Server-side Statistics Scripting in PHP

Jan de Leeuw
UCLA Statistics

June 22, 1997

1 Introduction

On the UCLA Statistics WWW server there are a large number of demos and calculators that can be used in statistics teaching and research. Some of these demos require substantial amounts of computation, others mainly use graphics. These calculators and demos are implemented in various different ways, reflecting developments in WWW based computing.

As usual, one of the main choices is between doing the work on the client-side (i.e. in the browser) or on the server-side (i.e. on our WWW server). Obviously, client-side computation puts fewer demands on the server. On the other hand, it requires that the client downloads Java applets, or installs plug-ins and/or helpers. If JavaScript is used, client-side computations will generally be slow. We also have to assume that the client is installed properly, and has the required capabilities. Requiring too much on the client-side has caused browsing machines such as Netscape Communicator to grow beyond all reasonable bounds, both in size and RAM requirements. Moreover requiring Java and JavaScript rules out such excellent browsers as Lynx or Emacs W3.

For server-side computing, we can configure the server and its resources ourselves, and we need not worry about browser capabilities and configuration. Nothing needs to be downloaded, except the usual HTML pages and graphics. In the same way as on the client side, there is a scripting solution, where code is interpreted, or a object-code solution using compiled code. For the server-side scripting, we use embedded languages, such as PHP/FI. The scripts in the HTML pages are interpreted by a CGI program, and the output of the CGI program is send to the clients. Of course the CGI program is compiled, but the statistics procedures will usually be interpreted, because PHP/FI does not have the appropriate functions in its scripting language. This will tend to be slow, because embedded languages do not deal efficiently with loops and similar constructs.

Thus a first step towards greater efficiency is to compile the necessary primitives into the PHP/FI executable. This is easy to do, because the API is quite simple. In the extensions below, we have added the complete `ranlib` and `dcdflib` to PHP, plus some additional useful functions. The source code for

these extensions, plus Solaris binaries for `libranlib.a` and `libdcdf.a` can be obtained from our server.

Interpreting a PHP script, even with our new primitives, still requires starting up a CGI process for each page that is read. Again, this can be improved upon. We could use FastCGI to keep the CGI process around on a permanent basis. Instead, we have chosen a more direct method. PHP can be compiled as an Apache module, i.e. it can be compiled into the Apache HTTPD server binary. This means that PHP scripts are interpreted by the WWW server, which is always around, and which will fork additional children if necessary. No CGI processes need to be started. The PHP install process creates a `libphp.a` and `mod_php.c` in the Apache source directories, which can be used to build an enhanced server. This has the additional advantage of security, because all security features of the server can be used, and none of the pitfalls of using CGI or Java apply.

Using PHP, in combination with the WWW server, also has some disadvantages. Although we can make simple static plots, using the gd library, we cannot use any dynamics, and interaction between the user and the page is somewhat limited. Java, or scripts using a client-side Xlisp-Stat as a helper, are more flexible in this respect. As a consequence, the UCLA Statistics pages still use a combined approach, with server-side PHP and CGI and client-side Xlisp-Stat and Java/JavaScript. Sometime this year, server-side Java scripting will become available, and then it seems advisable to switch as much of the code as possible to the server-side.

2 Scripting in PHP

We shall not give an extensive introduction to PHP/FI scripting here. For this we refer to the PHP/FI manual, and to the examples below. Basically, the scripting language is a simple subset of C, with additional support built-in for generation of GIF pictures using the gd library, and support for various database engines such as mSQL.

One useful thing to know about PHP/FI scripting is that in PHP/FI variables are *overloaded*. Thus each variable has three values, the variable as a long integer, the variable as a double, and the variable as a string. Thus running the following code fragment

```
<?
 $a = 0.999;
 $b = 1;
 $c = "melon";
 Echo intval($a); Echo "<BR>";
 Echo doubleval($a); Echo "<BR>";
 Echo strval($a); Echo "<BR>";
 Echo intval($b); Echo "<BR>";
 Echo doubleval($b); Echo "<BR>";
 Echo strval($b); Echo "<BR>";
```

```
 Echo intval($c); Echo "<BR>";
 Echo doubleval($c); Echo "<BR>";
 Echo strval($c); Echo "<BR>";
>
```

produces

```
0
0.9990000000
0.999
1
1.0000000000
1
0
0.0000000000
melon
```

This overloading makes it more or less unnecessary to specify the types of arguments that functions in PHP/FI require. In the description of the function we indicate the types we had in mind, which correspond with what the C routines expect. Generally, both `cdflib` and `ranlib` work with doubles, even degrees-of-freedom parameters and number of trials or successes can be doubles.

3 dcdflib

`ccdflib` is an excellent library for computation of cumulative distribution functions and their inverses. It is written by Brown, Lovato, and Russell [2]. We use the double precision version, written in ANSI-C. Since all functions use the double precision value of the arguments, and return a variable whose double precision value we are interested in, there is no need to indicate types.

3.1 FCdf

If $\mathcal{F}^{F}_{dfn,dfd}(f) = p$ then

`FCdf (f, dfn, dfd, 1)`	$\Longrightarrow$	p
`FCdf (p, dfn, dfd, 2)`	$\Longrightarrow$	f
`FCdf (p, f, dfd, 3)`	$\Longrightarrow$	dfn
`FCdf (p, f, dfn, 4)`	$\Longrightarrow$	dfd

3.2 BetaCdf

If $\mathcal{F}^{beta}_{a,b}(x) = p$ then

`BetaCdf (x, a, b, 1)`	$\Longrightarrow$	p
`BetaCdf (p, a, b, 2)`	$\Longrightarrow$	x
`BetaCdf (p, x, b, 3)`	$\Longrightarrow$	a
`BetaCdf (p, x, a, 4)`	$\Longrightarrow$	b

3.3 Chi2Cdf

If $\mathcal{F}_{df}^{chi2}(x) = p$ then

```
Chi2Cdf (x, df, 1)  ⟹  p
Chi2Cdf (p, df, 2)  ⟹  x
Chi2Cdf (p, x, 3)   ⟹  df
```

3.4 GammaCdf

If $\mathcal{F}_{shape,scale}^{gamma}(x) = p$ then

```
GammaCdf (x, scale, shape, 1)  ⟹  p
GammaCdf (p, scale, shape, 2)  ⟹  x
GammaCdf (p, x, shape, 3)      ⟹  scale
GammaCdf (p, x, scale, 4)      ⟹  shape
```

3.5 NormalCdf

If $\mathcal{F}_{\mu,\sigma}^{normal}(x) = p$ then

```
NormalCdf (x, mean, sd, 1)  ⟹  p
NormalCdf (p, mean, sd, 2)  ⟹  x
NormalCdf (p, x, sd, 3)     ⟹  mean
NormalCdf (p, x, mean, 4)   ⟹  sd
```

3.6 StudentCdf

If $\mathcal{F}_{df}^{student}(t) = p$ then

```
StudentCdf (t, df, 1)  ⟹  p
StudentCdf (p, df, 2)  ⟹  t
StudentCdf (p, t, 3)   ⟹  df
```

3.7 PoissonCdf

If $\mathcal{F}_{xlam}^{poisson}(x) = p$ then

```
PoissonCdf (x, xlam, 1)  ⟹  p
PoissonCdf (p, xlam, 2)  ⟹  x
PoissonCdf (p, x, 3)     ⟹  xlam
```

3.8 BinomialCdf

If $\mathcal{F}_{pr,xn}^{binom}(sn) = p$ then

```
BinomialCdf (sn, xn, pr, 1)  ⟹  p
BinomialCdf (p, xn, pr, 2)   ⟹  sn
BinomialCdf (p, sn, pr, 3)   ⟹  xn
BinomialCdf (p, sn, xn, 4)   ⟹  pr
```

3.9 NegBinomialCdf

If $\mathcal{F}^{negbinom}_{pr,xn}(sn) = p$ then

```
NegBinomialCdf (sn, xn, pr, 1)  ⟹  p
NegBinomialCdf (p, xn, pr, 2)   ⟹  sn
NegBinomialCdf (p, sn, pr, 3)   ⟹  xn
NegBinomialCdf (p, sn, xn, 4)   ⟹  pr
```

3.10 NonCentralFCdf

If $\mathcal{F}^{F}_{dfn,dfd,pnonc}(f) = p$ then

```
NonCentralFCdf (f, dfn, dfd, pnonc, 1)  ⟹  p
NonCentralFCdf (p, dfn, dfd, pnonc, 2)  ⟹  f
NonCentralFCdf (p, f, dfd, pnonc, 3)    ⟹  dfn
NonCentralFCdf (p, f, dfn, pnonc, 4)    ⟹  dfd
NonCentralFCdf (p, f, dfn, dfd, 5)      ⟹  pnonc
```

3.11 NonCentralChi2Cdf

If $\mathcal{F}^{chi2}_{df,pnonc}(x) = p$ then

```
NonCentralChi2Cdf (x, df, pnonc, 1)  ⟹  p
NonCentralChi2Cdf (p, df, pnonc, 2)  ⟹  x
NonCentralChi2Cdf (p, x, pnonc, 3)   ⟹  df
NonCentralChi2Cdf (p, x, df, 4)      ⟹  pnonc
```

4 ranlib

`ranlib` was also written by Brown and Lovato [1].

Observe that PHP/FI already has some random number support through the usual `Rand()`, `Srand((int) x)`, and `getRandMax()` functions. Here `Rand` returns a random integer between 0 and `RANDMAX`, `Srand` seeds the random number generator, and `getRandMax` returns `RANDMAX`. We add more sophisticated generators, more control, and generators for the same families of probability distributions in `cdflib`.

4.1 RanF

```
Ranf()  ⟹  x
```

`Ranf` does not take any arguments, and returns a random floating point number in the open interval $(0, 1)$.

4.2 PhrTsd

```
PhrTsd(phrase)  ⟹  seeds
```

`PhrTsd` takes a phrase an argument and returns a string of two concatenated seeds, separated by a space.

4.3 GetSeed

```
GetSeed()  ⟹  seeds
```

`GetSeed` takes no argument and returns a string of the two concatenated current seeds, separated by a space.

4.4 SetAll

```
SetAll(seed1, seed2)  ⟹  .
```

`SetAll` initializes all generators using the seeds in the argument.

4.5 Shuffle

```
Shuffle((array) a)
```

`Shuffle` randomly permutes an array (it is a PHP/FI interface to the `genprm` function in `ranlib`).

4.6 GenF

```
GenF(dfn, dfd)  ⟹  f
```

`GenF` generates a random deviate from an F distribution with `dfn` and `dfd` degrees of freedom.

4.7 GenGam

```
GenGam(a, r)  ⟹  x
```

`GenGam` generates a random deviate from an gamma distribution with location parameters `a` and shape parameter `r`.

4.8 GenNCh

```
GenNCh(df, xnonc)  ⟹  x
```

`GenNch` generates a random deviate from a noncentral chi-square distribution with `df` degrees of freedom and noncentrality parameter `xnonc`.

4.9 GenNF

```
GenNF(dfn, dfd, xnonc)  ⟹  x
```

`GenNF` generates a random deviate from a noncentral F distribution with `dfn` and `dfd` degrees of freedom and noncentrality parameter `xnonc`.

4.10 GenNor

```
GenNor(av, sd)  ⟹  x
```

`GenNor` generates a random deviate from a normal distribution with mean `av` and standard deviation `sd`.

4.11 GenUnf

```
GenUnf(low, high)  ⟹  x
```

`GenUnf` generates a random deviate from a uniform distribution between `low` (exclusive) and `high` (exclusive).

4.12 GenBet

```
GenBet(aa, bb)  ⟹  x
```

`GenBet` generates a random deviate from a beta distribution with parameters `aa` and `bb`. (exclusive).

4.13 GenChi

```
GenChi(df)  ⟹  x
```

`GenCh` generates a random deviate from a chi-square distribution with `df` degrees of freedom.

4.14 GenExp

```
GenExp(av)  ⟹  x
```

`GenExp` generates a random deviate from an exponential distribution with mean `av`.

4.15 IgnLgi

```
IgnLgi()  ⟹  x
```

`GenExp` generates a random integer, from the uniform distribution over $1, \ldots, 2147483562$.

4.16 IgnUin

```
IgnUin(low, high)  ⟹  x
```

`IgnUin` generates a random integer, from the uniform distribution over `low...high`.

4.17 IgnBin

```
IgnBin(n, pp)  ⟹  s
```

`IgnBin` generates a random deviate from a binomial distribution with `n` trials and probability of success `pp`.

4.18 IgnNbn

```
IgnNbn(n, pp)  ⟹  s
```

`IgnBin` generates a random deviate from a negative binomial distribution with `n` trials and probability of success `pp`.

4.19 IgnPoi

```
IgnPoi(mu)  ⟹  x
```

`IgnPoi` generates a random deviate from a Poisson distribution with mean `ave`.

5 density

This section contains functions to compute the most important probability density functions and probability mass functions.

5.1 NormalDens

```
NormalDens(x, ave, stdv)  ⟹  y
```

`NormalDens` computes the ordinate of the normal density with mean *ave* and standard deviation *stdv* at x.

5.2 Chi2Dens

```
Chi2Dens(x, dfr)  ⟹  y
```

`Chi2Dens` computes the ordinate of the chi-square density with dfr degrees of freedom at x.

5.3 TDens

```
TDens(x, dfr)  ⟹  y
```

`TDens` computes the ordinate of the student t density with dfr degrees of freedom at x.

5.4 FDens

```
FDens(x, shape, scale)  ⟹  y
```

`FDens` computes the ordinate of the F density with degrees of freedom $dfr1$ and $dfr2$ at x.

5.5 BetaDens

```
BetaDens(x, a, b)  ⟹  y
```

`BetaDens` computes the ordinate of the beta density with parameters a and b at x.

5.6 GammaDens

```
GammaDens(x, shape, scale)  ⟹  y
```

`GammaDens` computes the ordinate of the gamma density with parameters *shape* and *scale* at x.

5.7 BinomialPmf

```
BinomialPmf(x, N, pi)  ⟹  p
```

`BinomialPmf` computes the probability mass of the binomial with parameters N and pi at x.

5.8 PoissonPmf

```
PoissonPmf(x, lambda)  ⟹  p
```

`PoissonPmf` computes the probability mass of the Poisson with parameter *lambda* at x.

5.9 NegBinomialPmf

```
NegBinomialPmf(x, N, pi)  ⟹  p
```

`NegBinomialPmf` computes the probability mass of the negative binomial with parameters N and pi at x.

5.10 HypergeometricPmf

```
HypergeometricPmf(n, m, N, M)  ⟹  p
```

`HypergeometricPmf` computes the probability mass of the hypergeometric with parameters N and M at n and m.

6 statistics

This section contains some auxilary functions useful in statistical computing.

6.1 PowerSum

```
PowerSum(a, s)  ⟹  x
```

`PowerSum` takes an array a and a number s and computes the sum of the $s-th$ powers of the elements of a.

6.2 InnerProduct

```
InnerProduct(a, b)  ⟹  x
```

`PowerSum` computes the inner product of arrays a and b.

6.3 Independent_t

```
Independent\_t(a, b)  ⟹  x
```

`Independent_t` computes the two-sample t-statistic for arrays a and b.

6.4 Paired_t

```
Paired\_t(a, b)  ⟹  x
```

`Paired_t` computes the paired t-statistic for arrays a and b.

6.5 Correlation

```
Correlation(a, b)  ⟹  x
```

`Correlation` computes the correlation of arrays a and b.

6.6 Factorial

```
Factorial(k)  ⟹  n
```

`Factorial` computes the factorial of k.

6.7 BinomialCoefficient

```
BinomialCoefficient(k, n)  ⟹  b
```

`BinomialCoefficient` computes the binomial coefficient $C(n, k)$.

7 statplot

Additional plotting routines useful for statistics.

7.1 ScatterPlot

```
ScatterPlot(im, x, y, sl, sr, fcolor, bcolor, lcolor, connect)
```

`ScatterPlot` takes an image im created by the PHP/IP interface to gd, two arrays x and y of coordinates, two spikes sl and sl a foreground, a background color, and a flood color (0 for black, 1 for white, 2 for red, 3 for green, 4 for blue), and a parameter indicating wether or not we connect successive points (0 for no, 1 for yes, 2 for spikes). The area between sl and sb is flooded using the flood color.

7.2 Histogram

```
Histogram()
```

`Histogram` blabla

8 Examples

8.1 Normal Calculator

This is a relatively simple example of a "sticky form" implementing the normal cdf calculator. The PHP source is in the code directory.

8.2 t-test

The t-test calculator uses `Independent_t`, `Paired_t`, `StudentCdf`, and `Shuffle`. Again the PHP source is in the code directory.

References

[1] Barry W. Brown and James Lovato: *RANLIB.C. Library of C Routines for Random Number Generation*. Department of Biomathematics, M.D. Anderson Cancer Center, University of Texas, Houston.

[2] Barry W. Brown, James Lovato and Kathy Russell: *DCDFLIB. Library of C Routines for Cumulative Distribution Functions, Inverses, and Other Parameters*. Department of Biomathematics, M.D. Anderson Cancer Center, University of Texas, Houston.