

# Program 1 Report

Kelson A. Petersen, A02346768

October 4, 2025

## Contents

<b>1</b>	<b>Problem 1</b>	<b>2</b>
<b>2</b>	<b>Problem 2</b>	<b>3</b>
<b>3</b>	<b>Problem 3</b>	<b>5</b>
<b>4</b>	<b>Code</b>	<b>6</b>
<b>5</b>	<b>Scratch Work</b>	<b>9</b>

# 1 Problem 1

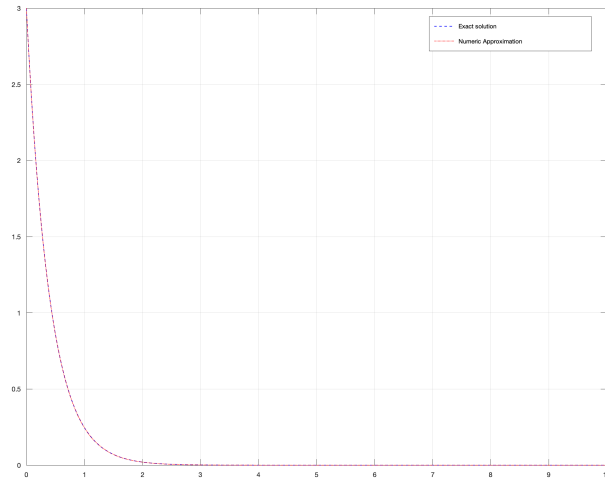


Figure 1: The Exact solution compared to the numeric approximation of Problem 1.

Using figure 1 above we see the C program approximate solution and the exact solution follow the same curve and we have a good approximation of the first order ODE given by  $(D + 2.5)y_0(t) = 0$  with the initial condition  $y_0(0) = 3$ . Scratch work is included at the end of the document.

The following code is the loop responsible for solving the ODE that was given above. The entirety of the code is included later in this document.

```
1 // Problem 1 Code
2 int problem_1(void) {
3     FILE* fout = fopen("./prog_out/prog_sol_1.txt", "w");
4     if (fout == NULL) {
5         perror("output file failed");
6         return EXIT_FAILURE;
7     }
8     // Declare and Initialize variables
9     double y      = 3.0;
10    double delta_t = 0.001;
11    double time     = 0;
12    double a        = -2.5;
13    // This line doesn't change in the loop
14    a = (1 + a * delta_t);
15    // Do the math iteratively in a loop
16    for (time = 0.0; time < 10.0; time += delta_t) {
17        // Print the t, y(t) coordinate
18        fprintf(fout, "%0.31f\t%0.101f\n", time, y);
19        // Do the calculation
20        y = a * y;
21    }
22    return EXIT_SUCCESS;
23 }
```

## 2 Problem 2

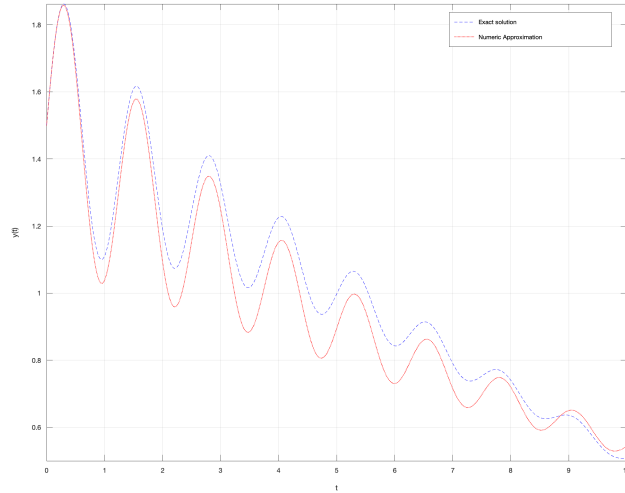


Figure 2: The Exact solution compared to the numeric approximation in Problem 2.

Using figure 2 above we compare the exact solution that is the blue dashed line and the approximate solution in the C code below follow a similar path; however, the approximation is less accurate than the exact solution in this case. The oscillatory nature of this function causes the ‘average’ to be less accurate. Below is the figure that has the roots of the exact solution.

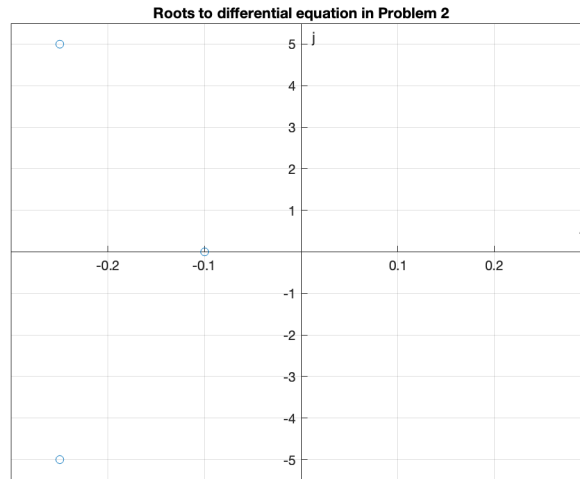


Figure 3: The roots of the differential equation for Problem 2.

Below is the equation that was found as the solution for the differential equation given by  $D^3 + 0.6D^3 + 25.1125D + 2.5063y_0(t) = 0$ . I used matlab to solve.

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

This is the specific code that was used to calculate the approximation.

```
1 // Problem 2 Code
2 int problem_2(void) {
3     FILE* fout = fopen("./prog_out/prog_sol_2.txt", "w");
```

```

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

```

### 3 Problem 3

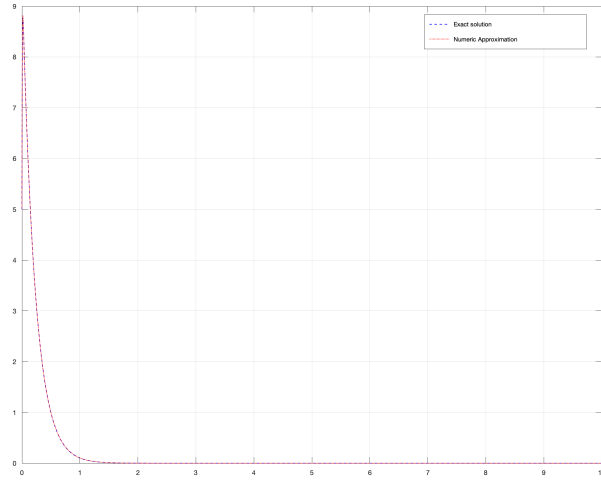


Figure 4: The Exact solution compared to the numeric approximation in Problem 3.

For the figure above we solved the circuit to find the differential equation that work is shown at the end of the document. The differential equation we end up with is  $f(t) = D^2LCR_2(\frac{1}{R+1} + \frac{1}{R_2}) + D(\frac{L}{R_1} + CR_2) + y(t)$ . We see the approximate solution worked well with this equation as it does not oscillate. If the circuit had a dependent source in it the analytical and numeric solutions would both become more complicated as the non-linearity of the circuit would create different frequency responses varying in a non-linear fashion. Additionally The numeric approximation I would expect to see diverge from the exact solution because the exact solution would not be linear.

Below is the code that did the calculation.

```
1 // Problem 3 Code
2 int problem_3(void) {
3     FILE* fout = fopen("./prog_out/prog_sol_3.txt", "w");
4     if (fout == NULL) {
5         perror("output file failed");
6         return EXIT_FAILURE;
7     }
8     double A[][2] = {{0, 1}, {195.651, 869.57}};
9     double I[][2] = {{1, 0}, {0, 1}};
10    double x_t[2] = {5, 847.83};
11    double delta_t = 0.001;
12    double time;
13
14    mat_scale2(delta_t, A, A);
15    mat_add2(I, A, A);
16    for (time = 0.0; time <= 10.0; time += delta_t) {
17        fprintf(fout, "0.31f\t0.101f\n", time, x_t[0]);
18        mat_vec_mult2(A, x_t, x_t);
19    }
20    return EXIT_SUCCESS;
21 }
```

## 4 Code

```
1 #include <stdbool.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 // Functions for the problems
6 int problem_1(void);
7 int problem_2(void);
8 int problem_3(void);
9 // 2x2 Matrix and Vector Functions
10 void mat_scale2(double scale, double mat[][2], double prod[][2]);
11 void mat_sub2(double left[][2], double right[][2], double diff[][2]);
12 void mat_add2(double left[][2], double right[][2], double sum[][2]);
13 void mat_vec_mult2(double mat[][2], double* vector, double* prod);
14 // 3x3 Matrix and Vector Functions
15 void mat_scale(double scale, double mat[][3], double prod[][3]);
16 void mat_sub(double left[][3], double right[][3], double diff[][3]);
17 void mat_add(double left[][3], double right[][3], double sum[][3]);
18 void mat_vec_mult(double mat[][3], double* vector, double* prod);
19
20 int main(int argc, char** argv) {
21     // Begin Program
22     int problem;
23     // If no arguments prompt and receive input
24     if (argc < 2) {
25         printf("Enter problem number: "); // Prompt user to select problem
26         scanf("%d", &problem);           // accept user input
27     } else {
28         // else use command line argument
29         problem = atoi(argv[1]); // Assign the value from the command line
30     }
31     // Do the selected problem
32     switch (problem) {
33         case 1: // Print success or fail for problem execution
34             !problem_1() ? printf("Problem 1 success\n")
35                           : printf("Error in Problem 1\n");
36             break;
37         case 2: //
38             !problem_2() ? printf("Problem 2 success\n")
39                           : printf("Error in Problem 2\n");
40             break;
41         case 3:
42             !problem_3() ? printf("Problem 3 success\n")
43                           : printf("Error in Problem 3\n");
44             break;
45         default:
46             printf("Error please choose 1-3.\n");
47     }
48     // End Program
49     return EXIT_SUCCESS;
50 }
51 // Problem 1 Code
52 int problem_1(void) {
53     FILE* fout = fopen("./prog_out/prog_sol_1.txt", "w");
54     if (fout == NULL) {
55         perror("output file failed");
56         return EXIT_FAILURE;
57     }
58     // Declare and Initialize variables
```

```

59 double y          = 3.0;
60 double delta_t    = 0.001;
61 double time       = 0;
62 double a          = -2.5;
63 // This line doesn't change in the loop
64 a = (1 + a * delta_t);
65 // Do the math iteratively in a loop
66 for (time = 0.0; time < 10.0; time += delta_t) {
67     // Print the t, y(t) coordinate
68     fprintf(fout, "%0.31f\t%0.101f\n", time, y);
69     // Do the calculation
70     y = a * y;
71 }
72 fclose(fout);
73 return EXIT_SUCCESS;
74 }
75 // Problem 2 Code
76 int problem_2(void) {
77     FILE* fout = fopen("./prog_out/prog_sol_2.txt", "w");
78     if (fout == NULL) {
79         perror("output file failed");
80         return EXIT_FAILURE;
81     }
82     double I[][3] = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};
83     double A[][3] = {{0, 1, 0}, {0, 0, 1}, {-2.5063, -25.1125, -0.6}};
84     double x_t[3] = {1.5, 2, -1};
85     double delta_t = 0.001f;
86     double time;
87     // avoid unnecessary function calls in loop
88     mat_scale(delta_t, A, A);
89     mat_add(I, A, A);
90     // Do iterative math
91     for (time = 0.0; time < 10.0; time += delta_t) {
92         // print the result
93         fprintf(fout, "%0.31f\t%0.101f\n", time, x_t[0]);
94         // get next x_t value
95         mat_vec_mult(A, x_t, x_t);
96     }
97     fclose(fout);
98     return EXIT_SUCCESS;
99 }
100 // Problem 3 Code
101 int problem_3(void) {
102     FILE* fout = fopen("./prog_out/prog_sol_3.txt", "w");
103     if (fout == NULL) {
104         perror("output file failed");
105         return EXIT_FAILURE;
106     }
107     double A[][2] = {{0, 1}, {-869.57, -195.651}};
108     double I[][2] = {{1, 0}, {0, 1}};
109     double x_t[2] = {5, 847.83};
110     double delta_t = 0.001;
111     double time;
112
113     mat_scale2(delta_t, A, A);
114     mat_add2(I, A, A);
115     for (time = 0.0; time <= 10.0; time += delta_t) {
116         fprintf(fout, "%0.31f,%0.101f\n", time, x_t[0]);
117         mat_vec_mult2(A, x_t, x_t);
118     }

```

```

119     return EXIT_SUCCESS;
120 }
121 // scale matrix 'mat' by a and save in prod
122 void mat_scale(double scale, double mat[][3], double prod[][3]) {
123     int i, j;
124     for (i = 0; i < 3; ++i) {
125         for (j = 0; j < 3; ++j) {
126             prod[i][j] = scale * mat[i][j];
127         }
128     }
129 }
130 // subtract the right from the left and store in diff
131 void mat_sub(double left[][3], double right[][3], double diff[][3]) {
132     int i, j;
133     for (i = 0; i < 3; i++) {
134         for (j = 0; j < 3; j++) {
135             diff[i][j] = left[i][j] - right[i][j];
136         }
137     }
138 }
139 // add the right and the left and store into sum
140 void mat_add(double left[][3], double right[][3], double sum[][3]) {
141     int i, j;
142     for (i = 0; i < 3; i++) {
143         for (j = 0; j < 3; j++) {
144             sum[i][j] = left[i][j] + right[i][j];
145         }
146     }
147 }
148 // Unsafe if used incorrectly!
149 // multiplies 3x3 matrix mat with 3x1 vector storing into 3x1 prod
150 void mat_vec_mult(double mat[][3], double* vector, double* prod) {
151     double sum;
152     int i, j;
153     for (i = 0; i < 3; i++) {
154         sum = 0;
155         for (j = 0; j < 3; j++) {
156             sum += mat[i][j] * vector[j];
157         }
158         prod[i] = sum;
159     }
160 }
161
162 // scale matrix 'mat' by a and save in prod
163 void mat_scale2(double scale, double mat[][2], double prod[][2]) {
164     int i, j;
165     for (i = 0; i < 2; ++i) {
166         for (j = 0; j < 2; ++j) {
167             prod[i][j] = scale * mat[i][j];
168         }
169     }
170 }
171 // subtract the right from the left and store in diff
172 void mat_sub2(double left[][2], double right[][2], double diff[][2]) {
173     int i, j;
174     for (i = 0; i < 3; i++) {
175         for (j = 0; j < 3; j++) {
176             diff[i][j] = left[i][j] - right[i][j];
177         }
178     }

```



```

179 }
180 // add the right and the left and store into sum
181 void mat_add2(double left[][2], double right[][2], double sum[][2]) {
182     int i, j;
183     for (i = 0; i < 2; i++) {
184         for (j = 0; j < 2; j++) {
185             sum[i][j] = left[i][j] + right[i][j];
186         }
187     }
188 }
189 // Unsafe if used incorrectly!
190 // multiplies 3x3 matrix mat with 3x1 vector storing into 3x1 prod
191 void mat_vec_mult2(double mat[][2], double* vector, double* prod) {
192     double sum;
193     int i, j;
194     for (i = 0; i < 2; i++) {
195         sum = 0;
196         for (j = 0; j < 2; j++) {
197             sum += mat[i][j] * vector[j];
198         }
199         prod[i] = sum;
200     }
201 }

```

## 5 Scratch Work

1.)

a.)

$$(D+2.5)y(t) = 0$$

$$\lambda = -2.5$$

$$C_1 e^{-2.5t}$$

$$C_1 e^0 = 3$$

$$C_1 = 3$$

$$\underline{\underline{y(t) = 3e^{-2.5t}}}$$

2.)

$$(D^3 + 0.6D^2 + 25.1125D + 2.5D63)y(t) = 0$$

$$y_0(0) = 1.5, \quad \dot{y}_0(0) = 2, \quad \ddot{y}_0(0) = -1$$

use Matlab to solve for Roots

$$\lambda = -0.1$$

$$\lambda = -0.25 + j5$$

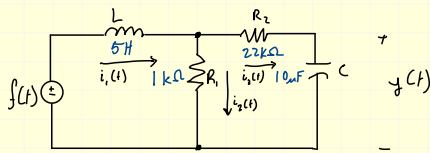
$$\lambda = -0.25 - j5$$

$$-0.1, -0.25 + j5, -0.25 - j5$$

$$C_1 e^{-0.1t} + C_2 e^{(-0.25+j5)t} + C_3 e^{(-0.25-j5)t}$$

using matlab to solve and find  
equation 1.

$$x(t) = \left( \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \Delta t \cdot \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -2.5063 & -25.1125 & -0.6 \end{bmatrix} \right) \begin{bmatrix} 1.5 \\ 2 \\ -1 \end{bmatrix}$$



(a)

$$i_1(t) = i_2(t) + i_3(t)$$

$$i_2(t) = \frac{v - y(t)}{R_2} = C \frac{dy(t)}{dt} + y(t)$$

$$i_3(t) = \frac{v}{R_1}$$

$$i_1 = \frac{v}{R_1} + \frac{v - y(t)}{R_2}$$

$$\frac{di_1(t)}{dt} = \frac{1}{R_1} \frac{dv}{dt} + \frac{1}{R_2} \left( \frac{dv}{dt} - \frac{dy(t)}{dt} \right)$$

$$f(t) = L \frac{di_1(t)}{dt} + v$$

$$f(t) = L \left[ \frac{1}{R_1} \frac{dv}{dt} + \frac{1}{R_2} \left( \frac{dv}{dt} - \frac{dy}{dt} \right) \right] + v$$

$$f(t) = L \left( \frac{1}{R_1} + \frac{1}{R_2} \right) \frac{dv}{dt} - \frac{L}{R_2} \frac{dy}{dt} + v$$

$$v = y + CR_2 \frac{dy}{dt} \Rightarrow \frac{dv}{dt} = \frac{dy}{dt} + CR_2 \frac{d^2 y}{dt^2}$$

$$f(t) = L \left( \frac{1}{R_1} + \frac{1}{R_2} \right) \left( \frac{dy}{dt} + CR_2 \frac{d^2 y}{dt^2} \right) - \frac{L}{R_2} \frac{dy}{dt} + y + CR_2 \frac{dy}{dt}$$

use D operators

$$f(t) = D L \left( \frac{1}{R_1} + \frac{1}{R_2} \right) + D^2 L C R_2 \left( \frac{1}{R_1} + \frac{1}{R_2} \right) - D \frac{L}{R_2} + y + D C R_2$$

$$f(t) = D^2 L C R_2 \left( \frac{1}{R_1} + \frac{1}{R_2} \right) + D \left( \frac{L}{R_2} + C R_2 \right) + y$$

(b)

$$L = 5 \text{ H}$$

$$C = 10 \mu\text{F}$$

$$R_2 = 22 \text{ k}\Omega$$

$$R_1 = 1 \text{ k}\Omega$$

$$1.1 \left( \frac{1}{1\text{k}} + \frac{1}{22\text{k}} \right) D^2 + (0.005 + 0.22) D + y$$

$$\frac{23}{20,000} D^2 + \frac{9}{40} D + y$$

$$\lambda_1 = -4.5503$$

$$\lambda_2 = -191.1019$$

$$C_1 e^{-4.5503t} + C_2 e^{-191.1019t} = 0$$

$$= 847.83$$

$$v = CR_2 \left( \frac{dy(t)}{dt} \right) + y(t)$$

$$i_1(0) = \frac{y(0)}{R_1} + \left( \frac{R_2 C}{R_1} \right) y'(0)$$

$$i_1(0) = \frac{0.2 - \frac{5}{1000}}{\frac{22,000}{1000}}$$

(c)

$$C_1 + C_2 = 5$$

$$-4.5503 C_1 + -191.1019 C_2 = 847.83$$

$$\begin{bmatrix} 1 & 1 & 5 \\ -4.5503 & -191.1019 & 847.83 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & -181.105 \\ 0 & 1 & -4.5503 \end{bmatrix}$$

$$\frac{9.6667 e^{-4.5503t} - 4.667 e^{-191.1019t}}{1000}$$

(d)

$$\left( \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \Delta t \cdot \begin{bmatrix} 0 & 1 \\ -869.57 & -195.651 \end{bmatrix} \right) \cdot \begin{bmatrix} 5 \\ 847.83 \end{bmatrix}$$