Corrigendum to the paper *"Drawing random floating-point numbers from an interval"*,

ACM TOMACS, 32:3, pp. 1–24

Frédéric Goualard

November 8, 2023

In Table IV of Section 5, the four functions γ`sectionCC()`, γ`sectionCO()`, γ`sectionOC()`, and γ`sectionOO()` present the same two flaws:

- If `ceilint(a,b,g)` returns an integer `hi` that is strictly greater than 2^p —with p the size of the significand of a and b— the random value `k` from the interval $[0, \mathtt{hi}]$ or $[0, \mathtt{hi}\text{-}1]$ may itself be greater than 2^p. Therefore, it may not be representable without rounding as a floating-point number when performing the multiplications `(k-1)*g`, `k*g` or `(k+1)*g`. The consequence is that some floating-point numbers in $[a, b]$ cannot be drawn even though they should;
- For some rare intervals $[a, b]$ where both bounds are very large in magnitude with opposite signs, the values `(k-1)*g`, `k*g`, or `(k+1)*g` may overflow.

Fortunately, there is a very simple fix to both problems: split `k` into two positive integers `k1` and `k2` such that:

$$\mathtt{k} = 2^v \times \mathtt{k1} + \mathtt{k2}$$

and compute, e.g., `b-k*g` as `2^v*(b*2^-v-k1*g)-k2*g`.

For the double precision format, we may choose $v = 2$: since $g \in [2^{-1074}, 2^{971}]$ and it can easily be proven that `hi=ceilint(a,b,g)` is always strictly smaller than 2^{55}, we have $\mathtt{k1} < 2^{53}$ and $\mathtt{k2} < 2^{53}$, and therefore $\mathtt{k*g} < 2^{1024}$, which precludes any overflow.

The following Julia code is a corrected version of the one in Table IV of the original article. It also presents an implementation of the function `splitint64()` to split a positive integer into two parts.

```
"""
Given a 64 bits positive integer,
return two values `vhi` and `vlo`
such that:

    v = 4*vhi + vlo
"""
function splitint64(v)
    vhi = Float64(v>>2)
    vlo = Float64(v & 0x3)
    return (vhi,vlo)
end

"""
    γsectionCC(a,b)

Draw a float from an interval [a,b]
uniformly at random.
"""
function γsectionCC(a,b)
   g = γ(a,b)
   hi = ceilint(a,b,g)
   k = rand(DiscreteUniform(0,hi))
   (k1,k2) = splitint64(k)
   if abs(a) <= abs(b)
      return (k == hi)
         ? a
         : 4*(b/4-k1*g)-k2*g
   else
      return (k == hi)
         ? b
         : 4*(a/4+k1*g)+k2*g
   end
end

"""
    γsectionCO(a,b)

Draw a float from an interval [a,b)
uniformly at random.
"""
function γsectionCO(a,b)
   g = γ(a,b)
   hi = ceilint(a,b,g)
   k = rand(DiscreteUniform(1,hi))
   (k1,k2) = splitint64(k)
   if abs(a) <= abs(b)
      return (k == hi)
         ? a
         : 4*(b/4-k1*g)-k2*g
   else
      return 4*(a/4+k1*g)+(k2-1)*g
   end
end

"""
    γsectionOC(a,b)

Draw a float from an interval (a,b]
uniformly at random.
"""
function γsectionOC(a,b)
   g = γ(a,b)
   hi = ceilint(a,b,g)
   k = rand(DiscreteUniform(0,hi-1))
   (k1,k2) = splitint64(k)
   if abs(a) <= abs(b)
       return 4*(b/4-k1*g)-k2*g
   else
       return (k == hi-1)
          ? b
          : 4*(a/4+k1*g)+(k2+1)*g
   end
end

"""
    γsectionOO(a,b)

Draw a float from an interval (a,b)
uniformly at random.
"""
function γsectionOO(a,b)
   g = γ(a,b)
   hi = ceilint(a,b,g)
   k = rand(DiscreteUniform(1, hi-1))
   (k1,k2) = splitint64(k)
   if abs(a) <= abs(b)
       return 4*(b/4-k1*g)-k2*g
   else
       return 4*(a/4+k1*g)+k2*g
   end
end
```