Appendix 1: How to use `ss3sim` for stock-assessment simulation

Sean C. Anderson and co-authors to be added

1 Installing the `ss3sim` `R` package

The package can be installed with:

```
# dependencies if needed:
install.packages(c("r4ss", "plyr", "gtools", "ggplot2", "lubridate",
  "reshape2"))
# install devtools to install ss3sim directly from GitHub:
install.packages("devtools")
devtools::install_github("ss3sim", "seananderson")
```

You can read the help files and access this vignette again with:

```
help(package = "ss3sim")
vignette("ss3sim")
```

`ss3sim` requires the `SS3` binary to be in your path. See the section "Putting `SS3` in your path" at the end of this document for details.

2 An example simulation with `ss3sim`

As an example, we will run a 2x2 simulation design in which we test (1) the effect of increasing or decreasing research survey effort and (2) the effect of fixing versus estimating natural mortality (M). All the files for this example are contained within the `ss3sim` package. So, to start, we'll locate three sets of folders: the folder with the plaintext case files, the folder with the operating model (OM), and the folder with the estimation model (EM).

```
library(ss3sim)
d <- system.file("extdata", package = "ss3sim")
case_folder <- paste0(d, "/eg-cases")
om <- paste0(d, "/models/cod-om")
em <- paste0(d, "/models/cod-em")
```

2.1 Setting up the file structure

The `ss3sim` package is set up assuming we have an established base-case OM and EM to work with. Each OM and EM should be in its own folder. The OM folder should have the files:

```
yourmodel.ctl
yourmodel.dat
ss3.par
starter.ss
forecast.ss
```

The EM folder should have:

```
yourmodel.ctl
yourmodel.dat # optional; not used
starter.ss
forecast.ss
```

In both cases, nothing more and nothing less. The names of the `.ctl` and `.dat` files are not important. The package functions will rename them after they are copied to appropriate folders. These files should be derived from the `.ss_new` files but named as listed above. It's important to use these `.ss_new` files so they have consistent formatting. Many of the functions in this package depend on that formatting.

2.2 Cases and scenarios

To use the high-level wrapper function `run_ss3sim`, we will have unique case identifiers that combine to create unique scenarios. The types of cases are: natural mortality (M), fishing mortality (F), data quality (D), selectivity (S), growth (G), and retrospective (R). These case IDs are followed by three-letter species identifier. It's important to use three letters since the functions assume that the last three letters represent a species (or some other identifier for a different project).

The different version of each case are identified with numbers. So, for example, the base-case scenario for our cod stock might be: `D0-E0-F0-G0-M0-R0-S0-cod`. The order of the letters doesn't matter, as long as we use them consistently.

2.3 Output file structure

The function `copy_ss3models` creates a folder structure and copies over the operating and estimation models. The folder structure looks like:

```
D0-E0-F0-G0-M0-R0-S0-cod/1/om
D0-E0-F0-G0-M0-R0-S0-cod/1/em
D0-E0-F0-G0-M0-R0-S0-cod/2/om
D0-E0-F0-G0-M0-R0-S0-cod/2/em
...
```

If you are using bias adjustment (`bias_adjust = TRUE`) then there will be some additional folders. In that case the folders will look like:

```
D0-E0-F0-G0-M0-R0-S0-cod/bias/1/om
D0-E0-F0-G0-M0-R0-S0-cod/bias/1/em
D0-E0-F0-G0-M0-R0-S0-cod/bias/2/om
D0-E0-F0-G0-M0-R0-S0-cod/bias/2/em
...
D0-E0-F0-G0-M0-R0-S0-cod/1/om
D0-E0-F0-G0-M0-R0-S0-cod/1/em
D0-E0-F0-G0-M0-R0-S0-cod/2/om
D0-E0-F0-G0-M0-R0-S0-cod/2/em
...
```

Note that the operating and estimation model folders have been renamed `om` and `em` within each iteration, the operating and estimation models have been checked to make sure they contain the minimal files (as listed above), the filenames have been made all lowercase, the data file has been renamed `data.dat`, the control files have been renamed `om.ctl` or `em.ctl`, and the starter and control files have been adjusted to reflect these new file names.

The functions in this package assume you've set your working directory in `R` to be the base folder where you will store the scenario folders.

2.4 Creating the input configuration files

We will need to have a folder containing "case" argument definitions. These plain text files are read by `get_caseval` and turned into argument lists that are passed to `run_ss3sim`. We can create template input files by running `create_argfiles`. It reads the various functions and parses the arguments and default values into plain text files. The default settings create these files:

1. `E0-spp.txt`
2. `F0-spp.txt`
3. `G0-spp.txt`
4. `M0-spp.txt`
5. `R0-spp.txt`
6. `S0-spp.txt`

7. `index0-spp.txt` (controlled by the `D` case)
8. `agecomp0-spp.txt` (controlled by the `D` case)
9. `lcomp0-spp.txt` (controlled by the `D` case)

Look in your working directory for the template files. Change the case ID number (defaults to `0`) and the species identifier to a three letter identifier. For example, `cod`, `sar`, or `fla` for cod, sardine, or flatfish. An example filename would therefore be `M1-sar.txt` or `lcomp2-fla.txt`. The case `D1` corresponds to the files `index1-spp.txt`, `agecomp1-spp.txt`, and `lcomp0-spp.txt`. The other case types have single argument files.

The first column in the text files denotes the argument to be passed to a function. The second argument denotes the value to be passed. See the help for a `change` function to see the arguments that need to be declared. For example, see `?change_f`.

You can use any simple `R` syntax to declare the argument values. For example: `c(1, 2, 4)`, or `seq(1, 100)` or `1:100` or `matrix()`. Character objects don't need to be quoted, but can be if you'd like. However, be careful not to use the delimiter (set up as a semicolon) anywhere else in the file besides to denote columns. You can add comments after any `#` symbols. Internally, the functions evaluate in `R` any entries that have no character values (e.g. `1:100`) or have an alpha-numeric character followed by a `(`. Anything that is character only or has character mixed with numeric but doesn't have the regular expression `"[A-Za-z0-9]("` gets turned into a character argument.

Putting that all together, here's what an example `F1-cod.txt` file might look like:

```
years; 1913:2012
years_alter; 1913:2012
fvals; c(rep(0,25), rep(0.114,75))
```

In our case, we will base the simulation around the base-case files used for the cod stock in the papers (cite our papers here). You can refer to these papers for details on how the models were set up.

To investigate the effect of increasing or decreasing sampling effort, we will manipulate argument `sd_obs_surv` that gets passed to `change_index()`. In case 1, we'll halve the standard deviation to 0.1 and in case 2 we'll double the standard deviation to 0.4.

We can do this by including the line:

```
sd_obs_surv; 0.1
```

in the file `D1.txt` and the line:

```
sd_obs_surv; 0.4
```

in the file `D2.txt`.

To investigate the effect of fixing versus estimating M, we'll manipulate the argument `natM_val` that gets passed to `change_e()`. In case 0, we'll set the phase that M gets estimated in to `NA`. In case 1, we'll set the phase that M gets estimated in to 3.

We can do this by including the line:

```
natM_val; c(NA,NA)
```

in the file `E0.txt` and the line:

```
natM_val; c(NA,3)
```

in the file `E1.txt`.

2.5 Running deterministic simulations to check the models for bias

We'll run some "deterministic" runs to check our model for bias when we don't have any process error. To do this, we'll start by setting up a matrix of recruitment deviations with 0 deviations. We need 100 rows (for 100 year simulations) and 20 columns (for 20 deterministic iterations).

```
recdevs_det <- matrix(0, nrow = 100, ncol = 20)
```

Then we'll set up case "estimation" files in which the recruitment deviations are set to the nominal level of `0.001`. We'll name these files `E100-cod.txt` and `E101-cod.txt`. In the control files, the key element is setting `par_name = SR_sigmaR` and `par_int = 0.001`.

When we run the simulations, we'll pass our deterministic recruitment deviations to the function `run_ss3sim`. Running 20 replicates should be enough to identify whether our models are performing as we expect.

```
run_ss3sim(iterations = 1:20, scenarios =
  c("D1-E100-F0-G0-R0-S0-M0-cod",
    "D2-E100-F0-G0-R0-S0-M0-cod",
    "D1-E101-F0-G0-R0-S0-M0-cod",
    "D2-E101-F0-G0-R0-S0-M0-cod"),
  case_folder = case_folder, om_model_dir = om, em_model_dir = em,
  bias_adjust = TRUE, user_recdevs = recdevs_det)
```

We have written out the scenario names in full for clarity, but `ss3sim` also contains a convenience function `expand_scenarios`. With this function we could have instead written:

```
x <- expand_scenarios(e = c(100, 101), d = c(1, 2), species = "cod")
run_ss3sim(iterations = 1:20, scenarios = x,
  case_folder = case_folder, om_model_dir = om, em_model_dir = em,
  bias_adjust = TRUE, user_recdevs = recdevs_det)
```

2.6 Running the stochastic simulations

Now we can run the stochastic simulations.

```
run_ss3sim(iterations = 1:100, scenarios =
  c("D1-E0-F0-G0-R0-S0-M0-cod",
    "D2-E0-F0-G0-R0-S0-M0-cod",
    "D1-E1-F0-G0-R0-S0-M0-cod",
    "D2-E1-F0-G0-R0-S0-M0-cod"),
  case_folder = case_folder, om_model_dir = om, em_model_dir = em,
  bias_adjust = TRUE)
```

The function `get_results_all` reads in a set of scenarios and combines the output into two `.csv` files: `final_results_scalar.csv` and `final_results_ts.csv`.

```
get_results_all(user.scenarios =
  c("D1-E100-F0-G0-R0-S0-M0-cod",
    "D2-E100-F0-G0-R0-S0-M0-cod",
    "D1-E101-F0-G0-R0-S0-M0-cod",
    "D2-E101-F0-G0-R0-S0-M0-cod",
    "D1-E0-F0-G0-R0-S0-M0-cod",
    "D2-E0-F0-G0-R0-S0-M0-cod",
    "D1-E1-F0-G0-R0-S0-M0-cod",
    "D2-E1-F0-G0-R0-S0-M0-cod"))
```

Let's read in the `.csv` files:

```
scalar_dat <- read.csv("final_results_scalar.csv")
ts_dat <- read.csv("final_results_ts.csv")
```

And calculate some useful values in new columns:

```r
scalar_dat <- transform(scalar_dat,
  SSB_MSY=(SSB_MSY_em-SSB_MSY_om)/SSB_MSY_om,
  log_max_grad = log(max_grad))

scalar_dat <- transform(scalar_dat,
  steep = (SR_BH_steep_om - SR_BH_steep_em)/SR_BH_steep_om,
  logR0 = (SR_LN_R0_om - SR_LN_R0_em)/SR_LN_R0_om,
  depletion = (depletion_om - depletion_em)/depletion_om,
  SSB_MSY = (SSB_MSY_em - SSB_MSY_om)/SSB_MSY_om,
  SR_sigmaR = (SR_sigmaR_em - SR_sigmaR_om)/SR_sigmaR_om,
  NatM = (NatM_p_1_Fem_GP_1_em - NatM_p_1_Fem_GP_1_om)/NatM_p_1_Fem_GP_1_om)

ts_dat <- transform(ts_dat,
  SpawnBio = (SpawnBio_em - SpawnBio_om)/SpawnBio_om,
  Recruit_0 = (Recruit_0_em - Recruit_0_om)/Recruit_0_om)
ts_dat <- merge(ts_dat, scalar_dat[,c("scenario", "replicate",
    "max_grad")])

scalar_dat_det <- subset(scalar_dat, E %in% c("E100", "E101"))
scalar_dat_sto <- subset(scalar_dat, E %in% c("E0", "E1"))
ts_dat_det <- subset(ts_dat, E %in% c("E100", "E101"))
ts_dat_sto <- subset(ts_dat, E %in% c("E0", "E1"))
```

Now we'll turn the scalar data into long-data format so we can make a multipanel plot with ggplot2.

```r
scalar_dat_long <- reshape2::melt(scalar_dat[,c("scenario", "D", "E",
  "replicate", "max_grad", "steep", "logR0", "depletion", "SSB_MSY",
  "SR_sigmaR", "NatM")], id.vars = c("scenario", "D", "E", "replicate",
  "max_grad"))
scalar_dat_long <- plyr::rename(scalar_dat_long, c("value" = "relative_error"))
```

Now let's look at boxplots of the deterministic model runs.

```r
library(ggplot2)
p <- ggplot(subset(scalar_dat_long, E %in% c("E100", "E101") &
       variable != "SR_sigmaR"), aes(D, relative_error)) +
     geom_boxplot() +
     geom_hline(aes(yintercept = 0), lty = 2) +
     facet_grid(variable~E) +
     geom_jitter(aes(colour = max_grad),
     position = position_jitter(height = 0, width = 0.1),
       alpha = 0.4, size = 1.5) +
     scale_color_gradient(low = "darkgrey", high = "red") +
     theme_bw()
print(p)
```

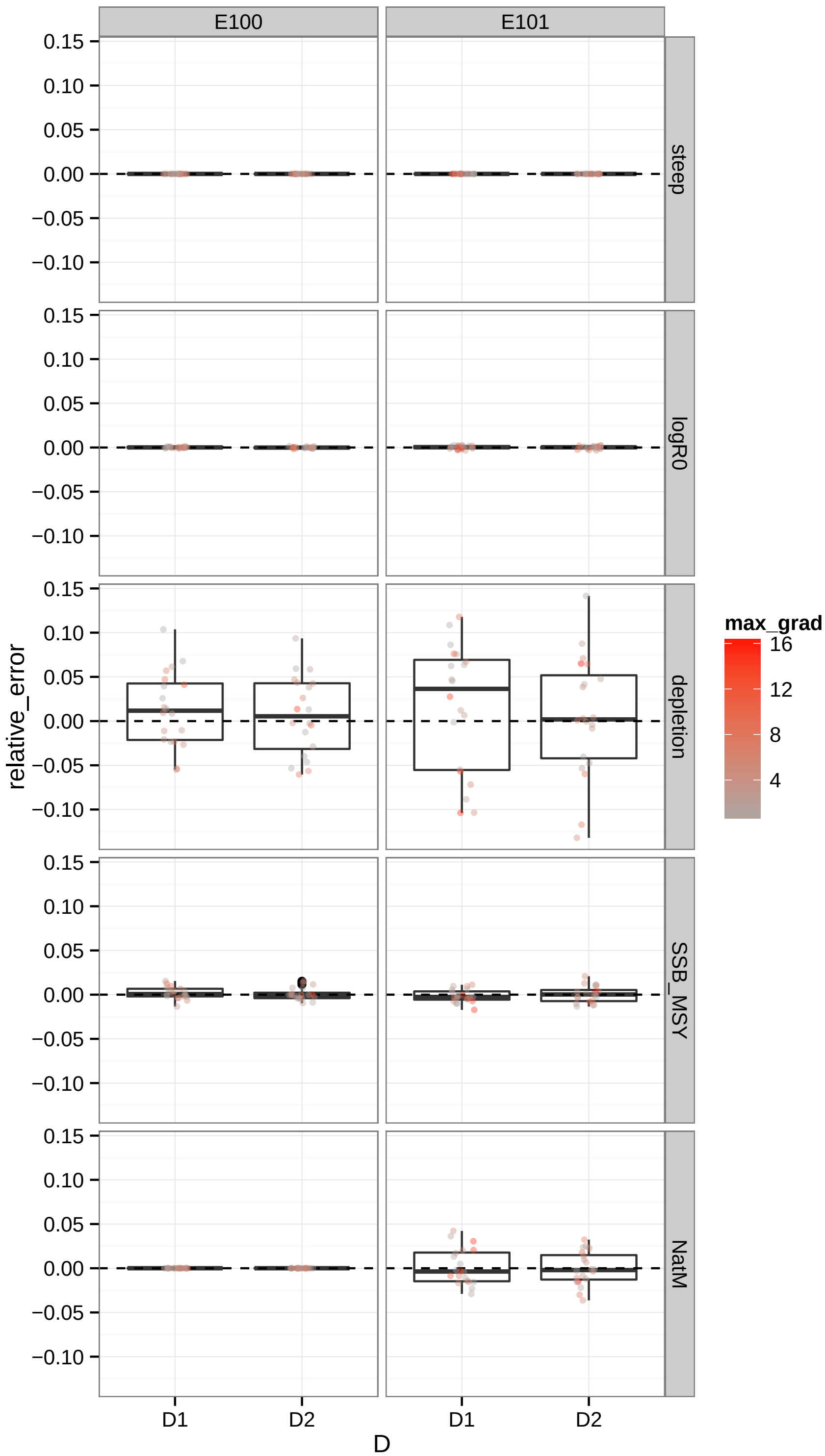

Figure 1: Relative error box plots for deterministic runs. In case E0, M is fixed at the historical value; in E1 we estimate M. In case D2, the standard deviation on the survey index observation error is 0.4. In case D1, the standard deviation is quartered representing an increase in survey sampling effort.

Let's look at the relative error in estimates of spawning biomass. We'll colour the time series according to the maximum gradient. Small values of the maximum gradient (approximately 0.001 or less) indicate that convergence is likely. Larger values (greater than 1) indicate that convergence is unlikely.

```
plot_ts_points(ts_dat_sto, y = "SpawnBio", vert = "D", vert2 = "E", color =
  "max_grad", relative_error = TRUE)
```

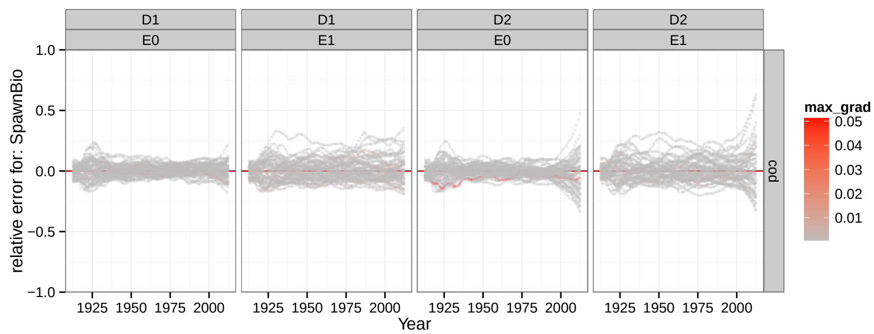

Figure 2: Time series of relative error in spawning stock biomass.

```
p <- ggplot(ts_dat_sto, aes(year, SpawnBio_em, group = replicate)) +
  geom_line(alpha = 0.3, aes(colour = max_grad)) + facet_grid(D~E) +
  scale_color_gradient(low = "darkgrey", high = "red") + theme_bw()
print(p)
```

```
p <- ggplot(subset(scalar_dat_long, E %in% c("E0", "E1")),
      aes(D, relative_error)) +
    geom_boxplot() + geom_hline(aes(yintercept = 0), lty = 2) +
    facet_grid(variable~E) +
    geom_jitter(aes(colour = max_grad),
      position = position_jitter(height = 0, width = 0.1),
      alpha = 0.4, size = 1.5) +
    scale_color_gradient(low = "darkgrey", high = "red") +
    theme_bw()
print(p)
```

3 Putting SS3 in your path

`SS3` must be in your path for the `ss3sim` package to work. Your "path" is a list of folders that your operating system looks in whenever you type the name

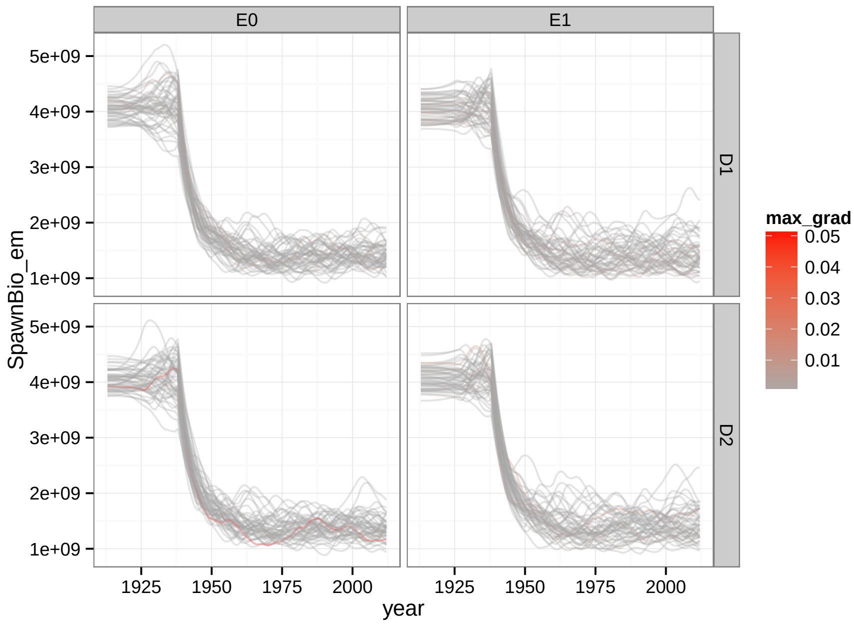

Figure 3: Spawning stock biomass time series.

of a program on the command line. Having a binary in your path means that your operating system knows where to look for the file regardless of what folder you're working in.

3.1 For Unix (Linux and OS X)

To check if `SS3` is in your path: open a Terminal window and type `which SS3` and hit enter. If you get nothing returned then SS is not in your path. The easiest way to fix this is to move the `SS3` binary to a folder that's already in your path. To find existing path folders type `echo $PATH` in the terminal and hit enter. Now move the `SS3` binary to one of these folders. For example, in a Terminal window type:

```
sudo cp ~/Downloads/SS3 /usr/bin/
```

You will need to use `sudo` and enter your password after to have permission to move a file to a folder like `/usr/bin/`.

If you've previously modified your path to add a non-standard location for the `SS3` binary, you may need to also tell `R` about the new path. The path that `R` sees may not include additional paths that you've added through a configuration

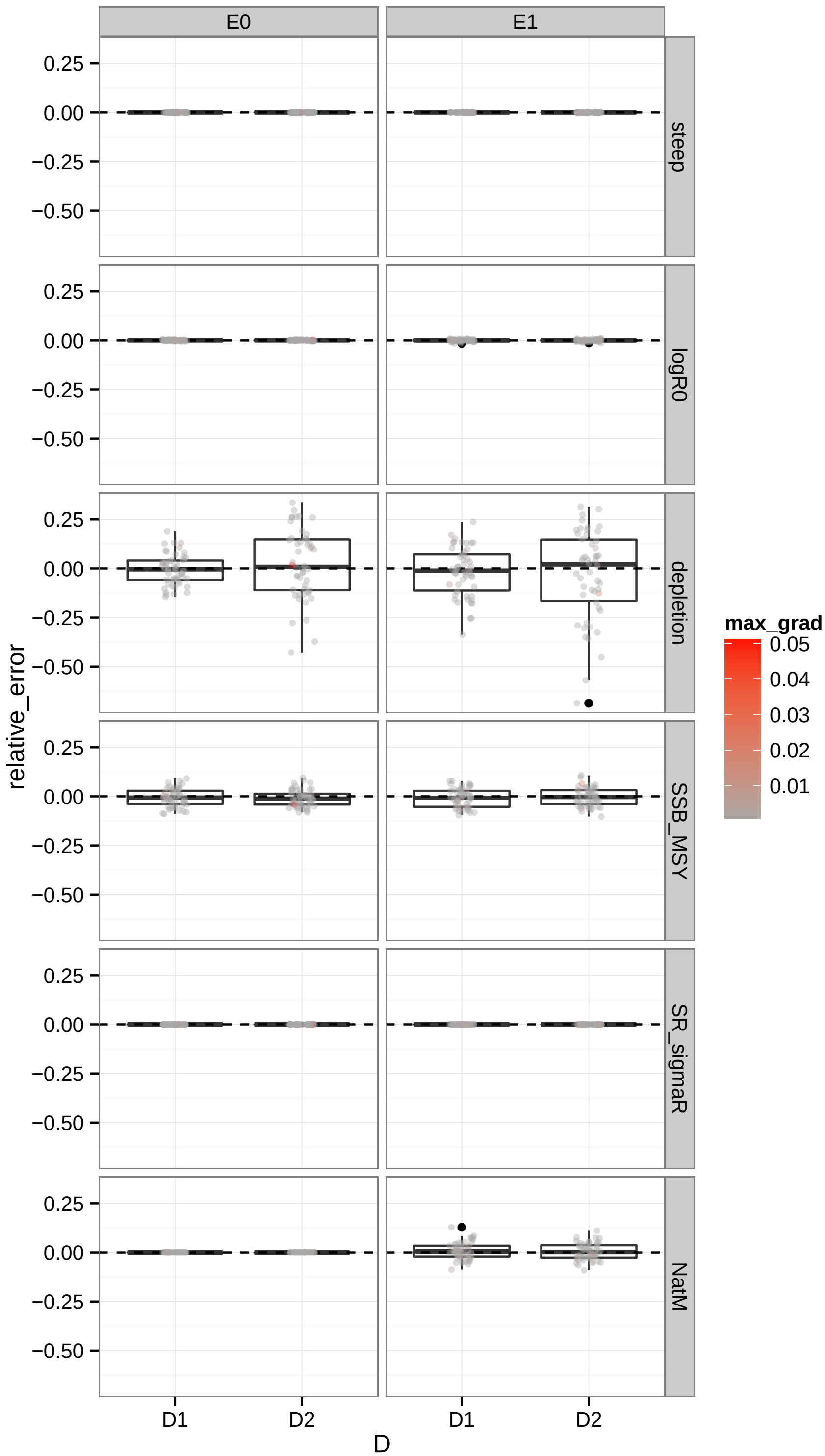

Figure 4: Relative error box plots for stochastic runs. In case E0, M is fixed at the historical value; in E1 we estimate M. In case D2, the standard deviation on the survey index observation error is 0.4. In case D1, the standard deviation is quartered representing an increase in survey sampling effort.

file like `.bash_profile`. You can add to the path that `R` sees by including a line like this in your `.Rprofile` file. (This is an invisible file in your home directory.)

```
Sys.setenv(PATH=paste(Sys.getenv("PATH"),"/my/folder",sep=":"))
```

3.2 For Windows

To check if SS is in your path: open a DOS prompt and type `ss3 -?` and hit enter. If you get a line like "ss3 is not recognized ... " then SS is not in your path. To add it to your path:

1. Find the latest version of the `ss3.exe` binary on your computer
2. Record the folder location. E.g. `C:/SS3.24o/`
3. Click on the start menu and type "environment"
4. Choose "Edit environment variables for your account" under Control Panel
5. Click on `PATH` if it exists, create it if doesn't exist
6. Choose `PATH` and click edit
7. In the "Edit User Variable" window add to the **end** of the "Variable value" section a semicolon and the `SS3` folder location you recorded earlier. E.g. `;C:/SS3.24o/`
8. Restart your computer
9. Go back to the DOS prompt and try typing `ss3 -?` and hitting return again.