Code

```c
#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) {
  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);
}
```

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

```c
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];
    }
  }
}
// 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
#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;
}
```

```
#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;
}
```