Fast(er) R Code

Paul Teetor

R/Finance Conference

May 2012

In R, "for" loops and data copying are slow.

- This code has performance problems:

```
for (i in …) {
    for (j in …) {
        dframe <- func(dframe,i,j)
    }
}
```

- Instead, use *vectorized operations*, the *"apply" functions*, and *functional programming*

Vectorized operations can replace a basic `for` loop

- Instead of explicit element-by-element loop

```
for (i in 1:N) { A[i] <- B[i] + C[i] }
```

  use the vectorized equivalent:

```
A <- B + C
```

- Works for many operators and functions: `A+B, A-B, A*B, A/B, A%%B, sqrt(A), log(A)`
- Check out: `rnorm(length(A),mean=A)` and even `paste(A,B)`!

lapply: Apply a function to a list

- Suppose *lst* is a list and *fun* is a function.
- Then `lapply(`*`lst`*`,`*`fun`*`)` returns a new list:

  fun(*lst*[[1]]), *fun*(*lst*[[2]]), *fun*(*lst*[[3]]), . . .

```
> lst <- list(1, 2, 9)
> sqrt(lst)   # sqrt wants a vector, not a list
Error in sqrt(lst) : Non-numeric argument to mathematical
function
> lapply(lst, sqrt)
[[1]]
[1] 1

[[2]]
[1] 1.414214

[[3]]
[1] 3
```

The 'apply' family has other members, all with a common theme

- `apply(`*`mat`*`,`*`n`*`,`*`fun`*`)` – Apply a function to the rows or columns of a matrix
- `mapply(`*`fun`*`,`*`lst`*$_1$`,`*`lst`*$_2$`,...)` – Apply a function to several lists in parallel
- `tapply(`*`x`*`,`*`factor`*`,`*`fun`*`)` – Apply a function to groups within *x* defined by *factor*
- Also `sapply(`*`lst`*`,`*`fun`*`)`, `replicate(`*`n`*`,`*`expr`*`)`, `va`*`pply(`*`lst`*`,`*`fun`*`,...)`

Functional Programming: Designed to eliminate assignments

- `Filter(f,x)` – Returns the elements of *x* for which *f* is true
- `Reduce(f,x)` – Iterates over a list or vector, *x*, applying *f* to its successive *results* of itself
- Suppose *x* = *x1, x2, x3, x4, x5,…* Then `Reduce(f,x)` successively applies *f* like this:
  - `f(x1,x2), x3, x4, x5, . . .`
  - `f(f(x1,x2), x3), x4, x5, . . .`
  - `f(f(f(x1,x2), x3), x4), x5, . . .`

Toy Examples of Reduce(f,x): sum and cume. product

- Iterative summation:

```
s <- x[1] + x[2]
for (i in 3:length(s)) s <- s + x[i]
```

- Implemented using Reduce:

```
f <- function(a,b) a + b
s <- Reduce(f, x)
```

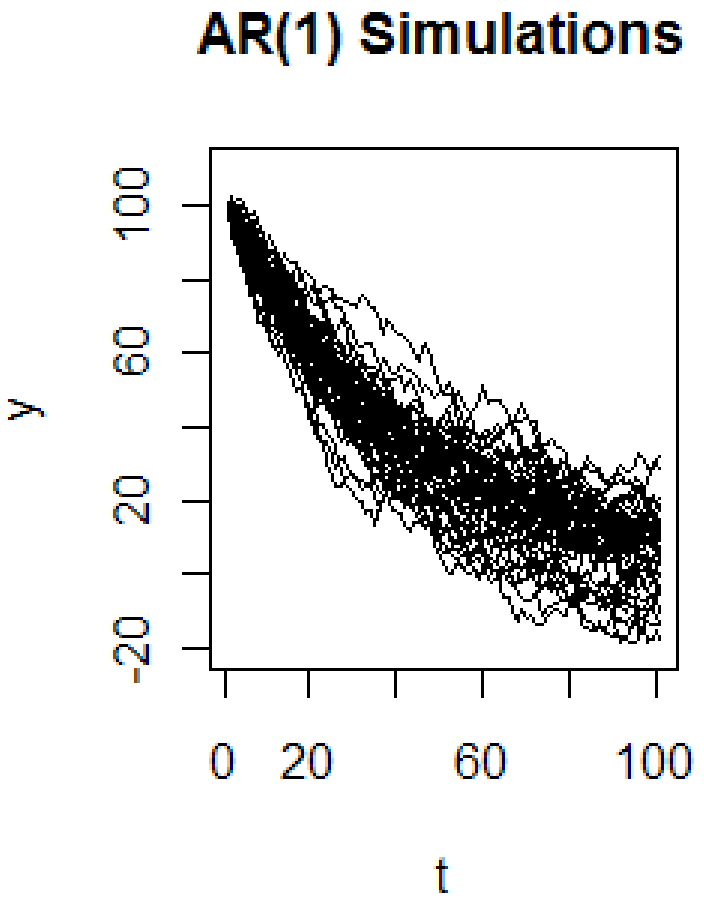
- Cumulative product:

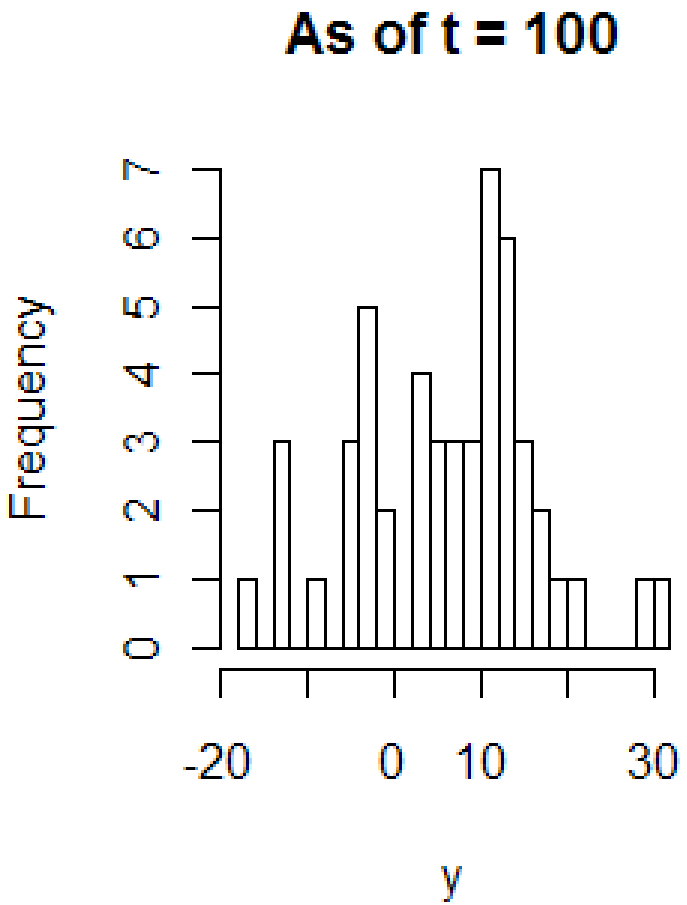
```
f <- function(a,b) a * b
prods <- Reduce(f, x, accumulate=TRUE)
```

Example: AR(1) Monte Carlo, no loops, no copying

```
phi <- 0.975; sigma <- 2.5
nSteps <- 100; nPaths <- 50
f <- function(x,eps) phi*x + eps
mkpath <- function()
   Reduce(f, rnorm(nSteps,sd=sigma),
          init=100, acc=T)
paths <- replicate(nPaths,mkpath())
```

AR(1) Simulations: 50 Paths





Fast(er) R Code

- Slides on-line at

  http://quanttrader.info/public

- Code snippets under

  https://github.com/pteetor/public

paulteetor@yahoo.com

@pteetor