Lab 5.1: Implementing an IDE Model in One-Dimensional Space

In this Lab we take a look at how one can implement a stochastic integro-difference equation (IDE) in one-dimensional space and time, from first principles. Specifically, we shall consider the dynamic model,

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

where $Y_t(\cdot)$ is the spatio-temporal process at time t; $\boldsymbol{\theta}_p$ are parameters that we fix (in practice, these will be estimated from data; see Lab 5.2); and $\eta_t(\cdot)$ is a spatial process, independent of $Y_t(\cdot)$, with covariance function that we shall assume is known.

We only need the packages **dplyr**, **ggplot2**, and **STRbook** for this lab and, for reproducibility purposes, we fix the seed.

```
library("dplyr")
library("ggplot2")
library("STRbook")
set.seed(1)
```

Constructing the Process Grid and Kernel

We start off by constructing a discretization of the one-dimensional spatial domain $D_s = [0, 1]$. We shall use this discretization, containing cells of width Δ_s, for both approximate integrations as well as visualizations. We call this our spatial grid.

```
ds <- 0.01
s_grid <- seq(0, 1, by = ds)
N <- length(s_grid)
```

Our space-time grid is formed by calling `expand.grid` with `s_grid` and our temporal domain, which we define as the set of integers spanning 0 up to $T = 200$.

```
nT <- 201
t_grid <- 0:(nT-1)
st_grid <- expand.grid(s = s_grid, t = t_grid)
```

The transition kernel $m(s, x; \boldsymbol{\theta}_p)$ is a bivariate function on our spatial grid. It is defined below to be a Gaussian kernel, where the entries of $\boldsymbol{\theta}_p = (\theta_{p,1}, \theta_{p,2}, \theta_{p,3})'$ are the amplitude, the scale (aperture, twice the variance), and the shift (offset) of the kernel, respectively. Specifically,

$$m(s, x; \boldsymbol{\theta}_p) \equiv \theta_{p,1} \exp\left(-\frac{1}{\theta_{p,2}}(x - \theta_{p,3} - s)^2\right),$$

which can be implemented as an `R` function as follows.

```
m <- function(s, x, thetap) {
  gamma <- thetap[1]                    # amplitude
  l <- thetap[2]                        # length scale
  offset <- thetap[3]                   # offset
  D <- outer(s + offset, x, '-')        # displacements
  gamma * exp(-D^2/l)                   # kernel eval.
}
```

Note the use of the function **outer** with the subtraction operator. This function performs an "outer operation" (a generalization of the outer product) by computing an operation between every two elements of the first two arguments, in this case a subtraction.

We can now visualize some kernels by seeing how the process at $s = 0.5$ depends on x. Four such kernels are constructed below: the first is narrow and centered on 0.5; the second is slightly wider; the third is shifted to the right; and the fourth is shifted to the left. We store the parameters of the four different kernels in a list `thetap`.

```
thetap <- list()
thetap[[1]] <- c(40, 0.0002, 0)
thetap[[2]] <- c(5.75, 0.01, 0)
thetap[[3]] <- c(8, 0.005, 0.1)
thetap[[4]] <- c(8, 0.005, -0.1)
```

Plotting proceeds by first evaluating the kernel for all x at $s = 0.5$, and then plotting these evaluations against x. The first kernel is plotted below; plotting the other three is left as an exercise for the reader.

```
m_x_0.5 <- m(s = 0.5, x = s_grid,              # construct kernel
             thetap = thetap[[1]]) %>%         # at s = 0.5
           as.numeric()                        # convert to numeric
df <- data.frame(x = s_grid, m = m_x_0.5)       # allocate to df
ggplot(df) + geom_line(aes(x, m)) + theme_bw() # plot
```

The last term we need to define is $\eta_t(\cdot)$. Here, we define it as a spatial process with an exponential covariance function with range parameter 0.1 and variance 0.1. The covariance matrix at each time point is then

```
Sigma_eta <- 0.1 * exp( -abs(outer(s_grid, s_grid, '-') / 0.1))
```

Simulating $\eta_t(s)$ over `s_grid` proceeds by generating a multivariate Gaussian vector with mean zero and covariance matrix `Sigma_eta`. To do this, one can use the function **mvrnorm** from the package **MASS**. Alternatively, one may use the lower Cholesky factor of `Sigma_eta` and multiply this by a vector of numbers generated from a mean-zero, variance-one, independent-elements Gaussian random vector (see Rue and Held, 2005, Algorithm 2.3).

```
L <- t(chol(Sigma_eta))  # chol() returns upper Cholesky factor
sim <- L %*% rnorm(nrow(Sigma_eta))  # simulate
```

Type `plot(s_grid, sim, 'l')` to plot this realization of $\eta_t(s)$ over `s_grid`.

Simulating from the IDE

Now we have everything in place to simulate from the IDE. Simulation is most easily carried out using a `for` loop as shown below. We shall carry out four simulations, one for each kernel constructed above, and store the simulations in a list of four data frames, one for each simulation. The following command initializes this list.

```
Y <- list()
```

For each simulation setting (which we iterate using the index `i`), we simulate the time points (which we iterate using `j`) to obtain the process. The "nested `for` loop" below accomplishes this. In the outer loop, the kernel is constructed and the process is initialized to zero. In the inner loop, the integration is approximated using a Riemann sum,

$$\int_{D_s} m(s, x; \boldsymbol{\theta}_p) Y_{t-1}(x) \mathrm{d}x \approx \sum_i m(s, x_i; \boldsymbol{\theta}_p) Y_{t-1}(x_i) \Delta_s,$$

where we recall that we have set $\Delta_s = 0.01$. Next, at every time point $\eta_t(s)$ is simulated on the grid and added to the sum (an approximation of the integral) above.

```
for(i in 1:4) {                              # for each kernel
  M <- m(s_grid, s_grid, thetap[[i]])        # construct kernel
  Y[[i]] <- data.frame(s = s_grid,           # init. data frame with s
                       t = 0,                # init. time point 0, and
                       Y = 0)                # init. proc. value = 0
  for(j in t_grid[-1]) {                     # for each time point
    prev_Y <- filter(Y[[i]],                 # get Y at t - 1
                     t == j - 1)$Y
    eta <- L %*% rnorm(N)                    # simulate eta
    new_Y <- (M %*% prev_Y * ds + eta) %>%
             as.numeric()                    # Euler approximation

    Y[[i]] <- rbind(Y[[i]],                  # update data frame
                    data.frame(s = s_grid,
                               t = j,
                               Y =  new_Y))
  }
}
```

Repeatedly appending data frames, as is done above, is computationally inefficient. For large systems it would be quicker to save a data frame for each time point in another list and then concatenate using `rbindlist` from the package **data.table**.

Since now `Y[[i]]`, for `i` $= 1, \ldots, 4$, contains a data frame in long format, it is straightforward to visualize. The code given below constructs the Hovmöller plot for the IDE process for `i` $= 1$. Plotting for `i` $= 2, 3, 4$ is left as an exercise for the reader.

```
ggplot(Y[[1]]) + geom_tile(aes(s, t, fill = Y)) +
    scale_y_reverse() + theme_bw() +
    fill_scale(name = "Y")
```

Simulating Observations

Now assume that we want to simulate noisy observations from one of the process models that we have just simulated from. Why would we want to do this? Frequently, the only way to test whether algorithms for inference are working as they should is to mimic both the underlying true process *and* the measurement process. Working with simulated data is the first step in developing reliable algorithms that are then ready to be applied to real data.

To map the observations to the data we need an incidence matrix that picks out the process value that has been observed. This incidence matrix is simply composed of several rows, one for each observation, with zeros everywhere except for the entry corresponding to the process value that has been observed.When the locations we are observing change over time, the incidence matrix correspondingly changes over time.

Suppose that at each time point we observe the process at 50 locations which, for convenience, are a subset of `s_grid`. (If this is not the case, some nearest-neighbor mapping or deterministic interpolation method can be used.)

```
nobs <- 50
sobs <- sample(s_grid, nobs)
```

Then the incidence matrix at time t, $\mathbf{H}_t$, can be constructed by matching the observation locations on the space-time grid using the function `which`.

```
Ht <- matrix(0, nobs, N)              # construct empty matrix
for(i in 1:nobs) {                    # for each obs
  idx <- which(sobs[i] == s_grid)     # find the element to set to 1
  Ht[i, idx] <- 1                     # set to 1
}
```

Note that `Ht` is sparse (contains many zeros), so sparse-matrix representations can be used to improve computational and memory efficiency; look up the packages **Matrix** or **spam** for more information on these representations.

We can repeat this procedure for every time point to simulate our data. At time t, the data are given by $\mathbf{Z}_t = \mathbf{H}_t\mathbf{Y}_t + \boldsymbol{\epsilon}_t$, where $\mathbf{Y}_t$ is the latent process on the grid at time t, and $\boldsymbol{\epsilon}_t$ is independent of $\mathbf{Y}_t$ and represents a Gaussian random vector whose entries are iid with mean zero and variance σ^2_ϵ. Assume $\sigma^2_\epsilon = 1$ and that $\mathbf{H}_t$ is the same for each t. Then observations are simulated using the following `for` loop.

```
z_df <- data.frame()                # init data frame
for(j in 0:(nT-1)) {                # for each time point
  Yt <- filter(Y[[1]], t == j)$Y    # get the simulated process
  zt <- Ht %*% Yt + rnorm(nobs)     # map to obs and add noise
  z_df <- rbind(z_df,               # update data frame
                data.frame(s = sobs, t = j, z = zt))
}
```

Plotting of the simulated observations proceeds using **ggplot2** as follows.

```
ggplot(z_df) + geom_point(aes(s, t, colour = z))  +
  col_scale(name = "z") + scale_y_reverse() + theme_bw()
```

Note that the observations are noisy and reveal sizeable gaps. Filling in these gaps by first estimating all the parameters in the IDE from the data and then predicting at unobserved locations is the subject of Lab 5.2.

Bibliography

Rue, H. and Held, L. (2005), *Gaussian Markov Random Fields: Theory and Applications*, Boca Raton, FL: Chapman & Hall/CRC.