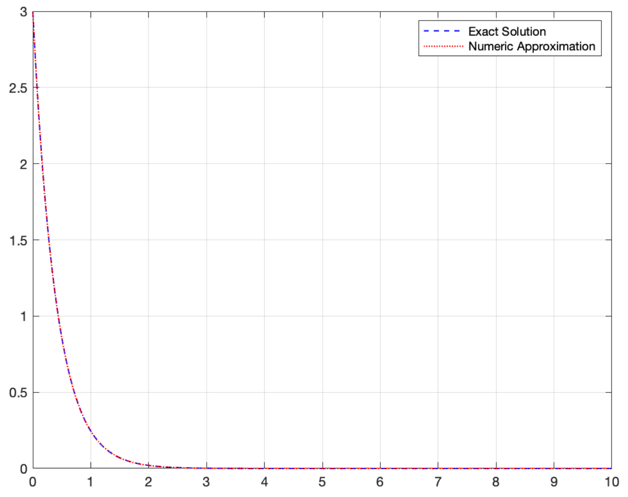

Figure 1: The Exact solution compared to the numeric approximation of Problem 1.

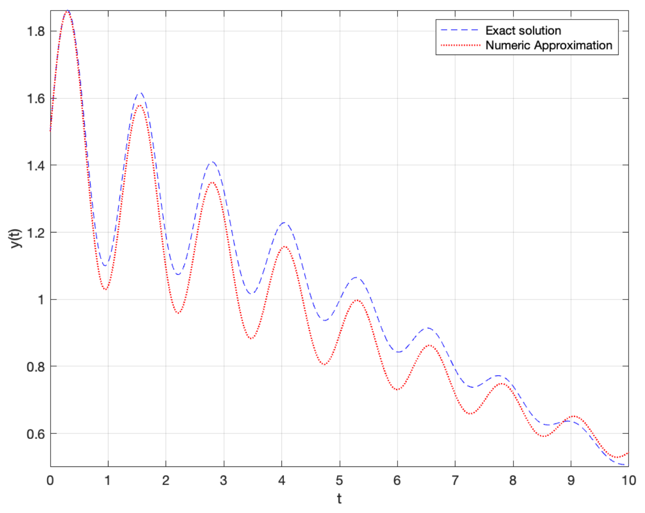

Figure 2: The Exact solution compared to the numeric approximation in Problem 2.

```c
#include "main.h"

/*****************************************************************************
 * Function Title: main
 * Summary: This funcion facilitates the selection and calling of a problem
 *          function and prints relevant error statements
 *
 * Inputs:
 *     int argc: the amount of arguements passed to main
 *     char** argv: the arguements passed as a string to main
 * Outputs:
 *     int: an error code to the operating system on program execution
 *
 * Compile Instructions: gcc main.c -o main
 *****************************************************************************
 * Pseudocode
 *   Begin
 *     if there was no number passed in the command line
 *       prompt input for problem selection
 *     else
 *       convert command line input
 *     call correct problem number 1,2, or 3
 *       call function
 *       if it returns error
 *         print Error
 *       else
 *         print success
 *   End
 ********************************************************************************/
int main(int argc, char** argv) {
  //  Begin Program
  int problem;
  // If no arguements prompt and recieve input
  if (argc < 2) {
    printf("Enter problem number: ");  // Prompt user to select problem
    scanf("%d", &problem);             // accept user input
  } else {
    // else use command line arguement
    problem = atoi(argv[1]); // Assign the value from the command line
  }
  // Do the selected problem
  switch (
      problem) { // if the problem fails to make a file print Error statement
    case 1:
      (problem_1() == 0) ? printf("Problem 1 success\n")
                         : printf("Error in Problem 1\n");
      break;
    case 2:
      (problem_2() == 0) ? printf("Problem 2 success\n")
                         : printf("Error in Problem 2\n");
      break;
    case 3:
      (problem_3() == 0) ? printf("Problem 3 success\n")
                         : printf("Error in Problem 3\n");
      break;
    default:
      printf("Error please choose 1-3.\n");
  }
  // End Program
  return EXIT_SUCCESS;
}

// Problem 1 Code
int problem_1(void) {
  FILE* fout = fopen("prog_sol_1.txt", "w");
  if (fout == NULL) {
    perror("output file failed");
    return EXIT_FAILURE;
  }
  // Declare and Initialize variables
  double y       = 3.0;
  double delta_t = 0.001;
  double time    = 0;
  double a       = -2.5;
  // This line doesn't change in the loop
  a = (1 + a * delta_t);
  // Do the math iteratively in a loop
  for (time = 0.0; time < 10.0; time += delta_t) {
    // Print the t, y(t) coordinate
    fprintf(fout, "%0.3lf\t%0.10lf\n", time, y);
    // Do the calculation
    y = a * y;
  }
  return EXIT_SUCCESS;
}

// Problem 2 Code
int problem_2(void) {
  FILE* fout = fopen("prog_sol_2.txt", "w");
  if (fout == NULL) {
    perror("output file failed");
    return EXIT_FAILURE;
  }
  double I[][3]  = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};
  double A[][3]  = {{0, 1, 0}, {0, 0, 1}, {-2.5063, -25.1125, -0.6}};
  double x_t[3]  = {1.5, 2, -1};
  double delta_t = 0.001f;
  double time;
  // avoid unecessary function calls in loop
  mat_scale(delta_t, A, A);
  mat_add(I, A, A);
  // Do iterative math
  for (time = 0.0; time < 10.0; time += delta_t) {
    // print the result
    fprintf(fout, "%.3lf\t%.10lf\n", time, x_t[0]);
    // get next x_t value
    mat_vec_mult(A, x_t, x_t);
  }
  return EXIT_SUCCESS;
}

// Problem 3 Code
int problem_3(void) {
  FILE* fout = fopen("prog_sol_3.txt", "w");
  if (fout == NULL) {
    perror("output file failed");
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}

// scale matrix `mat` by a and save in prod
void mat_scale(double scale, double mat[][3], double prod[][3]) {
  int i, j;
  for (i = 0; i < 3; ++i) {
    for (j = 0; j < 3; ++j) {
      prod[i][j] = scale * mat[i][j];
    }
  }
}

// subtract the right from the left and store in diff
void mat_sub(double left[][3], double right[][3], double diff[][3]) {
  int i, j;
  for (i = 0; i < 3; i++) {
    for (j = 0; j < 3; j++) {
      diff[i][j] = left[i][j] - right[i][j];
    }
  }
}

// add the right and the left and store into sum
void mat_add(double left[][3], double right[][3], double sum[][3]) {
  int i, j;
  for (i = 0; i < 3; i++) {
    for (j = 0; j < 3; j++) {
      sum[i][j] = left[i][j] + right[i][j];
    }
  }
}

// Unsafe if used incorrectly!
// multiplies 3x3 matrix mat with 3x1 vector storing into 3x1 prod
void mat_vec_mult(double mat[][3], double* vector, double* prod) {
  double sum;
  int    i, j;
  for (i = 0; i < 3; i++) {
    sum = 0;
    for (j = 0; j < 3; j++) {
      sum += mat[i][j] * vector[j];
    }
    prod[i] = sum;
  }
}
```