Endnu mere om tilfældige tal

Denne artikel bygger oven på de to forrige artikler om tilfældige tal.

Den indeholder lidt matematik og anvendelse på PHP & ASP inspireret af diverse artikler og spørgsmål.

Skrevet den **14. Feb 2010** af **arne_v** I kategorien **Programmering / Generelt** | ★★★☆☆

Historie:
V1.0 - 15/04/2006 - original
V1.1 - 29/12/2008 - tilføj links
V1.2 - 12/02/2010 - smårettelser

Tidligere artikler

Man får størst udbytte hvis man har læst artiklerne:
* http://www.eksperten.dk/guide/680 "Tilfældige tal"
* http://www.eksperten.dk/guide/686 "Mere om tilfældige tal"

Hvad kan man forvente af genererede tilfældige tal ?

Man vil forvente at de er pænt fordelt men ikke perfekt fordelt.

Lad os antage at vi genererer 10000 tilfældige tal og grupperer i 4 grupper som bør være lige store (enten 4 grupper med 1/4 af range hver eller 4 grupper udfra de 2 low bit).

Den forventede fordeling af en god generator er:
2500
2500
2500
2500

Men det er faktisk ikke sandsynligt at få præcis den.

Sandsynligheden for det kan udregnes v.h.a. formlen for multinominal fordelingen:

P(2500,2500,2500,2500) =
10000!/(2500!*2500!*2500!*2500!)*0.25^2500*0.25^2500*0.25^2500*0.25^2500

hvilket ifølge min lommeregner er ca. 1*10^-6 d.v.s. at man vil kun få den perfekte fordeling en gange ud af en million gange eller sagt på en anden måde det er særdeles usandsynligt at få den perfekte fordeling.

Hvis man genererer merer end 10000 tal bliver sandsynligheden endnu mindre.

Hvis man kun kigger på fordelingen af 1 gruppe, så er den
binominal fordelt og for så store N kan vi approximere med en normalfordeling:

BIN(0.25,2500,10000) = N(0.25*10000,10000*0.25*(1-0.25) = N(2500,1875)

hvilket giver en standardafvigelse på:

SQRT(VAR) = 43.3

og dermed en fordeling som siger at der er 95% sandsynlighed for at
antallet er i intervallet:

E-1.96*s ... E+1.96*s = 2415 ... 2585

PHP

PHP kommer standard med 2 random generatorer: rand() og mt_rand().

rand() bruger den rand() funktion der er i C runtime library på systemet
(d.v.s. at om det er en god eller dårlig generator afhænger af styre systemet
og/eller C compileren på systemet !)

mt_rand() bruger den såkaldte Mersenne Twister algoritme som:
- er meget hurtig
- har en meget lang cycle (2^19937-1)
- rimeligt gode fordelings egenskaber

Nu vil læseren af de tidligere artikler vide, at så skal man bruge mt_rand()
fremfor rand().

PHP docs er dog også ret klar:

```
Many random number generators of older libcs have dubious or unknown
characteristics and are slow. By default, PHP uses the libc random
number generator with the rand()  function. The mt_rand() function
is a drop-in replacement for this. It uses a random number generator
with known characteristics using the  Mersenne Twister, which will
produce random numbers four times faster than what the average
libc rand() provides.
```

Omend det ikke er hastigheden men derimod usikkerheden omkring
egenskaberne der er den væsentligste grund til at undgå rand().

Lad os lave en test på Windows XP med PHP 5.0.3.

rand.php

```
<?php
srand(1233567);
$high1 = array(0,0,0,0);
```

```
$high2 = array(array(0,0,0,0),array(0,0,0,0),array(0,0,0,0),array(0,0,0,0));
$low1 = array(0,0,0,0);
$low2 = array(array(0,0,0,0),array(0,0,0,0),array(0,0,0,0),array(0,0,0,0));
$prev = rand();
for($i=0;$i<10000;$i++) {
    $curr = rand();
    $high1[$curr*4/getrandmax()]++;
    $high2[$curr*4/getrandmax()][$prev*4/getrandmax()]++;
    $low1[$curr%4]++;
    $low2[$curr%4][$prev%4]++;
    $prev = $curr;
}
echo "High bits:\n";
echo "<table border>\n";
for($row=0;$row<4;$row++) {
    echo "<tr>\n";
    echo "<td>" . $high1[$row] . "</td>\n";
    echo "</tr>\n";
}
echo "</table>\n";
echo "<table border>\n";
for($row=0;$row<4;$row++) {
    echo "<tr>\n";
    for($col=0;$col<4;$col++) {
        echo "<td>" . $high2[$row][$col] . "</td>\n";
    }
    echo "</tr>\n";
}
echo "</table>\n";
echo "Low bits:\n";
echo "<table border>\n";
for($row=0;$row<4;$row++) {
    echo "<tr>\n";
    echo "<td>" . $low1[$row] . "</td>\n";
    echo "</tr>\n";
}
echo "</table>\n";
echo "<table border>\n";
for($row=0;$row<4;$row++) {
    echo "<tr>\n";
    for($col=0;$col<4;$col++) {
        echo "<td>" . $low2[$row][$col] . "</td>\n";
    }
    echo "</tr>\n";
}
echo "</table>\n";
?>
```

resultat:

High bits:

```
2438
2480
2577
2504
607 605 612 614
605 608 645 622
627 641 659 650
599 625 662 617
Low bits:
2500
2500
2500
2500
0 0 0 2500
2500 0 0 0
0 2500 0 0
0 0 2500 0
```

Ikke godt i low bits !

mtrand.php

```
<?php
mt_srand(1233567);
$high1 = array(0,0,0,0);
$high2 = array(array(0,0,0,0),array(0,0,0,0),array(0,0,0,0),array(0,0,0,0));
$low1 = array(0,0,0,0);
$low2 = array(array(0,0,0,0),array(0,0,0,0),array(0,0,0,0),array(0,0,0,0));
$prev = mt_rand();
for($i=0;$i<10000;$i++) {
    $curr = mt_rand();
    $high1[$curr*4/mt_getrandmax()]++;
    $high2[$curr*4/mt_getrandmax()][$prev*4/mt_getrandmax()]++;
    $low1[$curr%4]++;
    $low2[$curr%4][$prev%4]++;
    $prev = $curr;
}
echo "High bits:\n";
echo "<table border>\n";
for($row=0;$row<4;$row++) {
    echo "<tr>\n";
    echo "<td>" . $high1[$row] . "</td>\n";
    echo "</tr>\n";
}
echo "</table>\n";
echo "<table border>\n";
for($row=0;$row<4;$row++) {
    echo "<tr>\n";
    for($col=0;$col<4;$col++) {
        echo "<td>" . $high2[$row][$col] . "</td>\n";
    }
```

```
    echo "</tr>\n";
}
echo "</table>\n";
echo "Low bits:\n";
echo "<table border>\n";
for($row=0;$row<4;$row++) {
    echo "<tr>\n";
    echo "<td>" . $low1[$row] . "</td>\n";
    echo "</tr>\n";
}
echo "</table>\n";
echo "<table border>\n";
for($row=0;$row<4;$row++) {
    echo "<tr>\n";
    for($col=0;$col<4;$col++) {
        echo "<td>" . $low2[$row][$col] . "</td>\n";
    }
    echo "</tr>\n";
}
echo "</table>\n";
?>
```

resultat:

```
High bits:
2467
2522
2524
2487
594 603 648 622
641 647 617 617
610 646 619 649
623 625 640 599
Low bits:
2503
2515
2480
2502
634 630 622 617
634 627 638 616
629 616 609 626
606 641 612 643
```

Meget bedre i low bits.

Konklusionen er klar: brug altid mt_rand() og aldrig rand() - specielt hvis du kan risikere at bruge low bits (modulus af potenser af 2 og lignende).

ASP

VBS RND() returnerer en floating point værdi 0.0 <= x < 1.0 og er dermed anderledes end PHP rand() der returnerer en integer værdi.

Ofte vil man imidlertid have behov for at omregne til en integer.

Og gør man det skal man vide at VBS RND() baserer sig på en 24 bit LCG.

Det betyder at:
1) man kan ikke skalerere den højere end 24 bit
2) hvis man skalerer højt kan low bits være dårligt fordelt

Lad os prøve først med 12 bit.

random12.asp

```
<%
Randomize(1234567)
high1 = array(0,0,0,0)
high2 = array(array(0,0,0,0),array(0,0,0,0),array(0,0,0,0),array(0,0,0,0))
low1 = array(0,0,0,0)
low2 = array(array(0,0,0,0),array(0,0,0,0),array(0,0,0,0),array(0,0,0,0))
prev = Fix(Rnd() * 4096)
For i = 1 To 10000
   curr = Fix(Rnd() * 4096)
   high1(curr*4\4096) = high1(curr*4\4096) + 1
   high2(curr*4\4096)(prev*4\4096) = high2(curr*4\4096)(prev*4\4096) + 1
   low1(curr Mod 4) = low1(curr Mod 4) + 1
   low2(curr Mod 4)(prev Mod 4) = low2(curr Mod 4)(prev Mod 4) + 1
   prev = curr
Next
Response.Write "High bits:" & vbCrLf
Response.Write "<table border>" & vbCrLf
For row = 0 To 3
    Response.Write "<tr>" & vbCrLf
    Response.Write "<td>" & high1(row) & "</td>" & vbCrLf
    Response.Write "</tr>" & vbCrLf
Next
Response.Write "</table>" & vbCrLf
Response.Write "<table border>" & vbCrLf
For row = 0 To 3
    Response.Write "<tr>" & vbCrLf
    For col = 0 To 3
        Response.Write "<td>" & high2(row)(col) & "</td>" & vbCrLf
    Next
    Response.Write "</tr>" & vbCrLf
Next
Response.Write "</table>" & vbCrLf
Response.Write "Low bits:" & vbCrLf
Response.Write "<table border>" & vbCrLf
For row = 0 To 3
    Response.Write "<tr>" & vbCrLf
    Response.Write "<td>" & low1(row) & "</td>" & vbCrLf
```

```
    Response.Write "</tr>" & vbCrLf
Next
Response.Write "</table>" & vbCrLf
Response.Write "<table border>" & vbCrLf
For row = 0 To 3
    Response.Write "<tr>" & vbCrLf
    For col = 0 To 3
        Response.Write "<td>" & low2(row)(col) & "</td>" & vbCrLf
    Next
    Response.Write "</tr>" & vbCrLf
Next
Response.Write "</table>" & vbCrLf
%>
```

resultat:

```
High bits:
2528
2474
2509
2489
631 629 633 635
619 617 630 608
643 593 633 640
635 635 613 606
Low bits:
2525
2512
2450
2513
648 628 616 633
641 621 620 630
613 621 600 616
623 641 614 635
```

Hvilket jo ser nydeligt ud.

Så prøver vi med 16 bit.

random16.asp

```
<%
Randomize(1234567)
high1 = array(0,0,0,0)
high2 = array(array(0,0,0,0),array(0,0,0,0),array(0,0,0,0),array(0,0,0,0))
low1 = array(0,0,0,0)
low2 = array(array(0,0,0,0),array(0,0,0,0),array(0,0,0,0),array(0,0,0,0))
prev = Fix(Rnd() * 65536)
```

```
For i = 1 To 10000
   curr = Fix(Rnd() * 65536)
   high1(curr*4\65536) = high1(curr*4\65536) + 1
   high2(curr*4\65536)(prev*4\65536) = high2(curr*4\65536)(prev*4\65536) + 1
   low1(curr Mod 4) = low1(curr Mod 4) + 1
   low2(curr Mod 4)(prev Mod 4) = low2(curr Mod 4)(prev Mod 4) + 1
   prev = curr
Next
Response.Write "High bits:" & vbCrLf
Response.Write "<table border>" & vbCrLf
For row = 0 To 3
    Response.Write "<tr>" & vbCrLf
    Response.Write "<td>" & high1(row) & "</td>" & vbCrLf
    Response.Write "</tr>" & vbCrLf
Next
Response.Write "</table>" & vbCrLf
Response.Write "<table border>" & vbCrLf
For row = 0 To 3
    Response.Write "<tr>" & vbCrLf
    For col = 0 To 3
        Response.Write "<td>" & high2(row)(col) & "</td>" & vbCrLf
    Next
    Response.Write "</tr>" & vbCrLf
Next
Response.Write "</table>" & vbCrLf
Response.Write "Low bits:" & vbCrLf
Response.Write "<table border>" & vbCrLf
For row = 0 To 3
    Response.Write "<tr>" & vbCrLf
    Response.Write "<td>" & low1(row) & "</td>" & vbCrLf
    Response.Write "</tr>" & vbCrLf
Next
Response.Write "</table>" & vbCrLf
Response.Write "<table border>" & vbCrLf
For row = 0 To 3
    Response.Write "<tr>" & vbCrLf
    For col = 0 To 3
        Response.Write "<td>" & low2(row)(col) & "</td>" & vbCrLf
    Next
    Response.Write "</tr>" & vbCrLf
Next
Response.Write "</table>" & vbCrLf
%>
```

resultat:

```
High bits:
2528
2474
2509
2489
```

```
631 629 633 635
619 617 630 608
643 593 633 640
635 635 613 606
Low bits:
2509
2485
2506
2500
837 194 644 834
828 822 194 641
649 826 834 197
195 643 834 828
```

Og allerede her ser vi nogle afvigelser i low bits.

Vi prøver nu de 24 bit om er teoretisk maksimum.

random24.asp

```
<%
Randomize(1234567)
high1 = array(0,0,0,0)
high2 = array(array(0,0,0,0),array(0,0,0,0),array(0,0,0,0),array(0,0,0,0))
low1 = array(0,0,0,0)
low2 = array(array(0,0,0,0),array(0,0,0,0),array(0,0,0,0),array(0,0,0,0))
prev = Fix(Rnd() * 16777216)
For i = 1 To 10000
   curr = Fix(Rnd() * 16777216)
   high1(curr*4\16777216) = high1(curr*4\16777216) + 1
   high2(curr*4\16777216)(prev*4\16777216) =
high2(curr*4\16777216)(prev*4\16777216) + 1
   low1(curr Mod 4) = low1(curr Mod 4) + 1
   low2(curr Mod 4)(prev Mod 4) = low2(curr Mod 4)(prev Mod 4) + 1
   prev = curr
Next
Response.Write "High bits:" & vbCrLf
Response.Write "<table border>" & vbCrLf
For row = 0 To 3
    Response.Write "<tr>" & vbCrLf
    Response.Write "<td>" & high1(row) & "</td>" & vbCrLf
    Response.Write "</tr>" & vbCrLf
Next
Response.Write "</table>" & vbCrLf
Response.Write "<table border>" & vbCrLf
For row = 0 To 3
    Response.Write "<tr>" & vbCrLf
    For col = 0 To 3
        Response.Write "<td>" & high2(row)(col) & "</td>" & vbCrLf
    Next
    Response.Write "</tr>" & vbCrLf
```

```
Next
Response.Write "</table>" & vbCrLf
Response.Write "Low bits:" & vbCrLf
Response.Write "<table border>" & vbCrLf
For row = 0 To 3
    Response.Write "<tr>" & vbCrLf
    Response.Write "<td>" & low1(row) & "</td>" & vbCrLf
    Response.Write "</tr>" & vbCrLf
Next
Response.Write "</table>" & vbCrLf
Response.Write "<table border>" & vbCrLf
For row = 0 To 3
    Response.Write "<tr>" & vbCrLf
    For col = 0 To 3
        Response.Write "<td>" & low2(row)(col) & "</td>" & vbCrLf
    Next
    Response.Write "</tr>" & vbCrLf
Next
Response.Write "</table>" & vbCrLf
%>
```

resultat:

```
High bits:
2528
2474
2509
2489
631 629 633 635
619 617 630 608
643 593 633 640
635 635 613 606
Low bits:
2500
2500
2500
2500
0 2500 0 0
0 0 2500 0
0 0 0 2500
2500 0 0 0
```

Et typisk dårligt low bit resultat for LCG.

Konklusionen er at VBS RND() er acceptabel til at skalere til integer værdier 1-12 bit men at skalere til integer værdier med flere bits kræver stor varsomhed p.g.a. dårlig fordeling af low bits. Og man må naturligvis aldrig skalere til integer værdier med flere bits end 24.

ASP.NET og JSP

ASP.NET og JSP bruger .NET og Java random generatorer som er beskrevet i de første 2 artikler - og som har rimeligt gode egenskaber.

Kommentar af ismand d. 25. Apr 2006 | 1

Fortrinlig teoretisk gennemgang. Jeg kunne godt tænke mig at se .NET resultaterne.

Kommentar af tommyb d. 27. Jan 2009 | 2

Viser kun fordelingen ved et linært udtræk af tilfældige tal, havde håbet på en gennemgang af dets påvirkning ved fx. kryptering, altså ikke hvordan talne fordeler sig, men im det kan forudsiges.

Ellers ganske godt stykke arbejde.

Kommentar af 7degreez d. 23. Apr 2006 | 3

Kommentar af tuxic d. 28. Apr 2006 | 4

Kommentar af henrik6666 d. 16. Apr 2006 | 5

Kommentar af datasource d. 23. Apr 2006 | 6

SUPER

Kommentar af phpzer0 d. 25. Apr 2006 | 7

Kommentar af tha_painter d. 12. May 2007 | 8