Lab 6.1: Spatio-Temporal Model Validation

In this Lab we consider the validation of two spatio-temporal models that are fitted to the same data set. To show the importance of modeling dynamics, we shall consider the Sydney radar data set and compare predictions obtained using the IDE model to those obtained using the FRK model (which does not incorporate dynamics). We shall carry out validation on data that are held out. The hold-out data set will comprise (i) a block of data spanning two time points, and (ii) a 10% random sample of the data at the other time points. We expect the IDE model to perform particularly well when validating the block of data spanning two points, where information on the dynamics is pivotal for "filling in" the temporal gaps.

For this Lab we use the **IDE** and **FRK** packages for modeling,

```
library("FRK")
library("IDE")
```

the **scoringRules** and **verification** packages for probabilistic validation,

```
library("scoringRules")
library("verification")
```

and the usual packages for handling and plotting spatio-temporal data,

```
library("dplyr")
library("ggplot2")
library("sp")
library("spacetime")
library("STRbook")
library("tidyr")
```

Step 1: Training and Validation Data

First, we load the Sydney radar data set and create a new field `timeHM` that contains the time in an hours:minutes format.

```
data("radar_STIDF", package = "STRbook")
mtot <- length(radar_STIDF)
radar_STIDF$timeHM <- format(time(radar_STIDF), "%H:%M")
```

The initial stage of model verification is to hold out data prior to fitting the model, so that these data can be compared to the predictions once the model is fitted on the retained data. As explained above, we first leave out data at two time points, namely 09:35 and 09:45, by finding the indices of the observations that were made at these times, and then removing them from the complete set of observation indices.

```
valblock_idx <- which(radar_STIDF$timeHM %in% c("09:35",
                                                "09:45"))
obs_idx <- setdiff(1:mtot, valblock_idx)
```

We next leave out 10% of the data at the other time points by randomly sampling 10% of the elements from the remaining observation indices.

```
set.seed(1)
valrandom_idx <- sample(obs_idx,
                        0.1 * length(obs_idx),
                        replace = FALSE) %>% sort()
obs_idx <- setdiff(obs_idx, valrandom_idx)
```

We can now use the indices we have generated above to construct our training data set, a validation data set for the missing time points, and a validation data set corresponding to the data missing at random from the other time points.

```
radar_obs <- radar_STIDF[obs_idx, ]
radar_valblock <- radar_STIDF[valblock_idx, ]
radar_valrandom <- radar_STIDF[valrandom_idx, ]
```

Step 2: Fitting the IDE Model

In Lab 5.2 we fitted the IDE model to the entire data set. Here, instead, we fit the IDE model to the training data set created above. As before, since this computation takes a long time, we can load the results directly from cache.

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

fit_results_radar2 <- fit.IDE(IDEmodel,
                              parallelType = 1)
```

```
data("IDE_Radar_results2", package = "STRbook")
```

It is instructive to compare the estimated parameters from the full data set in Lab 5.2 to the estimated parameters from the training data set in this Lab. Reassuringly, we see that the intercept, the kernel parameters (which govern the system dynamics), as well as the variance parameters, have similar estimates.

```
## load results with full data set
data("IDE_Radar_results", package = "STRbook")
with(fit_results_radar$IDEmodel, c(get("betahat")[1,1],
                                   unlist(get("k")),
                                   get("sigma2_eps"),
                                   get("sigma2_eta")))

##          par1   par2   par3   par4   par5   par6
##  0.582  0.135  2.497 -5.488 -1.861 28.384  7.271

with(fit_results_radar2$IDEmodel, c(get("betahat")[1,1],
                                   unlist(get("k")),
                                   get("sigma2_eps"),
                                   get("sigma2_eta")))

##            par1    par2    par3    par4    par5    par6
##  0.5735  0.0909  3.6784 -5.2067 -1.8174 28.8660 10.1376
```

Prediction proceeds with the function **predict**. Since we wish to predict at specific locations we now use the argument newdata to indicate where and when the predictions need to be made. In this case we supply newdata with the STIDF objects we constructed above.

```
pred_IDE_block <- predict(fit_results_radar2$IDEmodel,
                          newdata = radar_valblock)
pred_IDE_random <- predict(fit_results_radar2$IDEmodel,
                          newdata = radar_valrandom)
```

Step 3: Fitting the FRK Model

For FRK we need to specify the spatial basis functions and temporal basis functions in order to construct the spatio-temporal basis functions. For the spatial basis functions we specify two resolutions of bisquare functions regularly distributed inside the domain.

```
G_spatial <- auto_basis(manifold = plane(),      # fns on plane
                        data = radar_obs,        # project
                        nres = 2,                # 2 res.
                        type = "bisquare",       # bisquare.
                        regular = 1)             # irregular
```

Type **show_basis**(G_spatial) to visualize the locations and apertures of these basis functions. For the temporal basis functions we regularly place five bisquare functions between 0 and 12 with an aperture of 3.5.

```
t_grid <- matrix(seq(0, 12, length = 5))
G_temporal <- local_basis(manifold = real_line(), # fns on R1
                          type = "bisquare",      # bisquare
                          loc = t_grid,           # centroids
                          scale = rep(3.5, 5))    # aperture par.
```

Type `show_basis(G_temporal)` to visualize these basis functions. Finally, we construct the spatio-temporal basis functions by taking their tensor product.

```
G <- TensorP(G_spatial, G_temporal)  # take the tensor product
```

Next we construct the BAUs. These are regularly placed space-time cubes covering our spatio-temporal domain. The `cellsize` we choose below is one that is similar to that which the **IDE** function constructed when specifying `grid_size = 41` above. We impose a convex hull as a boundary that is tight around the data points, and not extended.

```
BAUs <- auto_BAUs(manifold = STplane(),    # ST field on plane
                  type = "grid",           # gridded (not "hex")
                  data = radar_obs,        # data
                  cellsize = c(1.65, 2.38, 10), # BAU cell size
                  nonconvex_hull = FALSE, # convex boundary
                  convex = 0,              # no hull extension
                  tunit = "mins")          # time unit is "mins"
BAUs$fs = 1        # fs variation prop. to 1
```

As we did in Lab 4.2, we can take the measurement error to be that estimated elsewhere, in this case by the IDE model. Any remaining residual variation is then attributed to fine-scale variation that is modeled as white noise. Attribution of variation is less critical when validating against observational data, since the total variance is used when constructing prediction intervals.

```
sigma2_eps <- fit_results_radar2$IDEmodel$get("sigma2_eps")
radar_obs$std <- sqrt(sigma2_eps)
```

The function **FRK** is now called to fit the random-effects model using the chosen basis functions and BAUs.

```
S <- FRK(f = z ~ 1,
         BAUs = BAUs,
         data = list(radar_obs), # (list of) data
         basis = G,              # basis functions
         n_EM = 2,               # max. no. of EM iterations
         tol = 0.01)             # tol. on log-likelihood
```

Prediction proceeds using the **predict** function.

```
FRK_pred <- predict(S)
```

Since **predict** predicts over the BAUs, we need to associate each observation in our validation STIDFs to a BAU cell. This can be done simply using the function **over**. In the code below, the data frames df_block_over and df_random_over are data frames containing the predictions and prediction standard errors at the validation locations.

```
df_block_over <- over(radar_valblock, FRK_pred)
df_random_over <- over(radar_valrandom, FRK_pred)
```

Step 4: Organizing Predictions for Further Analysis

Having obtained our predictions and prediction standard errors from the two models, the next step is to combine them into one data frame. We take the hold-out STIDF from the two time points, convert it to a data frame, and then put in the FRK and IDE predictions, prediction standard errors on the process, and the prediction standard errors in observation space. We distinguish between the latter two by using the labels predse and predZse, respectively.

```
radar_valblock_df <- radar_valblock %>%
              data.frame() %>%
              mutate(FRK_pred = df_block_over$mu,
                     FRK_predse = df_block_over$sd,
                     FRK_predZse = sqrt(FRK_predse^2 +
                                        sigma2_eps),
                     IDE_pred = pred_IDE_block$Ypred,
                     IDE_predse = pred_IDE_block$Ypredse,
                     IDE_predZse = sqrt(IDE_predse^2 +
                                        sigma2_eps))
```

For plotting purposes, it is also convenient to construct a data frame in long format, where all the predictions are put into the same column, and a second column identifies to which model the prediction corresponds.

```
radar_valblock_df_long <- radar_valblock_df %>%
                          dplyr::select(s1, s2, timeHM, z,
                                 FRK_pred, IDE_pred) %>%
                          gather(type, num, FRK_pred, IDE_pred)
```

Construction of radar_valrandom_df and radar_valrandom_df_long proceeds in identical fashion to the code given above (with block replaced with random) and is thus omitted.

Step 5: Scoring

Now we have everything in place to start analyzing the prediction errors. We start by simply plotting histograms of the prediction errors to get an initial feel of the distributions of these errors from the two models. As before, we only show the code for the left-out data in `radar_valblock_df_long`.

```
ggplot(radar_valblock_df_long) +
    geom_histogram(aes(z - num, fill = type),
                   binwidth = 2, position = "identity",
                   alpha = 0.4, colour = 'black') + theme_bw()
```

The resulting distributions are relatively similar; however, a close look reveals that the errors from the FRK model have a slightly larger spread, especially for the data at the missing time points. This is a first indication that FRK, and the lack of consideration of dynamics, will be at a disadvantage when predicting the process across time points for which we have no data.

We next look at the correlation between the predictions and the observations, plotted below and shown in Figure 1. Again, there does not seem to be much of a difference in the distribution of the errors between the two models when the data are missing at random, but there is a noticeable difference when the data are missing for entire time points. In fact, the correlation between the predictions and observations for the FRK model is, in this case, 0.716, while that for the IDE model is 0.848.

```
ggplot(radar_valblock_df) + geom_point(aes(z, FRK_pred))
ggplot(radar_valblock_df) + geom_point(aes(z, IDE_pred))
```

It is interesting to see the effect of the absence of time points on the quality of the predictions. To this end, we can create a new data frame, which combines the validation data and the predictions, and compute the mean-squared prediction error (MSPE) for each time point.

```
MSPE_time <- rbind(radar_valrandom_df_long,
                   radar_valblock_df_long) %>%
         group_by(timeHM, type) %>%
         dplyr::summarise(MSPE = mean((z - num)^2))
```

The following code plots the evolution of the MSPE as a function of time.

```
ggplot(MSPE_time) +
  geom_line(aes(timeHM, MSPE, colour = type, group = type))
```

The evolution of the MSPE is shown in Figure 2, together with vertical dashed lines indicating the time points that were left out when fitting and predicting. It is remarkable to

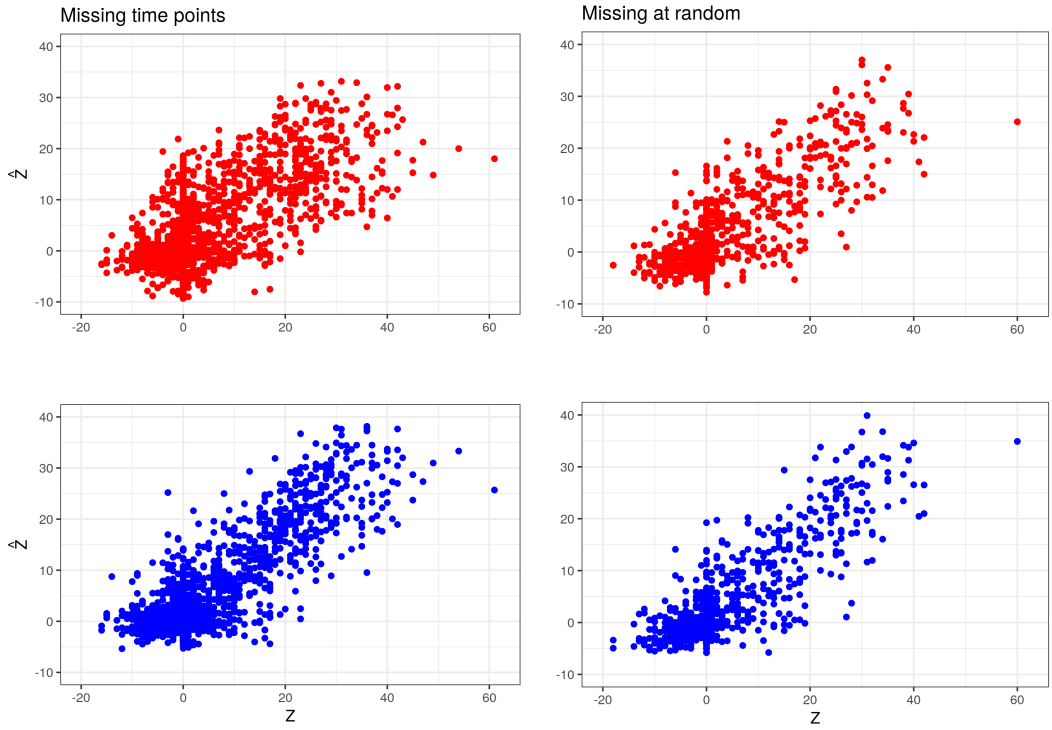

Figure 1: Scatter plots of the observations and predictions for the FRK model (red) and the IDE model (blue), when the data are missing for entire time points (left) and at random (right).

note how spatio-temporal FRK, due to its simple descriptive nature, suffers considerably, with an MSPE that is nearly twice that of the IDE model. Note that predictions close to this gap are also severely compromised. The IDE model is virtually unaffected by the missing data, as the trained dynamics are sufficiently informative to describe the evolution of the process at unobserved time points. At time points away from this gap, the MSPEs of the FRK and IDE models are comparable.

The importance of dynamics can be further highlighted by mapping the prediction standard errors at each time point. The plot in Figure 3, for which the commands are given below, reveals vastly constrasting spatial structures between the FRK prediction standard errors and the IDE prediction standard errors. Note that at other time points (we only show six adjoining time points) the prediction standard errors given by the two models are comparable.

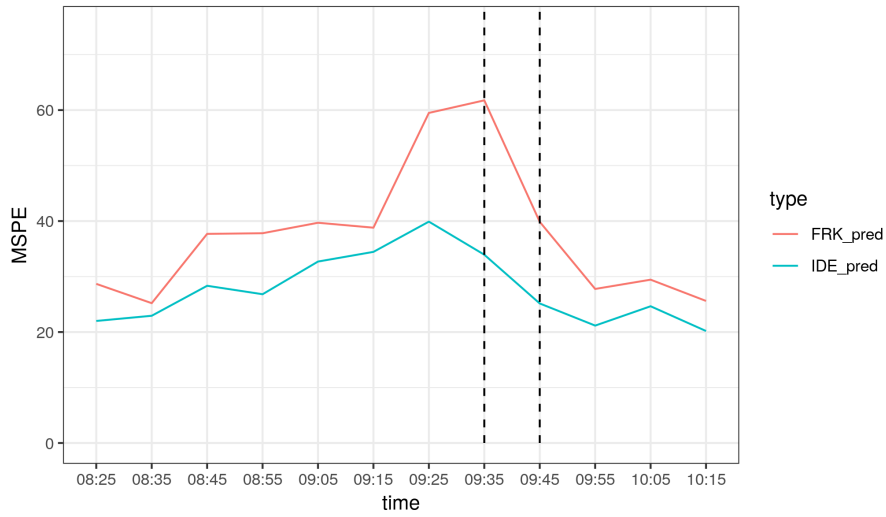

Figure 2: MSPE of the FRK predictions (red) and the IDE predictions (blue) as a function of time. The dotted black lines mark the times where no data were available for fitting or predicting.

```
ggplot(rbind(radar_valrandom_df_long, radar_valblock_df_long)) +
  geom_tile(aes(s1, s2, fill= z - num)) +
  facet_grid(type ~ timeHM) + coord_fixed() +
  fill_scale(name = "dBZ") +
  theme_bw()
```

Next, we compute some of the cross-validation diagnostics. We consider the bias, the predictive cross-validation (PCV) and the standardized cross-validation (SCV) measures, and the continuous ranked probability score (CRPS). Functions for the first three are simple enough to code from scratch.

```
Bias <- function(x,y) mean(x - y)            # x: val. obs.
PCV <- function(x,y) mean((x - y)^2)         # y: predictions
SCV <- function(x,y,v) mean((x - y)^2 / v)   # v: pred. variances
```

The last one, CRPS, is a bit more tedious to implement, but it is available through the `crps` function of the **verification** package. The function `crps` returns, among other things, a field `CRPS` containing the average CRPS across all validation observations.

```
## Compute CRPS. s is the pred. standard error
CRPS <- function(x, y, s) verification::crps(x, cbind(y, s))$CRPS
```

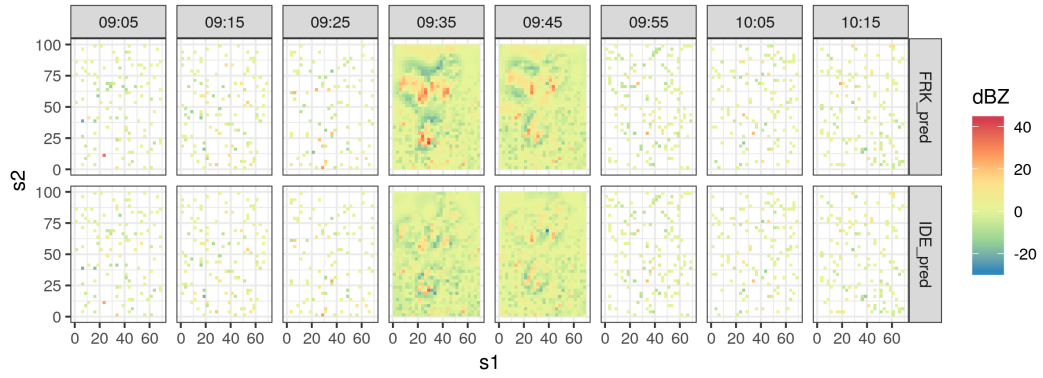

Figure 3: Spatial maps of prediction standard errors at the validation-data locations for the two missing time points and the adjoining six time points, based on the FRK model (top row) and the IDE model (bottom row).

Finally, we compute the diagnostics for each of the FRK and IDE models. In the code below, we show how to obtain them for the validation data at the missing time points; those for the validation data that are missing at random are obtained in a similar fashion. The diagnostics are summarized in Table 1, where it is clear that the IDE model outperforms the FRK model on most of the diagnostics considered here (note, in particular, the PCV for data associated with missing time points). For both models, we note that the SCV and CRPS need to be treated with care in a spatial or spatio-temporal setting, since the errors do exhibit some correlation, which is not taken into account when computing these measures.

```
Diagblock <- radar_valblock_df %>% summarise(
    Bias_FRK = Bias(FRK_pred, z),
    Bias_IDE = Bias(IDE_pred, z),
    PCV_FRK =  PCV(FRK_pred, z),
    PCV_IDE =  PCV(IDE_pred, z),
    SCV_FRK =  SCV(FRK_pred, z, FRK_predZse^2),
    SCV_IDE =  SCV(IDE_pred, z, IDE_predZse^2),
    CRPS_FRK = CRPS(z, FRK_pred, FRK_predZse),
    CRPS_IDE = CRPS(z, IDE_pred, IDE_predZse)
)
```

The multivariate energy score (ES) and variogram score of order p (VS_p) are available in R in the **scoringRules** package. The two functions we shall be using are `es_sample`

Table 1: Cross-validation diagnostics for the FRK and IDE models fitted to the Sydney radar data set on data that are left out for two entire time intervals (top row) and at random (bottom row). The IDE model fares better for most diagnostics considered here, namely the bias (closer to zero is better), the predictive cross-validation measure (PCV, lower is better), the standardized cross-validation measure (SCV, closer to 1 is better), and the continuous ranked probability score (CRPS, lower is better)

	Bias		PCV		SCV		CRPS	
	FRK	IDE	FRK	IDE	FRK	IDE	FRK	IDE
Missing time points	-0.27	0.61	50.81	29.54	1.33	0.56	3.75	3.02
Missing at random	-0.07	-0.06	34.20	26.79	0.78	0.88	3.14	2.74

and **vs_sample**. However, to compute these scores, we first need to simulate forecasts from the predictive distribution. To do this, we not only need the marginal prediction variances, but also all the prediction covariances. Due to the size of the prediction covariance matrices, multivariate scoring can only be done on at most a few thousand predictions at a time.

For this part of the Lab, we consider the validation data at 09:35 from the Sydney radar data set.

```
radar_val0935 <- subset(radar_valblock,
                        radar_valblock$timeHM == "09:35")
n_0935 <- length(radar_val0935)   # number of validation data
```

To predict with the IDE model and store the covariances, we simply set the argument `covariances` to `TRUE`.

```
pred_IDE_block <- predict(fit_results_radar2$IDEmodel,
                          newdata = radar_val0935,
                          covariances = TRUE)
```

To predict with the FRK model and store the covariances, we also set the argument `covariances` to `TRUE`.

```
FRK_pred_block <- predict(S,
                          newdata = radar_val0935,
                          covariances = TRUE)
```

The returned objects are lists that contain the predictions in the item `newdata` and the covariances in an item `Cov`. Now, both **es_sample** and **vs_sample** are designed to

compare a *sample* of forecasts to data, and therefore we need to simulate some realizations from the predictive distribution before calling these functions.

Recalling Lab 5.1, one of the easiest ways to simulate from a Gaussian random vector $\mathbf{x}$ with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$ is to compute the lower Cholesky factor of $\boldsymbol{\Sigma}$, call this $\mathbf{L}$, and then to compute

$$\mathbf{Z}_{\text{sim}} = \boldsymbol{\mu} + \mathbf{L}\mathbf{e},$$

where $\mathbf{e} \sim iid\ Gau(\mathbf{0}, \mathbf{I})$. In our case, $\boldsymbol{\mu}$ contains the estimated intercept plus the predictions, while $\mathbf{L}$ is the lower Cholesky factor of whatever covariance matrix was returned in `Cov` with the measurement-error variance, σ^2_ϵ, added onto the diagonal (since we are validating against observations, and not process values). Recall that we have set σ^2_ϵ to be the same for the FRK and the IDE models.

```
Veps <- diag(rep(sigma2_eps, n_0935))
```

Now the Cholesky factors of the predictive covariance matrices for the IDE and FRK models are given by the following commands.

```
L_IDE <- t(chol(pred_IDE_block$Cov + Veps))
L_FRK <- t(chol(FRK_pred_block$Cov + Veps))
```

The intercepts estimated by both models are given by the following commands.

```
IntIDE <- coef(fit_results_radar2$IDEmodel)
IntFRK <- coef(S)
```

We can generate 100 simulations at once by adding on the mean component (intercept plus prediction) to 100 realizations simulated using the Cholesky factor as follows.

```
nsim <- 100
E <- matrix(rnorm(n_0935*nsim), n_0935, nsim)
Sims_IDE <- IntIDE + pred_IDE_block$newdata$Ypred + L_IDE %*% E
Sims_FRK <- IntFRK + FRK_pred_block$newdata$mu + L_FRK %*% E
```

In Figure 4 we show one of the simulations for both the FRK and the IDE model, together with the validation data, at time point 09:35. Note how the IDE model is able to capture more structure in the predictions than the FRK model.

```
## Put into long format
radar_val0935_long <- cbind(data.frame(radar_val0935),
                            IDE = Sims_IDE[,1],
                            FRK = Sims_FRK[,1]) %>%
                      gather(type, val, z, FRK, IDE)
```

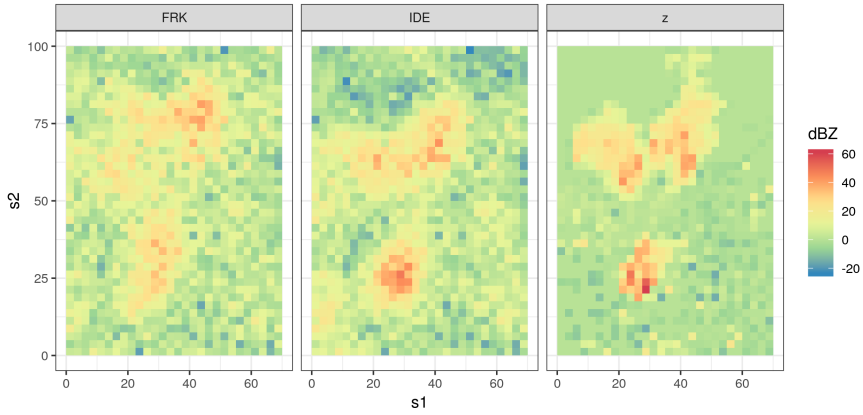

Figure 4: One of the 100 simulations from the predictive distribution of the FRK model (left) and the IDE model (center), and the data (not used to train the model, right) at 09:35.

```
## Plot
gsims <- ggplot(radar_val0935_long) +
    geom_tile(aes(s1, s2, fill = val)) +
  facet_grid(~ type) + theme_bw() + coord_fixed() +
  fill_scale(name = "dBZ")
```

We now compute the ES for both models by supplying the data and the simulations in matrix form to **es_sample**.

```
es_sample(radar_val0935$z, dat = as.matrix(Sims_IDE))

## [1] 144

es_sample(radar_val0935$z, dat = as.matrix(Sims_FRK))

## [1] 205
```

As with all proper scoring rules, lower is better, and we clearly see in this case that the IDE model has a lower ES than that for the FRK model for these validation data. For VS_p, we also need to specify weights. Here we follow the example given in the help file of **vs_sample** and set $w_{ij} = 0.5^{d_{ij}}$, where d_{ij} is the distance between the ith and jth prediction locations.

```
distances <- radar_val0935 %>%
           coordinates() %>%
           dist() %>%
```

```
            as.matrix()
weights <- 0.5^distances
```

The function **vs_sample** is then called in a similar way to **es_sample**, but this time specifying the weights and the order (we chose $p = 1$).

```
vs_sample(radar_val0935$z, dat = as.matrix(Sims_IDE),
          w = weights, p = 1)

## [1] 66965

vs_sample(radar_val0935$z, dat = as.matrix(Sims_FRK),
          w = weights, p = 1)

## [1] 78535
```

As expected, we find that the IDE model has a lower VS_1 than the FRK model. Thus, the IDE model in this case has provided better probabilistic predictions than the FRK model, both marginally and jointly.

Step 6: Model Comparison

We conclude this Lab by evaluating the Akaike information criterion (AIC) and Bayesian information criterion (BIC) for the two models. Recall that the AIC and BIC of a model $\mathcal{M}_\ell$ with p_l parameters estimated with m^* data points are given by

$$AIC(\mathcal{M}_\ell) = -2\log p(\mathbf{Z}|\widehat{\boldsymbol{\theta}}, \mathcal{M}_\ell) + 2p_l,$$
$$BIC(\mathcal{M}_\ell) = -2\log p(\mathbf{Z}|\widehat{\boldsymbol{\theta}}, \mathcal{M}_\ell) + \log(m^*)p_l.$$

For both AIC and BIC, we need the log-likelihood of the model at the estimated parameters. For the models we consider, these can be extracted using the function **loglik** in **FRK** and the negative of the function **negloglik** supplied with the `IDE` object.

```
loglikFRK <- FRK:::loglik(S)
loglikIDE <- -fit_results_radar2$IDEmodel$negloglik()
```

Before we can compute the AIC and BIC for our models, we first need to find out how many parameters were estimated. For the IDE model, the intercept, two variance parameters (one for measurement error and one for the temporal invariant disturbance term) and four kernel parameters were estimated, for a total of seven parameters. For the FRK model, the intercept, four variance parameters (one for measurement error, one for fine-scale variation, and one for each resolution of the basis functions) and four length-scale parameters (one spatial and one temporal length-scale for each resolution) were estimated, for a total of nine parameters.

```
pIDE <- 7
pFRK <- 9
```

The total number of data points used to fit the two models is

```
m <- length(radar_obs)
```

We now find the AIC and BIC for both models.

```
## Initialize table
Criteria <- data.frame(AIC = c(0, 0), BIC = c(0, 0),
                       row.names = c("FRK", "IDE"))

## Compute criteria
Criteria["FRK", "AIC"] <- -2*loglikFRK + 2*pFRK
Criteria["IDE", "AIC"] <- -2*loglikIDE + 2*pIDE
Criteria["FRK", "BIC"] <- -2*loglikFRK + pFRK*log(m)
Criteria["IDE", "BIC"] <- -2*loglikIDE + pIDE*log(m)
Criteria

##       AIC   BIC
## FRK 65992 66057
## IDE 45701 45751
```

Both the AIC and BIC are much smaller for the IDE model than for the FRK model. When the difference in the criteria is so large (in this case around 10,000), it safe to conclude that one model is a much better representation of the data. Combined with the other visualizations and diagnostics, we can conclude that the IDE model is preferrable to the FRK model for modeling and predicting with the Sydney radar data set.

As a final note, the AIC and BIC are not really appropriate for model selection in the presence of dependent random effects as the effective number of parameters in such settings is more than the number of parameters describing the fixed effects and covariance functions, and less than this number plus the number of basis-function coefficients (due to dependence; e.g., Hodges and Sargent, 2001; Overholser and Xu, 2014). Excluding the number of basis functions (i.e., the number of random effects) when computing the AIC and BIC clearly results in optimistic criteria; other measures such as the conditional AIC (e.g., Overholser and Xu, 2014) or the DIC, WAIC, and PPL are more suitable for such problems.

Bibliography

Hodges, J. S. and Sargent, D. J. (2001), "Counting degrees of freedom in hierarchical and other richly-parameterised models," *Biometrika*, 88, 367–379.

Overholser, R. and Xu, R. (2014), "Effective degrees of freedom and its application to conditional AIC for linear mixed-effects models with correlated error structures," *Journal of Multivariate Analysis*, 132, 160–170.