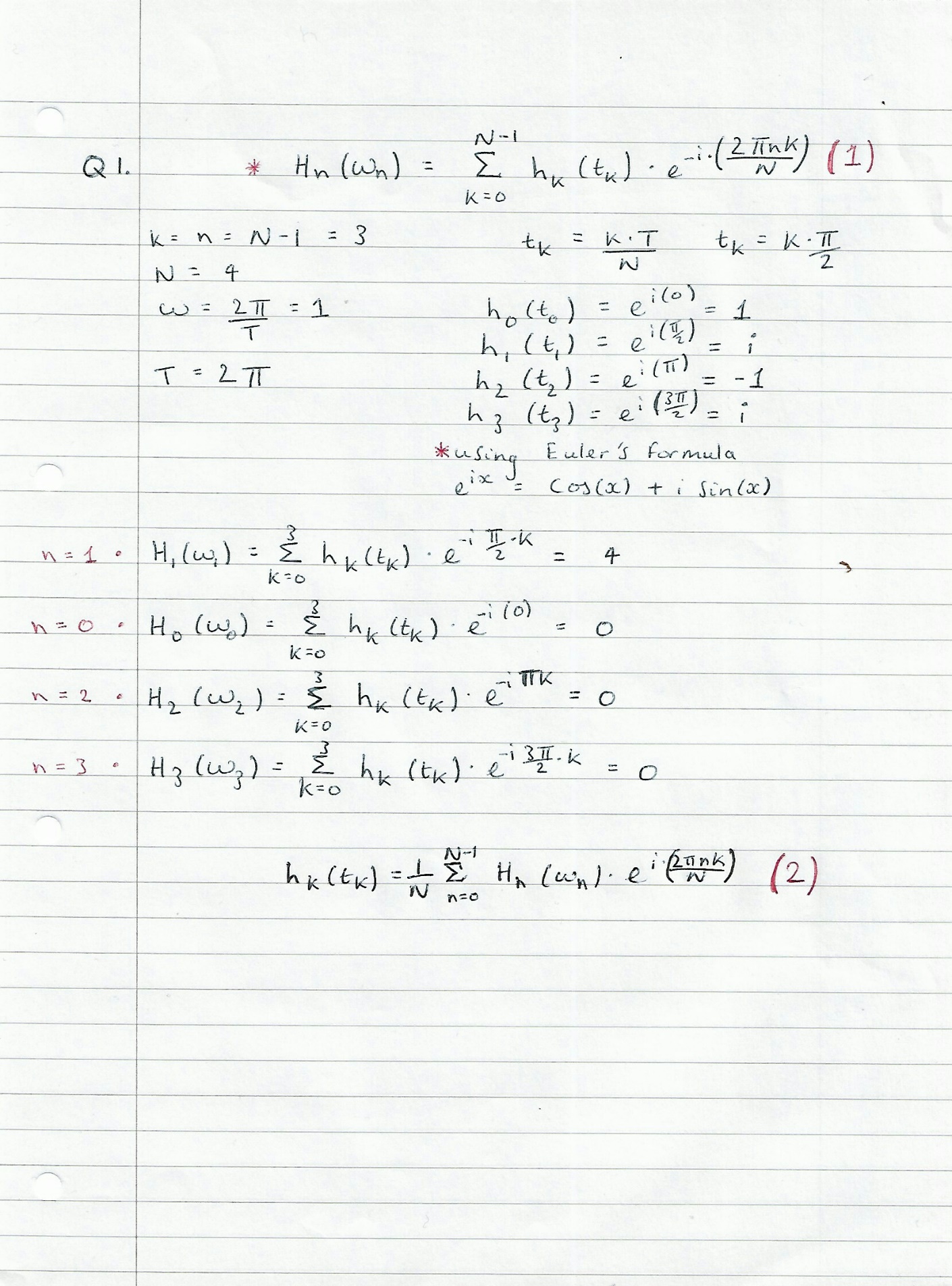
**Discrete Fourier transforms in C**

**Q1. Fourier transform theory**



For each Hn(wn), the complex terms within the summations were expressed as trigonometric functions using Euler’s formula, multiplied together and evaluated for the respective arguments.

**Q2. Program approach**

1. Separate the mathematical functions h1(t) and h2(t) into two global functions, the first defining the respective functions real part and the second defining the imaginary part, and then format a print statement, which requires the standard library <stdio.h> to be imported, to display the sum of the functions with the imaginary part being expressed in terms of i, for instance h1(t) = h1Re + i (h1Im) and similarly h2(t) = h2Re + i (h2Im).

The complex functions will be express as trigonometric functions using Euler’s formula, , and hence the <math.h> library will need to be imported. The function h2(t) has no imaginary part, however is required to explicitly display the imaginary part as zero, hence the function h2Im will be defined to return 0.

1. Open a file for writing the sampled data, create a for loop with respect to argument t with an array for each complex function, whose first element corresponds to the real part and second element to the imaginary part. The sample size, N, is equal to 100 and will be taken over a period of 2π, the interval per sample is , this will be starting point of the argument t in the for loop with a limiting condition of t <= 2π and increasing by for each iteration.
2. Part B will produce 100 values of h1(t) and h2(t) which are printed to their respective files, the print statement will be formatted to write the real part only to conveniently export the data into excel for plotting, this will be repeated for h1Im and h2Re functions, once all required data is taken the print statement will be changed back to produce the full complex functions.
3. The programme will be based on the workings in question 1, with an outer for loop corresponding to each respective n series expansion, with n = N -1 for this instance n = 99, starting with n = 0, boundary condition of n <100 and increasing n by 1 each iteration.

An inner for loop will also be required for the summation of all the sampled functions with respect to k, with k = 0 to k = 99, to create the summation a counter will be required and must reset at the start of each n iteration. To expand the series, the complex exponential and h(tk) are multiplied out in terms of trig functions using Euler’s formula

1. A print statement will be nested within the for loop of n but outside the for loop of k and formatted to print the summation per n and display the respective n for each Hn(wn) function, each Hn function was expressed as an array with elements of the real and imaginary parts.
2. - **h.** The inverse discrete Fourier transform uses the Hn(wn) values calculated in the previous part, thus a Fourier transform will be nested within the for loop of k for the inverse Fourier transform given by equation (2). Counters for the summation, opening a file to be written in and nested print to file statements are all required as in the previous parts to create text files for the computed h’1(t) and h’2(t) functions, which will be exported to excel and plotted over the original h1(t) and h2(t) functions.

The outermost loop of the inverse Fourier transform for k was adjusted to skip n = 1 for the summation of h’1(t) and for that of h’2(t) the for loop will start with n = 1 to skip n = 0. Before doing this the values for h’1(t) will be compared to h1(t) without removing n = 1, if the functions are the same then the inverse transform was successful.

**i. - m.** Open the h3 text file to be read, use a for loop and the scanf statement to read the 200 lines of data and store each variable, (k, t\_k, h3Re, h3Im), to global arrays so the h3 values can be read in other loops of the programs.

The programme written for the inverse Fourier transform will be adapted using a conditional if statement which only lets the for loop of n take the four n values with the largest H3 amplitudes and ignoring the remaining n values as required.

The largest amplitudes are to be found by Fourier transforming h3(t) data in order to find H3 values, apply Pythagoras’ theorem to find their amplitudes, , and filter them in order of the largest value. Lastly, the computed h’3(t) values were writing into a text file, exported to excel and plotted over the original h3(t) plot.

**Q4. Interpretation of Fourier transforms**

Figure 2: h1(t) & h’1(t) real parts

Figure 1: h1(t) & h’1(t) Imaginary parts

Figure 1 is a plot of the imaginary part of the sampled function h1(t) with the h’1(t) plotted over which is equivalent to h1(t) undergoing a Fourier transform to the frequency domain with an amplitude of H1(w) and angular frequency w. The original graph is observed to vary in amplitude, changing polarity about the mid-point of its period, which is roughly 3 (units of time), after the transform the function is observed to have a steady state amplitude.

Figure 2 depicts the real part of h1(t) which decays in amplitude till the mid-point and then exponentially increases beyond that point, similar to the imaginary part, once transformed the amplitude of h’1(t) no longer varies. Figure 3 below, depicts h2(t) and h’2(t) both consisting only of real parts, the transformed function is translated slightly below the original function.

Figure 3: h1(t) & h’1(t)

Figure 4: h3(t) & h’3(t) imaginary parts

Figure 5: h3(t) & h’3(t) real parts

For both figures 4 and 5 once the original h3 signals, real and imaginary parts, undergo transforms the resulting signals are observed to have fine-tuned amplitudes as shown by h’3(t) plots, as Fourier transforms can be used to remove noise. If the spectrum of the noise deviates from the spectrum of the original signal, then the original signal can be extracted by a Fourier transform by identifying the dominate frequency, as a Fourier transform of a function of time can be considered as the relative measure of how often the function oscillates at a given frequency. An inverse Fourier transform can then be used to reconstruct the signal, thus reducing the noise while minimizing any variation to the original signal.

**Appendix. Source code**

1. #include <stdio.h>    /\* Importing standard library for basic statement uses such as printf \*/
2. #include <math.h>     /\* Math library imported for use of trigonometic functions \*/
3. #define PI 3.14159265 /\* Constant PI defined to 8 significant figures  \*/
5. /\* Defining complex expotential functions h1(t) and h2(t) using Euler's formula \*/
6. **float** h1Re(**float** t)
7. {
8. **return** cos(t) + cos(5\*t);
9. }
10. **float** h1Im(**float** t)
11. {
12. **return** sin(t) + sin(5\*t);
13. }
14. **float** h2Re(**float** t)
15. {
16. **return** exp(pow((t - PI),2)/2);
17. }
18. **float** h2Im(**float** t)
19. {
20. **return** 0;
21. }
23. **float** main()
24. {
25. **FILE** \*h1, \*h2, \*h3, \*h1\_prime, \*h2\_prime, \*h3\_prime;   /\* Declaring all files pointers used \*/
26. **float** t, n, k, j, i, t\_k;                              /\* Declaring all variables used\*/
27. **float** h3Re[200], h3Im[200];                           /\* Declaring arrays  \*/
28. **int** H3\_amp[] = {1,3,4,13};

31. /\* h1(t) complex function \*/

34. h1 = fopen("h1.txt","w");
36. **for** (t = (PI/50); t  <= (2\*PI + PI/50); t += (PI/50))  /\* For loop used for sampling function h1(t) over a period of 2PI \*/
37. {
39. **float** t1 = t;
40. **float** cpx\_arrayh1[] = {h1Re(t), h1Im(t)};  /\* Array representing the h1(t) with elements corresponidng the real and imaginary parts \*/
42. fprintf(h1, " h1(t = %1.2f) = %1.2f + i(%1.2f)  \r\n", t, cpx\_arrayh1[0], cpx\_arrayh1[1]);
43. }
44. fclose(h1);

47. /\* discrete fourier transform of h1(t) > H(w) \*/

50. printf( "\n");
51. printf( " H1(w) functions  \n");    /\* Spaces between print to screen functions & header \*/
52. printf( "\n");
54. **for** (n = 0; n < 100; n++)  /\* outer for loop for n iterations of summation within discrete fourier transform equation \*/
55. {
56. **float** sum\_H1Re = 0;  /\* reset running counter of k summation at the start of each n iteration \*/
57. **float** sum\_H1Im = 0;
59. **for** (k = 0; k < 100; k ++) /\* inner for loop for the summation of all h(t) functions, where t is dependant on k \*/
60. {
61. **float** t\_1 = (k \* PI )/ 50;
62. **float** cpx\_arrayH1[] = {cos(PI\*n\*k/50)\*h1Re(t\_1) + sin(PI\*n\*k/50)\*h1Im(t\_1), cos(PI\*n\*k/50)\*h1Im(t\_1) - sin(PI\*n\*k/50)\*h1Re(t\_1)}; /\* expanded terms within summation of discrete fourier transform \*/
64. sum\_H1Re = sum\_H1Re + cpx\_arrayH1[0];  /\*  total summation counters of H1 real and imaginary parts \*/
65. sum\_H1Im = sum\_H1Im + cpx\_arrayH1[1];
66. }
67. printf( " H1(n = %1.2lf) = %1.2lf + i(%1.2lf)  \n", n, sum\_H1Re, sum\_H1Im);
68. }

71. /\* inverse fourier transform of H(w) > h'1(t) \*/

74. h1\_prime = fopen("h1\_prime.txt","w");
75. **for** (**float** j=0 ; j<100; j++) /\* outer for loop for j interations of summation within inverse fourier transform equation \*/
76. {
77. **float** sum\_h1\_primeRe = 0; /\* reset running counter of n summation at the start of each j iteration \*/
78. **float** sum\_h1\_primeIm = 0;
80. **for** (n = 2; n < 101; n++)
81. {
82. **float** sum\_H1Re = 0;
83. **float** sum\_H1Im = 0;
85. **for** (k = 0; k < 100; k ++)
86. {
87. **float** t\_1 = (k \* PI )/ 50;
88. **float** cpx\_arrayH1[] = {cos(PI\*n\*k/50)\*h1Re(t\_1) + sin(PI\*n\*k/50)\*h1Im(t\_1), cos(PI\*n\*k/50)\*h1Im(t\_1) - sin(PI\*n\*k/50)\*h1Re(t\_1)};
90. sum\_H1Re = sum\_H1Re + cpx\_arrayH1[0];
91. sum\_H1Im = sum\_H1Im + cpx\_arrayH1[1];
93. }
94. **float** cpx\_arrayh1prime[] = {cos(PI\*n\*j/50)\*sum\_H1Re - sin(PI\*n\*j/50)\*sum\_H1Im, cos(PI\*n\*j/50)\*sum\_H1Im + sin(PI\*n\*j/50)\*sum\_H1Re}; /\* expanded terms within summation of inverse fourier transform \*/
96. sum\_h1\_primeRe = sum\_h1\_primeRe + cpx\_arrayh1prime[0]; /\* total summation counters of H1 real and imaginary parts \*/
97. sum\_h1\_primeIm = sum\_h1\_primeIm + cpx\_arrayh1prime[1];
99. }
100. fprintf(h1\_prime, " h1\_prime = %1.2f + i(%1.2f)  \r\n", sum\_h1\_primeRe/100, sum\_h1\_primeIm/100);
101. }
102. fclose(h1\_prime);

105. /\* h2(t) complex function \*/

108. h2 = fopen("h2.txt","w");
109. **for** (t = (PI/50); t  < (2\*PI + PI/50); t += (PI/50))
110. {
111. **float** cpx\_array3[] = {h2Re(t), h2Im(t)};
112. fprintf(h2, " h2(t = %1.2f) = %1.2f + i(%1.2f)  \r\n", t, cpx\_array3[0], cpx\_array3[1]);
113. }
114. fclose(h2);

117. /\* discrete fourier transform of h2(t) > H(w) \*/

120. printf( "\n");
121. printf( " H2(w) functions  \n");
122. printf( "\n");
124. **for** (n = 0; n < 100; n++)
125. {
126. **float** sum\_H2Re = 0;
127. **float** sum\_H2Im = 0;
129. **for** (k = 0; k < 100; k++)
130. {
131. **float** t\_1= (k \* PI )/ 50;
132. **float** cpx\_arrayH2[] = {cos(PI\*n\*k/50)\*h2Re(t\_1), - sin(PI\*n\*k/50)\*h2Re(t\_1)};
134. sum\_H2Re = sum\_H2Re + cpx\_arrayH2[0];
135. sum\_H2Im = sum\_H2Im + cpx\_arrayH2[1];
136. }
137. printf( " H2(n = %1.2lf) = %1.2lf + i(%1.2lf)  \n", n, sum\_H2Re, sum\_H2Im);
138. }

141. /\* inverse fourier transform of H(w) > h'2(t) \*/

144. h2\_prime = fopen("h2\_prime.txt","w");
145. **for** (**float** j=0 ; j<100; j++)
146. {
147. **float** sum\_h2\_primeRe = 0;
148. **float** sum\_h2\_primeIm = 0;
150. **for** (n = 1; n < 100; n++)
151. {
152. **float** sum\_H2Re = 0;
153. **float** sum\_H2Im = 0;
155. **for** (k = 0; k < 100; k ++)
156. {
157. **float** t\_1 = (k \* PI )/ 50;
158. **float** cpx\_arrayH2[] = {cos(PI\*n\*k/50)\*h2Re(t\_1) , - sin(PI\*n\*k/50)\*h2Re(t\_1)};
160. sum\_H2Re = sum\_H2Re + cpx\_arrayH2[0];
161. sum\_H2Im = sum\_H2Im + cpx\_arrayH2[1];
163. }
164. **float** cpx\_arrayh2prime[] = {cos(PI\*n\*j/50)\*sum\_H2Re, sin(PI\*n\*j/50)\*sum\_H2Re};
166. sum\_h2\_primeRe = sum\_h2\_primeRe + cpx\_arrayh2prime[0];
167. sum\_h2\_primeIm = sum\_h2\_primeIm + cpx\_arrayh2prime[1];
169. }
170. fprintf(h2\_prime, " h2\_prime = %1.2f + i(%1.2f)  \r\n", sum\_h2\_primeRe/100, sum\_h2\_primeIm/100);
171. }
172. fclose(h2\_prime);


176. /\* fourier transform of h3(t) > H(w) \*/


180. h3 = fopen("h3.txt", "r");     /\* Reading data for h3 from text file and storing variables into arrays using pointers  \*/
181. **for** (**int** i = 0; i < 200; i++)
182. {
183. fscanf(h3, "%d, %e, %e, %e", &i, &t\_k, &h3Re[i], &h3Im[i]);
184. }
185. fclose(h3);
187. **for** (n = 0; n < 200; n++)
188. {
189. **float** sum\_H3Re = 0;
190. **float** sum\_H3Im = 0;
192. **for** (**int** k = 0; k < 200; k++)
193. {
194. **float** cpx\_arrayH3[] = {cos(PI\*n\*k/100)\*h3Re[k] + sin(PI\*n\*k/100)\*h3Im[k], cos(PI\*n\*k/100)\*h3Im[k]- sin(PI\*n\*k/100)\*h3Re[k]};
196. sum\_H3Re = sum\_H3Re + cpx\_arrayH3[0];
197. sum\_H3Im = sum\_H3Im + cpx\_arrayH3[1];
198. }
199. /\* printf( " H3(n = %1.2lf) = %1.2lf + i(%1.2lf)  \n", n, sum\_H3Re, sum\_H3Im);
200. H3(w) values printed to screen to find largest 4 ampltidues for a given n \*/
201. }

204. /\* inverse fourier transform of H(w) > h'3(t) \*/

207. h3\_prime = fopen("h3\_prime.txt","w");
208. **for** (**int** j=0 ; j<200; j++)
209. {
211. **float** sum\_h3\_primeRe = 0;
212. **float** sum\_h3\_primeIm = 0;
214. **for** (n = 1; n < 200; n++)
215. {
216. **if** ( n == H3\_amp[0] || n == H3\_amp[1] || n == H3\_amp[2] || n == H3\_amp[3]) /\* Coniditional if statement used to filter specific values of n through the for loop \*/
217. {
218. **float** sum\_H3Re = 0;
219. **float** sum\_H3Im = 0;
221. **for** (**int** k = 0; k < 200; k ++)
222. {
223. **float** cpx\_arrayH3[] = {h3Re[k]\*cos(PI\*n\*k/100) + h3Im[k]\*sin(PI\*n\*k/100), h3Im[k]\*cos(PI\*n\*k/100) - h3Re[k]\*sin(PI\*n\*k/100)};
225. sum\_H3Re = sum\_H3Re + cpx\_arrayH3[0];
226. sum\_H3Im = sum\_H3Im + cpx\_arrayH3[1];
228. }
229. **float** cpx\_arrayh3prime[] = {cos(PI\*n\*j/100)\*sum\_H3Re - sin(PI\*n\*j/100)\*sum\_H3Im, sin(PI\*n\*j/100)\*sum\_H3Re + cos(PI\*n\*j/100)\*sum\_H3Im};
231. sum\_h3\_primeRe = sum\_h3\_primeRe + cpx\_arrayh3prime[0];
232. sum\_h3\_primeIm = sum\_h3\_primeIm + cpx\_arrayh3prime[1];
234. }
236. }fprintf(h3\_prime, " h3\_prime = %1.2f + i(%1.2f)  \r\n", sum\_h3\_primeRe/200, sum\_h3\_primeIm/200);
238. }
239. fclose(h3\_prime);
240. **return** 0;
241. }