



TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

ACTA TECHNICA NAPOCENSIS

Series: Applied Mathematics, Mechanics, and Engineering
Vol. 66, Issue I, March, 2023

A REVIEW OF THE PHP SERVER-SIDE SCRIPTING LANGUAGE COMPARED TO C, C++ AND JAVA FOR NUMERICAL ENGINEERING APPLICATIONS

Tiberiu Alexandru ANTAL

Abstract: *The paper is exploring the PHP open-source server-side scripting language, developed for dynamic web pages usage, in the case of specific numerical application in the field of engineering. As a reference, the languages C, C++ and Java are considered, which are successfully used by engineers in solving numerical scientific problems. Being created as a loose typed programming language the problem that arises is related to the fact that under these conditions, which are adapted to the easiest possible use of the language for web pages, it might not face the specific requirements of numerical applications in the field of engineering.*

Key words: *equation, function, graph, PHP, solver, table.*

1. INTRODUCTION

1.1 Some concepts types on loosely typed languages

The type, or data type, is a classification of data based on a set of characteristics described by [1]:

- a domain or set of allowed values;
- a set of operators;
- axioms and
- preconditions.

Consider the mathematical description of a category or class of numbers called integers. Often these are defined by what they do not contain: no fractions of a number, no percentage, or no decimals. Under these conditions the number of allowed values of the integer set of is infinite. In some programming languages (C, C++, Java) the equivalent of the integer category of numbers from mathematics is defined under the *int* type name [2]. Digital computers store everything in binary. All data is stored in binary using specific representations to the type. Integer numbers are stored using unsigned or

signed integer representation, while real numbers use fixed point or floating point representation. The representation of numbers in computer science is based on standards (binary, two's complement, IEEE Standard 754) that limit the storage space that describes the representation. This way the mathematical infinite set of values for integers and reals are restricted to subdomains. The smallest unit of data in binary computers is the binary digit (usually shortened to "bit"). A bit can have one of two values at a given time - one or zero. In order to represent numbers greater than one the hardware on the computer we will have to use several bits. The smallest unit of data that can be addressed in computer memory is the called byte - a group of eight bits. Using a byte to store an unsigned integer reduces the possible set of values to be stored (using the binary representation) to [0, 255] or in the case of signed integer values (using two's complement representation) to [-128, +127]. The number of bits used for storage and the representation of the stored data will influence the domain and the precision of the stored values. Some languages use the same representation and different number of bits, for example, in Java, *byte*, *short*,

int and *long* are all used to store signed integers, however their domain is different ([-128, 127] for *byte* and [-9223372036854775808, 9223372036854775807] for *long*). The set of operators for the *int* type are: subtraction, addition, division, multiplication and remainder with the corresponding symbols (in C, C++ and Java): -, +, /, *, %. If x and y are values of *int* type then some related axioms are: $0+x$ is x and $1*x$ is x ; while a precondition can be formulate at the division x/y which can be solved only if y is different from 0. Once the concept of type is defined two groups of programming languages can be defined based on the how strict the policy of type usage is applied:

- strong typed language:
 - every name must be associated with a single type that is known at the compilation time;
 - name equivalence is used; and
 - every type inconsistency must be reported.
- weak typed language:
 - not every name has to be associated with a single type at the compilation time;
 - structural equivalence is used; and
 - a variable of a subtype is acceptable and implicit type conversion, called coercion, is allowed.

Type checking is defined as verifying that the data types of operands of an operator are legal or equivalent to the legal type. There are two major type equivalence policies: structural equivalence and name equivalence. Name equivalence is the most straightforward, two types are equal if, and only if, they have the same name. In structural equivalence two types are equal if, and only if, they have the same "structure", which can be interpreted in different ways. Consider the following Java sequence:

```
byte b = 1; int i = 2; float f = 1.F; double x;
x = b + i + f;
```

For the above expression the question is which types are equivalent, is *byte* equivalent to *int*, is *int* equivalent to *float*, is *float* equivalent to *double*? Or in other words are the above

operations legal in Java? Java is using the name equivalence policy and b , i , f and x variables are of different types. Some of the types have different domains, different set of operators and different internal representation, so how can the expression be computed by Java? The answer is related to special case of type conversion called widening cast which is done automatically when converting a smaller domain type to a larger size type [6], [7]. The operators in the expression are: assignment (the = symbol) and addition (the + symbol). Based on the priority and associativity rules the first addition must be solved, that is $b+i$, the result is added to f and then this result is assigned to x . For the first addition b which is of type *byte* is widened to *int* and the result is *int*, then for the second addition the *int* type result is widened to *float* and the result is of *float* type. Finally when the assignment operator is reached the *float* result is widened to *double* and stored to the x variable of *double* type. If the expression is written as:

```
b = i + f;
```

the following error is issued by the compiler "incompatible types: possible lossy conversion from float to byte" and in order to be compiled the expression has to be rewritten using the cast operator (mandatory in Java for narrowing cast) as:

```
b = (byte) (i + f);
```

Compared to the *float* type result, the *byte* type has a narrower domain and a different internal representation (with no decimal part), so the above operation might give wrong results. Still, the operation is allowed as the user must write the cast operator explicitly so the language considers that he knows that the result might be bad. Type checking can occur at compile time or at run time. Based on the moment the type checking takes place this can be described as static (checking happens at compile time) or dynamic (checking happens at run time). Java is a strong typed language with static type checking compared to the PHP language which is loose type and dynamically typed. Java is compiled to bytecode and then interpreted by the Java Virtual Machine while PHP is only

interpreted. In a loose typed language the type of a variable can change type depending on how it is used. Sometimes at runtime implicit type conversion can appear if required. PHP 7 (from 2015) introduces some support for statically declared types still the language is more dynamically typed oriented. PHP simplifies everything when it comes to types. For example unsigned types or *byte*, *short*, *int* and *long* (all integer variations in C, C++ or Java) are all simplified into a single integer type. The *char* type is simplified to a *string* type. PHP comes with three different types: primitive or scalar types (boolean, integer, float, string), aggregated or structured or compound types (array, object) and special types (resource, null). The basic of mathematical computations is based on scalar types. A scalar type variable can hold one value, compared the aggregate (homogeneous or heterogeneous) type variables that can hold more values. Languages like C, C++ and Java have some corresponding types from PHP still the PHP simplification might produce sometimes unexpected results. Consider the following valid PHP sequence:

```
$x = " 1 ";
$y = $x + 10; // add a string to an integer
echo $y; //it gives 11
```

This will print *11*, while the following sequence:

```
$x = " 1.1 Hi ";
$y = $x + 10; // add a string to a float
echo $y; //it gives 11.1
```

will print *11.1* while the next sequence will give an error:

```
$x = " Hi 3.14 ";
$y = $x + 10; // add a string to an integer
echo $y; //this gives the following error
```

Fatal error: Uncaught TypeError: Unsupported operand types: string + int in D:\PHP\test.php:2
To someone coming from a strong typed language the previous code sequences seem strange. The simplifications at the level of data types in the language, its quirks related to the conversions specific to the context of usual use

(server-side script language) seem that this is not the happiest choice for an engineer who wants to approach mathematical calculations, to solve equations or to create graphic representations of functions [3]-[5]. The article tries to show that the PHP language can be successfully used in engineering by addressing the three already mentioned contexts specific to engineering.

2. USING PHP IN ENGINEERING

Although the PHP language can be used on its own, in the following it is presented the work with it after installing XAMPP. The reason for this approach is due to the fact that most people who are familiar with the language are related to dynamic web pages, a situation in which the open-source web solution kit XAMPP is widely used. XAMPP is an abbreviation where X stands for Cross-Platform, A stands for Apache, M stands for MariaDB, and the Ps stand for PHP and Perl. Once the installation succeeds the following control panel (see Figure 1) can be used to manage all the components integrated under the XAMPP.



Fig. 1. - XAMPP control panel.

2.1 Function value computation in PHP

The calculation of the values of a function in a given domain at equidistant points in languages that use the procedural, modular or object-oriented paradigm involves writing the expression of the function in the executed code [8], [9]. In the situation where we want to create an application in which the function can be entered from scratch in the form of a string, the programmer must have the knowledge to implement a parser and an expression evaluator.

Languages like C, C++ or Java don't have any standard libraries for such a quick implementation; however in PHP this operation is straightforward. Consider the following HTML code which, by default (see Figure 2), computes the values of the $f(x) = \sin(x)$ function on the $[-10, 10]$ domain with step 1 with the results shown in Figure 4.

```

<!-- Start of FORM -->
<form method="post" action="while3f3.php">
a: <input type="text" name="a" value="-10"><br>
b: <input type="text" name="b" value="10"><br>
step: <input type="text" name="step" value="1"><br>
f(x): <input type="text" name="f" value="sin($x)"><br>
<p><input type="submit" name="submit" value="Submit">
<input type="reset" value="Reset">
</form>
<!-- End of FORM -->

```

Fig. 2. - Input form for $f(x) = \sin(x)$.

Fig. 3. - Input form for $f(x) = \sin(x) \cos(x/2) e^{-x}$.

The results are computed in the `while3f3.php` file with the code that follows:

```

<html>
<head>
<title>While statement with form - while3f3.php</title>
<body>
<TABLE BORDER=1 WIDTH=20%>
<THEADH>
<TR>
<TH>count</TH><TH>x</TH><TH>f(x)</TH>

```

```

</TR>
</THEAD>
<?php
//function is in fin; a - left margin; b - right margin
//step; fin is read in f from the f3.html form
$x = $a = $_POST["a"]; $b = $_POST["b"];
$step = $_POST["step"]; $fin = "\$f=". $_POST["f"]. ";";
$count = 1;
echo $fin.'<br>';
while ($x <= $b):
eval($fin);
printf("<TR><TD>%3d</TD><TD>%7.4f</TD><TD
STYLE=text-align:center>%14f</TD></TR>", $count, $x, $f);
$x += $step;
$count++;
endwhile;
?>
</TABLE>
</body>
</html>

```

count	x	f(x)
1)	-10.0000	0.54402111088937
2)	-9.0000	-0.41211848524176
3)	-8.0000	-0.98935824662338
4)	-7.0000	-0.65698659871879
5)	-6.0000	0.27941549819893
6)	-5.0000	0.95892427466314
7)	-4.0000	0.75680249530793
8)	-3.0000	-0.14112000805987
9)	-2.0000	-0.90929742682568
10)	-1.0000	-0.84147098480790
11)	0.0000	0.00000000000000
12)	1.0000	0.84147098480790
13)	2.0000	0.90929742682568
14)	3.0000	0.14112000805987
15)	4.0000	-0.75680249530793
16)	5.0000	-0.95892427466314
17)	6.0000	-0.27941549819893
18)	7.0000	0.65698659871879
19)	8.0000	0.98935824662338
20)	9.0000	0.41211848524176
21)	10.0000	-0.54402111088937

Fig. 4. - Results for $f(x) = \sin(x)$ computation.

PHP is able to evaluate the value of a correctly formed PHP code stored in a variable that comes from an input form.

Sf=sin(\$x)*cos(\$x/2)*exp(-\$x);

count	x	f(x)
1)	-10.0000	3399.08493383899395
2)	-9.0000	703.93795857354280
3)	-8.0000	1927.74888427248084
4)	-7.0000	674.69202930942629
5)	-6.0000	-111.59616893320634
6)	-5.0000	-114.01634066632155
7)	-4.0000	-17.19519402116493
8)	-3.0000	-0.20050255612002
9)	-2.0000	-3.63020998430194
10)	-1.0000	-2.00734311287590
11)	0.0000	0.00000000000000
12)	1.0000	0.27166434873412
13)	2.0000	0.06648961516275
14)	3.0000	0.00049699614741
15)	4.0000	0.00576834497363
16)	5.0000	0.00517633385803
17)	6.0000	0.00068567055974
18)	7.0000	-0.00056102579892
19)	8.0000	-0.00021693955751
20)	9.0000	-0.00001072096085
21)	10.0000	-0.00000700603622

Fig. 5. - Results for $f(x) = \sin(x) \cos(x/2) e^{-x}$ computation.

Above, the line `$fin="\$f=".$_POST['f'].':';` reads the string from the text field with the f name. To be evaluated based on the x variable the function string from the input must use exactly the same name as the x variable in the input expression. Then the while loop prints the value of the f with the current value of x and updates x based on the step until the upper margin b is reached. So basically PHP allows the user to insert a piece of valid code in the already existing code from the outside and run it as part of it. Although this approach can lead to dangerous code sequences, the presented example describes a clear advantage when we are in the situation of evaluating some functions that, for various reasons, cannot be included directly in the code sequence that wishes to calculate them.

2.2 Solving nonlinear equations with PHP

Often, in applications related to engineering, it is necessary to solve nonlinear equations. Among the known solution methods is that of the bisection (halving the interval) where the user must know the interval $[a, b]$ in which the

solution could be. The method finds the solution if in the given interval there is only one, in the situation where the number of these solutions is greater, in the best case, one will be found. One of the methods by which more solutions can be found in the search interval consists in dividing the initial interval into shorter subintervals. The method will be then applied to each subinterval in the initial interval. The code that follows implements in PHP the described algorithm. The `solve()` function is applied to the subintervals, while the `findallsolution()` function divides the initial interval into subintervals. The `eval()` function from `solve()` evaluates a string as PHP code. The string must be valid PHP code and must end with semicolon.

```
function solve($a, $b, $seps, $f) {
    $fin="\$f=".$f.';';
    $n = 0;
    $max_iter = 70;
```

```
    $x=$a; eval($fin); $fs = $f;
    $x=$b; eval($fin); $fd = $f;
    if (sign($fs) * sign($fd) > 0) {
        //printf("No solution!\n");
        return null;
    }
```

```
    while ((abs($b - $a) > $seps) && ($n++ <
    $max_iter)) {
        $sol = $x = ($a + $b) / 2.;
        eval($fin);
        $fm = $f;
        if ($fm == 0) {
            //printf("The exact root is: ".$x);
            return($x);
        }

        $x=$a;
        eval($fin);
        $fs=$f;

        if (sign($fs) * sign($fm) > 0)
            $a = $sol;
        else
            $b = $sol;
    }
    if (abs($b - $a) > $seps) {
        printf("Precision is not reached in ".$n - 1)."
        iterations.");
        return null;
    }
    else
    {
        //print("Approximate root is : ".$x);
        return $sol;
    }
}
```

```

}
}

function findallsolution($a, $b, $step, $f) {
    $t1="<br><TABLE BORDER=1 WIDTH=20%>
<THEADH> <TR> <TH>Solutions</TH></TR>
</THEAD>";
    $t2="</TABLE>";
    printf("%s", $t1);
    for ($x=$a; $x<$b; $x+=$step):
        $sol=solve($x, $x+$step, 1.e-10, $f);
        if ($sol <> null) echo
"<tr><td>". $sol."</td></tr>";
    endfor;
    printf("%s", $t2);
}

```

Applied to the functions in Figure 2 and Figure 3 the results are shown in Figure 6 and Figure 7 with the length of the subinterval equal to 1.

count	x	f(x)	Solutions
1)	-10.0000	0.54402111088937	-9.4247779608122
2)	-9.0000	-0.41211848524176	-6.2831853072275
3)	-8.0000	-0.98935824662338	-3.1415926536429
4)	-7.0000	-0.65698659871879	-5.8207660913467E-11
5)	-6.0000	0.27941549819893	5.8207660913467E-11
6)	-5.0000	0.95892427466314	3.1415926536429
7)	-4.0000	0.75680249530793	6.2831853072275
8)	-3.0000	-0.14112000805987	9.4247779608122
9)	-2.0000	-0.90929742682568	
10)	-1.0000	-0.84147098480790	
11)	0.0000	0.00000000000000	

Fig. 6. - Solutions for $\sin(x) = 0$ on $[-10, 10]$ interval.

Most of the languages used by engineers are general-purpose computer programming languages (or GPL). The term general-purpose means that the language is widely usable to develop application for many domains and that it has no specialized features for a particular domain. One of the paradoxes of the development of applications written by engineers (non-specialists in programming) consists in the use of the wrong general programming language in favor of specialized ones in an attempt to be in tune with the times.

count	x	f(x)	Solutions
1)	-10.0000	3399.08493383899395	-6.2831853072275
2)	-9.0000	703.93795857354280	-5.8207660913467E-11
3)	-8.0000	1927.74888427248084	5.8207660913467E-11
4)	-7.0000	674.69202930942629	6.2831853072275
5)	-6.0000	-111.59616893320634	
6)	-5.0000	-114.01634066632155	
7)	-4.0000	-17.19519402116493	
8)	-3.0000	-0.20050255612002	
9)	-2.0000	-3.63020998430194	
10)	-1.0000	-2.00734311287590	
11)	0.0000	0.00000000000000	

Fig. 7. - Solutions for $\sin(x) \cos(x/2) e^{-x} = 0$ on $[-10, 10]$ interval.

As can be seen, the only problems appear when a solution is exactly on the edge of the chosen subintervals, a situation in which it is repeated (doubled). The problem can be solved simply if the length of the subinterval changes slightly as shown in Figure 8.

count	x	f(x)	Solutions
1)	-10.0000	3399.08493383899395	-6.2831853072159
2)	-9.6000	225.21335748440126	-9.3127927591752E-11
3)	-9.2000	247.40513740356471	6.2831853072159
4)	-8.8000	1192.60009442042610	
5)	-8.4000	1863.21585017391340	
6)	-8.0000	1927.74888427247970	
7)	-7.6000	1529.80520615203750	
8)	-7.2000	953.31083004135633	
9)	-6.8000	428.90873955534590	
10)	-6.4000	70.02494840459025	
11)	-6.0000	-111.59616893320730	

Fig. 8. - Solutions for $\sin(x) \cos(x/2) e^{-x} = 0$ on $[-10, 10]$ interval with 0.8 subinterval length.

2.3 Tendencies in general purpose programming languages used in numerical engineering

For example, developed around the end of the 50s (1957), by John Backus, the FORTRAN GPL was created specially to numeric computation and scientific computing. Today's FORTRAN is multi-paradigm covering paradigms as: structured, procedural, object-oriented, generic, and is able to do concurrent as well as parallel

programming. However, today, FORTRAN is ignored by a huge part of the numerical programming community and in its place the C and C++ languages are preferred. The C GPL, developed by Denis Ritchie at the beginning of the 70s (1970), was created to reflect, at higher level, the architecture of the CPU on which the code was running. C is also a multi-paradigm language supporting paradigms as: structured and procedural. Today's programmer's community uses the language mainly to develop operating systems, translators, and drivers and to program microcontrollers. As you can see, C does not implement support for modern paradigms, which is why C++ appears. C++ is a GPL, developed by Bjarne Stroustrup first released in 1985 as an extension of the C or "C with Classes". Again C++ was conceived as a multi-paradigm solution covering paradigms as: procedural, imperative, functional, object-oriented, generic, and modular. The fields in which the language has found applicability today are: operating systems, microcontrollers, embedded systems desktop applications, video games, and servers. C and C++ were built to be able to interact with the hardware services of the Operating System (OS) called system functions. This means that all OS services can be accessed effectively from these languages, but the programmer must know the OS and how to use its system functions very well that is, he must be a computer specialist. However, most of the engineers that will do some programming are not and they must be able to create graphics, communicate with devices over the network, perform multitasking or interact with databases using some kind of GUI (Graphical User Interface). Java was developed as a GPL by James Gosling at Sun Microsystems. Released in 1995 and planned to let programmers Write Once, Run Anywhere (WORA). The syntax of Java is similar to C and C++, but has less low-level facilities to prevent writing dangerous sequences of code that might lead to errors or security problems. The language is multi-paradigm supporting with language constructs: generic, object-oriented (class-based), functional, imperative, reflective and concurrent paradigms. Today Java runs on smart cards, laptops, desktop computers, data centers, game

consoles and scientific supercomputers using libraries called packages through which part of the hardware services of the OS are made available to any Java programmer, in a simplified and unified form, but sufficient to develop applications that require access to the hardware services of the machine on which it is running. This means that a Java programmer can concentrate on the application development and not on the study of the OS used to run the application. Basically, 2D and 3D graphics, networking, database programming, and multitasking facilities are all part of the language or its packages. The price paid is related to learning the language, the language packages and the thinking of the solution that needs to be implemented using the object-oriented paradigm. The language learning effort is comparable to that of C and C++ (both strong typed languages) however, the time consumed to learn the packages is considerable longer as these provide many OS low level services at a higher level. For someone who needs fast results PHP is probably the best solution if graphics has to be involved. C and C++ provide no standard graphical primitives, while Java does by using AWT, Swing or JavaFX, which require time to be apprehended.

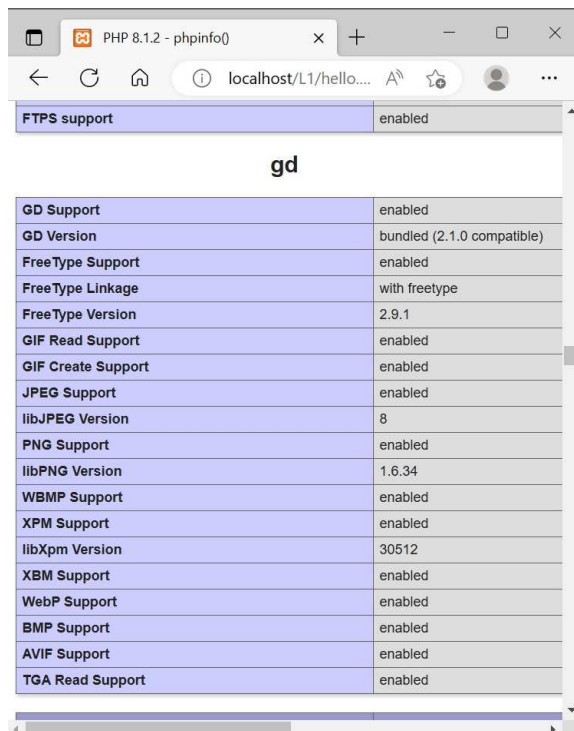
2.4 Graphical representation of functions in PHP

The PHP was created by Rasmus Lerdorf in 1993 as general-purpose scripting language oriented toward web development. PHP code is processed on a web server by a PHP interpreter implemented as a plug-in software (or module) component that adds the feature to the server. The language multi-paradigm support is: imperative, functional, object-oriented, procedural, and reflective. Beside the web context PHP has been used for computations, graphical applications and robotic drone control. Graphics creation in PHP is based on the GD extension (a special library that provides a functionality that can be used by all applications) which is using the open source GD graphics library. XAMPP will add this extension to the `php.ini` sometimes commented (preceded by the character semicolon as `;extension=gd`)

and all we need is to uncomment it in order to run. To check the state of the extensions the `phpinfo()` function [10] inserted in a PHP page is used. In the following example the source code of the page follows and the file name is `hello.php` and the code must run on the webserver:

```
<?php
echo "Hello world from a PHP script!";
phpinfo();
?>
```

The results are shown in Figure 9 in the section entitled **gd**.



gd	
GD Support	enabled
GD Version	bundled (2.1.0 compatible)
FreeType Support	enabled
FreeType Linkage	with freetype
FreeType Version	2.9.1
GIF Read Support	enabled
GIF Create Support	enabled
JPEG Support	enabled
libJPEG Version	8
PNG Support	enabled
libPNG Version	1.6.34
WBMP Support	enabled
XPM Support	enabled
libXpm Version	30512
XBM Support	enabled
WebP Support	enabled
BMP Support	enabled
AVIF Support	enabled
TGA Read Support	enabled

Fig. 9. - `phpinfo()` function results when GD graphics is enabled in PHP.

The file called `drawses.php` used to generate graphics in the application has the following code:

```
<?php
session_start();

//get the session variable for the data to be drawn
$xa=$_SESSION['xa'];
$ya=$_SESSION['ya'];

$xmin=min($xa);$xmax=max($xa);
$ymin=min($ya);$ymax=max($ya);
```

```
define("LX",600);define("LY",400);
define("x0", 0);define("y0", LY);

for($count=0; $count<count($xa); ++$count):
if ($ya == null) continue;
$xs[$count] = (int) (x0+($xa[$count] - $xmin)/($xmax-
$xmin)*LX);
$ys[$count] = (int) (y0-($ya[$count] - $ymin)/($ymax-
$ymin)*LY);
endfor;

// Create an image using imagecreate() function
$img = imagecreate(LX, LY) or die ("Cannot Initialize
new GD image stream");
```

```
// Allocate the colors
$white = imagecolorallocate($img, 255, 255, 255);
$pink = imagecolorallocate($img, 255, 105, 180);
$red = imagecolorallocate($img, 255, 0, 0);
$grey = imagecolorallocate($img, 22, 22, 22);

// Set the thickness of the line
imagesetthickness($img, 1);

// Draw three rectangles each with its own color
$py0=(int) (y0-(0 - $ymin)/($ymax-$ymin)*LY);
$px0=(int) (x0+(0 - $xmin)/($xmax-$xmin)*LX);
$stepx = pow(10,(int)(log10(($xmax-
$xmin)/2)/log10(10)))/($xmax-$xmin)*LX;
$stepy = pow(10,(int)(log10(($ymax-
$ymin)/2)/log10(10)))/($ymax-$ymin)*LY;
```

```
if ($xmin*$xmax <= 0) :
//draw oy at x=0
imageline($img, $px0, 0, $px0, LY, $pink);

//$divx = $xmin+($xmax-$xmin)/LX*$stepx;
$divx = ($xmax-$xmin)/LX*$stepx;

imagestring($img,5,$px0+$stepx,$py0,"step x >
". $divx,$pink);
```

```
for($x=$px0+$stepx; $x <= LX ; $x+= $stepx):
imageline($img, $x, 0, $x, LY, $grey);
endfor;
for($x=$px0-$stepx; $x >= 0 ; $x-= $stepx):
imageline($img, $x, 0, $x, LY, $grey);
endfor;
endif;
```

```
if ($ymin*$ymax <= 0) :
//draw ox at y=0
imageline($img, 0, $py0, LX, $py0, $pink);

//$divy = $ymin+($ymax-$ymin)/LY*$stepy;
$divy = ($ymax-$ymin)/LY*$stepy;

imagestring($img,5,$px0,$py0-$stepy,"step y ^
". $divy,$pink);
```



```

for($y=$py0+$stepy; $y <= LY ; $y+= $stepy):
    imageline($img, 0, $y, LX, $y, $grey);
endfor;
for($y=$py0-$stepy; $y >= 0 ; $y-= $stepy):
    imageline($img, 0, $y, LX, $y, $grey);
endfor;
endif;

imagesetthickness($img, 2);
imagerectangle($img,0,0,LX-1,LY-1,$grey);
for($count=0; $count<(count($xa)-1); ++$count):
    imageline($img, $xas[$count], $yas[$count],
$xas[$count+1], $yas[$count+1], $red);
endfor;
session_destroy();

// Output the image
header('Content-type: image/png');
imagepng($img);
imagedestroy($img);
?>

```

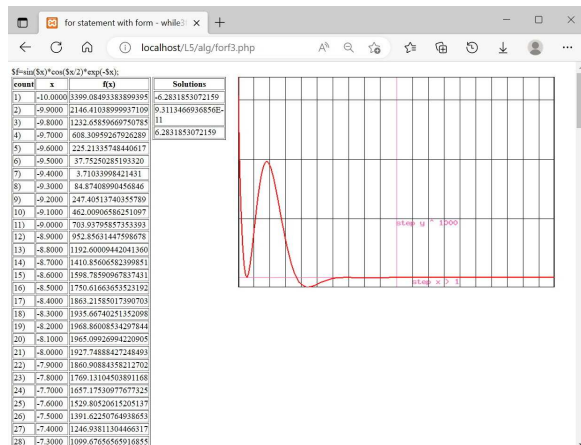


Fig. 10. - Graphical representation of the $f(x) = \sin(x) \cos(x/2) e^{-x}$ function on $[-10, 10]$ interval with 0.1 step.

The input data is transferred to the file with the help of the `$xa=$_SESSION['xa'];` `$ya=$_SESSION['ya'];` lines using the superglobal variable called `$_SESSION`. These variables contain arrays of values that are loaded by any part of the application that computes the values of the function over a given interval $[a, b]$ as shown in the `compute()` function that follows:

```

function compute($a, $b, $step, $f) {
    session_start();
    $t1="<TABLE BORDER=1 WIDTH=20%>
<THEADH> <TR> <TH>count</TH> <TH>x</TH>
<TH>f(x)</TH> </TR> </THEAD>";
    $t2="</TABLE>";
    $fin="\$f=" . $f . ";";
    echo $fin . "<br>";
    printf("%s", $t1);

```

```

for ($count = 1, $x=$a;
$x<=$b;$count++, $x+= $step):
    $xa[] = $x;
    try {
        eval($fin);
    } catch(DivisionByZeroError $e)
    {
        printf("<TR><TD>%3d</TD><TD>%7.4f</TD><TD
STYLE=text-align:center>%14f</TD></TR>", $count, $x, $f);
        $ya[] = $f;
    }
    printf("<TR><TD>%3d</TD><TD>%7.4f</TD><TD
STYLE=text-align:center>division by
zero</TD></TR>", $count, $x);
    $ya[] = null;
}
catch(Exception $e) {
    printf("<TR><TD>%3d</TD><TD>%7.4f</TD><TD
STYLE=text-align:center>got $s
</TD></TR>", $count, $x, $e);
    $ya[] = null;
}
endfor;
printf("%s", $t2);

```

//load the session variables with data

```

$_SESSION['xa'] = $xa;
$_SESSION['ya'] = $ya;
}

```

The PHP image generation is dynamic and starts with the `imagecreate()` function which returns an image handle. An image is a rectangle of pixels that has various colors. The colors used in the image must be allocated with `imagecolorallocate()` function. The first allocated color becomes the background color for the image. Color arguments are the numeric RGB (Red, Green, Blue) components of the color [11]. GD has many graphical primitives. `imageline()` is used to draw a line in the image. Once all the lines are drawn the next step is to send the Content-Type header with the appropriate content type of the created image to the browser. After this the appropriate output function is called, in our case `imagepng()`, to create the image file from the drawing content.

3. CONCLUSIONS

From what has been presented, it can be seen that PHP can be used successfully in solving some numerical problems used in engineering. Although the language is loose typed, it allows

the evaluation of complex mathematical expressions without jeopardizing the accuracy of the calculations or the duration of the evaluation. The presentation of the results is done through HTML pages whose content can be very easily redirected to text editors for further processing. PHP is an easy-to-learn language, which can also be run in virtualized contexts without problems. Being created for the development of dynamic web pages, it allows simple interaction with databases in the situation where the results of the calculations must be stored for further analysis.

4. REFERENCES

- [1] Bertrand Meyer, *Object-oriented software construction*. Prentice Hall PTR, 1997, p.1254, ISBN 0-13629155-4.
- [2] ANTAL, T.A., *Limbajul C ANSI*, Editura RISOPRINT, 2001, p. 253, ISBN 973-656-065-1.
- [3] ANTAL, T. A., *A java client-server model to solve the forward and the inverse robot kinematics*. ACTA TECHNICA NAPOCENSIS - Series: APPLIED MATHEMATICS, MECHANICS, and ENGINEERING, v. 62, n. 1, apr. 2019. ISSN 2393-2988.
- [4] ANTAL, T. A., *The power lost by friction, between the teeth flanks, for cylindrical spur gears, at the points where the meshing stars and ends*. ACTA TECHNICA NAPOCENSIS - Series: APPLIED MATHEMATICS, MECHANICS, and ENGINEERING, v. 57, n. 1, mar. 2014. ISSN 2393-2988.
- [5] ANTAL, T. A., *Some aspects of graphical process simulation*. ACTA TECHNICA NAPOCENSIS - Series: APPLIED MATHEMATICS, MECHANICS, and ENGINEERING, v. 55, n. 1, mar. 2012. ISSN 2393-2988.
- [6] ANTAL, T. A., *Elemente de Java cu JDeveloper - îndrumător de laborator*, Editura UTPRES, 2013, p.150, ISBN: 978-973-662-827-6.
- [7] ANTAL, T. A., *Java - Inițiere - îndrumător de laborator*, Editura UTPRES, 2013, p. 246, ISBN: 978-973-662-832-0.
- [8] PUSCA, A., RUS, G., BIRLESCU, I., VAIDA, C., PISLA, A., SCHONSTEIN, C., GHERMAN, B., TUCAN, P., PISLA, D., *Workspace analysis of two innovative parallel robots for single incision laparoscopic surgery*. ACTA TECHNICA NAPOCENSIS - Series: APPLIED MATHEMATICS, MECHANICS, and ENGINEERING, v. 65, n. 2S, nov. 2022. ISSN 2393-2988.
- [9] PISLA, D., BIRLESCU, I., MOIS, E., TUCAN, P., RADU, C., BURZ, A., GHERMAN, B., ANTAL, T., VAIDA, C., Nadim AL HAJJAR. *Simulation and control of an innovative medical parallel robot used for hcc treatment procedure*. ACTA TECHNICA NAPOCENSIS - Series: APPLIED MATHEMATICS, MECHANICS, and ENGINEERING, v. 64, n. 1-S2, mar. 2021. ISSN 2393-2988.
- [10] <https://www.php.net/docs.php>
- [11] ANTAL Tiberiu Alexandru, *Proiectarea paginilor Web cu HTML, VBScript și ASP - ediția a II-a*, Editura RISOPRINT, 2006, p.264, ISBN 973-751-349-5.

O examinare a limbajului de script PHP pe partea de server, în comparație cu C, C++ și Java, în contextul unor aplicațiilor numerice din inginerie

Lucrarea investighează caracteristicile limbajul de programare PHP, gratuit, de tip script pe partea de server, dezvoltat pentru utilizarea dinamică a paginilor web, în cazul aplicațiilor numerice specifice domeniul ingineresc. Drept referință sunt considerate limbajele C, C++ și Java care sunt utilizate cu succes de ingineri în soluționarea problemelor științifice cu caracter numeric. Fiind creat ca un limbaj de programare slab tipizat se pune problema că în aceste condiții, care sunt adaptate la utilizarea cât mai ușoară a limbajului pentru lucrul cu pagini web, să nu facă față cerințelor specifice aplicațiilor numerice din domeniul ingineriei.

ANTAL Tiberiu Alexandru, Professor, dr. eng., Technical University of Cluj-Napoca, Department of Mechanical System Engineering, antaljr@bavaria.utcluj.ro, 0264-401667, B-dul Muncii, Nr. 103-105, Cluj-Napoca, ROMANIA.