Работа №3. Составление и решение алгоритмов на примере задач моделирования последовательностей равномерно распределённых псевдослучайных чисел

Цель работы: Освоить навыки программирования алгоритмов на примере моделирования равномерно распределённых псевдослучайных целых чисел.

Задачи - написать программу:

1) генерирующую последовательность равномерно распределённых псевдослучайных целых чисел на заданном интервале [a, b];
2) выводящую сгенерированную последовательность в числовом и графическом видах;
3) формирующую и выводящую для визуального анализа последовательности: тест-квадрат и автокоррелограмму.

Время выполнения: 2 часа.

Краткие теоретические сведения

Одним из важнейших направлений использования вычислительной техники является так называемое имитационное моделирование, когда достаточно сложные характеристики исследуемого объекта изучаются на запрограммированной модели этого объекта. Как правило, исследуемые объекты и их модели являются стохастическими, то есть одной из составляющих их числовых характеристик является случайная составляющая. Поэтому одной из задач имитационного моделирования является получение случайных чисел.

В трёх последних работах будут рассмотрены три этапа имитационного моделирования:

1) программирование генерирования равномерно распределённых случайных чисел;
2) программирование получения случайных чисел с заданными законами распределения;
3) программирование характеристик некоторых моделей со случайной составляющей, имеющей заданный закон распределения.

Первый этап является предметом настоящей работы.

Предполагается, что ко времени выполнения настоящей лабораторной работы

используемые математические понятия рассматривались по дисциплине “Прикладная математика”.

Рекомендуется также освежить знания о числах в Javascript, повторив материал лекции №3 или материал размещённый здесь: http://learn.javascript.ru/number.

Очевидно, что последовательность случайных числел нельзя выработать, используя детерминированную последовательность действий (алгоритм), запрограммированную на вычислительной машине. Однако можно создать такую последовательность чисел, которая будет приближённо отражать те свойства случайных чисел, которые нужны для конкретной прикладной задачи. Такие числа называются псевдослучайными. Впервые предложил их использовать Джон фон Нейман в 1946 г. Его метод состоял в следующем: n-разрядное число возводилось в квадрат и из него выбирались средние n чисел. Метод был очень несовершенен, последовательности таких чисел часто вырождались в 0 или зацикливались с коротким периодом.

Позднее было предложено много различных алгоритмов выработки псевдослучайных чисел (ПСЧ) и последовательностей псевдослучайных чисел (ППСЧ). В языках программирования, в электронных таблицах и многих других пакетах программ имеются встроенные генераторы ПСЧ и ППСЧ. Но иногда эти генераторы могут выдавать не те результаты, которые требуются для конкретной задачи. В этом случае не трудно самостоятельно создать генератор с требуемыми свойствами. Некоторые, наиболее простые из них, но вполне достаточные для использования во многих задачах моделирования в гидробиологии, осваиваются при выполнении заданий настоящей лабораторной работы. Будем считать, что такие генераторы должны удовлетворять, как минимум, следующими требованиям:

1) число машинных операций, затрачиваемых на выработку одного случайного числа, должно быть минимальным;
2) числа в последовательности должны иметь постоянные математическое ожидание и дисперсию (среднее квадратическое отклонение);
3) числа в последовательности должны быть взаимно независимыми (последовательность не должна зацикливаться или цикл должен быть с периодом, превышающим длину используемой последовательности).

Следует обратить внимание на следующие особенности используемых методов генерации ППСЧ.

Линейные конгруэнтные методы (предложены Д. Х. Лемером). Вычислительная формула одного из таких методов, называемого смешанным конгруэнтным генератором,

записанная математическими символами

$$x_{n+1} = a \cdot x_n + c \pmod{m}, \qquad n=0, 1, 2, \ldots, \mathbf{N} \tag{3.8.1}$$

в терминах языка программирования Javascript будет иметь вид:

$$x[n+1] = (a * x[n] + c) \% m; \tag{3.8.1'}$$

где a, c, m и x[0] — задаваемые целые числа;

x[*n*] — вычисляемое целое псевдослучайное число из интервала [0, *m*);

N — количество псевдослучайных чисел или длина генерируемой последовательности.

Равномерно распределённые числа на интервале [0, 1) затем могут быть получены по формуле

$$r[n+1] = x[n+1] / m; \tag{3.8.1''}$$

Алгоритм с использованием формул (3.8.1')-(3.8.1") используется в методе Math.random() языка Javascript.

Алгоритм зацикливается с периодом не превышающим *m*. Коэффициенты a, m и x(0) могут принимать произвольные целые значения, за исключением 0. Коэффициент *c* может быть также и 0, но в этом случае сокращается период. Значение для *m* обычно выбирается равным 2^{32}, 2^{64} и т. д. в зависимости от разрядности чисел в ЭВМ, что делает ненужной операцию деления, которая автоматически выполнится при переполнении. a можно взять равным, например, 1664525, c - равным 1013904223.

Такой метод часто реализуется в современных системах программирования, хотя он не применим в криптогрфии, так как по четырём рядом расположенным значениям последовательности можно восстановить значения коэффициентов и затем воспроизводить последовательность. Но для задач моделирования в гидробиологии обычно не существенны проблемы криптографической защиты (хотя многое о методах генерации ППСП можно почерпнуть и из криптографии).

Важно иметь в виду, что при использовании линейного конгруэнтного метода период последовательности будет равен *m*, только при выполнении следующих условий:

если модуль *m:* $m>0$;

если множитель *a:* $(0<=a<m)$;

если приращение *c:* $(0<=c<m)$;

если начальное значение X_0: $(0<= X_0<m)$;

числа *c* и *m* взаимно простые;

(*a*-1) кратно *p* для каждого простого *p*, являющегося делителем *m.* Например: если *m* кратно 4, то и (*a*-1) должно быть кратно 4.

Мультипликативный конгруэнтный метод является частным случаем смешанного метода. Формула этого метода получается из (3.8.1) при $c = 0$ и может быть записана в виде:

$$x_{n+1} = a \cdot x_n \pmod{m}, \qquad n=0, 1, 2, \ldots, N \tag{3.8.2}$$

или в терминах языка программирования Javascript:

$$x[n+1] = (a * x[n]) \% m; \tag{3.8.2'}$$

Обобщённый аддитивный конгруэнтный метод, который предложили Грин, Смит и Клем (метод Фибоначчи – частный случай этого метода), при использовании которого псевдослучайная последовательность генерируется по формуле:

$$x_{n+1} = x_n + x_{n-j} \pmod{m}, \qquad n=0, 1, 2, \ldots, N \tag{3.8.3}$$

или в терминах языка программирования Javascript:

$$x[n+1] = (x[n] + x[n-j]) \% m, \tag{3.8.3'}$$

где первые два значения задаются, затем несколько, например 100, первых значений вычисляются по (3.8.3'), а последовательность генерируется по (3.8.3') со 101-го числа Фибоначчи.

Аддитивный конгруэнтный метод Фибоначчи – частный случай обобщённого аддитивного конгруэнтного метода (приведён выше), в котором последовательность чисел может быть получена, при x[0]=0 и x[1]=1, по формуле:

$$x_{n+1} = x_n + x_{n-1} \pmod{m}, \qquad n=0, 1, 2, \ldots, N \tag{3.8.3''}$$

или в терминах языка программирования Javascript:

$$x[n+1] = (x[n] + x[n-1]) \% m, \tag{3.8.3'''}$$

где первые два значения задаются (x[0]=0 и x[1]=1), затем несколько, например 100, первых значений вычисляются как обычные числа Фибонначчи (см. примеры с числами Фибоначчи в лекции №3), а последовательность генерируется по (3.8.3') со 101-го числа Фибоначчи.

Статистическим испытаниям должна быть подвергнута всякая программа генерации псевдослучайных последовательностей для определения степени близости различных статистических характеристик вырабатываемых ППСП к теоретическим значениям этих же характеристик для “истинно” случайных равномерно

распределённых чисел. Для этого используют целый набор количественных и визуальных методов. В настоящей лабораторной надо будет использовать визуальные методы (анализ графика последовательности и тест-квадрат) и один приближённый количественный метод (анализ автокоррелограммы).

График последовательности строится легко: по оси абсцисс откладывается номер числа в последовательности, а по оси ординат – значение числа. Визуально на таком графике можно обнаружить следующие особенности, свидетельствующие о явно выраженной неслучайности в нашей псевдослучайной последовательности (ППСП): тренд среднего положения, изменение разброса точек, и периодичности в последовательности точек. Ясно, что таких особенностей в генерируемой ППСП быть не должно.

Построение тест-квадрата (распределения на плоскости), состоит в построении на плоскости (*x, y*) множества точек, координаты которых определяются соседними элементами тестируемой ППСП:

$$x = \mathrm{int}(N_x \cdot r_i), \quad y = \mathrm{int}(N_y \cdot r_{i+1}) \tag{3.8.4}$$

где N_x и N_y – разрешение изображения; r_i – тестируемая последовательность; int() - математический оператор взятия целой части вещественного числа.

Обычно используется двухцветное изображение, когда на белом фоне наносятся, например, чёрные точки. При достаточно большом *N* неслучайность в псевдослучайной последовательности прослеживается в виде неравномерного расположения точек (полос, сгущений).

Можно также увеличить информативность изображения (или, говорят, мерность изображения), используя цвета. Например, в цветовой RGB-модели можно на плоскости отобразить *n*-мерное распределение точек, когда исследуются на зависимость не только соседние члены последовательности со сдвигом на единицу, но и до *n* сдвигов. Применительно к базовым RGB-цветам пикселей изображения в canvas-элементе формулы для цветного теста при трёх сдвигах можно записать в виде: $B = \mathrm{int}(255 \cdot r_{i+2})$, $G = \mathrm{int}(255 \cdot r_{i+3})$, $R = \mathrm{int}(255 \cdot r_{i+4})$.

$$x = \mathrm{int}(N_x \cdot r_i), \quad y_R = \mathrm{int}(N_y \cdot r_{i+1}), \quad y_G = \mathrm{int}(N_y \cdot r_{i+2}), \quad y_B = \mathrm{int}(N_y \cdot r_{i+3}) \tag{3.8.5}$$

Коэффициенты автокорреляции можно вычислить по формуле:

$$r_i = \frac{\frac{1}{N-i}\cdot\sum_{j=1}^{N-i}(x_j-\overline{x})\cdot(x_{j+i}-\overline{x})}{D_x}, \quad (3.8.6)$$

где N – количество чисел (или длина) последовательности, i – сдвиг, а величины $\overline{x}$ - среднее арифметическое и D_x – дисперсия, вычисляются по формулам:

$$\overline{x} = \frac{1}{N}\sum_{j=1}^{N} x_j, \quad (3.8.7)$$

$$D_x = \frac{1}{N}\sum_{j=1}^{N}(x_j-\overline{x})^2. \quad (3.8.8)$$

Знáчимость коэффициентов автокорреляции можно проверить, сравнивая их с удвоенной среднеквадратической (говорят, стандартной) ошибкой. Приближённое значение стандартной ошибки коэффициента автокорреляции можно рассчитать по формуле

$$S = \frac{1}{\sqrt{N-i}}. \quad (3.8.9)$$

где $(N - i)$ – количество значений, используемое для расчёта коэффициента автокорреляции.

Считается, что коэффициент автокорреляции статистически знáчим, если он превышает по величине свою *удвоенную* стандартную ошибку. Знáчимые «всплески» на автокоррелограмме могут означать наличие периодичностей в анализируемой последовательности с периодами, равными сдвигам, для которых знáчимы коэффициенты автокорреляции. Ясно, что качественные генераторы не должны генерировать последовательности с периодичностями. Эта проверка является приближённой (ориентировочной), аналогичной проверке «два сигма».

Задания

Задание в целом состоит в том, что каждому студенту необходимо написать программу в виде html-документа со скриптами на языке Javascript, на которой поэтапно, по событию, создаваемому пользователем (например, по клику мышкой), решались бы следующие задания:

1) генерация последовательности равномерно распределённых псевдослучайных целых чисел на задаваемом пользователем интервале [a, b] с выводом:
а) в виде чисел (номер числа: число);
б) в виде графика типа «xy», x — номер числа в последовательности, y — значение числа. Вариант алгоритма для генератора последовательности выбирается из таблицы 3.8.1. Номер варианта должен совпадать с двумя последними цифрами номера зачётной книжки;

2) построение и вывод тест-квадрата (одноцветного и трёхцветного);

3) расчёт коэффициентов автокорреляции (автокорреляционной функции) и их стандартных ошибок и вывод:
а) в виде числовых значений и в виде автокоррелограммы с нанесёнными значениями удвоенных стандартных ошибок, показывающих полосу, наличие которой на графике позволяет визуально легко представить о наличии или отсутствии зна́чимых коэффициентов.

Таблица 3.8.1 Варианты алгоритмов для генераторов псевдослучайных чисел

№ варианта	Алгоритм	Параметры

00	Использование встроенного метода random() объекта Math:
	`function int_rand(min,max) {`
	`  var i_rand = min + Math.floor((max - min + 1) * Math.random());`
	`  return i_rand;`
	`}`
	Вызов функции:
	`var x = new Array();`
	`for (var i = 0; i < n; i++) {`
	`  //x[i] = int_rand(a,b);`
	`  x.push(int_rand(a,b));`
	`}`
		-

01	Использование сочетания встроенных методов getTime() и getMilliseconds() объекта Date:
	`function int_rand_data(min,max) {`
	`  var now=new Date();`
	`  var num=(now.getTime() *now.getMilliseconds()`
	`  *now.getMilliseconds())%(max - min + 1);`
	`  var i_rand = min + num;`
	`  delete now;`
	`  return i_rand;`
	`}`
	Вызов функции:
	`var x = new Array();`
	`for (var i = 0; i < n; i++) {`
	`  for (var j = 0; j < 250000; j++) {`
	`  var temp=j*j;`
	`  }`
	`  x.push(int_rand_data(a,b));`
	`}`
		-

№ варианта	Алгоритм	Параметры

02	Аддитивный конгруэнтный метод (алгоритм Фибоначчи):
	function int_rand_Fibonachi(m_,x_i1,x_i0) {
	var x_i2=(x_i1+x_i0) % m_;
	var i_rand = x_i2;
	return i_rand;
	}
	Вызов функции:
	var x = new Array();
	m=565;
	x[0]=0;x[1]=1;
	for (var i = 2; i <= 100; i++) {
	x[i]=int_rand_Fibonachi(m,x[i-1],x[i-2]);
	}
	x[0]=x[99];x[1]=x[100];
	for (var i = 2; i < 101; i++) {
	x.shift();
	}
	for (var i = 2; i <= (n+1); i++) {
	x[i]=int_rand_Fibonachi(m,x[i-1],x[i-2]);
	}
	for (var i = 0; i < n; i++) {
	x[i]=x[i+2];
	x[i]=x[i]/m;
	x[i]=a+Math.floor(x[i]*(b-a));
	}
	x.shift();
	x.shift();
		m = 565;

№ варианта	Алгоритм	Параметры

03	Мультипликативный конгруэнтный метод:
	`function int_rand_LCM(m_,c_,a_,xi_1) {`
	`  var xi=(a_*xi_1+c_) % m_;`
	`  var i_rand=xi;`
	`  return i_rand;`
	`}`
	Вызов функции:
	`var x = new Array();`
	`x[0]=m-a_lcm;`
	`for (var i = 1; i <= n; i++) {`
	`  x[i]=int_rand_LCM(m,c,a_lcm,x[i-1]);`
	`}`
	`for (var i = 0; i < n; i++) {`
	`  x[i]=x[i+1];`
	`  x[i]=x[i]/m;`
	`  x[i]=a+Math.floor(x[i]*(b-a));`
	`}`
	`x.pop();`
		$m = 2^{63}$;
		c = 0;
		a_lcm = 16807

04	Смешанный конгруэнтный метод:
	`function int_rand_LCM(m_,c_,a_,xi_1) {`
	`  var xi=(a_*xi_1+c_) % m_;`
	`  var i_rand=xi;`
	`  return i_rand;`
	`}`
	Вызов функции:
	`var x = new Array();`
	`x[0]= 3887;`
	`for (var i = 1; i <= n; i++) {`
	`  x[i]=int_rand_LCM(m,c,a_lcm,x[i-1]);`
	`}`
	`for (var i = 0; i < n; i++) {`
	`  x[i]=x[i+1];`
	`  x[i]=x[i]/m;`
	`  x[i]=a+Math.floor(x[i]*(b-a));`
	`}`
	`x.pop();`
		$m = 2^{63}$;
		a = 16070093;
		c = 453816693

05	Арифметический оператор с синусоидой x[i+1)=i*x[i]*sin(i*x[i]):
	`function int_rand_arithmetic_1(i_,xi_1,min,max) {`
	`  var i_rand = (xi_1*Math.PI/180*i_)`
	`  *Math.sin(xi_1*Math.PI/180*i_);`
	`  return min +  Math.floor((max - min + 1)`
	`  * (i_rand - Math.floor(i_rand)));`
	`}`
	Вызов функции:
	`var x = new Array();`
	`  x[0]=Math.pow(2,64);`
	`for (var i = 0; i <= n; i++) {`
	`  x.push(int_rand_arithmetic_1(i+1,x[i],a,b));`
	`}`
	`x.shift();`
		$x[0] = 2^{64}$;

№ варианта	Алгоритм	Параметры

06	Арифметический оператор с синусоидой
	x[i+1)=(i*x[i])+sin(i*x[i])/(i*x[i]):
	`function int_rand_arithmetic_2(i_,xi_1,max,min) {`
	`  var i_rand = (xi_1*Math.PI/180*i_)`
	`  + Math.sin(xi_1*Math.PI/180*i_)`
	`  / (xi_1*Math.PI/180*i_);`
	`  return min +  Math.floor((max - min + 1)`
	`  * (i_rand - Math.floor(i_rand)));`
	`}`
	Вызов функции:
	`var x = new Array();`
	`x[0]=Math.pow(2,64);`
	`for (var i = 0; i <= n; i++) {`
	`  x.push(int_rand_arithmetic_2(i+1,x[i],a,b));`
	`}`
	`x.shift();`
		$x[0] = 2^{64}$;

07	Нелинейный (квадратичный) конгруэнтный метод :
	`function int_rand_NL2CM(m_,a_,b_,c_,xi_1) {`
	`  var xi=(a_*xi_1*xi_1+b_*xi_1+c_) % m_;`
	`  var i_rand=xi;`
	`  return i_rand;`
	`}`
	Вызов функции:
	`var x = new Array();`
	`var m=Math.pow(2,32);`
	`var a_=6, b_=7, c_=3;`
	`x[0]=4001;`
	`for (var i = 1; i <= n; i++) {`
	`  x.push(int_rand_NL2CM(m,a_,b_,c_,x[i-1]));`
	`}`
	`for (var i = 0; i < n; i++) {`
	`  x[i]=x[i+1];`
	`  x[i]=x[i]/m;`
	`  x[i]=a+Math.floor(x[i]*(b-a));`
	`}`
	`x.pop();`
		$m = 2^{32}$;
		a=6;
		b=7;
		c=3;
		x[0]=4001

08	Квадратичный метод Р.Ковэю (R. R. Coveyou) :
	`function int_rand_Coveyou(m_,x_i0) {`
	`  var x_i1=x_i0*(x_i0+1) % m_;`
	`  var i_rand = x_i1;`
	`  return i_rand;`
	`}`
	Вызов функции:
	`var x = new Array();`
	`var m=Math.pow(2,26);`
	`x[0]=Math.pow(2,5)-1;`
	`for (var i = 1; i <= n+1; i++) {`
	`  x.push(int_rand_Coveyou(m,x[i-1]));`
	`}`
	`for (var i = 1; i < n+1; i++) {`
	`  x[i]=x[i+1];`
	`  x[i]=x[i]/m;`
	`  x[i]=a+Math.floor(x[i]*(b-a));`
	`}`
	`x.shift();`
	`x.pop();`
		$m = 2^{26}$;
		$x[0] = 2^5 - 1$;

№ варианта	Алгоритм	Параметры

09	Аддитивный метод (предложили Грин, Смит и Клем):
	`function int_rand_Fibonachi(m_,x_i1,x_i0) {`
	`  var x_i2=(x_i1+x_i0) % m_;`
	`  var i_rand = x_i2;`
	`  return i_rand;`
	`}`
	Вызов функции:
	`var x = new Array();`
	`var lag=25;`
	`m=4096*4;`
	`x[0]=4091;x[1]=(m-5)*2;`
	`for (var i = 2; i < lag; i++) {`
	`  x[i]=int_rand_Fibonachi(m,x[i-1],x[i-2]);`
	`}`
	`for (var i = lag; i <= (n+lag); i++) {`
	`  x[i]=int_rand_Fibonachi(m,x[i-1],x[i-lag]);`
	`}`
	`for (var i = lag; i <= (n+lag); i++) {`
	`  x[i]=x[i]/m;`
	`  x[i]=a+Math.floor(x[i]*(b-a));`
	`}`
	`for (var i = 0; i < lag; i++) {`
	`x.shift();`
	`}`
	`break;`
		lag = 25;
		m = 4096 · 8;
		x[0]= 4091;
		x[1] = (m-5)· 2.

10	Мультипликативный конгруэнтный метод: см. вариант 03, но с другими значениями входных параметров	$m = 2^{31}$; c = 0; $a\_lcm = 2^{13} + 3$
11	Смешанный конгруэнтный метод: см. вариант 04, но с другими значениями входных параметров	$m = 2^{32}$; a = 1664525; c = 1013904223
12	Смешанный конгруэнтный метод: см. вариант 04, но с другими значениями входных параметров	$m = 2^{32}$; a = 69069; c = 5
13	Смешанный конгруэнтный метод: см. вариант 04, но с другими значениями входных параметров	$m = 2^{32}$; a = 1103515245; c = 12345
14	Арифметический оператор с синусоидой x[i+1)=i*x[i]*sin(i*x[i]): см. вариант 05, но с другими значением входного параметра	$x[0] = 2^{32}$;
15	Квадратичный метод Р.Ковэю (R. R. Coveyou) : см. вариант 08, но с другими значениями входных параметров	$m = 2^{22}$; $x[0] = 2^4 - 1$;
16	Аддитивный метод (предложили Грин, Смит и Клем): см. вариант 09, но с другими значениями входных параметров	Lag = 50; $m = 4096^2$; x[0]= 4091· 2-5; x[1] = m/4 - 5.
17	Арифметический оператор с синусоидой x[i+1)=(i*x[i])+sin(i*x[i])/(i*x[i]): см. вариант 06, но с другими значением входного параметра	$x[0] = 2^{32}$;

№ варианта	Алгоритм	Параметры

18	Использование сочетания встроенных методов getTime() и getMilliseconds() объекта Date.
	См. вариант 01, но необходимо изменить способ создания паузы:
	в варианте 01 пауза создаётся оператором цикла
	`for (var j = 0; j < 250000; j++) {`
	`  var temp=j*j;`
	`}`
	а в этом варианте нужно использовать метод window.setInterval(), позволяющий выполнять функцию Javascript через заданные интервалы времени.
		-

19	Использование сочетания встроенных методов getTime() и getMilliseconds() объекта Date.
	См. вариант 01, но необходимо изменить способ создания паузы:
	в варианте 01 пауза создаётся оператором цикла
	`for (var j = 0; j < 250000; j++) {`
	`  var temp=j*j;`
	`}`
	а в этом варианте нужно использовать метод window.setTimeout(), позволяющий выполнить функцию Javascript лишь по истечении заданного периода времени.
		-

3.8.3 Методика выполнения

Методически можно выделить три основных особенности, на которые необходимо обратить особое внимание при программировании заданий настоящей лабораторной работы:

методы вычислений на языке Javascript, используемых для генерации псевдослучайных чисел;

методы вывода результатов в виде последовательности числовых значений;

методы графического представления информации с использованием HTML5 Canvas API;

методы графического представления информации с использованием графической библиотеки Highcharts.

Программирование вычислений приходилось осваивать и в предыдущих лабораторных работах. Однако для программирования генераторов псевдослучайных чисел, основанных на максимально возможных и близких к ним чисел, необходимо более глубоко ознакомиться с представлением чисел в языке Javascript. Для этого, в дополнение к материалу лекций, следует усвоить материал, расположенный по адресу:

http://learn.javascript.ru/number .

Вывод результатов в виде числовых значений рекомендуется осуществлять, при необходимости, по событию (допустим по клику мышкой) в специально создаваемый для этого тэг ... . Допустим, в виде строки “номер1: значение1, номер2: значение2, ...” это сделать не сложно и не требует пояснений. Но не запрещается организовать более “красивый” вывод, допустим в виде нумерованного списка или табличных колонок. Такой вывод также можно сделать самостоятельно и вы это уже освоили в предыдущих работах. Не лишне также знать, что такой вывод эффективнее делать с помощью специальных библиотек, например, с помощью библиотеки D3, которую можно взять по адресу: http://d3js.org/ . Эту библиотеку можно рассматривать, как расширение языка Javascript, предназначенное для отображения данных в различных видах. Кстати, D3 позволяет программировать вывод не только в виде чисел, но и в графическом виде (на основе SVG). Хотя для графики лучше использовать специальную библиотеку protovis.js для Javascript, основанную на D3, которую можно взять по адресу: http://mbostock.github.io/protovis/ .

Элемент HTML5 Canvas служит основой для растровой графики. На основе HTML5 Canvas API имеются развитые библиотеки, предназначенные для графического представления получаемых результатов. В качестве такой библиотеки на настоящий момент рекомендуется Chart, которую можно взять по адресу: http://www.chartjs.org/ . Там же можно взять примеры использования для построения типовых графиков. На основе этих примеров легко самостоятельно научиться строить графики в программах на Javascript.

Однако на примере построения тест-квадрата можно сделать вывод, что для подобных построений нет необходмости использовать библиотеки, поскольку это несложно запрограммировать самостоятельно, зная лишь основы использования HTML5 Canvas API по материалу, расположенному, например, по адресам: http://habrahabr.ru/post/111308/ и http://www.pixelcom.crimea.ua/rukovodstvo-po-html5-canvas.html. При программированиии HTML5 Canvas полезно также иметь перед глазами сводку методов и свойств API по работе с этим элементом, расположенную по адресу: http://webonrails.ru/articles/html5/189/ .

Для более изощрённых методов графического отображения результатов предназначены библиотеки типа рекомендуемой для использования в настоящей лабораторной работе, а именно: графическая библиотека Highcharts. Эта библиотека основана на элементе HTML5 SVG, служащем основой для построения векторной графики, поэтому с её помощью целесообразнее делать более динамичные (масштабируемые, с более

“тонкими” анимационными эффектами).

Библиотека Highcharts по сути является расширением языка Javascript. Имея навыки построения операторов в языке Javascript, не трудно освоить навыки использования методов и свойств Highcharts. Правда, для использования Highcharts необходимо подключить ещё и библиотеку jQuery или аналогичную MooTools (популярные расширения языка Javascript), однако можно обойтись лишь минимальными средствами этих библиотек, выполняя всё необходимое средствами Highcharts и Javascript. Построение нужного типа графика не трудно освоить, подобрав нужный пример из коллекции примеров, которая загружается вместе с библиотекой. Взять библиотеку можно на сайте Highcharts: http://www.highcharts.com/download. Об использовании jQuery достаточно прочитать материал из этого источника: http://jquery.page2page.ru/ .

Освоение программирования с использованием Highcharts рекомендуется начать с повторения примеров, показанных здесь: http://troitskiy.net/2010/10/13/highcharts-javascript-biblioteka-postroeniya-grafikov/ .

Особое внимание следует обратить на возможности масштабирования (или зуммирования) выделенной части построенного графика. В работах студентов предыдущих лет, рекомендуемых в качестве образца (например, работа Яшина Вячеслава), эта возможность используется при построении графиков генерируемой числовой последовательности и коррелограммы. Выделите мышкой часть графика на коррелограмме и через некоторое время вы увидите эту часть в увеличенном масштабе. Такая возможность очень полезна для анализа результатов.

Рассмотрим помещённый ниже листинг возможного варианта кода функции для построения графика числовой последовательности.

Листинг 3.8.1.

```
/*****************************************************************************
function point_sequence() -
*****************************************************************************/
function point_sequence(x,id,x_axis_txt,y_axis_txt,title_txt,subtitle_txt) {
$(function () {
        $('#'+id).highcharts({
            chart: {
                type: 'line',
                marginRight: 60,
                marginBottom : 85,
                zoomType: 'x'
            },
            title: {
                text: title_txt
            },
            subtitle: {
                text: subtitle_txt
            },
            xAxis: {
             min: 0,
                max: x.length-1,
                title: {
                    enabled: true,
                    text: x_axis_txt
                },
                startOnTick: true,
                endOnTick: true,
                showLastLabel: true
            },
            yAxis: {
                title: {
                    text: y_axis_txt
                }
            },
            series: [{
                name: y_axis_txt,
                      marker: {
                         radius: 2,
                           states: {
                         hover: {
                                  enabled: true,
                                  radius: 4,
                                  lineColor: 'rgb(255,0,0)'
                         }
                       }
                },
                color: 'rgb(178,0,0)',
                lineColor: 'rgba(253,63,63,.4)',
                states: {
                          hover: {
                                  enabled: true,
                                  lineWidth: 1
                          }
                },
                data: x }]
        });
    });
}
```

В функции point_sequence() может быть непонятной конструкция

```
$(function () {
        $('#'+id).highcharts({
          ...
       })
}
```

Такая конструкция строится с помощью Javascript-библиотеки jQuery, обладающей собственным языком, отличающимся от "родного" языка Javascript. В функции point_sequence() только лишь приведённые выше два оператора свидетельствуют об использовании jQuery и объяснить их не составит труда по тексту ниже. Но, всё-таки, рекомендуется прочитать хотя бы минимум, например, "Введение в jQuery", здесь: http://jquery.page2page.ru/ .

Основное назначение библиотки jQuery – облегчить поиск элементов документа и манипуляции с этими элементами. В случае с HTML-документом - облегчить работу с объектами объектной модели документа (HTML-DOM). Конечно, это можно делать и средствами Javascript без jQuery, но jQuery позволяет реализовать динамический HTML намного легче.

Вы уже знаете, что графическая библиотека Highcharts реализована на основе графического формата SVG, основанного, в свою очередь на языке XML (*e**X**tensible **M**arkup **L**anguage* — расширяемый язык разметки; произносится [*икс-эм-э́ль*], подробнее см. http://ru.wikipedia.org/wiki/XML). Язык XML, также как и HTML, является языком разметки, но при помощи XML можно определять собственные наборы тегов и имена атрибутов для этих тэгов, а также программировать значения для атрибутов. Вкладывая XML-разметку внутрь HTML-разметки можно обойти ограничения HTML. Но в результате создаётся более сложная структура, работа с элементами которой посредством “чистого” Javascript существенно усложняется. Вот тут -то на помощь приходят библиотеки типа jQuery.

Итак, что означает символ $? Это сокращённая форма записи функции jQuery(), имя которой совпадает с наименованием библиотеки. Эта функция имеет очень много назначений и наиболее часто используется, поэтому для неё придумали более короткое имя. Одно из назначений этой команды – поиск HTML- элементов (и XML-элементов тоже) по различным параметрам. Например, чтобы найти div-элемент по его идентификатору id="graph_randoms" достаточно задать команду: $ ("#graph_randoms"). Это проще, чем, если бы мы использовали методы языка Javascript: document.getElementById(“graph_randoms”), document.getElementsByTagName("p") или путем перемещения по дереву объекта document с использованием отношений между узлами (см. подраздел 4.2 конспекта лекций на pvn.org.ru).

Конечно, поиск элементов, это лишь небольшая часть возможностей библиотеки jQuery. Так, приведённая выше запись из рассматриваемого примера: $ function () {$('#'+id).highcharts({...}) } , означает цепочку команд:

найти объект, соответстующий тэгу с идентификатором, хранящемся в переменной id , по команде $('#'+id);

добавить найденному объекту метод highcharts() библиотеки Highcharts,

являющийся методом с большим набором команд для XML-тэгов формата SVG (команда $('#'+id).highcharts({...})).

выполнить добавленный метод с помощью безымянной функции по команде:
$ function () {$('#'+id).highcharts({...}) } .
Вместо многоточия задаются XML-атрибуты и их значения. Атрибуты в библиотеке Highcharts описаны как соответствующие элементы графика, который необходимо построить. Эти элементы, являющиеся объектами со своими свойствами и методами, подробно описаны на сайте Highcharts: http://api.highcharts.com/highcharts.

Атрибуты задаются в виде свойств объекта highcarts, и являются, в свою очередь javascript-объектами, вкладываемыми друг в друга. Например, содержимое поля графика и содержимое поля заголовка графика (см. листинг 3.8.1) задаются с помощью двух объектов:

```
chart: {
    type: 'line',
    marginRight: 60,
    marginBottom: 85,
    zoomType: 'x'
},
title: {
    text: title_txt
},
```

где chart — объект, определяющий поле графика, которому заданы свойства и их значения, определяющие тип графика (в данном случае точки должны соединяться отрезками прямых), верхний и нижний отступы и зуммирование по оси x;

title — объект, задающий заголовок, которому задано лишь одно свойство со значением текста заголовка.

Подобным же образом заданы подзаголовок (subtitle), оси (xAxis и yAxis) и данные (series). При этом, значения свойств могут вычисляться с использованием операторов языка Javascript, а в случае, если некоторые из них не заданы, то подставляются значения по умолчанию.

3.8.4 Рекомендации по обработке и оформлению полученных результатов

Оформление html-документа с работой можно выполнить следуя примеру.

При загрузке документ может иметь следующий вид:

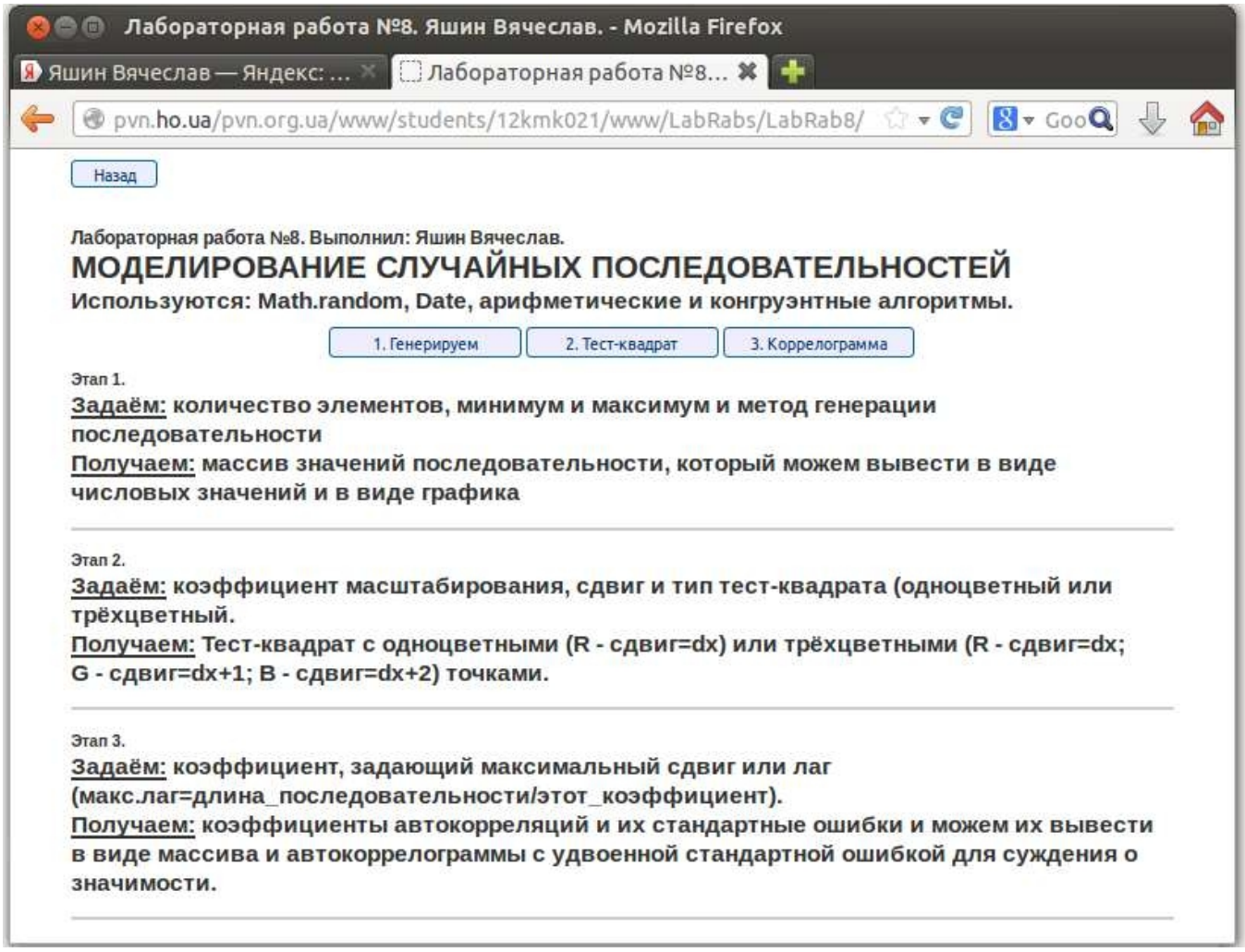


Рис.3.8.1 Образец оформления первоначального вида работы

С помощью каждого из трёх пунктов меню можно показать соответствующую форму, используя которую, можно в начале сгенерировать последовательность и вывести её в числовом и графическом видах, затем выести тест-квадрат, и, наконец, рассчитать и вывести автокоррелограмму.

По меню «Генерируем» выводится форма, показанная на рис. 3.8.2.

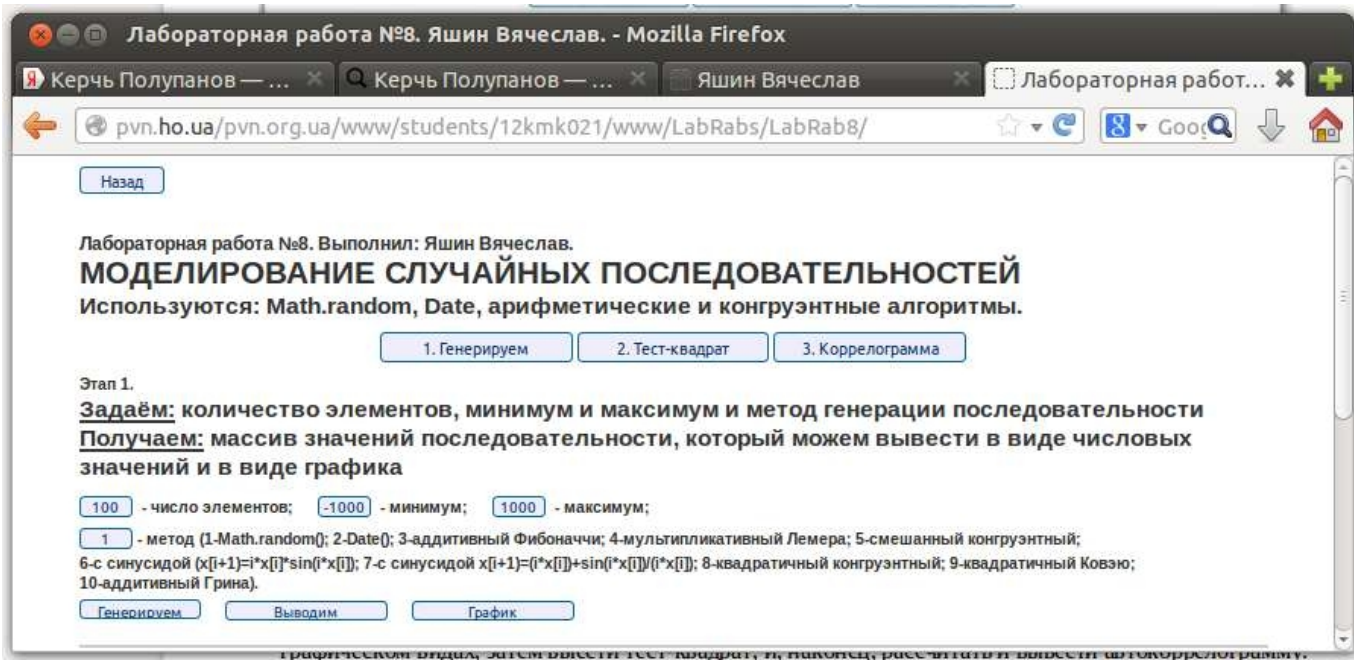


Рис.3.8.2 Образец формы для генерации последовательности

В этом примере предусмотрено несколько вариантов генераторов псевдослучайных чисел, а вам достаточно запрограммировать лишь по одному варианту в соответствии с табл.3.8.1.

По меню «Тест-квадрат» выводится форма, показанная на рис. 3.8.3.

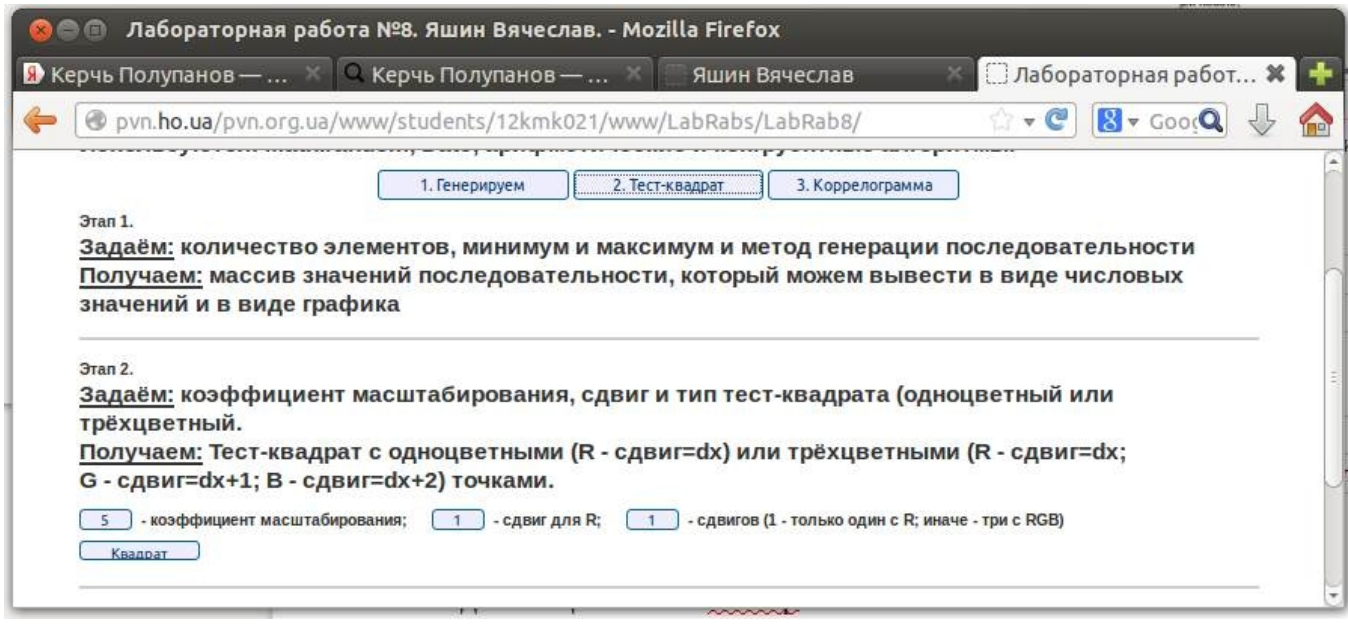


Рис.3.8.3 Образец формы для отображения тест-квадрата

По меню «Коррелограмма» выводится форма, показанная на рис. 3.8.4.

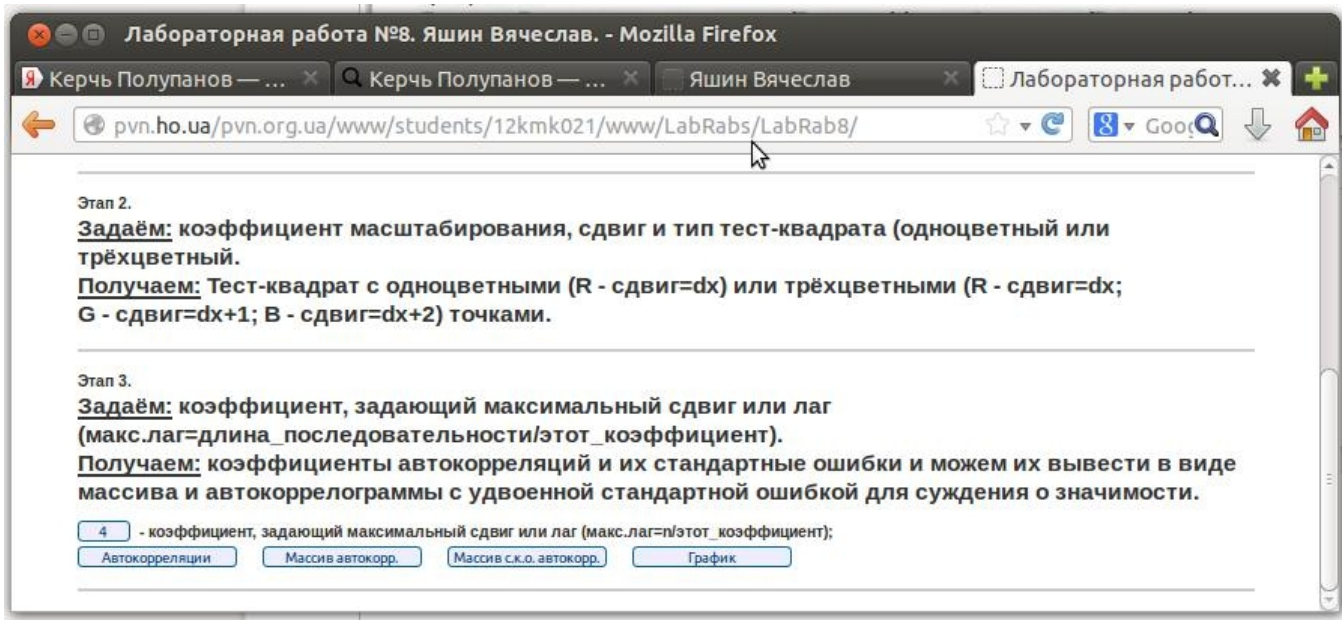


Рис.3.8.4 Образец формы для расчёта и отображения автокоррелограммы

На рис.3.8.5 приведён пример сгенерированной псевдослучайной последовательности в числовом и графическом видах, полученный в результате последовательного нажатия на кнопки формы, показанной на рис.3.8.2 (при длине последовательности $n = 80$).

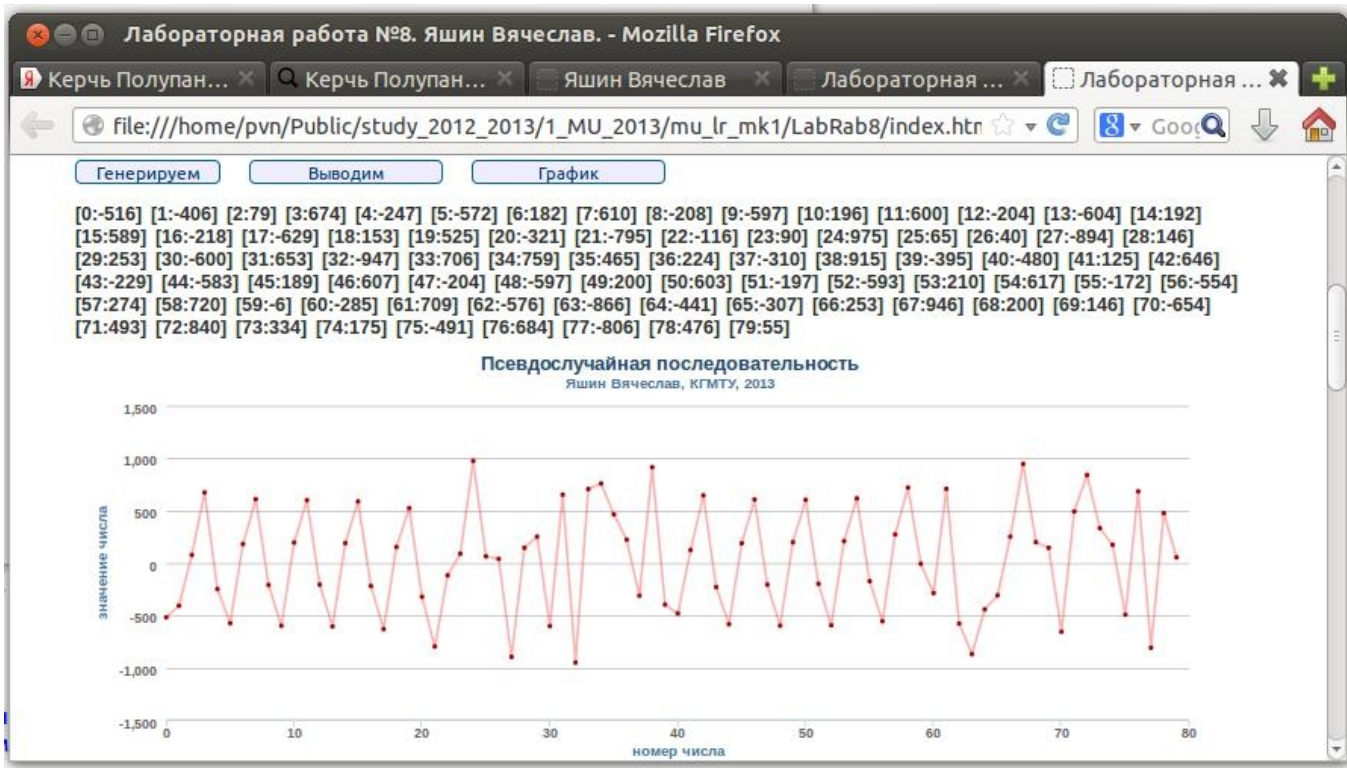
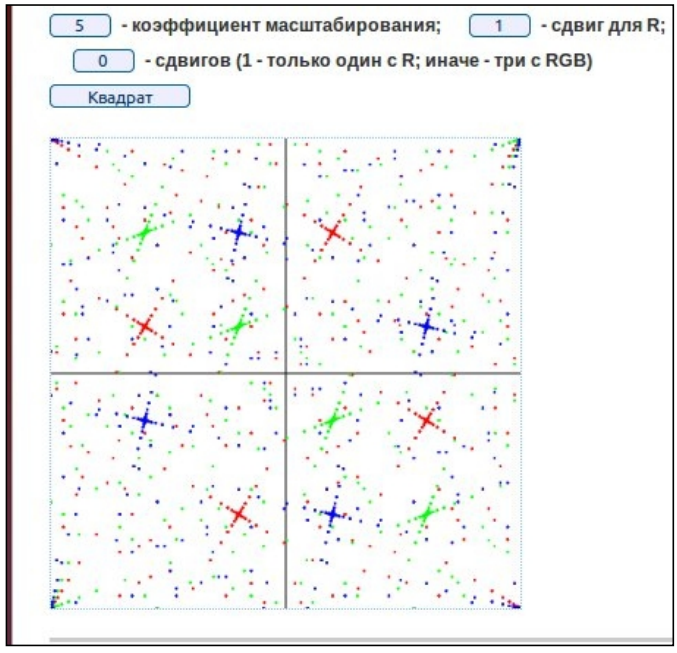


Рис.3.8.5 Образец вывода псевдослучайной последовательности

Вывод в числовом виде нецелесообразно делать при большом заданном числе

элементов последовательности (длине последовательности).

Образец тест-квадрата и автокорреллограммы для при n=2400 для одного из методов генерации псевдослучайных чисел показан на рис. 3.8.6 и 3.8.7.



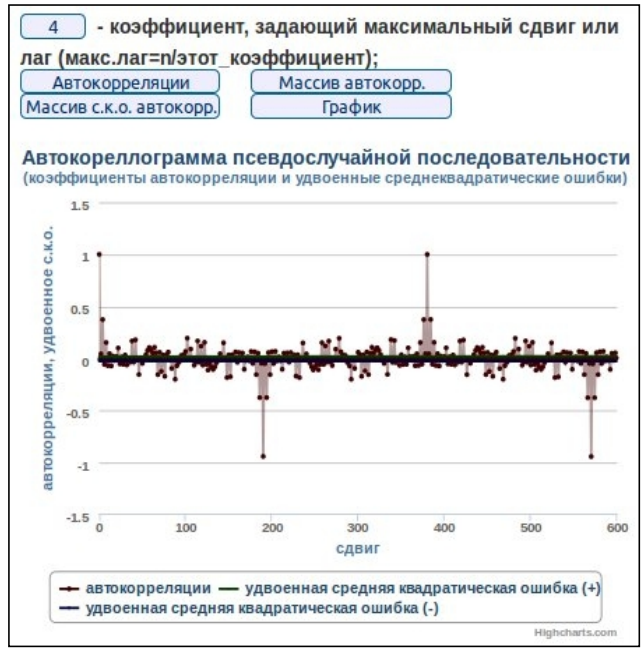


Рис.3.8.6 Образец тест-квадрата

Рис.3.8.7 Образец автокоррелограммы

На рис.3.8.6 приведён образец тест-квадрата, рассчитанного для трёх сдвигов. Скопление точек в квадрате в виде фигур различной формы свидетельствует о наличии взаимосвязей между числами последовательности, чего не должно быть для случайных чисел. Можно сделать вывод о неудовлетворительном качестве данного генератора ППСП.

Ещё более отчётливо это видно при рассмотрении автокоррелограммы на рис.3.8.7, на которой видны всплески примерно через каждые 200 чисел, свидетельствующие о явно выраженной периодичности в последовательности. На этом рисунке не показан вывод числовых значений коэффициентов автокорреляции автокоррелограммы и их среднеквадратических ошибок из-за слишком большого количества, хотя такая возможность предусмотрена согласно условию задания (имеются кнопки «Массив автокорр.» и «Массив с.к.о. автокорр.»).

Впрочем, при необходимости, числовые значения можно просмотреть и на графике. Такая возможность легко может быть предусмотрена при программировании графика с помощью соответствующих методов, имеющихся в библиотеке Highcharts. Кроме этого, отдельные участки графика можно просмотреть в увеличенном виде, используя «zoom». Для того, чтобы вспомнить программирование этих методов библиотеки Highcharts, рекомендуется просмотреть соответствующие примеры, входящие в набор типовых

примеров при скачивании библиотеки с сайта Highcharts. Примеры увеличенного участка автокоррелограммы, показанной на рис. 3.8.7, и вывода числового значения коэффициента автокорреляции при подведении к нужной точке, показаны на рис. 3.8.8 и 3.8.9.

Рис.3.8.8 Использование метода «zoom»

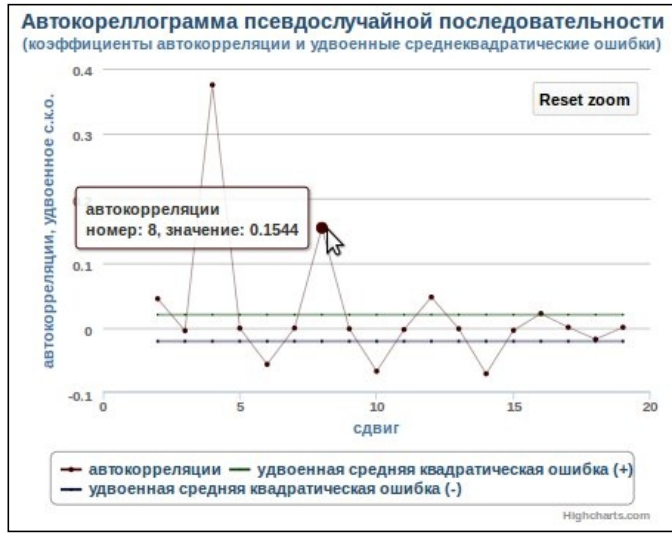


Рис.3.8.9 Показ числового значения

Возможность рассмотрения особенностей графика в увеличенном виде чрезвычайно полезна для анализа. Так, на увеличенном фрагменте автокоррелограммы явно прослеживается значимая периодичность с периодом в пять значений, что трудно рассмотреть без увеличения на рис. 3.8.7.

В заключение следует напомнить, что все вычисления и построения должны выводиться в виде, наиболее удобном для анализа, в данном случае, - псевдослучайной последовательности на предмет её случайности. Возможности продемонстрированного примера отвечают этому требованию. В частности, об используемом в примере генераторе ППСП можно сказать, что данный генератор выдаёт последовательности не отвечающие требованиям, при которых их можно считать случайными.

Вопросы для самоконтроля

1. Количество бит для представления чисел в Javascript?
2. Диапазон целых чисел в Javascript?
3. Диапазон целых чисел для побитовых операций в Javascript?
4. Какое число содержится в Number.MAX_VALUES?
5. Функция (метод) для вычисления случайного числа в языке Javascript»?
6. Способы округления чисел с плавающей точкой в языке Javascript?
7. Оператор для округления вещественных чисел с необходимой точностью с помощью методов Math.round и Math.float?

8. Округление вещественных чисел с заданной точностью с использованием метода toFixed объекта Number?
9. Оператор для задания равномерно распределенного случайного числа на заданном интервале с использованием метода random объекта Math?
10. В чем заключается конгруэнтный метод генерации равномерно распределенных случайных чисел?
11. Что такое период псевдослучайной последовательности?
12. Принцип построения тестового квадрата, используемого для визуального анализа качества псевдослучайной последовательности?
13. Что такое «API» на примере HTML5 Canvas API?
14. Нужно ли подключение специальных библиотек при использовании HTML5 Canvas в современных браузерах при построении тестового квадрата?
15. В каком html-тэге отображаются пиксели элемента HTML5 Canvas?
16. Нужно ли подключение специальных библиотек при использовании библиотеки HighCharts для построения графиков на языке Javascript?
17. В каком html-тэге обычно отображаются графики при использовании библиотеки HighCharts?
18. Принцип построения автокоррелограммы?
19. Как отображается наличие периодичности на автокоррелограмме, рассчитанной по значениям псевдослучайной равномерной последовательности?
20. Назначение библиотеки jQuery?
21. Чем вызвана необходимость подключения библиотеки jQuery при использовании графической библиотеки HighCharts?