

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

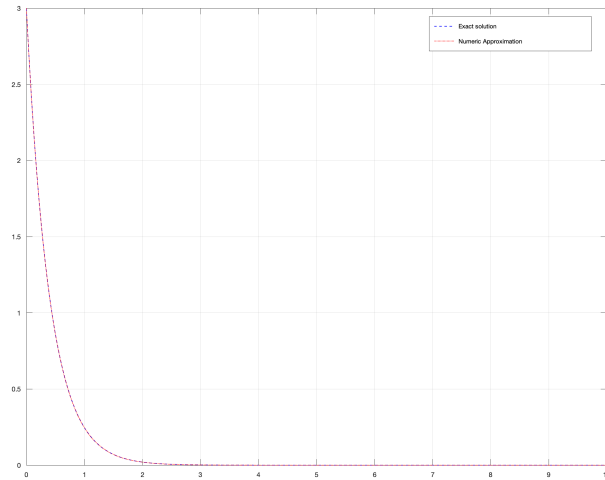


Figure 1: The Exact solution compared to the numeric approximation of Problem 1.

```
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

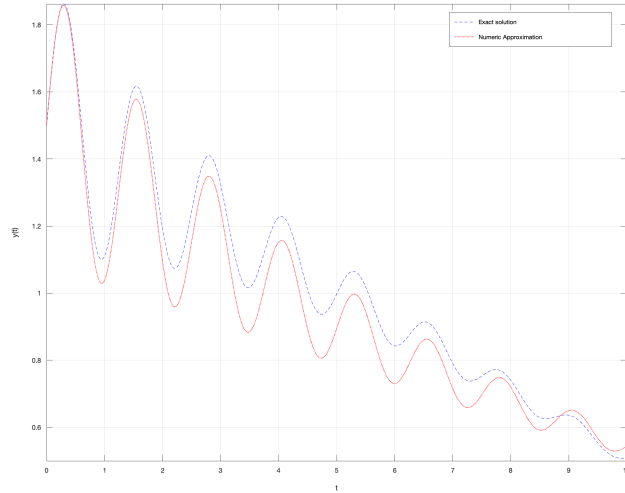


Figure 2: The Exact solution compared to the numeric approximation in Problem 2.

```

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, "%.31f\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 }

```

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

3 Problem 3

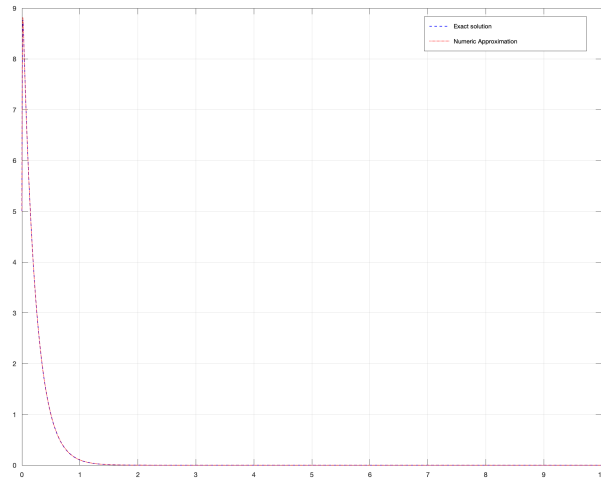


Figure 3: The Exact solution compared to the numeric approximation in Problem 3.

```
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 C and Matlab Code Used

```
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
10 // Matrix and Vector Functions
11 void mat_scale2(double scale, double mat[][2], double prod[][2]);
12 void mat_sub2(double left[][2], double right[][2], double diff[][2]);
13 void mat_add2(double left[][2], double right[][2], double sum[][2]);
14 void mat_vec_mult2(double mat[][2], double* vector, double* prod);
15
16 void mat_scale(double scale, double mat[][3], double prod[][3]);
17 void mat_sub(double left[][3], double right[][3], double diff[][3]);
18 void mat_add(double left[][3], double right[][3], double sum[][3]);
19 void mat_vec_mult(double mat[][3], double* vector, double* prod);
20
21 int main(int argc, char** argv) {
22     // Begin Program
23     int problem;
24     // If no arguments prompt and receive input
25     if (argc < 2) {
26         printf("Enter problem number: "); // Prompt user to select problem
27         scanf("%d", &problem);           // accept user input
28     } else {
29         // else use command line argument
30         problem = atoi(argv[1]); // Assign the value from the command line
31     }
32     // Do the selected problem
33     switch (
34         problem) { // if the problem fails to make a file print Error statement
35     case 1:
36         (problem_1() == 0) ? printf("Problem 1 success\n")
37                             : printf("Error in Problem 1\n");
38         break;
39     case 2:
40         (problem_2() == 0) ? printf("Problem 2 success\n")
41                             : printf("Error in Problem 2\n");
42         break;
43     case 3:
44         (problem_3() == 0) ? printf("Problem 3 success\n")
45                             : printf("Error in Problem 3\n");
46         break;
47     default:
48         printf("Error please choose 1-3.\n");
49     }
50     // End Program
51     return EXIT_SUCCESS;
52 }
53 // Problem 1 Code
54 int problem_1(void) {
55     FILE* fout = fopen("./prog_out/prog_sol_1.txt", "w");
56     if (fout == NULL) {
57         perror("output file failed");
58         return EXIT_FAILURE;
59     }
```

```

59 }
60 // Declare and Initialize variables
61 double y = 3.0;
62 double delta_t = 0.001;
63 double time = 0;
64 double a = -2.5;
65 // This line doesn't change in the loop
66 a = (1 + a * delta_t);
67 // Do the math iteratively in a loop
68 for (time = 0.0; time < 10.0; time += delta_t) {
69     // Print the t, y(t) coordinate
70     fprintf(fout, "%0.31f\t%0.101f\n", time, y);
71     // Do the calculation
72     y = a * y;
73 }
74 return EXIT_SUCCESS;
75 }
76 // Problem 2 Code
77 int problem_2(void) {
78     FILE* fout = fopen("./prog_out/prog_sol_2.txt", "w");
79     if (fout == NULL) {
80         perror("output file failed");
81         return EXIT_FAILURE;
82     }
83     double I[][3] = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};
84     double A[][3] = {{0, 1, 0}, {0, 0, 1}, {-2.5063, -25.1125, -0.6}};
85     double x_t[3] = {1.5, 2, -1};
86     double delta_t = 0.001f;
87     double time;
88     // avoid unnecessary function calls in loop
89     mat_scale(delta_t, A, A);
90     mat_add(I, A, A);
91     // Do iterative math
92     for (time = 0.0; time < 10.0; time += delta_t) {
93         // print the result
94         fprintf(fout, "%.31f\t%.101f\n", time, x_t[0]);
95         // get next x_t value
96         mat_vec_mult(A, x_t, x_t);
97     }
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}, {-195.651, -869.57}};
108     double A[][2] = {{0, 1}, {-869.57, -195.651}};
109     double I[][2] = {{1, 0}, {0, 1}};
110     double x_t[2] = {5, 847.83};
111     double delta_t = 0.001;
112     double time;
113
114     mat_scale2(delta_t, A, A);
115     mat_add2(I, A, A);
116     for (time = 0.0; time <= 10.0; time += delta_t) {
117         fprintf(fout, "%0.31f,%0.101f\n", time, x_t[0]);
118         mat_vec_mult2(A, x_t, x_t);

```

```

119     }
120     return EXIT_SUCCESS;
121 }
122 // scale matrix 'mat' by a and save in prod
123 void mat_scale(double scale, double mat[][3], double prod[][3]) {
124     int i, j;
125     for (i = 0; i < 3; ++i) {
126         for (j = 0; j < 3; ++j) {
127             prod[i][j] = scale * mat[i][j];
128         }
129     }
130 }
131 // subtract the right from the left and store in diff
132 void mat_sub(double left[][3], double right[][3], double diff[][3]) {
133     int i, j;
134     for (i = 0; i < 3; i++) {
135         for (j = 0; j < 3; j++) {
136             diff[i][j] = left[i][j] - right[i][j];
137         }
138     }
139 }
140 // add the right and the left and store into sum
141 void mat_add(double left[][3], double right[][3], double sum[][3]) {
142     int i, j;
143     for (i = 0; i < 3; i++) {
144         for (j = 0; j < 3; j++) {
145             sum[i][j] = left[i][j] + right[i][j];
146         }
147     }
148 }
149 // Unsafe if used incorrectly!
150 // multiplies 3x3 matrix mat with 3x1 vector storing into 3x1 prod
151 void mat_vec_mult(double mat[][3], double* vector, double* prod) {
152     double sum;
153     int i, j;
154     for (i = 0; i < 3; i++) {
155         sum = 0;
156         for (j = 0; j < 3; j++) {
157             sum += mat[i][j] * vector[j];
158         }
159         prod[i] = sum;
160     }
161 }
162
163 // scale matrix 'mat' by a and save in prod
164 void mat_scale2(double scale, double mat[][2], double prod[][2]) {
165     int i, j;
166     for (i = 0; i < 2; ++i) {
167         for (j = 0; j < 2; ++j) {
168             prod[i][j] = scale * mat[i][j];
169         }
170     }
171 }
172 // subtract the right from the left and store in diff
173 void mat_sub2(double left[][2], double right[][2], double diff[][2]) {
174     int i, j;
175     for (i = 0; i < 3; i++) {
176         for (j = 0; j < 3; j++) {
177             diff[i][j] = left[i][j] - right[i][j];
178         }
179     }
180 }

```

```

179     }
180 }
181 // add the right and the left and store into sum
182 void mat_add2(double left[][2], double right[][2], double sum[][2]) {
183     int i, j;
184     for (i = 0; i < 2; i++) {
185         for (j = 0; j < 2; j++) {
186             sum[i][j] = left[i][j] + right[i][j];
187         }
188     }
189 }
190 // Unsafe if used incorrectly!
191 // multiplies 3x3 matrix mat with 3x1 vector storing into 3x1 prod
192 void mat_vec_mult2(double mat[][2], double* vector, double* prod) {
193     double sum;
194     int i, j;
195     for (i = 0; i < 2; i++) {
196         sum = 0;
197         for (j = 0; j < 2; j++) {
198             sum += mat[i][j] * vector[j];
199         }
200         prod[i] = sum;
201     }
202 }

```