Simple density surface models

Solution 4, Intermediate Distance Sampling workshop, CREEM, 2018

Here is a "solution" for practical 4. As with any data analysis, there is no correct answer, but this shows how I would approach this analysis. The analysis here is conditional on selecting a detection function in the previous exercises; I've chosen `df_hn` and `df_hr_ss_size` that we saved previously (see detection function solutions).

Much of the text below is as in the exercise itself, so it should be relatively easy to navigate.

Additional text and code is highlighted using boxes like this.

Load the packages and data

```
library(Distance)
```

```
## Loading required package: mrds
```

```
## This is mrds 2.2.0
## Built: R 3.5.1; ; 2018-08-03 20:22:18 UTC; windows
```

```
##
## Attaching package: 'Distance'
```

```
## The following object is masked from 'package:mrds':
##
##     create.bins
```

```
library(dsm)
```

```
## Loading required package: mgcv
```

```
## Loading required package: nlme
```

```
## This is mgcv 1.8-24. For overview type 'help("mgcv-package")'.
```

```
## Loading required package: numDeriv
```

```
## This is dsm 2.2.16
## Built: R 3.4.2; ; 2017-11-02 15:05:30 UTC; windows
```

```
library(ggplot2)
library(knitr)
```

Loading the `RData` files where we saved our results:

```
load("sperm-data.RData")
load("df-models.RData")
```

Pre-model fitting

Before we fit a model using `dsm()` we must first remove the observations from the spatial data that we excluded when we fitted the detection function – those observations at distances greater than the truncation.

```
obs <- obs[obs$distance <= df_hn$ddf$meta.data$width, ]
```

Here we've used the value of the truncation stored in the detection function object, but we could also use the numeric value (which we can also find by checking the model's `summary()`).

Also note that if you want to fit DSMs using detection functions with different truncation distances, then you'll need to reload the `sperm-data.RData` and do the truncation again for that detection function.

Fitting DSMs

Using the data that we've saved so far, we can build a call to the `dsm()` function and fit out first density surface model. Here we're only going to look at models that include spatial smooths.

Let's start with a very simple model – a bivariate smooth of `x` and `y`:

```
dsm_nb_xy <- dsm(count~s(x,y),
                 ddf.obj=df_hn, segment.data = segs, observation.data=obs,
                 family=nb())
```

Note again that we try to have informative model object names so that we can work out what the main features of the model were from its name alone.

We can look at a `summary()` of this model. Look through the summary output and try to pick out the important information based on what we've talked about in the lectures so far.

```
summary(dsm_nb_xy)
```

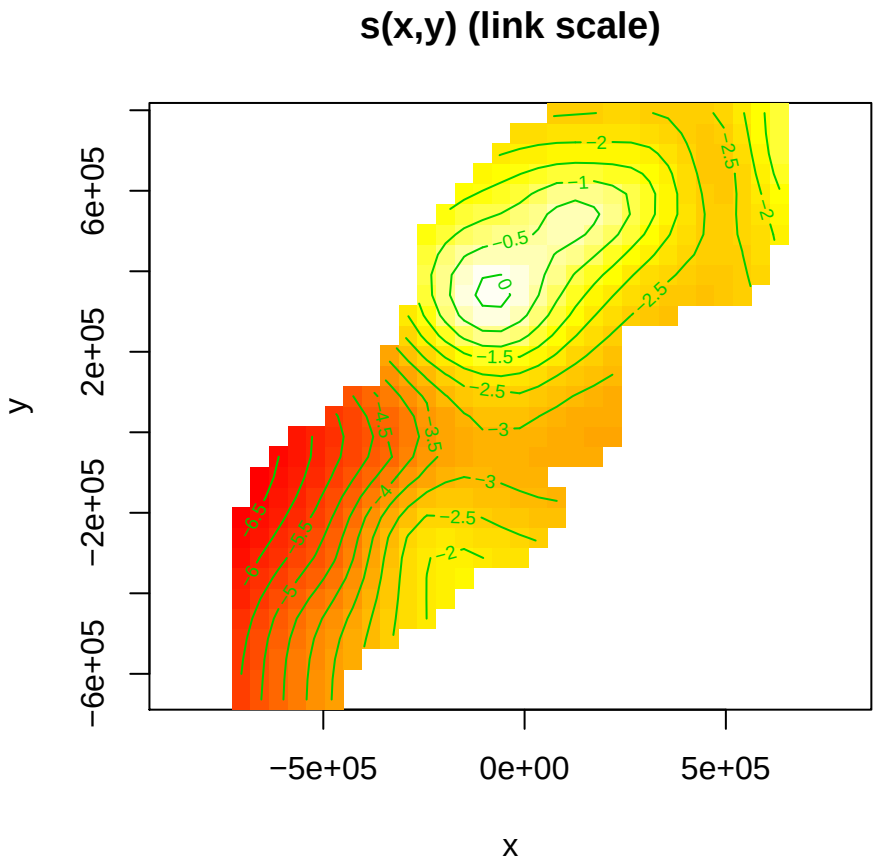
```
##
## Family: Negative Binomial(0.089)
## Link function: log
##
## Formula:
## count ~ s(x, y) + offset(off.set)
##
## Parametric coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -20.4207     0.2069  -98.67   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##          edf Ref.df Chi.sq  p-value
## s(x,y) 12.68   16.8  64.44 1.82e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.081   Deviance explained = 34.7%
## -REML = 285.45  Scale est. = 1         n = 949
```

Important things to look at here include the `EDF` column, the `R-sq.(adj)` and `Deviance explained`. Note that the significance of the terms isn't that useful at the moment, since we only have one smooth in the model (though if this wasn't significant we might be worried about including that variable!).

Visualising output

As discussed in the lectures, the `plot` output is not terribly useful for bivariate smooths like these. We'll use `vis.gam()` to visualise the smooth instead:

```
vis.gam(dsm_nb_xy, view=c("x","y"), plot.type="contour", too.far=0.1, main="s(x,y) (link scale)", asp=1)
```



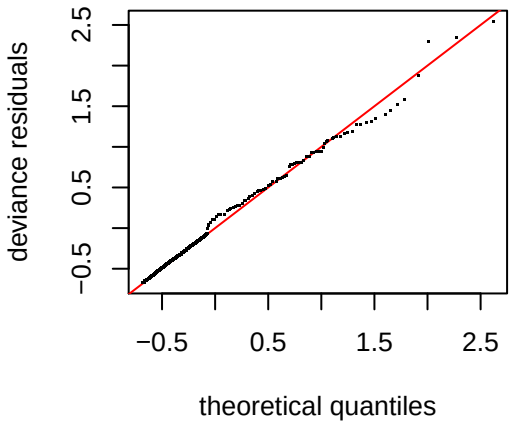
Notes:

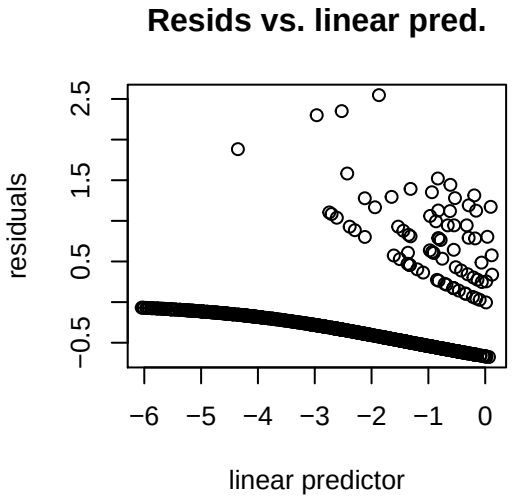
1. The plot is on the scale of the link function, the offset is not taken into account – the contour values do not represent abundance, just the "influence" of the smooth.
2. We set `view=c("x","y")` to display the smooths for `x` and `y` (we can choose any two variables in our model to display like this)
3. `plot.type="contour"` gives this "flat" plot, set `plot.type="persp"` for a "perspective" plot, in 3D.
4. The `too.far=0.1` argument displays the values of the smooth not "too far" from the data (try changing this value to see what happens.
5. `asp=1` ensures that the aspect ratio of the plot is 1, making the pixels square.
6. Read the `?vis.gam` manual page for more information on the plotting options.

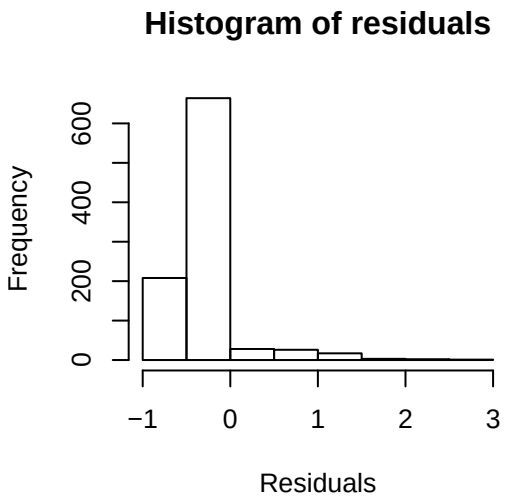
Checking the model

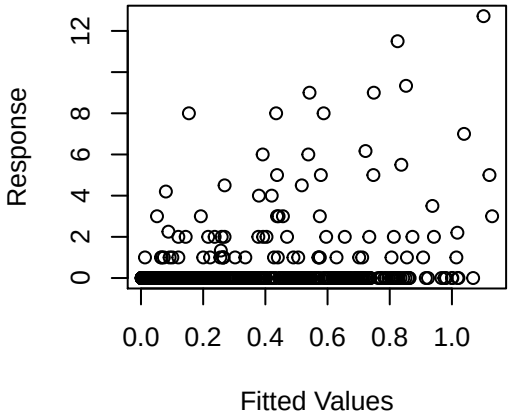
We can use the `gam.check()` and `rqgam.check` functions to check the model.

```
gam.check(dsm_nb_xy)
```



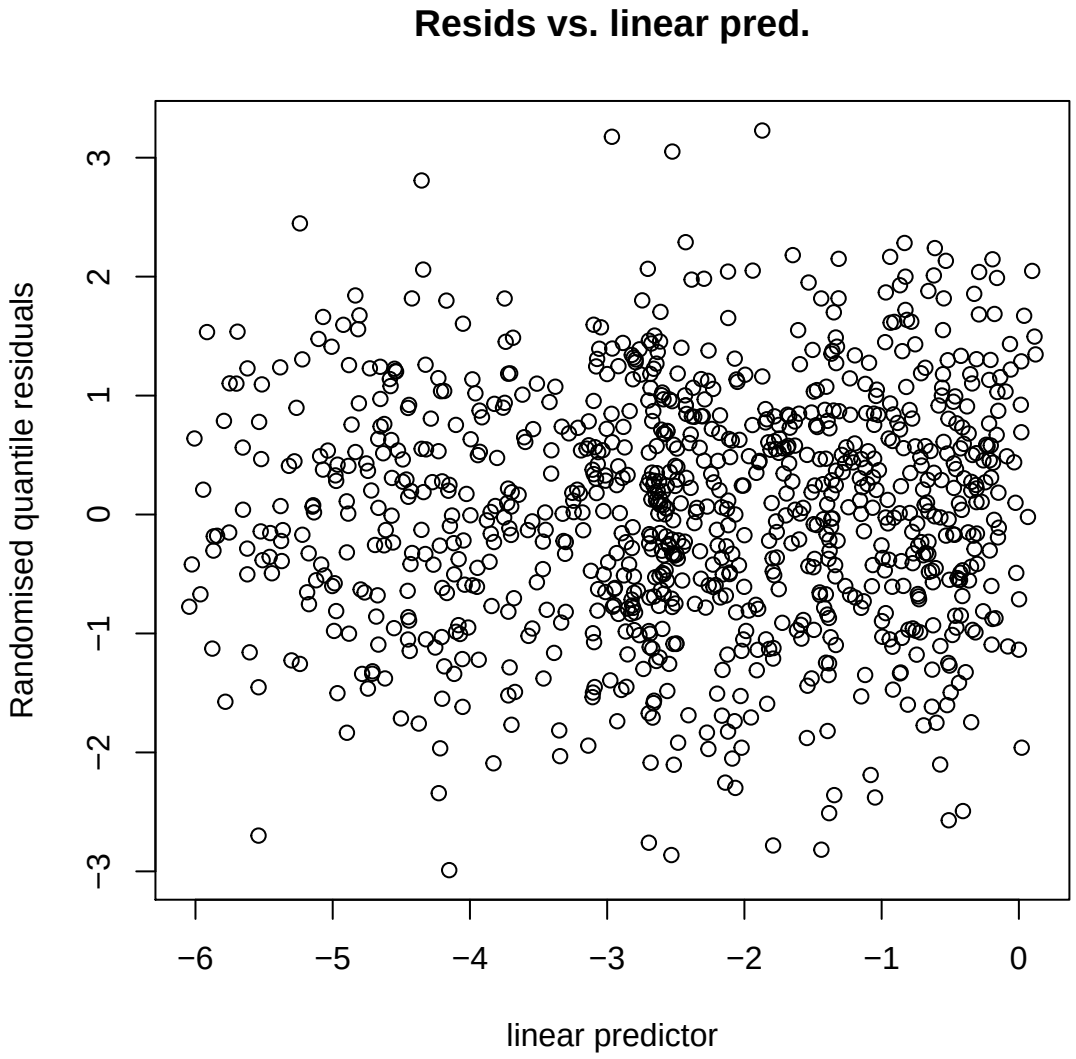







```
##
## Method: REML   Optimizer: outer newton
## full convergence after 5 iterations.
## Gradient range [-4.828208e-06,6.037089e-07]
## (score 285.4549 & scale 1).
## Hessian positive definite, eigenvalue range [1.156852,20.86974].
## Model rank =  30 / 30
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##          k'  edf k-index p-value
## s(x,y) 29.0 12.7     0.5  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
rqgam.check(dsm_nb_xy)
```



Remember that the left side of the `gam.check()` plot and the right side of the `rqgam.check()` plot are most useful.

Looking back through the lecture notes, do you see any problems in these plots or in the text output from `gam.check()`.

First looking at the `gam.check` plots, the Q-Q plot seems to show good agreement between theoretical and actual values when the residuals are small, but this performance degrades as the values get larger (though not horribly). This is reflected in the histogram below where we see the distribution is skewed. The left side plots are not terribly interpretable (more on that below), because the artifact of the zeros in the model (almost solid line of dots) it's hard to discern whether there is heteroskedasticity (non-constant variance).

The right side `rqgam.check` plots are much easier to interpret. The top right plot shows that we need not worry about non-constant variance here. Note that the left side doesn't tell us anything as the randomised quantile residuals are normally distributed (so the Q-Q looks okay and the histogram looks like a normal distribution, no surprises there).

Now looking at the text output underneath: although the `k-index` and `p-value` values are low, the value of `k'` (the basis complexity we set) is much higher than the value of `edf` (the effective degrees of freedom), so we don't need to worry about the basis being to small.

Setting basis complexity

We can set the basis complexity via the `k` argument to the `s()` term in the formula. For example the following re-fits the above model with a much smaller basis complexity than before:

```
dsm_nb_xy_smallk <- dsm(count~s(x, y, k=10),
                        ddf.obj=df_hn, segment.data = segs, observation.data=obs,
                        family=nb())
```

Compare the output of `vis.gam()` and `gam.check()` for this model to the model with a larger basis complexity.

Setting the basis size to be "too" small yields the following output:

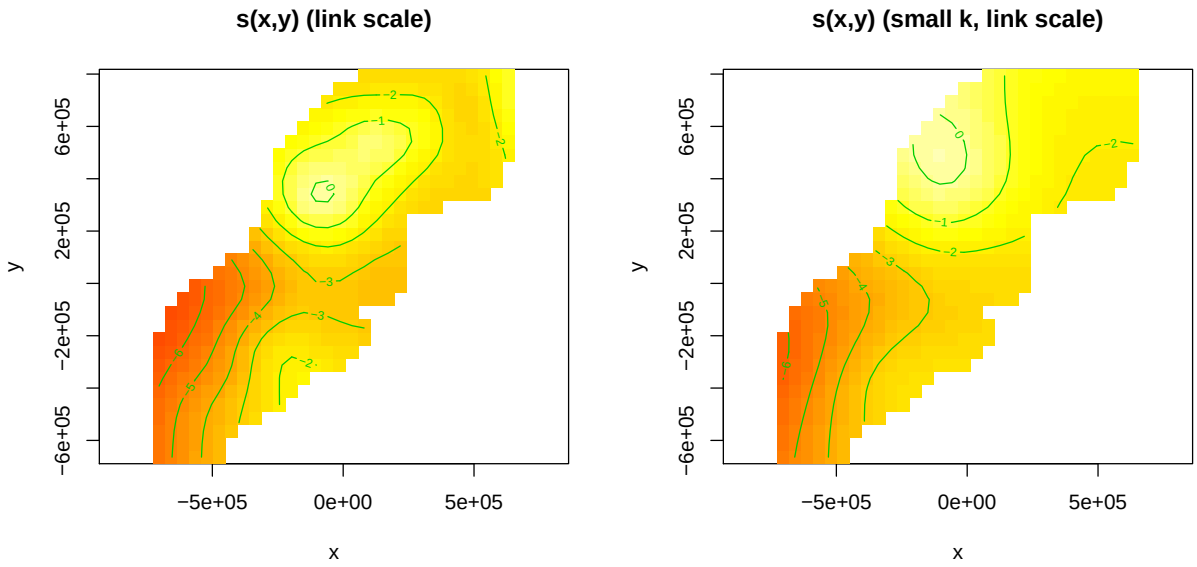
```
# here we suppress the plot using fig.keep="none" as a chunk option
gam.check(dsm_nb_xy_smallk)
```

```
##
## Method: REML   Optimizer: outer newton
## full convergence after 6 iterations.
## Gradient range [-1.139795e-08,8.769648e-09]
## (score 286.8822 & scale 1).
## Hessian positive definite, eigenvalue range [0.7891478,23.77281].
## Model rank =  10 / 10
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##          k'  edf k-index p-value
## s(x,y) 9.00 6.77    0.46  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Here we see that the `edf` and `k'` values are very close, so if we had set `k=10` to begin with, we'd try doubling it here to see what effect that had on the model.

Plotting the smooth side-by-side with the more flexible model shows the differences in the resulting surfaces. In particular note how the northern area of the surface is very flat when `k` is small:

```
par(mfrow=c(1,2))
vis.gam(dsm_nb_xy, view=c("x","y"), plot.type="contour", too.far=0.1, main="s(x,y) (link scale)", asp=1
vis.gam(dsm_nb_xy_smallk, view=c("x","y"), plot.type="contour", too.far=0.1, main="s(x,y) (small k, lin
```



Note we use `zlim` to ensure that the plots are on the same colourscale.

Estimated abundance as response

So far we've just used `count` as the response. That is, we adjusted the offset of the model to make it take into account the "effective area" of the segments (see lecture notes for a refresher).

Instead of using `count` we could use `abundance.est`, which will leave the segment areas as they are and calculate the Horvitz-Thompson estimates of the abundance per segment and use that as the response in the model. This is most useful when we have covariates in the detection function (though we can use it any time).

Try copying the code that fits the model `dsm_nb_xy` and make a model `dsm_nb_xy_ae` that replaces `count` for `abundance.est` in the model formula and uses the `df_hr_ss_size` detection function. Compare the results of summaries, plots and checks between this and the count model.

Suggested, changing the response to `abundance.est` and the detection function to `df_hr_ss_size` we can fit:

```
dsm_nb_xy_ae <- dsm(abundance.est~s(x,y),
                    ddf.obj=df_hr_ss_size, segment.data = segs, observation.data=obs,
                    family=nb(), method="REML")
```

looking at a summary:

```
summary(dsm_nb_xy_ae)
```

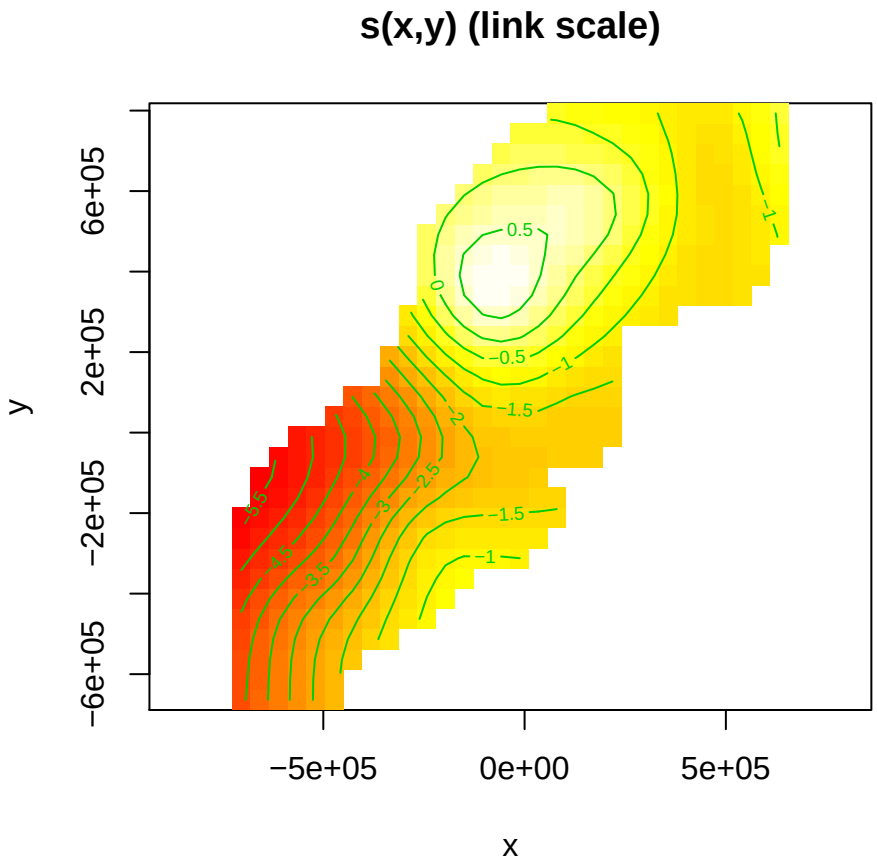
```
##
## Family: Negative Binomial(0.038)
## Link function: log
##
## Formula:
## abundance.est ~ s(x, y) + offset(off.set)
##
## Parametric coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -19.983      0.199  -100.4   <2e-16 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##          edf Ref.df Chi.sq  p-value
## s(x,y) 10.26   13.9  52.22 2.42e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.0649   Deviance explained = 26.1%
## -REML = 370.41  Scale est. = 1         n = 949
```

Note here the difference in the parameter of the negative binomial distribution, 0.038 vs. 0.089, due to the different response.

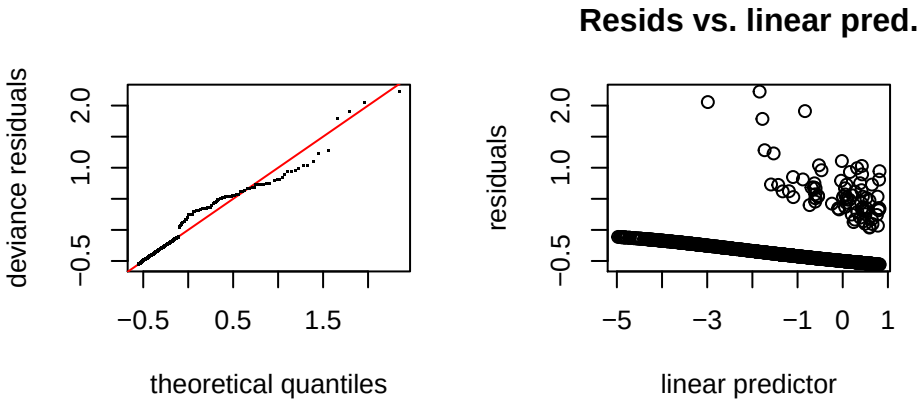
The EDF of the spatial smooth, however is rather similar. Looking at that spatial smooth:

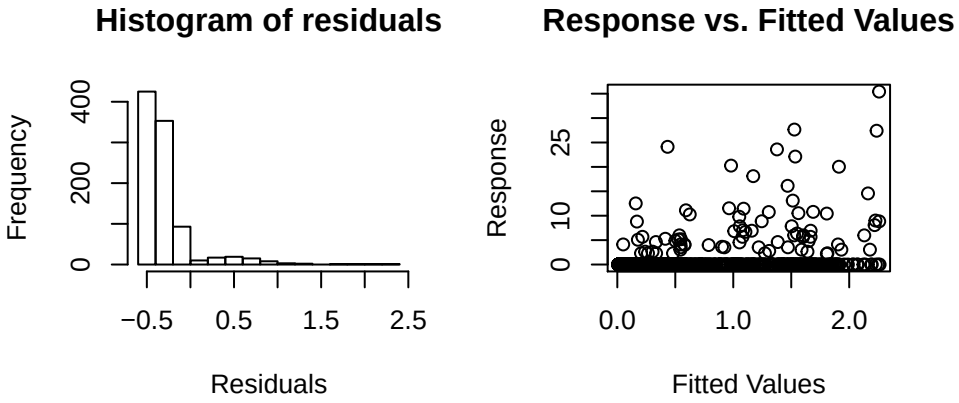
```
vis.gam(dsm_nb_xy_ae, view=c("x","y"), plot.type="contour", too.far=0.1, main="s(x,y) (link scale)", asp
```



Looking at check results:

```
gam.check(dsm_nb_xy_ae)
```






```
##
## Method: REML   Optimizer: outer newton
## full convergence after 7 iterations.
## Gradient range [-1.775404e-07,-2.314243e-08]
## (score 370.4095 & scale 1).
## Hessian positive definite, eigenvalue range [0.8100528,30.32358].
## Model rank =  30 / 30
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##          k'  edf k-index p-value
## s(x,y) 29.0 10.3     0.4  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Here we see a much bigger deviation from the $x = y$ line. We'll visit this again below when we look at assessing the response distribution.

Univariate models

Instead of fitting a bivariate smooth of `x` and `y` using `s(x, y)`, we could instead use the additive nature and fit the following model:

```
dsm_nb_x_y <- dsm(count~s(x) + s(y),
                  ddf.obj=df_hn, segment.data = segs, observation.data=obs,
                  family=nb())
```

Compare this model with `dsm_nb_xy` using `vis.gam()` (Note you can display two plots side-by-side using `par(mfrow=c(1,2))`). Investigate the output from `summary()` and the check functions too, comparing with the other models, adjust `k` if necessary.

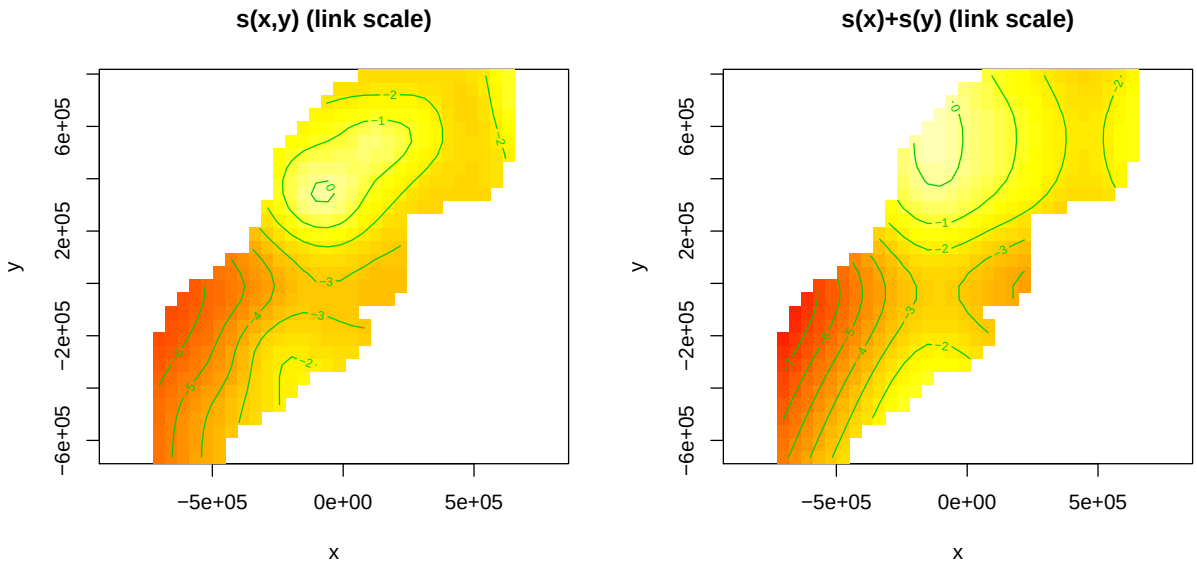
First we should check if the basis size needs to be increased for this model:

```
# again surpressing the figure output using fig.keep="none" in this chunk
gam.check(dsm_nb_x_y)
```

```
##
## Method: REML   Optimizer: outer newton
## full convergence after 5 iterations.
## Gradient range [-7.515378e-07,5.737452e-07]
## (score 286.0643 & scale 1).
## Hessian positive definite, eigenvalue range [0.5829725,23.22188].
## Model rank =  19 / 19
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##        k'  edf k-index p-value
## s(x) 9.00 4.26    0.64   0.200
## s(y) 9.00 4.11    0.62   0.015 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

(Everything looks good there.)

```
par(mfrow=c(1,2))
vis.gam(dsm_nb_xy, view=c("x","y"), plot.type="contour", too.far=0.1, main="s(x,y) (link scale)", asp=1
vis.gam(dsm_nb_x_y, view=c("x","y"), plot.type="contour", too.far=0.1, main="s(x)+s(y) (link scale)", as
```



The plot shows that due to the lack of interaction between the `x` and `y` terms, we don't get as detailed a plot. The contour lines are much simpler when the additive structure is used. Though we do note that the pattern is broadly-speaking similar in both cases.

Also note that the bivariate model has a higher percentage deviance explained than the additive model.

Tweedie response distribution

So far, we've used `nb()` as the response – the negative binomial distribution. We can also try out the Tweedie distribution as a response by replacing `nb()` with `tw()`.

Try this out and compare the resulting check plots.

First looking at the model with count as the response and changing the response to Tweedie by specifying `family=tw()`:

```
dsm_tw_xy <- dsm(count~s(x,y),
                 ddf.obj=df_hn, segment.data = segs, observation.data=obs,
                 family=tw())
```

Summarising that model:

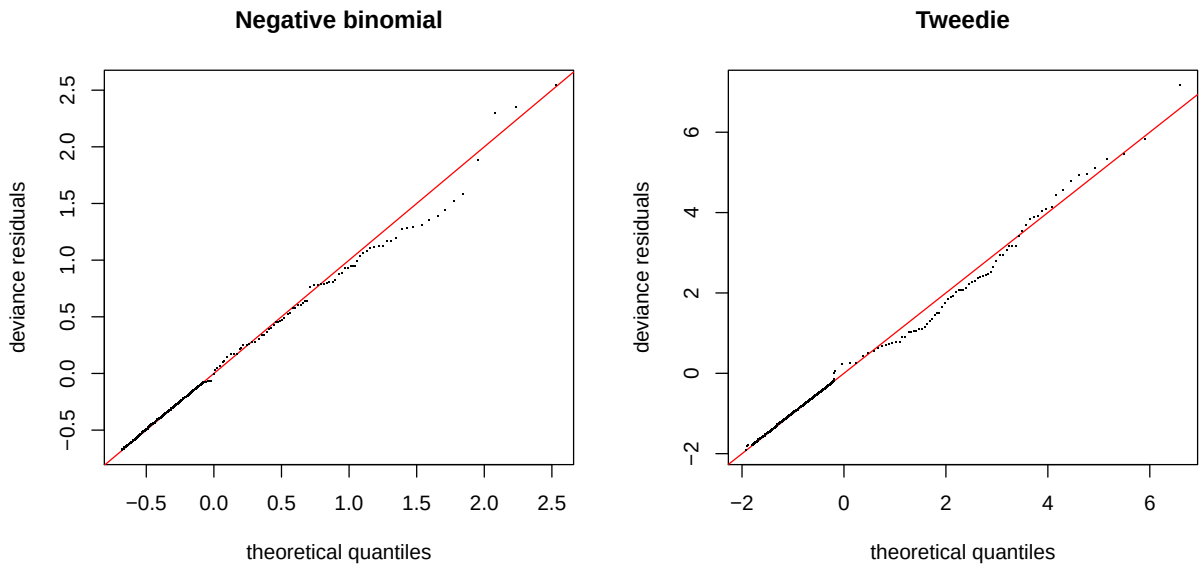
```
summary(dsm_tw_xy)
```

```
##
## Family: Tweedie(p=1.295)
## Link function: log
##
## Formula:
## count ~ s(x, y) + offset(off.set)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -20.389      0.219  -93.09   <2e-16 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##          edf Ref.df     F  p-value
## s(x,y) 13.64   17.8 4.607 1.01e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.0958   Deviance explained =   32%
## -REML = 287.04  Scale est. = 5.0519    n = 949
```

For brevity here, we can just compare the quantile-quantile plots using the `qq.gam()` function:

```
par(mfrow=c(1,2))
qq.gam(dsm_nb_xy, main="Negative binomial")
qq.gam(dsm_tw_xy, main="Tweedie")
```



The result here is far from clear-cut but (to my eye) it seems like the Tweedie model is better (in Q-Q plot terms at least).

We can also repeat our model with additive terms but using a Tweedie response, for the sake of completeness:

```
dsm_tw_x_y <- dsm(count~s(x) + s(y),
                  ddf.obj=df_hn, segment.data = segs, observation.data=obs,
                  family=tw(), method="REML")
summary(dsm_tw_x_y)
```

```
##
## Family: Tweedie(p=1.309)
## Link function: log
##
## Formula:
## count ~ s(x) + s(y) + offset(off.set)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
```

```
## (Intercept)  -20.253      0.212  -95.56   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##        edf Ref.df     F  p-value
## s(x) 4.032  5.060 3.110 0.007475 **
## s(y) 4.424  5.482 4.469 0.000652 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.061   Deviance explained = 25.7%
## -REML = 289.06  Scale est. = 5.4766    n = 949
```

Moving on to the estimated abundance response, we saw a pretty badly behaved Q-Q plot when we looked at the estimated abundance model with negative binomial response above. We can instead try using the Tweedie distribution, by specifying `family=tw()`:

```
dsm_tw_xy_ae <- dsm(abundance.est~s(x,y),
                    ddf.obj=df_hr_ss_size, segment.data = segs, observation.data=obs,
                    family=tw())
```

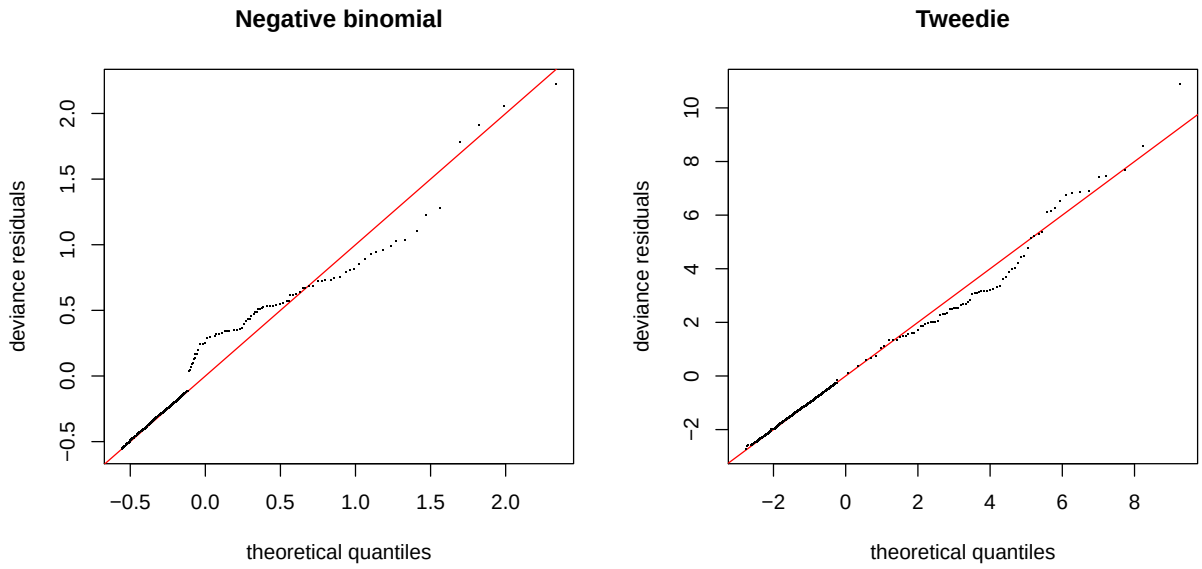
Summarising that model:

```
summary(dsm_tw_xy_ae)
```

```
##
## Family: Tweedie(p=1.247)
## Link function: log
##
## Formula:
## abundance.est ~ s(x, y) + offset(off.set)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -19.9225     0.2117  -94.12   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##          edf Ref.df     F  p-value
## s(x,y) 14.03  18.22 4.538 1.04e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.103   Deviance explained = 31.4%
## -REML = 339.23  Scale est. = 9.4515    n = 949
```

Comparing Q-Q plots:

```
par(mfrow=c(1,2))
qq.gam(dsm_nb_xy_ae, main="Negative binomial")
qq.gam(dsm_tw_xy_ae, main="Tweedie")
```



This plot is much easier to read, clearly the Tweedie response is much better!

Save models

It'll be interesting to see how these models compare to the more complex models we'll see later on. Let's save the fitted models at this stage.

```
# add your models here
save(dsm_nb_x_y, dsm_nb_xy, dsm_tw_x_y, dsm_tw_xy,
     file="dsms-xy.RData")
```

Extra credit

If you have time, try the following:

- What happens when we set `family=quasipoisson()`? Compare results of `gam.check` for this and the other models.

Fitting a quasi-Poisson model and looking at the summary:

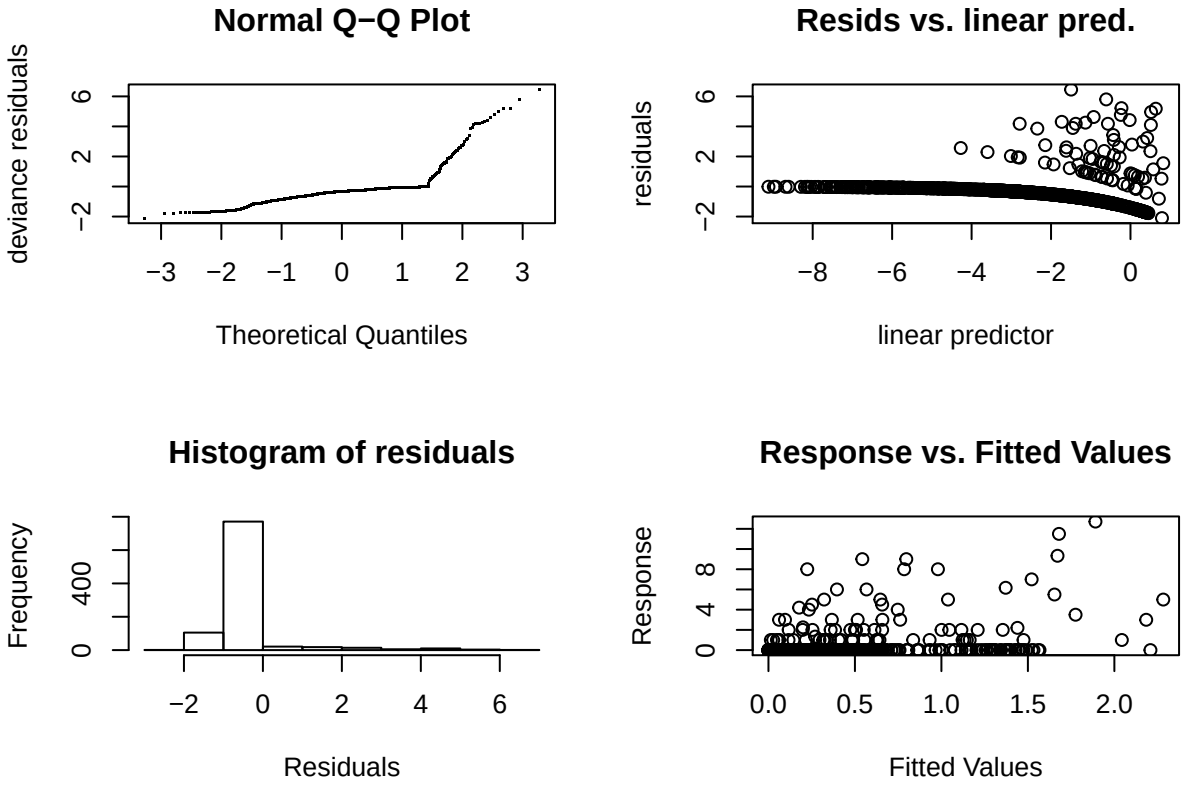
```
dsm_qp_xy <- dsm(count~s(x,y),
                 ddf.obj=df_hn, segment.data = segs, observation.data=obs,
                 family=quasipoisson())
summary(dsm_qp_xy)
```

```
##
## Family: quasipoisson
## Link function: log
##
## Formula:
## count ~ s(x, y) + offset(off.set)
##
## Parametric coefficients:
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -20.7681     0.4402  -47.18   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##         edf Ref.df     F  p-value
## s(x,y) 20.6  23.86 3.413 8.92e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.114   Deviance explained = 33.6%
## -REML = 358.93  Scale est. = 3.2107    n = 949
```

It looks like a significantly better R^2... what about checks...

```
gam.check(dsm_qp_xy)
```

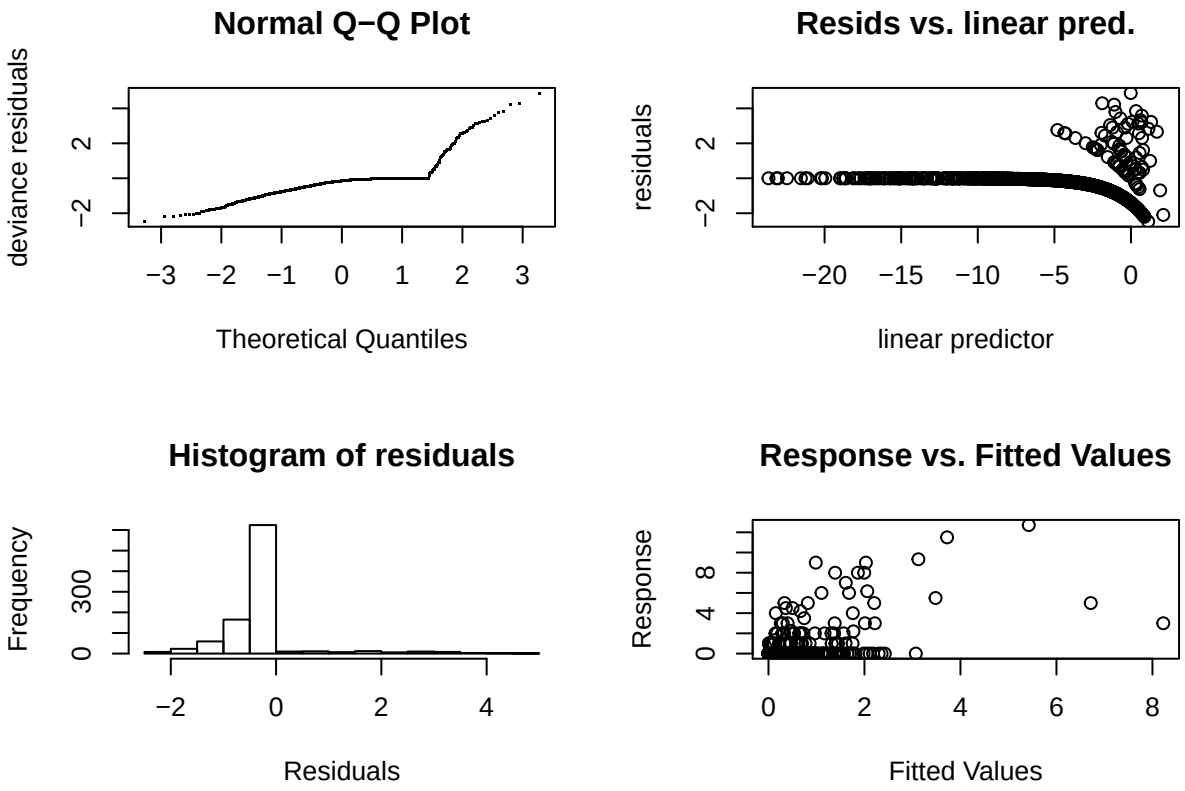



```
##
## Method: REML   Optimizer: outer newton
## full convergence after 7 iterations.
## Gradient range [-3.05987e-07,3.945393e-07]
## (score 358.9255 & scale 3.210668).
## Hessian positive definite, eigenvalue range [3.409044,337.6478].
## Model rank =  30 / 30
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
```

```
## indicate that k is too low, especially if edf is close to k'.
##
##          k'  edf k-index p-value
## s(x,y) 29.0 20.6     0.7  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Looks like we need to increase `k`, but also check out the Q-Q plot – not very nice!

```
dsm_qp_xy_k <- dsm(count~s(x, y, k=60),
                   ddf.obj=df_hn, segment.data = segs, observation.data=obs,
                   family=quasipoisson())
gam.check(dsm_qp_xy_k)
```

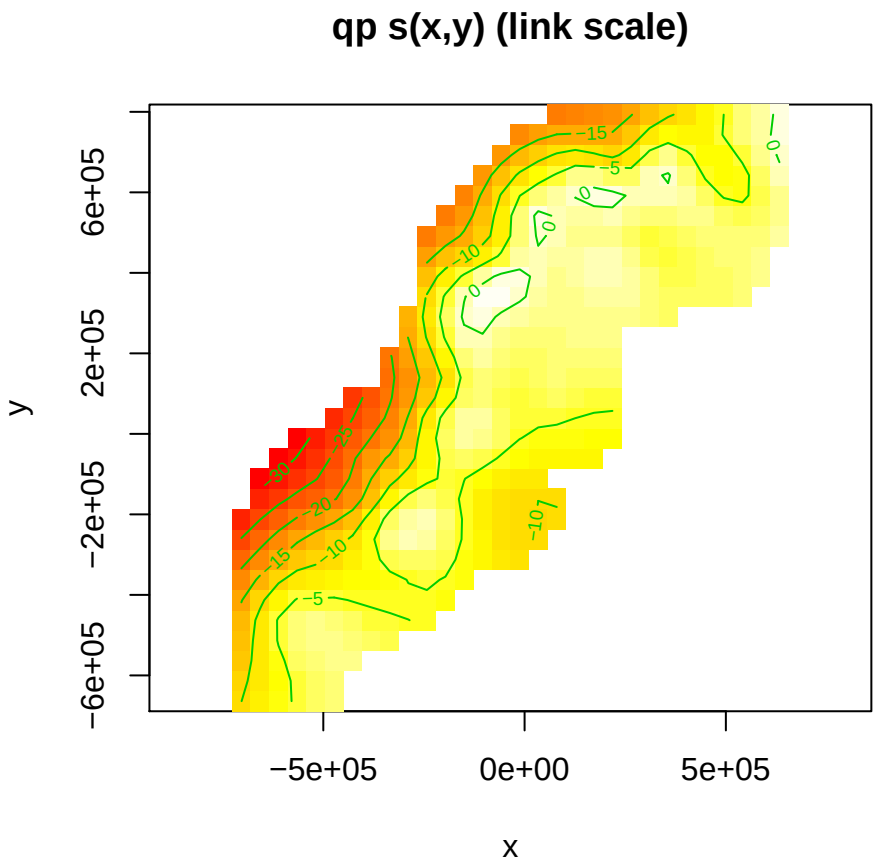



```
##
## Method: REML   Optimizer: outer newton
## full convergence after 6 iterations.
## Gradient range [-5.363825e-10,2.824088e-10]
## (score 305.9393 & scale 1.962812).
## Hessian positive definite, eigenvalue range [6.766839,338.2404].
## Model rank =  60 / 60
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##          k'  edf k-index p-value
## s(x,y) 59.0 41.6    0.82  <2e-16 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Looks like the complexity is high enough but the Q-Q plot is still not very nice.

```
vis.gam(dsm_qp_xy_k, view=c("x","y"), plot.type="contour", too.far=0.1, main="qp s(x,y) (link scale)", a
```



Broadly speaking the spatial pettern is rather similar, but seems much more intricate. Given the poor Q-Q plots, we're inclined to discard this model (though we'll revisit it in the prediction exercise).

- Make the `k` value very big (~100 or so), what do you notice?

Let's try setting `k=100` for the Tweedie model:

```
dsm_tw_xy_bigk <- dsm(count~s(x, y, k=100),
                      ddf.obj=df_hn, segment.data = segs, observation.data=obs,
                      family=tw())
summary(dsm_tw_xy_bigk)
```

```
##
## Family: Tweedie(p=1.288)
## Link function: log
##
## Formula:
## count ~ s(x, y, k = 100) + offset(off.set)
##
```

```
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -20.4666     0.2268  -90.25   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##          edf Ref.df     F  p-value
## s(x,y) 18.02  25.27 3.425 3.43e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.122   Deviance explained = 35.7%
## -REML = 286.66  Scale est. = 4.8499    n = 949
```

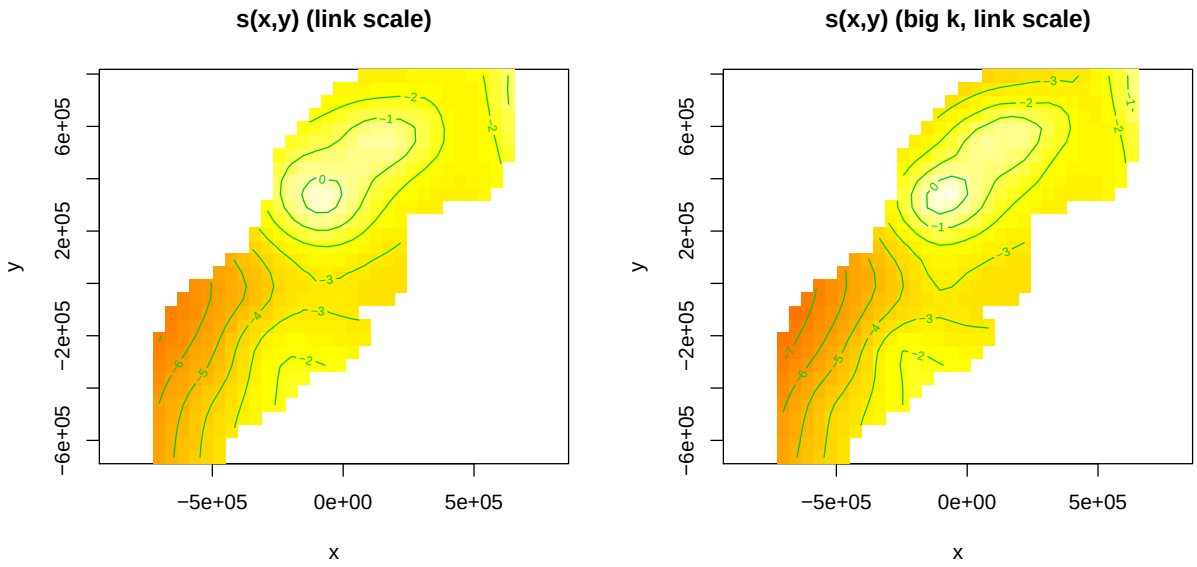
The first thing you should notice is that this model took longer to fit. Looking at the `summary()` and `gam.check()` results we see that the EDF has jumped up to ~18 much less than the 100 we gave it, though still much greater than the EDF of, say, `dsm_tw_xy`:

```
# here we suppress the plot using fig.keep="none" as a chunk option
gam.check(dsm_tw_xy_bigk)
```

```
##
## Method: REML   Optimizer: outer newton
## full convergence after 7 iterations.
## Gradient range [-8.737531e-07,1.148254e-06]
## (score 286.6564 & scale 4.849869).
## Hessian positive definite, eigenvalue range [1.467863,213.7068].
## Model rank =  100 / 100
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##        k' edf k-index p-value
## s(x,y) 99  18    0.61  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Now comparing the spatial effects, we see that there is not too much of a difference in the spatial pattern. The biggest difference is the much faster drop-off once we get nearer the coast (which may be better modelled by, say a depth covariate in the next exercise?)

```
par(mfrow=c(1,2))
vis.gam(dsm_tw_xy, view=c("x","y"), plot.type="contour", too.far=0.1, main="s(x,y) (link scale)", asp=1
vis.gam(dsm_tw_xy_bigk, view=c("x","y"), plot.type="contour", too.far=0.1, main="s(x,y) (big k, link sc
```


s(x,y) (link scale)
s(x,y) (big k, link scale)
y
x
−6e+05
−2e+05
2e+05
6e+05
−5e+05
0e+00
5e+05