# Lab 5.2: Spatio-temporal inference using the IDE model

In this Lab we use the package **IDE** to fit spatio-temporal IDE models as well as predict and forecast from spatio-temporal data. We explore three cases. The first two cases consider simulated data where the true model is known, and the third considers a radar data set.

For this Lab, we need the package **IDE** and also the package **FRK**, which will be used to construct basis functions to model the spatially varying parameters of the kernel. In addition, we shall use the package **plyr** for binding data frames with unequal column number later on in the Lab.

```
library("plyr")
library("dplyr")
library("IDE")
library("FRK")
library("ggplot2")
library("sp")
library("spacetime")
library("STRbook")
```

The first-order spatio-temporal IDE process model used in the package **IDE** is given by

$$Y_t(\mathbf{s}) = \int_{D_s} m(\mathbf{s}, \mathbf{x}; \boldsymbol{\theta}_p) Y_{t-1}(\mathbf{x})\, \mathrm{d}\mathbf{x} + \eta_t(\mathbf{s}), \quad \mathbf{s} \in D_s, \tag{1}$$

for $t = 1, 2, \ldots,$ where $m(\mathbf{s}, \mathbf{x}; \boldsymbol{\theta}_p)$ is a *transition kernel*, depending on parameters $\boldsymbol{\theta}_p$ that specify "redistribution weights" for the process at the previous time over the spatial domain, $D_s$, and $\eta_t(\mathbf{s})$ is a time-varying (but statistically independent in time) continuous mean-zero Gaussian spatial process. Note that we assume here, as one often does, that the parameter vector $\boldsymbol{\theta}_p$ does not vary with time, but it could do so in general. So, the process at location $\mathbf{s}$ and time $t$ is given by the weighted average (integral) of the process throughout the domain at the past time, where the weights are given by the transition kernel, $m(\cdot)$. The "adjustment," given by $\eta_t(\mathbf{s})$, is assumed to be Gaussian and accounts for spatial dependencies in $Y_t(\cdot)$ that are not captured by this weighted average. Another way to think about $\eta_t(\cdot)$ is that it adds smaller-scale dependence that is removed in the inherent smoothing that occurs when $\{Y_{t-1}(\cdot)\}$ is averaged over space, in order to give $Y_t(\cdot)$ a realistic spatial dependence structure. In general, $\int_{D_s} m(\mathbf{s}, \mathbf{x}; \boldsymbol{\theta}_p)\mathrm{d}\mathbf{x} < 1$ for the process to be stable (non-explosive) in time.

The kernel $m(\mathbf{s}, \mathbf{x}; \boldsymbol{\theta}_p)$ used by the package **IDE** is given by

$$m(\mathbf{s}, \mathbf{x}; \boldsymbol{\theta}_p) = \theta_{p,1}(\mathbf{s}) \exp\left(-\frac{1}{\theta_{p,2}(\mathbf{s})}\left[(x_1 - \theta_{p,3}(\mathbf{s}) - s_1)^2 + (x_2 - \theta_{p,4}(\mathbf{s}) - s_2)^2\right]\right), \tag{2}$$

where the spatially varying kernel amplitude is given by $\theta_{p,1}(\mathbf{s})$ and controls the temporal stationarity, the spatially varying length-scale (variance) parameter $\theta_{p,2}(\mathbf{s})$ corresponds to a kernel scale (aperture or width) parameter (i.e., the kernel width increases as $\theta_{p,2}$ increases), and the mean (shift) parameters $\theta_{p,3}(\mathbf{s})$ and $\theta_{p,4}(\mathbf{s})$ correspond to a spatially varying shift of the kernel relative to location $\mathbf{s}$. Spatially invariant kernels (i.e., where the elements of $\boldsymbol{\theta}_p$ are not functions of space) are also allowed.

The package **IDE** uses a bisquare spatial-basis-function decomposition for both the process $Y_t(\cdot)$ and the spatial process $\eta_t(\cdot)$, $t = 1, 2, \ldots$. The covariance matrix of the basis-function coefficients associated with $\eta_t(\cdot)$ is assumed to be proportional to the identity matrix, where the constant of proportionality is estimated. In **IDE**, the latent process $\widetilde{Y}_t(\mathbf{s})$ is the IDE dynamical process superimposed on some fixed effects, which can be expressed as a linear combination of known covariates $\mathbf{x}_t(\mathbf{s})$,

$$\widetilde{Y}_t(\mathbf{s}) = \mathbf{x}_t(\mathbf{s})'\boldsymbol{\beta} + Y_t(\mathbf{s}); \quad \mathbf{s} \in D_s, \tag{3}$$

for $t = 1, 2, \ldots$, where $\boldsymbol{\beta}$ are regression coefficients. The data vector $\mathbf{Z}_t \equiv (Z_t(\mathbf{r}_{1t}), \ldots, Z_t(\mathbf{r}_{m_t t}))'$ is then the latent process observed with noise,

$$Z_t(\mathbf{r}_{jt}) = \widetilde{Y}_t(\mathbf{r}_{jt}) + \epsilon_t(\mathbf{r}_{jt}), \quad j = 1, \ldots, m_t,$$

for $t = 1, 2, \ldots$, where $\epsilon_t(\mathbf{r}_{jt}) \sim iid\, Gau(0, \sigma^2_\epsilon)$.

## Simulation Example with a Spatially Invariant Kernel

The package **IDE** contains a function `simIDE` that simulates the behavior of a typical dynamic system governed by linear transport. The function can simulate from a user-defined IDE model, or from a pre-defined one. In the latter case, the number of time points to simulate (`T`), the number of (spatially fixed) observations to use (`nobs`), and a flag indicating whether to use a spatially invariant kernel (`k_spat_invariant = 1`) or not (`k_spat_invariant = 0`), need to be provided. The pre-defined model includes a linear trend in $s_1$ and $s_2$.

```
SIM1 <- simIDE(T = 10, nobs = 100, k_spat_invariant = 1)
```

The returned list `SIM1` contains the simulated process in the data frame `s_df`, the observed data in the data frame `z_df`, and the observed data as an `STIDF` object `z_STIDF`. It also contains two **ggplot2** plots, `g_truth` and `g_obs`, which can be readily plotted as follows.

```
print(SIM1$g_truth)
print(SIM1$g_obs)
```

While the action of the transport is clearly noticeable in the evolution of the process, there is also a clear spatial trend. Covariates are included through the use of a standard `R` formula when calling the function `IDE`. Additional arguments to `IDE` include the data set,

which needs to be of class `STIDF`, the temporal discretization to use (we will use 1 day) of class `difftime`, and the resolution of the grid upon which the integrations (as well as predictions) will be carried out. Other arguments include user-specified basis functions for the process and what transition kernel will be used, which for now we do not specify. By default, the IDE model will decompose the process using two resolutions of bisquare basis functions and will assume a spatially invariant Gaussian transition kernel.

```
IDEmodel <- IDE(f = z ~ s1 + s2,
                data = SIM1$z_STIDF,
                dt = as.difftime(1, units = "days"),
                grid_size = 41)
```

The returned object `IDEmodel` is of class `IDE` and contains initial parameter estimates, as well as predictions of $\boldsymbol{\alpha}_t$, for $t = 1, \ldots, T$, at these initial parameter estimates. The parameters in this case are the measurement-error variance, the variance of the random disturbance $\eta_t(\cdot)$ (whose covariance structure is fixed), the kernel parameters, and the regression coefficients $\boldsymbol{\beta}$.

Estimating the parameters in the IDE model using maximum likelihood is a computationally intensive procedure. The default method currently implemented uses a differential evolution optimization algorithm from the package **DEoptim**, which is a global optimization algorithm that can be easily parallelized. Fitting takes only a few minutes on a 60-core high-performance computer, but can take an hour or two on a standard desktop computer. Fitting can be done by running the following code.

```
fit_results_sim1 <- fit.IDE(IDEmodel,
                            parallelType = 1)
```

Here `parallelType = 1` ensures that all available cores on the computer are used for fitting. Alternatively, the results can be loaded from cache using the following command.

```
data("IDE_Sim1_results", package = "STRbook")
```

The list `fit_results_sim1` contains two fields: `optim_results` that contains the output of the optimization algorithm, and `IDEmodel` that contains the fitted IDE model. The fitted kernel can be visualized by using the function **`show_kernel`**.

```
show_kernel(fit_results_sim1$IDEmodel)
```

Note how the fitted kernel is shifted to the left and upwards, correctly representing the southeasterly transport evident in the data. The estimated kernel parameters $\boldsymbol{\theta}_p$ are given below.

```
fit_results_sim1$IDEmodel$get("k") %>% unlist()

##      par1      par2      par3      par4
## 152.83635   0.00198  -0.10160   0.10037
```

These estimates compare well to the true values `c(150, 0.002, -0.1, 0.1)` (see the help file for **simIDE**). The estimated regression coefficients are given below.

```
coef(fit_results_sim1$IDEmodel)

## Intercept         s1         s2
##     0.207      0.197      0.191
```

These also compare well to the true values `c(0.2, 0.2, 0.2)`. Also of interest are the moduli of the possibly complex eigenvalues of the evolution matrix $\mathbf{M}$. These can be extracted as follows.

```
abs_ev <- eigen(fit_results_sim1$IDEmodel$get("M"))$values %>%
          abs()
summary(abs_ev)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.003   0.290   0.366   0.331   0.399   0.464
```

Since the largest of these is less than 1, the IDE exhibits stable behavior.

For prediction, one may either specify a prediction grid or use the default one used for approximating the integrations set up by **IDE**. The latter is usually sufficient, so we use this without exception for the examples we consider. When a prediction grid is not supplied, the function **predict** returns a data frame with predictions spanning the temporal extent of the data (forecasts and hindcasts are explored later).

```
ST_grid_df <- predict(fit_results_sim1$IDEmodel)
```

The prediction map and prediction-standard-error map can now be plotted using standard **ggplot2** commands as follows.

```
gpred <- ggplot(ST_grid_df) +        # Plot the predictions
  geom_tile(aes(s1, s2, fill=Ypred)) +
  facet_wrap(~t) +
  fill_scale(name = "Ypred", limits = c(-0.1, 1.4)) +
  coord_fixed(xlim=c(0, 1), ylim = c(0, 1))

gpredse <- ggplot(ST_grid_df) +     # Plot the prediction s.es
  geom_tile(aes(s1, s2, fill = Ypredse)) +
```

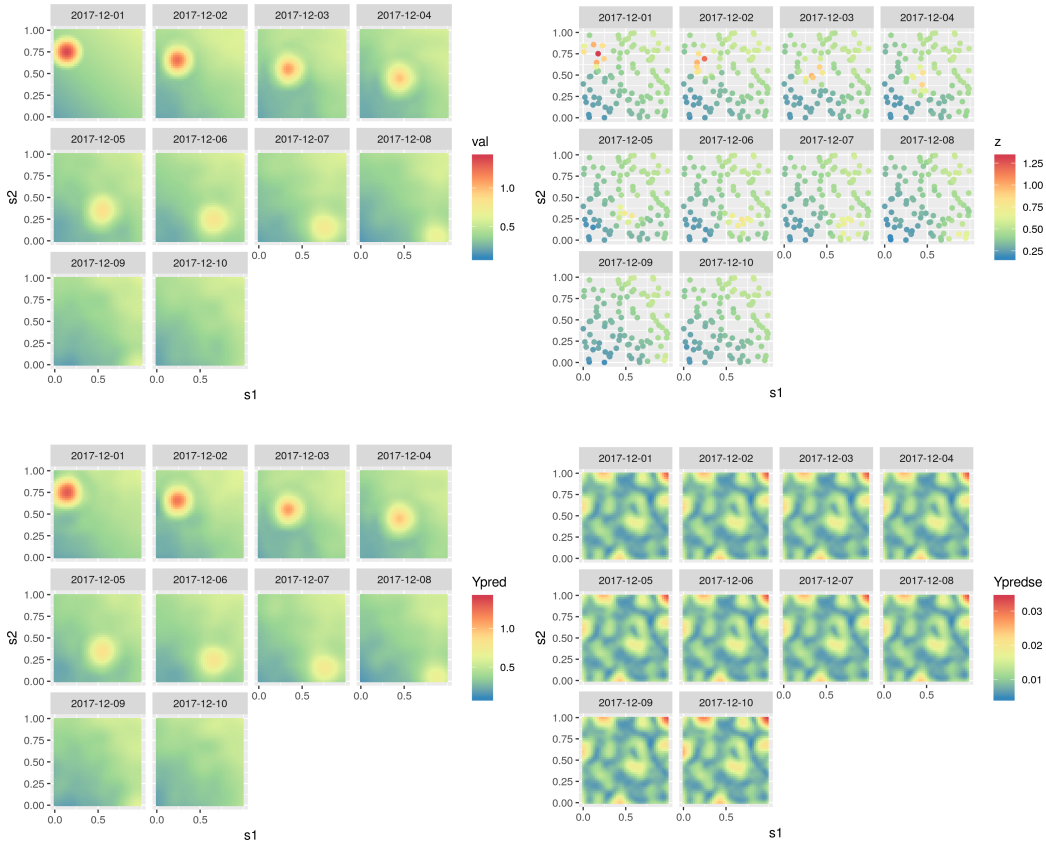


Figure 1: Simulated process (top left), simulated data (top right), predictions following the fitting of the IDE model (bottom left) and the respective prediction standard errors (bottom right).

```
  facet_wrap(~t) +
  fill_scale(name = "Ypredse") +
  coord_fixed(xlim=c(0, 1), ylim = c(0, 1))
```

In Figure 1, we show the observations, the true process, the predictions, and the prediction standard errors from the fitted model. Notice that the prediction standard errors are large in regions of sparse observations, as expected.

### Simulation Example with a Spatially Varying Kernel

In the previous example we considered the case of a spatially invariant kernel, that is, the case when the kernel $m(\mathbf{s}, \mathbf{x}; \boldsymbol{\theta}_p)$ is just a function of $\mathbf{x} - \mathbf{s}$. In this example, we consider the case when one or more of the $\boldsymbol{\theta}_p$ are allowed to be spatially referenced. Such models are needed when the spatio-temporal process exhibits, for example, considerable spatially varying drift (i.e., advection). Such a process can be simulated using the function **simIDE** by specifying `k_spat_invariant = 0`. To model data from a process of this sort, we need to have a large `nobs` and many time points; we set `T = 15`. This is important, as it is difficult to obtain reasonable estimates of spatially distributed parameters unless the data cover a large part of the spatial domain for a sustained amount of time.

```
SIM2 <- simIDE(T = 15, nobs = 1000, k_spat_invariant = 0)
```

As above, the process and the observed data can be plotted as two **ggplot2** plots.

```
print(SIM2$g_truth)
print(SIM2$g_obs)
```

Note how the process appears to rotate quickly counter-clockwise and come to a nearly complete standstill towards the lower part of the domain. The spatially varying advection that generated this field can be visualized using the following command.

```
show_kernel(SIM2$IDEmodel, scale = 0.2)
```

In this command, the argument `scale` is used to scale the arrow sizes by 0.2; that is, the shift per time point is five times the displacement indicated by the arrow.

Spatially varying kernels can be introduced by specifying the argument `kernel_basis` inside the call to **IDE**. The basis functions that **IDE** uses are of the same class as those used by **FRK**. We construct nine bisquare basis functions below that are equally spaced in the domain.

```
mbasis_1 <- auto_basis(manifold = plane(),    # fns on the plane
                       data = SIM2$z_STIDF,   # data
                       nres = 1,              # 1 resolution
                       type = 'bisquare')     # type of functions
```

To plot these basis functions, type `show_basis(mbasis_1)`.

Now, recall that $\theta_{p,1}$ (identified as `thetam1` in **IDE**) corresponds to the amplitude of the kernel, $\theta_{p,2}$ (`thetam2`) to the scale (width) or aperture, $\theta_{p,3}$ (`thetam3`) to the horizontal drift, and $\theta_{p,4}$ (`thetam4`) to the vertical drift. In what follows, suppose that $\theta_{p,1}$ and $\theta_{p,2}$ are spatially invariant (usually a reasonable assumption), and decompose $\theta_{p,3}$ and $\theta_{p,4}$ as sums of basis functions given in `mbasis_1`.

```
kernel_basis <- list(thetam1 = constant_basis(),
                     thetam2 = constant_basis(),
                     thetam3 = mbasis_1,
                     thetam4 = mbasis_1)
```

Modeling proceeds as before, except that now we specify the argument `kernel_basis` when calling **IDE**.

```
IDEmodel <- IDE(f = z ~ s1 + s2 + 1,
                data = SIM2$z_STIDF,
                dt = as.difftime(1, units = "days"),
                grid_size = 41,
                kernel_basis = kernel_basis)
```

Fitting also proceeds by calling the function **fit.IDE**. We use the argument `itermax = 400` below to specify the maximum number of iterations for the optimization routine to use.

```
fit_results_sim2 <- fit.IDE(IDEmodel,
                            parallelType = 1,
                            itermax = 400)
```

As above, since this is computationally intensive, we provide cached results that can be loaded using the following command.

```
data("IDE_Sim2_results", package = "STRbook")
```

The fitted spatially varying kernel can be visualized using the following command.

```
show_kernel(fit_results_sim2$IDEmodel)
```

The true and fitted spatially varying drift parameters are shown side by side in Figure 2. Note how the fitted drifts capture the broad directions and magnitudes of the true underlying process. Predictions and prediction standard errors can be obtained and mapped using **predict** as above. This is left as an exercise for the reader.

### The Sydney Radar Data Set

The radar data set contains data that are a subset of consecutive weather radar reflectivity images considered in the World Weather Research Programme (WWRP) Sydney 2000 Forecast Demonstration Project. There are 12 images at 10-minute intervals starting at 08:25 UTC on 3 November, 2000 (i.e., 08:25–10:15 UTC). The data were originally mapped to a $45 \times 45$ grid of 2.5-km pixels centered on the radar location. The data used here are for a

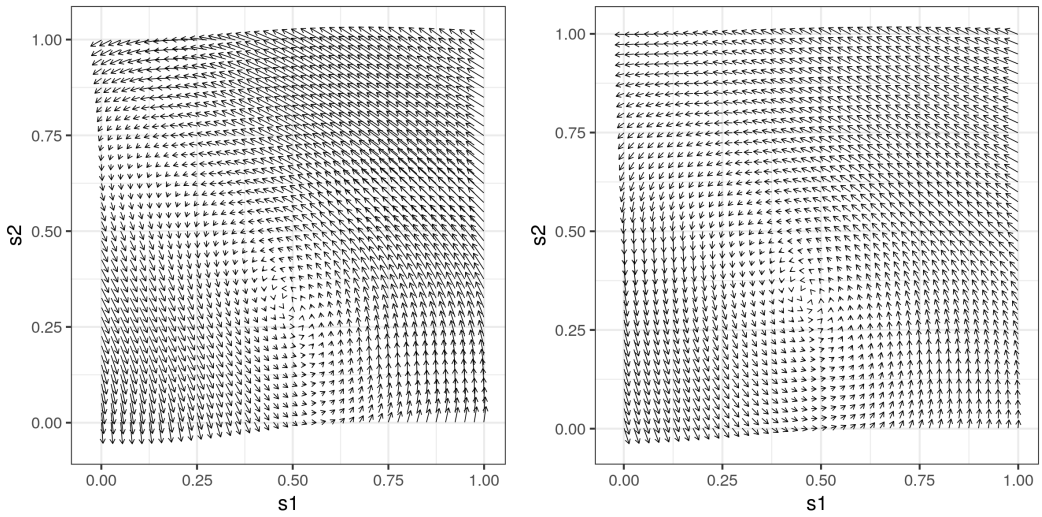


Figure 2: True drifts (left) and estimated drifts (right).

region of dimension $28 \times 40$, corresponding to a 70 km by 100 km domain. All reflectivities are given in "decibels relative to Z" (dBZ, a dimensional logarithmic unit used for weather radar relfectivities).

Analysis of the Sydney radar data set proceeds in much the same way as in the simulation examples. In this case, we choose to have a spatially invariant kernel, since the data are not suggestive of spatially varying dynamics. We first load the Sydney radar data set as an `STIDF` object.

```
data("radar_STIDF", package = "STRbook")
```

We now call the function **`IDE`** as before, with the added arguments `hindcast` and `forecast`, which indicate how many time intervals into the past, and how many into the future, we wish to predict for periods preceeding the training period (hindcast) and periods following the training period (forecast), respectively.In this case the data are at 10-minute intervals (one period), and we forecast and hindcast for two periods each (i.e., 20 minutes).

```
IDEmodel <- IDE(f = z ~ 1,
                data = radar_STIDF,
                dt = as.difftime(10, units = "mins"),
                grid_size = 41,
                forecast = 2,
                hindcast = 2)
```

Fitting proceeds by calling **`fit.IDE`**.

```
fit_results_radar <- fit.IDE(IDEmodel,
                             parallelType = 1)
```

Since this command will take a considerable amount of time on a standard machine, we load the results directly from cache.

```
data("IDE_Radar_results", package = "STRbook")
```

The fitted kernel can be visualized as it was above.

```
show_kernel(fit_results_radar$IDEmodel)
```

The kernel is again clearly shifted off-center and suggestive of transport in a predominantly easterly (and slightly northerly) direction. This is corroborated by visual inspection of the data. The estimated shift parameters are as follows.

```
shift_pars <- (fit_results_radar$IDEmodel$get("k") %>%
                   unlist())[3:4]
print(shift_pars)

## par3 par4
## -5.5 -1.9
```

The magnitude of the estimated shift vector is hence indicative of a transport of $\sqrt{(5.5)^2 + (1.9)^2} = 5.82$ km per 10-minute period, or 34.91 km per hour.

The modulus of the possibly complex eigenvalues of the evolution matrix $\mathbf{M}$ can be extracted as follows.

```
abs_ev <- eigen(fit_results_radar$IDEmodel$get("M"))$values %>%
          abs()
summary(abs_ev)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.01    0.58    0.67    0.62    0.72    0.79
```

The largest absolute eigenvalue is considerably larger than that in the simulation study, suggesting more field persistence (although, since it is less than 1, the process is still stable). This persistence is expected, since the data clearly show patches of precipitation that are sustained and transported, rather than decaying, over time.

When calling the function **IDE**, we set up the object to be able to forecast 20 minutes into the future and hindcast 20 minutes into the past. These forecasts and hindcasts will be in the object returned from **predict**.

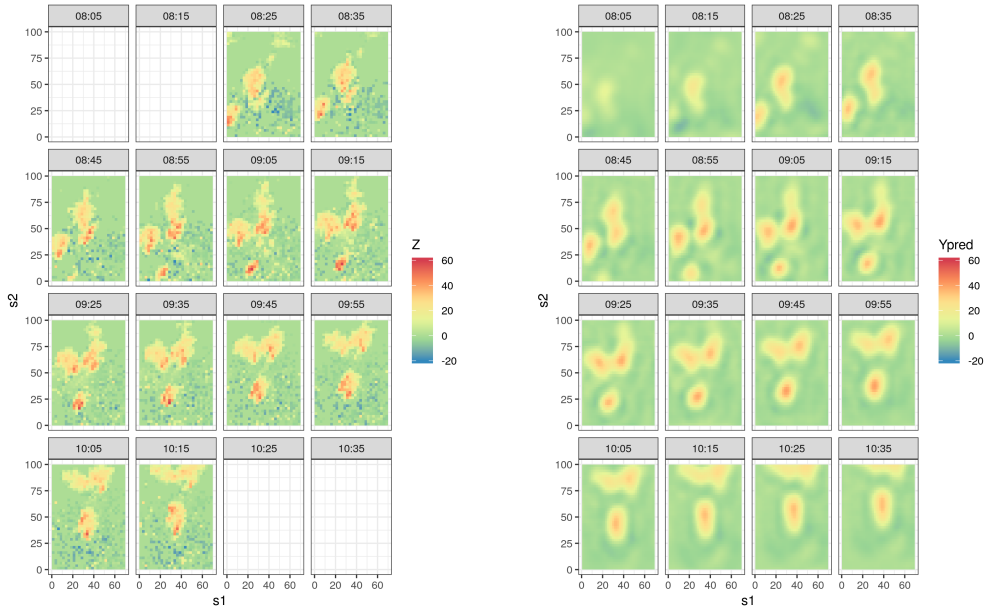


Figure 3: Observed data (left), and hindcasts, predictions, and forecasts using the IDE model (right).

```
ST_grid_df <- predict(fit_results_radar$IDEmodel)
```

The data frame `ST_grid_df` contains the predictions in the field `Ypred` and the prediction standard errors in the field `Ypredse`. The field `t`, in both our data and predictions, contains the date as well as the time; we now create another field `time` that contains just the time of day.

```
radar_df$time <- format(radar_df$t, "%H:%M")
ST_grid_df$time <- format(ST_grid_df$t, "%H:%M")
```

The code given below plots the data as well as the smoothed fields containing the hindcasts, the predictions, and the forecasts. So that we match the data plots with the prediction plots, timewise, we create empty fields corresponding to hindcast and forecast periods in the data frame containing the observations. This can be achieved easily using **`rbind.fill`** from the package **plyr**.

```
## Add time records with missing data
radar_df <- rbind.fill(radar_df,
                       data.frame(time = c("08:05", "08:15",
```

```
                                                "10:25", "10:35")))

## Plot of data, with color scale capped to (-20, 60)
gobs <- ggplot(radar_df) +
  geom_tile(aes(s1, s2, fill = pmin(pmax(z, -20), 60))) +
  fill_scale(limits = c(-20, 60), name = "Z") +
  facet_wrap(~time) + coord_fixed() + theme_bw()

## Plot of predictions with color scale forced to (-20, 60)
gpred <- ggplot(ST_grid_df) +
  geom_tile(aes(s1, s2, fill = Ypred)) +
  facet_wrap(~time) + coord_fixed() + theme_bw() +
  fill_scale(limits = c(-20, 60), name = "Ypred")
```

The plots are shown in Figure 3. Notice how both the forecasts and the hindcasts incorporate the information on transport that is evident in the data. We did not plot prediction standard errors in this case, which is left as an exercise for the reader.