symmetry

MDPI

Supplementary material

Supplementary material for "A test detecting the outliers for continuous distributions based on the cumulative distribution function of the data being tested"

Lorentz Jäntschi [1,2,*]

[1] Department of Physics and Chemistry, Technical University of Cluj-Napoca, 400641 Cluj, Romania; lorentz.jantschi@chem.utcluj.ro

[2] Chemical Doctoral School, Babeș-Bolyai University, 400084 Cluj-Napoca, Romania

* Correspondence: lorentz.jantschi@gmail.com; Tel.: +4-0264-401-775

Received: 12 June 2019; Accepted: 21 June 2019; Published: date

Summary: Software usage details for deriving the results given in figures and tables, algorithms and source codes are given in this supplementary material online available.

Introduction

FreePascal [1] were used to compile the executable for Monte-Carlo (MC) sampling (implements the peculiarities of MC simulation given in Table 2 in the paper). Data were generated by running the program (32bit executable). PHP v.7.3.1 cli [2] was used to assess numerically the agreement for "g1" statistic (Table 3), to calculate the statistics (eqs.6; used as given in Fig.3, Step 2 in Tables 5, 6, 7, 9, 10, and 11), and to implement (Fig.4) and run the second MC simulation study (Step 4 in Tables 5, 6, and 7). EasyFit [3] was used to obtain initial estimates (uses a hybrid CM-MLE method) for the population parameters. MS Excel [4] was used to do simple calculations, including FCS (eq.4). EasyFitXL [5] was used to calculate CDF of Gauss Laplace (Table 4), of χ^2 (eq.4; results of those calculations appears on Tables 5, 6, 7, 9, 10, and 11 in Step 2), and of Normal and Student (eq.17 and 19; Step 3 in Tables 5, 6, 7, 9, 10, and 11). Mathcad [6] was used to obtain MLE estimates (eq.3) for population parameters (Table 4). l.academicdirect.org/Statistics/tests [7] was used to calculate the associated probabilities with the statistics (eqs.6; used as given in Fig.3, Step 2 in Tables 5, 6, 7, 9, 10, and 11). Matlab [8] was used to obtain Figs. 1 and 2.

Algorithms and Source codes

*The algorithm for the first MC simulation study (referred in Table 2 in the paper; N ← 10; R ← 10; S ← 1000*100000)*

```
For n from 2 to N
  For resampling from 0 to R
    For sampling from 0 to S
      sample ← Generate n probabilities from uniform distribution
      g1[sampling] ← g1 statistic for the sample //eqs. 9-10 in the paper
    EndFor
    Sort the g1[0..S] array of statistics
    For grid_point from 1 to 999
      Extract g1g[grid_point][resampling] from g1[0..S]
    EndFor
```

```
  EndFor //collect 11 resamples of the g1 statistic in the given grid points
  For each grid_point from 1 to 999
    Sort the g1g[grid_point][0..R] array of statistics
    Extract g1g[grid_point][R/2] from g1g[grid_point][0..R]
    Output n, grid_point, g1g[grid_point][R/2] //output data of MC simulation
  EndFor //median (5 for 0..10) as statistic for order statistics
EndFor
```

The source code for the first MC simulation study software (referred in Table 2 in the paper; FreePascal language)

```
const
  s_resa=10; //'resa' in Table 2 in the paper (T2paper)
  s_leve=1000; //'p'+1 in T2paper
  s_bloc=100000; s_size=s_leve*s_bloc; s1size=s_size-1; //'m' in T2paper
  n_min=2; n_max=60; // 'n' in T2paper
type
  st_type=array[0..s1size]of extended;
  praguri=array[1..s_leve-1]of extended; //1..999
  probabi=array[0..n_max]of extended; //one sample of probabilities
  npragur=array[1..s_leve-1]of probabi; //grid for distribution of statistic
function Rnd2:Extended;var i:byte;r:Extended; begin r:=0.0; //binary random
  for i:=0 to 63 do r:=r/2.0+random(2);r:=r/2.0;if(r>1.0)then r:=1.0;Rnd2:=r;
end; //FreePascal random uses Mersenne twister
procedure QuickSortP(var A:probabi;Lo,Hi:LongInt);
  procedure SortP(l,r:LongInt);var i,j:LongInt;x,y:Extended; begin
    i:=l;j:=r;x:=a[(l+r)DIV 2];
    repeat while(a[i]<x)do i:=i+1; while(x<a[j])do j:=j-1;
      if(i<=j)then begin y:=a[i];a[i]:=a[j];a[j]:= y;i:=i+1;j:=j-1;end;
    until(i>j); if(l<j)then SortP(l,j);if(i<r)then SortP(i,r);
  end; begin SortP(Lo,Hi);
end; //sorts an array of probabilities
procedure QuickSortA(var A:st_type;Lo,Hi:LongInt);
  procedure SortP(l,r:LongInt);var i,j:LongInt;x,y:Extended; begin
    i:=l;j:=r;x:=a[(l+r)DIV 2];
    repeat while(a[i]<x)do i:=i+1; while(x<a[j])do j:=j-1;
      if(i<=j)then begin y:=a[i];a[i]:=a[j];a[j]:= y;i:=i+1;j:=j-1;end;
    until(i>j);if(l<j)then SortP(l,j);if(i<r)then SortP(i,r);
  end; begin SortP(Lo,Hi);
end; //sorts an array of statistics
function G1S(n:byte;var f:prob):Extended;var S,T:Extended;i:byte; begin
  S:=abs(f[n]-0.5);
  for i:=n-1 downto 1 do begin T:=abs(f[i]-0.5);if(T>S)then S:=T;end;
  G1s:=S;
```

```
end; //calculates the 'g1' statistic for a sample 'f' of size 'n'
procedure ST_Sample(n: LongInt; var A2:st_type; var A2_low, A2_upp: praguri);
  var i:LongInt;j,k:LongInt;p:probabi; begin
  for j:= s1size downto 0 do begin
    for i:= n-1 downto 0 do p[i]:=Rnd2; A2[j]:=G1S(n,p);
  end; QuickSortA(A2,0,s1size);
  for i:= s_leve-1 downto 1 do begin
    A2_low[i]:=A2[i*s_bloc-1]; A2_upp[i]:=A2[i*s_bloc];
  end;
end; //samples the statistic
procedure ST_Resample(n:LongInt; var A2:st_type; var A2lr,A2ur:praguri);
  var A2l_r,A2u_r:npragur;j,i:LongInt; begin
  for j:= 0 to s_resa do begin
    ST_Sample(n,A2,A2lr,A2ur);
    for i:= s_leve-1 downto 1 do begin
      A2l_r[i][j]:=A2lr[i];A2u_r[i][j]:=A2ur[i];
    end;
  end;
  for i:= s_leve-1 downto 1 do begin
    QuickSortP(A2l_r[i],0,s_resa);A2lr[i]:=A2l_r[i][s_resa div 2];
    QuickSortP(A2u_r[i],0,s_resa);A2ur[i]:=A2u_r[i][s_resa div 2];
  end;
end; //resamples the statistic
var
 b:st_type;
 b_low,b_upp:praguri;
 i,j:LongInt;
 f:text;s:string[2];
begin Randomize;
  for i:= n_min to n_max do begin
    str(i,s);
    ST_Resample(i,b,b_low,b_upp);
    assign(f,s+'_bin_rnd_'+'G1S.txt'); rewrite(f);
    for j:= 1 to s_leve-1 do writeln(f,j,chr(9),b_low[j],chr(9),b_upp[j]);
    close(f); writeln('resample for ',i,' completed.');
  end;
end. //b_low[j] and b_upp[j] have the same digits till a point (of accuracy)
```

Source code for calculation of the assessing of the agreement between observed and expected (implementing eq.12, as eq10 vs. eq.11 from the paper; PHP language)

```
for($n=10;$n>1;$n--){ $x=array();
  $a=explode("\r\n",file_get_contents($n."_bin_rnd_G1S.txt"));
  for($p=1;$p<1000;$p++){$b=explode("\t",$a[$p]);$x[$p]=$b[1];}
```

```
        $ss=0;
        for($p=999;$p>0;$p--) $ss+=pow($p/1000-pow(2*$x[$p],$n),2);
        $se=sqrt($s/999); echo($n."\t".$s."\t".$se."\r\n");
}
```

Source code for the second MC simulation study software (Fig.4; PHP language; it uses NormalDistribution.php [9])

```
<?php
   include("NormalDistribution.php");//implements normal distribution object
   define("K_runs",10000);
   define("S_size",10);//this is for Tables 6 and 7 calculations
   // define("S_size",206) for Table 11 calculations
   //below are data calculated outside and given here
   $po_gr=array(575.200, 8.702); //MC estimations for μ and σ
   $mm_gr=array(555.271, 595.129);//CI extreme val. grubbs
   $po_g0=array(575.200, 8.256); //MLE estimations for μ and σ
   $mm_g0=array(552.086, 598.314); //CI extreme val. g1
   //data from above is for Table 6 calculations given in the paper
   /*
   $po_gr=array(572.889, 5.011); //MC estimations for μ and σ
   $mm_gr=array(561.789, 583.989);//CI extreme val. grubbs
   $po_g0=array(572.889, 4.725); //MLE estimations for μ and σ
   $mm_g0=array(559.821, 585.957); //CI extreme val. g1
   //data from above is for Table 7 calculations given in the paper
   */
     /*
   $po_gr=array(6.481, 0.831); //MC estimations for μ and σ
   $mm_gr=array(3.492, 9.470); //CI extreme val. grubbs
   $po_g0=array(6.481, 0.829); //MLE estimations for μ and σ
   $mm_g0=array(3.444, 9.517); //CI extreme val. g1
   //data from above is for Table 11 calculations given in the paper
   */
   $st_gr=0;
   $st_g0=0;
   $my_norm = new NormalDistribution(0,1);
   for($i=0;$i<K_runs;$i++){//samples
      $s=array(); for($j=0;$j<S_size;$j++) $s[]=$my_norm->_getRNG();
      //_getRNG() uses mt_rand(); mt_rand() implements Mersenne twister
      for($j=0;$j<S_size;$j++){
         if($po_gr[0]+$s[$j]*$po_gr[1]<$mm_gr[0]){$st_gr++;break;}
         if($po_gr[0]+$s[$j]*$po_gr[1]>$mm_gr[1]){$st_gr++;break;}
      }//st_grubbs
      for($j=0;$j<S_size;$j++){
```

```
        if($po_g1[0]+$s[$j]*$po_g1[1]<$mm_g1[0]){$st_g1++;break;}
        if($po_g1[0]+$s[$j]*$po_g1[1]>$mm_g1[1]){$st_g1++;break;}
      }//st_g1
   }
   echo("outliers grubbs: ".$st_gr."\r\n");
   echo("outliers g1: ".$st_g1."\r\n");
   echo("expected number of outliers: ".(0.05*K_runs)."\r\n");
?>
```

It should be noted that the second MC simulation study software uses a mirror (or symmetrical) strategy for comparison of the "g1" and Grubbs methods - the same U(0,1) random drawings are used to feed the both methods (see $s[]=$my_norm->_getRNG(); in the code above).

Source code (MathCad language) for the MLE estimations for Gauss-Laplace distribution (eq.3; results given in Table 4 in the paper)

X := READPRN("d_206.txt")

$$c0(ka) := \sqrt{\frac{\Gamma\left(\frac{3}{ka}\right)}{\Gamma\left(\frac{1}{ka}\right)}} \quad c1(ka) := \frac{ka \cdot c0(ka)}{2 \cdot \Gamma\left(\frac{1}{ka}\right)}$$

$$MleGgl(x,md,si,ka) := -\ln(si) - \ln(c1(ka)) + \left(\left|c0(ka) \cdot \frac{x-md}{si}\right|\right)^{ka}$$

$$Mle(md,si,ka) := \sum_{i=0}^{rows(X)-1} MleGgl(X_i,md,si,ka)$$

md := 4806 si := 0.83017 ka := 1.4645

Given

$$\frac{d}{dmd}Mle(md,si,ka) = 0 \quad \frac{d}{dsi}Mle(md,si,ka) = 0 \quad \frac{d}{dka}Mle(md,si,ka) = 0$$

Y := Find(md,si,ka)

$$Y = \begin{pmatrix} 6.47938 \\ 0.82828 \\ 1.79106 \end{pmatrix}$$

Source code (MathCad language) for the MLE estimations for Normal distribution (eq.3; results given in Tables 6, 7, 9, 10 and 11 in the paper)

X := READPRN(Datafile)

$$MleNor(x,md,se) := -\ln\left(\sqrt{2 \cdot \pi}\right) - \ln(se) - \frac{1}{2} \cdot \left(\frac{x-md}{se}\right)^2$$

$$Mle(md,se) := \sum_{i=0}^{rows(X)-1} MleNor(X_i,md,se)$$

md := mean(X) se := Stdev(X)

Given

$$\frac{d}{dmd}Mle(md,se) = 0 \quad \frac{d}{dse}Mle(md,se) = 0$$

Y := Find(md,se)

$$Y = \begin{pmatrix} \quad \end{pmatrix}$$

Datafiles and the tables containing the results

Datafile	Table in the paper
"d_10.txt"	Table 6
"d_10_9.txt"	Table 7
"d_10_601.txt"	Table 9
"d_10_604.txt"	Table 10
"d_206.txt"	Table 11

Source code for the plots (to do Fig.1 and Fig.2; Matlab language)

<table>
<tr><th>Source code for Fig.1</th><th>Source code for Fig.2</th></tr>
<tr><td>XYZ = load('g1t_err_plot.txt');</td><td>XYZ = load('g1t_plot.txt');</td></tr>
<tr><td colspan="2">x = XYZ(:,1);
y = XYZ(:,2);
z = XYZ(:,3);
plot3(x,y,z,'.-')
tri = delaunay(x,y);
plot(x,y,'.')
h = trisurf(tri, x, y, z);
set(0,'defaulttextInterpreter','latex');</td></tr>
<tr><td>xlabel('n');
ylabel('p');
zlabel('CDF_{"g1"}(x;n) - p');</td><td>xlabel('x');
ylabel('n');
zlabel('CDF_{"g1"}(x;n)');</td></tr>
<tr><td colspan="2">view(-60,15);</td></tr>
</table>

References

1. Carl E. CODÈRE, Daniël MANTIONE, Florian KLÄMPFL, Jonas MAEBE, Michael Van CANNEYT, Peter VREMAN, Pierre MULLER, Marco van de VOORT, Leon de BOER, Armin DIEHL, Casey DUNCAN, Berczi GABOR, Sebastian GUENTHER, Tomas HAJNY, John LEE, Mark MAY, Mazen NEIFER, Olle RAAB, Thomas SCHATZL, Balazs SCHEIDLER, Nils SJOHOLM, MH SPIEGEL, Gernot TENCHIO, Erik WACHTMEESTER, Frank ZAGO, Gertjan SCHOUTEN, Karoly BALOGH, 1998. FreePascal: open source compiler for Pascal and Object Pascal (v.0.99.10); 2000: FreePascal v.1.0; 2005: FreePascal v.2.0; 2009: FreePascal v.2.2.4; 2011: FreePascal v.2.4.2; 2012: FreePascal v.2.6.0. 2017: FreePascal v.3.0.4.
2. Rasmus LERDORF, 1994. PHP/FI - initially from Personal Home Page tools (open source online since 1995; PHP v.1.0); Andi GUTMANS, Zeev SURASKI, 1997. PHP3 (open source online since 1998; PHP v.3.0); Andi GUTMANS, Zeev SURASKI, 1998. PHP4 (with 'Zend' engine; open source online since 1999; PHP v.4.0); Andi GUTMANS, Zeev SURASKI, 2004. PHP5 (with 'Zend' engine 2.0; open source online since 2004; PHP v.5.0). Dmitry STOGOV, Xinchen HUI, Nikita POPOV, 2014. PHP7 (with 'Zend' engine 3.0; open source online since 2015; PHP v.7.3.1).
3. MathWave Technologies, 2009. EasyFit Professional v.5.2 (© 2004-2009).
4. Microsoft, 1987. MS Excel v.2.05. Microsoft, 2013. MS Excel v.15.0.5127.
5. MathWave Technologies, 2009. EasyFitXL (MS Excel add-on), v.5.2.
6. Allen RAZDOW, 1986. MathCad v.0.3 (beta on 5 1/4 floppy, DOS version). PTC (Parametric Technology Corporation), 2007. Mathcad v.14.0.0.
7. Lorentz JÄNTSCHI, 2008. Statistics tests. Online: http://l.academicdirect.org/Statistics/tests
8. Jack LITTLE, Steve BANGERT, James Hardy WILKINSON, 1984. MATLAB v.1.0 (MatLab - from Matrix Laboratory); MathWorks Inc, 2015. MATLAB v.8.5.0 (version R2015a, 32bit, win32).
9. Jaco van KOOTEN, Paul MEAGHER, 2013. Version 1.3 of NormalDistribution class: an object for encapsulating normal distributions (PHP implementation).