student_t_distribution Class

Visual Studio 2015

For the latest documentation on Visual Studio 2017, see Visual Studio 2017 Documentation.

Generates a Student's *t*-distribution.

Syntax

```
class student_t_distribution
   {
   public:  // types
   typedef RealType result_type;
   struct param_type;  // constructor and reset functions
   explicit student_t_distribution(RealType n = 1.0);
   explicit student_t_distribution(const param_type& parm);
   void reset();
   // generating functions
   template <class URNG>
   result_type operator()(URNG& gen);
   template <class URNG>
   result_type operator()(URNG& gen, const param_type& parm);
   // property functions
   RealType n() const;
   param_type param() const;
   void param(const param_type& parm);
   result_type min() const;
   result_type max() const;
   };
```

Parameters

`RealType`

The floating-point result type, defaults to `double`. For possible types, see <random>.

Remarks

The template class describes a distribution that produces values of a user-specified integral type, or type `double` if none is provided, distributed according to the Student's *t*-Distribution. The following table links to articles about individual members.

student_t_distribution::student_t_distribution	`student_t_distribution::n`	`student_t_distribution::param`
`student_t_distribution::operator()`		student_t_distribution::param_type

The property function `n()` returns the value for the stored distribution parameter `n`.

For more information about distribution classes and their members, see <random>.

For detailed information about the Student's *t*-distribution, see the Wolfram MathWorld article Students t-Distribution.

Example

C++

```
// compile with: /EHsc /W4
#include <random>
#include <iostream>
#include <iomanip>
#include <string>
#include <map>

void test(const double n, const int s) {

    // uncomment to use a non-deterministic generator
    //    std::random_device gen;
    std::mt19937 gen(1701);

    std::student_t_distribution<> distr(n);

    std::cout << std::endl;
    std::cout << "min() == " << distr.min() << std::endl;
    std::cout << "max() == " << distr.max() << std::endl;
    std::cout << "n() == " << std::fixed << std::setw(11) << std::setprecision(10) << distr.n() << std::endl;

    // generate the distribution as a histogram
    std::map<double, int> histogram;
    for (int i = 0; i < s; ++i) {
        ++histogram[distr(gen)];
    }

    // print results
    std::cout << "Distribution for " << s << " samples:" << std::endl;
    int counter = 0;
    for (const auto& elem : histogram) {
        std::cout << std::fixed << std::setw(11) << ++counter << ": "
            << std::setw(14) << std::setprecision(10) << elem.first << std::endl;
    }
    std::cout << std::endl;
}

int main()
{
    double n_dist = 0.5;
    int samples = 10;

    std::cout << "Use CTRL-Z to bypass data entry and run using default values." << std::endl;
    std::cout << "Enter a floating point value for the 'n' distribution parameter (must be greater than zero): ";
    std::cin >> n_dist;
    std::cout << "Enter an integer value for the sample count: ";
    std::cin >> samples;

    test(n_dist, samples);
}
```

Output

```
Use CTRL-Z to bypass data entry and run using default values.
Enter a floating point value for the 'n' distribution parameter (must be greater than zero): 1
Enter an integer value for the sample count: 10

min() == -1.79769e+308
max() == 1.79769e+308
n() == 1.0000000000
Distribution for 10 samples:
    1: -1.3084956212
    2: -1.0899518684
    3: -0.9568771388
    4: -0.9372088821
    5: -0.7381334669
    6: -0.2488074854
    7: -0.2028714601
    8: 1.4013074495
    9: 5.3244792236
    10: 92.7084335614
```

Requirements

Header: <random>

Namespace: std

student_t_distribution::student_t_distribution

Constructs the distribution.

```
explicit student_t_distribution(RealType n = 1.0);


explicit student_t_distribution(const param_type& parm);
```

Parameters

`n`
The n distribution parameter.

`parm`
The parameter package used to construct the distribution.

Remarks

Precondition: `0.0 < n`

The first constructor constructs an object whose stored `n` value holds the value `n`.

The second constructor constructs an object whose stored parameters are initialized from `parm`. You can obtain and set the current parameters of an existing distribution by calling the `param()` member function.

student_t_distribution::param_type

Stores all the parameters of the distribution.

C++

```
struct param_type {
   typedef student_t_distribution<RealType> distribution_type;
   param_type(RealType n = 1.0);
   RealType n() const;
   .....
   bool operator==(const param_type& right) const;
   bool operator!=(const param_type& right) const;
```

```
    };
```

Parameters

See parent topic student_t_distribution Class.

Remarks

Precondition: `0.0 < n`

This structure can be passed to the distribution's class constructor at instantiation, to the `param()` member function to set the stored parameters of an existing distribution, and to `operator()` to be used in place of the stored parameters.

See Also

<random>