SCIENTIFIC COMPUTING AND PROGRAMMING IN THE CLOUD USING OPEN SOURCE PLATFORMS: AN ILLUSTRATION USING WEIGHTED VOTING SYSTEMS

Mohamed I Jamaloodeen
Georgia Gwinnet College
School of Science and Technology
1000 University Center Lane
Lawrenceville, GA 30043
mjamaloo@ggc.edu

Abstract:

There are by now several cloud computing engines, the most widely known of which is wolframalpha.com. One may type Mathematica-like commands and syntax into a dialog box on a web browser and have the computational results performed on a Mathematica server and these sent back to the client web browser. There are several other cloud computers including "R on cloud" (for R statistical computing) and Scilab on Cloud India (for Scilab numerical computing). Scilab is an open-source numerical computational platform similar in many respects to Matlab. We illustrate the ease and versatility of the Scilab cloud computer with several short voting routines. We present Scilab routines to calculate voter power indices based on (1) the Banzhaf power index, (2) the Shapley-Shubik power index, (3) The Johnston power index and (4) the randomized Shapley-Shubik power index calculator based on the Central Limit theorem. We use these algorithms to illustrate the ease of writing short Scilab programs as well as to illustrate the powerful built-in routines in Scilab especially for random number generation. We then perform these calculations by simply copying and pasting the instructions into a web browser to be performed by the Scilab cloud server and the results sent back to the client web browser. It can be argued that cloud computing is a trend that may compete, if not displace, the apps paradigm for mobile devices. As an alternative to installing apps on a mobile device a user with an Internet connection and a browser interface may choose the computing power offered through a cloud computing service.

Introduction

Three trends that are seen with STEM education, evident even at the secondary education level, are the moves toward introducing programming, introducing scientific computing and using mobile technologies in schools. We illustrate the versatility of the Scilab scientific computing environment not only as a powerful enterprise computing tool but one quite suited for introducing school students to scientific computing and programming, and in particular on a mobile platform using cloud computing. We show how the Scilab platform addresses each of these by applying its capabilities to solving simple weighted voting system problems.

1. The trend to programming in high school.

Consider the aims of the code.org consortium: "Every student in every school should have the opportunity to learn to code." Its goals include,

- Bringing Computer Science classes to every K-12 school in the United States, especially in urban and rural neighborhoods
- Changing policies in all 50 states to categorize Computer Science as part of the math/science "core" curriculum

2. The trend to scientific computing in high schools.

Scilab, for instance, is installed in Chinese and French high schools and is recognized of pedagogical interest by the French Ministry of Education. In the US some AP statistics teachers

use a similar powerful open source statistical computing and programming environment, called R to introduce statistical computing and graphics. There are also cloud based versions of the R statistical platform. Another powerful open source platform (also available in the cloud) is the Sage mathematical software suite that is also being promoted to teach calculus in schools. Its strengths include powerful graphics, the Python programming environment and unlike Matlab or Scilab which are computational, Sage also has sophisticated computer algebra capabilities similar to those in Maple or Mathematica. Sage features are also available in the cloud.

3. The trend to mobile computing

Students and scholars today use a variety of mobile devices for queries including mathematics, making use of such paid or free services or apps like wolframalpha.com and Math Ref. wolframalpha.com is a cloud-based service. Using the internet, even on a mobile device queries can be made to harness the computational power of the wolframalpha.com engine. Scilab, R, and Sage, can also be leveraged using cloud based services. In this paper we focus on Scilab, deployed either on a standalone platform, or in the cloud through a browser on a mobile device or desktop. It can be argued that cloud computing is a trend that may compete, if not displace, the apps paradigm for mobile devices. Rather than installing apps on a mobile device a user with an internet connection may use the computing power offered through a cloud computing service.

We show how an open source scientific platform like Scilab both can be used as a computational tool as well as a high level programming environment suitable for all users from school through the enterprise level. We illustrate its features using both a) the standalone installation of Scilab, and (b) deploying Scilab through the cloud using the browser on an internet-connected browser. We begin by introducing the Scilab scientific computing and programming platform.

Introducing Scilab

Scilab is a freely distributed open source scientific computing software package which can be a standalone installation on a Linux, Windows or Mac OS X system. The Scilab computational features can also be harnessed using a cloud service as offered by http://hotcalcul.com/on-line-calculator/compute and http://Scilab-test.garudaindia.in/. Scilab is similar to the commercial Matlab platform but in many ways is just as powerful. Scilab consists of three main components:

- an interpreter
- libraries of functions (Scilab procedures)
- libraries of Fortran and C routines

Scilab strengths include working with matrices (basic matrix manipulation, concatenation, transpose, inverse, etc.) and numerical computations. It also has an open programming environment that allows users to create their own functions and libraries. We demonstrate how a system like Scilab can be used to solve computational math problems that might be encountered in school. We illustrate with the weighted voting problem and analyze these using short Scilab programs. These routines can be saved in a text file, and then pasted in the Scilab console, either

on the Scilab desktop version, or through the cloud using a web interface to a Scilab cloud server. These examples demonstrate the programming, computational and mobile cloud computing capabilities of an open source platform like Scilab. They also serve to illustrate the strong simulation and probability tools available in Scilab. We proceed by introducing the weighted voting problems and solve these using Scilab scripts/routines and tools.

Weighted voting systems

In a weighted voting system the voters each have a voting weight (or number of votes) they can use to cast on a motion. There is also a quota that is the minimum number of votes necessary to carry a resolution. An example of a weighted voting problem would be:

Voter:	Weight	Quota =5
A	4	
B	2	
C	1	

Figure 1: A weighted voting system

We will use the notation [q; W] = [q; A B C] = [5; 4 2 1] to denote this weighted voting problem. A voter is critical in a winning coalition if without that voter's participation the coalition would not be a winning one. The power of each voter, takes into account the number of winning coalitions that voter is critical in, and is not necessarily a simple function of the weights. In this example the power of voter B and C is the same even though C has half the strength. The winning coalitions for this example are (A, B), (A, C) and (A, B, C). Notice that B and C participate in the same number of winning coalitions. There are two common methods used to determine the power of a voter—the Banzhaf power index and the Shapley-Shubik power index. We also discuss a third called the Johnston power index.

1) The Banzhaf power index for a voter is calculated by finding the total number of coalitions that player is critical in, say B_i, and then dividing B_i by the total sum of all B_k
Letting P_i denote the power index of candidate, we have

$$P_i = \frac{B_i}{\sum_{k=1}^{n} B_k}$$

Implementing a Scilab program to compute the Banzhaf power index involves computing all 2^10 coalitions and determining those that were winning and the voters critical in each winning coalition. In Scilab the built in binary conversion routines simplified the program. For example with 4 voters there are 16 coalitions. Coalitions were determined by finding the binary representation of each coalition. For example 13=1101_2 would be the coalition with A, B and D (not C). In general there are about 2^n computations.

2) The Shapley-Shubik power index considers all permutations of voters as coalitions using sequential coalitions to measure the order in which that voter entered the coalition. A critical player is the player in a sequential coalition who changes the coalition from a losing to a winning one. For example in the coalition (A,B,C) the critical player is B since until the 2nd person, B, entered the coalition A only had 4 voters, 1 less than needed to carry the vote. In the coalition (B,C,A) voter A is critical since before A entered the coalition the prior 2 voters had acquired only 3 voters, 2 less than needed to carry the vote. In Scilab a routine to calculate the Shapley-Shubik power index was greatly facilitated by the built in combinatorial routines including the "perms," routine which calculated all possible lexicographic permutations of n voters. Notice that there n! ordered coalitions to consider in the Shapley-Shubik power index which becomes computationally intensive beyond n=8 or n=9. For n=10 there are 10! = 3,628,800 permutations to consider many orders more than for the Banzhaf power index. To also keep the Scilab code for this routine simple we used the built in Scilab perms routine, which however stores all n! permutations requiring a lot of memory. For this reason the Shapley-Shubik index calculator is restricted to n=8. For $8 < n < 21$ we present a modified Shapley-Shubik power index calculator routine based on sampling theory. This routine uses a sample of random permutations to approximate the Shapley-Shubik power index vector for n voters. Use can be made of the Central Limit Theorem to approximate the error in the approximation depending on the sample size. The sample size is specified by the user. We illustrate, for the voting scenario [q; W]=[12; 9 9 9 9 8 8 8 6 5 5 4 3] with sample size, 100,000 of the 12! total sequential coalitions. (See Figure 2 and also a screenshots in Appendix 2).

Contestant by Weight	9	9	9	9	8	8	8	6	5	5	4	3
Shapley-Shubik Index (actual)	.103	.103	.103	.103	.0939	.0939	.0939	.0667	.0667	.0667	.0667	.0394
Random /sampled Shapley Index (sample size 100,000)	.10423	.10284	.10315	.10390	.09341	.09349	.09261	.06752	.06672	.06722	.06571	.03920

Figure 2: Comparing the Shapley-Shubik index found exactly vs one using a sample of 10,000 of the 12! coalitions.

The sampling routine illustrates one of the strong features of Scilab. The built-in Scilab random number generator 'grand,' had capabilities for many distributions (including multivariate) as well as for sampling from Markov chains and random permutations. It also offers a variety of at least six base-generating random algorithms to choose from—the default being the Mersenne-Twister.

3) The Johnston power index is a variation on the Banzhaf power in which a player's index takes into account the size of the coalition for which that voter is critical. Small coalitions for which a voter is critical contribute more to the power index of that voter than larger coalitions for which

that voter is critical. A coalition of three in which a voter is critical contributes a weighted critical value of 1/3 to that voters index, whereas a coalition of two in which that voter is critical contributes a weighted critical value of 1/2 to that voter's index. A voter's index is found by summing all that voter's critical values by the total sum of all voters' critical values. Denote by W_i the sum of all weighted critical values for candidate i, then again if P_i denotes the power index of candidate i we have

$$P_i = \frac{W_i}{\sum_{k=1}^{n} W_k}$$

The power indices for the 3 different measures for the voting system [5; 4 2 1] are summarized:

Voter	Banzhaf Index	Shapley-Shubik Index	Johnston Index
A	3/5	2/3	4/7
B	1/5	1/6	3/14
C	1/5	1/6	3/14

Figure 3: The voting power indices for the voting system [5; 4 2 1].

The Scilab power index routines/scripts for the 4 weighted voting procedures are provided in Appendix 1 as well as details on how to use them. In the next section we describe how to use and become familiar with Scilab and provide information on using the standalone desktop version of Scilab as well as on how to access two Scilab cloud servers.

Using Scilab

An introduction to using Scilab as part of an integrated math curriculum can be found at my blog, http://scilab-scholar.blogspot.com/ . The version suitable for high schools is the basic version of Scilab to which is added the Scilab module for high schools "Scholar." The "Scholar," module contains functions specific to teaching mathematics in high school and I have made it available http://atoms.scilab.org/toolboxes/Scholar_Lycee_in_English. The standalone or desktop version can be downloaded from Scilab at: http://www.scilab.org/download/5.4.1
An extensive repository of modules or toolboxes can be found at: http://atoms.scilab.org/.
Becoming familiar with Scilab primers and tutorial can be found at: http://www.scilab.org/resources/documentation/tutorials. Additional resources and Scilab books can be found at: http://www.scilab.org/resources/documentation/books. There are several Scilab cloud servers. We mention two that we used simply by pasting code in their web based consoles.
A) http://scilab-test.garudaindia.in/. This was the best Scilab cloud server we found for several reasons; including a) it also has graphics capabilities and b) users do not have to register. Use simply by typing or preferably pasting commands or Scilab routines from a notebook or text file.
B) http://hotcalcul.com/. We do recommend this cloud server even though it does require registration using a valid email address. This cloud server performed best on Android devices.

Conclusion

We described trends in STEM education especially at the secondary education level as the moves toward introducing computer programming, introducing scientific computing using mobile technologies in schools. We illustrated these using the powerful Scilab open source scientific computing and programming environment. The computational power of Scilab may be used on a standalone desktop or using a cloud server through a mobile device. Cloud computing may compete, if not displace, the apps paradigm for mobile devices. Instead of installing apps on a mobile device a user with an internet connection and a browser may choose the computing power offered by cloud computing services. These provide access to high-end computational resources such as the Scilab scientific computing and programming environment, which cannot be downloaded fully onto a mobile device. In follow-on work we plan on showcasing explicitly the programming and computational capabilities of a platform like Scilab, to be used solely for high school purposes showing how Scilab is used in French high schools for instance.

Appendix 1

The scripts are provided below, however links to ".txt "files are more useful and given first:

1) Banzhaf index calculator

https://docs.google.com/document/d/1oy_ozKjb5kXJFiU8uNDbrzeLqz75zuHIvDoyCRWQmLQ/edit?pli=1

2) Shapley-Shubik index calculator

https://docs.google.com/document/d/1GXcTj85mU-3Y72XEuSMchaBkkvxuW-4C1E1o5EeiBYY/edit?pli=1

3) Randomized Shapley-Shubik index calculator

https://docs.google.com/document/d/1i87FlxckbYVfOGrUsT_quE6S7lFxdFviWh4UkjC-3gI/edit?pli=1

4) Johnston index calculators

https://docs.google.com/document/d/1yU2tmWwHYkg-i029_7OrPfRaICOl8DD4GIn_Ggk91AY/edit?pli=1

Simply copy and paste the routines into the Scilab console (standalone version) or Scilab input dialog box (cloud version) after entering the appropriate input (voting weights and quota).Use the color-coded displayed routines as guides only—c opy and paste from the .txt versions. Comments are shown in red, Scilab code in blue and input fields in green. The user may change necessary inputs such as weights and quota, shown in green, before copying and pasting. These could be edited after pasting, in the appropriate browser dialog box using the cursor. In the case of the randomized Shapley-Shubik index calculator, one other input field, shown also in green, is the size of the sample taken from the n! possible sequential coalitions.

Note: *We recommend using the more stable .txt versions. The colorized versions which are primarily for guidance and illustrative purposes*.

```
// The Banzhaf power index routine  Start of the Banzhaf index calculator function
function  banzhaf_index = banzhaf_pir(weights,quota)
totalwinning =0;
for i = 1:length(weights)
tally(i)=0,
end
for i=1:(2^(length(weights))-1)
total(i)=0;
for k=1:length(weights)
total(i)=total(i)+bitget(i,k)*weights(k);
end
if (total(i)>=quota) then
totalwinning=totalwinning+1,
      for k=1:length(weights)
            if (((total(i)-bitget(i,k)*weights(k))
              tally(k)=tally(k)+bitget(i,k),
            end
      end
end
end
crit_sum=0;
for k=1:length(weights)
crit_sum=crit_sum+tally(k);
end
banzhaf_index=tally;
for k=1:length(weights)
banzhaf_index(k)=banzhaf_index(k)/crit_sum;
end
endfunction
//  End of the Banzhaf index calculator function
//************************************ Enter the weights and quota here ************************************
// Be sure the weights and quota are non-negative and that there are no more than 10 weights
weights = [4 2 1 ];
quota =5;
// ************************************ You have entered the weights and quota ************************************
banzhaf_voting_index="Null--Invalid input.Check that the quota and weights are non-negative and that there are no more than 10 candidates";
good_input=((quota>0)&(length(weights)<11 amp="" min="" weights="">0));
if (good_input) then  banzhaf_voting_index=banzhaf_pir(weights,quota);
end
```

```
banzhaf_voting_index
// The Shapley-Shubik power index routine Start of the Shapley-Shubik index calculator function
function shapley_index = shapley_pir(weights, quota)
for i= 1:length(weights)
totals(i)=0,
x(i)=i,
end
y=perms(x);
for j = 1:size(y,"r")
permutation=y(j,:);
i=1;
total=0;
decider(j) =0;
while (decider(j)==0)
total=total+weights(permutation(i)),
if (total>=quota) then
decider(j) = i,
else
i=i+1,
end
end
decider(j) = permutation(i);
end
for i = 1:length(decider)
totals(decider(i))=totals(decider(i))+1,
end
crit_sum=0;
for k=1:length(weights)
crit_sum=crit_sum+totals(k);
end
shapley_index=totals;
for k=1:length(weights)
shapley_index(k)=shapley_index(k)/crit_sum;
end
endfunction
//  End of the Shapley-Shubik index calculator function
// ************************************ Enter the weights and quota here ************************************
// Be sure the weights and quota are non-negative and that there are no more than 10 weights
weights = [4 2 1 ];
quota =5;
// ************************************You have entered the weights and quota ************************************
shapley_voting_index="Null--Invalid input.Check that the quota and weights are non-negative and that there are no more than 8 candidates";
good_input=((quota>0)&(length(weights)<9 amp="" min="" weights="">0));
if (good_input) then  shapley_voting_index=shapley_pir(weights,quota);
end
shapley_voting_index

// The randomized Shapley-Shubik power index routine Start of the randomized Shapley-Shubik index calculator function
function r_shapley_index = r_shapley_pir(weights, quota)
for i= 1:length(weights)
totals(i)=0,
x(i)=i,
end
//  ************************************Enter the sample size here ************************************
Sample_Size=100000;
//************************************
for j = 1:Sample_Size
permutation=grand(1,'prm',x);
i=1;
total=0;
decider(j) =0;
while (decider(j)==0)
total=total+weights(permutation(i)),
if (total>=quota) then
decider(j) = i,
else
i=i+1,
end
end
```

```
decider(j) = permutation(i);
end
for i = 1:length(decider)
totals(decider(i))=totals(decider(i))+1,
end
crit_sum=0;
for k=1:length(weights)
crit_sum=crit_sum+totals(k);
end
r_shapley_index=totals;
for k=1:length(weights)
r_shapley_index(k)=r_shapley_index(k)/crit_sum;
end
endfunction
//  End of the randomized Shapley-Shubik index calculator function
//  ************************************Enter the weights and quota here ************************************
// Be sure the weights and quota are non-negative and that there are between 9 and 20 weights
weights = [9 9 9 9 8 8 8 6 5 5 4 3  ];
quota =12;
// ************************************ You have entered the weights and quota ************************************
r_shapley_voting_index="Null--Invalid input.Check that the quota and weights are non-negative and that there are between 9 and 20";
good_input=((quota>0)&(length(weights)<21 amp="" length="" weights="">8)&(min(weights)>0));
if (good_input) then  r_shapley_voting_index=r_shapley_pir(weights, quota);
end
r_shapley_voting_index

// The Johnston power index routine Start of the Johnston index calculator function
function johnston_index = johnston_pir(weights,quota)
totalwinning =0;
temp=0;
for i = 1:length(weights)
tally(i)=0,
end
for i=1:(2^(length(weights))-1)
  total(i)=0;
  for k=1:length(weights)
      total(i)=total(i)+bitget(i,k)*weights(k);
 end
if (total(i)>=quota) then
totalwinning=totalwinning+1,
      for k=1:length(weights)
            if (((total(i)-bitget(i,k)*weights(k))
  temp=0;
               for l=1:length(weights)
             temp=temp+bitget(i,l),
               end
             tally(k)=tally(k)+1/temp,
            end
      end
  end
end
crit_sum=0;
for k=1:length(weights)
crit_sum=crit_sum+tally(k);
end
johnston_index=tally;
for k=1:length(weights)
johnston_index(k)=johnston_index(k)/crit_sum;
end
endfunction
//  End of the Johnston  index calculator function
// ************** Enter the weights and quota here. T weights and quota are non-negative and should be between 9 and 20 weights **********************
weights = [4 2 1 ];
quota =5;
// ************************************You have entered the weights and quota************************************
johnston_voting_index="Null--Invalid input.Check that the quota and weights are non-negative and that there are no more than 10 candidates";
good_input=((quota>0)&(length(weights)<11 amp="" min="" weights="">0));
if (good_input) then  johnston_voting_index=johnston_pir(weights,quota);
end
```

johnston_voting_index

Appendix 2

Screenshots of using Scilab in the cloud: Use the randomized Shapley-Shubik routine with 100,000 samples of the 12! coalitions of the weighted voting system [q; W]=[12; 9 9 9 9 8 8 8 6 5 5 4 3]:

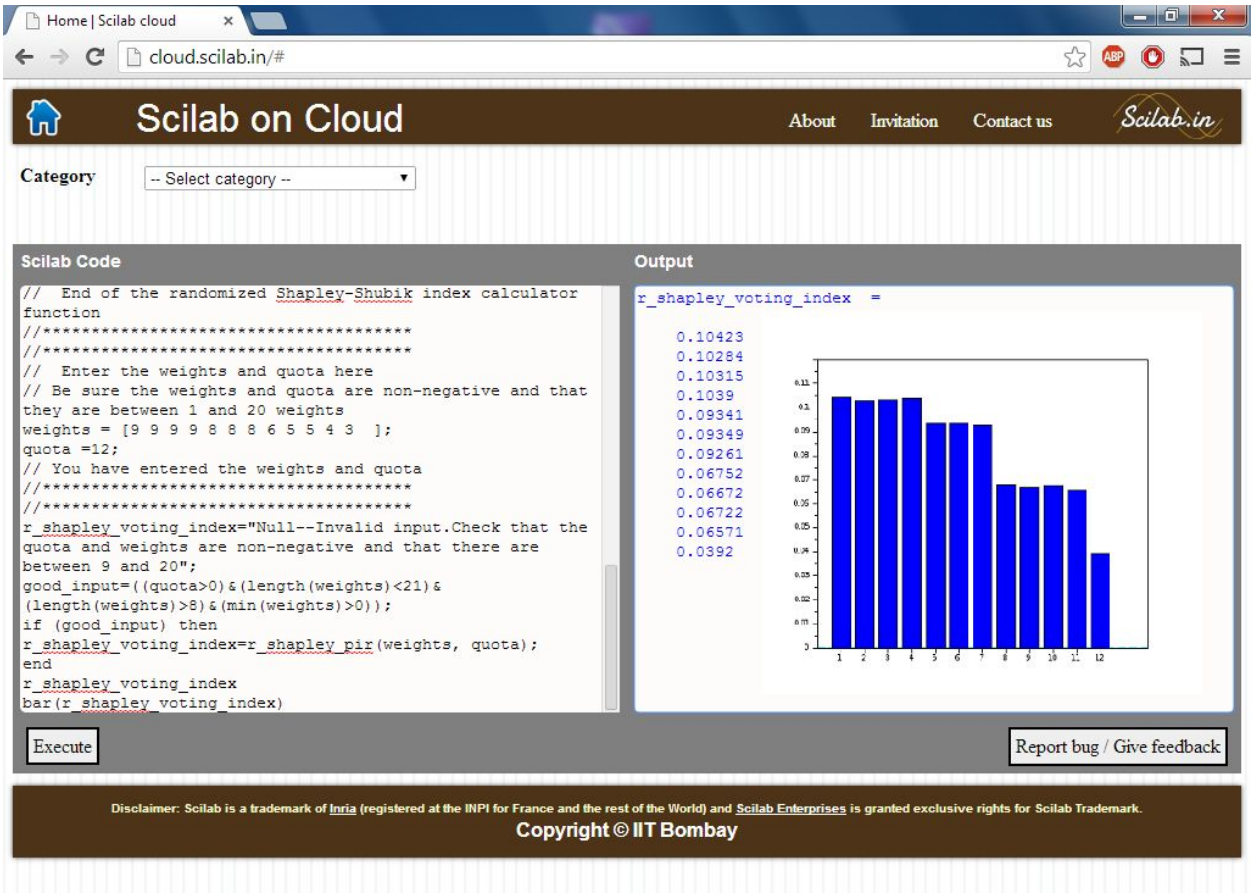

Figure 3: Using Scilab in the cloud to approximate the power indices for 12 candidates corresponding to the weighted voting system system [q; W]=[12; 9 9 9 9 8 8 8 6 5 5 4 3]. The randomized Shapley-Shubik routine was used.