Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего профессионального образования
КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

Институт математики и механики им. Н.И. Лобачевского
Кафедра теории функции и приближений

Направление: 010100 – математика
Профиль: действительный анализ

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

(Бакалаврская работа)

Решение задачи калибровки датчиков с помощью аппроксимации сплайнами на основе параллельных алгоритмов.

Работа завершена:

«____» _____2014 г. _____ Нугманов Р. А.

Работа допущена к защите:

Научный руководитель
ст. преподаватель каф. ТФиП
«____» _____2014 г. _____Маклецов С.В.

Заведующий кафедрой:

доктор физико-математических наук,
профессор каф. ТФиП
«____» _____2014 г. _____Авхадиев Ф.Г.

КАЗАНЬ — 2014

Оглавление

Введение 3

1.Аппроксимация и интерполяция таблично-заданных функций 4

2.Параллельные вычисления 9

3.Программная реализация 17

Заключение 21

Список используемой литературы 22

Приложение 1 24

Приложение 2 34

Введение

Согласно статистическим данным, около 40% всех производимых в промышленности измерений приходятся на температурные, а в таких отраслях, как энергетика, их число достигает 70% от общего количества измерений. Значимость температурных измерений постоянно возрастает, что объясняется, в частности, развитием новых перспективных научных направлений, например, нанотехнологии, требующих точного контроля температуры. Температура является критическим параметром в системах управления тепловыми режимами космических аппаратов и системах теплового контроля ядерных энергетических установок.[1]

Наиболее востребованными из современных высокоточных средств измерения температуры являются контактные электрические датчики температуры. Повышение точности измерения температуры с их помощью стало возможным не только благодаря усовершенствованию технологий изготовления самих датчиков температуры, но и благодаря возможности применения цифровых методов компенсации методических погрешностей.[1]

Целью данной выпускной квалификационной работы является освоение задачи интерполяции сплайном 1-й степени, создание на его основе алгоритма и его распараллеливание.

Для реализации поставленной цели необходимо решить следующую задачу для двух данных датчиков температуры (эталонного и калибруемого) необходимо построить функцию (аппроксимирующую) для точного определения температуры на калибруемом датчике. Для этого их необходимо поместить их в однородную термическую среду и сделать несколько замеров при разной температуре. В итоге получим массив данных, в котором будет храниться температура и показания калибруемого датчика. Затем по этим точкам с помощью интерполирования мы можем построить нужную нам функцию.

1.Аппроксимация и интерполяция таблично-заданных функций

При работе с научными и инженерными расчетами, часто приходится работать с большими массивами данных, полученных опытным путем. Как правило, необходимо как-то упорядочить эти данные, то есть построить функцию, на которую могли бы с высокой точностью попадать другие получаемые значения. Эта задача называется аппроксимацией.[2]

Аппроксимация, или **приближение** — научный метод, состоящий в замене одних объектов другими, в том или ином смысле близкими к исходным, но более простыми [3].

Аппроксимация позволяет исследовать числовые характеристики и качественные свойства объекта, сводя задачу к изучению более простых или более удобных объектов (например, таких, характеристики которых легко вычисляются, или свойства которых уже известны).

Другими словами, аппроксимация функций заключается в приближенной замене заданной функции $f(x)$ некоторой функцией φ(x), с расчетом, чтобы отклонение между этими функциями в заданной области было минимально. Функция φ(x) при этом называется аппроксимирующей. Одной из обычных задач аппроксимации является интерполяция. Ее использование в основном связано с двумя следующими причинами:

1)Заданная функция имеет сложное аналитическое описание, которое вызывает определенные трудности.

2)Описание заданной функции неизвестно, то есть она задана в виде таблицы. При этом необходимо иметь приближенное аналитическое представление $f(x)$ (например, для вычисления в произвольных точках).[4]

Интерполяция, **интерполирование** — в вычислительной математике способ нахождения промежуточных значений величины по имеющемуся дискретному набору известных значений.[2]

Простейшая задача интерполяции состоит в следующем. На отрезке [a,b] заданы n+1 точки $x_i = x_0, x_1, \dots, x_n$, которые называются *узлами интерполяции*, и значение некоторой функции $f(x)$ в этих точках

$$f(x_0) = y_0, f(x_1) = y_1, \dots, f(x_n) = y_n.$$

Требуется построить функцию $\Phi(x)$, принадлежащую известному классу и принимающую в узлах интерполяции те же значения, что и $f(x)$, т.е. такую, что

$$\Phi(x_0) = y_0, \Phi(x_1) = y_1, \dots, \Phi(x_n) = y_n.$$

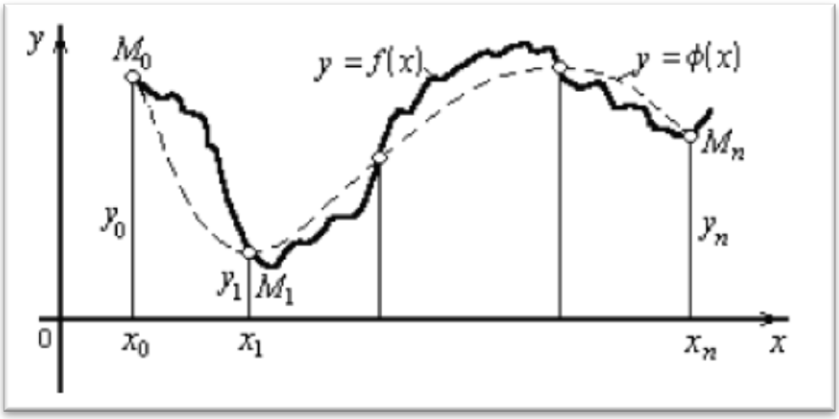


Рисунок 1. Геометрическая интерпретация

Геометрически это означает, что нужно найти кривую некоторого определенного типа, которая проходит через заданные точки. (Рисунок 1)

В такой обобщенной постановке наша задача может иметь бесчисленное множество решений или не иметь их вовсе.

Однако задача становится, однозначно разрешима, если в качестве произвольной функции $\Phi(x)$ взять полином $\varphi(x)$ степени не выше n, удовлетворяющий условию совпадения в узлах, т.е. такой, что

$$\varphi(x_0) = y_0, \varphi(x_1) = y_1, \dots, \varphi(x_n) = y_n.$$

Полученную формулу интерполяции

$$\varphi(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$$

обычно используют для приближенного вычисления значений, отличных от узлов интерполяции.

Таким образом, мы пришли к интерполяции алгебраическими полиномами, в качестве которых можно брать интерполяционные многочлены в форме Лагранжа или Ньютона.

К сожалению, есть существенное ограничение многочленной аппроксимации: если функция, подлежащая аппроксимации, имеет особенности в некоторых точках на интервале аппроксимации, то она плохо аппроксимируется на всем интервале. Этой общей зависимости от локальных свойств можно избежать, если использовать кусочно-многочленные аппроксимирующие функции.[5]

Поэтому в данной работе мы будем использовать сплайн-интерполяцию.

Суть сплайн-интерполяции заключается в том, что на каждом промежутке между узловыми точками мы будем осуществлять аппроксимацию в виде определенной полиноминальной зависимости $f(x)$. При этом на каждом шаге мы получим свой полином, причем его коэффициенты подбираются так, чтобы в узлах выполнялись условия сшивки. А именно, если применять сплайны в виде полиномов степени m, то их коэффициенты можно выбрать так, чтобы обеспечить непрерывность производных порядка (m-1)-й включительно.

Дадим точное определение сплайна.

Непрерывная функция g:[a,b]→R называется сплайном, если существует разбиение a= x_0<x_1<x_2<…<x_n=b такое, что на каждом частичном отрезке $[x_{k-1}, x_k]$ функция g является полиномом. Таким образом, ограничение $g|_{[x_{k-1},x_k]}$ является полиномом, для простоты мы обозначим его как $g_k[x_{k-1}, x_k]$ →R.

Функция $g(x) = S_n^m(f; x)$ является интерполяционным сплайном для f, степени $m \geq 1$, если выполняются следующие условия: [6]

1) g(x) непрерывна на [a,b], а на каждом частичном отрезке $[x_{k-1}, x_k]$ $g(x) = g_k(x)$,

где $g_k(x)$ – некоторый полином степени $\leq m$, т.е. имеет вид

$$g_k(x) = \sum_{j=0}^{m} a_{kj} x^j ;$$

2) для каждого узла x_j $(j = 0, \dots, n)$

$$g(x_j) = f(x_j);$$

3) если $m \geq 2$, то $g \in C^{(m-1)}[a, b]$.

Сплайны представляют собой удобный аппарат приближения функций конечной гладкости. В данной работе мы будем рассматривать сплайны порядка 1, т.к. они являются одни из наиболее употребительных на практике.

Рассмотрим сплайн 1-й степени $g(x) = S_n^1(f; x)$ для

$$f \in C[a, b], \text{ a= } x_0 < x_1 < x_2 < \ldots < x_n = \text{b}.$$

Интерполяционным сплайном будет функция $g \in C[a, b]$, $g(x_k) = f(x_k)$, $k = 0, \dots, n$, кроме того, на любом частичном отрезке $[x_{k-1}, x_k]$

$$g(x) = g_k(x) = a_k x + b_k.$$

Таким образом, речь идет об аппроксимации $f \in C[a, b]$, т.е. непрерывными, кусочно-линейными функциями.

Рассмотрим на практике линейную и кусочно-линейную интерполяцию.

Самый простой вид интерполяции – кусочно-постоянная. Её смысл состоит в том, что на каждом промежутке между экспериментальными точками функция представляет собой константу.

Минус такого подхода заключается в том, что ни полученная интерполяция, ни ее производная не являются непрерывными функциями.

Чуть сложнее выглядит кусочно-линейная интерполяция, которая представляет искомую зависимость в виде ломаной линии.

Коэффициенты находятся из формулы:

$$\mathrm{a_k} = \frac{f(x_k) - f(x_{k+1})}{x_k - x_{k+1}},$$

$$b_\mathrm{k} = f(x_k) - \mathrm{a_k} x_k.$$

Отметим также представление типа Лагранжа для сплайнов первой степени:

$$\mathrm{S_n^1(f; x)} = \sum_{\mathrm{j=0}}^{n} \mathrm{f(x_j)s_j}\,(x),$$

где $\mathrm{s_j}(x)$ – фундаментальные сплайны первой степени со стандартным свойством $\mathrm{s_j}(x_k) = \delta_{kj}$. Мы модем написать их в явном виде. Для крайних узлов

$$s_0(x) = \begin{cases} \frac{x_1 - x}{x_1 - a} \text{ при } a \le x \le x_1, \\ 0 \text{ при } x_1 \le x \le b; \end{cases}$$

$$s_n(x) = \begin{cases} 0 \text{ при } a \le x \le x_{n-1}, \\ \frac{x - x_{n-1}}{b - x_{n-1}} \text{ при } x_{n-1} \le x \le b; \end{cases}$$

и при любом $1 \le \mathrm{j} \le \mathrm{n} - 1$, т.е. для внутренних узлов

$$s_j(x) = \begin{cases} 0 \text{ при } a \le x \le x_{j-1}, \\ \frac{x - x_{j-1}}{x_j - x_{j-1}} \text{ при } x_{j-1} \le x \le x_j, \\ \frac{x_{j+1} - x}{x_{j+1} - x_j} \text{ при } x_j \le x \le x_{j+1}, \\ 0 \text{ при } x_{j+1} \le x \le b. \end{cases}$$

2.Параллельные вычисления.

Параллельные вычисления — такой способ организации компьютерных вычислений, при котором программы разрабатываются как набор взаимодействующих вычислительных процессов, работающих параллельно (одновременно). Термин охватывает совокупность вопросов параллелизма в программировании, а также создание эффективно действующих аппаратных реализаций. Теория параллельных вычислений составляет раздел прикладной теории алгоритмов.[7]

Существуют различные способы реализации параллельных вычислений. Например, каждый вычислительный процесс может быть реализован в виде процесса операционной системы, либо же вычислительные процессы могут представлять собой набор потоков выполнения внутри одного процесса ОС. Параллельные программы могут физически исполняться либо последовательно на единственном процессоре — перемежая по очереди шаги выполнения каждого вычислительного процесса, либо параллельно — выделяя каждому вычислительному процессу один или несколько процессоров (находящихся рядом или распределённых в компьютерную сеть).

Основная сложность при проектировании параллельных программ — обеспечить правильную последовательность взаимодействий между различными вычислительными процессами, а также координацию ресурсов, разделяемых между процессами.

В некоторых параллельных системах программирования передача данных между компонентами скрыта от программиста, тогда как в других она должна указываться явно. Явные взаимодействия могут быть разделены на два типа.

Взаимодействие через разделяемую память: на каждом процессоре мультипроцессорной системы запускается поток исполнения, который принадлежит одному процессу. Потоки обмениваются данными через общий

для данного процесса участок памяти. Количество потоков соответствует количеству процессоров. Потоки создаются либо средствами языка (например, в Java или C#), либо с помощью библиотек явно (например, в C/C++ с помощью PThreads), либо декларативно (например, с помощью библиотеки OpenMP), или автоматически встроенными средствами компилятора (например, High Performance Fortran). Данный вид параллельного программирования обычно требует какой-то формы захвата управления (мьютексы, семафоры, мониторы) для координации потоков между собой.

Взаимодействие с помощью передачи сообщений: на каждом процессоре многопроцессорной системы запускается однопоточный процесс, который обменивается данными с другими процессами, работающими на других процессорах, с помощью сообщений. Процессы создаются явно, путем вызова соответствующей функции операционной системы, а обмен сообщениями — с помощью библиотеки (например, реализация протокола MPI), или с помощью средств языка (например, High Performance Fortran, Erlang или occam). Обмен сообщениями может происходить асинхронно, либо с использованием метода «рандеву», при котором отправитель блокирован до тех пор, пока его сообщение не будет доставлено. Асинхронная передача сообщений может быть надёжной (с гарантией доставки) либо ненадёжной.

Параллельные системы, основанные на обмене сообщениями, зачастую более просты для понимания, чем системы с разделяемой памятью, и обычно рассматриваются как более совершенный метод параллельного программирования. Существует большой выбор математических теорий для изучения и анализа систем с передачей сообщений, включая модель акторов и различные виды исчислений процессов. Обмен сообщениями может быть эффективно реализован на симметричных мультипроцессорах как с разделяемой когерентной памятью, так и без неё.

У параллелизма с распределенной памятью и с передачей сообщений разные характеристики производительности. Обычно (но не всегда), накладные расходы памяти на процесс и времени на переключение задач у систем с передачей сообщений ниже, однако передача самих сообщений более накладна, чем вызовы процедур. Эти различия часто перекрываются другими факторами, влияющими на производительность.

Также есть спобсоб, который объединяет в себе два вышеперечисленных методы взаимодействия -- гибридный способ: на многопроцессорных системах с распределённой памятью (DM-MIMD), где каждый узел системы представляет собой мультипроцессор с общей памятью (SM-MIMD). В нем можно применить гибридный метод программирования. На каждом узле системы запускается многопоточный процесс, который распределяет потоки между процессорами данного узла. Обмен данными между потоками на узле осуществляется через общую память, а обмен данными между узлами — через передачу сообщений. В этом случае количество процессов определяется количеством узлов, а количество потоков — количеством процессоров на каждом узле. Гибридный способ программирования более сложен (требуется особым образом переписывать параллельную программу), но наиболее эффективен в использовании аппаратных ресурсов каждого узла многопроцессорной системы.

Разумеется в такой системе можно также использовать и исключительно метод передачи сообщений, то есть запустить на каждом процессоре каждого узла отдельный процесс. В этом случае количество процессов (и потоков) будет равно количеству процессоров на всех узлах. Этот способ проще (в параллельной программе надо только увеличить количество процессов), но является менее эффективным, так как процессоры одного и того же узла будут обмениваться друг с другом сообщениями, словно они находятся на разных машинах.[8]

Рассмотрим также классификацию алгоритмов по типу параллелизма. Способность алгоритма к распараллеливанию потенциально связана с одним из двух (или одновременно с обоими) внутренних свойств, которые характеризуются как параллелизм задач (message passing) и параллелизм данных (data parallel). [9]

Если алгоритм основан на параллелизме задач, вычислительная задача разбивается на несколько, относительно самостоятельных подзадач, каждая из которых загружается в "свой" процессор. Каждая подзадача реализуется независимо, но использует общие данные и/или обменивается результатами своей работы с другими подзадачами. Для реализации такого алгоритма на многопроцессорной системе необходимо выявлять независимые подзадачи, которые могут выполняться параллельно. Часто это оказывается далеко не очевидной и весьма трудной задачей.

При наличии в алгоритме свойства параллелизма данных, одна операция может выполняться сразу над всеми элементами массива данных. В этом случае различные фрагменты массива могут обрабатываться независимо на разных процессорах.

Рассмотрим в кратце библиотеки и методы, реализующие данные параллельные вычисления.

1. OpenMP (Open Multi-Processing) - это набор директив компилятора, библиотечных процедур и переменных окружения, которые предназначены для программирования многопоточных приложений на многопроцессорных системах с общей памятью (SMP-системах).[10]

Первый стандарт OpenMP был разработан в 1997 г. как API, ориентированный на написание легко переносимых многопоточных приложений. Сначала он был основан на языке Fortran, но позднее включил в себя и языки Си и Си++.

Интерфейс OpenMP стал одной из наиболее популярных технологий параллельного программирования. OpenMP успешно используется как при программировании суперкомпьютерных систем с большим количеством процессоров, так и в настольных пользовательских системах.

В OpenMP используется модель параллельного выполнения "ветвление-слияние". Программа OpenMP начинается как единственный поток выполнения, называемый начальным потоком. Когда поток встречает параллельную конструкцию, он создает новую группу потоков, состоящую из себя и некоторого числа дополнительных потоков, и становится главным в новой группе. Все члены новой группы (включая главный) выполняют код внутри параллельной конструкции. В конце параллельной конструкции имеется неявный барьер. После параллельной конструкции выполнение пользовательского кода продолжает только главный поток. В параллельный регион могут быть вложены другие параллельные регионы, в которых каждый поток первоначального региона становится основным для своей группы потоков. Вложенные регионы могут в свою очередь включать регионы более глубокого уровня вложенности.

На данный момент технология OpenMP поддерживается большинством компиляторов языка Си/Си++. Несколько хуже дело обстоит с инструментами тестирования параллельных OpenMP программ. Инструменты анализа, проверки и оптимизации параллельных программ хотя и существуют давно, до недавнего времени были мало востребованы при разработке прикладного программного обеспечения. Поэтому они часто являются менее удобными, чем иные инструментальные средства разработки.

2. MPI - это библиотека передачи сообщений, собрание функций на C/C++ (или подпрограмм в Фортране), облегчающих коммуникацию (обмен данными и синхронизацию задач) между процессами параллельной программы с распределенной памятью. Акроним MPI установлен для Message Passing Interface (интерфейс передачи сообщений). MPI на данный

момент является фактическим стандартом и самой развитой переносимой библиотекой параллельного программирования с передачей сообщений.[11]

Несмотря на то, что MPI-программы показывают высокий уровень производительности, сама технология имеет ряд недостатков:

- низкий уровень (программирование на MPI часто сравнивают с программированием на ассемблере), необходимость детального управления распределением массивов и витков циклов между процессами, а также обменом сообщениями между процессами – все это приводит к высокой трудоемкости разработки программ;
- необходимость избыточной спецификации типов данных в передаваемых сообщениях, а так же наличие жестких ограничений на типы передаваемых данных;
- сложность написания программ, способных выполняться при произвольных размерах массивов и произвольном количестве процессов – делает практически невозможным повторное использование имеющихся MPI-программ;
- отсутствие поддержки объектно-ориентированного подхода.

Возможно, некоторые из перечисленных недостатков и послужили причиной отсутствия поддержки интерфейса MPI в такой современной платформе разработки, как .NET Framework.

3. Технология CUDA — это программно-аппаратная вычислительная архитектура NVIDIA, основанная на расширении языка Си, которая даёт возможность организации доступа к набору инструкций графического ускорителя и управления его памятью при организации параллельных вычислений. CUDA помогает реализовывать алгоритмы, выполнимые на графических процессорах видеоускорителей GeForce восьмого поколения и старше (серии GeForce 8, GeForce 9, GeForce 200), а также Quadro и Tesla.

Хотя трудоёмкость программирования GPU при помощи CUDA довольно велика, она ниже, чем с ранними GPGPU решениями. Такие программы требуют разбиения приложения между несколькими мультипроцессорами подобно MPI программированию, но без разделения данных, которые хранятся в общей видеопамяти. И так как CUDA программирование для каждого мультипроцессора подобно OpenMP программированию, оно требует хорошего понимания организации памяти. Один из недостатков CUDA - слабая переносимость. Эта архитектура работает только на видеочипах этой компании, да ещё и не на всех, а начиная с серии GeForce 8 и 9 и соответствующих Quadro, ION и Tesla. NVIDIA приводит цифру в 90 миллионов CUDA-совместимых видеочипов.

4. C++/CLI — привязка языка программирования C++ к среде программирования .NET фирмы Microsoft. Она интегрирует C++ стандарта ISO с Объединённой Системой Типов (Unified Type System, UTS), рассматриваемой как часть Общей Языковой Инфраструктуры (Common Language Infrastructure, CLI). Она поддерживает и исходный уровень, и функциональную совместимость исполняемых файлов, скомпилированных с родного и управляемого C++. C++/CLI представляет собой дальнейшее развитие C++.[12]

При выборе программных средств распараллеливания для решаемой задачи необходимо учитывать следующие критерии: платформонезависимость, эффективность и распространенность (наличие реализаций для различных классов систем).

Поэтому мы выбрали язык высокого уровня C++/CLI, являющийся расширением языка C++ для платформы .NET, а в качестве среды разработки Microsoft Visual Studio Ultimate 2013.

.NET Framework — это фреймворк для создания приложений под ОС Windows, Windows Phone, Windows Server и Windows Azure. Она состоит из среды CLR и библиотеки классов .NET Framework, которая содержит классы,

интерфейсы и типы значений, поддерживающие широкий диапазон технологий. Платформа .NET Framework предоставляет управляемую среду выполнения, которая упрощает разработку и развертывание приложений, а также возможности интеграции с различными языками программирования, включая Visual Basic и Visual C#.[13]

Рассмотрим процесс распараллеливания, применительно к нашей задаче. В качестве примера возьмем отрезок [0,40], разобъем его на n равных частей и на каждом промежутке строим полином первой степени. Приведем фрагмент диаграммы:

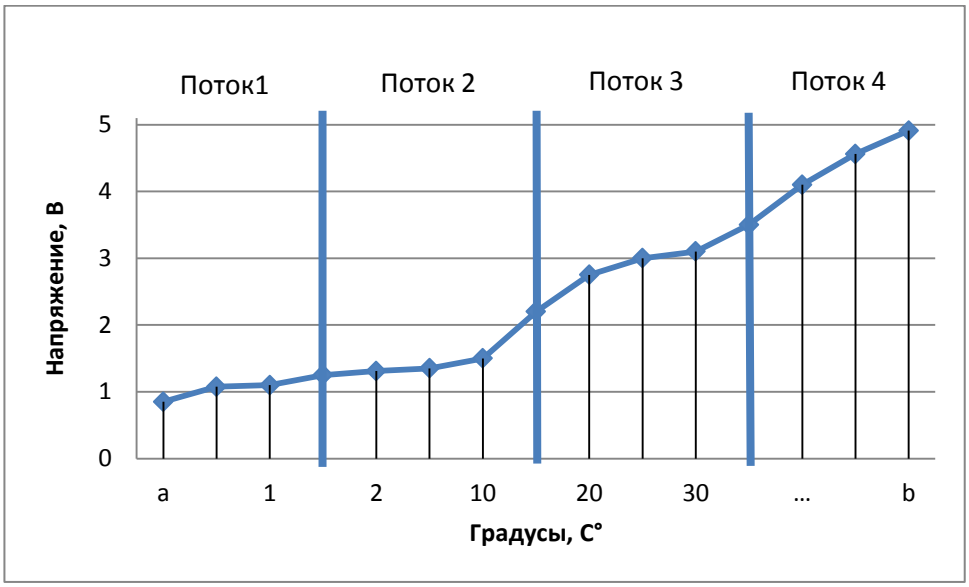


Рисунок 2. Построение сплайна с использованием многопоточного алгоритма.

Так как построение двух соседних многочленов независимы, мы можем выполнять это действие в параллельном режиме. Для этого поделим весь промежуток на 4 части и на каждой их них будем выполнять последовательные вычисления. На первом сегменте будем высчитывать полиномы в точках $x_{0,1}$, $x_{0,2}$, …, $x_{0,n}$. Следующие узлы необходимо взять с таким расчетом, чтобы $x_{0,n}$=$x_{1,0}$ и т.д. В итоге получим четыре потока, которые и будем запускать одновременно.

3.Программная реализация

В нашем проекте программный код расположен в трех файлах: «Form.cpp», «inter.cpp» и «main.cpp».

В первом файле содержится описание ссылочного класса:

public ref class Form1 : public System::Windows::Forms::Form, являющимся наследником от стандартного окна windows. В нем содержится графический интерфейс нашей программы.

В следующем файле находится непосредственно класс для создания сплайнов: ref class MyClass, основными полями которого являются:

array<MyStruct>^ arrst; − массив для узлов;

array<MyStruct>^ coeff; − массив для коэффициентов (последовательный алгоритм);

array<MyStruct>^ coeff1; − массив для коэффициентов (параллельный алгоритм), где

MyStruct, структура вида

```
value struct MyStruct {
        double x;
        double y;
    };
```

Далее идут методы этого класса, рассмотри для начала public-методы:

```
MyClass(String^ path);
MyClass();
void drw(Windows::Forms::TextBox^ myBox);
void drw1(Windows::Forms::TextBox^ myBox);
```

Первый их них – конструктор. В качестве аргумента он принимает «путь» файла с данными, которые представляют собой числа типа float, причем четное количество. Первое значение -- это температура эталонного датчика, а второе – напряжение на калибруемом датчике. Исходные данные расположены так, что на каждой строке только одно значение.

Далее идет деструктор, а за ним два метода для вывода сплайна. Первый из них реализует вывод сплайна, который строится последоватьльно,а второй – параллельно. В качестве аргумента они принимают TextBox, т.е. поле для вывода.

Перейдем к рассмотрению методов, отвечающих за непосредственное построение сплайнов, здесь содержатся три private-метода:

```
void crtSpl();
void crtSplParal();
void crtSplParal(int part);
```

Первый из них реализует построение сплайна обычным последовательным методом, который был описан в пункте 1, т.е в цикле:

```
for (int j = 0; j < (len-1);j++){
    coeff[j].x = (arrst[j].y - arrst[j + 1].y) / (arrst[j].x - arrst[j + 1].x);
    coeff[j].y = arrst[j].y - coeff[j].x * arrst[j].x;
}.
```

Напомним что между двумя узлами, мы строим прямую, которая задается двумя коэффициентами a и b. В данном цикле мы вначале находим первый коэффициет, затем подставляя его, находим второй. Мы видим что на каждом промежутке выполняем одни и те же действия, причем результат следующего не зависит от предыдущего, это наводит мысль на распараллеливание данной задачи.

Метод void crtSplParal(); реализует построение сплайна в параллельном режиме и метод void crtSplParal(int part) высчитывает полиномы на частичных промежутках.

В первом методе мы используем следующую конструкцию:

Parallel::For(0, parts, gcnew System::Action<int>(this, &MyClass::crtSplParal)), где parts – количество частей, на которые мы разбили последовательный процесс.

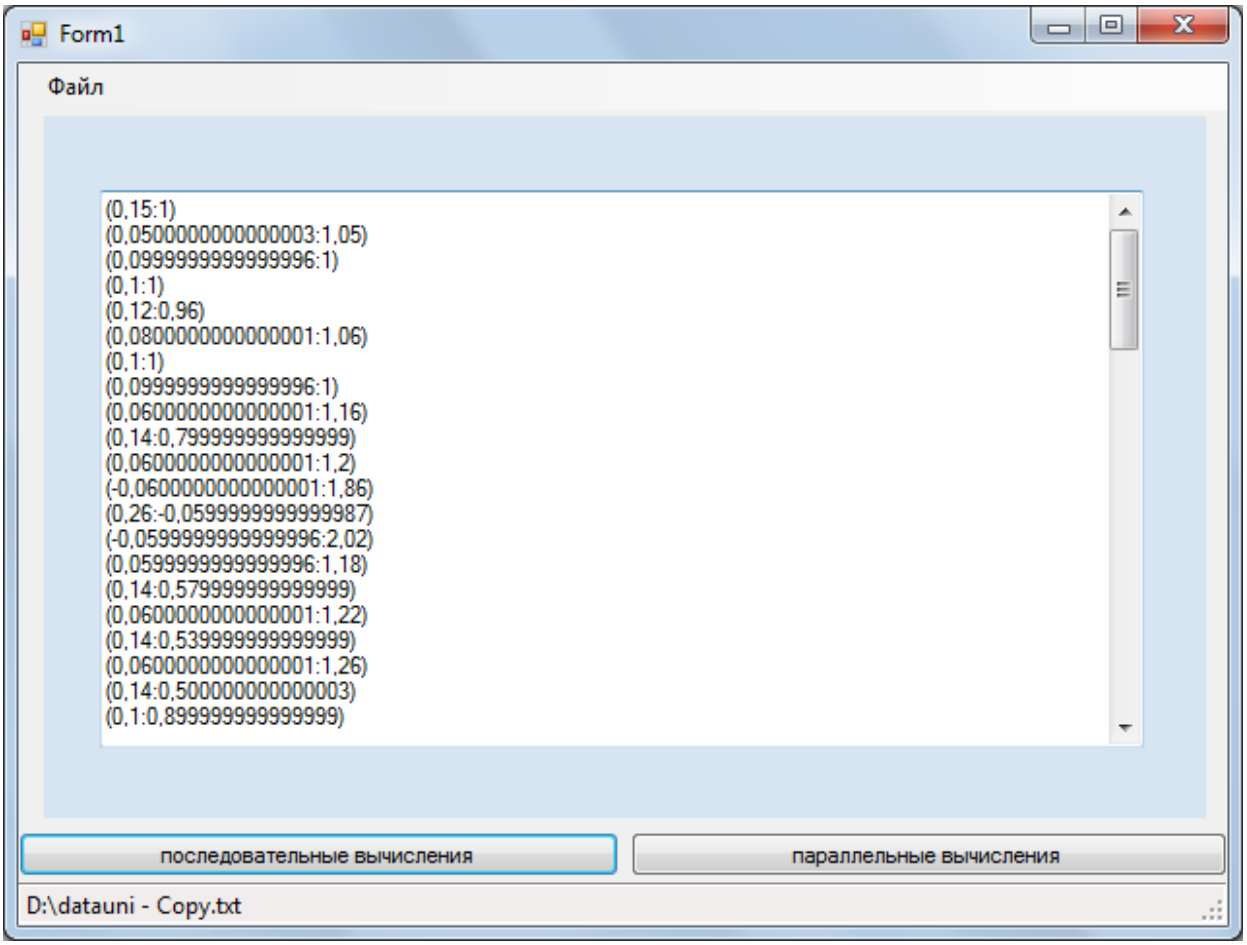
В методе void crtSplParal(int part); мы описываем алгоритм нахождения полиномов в каждой части. Входным параметром для данного метода является переменная part – номер части, далее высчитывает количество элементов в одной части *int partSize = (int)((len - 1) / parts);*, остаток элементов, которые нельзя поровну разделить между частями

*int ost = (len - 1) - partSize*parts;*, стартовые и конечные индексы *int st = (part*partSize) + ((part < ost) ? part : ost) + 1;*, *int fin = ((part+1)*partSize) + (((part+1) < ost) ? (part+1) : ost);*, эти индексы как раз таки и задают границы массива, которые мы просчитывает в данной части.

В итоге получим распараллеливание на нужное нам количество частей. Учтем также что вычисления буду производится на двухъядерном процессоре, поэтому оптимальное количество потоков будет равно 4.

И наконец в последнем файле «main.cpp» производится настройка графики и создание экземпляра окна.

Рассмотрим интерфейс программы:



Сверху имеется выпадающее меню, для выбора нужного файла.В центре рабочая область, куда выводятся результаты работы.Снизу две кнопки для выполнения последовательного или параллельного алгоритма и строка состояния, в которой отображается путь текущего файла.

Тестирование программы выполнялось на компьютере с двухядерным процессором Pentium® Dual-Core T4500 @2.3 Ггц. При количестве узлов равным 10000 были получены следующие результаты: последовательный способ вычисления занял 6,067с., а параллельный – 5,143с., что составляет разницу в 15%.

Заключение

В настоящей выпускной квалификационной работе была рассмотрена задача калибровки датчиков температуры, как наиболее распространненых, а также был предложен алгоритм их калибровки с использованием спалайн-интерполяции.

Подробно была рассмотрена теория аппроксимации и интерполяции с использованием многочленов и сплайнов, их преимущества и недостатки.

Также приведен обзор существующих на сегодняший день технологий параллельных вычислений, а также библиотек, которые их реализуют(OpenMP, MPI, NVIDIA CUDA, C++/CLI). Дан полный анализ по каждой технологии, оценены их плюсы и минусы.

Была написана программа на языке высокого уровня C++/CLI, реализующая алгоритм каллибровки как последовательный, так и параллельный. Даны сравнительные результаты работы этих алгоритмов.

Данный метод и программа, реализующая его, имеет большое прикладное значение, так как может применяться для калибровки не только датчиков температуры, но любых других сходных по физическим параметрам датчиков.

Список используемой литературы

1. Гуреев В.В Математические модели контактных датчиков температуры и цифровые методы компенсации их методических погрешностей : автореф.дис. …канд. техн. наук : 05.13.18. – Саратов.,2008. – 111л.
2. Интерполяция//Википедия-свободная энциклопедия. URL: http://ru.wikipedia.org/wiki/%C8%ED%F2%E5%F0%EF%EE%EB%FF%F6%E8%FF
3. Аппроксимация//Википедия-свободная энциклопедия. URL: http://ru.wikipedia.org/wiki/%D0%90%D0%BF%D0%BF%D1%80%D0%BE%D0%BA%D1%81%D0%B8%D0%BC%D0%B0%D1%86%D0%B8%D1%8F
4. Ханова А.А. Интерполяция функций. Методическое пособие. – Астрахань: Астраханский государственный технический университет, 2001. – 22с.
5. Де Бор К. Практическое руководство по сплайнам: Пер. с англ. – М. : Радио и связь, 1985. – 304с., ил.
6. Авхадиев Ф. Г. Численные методы анализа. − Казань: Казанский (Приволжский) федеральный университет, 2013. − 126 с.
7. Параллельные вычисления//Википедия-свободная энциклопедия. URL: http://ru.wikipedia.org/wiki/%CF%E0%F0%E0%EB%EB%E5%EB%FC%ED%FB%E5_%E2%FB%F7%E8%F1%EB%E5%ED%E8%FF
8. Воеводин В. В., Воеводин Вл. В. Параллельные вычисления. − СПб: БХВ-Петербург, 2002. − 608 с.
9. Гергель В.П. Лекции по параллельным вычислениям: учебн. пособие. – Самара: Изд-во Самар. гос. аэрокосм. ун-та, 2009. – 164 с.
10. OpenMP//Электронный ресурс. URL: https://software.intel.com/ru-ru/articles/About-OpenMP

11. Оленев Н.Н Основы параллельного программирования в системе MPI. – М.: РАН Вычислительный центр им. Дородницына, 2005. – 81 с.

12. C++/CLI //Википедия-свободная энциклопедия. URL: http://ru.wikipedia.org/wiki/C++/CLI

13. .Net Framework//Электронный ресурс. URL: http://msdn.microsoft.com/ru-ru/library/w0x726c2(v=vs.110).aspx

Приложение 1

Исходный код программы:

«Form1.h»

```
#pragma once
#include "inter.h"
namespace Form1Namespace {

	using namespace System;
	using namespace System::ComponentModel;
	using namespace System::Collections;
	using namespace System::Windows::Forms;
	using namespace System::Data;
	using namespace System::Drawing;

	/// <summary>
	/// Сводка для MyForm
	/// </summary>
	public ref class Form1 : public System::Windows::Forms::Form
	{
	public:
		Form1(void)
		{
			InitializeComponent();
			//
			//TODO: добавьте код конструктора
			//
		}

	protected:
		/// <summary>
		/// Освободить все используемые ресурсы.
		/// </summary>
		~Form1()
		{
			if (components)
			{
				delete components;
			}
		}
	private: System::Windows::Forms::MenuStrip^  menuStrip1;
	protected:
	private:	System::Windows::Forms::ToolStripMenuItem^
файлToolStripMenuItem;
	private:	System::Windows::Forms::ToolStripMenuItem^
открытьToolStripMenuItem;
	private: System::Windows::Forms::OpenFileDialog^  openFileDialog1;
	private: System::Windows::Forms::Button^  button1;
```

```
private: System::Windows::Forms::StatusStrip^  statusStrip1;
private: System::Windows::Forms::ToolStripStatusLabel^  toolStripStatusLabel1;
private: System::Windows::Forms::Panel^  panel1;
private: System::Windows::Forms::Button^  button2;
private: System::Windows::Forms::TextBox^  textBox1;


private:
        /// <summary>
        /// Требуется переменная конструктора.
        /// </summary>
        System::ComponentModel::Container ^components;
        MyClass^ my;
        Diagnostics::Stopwatch^ sw = gcnew Diagnostics::Stopwatch();

                Diagnostics::Stopwatch^ sw1 = gcnew Diagnostics::Stopwatch();
    #pragma region Windows Form Designer generated code
        /// <summary>
        /// Обязательный метод для поддержки конструктора - не изменяйте
        /// содержимое данного метода при помощи редактора кода.
        /// </summary>
        void InitializeComponent(void)
        {
                this->menuStrip1 = (gcnew System::Windows::Forms::MenuStrip());
                this->файлToolStripMenuItem = (gcnew System::Windows::Forms::ToolStripMenuItem());
                this->открытьToolStripMenuItem = (gcnew System::Windows::Forms::ToolStripMenuItem());
                this->openFileDialog1 = (gcnew System::Windows::Forms::OpenFileDialog());
                this->button1 = (gcnew System::Windows::Forms::Button());
                this->statusStrip1 = (gcnew System::Windows::Forms::StatusStrip());
                this->toolStripStatusLabel1 = (gcnew System::Windows::Forms::ToolStripStatusLabel());
                this->panel1 = (gcnew System::Windows::Forms::Panel());
                this->textBox1 = (gcnew System::Windows::Forms::TextBox());
                this->button2 = (gcnew System::Windows::Forms::Button());
                this->menuStrip1->SuspendLayout();
                this->statusStrip1->SuspendLayout();
                this->panel1->SuspendLayout();
                this->SuspendLayout();
                //
                // menuStrip1
                //
                this->menuStrip1->Items->AddRange(gcnew cli::array< System::Windows::Forms::ToolStripItem^  >(1) { this->файлToolStripMenuItem });
                this->menuStrip1->Location = System::Drawing::Point(0, 0);
```

```
this->menuStrip1->Name = L"menuStrip1";
this->menuStrip1->Size = System::Drawing::Size(624, 24);
this->menuStrip1->TabIndex = 0;
this->menuStrip1->Text = L"menuStrip1";
//
// файлToolStripMenuItem
//
this->файлToolStripMenuItem->DropDownItems-
>AddRange(gcnew cli::array< System::Windows::Forms::ToolStripItem^ >(1) { this-
>открытьToolStripMenuItem });
this->файлToolStripMenuItem->Name =
L"файлToolStripMenuItem";
this->файлToolStripMenuItem->Size = System::Drawing::Size(48,
20);
this->файлToolStripMenuItem->Text = L"Файл";
//
// открытьToolStripMenuItem
//
this->открытьToolStripMenuItem->Name =
L"открытьToolStripMenuItem";
this->открытьToolStripMenuItem->Size =
System::Drawing::Size(121, 22);
this->открытьToolStripMenuItem->Text = L"Открыть";
this->открытьToolStripMenuItem->Click += gcnew
System::EventHandler(this, &Form1::открытьToolStripMenuItem_Click);
//
// openFileDialog1
//
this->openFileDialog1->FileName = L"openFileDialog1";
this->openFileDialog1->FileOk += gcnew
System::ComponentModel::CancelEventHandler(this, &Form1::openFileDialog1_FileOk);
//
// button1
//
this->button1->AutoSize = true;
this->button1->Location = System::Drawing::Point(0, 394);
this->button1->Name = L"button1";
this->button1->Size = System::Drawing::Size(310, 23);
this->button1->TabIndex = 1;
this->button1->Text = L"последовательные вычисления";
this->button1->UseVisualStyleBackColor = true;
this->button1->Click += gcnew System::EventHandler(this,
&Form1::button1_Click);
//
// statusStrip1
//
this->statusStrip1->Items->AddRange(gcnew cli::array<
System::Windows::Forms::ToolStripItem^ >(1) { this->toolStripStatusLabel1 });
this->statusStrip1->Location = System::Drawing::Point(0, 420);
```

```
this->statusStrip1->Name = L"statusStrip1";
this->statusStrip1->Size = System::Drawing::Size(624, 22);
this->statusStrip1->TabIndex = 3;
this->statusStrip1->Text = L"Выбирите файл...";
//
// toolStripStatusLabel1
//
this->toolStripStatusLabel1->Name = L"toolStripStatusLabel1";
this->toolStripStatusLabel1->Size = System::Drawing::Size(103, 17);
this->toolStripStatusLabel1->Text = L"Выбирите файл...";
this->toolStripStatusLabel1->Click += gcnew
System::EventHandler(this, &Form1::toolStripStatusLabel1_Click);
//
// panel1
//
this->panel1->BackColor =
System::Drawing::SystemColors::GradientInactiveCaption;
this->panel1->Controls->Add(this->textBox1);
this->panel1->Location = System::Drawing::Point(13, 27);
this->panel1->Name = L"panel1";
this->panel1->Size = System::Drawing::Size(600, 360);
this->panel1->TabIndex = 4;
this->panel1->Paint += gcnew
System::Windows::Forms::PaintEventHandler(this, &Form1::panel1_Paint);
//
// textBox1
//
this->textBox1->Location = System::Drawing::Point(29, 38);
this->textBox1->Multiline = true;
this->textBox1->Name = L"textBox1";
this->textBox1->ScrollBars =
System::Windows::Forms::ScrollBars::Vertical;
this->textBox1->Size = System::Drawing::Size(539, 286);
this->textBox1->TabIndex = 0;
//
// button2
//
this->button2->Location = System::Drawing::Point(316, 394);
this->button2->Name = L"button2";
this->button2->Size = System::Drawing::Size(308, 23);
this->button2->TabIndex = 5;
this->button2->Text = L"параллельные вычисления";
this->button2->UseVisualStyleBackColor = true;
this->button2->Click += gcnew System::EventHandler(this,
&Form1::button2_Click);
//
// Form1
//
this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
```

```
this->AutoScaleMode =
System::Windows::Forms::AutoScaleMode::Font;
this->ClientSize = System::Drawing::Size(624, 442);
this->Controls->Add(this->button2);
this->Controls->Add(this->panel1);
this->Controls->Add(this->button1);
this->Controls->Add(this->menuStrip1);
this->Controls->Add(this->statusStrip1);
this->MainMenuStrip = this->menuStrip1;
this->Name = L"Form1";
this->Text = L"Form1";
this->menuStrip1->ResumeLayout(false);
this->menuStrip1->PerformLayout();
this->statusStrip1->ResumeLayout(false);
this->statusStrip1->PerformLayout();
this->panel1->ResumeLayout(false);
this->panel1->PerformLayout();
this->ResumeLayout(false);
this->PerformLayout();

}
#pragma endregion
private: System::Void открытьToolStripMenuItem_Click(System::Object^
sender, System::EventArgs^ e) {
openFileDialog1->ShowDialog();

}
private: System::Void openFileDialog1_FileOk(System::Object^ sender,
System::ComponentModel::CancelEventArgs^ e) {
this->toolStripStatusLabel1->Text=openFileDialog1-
>FileName;
}
private: System::Void label1_Click(System::Object^ sender, System::EventArgs^
e) {

}
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e) {
my = gcnew MyClass(openFileDialog1->FileName);
sw->Start();
my->drw(textBox1);
sw->Stop();
textBox1->Text += "Затраченное время:" + sw-
>ElapsedMilliseconds + "мс" + Environment::NewLine;


}
private: System::Void toolStripStatusLabel1_Click(System::Object^ sender,
System::EventArgs^ e) {
}
```

```
private: System::Void panel1_Paint(System::Object^ sender,
System::Windows::Forms::PaintEventArgs^ e) {

}
private: System::Void button2_Click(System::Object^ sender, System::EventArgs^ e) {
textBox1->Clear();
sw1->Start();
my->drw1(textBox1);
sw1->Stop();
textBox1->Text += "Затраченное время:" + sw1-
>ElapsedMilliseconds + "мс" + Environment::NewLine;
}
};
}
```

«inter.h»

```
#pragma once
using namespace System;
using namespace System::Collections;
using namespace System::IO;

ref class MyClass
{
	value struct MyStruct {
		double x;
		double y;
	};
	StreamReader^ sr;
	String^ line;
	array< Double >^ local = gcnew array< Double >(2004);
	array<MyStruct>^ arrst;
	array<MyStruct>^ coeff;
	array<MyStruct>^ coeff1;
	int len,i;
	int parts = 100;

public:
	MyClass(String^ path);
	~MyClass();
	void drw(Windows::Forms::TextBox^ myBox);
	void drw1(Windows::Forms::TextBox^ myBox);
private:
	void crtSpl();
	void crtSplParal();
	void crtSplParal(int part);

};
```

«inter.cpp»

```
#include "inter.h"
using namespace System;
using namespace System::Collections;
using namespace System::IO;
using namespace System::Threading;
using namespace System::Threading::Tasks;
MyClass::MyClass(System::String^ path)
{
	sr = File::OpenText(path);
	i = 0;
	while (sr->Peek() >= 0)
	{
		line = sr->ReadLine();
		local[i] = Double::Parse(line);
		i++;
	}
	len = i / 2;

}
MyClass::~MyClass()
{
}
void MyClass::crtSpl()
{
	arrst = gcnew array<MyStruct>(len);
	coeff = gcnew array<MyStruct>(len - 1);

	int l = 0;
	for (int k = 0; k < i; k+=2)
	{
		arrst[l].x = local[k];
		arrst[l].y = local[k + 1];
		l++;
	}
	for (int j = 0; j < (len-1);j++){
		coeff[j].x = (arrst[j].y - arrst[j + 1].y) / (arrst[j].x - arrst[j + 1].x);
		coeff[j].y = arrst[j].y - coeff[j].x * arrst[j].x;
	}
}

void MyClass::crtSplParal(){

	coeff1 = gcnew array<MyStruct>(len - 1);
	Parallel::For(0, parts, gcnew System::Action<int>(this, &MyClass::crtSplParal));

}
void MyClass::crtSplParal(int part){
```

```
        int partSize = (int)((len - 1) / parts);
        int ost = (len - 1) - partSize*parts;
        int st = (part*partSize) + ((part < ost) ? part : ost) + 1;
        int fin = ((part+1)*partSize) + (((part+1) < ost) ? (part+1) : ost);
        for (int j = st; j < fin; j++){
                coeff1[j].x = (arrst[j].y - arrst[j + 1].y) / (arrst[j].x - arrst[j + 1].x);
                coeff1[j].y = coeff1[j].x* arrst[j].x;
        }
}

void MyClass::drw(Windows::Forms::TextBox^ myBox){
        MyClass::crtSpl();
        String^ s;
        for (int i = 0; i < len-1; i++)
        {
                s = "(" + coeff[i].x + ":" + coeff[i].y + ")";
                myBox->Text += s + Environment::NewLine;
        }
        myBox->Text += "(Последовательное)Узлов:" + len + Environment::NewLine;
}

void MyClass::drw1(Windows::Forms::TextBox^ myBox){
        MyClass::crtSplParal();
        String^ s;
        for (int i = 0; i < len - 1; i++)
        {
                s = "(" + coeff1[i].x + ":" + coeff1[i].y + ")";
                myBox->Text += s + Environment::NewLine;
        }
        myBox->Text += "(Параллельное)Узлов:" + len + Environment::NewLine;
}
```

«main.cpp»

```
#include "Form1.h"
#include "inter.h"
using namespace Form1Namespace;
[STAThreadAttribute]
int main()
{
   using namespace System;
   using namespace System::Windows::Forms;

   Application::EnableVisualStyles();
   Application::SetCompatibleTextRenderingDefault(false);
   Application::Run(gcnew Form1());
}
```

Приложение 2

Исходные данные:

Температура/значение	Температура/значение
0,0 1,0	20,0 2,9
0,5 1,075	20,5 3,0
1,0 1,1	21,0 3,1
1,5 1,15	21,5 3,15
2,0 1,2	22,0 3,2
2,5 1,26	22,5 3,26
3,0 1,30	23,0 3,30
3,5 1,35	23,5 3,35
4,0 1,4	24,0 3,4
4,5 1,43	24,5 3,43
5,0 1,5	25,0 3,5
5,5 1,53	25,5 3,53
6,0 1,5	26,0 3,5
6,5 1,63	26,5 3,63
7,0 1,6	27,0 3,6
7,5 1,63	27,5 3,63
8,0 1,7	28,0 3,7
8,5 1,73	28,5 3,73
9,0 1,8	29,0 3,8
9,5 1,83	29,5 3,83
10,0 1,9	30,0 3,9
10,5 1,95	30,5 4,01
11,0 2,1	31,5 4,15
11,5 2,15	32,0 4,22
12,0 2,2	32,5 4,25
12,5 2,26	33,0 4,33
13,0 2,30	33,5 4,38
13,5 2,35	34,0 4,4
14,0 2,4	34,5 4,43
14,5 2,43	35,0 4,48
15,0 2,5	35,5 4,5
15,5 2,53	36,0 4,52
16,0 2,5	36,5 4,55
16,5 2,63	37,0 4,63
17,0 2,6	37,5 4,7
17,5 2,63	38,0 4,75
18,0 2,7	38,5 4,79
18,5 2,73	39,0 4,83
19,0 2,8	39,5 4,91
19,5 2,83	40,0 4,95