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

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 $0 < t < T$.

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

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:

$$1, Cos\omega t, Sin\omega t, \dots, Cosk\omega t, Sink\omega t$$

имеет вид:

$$f(t) = \frac{a_0}{2} + a_1 Cos\omega t + b_1 Sin\omega t + \dots + a_k Cosk\omega t + b_k Sink\omega t + \dots$$

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

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 переведа 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 превратится в Δt, также принимается допущение, что за пределами сетки функция периодически повторяется):

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

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***:

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

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:

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

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 $x_0, x_2, x_4, \ldots$, другое — из компонентов с нечетными индексами $x_1, x_3, x_5, \ldots$

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

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

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

И так далее рекурсивно разделяем вычисления на две части, для этого размер входных данных должен быть степенью двойки: $N = 2^q, q \in N$. 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:

```
void MyDataInitialization(complex<double>* mas, int size) {
	int task = 1;
	for (int i = 1; i < size; ++i) {
		switch (task) {
			case 1: mas[0] = sin(2 * PI * 10 * 1.0 / size * 0); mas[i] = sin(2 * PI * 10 * 1.0 / size * i);
			↪ break;
			case 2: mas[0] = 1; mas[i] = -log(2 * sin(PI * 1.0 / size * i)); break;
		}
	}
}

int main() {
	complex<double>* inputSignal = NULL;
	complex<double>* outputSignal = NULL;
	int size = 0;
	const int repeatCount = 16;
	double startTime;
	double duration;
	double minDuration = DBL_MAX;

	bool serial = false;
	bool parallel = true;
	bool test = false;

	cout << "Fast Fourier Transform" << endl;

	if (serial) {
		cout << "Serial:\n";
		// Memory allocation and data initialization
		ProcessInitialization(inputSignal, outputSignal, size);
		for (int i = 0; i < repeatCount; i++) {
			startTime = clock();

			// FFT computation
			SerialFFT(inputSignal, outputSignal, size);
			duration = (clock() - startTime) / CLOCKS_PER_SEC;
			if (duration < minDuration)
				minDuration = duration;
		}
		cout << setprecision(6);
		cout << "Execution time is " << minDuration << " s. " << endl;

		// Result signal output
		//PrintSignal(outputSignal, size);

		// Computational process termination
		ProcessTermination(inputSignal, outputSignal);
```

```
        cout << "\n";
    }

    if (parallel) {
        cout << "Parallel:\n";
        ProcessInitialization(inputSignal, outputSignal, size);
        for (int i = 0; i < repeatCount; i++) {
            startTime = clock();

            // FFT computation
            ParallelFFT(inputSignal, outputSignal, size);
            duration = (clock() - startTime) / CLOCKS_PER_SEC;
            if (duration < minDuration)
                minDuration = duration;
        }
        cout << setprecision(6);
        cout << "Execution time is " << minDuration << " s. " << endl;

        // Result signal output
        //PrintSignal(outputSignal, size);

        // ========================================= Task-1 ==================================================

        vector<double> A(size);
        double norm = 0;
        for (int i = 0; i < size; ++i) {
            A[i] = (outputSignal[i].real() * outputSignal[i].real() * 4) + (outputSignal[i].imag() * outputSignal[i].imag() * 4);
            norm += A[i];
            A[i] = sqrt(A[i]);
        }

        norm = sqrt(norm);
        for (int i = 0; i < size; ++i) {
            A[i] /= norm;
            cout << A[i] << "\n";
        }

        cout << "\n";

        // ========================================= Task-2 ==================================================

        /*double t = 1.0 / 1024 * 100;
        cout << "Accurate value: " << -log(2 * sin(PI * t)) << "\n";
        double sum = outputSignal[0].real() / size;
        for (int i = 1; i < size / 2; ++i)
            sum += outputSignal[i].real() * 2 * cos(i * 2 * PI * t) / size + outputSignal[i].imag() * -2 * sin(i * 2 * PI * t) / size;
```

```

        cout << "Calculated value: " << sum << "\n";*/

        // Computational process termination
        ProcessTermination(inputSignal, outputSignal);
    }

    if (test) {
        ProcessInitialization(inputSignal, outputSignal, size);
        ParallelFFT(inputSignal, outputSignal, size);
        PrintSignal(outputSignal, size);
        TestResult(inputSignal, outputSignal, size);
        ProcessTermination(inputSignal, outputSignal);
    }

    return 0;
}
```

Реализация 2

Фрагмент кода решения приведен ниже:

```
void MyDataInitialization(complex<double>* mas, int size) {
    int task = 2;
    for (int i = 1; i < size; ++i) {
        switch (task) {
            case 1: mas[0] = sin(2 * PI * 10 * 1.0 / size * 0); mas[i] = sin(2 * PI * 10 * 1.0 / size * i); break;
            case 2: mas[0] = 1; mas[i] = -log(2 * sin(PI * 1.0 / size * i)); break;
        }
    }
}

double calcFun(double t) {
    double sum = 0;
    for (int i = 1; i < 1000; ++i) {
        sum += cos(i * 2 * PI * t) / i;
    }

    return sum;
}

int main() {
    complex<double>* inputSignal = NULL;
    complex<double>* outputSignal = NULL;
    int size = 0;
    const int repeatCount = 16;
    double startTime;
    double duration;
    double minDuration = DBL_MAX;
```

```
    bool serial = false;
    bool parallel = true;
    bool test = false;

    cout << "Fast Fourier Transform" << endl;

    if (serial) {
        cout << "Serial:\n";
        // Memory allocation and data initialization
        ProcessInitialization(inputSignal, outputSignal, size);
        for (int i = 0; i < repeatCount; i++) {
            startTime = clock();

            // FFT computation
            SerialFFT(inputSignal, outputSignal, size);
            duration = (clock() - startTime) / CLOCKS_PER_SEC;
            if (duration < minDuration)
                minDuration = duration;
        }
        cout << setprecision(6);
        cout << "Execution time is " << minDuration << " s. " << endl;

        // Result signal output
        //PrintSignal(outputSignal, size);

        // Computational process termination
        ProcessTermination(inputSignal, outputSignal);
        cout << "\n";
    }

    if (parallel) {
        cout << "Parallel:\n";
        ProcessInitialization(inputSignal, outputSignal, size);
        for (int i = 0; i < repeatCount; i++) {
            startTime = clock();

            // FFT computation
            ParallelFFT(inputSignal, outputSignal, size);
            duration = (clock() - startTime) / CLOCKS_PER_SEC;
            if (duration < minDuration)
                minDuration = duration;
        }
        cout << setprecision(6);
        cout << "Execution time is " << minDuration << " s. " << endl;

        // Result signal output
        //PrintSignal(outputSignal, size);

```

```
        // ========================================= Task-1 ==================================================

        /*vector<double> A(size);
        double norm = 0;
        for (int i = 0; i < size; ++i) {
            A[i] = (outputSignal[i].real() * outputSignal[i].real() * 4) + (outputSignal[i].imag() * outputSignal[i].imag() * 4);
            norm += A[i];
            A[i] = sqrt(A[i]);
        }

        norm = sqrt(norm);
        for (int i = 0; i < size; ++i) {
            A[i] /= norm;
            cout << A[i] << "\n";
        }

        cout << "\n";*/

        // ========================================= Task-2 ==================================================

        for (double t = 1.0 / size; t < 1; t += 100.0 / size) {
            cout << "t = " << t << "\n";
            cout << "Accurate value: " << -log(2 * sin(PI * t)) << "\n";
            double sum = outputSignal[0].real() / size;
            for (int i = 1; i < size / 2; ++i)
                sum += outputSignal[i].real() * 2 * cos(i * 2 * PI * t) / size + outputSignal[i].imag() * -2 * sin(i * 2 * PI * t) / size;

            cout << "Fourier value: " << calcFun(t) << "\n";
            cout << "Calculated value: " << sum << "\n";
        }

        // Computational process termination
        ProcessTermination(inputSignal, outputSignal);
    }

    if (test) {
        ProcessInitialization(inputSignal, outputSignal, size);
        ParallelFFT(inputSignal, outputSignal, size);
        PrintSignal(outputSignal, size);
        TestResult(inputSignal, outputSignal, size);
        ProcessTermination(inputSignal, outputSignal);
    }

    return 0;
}
```

Результат работы

Рисунок 1 – Work-9-1

Рисунок 2 – Work-9-2

Характеристики устройства

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

Ядер: 6

Оперативная память: 16 Гб