Генератор псевдослучайных чисел

Цель работы

Разработка программного генератора псевдослучайных чисел (ГСПЧ) с использованием конгруэнтного метода.

Краткие сведения из теории построения генераторов

1. Мультипликативный конгруэнтный метод

Стандартным случайным числом называют величину z, равномерно распределенную на интервале (0,1). Плотность $f(z)$ распределения случайного числа z имеет вид:

$$f(z) = \begin{cases} 1, \textit{если } 0 \le z \le 1 \\ 0, \textit{в противном случае} \end{cases} \quad (1)$$

Генератор стандартных случайных чисел должен выдавать по запросу случайное число z. Повторяя запросы, можно получить последовательность $z_1, z_2, ..., z_n$ случайных чисел. Эти числа должны быть статистически независимы. При статистическом моделировании на ЭВМ вместо случайных последовательностей используют псевдослучайные последовательности $r_1, r_2, ..., r_n$, формируемые по детерминированным алгоритмам. При решении широкого класса задач они могут считаться как случайные.

Для генерации чисел $r_1, r_2, ..., r_n$ применяются различные программы, называемые генераторами стандартных псевдослучайных чисел (ГСПЧ). Среди разнообразных алгоритмов вычисления $r_1, r_2, ..., r_n$ широкое распространение получил так называемый мультипликативный конгруэнтный алгоритм. Он основан на применении операции A mod B вычисления неотрицательного остатка от деления на A и B. Этот алгоритм при известных значениях трех положительных целочисленных параметров m, a, x_0 предписывает последовательно вычислять целые псевдослучайные числа $x_i, i = 1,2...$ по рекуррентной формуле:

$$x_i = (ax_{i-1}) \bmod m \qquad (2)$$

и через них – стандартные псевдослучайные числа r_i по формуле:

$$r_i = x_i / m \qquad (3)$$

Получаемая с помощью ГСПЧ последовательность $r_1, r_2, ..., r_n$ должна быть как можно более близка по своим статистическим свойствам к математической последовательности $z_1, z_2, ..., z_n$ стандартных случайных чисел. Отсюда следует, что для идеального ГСПЧ должны выполняться следующие два требования:

- числа r_i должны иметь равномерное распределение на отрезке (0,1);

- числа r_i должны быть статистически независимы.

Близость мультипликативного конгруэнтного ГСПЧ к идеальному определяется тем, насколько удачно в формуле (2) подобраны параметры m, a, x_0 (модуль, множитель и начальное значение).

2. Выбор модуля, множителя и начального значения

Очевидно, что параметры m, a, x_0 следует выбирать таким образом, чтобы в последовательности (1) целых псевдослучайных чисел x_i ни при каком i не появилось число $x_i = 0$, так как в этом случае все последующие члены также будут нулевыми и последовательность не будет псевдослучайной. Чтобы избежать нулевых x_i достаточно выбрать параметр m четным, а множитель a и начальное значение x_0 - нечетными. В этом случае все x_i будут нечетными и тем самым исключается возможность появления нулевого члена.

Принадлежность чисел $r_1, r_2, ..., r_n$ отрезку (0,1) обеспечивается формулами (2) и (3). Действительно, каждое x_i есть неотрицательный остаток от деления некоторого числа на m. Следовательно, $0 \le x_i \le m$. Разделив это неравенство на m, получим $0 \le x_i / m \le 1$, или, с учетом (3), $0 \le r_i < 1$.

Нетрудно доказать, что последовательность $r_1, r_2, ..., r_n$ и $x_1, x_2, ..., x_n$ заведомо будут периодическими. Это вытекает хотя бы из того, что число возможных остатков от деления на m конечно и, таким образом, конечно число возможных

значений x. Длина периода должна быть достаточно велика, чтобы при практическом использовании $r_1, r_2, ..., r_n$ они не повторялись. Поэтому рекомендуется для построения ГСПЧ брать достаточно большое значение модуля m.

Еще одно заведомое отличие последовательности $r_1, r_2, ..., r_n$ от «идеальной» заключается в том, что эти числа могут принимать в интервале (0,1) лишь дискретный ряд значений, отстоящих друг от друга не менее чем на 1/m. Этот недостаток также становится практически несущественным если выбирается достаточно большое число m.

Равномерность распределения чисел r_i обычно достигается за счет выбора параметров m, a, x_0, не имеющих общих делителей. В этом случае множество остатков x_i, вычисляемых по формуле (1), имеет тенденцию равномерно заполнять промежуток между 0 и m.

Для обеспечения независимости следует выбирать достаточно большое значение множителя a, например, a > 1000, чтобы в (2) по возможности на каждом шаге произведение ax_i многократно превышало модуль m. Соблюдение приведенных простых рекомендаций повышает вероятность того, что выбираемые значения параметров ГСПЧ обеспечат достаточно высокое его качество. Для более достоверной оценки пригодности синтезированного ГСПЧ необходимо выполнить анализ статистических свойств генерируемой на выходе ГСПЧ псевдослучайной последовательности $r_1, r_2, ..., r_n$.

Контрольные вопросы

1. Какие свойства имеет последовательность $z_1, z_2, ..., z_n$ на выходе генератора стандартных чисел?
2. Какие требования предъявляются к ГСПЧ?
3. Как вычисляется последовательность $r_1, r_2, ..., r_n$ псевдослучайных чисел по мультипликативному конгруэнтному алгоритму?

4. Почему из конечного числа остатков от деления на модуль m вытекает периодичность мультипликативной конгруэнтной последовательности $r_1, r_2, ..., r_n$?

Задание

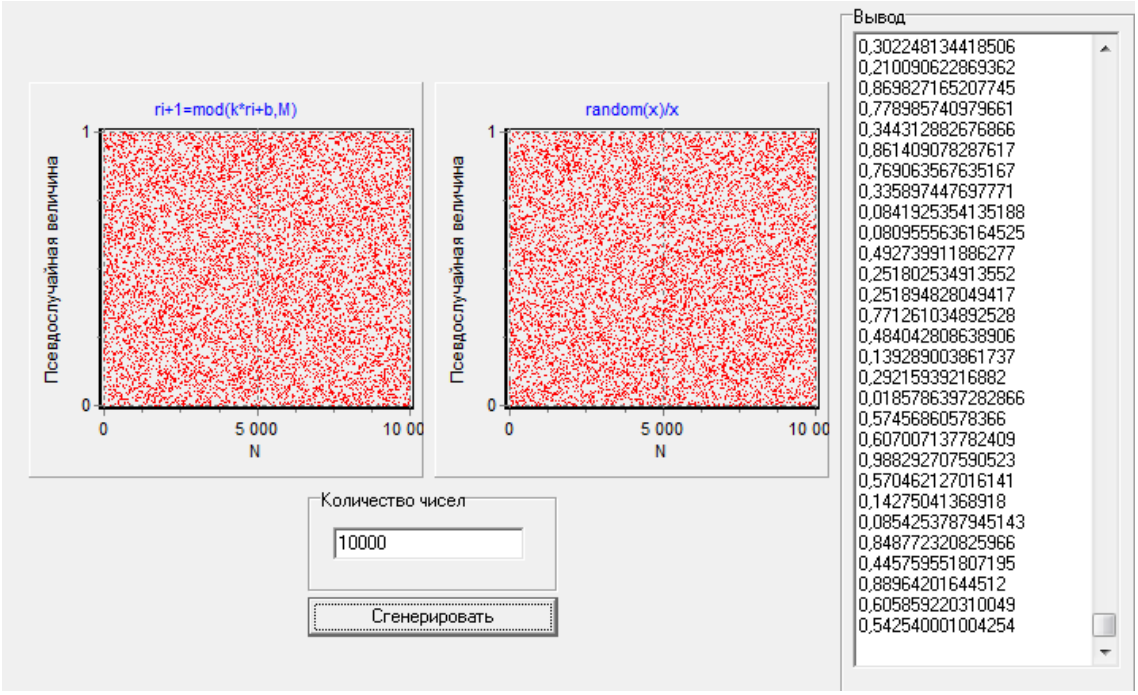
1. Изучите мультипликативный конгруэнтный метод построения ГСПЧ и рекомендации по выбору его параметров. Выберите в соответствии с этими рекомендациями значения параметров ГСПЧ.

2. Напишите программу для вычисления и вывода n чисел, порожденных генератором. В программе должна быть реализована функция выбора длины псевдослучайной последовательности (выбор n). Также, в программу должно быть включено сравнение с функцией random() построением графика функции r(n), где r – псевдослучайное число, n – номер числа.

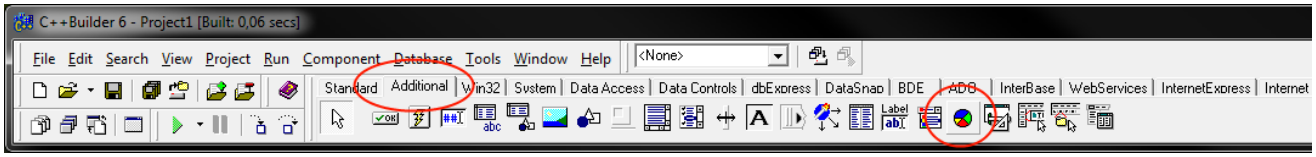
3. Оцените полученный результат визуально на графике ri(n) в сравнении с графиком функции random(). Результат должен быть схожим при одинаковом размере выборки.

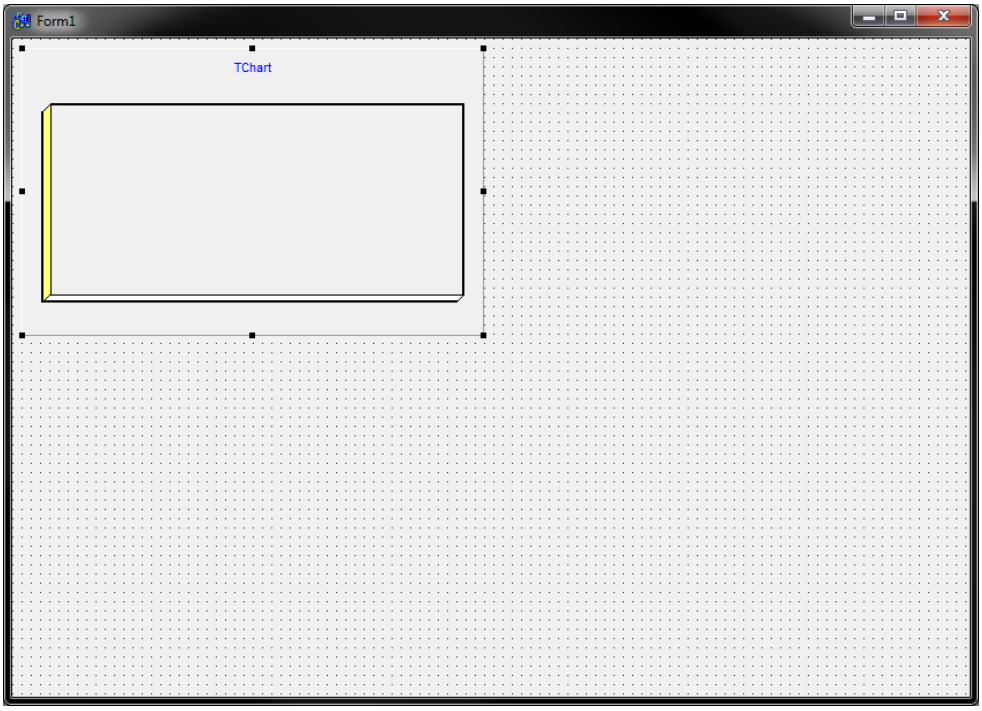
Рекомендации по выполнению работы

Пример программы-генератора и сравнение со встроенной функцией C++ random():



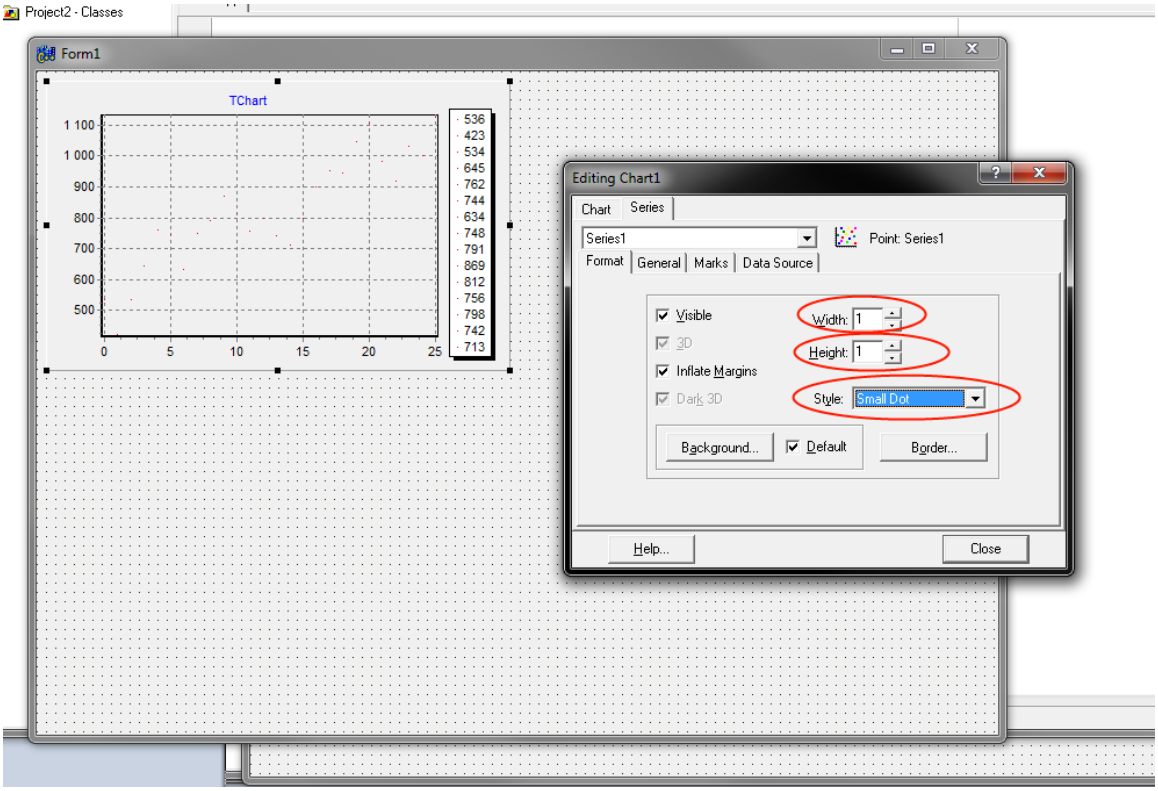
Для построения графиков необходимо использовать компонент Chart, поместив его на форму:





Затем добавить сам график двойным кликом мыши на установленном объекте и в открывшемся окне нажать Add, выбрав тип графика Point, галочку 3D убрать.

Перейти на вкладку Series и установить следующие параметры для графика:



Добавьте на форму объекты Button, Memo, Edit, Label, чтобы привести программу к виду как на рисунке.

Каждое последующее случайное число рассчитывается на основе предыдущего случайного числа по следующей формуле:

$$r_{i+1} = \mathrm{mod}(k \cdot r_i + b, M)$$

M — модуль ($0 < M$); k — множитель ($0 \le k < M$);

b — приращение ($0 \le b < M$); r0 — начальное значение ($0 \le r0 < M$)

По этой формуле генерируются псевдослучайные целые числа. Для приведения их к виду с плавающей запятой от 0 до 1 (X), следует r_{i+1} разделить на M:

$$X = r_{i+1}/M;$$

В C++ существует функция, позволяющая генерировать псевдослучайные последовательности чисел по равномерному закону распределения. Такой

функцией является random(x); где x – максимальное возможное значение псевдослучайного числа, например, функция random(10) генерирует число, имеющее возможные значения в интервале [0,10]. Данная функция генерирует только целочисленные значения.

Циклический вызов данной функции позволяет сгенерировать необходимую псевдослучайную последовательность, например, конструкция:

```
for (int i = 0; i < 10; i++)
   Series2->AddXY(i, (double)random(10000) / 10000);
```

строит на графике псевдослучайную последовательность из 10 чисел, имеющих значения от 0 до 1. Деление на 10000 производится для того, чтобы привести целые числа, выводимые функцией random() к типу с плавающей запятой (по аналогии с делением на M). Данная функция является аналогом той, которую создаем в данной лабораторной работе, поэтому ее удобно использовать для проверки корректности работы.

По нажатию кнопки программа должна начать генерацию любого количества чисел, указанного пользователем, для этого необходимо кликнуть 2 раза на объекте Button1 и вписать код обработчика события.

Считывание числа N из компоненты Edit1:

```
N = StrToFloat(Edit1->Text); // Преобразование в float
```

Остаток от деления определяется следующим образом:

```
a = b % c; // a, b, c – переменные типа long
```

Вывод числа последовательно в компоненту Memo:

```
Memo1->Lines->Add(ri); // ri – псевдослучайное число
```

```
// Пример генерации 1000 псевдослучайных значений
int m = 2147483647;
int count = 1000;
int b = 1;
double Xi = 0;
long ri = 7;
for (int i = 0; i < count; i++)
{
   ri = ri * k;
   ri += b;
   ri = ri % M;
   Xi = (double)ri / M;
   Memo1->Lines->Add(Xi);
   Series1->AddXY(i, Xi);
   Series2->AddXY(i, (double)random(10000) / 10000);
}
```

Анализ качества ГСПЧ

Цель работы

Анализ равномерности распределения и статистической независимости чисел на выходе ГСПЧ.

Краткие сведения из теории построения ГСПЧ

Для анализа качества ГСПЧ применяются различные статистические тесты, выявляющие соответствие ГСПЧ двум основным требованиям: равномерности распределения и независимости генерируемых чисел.

1. Тестирование равномерности распределения

Равномерность распределения можно проверять с помощью частотного теста. Суть этого теста состоит в построении эмпирического распределения чисел r_i и его сравнении с теоретическим, т.е. равномерным распределением.

Для этого интервал (0,1) возможных значений r_i разбивается на k одинаковых интервалов, генерируется выборка $r_1,...,r_n$, для каждого интервала (j=1,…,k) определяется количество n_j тех псевдослучайных чисел, которые попали в этот интервал и вычисляются относительные частоты $f_j = (n_j / n)$ попаданий. Для идеального генератора при $n \to \infty$ выполняются условия

$$f_j \to 1/k, \qquad j = 1,..., k \tag{4}$$

т.е. частота f_j попадания в интервал сходится к вероятности $1/k$ попадания в него стандартного случайного числа z.

Из (4) следует, что при $n \to \infty$

$$kf_j \to 1, \qquad j = 1,..., k, \tag{5}$$

Т.е. для идеального ГСПЧ эмпирическая плотность kf_j вероятностей на каждом интервале сходится к теоретической (1). Для реального ГСПЧ вычисляют величины kf_j для нескольких достаточно больших значений n и проверяют, приближаются ли они с ростом n к 1.

2. Косвенная проверка равномерности распределения

Косвенная проверка равномерности распределения может быть осуществлена путем оценивания математического ожидания и дисперсии псевдослучайных чисел $r_1,...,r_n$. При равномерном распределении на (0,1) математическое ожидание и дисперсия равны 1/2 и 1/12 соответственно. Оценки M (м.о.) и D (дисперсия) для выборки $r_1,...,r_n$ рассчитываются по известным статистическим формулам

$$M = (r_1 + ... + r_n)/n = S1/n \quad (6)$$

$$D = (r_1^2 + ... + r^2{}_n)/n - M^2 = S2/n - M^2 \quad (7)$$

Где суммы S1 и S2 накапливаются в процессе генерации выборки. Очевидно, если выборка $r_1,...,r_n$ характеризуется равномерным распределением, то M и D с ростом n должны сходиться к 1/2 и 1/12 соответственно. Это условие является необходимым для равномерности распределения чисел $r_1,...,r_n$, но не достаточным. Поэтому его выполнение лишь косвенно подтверждает (но не доказывает) гипотезу о равномерности распределения.

3. Проверка статистической независимости

Проверка независимости чисел на выходе ГСПЧ обычно производится путем измерения корреляции между r_i и r_{i+g}, где g > 0 – некоторое смещение. В случае, если установлена равномерность распределения, оценка R_g коэффициента корреляции между r_i и r_{i+g} может быть получена по формуле

$$R_g = \frac{M[r_i r_{i+g}] - 1/4}{1/12} = 12M[r_i r_{i+g}] - 3 \quad (8)$$

Здесь $M[r_i r_{i+g}]$ - оценка м.о. произведения чисел r_i и r_{i+g} (т.е. среднее значение произведений пар чисел, отстоящих в выборке на g шагов друг от друга):

$$M[r_i r_{i+g}] = \frac{1}{n-g}(r_1 r_{1+g} + ... + r_n r_{n+g}) = \frac{Sg}{n-g} \quad (9)$$

где сумму Sg можно накапливать в процессе генерации выборки. В случае независимости псевдослучайных чисел для любого $g = 1,2,...n$ должно выполняться условие $R_g \to 0$ при $n \to \infty$, которое означает попарную некоррелированность чисел. Следует обратить внимание на то, что некоррелированность является необходимым, но не достаточным условием независимости случайных величин.

4. Проверка длины периода

Поскольку последовательность чисел на выходе мультипликативного конгруэнтного ГСПЧ периодическая, она не может характеризоваться равномерным распределением и независимостью чисел в строгом смысле, но при большой длине l периода этот недостаток не приводит к ошибкам более существенным, чем, скажем, ограниченная длина разрядной сетки ЭВМ.

Выявление периода последовательности $r_1,...,r_n$ или, что то же самое, в последовательности $x_1,...,x_n$ и определение его длины l – непростая задача, поскольку некоторое число первых членов последовательности может и не принадлежать его периодической части.

Определенную информацию о периодичности можно получить, если запомнить x_1 и в процессе генерации выборки последовательно сравнивать с ним числа x_i, i=2,3,…,n. При первом совпадении $x_i = x_1$ определяется длина периода l = i-1. Если для $i \le n$ совпадения не произошло, то либо $l \ge n$, либо периодическая часть последовательности начинается при i > 1.

Контрольные вопросы

1. Как по выборке случайной величины рассчитываются оценки ее м.о. и дисперсия?
2. Как проверяется равномерность распределения чисел с помощью частотного теста?
3. Как проверяется статистическая независимость чисел на выходе ГСПЧ?

Задание

1. Изучите способы анализа равномерности распределения и независимости чисел на выходе ГСПЧ.

2. Дополните программу из лабораторной работы №1 следующим функционалом:

- тестирование равномерности распределения ГСПЧ частотным методом разбиения на 10 интервалов (построить два графика $f_i(n)$, для n=100 и n=10000, где f_i - частота встречаемости случайной величины на i-интервале);

- проверка равномерности распределения нахождением м.о. и дисперсии по формулам (6) и (7) (для n=100 и n=10000);

- проверка статистической независимости по формуле (8). Построить график Rg(n), для n от 10 до 1000.

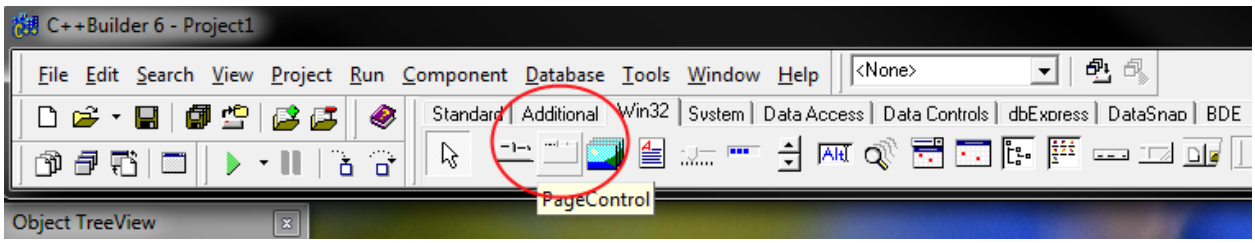
3. По рассчитанным оценкам сделайте вывод о пригодности ГСПЧ. Подберите для ГСПЧ подходящие параметры a, m.

Рекомендации по выполнению работы

Для выполнения данной работы следует обратиться к программе, написанной в лабораторной работе №1 и добавить в программу функции, соответствующие данной работе.

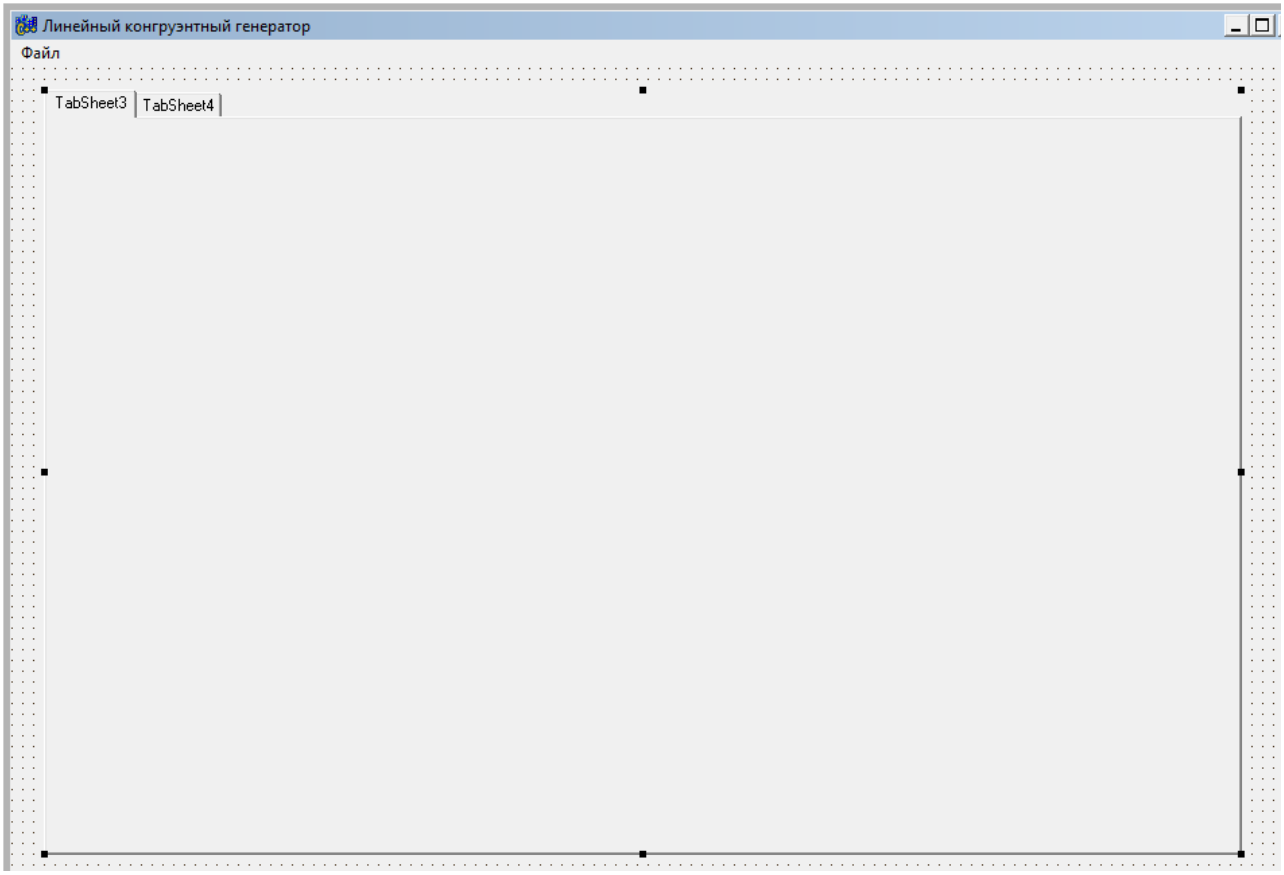
1. Откройте проект с программой в C++ Builder 6 (лабораторная работа №1);

2. Добавьте на форму объект PageControl с вкладки Win32:

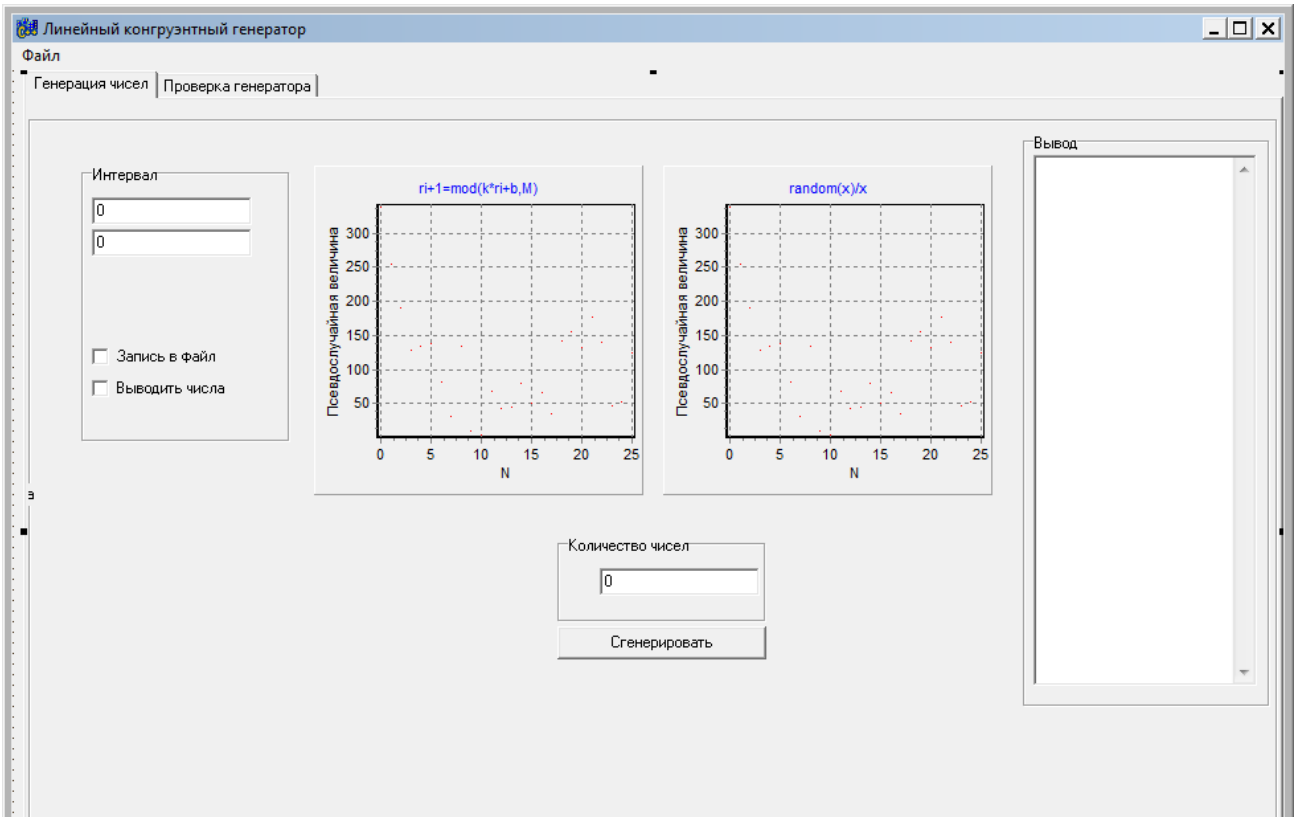


3. Нажмите правой кнопкой мыши на данном объекте и выберите пункт NewPage (для создания двух вкладок – 2 клика);

4. Установите данный объект по размеру формы приложения, оставив свободное место для него перемещением имеющихся объектов в любое свободное пространство на форме;

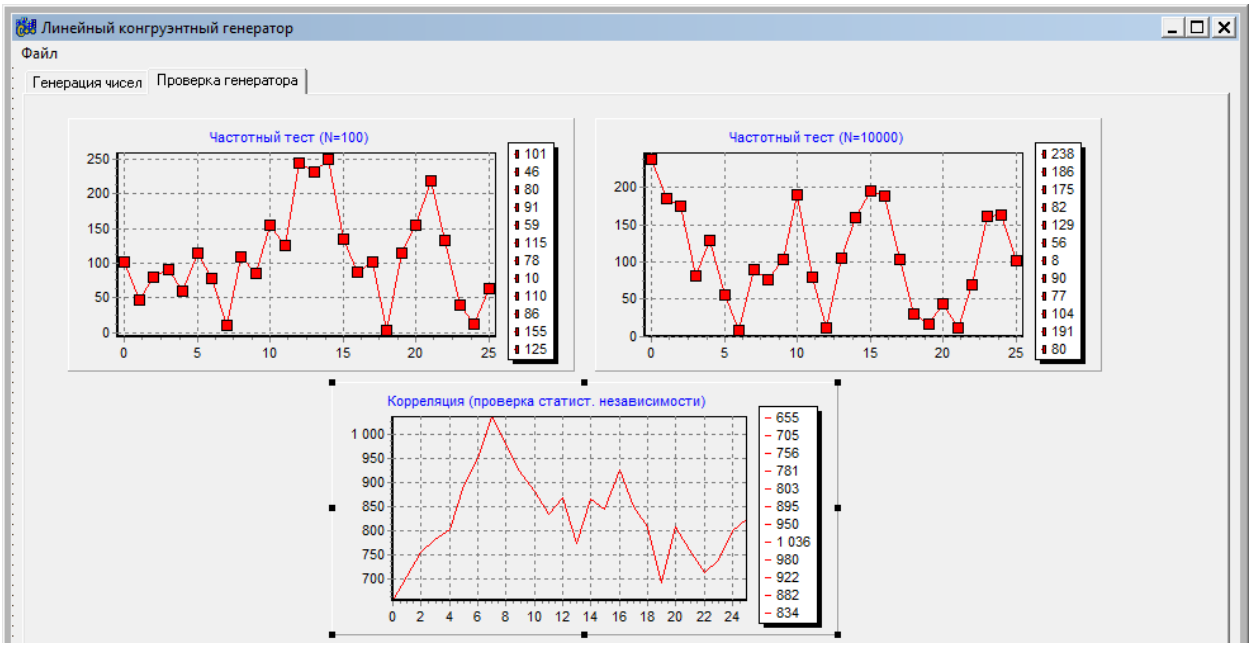


Переместите все существующие объекты на объект PageControl использованием выделения и копированием их в буфер обмена сочетанием клавиш ctrl+x и вставкой в PageControl с помощью ctrl+c. Переименуйте вкладки TabSheet1 и TabSheet2 в «Генерация чисел» и «Проверка генератора» соответственно (для этого кликните по объекту PageControl и слева, в object tree view выберите TabSheet1, переименуйте его в параметре Caption окна Object Inspector на «Генерация чисел»).



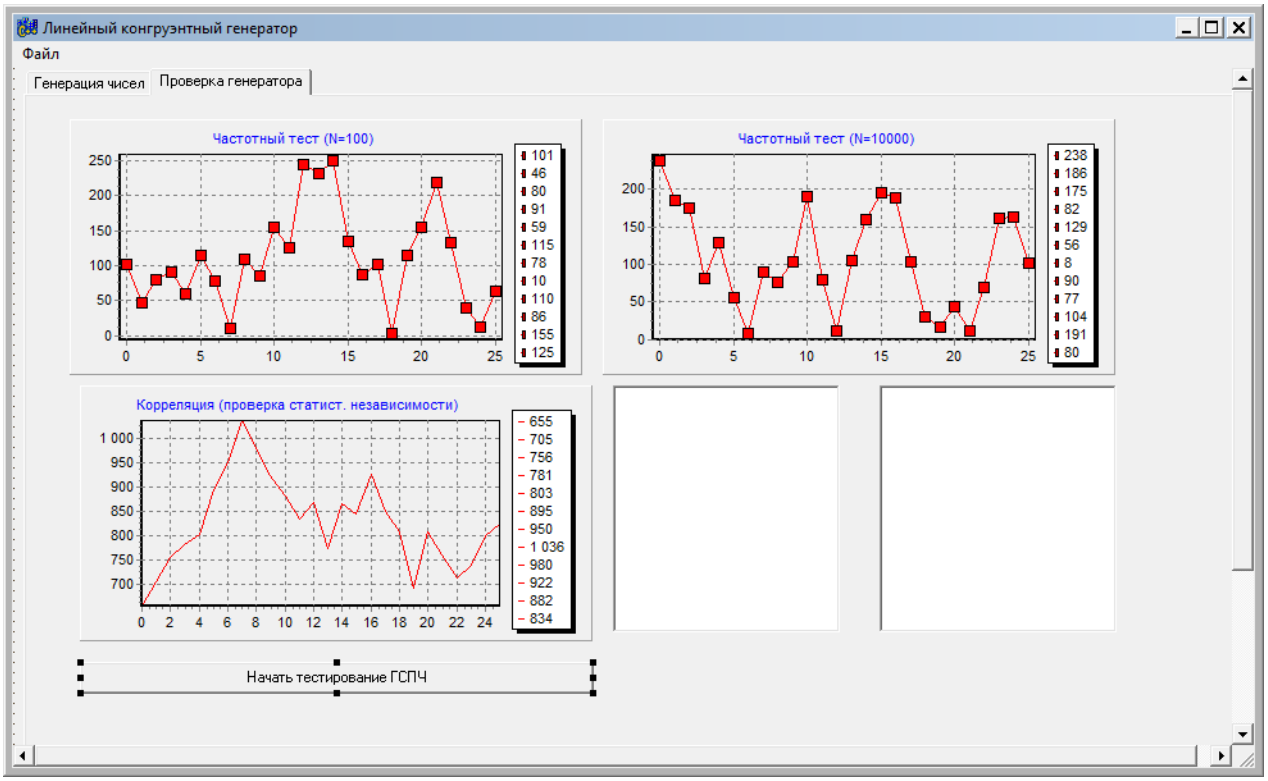
5. Последующие работы будут добавляться наращиванием функционала данной программы добавлением новых вкладок;

6. Перейдите на вкладку «Проверка генератора» и установите 3 новых объекта Chart с добавлением в каждый из них по одному графику Series. Дайте название каждому из графиков как указано на изображении (двойной клик на объекте Chart, на вкладке Titles).



7. Графики будут соответствовать следующим результатам: первые 2 графика для проверки частотным методом. Один график – зависимость частоты встречаемости случайной величины на каждом из интервалов разбиения, от номера интервала для N=100, другой – для N=10000. Третий график будет соответствовать зависимости коэффициента корреляции Rg от N (N следует брать от 10 до 10000).

8. Добавьте на форму кнопку Button, в поле caption дайте ей название («Начать тестирование ГСПЧ»). Добавьте 2 компонента Memo:



9. На кнопку «Начать тестирование ГСПЧ» назначить необходимые действия для заполнения всех трех графиков.

10. Сгенерировать первые 100 псевдослучайных чисел. Код можете скопировать из события кнопки, генерирующей числа из первой лабораторной работы, но с изменением одного параметра – генерация фиксированного количества чисел, равного 100.

11. Код необходимо модифицировать так, чтобы сгенерированные числа записывались в массив. Инициализация массива на 100 чисел в C++ выглядит следующим образом:

```
float randomValues[100];
```

12. По нажатию кнопки заполнение массива псевдослучайными числами можно реализовать следующим образом:

```
for (int i = 0; i < 100; i++)
{
   // Генерируем случайное число Xi
   // Присваиваем i-му элементу массива значение Xi
    randomValues[i] = Xi;
}
```

Проверьте, корректно ли заполнился массив randomValues[i] выводом его содержимого в компоненту Memo:

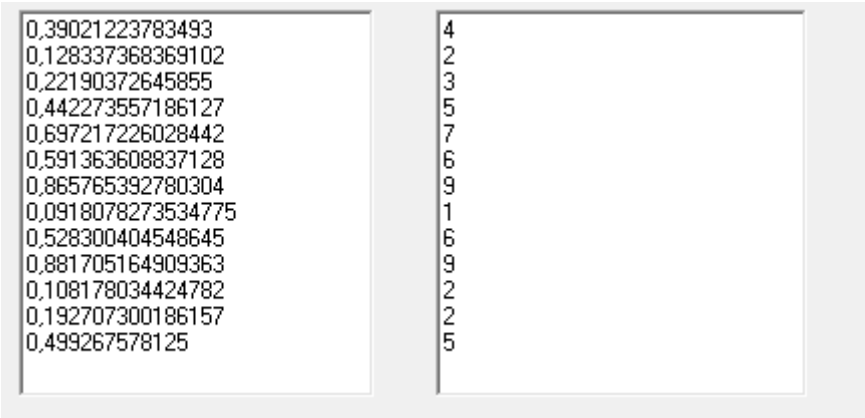
```
for (int i = 0; i < 100; i++)
   Memo2->Lines->Add(randomValues[i]);
```

13. Следующий этап – подсчет количества случайных величин в интервалы равной длины, составляющие всю сгенерированную последовательность. Разобьем последовательность на 10 интервалов: 0-0,1; 0,1-0,2; 0,2-0,3; 0,3-0,4; 0,4-0,5; 0,5-0,6; 0,6-0,7; 0,7-0,8; 0.8-0,9; 0,9-1,0. Реализация подсчета числа попавших чисел в интервалы может быть получена путем нахождения целого числа от умножения элемента randomValues[i] (случайная величина) на 10 и прибавлением единицы. Например, если случайное число «0,25» умножить на 10 и отбросить дробную часть, то мы получим число 2, добавив единицу,

получим номер интервала, равный 3 (действительно, 0,25 находится в интервале 0,2-0,3). Для проверки правильности подсчета, установите еще один компонент Memo и выводите в него значения номеров интервалов. Проинициализировав новый массив countInterval[10], содержащий информацию о каждом из интервалов, можно реализовать подсчет случайных чисел в каждом из интервалов следующим образом (выделение целой части можно сделать указанием *(int)* перед выводом числа):

```
for (int i = 0; i < 100; i++)
{
   Memo2->Lines->Add(randomValues[i]); // Вывод значений
   interval = randomValues[i] * 10 + 1; // Интервал
   Memo3->Lines->Add((int)interval); // Вывод номера
   countInterval[(int)interval] += 1; // Счетчик
}
```

случайная величина номер интервала



14. В массиве countInterval[i] теперь подсчитано количество попавших чисел для каждого из интервалов. Выведем на первый график значения из массива countInterval:

```
 // отношение значений в интервале к общему их числу
for (int i = 0; i < 10; i++)
   Series4->AddXY(i, countInterval[i] / 100);
```

15. Создадим копию блока кода для N = 100 и поместим его следующим за текущим.

16. В скопированном блоке внесем изменения, которые позволят генерировать 10000 значений и подсчитывать их число в каждом из десяти интервалов (заменим в циклах 100 на 10000).

17. При переходе к N=10000 частотная составляющая приближается к вероятности попадания на каждый из интервалов, а именно: 1 / k = 0,1.

18. Поместим на форму четыре компонента Edit для вывода значений м.о. и дисперсии для N = 100 и N = 10000.

19. Найдем значения м.о. для N=100 и N=10000 по формуле (6):

```
Edit9->Text = Mx; // м.о. для 100 чисел
Edit10->Text = Mx2; // м.о. для 10000 чисел
```

20. Сгенерируйте несколько выборок и следите за получаемыми значениями Mx. Для N = 10000 они должны быть ближе к 0,5, чем при N=100.

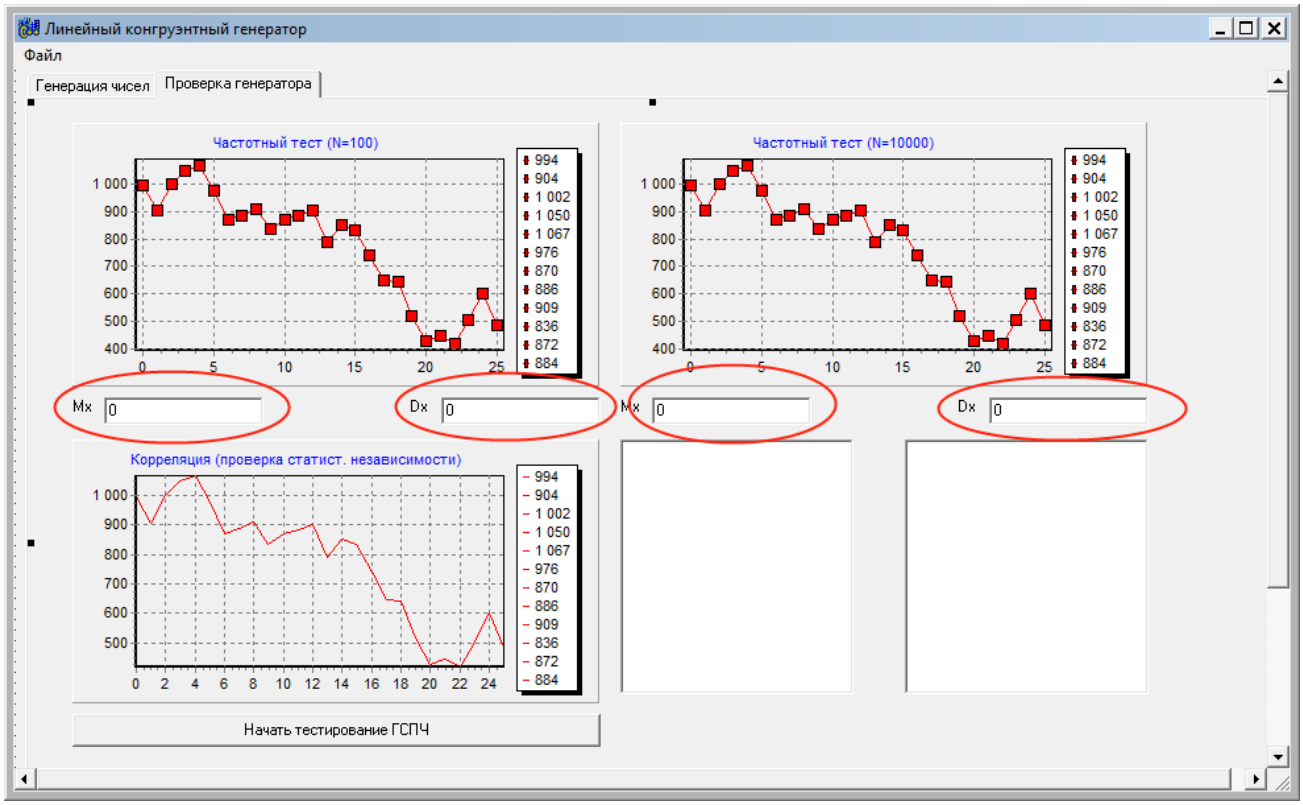
21. В остальные 2 поля выводите значения дисперсии для N = 100 и N = 10000 по формуле (7).

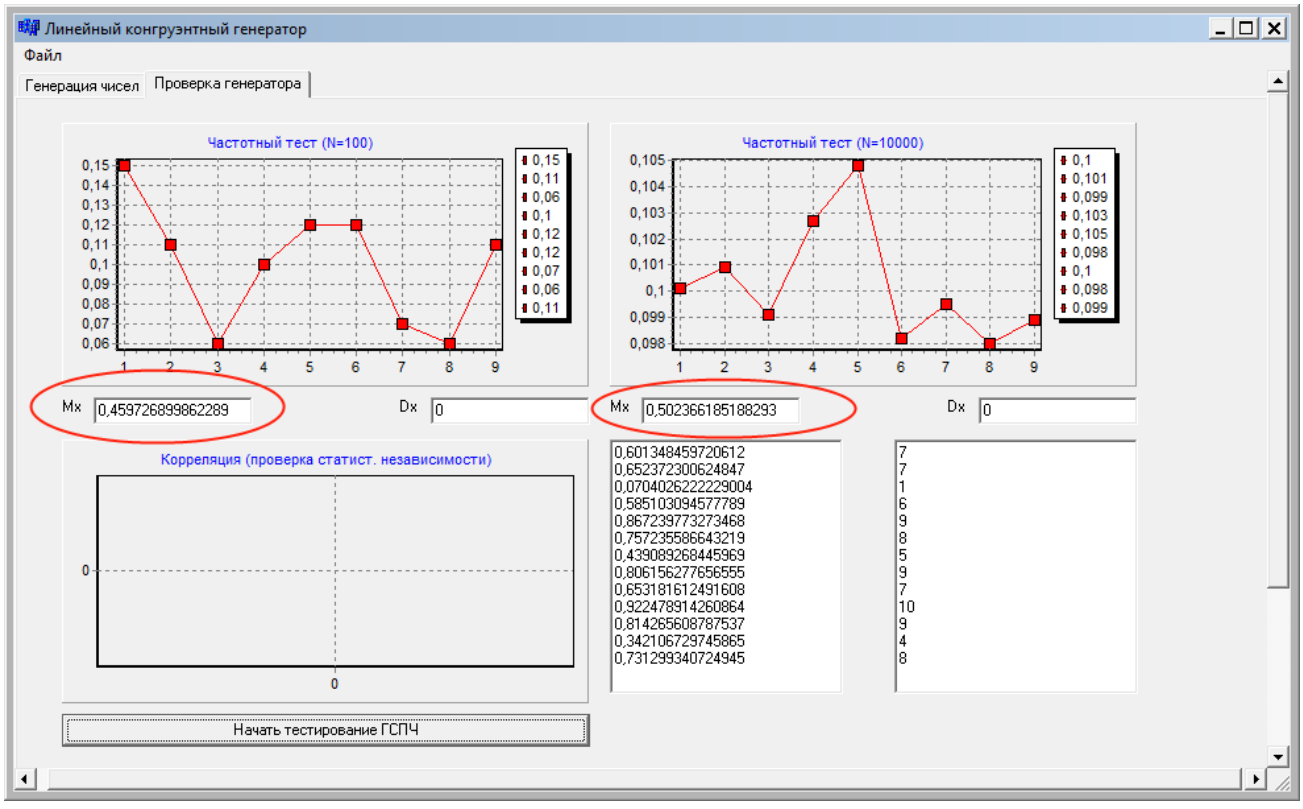
22. Значение дисперсии с ростом N должно стремиться к 0,83333333 (1/12) и быть ближе к нему с выборкой N = 10000.

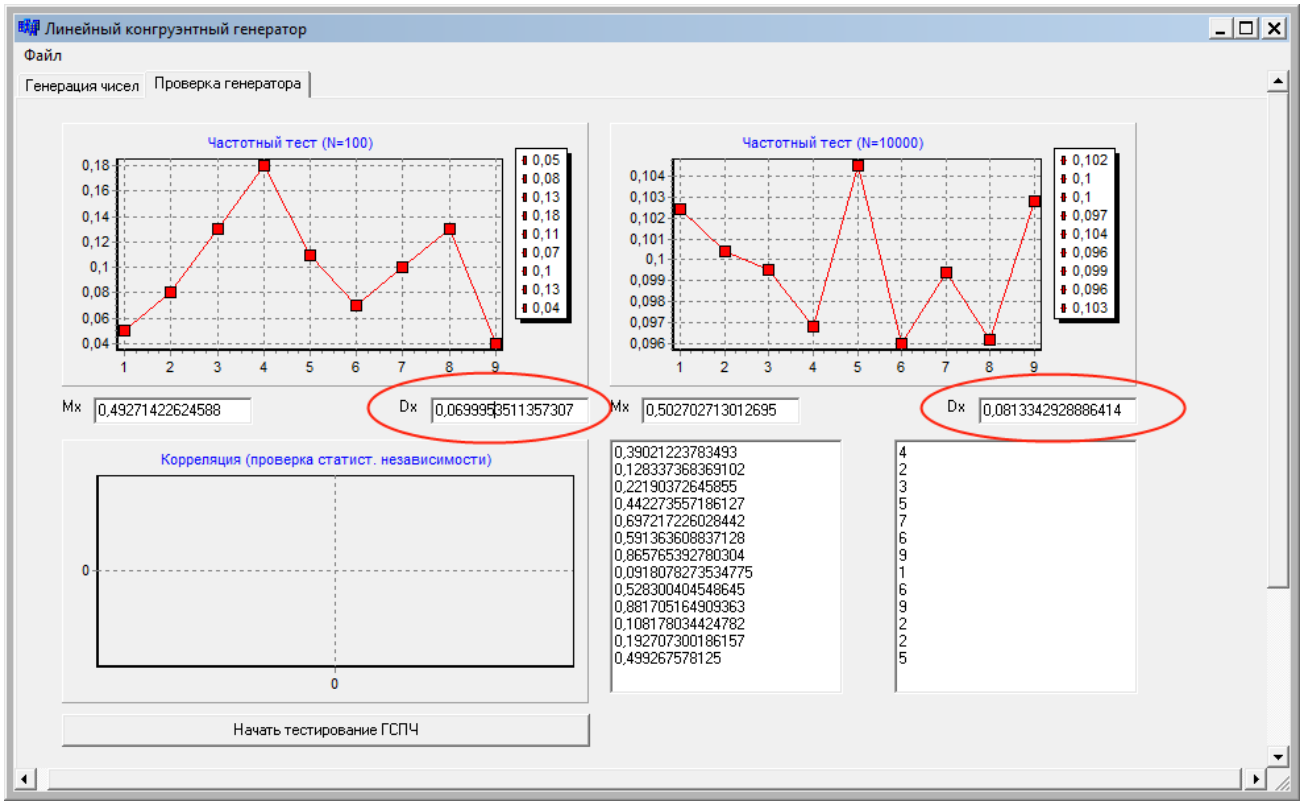
23. Рассчитайте значение Rg корреляции между парами чисел из выборки (g взять равным единице) при N от 10 до 1000 (генерируем 10 чисел, считаем Rg по формуле (8), выводим это значение на график, потом генерируем 11 чисел, считаем Rg и выводим на график, потом 12, 13 чисел и т.д. до 1000). Полученная кривая должна отражать приближение коэффициента корреляции Rg к нулю с повышением значения N.

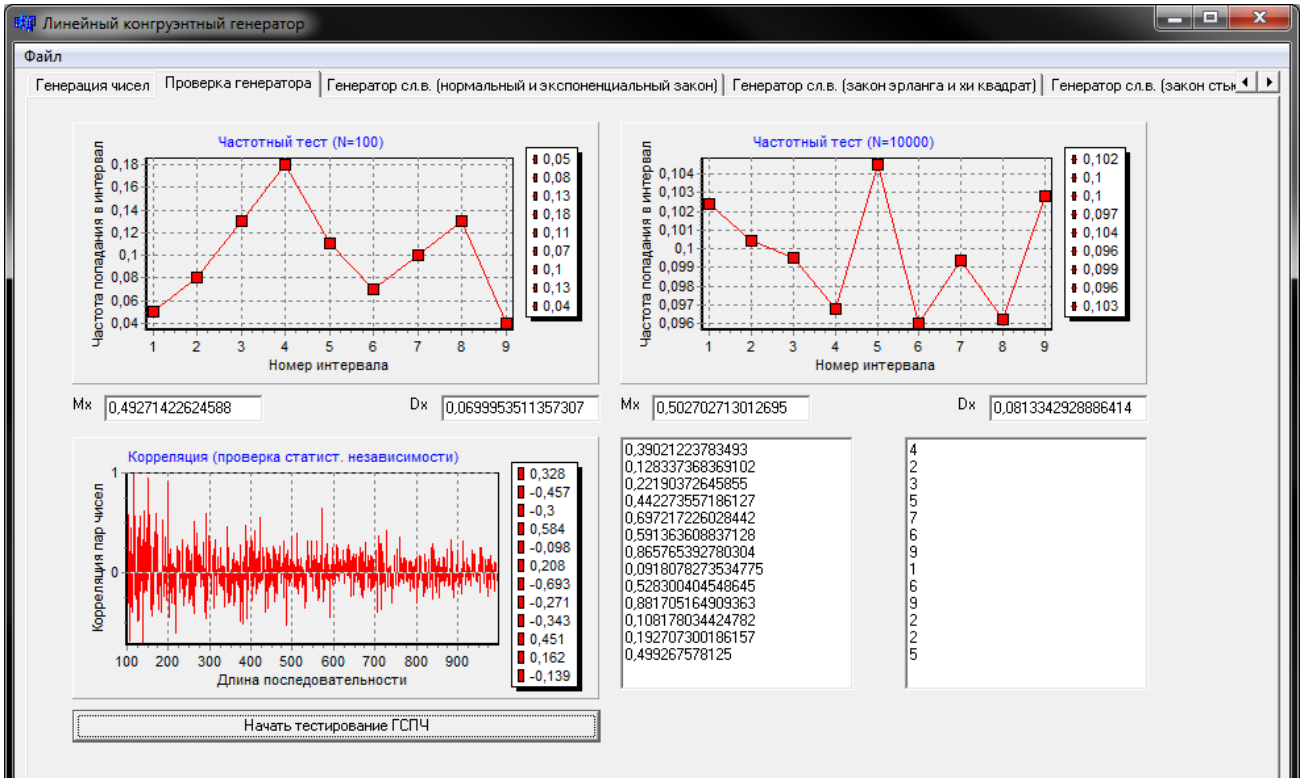
```
float Mx, Dx, Rg, sum=0;
float randomValues[10000], interval[1000];
double Xi = 0;
int m = 2147483647;
int b = 1;
long ri = 7;

for (int i = 0; i < 10000; i++)
   randomValues[i] = 0;
for (int i = 0; i < 1000; i++)
   countInterval[i] = 0;
// Генерация последовательности
for (int i = 0; i < 100; i++)
{
   ri = ri * k;
   ri += b;
   ri = ri % M;
   Xi = (double)ri / M;
   randomValues[i] = Xi;
}
// Подсчет интервалов
for (int i=0; i < 100; i++)
{
   interval = randomValues[i] * 10 + 1;
   countInterval[(int)interval] += 1;
}
for (int i = 1; i < 10; i++)
   Series->AddXY(i, countInterval[i] / 100);
// Mx
sum = 0;
for (int i = 1; i < 100; i++)
   sum = sum + randomValues[i];
Mx = sum / 100;
```

```
// Dx
sum = 0;
for (int i = 1; i < 100; i++)
    sum = sum + randomValues[i] * randomValues[i];
Dx = sum / 100 - Mx * Mx;
// График корреляции Rg(nm);
for (int nm = 100; nm < 1000; nm += 1)
{
    for (int i = 0; i < nm; i++)
    {
        ri = ri * k;
        ri += b;
        ri = ri % M;
        Xi = (double)ri / M;
        randomValues[i] = Xi;
    }
    sum = 0;
    for (int i = 0; i < nm; i++)
        sum = sum + randomValues[i] * randomValues[i + 1];
    Mx = (sum / (nm - 1));
    Rg = 12 * Mx - 3;
    Series2->AddXY(nm, Rg);
}
```


Линейный конгруэнтный генератор
Файл
Генерация чисел
Проверка генератора
Частотный тест (N=100)
Частотный тест (N=10000)
994
904
1 002
1 050
1 067
976
870
886
909
836
872
884
Mx
0
Dx
0
Корреляция (проверка статист. независимости)
Начать тестирование ГСПЧ


Линейный конгруэнтный генератор
Файл
Генерация чисел
Проверка генератора
Частотный тест (N=100)
Частотный тест (N=10000)
0,15
0,11
0,06
0,1
0,12
0,12
0,07
0,06
0,11
0,105
0,104
0,103
0,102
0,101
0,1
0,099
0,098
0,101
0,099
0,103
0,105
0,098
0,1
0,098
0,099
Mx
0,459726899862289
Dx
0
Mx
0,502366185188293
Dx
0
Корреляция (проверка статист. независимости)
0,601348459720612
0,652372300624847
0,0704026222229004
0,585103094577789
0,867239773273468
0,757235586643219
0,439089268445969
0,806156277656555
0,653181612491608
0,922478914260864
0,814265608787537
0,342106729745865
0,731299340724945
7
7
1
6
9
8
5
9
7
10
9
4
8
Начать тестирование ГСПЧ


Линейный конгруэнтный генератор
Файл
Генерация чисел
Проверка генератора
Частотный тест (N=100)
0,18
0,16
0,14
0,12
0,1
0,08
0,06
0,04
1 2 3 4 5 6 7 8 9
0,05
0,08
0,13
0,18
0,11
0,07
0,1
0,13
0,04
Частотный тест (N=10000)
0,104
0,103
0,102
0,101
0,1
0,099
0,098
0,097
0,096
1 2 3 4 5 6 7 8 9
0,102
0,1
0,1
0,097
0,104
0,096
0,099
0,096
0,103
Mx
0,49271422624588
Dx
0,0699953511357307
Mx
0,502702713012695
Dx
0,0813342928886414
Корреляция (проверка статист. независимости)
0
0
0,39021223783493
0,128337368369102
0,22190372645855
0,442273557186127
0,697217226028442
0,591363608837128
0,865765392780304
0,0918078273534775
0,528300404548645
0,881705164909363
0,108178034424782
0,192707300186157
0,499267578125
4
2
3
5
7
6
9
1
6
9
2
2
5
Начать тестирование ГСПЧ


Линейный конгруэнтный генератор
Файл
Генерация чисел
Проверка генератора
Генератор сл.в. (нормальный и экспоненциальный закон)
Генератор сл.в. (закон эрланга и хи квадрат)
Генератор сл.в. (закон
Частотный тест (N=100)
Частота попадания в интервал
0,18
0,16
0,14
0,12
0,1
0,08
0,06
0,04
1 2 3 4 5 6 7 8 9
Номер интервала
0,05
0,08
0,13
0,18
0,11
0,07
0,1
0,13
0,04
Частотный тест (N=10000)
Частота попадания в интервал
0,104
0,103
0,102
0,101
0,1
0,099
0,098
0,097
0,096
1 2 3 4 5 6 7 8 9
Номер интервала
0,102
0,1
0,1
0,097
0,104
0,096
0,099
0,096
0,103
Mx
0,49271422624588
Dx
0,0699953511357307
Mx
0,502702713012695
Dx
0,0813342928886414
Корреляция (проверка статист. независимости)
Корреляция пар чисел
1
0
100 200 300 400 500 600 700 800 900
Длина последовательности
0,328
-0,457
-0,3
0,584
-0,098
0,208
-0,693
-0,271
-0,343
0,451
0,162
-0,139
0,39021223783493
0,128337368369102
0,22190372645855
0,442273557186127
0,697217226028442
0,591363608837128
0,865765392780304
0,0918078273534775
0,528300404548645
0,881705164909363
0,108178034424782
0,192707300186157
0,499267578125
4
2
3
5
7
6
9
1
6
9
2
2
5
Начать тестирование ГСПЧ

Синтез генераторов случайных величин с заданными законами распределения

Цель работы

Разработка программного генератора псевдослучайных величин, имеющих заданный закон распределения.

Краткие сведения из теории

1. Общая схема реализации случайных величин

Случайная величина (сл.в.) с заданным законом распределения вероятностей всегда может быть получена путем подходящего преобразования стандартного случайного числа. При моделировании на ЭВМ такое подходящее преобразование выполняется над стандартным псевдослучайным числом и в результате получается псевдослучайная величина с требуемым распределением.

2. Метод обращения

Методом обращения можно получить сл.в. x с любой функцией распределения вероятностей (ф.р.в.) $F(t)$ имеющей обратную функцию F^{-1}. Метод обращения сводится к тому, чтобы сгенерировать стандартное случайное число z и вычислить x по формуле

$$x = F^{-1}(z) \qquad (10)$$

Тогда сл.в. x будет иметь распределение $F(t)$.

Если на «вход» формулы (10) подавать последовательность $z_1, z_2 \ldots$ стандартных случайных чисел, то на «выходе» получится последовательность $x_1, x_2 \ldots$ независимых сл.в., имеющих ф.р.в. $F(t)$.

Для конкретной ф.р.в. F выражение обратной функции $F^{-1}(z)$ можно получить известным из школьного курса математики способом, состоящим в разрешении уравнения $F(x) = z$ относительно x.

Докажем метод обращения. Пусть F – некоторая ф.р.в. Тогда F – монотонно неубывающая функция, принимающая значения в интервале [0,1]. Пусть она имеет обратную функцию F^{-1}. Определим x через стандартную сл.в. z в виде $x = F^{-1}(z)$ и найдем ф.р.в. $F_x(t)$ сл.в. x. По определению $F_x(t) = P[x \le t]$ (P – вероятность). Но $P[x \le t] = P[F^{-1}(z) \le t] = P[z \le F(t)] = F(t)$. Таким образом, $F_x(t) = F(t)$, что и требовалось.

В качестве примера рассмотрим использование метода обращения для реализации экспоненциальной сл.в.

3. Реализация экспоненциальной сл.в. методом обращения

Ф.р.в. экспоненциальной сл.в. x_{exp} имеет вид:

$$F(t) = \begin{cases} 1 - e^{-ut}, t \ge 0 \\ 0 \qquad , t < 0 \end{cases} \tag{11}$$

$$M[x_{\text{exp}}] = 1/u,\ D[x_{\text{exp}}] = 1/u^2$$

В соответствии с методом обращения запишем уравнение $z = F(x_{\text{exp}})$ или, с учетом (11):

$$z = 1 - e^{ux_{\text{exp}}} \quad (t \ge 0)$$

и, решая его относительно x_{exp} найдем формулу для генерации сл.в. x_{exp}

$$x_{\text{exp}} = -\frac{1}{\text{u}} \ln(1 - z) \tag{12}$$

В полученной формуле разность 1-z, как и сама величина z, является стандартным случайным числом. Этот факт позволяет упростить формулу для генерации x_{exp}, полагая

$$x_{\text{exp}} = -\frac{1}{\text{u}} \ln(z) \tag{13}$$

Назовем экспоненциальную сл.в. с м.о., равным единице, нормированной и обозначим через x_{exp}. Тогда для x_{exp} u=1 и для генерации x_{exp} можно использовать формулу

$$x_{\text{exp}} = -\ln(z) \tag{14}$$

Под линейным преобразованием (ЛП) сл.в. x будем понимать сл.в. y, полученную из x умножением на константу A и добавлением константы B:

$y = Ax + B$

Очевидно,

$$M[y] = M[Ax + B] = AM[x] + B,$$

$$D[y] = D[Ax + B] = A^2 D[x] \qquad (15)$$

При построении генераторов сл.в. часто используется свойство ЛП y сохранять вид закона распределения сл.в. x. Так, например, если экспоненциальную сл.в. умножить на константу, то получится снова экспоненциальная сл.в. Нормальная или равномерно распределенная сл.в. не меняет вида закона распределения как при умножении на константу, так и при добавлении константы.

Генератор (14) нормированной экспоненциальной сл.в. x_{exp} достаточен для реализации экспоненциальных сл.в. с любыми м.о. $M \geq 0$. Для получения экспоненциальной сл.в. x_{exp} с м.о. M достаточно вычислить ее в виде $x_{\text{exp}} = M * x_{\text{exp}}$.

Сл.в. $y = Az + B$, являющаяся ЛП стандартной сл.в. z, имет равномерное распределение на интервале *(B, A+B)* и, в соответствии с (15), имеет $M[y] = A/2 + B$, $D[y] = A^2/12$.

4. Реализация нормальной сл.в. методом суммирования

Плотность распределения вероятностей нормальной сл.в. x с м.о. M и дисперсией $D = \delta^2$ имеет вид:

$$f(t) = \frac{1}{\sqrt{2\pi D}} e^{-\frac{(t-M)^2}{2D}} \qquad (16)$$

Соответствующая ф.р.в.

$$F(t) = \int_{-\infty}^{t} f(s)ds$$

Не выражается в элементарных функциях, что затрудняет непосредственное использование метода обращения. Поэтому для реализации

нормальной сл.в. часто применяют метод суммирования, основанный на центральной предельной теореме теории вероятностей. Из этой теоремы вытекает, что сумма Sn достаточно большого числа n стандартных случайных чисел z_i будет иметь распределение, достаточно близкое к нормальному.

Поскольку м.о. суммы Sn и ее дисперсия равны сумме м.о. и сумме дисперсий ее слагаемых, то $M[Sn] = n/2, D[Sn] = n/12$. Применим к Sn линейное преобразование, чтобы получить нормированную центрированную сл.в. X (с м.о. 0 и дисперсией 1). Очевидно для этого надо положить $x = \frac{(S_n - n/2)}{\sqrt{n/12}}$ или, заменяя Sn,

$$x = \frac{(z_1 + ... + z_n) - n/2}{\sqrt{n/12}} \qquad (17)$$

На практике часто принимают n=12, т.к. во-первых, при таком числе слагаемых X имеет распределение, весьма близкое к нормальному, и, во-вторых, значительно упрощается вид формулы (17). При n=12 из (17) получаем формулу для генерации нормированной центрированной сл.в.

$$x_{no} = (z_1 + ... + z_{12}) - 6 \qquad (18)$$

Нормальную сл.в. с м.о. M и дисперсией δ^2 можно получить из x_{no} по формуле:

$$x_{no} = \sqrt{D}x_{no} + M$$

В отличие от метода обращения, метод суммирования является приближенным.

5. Применение функциональных преобразований сл.в.

Часто для реализации сл.в., имеющих сложные законы распределения, можно применять функциональные преобразования сл.в. с простыми распределениями. Это связано с тем, что многие сл.в., широко применяемые в теории вероятностей и математической статистике, вводятся именно как

функции других сл.в., которые имеют известные простые законы распределения.

Так, например, эрланговская сл.в., имеющая распределение Эрланга k-го порядка, вводится как сумма k независимых экспоненциальных сл.в. $x_{\exp i}$, которые имеют одинаковые м.о.:

$$x_{erl} = (x_{\exp 1} + ... + x_{\exp k}) \tag{19}$$

Эту формулу можно непосредственно использовать для генерации x_{erl} при наличии генератора для $x_{\exp}$. Если в (19) суммируются нормированные сл.в. $x_{erl} = x_{\exp}$, то, очевидно, $M[x_{erl}] = k * M[x_{\exp}] = k/u$, $D[x_{erl}] = k * D[x_{\exp}] = k/u^2$

Аналогично сл.в. χ^2 (хи-квадрат), имеющая распределение с k степенями свободы, вводится как сумма k независимых сл.в. x_{noi}, имеющих нормированное центрированное нормальное распределение:

$$\chi^2 = {x_{no1}}^2 + ... + {x_{nok}}^2 \tag{20}$$

$$M[\chi^2] = k, D[\chi^2] = 2k.$$

Из (20) ясно, как она может быть реализована при моделировании на ЭВМ.

Распределением Стьюдента с k степенями свободы называют распределение сл.в.

$$x_{stu} = \frac{x_{no0}}{\sqrt{({x^2}_{no1} + ... + {x^2}_{nok})/k}} \tag{21}$$

где $x_{no0},...,x_{nok}$ - независимые нормированные центрированные нормальные величины.

6. Генераторы дискретных сл.в.

Если x – дискретная сл.в., то ее закон распределения вероятностей задается обычно путем указания вероятностей p1, p2… для каждого ее возможного значения x1, x2… соответственно.

Для реализации дискретной сл.в. интервал (0,1) разбивают на интервалы $\delta_1, \delta_2, ...,$ имеющие длину $|\delta|_1 = p_1, |\delta|_2 = p_2, ...$ Такое разбиение всегда возможно,

поскольку для дискретной сл.в. p1 + p2 + … = 1. Далее сл.в. x может быть сгенерирована по следующему алгоритму:

1. генерируется стандартная сл.в. z;
2. определяется номер i интервала δ_i, в который попало значение z;
3. принимается $x = x_i$

В зависимости от значения z сл.в. x будет принимать различные значения из множества $x_1, x_2, ..., x_i$. При этом вероятность $P[x_i]$ принять некоторое значение x_i равно, очевидно, вероятности того, что z попадет в интервал δ_i, т.е. $P[x_i] = |\delta_i| / |(0,1)|$. Следовательно $P[x_i] = P_i$ для всех i=1, 2, 3… Это доказывает, что реализуется сл.в. x с требуемым распределением вероятностей.

При моделировании на ЭВМ вместо стандартной сл.в. z используется псевдослучайное число r. Строится сетка из значений вероятностей в зависимости от интервала i, точки которой рассчитываются по формуле:

$$P_i = \frac{M^i}{i!} e^{-M},\ i = 0,1,2,... \qquad (22)$$

Затем генерируется псевдослучайное число r и определяется, в какой интервал оно попало. Очевидно, формула Пуассона подразумевает различные длины интервалов, и в какие-то из них псевдослучайная величина будет попадать чаще, что и позволит построить плотность распределения Пуассона.

Для того, чтобы запрограммировать генерацию чисел по любому закону распределения д.с.в. (в данном случае – Пуассоновский), необходимо:

1. Сгенерировать псевдослучайное число Xi (код можно взять из лаб.1 и 2);

2. После генерации числа, создать цикл while с условием «пока сгенерированное число Xi больше Pi»;

3. В цикле while рассчитывать значение Pi по формуле (22), с каждым последующим i значение Pi будет расти до тех пор, пока не достигнет Xi;

4. Когда значение Pi превысит сгенерированное число Xi, реализовать выход из цикла;

5. После выхода из цикла while следует зафиксировать последнее значение i (для Pi) и отразить на графике это значение i по горизонтальной оси, и соответствующее ему значение Xi.

6. Повторить пункты 1-5 10000 раз.

Аналогично реализуются генераторы других дискретных псевдослучайных величин, например, бернуллиевская сл.в. x_{ber} с распределением вероятностей:

$P_i = C^i{}_n p^i (1-p)^{n-i},\ i = 0,1,\ldots, n$

для нее $M[x_{ber}] = np, D[x_{ber}] = np(p-1)$

Либо, геометрическое распределение:

$P_i = P^i(1-p), i = 0,1,\ldots n$. В данном случае $M[x_{geom}] = p/(1-p), M[x_{geom}] = p/(1-p)^2$

Контрольные вопросы

1. В чем заключается общий принцип реализации случайных (псевдослучайных) величин с заданными законами распределения вероятностей?
2. Как реализуется случайная величина по методу обращения?
3. Как при генерации сл.в. используют линейные преобразования?
4. Как можно реализовать нормальную сл.в.?
5. В каких случаях для реализации сл.в. со сложными законами распределения используются функциональные преобразования сл.в., имеющие более простые распределения?
6. Каким образом можно построить генератор дискретной случайной величины?

Задание

- Изучите методы синтеза генераторов сл.в. с заданными законами распределения.

- Напишите программу, реализующую генератор псевдослучайной величины, закон распределения которой указан в табл.1 (для вашего номера варианта), а также генератор чисел, распределенных по закону Пуассона.

Обеспечьте в программе:

- Генерацию n значений псевдослучайной величины (рекомендуется n = 10000).

- Определение и вывод частот встречаемости случайной величины (плотность распределения) на каждом из интервалов и их последующее сравнение со значениями в табл. 2 и табл. 3.

- По результатам сравнения с табличными значениями сделайте вывод о соответствии характера распределения случайных величин требуемому закону распределения. Разница между сгенерированными значениями и табличными не должна превышать 0,1.

Табл. 1

Вариант	**Название распределения вероятностей**	**Закон распределения**	**Параметры распределения**
1.	Нормальное	$f(t)=\frac{1}{\sqrt{2\pi D}}e^{-\frac{(t-M)^2}{2D}}$	M – любое (удобнее брать M>3, чтобы избежать работы с отрицательными числами) D=1
2.	Хи-квадрат с k степенями свободы	$f(t)=\frac{1}{2^{k/2}r(k/2)}t^{k/2-1}e^{-t/2}$	k=4
3.	Стьюдента с k степенями свободы	$f(t)=\frac{r((k+1)/2)}{\sqrt{nk}r(k/2)}(1+t^2/k)^{-(k+1)/2}$	k=4
4.	Экспоненциальное	$f(t)=ue^{-ut}, t\geq 0$	u=1
5.	Эрланга k-го порядка	$f(t)=\frac{u^k}{(k-1)!}t^{k-1}e^{-ut}, t\geq 0$	k=3 u=1
Для всех	Пуассона	$P_i=\frac{M^i}{i!}e^{-M},\ i=0,1,2,...$	M=4

Табл. 2

(-1,0]	(0,1]	(1,2]	(2,3]	(3,4]	(4,5]	(5,6]	(6,7]	(7,8]	(8,9]	(9,10]
Нормальное распределение, M=1, D=1:										
.3413	.3413	.1359	.0214	.0013	.00003	-	-	-	-	-
Хи-квадрат:										
-	.0902	.1740	.1780	.1518	.1187	.0881	.0633	.0443	.0305	.0207
Стьюдента с 4 степенями свободы:										
.3131	.3131	.1288	.0381	.0119	.0044	.0018	.0008	.0004	-	-
Экспоненциальное, u=1:										
-	.6321	.2325	.0855	.0315	.0116	.0043	.0016	.0006	.0002	.0001
Эрланга 3 порядка, u=1:										
-	.0803	.2430	.2535	.1851	.1135	.0627	.0323	.0159	.0075	.0035

Табл. 3

Вероятности значений 0,…,10 дискретных сл.в.										
0	1	2	3	4	5	6	7	8	9	10
Распределение Пуассона, m=4:										
.0183	.0733	.1465	.1954	.1954	.1563	.1042	.0595	.0298	.0132	.0053

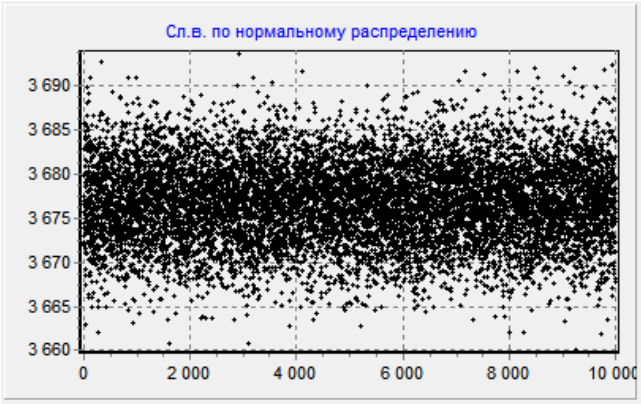
```
// Генерация значений по нормальному закону распределения
for (int j = 0; j < 50; j++)
{
    for (int i = 0; i < 12; i++)
    {
        ri = ri * k;
        ri += b;
        ri = ri % M;
        randomValue = (double)ri / M;
        randomValues[i] = randomValue;
        sum += randomValues[i];
    }
    randomSum[(int)sum] += 1;
    sum = sum * sqrt(20) + 3650; // Mx = 3375, Dx = 4729
    Memo4->Lines->Add(sum);
     // График значений
    Series->AddXY(j, sum);
    sum = 0;
}
// График распределения
for (int i = 0; i < 12; i++)
    Series2->AddXY(i, randomSum[i] / 10000);

// Генерация значений по экспоненциальному закону
for (int i = 0; i < 12; i++)
    randomSum[i] = 0;
for (int i = 0; i < 10000; i++)
{
    ri = ri * k;
    ri += b;
    ri = ri % M;
    randomValue = (double)ri / M;
    randomValues[i] = randomValue;
```

```
    sum = -log(randomValues[i]);
    // График значений
    Series3->AddXY(i, sum);
    randomSum[(int)sum] += 1;
 }
 // График распределения
 for (int i = 0; i < 10; i++)
    Series4->AddXY(i, randomSum[i] / 10000);

// Генерация значений по распределению Эрланга
 for (int i = 0; i < 10000; i++)
    randomSum[i] = 0;
 for (int j = 0; j < 10000; j++)
 {
    for (int i = 0; i < 3; i++)
    {
        ri = ri * k;
        ri += b;
        ri = ri % M;
        randomValue = (double)ri / M;
        randomValues[i] = randomValue;
        sum += -log(randomValues[i]);
    }
     // График значений
    Series5->AddXY(j, sum);
    randomSum[(int)sum] += 1;
    sum = 0;
 }
 // График распределения
 for (int i = 0; i < 10; i++)
    Series6->AddXY(i, randomSum[i] / 10000);
```

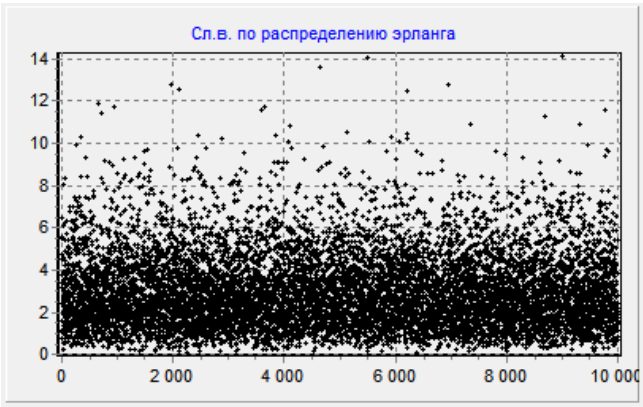
```
// Генерация значений по распределению Хи-квадрат (k = 4)
for (int j = 0; j < 10000; j++)
{
    for (int k = 0; k < 4; k++)
    {
        for (int i = 0; i < 12; i++) // Получение noi
        {
            ri = ri * k;
            ri += b;
            ri = ri % M;
            randomValue = (double)ri / M;
            randomValues[i] = randomValue;
            sum += randomValues[i];
        }
        sum = sum - 6; // (zo1+...zo12) - 6 = noi
        sm = sum * sum;
        smm += sm;
    }
     // График значений
    Series7->AddXY(j, smm);
    randomSum[(int)smm] += 1;
    sum = 0;
    sm = 0;
    smm = 0;
}
// График распределения
for (int i = 0; i < 10; i++)
    Series8->AddXY(i, randomSum[i] / 10000 / 2);
```

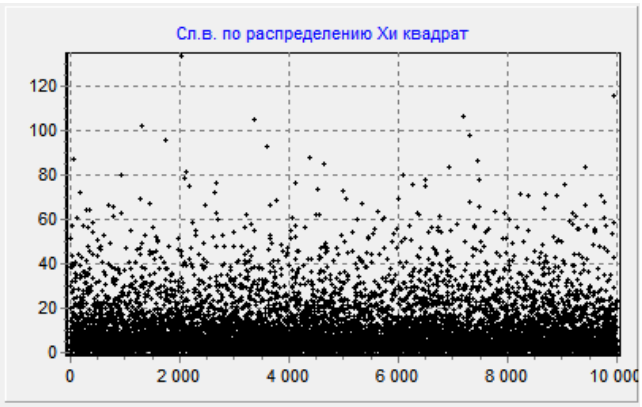
```
// Генерация значений по распределению Стьюдента (k = 4)
for (int j = 0; j < 10000; j++)
 {
    for (int k = 0; k < 4; k++)
    {
        for (int i = 0; i < 12; i++) // Получение noi
        {
            ri = ri * k;
            ri += b;
            ri = ri % M;
            randomValue = (double)ri / M;
            randomValues[i] = randomValue;
            sum += randomValues[i];
        }
        sum = sum - 6; // (zo1+...zo12) - 6 = noi
        if (k == 0)
            no_0 = sum;
        sm = sum * sum;
        smm += sm;
    }
    smm = no_0 / sqrt(smm / 4);
    smm += 6; // Mx=6
    // График значений
    Series9->AddXY(j, smm);
    randomSum[(int)smm] += 1;
    sum=0;
    sm=0;
    smm=0;
 }
 // График распределения
 for (int i = 0; i < 10; i++)
    Series10->AddXY(i, randomSum[i] / 10000);
```

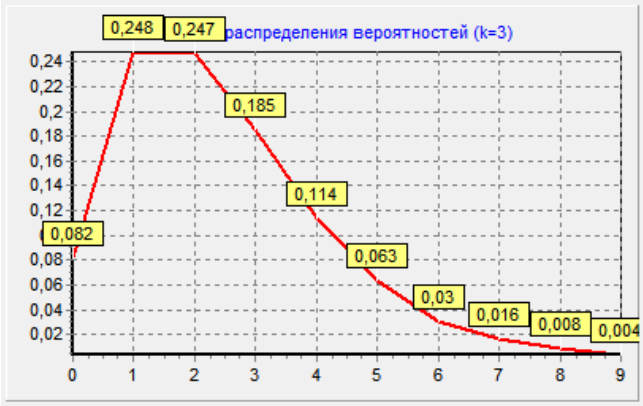
```
// Генерация значений по распределению Пуассона
double p, sum, sm, smm = 0;
int xpuass;
int countInterval[10];

for (int i = 0; i < 10; i++)
   countInterval[i] = 0;
for (int i = 0; i < 10000; i++)
{
   ri = ri * k;
   ri += b;
   ri = ri % M;
   randomValue = (double)ri / M;
   randomValues[i] = randomValue;
   p = exp(-4);
   sum = p;
   xpuass = 0;
   while (randomValue > sum)
   {
       xpuass += 1;
       p = 4 * p / xpuass;
       sum = sum + p;
   }
   // График значений
   Series11->AddXY(xpuass, randomValue);
   countInterval[xpuass] += 1;
   sum = 0;
}
// График распределения
for (int i = 0; i < 10; i++)
   Series12->AddXY(i, countInterval[i] / 10000);
```

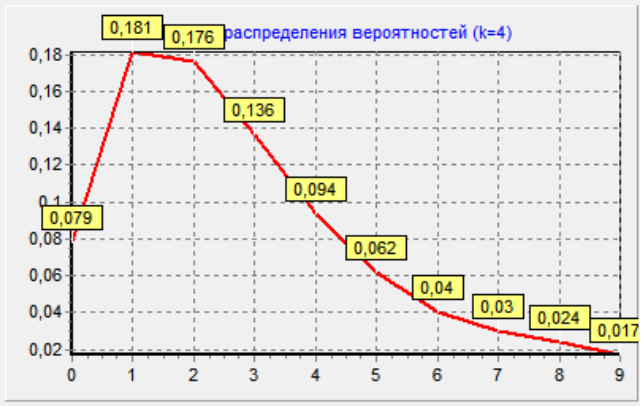


Генерация чисел

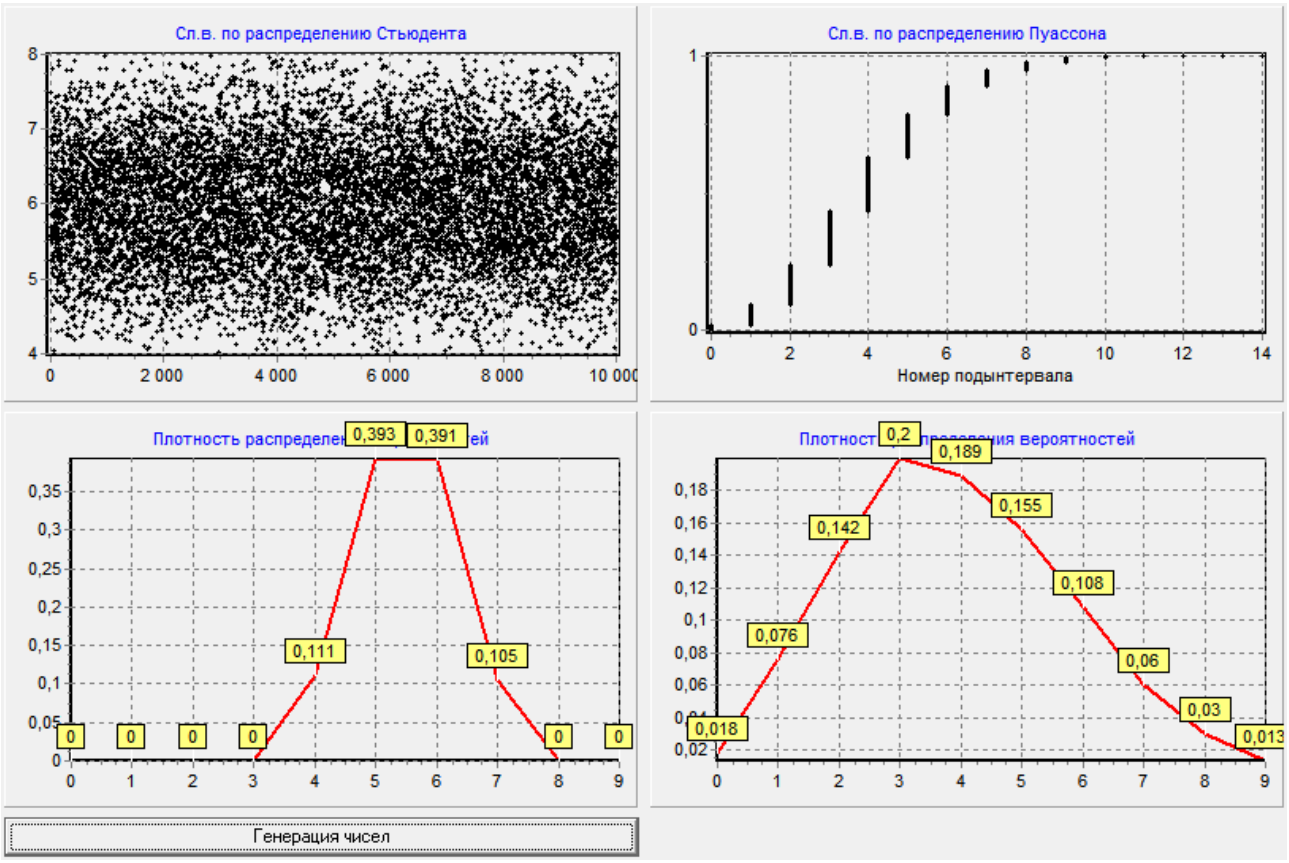








Генерация чисел


Сл.в. по распределению Стьюдента
Сл.в. по распределению Пуассона
Номер подынтервала
Плотность распределен
ей
Плотност
ределения вероятностей
0,393
0,391
0,111
0,105
0,2
0,189
0,155
0,142
0,108
0,076
0,06
0,03
0,018
0,013
Генерация чисел

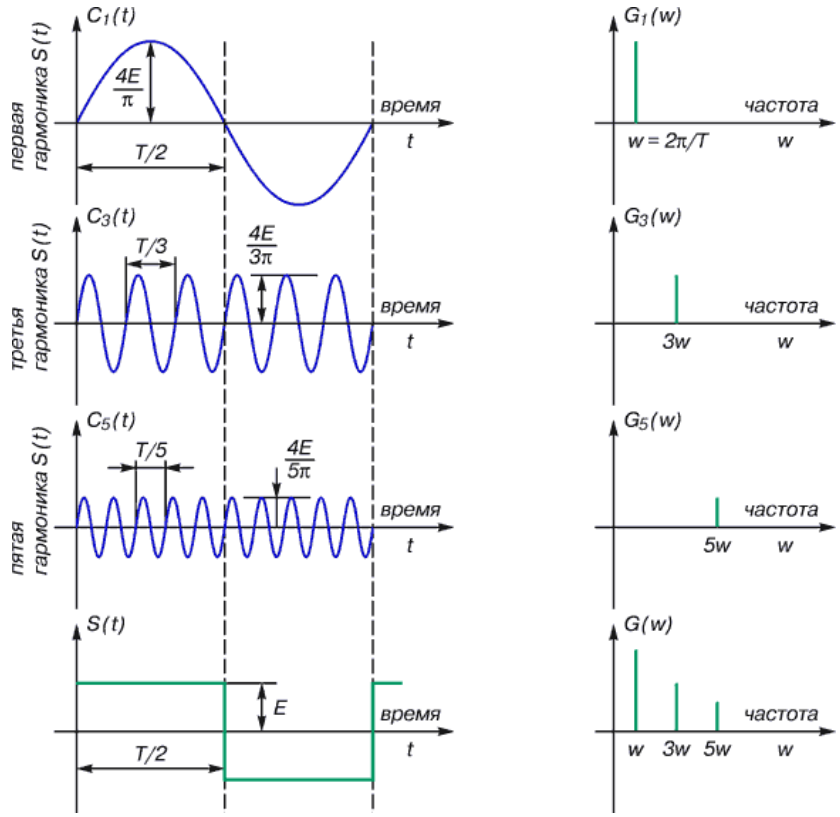
Генератор гармонического сигнала и дискретное преобразование Фурье

Цель работы

Разработка программного генератора гармонического сигнала заданной частоты с функцией разложения сигнала в спектр с помощью дискретного преобразования Фурье (ДПФ).

Краткие сведения из теории

В основе преобразования Фурье (ПФ) лежит чрезвычайно простая, но исключительно плодотворная идея – почти любую периодическую функцию можно представить суммой отдельных гармонических составляющих (синусоид и косинусоид с различными амплитудами A, периодами T и, следовательно, частотами ω). Пример одной из таких функций S(t), состоящей из гармоник Ci(t), приведен на рисунке:

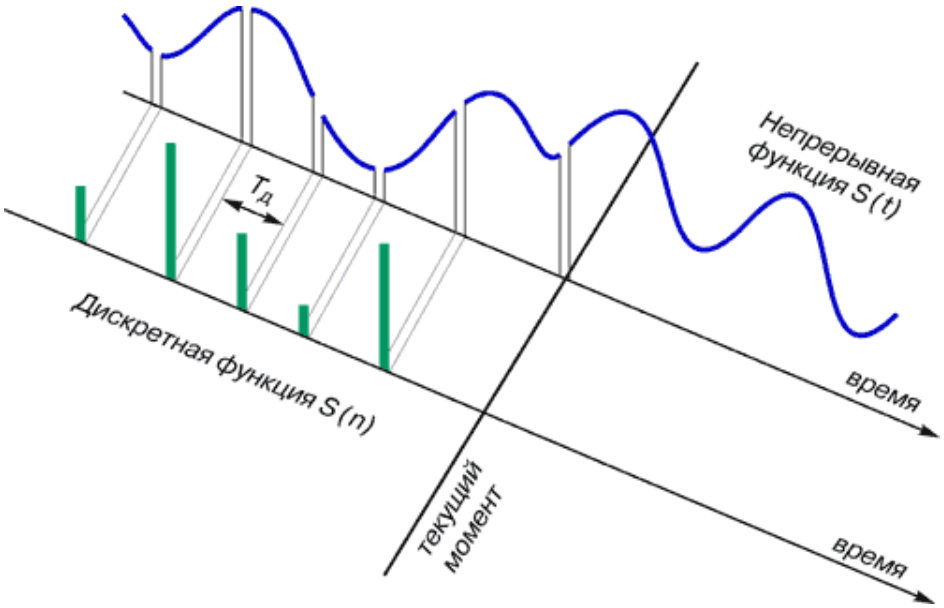


Представление прямоугольного импульса суммой гармонических составляющих (изображение с сайта http://n-t.ru)

Понятия «отобразить в частотной области некую функцию от времени» и «отобразить спектр этой функции» – равнозначны. Если скользнуть по рисунке взглядом по горизонтали слева направо, то произойдет переход от какой-либо функции времени к ее спектру. А нижняя часть рисунка есть иллюстрация одного из основных принципов ПФ – спектр суммарной функции времени равен сумме спектров ее гармонических составляющих.

Неоспоримым достоинством ПФ является его гибкость – преобразование может использоваться как для непрерывных функций времени, так и для дискретных. В последнем случае оно называется дискретным ПФ – ДПФ.

Для получения дискретной функции времени надо подвергнуть процессу дискретизации непрерывную функцию времени. Вырезаем отдельные значения из непрерывной функции, выстраивая дискретную функцию времени. Период одного цикла его работы называется «периодом дискретизации», или «интервалом дискретности».



Дискретное представление непрерывной функции

(изображение с сайта: http://n-t.ru)

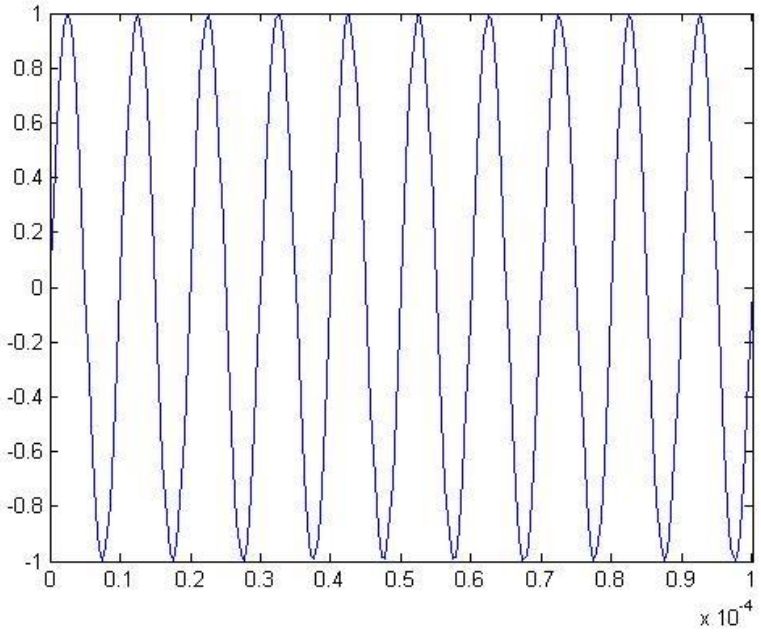
Одним из ключевых терминов в теории обработки сигналов является понятие о спектре сигнала. Прежде чем пояснить что это такое, введём термин гармонический сигнал. Гармонический сигнал – это сигнал, описываемый функцией синуса (или косинуса). Другими словами, гармонический сигнал может быть представлен следующей формулой

$$A(t) = A_0 \sin(2\pi\upsilon t + \phi), \quad (1)$$

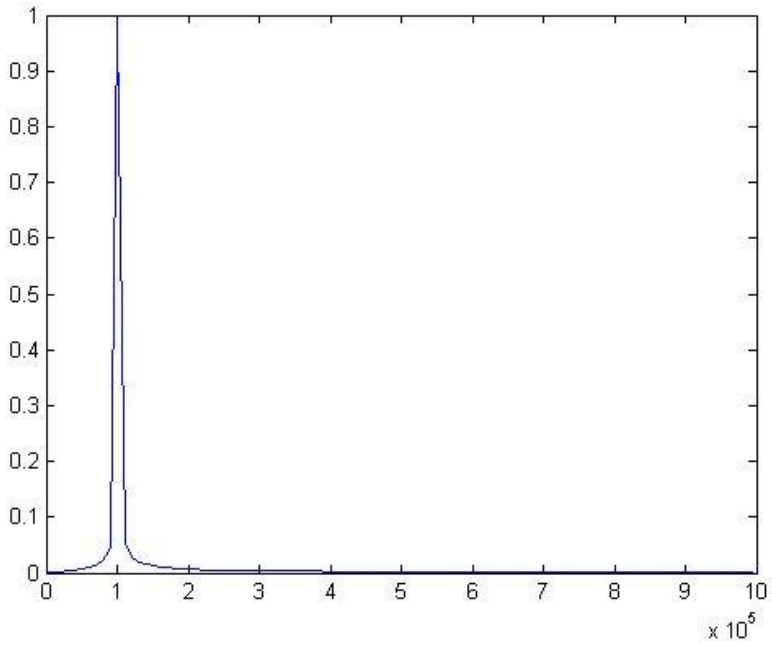
где A - амплитуда, $2\pi\upsilon = \omega$ - частота, ϕ - фаза

Важность понятия гармонического сигнала объясняется тем, что любой периодический сигнал (например, последовательность прямоугольных или треугольных импульсов) всегда может быть представлен конечной или бесконечной суммой гармонических сигналов – это есть теорема Фурье о разложение в ряды. Само собой разумеется, что каждый гармонический сигнал (гармоника), входящий в сумму имеет свою амплитуду и значение начальной фазы.

Гармонический сигнал конечной длительности – это простейший сигнал синусоидальной формы, который выдаёт генератор частоты, то есть сигнал, имеющий длительность много больше периода синусоиды.

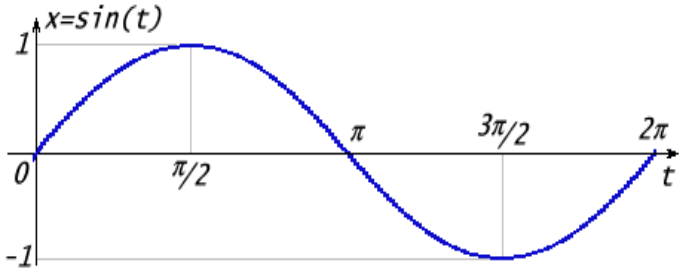


Исходный сигнал



Амплитудный спектр сигнала

Пусть у нас есть функция синуса x = sin(t):

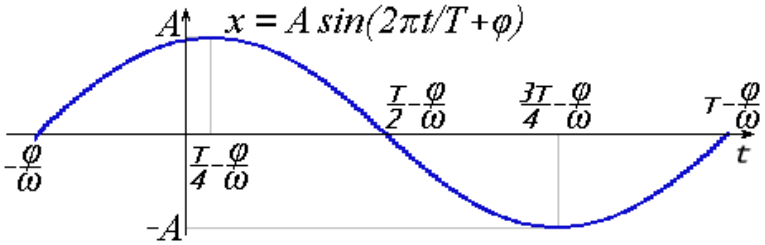


Максимальная амплитуда этого колебания равна 1. Если умножить его на некоторый коэффициент A, то получим тот же график, растянутый по вертикали в A раз: x = Asin(t).

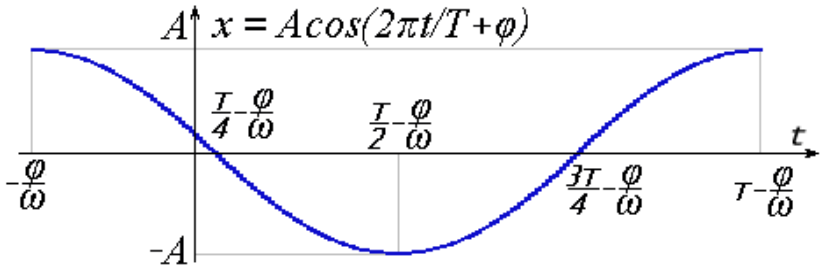
Период колебания равен 2π. Если мы хотим увеличить период до T, то надо умножить переменную t на коэффициент. Это вызовет растяжение графика по горизонтали: x = A sin(2πt/T).

Частота колебания обратна периоду: ν = 1/T. Также говорят о круговой частоте, которая вычисляется по формуле: ω= 2πν = 2πT. Откуда: x = A sin(ωt).

И, наконец, есть фаза, обозначаемая как φ. Она определяет сдвиг графика колебания влево. В результате сочетания всех этих параметров получается гармоническое колебание или просто - гармоника:



Схожим образом выглядит и выражение гармоники через косинус:



Достаточно изменить фазу на π/2, чтобы перейти от синуса к косинусу и обратно. Далее будем подразумевать под гармоникой функцию косинуса:

$$x = A\cos(2\pi t/T + \varphi) = A\cos(2\pi\nu t + \varphi) = A\cos(\omega t + \varphi) \qquad (2)$$

В природе и технике колебания, описываемые подобной функцией чрезвычайно распространены. Например, маятник, струна, водные и звуковые волны и прочее.

Преобразуем (2) по формуле косинуса суммы:

$$x = A\cos\varphi\cos(2\pi t/T) - A\sin\varphi\sin(2\pi t/T) \qquad (3)$$

Выделим в (3) элементы, независимые от t, и обозначим их как Re и Im:

$$x = Re\cos(2\pi t/T) - Im\sin(2\pi t/T) \qquad (4)$$

$$Re = A\cos\varphi,\ Im = A\sin\varphi$$

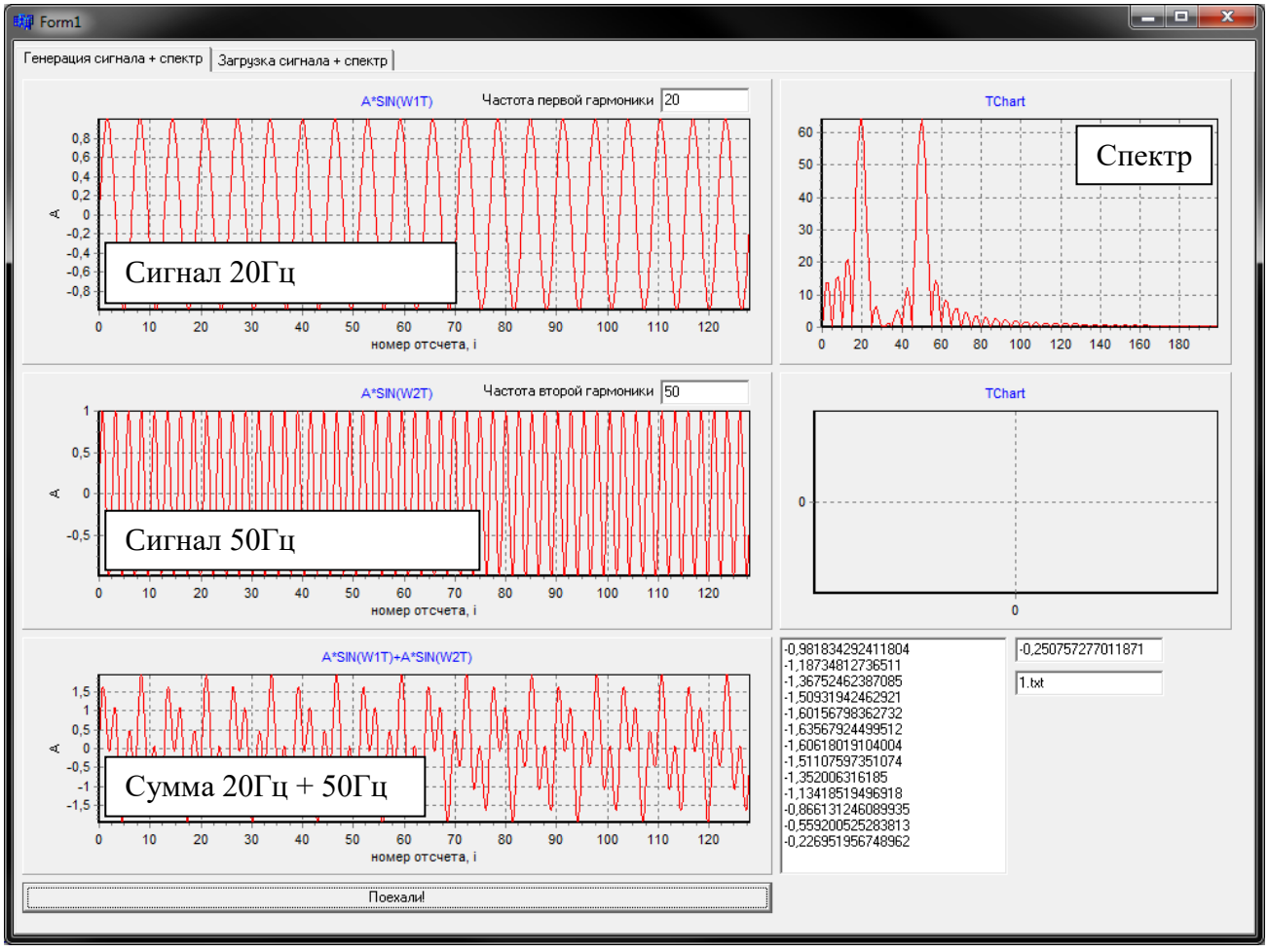
По величинам Re и Im можно однозначно восстановить амплитуду и фазу исходной гармоники:

$$\varphi = arctg\left(\frac{Im}{Re}\right)$$

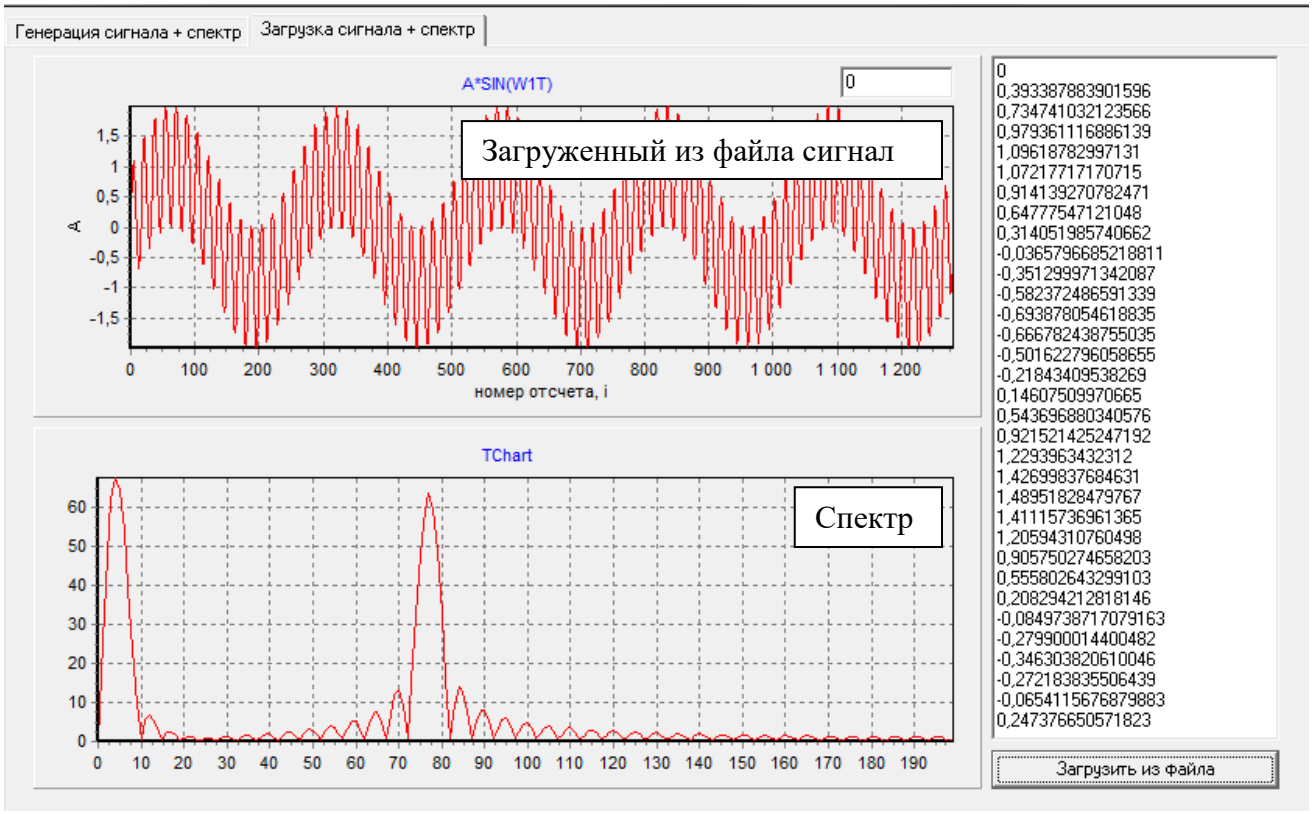
$$A = \sqrt{Re^2 + Im^2}$$

Задание

1. Изучить теоретическую часть.
2. Запрограммировать генерацию двух гармонических сигналов заданной частоты (любой) и их отображение на двух графиках.
3. Реализовать сигнал, равный сумме двух гармонических сигналов, сгенерированных по п.2.
4. Реализовать разложение итогового сигнала в амплитудный спектр.
5. Выполнить загрузку файла с сигналом (1280 точек), выданным преподавателем по вариантам, отобразить его на графике и реализовать его спектральное разложение:



Генерация двух сигналов, их сумма и амплитудный спектр

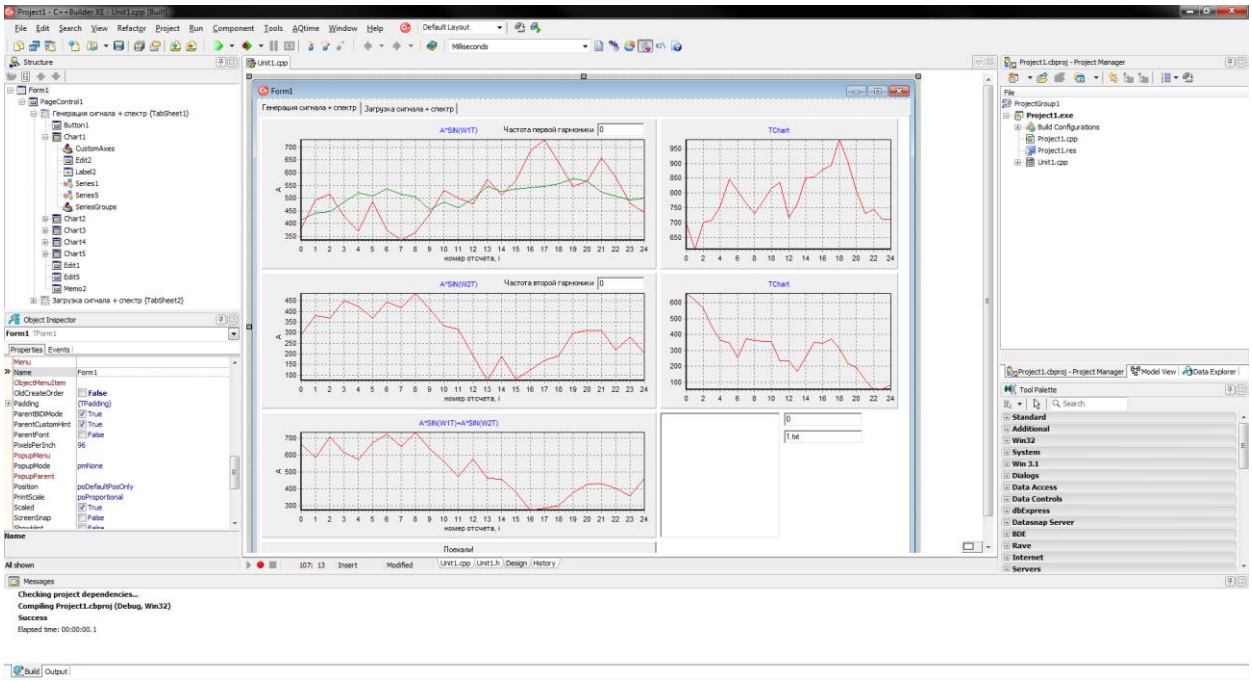


Загрузка файла с индивидуальным сигналом и его разложение в спектр (в данном случае обнаружены гармоники 5 Гц и 78 Гц)

Рекомендации по выполнению работы

1. Создаем новый проект в C++ Builder (в данном примере использовался C++ Builder XE).

2. Помещаем необходимые элементы на форму, как указано в примере:



3. В обработчике события нажатия кнопки необходимо прописать генерацию трех сигналов по формуле (1).

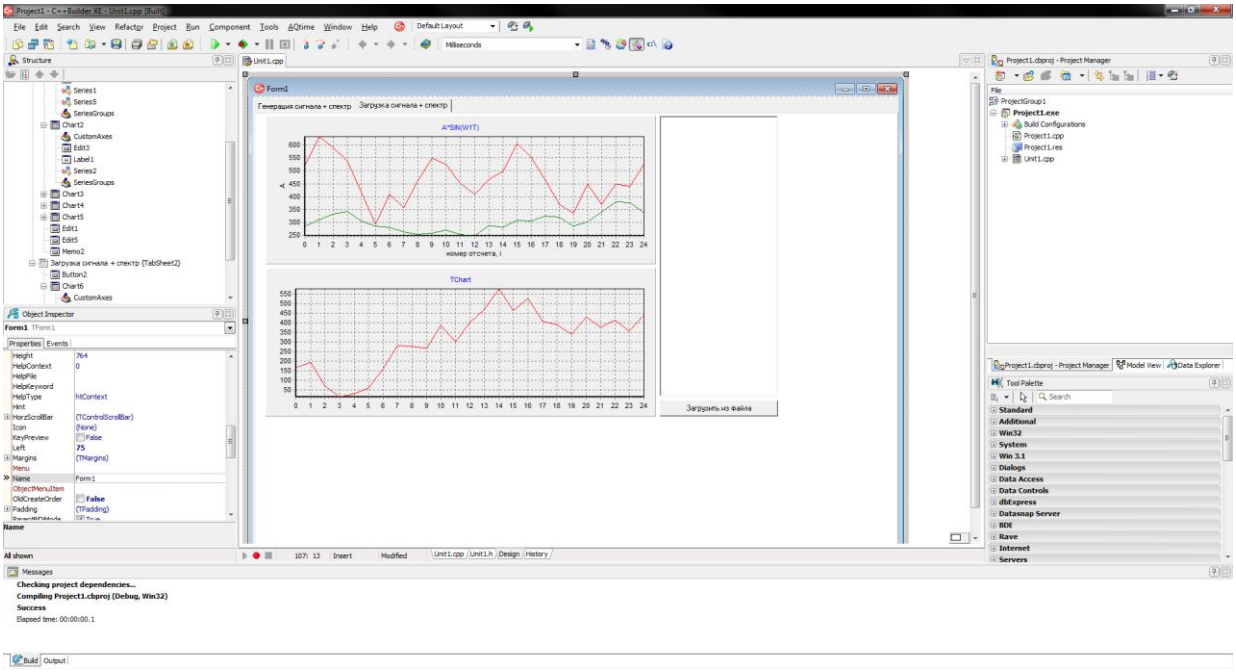
4. Для построения спектра, необходимо умножать каждую точку исходного сигнала на каждую точку синусоиды с частотой от 1 до 100 Гц и накапливать сумму этих произведений. Иными словами, построение спектра с помощью ДПФ – это умножение исходного сигнала на синусоиды частотами 1,2,3…100 Гц (или более, если необходимо) и фиксирование значения суммы произведений. В случае если исходный сигнал совпадет с искомой синусоидой, то сумма произведений окажется значительной и на графике появится ярко выраженный пик, что покажет наличие синусоиды искомой частоты. Построение спектра реализуется выражением:

sum = sum + sig[k] * sin(2 * 3.14 * j * i / 1280),

где j – перебираемая частота синусоиды (от 1 до 100), sig[k] – точка исходного сигнала, i/1280 – временной шаг (дискретизация), sum – накопитель.

Перебираем все значения i и производим умножение. Затем увеличиваем j на единицу и повторяем цикл нахождения суммы произведений пока все 100 синусоид не будут перемножены.

5. Создать вторую вкладку на форме для загрузки файла и построения спектра. Установить необходимые компоненты как указано на рисунке:



6. Поместить выданный файл .txt с сигналом в папку с проектом и загрузить сигнал в компонент Memo с помощью следующей строки кода:

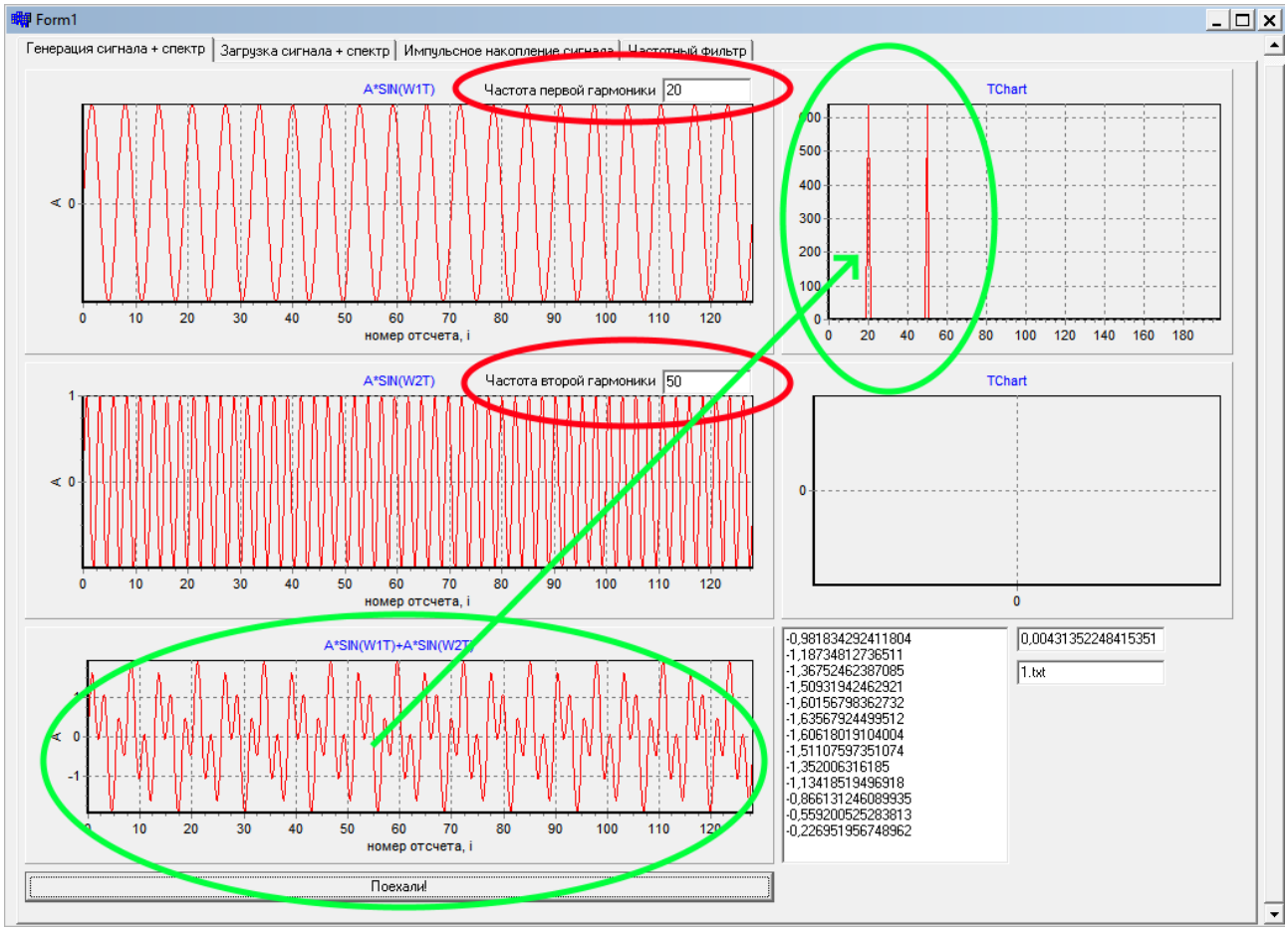
```
Memo2->Lines->LoadFromFile("myFile.txt");
```

7. Для обращения к данным из компоненты Memo2 и построения графика сигнала следует использовать код (в файле 1280 значений):

```
for (int i = 0; i < Memo2->Lines->Count; i++)
  LineSeries1->AddXY(i, StrToFloat(Memo2->Lines-
>Strings[i]));
```

8. Построить амплитудный спектр сигнала, обращаясь к элементам компоненты Memo2, как указано в п.4.

```
// Пример реализации ДПФ:
int j = 0, k = 0, f1 = 20, f2 = 50;
float Re = 0, Im = 0;
float signal[1281];
// Генерация сигнала с дискретизацией 1/1280 секунды:
for (float i = 0; i < 128; i = i + 0.1)
{
   firstSin[j] = sin(2*3.14*i/128*f1);
   secondSin[j] = sin(2*3.14*i/128*f2);
   signal[j] = firstSin[j] + secondSin[j];
   Form1->Series1->AddXY(i, firstSin[j]);
   Form1->Series2->AddXY(i, secondSin[j
   Form1->Series3->AddXY(i, signal[j]);
   j += 1;
}
// Дискретное преобразование Фурье
j = 0;
for (int j = 0; j < 200; j++)
{
   for (float i = 0; i < 128; i = i + 0.1)
   {
     Form1->Series6->AddXY(i, signal[k]*sin(2*3.14*j*i/128));
     Im = Im + signal[k]*sin(2*3.14*j*i/128);
     k += 1;
   }
   Form1->Series6->Clear();
   spectrum = sqrt(Re*Re + Im*Im);
   Form1->Series4->AddXY(j, spectrum);
   Re, Im = 0;
   k = 0;
}
```



Результат выполнения кода для сигнала 20 Гц + 50 Гц

Генератор гармонического сигнала с наложением шума и когерентное накопление

Цель работы

Разработка программного генератора гармонического сигнала заданной частоты с наложением помехи и восстановлением сигнала с помощью алгоритма когерентного накопления.

Краткие сведения из теории

Метод накопления с усреднением позволяет выделить сигнал из шума. Теоретически, возможно обнаружить сколь угодно слабый сигнал с помощью данного метода.

Введем обозначения:

$$\xi - помеха;$$

$$Sig - сигнал;$$

Пусть, необходимо выделить сигнал *Sig* на фоне помехи ξ. Помеха накладывается на импульс и в каждой точке она различна, в отличие от сигнала, поэтому можно записать:

$$Sig + \xi_1 + Sig + \xi_2 + ... + Sig + \xi_n \quad (1)$$

- сумма периодов следования импульсов с помехой, накладываемой на него. Запись (1) можно представить в виде:

$$S = \sum_{i=1}^{n} (Sig + \xi_i) = nSig + \sum_{i=1}^{n} (\xi_i), \quad (2)$$

где n – количество сигналов для суммирования

Отношение сигнал/шум можно записать в виде:

$$\frac{Sig}{\frac{1}{n}\sum_{i=1}^{n} (\xi_i)} \quad (3)$$

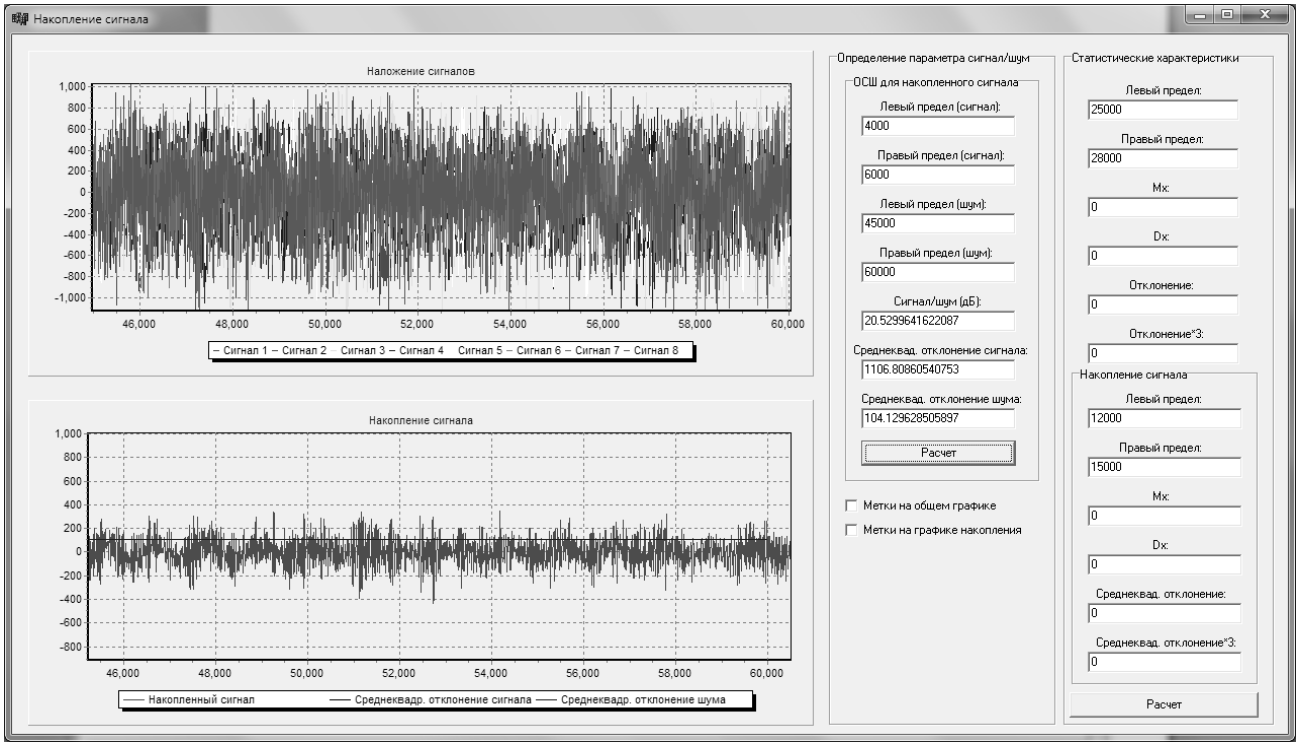
Знаменателем выражения (3) является среднее значение помехи, а так как оно колеблется около нуля (то в положительную область, то в отрицательную),

то с ростом n значение математического ожидания помехи будет стремиться к нулю и полезный сигнал начнет выделяться из шума.

Для реализации накопления сигнала необходимо знать, в какие моменты возникает периодичный сигнал (импульс), который требуется выделить из шума. Затем, временные промежутки, соответствующие следованию импульсов, суммируются и делятся на количество импульсов (определяется среднее арифметическое каждого из значений).

Отношение сигнал/шум в данном случае пропорционально значению $\sqrt{n}$, поскольку соотношение сигнал/шум можно представить в виде: $\frac{\sqrt{n}Sig}{\sigma}$, где σ - среднеквадратичное отклонение. Т.е. с ростом числа накопленных импульсов, соотношение сигнал/шум будет повышаться в $\sqrt{n}$ раз, где n – число накопленных импульсов.

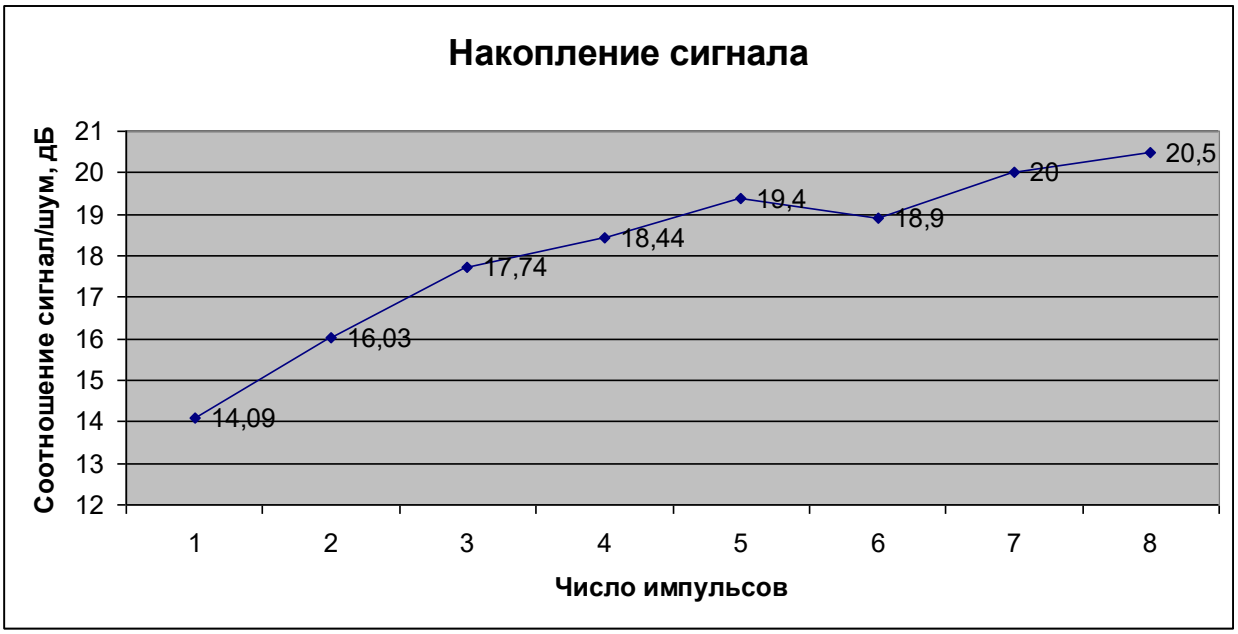
Сравнив график накопленных восьми импульсов и сводный график наложения этих импульсов, теория накопления работает. На рисунке изображен уровень шума с накоплением и уровень шума с наложением сигналов друг на друга (отображение максимальной амплитуды из восьми импульсов и шума).



Применив суммирование и усреднение восьми импульсов, нам удалось снизить максимальное значение амплитуды шума с 1000 до 300 единиц (значения амплитуды 16-битного файла находятся в интервале [-32767;32767]).

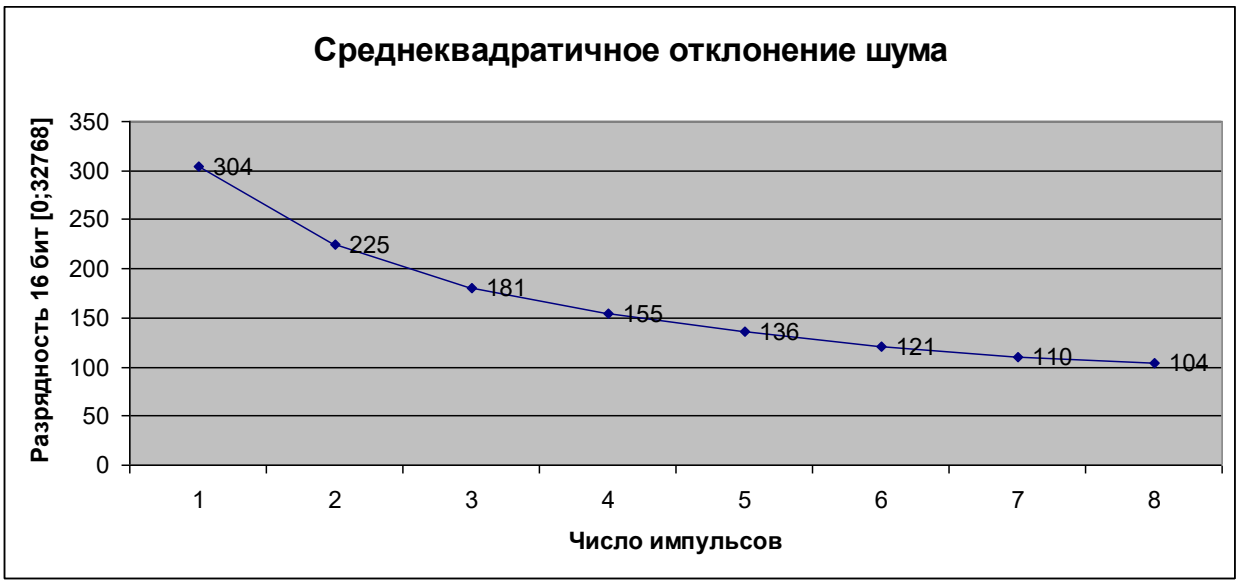
Таблица значений среднеквадратичной амплитуды сигнала, среднеквадратичного отклонения шума и отношения сигнал/шум

N	1	2	3	4	5	6	7	8
Ac	1541	1426	1398	1294	1268	1067	1100	1106
$\sigma_{ш}$	304	225	181	155	136	121	110	104
Сигнал/ шум, дБ	14,09	16,03	17,7	18,44	19,4	18,9	20	20,5



Применение метода накопления для зашумленных импульсов упругих колебаний трубопровода

Применение метода накопления сигнала позволило увеличить соотношение сигнал/шум с 14.09 дБ до 20.5 дБ при накоплении восьми импульсов. Значение параметра сигнал/шум в точке 6 меньше, чем в точке 5 ввиду смещения фазы и колебания амплитуд импульсов.



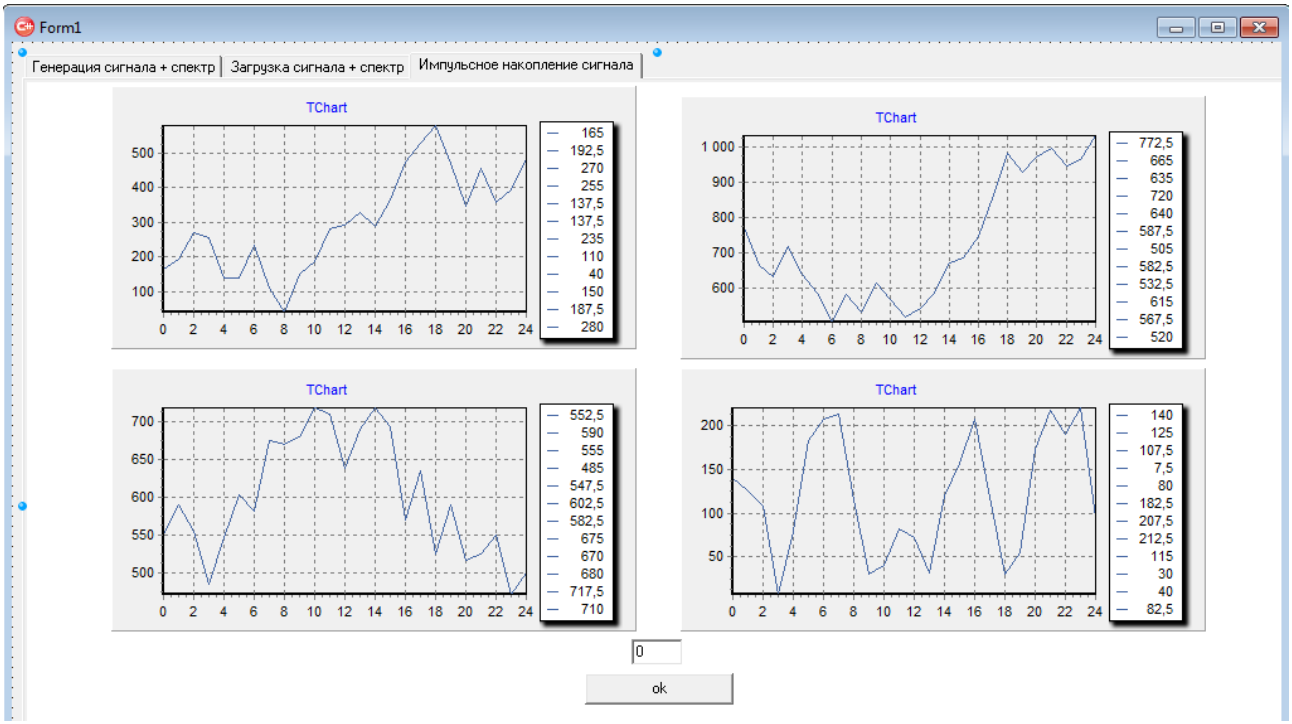
Снижение значения среднеквадратичного отклонения шума

Задание

1. Изучить теоретическую часть.
2. Запрограммировать генерацию гармонического сигнала.
3. Запрограммировать генерацию шума (использовать равномерный закон распределения).
4. Получить сигнал, равный сумме полученного гармонического сигнала и шума (имитация передачи сигнала по каналу связи).
5. Реализовать алгоритм накопления сигнала, описанный в теоретической части и графически показать, как происходит восстановление сигнала с повышением числа накопленных импульсов, варьируя их количество от 1 до 20 и отражая графики в отчете (можно сделать число импульсов следующим: 1, 3, 5, 10, 20 – 5 графиков).

Рекомендации по выполнению работы

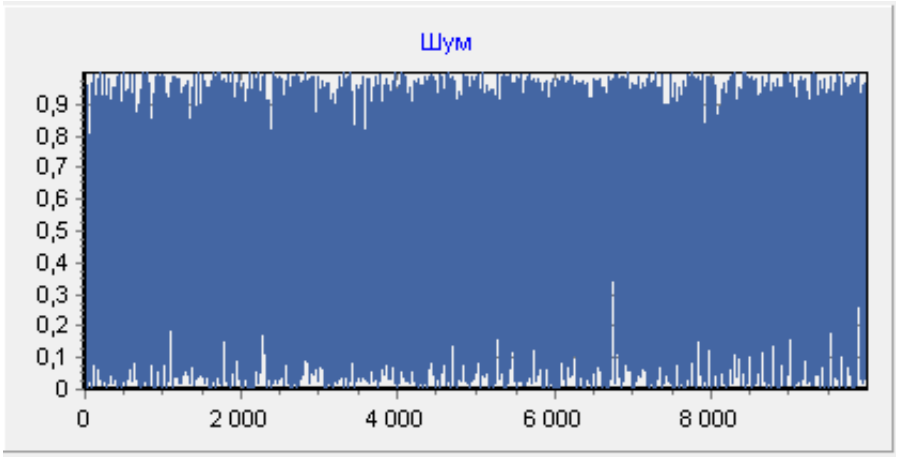
1. Создадим новый проект.
2. Помещаем необходимые элементы на форму, как указано в примере:

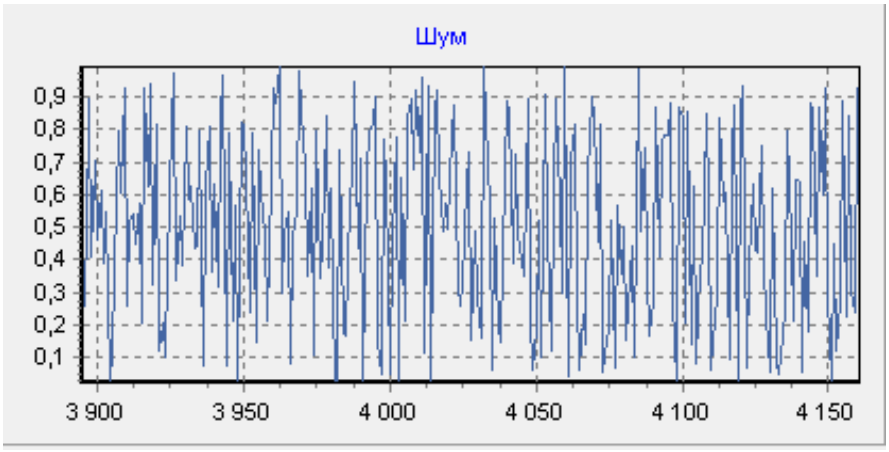


3. В обработчике события нажатия кнопки необходимо прописать генерацию синусоиды, помехи, синусоиды с помехой и результат накопления.

4. Следует использовать равномерный закон распределения для генерации чисел от 0 до 1 (возможен и другой диапазон). Пример кода для создания помехи встроенными средствами языка C++:

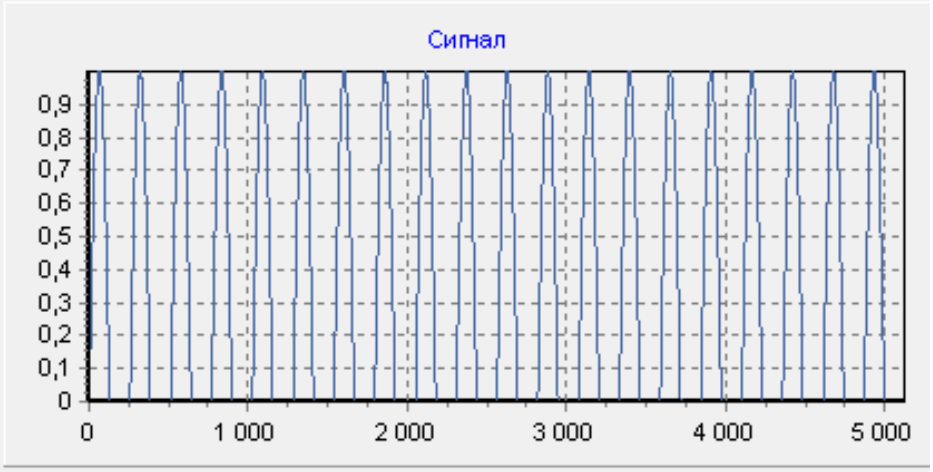
```
for (int i = 0; i < 10000; i++) // Генерируем 10000 чисел
{
   randomValue = (float)random(10000) / 10000 * md;
   randomValues[i] = randomValue;
   Series7->AddXY(i, randomValue); // График помехи
}
```



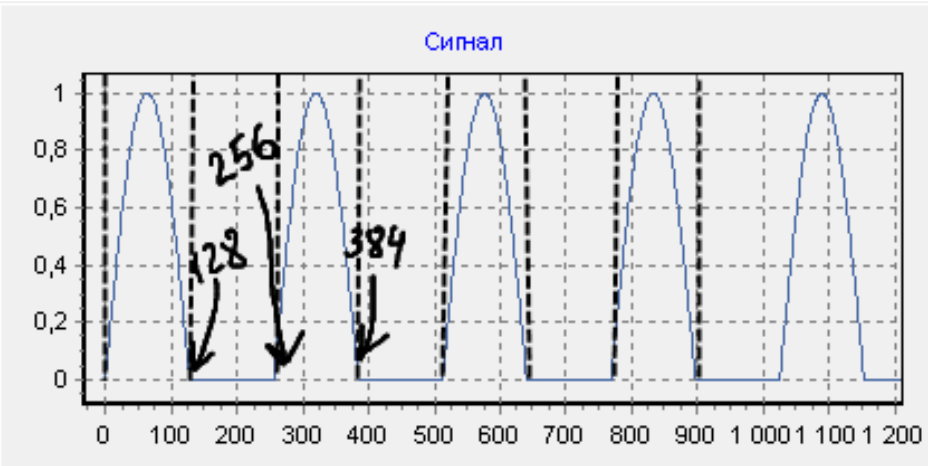


Для создания импульсного сигнала можно сгенерировать синусоиду выбранной вами частоты и обнулить отрицательную область. Мы получим точку начала импульса – 0, его завершение – 128. Точка начала второго импульса – 384 (со смещением 256), точка его завершения – 512 и т.д. до 20 импульсов (до 5120).

```
// Пример для сигнала 20 Гц
for (float i = 0; i < 512; i = i + 0.1) // 5120 отсчетов
{
   // Заполняем массив данными, md - модификатор амплитуды
   sig[j] = sin(2*3.14*i/512*20) * md;
   if (sig[j] < 0) // Обнуляем отрицательную область
      sig[j] = 0;
   Series8->AddXY(j, sig[j]); // Выводим сигнал
   j += 1;
}
```



Отобразим интервалы, в которых следует импульс (0-128, 256-384, 512-640 и т.д.):

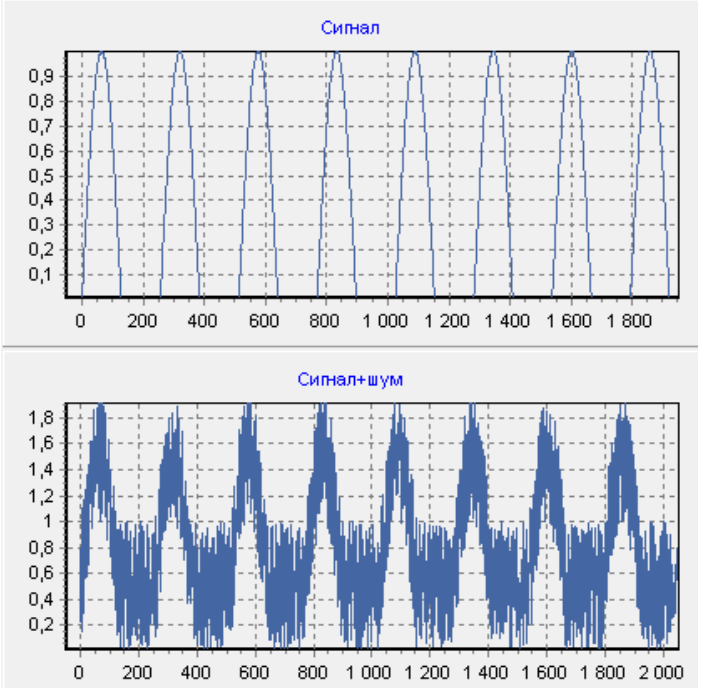


5. Наложение помехи на сигнал:

```
for (int i = 0; i < 5120; i++)
    sig[i] += randomValues[i];
```

6. Отображение сигнала с помехой:

```
for (float i = 0; i < 512; i = i + 0.1)
{
    LineSeries5->AddXY(j, sig[j]);
    j += 1;
}
```



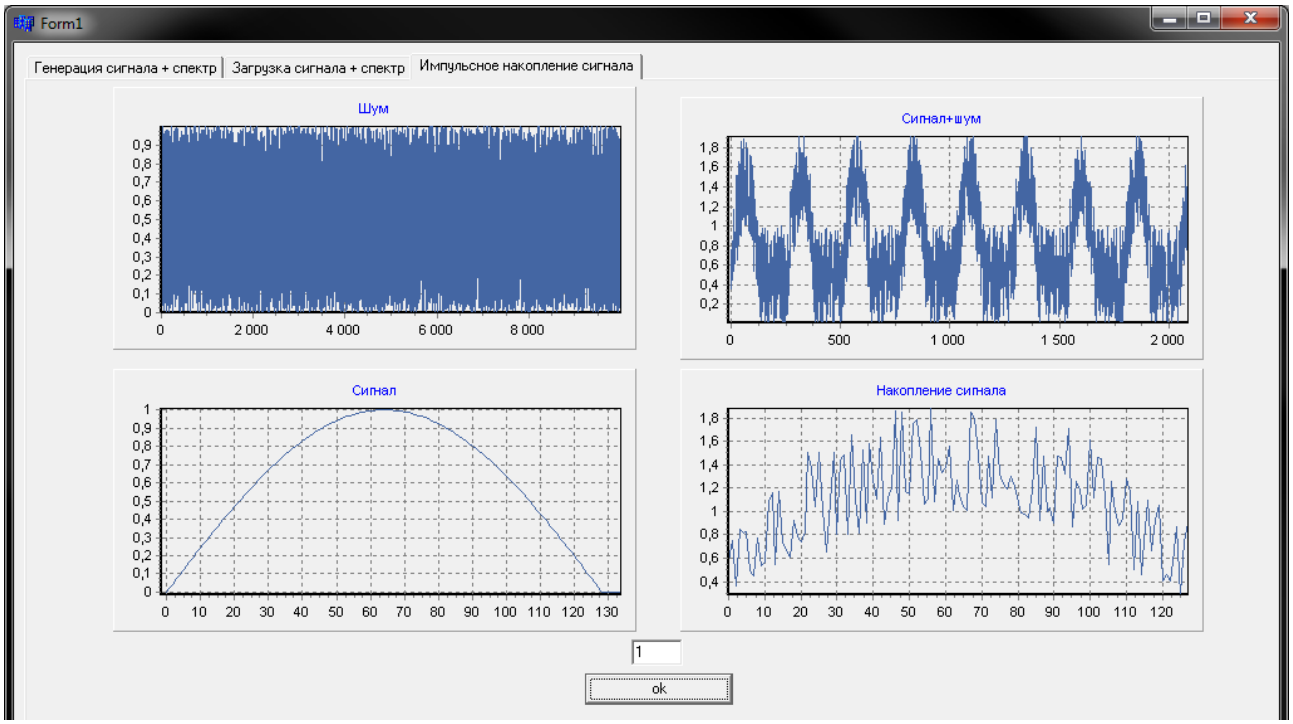
7. Накопление подразумевает суммирование каждого отсчета каждого импульса и деление суммы на количество импульсов. Отношение суммы к количеству импульсов можно заменить отношением каждого члена суммы к количеству импульсов $(a+b)/n=a/n+b/n$ и сразу разделить каждый отсчет на требуемое количество импульсов (столько, сколько будем накапливать, в нашем случае от 1 до 20):

```
for (int i = 0; i < 5120; i++)
   sig[i] = sig[i] / N; // N – число накапливаемых
импульсов
```

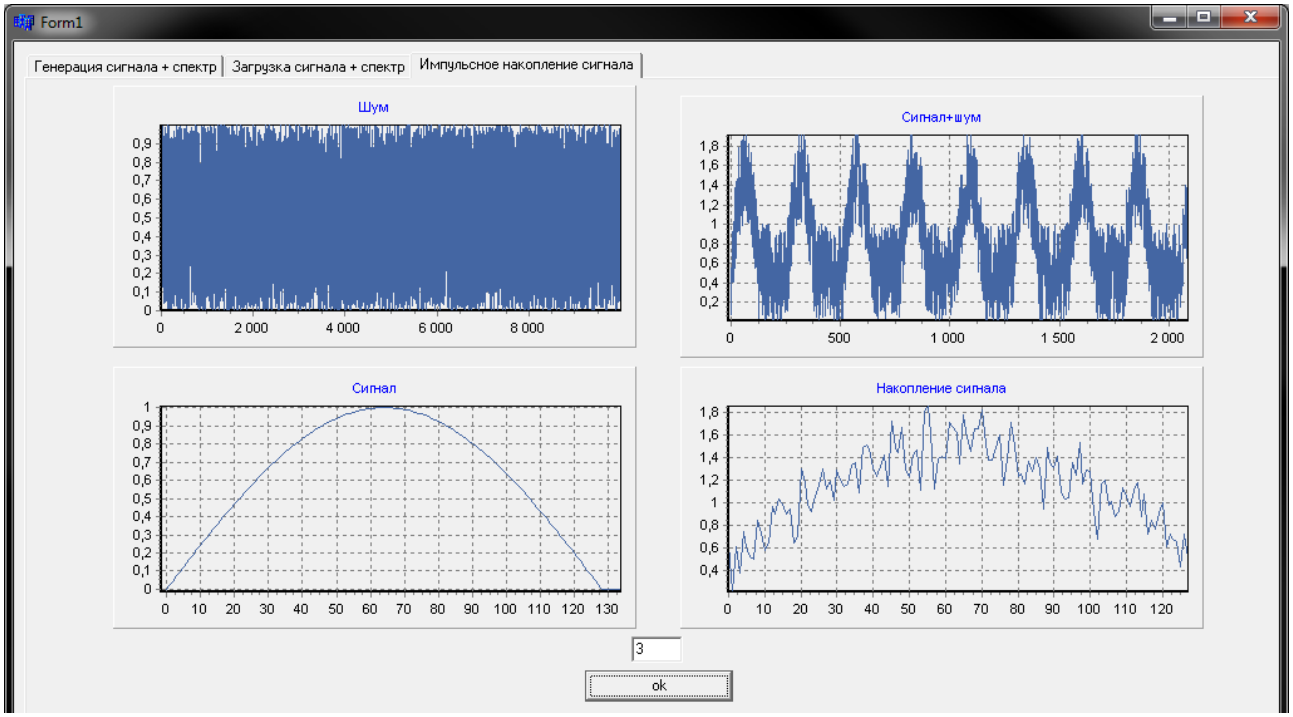
8. Суммируем i-й отсчет всех импульсов

```
// Длительность импульса - 128 отсчетов
for (int j = 0; j < 128; j++)
{
   // Проходим каждый i-отсчет в каждом N-импульсе
   for (int i = k; i < 256 * N; i+=256)
      sum = sum + sig[i]; // находим сумму
   LineSeries4->AddXY(j, sum);
   sum = 0;
   k++;
}
```

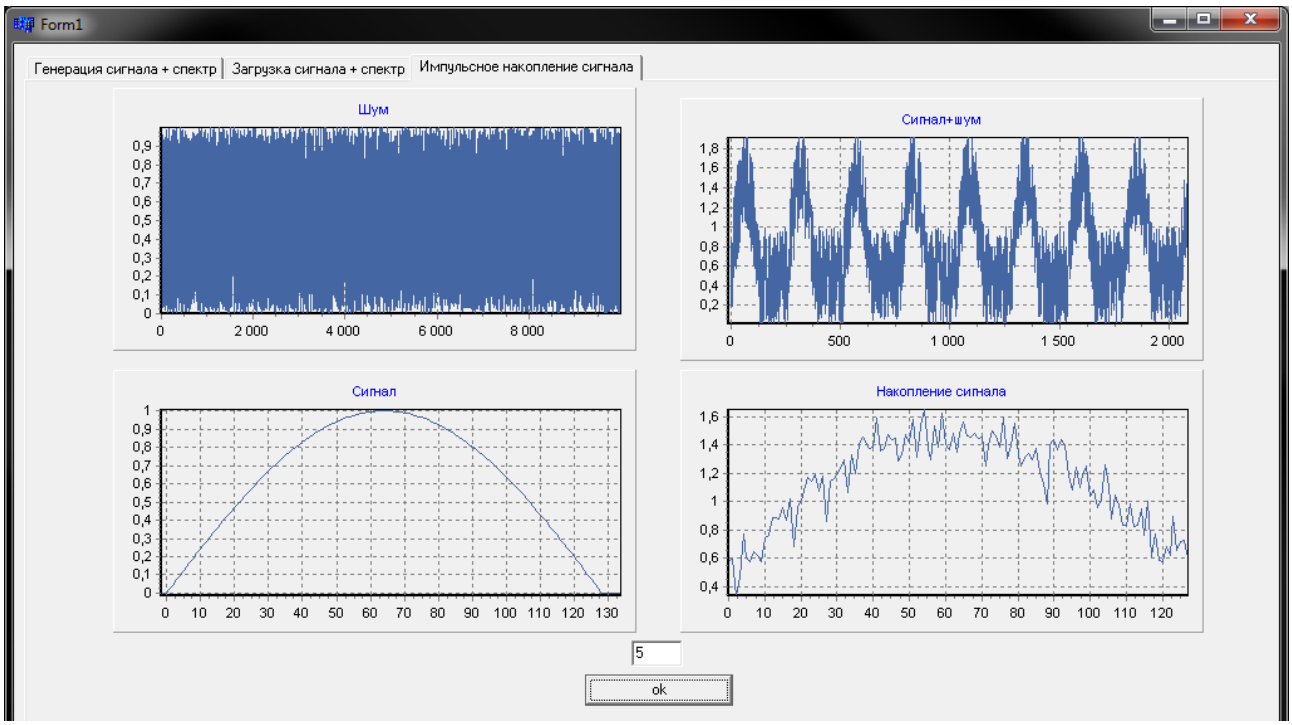
9. Результат работы программы должен выглядеть так, чтобы наблюдалось приближение накопленного импульса по форме к исходному с повышением числа накоплений:



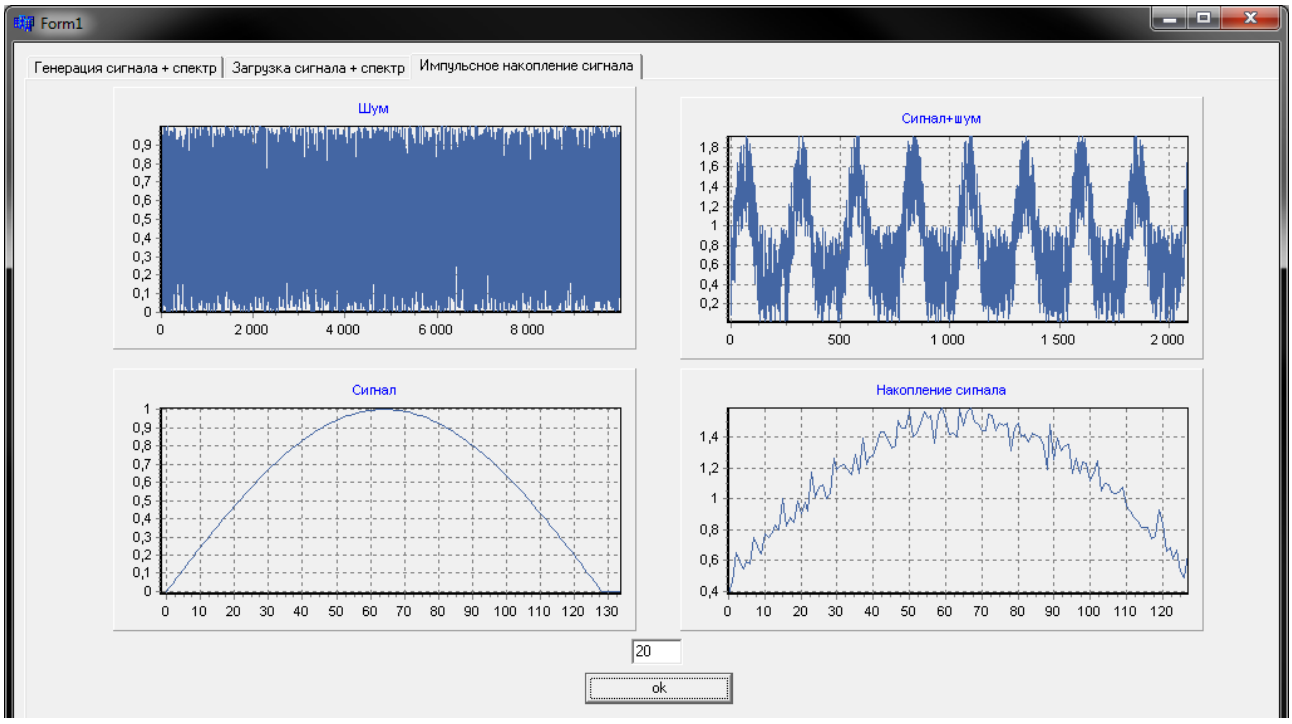
N = 1



N = 3


Form1
Генерация сигнала + спектр
Загрузка сигнала + спектр
Импульсное накопление сигнала
Шум
Сигнал+шум
Сигнал
Накопление сигнала
5
ok

N = 5


Form1
Генерация сигнала + спектр
Загрузка сигнала + спектр
Импульсное накопление сигнала
Шум
Сигнал+шум
Сигнал
Накопление сигнала
20
ok

N = 20

Генератор гармонического сигнала с наложением помехи, фильтрацией скользящим средним и оценкой эффективности фильтра корреляционным анализом

Цель работы

Разработка программного генератора гармонического сигнала заданной частоты с наложением помехи и восстановлением сигнала с помощью применения фильтра скользящего среднего с изменяемым размером окна.

Краткие сведения из теории

Скользящая средняя, скользящее среднее (англ. moving average, англ. MA) — общее название для семейства функций, значения которых в каждой точке определения равны среднему значению исходной функции за предыдущий период. Скользящие средние обычно используются с данными временных рядов для сглаживания краткосрочных колебаний и выделения основных тенденций или циклов. Математически скользящее среднее является одним из видов свёртки, и поэтому его можно рассматривать как фильтр нижних частот, используемых в обработке сигналов.

Скользящие средние используются:

- в статистике и экономике для сглаживания числовых рядов (в первую очередь временных). Например, для оценки ВВП, показателей занятости или других макроэкономических индикаторов;
- в технике, в качестве фильтра;
- в техническом анализе, в качестве самостоятельного технического индикатора либо в составе других инструментов.

Так как при расчёте скользящего среднего значение функции вычисляется каждый раз заново, при этом учитывается конечное значимое множество предыдущих значений и скользящее среднее «перемещается», как бы «скользя» по временному ряду.

Простое скользящее среднее, или арифметическое скользящее среднее (англ. simple moving average, англ. SMA) численно равно среднему арифметическому значений исходной функции за установленный период и вычисляется по формуле:

$$SMA_t = \frac{1}{n}\sum_{i=0}^{n-1} p_{t-i} = \frac{p_t + p_{t-1} + \cdots + p_{t-i} + \cdots + p_{t-n} + p_{t-n+1}}{n}, \quad (1)$$

Где SMA_t — значение простого скользящего среднего в точке t, SMA_{t-1} — предыдущее значение простого скользящего среднего, p_{t-n} — значение исходной функции в точке (в случае временного ряда, самое «раннее» значение исходной функции, используемое для вычисления предыдущей скользящей средней), p_t — значение исследуемой функции точке t (в случае временного ряда, текущее — последнее значение).

Например, простое скользящее среднее для временного ряда с десятью периодами вычисляется как:

$$SMA_t = \frac{p_t + p_{t-1} + p_{t-2} + p_{t-3} + p_{t-4} + p_{t-5} + p_{t-6} + p_{t-7} + p_{t-8} + p_{t-9}}{10} = SMA_{t-1} - \frac{p_{t-10}}{10} + \frac{p_t}{10}, \quad (2)$$

В данной работе реализуется применение простого скользящего среднего к двум сгенерированным сигналам одинаковой частоты (аналогичным образом, как в лаб. №5).

Для оценки эффективности алгоритма фильтрации предлагается использовать коэффициент корреляции Пирсона, определяющийся по формуле:

$$\mathbf{r}_{XY} = \frac{\mathbf{cov}_{XY}}{\sigma_X \sigma_Y} = \frac{\sum(X - \bar{X})(Y - \bar{Y})}{\sqrt{\sum(X - \bar{X})^2 \sum(Y - \bar{Y})^2}}, \quad (3)$$

где X – текущее значение функции первого сигнала, $\bar{X}$ - среднее значение функции первого сигнала, Y – текущее значение функции первого сигнала, $\bar{Y}$ - среднее значение функции второго сигнала

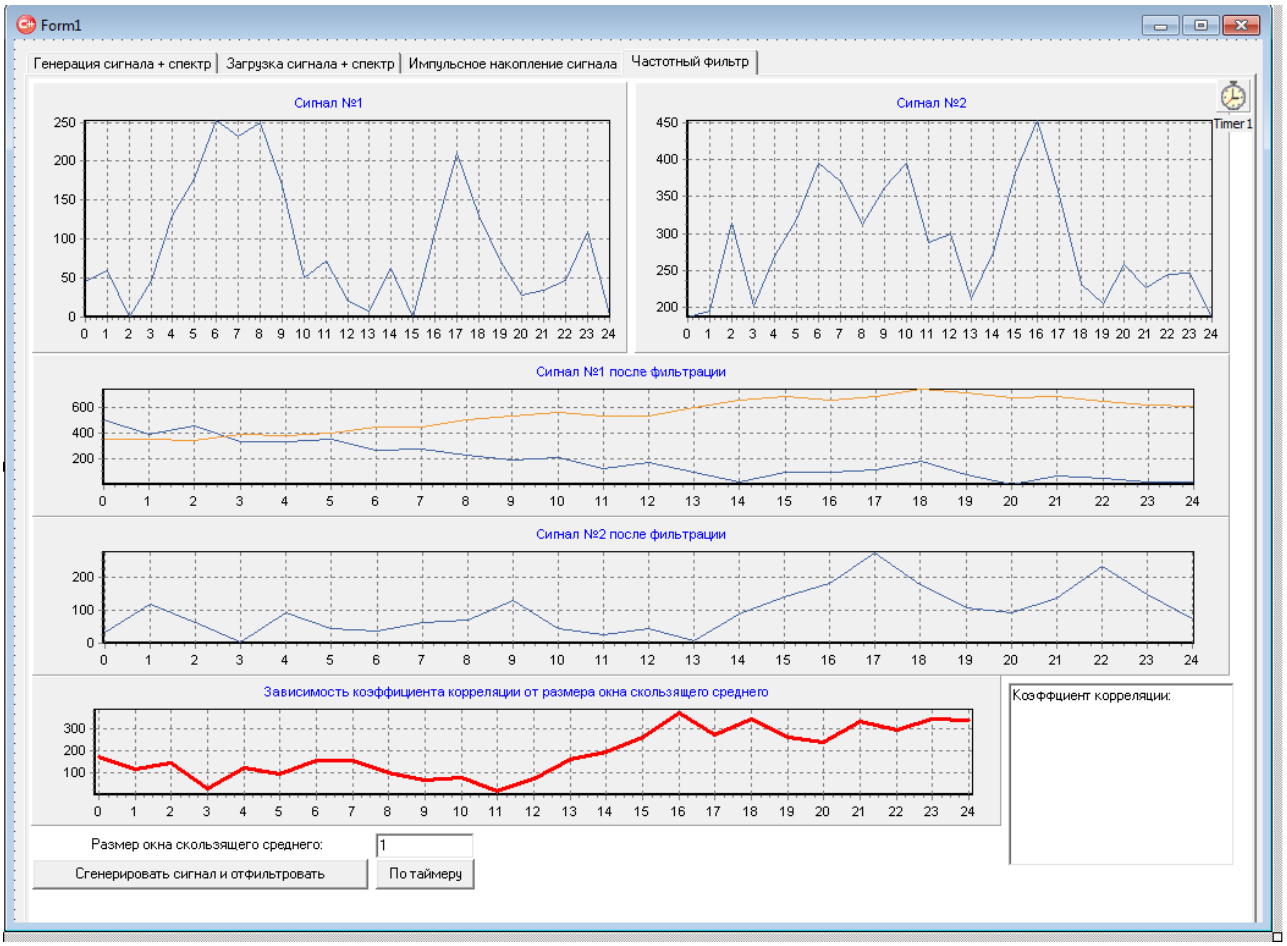
Задание

1. Изучить теоретическую часть.
2. Запрограммировать генерацию гармонического сигнала.
3. Запрограммировать генерацию шума (использовать равномерный закон распределения).
4. Сгенерировать второй гармонический сигнал той же частоты.
5. Наложить шум, сгенерированный по тому же закону распределения, на второй сигнал.
6. Применить скользящее среднее с окном в 1-100 точек (с шагом в одно значение) для двух сгенерированных сигналов с шумом.
7. При каждом фиксированном значении окна скользящего среднего провести оценку сходства двух сигналов определением коэффициента корреляции.
8. Построить графики: 2 исходных сигнала с шумом, 2 отфильтрованных сигнала, график зависимости коэффициента корреляции от размера окна скользящего среднего (100 точек).
9. Убедиться в повышении значения коэффициента корреляции с увеличением размера окна фильтра.

Рекомендации по выполнению работы

1. Добавляем новую вкладку в программе, написанной в рамках лабораторной работы №5.

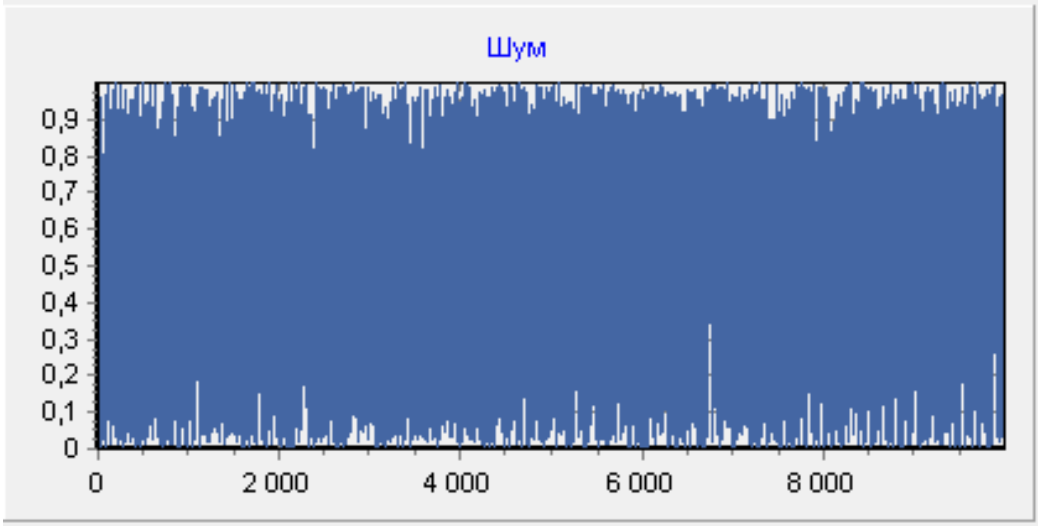
2. Помещаем необходимые элементы на форму, как указано в примере:

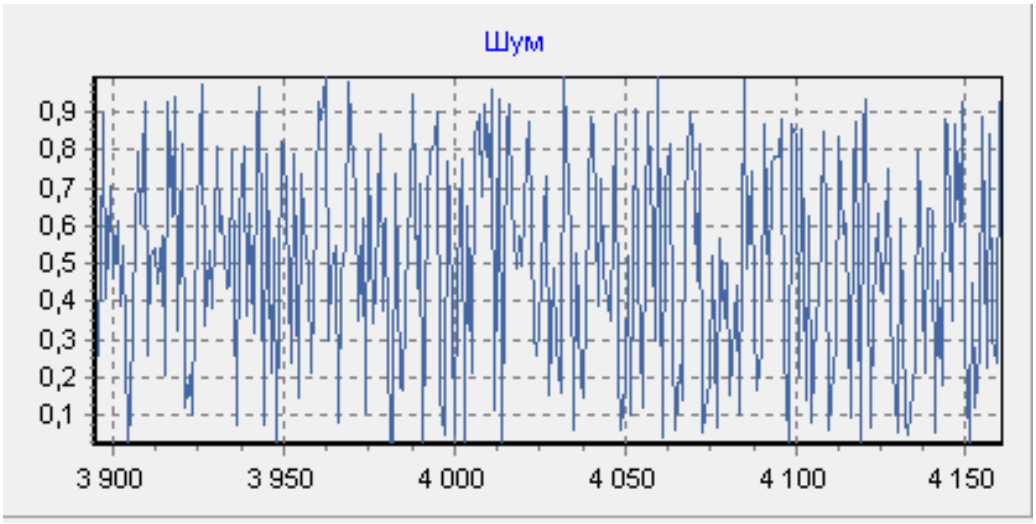


3. В обработчике события нажатия кнопки необходимо прописать генерацию сигнала, помехи, сигнала с помехой и результат накопления.

4. Для наложения помехи следует использовать равномерный закон распределения для генерации чисел от 0 до 1 (возможен и другой диапазон). Пример кода для создания помехи встроенными средствами языка C++:

```
for (int i = 0; i < 10000; i++) // Генерируем 10000 чисел
{
   randomValue = (float)random(10000) / 10000 * md;
   randomValues[i] = randomValue; // Заполняем массив
  данных случайными величинами
   Series->AddXY(i, randomValue); // Выводим помеху на
график
 }
```





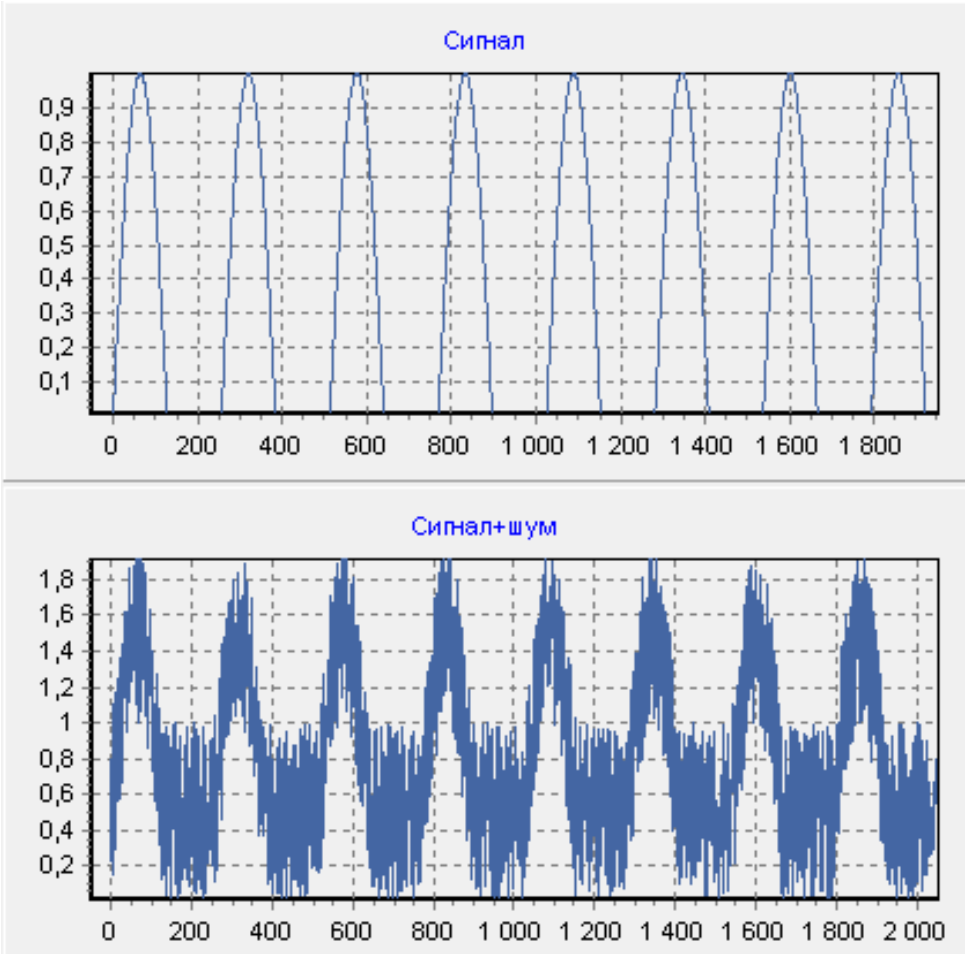
5. Для создания импульсного сигнала можно сгенерировать синусоиду выбранной вами частоты и обнулить отрицательную область. Мы получим точку начала импульса – 0, его завершение – 128. Точка начала второго импульса – 384 (со смещением 256), точка его завершения – 512 и т.д. до 20 импульсов (до 5120).

6. Повторяем пункты 3-6, чтобы сгенерировать второй сигнал, идентичный первому с наложением новой помехи.

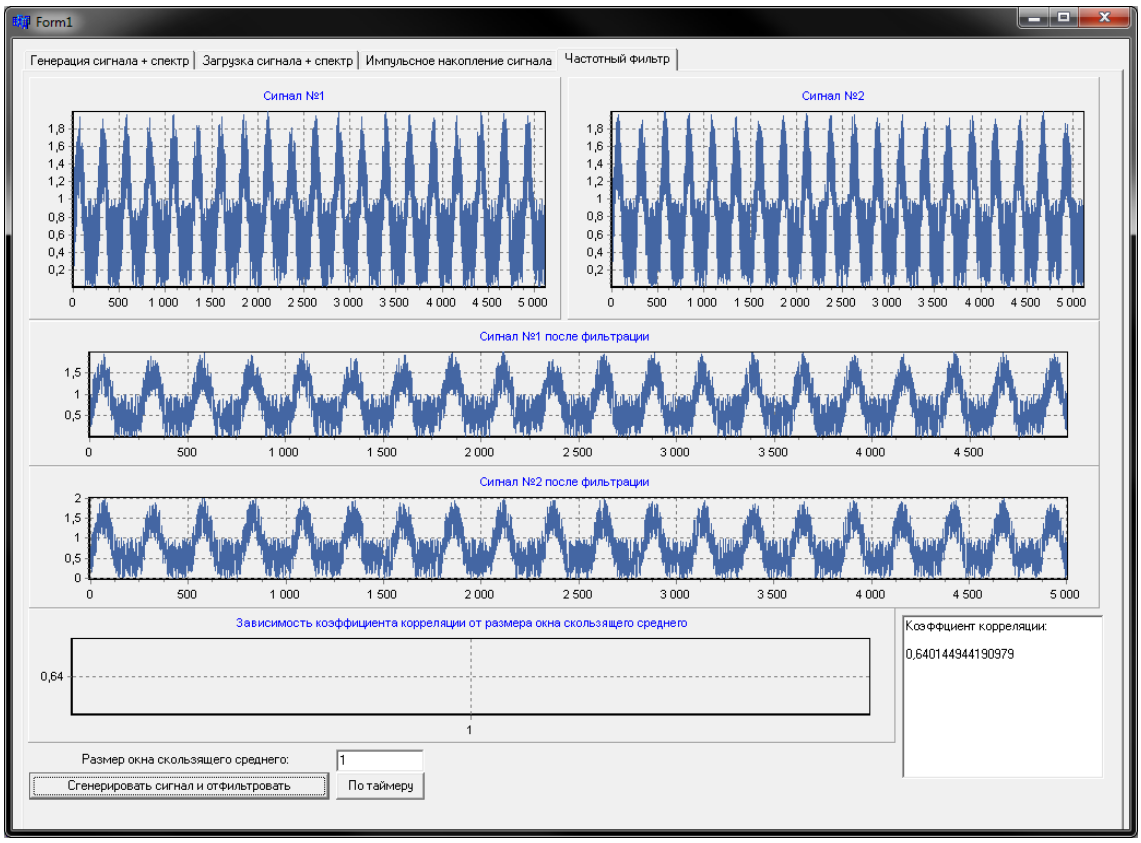
7. Находим коэффициент корреляции по формуле (3), строим график его зависимости от текущего значения окна фильтра и отображаем коэффициент.

8. Увеличиваем размер окна скользящего среднего (avgSize) на единицу и повторяем пункты 8-10 для нахождения коэффициента корреляции для увеличенного на единицу размера окна.

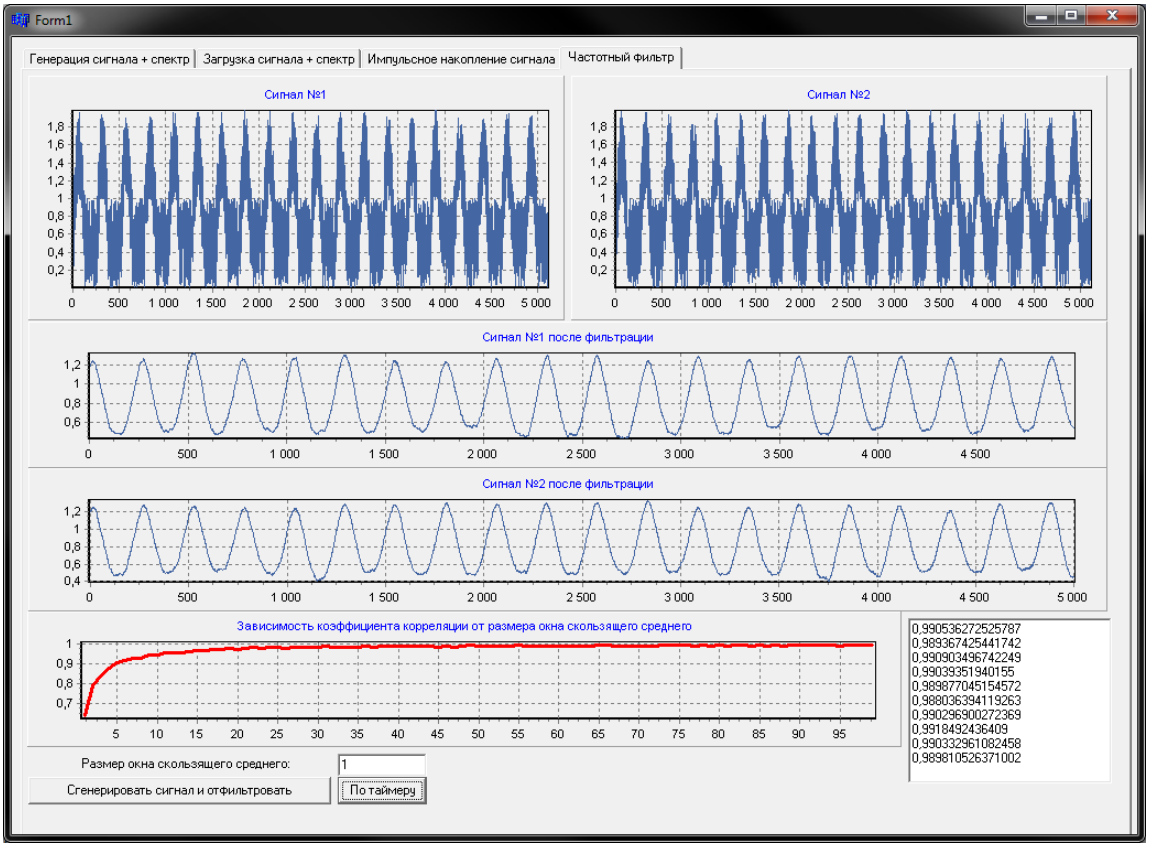
9. Фильтруем, находим коэффициент корреляции вплоть до размера окна фильтра, равного 100.



```
// Пример реализации для сигнала 20 Гц
for (float i = 0; i < 512; i = i + 0.1) // Генерация 5120
отсчетов
{
   // Заполняем массив данными
   sig[j] = sin(2*3.14*i/512*20) * md;
   if (sig[j] < 0) // Обнуляем отрицательную область
     sig[j] = 0;
   Series8->AddXY(j, sig[j]); // Выводим сигнал
   j += 1;
}

// Наложение помехи
for (int i = 0; i < 5120; i++)
  sig[i] += randomValues[i];

// Отображение сигнала с помехой
for (float i = 0; i < 512; i = i + 0.1)
{
   LineSeries5->AddXY(j, sig[j]);
   j += 1;
}

for (int i = 0; i < 5000; i++) // Проходим весь сигнал
{
   for (int j = k; j < k + avgSize; j++)
      avgSum += sig[j]; // Находим сумму значений сигнала
   k = k + 1;
   avgResult = avgSum / avgSize;
   avgArray[i] = avgResult;
   Series->AddXY(i, avgResult); // Отображаем результат
   avgSum = 0;
}
```



Исходные сигналы без фильтрации (коэффициент корреляции 0,64)



Сигналы после применения фильтра с окном 1-100 точек (коэффициент корреляции - 0,99)