R package getgrib: an Overview

Reto Stauffer
Universität Innsbruck

Abstract

Once upon a time I was working with Sascha on a small problem on how to efficiently read grib data in R. There is the **raster** package which offers some functionality, however, the **raster** package is neither quick nor does it provide the (often) required meta information of the grib messages or is able to read data from rotated grids (like COSMO).

This was the beginning of this `getgrib` package which offers some grib handling functionalities using the ECMWF GRIB_API. Over the time the *R* package **getgrib** got some updates and extensions, and some methods have been replaced and/or removed by better, faster, or more flexible methods (written in `C`/`Fortran`/`R`).

This vignette shows a short overview over the functionalities of the *R* package **getgrib**.

Keywords: R package grib.

Contents

1. Known Problems

Please note that this package is currently in version 1.2.4 but is still in a development state (or early beta?). There are some known problems which will be fixed somewhen if needed.

COSMO Just as an example: the COSMO grib messages do not contain a "`perturbationNumber`" (while ECMWF HIRES does). This leads to problems reading the data (`getdata` crashes). Has to be re-designed somewhen. **UPDATE:** the `bilinear` method (for bilinear interpolation) works if "`perturbationNumber`" is not specified (returns `perturbationNumber=0`).

Grid specification note that the getdata operation will stop (if not used with messagenumber) whenever the specification of the grib files change from message 1 to N. This might be a bit restrictive but is what I need at the moment. Adjustments might be possible. **UPDATE:** the `bilinear` method for interpolation supports changing grids.

2. Installation

This package is using the ECMWF GRIB_API which requires the api libraries for building the package. Please note that the code below is only an example and the location of the libraries might differ on your system.

```
# Bash/Shell: setting environment variables and flags
export PKG_FCFLAGS="-static-libgfortran -L/usr -I/usr/include -lgrib_api_f90 -lgrib_api"
export PKG_LIBS="-L/usr -I/usr/include -lgrib_api_f90 -lgrib_api"

# Compile and install package
version=`cat getgrib/DESCRIPTION | grep 'Version:' | awk '{print $2}'`
R CMD build --no-build-vignettes getgrib
R CMD INSTALL getgrib_${version}.tar.gz
```

3. Get Nearest Neighbor Grid Point Data

Reto: to test.

This is basically the first method which has been developed and somehow the reason for this package. This method was desidned for Sascha to get nearest neighbor data from COSMO grids in an efficient way. However, I have to test the function and to write a help page.

4. Getting Grib Inventory

The ECMWF GRIB_API offers a console tool called `grib_ls` to create an inventory of a grib file. This function mimiks this tool in R.

```
> # Path to package internal demo file
> file <- paste(path.package("getgrib"),"data/ECEPS_12.grib",sep="/")
```

```
> inv  <- grib_ls(file,where='step=12,shortName!=2t')
> print(head(inv))

  centre dataDate dataTime perturbationNumber shortName step
1   ecmf 20170805        0                  1      10fg   12
2   ecmf 20170805        0                  1       lsp   12
3   ecmf 20170805        0                  1        cp   12
4   ecmf 20170805        0                  1        sf   12
5   ecmf 20170805        0                  1       msl   12
6   ecmf 20170805        0                  1       tcc   12

> # Another example: GFS forecast demo file
> file <- paste(path.package("getgrib"),"data/GFS_12.grib",sep="/")
> inv  <- grib_ls(file,where='step=12,shortName=cape')
> print(head(inv))

  centre dataDate dataTime perturbationNumber shortName step
1   kwbc 20170805        0          not_found      cape   12
2   kwbc 20170805        0          not_found      cape   12
```

The first line specifies the path to a demo grib file included in this package. `grib_ls` simply returns a `data.frame` containing the inventory of the specified grib file. Note that the two inputs "`parameters=`" and "`where=`" mimik the `grib_ls` inputs "`-p`" and "`-w`" and can be used in a similar way. Please see help page for a more detailed description.

5. gribdata: The Common Data Handling Object

Note: based on Fortran code.

The package is using a special object called `gribdata` for the data handling offering some basic methods for data manipulation. Most methods of the `getgrib` package are based on this object type. It is basically a `matrix` with additional attributes. These attributes are neede for further processing steps.

6. Loading Data from a Grib File Using getdata

This is the main function to read data. The data will be returned as a `gribdata` object. There are currently two different methods on how to get the data. Option one: use the `shortName` selector. In this case the grib file is scanned and all messages with the corresponding `shortName` identifier in the grib message header will be returned. Example:

```
> # Path to package internal demo file
> file <- paste(path.package("getgrib"),"data/ECEPS_12.grib",sep="/")
> # Reading all messages with "t2m"
> gribdata <- getdata(file,'2t') # getting all 2t forecasts
> # Show summary
> gribdata
```

```
      Matrix dimension:      51 x 7012
      Number of grid points: 7008
      Source file:           /usr/local/lib/R/site-library/getgrib/data/ECEPS_12.grib

      Initial dates:      1 [20170805]
      Initial hours:      1 [0]
      Steps:              1 [12]
      Members:            51 [0,1,2,...,48,49,50]
      Longitude range:    5.75 - 17.625
      Latitude range:     45 - 54
      Data range (!NA):   282.375 - 299.931
      Number of NA:       0
```

```
> # Show size
> dim(gribdata)

[1]   51 7012
```

On the other hand data can be loaded via message number. The message number corresponds to the row number from `grib_ls`. Example:

```
> # Path to package internal demo file
> file <- paste(path.package("getgrib"),"data/ECEPS_12.grib",sep="/")
> # Reading all messages with "t2m"
> inv <- grib_ls(file) # getting all 2t forecasts
> print(head(inv,3))

  centre dataDate dataTime perturbationNumber shortName step
1   ecmf 20170805        0                  1      10fg   12
2   ecmf 20170805        0                  1       lsp   12
3   ecmf 20170805        0                  1        cp   12

> # Search for message
> idx <- which( inv$shortName == "mx2t" &
+               inv$perturbationNumber == 5 & inv$step == 12)
> print(idx)

[1] 121

> # Loading data
> gribdata <- getdata(file,idx)
> # Show summary and size
> gribdata

      Matrix dimension:      1 x 7012
      Number of grid points: 7008
```

```
    Source file:           /usr/local/lib/R/site-library/getgrib/data/ECEPS_12.grib
    From message number(s):   121

    Initial dates:      1 [20170805]
    Initial hours:      1 [0]
    Steps:              1 [12]
    Members:            1 [5]
    Longitude range:    5.75 - 17.625
    Latitude range:     45 - 54
    Data range (!NA):   282.819 - 298.765
    Number of NA:       0
```

```
> dim(gribdata)
```

```
[1]    1 7012
```

Well, as shown above one message has been loaded (message `idx`) and returned the corresponding `gribdata` object. This example is loading a 2m maximum temperature forecast. Originally these data are in Kelvin. You can easily scale the data:

```
> # Loading data
> gribdata <- getdata(file,idx,scale="- 273.15")
> gribdata
```

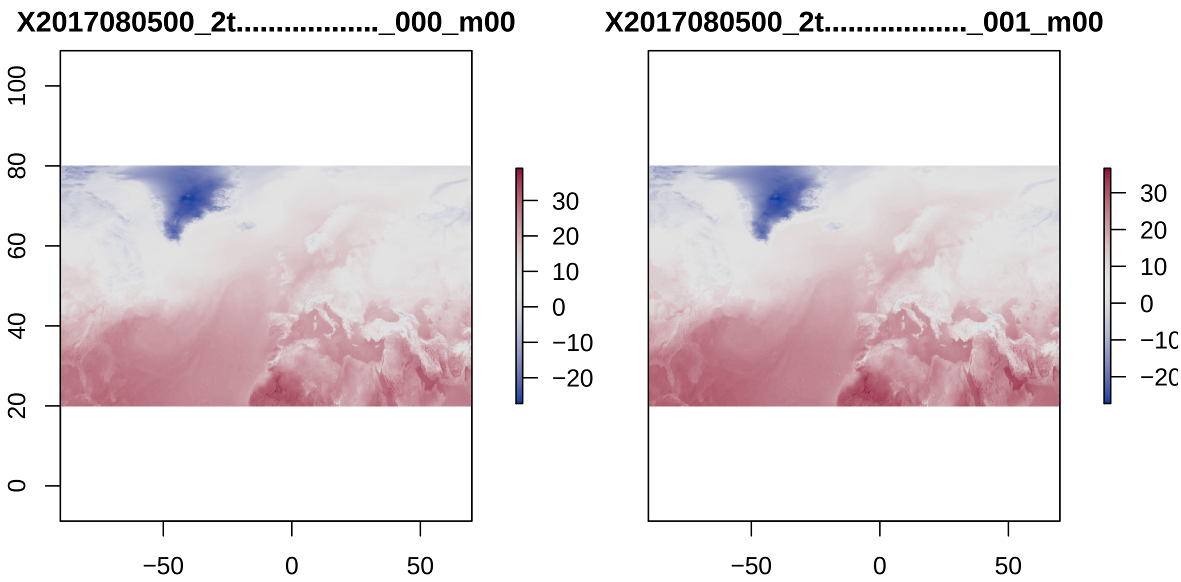
```
    Matrix dimension:      1 x 7012
    Number of grid points: 7008
    Source file:           /usr/local/lib/R/site-library/getgrib/data/ECEPS_12.grib
    From message number(s):   121

    Initial dates:      1 [20170805]
    Initial hours:      1 [0]
    Steps:              1 [12]
    Members:            1 [5]
    Longitude range:    5.75 - 17.625
    Latitude range:     45 - 54
    Data range (!NA):   9.669 - 25.615
    Number of NA:       0
```

Please note that the "`scale`" argument can be any valid mathematical expression leading to "`x scale`" where `x` are the data, `scale` the argument specified by you. Useful to e.g., scale precipitation from meters to millimeters, convert Kelvin to Celsius, or geopotential height to height.

7. Convert griddata to RasterStack Objects

Objects of type `gribdata` can easily be converted into `RasterStack` objects by simply calling `gribdata2raster`. Please note that this only works for regular latlon grids (orthogonal longitude latitude grids). This will be checked internally using `is_regular_ll_grid` using grid spacing returned by `get_grid_increments`.

```
> # Path to package internal demo file
> file <- paste(path.package("getgrib"),"data/ECMWF_t2m_demo.grib",sep="/")
> # Path to package internal demo file
> gribdata <- getdata(file,"2t",scale="-273.15")
> is_regular_ll_grid(gribdata)

[1] TRUE

> get_grid_increments(gribdata)

[1] 0.125 0.125

> # Convert to raster
> rastered <- gribdata2raster(gribdata,silent=T)
> rastered

class       : RasterStack
dimensions  : 481, 1281, 616161, 3  (nrow, ncol, ncell, nlayers)
resolution  : 0.125, 0.125  (x, y)
extent      : -90.0625, 70.0625, 19.9375, 80.0625  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=longlat +ellps=WGS84 +towgs84=0,0,0,0,0,0,0 +no_defs
names       : X2017080500_2t.................._000_m00, X2017080500_2t.................._0
min values  :                                -27.26075,                                 -27
max values  :                                 39.07714,                                  38

> # Plot
> require("colorspace")
> plot( rastered[[1:2]], col=diverge_hcl(101) )
```

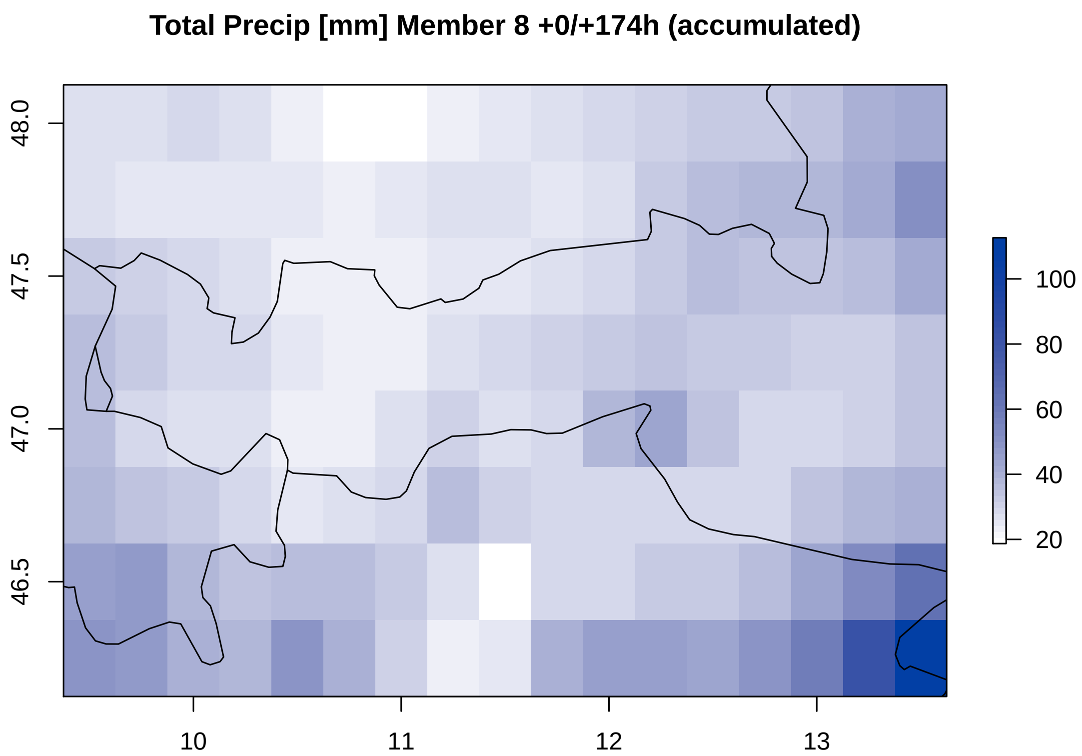


8. Deaccumulate Data in gribdata Objects

Note: based on the Fortran `gribdata` routines.

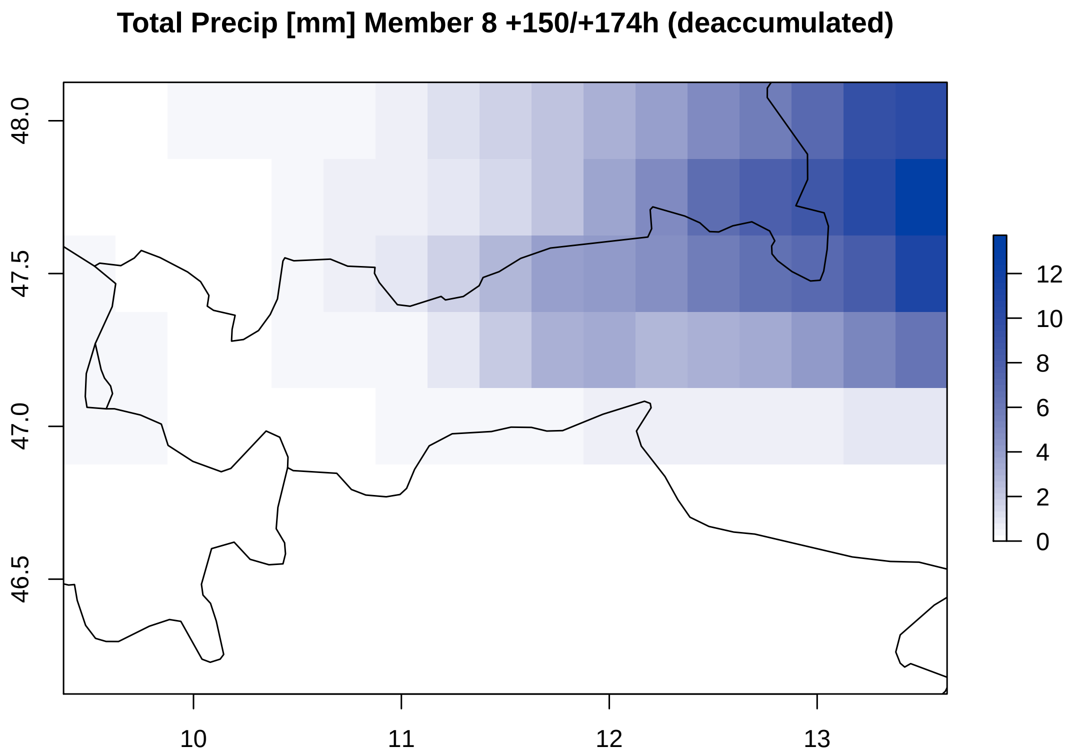
This code has been written in a few minutes and might not be the best one :). Think of reading precipitation forecast data from a grib file which are accumulated in ECMWF and ECEPS grib files. Maybe you would like to deaccumulate them. Simply do this on the `gribdata` basis. Example:

```
> # Path to package internal demo file
> file <- paste(path.package("getgrib"),
+               "data/SnowSafeHindcast_201610130000.grib",sep="/")
> # Loading file, creates gribdata object. As this is
> # total precipitation I already scale them from meters to
> # millimeters using the 'scale="*1000"' argument.
> gribdata <- getdata(file,'tp',scale='* 1000')
> # Now we can to two things. Watching member 4 accumulated fields:
> require("colorspace"); require("maps")
> cols <- sequential_hcl(51,h=260,c(0,80),l=c(100,30),power=2)
> g1 <- gribdata2raster(gribdata,silent=T)
> plot( g1[[ names(g1)[grepl(".*_174_m08$",names(g1))] ]],
+    col=cols, main="Total Precip [mm] Member 8 +0/+174h (accumulated)"  )
> map(add=T)
```



Deaccumulate on 24h-basis. Note that the additional options `setzero` and `zeroval` reduce all values below `zeroval` to `0` if `setzero=TRUE`. I used this for precipitation to remove interpolation or roundoff noise (`setzero=0.01` equals `0.01mm` per day).

```
> # Deaccumulate. For details see ?deaccumulate
> gribdata_deaccumulated <- deaccumulate(gribdata,deaccumulation=24,
+                                         setzero=TRUE,zeroval=0.01)
> g2 <- gribdata2raster(gribdata_deaccumulated,silent=T)
> plot( g2[[ names(g1)[grepl(".*_174_m08$",names(g1))] ]],
+    col=cols, main="Total Precip [mm] Member 8 +150/+174h (deaccumulated)"  )
> map(add=T)
```



9. Bilinear Interpolation of Grib Data

Note: C implementation.

Needed a quick method to bilinearely interpolate large amounts of data from grib data sets. After spending one or two nights trying to adjust my Fortran code I've decided to switch to C (using GRIB_API). The result is the method `bilinear` based on `src/bilinearlist.c`.

`bilinear` loops trough all the messages in a grib file and calculates the required weights for the interpolation. Therefore it does not matter whether grib message 1 has a different specification (e.g., shifted grid, larger domain) than grib message 2 which might sometimes be useful. In contrast to the Fortran routines C allows to directly return `SEXP` list objects allocated within the C routine which makes everything more flexible (and we do not have to loop trough the grib files twice as it is required in the Fortran based routines within this package).

Input to the method `bilinear` is the name of a grib (grib1/grib2) and a `SpatialPointsDataFrame` object. Note that the coordinate reference system is not used. The `SpatialPointsDataFrame` objects are required such that we have a clear and unique assignment between the coordinates and a station identifier, in this case a station number. An example:

```
> # Take the GFS forecast file in the demo data sets here
> file <- paste(path.package("getgrib"),"data/GFSreforecastV2_tmintmax.grib2",sep="/")
> # Define some stations
> set.seed(300)
> stations <- SpatialPointsDataFrame( data.frame("lon"=runif(10,5,17),"lat"=runif(10,45,54
+             data=data.frame("statnr"=sample(1000:2000,10)))
> print(as.data.frame(stations))

   statnr       lon      lat
1    1987 15.982960 53.47477
2    1493 14.159952 47.87581
3    1463 14.668228 52.13833
4    1060 13.805337 50.25238
5    1093 13.184814 50.88144
6    1019  5.144364 53.87073
7    1523 14.101166 52.98640
8    1889 10.987424 50.96929
9    1690 10.593305 50.74862
10   1223 15.812128 49.51763

> # Perform interpolation
> x <- bilinear(file,stations)
> head(x,3)

        init               valid step member shortName station_1987
1 2017-08-01 2017-08-01 12:00:00   12      0      tmax     303.1309
2 2017-08-01 2017-08-02 00:00:00   24      0      tmax     298.7007
3 2017-08-01 2017-08-02 12:00:00   36      0      tmax     296.3289
  station_1493 station_1463 station_1060 station_1093 station_1019 station_1523
1     306.3723     304.8222     304.1616     304.2515     291.4457     301.9116
2     297.9319     301.2691     299.2453     299.3858     291.4050     299.1144
3     304.4679     297.5939     298.2361     297.3697     291.1389     296.2375
  station_1889 station_1690 station_1223
1     302.6184     302.4580     302.4505
2     294.8688     294.5185     297.5080
3     297.3277     297.0922     300.5044
```

```
> dim(x)
```

```
[1] 352  15
```

In this case (`bilinear(file,stations)`) the result is a `data.frame` with message meta information in the first 5 columns followed by the interpolated values for all stations in `stations`. The interpolation method has a second mode where the interpolated values will be reshaped. An example:

```
> # Perform interpolation
> x <- bilinear(file,stations,reshape=TRUE)
> # Return is now a list object
> length(x)
```

```
[1] 10
```

```
> names(x)
```

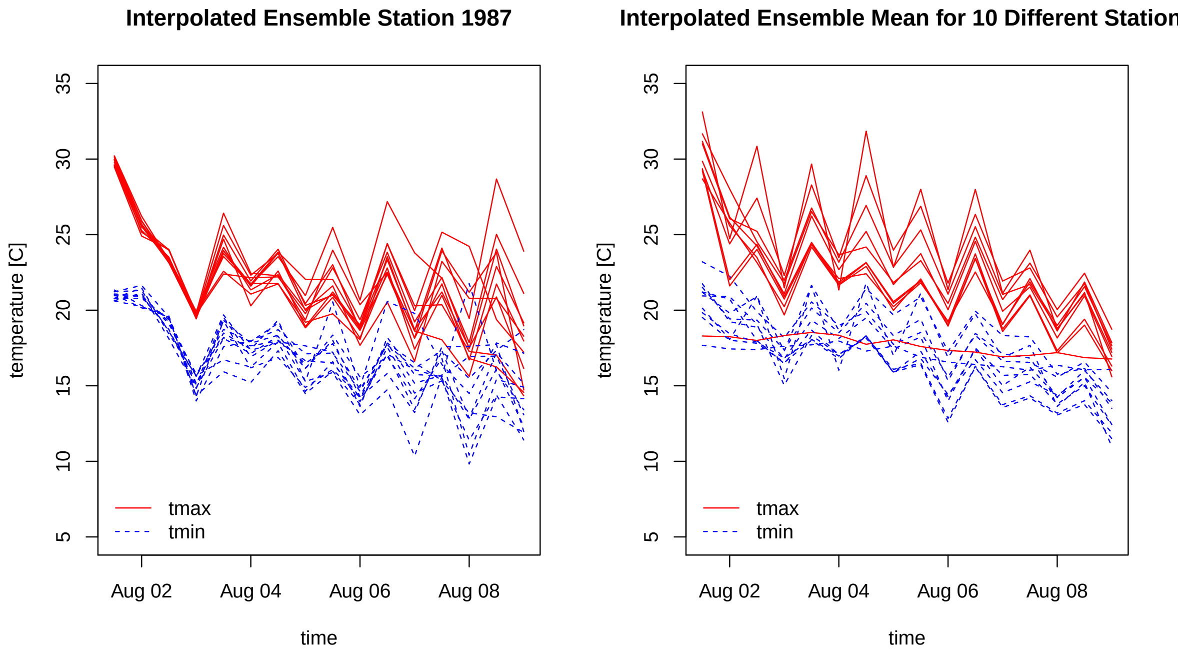
```
 [1] "station_1987" "station_1493" "station_1463" "station_1060" "station_1093"
 [6] "station_1019" "station_1523" "station_1889" "station_1690" "station_1223"
```

```
> # Each list entry contains the interpolated values.
> # If there are several members (ensemble) the columns 5-N
> # contain the different members indicated by the member number
> # (corresponds to the perturbationNumber meta information).
> print(dim(x[[1]]))
```

```
[1] 32 15
```

```
> head(x[[1]],2)
```

```
        init               valid step shortName member_0 member_1 member_2
1 2017-08-01 2017-08-01 12:00:00   12      tmax 303.1309 302.7705 303.0801
2 2017-08-01 2017-08-02 00:00:00   24      tmax 298.7007 298.8012 298.3133
  member_3 member_4 member_5 member_6 member_7 member_8 member_9 member_10
1 302.9381 303.3649 302.6891 303.3441 303.1147  303.262 302.7469  302.6224
2 298.6530 299.3680 298.4269 299.0763 299.0341  298.739 298.8430  298.0546
```



Should work for a wide range of grib files, see `?bilinear` examples for more details.

Affiliation:

Reto Stauffer
Department of Statistics
Faculty of Economics and Statistics
Universität Innsbruck
Universitätsstraße 15
6020 Innsbruck, Austria, *and*
Institute of Atmospheric and Cryospheric Sciences
Faculty of Geo- and Atmospheric Sciences
Universität Innsbruck
Innrain 52
6020 Innsbruck, Austria
E-mail: `Reto.Stauffer@uibk.ac.at`