Program 2 Report

Kelson Petersen

November 14, 2025

Objectives

To review programming in C or C++. To give practice in converting mathematical descriptions of ideaas into computer implementations. To provide practice and experience in convolution. To provide practice in finding the complete solution of differential equations.

Discrete Convolution in C

To perform convolution in a digital computer it is easier to make a continuous signal discrete. We do this by 'sampling' the continuous signal at specific points in time. We then are able to convolve that signal with other discrete signals. The method to convolve two discrete functions is create an array that holds $n \cdot m - 1$ elements where n is the length of signal one, and m is the length of signal two. The convolution is the sum of all diagonal elements in a table of products as seen in the scratch work section of the appendix.

To write this code it is fairly simple and only requires the following code.

```
1  #include "convolve.h"
2  #include <stdlib.h>
3  #include <stdio.h>
4  
5  int conv(double *f1, int len1, double *f2, int len2, double **y){
6    int leny = len1 + len2 - 1;
7    int in_b, m_start;
8    // Allocate The proper amount of memory for y
9    (*y) = (double *)malloc((sizeof(double) * leny));
10   for(int itr = 0; itr < leny; itr++){
11     (*y)[itr] = 0.0f;
12   }
13   // Add the diagonals
14   for(int i = 0; i < len1; i++){
15     for(int j = 0; j < len2; j++){
16       (*y)[i+j] += f1[i] * f2[j];
17     }
18   }
19   return leny;
20 }
```

The double nested loop in lines 14–18 iterate through each element of the two arrays passed to the function. While iterating throught the arrays the index $i + j$ is how we get the diagonal elements to sum together in the correct places.

The results for convolutions $f1 * f1, f1 * f2, f1 * f3, f2 * f3, f1 * f4$ are given below in the figures section. The program included in the appendix prints to the terminal the following.

```
f1:       0      1      2      3      2      1
f2:      -2     -2     -2     -2     -2     -2     -2
f3:       1     -1      1     -1
f4:       0      0      0     -3     -3

f1 * f1:       0      0      1      4     10     16     19     16     10      4      1
f1 * f2:       0     -2     -6    -12    -16    -18    -18    -18    -16    -12     -6     -2
f1 * f3:       0      1      1      2      0      0     -2     -1     -1
f2 * f3:      -2      0     -2      0      0      0      0      2      0      2
f1 * f4:       0      0      0      0     -3     -9    -15    -15     -9     -3%
```

Total Solution

Given the equation

$$(D^3 + 5D^2 + 12D + 15)y(t) = (D + 1.5)f(t) \tag{1}$$

With initial conditions $y(0) = -2, \dot{y}(0) = 3, \ddot{y}(0) = 4$ we find with the scratch work done in the appendix that the solution to the above equation is the following:

$$y_{\ total} = \left(-0.07e^{-2.6038t} + 1.94e^{-1.19t}\cos(2.08 - 2.98) + 0.0164\cos(7.85t + 2.03)\right)u(t) \tag{2}$$

With the above solution we are able to use the included code to get the graphs in the figures section below.

Figures

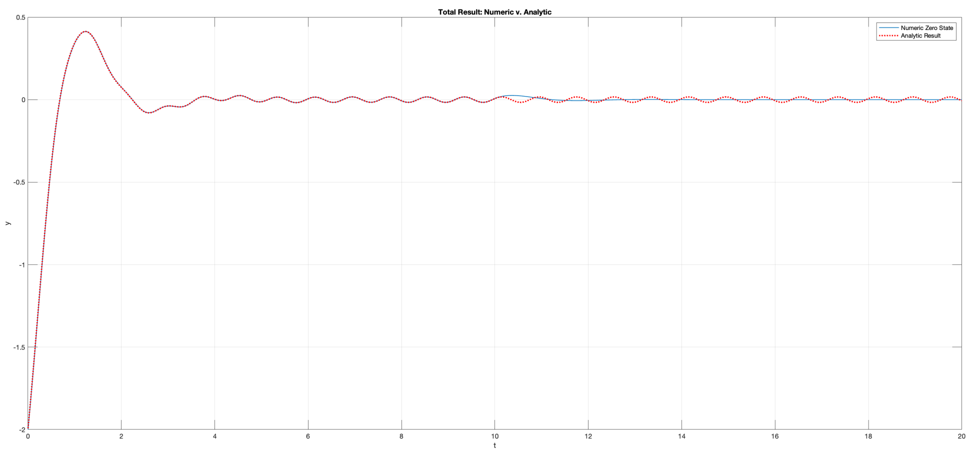

Figure 1: Program Numeric Solution vs. Analytical Solution

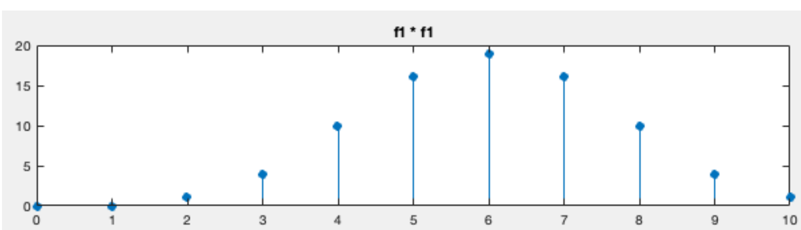

Figure 2: Convolution of f1 and f1

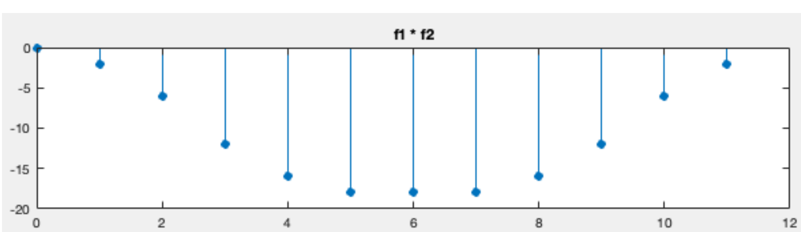

Figure 3: Convolution of f1 and f2

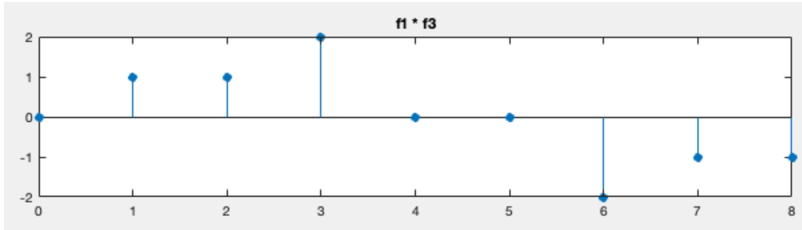

Figure 4: Convolution of f1 and f3

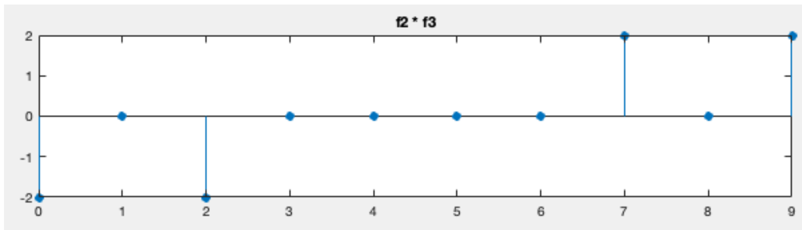

Figure 5: Convolution of f2 and f3

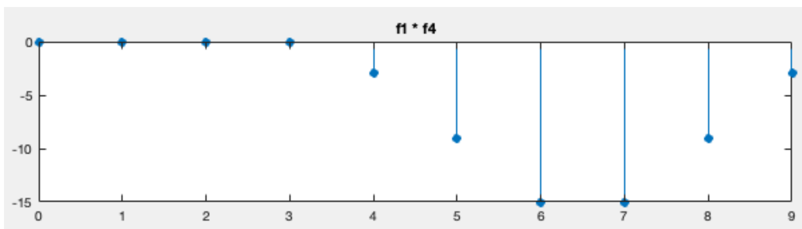

Figure 6: Convolution of f1 and f4

Conclusion

The final result ends up settling to a "steady state" oscillation with a magnitude of 0.0164. This magnitude is the result of the total solution having one part multiplied by a decaying exponential and the other part does not. The part without the decaying exponential is the resulting steady state output of the solution.

Appendix

In this section we have the code and the analytical solutions done by hand.

Code

```
data = load('./total_result.txt');
x = data(:,1);
y = data(:,2);


figure;
plot(x,y,'LineWidth',1,'DisplayName','Numeric Zero State')
hold on;
grid on;

t_fine = 0:0.001:20;

y_analytic = (-0.07 .* exp(-2.6038 .* t_fine) + 1.94 .* exp(-1.198 .* t_fine) .* cos(2.08 .* t_fine - 2.98) + 0.01

y_analytic = real(y_analytic);

plot(t_fine, y_analytic, 'r:','Linewidth',2,'DisplayName', 'Analytic Result')

legend('show');
xlabel('t');
ylabel('y');
title('Total Result: Numeric v. Analytic');
```

```
f1 = [0,1,2,3,2,1];
f2 = [-2,-2,-2,-2,-2,-2,-2];
f3 = [1,-1,1,-1];
f4 = [0,0,0,-3,-3];

% Compute convolutions
c1 = conv(f1, f1);
c2 = conv(f1, f2);
c3 = conv(f1, f3);
c4 = conv(f2, f3);
c5 = conv(f1, f4);

figure;

subplot(5,1,1);
stem(0:length(c1)-1, c1, 'filled');
title('f1 * f1');

subplot(5,1,2);
stem(0:length(c2)-1, c2, 'filled');
title('f1 * f2');

subplot(5,1,3);
stem(0:length(c3)-1, c3, 'filled');
title('f1 * f3');

subplot(5,1,4);
stem(0:length(c4)-1, c4, 'filled');
title('f2 * f3');

subplot(5,1,5);
stem(0:length(c5)-1, c5, 'filled');
title('f1 * f4');
```

```
#include <stdio.h>
#include <stdlib.h>
#include "convolve.h"
#include "matrix.h"

void Ft_conv_Ht(void);
int  program_1_code(void);

int main(void) {
```

```c
  int           itr, leny;
  double*       y;
  const int     len1 = 6, len2 = 7, len3 = 4, len4 = 5;
  const double f1[len1] = {0, 1, 2, 3, 2, 1};
  const double f2[len2] = {-2, -2, -2, -2, -2, -2, -2};
  const double f3[len3] = {1, -1, 1, -1};
  const double f4[len4] = {0, 0, 0, -3, -3};
  const double X[3]      = {0.4, 0.35, 0.25};
  const double Y[4]      = {0.25, 0.20, 0.20, 0.35};

  printf("f1: ");
  for (itr = 0; itr < len1; itr++) { printf(" %5.0lf", f1[itr]); }
  printf("\nf2: ");
  for (itr = 0; itr < len2; itr++) { printf(" %5.0lf", f2[itr]); }
  printf("\nf3: ");
  for (itr = 0; itr < len3; itr++) { printf(" %5.0lf", f3[itr]); }
  printf("\nf4: ");
  for (itr = 0; itr < len4; itr++) { printf(" %5.0lf", f4[itr]); }
  printf("\n\n");
  /*  Problem 2.A   */
  leny = conv(f1, len1, f1, len1, &y);
  printf("f1 * f1: ");
  for (itr = 0; itr < leny; ++itr) { printf(" %5.0lf", y[itr]); }
  free(y);
  /*   Problem 2.B  */
  printf("\nf1 * f2: ");
  leny = conv(f1, len1, f2, len2, &y);
  for (itr = 0; itr < leny; ++itr) { printf(" %5.0lf", y[itr]); }
  free(y);
  /* Problem 2.C */
  printf("\nf1 * f3: ");
  leny = conv(f1, len1, f3, len3, &y);
  for (itr = 0; itr < leny; ++itr) { printf(" %5.0lf", y[itr]); }
  free(y);
  /* Problem 2.D */
  printf("\nf2 * f3: ");
  leny = conv(f2, len2, f3, len3, &y);
  for (itr = 0; itr < leny; ++itr) { printf(" %5.0lf", y[itr]); }
  free(y);
  /* Problem 2.E */
  printf("\nf1 * f4: ");
  leny = conv(f1, len1, f4, len4, &y);
  for (itr = 0; itr < leny; ++itr) { printf(" %5.0lf", y[itr]); }
  free(y);
  // Get results for the zero input
  program_1_code();
  // Perform the convolution and summing of h_t and zero_input
  Ft_conv_Ht();
  return 0;
}

int program_1_code(void) {
  FILE* fout = fopen("zero_input.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}, {-15, -12, -5}};
  double x_t[3]  = {-2, 3, 4};
  double delta_t = 0.001f;
  double time;
  // avoid unnecessary 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, "%.10lf\n", x_t[0]);
    // get next x_t value
    mat_vec_mult(A, x_t, x_t);
  }
  fclose(fout);
  return EXIT_SUCCESS;
```

```
}

void Ft_conv_Ht(void) {
  double ft[10000] = {0};
  double ht[10000] = {0};
  double zi[10000] = {0};
  char   buff[100];
  int     leny = 0;
  double* y;
  FILE*   FT_FILE = fopen("sine.txt", "r");
  if (FT_FILE == NULL) {
    printf("Error: sine.txt doesn't exist\n");
    exit(1);
  }
  FILE* HT_FILE = fopen("h_t.txt", "r");
  if (HT_FILE == NULL) {
    printf("Error: h_t.txt doesn't exist\n");
    exit(1);
  }
  FILE* ZI_FILE = fopen("zero_input.txt", "r");
  if (ZI_FILE == NULL) {
    printf("error: zero_input.txt didn't open\n");
    exit(1);
  }
  FILE* TOTAL_RESPONSE = fopen("total_result.txt", "w");
  if (TOTAL_RESPONSE == NULL) {
    printf("total_result.txt not opened\n");
    exit(1);
  }
  for (int i = 0; i < 10000; i++) {
    if (!fgets(buff, sizeof(buff), FT_FILE)) { printf("Reached end of file at line %d\n", i); break; }
    char*  endptr;
    double value = strtod(buff, &endptr);
    if (endptr == buff) { printf("Invalid number on line %d: %s\n", i, buff); continue; }
    ft[i] = value;
  }
  fclose(FT_FILE);
  for (int i = 0; i < 10000; i++) {
    if (!fgets(buff, sizeof(buff), HT_FILE)) { printf("Reached end of file at line %d\n", i); break; }
    char*  endptr;
    double value = strtod(buff, &endptr);
    if (endptr == buff) { printf("Invalid number on line %d: %s\n", i, buff); continue; }
    ht[i] = value;
  }
  fclose(HT_FILE);
  // Perform the convolution
  leny = conv(ht, sizeof(ht) / sizeof(ht[0]), ft, sizeof(ft) / sizeof(ft[0]), &y);
  // Scale the convolution by 0.001
  for(int itr = 0; itr < leny; itr++){
    y[itr] *= 0.001;
  }
  for (int i = 0; i < 10000; i++) {
    if (!fgets(buff, sizeof(buff), ZI_FILE)) {printf("Reached end of file at line %d\n", i); break; }
    char*  endptr;
    double value = strtod(buff, &endptr);
    if (endptr == buff) { printf("Invalid number on line %d: %s\n", i, buff); continue; }
    zi[i] = value;
  }
  fclose(ZI_FILE);
  double temp = 0;
  for (int itr = 0; itr < 20000; itr++) {
    if (itr < 10000) {
      temp = zi[itr];
    } else {
      temp = 0;
    }
    fprintf(TOTAL_RESPONSE, "%lf\t%.10lf\n", (double) itr / 1000, y[itr] + temp);
  }
  free(y);
}
```

```
#ifndef __MATRIX_H_
#define __MATRIX_H_
```

```c
// 2x2 Matrix and Vector Functions
void mat_scale2(double scale, double mat[][2], double prod[][2]);
void mat_sub2(double left[][2], double right[][2], double diff[][2]);
void mat_add2(double left[][2], double right[][2], double sum[][2]);
void mat_vec_mult2(double mat[][2], double* vector, double* prod);
// 3x3 Matrix and Vector Functions
void mat_scale(double scale, double mat[][3], double prod[][3]);
void mat_sub(double left[][3], double right[][3], double diff[][3]);
void mat_add(double left[][3], double right[][3], double sum[][3]);
void mat_vec_mult(double mat[][3], double* vector, double* prod);
#endif
```

```c
#include "matrix.h"

// 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;
  }
}

// scale matrix `mat` by a and save in prod
void mat_scale2(double scale, double mat[][2], double prod[][2]) {
  int i, j;
  for (i = 0; i < 2; ++i) {
    for (j = 0; j < 2; ++j) {
      prod[i][j] = scale * mat[i][j];
    }
  }
}
// subtract the right from the left and store in diff
void mat_sub2(double left[][2], double right[][2], double diff[][2]) {
  int i, j;
  for (i = 0; i < 3; i++) {
    for (j = 0; j < 3; j++) {
      diff[i][j] = left[i][j] - right[i][j];
    }
  }
}
```

```c
// add the right and the left and store into sum
void mat_add2(double left[][2], double right[][2], double sum[][2]) {
  int i, j;
  for (i = 0; i < 2; i++) {
    for (j = 0; j < 2; 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_mult2(double mat[][2], double* vector, double* prod) {
  double sum;
  int    i, j;
  for (i = 0; i < 2; i++) {
    sum = 0;
    for (j = 0; j < 2; j++) {
      sum += mat[i][j] * vector[j];
    }
    prod[i] = sum;
  }
}
```

```c
#ifndef __CONVOLVE_H
#define __CONVOLVE_H

int conv(double *f1, int len1, double *f2, int len2, double **y);

#endif
```

```c
#include "convolve.h"
#include <stdlib.h>
#include <stdio.h>

int conv(double *f1, int len1, double *f2, int len2, double **y){
  int leny = len1 + len2 - 1;
  int in_b, m_start;
  // Allocate The proper amount of memory for y
  (*y) = (double *)malloc((sizeof(double) * leny));
  for(int itr = 0; itr < leny; itr++){
    (*y)[itr] = 0.0f;
  }
  // Add the diagonals
  for(int i = 0; i < len1; i++){
    for(int j = 0; j < len2; j++){
      (*y)[i+j] += f1[i] * f2[j];
    }
  }
  return leny;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <complex.h>

int main(void){
  double stop = 10.0,d_T = 0.001, c_dt;
  FILE *fout = fopen("sine.txt","w");
  for(c_dt =0; c_dt < stop; c_dt += d_T){
    fprintf(fout,"%lf\n",sin(2.5 * M_PI * c_dt));
  }
  fclose(fout);
  return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <complex.h>

```

```
int main(void){
  double stop = 10.0,d_T = 0.001, c_dt;
  double result = 0;
  FILE *fout = fopen("h_t.txt","w");
  for(c_dt =0; c_dt < stop; c_dt += d_T){
    result = -0.175 * exp(-2.604*c_dt) + 2 * (0.201*exp(-1.198*c_dt)*cos(-2.08*c_dt + 1.12));
    fprintf(fout,"%lf\n",result);
  }
  fclose(fout);
  return 0;
}
```

Scratch Work

This section contains the hand analysis. The following pages contain the hand work.

2.A.) f1 * f1 = 0, 0, 1, 4, 10, 16, 19, 16, 10, 4, 1

	0	1	2	3	2	1
0	0	0	0	0	0	0
1	0	1	2	3	2	1
2	0	2	4	6	4	2
3	0	3	6	9	6	3
2	0	2	4	6	4	2
1	0	1	2	3	2	1

m	y[m]	
0	0	= 0
1	0 + 0	= 0
2	0 + 1 + 0	= 1
3	0 + 2 + 2 + 0	= 4
4	0 + 3 + 4 + 3 + 0	= 10
5	0 + 2 + 6 + 6 + 2 + 0	= 16
6	1 + 4 + 9 + 4 + 1	= 19
7	2 + 6 + 6 + 2	= 16
8	3 + 4 + 3	= 10
9	2 + 2	= 4
10	1	= 1

2.B.) f1 * f2 = 0, -2, -6, -12, -16, -18, -18, -18, -16, -12, -6, -2

f1

f2	0	1	2	3	2	1
-2	0	-2	-4	-6	-4	-2
-2	0	-2	-4	-6	-4	-2
-2	0	-2	-4	-6	-4	-2
-2	0	-2	-4	-6	-4	-2
-2	0	-2	-4	-6	-4	-2
-2	0	-2	-4	-6	-4	-2
-2	0	-2	-4	-6	-4	-2

m	y[m]	
0	0	= 0
1	0 - 2	= -2
2	0 - 2 - 4	= -6
3	0 - 2 - 4 - 6	= -12
4	0 - 2 - 4 - 6 - 4	= -16
5	0 - 2 - 4 - 6 - 4 - 2	= -18
6	0 - 2 - 4 - 6 - 4 - 2	= -18
7	-2 - 4 - 6 - 4 - 2	= -18
8	-4 - 6 - 4 - 2	= -16
9	-6 - 4 - 2	= -12
10	-4 - 2	= -6
11	-2	= -2

2.C.) f1 * f3 = 0, 1, 1, 2, 0, 0, -2, -1, -1

	0	1	2	3	2	1
1	0	1	2	3	2	1
-1	0	-1	-2	-3	-2	-1
1	0	1	2	3	2	1
-1	0	-1	-2	-3	-2	-1

k	y[k]	
0	0	= 0
1	0 + 1	= 1
2	0 - 1 + 2	= 1
3	0 + 1 - 2 + 3	= 2
4	-1 + 2 - 3 + 2	= 0
5	-2 + 3 - 2 + 1	= 0
6	-3 + 2 - 1	= -2
7	-2 + 1	= -1
8	-1	= -1

2.D.) $f2 * f3 = -2, 0, -2, 0, 0, 0, 0, 2, 0, 2$

	-2	-2	-2	-2	-2	-2	-2
1	-2	-2	-2	-2	-2	-2	-2
-1	2	2	2	2	2	2	2
1	-2	-2	-2	-2	-2	-2	-2
-1	2	2	2	2	2	2	2

k	y[k]	
0	-2	= -2
1	2 - 2	= 0
2	-2 + 2 - 2	= -2
3	2 - 2 + 2 - 2	= 0
4	2 - 2 + 2 - 2	= 0
5	2 - 2 + 2 - 2	= 0
6	2 - 2 + 2 - 2	= 0
7	2 - 2 + 2	= 2
8	2 - 2	= 0
9	2	= 2

2.E.) $f1 * f4 = 0, 0, 0, 0, -3, -9, -15, -15, -9, -3$

	0	1	2	3	2	1
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
-3	0	-3	-6	-9	-6	-3
-3	0	-3	-6	-9	-6	-3

k	y[k]	
0	0	= 0
1	0 + 0	= 0
2	0 + 0 + 0	= 0
3	0 + 0 + 0 + 0	= 0
4	0 - 3 + 0 + 0 + 0	= -3
5	-3 - 6 + 0 + 0 + 0	= -9
6	-6 - 9 + 0 + 0	= -15
7	-9 - 6 + 0	= -15
8	-6 - 3	= -9
9	-3	= -3

$$(D^3 + 5D^2 + 12D + 15)\,y(t) = (D + 1.5)\,f(t)$$

$$H(s) = \frac{y(t)}{f(t)} = \frac{D+1.5}{D^3+5D^2+12D+15}$$

$$\mathcal{L}^{-1}\left[H(s)\right] = h(t)$$

$$h(t) = -0.175e^{-2.604t} + 2\left[0.201e^{-1.198t}\cos(-2.08t + 1.12)\right]$$

$$\text{zero state} = h(t) * \sin(2.5\pi t) = \frac{s+1.5}{s^3+5s^2+12s+15} \cdot \frac{2.5\pi}{s^2+(2.5\pi)^2}$$

$$= \mathcal{L}^{-1}\left[\frac{s+1.5}{s^3+5s^2+12s+15} \cdot \frac{2.5\pi}{s^2+(2.5\pi)^2}\right] = \mathcal{L}^{-1}\left[\frac{7.85s + 11.78}{s^5+5s^4+73.69s^3+323.45s^2+740.28s+925.35}\right]$$

$$\text{Zero state} = -0.02e^{-2.604t} + 2\left[0.027e^{-1.19t}\cos(-2.08t - 1.036)\right] + 2\left[0.008\cos(7.85t + 2.03)\right]$$

$$\text{zero input} = 0.05e^{-2.604t} + 2\left[0.98e^{-1.198t}\cos(2.08t - 3.01)\right]$$

$$\text{total response} = \text{zero state} + \text{zero input}$$

$$= -0.02e^{-2.604t} + 2\left[0.027e^{-1.19t}\cos(-2.08t - 1.036)\right] + 2\left[0.008\cos(7.85t + 2.03)\right] \cdots$$

$$+ 0.05e^{-2.604t} + 2\left[0.98e^{-1.198t}\cos(2.08t - 3.01)\right]$$