Reproducible code for Section 4.1 of 'Spatio-temporal bivariate statistical models for atmospheric trace-gas inversion'

Andrew Zammit Mangion*

August 29, 2015

Abstract

Here we present repdoducible code for the simulation study appearing in the paper 'Spatio-temporal bivariate statistical models for atmospheric trace-gas inversion', Section 4.1. The code requires the installation of two in-house developed packages for this application, `hmc` and `atminv`. The vignette itself is not 'polished', but gives the basic requirements for reproducing the figures and values given in the main text.

1 Setup

To run the simulation study, you need to first install the packages `atminv` and `hmc`. You can do this as follows:

```
library(devtools)
install_github("andrewzm/hmc")
install_github("andrewzm/atminv")
```

This only needs to be done once. Now that we have the development packages installed, we can now load the others that we will need. The first two, `ggplot2` and `grid` are for plotting purposes. The package `dplyr` is used for fast table manipulation and for 'piping' a sequence of commands. The package `Matrix` is needed for taking advantage of sparsity in some operations and `tidyr` is needed for rearranging tables. The package `gstat` is needed for variogram modelling of the flux field. Finally the in-house developed packages `hmc` and `atminv` are used for implementing the Hamiltonian Monte Carlo sampler and the EM algorithm for parameter estimation, respectively.

```
library(ggplot2)
library(grid)
library(dplyr)
library(Matrix)
library(tidyr)
library(gstat)
library(hmc)
library(atminv)
```

We will now set up our simulation. Here we will only be concerned with the model 'full' ('full_big' analyses the case with 1000 observations and 'diag' the case where the flux field is uncorrelated), which can be misspecified (`misspecification = 1`) or not (`misspecification =`

*National Institute for Applied Statistics Research Australia (NIASRA), University of Wollongong, Australia

0). Recall the by misspecification here we imply that the flux field is indeed spatially correlated, but is modelled as being uncorrelated. Below we set up the spatio-temporal grid and establish the parameters of the observation process and the mole-fraction discrepancy spatio-temporal field:

```
###------------------
### Parameters
###------------------
load_results  <- 0  # For quick vignette build
cache_results <- 0  # Development only
save_images <-   0  # Development only
model = "full"  ## Either sparse or full or full_big or diag
misspecification = 0
#set.seed(25)         # deprecated, ignore
ds <- 0.2             # 0.2 spacing.
                      # NB: If we change this we need to
                      # change the round() command further down
smin = -10 + ds/2     # first gridcell centre
smax =  10 - ds/2     # last gridcell centre

s_axis <- round(seq(smin,smax,by=ds),1) # create s-axis
if(model %in% c("full","diag")) {
  t_axis <- 1:100                       # create t-axis
  m_obs <- 6                            # number of obs. (including val.)
} else {
  t_axis <- 1:100                       # create t-axis
  m_obs <- 1000                         # number of obs.
}

ns <- length(s_axis)                    # no. of gridcells
nt <- length(t_axis)                    # no. of time points
st_grid <- expand.grid(s=s_axis,        # ST-grid (long format)
                       t=t_axis) %>%
            data.frame()

sigma_eps_true <- 10                    # observation error std.
if(model %in% c("full","full_big")) {
  sigma_zeta_true <- 50                 # discrepancy marginal std.
} else {
  sigma_zeta_true <- 10                 # discrepancy marginal std.
}

theta_t_true <- 0.8     # temporal correlation parameter ('a' in text)
theta_s_true <- 1       # spatial range parameter ('d' in text)
```

Now we are ready to create a stochastic process, the realisations of which exhibit similar characteristics to what we will be studying in the real example. Recall that the stochastic process we use is:

$$b_t(s,u \mid v_t(s)) \equiv \exp\left(-\frac{(u-s)^2}{2v_t(s)^2}\right) I\big(|u-s| < |v_t(s)|\big) J(s,u), \tag{1}$$

where

$$J(s,u) \equiv \begin{cases} I[(u-s) \geq 0]; & \upsilon_t(s) \geq 0, \\ I[(u-s) \leq 0]; & \upsilon_t(s) < 0, \end{cases}$$

where $\upsilon_t(s)$ is generated from a Gaussian process with separable spatio-temporal covariance structure and $I(\cdot)$ is the indicator function. In (1), the exponential function describes a bell-shaped curve centred at $u = s$, while the indicator function truncates this curve at $u = s \pm \upsilon_t(s)$. The third term, $J(s,u)$, then truncates the bottom half of the function if $\upsilon_t(s) \geq 0$ and the upper half otherwise. The function $b_t(s,u \mid \upsilon_t(s))$ is implemented as follows

```
###------------------
### Transition kernel
###------------------
b <- function(s,u,p) {
  absp <- max(abs(p),0.2)
  absp*sqrt(2*pi) * dnorm(u,mean = s, sd =absp) *
    ((sign(p) == sign(u-s)) | (u-s) == 0) *
    (abs(u - s) < absp)
}
```

while the spatio-temporal Gaussian parameter is simulated from a separable field as follows:

```
## Sample the "wind" vector
Q_s <- GMRF_RW(n = length(s_axis),
               order = 2,
               precinc = 2000)@Q +
          0.001*.symDiagonal(length(s_axis))  # spatial precision
Q_t <- GMRF_RW(n = length(t_axis),
               order = 1,
               precinc = 20)@Q +
          0.1*.symDiagonal(length(t_axis))    # temporal precision

Q_full = as(kronecker(Q_t,Q_s),"dgCMatrix")   # spatio-temporal precision

G <- GMRF(mu = matrix(rep(0,nrow(Q_full))),   # GMRF with final precision
          Q = Q_full,n=nrow(Q_full))

# Load the seed we used to simulate this parameter
data(sim.Random.seed)
#load("~/Desktop/Chemometrics_results/sim.Random.seed.rda")

# Now simulate this parameter by sampling from the GMRF
# p is a ST process and reflects the std of the truncated Gaussian.
# This problem ONLY works if b is of relatively local scope. Once we
# have b which has a very large scope we get oscillations/instability
st_grid$p <- sample_GMRF(G,reps = 1)
```

If we want we can take a look at what the realisation of the parameter $\upsilon_t(s)$ looks like through

```
print(LinePlotTheme() +
        geom_tile(data=st_grid,aes(s,t,fill=p)) +
```

```
        scale_fill_gradient2(low="blue",high="red",mid="white")
      )
```

the result of which is depicted in Figure 1.

2 Process and observation simulation

2.1 Simulating the flux field

Now that we have the parameters in place, we can simulate our dataset. We first re-set the seed to '1', then construct a semi-variogram with the parameters identical to those estimated from the emissions data, before simulating our vector $\mathbf{Y}_f$:

```
###------------------
### Lognormal flux field
###------------------
set.seed(1)
variogram_model <- vgm(range = 3.334,     # Construct spherical semi-variogram
                       nugget = 0.00533,
                       psill = 0.80429,
                       model= "Sph")
S_f_log <- variogramLine(variogram_model, # Find covariance matrix Sigma_f
                         dist_vector = sp::spDists(matrix(s_axis),
                                                   matrix(s_axis)),
                          covariance = TRUE)
mu_f_log <- matrix(rep(5,length(s_axis)))         # Construct mu_f
Yf_sim <- exp(mu_f_log + t(chol(S_f_log)) %*%     # Simulate Y_f
                rnorm(n = length(s_axis)))
st_grid <- st_grid %>%                            # Append Y_f to data frame
          left_join(data.frame(s=s_axis,
                               Yf = Yf_sim))

## Joining by:  "s"

if(misspecification)  {               # If we are assuming misspecification
    S_f_log <- diag(diag(S_f_log))    # We over-write Sigma_f to be diagonal
}
```

2.2 Simulating the mole fraction field

Since the spatio-temporal discrepancy term can be hard to simulate without further approximations, we will just simulate it at the observation locations. Therefore, the mole fraction (excluding the discrepancy) is just a linear transformation of the flux field. For each space-time location we find the SRR between the whole domain (the source) and that point (the receptor) and multiply it by the flux field:

```
###------------------
### Mole fraction field
###------------------
## Since we cannot put the discrepancy everywhere (too large),
## we will just put it at the observation locations
```

```
print(LinePlotTheme() +
        geom_tile(data=st_grid,aes(s,t,fill=p)) +
        scale_fill_gradient2(low="blue",high="red",mid="white")
      )
```

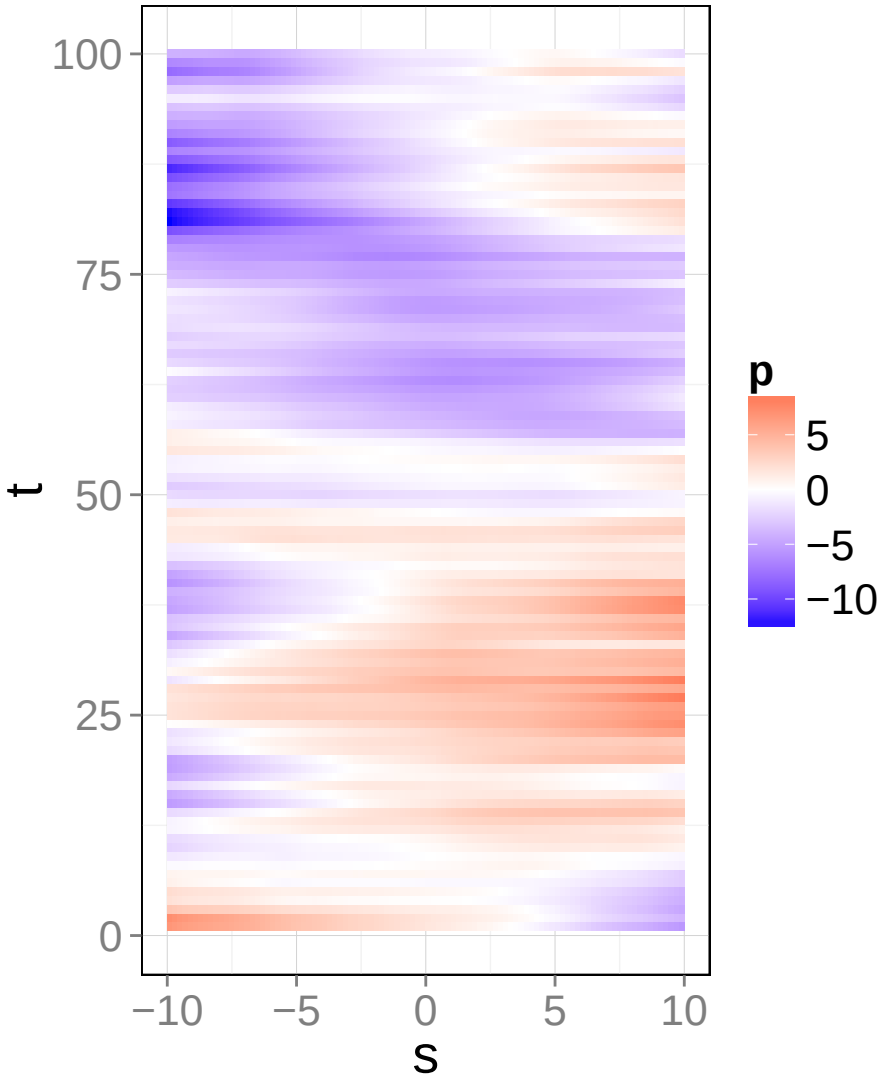

Figure 1: The parameter $v_t(s)$ simulated from the separable spatio-temporal process.

```
st_grid <- st_grid %>%
  group_by(t,s,p) %>% # for each space-time location
  summarise(Yf = Yf,  # find the mole-fraction by finding the SRR
            Ym = sum(b(s =s, u = s_axis,p = p) *
                       Yf_sim * ds))
```

2.3 Simulating the observations

We randomly choose `m_obs` observations from the spatial grid (excluding the lower and upper 10 grid cells) and assume these are observed. We replace the 6-th observation location with $s = 0.3$ that will be used for validation.

```
###------------------
### Observations
###------------------
if(model %in% c("full","diag")) {
      s_obs <- data.frame(s = sample(s_axis[-c(1:10,(ns-10):ns)],
                                     size = m_obs,
                                     replace=F),
                          m = 1:m_obs  )
    new_obs <- 0.3
    s_obs[6,]$s <- new_obs
} else {
  s_obs <- data.frame(s = sample(s_axis,size = m_obs, replace=T),
                      m = 1:m_obs)
}
```

Now we merge the observation data frame with the spaio-temporal grid and add the observation error, before sorting the data frame by time and space:

```
s_obs <- s_obs %>%
            left_join(st_grid) %>%
            mutate(z = Ym + rnorm(n = length(Ym),
                                  sd = sigma_eps_true)) %>%
            arrange(t,s)

## Joining by:  "s"

Qobs <- sigma_eps_true^(-2) * .symDiagonal(nrow(s_obs))
```

To add the discrepancy, we first compuate the spatio-temporal covariance matrix at the observation (space-time) locations using the `corr_zeta_fn` function in the `atminv` package, find its Cholesky decomposition and then use it to simulate the discrepancy at the required locations. For when we have 2000 observations (`model = 'full_big'`) we find the covariance matrix at every space-time grid location and use assign the discrepancy to the observations at the grid level. The variable `C_m` is a matrix that maps the location of the mole fraction observations to the mole fraction prediction grid. Note that for the 'full' model this is just the $m \times m$ since with this model we are choosing not predicting the mole fraction in every grid cell:

```r
## Now add the discrepancy
if(model %in% c("full","diag")) {

    corr_zeta_true <- corr_zeta_fn(s_obs$s[1:m_obs],
                                   t_axis,
                                   theta_t_true,
                                   theta_s_true)
    S_zeta_true <- sigma_zeta_true^2 * corr_zeta_true
    chol_S_zeta_true <- chol(S_zeta_true)

    s_obs <- s_obs %>%
      mutate(dis = t(chol_S_zeta_true) %*%  rnorm(n = nrow(s_obs)),
             z = z + dis)

    C_m <- .symDiagonal(nrow(s_obs))
} else if(model == "full_big") {

    corr_s_mat <- function(theta_s) atminv:::corr_s(s = s_axis,theta_s = theta_s)
    corr_t_mat <- function(theta_t) atminv:::corr_t(t = t_axis,theta_t = theta_t)
    d_corr_s_mat <- function(theta_s) atminv:::d_corr_s(s = s_axis,theta_s = theta_s)
    d_corr_t_mat <- function(theta_t) atminv:::d_corr_t(t = t_axis,theta_t = theta_t)

    C_idx <- st_grid %>%
      as.data.frame() %>%
      select(s,t) %>%
      mutate(n = 1:nrow(st_grid)) %>%
      left_join(s_obs,.)

    C_m <- sparseMatrix(i=1:nrow(C_idx),
                        j = C_idx$n,
                        x=1,
                        dims=c(nrow(s_obs),nrow(st_grid)))

    chol_S_zeta_true <-  sigma_zeta_true *
                             kronecker(chol(corr_t_mat(theta_t_true)),
                                       chol(corr_s_mat(theta_s_true)))

    s_obs <- s_obs %>%
        mutate(dis = as.vector(C_m %*%
                                   (t(chol_S_zeta_true) %*%
                                      rnorm(n = ns*nt))),
               z = z + dis)
}
```

2.4 Illustrative plots

Here we provide the code for generating the two plots in the paper (Figure 3). The first, in Figure 2, shows the average mole fraction and the flux field superimposed:

```
if(model == "full") print(g)
```

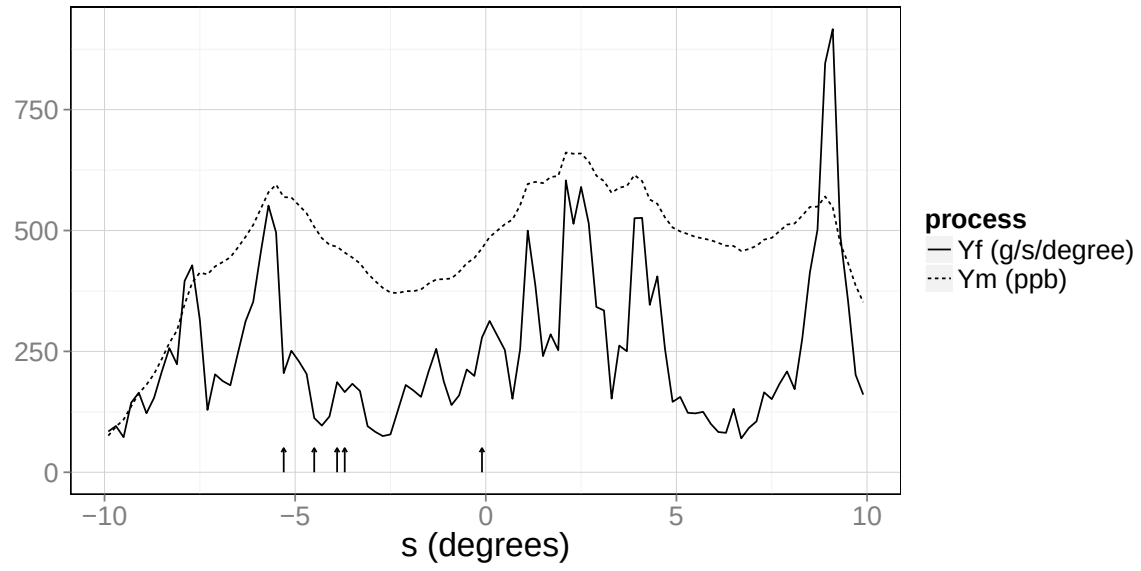

Figure 2: A sample realisation of the flux field (solid line), the resulting time-averaged mole-fraction field (dashed line) and the five observation locations (arrows).

```
if(model == "full") {
    X <- group_by(st_grid,s) %>%
        summarise(Yf = Yf[1],Ym_av = mean(Ym)) %>%
        gather(process,value,-s)
    g <- LinePlotTheme() +
          geom_line(data=subset(X,!(s==new_obs)),
                    aes(x=s,y=value,linetype=as.factor(process)))+
          geom_segment(data=s_obs[1:5,],
                       aes(x=s, xend=s, y = 0, yend = 50),
                        arrow=arrow(length=unit(0.1,"cm"))) +
        ylab("") +
        scale_linetype_discrete(guide=guide_legend(title="process"),
                                labels=c("Yf (g/s/degree)","Ym (ppb)")) +
        xlab("s (degrees)")
    if(save_images)
      ggsave(g,filename = "../img/Sim_plot.png",width=10,height=4)
}
```

The second, in Figure 3, shows the SRR at each observation location:

```
if(model %in% c("full","diag")) {
  df_for_B <- s_obs
} else {
  df_for_B <- st_grid
}

# TRUE B
```

```
B_true <- plyr::ddply(df_for_B,c("t","s"),function(df) {
  b = b(s=df$s[1],u=s_axis,p=df$p[1])}) %>%
  select(-s,-t) %>%
  as.matrix()*ds

B <- B_true
if(model == "sparse") {
  B <- as(B,"dgCMatrix")
}

B_true_df <- plyr::ddply(df_for_B,c("t","s"),function(df) {
  b = b(s=df$s[1],u=s_axis,p=df$p[1])}) %>%
  gather(s_grid,b,-t,-s) %>%
  separate(s_grid, into = c("V","s_fine"),sep="V") %>%
  select(-V) %>%
  mutate(s_fine = round(s_axis[as.numeric(s_fine)],1))# %>%
#left_join(model_pred_em,by = c("t","s","s_fine"))


if(model == "full") {
  ## Plot the source-receptor relationship at each observation

  ## The following code uses colours and shows the SRR on one plot
  #   B_plot <- LinePlotTheme() +
  #             geom_tile(data=subset(B_true_df ,b>0),
  #                       aes(x=s_fine,y=t,fill=as.factor(s),alpha=b)) +
  #     scale_alpha_continuous(range=c(0,1)) + xlab("u") +
  #     scale_fill_discrete(guide=guide_legend(title="s")) +
  #     coord_fixed(xlim=c(-10,10),ratio = 0.2)

  ## The following code shows one SRR per plot
  B_plot <- LinePlotTheme() +
    geom_tile(data=subset(B_true_df,b>0 & !(s==new_obs)),
              aes(x=s_fine,y=t,alpha=b),fill="black") +
    scale_alpha_continuous(guide=guide_legend(title="s/ng")) +
    scale_y_reverse()+
    scale_fill_discrete(guide=guide_legend(title="s")) +
    coord_fixed(xlim=c(-10,10),ratio = 0.5)  +
    facet_grid(~s) +
    theme(panel.margin = unit(1.5, "lines")) +
    xlab("u (degrees)") +
    ylab("t (2 h steps)")
  if(save_images)
    ggsave(filename = "../img/B_plot.png",width=12)
}
```

3 EM algorithm

In this section we show the code for carrying inference on the bivariate field $(\mathbf{Y}_f, \mathbf{Y}_m)'$. We first compute the mean flux density, which we will then also use as the initial value for the

```
if(model == "full") print(B_plot)
```

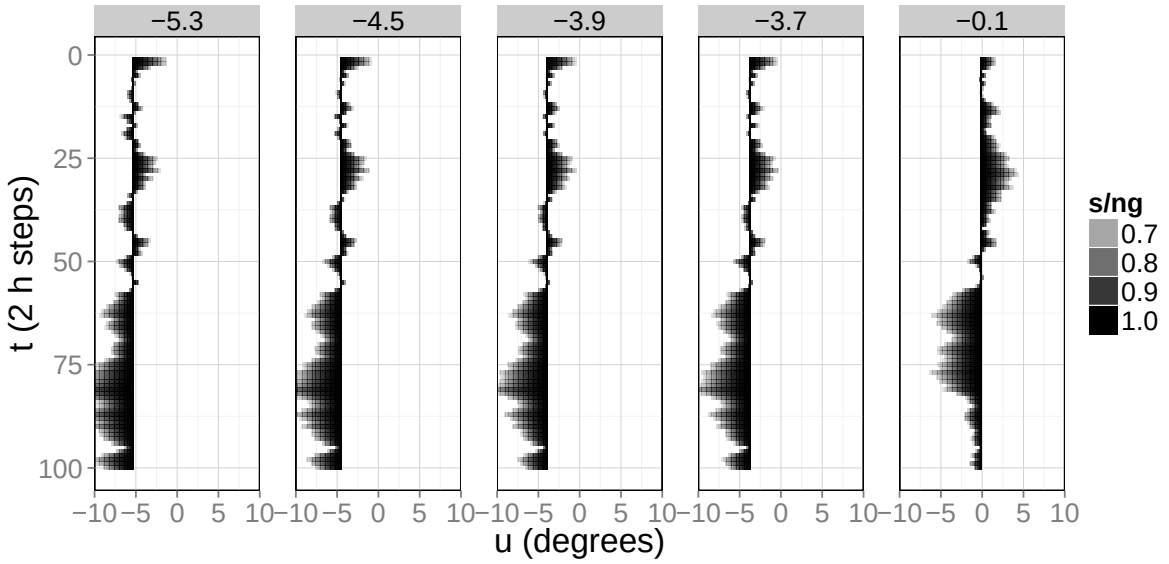

Figure 3: The source-receptor relationship $b_t(s,u)$ synthesised at five observation locations $s \in D_m^O = \{-5.3°, -4.5°, -3.9°, -3.7°, -0.1°\}$. Note that $b_t(s,u) = 0$ for $u > 4.3$.

conditional expectation of $\mathbf{Y}_f$ in the gradient descent:

```
mu_f <- matrix(exp(mu_f_log + 0.5*diag(S_f_log)))
```

We then set the settings for the Laplace approximation. These vary accoring to the model being used. For the model 'full', we alter the matrices so that the validation point is excluded.

```
###-----------------------------------------------------
### Laplace method -- use with caution because of mode close to zero
###-----------------------------------------------------


if(model == "full") {
  n_EM <- 100
  s_mol = s_obs$s[1:m_obs]          # prediction locs for mol fraction
  Y_init = c(mu_f,s_obs$z)          # initial expectation value for Y
  theta_init = c(1000,0.2,0.2)      # initial parameter vector
                                    # where theta = [sigma_zeta^2, theta_t, theat_s]

  # Keep station out for validation
  rm_idx <- seq(6,nrow(C_m),by=6) # index to be removed
  s_obs_old <- s_obs              # save old observation location data frame
  s_obs <- s_obs[-rm_idx,]        # new observation data frame
  C_m <- C_m[-rm_idx,]            # new incidence matrix
```

```
  Qobs <- Qobs[-rm_idx,-rm_idx]    # new (observation) precision matrix

} else if (model == "full_big") {
    n_EM <- 10
    s_mol = s_axis                 # prediction locs for mol fraction
    Y_init = c(mu_f,st_grid$Ym)    # initial expectation value for Y
    theta_init = c(1000,0.2,0.2)   # initial parameter vector
                                   # where theta = [sigma_zeta^2, theta_t, theat_s]
}
```

We are now ready to call the main function of the `EM_alg` package, which returns a function that can be iterated. The inputs to this function are given in-line:

```
EM_alg <- EM(s_obs = s_obs,                    # observation data frame
        C_m = C_m,                             # incidence matrix
        Qobs = Qobs,                           # observation precision matrix
        B = B,                                 # SRR matrix
        t_mol = t_axis,                        # time prediction locs
        s_mol = matrix(s_mol),                 # spatial prediction locs
        S_f_log = S_f_log,                     # cov. matrix of log(Yf)
        mu_f_log = mu_f_log,                   # expectation of log(Yf)
        Yf_thresh = 1e-4,                      # drop nodes with Yf < 1e-4
        Y_init =  Y_init,                      # initialise Yf
        theta_init = theta_init,               # initialise theta
        ind = which(!(colSums(B) == 0)),       # only consider indices with SRR>0
        n_EM = n_EM,                           # number of EM iterations
        model = model)                         # model we are using
```

To iterate the E- and M-steps we simply put the returned function in a loop, and indicate the maximum numer of gradient descents to carry out in the E- and M- steps. In this case we are going to limit the number of M-steps to 50. We also allow for a variable `fine_tune_E` that indicates on which iteration to carry out a second gradient descent at a higher convergence tolerance. This is particularly useful when the Hessian computed at 'convergence' is not positive definite due to the tolerance used.

```
filename <- paste0(model,"_misspec_",misspecification,".rda")
if(load_results) {
  load(system.file("extdata",filename, package = "atminv"))
} else {
  for(i in 1:(n_EM-2)) {
    X <- EM_alg(max_E_it = 1e6,
                max_M_it = 50,
                fine_tune_E = (i==0))
  }
}
if(cache_results & model == "full_big") {
    X$lap_approx <- NULL # Save space
    save(X,file=paste0("../inst/extdata/",filename))
}
```

The final parameter estimates are given by

```
plot(X$theta[1,])
plot(X$theta[2,])
plot(X$theta[3,])
```

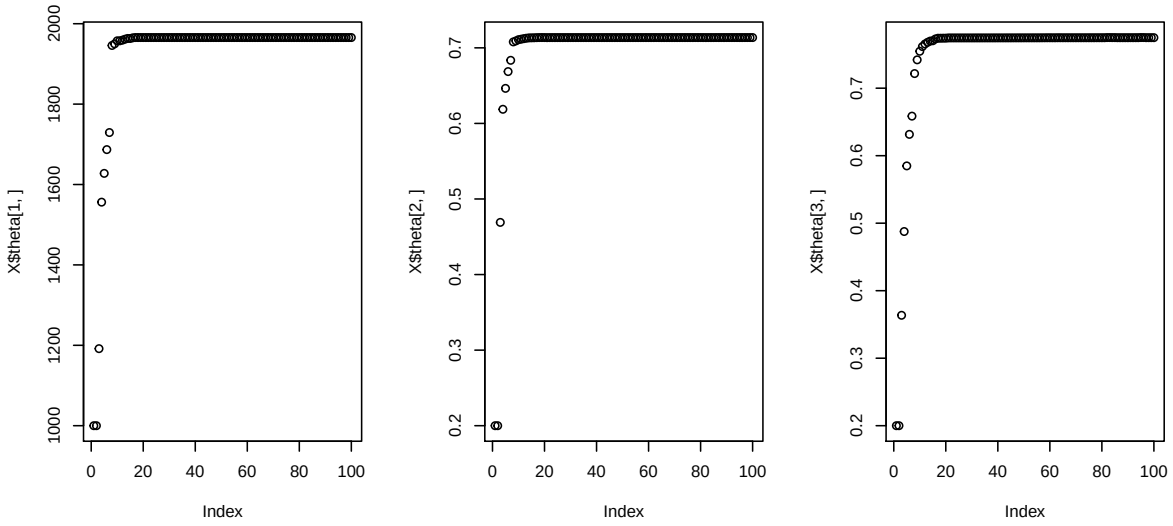

Figure 4: Convergence of the three parameters $\boldsymbol{\theta} = (\sigma_\zeta^2, a, d)'$. Left panel: The parameter σ_ζ^2. Centre panel: The parameter a. Right panel: The parameter d.

```
X$theta[,n_EM-1]

## [1] 1965.6092615    0.7136138    0.7748372
```

while a plot showing parameter convergene is given in Figure 4

4 HMC sampler

In this section we show the code for the HMC sampler, that fixes the parameters found using the EM algorithm above. We first compute the covariance matrix:

```
if(model == "full") {
  corr_zeta <- corr_zeta_fn(c(s_obs$s[1:(m_obs-1)],new_obs),
                            t_axis,X$theta[2,n_EM],X$theta[3,n_EM])
  S_zeta <- X$theta[1,n_EM] * corr_zeta
  Q_zeta <- chol2inv(chol(S_zeta))
}
```

Now we formulate the equations required for the log-likelihood and the gradient of the log-likelihood of $\mathbf{Y}_f$. These are provided in the helper function `Yf_marg_approx_fns` and its arguments are explained in-line.

```
lap2 <- Yf_marg_approx_fns(s = s_obs,              # obs. locations
                           C_m = C_m,              # incidence matrix
                           Qobs = Qobs,            # obs. precision matrix
```

```
                          B = B,                  # SRR matrix
                          S_zeta = S_zeta,        # Sigma_zeta
                          mu_f_log = mu_f_log,    # expectation of log(Y_f)
                          S_f_log = S_f_log,      # covariance of log(Y_f)
                          ind=X$ind)              # only consider indices with SRR>0
```

Next, we set some parameters for the sampler. We set the number of steps per sample $L = 10$, the number of samples $N = 10000$, and the step-size $\Delta \in [0.066, 0.068]$ (below denoted as 'eps_gen'). Other variables are defined in-line:

```
M <- diag(1/(X$lap_approx$Yf^2)) # scaling matrix: puts variables on same scale

if(model == "full"){             # Function for generating Delta
  eps_gen <- function()
    runif(n=1,
          min = 0.066,
          max = 0.068)
} else if (model =="diag") {
  eps_gen <- function()
    runif(n=1,
          min = 0.0065,
          max = 0.0067)
}
L <- 10L                    # step-size
N <- 10000                  # number of samples
q <- matrix(0,nrow(M),N)    # matrix for storing samples
qsamp <- X$lap_approx$Yf    # first sample
dither <- i <- count <- 1   # no dithering, initialise counts

 if(!(N==10000) & cache_results)
   stop("Cannot cache unless N =10000")
```

The sampler is set up using the function `sampler` in the `hmc` package. We set a lower limit of 0; no samples less or equal to zero are allowed:

```
sampler <- hmc_sampler(U = lap2$logf,                  # log-likelihood
                       dUdq = lap2$gr_logf,            # gr. of log-likelihood
                       M = M,                          # scaling matrix
                       eps_gen = eps_gen,              # step-size generator
                       L = L,                          # number of steps
                       lower = rep(0,length(X$ind)))   # lower limits
```

We now run the HMC sampler by repeatedly iterating through it. We also can monitor the acceptance rate. For conciseness the output below is not produced. We then calculate the acceptance rate again after the HMC is run.

```
if(!load_results) {
  while(i < N) {
    qsamp <- sampler(q = qsamp)

    if(count  == dither) {
```

```
      q[,i] <- qsamp
      count = 1
      i <- i + 1
      print(paste0("Sample: ",i," Acceptance rate: ",(nrow(unique(t(q)))-1)/i))
    } else {
      count <- count + 1
    }
  }
} else {
  load(system.file("extdata",filename, package = "atminv"))
}
if(cache_results)
    save(X,q,i,count,file=paste0("../inst/extdata/",filename))
```

```
print(paste0("Sample: ",i," Acceptance rate: ",(nrow(unique(t(q)))-1)/i))

## [1] "Sample: 10000 Acceptance rate: 0.5699"
```

5 Results

In this section we show the code used to generate the results. First, we put the samples into a format we can work with (the variable 'Q2') and then produce a box pot of the samples of the flux field at each of the prediction locations:

```
 Q <- as.data.frame(t(q[1:length(X$ind),-c(1:1000,i)])) %>%
  tidyr::gather(t,z) %>%
  separate(t, into=c("V","s"),sep="V") %>%
  select(-V) %>%
  mutate(s = s_axis[as.numeric(s)])

Q2 <- Q %>%
      mutate(s = as.factor(s))
g <- LinePlotTheme() +
  geom_boxplot(data=Q2,aes(x=s,y=z))  +
  scale_x_discrete(breaks = s_axis[seq(1,length(s_axis),length=7)]) +
  geom_point(data=subset(st_grid, s < 5),
             aes(x=as.factor(s),y = Yf),
             colour='black',size=3,shape=4) +
  geom_segment(data=s_obs,aes(x=as.factor(s_obs$s),
                              xend=as.factor(s_obs$s),
                              y = 0, yend = 50),
                 arrow=arrow(length=unit(0.1,"cm"))) +
  geom_line(data=data.frame(s=rep(3.9,2),
                            z=c(-200,1300)),
            aes(as.factor(s),z),linetype="dashed") +
  geom_line(data=data.frame(s=rep(-5.3,2),
                            z=c(-200,1300)),
            aes(as.factor(s),z),linetype="dashed") +
  coord_fixed(ratio = 0.03, ylim=c(0,1050)) +
```

```
print(g)
```

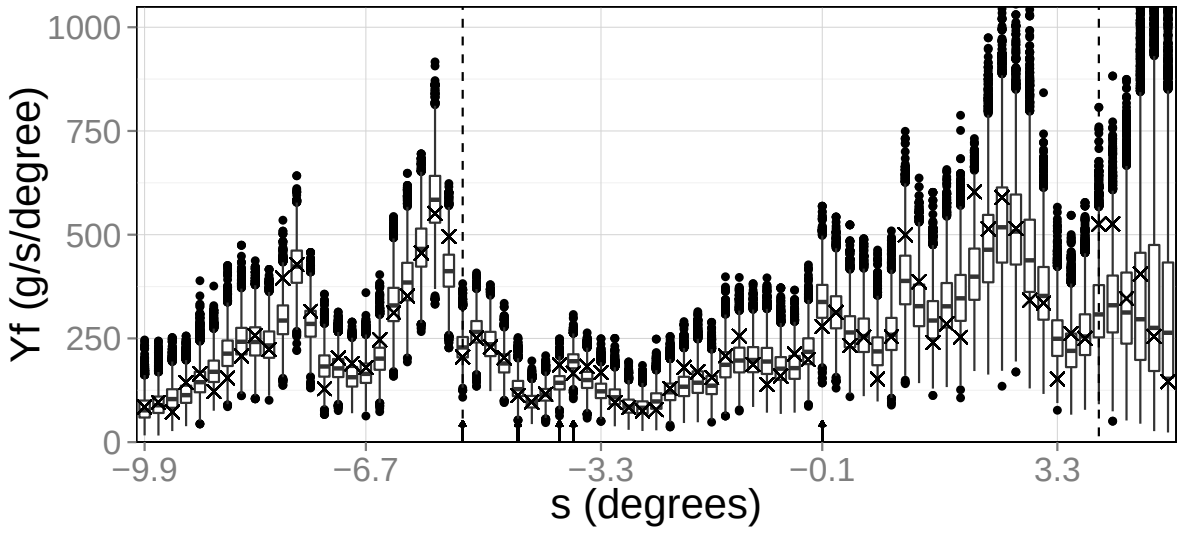

Figure 5: Samples from the posterior distribution of the lognormal flux field obtained using Hamiltonian Monte Carlo (HMC). The boxes denote the interquartile range, the whiskers extend to the last values that are within 1.5 times the interquartile range from the quartiles, and the dots show the samples that lie beyond the end of the whiskers. The crosses denote the true (simulated) fluxes, and the arrows denote the locations of the measurements. The vertical dashed lines show the spatial locations analysed in Figure 6. Since the observed mole fractions are insensitive to flux at $\{s : s > 4.3\}$, these locations were excluded from the model.

```
  ylab("Yf (g/s/degree)") + xlab("s (degrees)")
if(!misspecification & model=="full" & save_images)
  ggsave("../img/Sim1_samples.png",plot = g,width=10)
```

We now compare the Laplace approximation of the flux field to the distribution given by HMC. We first create a function `comp_density` that plots the HMC samples and the Laplace approximation for any arbitrary point, and then pass on two spatial locations to it for plotting:

```
comp_density <- function(j,xu,xl=0,yu=0.01) {
  x <- seq(xl,xu,by=1)
  hist(q[j,1:(N-1)],xlab=c("flux (g/s/degree)"),
       ylab="[Yf | Zm]",
       main="",
       freq=F,
       xlim=c(xl,xu),
       ylim=c(0,yu))
  lines(x,dnorm(x,mean=X$lap_approx$Yf[j],
                sd=sqrt(X$lap_approx$S_ff[j,j])),
        lty=2)
}

if(!misspecification & model=="full" & save_images) {
```

```
  png("../img/density_sim1.png", width=4, height=4, units="in", res=300)
  comp_density(70,800,-200,yu=0.005); dev.off()
  png("../img/density_sim10.png", width=4, height=4, units="in", res=300)
  comp_density(24,400,50,yu=0.014); dev.off()
}
```

```
par(mfrow=c(1,2))
comp_density(70,800,-200,yu=0.005);
comp_density(24,400,50,yu=0.014);
```

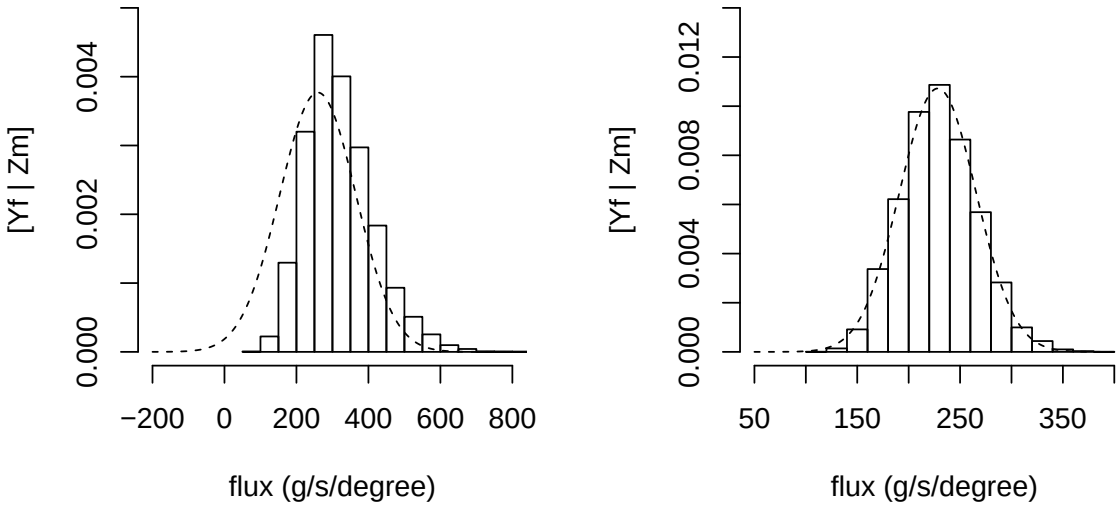

Figure 6: Laplace approximation (dashed line) and a histogram of the empirical posterior distribution from the MCMC samples for the flux at $s = 3.9°$ (left panel) and $s = -5.3°$ (right panel).

Having obtained the flux field samples, we now used the collapsing property of the Gibbs sampler to obtain samples for the mole-fraction field by simply running the samples through the forward model:

```
## Now get the mole fraction samples
mf_samp <- matrix(0,nrow(B),N)                # initialise mole-fraction samples array
mu_post <- matrix(0,nrow(B),N)                # initialise mole-fraction samples array
Q_post <- t(C_m) %*% Qobs %*% C_m + Q_zeta    # conditional precision of Ym
S_post <- chol2inv(chol(Q_post))              # conditional covariance of Ym
L_S_post <- t(chol(S_post))                   # Cholesky decomposition of above
SQB <- S_post %*% Q_zeta %*% B[,X$ind]        # weight given to flux sample
StCQoz <- S_post %*% t(C_m) %*% Qobs %*% s_obs$z # influence of observation
for (i in 1:N){
 mu_post[,i] <- as.vector(SQB %*% q[,i] +     # conditional mean
                          StCQoz)
 mf_samp[,i] <- as.vector(mu_post[,i] +       # generate sample
```

```
                              L_S_post %*% rnorm(nrow(B)))
}
```

Now that we have our mole-fraction samples we can plot the credibility intervals at the unobserved location. We do this by taking the 6th row (recall that $m = 6$) of the mole-fraction samples matrix, putting the rows into a data frame and plotting the result.

```
stat1 <-  seq(6,600,by=6)
mu <- apply(mf_samp[stat1,-c(1:1000,i)],1,median)
uq <- apply(mf_samp[stat1,-c(1:1000,i)],1,quantile,0.75)
lq <- apply(mf_samp[stat1,-c(1:1000,i)],1,quantile,0.25)
uuq <- apply(mf_samp[stat1,-c(1:1000,i)],1,quantile,0.95)
llq <- apply(mf_samp[stat1,-c(1:1000,i)],1,quantile,0.05)
df <- data.frame(mean = mu, uq=uq,lq=lq,uuq=uuq,llq=llq,t=1:length(mu))

g <- LinePlotTheme() +
  geom_ribbon(data=df,aes(x=t,ymax=uq,ymin=lq),alpha=0.6) +
  geom_ribbon(data=df,aes(x=t,ymax=uuq,ymin=llq),alpha=0.3) +
  geom_point(data=subset(s_obs_old,s==new_obs),
             aes(x=t,y = z),colour='black',size=3,shape=4)+
    ylab("Ym (ppb)") + xlab("t (2 h steps)")

if(!misspecification & model=="full" & save_images)
  ggsave("../img/MF_samples.png",plot = g,width=10,height=3)
```

Finally, we calculate the statistics $S_{1,f}$, $S_{2,f}$ and $S^{0.3}_{1,m}$ as defined in the paper:

```
# Flux errors
residual <-(st_grid$Yf[1:75] - apply(q,1,mean))
post_unc <- apply(q,1,var)
print(paste0("S1f = ", sqrt(mean(residual^2))))

## [1] "S1f = 61.6717095444956"

print(paste0("S2f = ", sqrt(mean(residual^2) /mean(post_unc))))

## [1] "S2f = 0.822369509343506"

# MF errors
residual <- (apply(mf_samp[rm_idx,],1,mean) - subset(st_grid,s == new_obs)$Ym)
post_unc <- apply(mf_samp[rm_idx,],1,var)
print(paste0("S1m = ", sqrt(mean(residual^2))))

## [1] "S1m = 32.4878731541322"
```

```
print(g)
```

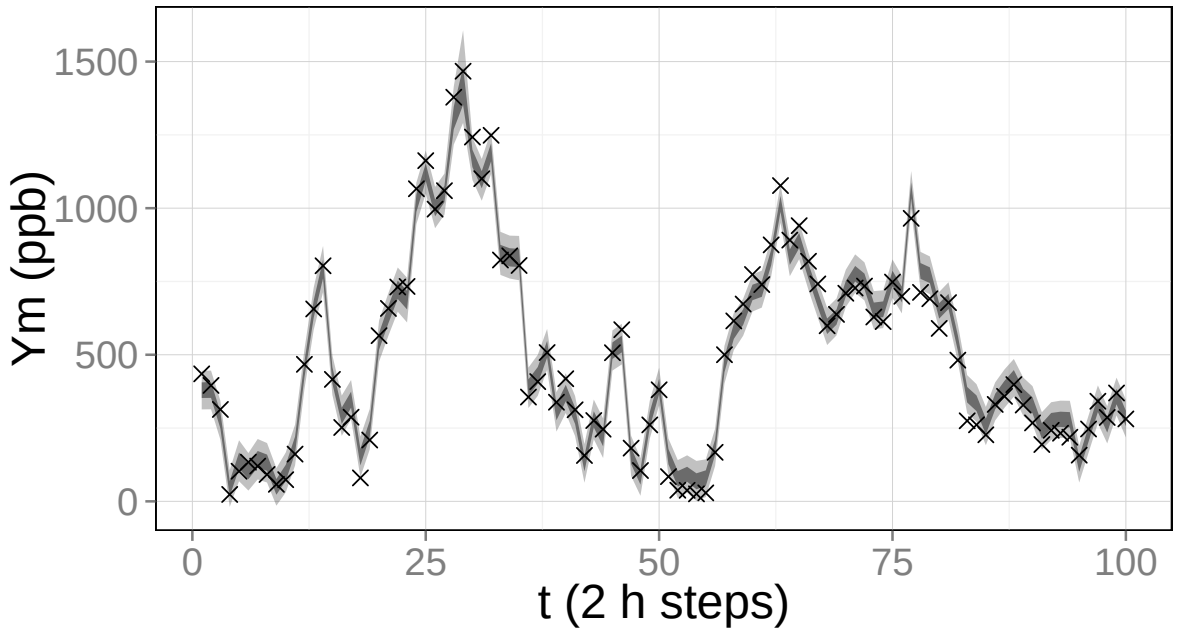

Figure 7: Distributions of mole fraction at $s = 0.3°$ and $t \in \mathcal{T}$, following Gibbs sampling. The dark and light shadings denote the interquartile and the 5–95 percentile ranges, respectively. The crosses denote the true (simulated) mole fractions.