

Лабораторная работа 3

Создано системой Doxygen 1.13.2

1	readme	1
2	Иерархический список классов	3
2.1	Иерархия классов	3
3	Алфавитный указатель классов	5
3.1	Классы	5
4	Список файлов	7
4.1	Файлы	7
5	Классы	9
5.1	Класс div_PRNG	9
5.1.1	Подробное описание	10
5.1.2	Конструктор(ы)	10
5.1.2.1	div_PRNG()	10
5.1.3	Методы	11
5.1.3.1	generate()	11
5.1.4	Данные класса	11
5.1.4.1	prev	11
5.1.4.2	seed	11
5.2	Класс LCG	12
5.2.1	Подробное описание	13
5.2.2	Конструктор(ы)	13
5.2.2.1	LCG()	13
5.2.3	Методы	13
5.2.3.1	generate()	13
5.2.4	Данные класса	14
5.2.4.1	b	14
5.2.4.2	k	14
5.2.4.3	M	14
5.2.4.4	seed	14
5.3	Класс mid_xor_PRNG	14
5.3.1	Подробное описание	15
5.3.2	Конструктор(ы)	15
5.3.2.1	mid_xor_PRNG()	15
5.3.3	Методы	16
5.3.3.1	generate()	16
5.3.4	Данные класса	16
5.3.4.1	seed	16
5.4	Класс mul_xor_PRNG	17
5.4.1	Подробное описание	18
5.4.2	Конструктор(ы)	18
5.4.2.1	mul_xor_PRNG()	18
5.4.3	Методы	18

5.4.3.1 generate()	18
5.4.4 Данные класса	19
5.4.4.1 seed	19
5.5 Класс PRNG	19
5.5.1 Подробное описание	20
5.5.2 Конструктор(ы)	20
5.5.2.1 PRNG()	20
5.5.3 Методы	20
5.5.3.1 generate()	20
5.5.3.2 generate_sample()	20
5.5.3.3 result()	20
5.5.4 Данные класса	21
5.5.4.1 max_lim	21
5.5.4.2 min_lim	21
6 Файлы	23
6.1 Файл create_samples.cpp	23
6.1.1 Подробное описание	23
6.1.2 Функции	24
6.1.2.1 create_samples()	24
6.1.2.2 main()	25
6.2 create_samples.cpp	25
6.3 Файл get_results.cpp	26
6.3.1 Подробное описание	27
6.3.2 Функции	27
6.3.2.1 main()	27
6.3.2.2 process_sample()	29
6.4 get_results.cpp	29
6.5 Файл include/defs.h	32
6.5.1 Подробное описание	32
6.5.2 Макросы	32
6.5.2.1 ALPHA	32
6.5.2.2 AP_ENTROPY_M	32
6.5.2.3 MAX_LIM	32
6.5.2.4 MIN_LIM	33
6.5.2.5 SAMPLES_DIR	33
6.5.2.6 SERIAL_M	33
6.6 defs.h	33
6.7 cephes.h	33
6.8 Файл include/prng.h	34
6.8.1 Подробное описание	34
6.9 prng.h	34
6.10 Файл include/tests.h	35

6.10.1 Подробное описание	36
6.10.2 Функции	36
6.10.2.1 get_apEn()	36
6.10.2.2 get_chi2()	37
6.10.2.3 get_cv()	38
6.10.2.4 get_mean()	39
6.10.2.5 get_pattern_counts()	39
6.10.2.6 get_psi2()	40
6.10.2.7 get_stdDev()	40
6.10.2.8 nist_apEntropy()	41
6.10.2.9 nist_cusum()	41
6.10.2.10 nist_frequency()	42
6.10.2.11 nist_runs()	42
6.10.2.12 nist_serial()	43
6.10.2.13 sample_to_bit_sequence()	43
6.11 tests.h	44
6.12 cephes.cpp	44
6.13 Файл src/nist_funcs.cpp	48
6.13.1 Подробное описание	49
6.13.2 Функции	49
6.13.2.1 get_apEn()	49
6.13.2.2 get_pattern_counts()	49
6.13.2.3 get_psi2()	50
6.13.2.4 sample_to_bit_sequence()	51
6.14 nist_funcs.cpp	51
6.15 Файл src/nist_tests.cpp	52
6.15.1 Подробное описание	53
6.15.2 Функции	53
6.15.2.1 nist_apEntropy()	53
6.15.2.2 nist_cusum()	54
6.15.2.3 nist_frequency()	55
6.15.2.4 nist_runs()	55
6.15.2.5 nist_serial()	56
6.16 nist_tests.cpp	57
6.17 Файл src/prng.cpp	58
6.17.1 Подробное описание	58
6.18 prng.cpp	58
6.19 Файл src/tests.cpp	60
6.19.1 Подробное описание	60
6.19.2 Функции	60
6.19.2.1 get_chi2()	60
6.19.2.2 get_cv()	61
6.19.2.3 get_mean()	61

6.19.2.4 <code>get_stdDev()</code>	62
6.20 <code>tests.cpp</code>	62
6.21 <code>test.cpp</code>	63
6.22 Файл <code>time_check.cpp</code>	64
6.22.1 Подробное описание	64
6.22.2 Функции	64
6.22.2.1 <code>check_time()</code>	64
6.22.2.2 <code>main()</code>	65
6.23 <code>time_check.cpp</code>	66
6.24 <code>time_graph.py</code>	67
Предметный указатель	69

Глава 1

readme

Подсказка к возможностям makefile:

- `make samples` - создаёт папку `samples` с наборами сгенерированных выборок разных размеров для каждого генератора
- `make results` - в каждом вышеуказанном наборе создаёт файл `.csv` содежращий все результаты всех измерений и тестов
- `make time` - создаёт файл `generation_time.csv` содержащий время генерации выборок разных объёмов для разных генераторов
- `make graph` - создаёт файл `generation_time.png` содежращий графики к вышеуказанному файлу
- `make doc` - конструирует `html` и `latex doxygen` документацию
- `make pdf` - запускает `make doc`, а затем собирает `pdf` версий `latex` документации

- `make clean` - удаляет созданные объектные и исполняемые файлы
- `make cleanall` - запускает `make clean` удаляет папки `samples`, `html` и `latex` со всем содержимым, а также файлы `generation_time.csv` и `generation_time.png` (потребуется подтверждение для удаления)
- `make` или `make all` запустит подряд все вышеуказанные команды, кроме `clean`, `cleanall`, `doc` и `pdf`

Глава 2

Иерархический список классов

2.1 Иерархия классов

Иерархия классов.

PRNG	19
LCG	12
div_PRNG	9
mid_xor_PRNG	14
mul_xor_PRNG	17

Глава 3

Алфавитный указатель классов

3.1 Классы

Классы с их кратким описанием.

div_PRNG	Генератор, основанный на делении чисел друг на друга	9
LCG	Генератор, основанный на линейном конгруэнтном методе	12
mid_xor_PRNG	Генератор, основанный на среднем квадрата и Хог	14
mul_xor_PRNG	Генератор, основанный на произведении и Хог	17
PRNG	Абстрактный базовый класс генераторов	19

Глава 4

Список файлов

4.1 Файлы

Полный список документированных файлов.

create_samples.cpp	23
Файл, предназначенный для генерации выборок разными генераторами	
get_results.cpp	26
Файл для вычисления измерений и проведения тестов над выборками	
test.cpp	63
time_check.cpp	
Файл, в котором производится измерение времени генерации выборок разных раз- меров разными генераторами и одним из стандартных генераторов	64
time_graph.py	67
include/defs.h	
Заголовочный файл, содержащий определения, используемые во всей лаборатор- ной работе	32
include/prng.h	
Заголовочный файл, в котором определяются классы генераторов	34
include/tests.h	
Заголовочный файл, содержащий объявления всех функций для вычисления зна- чений и проведения тестов	35
include/external/cephes.h	33
src/nist_funcs.cpp	
Файл, содержащий реализации вспомогательных функций для nist-тестов	48
src/nist_tests.cpp	
Файл, содержащий реализации 5 nist-тестов	52
src/prng.cpp	
Файл, с определением методов классов генераторов	58
src/tests.cpp	
Файл, в котором определяются все функции, производящие оценки выборок и хи- квадрат тест	60
src/external/cephes.cpp	44

Глава 5

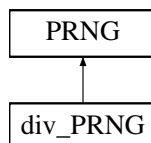
Классы

5.1 Класс div_PRNG

Генератор, основанный на делении чисел друг на друга

```
#include <prng.h>
```

Граф наследования:div_PRNG:



Открытые члены

- `div_PRNG ()=delete`
Запрет на конструктор по умолчанию
- `div_PRNG (uint32_t, uint32_t=0, uint32_t=-1)`
- `uint32_t generate () override`
Генерация следующего числа

Открытые члены унаследованные от `PRNG`

- `void generate_sample (std::ostream &, int)`
Метод для генерации и записи в поток целой выборки

Закрытые данные

- `uint32_t seed`
Семя генерации
- `uint32_t prev`
Предыдущий результат

Дополнительные унаследованные члены

Защищенные члены унаследованные от [PRNG](#)

- [PRNG](#) (uint32_t, uint32_t)
Конструктор для базового абстрактного класса
- uint32_t [result](#) (uint32_t) const
Метод для перевода сгенерированного числа в требуемый диапазон

Защищенные данные унаследованные от [PRNG](#)

- const uint32_t [min_lim](#)
Нижняя граница генерации
- const uint32_t [max_lim](#)
Верхняя граница генерации

5.1.1 Подробное описание

Генератор, основанный на делении чисел друг на друга

См. определение в файле [prng.h](#) строка [86](#)

5.1.2 Конструктор(ы)

5.1.2.1 div_PRNG()

```
div_PRNG::div_PRNG (
    uint32_t seed,
    uint32_t min_lim = 0,
    uint32_t max_lim = -1)
```

Аргументы

seed	Семя генерации
min_lim	Нижняя граница генерации
max_lim	Верхняя граница генерации

См. определение в файле [prng.cpp](#) строка [143](#)

```
00144 : PRNG(min_lim, max_lim), seed(seed), prev(0x9AE77B3D) {}
```


5.1.3 Методы

5.1.3.1 generate()

uint32_t div_PRNG::generate () [override], [virtual]

Генерация следующего числа

Возвращает

Следующее случайное число

Замещает [PRNG](#).

См. определение в файле [prng.cpp](#) строка 149

```
00150 {
00151     // Определяем меньшее и большее значения между seed и prev
00152     uint32_t min = (prev > seed) ? seed : prev;
00153     uint32_t max = (min != seed) ? seed : prev;
00154
00155     bool flag = (seed == min); // флаг, чтобы понимать, какие изменения проводятся над переменными
00156
00157     // Изменения, чтобы не возникло проблем с делением
00158     if ( min == UINT32_MAX ) min = 0x9AE77B3D;
00159     if ( min == max )
00160         max = (((unsigned long)max + min) > UINT32_MAX) ? UINT32_MAX : max + min;
00161
00162
00163     double div = double(min) / max; // Число в диапазоне (0,1)
00164
00165     seed = div * (1ul << 48); // Извлекаем 32 бита после запятой, начиная с 16 бита (первые 16 битов обрубаются за счёт
    типа seed)
00166
00167     if ( seed == 0 ) seed = 0x9AE77B3D; // Избавляемся от выражения в 0
00168     prev = (flag ? min : max); // Записываем предыдущее значение seed, включая преобразования
00169
00170     return result(seed);
00171 }
```

5.1.4 Данные класса

5.1.4.1 prev

uint32_t div_PRNG::prev [private]

Предыдущий результат

См. определение в файле [prng.h](#) строка 89

5.1.4.2 seed

uint32_t div_PRNG::seed [private]

Семя генерации

См. определение в файле [prng.h](#) строка 88

Объявления и описания членов классов находятся в файлах:

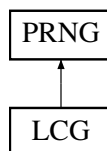
- [include/prng.h](#)
- [src/prng.cpp](#)

5.2 Класс LCG

Генератор, основанный на линейном конгруэнтном методе

```
#include <prng.h>
```

Граф наследования: LCG:



Открытые члены

- `LCG ()=delete`
Запрет на конструктор по умолчанию
- `LCG (uint32_t, uint32_t=0, uint32_t=-1)`
- `uint32_t generate () override`
Генерация следующего числа

Открытые члены унаследованные от PRNG

- `void generate_sample (std::ostream &, int)`
Метод для генерации и записи в поток целой выборки

Закрытые данные

- `uint32_t seed`
Семя генерации
- `uint32_t k`
Коэффициент
- `uint32_t b`
Смещение
- `uint32_t M`
Модуль

Дополнительные унаследованные члены

Защищенные члены унаследованные от PRNG

- `PRNG (uint32_t, uint32_t)`
Конструктор для базового абстрактного класса
- `uint32_t result (uint32_t) const`
Метод для перевода сгенерированного числа в требуемый диапазон

Защищенные данные унаследованные от PRNG

- `const uint32_t min_lim`
Нижняя граница генерации
- `const uint32_t max_lim`
Верхняя граница генерации

5.2.1 Подробное описание

Генератор, основанный на линейном конгруэнтном методе

См. определение в файле `prng.h` строка 68

5.2.2 Конструктор(ы)

5.2.2.1 LCG()

```
LCG::LCG (
    uint32_t seed,
    uint32_t min_lim = 0,
    uint32_t max_lim = -1)
```

Аргументы

seed	Семя генерации
min_lim	Нижняя граница генерации
max_lim	Верхняя граница генерации

См. определение в файле `prng.cpp` строка 110

```
00112 : PRNG(min_lim, max_lim), seed(seed), k(1'103'515'245), b(12345), M(1«31) {}
```

5.2.3 Методы

5.2.3.1 generate()

```
uint32_t LCG::generate () [override], [virtual]
```

Генерация следующего числа

Возвращает

Следующее случайное число

Замещает PRNG.

См. определение в файле `prng.cpp` строка 117

```
00118 {
00119     seed = (seed * k + b) & (M-1); // Если M = 2^N
00120     // seed = (seed * k + b) % (M); // иначе
00121
00122     // (k-1) - делится на все простые делители M (2)
00123     // b и M взаимно простые
00124     // M делится на 4 и (k-1) делится на 4
00125     //
00126     // Поэтому у этого линейного конгруэнтного генератора максимальная периодичность
00127
00128     return result(seed);
00129 }
```

5.2.4 Данные класса

5.2.4.1 b

```
uint32_t LCG::b [private]
```

Смещение

См. определение в файле [prng.h](#) строка 72

5.2.4.2 k

```
uint32_t LCG::k [private]
```

Коэффициент

См. определение в файле [prng.h](#) строка 71

5.2.4.3 M

```
uint32_t LCG::M [private]
```

Модуль

См. определение в файле [prng.h](#) строка 73

5.2.4.4 seed

```
uint32_t LCG::seed [private]
```

Семя генерации

См. определение в файле [prng.h](#) строка 70

Объявления и описания членов классов находятся в файлах:

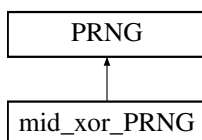
- [include/prng.h](#)
- [src/prng.cpp](#)

5.3 Класс mid_xor_PRNG

Генератор, основанный на среднем квадрата и Xor.

```
#include <prng.h>
```

Граф наследования:mid_xor_PRNG:



Открытые члены

- `mid_xor_PRNG ()=delete`
Запрет на конструктор по умолчанию
- `mid_xor_PRNG (uint32_t, uint32_t=0, uint32_t=-1)`
Конструктор класса `mid_xor_PRNG`.
- `uint32_t generate () override`
Генерация следующего числа

Открытые члены унаследованные от `PRNG`

- `void generate_sample (std::ostream &, int)`
Метод для генерации и записи в поток целой выборки

Закрытые данные

- `uint32_t seed`
Семя генерации

Дополнительные унаследованные члены

Защищенные члены унаследованные от `PRNG`

- `PRNG (uint32_t, uint32_t)`
Конструктор для базового абстрактного класса
- `uint32_t result (uint32_t) const`
Метод для перевода сгенерированного числа в требуемый диапазон

Защищенные данные унаследованные от `PRNG`

- `const uint32_t min_lim`
Нижняя граница генерации
- `const uint32_t max_lim`
Верхняя граница генерации

5.3.1 Подробное описание

Генератор, основанный на среднем квадрата и Xor.

См. определение в файле `prng.h` строка 40

5.3.2 Конструктор(ы)

5.3.2.1 `mid_xor_PRNG()`

```
mid_xor_PRNG::mid_xor_PRNG (
    uint32_t seed,
    uint32_t min_lim = 0,
    uint32_t max_lim = -1)
```

Конструктор класса `mid_xor_PRNG`.

Аргументы

seed	Семя генерации
min_lim	Нижняя граница генерации
max_lim	Верхняя граница генерации

См. определение в файле [prng.cpp](#) строка 55

```
00056 : PRNG(min_lim, max_lim), seed(seed) {}
```

5.3.3 Методы

5.3.3.1 generate()

```
uint32_t mid_xor_PRNG::generate () [override], [virtual]
```

Генерация следующего числа

Возвращает

Следующее случайное число

Замещает [PRNG](#).

См. определение в файле [prng.cpp](#) строка 61

```
00062 {
00063     uint64_t product = seed * seed;
00064
00065     seed = (product » 8) & 0xFFFFFFFF;
00066
00067     // Динамический сдвиг для XorShift
00068     uint8_t shift = (seed » 10) & 0x1F; // Сдвиг от 0 до 31
00069     seed ^= 0x9E3779B9 « (shift % 13 + 1); // Чтобы не было вырождения в 0
00070     seed ^= seed » (shift % 17 + 1);
00071
00072     return result(seed);
00073 }
```

5.3.4 Данные класса

5.3.4.1 seed

```
uint32_t mid_xor_PRNG::seed [private]
```

Семя генерации

См. определение в файле [prng.h](#) строка 42

Объявления и описания членов классов находятся в файлах:

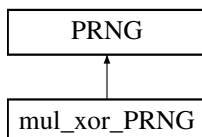
- [include/prng.h](#)
- [src/prng.cpp](#)

5.4 Класс mul_xor_PRNG

Генератор, основанный на произведении и Xor.

```
#include <prng.h>
```

Граф наследования:mul_xor_PRNG:



Открытые члены

- `mul_xor_PRNG ()=delete`
Запрет на конструктор по умолчанию
- `mul_xor_PRNG (uint32_t, uint32_t=0, uint32_t=-1)`
- `uint32_t generate () override`
Генерация следующего числа

Открытые члены унаследованные от `PRNG`

- `void generate_sample (std::ostream &, int)`
Метод для генерации и записи в поток целой выборки

Закрытые данные

- `uint32_t seed`
Семя генерации

Дополнительные унаследованные члены

Защищенные члены унаследованные от `PRNG`

- `PRNG (uint32_t, uint32_t)`
Конструктор для базового абстрактного класса
- `uint32_t result (uint32_t) const`
Метод для перевода сгенерированного числа в требуемый диапазон

Защищенные данные унаследованные от `PRNG`

- `const uint32_t min_lim`
Нижняя граница генерации
- `const uint32_t max_lim`
Верхняя граница генерации

5.4.1 Подробное описание

Генератор, основанный на произведении и Хор.

См. определение в файле [prng.h](#) строка 54

5.4.2 Конструктор(ы)

5.4.2.1 mul_xor_PRNG()

```
mul_xor_PRNG::mul_xor_PRNG (
    uint32_t seed,
    uint32_t min_lim = 0,
    uint32_t max_lim = -1)
```

Аргументы

seed	Семя генерации
min_lim	Нижняя граница генерации
max_lim	Верхняя граница генерации

См. определение в файле [prng.cpp](#) строка 85

```
00086 : PRNG(min_lim, max_lim), seed(seed) {}
```

5.4.3 Методы

5.4.3.1 generate()

```
uint32_t mul_xor_PRNG::generate () [override], [virtual]
```

Генерация следующего числа

Возвращает

Следующее случайное число

Замещает [PRNG](#).

См. определение в файле [prng.cpp](#) строка 91

```
00092 {
00093     seed ^= seed « 13;
00094     seed ^= (seed » 7) * 0x9AE77B3D;
00095     seed ^= (seed » 5) | (seed « 17);
00096
00097     return result(seed);
00098 }
```


5.4.4 Данные класса

5.4.4.1 seed

```
uint32_t mul_xor_PRNG::seed [private]
```

Семя генерации

См. определение в файле [prng.h](#) строка 56

Объявления и описания членов классов находятся в файлах:

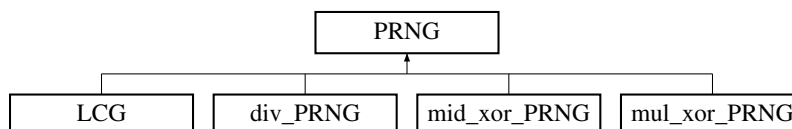
- [include/prng.h](#)
- [src/prng.cpp](#)

5.5 Класс PRNG

Абстрактный базовый класс генераторов

```
#include <prng.h>
```

Граф наследования:PRNG:



Открытые члены

- `void generate_sample (std::ostream &, int)`
Метод для генерации и записи в поток целой выборки

Защищенные члены

- `PRNG (uint32_t, uint32_t)`
Конструктор для базового абстрактного класса
- `virtual uint32_t generate ()=0`
Чистая функция для генерации следующего числа
- `uint32_t result (uint32_t) const`
Метод для перевода сгенерированного числа в требуемый диапазон

Защищенные данные

- `const uint32_t min_lim`
Нижняя граница генерации
- `const uint32_t max_lim`
Верхняя граница генерации

5.5.1 Подробное описание

Абстрактный базовый класс генераторов

См. определение в файле [prng.h](#) строка 19

5.5.2 Конструктор(ы)

5.5.2.1 PRNG()

```
PRNG::PRNG (
    uint32_t min_lim,
    uint32_t max_lim) [protected]
```

Конструктор для базового абстрактного класса

Аргументы

min_lim	Нижняя граница генерации
max_lim	Верхняя граница генерации

См. определение в файле [prng.cpp](#) строка 19

```
00020 : min_lim(min_lim), max_lim(max_lim) {}
```

5.5.3 Методы

5.5.3.1 generate()

```
virtual uint32_t PRNG::generate () [protected], [pure virtual]
```

Чистая функция для генерации следующего числа

Замещается в [div_PRNG](#), [LCG](#), [mid_xor_PRNG](#) и [mul_xor_PRNG](#).

5.5.3.2 generate_sample()

```
void PRNG::generate_sample (
    std::ostream & out,
    int size)
```

Метод для генерации и записи в поток целой выборки

Аргументы

out	Поток для записи
size	Размер генерируемой выборки

См. определение в файле [prng.cpp](#) строка 26

```
00027 {
00028     for (int i=0; i < size; ++i)
00029     {
00030         out << this->generate() << std::endl;
00031     }
00032 }
```

5.5.3.3 result()

```
uint32_t PRNG::result (
    uint32_t result) const [protected]
```

Метод для перевода сгенерированного числа в требуемый диапазон

Аргументы

result	Сгенерированное число
--------	-----------------------

Возвращает

Сгенерированное число в требуемом диапазоне

См. определение в файле [prng.cpp](#) строка 38

```
00039 {  
00040     return min_lim + result % (1ul + max_lim - min_lim); // поскольку max_lim - верхняя граница включительно, то  
                        нужно брать модуль +1  
00041                                     // (1 - unsigned long, чтобы не возникло переполнения и не было деления на 0)  
00042 }
```

5.5.4 Данные класса

5.5.4.1 max_lim

```
const uint32_t PRNG::max_lim [protected]
```

Верхняя граница генерации

См. определение в файле [prng.h](#) строка 24

5.5.4.2 min_lim

```
const uint32_t PRNG::min_lim [protected]
```

Нижняя граница генерации

См. определение в файле [prng.h](#) строка 23

Объявления и описания членов классов находятся в файлах:

- [include/prng.h](#)
- [src/prng.cpp](#)

Глава 6

Файлы

6.1 Файл `create_samples.cpp`

Файл, предназначенный для генерации выборок разными генераторами

```
#include <filesystem>
#include <fstream>
#include <iostream>
#include <string>
#include <vector>
#include "include/prng.h"
#include "include/defs.h"
```

Функции

- `template<class PRNG>`
`void create_samples` (`std::string path`, `std::string prng_name`, `std::vector< int > &sizes`, `std::vector< int > &seeds`, `uint32_t min_lim`, `uint32_t max_lim`)
Функция для генерации выборок для одного из генераторов
- `int main ()`
Основная функция, в которой происходит генерация выборок всеми генераторами

6.1.1 Подробное описание

Файл, предназначенный для генерации выборок разными генераторами

В этом файле генерируются выборки с помощью различных генераторов и записываются в соответствующие папки в папке `./samples`

См. определение в файле `create_samples.cpp`

6.1.2 Функции

6.1.2.1 create_samples()

```
template<class PRNG>
void create_samples (
    std::string path,
    std::string prng_name,
    std::vector< int > & sizes,
    std::vector< int > & seeds,
    uint32_t min_lim,
    uint32_t max_lim)
```

Функция для генерации выборок для одного из генераторов

Параметры шаблона

PRNG	Класс используемого генератора. Наследник класса PRNG
----------------------	---

Аргументы

path	Путь в котором будет создана папка с выборками
prng_name	Название генератора
sizes	Список размеров выборок
seeds	Список семян для генерации выборок. Размер списка должен делить количество размеров выборок
min_lim	Нижняя граница генерации
max_lim	Верхняя граница генерации

См. определение в файле [create_samples.cpp](#) строка 73

```
00076 {
00077
00078
00079     int mod;
00080
00081     // Проверка корректности размеров списков seeds и sizes
00082     if (sizes.size() == seeds.size())    mod = sizes.size();
00083     else if (sizes.size() % seeds.size() == 0) mod = sizes.size() / seeds.size();
00084     else
00085     {
00086         std::cerr << "! Неправильные размерности набора размеров и семян генераторов." << std::endl;
00087         return;
00088     }
00089
00090
00091     std::cout << "Generating samples by " + prng_name << std::endl;
00092
00093     // Создание папки для выборок
00094     fs::create_directory(path + prng_name);
00095
00096     // Цикл создания выборок
00097     for (int i=0; i < sizes.size(); ++i)
00098     {
00099
00100         // Создать файл выборки
00101         std::ofstream sample_file(path + prng_name + "/" + prng_name + "_" + std::to_string(i+1) + ".txt");
00102
00103         // Создать генератор с заданным семенем и границами
00104         PRNG prng (seeds.at(i % mod), min_lim, max_lim);
00105
00106         // Создать и записать выборку
00107         prng.generate_sample(sample_file, sizes.at(i));
00108
00109         // Закрывать файл выборки
00110         sample_file.close();
```

```

00111
00112     std::cout << "> Complete " + prng_name + " _" + std::to_string(i+1) + ".txt" << std::endl;
00113 }
00114
00115     std::endl(std::cout);
00116 }

```

6.1.2.2 main()

```
int main ()
```

Основная функция, в которой происходит генерация выборок всеми генераторами

См. определение в файле [create_samples.cpp](#) строка 31

```

00032 {
00033     std::ofstream sample_file;
00034
00035     // Создать папку для выборок, если её нет
00036     fs::create_directory(SAMPLES_DIR);
00037
00038
00039     // Размеры выборок
00040     std::vector<int> sizes = { 1000, 1000, 1000, 1000,
00041                             2000, 2000, 2000, 2000,
00042                             3000, 3000, 3000, 3000,
00043                             5000, 5000, 5000, 5000,
00044                             10000, 10000, 10000, 10000};
00045
00046     // Семена генераторов
00047     std::vector<int> seeds = { 42, 13973739, 323159976, 518977272,
00048                             667961784, 958390147, 567454690, 292832249,
00049                             583553826, 722672343, 621406124, 771825685,
00050                             398112944, 209114256, 242948276, 826041245,
00051                             573926780, 324066546, 454325408, 147431459};
00052
00053
00054     // Функции, в которых происходит генерация выборок для каждого из генераторов
00055     create_samples<mid_xor_PRNG> (SAMPLES_DIR, "mid_xor", sizes, seeds, MIN_LIM, MAX_LIM);
00056     create_samples<mul_xor_PRNG> (SAMPLES_DIR, "mul_xor", sizes, seeds, MIN_LIM, MAX_LIM);
00057     create_samples<LCG> (SAMPLES_DIR, "lcg", sizes, seeds, MIN_LIM, MAX_LIM);
00058     create_samples<div_PRNG> (SAMPLES_DIR, "div", sizes, seeds, MIN_LIM, MAX_LIM);
00059
00060
00061     std::endl(std::cout);
00062 }

```

6.2 create_samples.cpp

[См. документацию.](#)

```

00001
00008
00009
00010
00011 #include <filesystem>
00012
00013 #include <fstream>
00014 #include <iostream>
00015
00016 #include <string>
00017 #include <vector>
00018
00019 #include "include/prng.h"
00020 #include "include/defs.h"
00021
00022
00023 namespace fs = std::filesystem;
00024
00025 template <class PRNG>
00026 void create_samples (std::string, std::string, std::vector<int>&, std::vector<int>&, uint32_t = 0, uint32_t = -1);
00027
00028
00029
00031 int main()
00032 {

```

```

00033 std::ofstream sample_file;
00034
00035 // Создать папку для выборок, если её нет
00036 fs::create_directory(SAMPLES_DIR);
00037
00038
00039 // Размеры выборок
00040 std::vector<int> sizes = { 1000, 1000, 1000, 1000,
00041                          2000, 2000, 2000, 2000,
00042                          3000, 3000, 3000, 3000,
00043                          5000, 5000, 5000, 5000,
00044                          10000, 10000, 10000, 10000};
00045
00046 // Семена генераторов
00047 std::vector<int> seeds = { 42, 13973739, 323159976, 518977272,
00048                          667961784, 958390147, 567454690, 292832249,
00049                          583553826, 722672343, 621406124, 771825685,
00050                          398112944, 209114256, 242948276, 826041245,
00051                          573926780, 324066546, 454325408, 147431459};
00052
00053
00054 // Функции, в которых происходит генерация выборок для каждого из генераторов
00055 create_samples<mid_xor_PRNG> (SAMPLES_DIR, "mid_xor", sizes, seeds, MIN_LIM, MAX_LIM);
00056 create_samples<mul_xor_PRNG> (SAMPLES_DIR, "mul_xor", sizes, seeds, MIN_LIM, MAX_LIM);
00057 create_samples<LCG> (SAMPLES_DIR, "lcg", sizes, seeds, MIN_LIM, MAX_LIM);
00058 create_samples<div_PRNG> (SAMPLES_DIR, "div", sizes, seeds, MIN_LIM, MAX_LIM);
00059
00060
00061 std::endl(std::cout);
00062 }
00063
00072 template <class PRNG>
00073 void create_samples (std::string path, std::string prng_name,
00074                    std::vector<int>& sizes, std::vector<int>& seeds,
00075                    uint32_t min_lim, uint32_t max_lim)
00076 {
00077
00078
00079     int mod;
00080
00081     // Проверка корректности размеров списков seeds и sizes
00082     if (sizes.size() == seeds.size()) mod = sizes.size();
00083     else if (sizes.size() % seeds.size() == 0) mod = sizes.size() / seeds.size();
00084     else
00085     {
00086         std::cerr << "! Неправильные размерности набора размеров и семян генераторов." << std::endl;
00087         return;
00088     }
00089
00090
00091     std::cout << "Generating samples by " + prng_name << std::endl;
00092
00093     // Создание папки для выборок
00094     fs::create_directory(path + prng_name);
00095
00096     // Цикл создания выборок
00097     for (int i=0; i < sizes.size(); ++i)
00098     {
00099
00100         // Создать файл выборки
00101         std::ofstream sample_file(path + prng_name + "/" + prng_name + "_" + std::to_string(i+1) + ".txt");
00102
00103         // Создать генератор с заданным семенем и границами
00104         PRNG prng (seeds.at(i % mod), min_lim, max_lim);
00105
00106         // Создать и записать выборку
00107         prng.generate_sample(sample_file, sizes.at(i));
00108
00109         // Закрыть файл выборки
00110         sample_file.close();
00111
00112         std::cout << "> Complete " + prng_name + "_" + std::to_string(i+1) + ".txt" << std::endl;
00113     }
00114
00115     std::endl(std::cout);
00116 }
00117

```

6.3 Файл get_results.cpp

Файл для вычисления измерений и проведения тестов над выборками


```
#include <filesystem>
#include <fstream>
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
#include "include/tests.h"
#include "include/defs.h"
```

Функции

- void `process_sample` (std::istream &sample_file, std::vector< std::string > &vector)
Функция для полной обработки выборки генератора
- int `main` ()
Основная функция, в которой производится вычисление всех значений и проведение тестов для всех выборок всех генераторов

6.3.1 Подробное описание

Файл для вычисления измерений и проведения тестов над выборками

В этом файле происходит вычисление Среднего, стандартного отклонения, коэффициента вариации, тест хи-квадрат, а также 5 nist-тестов.

Среди nist-тестов:

- Частотный побитовый тест
- Тест на последовательность одинаковых битов
- Тест на периодичность
- Тест приближительной энтропии
- Тест кумулятивных сумм

См. определение в файле `get_results.cpp`

6.3.2 Функции

6.3.2.1 `main()`

```
int main ()
```

Основная функция, в которой производится вычисление всех значений и проведение тестов для всех выборок всех генераторов

См. определение в файле `get_results.cpp` строка 36

```
00037 {
00038
00039     std::ifstream sample_file;
00040     std::ofstream result_file;
00041
00042
```

```

00043 // Цикл прохождения по содержимому главной папки с выборками
00044 for (auto const& dir_entry : fs::directory_iterator(SAMPLES_DIR))
00045 {
00046     // Обрабатываем только папки с выборками генераторов
00047     if (!dir_entry.is_directory()) continue;
00048
00049     // Получаем имя генератора
00050     std::string subdir_name = dir_entry.path().filename().string();
00051
00052     std::cout << "Processing " << subdir_name << std::endl;
00053
00054     // Подготавливаем файл для записи результатов
00055     result_file.open(SAMPLES_DIR + subdir_name + "/" + subdir_name + "_results.csv");
00056     result_file << "No,Size,"
00057                  << "Mean,"
00058                  << "StdDev,"
00059                  << "CV,"
00060                  << "Chi2,Chi2 df,"
00061                  << "NIST frequency,frequency status,"
00062                  << "NIST runs,runs status,"
00063                  << "NIST serial,serial block size,serial status,"
00064                  << "NIST approximate entropy,ApEn block size,ApEn status,"
00065                  << "NIST cusum,cusum status\n";
00066
00067     // Проход по файлам папки произвольный, поэтому чтобы сохранить очерёдность будем временно сохранять
00068     результаты в вектор
00069     std::vector< std::pair< int,std::vector<std::string> > > result_vector;
00070
00071
00072
00073     // Цикл для прохода по всем выборкам генератора
00074     for (auto const& subdir_entry : fs::directory_iterator(SAMPLES_DIR + subdir_name))
00075     {
00076         // Файл для записи результата игнорируем
00077         if (subdir_entry.path().extension().string() == ".csv") continue;
00078
00079         // Открываем файл выборки
00080         sample_file.open(subdir_entry.path());
00081
00082         std::string subdir_entry_stem = subdir_entry.path().stem().string();
00083
00084         std::cout << "> Processing " << subdir_entry_stem << std::endl;
00085
00086
00087         // Вектор для временного хранения результатов
00088         std::pair<int,std::vector<std::string>> sample_result;
00089
00090         sample_result.first = std::stoi(subdir_entry_stem.substr(subdir_entry_stem.rfind("_") + 1));
00091
00092         // Обрабатываем выборку
00093         process_sample(sample_file, sample_result.second);
00094
00095         // Сохраняем результат
00096         result_vector.push_back(sample_result);
00097
00098
00099         sample_file.close();
00100     }
00101
00102
00103     // Сортируем результаты по увеличению номера выборки
00104     std::sort( result_vector.begin(), result_vector.end(), [](auto& el1, auto& el2){ return el1.first < el2.first; } );
00105
00106     // Записываем результаты в файл
00107     for (auto& pair : result_vector)
00108     {
00109         result_file << pair.first;
00110
00111         for (auto& val : pair.second)
00112         {
00113             result_file << ',' << val;
00114         }
00115         result_file << std::endl;
00116     }
00117
00118     // Закрываем файл результатов генератора
00119     result_file.close();
00120
00121     std::endl(std::cout);
00122 }
00123
00124
00125 std::endl(std::cout);
00126
00127 }

```

6.3.2.2 process_sample()

```
void process_sample (
    std::istream & sample_file,
    std::vector< std::string > & vector)
```

Функция для полной обработки выборки генератора

Аргументы

sample_file	Файл выборки
vector	Вектор для хранения найденных результатов

См. определение в файле [get_results.cpp](#) строка 133

```
00134 {
00135     // Записываем выборку в вектор для удобства
00136     std::vector<unsigned int> sample;
00137     unsigned int num;
00138
00139     while (true)
00140     {
00141         sample_file >> num;
00142         if (sample_file.eof()) break;
00143
00144         sample.push_back(num);
00145     }
00146
00147
00148
00149     vector.push_back( std::to_string(sample.size()) ); // Записываем размер выборки
00150     vector.push_back( std::to_string(get_mean(sample)) ); // Записываем среднее выборки
00151     vector.push_back( std::to_string(get_stdDev(sample)) ); // Записываем стандартное отклонение выборки
00152     vector.push_back( std::to_string(get_cv(sample)) ); // Записываем коэффициент вариации выборки
00153
00154     auto chi2_pair = get_chi2(sample); // Проводим хи-квадрат тест
00155     vector.push_back( chi2_pair.first == -1 ? "N/A" : std::to_string(chi2_pair.first) ); // Записываем результат или
00156     // N/A в случае ошибки
00157     vector.push_back( chi2_pair.second == -1 ? "N/A" : std::to_string(chi2_pair.second) ); // Записываем количество
00158     // степеней свободы или N/A в случае ошибки
00159
00160
00161     // Переводим вектор чисел в последовательность бит для проведения nist-тестов
00162     std::vector<bool> bit_sequence = sample_to_bit_sequence(sample);
00163
00164     double frequency = nist_frequency(bit_sequence); // Проводим частотный побитовый тест
00165     vector.push_back( std::to_string(frequency) ); // Записываем полученное значение p_value
00166     vector.push_back( (frequency >= ALPHA) ? "PASS" : "FAIL" ); // Записываем результат теста
00167
00168     double runs = nist_runs(bit_sequence); // Проводим тест на последовательность одинаковых битов
00169     vector.push_back( std::to_string(runs) ); // Записываем полученное значение p_value
00170     vector.push_back( (runs >= ALPHA) ? "PASS" : "FAIL" ); // Записываем результат теста
00171
00172     double serial = nist_serial(bit_sequence, SERIAL_M); // Проводим тест на периодичность
00173     vector.push_back( std::to_string(serial) ); // Записываем полученное значение p_value
00174     vector.push_back( (serial >= ALPHA) ? "PASS" : "FAIL" ); // Записываем длину рассматриваемого блока
00175     vector.push_back( (serial >= ALPHA) ? "PASS" : "FAIL" ); // Записываем результат теста
00176
00177     double apEntropy = nist_apEntropy(bit_sequence, AP_ENTROPY_M); // Проводим тест приближительной
00178     // энтропии
00179     vector.push_back( std::to_string(apEntropy) ); // Записываем полученное значение p_value
00180     vector.push_back( std::to_string(AP_ENTROPY_M) ); // Записываем длину рассматриваемого блока
00181     vector.push_back( (apEntropy >= ALPHA) ? "PASS" : "FAIL" ); // Записываем результат теста
00182
00183     double cusum = nist_cusum(bit_sequence); // Проводим тест кумулятивных сумм
00184     vector.push_back( std::to_string(cusum) ); // Записываем полученное значение p_value
00185     vector.push_back( (cusum >= ALPHA) ? "PASS" : "FAIL" ); // Записываем результат теста
00186 }
```

6.4 get_results.cpp

[См. документацию.](#)

```

00001
00015
00016 #include <filesystem>
00017
00018 #include <fstream>
00019 #include <iostream>
00020
00021 #include <vector>
00022 #include <string>
00023 #include <algorithm>
00024
00025 #include "include/tests.h"
00026 #include "include/defs.h"
00027
00028
00029
00030 namespace fs = std::filesystem;
00031
00032
00033 void process_sample (std::istream&, std::vector<std::string>&);
00034
00036 int main ()
00037 {
00038
00039     std::ifstream sample_file;
00040     std::ofstream result_file;
00041
00042
00043     // Цикл прохождения по содержимому главной папки с выборками
00044     for (auto const& dir_entry : fs::directory_iterator(SAMPLES_DIR))
00045     {
00046         // Обрабатываем только папки с выборками генераторов
00047         if (!dir_entry.is_directory()) continue;
00048
00049         // Получаем имя генератора
00050         std::string subdir_name = dir_entry.path().filename().string();
00051
00052         std::cout << "Processing " << subdir_name << std::endl;
00053
00054         // Подготавливаем файл для записи результатов
00055         result_file.open(SAMPLES_DIR + subdir_name + "/" + subdir_name + "_results.csv");
00056         result_file << "No,Size,"
00057             << "Mean,"
00058             << "StdDev,"
00059             << "CV,"
00060             << "Chi2,Chi2 df,"
00061             << "NIST frequency,frequency status,"
00062             << "NIST runs,runs status,"
00063             << "NIST serial,serial block size,serial status,"
00064             << "NIST approximate entropy,ApEn block size,ApEn status,"
00065             << "NIST cusum,cusum status\n";
00066
00067
00068         // Проход по файлам папки произвольный, поэтому чтобы сохранить очерёдность будем временно сохранять
результаты в вектор
00069         std::vector< std::pair< int,std::vector<std::string> > > result_vector;
00070
00071
00072
00073         // Цикл для прохода по всем выборкам генератора
00074         for (auto const& subdir_entry : fs::directory_iterator(SAMPLES_DIR + subdir_name))
00075         {
00076             // Файл для записи результата игнорируем
00077             if (subdir_entry.path().extension().string() == ".csv") continue;
00078
00079             // Открываем файл выборки
00080             sample_file.open(subdir_entry.path());
00081
00082             std::string subdir_entry_stem = subdir_entry.path().stem().string();
00083
00084             std::cout << "> Processing " << subdir_entry_stem << std::endl;
00085
00086
00087             // Вектор для временного хранения результатов
00088             std::pair<int,std::vector<std::string>> sample_result;
00089
00090             sample_result.first = std::stoi(subdir_entry_stem.substr(subdir_entry_stem.rfind("_") + 1));
00091
00092             // Обрабатываем выборку
00093             process_sample(sample_file, sample_result.second);
00094
00095             // Сохраняем результат
00096             result_vector.push_back(sample_result);
00097
00098
00099             sample_file.close();
00100         }

```

```

00101
00102
00103 // Сортируем результаты по увеличению номера выборки
00104 std::sort( result_vector.begin(), result_vector.end(), [](auto& el1, auto& el2){ return el1.first < el2.first; } );
00105
00106 // Записываем результаты в файл
00107 for (auto& pair : result_vector)
00108 {
00109     result_file << pair.first;
00110
00111     for (auto& val : pair.second)
00112     {
00113         result_file << ' ' << val;
00114     }
00115     result_file << std::endl;
00116 }
00117
00118 // Закрываем файл результатов генератора
00119 result_file.close();
00120
00121 std::endl(std::cout);
00122 }
00123
00124
00125 std::endl(std::cout);
00126
00127 }
00128
00129
00130 void process_sample (std::istream& sample_file, std::vector<std::string>& vector)
00131 {
00132     // Записываем выборку в вектор для удобства
00133     std::vector<unsigned int> sample;
00134     unsigned int num;
00135
00136     while (true)
00137     {
00138         sample_file >> num;
00139         if (sample_file.eof()) break;
00140
00141         sample.push_back(num);
00142     }
00143
00144     vector.push_back( std::to_string(sample.size()) ); // Записываем размер выборки
00145     vector.push_back( std::to_string(get_mean(sample)) ); // Записываем среднее выборки
00146     vector.push_back( std::to_string(get_stdDev(sample)) ); // Записываем стандартное отклонение выборки
00147     vector.push_back( std::to_string(get_cv(sample)) ); // Записываем коэффициент вариации выборки
00148
00149     auto chi2_pair = get_chi2(sample); // Проводим хи-квадрат тест
00150     vector.push_back( chi2_pair.first == -1 ? "N/A" : std::to_string(chi2_pair.first) ); // Записываем результат или
00151     // N/A в случае ошибки
00152     vector.push_back( chi2_pair.second == -1 ? "N/A" : std::to_string(chi2_pair.second) ); // Записываем количество
00153     // степеней свободы или N/A в случае ошибки
00154
00155     // Переводим вектор чисел в последовательность бит для проведения nist-тестов
00156     std::vector<bool> bit_sequence = sample_to_bit_sequence(sample);
00157
00158     double frequency = nist_frequency(bit_sequence); // Проводим частотный побитовый тест
00159     vector.push_back( std::to_string(frequency) ); // Записываем полученное значение p_value
00160     vector.push_back( (frequency >= ALPHA) ? "PASS" : "FAIL" ); // Записываем результат теста
00161
00162     double runs = nist_runs(bit_sequence); // Проводим тест на последовательность одинаковых битов
00163     vector.push_back( std::to_string(runs) ); // Записываем полученное значение p_value
00164     vector.push_back( (runs >= ALPHA) ? "PASS" : "FAIL" ); // Записываем результат теста
00165
00166     double serial = nist_serial(bit_sequence, SERIAL_M); // Проводим тест на периодичность
00167     vector.push_back( std::to_string(serial) ); // Записываем полученное значение p_value
00168     vector.push_back( (serial >= ALPHA) ? "PASS" : "FAIL" ); // Записываем длину рассматриваемого блока
00169     vector.push_back( (serial >= ALPHA) ? "PASS" : "FAIL" ); // Записываем результат теста
00170
00171     double apEntropy = nist_apEntropy(bit_sequence, AP_ENTROPY_M); // Проводим тест приближительной
00172     // энтропии
00173     vector.push_back( std::to_string(apEntropy) ); // Записываем полученное значение p_value
00174     vector.push_back( std::to_string(AP_ENTROPY_M) ); // Записываем длину рассматриваемого блока
00175     vector.push_back( (apEntropy >= ALPHA) ? "PASS" : "FAIL" ); // Записываем результат теста
00176
00177     double cusum = nist_cusum(bit_sequence); // Проводим тест кумулятивных сумм
00178     vector.push_back( std::to_string(cusum) ); // Записываем полученное значение p_value
00179     vector.push_back( (cusum >= ALPHA) ? "PASS" : "FAIL" ); // Записываем результат теста
00180
00181 }
00182
00183
00184
00185 }
00186
00187

```

6.5 Файл include/defs.h

Заголовочный файл, содержащий определения, используемые во всей лабораторной работе

Макросы

- `#define SAMPLES_DIR "./samples/"`
Папка, в которой создаются выборки
- `#define MIN_LIM 0`
Нижняя граница генерации чисел генераторами
- `#define MAX_LIM 4095`
Верхняя граница генерации чисел генераторами
- `#define ALPHA 0.01`
Уровень значимости для nist-тестов
- `#define SERIAL_M 2`
Длина блока в тесте на периодичность
- `#define AP_ENTROPY_M 2`
Длина блока в тесте приближенной энтропии

6.5.1 Подробное описание

Заголовочный файл, содержащий определения, используемые во всей лабораторной работе

См. определение в файле [defs.h](#)

6.5.2 Макросы

6.5.2.1 ALPHA

```
#define ALPHA 0.01
```

Уровень значимости для nist-тестов

См. определение в файле [defs.h](#) строка 15

6.5.2.2 AP_ENTROPY_M

```
#define AP_ENTROPY_M 2
```

Длина блока в тесте приближенной энтропии

См. определение в файле [defs.h](#) строка 17

6.5.2.3 MAX_LIM

```
#define MAX_LIM 4095
```

Верхняя граница генерации чисел генераторами

См. определение в файле [defs.h](#) строка 13

6.5.2.4 MIN_LIM

```
#define MIN_LIM 0
```

Нижняя граница генерации чисел генераторами

См. определение в файле [defs.h](#) строка 12

6.5.2.5 SAMPLES_DIR

```
#define SAMPLES_DIR "./samples/"
```

Папка, в которой создаются выборки

См. определение в файле [defs.h](#) строка 10

6.5.2.6 SERIAL_M

```
#define SERIAL_M 2
```

Длина блока в тесте на периодичность

См. определение в файле [defs.h](#) строка 16

6.6 defs.h

[См. документацию.](#)

```
00001
00005
00006
00007 #ifndef DEFS_H
00008 #define DEFS_H
00009
00010 #define SAMPLES_DIR "./samples/"
00011
00012 #define MIN_LIM 0
00013 #define MAX_LIM 4095
00014
00015 #define ALPHA 0.01
00016 #define SERIAL_M 2
00017 #define AP_ENTROPY_M 2
00018
00019
00020 #endif // DEFS_H
```

6.7 cephes.h

```
00001
00002 #ifndef _CEPHES_H_
00003 #define _CEPHES_H_
00004
00005 double cephes_igamc(double a, double x);
00006 double cephes_igam(double a, double x);
00007 double cephes_lgam(double x);
00008 double cephes_plevl(double x, double *coef, int N);
00009 double cephes_polevl(double x, double *coef, int N);
00010 double cephes_erf(double x);
00011 double cephes_erfc(double x);
00012 double cephes_normal(double x);
00013
00014 #endif /* _CEPHES_H_ */
```

6.8 Файл include/prng.h

Заголовочный файл, в котором определяются классы генераторов.

```
#include <cstdint>
```

Классы

- class [PRNG](#)
Абстрактный базовый класс генераторов
- class [mid_xor_PRNG](#)
Генератор, основанный на среднем квадрата и Хор.
- class [mul_xor_PRNG](#)
Генератор, основанный на произведении и Хор.
- class [LCG](#)
Генератор, основанный на линейном конгруэнтном методе
- class [div_PRNG](#)
Генератор, основанный на делении чисел друг на друга

6.8.1 Подробное описание

Заголовочный файл, в котором определяются классы генераторов.

Объявляемые классы:

- [PRNG](#)
- [mid_xor_PRNG](#)
- [mul_xor_PRNG](#)
- [LCG](#)

См. определение в файле [prng.h](#)

6.9 prng.h

[См. документацию.](#)

```
00001
00011
00012 #ifndef PRNG_H
00013 #define PRNG_H
00014
00015 #include <cstdint> // для типов int
00016
00017
00019 class PRNG
00020 {
00021
00022 protected:
00023     const uint32_t min_lim;
00024     const uint32_t max_lim;
00025
00026     PRNG (uint32_t, uint32_t);
00027
```



```

00028 virtual uint32_t generate() = 0;
00029
00030 uint32_t result(uint32_t) const;
00031
00032 public:
00033 void generate_sample(std::ostream&, int);
00034
00035 };
00036
00037
00038
00040 class mid_xor_PRNG: public PRNG
00041 {
00042     uint32_t seed;
00043
00044
00045 public:
00046     mid_xor_PRNG () = delete;
00047     mid_xor_PRNG ( uint32_t, uint32_t = 0, uint32_t = -1);
00048
00049     uint32_t generate() override;
00050 };
00051
00052
00054 class mul_xor_PRNG: public PRNG
00055 {
00056     uint32_t seed;
00057
00058
00059 public:
00060     mul_xor_PRNG () = delete;
00061     mul_xor_PRNG ( uint32_t, uint32_t = 0, uint32_t = -1);
00062
00063     uint32_t generate() override;
00064 };
00065
00066
00068 class LCG: public PRNG
00069 {
00070     uint32_t seed;
00071     uint32_t k;
00072     uint32_t b;
00073     uint32_t M;
00074
00075
00076 public:
00077     LCG () = delete;
00078     LCG ( uint32_t, uint32_t = 0, uint32_t = -1);
00079
00080     uint32_t generate() override;
00081 };
00082
00083
00084
00086 class div_PRNG: public PRNG
00087 {
00088     uint32_t seed;
00089     uint32_t prev;
00090
00091 public:
00092     div_PRNG () = delete;
00093     div_PRNG ( uint32_t, uint32_t = 0, uint32_t = -1);
00094
00095     uint32_t generate() override;
00096 };
00097
00098
00099 #endif // PRNG_H

```

6.10 Файл include/tests.h

Заголовочный файл, содержащий объявления всех функций для вычисления значений и проведения тестов

```
#include <vector>
```

Функции

- double `get_mean` (std::vector< unsigned int > &)
Оценка выборки на среднее значение
- double `get_stdDev` (std::vector< unsigned int > &)
Оценка выборки на стандартное отклонение
- double `get_cv` (std::vector< unsigned int > &)
Оценка выборки на коэффициент вариации
- std::pair< double, int > `get_chi2` (std::vector< unsigned int > &)
Тест хи-квадрат
- std::vector< bool > `sample_to_bit_sequence` (std::vector< unsigned int > &)
Функция, преобразовывающая вектор выборки в последовательность битов этой выборки
- std::vector< unsigned int > `get_pattern_counts` (std::vector< bool > &, int)
Функция, которая выявляет паттерны битов заданной длины в последовательности и считает их количество
- double `get_psi2` (std::vector< bool > &, int)
Вспомогательная функция теста на периодичность
- double `get_apEn` (std::vector< bool > &, int)
Вспомогательная функция теста приближенной энтропии
- double `nist_frequency` (std::vector< bool > &)
Частотный побитовый тест
- double `nist_runs` (std::vector< bool > &)
Тест на последовательность одинаковых битов
- double `nist_serial` (std::vector< bool > &, int)
Тест на периодичность
- double `nist_apEntropy` (std::vector< bool > &, int)
Тест приближенной энтропии
- double `nist_cusum` (std::vector< bool > &)
Тест кумулятивных сумм

6.10.1 Подробное описание

Заголовочный файл, содержащий объявления всех функций для вычисления значений и проведения тестов

См. определение в файле [tests.h](#)

6.10.2 Функции

6.10.2.1 `get_apEn()`

```
double get_apEn (
    std::vector< bool > & bit_seq,
    int m)
```

Вспомогательная функция теста приближенной энтропии

Аргументы

bit_seq	Последовательность битов
m	Длина блока

Возвращает

Искомое значение

См. определение в файле [nist_funcs.cpp](#) строка 89

```
00090 {
00091     std::vector<unsigned int> pattern_counts = get_pattern_counts(bit_seq, m);
00092
00093     double apEn = 0;
00094
00095     for ( auto count : pattern_counts)
00096     {
00097         double p = (double)count / bit_seq.size();
00098         apEn += (p > std::pow(10,-6)) ? (p * std::log(p)) : 0;
00099     }
00100
00101     return apEn;
00102 }
```

6.10.2.2 get_chi2()

```
std::pair< double, int > get_chi2 (
    std::vector< unsigned int > & vec)
```

Тест хи-квадрат

Аргументы

vec	Вектор выборки
-----	----------------

Возвращает

Пара значений (значение хи-квадрат, количество степеней свободы)

См. определение в файле [tests.cpp](#) строка 56

```
00057 {
00058     if (vec.empty())
00059         return {-1,-1};
00060
00061     auto min_val = *std::min_element(vec.begin(), vec.end()); // Наименьший элемент выборки
00062     auto max_val = *std::max_element(vec.begin(), vec.end()); // Наибольший элемент выборки
00063
00064     int bin_number = 1 + std::log2(vec.size()); // Количество промежутков по правилу Стерджеса
00065     float bin_width = (float)(max_val-min_val)/bin_number; // Ширина промежутка
00066
00067     float expected = (float)vec.size() / bin_number; // Количество ожидаемых наблюдений в каждом промежутке
00068     std::vector<int> observed(bin_number, 0); // Количество фактических наблюдений в каждом промежутке
00069
00070
00071     for ( auto val : vec )
00072     {
00073         int bin_ind = (int) ((val - min_val) / bin_width);
00074         if (bin_ind >= bin_number) --bin_ind;
00075         ++observed.at(bin_ind);
00076     }
00077
00078
00079
00080     double chi2 = 0;
00081
00082     for ( auto obs : observed)
00083     {
00084         chi2 += (obs - expected) * (obs - expected) / expected;
00085     }
00086
00087     return {chi2, bin_number-1};
00088
00089 }
```

6.10.2.3 get_cv()

```
double get_cv (  
    std::vector< unsigned int > & vec)
```

Оценка выборки на коэффициент вариации

Аргументы

vec	Вектор выборки
-----	----------------

Возвращает

Коэффициент вариации

См. определение в файле [tests.cpp](#) строка [47](#)

```
00048 {
00049     return get_stdDev(vec) / get_mean(vec);
00050 }
```

6.10.2.4 get_mean()

```
double get_mean (
    std::vector< unsigned int > & vec)
```

Оценка выборки на среднее значение

Аргументы

vec	Вектор выборки
-----	----------------

Возвращает

Среднее значение

См. определение в файле [tests.cpp](#) строка [14](#)

```
00015 {
00016     double sum = 0;
00017
00018     for ( auto& num : vec )
00019     {
00020         sum += num;
00021     }
00022
00023     return sum / vec.size();
00024 }
```

6.10.2.5 get_pattern_counts()

```
std::vector< unsigned int > get_pattern_counts (
    std::vector< bool > & bit_seq,
    int m)
```

Функция, которая выявляет паттерны битов заданной длины в последовательности и считает их количество

Аргументы

bit_seq	Последовательность битов
m	Длина паттерна

Возвращает

Количество встреченных паттернов одного вида. Паттерну, являющемуся двоичным представление числа num, соответствует индекс num

См. определение в файле [nist_funcs.cpp](#) строка 43

```
00044 {
00045     std::vector<unsigned int> pattern_counts(1«m, 0);
00046
00047     for (int i=0; i < bit_seq.size(); ++i)
00048     {
00049         int k=0;
00050         for (int j = m-1; j >= 0; --j) // Проходимся в обратном порядке для эстетичности, чтобы паттерн равный числу I
            был записан под индексом I
00051         {
00052             if (bit_seq.at((i+j) % bit_seq.size())) k = 2*k + 1;
00053             else k = 2*k;
00054         }
00055         ++pattern_counts[k];
00056     }
00057
00058     return pattern_counts;
00059 }
```

6.10.2.6 get_psi2()

```
double get_psi2 (
    std::vector< bool > & bit_seq,
    int m)
```

Вспомогательная функция теста на периодичность

Аргументы

bit_seq	Последовательность битов
m	Длина блока

Возвращает

Искомое значение

См. определение в файле [nist_funcs.cpp](#) строка 66

```
00067 {
00068     std::vector<unsigned int> pattern_counts = get_pattern_counts(bit_seq, m);
00069
00070
00071     int expected = bit_seq.size() / (1 « m);
00072
00073     double sum = 0;
00074     for ( auto count : pattern_counts)
00075     {
00076         sum += (count - expected) * (count - expected);
00077     }
00078
00079     sum = ( sum * (1 « m) ) / bit_seq.size() - bit_seq.size();
00080
00081     return sum;
00082 }
```

6.10.2.7 get_stdDev()

```
double get_stdDev (
    std::vector< unsigned int > & vec)
```

Оценка выборки на стандартное отклонение

Аргументы

vec	Вектор выборки
-----	----------------

Возвращает

Стандартное отклонение

См. определение в файле [tests.cpp](#) строка 30

```
00031 {
00032     unsigned long sum = 0;
00033     auto mean = get_mean(vec);
00034
00035     for ( auto& num : vec )
00036     {
00037         sum += (num - mean) * (num - mean);
00038     }
00039
00040     return std::sqrt(sum / vec.size());
00041 }
```

6.10.2.8 nist_apEntropy()

```
double nist_apEntropy (
    std::vector< bool > & bit_seq,
    int m)
```

Тест приближительной энтропии

Аргументы

bit_seq	Последовательность битов выборки
m	Длина рассматриваемого блока бит

Возвращает

p_value

См. определение в файле [nist_tests.cpp](#) строка 94

```
00095 {
00096     double apEn_m = get_apEn(bit_seq, m);
00097     double apEn_m1 = get_apEn(bit_seq, m+1);
00098
00099     double apEn = apEn_m - apEn_m1;
00100     double chi2 = 2 * bit_seq.size() * (std::log(2) - apEn);
00101
00102     double p_value = cephes_igamc(1 * (m-1), chi2/2);
00103     return p_value;
00104 }
```

6.10.2.9 nist_cusum()

```
double nist_cusum (
    std::vector< bool > & bit_seq)
```

Тест кумулятивных сумм

Аргументы

bit_seq	Последовательность битов выборки
---------	----------------------------------

Возвращает

p_value

См. определение в файле [nist_tests.cpp](#) строка 109

```

00110 {
00111     int S = 0;
00112     int max_S = 0;
00113
00114
00115     for ( auto bit : bit_seq )
00116     {
00117         bit ? ++S : --S;
00118         max_S = (std::abs(S) > max_S) ? std::abs(S) : max_S;
00119     }
00120
00121     double p_value = std::erfc(max_S / std::sqrt(2 * bit_seq.size()));
00122
00123     return p_value;
00124 }
```

6.10.2.10 nist_frequency()

```
double nist_frequency (
    std::vector< bool > & bit_seq)
```

Частотный побитовый тест

Аргументы

bit_seq	Последовательность битов выборки
---------	----------------------------------

Возвращает

p_value

См. определение в файле [nist_tests.cpp](#) строка 29

```

00030 {
00031     int sum = 0;
00032
00033     for (auto bit : bit_seq)
00034     {
00035         bit ? ++sum : --sum; // Итого: ++sum, если бит - 1, иначе --sum
00036     }
00037
00038     double s_obs = std::abs(sum) / std::sqrt(bit_seq.size());
00039     double p_value = std::erfc(s_obs / std::sqrt(2));
00040
00041     return p_value;
00042 }
```

6.10.2.11 nist_runs()

```
double nist_runs (
    std::vector< bool > & bit_seq)
```

Тест на последовательность одинаковых битов

Аргументы

bit_seq	Последовательность битов выборки
---------	----------------------------------

Возвращает

p_value

См. определение в файле [nist_tests.cpp](#) строка 48

```

00049 {
00050
00051     int sum = std::count(bit_seq.begin(), bit_seq.end(), true);
00052     double pi = (double)sum / bit_seq.size();
00053
00054     if (fabs(pi - 0.5) >= 2 / std::sqrt(bit_seq.size()))
00055     {
00056         std::cerr << "»> Не выполнен критерий для runs теста" << std::endl;
00057         return 0;
00058     }
00059
00060
00061     int runs = 1;
00062     for (int i=1; i < bit_seq.size(); ++i)
00063     {
00064         if (bit_seq.at(i) != bit_seq.at(i-1))
00065             ++runs;
00066     }
00067
00068     double erfc_arg = fabs(runs - 2.0 * bit_seq.size() * pi * (1-pi)) / (2.0 * pi * (1-pi) * sqrt(2*bit_seq.size()));
00069     double p_value = erfc(erfc_arg);
00070
00071     return p_value;
00072 }
00073
00074 }
```

6.10.2.12 nist_serial()

```

double nist_serial (
    std::vector< bool > & bit_seq,
    int m)

```

Тест на периодичность

Аргументы

bit_seq	Последовательность битов выборки
m	Длина рассматриваемого блока бит

Возвращает

p_value

См. определение в файле [nist_tests.cpp](#) строка 80

```

00081 {
00082     double psi_m = get_psi2(bit_seq, m);
00083     double psi_m_1 = get_psi2(bit_seq, m-1);
00084
00085
00086     double p_value = cephes_igamc(1 << (m-2), (psi_m - psi_m_1)/2);
00087     return p_value;
00088 }
```

6.10.2.13 sample_to_bit_sequence()

```

std::vector< bool > sample_to_bit_sequence (
    std::vector< unsigned int > & vec)

```

Функция, преобразовывающая вектор выборки в последовательность битов этой выборки

Аргументы

vec	Вектор выборки
-----	----------------

Возвращает

Последовательность битов

См. определение в файле [nist_funcs.cpp](#) строка 18

```

00019 {
00020     int max_len = 1 + std::log2(MAX_LIM); // Находим наибольший бит, который может принять значение 1
00021                                           // Старшие биты всегда будут 0, что будет нарушать тесты, поэтому обрубам их
00022
00023     std::vector<bool> bit_seq;
00024
00025     for (auto num : vec)
00026     {
00027         for (int i=0; i < max_len; ++i)
00028         {
00029             bit_seq.push_back(num » i & 1); // биты хранятся в обратном порядке, т.е. чем меньше индекс, тем младше бит
00030         }
00031     }
00032
00033     return bit_seq;
00034
00035
00036 }
```

6.11 tests.h

См. документацию.

```

00001
00005
00006
00007 #ifndef TESTS_H
00008 #define TESTS_H 1
00009
00010
00011 #include <vector>
00012
00013 double get_mean ( std::vector<unsigned int>& );
00014 double get_stdDev ( std::vector<unsigned int>& );
00015 double get_cv ( std::vector<unsigned int>& );
00016
00017 std::pair<double, int> get_chi2 ( std::vector<unsigned int>& );
00018
00019
00020 std::vector<bool> sample_to_bit_sequence ( std::vector<unsigned int>& );
00021
00022 std::vector<unsigned int> get_pattern_counts ( std::vector<bool>&, int );
00023 double get_psi2 ( std::vector<bool>&, int );
00024 double get_apEn ( std::vector<bool>&, int );
00025
00026
00027 double nist_frequency ( std::vector<bool>& );
00028 double nist_runs ( std::vector<bool>& );
00029 double nist_serial ( std::vector<bool>&, int );
00030 double nist_apEntropy ( std::vector<bool>&, int );
00031 double nist_cusum ( std::vector<bool>& );
00032
00033
00034 #endif // TESTS_H
```

6.12 cephes.cpp

```

00001 #include <stdio.h>
00002 #include <math.h>
00003 #include "../include/external/cephes.h"
00004
```

```

00005 static const double rel_error = 1E-12;
00006
00007 double MACHEP = 1.11022302462515654042E-16; // 2**-53
00008 double MAXLOG = 7.09782712893383996732224E2; // log(MAXNUM)
00009 double MAXNUM = 1.7976931348623158E308; // 2**1024*(1-MACHEP)
00010 double PI = 3.14159265358979323846; // pi, duh!
00011
00012 static double big = 4.503599627370496e15;
00013 static double biginv = 2.22044604925031308085e-16;
00014
00015 int sgngam = 0;
00016
00017 double
00018 cephes_igamc(double a, double x)
00019 {
00020     double ans, ax, c, yc, r, t, y, z;
00021     double pk, pkm1, pkm2, qk, qkm1, qkm2;
00022
00023     if ( (x <= 0) || (a <= 0) )
00024         return( 1.0 );
00025
00026     if ( (x < 1.0) || (x < a) )
00027         return( 1.e0 - cephes_igam(a,x) );
00028
00029     ax = a * log(x) - x - cephes_lgam(a);
00030
00031     if ( ax < -MAXLOG ) {
00032         printf("igamc: UNDERFLOW\n");
00033         return 0.0;
00034     }
00035     ax = exp(ax);
00036
00037     /* continued fraction */
00038     y = 1.0 - a;
00039     z = x + y + 1.0;
00040     c = 0.0;
00041     pkm2 = 1.0;
00042     qkm2 = x;
00043     pkm1 = x + 1.0;
00044     qkm1 = z * x;
00045     ans = pkm1/qkm1;
00046
00047     do {
00048         c += 1.0;
00049         y += 1.0;
00050         z += 2.0;
00051         yc = y * c;
00052         pk = pkm1 * z - pkm2 * yc;
00053         qk = qkm1 * z - qkm2 * yc;
00054         if ( qk != 0 ) {
00055             r = pk/qk;
00056             t = fabs( (ans - r)/r );
00057             ans = r;
00058         }
00059         else
00060             t = 1.0;
00061         pkm2 = pkm1;
00062         pkm1 = pk;
00063         qkm2 = qkm1;
00064         qkm1 = qk;
00065         if ( fabs(pk) > big ) {
00066             pkm2 *= biginv;
00067             pkm1 *= biginv;
00068             qkm2 *= biginv;
00069             qkm1 *= biginv;
00070         }
00071     } while ( t > MACHEP );
00072
00073     return ans*ax;
00074 }
00075
00076 double
00077 cephes_igam(double a, double x)
00078 {
00079     double ans, ax, c, r;
00080
00081     if ( (x <= 0) || (a <= 0) )
00082         return 0.0;
00083
00084     if ( (x > 1.0) && (x > a) )
00085         return 1.e0 - cephes_igamc(a,x);
00086
00087     /* Compute x**a * exp(-x) / gamma(a) */
00088     ax = a * log(x) - x - cephes_lgam(a);
00089     if ( ax < -MAXLOG ) {
00090         printf("igam: UNDERFLOW\n");
00091         return 0.0;

```

```

00092     }
00093     ax = exp(ax);
00094
00095     /* power series */
00096     r = a;
00097     c = 1.0;
00098     ans = 1.0;
00099
00100     do {
00101         r += 1.0;
00102         c *= x/r;
00103         ans += c;
00104     } while ( c/ans > MACHEP );
00105
00106     return ans * ax/a;
00107 }
00108
00109
00110 /* A[]: Stirling's formula expansion of log gamma
00111 * B[], C[]: log gamma function between 2 and 3
00112 */
00113 static unsigned short A[] = {
00114     0x6661,0x2733,0x9850,0x3f4a,
00115     0xe943,0xb580,0x7fbd,0xbf43,
00116     0x5ebb,0x20dc,0x019f,0x3f4a,
00117     0xa5a1,0x16b0,0xc16c,0xbf66,
00118     0x554b,0x5555,0x5555,0x3fb5
00119 };
00120 static unsigned short B[] = {
00121     0x6761,0x8ff3,0x8901,0xc095,
00122     0xb93e,0x355b,0xf234,0xc0e2,
00123     0x89e5,0xf890,0x3d73,0xc114,
00124     0xdb51,0xf994,0xbc82,0xc131,
00125     0xf20b,0x0219,0x4589,0xc13a,
00126     0x055e,0x5418,0xc67,0xc12a
00127 };
00128 static unsigned short C[] = {
00129     /*0x0000,0x0000,0x0000,0x3ff0,*/
00130     0x12b2,0x1cf3,0xfd0d,0xc075,
00131     0xd757,0x7b89,0xaa0d,0xc0d0,
00132     0x4c9b,0xb974,0xeb84,0xc10a,
00133     0x0043,0x7195,0x6286,0xc131,
00134     0xf34c,0x892f,0x5255,0xc143,
00135     0xe14a,0x6a11,0xce4b,0xc13e
00136 };
00137
00138 #define MAXLGM 2.556348e305
00139
00140
00141 /* Logarithm of gamma function */
00142 double
00143 cephes_lgam(double x)
00144 {
00145     double p, q, u, w, z;
00146     int i;
00147
00148     sgngam = 1;
00149
00150     if ( x < -34.0 ) {
00151         q = -x;
00152         w = cephes_lgam(q); /* note this modifies sgngam! */
00153         p = floor(q);
00154         if ( p == q ) {
00155             lgsing:
00156                 goto loverf;
00157         }
00158         i = (int)p;
00159         if ( (i & 1) == 0 )
00160             sgngam = -1;
00161         else
00162             sgngam = 1;
00163         z = q - p;
00164         if ( z > 0.5 ) {
00165             p += 1.0;
00166             z = p - q;
00167         }
00168         z = q * sin( PI * z );
00169         if ( z == 0.0 )
00170             goto lgsing;
00171         /* z = log(PI) - log( z ) - w; */
00172         z = log(PI) - log( z ) - w;
00173         return z;
00174     }
00175
00176     if ( x < 13.0 ) {
00177         z = 1.0;
00178         p = 0.0;

```

```

00179     u = x;
00180     while ( u >= 3.0 ) {
00181         p -= 1.0;
00182         u = x + p;
00183         z *= u;
00184     }
00185     while ( u < 2.0 ) {
00186         if ( u == 0.0 )
00187             goto lgsing;
00188         z /= u;
00189         p += 1.0;
00190         u = x + p;
00191     }
00192     if ( z < 0.0 ) {
00193         snggam = -1;
00194         z = -z;
00195     }
00196     else
00197         snggam = 1;
00198     if ( u == 2.0 )
00199         return( log(z) );
00200     p -= 2.0;
00201     x = x + p;
00202     p = x * cephes_polevl( x, (double *)B, 5 ) / cephes_p1evl( x, (double *)C, 6);
00203
00204     return log(z) + p;
00205 }
00206
00207 if ( x > MAXLGM ) {
00208 loverf:
00209     printf("lgam: OVERFLOW\n");
00210
00211     return snggam * MAXNUM;
00212 }
00213
00214 q = ( x - 0.5 ) * log(x) - x + log( sqrt( 2*PI ) );
00215 if ( x > 1.0e8 )
00216     return q;
00217
00218 p = 1.0/(x*x);
00219 if ( x >= 1000.0 )
00220     q += (( 7.9365079365079365079365e-4 * p
00221         - 2.77777777777777777778e-3 ) * p
00222         + 0.08333333333333333333 ) / x;
00223 else
00224     q += cephes_polevl( p, (double *)A, 4 ) / x;
00225
00226 return q;
00227 }
00228
00229 double
00230 cephes_polevl(double x, double *coef, int N)
00231 {
00232     double ans;
00233     int i;
00234     double *p;
00235
00236     p = coef;
00237     ans = *p++;
00238     i = N;
00239
00240     do
00241         ans = ans * x + *p++;
00242     while ( --i );
00243
00244     return ans;
00245 }
00246
00247 double
00248 cephes_p1evl(double x, double *coef, int N)
00249 {
00250     double ans;
00251     double *p;
00252     int i;
00253
00254     p = coef;
00255     ans = x + *p++;
00256     i = N-1;
00257
00258     do
00259         ans = ans * x + *p++;
00260     while ( --i );
00261
00262     return ans;
00263 }
00264
00265 double

```

```

00266 cephes_erf(double x)
00267 {
00268     static const double two_sqrtpi = 1.128379167095512574;
00269     double sum = x, term = x, xsqr = x * x;
00270     int j = 1;
00271
00272     if ( fabs(x) > 2.2 )
00273         return 1.0 - cephes_erfc(x);
00274
00275     do {
00276         term *= xsqr/j;
00277         sum -= term/(2*j+1);
00278         j++;
00279         term *= xsqr/j;
00280         sum += term/(2*j+1);
00281         j++;
00282     } while ( fabs(term)/sum > rel_error );
00283
00284     return two_sqrtpi*sum;
00285 }
00286
00287 double
00288 cephes_erfc(double x)
00289 {
00290     static const double one_sqrtpi = 0.564189583547756287;
00291     double a = 1, b = x, c = x, d = x*x + 0.5;
00292     double q1, q2 = b/d, n = 1.0, t;
00293
00294     if ( fabs(x) < 2.2 )
00295         return 1.0 - cephes_erf(x);
00296     if ( x < 0 )
00297         return 2.0 - cephes_erfc(-x);
00298
00299     do {
00300         t = a*n + b*x;
00301         a = b;
00302         b = t;
00303         t = c*n + d*x;
00304         c = d;
00305         d = t;
00306         n += 0.5;
00307         q1 = q2;
00308         q2 = b/d;
00309     } while ( fabs(q1-q2)/q2 > rel_error );
00310
00311     return one_sqrtpi*exp(-x*x)*q2;
00312 }
00313
00314 double
00315 cephes_normal(double x)
00316 {
00317     double arg, result, sqrt2=1.414213562373095048801688724209698078569672;
00318
00319     if (x > 0) {
00320         arg = x/sqrt2;
00321         result = 0.5 * ( 1 + erf(arg) );
00322     }
00323     else {
00324         arg = -x/sqrt2;
00325         result = 0.5 * ( 1 - erf(arg) );
00326     }
00327 }
00328
00329 return( result);
00330 }

```

6.13 Файл src/nist_funcs.cpp

Файл, содержащий реализации вспомогательных функций для nist-тестов

```

#include <cmath>
#include "../include/tests.h"
#include "../include/defs.h"
#include <iostream>

```

Функции

- `std::vector< bool > sample_to_bit_sequence (std::vector< unsigned int > &vec)`
Функция, преобразовывающая вектор выборки в последовательность битов этой выборки
- `std::vector< unsigned int > get_pattern_counts (std::vector< bool > &bit_seq, int m)`
Функция, которая выявляет паттерны битов заданной длины в последовательности и считает их количество
- `double get_psi2 (std::vector< bool > &bit_seq, int m)`
Вспомогательная функция теста на периодичность
- `double get_apEn (std::vector< bool > &bit_seq, int m)`
Вспомогательная функция теста приближенной энтропии

6.13.1 Подробное описание

Файл, содержащий реализации вспомогательных функций для nist-тестов

См. определение в файле [nist_funcs.cpp](#)

6.13.2 Функции

6.13.2.1 get_apEn()

```
double get_apEn (
    std::vector< bool > & bit_seq,
    int m)
```

Вспомогательная функция теста приближенной энтропии

Аргументы

bit_seq	Последовательность битов
m	Длина блока

Возвращает

Искомое значение

См. определение в файле [nist_funcs.cpp](#) строка 89

```
00090 {
00091     std::vector<unsigned int> pattern_counts = get_pattern_counts(bit_seq, m);
00092
00093     double apEn = 0;
00094
00095     for ( auto count : pattern_counts)
00096     {
00097         double p = (double)count / bit_seq.size();
00098         apEn += (p > std::pow(10,-6)) ? (p * std::log(p)) : 0;
00099     }
00100
00101     return apEn;
00102 }
```

6.13.2.2 get_pattern_counts()

```
std::vector< unsigned int > get_pattern_counts (
    std::vector< bool > & bit_seq,
    int m)
```

Функция, которая выявляет паттерны битов заданной длины в последовательности и считает их количество

Аргументы

bit_seq	Последовательность битов
m	Длина паттерна

Возвращает

Количество встреченных паттернов одного вида. Паттерну, являющемуся двоичным представлением числа num, соответствует индекс num

См. определение в файле [nist_funcs.cpp](#) строка 43

```

00044 {
00045     std::vector<unsigned int> pattern_counts(1<<m, 0);
00046
00047     for (int i=0; i < bit_seq.size(); ++i)
00048     {
00049         int k=0;
00050         for (int j = m-1; j >= 0; --j) // Проходимся в обратном порядке для эстетичности, чтобы паттерн равный числу I
            был записан под индексом I
00051         {
00052             if (bit_seq.at((i+j) % bit_seq.size())) k = 2*k + 1;
00053             else k = 2*k;
00054         }
00055         ++pattern_counts[k];
00056     }
00057
00058     return pattern_counts;
00059 }
```

6.13.2.3 get_psi2()

```

double get_psi2 (
    std::vector< bool > & bit_seq,
    int m)
```

Вспомогательная функция теста на периодичность

Аргументы

bit_seq	Последовательность битов
m	Длина блока

Возвращает

Искомое значение

См. определение в файле [nist_funcs.cpp](#) строка 66

```

00067 {
00068     std::vector<unsigned int> pattern_counts = get_pattern_counts(bit_seq, m);
00069
00070     int expected = bit_seq.size() / (1 << m);
00071
00072     double sum = 0;
00073     for ( auto count : pattern_counts)
00074     {
00075         sum += (count - expected) * (count - expected);
00076     }
00077
00078     sum = ( sum * (1 << m) ) / bit_seq.size() - bit_seq.size();
00079
00080
00081     return sum;
00082 }
```


6.13.2.4 sample_to_bit_sequence()

```
std::vector< bool > sample_to_bit_sequence (
    std::vector< unsigned int > & vec)
```

Функция, преобразовывающая вектор выборки в последовательность битов этой выборки

Аргументы

vec	Вектор выборки
-----	----------------

Возвращает

Последовательность битов

См. определение в файле [nist_funcs.cpp](#) строка 18

```
00019 {
00020     int max_len = 1 + std::log2(MAX_LIM); // Находим наибольший бит, который может принять значение 1
00021                                           // Старшие биты всегда будут 0, что будет нарушать тесты, поэтому обрубам их
00022
00023     std::vector<bool> bit_seq;
00024
00025     for (auto num : vec)
00026     {
00027         for (int i=0; i < max_len; ++i)
00028         {
00029             bit_seq.push_back(num » i & 1); // биты хранятся в обратном порядке, т.е. чем меньше индекс, тем младше бит
00030         }
00031     }
00032
00033     return bit_seq;
00034
00035
00036 }
```

6.14 nist_funcs.cpp

См. документацию.

```
00001
00005
00006 #include <cmath>
00007
00008 #include "../include/tests.h"
00009 #include "../include/defs.h"
00010
00011
00012 #include<iostream>
00013
00014
00018 std::vector<bool> sample_to_bit_sequence ( std::vector<unsigned int>& vec )
00019 {
00020     int max_len = 1 + std::log2(MAX_LIM); // Находим наибольший бит, который может принять значение 1
00021                                           // Старшие биты всегда будут 0, что будет нарушать тесты, поэтому обрубам их
00022
00023     std::vector<bool> bit_seq;
00024
00025     for (auto num : vec)
00026     {
00027         for (int i=0; i < max_len; ++i)
00028         {
00029             bit_seq.push_back(num » i & 1); // биты хранятся в обратном порядке, т.е. чем меньше индекс, тем младше бит
00030         }
00031     }
00032
00033     return bit_seq;
00034
00035
00036 }
00037
```

```

00038
00043 std::vector<unsigned int> get_pattern_counts ( std::vector<bool>& bit_seq, int m )
00044 {
00045     std::vector<unsigned int> pattern_counts(1«m, 0);
00046
00047     for (int i=0; i < bit_seq.size(); ++i)
00048     {
00049         int k=0;
00050         for (int j = m-1; j >= 0; --j) // Проходимся в обратном порядке для эстетичности, чтобы паттерн равный числу I
            был записан под индексом I
00051         {
00052             if (bit_seq.at((i+j) % bit_seq.size())) k = 2*k + 1;
00053             else k = 2*k;
00054         }
00055         ++pattern_counts[k];
00056     }
00057
00058     return pattern_counts;
00059 }
00060
00061
00066 double get_psi2 ( std::vector<bool>& bit_seq, int m )
00067 {
00068     std::vector<unsigned int> pattern_counts = get_pattern_counts(bit_seq, m);
00069
00070
00071     int expected = bit_seq.size() / (1 « m);
00072
00073     double sum = 0;
00074     for ( auto count : pattern_counts )
00075     {
00076         sum += (count - expected) * (count - expected);
00077     }
00078
00079     sum = ( sum * (1 « m) ) / bit_seq.size() - bit_seq.size();
00080
00081     return sum;
00082 }
00083
00084
00089 double get_apEn ( std::vector<bool>& bit_seq, int m )
00090 {
00091     std::vector<unsigned int> pattern_counts = get_pattern_counts(bit_seq, m);
00092
00093     double apEn = 0;
00094
00095     for ( auto count : pattern_counts )
00096     {
00097         double p = (double)count / bit_seq.size();
00098         apEn += (p > std::pow(10,-6)) ? (p * std::log(p)) : 0;
00099     }
00100
00101     return apEn;
00102 }
00103
00104

```

6.15 Файл src/nist_tests.cpp

Файл, содержащий реализации 5 nist-тестов.

```

#include <cmath>
#include <algorithm>
#include "../include/tests.h"
#include "../include/defs.h"
#include "../include/external/cephes.h"
#include <iostream>

```

Функции

- double `nist_frequency` (std::vector< bool > &bit_seq)
Частотный побитовый тест

- double `nist_runs` (`std::vector< bool > &bit_seq`)
Тест на последовательность одинаковых битов
- double `nist_serial` (`std::vector< bool > &bit_seq`, `int m`)
Тест на периодичность
- double `nist_apEntropy` (`std::vector< bool > &bit_seq`, `int m`)
Тест приближенной энтропии
- double `nist_cusum` (`std::vector< bool > &bit_seq`)
Тест кумулятивных сумм

6.15.1 Подробное описание

Файл, содержащий реализации 5 nist-тестов.

Реализованные nist-тесты:

- Частотный побитовый тест
- Тест на последовательность одинаковых битов
- Тест на периодичность
- Тест приближенной энтропии
- Тест кумулятивных сумм

См. определение в файле `nist_tests.cpp`

6.15.2 Функции

6.15.2.1 `nist_apEntropy()`

```
double nist_apEntropy (
    std::vector< bool > & bit_seq,
    int m)
```

Тест приближенной энтропии

Аргументы

<code>bit_seq</code>	Последовательность битов выборки
<code>m</code>	Длина рассматриваемого блока бит

Возвращает

`p_value`

См. определение в файле `nist_tests.cpp` строка 94

```
00095 {
00096     double apEn_m = get_apEn(bit_seq, m);
00097     double apEn_m1 = get_apEn(bit_seq, m+1);
00098
00099     double apEn = apEn_m - apEn_m1;
00100     double chi2 = 2 * bit_seq.size() * (std::log(2) - apEn);
00101
00102     double p_value = cephes_igamc(1 « (m-1), chi2/2);
00103     return p_value;
00104 }
```

6.15.2.2 nist_cusum()

```
double nist_cusum (  
    std::vector< bool > & bit_seq)
```

Тест кумулятивных сумм

Аргументы

bit_seq	Последовательность битов выборки
---------	----------------------------------

Возвращает

p_value

См. определение в файле [nist_tests.cpp](#) строка 109

```
00110 {
00111     int S = 0;
00112     int max_S = 0;
00113
00114
00115     for ( auto bit : bit_seq )
00116     {
00117         bit ? ++S : --S;
00118         max_S = (std::abs(S) > max_S) ? std::abs(S) : max_S;
00119     }
00120
00121     double p_value = std::erfc(max_S / std::sqrt(2 * bit_seq.size()));
00122
00123     return p_value;
00124 }
```

6.15.2.3 nist_frequency()

```
double nist_frequency (
    std::vector< bool > & bit_seq)
```

Частотный побитовый тест

Аргументы

bit_seq	Последовательность битов выборки
---------	----------------------------------

Возвращает

p_value

См. определение в файле [nist_tests.cpp](#) строка 29

```
00030 {
00031     int sum = 0;
00032
00033     for (auto bit : bit_seq)
00034     {
00035         bit ? ++sum : --sum; // Итого: ++sum, если бит - 1, иначе --sum
00036     }
00037
00038     double s_obs = std::abs(sum) / std::sqrt(bit_seq.size());
00039     double p_value = std::erfc(s_obs / std::sqrt(2));
00040
00041     return p_value;
00042 }
```

6.15.2.4 nist_runs()

```
double nist_runs (
    std::vector< bool > & bit_seq)
```

Тест на последовательность одинаковых битов

Аргументы

bit_seq	Последовательность битов выборки
---------	----------------------------------

Возвращает

p_value

См. определение в файле [nist_tests.cpp](#) строка 48

```

00049 {
00050
00051     int sum = std::count(bit_seq.begin(), bit_seq.end(), true);
00052
00053     double pi = (double)sum / bit_seq.size();
00054
00055     if (fabs(pi - 0.5) >= 2 / std::sqrt(bit_seq.size()))
00056     {
00057         std::cerr << "> Не выполнен критерий для runs теста" << std::endl;
00058         return 0;
00059     }
00060
00061
00062     int runs = 1;
00063     for (int i=1; i < bit_seq.size(); ++i)
00064     {
00065         if (bit_seq.at(i) != bit_seq.at(i-1))
00066             ++runs;
00067     }
00068
00069     double erfc_arg = fabs(runs - 2.0 * bit_seq.size() * pi * (1-pi)) / (2.0 * pi * (1-pi) * sqrt(2*bit_seq.size()));
00070     double p_value = erfc(erfc_arg);
00071
00072     return p_value;
00073 }
00074 }
```

6.15.2.5 nist_serial()

```

double nist_serial (
    std::vector< bool > & bit_seq,
    int m)
```

Тест на периодичность

Аргументы

bit_seq	Последовательность битов выборки
m	Длина рассматриваемого блока бит

Возвращает

p_value

См. определение в файле [nist_tests.cpp](#) строка 80

```

00081 {
00082     double psi_m = get_psi2(bit_seq, m);
00083     double psi_m_1 = get_psi2(bit_seq, m-1);
00084
00085
00086     double p_value = cephes_igamc(1 << (m-2), (psi_m - psi_m_1)/2);
00087     return p_value;
00088 }
```

6.16 nist_tests.cpp

[См. документацию.](#)

```

00001
00012
00013 #include <cmath>
00014 #include <algorithm>
00015
00016 #include "../include/tests.h"
00017 #include "../include/defs.h"
00018
00019 #include "../include/external/cephes.h"
00020
00021
00022 #include<iostream>
00023
00024
00025
00029 double nist_frequency ( std::vector<bool>& bit_seq )
00030 {
00031     int sum = 0;
00032
00033     for (auto bit : bit_seq)
00034     {
00035         bit ? ++sum : --sum; // Итого: ++sum, если бит - 1, иначе --sum
00036     }
00037
00038     double s_obs = std::abs(sum) / std::sqrt(bit_seq.size());
00039     double p_value = std::erfc(s_obs / std::sqrt(2));
00040
00041     return p_value;
00042 }
00043
00044
00048 double nist_runs ( std::vector<bool>& bit_seq )
00049 {
00050
00051     int sum = std::count(bit_seq.begin(), bit_seq.end(), true);
00052
00053     double pi = (double)sum / bit_seq.size();
00054
00055     if (fabs(pi - 0.5) >= 2 / std::sqrt(bit_seq.size()))
00056     {
00057         std::cerr << "> Не выполнен критерий для runs теста" << std::endl;
00058         return 0;
00059     }
00060
00061
00062     int runs = 1;
00063     for (int i=1; i < bit_seq.size(); ++i)
00064     {
00065         if (bit_seq.at(i) != bit_seq.at(i-1))
00066             ++runs;
00067     }
00068
00069     double erfc_arg = fabs(runs - 2.0 * bit_seq.size() * pi * (1-pi)) / (2.0 * pi * (1-pi) * sqrt(2*bit_seq.size()));
00070     double p_value = erfc(erfc_arg);
00071
00072     return p_value;
00073
00074 }
00075
00080 double nist_serial ( std::vector<bool>& bit_seq, int m )
00081 {
00082     double psi_m = get_psi2(bit_seq, m);
00083     double psi_m_1 = get_psi2(bit_seq, m-1);
00084
00085
00086     double p_value = cephes_igamc(1 < (m-2), (psi_m - psi_m_1)/2);
00087     return p_value;
00088 }
00089
00094 double nist_apEntropy ( std::vector<bool>& bit_seq, int m )
00095 {
00096     double apEn_m = get_apEn(bit_seq, m);
00097     double apEn_m1 = get_apEn(bit_seq, m+1);
00098
00099     double apEn = apEn_m - apEn_m1;
00100     double chi2 = 2 * bit_seq.size() * (std::log(2) - apEn);
00101
00102     double p_value = cephes_igamc(1 < (m-1), chi2/2);
00103     return p_value;
00104 }
00105
00109 double nist_cusum ( std::vector<bool>& bit_seq )

```

```

00110 {
00111     int S = 0;
00112     int max_S = 0;
00113
00114
00115     for ( auto bit : bit_seq )
00116     {
00117         bit ? ++S : --S;
00118         max_S = (std::abs(S) > max_S) ? std::abs(S) : max_S;
00119     }
00120
00121     double p_value = std::erfc(max_S / std::sqrt(2 * bit_seq.size()));
00122
00123     return p_value;
00124 }
00125
00126

```

6.17 Файл src/prng.cpp

Файл, с определением методов классов генераторов

```

#include <iostream>
#include "../include/prng.h"

```

6.17.1 Подробное описание

Файл, с определением методов классов генераторов

См. определение в файле [prng.cpp](#)

6.18 prng.cpp

[См. документацию.](#)

```

00001
00005
00006 #include <iostream>
00007
00008 #include "../include/prng.h"
00009
00010
00012 //          PRNG          //
00014
00015
00019 PRNG::PRNG ( uint32_t min_lim, uint32_t max_lim )
00020 : min_lim(min_lim), max_lim(max_lim) {}
00021
00022
00026 void PRNG::generate_sample (std::ostream& out, int size)
00027 {
00028     for (int i=0; i < size; ++i)
00029     {
00030         out << this->generate() << std::endl;
00031     }
00032 }
00033
00034
00038 uint32_t PRNG::result (uint32_t result) const
00039 {
00040     return min_lim + result % (1ul + max_lim - min_lim); // поскольку max_lim - верхняя граница включительно, то
        //              нужно брать модуль +1
00041
        // (1 - unsigned long, чтобы не возникло переполнения и не было деления на 0)
00042 }
00043
00044
00045
00046

```



```

00048 //      mid_xor      //
00050
00055 mid_xor_PRNG::mid_xor_PRNG (uint32_t seed, uint32_t min_lim, uint32_t max_lim)
00056 : PRNG(min_lim, max_lim), seed(seed) {}
00057
00058
00061 uint32_t mid_xor_PRNG::generate ()
00062 {
00063     uint64_t product = seed * seed;
00064
00065     seed = (product » 8) & 0xFFFFFFFF;
00066
00067     // Динамический сдвиг для XorShift
00068     uint8_t shift = (seed » 10) & 0x1F; // Сдвиг от 0 до 31
00069     seed ^= 0x9E3779B9 « (shift % 13 + 1); // Чтобы не было вырождения в 0
00070     seed ^= seed » (shift % 17 + 1);
00071
00072     return result(seed);
00073 }
00074
00075
00076
00078 //      mul_xor      //
00080
00085 mul_xor_PRNG::mul_xor_PRNG (uint32_t seed, uint32_t min_lim, uint32_t max_lim)
00086 : PRNG(min_lim, max_lim), seed(seed) {}
00087
00088
00091 uint32_t mul_xor_PRNG::generate ()
00092 {
00093     seed ^= seed « 13;
00094     seed ^= (seed » 7) * 0x9AE77B3D;
00095     seed ^= (seed » 5) | (seed « 17);
00096
00097     return result(seed);
00098 }
00099
00100
00101
00103 //      LCG      //
00105
00110 LCG::LCG (uint32_t seed, uint32_t min_lim, uint32_t max_lim)
00111 // : PRNG(min_lim, max_lim), seed(seed), k((1«16)+1), b((1«8)-1), M(1«31) {}
00112 : PRNG(min_lim, max_lim), seed(seed), k(1'103'515'245), b(12345), M(1«31) {}
00113
00114
00117 uint32_t LCG::generate ()
00118 {
00119     seed = (seed * k + b) & (M-1); // Если M = 2^N
00120     // seed = (seed * k + b) % (M); // иначе
00121
00122     // (k-1) - делится на все простые делители M (2)
00123     // b и M взаимно простые
00124     // M делится на 4 и (k-1) делится на 4
00125     //
00126     // Поэтому у этого линейного конгруэнтного генератора максимальная периодичность
00127
00128     return result(seed);
00129 }
00130
00131
00132
00133
00134
00136 //      div      //
00138
00143 div_PRNG::div_PRNG (uint32_t seed, uint32_t min_lim, uint32_t max_lim)
00144 : PRNG(min_lim, max_lim), seed(seed), prev(0x9AE77B3D) {}
00145
00146
00149 uint32_t div_PRNG::generate ()
00150 {
00151     // Определяем меньшее и большее значения между seed и prev
00152     uint32_t min = (prev > seed) ? seed : prev;
00153     uint32_t max = (min != seed) ? seed : prev;
00154
00155     bool flag = (seed == min); // флаг, чтобы понимать, какие изменения проводятся над переменными
00156
00157     // Изменения, чтобы не возникло проблем с делением
00158     if (min == UINT32_MAX) min = 0x9AE77B3D;
00159     if (min == max)
00160         max = (((unsigned long)max + min) > UINT32_MAX) ? UINT32_MAX : max + min;
00161
00162     double div = double(min) / max; // Число в диапазоне (0,1)
00163
00164     seed = div * (1ul « 48); // Извлекаем 32 бита после запятой, начиная с 16 бита (первые 16 битов обрубаются за счёт
00165

```

```

        типа seed)
00166
00167 if ( seed == 0 ) seed = 0x9AE77B3D; // Избавляемся от выражения в 0
00168 prev = (flag ? min : max); // Записываем предыдущее значение seed, включая преобразования
00169
00170 return result(seed);
00171 }
00172

```

6.19 Файл src/tests.cpp

Файл, в котором определяются все функции, производящие оценки выборок и хи-квадрат тест

```

#include <cmath>
#include <algorithm>
#include "../include/tests.h"

```

Функции

- double `get_mean` (std::vector< unsigned int > &vec)
Оценка выборки на среднее значение
- double `get_stdDev` (std::vector< unsigned int > &vec)
Оценка выборки на стандартное отклонение
- double `get_cv` (std::vector< unsigned int > &vec)
Оценка выборки на коэффициент вариации
- std::pair< double, int > `get_chi2` (std::vector< unsigned int > &vec)
Тест хи-квадрат

6.19.1 Подробное описание

Файл, в котором определяются все функции, производящие оценки выборок и хи-квадрат тест

См. определение в файле [tests.cpp](#)

6.19.2 Функции

6.19.2.1 `get_chi2()`

```

std::pair< double, int > get_chi2 (
    std::vector< unsigned int > & vec)

```

Тест хи-квадрат

Аргументы

vec	Вектор выборки
-----	----------------

Возвращает

Пара значений (значение хи-квадрат, количество степеней свободы)

См. определение в файле [tests.cpp](#) строка 56

```
00057 {
00058     if (vec.empty())
00059         return {-1,-1};
00060
00061     auto min_val = *std::min_element(vec.begin(), vec.end()); // Наименьший элемент выборки
00062     auto max_val = *std::max_element(vec.begin(), vec.end()); // Наибольший элемент выборки
00063
00064     int bin_number = 1 + std::log2(vec.size()); // Количество промежутков по правилу Стерджеса
00065     float bin_width = (float)(max_val-min_val)/bin_number; // Ширина промежутка
00066
00067     float expected = (float)vec.size() / bin_number; // Количество ожидаемых наблюдений в каждом промежутке
00068     std::vector<int> observed(bin_number, 0); // Количество фактических наблюдений в каждом промежутке
00069
00070
00071     for ( auto val : vec )
00072     {
00073         int bin_ind = (int) ((val - min_val) / bin_width);
00074         if (bin_ind >= bin_number) --bin_ind;
00075
00076         ++observed.at(bin_ind);
00077     }
00078
00079     double chi2 = 0;
00080
00081     for ( auto obs : observed )
00082     {
00083         chi2 += (obs - expected) * (obs - expected) / expected;
00084     }
00085
00086     return {chi2, bin_number-1};
00087
00088 }
00089 }
```

6.19.2.2 get_cv()

```
double get_cv (
    std::vector< unsigned int > & vec)
```

Оценка выборки на коэффициент вариации

Аргументы

vec	Вектор выборки
-----	----------------

Возвращает

Коэффициент вариации

См. определение в файле [tests.cpp](#) строка 47

```
00048 {
00049     return get_stdDev(vec) / get_mean(vec);
00050 }
```

6.19.2.3 get_mean()

```
double get_mean (
    std::vector< unsigned int > & vec)
```

Оценка выборки на среднее значение

Аргументы

vec	Вектор выборки
-----	----------------

Возвращает

Среднее значение

См. определение в файле [tests.cpp](#) строка 14

```
00015 {
00016     double sum = 0;
00017
00018     for ( auto& num : vec )
00019     {
00020         sum += num;
00021     }
00022
00023     return sum / vec.size();
00024 }
```

6.19.2.4 get_stdDev()

```
double get_stdDev (
    std::vector< unsigned int > & vec)
```

Оценка выборки на стандартное отклонение

Аргументы

vec	Вектор выборки
-----	----------------

Возвращает

Стандартное отклонение

См. определение в файле [tests.cpp](#) строка 30

```
00031 {
00032     unsigned long sum = 0;
00033     auto mean = get_mean(vec);
00034
00035     for ( auto& num : vec )
00036     {
00037         sum += (num - mean) * (num - mean);
00038     }
00039
00040     return std::sqrt(sum / vec.size());
00041 }
```

6.20 tests.cpp

См. документацию.

```
00001
00005
00006 #include <cmath>
00007 #include <algorithm>
00008
00009 #include "../include/tests.h"
00010
00014 double get_mean ( std::vector<unsigned int>& vec )
```

```

00015 {
00016     double sum = 0;
00017
00018     for ( auto& num : vec )
00019     {
00020         sum += num;
00021     }
00022
00023     return sum / vec.size();
00024 }
00025
00026
00030 double get_stdDev ( std::vector<unsigned int>& vec )
00031 {
00032     unsigned long sum = 0;
00033     auto mean = get_mean(vec);
00034
00035     for ( auto& num : vec )
00036     {
00037         sum += (num - mean) * (num - mean);
00038     }
00039
00040     return std::sqrt(sum / vec.size());
00041 }
00042
00043
00047 double get_cv ( std::vector<unsigned int>& vec )
00048 {
00049     return get_stdDev(vec) / get_mean(vec);
00050 }
00051
00052
00056 std::pair<double,int> get_chi2 ( std::vector<unsigned int>& vec )
00057 {
00058     if (vec.empty())
00059         return {-1,-1};
00060
00061     auto min_val = *std::min_element(vec.begin(), vec.end()); // Наименьший элемент выборки
00062     auto max_val = *std::max_element(vec.begin(), vec.end()); // Наибольший элемент выборки
00063
00064     int bin_number = 1 + std::log2(vec.size()); // Количество промежутков по правилу Стерджеса
00065     float bin_width = (float)(max_val-min_val)/bin_number; // Ширина промежутка
00066
00067     float expected = (float)vec.size() / bin_number; // Количество ожидаемых наблюдений в каждом промежутке
00068     std::vector<int> observed(bin_number, 0); // Количество фактических наблюдений в каждом промежутке
00069
00070
00071     for ( auto val : vec )
00072     {
00073         int bin_ind = (int) ((val - min_val) / bin_width);
00074         if (bin_ind >= bin_number) --bin_ind;
00075
00076         ++observed.at(bin_ind);
00077     }
00078
00079
00080     double chi2 = 0;
00081
00082     for ( auto obs : observed )
00083     {
00084         chi2 += (obs - expected) * (obs - expected) / expected;
00085     }
00086
00087     return {chi2, bin_number-1};
00088 }
00089 }
00090
00091

```

6.21 test.cpp

```

00001 #include <iostream>
00002
00003 int main()
00004 {
00005     double div = 1./4;
00006
00007     std::cout << div << "\n";
00008     std::cout << (div << 3) << "\n";
00009     std::cout << (uint32_t) (div * (1ul << 32)) << "\n";
00010 }

```

6.22 Файл `time_check.cpp`

Файл, в котором производится измерение времени генерации выборок разных размеров разными генераторами и одним из стандартных генераторов

```
#include <fstream>
#include <vector>
#include <chrono>
#include <experimental/random>
#include "include/prng.h"
#include "include/defs.h"
```

Функции

- void `check_time` (std::ostream &out, int size)
Функция, в которой производится измерение и запись времени генерации выборки для каждого из генераторов
- int `main` ()
Основная функция, в которой происходит измерение времени генерации выборок разных размеров всеми генераторами и одним из стандартных генераторов

6.22.1 Подробное описание

Файл, в котором производится измерение времени генерации выборок разных размеров разными генераторами и одним из стандартных генераторов

См. определение в файле `time_check.cpp`

6.22.2 Функции

6.22.2.1 `check_time()`

```
void check_time (
    std::ostream & out,
    int size)
```

Функция, в которой производится измерение и запись времени генерации выборки для каждого из генераторов

Аргументы

out	Поток, куда будет записан результат
size	Размер генерируемой выборки

См. определение в файле `time_check.cpp` строка 44

```

00045 {
00046     namespace ch = std::chrono;           // Для удобства
00047     using clock = ch::high_resolution_clock; // Для удобства
00048
00049     // Записываем размер выборки
00050     out << size << ',';
00051
00052     ch::time_point<clock> start, end; // Метки начала и конца измерения
00053     std::ofstream trash("/dev/null"); // Мусорка, чтобы не хранить сгенерированные выборки
00054
00055
00056     // Замер для генератора mid_xor
00057     {
00058         mid_xor_PRNG prng(42, MIN_LIM, MAX_LIM); // Создание генератора
00059         start = clock::now();                     // Начало измерения
00060         prng.generate_sample(trash, size);         // Генерация выборки
00061         end = clock::now();                       // Конец измерения
00062
00063         out << ch::duration_cast<ch::microseconds>(end-start).count() << ','; // Запись результата
00064     }
00065
00066
00067     // Замер для генератора mul_xor
00068     {
00069         mul_xor_PRNG prng(42, MIN_LIM, MAX_LIM); // Создание генератора
00070         start = clock::now();                     // Начало измерения
00071         prng.generate_sample(trash, size);         // Генерация выборки
00072         end = clock::now();                       // Конец измерения
00073
00074         out << ch::duration_cast<ch::microseconds>(end-start).count() << ','; // Запись результата
00075     }
00076
00077
00078     // Замер для генератора LCG
00079     {
00080         LCG prng(42, MIN_LIM, MAX_LIM);           // Создание генератора
00081         start = clock::now();                     // Начало измерения
00082         prng.generate_sample(trash, size);         // Генерация выборки
00083         end = clock::now();                       // Конец измерения
00084
00085         out << ch::duration_cast<ch::microseconds>(end-start).count() << ','; // Запись результата
00086     }
00087
00088
00089     // Замер для генератора randint
00090     {
00091         std::srand(42);                          // Задание семени
00092         start = clock::now();                     // Начало измерения
00093         for (int i=0; i<size; ++i)                // Генерация выборки
00094             std::experimental::randint(MIN_LIM, MAX_LIM);
00095         end = clock::now();                       // Конец измерения
00096
00097         out << ch::duration_cast<ch::microseconds>(end-start).count() << '\n'; // Запись результата
00098     }
00099 }

```

6.22.2.2 main()

```
int main ()
```

Основная функция, в которой происходит измерение времени генерации выборок разных размеров всеми генераторами и одним из стандартных генераторов

Возвращает

См. определение в файле `time_check.cpp` строка 21

```

00022 {
00023     // Файл для записи результатов

```

```

00024 std::ofstream time_results("generation_time.csv");
00025
00026 time_results << "Size,Mid_xor,Mul_xor,LCG,randint\n";
00027
00028 // Список размеров выборок
00029 std::vector<int> sizes = { 1'000, 2'000, 3'000, 5'000, 10'000,
00030                          20'000, 30'000, 50'000, 75'000, 100'000,
00031                          200'000, 300'000, 500'000, 750'000, 1'000'000};
00032
00033
00034 // Цикл измерения и записи времени генерации выборок для разных размеров
00035 for ( auto size : sizes )
00036     check_time (time_results, size);
00037
00038 }

```

6.23 time_check.cpp

[См. документацию.](#)

```

00001
00005
00006 #include <fstream>
00007 #include <vector>
00008
00009 #include <chrono>
00010 #include <experimental/random>
00011
00012 #include "include/prng.h"
00013 #include "include/defs.h"
00014
00015
00016 void check_time (std::ostream&, int);
00017
00018
00021 int main()
00022 {
00023     // Файл для записи результатов
00024     std::ofstream time_results("generation_time.csv");
00025
00026     time_results << "Size,Mid_xor,Mul_xor,LCG,randint\n";
00027
00028     // Список размеров выборок
00029     std::vector<int> sizes = { 1'000, 2'000, 3'000, 5'000, 10'000,
00030                             20'000, 30'000, 50'000, 75'000, 100'000,
00031                             200'000, 300'000, 500'000, 750'000, 1'000'000};
00032
00033
00034     // Цикл измерения и записи времени генерации выборок для разных размеров
00035     for ( auto size : sizes )
00036         check_time (time_results, size);
00037
00038 }
00039
00040
00044 void check_time (std::ostream& out, int size)
00045 {
00046     namespace ch = std::chrono;           // Для удобства
00047     using clock = ch::high_resolution_clock; // Для удобства
00048
00049     // Записываем размер выборки
00050     out << size << ' ';
00051
00052     ch::time_point<clock> start, end; // Метки начала и конца измерения
00053     std::ofstream trash("/dev/null"); // Мусорка, чтобы не хранить сгенерированные выборки
00054
00055
00056     // Замер для генератора mid_xor
00057     {
00058         mid_xor_PRNG prng(42, MIN_LIM, MAX_LIM); // Создание генератора
00059         start = clock::now();                     // Начало измерения
00060         prng.generate_sample(trash, size);         // Генерация выборки
00061         end = clock::now();                       // Конец измерения
00062
00063         out << ch::duration_cast<ch::microseconds>(end-start).count() << ' '; // Запись результата
00064     }
00065
00066
00067     // Замер для генератора mul_xor
00068     {
00069         mul_xor_PRNG prng(42, MIN_LIM, MAX_LIM); // Создание генератора
00070         start = clock::now();                     // Начало измерения

```



```

00071 prng.generate_sample(trash, size); // Генерация выборки
00072 end = clock::now(); // Конец измерения
00073
00074 out « ch::duration_cast<ch::microseconds>(end-start).count() « ','; // Запись результата
00075 }
00076
00077 // Замер для генератора LCG
00078 {
00079     LCG prng(42, MIN_LIM, MAX_LIM); // Создание генератора
00080     start = clock::now(); // Начало измерения
00081     prng.generate_sample(trash, size); // Генерация выборки
00082     end = clock::now(); // Конец измерения
00083
00084     out « ch::duration_cast<ch::microseconds>(end-start).count() « ','; // Запись результата
00085 }
00086
00087 // Замер для генератора randint
00088 {
00089     std::srand(42); // Задание семени
00090     start = clock::now(); // Начало измерения
00091     for (int i=0; i<size; ++i) // Генерация выборки
00092         std::experimental::randint(MIN_LIM, MAX_LIM);
00093     end = clock::now(); // Конец измерения
00094
00095     out « ch::duration_cast<ch::microseconds>(end-start).count() « '\n'; // Запись результата
00096 }
00097 }
00098 }
00099 }
00100

```

6.24 time_graph.py

```

00001 import pandas as pd
00002 import matplotlib.pyplot as plt
00003
00004
00005 res_filepath = "./generation_time.csv"
00006 res_picpath = "./generation_time.png"
00007
00008
00009 results = pd.read_csv(res_filepath)
00010
00011 names = results.columns[1:]
00012
00013
00014 plt.figure(figsize=(10,10),layout="tight")
00015
00016
00017
00018 plt.subplot(211)
00019
00020 plt.title("Зависимость времени генерации от объёма выборки для различных методов")
00021
00022 for name in names:
00023     plt.plot(results["Size"], results[name], "--o")
00024
00025 plt.grid(linestyle=":")
00026 plt.legend(results.columns[1:])
00027
00028 plt.xlabel("Количество элементов")
00029 plt.ylabel("Время генерации, µs")
00030
00031
00032 plt.subplot(212)
00033
00034 for name in names:
00035     plt.plot(results["Size"], results[name], "--o")
00036
00037 plt.grid(linestyle=":")
00038 plt.legend(results.columns[1:])
00039
00040 plt.xlabel("Количество элементов")
00041 plt.ylabel("Время генерации, µs")
00042
00043 plt.yscale("log")
00044
00045
00046 plt.savefig(res_picpath)

```


Предметный указатель

- ALPHA
 - defs.h, [32](#)
- AP_ENTROPY_M
 - defs.h, [32](#)
- b
 - LCG, [14](#)
- check_time
 - time_check.cpp, [64](#)
- create_samples
 - create_samples.cpp, [24](#)
- create_samples.cpp, [23](#)
 - create_samples, [24](#)
 - main, [25](#)
- defs.h
 - ALPHA, [32](#)
 - AP_ENTROPY_M, [32](#)
 - MAX_LIM, [32](#)
 - MIN_LIM, [32](#)
 - SAMPLES_DIR, [33](#)
 - SERIAL_M, [33](#)
- div_PRNG, [9](#)
 - div_PRNG, [10](#)
 - generate, [11](#)
 - prev, [11](#)
 - seed, [11](#)
- generate
 - div_PRNG, [11](#)
 - LCG, [13](#)
 - mid_xor_PRNG, [16](#)
 - mul_xor_PRNG, [18](#)
 - PRNG, [20](#)
- generate_sample
 - PRNG, [20](#)
- get_apEn
 - nist_funcs.cpp, [49](#)
 - tests.h, [36](#)
- get_chi2
 - tests.cpp, [60](#)
 - tests.h, [37](#)
- get_cv
 - tests.cpp, [61](#)
 - tests.h, [37](#)
- get_mean
 - tests.cpp, [61](#)
 - tests.h, [39](#)
- get_pattern_counts
 - nist_funcs.cpp, [49](#)
 - tests.h, [39](#)
- get_psi2
 - nist_funcs.cpp, [50](#)
 - tests.h, [40](#)
- get_results.cpp, [26](#)
 - main, [27](#)
 - process_sample, [28](#)
- get_stdDev
 - tests.cpp, [62](#)
 - tests.h, [40](#)
- include/defs.h, [32](#), [33](#)
- include/external/cephes.h, [33](#)
- include/prng.h, [34](#)
- include/tests.h, [35](#), [44](#)
- k
 - LCG, [14](#)
- LCG, [12](#)
 - b, [14](#)
 - generate, [13](#)
 - k, [14](#)
 - LCG, [13](#)
 - M, [14](#)
 - seed, [14](#)
- M
 - LCG, [14](#)
- main
 - create_samples.cpp, [25](#)
 - get_results.cpp, [27](#)
 - time_check.cpp, [65](#)
- MAX_LIM
 - defs.h, [32](#)
- max_lim
 - PRNG, [21](#)
- mid_xor_PRNG, [14](#)
 - generate, [16](#)
 - mid_xor_PRNG, [15](#)
 - seed, [16](#)
- MIN_LIM
 - defs.h, [32](#)
- min_lim
 - PRNG, [21](#)
- mul_xor_PRNG, [17](#)
 - generate, [18](#)
 - mul_xor_PRNG, [18](#)
 - seed, [19](#)

- nist_apEntropy
 - nist_tests.cpp, 53
 - tests.h, 41
- nist_cusum
 - nist_tests.cpp, 53
 - tests.h, 41
- nist_frequency
 - nist_tests.cpp, 55
 - tests.h, 42
- nist_funcs.cpp
 - get_apEn, 49
 - get_pattern_counts, 49
 - get_psi2, 50
 - sample_to_bit_sequence, 50
- nist_runs
 - nist_tests.cpp, 55
 - tests.h, 42
- nist_serial
 - nist_tests.cpp, 56
 - tests.h, 43
- nist_tests.cpp
 - nist_apEntropy, 53
 - nist_cusum, 53
 - nist_frequency, 55
 - nist_runs, 55
 - nist_serial, 56
- prev
 - div_PRNG, 11
- PRNG, 19
 - generate, 20
 - generate_sample, 20
 - max_lim, 21
 - min_lim, 21
 - PRNG, 20
 - result, 20
- process_sample
 - get_results.cpp, 28
- readme, 1
- result
 - PRNG, 20
- sample_to_bit_sequence
 - nist_funcs.cpp, 50
 - tests.h, 43
- SAMPLES_DIR
 - defs.h, 33
- seed
 - div_PRNG, 11
 - LCG, 14
 - mid_xor_PRNG, 16
 - mul_xor_PRNG, 19
- SERIAL_M
 - defs.h, 33
- src/external/cephes.cpp, 44
- src/nist_funcs.cpp, 48, 51
- src/nist_tests.cpp, 52, 57
- src/prng.cpp, 58
- src/tests.cpp, 60, 62
- tests.cpp
 - get_chi2, 60
 - get_cv, 61
 - get_mean, 61
 - get_stdDev, 62
- tests.h
 - get_apEn, 36
 - get_chi2, 37
 - get_cv, 37
 - get_mean, 39
 - get_pattern_counts, 39
 - get_psi2, 40
 - get_stdDev, 40
 - nist_apEntropy, 41
 - nist_cusum, 41
 - nist_frequency, 42
 - nist_runs, 42
 - nist_serial, 43
 - sample_to_bit_sequence, 43
- time_check.cpp, 64
 - check_time, 64
 - main, 65