```lua
-- __L5 = A Little Light Learner Lab, in LUA__
-- <img src=img/l5.png align=left width=220>
--
-- [&copy; 2022](https://github.com/timm/l5/blob/master/LICENSE.md#top)
-- Tim Menzies, timm@ieee.org
--
-- [Contribute](https://github.com/timm/l5/blob/master/CONTRIBUTE.md#top)
-- | [Github](http://github.com/timm/l5)
-- | [Issues](https://github.com/timm/l5/issues)
--
-- <a href="https://github.com/timm/l5/actions/workflows/tests.yml"><img
-- src="https://github.com/timm/l5/actions/workflows/tests.yml/badge.svg"></a>
-- <a href="https://zenodo.org/badge/latestdoi/206205826"> <img
--  src="https://zenodo.org/badge/206205826.svg" alt="DOI"></a>
--
-- This is an experiment in  writing the _most_ learners using the
-- _least_ code.  Each learner should be few lines of code (based on  a
-- shared underlying code base).
--
-- Why LUA? Well, it’s a simple langauge. LUA supports simple teaching
-- (less than 2 dozen keywords). Heck, children use it to code up their
-- own games.
--
-- While simple, LUA is also very powerful. LUA supports many advanced
-- programming techniques (first class objects, functional programming,
-- etc) without, e.g.  (**L**ots of (**I**nfuriating (**S**illy
-- (**P**arenthesis)))).  For example, the entire object system used here
-- is just five lines of code (see **is()**).
--
-- Further, LUA code can be really succinct. The other great secret is
-- that, at their core, many of these learners is essential simple. So by
-- coding up those algorithms, in just a few lines of LUA, we are
-- teaching students that AI is something they can understand and
-- improve.
--
-- Lastly,  paradoxically, LUA is useful for teaching _because_ not many
-- people code in that language.  This means it supports the following
-- kind of assignment:  "here is  a worked solution, now code it up in
-- any other language". In that approach, students can get a fully worked
-- solution, yet still have the learning experience of working it out for
-- themselves in their language du jour.

-----------------------------------------------------------------------------
--      _.  |_    _         _|_
--     (_|  |_)  (_)  |_|    |_

local help=[[
L5: a little light learner lab in LUA
(c) 2022 Tim Menzies, timm@ieee.org, BSD2 license

INSTALL:
  requires: lua 5.4+
  download: l5.lua  and data/* from github.com/timm/l5
  test    : lua l5.lua -f data/auto93.csv; echo $? # expect "0"

USAGE:
  lua l5.lua [OPTIONS]
                                                defaults
                                                ~~~~~~~~
  -S  --Seed  random number seed               = 10019
  -H  --How   optimize for (helps,hurts,tabu)  = helps
  -b  --bins  number of bins                   = 16
  -m  --min   min1 size (for pass1)            = .5
  -M  --Min   min2 size (for pass2)            = 10
  -p  --p     distance coefficient             = 2
  -s  --some  sample size                      = 512

OPTIONS (other):
  -f  --file  csv file with data  = data/auto93.csv
  -g  --go    start up action     = nothing
  -v  --verbose show details      = false
  -h  --help  show help           = false]]

-----------------------------------------------------------------------------
--      _
--     -|-        ._    _   -|-  o   _   ._    _
--      |   |_|  | |  (_    |_  |  (_)  | |  _>

-- Define library
local lib={}
-- Trap info needed for finding rogue variables
local b4={}; for k,_ in pairs(_ENV) do b4[k]=k end
-- Large number
lib.big = math.huge

-- __csv(csvfile:str)__ :<br>Iterator. Return one table per line, split on ",".
function lib.csv(csvfile)
  csvfile = io.input(csvfile)
  return function(s, t)
    s=io.read()
    if not s then io.close(csvfile) else
      t={}; for x in s:gmatch("([^,]+)") do t[1+#t] = lib.read(x) end
      return t end end end

-- __cli(t:tab):tab__<br>Check the command line for updates to keys in `t`
function lib.cli(t, help)
  for key,x in pairs(t) do
    x = lib.str(x)
    for n,flag in ipairs(arg) do
     if flag==("-"..key:sub(1,1)) or flag==("--"..key) then
        x= x=="false" and"true" or x=="true" and"false" or arg[n+1] end end
    t[key] = lib.read(x) end
  if t.help then os.exit(print(help:gsub("[%u][%u%d]+","\27[1;31m%1\27[0m"),"")) end
  return t end

-- __demo(THE:tab,go:tab)__<br>Run the demos (or just `THE.go`).
function lib.demos(THE,go)
  local fails,backup = 0,{}
  for k,v in pairs(THE) do backup[k]=v end
  for _,todo in pairs(go[THE.go] and {go[THE.go]} or go) do
    for k,v in pairs(backup) do THE[k]=v end -- reset THE settings to the backup
    math.randomseed(THE.Seed)                -- reset the randomseed
    io.write(".")
    local result = todo()
    if result ~= true then          -- report errors if demo does not return "true"
      fails = fails + 1
      print("--Error",s,status) end end
  for k,v in pairs(_ENV) do  -- Check for rogue locals
    if not b4[k] then print("?",k,type(v)) end end
  os.exit(fails) end -- return the error counts (defaults to zero).


-- __fmt(control:str, arg1,arg2...)__<br>sprintf emulation.
lib.fmt = string.format

-- __gt(x:str):fun__ <br>Return a sort down function on slot `x`.
function lib.gt(x) return function(a,b) return a[x] > b[x] end end

-- __is(name:str) :klass__
-- Object creation.<br>(1) Link to pretty print.<br>(2) Assign a unique id.
-- (3) Link new object to the class.<br>Map klass(i,...) to klass.new(...).
local _id=0
function lib.is(name,    t)
  local function new(kl,...)
    _id = _id+1
    local x=setmetatable({id=_id},kl); kl.new(x,...); return x end
  t = {__tostring=lib.str, is=name}; t.__index=t
  return setmetatable(t, {__call=new}) end

-- __lt(x:str):fun__ <br>Return a sort function on slot `x`.
function lib.lt(x) return function(a,b) return a[x] < b[x] end end

-- __map(t:tab, f:fun):tab__ <br>Return a list, items filtered through `f`.
-- If `f` returns nil, then that item is rejected.
function lib.map(t,f,  u) u={}; for k,v in pairs(t) do u[1+#u]=f(v) end return u end

-- __oo(i:tab)__ : <br>Pretty print `i`.
function lib.oo(i) print(lib.str(i)) end

-- __per(t:tab, p:float):float__
-- Return `p`-th item (e.g. `p=.5` means return the medium).
function lib.per(t,p) p=p*#t//1; return t[math.max(1,math.min(#t,p))] end

-- __push(t:tab, x:atom):x__ <br>Push `x` onto `t`, returning `x`.
function lib.push(t,x) t[1+#t]=x; return x end

-- __rand(?x:num=1):num__<br> Generate a random number `1..x`.
lib.rand= math.random

-- __rnd(n:num, places:int):num__<br>Round `n` to `p` places.
function lib.rnd(n, p)   local m=10^(p or 0); return math.floor(n*m+0.5)/m  end

-- __split(t, ?lo:float=1, ?j:float=#t, ?k:float=1):tab__
-- Return parts of `t` from `i` to `j` by steps `k`.
function lib.splice( t, i, j, k,    u)
  u={}; for n=(i or 1)//1, (j or #t)//1, (k or 1)//1 do u[1+#u]=t[n] end return u end

-- __read(str:str) :bool | int | str__ <br> String to thing.
function lib.read(str)
  str = str:match"^%s*(.-)%s*$"
  if str=="true" then return true elseif str=="false" then return false end
  return math.tointeger(str) or tonumber(str) or str  end

-- __str(i:any) :str__
-- Make pretty print string from tables. Print  slots of associative arrays
-- in sorted order. To actually print this string, use `oo(i)` (see below).
function lib.str(i,       j)
  if type(i)~="table" then return tostring(i) end
  if #i> 0  then j = lib.map(i,tostring) else
    j={}; for k,v in pairs(i) do j[1+#j] = string.format(":%s %s",k,v) end
    table.sort(j) end
  return (i.is or "").."{"..table.concat(j," ").."}" end

-----------------------------------------------------------------------------
--     ._    _.  ._ _    _    _
--     | |  (_|  | | |  (/_  _>

-- Make our classes
-- (1) Data is stored as set of ROW.
-- (2) ROWS are containers for ROW.
-- (3) Columns are summarized as SYMbolics or NUMerics.
-- (4) SOME is a helper class for NUM.
-- (5) RANGE is a helper class for EGS.
-- (6) RANGES is a set of factory functions for making RANGES
local is = lib.is
local ROW,ROWS,SYM,NUM     = is"ROW",   is"ROWS",   is"SYM", is"NUM"
local RANGE,RANGES,SOME    = is"RANGE", is"RANGES", is"SOME"

local add,big,cli,col,csv  = lib.add,   lib.big,    lib.cli,    lib.col,lib.csv
local demos,fmt,gt         = lib.demos, lib.fmt,    lib.gt
local id,klass,lt          = lib.id,    lib.klass,  lib.lt
local map,oo,per,push      = lib.map,   lib.oo,     lib.per,    lib.push
local rand,read,result,rnd = lib.rand,  lib.read,   lib.result, lib.rnd
local seed,splice,str      = lib.seed,  lib.splice, lib.str

local THE = {}
help:gsub(" [-][-]([^%s]+)[^\n]*%s([^%s]+)",function(key,x) THE[key] = read(x) end)
```

```lua
-------------------------------------------------------------------------------
--      ._ _    _   _|_  |_    _    _|   _
--      | | |  (/_   |_  | |  (_)  (_|  _>

-- ##  SOME methods
-- If we keep more than
-- `THE.some` items then SOME replaces old items with the new old items.

-- __col(i:column, has:t, ?at:int=1, ?txt:str="")__
-- For SOME (and NUM and SYM), new columns have a container `has` and appear in
-- column `at` and have name `txt`. If a column name ends in `-`, set its weight
-- to -1.
function col(i,has,at,txt)
  i.n, i.at, i.txt = 0, at or 0, txt or ""
  i.w= i.txt:find"-$" and -1 or 1
  i.has = has end

-- __add(i:column, x:any, nil | inc:int=1, fun:function):x)__
-- Don't add missing values. When you add something, inc the `i.n` count.
function add(i,x,inc,fun)
  if x ~= "?" then
    inc = inc or 1
    i.n = i.n + inc
    fun() end
  return  end

-- __SOME(?at:int=1, ?txt:str="") :SOME__
function SOME.new(i, ...) col(i,{},...); i.ok=false; end
-- __SOME:add(x:num):x__
function SOME.add(i,x)
  return add(i,x,1,function(     a)
    a = i.has
    if        #a < THE.some      then i.ok=false; push(a,x)
  elseif rand() < THE.some/i.n then i.ok=false; a[rand(#a)]=x end end) end

-- __SOME:sorted(): [num]*__ <br>Return the contents, sorted.
function SOME.sorted(i,  a)
if not i.ok then table.sort(i.has) end; i.ok=true; return i.has end

-- ##  NUM methods

-- (1) Incrementally update a  sample of numbers including its mean `mu`,
--     min `lo` and max `hi`.
-- (2) Knows how to calculate the __div__ ersity of a sample (a.k.a.
--     standard deviation).

-- __NUM(?at:int=1, ?txt:str="") :NUM__
function NUM.new(i, ...) col(i,SOME(),...); i.mu,i.lo,i.hi=0,big,-big end
-- __NUM:add(x:num):x__
function NUM.add(i,x)
return add(i,x,1,function(     d)
  i.has:add(x)
  d = x - i.mu
  i.mu = i.mu + d/i.n
  i.hi = math.max(x, i.hi); i.lo=math.min(x, i.lo) end ) end

-- __NUM:clone():NUM__  <br> Duplicate structure
function NUM.clone(i)    return NUM(i.at, i.txt) end

-- __NUM:mid():num__ <br>mid is `mu`.
function NUM.mid(i,p) return rnd(i.mu,p or 3) end
-- __NUM:div():num__ <br>div is entropy
function NUM.div(i, a)
  a=i.has:sorted(); return (per(a, .9) - per(a, .1))/2.56 end

-- __NUM:bin(x:num):num__
-- NUMs get discretized to bins of size `(hi - lo)/THE.bins`.
function NUM.bin(i,x,   b)
  if i.lo==i.hi then return 1 end
  b = (i.hi - i.lo)/THE.bins; return math.floor(x/b+.5)*b end

-- __NUM:norm(x:num):num__<br>Normalize `x` 0..1 for `lo`..`hi`.
function NUM.norm(i,x)
  return i.hi - i.lo < 1E-9 and 0 or (x-i.lo)/(i.hi - i.lo + 1/big) end

-- __NUM:merge(j:num):NUM__ <br> Combine two NUMs.
function NUM.merge(i,j,      k)
  local k = NUM(i.at, i.txt)
  for _,x in pairs(i.has.has) do k:add(x) end
  for _,x in pairs(j.has.has) do k:add(x) end
  return k end

-- ## SYM methods

-- Incrementally update a  sample of numbers including its mode
-- and **div**ersity (a.k.a. entropy)
function SYM.new( i, ...) col(i,{},...);     i.most, i.mode=0,nil end

-- __SYM.clone():SYM__<br>Duplicate the structure.
function SYM.clone(i) return SYM(i.at, i.txt) end

-- __SYM:add(x:any):x__
function SYM.add(i,x,inc)
return add(i,x,inc, function()
  i.has[x] = (inc or 1) + (i.has[x] or 0)
  if i.has[x] > i.most then i.most,i.mode = i.has[x],x end end) end

-- __SYM:merge(j:num):SYM__ <br> Combine two NUMs.
function SYM.merge(i,j,      k)
  local k = SYM(i.at, i.txt)
  for x,n in pairs(i.has) do k:add(x,n) end
  for x,n in pairs(j.has) do k:add(x,n) end
  return k end

-- __SYM:mid():any__ <br>Mode.
function SYM.mid(i,...) return i.mode end
-- __SYM:div():float__ <br>Entropy.
function SYM.div(i,      e)
  e=0;for k,n in pairs(i.has) do if n>0 then e=e-n/i.n*math.log(n/i.n,2)end end
  return e end

-- __SYM:bin(x:any):x__<br>SYMs get discretized to themselves.
function SYM.bin(i,x) return x end

-- __SYM:score(want:any, wants:int, donts:init):float__
-- SYMs get discretized to themselves.
function SYM.score(i,want, wants,donts)
  local b, r, z, how = 0, 0, 1E-10, {}
  how.helps= function(b,r) return (b<r or b+r < .05) and 0 or b^2/(b+r+z) end
  how.hurts= function(b,r) return (r<b or b+r < .05) and 0 or r^2/(b+r+z) end
  how.tabu = function(b,r) return 1/(b+r+z) end
  for v,n in pairs(i.has) do if v==want then b=b+n else r=r+n end end
  return how[THE.How](b/(wants+z), r/(donts+z)) end

-- ##  ROW methods

-- The `cells` of one ROW store one record of data (one ROW per record).
-- If ever we read the y-values then that ROW is `evaluated`. For many
-- tasks, data needs to be __normalized__ in which case -- we need to
-- know the space `of` data that holds this data.
function ROW.new(i,of,cells) i.of,i.cells,i.evaluated = of,cells,false end

-- <b>i:ROW < j:ROW</b> <br>`i` comes before `j` if its y-values are better.
-- This is Zitzler's continuous domination predicate. In summary, it is a small
-- "what-if" study that walks from one way, then the other way, from one
-- example to another. The best row is the one that looses the least.
function ROW.__lt(i,j,        n,s1,s2,v1,v2)
  i.evaluated = true
  j.evaluated = true
  s1, s2, n = 0, 0, #i.of.ys
  for _,col in pairs(i.of.ys) do
    v1,v2 = col:norm(i.cells[col.at]), col:norm(j.cells[col.at])
    s1    = s1 - 2.7183^(col.w * (v1 - v2) / n)
    s2    = s2 - 2.7183^(col.w * (v2 - v1) / n) end
  return s1/n < s2/n end

-- __ROW:within(range):bool__
function ROW.within(i,range,         lo,hi,at,v)
   lo, hi, at = range.xlo, range.xhi, range.ys.at
   v = i.cells[at]
   return  v=="?" or (lo==hi and v==lo) or (lo<v and v<=hi) end

-- ## ROWS methods
-- Sets of ROWs are stored in ROWS. ROWS summarize columns and those summarizes
-- are stored in `cols`. For convenience, all the columns we are not skipping
-- are also contained into the goals and non-goals `xs`, `ys`.

-- __ROWS(src:str | tab):ROWS__
-- Load in examples from a file string, or a list or rows.
function ROWS.new(i,src)
  i.has={}; i.cols={}; i.xs={}; i.ys={}; i.names={}
  if type(src)=="string" then for   row in csv(  src) do i:add(row) end
                         else for _,row in pairs(src) do i:add(row) end end end

-- __ROWS:clone(?with:tab):ROWS__
-- Duplicate structure, then maybe fill it in  `with` some data.
function ROWS.clone(i,with,    j)
  j=ROWS({i.names}); for _,r in pairs(with or {}) do j:add(r) end; return j end

-- __ROWS:add(row: (tab| ROW))__
-- If this is the first row, create the column summaries.
-- Else, if this is not a ROW, then make  one and set its `of` to `i`.
-- Else, add this row to `ROWS.has`.
-- When adding a row, update the column summaries.
function ROWS.add(i,row)
  local function header(   col)
    i.names = row
    for at,s in pairs(row) do
      col = push(i.cols, (s:find"^[A-Z]" and NUM or SYM)(at,s))
      if not s:find":$" then
        if s:find"!$" then i.klass = col end
        push(s:find"[!+-]$" and i.ys or i.xs, col) end end
  end -------------------------------
  if #i.cols==0 then header(row) else
    row = push(i.has, row.cells and row or ROW(i,row))
    for _,col in pairs(i.cols) do col:add(row.cells[col.at]) end end end

-- __ROWS:bestRest()__<br>Return the rows, divided into the best or rest.
function ROWS.bestRest(i,  n,m)
  table.sort(i.has)
  n = #i.has
  m = n^THE.min
  return splice(i.has, 1,  m), splice(i.has, n - m) end

-- __ROWS:mid(?p:int=3) :tab__<br>Return the `mid` of the goal columns.
-- Round numerics to `p` places.
function ROWS.mid(i,p,    t)
  t={}; for _,col in pairs(i.ys) do t[col.txt]=col:mid(p) end; return t end

-- __ROWS:splits(best0:[ROW], rests:[ROW]):[ROW],[ROW],RANGE}__
-- Supervised discretization: get ranges most different between rows.
function ROWS.splits(i,klass,bests0,rests0)
  local most,range1,score = -1
  for m,col in pairs(i.xs) do
    for n,range0 in pairs(RANGES(col,klass,bests0,rests0).out) do
      score = range0.ys:score(1,#bests0,#rests0)
      if score > most then
         most,range1 = score,range0 end end end
  local bests1, rests1 = {},{}
  for _,rows in pairs{bests0,rests0} do
    for _,row in pairs(rows) do
      push(row:within(range1) and bests1 or rests1, row) end end
  return bests1, rests1, range1 end

-- __ROWS:contrast(best0:[row], rests0:[row]):[row]__
-- Recursively find ranges that selects for the best rows.
function ROWS.contrast(i,klass, bests0,rests0,    hows,stop)
  stop = stop or #bests0/8
  hows = hows or {}
  local  bests1, rests1,range = i:splits(klass,bests0, rests0)
  push(hows,range)
  if (#bests1 + #rests1) > stop and (#bests1 < #bests0 or #rests1 < #rests0) then
    return i:contrast(klass,bests1, rests1, hows, stop) end
  return hows,bests0 end

-- ## RANGE methods

-- Given some x values running from `xlo` to `xhi`, store the
-- `ys`  y values seen
function RANGE.new(i, xlo, xhi, ys) i.xlo, i.xhi, i.ys = xlo, xhi, ys end

-- __RANGE:add(x:atom, y:atom)__
function RANGE.add(i,x,y)
  if x < i.xlo then i.xlo = x end -- works for string or num
  if x > i.xhi then i.xhi = x end -- works for string or num
  i.ys:add(y) end

-- **RANGE:__tostring()**<br>Pretty print.
function RANGE.__tostring(i)
  local x, lo, hi = i.ys.txt, i.xlo, i.xhi
  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

-- ## RANGES methods
-- This function generates ranges.
-- Return a useful way to divide the values seen in this column,
-- in these different rows.

-- **RANGES(col: NUM | SYM, rows1:[row], rows2:[row], ...):[RANGE]**
function RANGES.new(i,col,klass, bests,rests)
  i.out={}
  local ranges,n = {}, 0
  for label,rows in pairs{bests,rests} do -- for each set..
    n = n + #rows
    for _,row in pairs(rows) do   -- for each row...
      local v = row.cells[col.at]
      if v ~= "?" then            -- count how often we see some value
        local r = col:bin(v)      -- accumulated into a few bins
        ranges[r] = -- This idiom means "ranges[x]" exists, and is stored in "out".
          ranges[r] or push(i.out,RANGE(v, v, klass(col.at,col.txt)))
        ranges[r]:add(v,label) end end end   -- do the counting
  table.sort(i.out,lt("xlo"))
  i.out = col.is=="NUM" and i:xpand(i:merge(i.out, n^THE.min)) or i.out
  i.out = #i.out < 2 and {} or i.out end -- less than 2 ranges? then no splits found!

-- For numerics, **xpand** the ranges to cover the whole number line.
function RANGES:xpand(t)
  for j=2,#t do t[j].xlo = t[j-1].xhi end
  t[1].xlo, t[#t].xhi = -big, big
  return t end

-- **Merge** adjacent ranges if   they have too few examples, or
-- the whole is simpler than that parts. Keep merging, until we
-- can't find anything else to merge.
function RANGES.merge(i,b4,min,      t,j,a,b,c)
  t,j = {},1
  while j <= #b4 do
    a, b = b4[j], b4[j+1]
    if b then
       c = i:merged(a.ys, b.ys, min) -- merge small and/or complex bins
       if c then
         j = j + 1
         a = RANGE(a.xlo, b.xhi, c) end end
    t[#t+1] = a
    j = j + 1 end
  return #b4 == #t and t or i:merge(t,min) end -- and maybe loop

-- __rangesMerged(i:col,  j:com, min:num): (col | nil)__
-- Returns "nil" if the merge would actually complicate things
--  For discretized values at `col.at`, create ranges that count how
-- often those values  appear in a set of rows (sorted 1,... for best...worst).
function RANGES:merged(x,y,min,      z)
  z = x:merge(y)
  if x.n < min or y.n < min or z:div()<=(x.n*x:div() + y.n*y:div())/z.n then
    return z end end
```

```lua
--------------------------------------------------------------------------------
--       _|   _   ._ _    _    _
--      (_|  (/_  | | |  (_)  _>

-- Place to store tests. To disable a test, rename `go.xx` to `no.xx`.
local go,no={},{}

local function fyi(...) if THE.verbose then print(...) end end

function go.the() fyi(str(THE));  str(THE) return true end

function go.some( s)
  THE.some = 16
  s=SOME(); for i=1,10000 do s:add(i) end; oo(s:sorted())
  oo(s:sorted())
  return true end

function go.num( n)
  n=NUM(); for i=1,10000 do n:add(i) end; oo(n)
  return true end

function go.sym( s)
  s=SYM(); for i=1,10000 do s:add(math.random(10)) end;
  return s.has[9]==1045  end

function go.csv()
  for row in csv(THE.file) do fyi(str(row)) end; return true; end

function go.rows( rows)
  rows = ROWS(THE.file);
  if THE.verbose then map(rows.ys,print) end; return true; end

function go.mid(  r,bests,rests)
  r= ROWS(THE.file);
  bests,rests = r:bestRest()
  print("all",  str(r:mid(2)))
  print("best", str(r:clone(bests):mid(2)))
  print("rest", str(r:clone(rests):mid(2)))
  return true end

function go.range(  r,bests,rests)
  r= ROWS(THE.file);
  bests,rests = r:bestRest()
  for _,col in pairs(r.xs) do
    print("")
    for _,range in pairs(RANGES(col, SYM, bests, rests).out) do
       print(range, range.ys:score(1, #bests, #rests)) end  end
  return true end

function go.contrast(  r,bests,rests)
  r= ROWS(THE.file);
  bests,rests = r:bestRest()
  local how,bests1 = r:contrast(SYM, bests, rests)
  print("all",   str(r:mid(2)))
  print("best",  str(r:clone(bests):mid(2)))
  print("rest",  str(r:clone(rests):mid(2)))
  print("found", str(r:clone(bests1):mid(2)))
  print(#how,str(how))
  return true end

--------------------------------------------------------------------------------
--      _  _|_   _.  ._  _|_
--     _>   |_  (_|  |    |_

if    pcall(debug.getlocal, 4, 1)
then  return {ROW=ROW,      ROWS=ROWS,      SYM=SYM,   NUM=NUM,
              RANGE=RANGE, RANGES=RANGES, SOME=SOME, THE=THE, lib=lib}
else  THE = cli(THE,help)
      demos(THE,go) end
```