Extreme value theory for modelling snowfalls in the Alps.

Van Thuan Romoli

Summer 2024

1 Introduction

Extreme value analysis (EVA) has emerged as one of the most important statistical disciplines for the applied sciences over the last 50 years. The distinguishing feature of EVA is the objective to quantify the stochastic behaviour of a process at unusually large, or small, levels. In particular, EVA usually require estimation of the probability of events that are more extreme than any that have already been observed.

In this project, we produce `R` code to model extreme snowfalls in the Alps using a specific EVA model, called threshold model (TM). In Section 2, we give a general introduction to EVA, then we discuss the generalised Pareto distribution (GPD) and finally we give a framework for threshold modelling. In Section 3, we introduce the ERA5 dataset used in our project and show the steps taken to model snowfalls. In Section 4, we make inferences and predictions using the model we develop and give the results of the analysis. Finally, we write a conclusion discussing the limitations of the model and the improvements that can be made to achieve better inference or prediction.

2 Extreme value analysis

2.1 Background

Let $X_1, X_2, \ldots$ be a time series. Then $M_n = \max\{X_1, \ldots, X_n\}$ is the maximum value of the process over an "n-observation" period. If the exact statistical behaviour of the X_i were known, the corresponding behaviour of M_n could be calculated exactly. In practice, the behaviour of the X_i is unknown, making exact calculations on M_n impossible. However, under suitable assumptions, the approximate behaviour of M_n for large values of n follows from detailed limit arguments by letting $n \to \infty$, leading to a family of models that can be calibrated by the observed values of M_n.

This approach might be termed the extreme value paradigm, since it comprises a principle for model extrapolation based on the implementation of mathematical limits as finite-level approximations. It is easy to object to this procedure on the grounds that, even with the support of asymptotic argument, there is an implicit assumption that the underlying stochastic mechanism of the process being modelled is sufficiently smooth to enable extrapolation to unobserved levels. However, no more credible alternative has been proposed to date.

From the outset it is important to be aware of the limitations implied by adoption of the extreme value paradigm. First, the models are developed using asymptotic arguments, and care is needed in treating them as exact results for finite samples. Second, the models themselves are derived under idealized circumstances, which may not be exact (or even reasonable) for a process under study. Third, the models may lead to a wastage of information when implemented in practice.

All of these points emphasize the importance of statistical implementation as a complement to the development of appropriate models for extremes.

2.2 The generalised Pareto distribution

Let $X_1, X_2, \ldots$ be a sequence of independent and identically distributed random variables, having marginal distribution F. It is natural to regard as extreme events those of the X_i that exceed some high threshold u. Denoting an arbitrary term in the X_i sequence by X, it follows that a description of the stochastic behaviour of extreme events is given by the conditional probability

$$\mathbb{P}(X > u + y \mid X > u) = \frac{1 - F(u+y)}{1 - F(u)}, \quad y > 0. \tag{1}$$

If the parent distribution F were known, the distribution of threshold exceedances in (1) would also be known. Since, in practical applications, this is not the case, approximations that are broadly applicable for high values of the threshold are sough.

Suppose that, for large n, the distribution of M_n can be approximated by a member of the Generalized Extreme Value Distribution family, i.e.

$$\mathbb{P}(M_n \leqslant z) \approx G(z),$$

where

$$G(z) = \exp\left\{ - \left[1 + \xi \left(\frac{z - \mu}{\sigma} \right) \right]^{-1/\xi} \right\}$$

for some $\mu, \sigma > 0$ and ξ. Then, for large enough u, the distribution function of $(X - u)$, conditional on $X > u$, is approximately

$$H(y) = 1 - \left(1 + \frac{\xi y}{\tilde{\sigma}} \right) \tag{2}$$

defined on $\{y : y > 0 \text{ and } (1 + \xi y/\tilde{\sigma}) > 0\}$, where

$$\tilde{\sigma} = \sigma + \xi(u - \mu).$$

A full proof of the theorem is given by Leadbetter et al. (1983).

The family of distributions defined by (2) is called the generalised Pareto (GPD) family. The shape parameter ξ is dominant in determining the qualitative behaviour of the GPD:

- If $\xi < 0$, then the distribution of excesses has an upper bound of $u - \tilde{\sigma}/\xi$.
- If $\xi > 0$, then the distribution has no upper limit.
- If $\xi = 0$, then by taking the limit $\xi \to 0$ in (2), we have that $H(y) = 1 - \exp\{-y/\tilde{\sigma}\}$, $y > 0$, corresponding to an exponential distribution with parameter $1/\tilde{\sigma}$.

2.3 Modelling threshold excesses

Equation (2) suggests the following framework for extreme value modelling. The raw data consist of a sequence of independent and identically distributed measurements $x_1, \ldots, x_n$. Extreme events are identified by defining a high threshold u, for which the exceedances are $\{x_i : x_i > u\}$. Label these exceedances by $x_{(1)}, \ldots, x_{(k)}$, and define threshold excesses by $y_j = x(j) - u$, for $j = 1, \ldots, k$. The y_j may be regarded as independent realizations of a random variable whose distribution can be approximated by a member of the GPD family. Inference consists of fitting the GPD family to the observed threshold exceedances, followed by model verification and extrapolation. We will show how this looks in practice in Section 3.

A clear issue that arises is the one of threshold choice, where we seek a balance between bias and variance. Too low a threshold is likely to violate the asymptotic basis of the model, leading to bias; too high a threshold will generate few excesses with which the model can be estimated, leading to high variance. The standard practice is to adopt as low a threshold as possible, subject to the limit model providing a reasonable approximation. Two methods are available for this purpose: one is an exploratory technique carried out prior to model estimation; the other is an assessment of the stability of parameter estimates, based on the fitting of models across a range of different thresholds. The details of this is beyond the scope of this project, and as mentioned earlier, real-world example is provided in Section 3.

3 Modelling excesses with the ERA5 Copernicus dataset

3.1 The ERA5 Copernicus dataset

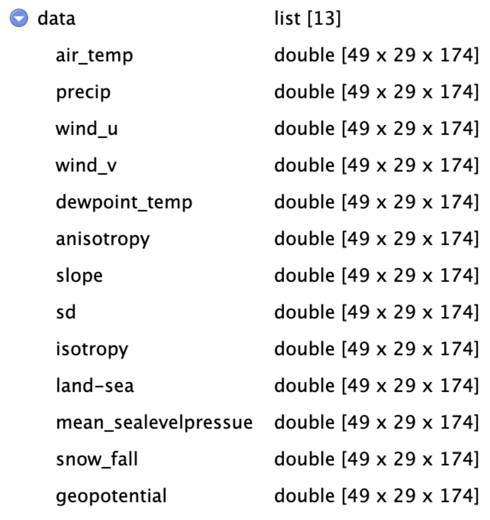
The ERA5 is a global atmospheric dataset produced by the European Centre for Medium-Range Weather Forecasts under the Copernicus Climate Change Service. It provides a consistent, long-term record of atmospheric, land, and oceanic variables by combining observations with numerical weather prediction. It provides hourly data from 1940 to the present on a lat-lon grid of 0.25 degrees, with 137 vertical levels extending up to 0.01 hPa. The dataset includes key variables such as temperature, precipitation, altitude, etc..., along with other surface and upper-air parameters.

In this project, the ERA5 dataset is modified to only consider 174 monthly data from 2010, on a 29×49 grid overlaying the Italian Alps, with one vertical level, namely the land level, with the following land variables: air temperature, precipitations, u- and v-components of wind speed, dew-point temperature, anisotropy, slope, snow depth, isotropy, whether it is land or sea, mean sea level pressure, snow falls, and geo-potential.

3.2 Data exploration and visualisation

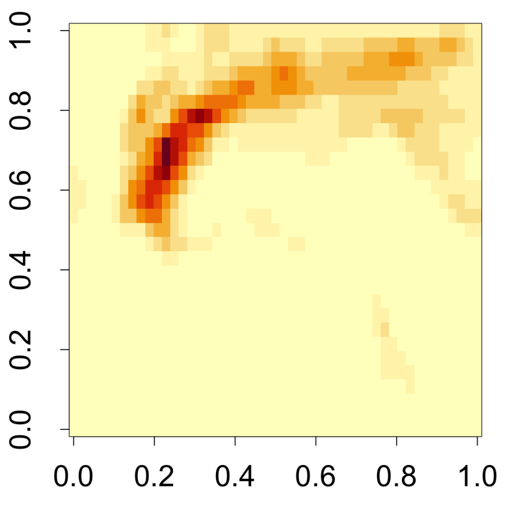
This modified dataset can be imported to `R` as a list consisting of land variables. Each of these variables is organised as a $49 \times 29 \times 174$ matrix where 49×29 is the spatial grid overlaying the Italian Alps and the last number indicates the month number, starting from January 2010. This is summarised in Figure 1a. For example, if we want to extract snowfalls from December 2021, we may write the following code, with output in Figure 1b.

```r
load("Alps_precip.Rdata")

example <- data$snow_fall[,,(12 + 11 * 12)]
image(example)
```



data	list [13]
air_temp	double [49 x 29 x 174]
precip	double [49 x 29 x 174]
wind_u	double [49 x 29 x 174]
wind_v	double [49 x 29 x 174]
dewpoint_temp	double [49 x 29 x 174]
anisotropy	double [49 x 29 x 174]
slope	double [49 x 29 x 174]
sd	double [49 x 29 x 174]
isotropy	double [49 x 29 x 174]
land-sea	double [49 x 29 x 174]
mean_sealevelpressue	double [49 x 29 x 174]
snow_fall	double [49 x 29 x 174]
geopotential	double [49 x 29 x 174]

(a) Summary of the modified ERA5 dataset.



(b) Output of code `image(example)`.

Figure 1: Exploration and visualisation of the modified ERA5 dataset.

3.3 Long format, data cleaning, and normalisation

To be able to analyse the data set, we need to reshape our $49 \times 29 \times 174$ matrix to a long data frame of 18 columns and 247254 rows, where each column represents a variable (including land variables, longitude, latitude, and month) and each row contains the information of a specific location (identified through the variables latitude and longitude) in a specific month. We store this to a data frame called `alps_df`.

```
colnames(alps_df)

# [1] "air_temp"    "alt"   "wind_u"    "wind_v"    "dewpoint_temp"     "anisotropy"
# [7] "slope"     "sd"    "isotropy"    "land_sea"  "mean_sealevelpressure"
# [12] "geopotential"    "dates"     "months"    "lon"   "lat"
# [17] "snow_fall"  "precip"
```

The next step is to clean the data as certain entries have no data, represented by a `na` value. There are several ways to overcome this issue, for example we could replace the `na` values with the average for that variable, but given the large amount of data, we opted for the simpler approach. We remove each row that contain a `na` value.

```
alps_df <- na.omit(alps_df)
```

The dependent variable in our model will be the `snow_fall` covariate. In our data frame, `snow_fall` is a small non-negative real number and is measured in meters. To help with the analysis we remove the zero values and change the measurement unit to millimetres.

```
alps_extremes <- alps_df[alps_df$snow_fall > 0,]
alps_extremes$snow_fall <- alps_extremes$snow_fall * 1000
```

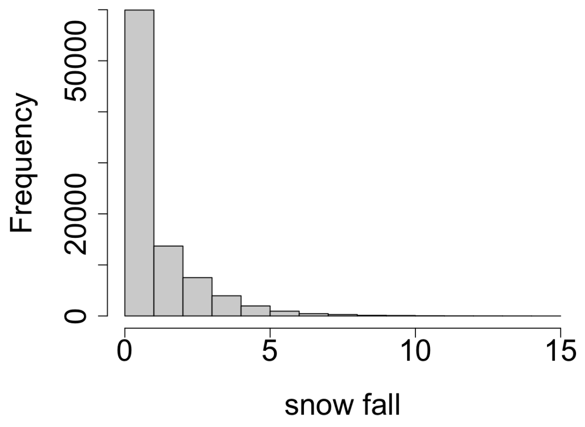
The last step to make our data frame ready for analysis is normalisation of every land variable.

```
for (i in 1:12) {
  alps_extremes[,i] <- (alps_extremes[,i] - mean(alps_extremes[,i])) / sd(alps_extremes
    [,i])
}
```

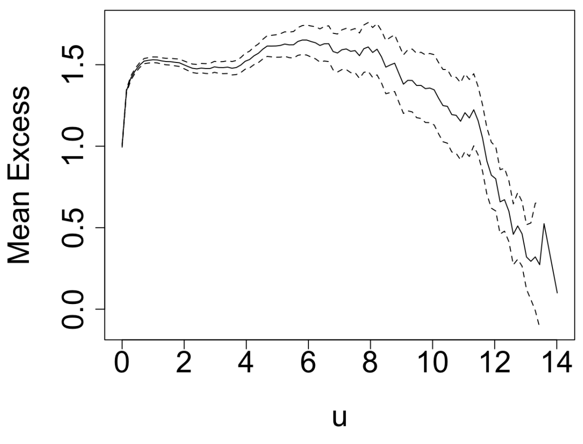
3.4 Threshold selection

As mentioned in Section 2.3, it is key to choose a threshold value, labelled u, that strikes the right balance between bias and variance. This is done graphically, and Figure 2 shows the plots required for threshold selection.

```
hist(alps_extremes$snow_fall, xlab = "snow fall")

mrl.plot(alps_extremes$snow_fall)

gpd.fitrange(alps_extremes$snow_fall,0.4,8)

u <- 6
```



(a) Histogram for the snowfalls.



(b) Mean residual plot for different values of u.

Figure 2: Plots for threshold selection.

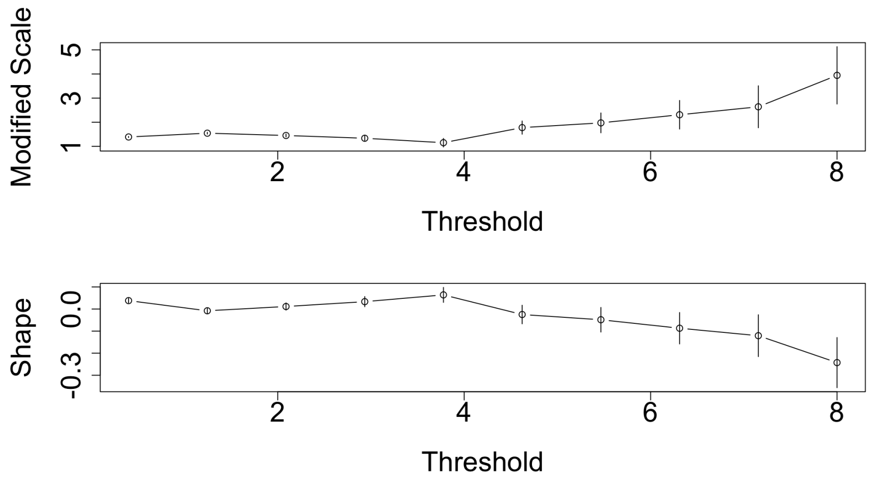


Figure 3: Stability of parameter estimates, as u varies.

The histogram in Figure 2a shows that the majority of data satisfies `snow_fall` < 5. Hence, we seek a threshold $u_0 \geqslant 5$ for which the excesses can be modelled by the GPD.

Above a threshold u_0 at which the GPD provides a valid approximation to the excess distribution, the mean residual life plot should be approximately linear in u, furthermore the estimates of the shape parameter ξ should be approximately constant, while estimates of $\tilde{\sigma}$ should be linear in u.

Figure 2b shows linearity from approximately $u = 6$ to $u = 11$ and also from $u = 11$ to $u = 13$. While we could choose a threshold value of 11, we would not have enough data to do any meaningful inference. Figure 3 shows linearity of the modified scale parameter from around $u = 5$ and constant estimates of the shape parameter from $u = 6$.

Using this information, we settle for a threshold value $u_0 = 6$.

Now we are ready to fit the GPD to our excesses.

```
fit0 <- gpd.fit(xdat = alps_extremes$snow_fall, threshold = u)

# diagnostic plots
gpd.diag(fit0)

# distribution parameters
fit0$mle

# [1]  1.82924449 -0.09922936
```

The log of the modified scale parameter $\log \tilde{\sigma}$ is 1.82924449 and the shape parameter ξ is -0.09922936.

A model diagnostic plot is given in Figure 4. None of the plots gives any real cause for concern about the assumption of the fitted model.

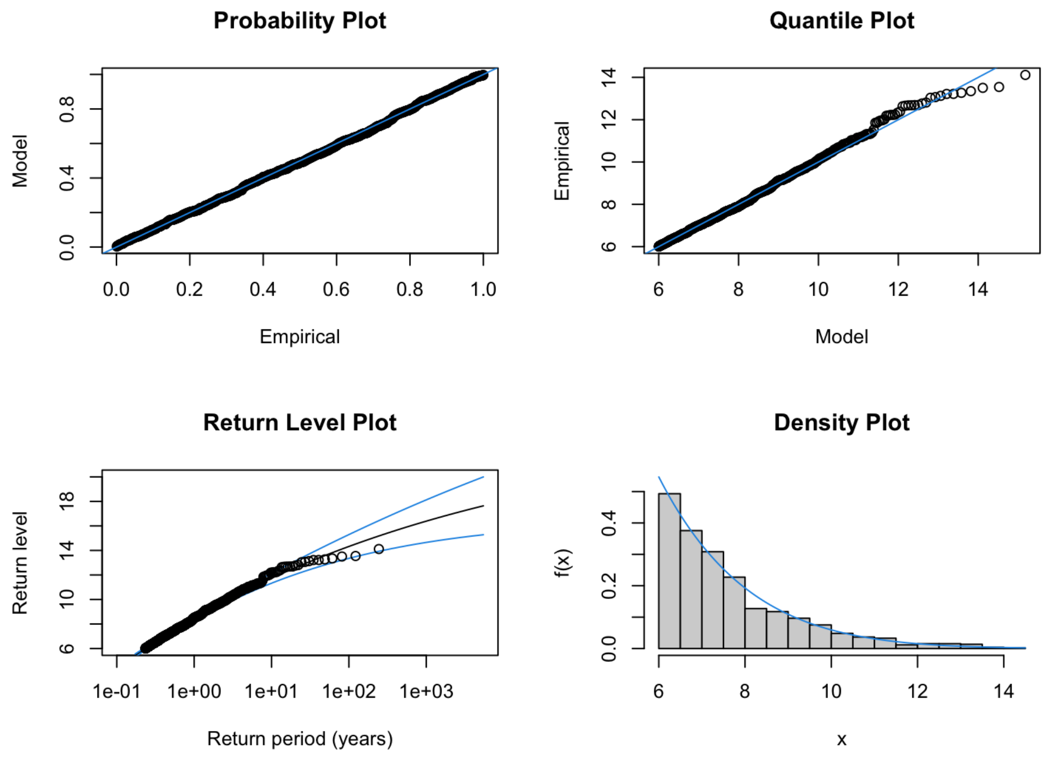


Figure 4: Model diagnostic plots.

3.5 Model selection

So far, we have only worked with the `snow_fall` dependent variable. To build the full model we now consider the covariates (land variables). For simplicity and explainability, we choose to build a linear model on the log of the scale parameter, and use Forward AIC model selection.

Forward AIC is a model selection technique that builds a regression model by starting with no predictors and incrementally adding variables one at a time based on the Akaike Information Criterion (AIC). At each step, the variable that most improves the model, by producing the greatest decrease in AIC, is added. AIC balances goodness of fit with model complexity (lower AIC is better). This process continues until no additional variable results in a lower AIC, meaning further additions would worsen the model's balance between fit and simplicity. Forward AIC is a greedy, stepwise approach and may not always find the globally optimal model, but it provides an efficient way to identify a well-performing subset of predictors.

To illustrate this process, we show how the first covariate is determined. The starting AIC value for the intercept only model is

```
print(2 * fit0$nllh + 2 * 2)
# 3127.999
```

Then, in turn, we calculate the AIC for each model given different covariates.

```
fit1covariate <- list()
for (i in 1:12) {
  fit1covariate[[i]] <- gpd.fit(xdat = alps_extremes$snow_fall,
                 ydat = cbind(alps_extremes[,i]),
                 threshold = u,
                 sigl = c(1), siglink = exp)
}

for (i in 1:12) {
  print(c(2 * fit1covariate[[i]]$nllh + 2 * 3, names(alps_extremes)[i]))
}

#[1] "3127.69324686521" "air_temp"
#[1] "3105.23985048198" "alt"
#[1] "3129.08812818259" "wind_u"
#[1] "3129.99035017109" "wind_v"
#[1] "3129.50469415774" "dewpoint_temp"
#[1] "3129.13906086469" "anisotropy"
#[1] "3120.52539122854" "slope"
#[1] "3119.24366459593" "sd"
#[1] "3121.44788396319" "isotropy"
#[1] "3117.8698286819"  "land_sea"
#[1] "3129.35766154901" "mean_sealevelpressure"
#[1] "3108.11880184508" "geopotential"
```

The covariate that improves model fit the most is `alt`, altitude. Hence we add it to our model:

```
# Add second covariate, i.e. ydat = altitude

gpd.fit(xdat = alps_extremes$snow_fall,
                ydat = cbind(alps_extremes$alt),
                threshold = u,
                sigl = c(1), siglink = exp)
```

The full linear model is the following:

```
# ydat = altitude + isotropy + land sea + dewpoint temp + air temp + sd + slope +
    geopotential + land_sea*sd

fitlin <- gpd.fit(xdat = alps_extremes$snow_fall,
                  ydat = cbind(alps_extremes$alt,
                               alps_extremes$isotropy,
                               alps_extremes$land_sea,
                               alps_extremes$dewpoint_temp,
                               alps_extremes$air_temp,
                               alps_extremes$sd,
                               alps_extremes$slope,
                               alps_extremes$geopotential,
                               alps_extremes$land_sea*alps_extremes$sd),
                threshold = u,
                sigl = c(1,2,3,4,5,6,7,8,9), siglink = exp)

gpd.diag(fitlin)
```

Note that we added only one interaction term. Calculating all interaction terms is computationally expensive and adds complexity to the model. We choose to stick with a simpler model, with only one interaction term.

A model diagnostic plot is given in Figure 5. Once again, there are no real concerns about the assumptions of the fitted model.

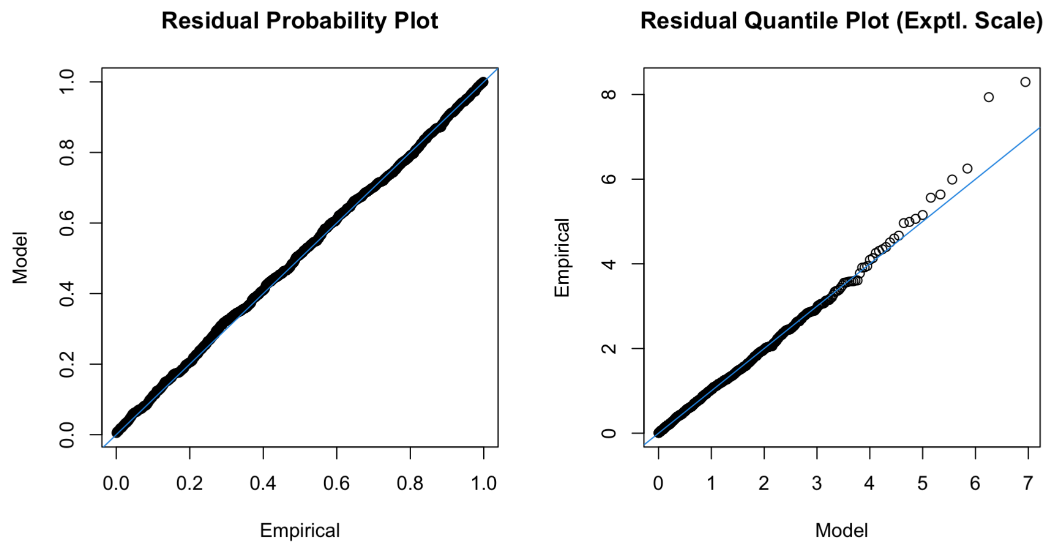


Figure 5: Diagnostic plots of the full linear model.

4 Results

Firstly, we want to obtain a table for the coefficients of the covariates. To obtain the confidence levels for the coefficients, we use non-parametric bootstrapping, where the code is given below.

```
# Non-parametric bootstrap
fit.boot <- list()
MLE.boot <- array(dim = c(11,1000))
for (i in 1:1000) {
  alps_bootstrap <- alps_extremes[sample(1:nrow(alps_extremes), replace = TRUE),]

  fit.boot[[i]] <- gpd.fit(xdat = alps_bootstrap$snow_fall,
                        ydat = cbind(alps_bootstrap$alt,
                                     alps_bootstrap$isotropy,
                                     alps_bootstrap$land_sea,
                                     alps_bootstrap$dewpoint_temp,
                                     alps_bootstrap$air_temp,
                                     alps_bootstrap$sd,
                                     alps_bootstrap$slope,
                                     alps_bootstrap$geopotential,
                                     alps_bootstrap$land_sea*alps_bootstrap$sd),
                        threshold = u,
                        sigl = c(1,2,3,4,5,6,7,8,9), siglink = exp)
  for (j in 1:11) {
    MLE.boot[j,i] <-  fit.boot[[i]]$mle[j]
  }
}

MLE.ci <- list()

for (i in 1:11) {
  low <- quantile(MLE.boot[i,],0.025)
  high <- quantile(MLE.boot[i,],0.975)
  med <- quantile(MLE.boot[i,],0.5)

  MLE.ci[[i]] <- c(low,high,med)
}
```

The full model and the coefficients of the covariates, with a 95% confidence interval, is the following:

$$\begin{aligned}\log \tilde{\sigma} = \beta_0 + \beta_{altitude} \cdot \text{altitude} + \beta_{isotropy} \cdot \text{isotropy} + \beta_{land\_sea} \cdot \text{land\_sea} + \\ + \beta_{dewpoint\ temp} \cdot \text{dewpoint temp} + \beta_{aor\ temp} \cdot \text{air temp} + \beta_{sd} \cdot \text{sd} + \beta_{slope} \cdot \text{slope} + \\ + \beta_{geopotential} \cdot \text{geopotential} + \beta_{interaction} \cdot \text{land\_sea} * \text{sd}\end{aligned}$$

$$\xi = -0.163 \quad (-0.232, -0.092)$$

where

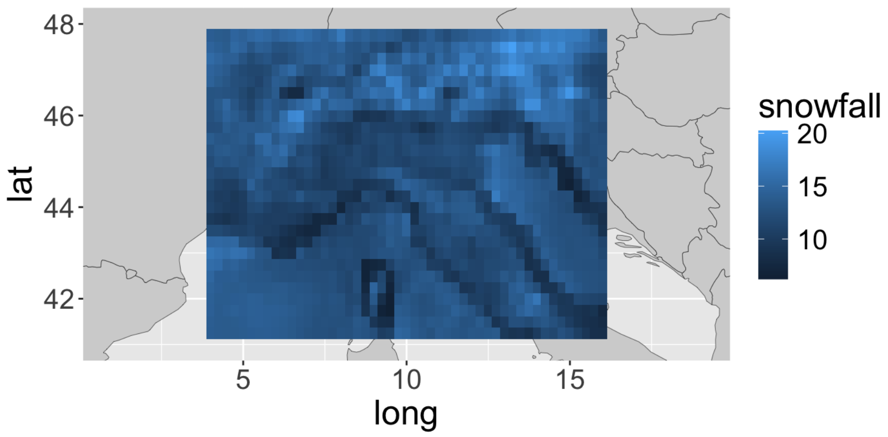
$$\begin{aligned}\beta_0 &= -0.422 \quad (-1.284, 0.753)\\ \beta_{altitude} &= +0.129 \quad (0.030, 0.237)\\ \beta_{isotropy} &= +0.135 \quad (-0.001, 0.274)\\ \beta_{land\_sea} &= +0.772 \quad (-2.772, 3.964)\\ \beta_{dewpoint\ temp} &= +1.356 \quad (0.229, 1.891)\\ \beta_{air\ temp} &= -1.388 \quad (-2.045, -0.159)\\ \beta_{sd} &= -0.258 \quad (-1.418, 0.762)\\ \beta_{slope} &= +0.255 \quad (-0.231, 0.624)\\ \beta_{geopotential} &= -0.004 \quad (-0.198, 0.200)\\ \beta_{interaction} &= +0.613 \quad (-1.98, 3.966).\end{aligned}$$

The first thing to note is that the shape parameter is negative, which suggests that the distribution has a finite upper bound. This makes sense as we don't expect an unbounded amount of snowfall, and given that it does snow, we can predict an upper bound for the amount of snowfall (this can be calculated using $u - \tilde{\sigma}/\xi$).

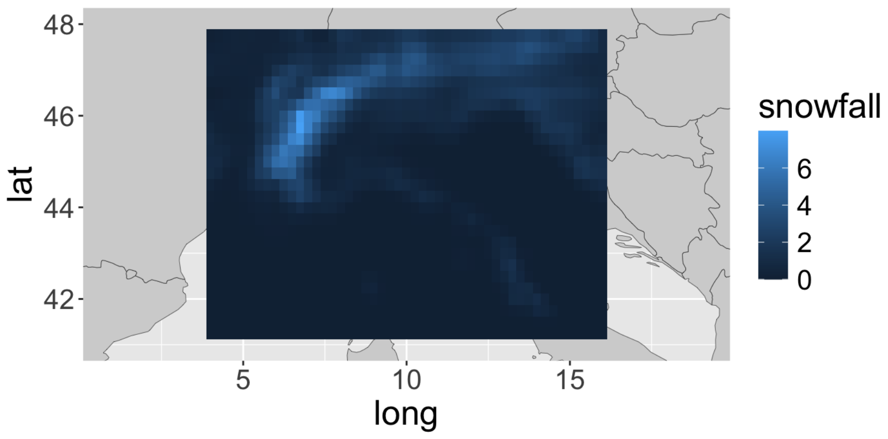
Furthermore, we see that the statistically significant covariates are altitude, air temperature, and dew point temperature. As expected, higher altitude, lower temperature, and higher dew point temperature increase the amount of snowfalls.

We conclude this analysis by showing how this model can be used to predict an upper bound for snowfalls in the Alps, given it does snow. We use the earlier example of snowfalls in December 2021. Figure 6a shows the plot that the code below produces and is compared to the observed values in that month.

```r
# Creates a dataframe with only the significant covariates
alps_significant <- data.frame(alt,isotropy,land_sea,dewpoint_temp,air_temp,sd,slope,
    geopotential,snow_fall,lon,lat,dates)
alps_significant <- na.omit(alps_significant)

alps_significant$snow_fall <- alps_significant$snow_fall * 1000
for (i in 1:8) {
  alps_significant[,i] <- (alps_significant[,i] - mean(alps_significant[
    alps_significant$snow_fall > 0,i])) / sd(alps_significant[alps_significant$snow_fall
     > 0,i])
}

# Sets a specific date
date <- "2021-12"

alps_significant <- alps_significant[alps_significant$dates == date,]

# Calculates the return levels for each row in the dataframe
X1 <- cbind(rep(c(1),nrow(alps_significant)),
                alps_significant$alt,
                alps_significant$isotropy,
                alps_significant$land_sea,
                alps_significant$dewpoint_temp,
                alps_significant$air_temp,
                alps_significant$sd,
                alps_significant$slope,
                alps_significant$geopotential,
                alps_significant$land_sea*alps_significant$sd)

beta.low <- numeric()
for (i in 1:10) {
  beta.low[i] <- MLE.ci[[i]][1]
}

beta.high <- numeric()
for (i in 1:10) {
  beta.high[i] <- MLE.ci[[i]][2]
}

beta.avg <- numeric()
for (i in 1:10) {
  beta.avg[i] <- MLE.ci[[i]][3]
}

sigma.low <- exp(X1 %*% beta.low)
sigma.high <- exp(X1 %*% beta.high)
sigma.avg <- exp(X1 %*% beta.avg)

returnlvl <- u - sigma.avg / MLE.ci[[11]][3]
```



(a) Plot of predicted upper bound for snowfalls.



(b) Plot of observed snowfalls.

Figure 6: Comparison of prediction and realisation of snowfalls in December 2021, in mm.

5 Conclusion

In this project, we explore how extreme value theory can be used in the real-world to make predictions and inferences. In particular, we model extreme snowfalls in the Italian Alps. Although we are able to make a simple explainable model, in reality things are more complex and require more sophisticated tools to capture these intricacies. For example, we used a linear link function, but non-linear functions such as polynomial regression, neural networks, or decision trees might be more effective at capturing complex patterns. Another issue is that we haven't used any specialised tools for the analysis of temporal and spatial data. For example, we assume that the time series is independent and stationary, while in practice is likely not the case. We also used forward AIC for model selection however, a more robust method such as k-fold cross-validation might yield better results.

Lastly, while this project is focused on statistics, it is worth noting a few improvements that can make computations faster. The code for model selection is copy and pasted and should be automated using classes or functions. Furthermore, there are a few nested loops that are computationally taxing and array broadcasting should be used instead.

A Model selection with k-fold cross-validation

In Section 3, we have used forward AIC model selection. In this appendix, we investigate the use of k-fold cross-validation for model selection.

k-fold cross-validation is primarily used to evaluate a model's predictive performance by dividing the data into k subsets (folds), training the model on $k-1$ folds, and validating it on the remaining fold, repeating this process k times. It provides an estimate of how well a model generalises to unseen data and is applicable to any model type, though it can be computationally intensive. It estimates the model performance using a loss function that captures the difference between the model prediction and the remaining fold.

In contrast, forward AIC model selection is a stepwise feature selection method typically used with linear models. It begins with no predictors and adds variables one at a time based on which addition most improves the Akaike Information Criterion (AIC), a metric that balances model fit and complexity. Unlike cross-validation, it evaluates models based on likelihood and assumes a specific model form. While AIC-based selection is faster and convenient for linear models, it may not always yield the best predictive performance on new data.

We choose to use 5 folds and a likelihood loss function. Below we provide that code to achieve this, we will only show one covariate as the process is repeated for two or more covariates.

Firstly, we create a function for shuffling the data randomly.

```r
shuffleshuffle <- function() {

  tobereturned <- list()

  # Removes temporal dependencies and thresholds the data
  alps_shuffled <- alps_extremes[sample(1:nrow(alps_extremes), replace = FALSE),]
  alps_shuffled <- alps_shuffled[alps_shuffled$snow_fall > 6,]

  # Separates dataframe in 5 folds
  folds <- list()
  for (i in 1:5) {
    folds[[i]] <- alps_shuffled[((nrow(alps_shuffled)*((i-1)/5))+1):(nrow(alps_shuffled)
    *(i/5)),]
  }
  # Groups together 4 of the 5 folds for fitting and leaves out 1 for evaluation
  foldtraining <- list()
  for (i in 1:5) {
    foldtraining[[i]] <- rbind(folds[[(((i) %% 5) + 1)]],folds[[(((i+1) %% 5) + 1)]],
    folds[[(((i+2) %% 5) + 1)]],folds[[(((i+3) %% 5) + 1)]])

    #print(c(  (((i) %% 5) + 1), (((i+1) %% 5) + 1), (((i+2) %% 5) + 1), (((i+3) %% 5) +
     1)               ))
  }

  tobereturned[[1]] <- folds
  tobereturned[[2]] <- foldtraining

  return(tobereturned)
}
```

Then, for 50 times and for each covariate, we separate the shuffled data into 4 training folds and 1 validation fold. We fit the GPD using the training fold and calculate the likelihood on the validation fold and applied a likelihood loss function. We take the average of the 50 times for each covariate. The covariate that yields the lowest loss is then chosen.

```r
mylistpacked <- list()
for (k in 1:50) {
  fit1.5kfold <- list()
  folds <- shuffleshuffle()[[1]]
  foldtraining <- shuffleshuffle()[[2]]
  for (j in 1:12) {
    nllh1.5kfold <- numeric()
    fit <- list()
    for (i in 1:5) {
      fit[[i]] <- gpd.fit(xdat = foldtraining[[i]]$snow_fall,
                          ydat = cbind(foldtraining[[i]][,j]),
                          threshold = 6,
                          sigl = c(1), siglink = exp)
      X_1 <- cbind(rep(c(1),nrow(folds[[i]])),
                   folds[[i]][,j])
      beta <-c(fit[[i]]$mle[1],
               fit[[i]]$mle[2])
      sigma <- exp(X_1 %*% beta)

      nllh1.5kfold[i] <- -sum(dgpd(folds[[i]]$snow_fall,
                                   sigma = sigma,
                                   xi = fit[[i]]$mle[3],
                                   log.d = TRUE, u = 6))
    }
    fit1.5kfold[j] <- (sum(nllh1.5kfold)/5)
  }
  mylistpacked[[k]] <- fit1.5kfold
}

mylistunpacked <- list()
for (i in 1:12) {
  templist <- numeric()
  for (j in 1:50) {
    templist[j] <- mylistpacked[[j]][i]
  }
  mylistunpacked[i] <- sum(as.numeric(templist)) / 50
}

for (i in 1:12) {
  print(c(mylistunpacked[[i]],names(alps_extremes)[i]))
}

#[1] "311.295959842459" "air_temp"
#[1] "309.076113194956" "alt"
#[1] "311.45103637463" "wind_u"
#[1] "311.544053375915" "wind_v"
#[1] "311.482194694513" "dewpoint_temp"
#[1] "311.472261966647" "anisotropy"
#[1] "310.605842648072" "slope"
#[1] "310.473853203316" "sd"
#[1] "310.672747335444" "isotropy"
#[1] "310.348674830833" "land_sea"
#[1] "311.468408361884" "mean_sealevelpressure"
#[1] "309.360647482484" "geopotential"
# Add second covariate, i.e. ydat = altitude
```

It turns out that until the 5th covariate the result is the same as using AIC. In the 6th covariate selection, when fitting the model to the training data, the model is not valid in the validation fold. A different method would be required to do model selection; however, since the choice of covariates hasn't differed from the AIC method, we suggest to proceed to take the full model as found earlier.