

Program 1 Report

Kelson A. Petersen

October 3, 2025

Contents

1	Problem 1	2
2	Problem 2	3
3	Problem 3	4
4	Code	5

1 Problem 1

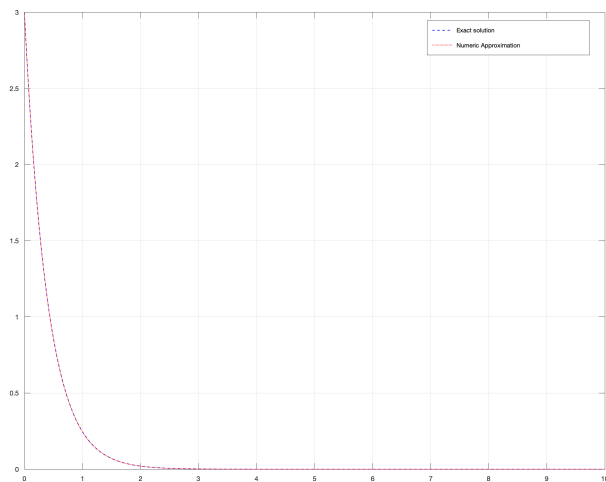


Figure 1: The Exact solution compared to the numeric approximation of Problem 1.

2 Problem 2

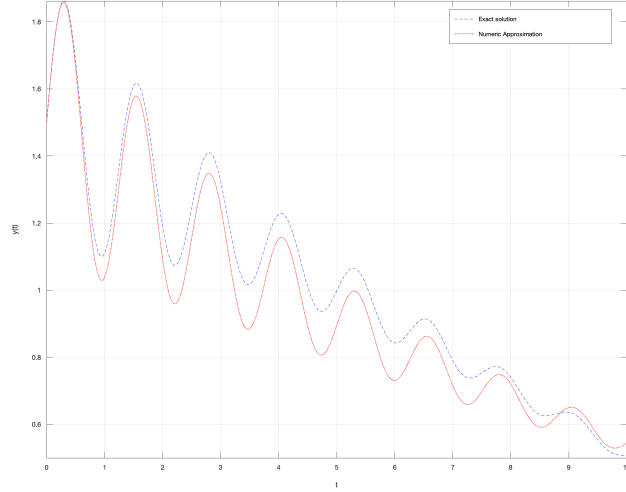


Figure 2: The Exact solution compared to the numeric approximation in Problem 2.

$$\frac{21156110196 \sqrt{10009} e^{-\frac{3t}{10}} \sin\left(\frac{\sqrt{10009}t}{20}\right)}{5049641841125} - \frac{5221024 e^{-\frac{3t}{10}} \cos\left(\frac{\sqrt{10009}t}{20}\right)}{504510125} - \frac{25063t}{251125} + \frac{1523972423}{1009020250} \quad (1)$$

3 Problem 3

4 Code

In this section there will be brief descriptions of the code followed by the code. I split the program into multiple files to have smaller blocks of code in this document. This code could all be included in one file just as easily.

The first file included here is a file named 'main.h'. This file is the header file that contains the preview of the function for the compiler allowing the definitions of the functions to be in different files.

```
1 #ifndef MAIN_H
2 #define MAIN_H
3
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <stdbool.h>
7
8 // Functions for the problems
9 int problem_1(void);
10 int problem_2(void);
11 int problem_3(void);
12
13 // Matrix and Vector Functions
14 void mat_scale(double scale, double mat[][3], double prod[][3]);
15 void mat_sub(double left[][3], double right[][3], double diff[][3]);
16 void mat_add(double left[][3], double right[][3], double sum[][3]);
17 void mat_vec_mult(double mat[][3], double *vector, double *prod);
18
19 #endif
```

The following code is the entry point of the program that simply calls the correct function that corresponds to the problem selected by the user. There is basic error messages that are printed if the function called fails to create the necessary files.

```
1 #include "main.h"
2
3 /*****
4  * Function Title: main
5  * Summary: This function facilitates the selection and calling of a problem
6  *          function and prints relevant error statements
7  *
8  * Inputs:
9  *   int argc: the amount of arguments passed to main
10  *   char** argv: the arguments passed as a string to main
11  * Outputs:
12  *   int: an error code to the operating system on program execution
13  *
14  * Compile Instructions: gcc main.c -o main
15  *****/
16 * Pseudocode
17 *   Begin
18 *   if there was no number passed in the command line
19 *   prompt input for problem selection
20 *   else
21 *   convert command line input
22 *   call correct problem number 1,2, or 3
23 *   call function
24 *   if it returns error
25 *   print Error
26 *   else
```

```

27 *          print success
28 *      End
29 *****/
30 int main(int argc, char** argv) {
31     // Begin Program
32     int problem;
33     // If no arguments prompt and receive input
34     if (argc < 2) {
35         printf("Enter problem number: "); // Prompt user to select problem
36         scanf("%d", &problem);           // accept user input
37     } else {
38         // else use command line argument
39         problem = atoi(argv[1]); // Assign the value from the command line
40     }
41     // Do the selected problem
42     switch (
43         problem) { // if the problem fails to make a file print Error statement
44     case 1:
45         (problem_1() == 0) ? printf("Problem 1 success\n")
46                             : printf("Error in Problem 1\n");
47         break;
48     case 2:
49         (problem_2() == 0) ? printf("Problem 2 success\n")
50                             : printf("Error in Problem 2\n");
51         break;
52     case 3:
53         (problem_3() == 0) ? printf("Problem 3 success\n")
54                             : printf("Error in Problem 3\n");
55         break;
56     default:
57         printf("Error please choose 1-3.\n");
58     }
59     // End Program
60     return EXIT_SUCCESS;
61 }

```

The following code is the code solution for problem one. This code was mostly given in the program assignment description, but the loop creates a file containing x and y coordinates to generate a graph.

```

1  #include "main.h"
2
3  // Problem 1 Code
4  int problem_1(void) {
5      FILE* fout = fopen("prog_sol_1.txt", "w");
6      if (fout == NULL) {
7          perror("output file failed");
8          return EXIT_FAILURE;
9      }
10     // Declare and Initialize variables
11     double y      = 3.0;
12     double delta_t = 0.001;
13     double time    = 0;
14     double a       = -2.5;
15     // This line doesn't change in the loop
16     a = (1 + a * delta_t);
17     // Do the math iteratively in a loop
18     for (time = 0.0; time < 10.0; time += delta_t) {
19         // Print the t, y(t) coordinate
20         fprintf(fout, "%0.3lf\t%0.10lf\n", time, y);
21         // Do the calculation

```

```

22     y = a * y;
23 }
24 return EXIT_SUCCESS;
25 }

```

The following code is the code solution for problem two. This is an extension of the code in problem one, but for higher order equations. We are able to use the state-space form of a differential equation to make this process work. We put the equation in the state-space form by using matrices.

```

1  #include "main.h"
2
3  // Problem 2 Code
4  int problem_2(void) {
5      FILE* fout = fopen("prog_sol_2.txt", "w");
6      if (fout == NULL) {
7          perror("output file failed");
8          return EXIT_FAILURE;
9      }
10     double I[][3] = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};
11     double A[][3] = {{0, 1, 0}, {0, 0, 1}, {-2.5063, -25.1125, -0.6}};
12     double x_t[3] = {1.5, 2, -1};
13     double delta_t = 0.001f;
14     double time;
15     // avoid unnecessary function calls in loop
16     mat_scale(delta_t, A, A);
17     mat_add(I, A, A);
18     // Do iterative math
19     for (time = 0.0; time < 10.0; time += delta_t) {
20         // print the result
21         fprintf(fout, "%.3lf\t%.10lf\n", time, x_t[0]);
22         // get next x_t value
23         mat_vec_mult(A, x_t, x_t);
24     }
25     return EXIT_SUCCESS;
26 }

```

The following code is the code for a differential equation that was derived from a RLC circuit in problem three. It is solve the same way as problem 2.

```

1  #include "main.h"
2
3  // Problem 3 Code
4  int problem_3(void) {
5      FILE* fout = fopen("prog_sol_3.txt", "w");
6      if (fout == NULL) {
7          perror("output file failed");
8          return EXIT_FAILURE;
9      }
10
11     return EXIT_SUCCESS;
12 }

```

The following code was given in the program 1 description. It is a collection of matrix operation functions, and the functions allow the matrix operations.

```

1  #include "main.h"
2
3  // scale matrix 'mat' by a and save in prod
4  void mat_scale(double scale, double mat[][3], double prod[][3]) {

```

```

5   int i, j;
6   for (i = 0; i < 3; ++i) {
7       for (j = 0; j < 3; ++j) {
8           prod[i][j] = scale * mat[i][j];
9       }
10  }
11 }
12
13 // subtract the right from the left and store in diff
14 void mat_sub(double left[][3], double right[][3], double diff[][3]) {
15     int i, j;
16     for (i = 0; i < 3; i++) {
17         for (j = 0; j < 3; j++) {
18             diff[i][j] = left[i][j] - right[i][j];
19         }
20     }
21 }
22
23 // add the right and the left and store into sum
24 void mat_add(double left[][3], double right[][3], double sum[][3]) {
25     int i, j;
26     for (i = 0; i < 3; i++) {
27         for (j = 0; j < 3; j++) {
28             sum[i][j] = left[i][j] + right[i][j];
29         }
30     }
31 }
32
33 // Unsafe if used incorrectly!
34 // multiplies 3x3 matrix mat with 3x1 vector storing into 3x1 prod
35 void mat_vec_mult(double mat[][3], double* vector, double* prod) {
36     double sum;
37     int i, j;
38     for (i = 0; i < 3; i++) {
39         sum = 0;
40         for (j = 0; j < 3; j++) {
41             sum += mat[i][j] * vector[j];
42         }
43         prod[i] = sum;
44     }
45 }

```

Below is the makefile that compiles the program.

```

1  CC = gcc
2  CFLAGS = -Werror
3
4  obj_files = vec_funcs.o prob1.o prob2.o prob3.o
5  src_files = main.c vec_funcs.c prob1.c prob2.c prob3.c
6
7  .PHONY: all, clean
8
9  main: main.c $(obj_files)
10     $(CC) main.c $(obj_files) $(CFLAGS) -o main
11
12 $(objects): %.o: %.c
13     $(CC) -c $^ -o $@ $(CFLAGS)
14
15 clean:
16     rm -rf *.o

```