**Курсовая работа  
по информационным технологиям  
по теме «Графическое приложение: вычисление дискретного преобразования Фурье для синусоиды с линейно изменяющейся частотой»**

|  |  |
| --- | --- |
| Выполнил студент ФРТ гр. 1183 | Чаминов Д. А. |
| Преподаватель | Ситников И. Ю. |

# Содержание

[Содержание 2](#_Toc104308145)

[Спецификация задания 3](#_Toc104308146)

[Постановка задачи 5](#_Toc104308147)

[Формализованное описание алгоритма решения задачи 7](#_Toc104308148)

[Варианты взаимодействия оператора и программы (Use Case) 10](#_Toc104308149)

[TODO: Диаграмма потоков данных (Data flow diagram) 11](#_Toc104308150)

[Выбор и обоснование типов переменных. Разработка структур данных 12](#_Toc104308151)

[Вводимые и выводимые параметры и их типы 15](#_Toc104308152)

[Диаграмма классов 16](#_Toc104308153)

[Структура проекта, перечисление нужных файлов 17](#_Toc104308154)

[Инструкция по использованию 19](#_Toc104308155)

[TODO: Текст программы и файлов заголовков с комментариями 23](#_Toc104308156)

[TODO: Рисунки с копиями экрана при работе программы 24](#_Toc104308157)

[Контрольный пример, сравнение результата с эталоном 25](#_Toc104308158)

[TODO: Ведомость соответствия программы спецификации 30](#_Toc104308159)

[TODO: Выводы 31](#_Toc104308160)

# Спецификация задания

* 1. Требования к расчёту и программе:
     1. Программа должна выполнять дискретное преобразование Фурье (ДПФ) для сигнала вида в соответствии с введенными пользователем параметрами и визуализировать результат расчёта в виде графика модуля ДПФ;
     2. Программа должна строить график сигнальной функции с возможностью его масштабирования, смещения, использования логарифмического масштаба по оси ординат, изменения этих и параметров сигнальной функции пользователем, а также иметь возможность сохранения графика в файл в формате BMP (bitmap image);
     3. Поддержка универсальной символьной кодировки Unicode;
     4. Наличие графического интерфейса;
     5. Использование пользовательского типа данных и оперирование динамическими массивами;
     6. Программа должна разрабатываться в среде MS Visual Studio на языке программирования C++ с применением библиотек Microsoft Foundation Classes (MFC) как оконное приложение на базе диалогов для операционных систем MS Windows 7 (x32) и выше.
     7. Приложение должно иметь сведения о программе, авторе и авторских правах, название и иконку;
     8. Интерфейс должен использовать кнопки для управления приложением, поля ввода данных для задания сигнальной функции;
     9. В приложении должны содержаться списки выбора;
     10. Интерфейс должен быть русскоязычным, интуитивно понятным, соответствовать понятию «дружественный интерфейс»;
  2. Требования к отчёту:
     1. Отчёт должен соответствовать ГОСТу 19.701–90 единой системы программной документации;
     2. Отчет должен содержать описание программного интерфейса, диаграмму классов и диаграмму потоков данных, выбор и обоснование переменных, пользовательских типов и классов, код программы с комментариями, пример работы программы и контрольный пример;

Срок сдачи отчета:

Срок сдачи курсовой работы:

|  |  |  |
| --- | --- | --- |
| Преподаватель |  | Ситников И. Ю. |
| Студент |  | Чаминов Д. А. |

# Постановка задачи

Задача: для сигнала вида , где – параметры, задаваемые пользователем рассчитать дискретное преобразование Фурье, построить график сигнальной функции и график модуля ее ДПФ.

Дискретное преобразование Фурье переводит вектор в вектор частотных отсчетов . Такое преобразование можно рассматривать умножение матрицы на вектор , где – матрица преобразования (унитарная матрица для некоторого базиса пространства ).

Для компонента преобразования справедлива следующая формула:  
где – число отсчетов, – отсчет сигнальной функции с номером , – число Эйлера, – число пи и – мнимая единица. Применяя к этому выражению формулу Эйлера, получим следующее:

Учитывая четность и нечетность , а также то, что в данной задаче сигнал вещественный, получим:

Модуль

Для расчёта положения точки на экране необходимо перейти к другому масштабу и учесть смещение. Если отрезок длиной 1 должен на экране по оси совпадать с длиной пикселей, а по оси – , левая верхняя точка графика имеет координаты , а высота графика равна пикселей, то для перехода к координатам окна можно использовать следующие преобразования:

Разность координат показывает смещение точки относительно координат окна, но в координатах графика; умножение на коэффициент масштаба показывает длину в пикселях проекции радиус-вектора, проведенного из нового начала координат в эту точку; знак минус перед показывает отражение системы координат, а компенсирует вызванное таким поворотом смещение.

# Формализованное описание алгоритма решения задачи

Для выполнения расчётов в программе определено несколько пользовательских классов. Все классы диалогов являются наследниками CDialogEx.

Первый из них – класс начального окна courseworkDlg. Он содержит один CStatic элемент, в котором дано краткое описание задачи, кнопку открытия диалога с более подробными сведениями о задаче, кнопку открытия диалога с расчётом и кнопку закрытия программы.

Диалог с описанием – MyTaskAboutDlg содержит CStatic с более подробным описанием задания.

Диалог с расчётом Calculator содержит в себе элементы управления для настройки параметров сигнальной функции и графиков: 7 CEdit (3 для параметров функции и 4 для настроек масштаба) , 1 CSliderCtrl для удобной настройки шага, CButton – check box (флажок) для установки логарифмического масштаба, 4 CMFCColorButton – выпадающие списки для выбора цвета, 4 кнопки (обновление графика, его сохранение, сброс к начальным параметрам и закрытие диалога), 15 CStatic (1 для отображения сигнальной функции и 14 для пояснения приглашений ко вводу), а также объект пользовательского класса CMyGraph для отрисовки графиков и объекты пользовательских классов SignalFunction и DFTFunction – объекты, описывающие сигнальную функцию и функцию дискретного преобразования Фурье, необходимые для построения графиков.

При нажатии кнопки сохранения должен вызываться метод для вызова диалога сохранения void Calculator::OnBnClickedButtonSaveGr(), в котором из объекта CMyGraph извлекается картинка и сохраняется в выбранное место.

При нажатии кнопки сброса вызывается метод void Calculator::OnBnClickedButtonreset(), в котором сбрасываются значения всех элементов управления.

При нажатии кнопки обновления вызывается метод void Calculator::OnBnClickedButtonreset(), в которой вызывается метод получения параметров из всех элементов управления void Calculator::UpdateCalculatorParams(), в котором происходит проверка заполнения элементов (с созданием диалогового окна об ошибке или присвоением соответствующих параметров через методы CMyGraph или напрямую объектам сигнальной или полученной функций). Помимо обновления параметров, из этого метода через CWnd::RedrawWindow() система отправляет элементу CMyGraph сообщение о необходимости перерисовать окно.

CMyGraph является наследником CStatic. Он содержит в себе автоматический динамический массив объектов типа MathFunction\* – ссылки на функции, графики которых необходимо построить, а также 2 структуры из 2-х чисел типа double – границы области построения.

При получении сообщений о перерисовке вызывается метод void CMyGraph::OnPaint(), который получает от каждой из функций массив точек (при помощи метода MathFunction::get\_points() и строит график, последовательно соединяя точки при помощи CPaintDC::LineTo(Point), предварительно залив фон области построения необходимым цветом.

Класс CMyGraph обладает множеством set’еров, которые вызывают соответствующие методы для каждой из функций.

MathFunction – абстрактный класс (интерфейс), объекты наследников (реализаций) которого содержат сведения об области определения, области построения, размерах расчётного промежутка, типе масштаба, цвете линий функции на графике, а также о том, являются ли рассчитанные точки актуальными; методы установки параметров, расчёта математических точек (чисто виртуальная функция), расчёта точек (в координатах окна), пересчёта точек в координаты окна, а также автоматический динамический массив объектов POINT – точек в координатах окна и метод его передачи для отрисовки.

Метод передачи точек MathFunction::get\_points() проверяет их актуальность и вызывает (в случае если данные устарели) метод пересчёта точек MathFunction::calulate(), который с заданным шагом на промежутке пересечения области определения и расчётной области вызывает метод MathFunction::f() расчёта функции в этой точке и метод перевода координат, полученной таким образом точки в координаты окна POINT MathFunction::to\_the\_new\_coords\_system(double x, double y) const, которая основываясь на формулах пересчета приведенных выше вычисляет координаты точки в окне.

Классы SignalFunction и DFTFunction содержат в себе некоторые дополнительные сведения (коэффициенты и значения функции в отсчетах для сигнальной функции и ссылка на сигнальную функцию для класса функции ДПФ) и методы установки этих параметров.

Расчёт в классе SignalFunction отличается тем, что значения функции в точках на расчетном промежутке сохраняются в vector<double> data для дальнейшего применения ДПФ.

В классе DFTFunction расчёт отличается тем, что в качестве параметра функции f передается номер отсчета, для которого производятся вычисления. В самой функции f происходит расчёт мнимой и действительной частей при помощи суммирования данных сигнальной функции с коэффициентами, соответствующими номерам отсчета. Затем находится модуль ДПФ путем извлечения корня из суммы квадратов мнимой и реальной частей.

# Варианты взаимодействия оператора и программы (Use Case)

# TODO: Диаграмма потоков данных (Data flow diagram)

# Выбор и обоснование типов переменных. Разработка структур данных

В программе для данных о координатах окна или о координатах объекта в окне используется типы POINT и Cpoint, которые хранят два целых числа int, диапазона которых достаточно, чтобы однозначно указать одну из координат точки экрана.

Для хранения данных о границах элементов используются типы CRect и RECT, которые хранят координаты двух противолежащих точек прямоугольника в виде целых чисел int.

Для представления цвета используется тип COLORREF, который является синонимом беззнакового 32-разрядного целого (unsigned long int) хранит данные о трех компонентах цвета (красной, зеленой и голубой) в виде битовых полей. Для краткой записи преобразования 3-х восьмибитных компонент используется макрос RGB.

В качестве блока расчёта выступают несколько объектов типа MathFunction. Этот тип содержит базовые данные о «функции» (здесь под функцией MathFunction подразумевается некоторый объект, который численными методами генерирует последовательность точек в данной области с заданным шагом для отрисовки графика, причем точки определяются в координатах области построения): область определения некоторой функции (в виде 2х чисел с плавающей точкой двойной точности double), данные об области вычислений (в виде анонимной структуры из 4х чисел double, содержащей начальные и конечные значения по каждой из осей), данные об области отрисовки (CRect), шаг расчёта (double), данные о цвете графика функции (COLORREF), индикатор того, произведен ли уже расчёт (bool) и индикатор логарифмического масштаба (bool). Все вышеперечисленные параметры кроме цвета являются защищенными (недоступны вне дочерних классов), поскольку их изменение напрямую может привести к недостоверности сведений об актуальности рассчитанных координат (несоответствие индикатора is\_calculated действительности, которое в свою очередь приводит к сохранению неактуальных данных). Для этих параметров определены соответствующие set’теры; get’теры определены только для массива точек и индикатора готовности.

Наследником класса MathFunction является SignalFunction, который дополнительно хранит параметры сигнальной функции (3 числа double) и автоматический динамический массив std::vector<double> - массив значений функции для дальнейшего расчёта ДПФ и get’тер для этого массива, возвращающий константную ссылку.

Наследником SignalFunction является DFTFunction, который производит дискретное преобразование Фурье. Этот класс обладает 1 дополнительным атрибутом – ссылкой на объект сигнальной функции (SignalFunction\*), через которую осуществляется доступ к данным отсчетов сигнальной функции. Атрибуты SignalFunction (параметры ) в этом классе необходимы для того, чтобы отслеживать изменения сигнальной функции независимо от неё – обращение с обновлением к DFTFunction, а затем только к SignalFunction приведет к обновлению только сигнальной функции. Причем этот порядок может быть любым, поэтому появляется необходимость в этих параметрах.

Построением графиков занимаются объекты класса CMyGraph – наследник класса CStatic из библиотек MFC. Переменные-члены этого класса можно условно разделить на 4 группы: необходимые для анимации, отвечающие за растровые изображения графиков, влияющие только на визуальную составляющую (в их отсутствии график был бы правильным, но выглядел бы хуже), отвечающие за отрисовку самого графика.

К первой группе относятся 3 переменные – текущая фаза анимации деленная на  (т. е. в долях от единицы) (float), скорость анимации – то, на сколько увеличится фаза за 1 кадр (циклическая частота) (float), показывающий идет ли анимация индикатор (bool), который при включенной анимации эквивалентен выражению .

Ко второй группе можно отнести 2 переменные CBitmap (из библиотек MFC), в которых хранятся фоновое изображение с осями и сам график, индикаторы готовности и актуальности этих растровых изображений – 2 переменных типа bool, 2 объекта CDC (из библиотек MFC) – инструменты для отрисовки фона и графика и 2 HGDIOBG – указатели на объекты начальных настроек CDC.

К третьей группе относятся 1 переменная POINT, показывающая смещение графика относительно области построения (вдоль осей графика, но в точках экрана), индикатор, показывающий, является ли график гистограммой – 1 bool, 1 переменная типа int, которая содержит в себе длину засечек на осях, 1 объект, представляющий анонимную структуру из 2х чисел int, показывающий число засечек на каждой из осей, 1 переменная COLORREF – цвет фона.

К последней группе относятся 2 экземпляра анонимной структуры из двух чисел double, определяющие область построения (границы значений по каждой из осей), индикатор логарифмического масштаба bool, автоматический динамический массив указателей на объекты «функций» std::vector<MathFunction\*>, которые необходимо отобразить на графике. Поскольку SignalFunction и DFTFunction являются реализациями интерфейса MathFunction, указатели на них можно использовать в качестве указателей на MathFunction. Иными словами – объекты типов SignalFunction и DFTFunction являются и объектами типа MathFunction.

Следующие пользовательские типы – классы окон приложения. Они являются наследниками класса CDialogEx (из библиотек MFC). Всего в программе определено 4 класса окна: главное окно с кратким описанием задания, окно с более подробными теоретическими сведениями, окно «О программе» и окно для расчёта. Окна с описанием программы и теоретическими сведениями не содержат атрибутов. Начальное окно содержит строку CString – краткий текст задания и иконку HICON. Окно расчёта содержит в себе множество элементов контроля: 3 CMFCColorButton для выбора цвета, CSliderCtrl для выбора шага, 9 CEdit для ввода значений, 3 CButton для установки логарифмического масштаба и включения анимации. Кроме элементов контроля это окно содержит в себе 2 CMyGraph для отображения графиков сигнальной функции и гистограммы модуля её ДПФ.

Само оконное приложение – единственный экземпляр класса CcourseworkApp, являющегося наследником CWinApp и не имеющего атрибутов.

# Вводимые и выводимые параметры и их типы

В программу через элементы контроля поступают следующие данные:

1. Через CEdit поступают строки CString, содержащие основную информацию о сигнальной функции и областях построения графиков. Эти данные преобразуются к типу double, поэтому вводимые строки могут содержать только знак (+ или -), цифры, точку и символы “e” или “E” для записи числа в экспоненциальной форме.
2. Через кнопки CButton поступает информация о режимах масштаба и анимации в виде логических переменных типа bool.
3. Через ползунок CSliderCtrl в программу поступает целое число – шаг расчёта (в пикселях).
4. Через элементы CMFCColorButton поступают переменные типа COLORREF, содержащие информацию о цветах графика, выбранных пользователем.

Также в программу поступают данные о состоянии курсора (для трассировки графиков) в виде переменной CPoint, содержащей его координаты, и переменной UINT\_PTR – набора флагов состояния курсора и основных управляющих клавиш.

Вывод текстовых данных, хранящихся в строках CString, осуществляется с помощью диалоговых окон AfxMessageBox и элементов CStatic.

Вывод графических данных осуществляется 2 способами: через элементы CMyGraph и через сохранение на жестком диске в форматах bmp, gif, png, jpg.

# Диаграмма классов



# Структура проекта, перечисление нужных файлов

Основные файлы программы находятся в директории coursework.

Файлы Calculator.h, courseworkDlg.h содержат объявления классов диалоговых окон (расчёта, основного окна и окна «О программе»), а соответствующие .cpp файлы содержат реализации этих классов. В файлах CMyGraph.h и CMyGraph.cpp содержится объявление и определение класса CMyGraph – элемента окна, в котором происходит отрисовка графиков. В файлах MathFunction.h и MathFunction.cpp содержатся объявления и определения классов, которые определяют вид функций для отрисовки в элементах CMyGraph. Файл logo.ico содержит иконку окна приложения, а coursework.ico – иконку исполняемого файла. Файлы coursework.rc и coursework.rc2 содержат разметку диалоговых окон и данные о программе. Остальные файлы являются служебными, созданы средой разработки MS Visual Studio и являются необходимыми для сборки приложения. Дерево структуры директорий и файлов приведено ниже.

**C:\USERS\D.MON\DOCUMENTS\УЧЕБА\ИНФОРМАТИКА\CHAMINOV\_DMEATRY\COURSEWORK**

**│ coursework.sln**

**│ Курсовая работа.docx**

**│**

**└───coursework**

**│ Calculator.cpp**

**│ Calculator.h**

**│ ClassDiagram.cd**

**│ CMyGraph.cpp**

**│ CMyGraph.h**

**│ coursework.aps**

**│ coursework.cpp**

**│ coursework.h**

**│ coursework.rc**

**│ coursework.vcxproj**

**│ coursework.vcxproj.filters**

**│ coursework.vcxproj.user**

**│ courseworkDlg.cpp**

**│ courseworkDlg.h**

**│ framework.h**

**│ logo.ico**

**│ MathFunction.cpp**

**│ MathFunction.h**

**│ pch.cpp**

**│ pch.h**

**│ resource.h**

**│ targetver.h**

**│**

**└───res**

**coursework.ico**

**coursework.rc2**

# Инструкция по использованию

1. Запустите приложение (двойной щелчок левой кнопки мыши (далее ЛКМ) по файлу coursework.exe).
2. Для просмотра дополнительных сведение о преобразовании щелкните левой кнопкой мыши по кнопке с надписью «Подробнее».
3. Для закрытия окна с подробными сведениями щелкните ЛКМ по кнопке «Закрыть» или кнопке с изображением креста в правом верхнем углу окна.
4. Для перехода к вычислениям из главного окна (окно с кратким описанием задания и кнопками «Закрыть», «Подробнее» и «Вычислить») щелкните ЛКМ по кнопке «Вычислить».
5. В поля ввода в соответствии с параметрами, указанными возле них, введите значения соответствующих переменных.
6. Вводимые в эти поля данные могут содержать только число, состоящее из арабских цифр, знака («+» или «-»), точки, или число записанное в экспоненциальном виде (см п. 2.7)
7. При записи числа в экспоненциальном формате выглядят так: “-1.23e45”, где символ “e” или “E” (оба символа относятся к латинскому алфавиту) разделяет число на 2 части, которые будут восприняты программой как число . Такой формат числа (как “-1.23e45”) может быть использован при выводе значений на штрихах рядом с осями, его следует трактовать тем же образом.
8. Вместо числа в программу может быть передано (как и выведено на экран) значение “+inf” или “-inf”. При вводе они будут соответствовать самым большим числам, помещающимся в памяти (границам области допустимых значений данных). Такие числа не рекомендуются к использованию по понятным причинам. В случае, если программа вместо чисел выводит эти значения следует уменьшить диапазон, в котором производятся вычисления.
9. Для навигации между полями ввода и другими элементами окна можно использовать ЛКМ или клавишу «Tab».
10. Поля, между которыми стоит многоточие («…») обозначают диапазоны расчётов и границы области построение. В левое поле следует вводить меньшую из границ, а в правую – большую.
11. При использовании логарифмического масштаба графика следует указывать положительные значения границ построения по оси, для которой включен логарифмический масштаб.
12. Для построения гистограммы отображающей модуль ДПФ в целях удобного отображения диаграммы рекомендуется в качестве нижней границы использовать неотрицательные числа.
13. Для настройки шага расчёта используйте «ползунок» с подписью «Шаг расчёта». Крайнее левое положение соответствует состоянию, при котором в результате масштабирования 1 отсчёт будет иметь ширину в 1 точку экрана вдоль оси абсцисс, а при крайнем правом – 10 точек экрана. Таким образом «действительный» шаг расчёта настраивается еще и шириной диапазона расчёта, т. е. для получения меньшего шага следует передвинуть ползунок в крайнее левое положение и (при необходимости) взять более узкий промежуток по оси абсцисс.
14. Для изменения состояния настройки логарифмического масштаба щелкните ЛКМ по квадрату рябом с надписью «Логарифмический масштаб» или по самой надписи.
15. Состояние, при котором в квадрате отображается «галочка», соответствует состоянию «включен», а при отсутствии «галочки» – «выключен».
16. При включении масштаба границы области построения меняются по правилу , где – новое значение, а – старое, что позволяет удобно задать границы до включения логарифмического масштаба путем ввода показателя степени 10. При выключении происходит обратное преобразование .
17. Включение и выключение анимации происходит аналогично настройке логарифмического масштаба, но с полем «Анимация».
18. Для настройки цветов фона, графика сигнальной функции, столбцов гистограммы модуля ДПФ щелкните ЛКМ по цветному квадрату, расположенному рядом с соответствующей надписью («Фон», «Сигнал», «ДПФ»); в открывшейся палитре выберите цвет, щелкнув по квадрату соответствующего цвета, расположенном в этой палитре.
19. Для отображения графика и диаграммы используйте кнопку «Обновить». Если в квадрате рядом с надписью «Анимация» стоит «галочка», то при нажатии на кнопку «Обновить» автоматически запустится анимация.
20. Для сброса параметров и настроек к начальным используйте кнопку «Сбросить». Все параметры и настройки кроме настроек анимации будут сброшены.
21. Для сохранения картинки с графиком и гистограммой воспользуйтесь кнопкой «Сохранить» или пунктом системного меню (вызывается щелчком правой кнопки мыши по заголовку окна) «Сохранить как», в открывшемся диалоге выберите имя и тип (расширение) файла, нажмите «Сохранить». В случае ошибки во время сохранения вам будет показано сообщение об этом.
22. Для закрытия окна с вычислениями воспользуйтесь кнопкой «Закрыть» или кнопкой с крестиком в заголовке окна.
23. Для просмотра сведений о программе из начального диалогового окна выберите пункт системного меню (см п 2.21) «О программе»
24. В открывшемся окне предоставлена информация о приложении. Чтобы закрыть его воспользуйтесь кнопкой «Закрыть» или кнопкой с крестиком в заголовке окна.
25. Для завершения работы с приложением необходимо закрыть все его окна (при помощи кнопок «Закрыть» или кнопок с крестиком в заголовках окон).

# TODO: Текст программы и файлов заголовков с комментариями

# TODO: Рисунки с копиями экрана при работе программы

# Контрольный пример, сравнение результата с эталоном

Для проверки ДПФ были использованы тесты GoogleTest (gtest), а в качестве эталона взята функция, реализующая быстрое преобразование Фурье из библиотеки numpy для языка программирования python. Объект теста (NumpyComparisonTest – наследник ::testing::Test) при инициализации запускал генератор тестов (файл “the standard.py”, написанный на языке python 3.6) и открывал файл “test.txt”, в который производились записи при исполнении “the standard.py”. Затем для каждой из 100 пар строк (исходные данные и само преобразование, рассчитанное с помощью библиотечной функции numpy.fft.fft) производилось ДПФ и сравнение с эталоном. Исходный код тестов приведен ниже.

//

// pch.h

//

#define \_USE\_MATH\_DEFINES // for pi constant

#pragma once

// including gtest lib

#include "gtest/gtest.h"

//

// pch.cpp

//

#include "pch.h"

//

// test.cpp

//

#include "pch.h" //for gtest

#include <process.h> // for 'system'

#include <fstream> // for file reading

// for pi constant and

// trigonometry

#include <cmath>

/// <summary>

/// Class of the tests

/// </summary>

class NumpyComparisonTest : public ::testing::Test

{

/// <summary>

/// On initialization

/// </summary>

void SetUp() override {

genTest();

fin.open("test.txt");

}

/// <summary>

/// On destruction

/// </summary>

void TearDown() override {

fin.close();

}

public:

/// <summary>

/// vector of calculated values

/// </summary>

std::vector<double> calculated\_dft;

/// <summary>

/// vector of read dft values

/// </summary>

std::vector<double> true\_dft;

/// <summary>

/// Samples size

/// </summary>

size\_t N = 1000;

protected:

/// <summary>

/// input file stream

/// </summary>

std::ifstream fin;

/// <summary>

/// vector of read samples

/// </summary>

std::vector<double> samples;

/// <summary>

/// asks python file to generate the tests

/// </summary>

void genTest() {

system("python \"./the standard.py\"");

}

/// <summary>

/// DFT function that copied from the project

/// </summary>

/// <param name="samples">a vector of samples</param>

/// <param name="m">number of dft sample to calculate</param>

/// <returns>absolute value of the DFT m sample</returns>

double dft(int m) {

// real and imaginary parts

double re = 0, im = 0;

// for each sample

for (size\_t n = 0; n < N; n++) {

// summing

re += samples[n] \* cos(2 \* M\_PI \* m \* n / N);

im += samples[n] \* sin(-2 \* M\_PI \* m \* n / N);

}

// return the abs(X)

return sqrt(re \* re + im \* im);

}

/// <summary>

/// Reads 'n' numbers from the stream

/// </summary>

/// <param name="in">stream to read</param>

/// <param name="out">output vector</param>

/// <param name="n">amount of numbers</param>

void readTheTestLine(std::vector<double>& out) {

// reserving memory

out.resize(N);

// for each i = 0, 1, ..., n-1

for (int i = 0; i < N; i++) {

// read a number from the stream to the vector[i]

fin >> out[i];

}

}

/// <summary>

/// Calculates dft absolute values

/// </summary>

/// <param name="signal">a vector of source data</param>

/// <param name="out">a place for result</param>

void genDft() {

// reserve memory

calculated\_dft.resize(N);

// for each i = 0, 1, ..., N-1

for (int i = 0; i < N; i++) {

// calculate DFT sample and

// write it to the vector

calculated\_dft[i] = dft(i);

}

}

/// <summary>

/// Calculates one test

/// </summary>

void calculate\_one() {

readTheTestLine(samples);

readTheTestLine(true\_dft);

genDft();

}

};

// Test it

TEST\_F(NumpyComparisonTest, RandomTests) {

// or each of 100 tests

for (int i = 0; i < 100; i++) {

// calculate one

calculate\_one();

// comparison loop

for (int i = 0; i < N; i++) {

ASSERT\_NEAR(true\_dft[i], calculated\_dft[i], 1e-10);

}

}

}

#

# the standart.py

#

# including lib for dft

import numpy as np

# including lib for generating pseudo random numbers

from random import random as rnd

# define the function of 1 argument

def generate\_sequence(n):

"""

generates sequence of numbers in [0.0, 1.0)

with length 'n' and returns 'np.array' of them

"""

# result array

result = np.array([],dtype=np.double)

# reserving memory

result.resize(n)

# repeat for each i = 0, 1, 2, ..., n-1

for i in range(n):

# assign array item to random number

result[i] = rnd()

# return result array

return result

# define the function of 3 arguments

def generate\_test(test\_amount, n, filename):

"""

creates file 'filename' and writes to

it 'testamount'\*2 strings of 'n' elements

alternating source samples and resulting DFT

"""

# open the file

with open(filename, 'w') as f:

# for each i = 0, 1, 2, ..., 'test\_amount'-1

for i in range(test\_amount):

# get sequence of random numbers

s = generate\_sequence(n)

# convert numbers to the strings

# and write it to string with space separtion

string = ' '.join(map(str,s))

# write a line

f.write(string)

f.write('\n')

# calculate DFT using library Fast

# Fourier transform function

dft = np.absolute(np.fft.fft(s))

# convert numbers to the strings

# and write it to string with space separtion

string = ' '.join(map(str,dft))

# write a line

f.write(string)

f.write('\n')

# call the function

"""

create file 'text.txt' in current

path and write 100 numbers sequences

DFT's with length 1000 to it

"""

generate\_test(100,1000,'./test.txt')

Изображение выглядит как текст

Автоматически созданное описаниеТаким образом проверка преобразования была проведена на 100 случайных выборках, каждую из которых составляло 1000 чисел в диапазоне от 0 до 1. Снимок экрана после запуска тестов приведен ниже.

# TODO: Ведомость соответствия программы спецификации

# TODO: Выводы