```
naic.arima.3  = function (model, nruns, t, block) {

# function runs “nruns” simulations
# model  needs to be of the Arima type
# starts at an arbitrary place on the actual data and then bootstraps from centered residuals
# assumes data is log transformed

# extract and center residuals
resid_cent = model$residuals -  mean(model$residuals)
min_start = max(model$arma)+2
max_start = length(model$x)


# initiate result matrix
sim_normal = vector(length=nruns)
sim_results = matrix(NA, nrow = nruns, ncol = t)


for (i in 1:nruns)  {

#re-initializing vectors

       init_location = 0
       resid_vector = 0
       short_data = 0
       stack = ceiling(t/block)

       #creating run-specific
       #random selection of simulation starting location is at the core of “through the cycle”
       init_location = sample((min_start:max_start), size=1)
       init_location2 = init_location-1
```

```
        resid_vector = vector()


        for (j in 1:stack) {

        # samples blocks of residuals to create a new vector for simulation


                resid = 0

                resid = sample((max_start-block), size= 1)

                resid_vector = append(resid_vector, resid_cent[resid:(resid+block)])


        }


        resid_vector = resid_vector[1:t] # truncates vector so that length = t

        short_data = model$x[1:init_location2]


        # create an Arima object and simulate


        sim_arima = Arima(short_data, model = model) # version 1

        sim_normal[i] = model$x[init_location]

        sim_results [i,] = simulate (sim_arima, nsim = t, bootstrap = FALSE, innov =
        resid_vector)


}


results = cbind(sim_normal,sim_results)

results = exp(results)  # assuming log model

results = results/results[,1]

return (results)


}
```