```lua
local help = [[

BORE: best or rest multi-objective optimization.
(c)2022 Tim Menzies, timm@ieee.org, opensource.org/licenses/Fair
"I think the highest and lowest points are the important ones.
 Anything else is just...in between." Jim Morrison

USAGE:
  alias bore="lua bore.lua "
  bore [OPTIONS]

OPTIONS:
  -b  --bins  max bins                 = 16
  -S  --some  number of nums to keep   = 256

OPTIONS (other):
  -s  --seed  random number seed       = 10019
  -f  --file  where to find data       = ../etc/data/auto93.csv
  -d  --dump  dump stack+exit on error = false
  -h  --help  show help                = false
  -g  --go    start up action          = nothing
]]

local function thing(x)
  x = x:match"^%s*(.-)%s*$"
  if x=="true" then return true elseif x=="false" then return false end
  return math.tointeger(x) or tonumber(x) or x end

local the={}
help:gsub("\n ([-][^%s]+)[%s]+([-][-]([^%s]+))[^\n]*%s([^%s]+)",function(f1,f2,k,x)
  for n,flag in ipairs(arg) do if flag==f1 or flag==f2 then
    x = x=="false" and"true" or x=="true" and"false" or arg[n+1] end end
  the[k] = thing(x) end)
--------------------------------------------------------------------------------
local as,atom,csv,has,map,merge,o,oo,obj,ok,patch,per,push,rows,slice,sort
local _,GO,RANGE,SOME,NUM,SYM,COLS,ROW,EGS
local R,big,fmt

big = math.huge
R   = math.random
fmt = string.format

function push(t,x)    t[1+#t]=x;       return x end
function sort(t,f)    table.sort(t,f); return t end
function map(t,f, u)  u={}; for k,v in pairs(t) do u[1+#u]=f(v) end;return u end
function slice(t,i,j,   u)
  u={}; for k=(i or 1), (j or #t) do u[1+#u] = t[k] end return u end

function has(i, defaults, also)
  for k,v in pairs(defaults) do i[k] = v end
  for k,v in pairs(also or {}) do assert(i[k]~=nil,"unknown:"..k);i[k]=v end end

function csv(src)
  src = io.input(src)
  return function(line, row)
    line=io.read()
    if not line then io.close(src) else
      row={}; for x in line:gmatch("([^,]+)") do row[1+#row]=thing(x) end
      return row end end end

function merge(b4,        a,b,c,j,n,tmp,fillInTheGaps)
  function expand(t)
    for j=2,#t do  t[j].lo = t[j-1].hi end
    t[1].lo,  t[#t].hi  = -big, big
    return t
  end ------------------
  j, n, tmp = 1, #b4, {}
  while j<=n do
    a, b = b4[j], b4[j+1]
    if b then
      c = a:merged(b)
      if c then
        a, j = c, j+1 end end
    tmp[#tmp+1] = a
    j = j+1 end
  return #tmp==#b4 and expand(tmp) or merge(tmp) end

function oo(t) print(o(t)) end
function o(t,    u)
  if #t>0 then return "{"..table.concat(map(t,tostring)," ").."}" else
    u={}; for k,v in pairs(t) do u[1+#u] = fmt(":%s %s",k,v) end
    return (t.is or "").."{"..table.concat(sort(u)," ").."}" end end

function obj(name,    t,new)
  function new(kl,...)
    local x=setmetatable({},kl); kl.new(x,...); return x end
  t = {__tostring=o, is=name or ""}; t.__index=t
  _ = t
  return setmetatable(t, {__call=new}) end
--------------------------------------------------------------------------------
RANGE=obj"RANGE"
function _.new(i,t)  has(i,{at=0, txt="", lo=big, hi= -big, ys=SYM()},t) end
function _.of(i,x)   return i.ys.all[x] or 0 end
function _.__lt(i,j) return i.lo < j.lo end
function _.add(i,x,y)
  if x=="?" then return x end
  if x>i.hi then i.hi=x end
  if x<i.lo then i.lo=x end
  i.ys:add(y) end

function _.select(i,t,     x)
  t = t.cells and t.cells or t
  x = t[i.pos]
  return x=="?" or i.lo == i.hi and i.lo == x or i.lo <= x and x < i.hi end

function _.__tostring(i)
  local x, lo, hi = i.txt, i.lo, i.hi
  if     lo ==  hi  then return fmt("%s == %s",x, lo)
  elseif hi ==  big then return fmt("%s >= %s",x, lo)
  elseif lo == -big then return fmt("%s < %s", x, hi)
  else                   return fmt("%s <= %s < %s",lo,x,hi) end end

function _.merged(i,j,    k)
  if i.at == j.at then
    k = i.ys:merged(j.ys)
    if k then
      return RANGE{at=i.at, txt=i.txt, lo=i.lo, hi=j.hi, ys=k} end end end
--------------------------------------------------------------------------------
SOME=obj"SOME"
function _.new(i) i.all, i.ok, i.n = {}, false,0 end

function _.add(i,x,     a)
  i.n, a = 1 + i.n, i.all
  if     #a  < the.some     then i.ok=false; push(a,x)
  elseif R() < the.some/i.n then i.ok=false; a[R(#a)]=x end end

function _.nums(i) i.all=i.ok and i.all or sort(i.all);i.ok=true;return i.all end
function _.per(i,p,   a)
  p,a=(p or .5),i:nums(); return a[math.max(1,math.min(#a, p*#a//1))] end
--------------------------------------------------------------------------------
SYM=obj"SYM"
function _.new(i,t)    has(i,{at=0, txt="", all={}},t) end
function _.add(i,x,n)  if x~="?" then i.all[x]=(n or 1)+(i.all[x] or 0) end end

function _.mid(i,   m,x)
  m=0; for y,n in pairs(i.all) do if n>m then m,x=n,y end end; return x end

function _.div(i,   n,e)
  n=0; for k,m in pairs(i.all) do n = n + m end
  e=0; for k,m in pairs(i.all) do e = e - m/n*math.log(m/n,2) end
  return e,n end

function _.merged(i,j,    k,div1,n1,div2,n2,n)
  k = SYM{at=i.at, txt=i.txt}
  for x,n in pairs(i.all) do k:add(x,n) end
  for x,n in pairs(j.all) do k:add(x,n) end
  div1, n1 = i:div()
  div2, n2 = j:div()
  n        = n1+n2
  if k:div() < (div1*n1/n + div2*n2/n) then return k end end

function _.range(i,x,y,ranges)
  if x=="?" then return x end
  ranges[x] = ranges[x] or RANGE{at=i.at, txt=i.txt, lo=x, hi=x}
  ranges[x]:add(x,y) end
--------------------------------------------------------------------------------
NUM=obj"NUM"
function _.new(i,t)
  has(i,{at=0,txt="",lo= big,hi= -big, all=SOME()},t)
  i.w = i.txt:find"-$" and -1 or 1 end

function _.mid(i)    return  i.all:per(.5) end
function _.div(i)    return (i.all:per(.9) - i.all:per(.1)) / 2.56 end
function _.norm(i,x) return x=="?" and x or (x-i.lo)/(i.hi - i.lo) end

function _.add(i,x)
  if x=="?" then return x end
  if x>i.hi then i.hi=x end
  if x<i.lo then i.lo=x end
  i.all:add(x) end

function _.range(i,x,y,ranges,   gap,r)
  if x=="?" then return x end
  gap = (i.hi - i.lo)/the.bins
  r   = (x - i.lo)//gap * gap
  ranges[r] = ranges[r] or RANGE{at=i.at, txt=i.txt}
  ranges[r]:add(x,y) end
--------------------------------------------------------------------------------
ROW=obj"ROW"
function _.new(i,t) has(i,{cells={},data={}},t) end

function _.__lt(i,j,     s1,s2,e,y,a,b)
  y = i.data.cols.y
  s1, s2, e = 0, 0,  math.exp(1)
  for _,col in pairs(y) do
     a = col:norm(i.cells[col.at])
     b = col:norm(j.cells[col.at])
     s1= s1 - e^(col.w * (a - b) / #y)
     s2= s2 - e^(col.w * (b - a) / #y) end
  return s1/#y < s2/#y  end
--------------------------------------------------------------------------------
COLS=obj"COLS"
function _.new(i,t,     col)
  has(i, {all={}, x={}, y={}, names={}},t)
  for at,txt in pairs(i.names) do
    col = push(i.all, (txt:find"^[A-Z]" and NUM or SYM){at=at, txt=txt})
    if not txt:find":$" then
      push(txt:find"[-+!]$" and i.y or i.x, col) end end end
--------------------------------------------------------------------------------
EGS=obj"EGS"
function _.new(i)       i.rows,i.cols= {},nil end
function _.file(i,file) for row in csv(file) do  i:add(row) end; return i end
function _.add(i,row)
  if   i.cols
  then row = push(i.rows, row.cells and row or ROW{data=i, cells=row}).cells
       for k,col in pairs(i.cols.all) do col:add(row[col.at]) end
  else i.cols = COLS{names=row} end
  return i end

function _.mid(i,cs) return map(cs or i.cols.y,function(c)return c:mid() end)end
function _.div(i,cs) return map(cs or i.cols.y,function(c)return c:div() end)end

function _.copy(i,rows,   out)
  out=EGS():add(i.cols.names)
  for _,row in pairs(rows or {}) do out:add(row) end
  return out end
--------------------------------------------------------------------------------
GO=obj"GO"
function ok(test,msg)
  print("", test and "PASS "or "FAIL ", msg or "")
  if not test then
    GO.fails= GO.fails+1
    if the.dump then assert(test,msg) end end end

function _.new(i,todo,    defaults,go)
  defaults={}; for k,v in pairs(the) do defaults[k]=v end
  go={}; for k,_ in pairs(GO) do
           if k~="new" and type(GO[k])=="function" then go[1+#go]=k end end
  GO.fails = 0
  for _,x in pairs(todo=="all" and sort(go) or {todo}) do
    for k,v in pairs(defaults) do the[k]=v end
    math.randomseed(the.seed)
    if GO[x] then print(x); GO[x]() end end
  GO.rogue()
  os.exit(GO.fails) end

function GO.rogue( t)
  t={}; for _,k in pairs{ "_G", "_VERSION", "arg", "assert", "collectgarbage",
  "coroutine", "debug", "dofile", "error", "getmetatable", "io", "ipairs",
  "load", "loadfile", "math", "next", "os", "package", "pairs", "pcall",
  "print", "rawequal", "rawget", "rawlen", "rawset", "require", "select",
  "setmetatable", "string", "table", "tonumber", "tostring", "type", "utf8",
  "warn", "xpcall"} do t[k]=k end
  for k,v in pairs(_ENV) do if not t[k] then print("?",k, type(v)) end end end

function GO.cols()
  oo(COLS{names={"Cyldrs", "Acc+"}}) end

function GO.egs(  egs)
  egs = EGS():file(the.file)
  sort(egs.rows)
  print("all", o(egs:mid()))
  print("best",o(egs:copy(slice(egs.rows,1,50)):mid()))
  print("rest",o(egs:copy(slice(egs.rows,#egs.rows-50)):mid()))
  end

function GO.egs1(  egs,a)
  egs = EGS():file(the.file)
  a=egs.rows
  sort(a)
  for j=1,5 do
    for _,col in pairs(egs.cols.x) do col:addy(a[j].cells[col.at],true)  end end
  for j=#a-5,#a do
    for _,col in pairs(egs.cols.x) do col:addy(a[j].cells[col.at],false) end end
  end
--------------------------------------------------------------------------------
if   the.help
then print(help:gsub("%u%u+", "\27[33m%1\27[0m")
               :gsub("(%s)([-][-]?[^%s]+)(%s)","%1\27[32m%2\27[0m%3"),"")
else GO(the.go) end
```