Massachusetts Institute of Technology
Department of Aeronautics and Astronautics
Cambridge, MA 02139

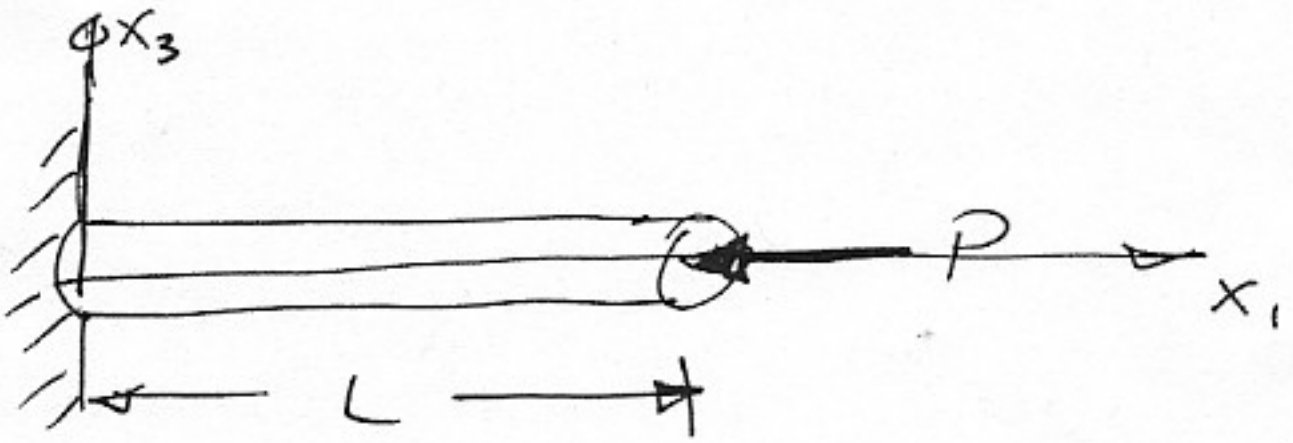
Unified Engineering
Spring 2005
Problem Set #10
Solutions

PAL
4/11/05
(1)

UNIFIED ENGINEERING

Problem Set #10 -- SOLUTIONS

M.10.



Cross section = A
Moment of Inertia = I
Modulus = E

The basic governing equation is:

$$\frac{d^2 u_3}{dx_1^2} + \frac{P}{EI} u_3 = 0 \qquad (1)$$

with the general homogeneous solution:

$$u_3 = A \sin\left(\sqrt{\frac{P}{EI}} x_1\right) + B \cos\left(\sqrt{\frac{P}{EI}} x_1\right) + C + D x_1 \qquad (2)$$

To determine the constants, the boundary conditions are needed.

- At the clamped end ($x_1 = 0$): $u_3 = 0$

  $$\frac{du_3}{dx_1} = 0$$

- At the free end with applied load ($x_1 = L$):

$$M = 0 \Rightarrow \frac{d^2 u_3}{dx_1^2} = 0$$

$$S = -P\frac{du_3}{dx_1}$$

$$\text{and } S = EI\frac{d^3 u_3}{dx_1^3}$$

$$\Rightarrow \frac{d^3 u_3}{dx_1^3} = -\frac{P}{EI}\frac{du_3}{dx_1}$$

To use these in the basic solution, need derivatives of that (eq. (2)). So:

$$u_3 = A\sin\left(\sqrt{\frac{P}{EI}}x_1\right) + B\cos\left(\sqrt{\frac{P}{EI}}x_1\right) + C + Dx_1$$

$$\frac{du_3}{dx_1} = \sqrt{\frac{P}{EI}}A\cos\left(\sqrt{\frac{P}{EI}}x_1\right) - \sqrt{\frac{P}{EI}}B\sin\left(\sqrt{\frac{P}{EI}}x_1\right) + D$$

$$\frac{d^2 u_3}{dx_1^2} = -\frac{P}{EI}A\sin\left(\sqrt{\frac{P}{EI}}x_1\right) - \frac{P}{EI}B\cos\left(\sqrt{\frac{P}{EI}}x_1\right)$$

$$\frac{d^3 u_3}{dx_1^3} = -\left(\frac{P}{EI}\right)^{3/2}A\cos\left(\sqrt{\frac{P}{EI}}x_1\right) + \left(\frac{P}{EI}\right)^{3/2}B\sin\left(\sqrt{\frac{P}{EI}}x_1\right)$$

Now apply each of the 4 Boundary conditions to get 4 equations:

@ $x_1 = 0$, $u_3 = 0 \Rightarrow B + C = 0$ (3)

@ $x_1 = 0$, $\frac{du_3}{dx_1} = 0 \Rightarrow \sqrt{\frac{P}{EI}}A + D = 0$ (4)

@ $x_1 = L$, $\frac{d^2 u_3}{dx_1^2} = 0 \Rightarrow -\frac{P}{EI}A\sin\left(\sqrt{\frac{P}{EI}}L\right) - \frac{P}{EI}B\cos\left(\sqrt{\frac{P}{EI}}L\right) = 0$

giving: $A\sin\left(\sqrt{\frac{P}{EI}}L\right) + B\cos\left(\sqrt{\frac{P}{EI}}L\right) = 0$ (5)

@ $x_1 = L$, $\frac{d^3 u_3}{dx_1^3} = \frac{-P}{EI}\frac{du_3}{dx_1}$

$$\Rightarrow -\left(\frac{P}{EI}\right)^{3/2} A\cos\left(\sqrt{\frac{P}{EI}}L\right) + \left(\frac{P}{EI}\right)^{3/2} B\sin\left(\sqrt{\frac{P}{EI}}L\right) = -\frac{P}{EI}\left[A\sqrt{\frac{P}{EI}}\cos\left(\sqrt{\frac{P}{EI}}L\right) - B\sqrt{\frac{P}{EI}}\sin\left(\sqrt{\frac{P}{EI}}L\right) + D\right] \quad (6)$$

Working on equation (6):

$$-A\cos\left(\sqrt{\frac{P}{EI}}L\right) + B\sin\left(\sqrt{\frac{P}{EI}}L\right) = -A\cos\left(\sqrt{\frac{P}{EI}}L\right) + B\sin\left(\sqrt{\frac{P}{EI}}L\right) + \frac{D}{(P/EI)^{1/2}}$$

giving: $\boxed{D = 0}$

use this in equation (4) to get:

$$\sqrt{\frac{P}{EI}}A = 0 \Rightarrow \boxed{A = 0}$$

using this in equation (5), we get:

$$(0)\cdot\sin\left(\sqrt{\frac{P}{EI}}L\right) + B\cos\left(\sqrt{\frac{P}{EI}}L\right) = 0$$

$$\Rightarrow B\cos\left(\sqrt{\frac{P}{EI}}L\right) = 0$$

This gives 2 possible solutions:

$B = 0$ (trivial)

or

$$\cos\left(\sqrt{\frac{P}{EI}}L\right) = 0$$

$$\Rightarrow \sqrt{\frac{P}{EI}}L = \frac{n\pi}{2} \quad (\text{for } n \text{ odd})$$

So: $\boxed{P = \frac{n^2\pi^2}{4L^2}EI}$ (for n odd)

The lowest load occurs for $n=1$

$$\Rightarrow \boxed{P_{cr} = \frac{\pi^2 EI}{4L^2}} \qquad \underline{\text{Buckling load}}$$

In general, the form for P_{cr} is:

$$P_{cr} = c\frac{\pi^2 EI}{L^2}$$

where: c = coefficient of (end) fixity

So for this case, $c = 1/4$

To get the mode, go to equation (3) giving:

$$B + C = 0$$

$$\Rightarrow B = -C$$

using this in the expression for u_3, equation (2), along with the other results:

$$u_3 = B\cos\left(\sqrt{\frac{P}{EI}}x_1\right) - B$$

$$\Rightarrow u_3 = B\left[\cos\left(\sqrt{\frac{P}{EI}}x_1\right) - 1\right]$$

using the result for $P_{(cr)}$:

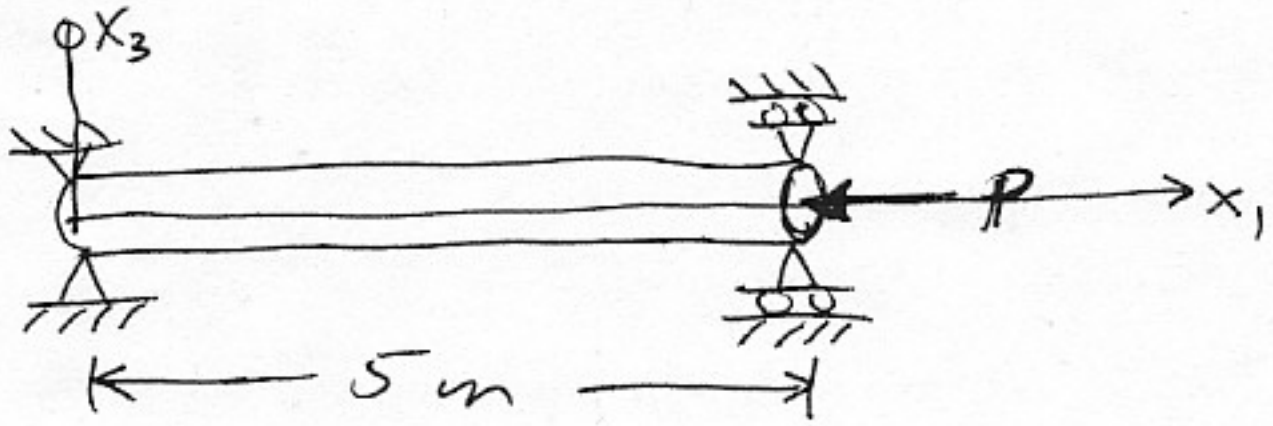
$$u_3 = B\left[\cos\left(\sqrt{\frac{n^2\pi^2 EI/4L^2}{EI}}x_1\right) - 1\right]$$

$$= B\left[\cos\left(\sqrt{\frac{n^2\pi^2}{4L^2}}x_1\right) - 1\right]$$

for the $n=1$, P_{cr} case:

$$\boxed{u_3 = B\left[\cos\left(\frac{\pi}{2L}x_1\right) - 1\right]} \qquad \underline{\text{buckling mode}}$$

M 11.



a) Model this as a simply-supported column

For a simply-supported configuration:

$$P_{cr} = \frac{\pi^2 EI}{L^2}$$

for a solid circular cross-section, one can find (e.g. Handout M-6)

$$I = \frac{\pi R^4}{4}$$

with $D = 2R \Rightarrow R = D/2$

gives: $$I = \frac{\pi (D/2)^4}{4} = \frac{\pi D^4}{64}$$

Now need a number for E. For steel, modulus is ~ 200 GPa. So using these values:

$$P_{cr} = \frac{\pi^2 (200 \times 10^9 \frac{N}{m^2})(\frac{\pi D^4}{64})}{(5m)^2}$$

Working through this gives:

$$\boxed{P_{cr} = \pi^3 (1.25 \times 10^8) D^4}$$

D in [m]
P in [N]

(5) To determine the squashing load, the material compressive ultimate is needed.

For D6ac steel: $\sigma_{cu} = 1375\ MPa$

Had: $\frac{P_{squash}}{A} = \sigma_{cu}$

and: $A = \pi R^2 = \pi (D/2)^2 = \frac{\pi D^2}{4}$

so: $P_{sq} = (1375 \times 10^6 \frac{N}{m^2})(\frac{\pi D^2}{4})$

$$\Rightarrow \boxed{P_{sq} = 1080 \times 10^6 D^2}$$

D in [m]
P in [N]

one can also determine the start of a "transition" zone via:

$$\frac{P_{transition}}{A} = \sigma_{cy}$$

Here: $\sigma_{cy} = 1305\ MPa$

$$\Rightarrow P_{trans} = (1305 \times 10^6 \frac{N}{m^2})(\frac{\pi D^2}{4})$$

$$\Rightarrow \boxed{P_{trans\ (yield)} = 1025 \times 10^6 D^2}$$

D in [m]
P in [N]

(c) The key to drawing the design chart is to determine the points (P and D) where the mode of failure goes from "buckling" to "transition" to "crushing/squashing". Do this by equating the buckling case with the latter two, solving for D, and substituting the result to get P. Then plot each curve.

Summarizing:

(A) Buckling: $P_{cr} = 3.876 \times 10^9 D^4$

(B) Transition (yielding): $P_{trans} = 1.025 \times 10^9 D^2$

(C) Squashing: $P_{sq} = 1.080 \times 10^9 D^2$

Going from (A) to (B):

$$3.876 \times 10^9 D^4 = 1.025 \times 10^9 D^2$$

$$\Rightarrow D^2 = 0.264$$

$$\Rightarrow D = 0.514 \text{ m}$$

giving: $P = 2.71 \times 10^8 \text{ N}$

Going from (A) to (C):

$$3.876 \times 10^9 D^4 = 1.080 \times 10^9 D^2$$

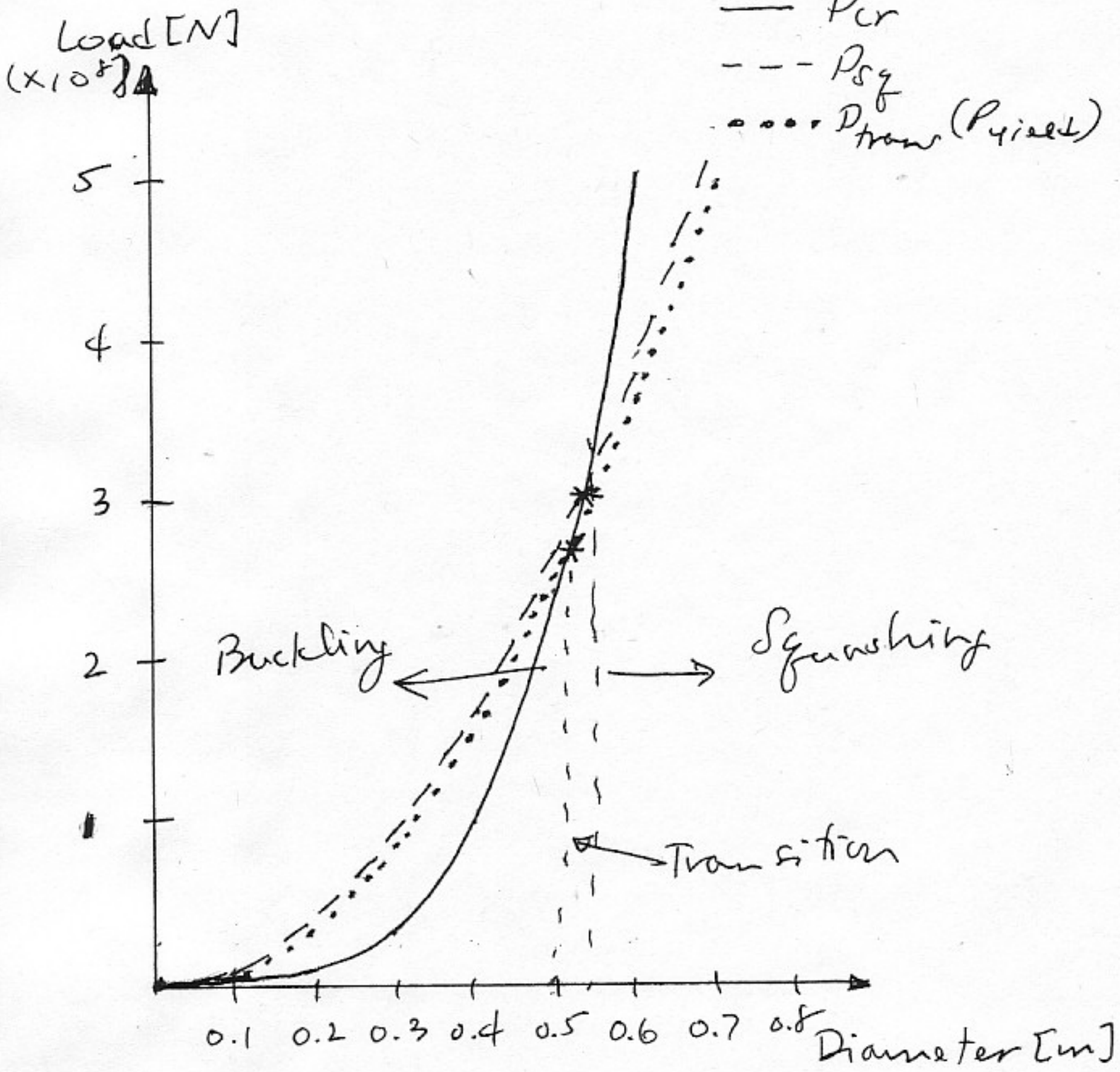
$$\Rightarrow D^2 = 0.279$$

$$\Rightarrow D = 0.528 \text{ m}$$

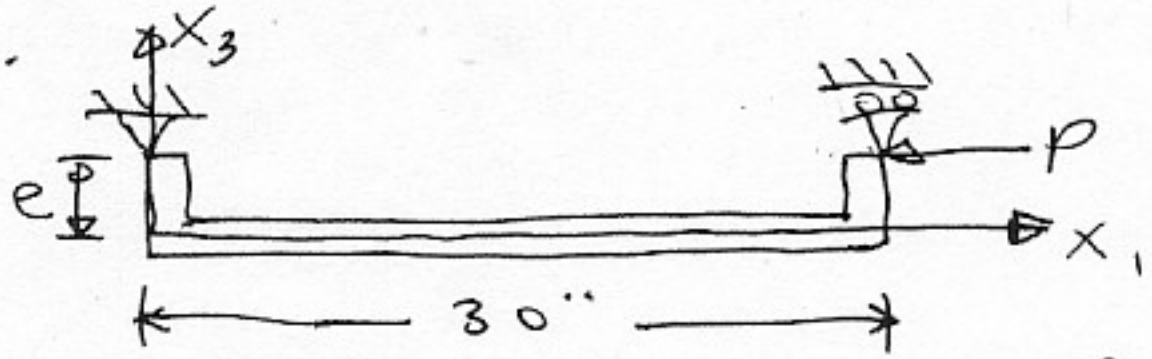
giving: $P = 3.01 \times 10^8 \text{ N}$

Now draw the plots of each curve and label these key points

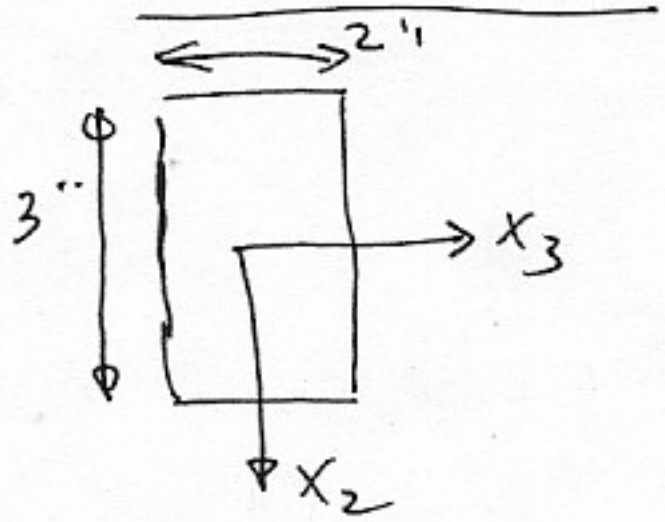
Design Chart



M12.



Cross-Section



(a) The maximum load is the limit placed by the buckling load. This does not change due to the eccentric loading.

This is a simply-supported configuration, so:

$$P_{cr} = \frac{\pi^2 E I}{L^2}$$

Have $E = 10.3\,Msi = 10.3 \times 10^6 \frac{lbf}{in^2}$

Need moment of inertia I. For a rectangular cross-section:

$$I = \frac{bh^3}{12}$$

The structure will buckle in the direction with the lowest I so where "h" is smallest:

$$\Rightarrow I = \frac{(3\,in)(2\,in)^3}{12} = 2.0\,in^4$$

This gives:

$$P_{cr} = \frac{\pi^2 (10.3 \times 10^6 \frac{lbf}{in^2})(2.0\ in^4)}{(30\ in)^2}$$

$$\Rightarrow \boxed{P_{cr} = 2.259 \times 10^5\ lbf}$$

Check σ_{cr} to see if it is below σ_{cy} and σ_{cu}:

$$\sigma_{cr} = \frac{P_{cr}}{A} = \frac{2.259 \times 10^5\ lbf}{(3\ in)(2\ in)} = 3.765 \times 10^4 \frac{lbf}{in^2}$$

$$= 37.7\ ksi$$

and this is well below the yield and ultimate stresses

(6) For the case of a simply-supported configuration loaded eccentrically, the governing equation is:

$$u_3 = e\left[\frac{(1-\cos\sqrt{\frac{P}{EI}}L)}{\sin\sqrt{\frac{P}{EI}}L}\sin\sqrt{\frac{P}{EI}}x_1 + \cos\sqrt{\frac{P}{EI}}x_1 - 1\right]$$

use the pertinent values of P_{cr}, E, I, and L and x_1 determine the deflection at the column center, set $x_1 = 15$ in.

Normalize that deflection by the length and the applied load by the critical load.

To do this

Multiply P by $\frac{P_{cr}}{P_{cr}} = \frac{\pi^2 EI}{P_{cr} L^2}$

$$\Rightarrow \sqrt{\frac{P}{EI}} = \sqrt{\frac{P}{EI} \cdot \frac{\pi^2 EI}{P_{cr} L^2}} = \sqrt{\frac{P}{P_{cr}} \frac{\pi^2}{L^2}}$$

$$\text{so: } \sqrt{\frac{P}{EI}} = \frac{\pi}{L} \sqrt{\frac{P}{P_{cr}}}$$

Put this back into the earlier equation to get:

$$u_3 = e \left[\frac{1 - \cos\left(\frac{\pi}{L}\sqrt{\frac{P}{P_{cr}}}\, L\right)}{\sin\left(\frac{\pi}{L}\sqrt{\frac{P}{P_{cr}}}\, L\right)} \sin\left(\frac{\pi}{L}\sqrt{\frac{P}{P_{cr}}}\, x_1\right) + \cos\left(\frac{\pi}{L}\sqrt{\frac{P}{P_{cr}}}\, x_1\right) - 1 \right]$$

continuing on and dividing through by L:

$$\frac{u_3}{L} = \frac{e}{L} \left[\frac{1 - \cos \pi\sqrt{\frac{P}{P_{cr}}}}{\sin \pi\sqrt{\frac{P}{P_{cr}}}} \sin\left(\pi\sqrt{\frac{P}{P_{cr}}}\, \frac{x_1}{L}\right) + \cos\left(\pi\sqrt{\frac{P}{P_{cr}}}\, \frac{x_1}{L}\right) - 1 \right]$$

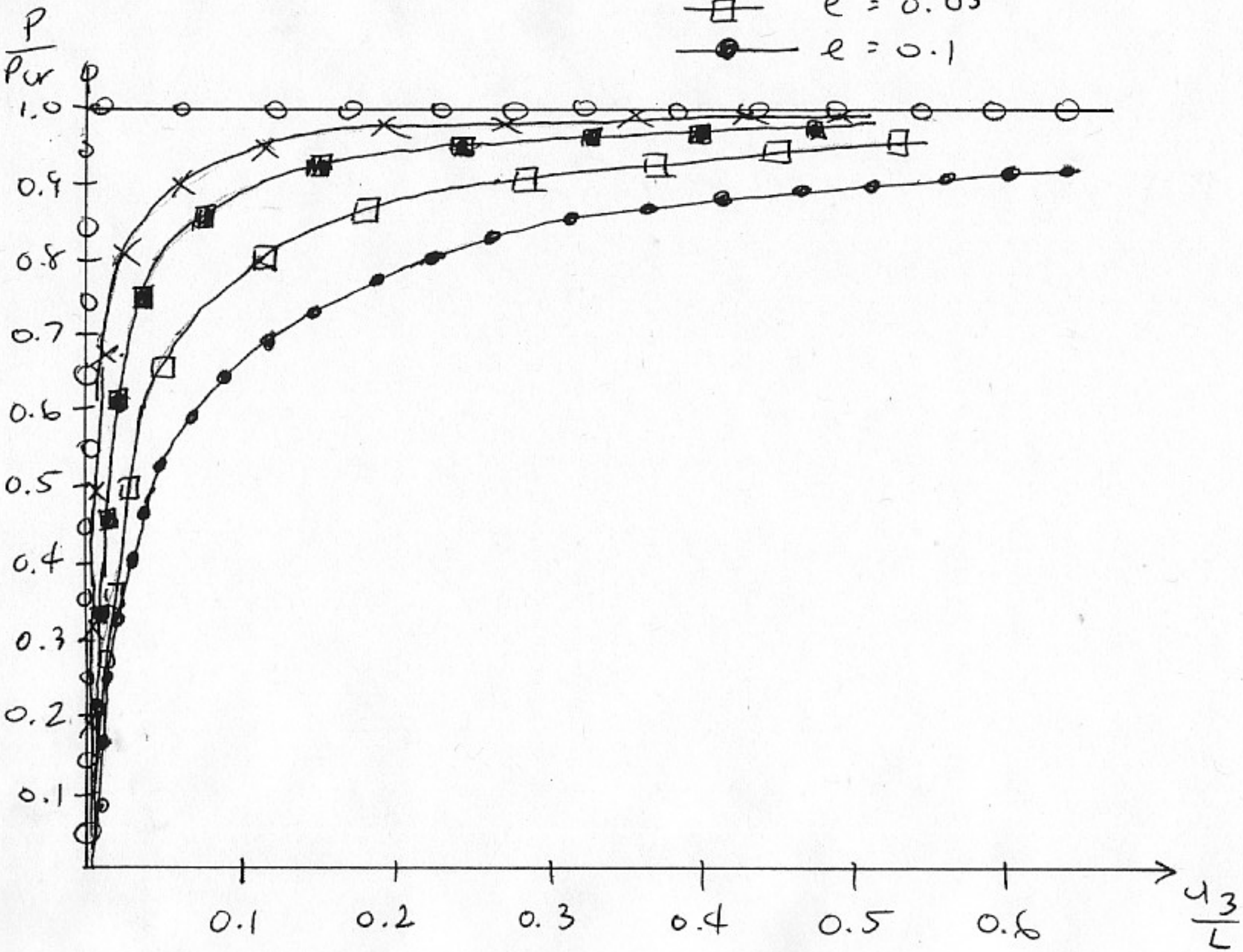
and at the center $\left(\frac{x_1}{L} = 0.5\right)$:

$$\boxed{\frac{u_3}{L} = \frac{e}{L} \left[\frac{1 - \cos \pi\sqrt{\frac{P}{P_{cr}}}}{\sin \pi\sqrt{\frac{P}{P_{cr}}}} \sin\left(\frac{\pi}{2}\sqrt{\frac{P}{P_{cr}}}\right) + \cos\left(\frac{\pi}{2}\sqrt{\frac{P}{P_{cr}}}\right) - 1 \right]}$$

This is the same expression for <u>all</u> cases (just use specific value of e)

(c) use this relationship to make plots for the five cases of $\frac{e}{L} = 0,\ 0.01,\ 0.02,\ 0.05,\ 0.1$

Normalized load vs. Normalized Deflection



Problem C11. Heap Sort

a. Modify heap_sort_pset.ads:

```
-- the binary tree package specifies the data structures and
-- subprograms needed to create a binary tree, and the necessary
-- heap_array
with Binary_Tree;
use Binary_Tree;

package Heap_Sort_Pset is

  -- set the maximum array size to be 10
   Max_Array_Size : constant Integer := 10;

   --YOU NEED TO MODIFY THIS FOR the problem set

   --REPLACED: Heap_Array -> Node_Pointer_Array, Integer -> Nodeptr
   type Node_Pointer_Array is array (1 .. Max_Array_Size) of Nodeptr;

   --the procedure to create a heap array
   --REPLACED: Heap_Array -> Node_Pointer_Array,
   procedure Create_Heap_Array (Root : in Nodeptr; Output_Heap_Array : out Node_Pointer_Array; Size : Out Integer);

   --procedure to heapify the array
   --REPLACED: Heap_Array -> Node_Pointer_Array,
   procedure Heapify( My_Array : in out Node_Pointer_Array; Parent : in integer; Size : in integer);

   -- procedure to transform an array inorder to satisfy the heap property
   --REPLACED: Heap_Array -> Node_Pointer_Array,
   procedure Build_Heap ( My_Array : in out Node_Pointer_Array; Size : in Integer);

   -- procedure to heap_sort the array
   --REPLACED: Heap_Array -> Node_Pointer_Array,
   procedure Heap_Sort(My_Array : in out Node_Pointer_Array; Size : in Integer);

   --this procedure creates the links across the nodes of the tree after it has been
   -- sorted.
   -- THIS SUBPROGRAM HAS TO BE IMPLEMENTED IN THE PSET
   --REPLACED: Heap_Array -> Node_Pointer_Array,
   procedure Create_Links(My_Array : in out Node_Pointer_Array; Size : in integer);

end Heap_Sort_Pset;
```

b. Modify heap_sort_pset.adb:

```
with Binary_Tree;
use Binary_Tree;
with Ada.Text_Io;

package body Heap_Sort_Pset is

   --the procedure to create a heap_array through the BFS of
   -- of the tree.
   procedure Create_Heap_Array (
         Root              : in     Nodeptr;
         Output_Heap_Array :    out Node_Pointer_Array;       -- REPLACED: Heap_Array -> Node_Pointer_Array
         Size :    out Integer ) is
      Index        : Integer;
      Temp_Array   : Node_Pointer_Array;                      -- REPLACED: Heap_Array -> Node_Pointer_Array
      My_Bfs_Queue : My_Queue;
      Temp         : Nodeptr;
   begin
      -- initialize the queue
      Init_Queue(My_Bfs_Queue);
      Index := 1;

      --queue in the root node
      Enqueue(My_Bfs_Queue, Root);

      -- loop until there are no nodes in the queue
      loop
         exit when Empty_Queue(My_Bfs_Queue);

         --get the first node from the queue
         Dequeue(My_Bfs_Queue, Temp);

         --THIS LINE NEEDS TO BE CHANGED
         -- store the element in temp_array
         Temp_Array(Index) := Temp;                           -- DELETED: .Element

         Index := Index +1;
```

```
      --if the left child is not null, enqueue it
      if Temp.Left_Child /= null then
         Enqueue(My_Bfs_Queue, Temp.Left_Child);
      end if;
      --if the right child is not null, enqueue it
      if Temp.Right_Child /= null then
         Enqueue(My_Bfs_Queue, Temp.Right_Child);
      end if;

   end loop;
   --store the actual number of nodes in the tree into size
   Size := Index -1;
   -- store the temp_array into my_array
   Output_Heap_Array := Temp_Array;
end Create_Heap_Array;


--procedure to get the array to satisfy the heap property
procedure Heapify (
      My_Array : in out Node_Pointer_Array;               -- REPLACED: Heap_Array -> Node_Pointer_Array
      Parent : in     Integer;
      Size   : in     Integer  ) is
   Lchild  : Integer;
   Rchild  : Integer;
   Largest : Integer;
   --THIS DECLARATION NEEDS TO BE CHANGED
   Temp : Nodeptr;                                        -- REPLACED: Integer -> Nodeptr

begin
   Lchild := Parent*2;
   Rchild := Lchild+1;

   Largest := Parent;
   if (Lchild <= Size) then
      --THIS LINE OF CODE NEEDS TO BE MODIFIED
      if (My_Array(Lchild).Element > My_Array(Parent).Element) then       -- ADDED: .Element
         Largest := Lchild;
      end if;
   end if;
```

```
   if (Rchild <= Size) then
      --THIS LINE OF CODE NEEDS TO BE MODIFIED
      if (My_Array(Rchild).Element > My_Array(Largest).Element) then      -- ADDED: .Element
         Largest := Rchild;
      end if;
   end if;

   if Largest /= Parent then
      Temp := My_Array(Largest);
      My_Array(Largest):= My_Array(Parent);
      My_Array(Parent):= Temp;
      Heapify(My_Array, Largest, Size);
   end if;
end Heapify;

procedure Build_Heap (
      My_Array : in out Node_Pointer_Array;               -- REPLACED: Heap_Array -> Node_Pointer_Array
      Size     : in     Integer             ) is
begin

   for I in reverse 1 .. Size/2 loop
      Heapify(My_Array, I, Size);
   end loop;
   Ada.Text_Io.New_Line;
end Build_Heap;

--heap_sort the array
procedure Heap_Sort (
      My_Array : in out Node_Pointer_Array;               -- REPLACED: Heap_Array -> Node_Pointer_Array
      Size : in     Integer ) is
   --THIS DECLARATION NEEDS TO BE MODIFIED
   Temp          : Nodeptr;                               -- REPLACED: Integer -> Nodeptr
   Reducing_Size : Integer;
begin
   Reducing_Size := Size;
   Build_Heap(My_Array, Reducing_Size);
   for I in reverse 2 .. Size loop
      Temp := My_Array(I);
      My_Array(I) := My_Array(1);
      My_Array(1) := Temp;
```

```
         Reducing_Size := Reducing_Size -1;
         Heapify(My_Array, 1, Reducing_Size);
      end loop;
   end Heap_Sort;

   --this procedure creates the links across the nodes of the tree after it has been
   -- sorted.
   -- THIS SUBPROGRAM HAS TO BE IMPLEMENTED IN THE PSET
   procedure Create_Links (
         My_Array : in out Node_Pointer_Array;              -- REPLACED: Heap_Array -> Node_Pointer_Array
         Size     : in     Integer             ) is
   begin
                                                           -- ADDED: A BUNCH OF STUFF:
      -- loop through the array, changing the links in the tree by using the rule:
      -- Left_Child : 2*I
      -- Right_Child (2*I)+1
      for I in 1..Size loop
         -- Link the Left Child if one exists
         if I*2 <= Size then
            -- if the index for our child is in the Array, link the parent to its left child
            My_Array(I).Left_Child := My_Array(I*2);
         else
            -- otherwise, this parent doesn't have a left child
            My_Array(I).Left_Child := null;
         end if;
         -- Link the Right Child if one exists
         if (I*2)+1 <= Size then
            -- if the index for our child is in the Array, link the parent to its right child
            My_Array(I).Right_Child := My_Array((I*2)+1);
         else
            -- otherwise, this parent doesn't have a right child
            My_Array(I).Right_Child := null;
         end if;
      end loop;
   end Create_Links;

end Heap_Sort_Pset;
```

Problem C12. Robust Programming

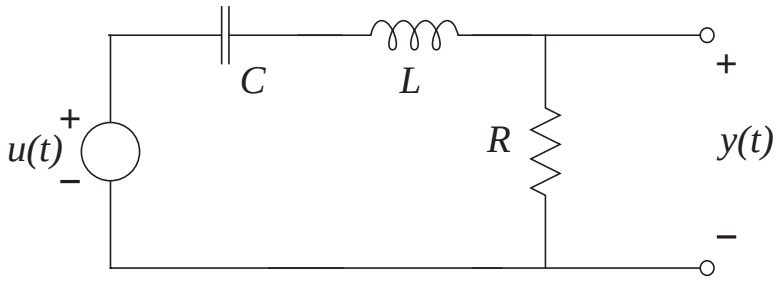
```
with Ada.Text_Io;
with Ada.Integer_Text_Io;

package body Robust_Array_Package is


  procedure Create_Array (
       Robust_Input_Array : in out Robust_Array ) is
  begin
    Ada.Text_Io.Put("Please Enter the size of the array");
    Ada.Integer_Text_Io.Get(Robust_Input_Array.Size);
    Ada.Text_Io.New_Line;
    Ada.Text_Io.Skip_Line;

    for I in 1  .. Robust_Input_Array.Size loop
      Ada.Text_Io.Put("Please Enter the element");
      Ada.Integer_Text_Io.Get(Robust_Input_Array.User_Array(I));
      Ada.Text_Io.Skip_Line;
      Ada.Text_Io.New_Line;
    end loop;
  end Create_Array;

  procedure Get_Element (
       Input_Array : in     Robust_Array;
       Index       : in     Integer;
       Element     :    out Integer       ) is
  begin
    --check for overflow and underflow, and raise appropriate exceptions
    --NEED TO IMPLEMENT THE RAISING OF EXCEPTIONS HERE

    -- There is nothing particularly special in checking for an exception.
    -- You can use an if-else statement like usual, the only difference is that you 'raise'
    -- a error of type 'exception'

    if Index <= 0 then
      raise Underflow;
    elsif Index > Input_Array.Size then
```

```
      raise Overflow;
    end if;

    Element := Input_Array.User_Array(Index);

  exception
    when Constraint_Error =>
      Ada.Text_Io.Put_Line("Raised a constraint error");
      raise;

      --NEED TO IMPLEMENT THE EXCEPTION HANDLERS HERE

      -- After the keyword 'exception,' you insert a bunch of when statements to take care of exceptions
      -- your procedure could raise. There are some exceptions built into Ada, such as "Constraint_Error."
      -- Although the words 'raise Constraint_Error' never occur in the procedure above, the Array type
      -- has this exception built right in.
      --
      -- Below, we implement our exception 'handlers.' A handler consists of the code following the word
      -- "when" and before the next word "when."
      --
      -- Imagine that your computer is a person executing commands
      -- line by line when all of a sudden something bad happens. This computer person 'raises' a sign
      -- indicating what has happened. Another person or set of people, the exception handler[s],
      -- look at the sign to tell which handler is the right one for the job. Once identified, the handler
      -- tells the computer person what steps it should take to solve the problem.
      -- That's it!
      --
      -- The following exception handlers use the word "raise" at the end. This tells the program to
      -- crash and exit. If you removed the words "raise" below, the program would simple SKIP the rest
      -- of the current procedure and continue chugging through the remaining code.

    when Underflow =>
      Ada.Text_Io.Put_Line("Underflow Error: Attempted to reference index" & Integer'Image(Index));
      raise;
    when Overflow=>
      Ada.Text_Io.Put_Line("Overflow Error: Attempted to reference index" & Integer'Image(Index));
      raise;
  end;
end Robust_Array_Package;
```

Unified Engineering II **Spring 2005**

Problem S11 Solution (Signals and Systems)

Find the step response of the circuit below, using Laplace methods. The component values are $C = 0.5$ F, $L = 1$ H, and $R = 3\ \Omega$.



Solution: Treat the components as comples impedances. The impedances are $1/Cs$, Ls, and R. The circuit is a voltage divider, so the transfer function is

$$G(s) = \frac{R}{\frac{1}{Cs} + Ls + R} = \frac{RCs}{LCs^2 + RCs + 1}$$

Plugging in component values,

$$G(s) = \frac{1.5s}{0.5s^2 + 1.5s + 1} = \frac{3s}{s^2 + 3s + 2} = \frac{3s}{(s+1)(s+2)}$$

Since we are tying to find the step response, the input is $u(t) = \sigma(t)$, which implies that

$$U(s) = \frac{1}{s}, \qquad \mathrm{Re}[s] > 0$$

Therefore,

$$Y(s) = G(s)U(s) = \frac{3s}{s(s+1)(s+2)} = \frac{3}{(s+1)(s+2)}$$

The r.o.c. of $Y(s)$ must be $\mathrm{Re}[s] > -1$, because $G(s)$ is causal. The partial fraction expansion is

$$Y(s) = \frac{3}{s+1} - \frac{3}{s+2}$$

The inverse LT is then

$$y(t) = g_s(t) = \sigma(t)\left(3e^{-t} - 3e^{-2t}\right)$$

Unified Engineering II **Spring 2005**

Problem S12 (Signals and Systems)

1. From the problem statement,

$$\omega_n = \sqrt{2}\ \frac{9.82\ \text{m/s}^2}{129\ \text{m/s}} = 0.1077\ \text{r/s}$$
$$\zeta = \frac{1}{\sqrt{2}(L_0/D_0} = \frac{1}{\sqrt{2}\cdot 15} = 0.0471$$

Therefore,

$$\bar{G}(s) = \frac{1}{s\left(s^2 + 0.01015s + 0.0116\right)}$$

The roots of the denominator are at $s = 0$, and

$$s = \frac{-0.01915 \pm \sqrt{0.01015^2 - 4\cdot 0.0116}}{2}$$
$$= -0.005075 \pm 0.1075j$$

So

$$\bar{G}(s) = \frac{1}{s\left(s - [-0.005075 + 0.1075j]\right)\left(s - [-0.005075 - 0.1075j]\right)}$$

Use the coverup method to obtain the partial fraction expansion

$$\bar{G}(s) = \frac{86.283}{s} + \frac{-43.142 + 2.036j}{s - [-0.005075 + 0.1075j]} + \frac{-43.142 - 2.036j}{s - [-0.005075 - 0.1075j]}$$

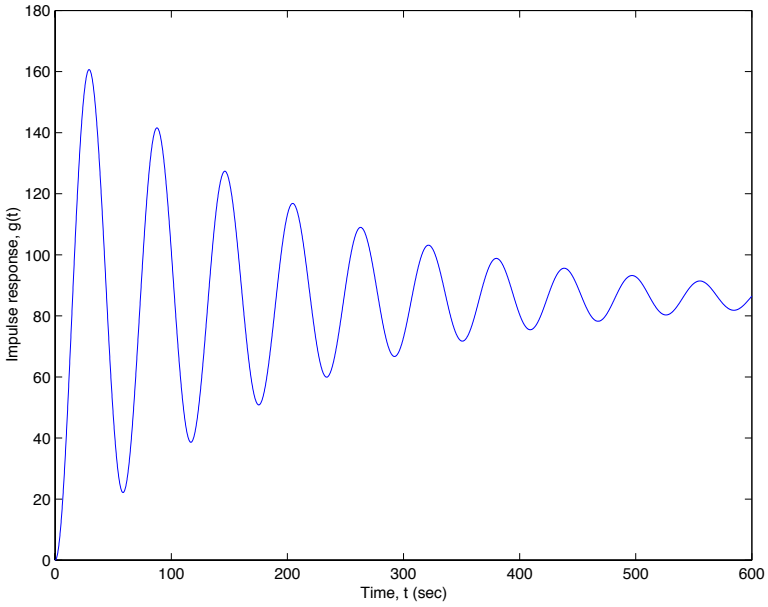
Taking the inverse Laplace transform (assuming that $\bar{g}(t)$ is causal), we have

$$\begin{aligned}\bar{g}(t) =& 86.283\sigma(t) \\ &+ (-43.142 + 2.036j)e^{(-0.005075+0.1075j)t} \\ &+ (-43.142 - 2.036j)e^{(-0.005075-0.1075j)t}\end{aligned}$$

Therefore,

$$\begin{aligned}\bar{g}(t) &= \sigma(t)\left[86.283 + 2e^{-0.005075t}\left(-43.142\cos\omega_d t - 2.036\sin\omega_d t\right)\right] \\ &= \sigma(t)\left[86.283 + \left(-86.284\cos\omega_d t - 4.072\sin\omega_d t\right)e^{-0.005075t}\right]\end{aligned}$$

where $\omega_d = 0.1075$ r/s. See below for the impulse response.



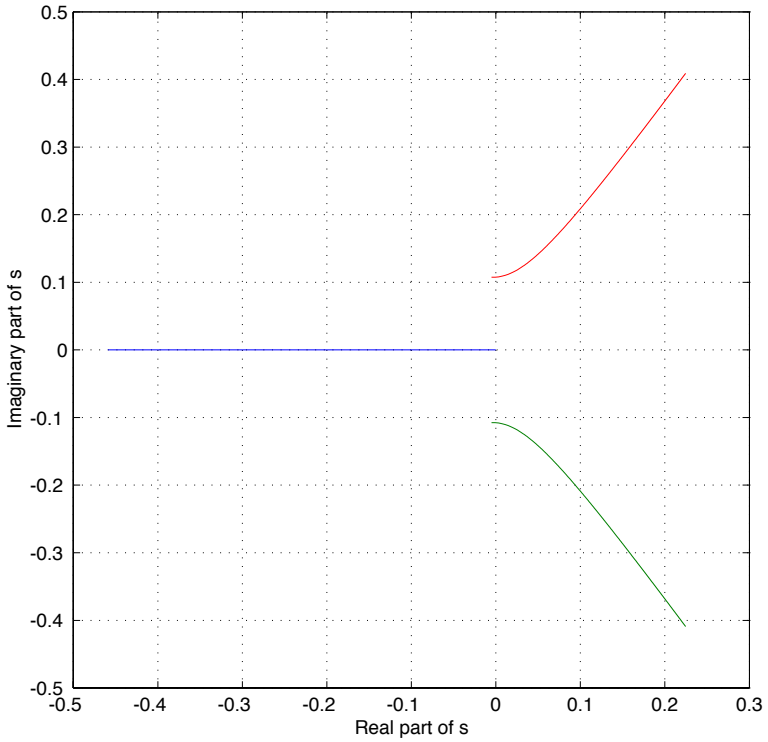
2. From the problem statement,

$$\begin{aligned}\frac{H(s)}{R(s)} &= \frac{k\bar{G}(s)}{1+k\bar{G}(s)} \\ &= \frac{k\,\dfrac{1}{s(s^2+2\zeta\omega_n s+\omega_n^2)}}{1+k\,\dfrac{1}{s(s^2+2\zeta\omega_n s+\omega_n^2)}} \\ &= \frac{k}{s^3+2\zeta\omega_n s^2+\omega_n^2 s+k}\end{aligned}$$

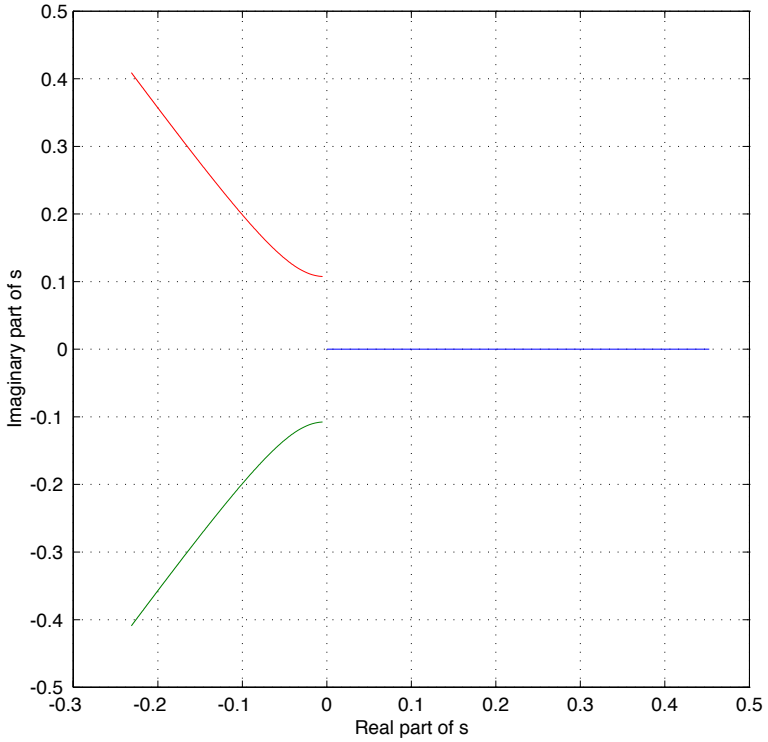
So the poles of the system are the roots of the denominator polynomial,

$$\phi(s) = s^3+2\zeta\omega_n s^2+\omega_n^2 s+k = 0$$

The roots can be found using Matlab, a programmable calculator, etc. The plot of the roots (the "root locus") is shown below. Note that the oscillatory poles go unstable at a gain of only $k = 0.000118$.



3. The roots locus for negative gains can be plotted in a similar way, as below. Note that the real pole is unstable for all negative k.



Problem S13 (Signals and Systems)

1. $g(t) = \cos(at)\sigma(-t)$. To do this problem, expand the sinusoid as complex exponentials, so that

$$g(t) = \left[\frac{e^{ajt} + e^{-ajt}}{2}\right]\sigma(-t)$$

Therefore, the LT is given by

$$G(s) = \int_{-\infty}^{0} \left[\frac{e^{ajt} + e^{-ajt}}{2}\right] e^{-st}\, dt$$

For the LT to converge, the integrand must go to zero as t goes to $-\infty$. Therefore, the integral converges only for $\mathrm{Re}[s] < 0$. The integral is then

$$\begin{aligned} G(s) &= \int_{-\infty}^{0} \left[\frac{e^{ajt} + e^{-ajt}}{2}\right] e^{-st}\, dt \\ &= \frac{1}{2}\left[\frac{1}{-s+aj} e^{(aj-s)t}\Big|_{-\infty}^{0} + \frac{1}{-s-aj} e^{(-aj-s)t}\Big|_{-\infty}^{0}\right] \\ &= \frac{1}{2}\left[\frac{1}{-s+aj} + \frac{1}{-s-aj}\right] \\ &= \frac{-s}{s^2+a^2}, \qquad \mathrm{Re}[s] < 0 \end{aligned}$$

2.

$$g(t) = te^{at}\sigma(-t)$$

The LT is given by

$$G(s) = \int_{-\infty}^{0} te^{at}e^{-st}\, dt = \int_{-\infty}^{0} te^{(a-s)t}\, dt$$

For the LT to converge, the integrand must go to zero as t goes to $-\infty$. Therefore, the integral converges only for $\mathrm{Re}[s] < a$. To find the integral, integrate by parts:

$$\begin{aligned} G(s) &= \int_{-\infty}^{0} te^{(a-s)t}\, dt \\ &= t\frac{1}{a-s} e^{(a-s)t}\Big|_{-\infty}^{0} - \frac{1}{a-s}\int_{-\infty}^{0} e^{(a-s)t}\, dt \\ &= 0 - \frac{1}{a-s}\int_{-\infty}^{0} e^{(a-s)t}\, dt \\ &= -\frac{1}{(a-s)^2} e^{(a-s)t}\Big|_{-\infty}^{0} \\ &= -\frac{1}{(s-a)^2}, \qquad \mathrm{Re}[s] < a \end{aligned}$$

3.

$$g(t) = \sin(\omega_0 t)\, e^{-a|t|}, \qquad \text{for all } t$$

The LT is given by

$$G(s) = \int_{-\infty}^{\infty} \sin(\omega_0 t)\, e^{-a|t|} e^{-st}\, dt$$

For the LT to converge, the integrand must go to zero as t goes to $-\infty$ and ∞. Therefore, the integral converges only for $-a < \mathrm{Re}[s] < a$. The integral is given by

$$\begin{aligned} G(s) &= \int_{-\infty}^{\infty} \sin(\omega_0 t)\, e^{-a|t|} e^{-st}\, dt \\ &= \int_{-\infty}^{0} \sin(\omega_0 t)\, e^{at} e^{-st}\, dt \quad + \quad \int_{0}^{\infty} \sin(\omega_0 t)\, e^{-at} e^{-st}\, dt \end{aligned}$$

Expanding the sine term as

$$\sin(\omega_0 t) = \frac{e^{j\omega_0 t} - e^{-j\omega_0 t}}{2j}$$

yields

$$\begin{aligned} G(s) &= \int_{-\infty}^{0} \frac{e^{j\omega_0 t} - e^{-j\omega_0 t}}{2j}\, e^{at} e^{-st}\, dt \quad + \quad \int_{0}^{\infty} \frac{e^{j\omega_0 t} - e^{-j\omega_0 t}}{2j}\, e^{-at} e^{-st}\, dt \\ &= \int_{-\infty}^{0} \frac{e^{(j\omega_0 + a - s)t} - e^{(-j\omega_0 + a - s)t}}{2j}\, dt \quad + \quad \int_{0}^{\infty} \frac{e^{(j\omega_0 - a - s)t} - e^{(-j\omega_0 - a - s)t}}{2j}\, dt \\ &= \frac{1}{2j}\left[\frac{1}{j\omega_0 + a - s} - \frac{1}{-j\omega_0 + a - s} - \frac{1}{j\omega_0 - a - s} + \frac{1}{-j\omega_0 - a - s}\right] \\ &= \frac{-\omega_0}{s^2 - 2as + a^2 + \omega_0^2} + \frac{\omega_0}{s^2 + 2as + a^2 + \omega_0^2}, \qquad -a < \mathrm{Re}[s] < a \end{aligned}$$