Analysing Netlogo experiments

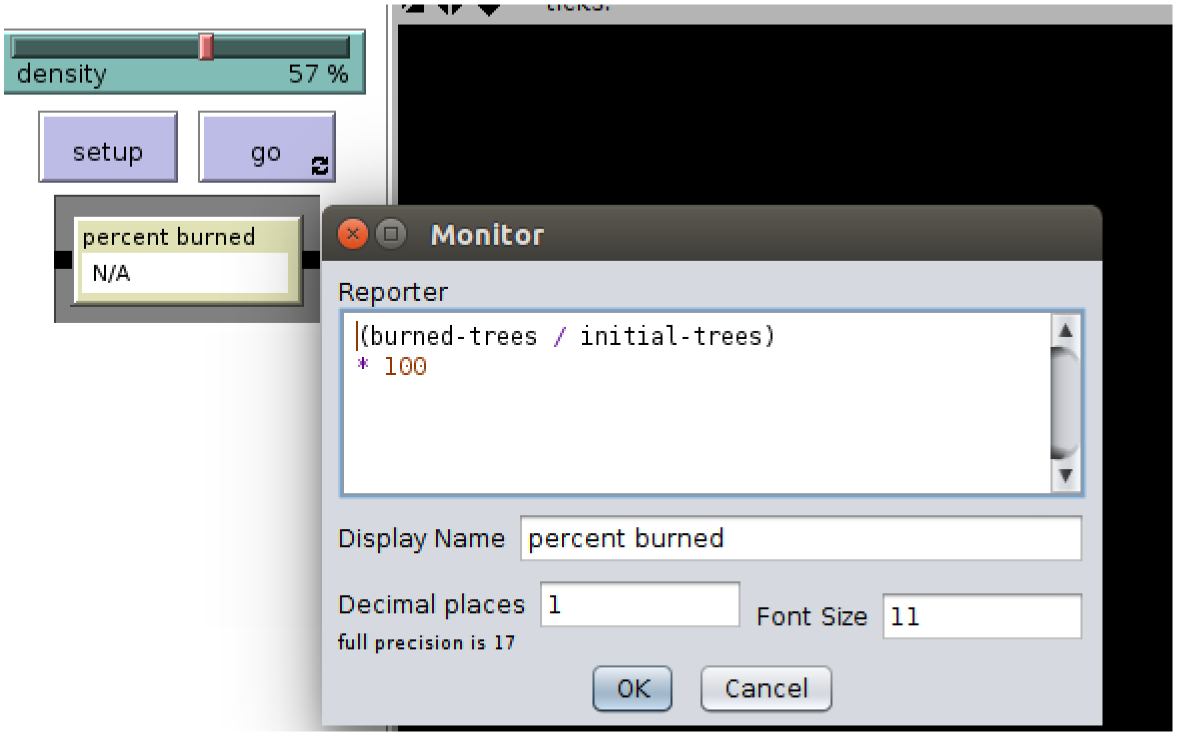
Duncan Golicher

25/01/2016

Using behaviour space

First load the fire model, found in the models library environmental folder. Play with the model to understand its behaviour. This is a simple model that only has one input parameter. This is density. We are going to evaluate the sensitivity of the model to this parameter.

The result that we are interested in is the percent of the landscape burned after the fire has run its course. As it stands the model reporter on the interface calculates this, but there is no named variable called percent-burned. We need to alter this. Edit the monitor box on the interface.



Reporter

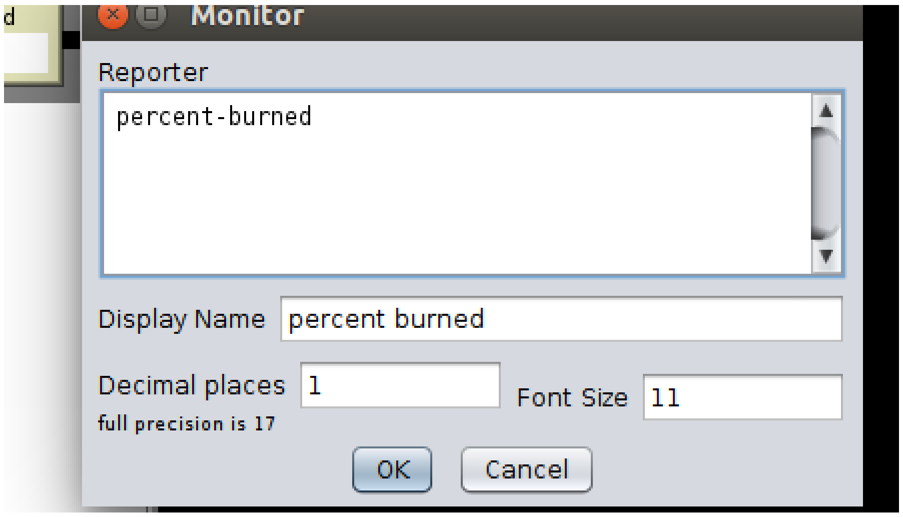
```
(burned-trees / initial-trees)
* 100
```

Display Name: percent burned

Decimal places: 1 (full precision is 17)

Font Size: 11

Remove this text and replace it by percent-burned.



Reporter

```
percent-burned
```

Display Name: percent burned

Decimal places: 1

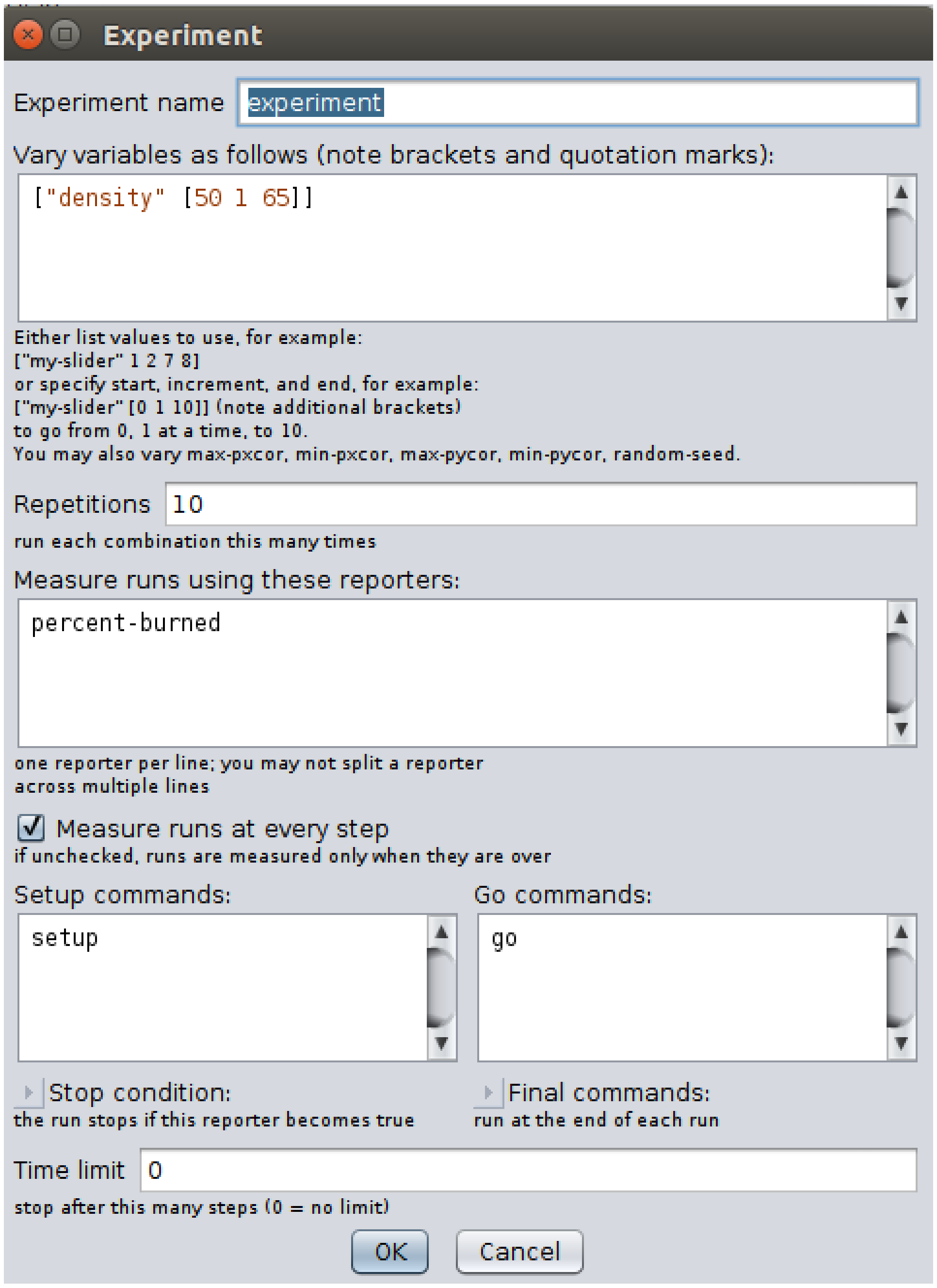
full precision is 17

Font Size: 11

Now you need to add a reporter within the code that carries out the same calculation.

```
to-report percent-burned
  report (burned-trees / initial-trees)
* 100
end
```

Now open the behaviour space tool and design an experiment. After playing with the model it becomes apparent that the critical range at which the landscape switches from being largely unburned to burned lies between densities of 50 and 65.


Experiment

Experiment name: experiment

Vary variables as follows (note brackets and quotation marks):

```
["density" [50 1 65]]
```

Either list values to use, for example:
["my-slider" 1 2 7 8]
or specify start, increment, and end, for example:
["my-slider" [0 1 10]] (note additional brackets)
to go from 0, 1 at a time, to 10.
You may also vary max-pxcor, min-pxcor, max-pycor, min-pycor, random-seed.

Repetitions: 10

run each combination this many times

Measure runs using these reporters:

```
percent-burned
```

one reporter per line; you may not split a reporter
across multiple lines

[✓] Measure runs at every step
if unchecked, runs are measured only when they are over

Setup commands:

```
setup
```

Go commands:

```
go
```

Stop condition:
the run stops if this reporter becomes true

Final commands:
run at the end of each run

Time limit: 0

stop after this many steps (0 = no limit)

Inspecting the results

The behaviour space tool produces two types of output file. The spreadsheet file attempts to summarise some of the results for you. The table output is much more useful if you intend to analyse the results in standard statistical software. For example, the results from this experiment can be turned into figures using R

We need two packages for this.

```r
library(ggplot2)
library(reshape)
```

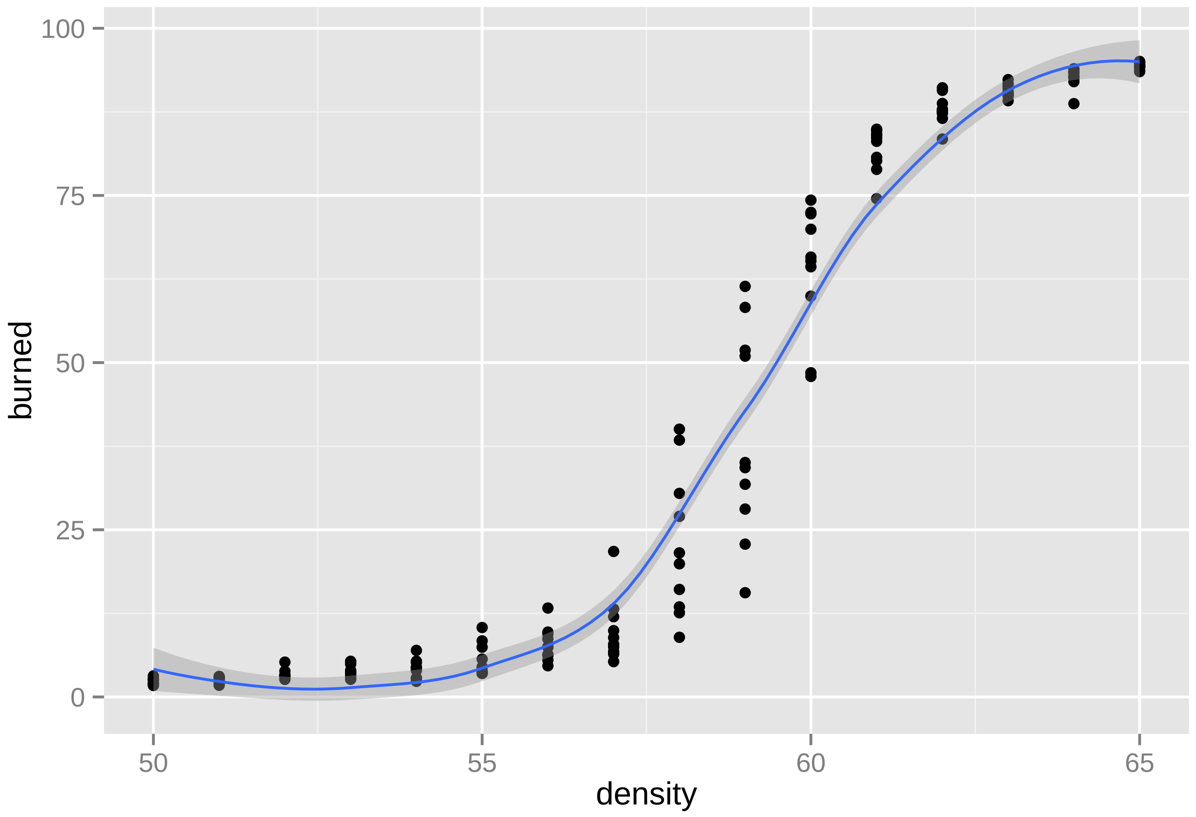
The file can be read in by skipping the first 6 lines. The variable names are too long and are best changed.

```r
d<-read.csv("Fire experiment-table.csv",skip=6)
names(d)<-c("run","density","step","burned")
```

As we are only really interested in the maximum ammount burned for each run we need to reshape the data.

```r
d1<-melt(d,m="burned")
d2<-cast(d1,run+density~variable,max)
```

```r
g0<-ggplot(data=d2,aes(y=burned,x=density))
g0+geom_point()+geom_smooth()
```

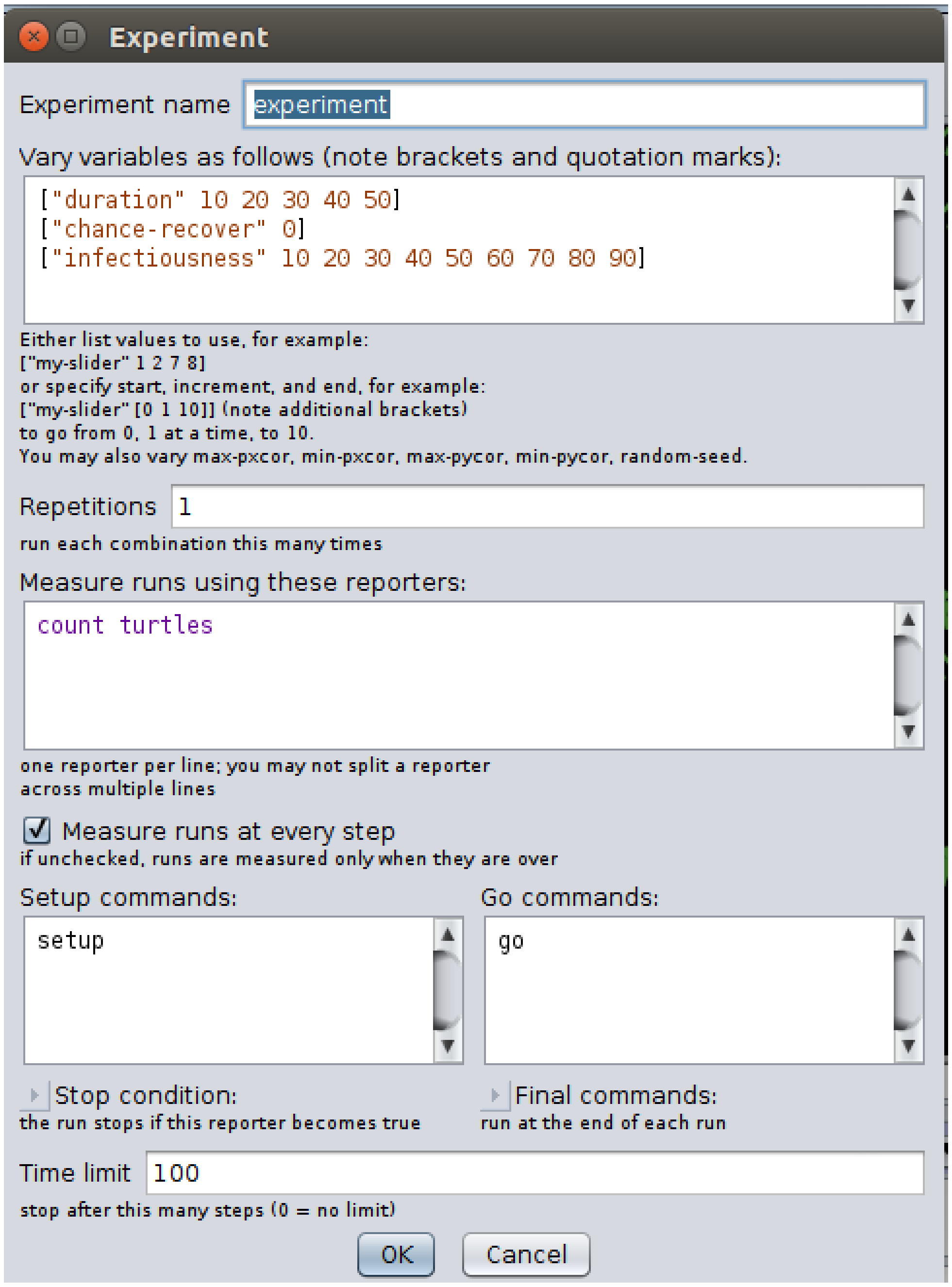


Varying two parameters

More complex experiments involve varying two or more parameters. Load the virus model and play with it to investigate its behaviour. Notice that in this model people can reproduce. This complicates the model too much as we may simply want to know how many people are left alive after a viral outbreak. So find the parameter for chance of reproduction in the setup-constants block of the code and set it to zero.

```
;; This sets up basic constants of the model.
to setup-constants
  set lifespan 50 * 52      ;; 50 times 52 weeks = 50 years = 2600 weeks old
  set carrying-capacity 300
  set chance-reproduce 0
  set immunity-duration 52
end
```

Now we can run some experiments. To simplify analysis we will only vary two parameters. The setting below allows the duration to vary between 10 and 50 and infectiousness to vary between 10 and 90 with a time limit of 100 months for the epidemic to run .



Experiment name experiment

Vary variables as follows (note brackets and quotation marks):

```
["duration" 10 20 30 40 50]
["chance-recover" 0]
["infectiousness" 10 20 30 40 50 60 70 80 90]
```

Either list values to use, for example:
["my-slider" 1 2 7 8]
or specify start, increment, and end, for example:
["my-slider" [0 1 10]] (note additional brackets)
to go from 0, 1 at a time, to 10.
You may also vary max-pxcor, min-pxcor, max-pycor, min-pycor, random-seed.

Repetitions 1

run each combination this many times

Measure runs using these reporters:

```
count turtles
```

one reporter per line; you may not split a reporter across multiple lines

[✓] Measure runs at every step

if unchecked, runs are measured only when they are over

Setup commands:

```
setup
```

Go commands:

```
go
```

Stop condition:

the run stops if this reporter becomes true

Final commands:

run at the end of each run

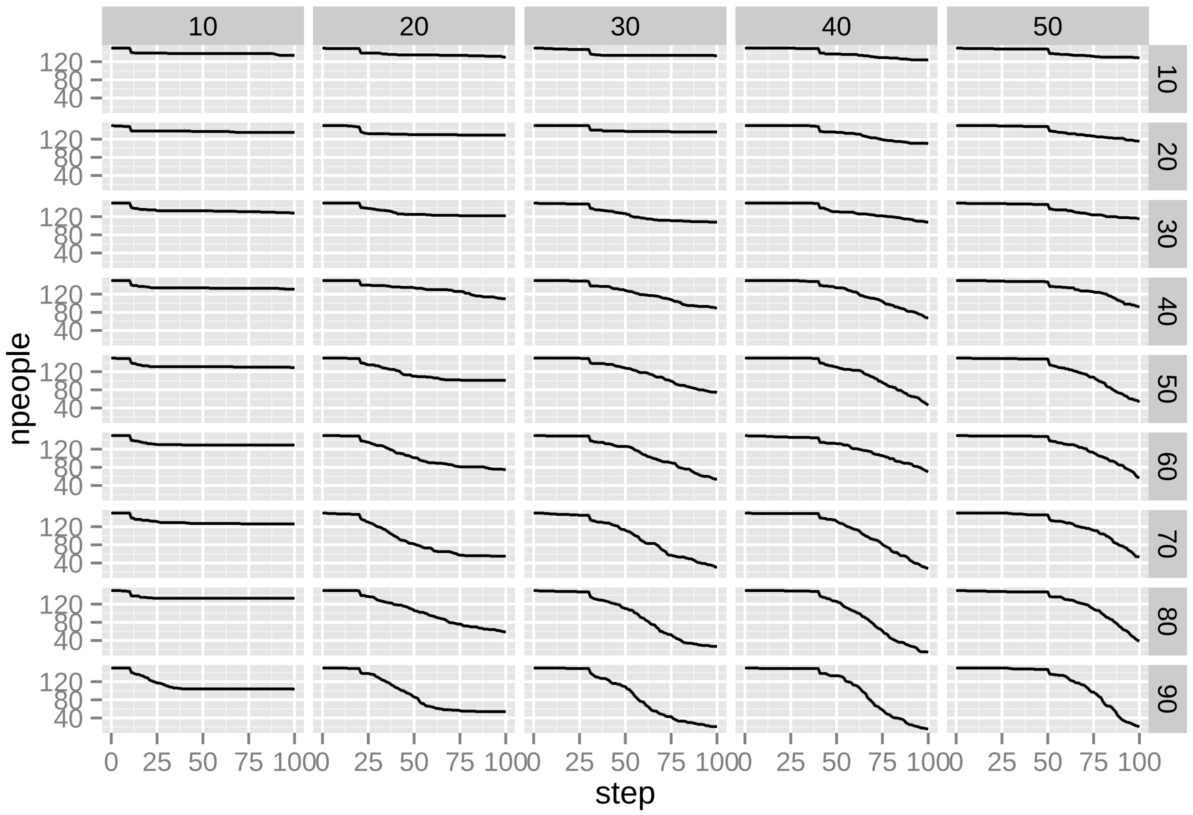
Time limit 100

stop after this many steps (0 = no limit)

```
d<-read.csv("Virus experiment-table.csv",skip=6)
names(d)<-c("run","duration","recover","infect","step","npeople")
```

We can look at each run using a facet grid.

```
g0<-ggplot(data=d,aes(x=step,y=npeople))
g0+geom_line()+facet_grid(infect~duration)
```

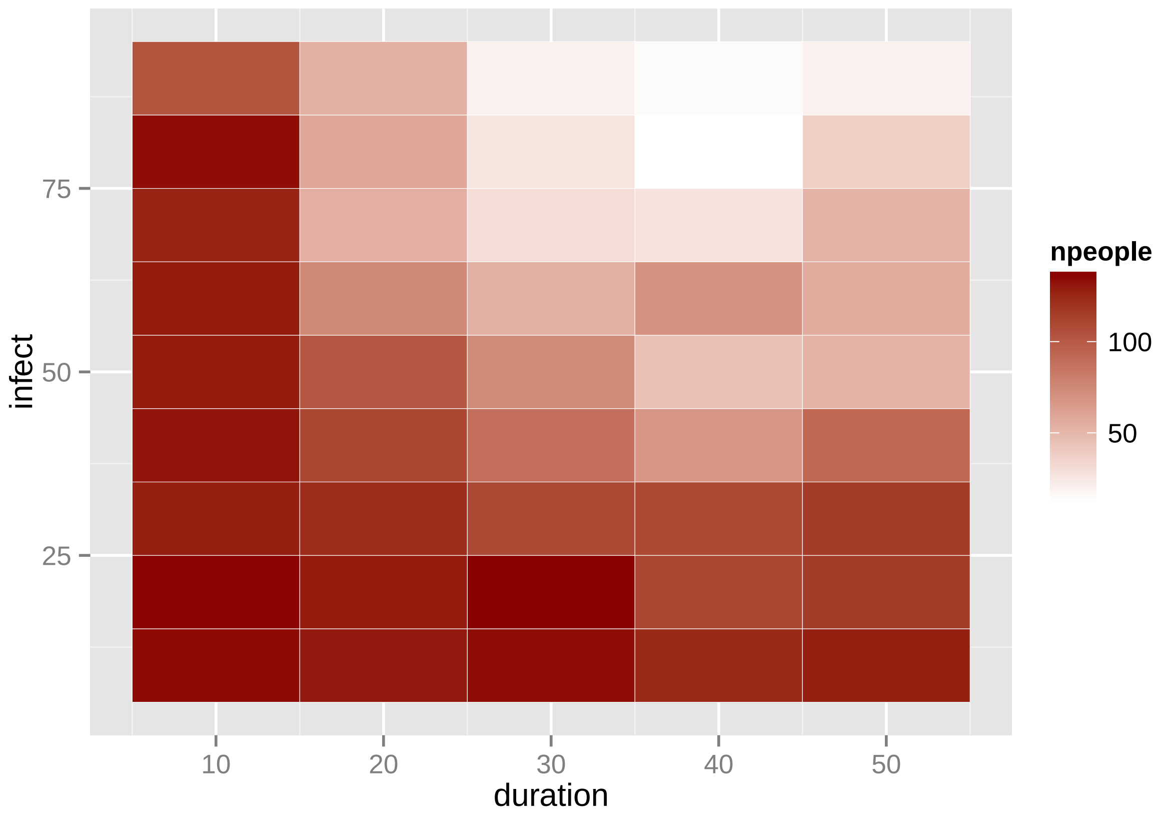


However, just as in the previous example, we are more interested in analysing just the results at the end of the model runs. We can do this again by reshaping the results. In this case we take the minimum values.

```
library(reshape)
d1<-melt(d,m="npeople")
d2<-cast(d1,duration+infect~variable,min)
```

One way of looking at the results in two dimensions is to use a heat map.

```
g0<- ggplot(d2, aes(duration, infect))
g1 <- g0 + geom_tile(aes(fill = npeople), colour = "white")
g1 + scale_fill_gradient(low = "white",high = "darkred")
```



Running netlog from R

Netlogo can be run directly from R. This can be useful for setting up more complex experiments, particularly if you want to use some of the functions in R to draw parameters from known distributions.

The first step is to start Netlogo and load the model. You need to provide the paths to these first.

```
library(RNetLogo)
nl.path <- "/home/duncan/Downloads/netlogo/app"
NLStart(nl.path,gui=FALSE)
model.path <- "/models/Sample Models/Earth Science/Fire.nlogo"
absolute.model.path <- paste(nl.path,model.path,sep="")
NLLoadModel(absolute.model.path)
```

Now you need to write a function to run the model and report the output. This involves running netlogo commands from R.

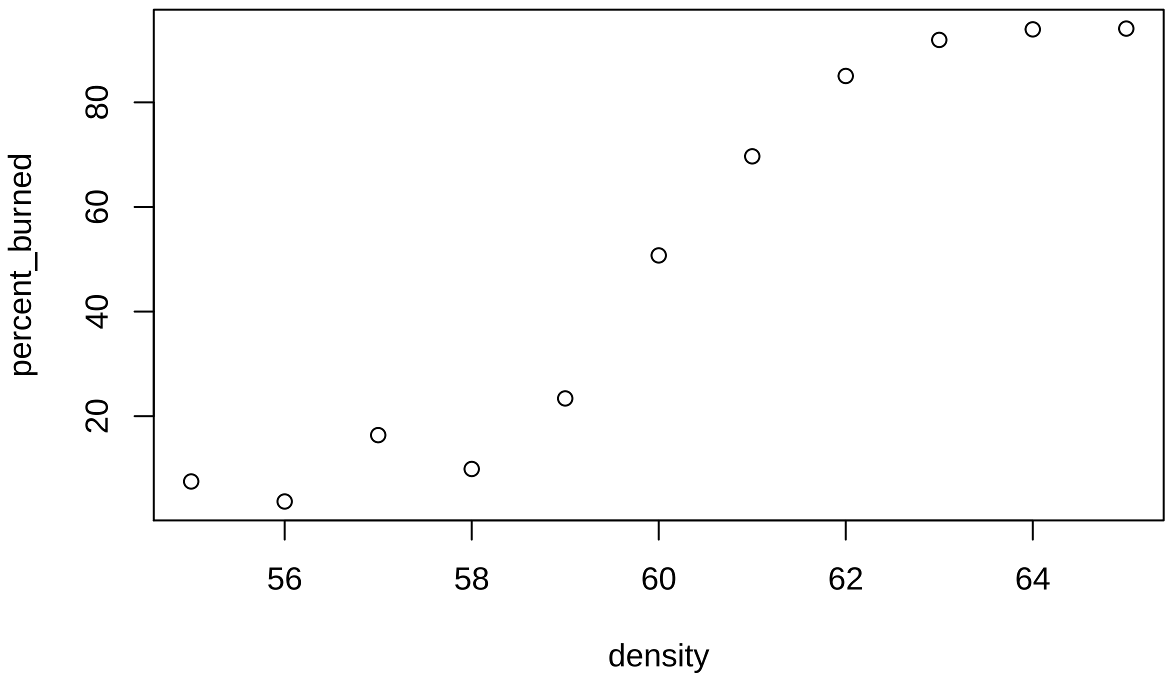
```
run_model<-function(density){
NLCommand("set density ", density)
NLCommand("setup")
NLDoCommand(500,"go")
NLReport("percent-burned")
}
```

Once you have a function you can apply it to a list of parameter values.

```
density<-55:65
system.time(percent_burned<-sapply(density,run_model))
```

```
##    user  system elapsed
##  27.415   0.458  20.221
```

```
d<-data.frame(density,percent_burned)
plot(d)
```



```
NLQuit()
```

Using parallel processing

You may have noticed that running netlogo from R was much slower than using the behaviour space tool. This is because by default there is no parallel processing occuring. Setting up parallel processing is slightly more complex but does speed up the time taken to run the models.

```
require(parallel)
```

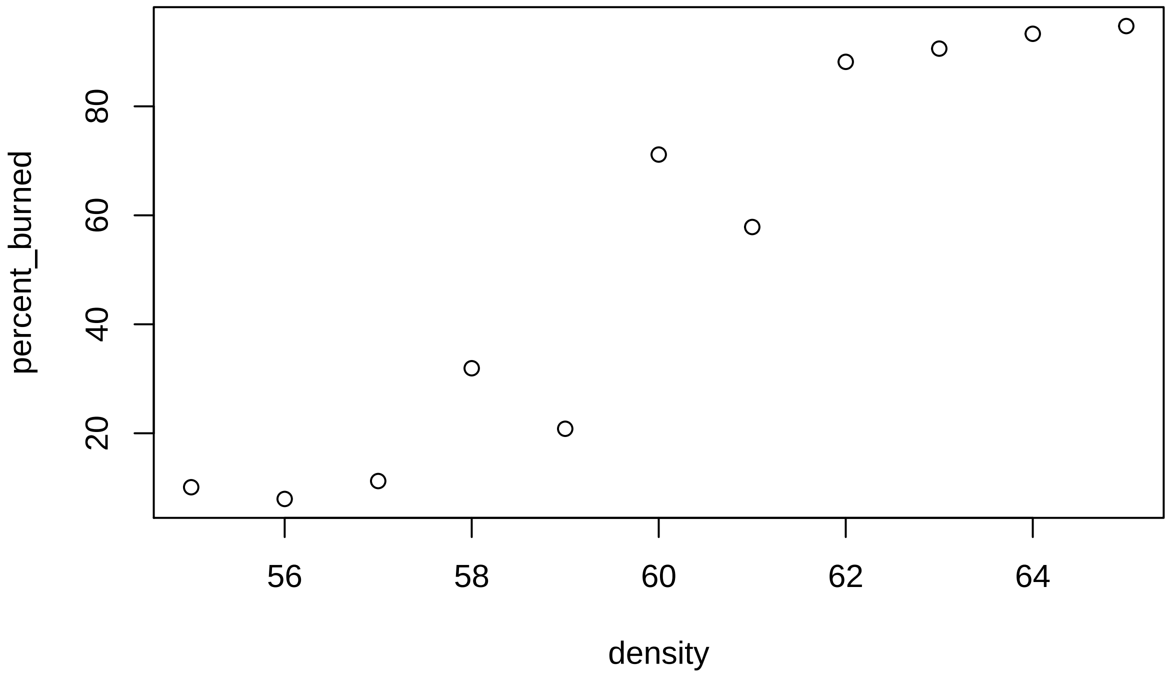
```
## Loading required package: parallel
```

```
# detect the number of cores available
processors <- detectCores()
# create cluster
cl <- makeCluster(processors)
# set variables for the start

prepro <- function(dummy, gui, nl.path, model.path) {
  library(RNetLogo)
NLStart(nl.path, gui=gui)
NLLoadModel(model.path)
}

invisible(parLapply(cl, 1:processors, prepro, gui=FALSE,
nl.path=nl.path, model.path=absolute.model.path))
```

```
run_model<-function(density){
NLCommand("set density ", density)
NLCommand("setup")
NLDoCommand(500,"go")
NLReport("percent-burned")
}

density<-55:65
system.time(percent_burned<-parSapply(cl,density,run_model))
```

```
##    user  system elapsed
##   0.009   0.000   9.490
```

```
d<-data.frame(density,percent_burned)
plot(d)
```



```
stopCluster(cl)
```