Spatial model

James T. Thorson

```r
library(tinyVAST)
library(pdp)  # approx = TRUE gives effects for average of other covariates
library(lattice)
library(visreg)
library(fmesher)
set.seed(101)
```

`tinyVAST` is an R package for fitting vector autoregressive spatio-temporal (VAST) models using a minimal and user-friendly interface. We here show how it can fit spatial autoregressive model

```r
# Simulate a 2D AR1 spatial process with a cyclic confounder w
n_x = n_y = 25
n_w = 10
R = exp(-0.4 * abs(outer(1:n_x, 1:n_y, FUN="-")) )
z = mvtnorm::rmvnorm(1, sigma=kronecker(R,R) )
w = sample(1:n_w, replace=TRUE, size=length(z))
Data = data.frame( expand.grid(x=1:n_x, y=1:n_y), w=w, z=as.vector(z) + cos(w/n_w*2*pi))
Data$n = Data$z + rnorm(nrow(Data), sd=1)

# Add columns for multivariate and temporal dimensions
Data$time = 1
Data$var = "n"

# make mesh
mesh = fm_mesh_2d( Data[,c('x','y')], n=100 )

# fit model
out = fit( data = Data,
           formula = n ~ s(w),
           spatial_graph = mesh,
           quiet = TRUE,
           sem = "" )
#>   0:     1072.7221:  0.00000  0.00000  1.00000  0.00000  0.00000
#>   1:     1071.9598: -0.0398872 -0.0398803 0.960488 -0.0540440 -0.0483390
#>   2:     1070.3226: -0.0568636 -0.0788920 0.982398 -0.115966 0.0139143
#>   3:     1068.1026: -0.128898 -0.153888 0.937133 -0.279548 -0.00554493
#>   4:     1064.6958: -0.205442 -0.118794 0.908692 -0.658465 0.0867748
#>   5:     1063.0755: -0.248544 -0.0721380  1.07390 -0.997942 -0.0290980
#>   6:     1060.8336: -0.0274487 -0.0626704 0.948663 -1.30652 -0.0180276
#>   7:     1060.5076: -0.0578397 -0.0961313 0.977313 -1.33220 0.0111084
#>   8:     1059.9650: -0.0978950 -0.125667 0.949843 -1.35915 -0.00925803
#>   9:     1059.5865: -0.135584 -0.126872 0.950353 -1.40413 0.0211852
#>  10:     1058.1162: -0.222109 -0.0737201 0.934388 -1.81382 0.0365153
#>  11:     1057.6075: -0.145880 -0.283610 0.936120 -2.17096 0.00147891
#>  12:     1056.9569: -0.157857 -0.180002 0.983939 -2.57775 -0.00117607
#>  13:     1056.7318: -0.246091 -0.176631 0.918994 -2.61674 0.0414908
#>  14:     1056.7112: -0.296303 -0.180618 0.941564 -2.71808 0.0415180
#>  15:     1056.6918: -0.270848 -0.181148 0.941673 -2.76543 0.0421429
#>  16:     1056.6883: -0.245103 -0.177200 0.934624 -2.76879 0.0348163
#>  17:     1056.6843: -0.258286 -0.180162 0.935432 -2.76049 0.0392699
#>  18:     1056.6843: -0.257770 -0.180092 0.935747 -2.76256 0.0389874
#>  19:     1056.6843: -0.257789 -0.180113 0.935671 -2.76240 0.0390217
#>  20:     1056.6843: -0.257795 -0.180116 0.935671 -2.76241 0.0390228
```

We can then calculate the area-weighted total abundance:

```
# Predicted sample-weighted total
integrate_output(out, newdata = out$data)
#>            Estimate          Std. Error Est. (bias.correct) Std. (bias.correct)
#>          -104.15032            25.99485          -104.15032                  NA
# integrate_output(out, apply.epsilon=TRUE )

# True (latent) sample-weighted total
sum( Data$z )
#> [1] -92.89517
```

Percent deviance explained

We can compute deviance residuals and percent-deviance explained:

```
R1 = sum( residuals(out, type="deviance")^2 )

# tinyVAST null model with just a single intercept
null = fit( data = Data, formula = n ~ 1, quiet = TRUE )
#>   0:     1374.7826:  0.00000  0.00000
#>   1:     1304.3352: -0.106121 0.994353
#>   2:     1211.1263: -0.494707 0.679709
#>   3:     1186.8655: -0.445286 0.491007
#>   4:     1178.4553: -0.254384 0.450911
#>   5:     1177.8581: -0.221017 0.480296
#>   6:     1177.3801: -0.178239 0.468175
#>   7:     1177.3567: -0.166262 0.464657
#>   8:     1177.3566: -0.166647 0.464831
#>   9:     1177.3566: -0.166641 0.464832
R0 = sum( residuals(null, type="deviance")^2 )

# Percent deviance explained
1 - R1/R0
#> [1] 0.7063608
```

We can then compare this with the PDE reported by `mgcv`

```
mygam = gam( n ~ s(w) + s(x,y), data=Data ) #
summary(mygam)$dev.expl
#> [1] 0.3517756
```

where this comparison shows that using the SPDE method in tinyVAST results in higher percent-deviance-explained. This reduced performance for splines relative to the SPDE method presumably arises due to the reduced rank of the spline basis expansion, and the better match for the Matern function (in the SPDE method) relative to the true (simulated) exponential semivariogram.

It is then easy to confirm that mgcv and tinyVAST give (essentially) identical PDE when excluding the spatial term from each.

```
out_reduced = fit( data = Data, formula = n ~ s(w), quiet = TRUE, sem = "" )
#>   0:     1255.4413:  0.00000  1.00000  0.00000  0.00000
#>   1:     1141.8207: -8.89267e-06 0.999331 -0.00994227 0.471886
#>   2:     1130.0023: -0.00179144 0.865279 -1.29960 0.427848
#>   3:     1125.9230: -0.0257632 -0.618202 -2.32366 0.357518
#>   4:     1125.2488: -0.0513104 0.437569 -2.60466 0.367953
#>   5:     1124.3697: -0.0816656 -0.185559 -2.67171 0.379402
#>   6:     1122.3417: -0.116881 0.00953818 -2.69348 0.377903
#>   7:     1122.1413: -0.130171 -0.00505463 -2.69362 0.372610
#>   8:     1122.1127: -0.144400 0.00941808 -2.69387 0.374981
#>   9:     1121.9576: -0.159224 -0.00430108 -2.69486 0.372038
#>  10:     1121.9267: -0.166461 0.00225992 -2.69547 0.371741
#>  11:     1121.9179: -0.169008 -0.000614985 -2.69638 0.372060
#>  12:     1121.9147: -0.165668 0.000302883 -2.69653 0.373974
#>  13:     1121.9145: -0.165758 -5.13786e-05 -2.69656 0.373923
#>  14:     1121.9145: -0.166034 0.000137126 -2.69669 0.373824
#>  15:     1121.9143: -0.166503 -0.000108729 -2.69721 0.373825
#>  16:     1121.9142: -0.166541 3.74888e-05 -2.69794 0.373833
#>  17:     1121.9134: -0.165010 6.29350e-05 -2.70971 0.373918
#>  18:     1121.9113: -0.166386 3.57378e-05 -2.73905 0.373877
#>  19:     1121.9110: -0.166636 3.21398e-07 -2.75544 0.373802
#>  20:     1121.9110: -0.166642 -2.10002e-07 -2.75558 0.373827
#>  21:     1121.9110: -0.166641 -1.34586e-10 -2.75557 0.373821
R1_reduced = sum( residuals(out_reduced, type="deviance")^2 )
1 - R1_reduced/R0
#> [1] 0.1721612

#
mygam_reduced = gam( n ~ s(w), data=Data ) #
summary(mygam_reduced)$dev.expl
#> [1] 0.1713967
```

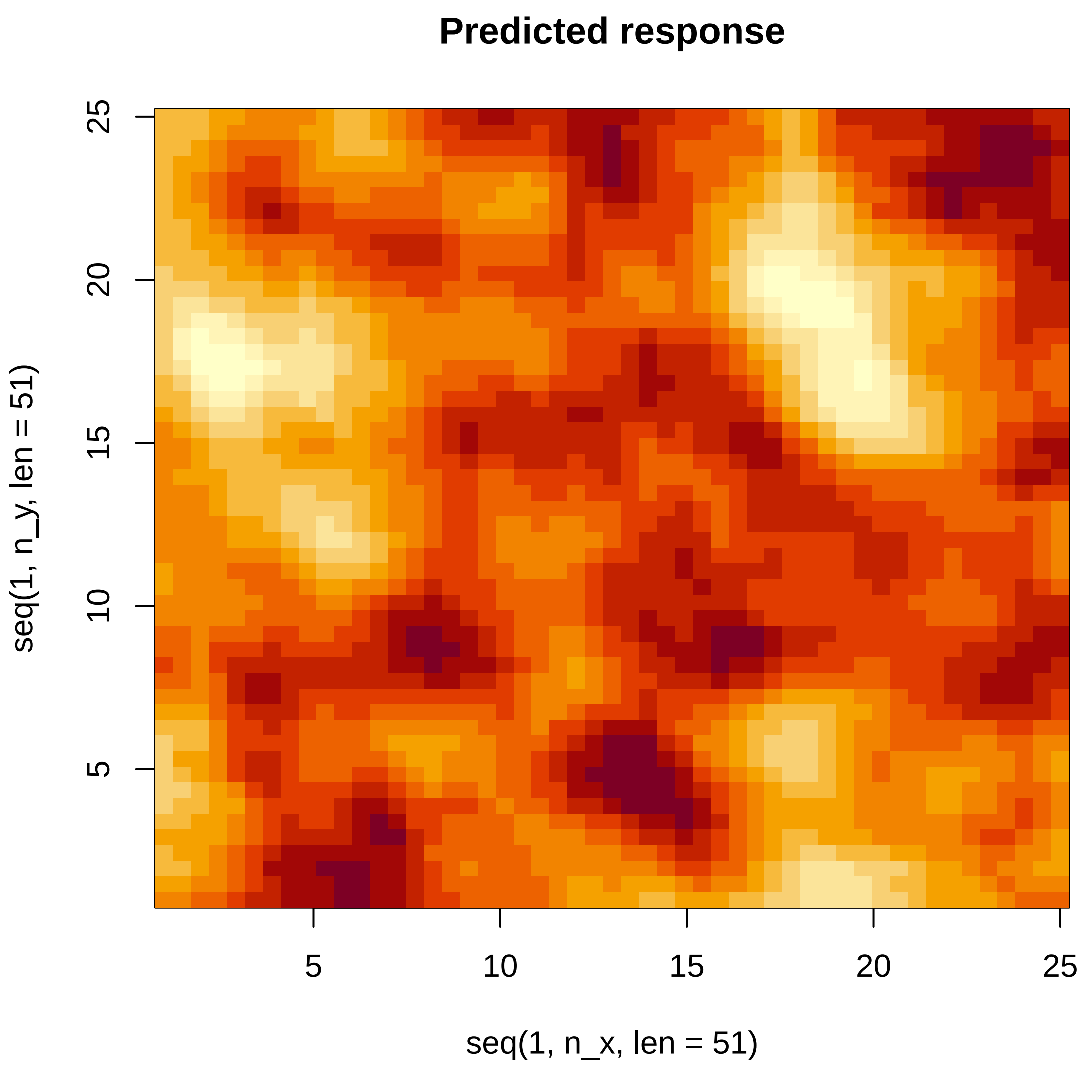
Visualize spatial response

`tinyVAST` then has a standard `predict` function:

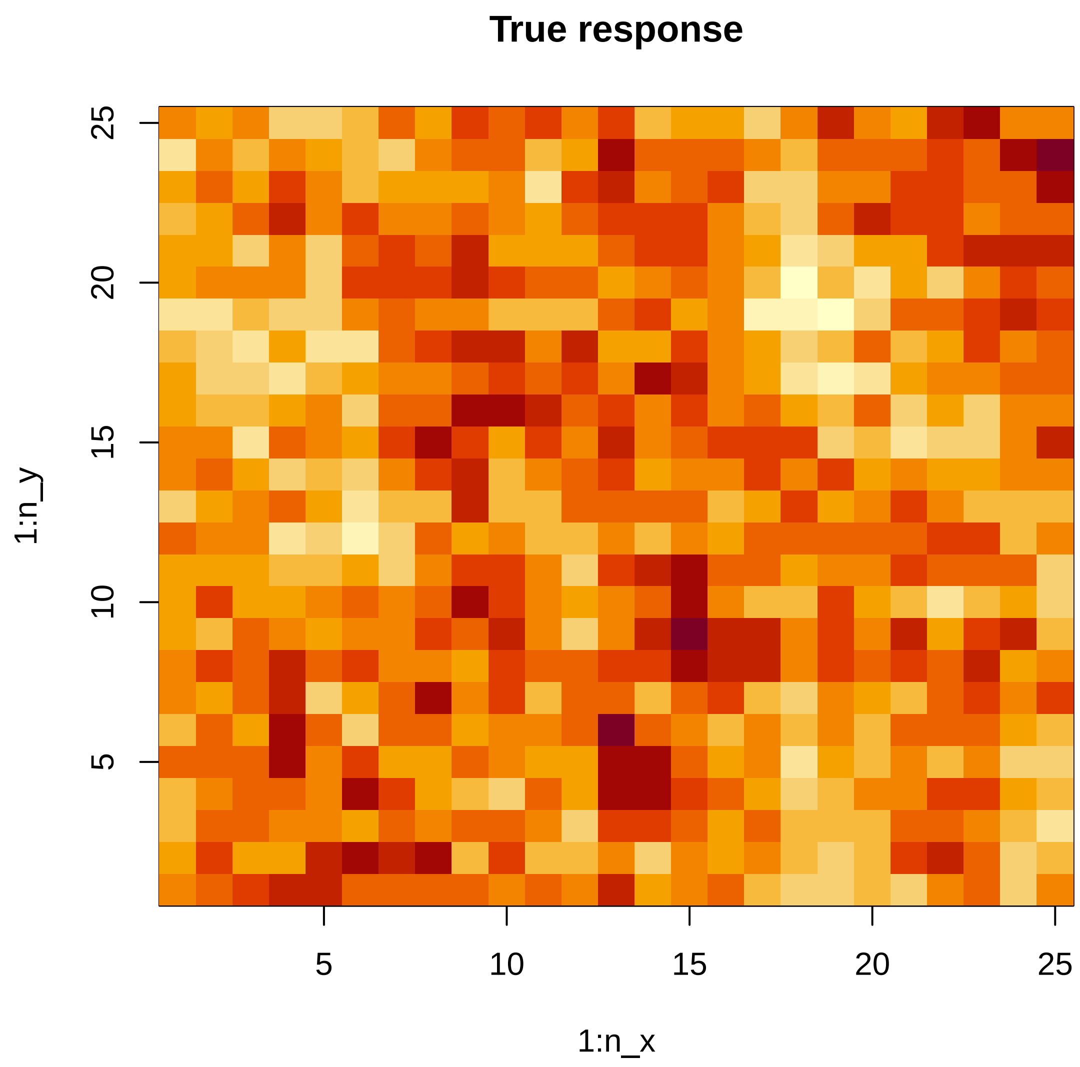
```
predict(out, newdata=data.frame(x=1, y=1, time=1, w=1, var="n") )
#> [1] 0.2978245
```

and this is used to compute the spatial response

```
# Prediction grid
pred = outer( seq(1,n_x,len=51),
              seq(1,n_y,len=51),
                FUN=\(x,y) predict(out,newdata=data.frame(x=x,y=y,w=1,time=1,var="n")) )
image( x=seq(1,n_x,len=51), y=seq(1,n_y,len=51), z=pred, main="Predicted response" )
```



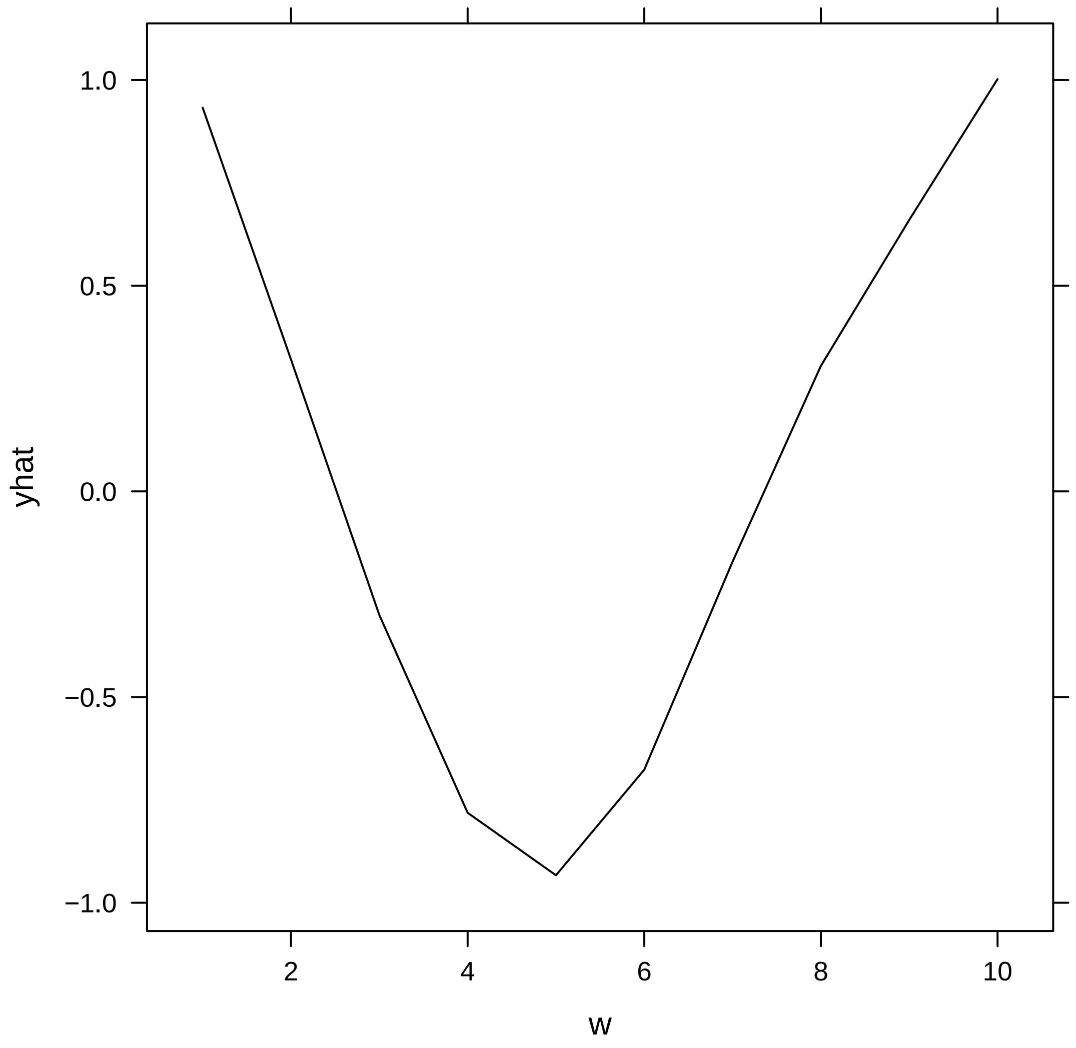
```
# True value
image( x=1:n_x, y=1:n_y, z=matrix(Data$z,ncol=n_y), main="True response" )
```



We can also compute the marginal effect of the cyclic confounder

```
# compute partial dependence plot
Partial = partial( object = out,
                   pred.var = "w",
                   pred.fun = \(object,newdata) predict(object,newdata),
                   train = Data,
                   approx = TRUE )

# Lattice plots as default option
plotPartial( Partial )
```


1.0
0.5
0.0
−0.5
−1.0
yhat
2
4
6
8
10
w