СПб НИУ ИТМО

кафедра ИПМ

Методы цифровой обработки информации

Курсовая работа

Исследование влияния конечной разрядности АЛУ спецпроцессора на точность результатов при выполнении линейной фильтрации сигналов

Вариант 4

Работу выполнил

Студент 4 курса

Группы № P3418

Журавлев Виталий

Санкт-Петербург

2015 г.

**Цель работы:**

 Изучение влияния конечной разрядной сетки специализированного процессора при выполнении линейной фильтрации сигналов на точность формируемого результата.

**Задание:**

1) Разработать алгоритм и написать программу, реализующую заданный согласно номеру варианта алгоритм линейной фильтрации, полагая использование форматов чисел с плавающей точкой для исходных данных и получаемых результатов

2) Разработать алгоритм и написать программу, реализующую тот же алгоритм линейной фильтрации, полагая использование форматов чисел с фиксированной точкой для исходных данных и получаемых результатов с учетом заданной разрядности n1,n2,n3

 3) Использую полученные в результате этапа 1 программные средства для получения эталонных результатов

* построить зависимости среднеквадратичной погрешности от длины N обрабатываемого вектора данных и (или) длины ядра преобразования W;
* проанализировать влияние на точность формируемого результата способа формирования мало разрядного результата - с отсечением младших разрядов, с отсечением и увеличением младшего разряда на единицу, с округлением;
* построить зависимости СКО от длины обрабатываемого вектора для всех трех указанных способов округления;
* построить зависимости СКО от длины обрабатываемого вектора (при формировании результата с округлением) при изменении:

а) разрядности исходных данных на 2 и 4 бита при той же разрядности весовых множителей;

б) разрядности весовых множителей на 2 и 4 бита при неизменной разрядности исходных данных.

**Вариант:**

Реализовать алгоритм быстрого преобразования Фурье (БПФ) при N = 8 – 1024 n1 =10, n2 = 2, n3 = 16

**Код программы:**

public struct Complex : ICloneable, ISerializable

 {

 // Initializes a new instance

 public Complex( Complex c )

 {

 this.Re = c.Re;

 this.Im = c.Im;

 }

 // Initializes a new instance

 public Complex( double re, double im )

 {

 this.Re = re;

 this.Im = im;

 }

 public static readonly Complex Zero = new Complex( 0, 0 ); // A double-precision complex number that represents zero.

 public static readonly Complex One = new Complex( 1, 0 ); // A double-precision complex number that represents one.

 public static readonly Complex I = new Complex( 0, 1 ); // A double-precision complex number that represents the squere root of (-1).

 public double Re; // Real part of the complex number

 public double Im; //part of the complex number.

 // Phase value of the complex number.

 public double Phase

 {

 get { return System.Math.Atan2( Im, Re ); }

 }

 // Magnitude value of the complex number.

 public double Magnitude

 {

 get { return System.Math.Sqrt( Re \* Re + Im \* Im ); }

 }

 // Squared magnitude value of the complex number.

 public double SquaredMagnitude

 {

 get { return ( Re \* Re + Im \* Im ); }

 }

}

public struct Harmonic

 {

 public double Amplitude { get; set; }

 public double Frequency { get; set; }

 public double Phase { get; set; }

 public Harmonic(double amplitude, double frequency, double phase) : this()

 {

 Amplitude = amplitude;

 Frequency = frequency;

 Phase = phase;

 }

 }

// Set harmonics (multiples of frequencies of the low frequency) for which "decomposed" signal (function)

public class Spectrum

 {

 // Scaling the result of the Fourier transform in the frequency spectrum (frequency amplitude and phase)

 public Harmonic[] Get(Complex[] data, double time)

 {

 int n = (data.Length + 1) / 2;

 var spectr = new Harmonic[n];

 spectr[0].Amplitude = data[0].Re / data.Length; spectr[0].Frequency = 0; spectr[0].Phase = 0;

 for (int i = 1; i < n; i++)

 {

 spectr[i].Frequency = i / time;

 spectr[i].Amplitude = 2.0 \* data[i].Magnitude;

 spectr[i].Phase = data[i].Phase;

 spectr[i].Phase += Math.PI / 2;

 if (spectr[i].Phase > Math.PI) spectr[i].Phase -= 2 \* Math.PI;

 }

 return spectr;

 }

 };

 //Get Rotation of Complex

 private static Complex[] GetComplexRotation(int numberOfBits, Direction direction)

 {

 int directionIndex = (direction == Direction.Forward) ? 0 : 1;

 // Сheck if array already calculated

 if (complexRotation[numberOfBits - 1, directionIndex] == null)

 {

 int n = 1 << (numberOfBits - 1);

 double uR = 1.0;

 double uI = 0.0;

 double angle = System.Math.PI / n \* (int)direction;

 double wR = System.Math.Cos(angle);

 double wI = System.Math.Sin(angle);

 double t;

 Complex[] rotation = new Complex[n];

 for (int i = 0; i < n; i++)

 {

 rotation[i] = new Complex(uR, uI);

 t = uR \* wI + uI \* wR;

 uR = uR \* wR - uI \* wI;

 uI = t;

 }

 complexRotation[numberOfBits - 1, directionIndex] = rotation;

 }

 return complexRotation[numberOfBits - 1, directionIndex];

 }

 public static void FFT(Complex[] data, Direction direction)

 {

 int n = data.Length;

 int m = Tools.Log2(n);

 // Reorder data

 ReorderData(data);

 // Calculate FFT

 int tn = 1, tm;

 for (int k = 1; k <= m; k++)

 {

 Complex[] rotation = FourierTransform.GetComplexRotation(k, direction);

 tm = tn;

 tn <<= 1;

 for (int i = 0; i < tm; i++)

 {

 Complex t = rotation[i];

 for (int even = i; even < n; even += tn)

 {

 int odd = even + tm;

 Complex ce = data[even];

 Complex co = data[odd];

 double tr = co.Re \* t.Re - co.Im \* t.Im;

 double ti = co.Re \* t.Im + co.Im \* t.Re;

 data[even].Re += tr;

 data[even].Im += ti;

 data[odd].Re = ce.Re - tr;

 data[odd].Im = ce.Im - ti;

 }

 }

 }

 if (direction == Direction.Forward)

 {

 for (int i = 0; i < n; i++)

 {

 data[i].Re /= (double)n;

 data[i].Im /= (double)n;

 }

 }

 }

 // Data reordering

 private static void ReorderData(Complex[] data)

 {

 int len = data.Length;

 // Check length

 if ((len < minLength) || (len > maxLength) || (!Tools.IsPowerOf2(len)))

 throw new ArgumentException("Incorrect data length.");

 int[] rBits = GetReversedBits(Tools.Log2(len));

 for (int i = 0; i < len; i++)

 {

 int s = rBits[i];

 if (s > i)

 {

 Complex t = data[i];

 data[i] = data[s];

 data[s] = t;

 }

 }

 }

}

**Результаты выполнения программы:**

Разложение прямоугольного сигнала и обратное преобразование:



Из графиков видно, что прямоугольный сигнал раскладывается на бесконечное число гармоник, а при обратном преобразовании дает горбинки при переходе от 0 к 1 и от 1 к 0.

Используя эталонные результаты построим зависимости среднеквадратичного отклонения от длины обрабатываемых данных. Для этого используем аппроксимацию:

//Function approximation method

public double ApproximatingFunction(double x)

 {

 double result = 0;

 double[] coeff = (double[]) this.coeff.Clone();

 for (int k = 0; k < n; k++)

 {

 double mult = 1;

 for (int i = 0; i < k; i++)

 {

 mult \*= (x - X[i]);

 }

 result += mult\*coeff[k];

 }

 return result;

 }

// Calculate the coefficients

private double[] CalcCoefficients()

 {

 double[] result = new double[n];

 double[][] diff = FiniteDifferences();

 result[0] = Y[0];

 //calc

 for (int k = 1; k < n - 1; k++)

 {

 double fact = 1;

 for (int i = 1; i <= k; i++)

 {

 fact \*= i;

 }

 result[k] = diff[k][0]/Math.Pow(step, k)/fact;

 }

 return result;

 }

**Результаты преобразований при различных длинах исходного вектора:**



 8 16

  32 64



 128 256



 512 1024

При увеличении числа гармоник стенки прямоугольника все больше и больше становятся вертикальными. Однако, идеального прямоугольника добиться не получится, так как требуется бесконечно число гармоник.

**График среднеквадратичных отклонений от длины исходных данных.**



Заменим исходные данные и все вычисления на числа с фиксированной точкой.



Небольшое уменьшение погрешности при добавлении 1 к младшему разряду обусловлено тем, что получившиеся в результате обратного преобразования функции вписаны в прямоугольную функцию.

**Вывод:**

В ходе выполнения работы был реализован алгоритм быстрого преобразования Фурье (БПФ). Рассчитаны результаты его работы на исходных данных различной длины (8 – 1024). Для вычисления среднеквадратичной погрешности для нахождения промежуточных значений функций был реализован и использован метод аппроксимаций. Также произведены вычисления для чисел в формате с фиксированный точкой и расчет среднеквадратичного отклонения для них.