Spatio-temporal integro-difference equation (IDE) modelling: The `R package`

Andrew Zammit-Mangion

May 5, 2018

In this vignette we use the package **IDE** to fit spatio-temporal integro-difference equation (IDE) models, as well as predict and forecast from spatio-temporal data. We will explore two cases using simulated data where the true model is known. For this vignette, we need the following packages:

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

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

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

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 it is assumed here that the parameter vector $\boldsymbol{\theta}_p$ does not vary with time, as one often does, but it could 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 redistribution 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) 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.

Simulation example with a spatially-invariant kernel

The package **IDE** contains a function `simIDE` that simulates the behaviour of a typical dynamic system governed by linear transport. The function takes just three arguments, the number of time points to simulate, the number of (spatially fixed) observations to use, and a flag indicating whether to use a spatially-invariant kernel or not.

```
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 also 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 transport action is clearly noticeable in the process evolution, 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 dataset, which needs to be of class `STIDF`, the temporal discretization to use (we will use 1 day) of class `difftime`, and the gridsize on which the integrations (as well as predictions) will be carried out. Other arguments include user-specified basis functions for the process and the transition kernel, which for now we will not specify. By default, the IDE model will decompose the process using two resolutions of bisquare basis functions, and 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 for $\boldsymbol{\alpha}_t, t = 1, \ldots, T$ at these initial parameter estimates. The parameters in this case are the measurement-error variance, the variance of the random disturbance $\eta_t(\mathbf{s})$ (whose covariance structure is fixed), the kernel parameters, and the regression coefficients $\boldsymbol{\beta}$.

Fitting the IDE 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 machine but can take an hour or two on a standard desktop. Fitting can be done by running

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

where `parallelType = 1` ensures that all available cores on the machine are used for fitting.

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)
```

The fitted kernel is shifted to the left and upwards, correctly representing the south-easterly transport evident in the data. The estimated kernel parameters $\boldsymbol{\theta}_p$ are

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

which can be compared to the true values `c(150, 0.002, -0.1, 0.1)`. The estimated amplitude is lower and the aperture is larger than those values used to simulate the true field, but the shift parameters are remarkably similar. The estimated regression coefficients are

```
fit_results_sim1$IDEmodel$get("betahat")
```

which also compare well to the true values `c(0.2, 0.2, 0.2)`. Also of interest are the modulus of the 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)
```

Since the largest of these is less than 1, the IDE process 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 data temporal horizon (forecasts and hindcasts are explored later).

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

The prediction and prediction standard error 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) +
  scale_fill_distiller(palette="Spectral", 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)) +
  facet_wrap(~t) +
  scale_fill_distiller(palette="Spectral") +
  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. Note how prediction standard errors are high 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 $m(\mathbf{s}, \mathbf{r}; \boldsymbol{\theta}_p)$ is just a function of $\|\mathbf{r} - \mathbf{s}\|$. 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, considerabe spatially-varying drift. One such process can be simulated using the function **simIDE** and specifying `k_spat_invariant = 0`. To model such data we need to have a large `nobs` and many time points, and we set `T = 15`. This is important, as it is difficult to obtain reasonable estimates of spatially distributed parameters unless the data covers a large part of the spatial domain for at least a few consecutive time points.

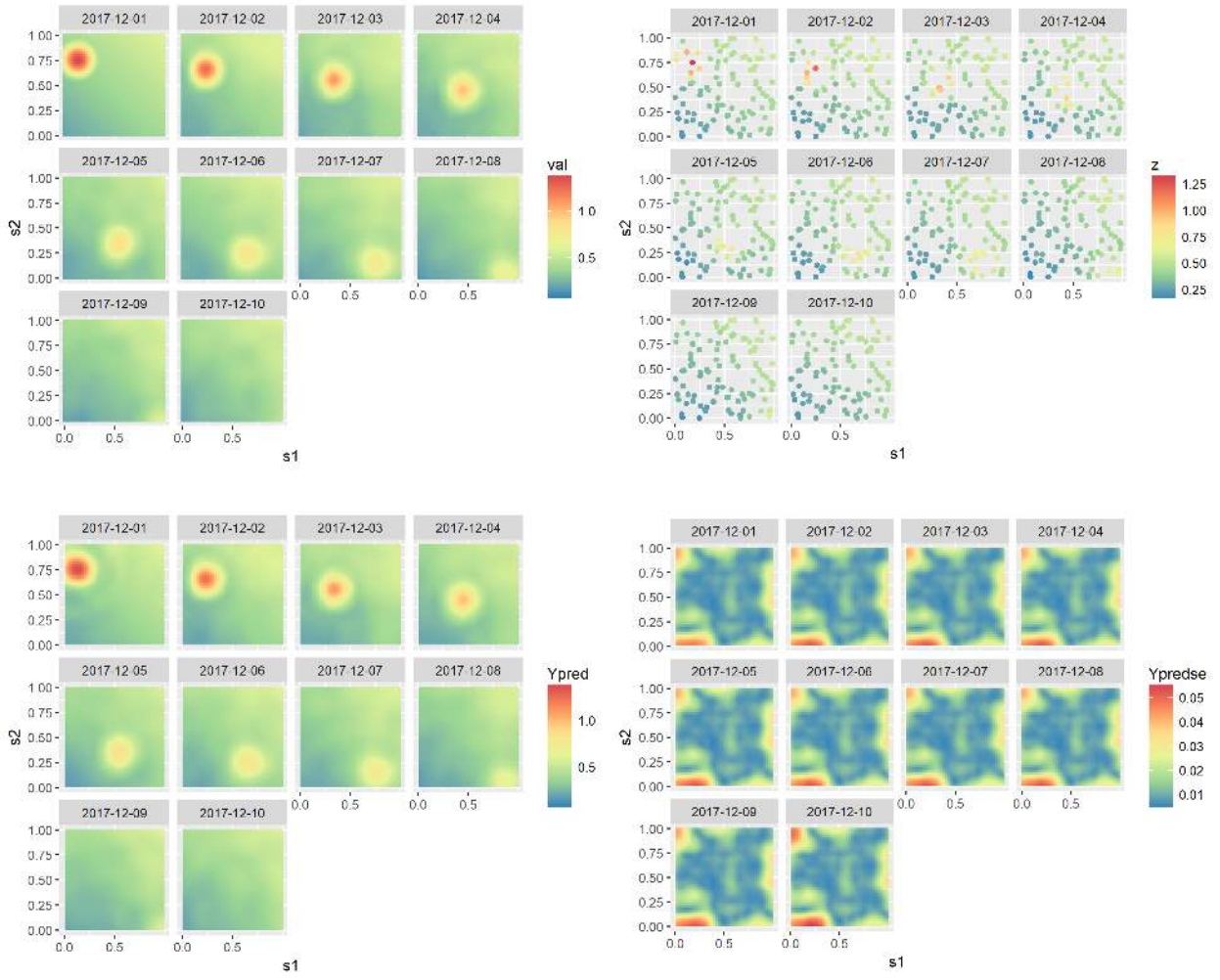

Figure 1: Simulated process (top-left panel), simulated data (top-right panel), predictions following the fitting of the IDE model (bottom-left panel) and the respective prediction standard errors (bottom-right panel).

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

As above, the process and the data can be plotted through

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

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

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

In this command, the argument `scale` scales 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 the of the same class as those used by **FRK**. Below we construct 9 bisquare basis functions that are equally spaced in the domain.

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

Type `show_basis(mbasis_1)` to plot these basis functions.

Now, recall that $\theta_{p,1}$ corresponds to the amplitude of the kernel, $\theta_{p,2}$ to the scale or aperture, $\theta_{p,3}$ to the x-direction shift "drift," and $\theta_{p,4}$ to the y-direction shift (drift). Here we decide to let $\theta_{p,1}$ and $\theta_{p,2}$ be 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)
```

Modelling 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`. Below we use the argument `itermax = 400` 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, be warned that this operation is very computationally intensive. The fitted spatially-varying kernel can be visualized through

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

and 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 model. Predictions and prediction standard errors can be obtained and visualized using `predict` as above.

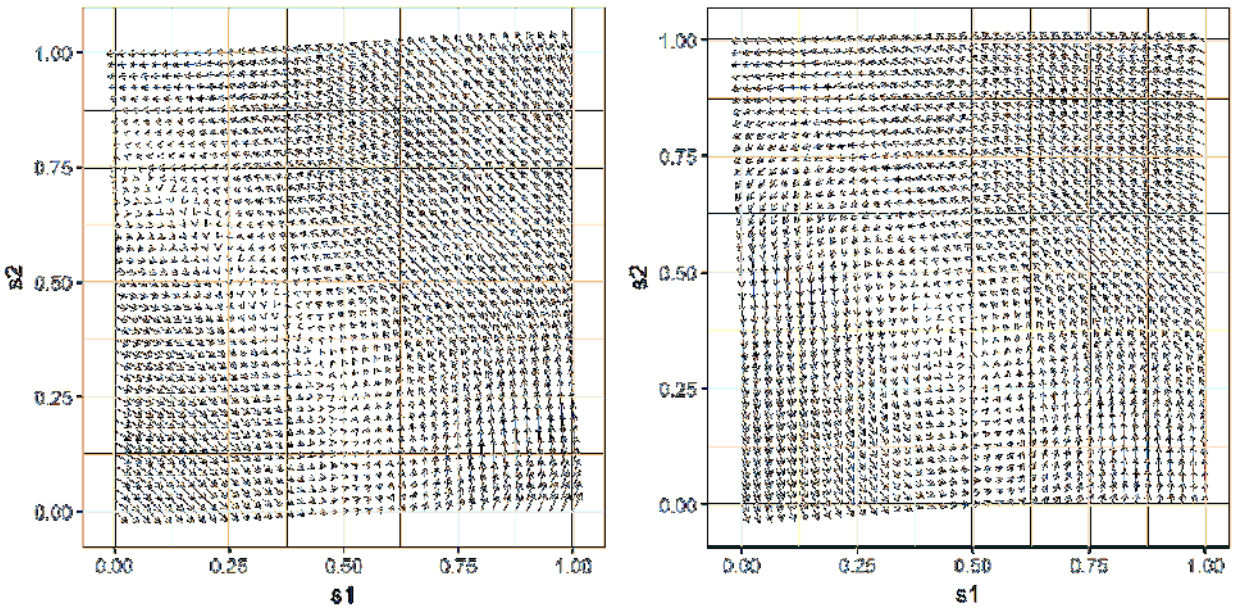

Figure 2: True drifts (left panel) and estimated drifts (right panel).