

МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**

ПРЕОБРАЗОВАНИЕ ФУРЬЕ. БЫСТРОЕ ПРЕОБРАЗОВАНИЕ ФУРЬЕ
ОТЧЕТ О ПРАКТИКЕ

Студента 3 курса 311 группы
направления 02.03.02 — Фундаментальная информатика и информационные
технологии
факультета КНиИТ
Аношкина Андрея Алексеевича

Проверил
Старший преподаватель

М. С. Портенко

Саратов 2024

СОДЕРЖАНИЕ

1	Work 09. Вариант 2	3
---	--------------------------	---

1 Work 09. Вариант 2

Задание

Рассчитайте амплитудный спектр тестового сигнала $\sin(2\pi 10t)$ с частотой 10 Hz и амплитудой 1. Длительность сигнала составляет 1 секунду. Частота дискретизации равна числу отсчетов и равна 128. Для значений амплитуды, полученных при помощи БПФ, выполните операцию нормализации.

Периодический сигнал с периодом T , равным 1 секунде, задается функцией слева от знака равенства. Выполните дискретизацию сигнала таким образом, чтобы разрешение по частоте составляло 1 Hz при числе отсчетов 1024. Согласно своему варианту:

- рассчитайте коэффициенты ряда Фурье, используя параллельную программу БПФ;
- вычислите значения функции $f = \frac{a_0}{2} + \sum_{k=1}^{511} (a_k \cos(k \frac{2\pi}{T} t) + b_k \sin(k \frac{2\pi}{T} t))$ при $0 < t < T$, подставив в выражение рассчитанные коэффициенты ряда Фурье;
- сравните подсчитанные значения функции с полученными аналитическим разложением в ряд Фурье и с точными значениями функции при $0 < t < T$.

$$-\ln(2\sin \frac{\pi t}{T}) = \sum_{k=1}^{\infty} \frac{\cos(k \frac{2\pi}{T} t)}{k}, 0 < t < T$$

Быстрое преобразование Фурье

В работе изучаются общие идеи преобразования Фурье и алгоритма быстрого преобразования Фурье (БПФ) с последующей программной реализацией алгоритма БПФ в последовательном и параллельном случаях.

Общие сведения

Основу преобразования Фурье составляет идея о том, что почти любую периодическую функцию можно представить суммой гармонических составляющих или гармоник (синусоид с различными амплитудами, фазами и частотами).

Преобразование Фурье позволяет перейти от рассмотрения сигналов во временной области к их анализу и обработке в частотной области. Во временной области функция времени задается привычным образом, так как по оси

абсцисс откладывается время. В частотной области функция времени отображается несколько иначе за счет того, что по оси абсцисс откладывается частота, а по оси ординат — амплитуда гармоник, составляющих функцию.

Представление функции в частотной области называют спектром функции.

Ряды Фурье

Пусть функция $f(t)$ представляет собой периодический сигнал, имеющий период T . Ряд Фурье функции $f(t)$ по ортогональной системе функций:

$$1, \cos \omega t, \sin \omega t, \dots, \cos k \omega t, \sin k \omega t$$

имеет вид:

$$f(t) = \frac{a_0}{2} + a_1 \cos \omega t + b_1 \sin \omega t + \dots + a_k \cos k \omega t + b_k \sin k \omega t + \dots$$

- ◇ Основная частота $\omega = \frac{2\pi}{T}$ соответствует периоду T , остальные частоты кратны ей.
- ◇ Система функций ортогональна относительно скалярного произведения вида: $\int_{\alpha}^{\alpha+T} g(t)h(t)dt$
- ◇ $a_k = \frac{2}{T} \int_{\alpha}^{\alpha+T} f(t) \cos k \cdot \omega t dt, k = 0, 1, \dots$
- ◇ $b_k = \frac{2}{T} \int_{\alpha}^{\alpha+T} f(t) \sin k \cdot \omega t dt, k = 1, 2, \dots$

Дискретное преобразование Фурье

- ◇ Рассмотрим выражение для комплексного коэффициента c_k :

$$\frac{1}{T} \int_{\alpha}^{\alpha+T} f(t) \cdot e^{-ik\omega t} dt,$$

где ω — основная частота.

- ◇ Выберем дискретные моменты времени для перевода задачи в дискретную форму.: $t_n = n \cdot \Delta t$, где Δt — период дискретизации.
- ◇ Выберем дискретные значения функции в эти моменты: $x_n = f(n \cdot \Delta t)$ (на полном периоде функции оказывается N точек: $N \cdot \Delta t = T$).
- ◇ Подставим полученные выражения в формулу для коэффициентов c_k .

Таким образом, интеграл аппроксимируется интегральной суммой (dt превращается в Δt , также принимается допущение, что за пределами сетки функция периодически повторяется):

$$c_k = \frac{1 \cdot \Delta t}{N \cdot \Delta t} \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi \cdot i \cdot n \cdot \Delta t \cdot k}{N \cdot \Delta t}}, k = 0, \dots, N-1.$$

Масштабный коэффициент $\frac{1}{N}$ не влияет на относительную величину c_k , поэтому указывать его не будем.

Обозначим относительную величину коэффициента c_k через $X(k)$ и получим выражение для **дискретного преобразования Фурье**:

$$X(k) = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i n k}{N}}, k = 0, \dots, N-1.$$

Дискретное преобразование Фурье (ДПФ) ставит в соответствие N отсчетам дискретного сигнала, N отсчетов дискретного спектра, при этом предполагается, что и сигнал, и спектр являются периодическими и анализируются на одном периоде.

Быстрое преобразование Фурье

- ◇ Представленная выше формула для вычисления ДПФ требует значительных затрат. Трудоемкость такого алгоритма имеет порядок $O(N^2)$.
- ◇ В настоящее время существует целая серия оптимизированных алгоритмов расчета ДПФ, которые объединяют под общим названием **быстрое преобразование Фурье (БПФ, FFT, Fast Fourier Transform)**.
- ◇ БПФ не является аппроксимацией ДПФ. Это в точности ДПФ, но с уменьшенным количеством арифметических операций.
- ◇ БПФ — это алгоритм эффективного вычисления ДПФ с трудоемкостью $O(\frac{N}{2} \log_2 N)$

Основные концепции БПФ

Пусть $W_N = e^{-\frac{2\pi i}{N}}$ — **комплексный поворачивающий множитель**, который постоянен для заданного N , тогда выражение для ДПФ примет вид:

$$X(k) = \sum_{n=0}^{N-1} x_n W_N^{nk}, k = 0, \dots, N-1.$$

Разделение исходной последовательности прореживанием по времени

Прореживание по времени заключается в разделении ДПФ на сумму двух ДПФ длиной $\frac{N}{2}$: одно формируется из компонентов с четными индексами x_0, x_2, x_4, \dots , другое — из компонентов с нечетными индексами x_1, x_3, x_5, \dots .

$$\begin{aligned} X(k) &= \sum_{n=0}^{N/2-1} x_{2n} W_N^{2nk} + \sum_{n=0}^{N/2-1} x_{2n+1} W_N^{(2n+1)k} \\ &= \sum_{n=0}^{N/2-1} x_{2n} W_{N/2}^{2nk} + W_N^k \sum_{n=0}^{N/2-1} x_{2n+1} W_{N/2}^{2nk}, k = 0, \dots, N-1. \end{aligned}$$

Теперь заменим каждую полученную сумму на две суммы длиной $N/4$, в свою очередь состоящие из четных и нечетных слагаемых:

$$\begin{aligned} X(k) &= \sum_{n=0}^{N/4-1} x_{4n} W_{N/4}^{nk} + W_{N/2}^k \sum_{n=0}^{N/4-1} x_{4n+2} W_{N/4}^{2nk} + \\ &W_N^k \left(\sum_{n=0}^{N/4-1} x_{4n+1} W_{N/4}^{nk} + W_{N/2}^k \sum_{n=0}^{N/4-1} x_{4n+3} W_{N/4}^{2nk} \right), k = 0, \dots, N-1. \end{aligned}$$

И так далее рекурсивно разделяем вычисления на две части, для этого размер входных данных должен быть степенью двойки: $N = 2^q, q \in \mathbb{N}$. Вычисления разбиваются до тех пор, пока не дойдем до одного элемента. Далее выполняем фиктивное ДПФ над одним элементом, так как для одного числа поворачивающий множитель вычислять не нужно и ДПФ над числом x есть само число x .

Процедура объединения

В результате математических преобразований получили, что два ДПФ четных и нечетных временных отсчетов входного сигнала можно объединить в ДПФ полной длины, если просуммировать отсчеты четной последовательности с произведением отсчетов нечетной последовательности входных сигналов на поворачивающий множитель. Количество операций умножения при этом значительно уменьшается по сравнению с прямым вычислением ДПФ.

Реализация 1

Фрагмент кода решения приведен ниже:

```
1 void MyDataInitialization(complex<double>* mas, int size) {
2     int task = 1;
3     for (int i = 1; i < size; ++i) {
4         switch (task) {
5             case 1: mas[0] = sin(2 * PI * 10 * 1.0 / size * 0); mas[i] = sin(2 * PI * 10 * 1.0 / size * i);
6                     ↪ break;
7             case 2: mas[0] = 1; mas[i] = -log(2 * sin(PI * 1.0 / size * i)); break;
8         }
9     }
10
11 int main() {
12     complex<double>* inputSignal = NULL;
13     complex<double>* outputSignal = NULL;
14     int size = 0;
15     const int repeatCount = 16;
16     double startTime;
17     double duration;
18     double minDuration = DBL_MAX;
19
20     bool serial = false;
21     bool parallel = true;
22     bool test = false;
23
24     cout << "Fast Fourier Transform" << endl;
25
26     if (serial) {
27         cout << "Serial:\n";
28         // Memory allocation and data initialization
29         ProcessInitialization(inputSignal, outputSignal, size);
30         for (int i = 0; i < repeatCount; i++) {
31             startTime = clock();
32
33             // FFT computation
34             SerialFFT(inputSignal, outputSignal, size);
35             duration = (clock() - startTime) / CLOCKS_PER_SEC;
36             if (duration < minDuration)
37                 minDuration = duration;
38         }
39         cout << setprecision(6);
40         cout << "Execution time is " << minDuration << " s. " << endl;
41
42         // Result signal output
43         //PrintSignal(outputSignal, size);
44
45         // Computational process termination
46         ProcessTermination(inputSignal, outputSignal);
```

```

47         cout << "\n";
48     }
49
50     if (parallel) {
51         cout << "Parallel:\n";
52         ProcessInitialization(inputSignal, outputSignal, size);
53         for (int i = 0; i < repeatCount; i++) {
54             startTime = clock();
55
56             // FFT computation
57             ParallelFFT(inputSignal, outputSignal, size);
58             duration = (clock() - startTime) / CLOCKS_PER_SEC;
59             if (duration < minDuration)
60                 minDuration = duration;
61         }
62         cout << setprecision(6);
63         cout << "Execution time is " << minDuration << " s. " << endl;
64
65         // Result signal output
66         //PrintSignal(outputSignal, size);
67
68         // ===== Task-1
69         ↪ =====
70
71         vector<double> A(size);
72         double norm = 0;
73         for (int i = 0; i < size; ++i) {
74             A[i] = (outputSignal[i].real() * outputSignal[i].real() * 4) + (outputSignal[i].imag() *
75                 ↪ outputSignal[i].imag() * 4);
76             norm += A[i];
77             A[i] = sqrt(A[i]);
78         }
79
80         norm = sqrt(norm);
81         for (int i = 0; i < size; ++i) {
82             A[i] /= norm;
83             cout << A[i] << "\n";
84         }
85
86         cout << "\n";
87
88         // ===== Task-2
89         ↪ =====
90
91         /*double t = 1.0 / 1024 * 100;
92         cout << "Accurate value: " << -log(2 * sin(PI * t)) << "\n";
93         double sum = outputSignal[0].real() / size;
94         for (int i = 1; i < size / 2; ++i)
95             sum += outputSignal[i].real() * 2 * cos(i * 2 * PI * t) / size + outputSignal[i].imag() * -2 *
96         ↪ sin(i * 2 * PI * t) / size;

```



```

93
94         cout << "Calculated value: " << sum << "\n";*/
95
96         // Computational process termination
97         ProcessTermination(inputSignal, outputSignal);
98     }
99
100     if (test) {
101         ProcessInitialization(inputSignal, outputSignal, size);
102         ParallelFFT(inputSignal, outputSignal, size);
103         PrintSignal(outputSignal, size);
104         TestResult(inputSignal, outputSignal, size);
105         ProcessTermination(inputSignal, outputSignal);
106     }
107
108     return 0;
109 }

```

Реализация 2

Фрагмент кода решения приведен ниже:

```

1  void MyDataInitialization(complex<double>* mas, int size) {
2      int task = 2;
3      for (int i = 1; i < size; ++i) {
4          switch (task) {
5              case 1: mas[0] = sin(2 * PI * 10 * 1.0 / size * 0); mas[i] = sin(2 * PI * 10 * 1.0 / size * i);
6                  ↪ break;
7              case 2: mas[0] = 1; mas[i] = -log(2 * sin(PI * 1.0 / size * i)); break;
8          }
9      }
10
11     double calcFun(double t) {
12         double sum = 0;
13         for (int i = 1; i < 1000; ++i) {
14             sum += cos(i * 2 * PI * t) / i;
15         }
16
17         return sum;
18     }
19
20     int main() {
21         complex<double>* inputSignal = NULL;
22         complex<double>* outputSignal = NULL;
23         int size = 0;
24         const int repeatCount = 16;
25         double startTime;
26         double duration;
27         double minDuration = DBL_MAX;

```

```

28
29     bool serial = false;
30     bool parallel = true;
31     bool test = false;
32
33     cout << "Fast Fourier Transform" << endl;
34
35     if (serial) {
36         cout << "Serial:\n";
37         // Memory allocation and data initialization
38         ProcessInitialization(inputSignal, outputSignal, size);
39         for (int i = 0; i < repeatCount; i++) {
40             startTime = clock();
41
42             // FFT computation
43             SerialFFT(inputSignal, outputSignal, size);
44             duration = (clock() - startTime) / CLOCKS_PER_SEC;
45             if (duration < minDuration)
46                 minDuration = duration;
47         }
48         cout << setprecision(6);
49         cout << "Execution time is " << minDuration << " s. " << endl;
50
51         // Result signal output
52         //PrintSignal(outputSignal, size);
53
54         // Computational process termination
55         ProcessTermination(inputSignal, outputSignal);
56         cout << "\n";
57     }
58
59     if (parallel) {
60         cout << "Parallel:\n";
61         ProcessInitialization(inputSignal, outputSignal, size);
62         for (int i = 0; i < repeatCount; i++) {
63             startTime = clock();
64
65             // FFT computation
66             ParallelFFT(inputSignal, outputSignal, size);
67             duration = (clock() - startTime) / CLOCKS_PER_SEC;
68             if (duration < minDuration)
69                 minDuration = duration;
70         }
71         cout << setprecision(6);
72         cout << "Execution time is " << minDuration << " s. " << endl;
73
74         // Result signal output
75         //PrintSignal(outputSignal, size);
76

```

```

77 // ===== Task-1
78 ↪ =====
79
80 /*vector<double> A(size);
81 double norm = 0;
82 for (int i = 0; i < size; ++i) {
83     A[i] = (outputSignal[i].real() * outputSignal[i].real() * 4) + (outputSignal[i].imag() *
84 ↪ outputSignal[i].imag() * 4);
85     norm += A[i];
86     A[i] = sqrt(A[i]);
87 }
88
89 norm = sqrt(norm);
90 for (int i = 0; i < size; ++i) {
91     A[i] /= norm;
92     cout << A[i] << "\n";
93 }
94
95 cout << "\n";*/
96
97 // ===== Task-2
98 ↪ =====
99
100 for (double t = 1.0 / size; t < 1; t += 100.0 / size) {
101     cout << "t = " << t << "\n";
102     cout << "Accurate value: " << -log(2 * sin(PI * t)) << "\n";
103     double sum = outputSignal[0].real() / size;
104     for (int i = 1; i < size / 2; ++i)
105         sum += outputSignal[i].real() * 2 * cos(i * 2 * PI * t) / size + outputSignal[i].imag()
106         ↪ * -2 * sin(i * 2 * PI * t) / size;
107
108     cout << "Fourier value: " << calcFun(t) << "\n";
109     cout << "Calculated value: " << sum << "\n";
110 }
111
112 // Computational process termination
113 ProcessTermination(inputSignal, outputSignal);
114 }
115
116 if (test) {
117     ProcessInitialization(inputSignal, outputSignal, size);
118     ParallelFFT(inputSignal, outputSignal, size);
119     PrintSignal(outputSignal, size);
120     TestResult(inputSignal, outputSignal, size);
121     ProcessTermination(inputSignal, outputSignal);
122 }
123
124 return 0;
125 }

```

Результат работы



```
Выбрать Консоль отладки Microsoft Visual Studio
Parallel:
Enter the input signal length: 128
Input signal length = 128
Execution time is 0 s.
5.3321e-17
1.11032e-16
1.33184e-16
1.1979e-16
9.58979e-17
4.73915e-17
1.43648e-16
5.95877e-17
1.744e-16
2.55549e-16
0.707107
3.15735e-16
1.17378e-16
1.18433e-16
5.75614e-17
1.21573e-16
1.36729e-16
9.43416e-17
1.15784e-16
1.72368e-16
3.20549e-17
1.19975e-16
2.88929e-17
```

Рисунок 1 – Work-9-1



```
Консоль отладки Microsoft Visual Studio
Fast Fourier Transform
Parallel:
Enter the input signal length: 1024
Input signal length = 1024
Execution time is 0 s.
t = 0.000976562
Accurate value: 5.0936
Fourier value: 5.04691
Calculated value: 5.08916
t = 0.0986328
Accurate value: 0.494528
Fourier value: 0.493709
Calculated value: 0.49009
t = 0.196289
Accurate value: -0.145509
Fourier value: -0.144703
Calculated value: -0.149947
t = 0.293945
Accurate value: -0.467113
Fourier value: -0.467711
Calculated value: -0.471552
t = 0.391602
Accurate value: -0.634005
Fourier value: -0.633709
Calculated value: -0.638444
t = 0.489258
Accurate value: -0.692578
```

Рисунок 2 – Work-9-2

Характеристики устройства

Процессор: Intel(R) Core(TM) i5-10400F

Ядер: 6

Оперативная память: 16 Гб