

Program 2 Report

Kelson Petersen

November 14, 2025

Objectives

To review programming in C or C++. To give practice in converting mathematical descriptions of ideas 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 through the arrays the index $i + j$ is how we get the diagonal elements to sum together in the correct places.

Conclusion

Words

Appendix

Words

Code

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "convolve.h"
4 #include "matrix.h"
5
6 void Ft_conv_Ht(void);
7 int program_1_code(void);
8
9 int main(void) {
10     int         itr, leny;
11     double*      y;
12     const int    len1 = 6, len2 = 7, len3 = 4, len4 = 5;
13     const double f1[len1] = {0, 1, 2, 3, 2, 1};
14     const double f2[len2] = {-2, -2, -2, -2, -2, -2, -2};
15     const double f3[len3] = {1, -1, 1, -1};
16     const double f4[len4] = {0, 0, 0, -3, -3};
17     const double X[3]     = {0.4, 0.35, 0.25};
18     const double Y[4]     = {0.25, 0.20, 0.20, 0.35};
19
20     printf("f1: ");
21     for (itr = 0; itr < len1; itr++) { printf(" %5.0lf", f1[itr]); }
22     printf("\nf2: ");
23     for (itr = 0; itr < len2; itr++) { printf(" %5.0lf", f2[itr]); }
24     printf("\nf3: ");
25     for (itr = 0; itr < len3; itr++) { printf(" %5.0lf", f3[itr]); }
26     printf("\nf4: ");
27     for (itr = 0; itr < len4; itr++) { printf(" %5.0lf", f4[itr]); }
28     printf("\n\n");
29     /* Problem 2.A */
30     leny = conv(f1, len1, f1, len1, &y);
31     printf("f1 * f1: ");
32     for (itr = 0; itr < leny; ++itr) { printf(" %5.0lf", y[itr]); }
33     free(y);
34     /* Problem 2.B */
35     printf("\nf1 * f2: ");
36     leny = conv(f1, len1, f2, len2, &y);
37     for (itr = 0; itr < leny; ++itr) { printf(" %5.0lf", y[itr]); }
38     free(y);
39     /* Problem 2.C */
40     printf("\nf1 * f3: ");
41     leny = conv(f1, len1, f3, len3, &y);
42     for (itr = 0; itr < leny; ++itr) { printf(" %5.0lf", y[itr]); }
43     free(y);
44     /* Problem 2.D */
45     printf("\nf2 * f3: ");
46     leny = conv(f2, len2, f3, len3, &y);
47     for (itr = 0; itr < leny; ++itr) { printf(" %5.0lf", y[itr]); }
48     free(y);
49     /* Problem 2.E */
50     printf("\nf1 * f4: ");
51     leny = conv(f1, len1, f4, len4, &y);
52     for (itr = 0; itr < leny; ++itr) { printf(" %5.0lf", y[itr]); }
53     free(y);
54     // Get results for the zero input
55     program_1_code();
56     // Perform the convolution and summing of zero_state and zero_input
57     Ft_conv_Ht();
58     return 0;
59 }
60
61 int program_1_code(void) {
62     FILE* fout = fopen("zero_input.txt", "w");
63     if (fout == NULL) {
64         perror("output file failed");
65         return EXIT_FAILURE;
66     }
67     double I[][3] = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};
```

```

68     double A[][3] = {{0, 1, 0}, {0, 0, 1}, {-15, -12, -5}};
69     double x_t[3] = {-2, 3, 4};
70     double delta_t = 0.0005f;
71     double time;
72     // avoid unnecessary function calls in loop
73     mat_scale(delta_t, A, A);
74     mat_add(I, A, A);
75     // Do iterative math
76     for (time = 0.0; time < 10.0; time += delta_t) {
77         // print the result
78         fprintf(fout, "%.10lf\n", x_t[0]);
79         // get next x_t value
80         mat_vec_mult(A, x_t, x_t);
81     }
82     fclose(fout);
83     return EXIT_SUCCESS;
84 }

85 void Ft_conv_Ht(void) {
86     double ft[10000] = {0};
87     double ht[10000] = {0};
88     double zi[10000] = {0};
89     char buff[100];
90     int leny = 0;
91     double* y;
92     FILE* FT_FILE = fopen("sine.txt", "r");
93     if (FT_FILE == NULL) {
94         printf("Error: sine.txt doesn't exist\n");
95         exit(1);
96     }
97     FILE* HT_FILE = fopen("zero_state.txt", "r");
98     if (HT_FILE == NULL) {
99         printf("Error: zero_state.txt doesn't exist\n");
100        exit(1);
101    }
102    FILE* ZI_FILE = fopen("zero_input.txt", "r");
103    if (ZI_FILE == NULL) {
104        printf("error: zero_input.txt didn't open\n");
105        exit(1);
106    }
107    FILE* TOTAL_RESPONSE = fopen("total_result.txt", "w");
108    if (TOTAL_RESPONSE == NULL) {
109        printf("total_result.txt not opened\n");
110        exit(1);
111    }
112    for (int i = 0; i < 10000; i++) {
113        if (!fgets(buff, sizeof(buff), FT_FILE)) {
114            printf("Reached end of file at line %d\n", i);
115            break;
116        }
117        char* endptr;
118        double value = strtod(buff, &endptr);
119        if (endptr == buff) {
120            printf("Invalid number on line %d: %s\n", i, buff);
121            continue;
122        }
123        ft[i] = value;
124    }
125    fclose(FT_FILE);
126    for (int i = 0; i < 10000; i++) {
127        if (!fgets(buff, sizeof(buff), HT_FILE)) {
128            // Stop if file ends early
129            printf("Reached end of file at line %d\n", i);
130            break;
131        }
132        char* endptr;
133        double value = strtod(buff, &endptr);
134        if (endptr == buff) {
135            // No valid number on this line
136            printf("Invalid number on line %d: %s\n", i, buff);
137            continue;
138        }
139        ht[i] = value;
140        // printf("%lf\n", ht[i]);
141    }

```

```

142     }
143     fclose(ht_file);
144     // Perform the convolution
145     leny = conv(ht, sizeof(ht) / sizeof(ht[0]), ft, sizeof(ft) / sizeof(ft[0]), &y);
146     // Scale the convolution by 0.001
147     for(int itr = 0; itr < leny; itr++){
148         y[itr] *= 0.001;
149     }
150     for (int i = 0; i < 10000; i++) {
151         if (!fgets(buff, sizeof(buff), zi_file)) {
152             // Stop if file ends early
153             printf("Reached end of file at line %d\n", i);
154             break;
155         }
156         char* endptr;
157         double value = strtod(buff, &endptr);
158         if (endptr == buff) {
159             // No valid number on this line
160             printf("Invalid number on line %d: %s\n", i, buff);
161             continue;
162         }
163         zi[i] = value;
164     }
165     fclose(zi_file);
166     double temp = 0;
167     for (int itr = 0; itr < 20000; itr++) {
168         if (itr < 10000) {
169             temp = zi(itr);
170         } else {
171             temp = 0;
172         }
173         fprintf(TOTAL_RESPONSE, "%lf\t%.10lf\n", (double) itr / 1000, y[itr] + temp);
174     }
175     free(y);
176 }
```

```

1 #ifndef __CONVOLVE_H
2 #define __CONVOLVE_H
3
4 int conv(double *f1, int len1, double *f2, int len2, double **y);
5
6 #endif
```

```

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 }
```

```

1 #ifndef __MATRIX_H_
2 #define __MATRIX_H_
3 // 2x2 Matrix and Vector Functions
4 void mat_scale2(double scale, double mat[][2], double prod[][2]);
5 void mat_sub2(double left[][2], double right[][2], double diff[][2]);
6 void mat_add2(double left[][2], double right[][2], double sum[][2]);
7 void mat_vec_mult2(double mat[][2], double* vector, double* prod);
8 // 3x3 Matrix and Vector Functions
```

```

9 void mat_scale(double scale, double mat[][3], double prod[][3]);
10 void mat_sub(double left[][3], double right[][3], double diff[][3]);
11 void mat_add(double left[][3], double right[][3], double sum[][3]);
12 void mat_vec_mult(double mat[][3], double* vector, double* prod);
13 #endif

```

```

1 #include "matrix.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 // subtract the right from the left and store in diff
13 void mat_sub(double left[][3], double right[][3], double diff[][3]) {
14     int i, j;
15     for (i = 0; i < 3; i++) {
16         for (j = 0; j < 3; j++) {
17             diff[i][j] = left[i][j] - right[i][j];
18         }
19    }
20 }
21 // add the right and the left and store into sum
22 void mat_add(double left[][3], double right[][3], double sum[][3]) {
23     int i, j;
24     for (i = 0; i < 3; i++) {
25         for (j = 0; j < 3; j++) {
26             sum[i][j] = left[i][j] + right[i][j];
27         }
28    }
29 }
30 // Unsafe if used incorrectly!
31 // multiplies 3x3 matrix mat with 3x1 vector storing into 3x1 prod
32 void mat_vec_mult(double mat[][3], double* vector, double* prod) {
33     double sum;
34     int i, j;
35     for (i = 0; i < 3; i++) {
36         sum = 0;
37         for (j = 0; j < 3; j++) {
38             sum += mat[i][j] * vector[j];
39         }
40         prod[i] = sum;
41     }
42 }
43
44 // scale matrix 'mat' by a and save in prod
45 void mat_scale2(double scale, double mat[][2], double prod[][2]) {
46     int i, j;
47     for (i = 0; i < 2; ++i) {
48         for (j = 0; j < 2; ++j) {
49             prod[i][j] = scale * mat[i][j];
50         }
51    }
52 }
53 // subtract the right from the left and store in diff
54 void mat_sub2(double left[][2], double right[][2], double diff[][2]) {
55     int i, j;
56     for (i = 0; i < 3; i++) {
57         for (j = 0; j < 3; j++) {
58             diff[i][j] = left[i][j] - right[i][j];
59         }
60    }
61 }
62 // add the right and the left and store into sum
63 void mat_add2(double left[][2], double right[][2], double sum[][2]) {
64     int i, j;
65     for (i = 0; i < 2; i++) {
66         for (j = 0; j < 2; j++) {
67             sum[i][j] = left[i][j] + right[i][j];
68         }
69    }
70 }

```

```

68     }
69 }
70 // Unsafe if used incorrectly!
71 // multiplies 3x3 matrix mat with 3x1 vector storing into 3x1 prod
72 void mat_vec_mult2(double mat[][2], double* vector, double* prod) {
73     double sum;
74     int i, j;
75     for (i = 0; i < 2; i++) {
76         sum = 0;
77         for (j = 0; j < 2; j++) {
78             sum += mat[i][j] * vector[j];
79         }
80         prod[i] = sum;
81     }
82 }
83

```

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4 #include <complex.h>
5
6 int main(void){
7     double stop = 10.0,d_T = 0.001, c_dt;
8     FILE *fout = fopen("sine.txt","w");
9     for(c_dt =0; c_dt < stop; c_dt += d_T){
10         fprintf(fout,"%lf\n",sin(2.5 * M_PI * c_dt));
11     }
12     fclose(fout);
13     return 0;
14 }

```

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4 #include <complex.h>
5
6 int main(void){
7     double stop = 10.0,d_T = 0.001, c_dt;
8     double result = 0;
9     FILE *fout = fopen("zero_state.txt","w");
10    for(c_dt =0; c_dt < stop; c_dt += d_T){
11        result = -0.175 * exp(-2.604*c_dt) + 2 * (0.201*exp(-1.198*c_dt)*cos(2.08*c_dt + 1.12));
12        fprintf(fout,"%lf\n",result);
13    }
14    fclose(fout);
15    return 0;
16 }

```

```

1 data = load('./total_result.txt');
2 x = data(:,1);
3 y = data(:,2);
4
5
6 figure;
7 plot(x,y,'LineWidth',1,'DisplayName','Numeric Zero State')
8 hold on;
9 grid on;
10
11 t_fine = 0:0.001:20;
12
13 y_analytic = (-0.07 .* exp(-2.6038 .* t_fine) + 1.94 .* exp(-1.198 .* t_fine) .* cos(2.08 .* t_fine - 2.98)) + 0.0;
14
15 y_analytic = real(y_analytic);
16
17 plot(t_fine, y_analytic, 'r-','Linewidth',2,'DisplayName', 'Analytic Result')
18
19 legend('show');
20 xlabel('t');
21 ylabel('y');
22 title('Total Result: Numeric v. Analytic');

```

```

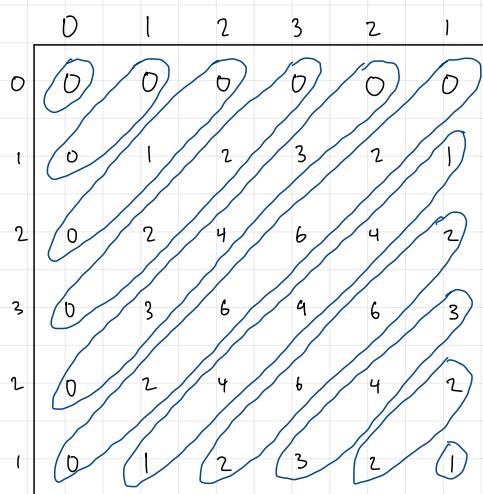
1 f1 = [0,1,2,3,2,1];
2 f2 = [-2,-2,-2,-2,-2,-2];
3 f3 = [1,-1,1,-1];
4 f4 = [0,0,0,-3,-3];
5
6 % Compute convolutions
7 c1 = conv(f1, f1);
8 c2 = conv(f1, f2);
9 c3 = conv(f1, f3);
10 c4 = conv(f2, f3);
11 c5 = conv(f1, f4);
12
13 figure;
14
15 subplot(5,1,1);
16 stem(0:length(c1)-1, c1, 'filled');
17 title('f1 * f1');
18
19 subplot(5,1,2);
20 stem(0:length(c2)-1, c2, 'filled');
21 title('f1 * f2');
22
23 subplot(5,1,3);
24 stem(0:length(c3)-1, c3, 'filled');
25 title('f1 * f3');
26
27 subplot(5,1,4);
28 stem(0:length(c4)-1, c4, 'filled');
29 title('f2 * f3');
30
31 subplot(5,1,5);
32 stem(0:length(c5)-1, c5, 'filled');
33 title('f1 * f4');

```

Scratch Work

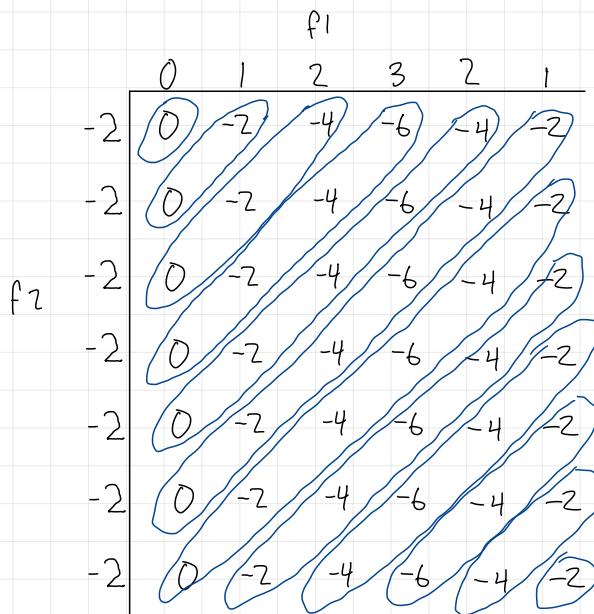
This section contains the hand analysis. The following pages contain the hand work.

$$2.A.) f_1 * f_1 = 0, 0, 1, 4, 10, 16, 19, 16, 10, 4, 1$$



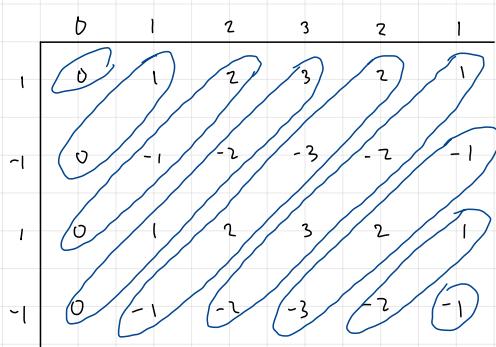
m	$y[m]$	
0	0	= 0
1	$0+0$	= 0
2	$0+1+0$	= 1
3	$0+2+2+0$	= 4
4	$0+3+4+3+0$	= 10
5	$0+2+6+6+2+0$	= 16
6	$1+4+9+4+1$	= 19
7	$2+6+6+2$	= 16
8	$3+4+3$	= 10
9	$2+2$	= 4
10	1	= 1

$$2.B.) f_1 * f_2 = 0, -2, -6, -12, -16, -18, -18, -18, -16, -12, -6, -2$$



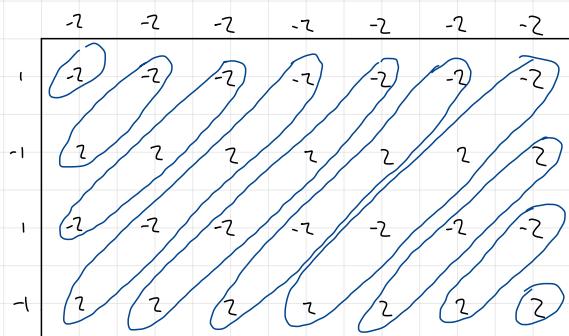
m	$y[m]$	
0	0	= 0
1	$0-2$	= -2
2	$0-2-4$	= -6
3	$0-2-4-6$	= -12
4	$0-2-4-6-4$	= -16
5	$0-2-4-6-4-2$	= -18
6	$0-2-4-6-4-2$	= -18
7	$-2-4-6-4-2$	= -18
8	$-4-6-4-2$	= -16
9	$-6-4-2$	= -12
10	$-4-2$	= -6
11	-2	= -2

$$2.C.) f_1 * f_3 = 0, 1, 1, 2, 0, 0, -2, -1, -1$$



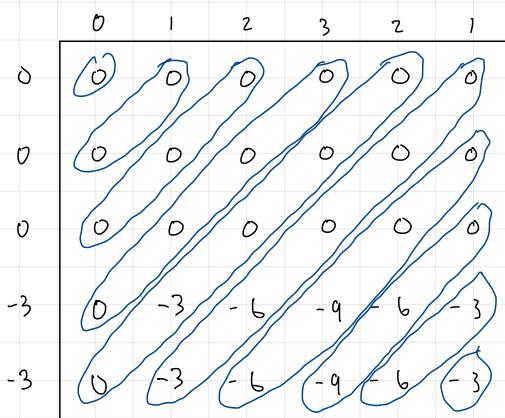
k	$y[k]$	
0	0	= 0
1	$0+1$	= 1
2	$0-1+2$	= 1
3	$0+1-2+3$	= 2
4	$-1+2-3+2$	= 0
5	$-2+3-2+1$	= 0
6	$-3+2-1$	= -2
7	$-2+1$	= -1
8	-1	= -1

$$2.D.) f_2 * f_3 = -2, 0, -2, 0, 0, 0, 0, 2, 0, 2$$



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

$$2.E.) f_1 * f_4 = 0, 0, 0, 0, -3, -9, -15, -15, -9, -3$$



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