Faster Computing - Part 1

Laura Vary

11/18/2021

Initial Remarks

We're going to create models today. This is a model one could call 'noisy geometric growth'. Note: Sandwiching something with two dollar signs on either side creates a math chunk.

$$N_{t+1} = \lambda N_t$$

Now consider that the growth rate could change at different time steps.

$$N_{t+1} = \lambda_t N_t$$

Now add in the noisy growth.

$$N_{t+1} = \lambda_t N_t \lambda_t = \bar{\lambda} e^{Z_t} Z_t \sim normal(0, \sigma^2)$$

Note: the double \ stops an equation and allows you to start a next one on the next line.
This is a logarithmic growth model, normal distribution, with error randomly distributed. For every timestep t, we draw a number from the normal distribution, find our lambda, and apply that to the next timestep knowing the value at the initial timestep (N_0).

Another note on math environments. Can do inline math as well: Ex. we are discussing two parameters: $\bar{\lambda}$ and σ^2. The 1 dollar sign sandwich signifies in-line math.

Now to implement the model:

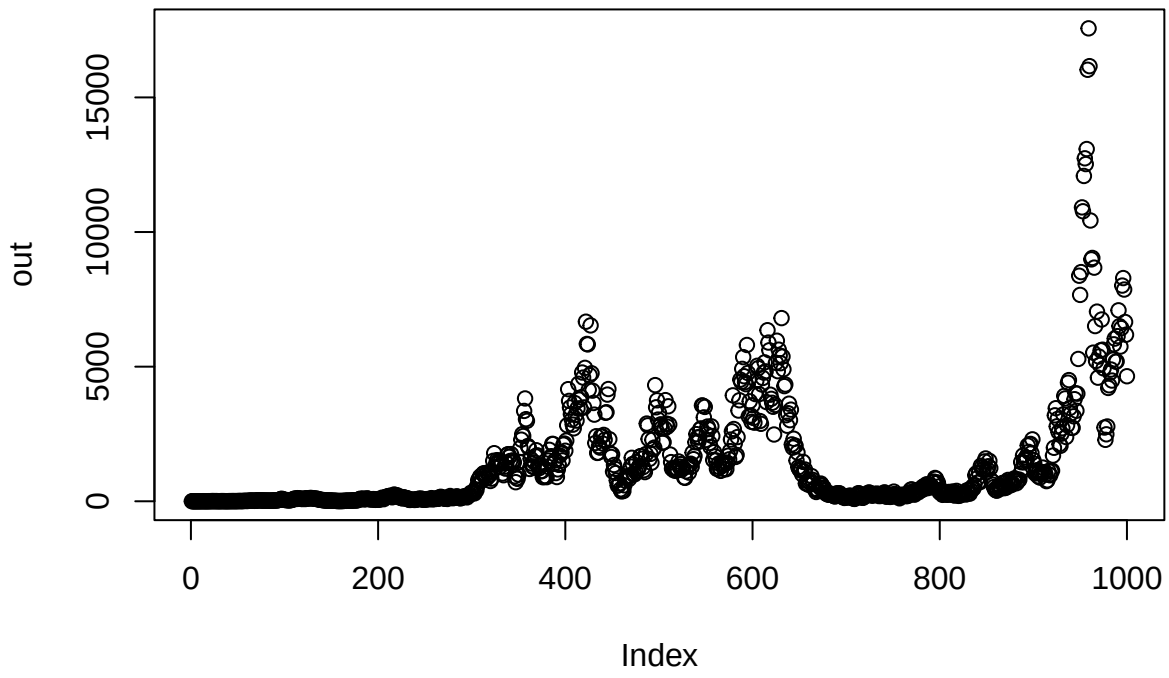
```
geom_growth_base <- function(N0=2, lambda=1.01, sigma=0.2, tmax=999){
  Nvals <- vector('numeric') #create an empty vector to store output
  Nvals[1] <- N0 #create an initial population size

  for(t in 1:tmax){
    Z_t <- rnorm(1, 0, sigma) #pull a random number from a normal distribution
    lambda_t <- lambda*exp(Z_t)
    Nvals[t+1] <- lambda_t*Nvals[t]
  }
  return(Nvals)
}
```

New code chunk that calls that function and then plots the output.

```
set.seed(1)
out <- geom_growth_base()

plot(out)
```



Given data on population sizes, i.e., the N_ts, you could estimate the growth rate as follows.

$$\hat{\lambda}_t = \frac{N_{t+1}}{N_t}$$

Benchmark Operations

Now we can estimate how long certain applications take long in computing time. Here, we'll use "Sys.time()" which is essentially a stopwatch timing how long your function takes to run. One should always set the same seed and use the same random numbers for benchmarking. Setting a seed anew makes sure that we're using the same random number to ensure reproducibility.

```
#Using a default number of time-points:
start_time <- Sys.time()
out <- geom_growth_base()
end_time <- Sys.time()

end_time - start_time
```

```
## Time difference of 0.01994705 secs
```

Without setting a seed, the time difference will vary with every iteration.

```
start_time <- Sys.time()

set.seed(1)
out <- geom_growth_base()
```

```
end_time <- Sys.time()

end_time - start_time
```

```
## Time difference of 0.01994681 secs
```

Now we can test how different methods impact the computation time.

```
#Increase time stamps:
start_time <- Sys.time()

set.seed(1)
out <- geom_growth_base(tmax=10^6)

end_time <- Sys.time()

end_time - start_time
```

```
## Time difference of 5.135203 secs
```

```
#Change constant:
start_time <- Sys.time()

set.seed(1)
out <- geom_growth_base(sigma=1)

end_time <- Sys.time()

end_time - start_time
```

```
## Time difference of 0.01196098 secs
```

Benchmarking with the microbenchmark package:

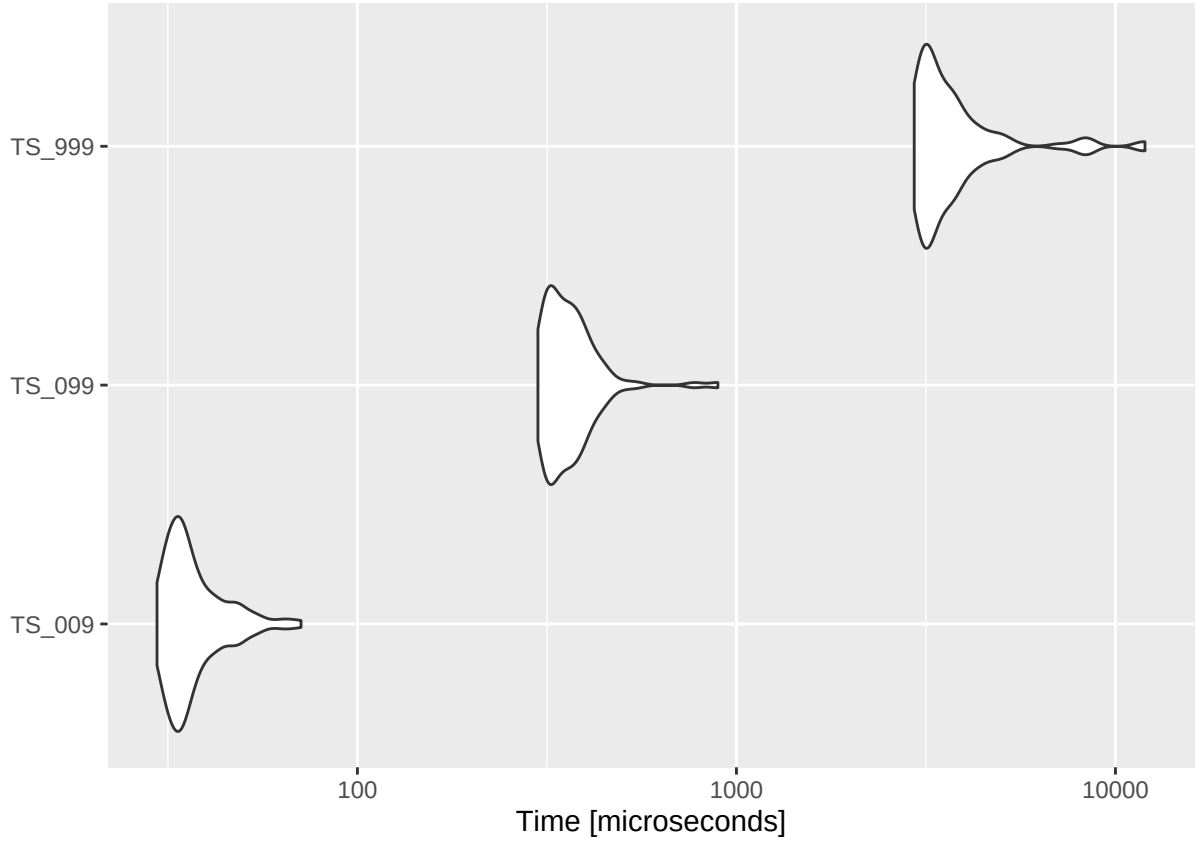
This chunk evaluates the computational time for different maximum time stamps and plots the range of microseconds. Microbenchmark evaluates the speed of multiple functions repeatedly (default neval = 100 times) and return summary statitsics. It also plays well with ggplot to enable quick visual comparisons.

```
comp <- microbenchmark(TS_009 = {geom_growth_base(tmax=9)},
                       TS_099 = {geom_growth_base(tmax=99)},
                       TS_999 = {geom_growth_base(tmax=999)})
comp
```

```
## Unit: microseconds
##    expr    min      lq     mean  median      uq     max neval
##  TS_009   29.6   32.50   37.556   34.60   39.70    71.0   100
##  TS_099  299.4  320.75  368.829  353.05  390.25   893.5   100
##  TS_999 2944.9 3135.75 3926.796 3385.25 3903.25 11965.5   100
```

```
autoplot(comp)
```

```
## Coordinate system already present. Adding new coordinate system, which will replace the existing one
```



Now we can test what happens if we pre-allocate the vector Nval in terms of space, and compare to the base function where we didn't do that using microbenchmark.

```
#Now we can test what happens if we pre-allocate amount of space in Nvals.

geom_growth_preallocated <- function(N0=2, lambda=1.01, sigma=0.2, tmax=999){
  Nvals <- rep(NA,tmax) #create an empty vector to store output
  Nvals[1] <- N0

  for(t in 1:tmax){
    Z_t <- rnorm(1, 0, sigma)
    lambda_t <- lambda*exp(Z_t)
    Nvals[t+1] <- lambda_t*Nvals[t]
  }
  return(Nvals)
}
```

```
geom_growth_base <- function(N0=2, lambda=1.01, sigma=0.2, tmax=999){
  Nvals <- vector('numeric') #create an empty vector to store output
  Nvals[1] <- N0 #create an initial population size
```

```
  for(t in 1:tmax){
    Z_t <- rnorm(1, 0, sigma) #pull a random number from a normal distribution
    lambda_t <- lambda*exp(Z_t)
    Nvals[t+1] <- lambda_t*Nvals[t]
  }
  return(Nvals)
}
```

```
microbenchmark(old = {geom_growth_base(tmax=9999)},
               new = {geom_growth_preallocated(tmax=9999)})
```

```
## Unit: milliseconds
##  expr     min       lq     mean   median       uq      max neval
##   old 29.7867 35.38045 42.93228 38.75105 47.03895 112.1046   100
##   new 26.2549 31.92330 38.93029 36.26710 46.73480  74.6131   100
```

Pre-allocation gives slightly shorter computational time, but not a huge difference.

Thinking In Vectors - Generate some 'data' on population sizes

```
Nobsv <- geom_growth_preallocated(tmax=9999) #observed population sizes

start_time <- Sys.time()

growth_rates <- vector('numeric',(length(Nobsv)-1))
for(i in 1:(length(Nobsv)-1)){
  growth_rates[i] <- Nobsv[i+1]/Nobsv[i]
}

end_time <- Sys.time()

end_time - start_time
```

```
## Time difference of 0.02561593 secs
```

Computational power can be decreased by creating entire vectors to conduct operations between, rather than making a relatively drawn out set of operations that are by individual numbers. E.g., dividing one value by the next value in a series takes longer than dividing one entire vector by another entire vector.

```
tmax <- 9999
Nobsv <- geom_growth_preallocated(tmax=tmax)

start_time <- Sys.time()

growth_rates <- vector('numeric',length(data)-1)

Nnow <- Nobsv[-tmax]
Nnext <- Nobsv[-1]

growth_rates <- Nnext/Nnow #dividing one vector by another,  which saves computational time.
```

```r
end_time <- Sys.time()

end_time - start_time
```

```
## Time difference of 0.008973122 secs
```