = 9.81; % [m/s^2] gravitational acceleration

X_a_module_num_g = eval(subs(X_a_module,{R_c K_i k_s C M R_L A_0 w},{Rc Ki ks c m
R_L_opt g w_sweep}));
```

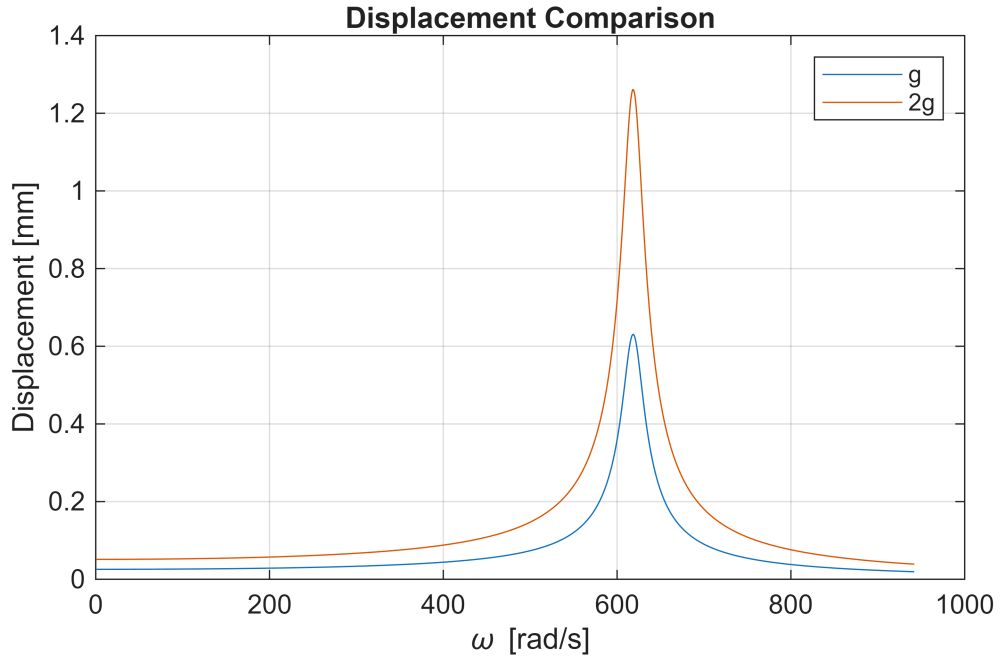
```
Warning: The function sym/eval is deprecated and will be removed in a future release. Depending on the
usage, use subs or double instead.
```

```
X_a_module_num_2g = eval(subs(X_a_module,{R_c K_i k_s C M R_L A_0 w},{Rc Ki ks c m
R_L_opt 2*g w_sweep}));
```

```
Warning: The function sym/eval is deprecated and will be removed in a future release. Depending on the
usage, use subs or double instead.
```

```
figure();
plot(X_a_module_num_g*1e3)
hold on
plot(X_a_module_num_2g*1e3)
xlabel('\omega [rad/s]')
```

```
ylabel('Displacement [mm]')
grid on
legend('g','2g')
title('Displacement Comparison')
hold off
```




```
[peak_value_X_g, peak_idx_X_g] = findpeaks(X_a_module_num_g);
[peak_value_X_2g, peak_idx_X_2g] = findpeaks(X_a_module_num_2g);
peak_ratio_X = peak_value_X_2g / peak_value_X_g;
disp(['Peak ratio displacement X: ',num2str(peak_ratio_X)]);
```

```
Peak ratio displacement X: 2
```

Curiosity:

Let's compute the displacement for constant acceleration e.g. zero frequency

```
X_zero_frequency = subs(X_a_module, {w}, {0});
disp('Displacement for zero frequency:')
```

```
Displacement for zero frequency:
```

```
disp(X_zero_frequency)
```

$$\frac{A_0 M}{k_s}$$

It is in the form of $A_0 \frac{1}{\omega_n^2}$, --> $\omega_n^2 = \frac{k_s}{m}$

c2) Voltage frequency response $V(\omega)$

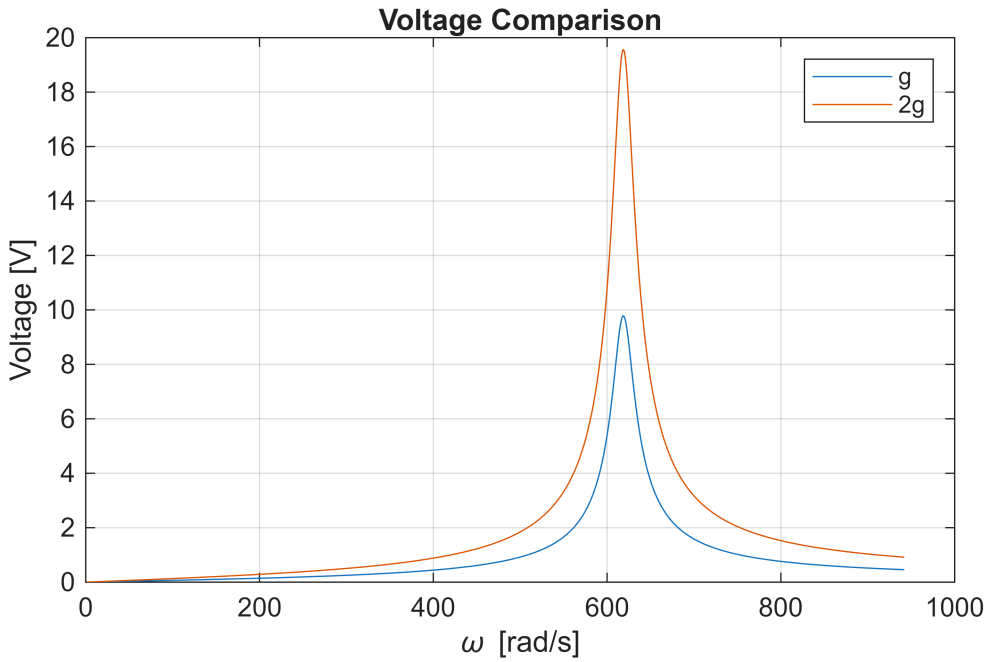
```
V_a_module_num_g = eval(subs(V_a_module,{R_c K_i k_s C M R_L A_0 w},{Rc Ki ks c m
R_L_opt g w_sweep}));
```

```
Warning: The function sym/eval is deprecated and will be removed in a future release. Depending on the
usage, use subs or double instead.
```

```
V_a_module_num_2g = eval(subs(V_a_module,{R_c K_i k_s C M R_L A_0 w},{Rc Ki ks c m
R_L_opt 2*g w_sweep}));
```

```
Warning: The function sym/eval is deprecated and will be removed in a future release. Depending on the
usage, use subs or double instead.
```

```
figure();
plot(V_a_module_num_g)
hold on
plot(V_a_module_num_2g)
xlabel('\omega [rad/s]')
ylabel('Voltage [V]')
grid on
legend('g','2g')
title('Voltage Comparison')
hold off
```




```
[peak_value_V_g, peak_idx_V_g] = findpeaks(V_a_module_num_g);
[peak_value_V_2g, peak_idx_V_2g] = findpeaks(V_a_module_num_2g);
peak_ratio_V = peak_value_V_2g / peak_value_V_g;
disp(['Peak ratio voltage V: ',num2str(peak_ratio_V)]);
```

```
Peak ratio voltage V: 2
```

c3) Power frequency response $P(\omega)$

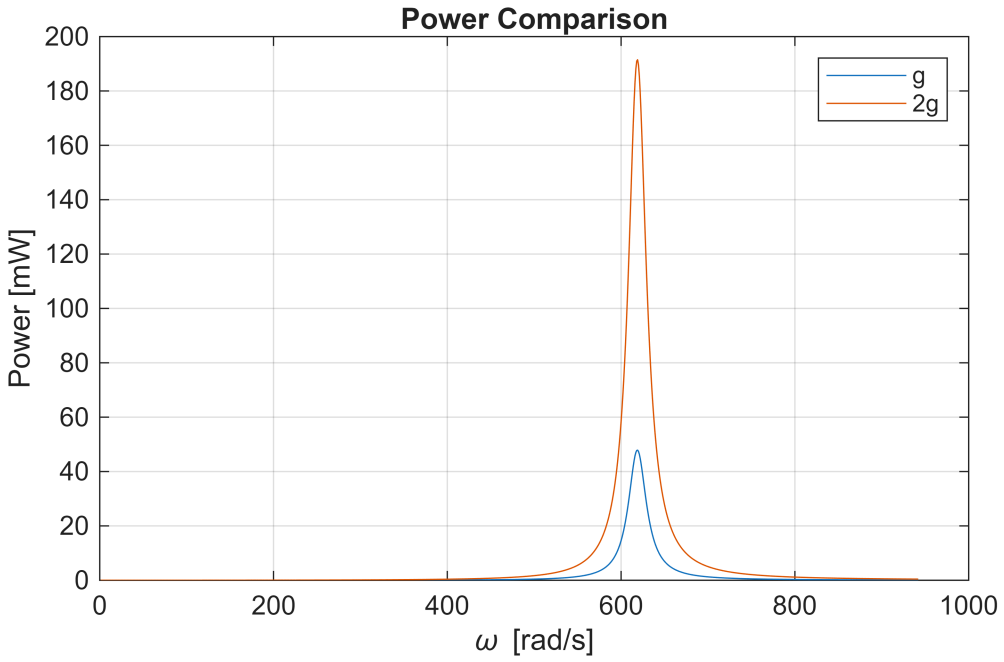
```
P_a_module_num_g = eval(subs(P_a_module,{R_c K_i k_s C M R_L A_0 w},{Rc Ki ks c m
R_L_opt g w_sweep}));
```

```
Warning: The function sym/eval is deprecated and will be removed in a future release. Depending on the
usage, use subs or double instead.
```

```
P_a_module_num_2g = eval(subs(P_a_module,{R_c K_i k_s C M R_L A_0 w},{Rc Ki ks c m
R_L_opt 2*g w_sweep}));
```

```
Warning: The function sym/eval is deprecated and will be removed in a future release. Depending on the
usage, use subs or double instead.
```

```
figure();
plot(P_a_module_num_g * 1e3)
hold on
plot(P_a_module_num_2g * 1e3)
xlabel('\omega [rad/s]')
ylabel('Power [mW]')
grid on
legend('g','2g')
title('Power Comparison')
hold off
```




```
[peak_value_P_g, peak_idx_P_g] = findpeaks(P_a_module_num_g);
[peak_value_P_2g, peak_idx_P_2g] = findpeaks(P_a_module_num_2g);
peak_ratio_P = peak_value_P_2g / peak_value_P_g;
disp(['Peak ratio power P: ',num2str(peak_ratio_P)]);
```

```
Peak ratio power P: 4
```

Because Power scale with the square of the acceleration (see point **a)**

```
P_a_module
```

```
P_a_module =
```

$$\frac{A_0^2\, K_i^2\, M^2\, R_L\, w^2}{2\,(R_L+R_c)^2\left(w^2\left(C+\frac{K_i^2}{R_L+R_c}\right)^2+\left(k_s-M\,w^2\right)^2\right)}$$

2) Numerical Analysis

a) time-domain dynamic model of the harvester

a1) LINEAR HARVESTER

$$m\ddot{x}+c\dot{x}+\mathrm{kx}+\frac{K_i^2}{R_c+R_L}\dot{x}=-m\ddot{y}$$

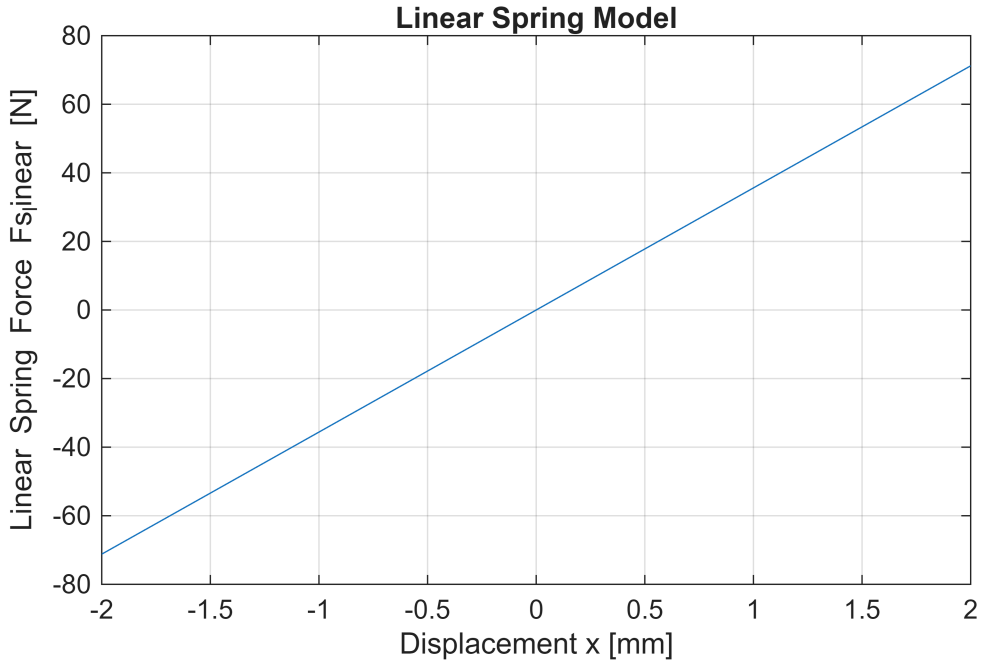
$$\ddot{x}=-\frac{c+\frac{K_i^2}{R_c+R_L}}{m}\,\dot{x}\,-\frac{k}{m}\,x-\ddot{y}$$

```
% Simulink parameters
coeff_xdot = (c + (Ki^2)/(Rc + R_L_opt))/(m);
coeff_x = ks/m;

syms Fs_linear k x
eq_linear = Fs_linear == k*x;
disp(eq_linear)
```

$$\mathrm{Fs}_{\mathrm{linear}}=k\,x$$

```
x_min = -2 * 1e-3; % [m]
x_max = 2 * 1e-3; % [m]
delta_x = x_max *0.01;
x_sweep = (x_min:delta_x:x_max)' ;

Fs_linear_sweep = ks * x_sweep;

figure();
plot(x_sweep * 1e3, Fs_linear_sweep)
hold on
xlabel('Displacement x [mm]')
ylabel('Linear Spring Force Fs_linear [N]')
title('Linear Spring Model')
grid on
```

```
hold off
```



a1.1) Simulink: linear harvester HARMONIC input

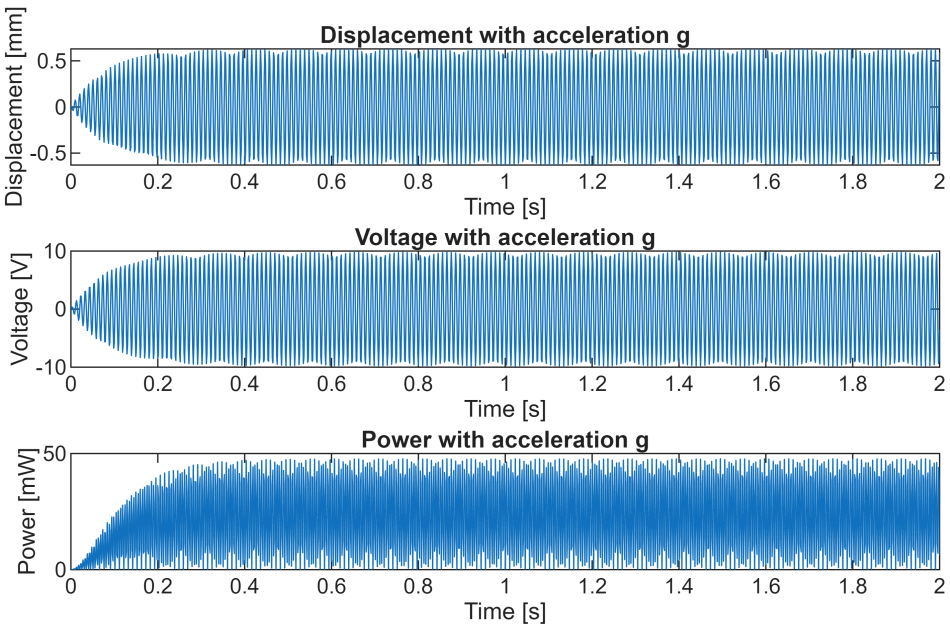
Harmonic = Steady State Amplitude

```
inputData_harmonic = {g, 2*g};
size_inputData_harmonic = length(inputData_harmonic);

% f_harmonic = f_e; % [Hz]
w_harmonic =618.7048560701065; % [rad/s]
T_sim_harmonic = 2; % [s]

% Initialize a cell array to store output data
Displacement_linear_harmonic = cell(1,size_inputData_harmonic);
Voltage_linear_harmonic = cell(1,size_inputData_harmonic);
Power_linear_harmonic = cell(1,size_inputData_harmonic);

modelName = 'Linear_harvester_harmonic_1_1';

for i = 1:size_inputData_harmonic
    % Assign the current time series to the Simulink input
    assignin('base', 'inputAcceleration', inputData_harmonic{i});

    % Run the simulation
    simOut = sim(modelName, 'ReturnWorkspaceOutputs', 'on');

    % Extract output data from the simulation
    Displacement_linear_harmonic{i} = simOut.get('Displacement');
    Voltage_linear_harmonic{i} = simOut.get('Voltage');
    Power_linear_harmonic{i} = simOut.get('Power');
```

```
    % Save output with a unique name in the workspace
    assignin('base', ['Displacement_linear_harmonic_' num2str(i)],
Displacement_linear_harmonic{i});
    assignin('base', ['Voltage_linear_harmonic_' num2str(i)],
Voltage_linear_harmonic{i});
    assignin('base', ['Power_linear_harmonic_' num2str(i)],
Power_linear_harmonic{i});
end

disp('HARMONIC ACCELERATION g')
```

```
HARMONIC ACCELERATION g
```

```
figure();
subplot(3, 1, 1)
plot(Displacement_linear_harmonic_1 * 1e3)
xlabel('Time [s]')
ylabel('Displacement [mm]')
title('Displacement with acceleration g');
subplot(3, 1, 2)
plot(Voltage_linear_harmonic_1)
xlabel('Time [s]')
ylabel('Voltage [V]')
title('Voltage with acceleration g');
subplot(3, 1, 3)
plot(Power_linear_harmonic_1 * 1e3)
xlabel('Time [s]')
ylabel('Power [mW]')
title('Power with acceleration g');
```



```
% Mean Power
P_a_module_num_g_harmonic = eval(subs(P_a_module,{R_c K_i k_s C M R_L A_0 w},{Rc Ki
ks c m R_L_opt g w_harmonic}))*1e3;
```

```
Warning: The function sym/eval is deprecated and will be removed in a future release. Depending on the
usage, use subs or double instead.
```

```
disp(['Expected (analytical) power at: ',num2str(w_harmonic),' [rad/s] is:
',num2str(P_a_module_num_g_harmonic),' [mW]'])
```

```
Expected (analytical) power at: 618.7049 [rad/s] is: 47.8979 [mW]
```

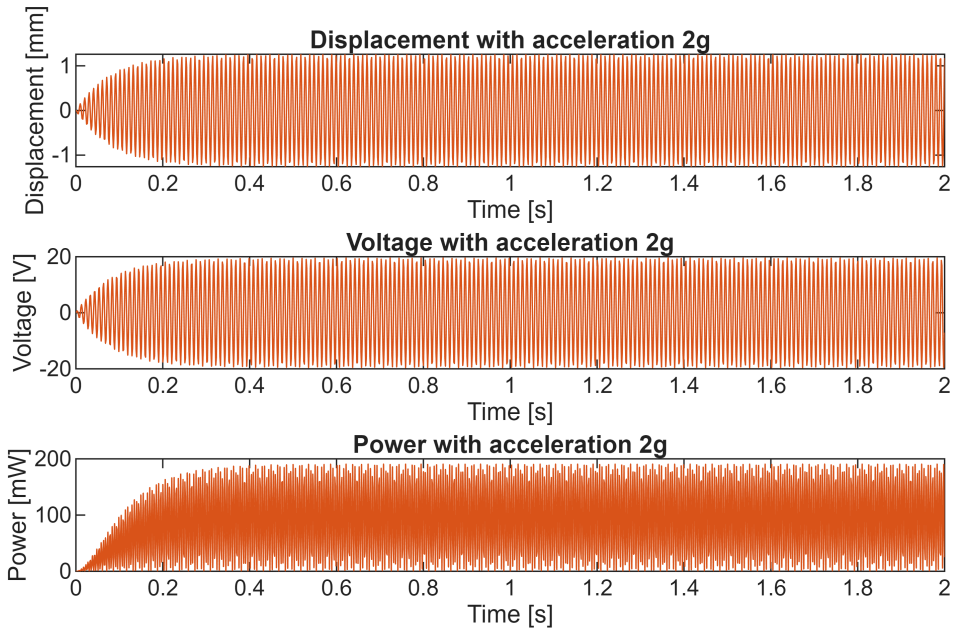
```
idx_i_1 = round(0.2*length(Power_linear_harmonic_1.Data));
P_mean_g = mean(Power_linear_harmonic_1.Data(idx_i_1:end))*2* 1e3; % [mW]
disp(['Computed (numerical) power at: ',num2str(w_harmonic),' [rad/s] is:
',num2str(P_mean_g),' [mW]'])
```

```
Computed (numerical) power at: 618.7049 [rad/s] is: 49.1186 [mW]
```

```
disp('HARMONIC ACCELERATION 2g')
```

```
HARMONIC ACCELERATION 2g
```

```
figure();
subplot(3, 1, 1)
plot(Displacement_linear_harmonic_2 * 1e3, 'Color','#D95319')
xlabel('Time [s]')
ylabel('Displacement [mm]')
title('Displacement with acceleration 2g');
subplot(3, 1, 2)
plot(Voltage_linear_harmonic_2, 'Color','#D95319')
xlabel('Time [s]')
ylabel('Voltage [V]')
title('Voltage with acceleration 2g');
subplot(3, 1, 3)
plot(Power_linear_harmonic_2 * 1e3, 'Color','#D95319')
xlabel('Time [s]')
ylabel('Power [mW]')
title('Power with acceleration 2g');
```




```
% Mean Power
P_a_module_num_2g_harmonic = eval(subs(P_a_module,{R_c K_i k_s C M R_L A_0 w},{Rc
Ki ks c m R_L_opt 2*g w_harmonic}))*1e3;
```

```
Warning: The function sym/eval is deprecated and will be removed in a future release. Depending on the
usage, use subs or double instead.
```

```
disp(['Expected (analytical) power at: ',num2str(w_harmonic),' [rad/s] is:
',num2str(P_a_module_num_2g_harmonic),' [mW]'])
```

```
Expected (analytical) power at: 618.7049 [rad/s] is: 191.5916 [mW]
```

```
idx_i_2 = round(0.2*length(Power_linear_harmonic_2.Data));
P_mean_2g = mean(Power_linear_harmonic_2.Data(idx_i_2:end))*2 * 1e3; % [mW]
disp(['Computed (numerical) power at: ',num2str(w_harmonic),' [rad/s] is:
',num2str(P_mean_2g),' [mW]'])
```

```
Computed (numerical) power at: 618.7049 [rad/s] is: 190.5061 [mW]
```

Conclusion:

Steady state solution matches the analytical solution

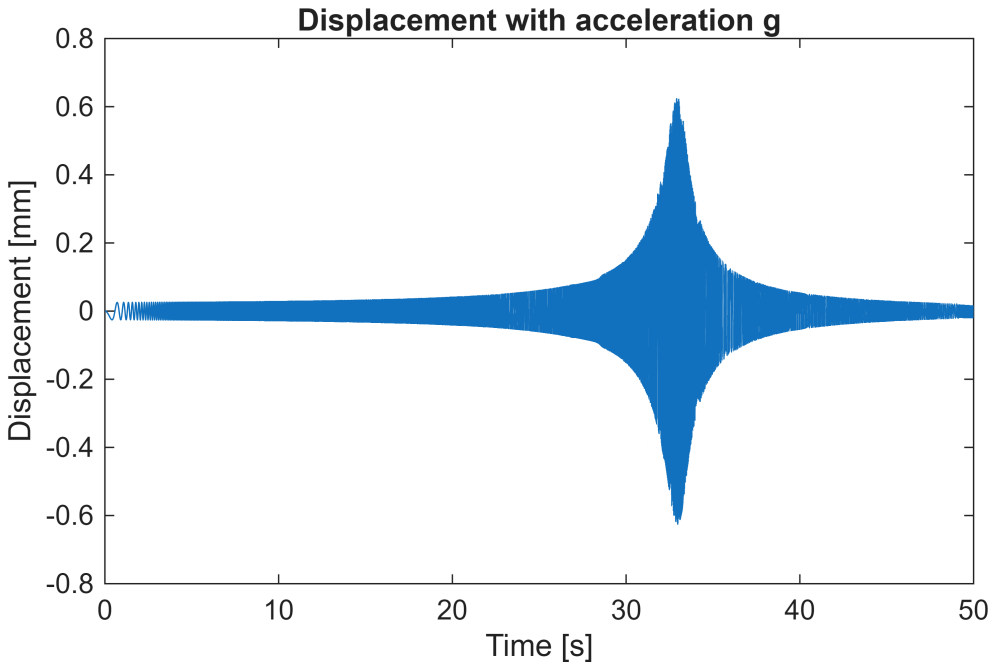
a1.2) Simulink: linear harvester CHIRP input

```
inputData = {g, 2*g};
size_inputData = length(inputData);
freq_ini_chirp = 0; % [Hz]
freq_fin_chirp = 150; % [Hz]
T_sim_chirp = 50; % [s]
```

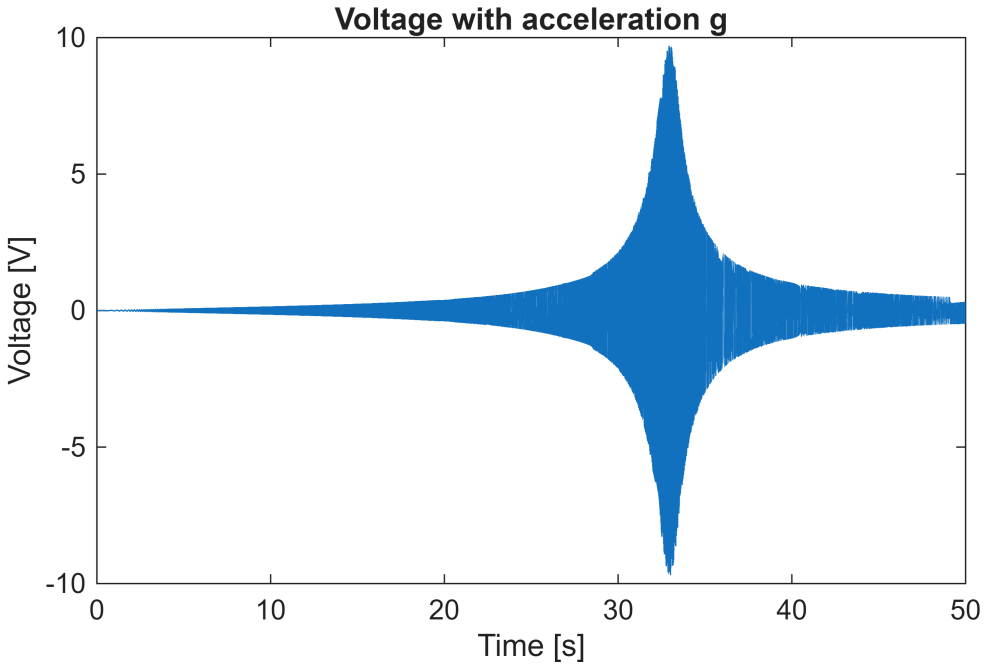
```
% Initialize a cell array to store output data
Displacement_linear = cell(1,size_inputData);
Voltage_linear = cell(1,size_inputData);
Power_linear = cell(1,size_inputData);

modelName = 'Linear_harvester_chirp_1_1';

for i = 1:size_inputData
    % Assign the current time series to the Simulink input
    assignin('base', 'inputAcceleration', inputData{i});

    % Run the simulation
    simOut = sim(modelName, 'ReturnWorkspaceOutputs', 'on');

    % Extract output data from the simulation
    Displacement_linear{i} = simOut.get('Displacement');
    Voltage_linear{i} = simOut.get('Voltage');
    Power_linear{i} = simOut.get('Power');

    % Save output with a unique name in the workspace
    assignin('base', ['Displacement_linear_' num2str(i)], Displacement_linear{i});
    assignin('base', ['Voltage_linear_' num2str(i)], Voltage_linear{i});
    assignin('base', ['Power_linear_' num2str(i)], Power_linear{i});
end

disp('CHIRP ACCELERATION g')
```

```
CHIRP ACCELERATION g
```

```
figure();
plot(Displacement_linear_1 * 1e3)
xlabel('Time [s]')
ylabel('Displacement [mm]')
title('Displacement with acceleration g');
```

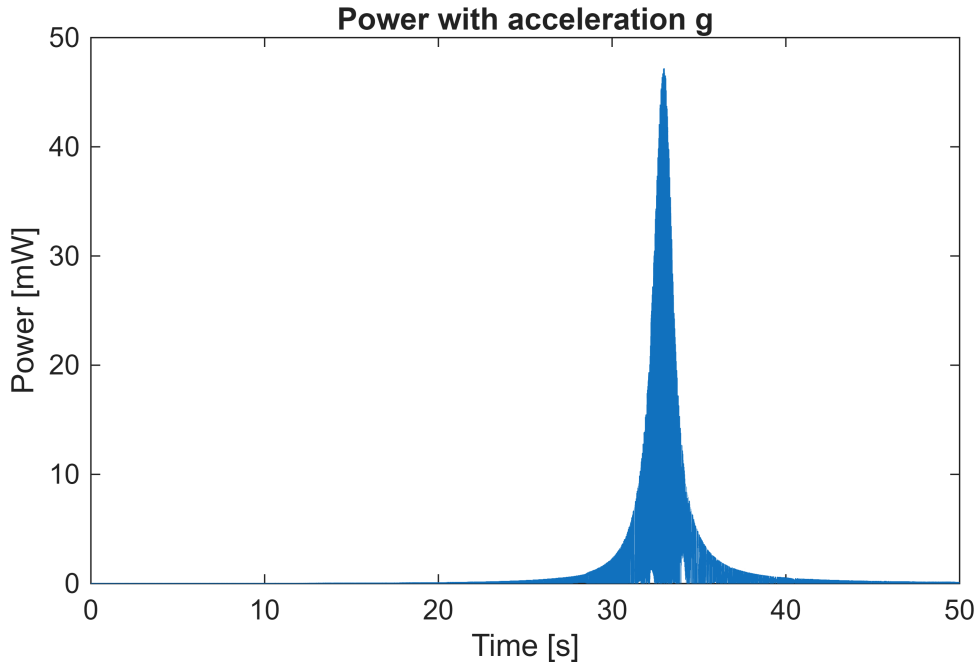



```
peak_value_Dis_linear_g = max(Displacement_linear_1);

figure();
plot(Voltage_linear_1)
xlabel('Time [s]')
ylabel('Voltage [V]')
title('Voltage with acceleration g');
```




```
peak_value_Vol_linear_g = max(Voltage_linear_1);

figure();
```

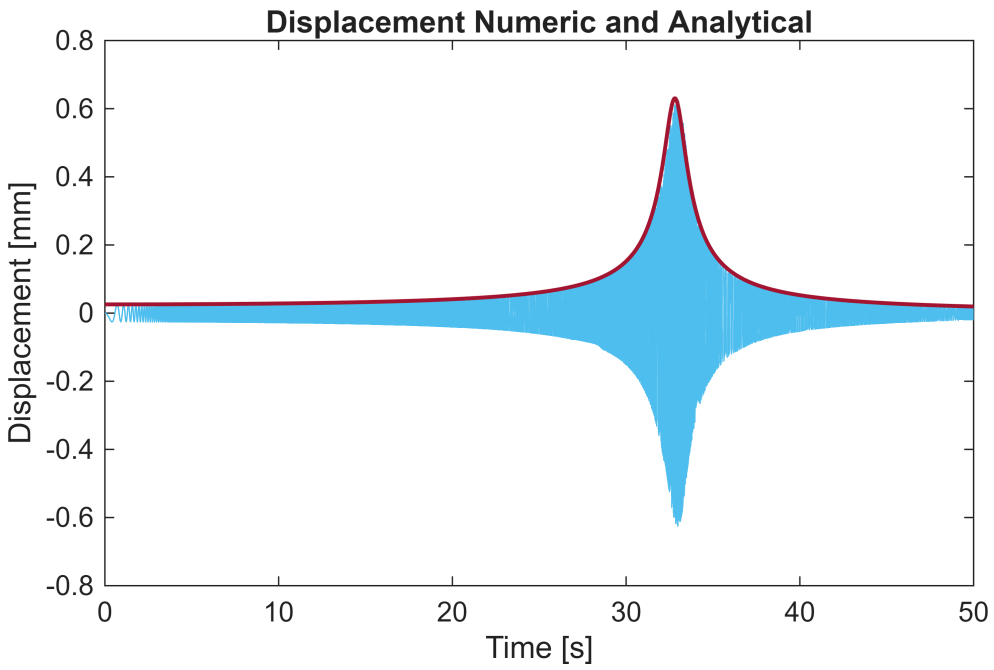
```
plot(Power_linear_1 *1e3)
xlabel('Time [s]')
ylabel('Power [mW]')
title('Power with acceleration g');
```




```
peak_value_Pow_linear_g = max(Power_linear_1);
```

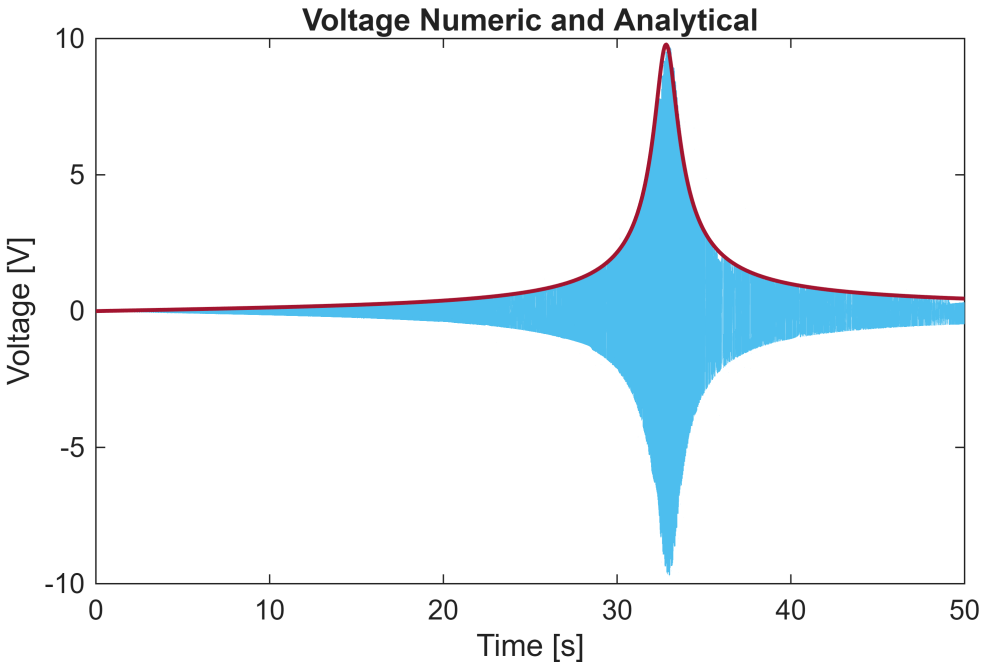
Overlap Displacement graphs:

```
% Data for the first plot
x1 = Displacement_linear_1.Time;
y1 = Displacement_linear_1.Data*1e3;
% Data for the second plot
x2 = w_sweep;
y2 = X_a_module_num_g*1e3;
x2_normalized = (x2-min(x2))/(max(x2)-min(x2))*(max(x1)-min(x1))+min(x1);
% Overlapped plot
figure();
plot(x1, y1, 'Color','#4DBEEE');
hold on;
plot(x2_normalized, y2,'Color','#A2142F','LineWidth', 1.4);
xlabel('Time [s]')
ylabel('Displacement [mm]')
title('Displacement Numeric and Analytical')
hold off;
```



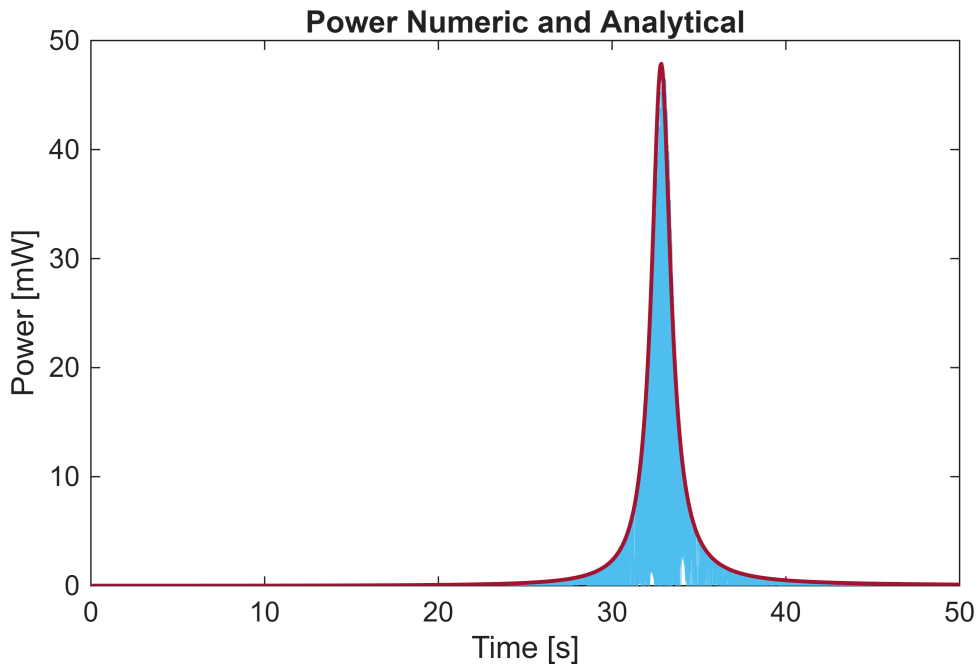
Overlap Voltage graphs:

```
% Data for the first plot
x1 = Voltage_linear_1.Time;
y1 = Voltage_linear_1.Data;
% Data for the second plot
x2 = w_sweep;
y2 = V_a_module_num_g;
x2_normalized = (x2-min(x2))/(max(x2)-min(x2))*(max(x1)-min(x1))+min(x1);
% Overlapped plot
figure();
plot(x1, y1, 'Color','#4DBEEE');
hold on;
plot(x2_normalized, y2,'Color','#A2142F','LineWidth', 1.4);
xlabel('Time [s]')
ylabel('Voltage [V]')
title('Voltage Numeric and Analytical')
hold off;
```



Overlap Power graphs:

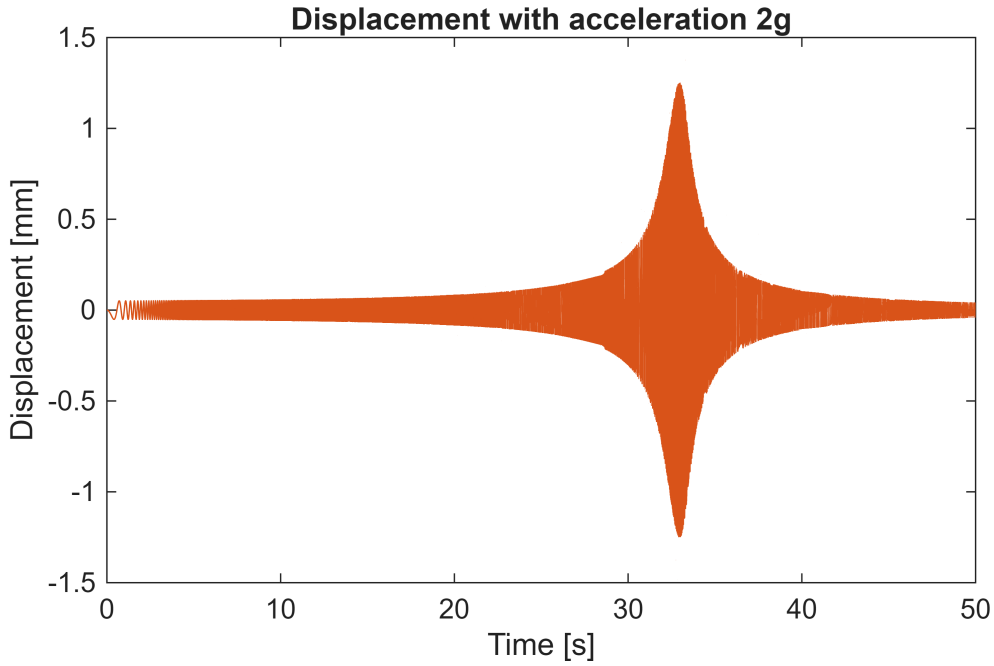
```
% Data for the first plot
x1 = Power_linear_1.Time;
y1 = Power_linear_1.Data*1e3; % FACTOR 1/2 for mean power
% Data for the second plot
x2 = w_sweep;
y2 = P_a_module_num_g*1e3;
x2_normalized = (x2-min(x2))/(max(x2)-min(x2))*(max(x1)-min(x1))+min(x1);
% Overlapped plot
figure();
plot(x1, y1, 'Color','#4DBEEE');
hold on;
plot(x2_normalized, y2,'Color','#A2142F','LineWidth', 1.4);
xlabel('Time [s]')
ylabel('Power [mW]')
title('Power Numeric and Analytical')
hold off;
```



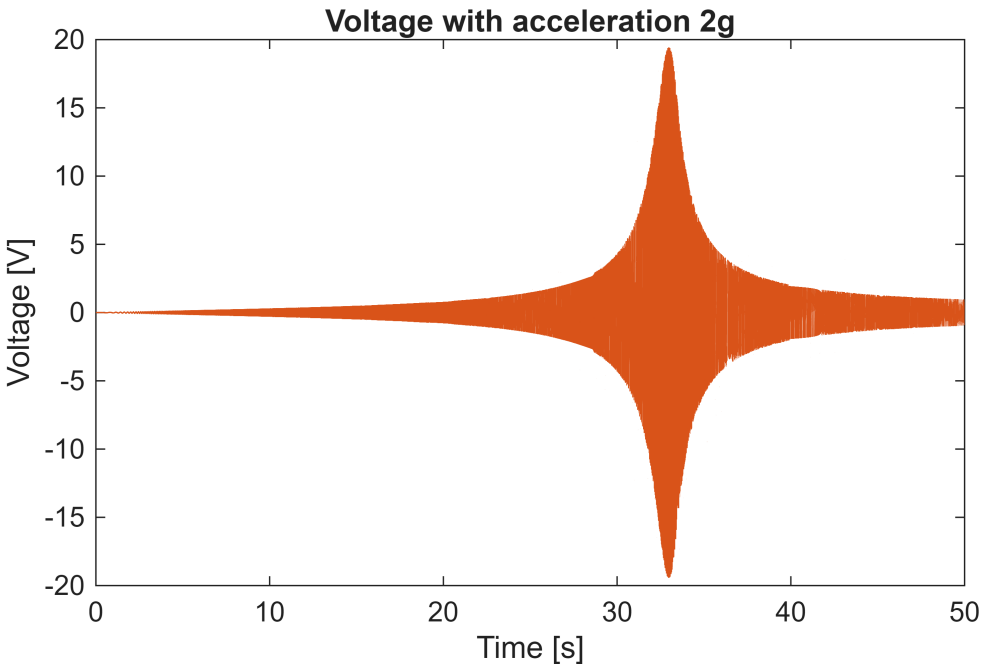

```
disp('CHIRP ACCELERATION 2g')
```

```
CHIRP ACCELERATION 2g
```

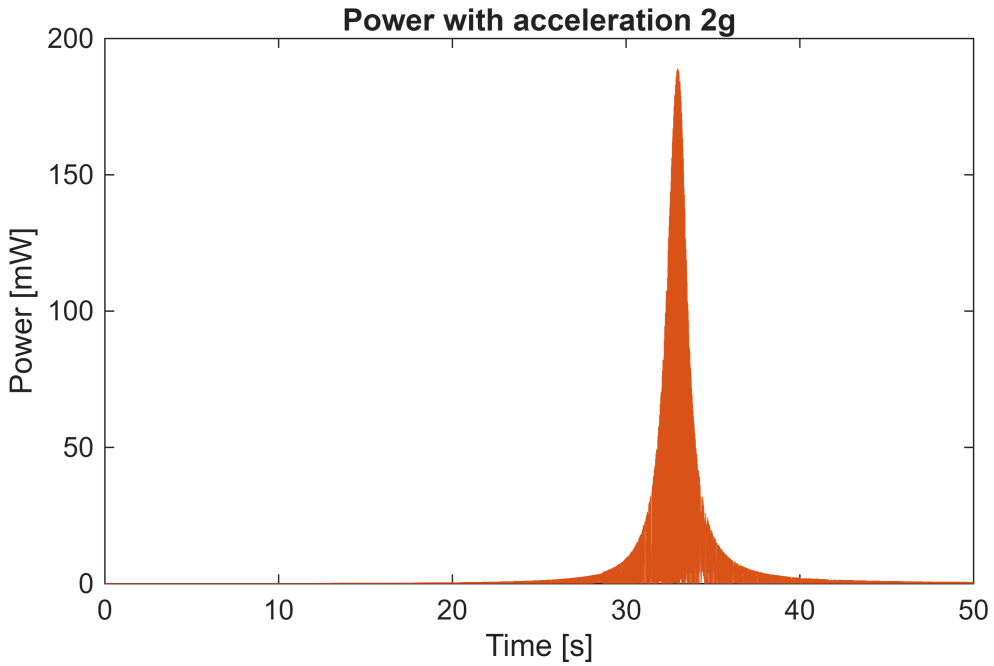
```
figure();
plot(Displacement_linear_2 *1e3, 'Color','#D95319')
xlabel('Time [s]')
ylabel('Displacement [mm]')
title('Displacement with acceleration 2g');
```



```
peak_value_Dis_linear_2g = max(Displacement_linear_2);


figure();
plot(Voltage_linear_2, 'Color','#D95319')
xlabel('Time [s]')
ylabel('Voltage [V]')
title('Voltage with acceleration 2g');
```




```
peak_value_Vol_linear_2g = max(Voltage_linear_2);


figure();
plot(Power_linear_2 *1e3, 'Color','#D95319')
xlabel('Time [s]')
ylabel('Power [mW]')
title('Power with acceleration 2g');
```




```
peak_value_Pow_linear_2g = max(Power_linear_2);

disp('-')
```

```
-
```

```
peak_ratio_Dis_linear = round(peak_value_Dis_linear_2g / peak_value_Dis_linear_g);
peak_ratio_Vol_linear = round(peak_value_Vol_linear_2g / peak_value_Vol_linear_g);
peak_ratio_Pow_linear = round(peak_value_Pow_linear_2g / peak_value_Pow_linear_g);

disp(['Peak ratio Displacement: ', num2str(peak_ratio_Dis_linear)])
```

```
Peak ratio Displacement: 2
```

```
disp(['Peak ratio Voltage: ', num2str(peak_ratio_Vol_linear)])
```

```
Peak ratio Voltage: 2
```

```
disp(['Peak ratio Power: ', num2str(peak_ratio_Pow_linear)])
```

```
Peak ratio Power: 4
```

a2) BISTABLE HAREVSTER

Assume **k** is the same between the two cases

```
syms Fs_bistable h
eq_bistable = Fs_bistable == -k*x + h*x^3
```

```
eq_bistable = Fs_bistable = h x^3 - k x
```

We have to define the **h** parameter:

```
syms x H
eq1 = 0 == -ks * x + H * x^3;

% Set the equilibroim point
x_eq = 0.5 * 1e-3; % [m]
eq1_x_eq = subs(eq1, {x}, {x_eq});
h = eval(solve(eq1_x_eq, H));
```

```
Warning: The function sym/eval is deprecated and will be removed in a future release. Depending on the
usage, use subs or double instead.
```

```
disp(['h = ',num2str(h), ' [N/m^3]'])
```

```
h = 142400000000 [N/m^3]
```

The reference equation of motion for the bistable harvester is:

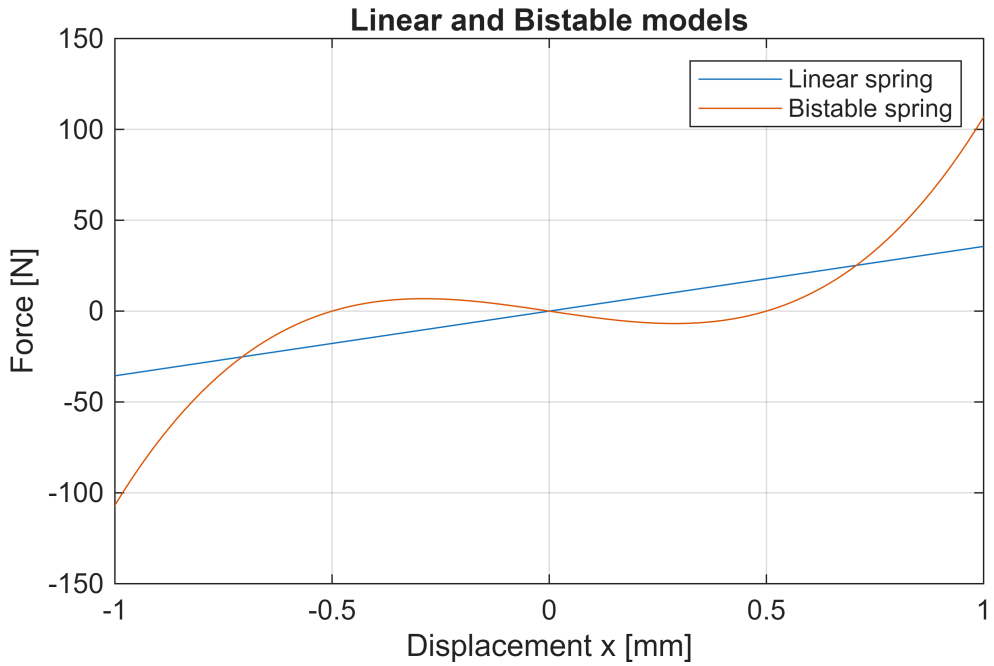
$$m\ddot{x} + c\dot{x} - kx + hx^3 + \frac{K_i^2}{R_c + R_L}\dot{x} = -m\ddot{y}$$

$$\ddot{x} = -\frac{c + \frac{K_i^2}{R_c + R_L}}{m}\dot{x} + \frac{k}{m}x - \frac{h}{m}x^3 - \ddot{y}$$

```
% Simulink parameter
coeff_x3 = h/m;
```

Visualize the non linear spring in respect to the linear case:

```
Fs_bistable_sweep = -ks * x_sweep + h * x_sweep.^3;

figure();
plot(x_sweep * 1e3, Fs_linear_sweep)
hold on
plot(x_sweep * 1e3, Fs_bistable_sweep)
xlim([-1 1])
xlabel('Displacement x [mm]')
ylabel('Force [N]')
title('Linear and Bistable models')
legend('Linear spring','Bistable spring')
grid on
hold off
```



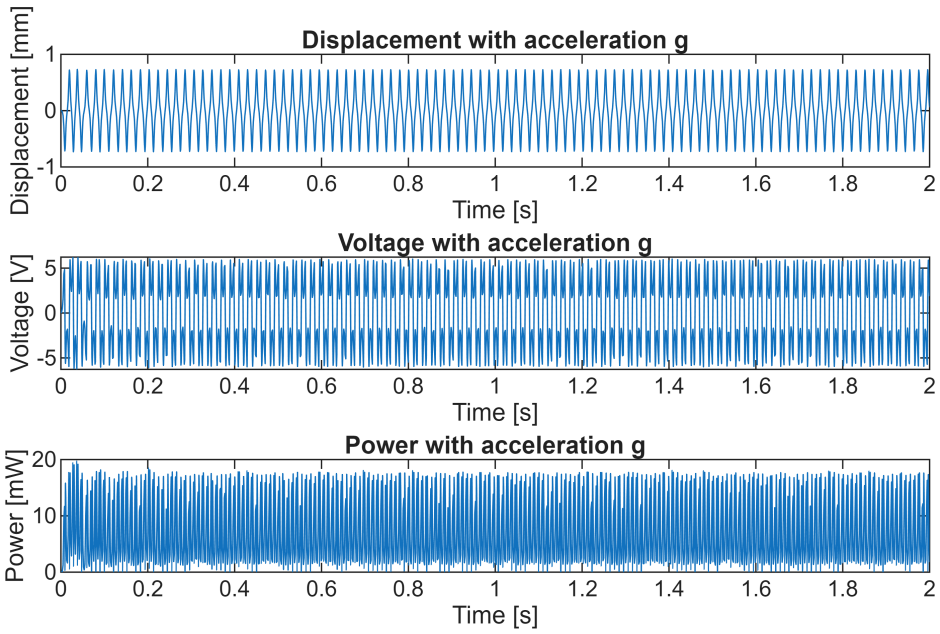
a2.1) Simulink: bistable harvester HARMONIC input

```
inputData_bistable_harmonic = {g, 2*g};
size_inputData_bistable_harmonic = length(inputData_bistable_harmonic);

% f_harmonic = f_e; % [Hz]
w_bistable_harmonic =305; % [rad/s]
T_sim_bistable_harmonic = 2; % [s]

% Initialize a cell array to store output data
Displacement_bistable_harmonic = cell(1,size_inputData_bistable_harmonic);
Voltage_bistable_harmonic = cell(1,size_inputData_bistable_harmonic);
Power_bistable_harmonic = cell(1,size_inputData_bistable_harmonic);

modelName = 'Bistable_harvester_harmonic_1_1';

for i = 1:size_inputData_bistable_harmonic
    % Assign the current time series to the Simulink input
    assignin('base', 'inputAcceleration', inputData_bistable_harmonic{i});

    % Run the simulation
    simOut = sim(modelName, 'ReturnWorkspaceOutputs', 'on');

    % Extract output data from the simulation
    Displacement_bistable_harmonic{i} = simOut.get('Displacement');
    Voltage_bistable_harmonic{i} = simOut.get('Voltage');
    Power_bistable_harmonic{i} = simOut.get('Power');
```

```
    % Save output with a unique name in the workspace
    assignin('base', ['Displacement_bistable_harmonic_' num2str(i)],
Displacement_bistable_harmonic{i});
    assignin('base', ['Voltage_bistable_harmonic_' num2str(i)],
Voltage_bistable_harmonic{i});
    assignin('base', ['Power_bistable_harmonic_' num2str(i)],
Power_bistable_harmonic{i});
end

disp('BISTABLE HARMONIC ACCELERATION g')
```

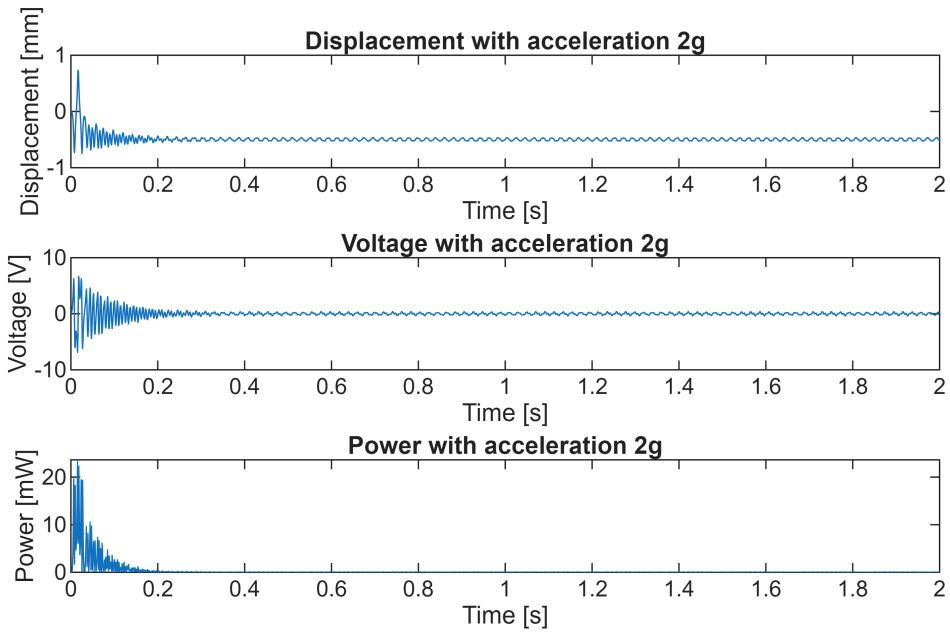
```
BISTABLE HARMONIC ACCELERATION g
```

```
figure();
subplot(3, 1, 1)
plot(Displacement_bistable_harmonic_1 * 1e3)
xlabel('Time [s]')
ylabel('Displacement [mm]')
title('Displacement with acceleration g');
subplot(3, 1, 2)
plot(Voltage_bistable_harmonic_1)
xlabel('Time [s]')
ylabel('Voltage [V]')
title('Voltage with acceleration g');
subplot(3, 1, 3)
plot(Power_bistable_harmonic_1 * 1e3)
xlabel('Time [s]')
ylabel('Power [mW]')
title('Power with acceleration g');
```




```
figure();
```

```
subplot(3, 1, 1)
plot(Displacement_bistable_harmonic_2 * 1e3)
xlabel('Time [s]')
ylabel('Displacement [mm]')
title('Displacement with acceleration 2g');
subplot(3, 1, 2)
plot(Voltage_bistable_harmonic_2)
xlabel('Time [s]')
ylabel('Voltage [V]')
title('Voltage with acceleration 2g');
subplot(3, 1, 3)
plot(Power_bistable_harmonic_2 * 1e3)
xlabel('Time [s]')
ylabel('Power [mW]')
title('Power with acceleration 2g');
```



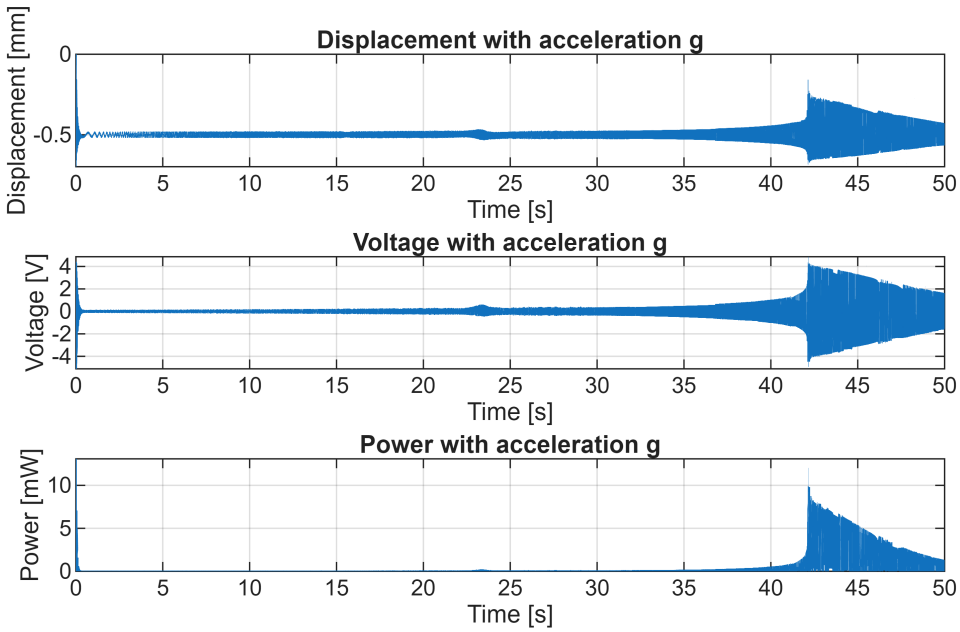
a2.2) Simulink: bistable harvester CHIRP input

```
inputData_bistable = {g, 2*g};
size_inputData_bistable = length(inputData_bistable);
freq_ini_chirp_bistable = 0; % [Hz]
freq_fin_chirp_bistable = 150; % [Hz]
T_sim_chirp_bistable = 50; % [s]

% Initialize a cell array to store output data
Displacement_bistable = cell(1,size_inputData_bistable);
Voltage_bistable = cell(1,size_inputData_bistable);
Power_bistable = cell(1,size_inputData_bistable);

modelName = 'Bistable_harvester_chirp_1_1';
```

```
for i = 1:size_inputData_bistable
    % Assign the current time series to the Simulink input
    assignin('base', 'inputAcceleration', inputData_bistable{i});

    % Run the simulation
    simOut = sim(modelName, 'ReturnWorkspaceOutputs', 'on');

    % Extract output data from the simulation
    Displacement_bistable{i} = simOut.get('Displacement');
    Voltage_bistable{i} = simOut.get('Voltage');
    Power_bistable{i} = simOut.get('Power');

    % Save output with a unique name in the workspace
    assignin('base', ['Displacement_bistable_' num2str(i)],
Displacement_bistable{i});
    assignin('base', ['Voltage_bistable_' num2str(i)], Voltage_bistable{i});
    assignin('base', ['Power_bistable_' num2str(i)], Power_bistable{i});
end

disp('BISTABLE CHIRP ACCELERATION g')
```

```
BISTABLE CHIRP ACCELERATION g
```

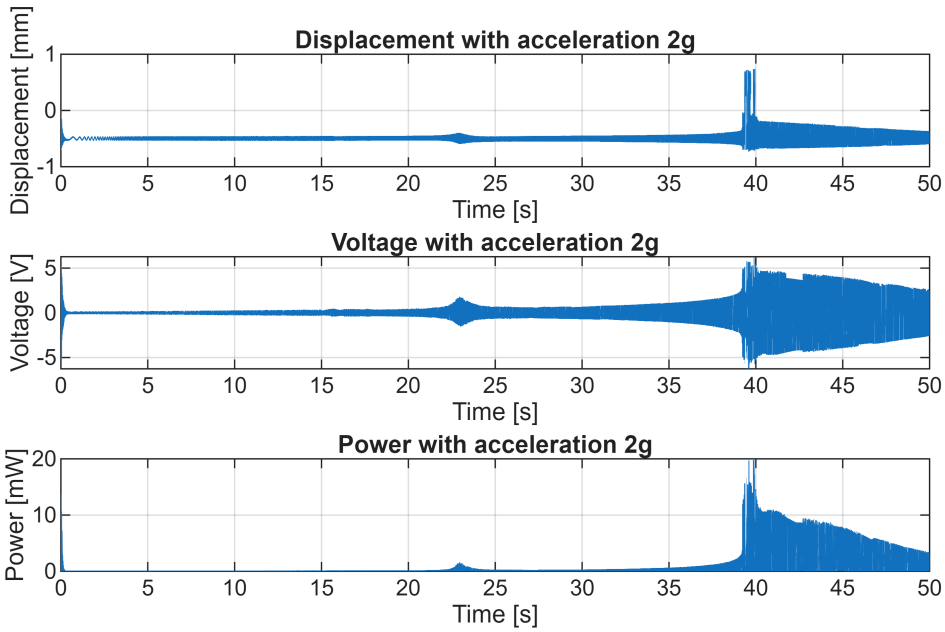
```
figure();
subplot(3, 1, 1)
plot(Displacement_bistable_1 * 1e3)
xlabel('Time [s]')
ylabel('Displacement [mm]')
title('Displacement with acceleration g');
grid on
subplot(3, 1, 2)
plot(Voltage_bistable_1)
xlabel('Time [s]')
ylabel('Voltage [V]')
title('Voltage with acceleration g');
grid on
subplot(3, 1, 3)
plot(Power_bistable_1 * 1e3)
xlabel('Time [s]')
ylabel('Power [mW]')
title('Power with acceleration g');
grid on
```




```
disp('BISTABLE CHIRP ACCELERATION 2g')
```

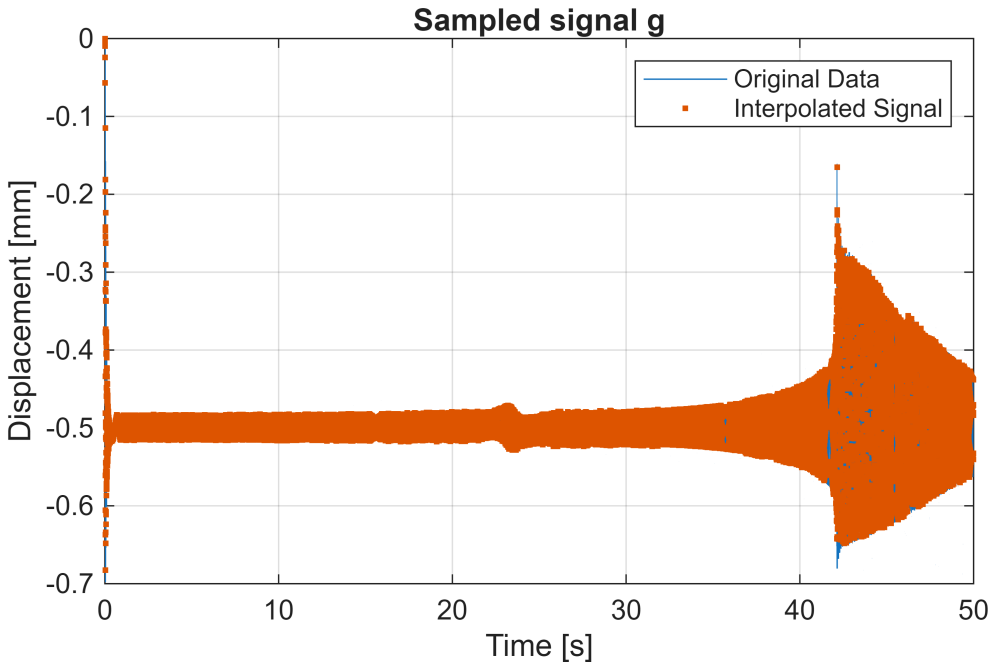
```
BISTABLE CHIRP ACCELERATION 2g
```

```
figure();
subplot(3, 1, 1)
plot(Displacement_bistable_2 * 1e3)
xlabel('Time [s]')
ylabel('Displacement [mm]')
title('Displacement with acceleration 2g');
grid on
subplot(3, 1, 2)
plot(Voltage_bistable_2)
xlabel('Time [s]')
ylabel('Voltage [V]')
title('Voltage with acceleration 2g');
grid on
subplot(3, 1, 3)
plot(Power_bistable_2 * 1e3)
xlabel('Time [s]')
ylabel('Power [mW]')
title('Power with acceleration 2g');
grid on

ax = findall(gcf, 'type', 'axes'); % linkaxes
linkaxes(ax, 'x');
```



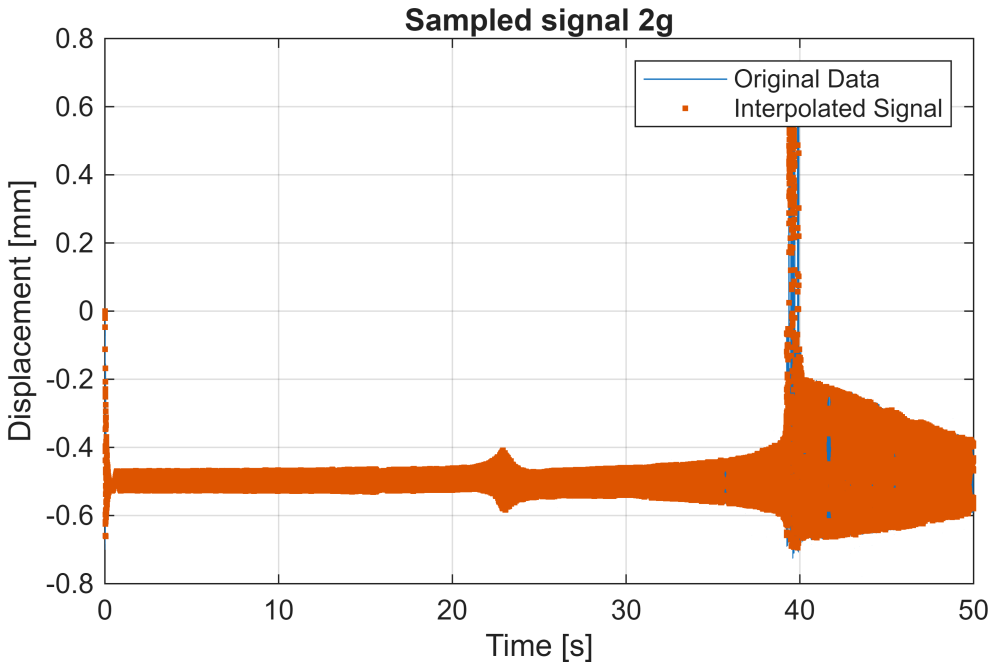
- Displacement chirp g

```
% resample the signal in order to have fixed step
time_d_1 = Displacement_bistable_1.Time;
data_d_1 = Displacement_bistable_1.Data;

% Follow Nyquist theorem to not lose information: fs >= 2 * fmax
fs_d_1 = 5*freq_fin_chirp_bistable;
dt_d_1 = 1/fs_d_1;

% Uniform the time vector
time_d_1_s = (time_d_1(1):dt_d_1:time_d_1(end))'; % Uniform time vector

% Interpolate the signal to the new time vector
data_d_1_s = interp1(time_d_1, data_d_1, time_d_1_s, 'linear');
len_d_1_s = length(time_d_1_s);

% Plot sampled signal over the original
figure;
plot(time_d_1, data_d_1*1e3, 'DisplayName', 'Original Data'); % Original points
hold on;
plot(time_d_1_s, data_d_1_s*1e3, '.', 'DisplayName', 'Interpolated Signal'); %
Resampled signal
xlabel('Time [s]');
ylabel('Displacement [mm]');
title('Sampled signal g')
legend;
grid on;
```



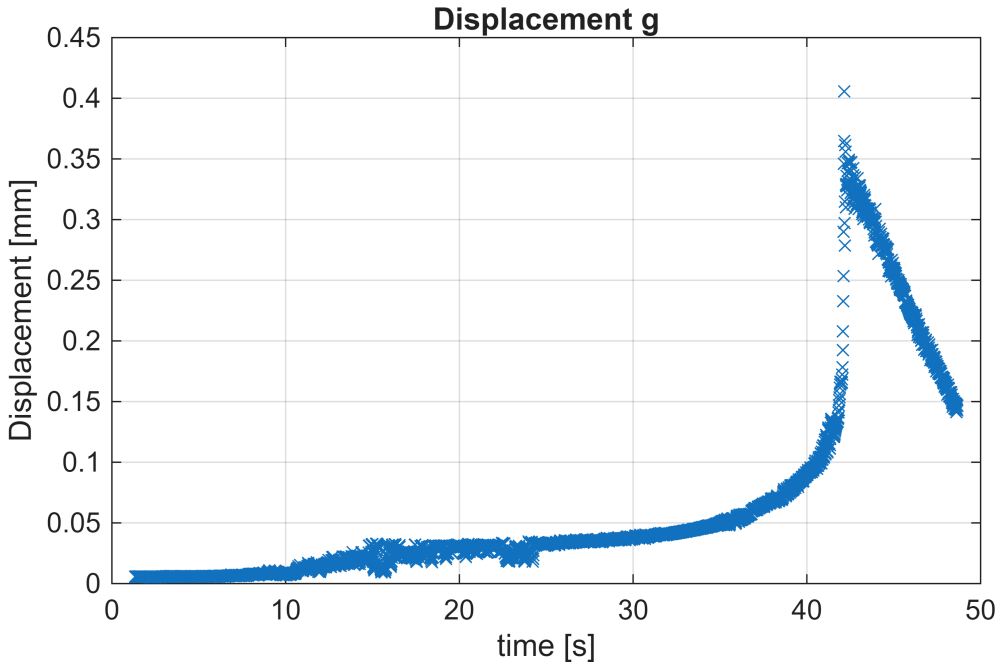
- Displacement chirp 2g

```
% resample the signal in order to have fixed step
time_d_2 = Displacement_bistable_2.Time;
data_d_2 = Displacement_bistable_2.Data;

% Follow Nyquist theorem to not lose information: fs >= 2 * fmax
fs_d_2 = 5*freq_fin_chirp_bistable;
dt_d_2  = 1/fs_d_2 ;

% Uniform the time vector
time_d_2_s = (time_d_2(1):dt_d_2:time_d_2(end))'; % Uniform time vector

% Interpolate the signal to the new time vector
data_d_2_s = interp1(time_d_2, data_d_2, time_d_2_s, 'linear');
len_d_2_s = length(time_d_2_s);

% Plot sampled signal over the original
figure;
plot(time_d_2, data_d_2*1e3, 'DisplayName', 'Original Data'); % Original points
hold on;
plot(time_d_2_s, data_d_2_s*1e3, '.', 'DisplayName', 'Interpolated Signal'); %
Resampled signal
xlabel('Time [s]');
ylabel('Displacement [mm]');
title('Sampled signal 2g')
legend;
grid on;
```



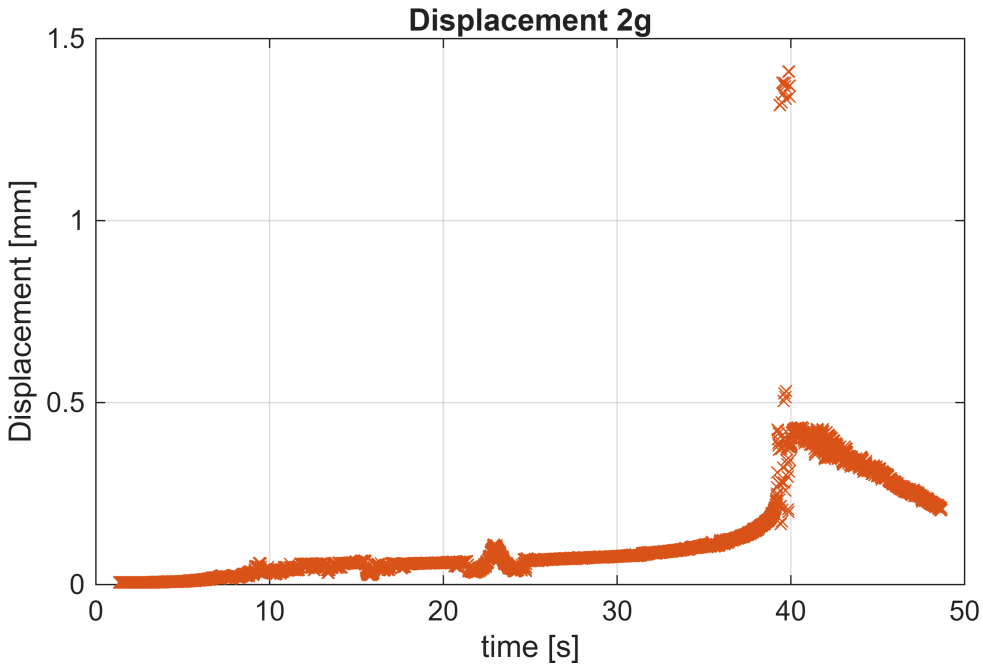

```
% Displacement mobile window g
v_1 = 1000:10:len_d_1_s-1000;
d1_bistable_chirp = zeros(length(v_1), 1);
% 1:length(v_1) 2930
for i = 1:length(v_1)
    ti = time_d_1_s(v_1(i));
    fi = freq_fin_chirp_bistable/T_sim_chirp_bistable * time_d_1_s(v_1(i));
    Ti = 1/fi;
    wi = 5*Ti;
    if wi/2 < time_d_1_s(v_1(i))
        idx_tmp = find( time_d_1_s > ti - wi/2 & time_d_1_s < ti + wi/2 );
        time_tmp = time_d_1_s(idx_tmp);
        data_tmp = data_d_1_s(idx_tmp);
    end
    [value_peaks_str, loc_peaks_str] = findpeaks(data_tmp);
    [value_peaks_inv, loc_peaks_inv] = findpeaks(-data_tmp);

    if sum(value_peaks_str < 0) ~= length(value_peaks_str) && sum(value_peaks_str <
0) ~= 0
        max_str = max(abs(value_peaks_str));
        max_inv = max(abs(value_peaks_inv));
        [v_max, idx_max] = max([max_str max_inv]);
        if idx_max == 1
            v_min = min(-value_peaks_inv);
        elseif idx_max == 2
            v_min = min(-value_peaks_str);
        else
            disp('Error')
        end
```

```
        d1_bistable_chirp(i) = v_max + (- v_min);
    % if monostable
    else
        d1_bistable_chirp(i) = abs((mean(value_peaks_str) + mean(value_peaks_inv)));
    end
end

figure();
plot(time_d_1_s(v_1), d1_bistable_chirp*1e3, 'x')
xlabel('time [s]')
ylabel('Displacement [mm]')
title('Displacement g')
grid on
```




```
% Displacement mobile window 2g
v_2 = 1000:10:len_d_2_s-1000;
d2_bistable_chirp = zeros(length(v_2), 1);
% 1:length(v_2) 2930
for i = 1:length(v_2)
    ti = time_d_2_s(v_2(i));
    fi = freq_fin_chirp_bistable/T_sim_chirp_bistable * time_d_2_s(v_2(i));
    Ti = 1/fi;
    wi = 5*Ti;
    if wi/2 < time_d_2_s(v_2(i))
        idx_tmp = find( time_d_2_s > ti - wi/2 & time_d_2_s < ti + wi/2 );
        time_tmp = time_d_2_s(idx_tmp);
        data_tmp = data_d_2_s(idx_tmp);
    end
```

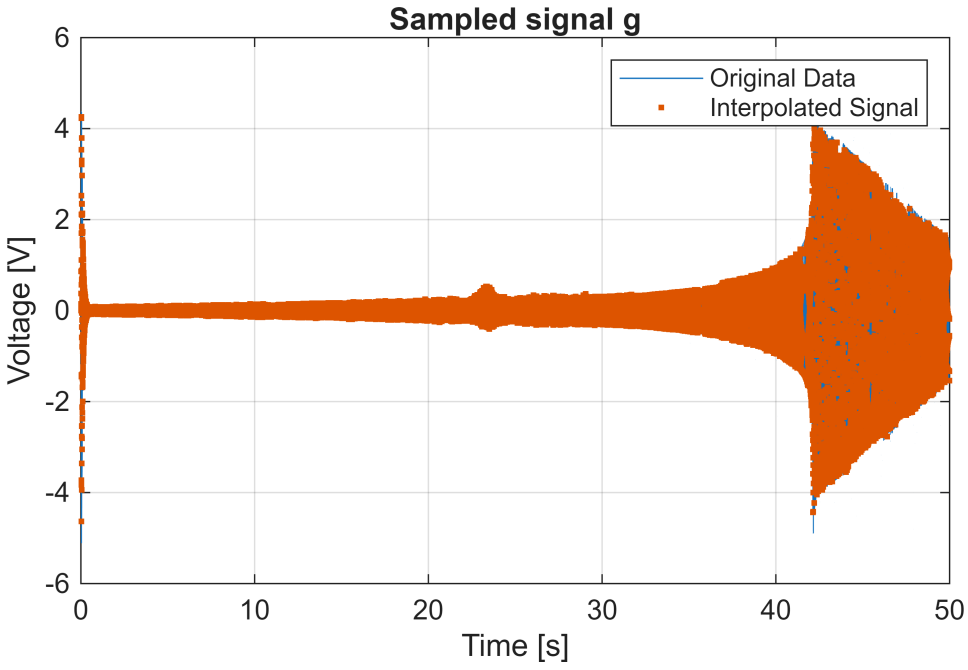
```
    [value_peaks_str, loc_peaks_str] = findpeaks(data_tmp);
    [value_peaks_inv, loc_peaks_inv] = findpeaks(-data_tmp);

    if sum(value_peaks_str < 0) ~= length(value_peaks_str) && sum(value_peaks_str <
0) ~= 0
        max_str = max(abs(value_peaks_str));
        max_inv = max(abs(value_peaks_inv));
        [v_max, idx_max] = max([max_str max_inv]);
        if idx_max == 1
            v_min = min(-value_peaks_inv);
        elseif idx_max == 2
            v_min = min(-value_peaks_str);
        else
            disp('Error')
        end
        d2_bistable_chirp(i) = v_max + (- v_min);
    % if monostable
    else
        d2_bistable_chirp(i) = abs((mean(value_peaks_str) + mean(value_peaks_inv)));
    end
end

figure();
plot(time_d_2_s(v_2), d2_bistable_chirp*1e3,'x','Color','#D95319')
xlabel('time [s]')
ylabel('Displacement [mm]')
title('Displacement 2g')
grid on
```




```
figure();
```

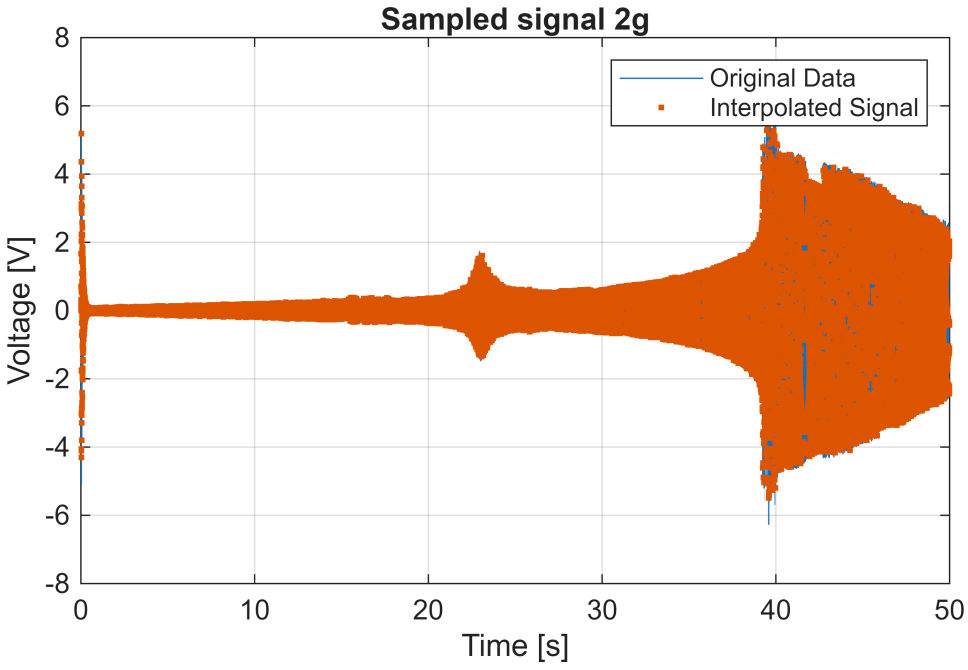
- Voltage chirp g

```
% resample the signal in order to have fixed step
time_v_1 = Voltage_bistable_1.Time;
data_v_1 = Voltage_bistable_1.Data;

% Follow Nyquist theorem to not lose information: fs >= 2 * fmax
fs_v_1 = 5*freq_fin_chirp_bistable;
dt_v_1 = 1/fs_v_1;

% Uniform the time vector
time_v_1_s = (time_v_1(1):dt_d_1:time_v_1(end))'; % Uniform time vector

% Interpolate the signal to the new time vector
data_v_1_s = interp1(time_v_1, data_v_1, time_v_1_s, 'linear');
len_v_1_s = length(time_v_1_s);

% Plot sampled signal over the original
figure;
plot(time_v_1, data_v_1, 'DisplayName', 'Original Data'); % Original points
hold on;
plot(time_v_1_s, data_v_1_s, '.', 'DisplayName', 'Interpolated Signal'); %
Resampled signal
xlabel('Time [s]');
ylabel('Voltage [V]');
title('Sampled signal g')
legend;
grid on;
```

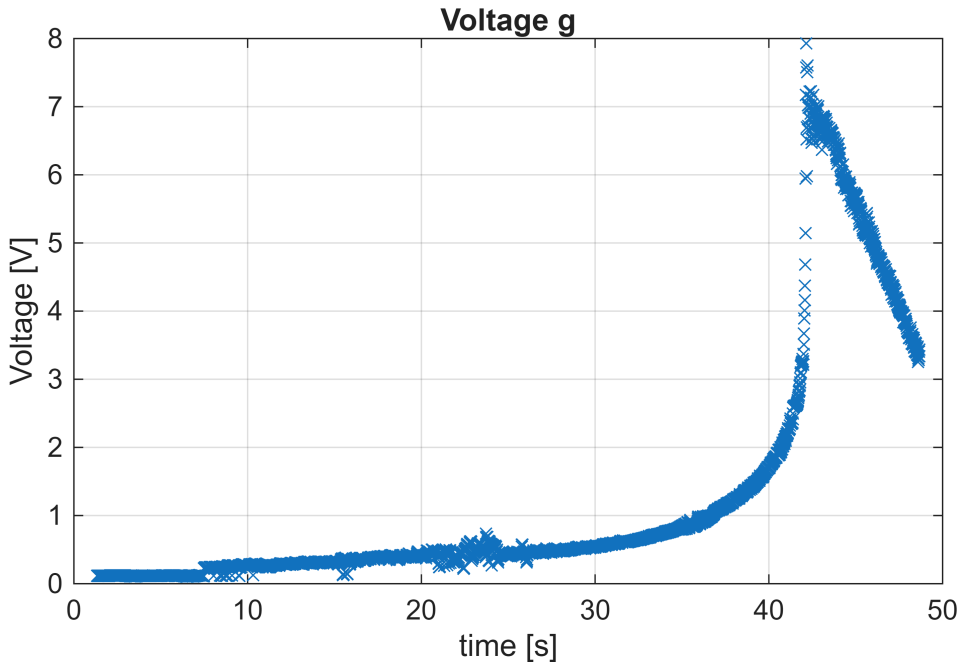


- Voltage chirp 2g

```
% resample the signal in order to have fixed step
time_v_2 = Voltage_bistable_2.Time;
data_v_2 = Voltage_bistable_2.Data;

% Follow Nyquist theorem to not lose information: fs >= 2 * fmax
fs_v_2 = 5*freq_fin_chirp_bistable;
dt_v_2 = 1/fs_v_2;

% Uniform the time vector
time_v_2_s = (time_v_2(1):dt_d_2:time_v_2(end))'; % Uniform time vector

% Interpolate the signal to the new time vector
data_v_2_s = interp1(time_v_2, data_v_2, time_v_2_s, 'linear');
len_v_2_s = length(time_v_2_s);

% Plot sampled signal over the original
figure;
plot(time_v_2, data_v_2, 'DisplayName', 'Original Data'); % Original points
hold on;
plot(time_v_2_s, data_v_2_s, '.', 'DisplayName', 'Interpolated Signal'); %
Resampled signal
xlabel('Time [s]');
ylabel('Voltage [V]');
title('Sampled signal 2g')
legend;
grid on;
```

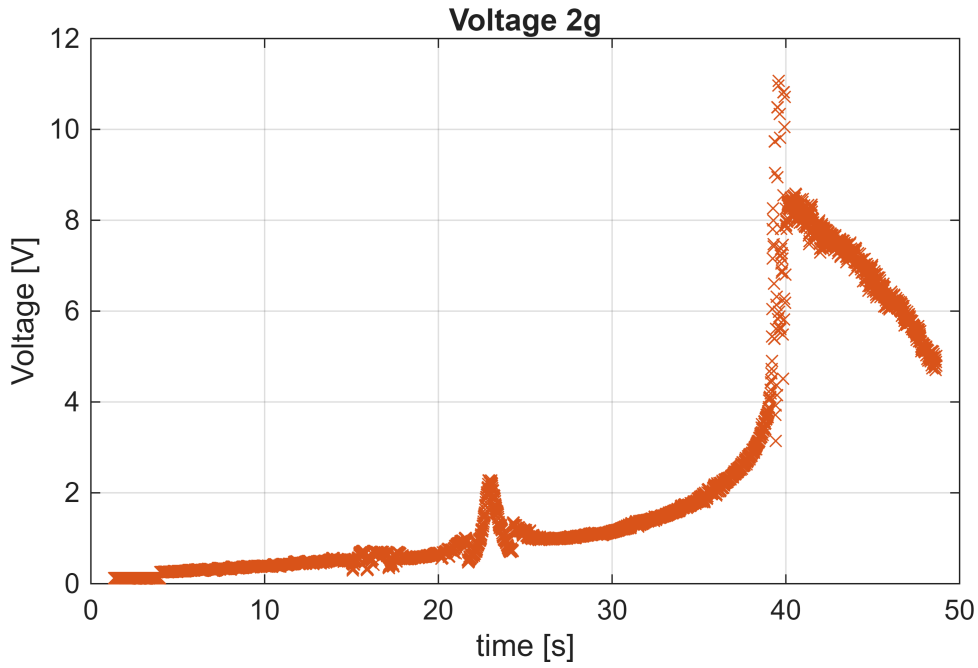


```
% Voltage mobile window g1
v_V_1 = 1000:10:len_v_1_s-1000;
v1_bistable_chirp = zeros(length(v_V_1), 1);
% 1:length(v_V_1) 2930
for i = 1:length(v_V_1)
    ti = time_v_1_s(v_V_1(i));
    fi = freq_fin_chirp_bistable/T_sim_chirp_bistable * time_v_1_s(v_V_1(i));
    Ti = 1/fi;
    wi = 5*Ti;
    if wi/2 < time_v_1_s(v_V_1(i))
        idx_tmp = find( time_v_1_s > ti - wi/2 & time_v_1_s < ti + wi/2 );
        time_tmp = time_v_1_s(idx_tmp);
        data_tmp = data_v_1_s(idx_tmp);
    end
    [value_peaks_str, loc_peaks_str] = findpeaks(data_tmp);
    [value_peaks_inv, loc_peaks_inv] = findpeaks(-data_tmp);

    if sum(value_peaks_str < 0) ~= length(value_peaks_str) && sum(value_peaks_str <
0) ~= 0
        max_str = max(abs(value_peaks_str));
        max_inv = max(abs(value_peaks_inv));
        [v_max, idx_max] = max([max_str max_inv]);
        if idx_max == 1
            v_min = min(-value_peaks_inv);
        elseif idx_max == 2
            v_min = min(-value_peaks_str);
        else
            disp('Error')
        end
        v1_bistable_chirp(i) = v_max + (- v_min);
    % if monostable
    else
        v1_bistable_chirp(i) = abs((mean(value_peaks_str) + mean(value_peaks_inv)));
    end
end

figure();
plot(time_v_1_s(v_V_1), v1_bistable_chirp, 'x')
xlabel('time [s]')
ylabel('Voltage [V]')
title('Voltage g')
grid on
```




```
% Voltage mobile window g2
v_V_2 = 1000:10:len_v_2_s-1000;
v2_bistable_chirp = zeros(length(v_V_2), 1);
% 1:length(v_V_2) 2930
for i = 1:length(v_V_2)
    ti = time_v_2_s(v_V_2(i));
    fi = freq_fin_chirp_bistable/T_sim_chirp_bistable * time_v_2_s(v_V_2(i));
    Ti = 1/fi;
    wi = 5*Ti;
    if wi/2 < time_v_2_s(v_V_2(i))
        idx_tmp = find( time_v_2_s > ti - wi/2 & time_v_2_s < ti + wi/2 );
        time_tmp = time_v_2_s(idx_tmp);
        data_tmp = data_v_2_s(idx_tmp);
    end
    [value_peaks_str, loc_peaks_str] = findpeaks(data_tmp);
    [value_peaks_inv, loc_peaks_inv] = findpeaks(-data_tmp);

    if sum(value_peaks_str < 0) ~= length(value_peaks_str) && sum(value_peaks_str <
0) ~= 0
        max_str = max(abs(value_peaks_str));
        max_inv = max(abs(value_peaks_inv));
        [v_max, idx_max] = max([max_str max_inv]);
        if idx_max == 1
            v_min = min(-value_peaks_inv);
        elseif idx_max == 2
            v_min = min(-value_peaks_str);
        else
            disp('Error')
        end
```

```
        v2_bistable_chirp(i) = v_max + (- v_min);
    % if monostable
    else
        v2_bistable_chirp(i) = abs((mean(value_peaks_str) + mean(value_peaks_inv)));
    end
end
figure();
plot(time_v_2_s(v_V_2), v2_bistable_chirp,'x','Color','#D95319')
xlabel('time [s]')
ylabel('Voltage [V]')
title('Voltage 2g')
grid on
```

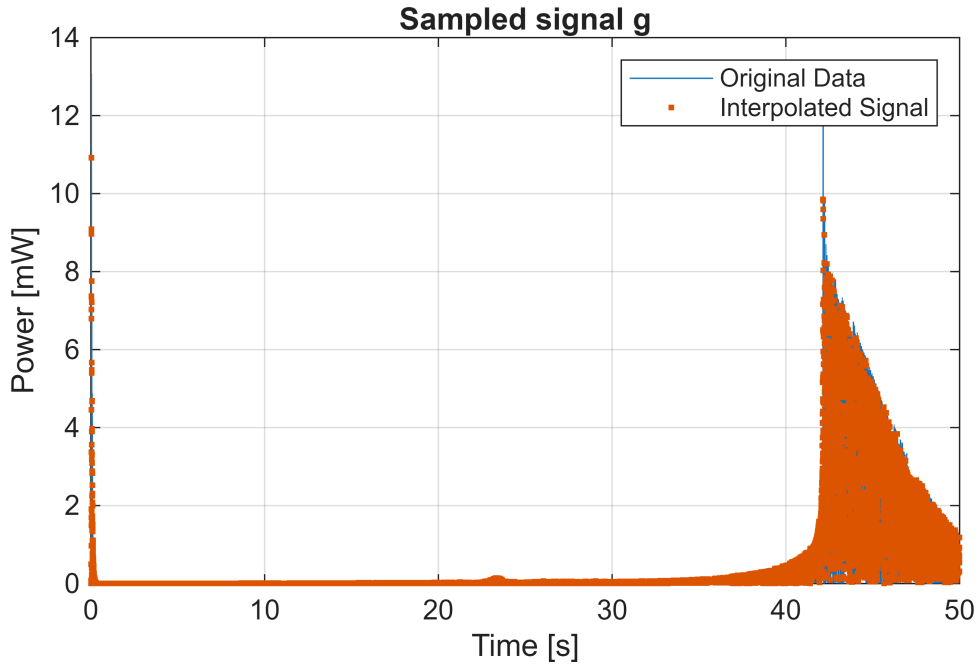



```
figure();
```

- Power chirp g

```
% resample the signal in order to have fixed step
time_p_1 = Power_bistable_1.Time;
data_p_1 = Power_bistable_1.Data;

% Follow Nyquist theorem to not lose information: fs >= 2 * fmax
fs_p_1 = 5*freq_fin_chirp_bistable;
dt_p_1 = 1/fs_p_1;

% Uniform the time vector
time_p_1_s = (time_p_1(1):dt_p_1:time_p_1(end))'; % Uniform time vector

% Interpolate the signal to the new time vector
data_p_1_s = interp1(time_p_1, data_p_1, time_p_1_s, 'linear');
```

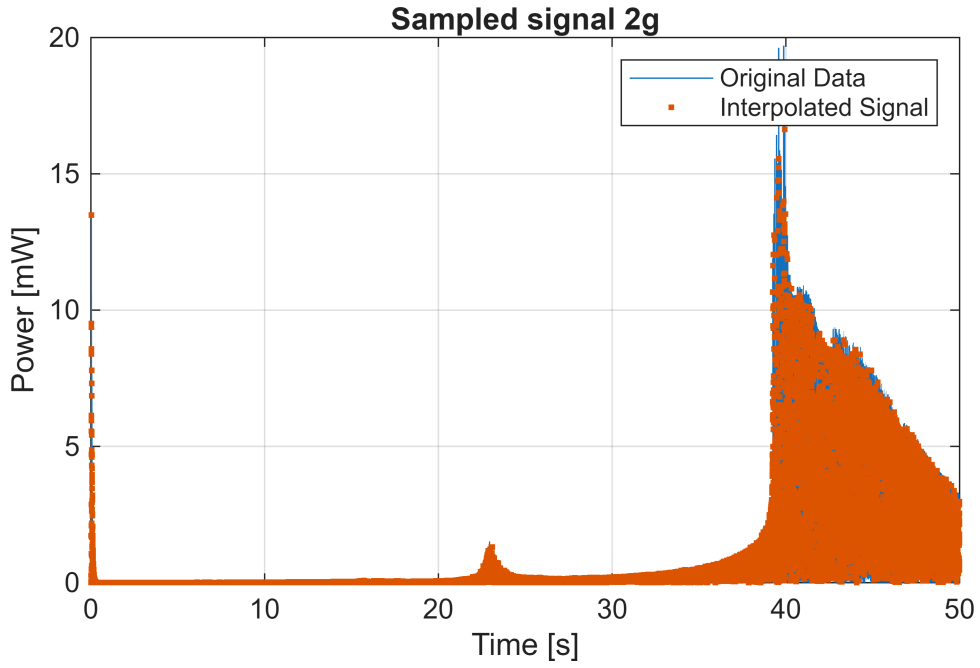
```
len_p_1_s = length(time_p_1_s);

% Plot sampled signal over the original
figure;
plot(time_p_1, data_p_1 *1e3, 'DisplayName', 'Original Data'); % Original points
hold on;
plot(time_p_1_s, data_p_1_s *1e3, '.', 'DisplayName', 'Interpolated Signal'); %
Resampled signal
xlabel('Time [s]');
ylabel('Power [mW]');
title('Sampled signal g')
legend;
grid on;
```



- Power chirp 2g

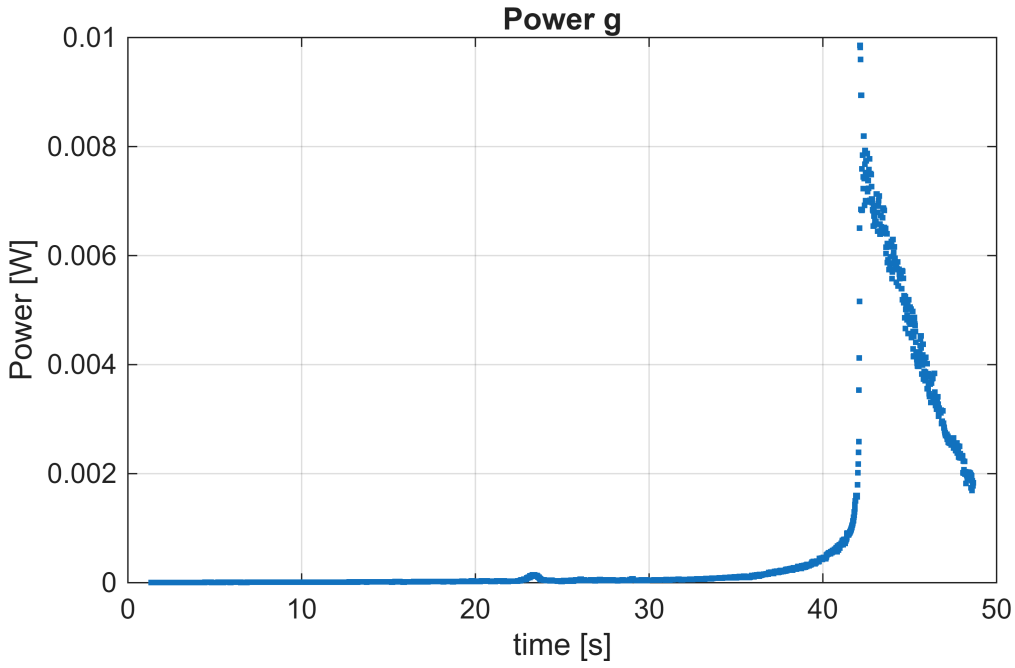
```
% resample the signal in order to have fixed step
time_p_2 = Power_bistable_2.Time;
data_p_2 = Power_bistable_2.Data;

% Follow Nyquist theorem to not lose information: fs >= 2 * fmax
fs_p_2 = 5*freq_fin_chirp_bistable;
dt_p_2 = 1/fs_p_2;

% Uniform the time vector
time_p_2_s = (time_p_2(1):dt_p_2:time_p_2(end))'; % Uniform time vector

% Interpolate the signal to the new time vector
data_p_2_s = interp1(time_p_2, data_p_2, time_p_2_s, 'linear');
len_p_2_s = length(time_p_2_s);
```

```
% Plot sampled signal over the original
figure;
plot(time_p_2, data_p_2 *1e3, 'DisplayName', 'Original Data'); % Original points
hold on;
plot(time_p_2_s, data_p_2_s *1e3, '.', 'DisplayName', 'Interpolated Signal'); %
Resampled signal
xlabel('Time [s]');
ylabel('Power [mW]');
title('Sampled signal 2g')
legend;
grid on;
```



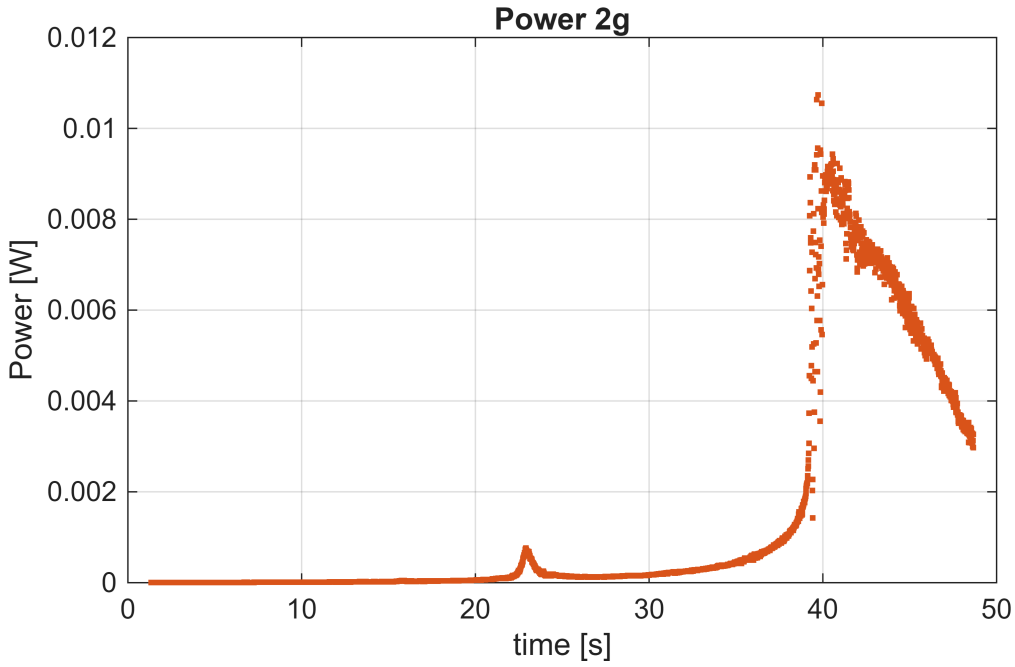

```
% Power mobile window g
% mean value of the peaks in a mobile window
v_P_1 = 1000:10:len_v_1_s-1000;
p1_bistable_chirp = zeros(length(v_P_1), 1);
% 1:length(v_P_1) 2930
for i = 1:length(v_P_1)
    ti = time_p_1_s(v_P_1(i));
    fi = freq_fin_chirp_bistable/T_sim_chirp_bistable * time_p_1_s(v_P_1(i));
    Ti = 1/fi;
    wi = 5*Ti;
    if wi/2 < time_p_1_s(v_P_1(i))
        idx_tmp = find( time_p_1_s > ti - wi/2 & time_p_1_s < ti + wi/2 );
        time_tmp = time_p_1_s(idx_tmp);
        data_tmp = data_p_1_s(idx_tmp);
    end
    [value_peaks_str, loc_peaks_str] = findpeaks(data_tmp);

    % figure();
```

```
    % plot(time_tmp, data_tmp)
    % hold on
    % plot(time_tmp(loc_peaks_str), value_peaks_str, 'rx')
    % hold off

    p1_bistable_chirp(i) = max(value_peaks_str);
end

figure();
plot(time_p_1_s(v_P_1), p1_bistable_chirp, '.')
xlabel('time [s]')
ylabel('Power [W]')
title('Power g')
grid on
```




```
% Power mobile window g2
% mean value of the peaks in a mobile window
v_P_2 = 1000:10:len_v_2_s-1000;
p2_bistable_chirp = zeros(length(v_P_2), 1);
% 1:length(v_P_2) 2930
for i = 1:length(v_P_2)
    ti = time_p_2_s(v_P_2(i));
    fi = freq_fin_chirp_bistable/T_sim_chirp_bistable * time_p_2_s(v_P_2(i));
    Ti = 1/fi;
    wi = 5*Ti;
    if wi/2 < time_p_2_s(v_P_2(i))
        idx_tmp = find( time_p_2_s > ti - wi/2 & time_p_2_s < ti + wi/2 );
        time_tmp = time_p_2_s(idx_tmp);
        data_tmp = data_p_2_s(idx_tmp);
    end
```

```
    [value_peaks_str, loc_peaks_str] = findpeaks(data_tmp);

    % figure();
    % plot(time_tmp, data_tmp)
    % hold on
    % plot(time_tmp(loc_peaks_str), value_peaks_str, 'rx')
    % hold off

    p2_bistable_chirp(i) = mean(value_peaks_str);
end

figure();
plot(time_p_2_s(v_P_2), p2_bistable_chirp,'.','Color','#D95319')
xlabel('time [s]')
ylabel('Power [W]')
title('Power 2g')
grid on
```

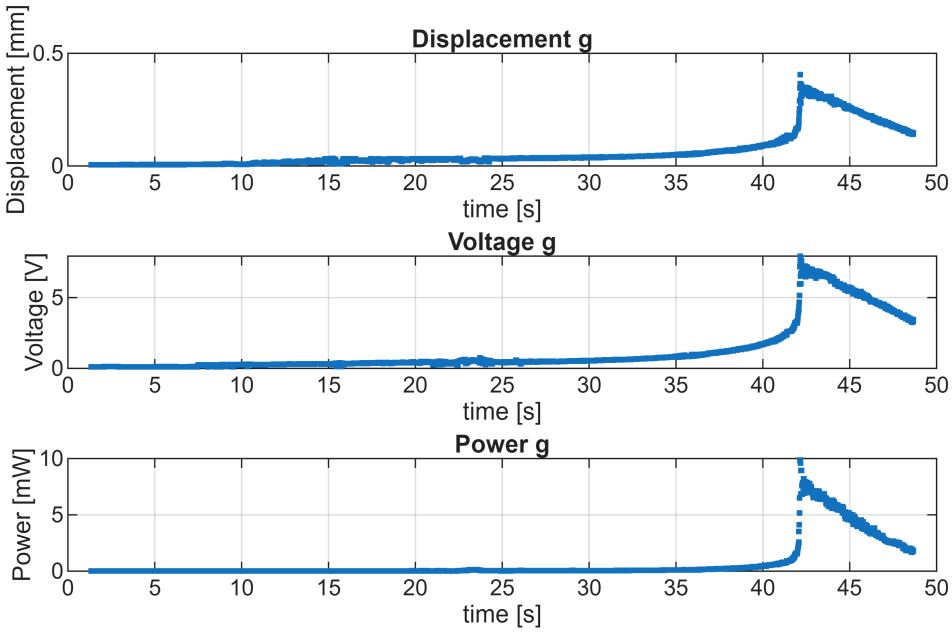



```
disp('BISTABLE CHIRP ACCELERATION g')
```

```
BISTABLE CHIRP ACCELERATION g
```

```
figure();
subplot(3,1,1)
plot(time_d_1_s(v_1), d1_bistable_chirp *1e3, '.')
xlabel('time [s]')
ylabel('Displacement [mm]')
title('Displacement g')
grid on
subplot(3,1,2)
```

```
plot(time_v_1_s(v_V_1), v1_bistable_chirp,'.')
xlabel('time [s]')
ylabel('Voltage [V]')
title('Voltage g')
grid on
subplot(3,1,3)
plot(time_p_1_s(v_P_1), p1_bistable_chirp *1e3, '.')
xlabel('time [s]')
ylabel('Power [mW]')
title('Power g')
grid on
ax = findall(gcf, 'type', 'axes');
linkaxes(ax, 'x');
```

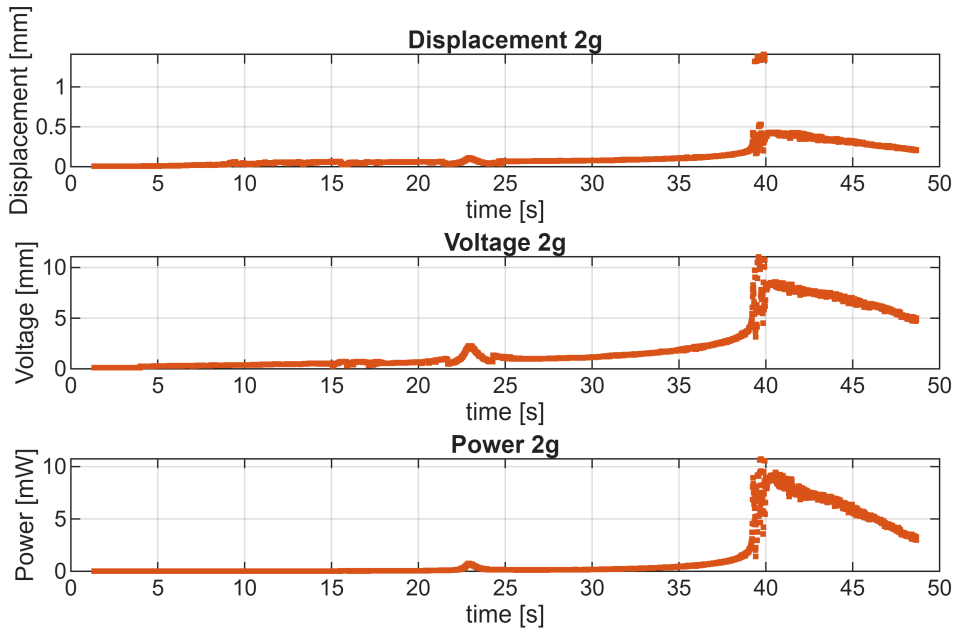



```
disp('BISTABLE CHIRP ACCELERATION 2g')
```

```
BISTABLE CHIRP ACCELERATION 2g
```

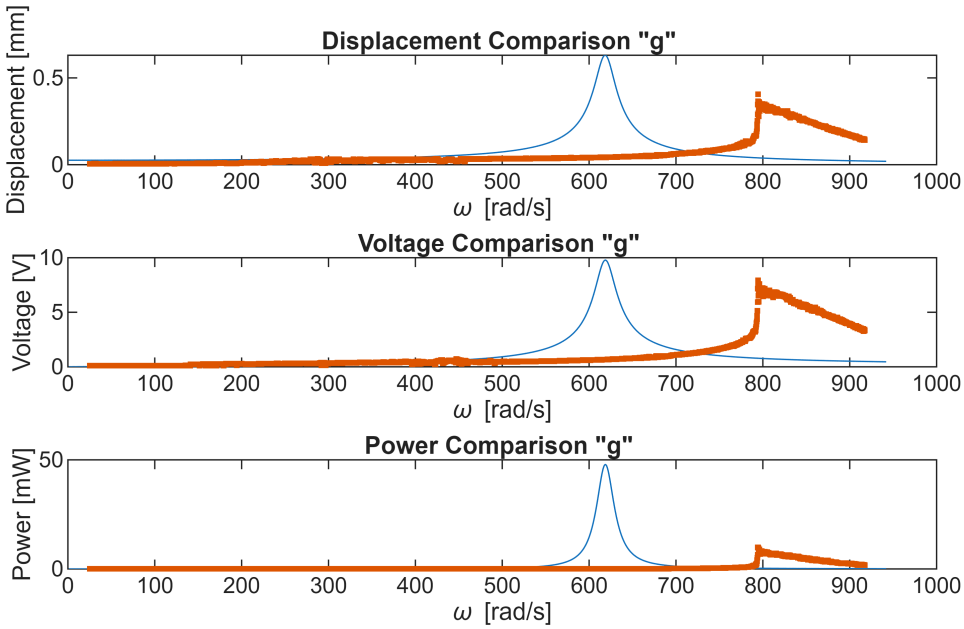
```
figure();
subplot(3,1,1)
plot(time_d_2_s(v_2), d2_bistable_chirp *1e3, '.','Color','#D95319')
xlabel('time [s]')
ylabel('Displacement [mm]')
title('Displacement 2g')
grid on
subplot(3,1,2)
plot(time_v_2_s(v_V_2), v2_bistable_chirp,'.','Color','#D95319')
xlabel('time [s]')
ylabel('Voltage [mm]')
title('Voltage 2g')
```

```
grid on
subplot(3,1,3)
plot(time_p_2_s(v_P_2), p2_bistable_chirp *1e3,'.','Color','#D95319')
xlabel('time [s]')
ylabel('Power [mW]')
title('Power 2g')
grid on
ax = findall(gcf, 'type', 'axes');
linkaxes(ax, 'x');
```



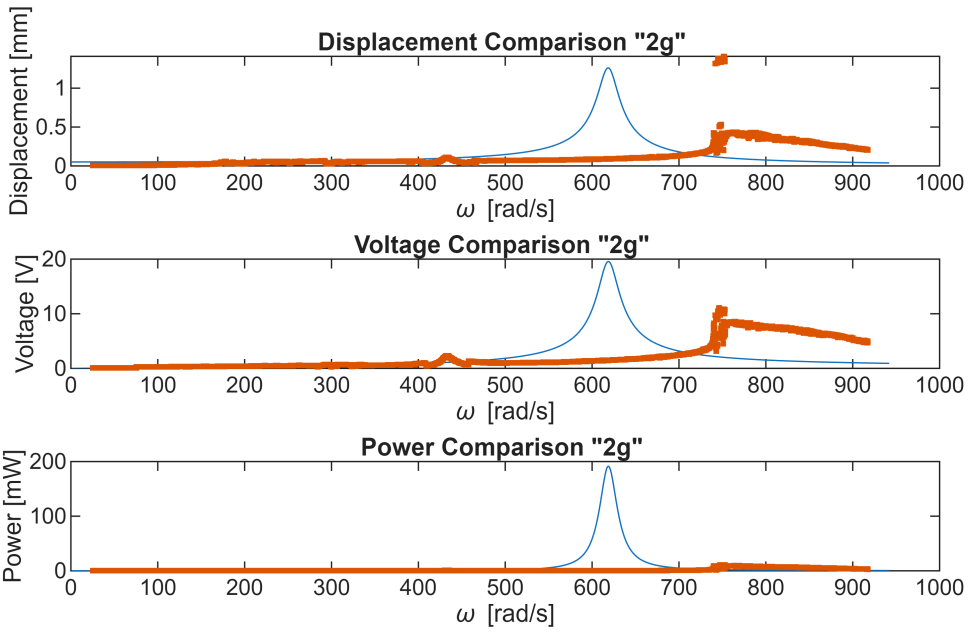

```
% Displacement g
f_axis_d1 = freq_fin_chirp_bistable/T_sim_chirp_bistable * time_p_1_s(v_1); % [Hz]
w_axis_d1 = f_axis_d1 * 2*pi;% [rad/s]
figure();
subplot(3,1,1)
plot(X_a_module_num_g*1e3)
hold on
plot(w_axis_d1, d1_bistable_chirp *1e3,'.')
hold off
xlabel('\omega [rad/s]')
ylabel('Displacement [mm]')
title('Displacement Comparison "g" ')
% Voltage g
f_axis_v1 = freq_fin_chirp_bistable/T_sim_chirp_bistable * time_v_1_s(v_V_1); % [Hz]
w_axis_v1 = f_axis_v1 * 2*pi;% [rad/s]
subplot(3,1,2)
plot(V_a_module_num_g)
hold on
```

```
plot(w_axis_v1, v1_bistable_chirp,'.')
hold off
xlabel('\omega [rad/s]')
ylabel('Voltage [V]')
title('Voltage Comparison "g" ')
% Power g
f_axis_p1 = freq_fin_chirp_bistable/T_sim_chirp_bistable * time_p_1_s(v_P_1); % [Hz]
w_axis_p1 = f_axis_p1 * 2*pi;% [rad/s]
subplot(3,1,3)
plot(P_a_module_num_g *1e3)
hold on
plot(w_axis_p1, p1_bistable_chirp *1e3,'.')
hold off
xlabel('\omega [rad/s]')
ylabel('Power [mW]')
title('Power Comparison "g" ')
ax = findall(gcf, 'type', 'axes');
linkaxes(ax, 'x');
```




```
% Displacement 2g
f_axis_d2 = freq_fin_chirp_bistable/T_sim_chirp_bistable * time_d_2_s(v_2); % [Hz]
w_axis_d2 = f_axis_d2 * 2*pi;% [rad/s]
figure();
subplot(3,1,1)
plot(X_a_module_num_2g * 1e3)
hold on
plot(w_axis_d2, d2_bistable_chirp *1e3,'.')
hold off
xlabel('\omega [rad/s]')
ylabel('Displacement [mm]')
```

```
title('Displacement Comparison "2g" ')
% Voltage 2g
f_axis_v2 = freq_fin_chirp_bistable/T_sim_chirp_bistable * time_v_2_s(v_V_2); % [Hz]
w_axis_v2 = f_axis_v2 * 2*pi;% [rad/s]
subplot(3,1,2)
plot(V_a_module_num_2g)
hold on
plot(w_axis_v2, v2_bistable_chirp,'.')
hold off
xlabel('\omega [rad/s]')
ylabel('Voltage [V]')
title('Voltage Comparison "2g" ')
% Power 2g
f_axis_p2 = freq_fin_chirp_bistable/T_sim_chirp_bistable * time_p_2_s(v_P_2); % [Hz]
w_axis_p2 = f_axis_p2 * 2*pi;% [rad/s]
subplot(3,1,3)
plot(P_a_module_num_2g * 1e3)
hold on
plot(w_axis_p2, p2_bistable_chirp *1e3,'.')
hold off
xlabel('\omega [rad/s]')
ylabel('Power [mW]')
title('Power Comparison "2g" ')
ax = findall(gcf, 'type', 'axes');
linkaxes(ax, 'x');
```



b) Polychromatic acceleration profiles

- Weibull distribution

$$S_{\text{weibull}}(f) = {A_0}^2 \frac{k}{\lambda} \left(\frac{f}{\lambda}\right)^{k-1} e^{-\left(\frac{f}{\lambda}\right)^k}$$

```
% known parameters
k_w = 1.5; % []

% peak frequency equal to the system linear natural  frequency
```

Let's find the missing parameters A_0 and λ:

```
f_e % peak frequency
```

```
f_e =
98.4699
```

```
syms A_0_w K_w lambda_w f
assume([A_0_w K_w lambda_w f] >= 0);
assume([A_0_w K_w lambda_w f] < Inf);
S_Weibull = A_0_w^2 * (K_w)/(lambda_w) * ((f)/(lambda_w))^(K_w-1) * exp(-((f)/
(lambda_w))^(K_w))
```

```
S_Weibull =
```

$$\frac{{A_{0,w}}^2 K_w \, \mathrm{e}^{-\left(\frac{f}{\lambda_w}\right)^{K_w}} \left(\frac{f}{\lambda_w}\right)^{K_w-1}}{\lambda_w}$$

$$\frac{\mathrm{d}}{\mathrm{d}f} S(f) = 0$$

```
S_Weibull_derivative1 = simplify(diff(S_Weibull, f))
```

```
S_Weibull_derivative1 =
```

$$-\frac{{A_{0,w}}^2 K_w \, \mathrm{e}^{-\left(\frac{f}{\lambda_w}\right)^{K_w}} \left(\frac{f}{\lambda_w}\right)^{K_w} \left(K_w \left(\frac{f}{\lambda_w}\right)^{K_w} - K_w + 1\right)}{f^2}$$

```
lambda1 = solve(S_Weibull_derivative1 == 0, lambda_w)
```

```
Warning: Solutions are only valid under certain conditions. To include parameters and conditions in the
solution, specify the 'ReturnConditions' value as 'true'.
lambda1 =
```

$$\frac{f}{\left(\frac{K_w - 1}{K_w}\right)^{1/K_w}}$$

```
lambda2 = subs(lambda1,{K_w},{k_w})
```

lambda2 = $3^{2/3}\,f$

```
lambda_w_num = eval(subs(lambda2, {f}, {f_e}))
```

```
Warning: The function sym/eval is deprecated and will be removed in a future release. Depending on the
usage, use subs or double instead.
lambda_w_num =
204.8257
```

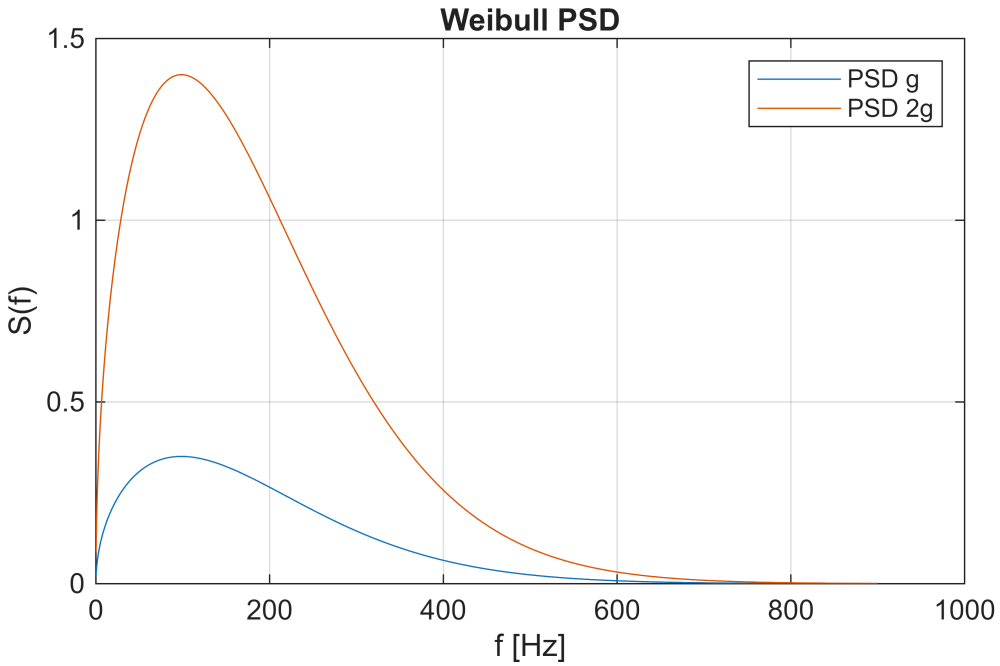
```
T_sim_ploychromatic = 5; % Simulation time (T_sim_chirp_bistable)
% Follow Nyquist theorem to not lose information: fs >= 2 * fmax
fs = 6*freq_fin_chirp_bistable;
N = T_sim_ploychromatic*fs; % number of points to compute fft on
deltaf = 1/T_sim_ploychromatic; % frequency increment;

frequencies = (deltaf:deltaf:(fs))';
% f_sweep = (1:1:150*7)';

S_Weibull_g = eval(subs(S_Weibull, {A_0_w K_w lambda_w f}, {g k_w lambda_w_num
frequencies}));
```

```
Warning: The function sym/eval is deprecated and will be removed in a future release. Depending on the
usage, use subs or double instead.
```

```
S_Weibull_2g = eval(subs(S_Weibull, {A_0_w K_w lambda_w f}, {2*g k_w lambda_w_num
frequencies}));
```

```
Warning: The function sym/eval is deprecated and will be removed in a future release. Depending on the
usage, use subs or double instead.
```

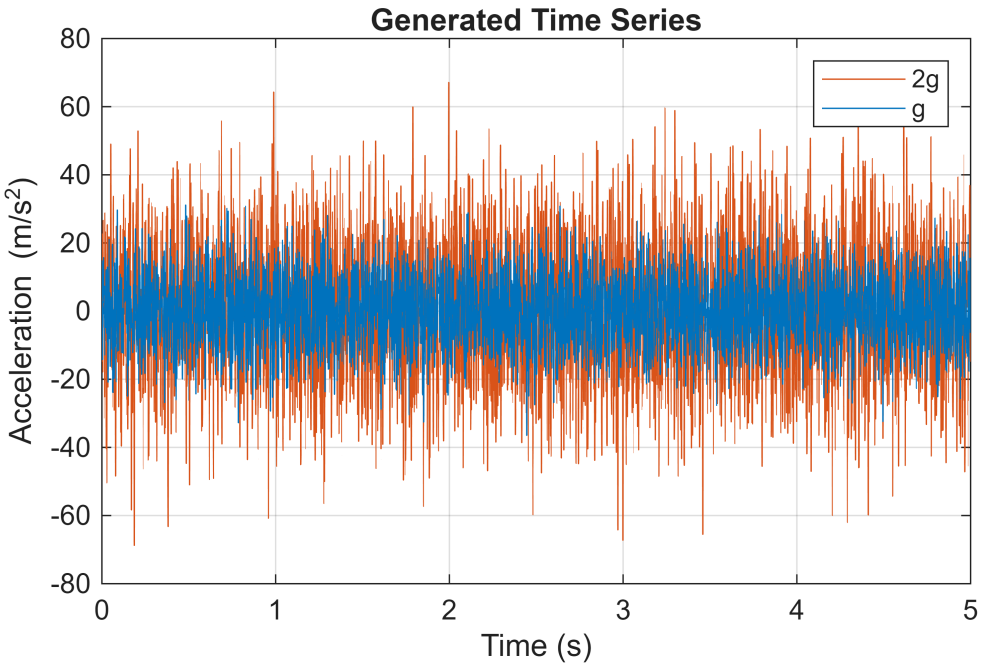
```
figure();
plot(frequencies, S_Weibull_g);
hold on
plot(frequencies, S_Weibull_2g);
xlabel('f [Hz]')
ylabel('S(f)')
title('Weibull PSD')
legend('PSD g','PSD 2g')
grid on
```



Generate the acceleration profiles with random phases:

```
% randomly generated phases
num_frequencies = length(frequencies);
phases_1 = 0 + (2*pi - 0) * rand(1, num_frequencies)'; % random phases [0,2*pi]
phases_2 = 0 + (2*pi - 0) * rand(1, num_frequencies)';

% A superimposition of armonics
amplitudes_1 = sqrt(2*deltaf*(S_Weibull_g)); % amplitude for i-th component of the
spectrum
amplitudes_2 = sqrt(2*deltaf*(S_Weibull_2g)); % amplitude for i-th component of the
spectrum

t = (0:N-1)'/fs;     % time vector

p_t_1 = zeros(N, 1)*g;  % initialize p(t) vector
p_t_2 = zeros(N, 1)*2*g;  % initialize p(t) vector

% superimposition of armonics
for i = 1:N
    p_t_1 = p_t_1 + amplitudes_1(i)*sin(2*pi*frequencies(i)*t + phases_1(i));
    p_t_2 = p_t_2 + amplitudes_2(i)*sin(2*pi*frequencies(i)*t + phases_2(i));
end

figure();
plot(t, p_t_2,'Color','#D95319');
hold on;
plot(t, p_t_1,'Color','#0072BD');
xlabel('Time (s)');
ylabel('Acceleration (m/s^2)');
```

```
title('Generated Time Series');
legend(['2g'],['g'])
grid on;
```




```
% Timeseries to feed simulink model
Acc_timeseries_1 = timeseries(p_t_1, t);
Acc_timeseries_2 = timeseries(p_t_2, t);
```

Automatic Simulink simulations:

```
inputData_ploychromatic = {Acc_timeseries_1, Acc_timeseries_2};
size_inputData_polychromatic = length(inputData_ploychromatic);
T_sim_ploychromatic;
```

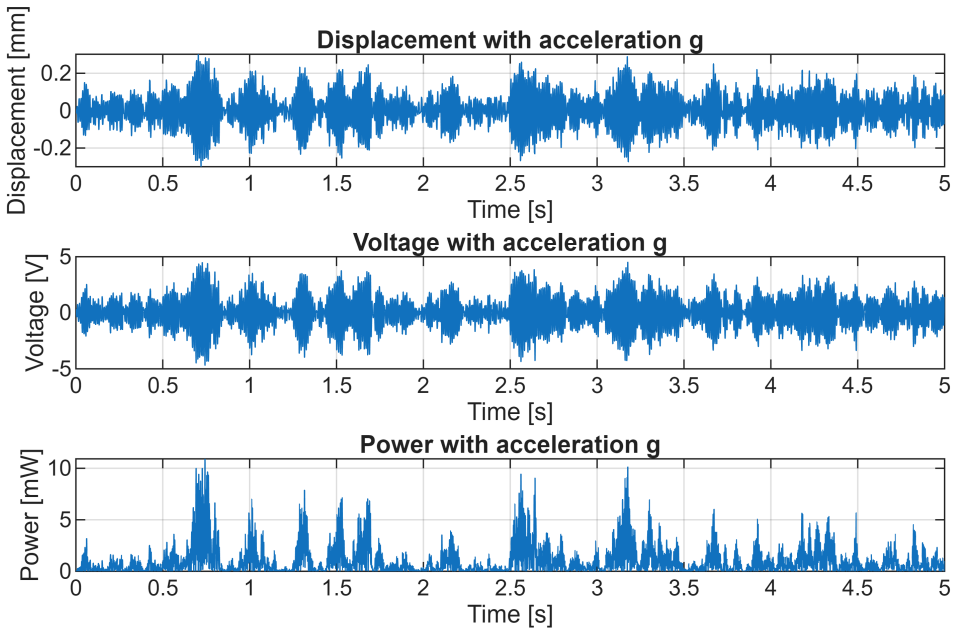
- linear harvester

```
% Initialize a cell array to store output data
Displacement_linear_polychromatic = cell(1,size_inputData_polychromatic);
Voltage_linear_polychromatic = cell(1,size_inputData_polychromatic);
Power_linear_polychromatic = cell(1,size_inputData_polychromatic);

modelName = 'Linear_harvester_polychromatic';

for i = 1:size_inputData_polychromatic
    % Assign the current time series to the Simulink input
    assignin('base', 'inputTimeSeries', inputData_ploychromatic{i});

    % Run the simulation
    simOut = sim(modelName, 'ReturnWorkspaceOutputs', 'on');
```

```
    % Extract output data from the simulation
    Displacement_linear_polychromatic{i} = simOut.get('Displacement');
    Voltage_linear_polychromatic{i} = simOut.get('Voltage');
    Power_linear_polychromatic{i} = simOut.get('Power');

    % Save output with a unique name in the workspace
    assignin('base', ['Displacement_linear_polychromatic_' num2str(i)],
Displacement_linear_polychromatic{i});
    assignin('base', ['Voltage_linear_polychromatic_' num2str(i)],
Voltage_linear_polychromatic{i});
    assignin('base', ['Power_linear_polychromatic_' num2str(i)],
Power_linear_polychromatic{i});
end

disp('LINEAR HARVESTER POLYCHROMATIC ACCELERATION g')
```

```
LINEAR HARVESTER POLYCHROMATIC ACCELERATION g
```

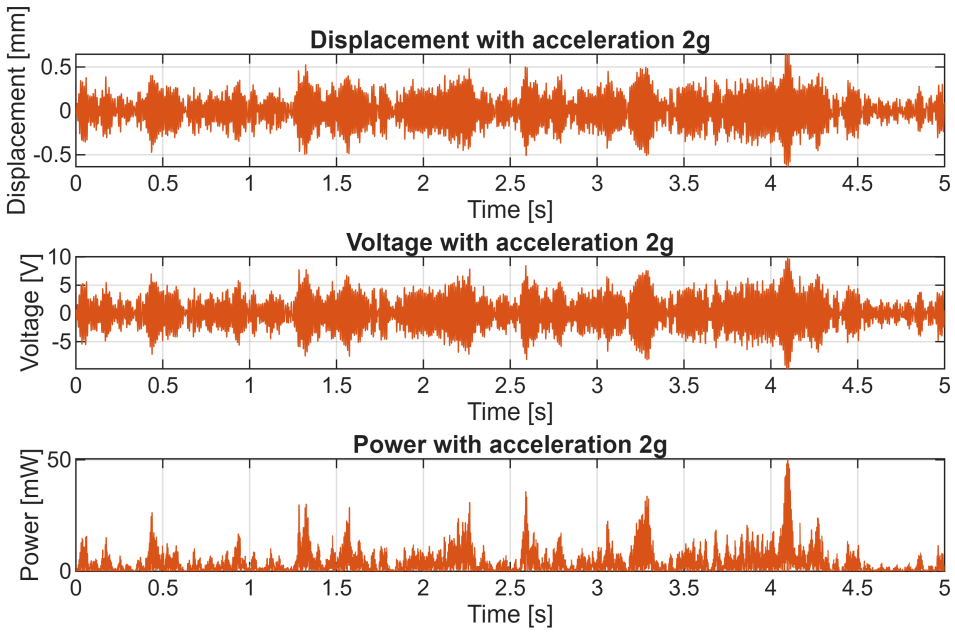
```
figure();
subplot(3, 1, 1)
plot(Displacement_linear_polychromatic_1 * 1e3)
xlabel('Time [s]')
ylabel('Displacement [mm]')
title('Displacement with acceleration g');
grid on
subplot(3, 1, 2)
plot(Voltage_linear_polychromatic_1)
xlabel('Time [s]')
ylabel('Voltage [V]')
title('Voltage with acceleration g');
grid on
subplot(3, 1, 3)
plot(Power_linear_polychromatic_1 * 1e3)
xlabel('Time [s]')
ylabel('Power [mW]')
title('Power with acceleration g');
grid on
```




```
disp('LINEAR HARVESTER POLYCHROMATIC ACCELERATION 2g')
```

```
LINEAR HARVESTER POLYCHROMATIC ACCELERATION 2g
```

```
figure();
subplot(3, 1, 1)
plot(Displacement_linear_polychromatic_2 * 1e3,'Color','#D95319')
xlabel('Time [s]')
ylabel('Displacement [mm]')
title('Displacement with acceleration 2g');
grid on
subplot(3, 1, 2)
plot(Voltage_linear_polychromatic_2,'Color','#D95319')
xlabel('Time [s]')
ylabel('Voltage [V]')
title('Voltage with acceleration 2g');
grid on
subplot(3, 1, 3)
plot(Power_linear_polychromatic_2 * 1e3,'Color','#D95319')
xlabel('Time [s]')
ylabel('Power [mW]')
title('Power with acceleration 2g');
grid on
```



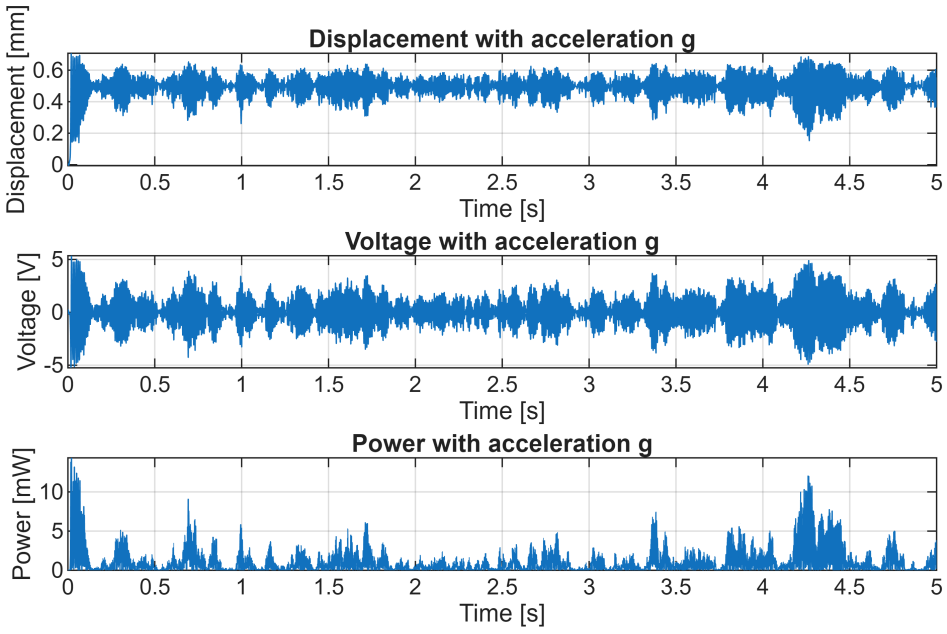
- bistable harvester

```
% Initialize a cell array to store output data
Displacement_bistable_polychromatic = cell(1,size_inputData_polychromatic);
Voltage_bistable_polychromatic = cell(1,size_inputData_polychromatic);
Power_bistable_polychromatic = cell(1,size_inputData_polychromatic);

modelName = 'Bistable_harvester_polychromatic';

for i = 1:size_inputData_polychromatic
    % Assign the current time series to the Simulink input
    assignin('base', 'inputTimeSeries', inputData_ploychromatic{i});

    % Run the simulation
    simOut = sim(modelName, 'ReturnWorkspaceOutputs', 'on');

    % Extract output data from the simulation
    Displacement_bistable_polychromatic{i} = simOut.get('Displacement');
    Voltage_bistable_polychromatic{i} = simOut.get('Voltage');
    Power_bistable_polychromatic{i} = simOut.get('Power');

    % Save output with a unique name in the workspace
    assignin('base', ['Displacement_bistable_polychromatic_' num2str(i)],
Displacement_bistable_polychromatic{i});
    assignin('base', ['Voltage_bistable_polychromatic_' num2str(i)],
Voltage_bistable_polychromatic{i});
    assignin('base', ['Power_bistable_polychromatic_' num2str(i)],
Power_bistable_polychromatic{i});
```

```
end

disp('BISTABLE HARVESTER POLYCHROMATIC ACCELERATION g')
```

```
BISTABLE HARVESTER POLYCHROMATIC ACCELERATION g
```

```
figure();
subplot(3, 1, 1)
plot(Displacement_bistable_polychromatic_1 * 1e3)
xlabel('Time [s]')
ylabel('Displacement [mm]')
title('Displacement with acceleration g');
grid on
subplot(3, 1, 2)
plot(Voltage_bistable_polychromatic_1)
xlabel('Time [s]')
ylabel('Voltage [V]')
title('Voltage with acceleration g');
grid on
subplot(3, 1, 3)
plot(Power_bistable_polychromatic_1 * 1e3)
xlabel('Time [s]')
ylabel('Power [mW]')
title('Power with acceleration g');
grid on
```

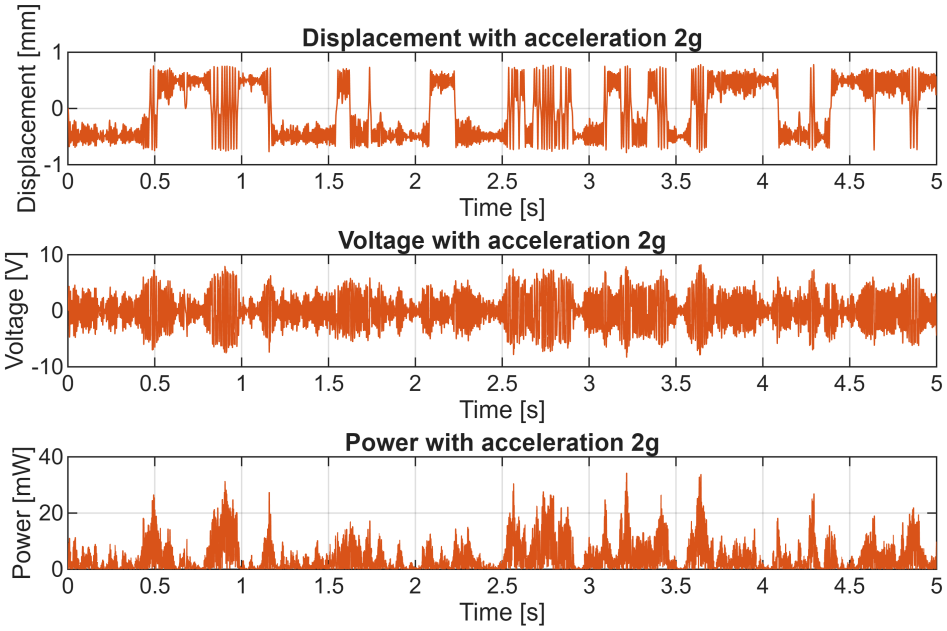



```
disp('BISTABLE HARVESTER POLYCHROMATIC ACCELERATION 2g')
```

```
BISTABLE HARVESTER POLYCHROMATIC ACCELERATION 2g
```

```
figure();
```

```
subplot(3, 1, 1)
plot(Displacement_bistable_polychromatic_2 * 1e3,'Color','#D95319')
xlabel('Time [s]')
ylabel('Displacement [mm]')
title('Displacement with acceleration 2g');
grid on
subplot(3, 1, 2)
plot(Voltage_bistable_polychromatic_2,'Color','#D95319')
xlabel('Time [s]')
ylabel('Voltage [V]')
title('Voltage with acceleration 2g');
grid on
subplot(3, 1, 3)
plot(Power_bistable_polychromatic_2 * 1e3,'Color','#D95319')
xlabel('Time [s]')
ylabel('Power [mW]')
title('Power with acceleration 2g');
grid on
```



- Linear Displacement obtained from weibull $A_0 = g$

```
% resample the signal in order to have fixed step
t_l_1_d = Displacement_linear_polychromatic_1.Time;
d_l_1_d = Displacement_linear_polychromatic_1.Data;
% Follow Nyquist theorem to not lose information: fs >= 2 * fmax
fs_l_1_d = 13*freq_fin_chirp_bistable;
dt_l_1_d = 1/fs_l_1_d;
% Uniform the time vector
t_l_1_d_s = (t_l_1_d(1):dt_l_1_d:t_l_1_d(end))'; % Uniform time vector
```

```
% Interpolate the signal to the new time vector
d_l_1_d_s = interp1(t_l_1_d, d_l_1_d, t_l_1_d_s, 'linear');

% Plot sampled signal over the original
% figure();
% plot(t_l_1, d_l_1 *1e3, 'DisplayName', 'Original Data'); % Original points
% hold on;
% plot(t_l_1_s, d_l_1_s *1e3, '.', 'DisplayName', 'Interpolated Signal'); %
Resampled signal
% xlabel('Time [s]');
% ylabel('Power [mW]');
% title('Sampled signal 2g')
% legend;
% grid on;
% hold off

% % Compute PSD using Welch's method
[pxx_l_1_d, f_l_1_d] = pwelch(d_l_1_d_s, [], [], [], fs_l_1_d);
```

- Linear Displacement obtained from weibull $A_0 = 2g$

```
% resample the signal in order to have fixed step
t_l_2_d = Displacement_linear_polychromatic_2.Time;
d_l_2_d = Displacement_linear_polychromatic_2.Data;
% Follow Nyquist theorem to not lose information: fs >= 2 * fmax
fs_l_2_d = 13*freq_fin_chirp_bistable;
dt_l_2_d = 1/fs_l_2_d;
% Uniform the time vector
t_l_2_d_s = (t_l_2_d(1):dt_l_2_d:t_l_2_d(end))'; % Uniform time vector
% Interpolate the signal to the new time vector
d_l_2_d_s = interp1(t_l_2_d, d_l_2_d, t_l_2_d_s, 'linear');
% % Compute PSD using Welch's method
[pxx_l_2_d, f_l_2_d] = pwelch(d_l_2_d_s, [], [], [], fs_l_2_d);
```

- Bistable Displacement obtained from weibull $A_0 = g$

```
% resample the signal in order to have fixed step
t_b_1_d = Displacement_bistable_polychromatic_1.Time;
d_b_1_d = Displacement_bistable_polychromatic_1.Data;
% Follow Nyquist theorem to not lose information: fs >= 2 * fmax
fs_b_1_d = 13*freq_fin_chirp_bistable;
dt_b_1_d = 1/fs_b_1_d;
% Uniform the time vector
t_b_1_d_s = (t_b_1_d(1):dt_b_1_d:t_b_1_d(end))'; % Uniform time vector
% Interpolate the signal to the new time vector
d_b_1_d_s = interp1(t_b_1_d, d_b_1_d, t_b_1_d_s, 'linear');
% % Compute PSD using Welch's method
[pxx_b_1_d, f_b_1_d] = pwelch(d_b_1_d_s, [], [], [], fs_b_1_d);
```

- Bistable Displacement obtained from weibull $A_0 = 2g$

```
% resample the signal in order to have fixed step
t_b_2_d = Displacement_bistable_polychromatic_2.Time;
d_b_2_d = Displacement_bistable_polychromatic_2.Data;
% Follow Nyquist theorem to not lose information: fs >= 2 * fmax
fs_b_2_d = 13*freq_fin_chirp_bistable;
dt_b_2_d = 1/fs_b_2_d;
% Uniform the time vector
t_b_2_d_s = (t_b_2_d(1):dt_b_2_d:t_b_2_d(end))'; % Uniform time vector
% Interpolate the signal to the new time vector
d_b_2_d_s = interp1(t_b_2_d, d_b_2_d, t_b_2_d_s, 'linear');
% % Compute PSD using Welch's method
[pxx_b_2_d, f_b_2_d] = pwelch(d_b_1_d_s, [], [], [], fs_b_2_d);
```

- Linear Voltage obtained from weibull $A_0 = g$

```
% resample the signal in order to have fixed step
t_l_1_v = Voltage_linear_polychromatic_1.Time;
d_l_1_v = Voltage_linear_polychromatic_1.Data;
% Follow Nyquist theorem to not lose information: fs >= 2 * fmax
fs_l_1_v = 13*freq_fin_chirp_bistable;
dt_l_1_v = 1/fs_l_1_v;
% Uniform the time vector
t_l_1_v_s = (t_l_1_v(1):dt_l_1_v:t_l_1_v(end))'; % Uniform time vector
% Interpolate the signal to the new time vector
d_l_1_v_s = interp1(t_l_1_v, d_l_1_v, t_l_1_v_s, 'linear');

% % Compute PSD using Welch's method
[pxx_l_1_v, f_l_1_v] = pwelch(d_l_1_v_s, [], [], [], fs_l_1_v);
```

- Linear Voltage obtained from weibull $A_0 = 2g$

```
% resample the signal in order to have fixed step
t_l_2_v = Voltage_linear_polychromatic_2.Time;
d_l_2_v = Voltage_linear_polychromatic_2.Data;
% Follow Nyquist theorem to not lose information: fs >= 2 * fmax
fs_l_2_v = 13*freq_fin_chirp_bistable;
dt_l_2_v = 1/fs_l_2_v;
% Uniform the time vector
t_l_2_v_s = (t_l_2_v(1):dt_l_2_v:t_l_2_v(end))'; % Uniform time vector
% Interpolate the signal to the new time vector
d_l_2_v_s = interp1(t_l_2_v, d_l_2_v, t_l_2_v_s, 'linear');
% % Compute PSD using Welch's method
[pxx_l_2_v, f_l_2_v] = pwelch(d_l_2_v_s, [], [], [], fs_l_2_v);
```

- Bistable Voltage obtained from weibull $A_0 = g$

```
% resample the signal in order to have fixed step
t_b_1_v = Voltage_bistable_polychromatic_1.Time;
d_b_1_v = Voltage_bistable_polychromatic_1.Data;
% Follow Nyquist theorem to not lose information: fs >= 2 * fmax
fs_b_1_v = 13*freq_fin_chirp_bistable;
dt_b_1_v = 1/fs_b_1_v;
% Uniform the time vector
t_b_1_v_s = (t_b_1_v(1):dt_b_1_v:t_b_1_v(end))'; % Uniform time vector
% Interpolate the signal to the new time vector
d_b_1_v_s = interp1(t_b_1_v, d_b_1_v, t_b_1_v_s, 'linear');
% % Compute PSD using Welch's method
[pxx_b_1_v, f_b_1_v] = pwelch(d_b_1_v_s, [], [], [], fs_b_1_v);
```

- Bistable Voltage obtained from weibull $A_0 = 2g$

```
% resample the signal in order to have fixed step
t_b_2_v = Voltage_bistable_polychromatic_2.Time;
d_b_2_v = Voltage_bistable_polychromatic_2.Data;
% Follow Nyquist theorem to not lose information: fs >= 2 * fmax
fs_b_2_v = 13*freq_fin_chirp_bistable;
dt_b_2_v = 1/fs_b_2_v;
% Uniform the time vector
t_b_2_v_s = (t_b_2_v(1):dt_b_2_v:t_b_2_v(end))'; % Uniform time vector
% Interpolate the signal to the new time vector
d_b_2_v_s = interp1(t_b_2_v, d_b_2_v, t_b_2_v_s, 'linear');
% % Compute PSD using Welch's method
[pxx_b_2_v, f_b_2_v] = pwelch(d_b_1_v_s, [], [], [], fs_b_2_v);
```

- Linear Power obtained from weibull $A_0 = g$

```
% resample the signal in order to have fixed step
t_l_1 = Power_linear_polychromatic_1.Time;
d_l_1 = Power_linear_polychromatic_1.Data;
% Follow Nyquist theorem to not lose information: fs >= 2 * fmax
fs_l_1 = 13*freq_fin_chirp_bistable;
dt_l_1 = 1/fs_l_1;
% Uniform the time vector
t_l_1_s = (t_l_1(1):dt_l_1:t_l_1(end))'; % Uniform time vector
% Interpolate the signal to the new time vector
d_l_1_s = interp1(t_l_1, d_l_1, t_l_1_s, 'linear');

% Plot sampled signal over the original
% figure();
% plot(t_l_1, d_l_1 *1e3, 'DisplayName', 'Original Data'); % Original points
```

```
% hold on;
% plot(t_l_1_s, d_l_1_s *1e3, '.', 'DisplayName', 'Interpolated Signal'); %
Resampled signal
% xlabel('Time [s]');
% ylabel('Power [mW]');
% title('Sampled signal 2g')
% legend;
% grid on;
% hold off

% % Compute PSD using Welch's method
[pxx_l_1, f_l_1] = pwelch(d_l_1_s, [], [], [], fs_l_1);
```

- Linear Power obtained from weibull $A_0 = 2g$

```
% resample the signal in order to have fixed step
t_l_2 = Power_linear_polychromatic_2.Time;
d_l_2 = Power_linear_polychromatic_2.Data;
% Follow Nyquist theorem to not lose information: fs >= 2 * fmax
fs_l_2 = 13*freq_fin_chirp_bistable;
dt_l_2 = 1/fs_l_2;
% Uniform the time vector
t_l_2_s = (t_l_2(1):dt_l_2:t_l_2(end))'; % Uniform time vector
% Interpolate the signal to the new time vector
d_l_2_s = interp1(t_l_2, d_l_2, t_l_2_s, 'linear');
% % Compute PSD using Welch's method
[pxx_l_2, f_l_2] = pwelch(d_l_2_s, [], [], [], fs_l_2);
```

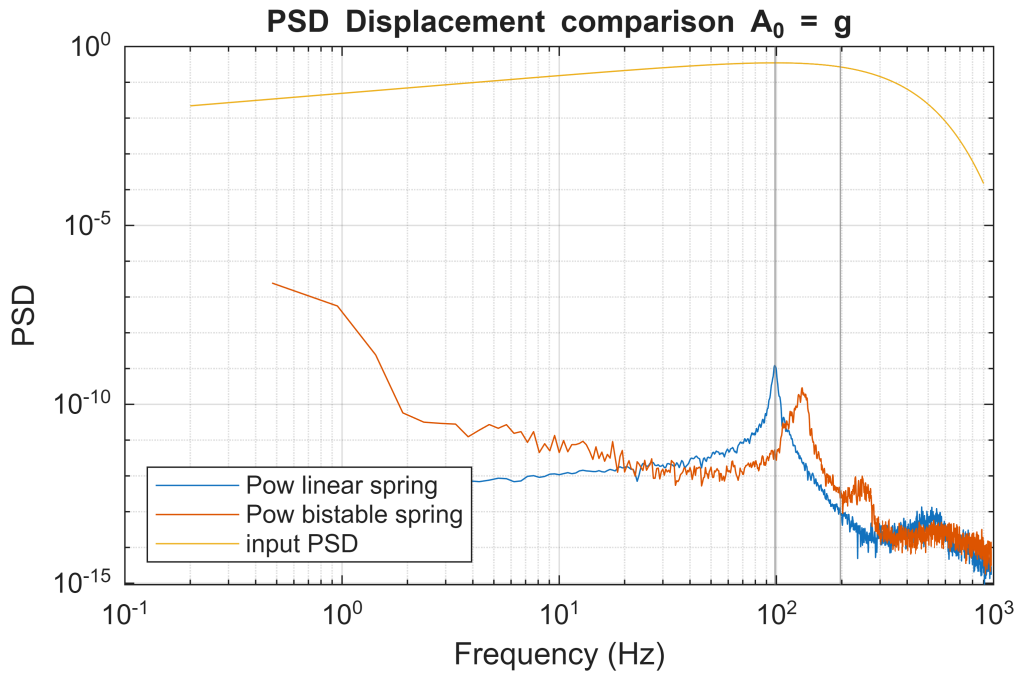
- Bistable Power obtained from weibull $A_0 = g$

```
% resample the signal in order to have fixed step
t_b_1 = Power_bistable_polychromatic_1.Time;
d_b_1 = Power_bistable_polychromatic_1.Data;
% Follow Nyquist theorem to not lose information: fs >= 2 * fmax
fs_b_1 = 13*freq_fin_chirp_bistable;
dt_b_1 = 1/fs_b_1;
% Uniform the time vector
t_b_1_s = (t_b_1(1):dt_b_1:t_b_1(end))'; % Uniform time vector
% Interpolate the signal to the new time vector
d_b_1_s = interp1(t_b_1, d_b_1, t_b_1_s, 'linear');
% % Compute PSD using Welch's method
[pxx_b_1, f_b_1] = pwelch(d_b_1_s, [], [], [], fs_b_1);
```

- Bistable Power obtained from weibull $A_0 = 2g$

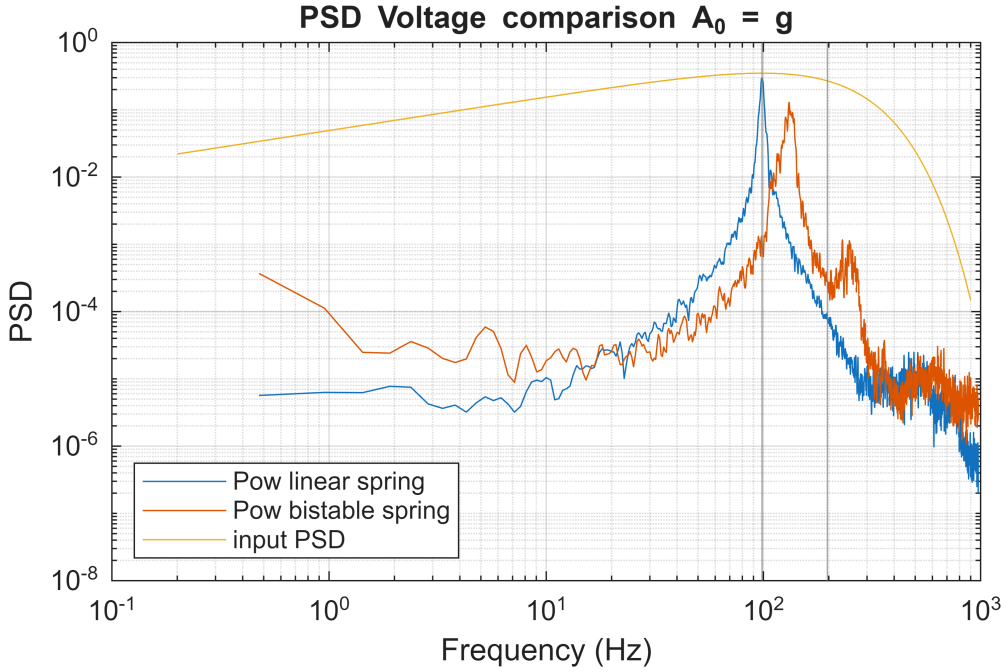
```
% resample the signal in order to have fixed step
t_b_2 = Power_bistable_polychromatic_2.Time;
```

```
d_b_2 = Power_bistable_polychromatic_2.Data;
% Follow Nyquist theorem to not lose information: fs >= 2 * fmax
fs_b_2 = 13*freq_fin_chirp_bistable;
dt_b_2 = 1/fs_b_2;
% Uniform the time vector
t_b_2_s = (t_b_2(1):dt_b_2:t_b_2(end))'; % Uniform time vector
% Interpolate the signal to the new time vector
d_b_2_s = interp1(t_b_2, d_b_2, t_b_2_s, 'linear');
% % Compute PSD using Welch's method
[pxx_b_2, f_b_2] = pwelch(d_b_1_s, [], [], [], fs_b_2);

% Displacement 1 plot
figure();
loglog(f_l_1_d, pxx_l_1_d);
hold on
loglog(f_b_1_d, pxx_b_1_d);
loglog(frequencies, S_Weibull_g)
xline(f_n, 'Color',[0.5, 0.5, 0.5])
xline(2*f_n, 'Color',[0.5, 0.5, 0.5])
xlabel('Frequency (Hz)'); ylabel('PSD');
legend('Pow linear spring','Pow bistable spring','input PSD','Location','SouthWest')
title('PSD Displacement comparison A_0 = g')
grid on
```

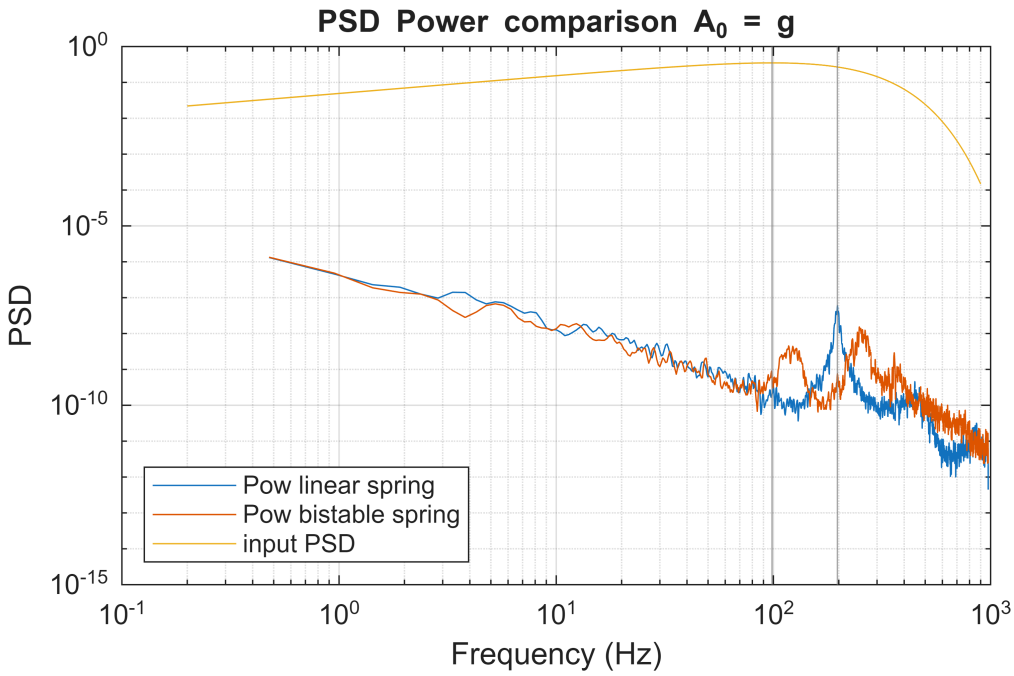



```
% Voltage 1 plot
figure();
loglog(f_l_1_v, pxx_l_1_v);
hold on
loglog(f_b_1_v, pxx_b_1_v);
loglog(frequencies, S_Weibull_g)
```

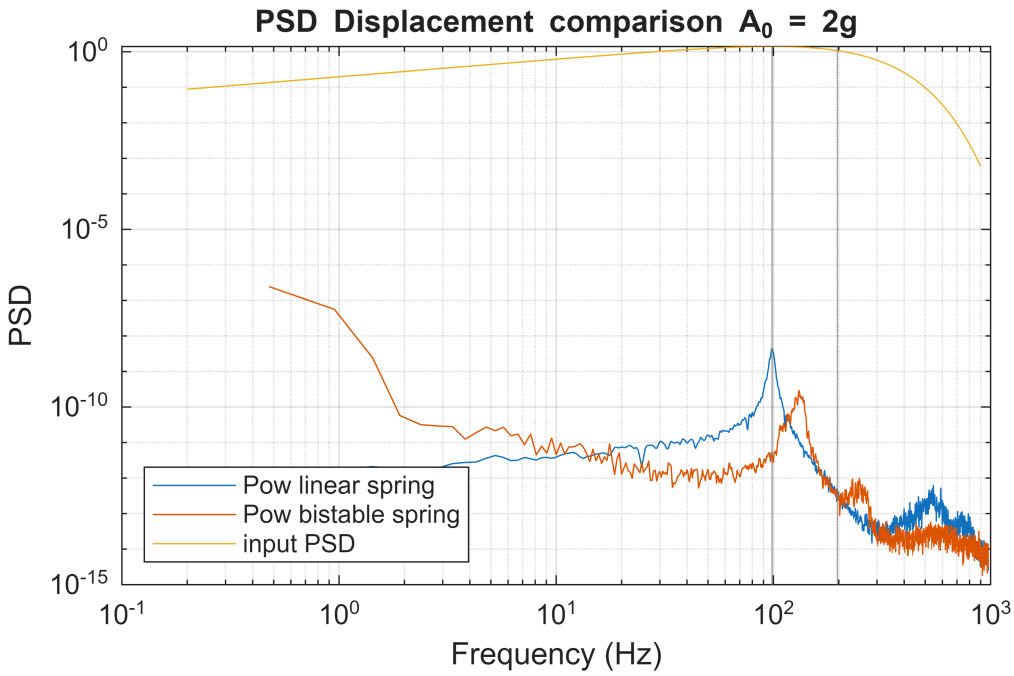
```
xline(f_n, 'Color',[0.5, 0.5, 0.5])
xline(2*f_n, 'Color',[0.5, 0.5, 0.5])
xlabel('Frequency (Hz)'); ylabel('PSD');
legend('Pow linear spring','Pow bistable spring','input PSD','Location','SouthWest')
title('PSD Voltage comparison A_0 = g')
grid on
```



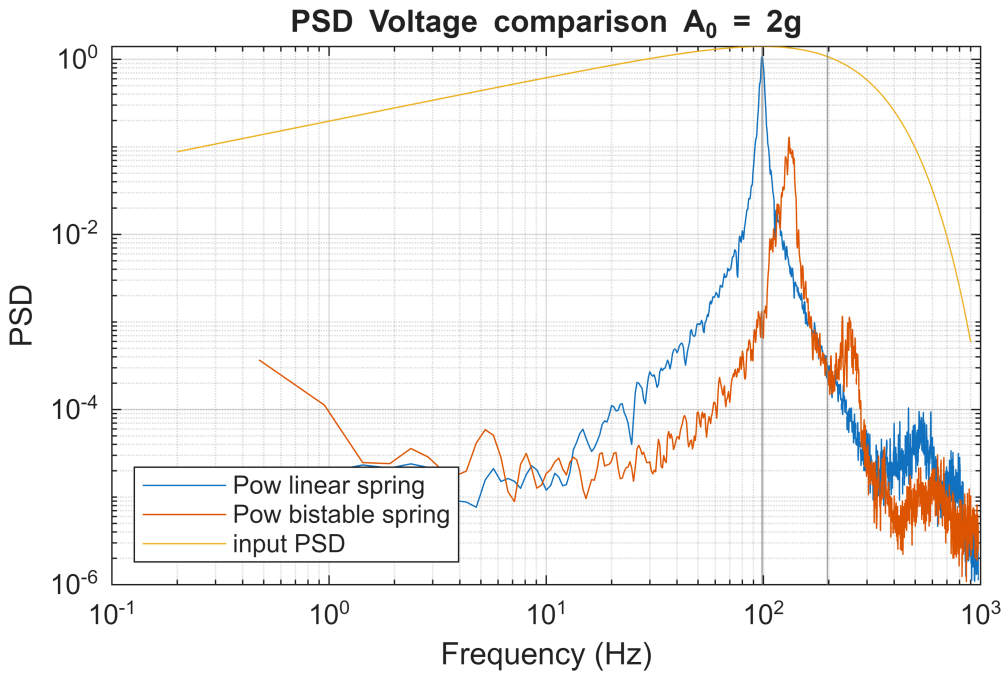

```
% Power 1 plots
figure();
loglog(f_l_1, pxx_l_1);
hold on
loglog(f_b_1, pxx_b_1);
loglog(frequencies, S_Weibull_g)
xline(f_n, 'Color',[0.5, 0.5, 0.5])
xline(2*f_n, 'Color',[0.5, 0.5, 0.5])
xlabel('Frequency (Hz)'); ylabel('PSD');
legend('Pow linear spring','Pow bistable spring','input PSD','Location','SouthWest')
title('PSD Power comparison A_0 = g')
grid on
```



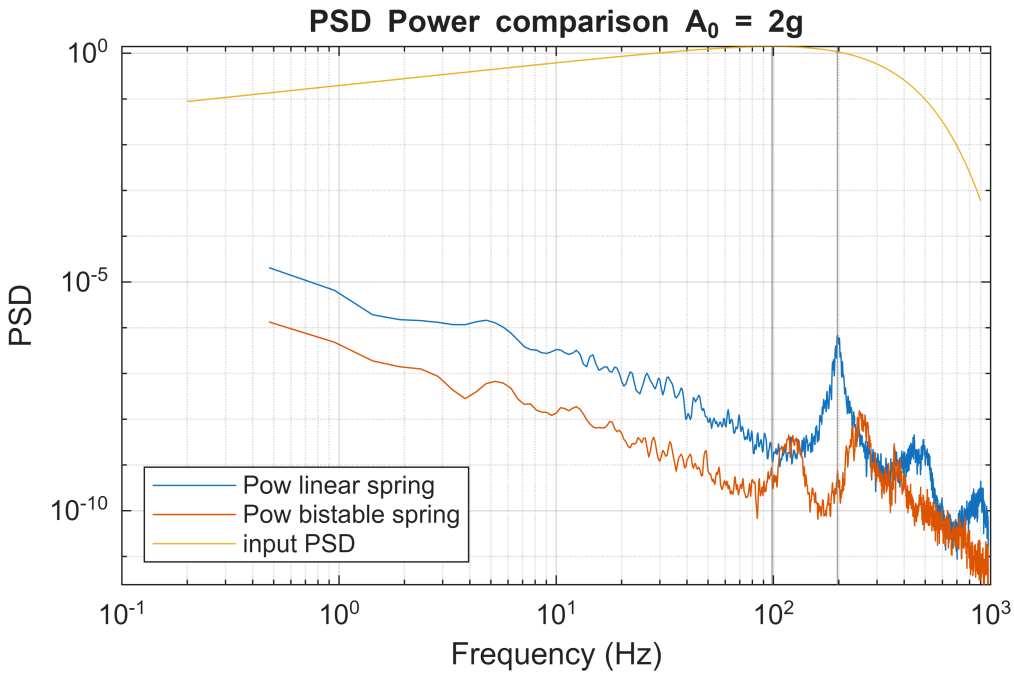

```
% Displacement 2 plot
figure();
loglog(f_l_2_d, pxx_l_2_d);
hold on
loglog(f_b_2_d, pxx_b_2_d);
loglog(frequencies, S_Weibull_2g)
xline(f_n, 'Color',[0.5, 0.5, 0.5])
xline(2*f_n, 'Color',[0.5, 0.5, 0.5])
xlabel('Frequency (Hz)'); ylabel('PSD');
legend('Pow linear spring','Pow bistable spring','input PSD','Location','SouthWest')
title('PSD Displacement comparison A_0 = 2g')
grid on
```




```
% Voltage 2 plot
figure();
loglog(f_l_2_v, pxx_l_2_v);
hold on
loglog(f_b_2_v, pxx_b_2_v);
loglog(frequencies, S_Weibull_2g)
xline(f_n, 'Color',[0.5, 0.5, 0.5])
xline(2*f_n, 'Color',[0.5, 0.5, 0.5])
xlabel('Frequency (Hz)'); ylabel('PSD');
legend('Pow linear spring','Pow bistable spring','input PSD','Location','SouthWest')
title('PSD Voltage comparison A_0 = 2g')
grid on
```




```
% Power 2 plot
figure();
loglog(f_l_2, pxx_l_2);
hold on
loglog(f_b_2, pxx_b_2);
loglog(frequencies, S_Weibull_2g)
xline(f_n, 'Color',[0.5, 0.5, 0.5])
xline(2*f_n, 'Color',[0.5, 0.5, 0.5])
xlabel('Frequency (Hz)'); ylabel('PSD');
legend('Pow linear spring','Pow bistable spring','input PSD','Location','SouthWest')
title('PSD Power comparison A_0 = 2g')
grid on
```


PSD Power comparison A_0 = 2g
PSD
Frequency (Hz)
Pow linear spring
Pow bistable spring
input PSD