mgcViz: scalable visualizations for GAMs

Matteo Fasiolo (University of Bristol, UK)

Joint work with:

Simon N. Wood (University of Bristol, UK)
Yannig Goude (EDF R&D)
Raphael Nedellec (EDF R&D)

matteo.fasiolo@bristol.ac.uk

October 19, 2017

Intro to mgcViz

`mgcViz` is meant to improve and extend the visualisations offered by `mgcv`.

We want to achieve more:

- **scalability**: faster plotting for large GAMs (e.g. `bam` output). Achieved using binning/discretization or sub-sampling when n large.
- **flexibility**: easier to customize plots (e.g. colours, line types, etc). Achieved using layering system (similar to `ggplot2`).
- **extensibility**: easier to extend existing plotting methods. Extensions implemented as new layers.
- **interactivity**: point-and-click interactivity and manipulation of 3D objects. Achieved using `plotly`, `rgl` and `shiny` packages.

In addition, `mgcViz` offers more tools for model checking or buiding.

Intro to mgcViz

`mgcViz` basics: let `fit` be the output of `gam()` or `bam()`.

First thing to do is to convert it to a `"gamViz"` object:

```
fit <- getViz(fit, nsim = 50)
```

`nsim` number of simulated responses data sets. Useful for model-checking.

All methods from `mgcv` (e.g. `summary`) should still work, but some over-written. Examples:

```
plot.gam(fit) # Call mgcViz::plot.gam not mgcv::plot.gam
qq.gam(fit)   # Call mgcViz::qq.gam not mgcv::qq.gam
```

If you want to use methods from `mgcv` you need to be explicit:

```
mgcv::plot.gam(fit) # Calls mgcv function
```

Visualizing the smooth effects

Suppose formula for `fit` is `y ~ x0 + s(x1) + s(x2, x3)`.

You can extract smooth effects by doing:

```
ef1 <- sm(fit, 1) # class(ef1) contains "mgcv.smooth.1D"
```

You can produce plotting objects:

```
pl1 <- plot(ef1) # calls plot.mgcv.smooth.1D()
```

`pl1` is of class `"plotSmooth"` and contains:

- `$ggObj` which is a `ggplot2::ggplot` object;
- `$data` a **list** of data to be used by the layers.

If you evaluate `pl1` on console you'll get an empty plot.
You need to add layers!

Visualizing the smooth effects

Adding layers:

```
pl1 <- pl1 +
       l_points() + l_fitLine() +   # mgcViz layers
       xlim(1, 2) + theme_dark()    # ggplot2 layers
```

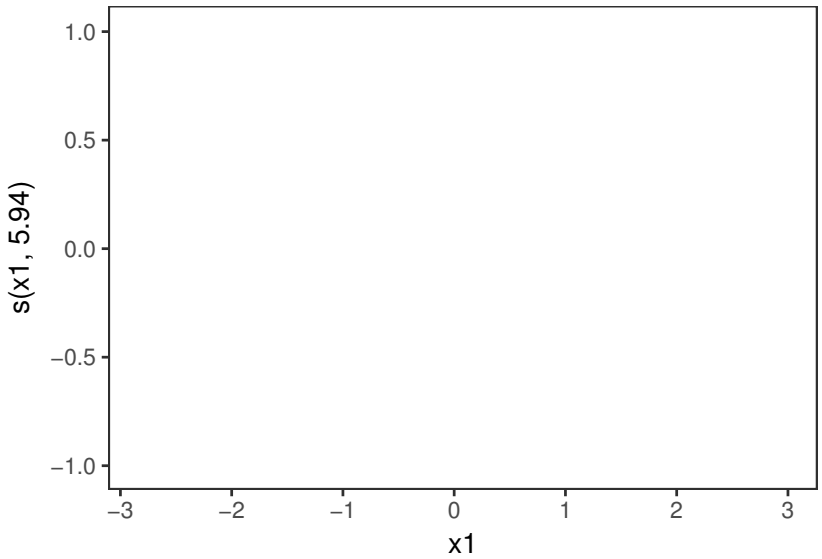
Layers from `mgcViz` start with `l_` and return object of class `"gamLayer"`.
You can do:

```
listLayers(pl) # lists all available layers
```

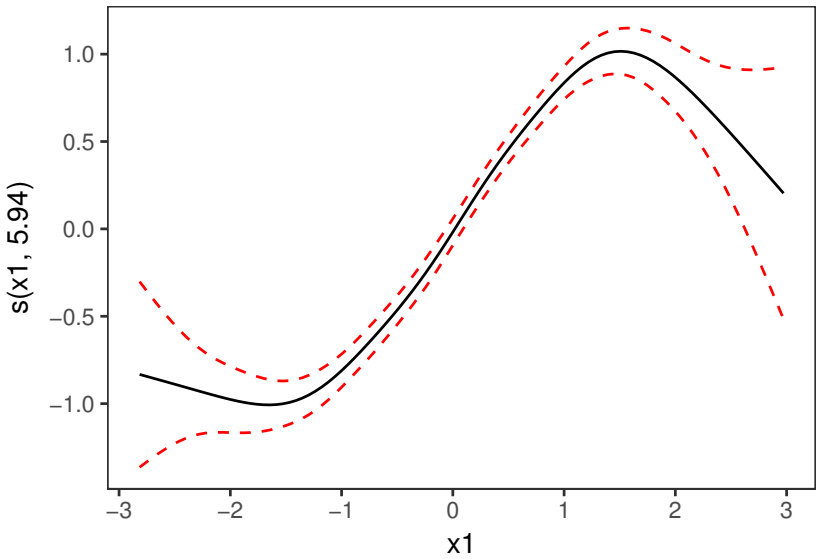
Above "+" calls `mgcViz:::"+.gg" = function(e1, e2)`, where `class(e1)` contains `"plotSmooth"`.
`e1` is modified using `e2` and returned.

Visualizing the smooth effects

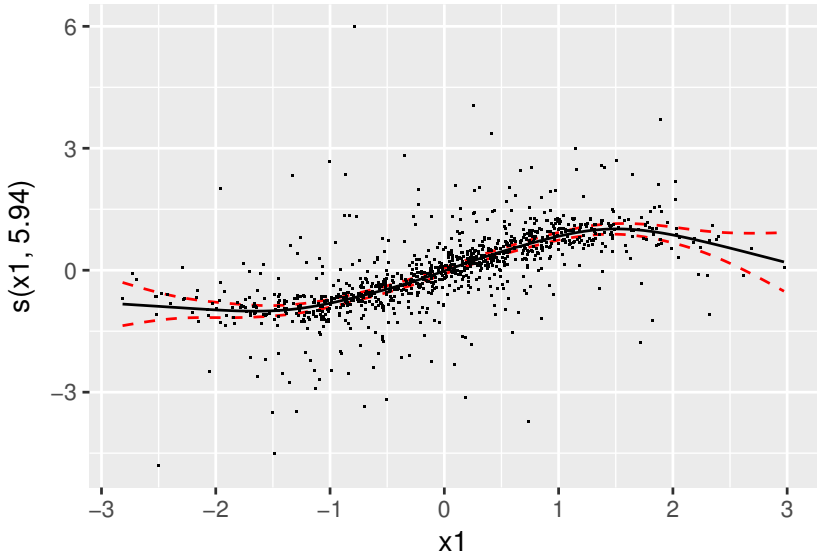
o <– plot(sm(b, 1))



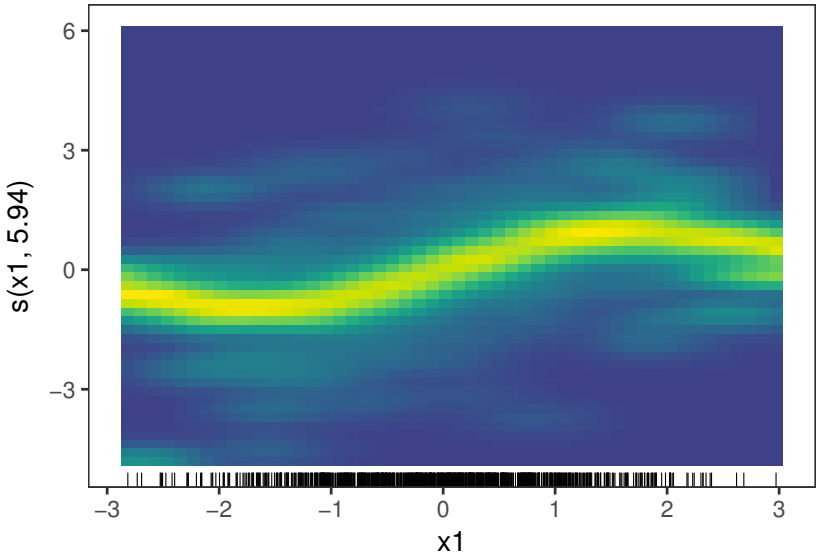
o + l_fitLine() + l_ciLine(col=2)



o + ... + l_points() + theme_get()



o + l_dens(type = "cond") + l_rug()



Visualizing the smooth effects

Arguments of `mgcViz` layers, e.g. `args(l_ciLine)`:

- `mul = 2` : means we use 2*sigma conf. int.;
- `...` : see `?ggplot2::geom_line`.

Recall model `y ~ x0 + s(x1) + s(x2, x3)`. To plot `s(x2, x3)`:

```
ef2 <- sm(fit, 2) #class(ef2) contains "mgcv.smooth.2D"
pl2 <- plot(ef2)  #class(pl2) contains "plotSmooth" "2D"
plot( sm(fit, 2) )  #Faster
```

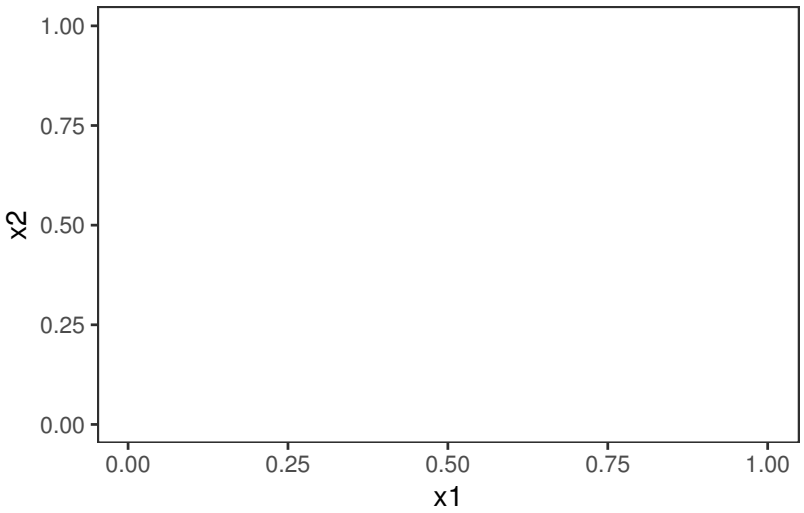
Add layers to 2D plot:

```
pl2 + l_fitContour() + l_rug() # Ok
pl2 + l_fitLine()              # Gives error!
```

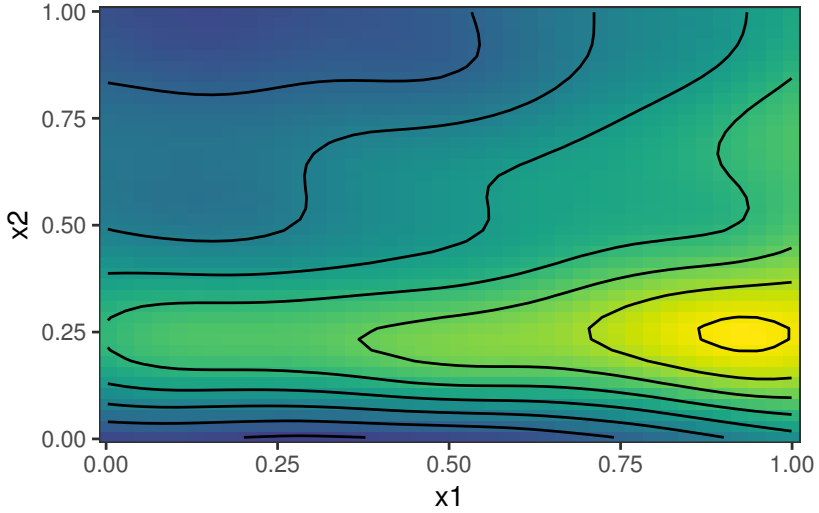
because `"l_fitLine"` not in `listLayers(pl2)`.

Visualizing the smooth effects

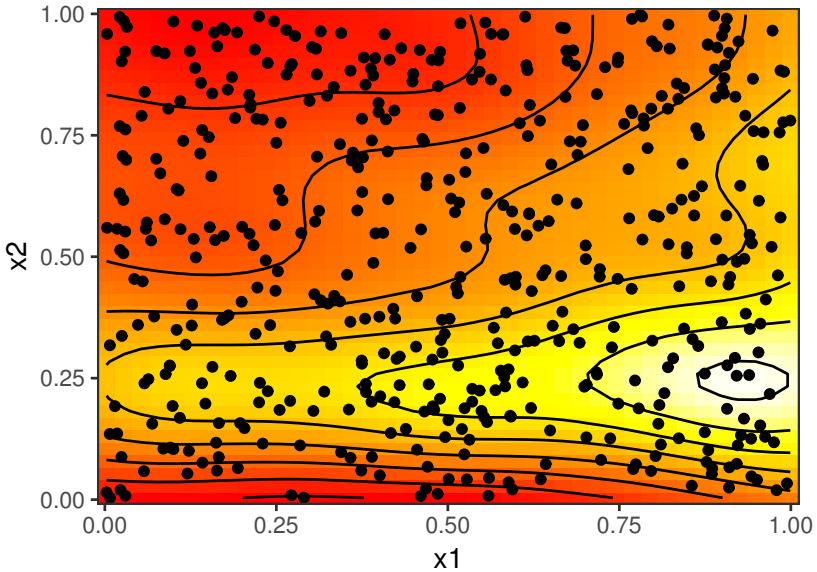
o <- plot(sm(b, 2))



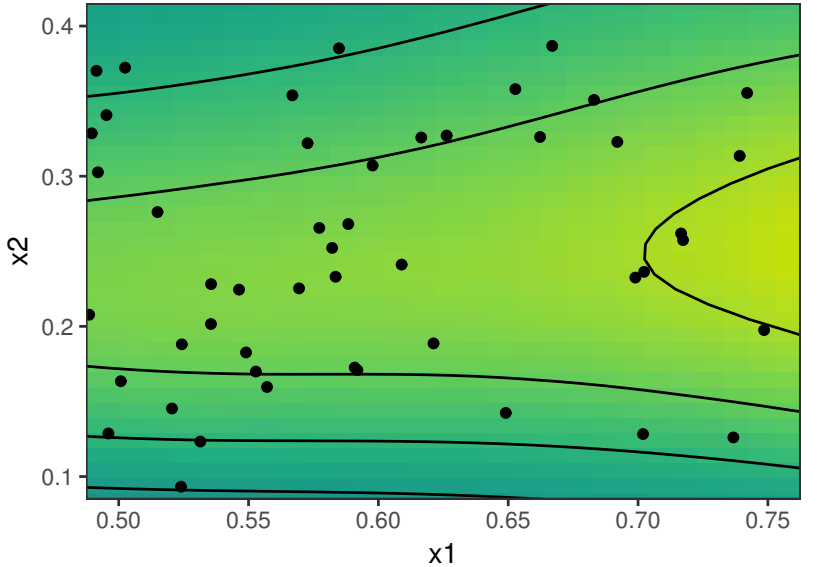
o + l_fitRaster() + l_fitContour()



o + l_points() +
scale_fill_gradientn(colours = heat.colors(50))

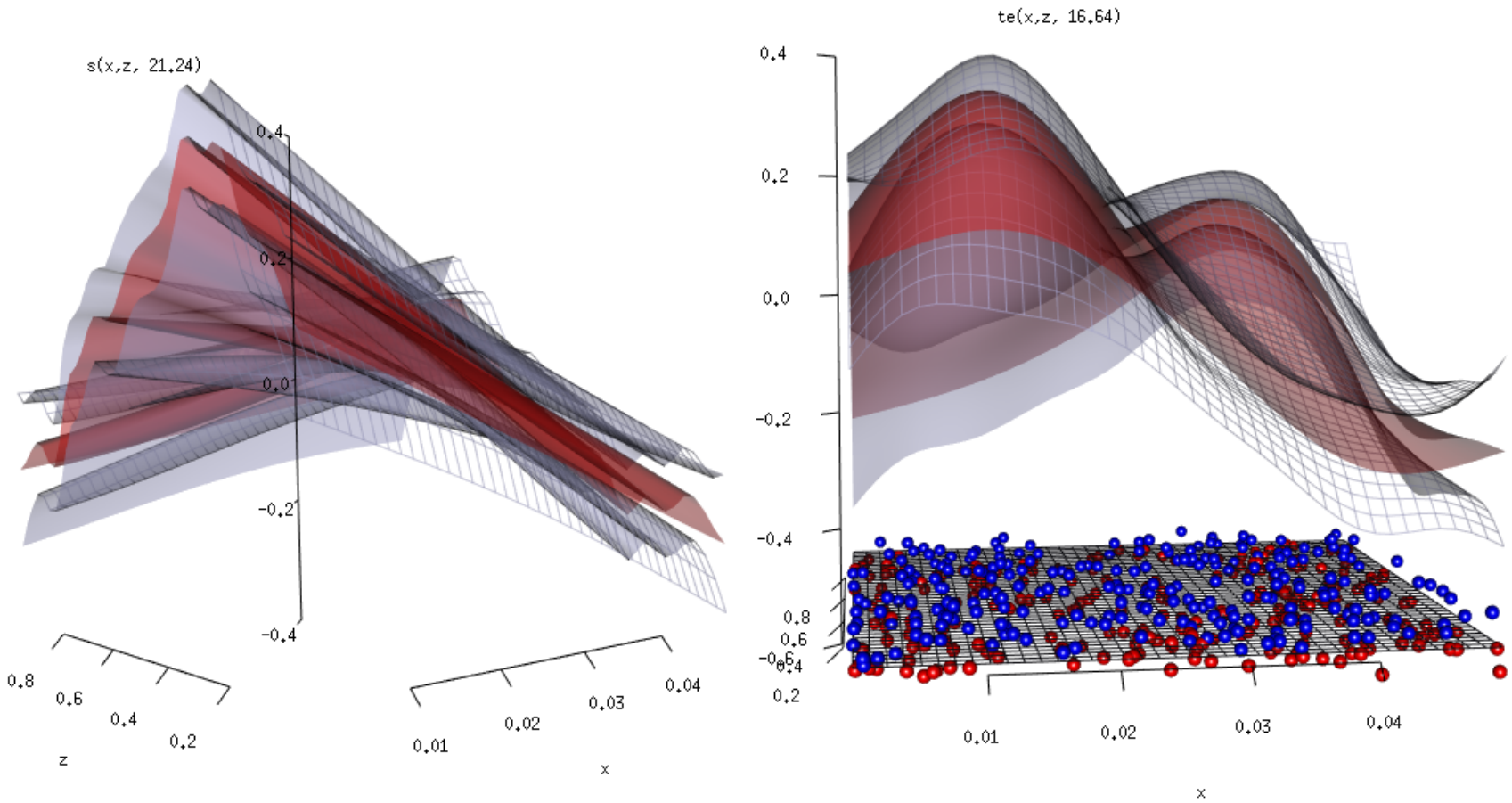


o + coord_cartesian(xlim =
c(0.5, 0.75), ylim = c(0.1, 0.4))



Visualizing the smooth effects

We can get interactive 3D effect plots using `plotRGL(sm(fit, 2))`.



Visualizing the smooth effects

Notice that `plotRGL` **not** layered at the moment:

```
plotRGL(o, se = TRUE, n = 40, residuals = FALSE, ...)
```

and returns `NULL`, not a `"plotSmooth"` object. See `?rgl`.

Back to layered plots. How to plot several on one page?

```
pl1 <- plot(sm(fit, 1)) + l_ciPoly() + l_fitLine()
pl2 <- plot(sm(fit, 2)) + l_fitRaster() + l_points()

# Extract ggplot objects
ml <- list(pl1[["ggObj"]], pl2[["ggObj"]])

# Two plots side by side
library(gridExtra)
grid.arrange(grobs = ml, ncol = 2)
```

Visualizing the smooth effects

Faster, but less flexible, approach using new `plot.gam` function:

```
fit <- getViz(fit, nsim = 50)
plot(fit, select = c(1, 3))
```

plots 1st and 3rd effect asking "Hit <Return> to see next plot".
All on one page using:

```
oo <- plot(fit)       # class(oo) includes "plotGam"
print(oo, pages = 1) # Calls print.plotGam()
```

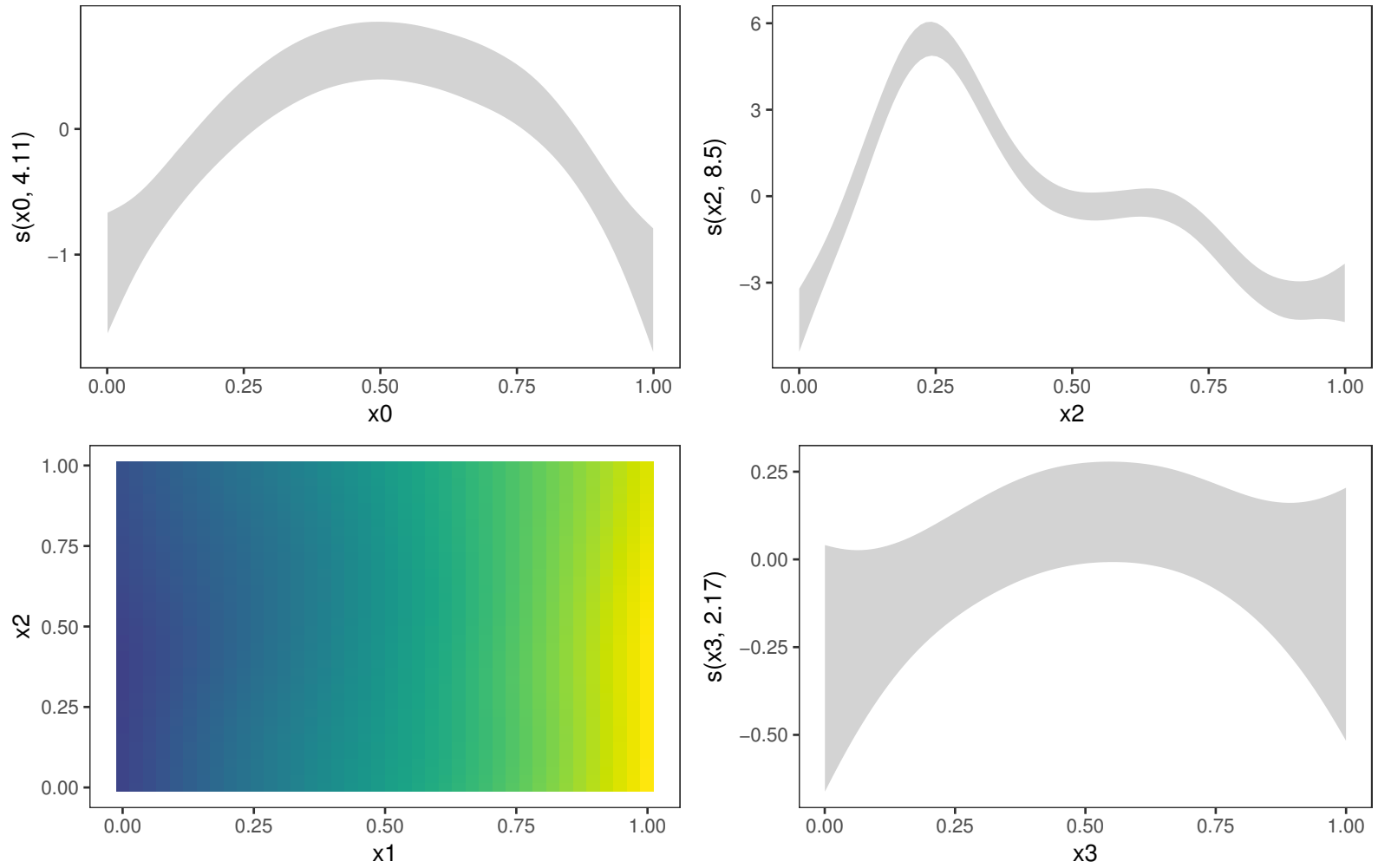
You can still add layers:

```
plot(fit) + l_fiRaster() + l_rug()
```

NB `plot.gam` does **no error checking** using `listLayers`.

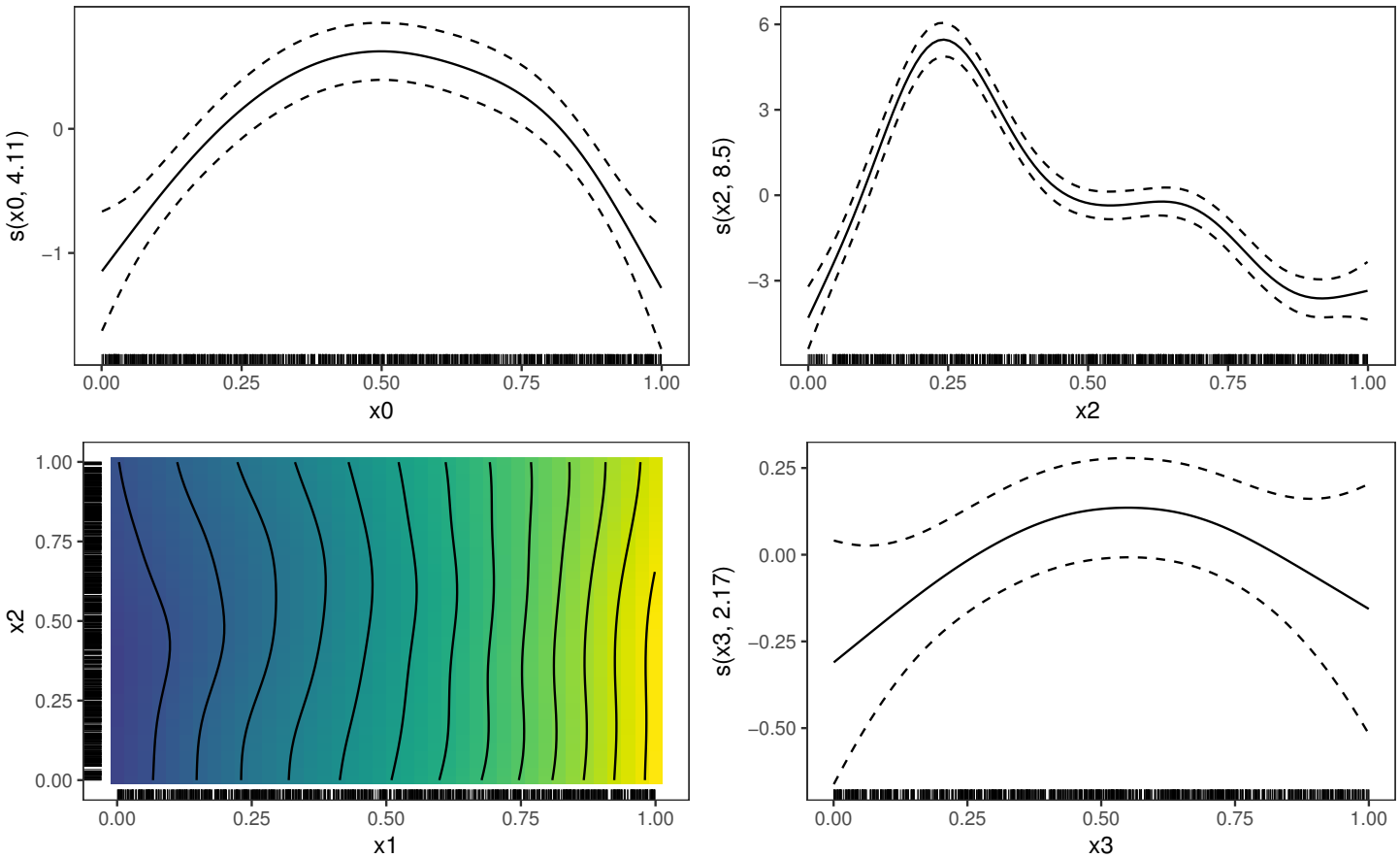
Visualizing the smooth effects

Output of
`print(plot(fit) + l_ciPoly() + l_fitRaster(), pages = 1)`.



Visualizing the smooth effects

Output of `print(plot(fit), pages = 1)`.



Default layers added by `print.plotGam`. To get empty plot:
`print(plot(fit), pages = 1, addLay = FALSE)`

Visualizing the smooth effects

So far `mgcViz` can plot:

- standard 1D, 2D (isotropic, tensor, soap) and slices of HD effects;
- factor smooth interactions (`bs = "fs"` or by-variable smooths).

Work-in-progress:

- smooth on sphere;
- Markov Random Fields;
- parametric terms.

Apart from `rgl`, interactivity via `plotly`:

```
pl <- plot( sm(fit, 1) ) + l_rug() + l_fitLine()

library(plotly)
ggplotly(pl)     # Produces an interactive plot
```

Moving to model checking and building now!

Interactive visual model-building

As of version 1.8 `mgcv` provides tools for fitting GAMLSS models.
For example the `mgcFam::shash` family of Jones and Pewsey (2009):

```
fit <- gam(list(y ~ s(x1),       # Location
                ~ x5 + s(x2), # Scale
                ~ s(x0) + x7, # Asymmetry (skewness)
                ~ s(x2, x4) ) # Tails (kurtosis)
           family = shash )
```

How to choose which effects to include? Exhaustive search not possible:

1. too many possible models;
2. model fitting slow (`bam` unavailable).

Better to start with a simple model and build it up interactively.
`mgcViz` offers tools for helping you doing this.

Interactive visual model-building

Continuing with the `shash` example:

```
fit <- gam(list(y ~ ?, ~ ?, ~ ?, ~ ?), family = shash)
```

For each of the four parameters we need to choose:

1. which variables to include;
2. the complexity of each effect.

Main model checking methods in `mgcViz` are:

- `check1D`/`check2D` + available layers;
- `check.gam` (similar to `gam.check` in `mgcv`);
- `qq.gam` (overloads `mgcv::qq.gam`).

Interactive visual model-building

Let `fit` be output of `gam` and consider a variable `x3`.
Should we include `x3` in any of the four models?
`check1D` lets you visualise the residuals along `x3`:

```
fit <- gamViz(fit, nsim = 100)
o <- check1D(fit, x=X[["x3"]]) # class(o) = "plotSmooth"
```

or simply `x = "x3"` if `x3` already in model formula.
Object `o` is empty, but we can add layers:

```
o + l_points() + l_rug() # as for smooth effects
o + l_densCheck()        # new
o + l_gridCheck1d()      # new
```

Do `listLayers(o)` for list of layers.
As before `o$ggObj` contains `gglot2::ggplot` object.

Interactive visual model-building

`l_densCheck` compares empirical and theoret. residuals distribution.

Generates heat-map of:

$$\delta(x, r) = \texttt{dFun}\{\hat{p}(r|x) - \phi(r|x)\},$$

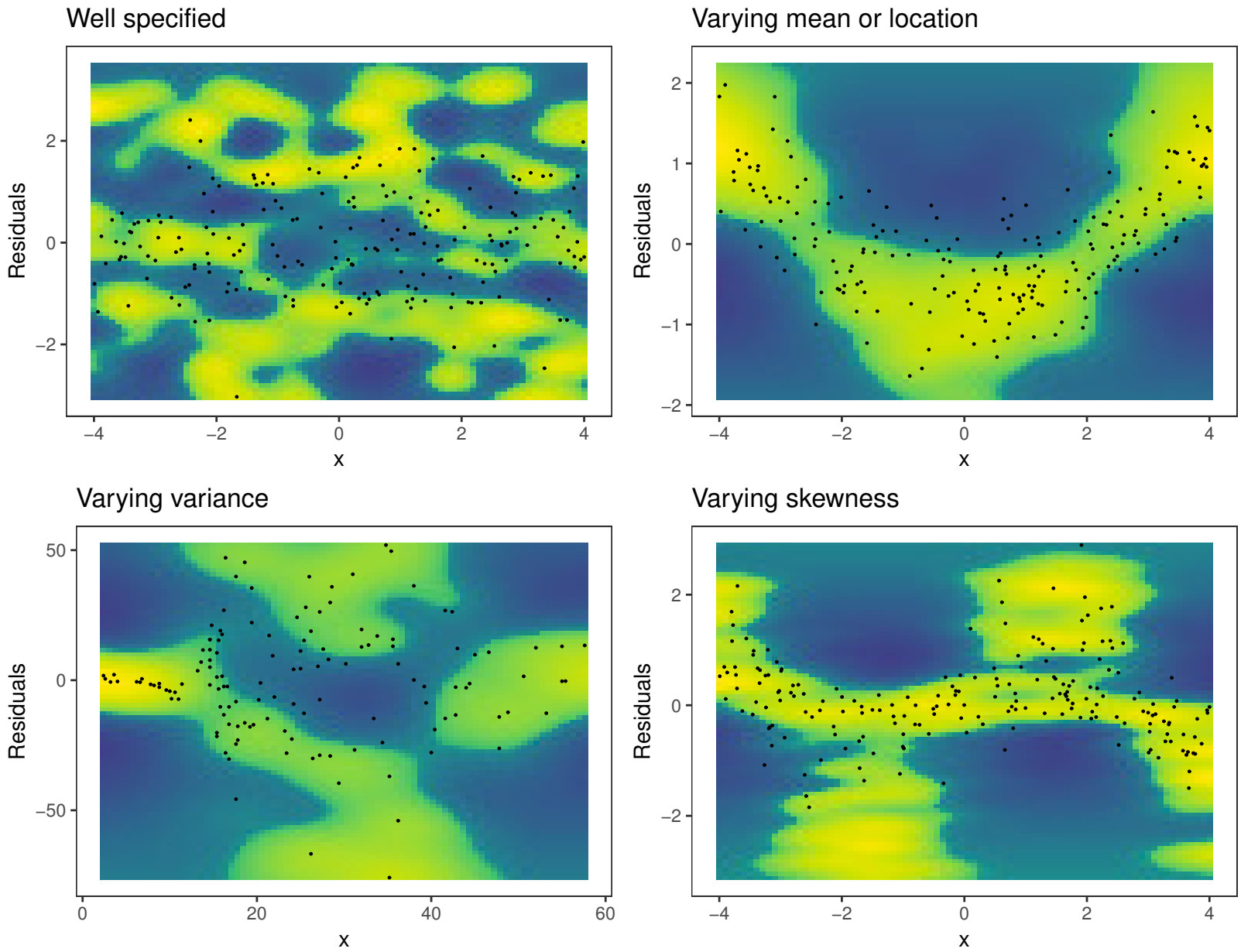
where

- $\hat{p}(r|x)$ fast binned k.d.e. of $p(r|x)$;
- $\phi(r|x)$ is theoretical distribution (currently only $N(0, 1)$);
- `dFun` is a distance function. By default:
  `dFun <- function(emp, th) abs(sqrt(em)-sqrt(th))^(1/3);`

Useful for detecting:

- anomalies (e.g. outliers);
- non-constant variance, skewness etc.

Interactive visual model-building



Interactive visual model-building

`l_gridCheck1D` does the following:

1. bins the residuals r along x;
2. summarizes the residuals in each bin using `gridFun()`;

Step 1 and 2 are used also on simulated data to obtain conf. int..

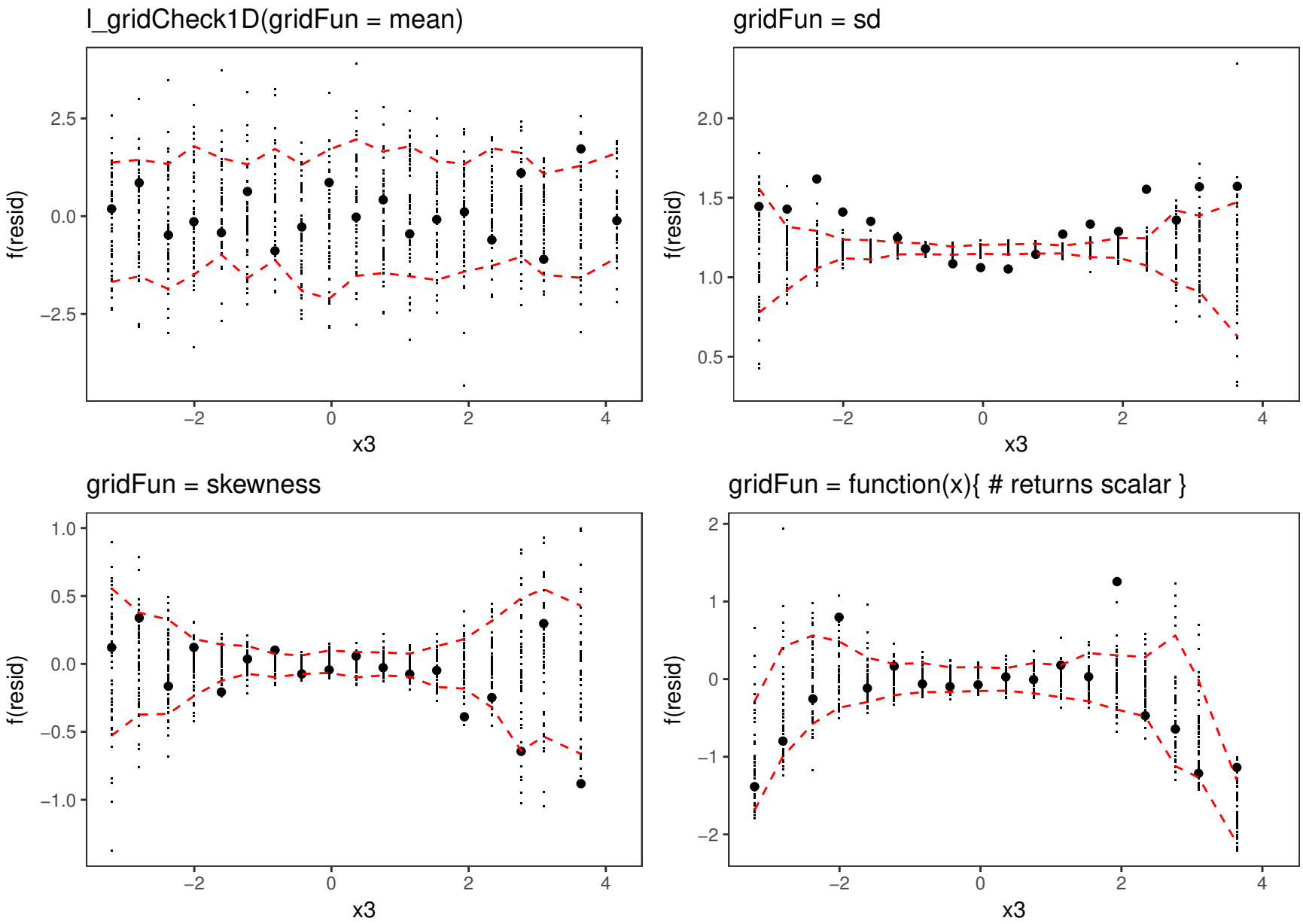
`l_gridCheck2D` generalizes this to 2D:

```
o <- check2D(fit, x1 = "x3", x2 = "x7")
o + l_gridCheck2D(gridFun = sd)
```

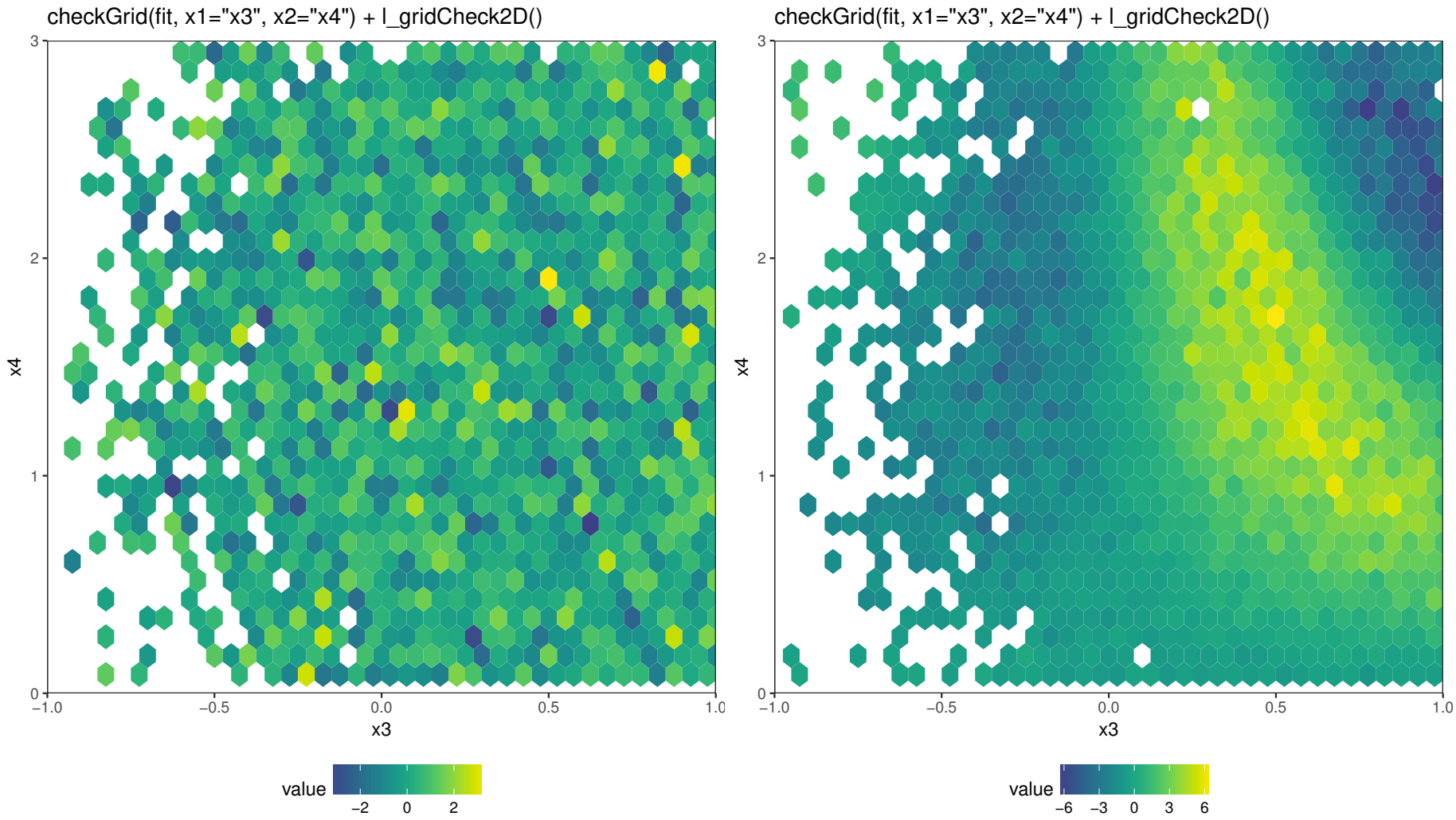
Main difference is that we can't add confidence intervals, but can still used simulations to standardize:

```
std_stat = (obs_stat-mean(sim_stat)) / sd(sim_stat)
```

Interactive visual model-building



Interactive visual model-building




```
check2D(fit, x1="x3", x2="x4") +
        l_gridCheck2D(gridFun=sd, bw=c(0.2, 0.5))
```

Interactive visual model-building

QQ-plots produced by `qq.gam(fit)`. New features:

1. `qq.gam(fit)[[1]]` contains `ggplot` object;
2. scales to large data sets (if `discrete = TRUE`);
3. interactivity by `shine(qq.gam(fit))`.

Also available `check.gam` similar to `gam.check`:

```
ck <- check(fit) # calls check.gam()
```

where *ck* is a list of `ggplots`.

`check.gam` prints out useful info like before:

```
        k'  edf k-index p-value
s(x0) 9.00 2.32    1.00    0.47
s(x1) 9.00 2.31    0.97    0.33
.............................
```

References I

Jones, M. and A. Pewsey (2009). Sinh-arcsinh distributions. *Biometrika 96*(4), 761–780.