Program Guide to Market 8.1

Ralph Abraham

January 9, 2007

This is the PDF of market081.05Bguide.tex. It is a program guide for market081.05B.nlogo, a NetLogo model for a market of portfolio managers. It is an extension of our basic market model 8.0. We are going to make two extensions to 8.0, as described in Section 4 of *Bubbles and Crashes* (B+C) of 31 December 2006: *surprise*, and *losses*.

First we impement *surprise*, a stochastic novelty that is local (that is, different for each manager) at each step of the simulation. This is a luck factor. Here we will follow Subsection 4.1 of B+C, and the article, *Internet Congestion* by Friedman and Huberman, to implement surprise using an Ornstein-Uhlenbeck (OU) process. After this, we will return to B+C 4.1 to implement losses and the c2 dynamic.

The payoff function, equation (11) of B+C, is

$$\phi(x, F) = R(x) = x[(R_s - g_s)\bar{x}^{-\alpha} + g_s - R_o + \alpha\dot{\bar{x}}/\bar{x}] - \frac{1}{2}c_2x^2, \tag{1}$$

Its gradient is thus,

$$\phi_x = (R_s - g_s)\bar{x}^{-\alpha} + g_s - R_o + \alpha\dot{\bar{x}}/\bar{x} - c_2x. \tag{2}$$

As always we write $x = u$. In this model we set $g_s = 0.00$, $R_0 = 0.03$, so $R_s - g_s = 0.08$. Also, we set $\alpha = 2$ as before.

In Model 8.0, the risk cost parameter, c_2, was determined by a slider. But here, c_2 will be determined by losses. Losses accrue to each manager independently, through exogenous perturbations generated by an OU process in each step of the simulation. A local (that is, manager's own) variable, $\hat{L} =$ `Lhat` accumulates the negative values of these perturbations. The perturbations are regarded as annual yield rate, not z-weighted. These local loss functions, the managers' `Lhat` values, are then combined in a global, z-weighted mean, `mean-Lhat`. Finally, a global variable c_2 is set to $\beta\bar{\hat{L}} =$ `beta * mean-Lhat`, where β is determined by a slider. We call this process the c_2 *dynamic*.

Note that the dependence of $R = \phi$ on F is only through $\bar{u}$ and $\dot{\bar{u}}$. So sometimes we write $R(u, \bar{u}, \dot{\bar{u}})$, or equally, $\phi(u, \bar{u}, \dot{\bar{u}})$.

1 Surprise

In each step of the simulation, each manager changes her strategy, u, and thus also the mean strategy, $\bar{u}$ and the payoff function, $\phi(u, \bar{u})$. But even if two managers have chosen the same risk strategy, say u_0, so they have the same mix of riskless and risky investments, they most likely have chosen different risky investments. Thus, their payoffs at the end of this step may differ by a random variable. This difference we call *surprise*. We will implement the surprise for the $i-th$ manager by adding a random term to her payoff.

While the overall payoff function, $\phi(t) = R(u(t), F(t))$ at time t is not changed, we will consider a local modification by addition of a time-dependent surprise term like this, for the $i-th$ manager:

$$R_i(u_i(t)) = R(u_i(t)) + u_i(t)e_i(t)$$

Here we have suppressed notation for the dependence on $\bar{u}$ and $\dot{\bar{u}}$ for convenience. The portfolio size adjustment (and vertical position, `ycor`) is changed. The gradient-based strategy adjustment to u_i (and horizontal position, `xcor`) will be affected as well. In the NetLogo code, the local surprise variable, e_i is denoted `jiggle`.

We now describe the modifications to the procedures for vertical and horizontal jiggle. Note: Vertical movements depend on gross payoffs, while horizontal movements are determined by net payoffs.

1.1 Vertical jiggle

In our basic model, 080.02a, we have this NetLogo code,

```
to update-managers-z ;;; update turtle variable z and move its ycor
  ask managers [
    let annual-gross ( u * ( R1 - R0 ) + R0 ) ;;; this is gross return
    let pay cut annual-gross ;;; this is the annual yield reduced to stepsize
    set z ( z * ( 1 + pay ) ) ;;; increment size of portfolio for timestep
    if (z < zlim) [ set z zlim ] ;;; clipping
    if (z > zmax) [ set z zmax ]
    let ytemp ( z - zmin ) * ( width-y - 1 ) / ( 2 * width-z ) ;;; convert z to ycor
    set ycor ytemp ;;; keep as float
  ]
end
```

This line of NetLogo code, `let pay cut annual-gross`, means, let $pay = cut(annual-gross)$, where *cut* is the adjustment of returns from an annual to a stepsize increment. We are going to change this line by adding, for each manager, a random term to annual-pay.

For each manager, $Z[i]$, we will keep track of a local variable, e_i, the surprise factor. Then $annual - pay$ above will be replaced by $annual - pay + u_i e_i$. Now we are going to use an OU process of the following form, from [Friedman and Huberman, 2004]. Let ν

be an independent draw from the normal distribution with mean zero, with unit standard deviation. We will adjust the deviation externally, by multiplying with the variable, σ, (`sigma` in the NetLogo code), set by a slider.

Let τ be a small positive real, the *persistence parameter*. In the NetLogo code, this is denoted `tau`, set by a slider. Then set,

$$e_i(t + \Delta) = e_i(t)exp(-\tau\Delta) + \nu\sigma\sqrt{W}$$

where W is $[1 - exp(-2\tau\Delta)]/2\tau$. Translating to NetLogo code, this process is written as two procedures:

```
to update-managers-jiggle ;;; REV 081
  ask managers [
    let oldjiggle jiggle
    ;;; this is X of  "internet congestion"
    let nu random-normal 0 1.0 ;;; unit normal, mean 0, sigma 1.0
    ;;; this is ZEE of "internet congestion"
    let temp1  ( -1 * tau * stepsize )
    ;;; exponent in first term of E, new jiggle
    ;;; tau is TAU, persistence parameter (shrink-rate)
    let temp2 sqrt ( ( 1 - exp ( 2 * temp1 ) ) / ( 2 * tau ) )
    set jiggle exp ( temp1 ) * oldjiggle + temp2 * sigma * nu
    ;;; here nu unit becomes random with sigma = SIGMA of paper
    ]
end
```

Next we implement vertical jiggle with this procedure.

```
to update-managers-z ;;; update turtle variable z and move its ycor
  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 ;;; annual yield reduced to stepsize
    set z ( z * ( 1 + pay ) ) ;;; increment size of portfolio for timestep
    if (z < zlim) [ set z zlim ]  ;;; clipping
    if (z > zmax) [ set z zmax ]
    let ytemp ( z - zmin ) * ( width-y - 1 ) / ( 2 * width-z ) ;;; its a float
    set ycor ytemp ;;; convert z to ycor, keep as float
  ]
end
```

1.2 Horizontal jiggle

The suprise factor affects the horizontal motion of managers also, in each substep. In our basic model, 8.0, we have this NetLogo code,

```
to move-managers-u ;;; move one substep up slope (all managers) ;;; REV 080
  ask managers [ ;; update u and clip if necc
    ;;;;; recompute slope, each manager
    do-inner-math ;;; compute mean-u, mean-u-dot, and R1
    let phisubx ( R1 - R0 - c2 * u )
    let u-jump ( stepsize-u * phisubx )
    ;;; now clipping
    if (u-jump > max-u-jump) [ set u-jump max-u-jump ] ;;; clip u-jump right
    if (u-jump < ( - max-u-jump)) [ set u-jump ( - max-u-jump ) ] ;;; clip u-jump left
    ;;; now move u
    set u ( u + u-jump )
    ;;; more clipping
    if (u < umin ) [ set u umin ] ;;; clipping left edge
    if (u > umax ) [ set u umax ] ;;; clipping right edge
  ]
end
```

Now we are going to change the line,

```
    let phisubx ( R1 - R0 - c2 * u )
```

to

```
    let temp ( R1 - R0 - c2 * u + jiggle )
```

where `jiggle` is the current value of the manager's surprise variable, e_i.

That completes the surprise extension for portfolio worth. These two jiggles, horizontal and vertical, complete Section 1, our implementation of the surprise part of extension from Market 8.0 to Market 8.1. Now, on to the second and final extension, losses.

2 Losses

This extension of the basic model allows for the risk cost parameter, c_2 (or `c2`), to vary in proportion to loss. We begin with a reprise of B+C, 4.1. Here we find the four variables,

- β, or `beta`, sensitivity
- η, or `eta`, investor persistence
- γ, or `gamma`, surprise persistence

- σ, or `sigma`, volatility of surprise

But, as we have determined surprise by the OU process, with its parameters, σ and τ, the variables γ and σ in the above list of four are no longer needed. Only β and η will be used in this section.

2.1 Loss variables and c2 dynamic

We add two new global variables to our model, $L_n =$ `Lnow`, and $\hat{L} =$ `Lhat`, current cumulative loss and exponential average loss, respectively. Then, with each step of the simulation, and for each manager, we update `jiggle`, then update $\hat{L}$, and finally, reset $c_2 = \beta\hat{L}$. (This is the only use of β.)

Assuming that the surprise, `jiggle` (which is used in all three local adjustments – horizontal jiggle, vertical jiggle, and loss – has been updated) then $\hat{L}$ is updated by this rule (B+C, 4.1):

$$\hat{L}_i = \eta\Delta L_i + (1 - \eta\Delta)\hat{L}_{i-1}$$

where $L_i = max\{0, -u_i e_i\}$, and Δ is the timestep, from the `frequency` menu in the model.

```
to update-managers-loss ;;; update local Lhat (NOT z-weighted)
  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 st
    set L 0
    if ( pay < 0 ) [ set L ( - pay ) ]
    let oldLhat Lhat
    let temp1 eta * stepsize
    let temp3 exp (  (-1) * temp1 )
    let temp4 ( 1 - temp3 )
    set Lhat ( temp4 * L + temp3 * oldLhat )
  ]
end
```

3 To step or go

Above we have described some of the bits and pieces. Now we assemble them into the "step" procedure. Each step is accomplished as a number, `u-steps`, of substeps. The main step in our market simulation is a carefully orchestrated sequence of four stages.

3.1 Substeps

Horizontal movement is the result of a massive process, outlined in Figure 1.

3.2 Steps

Vertical movement is the result of a simple process, outlined in Figure 2. The computed values, mean-u, mean-u-dot, R1, and jiggle are read only in this stage, having been computed at the end of the preceding step.

Step 2a (adjusting the position of the yellow king) is a two-step Newton process is applied to the function,

$$f(u) = (R_s - g_s)u^{-2} + g_s - R_o - c_2 u$$

to locate its zero, u^*. This is our new code.

```
;;; to clip number outside of interval (-0.1, +0.1)
to-report clip-away [ number ]
    let tempmag abs number
    let tempsign 1
    let tempresult 0
    if not ( tempmag = 0 ) [ set tempsign ( number / tempmag ) ]
    ifelse ( tempmag < 0.1 ) [ set tempresult ( tempsign * 0.1 ) ] [set tempresult number ]
    report tempresult
end

to adjust-yellow-king ;;; move him to crit position
  ask turtle 2 [
    let a ( Rs - gs1 ) ;;; coeff of u ^ (-alpha) term in func, aka dR
    let b ( gs1 - R0 ) ;;; constant term in func
    let u0 u ;;; to begin newton method
    let func0 a / ( u0 ^ 2 ) + b  - ( c2 * u0 )
    let funcdot0 -2 * a / ( u0 ^ 3 ) - c2
    set funcdot0 clip-away funcdot0 ;;; not to divide by zero
    let u1 u0 - func0 / funcdot0 ;;; first step of newton
    set u1 clip-away u1 ;;; not to divide by zero
    let func1 a / ( u1 ^ 2 ) + b  - ( c2 * u1 )
    let funcdot1 -2 * a / ( u1 ^ 3 ) - c2
    set funcdot1 clip-away funcdot1 ;;; not to divide by zero
    let u2 u1 - func1 / funcdot1 ;;; 2nd step of newton
    ;;; clip u2
    if ( u2 > umax ) [ set u2 umax ]
    if ( u2 <  umin ) [ set u2 umin ]
    set u u2  ;;; record current u2 as u0 for next step
    set xcor ( (u2 - umin) * width-x / width-u + xmin )
    set ycor -1
] end
```

These processes are outlined graphically in the following figures.

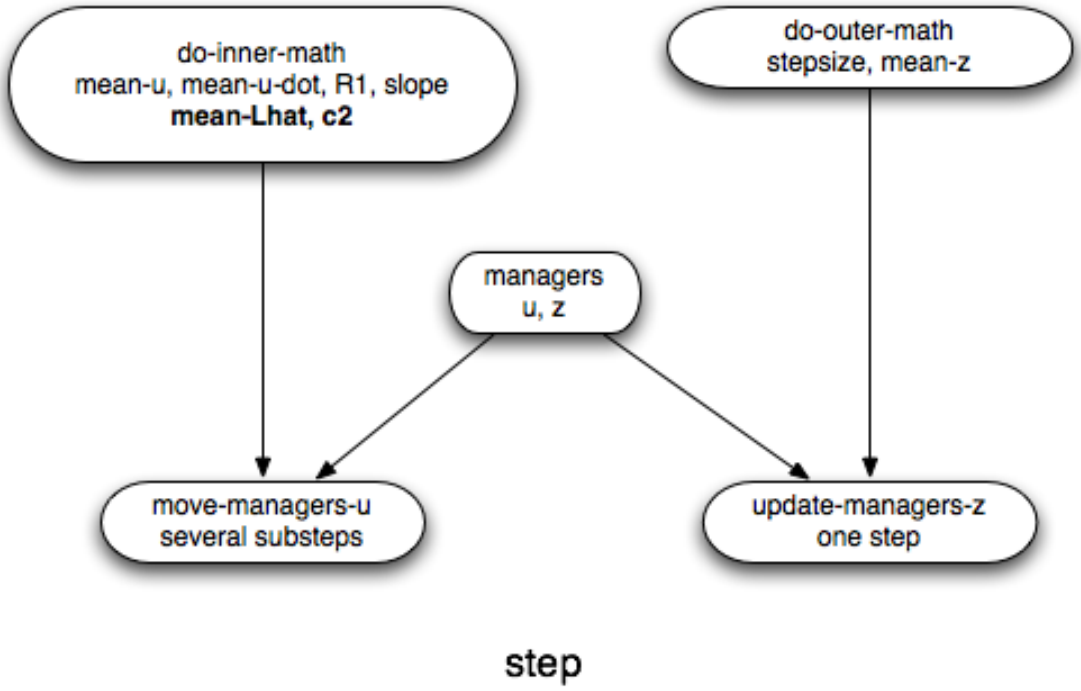

Figure 1: The Big Picture: Outline of the Step Procedure. Details new to the first extension are bold face.

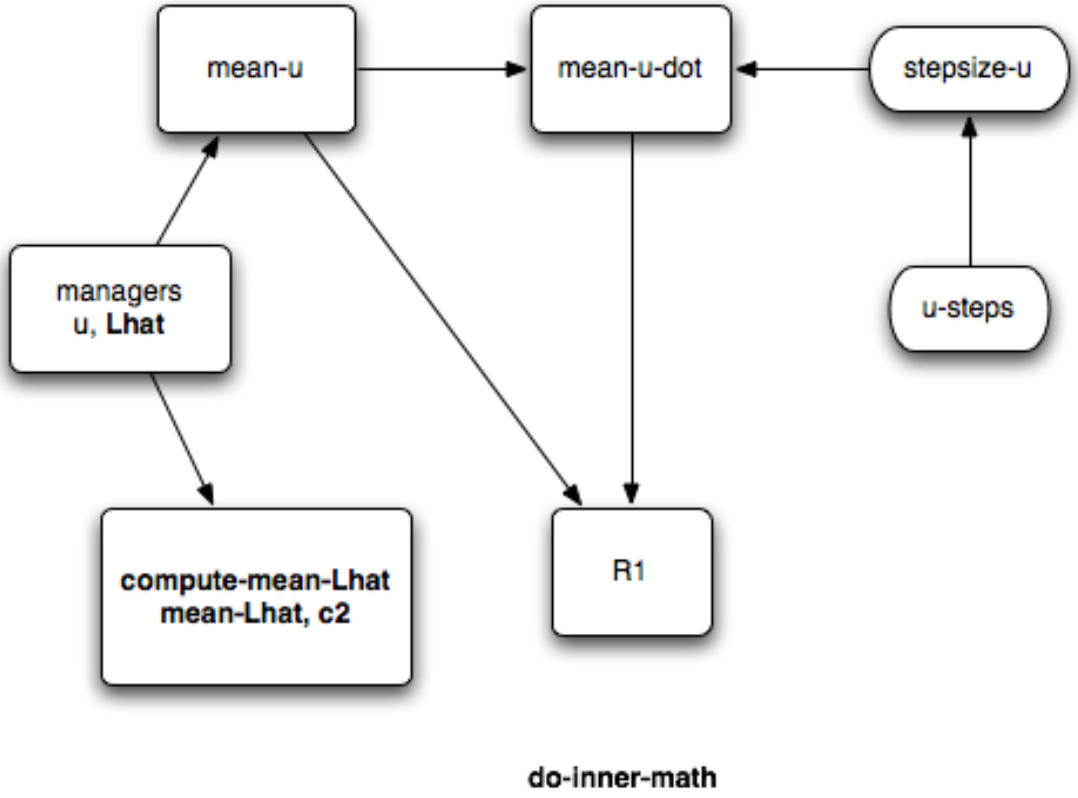

Figure 2: The "do-math" procedure: the inner loop.

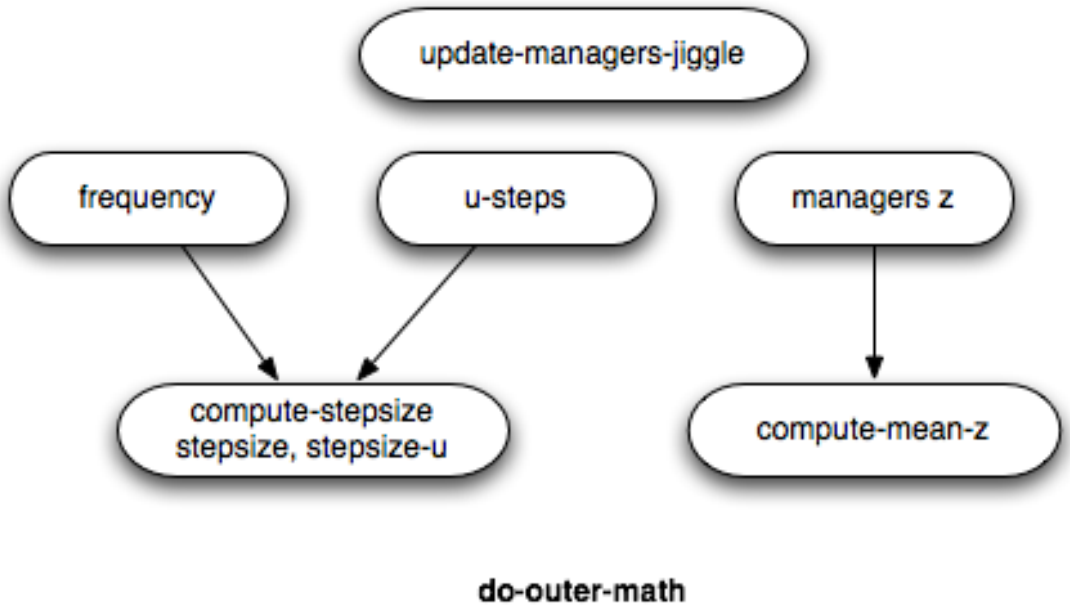

Figure 3: The "do-math" procedure: the outer loop.

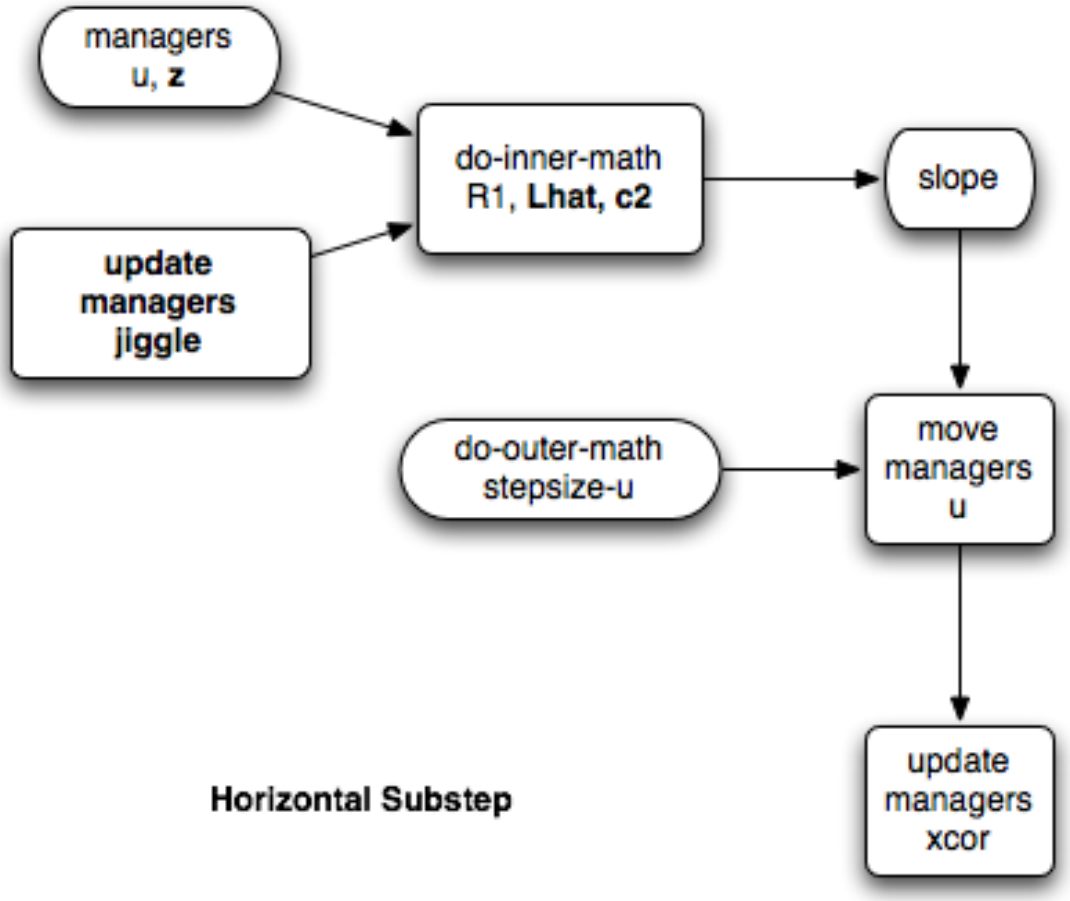

Figure 4: Outline of the horizontal substep procedure. Note: the c2 slider has been replaced by do-inner-math.

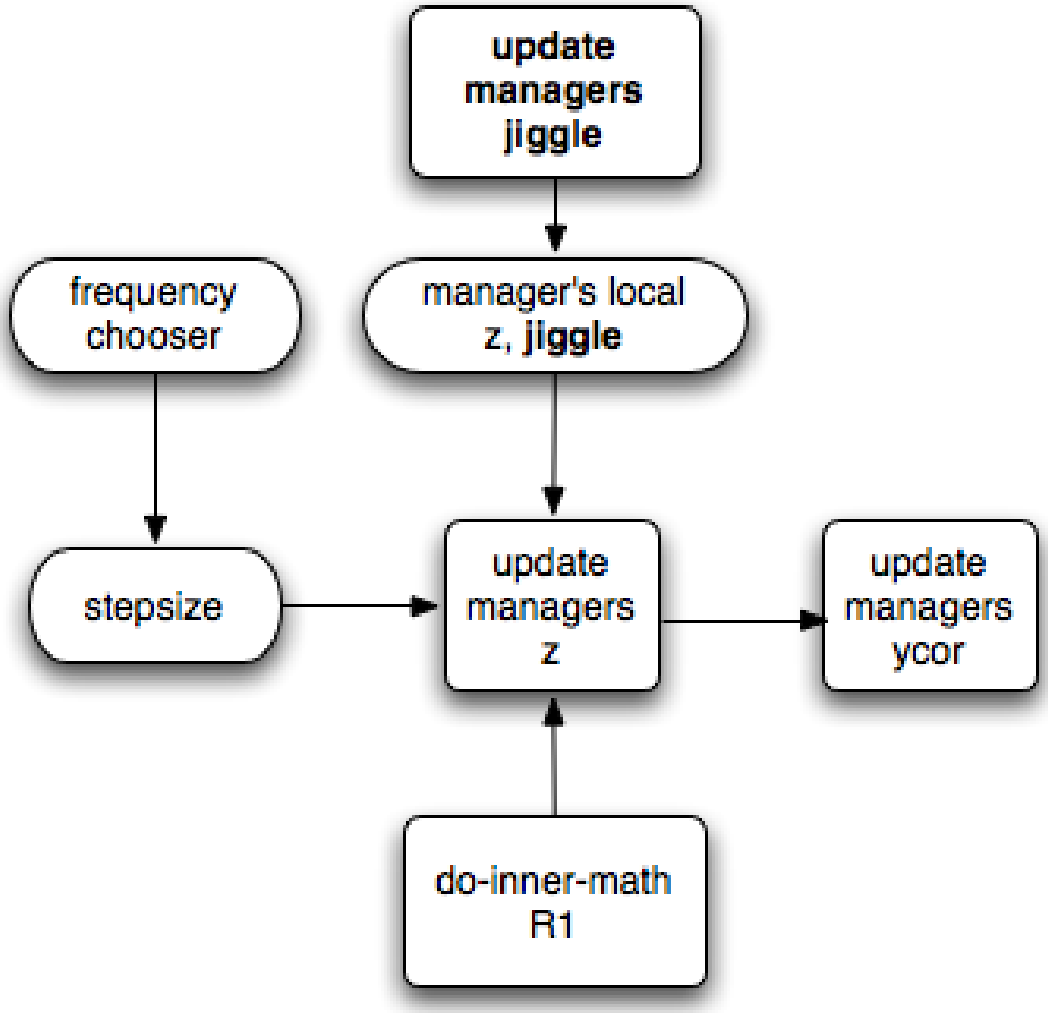

Figure 5: Outline of the vertical step procedure.