Random number generators

How can deterministic algorithms give random numbers?

> pseudo-random numbers generators

Linear congruential generators; recurrence relation

\[ x_{n+1} = \text{mod}(a \cdot x_n + c, m) \]

can generate all numbers 0,...,m-1 in seemingly random order (for suitable a, m, c odd). Test with small m=2^k, c=1:

<table>
<thead>
<tr>
<th>m=4</th>
<th>a</th>
<th>sequence:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1 2 3 0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1 3 3 3</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1 0 1 0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1 1 1 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>m=8</th>
<th>a</th>
<th>sequence:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1 2 3 4 5 6 7 0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1 3 7 7 7 7 7 7</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1 4 5 0 1 4 5 0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1 5 5 5 5 5 5 5</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1 6 7 4 5 2 3 0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1 7 3 3 3 3 3 3</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1 0 1 0 1 0 1 0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>1 1 1 1 1 1 1 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>m=16</th>
<th>a</th>
<th>sequence:</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0</td>
<td>1 4 13 8 9 12 5 0 1 4 13 8 9 12 5 0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1 6 15 12 13 2 11 8 9 14 7 4 5 10 3 0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>1 12 5 8 9 4 13 0 1 12 5 8 9 4 13 0</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>1 14 7 12 13 10 3 8 9 6 15 4 5 2 11 0</td>
</tr>
</tbody>
</table>
On the computer, integer overflow is a modified modulus $2^{32}$ (or $2^{64}$) operation; can be used for random numbers:

\[
\begin{align*}
    n & = 69069 \times n + 1013904243 \\
    \text{ran} & = 0.5d0 + n \times 0.23283064d - 9
\end{align*}
\]

Many systems use this type of intrinsic random number generator:

▷ don’t use in serious work (period too short, not random enough)
▷ 64-bit integer version is quite a good generator

This one is recommended:

\[
\begin{align*}
    n & = 286293355577941757 \times n + 1013904243 \\
    \text{ran} & = 0.5d0 + \text{dble}(n) \times \text{dmul}
\end{align*}
\]

Where \(\text{dmul}\) is precalculated as

\[
\text{dmul} = 1.d0 / \text{dble}(2 \times (2_8**62 - 1) + 1)
\]
Addition, subtraction can also be used, e.g.,

\[ x_{n+1} = \text{mod}(x_{n-3} - x_{n-1}, m) \quad m = 2^k - \text{prime} \]

Mixed generators; longer periods, more random, e.g.,

```plaintext
mzran=iir-kkr
if (mzran < 0) mzran=mzran+2147483579
iir=jjr; jjr=kkr; kkr=mzran
nnr=69069*nnr+1013904243
mzran=mzran+nnr
rand=0.5d0+mzran*0.23283064d-9
```

Four seeds; iir, jjr, kkr, nnr. Period > \(10^{28}\)
More Fortran: Keyword and optional arguments
If the interface of the procedure is explicit (e.g., in a module)
• one does not have to use all arguments in a procedure call
  - omissions at the end of the argument list ok
  - any omission ok if keyword (dummy variable name) is used
• one can use any order of the arguments if keywords are used

Example: keyword.f90

```fortran
module test
  contains
    subroutine keywordsub(a,b)
      integer, optional :: a
      integer, optional :: b
      if (present(a)) write(*,*)'a = ',a
      if (present(b)) write(*,*)'b = ',b
    end subroutine keywordsub
  end contains
end module test
```
program testkeyword
use test
integer :: arg1,arg2

read(*,*)arg1,arg2

write(*,*)
call keywordsub(arg1)
write(*,*)
call keywordsub(b=arg2)
write(*,*)
call keywordsub(a=arg1,b=arg2)

end program testkeyword

If arg1=1 and arg2=2 are read in, this is the output:
  a=1

  b=2

  a=1
  b=2
Fortran 90 intrinsic random number generator

random_number(r) initialized with random_seed()

integer :: i, size
integer, allocatable :: seed(:)
real :: r

call random_seed(size)
allocate (seed(size))
write(*,*)'give ',size,' random seeds '
read(*,*)seed

call random_seed(put=seed)
do i=1,10
   call random_number(r)
   write(*,*)r
end do

call random_seed(get=seed)
write(*,*)seed