# VAST

James T. Thorson

```
library(tinyVAST)
library(fmesher)
set.seed(101)
```

`tinyVAST` is an R package for fitting vector autoregressive spatio-temporal (VAST) models. We here explore the capacity to specify the vector-autoregressive spatio-temporal component.

## Spatio-temporal autoregressive model

We first explore the ability to specify a first-order autoregressive spatio-temporal process:

```
# Simulate settings
theta_xy = 0.4
n_x = n_y = 10
n_t = 15
rho = 0.8
spatial_sd = 0.5

# Simulate GMRFs
R = exp(-theta_xy * abs(outer(1:n_x, 1:n_y, FUN="-")) )
d = mvtnorm::rmvnorm(n_t, sigma=spatial_sd^2*kronecker(R,R) )

# Project through time and add mean
for( t in seq_len(n_t) ){
  if(t>1) d[t,] = rho*d[t-1,] + d[t,]
}
#d = d + 0.5

# Shape into longform data-frame and add error
#Data = data.frame( expand.grid(time=1:n_t, x=1:n_x, y=1:n_y), "var"="logn", z=as.vector(d))
#Data$n = Data$z + rnorm(nrow(Data), sd=0.2)
Data = data.frame( expand.grid(time=1:n_t, x=1:n_x, y=1:n_y), "var"="logn", z=exp(as.vector(d)))
Data$n = tweedie::rtweedie( n=nrow(Data), mu=Data$z, phi=0.5, power=1.5 )
mean(Data$n==0)
#> [1] 0.046

# make mesh
mesh = fm_mesh_2d( Data[,c('x','y')] )

# fit model
out = fit( sem = "logn -> logn, 1, rho",
           data = Data,
```

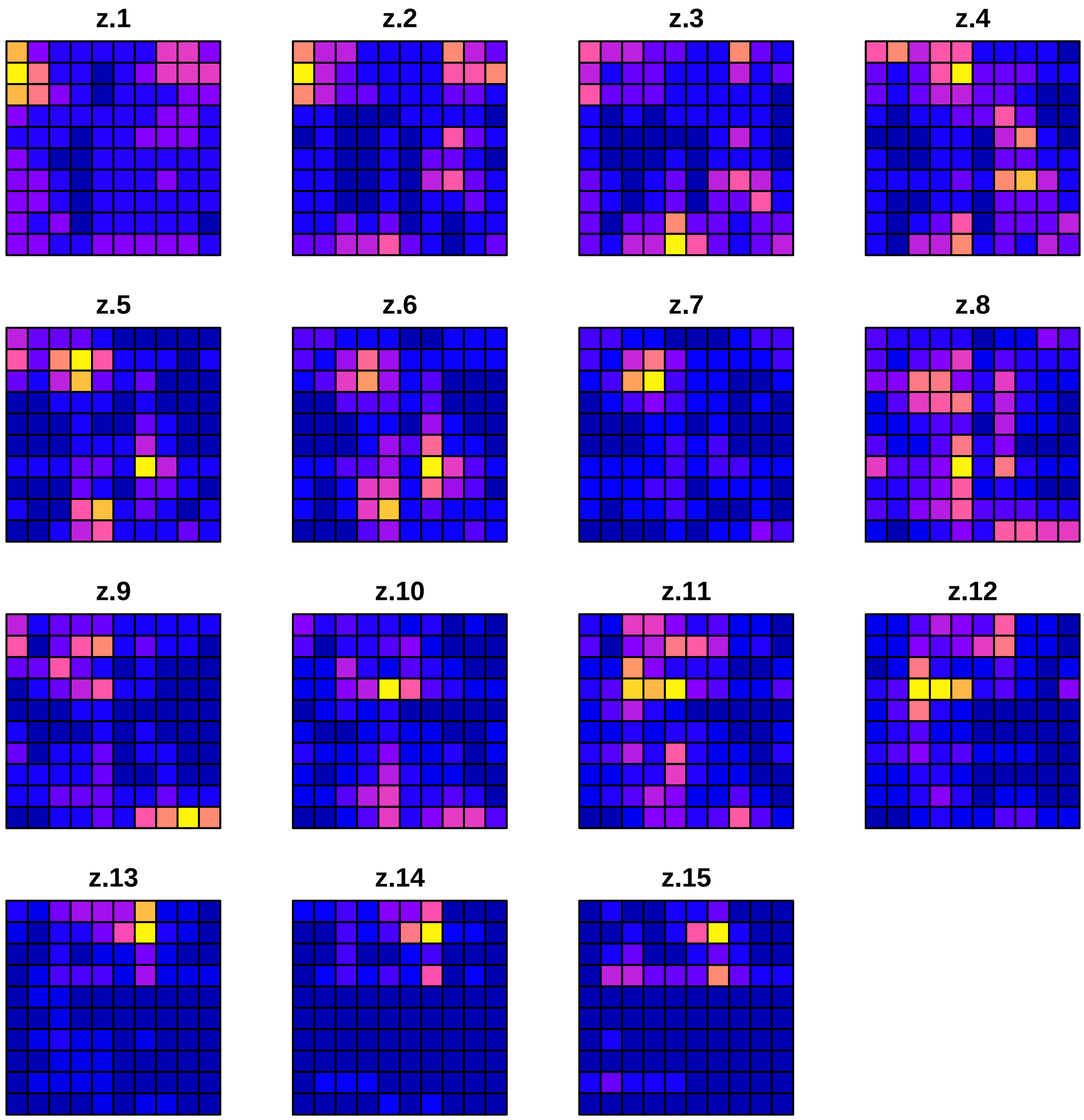
```
          formula = n ~ 0 + factor(time),
          spatial_graph = mesh,
          quiet = TRUE,
          family_link = c(1,0) )
out
#> $call
#> fit(data = Data, formula = n ~ 0 + factor(time), sem = "logn -> logn, 1, rho",
#>     family_link = c(1, 0), spatial_graph = mesh, quiet = TRUE)
#>
#> $opt
#> $opt$par
#>   log_kappa     alpha_j     alpha_j     alpha_j     alpha_j     alpha_j     alpha_j     alpha_j     a
#>  0.07228542 -0.08323604 -0.13549104 -0.10579217 -0.14499113 -0.37823868 -0.21633306 -0.41489958 -0.6
#>     alpha_j     alpha_j      beta_z      beta_z   log_sigma   log_sigma
#>  0.16887044  0.30040122  0.81229112  0.40988915 -0.64868475  0.04394543
#>
#> $opt$objective
#> [1] 1717.689
#>
#> $opt$convergence
#> [1] 0
#>
#> $opt$iterations
#> [1] 77
#>
#> $opt$evaluations
#> function gradient
#>      107       77
#>
#> $opt$message
#> [1] "relative convergence (4)"
#>
#>
#> $sdrep
#> sdreport(.) result
#>                Estimate Std. Error
#> log_kappa   0.07228542 0.10755269
#> alpha_j    -0.08323604 0.15196456
#> alpha_j    -0.13549104 0.18670340
#> alpha_j    -0.10579217 0.20529851
#> alpha_j    -0.14499113 0.21780791
#> alpha_j    -0.37823868 0.22691800
#> alpha_j    -0.21633306 0.23026450
#> alpha_j    -0.41489958 0.23456777
#> alpha_j    -0.67168423 0.23833635
#> alpha_j    -0.49463135 0.23869331
#> alpha_j    -0.13968721 0.23733095
#> alpha_j     0.14836186 0.23640201
#> alpha_j    -0.21516693 0.23873590
#> alpha_j    -0.20120060 0.23979214
#> alpha_j     0.16887044 0.23708655
#> alpha_j     0.30040122 0.23660035
#> beta_z      0.81229112 0.03708631
```

```
#> beta_z     0.40988915 0.03291043
#> log_sigma -0.64868475 0.05422114
#> log_sigma  0.04394543 0.07275797
#> Maximum gradient component: 0.006323539
#>
#> $run_time
#> Time difference of 22.6486 secs
```

The estimated values for `beta_z` then correspond to the simulated value for `rho` and `spatial_sd`.

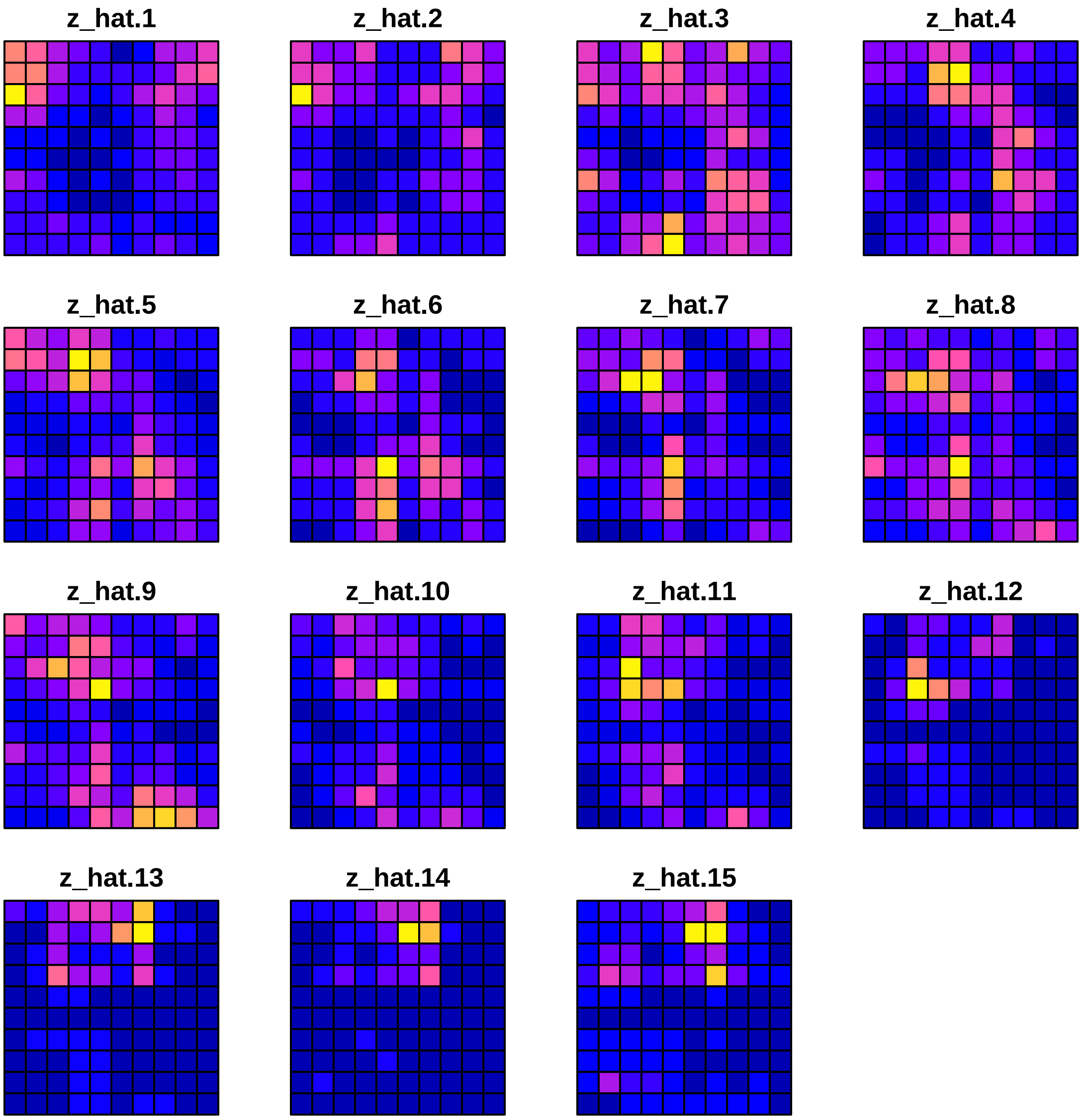
We can compare the true densities:

```
library(sf)
#> Warning: package 'sf' was built under R version 4.3.1
data_wide = reshape( Data[,c('x','y','time','z')],
                     direction = "wide", idvar = c('x','y'), timevar = "time")
sf_data = st_as_sf( data_wide, coords=c("x","y"))
sf_grid = sf::st_make_grid( sf_data )
sf_plot = st_sf(sf_grid, st_drop_geometry(sf_data) )
plot(sf_plot, max.plot=n_t )
```



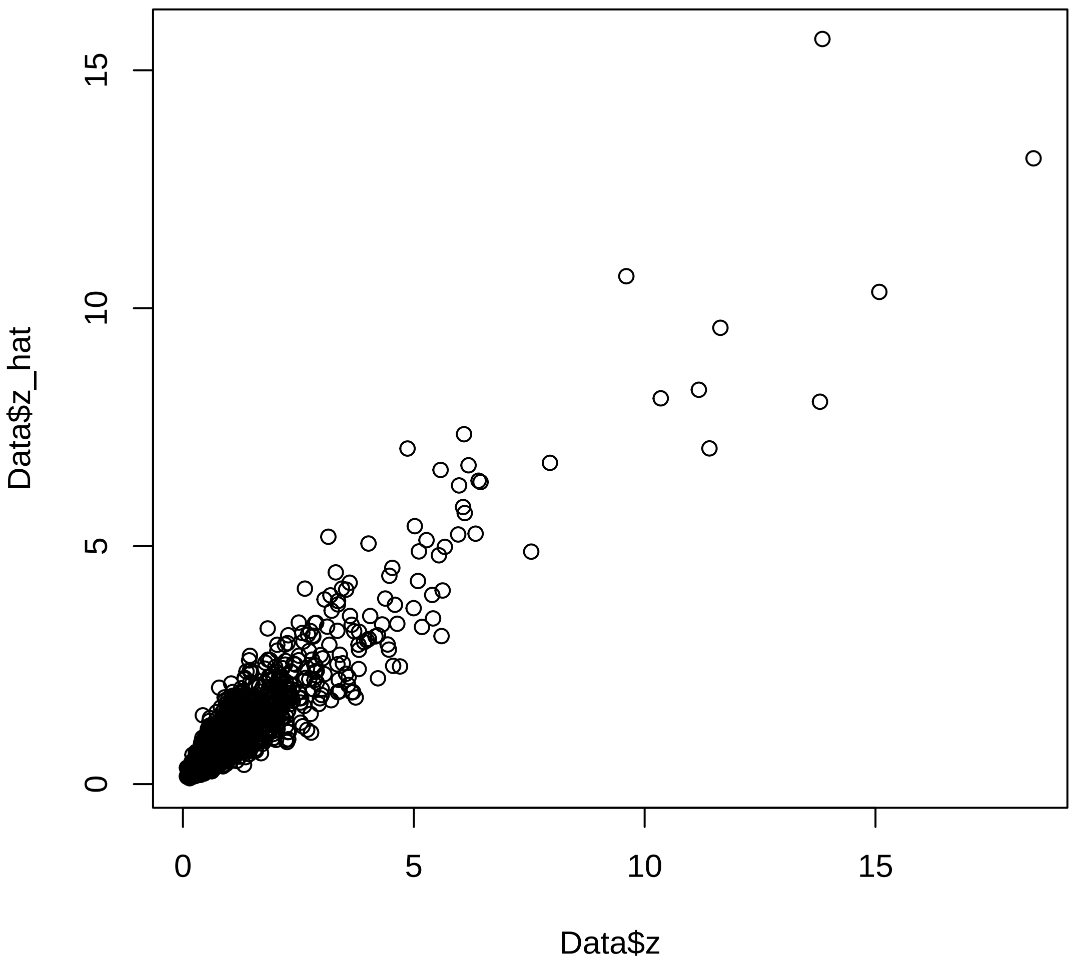

with the estimated densities:

```
Data$z_hat = predict(out)
data_wide = reshape( Data[,c('x','y','time','z_hat')],
                     direction = "wide", idvar = c('x','y'), timevar = "time")
sf_data = st_as_sf( data_wide, coords=c("x","y"))
sf_plot = st_sf(sf_grid, st_drop_geometry(sf_data) )
plot(sf_plot, max.plot=n_t )
```




where a scatterplot shows that they are highly correlated:

```
plot( x=Data$z, y=Data$z_hat )
```




We can then calculate the area-weighted total abundance and compare it with its true value:

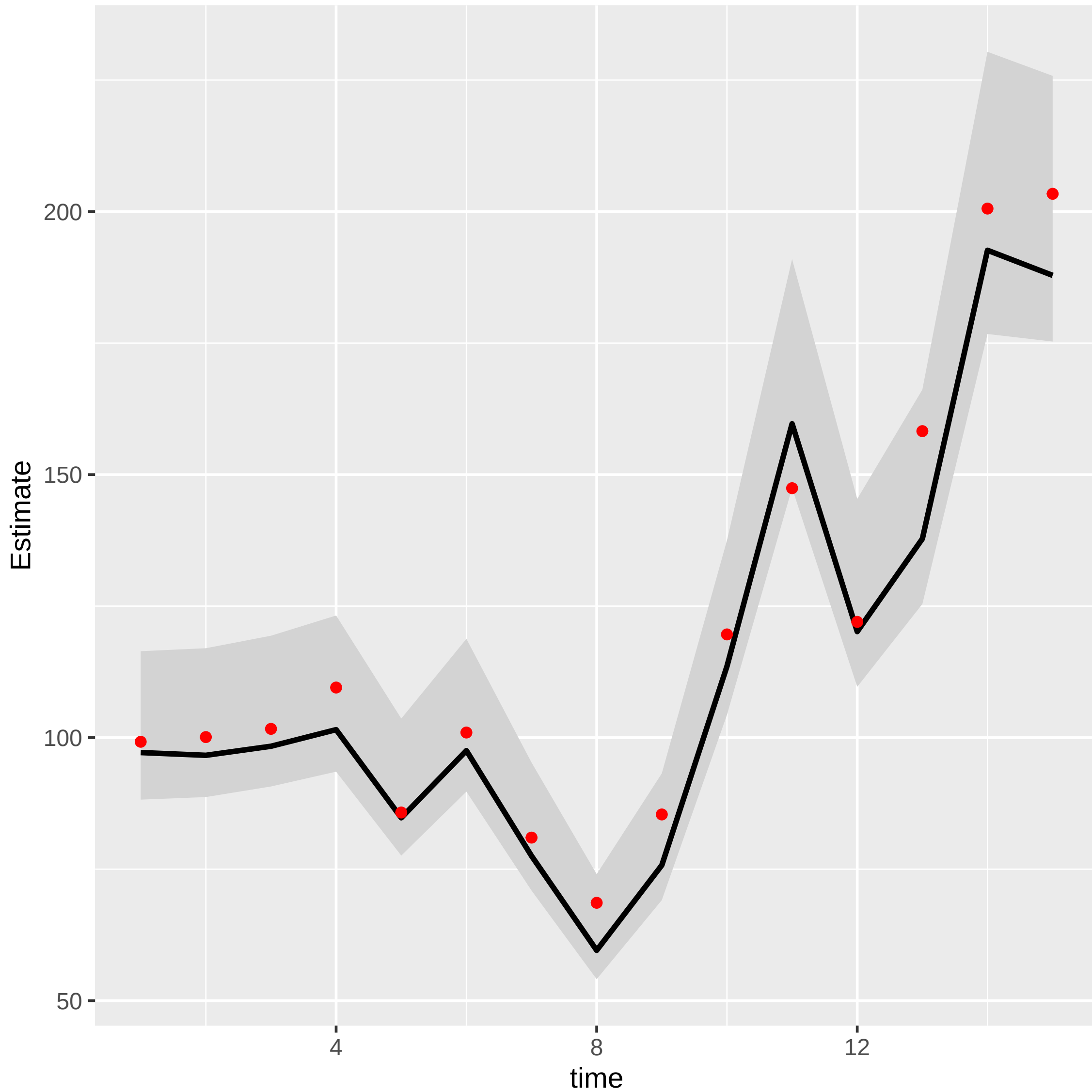
```
# Predicted sample-weighted total
(Est = sapply( seq_len(n_t), FUN=\(t) integrate_output(out, newdata=subset(Data,time==t)) ))
#>                              [,1]       [,2]       [,3]       [,4]      [,5]       [,6]      [,7]
#> Estimate                97.164903  96.643634  98.362457 101.517620 84.760587  97.538111 77.520820 59.56
#> Std. Error               7.194683   7.216494   7.309313   7.572241  6.643419   7.406226  6.207919  5.09
#> Est. (bias.correct) 102.324275 102.850496 105.043004 108.373177 90.604659 104.258111 83.102278 64.06
#> Std. (bias.correct)         NA         NA         NA         NA        NA         NA        NA
#>                           [,15]
#> Estimate              187.86973
#> Std. Error             12.88189
#> Est. (bias.correct) 200.54754
#> Std. (bias.correct)         NA

# True (latent) sample-weighted total
(True = tapply( Data$z, INDEX=Data$time, FUN=sum ))
```

```
#>          1          2          3          4          5          6          7          8          9         10
#>  99.21643 100.10603 101.66846 109.52622  85.76973 100.97116  80.99847  68.60738  85.39974 119.62380

#
Index = data.frame( time=seq_len(n_t), t(Est), True )
Index$low = Index[,'Est...bias.correct.'] - 1.96*Index[,'Std..Error']
Index$high = Index[,'Est...bias.correct.'] + 1.96*Index[,'Std..Error']

#
library(ggplot2)
ggplot(Index, aes(time, Estimate)) +
  geom_ribbon(aes(ymin = low,
                ymax = high),    # shadowing cnf intervals
            fill = "lightgrey") +
  geom_line( color = "black",
           linewidth = 1) +
  geom_point( aes(time, True), color = "red" )
```




Next, we compare this against the current version of VAST

```
#library(VAST)
#settings = make_settings( purpose="index3",
#                          n_x = n_x*n_y,
#                          Region = "Other",
#                          bias.correct = FALSE,
#                          use_anisotropy = FALSE )
#settings$FieldConfig['Epsilon','Component_1'] = 0
#settings$FieldConfig['Omega',] = 0
#settings$RhoConfig['Epsilon2'] = 4
#settings$RhoConfig['Beta1'] = 3
#
#myVAST = fit_model( settings=settings,
#                  Lat_i = Data[,'y'],
#                  Lon_i = Data[,'x'],
#                  t_i = Data[,'time'],
```

```
#                 b_i = exp(Data[,'n']),
#                 a_i = rep(1,nrow(Data)),
#                 observations_LL = cbind(Lat=Data[,'y'],Lon=Data[,'x']),
#                 grid_dim_km = c(100,100),
#                 ObsModel = c(10,2),
#                 newtonsteps = 0,
#                 loopnum = 0,
#                 control = list(eval.max=100, iter.max=100, trace=0) )
#myVAST
```

Or with sdmTMB

```
#library(INLA)
library(sdmTMB)
#> Warning: package 'sdmTMB' was built under R version 4.3.1
mesh = make_mesh(Data, c("x","y"), n_knots=n_x*n_y )

start_time = Sys.time()
mysdmTMB = sdmTMB(
  formula = n ~ 0 + factor(time),
  data = Data,
  mesh = mesh,
  spatial = "off",
  spatiotemporal = "ar1",
  time = "time",
  family = tweedie()
)
Sys.time() - start_time
#> Time difference of 13.78178 secs
```

## Bivariate spatio-temporal autoregressive model

We next highlight how to specify a bivariate spatio-temporal model with a cross-laggged (vector autoregressive) interaction.

```
# Simulate settings
theta_xy = 0.2
n_x = n_y = 10
n_t = 20
B = rbind( c( 0.5, -0.25),
           c(-0.1,  0.50) )

# Simulate GMRFs
R = exp(-theta_xy * abs(outer(1:n_x, 1:n_y, FUN="-")) )
d1 = mvtnorm::rmvnorm(n_t, sigma=0.2*kronecker(R,R) )
d2 = mvtnorm::rmvnorm(n_t, sigma=0.2*kronecker(R,R) )
d = abind::abind( d1, d2, along=3 )

# Project through time and add mean
for( t in seq_len(n_t) ){
  if(t>1) d[t,,] = t(B%*%t(d[t-1,,])) + d[t,,]
```

```r
}

# Shape into longform data-frame and add error
Data = data.frame( expand.grid(time=1:n_t, x=1:n_x, y=1:n_y, "var"=c("d1","d2")), z=exp(as.vector(d)))
Data$n = tweedie::rtweedie( n=nrow(Data), mu=Data$z, phi=0.5, power=1.5 )

# make mesh
mesh = fm_mesh_2d( Data[,c('x','y')] )

# Define sem
sem = "
  d1 -> d1, 1, b11
  d2 -> d2, 1, b22
  d2 -> d1, 1, b21
  d1 -> d2, 1, b12
  d1 <-> d1, 0, var1
  d2 <-> d2, 0, var1
"

# fit model
out = fit( sem = sem,
           data = Data,
           formula = n ~ 0 + var,
           spatial_graph = mesh,
           quiet = TRUE,
           family_link = c(1,0) )
out
#> $call
#> fit(data = Data, formula = n ~ 0 + var, sem = sem, family_link = c(1,
#>     0), spatial_graph = mesh, quiet = TRUE)
#>
#> $opt
#> $opt$par
#>    log_kappa      alpha_j      alpha_j       beta_z       beta_z       beta_z       beta_z       bet
#> -0.669057055 -0.090128404 -0.002000425  0.509529386  0.529236420 -0.200418869 -0.117205375  0.294319
#>
#> $opt$objective
#> [1] 4365.006
#>
#> $opt$convergence
#> [1] 0
#>
#> $opt$iterations
#> [1] 52
#>
#> $opt$evaluations
#> function gradient
#>       66       53
#>
#> $opt$message
#> [1] "relative convergence (4)"
#>
#>
```

```
#> $sdrep
#> sdreport(.) result
#>              Estimate Std. Error
#> log_kappa -0.669057055 0.09746707
#> alpha_j   -0.090128404 0.09771148
#> alpha_j   -0.002000425 0.09611298
#> beta_z     0.509529386 0.07886506
#> beta_z     0.529236420 0.07336726
#> beta_z    -0.200418869 0.08304602
#> beta_z    -0.117205375 0.07264402
#> beta_z     0.294319072 0.01800602
#> log_sigma -0.646266076 0.02660900
#> log_sigma  0.012846262 0.04964136
#> Maximum gradient component: 0.004487158
#>
#> $run_time
#> Time difference of 2.29318 mins
```

The values for `beta_z` again correspond to the specified value for interaction-matrix `B`

We can again calculate the area-weighted total abundance and compare it with its true value:

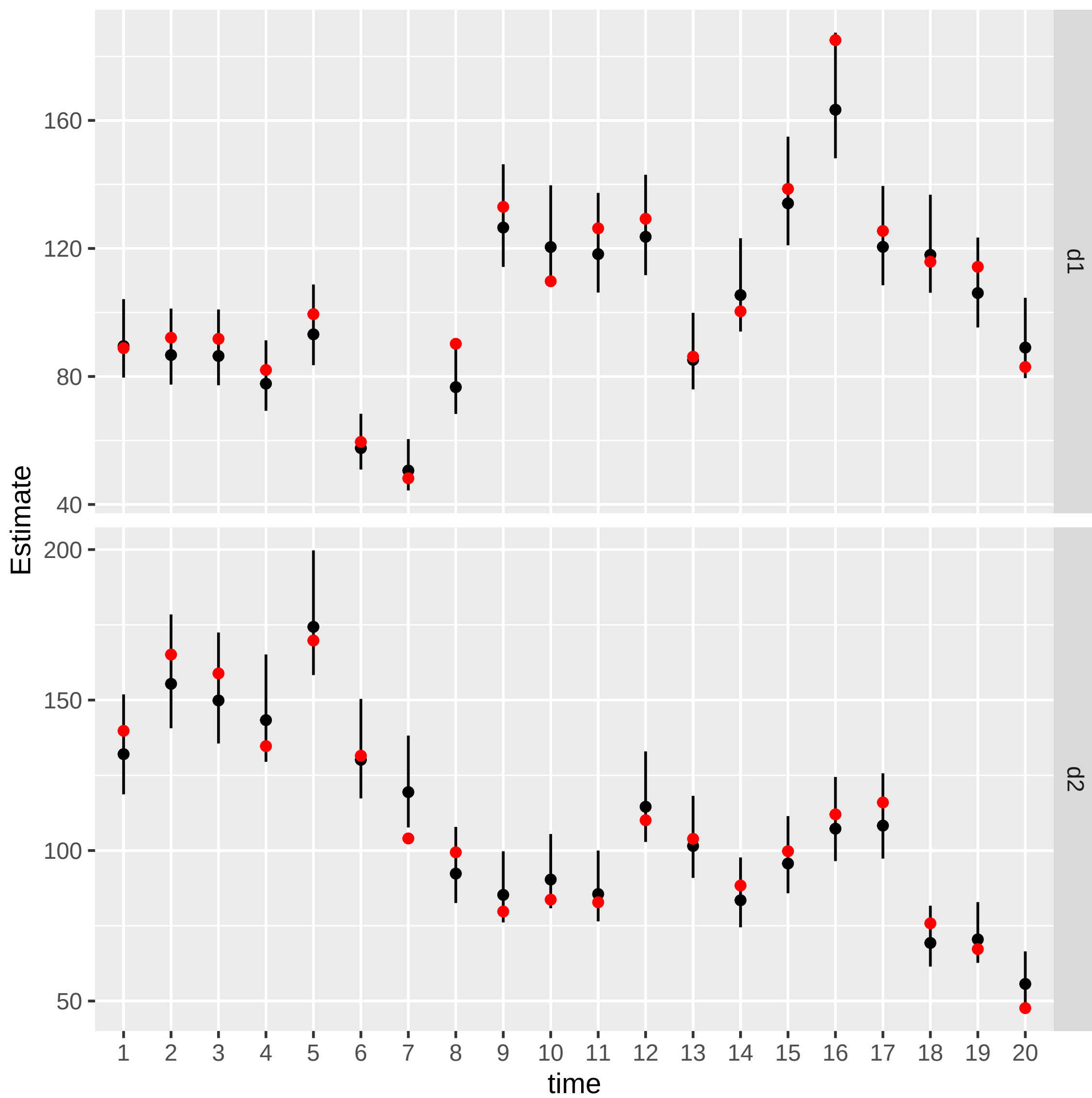
```
# Predicted sample-weighted total
integrate_output(out, newdata=subset(Data,time==1 & var=="d1") )
#>           Estimate          Std. Error Est. (bias.correct) Std. (bias.correct)
#>           89.49019             6.25281            91.90724                  NA

Est1 = sapply( seq_len(n_t), FUN=\(t) integrate_output(out, newdata=subset(Data,time==t & var=="d1")) )
Est2 = sapply( seq_len(n_t), FUN=\(t) integrate_output(out, newdata=subset(Data,time==t & var=="d2")) )

# True (latent) sample-weighted total
True = tapply( Data$z, INDEX=list("time"=Data$time,"var"=Data$var), FUN=sum )

#
Index = data.frame( expand.grid(dimnames(True)), "True"=as.vector(True) )
Index = data.frame( Index, rbind(t(Est1), t(Est2)) )
Index$low = Index[,'Est...bias.correct.'] - 1.96*Index[,'Std..Error']
Index$high = Index[,'Est...bias.correct.'] + 1.96*Index[,'Std..Error']

#
library(ggplot2)
ggplot(Index, aes( time, Estimate )) +
  facet_grid( rows=vars(var), scales="free" ) +
  geom_segment(aes(y = low,
                   yend = high,
                   x = time,
                   xend = time) ) +
  geom_point( aes(x=time, y=Estimate), color = "black") +
  geom_point( aes(x=time, y=True), color = "red" )
```


Estimate
160
120
80
40
200
150
100
50
d1
d2
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
time