Reproduce_conditional2

XC

27/09/2021

what is \newcommand ?

ref: https://stackoverflow.com/questions/41655383/r-markdown-similar-feature-to-newcommand-in-latex

\newcommand{bI} {I}

Setting up

```
#install.packages("devtools")
#library(devtools)
#install_github("andrewzm/bicon")
```

```
# INLA
#install.packages("INLA",repos=c(getOption("repos"),INLA="https://inla.r-inla-download.org/R/stable"),
library(INLA)
```

```
# For core operation
library(Matrix)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(tidyr)
```

```
##
## Attaching package: 'tidyr'

## The following objects are masked from 'package:Matrix':
##
##     expand, pack, unpack
```

```
# for plotting and for arranging the figures into panels for publication
library(ggplot2)
library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
library(grid)
#install.packages("extrafont")
library(extrafont)
```

```
## Registering fonts with R
```

```
# provides the data
#install.packages("maptools")
library(maptools)
```

```
## Checking rgeos availability: TRUE
## Please note that 'maptools' will be retired by the end of 2023,
## plan transition at your earliest convenience;
## some functionality will be moved to 'sp'.
```

```
#install.packages("mapproj")
library(mapproj)
```

```
## Loading required package: maps
```

```
library(RandomFields)
```

```
## Loading required package: RandomFieldsUtils
```

```
##
## Attaching package: 'RandomFields'
```

```
## The following object is masked from 'package:RandomFieldsUtils':
##
##     RFoptions
```

```
# contains a handy routing for computing CRPSs, crps: Continuous Ranked Probability Score

#install.packages("verification")
library(verification)
```

```
## Loading required package: fields
```

```
## Loading required package: spam

## Loading required package: dotCall64

## Spam version 2.7-0 (2021-06-25) is loaded.
## Type 'help( Spam)' or 'demo( spam)' for a short introduction
## and overview of this package.
## Help for individual functions is also obtained by adding the
## suffix '.spam' to the function name, e.g. 'help( chol.spam)'.

##
## Attaching package: 'spam'

## The following object is masked from 'package:INLA':
##
##     Oral

## The following object is masked from 'package:Matrix':
##
##     det

## The following objects are masked from 'package:base':
##
##     backsolve, forwardsolve

## Loading required package: viridis

## Loading required package: viridisLite

##
## Attaching package: 'viridis'

## The following object is masked from 'package:maps':
##
##     unemp

## See https://github.com/NCAR/Fields for
##  an extensive vignette, other supplements and source code

## Loading required package: boot

## Loading required package: CircStats

## Loading required package: MASS

##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select
```

```
## Loading required package: dtw

## Loading required package: proxy

##
## Attaching package: 'proxy'

## The following object is masked from 'package:spam':
##
##     as.matrix

## The following object is masked from 'package:Matrix':
##
##     as.matrix

## The following objects are masked from 'package:stats':
##
##     as.dist, dist

## The following object is masked from 'package:base':
##
##     as.matrix

## Loaded dtw v1.22-3. See ?dtw for help, citation("dtw") for use in publication.
```

```
# for parallel operations we will be requiring `foreach` and `doParallel`
#install.packages("foreach")
library(foreach)
library(doParallel)
```

```
## Loading required package: iterators
```

Example below consider four models that vary only through the interaction function $b_o(h)$. The models are

$$\begin{array}{ll} \text{Model 1 (independent Matérns):} & b_o(h) \equiv 0, \\ \text{Model 2 (pointwise dependence):} & b_o(h) \equiv A\delta(h), \\ \text{Model 3 (diffused dependence):} & \text{Model 4 with } \Delta = 0 \\ \text{Model 4 (asymmetric dependence):} & b_o(h) \equiv \begin{cases} A\{1-(\|h-\Delta\|/r)^2\}^2, & \|h-\Delta\| \le r \\ 0, & \text{otherwise,} \end{cases} \end{array}$$

where $\Delta = (\Delta_1, \Delta_2)^{\mathrm{T}}$ is a shift-parameter vector that captures asymmetry, r is the aperture parameter, and A is a scaling parameter. In Models 3 and 4, $b_o(h)$ is a shifted bisquare function defined on $\mathbb{R}^2$. The covariance functions $C_{11}(\cdot)$ and $C_{2|1}(\cdot)$ are Matérn covariance functions. For each model we also consider a *reversed* dependence, where we switch Y_2 and Y_1. This gives us a total of eight models to fit and compare.

First, set program options, indicating which parts of the program we want to run and which parts we want to skip

```
### Model choice
model_names <- c("independent","pointwise","moving_average_delta0","moving_average")
# img_path <- "../paper/art"                  ## Where to save the figures
show_figs <- 1                               ## Show the figures in document
print_figs <- 0                              ## Print figures to file (leave =0)
LK_analysis <- 0                             ## Carry out likelihood analysis
LOO_analysis <- 0                            ## Carry out LOO analysis
Shifted_Pars_estimation <- 0                 ## Fit shifted parsimonious Matern
RF_estimation <- 0                           ## Carry out LOO with RFields
useMPI <- 0                                  ## MPI backend available?
```

The data

```
data(weather,package = "RandomFields")
weather <- weather %>% data.frame()
print(head(weather))
```

```
##   pressure temperature      lon    lat
## 1 200.4844  0.60537720 -131.000 46.000
## 2 384.8516 -0.02233887 -124.400 41.900
## 3 156.8984 -0.26644897 -124.500 46.100
## 4 248.4297 -1.30670166 -124.700 47.300
## 5 253.2266  0.14398193 -124.500 44.600
## 6 159.2031 -0.27954102 -124.985 49.907
```

The `weather` table contains four fields, with latitude, longitude, `pressure forecasting errors`, and `temperature forecasting errors` for December 13, 2003 at 4 p.m. in the North American Pacific Northwest.

Since pressure and temperature have different units, we find a scaling factor by taking the ratio of the sample variances of the two variates, and computing its square root. use this factor to scale the pressure variable

```
p_scale <- var(weather$pressure) / var(weather$temperature) %>%
  sqrt() %>%
  as.numeric()
```

A function `form_Z` extract Z_1 and Z_2 and concatenate them into one long vector Z. If model number is $>$ 4, then vectors Z_1 and Z_2 are inverted.

Also define `m1` as the number of observations of Y_1, `m2` as the number of observations of Y_2 and `m` as the total number of observations.

```
form_Z <- function(model_num, scale = T) {
  Z1 <- weather$temperature
  Z2 <- weather$pressure

  if (scale) Z2 <- Z2 / p_scale   # scale press

  if (model_num > 4) {
     temp <- Z1
     Z1 <- Z2
```

```
    Z2 <- temp                    # reverse Z1, Z2
  }
  Z <- rbind(Z1, Z2)
}

m1 <- m2 <- nrow(weather)         # Numb of obs of Y1, Y2
m  <- m1 + m2                     # total numb of obs
I_m1 <- diag(m1)                  # Identity matrix of m1 by m1
```

Process discretisation

approximate the processes as a sum of elemental basis functions (tent functions) constructed on a triangulation.

The triangulation is formed using the mesher in the `INLA` package

provide a tailored function in the package `bicon`, `initFEbasis`, which takes information from the `INLA` mesher and casts it into a `Mesh` object

Importantly, the `Mesh` object also contains information on the areas of the elements in the Voronoi tesselation, which will be used to approximate the integrations.

```
## Constructing mesh
##------------------

mesh <- inla.mesh.2d(loc = weather[c("lon", "lat")],
             cutoff = 0, max.edge = 0.75, offset = 4)  # fine mesh
# Create a triangle mesh based on initial point locations


mesh_locs <- mesh$loc[, 1:2]     # in 2-D, only need lon, lat of the nodes


#str(mesh)
# $ n       : int 2071
# $ loc     : num [1:2071, 1:3] -130 -114 -111 -111 -114 ...
# $ graph   :List of 5
#  ..$ tv : int [1:3982, 1:3] 289 157 162 658 172 1275 336 1613 9 165 ...
#  ..$ vt : int [1:2071, 1] 2619 1734 3559 3375 3629 1738 3413 2356 2165 202 ...
#  ..$ tt : int [1:3982, 1:3] 2967 3869 461 3041 69 2399 3314 208 2705 22 ...
#  ..$ tti: int [1:3982, 1:3] 1 1 1 1 2 2 1 1 1 2 ...
#  ..$ vv :Formal class 'dgTMatrix' [package "Matrix"] with 6 slots
#  .. .. ..@ i       : int [1:12104] 1208 1238 1378 911 1046 1117 847 848 1943 1073 ...
#  .. .. ..@ j       : int [1:12104] 0 0 0 1 1 1 2 2 2 3 ...
#  .. .. ..@ Dim     : int [1:2] 2071 2071
#  .. .. ..@ x       : num [1:12104] 1 1 1 1 1 1 1 1 1 1 ...
```

```
head(mesh_locs)
```

```
##           [,1]     [,2]
## [1,] -129.5832 36.82639
## [2,] -114.1306 36.82639
## [3,] -110.8833 40.07370
```

```
## [4,] -110.8833 52.55684
## [5,] -113.9965 55.67000
## [6,] -130.7869 55.67000
```

```
str(mesh_locs)
```

```
##  num [1:2071, 1:2] -130 -114 -111 -111 -114 ...
```

```
# num [1:2071, 1:2]
```

```
head(as.matrix(mesh_locs))
```

```
##           [,1]     [,2]
## [1,] -129.5832 36.82639
## [2,] -114.1306 36.82639
## [3,] -110.8833 40.07370
## [4,] -110.8833 52.55684
## [5,] -113.9965 55.67000
## [6,] -130.7869 55.67000
```

```
str(as.matrix(mesh_locs))  # num [1:2071, 1:2]
```

```
##  num [1:2071, 1:2] -130 -114 -111 -111 -114 ...
```

```
## Compute distances as in Gneiting (2010)
# -- great circle distances
##--------------------------------------------

# str(RFearth2dist(coord = as.matrix(mesh_locs)))
# calculates distances, cf. dist, assuming that the earth is an ellipsoid

# Angle mode switches to 'degree'.
#  'dist' num [1:2143485] 1373 1662 2264 2387 2087 ...

D <- as.matrix(RFearth2dist(coord = as.matrix(mesh_locs)))
```

```
## Angle mode switches to 'degree'.
```

```
str(D)  # num [1:2071, 1:2071] 0 1373 1662 2264 2387
```

```
##  num [1:2071, 1:2071] 0 1373 1662 2264 2387 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:2071] "1" "2" "3" "4" ...
##   ..$ : chr [1:2071] "1" "2" "3" "4" ...
```

```
str(c(D))  #  # 2071 * 2071 = 4289041
```

```
##  num [1:4289041] 0 1373 1662 2264 2387 ...
```

```
# num [1:4289041] 0 1373 1662 2264 2387 ...
Dvec <- as.double(c(D))
```

```
Dobs <- as.matrix(RFearth2dist(coord = as.matrix(weather[c("lon", "lat")])))
```

```
## Angle mode switches to 'degree'.
```

```
str(Dobs) # num [1:157, 1:157] 0 697 502 502 532 ..
```

```
##  num [1:157, 1:157] 0 697 502 502 532 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:157] "1" "2" "3" "4" ...
##   ..$ : chr [1:157] "1" "2" "3" "4" ...
```

```
Dobs_vec <- as.double(c(Dobs))
```

Below is optional

```
## Cast into custom Mesh object (optional)
##-----------------------------

#install.packages("initFEbasis")
#library(initFEbasis)
#Mesh <- initFEbasis(p = mesh_locs,         # n * 2 matrix of vertex locations
#            t = mesh$graph$tv,     # m * 3 matix of triangulation; each row identify which row of p to
#            K = mesh$graph$vv     # connection matrix
#)
```

?initFEbasis: initialise a finite element basis which initialises an object of class FEBasis which defines a set of elemental 'tent' basis functions over a pre-specified triangulation in 2-D

```
#str(Mesh)
#$ vars:'data.frame':   2071 obs. of  4 variables:
#  .. .. ..$ x        : num [1:2071] -130 -114 -111 -111 -114 ...
#  .. .. ..$ y        : num [1:2071] 36.8 36.8 40.1 52.6 55.7 ...
#  .. .. ..$ n        : int [1:2071] 1 2 3 4 5 6 7 8 9 10 ...
#  .. .. ..$ area_tess: num [1:2071] 0.112 0.137 0.146 0.128 0.111 ...
```

Alternative to Mesh obj of FEbasis package, we could use `deldir` package to 1. construct Voronoi tesselation from INLA mesh points; 2. from the output summary, extract the tessaltion area directly.

```
#install.packages("deldir")
library(deldir)
```

```
## deldir 0.2-10      Nickname: "Morpheus and Euripides"
```

```
##
##      Note 1: As of version 0.2-1, error handling in this
##      package was amended to conform to the usual R protocol.
##      The deldir() function now actually throws an error
```

```
##      when one occurs, rather than displaying an error number
##      and returning a NULL.
##
##      Note 2:  As of version 0.1-29 the arguments "col"
##      and "lty" of plot.deldir() had their names changed to
##      "cmpnt_col" and "cmpnt_lty" respectively basically
##      to allow "col" and and "lty" to be passed as "..."
##      arguments.
##
##      Note 3: As of version 0.1-29 the "plotit" argument
##      of deldir() was changed to (simply) "plot".
##
##      See the help for deldir() and plot.deldir().
```

```
Voroni <- deldir(x = mesh_locs[, 1],
       y = mesh_locs[, 2],
       rw = c(min(mesh_locs[, 1]) - 0.00001,
              max(mesh_locs[, 1]) + 0.00001,
              min(mesh_locs[, 2]) - 0.00001,
              max(mesh_locs[, 2]) + 0.00001))

str(Voroni)
```

```
## List of 9
##  $ delsgs  :'data.frame':    6052 obs. of  6 variables:
##   ..$ x1  : num [1:6052] -132 -128 -128 -128 -128 ...
##   ..$ y1  : num [1:6052] 39.2 37.3 38 38 38 ...
##   ..$ x2  : num [1:6052] -132 -129 -129 -129 -128 ...
##   ..$ y2  : num [1:6052] 38.9 36.8 38.6 37.7 37.3 ...
##   ..$ ind1: int [1:6052] 1375 803 840 840 840 932 932 932 971 995 ...
##   ..$ ind2: int [1:6052] 522 580 440 581 803 440 650 781 440 512 ...
##  $ dirsgs  :'data.frame':    6052 obs. of  10 variables:
##   ..$ x1     : num [1:6052] -134 -128 -128 -129 -128 ...
##   ..$ y1     : num [1:6052] 36.8 37.1 38.3 37.9 37.6 ...
##   ..$ x2     : num [1:6052] -132 -128 -129 -129 -128 ...
##   ..$ y2     : num [1:6052] 39.1 37.1 38.3 37.7 37.7 ...
##   ..$ ind1   : int [1:6052] 1375 803 840 840 840 932 932 932 971 995 ...
##   ..$ ind2   : int [1:6052] 522 580 440 581 803 440 650 781 440 512 ...
##   ..$ bp1    : logi [1:6052] TRUE FALSE FALSE FALSE FALSE FALSE ...
##   ..$ bp2    : logi [1:6052] FALSE FALSE FALSE FALSE FALSE FALSE ...
##   ..$ thirdv1: num [1:6052] -1 1208 1092 1404 1208 ...
##   ..$ thirdv2: num [1:6052] 1275 1357 1404 1208 582 ...
##  $ summary :'data.frame':    2071 obs. of  9 variables:
##   ..$ x       : num [1:2071] -130 -114 -111 -111 -114 ...
##   ..$ y       : num [1:2071] 36.8 36.8 40.1 52.6 55.7 ...
##   ..$ n.tri   : num [1:2071] 2 2 2 2 2 3 2 2 6 6 ...
##   ..$ del.area: num [1:2071] 0.0767 0.0876 0.1005 0.0853 0.0684 ...
##   ..$ del.wts : num [1:2071] 0.000182 0.000208 0.000239 0.000203 0.000162 0.000363 0.000237 0.000196
##   ..$ n.tside : num [1:2071] 3 3 3 3 3 4 3 3 6 6 ...
##   ..$ nbpt    : num [1:2071] 2 2 2 2 2 2 2 2 0 0 ...
##   ..$ dir.area: num [1:2071] 0.112 0.137 0.146 0.128 0.111 ...
##   ..$ dir.wts : num [1:2071] 0.000246 0.000302 0.00032 0.000282 0.000244 0.000446 0.000264 0.000262 0
##  $ n.data  : int 2071
##  $ n.dum   : num 0
```

```
##  $ del.area: num 421
##  $ dir.area: num 454
##  $ rw      : num [1:4] -135 -110.9 36.8 55.7
##  $ ind.orig: int [1:2071] 1 2 3 4 5 6 7 8 9 10 ...
##  - attr(*, "round")= logi TRUE
##  - attr(*, "digits")= num 6
##  - attr(*, "class")= chr "deldir"
```

```
Area.tess <- Voroni$summary$dir.area
```

Next establish the dimension of our grids. Since we will be evaluating Y_1 and Y_2 on the same grid, `n1` = `n2`

```
## Mesh Size
##-----------

n2 <- n1 <- nrow(mesh_locs)
n = n1 + n2
```

Will approximate the integration using the rectangular rule. When using finite elements, this reduces to using the areas of the Voronoi tessellation as integration weights.

We first compute the vector of displacements h which will be of length (`n2` $\times$ `n1`).

Then, with each element we associate an integration weight equal to the area of the Voronoi tessellation of the element.

```
str(mesh_locs)
```

```
##  num [1:2071, 1:2] -130 -114 -111 -111 -114 ...
```

```
str(mesh_locs[1, ])   # vector
```

```
##  num [1:2] -129.6 36.8
```

```
str(t(mesh_locs) - mesh_locs[1, ])
```

```
##  num [1:2, 1:2071] 0.00 0.00 1.55e+01 2.84e-14 1.87e+01 ...
```

```
# num [1:2, 1:2071] 0.00 0.00 1.55e+01

head(t(t(mesh_locs) - mesh_locs[1, ]))
```

```
##           [,1]         [,2]
## [1,]  0.000000 0.000000e+00
## [2,] 15.452598 2.842171e-14
## [3,] 18.699911 3.247313e+00
## [4,] 18.699911 1.573045e+01
## [5,] 15.586753 1.884361e+01
## [6,] -1.203609 1.884361e+01
```

```r
# Mesh Integration points
#-------------------------

h <- matrix(0, n1 * n2, ncol = 2)
Area <- rep(0, n1 * n2)
for (i in 1:n1) {
  h[((i - 1) * n1 + 1) : (i * n1), ] <- t(t(mesh_locs) - mesh_locs[i])
  Area[((i - 1) * n1 + 1) : (i * n1) ] <- Area.tess

}

str(h) # num [1:4289041, 1:2]
```

```
##  num [1:4289041, 1:2] 0 15.5 18.7 18.7 15.6 ...
```

```r
h1_double <- as.double(h[, 1])
h2_double <- as.double(h[, 2])
str(h1_double)
```

```
##  num [1:4289041] 0 15.5 18.7 18.7 15.6 ...
```

Organizing the Observations

Map process to observations by constructing an incidence matrix, which is `1` if obs conincides with a vertex of triangualtion mesh, and `0` otherwise.

Dimension of the whole incidence matrix is $(\texttt{m1} + \texttt{m2}) \times (\texttt{n1} + \texttt{n2})$, where `m1 <- m2<- nrow(weather)`

Since in this problem we have co-located observations, we find the incidence matrix for one of the observations, Z_1, and then form the whole incidence matrix by simply constructing a block diagonal matrix (using `bdiag`).

We use `left_join` to find the tri grid vertex with which obs location coincide, which returns `NA` if no coincides with vertex.

```r
mesh_locs <- data.frame(lon = mesh_locs[, 1], lat = mesh_locs[, 2])
#head(left_join(x = mesh_locs, y = weather))
idx <- which(!is.na(left_join(x = mesh_locs, y = weather)$temperature))
```

```
## Joining, by = c("lon", "lat")
```

```r
str(idx)  # int [1:157] 9 10 11 12 13 14 15 16 17 18 ...
```

```
##  int [1:157] 9 10 11 12 13 14 15 16 17 18 ...
```

```r
C1 <- sparseMatrix(i = 1:m1, j = idx, x = 1, dims = c(m1, n1)) # incidence matrix
C <- bdiag(C1, C1)  # total incidence matrix

str(C)
```

```
## Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
##   ..@ i       : int [1:314] 0 1 2 3 4 5 6 7 8 9 ...
##   ..@ p       : int [1:4143] 0 0 0 0 0 0 0 0 0 1 ...
##   ..@ Dim     : int [1:2] 314 4142
##   ..@ Dimnames:List of 2
##   .. ..$ : NULL
##   .. ..$ : NULL
##   ..@ x       : num [1:314] 1 1 1 1 1 1 1 1 1 1 ...
##   ..@ factors : list()
```

Maximum likelihood estimation

Since the optimisation algorithm requires a parameter vector of the same length (irrespective of the model number)

we first define a function `append_theta` that takes the parameter vector associated with the model in question and appends it so it is of the required size (in this case of length 12).

M1(M5): append 4 0's, M2(M6): append 3 0's, M3(M7): append 2 0's;

```
append_theta <- function(theta, model_num) {
  if (model_num %in% c(1, 5)) {  # M1 and M5
    theta <- c(theta, rep(0, 4))
    #theta[10] <- 0.001
  } else if (model_num %in% c(2, 6)) {
    theta <- c(theta, rep(0, 3))
    theta[10] <- 0.001
  } else if (model_num %in% c(3, 7)) {
    theta <- c(theta, rep(0, 2))
  }
  theta
}
```

Next, we require a function that, given the parameter vector `theta` and the model number `model_num`, returns the required matrices and vectors used in fitting. These are the matrices

$$\texttt{SY} = \begin{bmatrix} \Sigma_{11} & \Sigma_{11}B^{\mathrm{T}} \\ B\Sigma_{11} & \Sigma_{2|1} + B\Sigma_{11}B^{\mathrm{T}} \end{bmatrix}, \qquad \texttt{So} = \begin{bmatrix} \tau_1^2 I_m & 0 \\ 0 & \tau_2^2 I_m \end{bmatrix}. \tag{1}$$

We then add these two together to obtain the matrix $\mathrm{cov}((Y_1^{\mathrm{T}}, Y_2^{\mathrm{T}})^{\mathrm{T}})$ which, recall that for this example is identical to $\mathrm{cov}((Z_1^{\mathrm{T}}, Z_2^{\mathrm{T}})^{\mathrm{T}})$ since the data is equal to the process at the observed locations. If `whole_mesh` is `TRUE`, then the process covariance matrix is evaluated over the entire mesh (used for cokriging at unobserved locations).