```lua
local _, the, COL = require"lib", require"the", require"col"
local class,merge,per,push,sort,upx = _.class,_.merge,_.per,_.push,_.sort,_.upx
local sd = _.sd
local norm,oo = _.norm,_.oo

local NUM = class("NUM",COL)
function NUM:new(at,name)
  self:super(at,name)
  self.has, self.ok = {}, false
  self.lo, self.hi  = math.huge, -math.huge end

local r=math.random
function NUM:add1(x,inc,        pos)
  for i=1,inc do
    self.lo = math.min(x, self.lo)
    self.hi = math.max(x, self.hi)
    if #self.has < the.some        then pos = 1 + #self.has
    elseif r()   < the.some/self.n then pos = 1 + ((r()*#self.has)//1) end
    if pos then
      self.ok = false
      self.has[pos] = x end end end

function NUM:div(   a) a=self:all(); return (per(a,.9) - per(a,.1))/2.56 end
function NUM:mid()     return per(self:all(), .5)  end
function NUM:same(x,y) return math.abs(x - y) <= the.cohen * self:div() end

function NUM:dist1(x,y)
  if     x=="?" then y = norm(self.lo, self.hi, y); x=y<.5 and 1 or 0
  elseif y=="?" then x = norm(self.lo, self.hi, x); y=x<.5 and 1 or 0
  else              x,y = norm(self.lo, self.hi, x), norm(self.lo, self.hi,y) end
  return math.abs(x-y) end

function NUM:like1(i,x)
  local sd= self:div()
  if x < self.mu - 4*sd then return 0 end
  if x > self.mu + 4*sd then return 0 end
  local denom = (math.pi*2*sd^2)^.5
  local nom   =  math.exp(1)^(-(x-self.mu)^2/(2*sd^2+1E-32))
  return nom/(denom + 1E-32) end

function NUM:merge(other,   out)
  out = NUM(self.at, self.name)
  for _,x in self(self.has) do out:add(x) end
  for _,x in self(other.has) do out:add(x) end
  return out end

function NUM:all()
  if not self.ok then table.sort(self.has) end
  self.ok=true
  return self.has end

function NUM:bins(other, BIN)
  local tmp,out = {},{}
  for _,x in pairs(self.has ) do push(tmp, {x=x, y="left"}) end
  for _,x in pairs(other.has) do push(tmp, {x=x, y="right"}) end
  tmp = sort(tmp,upx) -- ascending on x
  local now     = push(out, BIN(self.at, self.name, tmp[1].x))
  local epsilon = sd(tmp,function(z) return z.x end) * the.cohen
  local minSize = (#tmp)^the.leaves
  for j,xy in pairs(tmp) do
    if j > minSize and j + minSize < #tmp then -- leave enough for other bins
      if now.ys.n > minSize then               -- enough in this bins
        if xy.x ~= tmp[j+1].x then             -- there is a break in the data
          if now.hi - now.lo > epsilon then    -- "now" not trivially small
            now = push(out,  BIN(self.at, self.name, now.hi)) end end end end
    now:add(xy.x, xy.y) end
  out[1].lo    = -math.huge
  out[#out].hi =  math.huge
  return merge(out, BIN.mergeSameDivs) end

return NUM
```