ФЕДЕРАЛЬНОЕ АГЕНСТВО ПО ОБРАЗОВАНИЮ РФ

ГОУ ВПО «УДМУРТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет информационных технологий и вычислительной техники

**Курсовая работа**

**Визуализация понятия предела функции в точке**

**Выполнил:**

Студент 2 курса

Группы 38-21

Пискарев Николай Сергеевич

**Научный руководитель:**

Новикова Елена Вениаминовна

Ижевск 2011

**Содержание**

1. Введение 3
2. Предел функции 4
3. О программе 5
4. Структура программы 9
5. Примеры 26
6. Заключение 28
7. Литература 29

**Глава 1. Введение.**

Предел функции является основой высшей математики, однако достаточно непрост для студента-первокурсника. От правильной интерпретации этого понятия зависит успешное усвоение дальнейшего материала по многим «точным» предметам. Именно для визуализации (наглядности) понятия предела функции в точке и предназначена программа GraphicConstructor.

**Глава 2. Предел функции.**

Определение (по Коши, на языке ε-δ): Число А называется пределом функции f(x) в точке х0 (при х→ х0), если для любого ε> 0 существует такое δ, зависящее от ε, что для всех х, принадлежащих проколотой окрестности точки х0, удовлетворяющих условию <δвыполняется <ε, т.е. <=> ∀ε> 0 ∃δ(ε) > 0 ∀ x: 0 <<δ: <ε.

Определение (по Гейне, на языке последовательности): Число А называется пределом функции f(x) в точке x0, если ∀{xn} ⊂V{x0}\{x0}: xnх0: f(x)A.

Определение: f(x) непрерывна в точке x0, если

1. = f(x0);
2. ∀ε> 0 ∃δ(ε) > 0 ∀ x: <δ: <ε;
3. ∀xn х0при n→∞: f(xn)f(х0);
4. f(x0 + 0) = f(x0 – 0) = f(x0);
5. == 0.

Примеры:

1. f(x) = x2 + 2x – 5, x0 = 2.

∀ε> 0 ∃δ(ε) > 0 ∀x: |x – 2| <δ: |x2 + 2x – 5 – 3| <ε. Пусть ε> 0, рассмотрим следующее выражение: = ( + 4). Пусть функция определена в окрестности радиуса точки х = 2, тогда и . Тогда ( + 4) <<<ε =>δ =min().

1. f(x) = sin(), x0 = .

∀ε> 0 ∃δ(ε) > 0 ∀ x:<δ:<ε. = 2 =>=>δ = ε.

**Глава 3. О программе**.

1. Перед началом работы необходимо объяснить структуру интерфеса программы GraphicConstructor (прил. 1):

− в окно «f(x)»(окна, в которые пользователь вводит значения, подписаны слева) вводится функция с переменной х (различные возможности задания функции описаны в окне, появляющемся по нажатию на кнопку «О программе»).

− в окна «От» и «До» вводятся границы отрезка построения графика (вводиться могут не только цифры, но и числа , и операции с ними).

− в окно «r» вводится значение : именно с таким шагом по оси абсцисс будет строиться график.

− в окно «y0» вводится координата по оси ординат расположения -трубки.

− в окнах «Дельта-трубки», «Все значения», «Минимальной» будут отображаться координаты и вычисленные значения ширин -трубок и минимальная ширина из представленных соответственно.

− кнопка «О программе» выполняет роль справки.

− кнопка «Построить» строит график функции (при условии, что все значения введены корректно, иначе возникнет сообщение об ошибке).

− кнопка «Обновить» уменьшает разбиение в 100 раз, что позволяет наиболее полно представлять поведение графика в некоторой окрестности.

− кнопка «Показать» строит трубки.

− кнопка «Очистить холст» стирает все графики и все трубки с панели (далее холст), на которой отображаются графики и трубки (в коде программы она называется zedGraphControl1).

− галочка в окне «с точками» учитывается при построении графика. Если она есть, то на холсте отобразится не только сам график, но и точки, по которым он строился, в противном случае будет виден лишь сам график (прил. 2.1 – 2.2).

− пункт меню в верхнем левом углу окна программы «График» позволяет по усмотрению пользователя показывать или нет сетку (рис. 3), – оси координат (прил. 2.1 – 2.2).

− пункт в меню «Фон» раскрашивает фон позади рамки графика.

2. Итак, строим график функции f(x) = sqrt(abs(sin(1/x))). Получаем результат (см. прил. 2). По умолчанию после построения графика на холсте отобразится область графика в квадрате 1х1 с началом координат в нижнем левом углу. Для того, чтобы посмотреть весь график воспользуемся колёсиком мышки. Если нужно увеличить некоторую область на холсте, то удерживая нажатой левую клавишу мыши выделяем необходимую область, после отпускаем клавишу и облать увеличится.

3. Если пользователь захочет сохранить изображение графика, то следует поместить курсор мышки в холст и нажать правую кнопку, «выпадет» подменю и при нажатии на «Копировать» в буфер сохранится изображение графика (потом можно будет вставить в какой-нибудь графический редактор или в Microsoft Word).

4. Чтобы построить трубки необходимо нажать на кнопку «Построить». После чего выждать необходимое для пересчета время (судить по зелёному бегунку загрузки в левом нижнем углу окна программы) в окнах «Дельта-трубки», «Все значения», «Минимальной» появятся вычисленные значения координат по оси абсцисс месторасположения левых границ -трубок, все вычисленные их ширины и минимальная ширина соответственно. Для того, чтобы построились трубки, необходимо навести курсор мышки на холст (он изменит свой вид на крест) и нажать левую клавишу. Трубки построятся (рис. 4). После можно анализировать результаты в окнах «Дельта-трубки», «Все значения», «Минимальной», увеличивать и рассматривать трубки (прил. 5).

5. В окне «Выделенной» отображается ширина конкретной выделенной трубки в окне «Дельта-трубки» (прил. 6).

6. Если понадобится сохранить изображение, то жмём правую кнопку мыши по холсту и выбираем либо «Копировать», либо «Сохранить изображение как…» (что позволит сохранить холст с изображением в различных графических форматах). Также можно уменьшать масштаб видимой области, нажатием на кнопку «Отмена масштабирования» (если до этого увеличивали). Нажатие на «Масштаб в начальное состояние» вернёт квадрат со стороной 1 и левым нижним углом в начале координат.

7. Если кликнуть правой кнопкой по окну «Дельта-трубки» и нажать на «Показать таблицу», то возникнет сообщение, демонстрирующее ширины всех найденных -трубок (прил. 8).

8. При клике по окну ввода функции «f(x)» правой кнопкой мышки и нажатии на «Предыдущие значение», можно посмотреть значения, либо введённые за текущий сеанс работы программы, либо значения по умолчанию, а нажатие на «Очистить значения» удалит всё из этого выпадающего списка (прил. 9). Чтобы значение сохранилось в этом списке необходимо просто нажать левую кнопку мыши на окне «f(x)» и если текст функции выделился, то, значит, данная функция записалась. Чтобы закрыть этот подсписок, нужно кликнуть по нему левой кнопкой мыши (или нажать на появившуюся кнопку «Закрыть»), причем выделившаяся функция заменит предыдущую функцию в окне ввода функции.

11. Чтобы продемонстрировать функциональность кнопки «Обновить» достаточно сравнить изображения до и после нажатия на кнопку (прил. 10.1-10.2). Заметим, что программа в определённых случаях строит -трубки с уменьшенным в 100 раз значением , поэтому если кажется, что трубка построена неправильно, следует нажать на кнопку «Обновить» и, скорей всего, перерисовка уточнит координаты графика и ошибка исчезнет (прил. 11: синий график – до нажатия на кнопку «Обновить», тёмно-синий – после нажатия).

12. Если нужно простроить ещё график, то вводим новую функцию в соответствующее окно ввода, если необходимо – меняем отрезок, шаг построения или ординату , и нажимаем на «Построить». Следует отметить, что в случае нажатия на кнопку «Показать», программа будет искать пересечения с последним построенным графиком (прил. 12).

13. Если необходимо очистить холст, то нажимаем соответствующую кнопку и возникнет предупреждение, предостерегающее от случайного нажатия (прил. 13).

14. В проблемных случаях, когда число -трубок велико или стремится к бесконечности возникает окно-сообщение (прил. 14), которое ограничивает количество трубок до 50 и рисует их (прил. 15).

Предупреждение: старайтесь не копировать в окно «f(x)» выражения, набранные в программе Microsoft Word или других текстовых редакторах (за исключением Блокнота (Notepad)), поскольку существует вероятность, что Вы скопируете не только само выражение, но и специальные невидимые программные символы. Программа GraphicConstructor проверяет корректность введённого (вставленного) выражения и подобные программные символы могут вызвать ошибку. Старайтесь вводить выражения непосредственно в окно «f(x)», минуя другие текстовые редакторы.

**Глава 4. Структура программы.**

Программа состоит из следующих компонентов: Form1.cs, Parser.cs, Memessage.cs, ZedGraph.dll.

1. Компонент Form1.cs является корневым, использующим все остальные компоненты. Он содержит обработку событий ввода данных (значений) в текстовые окна (в программе они обозначаются как textBox), нажатия на кнопки и многие другие события.

Структурно Form1.cs состоит из следующего списка объектов, переменных и методов:

*List<List<double>> listDlt = new List<List<double>>();*

Список списков listDlt предназначен для записи в него значений окрестностей и точек, содержащихся в этих окрестностях, образуемых пересечением эпсило-трубки с графиком функции.

*List<double> listFirstElem = new List<double>();*

В список list FirstElem записываются точки входа в левые крайние значения -трубки). Он необходим для подсчета количества обнаруженных -трубок и нахождения наименьшей из найденных ширин -трубок.

Примечание: в этот список записываются левые крайние значения дельта-трубок не зависимо от того, находится ли -трубка в положительной части оси абсцисс или в отрицательной её части. Это связано с особенностями библиотеки ZedGraph, о которой я расскажу чуть позже.

*List<double> listWidth = new List<double>();*

Список для значений ширин дельта-трубок.

*public Form1()*

Данный метод (точнее конструктор) инициализирует значения по умолчанию окон-ввода, определяет форматирование текста, устанавливает разделитель целой и дробной части числа в значение «,», обнуляет счетчик ошибок и дезактивирует кнопку «Очистить».

*private void textBoxDbeg\_Click(object sender, EventArgs e)*

*private void textBoxDend\_Click(object sender, EventArgs e)*

*private void textBoxEpsStep\_Click(object sender, EventArgs e)*

*private void textBoxY0\_Click(object sender, EventArgs e)*

Данные четыре метода обрабатывают событие нажатие на окна-ввода «От» (textBoxDbeg), «До» (textBoxDend), «r» (textBoxEpsStep), «y0» (textBoxY0) соответственно.

*private void textBoxDbeg\_KeyPress(object sender, KeyPressEventArgs e)*

*private void textBoxDend\_KeyPress(object sender, KeyPressEventArgs e)*

В этих методах обрабатывается событие нажатие клавиш клавиатуры при условии, что фокус ввода находится либо в окне ввода «От» (textBoxDbeg), «До» (textBoxDend). Пользователю разрешено вводить в эти окна:

цифры (в том числе и константы и *е* в буквенном виде);

знаки математических операций (в частности: сложение «+», вычитание «-», умножение «\*», деление «/», возведение в степень «^», скобки «(» и «)»).

разделитель: запятая «,» (причем лишь один раз, что естественно).

*private void textBoxEpsStep\_KeyPress(object sender, KeyPressEventArgs e)*

*private void textBoxY0\_KeyPress(object sender, KeyPressEventArgs e)*

Эти методы аналогичны предыдущим двум, различие заключается лишь в том, что в окна ввода «r» (textBoxEpsStep) и «y0» (textBoxY0) нельзя вводить знаки математических операций.

*private void textBox\_Function\_Click(object sender, EventArgs e)*

Обработка события нажатия на окно ввода функции «f(x)» (textBox\_Function). При нажатии на это окно выделяется функция.

*public double Dbeg*

*public double Dend*

*public double y0*

*public double eps\_step*

Метод Dbeg возвращает значение левого края отрезка, задаваемого пользователем. Если значение не было введено и окно останется «пустым», то в метод вернёт 0.

Метод Dend возвращает значение правого края отрезка, задаваемого пользователем. Если значение не было введено, то в метод вернёт 0.

Метод y0 возвращает значение точки, из которой строится -трубка. Если значение не было введено и окно останется «пустым», то в метод вернёт 0.

Метод eps\_step возвращает значение , введённое пользователем или заданное по умолчанию. Если значение не было введено, то в метод вернёт 0.00.

*public string func*

Метод func возвращает строку, содержащую текст введённой пользователем функции в окно «f(x)» (textBox\_Function). Если значение не было введено, то в метод вернёт пустую строку.

*private void Build\_button\_Click(object sender, EventArgs e)*

Обработка события нажатия на кнопку «Построить» (Build\_button). После нажатия на кнопку последовательно выполняются следующие действия:

обнуляется счетчик ошибок (для случая, когда пользователь строит несколько графиков);

вычисляются значения начала и конца отрезка построения (для случая, когда пользователь вводит константы Pi и Eps или же какое-нибудь выражение);

проверяется наличие введённой функции в поле «f(x)»(textBox\_Function), если в этом окне текста не содержится (содержится пустая строка), возникнет сообщение «Функция не была введена» и пока пользователь не введёт функцию построения её графика не произойдёт;

сравниваются значения границ отрезка построения, в случае если левый конец больше правого, возникнет сообщение «Ошибка: Введите корректно концы интервала построения! Замечание: левый конец интервала должен быть меньше правого», пока пользователь не изменит значения границ отрезка построения на допустимые график функции строится не будет;

если значения полей ввода функции и границ отрезка построения её графика удовлетворили всем условиям, то вызывается метод CreateGraph(zedGraphControl1, MyGraphColor, eps\_step), который в качестве входных значений получает zedGraphControl1 - экземпляр класса ZedGraphControl (холст, на котором происходит построение графика), MyGraphColor - цвет построения графика, eps\_step – значение поля «r» (textBoxEpsStep), необходимое для «пошагового» построения графика с шагом, равным eps\_step. Значение булевских величин IsGraphCreate (необходимо для определения, построен ли график функции), Clear\_button.Enabled (необходимо для активации кнопки «Очистить» (Clear\_button)), Okrestnost\_button.Enabled (необходимо для активации кнопки «Показать» (Okrestnost\_button)) устанавливаются в значение истина (true).

в случае возникновения ошибки в ходе выполнения вышеперечисленных действий возникнет сообщение «Ошибка функционирования программы» и программа прекратит построение графика (в данном случае необходимо связаться с разработчиком для выяснения и утсранения неполадки).

*private void Update\_button\_Click(object sender, EventArgs e)*

Обработка события нажатия на кнопку «Обновить» (Update\_button). Данная кнопка была добавлена для случая более детальной прорисовки «сложных» графиков (например, y = sin(1/x)): происходит уменьшение значения , введённого пользователем в окно «r», в такое количество раз, которое осуществила программа при вычислении ширин дельта трубок в методе BuildArrDelt (о нём ниже) и, исходя уже из измененного значения перерисовывается график. Алгоритм перерисовки абсолютно аналогичен алгоритму построения графика в перый раз (см. обработку события нажатия на кнопку «Построить» (Build\_button)). Данная кнопка становится активной только в случает последовательного нажатия кнопок «Построить» и «Показать».

*private void Okrestnost\_button\_Click(object sender, EventArgs e)*

Обработка события нажатия на кнопку «Показать» (Okrestnost\_button). После нажатия начинается построение - и -трубок, исходя из следующего алгоритма:

в начале осуществляется проверка построен ли график функции (именно здесь используется булевская величина IsGraphCreate, упоминавшаяся выше);

обнуляются (очищаются) все значения списков listDlt, listFirstElem, listWidth, окон listBox\_delta, listBox\_Width, textBox\_MinWidth, textBox\_CurentWidth;

вызывается метод BuildEpsTrub, строящий -трубку;

вызывается метод BuildArrDelt, строящий -трубку(-и);

свойство активации кнопки Update\_button устанавливается в значение истина (true);

свойство активации кнопки Okrestnost\_button устанавливается в значение ложь (false);

в случае невыполнения условия IsGraphCreate, проверяющего построен ли график, возникнет сообщение «Ошибка: Сначала постройте график! Замечание: Нажмите на кнопку «Построить»»;

в случае возникновения ошибки в ходе выполнения вышеперечисленных действий возникнет сообщение «Ошибка функционирования программы» и программа прекратит построение графика (в данном случае необходимо связаться с разработчиком для выяснения причины неисправности и её утсранения).

*private void Form1\_Resize( object sender, EventArgs e )*

Обработчик события изменения размера окна программы (формы (Form1)). В случае изменения размера формы все элементы, содержащиеся на форме масштабируются относительно размера окна программы (одни лишь меняют координаты, другие – пропорционально увеличиваются).

*private void SetSize()*

Метод обрабатывает событие изменения размера формы: изменяет размер и положение графика.

*private void CreateGraph(ZedGraphControl zgc, Color MyGraphColor, double eps)*

Метод предназначен для построения графика, используя следующие три входных значения: zgc – экземпляр класса ZedGraphControl (холст, на котором изображен график), MyGraphColor – экземпляр класса Color (цвет, которым будет рисоваться график), eps – переменная типа double (величина ). Метод реализует следующий алгоритм:

объявление переменных x и y типа double.

вычисление значений графика функции с «отлавливанием» ошибок. Ошибки могут быть следующих типов: DIVBYZERO (деление на нуль), NOEXP (отсутствие вычисляемого выражения), SYNTAX (синтаксическая ошибка (некорректный ввод аргументов, неправильный ввод операций, неверное название функций и т. п.)), UNBALPARENTS (количество открывающих скобок не совпадает с количеством закрывающих), OTHER (ошибка алгоритма), NOERR (нет ошибки).

если экземпляр p.Errors.NOERR класса Parser.Errors имеет истинное значение (true) (т.е. все остальные виды ошибок ложны), то последует построение функции: цикл накопления значения переменной х. Он начинается со значения х=Dbeg (начальному значению – левой границе отрезка) и заканчивая значением х=Dend (конечное значение – правая граница отрезка), происходит выполнение метода Evaluate(func, x) класса Parser, использующего в качестве входных значений переменную типа string, в которой должно содержаться текстовое значение функции, вводимое пользователем, и переменную х типа double, точку в которой будет вычисляться значение функции func. Вычисленное значение будет записано в переменную y. После вызывается метод Add(x, y) объекта метода list1 класса ZedGraphControl библиотеки ZedGraph, который добавляет на холст точку с вычисленными координатами x и y, и соединяет её с предыдущей нанесённой точкой (если такая имеется). При ошибке DIVBYZERO точка просто не будет записана в у и не будет нанесена на холст.

далее выполняется следующий код: LineItem myCurve = myPane.AddCurve(sName, list1, MyGraphColor, SymbolType.Circle), который наносит на холст название функции (func), инструмент, которым будет строиться график (list1), цвет построения графика (MyGraphColor), геометрическое изображение точек (SymbolType.Circle).

*Color MyGraphColor*

Метод случайным образом выбирает значение типа Color из следующих: Blue, Navy, BlueViolet, CadetBlue, CornflowerBlue, Chartreuse, DarkBlue, DarkSlateBlue, DodgerBlue, Indigo, MediumBlue, MidnightBlue, RoyalBlue. Этим цветом будет строиться график функции.

*Color MyEpsColor*

Метод случайным образом выбирает значение типа Color из следующих: Gainsboro, Lavender, LightSalmon, Lime, LimeGreen, MediumAquamarine, MediumSpringGreen, Moccasin, PaleGreen, PeachPuff, Red, YellowGreen, Yellow, DarkRed. Этим цветом будет строиться -трубка.

*Color MyDeltColor*

Метод случайным образом выбирает значение типа Color из следующих: DarkOrange, DarkViolet, DarkOrchid, DarkSlateBlue, DarkGoldenrod, Tomato, DimGray, DarkCyan, Teal, Thistle, DeepPink, DarkTurquoise, Crimson, Fuchsia. Этим цветом будет строиться -трубка.

*private void TextBoxEnter(objectsender, EventArgse)*

Метод обрабатывает событие нажатия на окно. При нажатии на окна «f(x)», «От», «До», «r», «y0» текст в них выделяется, причем при выделении текста в окне «f(x)» он записывается в список предыдущих введённых значений listBoxFunc, который можно вызвать, кликнув правой кнопкой мыши по окну «f(x)».

*void BuildEpsTrub(ZedGraphControl zgc, double Y, Color MyColor)*

Строит -трубку, в точке Y, цвета MyColor, на холсте zgc. Цикл, начиная с x = Dbeg, заканчивая x = Dend, аналогично методу CrateGraph строит -трубку.

*void BuildDeltTrub(ZedGraphControl zgc, double Xbeg, double MinCountEps, double Y, Color MyColor)*

Строит -трубку, радиуса MinCountEps, в точке Xbeg=0, до значения Y, цвета MyColor, на холсте zgc. Цикл, начиная с x = Dbeg, заканчивая x = Dend, аналогично методу CrateGraph строит -трубку.

*bool yInTbk(double y)*

Метод возвращает истинное значение (true), если переменная y содержится в -трубке.

*void BuildArrDelt(ZedGraphControlzgc)*

Метод формирует массив массивов абсцисс точек, у которых ордината удовлетворяет условию yInTbk. Для этой задачи потребуется «ловить» события входа и выхода из -трубки, т.е. для каждой итерации цикла мы будем определять удовлетворяет ли ордината, вычисленная методом Evaluate, условию yInTbk. Итак, имеется цикл, начинающий перебор со значения x=Dbeg до значения x=Dend с шагом итерации eps = eps\_step/10 (перебирать ординаты с шагом eps\_step было бы несовсем рационально, поскольку программа перескакивала бы через относительно большое количество -трубок (здесь наиболее приемлемый вариант соотношения *затраченное время – полученный результат,* т.к. при большем разбиении программа анализировала бы результаты дольше, а при меньшем – не обладала бы необходимой точностью)), т.е., начиная с x=Dbeg к x последовательно прибавляется значение eps (каждую итерацию цикла), вычисляется ордината точки. Также для работы нам небходимо знать не только значение ординаты в точке с абсциссой х, но и следующее значение ординаты (значение в точке с абциссой, равной x+eps), чтобы можно было ловить события входа/выхода в -трубку.

Цикл по переменной х выполняет следующий алгоритм:

*–* вычисляет значения y (для x) и yNext(для x+eps).

*–* проверяет, удовлетворяет ли условию *yInTbk(y) == false&&yInTbk(yNext)* («y не в трубке, а yNext в трубке») значения обоих вычисленных переменных. Если они удовлетворяют, то булевский параметр Ent ставится в значение истина (Ent=true), в список listDlt объявляется новый подсписок.

*–* в случае невыполнения предыдущего условия проверяется, удовлетворяет ли условию *yInTbk(y)&&yInTbk(yNext) == false* («у в трубке, а yNextне в трубке»), и выполняется (в случае истинности выражения) присвоение к булевскому параметру Ext значение true.

*–* в иных случаях (в случаях, когда и у и yNext находятся вне -трубки) обоим булевским переменным присваиваются значение ложь (false).

Далее требуется обработать случай нахождения y в -трубке, т.е проверяется условие *yInTbk(y).* Причем:

*–* сначала необходимо проверить первое ли вхождение в -трубку сейчас. Эта проверка необходима для случая, когда ордината левой границы отрезка построения графика уже находится в -трубке и «поймать» событие входа в трубку не представляется возможным. В случае выполнения этого условия выполняется алгоритм, аналогичный для условия входа в -трубку.

*–* далее (в ином случае) необходимо проверить первый ли элемент будет добавляться в список значений listDlt, если да, то инкрементируем счетчик индексов для этого списка.

*–* иначе проверяем условие пустоты списка первых точек вхождений -трубок в -трубку (listFirstElem), если он пуст, то просто добавляем в него первый элемент иначе проверяем условие *Math.Abs(listFirstElem[listFirstElem.Count - 1] - x) > eps* (если абсолютная величина разности последнего значения в списке listFirstElem и х больше значения шага итерации eps). При значении true этого условия в список первых элементов добавляется элемент х.

*–* после выполнения вышеперечисленных проверок выполняется присвоение булевскому счетчику firstEl, определяющему первый «пойманный» элемент -трубки, значения false и добавление актуального значения х в список listDlt.

В случае, когда условие yInTbk(y) ложно, будет проверяться условие *!yInTbk(y) && ! yInTbk(yNext) && (y<y0 - eps\_step) && (yNext>y0 + eps\_step) || !yInTbk(y) && ! yInTbk(yNext) && (y>y0 - eps\_step) && (yNext<y0 + eps\_step)* («у не в трубке И yNext не в трубке И у меньше разности у0 (ордината расположения -трубки) и eps\_step (первоначального значения ) И yNext больше суммы у0 и eps\_step» ИЛИ («у не в трубке И yNext не в трубке И у больше разности у0 и eps\_step И yNext меньше суммы у0 и eps\_step»). Условие ИЛИ разделяет здесь случаи возрастания и убывания графика функции, ведь в первом случае значение следующей ординаты будет больше значения предыдущей, а во втором – меньше, что необходимо учитывать при анализе состояния нахождения в/вне -трубки. В случае истинности этого условия происходит уменьшение шага в 100 раз и выполняется тот же алгорим, что описан выше, только для измененного значения . Программное уменьшение шага итерации необходимо для случаев быстрого возрастания графика функции, когда шаг итерации может просто перескочить через -трубку. В данном случае вероятность перескока сокращается ещё в 100 раз.

Следующим шагом необходимо вывести все обнаруженные -трубки (им соответствуют первые элементы каждой из -трубок), поэтому пользователь в окне «Дельта-трубки» (listBox\_delta) увидит самые первые попавшие точки этих трубок (по одной точке от каждой трубки).

Далее необходимо построить -трубки минимальной ширины. Для этого происходит вызов методов GetWidth (о нём ниже) и BuildDeltTrub (о нём рассказывалось выше). Причем если ни одна ширина -трубок не является меньше заданного пользователем , то за ширину этой трубки будет приниматься величина уменьшенная в 10 раз. Это уменьшение необходимо для случая, когда ордината точки экстремума графика функции находится в -трубке, и для некоторых других случаев.

После выводится в окно «Минимальная» (MinWidth) значение минимальной ширины -трубки.

*double GetWidth()*

Метод возвращает наименьшую ширину -трубки, вычисляя сначала значение каждой, а в последствии сравнивая их между собой. В начале работы метода обнуляются все необходимые перерменные, счетчик выхода из трубки ставится в положение false.

Далее выполняется проверка на пустоту списка точек пересечения -трубки с графиком функции listDlt, если список пуст, то возникнет сообщение «Эпсилон-трубка не пересекает график функции у = <текст функции>».

Иначе переменной х1 присвоится значение первого элемента первого подсписка списка listDlt (*x1 = listDlt[0][0]*) и начнёт выполняться цикл по всем индексам i, пока значение i меньше количества подсписков в списке listDlt (*while (i < listDlt.Count)*). Если значение i равно нулю, то переменной firstX присвоится значение первого элемента первого подсписка списка listDlt (*firstX = listDlt[0][0]*). Далее последует цикл по элементам подсписков списка listDlt, начиная с нуля и заканчивая количеством элементов актуального подсписка (*for (intj = 0; j<listDlt[i].Count; j++*)). в котором переменной x2 присваивается значение j элемента i подсписка (*x2 = listDlt[i][j]*) и выполнится проверка условия *Math.Abs(x1 - x2) < eps\_step*(абсолютная величина разности х1 и х2 меньше ). В случае значения истины этой проверки произойдёт переприсваение переменной х1 значения х2 (*х1=х2*) и проверится значение булевской переменной ExitTbk, в случае true (истины) переменной lastX присвоится значение переменной х2 и в переменную TbkWidth запишется длина отрезка [firstX,lastX] (*TbkWidth = Math.Abs(firstX - lastX)*), значение переменной ExitTbk сменится на false.

В случае, противном случаю *Math.Abs(x1 - x2) < eps\_step*, переменная ExitTbk примет значение истина, переменные х1, firstX получат значения j-го элемента i-го подсписка списка listDlt (*x1 = firstX = listDlt[i][j]*).

После выполнения цикла *for (intj = 0; j<listDlt[i].Count; j++*) инкрементируется i.

Далее с помощью метода GetCurWidth(indx), который в качестве входного параметра принимает индекс первых элементов списка -трубок listFirstElem, выполняется цикл перебора всех значений ширин -трубок, записи этих значений в окно «Все значения» (listBox\_Width), причем в случае, когда ширина -трубки больше -шага, значению этой ширины присваивается значение -шага, уменьшенного в 10 раз. Также записываются значения в список ширин -трубок listWidth.

После выполняется поиск минимальной ширины путём обычной «пузырьковой» сортировкой (наименьшие элемент помещается в начало списка и всё это делается в двойном цикле).

*private void listBox\_delta\_SelectedIndexChanged(object sender, EventArgs e)*

Обработка события нажатия на окно «Дельта-трубки» (listBox\_delta). Данный метод возвращает ширину конкретной -трубки, которую выделил пользователь, в окно «Выделенной» (textBox\_CurentWidth).

*double GetCurWidth(intidx)*

Метод, получая на вход индекс -трубки, возвращает её ширину, сравнивая разницу между каждой парой элементов списка listDlt с величиной . В случае, если разница меньше , то в переменную lastX записывается последний взятый для сравнения элемент списка listDlt и разница между этим элементом и актуальным элементом списка listFirstElem будет являться актуальной шириной -трубки (актуальность определяется индексом, который является входным значением для этого метода).

*private void Clear\_button\_Click(object sender, EventArgs e)*

Обработка события нажатия на кнопку «Очистить». Программа запросит подтверждение очистки холста и после согласия пользователя на продолжение очистит панель отображения графика (холст), обнулит все списки (точек, лежащих между точкой входа и точкой выхода из -трубки; точек входа в -трубку; ширин -трубок), приведёт форму в первоначальный вид.

*private void Form1\_FormClosing(object sender, FormClosingEventArgs e)*

Обработка события нажатия на кнопку «Закрыть» (кнопка «красный крест»). Во избежание случайного закрытия программы пользователем программа запросит подтверждение выхода.

*private void button\_help\_Click(object sender, EventArgs e)*

При нажатии на кнопку «Справка» открывается окно-сообщение, содержащее сведения о возможностях программы, о её разработчике, месте и времени создания.

2. Компонент Parser.cs обрабатывает вводимую пользователем функцию.

Для обработки текста функции нам понадобится разбивать его на единицы (лексемы), которые впоследствии и будем обрабатывать. Лексема может быть следующих типов: NONE (пустая), DELIMITER (разделитель), VARIABLE (переменная), FUNCTION (функция), NUMBER (число). Также необходимо будет обрабатывать ошибки (иначе – некорректно введённые пользователем значения, функции и др.). Для этого разобьём ошибки на следующие типы: SYNTAX (синтаксическая), UNBALPARENTS (дисбаланс скобок), NOEXP (отсутствие выражения), DIVBYZERO (деление на нуль), OTHER (другая), NOERR (нет ошибки). Понадобятся переменные exp (ссылка на строку выражения), expIdx (текущий индекс в выражении), token (текущая лексема), tokType (тип лексемы), varX (переменная х), err (тип ошибки).

*public double Evaluate (string expstr, double var)*

Этот метод - входная точка анализатора, который на вход требует строку с функцией и значение переменной вещественного типа. После инициализации переменных и обнуления текущего индекса expIdx вызывается метод GetToken, которые возвращает текущую лексему. Далее следует проверка на пустоту этой лексемы, если она пуста, то переменной err присваивается NOEXP и анализатор возвращает 0. Далее следует цепь последовательных рекурсивных вызовов процедур, проверяющих совпадение текущей лексемы с различными операциями или функциями: вызывается метод EvalExp2 (о нём и о рекурсивном вызове чуть позже), после (рекурсивный вызов «посчитает» всю строку) проверяется текущая лексема, которая должна быть нуль-значением (пустой строкой), если это не так, то переменной err присвоится значение SYNTAX, иначе метод Evaluate возвратит значение функции в точке var.

*void EvalExp2(outdoubleresult)*

Метод EvalExp2 сначала инициализирует локальную переменную ор, вызывает метод EvalExp3, который «идет» дальше по рекурсивному вызову и возвращает вычисленный результат в переменной result, после сравнивает лексему с «-» или с «+», причем сравнение зацикленно: (переменной op присвоили значение лексемы) пока ор равна одной из этих операций будет выполняться запрос следующей лексемы, вызов метода EvalExp3. После прохождения рекурсивного вызова уже по методу EvalExp3, который вернет результат выражения, записанный в переменную partialResult, в зависимости от значения ор: если ор = «-», то переменной result присваиваем результат разности result и partialResult и выходим из цикла, если ор = «+», то – их сумму и также выходим из цикла.

*void EvalExp3(outdouble result)*

Метод EvalExp3 сначала инициализирует локальную переменную ор, обнуляет переменную partialResult, вызывает метод EvalExp4, который также как и в предыдущем случае «опускается» ниже по рекурсивному вызову и возвращает результат вычисленного в переменной result. После также, как и в предыдущем методе, зацикливается сравнение лексемы, записанной в переменную ор со следующими значениями: «\*», «/»; запрашивается следующая лексема и вызывается метод EvalExp4, с результатом в переменной partialResult. В результате совпадения ор с одним из них выполняется соотвественно перемножение partialResult и result или деление result на partialResult (причем проверяется деление на нуль).

*void EvalExp4(outdoubleresult)*

Метод EvalExp4 инициализирует локальную переменную ор, обнуляет значение partialResult, вызывает метод EvalExp5, возвращающий в переменной result значение посчитанного выражения, сравнивает ор с «^» и выполняет запрос следующей лексемы, вызов метода EvalExp4, возвращающего результат в partialResult, записывает в переменную exзначение result. В случае, когда partialResult равен 0, то переменной result присваеваем 1 и выходим из метода, иначе вход в цикл и начинается перебор с t = partialResult – 1 до t = 0 с декрементированием индекса t: в result присваеваем значение произведения result и ex. Тем самым получаем возведение в степень.

*void EvalExp5(outdoubleresult)*

Метод сначала инициализирует локальную переменную ор и присваивает её пустое значение. Следует сравнение

*if (((tokType == Types.DELIMITER) && token == "+" || token == "-" ) ||*

*((tokType == Types.FUNCTION) &&(token == "sin" || token == "cos" || token == "tg" || token == "sqrt" || token == "arcsin" ||token == "arccos" || token == "arctg" || token == "cosh" || token == "abs" || token == "sinh" || token == "tgh" || token == "log" || token == "ln"))*

Если значение сравнения истинно, то переменной ор присваиваем значение текущей лексемы, получаем следующую лексему. В случае, если тип следующей лексемы не равен VARIABLE, то вызываем метод EvalExp6, возвращающий нам результат в переменной result и далее в зависимости от значения ор производим вычисления синуса, косинуса, тангенса, квадратного корня, арксинуса, арккосинуса, арктангенса, гиперболических синуса, косинуса и тангенса, модуля, десятичного и обычного логарифмов от значения, полученного методом EvalExp6. Также метод EvalExp5 обрабатывает значения чисел и *e*, и переменной х.

*void EvalExp6(outdoubleresult)*

Метод обрабатывает выражение в круглых скобках. Вначале происходит сравнение лексемы с открывающей круглой скобкой, далее запрашивается следующая лексема и вызывается метод EvalExp2 (поэтому и называется рекурсивный вызов). После происходит сравнение текущей лексемы с закрывающей круглой скобкой и в случае несоответствия переменной err присвоится UNBALPARENTS, запросится следующая лексема. В противном случае вызовется метод Atom.

*void Atom(outdoubleresult)*

Метод возвращает значение числа, сравнивая тип текущей лексемы с типом NUMBER, в случае соответствия вернётся само число, иначе в переменную err запишется ошибка SYNTAX и метод вернёт 0. После сравнения запрашивается следующая лексема.

*void SyntaxErr(Errorserror)*

Метод обрабатывает сообщения об ошибках.

*void GetToken()*

Метод записывает в переменную token следующую лексему. Сначала происходит присвоение переменным tokType и token значений NONE и "" соответственно. После текущий индекс сравнивается с длинной строки функции и в случае равенства (вся строка пройдена) происходит выход из метода. Далее, пропуская пробелы в тексте функции, проверяем (с помощью метода IsDelim), является ли следующая лексема оператором: присваиваем в переменную token символ строки функции с индексом expIdx, инкриментируем этот индекс и присваиваем tokType тип DELIMITER. Если следующая лексема не является оператором, то проверяем, является ли она переменной или функцией: в цикле с условием пока лексема не является разделителем (знаком какой-либо математической операции) накапливает значение переменной token. Далее token сравнивается со списком определённых в программе функций (см. выше) и если значение этой переменной не совпадает ни с одной строкой-названием этой функции (например, token не равен «sin» И не равен «cos» и т. д.), то tokType присваивается VARIABLE, иначе – FUNCTION.

После проверяем, является ли лексема числом: также, как и в сравнении с функциями, накапливаем значение token в цикле до тех пор, пока лексема не разделитель и присваиваем tokType тип NUMBER.

*bool IsDelim(charc)*

Метод определяет, является ли одним из следующих символов  *+-/\*%^=()* входной символ c. В случае совпадения возвращает истину (true), иначе ложь (false).

3. Компонент MyMessage предназначен для вывода пользователю сообщений разного рода (ошибок, пояснений). Состоит из следующих методов:

*public MyMessage()*

Инициализация компонента.

*public static void Show (string msg)*

Обработка события появления окна. На вход принимает строку, в которой тегами выделен текст, который необходимо выделить жирным, курсивом или подчеркнуть, таким образом метод обрабатывает форматирование текста.

*static string MyReplase(int StartIdx, string SubStr, string Str)*

Метод заменяет в строке Str подстроку SubStr начиная с индекса StartIdx.

*static string GetLeftRightSelect(string TegStr1, string TegStr2, string str, List<int> leftL, List<int> rightL)*

Метод удаляет из строки теги и заполняет списки leftL и rightL индексами мест в строке, где раньше были теги, что позволяет определить с какого и до какого места выделять тем или иным способом.

*public string FindWort(string wort, string str)*

Метод ищет слово wort в строке str.

4. ZedGraph.dll – заимствованная библиотека (с сайта <http://2lx.ru/2009/03/cs-graphics/>). Она предоставляет новый компанент в Visual Studio 2008 в окно ToolBox: ZedGraphControl – холст, накотором строятся графики. В этой библиотеке заложены функции изменения осей скроллингом мыши, пересчет масштаба графика в зависимости от вида развёрнутого окна, подписи значений координат построенных и соединённых точек и многое другое. Более полную информацию о методе можно получить на сайте <http://jenyay.net/Programming/ZedGraph>.

**Глава 5. Примеры.**

1. f(x) = abs(15\*x + sqrt(23/pi)) (рис. 1).

Строим график с частотой разбиения и у0 = 10. Проблем не обнаружено, график и трубки построены, минимальная ширина вычислена: .

2. f(x) = (x^2 – 5\*x + 6)\*(x – 1)\*(x – 4)^3 (рис. 2).

Строим график с частотой разбиения = 0,01 и у0 = 1. Обнаружено 4 , определена минимальная ширина: .

3. f(x) = (x^2 – 2\*x – 1)/((x – 1)\*(x – 3)\*(x + 1)\*(x +3) (рис. 3).

Строим график с частотой разбиения = 0,01 и у0 = –1. Обнаружено 4 , определена минимальная ширина: .

4. f(x) = abs(sinh(x)\*sin(x)\*cos(x)) (рис. 4).

Строим график с частотой разбиения = 0,01 и у0 = 5. Обнаружено 16 , определена минимальная ширина: .

5. f(x) = sin(1/x) (рис. 5).

Строим график с частотой разбиения и у0 = 1. После анализа графика нажимаем кнопку «Показать» и программа начинает вычисление -дельта трубок. После построения их видим, что количество прорисованных трубок на холсте значительно меньше их реального (программного) количества. В данном случае программа в процессе работы уменьшала значение величины после определения вхождения значений функции в -трубку и вычисляла ширину -трубки с уменьшенным -шагом. Визуализированное представление графика на холсте использовало начальное значение -шага. Для таких ситуаций предназначена кнопка «Обновить», которая строит поверх начального графика такой же, но с величиной шага = /100. После обновления графика (рис. 6) можно уже более подробно рассмотреть определённые «пойманные» -окрестности.

Очевидно, что программа не сможет определить всё -окрестности, поскольку для любого, заданного пользователем значения ε, найдется -трубка, значительно меньшего радиуса так, что даже при программном уменьшении числа ε в 1000 раз (именно во столько уменьшает программа при обновлении графика), . В данном (одном из тяжёлых случаев) программа определила лишь некоторую часть окрестностей, но и она удовлетворяет нашей задаче – вычислению минимальной ширины -трубки. Как видно из (рис. 7) все значения меньше наперёд заданного , и в окне «Минимальная» отобразилось минимальное значение из всех определённых. Ширина трубок под номерами 3, 5, 6 и 7 равна нулю из-за недостатка точности (слишком «большого» значения для названных трубок). Программа определила их наличие, но не смогла вычислить ширину, т.к. не только ни одна точка, по которым строился график, не попала в эту окрестность, но и сама искомая окрестность не переселка ни одно из дополнительных разбиений (). Если уменьшить разбиение, то ширина данных окрестностей вычислится, но найдутся другие ширины, формально равные 0 по тем же причинам (рис. 8).

Отмечу, что поскольку в алгоритме программы заложено вычисление минимальной ширины -трубки из определённых, то в случае, когда одна или несколько трубок формально равны 0, то минимальной величине присваивается значение , уменьшенное в 10 раз, и именно оно отображается в окне «Минимальная».

Замечание: автор настоятельно советует выбирать для построения графика и анализа трубок отрезки необходимой величины, поскольку при достаточно большой длине отрезка построения и достаточно малом значении анализ результатов вычислений и вывод на экран (холст) трубок может занять ни одну минуту. Также при достаточно «тяжёлых» функциях, со множеством предполагаемых он советует набраться терпения не не завершать экстренно программу, т.к. в любом случае, независимо от точности вычислений, программа выдаст вычисленную величину .

Сущуствует вероятность сбоя в процессе визуализации при большом количестве , потому что их заголовки могут не поместиться на холст. В данных случаях программа не визуализирует трубки, а лишь выдаёт значения . Если произошло нечто подобное, то следует уменьшить отрезок построения и анализа функции.

**Глава 6. Заключение.**

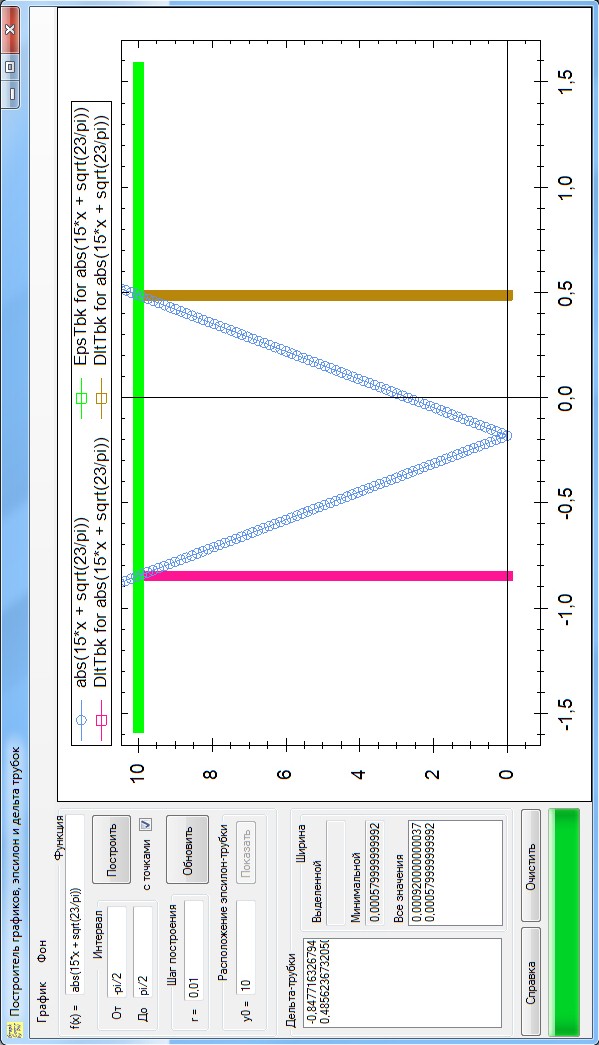
Мною была поставлена и в дальнейшем реализована задача визуализации понятия предела функции в точки. В ходе тестирования программы были обнаружены проблемные моменты, которые детально описаны в главах 3 и 5. Была произведена оптимизация процессов вычислений ширин , однако отмечу, что в некоторых случаях расчет результатов занимает достаточно большое количество времени.

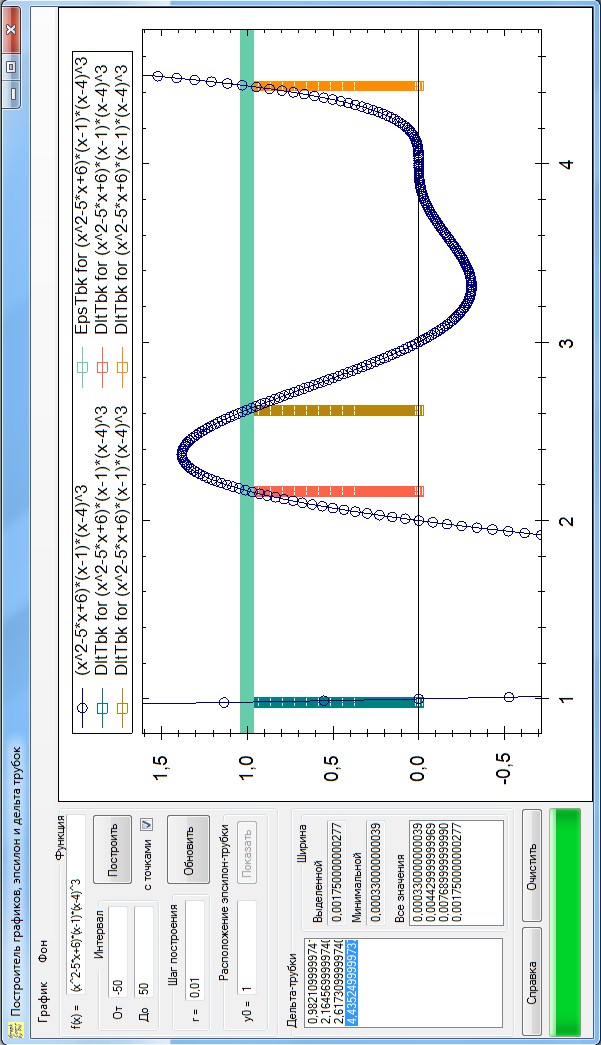
Подводя итог, считаю, что программа готова к использованию и может предоставляться в качестве методического пособия студентам начальных курсов.

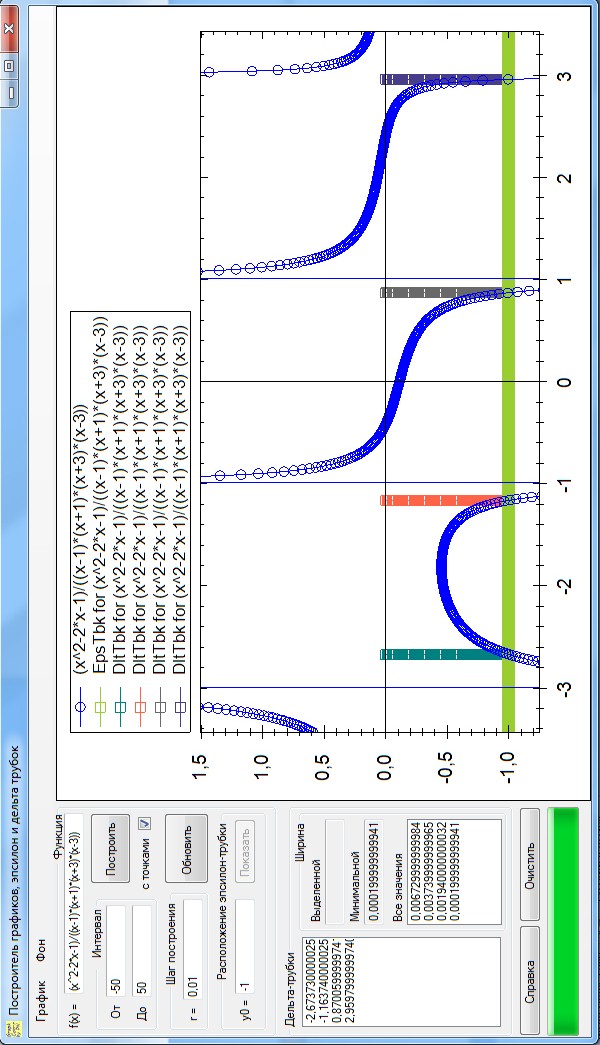
Автор выражает благодарность Трусову Андрею Сергеевичу за проведённые консультации по вопросам программирования и помощь в создании программы.

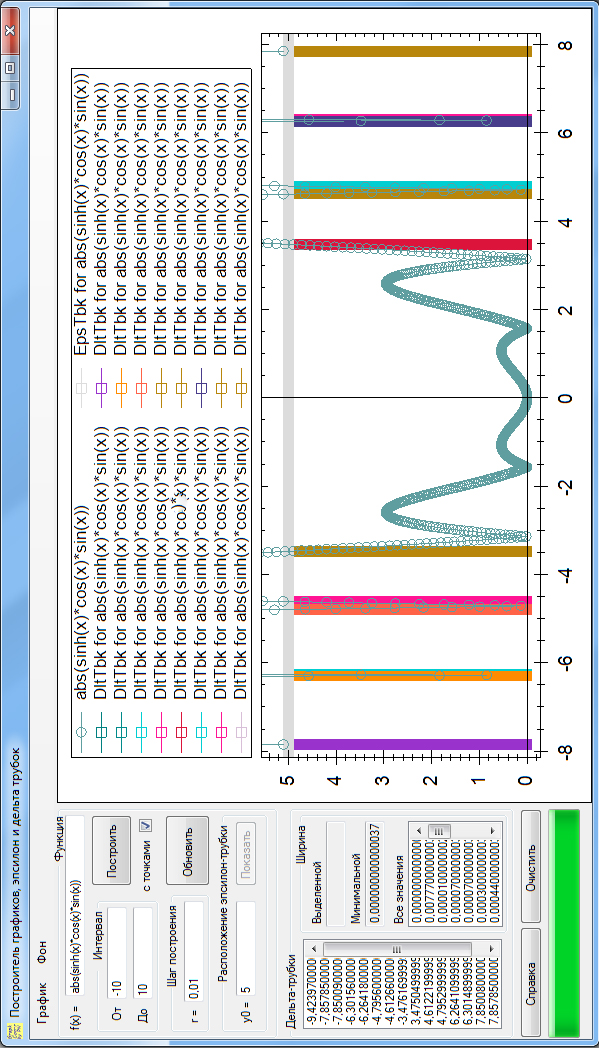
**Литература**

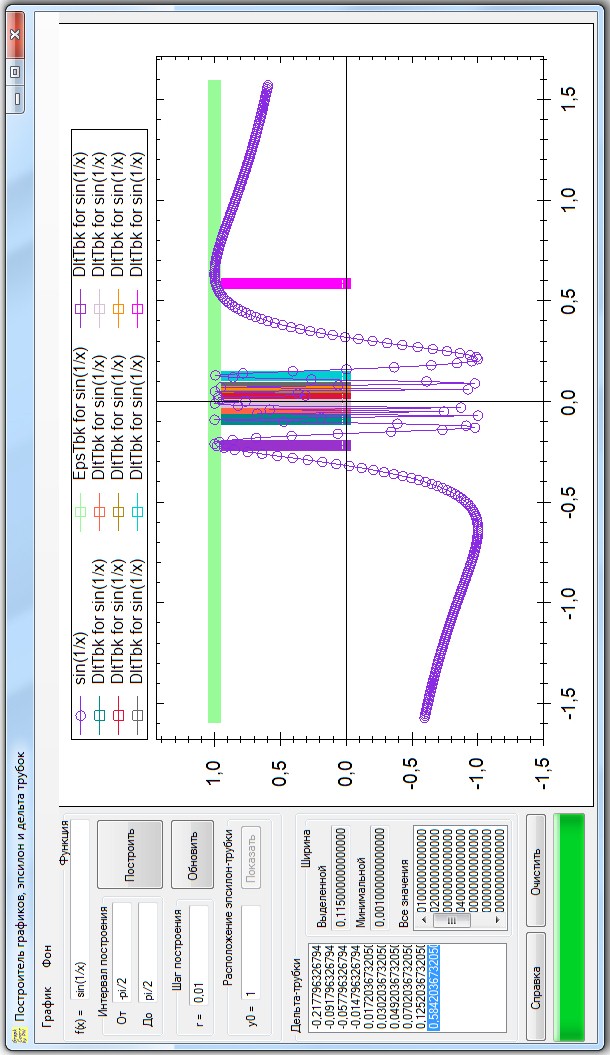
1. Кудрявцев, Л. Д. Курс математического анализа: в 3 т. /Л. Д. Кудрявцев. М.: Дрофа, 2003-2004. Т. 2. С. 165-193.
2. Шилдт , Г. Полный справочник по C# / Г. Шилдт – М.: Издательский дом «Вильямс», 2004.
3. <http://2lx.ru/2009/03/cs-graphics/>
4. <http://jenyay.net/Programming/ZedGraph>.
5. <http://mini-soft.ru/book/c_sharp/index.php>
6. <http://2lx.ru/2009/03/cs-graphics/>
7. <http://www.kbss.ru/blog/lang_c_sharp/147.html>

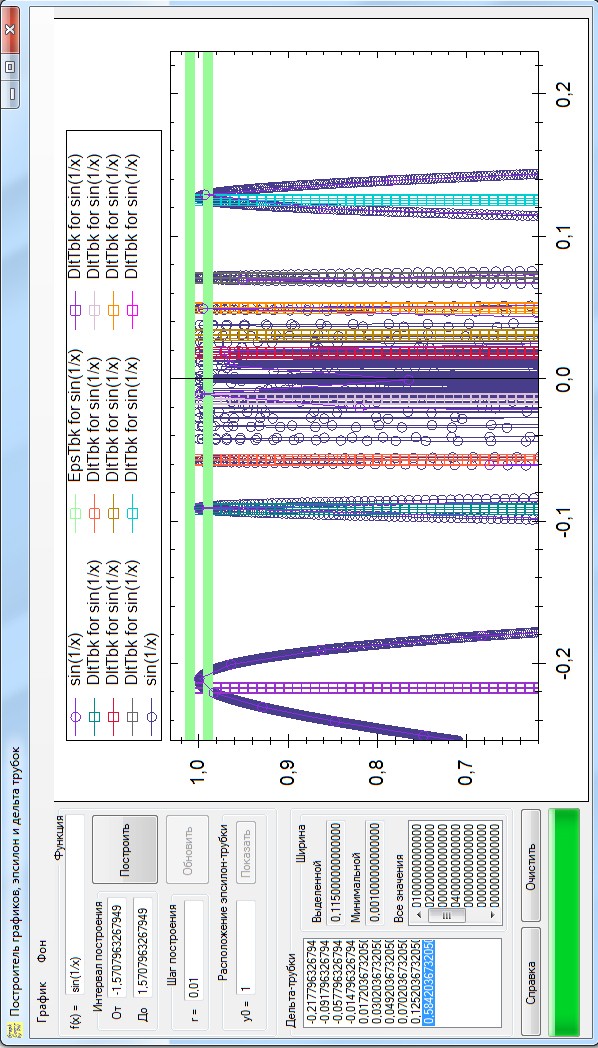
Рис. 1.

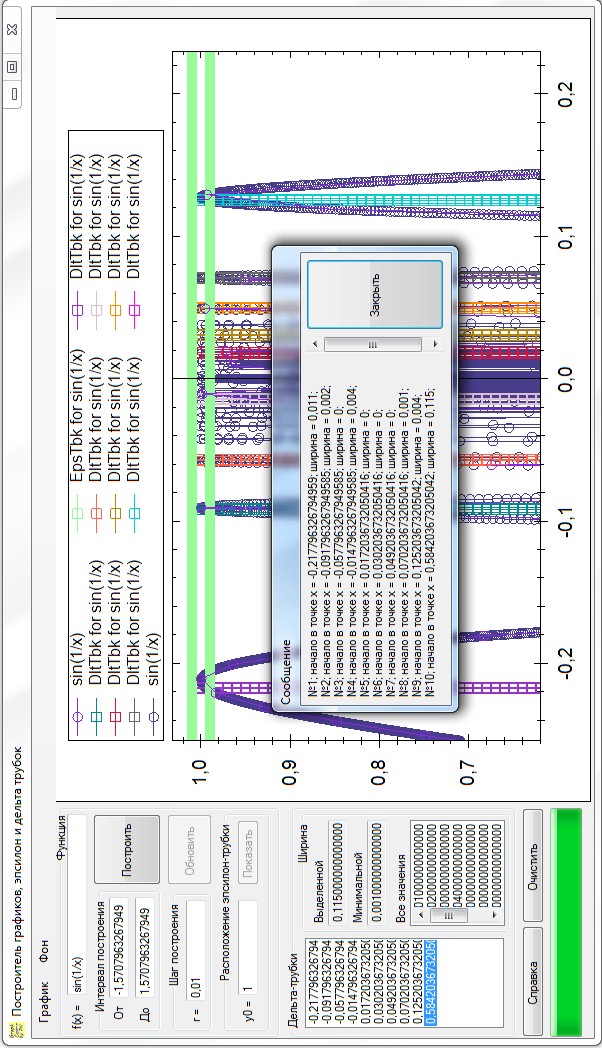
Рис. 2.

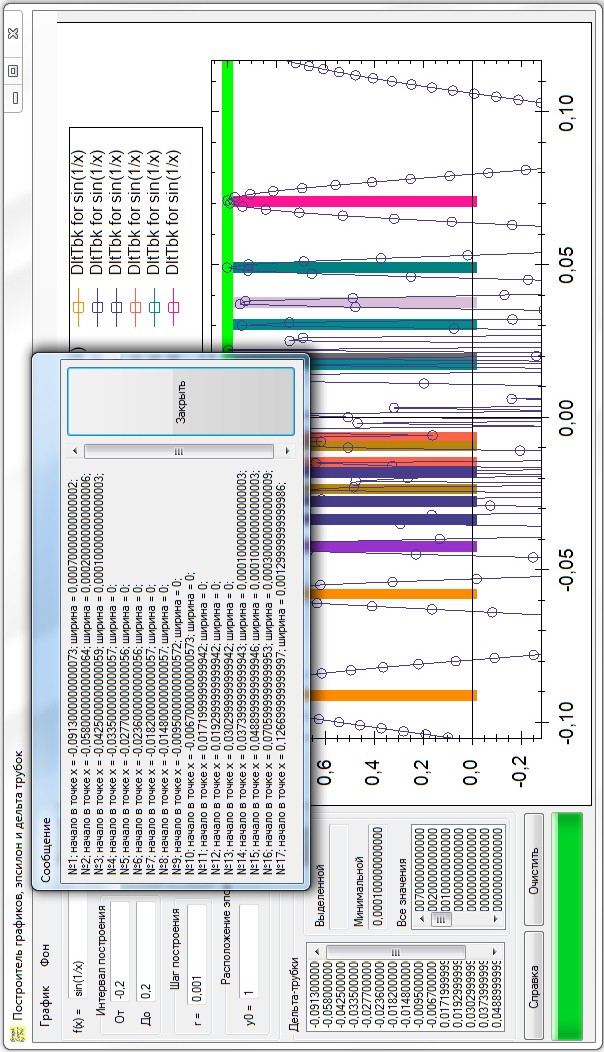
Рис. 3.

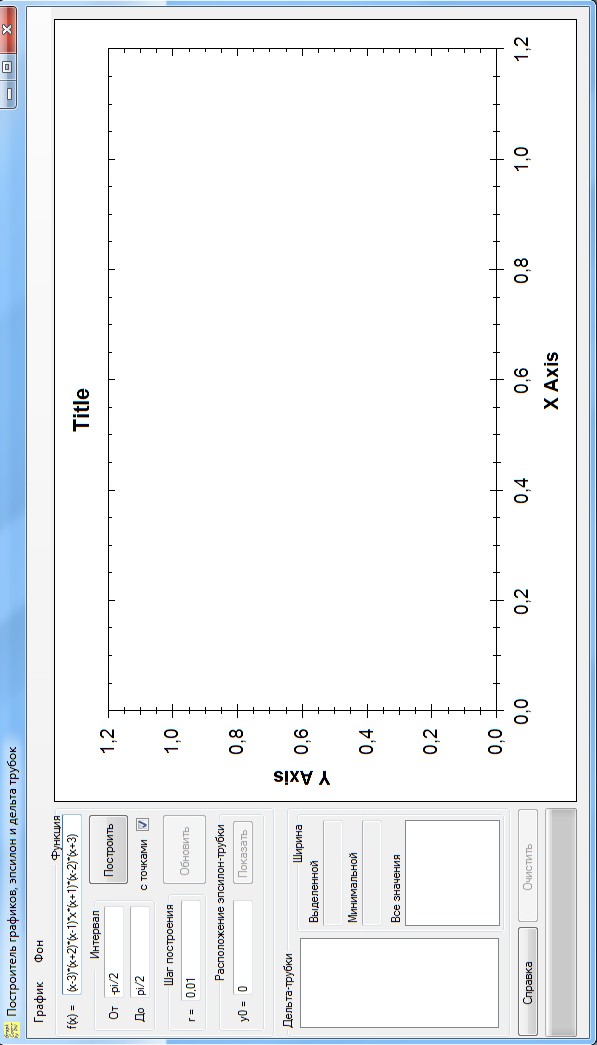
Рис. 4.

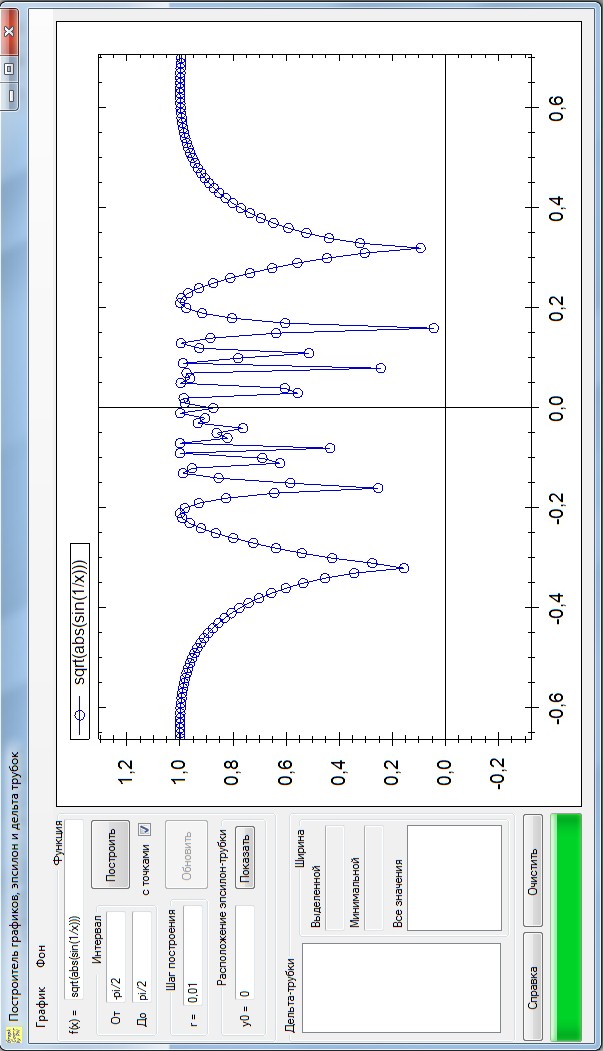
Рис. 5.

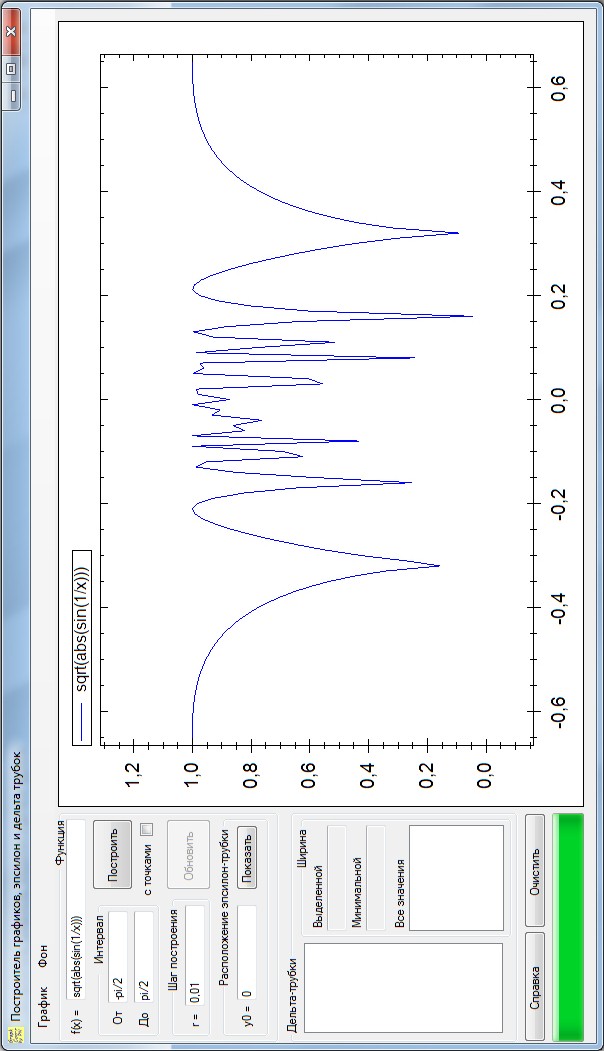
Рис. 6.

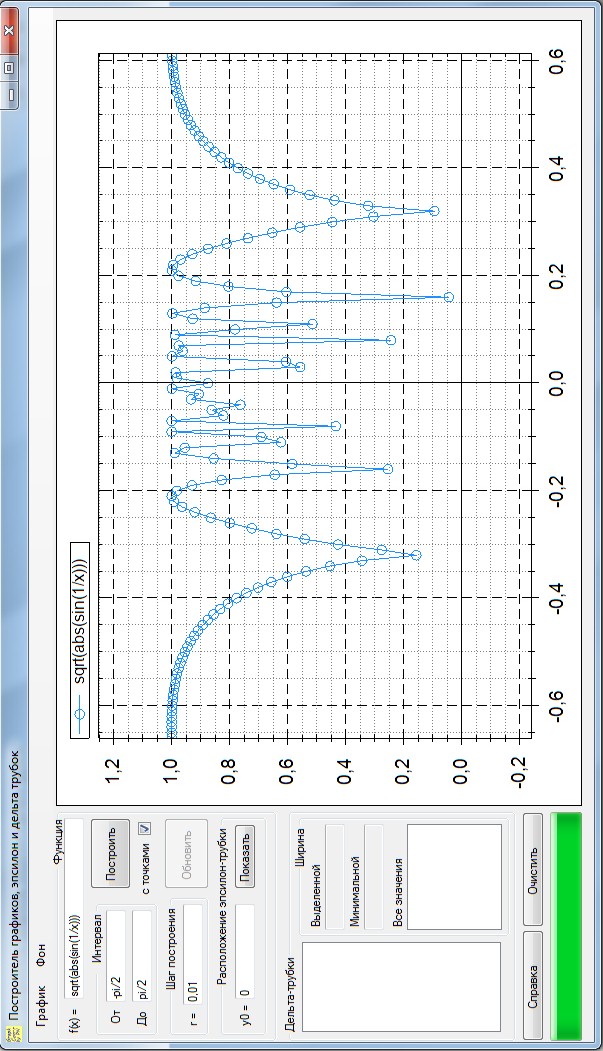
Рис. 7.

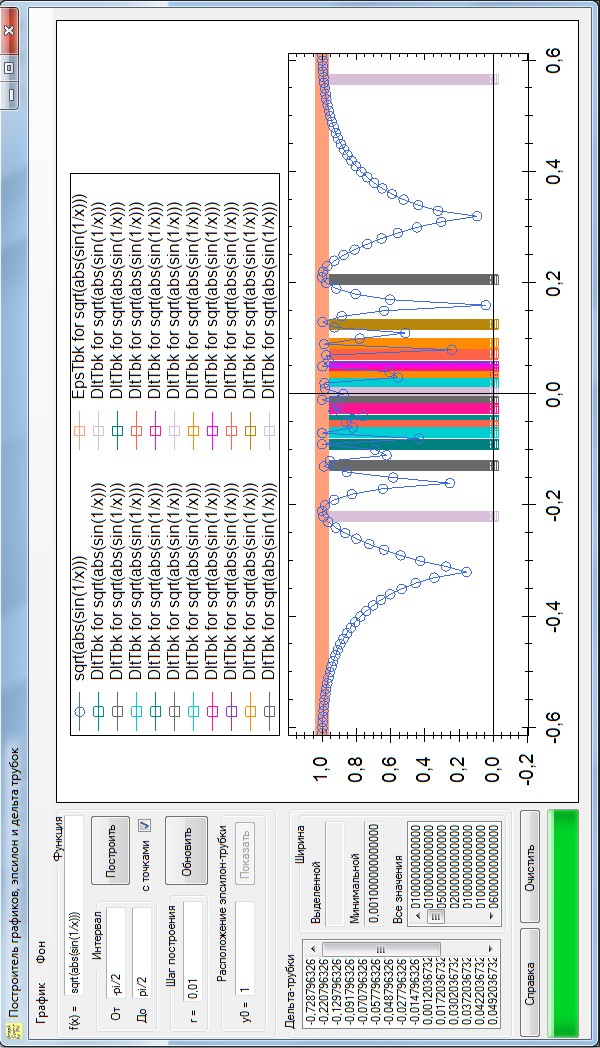
Рис. 8.

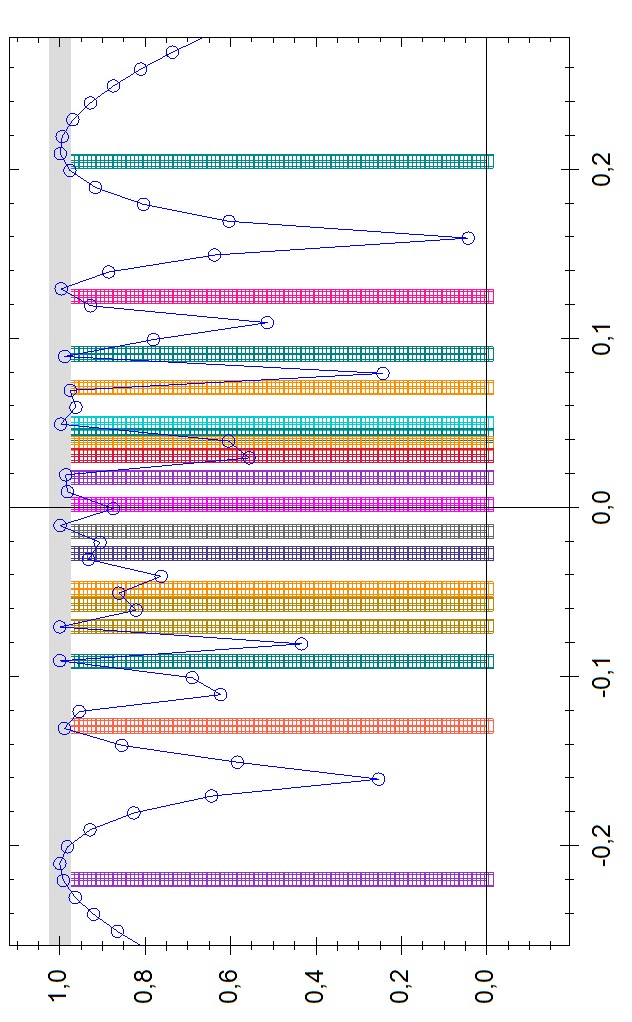
Приложение 1. Запуск программы.

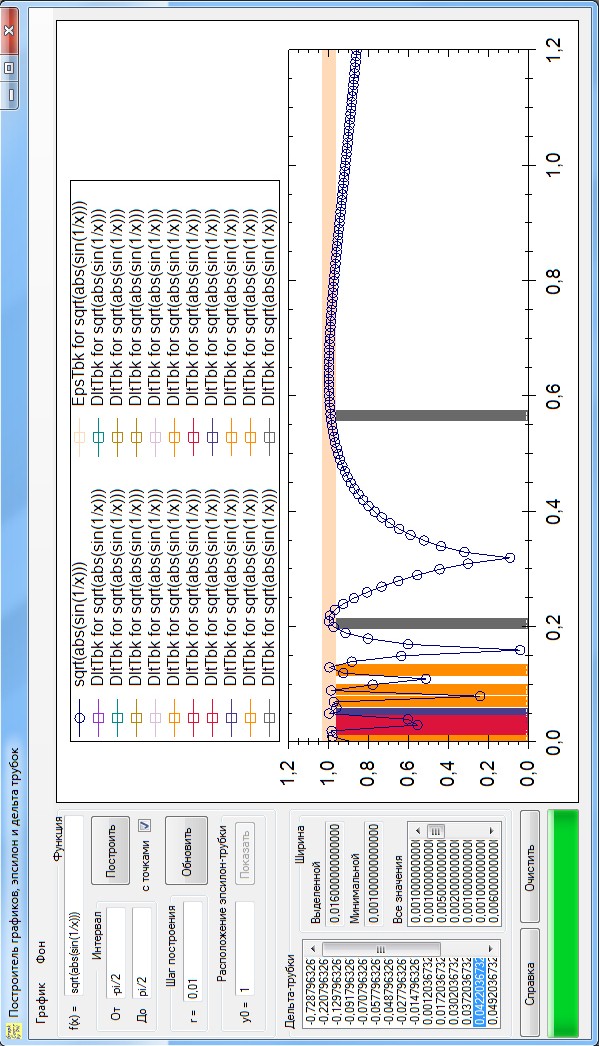
Приложение 2.1 Нажатие на кнопку «Построить» с осями и с точками.

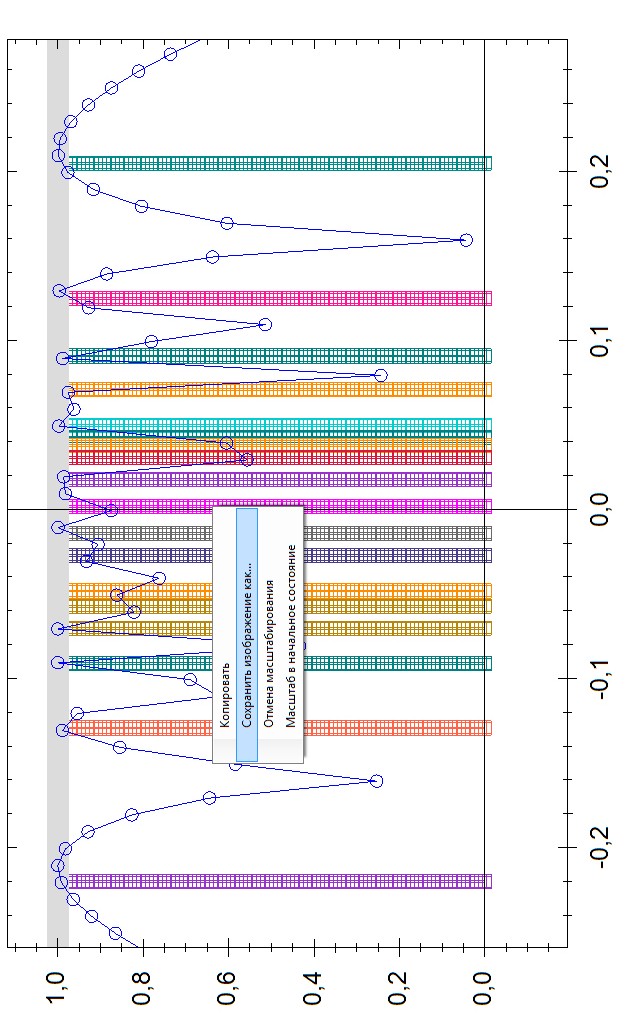
Приложение 2.2 Нажатие на кнопку «Построить» без осей и без точек.

Приложение 3. График – Показать сетку.

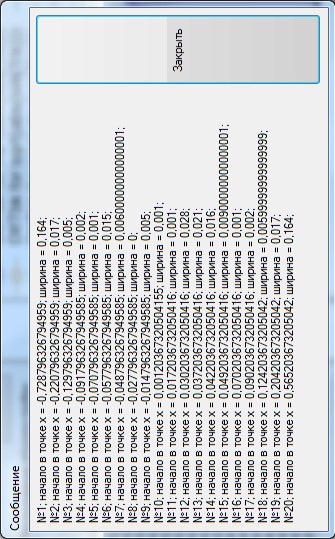
 Приложение 4. Трубки построены.

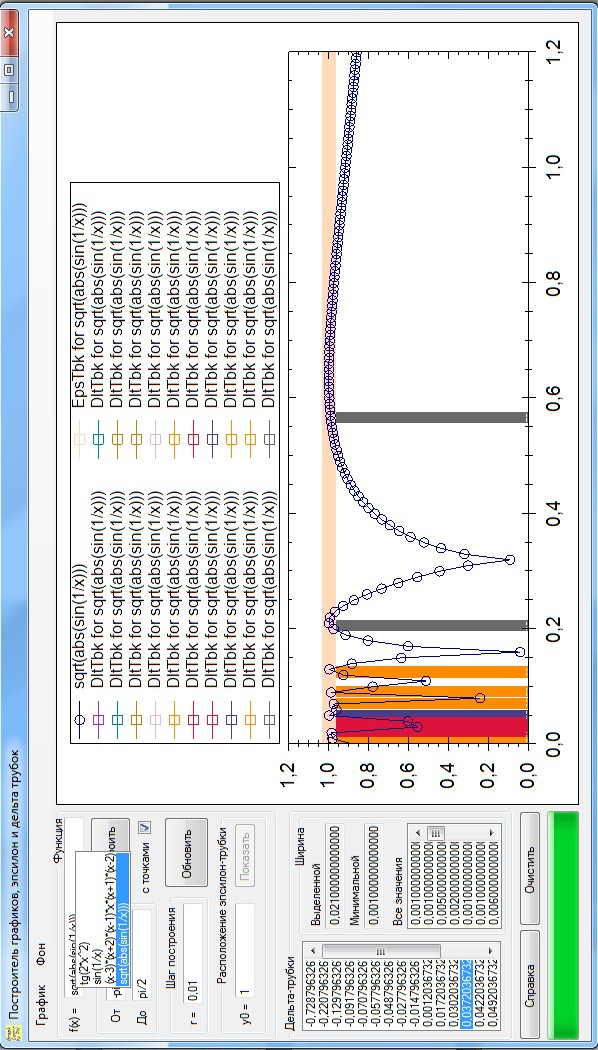
Приложение 5. Увеличенная область.

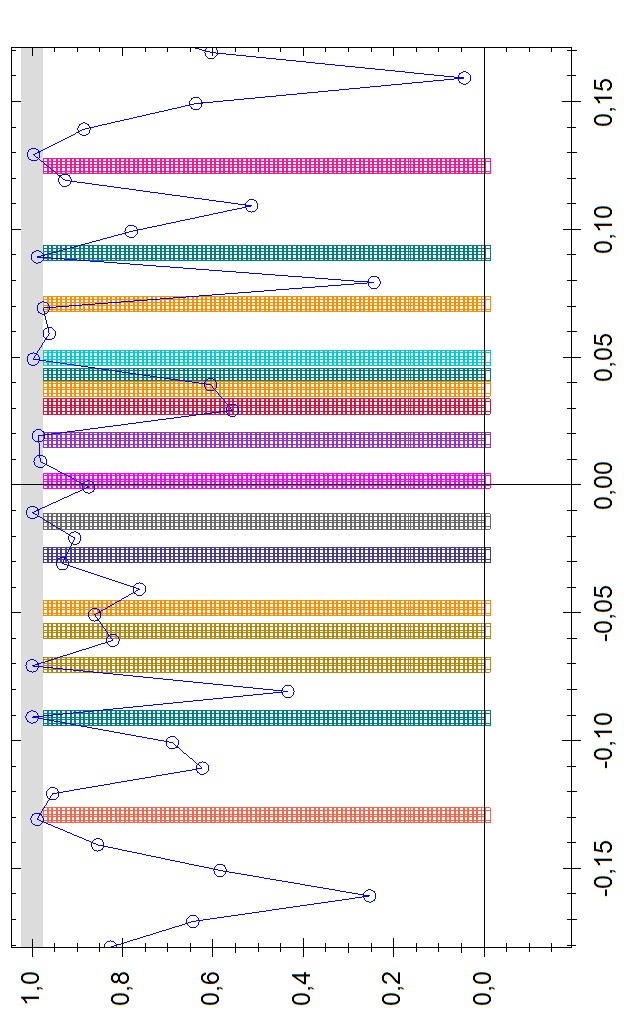
Приложение 6. Ширина конкретной -трубки.

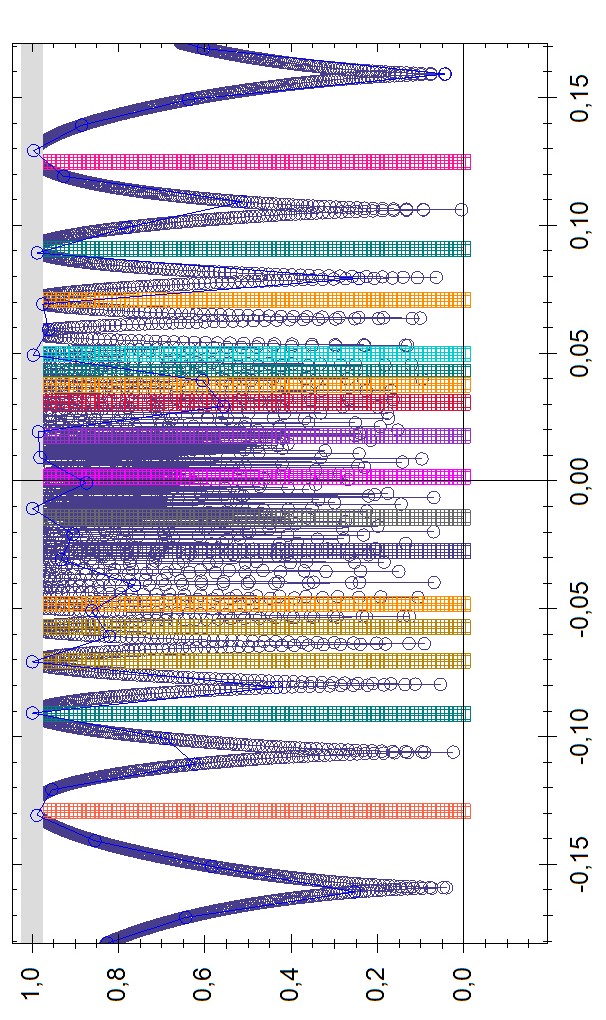
Приложение 7. Клик правой кнопкой по холсту.

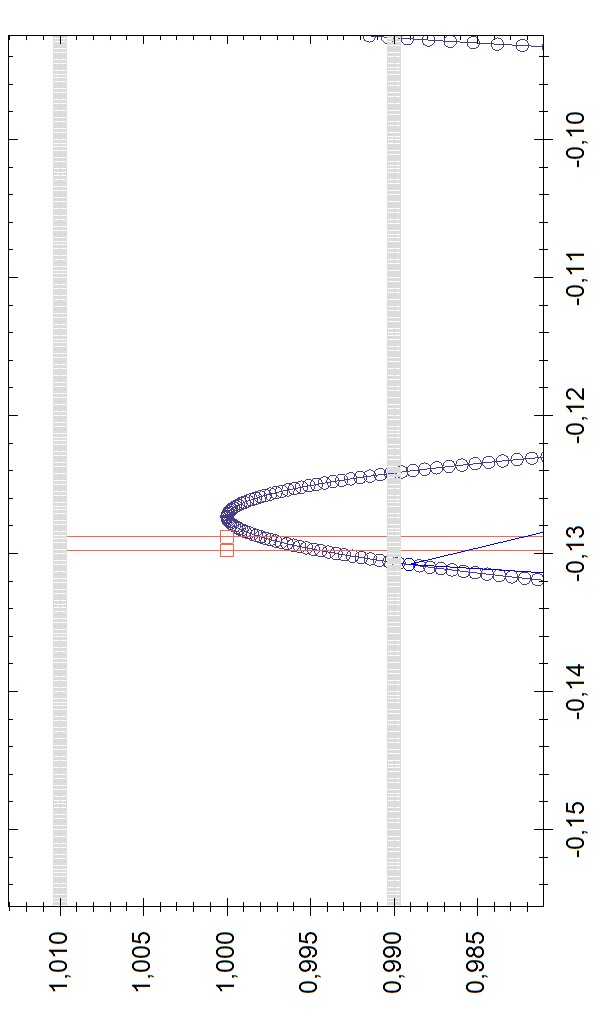
Приложение 8. Таблица всех значений-трубок.

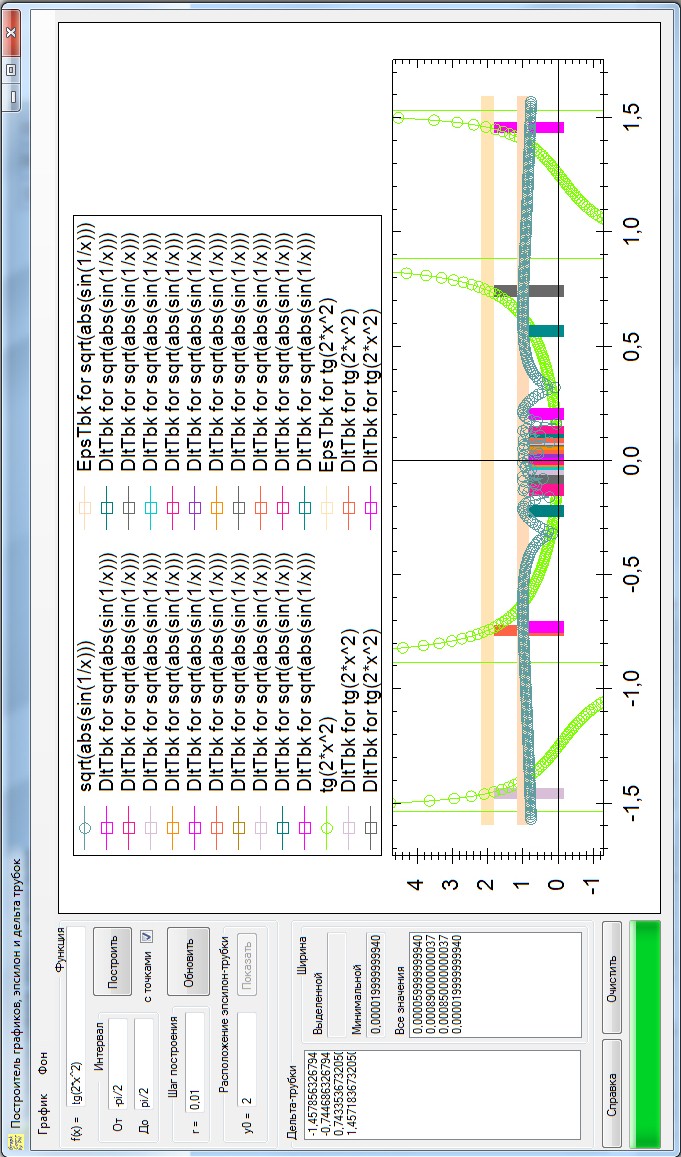


Приложение 9. Предыдущие значения.

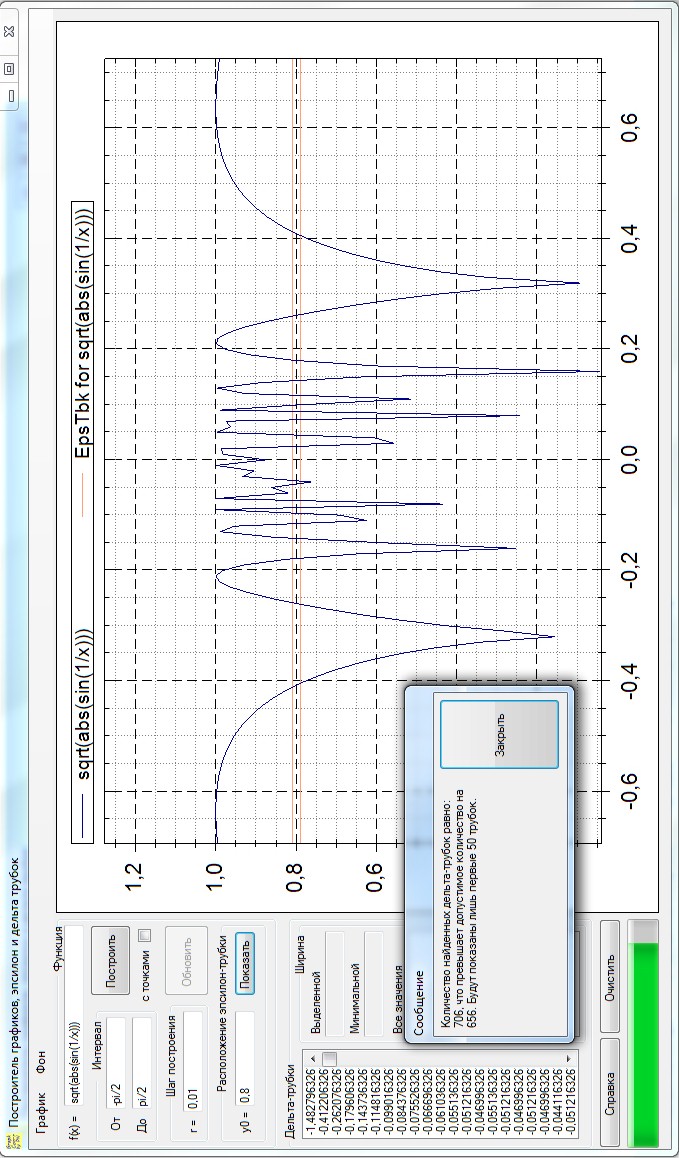
Приложение 10.1. До нажатия на «Обновить».

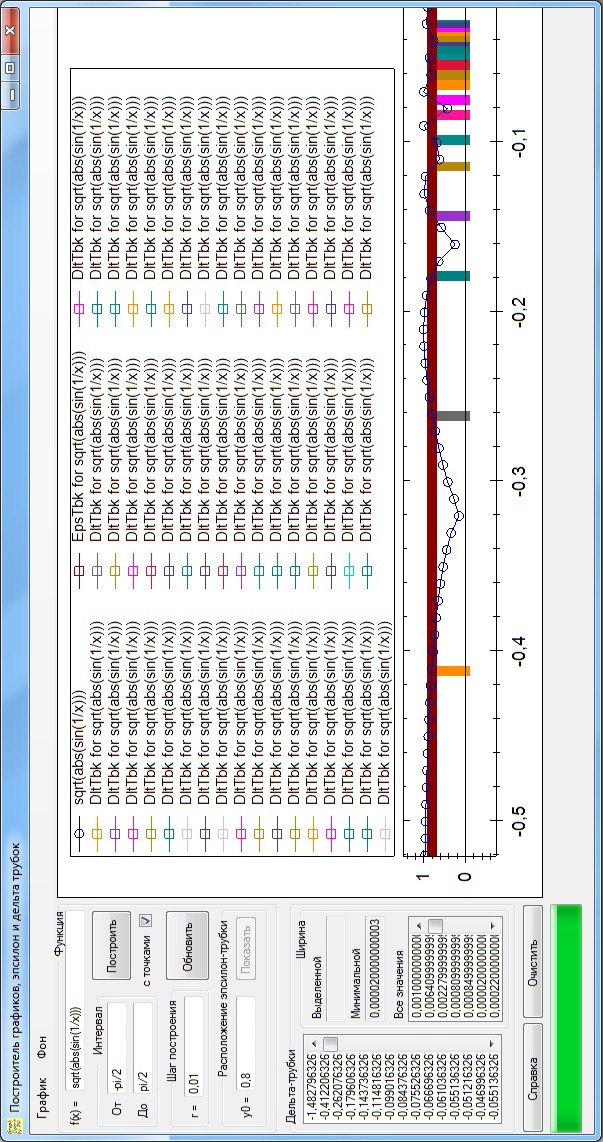
Приложение 10.2. После нажатия на «Обновить».

Приложение 11.

Приложение 12.

Приложение 13. Предупреждение.

Приложение 15. Проблемнй случай.

Приложение 16. 50 трубок построены.