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

Unified Engineering
Spring 2005
Problem Set #7
Solutions

Problem C1. Understanding .ali files

With the following files:

- ❖ `linked_list.ads`
- ❖ `linked_list.adb`
- ❖ `list_test.adb`
- ❖ `linked_list-addtofront.adb`

a. Compile `linked_list-addtofront.adb.`

i. What was the message from the compiler regarding generating code for the file?

Compiling…   No code generated for file linked_list-addtofront.adb ()subunit   Completed successfully.

ii. Turn in a hard copy of the header of `linked_list-addtofront.ali`

```
V "GNAT Lib v3.15"
A -gnatwu
A -g
A -gnato
P NO
R nnnvnnnvnnnnnnnnvvnnnnvnnnnnnnnnnvnnnnnnnnnvnnnnvvnn

U linked_list.addtofront%b  linked_list-addtofront.adb  d0d12a47
W ada%s                     ada.ads                     ada.ali
W ada.unchecked_deallocation%s
W linked_list%s             linked_list.adb             linked_list.ali
```

b. Compile `list_test.adb`

i. Turn in a hard copy of the header of `list_test.ali`

```
V "GNAT Lib v3.15"
M P W=b
A -gnatwu
A -g
A -gnato
P
R nnvnnnnvnnnnnvvvvvvnnvnnnnnnnnnnnvnnnnnnnnnnnnnnvvnn

U list_test%b           list_test.adb           1a771bb9 NE SU
W ada%s                 ada.ads                 ada.ali
W ada.integer_text_io%s a-inteio.ads            a-inteio.ali
W ada.text_io%s         a-textio.adb            a-textio.ali
W linked_list%s         linked_list.adb         linked_list.ali
```

ii. Turn in a hard copy of the header of `linked_list.ali`

```
V "GNAT Lib v3.15"
A -gnatwu
A -gnato
A -g
P
R nnvvnnnvnnnnnvvvvvvnnvvnnnnnnnnnnvnnnnnnnnnvnnnnvvnn

U linked_list%b         linked_list.adb         48c5876e NE PK
W ada%s                 ada.ads                 ada.ali
W ada.integer_text_io%s a-inteio.ads            a-inteio.ali
W ada.text_io%s         a-textio.adb            a-textio.ali
W ada.unchecked_deallocation%s
```

Turn in a hardcopy of your answer.

```
-- PROBLEM C2
-----------------------------------------------
-- Package implementation for recursive binary search
-- search algorithm
-- Specifier: Jane B.
-- Date Last Modified: March 27, 2005
--
-- BINARY SEARCH FUNCTION
-- Pre-Conditions: An integer array with known length (lower_bound and upper_bound) and Integer Element to search for
--
-- Post-Conditions: Index of Element in array if found, else -1
--
-- Assumptions   :  Array is indexed starting from a number greater than -1
--
-- Pseudo-Code:
--   1. Calculate the mid-index of the array
--
--   2. If the lower_bound is greater than the upper_bound, return -1
--
--   3. If the element is at the mid-index location, return index.
--
--   4. If the element is less than the value at the current index location, repeat step 1 with lower half of array
--
--   5. If the element is greater than the value at the current index location, repeat step 1 with lower upper of array
--
------------------------------------------
with Ada.Text_Io;
with Ada.Integer_Text_Io;

package body PSET1_2 is

   procedure Initialize (My_Search_Array : in out My_Array) is
   begin
      for I in 1 .. Max_Array_Size loop
         Ada.Text_Io.Put("Please enter an integer : ");
         Ada.Integer_Text_Io.Get(My_Search_Array(I));
         Ada.Text_Io.New_Line;
         Ada.Text_Io.Skip_Line;
      end loop;
   end Initialize;


   -- THIS FUNCTION HAS BEEN MODIFIED TO REPRESENT THE ANSWER TO PROBLEM C2
   -- The original values for each line can be found following the comment "DELETED" or "MODIFIED"
   -- Your Answer may deviate from this one, but the answer should NOT involve a "loop"
   function Binary_Search (My_Search_Array : My_Array; Lb : Integer; Ub: Integer; Element : Integer) return Integer is
                                                                                             -- DELETED   Index : Integer := - 1;

      Lower_Bound   : Integer := Lb;
      Upper_Bound   : Integer := Ub;
      Current_Index : Integer;

   begin
                                                                                             -- DELETED   loop
         -- find midpoint
         Current_Index := (Upper_Bound + Lower_Bound)/2;
```

```
      -- condition for exiting the loop
      if (Lower_Bound > Upper_Bound) then
         return -1;                                                                -- MODIFIED  exit;
      end if;
      -- check if the element is found
      if (My_Search_Array(Current_Index) = Element) then
         return Current_Index;                                                     -- MODIFIED  Index := Current_Index;
                                                                                   -- DELETED   exit;
      end if;
      -- determine which portion of the array to search in
      if (My_Search_Array(Current_Index) < Element) then                           -- REPLACED > with <
         -- narrow the searchspace to the lower half of the array
         return Binary_Search(My_Search_Array, Lower_Bound, Current_Index-1, Element);  -- MODIFIED  Upper_Bound := Current_Index -1;
      else
         -- search in the upper half of the array.
         return Binary_Search(My_Search_Array, Current_Index+1, Upper_Bound, Element);  -- MODIFIED  Lower_Bound := Current_Index +1;
      end if;
                                                                                   -- DELETED   end loop;
                                                                                   -- DELETED   return Index;
   end Binary_Search;

--   Now, let's look at this RECURSIVE function in DETAIL and see what is going on. . .

--   function Binary_Search (My_Search_Array : My_Array; Lb : Integer; Ub: Integer; Element : Integer) return Integer is
--                       ## The 'index' variable was originally used to store the value -1. If the 'if' statements
--                       ## didn't find a matching 'element' in the array, the 'Index' value would not get changed and the loop
--                       ## would exit, returning an Index of -1 (a.k.a, "Found Nothing")
--
--                       ## Note that variables 'Lower_Bound' and 'Upper_Bound' are not necessary. They are used in both
--                       ## the iterative case (original function that uses 'loop') and in the recursive answer simply to make
--                       ## the variables 'Lb' and 'Ub' more understandable.
--      Lower_Bound   : Integer := Lb;
--      Upper_Bound   : Integer := Ub;
--
--                       ## This variable is very important. It stores the newly calculated array index to be checked for the 'element'.

--      Current_Index : Integer;
--   begin
--                       ## find midpoint, a.k.a divide the array in two halves, guessing the answer to be at the half-way point.
--         Current_Index := (Upper_Bound + Lower_Bound)/2;
--
--                       ## BASE CASE! If you try dividing the array until your Lower search limit is greater than your Upper
--                       ## and still haven't found anything, you should quit and return the answer that corresponds to finding
--                       ## nothing "-1"
--         if (Lower_Bound > Upper_Bound) then
--            return -1;
--         end if;
--
--                       ## CHECK FOR ANSWER! Now that we know we aren't doing anything illogical, like checking an array where the
--                       ## lower bound is greater than the upper bound (Base Case), let's see if we have divided the array at the
--                       ## correct midpoint to find the 'Element.' If so, return the index of the 'Element.'
--         if (My_Search_Array(Current_Index) = Element) then
--            return Current_Index;
--         end if;
--
--                       ## RECURSIVE CASE! Well... the array is still divided in a logical manner AND we didn't bisect the array
```

```
--                       ## at an index at which we would find "Element," so we need to decide which half of the array to continue
--                       ## searching. Fortunately, the array is ordered from least to greatest. If the "Element" has a lesser value
--                       ## than the Index we have guessed at, search the lower half of the REMAINING array. Otherwise, visa-versa.
--                       ## Look! We are just searching another, smaller array, don't we have a function that does that? Sure do, it's
--                       ## called 'Binary_Search', so let's just call that function again.
--
--                       ## Note that the greater than sign was replaced with the less than sign. . .this is the only change needed
--                       ## to search an ascending vs descending list.
--
--         if (My_Search_Array(Current_Index) < Element) then
--            -- narrow the searchspace to the lower half of the array
--            return Binary_Search(My_Search_Array, Lower_Bound, Current_Index-1, Element);
--         else
--            -- search in the upper half of the array.
--            return Binary_Search(My_Search_Array, Current_Index+1, Upper_Bound, Element);
--         end if;
--
--   end Binary_Search;


end PSET1_2;
```

```
-- PROBLEM C3
-----------------------------------------------------------------------
-- Program to evaluate an expression in Postfix form
-- Pre-conditions: An arithmetic expression in legal postfix form
--
-- Post-Conditions: Evaluation of the arithmetic Expression
--
-- Assumptions   :  The expression is represented using a string
--                  operands are single charcter digits
--                  operators are the binary operators *,/,+,-
--                  performing integer operations
-- Pseudo-Code:
--   1. Create a stack for holding operands
--
--   2. Start from left to right (Set index to leftmost element)
--
--   3. Repeat the following steps till index is 1
--
--      i.   Get character at Postfix(Index)
--
--      ii.  If it is an operand, push it onto the stack
--
--      iii. If it is an operator
--            a. If There are < 2 elements in the stack, return error
--            b. Pop operand Y from the stack
--            c. Pop operand X from the stack
--            d. Evaluate X and Y operator and push the result onto the stack (be aware of order of X and Y)
--
--      iv.  Decrement index by 1
--
--   4. There should only be one element in the stack
--
--   5. If there is more than one element on the stack, return error
--
--   6. Return the value of the expression
--
--    Programmer : Joe B
--    Date Last Modified : March 27, 2005
-------------------------------------------------------------------------------------

with Ada.Text_Io;
with Ada.Integer_Text_Io;
with Ada.Characters.Handling;
with Ada.Strings;

use Ada.Text_Io;
use Ada.Integer_Text_Io;
use Ada.Characters.Handling;
use Ada.Strings;

with My_Expression_Evaluator;


-- THIS FUNCTION HAS BEEN MODIFIED TO REPRESENT THE ANSWER TO PROBLEM C3
-- The original values for each line can be found following the comment "DELETED" or "REPLACED"
-- REPLACED the word "Prefix" with the word "Postfix" just to make things look pretty.
```

```
-- Look at the modifications below, besides replacing the word "Prefix" with "Postfix," there are only
-- 3 real changes (2 if you pick your changes carefully). Prefix and Postfix notation are remarkably similar!
--
-- 1. We need to un-reverse the direction of the original loop by removing the word "reverse"
-- 2. We need to reverse the order in which we are evaluating the two operands since we have reversed
--    the order with which we are looping through the expression.
-- THAT'S IT!

procedure Postfix_Evaluator is
   Operand_Stack      : My_Expression_Evaluator.My_Stack;
   Postfix_Expression : String (1 .. 25);                          -- define a string of length 25
   Length             : Natural;
   X,
   Y                  : Integer;

begin
   My_Expression_Evaluator.Create(Operand_Stack);

   -- loop to get the Postfix expression as a string
   loop
      Put_Line("Please Enter the expression in postfix form");
      Get_Line(Postfix_Expression, Length);
      if ((Length <=0) or (Length > My_Expression_Evaluator.Stack_Size)) then
         Put_Line("Invalid Expression : Please enter another expression");
      else
         exit;
      end if;
   end loop;

   -- evaluate the expression from right to left
   for I in 1 .. Length loop                                                  -- DELETED reverse
      if Is_Digit(Postfix_Expression(I)) then
         My_Expression_Evaluator.Push(Operand_Stack, (Character'Pos(Postfix_Expression(I))-Character'Pos('0')));

      elsif My_Expression_Evaluator.Isoperator(Postfix_Expression(I)) then
         -- check if there are less than two operands on the stack
         if ((My_Expression_Evaluator.Stacklength(Operand_Stack)) < 2) then
            Put_Line("Illegal Postfix Expression");
            exit;
         else
            -- pop the operand
            My_Expression_Evaluator.Pop(Operand_Stack, Y);                    -- REPLACED X with Y
            -- pop the other operand
            My_Expression_Evaluator.Pop(Operand_Stack, X);                    -- REPLACED Y with X
            -- evaluate the expression and push the result onto the operand stack
            My_Expression_Evaluator.Evaluate(Operand_Stack, Postfix_Expression(I), X,Y);
         end if;
      else
         Put_Line("Illegal Postfix Expression");
         exit;
      end if;

   end loop;
   -- check if only one value is left on the stack
   if (My_Expression_Evaluator.Stacklength(Operand_Stack)/= 1) then
```

```
      Put_Line("Illegal Postfix Expression");
   else
      Put("The expression evaluates to : ");
      My_Expression_Evaluator.Pop(Operand_Stack, X);
      Put(X);
      New_Line;
   end if;
end Postfix_Evaluator;

---------------------------------------------------------------------------------------------------------
--
-- CAN YOU IMAGINE WHAT"S HAPPENING?
--
-- Let's keep track of:
-- # Prefix_Expression (String)
-- # Stack
-- # X
-- # Y
--
-- This is what happens in the original Prefix notation
-- (Error Checking has been left out of this breakdown for ease of understanding) :
--                                           | Prefix_Expression | Stack | X | Y |
--                                           |                   |       |   |   |
-- 1. START!                                 |       -23         | EMPTY |   |   |
--                                           |                   |       |   |   |
--                                           |         v         |       |   |   |
-- 2. Look at the END of the list and        |       -23         |   3   |   |   |
--    store the last value into the Stack    |                   | EMPTY |   |   |
--                                           |        v          |       |   |   |
-- 3. Loop! Look at the END of the list and  |       -23         |   2   |   |   |
--    store the last value into the Stack    |                   |   3   |   |   |
--                                           |       v           | EMPTY |   |   |
-- 4. Loop! Now we've got an operator.       |       -23         |       |   |   |
--    So call the evaluator                  |                   |       |   |   |
--                                           |                   |       |   |   |
--    a. Pop the stack and store the value   |                   |   3   | 2 |   |
--       to variable X                       |                   | EMPTY |   |   |
--                                           |                   |       |   |   |
--    b. Pop the stack and store the value   |                   | EMPTY | 2 | 3 |
--       to variable Y                       |                   |       |   |   |
--                                           |                   |       |   |   |
--    c. Call the evaluator which does this  |                   |  -1   | 2 | 3 |
--       calculation: (X operator Y) and     |                   | EMPTY |   |   |
--       puts the answer back on the stack   |                   |       |   |   |
--                                           |       v           |  -1   | 2 | 3 |
-- 5. We are at the HEAD of the array,       |       -23         | EMPTY |   |   |
--    so no more looping is necessary        |                   |       |   |   |
--                                           |                   |       |   |   |
-- 6. Since there SHOULD be only one item    |       -23         | EMPTY |-1 | 3 | ANSWER = -1
--    on the stack, return the answer by     |                   |       |   |   |
--    popping the Stack.                     |                   |       |   |   |
--
-- Note: In this implementation, the answer is popped into variable X. . .but it doesn't really matter,
-- the answer could be popped into variable 'blah.'
--
-- Can you figure out, generally, where these steps happen in the original Prefix_Evaluator.adb code?
```

```
--
-------------------------------------------------------------------------------------------------------
--
-- Now. . .Let's take a look at Postfix_Evaluator and see what those few changes did:
--                                          | Postfix_Expression| Stack | X | Y |
--                                          |                   |       |   |   |
-- 1. START!                                |       23-         | EMPTY |   |   |
--                                          |                   |       |   |   |
--                                          |       v           |       |   |   |
-- 2. Look at the HEAD of the list and      |       23-         |   2   |   |   |  <-- CHANGE, it loops Forward!
--    store the last value into the Stack   |                   | EMPTY |   |   |
--                                          |        v          |       |   |   |
-- 3. Loop! Look at the END of the list and |       23-         |   3   |   |   |
--    store the last value into the Stack   |                   |   2   |   |   |
--                                          |         v         | EMPTY |   |   |
-- 4. Loop! Now we've got an operator.      |       23-         |       |   |   |
--    So call the evaluator                 |                   |       |   |   |
--                                          |                   |       |   |   |
--    a. Pop the stack and store the value  |                   |   2   |   | 3 |  <-- CHANGE, var Y is stored first!
--       to variable Y                      |                   | EMPTY |   |   |
--                                          |                   |       |   |   |
--    b. Pop the stack and store the value  |                   | EMPTY | 2 | 3 |  <-- CHNAGE, var X is stored second!
--       to variable X                      |                   |       |   |   |
--                                          |                   |       |   |   |
--    c. Call the evaluator which does this |                   |  -1   | 2 | 3 |
--       calculation: (X operator Y) and    |                   | EMPTY |   |   |
--       puts the answer back on the stack  |                   |       |   |   |
--                                          |         v         |  -1   | 2 | 3 |
-- 5. We are at the END of the array,       |       23-         | EMPTY |   |   |
--    so no more looping is necessary       |                   |       |   |   |
--                                          |                   |       |   |   |
-- 6. Since there SHOULD be only one item   |       23-         | EMPTY |-1 | 3 | ANSWER = -1
--    on the stack, return the answer by    |                   |       |   |   |
--    popping the Stack.                    |                   |       |   |   |
--
-- Note: In this implementation, the answer is popped into variable X. . .but it doesn't really matter,
-- the answer could be popped into variable 'blah.'
--
-- Can you figure out, generally, where these steps happen in the original Postfix_Evaluator.adb code?
--
-------------------------------------------------------------------------------------------------------
```

```ada
-- PROBLEM C4
------------------------------------------------------------------------
-- Specification for  doubly linked lists
-- Specified: Joe B
-- Last Modified: March 27, 2005
------------------------------------------------------------------------

package Doubly_Linked_List is

   subtype Elementtype is Integer;

   type Doublelistnode;
   type Doublelistptr is access Doublelistnode;

   -- THIS TYPE HAS BEEN MODIFIED TO REPRESENT THE ANSWER TO PROBLEM C4
   -- Note that only ONE change has been made, "ADDED"
   type Doublelistnode is
      record
         Element : Elementtype;
         Next    : Doublelistptr;
         Prev    : Doublelistptr;      -- ADDED
      end record;

   type Doublelist is
      record
         Head : Doublelistptr;
      end record;

   procedure Makeempty (
         L : in out Doublelist );
   -- Pre:    L is defined
   -- Post:   L is empty

   function Isempty (
         L : in     Doublelist )
     return Boolean;
   -- Pre:    L is defined
   -- Post:   returns True if L is empty, False otherwise

   procedure Display (
         L : in     Doublelist );
   -- Pre:  L may be empty
   -- Post: displays the contents of L's Element fields, in the
   --   order in which they appear in L

   procedure Initialize (
         L : in out Doublelist );

   -- Pre: L may be empty
   -- Post: Elements inserted into the list at correct position
   procedure Insert_In_Reverse_Order (
         L       : in out Doublelist;
         Element : in     Elementtype );

end Doubly_Linked_List;
```

```
-- PROBLEM C4
------------------------------------------------------------------------
-- Implementation for linked_list package
--
-- INSERT_IN_REVERSE_ORDER PROCEDURE
-- Pre-conditions: An initialized doubly linked list and an integer value to insert
--
-- Post-Conditions: a doubly linked list of descending order
--
-- Assumptions:
--
-- Pseudo-Code:
--   1. Create three pointers, one to reference the New Node, the other two to keep track of traversal
--
--   2. Initialize the New Node with the Element Value
--
--   3. If there are no nodes in the list, create the first node
--
--      i. Make the list Head point to New Node
--      ii. Make the Previous and Next pointer null
--
--   4. If the node to be inserted is greater than the first node, insert it.
--
--      i. Make the Head's Previous point to temp
--      ii. Make the Temp's Next point to Head
--      iii. Make Head point to temp
--      iv. Make Temp's Previous point to null
--
--   5. Loop through the list until the appropriate node location is found and insert it.
--
--      i. Set Previous equal to Current
--      ii. Set Current equal to Current.Next, checking for a Current.Next = null
--      iii. Loop through i and ii until Current.Element is greater than the Element to be inserted.
--      iv. Set Temp's Next to Current
--      v. Set Temp's Previous to Previous
--      vi. Set Previous' Next to Temp
--      vii. Set Current's Previous to Temp
--
-- Programmer: Joe B
-- Last Modified: March 27, 2005
------------------------------------------------------------------------

with Ada.Text_Io;
with Ada.Integer_Text_Io;
with Ada.Unchecked_Deallocation;

use Ada.Text_Io;
use Ada.Integer_Text_Io;

package body Doubly_Linked_List is

   -- create an instance of the free procedure
   procedure Free is
   new Ada.Unchecked_Deallocation(Doublelistnode, Doublelistptr);

   -- check if list is empty. List.Head will be null
```

```
function Isempty (
      L : in     Doublelist )
  return Boolean is
begin
   if L.Head = null then
      return True;
   else
      return False;
   end if;

end Isempty;

-- free all allocated memory at the end of the program
procedure Makeempty (
      L : in out Doublelist ) is
   Temp : Doublelistptr;

begin
   loop
      exit when Isempty(L);
      Temp := L.Head;
      L.Head := Temp.Next;
      Free(Temp);
   end loop;
   L.Head := null;
end Makeempty;

-- initialize the list by setting the head pointed to null
procedure Initialize (
      L : in out Doublelist ) is
begin
   L.Head := null;
end Initialize;

-- displays the contents of the list
procedure Display (
      L : in     Doublelist ) is
   Temp : Doublelistptr;
begin
   -- set the pointer to the head of the node
   Temp:= L.Head;
   while Temp /= null loop
      Put(Temp.Element);
      Put(" , ");
      -- move pointer to the next node
      Temp :=Temp.Next;
   end loop;
   New_Line;
end Display;

-- THIS FUNCTION HAS BEEN MODIFIED TO REPRESENT THE ANSWER TO PROBLEM C4
-- Your Answer may deviate from this one, but the answer should be similar.
-- insert elements in descending order
procedure Insert_In_Reverse_Order (
      L       : in out Doublelist;
      Element : in     Elementtype ) is
```

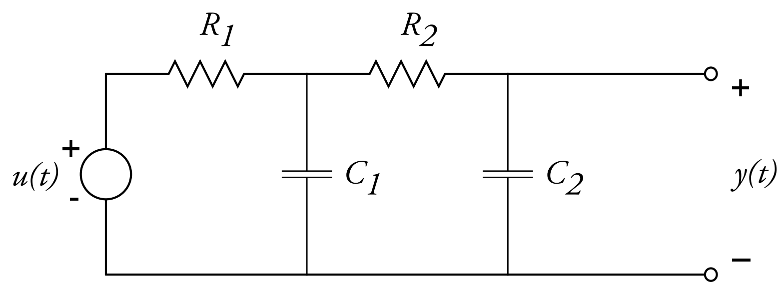
```
      Temp : Doublelistptr;                  -- Create a pointer for a new node
      Current : Doublelistptr;               -- Create pointers to keep track of list traversal
      Previous : Doublelistptr;
   begin
      -- assign Temp the Element value
      Temp := new Doublelistnode;            -- Give the pointer something to point to
      Temp.Element := Element;               -- Store the Element value into the dereferenced pointer
      Current := L.Head;                     -- Give our traversal pointers some initial values
      Previous := null;

      if Isempty(L) then
         -- when there are NO nodes on the list, create the first node
         L.Head := Temp;
         Temp.Next := null;
         Temp.Prev := null;

      elsif Temp.Element > L.Head.Element then
         -- when the node to be inserted should be the first in the list
         L.Head.Prev := Temp;
         Temp.Next := L.Head;
         L.Head := Temp;
         Temp.Prev := null;

      else
         -- when the node to be inserted requires traversal of the list
         while Temp.Element < Current.Element loop
            -- traverse the list until you find the location where temp node should be inserted.
            -- You want to updated Previous and Current, so that they represent the nodes immediately before
            -- and after the location where you want your node inserted.
            Previous := Current;
            Current := Current.Next;
            -- If you reach the end of the list 'Current' will be null. You HAVE TO exit.
            -- Failure to exit will result in the while loop checking for "Current.Element"
            -- Since Current points to nothing, this will result in a pointer dereferencing error!
            if Current = null then exit; end if;
         end loop;

         -- go about assigning pointers. Note that the order of pointer assignment is important!
         -- failure to assign the pointers in a certain order may result in losing track of where in the list you are.
         Temp.Next := Current;
         Temp.Prev := Previous;
         Previous.Next := Temp;
         -- If the node needs to be inserted at the end of the list, Current.Prev will result in a dereferencing error.
         if Current /= null then
           Current.Prev := Temp;
         end if;
      end if;
   end Insert_In_Reverse_Order;


end Doubly_Linked_List;
```

```
-----------------------------------------------------------------------
-- Test procedure for double linked lists (Problem C4)
-- Specified: David Wang
-- Last Modified: March 27, 2005
-----------------------------------------------------------------------

with Ada.Text_Io;
with Ada.Integer_Text_Io;
with Doubly_Linked_List;

use Doubly_Linked_List;

procedure Test_Doubly_Linked_List is

   Test_List : Doublelist;
   type Integer_Array is array (1..15) of Integer;
   -- Create an array of integers to be inserted into the list
   Test_Array : Integer_Array := (5,10,9,22,135,6,4,8,2,1,0,-5,1,-10,672);

begin
   Initialize(Test_List);
   Ada.Text_Io.Put("To Start Default Test, Press [Enter].");
   Ada.Text_Io.Skip_Line;
   Ada.Text_Io.New_Line;
   Ada.Integer_Text_IO.Default_Width := 1;

   -- Cycle through the test array and insert the integer values
   for I in Test_Array'Range loop
      Ada.Text_Io.Put("Insert ");
      Ada.Integer_Text_Io.Put(Test_Array(I));
      Ada.Text_Io.Put(" into the doubly linked list. The result:");
      Ada.Text_Io.New_Line;
      Insert_In_Reverse_Order(Test_List,Test_Array(I));
      Display(Test_List);
      Ada.Text_Io.Skip_Line;
   end loop;

end Test_Doubly_Linked_List;
```

Unified Engineering II **Spring 2005**

Problem S1 Solution

1. Find and plot the step response of the system



where $C_1 = 1$ F, $C = 1/3$ F, $R_1 = 1\ \Omega$, and $R_2 = 4\ \Omega$.

Solution. The solution procedure is the usual procedure for solving linear, constant coefficient differential equations:

(a) Find the differential equation

(b) Find the homogeneous solutions

(c) Find a particular solutions

(d) Express the total solution as the sum of the particular solution and the homogeneous solution

(e) Find the unknown coefficients in the homogenous part to match the initial conditions

The overall solution is long, but not conceptually hard. We can use the node method to find the differential equations. The equations are

$$\begin{array}{rcrcl} \left(C_1\frac{d}{dt} + G_1 + G_2\right) e_1 & - & G_2\, e_2 & = & G_1 u(t) \\ -G_2\, e_1 & + & \left(C_2\frac{d}{dt} + G_2\right) e_2 & = & 0 \end{array} \tag{1}$$

In terms of the component values, we have

$$\begin{array}{rcrcl} \left(\frac{d}{dt} + 1.25\right) e_1 & - & 0.25\, e_2 & = & u(t) \\ -0.25\, e_1 & + & \left(\frac{1}{3}\frac{d}{dt} + 0.25\right) e_2 & = & 0 \end{array} \tag{2}$$

The solution to the differential equations can be found as the sum the the particular and homogenous solutions. Let's start with the homogenous solutions. As always, we assume solutions of the form

$$\begin{bmatrix} e_1(t) \\ e_2(t) \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \end{bmatrix} e^{st} \tag{3}$$

For the homogenous solution, we also set $u(t) = 0$. Then Equation (**??**) becomes

$$\begin{array}{rcrcl} (s+1.25)\,E_1 & - & 0.25\,E_2 & = & 0 \\ -0.25\,E_1 & + & \left(\frac{1}{3}\,s+0.25\right)\,E_2 & = & 0 \end{array} \tag{4}$$

In matrix form,

$$\begin{pmatrix} s+1.25 & -0.25 \\ -0.25 & \frac{1}{3}\,s+0.25 \end{pmatrix} \begin{bmatrix} E_1 \\ E_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{5}$$

For there to be a nontrivial solution, we must have that

$$\det \begin{pmatrix} s+1.25 & -0.25 \\ -0.25 & \frac{1}{3}\,s+0.25 \end{pmatrix} = \frac{1}{3}s^2 + \frac{2}{3}s + \frac{1}{4} = 0 \tag{6}$$

This characteristic equation may be solved by using the quadratic formula or by inspection. The roots are

$$\begin{aligned} s_1 &= -0.5 \\ s_2 &= -1.5 \end{aligned} \tag{7}$$

For each characteristic value, there is a corresponding characteristic vector. We will solve for each of these in turn.

s_1: Substituting s_1 into Equation (3), we obtain

$$\begin{pmatrix} 0.75 & -0.25 \\ -0.25 & 0.0833 \end{pmatrix} \begin{bmatrix} E_1 \\ E_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{8}$$

This equation can be solved symbolically or numerically using elimination of variables. The result is

$$\begin{bmatrix} E_1 \\ E_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \end{bmatrix} \tag{9}$$

(Note that the characteristic vector is not unique — it can be scaled by any number.)

s_2: Substituting s_2 into Equation (3), we obtain

$$\begin{pmatrix} -0.25 & -0.25 \\ -0.25 & -0.25 \end{pmatrix} \begin{bmatrix} E_1 \\ E_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{10}$$

This equation can be solved symbolically or numerically using elimination of variables. The result is

$$\begin{bmatrix} E_1 \\ E_2 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \tag{11}$$

Putting these two results together, we have that the homogenous solution is

$$\underline{E}_h = c_1 \begin{bmatrix} 1 \\ 3 \end{bmatrix} e^{-0.5t} + c_2 \begin{bmatrix} 1 \\ -1 \end{bmatrix} e^{-1.5t} \tag{12}$$

To find the particular solution for $t \geq 0$, we guess. Because $u(t) = \sigma(t)$ is a constant (1) for $t \geq 0$, we guess that $e_1(t)$ and $e_2(t)$ are constants. Then all the d/dt terms become zero; thus, Equation (**??**) becomes

$$\begin{pmatrix} 1.25 & -0.25 \\ -0.25 & 0.25 \end{pmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{13}$$

This equation may be solved by row reduction, Cramer's rule, etc. The solution is

$$e_p(t) = \begin{bmatrix} e_1(t) \\ e_2(t) \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{14}$$

for $t \geq 0$. The total solution is the sum of the homogeneous and particular solutions. The constants c_1 and c_2 are found by requiring that the initial charge on the capacitors is zero, so that $e_1(0) = 0$ and $e_2(0) = 0$. Then

$$\underline{e}(0) = \begin{bmatrix} e_1(0) \\ e_2(0) \end{bmatrix} = c_1 \begin{bmatrix} 1 \\ 3 \end{bmatrix} + c_2 \begin{bmatrix} 1 \\ -1 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{15}$$

In matrix form,

$$\begin{pmatrix} 1 & 1 \\ 3 & -1 \end{pmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \end{bmatrix} \tag{16}$$

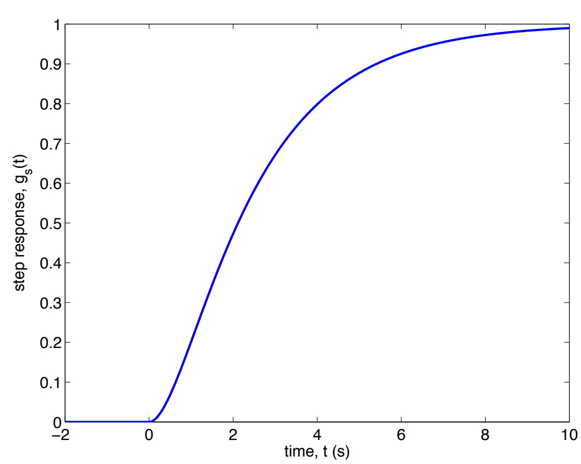
The solution is

$$\begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} -1/2 \\ -1/2 \end{bmatrix} \tag{17}$$

Therefore, the step response is

$$\begin{aligned} g_s(t) &= y(t) && \text{(because we have solved for step input)} \\ &= e_2(t) && \text{(because } y(t) \text{ is measured from } e_2 \text{ to ground.)} \\ &= \sigma(t)\left(1 - \frac{3}{2}\,e^{-0.5t} + \frac{2}{2}\,e^{-1.5t}\right) && \end{aligned} \tag{18}$$

The last line was obtained by multiplying out the constants from the characteristic vectors with c_1 and c_2. Also, we multiplied by the unit step, so the answer is valid for all time. The step response is shown below:



2. For the input signal

$$u(t) = \begin{cases} 0, & t < -1 \\ 1, & -1 \leq t < 1 \\ -2, & t \geq 1 \end{cases} \tag{19}$$

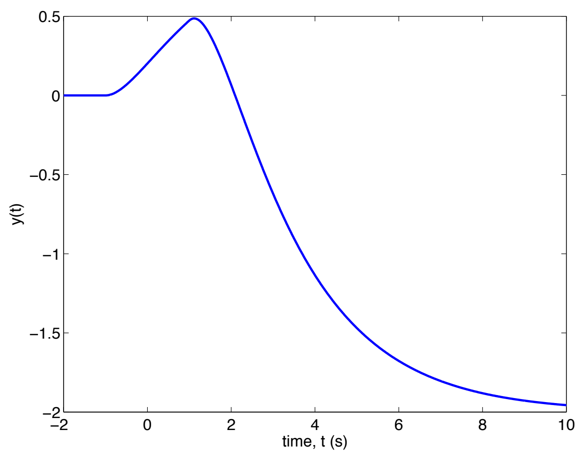
find and plot the output $y(t)$, using superposition. $u(t)$ can be represented simply as

$$u(t) = \sigma(t+1) - 3\sigma(t-1) \tag{20}$$

By linearity and time invariance, the response must to this input must be given by

$$y(t) = g_s(t+1) - 3g_s(t-1) \tag{21}$$

The response to $u(t)$ is shown below:



Problem S2 Solution

A system has step response given by

$$g_s(t) = \begin{cases} 0, & t < 0 \\ e^{-t} - e^{-3t}, & t \geq 0 \end{cases}$$

Find the response of the system to the input

$$u(t) = \begin{cases} 0, & t < 0 \\ 1 - e^{-2t}, & t \geq 0 \end{cases}$$

using Duhamel's integral.

Solution. Duhamel's integral is

$$y(t) = u(0)g_s(t) + \int_0^\infty g_s(t-\tau)\frac{du(\tau)}{d\tau}\, d\tau$$

For $\tau \geq 0$,

$$\frac{du(\tau)}{d\tau} = 2e^{-2\tau}$$

For $\tau < 0$, $u^{'}(\tau) = 0$. Therefore, we can express $u^{'}(\tau)$ as

$$\frac{du(\tau)}{d\tau} = 2e^{-2\tau}\sigma(\tau)$$

Likewise, $g_s(t)$ can be expressed more compactly as

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

Also, note that $u(0) = 0$. Therefore, for $t \geq 0$,

$$\begin{aligned} y(t) &= \int_0^\infty \left(e^{-(t-\tau)} - e^{-3(t-\tau)}\right)\sigma(t-\tau)2e^{-2\tau}\sigma(\tau)\, d\tau \\ &= \int_0^t \left(e^{-(t-\tau)} - e^{-3(t-\tau)}\right)2e^{-2\tau}\, d\tau \\ &= \int_0^t \left(2e^{-t-\tau} - 2e^{-3t+\tau}\right)\, d\tau \\ &= 2e^{-t}(-1)\left(e^{-t} - 1\right) - 2e^{-3t}\left(e^{t} - 1\right) \end{aligned}$$

Therefore, the result is

$$y(t) = \left(2e^{-t} - 4e^{-2t} + 2e^{-3t}\right)\sigma(t)$$

The factor $\sigma(t)$ is added so that the result is the same as derived above for $t \geq 0$, and so that $y(t) = 0$ for $t < 0$, as required.

Solution S3

Determine and Plot $C_L(t)$

$$C_L(t) = 2\pi \alpha_0 \psi(\bar{t}) \quad \text{where } \bar{t} = 2Ut/c$$

$$\psi(\bar{t}) = \begin{cases} 0 & \bar{t} < 0 \\ 1 - \frac{1}{2}e^{-.13\bar{t}} - \frac{1}{2}e^{-\bar{t}} & \bar{t} \geq 0 \end{cases}$$

$\alpha = w/U$ if w is small compared to U

Known values: $c = 1\,m$ $U = 1\,m/s$ we have

$\alpha = w$ & $\bar{t} = 2t$

given w which is

$$w(t) = \begin{cases} 0\ m/s & t < 0\,s \\ .1(1 - e^{-2t})\ m/s & t \geq 0\,s \end{cases}$$

now lets also put $\psi(\bar{t})$ into the time domain

$$\psi(t) = \begin{cases} 0 & t < 0 \\ 1 - \frac{1}{2}{}^{-.26t} - \frac{1}{2}e^{-2t} & t \geq 0 \end{cases}$$

now since $\psi(t)$ is a step responce let's use Duhamel's superposition integr

$$y(t) = \psi(t)\,w(0) + \int_0^{\infty} \psi(t-\tau)\,w'(\tau)\,d\tau$$

to solve this we need $w'(t)$ so

$$w'(t) = \begin{cases} 0 & t < 0 \\ .2e^{-2t} & t \geq 0 \end{cases} = .2e^{-2t}\sigma(t)$$

also $w(0) = 0$ thus we have

$$y(t) = \int_0^t (1 - \tfrac{1}{2}e^{-.76(t-\tau)} - \tfrac{1}{2}e^{-2(t-\tau)}(.02e^{-2\tau})\,d\tau$$

multipling and rearranging

$$y(t) = \int_0^t .02e^{-2\tau} - .01e^{-1.74\tau}e^{-.76t} - .1e^{-2t}\,d\tau$$

integrating and evaluting from 0 to t

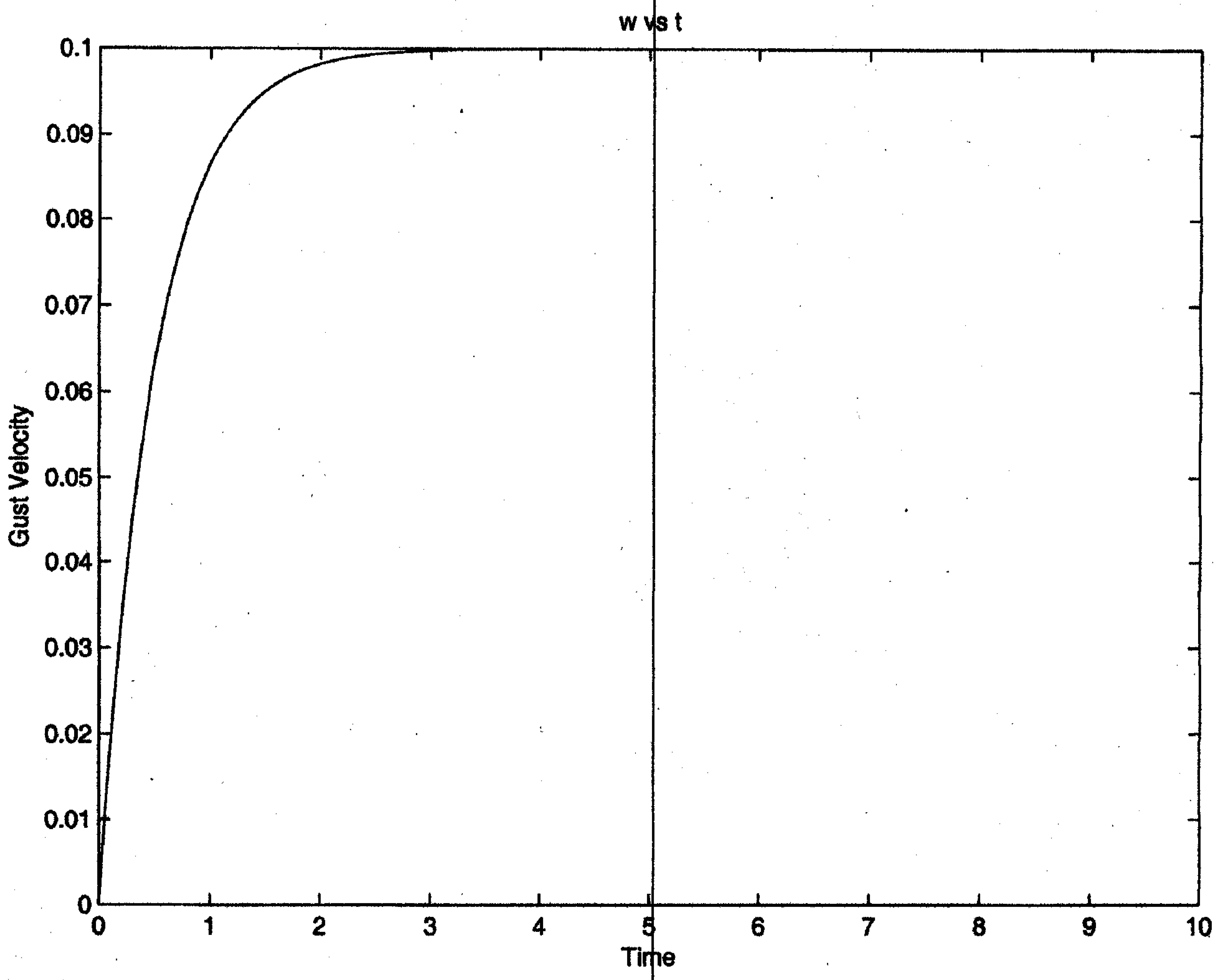
$$Y(t) = -.1e^{-2t} + \frac{.1}{1.74}e^{-2t} - .1te^{-2t} + .1 - \frac{.1}{1.74}e^{-.26t}$$

$$\Rightarrow \quad y(t) = (-.0425e^{-2t} - .0575e^{-.76t} - .01e^{-2t}t + .1)\,\sigma(t)$$

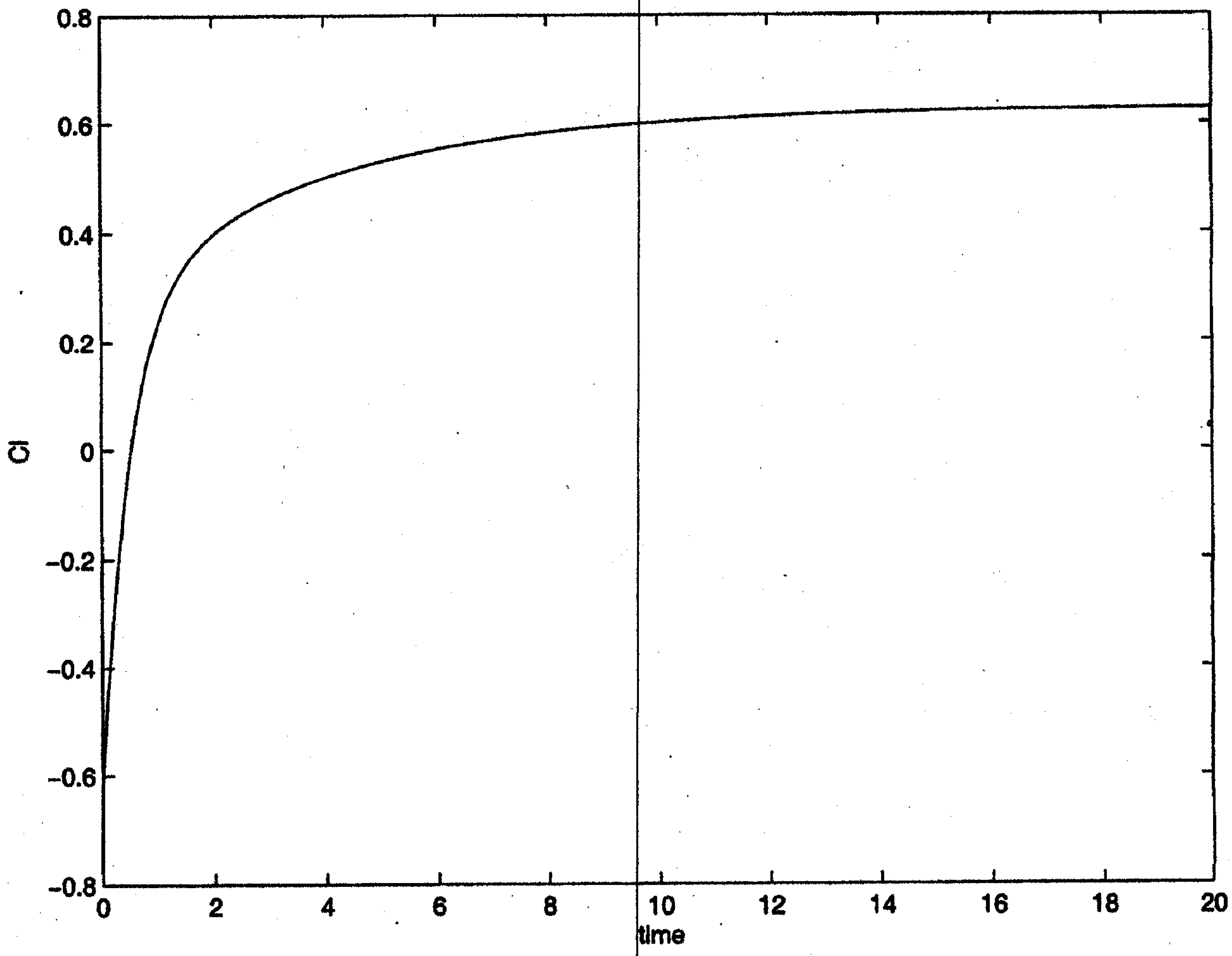
and we know $C_L = 2\pi y(t)$ because $\omega = \delta$

so $$\boxed{C_L(t) = 2\pi(-.0425e^{-2t} - .0575e^{-.026t} - .1te^{-2t} + .1)\,\sigma(t)}$$

plotted on the following two pages


w vs t
Gust Velocity
Time
0.1
0.09
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
0 1 2 3 4 5 6 7 8 9 10

Cl vs. t



Unified Engineering II **Spring 2005**

Problem S4 Solution (Signals and Systems)

1. Find the step response of the system

$$\frac{d^2}{dt^2}y(t) + 3\frac{d}{dt}y(t) + 2y(t) = u(t)$$

Solution. First, find the homogeneous solution, that is, the solution to

$$\frac{d^2}{dt^2}y(t) + 3\frac{d}{dt}y(t) + 2y(t) = 0$$

If we assume an exponential solution of the form $y(t) = e^{st}$, then the characteristic equation becomes

$$s^2 + 3s + 2 = 0$$

The roots are $s_1 = -1$, $s_2 = -2$. The general homogeneous olutions is then

$$y_h(t) = c_1 e^{-t} + c_2 e^{-2t}$$

Next, we find the particular solution, for the input $u(t) = 1$, $t > 0$. Guess that the solution is a constant, $y(t) = c$. Plugging into the d.e. yields

$$2c = 1$$

and therefore $c = 1/2$. Therefore, the total solution is

$$y(t) = y_h(t) + y_p(t) = \frac{1}{2} + c_1 e^{-t} + c_2 e^{-2t}$$

The inital conditions are $y(0) = y^{'}(0) = 0$. Solving for the constants, $c_1 = -1$, $c_2 = 1/2$. Therefore,

$$g_s(t) = \left(\frac{1}{2} - e^{-t} + \frac{1}{2}\,e^{-2t}\right)\sigma(t)$$

2. Take the derivative of the step response to find the impulse response.

Solution. Take the derivative using the chain rule for multiplication

$$\begin{aligned} g(t) &= \frac{d}{dt}\left[\left(\frac{1}{2} - e^{-t} + \frac{1}{2}\,e^{-2t}\right)\sigma(t)\right] \\ &= \left(e^{-t} - e^{-2t}\right)\sigma(t) + \left(\frac{1}{2} - e^{-t} + \frac{1}{2}\,e^{-2t}\right)\delta(t) \end{aligned}$$

Since $\frac{1}{2} - e^{-t} + \frac{1}{2}\,e^{-2t} = 0$ at $t = 0$,

$$g(t) = \left(e^{-t} - e^{-2t}\right)\sigma(t)$$

3. Now assume that the input is given by

$$u(t) = e^{-2t}\sigma(t)$$

Before doing part (4), try to find the particular solution by the usual method, that is, by intelligent guessing. Be careful — it may not be what you expect!

Solution. The obvious thing to do is to guess that

$$y_p(t) = c\,e^{-2t}$$

But we already know that e^{-2t} is a solution to the homogeneouos equations. It can't be a particular solution! So instead, guess that

$$y_p(t) = c\,t\,e^{-2t}$$

Plugging into the differential equation gives

$$\begin{aligned}\frac{d^2}{dt^2}\,y_p(t) + 3\frac{d}{dt}\,y_p(t) + 2y_p(t) &= -c\,e^{-2t} \\ &= u(t) = e^{-2t}\end{aligned}$$

Therefore, $c = -1$, and

$$y_p(t) = -t\,e^{-2t}$$

4. Now find $y(t)$ using the superposition integral. Is the particular solution what you expected?

Solution. Perform the convolution integral,

$$y(t) = \int_{-\infty}^{\infty} g(t-\tau)u(\tau)\,d\tau$$

The result is

$$\begin{aligned}y(t) &= \int_0^t \left(e^{-(t-\tau)} - e^{-2(t-\tau)}\right) e^{-2\tau}\,d\tau \\ &= \left(e^{-t} - e^{-2t} - te^{-2t}\right)\sigma(t)\end{aligned}$$

Note that the correct particular solution is in the final result.