Program Guide to Market 8.2

Ralph Abraham

July 13, 2007

This is the PDF of market082.04guide.tex. It is a program guide for market082.04.nlogo, a NetLogo model for a market of portfolio managers. It is a second extension of our basic market model 8.0. We are going to make one extension to 8.1, as described the article *Bubbles and Crashes* (B+C) of 10 January 2007, in Section 5.1 on fickle investors. [See #1 at http://leeps.ucsc.edu/papers.] The changes from 8.1 to 8.2 are:

- Each manager has a new local variable, Rhat, perceived net returns.
- There is a new global variable, $z_0 =$ `z-pool` which receives and stores holdings withdrawn from managers by fickle investors.
- There is a new slider, $\delta =$ `d`, controlling the portion of holdings to withdraw.
- There is a new slider, $\lambda =$ `lambda`, the exponential rate of recruiting from z-pool by a successful manager due to her $\hat{R} =$ `Rhat`.
- There is a new slider, $\rho =$ `rate`, controlling the overall recruiting rate. (see equation 17 of B+C).
- There is new global variable,`total-elr`, for total of terms `exp ( lambda * Rhat` , summed over all managers. This is not described in B+C, so see the code below.

The procedure "update-managers-z" requires the procedures:

- update-managers-loss (as in 8.1, but now involving Rhat),
- compute-mean-Lhat (as in 8.1, but now involving Rhat), and
- compute-total-elr (new to 8.2).

Here we have an expression that does not appear in B+C so we spell it out. When N agents have local variables w_i, A_i, we may form the logit denominator for the A_i with weights, w_i,

$$\bar{A} = \sum \exp w_i A_i,$$

summed from 1 to N, and then the local logit expression,

$$\exp(w_i A_i)/\bar{A}.$$

In this model we apply the logit transformation to the managers local variables, `Rhat`, with weights `z * lambda`, where `lambda` is set by a slider. Thus, we have a global variable, `total-elr`, which in our NetLogo code is computed by the procedure, `compute-total-elr` in each step of the simulation. This procedure also updates each manager's' local variable, `z-pool-draw`, which plays a role in the procedure, `update-managers-z`.

Here are the four procedures from 8.2 that differ from 8.1.

```
to update-managers-loss ;;; update local Lhat (NOT z-weighted) and local Rhat
  ask managers [
    let annual-gross ( u * ( R1 - R0 ) + R0 ) ;;; this is gross return
    let annual-gross-jiggled annual-gross + u * jiggle
    let pay annual-gross-jiggled
    ;;; this is the annual yield NOT reduced to interval of stepsize
    set L 0
    if ( pay < 0 ) [ set L ( - pay ) ]
    let oldLhat Lhat
    let temp1 eta * stepsize
    let temp3 exp (  (-1) * temp1 ) ;;; new to 07a
    let temp4 ( 1 - temp3 )
    set Lhat ( temp4 * L + temp3 * oldLhat ) ;;; new to 07b
    let oldRhat Rhat ;;; update Rhat ;;; REV 082
    let temp2 z * pay
    set Rhat ( temp4 * pay + temp3 * oldRhat )
    ;;; same as Lhat ;;; REV 082
  ]
end

to compute-mean-Lhat
;;; average all managers, IS z-weighted, update c2 as well ;;; REV 081
  let sum-Lhat 0 ;;; temp for running weighted sum
  let sum-Rhat 0 ;;; temp ;;; REV 082
  let sum-z 0 ;;; temp for running weighted sum
  ask managers [ ;;; add Lhat's all manaers
    set sum-Lhat ( sum-Lhat + Lhat * z )
    set sum-Rhat ( sum-Rhat + Rhat * z ) ;;; REV 082
    set sum-z ( sum-z + z )
  ]
  set mean-Lhat sum-Lhat / sum-z  ;;; error repaired in 07f
  set mean-Rhat sum-Rhat / sum-z  ;;; REV 082.0
  set c2 beta * mean-Lhat
  if ( c2 < 0.01 ) [ set c2 0.01 ] ;;; clip small c2
end

to compute-total-elr
;;; denominator in total-elr formula, z-weighted logit total Rhat
  let sum-elr 0 ;;; temp for running weighted sum
  ask managers [ ;;; add elr's  of all managers
    let temp z * exp ( lambda * Rhat ) ;;; z-weighted lambda-logit total Rhat
    set sum-elr ( sum-elr + temp )
    set total-elr sum-elr ;;; global variable, denominator of local z-pool-draw
  ]
  ask managers [ ;;; when total-elr is done, then reset all local draws
    let temp z * exp ( lambda * Rhat ) ;;; z-weighted lambda-logit total Rhat
    set z-pool-draw temp / total-elr ;;; set local draw
  ]
end

to update-managers-z ;;; update turtle variable z and move its ycor, includes clipping
  ask managers [
    let annual-gross ( u * ( R1 - R0 ) + R0 ) ;;; this is gross return
    let annual-gross-jiggled annual-gross + u * jiggle
    let pay cut annual-gross-jiggled
    ;;; this is the annual yield reduced to interval of stepsize
    let z-now z ;;; hold for later
    let z-temp1 z-now * ( 1 + pay ) ;;; add payoff for this step
    let fickle d * stepsize * Lhat ;;; portion to defect
    let z-temp2 ( 1 - fickle ) * z-temp1 ;;; let defectors go
    let recruits rate * stepsize * z-pool * z-now * exp ( lambda * Rhat)
    if recruits < 0 [ set recruits 0 ] ;;; clip
    set z ( z-temp2 + lambda * recruits ) ;;; include recruits
    if (z < zlim) [ set z zlim ]  ;;; clipping
    if (z > zmax) [ set z zmax ]
    ;;; vertical move
    let ytemp ( z - zmin ) * ( width-y - 1 ) / ( 2 * width-z ) ;;; its a float
    set ycor ytemp ;;; convert z to ycor, keep as float
    ;;; update z-pool
    set z-pool ( z-pool + fickle * z-now - lambda * recruits )
    if z-pool < 0 [ set z-pool 0 ] ;;; clip
  ]
```

```
end
```

The processes are outlined graphically in the following figures.

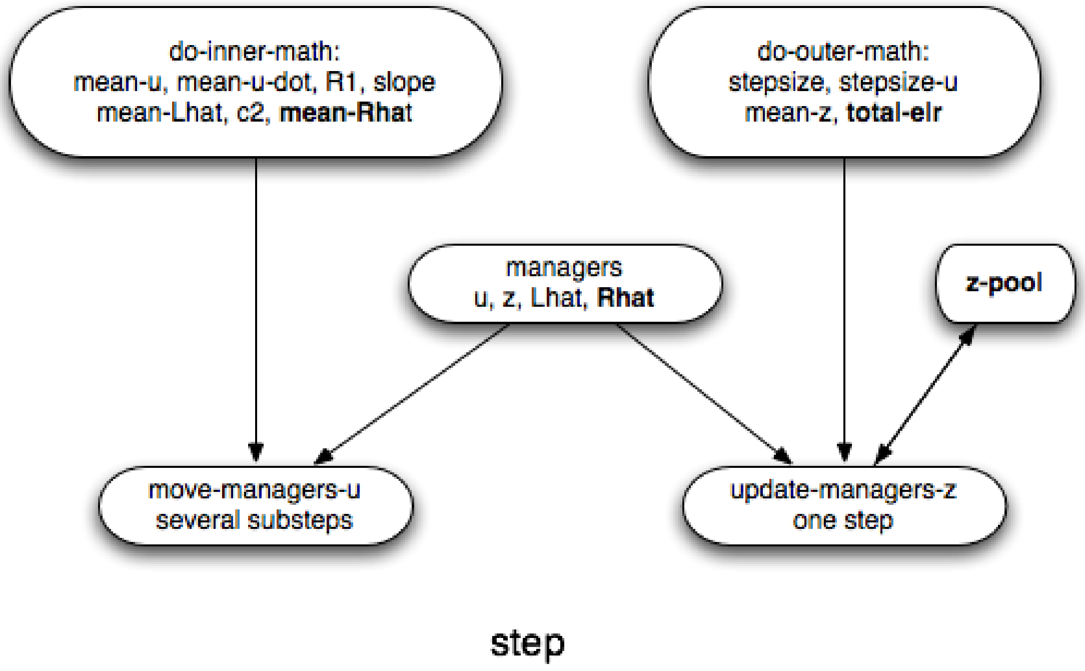

Figure 1: The Big Picture: Outline of the Step Procedure. Details new to the first extension are bold face.

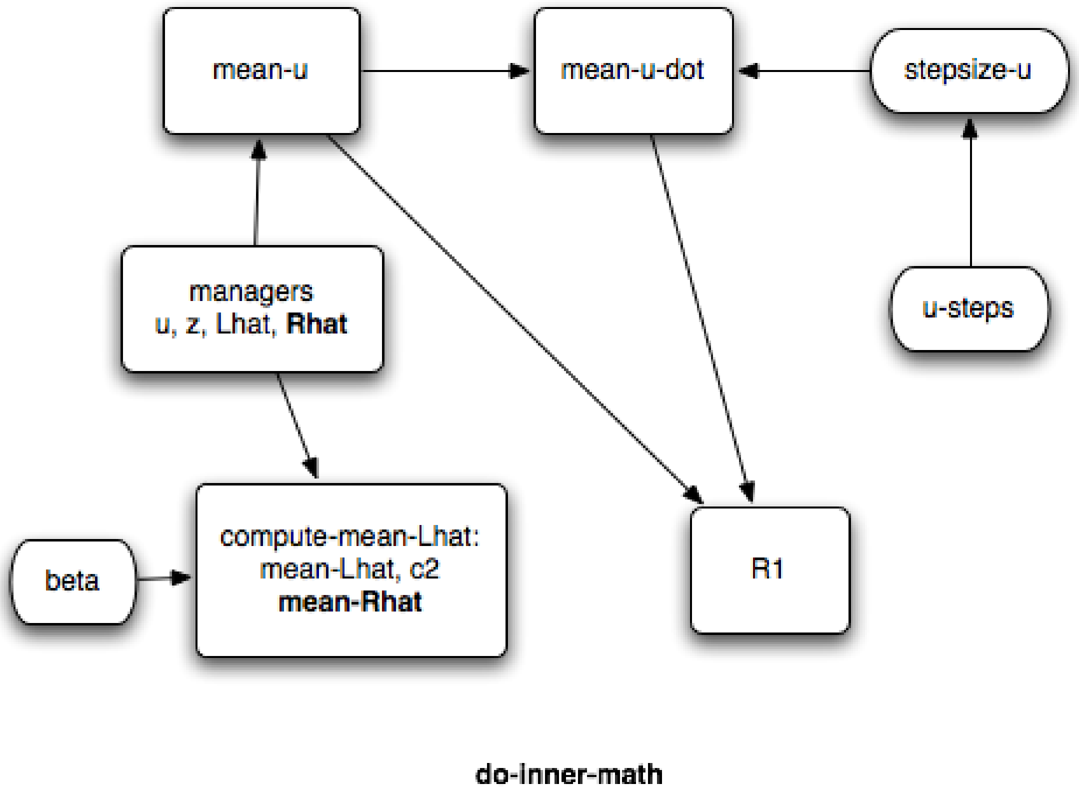

Figure 2: The "do-math" procedure: the inner loop.

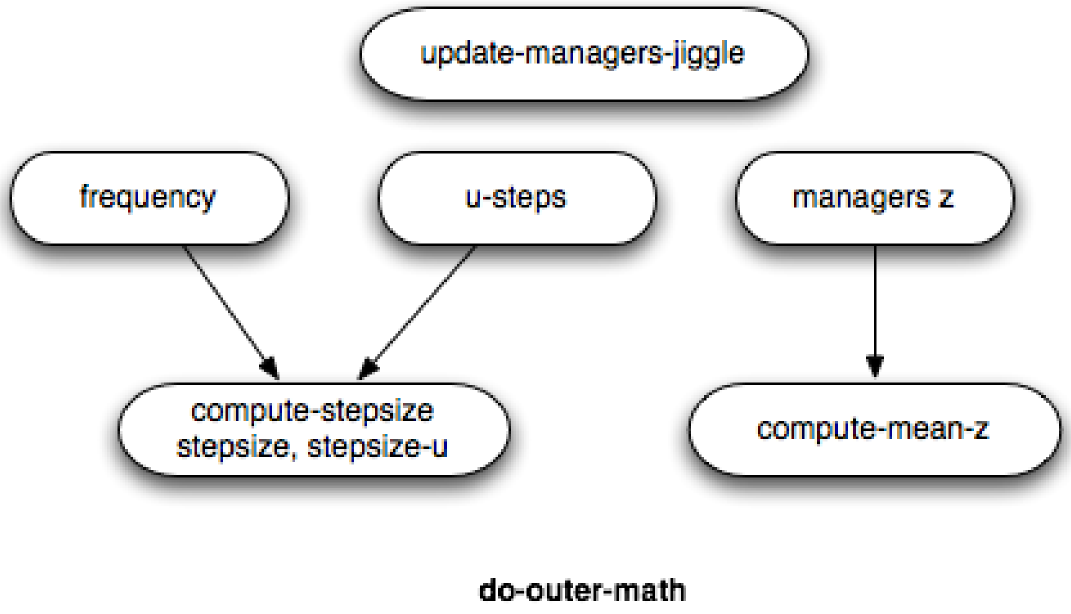

Figure 3: The ”do-math” procedure: the outer loop.

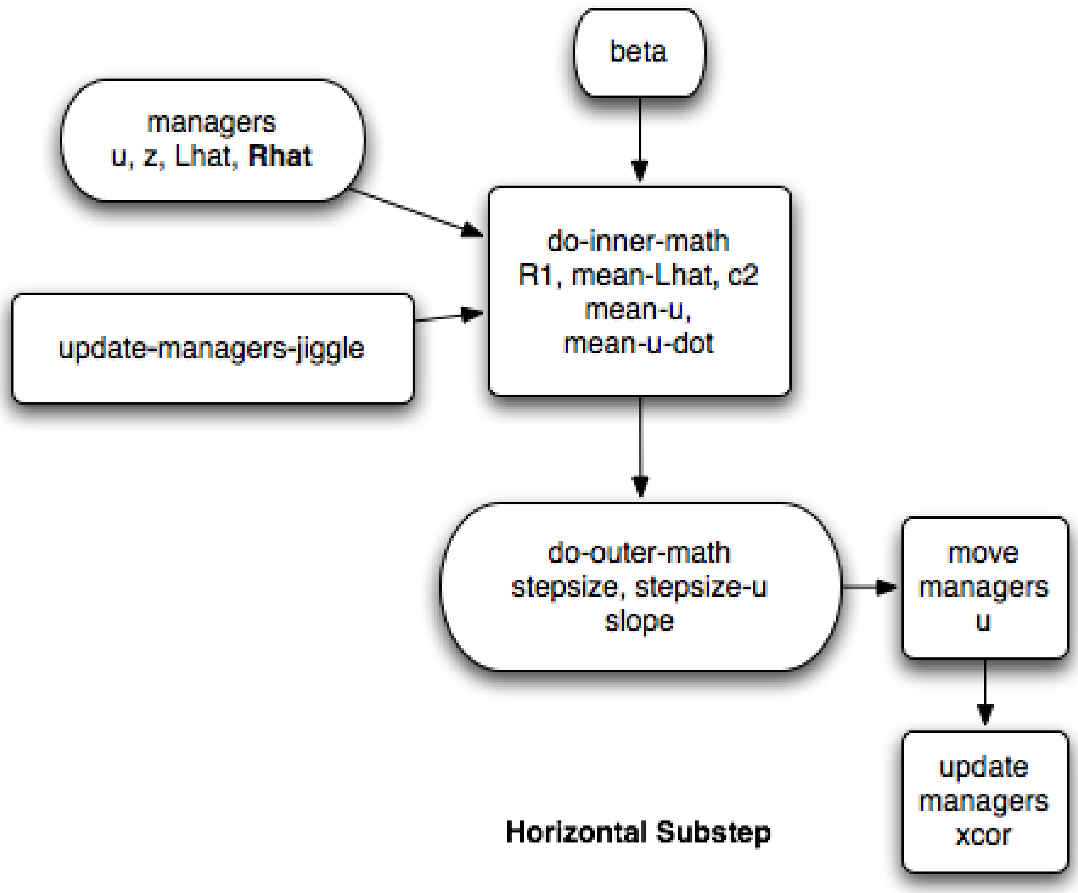

Figure 4: Outline of the horizontal substep procedure. Note: the c2 slider has been replaced by do-inner-math.

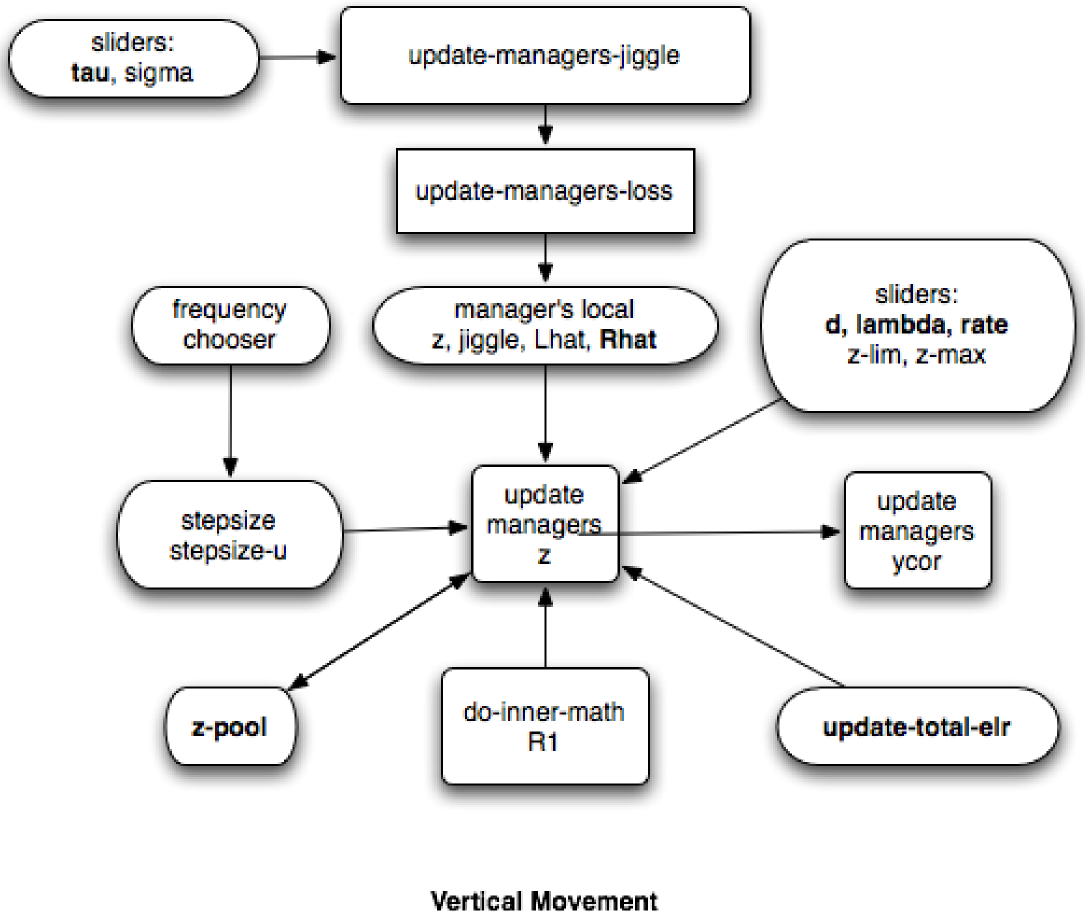

Figure 5: Outline of the vertical step procedure.