Euler-Bernoulli Beam Project
CSCI 4650
Ameen Sassi
Christopher Renden

Problem 1:

driver.m

```
global L w d g n rho E I p mDiver;
L = 2;
w = 0.3;
d = 0.03;
g = -9.81;
n = 10;
rho = 480;
E = 1.3e+10;
p = 100;
mDiver = 70;

I = w*d*d*d/12;
h = L/n;


%Problem 1
sm = structuremat(n);
f = beamforces(@gravity,n,L/n);
def = cat(1, [0], (sm\f));  %def = deflection at each grid point
def = def*h*h*h*h/E/I;
disp("Problem 1:");
disp(def);
```

structuremat.m

```
function mat = structuremat(n)
    mat = sparse(n,n);
    mat(1,1) = 16;
    mat(1,2) = -9;
    mat(1,3) = 8/3;
    mat(1,4) = -1/4;
    mat(2,1) = -4;
    mat(2,2) = 6;
    mat(2,3) = -4;
    mat(2,4) = 1;
    for y = 3:n-2
        mat(y, y-2) = 1;
        mat(y, y-1) = -4;
        mat(y, y) = 6;
        mat(y, y+1) = -4;
        mat(y, y+2) = 1;
    end
    mat(n-1,n-3) = 16/17;
    mat(n-1,n-2) = -60/17;
    mat(n-1,n-1) = 72/17;
    mat(n-1,n) = -28/17;
    mat(n,n-3) = -12/17;
    mat(n,n-2) = 96/17;
    mat(n,n-1) = -156/17;
    mat(n,n) = 72/17;
end
```

For activity 1, we were asked to define the matrix in 2.34 (from the book) and then solve the system for the displacements. We implemented a function structuremat which constructs the structure matrix for a given number of grid steps n. By solving the structure matrix for a vector (in this case of constants) representing the force on the board, and multiplying by h^4/EI, we get an approximation for the deflection of the board at each position. Note that for completeness sake we concatenate 0 to the start of this list of deflections to represent the deflection at x=0.

In the case of n=10, the calculated deflection is:

```
 0
 -0.0001806
 -0.0006748
 -0.001417
 -0.002349
 -0.003421
 -0.004590
 -0.005822
 -0.007088
 -0.008372
 -0.009659
```

Giving a total deflection of just under a centimeter at the end.

Problem 2:

driver.m

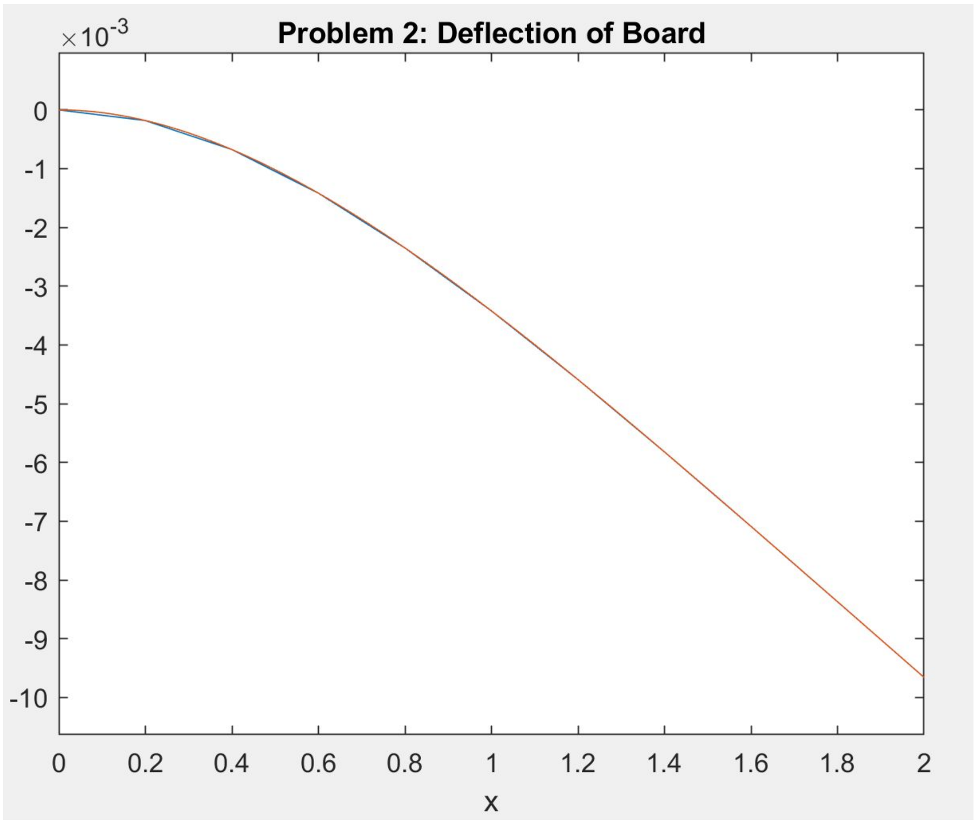
```
%Problem 2
plot(0:h:L, def); hold on
disp("Problem 2:");
rel = ((dis(L)-def(n+1))/dis(L));
fprintf("\tRelative Error: %d\n", rel);
clf;
ezplot(@dis, [0 L]);
title("Problem 2: Deflection of Board");
```

driver.m

```
%Known closed-form solution given constant load
function out = dis(x)
    global E I L;
    f = gravity(0);
    out = (f/(24*E*I))*(x*x)*((x*x)-(4*L*x)+(6*(L*L)));
end
```

For activity 2, we were asked to simply plot our solution from activity 1 against the correct solution. In order to do this, we first needed to implement the correct solution into the program. A function, dis(x), was created and the solution was implemented utilizing the previously created gravity function, a few global variables, and the formula from the book. The next step was plotting def and dis against each other with the plot and ezplot functions. Activity 2 also asks that we check the error at the end of the beam. Our professor recommended finding the relative error, so we calculated the relative error and placed it in a variable named rel.

When the solutions are plotted against each other they appear as a single line. This is because the numbers are so close and the error is so small that it confirms our solution is correct.



```
Problem 2:
    Relative Error: -1.975547e-14
```

Problem 3

driver.m

```
%Problem 3
disp("Problem 3:");
fprintf("\tn\t|\tRelative Error\t|\tCondition Number\ n");
err = zeros(12,1);
for i=0:11
    n = 10*(2^i);
    h = L/n;
    sm = structuremat(n);
    f = beamforces(@gravity,n,L/n);
    def = cat(1, [0], (sm\f));
    def = def*h*h*h*h/E/I;
    err(i+1) = abs((dis(L)-def(n+1))/dis(L));
    fprintf("\t%d\t|\t%d\t|\t%d\n", n, err(i+1), cond(sm));
end
clf;
semilogy(0:11, err);
title("Problem 3: R.E. of Unloaded Board with 10*2^x Samples");
```

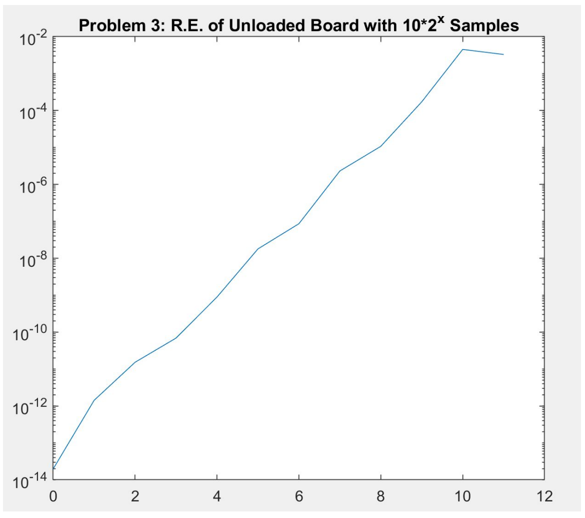
beamforces.m

```
function force = beamforces(func, count, step)
    force = zeros(count,1);
    position = 0;
    for i = 1:count
       force(i,1) = func(position);
       position = position + step;
    end
end
```

For activity 3, we are asked rerun our calculations from activity 1 but with new n variables. We first created a matrix named err and used the function zeros(12, 1) to set err to a 12 by 1 matrix of 0s. This matrix will be used to contain the errors at x = L for each n. Then we contained our code from activity 1 in a for loop that will run 12 times. Next we added n to the for loop and set it to the specified value: 10*(2^i), with i being 0 through 11. At the end of each loop, we calculate the relative error for each of the n values and place it within the err matrix. After the for loop completes, we graph a semilogarithmic plot to go along with our table.

The table and graph :

```
Problem 3:
     n    |   Relative Error  |   Condition Number
    10    |   1.975547e-14     |    3.325437e+04
    20    |   1.424190e-12     |    5.303025e+05
    40    |   1.525248e-11     |    8.449280e+06
    80    |   6.898753e-11     |    1.348213e+08
    160   |   8.913843e-10     |    2.153877e+09
    320   |   1.789781e-08     |    3.443465e+10
    640   |   8.613172e-08     |    5.507300e+11
1280      |   2.307145e-06     |    8.809861e+12
2560      |   1.075480e-05     |    1.409426e+14
5120      |   1.724371e-04     |    2254886342269035
10240     |   4.523717e-03     |    36072572533103696
20480     |   3.297542e-03     |    577022291760933760
```



It can be seen that the smallest error corresponds to when the n value is 20 (the 10 value occurs when i = 0 and the question asks for 1 through 11. The n = 10 was only used for a point of reference). It can also be seen that when the n value is larger, so too is the error. This is due to the size of the matrix. Solving the matrix with constant force will always give the correct answer no matter the n. The error is introduced by the computation itself and as the matrix grows, more computations occur and that means more error. This can be seen in the fact that the relative error is directly proportional to the condition number of the structure matrix.

Problem 4

driver.m

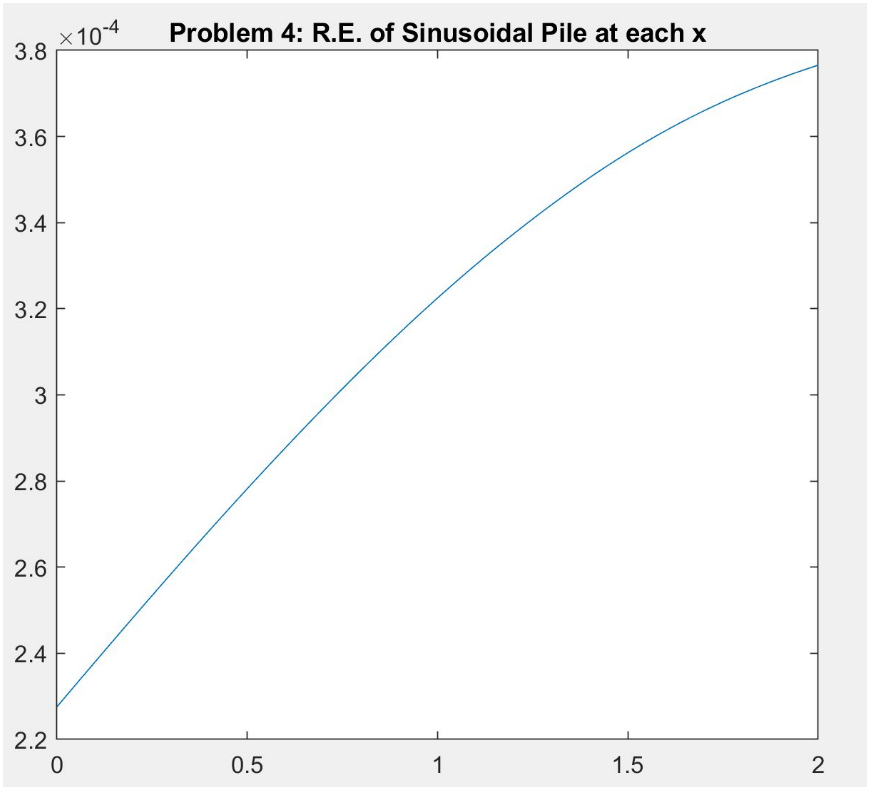
```
%Problem 4
disp("Problem 4:");
n = 5120;
h = L/n;
sm = structuremat(n);
f = beamforces(@pile,n,L/n);
def = cat(1, [0], (sm\f));  %def = deflection at each grid point
def = def*h*h*h*h/E/I;
err = zeros(n+1,1);
for i=1:n+1
    err(i) = abs((correctsin((i-1)*h)-def(i))/correctsin((i-1)*h));
end
clf;
plot(0:h:L, err);
title("Problem 4: R.E. of Sinusoidal Pile at each x");
```

correctsin.m

```
function out = correctsin(x)
    global L w d g p E I rho;
    grav = rho*w*d*g;

    out = grav/24/E/I;
    out = out * x*x;
    out = out * (x*x - 4*L*x + 6*L*L);

    t2 = p*g*L/E/I/pi;
    t2 = t2 * (L*L*L/pi/pi/pi*sin(pi*x/L) - x*x*x/6 + L*x*x/2 - L*L/pi/pi*x);

    out = out+t2;
end
```

In activity 4, we are asked to prove a solution provided in the book satisfies the Euler-Bernoulli beam equation and the clamped-free boundary conditions. To do this, we created another function, correctsin(x), that would implement the given solution. We then construct a force vector by calculating the force at each point x with the function pile(x). After the force vector is constructed, we find and plot the error according to the given closed form.

Graph:



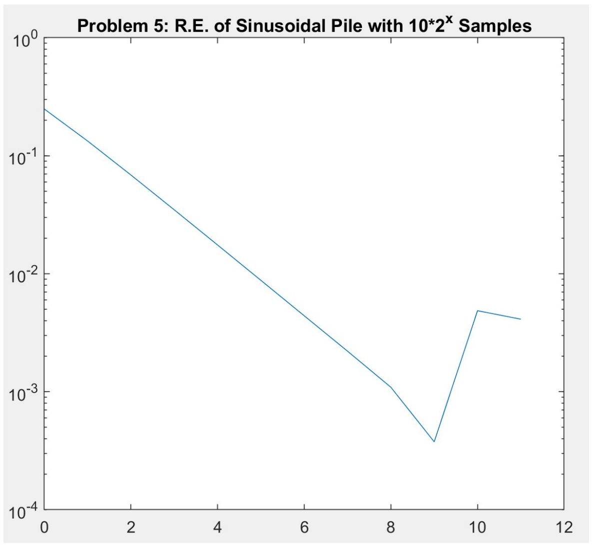
As it can be seen on the graph, the relative errors are quite small and thus prove the correctness of the given solution.

Problem 5

driver.m

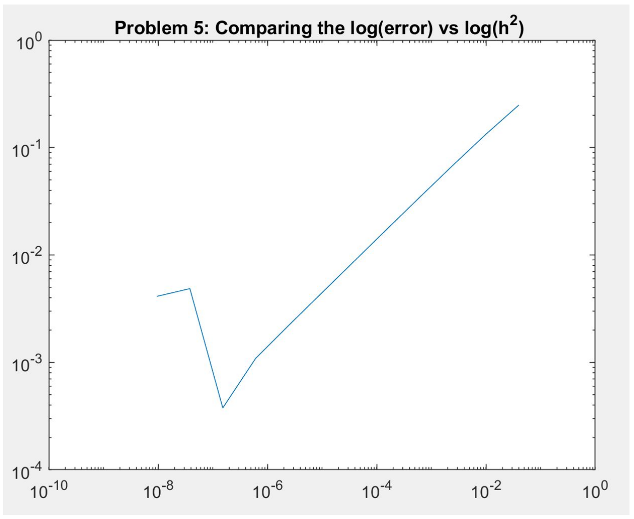
```
%Problem 5
disp("Problem 5:");
fprintf("\tn\t|\tRelative Error\t|\tCondition Number\n");
err = zeros(12,1);
hm = zeros(12,1);
for i=0:11
    n = 10*(2^i);
    h = L/n;
    sm = structuremat(n);
    f = beamforces(@pile,n,L/n);
    def = cat(1, [0], (sm\f));
    def = def*h*h*h*h/E/I;
    err(i+1) = abs((correctsin(L)-def(n+1))/correctsin(L));
    hm(i+1) = h^2;
    fprintf("\t%d\t|\t%d\t|\t%d\n", n, err(i+1), cond(sm));
end

clf;
semilogy(0:11, err); hold on
title("Problem 5: R.E. of Sinusoidal Pile with 10*2^x Samples");
pause;
title("Problem 5: Comparing the log(error) vs log(h^2) ");
plot(log(hm), log(err));

function out = pile(x)
    global p g L;
    out = p*g*sin(x*pi/L) + gravity(x);
end
```

In activity 5, we are tasked with re-doing activity 3 with the sinusoidal load. The only difference between how we completed activity 3 and 5 is that we use pile(x) for our force and calculate the relative error with the correct solution from activity 4. The table and graph are below:



```
Problem 5:
      n   |   Relative Error   |   Condition Number
      10  |   2.489651e-01     |   3.325437e+04
      20  |   1.332775e-01     |   5.303025e+05
      40  |   6.862665e-02     |   8.449280e+06
      80  |   3.479685e-02     |   1.348213e+08
      160 |   1.751846e-02     |   2.153877e+09
      320 |   8.789167e-03     |   3.443465e+10
      640 |   4.402166e-03     |   5.507300e+11
1280      |   2.205219e-03     |   8.809861e+12
2560      |   1.091179e-03     |   1.409426e+14
5120      |   3.765669e-04     |   2254886342269035
10240     |   4.861643e-03     |   36072572533103696
20480     |   4.120041e-03     |   577022291760933760
```

It can be seen that the smallest error corresponds to when the n value is 5120. It is also the best balance between smallest real error and computationally induced error. The number of errors increase with the size of n for the same reason as stated in activity 3. The larger the matrix, the more computations need to take place which causes a larger error. After 5120, the size of the matrix simply gets too large and the computations that occur gain more error as the bump up against machine round-off.



It can be seen by plotting Relative Error vs. h^2 on a log-log plot that the relative error is roughly proportional to h, rather than to h^2 (note that the slope of this plot is quite close to 1/2). Examining the error relative to the Condition Number, The error appears to be roughly inversely proportional to the square of the Condition Number, showing that the computationally-induced error is not the major factor in the error until we reach $n > 5120$.

Problem 6

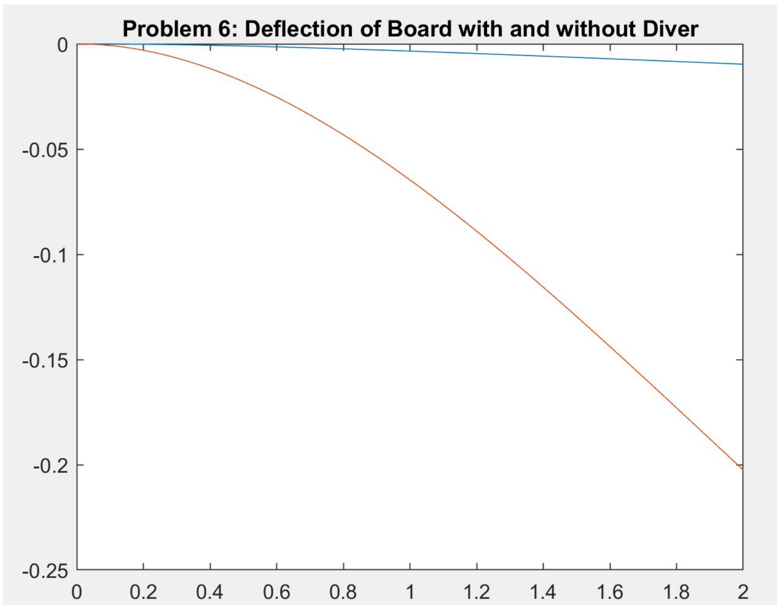
driver.m

```
%problem 6
disp("Problem 6:");
n = 5120;
h = L/n;
sm = structuremat(n);
f = beamforces(@diver,n,L/n);
divDef = cat(1, [0], (sm\f));
divDef = divDef*h*h*h*h/E/I;
f = beamforces(@gravity,n,L/n);
def = cat(1, [0], (sm\f));
def = def*h*h*h*h/E/I;
clf;
plot(0:h:L, def); hold on
plot(0:h:L, divDef); hold on
title("Problem 6: Deflection of Board with and without Diver");
fprintf("\tDeflection of end is: %d\n", divDef(n+1));

function out = diver(x)
    global g mDiver;
    out = gravity(x);
    if x < 1.8
        return;
    end
    out = out + g * (mDiver/.2);
end
```

In activity 6, we are asked to add 70kg diver to the beam and then calculate the deflection of the board with the optimal n value we found in activity 5. To do this, we create a function, diver(x), that sets the forces. Then we calculate the deflection of the board with n = 5120 and plot it on top of the deflection of the unladen board. This shows the difference the 70kg diver makes. The deflection of the end came out to be about 20 cm, which feels about right given our own experience with diving boards.

The graph and deflection are below:

```
Problem 6:
    Deflection of end is: -2.025104e-01
```


Problem 6: Deflection of Board with and without Diver
0
-0.05
-0.1
-0.15
-0.2
-0.25
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2