UCLA Stat 130D
Statistical Computing and Visualization in C++/Java

**Instructor:** Ivo Dinov, Asst. Prof. in Statistics / Neurology

University of California, Los Angeles, Winter 2007

http://www.stat.ucla.edu/~dinov/courses_students.html

# Statistical Methods

## Statistical Computing

- *The birthday example* The birthday paradox is an important example in probability and statistics. If we write down the birth date of every student in the class, and find that even though there is a 1 in 365 chance that any given pair of students have the same birthday, the odds are good that there is a match in the class at large.

## Statistical Computing

```
1 #include <math.h>
2 #include <stdio.h>
3
4 int main(){
5    double noMatch = 1;
6    double matchesMe;
7    int ct;
8     Cout << "People\t MatchesMe\t Any match" << endl;
9    for (ct=2; ct<=40; ct ++){
10          matchesMe = 1- pow(364/365., ct-1);
11          noMatch *= (1 - (ct-1)/365.);
12          cout << ct << " " << matchesMe << " " << (1-noMatch)<< endl;
13  }
14  return 0;
15 }
```

Print the odds that other students share my birthday, and that any two students in the room share a birthday.

## Statistical Computing

- The exact probability is easier to calculate by using the complement—the odds that nobody shares a birthday. The odds that one person does not share the first person's birthday is 364/365; the odds that two people both do not share the first person's birthday is $(364/365)^2$, etc.
- Thus, the odds that among ct-1 additional people, none have the same birthday as the first person is $(1 - (364/365)^{ct-1})$. You can see this calculation on line 10.

## Statistical Computing

- Here is another example of the Birthday problem:
- We stores each day's data in a **struct**. Lines 4-7 define the structure: it will hold one variable indicating the probability of somebody matching the first person's birthday, and one variable giving the probability that no two people share a birthday.

## Statistical Computing

```
#include <math.h>
#include <stdio.h>

    typedef struct {
            double one match;
            double none match;
    }  bday struct;

    int upto = 40;

    int main(){
            bday struct days[upto+1];
            int ct;
            days[1].none match = 1;
            for (ct=2; ct<=upto; ct ++){
                        days[ct].oneMatch = 1- pow(364/365., ct-1);
                        days[ct].noneMatch = days[ct-1].noneMatch * (1 - (ct-1)/365.);
            }
            cout <<"People\t Matches me\t Any match\n" << endl;
            for (ct=2; ct<=upto; ct ++){
                cout << ct << " " << days[ct] << " " " << oneMatch << " " " << 1-days[ct].noneMatch <<endl;
            }
    return 0;
}
```

## GSL

- **#include** <gsl/gsl matrix.h>
- **#include** <stdio.h>
  - **int** main(**void**){
  - gsl_matrix_*m = gsl_matrix_alloc(10,10);
  - gsl vector *v = gsl vector calloc(10);
  - **int** i;
  - **for** (i=0;i< m->size1; i++){
    gsl_matrix_set(m, i, 0, i) ;
  - }
  - printf("Here is point (3,0): %g\n", gsl_matrix_get(m, 3,0));
  - gsl_matrix_set row(m, 3, v);
  - printf("Here is point (3,0) again: %g", gsl_matrix_get(m, 3,0));
  - gsl_matrix_free(m);
  - gsl vector free(v);
- **return** 0;
- http://www.gnu.org/software/gsl/manual/html_node/
- http://gnuwin32.sourceforge.net/packages/gsl.htm

## gsl_matrix_Manipulations

- gsl_matrix_add (a,b); // aij = aij + bij , 8 i, j
- gsl_matrix_sub (a,b); // aij = aij − bij , 8 i, j
- gsl_matrix_mul elements (a,b); // aij = aij · bij , 8 i, j
- gsl_matrix_div elements (a,b);// aij = aij/bij , 8 i, j
- gsl vector add (a,b); // ai = ai + bi, 8 i
- gsl vector sub (a,b); // ai = ai − bij , 8 i
- gsl vector mul (a,b); // ai = ai · bi, 8 i
- gsl vector div (a,b); // ai = ai/bi, 8 i
- gsl_matrix_scale (a,x); // aij = aij · x, 8 i, j, x 2 R
- gsl_matrix_add_constant (a,x); // aij = aij + x, 8 i, j, x 2 R
- gsl_vector_scale_(a,x); // ai = ai · x, 8 i, x 2 R
- gsl_vector_add_constant (a,x); // ai = ai + x, 8 i, x 2 R

## gsl_matrix_Manipulations

- gsl_matrix_add (a,b); // aij = aij + bij , 8 i, j
- gsl_matrix_sub (a,b); // aij = aij − bij , 8 i, j
- gsl_matrix_mul elements (a,b); // aij = aij · bij , 8 i, j
- gsl_matrix_div elements (a,b);// aij = aij/bij , 8 i, j
- gsl vector add (a,b); // ai = ai + bi, 8 i
- gsl vector sub (a,b); // ai = ai − bij , 8 i
- gsl vector mul (a,b); // ai = ai · bi, 8 i
- gsl vector div (a,b); // ai = ai/bi, 8 i
- gsl_matrix_scale (a,x); // aij = aij · x, 8 i, j, x 2 R
- gsl_matrix_add_constant (a,x); // aij = aij + x, 8 i, j, x 2 R
- gsl_vector_scale_(a,x); // ai = ai · x, 8 i, x 2 R
- gsl_vector_add_constant (a,x); // ai = ai + x, 8 i, x 2 R

## Distributions

- http://www.socr.ucla.edu/docs/edu/ucla/stat/SOCR/core/Distribution.html

## MGFs and PGFs

- The **moment-generating function** generates the moments of the probability distribution.
  - http://en.wikipedia.org/wiki/Moment_generating_function

$$M_X(t) = E\left(e^{tX}\right), \quad t \in \mathbb{R}.$$

- Provided the moment-generating function exists in an interval around $t=0$, the $n$th moment is given by

$$E\left(X^n\right) = M_X^{(n)}(0) = \left.\frac{\mathrm{d}^n}{\mathrm{d}t^n}\right|_{t=0} M_X(t).$$

- For **Independent** (not necessarily IID) variables, $X_i$, $S_n = \sum_{i=1}^{n} a_i X_i$, the MGF ($S_n$) is:

$$M_{S_n}(t) = M_{X_1}(a_1 t) M_{X_2}(a_2 t) \cdots M_{X_n}(a_n t).$$

## MGFs and PGFs

- **Probability-generating function** of a discrete random variable is a power series representation (the generating function) of the probability mass function of the random variable. Probability-generating functions are often employed for their succinct description of the sequence of probabilities Pr(*X* = *i*), and to make available the well-developed theory of power series with non-negative coefficients.
- If *N* is a discrete random variable taking values on some subset of the non-negative integers, {*0,1, …*}, then the *probability-generating function* of *N* is defined as:

$$G(x) = \mathrm{E}(x^N) = \sum_{n=0}^{\infty} f(n)x^n,$$

  - where *f* is the probability mass function of *N*.
- The probability mass function of *X* is recovered by differentiating *G*

$$f(k) = \Pr(X = k) = \frac{G^{(k)}(0)}{k!}.$$

- http://en.wikipedia.org/wiki/Probability-generating_function

## MGFs and PGFs

- **MGF and PGF are important because:**
  - If $X_1, X_2, \ldots, X_n$ is a sequence of independent (and not necessarily IID) variables and $S_n = \sum_{i=1}^{n} a_i X_i$, we may compute the distribution of $S_n$
  - Completely describing the (nice) distributions?
    - There are distributions for which knowing all moments does not determine the distribution (e.g., Log-Normal).
    - For distributions with finite range, the moments always uniquely determine the distributions.
    - For infinite-range distributions, the moments uniquely determine the distribution if these series Diverge:

$$\sum_{j=0}^{\infty} \frac{1}{(\mu_{2j})^{1/2j}}; -\infty < x < \infty \text{ and } \sum_{j=0}^{\infty} \frac{1}{(\mu_j)^{1/2j}}; 0 < x < \infty$$

## Simulation

- **How do we sample (simulate) data from an arbitrary distribution?**
- If X~f and F(x) is invertible, then $Y=F^{-1}(U)$ ~ f, where U~Uniform(0,1)
- Otherwise

```
/* This method computes a default simulation of a value from the distribution */
public double simulate() {  return getQuantile(Math.random());  }
public double getQuantile(double p) {
    double sum = 0, x, w;
    int n = domain.getSize(), i = 0;
        x = domain.getValue(i);    sum = getDensity(x) * w;
        while ((sum < p) & (i < n)) {  i++;  x = domain.getValue(i);
            sum = sum + getDensity(x) * w; }
        return x;
}
/* This method return the midpoint of the interval corresponding to a given index */
public double getValue (int i) {  return lowerValue + i * width;  }
```

## Monte Carlo Integration

- MonteCarlo_Integration.pdf (slides 12-14)
- Rejection Sampling