|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ | | | | | | | | |
|  | |  |  | | |  |  | |
| Пермский государственный национальный  исследовательский университет | | | | | | | | |
|  |  | |  | | |  | |  |
|  | ОТЧЕТ  по лабораторной работе  «Алгоритм БПФ с прореживанием по частоте. Двоично-инверсная перестановка» | | | | | | |  |
|  |  | | |  |  | | |  |
|  | Работу выполнил студент гр. КМБ-19  Бузмаков Иван Григорьевич  12.04.2023 | | |  | Проверил  старший преподаватель кафедры радиоэлектроники и защиты информации  Манцуров Алексей Валерьевич | | |  |
|  |  | | |  |  | | |  |
| Пермь, 2022 г. | | | | | | | | |

Оглавление

[**Постановка задачи** 3](#_Toc132061201)

[**Теория** 3](#_Toc132061202)

[***Реализация - https://github.com/INDRA909/PhysicsLab*** 6](#_Toc132061203)

[***Результат выполнения(Вариант 2)*** 8](#_Toc132061204)

[**Вывод** 8](#_Toc132061205)

# **Постановка задачи**

Разработать алгоритм преобразующий последовательность вещественных чисел в символьную комбинацию, используя преобразование Фурье и другие физическо-математические преобразования.

# **Теория**

Алгоритм быстрого преобразования Фурье (БПФ) является важным инструментом для анализа и обработки сигналов различной природы. Он позволяет восстанавливать амплитуду и фазовый спектр сигнала в частотной области по выборке амплитуды во временной области.

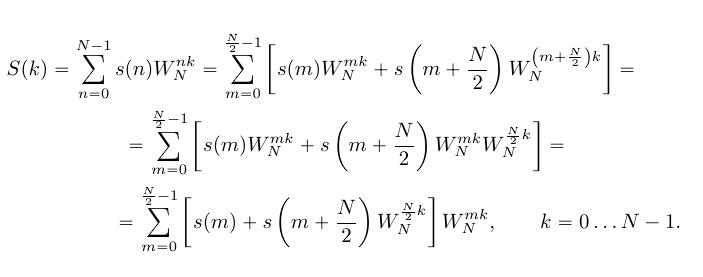
Пусть имеется N отсчетов входного сигнала s(n), n = 0 \ldots N-1, при этом N представляет собой целую степень двойки N = 2^L.

В алгоритме БПФ с прореживанием по времени производилось разделение исходного сигнала в соответствии с двоично-инверсной перестановкой.

Таким образом, мы получили первую и вторую половины дискретного преобразования Фурье (ДПФ).

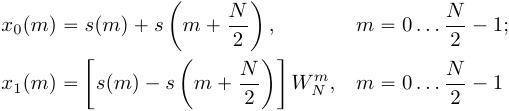
В алгоритме с прореживанием по частоте исходный сигнал s(n), n = 0 \ldots N-1, делится пополам, т.е. s_0(m) = s(m) и s_1(m) = s\left(m+\frac{N}{2}\right), m = 0 \ldots \frac{N}{2} - 1.

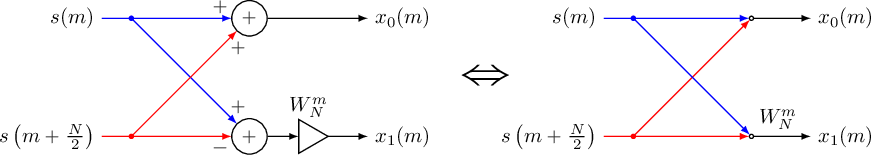
Тогда ДПФ сигнала s(n) может быть записано в виде:



*Для выполнения данной работы используется модификация операции бабочка в прореживанием по частоте.*

Процедура расчета сигналов половинной длительности





*Рисунок 1. Граф «бабочка» алгоритма БПФ с прореживанием по частоте*

Отличие графа «бабочка» алгоритма с прореживанием по частоте от аналогичного [графа для алгоритма с прореживанием по времени](https://ru.dsplib.org/content/fft_dec_in_time/fft_dec_in_time.html#r4)заключается в том, что умножение на поворотный коэффициент W_N^m производится после вычитания ветвей.

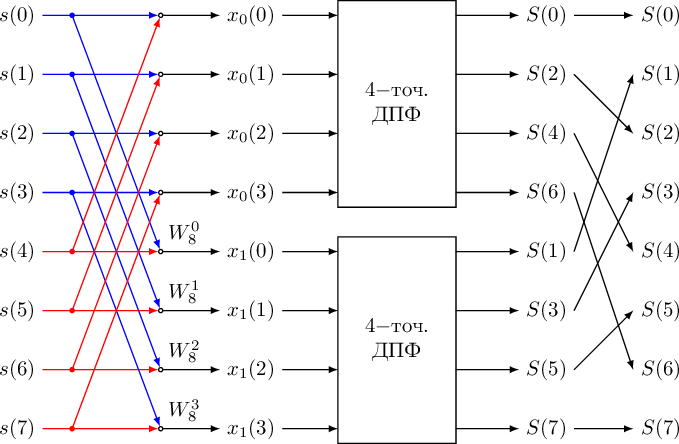


Рисунок 2. Граф алгоритма БПФ с прореживанием по частоте для N=8

При этом каждое из \frac{N}{2}-точечных ДПФ также может быть рассчитано с использованием алгоритма с прореживанием по частоте.

В результате получим полный граф алгоритма БПФ с прореживанием по частоте, как это показано следующем рисунке.

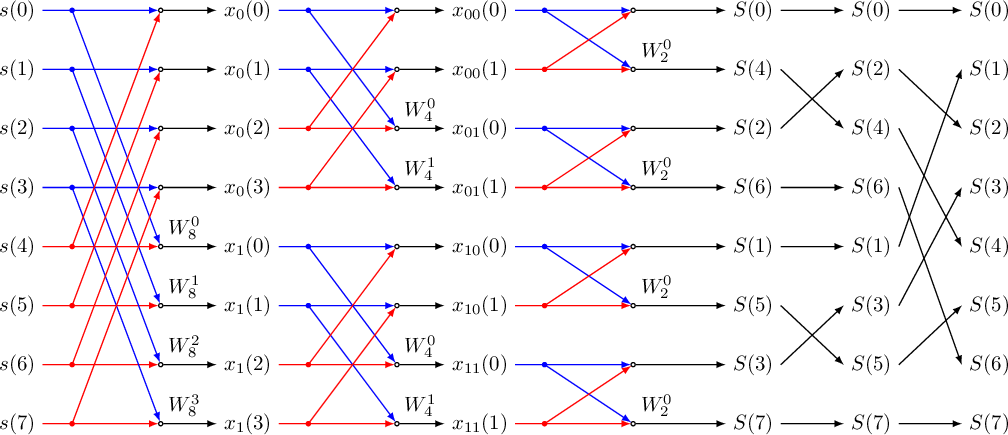
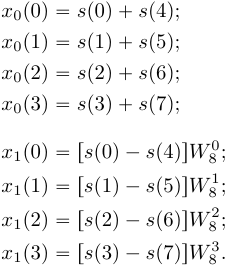
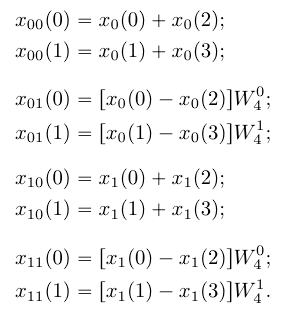


Рисунок 3. Полный граф алгоритма БПФ с прореживанием по частоте для N=8

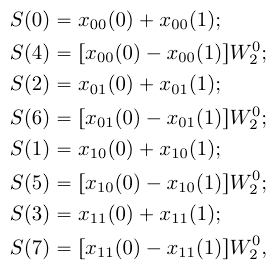
На первом этапе получаем x_0(m) и x_1(m):



Для расчета 4-точечных ДПФ сигналов x_0(m) и x_1(m), m = 0 \ldots 3, снова используем алгоритм с прореживанием по частоте. Тогда получим сигналы:



После расчета двухточечных ДПФ на последнем уровне разбиения получаем переставленные спектральные отсчеты:

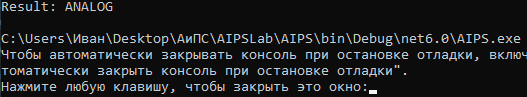


Эти спектры мы должны подвергнуть двоично-инверсной перестановке аналогично тому, как производится эта процедура в алгоритме с прореживанием по времени.

# ***Реализация -*** [***https://github.com/INDRA909/PhysicsLab***](https://github.com/INDRA909/PhysicsLab)

|  |
| --- |
| /// <summary>  /// Этот метод реализует прореживание в частотной области (Decimation in Frequency) при использовании Быстрого преобразования Фурье (FFT).  /// Он принимает в качестве параметров массив комплексных чисел "sample" и функцию "getOmega", возвращающую массив комплексных чисел для  /// вычисления отношения между элементами при прореживании.  /// Перестановка выполняется на каждом уровне рекурсии, когда массив делится на две части.  /// Эти части передаются в рекурсивные вызовы метода DecimationInFrequency, и на каждом уровне массив делится пополам и элементы переставляются  /// в соответствии с двоично-инверсной последовательностью.  /// Таким образом, метод DecimationInFrequency содержит неявную реализацию двоично-инверсной перестановки элементов входного массива.  /// </summary>  /// <param name="input"></param>  /// <param name="getOmega"></param>  private static void DecimationInFrequency(ref Complex[] input, Func<int, Complex[]> getOmega)  {  if (input.Length < 2) return;  var length = input.Length / 2;  //разбивает входной массив на две части и создает два новых массива  // для хранения результата преобразования Фурье каждой части.  var part1 = new Complex[length];  var part2 = new Complex[length];  var omaga = getOmega(length);  // Вычисляет преобразование Фурье для каждой из двух частей входного массива,  // используя поворотные множители, вычисленные заранее.  for (int i = 0, j = length; i < length; i++, j++)  {  var a = input[i];  var b = input[j];  part1[i] = a + b;  part2[i] = (a - b) \* omaga[i];  }  // Рекурсивный вызов функции для вычисления ДПФ для четных и нечетных элементов  DecimationInFrequency(ref part1, getOmega);  DecimationInFrequency(ref part2, getOmega);  // Собирает результаты преобразования Фурье из двух частей входного массива.  for (int i = 0, j = 0; i < length; i++) // j += 2  {  input[j++] = part1[i];  input[j++] = part2[i];  }  }  /// <summary>  /// Преобразование Фурье, нормализация  /// </summary>  /// <param name="sample"></param>  /// <returns></returns>  public static Complex[] GetFFTransform(this IEnumerable<Complex> sample)  {  var input = sample.ToArray();  var omega = DirectOmega ;  Complex[] getOmega(int length)  {  if (omega.TryGetValue(length, out var rotor))  return rotor;  lock (omega)  {  if (omega.TryGetValue(length, out rotor))  return rotor;  return omega[length] = GetOmega(length);  }  };  // Вычисляем преобразование Фурье  DecimationInFrequency(ref input, getOmega);    // Нормализуем результат  double normalizationFactor = input.Length / 2 ;  for (var i = 0; i < input.Length; i++)  input[i] /= normalizationFactor;  return input;  }  static readonly Dictionary<int, Complex[]> DirectOmega = new();  /// <summary>  /// Метод вычисляет и возвращает массив Complex чисел,  /// которые представляют собой корни из единицы - так называемые "омега-числа".  /// Используется для преобразования последовательности данных в спектральное представление.  /// </summary>  /// <param name="length"></param>  /// <returns></returns>  static Complex[] GetOmega(int length)  {  var abs = -(PI / length) ;  var omegaBase = new Complex(Cos(abs), Sin(abs));  var omega = new Complex[length];  for (var i = 0; i < length; i++)  omega[i] = Complex.Pow(omegaBase, i);  return omega;  }  /// <summary>  /// Метод ДПФ  /// </summary>  /// <param name="sample"></param>  /// <returns></returns>  public static Complex[] GetDFTTransform(this IEnumerable<Complex> sample)  {  var input = sample.ToArray();  int N = input.Length;  // Создаем массив для хранения результатов ДПФ  Complex[] X = new Complex[N];  // Вычисляем ДПФ для каждого элемента  for (int k = 0; k < N; k++)  {  Complex sum = 0;  // Вычисляем сумму для текущего элемента  for (int n = 0; n < N; n++)  {  //Для каждого элемента X[k] вычисляется сумма, которая состоит из произведения каждого элемента входного сигнала  //workSample[n] на поворотный множитель Complex.Exp(-2 \* Math.PI \* Complex.ImaginaryOne \* k \* n / N).  sum += input[n] \* Complex.Exp(-2 \* Math.PI \* Complex.ImaginaryOne \* k \* n / N);  }  X[k] = sum;  }  // Нормализуем результат  double normalizationFactor = N / 2;  for (var i = 0; i < X.Length; i++)  X[i] /= normalizationFactor;  return X;  }  }  public static void Main()  {  var signal = new StreamReader(@"C:\Users\Иван\Desktop\АиПС\Сигналы\2.txt")  .ReadToEnd()  .Trim()  .Split("\n")  .Select(sig => float.Parse(sig));  var signalComplex = signal.Select(sig => new Complex(sig, 0))  .ToArray();  var result = Butterfly.GetDFTTransform(signalComplex)  .Select(compl => Complex.Abs(compl))  .Take(7)  .Select(modulo => Math.Round(modulo));  var symbols = result.Select(modulo => (char)modulo);  var str = string.Join("", symbols);  Console.WriteLine("Result: " + str);  } |

# ***Результат выполнения(Вариант 2)***



# **Вывод**

Конечная цель лабораторной работы состояла в получении букв на основе амплитуд спектров вещественных чисел. Для этого было применено быстрое преобразование Фурье с прореживанием по частоте.

БПФ является одним из наиболее важных алгоритмов в обработке сигналов и анализе данных. БПФ может быть использован для анализа звуковых сигналов, изображений, видео, сигналов в телекоммуникациях и других приложений. БПФ работает за время O(N log N), что делает его значительно быстрее обычного преобразования Фурье, которое работает за время O(N^2). Он позволяет разложить сигнал на частотные составляющие и определить их амплитуды и фазы.

Прореживание по частоте является процессом, при котором производится уменьшение количества отсчетов в сигнале, что уменьшает объем данных и упрощает анализ.

Таким образом, данная лабораторная работа позволила получить буквы из амплитуд спектров вещественных чисел с помощью быстрого преобразования Фурье с прореживанием по частоте, двоично инверсной перестановкой и нормализацией.