Содержание

[1 Постановка задачи. 3](#_Toc405191055)

[2 Анализ методов решения задачи. 5](#_Toc405191056)

[2.1 Оптимальный дискретный фильтр Винера 5](#_Toc405191057)

[2.2 Метод наименьших квадратов (LMS) 6](#_Toc405191058)

[2.3 Рекурсивный метод наименьших квадратов (RLS) 8](#_Toc405191059)

[3 Описание выбранных методов. 10](#_Toc405191060)

[3.1 Дискретный прогнозирующий фильтр Винера 10](#_Toc405191061)

[3.2 Метод наименьших квадратов (LMS) 12](#_Toc405191062)

[4 Структурно-функциональная схема. 14](#_Toc405191063)

[5 Описание Разработанной системы. 15](#_Toc405191064)

[6 Результаты моделирования. 19](#_Toc405191065)

[Генерация процессов. 19](#_Toc405191066)

[Прогнозирование процессов 20](#_Toc405191067)

[Приложение 47](#_Toc405191068)

[Программный код приложения генерации 47](#_Toc405191069)

[Программный код приложения сервера 51](#_Toc405191070)

[Программный код приложения прогнозатора 53](#_Toc405191071)

# 1 Постановка задачи.

В рамках данной курсовой работы требуется разработать программную систему прогнозирования процессов, которая должна содержать: приложение, формирующее значения многомерного случайного процесса и передающие сформированные значения в сервер базы данных системы.

Сервер базы данных, регистрирующий отсчеты формируемого многомерного случайного процесса.

Приложение, реализующее прогнозирующий фильтр Винера и адаптивный фильтр LMS, которые формируют на основе получаемых с сервера зарегистрированных отсчетов многомерного случайного процесса прогнозные значения компонентов многомерного случайного процесса. Заданы следующие исходные требования:

Отсчеты случайного процесса S(j,k) (j- номер компоненты процесса , k – номер отсчета) формируются путем прохождения некоррелированных случайных чисел в X(k) диапазоне [ -1, +1 ] через цифровые фильтры нижних частот с задаваемыми параметрами.

Частота среза фильтров нижних частот Fs = 0,5 Гц.

Частота дискретизации Fd = 50 Гц.

Ts = 1/ 2π Fs , Td = 1/ Fd – период дискретизации.

Рекурсивный цифровой фильтр второго порядка задается выражением:

*Y(j,k) = A(j)\*X(k) + B(j)\*Y(j,k-1), j = [0,3], Y(j,0) = 0, S(j,0) = 0 –начальные значения,*

*S(j,k) = Km(j,k)\*[A(j)\*Y(j,k) + B(j)\*S(j,k-1)], k =1,2,3,………*

Коэффициенты цифрового фильтра определяются как

*A(j) = Td / (Td + Ts), B(j) = Ts /(Td + Ts), j = [0,3],*

*Km(0,k) = 1 – const,*

*Km(1,k) = 1 + Λ1 \* k/TM,*

*Km(2,k) = 1 + Λ2\*(k/TM)^2,*

*Km(3,k) = Λ3\*EXP(k/TM),*

Где TM – время моделирования (вводится, значение до Е7),

*Λ1, Λ2, Λ3* – параметры нестационарности формируемого процесса (вещественные числа) – вводятся.

Приложение вычисляет корреляционную функцию компонентов сформированного многомерного случайного процесса, их математические ожидания и дисперсии (среднеквадратичные значения). Реализации компонентов процесса и их корреляционных функций выводятся в соответствующие окна. Интервалы вывода должны задаваться.

Отсчеты компонентов сформированного многомерного случайного процесса записываются в таблицу базы данных.

Каждое значение сформированного многомерного случайного процесса записываются в таблицу базы данных. Запись содержит *S(0,k), S(1,k), S(2,k). S(3,k)* и значение соответствующего момента времени Tk. Затем записи передаются прогнозирующим фильтрам.

Прогнозуриющие фильтры Винера и LMS получают записи из базы данных и формируют прогнозные значения компонентов процесса *S(0), S(1), S(2). S(3)* на время прогнозирования *Tp = Kp\*Td (Kp = 1, 2, 3, 4,5).*

Вычисляются среднеквадратичные ошибки прогнозирования компонентов процесса фильтрами Винера и LMS за время моделирования Tm на один, два, три, четыре и пять шагов вперед *(Tp = 1Td, 2Td, 3Td, 4Td, 5Td).*

Результаты прогнозирования фильтрами Винера и LMS выводятся в соответствующие окна.

Сравнивается качество прогнозирования фильтрами Винера и LMS на всех компонентах процесса *S(0), S(1), S(2). S(3).*

# 2 Анализ методов решения задачи.

Рассмотрим три адаптивных алгоритма настройки адаптивных фильтров с использованием образцового сигнала, часто применяемых на практике в различных системах обработки информации. Для упрощения математических выкладок предположим, что сигналы и фильтры являются вещественными.

## 2.1 Оптимальный дискретный фильтр Винера

Прежде чем рассматривать собственно алгоритмы адаптации, необходимо определить те оптимальные параметры фильтра, к которым эти алгоритмы должны стремиться. Подход к задаче оптимальной фильтрации может быть как статистическим, так и детерминированным. Сначала рассмотрим статистический вариант.

Пусть входной дискретный случайный сигнал x(k) обрабатывается нерекурсивным дискретным фильтром порядка N, коэффициенты которого могут быть представлены вектор-столбцом .



Выходной сигнал фильтра равен

(1)

где *U(k)=[x(k), x(k-1),…, x(k-N)]T* вектор-столбец содержимого линии задержки фильтра на *k-*м шаге.

Кроме того, имеется образцовый (также случайный) сигнал (S(k)). Ошибка воспроизведения образцового сигнала равна

(2)

Необходимо найти такие коэффициенты фильтра **w,** которые обеспечивают максимальную близость выходного сигнала фильтра к образцовому, то есть минимизируют ошибку e(k).Поскольку e(k)также является случайным процессом, в качестве меры ее величины разумно принять средний квадрат. Таким образом, оптимизируемый функционал выглядит так:



Квадрат ошибки равен



Статистически усредняя это выражение, получаем следующее:

 (3)

Входящие в полученную формулу усредненные величины имеют следующий смысл:

— средний квадрат образцового сигнала;

— транспонированный вектор-столбец **p** взаимных корреляций между *k-*м отсчетом образцового сигнала и содержимым линии задержки фильтра. Если рассматриваемые случайные процессы x(t) и d(k) являются совместно стационарными, вектор взаимных корреляций не зависит от номера шага *k;*

- корреляционная матрица сигнала, имеющая размер *(N+1)\*(N+1)*. Для стационарного случайного процесса корреляционная матрица имеет вид матрицы Теплица, то есть на ее диагоналях стоят одинаковые величины:



здесь — корреляционная функция (КФ) случайного процесса 

С учетом введенных обозначений принимает следующий вид:

 (4)

Данное выражение представляет собой квадратичную форму относительно **w** и по тому при невырожденной матрице **R** имеет единственный минимум, для нахождения которого необходимо приравнять нулю вектор градиента:



Отсюда получаем искомое решение для оптимальных коэффициентов фильтра:

 (5)

Такой фильтр называется фильтром Винера. Подстановка (5) в (4) дает минимально достижимую дисперсию сигнала ошибки.

## 2.2 Метод наименьших квадратов (LMS)

Один из наиболее распространенных адаптивных алгоритмов основан на поиске минимума целевой функции методом наискорейшего спуска. При использовании данного способа оптимизации вектор коэффициентов фильтра w(k) должен рекурсивно обновляться следующим образом:

(6)

где  - положительный коэффициент, называемый размером шага. Показано, что алгоритм сходиться если , где - максимальное собственное число корреляционной матрицы R. Скорость сходимости при этом зависит от разброса собственных чисел корреляционной матрицы R – чем меньше отношение , тем скорее сходится итерационный процесс.

Однако для расчета градиента необходимо знать значение матрицы R и вектора P. На практике могут быть доступны лишь оценки этих значений, получаемых по входным данным. Простейшими такими оценками являются мгновенныезначения корреляционной матрицы и вектора взаимных корреляций, получаемые без какого-либо усреднения:



При использовании оценок формула принимает следующий вид :

(7)

Выражение, стоящее в скобках представляет собой разность между образцовым сигналом и выходным сигналом фильтра на *k-*м шаге, то есть ошибку фильтрации e(k). С учетом этого выражение для рекурсивного обновления коэффициентов фильтра оказывается очень простым:

(8)

Алгоритм адаптивной фильтрации, основанный на формуле (8), получил название LMS (Least Mean Square, метод наименьших квадратов). Можно получить ту же формулу и несколько иным образом: использовав вместо градиента статистически усредненного квадрата ошибки  градиент его мгновенного значения .

Анализ сходимости алгоритма LMS показывает, что верхняя граница для размера шага в данном случае является меньшей, чем при использовании истинных значений градиента. Эта граница примерно равна



где — собственные числа корреляционной матрицы, **R** а — средний квадрат входного сигнала фильтра.

Скорость адаптации, а также составляющая шума вектора весовых коэффициентов зависит от значения параметра μ. Чем меньше μ, тем ближе вектор коэффициентов фильтра приближается к оптимальному но и тем больше времени на это затрачивается. Следовательно размер шага должен находиться в следующих пределах:

.

В общем случае собственные числа корреляционной матрицы неизвестны. Но для вычисления длины шага можно заменить его верхней границей



Формула для расчета коэффициентов фильтра будет следующая:



Основным достоинством алгоритма LMS является предельная вычислительная простота — для подстройки коэффициентов фильтра на каждом шаге нужно выполнить *N+1* пар операций «умножение–сложение». Платой за простоту является медленная сходимость и повышенная (по сравнению с минимально достижимым значением) дисперсия ошибки в установившемся режиме — коэффициенты фильтра всегда флуктуируют вокруг оптимальных значений , что и увеличивает уровень выходного шума.

Существует большое число модификаций алгоритма LMS, направленных на ускорение сходимости либо на уменьшение числа арифметических операций. Ускорение сходимости может быть достигнуто за счет улучшения используемой оценки градиента, а также за счет преобразования входного сигнала с целью сделать его отсчеты некоррелированными. Уменьшение вычислительной сложности может быть достигнуто, в частности, за счет использования в не самих сигнала ошибки и содержимого линии задержки фильтра, а лишь их знаков. Это позволяет полностью избавиться от операций умножения при обновлении коэффициентов фильтра. В целом следует отметить, что требования ускорения сходимости и сокращения вычислительных затрат являются противоречивыми.

## 2.3 Рекурсивный метод наименьших квадратов (RLS)

Рассматривая статистическую задачу оптимизации, мы считали входной сигнал случайным процессом и минимизировали средний квадрат ошибки воспроизведения входного сигнала. Однако возможен другой подход, не использующий статистические методы.

Пусть, как и раньше, обработке подвергается последовательность, состоящая из *K* отсчетов *k(x),* коэффициенты нерекурсивного фильтра образуют вектор-столбец W**,** а отсчеты образцового сигнала равны  *d(k).* Выходной сигнал фильтра определяется формулой (1), а ошибка воспроизведения образцового сигнала — формулой (2). Теперь оптимизационная задача формулируется так: нужно отыскать такие коэффициенты фильтра W**,** чтобы норма ошибки воспроизведения образцового сигнала была минимальной:

 (9)

Для решения задачи в выражениях (1) и (2) необходимо перейти к матричной записи вдоль координаты *k,* получив формулы для векторов столбцов выходного сигнала y и для ошибки воспроизведения входного сигнала e:

 (10)

Здесь d- вектор-столбец отсчетов образцового сигнала, а U – матрица, столбцы которой представляют собой содержимое линии задержки фильтра на разных тактах:



Выражение для нормы ошибки можно переписать в матричном виде следующим образом:

 (11)

Подставив (2.16) в (2.17), имеем



Для нахождения минимума необходимо вычислить градиент данного функционала и приравнять его к нулю:



Отсюда легко получается искомое оптимальное решение:

 (12)

В принципе, в процессе приема сигнала можно на каждом очередном шаге пересчитывать коэффициенты фильтра непосредственно по формуле (12), однако это связано с неоправданно большими вычислительными затратами. Действительно, размер матрицы **U** постоянно увеличивается и, кроме того, необходимо каждый раз заново вычислять обратную матрицу .

Сократить вычислительные затраты можно, если заметить, что на каждом шаге к матрице **U** добавляется лишь один новый столбец, а к вектору **d** — один новый элемент. Это дает возможность организовать вычисления *рекурсивно*. Соответствующий алгоритм называется *рекурсивным методом наименьших квадратов* (Recursive Least Square, RLS).

При использовании алгоритма RLS производится рекурсивное обновление оценки обратной корреляционной матрицы , а вывод формул основывается на следующем матричном тождестве:

 (13)

где A и C — квадратные невырожденные матрицы (необязательно одинаковых размеров), а B и D — матрицы совместимых размеров.

Применение формулы (13) для рекурсивного обновления обратной корреляционной матрицы P в сочетании с исходной формулой (12) для коэффициентов оптимального фильтра дает следующую последовательность шагов адаптивного алгоритма RLS.

1. При поступлении новых входных данных u(k) производится фильтрация сигнала с использованием текущих коэффициентов фильтра w(k-1) и вычисление величины ошибки воспроизведения образцового сигнала:



2. Рассчитывается вектор-столбец коэффициентов усиления(следует отметить, что знаменатель дроби в следующих двух формулах является скаляром, а не матрицей):

 (14)

3. Производится обновление оценки обратной корреляционной матрицы сигнала:

 (15)

4. Наконец, производится обновление

коэффициентов фильтра:

 (16)

Начальное значение вектора w обычно принимается нулевым, а в качестве исходной оценки матрицы Pиспользуется диагональная матрица вида где C>>1.

В формулах (13) и (15) значениям ошибки на всех временных тактах придается одинаковый вес. В результате, если статистические свойства входного сигнала со временем изменяются, это приводит к ухудшению качества фильтрации. Чтобы дать фильтру возможность отслеживать нестационарный входной сигнал, можно применить в (13) экспоненциальное забывание, при котором вес прошлых значений сигнала ошибки экспоненциально уменьшается:



При использовании экспоненциального забывания формулы (14) и (15) принимают следующий вид:



Главным достоинством алгоритма RLS является быстрая сходимость. Однако достигается это за счет значительно более высокой (по сравнению с алгоритмом LMS) вычисли тельной сложности. При оптимальной организации вычислений для обновления коэффициентов фильтра на каждом такте требуется (2.5*N2+4 N)* пар операций «умножение–сложение».

# 3 Описание выбранных методов.

## 

При решении задачи прогнозирования желаемый сигнал на выходе оптимального фильтра *d(t) = S(t+T)* , где T - время прогнозирования.

На вход прогнозирующего фильтра поступают отсчеты зашумленного сигнала *X(k) = S(k) + ξ(k).*

Введем квадратичный критерий качества прогнозирования

 (173.1)

Реализуем прогнозирующий фильтр в виде цифрового фильтра (ЦФ)

 (183.2)

где W(j) – коэффициенты ЦФ, *k* – номер отсчета дискретизированных сигналов.

 (193.3)

Введем вектор сигнала

 (203.4)

В матричной форме можно записать

 (213.5)

Считаем, что *M{ξ(k)}=0.*

Желаемый сигнал *d(k) =S(k+T)=M(S(k+T) + ξ(k+T)}=M{X(k+T}.*

Ошибка прогнозирования



Квадратичный критерий качества прогнозирования



## 3.1 Дискретный прогнозирующий фильтр Винера

Рассмотрим алгоритм настройки ОФ на основе оптимальной винеровской фильтрации.

Обозначим через Rxкорреляционную матрицу сигнала помехи Х размерности N.





Обозначим через  вектор взаимных корреляций между отсчетами образцового сигнала d(k) и отсчетами вектора .

 .



Статистически усреднив, получаем:



 - средний квадрат сигнала S;

 - транспонированный вектор – столбец  взаимных корреляций между k-ым отсчетом желаемого сигнала и задержанными отсчетами сигнала Х***.*** Если рассматриваемые случайные процессы являются совместно стационарными, вектор взаимных корреляций не зависит от номера шага k;

 - корреляционная матрица сигнала ***Z***, имеющая размер [R+1, R+1]. Для стационарного случайного процесса корреляционная матрица имеет вид матрицы Теплица, т.е. на ее диагоналях стоят одинаковые величины.

**

**

*X(k-i) = S(k-I) + ξ(k-I)*

Тогда

*P(i) = M{X(k+T)\*X(k-i)} – M{ξ(k+T)\*S(k-i)} – M{ξ(k+T)\*ξ(k-i)}*

Если отсчеты помехи не коррелированы, т.е. *M{ξ(k+T)\*S(k-i)}=0*, и помеха не коррелирована с информативным сигналом, т.е. *M{ξ(k+T)\*ξ(k-i)} = 0*, то

*P(i) = M{X(k+T)\*X(k-I)} .*

Тогда получаем:



Данное выражение представляет собой квадратичную форму относительно  и при невырожденной матрице  имеет единственный минимум, для нахождения которого необходимо приравнять нулю вектор градиента.



Отсюда получаем искомое решение для оптимальных коэффициентов фильтра:



Прогнозируемое значение S(k+T) получаем в виде:

.

**Реализация алгоритма прогнозирования.**

Имеется реализация зашумленного информативного процесса X(k) из M отсчетов (k=1,2….M), на которой осуществляется настройка прогнозирующего оптимального фильтра Винера.

Вычисляется оценка математического ожидания сигнала Х.



Далее производится оценка дисперсии сигнала Х Dx и производится оценка автокорреляционной функции сигнала X.

 .

Km – количество вычисляемых значений автокорреляционной функции (Km<<M). По виду автокорреляционной функции определяется порядок прогнозирующего фильтра N.

Вычислим элементы корреляционной матрицы сигнала по формуле:

. , где N≈20 – начальное значение из списка, обозначающее порядок прогнозирующего фильтра.

Теперь построим Rx- корреляционную матрицу сигнала помехи Х размерности N.

Затем рассчитаем оценку вектора взаимных корреляций между отсчетами образцового сигнала *d(k)* и отсчетами вектора :



И определим параметры оптимального прогнозирующего фильтра Винера W по следующей формуле



Получив искомое решение для оптимальных коэффициентов фильтра, рассчитаем прогнозируемое значение S(k+T) по формуле:

 , где k=N,N+1..,M

## 3.2 Метод наименьших квадратов (LMS)

Для прогнозирующего адаптивного фильтра LMS итерационный алгоритм настройки коэффициентов W представляется в виде:

, ,

,

,

где Y(k - T) – оценка (прогноз) значения S(k),

-EY(k) – зашумленная оценка погрешности прогнозирования значения S(k), произведенная в момент дискретного времени k-Т, т.е. Y(k-T).

В методе LMS прогнозирование и настройка W производятся одновременно в каждый момент дискретного времени к. Параметр настройки μ подбирается экспериментально при моделировании прогнозирующего фильтра LMS. При большом значении μ нарушается устойчивость настройки (коэффициенты W устремляются в бесконечность). При малом значении μ падает скорость настройки. Поэтому находится значение μ, при котором сохраняется устойчивость и скорость настойки будет достаточно большой.

**Реализация алгоритма прогнозирования**

Необходимо задать два вектора размерностью N+1, где N – порядок фильтра.

 – вектор цифрового фильтра, где на начальном этапе все значения равны 0.

Затем нужно организовать цикл от *N* до *Km*, в котором будут производиться следующие вычисления..

Вычислим выходной сигнал фильтра:

, где *U(k)* – вектор сигнала размерности N+1и представляется как



Таким образом, выходной сигнал фильтра можно записать как



Далее необходимо подсчитать ошибку по формуле .

И в конце цикла пересчитать коэффициенты W фильтра по формуле:



# 4 Структурно-функциональная схема.

Генерация процесса

Генерация процессов

Формирование данных

Отображение процессов

Расшифровка данных

Формирование данных

Запись в базу данных

H

D

D

Расшифровка данных

Прогнозирование процессов

Отображение процессов

Отправка данных на сервер

Сервер

Чтение данных

Запись данных

Отправка данных клиенту

Прогнозирование процесса

# 5 Описание Разработанной системы.

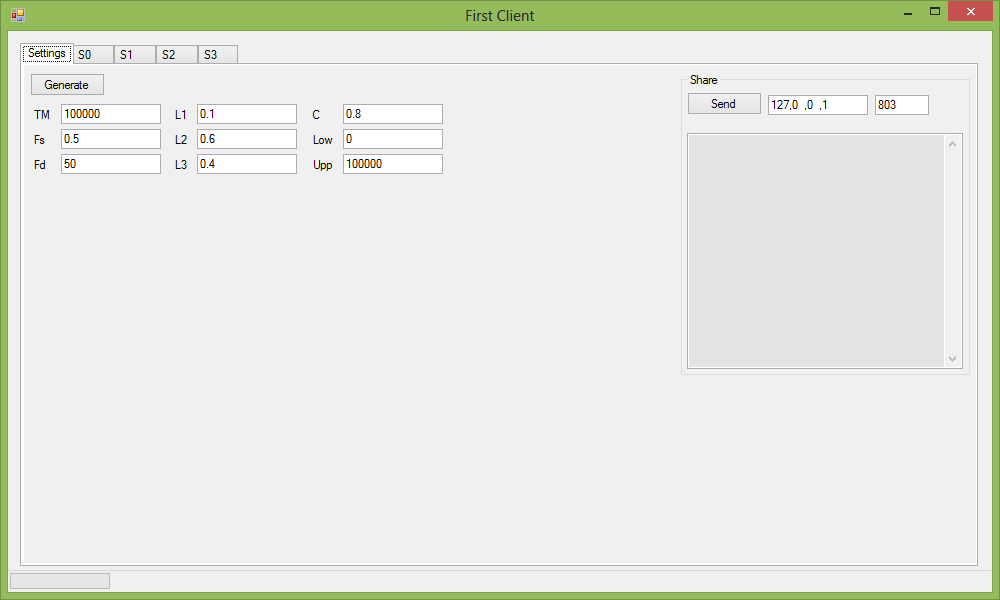


Рис.5.1. Приложение генерации процессов. Вкладка Settings

На Рис.5.1. показано окно программы генерации процессов. Приложение необходимо в качестве псевдослучайного генератора исходных данных для прогнозирования. Окно содержит 5 вкладок, четыре из которых содержат поле для графика одного из четырех генерируемых процессов, а пятая вкладка с именем Settings имеет поля ввода для задания необходимых коэффициентов, кнопку Generate и секцию Share.

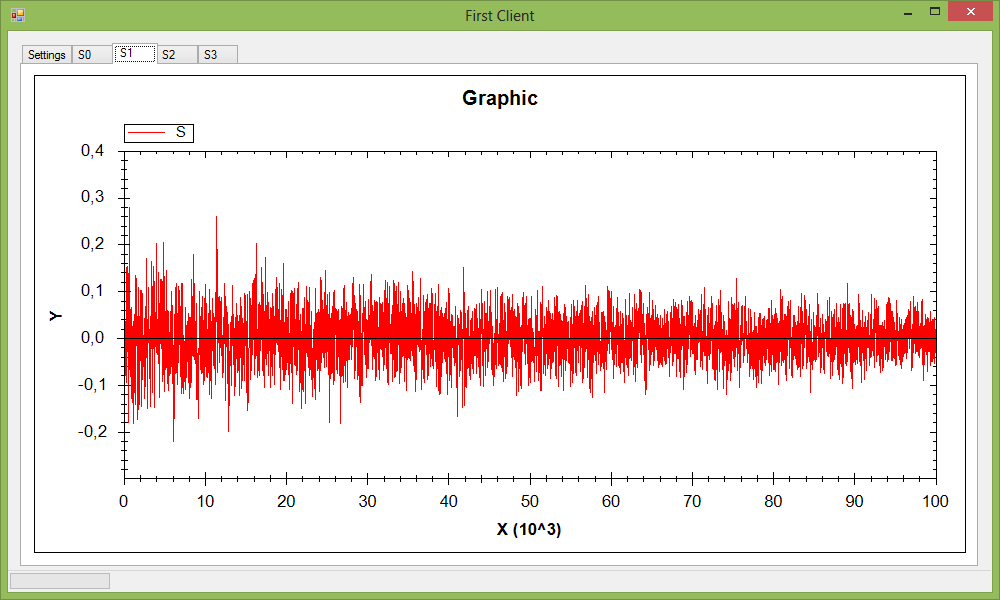


Рис.5.2. Приложение генерации процессов. Вкладка одного из процессов

Кнопка Generate запускает алгоритм генерации 4 процессов, а также запись каждого из отсчетов, количество которых можно задать, в файл.

Секция Share имеет все, что необходимо для передачи файла по сети: поле ввода IP-адреса сервера и поле ввода необходимого порта, кнопка Send, выполняющая команду отправки файла и лог для записи принятых/отправленных сообщений серверу и записи выполнений команд генерации.

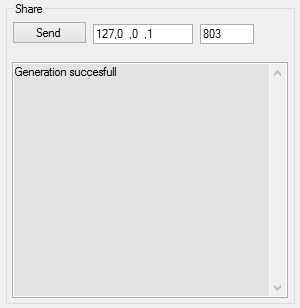


Рис.5.3. Приложение генерации процессов. Секция Share

После нажатия кнопки Send происходит попытка подключения к серверу, если подключение состоялось, то открывается поток записи и чтения и на сервер передаются запросы в текстовой форме. Если сервер положительно ответил на запрос о передаче ему файла, то начинается пересылка данных из файла на сервер.

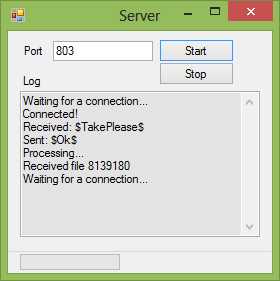


Рис.5.4.Приложение сервера хранения данных

Сервер хранения данных принимает и отправляет данные клиентам по их запросам. Для запуска сервера необходимо ввести порт, через который разрешено соединение на компьютере, и нажать кнопку Start. Сервер создаст «слушателя» в системе и будет ожидать любого подключения к порту со стороны.

Если соединение будет установлено, то сервер перейдет в состояние чтения запросов и будет ожидать всего лишь два возможных запроса в виде текстовых команд: $TakePlease$ и $GivePlease$. Первая команда требует сервер принять данные со стороны клиента, что успешно выполняется и данные записываются в файл. Вторая команда запрашивает у сервера данные из файла и, если данные были приняты ранее, то сервер отправит их клиенту. Каждое действия сервера и команды клиента записываются в окно логгирования. После выполнения запроса сервер вновь ожидает подключения.

Для остановки работы сервера необходимо нажать кнопку Stop, тем самым закрыв любые соединения.

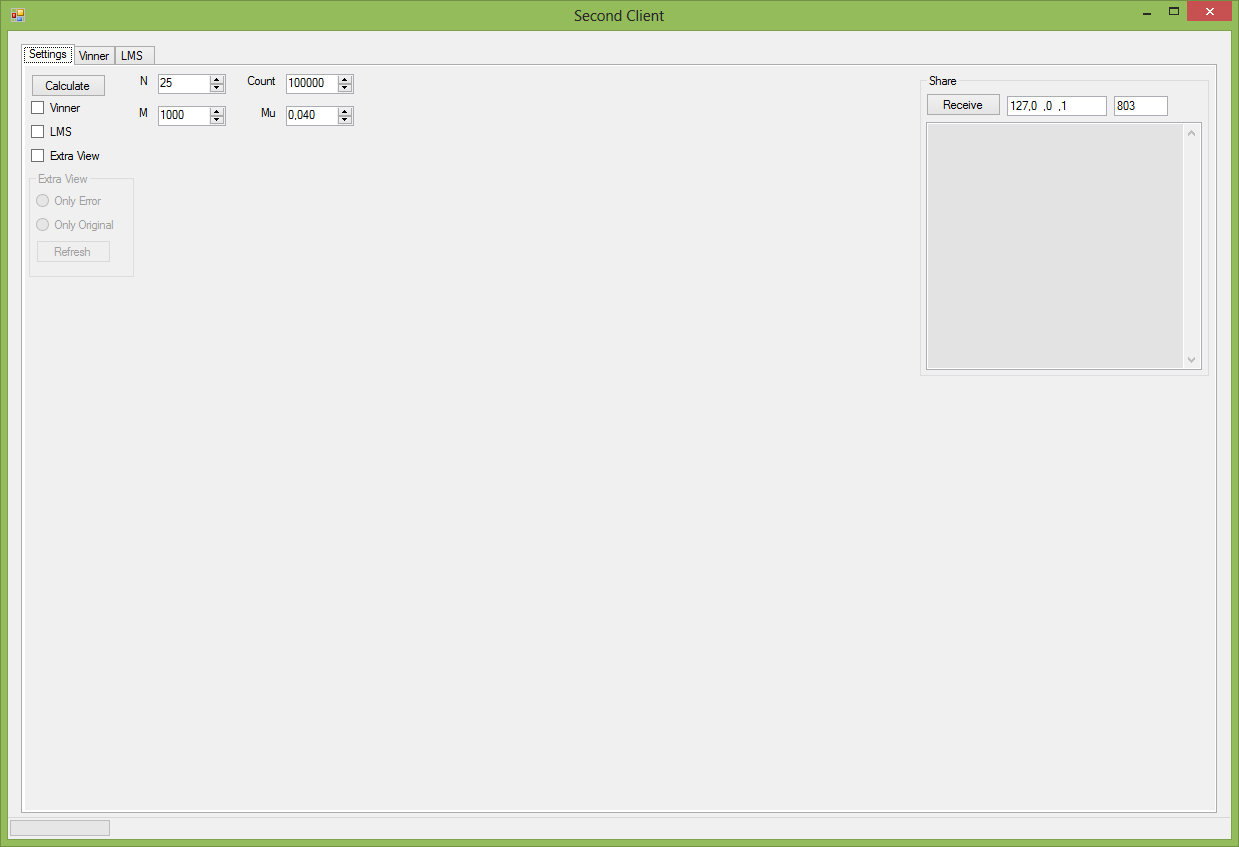


Рис.5.5. Приложения прогнозирования процесса. Главная форма

Приложение прогнозирования процесса принимает данные о процессах с сервера и строит по ним прогнозируемый график. Главная форма содержит три вкладки.

На первой вкладке Settings имеются поля ввода для необходимых коэффициентов, флажки для выбора алгоритма прогнозирования и отображения графиков и секция Share полностью аналогичная одноименной секции в приложении генерации, с отличием лишь в том, что кнопка Receive запускает прием данных.

Кнопка Calculate запускает алгоритм прогнозирования на основе данных принятых от сервера. Также выбор алгоритма осуществляется при помощи выбора флажков на форме. В той же области на форме можно выбрать отображение графиков не только в виде прогнозируемого значения, но и в виде только ошибок или только оригинального графика. Для этого, после выбора флажков, необходимо нажать кнопку Refresh.

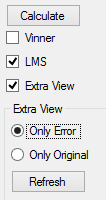


Рис.5.6. Приложения прогнозирования процесса. Выбор алгоритмов прогнозирования

На второй и третьей вкладке расположены графики для всех четырех процессов с прогнозированием по алгоритмам Виннера и LMS соответственно. Как было сказано раньше, можно менять изображение графиков не только прогнозируемых значений, но и оригинала и ошибок.

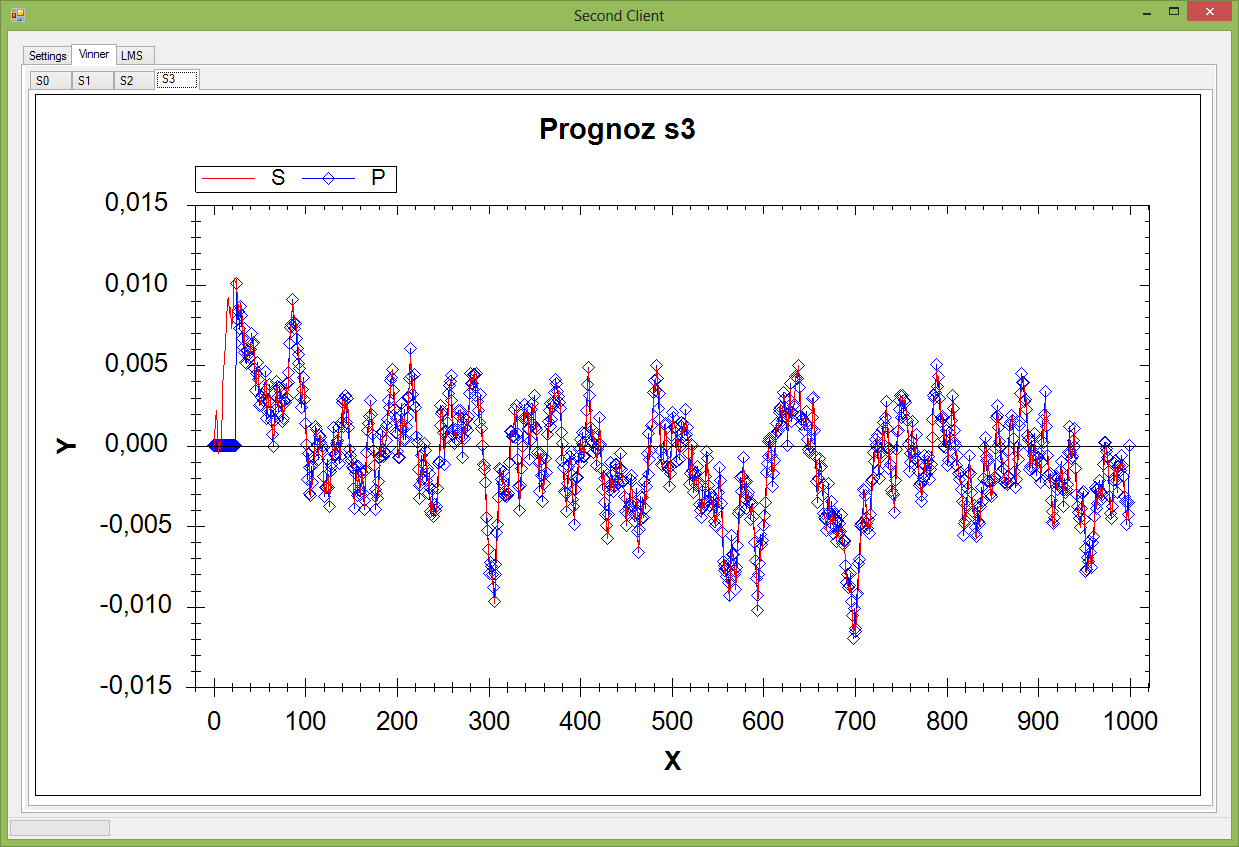


Рис.5.7. Приложения прогнозирования процесса. Вид вкладки с процессами

Стоит отметить, что график прогноза «накладывается» на оригинальный график для более полного сравнения. Однако в качестве отличительных черт оригинального и прогнозируемого графика использовались цвета – оригинальный график имеет красный цвет и не выделяет точку каждого отсчета, прогнозируемый график имеет синий цвет и точка на графике, соответствующая каждому отсчету изображается в виде ромба.

# 6 Результаты моделирования.

## Генерация процессов.

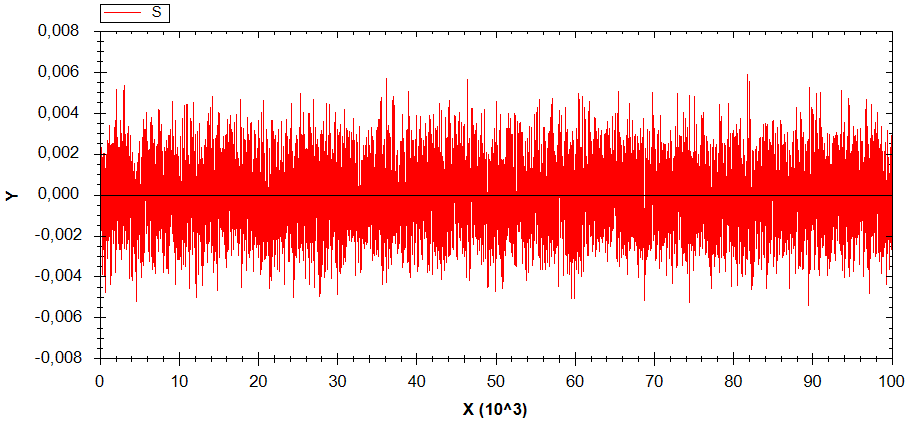


Рис.6.1.Первый процесс

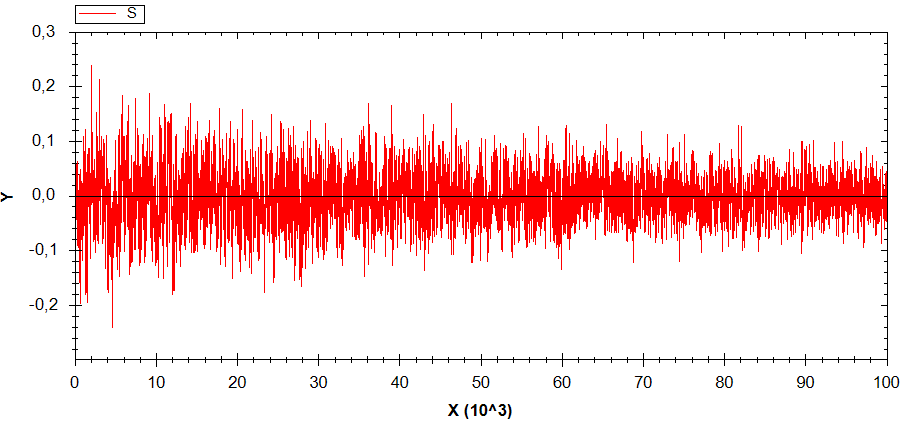


Рис.6.2.Второй процесс

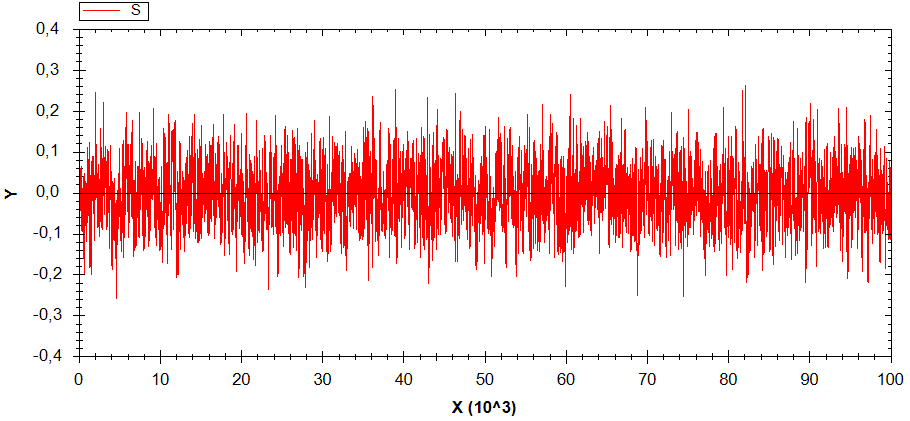


Рис.6.3.Третий процесс

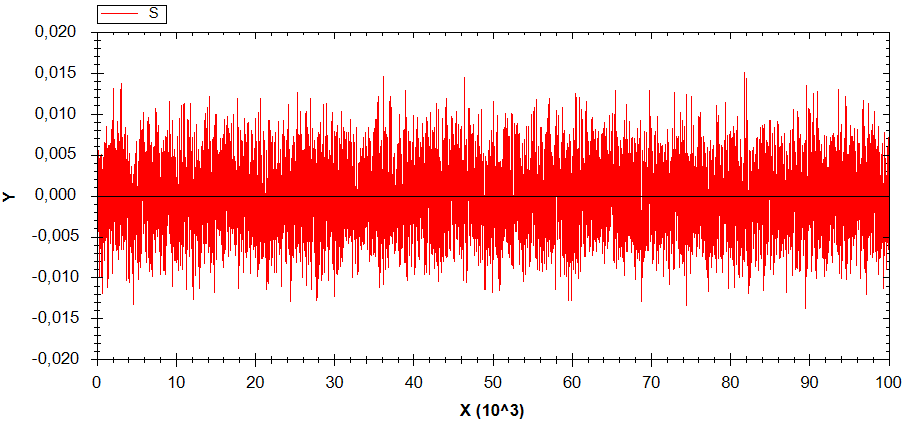


Рис.6.4.Четвертый процесс

## Прогнозирование процессов

Фильтр Виннера

Процесс 1

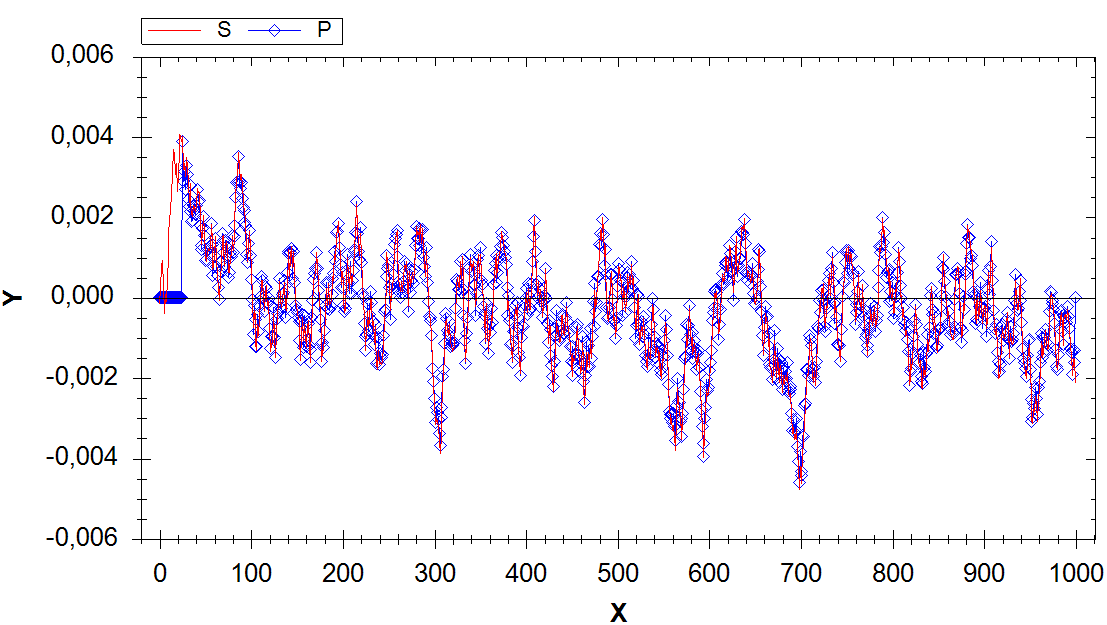


Рис.6.5.Т = 1

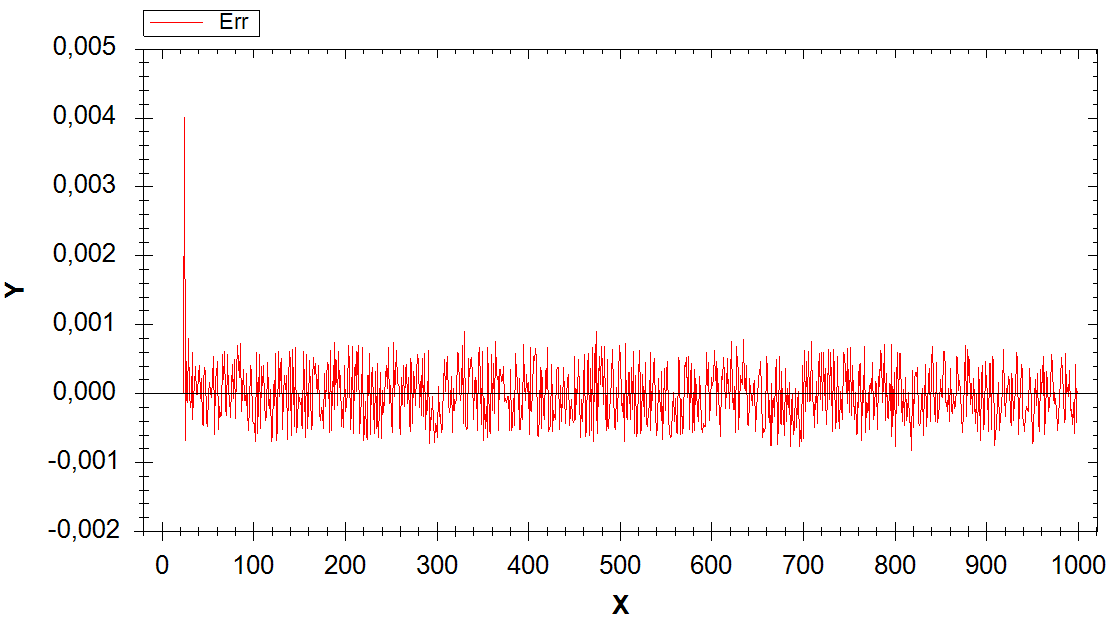


Рис.6.6.График ошибки

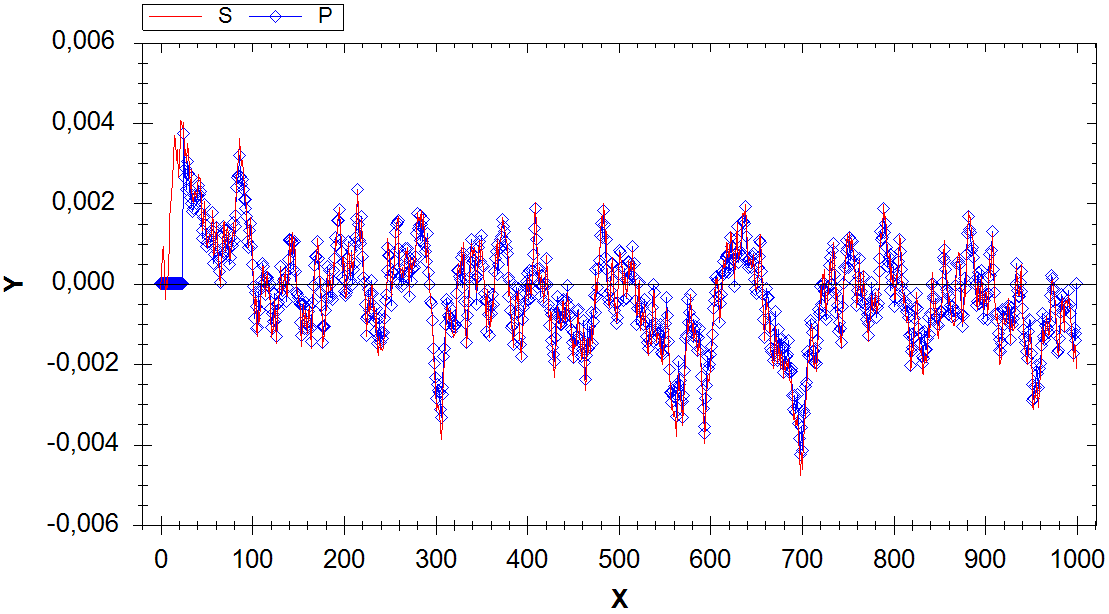


Рис.6.7.Т =2

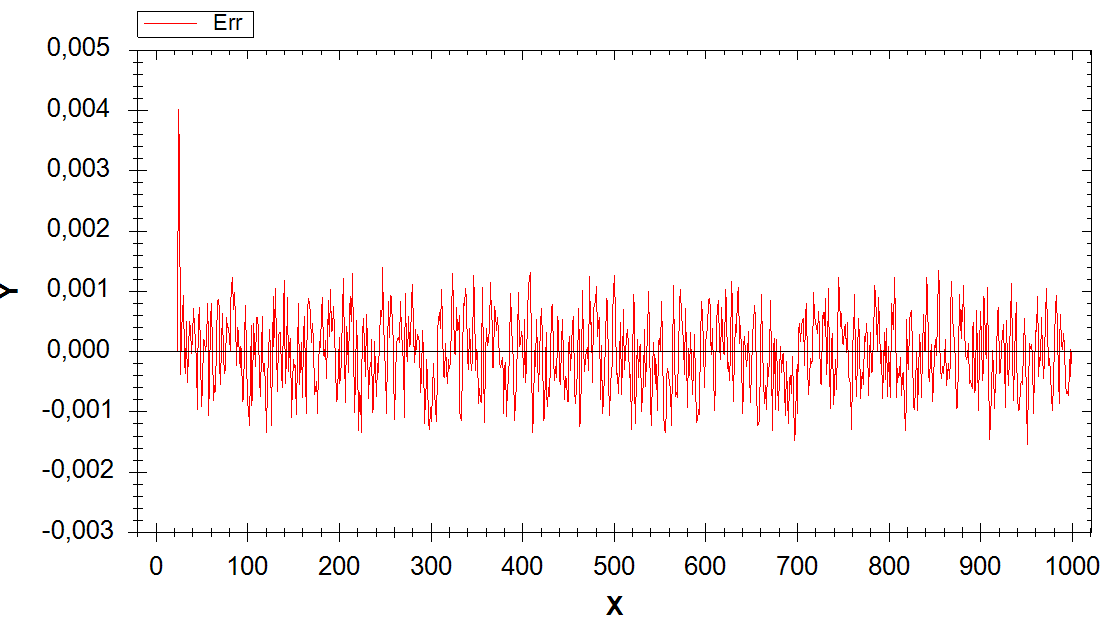


Рис. 6.8.График ошибки

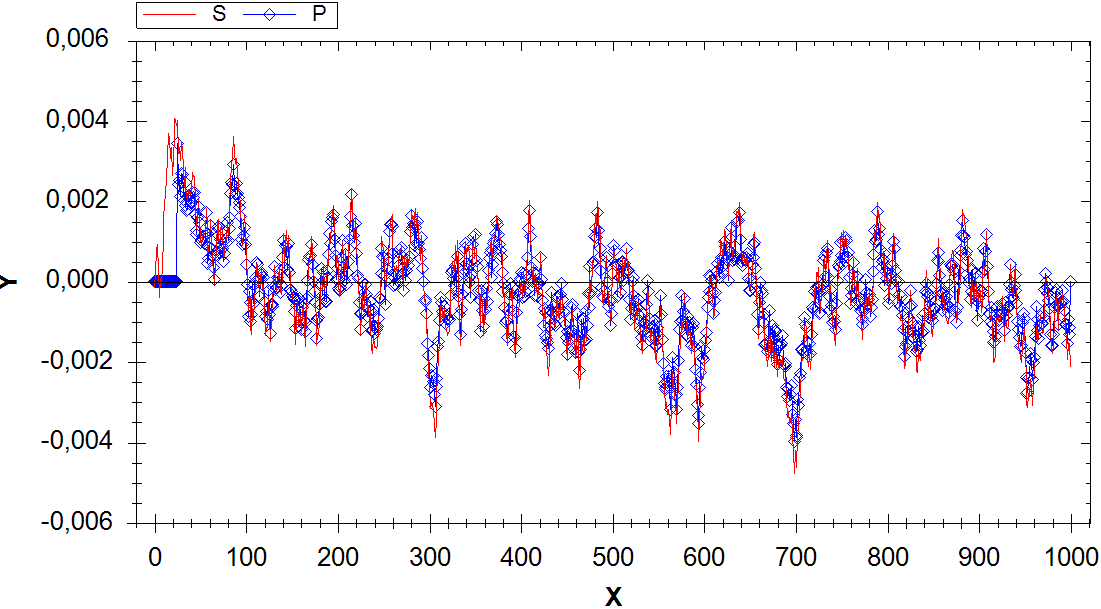


Рис. 6.9.Т=3

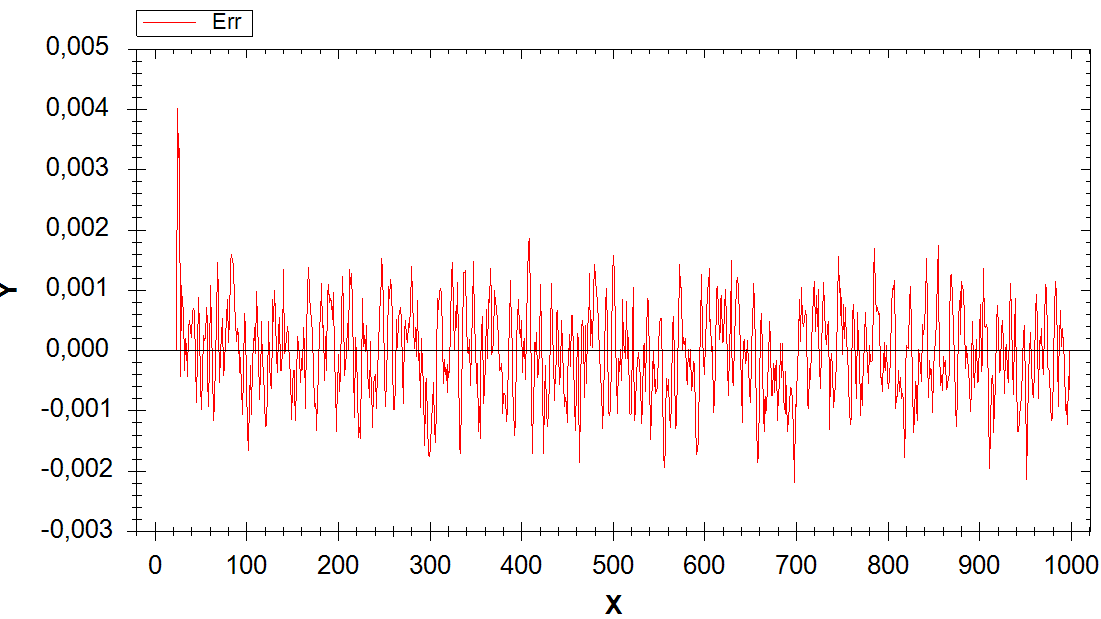


Рис. 6.10.График ошибки

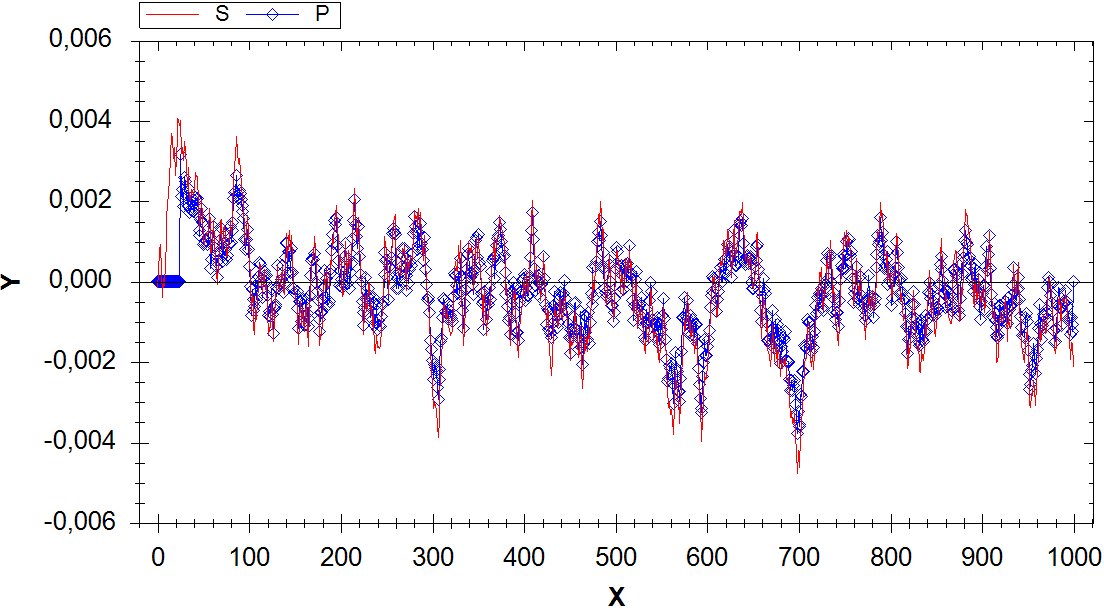


Рис. 6.11.Т=4

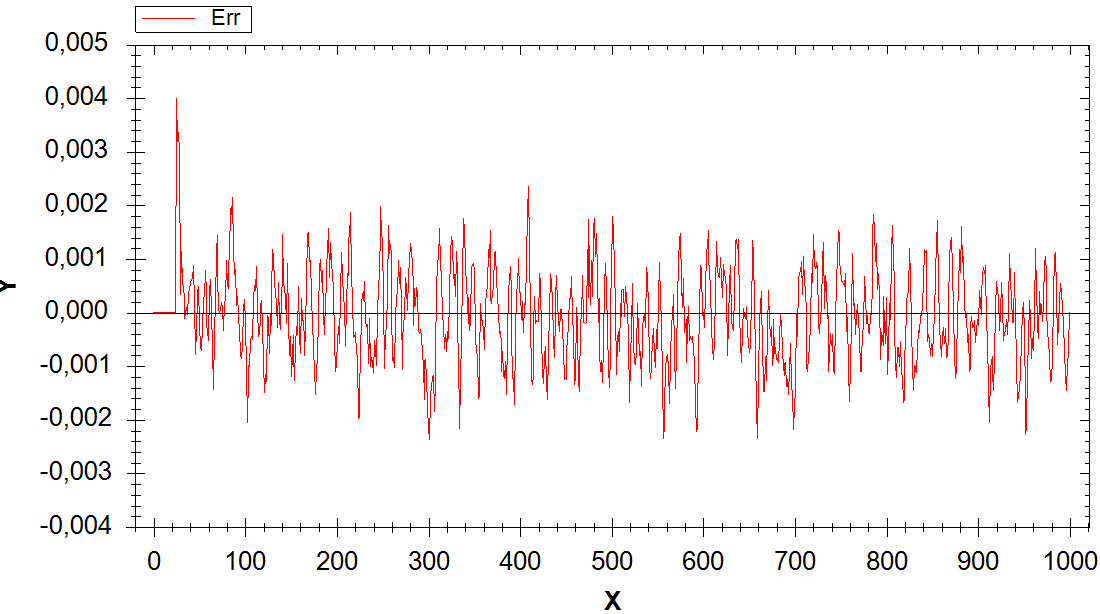


Рис. 6.12.График ошибки

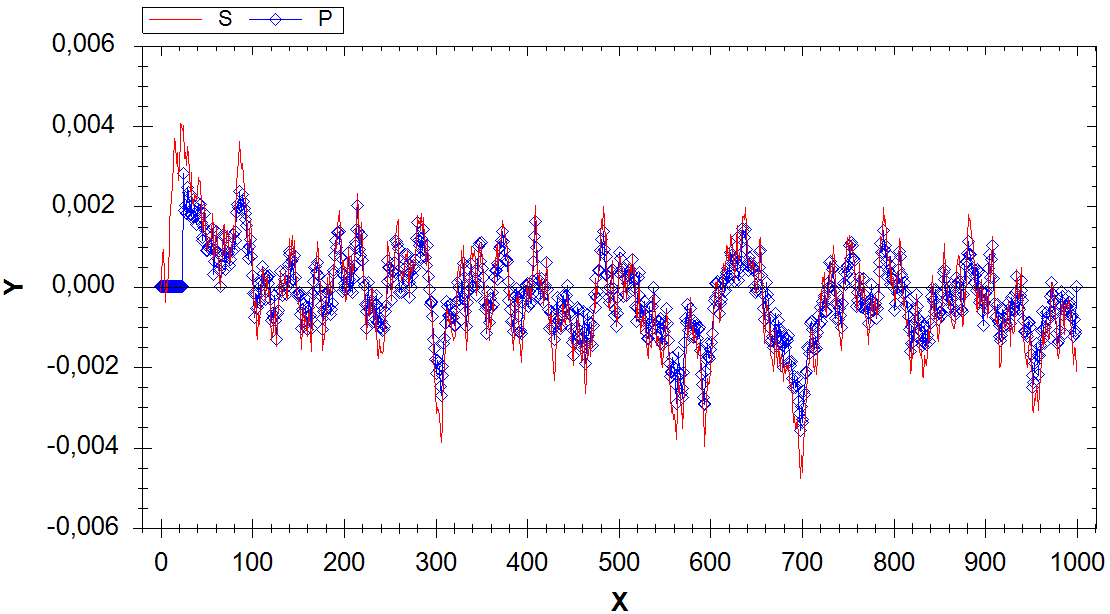


Рис. 6.13.Т=5

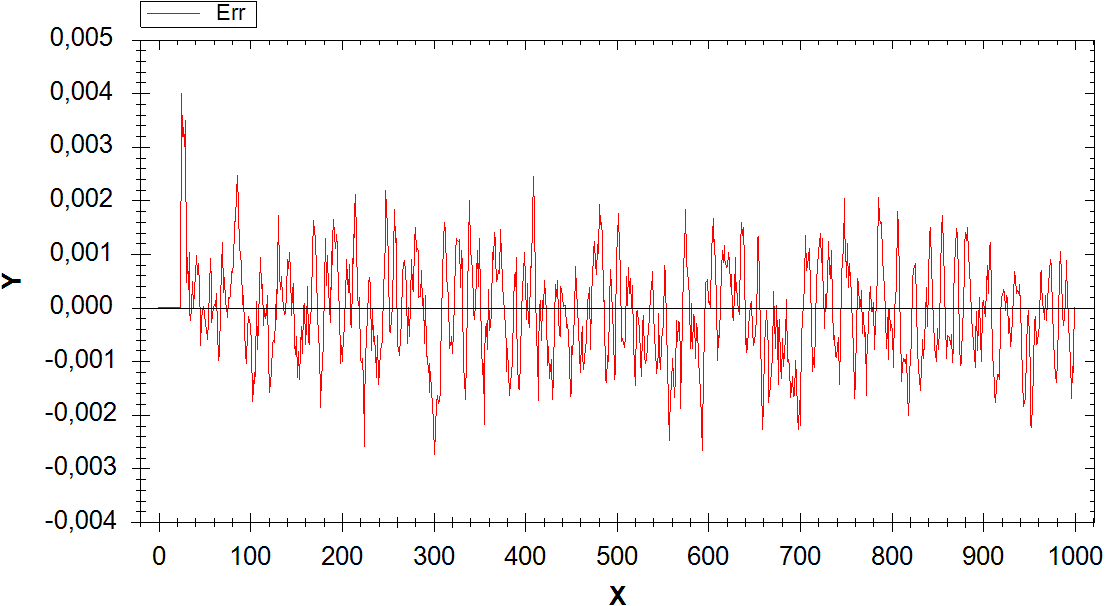


Рис. 6.14.График ошибки

Процесс 2

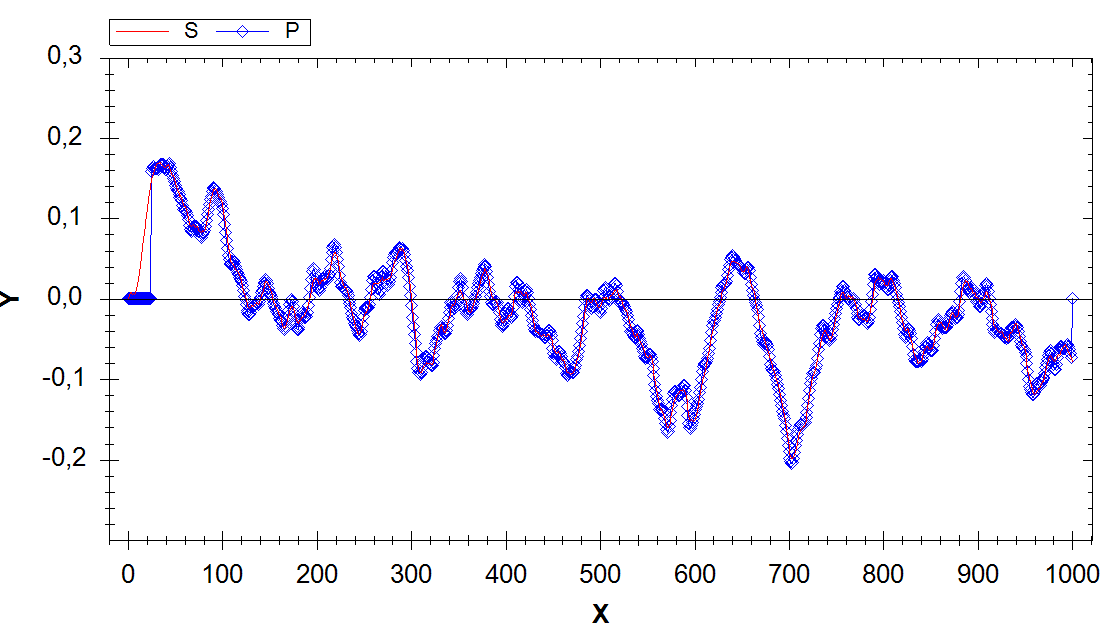


Рис. 6.15.Т=1

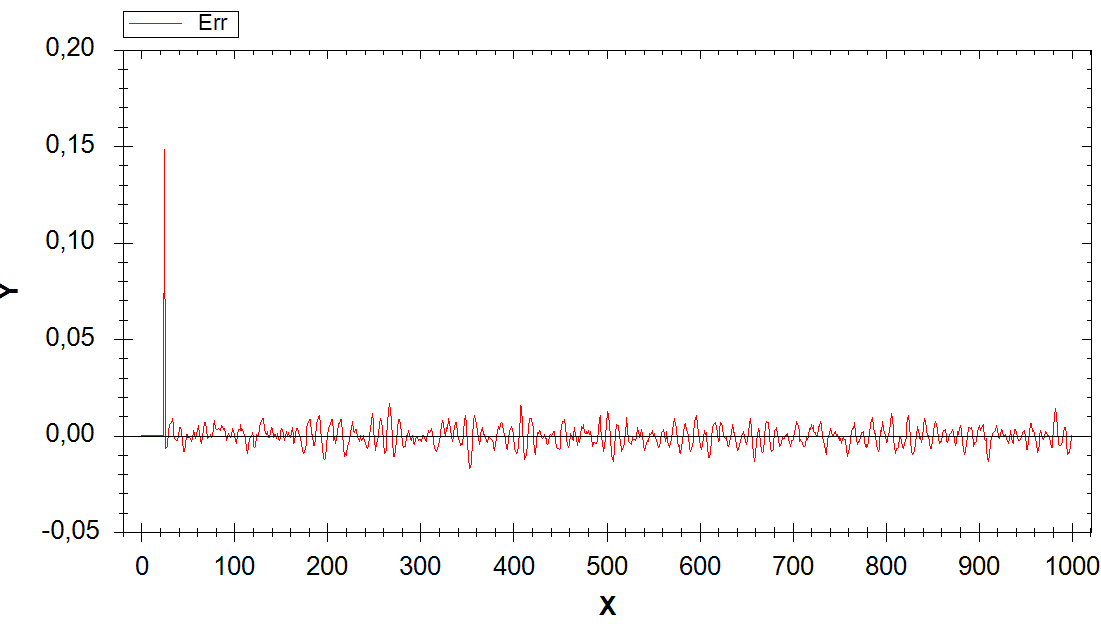


Рис. 6.16.График ошибок

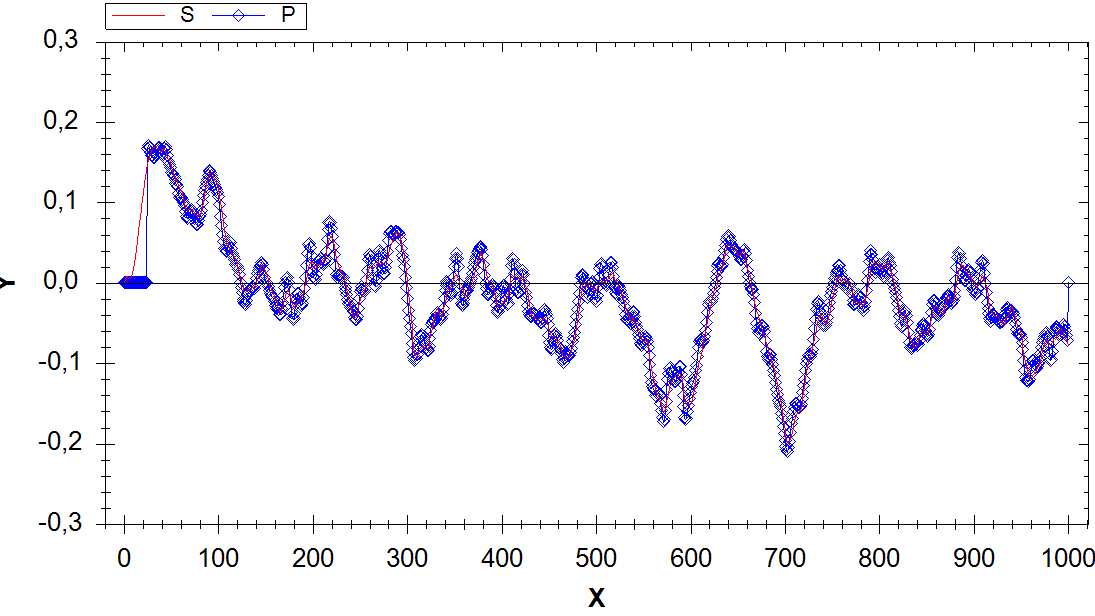


Рис. 6.17.Т=2

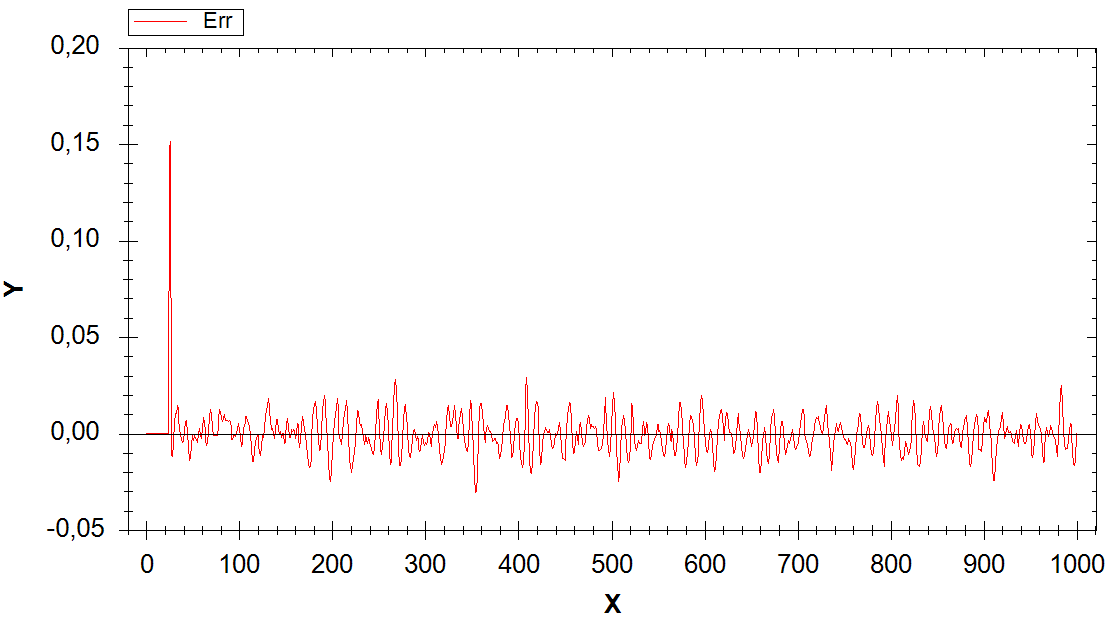


Рис. 6.18.График ошибок

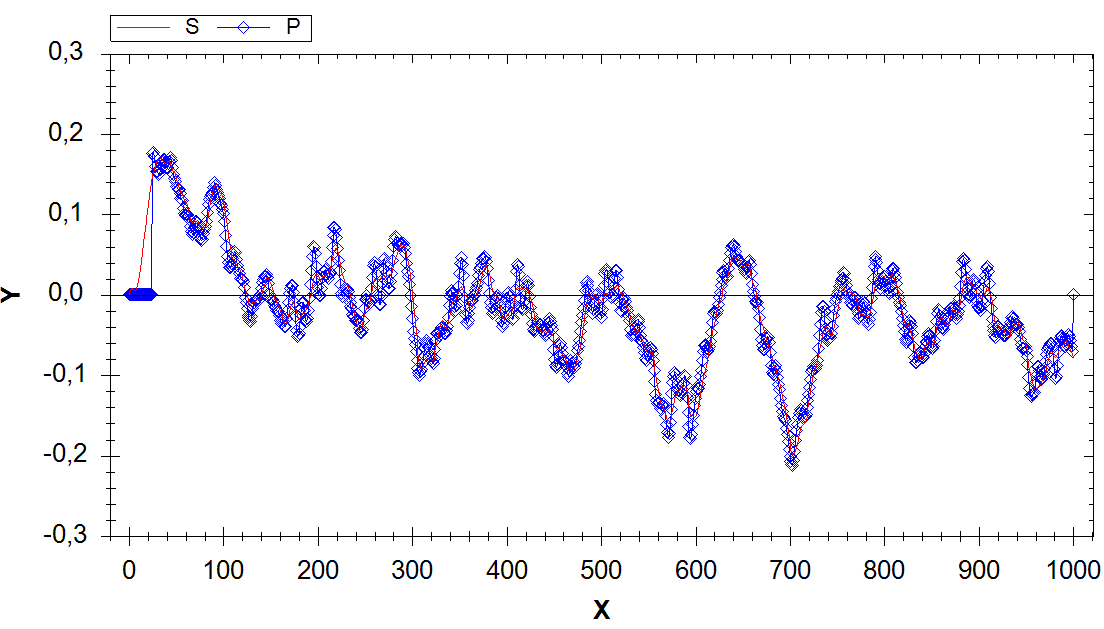


Рис. 6.19.Т=3

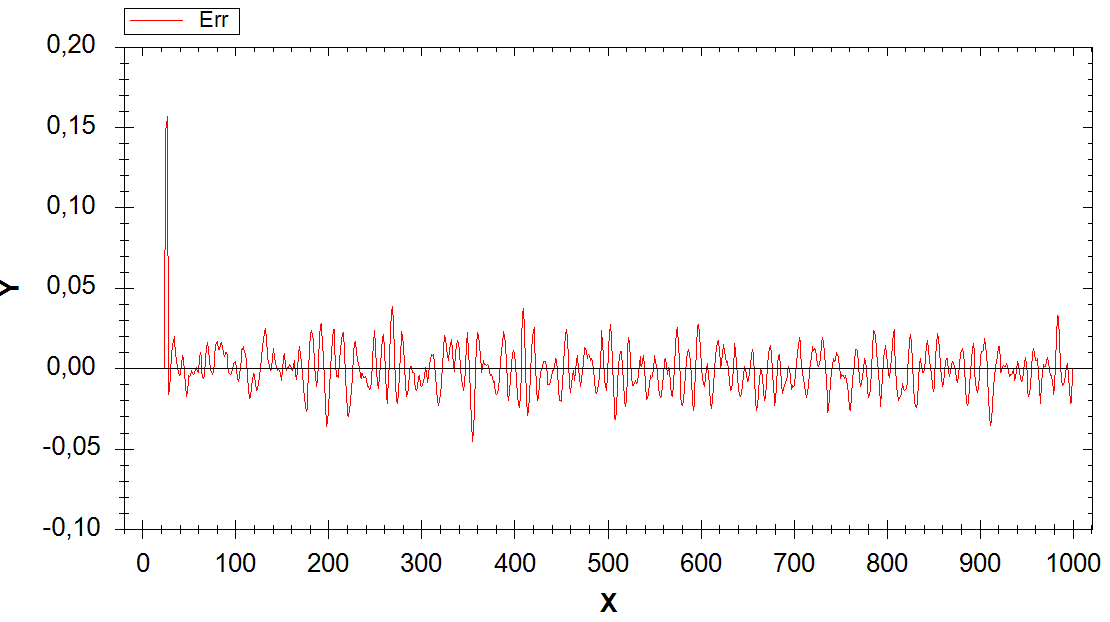


Рис. 6.20.График ошибок

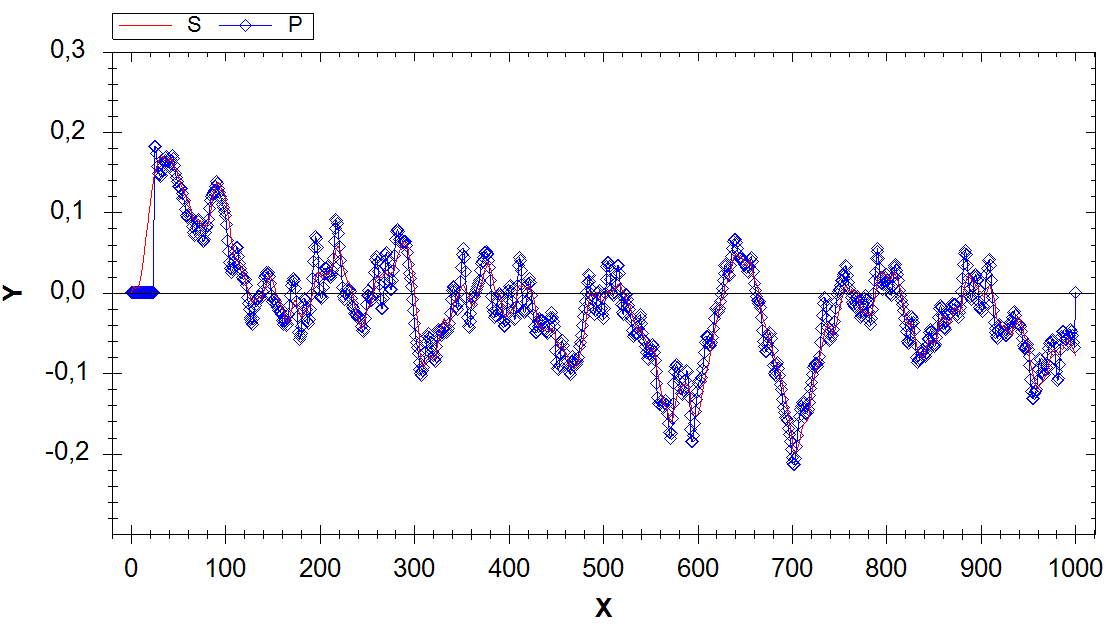


Рис. 6.21.Т=4

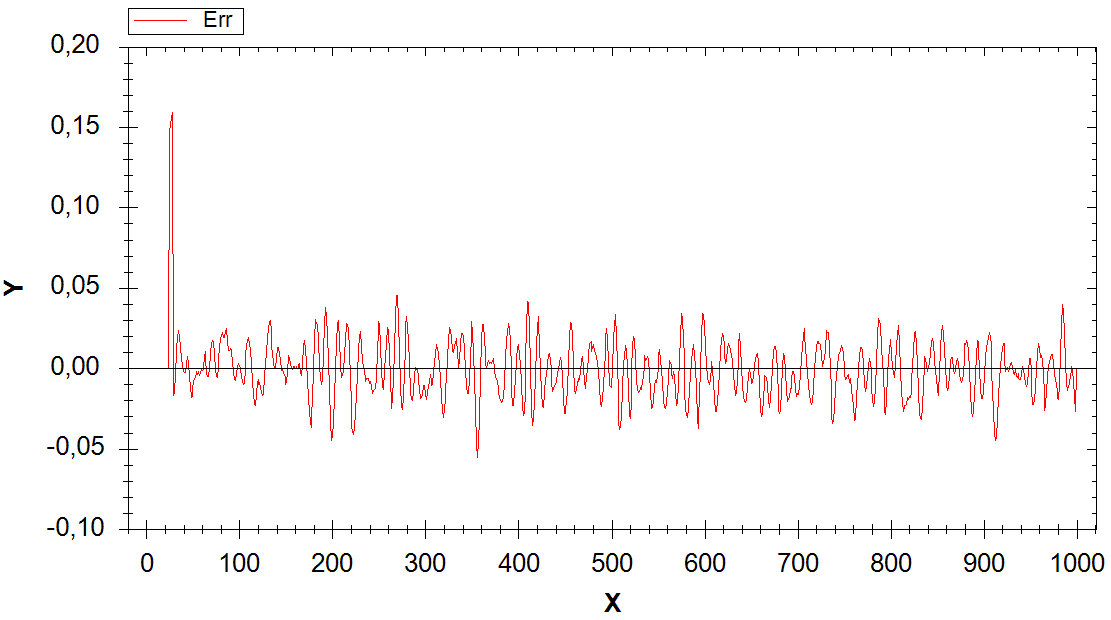


Рис. 6.22.График ошибок

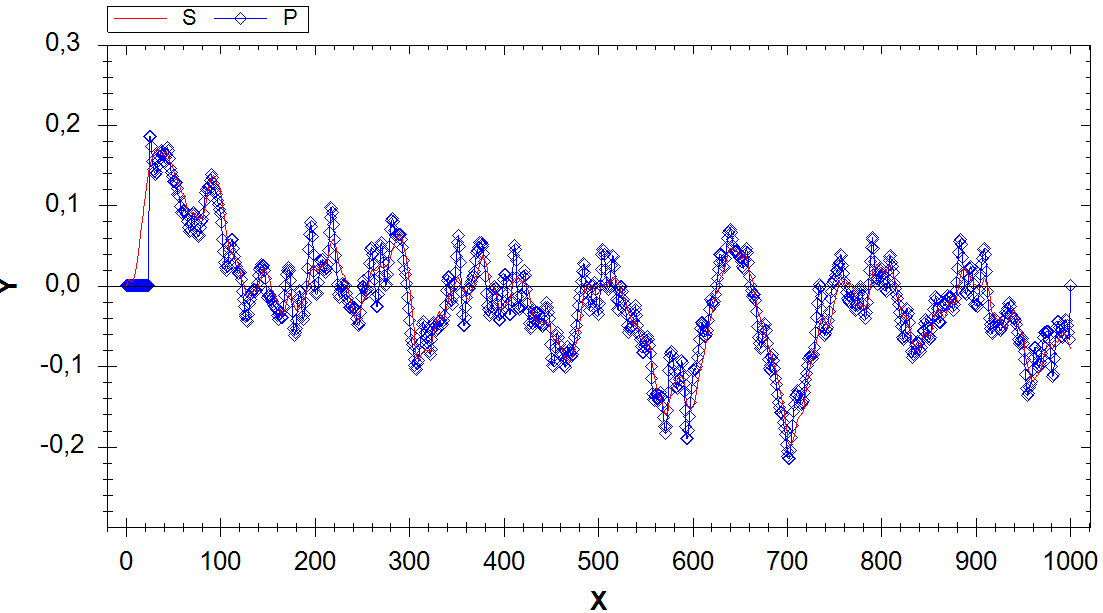


Рис. 6.23.Т=5

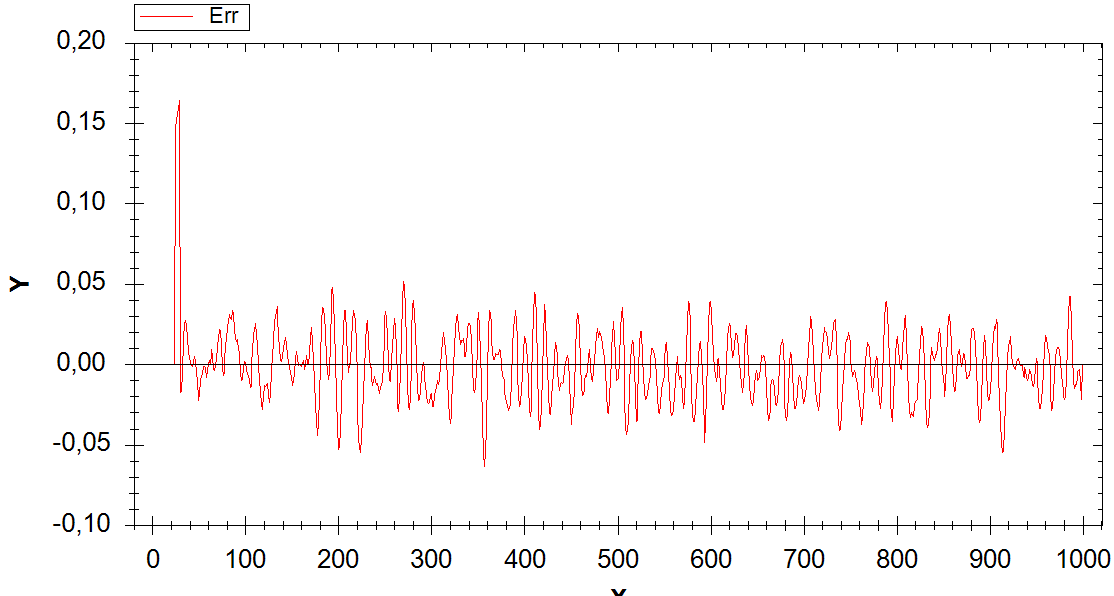


Рис. 6.24.График ошибок

Процесс 3

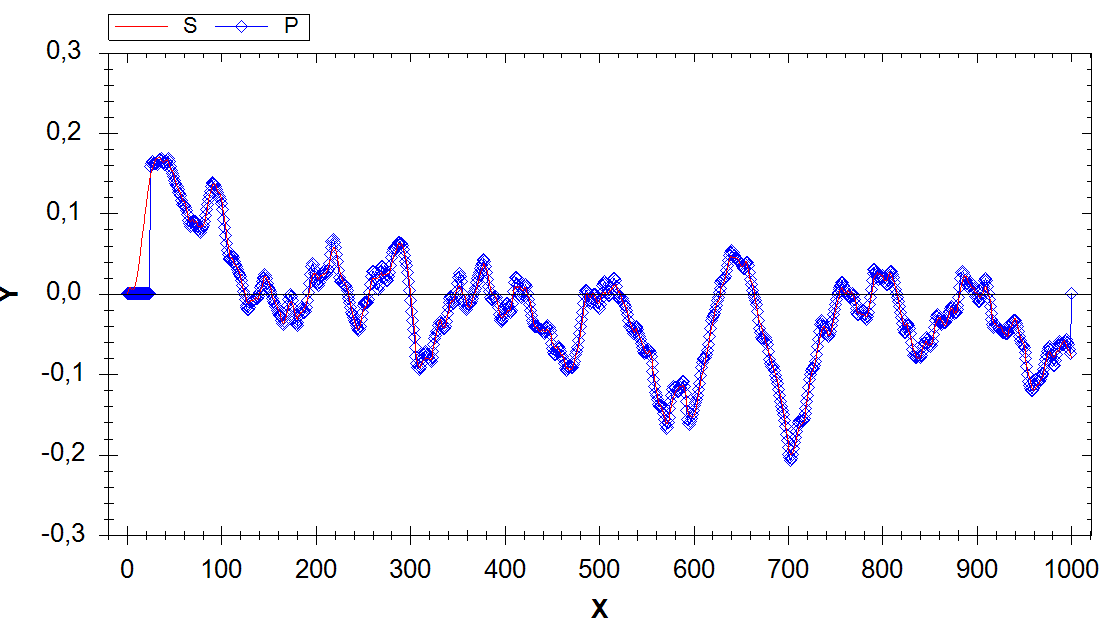


Рис. 6.25.Т=1

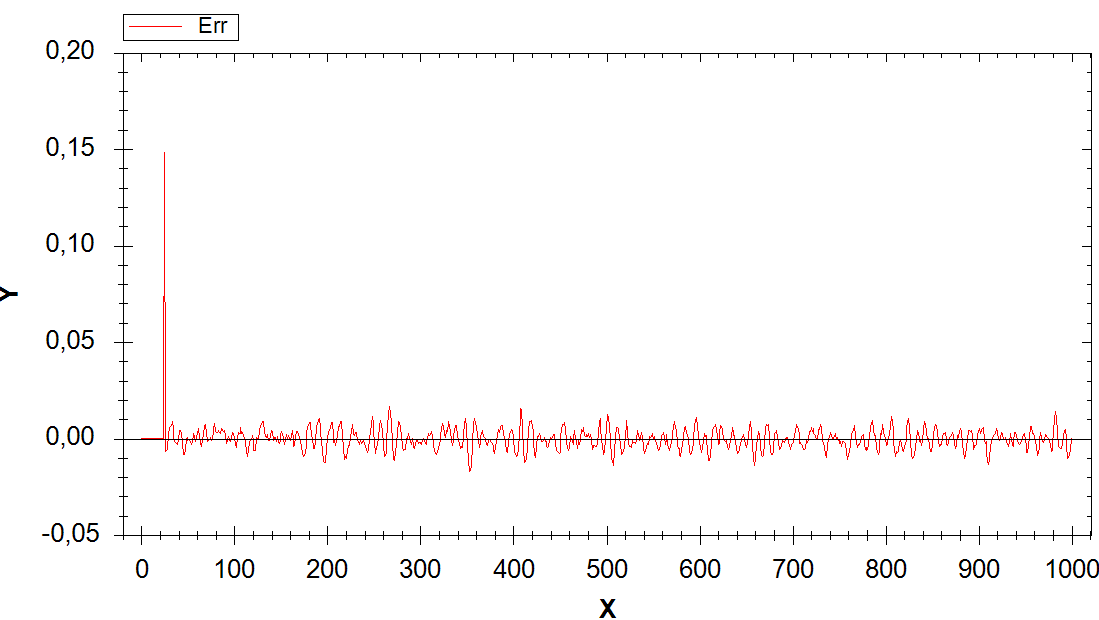


Рис. 6.26.График ошибок

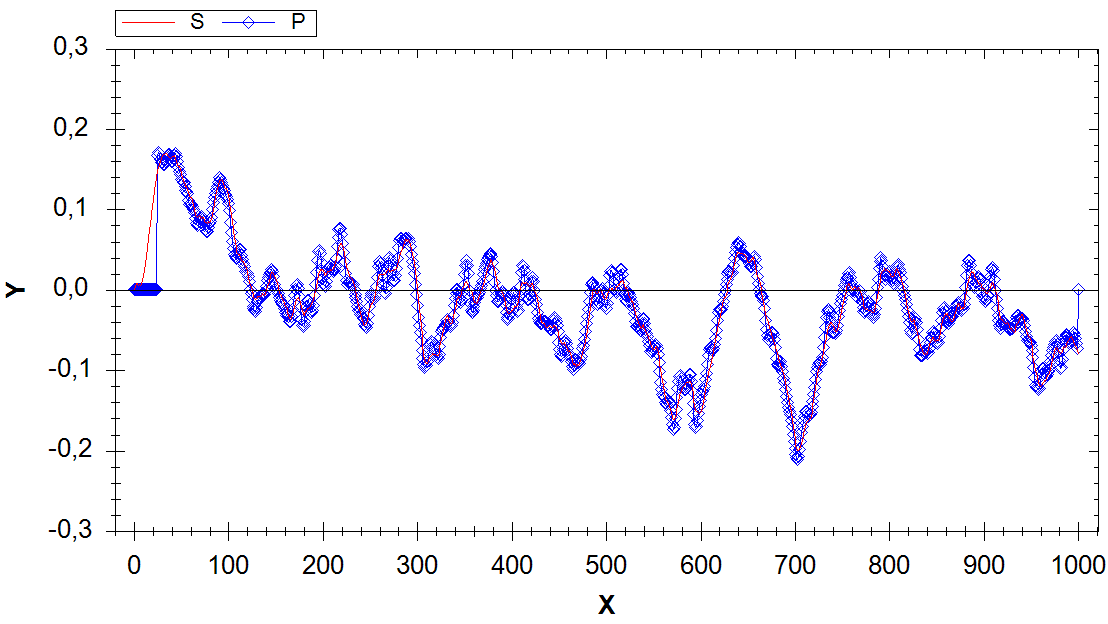


Рис. 6.27.Т=2

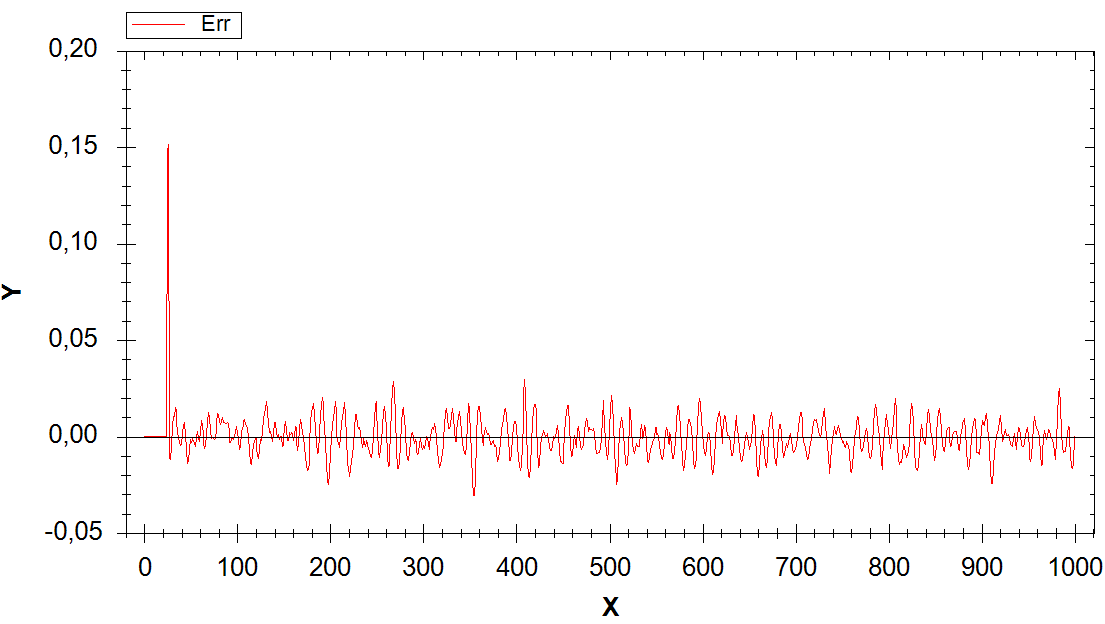


Рис. 6.28.График ошибок

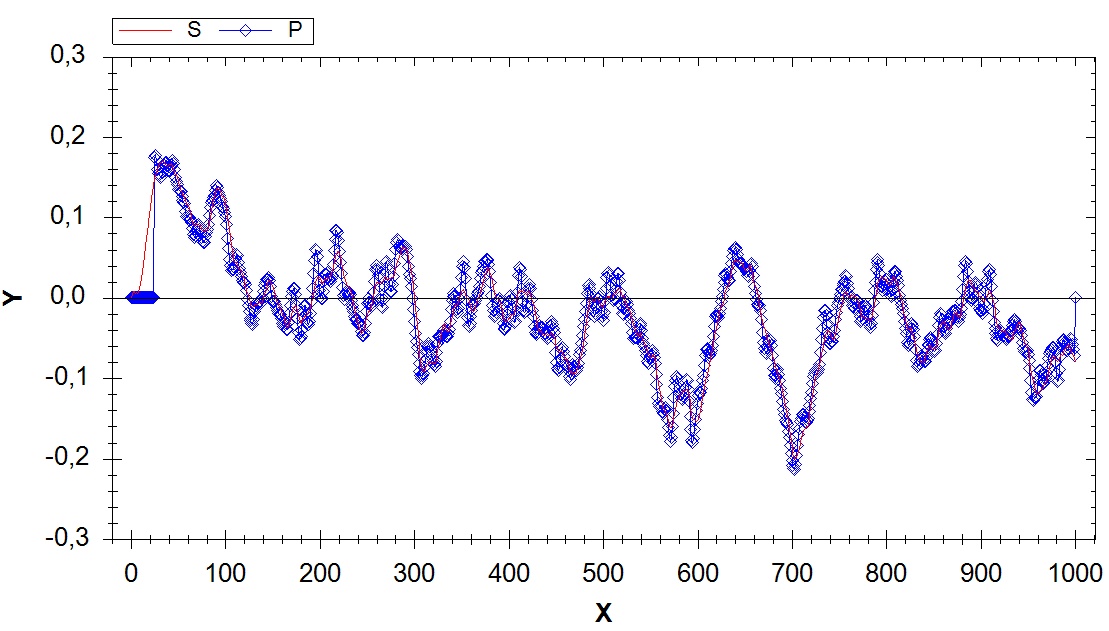


Рис. 6.29.Т=3

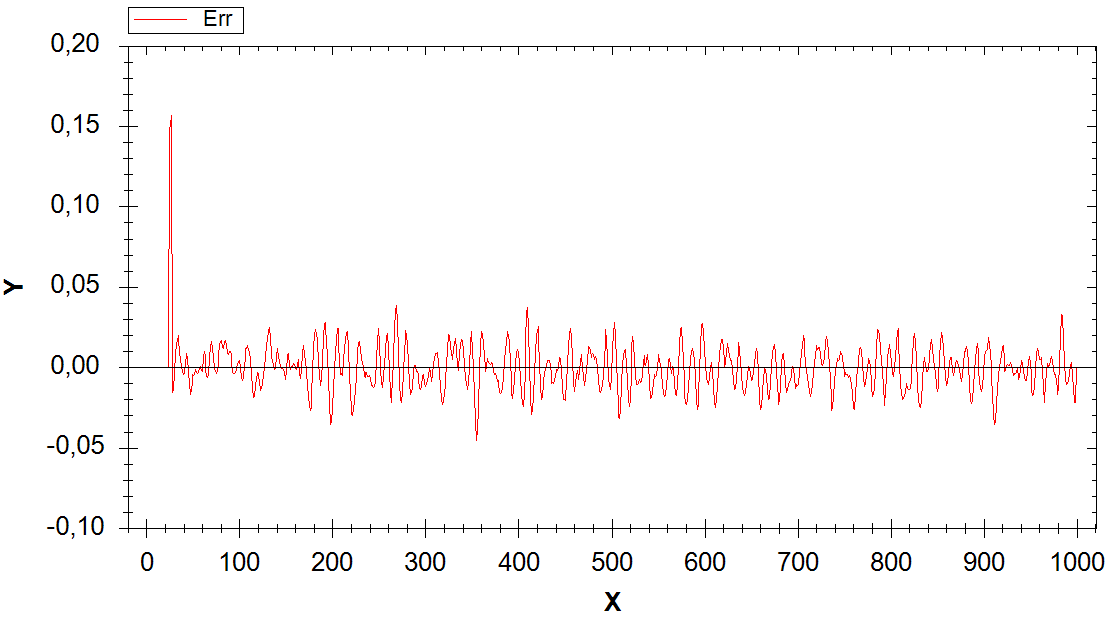


Рис. 6.30.График ошибок

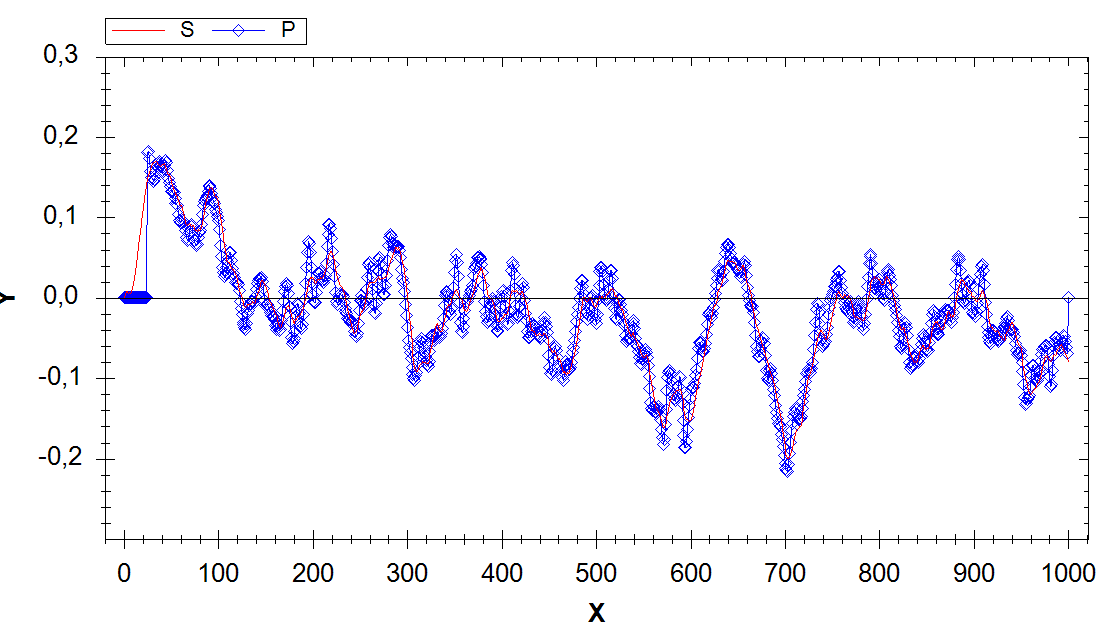


Рис. 6.31.Т=4

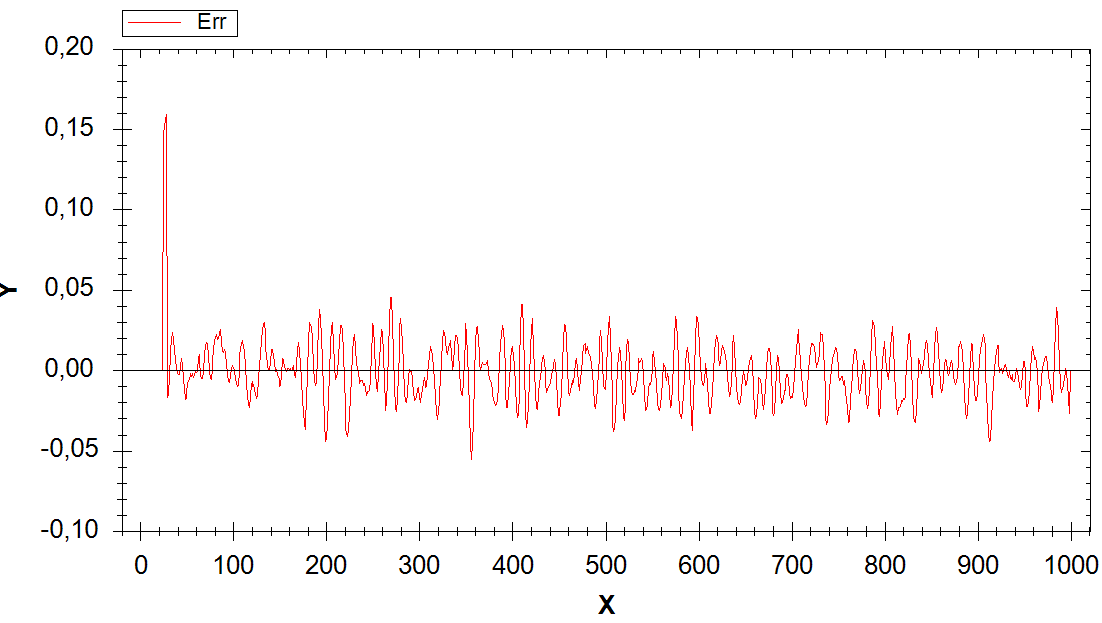


Рис 6.32.График ошибок

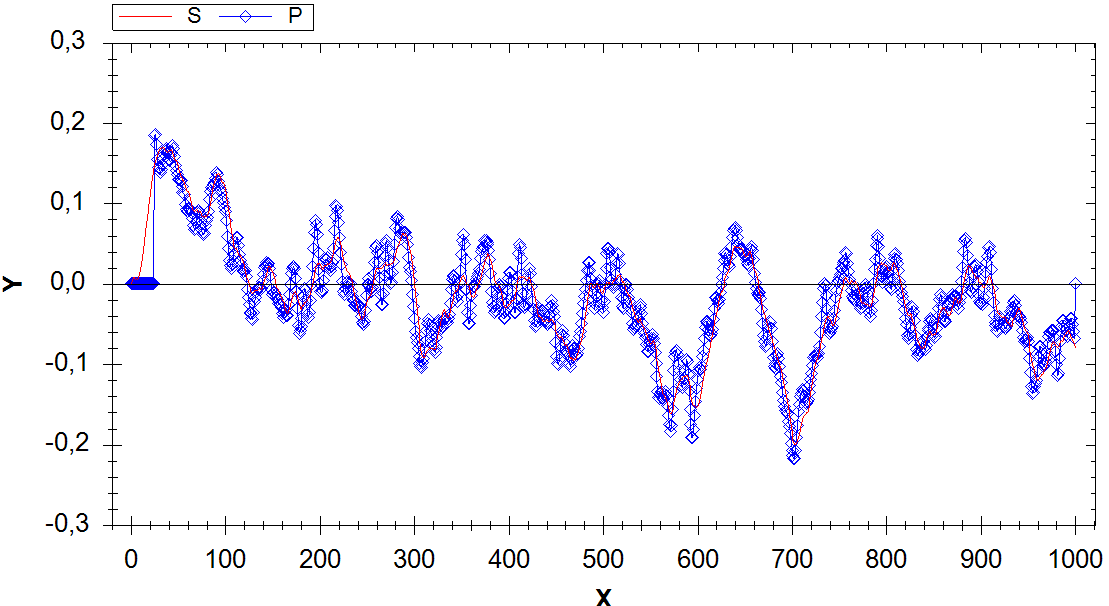


Рис. 6.33.Т=5

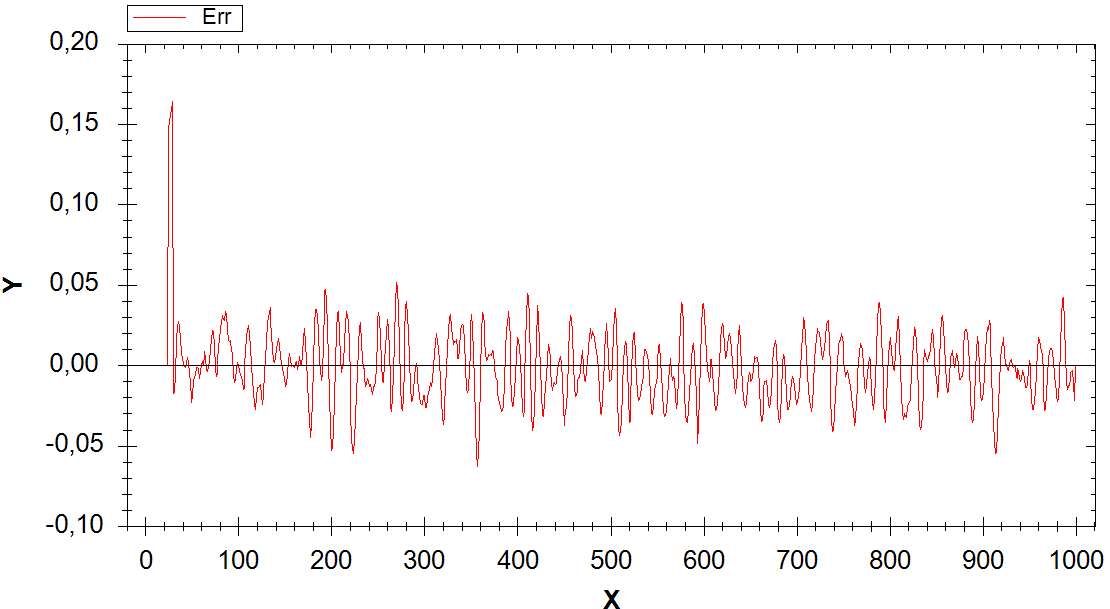


Рис. 6.34.График ошибок

Процесс 4

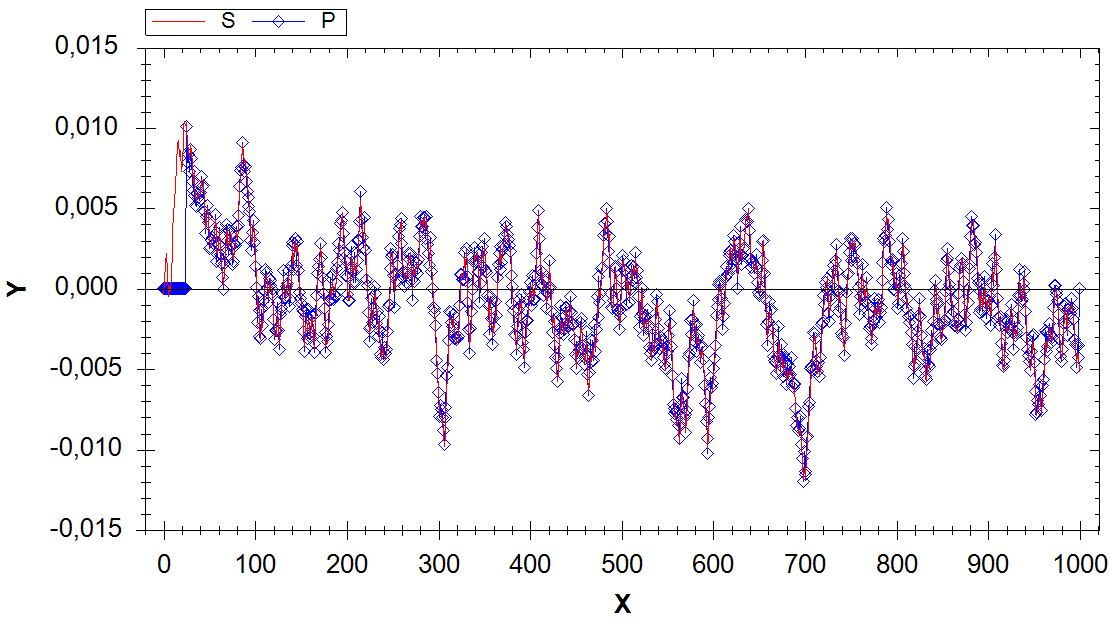


Рис. 6.35.Т=1

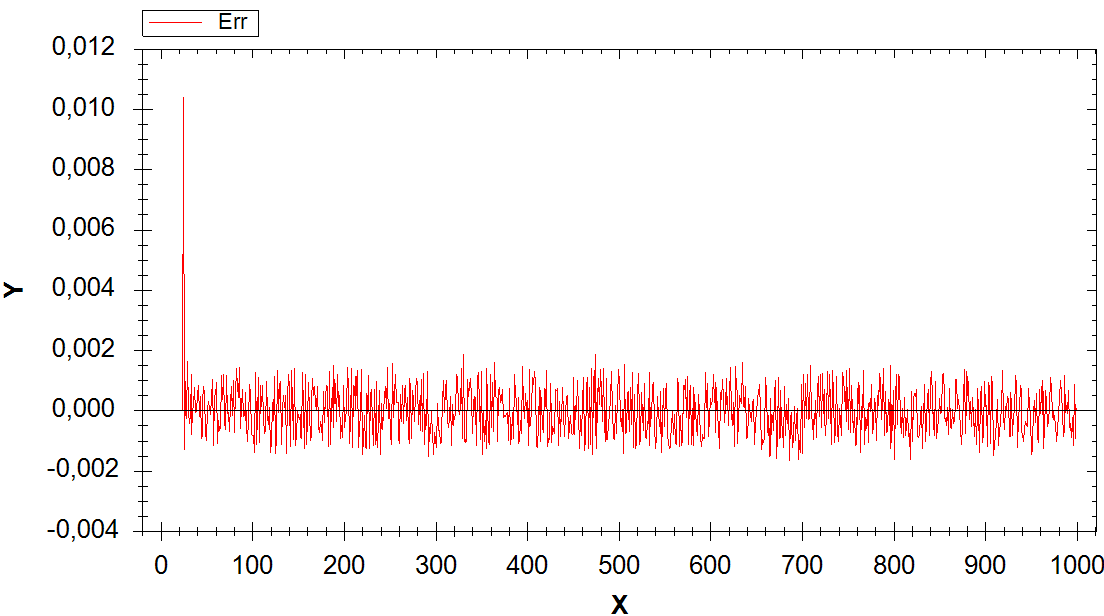


Рис. 6.36.График ошибок

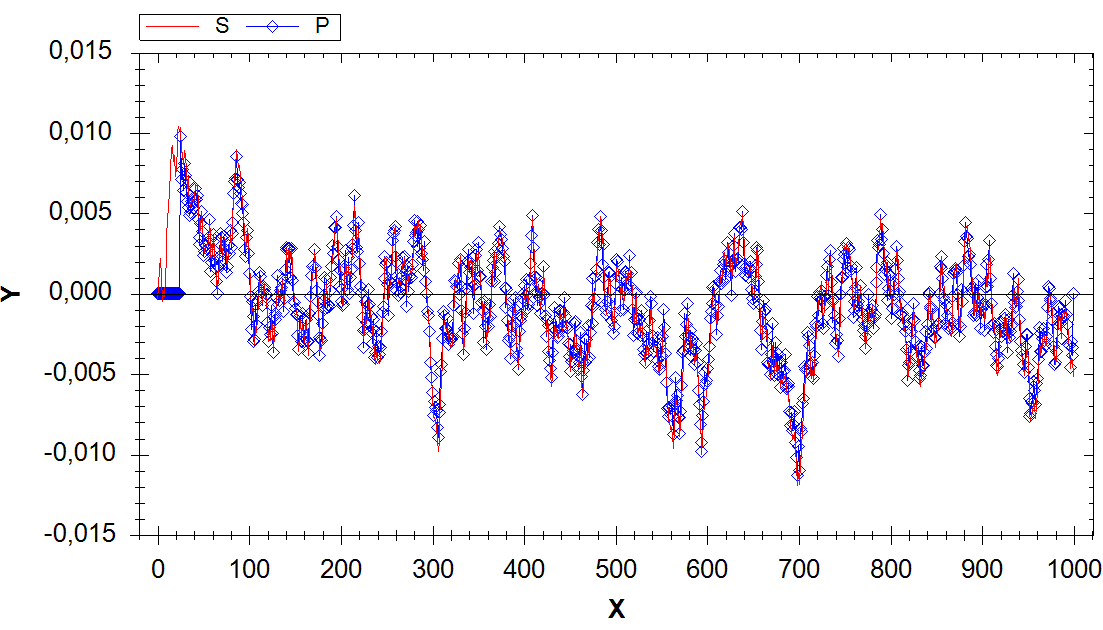


Рис. 6.37.Т=2

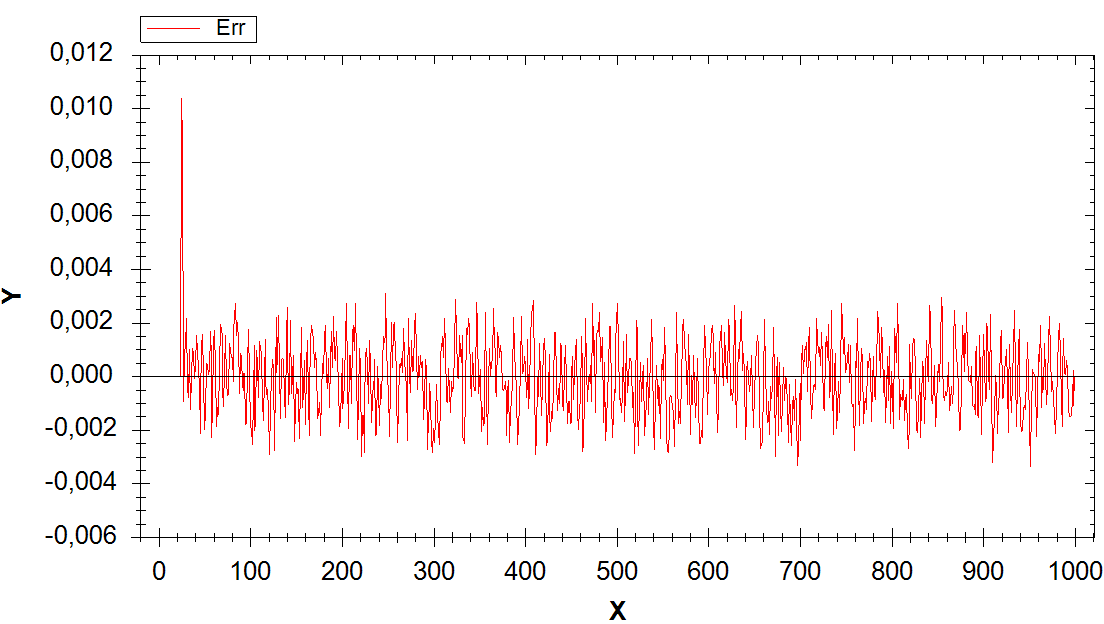


Рис. 6.38.График ошибок

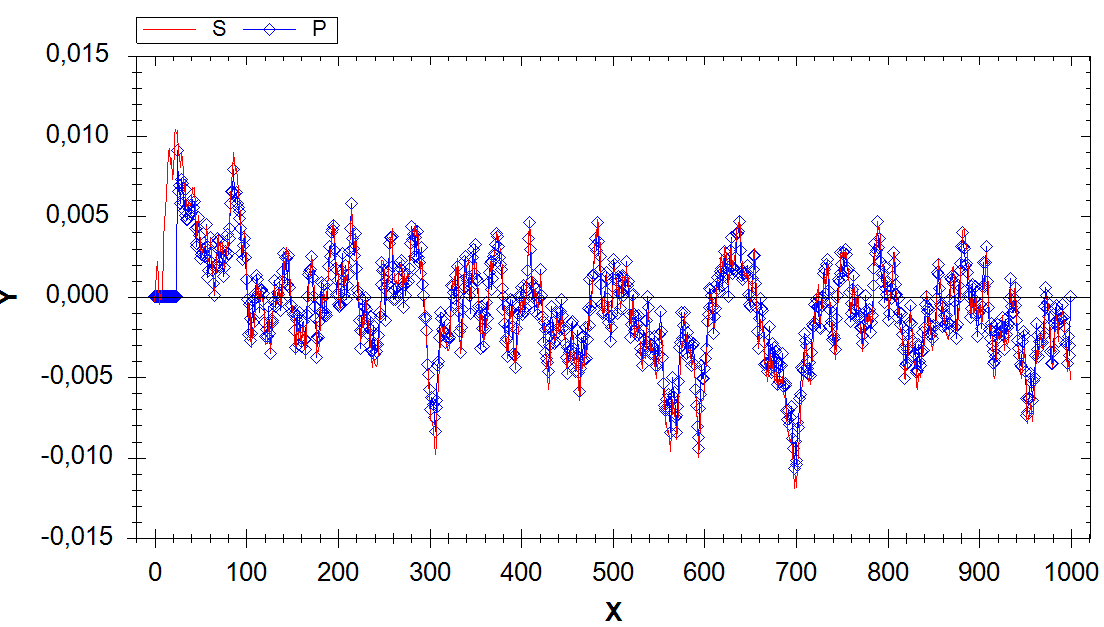


Рис. 6.39.Т=3

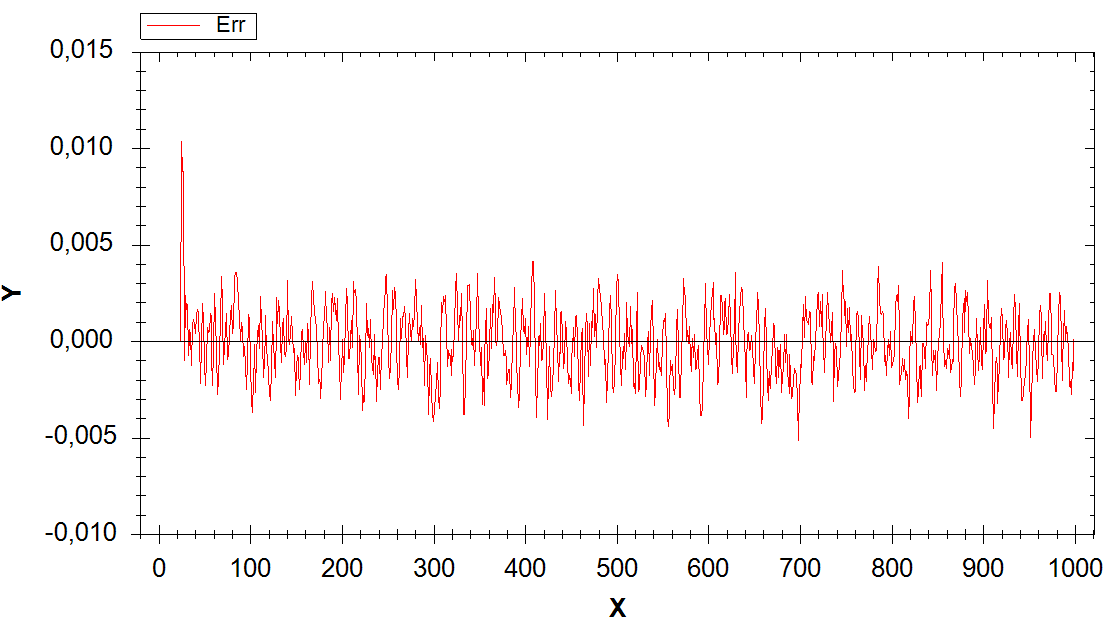


Рис. 6.40.График ошибок

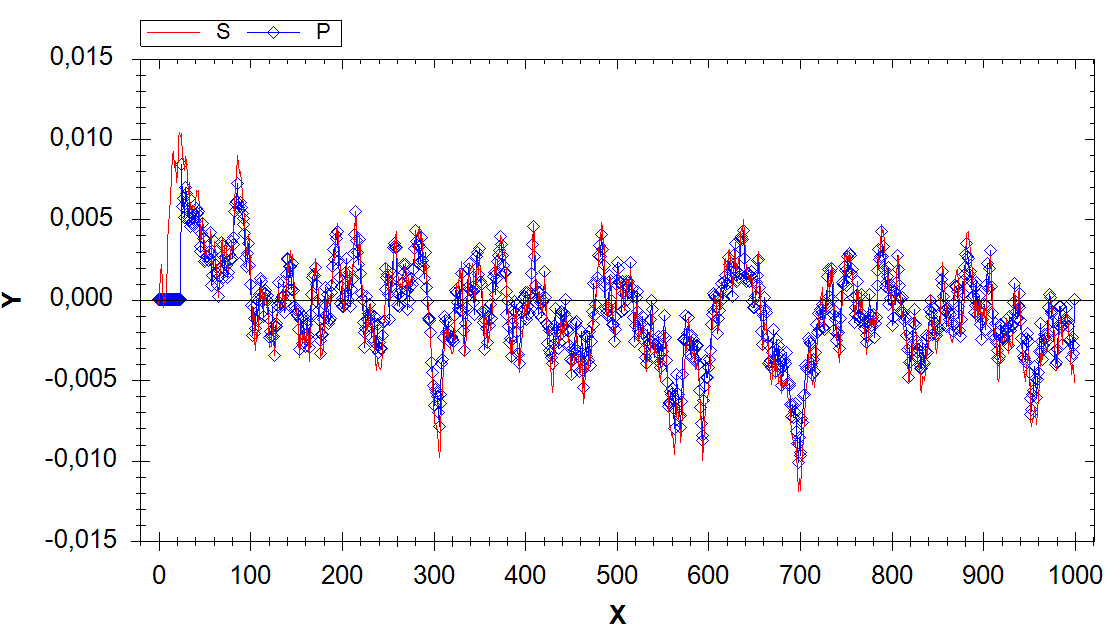


Рис. 6.41.Т=4

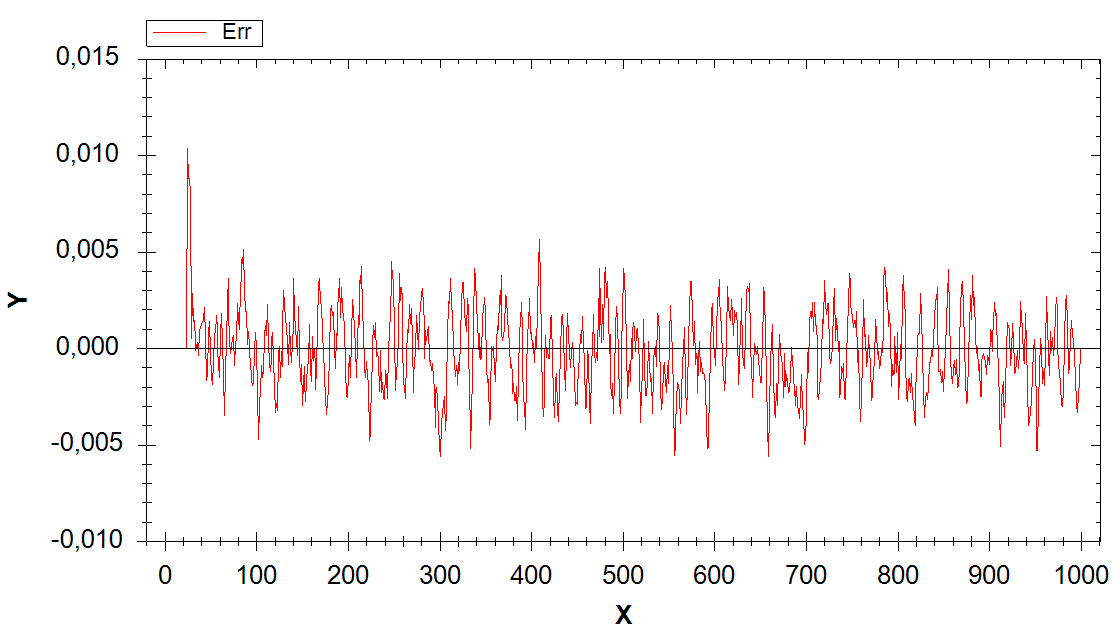


Рис. 6.42.График ошибок

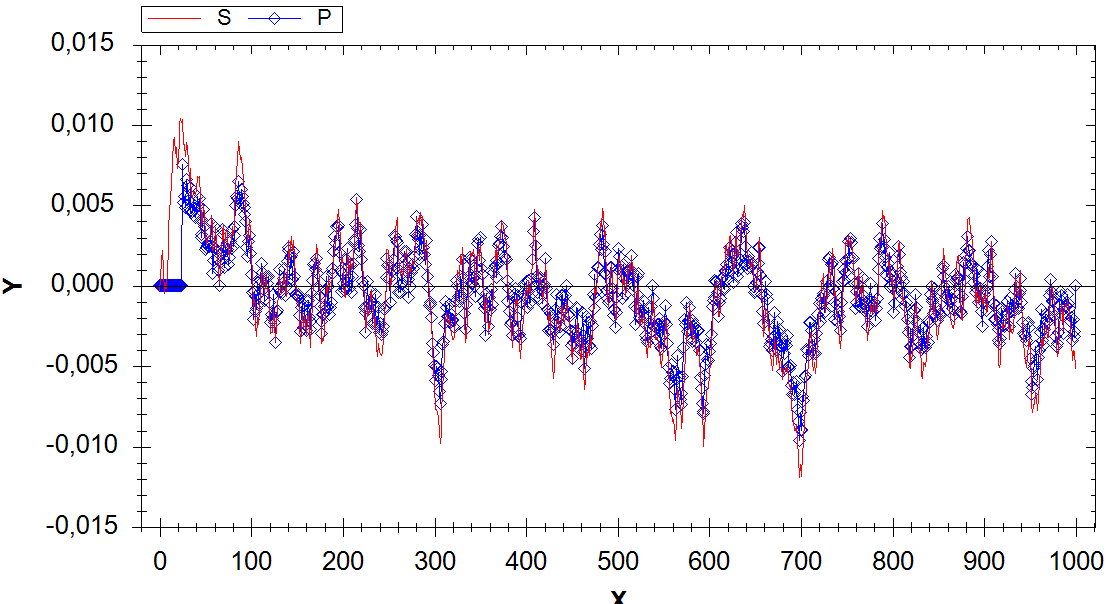


Рис. 6.43.Т=5

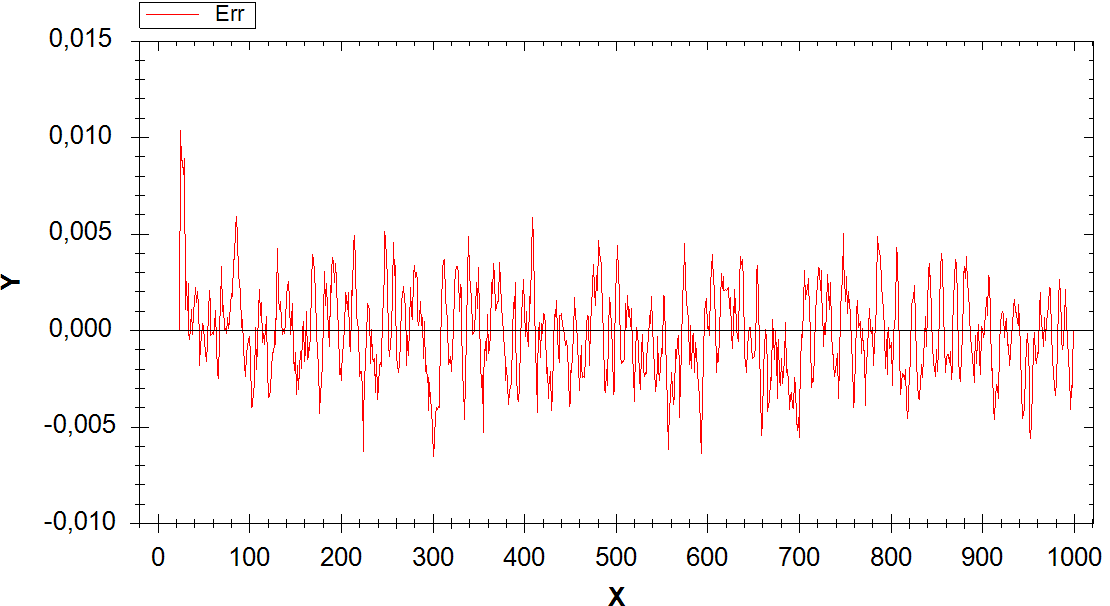


Рис. 6.44.График ошибок

Фильтр LMS

Процесс 1

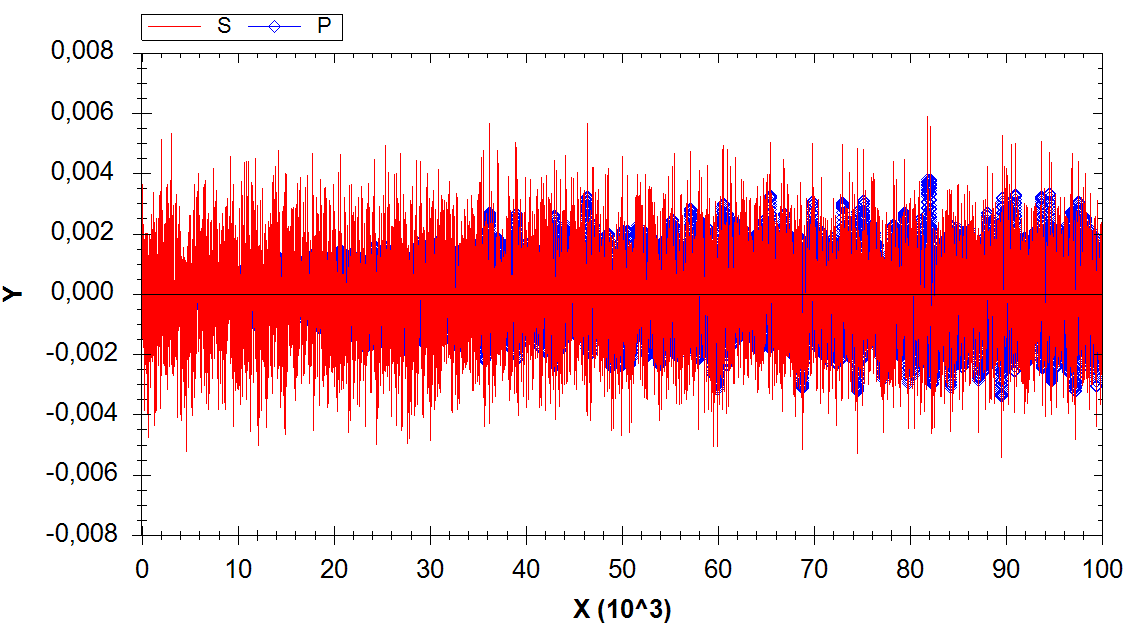


Рис. 6.45.Т=1

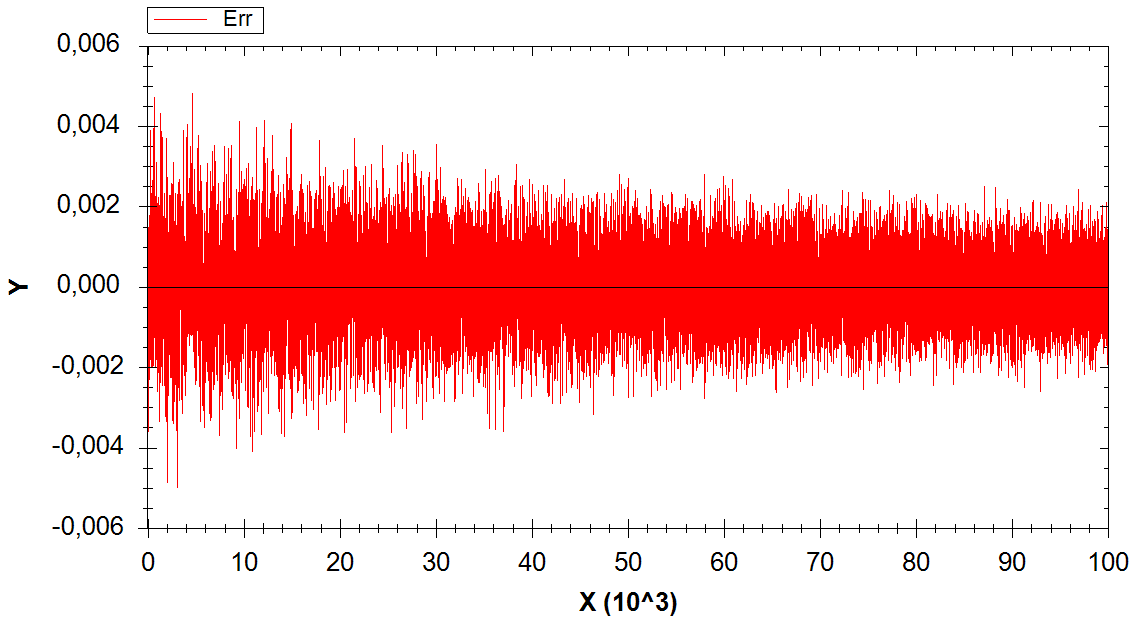


Рис. 6.46.График ошибок

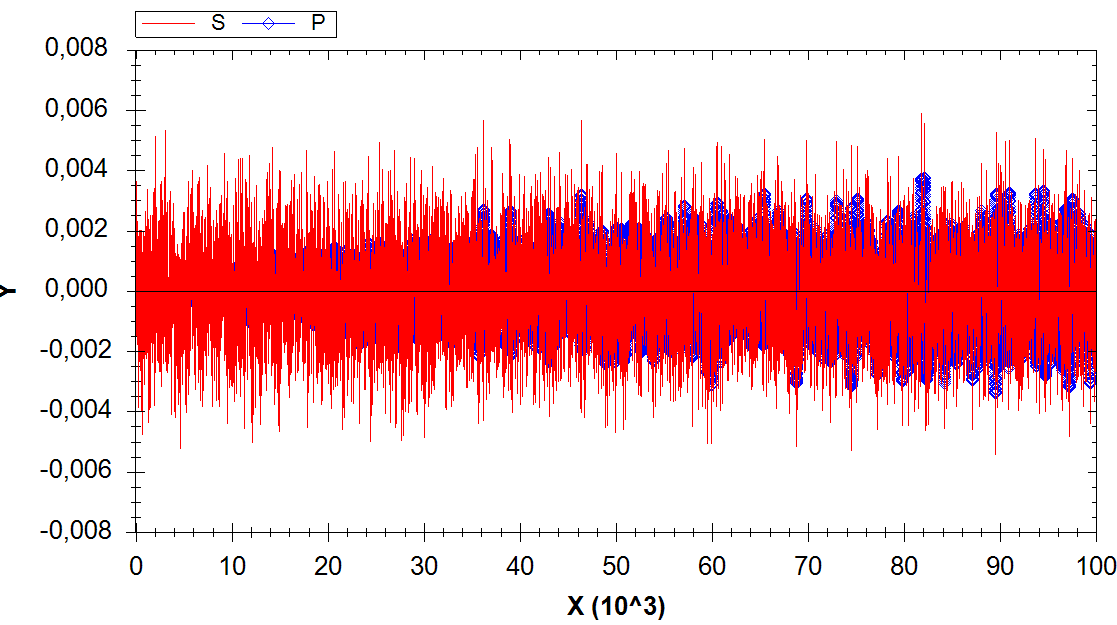


Рис. 6.47.Т=2

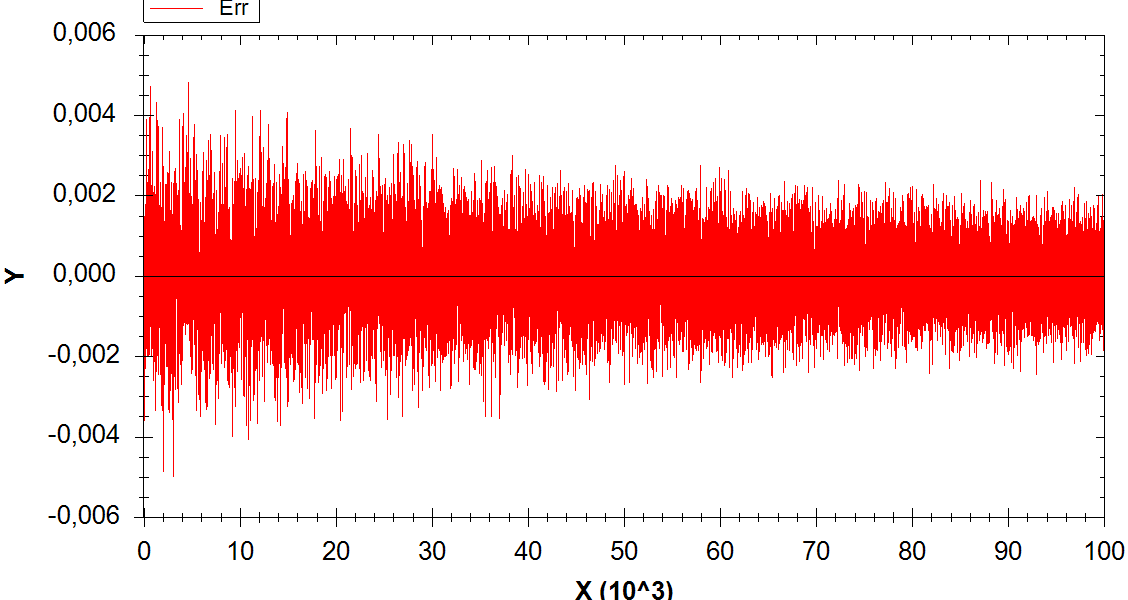


Рис. 6.48.График ошибок

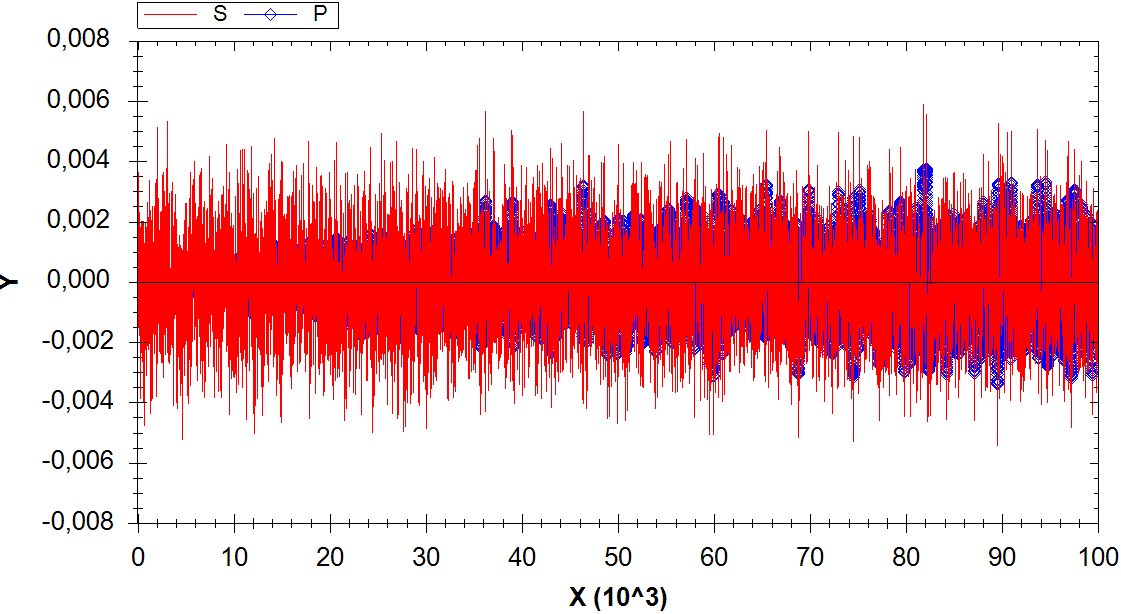


Рис. 6.49.Т=3

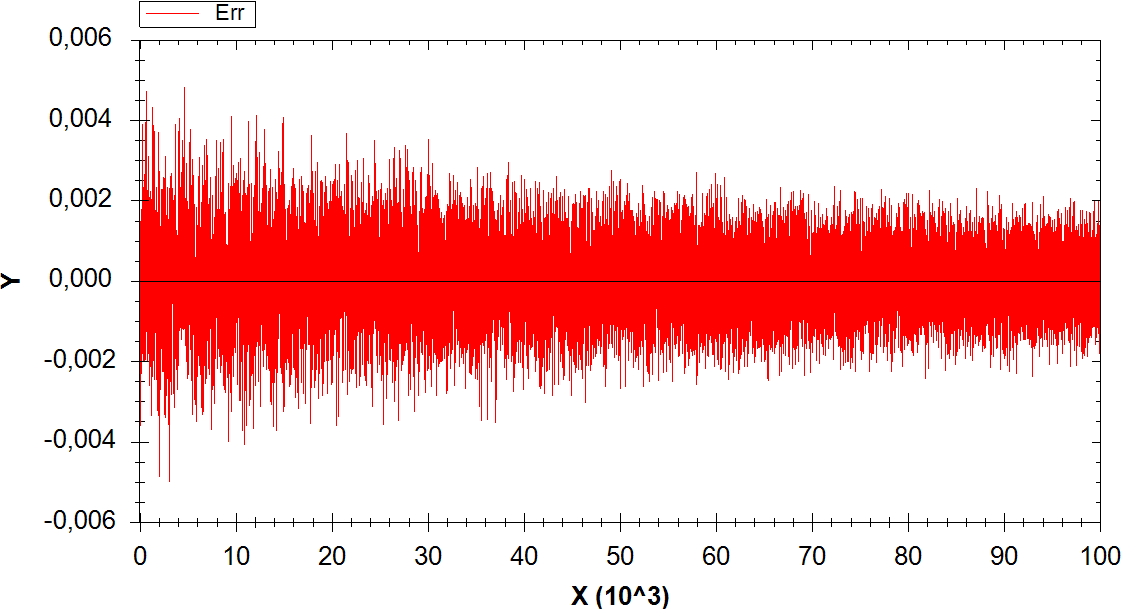


Рис. 6.50.График ошибок

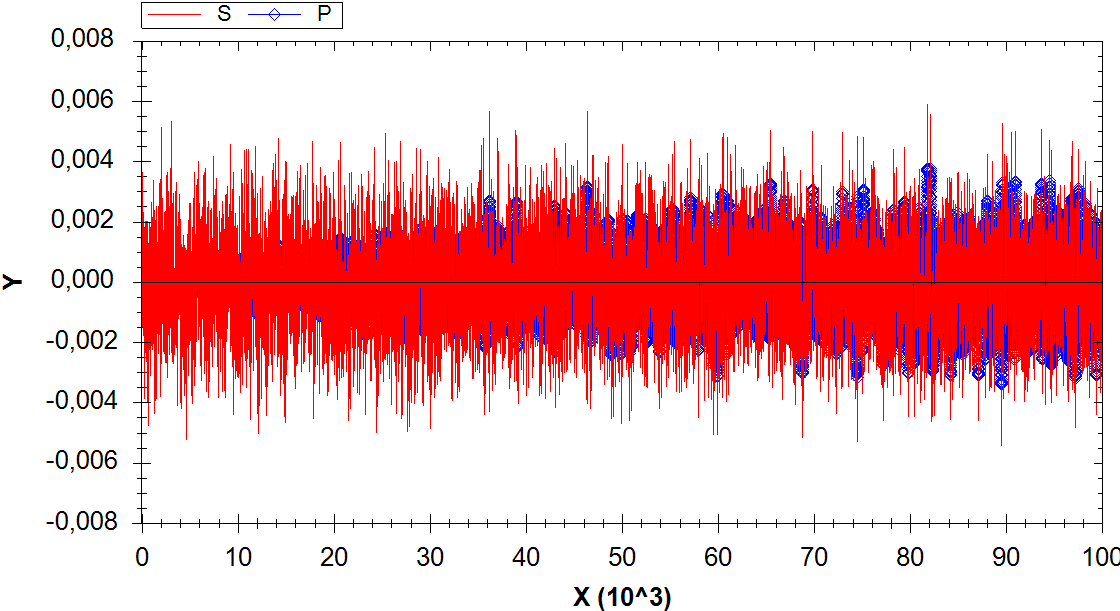


Рис. 6.51.Т=4

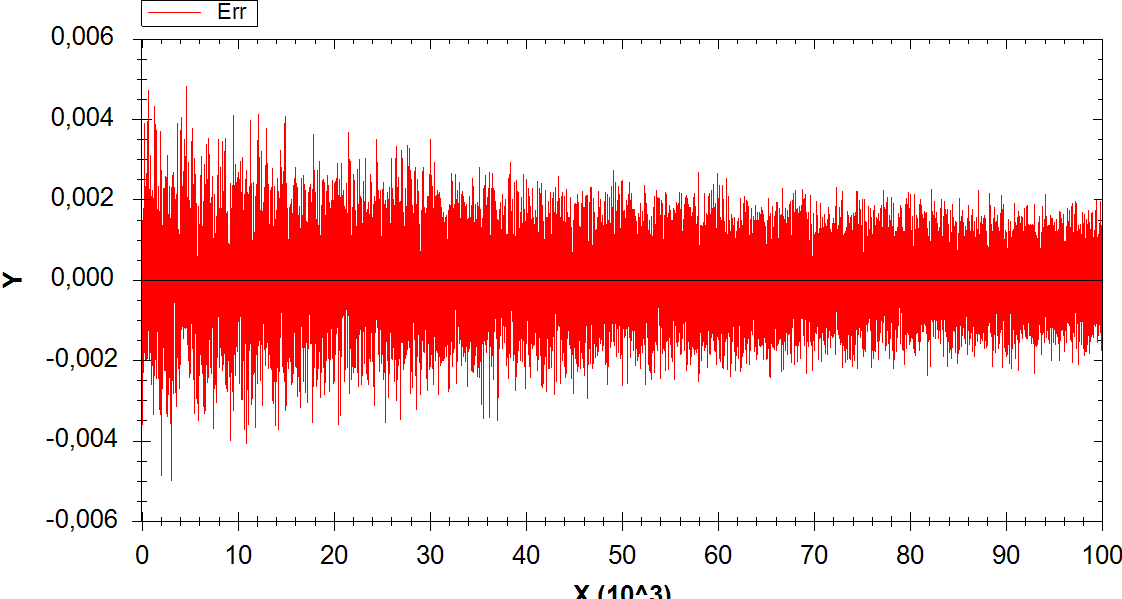


Рис. 6.52.График ошибок

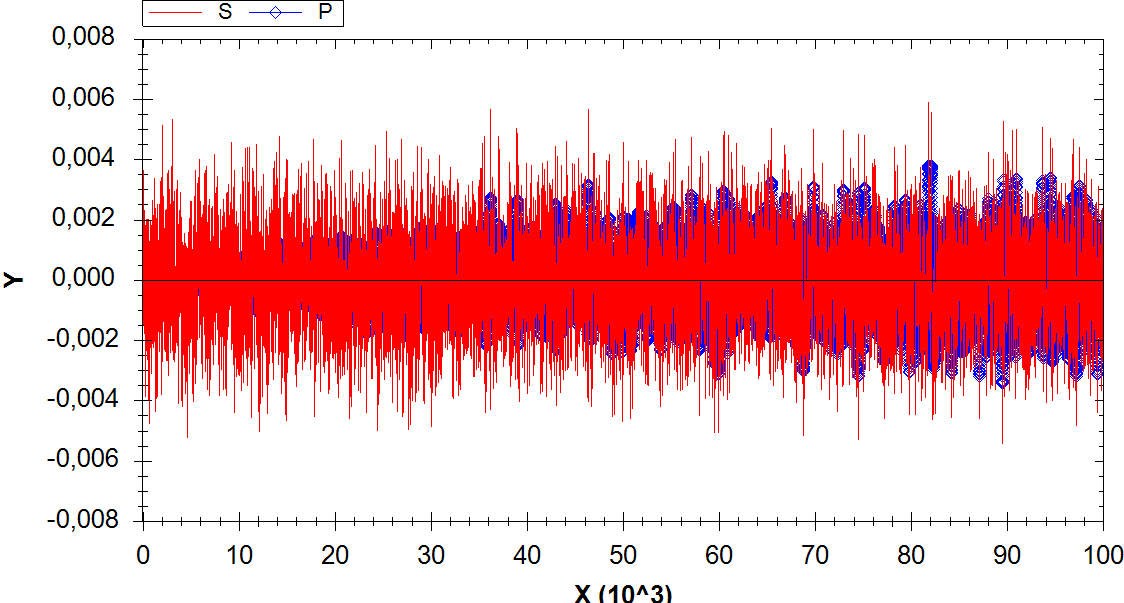


Рис. 6.53.Т=5

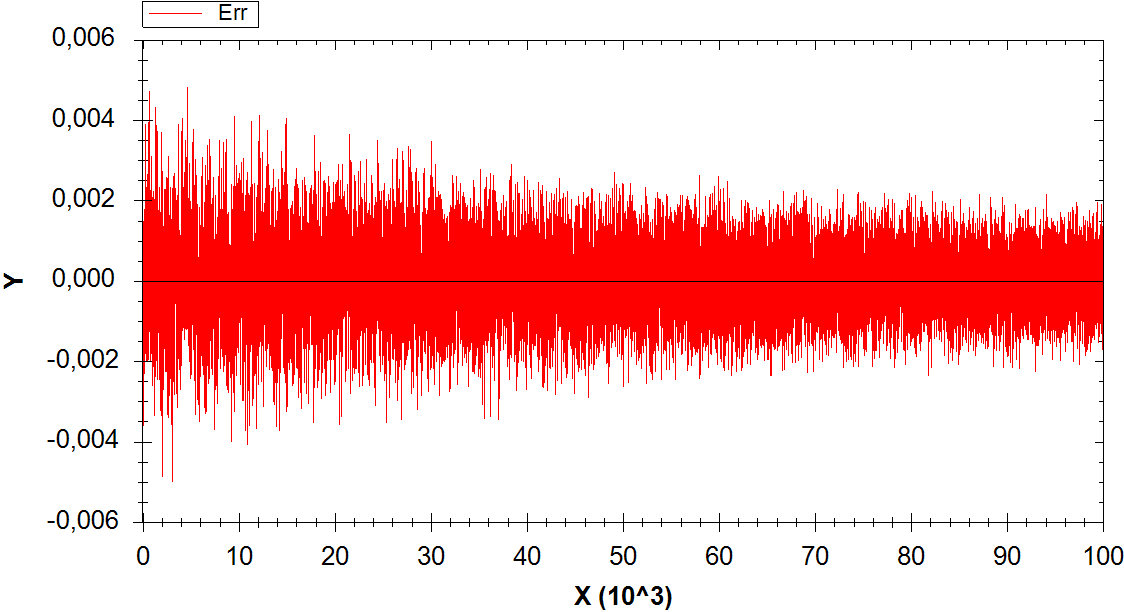


Рис. 6.54.График ошибок

Процесс 2

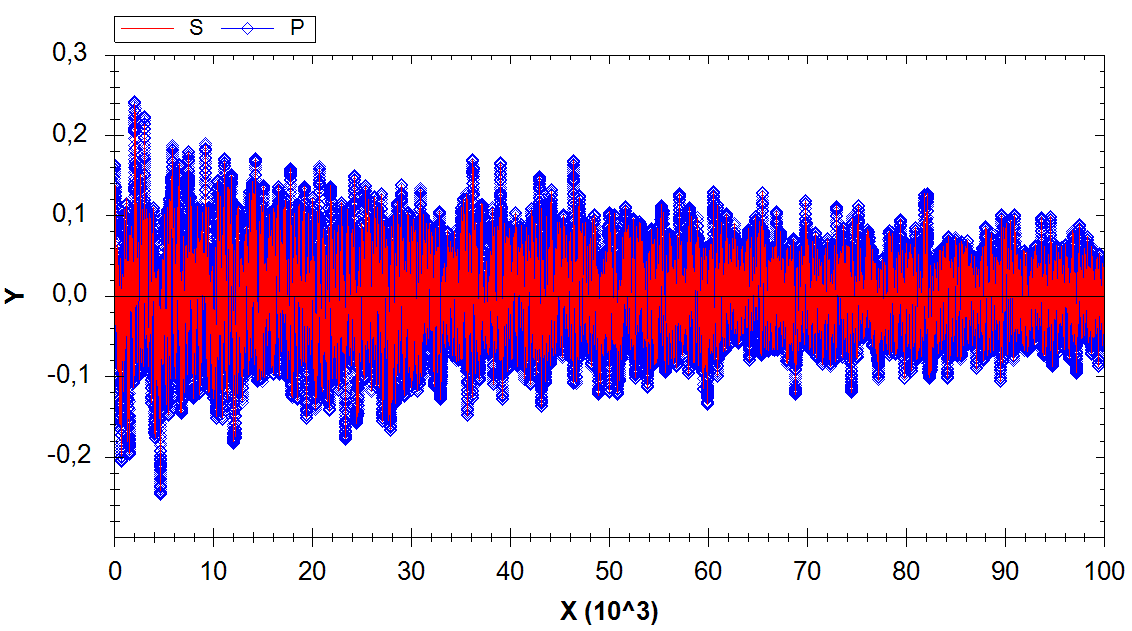


Рис. 6.55.Т=1

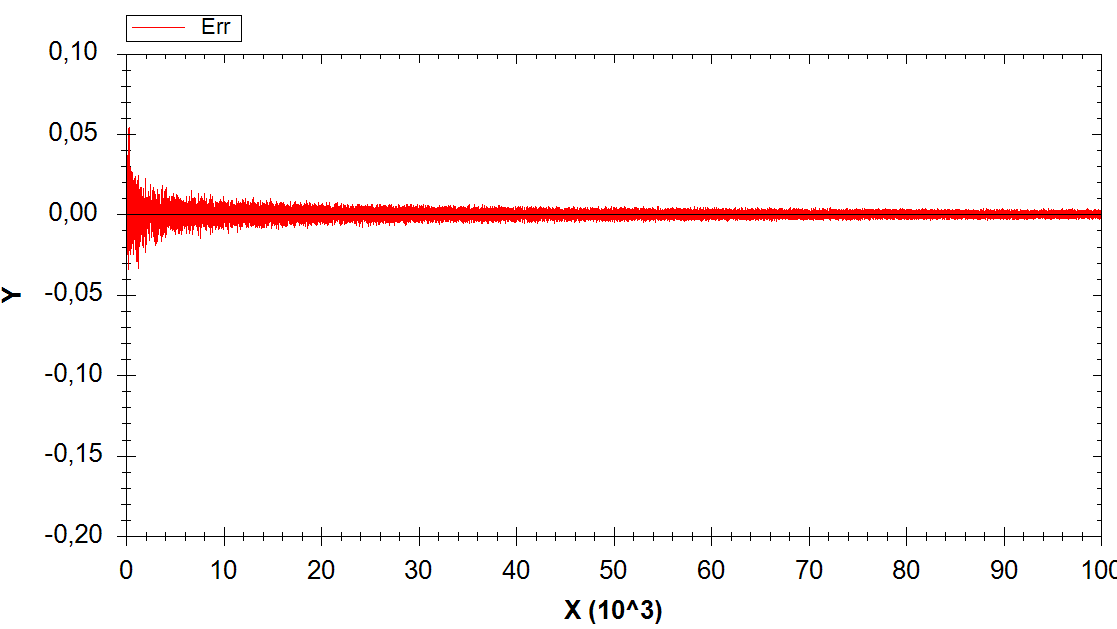


Рис. 6.56.График ошибок

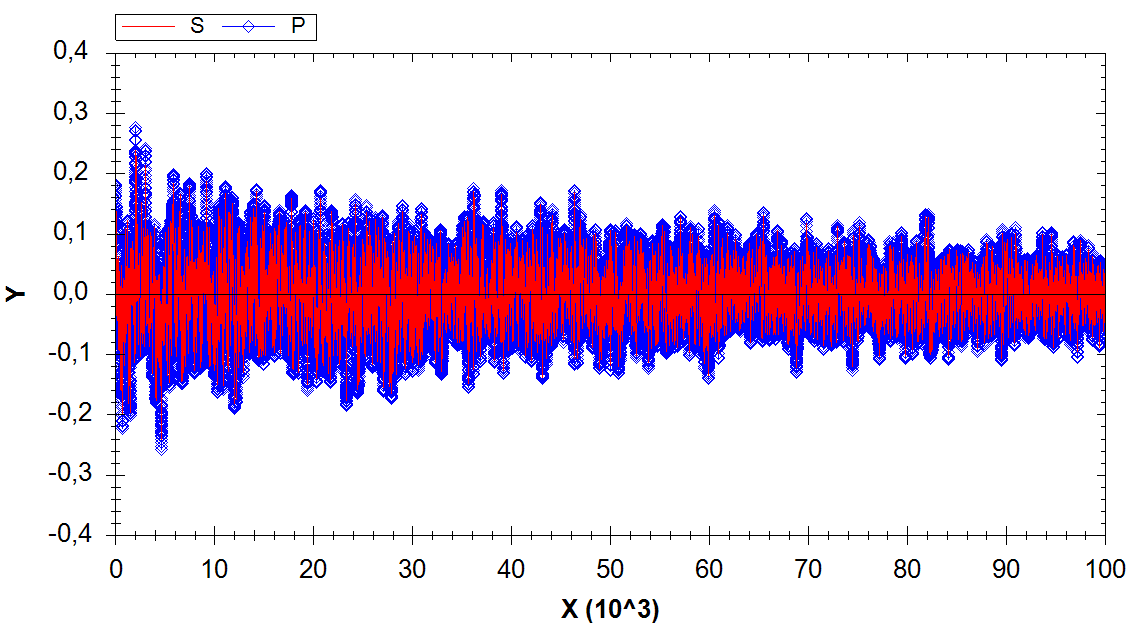


Рис. 6.57.Т=2

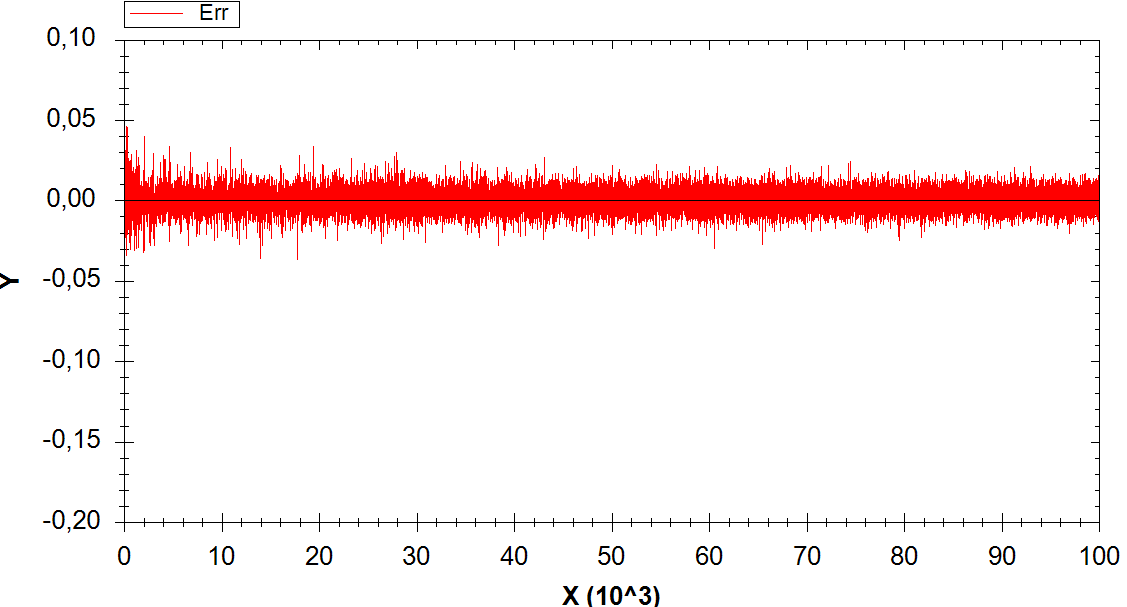


Рис. 6.58.График ошибок

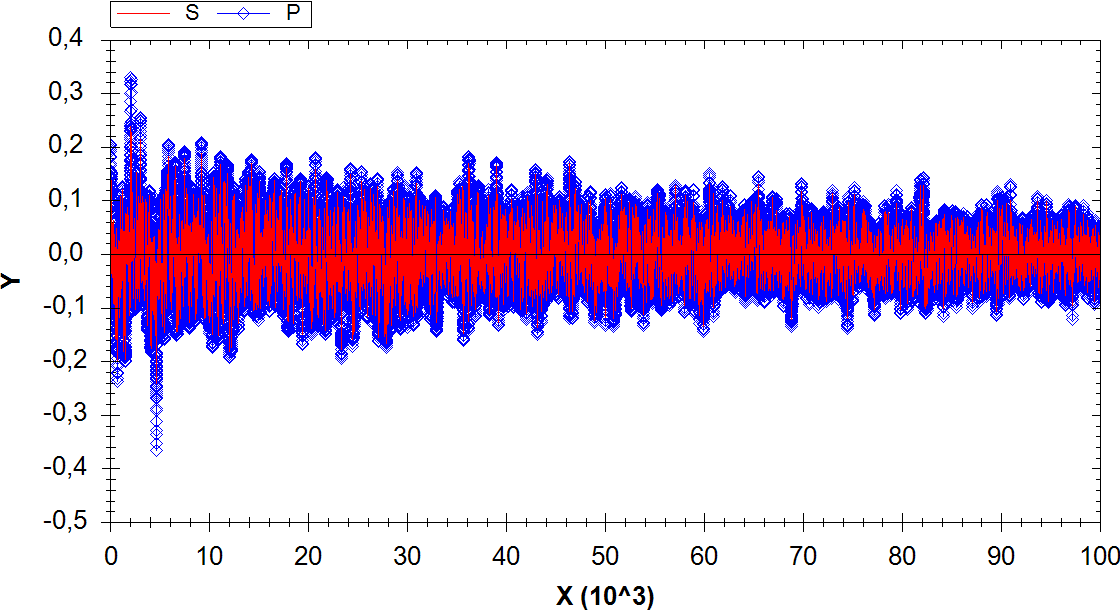


Рис. 6.59.Т=3

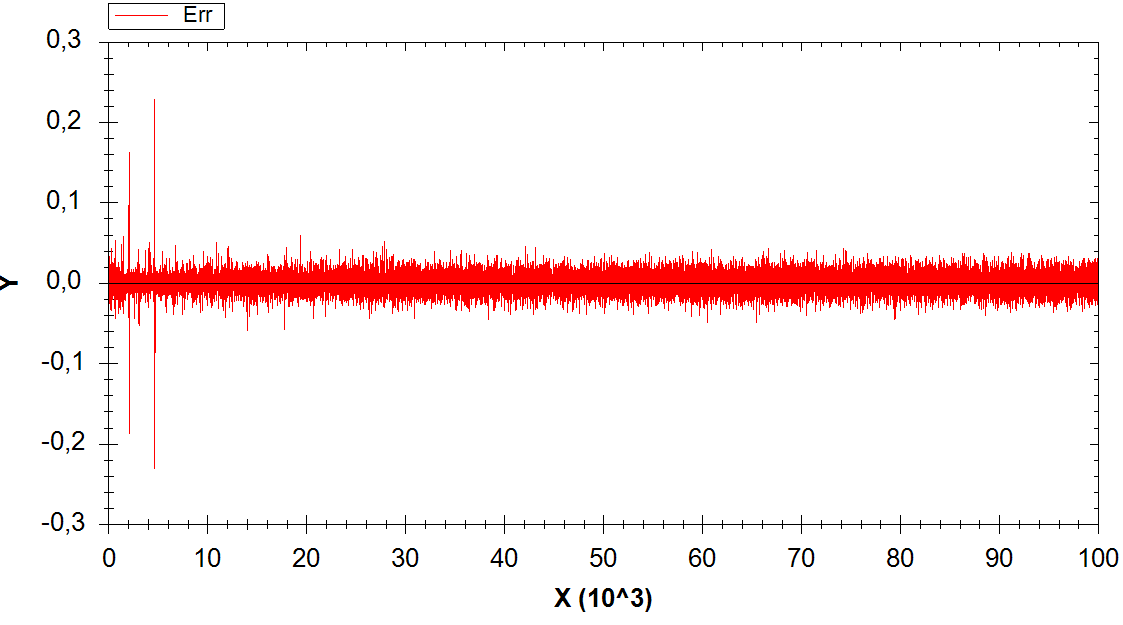


Рис. 6.60.График ошибок

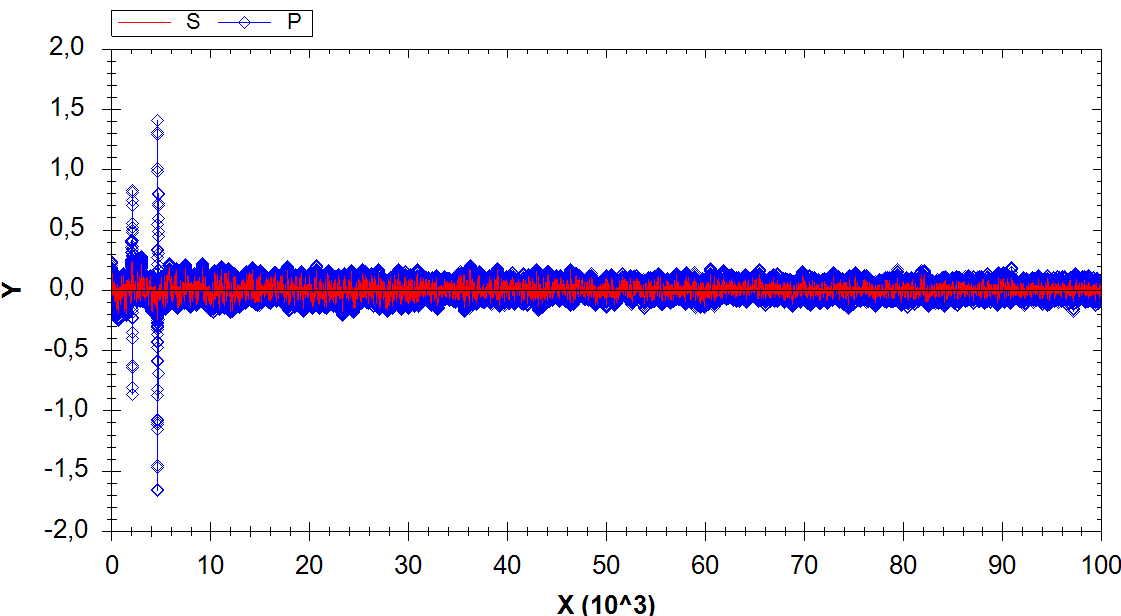


Рис. 6.61.Т=4

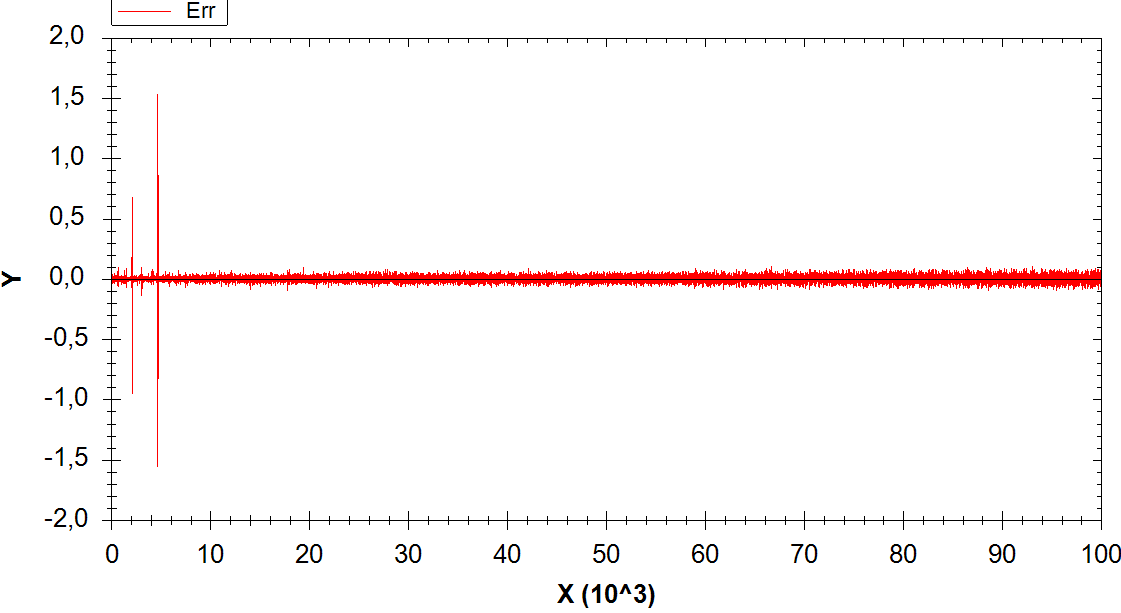


Рис. 6.62.График ошибок

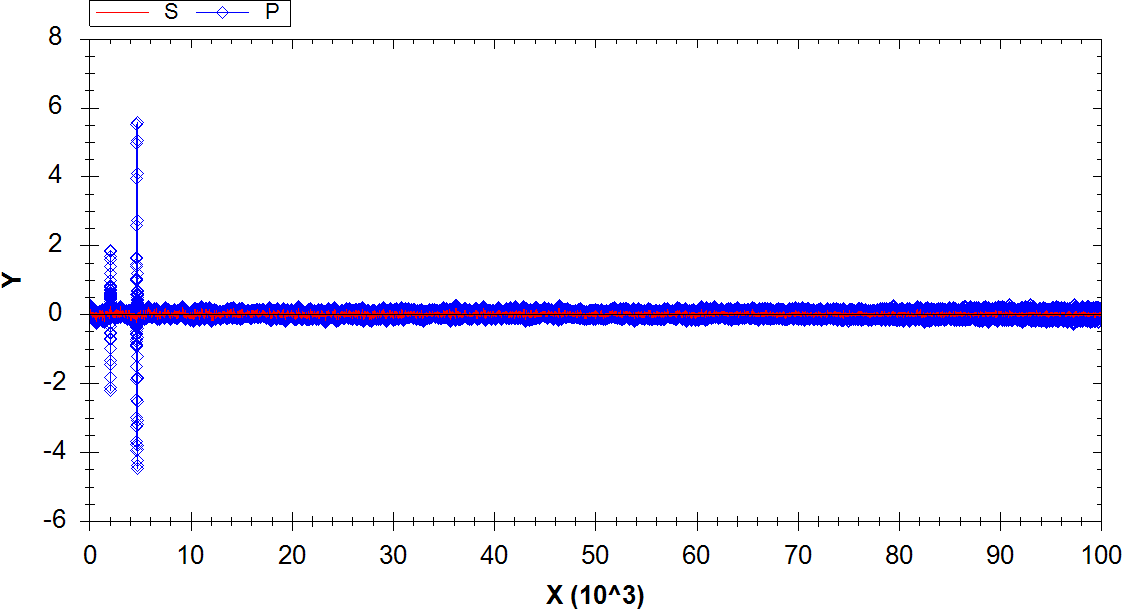


Рис. 6.63.Т=5

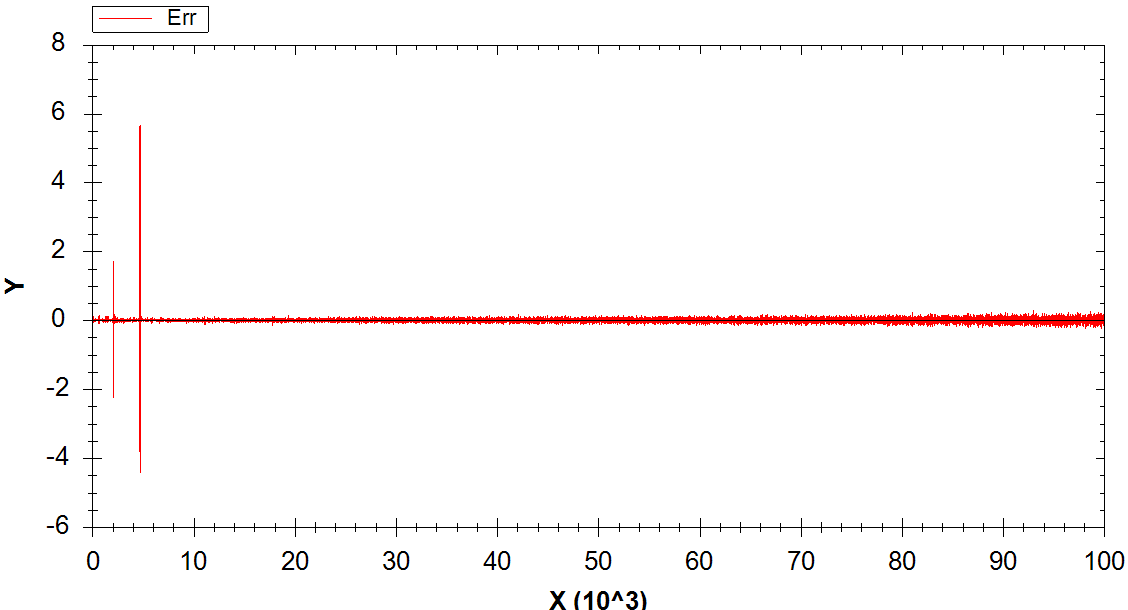


Рис. 6.64.График ошибок

Процесс 3

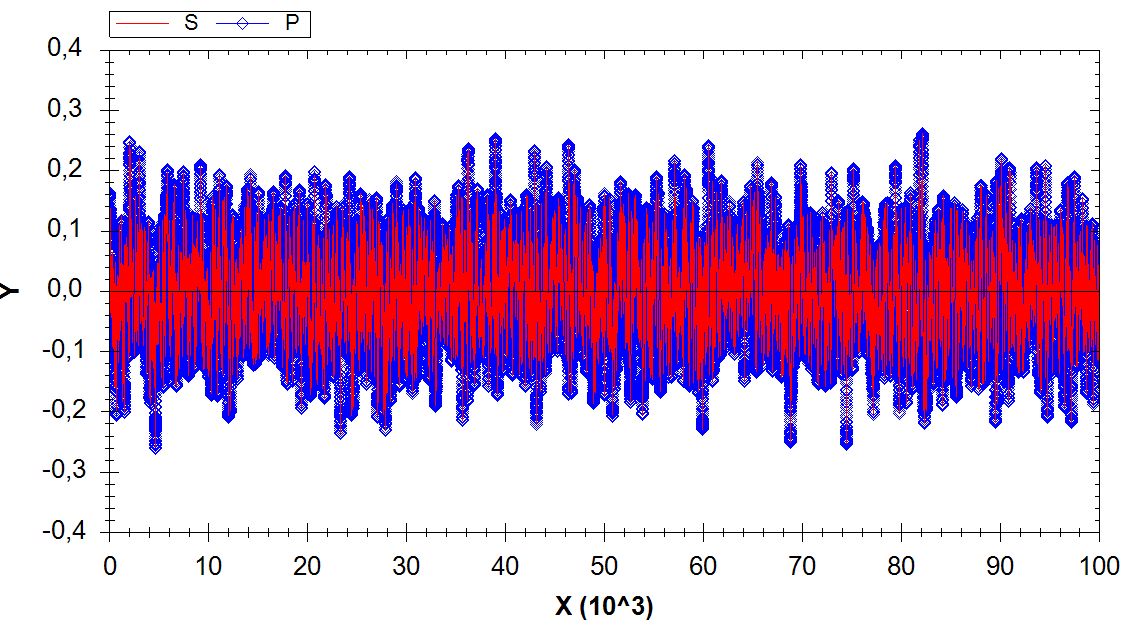


Рис. 6.65.Т=1

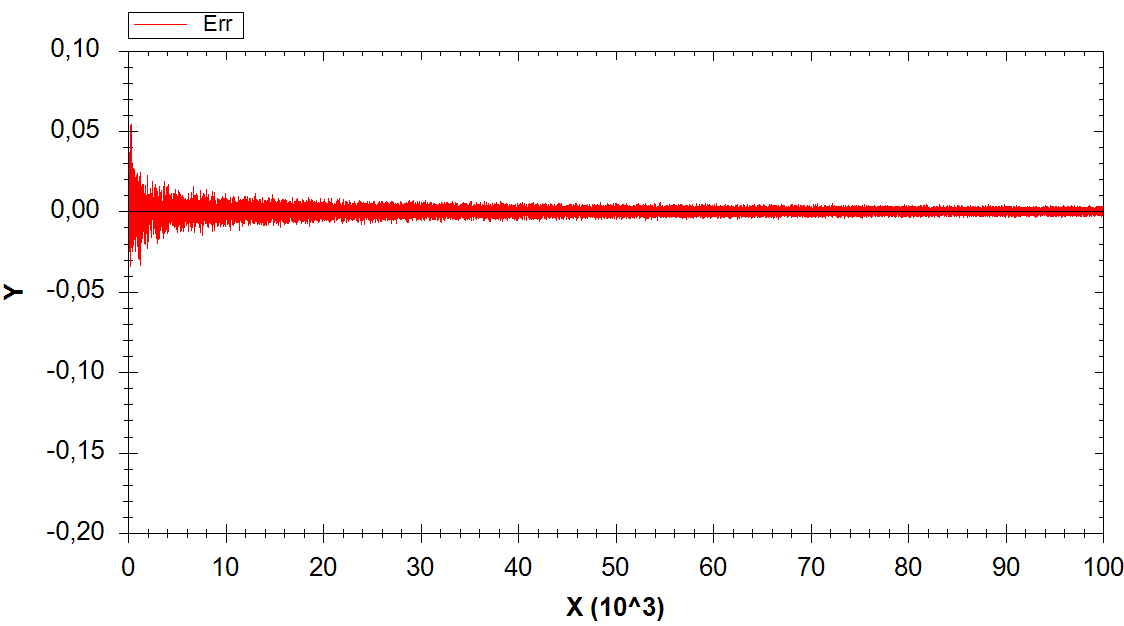


Рис. 6.66.График ошибок

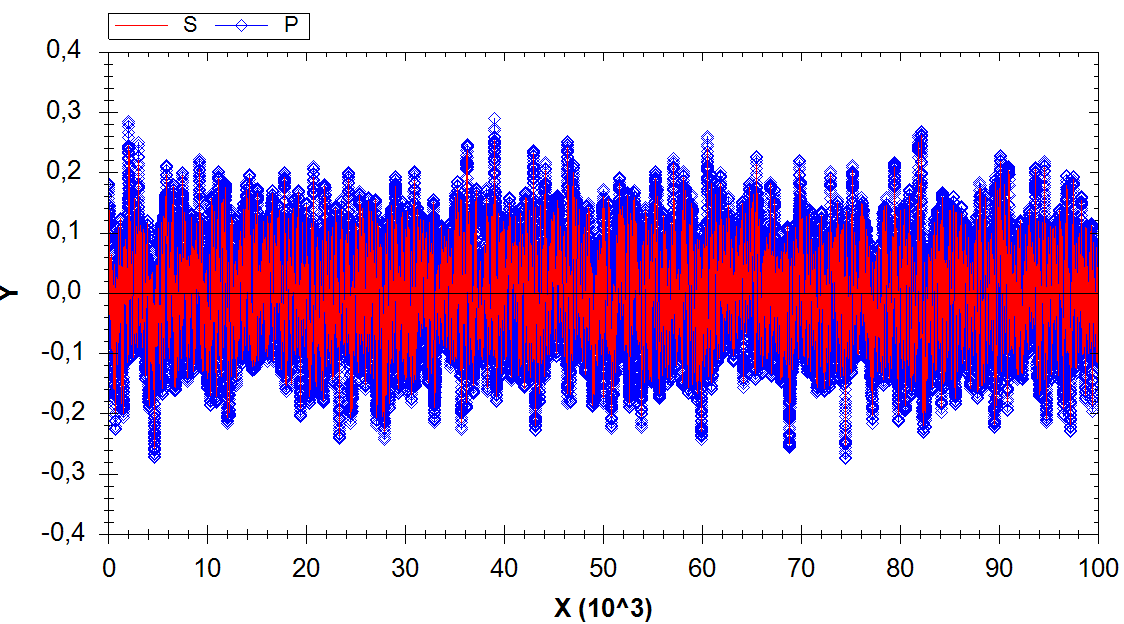


Рис. 6.67.Т=2

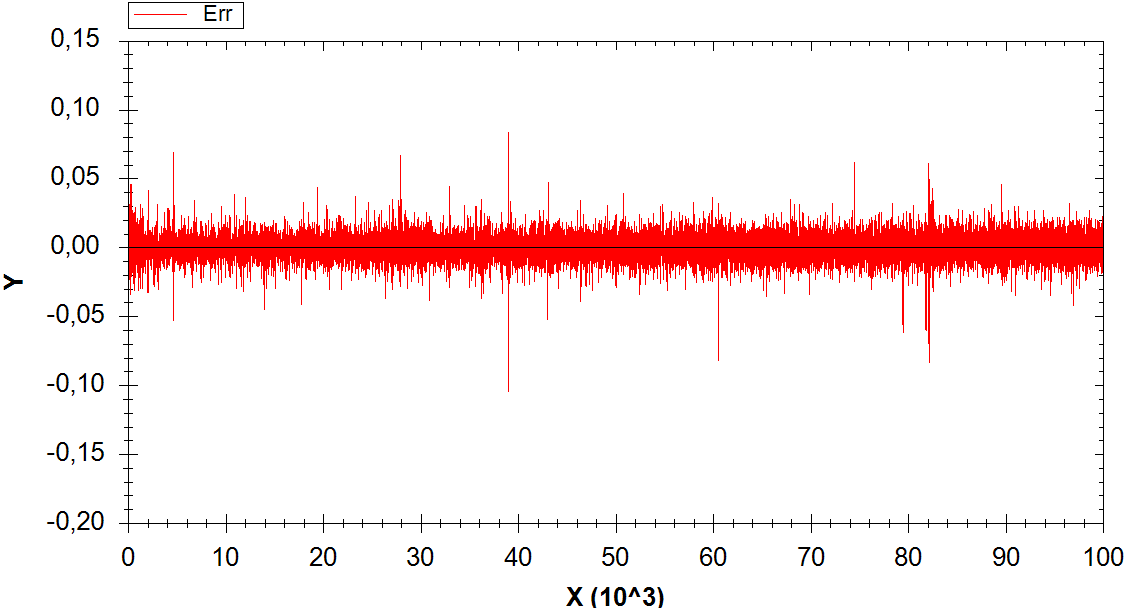


Рис. 6.68.График ошибок

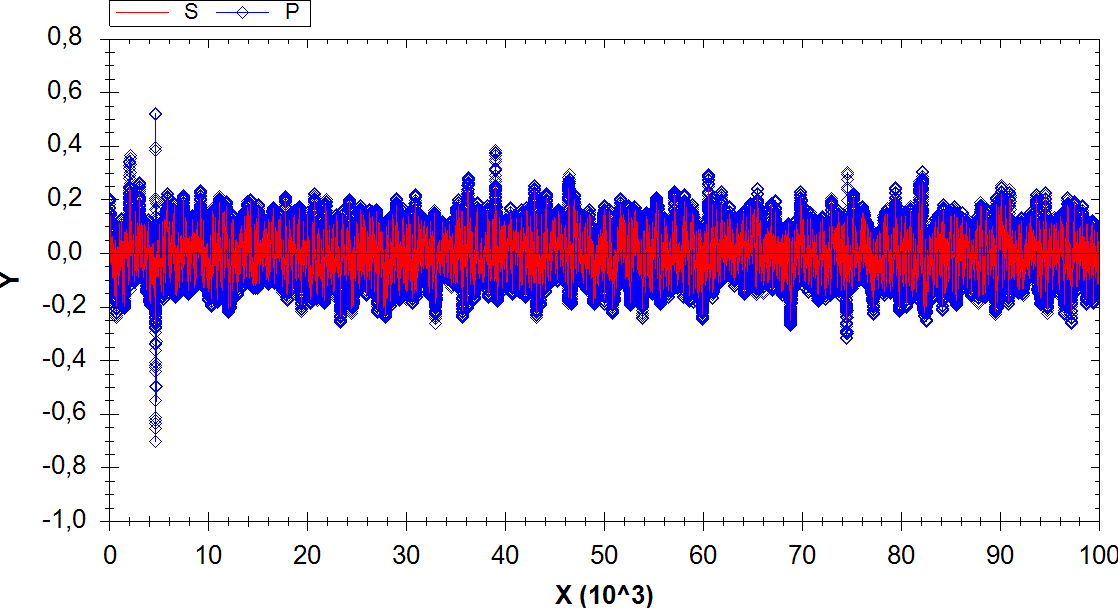


Рис. 6.69.Т=3

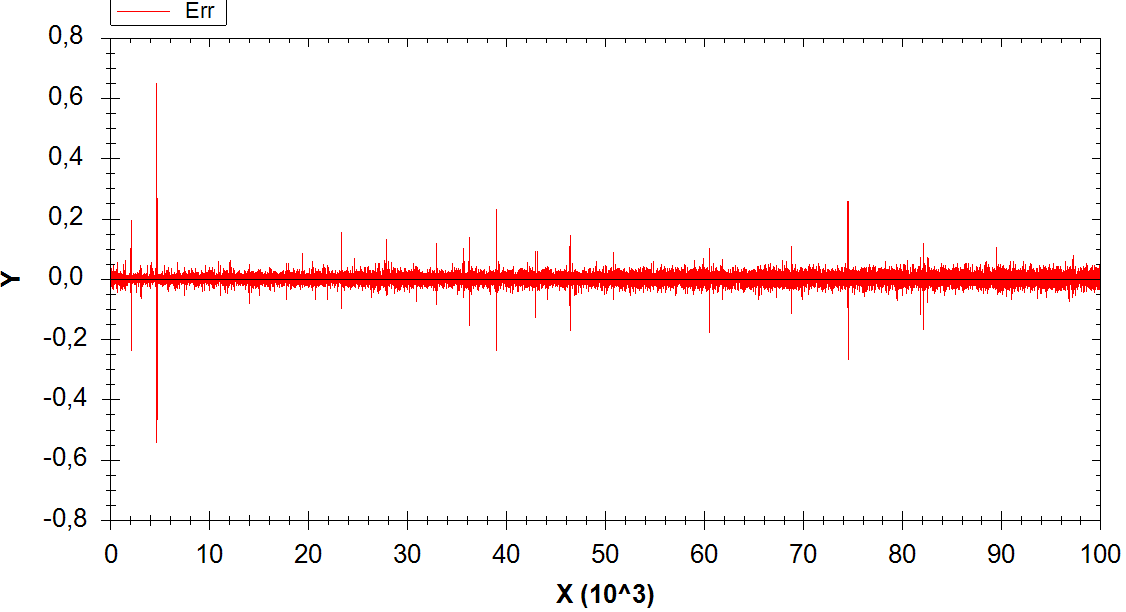


Рис. 6.70.График ошибок

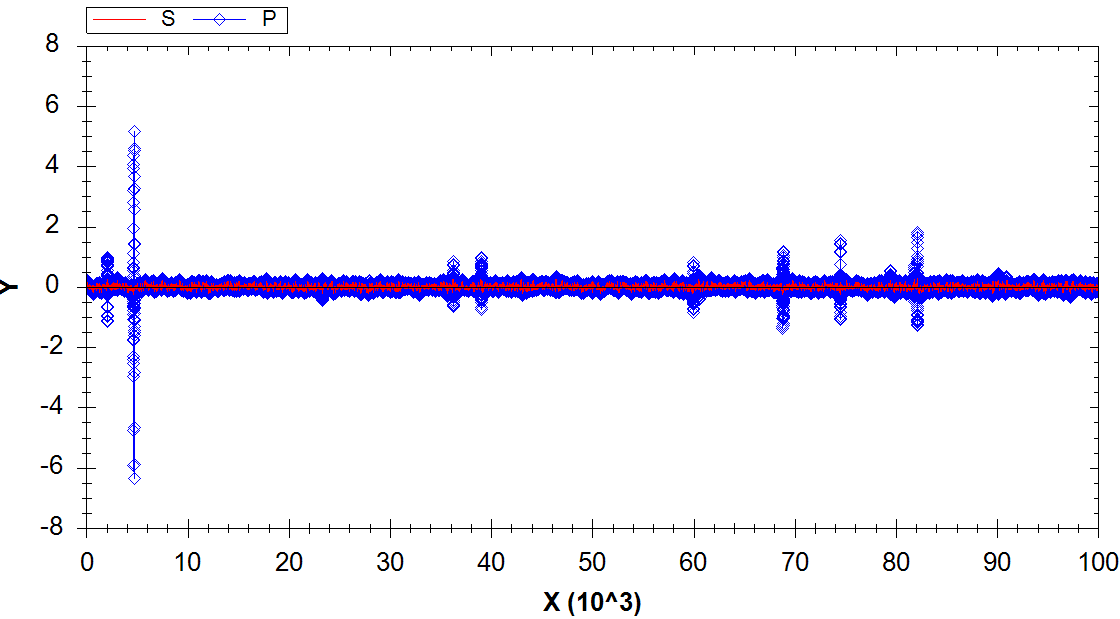


Рис. 6.71.Т=4

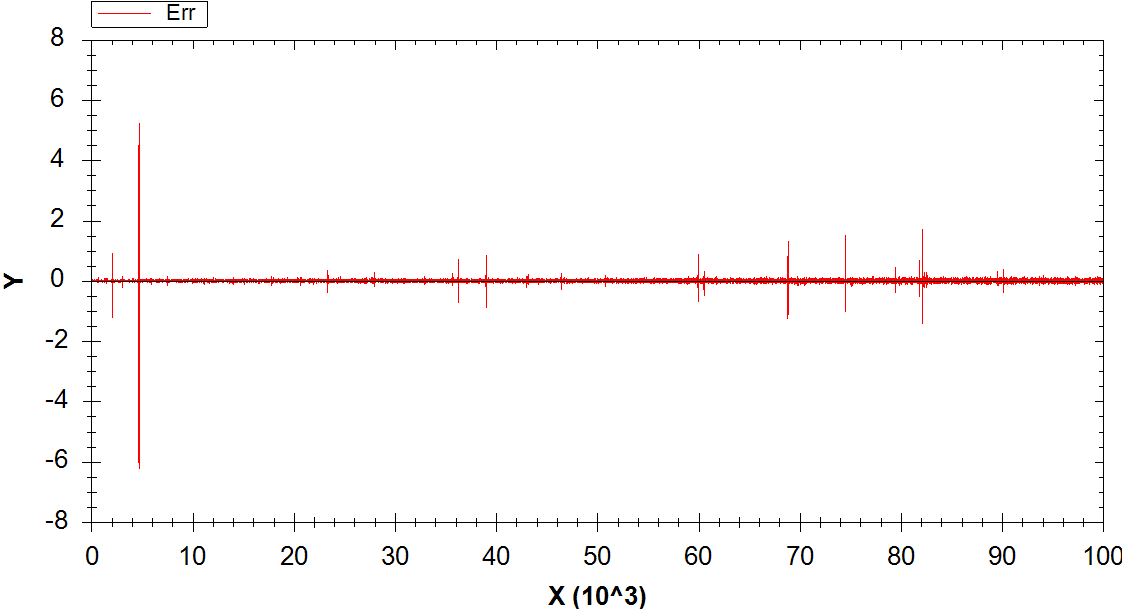


Рис. 6.72.График ошибок

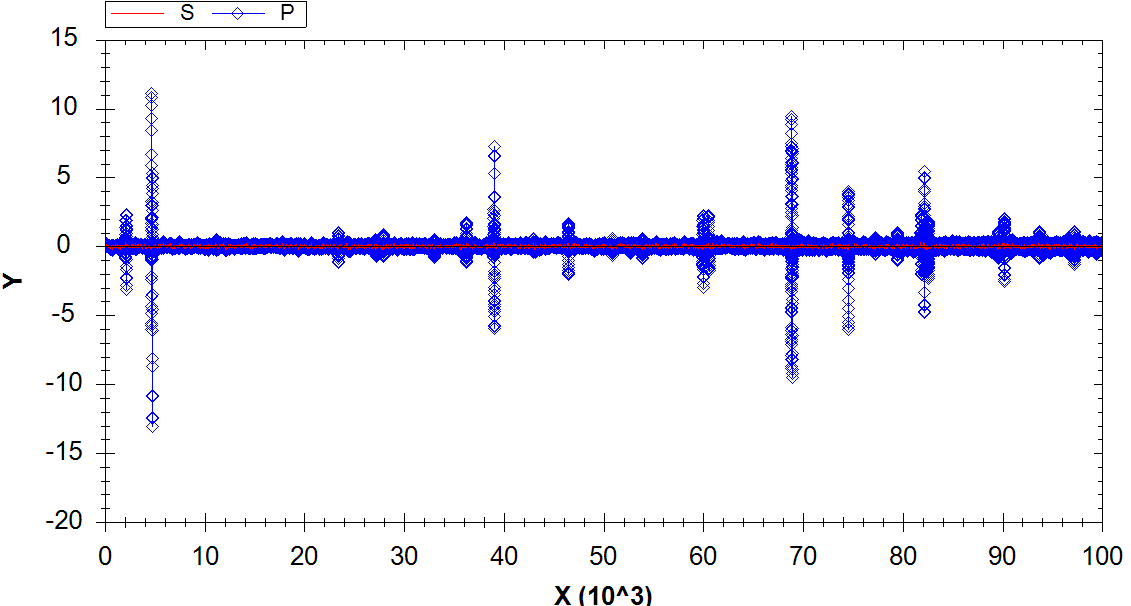


Рис. 6.73.Т=5

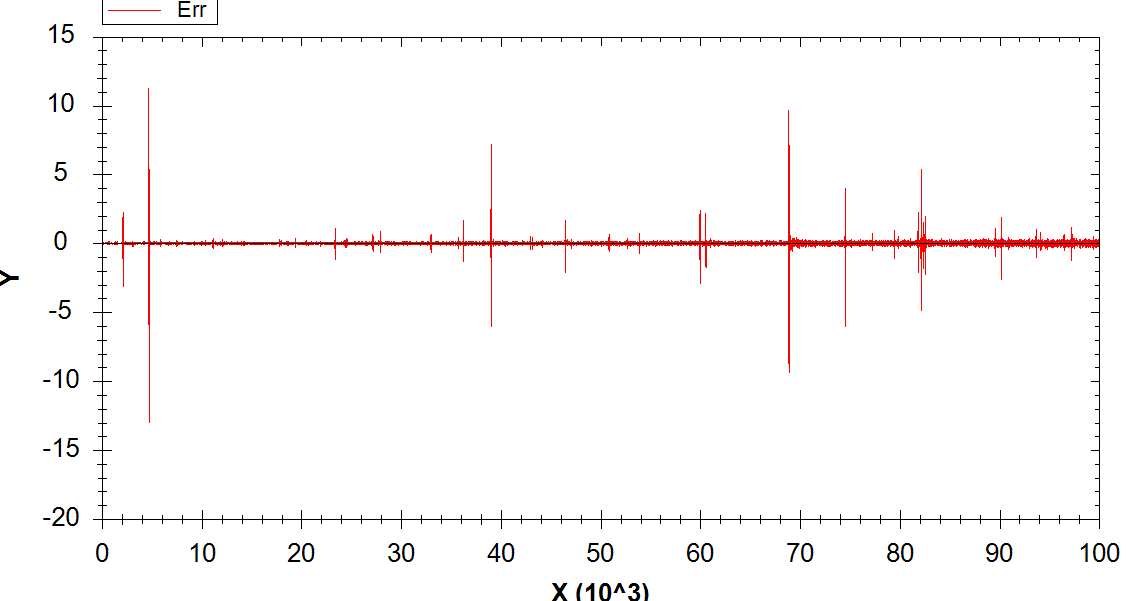


Рис. 6.74.График ошибок

Процесс 4

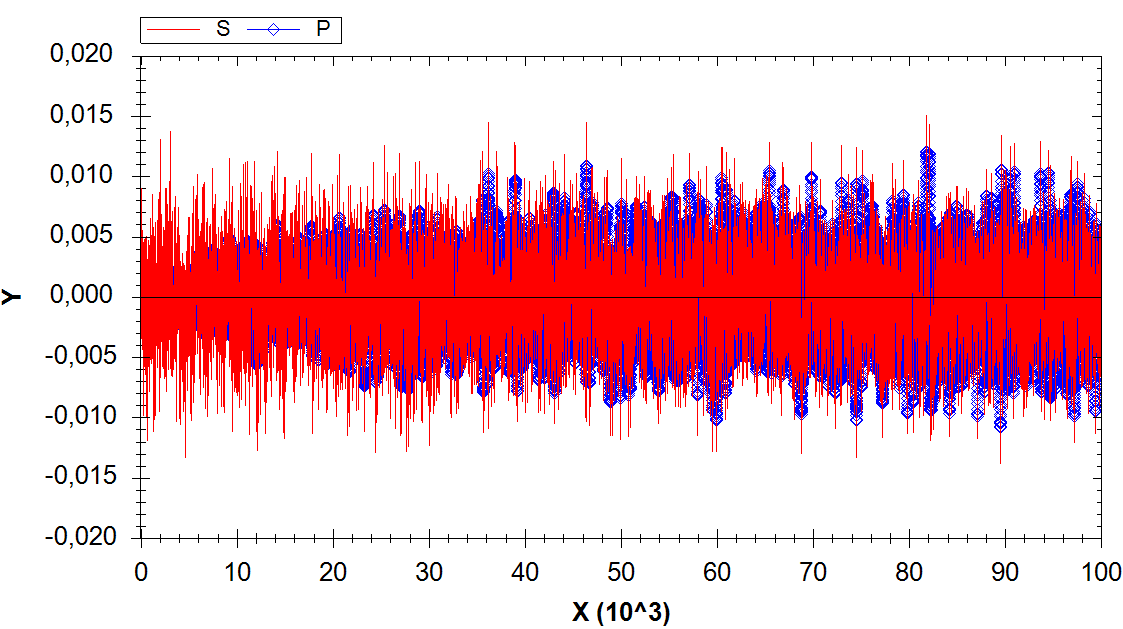


Рис. 6.75.Т=1

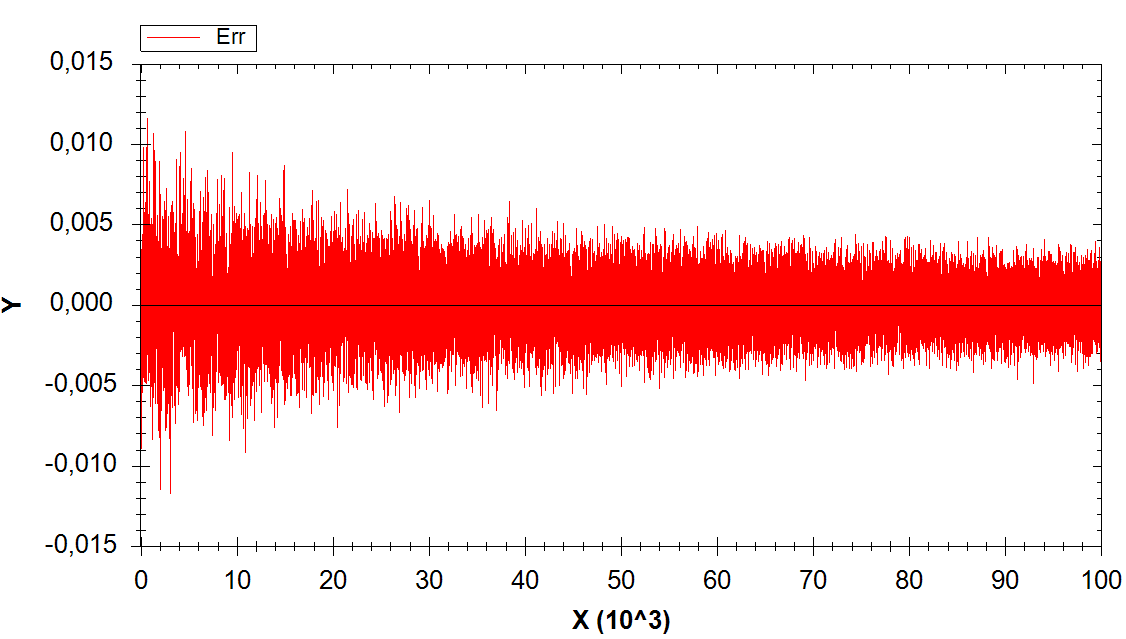


Рис. 6.76.График ошибок

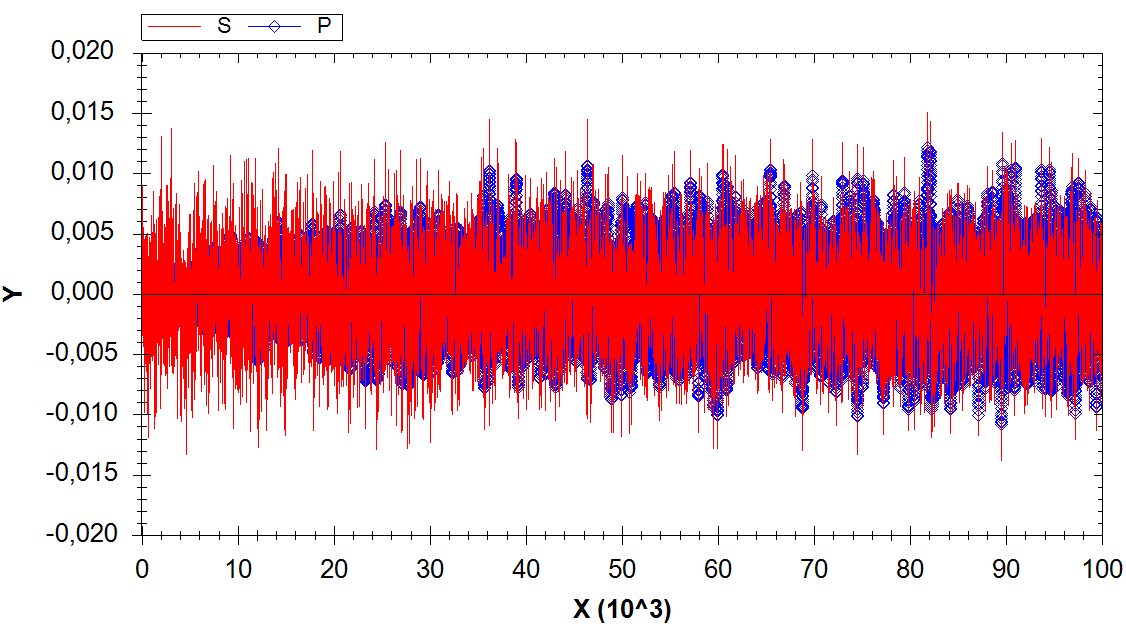


Рис. 6.77.Т=2

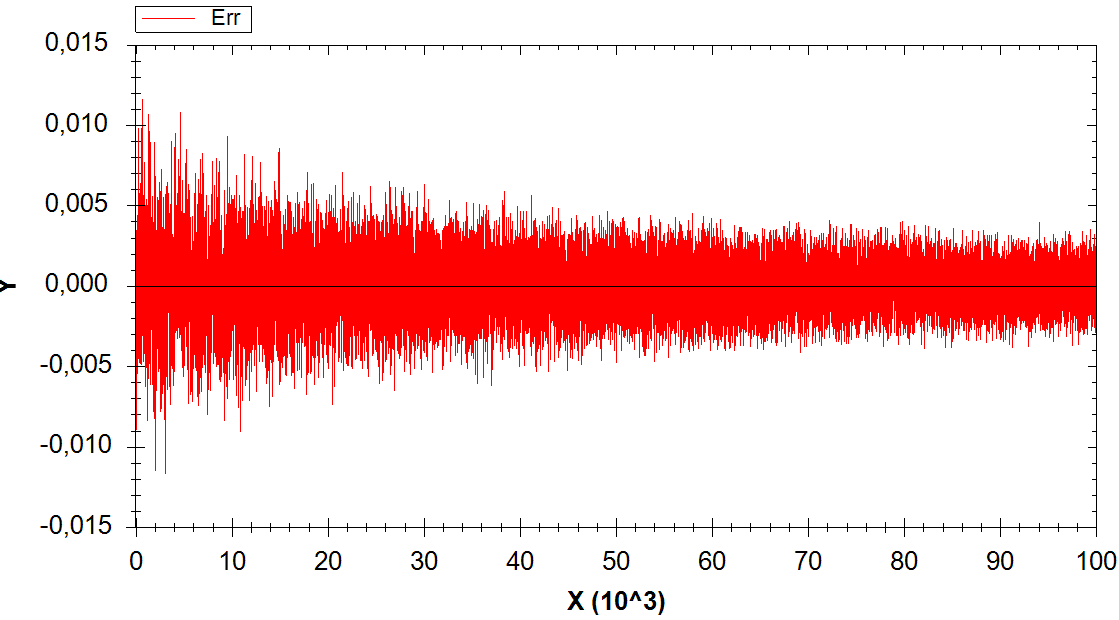


Рис. 6.78.График ошибок

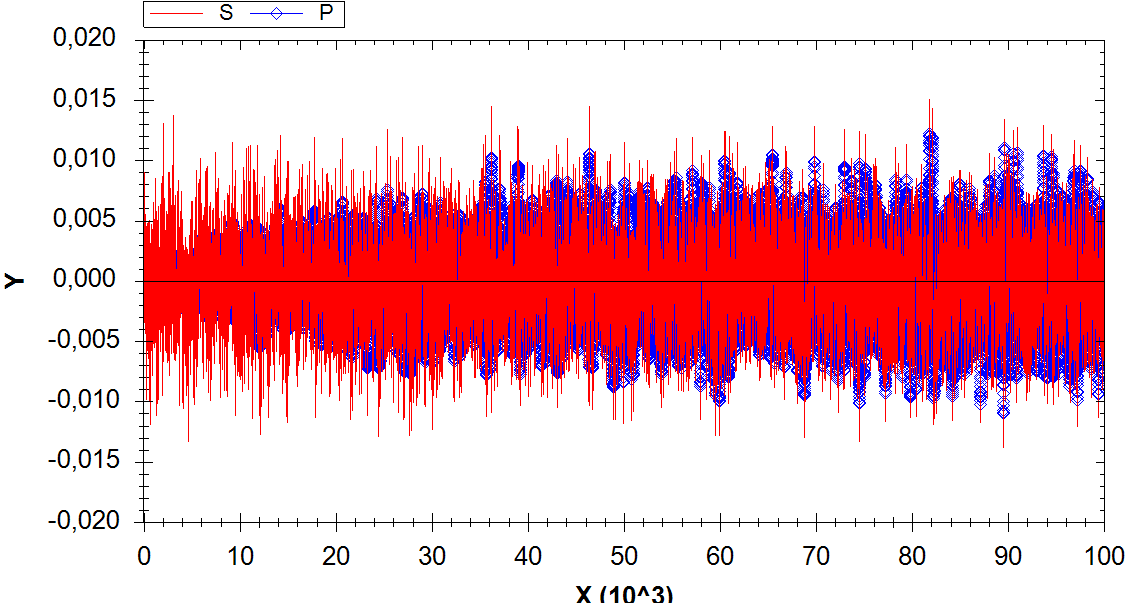


Рис. 6.79.Т=3

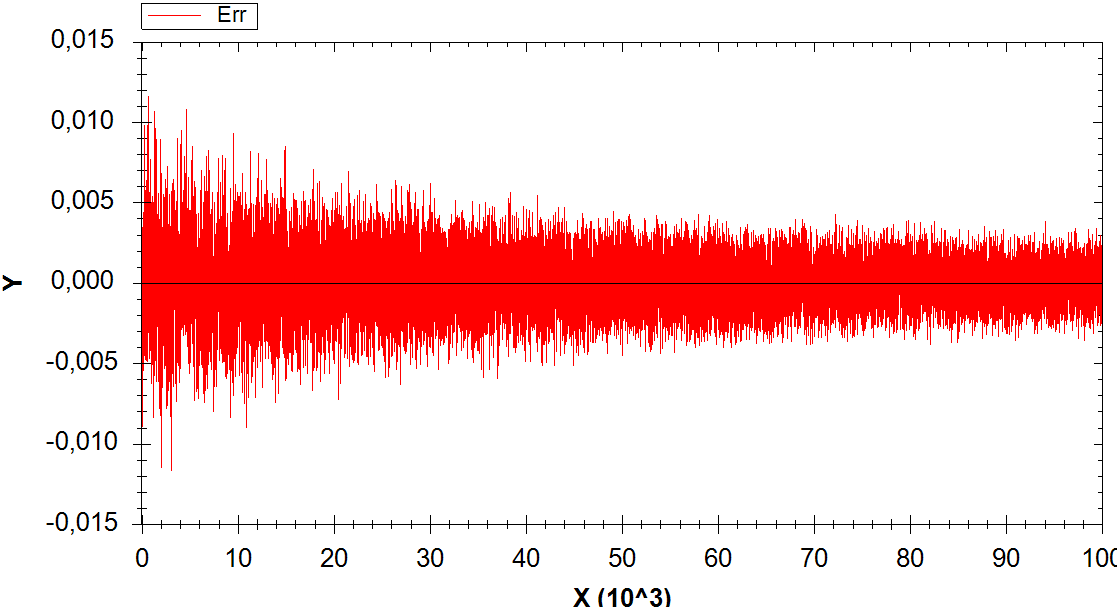


Рис. 6.80.График ошибок

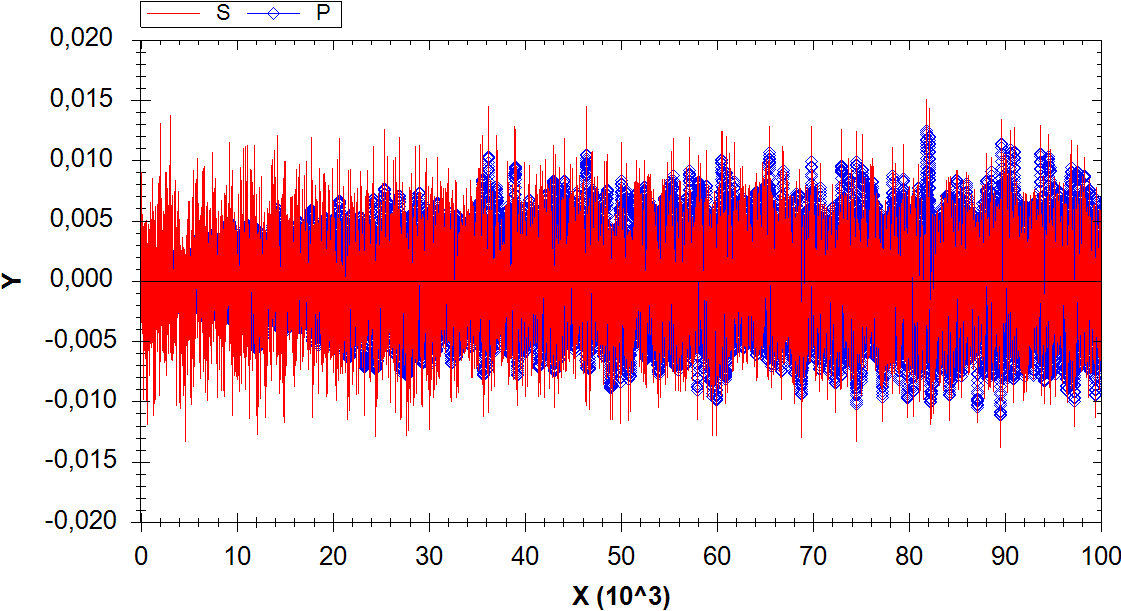


Рис. 6.81.Т=4

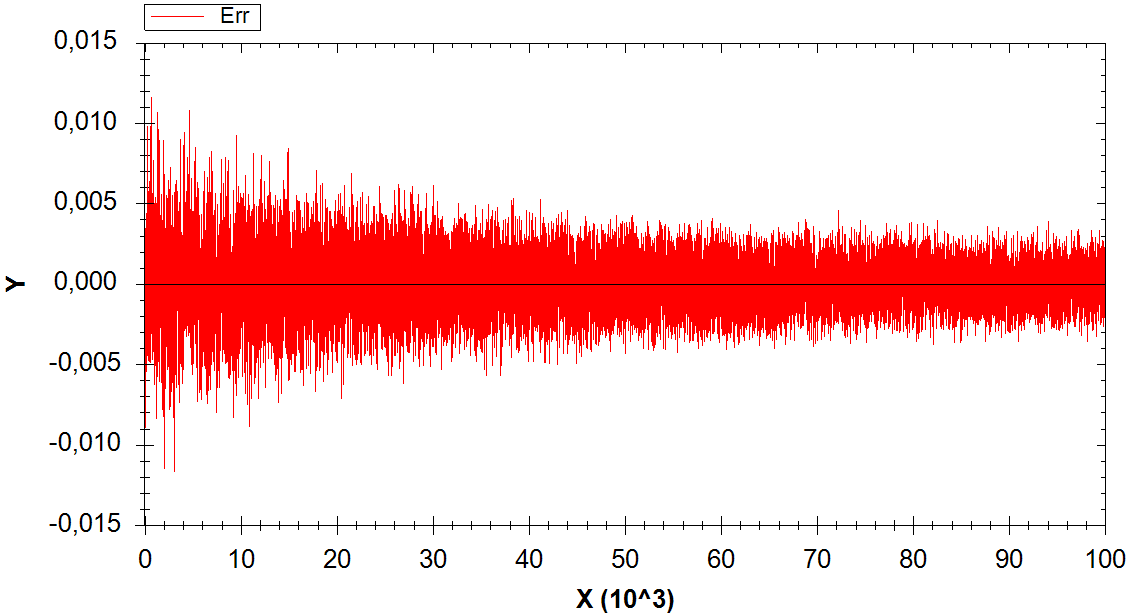


Рис. 6.82.График ошибок

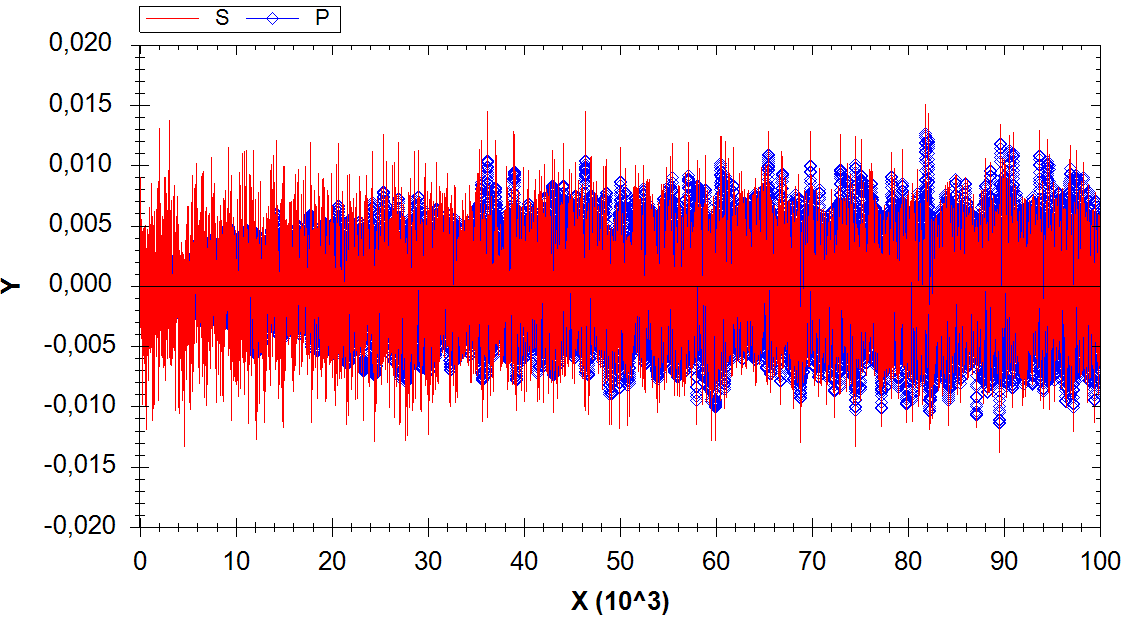


Рис. 6.83.Т=5

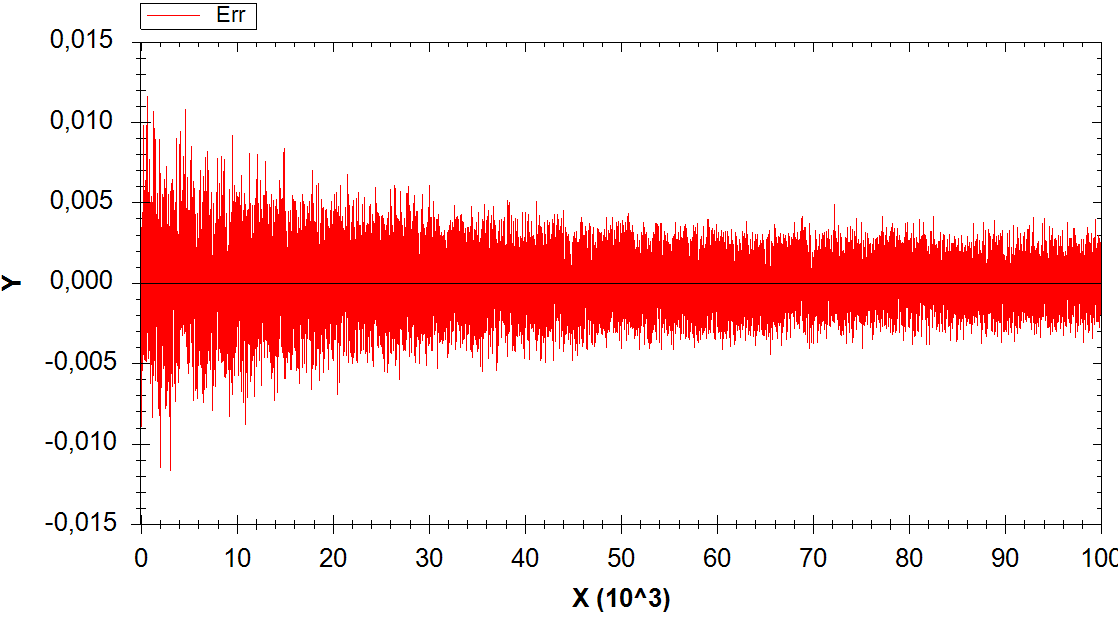


Рис. 6.84.График ошибок

# Приложение

## Программный код приложения генерации

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

using System.Net.Sockets;

using System.Net;

using System.Threading;

using System.IO;

using System.Globalization;

using ZedGraph;

namespace vs\_curs\_FirstClient

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

#region data

const int T = 1;//интервал прогнозирования

int TM;

double[,] S;//Модел сигнала, массив отсчетов

double[] E ;//Случайная величина

double[] Y;

double[] Um = new double[5];//амплитуный спектр

double[] Am = new double[5];//амплитуда

double[] Ph = new double[5];//фаза

int[] f = new int[5];//частоты по гармоникам

const double SQE = 0.01;//Среднеквадратичное значение помех

const double Mu = 0.04;

int Fd=50;//частота дискретизации

double Fs = 0.5;//Частота среза фильтров нижних частот

double Td,Ts;//Период дескритизации

double L1, L2, L3, C;//параметры нестационарности

string[] WW = new string[11];

string[] PP = new string[10];

string[] QQ = { "0", "200", "400", "600", "800", "1000", "1200", "1400", "1600", "1800", "2000" };

string pathname = "x1.txt";

#endregion

private void generate()

{

byte[] StoFile;

NumberFormatInfo formatInfo = (NumberFormatInfo)CultureInfo.GetCultureInfo("en-US").NumberFormat.Clone();

formatInfo.NumberDecimalSeparator = ".";

TM = Convert.ToInt32(TMBox.Text);

Y = new double[TM];

S = new double[4, TM];

Fd = Convert.ToInt32(FdBox.Text);

Fs = Convert.ToDouble(FsBox.Text, formatInfo);

L1 = Convert.ToDouble(L1Box.Text, formatInfo);

L2 = Convert.ToDouble(L2Box.Text, formatInfo);

L3 = Convert.ToDouble(L3Box.Text, formatInfo);

C = Convert.ToDouble(CBox.Text, formatInfo);

Random rnd = new Random();

Td = 1 / (double)Fd;

Ts = 1 / (2 \* Math.PI \* Fs);

FileStream fs = new FileStream(pathname,

FileMode.Create,

FileAccess.Write,

FileShare.None,

4096,

FileOptions.SequentialScan);

//Коэф. цифрового фиьтра

double A = Td / (Td + Ts);

double B = Ts / (Td + Ts);

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

{

Y[k] = A \* Math.Pow(-1, rnd.Next(-1,1)) \* rnd.NextDouble() + B \* Y[k - 1];

for (int j = 0; j < 4; j++)

{

S[j, k] = Km(j, k) \* (A \* Y[k] + B \* S[j, k - 1]);

StoFile = Encoding.ASCII.GetBytes(S[j,k].ToString()+"\t");

fs.Write(StoFile, 0, StoFile.Length);

}

StoFile = Encoding.ASCII.GetBytes("\n");

fs.Write(StoFile, 0, StoFile.Length);

}

SetLog("Generation succesfull" + Environment.NewLine);

fs.Close();

CreateGraph(zedGraphControl1, S, 0);

CreateGraph(zedGraphControl2, S, 1);

CreateGraph(zedGraphControl3, S, 2);

CreateGraph(zedGraphControl4, S, 3);

SetSize();

}

private double Km(int version, int index)

{

switch (version)

{

case 0:

return 1-C;

case 1:

return 1 - L1 \* index / TM;

case 2:

return 1 + L2 \* (index / TM) \* (index / TM);

case 3:

return L3 \* Math.Exp(index / TM);

default:

return 0;

}

}

//Копирование потоков бинарного типа передачи

private static long CopyStream(Stream input, Stream output, int bufferSize, long max)

{

byte[] buffer = new byte[bufferSize];

int count = 0;

long total = 0;

Form1 fr = new Form1();

while ((count = input.Read(buffer, 0, buffer.Length)) > 0)

{

output.Write(buffer, 0, count);

total += count;

}

return total;

}

public void SetLog(string str)

{

if (textBox2.InvokeRequired)

textBox2.Invoke(new Action<string>((s) => textBox2.Text += s), str);

else textBox2.Text += str;

}

private void ClientTcp()

{

//Соединяемся с удаленным устройством

try

{

//Устанавливаем удаленную конечную точку для сокета

string[] addr = maskedTextBox1.Text.Split(',');

string HostName = Convert.ToInt32(addr[0]).ToString() + "." +

Convert.ToInt32(addr[1]).ToString() + "." +

Convert.ToInt32(addr[2]).ToString() + "." +

Convert.ToInt32(addr[3]).ToString();

IPHostEntry ipHost = Dns.GetHostEntry(HostName);

IPAddress ipAddr = ipHost.AddressList[0];

Int32 port = Convert.ToInt32(textBox1.Text);

//IPEndPoint ipEndPoint = new IPEndPoint(ipAddr, Convert.ToInt32(textBox1.Text));//номер порта

TcpClient client = new TcpClient(HostName,port);

//Соединяем сокет с удаленной конечной точкой

SetLog("Connection..."+ipAddr.ToString()+Environment.NewLine);

NetworkStream stream = client.GetStream();

//Декодинг приветствия

string hello="$TakePlease$";

byte[] msg = Encoding.ASCII.GetBytes(hello);

//отправляем данные через сокет

stream.Write(msg,0,msg.Length);

SetLog(hello + Environment.NewLine);

//Получаем ответ от удаленного устройства

byte[] bytes = new byte[1024];

int RecByte = stream.Read(bytes,0,bytes.Length);

SetLog("Server reply : "+ Encoding.ASCII.GetString(bytes, 0, RecByte)+Environment.NewLine);

//Проверка установки соединения

if (Encoding.ASCII.GetString(bytes, 0, RecByte) == "$Ok$")

{

//Декодинг сообщенияя

//byte[] data = Encoding.ASCII.GetBytes(\_answer);

//отправляем данные через сокет

//stream.Write(data, 0, data.Length);

long col = 0;

using (FileStream fs = new FileStream(pathname, FileMode.Open, FileAccess.Read))

{

col = CopyStream(fs, stream, 4096, fs.Length);

fs.Close();

SetLog("File sending " + col.ToString()+Environment.NewLine);

}

SetLog("Server says : " + Encoding.ASCII.GetString(bytes, 0, RecByte) + Environment.NewLine);

}

//Освобождаем сокет

stream.Close();

client.Close();

}

catch (Exception e)

{

SetLog(e.ToString());

}

}

private void button1\_Click(object sender, EventArgs e)

{

if (S != null)

{

Thread Client = new Thread(new ThreadStart(ClientTcp));

Client.Start(); // Вызываем поток

//sending(answer);

}

}

private void button2\_Click(object sender, EventArgs e)

{

generate();

}

private void Form1\_Load(object sender, EventArgs e)

{

}

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

{

SetSize();

}

private void SetSize()

{

zedGraphControl1.Location = new Point( 10, 10 );

zedGraphControl1.Size = new Size(tabPage1.Width - 20,

tabPage1.Height - 20);

zedGraphControl2.Location = new Point(10, 10);

zedGraphControl2.Size = new Size(tabPage2.Width - 20,

tabPage2.Height - 20);

zedGraphControl3.Location = new Point(10, 10);

zedGraphControl3.Size = new Size(tabPage3.Width - 20,

tabPage3.Height - 20);

zedGraphControl4.Location = new Point(10, 10);

zedGraphControl4.Size = new Size(tabPage4.Width - 20,

tabPage4.Height - 20);

}

private void CreateGraph( ZedGraphControl zgc, double[,] func, int index )

{

// get a reference to the GraphPane

GraphPane myPane = zgc.GraphPane;

// Очистим список кривых на тот случай, если до этого сигналы уже были нарисованы

myPane.CurveList.Clear();

// Set the Titles

myPane.Title.Text = "Graphic";

myPane.XAxis.Title.Text = "X";

myPane.YAxis.Title.Text = "Y";

// Make up some data arrays based on the Sine function

double x, y;

PointPairList list1 = new PointPairList();

//PointPairList list2 = new PointPairList();

for ( int i = 0; i < func.GetLength(1); i++ )

{

x = (double)i ;

y = func[index,i];

list1.Add( x, y );

}

// Generate a red curve with diamond // symbols, and "Porsche" in the legend

LineItem myCurve = myPane.AddCurve( "S",

list1, Color.Red, SymbolType.None );

int xMax = Convert.ToInt32(UppBox.Text),

xMin = Convert.ToInt32(LowBox.Text);

myPane.XAxis.Scale.Max = xMax+20;

myPane.XAxis.Scale.Min = xMin-20;

// Вызываем метод AxisChange (), чтобы обновить данные об осях.

// В противном случае на рисунке будет показана только часть графика,

// которая умещается в интервалы по осям, установленные по умолчанию

zgc.AxisChange();

// Обновляем график

zgc.Invalidate();

}

}

}

## Программный код приложения сервера

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

using System.Net.Sockets;

using System.Net;

using System.Threading;

using System.IO;

namespace vs\_curs\_Server

{

public partial class Form1 : Form

{

string pathname = "example.txt";

Thread newServer = null;

TcpListener server = null;

public Form1()

{

InitializeComponent();

}

public void SetLog(string str)

{

if (textBox3.InvokeRequired)

textBox3.Invoke(new Action<string>((s) => textBox3.Text += s), str);

else textBox3.Text += str;

}

//Копирование потоков бинарного типа передачи

private static long CopyStream(Stream input, Stream output, int bufferSize)

{

byte[] buffer = new byte[bufferSize];

int count = 0;

long total = 0;

while ((count = input.Read(buffer, 0, buffer.Length)) > 0)

{

output.Write(buffer, 0, count);

total += count;

}

return total;

}

public void TpcServer() // Функция сервера

{

try

{

Int32 port = Convert.ToInt32(textBox1.Text); //порт сервера

IPAddress localAddr = IPAddress.Parse("127.0.0.1");//ip-адрес сервера (интерфейс)

//TcpListener - класс TCP-сервера из .Net Framework Class Library

server = new TcpListener(localAddr, port);

// начинаем ожидание подсоединений клиентов на интерфейсе localAddr и порту port

server.Start();

// буффер для приема сообщений и соответствующая ему строка для вывода на экран

Byte[] bytes = new Byte[1024];

Byte[] msg = null;

FileStream fs = null;

String data;

long col=0;

//ответ клиенту

String answer\_message;

//цикл обработки подсоединений клиентов

while (true)

{

SetLog("Waiting for a connection... " + Environment.NewLine);

// Ждем соединения клиента

TcpClient client = server.AcceptTcpClient();

//Ура! Кто-то подсоединился!

SetLog("Connected!" + Environment.NewLine);

// вводим поток stream для чтения и записи через установленное соединение

NetworkStream stream = client.GetStream();

int RecByte = stream.Read(bytes, 0, bytes.Length);

if (RecByte > 0)

{

// преобразуем принятые данные в строку ASCII string.

data = System.Text.Encoding.ASCII.GetString(bytes, 0, RecByte);

//печатаем то, что получили

SetLog("Received: " + data + Environment.NewLine);

//анализируем запрос клиента и вычисляем результат

switch (data)

{

case "$TakePlease$":

answer\_message = "$Ok$";

SetLog("Sent: " + answer\_message + Environment.NewLine);

msg = System.Text.Encoding.ASCII.GetBytes(answer\_message);

stream.Write(msg, 0, msg.Length);

SetLog("Processing..."+Environment.NewLine);

using (fs = new FileStream(pathname,

FileMode.Create,

FileAccess.Write,

FileShare.None,

4096,

FileOptions.SequentialScan))

{

col=CopyStream(stream, fs, 4096);

fs.Close();

SetLog("Received file " + col.ToString() + Environment.NewLine);

}

col = 0;

break;

case "$GivePlease$":

answer\_message = "$Ok$";

SetLog("Sent: " + answer\_message + Environment.NewLine);

msg = System.Text.Encoding.ASCII.GetBytes(answer\_message);

stream.Write(msg, 0, msg.Length);

fs = new FileStream(pathname, FileMode.Open, FileAccess.Read);

col=CopyStream(fs, stream, 4096);

fs.Close();

SetLog("Sending file" + col.ToString()+Environment.NewLine);

col = 0;

break;

}

}

// закрываем соединение

client.Close();

}

}

catch (SocketException expt)

{

SetLog("SocketException:" + expt + Environment.NewLine);

}

catch (ThreadAbortException expt)

{

//SetLog("SocketException:" + expt + Environment.NewLine);

}

finally

{

// Stop listening for new clients.

server.Stop();

}

}

private void button1\_Click(object sender, EventArgs e)

{

newServer = new Thread(new ThreadStart(TpcServer));

newServer.Start(); // Вызываем поток

}

private void button2\_Click(object sender, EventArgs e)

{

server.Stop();

newServer.Abort();

SetLog("Server stopped");

}

}

}

## Программный код приложения прогнозатора

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

using System.Net.Sockets;

using System.Net;

using System.Threading;

using System.IO;

using System.Globalization;

using ZedGraph;

namespace vs\_curs\_SecondClient

{

public partial class Form1 : Form

{

#region data

int T = 1;//интервал прогнозирования

int \_M;//Длительность интервала (количество отсчетов)

double[,] S;//Модел сигнала, массив отсчетов

double[] Um = new double[5];//амплитуный спектр

double[] Am = new double[5];//амплитуда

double[] Ph = new double[5];//фаза

int[] f = new int[5];//частоты по гармоникам

double[,] \_W;

double[,] P\_LMS;//Прогнозируемый график

double[,] P\_Vin;//Прогнозируемый график

double[,] Error\_LMS;//Ошибка

double[,] Error\_Vin;//Ошибка

const double SQE = 0.01;//Среднеквадратичное значение помех

double Mu;

int \_N;//Порядок фильтра

int \_L;//Длительность интервала проверки качества

int \_Km;

string pathname = "x2.txt";

#endregion

public Form1()

{

InitializeComponent();

}

private void button2\_Click(object sender, EventArgs e)

{

NumberFormatInfo formatInfo = (NumberFormatInfo)CultureInfo.GetCultureInfo("en-US").NumberFormat.Clone();

formatInfo.NumberDecimalSeparator = ",";

int count = 0;

string line;

StreamReader file =

new StreamReader(pathname);

SetLog("Begin reading" + Environment.NewLine);

SetProgressBar(0);

while ((line = file.ReadLine()) != null)

{

count++;

if (count == (Int32)numericUpDown1.Value)

break;

}

S = new double[4, count];

int maxprogres = count;

file.BaseStream.Position = 0;

count = 0;

while ((line = file.ReadLine()) != null)

{

string[] separate = line.Split('\t');

S[0, count] = Convert.ToDouble(separate[0], formatInfo);

try

{

S[1, count] = Convert.ToDouble(separate[1], formatInfo);

}

catch (FormatException exp)

{

S[1, count] = 0;

}

S[2, count] = Convert.ToDouble(separate[2], formatInfo);

S[3, count] = Convert.ToDouble(separate[3], formatInfo);

SetProgressBar((int)(100 \* count / maxprogres));

count++;

if (count == S.GetLength(1))

break;

if (count % 50000 == 0)

{

SetLog("Counts: " + count.ToString() + Environment.NewLine);

textBox2.Refresh();

}

}

file.Close();

SetProgressBar(100);

\_Km = (Int32)numericUpDown1.Value-1;

Mu = (double)numericUpDown2.Value;

\_N = (Int32)numericUpDown3.Value;

\_M = (Int32)numericUpDown4.Value;

\_L=\_M;

SetLog("Convertation succesfull" + Environment.NewLine);

SetLog("Counts: " + count.ToString() + Environment.NewLine);

textBox2.Refresh();

if (checkBox1.Checked == true)

{

VinnerCalculate();

}

if (checkBox2.Checked == true)

{

LMSCalculate();

}

}

private void VinnerCalculate()

{

int N = \_N;

int M = \_M;

P\_Vin = new double[4, M + 1];

Error\_Vin = new double[4, M + 1];

int count = 0;

SetProgressBar(count);

for (int index = 0; index < 4; index++)

{

double[] fR = new double[N + 1];

double[] Yk = new double[M + 1];

double[] d = new double[M + 1];

double koef = Convert.ToDouble(1 / (double)(M - N - T));

SetLog(Environment.NewLine + "VinnerCalculate begining: " + (index+1).ToString() + Environment.NewLine);

//Оценки элементов корреляционной матрицы сигнала

SetLog("Оценки элементов корреляционной матрицы сигнала" + Environment.NewLine);

for (int j = 0; j <= N; j++)

{

for (int k = N; k <= M - T; k++)

{

fR[j] += S[index, k] \* S[index, k - j];

}

fR[j] = fR[j] \* koef;

}

//Кореляционная матрица сигнла помехи

SetLog("Кореляционная матрица сигнла помехи" + Environment.NewLine);

double[,] Rx = new double[N + 1, N + 1];

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

{

for (int j = i; j <= N; j++)

{

Rx[j, i] = fR[j - i];

Rx[i, j] = fR[j - i];

}

}

SetProgressBar(count += 5);

//Вектор взаимных корреляций между отсчетами образца

SetLog("Вектор взаимных корреляций между отсчетами образца" + Environment.NewLine);

double[,] P = new double[1, N + 1];

for (int j = 0; j <= N; j++)

{

for (int k = N; k < M - T; k++)

{

P[0, j] += S[index, k + T] \* S[index, k - j];

}

P[0, j] = P[0, j] \* koef;

}

//Транспонированная матрица

SetLog("Транспонированная матрица" + Environment.NewLine);

double[,] Ptrans = new double[N + 1, 1];

for (int j = 0; j <= N; j++)

{

Ptrans[j, 0] = P[0, j];

}

SetProgressBar(count += 5);

//Обратная матрица

SetLog("Обратная матрица");

double[,] Rxrev = new double[N + 1, N + 1];

Rxrev = ObrMatrix(Rx,1);

//Параметры оптимально прогнозирующего фильтра

SetLog("Параметры оптимально прогнозирующего фильтра" + Environment.NewLine);

double[,] W = new double[N + 1, N + 1];

W = Multiplication(Rxrev, Ptrans);

//Прогнозируемое значение

SetLog("Прогнозируемое значение" + Environment.NewLine);

for (int k = N; k < M; k++)

{

for (int j = 0; j <= N; j++)

{

Yk[k] += S[index, k - j] \* W[j, 0];

}

}

SetProgressBar(count += 5);

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

{

P\_Vin[index, i] = Yk[i];

}

//Ошибка прогнозирования в k точке

SetLog("Ошибка прогнозирования в k точке" + Environment.NewLine);

for (int k = N; k < M; k++)

{

d[k] = S[index, k] - Yk[k - T];

}

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

{

Error\_Vin[index, i] = d[i];

}

SetProgressBar(count += 5);

//СКЗ ошибки прогнозирования

SetLog("СКЗ ошибки прогнозирования" + Environment.NewLine);

double SQd = 0;

double preSQd = 0;

int L = \_L;

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

{

preSQd += d[k] \* d[k];

}

SQd = Math.Sqrt((1 / L) \* preSQd);

SetProgressBar(count += 5);

}

SetProgressBar(100);

//Графики

CreateGraph(Vinner\_S0\_ZGC, S, P\_Vin, 0, "Prognoz s0");

CreateGraph(Vinner\_S1\_ZGC, S, P\_Vin, 1, "Prognoz s1");

CreateGraph(Vinner\_S2\_ZGC, S, P\_Vin, 2, "Prognoz s2");

CreateGraph(Vinner\_S3\_ZGC, S, P\_Vin, 3, "Prognoz s3");

SetProgressBar(0);

}

//Обратная матрица

public double[,] ObrMatrix(double[,] A, int cn)

{

double[,] Mat;

Mat = MulMatxix(A, cn);

int info = 0;

alglib.matinv.matinvreport port = new alglib.matinv.matinvreport();

alglib.matinv.rmatrixinverse(ref Mat, Mat.GetLength(1), ref info, port);

Mat = DivMatxix(Mat, cn);

return Mat;

}

//Для вычисления W

public double[,] Multiplication(double[,] a, double[,] b)

{

if (a.GetLength(1) != b.GetLength(0)) throw new Exception("Матрицы нельзя перемножить");

double[,] r = new double[a.GetLength(0), b.GetLength(1)];

for (int i = 0; i < a.GetLength(0); i++)

{

for (int j = 0; j < b.GetLength(1); j++)

{

for (int k = 0; k < b.GetLength(0); k++)

{

r[i, j] += a[i, k] \* b[k, j];

}

}

}

return r;

}

//Для обратной матрицы

public double[,] MulMatxix(double[,] A, double kf)

{

double[,] resMat = new double[A.GetLength(0), A.GetLength(1)];

for (int i = 0; i < A.GetLength(0); i++)

for (int j = 0; j < A.GetLength(1); j++)

{

resMat[i, j] = A[i, j] \* kf;

}

return resMat;

}

//Для обратной матрицы

public double[,] DivMatxix(double[,] A, double kf)

{

double[,] resMat = new double[A.GetLength(0), A.GetLength(1)];

for (int i = 0; i < A.GetLength(0); i++)

for (int j = 0; j < A.GetLength(1); j++)

{

resMat[i, j] = A[i, j] / kf;

}

return resMat;

}

private void LMSCalculate()

{

P\_LMS = new double[4, \_Km];

Error\_LMS = new double[4, \_Km];

\_W = new double[\_Km + 1, \_N];

int progres = 0;

SetProgressBar(0);

for (int i = 0; i < 4; i++)

{

double[] buffer = new double[\_Km];

double[] ERRbuffer = new double[\_Km];

buffer = CalcLMS(S, i, \_N, \_Km, Mu, out \_W, out ERRbuffer);

for (int j = 0; j < P\_LMS.GetLength(1); j++)

{

P\_LMS[i, j] = buffer[j];

Error\_LMS[i, j] = ERRbuffer[j];

progres++;

if((progres%10)==0)

SetProgressBar((int)(100 \* progres / P\_LMS.Length));

}

}

SetProgressBar(100);

statusStrip1.Refresh();

//Графики

CreateGraph(LMS\_S0\_ZGC, S, P\_LMS, 0, "Prognoz s0");

CreateGraph(LMS\_S1\_ZGC, S, P\_LMS, 1, "Prognoz s1");

CreateGraph(LMS\_S2\_ZGC, S, P\_LMS, 2, "Prognoz s2");

CreateGraph(LMS\_S3\_ZGC, S, P\_LMS, 3, "Prognoz s3");

SetProgressBar(0);

statusStrip1.Refresh();

}

public void SetProgressBar(int c)

{

this.toolStripProgressBar1.ProgressBar.Value = c;

this.toolStripProgressBar1.ProgressBar.Refresh();

}

public void SetStatusLabel(string s)

{

this.toolStripStatusLabel1.Text = s;

statusStrip1.Refresh();

}

private double[] CalcLMS(double[,] X, int num\_Graph, int N, int Km, double mu, out double[,] W, out double[] Er)

{

W = new double[Km + 1, N];

Er = new double[Km];

double[] Y = new double[Km];

double E;

for (int n = N; n < Km; n++)

{

for (int j = 0; j < N; j++)

{

Y[n] += W[n-1, j] \* X[num\_Graph, n - j];

}

E = X[num\_Graph, n] - Y[n];

for (int j = 0; j < N; j++)

{

W[n, j] = W[n-1, j] + mu \* E \* X[num\_Graph, n - j];

}

}

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

{

Er[k] = Y[k] - X[num\_Graph, k];

}

return Y;

}

//Копирование потоков бинарного типа передачи

private static long CopyStream(Stream input, Stream output, int bufferSize)

{

byte[] buffer = new byte[bufferSize];

int count = 0;

long total = 0;

while ((count = input.Read(buffer, 0, buffer.Length)) > 0)

{

output.Write(buffer, 0, count);

total += count;

}

return total;

}

public void SetLog(string str)

{

if (textBox2.InvokeRequired)

textBox2.Invoke(new Action<string>((s) => textBox2.Text += s), str);

else textBox2.Text += str;

textBox2.Refresh();

}

private void ClientTcp()

{

//Соединяемся с удаленным устройством

try

{

//Устанавливаем удаленную конечную точку для сокета

string[] addr = maskedTextBox1.Text.Split(',');

string HostName = Convert.ToInt32(addr[0]).ToString() + "." +

Convert.ToInt32(addr[1]).ToString() + "." +

Convert.ToInt32(addr[2]).ToString() + "." +

Convert.ToInt32(addr[3]).ToString();

IPHostEntry ipHost = Dns.GetHostEntry(HostName);

IPAddress ipAddr = ipHost.AddressList[0];

Int32 port = Convert.ToInt32(textBox1.Text);

//IPEndPoint ipEndPoint = new IPEndPoint(ipAddr, Convert.ToInt32(textBox1.Text));//номер порта

TcpClient client = new TcpClient(HostName, port);

//Соединяем сокет с удаленной конечной точкой

SetLog("Connection..." + ipAddr.ToString() + Environment.NewLine);

NetworkStream stream = client.GetStream();

//Декодинг приветствия

string hello = "$GivePlease$";

byte[] msg = Encoding.ASCII.GetBytes(hello);

//отправляем данные через сокет

stream.Write(msg, 0, msg.Length);

SetLog(hello + Environment.NewLine);

//Получаем ответ от удаленного устройства

byte[] bytes = new byte[1024];

int RecByte = stream.Read(bytes, 0, bytes.Length);

SetLog("Server reply : " + Encoding.ASCII.GetString(bytes, 0, RecByte) + Environment.NewLine);

//Проверка установки соединения

if (Encoding.ASCII.GetString(bytes, 0, RecByte) == "$Ok$")

{

//Декодинг сообщенияя

//byte[] data = Encoding.ASCII.GetBytes(\_answer);

//отправляем данные через сокет

//stream.Write(data, 0, data.Length);

long col = 0;

FileStream fs = new FileStream(pathname,

FileMode.Create,

FileAccess.Write,

FileShare.None,

4096,

FileOptions.SequentialScan);

col=CopyStream(stream, fs, 4096);

fs.Close();

SetLog("File received " + col.ToString()+Environment.NewLine);

}

//Освобождаем сокет

stream.Close();

client.Close();

}

catch (Exception e)

{

SetLog(e.ToString());

}

}

private void button1\_Click(object sender, EventArgs e)

{

Thread Client = new Thread(new ThreadStart(ClientTcp));

Client.Start(); // Вызываем поток

//sending(answer);

}

private void SetSize()

{

Vinner\_S0\_ZGC.Location = new Point(10, 10);

Vinner\_S0\_ZGC.Size = new Size(Vinner\_S0.Width - 20,

Vinner\_S0.Height - 20);

Vinner\_S1\_ZGC.Location = new Point(10, 10);

Vinner\_S1\_ZGC.Size = new Size(Vinner\_S1.Width - 20,

Vinner\_S1.Height - 20);

Vinner\_S2\_ZGC.Location = new Point(10, 10);

Vinner\_S2\_ZGC.Size = new Size(Vinner\_S2.Width - 20,

Vinner\_S2.Height - 20);

Vinner\_S3\_ZGC.Location = new Point(10, 10);

Vinner\_S3\_ZGC.Size = new Size(Vinner\_S3.Width - 20,

Vinner\_S3.Height - 20);

LMS\_S0\_ZGC.Location = new Point(10, 10);

LMS\_S0\_ZGC.Size = new Size(LMS\_S0.Width - 20,

LMS\_S0.Height - 20);

LMS\_S1\_ZGC.Location = new Point(10, 10);

LMS\_S1\_ZGC.Size = new Size(LMS\_S1.Width - 20,

LMS\_S1.Height - 20);

LMS\_S2\_ZGC.Location = new Point(10, 10);

LMS\_S2\_ZGC.Size = new Size(LMS\_S2.Width - 20,

LMS\_S2.Height - 20);

LMS\_S3\_ZGC.Location = new Point(10, 10);

LMS\_S3\_ZGC.Size = new Size(LMS\_S3.Width - 20,

LMS\_S3.Height - 20);

}

private void CreateGraph(ZedGraphControl zgc, double[,] func, int index, string name)

{

// get a reference to the GraphPane

GraphPane myPane = zgc.GraphPane;

// Очистим список кривых на тот случай, если до этого сигналы уже были нарисованы

myPane.CurveList.Clear();

// Set the Titles

myPane.Title.Text = name;

myPane.XAxis.Title.Text = "X";

myPane.YAxis.Title.Text = "Y";

// Make up some data arrays based on the Sine function

double x, y1;

PointPairList list1 = new PointPairList();

for (int i = 0; i < func.GetLength(1); i++)

{

x = (double)i;

y1 = func[index, i];

list1.Add(x, y1);

}

// Generate a red curve with diamond // symbols, and "Porsche" in the legend

LineItem myCurve1 = myPane.AddCurve("Err",

list1, Color.Red, SymbolType.None);

int xMax = func.GetLength(1),

xMin = 0;

myPane.XAxis.Scale.Max = xMax + 20;

myPane.XAxis.Scale.Min = xMin - 20;

// Вызываем метод AxisChange (), чтобы обновить данные об осях.

// В противном случае на рисунке будет показана только часть графика,

// которая умещается в интервалы по осям, установленные по умолчанию

zgc.AxisChange();

// Обновляем график

zgc.Invalidate();

}

private void CreateGraph(ZedGraphControl zgc, double[,] func, double[,] prognoz, int index, string name)

{

// get a reference to the GraphPane

GraphPane myPane = zgc.GraphPane;

// Очистим список кривых на тот случай, если до этого сигналы уже были нарисованы

myPane.CurveList.Clear();

// Set the Titles

myPane.Title.Text = name;

myPane.XAxis.Title.Text = "X";

myPane.YAxis.Title.Text = "Y";

// Make up some data arrays based on the Sine function

double x, y1,y2;

PointPairList list1 = new PointPairList();

PointPairList list2 = new PointPairList();

for (int i = 0; i < prognoz.GetLength(1); i++)

{

x = (double)i;

try

{

y1 = func[index, i];

list1.Add(x, y1);

}

catch (IndexOutOfRangeException exp)

{

}

y2 = prognoz[index, i];

list2.Add(x, y2);

}

// Generate a red curve with diamond // symbols, and "Porsche" in the legend

LineItem myCurve1 = myPane.AddCurve("S",

list1, Color.Red, SymbolType.None);

LineItem myCurve2 = myPane.AddCurve("P",

list2, Color.Blue, SymbolType.Diamond);

int xMax = prognoz.GetLength(1),

xMin = 0;

myPane.XAxis.Scale.Max = xMax + 20;

myPane.XAxis.Scale.Min = xMin - 20;

zgc.AxisChange();

// Обновляем график

zgc.Invalidate();

}

private void checkBox3\_CheckedChanged(object sender, EventArgs e)

{

if (checkBox3.Checked == true)

groupBox2.Enabled = true;

else

{

groupBox2.Enabled = false;

if (checkBox2.Checked == true)

{

//Графики

CreateGraph(LMS\_S0\_ZGC, S, P\_LMS, 0, "Prognoz s0");

CreateGraph(LMS\_S1\_ZGC, S, P\_LMS, 1, "Prognoz s1");

CreateGraph(LMS\_S2\_ZGC, S, P\_LMS, 2, "Prognoz s2");

CreateGraph(LMS\_S3\_ZGC, S, P\_LMS, 3, "Prognoz s3");

}

if (checkBox1.Checked == true)

{

//Графики

CreateGraph(Vinner\_S0\_ZGC, S, P\_Vin, 0, "Prognoz s0");

CreateGraph(Vinner\_S1\_ZGC, S, P\_Vin, 1, "Prognoz s1");

CreateGraph(Vinner\_S2\_ZGC, S, P\_Vin, 2, "Prognoz s2");

CreateGraph(Vinner\_S3\_ZGC, S, P\_Vin, 3, "Prognoz s3");

}

}

}

private void button3\_Click(object sender, EventArgs e)

{

if (radioButton1.Checked == true)

{

if (checkBox1.Checked == true)

{

if (Error\_Vin != null)

{

//Ошибки

CreateGraph(Vinner\_S0\_ZGC, Error\_Vin, 0, "Err s0");

CreateGraph(Vinner\_S1\_ZGC, Error\_Vin, 1, "Err s1");

CreateGraph(Vinner\_S2\_ZGC, Error\_Vin, 2, "Err s2");

CreateGraph(Vinner\_S3\_ZGC, Error\_Vin, 3, "Err s3");

}

}

if (checkBox2.Checked == true)

{

if (Error\_LMS != null)

{

//Ошибки

CreateGraph(LMS\_S0\_ZGC, Error\_LMS, 0, "Err s0");

CreateGraph(LMS\_S1\_ZGC, Error\_LMS, 1, "Err s1");

CreateGraph(LMS\_S2\_ZGC, Error\_LMS, 2, "Err s2");

CreateGraph(LMS\_S3\_ZGC, Error\_LMS, 3, "Err s3");

}

}

}

if (radioButton2.Checked == true)

{

//оригинал

CreateGraph(Vinner\_S0\_ZGC, S, 0, "Original s0");

CreateGraph(Vinner\_S1\_ZGC, S, 1, "Original s1");

CreateGraph(Vinner\_S2\_ZGC, S, 2, "Original s2");

CreateGraph(Vinner\_S3\_ZGC, S, 3, "Original s3");

CreateGraph(LMS\_S0\_ZGC, S, 0, "Original s0");

CreateGraph(LMS\_S1\_ZGC, S, 1, "Original s1");

CreateGraph(LMS\_S2\_ZGC, S, 2, "Original s2");

CreateGraph(LMS\_S3\_ZGC, S, 3, "Original s3");

}

}

}

}