Biostatistics 615/815
Implementing Fisher's Exact Test

Hyun Min Kang

Januray 13th, 2011

Recap - helloWorld - Have all tried?

Writing helloWorld.cpp

```
#include <iostream> // import input/output handling library
int main(int argc, char** argv) {
  std::cout << "Hello, World" << std::endl;
  return 0; // program exits normally
}
```

Compiling helloWorld.cpp

Install Cygwin (Windows), Xcode (MacOS), or nothing (Linux).

```
user@host:~/$ g++ -o helloWorld helloWorld.cpp
```

Running helloWorld

```
user@host:~/$ ./helloWorld
Hello, World
```


Recap - precisionExample.cpp

precisionExample.cpp

```
#include <iostream>
int main(int argc, char** argv) {
  float smallFloat = 1e-8; // a small value
  float largeFloat = 1.;   // difference in 8 (>7.2) decimal figures.
  std::cout << "smallFloat = " << smallFloat << std::endl;
  smallFloat = smallFloat + largeFloat;
  smallFloat = smallFloat - largeFloat;
  std::cout << "smallFloat = " << smallFloat << std::endl;
  // similar thing happens for doubles (e.g. 1e-20 vs 1).
  return 0;
}
```

Running precisionExample

```
user@host:~/$ ./precisionExample
smallFloat = 1e-08
smallFloat = 0
```


Recap - Handling command line arguments

echo.cpp - echoes command line arguments to the standard output

```
#include <iostream>
int main(int argc, char** argv) {
  for(int i=1; i < argc; ++i) { // i=1 : 2nd argument (skip program name)
    if ( i > 1 ) // print blank if there is an item already printed
      std::cout << " ";
    std::cout << argv[i];  // print each command line argument
  }
  std::cout << std::endl; // print end-of-line at the end
}
```

Compiling and running echo.cpp

```
user@host:~/$ g++ -o echo echo.cpp
user@host:~/$ ./echo you need to try this out by yourself!
you need to try this out by yourself!
```

Annoucements

- 815 Projects will be annouced in the next lecture
- Midterm date will be scheduled on March 10 - any objections?
- Final exame date will be April 21st, 10:30am-12:30pm, just like the official schedule.
- Homework #1 will be annouced today

Let's implement Fisher's exact Test

Input - A 2×2 table

	Placebo	Treatment	Total
Diseased	a	b	a+b
Cured	c	d	c+d
Total	a+c	b+d	n

Desired Program Interface and Results

```
user@host:~/$ ./fishersExactTest 1 2 3 0
Two-sided p-value is 0.4
user@host:~/$ ./fishersExactTest 2 7 8 2
Two-sided p-value is 0.0230141
user@host:~/$ ./fishersExactTest 20 70 80 20
Two-sided p-value is 5.90393e-16
```


Simple Math under Fisher's Exact Test

Possible 2×2 tables

	Placebo	Treatment	Total
Diseased	x	a+b-x	a+b
Cured	a+c-x	d-a+x	c+d
Total	a+c	b+d	n

Hypergeometric distribution

Given $a+b, c+d, a+c, b+d$ and $n = a+b+c+d$,

$$\Pr(x) = \frac{(a+b)!(c+d)!(a+c)!(b+d)!}{x!(a+b-x)!(a+c-x)!(d-a+x)!n!}$$

Fishers's Exact Test (2-sided)

$p_{FET}(a,b,c,d) = \sum_x \Pr(x) I[\Pr(x) \leq \Pr(a)]$

intFishersExactTest.cpp - main() function

```
#include <iostream>
double hypergeometricProb(int a, int b, int c, int d); // defined later
int main(int argc, char** argv) {
  // read input arguments
  int a = atoi(argv[1]), b = atoi(argv[2]), c = atoi(argv[3]), d = atoi(argv[4]);
  int n = a + b + c + d;
  // find cutoff probability
  double pCutoff = hypergeometricProb(a,b,c,d);
  double pValue = 0;
  // sum over probability smaller than the cutoff
  for(int x=0; x <= n; ++x) { // among all possible x
    if ( a+b-x >= 0 && a+c-x >= 0 && d-a+x >=0 ) { // consider valid x
      double p = hypergeometricProb(x,a+b-x,a+c-x,d-a+x);
      if ( p <= pCutoff ) pValue += p;
    }
  }
  std::cout << "Two-sided p-value is " << pValue << std::endl;
  return 0;
}
```

intFishersExactTest.cpp

hypergeometricProb() function

```
int fac(int n) { // calculates factorial
  int ret;
  for(ret=1; n > 0; --n) { ret *= n; }
  return ret;
}
double hypergeometricProb(int a, int b, int c, int d) {
  int num = fac(a+b) * fac(c+d) * fac(a+c) * fac(b+d);
  int den = fac(a) * fac(b) * fac(c) * fac(d) * fac(a+b+c+d);
  return (double)num/(double)den;
}
```

Running Examples

```
user@host:~/$ ./intFishersExactTest 1 2 3 0
Two-sided p-value is 0.4      // correct
user@host:~/$ ./intFishersExactTest 2 7 8 2
Two-sided p-value is 4.41018  // INCORRECT
```


Considering Precision Carefully

factorial.cpp

```
int fac(int n) { // calculates factorial
  int ret;
  for(ret=1; n > 0; --n) { ret *= n; }
  return ret;
}
int main(int argc, char** argv) {
  int n = atoi(argv[1]);
  std::cout << n << "! = " << fac(n) << std::endl;
}
```

Running Examples

```
user@host:~/$ ./factorial 10
10! = 362880     // correct
user@host:~/$ ./factorial 12
12! = 479001600  // correct
user@host:~/$ ./factorial 13
13! = 1932053504 // INCORRECT : 13! > INT_MAX == 2147483647
```


doubleFishersExactTest.cpp

new hypergeometricProb() function

```
double fac(int n) { // main() function remains the same
  double ret; // use double instead of int
  for(ret=1.; n > 0; --n) { ret *= n; }
  return ret;
}
double hypergeometricProb(int a, int b, int c, int d) {
  double num = fac(a+b) * fac(c+d) * fac(a+c) * fac(b+d);
  double den = fac(a) * fac(b) * fac(c) * fac(d) * fac(a+b+c+d);
  return num/den; // use double to calculate factorials
}
```

Running Examples

```
user@host:~/$ ./doubleFishersExactTest 2 7 8 2
Two-sided p-value is 0.023041
user@host:~/$ ./doubleFishersExactTest 20 70 80 20
Two-sided p-value is 0 (fac(190) > 1e308 - beyond double precision)
```


How to perform Fisher's exact test with large values

Problem - Limited Precision

- `int` handles only up to fac(12)
- `double` handles only up to fac(170)

Solution - Calculate in logarithmic scale

$$\begin{aligned}
\log \Pr(x) &= \log(a+b)! + \log(c+d)! + \log(a+c)! + \log(b+d)! - \log x! \\
&\quad - \log(a+b-x)! - \log(a+c-x)! - \log(d-a+x)! - \log n! \\
\log(p_{FET}) &= \log\left[\sum_x \Pr(x) I(\Pr(x) \le \Pr(a))\right] \\
&= \log \Pr(a) + \log\left[\sum_x \exp(\log \Pr(x) - \log \Pr(a)) I(\log \Pr(x) \le \log \Pr(a))\right]
\end{aligned}$$

logFishersExactTest.cpp - main() function

```
#include <iostream>
#include <cmath>    // for calculating log() and exp()
double logHypergeometricProb(int a, int b, int c, int d); // defined later
int main(int argc, char** argv) {
  int a = atoi(argv[1]), b = atoi(argv[2]), c = atoi(argv[3]), d = atoi(argv[4]);
  int n = a + b + c + d;
  double logpCutoff = logHypergeometricProb(a,b,c,d);
  double pFraction = 0;
  for(int x=0; x <= n; ++x) { // among all possible x
    if ( a+b-x >= 0 && a+c-x >= 0 && d-a+x >=0 ) { // consider valid x
      double l = logHypergeometricProb(x,a+b-x,a+c-x,d-a+x);
      if ( l <= logpCutoff ) pFraction += exp(l - logpCutoff);
    }
  }
  double logpValue = logpCutoff + log(pFraction);
  std::cout << "Two-sided log10-p-value is " << logpValue/log(10.) << std::endl;
  std::cout << "Two-sided p-value is " << exp(logpValue) << std::endl;
  return 0;
}
```


Filling the rest

logHypergeometricProb()

```
double logFac(int n) {
  double ret;
  for(ret=0.; n > 0; --n) { ret += log((double)n); }
  return ret;
}
double logHypergeometricProb(int a, int b, int c, int d) {
  return logFac(a+b) + logFac(c+d) + logFac(a+c) + logFac(b+d) - logFac(a)
       - logFac(b) - logFac(c) - logFac(d) - logFac(a+b+c+d);
}
```

Running Examples

```
user@host:~/$ ./logFishersExactTest 2 7 8 2
Two-sided log10-p-value is -1.63801, p-value is 0.0230141
user@host:~/$ ./logFishersExactTest 20 70 80 20
Two-sided log10-p-value is -15.2289, p-value is 5.90393e-16
user@host:~/$ ./logFishersExactTest 200 700 800 200
Two-sided log10-p-value is -147.563, p-value is 2.73559e-148
```


Even faster

Computational speed for large dataset

```
time ./logFishersExactTest 2000 7000 8000 2000
Two-sided log10-p-value is -1466.13, p-value is 0
real 0m42.614s

time ./fastFishersExactTest 2000 7000 8000 2000
Two-sided log10-p-value is -1466.13, p-value is 0
real 0m0.007s
```

How to make it faster?

- Most time consuming part is the repetitive computation of factorial
  - # of `logHypergeometricProbs` calls is $\leq a+b+c+d=n$
  - # of `logFac` call $\leq 9n$
  - # of `log` calls $\leq 9n^2$ - could be billions in the example above
- Key Idea is to store `logFac` values to avoid repetitive computation

fastFishersExactTest.cpp - main() function

```
#include <iostream> // everything remains the same except for lines marked with ***
#include <cmath>
double logHypergeometricProb(double* logFacs, int a, int b, int c, int d); // ***
void initLogFacs(double* logFacs, int n); // *** New function ***
int main(int argc, char** argv) {
  int a = atoi(argv[1]), b = atoi(argv[2]), c = atoi(argv[3]), d = atoi(argv[4]);
  int n = a + b + c + d;
  double* logFacs = new double[n+1]; // *** dynamically allocate memory logFacs[0..n] ***
  initLogFacs(logFacs, n);           // *** initialize logFacs array ***
  double logpCutoff = logHypergeometricProb(logFacs,a,b,c,d); // *** logFacs added
  double pFraction = 0;
  for(int x=0; x <= n; ++x) {
    if ( a+b-x >= 0 && a+c-x >= 0 && d-a+x >=0 ) {
      double l = logHypergeometricProb(x,a+b-x,a+c-x,d-a+x);
      if ( l <= logpCutoff ) pFraction += exp(l - logpCutoff);
    }
  }
  double logpValue = logpCutoff + log(pFraction);
  std::cout << "Two-sided log10-p-value is " << logpValue/log(10.) << std::endl;
  std::cout << "Two-sided p-value is " << exp(logpValue) << std::endl;
  return 0;
}
```

fastFishersExactTest.cpp - other functions

function initLogFacs()

```
void initLogFacs(double* logFacs, int n) {
  logFacs[0] = 0;
  for(int i=1; i < n+1; ++i) {
    logFacs[i] = logFacs[i-1] + log((double)i); // only n times of log() calls
  }
}
```

function logHyperGeometricProb()

```
double logHypergeometricProb(double* logFacs, int a, int b, int c, int d) {
  return logFacs[a+b] + logFacs[c+d] + logFacs[a+c] + logFacs[b+d]
    - logFacs[a] - logFacs[b] - logFacs[c] - logFacs[d] - logFacs[a+b+c+d];
}
```


Summary so far

- Algorithms are computational steps
- towerOfHanoi utilizing recursions
- Data types and floating-point precisions
- Operators, if, for, and while statements
- Arrays and strings
- Pointers and References
- Functions
- Fisher's Exact Test
  - ✓ intFishersExactTest - works only tiny datasets
  - ✓ doubleFishersExactTest - handles small datasets
  - ✓ logFishersExactTest - handles hundreds of observations
  - ✓ fastFishersExactTest - equivalent to logFisherExactTest but faster
- At Home : Reading material for novice C++ users : *http://www.cplusplus.com/doc/tutorial/* - Until "Control Structures"

Homework #1

Problem #1

Implement a program fullFastFishersExactTest which

- Prints out an error message and return -1 when number of input arguments are not 4 (excluding the program name itself)
- and outputs the two-sided p-value, and one-sided p-values
  - $p_{2sided}(a,b,c,d) = \sum_x \Pr(x) I[\Pr(x) \leq \Pr(a)]$
  - $p_{greater}(a,b,c,d) = \sum_{x \geq a} \Pr(x)$
  - $p_{less}(a,b,c,d) = \sum_{x \leq a} \Pr(x)$
- based on fastFishersExactTest.cpp

Homework #1

Problem #1 - Example Program Interface

```
user@host:~/$ ./fullFastFishersExactTest 2 7 8 2
Two-sided log10(p) = -1.63801, p-value = 0.0230141
One-sided (less) log10(p) = -1.73232, p-value = 0.0185217
One-sided (greater) log10(p) = -0.000428027, p-value = 0.999015

user@host:~/$ ./fullFastFishersExactTest 20 70 80 20
Two-sided log10(p) = -15.2289, p-value = 5.90393e-16
One-sided (less) log10(p) = -15.3764, p-value = 4.20368e-16
One-sided (greater) log10(p) = 8.0232e-14, p-value = 1

user@host:~/$ ./fullFastFishersExactTest
Usage: fullFastFishersExactTest [#row1col1] [#row1col2] [#row2col1] [#row2col2]
```

More Homework Problems

- Several example codes asking for the outputs (with explanations)
- Absolutely no discussion with your classmates.
- But you can consult to your computer (to compiler, not google!)
  - If you are certain what the answer is, you can just write your answer
  - If you are not certain, you can write the code, and see what happens
- Turn in your hard copy of your answers before Thursday (19th) lecture
- And email the source code (.cpp only) for Homework #1 to hmkang@umich.edu with title "BIOSTAT 615/815 Homework #1 - [your name]"

Next Lecture

More on C++ Programming

- Standard Template Library
- User-defined data types

Divide and Conquer Algorithms

- Binary Search
- Insertion Sort (skipped in lecture 1)
- Merge Sort