Comparison with mgcv

James T. Thorson

```
library(tinyVAST)
library(pdp)  # approx = TRUE gives effects for average of other covariates
library(lattice)
library(visreg)
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 replicate analysis using splines specified via `mgcv`

```
# Simulate
n_obs = 1000
x = rnorm(n_obs)
group = sample( x=1:5, size=n_obs, replace=TRUE )
w = runif(n_obs, min=0, max=2)
z = 1 + x^2 + cos((w+group/5)*2*pi) + rnorm(5)[group]
y = z + rnorm(n_obs, sd=0.2)
Data = data.frame( x=x, y=y, w=w, z=z, group=factor(group) )

# fit model
Formula = y ~ 1 + s(group, bs="re") + poly(x, 2, raw=TRUE) + s(w, by=group)
myfit = fit( data = Data,
             formula = Formula,
             control = tinyVASTcontrol(quiet = TRUE) )
#>   0:     1976.7744:  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
#>   1:     1562.1834: 0.00437068 -0.0342385 0.862116 -0.00287401 -0.00257969 -0.00124244 -0.00256578 -
#>   2:     1001.6065: 0.0108260 -0.0313340  1.20465 -0.00402356 -0.00421038 -0.00134441 -0.00332393 -0
#>   3:     912.62104: 0.0112952 -0.0127428  1.06743 -0.00387618 -0.00802293 -0.00462329 -0.00641537 -0
#>   4:     859.34404: 0.0123019 0.000730758 0.989525 -0.00374398 -0.0152797 -0.0108421 -0.0126533 -0.0
#>   5:     813.65909: 0.0158268 -0.00565141  1.09729 -0.00357733 -0.0455068 -0.0371431 -0.0399346 -0.0
#>   6:     764.55442: 0.0240904 -0.0630139 0.931722 -0.00348747 -0.184361 -0.161128 -0.163770 -0.17418
#>   7:     706.94323: 0.0260986 0.0335145  1.00950 -0.00341487 -0.223795 -0.195695 -0.198851 -0.212119
#>   8:     704.25425: 0.0275829 -0.0777339  1.00790 -0.00329026 -0.255852 -0.223107 -0.226303 -0.24307
#>   9:     659.88167: 0.0297522 0.0137678 0.983669 -0.00314811 -0.308348 -0.268467 -0.271686 -0.293699
#>  10:     657.69183: 0.0317360 -0.0128839  1.06500 -0.00298657 -0.365113 -0.316502 -0.319994 -0.34886
#>  11:     619.18732: 0.0323670 -0.00572801  1.00002 -0.00292769 -0.382315 -0.331170 -0.334471 -0.3654
#>  12:     608.55237: 0.0333633 0.00527764  1.03072 -0.00284641 -0.409339 -0.353858 -0.357300 -0.39162
#>  13:     525.88626: 0.0507077 -0.101179  1.00772 -0.00167399 -0.885540 -0.757920 -0.761744 -0.853029
#>  14:     483.34197: 0.0569613 0.0958905 0.972709 -0.000937276 -1.08828 -0.928204 -0.932422 -1.05036
#>  15:     303.91260: 0.0690215 -0.00868524  1.03191 0.00229934 -1.31424 -1.11918 -1.12786 -1.26651 -1
#>  16:     301.86314: 0.0812573 -0.00853864 0.964464 0.00443585 -1.53110 -1.30945 -1.33004 -1.47985 -1
#>  17:     238.72339: 0.0986219 -0.00210640  1.01524 0.00975726 -1.63418 -1.41990 -1.44808 -1.57621 -1
#>  18:     224.44488: 0.126627 0.0183680 0.994627 0.0191177 -1.52089 -1.35890 -1.40724 -1.46252 -1.217
#>  19:     196.11392: 0.153303 -0.00674083  1.01141 0.0272378 -1.61591 -1.46150 -1.52015 -1.55693 -1.3
#>  20:     177.62225: 0.173759 0.00153448 0.993107 0.0343869 -1.72301 -1.56227 -1.61960 -1.66235 -1.39
```

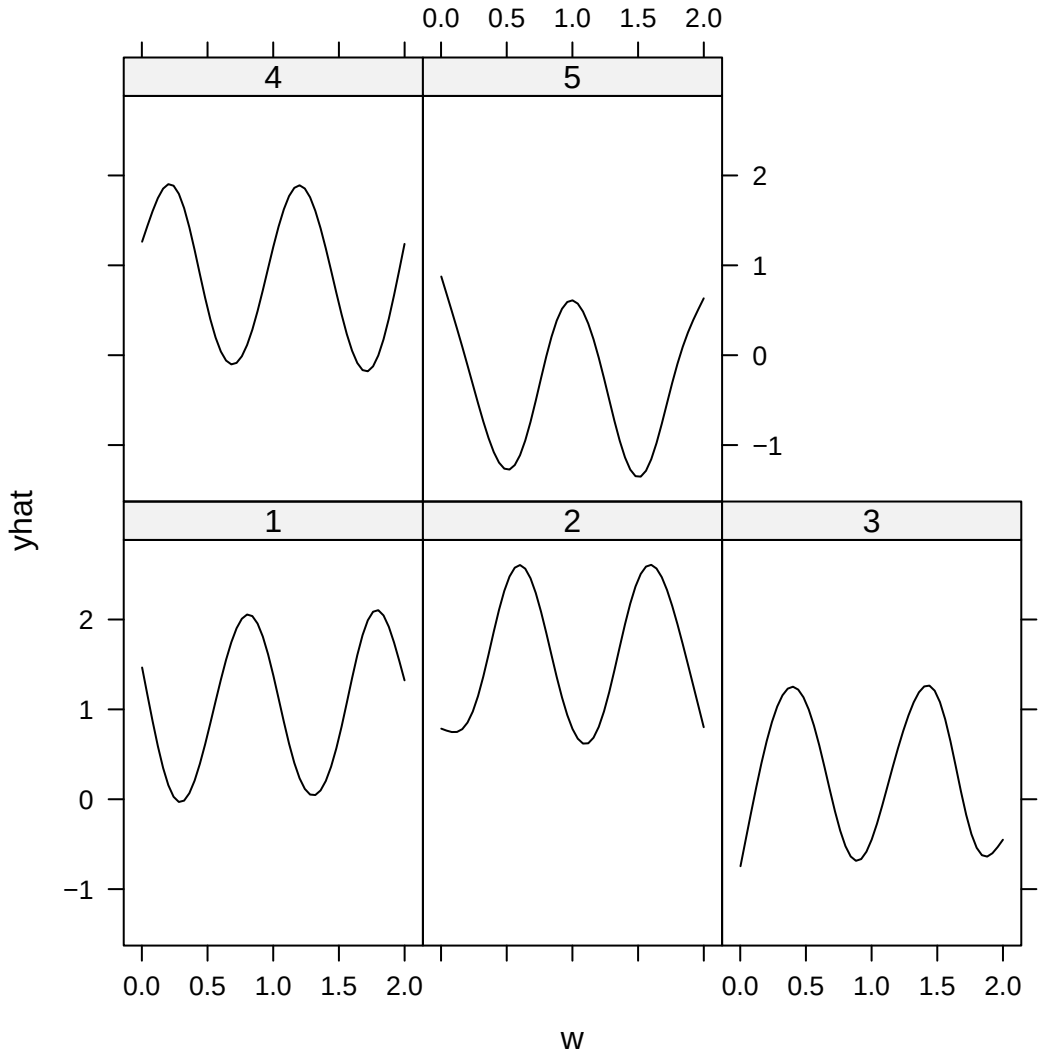
```
#>  21:     159.51348: 0.202402 0.000563049  1.01292 0.0471166 -1.83545 -1.65622 -1.69851 -1.77336 -1.4
#>  22:     101.59513: 0.470432 -0.0216507 0.998074 0.180786 -2.17169 -1.96129 -2.04506 -2.13528 -2.159
#>  23:     47.584629: 0.909558 0.0529657  1.00520 0.475688 -2.58516 -2.28823 -2.29996 -2.54549 -2.5515
#>  24:     18.494471: 0.799795 -0.00268756  1.00508 0.598926 -2.76796 -2.71280 -2.73686 -2.61461 -1.81
#>  25:    -0.62665605: 0.306764 -0.00836862 0.996509 0.786627 -2.99567 -3.16697 -3.24546 -2.87069 -2.13
#>  26:    -12.349279: 0.528741 -0.0120579 0.999844 0.784273 -3.02999 -2.86123 -3.00591 -2.96637 -3.021
#>  27:    -24.757859:  1.10934 0.0212368  1.00735 0.790221 -3.56711 -3.12253 -3.16562 -3.48197 -2.9939
#>  28:    -29.990500: 0.267535 0.0186702  1.00565 0.586466 -3.47000 -3.44026 -3.39542 -3.42933 -3.2603
#>  29:    -36.923483: 0.719926 0.000577888 0.996785 0.162554 -3.79807 -3.62076 -3.72617 -3.67169 -3.47
#>  30:    -40.732766: 0.644831 -0.00193086  1.00539 0.832069 -3.99596 -3.75906 -3.94755 -4.04321 -3.68
#>  31:    -41.060581: 0.812613 0.00478312  1.00649 0.850881 -4.43899 -4.31826 -4.08411 -3.72390 -3.968
#>  32:    -41.456528: 0.633724 0.00989424  1.00381 0.723930 -4.53108 -4.19470 -4.12385 -4.54062 -3.978
#>  33:    -41.902987: 0.768075 0.00437506  1.00437 0.872654 -4.15794 -4.03811 -4.40715 -4.36349 -3.951
#>  34:    -42.107560: 0.697696 0.00243829  1.00447 0.863534 -4.30905 -4.02062 -4.24443 -4.27338 -3.913
#>  35:    -42.114451: 0.710845 0.00355067  1.00474 0.748217 -4.29795 -4.19476 -4.10511 -4.30035 -3.926
#>  36:    -42.155444: 0.689452 0.00450563  1.00470 0.747527 -4.36719 -4.12583 -4.17746 -4.27200 -3.930
#>  37:    -42.159487: 0.691132 0.00384926  1.00433 0.869963 -4.35544 -4.09712 -4.18640 -4.26951 -3.926
#>  38:    -42.165400: 0.692170 0.00382985  1.00457 0.795716 -4.34508 -4.09445 -4.19234 -4.26452 -3.921
#>  39:    -42.165966: 0.691520 0.00402306  1.00454 0.804919 -4.35236 -4.10493 -4.18444 -4.26823 -3.924
#>  40:    -42.166062: 0.691704 0.00396178  1.00454 0.807011 -4.35036 -4.10147 -4.18666 -4.26720 -3.923
#>  41:    -42.166063: 0.691715 0.00396338  1.00454 0.806670 -4.35033 -4.10155 -4.18666 -4.26718 -3.923
#>  42:    -42.166063: 0.691721 0.00396427  1.00454 0.806610 -4.35033 -4.10163 -4.18670 -4.26718 -3.923
#>  43:    -42.166063: 0.691717 0.00396403  1.00454 0.806683 -4.35036 -4.10169 -4.18675 -4.26719 -3.922
#>  44:    -42.166063: 0.691712 0.00396374  1.00454 0.806719 -4.35037 -4.10170 -4.18677 -4.26720 -3.922
#>  45:    -42.166063: 0.691711 0.00396367  1.00454 0.806723 -4.35038 -4.10170 -4.18678 -4.26720 -3.922
```

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

```
predict(myfit, newdata=data.frame(x=0, y=1, w=0.4, group=2) )
#> [1] 1.852686
```

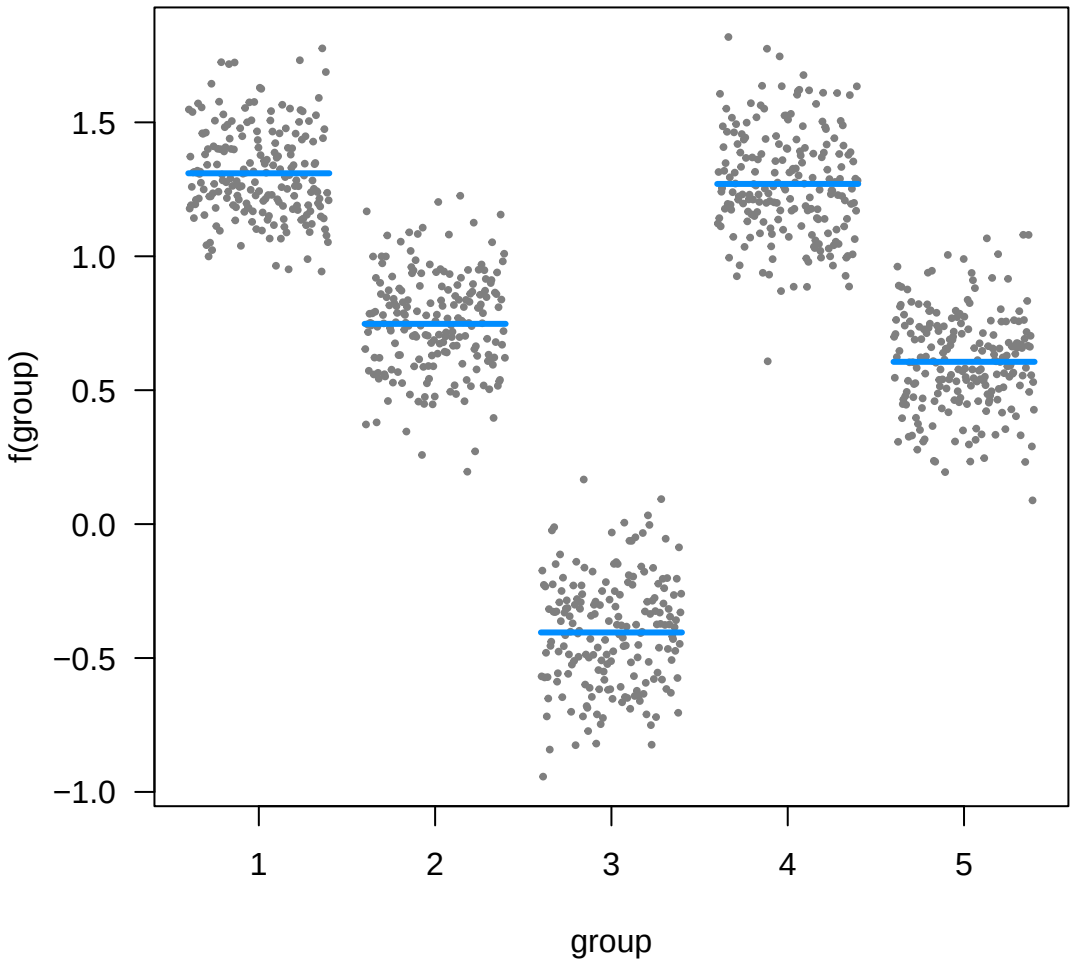
and this is used to compute partial-dependence plots using package `pdp`

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

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

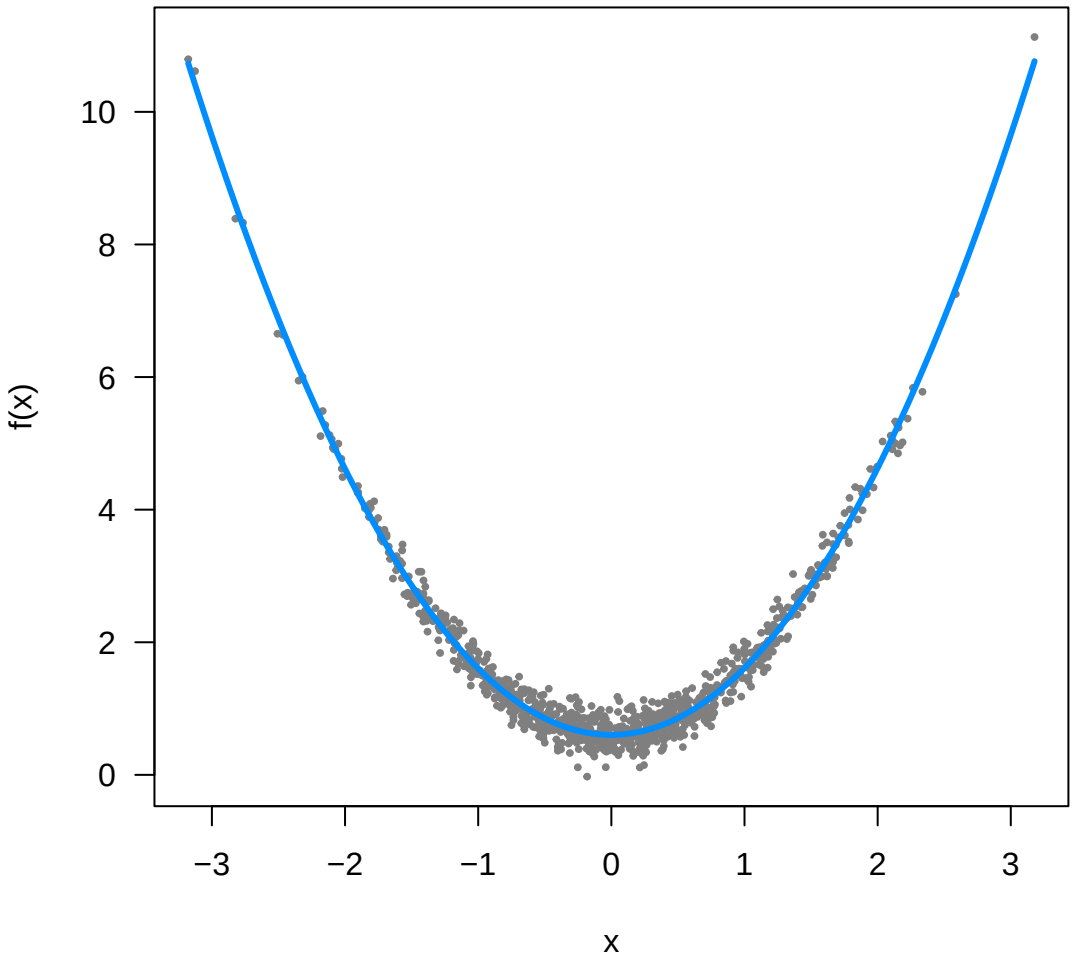


Alternatively, we can use `visreg` to visualize output:

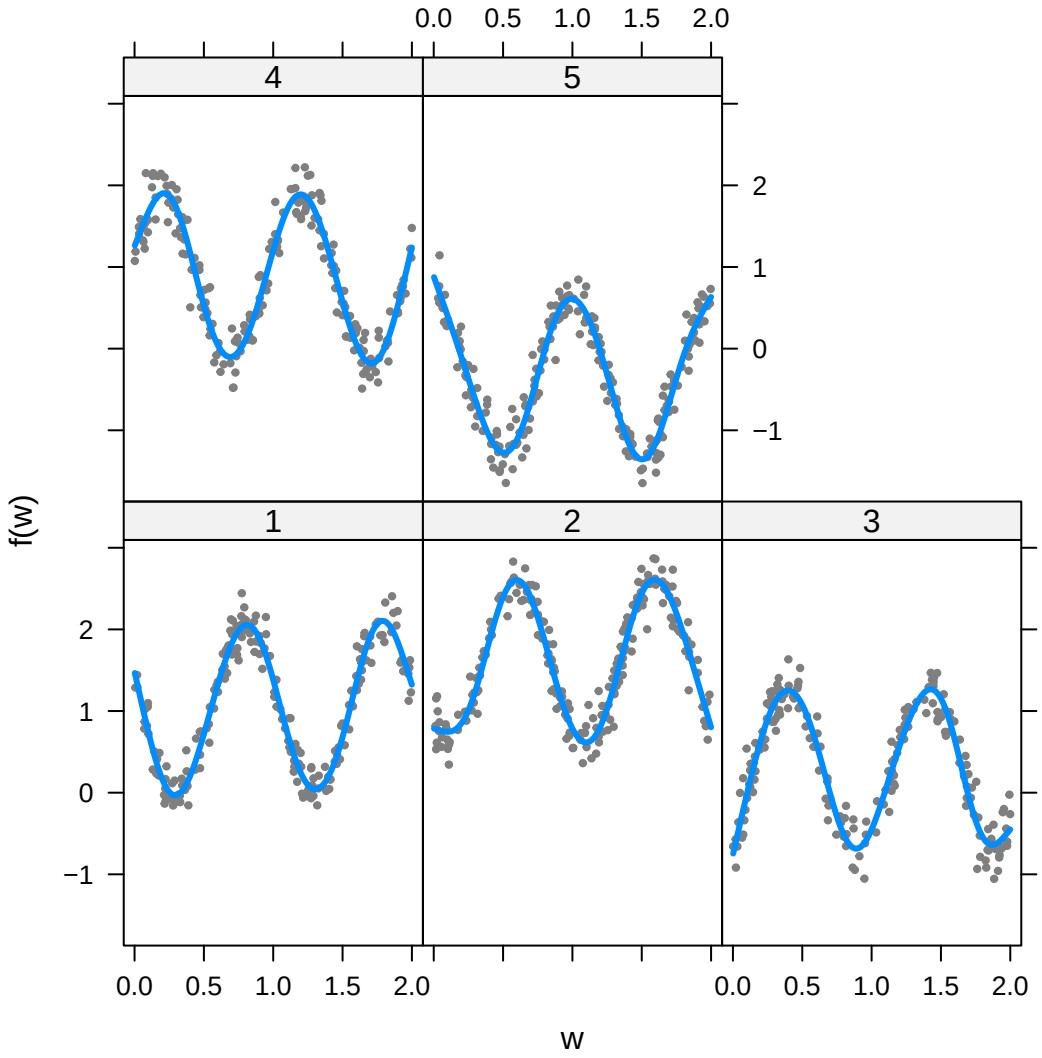
```
visreg(myfit, xvar="group")
```




```
visreg(myfit, xvar="x")
```




```
visreg(myfit, xvar="w", by="group")
```



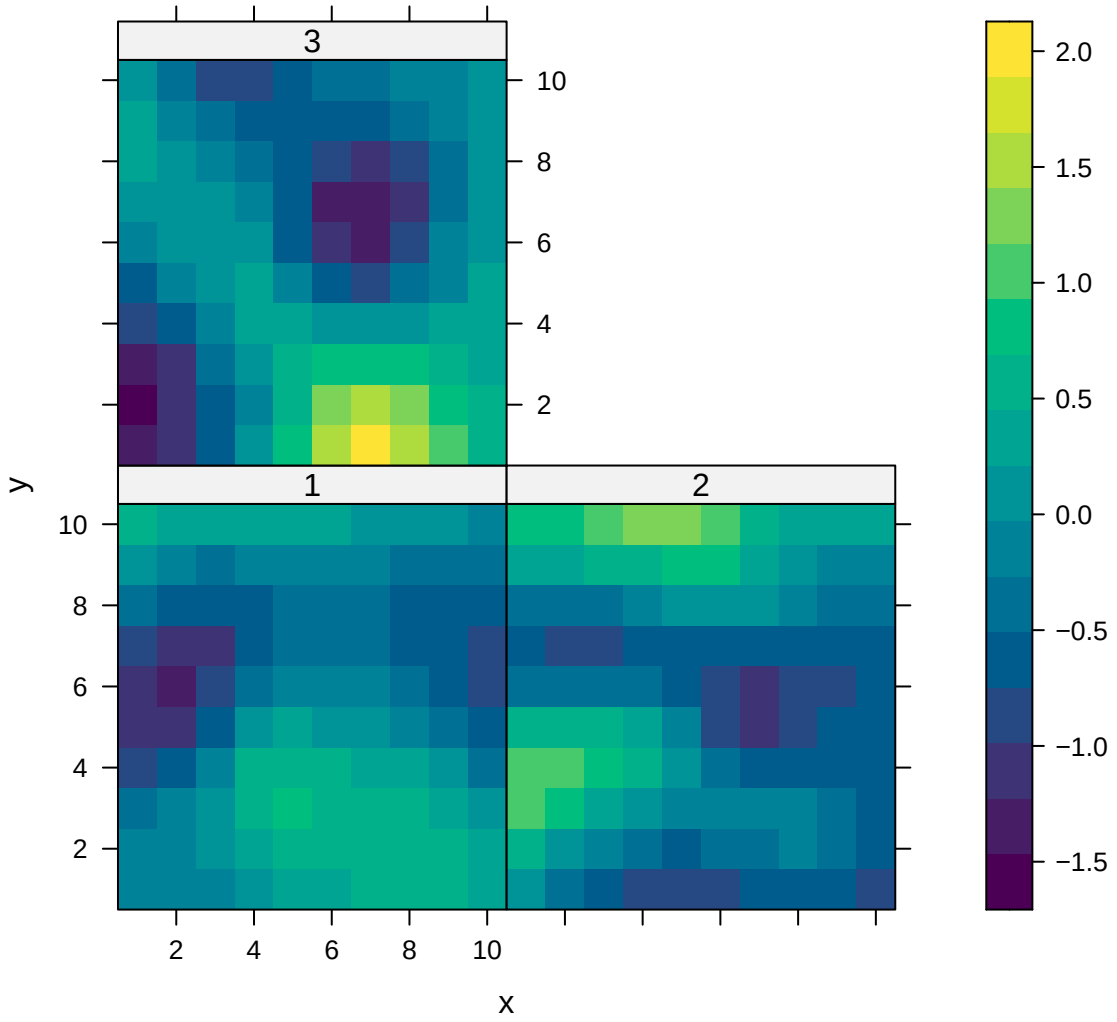
Alternatively, we can calculate derived quantities via Monte Carlo integration of the estimated density function:

```
# Predicted sample-weighted total
integrate_output(myfit)
#>            Estimate          Std. Error Est. (bias.correct) Std. (bias.correct)
#>         1617.352203            6.120346         1617.352203                  NA

# True (latent) sample-weighted total
sum( Data$z )
#> [1] 1606.42
```

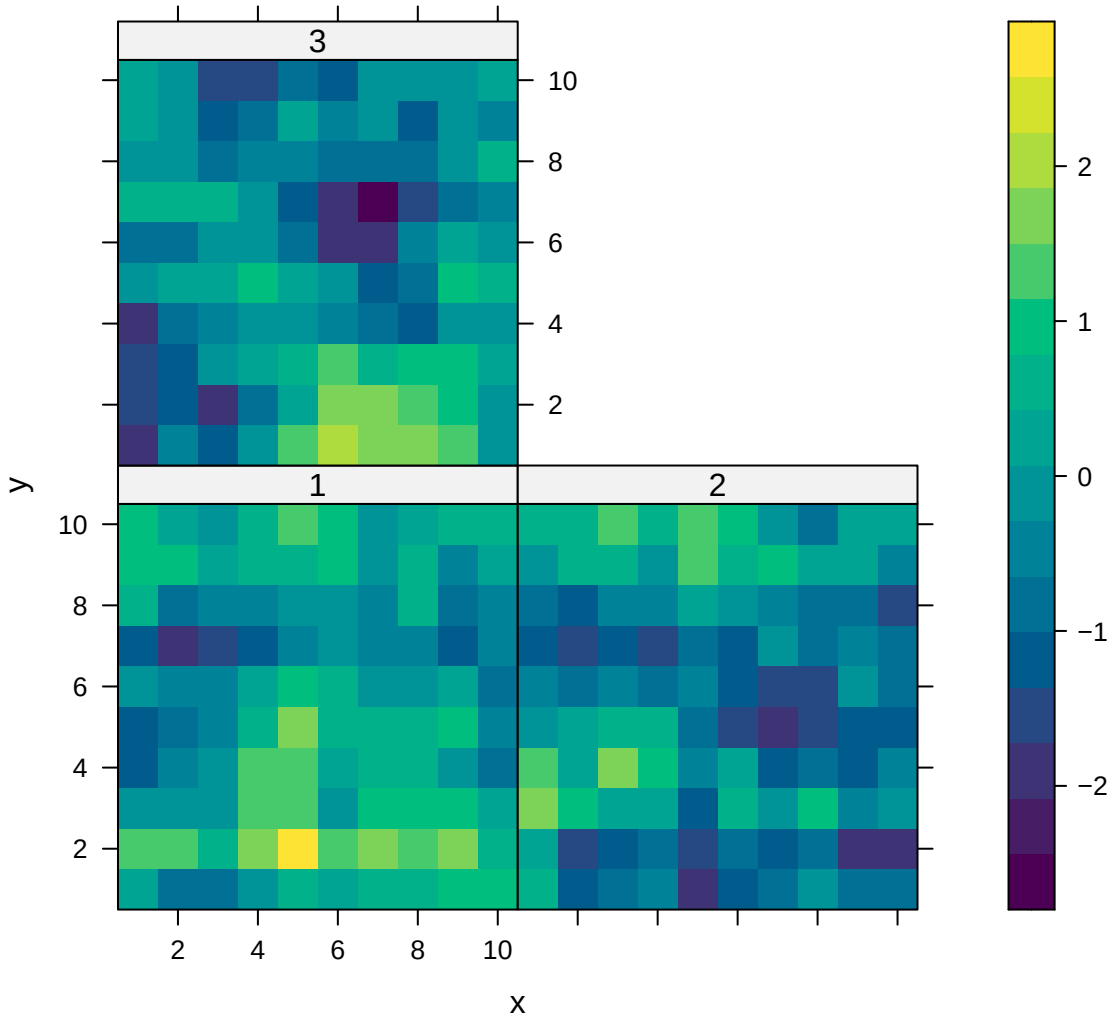
Similarly, we can fit a grouped 2D spline

```
# Simulate
R = exp(-0.4 * abs(outer(1:10, 1:10, FUN="-")) )
z = mvtnorm::rmvnorm(3, sigma=kronecker(R,R) )
```

```r
Data = data.frame( expand.grid(x=1:10, y=1:10, group=1:3), z=as.vector(t(z)))
Data$n = Data$z + rnorm(nrow(Data), sd=0.1)
Data$group = factor(Data$group)

# fit model
Formula = n ~ s(x, y, by=group)
myfit = fit( data = Data,
           formula = Formula,
           quiet = TRUE )
#>   0:     493.93839:  0.00000  0.00000  0.00000  0.00000  0.00000
#>   1:     462.60583: -0.0885898 0.00300351 0.00126287 -0.000998951 -0.301469
#>   2:     459.80222: -0.121472 0.00314320 -0.00168522 -0.00491012 -0.367792
#>   3:     459.20427: -0.123930 0.000784431 -0.0188557 -0.0247678 -0.437097
#>   4:     459.04539: -0.138775 -0.00145792 -0.0298865 -0.0368016 -0.411010
#>   5:     458.93854: -0.118424 -0.00434364 -0.0465828 -0.0553476 -0.422288
#>   6:     458.76002: -0.138256 -0.00767767 -0.0651806 -0.0757362 -0.424825
#>   7:     458.61600: -0.122871 -0.0114068 -0.0851827 -0.0972518 -0.417156
#>   8:     457.73354: -0.118070 -0.0517778 -0.305946 -0.329676 -0.480967
#>   9:     456.67248: -0.164836 -0.0945062 -0.527081 -0.556830 -0.417967
#>  10:     455.56303: -0.136110 -0.143456 -0.762094 -0.775900 -0.463110
#>  11:     455.42749: -0.134391 -0.208422 -0.892343 -0.809016 -0.458241
#>  12:     455.40995: -0.130910 -0.243191 -0.901955 -0.874981 -0.458835
#>  13:     455.40951: -0.133401 -0.247578 -0.976155 -0.885641 -0.463363
#>  14:     455.40859: -0.128031 -0.249378 -0.982345 -0.872039 -0.459744
#>  15:     455.40422: -0.130846 -0.252754 -0.975455 -0.872604 -0.459878
#>  16:     455.40309: -0.130849 -0.257382 -0.959812 -0.871080 -0.459657
#>  17:     455.40286: -0.131178 -0.247567 -0.954150 -0.859254 -0.460067
#>  18:     455.40278: -0.131080 -0.247488 -0.952009 -0.865909 -0.459589
#>  19:     455.40275: -0.131014 -0.254224 -0.953539 -0.864734 -0.459756
#>  20:     455.40273: -0.131065 -0.251055 -0.953407 -0.863894 -0.459786
#>  21:     455.40273: -0.131062 -0.251121 -0.953341 -0.864057 -0.459774
#>  22:     455.40273: -0.131062 -0.251128 -0.953347 -0.864048 -0.459775

# compute partial dependence plot
mypartial = partial( object = myfit,
                     pred.var = c("x","y","group"),
                     pred.fun = \(object,newdata) predict(object,newdata),
                     train = Data,
                     approx = TRUE )

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

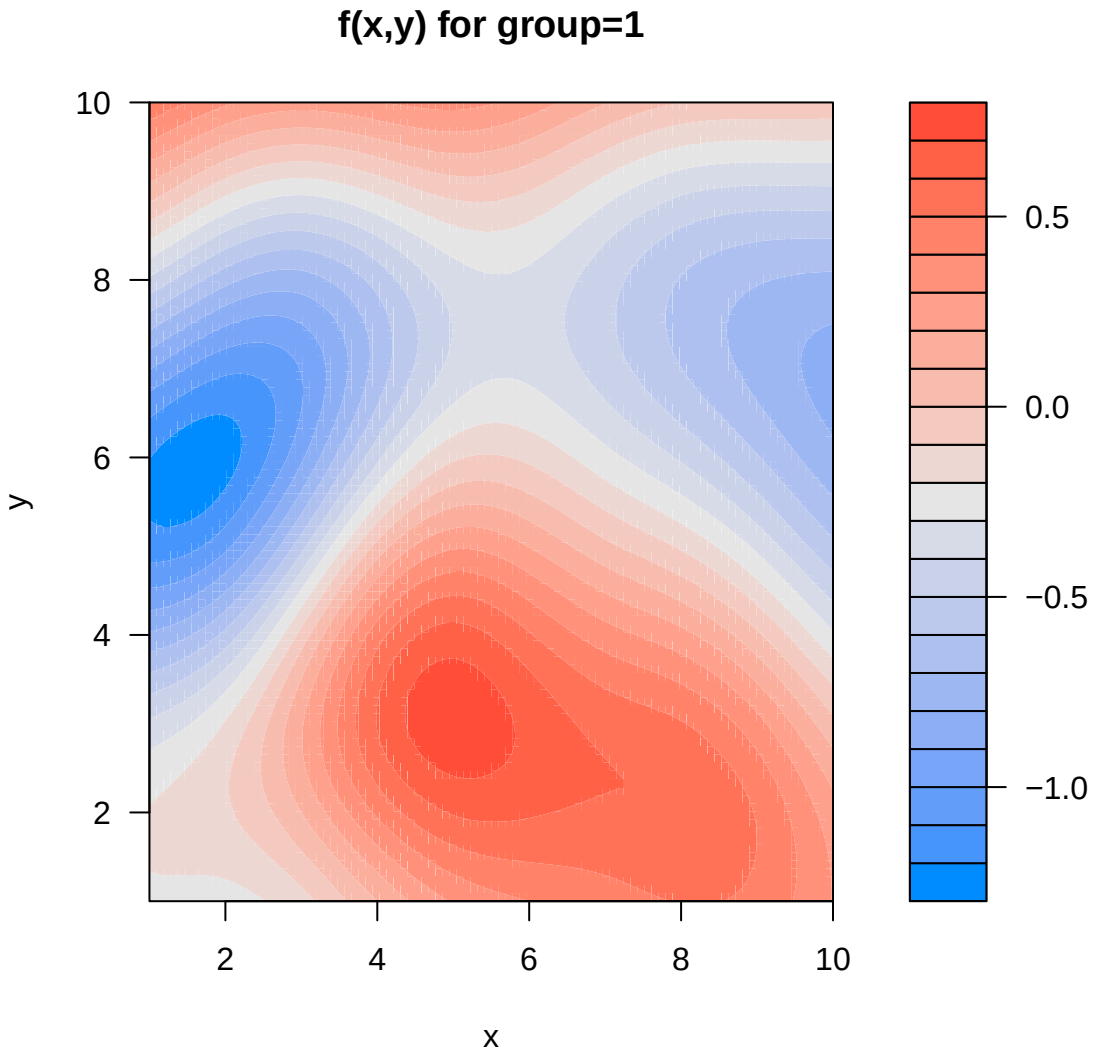



```
# Lattice plot of true values
mypartial$yhat = Data$z
plotPartial( mypartial )
```



We can again use `visreg` to visualize response surfaces, although it doesn't seem possible to extract a grouped spatial term, so we here show only a single term:

```
out = visreg2d( myfit, "x", "y", cond=list("group"=1), plot=FALSE )
plot( out, main="f(x,y) for group=1")
```


f(x,y) for group=1
10
8
6
4
2
y
2
4
6
8
10
x
0.5
0.0
−0.5
−1.0