Introducing the package MVST

RATES

June 4, 2014

Abstract

MVST is an *R Software* package which facilitates the construction of Gaussian multi-variate spatio-temporal fields. These fields can be a-priori independent or co-regionalised. The package also allows the variates to be spatial-only, in which case they are assumed to be repeatedly observed in time. MVST is designed so that spatio-temporal fields can be decomposed using arbitrary basis functions. To date only finite elements are implemented, but in principle the package allows for any arbitrary decomposition, which can be different for each field.

In this first vignette, we show the basic functionality of the package by carrying out a simplified mass-balance study of the Antarctic ice sheet using satellite altimetry. Further functionality will be exemplified in a second vignette.

1 Introduction

In this introductory vignette we will adopt a very simple approach to modelling the Antarctic ice sheet. We will consider a single spatio-temporal data set, that from the Ice, Cloud and Elevation Satellite (ICESat), and assume that it measures only a subset of all processes. We also model the system of processes being observed on a rather coarser mesh in order to reduce computational times. However, the example is sufficiently general to indicate possible extensions, and expose limitations of the approach.
Note: You need at least 8GB of random-access memory to run the code in this vignette.

2 Problem setup

For this example we will need the packages `Matrix`, `devtools`, `ggplot2`, `MVST` and `linalg`. The last two packages are available from the github page `http://github.com/andrwzm` and can be installed using the command `install_git` from the package `devtools`. Since `MVST` is still under development we assume that we have it on disk somewhere and can load it accordingly using the `load_all` command in `devtools`:

```
library(Matrix)
library(devtools)
library(ggplot2)

##
## Attaching package:  'ggplot2'
##
## The following objects are masked from 'package:MVST':
##
##     scale_colour_brewer, scale_fill_brewer
```

```
library(linalg)
load_all("..")

## Loading MVST
```

The package MVST provides three datasets for this example.

- `icesat`: A sample of the entire ICESat data set, which denote height changes in m yr^{-1} at certain locations in space and time.
- `surf_fe`: The triangulation on which we will be modelling the ice sheet processes.
- `shapefiles`: Some polygons such as the Antarctic grounding line and coastline which will be used for plotting purposes.

```
data(icesat)
data(surf_fe)
data(shapefiles)
```

A crucial variable in the program is `t_axis`. This denotes the time horizon over which the processes will be modelled. It is important that all data contain a column `t` which has elements within the array `t_axis`. In this example we will take the year 2003 to be our starting point $t = 0$ and assume that each unit is a yearly interval. Thus for the period 2003–2009 we set

```
t_axis=0:6
```

If we have data sets extending further back in time we can set `t_axis` to be negative too. Now that we have the data and libraries loaded we can proceed to construct the meshes and assemble data into objects for use with the MVST package. First, however, we outline the use of a useful function which helps to minimise analysis time, `md5_cache`.

`md5_cache` returns a function which caches the results of a program function evaluation, so that it can be quickly loaded on re-run (this is extensivelly used in this vignette). In this vignette we will cache temporary results in the `cache` directory located in our home folder by initialising

```
md5_wrapper <- md5_cache("~/cache/")
```

`md5_wrapper` is itself a function which we shall use to cache the results from function evaluations, and takes the function name as the first argument. So, `md5_wrapper(fn,a,b)` will evaluate (and cache) results from `fn` with arguments `a` and `b`. Examples will be seen in the remainder of the text.

2.1 Meshes

All processes, are assumed to be accurately constructed on a triangulation. We provide standard triangulations for the Antarctic ice sheet in the package. However it is still useful to note how these are constructed.

To define a triangulation we need at least two items, a two-column matrix of vertices `p` and a 3 column matrix `t` in which each row identifies the nodes composing a single triangle. The triangulations construct what are known as 'tent functions', which are the basis function we employ. From these tent functions two matrices can be found, known as the *mass matrix* `M` and the *stiffness matrix* `K`. These matrices, constructed using inner products and derivatives, are essential for defining the smoothness properties of the field. Standard routines exist to construct these matrices from `p` and `t`.

A `Mesh` object is set up by calling the function `initFEbasis` using the appropriate arguments:

```
Mesh <- md5_wrapper(initFEbasis,p=surf_fe$p,
                                t=surf_fe$t,
                                M=surf_fe$M,
                                K=surf_fe$K)
```

The object `Mesh` will be used for several things, not least for plotting the results following inference.

An elegant feature of the package is the ease with which we can attribute properties to each node in the mesh. For instance, we will want to know whether a node is within the grounding line or not, or whether it lies in an island or within the mainland. We can do this by using the `attribute_polygon` function. In the following we take the locations `x,y` of each node in the mesh using `Mesh[c("x","y")]` and attribute to it an island number (using `shapefiles$Islands`), a flag indicating whether it is within the grounding line (using `shapefiles$grounding_sub`) and another flag indicating whether it is within the coastline (using `shapefiles$coast_sub`). Note that for this problem, the coastline includes floating ice shelves and thus encloses the grounding line.

```
Mesh["island"] <- md5_wrapper(attribute_polygon,Mesh[c("x","y")],shapefiles$Islands)
Mesh["in_land"] <- md5_wrapper(attribute_polygon,Mesh[c("x","y")],shapefiles$grounding_sub) |
                                                 Mesh["island"]
Mesh["in_coast"] <-md5_wrapper(attribute_polygon,Mesh[c("x","y")],shapefiles$coast_sub) |
                                                 Mesh["island"]
```

We can also attribute a 'mass per unit height change' field to each node. This is facilitated by the field `area_tess_km2` of `Mesh` which contains the area of the Voronoi tessellation associated with the triangulation. In general, the mass attributed to each element will vary from process to process. In this example, we will simply assume that the mass change of a single Voronoi cell occurs at a density of 500 kg m^{-3} so that

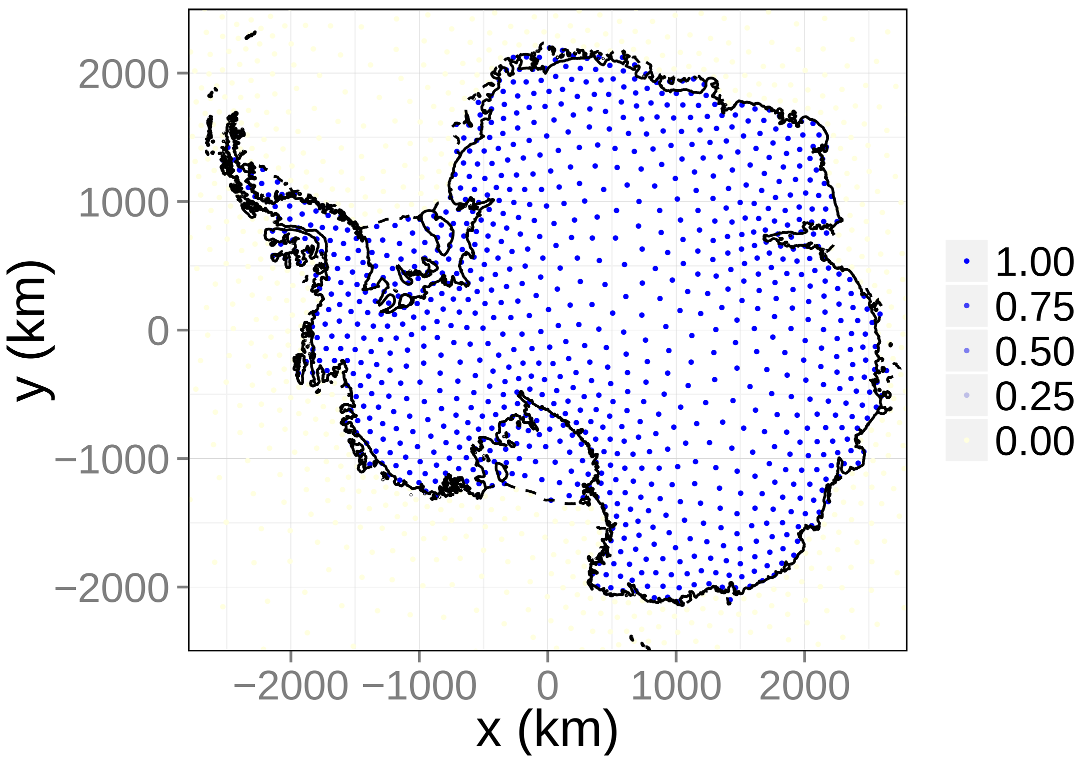
```
Mesh["mass_GT_per_year"] <- Mesh["area_tess_km2"] * 0.5/1000
```

We can plot any variable associated with the mesh, both for checking and visualisation purposes. First we define a function which will be used for plotting Antarctica's coastline and grounding line and nearby islands using `ggplot2`. The function takes a `ggplot` object and overlays it with the required polygons:

```
 PlotAntarctica <- function(g,shapefiles) {
   AIS_x <- c(-2800,2800)
   AIS_y <- c(-2500,2500)
   g <- g +
     geom_path(data = shapefiles$grounding_sub,aes(x,y),colour="black") +
     geom_path(data= shapefiles$coast_sub,aes(x,y),linetype=2,size=0.5) +
     geom_path(data=shapefiles$Islands,aes(x,y,group=id))  + xlab("x (km)") + ylab("y (km)") +
     coord_fixed(xlim=AIS_x,ylim=AIS_y)
 }
```

To plot the nodes which are within the coastline, we can then simply type:

```
g <- plot(Mesh,"in_coast",size=1)
print(PlotAntarctica(g,shapefiles))
```



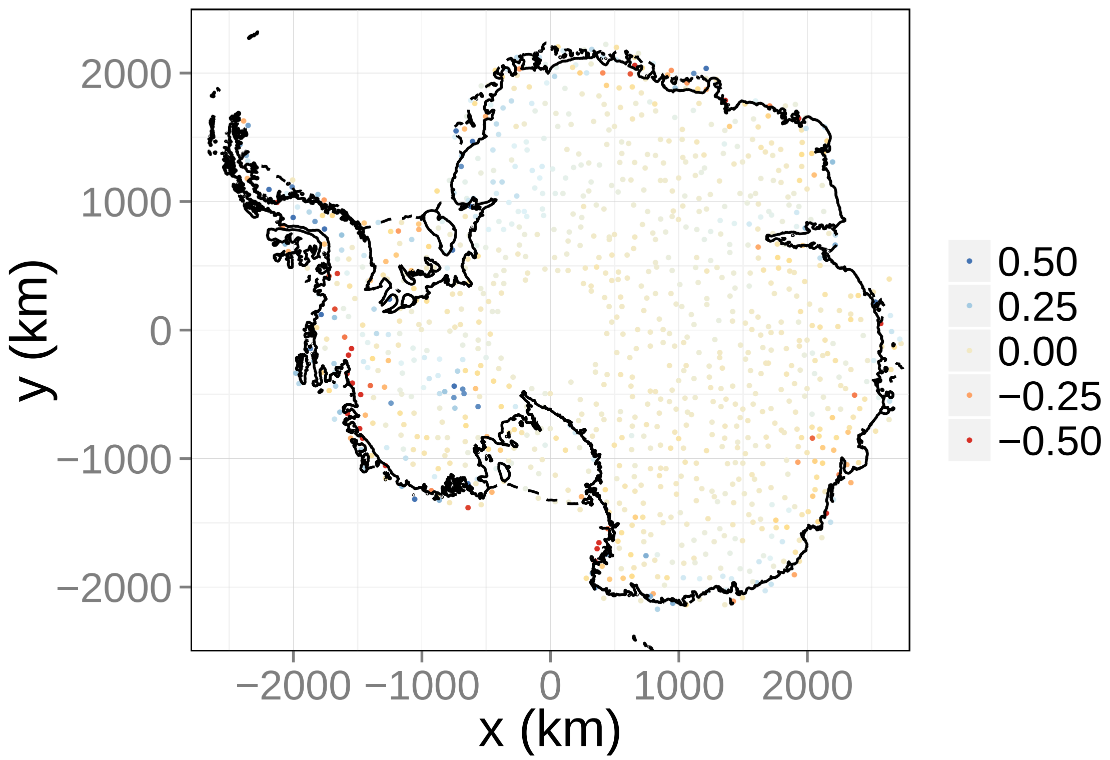
2.2 Observations

MVST provides a constructor for observations which takes care of most pre-processing typical in these types of problems. In the following we take the raw ICESat data, throw away data which is greater than 5 in amplitude, construct a grid of 100 × 100 km, and take the processed data to be the median in every grid box (the error will be the median absolute deviation of readings in each box). We also attribute the object with a name `icesat`. For a list of further arguements which can be suppled, plrease refer to the manual.

```
icesat_obs <- Obs(df=icesat,
                  abs_lim = 5,
                  avr_method = "median",
                  box_size=100,
                  name="icesat")
```

Once we have constructed an observation object we can easily plot it. In the following, we plot the data in 2006, and set saturation limits at -0.5 and 0.5.

```
g <- plot(subset(icesat_obs,t==3),"z",pt_size=1,max=0.5,min=-0.5)
print(PlotAntarctica(g,shapefiles))
```



Other fields related to the observations can be plotted in the same way. Note that we can also attribute certain characteristics to the observations using the `attribute_polygon` command (see above) and the `attribute_data` command, see manual.

Pseudo-observations

In the Antarctic ice sheet we would like to constrain the field to be zero over the ocean. For this reason we can create a pseudo-data set in which the data record exactly zero on the mesh vertices which lie outside the continent's coastline. This can be implemented as follows. First, create a data frame with `x,y` coordinates which coincide with the required vertices

```
pseudo_single_frame <- subset(Mesh,in_coast==F,select=c(x,y))
pseudo_single_frame$z <- 0
pseudo_single_frame$std <- 1e-6
```

and then extend it to repeat for each time point

```
pseudo_df <- plyr::adply(t_axis,1,function(x) {
                            return(cbind(pseudo_single_frame,data.frame(t=x)))
                            })
```

Once we have the required data frame we can assemble it into an `MVST` object as above (this time without any pre-processing)

```
pseudo <- Obs(df=pseudo_df,name="pseudo")
```

2.3 The spatio-temporal process

In this section we describe how we can construct a spatio-temporal model over the defined mesh. The model we will consider is of the form

$$\boldsymbol{x}_{t+1} = \boldsymbol{\mu}_t + \boldsymbol{A}_t\boldsymbol{x}_t + \boldsymbol{D}\boldsymbol{\beta} + \boldsymbol{e}_t \tag{1}$$

where $\boldsymbol{x}_t$ is the quantity we are interested in, $\boldsymbol{\mu}_t$ is a temporally evolving known mean, the matrix $\boldsymbol{A}_t$ describes how $\boldsymbol{x}_t$ evolves from on time point to the next, $\boldsymbol{D}\boldsymbol{\beta}$ are some temporally-invariant effects and $\boldsymbol{e}_t$ is the added disturbance (random) at each time point. Note that since $\boldsymbol{e}_t$ is random, so is $\boldsymbol{x}_t$, and our uncertainty in this system is characterised by incomplete knowledge of the initial condition ($\boldsymbol{x}_0$), weights on the fixed effects ($\boldsymbol{\beta}$) and the disturbance $\boldsymbol{e}_t$. The task is thus to estimate $\boldsymbol{x}_t$ at each time point and, possibly, $\boldsymbol{\beta}$. First, however, we need to define all the other quantities, which are temporally evolving functions.

We shall assume that $\boldsymbol{\mu}_t$ is zero everywhere. It is important that this function returns, at each time point, a matrix of the appropriate size (in this case the number of vertices):

```
mu <- function(k) { # time-varying matrix for mu
  return(matrix(0,nrow(Mesh),1))
}
```

For $\boldsymbol{A}$, we shall assume that the process at each vertex depends only on the process at the same location at the previous time point, so that $\boldsymbol{A}$ is diagonal. In this case we shall take the diagonal elements to be 0.2, indicating poor association across time points. This is not true for the system under study, since changes due to ice dynamics are temporally smooth, however this suffices for this simple example:

```
A <- function(k) {  # time-varying matrix for A
  0.2*Imat(nrow(Mesh))
 }
```

Again, it is important that the function returns a matrix of the appropriate size.

The difficult term to define in this problem is $\boldsymbol{e}_t$. In principle, $\boldsymbol{e}_t$ is not spatially uncorrelated and is, in itself, a spatial random field. Defining a spatial random field on a triangulation is facilitated using the stochastic partial differential approach of [1]. Details of this approach are beyond the scope of this vignette, however it suffices to say that we can define the approximate smoothness properties `nu`, the squared inverse expected amplitude `desired_prec` and the practical range `l` of $\boldsymbol{e}_t$ on the triangulation through its precision matrix $\boldsymbol{Q}$. In the following we define a field on the `Mesh` which has smoothness 1, an expected amplitude of 1 (note that I use expected amplitude loosely to describe the square-root marginal standard deviation of the field) and a practical range of 1200km:

```
Q <- function(k) {  # time-varying matrix for Q
   Prec_from_SPDE_wrapper(M = mass_matrix(Mesh),
                          K = stiffness_matrix(Mesh),
                          nu = 1,
                          desired_prec = 1,
                          l = 1200)
}
```

Note that `desired_prec` = '1/(expected amplitude)^2'. Both `desired_prec` and `l` can be supplied as vectors, in which case the properties of $\boldsymbol{e}_t$ also vary in space. This will not be considered in this vignette.

Finally, now that we have the required matrices we need to assemble them into a variable auto-regressive model object. This can be done using the command `VAR_Gauss`. Note how the time axis is also supplied to the object; this defines the size of the auto-regressive object which will be generated. Not that we can define time points beyond or previous to where we have data for hindcasting and forecasting.

```
SURF_VAR <- VAR_Gauss( mu = mu,A=A, Qw = Q,t_axis = t_axis,name="SURF")
```

Once we have the auto-regressive model set up, all that remains is to 'attach' it to the basis functions we are using. In this case our basis functions constitute a set of tent functions, however these could have been any functions (which need not be complete or orthonormal):

```
SURF <- GMRF_basis(G = SURF_VAR, Basis = Mesh)
```

The `SURF` object is the quantity of interest here. This contains all the information we need to carry out inference with. In general we can set many similar objects and link them together through the observations. In this example we only have two observation data sets so the graph is actually quite simple:

$$\begin{tikzcd} & \texttt{pseudo} \\ \texttt{SURF} \arrow[ur] \arrow[dr] & \\ & \texttt{icesat\_obs} \end{tikzcd}$$

since we assume that both the pseudo-observations and `ICESat` are recording the process of interest, in this case `SURF`. Note the directionality of the arrow; this indicates the direction of causality, it is the process which is generating the observations and not the other way round.

We implement these links as follows:

```
L1 <- link(SURF,icesat_obs)
L2 <- link(SURF,pseudo)
```

where the main function of the `link` function is to find the matrix $\boldsymbol{C}$ in the observation equations

$$\boldsymbol{z}_t = \boldsymbol{C}_t \boldsymbol{x}_t + \boldsymbol{e}_t \tag{2}$$

since this is generally quite involved. In this equation, $\boldsymbol{z}_t$ is the observation at time t and $\boldsymbol{e}_t$ is the observation standard deviation. The `link` function allows for both point-wise observations (such as ICESat or GPS) and observations with a large spatial footprint defined using `Obs_poly()` (explored in the next vignette).

Once we have the links we can create a graph object, which is simply a set of link and vertices as follows:

```
e <- link_list(list(L1,L2))
v <- block_list(list(G1=SURF,O1=icesat_obs,O2 = pseudo))
G <- Graph(e=e,v=v)
```

The object `G` can be rather complex, particularly when we consider multiple observations and processes. For this reason we compress it into a two-node network, with all the processes concatenated together as one batch, and all the observations grouped together. This is done using the following command

```
G_reduced <- compress(G)

## Warning:  Keeping only common columns in data sets when compressing
```

Note the warning: This highlights the fact that there were attributes in `icesat_obs` such as `in_land`, which were not present in the description of `pseudo`. In this case only the common fields are kept.

`G_reduced` consists of two (enormous) vertices and just one link. This is a graph which is very simple to deal with, and we shall use this to carry out inference on the unknowns appearing in our spatio-temporal model.

3 Inference

The equations for estimating $\boldsymbol{x}_t$ and $\boldsymbol{\beta}_t$ in this simple case are relatively straightforward, and we provide the function `Infer` for this. In most cases, `Infer` is sufficient, provided that reasonable values are set for all unknowns appearing above. If this is not the case, and other parameters also need to be estimated, we need to carry out some further work; this is beyond the scope of this vignette.

3.1 Marginal inference on all vertices

If we wish to carry out inference on all vertices (warning: this might be quite computationally intensive), then we can simply call `Infer` as follows:

```
 Results <- Infer(G_reduced)

## Doing Cholesky and Takahashi
```

`Results` is a list with at least two elements. The first, `Graph`, is a copy of the graph used for inference (in our case `G_reduced`. The second `Post_GMRF` contains details on the expectation and the uncertainty of our estimates in fields `x_mean` and `x_margvar` respectively.

As shown above, `MVST` provides several useful functions for simply plotting the results following an update. Here we will show how we can plot the expectation of $\boldsymbol{x}_t$ for some t, which is stippled when the absolute expectation is greater than the standard deviation of the estimate ('significant'). First, we assign the time point we wish to plot (in this case $t = 2$) to a field `to_plot`:

```
Mesh["to_plot"] <- c(subset(Results$Post_GMRF,t==2,select=x_mean))
```

and then create another field `marg_std`, which will be used for stippling:

```
 Mesh['marg_std'] <- sqrt(subset(Results$Post_GMRF,t==2,select=x_margvar))
```

Next, we can make use of the function `plot_interp` to produce an interpolated map of the field's expectation. Here we pass as arguments the number of grid points to use on each axis `ds`, colour thresholds `min, max` and the legent title through `leg_title`.

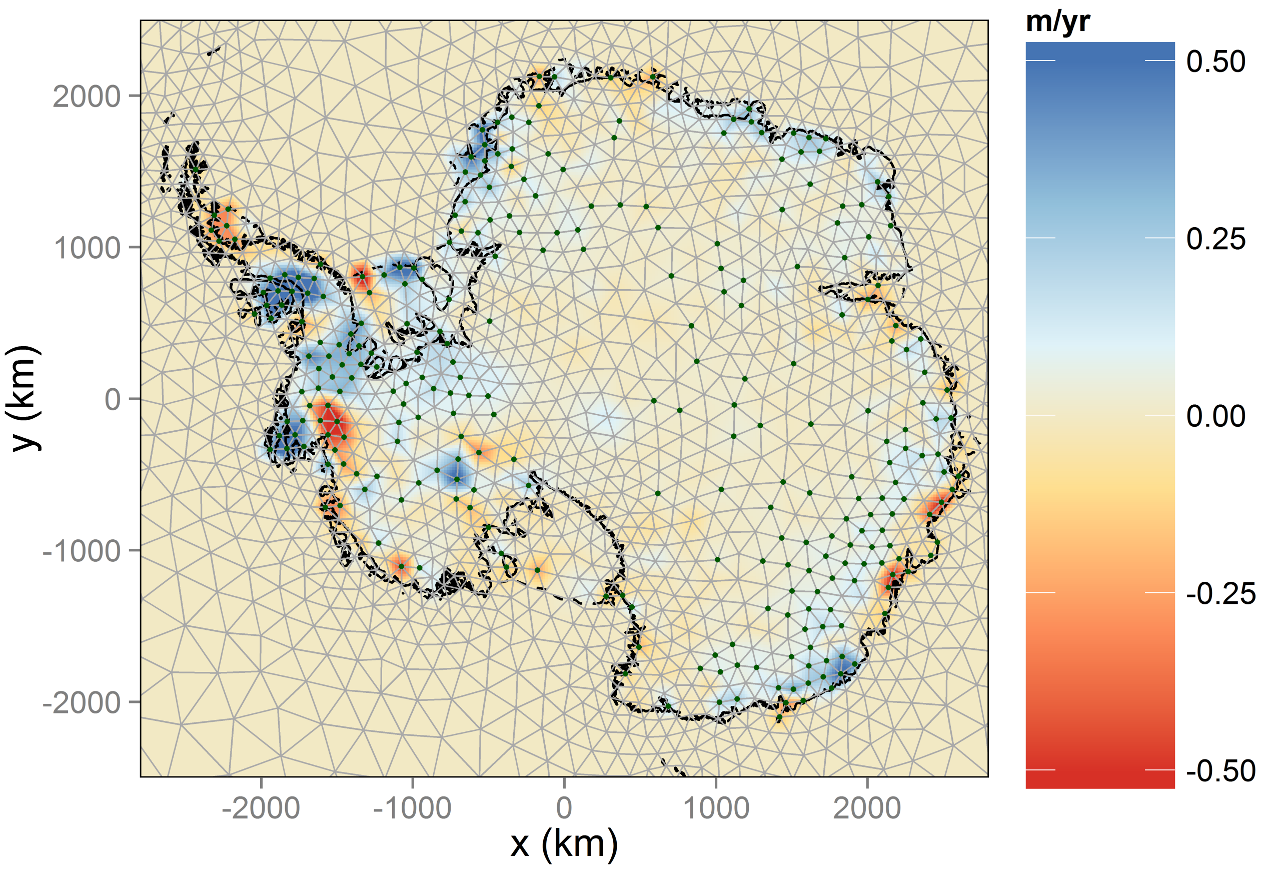
```
g <- plot_interp(Mesh,"to_plot",ds=500,min=-0.5,max=0.5,leg_title="m/yr")
```

We can plot the triangulation by simply plotting the mesh through `plot(Mesh, plot_dots = F)` where `plot_dots = F` indicates that we do not wish to plot a point at each vertex node. Since we wish to overlay it over what we have plotted so far in `g` and add the Antarctic shapefiles we use

```
g <- plot(Mesh,g=PlotAntarctica(g,shapefiles),plot_dots=F)
```

Finally we add the stipples where required in green and print the plot.

```
g <- g +  geom_point(data = subset(Mesh,abs(to_plot) > (marg_std)),
                      aes(x,y),size=1,colour=scales::muted("green")) +
    theme(text = element_text(size=15))
print(g)
```



3.2 Inference on linear combination of vertices

Frequently, producing whole maps of the processes under investigation is overly costly. For the Antarctica example, a Gaussian update on the full model with all data sets can only be run on a high memory machine (> 72GB RAM) and takes several hours. We can, however, obtain summary estimates in a few minutes on a high-end desktop with reasonable memory requirements. These summaries might be the expectation and uncertainty over a region, or sector in the Antarctica case, over multiple processes and even over multiple time points. Although this is particularly useful for when we are using observations with a large spatial support, we shall outline the basic method here.

Recall that the graph object has only two nodes, one pertaining to the processes and one to the observations. The first task is to access the data frame of the processes (by default the first vertex in the graph) which we can do through

```r
graph_df <- getDf(G@v[[1]])
```

where **getDf** simply returns the required data frame from any `MVST` object. Next, we instruct what combinations we want. In this example we will generate two linear combinations, the first the total grounded mass change over all time points and the second the total grounded mass change in 2007 ($t == 4$). The vertices which are needed are flagged as follows:

```r
Comb <- rbind(((graph_df$t== 2) & (graph_df$in_land == T))*graph_df$mass_GT_per_year,
                        ((graph_df$t== 4) & (graph_df$in_land == T))*graph_df$mass_GT_per_year)
```

With the graph and the combination matrix `Comb` defined we are now in a position to carry out inference on the linear combination through:

```r
Results_linear_comb <- Infer(G_reduced,fine_scale=F,Comb = Comb)

## Doing Cholesky and Takahashi
```

`Results_linear_comb` now contains a third field, which summarises the inferences over the linear combinations, `Comb_results`. `Comb_results` is a list with two fields, `mu` (the expectation of the estimate) and `cov` (the covariance of the estimate). For this simple analysis we get

```r
print(Results_linear_comb$Comb_results)

## $mu
##         [,1]
## [1,] 178.131
## [2,]   8.724
##
## $cov
## 2 x 2 sparse Matrix of class "dgCMatrix"
##
## [1,] 3899.06   92.34
## [2,]   92.34 3781.27
```

which indicates that we estimate a mass balance of $+178 \pm 62$ Gt yr $^{-1}$ for 2005 and 8 ± 61 Gt yr^{-1} for 2007. These values are unrealistically high for the years considered, this is predominantly due to lack of data at the main glacier outlets where most height loss is occurring.

4 Conclusion

In this vignette we have considered the modelling and inference of a very simple spatio-temporal process from satellite data. In the next vignette we will analyse the case where we have multiple interacting spatio-temporal processes and observations with a large spatial footprint.

References

[1] F. Lindgren, H. Rue, and J. Lindström. An explicit link between Gaussian fields and Gaussian Markov random fields: the stochastic partial differential equation approach. *Journal of the Royal Statistical Society B*, 73(4):423–498, Sep. 2011.