Proceedings of the 14th International Fronk Conference on Vibrations
Oct. 31, 2016, Logan, Utah, USA

BASE RESONATORS AND COMPOSITE BEAMS

Steven Proctor
Utah State University
Logan, UT, United States

ABSTRACT

Introduction to Resonators and their use in designing physical systems. Derivation of governing equations and equations of motion. Comparison of various types of resonators under the same conditions using custom program. Visual responses of each resonator over a range of frequencies. Design and analysis of composite beam structure using a Case I resonator.

INTRODUCTION

Resonators are a type of a spring mass damper system used to determine the response of an equivalent system. Many physical systems can be modeled as a resonator. This allows us to easily determine the physical response of the system under various frequencies. Some examples where this tool would prove valuable include beam and bridge design and suspension systems on a car.

EQUATIONS OF MOTION

A typical base resonator consists of a mass connected to two springs and dampers on opposing sides of the mass. One spring and damper connect to a rigid structure, the other side is free to be acted upon by a force. An example of this system can be seen in Figure 1.

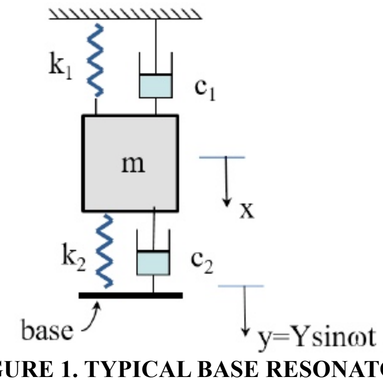

FIGURE 1. TYPICAL BASE RESONATOR

In order to obtain the equations that describe the response of this system, we start with a free-body diagram (Figure 2).

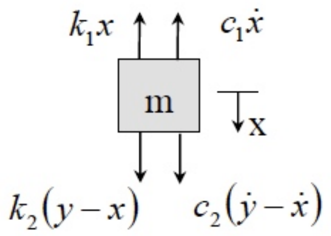

FIGURE 2. FREE BODY DIAGRAM OF MASS

By using Newton's 2nd Law, we obtain the governing equation (Equation 1).

$$m\ddot{x} + c_1\dot{x} + k_1x = (k_2(y\text{-}x) + c_2(\dot{y}\text{-}\dot{x})) \quad (1)$$

or

$$m\ddot{x} + c_{eq}\dot{x} + k_{eq}x = k_2y + c_2\dot{y}$$

where $c_{eq} = c_1 + c_2$

and $k_{eq} = k_1 + k_2$

At this point, we can also define the damping ratio (Equation 2) and the natural frequency (Equation 3) of the system.

$$\omega_n = \sqrt{\frac{k_{eq}}{m}} \quad (2)$$

$$\zeta = \frac{c_{eq}}{c_c} = \frac{c_{eq}}{2\sqrt{mk_{eq}}} \quad (3)$$

In order to derive the response of the system, the governing equation equated to a function of force over time (Equation 4).

$$m\ddot{x} + c_{eq}\dot{x} + k_{eq}x = F(t) \quad (4)$$

The function of force is defined as a sinusoidal forcing function as show in Equation 5.

$$F(t) = F_0 \sin(\omega t \text{-} \alpha) \quad (5)$$

where $F_0 = Y[k_2^2 + (c_2\omega)^2]^{1/2}$

and $\alpha = \tan^{-1}\left(\text{-}\frac{c_2\omega}{k_2}\right)$

The steady state solution for a resonator is also a sinusoidal function (Equation 6).

$$x_p = X_p \sin(\omega t \text{-} \phi) \quad (6)$$

By plugging the steady state solution into the governing equation shown in Equation 4 we obtain the steady state amplitude. After taking the derivatives of Equation 6 and solving for the amplitude we derive Equation 7. Note that all occurrences of the driving frequency or natural frequency have been replaced with the frequency ratio (r) as defined below.

$$X_p = \frac{F_0/k_{eq}}{[(1-r^2)^2 + (2\zeta r)^2]^{1/2}} \quad (7)$$

where

$$r = \frac{\omega}{\omega_n}$$

The phase angle for the steady state solution is derived by using the identity shown in Equation 8 and Equation 9. After solving for the phase angle and substituting in the frequency ratio, we are left with Equation 10.

$$\tan\phi = \frac{\tan\alpha + tan\phi_1}{1 - \tan\alpha\, tan\phi_1} = \tan(\alpha + \phi_1) \quad (8)$$

$$\phi_1 = tan^{-1}\left(\frac{c_{eq}\omega}{k_{eq}\text{-}m\omega^2}\right) \quad (9)$$

$$\phi = tan^{-1}\left(\frac{r\left(2\zeta k_2 \text{-} c_2\omega_n(1-r^2)\right)}{k_2(1-r^2) + 2c_2\omega_n\zeta r^2}\right) \quad (10)$$

The displacement transmissibility is obtained by diving the steady state amplitude by the amplitude of the forcing function and results in Equation 11.

$$T_d = \frac{X_p}{Y} = \frac{F_0/Yk_{eq}}{[(1-r^2)^2 + (2\zeta r)^2]^{1/2}} \quad (11)$$

Similarly, the frequency response of the system is derived by diving the steady state amplitude by the initial force divided the equivalent spring constant for the system (Equation 12).

$$|H(i\omega)| = \frac{X_p}{F_0/k_{eq}} = \frac{1}{[(1-r^2)^2 + (2\zeta r)^2]^{1/2}} \quad (12)$$

In order to derive the the equation that describe the force exerted on the base we must start again with a free body diagram. This time of only the base (Figure 3).

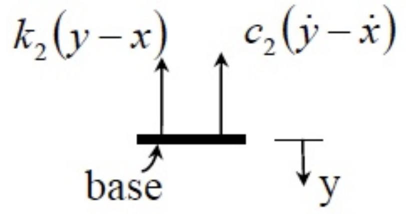

FIGURE 3. FREE BODY DIAGRAM OF BASE

By applying Newton's 2nd Law to this diagram we obtain an equation for the force of the base (Equation 13).

$$F_{base}(t) = \text{-}(k_2(y\text{-}x) + c_2(\dot{y}\text{-}\dot{x})) \quad (13)$$

If we look back at our governing equation for the mass, we can see that the force on the base is the negative of the right hand side of the equation which allows us to write Equation 14.

$$m\ddot{x} + c_1\dot{x} + k_1 x = \text{-}F_{base}(t) \quad (14)$$

As done previously we plug the steady state equation into this result and set it equal to a sinusoidal force (Equation 15). We can then solve for the maximum force that the base will experience (Equation 16).

$$F_{base}(t) = -X_p\big((k_1 - m\omega^2)\sin(\omega t - \phi) + c_1\cos(\omega t - \phi)\big) \\ = F_T \sin(\omega t - \psi) \qquad (15)$$

$$F_{T\text{-}base_{max}} = -X_p\sqrt{(k_1 - k_{eq}r^2)^2 + (c_1\omega_n r)^2} \qquad (16)$$

Just as was done to find the displacement transmissibility, we can find the the force transmissibility of the base by diving the maximum force the base experiences by the amplitude of the forcing function multiplied by the spring constant of the spring between the base and the mass (Equation 17).

$$F_{trans_{base}} = \frac{X_p\sqrt{(k_1 - k_{eq}r^2)^2 + (c_1\omega_n r)^2}}{Yk_2} \qquad (17)$$

We then repeat this process on the wall, beginning with a free body diagram of the wall (Figure 4).

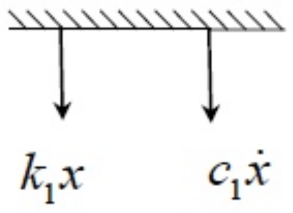

FIGURE 4. FREE BODY DIAGRAM OF WALL

Newton's 2nd Law leads us to Equation 18.

$$F_{wall}(t) = k_1 x + c_1\dot{x} \qquad (18)$$

Again we insert the steady state solution and its derivatives into Equation 18 to gain Equation 19.

$$F_{wall}(t) = k_1 X_p \sin(\omega t - \phi) + c_1 X_p \cos(\omega t - \phi) \qquad (19)$$

By setting Equation 19 equal to a sinusoidal function (Equation 20) we can solve for the maximum force on the wall (Equation 21).

$$F_{wall}(t) = k_1 X_p \sin(\omega t - \phi) + c_1 X_p \cos(\omega t - \phi) \\ = F_T \sin(\omega t - \phi_2) \qquad (20)$$

$$F_{T\text{-}wall_{max}} = X_p\sqrt{k_1^2 + (c_1 r\omega_n)^2} \qquad (21)$$

As before, we divide the maximum force the wall experiences by the amplitude of the forcing function multiplied by the spring constant of the spring between the base and the mass (Equation 22).

$$F_{trans_{wall}} = \frac{X_p\sqrt{k_1^2 + (c_1 r\omega_n)^2}}{Yk_2} \qquad (22)$$

These equations were derived using a typical resonator. However, there are five different types of resonators that are frequently used. All five types are variations of the resonator used to derive these equations of motions. These variations can be seen in Figure 5.

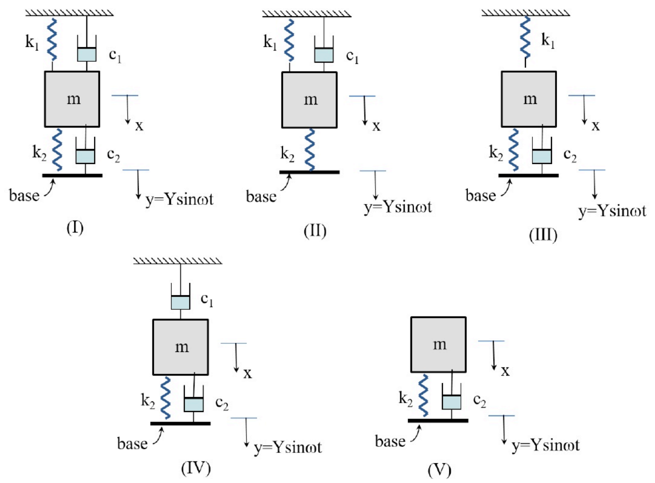

FIGURE 5. TYPES OF RESONATORS

Each variations eliminates one or more of the spring damper components. As such, no additional derivations need be performed to find the appropriate equations of motion. Instead the missing component is set equal to zero in any of the previously derived equations. This property makes is easy to compare each type of resonator.

RESPONSES OF VARIOUS RESONATORS

In order to see the effect of each type of resonator, we coded a simple program to output results of a specific resonator based on any given input force. The first test was to output the frequency response, phase angle, displacement transmissibility, and force transmissibility of both the base and the wall.

The resonator cases are shown in Table 1. For each of these tests a frequency ratio of 0.8 was used.

TABLE 1. RESONATOR TEST CASES

CASE	Y (m)	m (kg)	c_1 (N-s/m)	c_2 (N-s/m)	k_1 (N/m)	k_2 (N/m)
I	0.1	2	2	5	200	300
II	0.1	2	2	0	200	300
III	0.1	2	0	5	200	300
IV	0.1	2	2	5	0	300
V	0.1	2	0	5	0	300

The results of these test can be seen in Table 2. All results seen in this table were verified by hand as well.

TABLE 2. TEST CASE RESULTS

	Case I	Case II	Case III	Case IV	Case V
Natural Frequency	15.811	15.811	15.811	12.25	12.15
Damping Ratio	0.1107	0.0316	0.079	0.143	0.102
Static Amplitude	0.0613	0.06	0.0613	0.101	0.101
Frequency Response	2.4925	2.75	2.62	2.3449	2.53
Phase Angle	0.2494	0.139	0.13	0.404	0.264
Displacement Trans.	1.528	1.65	1.6	2.38	2.56
Force Trans. - Base	0.625	0.675	0.64	1.53	1.64
Force Trans. - Wall	1.027	1.109	1.07	0.155	0

In order to gain a better understanding of the response of each resonator, the same test cases were used, but this time with a range of frequency ratios from 0.1 to 2. The results of these test were plotted against each other in Figures 6 through 10. The cases labeled below each graph correspond to the cases shown in Figure 5.

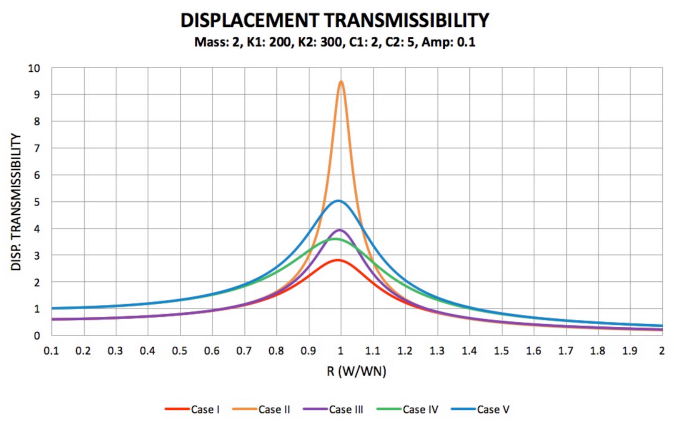

FIGURE 6. DISPLACEMENT TRANSMISSIBILITY

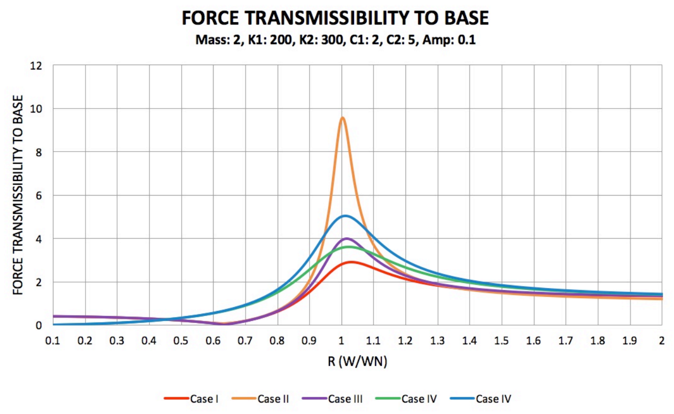

FIGURE 7. FORCE TRANSMISSIBILITY OF BASE

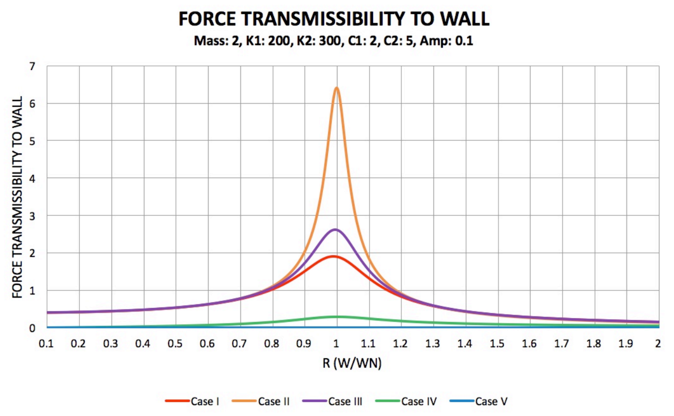

FIGURE 8. FORCE TRANSMISSIBILITY OF WALL

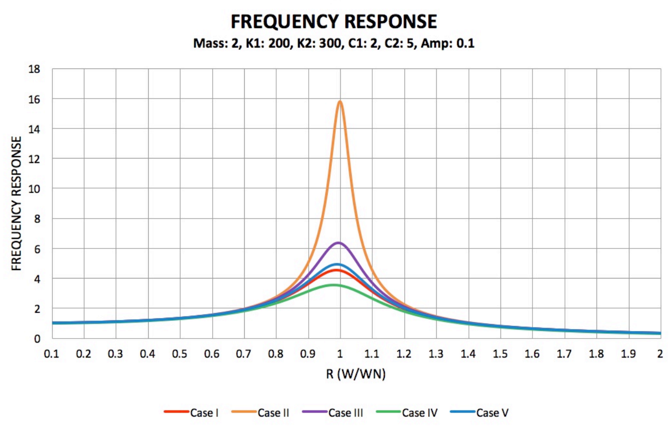

FIGURE 9. FREQUENCY RESPONSE

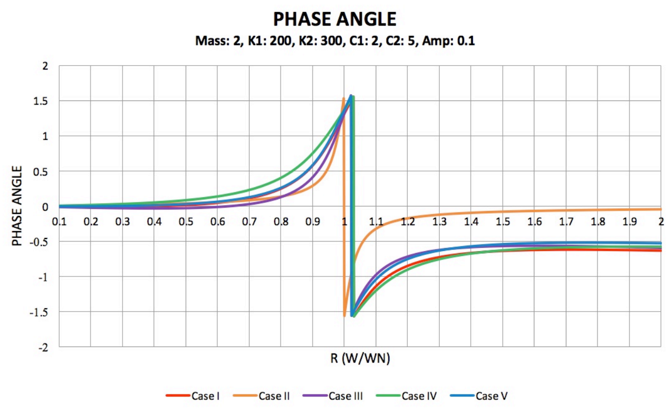

FIGURE 10. PHASE ANGLE

BEAM DESIGN

Finally, using this information and tools, we can design a composite graphite beam structure according to specific constraints. We are interested in building a cantilever beam structure as shown in Figure 11.

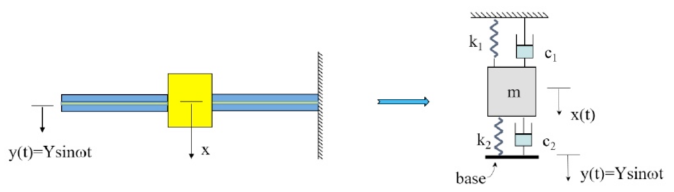

FIGURE 11. COMPOSITE BEAM DESIGN

This structure will be built out of layers of composite graphite designed to support a weight in the middle of two connected beams. Each layer of graphite will be separated from the other by a viscoelastic layer, which serves as a damper of sorts. Each graphite layer has a corresponding spring constant which allows the system to be modeled as a Case I resonator as shown in Figure 11.

The final length of the beam could next extend beyond 0.6 meters from the wall. Each beam segment could not exceed 0.5 meters, nor be less than 0.2 meters in length. The width of the beams could not exceed 2.5 cm, and the height of each layer was 150E-6 meters. The mass in the middle was not allowed to displace more than 1 cm from rest. The force experienced by the wall could not exceed 100 N.

The graphite layers had a Young's Modulus of 200 GPa, and the viscoelastic layers had a damping constant of 10 N-s/m. The mass weighed 1kg, and the amplitude of the forcing function was 2 cm. Under these conditions the beam was subjected driving frequency ranging from 1-100Hz.

The spring constant of each beam was determined by multiplying the Young's Modulus with the width times the thickness cubed. This was then divided by the length cubed times four (Equation 23).

$$k_{graphite} = \frac{E\,w\,t^3}{4\,L^3} \qquad (23)$$

Our final design was 0.6 meters in length, with each segment measuring 0.3 meters. We used only two layers of graphite on each segment, with one layer of viscoelastic material in between. This resulted in a spring constant of 1 N-m and damping constant of 10 N-s/m for each beam. By converting the driving frequencies to frequency ratios for the system we obtain a range of 0 to 70. The responses of this design are shown through Figures 12 through 18.

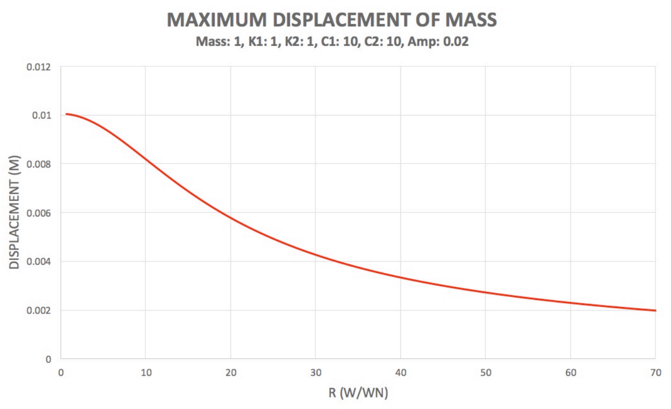

FIGURE 12. MAX DISPLACEMENT

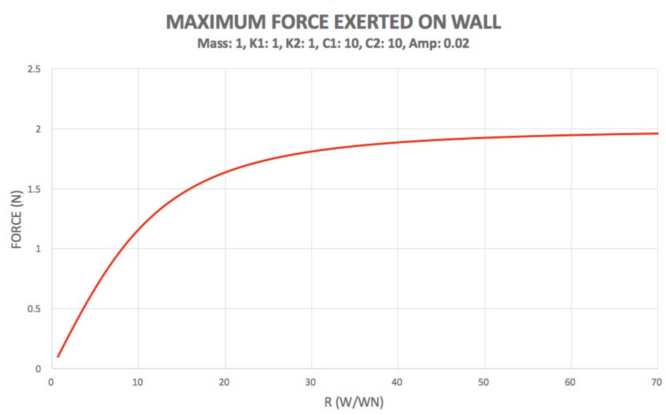

FIGURE 13. MAXIMUM FORCE AT WALL

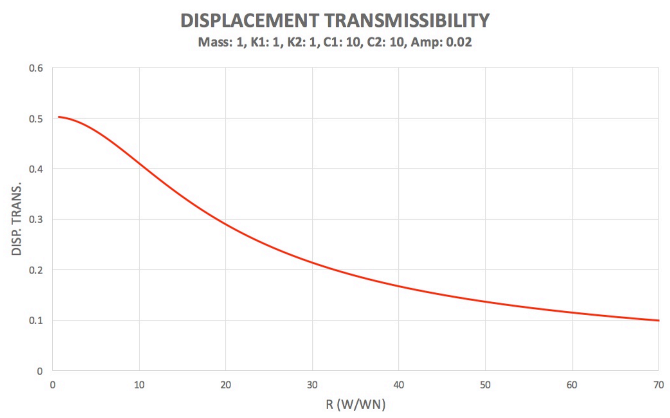

FIGURE 14. DISPLACEMENT TRANSMISSIBILITY

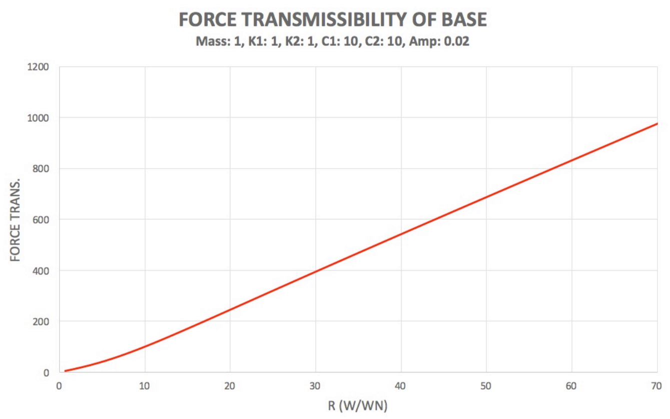

FIGURE 15. FORCE TRANSMISSIBILITY OF BASE

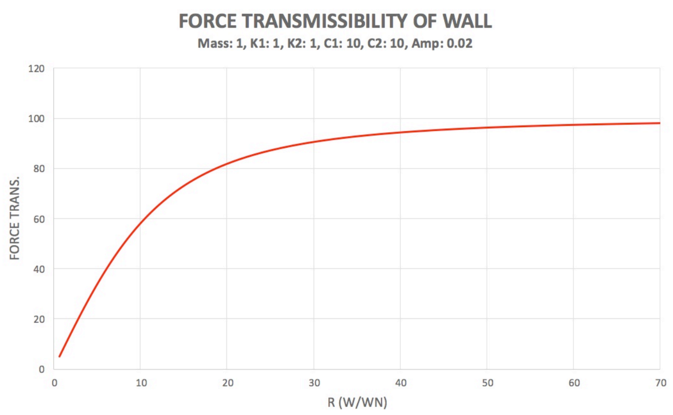

FIGURE 8. FORCE TRANSMISSIBILITY OF WALL

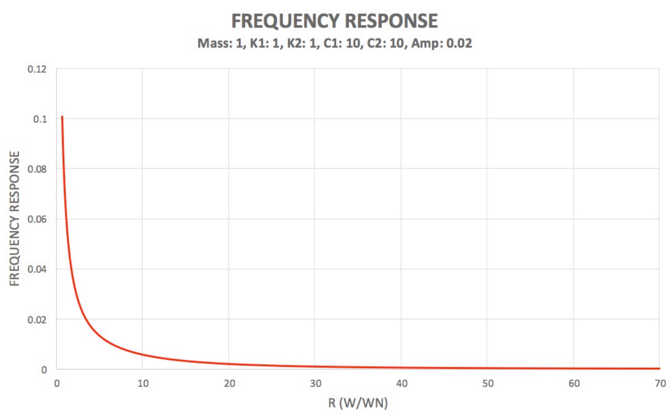

FIGURE 9. FREQUENCY RESPONSE

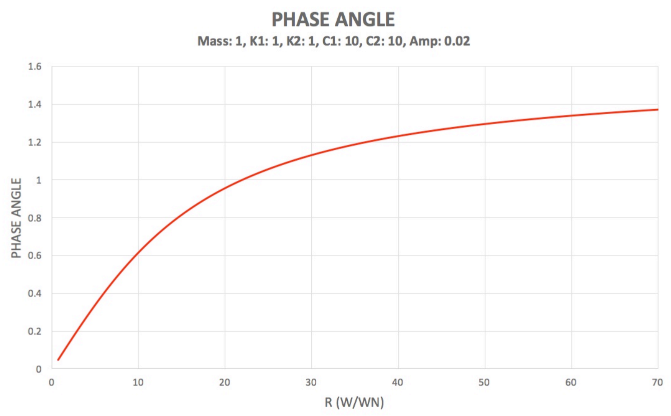

FIGURE 10. PHASE ANGLE

As seen from the graphs above, the design works as required. The mass initially moves 1 cm but never more, and the force on the wall never reaches 100 N.

It may be worthwhile to note that typically speaking, the Case I resonator has the lowest response of all resonators. If you want to design something that doesn’t amplify the input force the Case I resonator is ideal. On the other hand, if you want the system to amplify the input, the Case II resonator would be the best choice.

ACKNOWLEDGMENTS

I would like to thank Dr. Thomas Fronk for providing us with fantastic graphics a figures to use in this report. It made the process a lot less stressful.

ANNEX A

RESONATOR SOLVER CODE (C++)

```
//
//  main.cpp
//  Resonators - Vibrations
//
//  Created by Steven Proctor on 10/22/16.
//  Copyright © 2016 Steven Proctor. All rights reserved.
//

#include <iostream>
#include <string>
#include <sstream>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <fstream>
#include <vector>
#include "driving_frequency.hpp"

using namespace std;

void cinProperties(double &, double &, double &, double &, double
&, double &, double &);
void cinProperties(double &, double &, double &, double &, double
&, double &, double &, double &);
string outputResults(int, Resonator, double, double, double, double,
double, double, double);
void plotOutput(int);
void plotResults(double [], double [], int);
void toCSV(vector<double>, vector<double>, int, string, double,
double, double, double, double, double, string var);
void saveToFile(stringstream &);

int main() {

   double m, k1, k2, c1, c2, Y, r;
   char entry;
   int i = 0;
   stringstream singleFreq, temp;

   singleFreq.str("");

   cout << "Type any key to solve a Resonator System with a driving
frequency.\n(Type 'q' to skip)\n";
   cin >> entry;
   cout << endl;

   while (entry != 'q') {
      Resonator singleInput;
      temp.str("");

      cinProperties(m, k1, k2, c1, c2, r, Y);
      singleInput.solveResonator(m, k1, k2, c1, c2, r, Y);
      temp << outputResults(i, singleInput, m, k1, k2, c1, c2, r, Y);
      cout << temp.str();
      singleFreq << temp.str();

      i++;
      cout << "Type any key to enter another system ('q' to quit): ";
      cin >> entry;
      cout << endl << endl;
   }
   saveToFile(singleFreq);

   cout << "Type any key to solve a Resonator System with a range of
frequencies.\n(Type 'q' to skip)\n";
   cin >> entry;
   cout << endl;

   int run = 0;
   while (entry != 'q') {
      plotOutput(run++);
      cout << "Type any key to enter another system ('q' to quit): ";
      cin >> entry;
      cout << endl << endl;
   }


   return 0;
}


void cinProperties(double &m, double &k1, double &k2, double &c1,
double &c2, double &r, double &Y) {
   cout << "Enter mass: ";
   cin >> m;
   cout << "Enter k1: ";
   cin >> k1;
   cout << "Enter k2: ";
   cin >> k2;
   cout << "Enter c1: ";
   cin >> c1;
   cout << "Enter c2: ";
   cin >> c2;
   cout << "Enter r: ";
   cin >> r;
   cout << "Enter Amplitude: ";
   cin >> Y;

   return;
}

void cinProperties(double &m, double &k1, double &k2, double &c1,
double &c2, double &Y, double & r_i, double & r_f) {
   cout << "Enter mass: ";
   cin >> m;
   cout << "Enter k1: ";
   cin >> k1;
   cout << "Enter k2: ";
```

```
    cin >> k2;
    cout << "Enter c1: ";
    cin >> c1;
    cout << "Enter c2: ";
    cin >> c2;
    cout << "Enter Amplitude: ";
    cin >> Y;
    cout << "Enter the range of r: \n";
    cout << "r_i: ";
    cin >> r_i;
    cout << "r_f: ";
    cin >> r_f;

    return;
}

string outputResults(int i, Resonator R, double m, double k1, double
k2, double c1, double c2, double r, double Y) {
    stringstream ss;

    ss << endl;
    ss << "Mass: " << m << ", r: " << r << ", Amp: " << Y << endl;
    ss << "k1: " << k1 << ", k2: " << k2 << ", c1: " << c1 << ", c2: " <<
c2 << endl;
    ss << "********* Solution *********" << endl;
    ss << "Natural Frequency: " << R.get_w_n() << endl;
    ss << "Damping Ratio: " << R.get_z() << endl;
    ss << "Static Amplitude: " << R.get_d_st() << endl;
    ss << "Frequency Response: " << R.get_H() << endl;
    ss << "Phase Angle: " << R.get_phi() << endl;
    ss << "Displacement Transmissibility: " << R.get_T_d() << endl;
    ss << "Force Transmissibility to the base: " << R.get_Ft_base() <<
endl;
    ss << "Force Transmissibility to the wall: " << R.get_Ft_wall() <<
endl;
    ss << endl;

    return ss.str();
}

void plotOutput(int run) {
    int numPoints = 1000;
    vector<double> r, H, phi, T_d, Ft_base, Ft_wall, X_p, Fmax_wall;
    double m, k1, k2, c1, c2, Y, r_i, r_f;
    double step = 0;
    stringstream H_title, phi_title, T_d_title, Ft_base_title, Ft_wall_title,
X_p_title, Fmax_wall_title;

    r.clear();
    H.clear();
    phi.clear();
    T_d.clear();
    Ft_base.clear();
    Ft_wall.clear();
    X_p.clear();

    cinProperties(m, k1, k2, c1, c2, Y, r_i, r_f);
    step = (r_f - r_i) / double(numPoints);

    for (int i = 0; i <= numPoints; i ++) {
        r.push_back(r_i + (step * i));
    }
    for (int i = 0; i <= numPoints; i++) {
        Resonator Loop;
        Loop.solveResonator(m, k1, k2, c1, c2, r[i], Y);
        H.push_back(Loop.get_H());
        phi.push_back(Loop.get_phi());
        T_d.push_back(Loop.get_T_d());
        Ft_base.push_back(Loop.get_Ft_base());
        Ft_wall.push_back(Loop.get_Ft_wall());
        X_p.push_back(Loop.get_X_p());
        Fmax_wall.push_back(Loop.get_FT_wall());
    }

    H_title << "Frequency_Response" << run << ".csv";
    phi_title << "Phase_Angle" << run << ".csv";
    T_d_title << "Displacement_Transmissibility" << run << ".csv";
    Ft_base_title << "Force_Transmissibility_Base" << run << ".csv";
    Ft_wall_title << "Force_Transmissibility_Wall" << run << ".csv";
    X_p_title << "Max_Amplitude" << run << ".csv";
    Fmax_wall_title << "Max_Force_Wall" << run << ".csv";

    toCSV(r, H, numPoints, H_title.str(), m, k1, k2, c1, c2, Y, "H");
    toCSV(r, phi, numPoints, phi_title.str(), m, k1, k2, c1, c2, Y, "phi");
    toCSV(r, T_d, numPoints, T_d_title.str(), m, k1, k2, c1, c2, Y,
"Disp. Trans.");
    toCSV(r, Ft_base, numPoints, Ft_base_title.str(), m, k1, k2, c1, c2,
Y, "Force Trans. Base");
    toCSV(r, Ft_wall, numPoints, Ft_wall_title.str(), m, k1, k2, c1, c2,
Y, "Force Trans. Wall");
    toCSV(r, X_p, numPoints, X_p_title.str(), m, k1, k2, c1, c2, Y,
"X_p");
    toCSV(r, Fmax_wall, numPoints, Fmax_wall_title.str(), m, k1, k2,
c1, c2, Y, "Max Force Wall");

    cout << endl << "******** Solution printed to CSV file
********\n" << endl;

    return;
}

void toCSV(vector<double> r, vector<double> y, int size, string
fileName, double m, double k1, double k2, double c1, double c2,
double Y, string var) {
    ofstream fout;
    fout.open(fileName);

    fout << fileName << ",\n\n\n";
    fout << "mass: " << m << ",";
    fout << "k1: " << k1 << ",";
    fout << "k2: " << k2 << ",";
    fout << "c1: " << c1 << ",";
    fout << "c2: " << c2 << ",";
    fout << "amp: " << Y << ",\n\n\n";

    fout << "r," << var << ",\n";
    for (int i = 0; i <= size; i++) {
        fout << r[i];
        fout << ",";
        fout << y[i];
        fout << ",";
        fout << "\n";
    }
    fout.close();

    return;
}

void saveToFile(stringstream & ss) {
    ofstream fout;
    fout.open("Driving_Frequency_Response.txt");
    fout << ss.str();
    fout.close();
        }


//
//  driving_frequency.hpp
//  Resonators - Vibrations
//
//  Created by Steven Proctor on 10/24/16.
//  Copyright © 2016 Steven Proctor. All rights reserved.
//

#ifndef driving_frequency_hpp
#define driving_frequency_hpp

#include <stdio.h>
#include <iostream>
#include <math.h>

using namespace std;

class Resonator {
public:

    double m, k1, k2, c1, c2, r, Y;

    Resonator();
    ~Resonator();

    void solveResonator(double, double, double, double, double,
double, double);
    double get_w_n() { return w_n; }
    double get_z() { return z; }
    double get_d_st() { return d_st; }
    double get_H() { return H; }
    double get_phi() { return phi; }
    double get_T_d() { return T_d; }
    double get_Ft_base() { return Ft_base; }
    double get_Ft_wall() { return Ft_wall; }
    double get_X_p() { return X_p; }
    double get_FT_wall() { return FT_wall; }

private:

    double k_eq, c_eq, w_n, w, z, F_0, alpha, X_p, phi;
    double T_d, H, FT_base, Ft_base, FT_wall, Ft_wall, d_st;

    double springEquivilent();
    double damperEquivilent();
    double naturalFrequency();
    double drivingFrequency();
    double dampingRatio();
    double staticAmplitude();
    double initialForce();
    double phaseAngle_force();
    double amplitude_particularSolution();
    double phaseAngle_particularSolution();
    double displacementTransmissibility();
    double frequencyResponse();
    double maxForce_base();
    double forceTransmissibility_base();
    double maxForce_wall();
    double forceTransmissibility_wall();

};

#endif /* driving_frequency_hpp */


//
//  driving_frequency.cpp
//  Resonators - Vibrations
//
//  Created by Steven Proctor on 10/24/16.
//  Copyright © 2016 Steven Proctor. All rights reserved.
//

#include "driving_frequency.hpp"

Resonator::Resonator() {
    m = 0;
    k1 = 0;
    k2 = 0;
    c1 = 0;
    c2 = 0;
    Y = 0;
    r = 0;
}

Resonator::~Resonator() {

}

void Resonator::solveResonator(double mass, double spring1, double
spring2, double damper1, double damper2, double ratio, double
amplitude) {
    m = mass;
    k1 = spring1;
    k2 = spring2;
    c1 = damper1;
    c2 = damper2;
    r = ratio;
    Y = amplitude;
    k_eq = springEquivilent();
    c_eq = damperEquivilent();
    w_n = naturalFrequency();
    w = drivingFrequency();
    z = dampingRatio();
    F_0 = initialForce();
    alpha = phaseAngle_force();
    X_p = amplitude_particularSolution();
    phi = phaseAngle_particularSolution();
    H = frequencyResponse();
    FT_base = maxForce_base();
    Ft_base = forceTransmissibility_base();
    FT_wall = maxForce_wall();
    Ft_wall = forceTransmissibility_wall();
    d_st = staticAmplitude();
    T_d = displacementTransmissibility();
}

double Resonator::springEquivilent() {
    return k1 + k2;
}

double Resonator::damperEquivilent() {
    return c1 + c2;
}

double Resonator::naturalFrequency() {
    return sqrt(k_eq / m);
}

double Resonator::drivingFrequency() {
    return w_n * r;
}

double Resonator::dampingRatio() {
    return (c_eq / (2 * sqrt(m * k_eq)));
}


double Resonator::initialForce() {
    return Y * sqrt(k2 * k2 + pow((c2 * w), 2));
}

double Resonator::phaseAngle_force() {
    return atan(-(c2 * w) / k2);
}

double Resonator::amplitude_particularSolution() {
    return (F_0 / k_eq) / sqrt(pow((1 - r * r), 2) + pow((2 * z * r), 2));
}

double Resonator::phaseAngle_particularSolution() {
    return atan((r * (2 * z * k2 - c2 * w_n * (1.0 - r * r))) / (k2 * (1 - r *
r) + 2 * c2 * w_n * z * r * r));
}

double Resonator::displacementTransmissibility() {
    return X_p / Y;
}

double Resonator::frequencyResponse() {
    return X_p / (F_0 / k_eq);
}

double Resonator::maxForce_base() {
    return -X_p * sqrt(pow((k1 - k_eq * r * r), 2) + pow((c1 * w_n * r),
2));
}

double Resonator::forceTransmissibility_base() {
    return -FT_base / (Y * k2);
}

double Resonator::maxForce_wall() {
    return X_p * sqrt(k1 * k1 + pow((c1 * r * w_n), 2));
}

double Resonator::forceTransmissibility_wall() {
    return FT_wall / (Y * k2);
}

double Resonator::staticAmplitude() {
    return X_p * sqrt(pow((1 - r * r), 2) + pow((2 * z * r), 2));
}
```