Simulate from a Fitted TMB Model

This example contains a simple linear model with one fixed effect. First we simulate data for the process to be modeled. Then we show how the model is fit in TMB. Then we show how to simulate observations from the fitted model. This is useful to check that the structure of the fitted model is a good representation of the process that created the data. To illustrate this, we finally fit a model that is a bad representation of the underlying process and show that the simulated data from that fitted model does not look like the observed data.

Simulate observations from the true underlying process

Assume that we observe values of `y` in 100 replicates, half the replicates are controls `trt=0` and the other half were treated in some way `trt=1`. The baseline mean of the control replicates is `mu` and the treatment effect (`trt_eff`) adds to that mean.

```r
library(TMB)
set.seed(1)
mu=5 #mean of the control group
trt_eff=10 #treatment effect
resid_sd=1 #residual error
nreps=50 #number of replicates per treatment
obs0=expand.grid(rep=factor(1:nreps), trt=c(0,1)) #set up data structure
obs=transform(obs0, y=mu+trt_eff*trt) #predict mean observed value of y
obs$y=obs$y+rnorm(nrow(obs),0,resid_sd) #add residual error to y
X=model.matrix(~trt, data=obs) #fixed effects design matrix
```

Fit a model that matches the underlying process

The following C++ code to fit this model is stored in `FE.cpp`. We will be able to later use this model to simulate values of `y` because of lines 16 to 18 and lines 20 to 22. These two chunks of code are wrapped in by `SIMULATE{}` which tells TMB to only do the enclosed commands when it's in simulation mode i.e., when we tell it to simulate values from the fitted model. When we are simulating, line 17 will produce a random number from a normal distribution with mean `Xbeta(i)` and standard deviation `resid_sd`. This is done using the function `rnorm` which was written in TMB to closely resemble `rnorm` in R; the difference is that `rnorm` in R takes `n`, how many random numbers to generate, as its first argument. Then, when we are simulating, line 21 will return the simulated values in `y` as part of a list.

```cpp
1  #include <TMB.hpp>
2  template<class Type>
3  Type objective_function<Type>::operator() ()
4  {
5    DATA_VECTOR(y);
6    DATA_MATRIX(X);
7    PARAMETER(log_resid_sd);
8    Type resid_sd=exp(log_resid_sd);
9    ADREPORT(resid_sd);
10   PARAMETER_VECTOR(beta);
11   Type nll;
12   vector<Type> Xbeta= X*beta;
13   for(int i=0; i<y.size(); i++)
14    {
15      nll -= dnorm(y(i), Xbeta(i), resid_sd, true);
16      SIMULATE {
17        y(i) = rnorm(Xbeta(i), resid_sd);
18      }
19    }
20    SIMULATE {
21      REPORT(y);
22    }
23    return nll;
24  }
```

Both `dnorm` and `rnorm` are vectorized, so we could replace lines 13 to 19 of the code above with the following code. In the code below, `rnorm` returns random numbers with the same dimension as `Xbeta`, a vector of length 100. To do this in R, we would have needed to specify the length of the vector to be returned.

```
  nll = -sum(dnorm(y, Xbeta, resid_sd, true));
  SIMULATE {
    y = rnorm(Xbeta, resid_sd);
  }
```

We can compile and fit the model in the standard way.

```
compile("FE.cpp")
```

```
## Note: Using Makevars in /Users/molliebrooks1/.R/Makevars
```

```
## [1] 0
```

```
dyn.load(dynlib("FE"))
obj=MakeADFun(data=list(y=obs$y, X=X),
            parameters=list(log_resid_sd=1, beta=rep(0, ncol(X))),
            DLL="FE",
            silent = TRUE)
opt = nlminb(obj $par, obj $fn, obj $gr)
sdr=sdreport(obj)
sdr
```

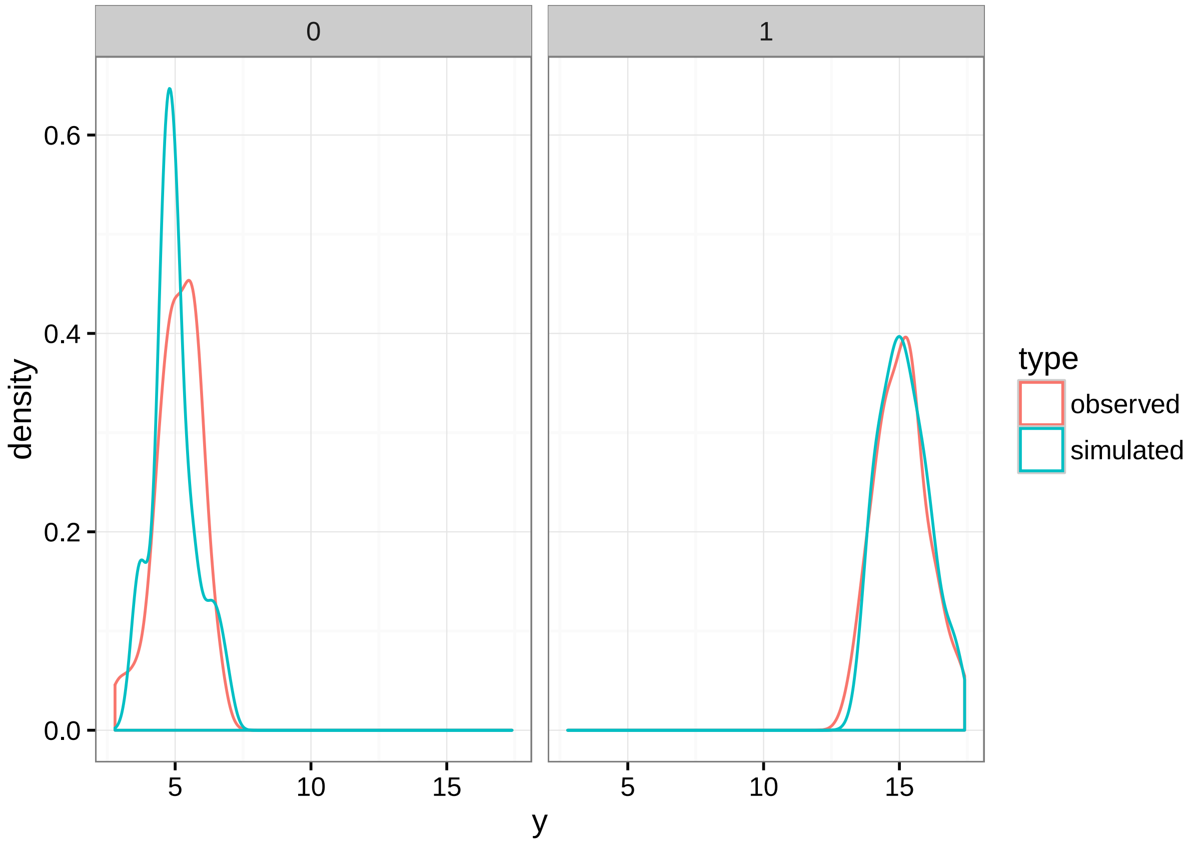
```
## sdreport(.) result
##                 Estimate Std. Error
## log_resid_sd -0.1124329 0.07071066
## beta          5.1004482 0.12638223
## beta         10.0168783 0.17873146
## Maximum gradient component: 1.197141e-05
```

Compare simulated values from the fitted model to the observed data

We can generate simulated values of `y` using the `simulate()` function which is now part of our model object `obj`. Calling `obj$simulate()` returns a list which would contain mutliple elements if we were simulating multiple objects; in this case, it only contains `y`.

```
ysim=obj$simulate()$y
```

Check that the observed and simulated values of y have similar distributions by plotting them together, separated by treatment.



The values simulated from the fitted model are very close to the observations because the structure of our fitted model perfectly matched the process that created the observations.

A model that does not represent the true underlying process

In this section we show what happens when we fit a model that is not a good representation of the process that generated the data. This second version of the model ignores the treatment effect and instead fits a global mean parameter `mu`. The following C++ code which is stored in `FE0.cpp`. In this version of the code, the predicted mean is a scalar, so we must use a for-loop (lines 11 to 17) to simulate repeated draws from the same distribution. This is one instance where it could be useful to have an arguement of `rnorm` specifying the number of repeated draws, but these instances are rare.

```
1 #include <TMB.hpp>
2 template<class Type>
3 Type objective_function<Type>::operator() ()
4 {
5   DATA_VECTOR(y);
6   PARAMETER(mu);
7   PARAMETER(log_resid_sd);
8   Type resid_sd=exp(log_resid_sd);
9   ADREPORT(resid_sd);
```

```
  Type nll;
  for(int i=0; i<y.size(); i++)
  {
    nll -= dnorm(y(i), mu, resid_sd, true);
    SIMULATE {
      y(i) = rnorm(mu, resid_sd);
    }
  }

  SIMULATE {
    REPORT(y);
  }
  return nll;
}
```

Then we fit `FE0.cpp` with the following R code and create simulated values from the fitted model.

```
compile("FE0.cpp")
```

```
## Note: Using Makevars in /Users/molliebrooks1/.R/Makevars
```

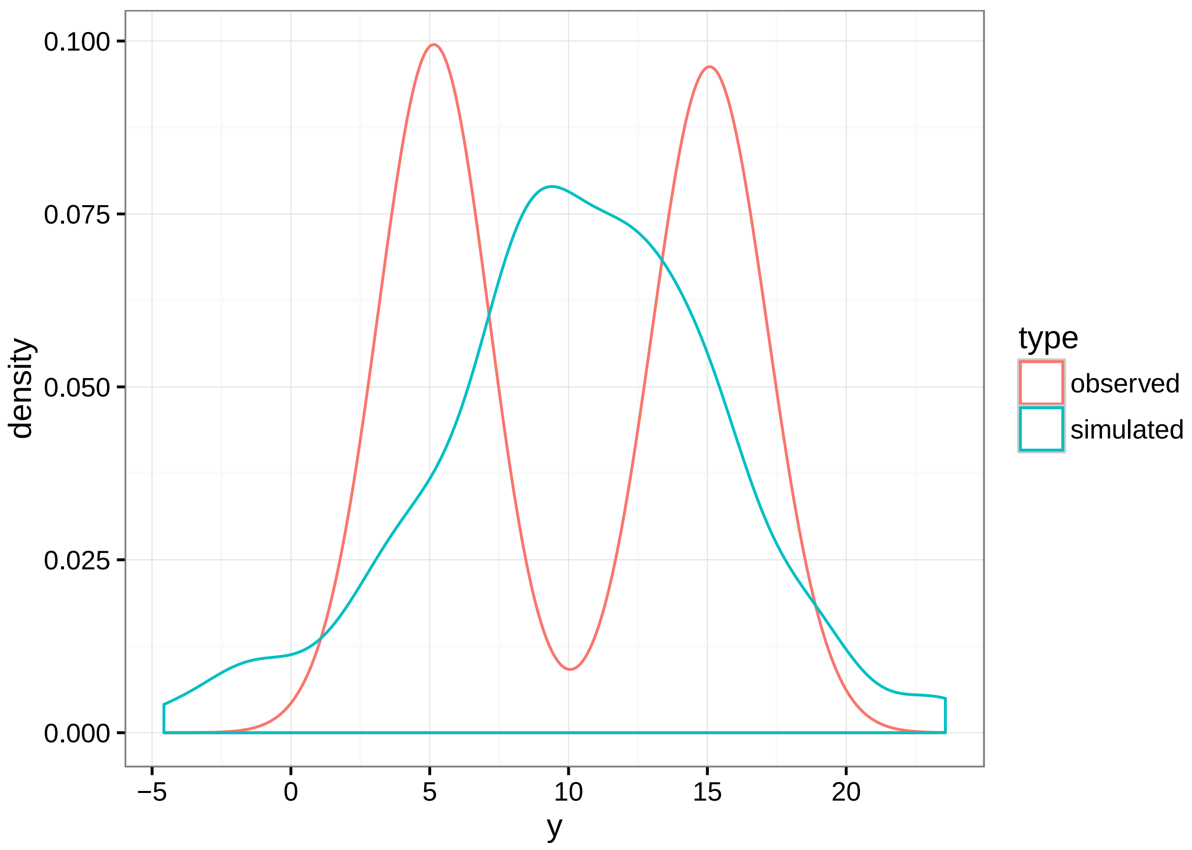
```
## [1] 0
```

```
dyn.load(dynlib("FE0"))
obj0=MakeADFun(data=list(y=obs$y),
            parameters=list(log_resid_sd=1, mu=0),
            DLL="FE0",
            silent = TRUE)
opt0 = nlminb(obj0 $par, obj0 $fn, obj0 $gr)
sdr0=sdreport(obj0)
sdr0
```

```
## sdreport(.) result
##               Estimate Std. Error
## mu           10.108887 0.50875421
## log_resid_sd  1.626795 0.07071065
## Maximum gradient component: 2.618796e-06
```

```
ysim0=obj0$simulate()$y #simulated values from the fitted model
```

Based on the `sdreport`, it looks like the model fit fine. Next we can examine the simulated data as if we were oblivious to the treatment.



Simulations from the fitted model do not have the bimodal distribution that the observations have. This indicates that the model structure is missing something. . . the treatment effect!

Conclusion

Comparing observations to simulations from a fitted model can reveal weaknesses in the structure of the fitted model.