

Mathematical Programming with PHP

PHP Quebec conference

Paul Meagher <paul@datavore.com>

April 2005

Intelligent websites require number crunching yet PHP lacks many mathematical programming tools that would help make advanced number crunching possible. In this presentation we will ask why PHP lacks math programming tools and how we might go about adding math processing capabilities to PHP. My current preference is to implement math programming tools as packages and we will spend most of our time discussing two strategic PHP-based math packages: the Probability Distributions Library Package (aka PDL Package) and the JAMA Package for linear algebra. Finally I will discuss whether we should compete with or bind to other mature open-source math libraries. My preference is to compete.

Introduction

This article is an expanded version of a talk "Mathematical Programming with PHP" that I presented at the PHP Quebec conference on Mar 31-Apr 1, 2005. I decided to convert my "slides" into an article because my notes were more for myself than for public consumption and because I figured an expanded version of my notes would make for a nice article-length piece.

Note: This article is currently under construction until I finish blogging about each of my conference slides, have added these blogs to the article, and finished polishing the final article product.

Overcoming barriers to entry

The concept of "barriers to entry" was inspired by [Micheal Porter](#) who has written many important books on competitive strategy.

Many companies pay people to develop math-based applications in other languages beside PHP. We need to ask how we might begin to grow a market for the development of PHP-based math applications. Below I analyze some barriers to entry in this space and formulate a strategy to overcome them.

Intellectual barriers to entry

1. Our math education system sucks so the PHP developer base is small.
2. Preconceptions that you can't do advanced math with PHP.
3. Math is hard, lets go shopping. ~ Barbie

Economic barriers to entry

1. Why reinvent the wheel.
2. No market for these services so less spinoff code.
3. Insufficient academic awareness to sponsor formative projects.

A Strategy

Develop packages that give you practical usefulness and broad coverage first: [JAMA Package](#), [PDL Package](#).

Opensource a number of significant math libraries *and projects* (i.e., web-based multiple regression) to create an awareness that mathematically sophisticated websites and web applications can be built using PHP.

Emphasize web-based usage examples of PHP + Math code to appeal to it's largest user base among web developers and to even the balance against competing solutions in this space (versus the command line space and desktop GUI space where PHP is not as strong).

Bind or compete?

When developing math applications, should PHP developers attempt to bind to a mature and popular opensource math server like R or should we compete with R?

Binding to the R math server

Why do you want to reinvent the wheel?

```
# downloads Boston dataset from STATLIB
use LWP::Simple;
getstore(
    "http://lib.stat.cmu.edu/datasets/boston",
    "boston.raw" );

# corrects for record spanning two lines
open( IN, "boston.raw" );
open( OUT, ">boston.asc" );
do { $line = <IN> } until $. == 22
    or eof; # Skips the first 22 lines of header
while ( $line = <IN> ) {
    chomp $line;
    $line .= <IN>; # joins two lines
    print OUT $line;
}
close(OUT);

# sends data to R for regression analysis
open( RPROG,
    "| c:/rw1081/bin/Rterm.exe --no-restore --no-save"
);
select(RPROG);
print <<'CODE';
bost<-read.table("boston.asc",header=F)
names(bost)<- c( "CRIM", "ZN", "INDUS",
               "CHAS", "NOX", "RM",
               "AGE", "DIS", "RAD",
               "TAX", "PTRAT", "B",
               "LSTAT", "MEDV")

attach(bost)
boston.fit <- lm(log(MEDV) ~ CRIM + ZN + INDUS +
    CHAS + I(NOX^2) + I(RM^2) + AGE + log(DIS) +
    log(RAD) + TAX + PTRAT + B + log(LSTAT))
sum <- summary(boston.fit)$coe[,1:2]
write.table(sum,"boston.out",quote = FALSE)
q()
CODE
close(RPROG);
```

A similar approach is easy to implement in PHP. But we have to ask:

Can we do this type of computation before a web page is generated, and without noticeable delay, so as to determine the content of that page for a particular user?

No, implementing all the math logic in PHP would make more sense than using this CGI-type of binding. The last time I benchmarked this sessionless CGI-type of binding between PHP and R (about 2 years ago) there was a 2 to 3 second delay in web script output mostly attributable to the time it took to (re)start the [R math server](#). A more promising approach to fast and extended communication with R is intimated by the [php-Rserve Project](#). Unfortunately, creating this binding requires a working knowledge of binary communication protocols which has slowed progress on this project down.

Virtues of PHP as a web-based mathematics programming language

1. Excellent language for prototyping web-resident math algorithms meant to be integrated into web sites.
2. If you need more speed, turn it into an extension.
3. Easy to learn as indicated by its immense popularity as the scripting language of choice among web developers.
4. Elementary, junior high, high school and post secondary math instructors would be using PHP more if they applied constructivist and situated activity theory to their practice of guiding math learners.

Flaws of PHP as a web-based mathematics programming language

As a general guideline, the execution speed of a PHP-based linear algebra math algorithm *in the context of a web-based script* is an order of magnitude slower than the execution of that algorithm as a compiled Java program from the command line. Multiply this relative slowness by the number of concurrent users who might access the script

and you can see how some feasibility issues start to rear their head. You may need to create a PECL extension to get things working faster or use a PHP-based heuristic algorithm that might achieve comparable performance (see Daniel Lemire's [Slope One Predictors](#) research).

Some math programmers might regard PHP's indifference to declaring the type of your variable as a problem in achieving verifiable and correct numerical results. They feel the need to declare the "mode" of their variables (e.g., `int $foo = 0;`) before they invoke them. Personally, I prefer to remain indifferent to this aspect of my programs so that the algorithms themselves occupy most of the code and not variable declarations and initializations. Experience to date developing mathematically-oriented programs that are able to exactly reproduce results from the R math server has given me no reason to change this "modeless" approach to variable invocation.

The term "modeless" is how George S. Fishman characterized PHP's type handling in my implementation of his pseudocode for a hypergeometric random variate generator. George's pseudocode was more strict in declaring the specific type or mode of each variable before invoking it, however, it is demonstrably the case that you can let the PHP runtime engine manage this for you and get the same results.

```
<?php
```

```
class HyperGeometricDistribution extends ProbabilityDistribution {

    // snip....

    /**
     * This method set the parameters of the distribution.
     * @param int $m the number of white balls in the urn.
     * @param int $n the number of black balls in the urn.
     * @param int $k the number of balls drawn from the urn.
     */
    function HyperGeometricDistribution($m, $n, $k){
        $this->m = $m;
        $this->n = $n;
        $this->k = $k;
    }

    // snip....

    /**
     * Private hypergeometric RNG method.
     * Implements the "hyp" algorithm described in:
     * George Fishman (2001) Discrete-event simulation: modeling,
     * programming, and analysis. New York : Springer.
     * @returns single hypergeometric deviate
     */
    function _getRNG() {
        $d1 = $this->m + $this->n - $this->k;
        $d2 = min($this->m, $this->n);
        $y = $d2;
        $i = $this->n;
        while (($y * $i) > 0) {
            $u = mt_rand() / mt_getrandmax();
            $y = $y - (int) ( $u + ($y / ($d1 + $i)) );
            $i = $i - 1;
        }
        $z = $d2 - $y;
        if ($this->m <= $this->n)
            return $z;
        else
            return $this->k - $z;
    }
}

?>
```

Constructivism and situated activity

Here I discuss some of my reasons for getting into php + math programming and the philosophical lens through which I approach learning mathematics.

Why I do PHP + Math programming

About 3 years ago I started to think about dedicating more time to my interests in mathematics. I called it my hobby at first and it still is. In the fullness of time it will perhaps evolve into something more, but I also share the Aristotelian view that some things like math, and math programming, are intrinsically worth knowing (i.e., autotelic or an "end in itself").

Here are some of my reasons for beginning to explore the interface between PHP and math:

1. Studied math cognition in graduate school.
2. I know PHP well and math less well. Use PHP to anchor my math learning.
3. The PHP + Math hobby has cumulative effects so it becomes more satisfying with time.
4. Wanted a long-term intellectual challenge (learn math or JAVA).
5. Increase employability
6. Overcome barriers

Why don't others get into this hobby?

Perhaps it would help to know some strategies for learning math on your own. Instead of offering specific methods or techniques, I would suggest that you learn something about the philosophy of mathematics, namely constructivist and situated views of mathematical knowledge.

Constructivist theory of math knowledge

In what follows, I will be painting a cartoon sketch of what mathematical constructivism is and why you should care. Mathematical constructivism is defined in opposition to the view that learning math consists of transmitting concepts from an instructor's head to the student's head via a blackboard or some other broadband transmission device. In contrast, learning math is viewed as a constructive process where the learner is assimilating new knowledge in terms of what they already know using meta-cognitively controlled learning approaches which instructors might help coach them into using or allow them to discover in a tool enriched peer-learning environment. Radical versions of constructivism suggest that there is no mathematical truth to be transmitted from the mind of the instructor to the mind of a learner because truth can only be constructed by the learner in terms of what they already know and care about. [Richard Rorty](#), in *Philosophy and the Mirror of Nature* (1979), did an excellent hatchet job (or deconstruction) on the transmission view of knowledge and is often considered a constructivist sympathizer along with Dewey, James, Piaget, Vygotsky and other latter day pragmatists.

Situated activity theory of math knowledge

Whereas the origins of the constructivist theory of knowledge can be derived from philosophical reflection on the nature of knowledge and fraternizing with literary deconstructivists, the ideas behind the situated activity approach to learning math were the result of trying to come to grips with empirical studies of "math in the wild" (to quote a phrase by Jean Lave who is one of the primary theorists of this approach).

Here is one much-discussed anthropological data point that inspired this approach:

Field anthropologists (see J.Lave, 1988, *Cognition in Practice*) have documented how shoppers use sophisticated cognitive strategies that leverage situated aspects of the problem solving situation to make "best buy" decisions for a particular shelf item (e.g., pack of soap). Shoppers must continually compare costs between products with packages in different sizes at different prices to come up with the "best buy" for a particular consumer good. Experienced house hold shoppers are often successful in making best buy decisions when their purchase decisions "in the wild" are retrospectively examined. When asked to solve formally identical problems expressed via a "word problem" they are often unable to solve the problem. Their mathematical knowledge appears "situated". Detractors of situated theory say the inertness of such knowledge is a bad thing and we should not accept this result as part of our learning theory for math. Detractors are unwilling to accept the idea that much of what we call mathematical knowledge might be situation bound in just these ways - that mathematical knowledge may not easily "transfer" across disparate domains. That context is everything.

The role of PHP in the math curriculum

What will be the role of PHP in the mathematics curriculum of the near future?

PHP will increasingly become the enriched web-aware tool environment used to assist in mathematical discovery and mathematical knowledge construction by elementary students to university professors. One of its primary selling points will be the fact that not only can you do the mathematical computations in PHP but you can also easily

package the math into a web site or web application that shares and applies the knowledge. PHP is a web media programming language and what elementary students to university professors will want to increasingly do with their math knowledge is put it on the public or private web in one form or another. PHP will be increasingly used to help make this happen in contexts where math is authentically taught and learned.

PDL Package

The need for a Probability Distribution Library Package (aka PDL Package) implemented in PHP arose in my first IBM developerWorks article where I [implemented the simple linear regression algorithm](#) and needed to evaluate the significance of the obtained T score. Initially, I did a system call to R to evaluate the probability of the obtained T score. I discovered, however, that doing a system call to R was relatively slow and felt that implementing the probability evaluation in PHP would speed things up. In the [next article in my simple linear regression series](#), I implemented a couple of distribution functions for the F and T distributions in PHP and eventually recognized that I might need to take a more systematic approach to organizing the distribution functions for each probability distribution. I then discovered the [JSci Project](#), in particular, Mark Hale and Jaco van Kooten's library of probability distribution classes and special functions. I ported these classes and functions from Java to PHP, changed the API to use abbreviations, added more distributions and distribution methods, added support for vector-based input/output for the main distribution methods (i.e., PDF, CDF, ICDF, RNG), and created a web demo for each probability distribution. I wrote up some of my early thinking in my 4th article for IBM developerWorks called [Apply probability models to Web data using PHP](#). The very useful Chi-square distribution deserved an article of its own and was the focus of my 3rd article for IBM developerWorks called [Take Web data analysis to the next level with PHP](#).

Current status of the PDL package

The PDL package currently implements distribution functions for 18 distributions. Each distribution implements a probability distribution function PDF, a cumulative distribution function CDF, an inverse cumulative distribution function ICDF, and a random number generator RNG method.

Table 1. Constructor parameterization and method implementation status of current PDL classes.

Distribution	Parameters	PDF	CDF	ICDF	RNG
NormalDistribution.php	mu, sigma	yes	yes	yes	yes
BinomialDistribution.php	n, p	yes	yes	yes	yes
PoissonDistribution.php	lambda	yes	yes	yes	yes
ExponentialDistribution.php	rate	yes	yes	yes	yes
BetaDistribution.php	p, q	yes	yes	yes	yes
FDistribution.php	p, q	yes	yes	yes	yes
GammaDistribution.php	shape	yes	yes	yes	yes
ChiSqrDistribution.php	df	yes	yes	yes	yes
TDistribution.php	df	yes	yes	yes	yes
WeibullDistribution.php	shape	yes	yes	yes	yes
CauchyDistribution.php	location, scale	yes	yes	yes	yes
LognormalDistribution.php	mu, sigma	yes	yes	yes	yes
GeometricDistribution.php	p	yes	yes	yes	yes
ParetoDistribution.php	shape, scale	yes	yes	yes	yes
UniformDistribution.php	min, max	yes	yes	yes	yes
TriangleDistribution.php	a, b, i	yes	yes	yes	yes
NegativeBinomialDistribution.php	k, p	yes	yes	yes	yes
HyperGeometricDistribution.php	m, n, k	yes	yes	yes	yes

Mike Bommarito has also started to research the requirements for integrating multivariate distributions into the PDL Package. Mike has started work on the `MultiNormalDistribution.php`, `MultiNormalDistribution.php`, and `BivariateNormalDistribution.php` classes. It has been difficult to figure out what the architectural relationship of the multivariate classes to the univariate classes should be. Should the multivariate distributions extend the `ProbabilityDistribution.php` base class just like the other univariate classes do? If we want to go this route, then it looks like more polymorphism needs to be added to the `ProbabilityDistribution.php` base class. In addition to adding multivariate distributions to the base class, there are still many more univariate distributions that could be added (e.g., Erlang, Bernoulli, Rayleigh, Gumbel, Pearson family, Johnson family, etc...).

LGPL licensing of the PDL package

I consider the PDL Package to be a descendent of the JSci source tree which is released under the LGPL license. I also think that the LGPL license is a workable licensing scheme for this type of package.

The code base also includes code from other sources besides Mark Hale and Jako van Kooten. I have also lifted code from [Kyle Siegrist](#), [Dawn Duehring](#), [Numerical Recipes in C](#), [Taygata Research](#), [George S. Fishman](#), [Pierre L'Ecuyer](#), [Richard Simard](#), [John C. Pezzullo](#) and many others (i.e., I learned a lot by reviewing code from other opensource PDL-type projects). I also added my own code here and there. I have tried to port from sources with licensing schemes that are either equal to or less restrictive than the LGPL. In the case of Numerical Recipes, I believe that the few places I used it, and the manner of my usage, would continue to make the LGPL a warranted licensing scheme.

Support for vector input

One of the features that I spent a lot of time perfecting was ability to handle scalar or vector input to the probability distribution methods. This means that you can use the same method to obtain either a single return value (if a scalar-type argument is passed to the method) or an array of return values (if an array-type argument is passed to the method).

Support for vector input is useful, for example, if you want to find the shape of your distribution function by supplying the PDF method with a series of values to evaluate the probability density of. You could easily graph the return values to view the shape of the probability distribution.

I wrote an essay length blog on the [Vectorized Function Design Pattern](#) that explains in detail how vector handling is implemented in the PDL Package and argues that the same architectural solution can be used more generally in mathematical programming.

Random number generation

The hardest method to implement for many of the distribution classes was the RNG method. There are many different approaches one might take to generating random variates for a particular distribution. Some rationale is therefore required for why a particular approach was adopted over others. I would like to say that I had the time to survey all the approaches and compare them, but reality dictated that I select RNG algorithms that did not require a lot of code to implement them. At the end of the day, what this means is that many of my RNG methods are not as efficient at random variate generation as they could be. It does not mean that my existing approach generates incorrect variates, only that there may be faster methods of doing it then I am currently using. This can become important in heavy simulation work.

Structural equation modelling

One useful analytical approach for building causal models for a variety of situations is structural equation modelling or SEM. I will illustrate the basic flavor of the approach via an example.

[Bill Shipley](#), in [Cause and Correlation in Biology](#), offers up this structural equation model for photosynthesis yeild as a function of leaf nitrogen and stomatal density.

```
<?php
/**
 * @package PDL
 * @path examples/photosynthesis.php
 *
 * @author Bill Shipley
 * @author Paul Meagher
 *
 * Program to simulate net photosynthesis as a function
 * of leaf nitrogen and stomatal density.
 */
require_once "../NormalDistribution.php";
$num_simulations = 1000;
$leaf_nitrogen = new NormalDistribution(0.035, 1.006);
$stomatal_density = new NormalDistribution(-0.031, 1.017);
for($i=0; $i < $num_simulations; $i++) {
    $leaf_nitrogens[$i] = $leaf_nitrogen->RNG();
    $stomatal_densities[$i] = $stomatal_density->RNG();
    $net_photosynthesis[$i] = 0.003 + 0.527 * $leaf_nitrogens[$i]
```

```
+ 0.498 * $stomatal_densities[$i];
```

```
}  
>
```

JAMA Package

The JAMA project also found its impetus in my earliest research on implementing the simple linear regression algorithm. At the conclusion of that series I was aware that I would need a matrix algebra library in order to extend this research into multiple regression contexts. From scanning some linear algebra and statistics books, I also knew that I would need to perform QR decomposition on the data matrix in order to arrive at a least-squares estimate of the regression coefficients. I found the JAMA package because it has class names that map onto the names of the most common matrix algebra decompositions you are likely to want to use. The simple and elegant structure of the JAMA package combined with the fact that some excellent minds were behind this NIST + MathWorks sponsored project eventually made me want to port the JAMA package to PHP. Along the way, I received help from Lukasz Karapuda, Bartek Matosiuk, and Mike Bommarito.

Most recently, Mike Bommarito added polymorphism to the Matrix.php class, added error handling, developed test and example code, and help to assemble to [documentation for the JAMA package](#)

The joy of matrix algebra

Lately I have started to call the JAMA package a matrix algebra library instead of a linear algebra library. Not sure that it makes much difference, but I do it to emphasize the centrality of the matrix concept in the design and use of this library.

To use this package effectively you need to view all numbers as being housed in matrices and all operations as operations on such matrices. A typical JAMA script will involve instantiating a number of matrices, manipulating those matrices to produce other matrices, and arriving at a final matrix-type result (even if contains only a single value).

Something else you need to know is what a decomposition is. In ordinary arithmetic we can use +, -, *, / operators to take a left hand quantity and re-express it as the arithmetic combination of other quantities (e.g., $5 = 2 + 3$). Similarly in matrix algebra we can re-express a matrix in terms of other matrices bearing arithmetic relationships to each other. The main tools you can use to decompose a matrix into other matrices are these 5 decompositions:

- LUDecomposition.php
- QRDecomposition.php
- CholeskyDecomposition.php
- SingularvalueDecomposition.php
- EigenvalueDecomposition.php

You can learn more about these decompositions by consulting a linear algebra text. Learning linear algebra involves learning these decompositions and how they are used to solve systems of linear equations. JAMA is an excellent package to use for learning and applying linear algebra concepts.

Structure of JAMA

The Matrix.php class is typically the only class you need to include (i.e., `require_once "JAMA/Matrix.php"`) and instantiate (i.e., `$mat = new Matrix($A)`) in your matrix algebra scripts. This is because the Matrix.php class includes all the decomposition methods in the initial part of the Matrix.php class:

```
<?php  
/**  
 * @package JAMA  
 */  
  
define('RAND_MAX', mt_getrandmax());  
define('RAND_MIN', 0);  
  
require_once 'util/Error.php';  
require_once 'util/Maths.php';  
require_once 'CholeskyDecomposition.php';  
require_once 'LUDecomposition.php';  
require_once 'QRDecomposition.php';  
require_once 'EigenvalueDecomposition.php';  
require_once 'SingularValueDecomposition.php';
```

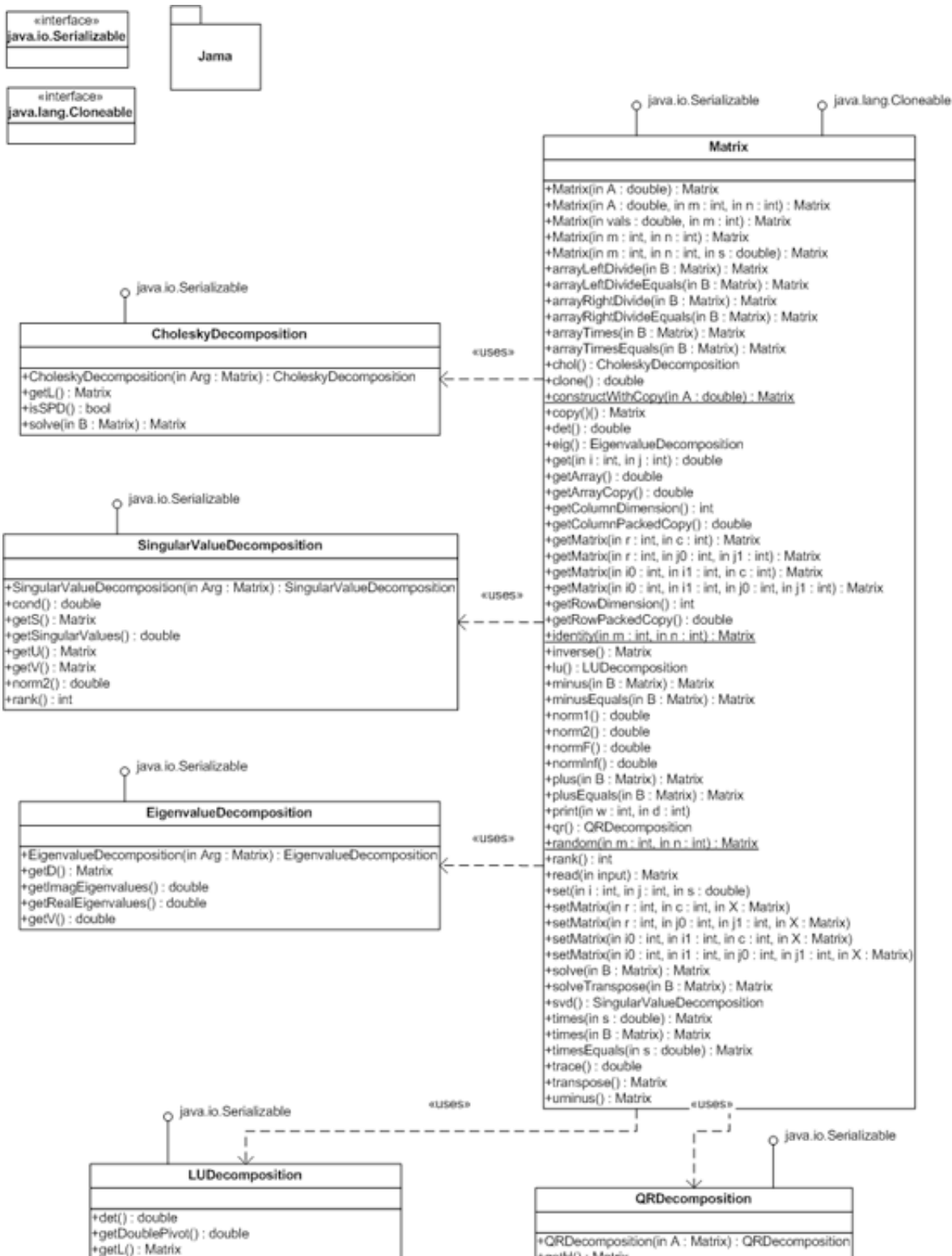
?>

The `Matrix.php` class provides compact public methods for accessing the decomposition methods:

- `$mat->eig()`
- `$mat->lu()`
- `$mat->chol()`
- `$mat->qr()`
- `$mat->svd()`
- `$mat->solve()`

In addition to providing access to these decomposition methods, the `Matrix.php` class also provides a large array of methods that one might expect for manipulating matrices (e.g., `$mat->times($b)`, `$mat->minus($Y)`, etc...).

The complete set of methods supplied by the matrix class as well as its relationship to the decomposition methods can be gleaned from the UML diagram for the package prepared by Lukasz Karapuda:



```
+getPivot(): int
+getU(): Matrix
+isNonsingular(): bool
+solve(): Matrix
+LUdecomposition(in A: Matrix): LUdecomposition
```

```
+getQ(): Matrix
+getC(): Matrix
+getR(): Matrix
+isFullRank(): bool
+solve(in B: Matrix): Matrix
```

Current status

The JAMA package is released as version 1.0.1 which corresponds to the version release number of the original Java-based JAMA package. The version release number is less important than the build number (currently `build-05`) which reflects successive approximations to reproducing all of the original JAMA functionality. While most of the major functionality of the library has been ported there are still a couple areas that should be reimplemented in light of PHP5's new object model. The two main areas are error handling and support for method chaining.

Currently JAMA uses the `trigger_error()` method of error handling because this is a powerful and flexible method that works for both PHP4 and PHP5. The original version of JAMA throws exceptions and we anticipate that as PHP5 comes into more widespread use there will be more incentive to implement the original exception-based method of error handling.

The second major issue that needs to be addressed is what I call "method chaining".

In the original Java version of the JAMA package you can do this:

```
R = L.times(U).minus(M.getMatrix(p,0,n-1));
```

Currently, to reproduce this computation we need to do it in two steps:

```
$S = $L->times($U);
$R = $S->minus($M->getMatrix($p,0,$n-1));
```

I believe that using PHP5's object model we should be able chain methods like this:

```
$R = $L->times($U)->minus($M->getMatrix($p,0,$n-1));
```

I welcome any assistance in developing a `build-06` version of the JAMA v1.0.1 code base that supports exceptions and chained methods.

Learn more

You can learn more about the JAMA package by [consulting the JAMA documentation](#) that accompanies the package. The documentation includes a `TextMatrix.php` class for unit testing all the methods along with a `MagicSquare.php` example script that is useful for benchmarking performance. Both of these scripts should be studied in detail to learn how to use the JAMA package for matrix algebra programming.

Multiple regression

Multiple regression is probably the most popular multivariate statistical technique so it makes an appropriate starting point for illustrating the combined power of the JAMA and PDL Packages.

Zarthan Company example

To illustrate how to solve a multiple regression problem using JAMA and PDL we will develop a script that solves a textbook multiple regression problem. The textbook I consulted is the following book (now available in later editions):

Neter, J., Wasserman, W., Kutner, M.H. (1990). Applied Linear Statistical Models. 3rd Edition. IRWIN: Boston, MA.

This textbook is unique in the clarity and detail that it provides on the matrix algebra involved in solving multiple regression problems. Recommended for independent learners.

The data we will analyze comes from the hypothetical Zarthan Company:

Zarthan company sells a special skin cream through fashion stores exclusively. It operates in 15 marketing districts and is interested in predicting district sales. Table 2 contains data on sales by district, as well as district data on target population and per capita discretionary income. Sales are to be treated as the dependent variable Y , and target population and per capita discretionary income as independent variables X_1 and X_2 , respectively, in an exploration of the feasibility of predicting sales from target population and per capita discretionary income.

Table 2. District population, discretionary income and sales for Zarthan Company

i	Y_i	X_{i1}	X_{i2}
1	162	274	2450
2	120	180	3254
3	223	375	3802
4	131	205	2838
5	67	86	2347
6	169	265	3782
7	81	98	3008
8	192	330	2450
9	116	195	2137
10	55	53	2560
11	252	430	4020
12	232	372	4427
13	144	236	2660
14	103	157	2088
15	212	370	2605

$$Y = \begin{pmatrix} 162 \\ 120 \\ 223 \\ 131 \\ 67 \\ 169 \\ 81 \\ 192 \\ 116 \\ 55 \\ 252 \\ 232 \\ 144 \\ 103 \\ 212 \end{pmatrix} \quad X = \begin{pmatrix} 1 & 274 & 2450 \\ 1 & 180 & 3254 \\ 1 & 375 & 3802 \\ 1 & 205 & 2838 \\ 1 & 86 & 2347 \\ 1 & 265 & 3782 \\ 1 & 98 & 3008 \\ 1 & 330 & 2450 \\ 1 & 195 & 2137 \\ 1 & 53 & 2560 \\ 1 & 430 & 4020 \\ 1 & 372 & 4427 \\ 1 & 236 & 2660 \\ 1 & 157 & 2088 \\ 1 & 370 & 2605 \end{pmatrix}$$

Matrix algebra

The first order regression model that we will estimate and test is:

$$Y_i = b_0 + b_1X_{i1} + b_2X_{i2} + e_i$$

The following is a script that estimates the b parameters, measures the discrepancy between the observed and predicted y values, and partials the total y variance into explained and residual error variance.

```
<?php
```

```
require_once "../Matrix.php";
```

```
$yData = array(162,120,223,131,67,169,81,192,116,55,252,232,144,103,212);
```

```
$xData = array(array(1, 274, 2450),
                array(1, 180, 3254),
                array(1, 375, 3802),
                array(1, 205, 2838),
                array(1, 86, 2347),
                array(1, 265, 3782),
                array(1, 98, 3008),
                array(1, 330, 2450),
                array(1, 195, 2137),
                array(1, 53, 2560),
                array(1, 430, 4020),
                array(1, 372, 4427),
                array(1, 236, 2660),
                array(1, 157, 2088),
                array(1, 370, 2605));
```

```
$X = new Matrix($xData);
```

```
$nY = count($yData);
```

```
$Y = new Matrix($yData, $nY);
```

```
$Xt = $X->transpose();
```

```
$XtX = $Xt->times($X);
```

```

$XtY = $Xt->times($Y);
$XtXi = $XtX->inverse();
$b = $XtXi->times($XtY);

$Yp = $X->times($b);
$e = $Y->minus($Yp);
$Yt = $Y->transpose();
$Ytn = $Yt->times(1/$nY);
$YtY = $Yt->times($Y);

$J = new Matrix( array_fill(0, $nY, array_fill(0, $nY, 1)) );
$YtnJ = $Ytn->times($J);
$YtnJY = $YtnJ->times($Y);

$SSTO = $YtY->minus($YtnJY);

$bt = $b->transpose();
$btXt = $bt->times($Xt);
$btXtY = $btXt->times($Y);

$SSE = $YtY->minus($btXtY);

$SSR = $SSTO->minus($SSE);
?>

```

This version of the script does not generate any output and is being shown so that you can see how compact the solution of multivariate analysis problems are when you have a good matrix algebra library at your disposal. I have also prepared [a more verbose version of this script](#) that outputs all the intermediate matrix results. Note that every intermediate result that is generated and displayed is a matrix even if it is only one value.

ANOVA source table

It is left as an exercise for you to take the above intermediate results and display them in the context of a traditional ANOVA source table report that looks like this:

Table 3. ANOVA Table - Zarthan Company Example

Source of variation	SS	df	MS	F Value
Regression	SSR = 53,844.716	2	MSR = 26,922.358	MSR/MSE = 5679.47
Error	SSE = 56.884	12	MSE = 4.740	
Total	SSTO = 53,901.600	14		

This table partitions the total variance in our Y values into variance that can be accounted for by our first-order regression model (see Regression row) and variance that our model does not account for (see Error row). To formally test whether our multiple regression model is any good (i.e., better than using the mean of the Y values to estimate Y), we can compute an F score that divides our explained variance by our unexplained variance. A large obtained F value means our regression model is accounting for most of the variance in our Y values.

Probability evaluation

Matrix algebra is needed to estimate our model coefficients and generate the values for our ANOVA source table (that summarizes the adequacy of our model). Matrix algebra, however, cannot be used to solve all aspects of multiple regression analysis. Like most other forms of multivariate analysis, you also need a probability distributions library to evaluate the probability of the results obtained via matrix algebra. In this case, we need to use the [FDistribution.php](#) to evaluate the probability of obtaining an F score this extreme by chance.

```

<?php
require_once "PDL/FDistribution.php";
$f = new FDistribution(2, 14);
$p = 1 - $f->CDF(5679.47);
echo $p;
?>

```

The returned p value is 0. This refers to the area of the F distribution curve to the right of our F score value.

Because the F score is so large there is effectively 0 area to the left of the F score which suggests that our first-order regression model is very well supported by the data.

Testing of regression parameters

The multiple regression technique involves some additional steps where we test of the significance of our regression coefficients using the T test. Generation of the table below is left as an exercise.

Table 4. *Testing of regression parameters*

Parameter	Estimate	T score	PR > T	STD Error
Intercept	3.45261279	1.42	0.1809	2.43065049
Population	0.49600498	81.92	0.0001	0.00605444
Income	0.00919908	9.50	0.001	0.00096811

Data mining applications

The JAMA package can be used for the multivariate analysis of discrete random variables, however, it requires the use of dummy coding and is arguably less useful and straightforward than using data mining and categorical data analysis CDA techniques.

Discrete random variables

One plentiful source of discrete random variables are database columns. Many of our database columns can be considered random variables having a finite set of values. Columns that are not discrete can be transformed into discrete columns through various discretization methods. Data mining algorithms are often designed to work on discrete random variables and often work best on such data (see Ian H. Witten, Eibe Frank (1999) [Data Mining](#). Morgan Kaufman).

Using PHP for data mining

PHP is an excellent tool for developing data mining algorithms for two main reasons:

1. You can use a database to do much of the heavy lifting which means your algorithms are very competitive speed-wise with algorithms implemented in other languages. PHP also has excellent integration with all major databases.
2. You can display the results as a web page with more ease and speed than languages that are not integrated into a web server.

Multivariate joint probability

I have written many articles on the analysis of discrete random variables which you can find listed on the left-hand column of [phpmath.com](#). Instead of re-iterating this research, I would rather discuss a problem that is currently occupying my attention. The problem is how to compute the joint probability distribution among all the database columns in a table where all the columns can be viewed as discrete random variables.

In my most recent article, [Calculating Entropy for Data Miners](#), I identified the need to compute multivariate joint and conditional entropy as "future work". In studying this problem, I realized that I would need to solve the prior problem of computing a multivariate joint probability distribution over a set of columns. This is also a foundational problem in data mining in general.

The script

The following is a script that computes the joint probability for 4 columns in the `AlIElectronics.Customers` table mentioned in my [Calculating Entropy for Data Miners](#) article.

```
<?php
/**
 * @package IT
 */
require_once "../config.php";
require_once PHPMATH."/IT/JointProbability.php";
/**
 * Example of how to use JointProbability class.
 */
```

```

$jp = new JointProbability;
$jp->setTable("Customers");
$jp->setQuery("income=high,student=no,credit_rating=excellent,buys_computer=no");
$jp->evaluate();
echo "The joint probability is: ". $jp->p;
?>

```

The output generated by this script is:

```
The joint probability is: 0.071428571428571.
```

This is the probability associated with this particular combination of database column values.

A query we are more likely to be interested in is shown below (notice the change to "yes" at the end of the query):

```
$jp->setQuery("income=high,student=no,credit_rating=excellent,buys_computer=yes");
```

The probability of this particular combination of values is unfortunately 0.

The class

The `JointProbability.php` class uses a simple and fast technique for computing the joint probability over several columns: it uses the column values *to form a key* that can be used to count the number of occurrences of the key in each row and lookup the frequency and probability values for a particular key.

```

<?php
/**
 * @package IT
 *
 * Computes the joint probabilities for a database table and
 * uses these joint probabilities to answer specific joint
 * probability queries.
 *
 * Below is a test table and some test data.
 *
 * CREATE TABLE Customers (
 *   id int(11) NOT NULL default '0',
 *   age enum('<=30','31..40','>40') NOT NULL default '<=30',
 *   income enum('low','medium','high') NOT NULL default 'high',
 *   student enum('no','yes') NOT NULL default 'yes',
 *   credit_rating enum('fair','excellent') NOT NULL default 'fair',
 *   buys_computer enum('no','yes') NOT NULL default 'no',
 *   PRIMARY KEY (id)
 * ) TYPE=MyISAM;
 *
 * Data dump for table `Customers`
 *
 * INSERT INTO Customers VALUES (1, '<=30', 'high', 'no', 'fair', 'no');
 * INSERT INTO Customers VALUES (2, '<=30', 'high', 'no', 'excellent', 'no');
 * INSERT INTO Customers VALUES (3, '31..40', 'high', 'no', 'fair', 'yes');
 * INSERT INTO Customers VALUES (4, '>40', 'medium', 'no', 'fair', 'yes');
 * INSERT INTO Customers VALUES (5, '>40', 'low', 'yes', 'fair', 'yes');
 * INSERT INTO Customers VALUES (6, '>40', 'low', 'yes', 'excellent', 'no');
 * INSERT INTO Customers VALUES (7, '31..40', 'low', 'yes', 'excellent', 'yes');
 * INSERT INTO Customers VALUES (8, '<=30', 'medium', 'no', 'fair', 'no');
 * INSERT INTO Customers VALUES (9, '<=30', 'low', 'yes', 'fair', 'yes');
 * INSERT INTO Customers VALUES (10, '>40', 'medium', 'yes', 'fair', 'yes');
 * INSERT INTO Customers VALUES (11, '<=30', 'medium', 'yes', 'excellent', 'yes');
 * INSERT INTO Customers VALUES (12, '31..40', 'medium', 'no', 'excellent', 'yes');
 * INSERT INTO Customers VALUES (13, '31..40', 'high', 'yes', 'fair', 'yes');
 * INSERT INTO Customers VALUES (14, '>40', 'medium', 'no', 'excellent', 'no');
 */
class JointProbability {

    var $n = 0;

```

```

var $columns = array();
var $values = array();

var $col_freqs = array();
var $col_probs = array();
var $col_labels = array();

var $joint_freqs = array();
var $joint_probs = array();

var $data = array();

var $table = "";
var $select = "";
var $where = "";

var $p = 0;

/* Methods for handling database table input */

function setTable($table) {
    $this->table = $table;
}

function setSelect($sql) {
    $this->select = $sql;
}

function setWhere($sql) {
    $this->where = " WHERE ".$sql;
}

function getSQL() {
    if (empty($this->select)) {
        $sql = " SELECT ";
        foreach($this->columns AS $column)
            $sql .= "$column,";
        $sql = substr($sql, 0, strlen($sql)-1);
        $sql .= " FROM $this->table ";
        if (empty($this->where))
            return $sql;
        else
            return $sql . $this->where;
    } else
        return $this->select;
}

function getFrequenciesFromTable() {
    global $db;
    $sql = $this->getSQL();
    $result = $db->query($sql);
    if (DB::isError($result))
        die($result->getMessage());
    else {
        $n = 0;
        while($row = $result->fetchRow()) {
            $num_cols = count($this->columns);
            $key = "";
            foreach($this->columns AS $column) {
                $val = $row[$column];
                $key .= $val;
                $this->col_freqs[$column][$val]++;
            }
            $this->joint_freqs[$key]++;
        }
    }
}

```

```

        $n++;
    }
    $this->n = $n;
}
return true;
}

/* Methods for handling array input */

function setArray($data) {
    $this->data = $data;
}

function getFrequenciesFromArray() {
    $this->n = count($this->data);
    $num_cols = count($this->columns);
    for ($i=0; $i < $this->n; $i++) {
        $key = "";
        $j = 0;
        foreach($this->columns AS $column) {
            $val = $this->data[$i][$j];
            $key .= $val;
            $this->col_freqs[$column][$val]++;
            $j++;
        }
        $this->joint_freqs[$key]++;
    }
}

/* Shared methods */

function setQuery($columns) {
    $columns = str_replace(" ", "", $columns);
    $clauses = explode(",", $columns);
    foreach($clauses AS $clause)
        list($this->columns[], $this->values[]) = explode("=", $clause);
}

function clear() {
    $this->n = 0;
    $this->col_freqs = array();
    $this->col_probs = array();
    $this->col_labels = array();
    $this->joint_freqs = array();
    $this->joint_probs = array();
}

function evaluate() {
    $this->clear();
    if (empty($this->table))
        $this->getFrequenciesFromArray();
    else
        $this->getFrequenciesFromTable();
    $this->col_labels = array_keys($this->col_freqs);
    $this->getProbabilities();
    $this->getQueryAnswer();
}

function getProbabilities() {
    foreach($this->joint_freqs AS $key=>$val)
        $this->joint_probs[$key] = $this->joint_freqs[$key] / $this->n;
    foreach($this->col_freqs as $column=>$counts)
        foreach($counts as $attribute=>$count)
            $this->col_probs[$column][$attribute] = $count / $this->n;
}

```

```

function getQueryAnswer() {
    foreach($this->values as $value)
        $key .= $value;
    if (!isset($this->joint_probs[$key]))
        $this->p = 0;
    else
        $this->p = $this->joint_probs[$key];
}
}
?>

```

One might also experiment with using multiple keys (i.e., one key per column) to store and lookup joint frequencies and probabilities.

Multivariate conditional probability

You can compute a multivariate conditional probability in terms of a multivariate joint probability using the following formula:

$$p(s \mid s_1, s_2, \dots, s_m) = p(s_1, s_2, \dots, s_m, s) / p(s_1, s_2, \dots, s_m)$$

It is left as an exercise to develop a `ConditionalProbability.php` class that computes a conditional probability using this formula and the `JointProbability.php` class. The script to compute a multivariate conditional probability would look like this:

```

<?php
/**
 * @package IT
 */
require_once "../config.php";
require_once PHPMATH."/IT/ConditionalProbability.php";
/**
 * Example of how to use ConditionalProbability class.
 */
$cp = new ConditionalProbability;
$cp->setTable("Customers");
$cp->setQuery("buys_computer=yes|income=high,student=no,credit_rating=excellent");
$cp->evaluate();
echo "The conditional probability is: ". $cp->p;
?>

```

Computing a conditional probability via joint probability distribution lookups is referred to as "exact inference" and can be contrasted with approximate methods that scale better to very large numbers of variables. I recommend using "exact inference" approaches where possible.

The point of this discussion is to show you that PHP can be aligned with a database to solve complex data mining problems involving more than one or two database columns. Also, the computation of multivariate joint and conditional probabilities is the foundation upon which many data mining algorithms can be built.

Future work

It is worth exploring methods for answering queries involving a mixture of discrete and continuous random variables. The representation of information in terms of probabilities between 0 and 1 provides a unifying framework for combining the continuous and discrete data stored in database columns. The PDL package can be used to re-express discrete and continuous values as probabilities.

Intelligent websites

One of the reasons why PHP developers should care about mathematical programming with PHP is if they want to develop the next-generation of intelligent websites.

The term "intelligence" is a slippery word and there abounds a large amount of useless discussion about the meaning of this term (see "Turing Test" for an example). Jeff Hawkins and Sandra Blakeslee have recently written a book called [On Intelligence](#) which I have scanned and which argues that intelligence comes down to generating predictions:

The brain uses vast amounts of memory to create a model of the world. Everything you know and have learned is stored in this model. The brain uses this memory-based model to make continuous predictions of future events. It is the ability to make predictions about the future that is the crux of intelligence.

I think this is a good working definition of intelligence and one that a PHP-based mathematical programmer can run with.

Patterns of anticipation

In the past I have written about Markov process theory, Bayesian inference, probability modelling, and entropic analysis and how they can be applied to website data via PHP in order to generate predictions. Perhaps my most direct attack on this issue was my IBM developerWorks tutorial [Website user modelling with PHP](#) where I applied Markov process theory to modelling website users and making predictions about what they will do next.

In that tutorial, I cite a passage from chapter 4 of John Lenker's book, [Train of Thoughts](#) (2002), where he defines Intelligent Flowpath Management Systems as follows:

Because Web enterprises seem to struggle so much to provide information in a "structure" that's effective for people, my assertion is that "architecture", both as metaphor and as a process, has outlived its usefulness. The time has come to enlist a more powerful vision that reaches farther. Instead of information architecture, I believe we need to begin developing Intelligent Flowpath Management Systems that procedurally analyze the needs of individuals and then release appropriate sequences of notions. These sequences must be programmed to guide people's minds naturally and in a manner that helps them to build anticipation of that which will succeed in fulfilling their expectations.
- p. 106

While this is an excellent literary account of what we should be looking for in next generation websites, we must keep in mind that the patterns that make this all possible do not really exist as some mystical entity but rather as mathematical relationships between quantitative and qualitative data points. The barrier to entry in terms of building these websites is a knowledge of math and the ability to incorporate it into our websites.

It should be noted that multiple regression is the most commonly used technique for generating predictions about the future. How might we incorporate this multivariate technique, or some variant of it, into generating predictions about our website users?

Nature of the task environment

To develop intelligent websites we need a framework that allows us to formulate the requirements. We might, for example, characterize our predictive variables in terms of whether they are discrete or continuous variables and what probability distribution best represents them. We might also classify our predictive variables as nominal, ordinal, interval or ratio scaled data and then use statistical techniques that are appropriate for such data. None of these classification systems, however, was developed specifically to analyze the requirements for developing intelligent systems. Fortunately, Stuart Russell and Peter Norvig have published such a framework in their excellent 2003 textbook [Artificial Intelligence: A Modern Approach](#). Their framework is called **PEAS** which is short for **P**erformance, **E**valuation, **A**ctuators, and **S**ensors. These are the aspects of the task environment that need to be specified when developing an intelligent system and the agents technology that drives it. Here is a figure from their book (p. 39-40) that illustrates how a **PEAS** specification can be applied to a number of different types of intelligent systems:

Figure 5. Examples of agent types and their PEAS Descriptions.

Agent Type	Performance Measure	Environment	Actuators	Sensors
Taxi driver	Safe, fast, legal, comfortable trip, maximize profits	Roads, other traffic, pedestrians, customers	Steering, accelerator, brake, signal, horn, display	Cameras, sonar, speedometer, GPS, odometer, accelerometer, engine sensors, keyboard
Medical diagnosis system	Healthy patient, minimize costs, lawsuits	Patient, hospital staff	Display questions, tests, diagnoses, treatments, referrals	Keyboard entry of symptoms, findings, patient's answers
Satellite image	Correct image	Downlink from	Dipslay categorization	

analysis system	categorization	orbiting satellite	of scene	
Part-picking robot	Percentage of parts in correct bins	Convey belt with parts; bins	Jointed arm and hand	Camera, joint angle sensors
Refinery controller	Maximize purity yeild, safety	Refinery, operators	Valves, pumps, heaters, displays	Temperature, pressure, chemical sensors
Interactive English tutor	Maximize student's score on test	Set of students, testing agency	Display exercises, suggestions, corrections	Keyboard entry

Part of my reason for showing this table is to broaden our horizons as to the varied types of intelligent applications that PHP might be used for and what types of problems would need to be solved in order to meet these requirements. Each of these examples either involves a web-based component or could be completely implemented using a web programming language such as PHP.

Russell and Norvig go futher in specifying the requirements for intelligent systems by suggesting that we should also characterize the properties of task enviroments (p. 40-44) along the following dimensions:

1. Fully observable versus partially observable
2. Deterministic versus stochoastic
3. Episodic versus sequential
4. Static versus dynamic
5. Discrete versus continuous
6. Single agent versus multiagent

The task environment for a medical diagnosis system, for example, can be characterized as being partially observable, stochoastic, sequential, dynamic, continuous, and single agent. The task environment for an interactive english tutor can be characterized as being partially observable, stochoastic, sequential, dynamic, discrete, and multi agent. We can, of course, disagree with Russell and Norvig's particular characterizations, however, the point is that we should attempt to make such characterizations in order to better understand the requirements for designing intelligent web technologies.

Websites as rational agents

I would like to conclude this section by drawing your attention to Russell and Norvig's definition of a rational agent:

For each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has (p. 36).

If we are to incorporate agent technology into our websites, then we have to ask how we are going to represent, compute, and maximize performance measures. My argument is that *it is all math at the end of the day*. Ergo, we will need to engage in mathematical programming with PHP if we are going to build intelligent websites incorporating agent technology.

The math layer

Do we need web developers who specialize in the design of the math layer for intelligent websites?

If a big company has enough money, should they think about hiring someone to design the "math layer" of their website or should they just treat this as a secondary aspect of the design of their website? If they want to build an "intelligent" website, then they will need to use math libraries to generate the predictions that are characteristic of such websites. Can the web-designer simply plug these math libraries into their existing website or does the website need to be planned from the ground up to accomodate a math layer that can effectively generate real-time predictions?

One example of how the math layer can affect the design of an intelligent website is if you want to use a transition matrix data structure to represent a site visitor or user over time. The usefulness of your transition matrix in generating "next-state" predictions will be dependent on how you define the "states" that your website visitor/user can take on. Whereas a web designer might take a very page centric view on this (just log the page title to the logging engine along with other details about the page access), the person charged with designing the math layer might have a different view on what state should be logged when a user hits the page (mostly associated with the level of granularity to use in assigning states - might be at a higher level then the page level to avoid storing/processing an overly large transition matrix for all users - or might be at a lower level then the page level if

page actions are critical to making useful predictions). Indeed, the math layer consultant might even suggest that the site be designed in certain ways to facilitate the tracking of state information.

Math layer or math priority

Computer scientist like the concept of "layers" because it is a useful tool for managing complexity - assign subsets of the tasks required to solve a problem to specific layers that can be worked on separately from the other layers.

It is too early yet to say whether next-generation websites will require a software layer that can be identified as the "math layer", or whether calling it a layer is as non-sensical as talking about a "security layer". Perhaps we need to talk instead about paying more attention to the mathematical underpinnings of our websites, especially websites that we expect to be intelligent in some way. Do we need to prioritize the search for mathematical techniques to add value to our website with the same vigor that we search for security holes or usability problems in our websites?

Return on investment

The usability folks like to justify their existence by pointing to the return on investment ROI that can be achieved by hiring them to assess the usability of a website and to make suggestions about how to improve it. I think a similar case could be made for hiring an expert in the design of the math layer for a website.

There are two basic approaches to strategy - either achieve profits by focusing on reducing your costs and becoming more efficient (the lean company), or achieve profits by innovating better than your competitors (the idea company). It strikes me that paying someone to assess usability of your website is similar to the tinkering approach characteristic of lean companies, whereas paying someone to consult on the design of your website math layer is more likely to lead to innovating better than your competitors.

While I consider usability to be an important feature of a website I don't think it is a source of bold new ideas for how to build next-generation intelligent websites. You are more likely to find that source in mathematics than usability.

The challenge

As I see it, the challenge for PHP developers interested in building next-generation intelligent websites and web applications is to begin developing the mathematical infrastructure required for such websites. The PDL and JAMA packages offer a strategic starting point for developing that infrastructure.