unseen_open

Timo Kelder

Sep 14, 2021

INTRODUCTION:

An open, reproducible and transferable workflow to assess and anticipate climate extremes beyond the observed record.

UNSEEN-open is an open source project using the *global* SEAS5 and ERA5 datasets. It makes evaluation of model simulations and extreme value analysis easy in order to **anticipate climate extremes** beyond the observed record. The project is developed as part of the ECMWF summer of weather code 2020 (esowc), which is funded by Copernicus.

UNSEEN-open relies on xarray for data preprocessing and uses ggplot and extRemes for the extreme value analysis. The extreme value utilities are being developed into an UNSEEN Rpackage.

Read all about UNSEEN-open in our preprint!

CHAPTER

ONE

APPLICATIONS

In our recent NPJ Climate and Atmospheric Science paper we outline four potential applications where we believe UNSEEN might prove to be useful:

1. Help estimate design values, especially relevant for data scarce regions
2. Improve risk estimation of natural hazards by coupling UNSEEN to impact models
3. Detect trends in rare climate extremes
4. Increase our physical understanding of the drivers of (non-stationarity of) climate extremes

We hope this approach may see many applications across a range of scientific fields!

1.1 What is UNSEEN?

The UNprecedented Simulated Extremes using ENsembles (UNSEEN, Thompson et al., 2017) approach uses forecast ensemble members to compute robust statistics for rare events, which is challenging to compute from historical records. UNSEEN may therefore help to identify plausible – yet unseen – weather extremes and to stress-test adaptation measures with maximum credible events. For more info about UNSEEN, see our preprint, BOX A in particular.

We believe UNSEEN has large potential as a tool to inform decision-making about unforeseen hydro-climatic risks. In order to apply UNSEEN: 1. Model ensemble members must be applicable for generating large samples of weather events (see Box B in paper); and 2. Large volumes of data must be handled.

Our paper presents a *6-step protocol* (see below) and, as part of the protocol, the *UNSEEN-open workflow*, to guide users in applying UNSEEN more generally. The paper discusses the protocol in detail, including the practicalities of the workflow and its potential application to other datasets. The technical steps and relevant code are documented here. The protocol is applicable to any prediction system, whilst the code and guidance for UNSEEN-open is developed to work with the Copernicus Data Store (CDS, https://cds.climate.copernicus.eu/).

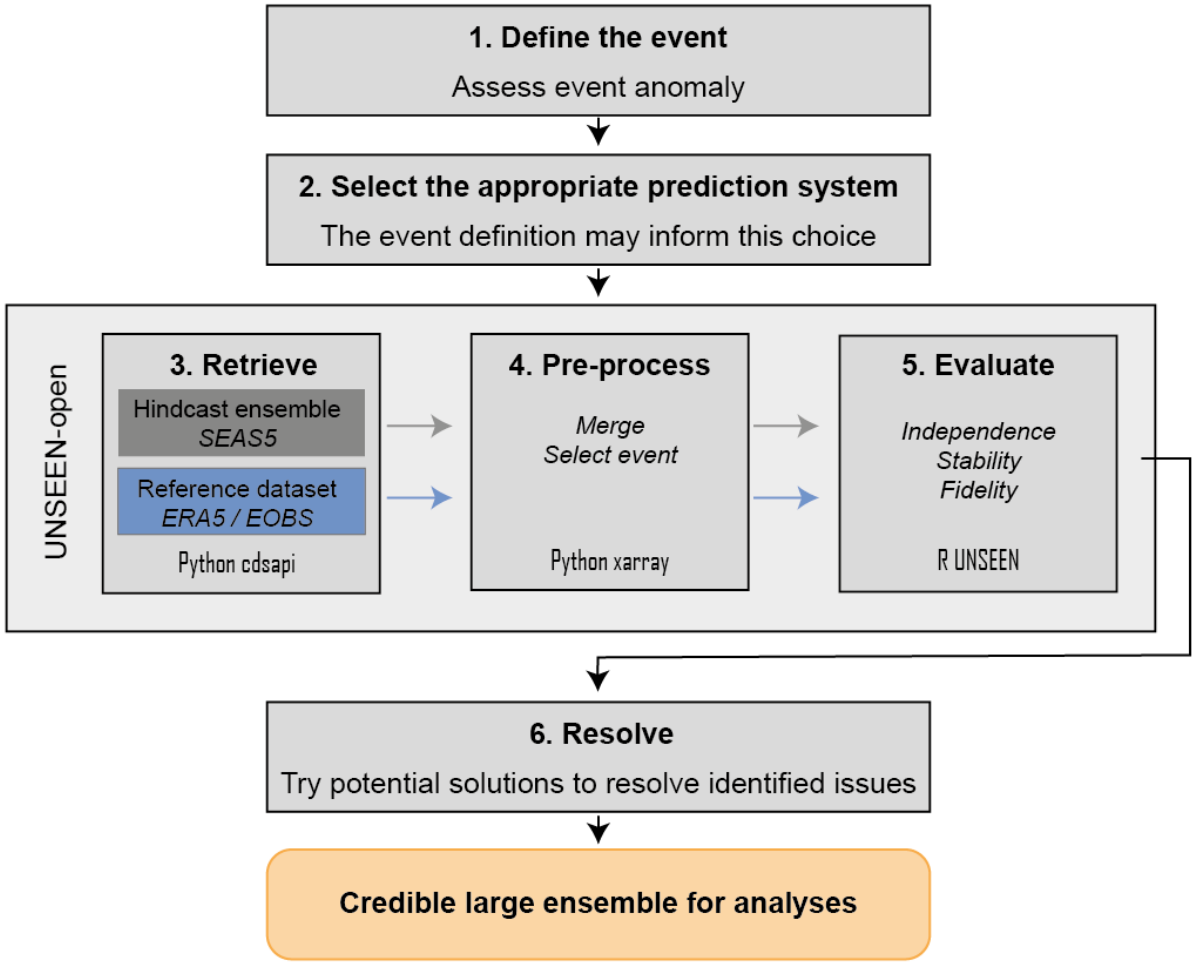
1.1.1 UNSEEN-open

In this project, the aim is to build an open, reproducible, and transferable workflow for UNSEEN.

This means that we aim for **anyone** to be able to assess **any** climate extreme event **anywhere** in the world!

UNSEEN-open was therefore developed with a focus on Copernicus SEAS5 forecasts, because it is an openly available, stable, homogeneous, global, high-resolution, large ensemble with continuous evaluation at ECMWF. We refer to section 4.2 of our paper for a discussion of other relevant datasets.

All code showing how UNSEEN data can be handled is documented on Jupyter notebooks. This means that some familiarity with python and R is (currently) required. Future further developments of tools and applications that do not require coding by the user themselves would be very interesting if time and funding allows!



1.1.2 Overview

Here we provide an overview of steps 3-5 for UNSEEN-open.

Retrieve

We use *global open* Copernicus C3S data: the seasonal prediction system SEAS5 and the reanlysis ERA5.

The functions to retrieve all forecasts (SEAS5) and reanalysis (ERA5) are `retrieve_SEAS5` and `retrieve_ERA5`. You can select the climate variable, the target month(s) and the area - for more explanation see *retrieve*.

[2]:
```
retrieve.retrieve_SEAS5(
    variables=['2m_temperature', '2m_dewpoint_temperature'],
    target_months=[3, 4, 5],
    area=[70, -11, 30, 120],
    years=np.arange(1981, 2021),
    folder='../Siberia_example/SEAS5/')
```

```
[3]: retrieve.retrieve_ERA5(variables=['2m_temperature', '2m_dewpoint_temperature'],
                            target_months=[3, 4, 5],
                            area=[70, -11, 30, 120],
                            folder='../Siberia_example/ERA5/')
```

Preprocess

In the preprocessing step, we first merge all downloaded files into one netcdf file. Then the rest of the preprocessing depends on the definition of the extreme event. For example, for the UK case study, we want to extract the UK average precipitation while for the Siberian heatwave we will just used the defined area to spatially average over. For the MAM season, we still need to take the seasonal average, while for the UK we already have the average February precipitation.

Read the docs on *preprocessing* for more info.

Evaluate

The evaluation step is important to assess whether the forecasts are realistic and consistent to the observations. There are three statistical tests available through the UNSEEN R package. See the *evaluation section* for more info.

Case studies

So what can we learn from UNSEEN-open?

Have a look at the examples!

1.2 Installation

Anaconda is used as package manager.

1.2.1 Python

For the retrieval and pre-processing of the data, the Copernicus Data Store (CDS) Python API cdsapi and xarray python packages are used. These can be installed using the environment provided in this directory.

```
conda env create -f environment.yml
```

This creates a conda environment called '*basic_analysis*'. The environment can be activated using:

```
conda activate basic_analysis
```

To get this environment as a kernel on jupyter, we need to install 'ipykernel' in the activated environment:

```
conda install -c anaconda ipykernel
```

And then install the environment:

```
python -m ipykernel install --user --name=basic_analysis
```

Hopefully, you will see the environment now as an available kernel!

1.2.2 R

For the evaluation, extreme value analysis and visualization, we use R ggplot and extRemes packages. The evaluation tests have been developed into an 'UNSEEN' R-package. These packages can be installed as follows:

```
[ ]: ### install regular packages
install.packages("extRemes") # for extreme value statistics
install.packages("ggplot2") # for plotting

### install GitHub packages (tag = commit, branch or release tag)
install.packages("devtools")
devtools::install_github("timokelder/UNSEEN") # for evaluation
```

1.3 Examples

In this project, UNSEEN-open is applied to assess two extreme events in 2020: February 2020 UK precipitation and the 2020 Siberian heatwave.

Launch in Binder

1.3.1 Siberian Heatwave

Prolonged heat events with an average temperature above 0 degrees over Siberia can have enormous impacts on the local environment, such as wildfires, invasion of pests and infrastructure failure, and on the global environment, through the release of greenhouse gasses during permafrost thawing.

The 2020 Siberian heatwave was a prolonged event that consistently broke monthly temperature the records. We show a gif of the monthly 2020 temperature rank within the observations from 1979-2020, see *this section* for details. - Rank 1 mean highest on record - Rank 2 means second highest - etc..

This attribution study by World Weather Attribution (WWA) has shown that the event was made much more likely (600x) because of human induced climate change but also that the event was a very rare event within our present climate.

Could such a thawing event be anticipated with UNSEEN?

With UNSEEN-open, we can assess whether extreme events like the Siberian heatwave have been forecasted already, i.e. whether we can anticipate such an event by exploiting all forecasts over the domain.

Retrieve data

The main functions to retrieve all forecasts (SEAS5) and reanalysis (ERA5) are `retrieve_SEAS5` and `retrieve_ERA5`. We want to download 2m temperature, for the March-May target months over the Siberian domain. By default, the hindcast years of 1981-2016 are downloaded for SEAS5. We include the years 1981-2020. The folder indicates where the files will be stored, in this case outside of the UNSEEN-open repository, in a 'Siberia_example' directory. For more explanation, see *retrieve*.

[1]:
```
import os
import sys
sys.path.insert(0, os.path.abspath('../../../'))
os.chdir(os.path.abspath('../../../'))

import src.cdsretrieve as retrieve
import src.preprocess as preprocess

import numpy as np
import xarray as xr
```

[2]:
```
retrieve.retrieve_SEAS5(
    variables=['2m_temperature', '2m_dewpoint_temperature'],
    target_months=[3, 4, 5],
    area=[70, -11, 30, 120],
    years=np.arange(1981, 2021),
    folder='../Siberia_example/SEAS5/')
```

[3]:
```
retrieve.retrieve_ERA5(variables=['2m_temperature', '2m_dewpoint_temperature'],
                       target_months=[3, 4, 5],
                       area=[70, -11, 30, 120],
                       folder='../Siberia_example/ERA5/')
```

Preprocess

In the preprocessing step, we first merge all downloaded files into one xarray dataset, then take the spatial average over the domain and a temporal average over the MAM season. Read the docs on *preprocessing* for more info.

[4]:
```
SEAS5_Siberia = preprocess.merge_SEAS5(folder = '../Siberia_example/SEAS5/', target_
↪months = [3,4,5])
```

```
Lead time: 02
1
12
```

And for ERA5:

[5]:
```
ERA5_Siberia = xr.open_mfdataset('../Siberia_example/ERA5/ERA5_????.nc',combine='by_
↪coords')
```

Then we calculate the day-in-month weighted seasonal average:

[6]:
```
SEAS5_Siberia_weighted = preprocess.season_mean(SEAS5_Siberia, years = 39)
ERA5_Siberia_weighted = preprocess.season_mean(ERA5_Siberia, years = 42)
```

And we select the 2m temperature, and take the average over a further specified domain. This is an area-weighed average, since grid cell area decreases with latitude, see *preprocess*.

[8]:
```
area_weights = np.cos(np.deg2rad(SEAS5_Siberia_weighted.latitude))

SEAS5_Siberia_events_zoomed = (
    SEAS5_Siberia_weighted['t2m'].sel(   # Select 2 metre temperature
        latitude=slice(70, 50),          # Select the latitudes
        longitude=slice(65, 120)).       # Select the longitude
    weighted(area_weights).              # Apply the weights
```

(continues on next page)

(continued from previous page)

```
    mean(['longitude', 'latitude']))     # and take the spatial average

SEAS5_Siberia_events_zoomed_df = SEAS5_Siberia_events_zoomed.rename('t2m').to_
↪dataframe() # weights remove the DataArray name, so I renamed the DaraArray after␣
↪applying the weight.
```

In this workflow, ERA5 and SEAS5 are on the same grid and hence have the same weights:

[9]:
```
area_weights_ERA = np.cos(np.deg2rad(ERA5_Siberia_weighted.latitude))
area_weights_ERA == area_weights
```

[9]:
```
<xarray.DataArray 'latitude' (latitude: 41)>
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True])
Coordinates:
  * latitude  (latitude) float32 70.0 69.0 68.0 67.0 ... 33.0 32.0 31.0 30.0
```

And here I take the spatial average 2m temperature for ERA5.

[42]:
```
ERA5_Siberia_events_zoomed = (
    ERA5_Siberia_weighted['t2m'].sel(  # Select 2 metre temperature
        latitude=slice(70, 50),        # Select the latitudes
        longitude=slice(65, 120)).     # Select the longitude
    weighted(area_weights).            # weights
    mean(['longitude', 'latitude']))   # Take the average

ERA5_Siberia_events_zoomed_df = ERA5_Siberia_events_zoomed.rename('t2m').to_
↪dataframe()
```

Evaluate

Note

From here onward we use R and not python!

We switch to R since we believe R has a better functionality in extreme value statistics.

Is the UNSEEN ensemble realistic?

To answer this question, we perform three statistical tests: independence, model stability and model fidelity tests. These statistical tests are available through the UNSEEN R package. See *evaluation* for more info.

[3]:
```
require(UNSEEN)
require(ggplot2)
require(ggpubr)
```

```
Loading required package: UNSEEN

Loading required package: ggplot2
```

(continues on next page)

(continued from previous page)

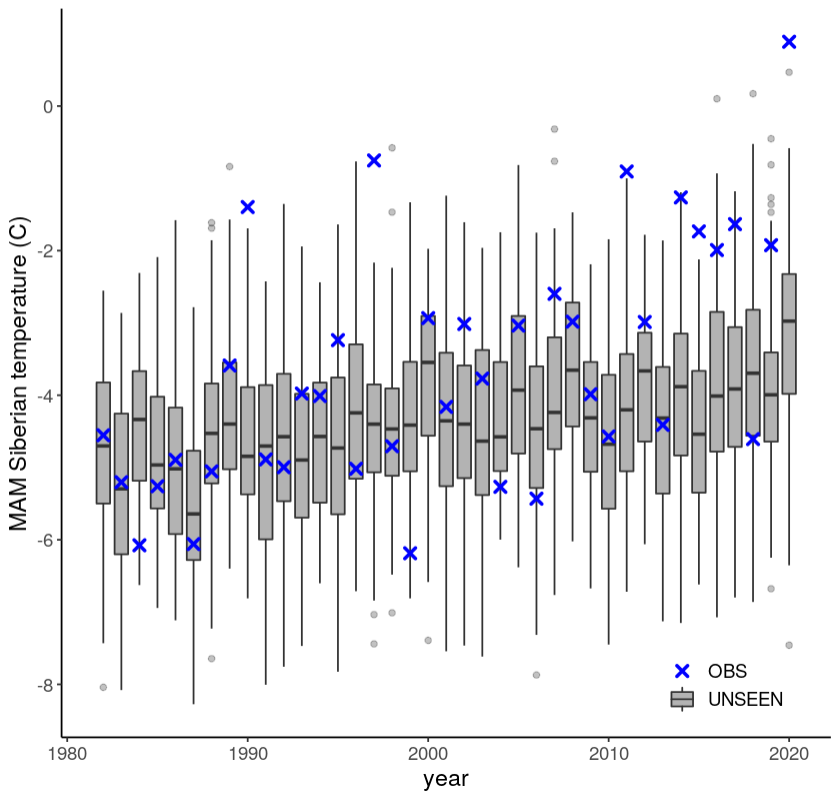
```
Warning message:
"replacing previous import 'vctrs::data_frame' by 'tibble::data_frame' when loading␣
↪'dplyr'"
Loading required package: ggpubr
```

Timeseries

We plot the timeseries of SEAS5 (UNSEEN) and ERA5 (OBS) for the the Siberian Heatwave.

[4]:
```
unseen_timeseries(
    ensemble = SEAS5_Siberia_events_zoomed_df,
    obs = ERA5_Siberia_events_zoomed[ERA5_Siberia_events_zoomed$year > 1981,],
    ensemble_yname = "t2m",
    ensemble_xname = "year",
    obs_yname = "t2m",
    obs_xname = "year",
    ylab = "MAM Siberian temperature (C)") +
theme(text = element_text(size = 14)) #This is to increase the figure font
```

```
Warning message:
"Removed 2756 rows containing non-finite values (stat_boxplot)."
```



The timeseries consist of **hindcast (years 1982-2016)** and **archived forecasts (years 2017-2020)**. The datasets are slightly different: the hindcasts contains 25 members whereas operational forecasts contain 51 members, the native resolution is different and the dataset from which the forecasts are initialized is different.

For the evaluation of the UNSEEN ensemble we want to only use the SEAS5 hindcasts for a consistent dataset. Note, 2017 is not used in either the hindcast nor the operational dataset, since it contains forecasts both initialized in 2016 (hindcast) and 2017 (forecast), see *retrieve*. We split SEAS5 into hindcast and operational forecasts:

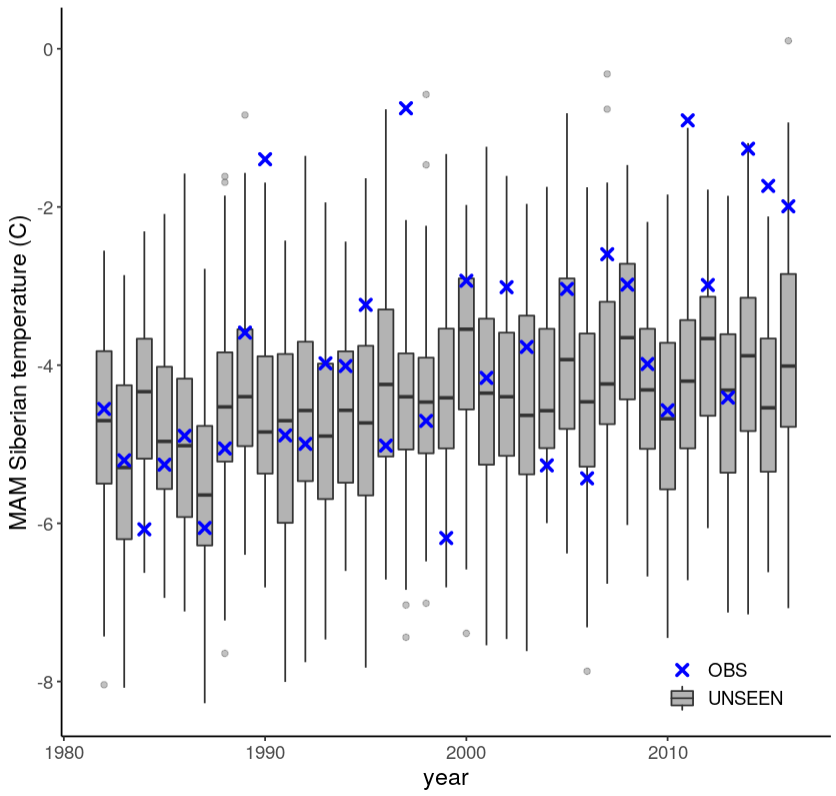
[5]:
```
SEAS5_Siberia_events_zoomed_hindcast <- SEAS5_Siberia_events_zoomed_df[
    SEAS5_Siberia_events_zoomed_df$year < 2017 &
    SEAS5_Siberia_events_zoomed_df$number < 25,]

SEAS5_Siberia_events_zoomed_forecasts <- SEAS5_Siberia_events_zoomed_df[
    SEAS5_Siberia_events_zoomed_df$year > 2017,]
```

And we select the same years for ERA5.

[6]:
```
ERA5_Siberia_events_zoomed_hindcast <- ERA5_Siberia_events_zoomed[
    ERA5_Siberia_events_zoomed$year < 2017 &
    ERA5_Siberia_events_zoomed$year > 1981,]
```

Which results in the following timeseries:

[7]:
```
unseen_timeseries(
    ensemble = SEAS5_Siberia_events_zoomed_hindcast,
    obs = ERA5_Siberia_events_zoomed_hindcast,
    ensemble_yname = "t2m",
    ensemble_xname = "year",
    obs_yname = "t2m",
    obs_xname = "year",
    ylab = "MAM Siberian temperature (C)") +
    theme(text = element_text(size = 14))
```



Evaluation tests

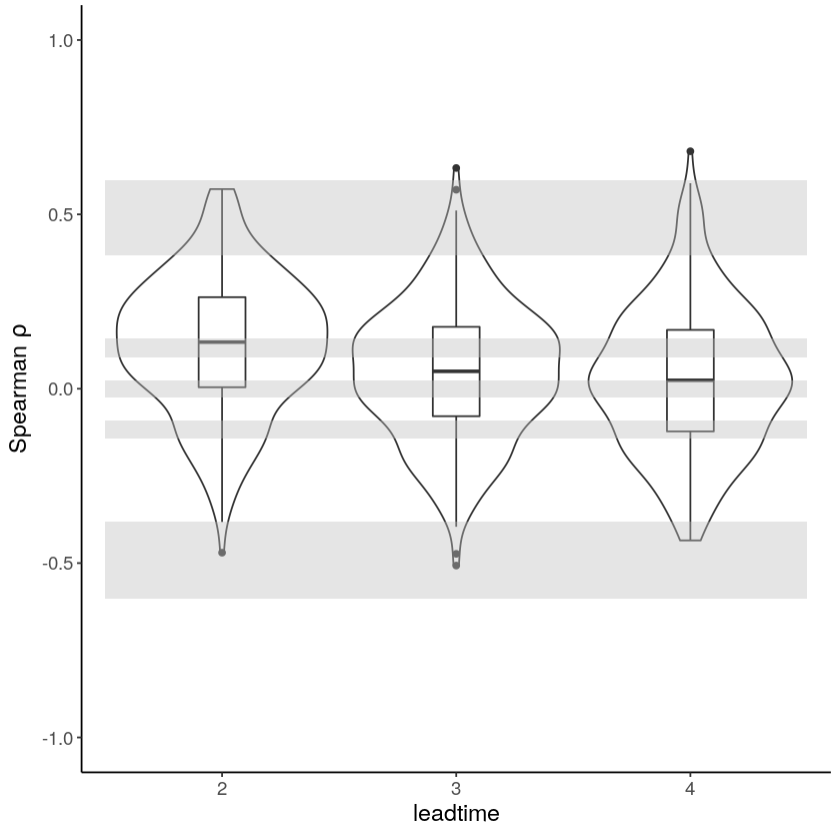
With the hindcast dataset we evaluate the independence, stability and fidelity. Here, we plot the results for the fidelity test, for more detail on the other tests see the *evaluation section*.

The fidelity test shows us how consistent the model simulations of UNSEEN (SEAS5) are with the observed (ERA5). The UNSEEN dataset is much larger than the observed – hence they cannot simply be compared. For example, what if we had faced a few more or a few less heatwaves purely by chance?

This would influence the observed mean, but not so much influence the UNSEEN ensemble because of the large data sample. Therefore we express the UNSEEN ensemble as a range of plausible means, for data samples of the same length as the observed. We do the same for higher order statistical moments.

[7]:
```
independence_test(
    ensemble = SEAS5_Siberia_events_zoomed_hindcast,
    n_lds = 3,
    var_name = "t2m",
    detrend = TRUE
) +
    theme(text = element_text(size = 14))
```

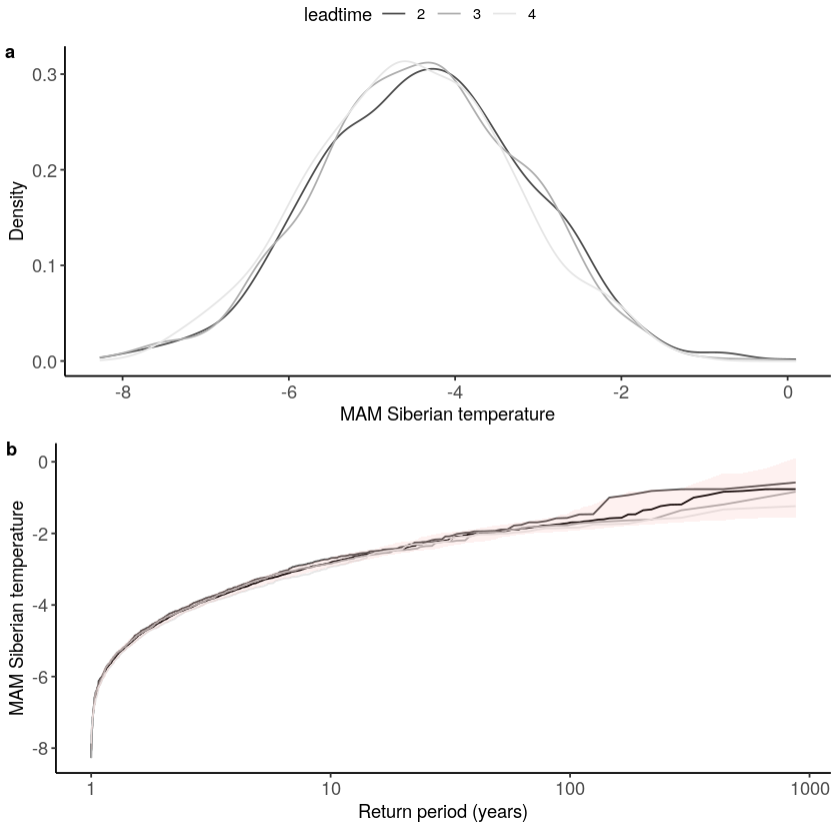
```
Warning message:
"Removed 975 rows containing non-finite values (stat_ydensity)."
Warning message:
"Removed 975 rows containing non-finite values (stat_boxplot)."
```



Is the model stable over leadtimes?

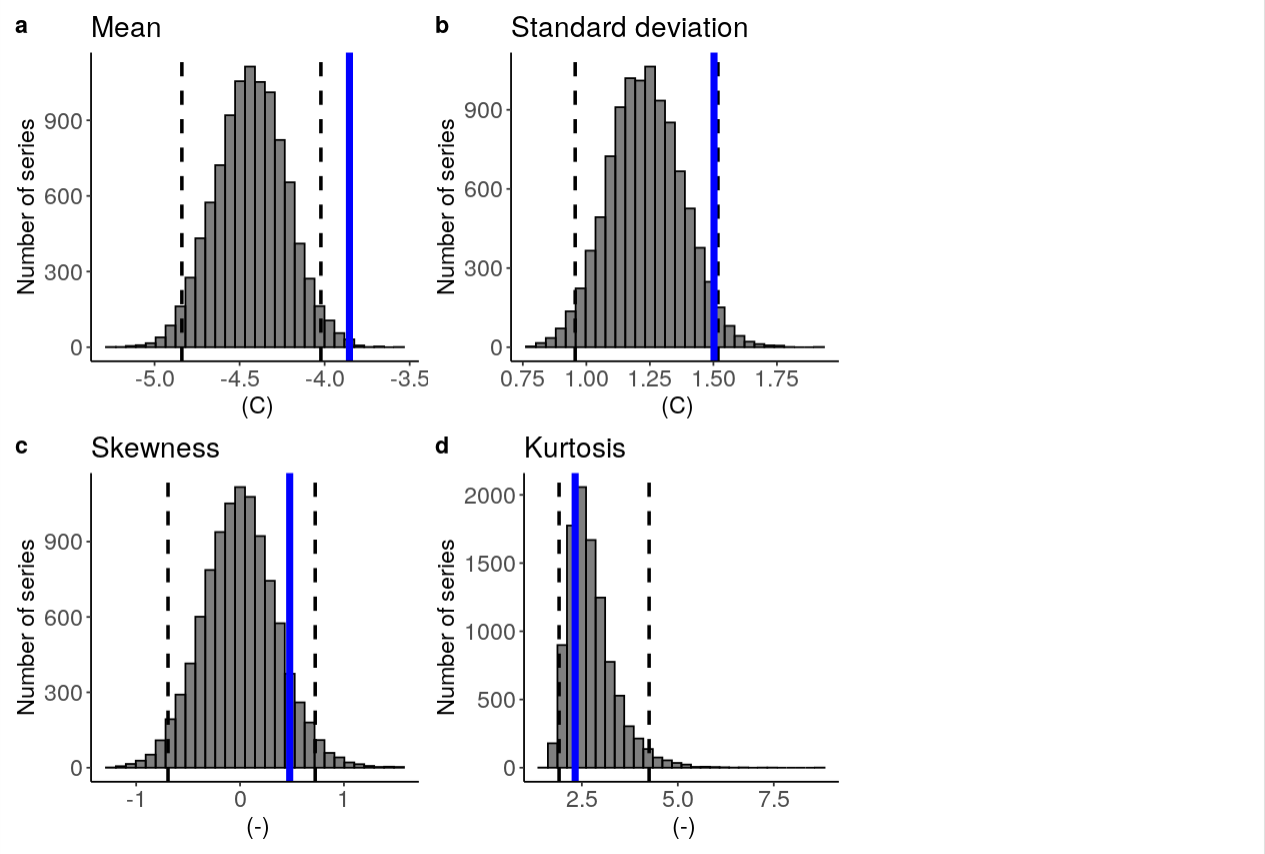
[8]:
```
stability_test(
    ensemble = SEAS5_Siberia_events_zoomed_hindcast,
    lab = 'MAM Siberian temperature',
    var_name = 't2m'
)
```

```
Warning message:
"Removed 2 row(s) containing missing values (geom_path)."
```



Is the model consistent with ERA5?

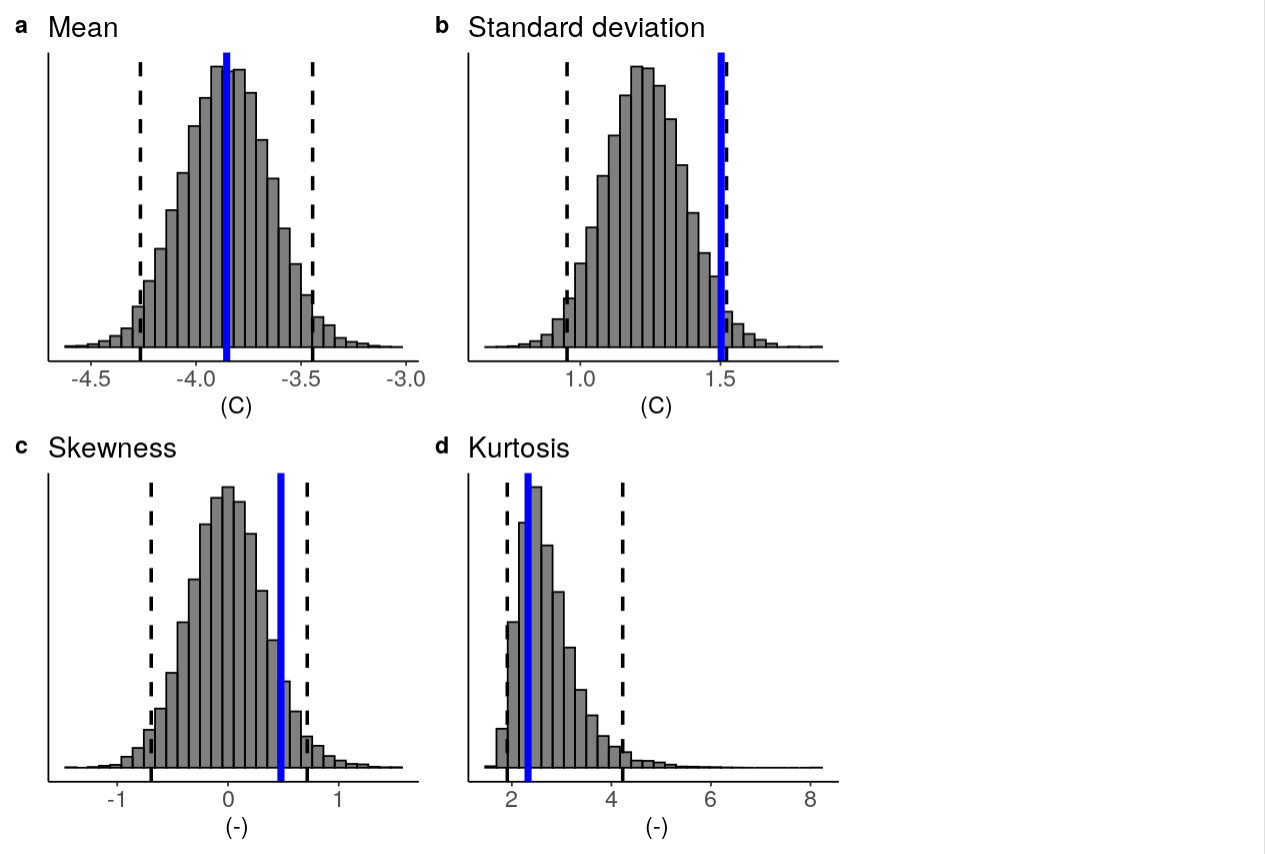
[9]:
```
fidelity_test(
    obs = ERA5_Siberia_events_zoomed_hindcast$t2m,
    ensemble = SEAS5_Siberia_events_zoomed_hindcast$t2m,
    units = 'C',
    biascor = FALSE,
    fontsize = 14
)
```



The fidelity test shows that the mean of the UNSEEN ensemble is too low compared to the observed – the blue line falls outside of the model range in a. To correct for this low bias, we can apply an additive bias correction, which only corrects the mean of the simulations.

Lets apply the additive biascor:

[11]:
```
obs <- ERA5_Siberia_events_zoomed_hindcast$t2m
ensemble <- SEAS5_Siberia_events_zoomed_hindcast$t2m
ensemble_biascor <- ensemble + (mean(obs) - mean(ensemble))

fidelity_test_biascor <- fidelity_test(
    obs = obs,
    ensemble = ensemble_biascor,
    units = 'C',
    ylab = '',
    yticks = FALSE,
    biascor = FALSE,
    fontsize = 14
)

fidelity_test_biascor
# ggsave(fidelity_test_biascor,height = 90, width = 90, units = 'mm', filename =
↪"graphs/Siberia_biascor.pdf")
```



This shows us what we expected: the mean bias is corrected because the model simulations are shifted up (the blue line is still the same, the axis has just shifted along with the histogram), but the other statistical moments are the same.

Illustrate

So could thawing events (with an average temperature above 0 degrees) have been anticipated?

Here we create a bias adjusted dataframe:

[13]:
```
SEAS5_Siberia_events_zoomed_df_bc <- SEAS5_Siberia_events_zoomed_df
SEAS5_Siberia_events_zoomed_df_bc['t2m'] <- SEAS5_Siberia_events_zoomed_df_bc['t2m']␣
↪+ (mean(obs) - mean(ensemble))

str(SEAS5_Siberia_events_zoomed_df_bc)
```

```
'data.frame':   5967 obs. of  4 variables:
 $ year    : int  1982 1982 1982 1982 1982 1982 1982 1982 1982 1982 ...
 $ leadtime: int  2 2 2 2 2 2 2 2 2 2 ...
 $ number  : int  0 1 2 3 4 5 6 7 8 9 ...
 $ t2m     : num  -3.16 -5.11 -3.64 -5.81 -2.93 ...
```

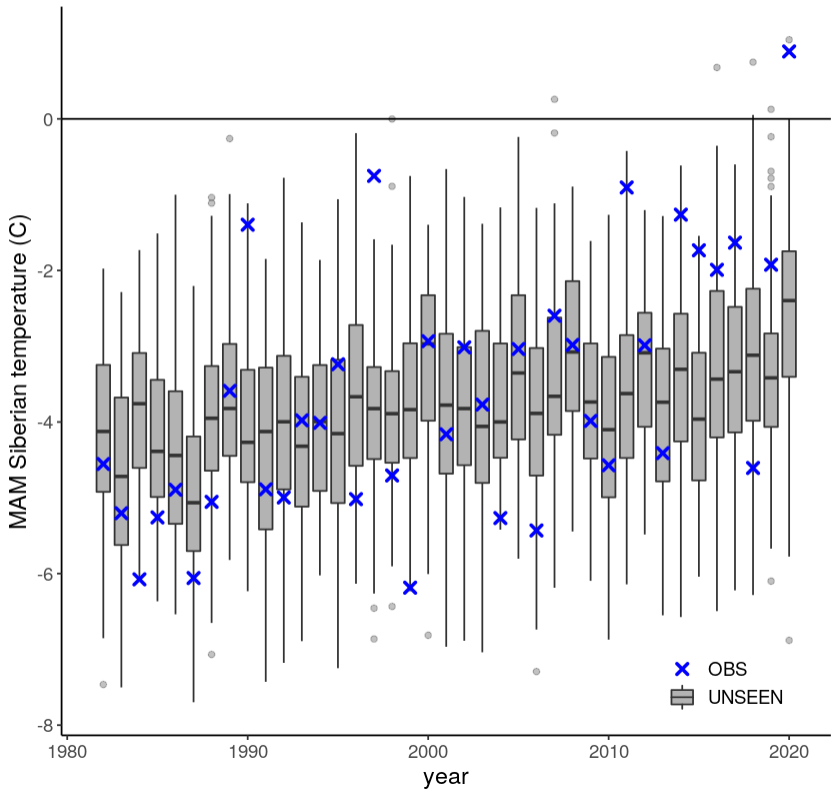
[14]:
```
unseen_timeseries(
    ensemble = SEAS5_Siberia_events_zoomed_df_bc,
    obs = ERA5_Siberia_events_zoomed[ERA5_Siberia_events_zoomed$year > 1981,],
    ensemble_yname = "t2m",
    ensemble_xname = "year",
    obs_yname = "t2m",
```

(continues on next page)

(continued from previous page)

```
    obs_xname = "year",
    ylab = "MAM Siberian temperature (C)") +
theme(text = element_text(size = 14)) + #This is to increase the figure font
geom_hline(yintercept = 0) #+
# ggsave(height = 90, width = 90, units = 'mm', filename = "graphs/Siberia_timeseries_
↪biascor.pdf")
```

```
Warning message:
"Removed 2756 rows containing non-finite values (stat_boxplot)."
```



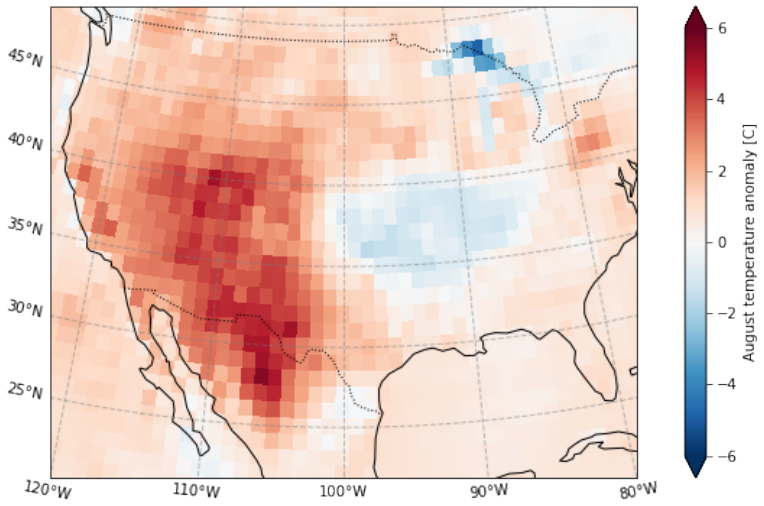
Applications:

With UNSEEN-open, we can: 1. *Assess the drivers* of the most severe events. The 2020 event seemed to be caused by a very anomalous Indian Ocean Dipole (IOD). What can we learn from the thawing events within UNSEEN? To what extent are these also driven by an anomalous IOD or are there other drivers of such severe heat events? 2. Perform *nonstationary analysis* in rare extremes, such as the 100-year event. There seems to be a trend over the hindcast period in the severe heatwaves. We can perform non-stationary analysis to estimate the change in the magnitude and frequency of the heatwaves and, if we find a change, we could explore the drivers of this change. 3. *Evaluate forecasts*. Since we are using seasonal forecasts in this setup, we could explore the forecast skill in simulating heatwaves over Siberia.

Launch in Binder

1.3.2 California fires

In August 2020 in California, wildfires have burned more than a million acres of land. The wildfires coinciding with record high temperature anomalies, see *August 2020 temperature anomaly*.



In this example, we evaluate the UNSEEN ensemble and show that there is a clear trend in temperature extremes over the last decades.

Retrieve data

The main functions to retrieve all forecasts (SEAS5) and reanalysis (ERA5) are `retrieve_SEAS5` and `retrieve_ERA5`. We want to download 2m temperature for August over California. By default, the hindcast years of 1981-2016 are downloaded for SEAS5. We include the years 1981-2020. The folder indicates where the files will be stored, in this case outside of the UNSEEN-open repository, in a 'California_example' directory. For more explanation, see *retrieve*.

[1]:
```
import os
import sys
sys.path.insert(0, os.path.abspath('../../../'))
os.chdir(os.path.abspath('../../../'))

import src.cdsretrieve as retrieve
import src.preprocess as preprocess

import numpy as np
import xarray as xr
```

[2]:
```
retrieve.retrieve_SEAS5(
    variables=['2m_temperature', '2m_dewpoint_temperature'],
    target_months=[8],
    area=[70, -130, 20, -70],
    years=np.arange(1981, 2021),
    folder='../California_example/SEAS5/')
```

[3]:
```
retrieve.retrieve_ERA5(variables=['2m_temperature', '2m_dewpoint_temperature'],
                       target_months=[8],
                       area=[70, -130, 20, -70],
                       folder='../California_example/ERA5/')
```

Preprocess

In the preprocessing step, we first merge all downloaded files into one xarray dataset, then take the spatial average over the domain and a temporal average over the MAM season. Read the docs on *preprocessing* for more info.

[4]:
```
SEAS5_California = preprocess.merge_SEAS5(folder ='../California_example/SEAS5/',␣
↪target_months = [8])
```

```
Lead time: 07
6
5
4
3
```

And for ERA5:

[5]:
```
ERA5_California = xr.open_mfdataset('../California_example/ERA5/ERA5_????.nc',combine=
↪'by_coords')
```

We calculate the standardized anomaly of the 2020 event and select the 2m temperature over the region where 2 standard deviations from the 1979-2010 average was exceeded, *see this page*. This is an area-weighed average, since grid cell area decreases with latitude, see *preprocess*.

[6]:
```
ERA5_anomaly = ERA5_California['t2m'] - ERA5_California['t2m'].sel(time=slice('1979',
↪'2010')).mean('time')
ERA5_sd_anomaly = ERA5_anomaly / ERA5_California['t2m'].sel(time=slice('1979','2010
↪')).std('time')
```

We use a land-sea mask to select land-only gridcells:

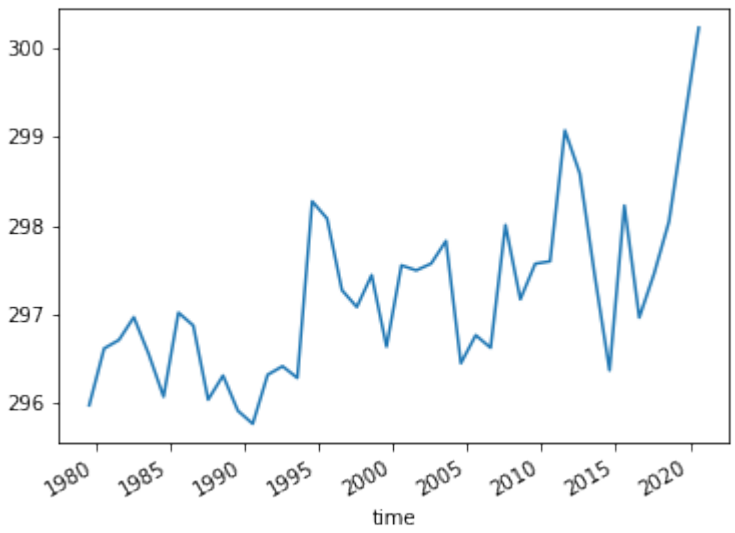
[7]:
```
LSMask = xr.open_dataset('../California_example/ERA_landsea_mask.nc')
# convert the longitude from 0:360 to -180:180
LSMask['longitude'] = (((LSMask['longitude'] + 180) % 360) - 180)
```

[8]:
```
area_weights = np.cos(np.deg2rad(ERA5_sd_anomaly.latitude))

ERA5_California_events = (
    ERA5_California['t2m'].sel(  # Select 2 metre temperature
        longitude = slice(-125,-100),     # Select the longitude
        latitude = slice(45,20)).         # And the latitude
    where(ERA5_sd_anomaly.sel(time = '2020').squeeze('time') > 2). ##Mask the region␣
↪where 2020 sd >2.
    where(LSMask['lsm'].sel(time = '1979').squeeze('time') > 0.5). #Select land-only␣
↪gridcells
    weighted(area_weights).
    mean(['longitude', 'latitude']) #And take the mean
)
```

Plot the August temperatures over the defined California domain:

[9]:
```
ERA5_California_events.plot()
```

[9]:
```
[<matplotlib.lines.Line2D at 0x7f99b808f8b0>]
```



Select the same domain for SEAS5 and extract the events.

[10]:
```
SEAS5_California_events = (
    SEAS5_California['t2m'].sel(
        longitude = slice(-125,-100),     # Select the longitude
        latitude = slice(45,20)).         # And the latitude
    where(ERA5_sd_anomaly.sel(time = '2020').squeeze('time') > 2). #Mask the region␣
↪where 2020 sd >2.
    where(LSMask['lsm'].sel(time = '1979').squeeze('time') > 0.5). #Select land-only␣
↪gridcells
    weighted(area_weights).
    mean(['longitude', 'latitude']))
```

And here we store the data in the Data section so the rest of the analysis in R can be reproduced.

[11]:
```
SEAS5_California_events.rename('t2m').to_dataframe().to_csv('Data/SEAS5_California_
↪events.csv')
ERA5_California_events.rename('t2m').to_dataframe().to_csv('Data/ERA5_California_
↪events.csv')
```

Evaluate

Note

From here onward we use R and not python!

We switch to R since we believe R has a better functionality in extreme value statistics.

Is the UNSEEN ensemble realistic?

To answer this question, we perform three statistical tests: independence, model stability and model fidelity tests.

These statistical tests are available through the UNSEEN R package. See *evaluation* for more info.

[4]:
```
require(UNSEEN)
require(ggplot2)
```

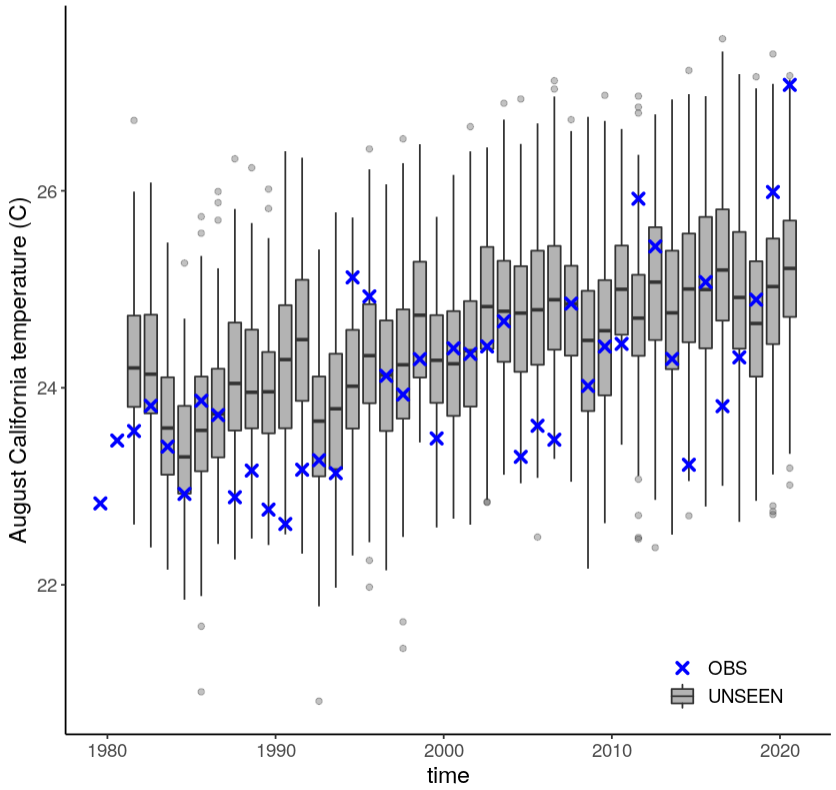
```
Loading required package: UNSEEN

Loading required package: ggplot2

Warning message:
"replacing previous import 'vctrs::data_frame' by 'tibble::data_frame' when loading␣
↪'dplyr'"
```

Timeseries

We plot the timeseries of SEAS5 (UNSEEN) and ERA5 (OBS) for the the Siberian Heatwave.

[5]:
```
timeseries = unseen_timeseries(
    ensemble = SEAS5_California_events,
    obs = ERA5_California_events,
    ensemble_yname = "t2m",
    ensemble_xname = "time",
    obs_yname = "t2m",
    obs_xname = "time",
    ylab = "August California temperature (C)")

timeseries + theme(text = element_text(size = 14))
```

```
Warning message:
"Removed 4680 rows containing non-finite values (stat_boxplot)."
```



The timeseries consist of **hindcast (years 1982-2016)** and **archived forecasts (years 2017-2020)**. The datasets are slightly different: the hindcasts contains 25 members whereas operational forecasts contain 51 members, the native resolution is different and the dataset from which the forecasts are initialized is different.

For the evaluation of the UNSEEN ensemble we want to only use the SEAS5 hindcasts for a consistent dataset. Note, 2017 is not used in either the hindcast nor the operational dataset, since it contains forecasts both initialized in 2016 (hindcast) and 2017 (forecast), see *retrieve*. We split SEAS5 into hindcast and operational forecasts:

[6]:
```
SEAS5_California_events_hindcast <- SEAS5_California_events[
    SEAS5_California_events$time < '2017-02-01' &
    SEAS5_California_events$number < 25,]

SEAS5_California_events_forecasts <- SEAS5_California_events[
    SEAS5_California_events$time > '2017-02-01',]
```

And we select the same years for ERA5.

[7]:
```
ERA5_California_events_hindcast <- ERA5_California_events[
    ERA5_California_events$time > '1981-02-01' &
    ERA5_California_events$time < '2017-02-01',]
```

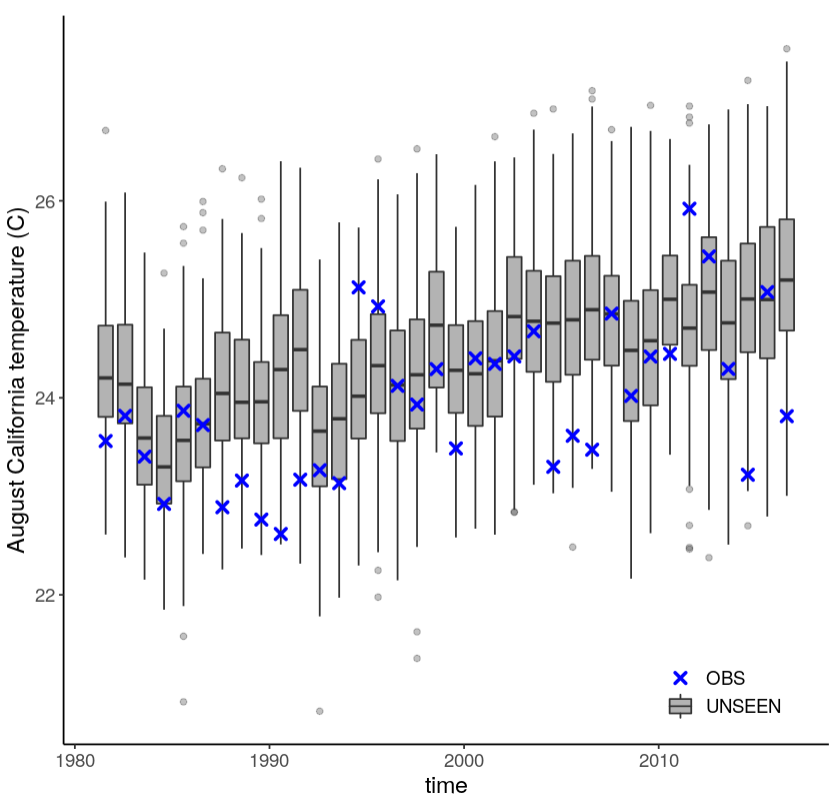
Which results in the following timeseries:

[8]:
```
unseen_timeseries(
    ensemble = SEAS5_California_events_hindcast,
    obs = ERA5_California_events_hindcast,
    ensemble_yname = "t2m",
    ensemble_xname = "time",
```

(continues on next page)

(continued from previous page)

```
    obs_yname = "t2m",
    obs_xname = "time",
    ylab = "August California temperature (C)") +
theme(text = element_text(size = 14))
```

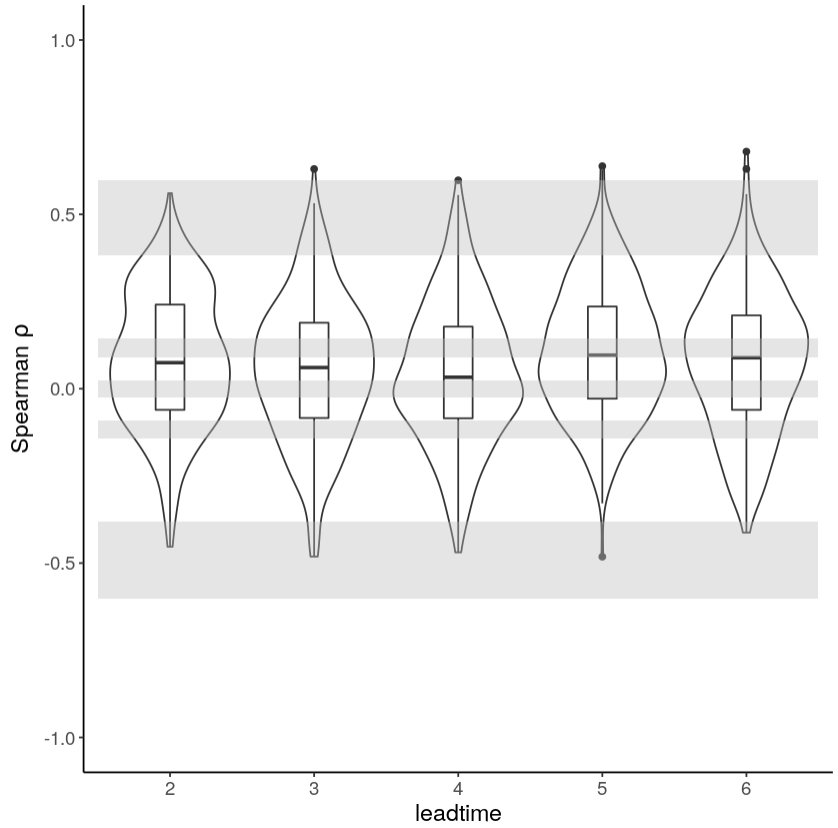


Evaluation tests

With the hindcast dataset we evaluate the independence, stability and fidelity. Here, we plot the results for the fidelity test, for more detail on the other tests see the *evaluation section*.

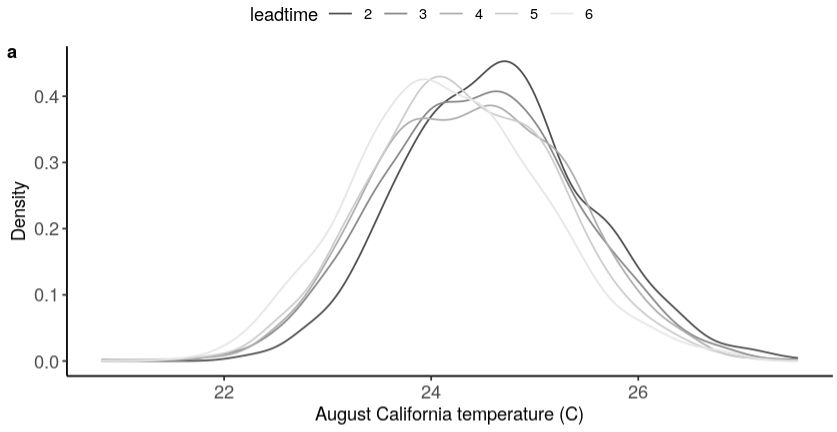
[9]:
```
Independence_California = independence_test(
    ensemble = SEAS5_California_events_hindcast,
    var_name = "t2m"
    )

Independence_California +
    theme(text = element_text(size = 14))
```

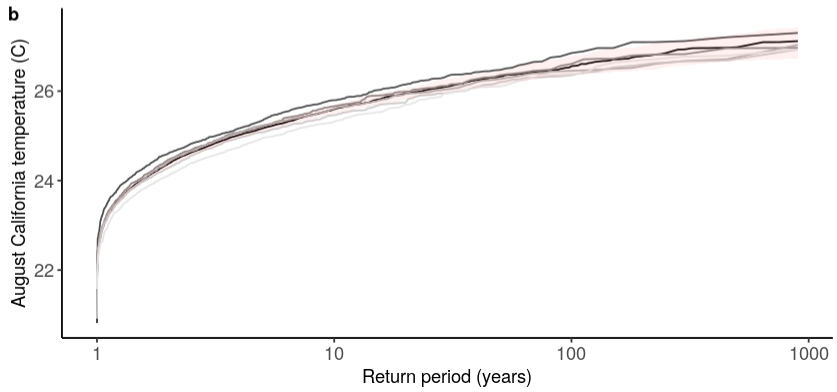
```
Warning message:
"Removed 1625 rows containing non-finite values (stat_ydensity)."
Warning message:
"Removed 1625 rows containing non-finite values (stat_boxplot)."
```



[10]:
```
Stability_California = stability_test(
        ensemble = SEAS5_California_events_hindcast,
        lab = 'August California temperature (C)',
        var_name = 't2m'
    )
Stability_California
```

```
Warning message:
“Removed 4 row(s) containing missing values (geom_path).”
```





[11]:
```
Stability_California = stability_test(
        ensemble = SEAS5_California_events_hindcast,
        lab = 'August temperature (C)',
        var_name = 't2m',
        fontsize = 10

    )
# ggsave(Stability_California,height = 120, width = 120, units = 'mm', filename =
↪"graphs/California_stability.pdf")
```

```
Warning message:
"Removed 4 row(s) containing missing values (geom_path)."
```

The fidelity test shows us how consistent the model simulations of UNSEEN (SEAS5) are with the observed (ERA5). The UNSEEN dataset is much larger than the observed – hence they cannot simply be compared. For example, what if we had faced a few more or a few less heatwaves purely by chance?

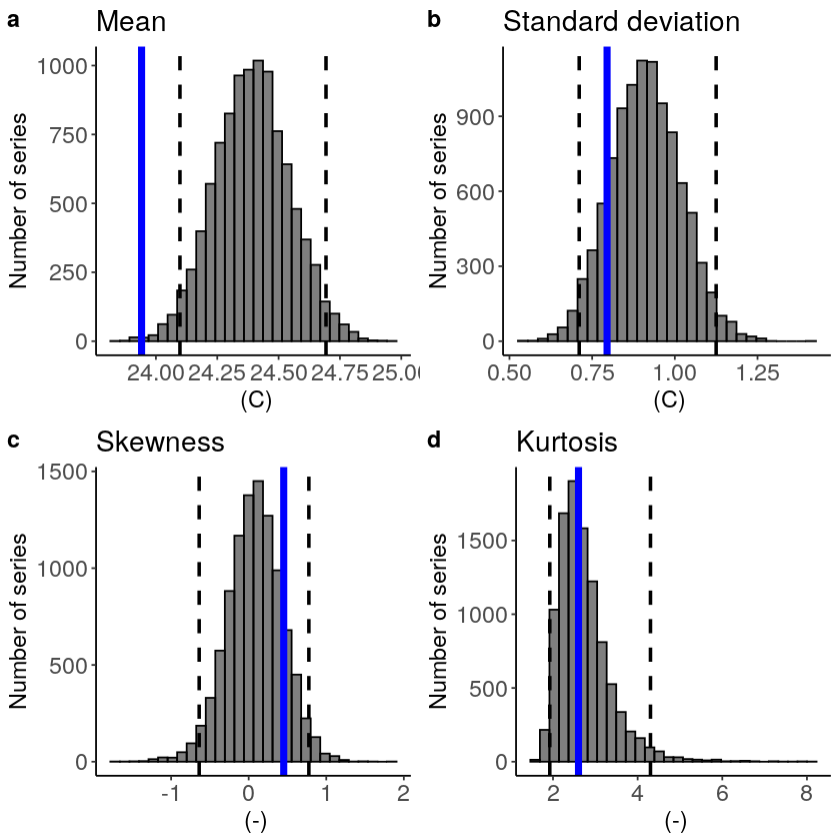
This would influence the observed mean, but not so much influence the UNSEEN ensemble because of the large data sample. Therefore we express the UNSEEN ensemble as a range of plausible means, for data samples of the same length as the observed. We do the same for higher order statistical moments.

[12]:
```
Fidelity_California = fidelity_test(
    obs = ERA5_California_events_hindcast$t2m,
    ensemble = SEAS5_California_events_hindcast$t2m,
    units = 'C',
    biascor = FALSE,
    fontsize = 14
)
```

(continues on next page)

(continued from previous page)

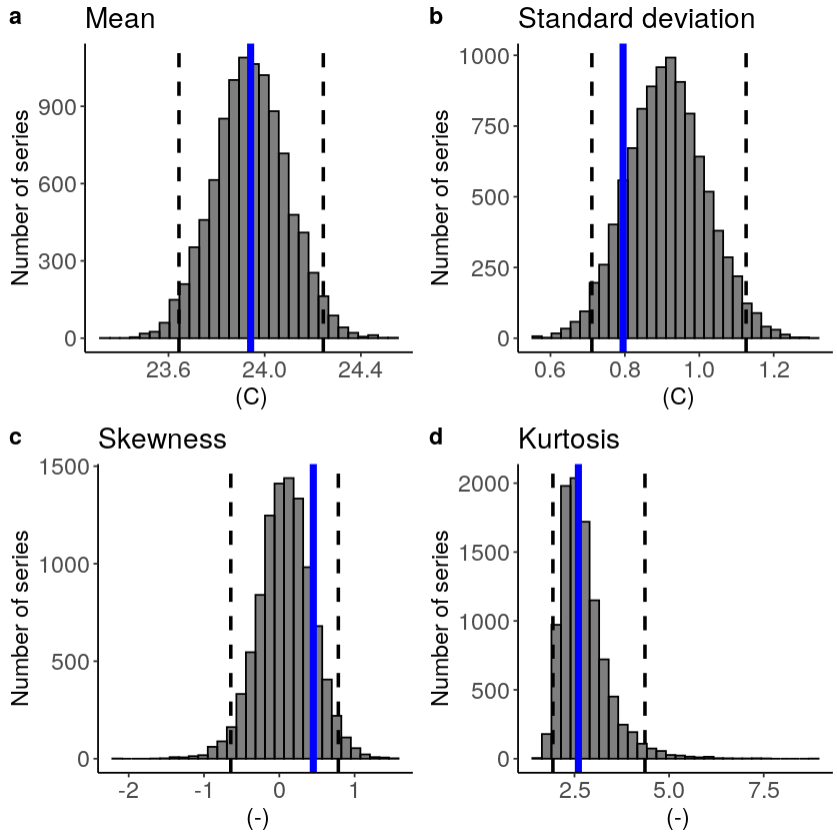
```
Fidelity_California
```



The fidelity test shows that the mean of the UNSEEN ensemble is too low compared to the observed – the blue line falls outside of the model range in a. To correct for this low bias, we can apply an additive bias correction, which only corrects the mean of the simulations.

Lets apply the additive biascor:

[10]:
```
obs = ERA5_California_events_hindcast$t2m
ensemble = SEAS5_California_events_hindcast$t2m
ensemble_biascor = ensemble + (mean(obs) - mean(ensemble))

fidelity_test(
    obs = obs,
    ensemble = ensemble_biascor,
    units = 'C',
    biascor = FALSE,
    fontsize = 14
)
```



This shows us what we expected: the mean bias is corrected because the model simulations are shifted up (the blue line is still the same, the axis has just shifted along with the histogram), but the other statistical moments are the same.

Publication-ready plots

We combine the timeseries and the three evaluation plots in one plot for the manuscript. We want the font size to be 10 for all plots and we need to adjust the panel labels for the stability and fidelity plots. For the fidelity plot we also remove redundant ylabels and yticks.

[14]:
```
timeseries_font10 = timeseries + theme(text = element_text(size = 10))
Independence_font10 = Independence_California + theme(text = element_text(size = 10))
Stability_font10 = stability_test(
    ensemble = SEAS5_California_events_hindcast,
    lab = 'August temperature (C)',
    var_name = 't2m',
    fontsize = 10,
    panel_labels = c("c", "d")
)
Fidelity_font10 = fidelity_test(
    obs = ERA5_California_events_hindcast$t2m,
    ensemble = SEAS5_California_events_hindcast$t2m,
    ylab = '',
    yticks = FALSE,
    units = 'C',
    biascor = FALSE,
    fontsize = 10,
```

(continues on next page)

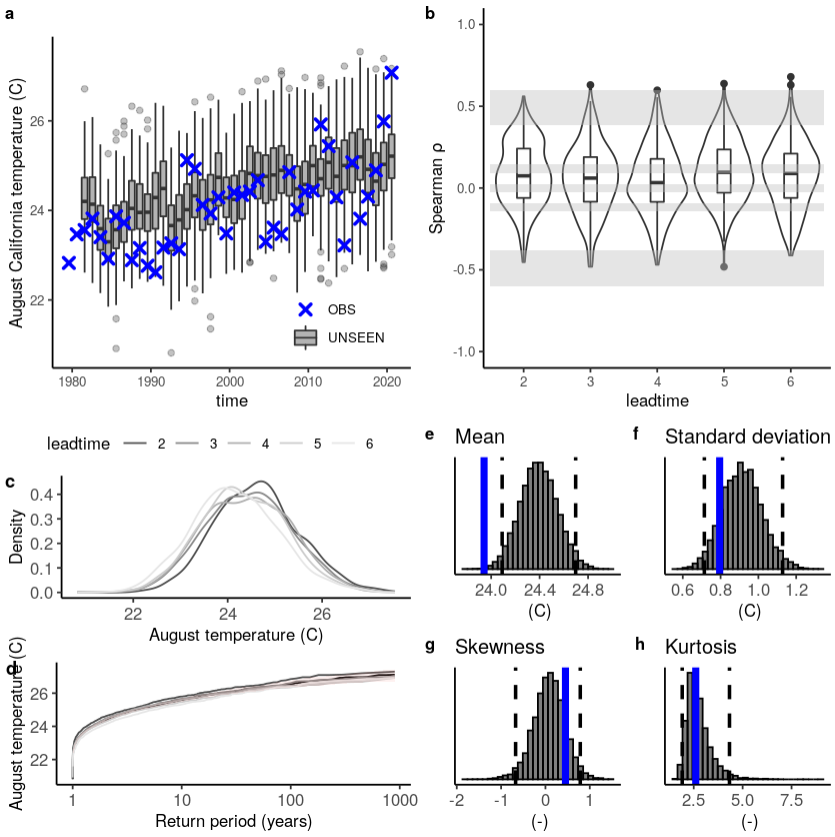
(continued from previous page)

```
    panel_labels = c("e", "f", "g", "h")
)
```

```
Warning message:
"Removed 4 row(s) containing missing values (geom_path)."
```

[15]:
```
Evaluations = ggpubr::ggarrange(timeseries_font10,
                                Independence_font10,
                                Stability_font10,
                                Fidelity_font10,
                                labels = c("a","b", "", ""),
                                font.label = list(size = 10,
                                                  color = "black",
                                                  face = "bold",
                                                  family = NULL),
                                ncol = 2,
                                nrow = 2)
Evaluations
# ggsave(Evaluations,height = 180, width = 180, units = 'mm', filename = "graphs/
↪California_evaluation_test2.pdf")
```

```
Warning message:
"Removed 4680 rows containing non-finite values (stat_boxplot)."
Warning message:
"Removed 1625 rows containing non-finite values (stat_ydensity)."
Warning message:
"Removed 1625 rows containing non-finite values (stat_boxplot)."
```



Illustrate

Was there a trend in the temperature extremes over the last decades? Let's investigate!

First, we are loading the required extRemes package:

[9]:
```
require('extRemes')
```

```
Loading required package: extRemes

Loading required package: Lmoments

Loading required package: distillery


Attaching package: 'extRemes'


The following objects are masked from 'package:stats':

    qqnorm, qqplot
```

We also source some R code to make the unseen-trends plots. These functions were written for this case study and we cannot ensure robustness to other case studies.

[24]:
```
source('src/evt_plot.r')
```

We use ERA5 events from 1981 to match the starting date of SEAS5, which we call 'obs'. In addition, we use the bias corrected UNSEEN ensemble with ld 6 removed. We remove the first two years from ERA5 and we remove lead time 6 from the SEAS5 ensemble:

[10]:
```
obs <- ERA5_California_events[
    ERA5_California_events$time > '1981-02-01',]

UNSEEN_bc <- SEAS5_California_events[SEAS5_California_events$leadtime < 6 &
                                     SEAS5_California_events$number < 25,]
```

And then we correct the SEAS5 temperature bias using a mean adjustment calculated over the hindcast period.

[11]:
```
UNSEEN_bc$t2m <- (UNSEEN_bc$t2m +
                  mean(ERA5_California_events_hindcast$t2m) - mean(SEAS5_California_
↪events_hindcast$t2m)
                                 )
str(UNSEEN_bc)
```

```
'data.frame':   4000 obs. of  4 variables:
 $ leadtime: int  2 2 2 2 2 2 2 2 2 2 ...
 $ time    : Date, format: "1981-08-01" "1981-08-01" ...
 $ number  : int  0 1 2 3 4 5 6 7 8 9 ...
 $ t2m     : num  23 24.8 23.2 23.9 24.6 ...
```

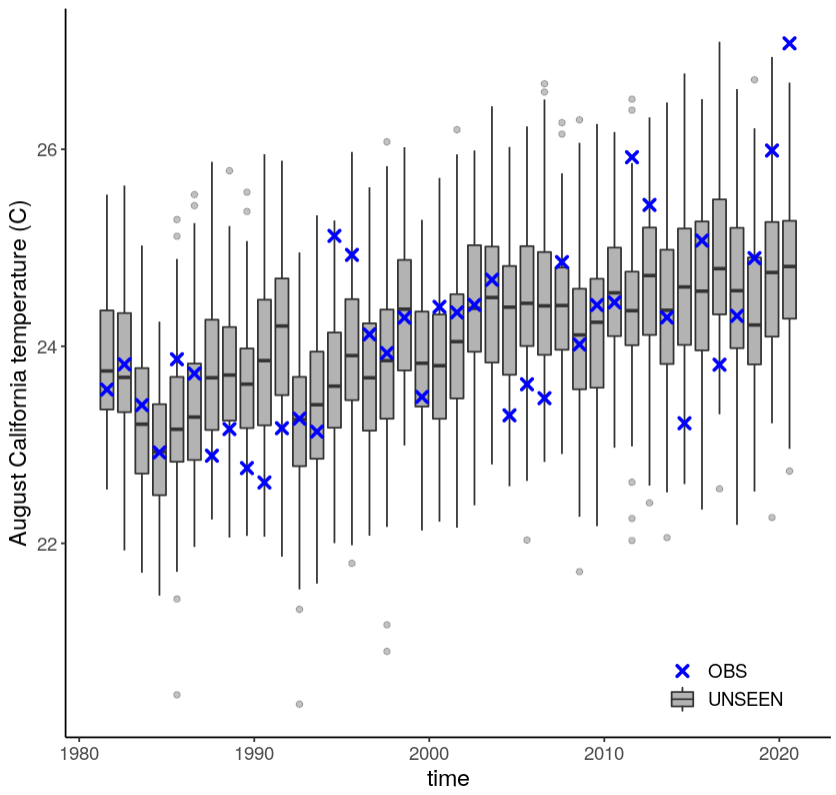
Lets plot the data to see whats going on

[12]:
```
timeseries = unseen_timeseries(
    ensemble = UNSEEN_bc,
    obs = obs,
    ensemble_yname = "t2m",
```

(continues on next page)

(continued from previous page)

```
    ensemble_xname = "time",
    obs_yname = "t2m",
    obs_xname = "time",
    ylab = "August California temperature (C)")

timeseries + theme(text = element_text(size = 14))
```



We apply extreme value theory to analyze the likelihood and trend of the temperature extremes. There are different extreme value distributions that can be used to fit to the data. First, we fit a stationary Gumbel and a GEV distribution (including shape parameter) to the observed extremes, which shows that GEV better describes the data with a p-value of 0.0001 using the LR-test. Then we fit a nonstationary GEV distribution to the observed temperatures and show that this best fits the data with a very small p-value of 3.633e-05 as compared to the stationary distribution (much below 0.05 based on 5% significance with the likelihood ratio test).

[13]:
```
## Fit stationary distributions
fit_obs_Gumbel <- fevd(x = obs$t2m,
                       type = "Gumbel"
                      )
fit_obs_GEV <- fevd(x = obs$t2m,
                       type = "GEV"
                      )
## And the nonstationary distribution
fit_obs_GEV_nonstat <- fevd(x = obs$t2m,
                            type = "GEV",
                            location.fun = ~ c(1:length(obs$time)), ##Fitting the gev␣
↪with a location and scale parameter linearly correlated to the covariate (years)
                            scale.fun = ~ c(1:length(obs$time)),
```

(continues on next page)

(continued from previous page)

```
                            use.phi = TRUE
                           )
#And test the fit
##1. Stationary Gumbel vs stationary GEV
lr.test(fit_obs_Gumbel, fit_obs_GEV_nonstat)
##2. Stationary GEV vs Nonstationary GEV
lr.test(fit_obs_GEV, fit_obs_GEV_nonstat)
```

```
        Likelihood-ratio Test

data:  obs$t2mobs$t2m
Likelihood-ratio = 20.446, chi-square critical value = 7.8147, alpha =
0.0500, Degrees of Freedom = 3.0000, p-value = 0.0001372
alternative hypothesis: greater
```

```
        Likelihood-ratio Test

data:  obs$t2mobs$t2m
Likelihood-ratio = 20.446, chi-square critical value = 5.9915, alpha =
0.0500, Degrees of Freedom = 2.0000, p-value = 3.633e-05
alternative hypothesis: greater
```

For the unseen ensemble this analysis is slightly more complicated since we need a covariate that has the same length as the ensemble:

[14]:
```
#Create the ensemble covariate
year_vector = as.integer(format(UNSEEN_bc$time, format="%Y"))
covariate_ens = year_vector - 1980

# Fit the stationary distribution
fit_unseen_GEV <- fevd(x = UNSEEN_bc$t2m,
                       type = 'GEV',
                       use.phi = TRUE)

fit_unseen_Gumbel <- fevd(x = UNSEEN_bc$t2m,
                          type = 'Gumbel',
                          use.phi = TRUE)

# Fit the nonstationary distribution
fit_unseen_GEV_nonstat <- fevd(x = UNSEEN_bc$t2m,
                               type = 'GEV',
                               location.fun = ~ covariate_ens, ##Fitting the gev with␣
↪a location and scale parameter linearly correlated to the covariate (years)
                               scale.fun = ~ covariate_ens,
                               use.phi = TRUE)
```

And the likelihood ratio test tells us that the nonstationary GEV distribution is the best fit, both p-values < 2.2e-16:

[15]:
```
#And test the fit
##1. Stationary Gumbel vs stationary GEV
lr.test(fit_unseen_Gumbel,fit_unseen_GEV)
##2. Stationary GEV vs Nonstationary GEV
lr.test(fit_unseen_GEV, fit_unseen_GEV_nonstat)
```

```
        Likelihood-ratio Test

data:  UNSEEN_bc$t2mUNSEEN_bc$t2m
Likelihood-ratio = 568.39, chi-square critical value = 3.8415, alpha =
0.0500, Degrees of Freedom = 1.0000, p-value < 2.2e-16
alternative hypothesis: greater
```

```
        Likelihood-ratio Test

data:  UNSEEN_bc$t2mUNSEEN_bc$t2m
Likelihood-ratio = 945.52, chi-square critical value = 5.9915, alpha =
0.0500, Degrees of Freedom = 2.0000, p-value < 2.2e-16
alternative hypothesis: greater
```

We plot unseen trends in 100-year extremes. The function `unseen-trends1` fits the trend for a selected return period (rp) for both the observed and ensemble datasets. For the observed dataset, the year 2020 was not used in the fit. For more info on the UNSEEN-trend method see this paper and for more details on the results, see section 3.2 of the open workflow paper. The function was written for this case study in specific, and we cannot ensure robustness to other case studies.

[19]:
```
year_vector = as.integer(format(UNSEEN_bc$time, format="%Y"))
covariate_ens = year_vector - 1980

Trend_2year <- unseen_trends1(ensemble = UNSEEN_bc$t2m,
                      x_ens = year_vector,
                      x_obs = 1981:2020,
                      rp = 2,
                      obs = obs$t2m,
                      covariate_ens = covariate_ens,
                      covariate_obs = c(1:(length(obs$time)-1)),
                      covariate_values = c(1:length(obs$time)),
                      GEV_type = 'GEV',
                      ylab = 'August temperature (C)',
                      title = '2-year') +
ylim(c(20,28.5))

Trend_100year <- unseen_trends1(ensemble = UNSEEN_bc$t2m,
                      x_ens = year_vector,
                      x_obs = 1981:2020,
                      rp = 100,
                      obs = obs$t2m,
                      covariate_ens = covariate_ens,
                      covariate_obs = c(1:(length(obs$time)-1)),
                      covariate_values = c(1:length(obs$time)),
                      GEV_type = 'GEV',
                      ylab = '',
                      title = '100-year') +
ylim(c(20,28.5))
```

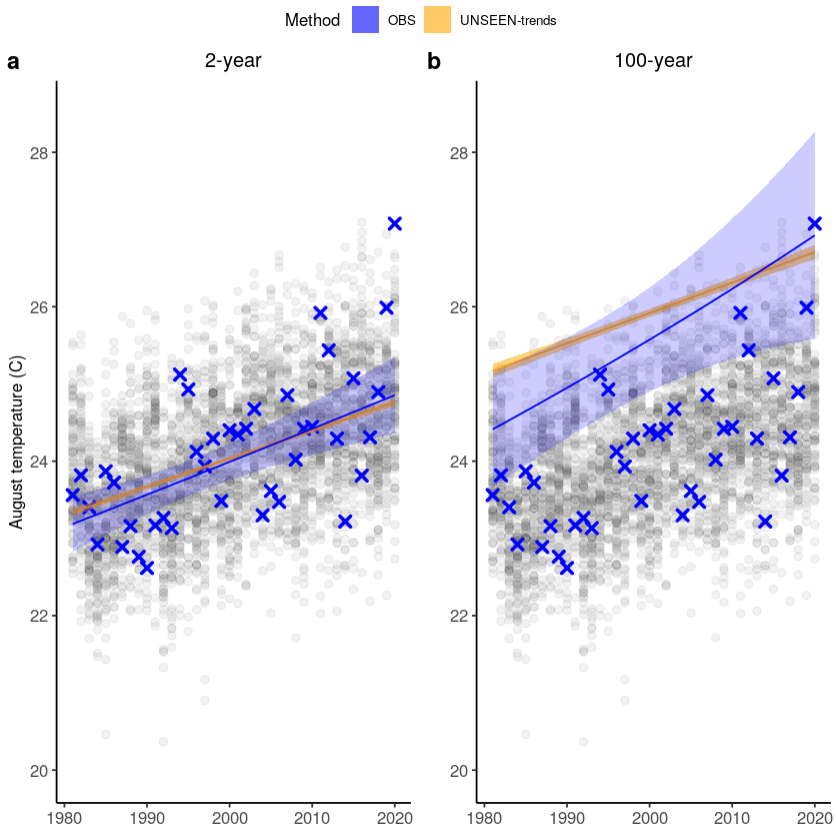
We combine the two plots:

[21]:
```
ggpubr::ggarrange(Trend_2year,Trend_100year,
                  labels = c("a","b"),
```

(continues on next page)

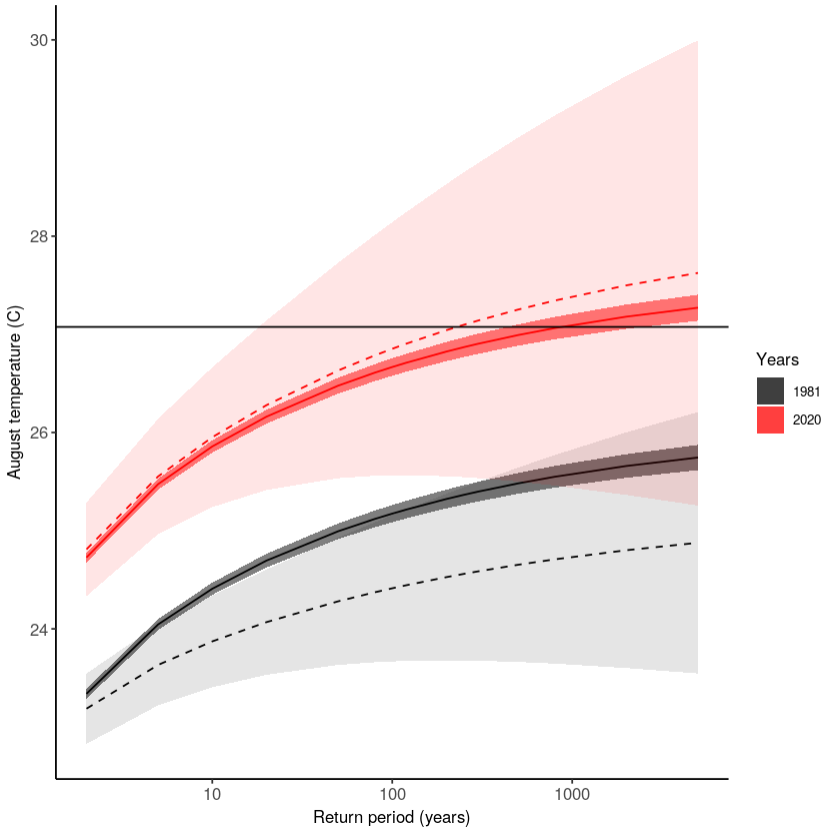
(continued from previous page)

```
                  common.legend = TRUE,
                  font.label = list(size = 14,
                                    color = "black",
                                    face = "bold",
                                    family = NULL),
                  ncol = 2,
                  nrow = 1) #+
# ggsave(height = 100, width = 180, units = 'mm', filename = "graphs/California_
↪trends.png")
```



There is a clear trend in the temperature extremes over last 40 years. How has this trend influenced the likelihood of occurrence of the 2020 event? The function `unseen_trends2` plots the extreme value distributions for the 'year' covariate 1981 and 2020. There is a clear difference – the distribution for 1981 does not even reach the 2020 event. See section 3.2 of the open workflow paper for more details on this exciting but scary result! Note that also this function was written for this case study in specific, and we cannot ensure robustness to other case studies.

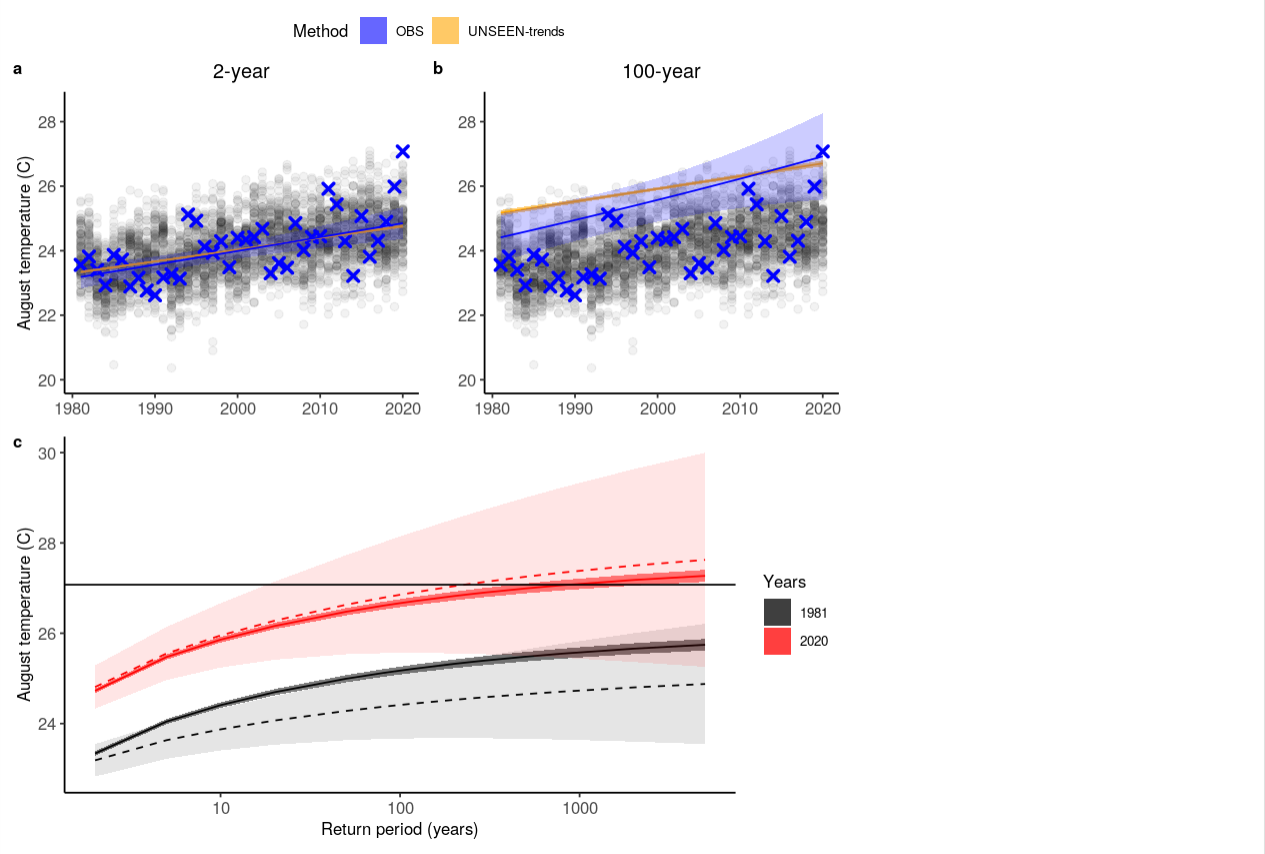
[25]:
```
p2 <- unseen_trends2(ensemble = UNSEEN_bc$t2m,
                     obs = obs[1:(length(obs$time)-1),]$t2m,
                     covariate_ens = covariate_ens,
                     covariate_obs = c(1:(length(obs$time)-1)),
                     GEV_type = 'GEV',
                     ylab = 'August temperature (C)')

Distributions = p2 + geom_hline(yintercept = obs[obs$time == '2020-08-01',]$t2m) #+
Distributions
```



Let's make a publication-ready plot by combining the above figures.

[135]:
```
Trends = ggpubr::ggarrange(Trend_2year,Trend_100year,
                  labels = c("a","b"),
                  common.legend = TRUE,
                  font.label = list(size = 10,
                                    color = "black",
                                    face = "bold",
                                    family = NULL),
                  ncol = 2,
                  nrow = 1)
```

[136]:
```
ggpubr::ggarrange(Trends,Distributions,
                  labels = c("","c"),
                  font.label = list(size = 10,
                                    color = "black",
                                    face = "bold",
                                    family = NULL),
                  ncol = 1,
                  nrow = 2) +
ggsave(height = 180, width = 180, units = 'mm', filename = "graphs/California_trends2.
↪pdf")
```



Applications:

We have seen the worst fire season over California in 2020. Such fires are likely part of a chain of impacts, from droughts to heatwaves to fires, with feedbacks between them. Here we assess August temperatures and show that the 2020 August average temperature was very anomalous. We furthermore use SEAS5 forecasts to analyze the trend in rare extremes. Evaluation metrics show that the model simulations have a high bias, which we correct for using an additive bias correction. UNSEEN trend analysis shows a clear trend over time, both in the model and in the observed temperatures. Based on this analysis, temperature extremes that you would expect to occur once in 1000 years in 1981 might occur once in <10 years at present (2020).

Note

Our analysis shows the results of a *linear* trend analysis of August temperature averages over 1981-2020. Other time windows, different trends than linear, and spatial domains could (should?) be investigated, as well as drought estimates in addition to temperature extremes.

Launch in Binder

1.3.3 UK Precipitation

February 2020 case study

February 2020 was the wettest February on record in the UK (since 1862), according to the Met Office. The UK faced three official storms during February, and this exceptional phenomena attracted media attention, such as an article from the BBC on increased climate concerns among the population.

Here, we will test the applicability and potential of using SEAS5 for estimating the likelihood of the 2020 UK February precipitation event.

Retrieve data

The main functions to retrieve all forecasts (SEAS5) is `retrieve_SEAS5`. We want to download February average precipitation over the UK. By default, the hindcast years of 1981-2016 are downloaded for SEAS5. The folder indicates where the files will be stored, in this case outside of the UNSEEN-open repository, in a 'UK_example' directory. For more explanation, see *retrieve*.

[2]:
```
import os
import sys
sys.path.insert(0, os.path.abspath('../../../'))
os.chdir(os.path.abspath('../../../'))

import src.cdsretrieve as retrieve
import src.preprocess as preprocess
```

[29]:
```
import numpy as np
import xarray as xr
```

[4]:
```
retrieve.retrieve_SEAS5(variables = 'total_precipitation',
                        target_months = [2],
                        area = [60, -11, 50, 2],
                        years=np.arange(1981, 2021),
                        folder = '../UK_example/SEAS5/')
```

We use the EOBS observational dataset to evaluate the UNSEEN ensemble. I tried to download EOBS through the Copernicus Climate Data Store, but the Product is temporally disabled for maintenance purposes. As workaround I downloaded EOBS (from 1950 - 2019) and the most recent EOBS data (2020) here. Note, you have to register as E-OBS user.

Preprocess

In the preprocessing step, we first merge all downloaded files into one xarray dataset, see preprocessing.

[6]:
```
SEAS5_UK = preprocess.merge_SEAS5(folder = '../UK_example/SEAS5/', target_months =␣
↪[2])
```

```
Lead time: 01
12
11
10
9
```

The SEAS5 total precipitation rate is in m/s. You can easily convert this and change the attributes. Click on the show/hide attributes button to see the assigned attributes.

[22]:
```
SEAS5_UK['tprate'] = SEAS5_UK['tprate'] * 1000 * 3600 * 24 ## From m/s to mm/d
SEAS5_UK['tprate'].attrs = {'long_name': 'rainfall',
 'units': 'mm/day',
 'standard_name': 'thickness_of_rainfall_amount'}
SEAS5_UK
```

[22]:
```
<xarray.Dataset>
Dimensions:    (latitude: 11, leadtime: 5, longitude: 14, number: 51, time: 39)
Coordinates:
  * longitude  (longitude) float32 -11.0 -10.0 -9.0 -8.0 ... -1.0 0.0 1.0 2.0
  * number     (number) int64 0 1 2 3 4 5 6 7 8 9 ... 42 43 44 45 46 47 48 49 50
  * time       (time) datetime64[ns] 1982-02-01 1983-02-01 ... 2020-02-01
  * latitude   (latitude) float32 60.0 59.0 58.0 57.0 ... 53.0 52.0 51.0 50.0
  * leadtime   (leadtime) int64 2 3 4 5 6
Data variables:
    tprate     (leadtime, time, number, latitude, longitude) float32 dask.array
↪<chunksize=(1, 1, 51, 11, 14), meta=np.ndarray>
Attributes:
    Conventions:  CF-1.6
    history:      2020-05-13 14:49:43 GMT by grib_to_netcdf-2.16.0: /opt/ecmw...
```

Then I open the EOBS dataset and extract February monthly mean precipitation. I have taken the average mm/day over the month, which is more fair than the total monthly precipitation because of leap days.

[30]:
```
EOBS = xr.open_dataset('../UK_example/EOBS/rr_ens_mean_0.25deg_reg_v20.0e.nc') ##␣
↪open the data
EOBS = EOBS.resample(time='1m').mean() ## Monthly averages
EOBS = EOBS.sel(time=EOBS['time.month'] == 2) ## Select only February
EOBS
```

```
/soge-home/users/cenv0732/.conda/envs/UNSEEN-open/lib/python3.8/site-packages/xarray/
↪core/nanops.py:142: RuntimeWarning: Mean of empty slice
  return np.nanmean(a, axis=axis, dtype=dtype)
```

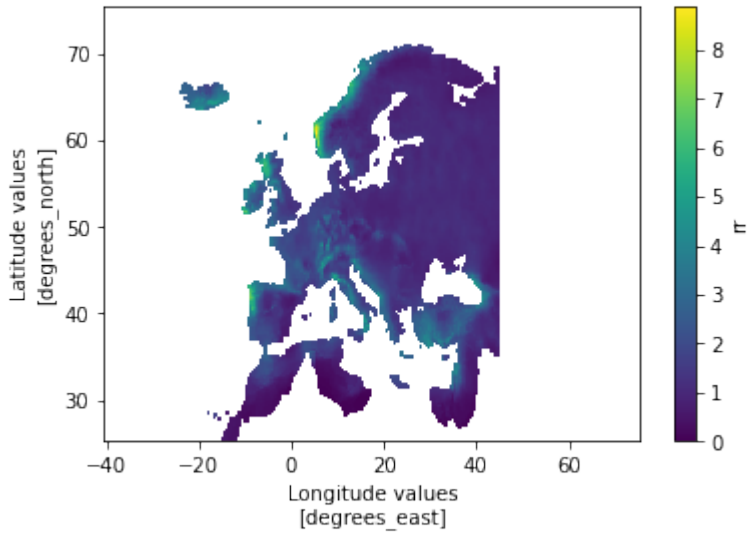
[30]:
```
<xarray.Dataset>
Dimensions:    (latitude: 201, longitude: 464, time: 70)
Coordinates:
  * time       (time) datetime64[ns] 1950-02-28 1951-02-28 ... 2019-02-28
  * latitude   (latitude) float64 25.38 25.62 25.88 26.12 ... 74.88 75.12 75.38
  * longitude  (longitude) float64 -40.38 -40.12 -39.88 ... 74.88 75.12 75.38
Data variables:
    rr         (time, latitude, longitude) float32 nan nan nan ... nan nan nan
```

Here I define the attributes, that xarray uses when plotting

[31]:
```
EOBS['rr'].attrs = {'long_name': 'rainfall',  ##Define the name
 'units': 'mm/day', ## unit
 'standard_name': 'thickness_of_rainfall_amount'} ## original name, not used
EOBS['rr'].mean('time').plot() ## and show the 1950-2019 average February␣
↪precipitation
```

```
/soge-home/users/cenv0732/.conda/envs/UNSEEN-open/lib/python3.8/site-packages/xarray/
↪core/nanops.py:142: RuntimeWarning: Mean of empty slice
  return np.nanmean(a, axis=axis, dtype=dtype)
```

[31]:
```
<matplotlib.collections.QuadMesh at 0x7f93884292b0>
```

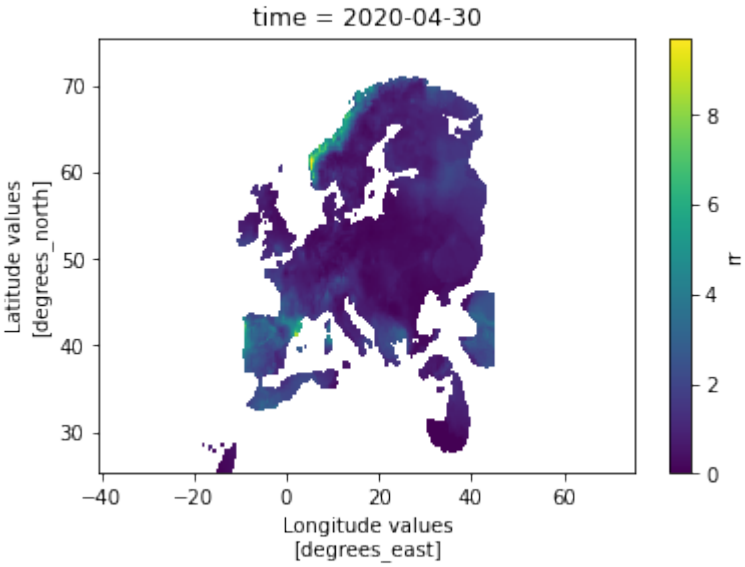


The 2020 data file is separate and needs the same preprocessing:

[32]:
```
EOBS2020 = xr.open_dataset('../UK_example/EOBS/rr_0.25deg_day_2020_grid_ensmean.nc.1
↪') #open
EOBS2020 = EOBS2020.resample(time='1m').mean() #Monthly mean
EOBS2020['rr'].sel(time='2020-04').plot() #show map
EOBS2020 ## display dataset
```

```
/soge-home/users/cenv0732/.conda/envs/UNSEEN-open/lib/python3.8/site-packages/xarray/
↪core/nanops.py:142: RuntimeWarning: Mean of empty slice
  return np.nanmean(a, axis=axis, dtype=dtype)
```

[32]:
```
<xarray.Dataset>
Dimensions:    (latitude: 201, longitude: 464, time: 12)
Coordinates:
  * time       (time) datetime64[ns] 2020-01-31 2020-02-29 ... 2020-12-31
  * latitude   (latitude) float64 25.38 25.62 25.88 26.12 ... 74.88 75.12 75.38
  * longitude  (longitude) float64 -40.38 -40.12 -39.88 ... 74.88 75.12 75.38
Data variables:
    rr         (time, latitude, longitude) float32 nan nan nan ... nan nan nan
```



We then extract UK averaged precipitation for SEAS5 and EOBS. We upscale EOBS to the SEAS5 grid and apply the same UK mask to extract the UK average for both datasets. *Using EOBS + upscaling* shows how to regrid and extract the country average timeseries.

Here, we export the SEAS5 and EOBS datasets as NetCDF files to be imported in the other notebook. Note that for EOBS we had to download two separate files, which we concatenate below before exporting as nc.

[33]:
```
SEAS5_UK.to_netcdf('../UK_example/SEAS5/SEAS5_UK.nc') ## Store SEAS5 as NetCDF for␣
↪future import

EOBS_concat = xr.concat([EOBS,EOBS2020.sel(time='2020-02')],dim='time') ##␣
↪Concatenate the 1950-2019 and 2020 datasets.
EOBS_concat.to_netcdf('../UK_example/EOBS/EOBS_UK.nc') ## And store the 1950-2010␣
↪February precipitation into one nc for future import
```

Evaluate

Note

From here onward we use R and not python!

We switch to R since we believe R has a better functionality in extreme value statistics.

[1]:
```
setwd('../../..')
# getwd()
EOBS_UK_weighted_df <- read.csv("Data/EOBS_UK_weighted_upscaled.csv",␣
↪stringsAsFactors=FALSE)
SEAS5_UK_weighted_df <- read.csv("Data/SEAS5_UK_weighted_masked.csv",␣
↪stringsAsFactors=FALSE)

## Convert the time class to Date format
EOBS_UK_weighted_df$time <- lubridate::ymd(EOBS_UK_weighted_df$time)
str(EOBS_UK_weighted_df)
```

(continues on next page)

(continued from previous page)

```
EOBS_UK_weighted_df_hindcast <- EOBS_UK_weighted_df[
    EOBS_UK_weighted_df$time > '1982-02-01' &
    EOBS_UK_weighted_df$time < '2017-02-01',
    ]


SEAS5_UK_weighted_df$time <- lubridate::ymd(SEAS5_UK_weighted_df$time)
str(SEAS5_UK_weighted_df)
```

```
'data.frame':   71 obs. of  2 variables:
 $ time: Date, format: "1950-02-28" "1951-02-28" ...
 $ rr  : num  4.13 3.25 1.07 1.59 2.59 ...
'data.frame':   9945 obs. of  4 variables:
 $ leadtime: int  2 2 2 2 2 2 2 2 2 2 ...
 $ number  : int  0 0 0 0 0 0 0 0 0 0 ...
 $ time    : Date, format: "1982-02-01" "1983-02-01" ...
 $ tprate  : num  1.62 2.93 3.27 2 3.31 ...
```

Is the UNSEEN ensemble realistic?

To answer this question, we perform three statistical tests: independence, model stability and model fidelity tests. These statistical tests are available through the UNSEEN R package. See *evaluation* for more info.

[2]:
```
require(UNSEEN)
require(ggplot2)
require(ggpubr)
```

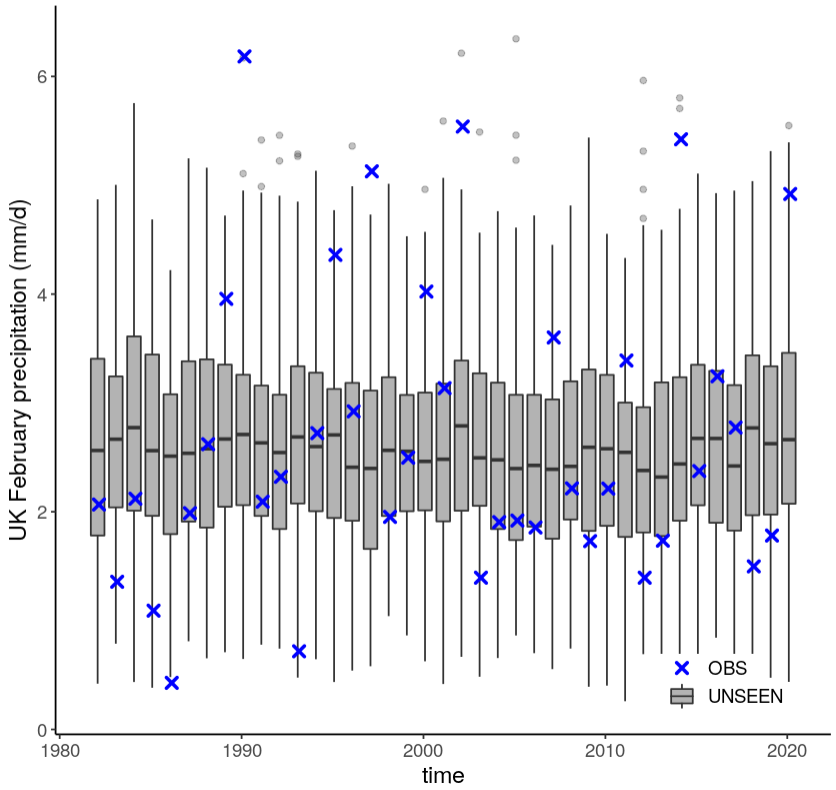
```
Loading required package: UNSEEN

Loading required package: ggplot2

Warning message:
"replacing previous import 'vctrs::data_frame' by 'tibble::data_frame' when loading␣
↪'dplyr'"
Loading required package: ggpubr
```

Timeseries

We plot the timeseries of SEAS5 (UNSEEN) and EOBS (OBS) for UK February precipitation.

[3]:
```
unseen_timeseries(ensemble = SEAS5_UK_weighted_df,
                                obs = EOBS_UK_weighted_df[EOBS_UK_weighted_df$time >
↪'1982-02-01',],
                                ylab = 'UK February precipitation (mm/d)') +
theme(text = element_text(size = 14)) #This is just to increase the figure font
```

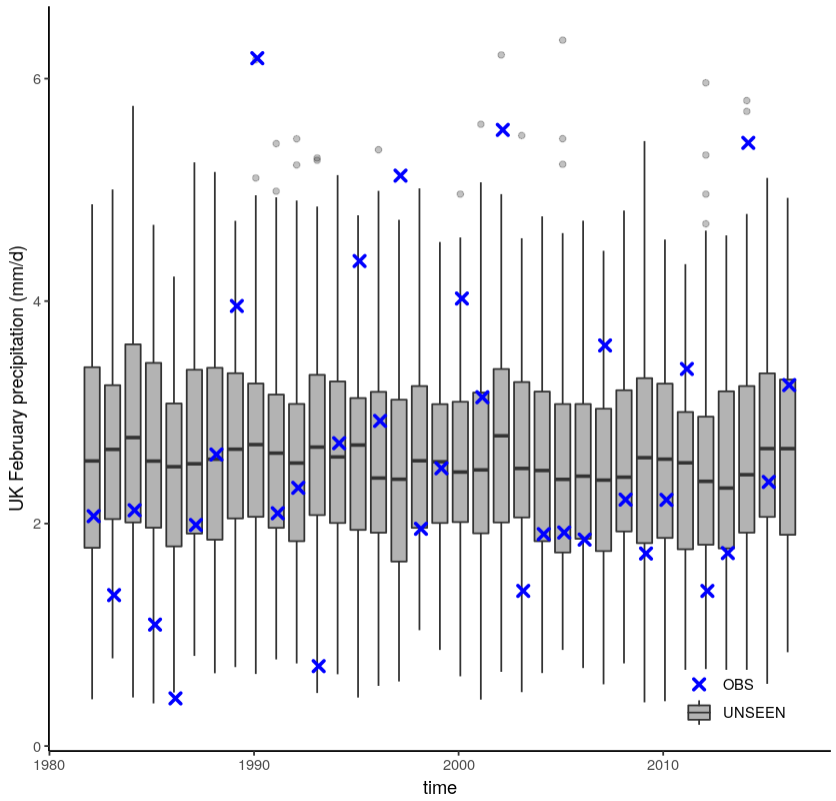
```
Warning message:
"Removed 4654 rows containing non-finite values (stat_boxplot)."
```



We select the timeseries for the **hindcast years 1981-2016**.

[4]:
```
SEAS5_UK_hindcast <- SEAS5_UK_weighted_df[
    SEAS5_UK_weighted_df$time < '2017-02-01' &
    SEAS5_UK_weighted_df$number < 25,]
```

[5]:
```
unseen_timeseries(ensemble = SEAS5_UK_hindcast,
                  obs = EOBS_UK_weighted_df_hindcast,
                  ylab = 'UK February precipitation (mm/d)')#  %>%
# ggsave(height = 5, width = 6,   filename = "graphs/UK_timeseries.png")
```



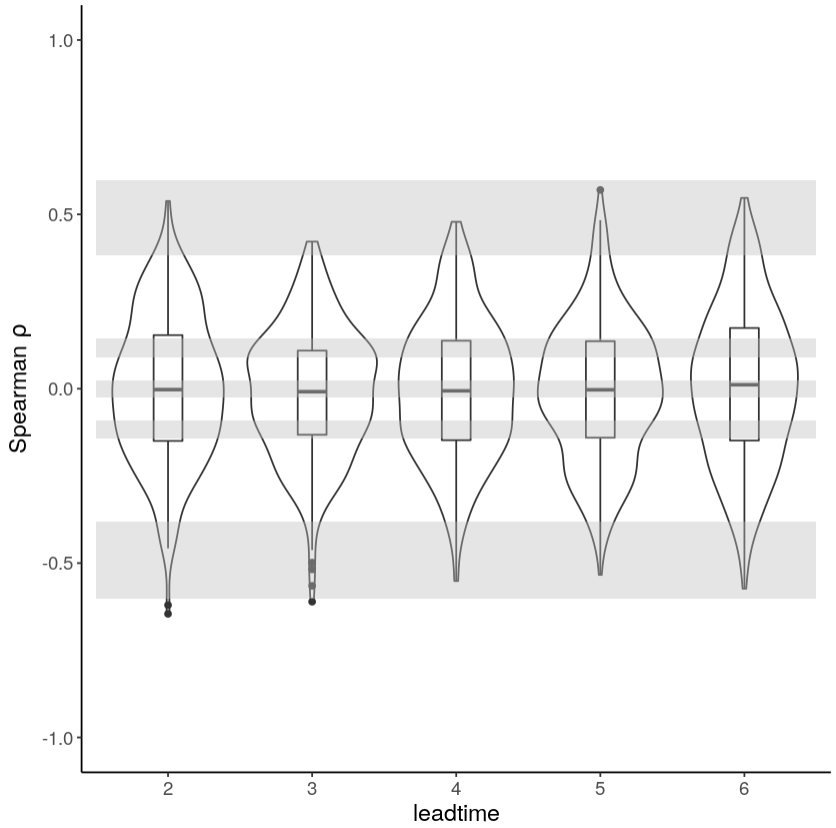
Evaluation tests

With the hindcast dataset we evaluate the independence, stability and fidelity.

First the *independence test*. This test checks if the forecasts are independent. If they are not, the event are not unique and care should be taken in the extreme value analysis. Because of the chaotic behaviour of the atmosphere, independence of precipitation events is expected beyond a lead time of two weeks. Here we use lead times 2-6 months and find that the boxplots are within the expected range (perhaps very small dependence in lead time 2). More info in our paper: https://doi.org/10.1038/s41612-020-00149-4.

[6]:
```
Independence_UK = independence_test(ensemble = SEAS5_UK_hindcast,
                                    detrend = TRUE) +
    theme(text = element_text(size = 14))

Independence_UK
```

```
Warning message:
"Removed 1625 rows containing non-finite values (stat_ydensity)."
Warning message:
"Removed 1625 rows containing non-finite values (stat_boxplot)."
```

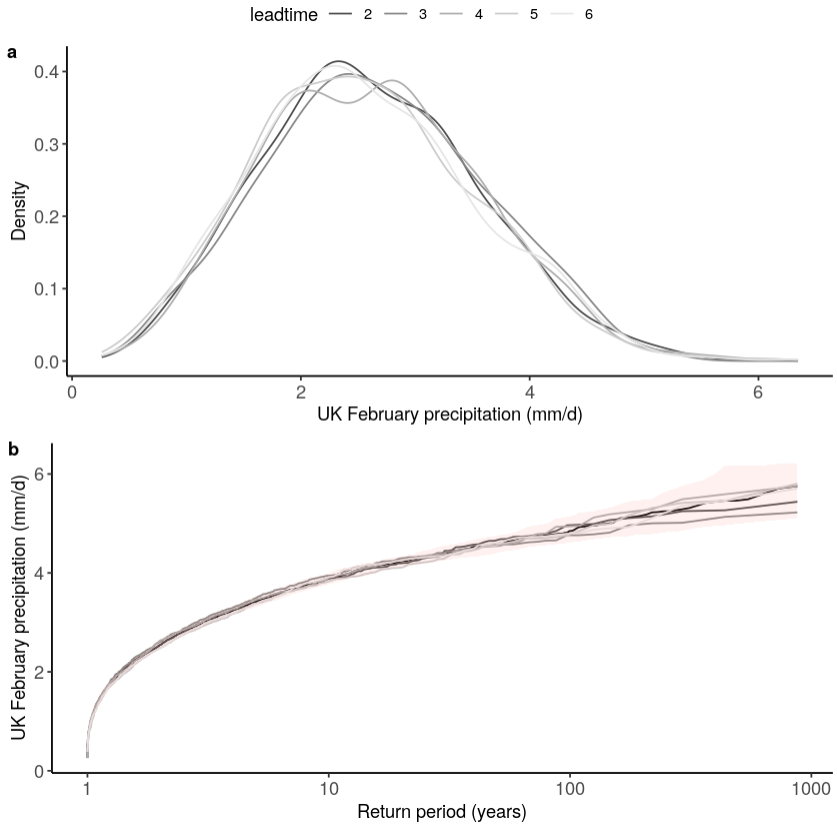


The test for *model stability*: Is there a drift in the simulated precipitation over lead times?

We find that the model is stable for UK February precipitation.

[7]:
```
Stability_UK = stability_test(ensemble = SEAS5_UK_hindcast,
                              lab = 'UK February precipitation (mm/d)')

Stability_UK
```

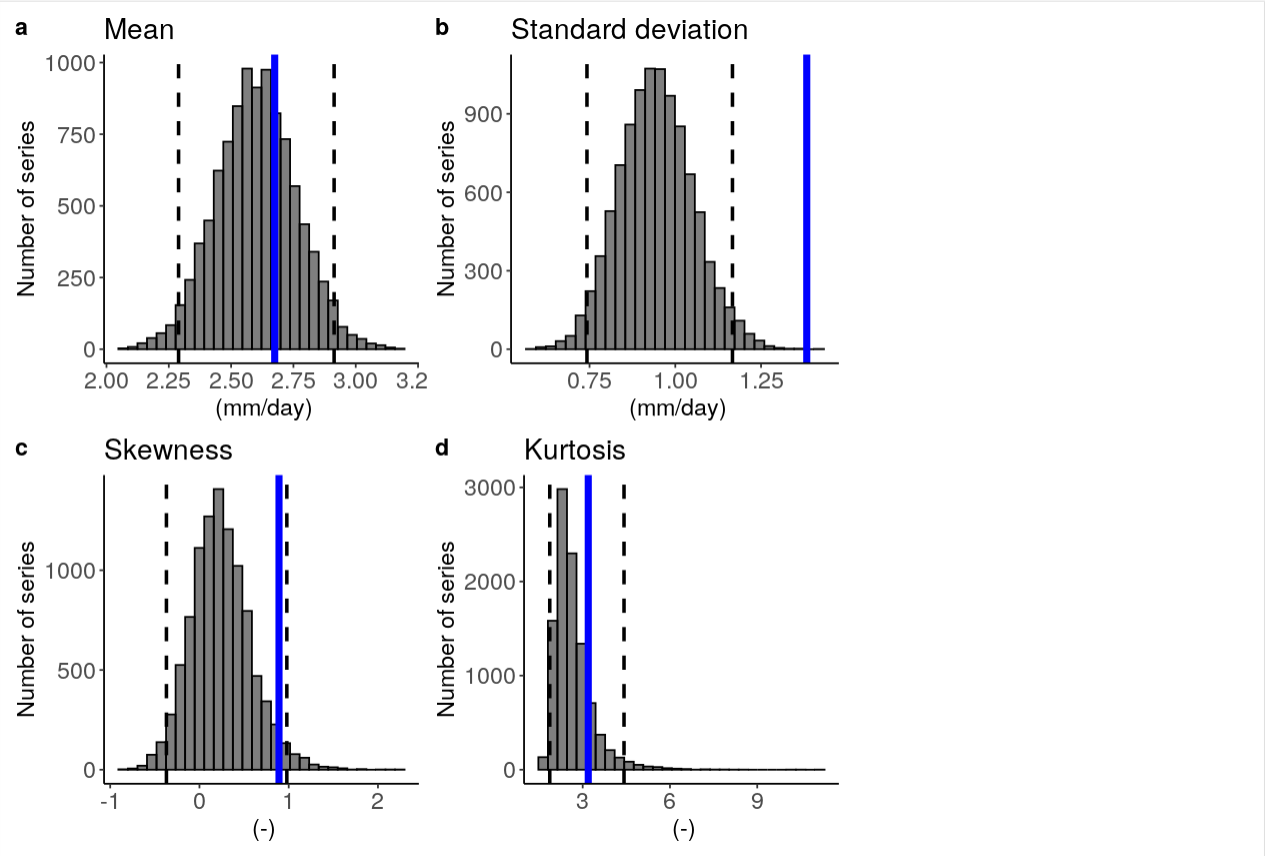
```
Warning message:
"Removed 4 row(s) containing missing values (geom_path)."
```



The *fidelity test* shows us how consistent the model simulations of UNSEEN (SEAS5) are with the observed (EOBS). With this test we can asses systematic biases. The UNSEEN dataset is much larger than the observed – hence they cannot simply be compared. For example, what if we had faced a few more or a few less precipitation extremes purely by chance?

This would influence the observed mean, but not so much influence the UNSEEN ensemble because of the large data sample. Therefore we express the UNSEEN ensemble as a range of plausible means, for data samples of the same length as the observed. We do the same for higher order statistical moments.

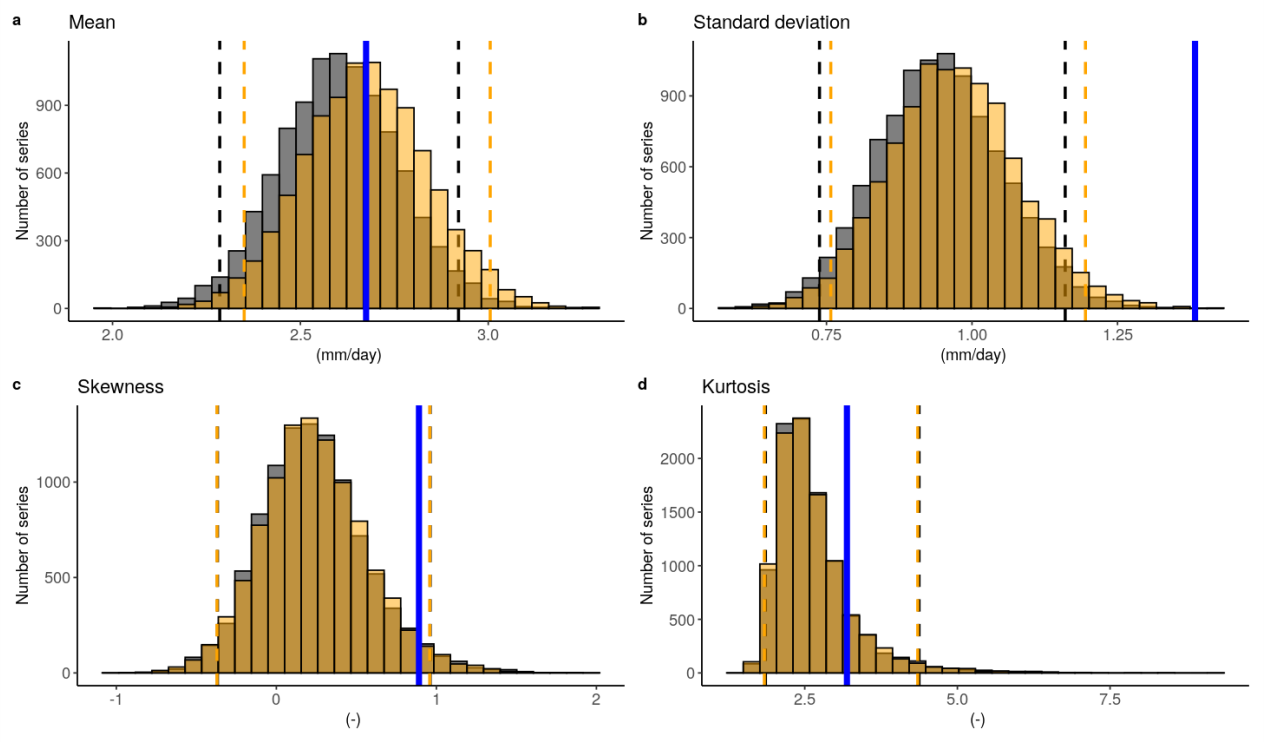
[8]:
```
Fidelity_UK = fidelity_test(obs = EOBS_UK_weighted_df_hindcast$rr,
                            ensemble = SEAS5_UK_hindcast$tprate,
                            fontsize = 14
                            )
Fidelity_UK
```



We find that the standard deviation within the model (the grey histograms and lines) are too low compared to the observed.

We can include a simple mean-bias correction (ratio) in this plot by setting `biascor = TRUE`. However, in this case it won't help:

[16]:
```
fidelity_test(obs = EOBS_UK_weighted_df_hindcast$rr,
              ensemble = SEAS5_UK_hindcast$tprate,
              biascor = TRUE
             )
```



Check the documentation of the test `?fidelity_test`

Illustrate

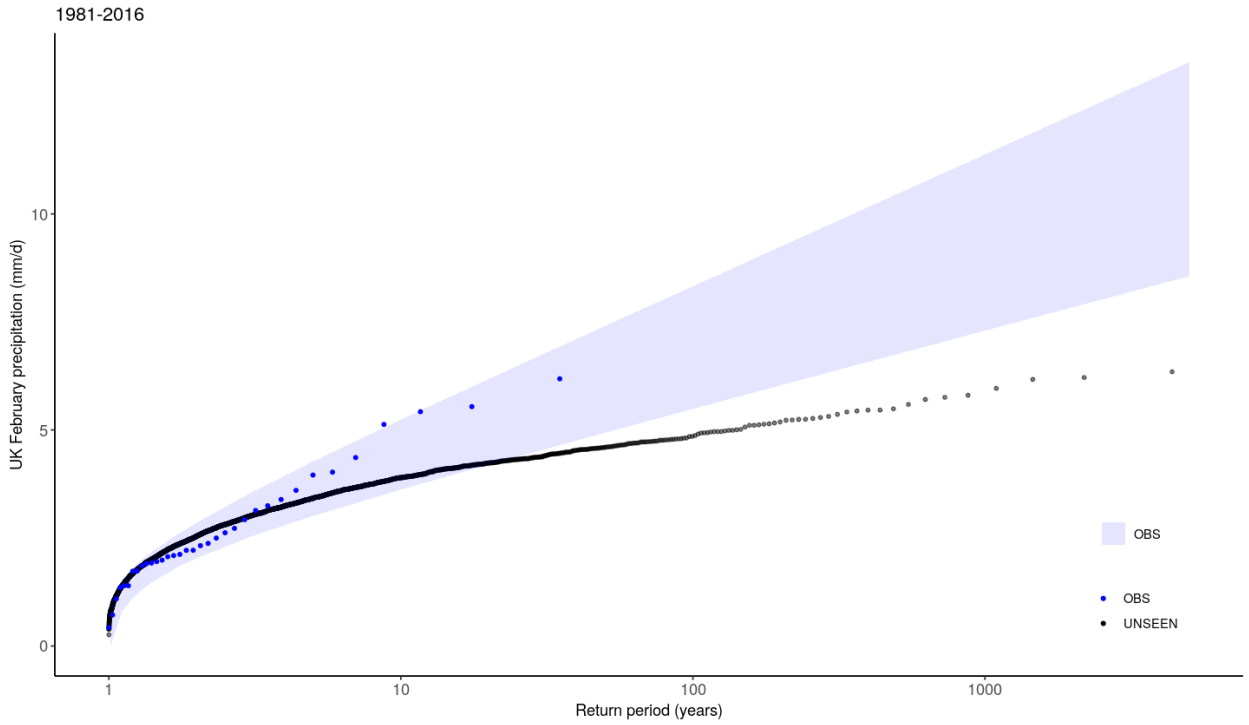
First, we fit a Gumbel and a GEV distribution (including shape parameter) to the observed extremes. The Gumbel distribution best describes the data because the p-value of 0.9 is much above 0.05 (based on the likelihood ratio test).

[9]:
```
fit_obs_Gumbel <- fevd(x = EOBS_UK_weighted_df_hindcast$rr,
                       type = "Gumbel"
                      )
fit_obs_GEV <- fevd(x = EOBS_UK_weighted_df_hindcast$rr,
                       type = "GEV"
                      )
lr.test(fit_obs_Gumbel, fit_obs_GEV)
```

```
        Likelihood-ratio Test

data:  EOBS_UK_weighted_df_hindcast$rrEOBS_UK_weighted_df_hindcast$rr
Likelihood-ratio = 0.014629, chi-square critical value = 3.8415, alpha
= 0.0500, Degrees of Freedom = 1.0000, p-value = 0.9037
alternative hypothesis: greater
```

We show the gumbel plot for the observed (EOBS) and UNSEEN (SEAS5 hindcast data). This shows that the UNSEEN simulations are not within the uncertainty range of the observations. This has to do with the variability of the model that is too low, as indicated in the evaluation section. The `EVT_plot` function was written for this case study, but we cannot ensure robustness to other case studies.

[14]:
```
source('src/evt_plot.r')
options(repr.plot.width = 12)
EVT_plot(ensemble = SEAS5_UK_hindcast$tprate,
                      obs = EOBS_UK_weighted_df_hindcast$rr,
                      main = "1981-2016",
                      GEV_type = "Gumbel",
#                           ylim = 3,
                      y_lab = 'UK February precipitation (mm/d)'
         )
```



Potential

We find that there is there too little variability within SEAS5 hindcasts of UK february precipitation. It might be resolution dependent or related to the signal-to-noise that is a problem over this region. The results can be fed back to model developers to help improve the models.

The use of other observational datasets and other model simulations can be further explored. For example, the UK Met Office studied UK monthly precipitation extremes using the UNSEEN method (Thompson et al., 2017). They showed that monthly precipitation records for south east England have a 7% chance of being exceeded in at least one month in any given winter. Their work was taken up in the UK National Flood Resilience Review (2016), showing the high relevance of the method.

1.4 Retrieve

We want to download the monthly precipitation for February. I use the automatically generated request from the CDS server. There are two datasets we can use to download the data: Seasonal forecast daily data on single levels and Seasonal forecast monthly statistics on single levels. We will use the latter for easy downloading of the monthly values. If we want to go to higher temporal resolution, such as daily extremes, we will have to consult the other dataset.

To get started with CDS, you have to register at https://cds.climate.copernicus.eu/ and copy your UID and API key from https://cds.climate.copernicus.eu/user in the `~/.cdsapirc` file in the home directory of your user. See the ml-flood project for more details

[7]:
```
UID = 'UID'
API_key = 'API_key'
```

[8]:
```
import os
#Uncomment the following lines to write the UID and API key in the .cdsapirc file
# with open(os.path.join(os.path.expanduser('~'), '.cdsapirc'), 'w') as f:
#     f.write('url: https://cds.climate.copernicus.eu/api/v2\n')
#     f.write(f'key: {UID}:{API_key}')
```

[8]:
```
46
```

[8]:
```
47
```

1.4.1 Import packages

[1]:
```
##This is so variables get printed within jupyter
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

[2]:
```
##import packages
import os
import cdsapi ## check the current working directory, which should be the UNSEEN-open
↪directory
import numpy as np
import xarray as xr
import matplotlib.pyplot as plt
import numpy as np
import cartopy
import cartopy.crs as ccrs
```

[3]:
```
##We want the working directory to be the UNSEEN-open directory
pwd = os.getcwd() ##current working directory is UNSEEN-open/Notebooks/1.Download
pwd #print the present working directory
os.chdir(pwd+'/../../../') # Change the working directory to UNSEEN-open
os.getcwd() #print the working directory
```

[3]:
```
'C:\\Users\\Timo\\OneDrive - Loughborough University\\GitHub\\UNSEEN-open\\doc\\
↪Notebooks\\1.Download'
```

[3]:
```
'C:\\Users\\Timo\\OneDrive - Loughborough University\\GitHub\\UNSEEN-open\\doc'
```

1.4.2 First download

In our request, we will use the monthly mean. Interestingly, there is also the option to use the monthly maximum! We previously downloaded the data on daily resolution and extracted the monthly (or seasonal) maximum from that data. If we could just download the monthly maximum instead that might save a lot of processing power! However, you would be restricted to daily extremes only, for multi-day extremes (5 days is often used), you would have to do the original processing workflow. We select the UK domain to reduce the size of the download.

Here I download the monthly mean total precipitation (both convective and large scale precipitation) forecast for February 1993. It downloads all 25 ensemble members for the forecasts initialized in january.

[4]:
```
##Our first download:

c = cdsapi.Client()

c.retrieve(
    'seasonal-monthly-single-levels',
    {
        'format': 'netcdf',
        'originating_centre': 'ecmwf',
        'system': '5',
        'variable': 'total_precipitation',
        'product_type': [
            'monthly_mean', #'monthly_maximum',, 'monthly_standard_deviation',
        ],
        'year': '1993', #data before 1993 is available.
        'month': '01', #Initialization month. Target month is February (2),␣
↪initialization months are August-January (8-12,1)
        'leadtime_month': [ ##Use of single months is much faster. Leadtime 0 does␣
↪not exist. The first lead time is 1.
            '1', '2',
        ],
        'area': [##Select UK domain to reduce the size of the download
            60, -11, 50,
            2,
        ],
    },
    'Data/First_download.nc') ##can I use nc? yes!
```

```
2020-05-13 10:08:56,140 INFO Welcome to the CDS
2020-05-13 10:08:56,142 INFO Sending request to https://cds.climate.copernicus.eu/api/
↪v2/resources/seasonal-monthly-single-levels
2020-05-13 10:08:56,983 INFO Request is completed
2020-05-13 10:08:56,984 INFO Downloading http://136.156.132.110/cache-compute-0001/
↪cache/data0/adaptor.mars.external-1589266964.5635436-26283-29-a38e8975-b0ec-49ee-
↪8f9b-7dea389f59cf.nc to Data/First_download.nc (16.4K)
2020-05-13 10:08:57,131 INFO Download rate 112.7K/s
```

[4]:
```
Result(content_length=16800,content_type=application/x-netcdf,location=http://136.156.
↪132.110/cache-compute-0001/cache/data0/adaptor.mars.external-1589266964.5635436-
↪26283-29-a38e8975-b0ec-49ee-8f9b-7dea389f59cf.nc)
```

Use xarray to visualize the netcdf file

I open the downloaded file and plot February 1993 precipitation over the UK.

[5]:
```
pr_1993_ds=xr.open_dataset('Data/First_download.nc')
pr_1993_ds
```

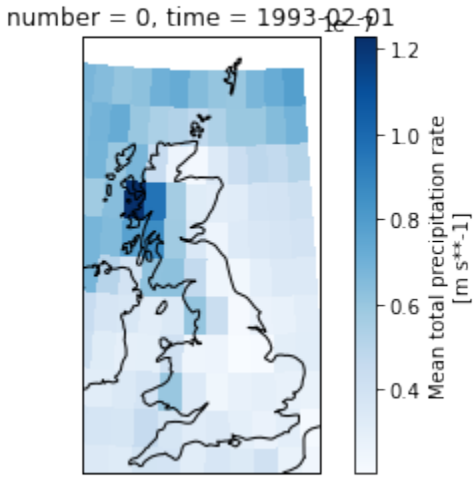
[5]:
```
<xarray.Dataset>
Dimensions:    (latitude: 11, longitude: 14, number: 25, time: 2)
Coordinates:
  * longitude  (longitude) float32 -11.0 -10.0 -9.0 -8.0 ... -1.0 0.0 1.0 2.0
  * latitude   (latitude) float32 60.0 59.0 58.0 57.0 ... 53.0 52.0 51.0 50.0
  * number     (number) int32 0 1 2 3 4 5 6 7 8 9 ... 16 17 18 19 20 21 22 23 24
  * time       (time) datetime64[ns] 1993-01-01 1993-02-01
Data variables:
    tprate     (time, number, latitude, longitude) float32 ...
Attributes:
    Conventions:  CF-1.6
    history:      2020-05-12 07:02:45 GMT by grib_to_netcdf-2.16.0: /opt/ecmw...
```

I select ensemble member 0 and february precipitation ('tprate' called apparently) and I use cartopy to make the map.

[6]:
```
## Use cartopy for nicer maps
ax = plt.axes(projection= ccrs.OSGB())
pr_1993_ds['tprate'].sel(number=0,time='1993-02').plot(transform=ccrs.PlateCarree(),
↪cmap=plt.cm.Blues, ax=ax)  #,cmap=plt.cm.Blues,

# ax.set_extent(extent)
ax.coastlines(resolution='50m')
plt.draw()
```

[6]:
```
<matplotlib.collections.QuadMesh at 0x7f5435adfa60>
```

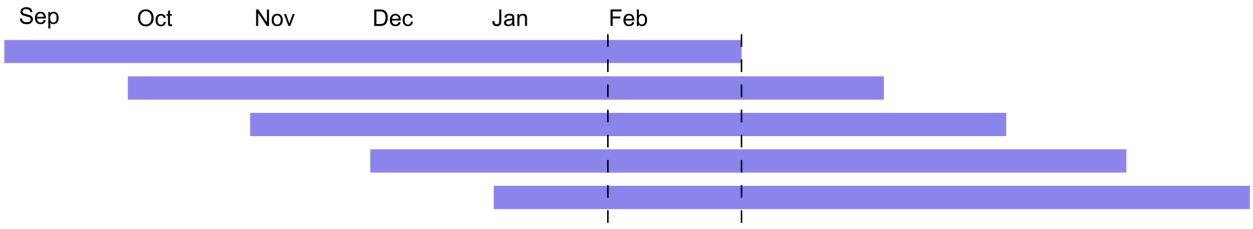
[6]:
```
<cartopy.mpl.feature_artist.FeatureArtist at 0x7f5435b6ff10>
```



Download all data

We will be using the SEAS5 hindcast, which is a dataset running from 1981-2016. The hindcast is initialized every month with 25 ensemble members and the forecast run for 6 months, indicated by blue horizontal bars below. February is forecasted by 6 initialization months (September-February). We discard the first month of the forecast because of dependence between the forecasts, explained in the evaluation section and are left with 5 initialization months (Sep-Jan) and 25 ensemble members forecasting february precipitation each year, totalling to an increase of 125 times the observed length.

For a summary of all available C3S seasonal hindcasts, their initialization months and more specifics, please see ECMWF page and the SEAS5 paper.



The first download example above downloaded all 25 ensemble members for the forecast initialized in January (the bottom bar). We should repeat this over the other initialization month and over all years (1981-2016).

[58]:
```
init_months = np.append(np.arange(9,13),1) ## Initialization months 9-12,1 (Sep-Jan)
init_months
years = np.arange(1982,2017)
years
```

[58]:
```
array([ 9, 10, 11, 12,  1])
```

[58]:
```
array([1982, 1983, 1984, 1985, 1986, 1987, 1988, 1989, 1990, 1991, 1992,
       1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003,
       2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014,
       2015, 2016])
```

For our download, we loop over initialization months and years. Because we only want February precipitation, the leadtime month (forecast length) changes with the initialization month. For example, in the September initialized forecasts, we only want the leadtime month 6 = February. For August initialized this is leadtime 5, etc. Furthermore, the year the forecast is initialized is required for the download. For September - December initialized forecasts, this is the target year-1. For January it is the same year as the target year. For example, for the first two target years this looks like the following:

[101]:
```
for j in range(2):#len(years)):
    for i in range(len(init_months)):
        init_month = init_months[i]
        leadtime_month = 6-i
        if init_month == 1:
            year = years[j]
        else:
            year = years[j]-1
        print ('year = ' + str(year) +' init_month = ' + str(init_month) + ' leadtime_
↪month = ' + str(leadtime_month))
```

```
year = 1981 init_month = 9 leadtime_month = 6
year = 1981 init_month = 10 leadtime_month = 5
year = 1981 init_month = 11 leadtime_month = 4
```

(continues on next page)

(continued from previous page)

```
year = 1981 init_month = 12 leadtime_month = 3
year = 1982 init_month = 1 leadtime_month = 2
year = 1982 init_month = 9 leadtime_month = 6
year = 1982 init_month = 10 leadtime_month = 5
year = 1982 init_month = 11 leadtime_month = 4
year = 1982 init_month = 12 leadtime_month = 3
year = 1983 init_month = 1 leadtime_month = 2
```

Write a function that is used for the download.

[72]:
```
def retrieve(variable, originating_centre, year, init_month, leadtime_month):

    c.retrieve(
        'seasonal-monthly-single-levels',
        {
            'format': 'netcdf',
            'originating_centre': originating_centre,
            'system': '5',
            'variable': variable,
            'product_type': [
                'monthly_mean', #'monthly_maximum',, 'monthly_standard_deviation',
            ],
            'year': str(year), #data before 1993 is available.
            'month': "%.2i" % init_month, #Initialization month. Target month is␣
↪February (2), initialization months are August-January (8-12,1)
            'leadtime_month': [ ##The lead times you want. Use of single months is␣
↪much faster. Leadtime 0 does not exist. The first lead time is 1.
                #For initialization month 1 (January), the leadtime months is 2␣
↪(February). For initialization month 12 (december), the lead time month is 3␣
↪(February).
                str(leadtime_month),
            ],
            'area': [##Select UK domain to reduce the size of the download
                     ## 25N-75N x. 40W-75E
                60, -11, 50, 2,
            ],
        },
        '../UK_example/'+ str(year) + "%.2i" % init_month + '.nc')

# retrieve(variable = 'total_precipitation',originating_centre = 'ecmwf', year =␣
↪years[0], init_month = "%.2i" % init_months[0])
```

And start the download! In total, we request 35 years x initialization dates = 175 requests. I could try sending just 5 request of the different initialization dates for all years?

[ ]:
```
for j in range(len(years)):  ##add if error still continue
    for i in range(len(init_months)):
        init_month = init_months[i]
        leadtime_month = 6 - i
        if init_month == 1:
            year = years[j]
        else:
            year = years[j] - 1
        retrieve(variable='total_precipitation',
                 originating_centre='ecmwf',
                 year=year,
```

(continues on next page)

(continued from previous page)

```
                init_month=init_month,
                leadtime_month=leadtime_month)
```

```
2020-05-18 10:14:48,767 INFO Welcome to the CDS
2020-05-18 10:14:48,768 INFO Sending request to https://cds.climate.copernicus.eu/api/
↪v2/resources/seasonal-monthly-single-levels
2020-05-18 10:14:49,485 INFO Downloading http://136.156.132.235/cache-compute-0006/
↪cache/data5/adaptor.mars.external-1589380912.7108843-4209-7-1add31ae-a0cd-44ce-83ac-
↪9ff7c97f1b01.nc to ../UK_example/198109.nc (8.9K)
2020-05-18 10:14:49,575 INFO Download rate 101.5K/s
2020-05-18 10:14:49,803 INFO Welcome to the CDS
2020-05-18 10:14:49,804 INFO Sending request to https://cds.climate.copernicus.eu/api/
↪v2/resources/seasonal-monthly-single-levels
2020-05-18 10:14:50,498 INFO Downloading http://136.156.132.153/cache-compute-0002/
↪cache/data4/adaptor.mars.external-1589381056.172494-12462-1-c9714216-87ac-49bc-be19-
↪260627a9077d.nc to ../UK_example/198110.nc (8.9K)
2020-05-18 10:14:50,571 INFO Download rate 124.6K/s
2020-05-18 10:14:51,070 INFO Welcome to the CDS
2020-05-18 10:14:51,071 INFO Sending request to https://cds.climate.copernicus.eu/api/
↪v2/resources/seasonal-monthly-single-levels
2020-05-18 10:14:51,213 INFO Downloading http://136.156.132.235/cache-compute-0006/
↪cache/data9/adaptor.mars.external-1589381301.6300867-8112-3-49ba0ab2-34fe-4364-9dec-
↪700bf911b079.nc to ../UK_example/198111.nc (8.9K)
2020-05-18 10:14:51,254 INFO Download rate 219.7K/s
2020-05-18 10:14:51,415 INFO Welcome to the CDS
2020-05-18 10:14:51,416 INFO Sending request to https://cds.climate.copernicus.eu/api/
↪v2/resources/seasonal-monthly-single-levels
2020-05-18 10:14:51,548 INFO Request is queued
```

The download sometimes fails. When redoing the request it does download. I don't know what is causing the failure? Below I donwload the file that failed.

[97]:
```
#201501 missing

year = 2015
init_month = 1
leadtime_month = 2
retrieve(variable = 'total_precipitation',originating_centre = 'ecmwf', year = year,
                init_month = init_month, leadtime_month = leadtime_month)
```

```
2020-05-15 11:51:16,127 INFO Welcome to the CDS
2020-05-15 11:51:16,129 INFO Sending request to https://cds.climate.copernicus.eu/api/
↪v2/resources/seasonal-monthly-single-levels
2020-05-15 11:51:16,327 INFO Downloading http://136.156.133.46/cache-compute-0015/
↪cache/data7/adaptor.mars.external-1589527607.2123153-8094-37-3b786f72-2e2a-462f-
↪bbb8-9c8d89c05102.nc to ../UK_example/201501.nc (8.9K)
2020-05-15 11:51:16,485 INFO Download rate 56.7K/s
```

Retrieve function

We have written a module where the above procedure is done automatically. Here we load the retrieve module and retrieve SEAS5 and ERA5 data for the examples by selecting the variable, target month(s), area and folder where we want to download the file in.

The main function to download the data is `retrieve.retrieve_SEAS5`. The function only downloads the target months, for each year and each intialization month. To do this, it obtains the initialization months and leadtimes from the selected target month(s). For the UK example, we select February as our target month, hence sep-jan will be our initialization months with leadtimes 2-6, see *Download all*.

[7]:
```
retrieve.print_arguments([2])
```

```
year = 1982 init_month = 1 leadtime_month = [2]
year = 1981 init_month = 12 leadtime_month = [3]
year = 1981 init_month = 11 leadtime_month = [4]
year = 1981 init_month = 10 leadtime_month = [5]
year = 1981 init_month = 9 leadtime_month = [6]
```

For the Siberia example this will be different, since the target months are march-may:

[8]:
```
retrieve.print_arguments([3,4,5])
```

```
year = 1982 init_month = 2 leadtime_month = [2 3 4]
year = 1982 init_month = 1 leadtime_month = [3 4 5]
year = 1981 init_month = 12 leadtime_month = [4 5 6]
```

Call `?retrieve.retrieve_SEAS5` to see the documentation.

For the California example, we use:

[ ]:
```
retrieve.retrieve_SEAS5(
    variables=['2m_temperature', '2m_dewpoint_temperature'],
    target_months=[8],
    area=[70, -130, 20, -70],
    years=np.arange(1981, 2021),
    folder='E:/PhD/California_example/SEAS5/')
```

[ ]:
```
retrieve.retrieve_ERA5(variables=['2m_temperature', '2m_dewpoint_temperature'],
                       target_months=[8],
                       area=[70, -130, 20, -70],
                       folder='E:/PhD/California_example/SEAS5/')
```

For the Siberia example:

[ ]:
```
retrieve.retrieve_SEAS5(
    variables=['2m_temperature', '2m_dewpoint_temperature'],
    target_months=[3, 4, 5],
    area=[70, -11, 30, 120],
    years=np.arange(1981, 2021),
    folder='../Siberia_example/SEAS5/')
```

[ ]:
```
retrieve.retrieve_ERA5(variables = ['2m_temperature','2m_dewpoint_temperature'],
                       target_months = [3,4,5],
                       area = [70, -11, 30, 120],
                       folder = '../Siberia_example/ERA5/')
```

And for the UK example:

[ ]:
```
retrieve.retrieve_SEAS5(variables = 'total_precipitation',
                        target_months = [2],
                        area = [60, -11, 50, 2],
                        folder = '../UK_example/SEAS5/')
```

[ ]:
```
retrieve.retrieve_ERA5(variables = 'total_precipitation',
                       target_months = [2],
                       area = [60, -11, 50, 2],
                       folder = '../UK_example/ERA5/')
```

EOBS data download

I tried to download EOBS through CDS, but the Product is temporally disabled for maintenance purposes (see below). As workaround I downloaded EOBS (from 1950 - 2019) and the most recent EOBS data (2020) here. Note, you have to register as E-OBS user.

[99]:
```
c.retrieve(
    'insitu-gridded-observations-europe',
    {
        'version': 'v20.0e',
        'format': 'zip',
        'product_type': 'ensemble_mean',
        'variable': 'precipitation_amount',
        'grid_resolution': '0_25',
        'period': 'full_period',
    },
    '../UK_example/EOBS/EOBS.zip')
```

```
2020-05-15 14:06:44,721 INFO Welcome to the CDS
2020-05-15 14:06:44,722 INFO Sending request to https://cds.climate.copernicus.eu/api/
↪v2/resources/insitu-gridded-observations-europe
```

```
---------------------------------------------------------------------------
HTTPError                                 Traceback (most recent call last)
~/.conda/envs/UNSEEN-open/lib/python3.8/site-packages/cdsapi/api.py in _api(self, url,
↪ request, method)
    388         try:
--> 389             result.raise_for_status()
    390             reply = result.json()

~/.conda/envs/UNSEEN-open/lib/python3.8/site-packages/requests/models.py in raise_for_
↪status(self)
    940         if http_error_msg:
--> 941             raise HTTPError(http_error_msg, response=self)
    942

HTTPError: 403 Client Error:  for url: https://cds.climate.copernicus.eu/api/v2/
↪resources/insitu-gridded-observations-europe

During handling of the above exception, another exception occurred:

Exception                                 Traceback (most recent call last)
<ipython-input-99-d12768b41b79> in <module>
----> 1 c.retrieve(
      2     'insitu-gridded-observations-europe',
      3     {
```

(continues on next page)

(continued from previous page)

```
      4         'version': 'v20.0e',
      5         'format': 'zip',

~/.conda/envs/UNSEEN-open/lib/python3.8/site-packages/cdsapi/api.py in retrieve(self,␣
↪name, request, target)
    315
    316     def retrieve(self, name, request, target=None):
--> 317         result = self._api('%s/resources/%s' % (self.url, name), request,
↪'POST')
    318         if target is not None:
    319             result.download(target)

~/.conda/envs/UNSEEN-open/lib/python3.8/site-packages/cdsapi/api.py in _api(self, url,
↪ request, method)
    408                                     "of '%s' at %s" % (t['title'], t['url']))
    409                     error = '. '.join(e)
--> 410                 raise Exception(error)
    411             else:
    412                 raise

Exception: Product temporally disabled for maintenance purposes. Sorry for the␣
↪inconvenience, please try again later.
```

1.5 Preprocess

The preprocessing steps consist of merging all retrieved files into one xarray dataset and extracting the spatial and temporal average of the event of interest.

1.5.1 Merge

Here it is shown how all retrieved files are loaded into one xarray dataset, for both SEAS5 and for ERA5.

SEAS5

All retrieved seasonal forecasts are loaded into one xarray dataset. The amount of files retrieved depends on the temporal extent of the extreme event that is being analyzed (i.e are you looking at a monthly average or a seasonal average?). For the Siberian heatwave, we have retrieved 105 files (one for each of the 35 years and for each of the three lead times, (*see Retrieve*). For the UK, we are able to use more forecasts, because the target month is shorter: one month as compared to three months for the Siberian example. We retrieved 5 leadtimes x 35 = 175 files.

Each netcdf file contains 25 ensemble members, hence has the dimensions lat, lon, number (25 ensembles). Here we create an xarray dataset that also contains the dimensions time (35 years) and leadtime (5 initialization months). To generate this, we loop over lead times, and open all 35 years of the lead time and then concatenate those leadtimes.

[1]:
```
##This is so variables get printed within jupyter
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

[2]:
```
import os
import sys
```

(continues on next page)

(continued from previous page)

```
sys.path.insert(0, os.path.abspath('../../../'))
import src.cdsretrieve as retrieve
```

[3]:
```
os.chdir(os.path.abspath('../../../'))
os.getcwd() #print the working directory
```

[3]:
```
'/lustre/soge1/projects/ls/personal/timo/UNSEEN-open'
```

[4]:
```
import xarray as xr
import numpy as np

def merge_SEAS5(folder, target_months):
    init_months, leadtimes = retrieve._get_init_months(target_months)
    print('Lead time: ' + "%.2i" % init_months[0])
    SEAS5_ld1 = xr.open_mfdataset(
        folder + '*' + "%.2i" % init_months[0] + '.nc',
        combine='by_coords')  # Load the first lead time
    SEAS5 = SEAS5_ld1  # Create the xarray dataset to concatenate over
    for init_month in init_months[1:len(init_months)]:  ## Remove the first that we␣
↪already have
        print(init_month)
        SEAS5_ld = xr.open_mfdataset(
            folder + '*' + "%.2i" % init_month + '.nc',
            combine='by_coords')
        SEAS5 = xr.concat([SEAS5, SEAS5_ld], dim='leadtime')
    SEAS5 = SEAS5.assign_coords(leadtime = np.arange(len(init_months)) + 2) # assign␣
↪leadtime coordinates
    return(SEAS5)
```

[5]:
```
SEAS5_Siberia = merge_SEAS5(folder='../Siberia_example/SEAS5/',
                            target_months=[3, 4, 5])
```

```
Lead time: 02
1
12
```

[6]:
```
SEAS5_Siberia
```

[6]:
```
<xarray.Dataset>
Dimensions:    (latitude: 41, leadtime: 3, longitude: 132, number: 51, time: 117)
Coordinates:
  * longitude  (longitude) float32 -11.0 -10.0 -9.0 -8.0 ... 118.0 119.0 120.0
  * latitude   (latitude) float32 70.0 69.0 68.0 67.0 ... 33.0 32.0 31.0 30.0
  * number     (number) int64 0 1 2 3 4 5 6 7 8 9 ... 42 43 44 45 46 47 48 49 50
  * time       (time) datetime64[ns] 1982-03-01 1982-04-01 ... 2020-05-01
  * leadtime   (leadtime) int64 2 3 4
Data variables:
    t2m        (leadtime, time, number, latitude, longitude) float32 dask.array
↪<chunksize=(1, 3, 51, 41, 132), meta=np.ndarray>
    d2m        (leadtime, time, number, latitude, longitude) float32 dask.array
↪<chunksize=(1, 3, 51, 41, 132), meta=np.ndarray>
Attributes:
    Conventions:  CF-1.6
    history:      2020-09-08 09:33:24 GMT by grib_to_netcdf-2.16.0: /opt/ecmw...
```

You can for example select a the lat, long, time, ensemble member and lead time as follows (add `.load()` to see the values):

```
[ ]: SEAS5_Siberia.sel(latitude=60,
                       longitude=-10,
                       time='2000-03',
                       number=26,
                       leadtime=3).load()
```

We can repeat this for the UK example, where just February is the target month:

```
[10]: SEAS5_UK = merge_SEAS5(folder = '../UK_example/SEAS5/', target_months = [2])
```

```
Lead time: 01
12
11
10
9
```

The SEAS5 total precipitation rate is in m/s. You can easily convert this and change the attributes. Click on the show/hide attributes button to see the assigned attributes.

```
[11]: SEAS5_UK['tprate'] = SEAS5_UK['tprate'] * 1000 * 3600 * 24 ## From m/s to mm/d
SEAS5_UK['tprate'].attrs = {'long_name': 'rainfall',
 'units': 'mm/day',
 'standard_name': 'thickness_of_rainfall_amount'}
SEAS5_UK
```

```
[11]: <xarray.Dataset>
Dimensions:    (latitude: 11, leadtime: 5, longitude: 14, number: 25, time: 35)
Coordinates:
  * time       (time) datetime64[ns] 1982-02-01 1983-02-01 ... 2016-02-01
  * latitude   (latitude) float32 60.0 59.0 58.0 57.0 ... 53.0 52.0 51.0 50.0
  * number     (number) int32 0 1 2 3 4 5 6 7 8 9 ... 16 17 18 19 20 21 22 23 24
  * longitude  (longitude) float32 -11.0 -10.0 -9.0 -8.0 ... -1.0 0.0 1.0 2.0
  * leadtime   (leadtime) int64 2 3 4 5 6
Data variables:
    tprate     (leadtime, time, number, latitude, longitude) float32 dask.array
↪<chunksize=(1, 1, 25, 11, 14), meta=np.ndarray>
Attributes:
    Conventions:  CF-1.6
    history:      2020-05-13 14:49:43 GMT by grib_to_netcdf-2.16.0: /opt/ecmw...
```

ERA5

For each year a netcdf file is downloaded. They are named ERA5_yyyy, for example ERA5_1981. Therefore, we can load ERA5 by combining all downloaded years:

```
[9]: ERA5_Siberia = xr.open_mfdataset('../Siberia_example/ERA5/ERA5_????.nc',combine='by_
↪coords') ## open the data
ERA5_Siberia
```

```
[9]: <xarray.Dataset>
Dimensions:    (latitude: 41, longitude: 132, time: 126)
Coordinates:
  * longitude  (longitude) float32 -11.0 -10.0 -9.0 -8.0 ... 118.0 119.0 120.0
  * latitude   (latitude) float32 70.0 69.0 68.0 67.0 ... 33.0 32.0 31.0 30.0
  * time       (time) datetime64[ns] 1979-03-01 1979-04-01 ... 2020-05-01
Data variables:
    t2m        (time, latitude, longitude) float32 dask.array<chunksize=(3, 41, 132),␣
↪meta=np.ndarray>
```

(continues on next page)

(continued from previous page)

```
    d2m        (time, latitude, longitude) float32 dask.array<chunksize=(3, 41, 132),␣
↪meta=np.ndarray>
Attributes:
    Conventions:  CF-1.6
    history:      2020-09-08 13:26:08 GMT by grib_to_netcdf-2.16.0: /opt/ecmw...
```

[13]:
```
ERA5_UK = xr.open_mfdataset('../UK_example/ERA5/ERA5_????.nc',combine='by_coords') ##␣
↪open the data
ERA5_UK
```

[13]:
```
<xarray.Dataset>
Dimensions:    (latitude: 11, longitude: 14, time: 42)
Coordinates:
  * latitude   (latitude) float32 60.0 59.0 58.0 57.0 ... 53.0 52.0 51.0 50.0
  * longitude  (longitude) float32 -11.0 -10.0 -9.0 -8.0 ... -1.0 0.0 1.0 2.0
  * time       (time) datetime64[ns] 1979-02-01 1980-02-01 ... 2020-02-01
Data variables:
    tp         (time, latitude, longitude) float32 dask.array<chunksize=(1, 11, 14),␣
↪meta=np.ndarray>
Attributes:
    Conventions:  CF-1.6
    history:      2020-09-08 13:36:54 GMT by grib_to_netcdf-2.16.0: /opt/ecmw...
```

1.5.2 Event definition

Time selection

For the UK, the event of interest is UK February average precipitation. Since we download monthly averages, we do not have to do any preprocessing along the time dimension here. For the Siberian heatwave, we are interested in the March-May average. Therefore we need to take the seasonal average of the monthly timeseries. We cannot take the simple mean of the three months, because they have a different number of days in the months, see this example. Therefore we take a weighted average:

[11]:
```
month_length = SEAS5_Siberia.time.dt.days_in_month
month_length
```

[11]:
```
<xarray.DataArray 'days_in_month' (time: 117)>
array([31, 30, 31, 31, 30, 31, 31, 30, 31, 31, 30, 31, 31, 30, 31, 31, 30,
       31, 31, 30, 31, 31, 30, 31, 31, 30, 31, 31, 30, 31, 31, 30, 31, 31,
       30, 31, 31, 30, 31, 31, 30, 31, 31, 30, 31, 31, 30, 31, 31, 30, 31,
       31, 30, 31, 31, 30, 31, 31, 30, 31, 31, 30, 31, 31, 30, 31, 31, 30,
       31, 31, 30, 31, 31, 30, 31, 31, 30, 31, 31, 30, 31, 31, 30, 31, 31,
       30, 31, 31, 30, 31, 31, 30, 31, 31, 30, 31, 31, 30, 31, 31, 30, 31,
       31, 30, 31, 31, 30, 31, 31, 30, 31, 31, 30, 31, 31, 30, 31])
Coordinates:
  * time     (time) datetime64[ns] 1982-03-01 1982-04-01 ... 2020-05-01
```

[12]:
```
# Calculate the weights by grouping by 'time.season'.
weights = month_length.groupby('time.year') / month_length.groupby('time.year').sum()
weights
```

[12]:
```
<xarray.DataArray 'days_in_month' (time: 117)>
array([0.33695652, 0.32608696, 0.33695652, 0.33695652, 0.32608696,
       0.33695652, 0.33695652, 0.32608696, 0.33695652, 0.33695652,
       0.32608696, 0.33695652, 0.33695652, 0.32608696, 0.33695652,
```

(continues on next page)

(continued from previous page)

```
        0.33695652, 0.32608696, 0.33695652, 0.33695652, 0.32608696,
        0.33695652, 0.33695652, 0.32608696, 0.33695652, 0.33695652,
        0.32608696, 0.33695652, 0.33695652, 0.32608696, 0.33695652,
        0.33695652, 0.32608696, 0.33695652, 0.33695652, 0.32608696,
        0.33695652, 0.33695652, 0.32608696, 0.33695652, 0.33695652,
        0.32608696, 0.33695652, 0.33695652, 0.32608696, 0.33695652,
        0.33695652, 0.32608696, 0.33695652, 0.33695652, 0.32608696,
        0.33695652, 0.33695652, 0.32608696, 0.33695652, 0.33695652,
        0.32608696, 0.33695652, 0.33695652, 0.32608696, 0.33695652,
        0.33695652, 0.32608696, 0.33695652, 0.33695652, 0.32608696,
        0.33695652, 0.33695652, 0.32608696, 0.33695652, 0.33695652,
        0.32608696, 0.33695652, 0.33695652, 0.32608696, 0.33695652,
        0.33695652, 0.32608696, 0.33695652, 0.33695652, 0.32608696,
        0.33695652, 0.33695652, 0.32608696, 0.33695652, 0.33695652,
        0.32608696, 0.33695652, 0.33695652, 0.32608696, 0.33695652,
        0.33695652, 0.32608696, 0.33695652, 0.33695652, 0.32608696,
        0.33695652, 0.33695652, 0.32608696, 0.33695652, 0.33695652,
        0.32608696, 0.33695652, 0.33695652, 0.32608696, 0.33695652,
        0.33695652, 0.32608696, 0.33695652, 0.33695652, 0.32608696,
        0.33695652, 0.33695652, 0.32608696, 0.33695652, 0.33695652,
        0.32608696, 0.33695652])
Coordinates:
  * time     (time) datetime64[ns] 1982-03-01 1982-04-01 ... 2020-05-01
    year     (time) int64 1982 1982 1982 1983 1983 ... 2019 2019 2020 2020 2020
```

[13]:

```
# Test that the sum of the weights for the season is 1.0
np.testing.assert_allclose(weights.groupby('time.year').sum().values, np.ones(39)) ##␣
↪the weight is one for each year
```

[14]:

```
# Calculate the weighted average
SEAS5_Siberia_weighted = (SEAS5_Siberia * weights).groupby('time.year').sum(dim='time
↪', min_count = 3)
SEAS5_Siberia_weighted
```

[14]:

```
<xarray.Dataset>
Dimensions:    (latitude: 41, leadtime: 3, longitude: 132, number: 51, year: 39)
Coordinates:
  * longitude  (longitude) float32 -11.0 -10.0 -9.0 -8.0 ... 118.0 119.0 120.0
  * leadtime   (leadtime) int64 2 3 4
  * latitude   (latitude) float32 70.0 69.0 68.0 67.0 ... 33.0 32.0 31.0 30.0
  * number     (number) int64 0 1 2 3 4 5 6 7 8 9 ... 42 43 44 45 46 47 48 49 50
  * year       (year) int64 1982 1983 1984 1985 1986 ... 2017 2018 2019 2020
Data variables:
    t2m        (year, leadtime, number, latitude, longitude) float64 dask.array
↪<chunksize=(1, 1, 51, 41, 132), meta=np.ndarray>
    d2m        (year, leadtime, number, latitude, longitude) float64 dask.array
↪<chunksize=(1, 1, 51, 41, 132), meta=np.ndarray>
```

Or as function:

[15]:

```
def season_mean(ds, years, calendar='standard'):
    # Make a DataArray with the number of days in each month, size = len(time)
    month_length = ds.time.dt.days_in_month

    # Calculate the weights by grouping by 'time.season'
    weights = month_length.groupby('time.year') / month_length.groupby('time.year').
↪sum()
```

(continues on next page)

(continued from previous page)

```
    # Test that the sum of the weights for each season is 1.0
    np.testing.assert_allclose(weights.groupby('time.year').sum().values, np.
↪ones(years))

    # Calculate the weighted average
    return (ds * weights).groupby('time.year').sum(dim='time', min_count = 3)
```

[16]:
```
ERA5_Siberia_weighted = season_mean(ERA5_Siberia, years = 42)
ERA5_Siberia_weighted
```

[16]:
```
<xarray.Dataset>
Dimensions:    (latitude: 41, longitude: 132, year: 42)
Coordinates:
  * longitude  (longitude) float32 -11.0 -10.0 -9.0 -8.0 ... 118.0 119.0 120.0
  * latitude   (latitude) float32 70.0 69.0 68.0 67.0 ... 33.0 32.0 31.0 30.0
  * year       (year) int64 1979 1980 1981 1982 1983 ... 2017 2018 2019 2020
Data variables:
    t2m        (year, latitude, longitude) float64 dask.array<chunksize=(1, 41, 132),␣
↪meta=np.ndarray>
    d2m        (year, latitude, longitude) float64 dask.array<chunksize=(1, 41, 132),␣
↪meta=np.ndarray>
```

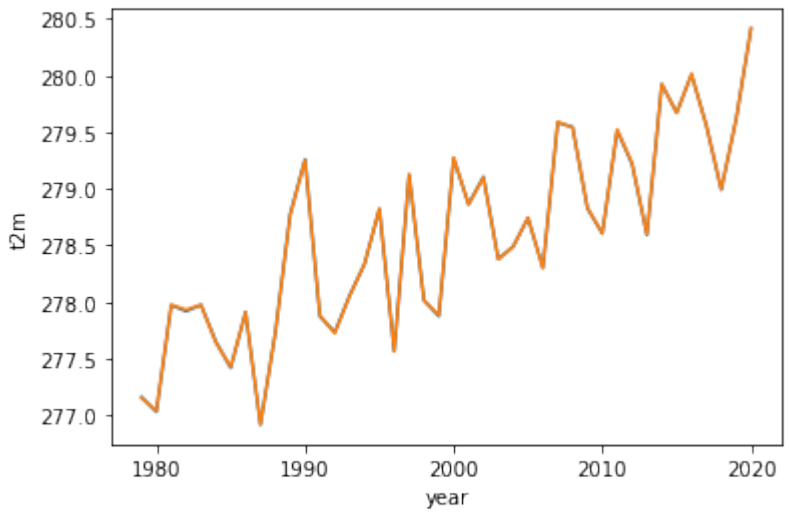
What is the difference between the mean and weighted mean?

Barely visible the difference

[17]:
```
ERA5_Siberia_weighted['t2m'].mean(['longitude', 'latitude']).plot()
ERA5_Siberia['t2m'].groupby('time.year').mean().mean(['longitude','latitude']).plot()
```

[17]:
```
[<matplotlib.lines.Line2D at 0x7fa13055f970>]
```

[17]:
```
[<matplotlib.lines.Line2D at 0x7fa130531be0>]
```



Spatial selection

What spatial extent defines the event you are analyzing? The easiest option is to select a lat-lon box, like we did for the Siberian heatwave example (i.e we average the temperature over 50-70N, 65-120E, also used here).

In case you want to specify another domain than a lat-lon box, you could mask the datasets. For the California Fires example, we select the domain with high temperature anomalies (>2 standard deviation), see California_august_temperature_anomaly. For the UK example, we want a country-averaged timeseries instead of a box. In this case, we use another observational product: the EOBS dataset that covers Europe. We *upscale* this dataset to the same resolution as SEAS5 and create a *mask* to take the spatial average over the UK, see *Using EOBS + upscaling*.

We have to take the latitude-weighted average, since since grid cell area decreases with latitude. We first take the 'normal' average:

[36]:
```
ERA5_Siberia_events_zoomed = (
    ERA5_Siberia_weighted['t2m'].sel(  # Select 2 metre temperature
        latitude=slice(70, 50),        # Select the latitudes
        longitude=slice(65, 120)).    # Select the longitude
    mean(['longitude', 'latitude']))
```

And we repeat this for the SEAS5 events

[37]:
```
SEAS5_Siberia_events = (
    SEAS5_Siberia_weighted['t2m'].sel(
        latitude=slice(70, 30),
        longitude=slice(-11, 120)).
    mean(['longitude', 'latitude']))
SEAS5_Siberia_events.load()
```

```
/soge-home/users/cenv0732/.conda/envs/UNSEEN-open/lib/python3.8/site-packages/xarray/
↪core/nanops.py:142: RuntimeWarning: Mean of empty slice
  return np.nanmean(a, axis=axis, dtype=dtype)
```

[37]:
```
<xarray.DataArray 't2m' (year: 39, leadtime: 3, number: 51)>
array([[[277.70246026, 277.08011538, 277.05805243, ...,          nan,
                 nan,          nan],
        [276.54063304, 277.63527276, 276.53540684, ...,          nan,
                 nan,          nan],
        [276.94382457, 277.22540106, 277.25375804, ...,          nan,
                 nan,          nan]],

       [[276.68638666, 276.64418409, 276.88169219, ...,          nan,
                 nan,          nan],
        [277.06362955, 277.30470221, 276.49967939, ...,          nan,
                 nan,          nan],
        [276.37166345, 276.63563118, 277.05456392, ...,          nan,
                 nan,          nan]],

       [[277.53103277, 277.49691758, 278.32115366, ...,          nan,
                 nan,          nan],
        [277.53911427, 278.11393678, 277.66278741, ...,          nan,
                 nan,          nan],
        [278.24589375, 277.71341079, 277.3067712 , ...,          nan,
                 nan,          nan]],

       ...,

       [[278.78906418, 278.0626699 , 278.09438675, ..., 278.10947691,
         278.27620377, 278.18620179],
```

(continues on next page)

(continued from previous page)

```
        [278.65929139, 277.33954004, 278.65951576, ..., 278.22959696,
         278.32163068, 278.94724957],
        [278.90689211, 277.9030209 , 279.13818072, ..., 278.76767259,
         279.13397914, 277.76992423]],

       [[278.6218426 , 277.95006232, 278.22900254, ..., 277.56330007,
         277.99480916, 277.66857676],
        [278.39808792, 277.65889255, 277.92266928, ..., 278.39390445,
         277.90353039, 278.01793147],
        [278.63429219, 278.11630486, 278.58465727, ..., 277.7081865 ,
         277.73949614, 278.65482764]],

       [[279.28124227, 278.53474142, 278.47436518, ..., 278.93209185,
         278.11716261, 279.3904762 ],
        [277.77935773, 279.15571385, 279.02168652, ..., 279.25803913,
         278.8991169 , 278.72803803],
        [278.54721722, 278.25816177, 279.65502139, ..., 279.01242149,
         278.80174459, 278.23615329]]])
Coordinates:
  * leadtime  (leadtime) int64 2 3 4
  * number    (number) int64 0 1 2 3 4 5 6 7 8 9 ... 42 43 44 45 46 47 48 49 50
  * year      (year) int64 1982 1983 1984 1985 1986 ... 2016 2017 2018 2019 2020
```

[39]:
```
SEAS5_Siberia_events_zoomed = (
    SEAS5_Siberia_weighted['t2m'].sel(
        latitude=slice(70, 50),
        longitude=slice(65, 120)).
    mean(['longitude', 'latitude']))
SEAS5_Siberia_events_zoomed.load()
```

[39]:
```
<xarray.DataArray 't2m' (year: 39, leadtime: 3, number: 51)>
array([[[269.41349502, 267.46724112, 268.92858923, ...,          nan,
                 nan,          nan],
        [267.30541714, 269.45786213, 267.88083134, ...,          nan,
                 nan,          nan],
        [266.70463988, 269.36559057, 267.85380896, ...,          nan,
                 nan,          nan]],

       [[267.65255078, 267.85459971, 267.93397041, ...,          nan,
                 nan,          nan],
        [267.86908721, 269.16875401, 266.25505375, ...,          nan,
                 nan,          nan],
        [267.37705396, 268.0216673 , 269.216725  , ...,          nan,
                 nan,          nan]],

       [[269.11559244, 270.30993792, 268.97992022, ...,          nan,
                 nan,          nan],
        [268.42632866, 270.23730451, 268.14872887, ...,          nan,
                 nan,          nan],
        [269.91675564, 269.37623754, 268.62430776, ...,          nan,
                 nan,          nan]],

       ...,

       [[270.1395106 , 269.2763681 , 269.63408345, ..., 267.98167445,
         270.99148119, 269.2334804 ],
        [269.47838693, 267.62436995, 270.46814617, ..., 269.60061317,
```

(continues on next page)

(continued from previous page)

```
         269.2030878 , 270.34174847],
        [271.39762301, 268.06102893, 271.27933216, ..., 269.56866515,
         270.64943148, 266.36754614]],

       [[269.32829344, 268.54758963, 269.10385302, ..., 268.03762412,
         269.16690743, 268.41760566],
        [269.09579833, 268.27844902, 268.80430384, ..., 269.38437353,
         269.58112924, 269.15390309],
        [269.2755198 , 269.28402279, 270.34429505, ..., 268.05044769,
         269.01128231, 270.09822017]],

       [[271.35447849, 269.73507649, 269.16852164, ..., 270.77523033,
         268.30803885, 271.87237937],
        [269.33923167, 270.81122764, 269.97102764, ..., 272.56708782,
         270.69195734, 269.90622653],
        [270.37695877, 269.72247595, 272.524012  , ..., 270.04026793,
         269.81585811, 267.24061429]]])
Coordinates:
  * leadtime  (leadtime) int64 2 3 4
  * number    (number) int64 0 1 2 3 4 5 6 7 8 9 ... 42 43 44 45 46 47 48 49 50
  * year      (year) int64 1982 1983 1984 1985 1986 ... 2016 2017 2018 2019 2020
```

[40]:
```
SEAS5_Siberia_events.to_dataframe().to_csv('Data/SEAS5_Siberia_events.csv')
ERA5_Siberia_events.to_dataframe().to_csv('Data/ERA5_Siberia_events.csv')
```

[41]:
```
SEAS5_Siberia_events_zoomed.to_dataframe().to_csv('Data/SEAS5_Siberia_events_zoomed.
↪csv')
ERA5_Siberia_events_zoomed.to_dataframe().to_csv('Data/ERA5_Siberia_events_zoomed.csv
↪')
```

1.6 Evaluate

Can seasonal forecasts be used as 'alternate' realities? Here we show how a set of evaluation metrics can be used to answer this question. The evaluation metrics are available through an R package for easy evaluation of the UNSEEN ensemble. Here, we illustrate how this package can be used in the UNSEEN workflow. We will evaluate the generated UNSEEN ensemble of UK February precipitation and of MAM Siberian heatwaves.

The framework to evaluate the UNSEEN ensemble presented here consists of testing the ensemble member independence, model stability and model fidelity, see also NPJ paper.

Note

This is R code and not python!

We switch to R since we believe R has a better functionality in extreme value statistics.

We load the UNSEEN package and read in the data.

[2]:
```
library(UNSEEN)
```

The data that is imported here are the files stored at the end of the *preprocessing step*.

[3]:
```
SEAS5_Siberia_events <- read.csv("Data/SEAS5_Siberia_events.csv",␣
↪stringsAsFactors=FALSE)
ERA5_Siberia_events <- read.csv("Data/ERA5_Siberia_events.csv",␣
↪stringsAsFactors=FALSE)
```

[4]:
```
SEAS5_Siberia_events_zoomed <- read.csv("Data/SEAS5_Siberia_events_zoomed.csv",␣
↪stringsAsFactors=FALSE)
ERA5_Siberia_events_zoomed <- read.csv("Data/ERA5_Siberia_events_zoomed.csv",␣
↪stringsAsFactors=FALSE)
```

[5]:
```
SEAS5_Siberia_events$t2m <- SEAS5_Siberia_events$t2m - 273.15
ERA5_Siberia_events$t2m <- ERA5_Siberia_events$t2m - 273.15
SEAS5_Siberia_events_zoomed$t2m <- SEAS5_Siberia_events_zoomed$t2m - 273.15
ERA5_Siberia_events_zoomed$t2m <- ERA5_Siberia_events_zoomed$t2m - 273.15
```

[6]:
```
head(SEAS5_Siberia_events_zoomed,n = 3)
head(ERA5_Siberia_events, n = 3)
```

A data.frame: 3 × 4

	year <int>	leadtime <int>	number <int>	t2m <dbl>
1	1982	2	0	-3.736505
2	1982	2	1	-5.682759
3	1982	2	2	-4.221411

A data.frame: 3 × 2

	year <int>	t2m <dbl>
1	1979	4.010750
2	1980	3.880965
3	1981	4.822891

[7]:
```
EOBS_UK_weighted_df <- read.csv("Data/EOBS_UK_weighted_upscaled.csv",␣
↪stringsAsFactors=FALSE)
SEAS5_UK_weighted_df <- read.csv("Data/SEAS5_UK_weighted_masked.csv",␣
↪stringsAsFactors=FALSE)
```

And then convert the time class to Date format, with the ymd function in lubridate:

[8]:
```
EOBS_UK_weighted_df$time <- lubridate::ymd(EOBS_UK_weighted_df$time)
str(EOBS_UK_weighted_df)

EOBS_UK_weighted_df_hindcast <- EOBS_UK_weighted_df[
    EOBS_UK_weighted_df$time > '1982-02-01' &
    EOBS_UK_weighted_df$time < '2017-02-01',
    ]


SEAS5_UK_weighted_df$time <- lubridate::ymd(SEAS5_UK_weighted_df$time)
str(SEAS5_UK_weighted_df)
```

```
'data.frame':   71 obs. of  2 variables:
 $ time: Date, format: "1950-02-28" "1951-02-28" ...
 $ rr  : num  4.13 3.25 1.07 1.59 2.59 ...
'data.frame':   9945 obs. of  4 variables:
 $ leadtime: int  2 2 2 2 2 2 2 2 2 2 ...
 $ number  : int  0 0 0 0 0 0 0 0 0 0 ...
```

(continues on next page)

(continued from previous page)

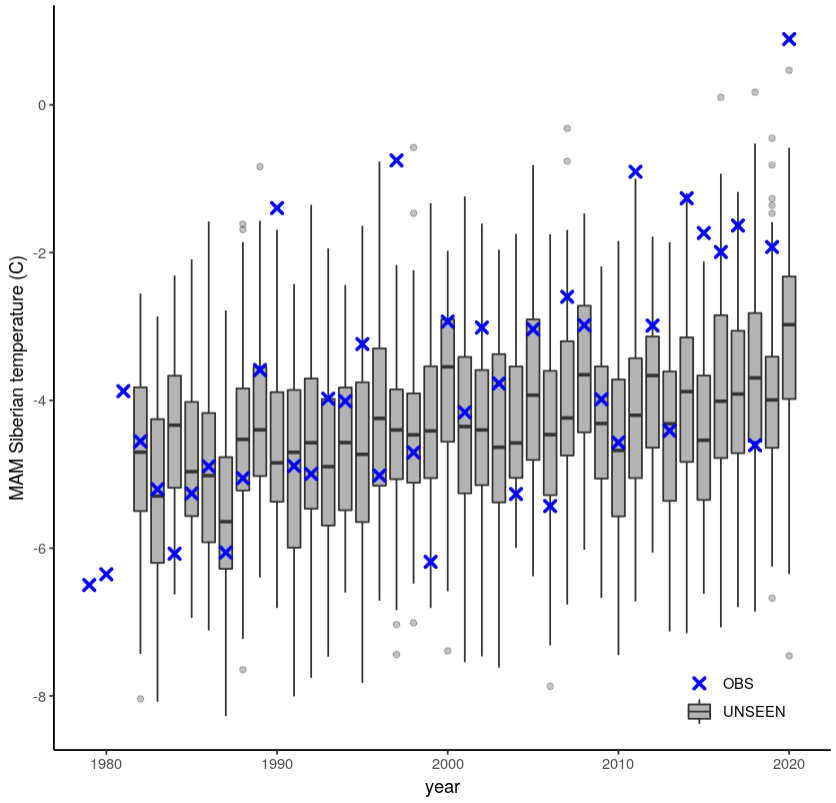
```
$ time    : Date, format: "1982-02-01" "1983-02-01" ...
$ tprate  : num  1.62 2.93 3.27 2 3.31 ...
```

1.6.1 Timeseries

Here we plot the timeseries of SEAS5 (UNSEEN) and ERA5 (OBS) for the Siberian Heatwave.

[10]:
```
unseen_timeseries(
    ensemble = SEAS5_Siberia_events_zoomed,
    obs = ERA5_Siberia_events_zoomed,
    ensemble_yname = "t2m",
    ensemble_xname = "year",
    obs_yname = "t2m",
    obs_xname = "year",
    ylab = "MAM Siberian temperature (C)")
```

```
Warning message:
"Removed 2756 rows containing non-finite values (stat_boxplot)."
```



The timeseries consist of hindcast (years 1982-2016) and archived forecasts (years 2017-2020). The datasets are slightly different: the hindcasts contains 25 members whereas operational forecasts contain 51 members, the native resolution is different and the dataset from which the forecasts are initialized is different.

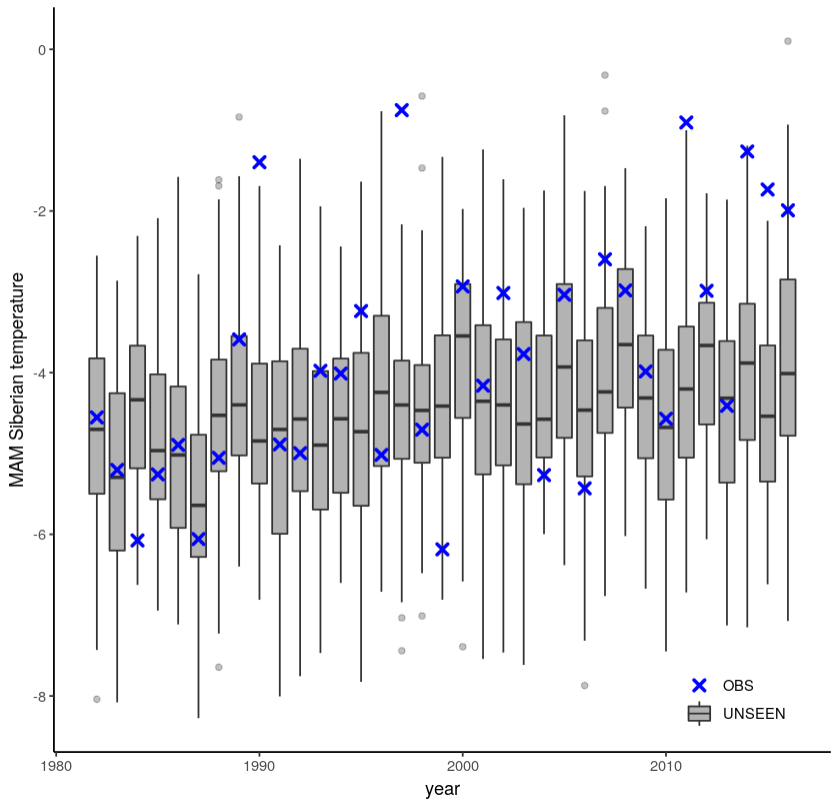
For the evaluation of the UNSEEN ensemble we want to only use the SEAS5 hindcasts for a consistent dataset. Note, 2017 is not used in either the hindcast nor the operational dataset in this example, since it contains forecasts both initialized in 2016 (hindcast) and 2017 (forecast), see *retrieve*. We split SEAS5 into hindcast and operational forecasts:

[11]:
```
SEAS5_Siberia_events_zoomed_hindcast <- SEAS5_Siberia_events_zoomed[
    SEAS5_Siberia_events_zoomed$year < 2017 &
    SEAS5_Siberia_events_zoomed$number < 25,]

SEAS5_Siberia_events_zoomed_forecasts <- SEAS5_Siberia_events_zoomed[
    SEAS5_Siberia_events_zoomed$year > 2017,]
```

And we select the same years for ERA5.

[12]:
```
ERA5_Siberia_events_zoomed_hindcast <- ERA5_Siberia_events_zoomed[
    ERA5_Siberia_events_zoomed$year < 2017 &
    ERA5_Siberia_events_zoomed$year > 1981,]
```

[13]:
```
unseen_timeseries(
    ensemble = SEAS5_Siberia_events_zoomed_hindcast,
    obs = ERA5_Siberia_events_zoomed_hindcast,
    ensemble_yname = "t2m",
    ensemble_xname = "year",
    obs_yname = "t2m",
    obs_xname = "year",
    ylab = "MAM Siberian temperature")
```

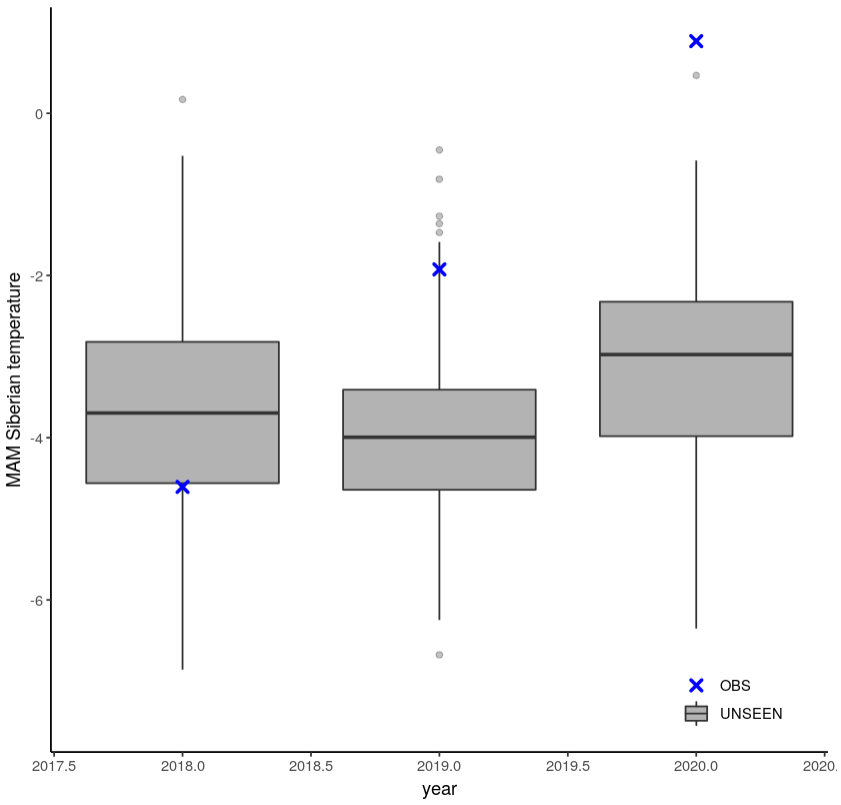


[14]:
```
unseen_timeseries(
    ensemble = SEAS5_Siberia_events_zoomed_forecasts,
    obs = ERA5_Siberia_events_zoomed[ERA5_Siberia_events_zoomed$year > 2017,],
    ensemble_yname = "t2m",
```

(continues on next page)

(continued from previous page)

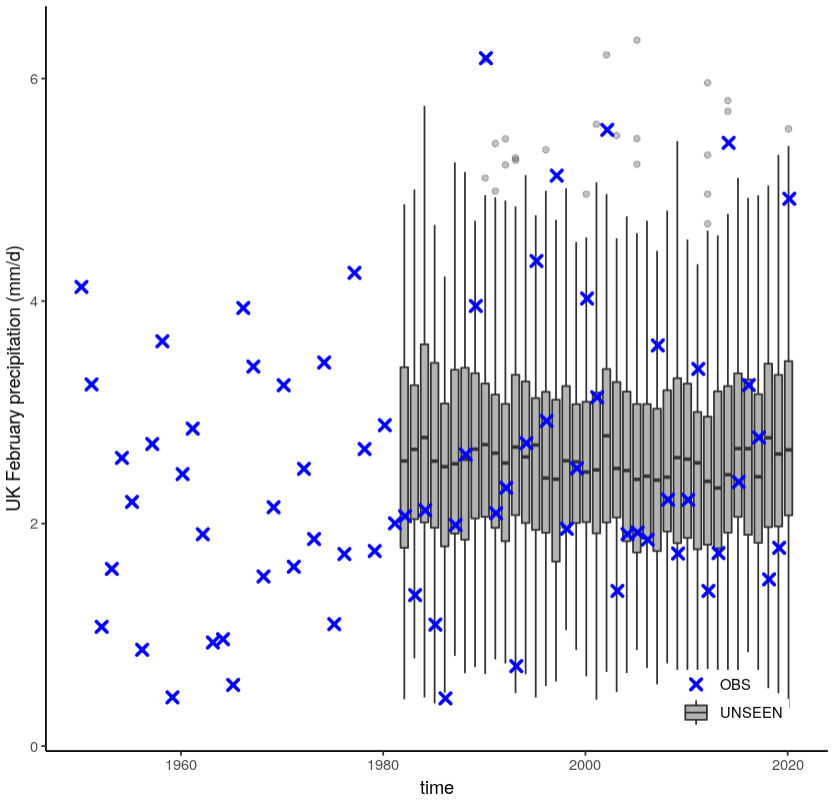
```
    ensemble_xname = "year",
    obs_yname = "t2m",
    obs_xname = "year",
    ylab = "MAM Siberian temperature")
```



For the UK we have a longer historical record available from EOBS:

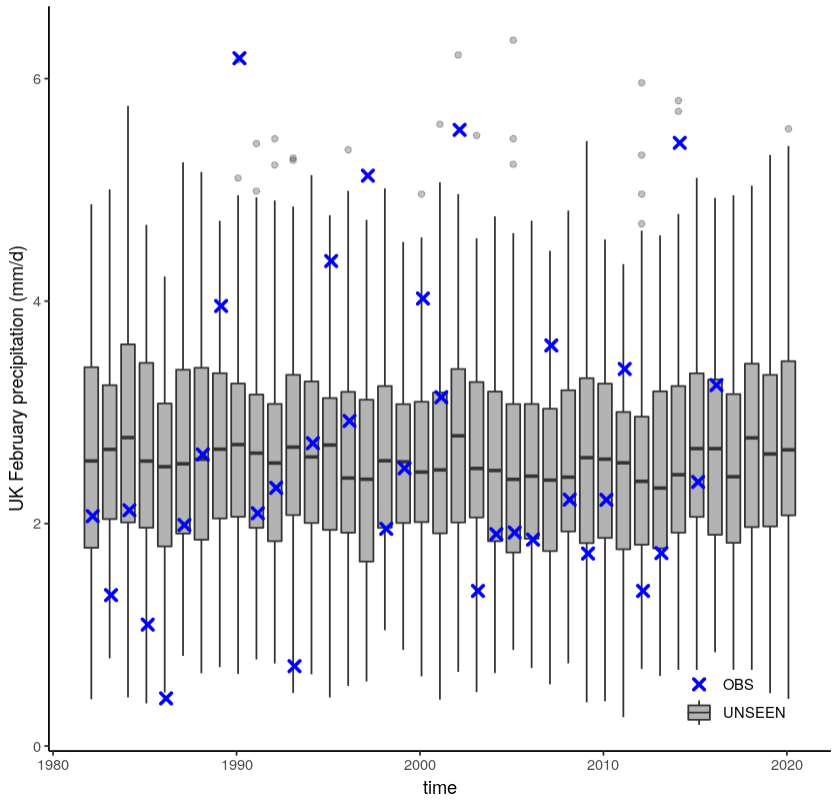
[15]:
```
unseen_timeseries(ensemble = SEAS5_UK_weighted_df,
                  obs = EOBS_UK_weighted_df,
                  ylab = 'UK February precipitation (mm/d)')
```

```
Warning message:
"Removed 4654 rows containing non-finite values (stat_boxplot)."
```



[16]:
```
unseen_timeseries(ensemble = SEAS5_UK_weighted_df,
                  obs = EOBS_UK_weighted_df_hindcast,
                  ylab = 'UK February precipitation (mm/d)')
```

```
Warning message:
"Removed 4654 rows containing non-finite values (stat_boxplot)."
```



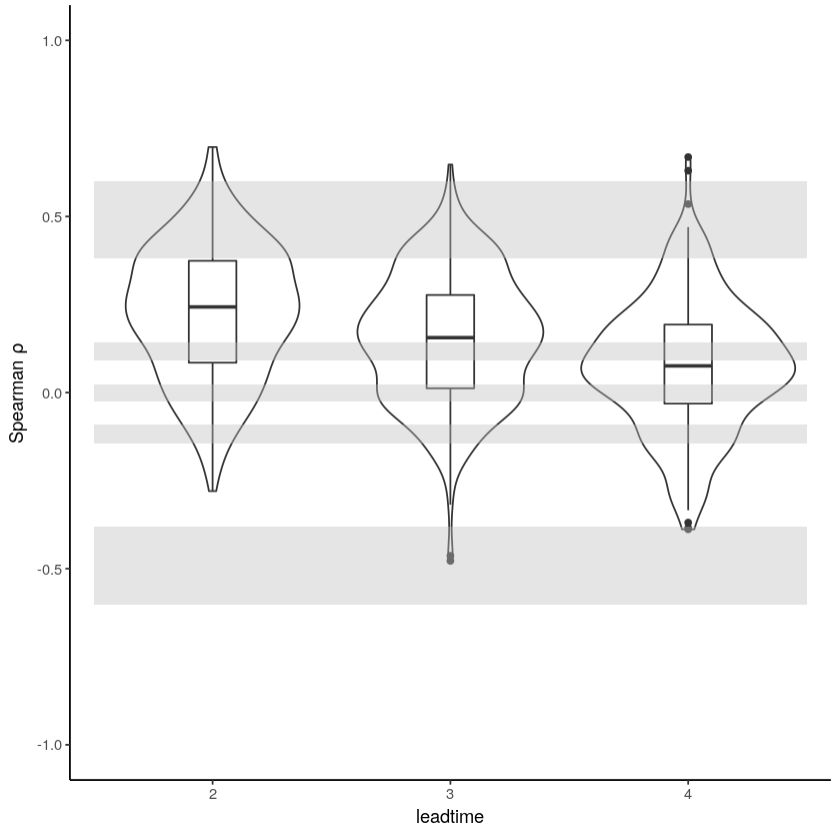
Call the documentation of the function with `?unseen_timeseries`

1.6.2 Independence

Significance ranges need fixing + detrend method (Rob)

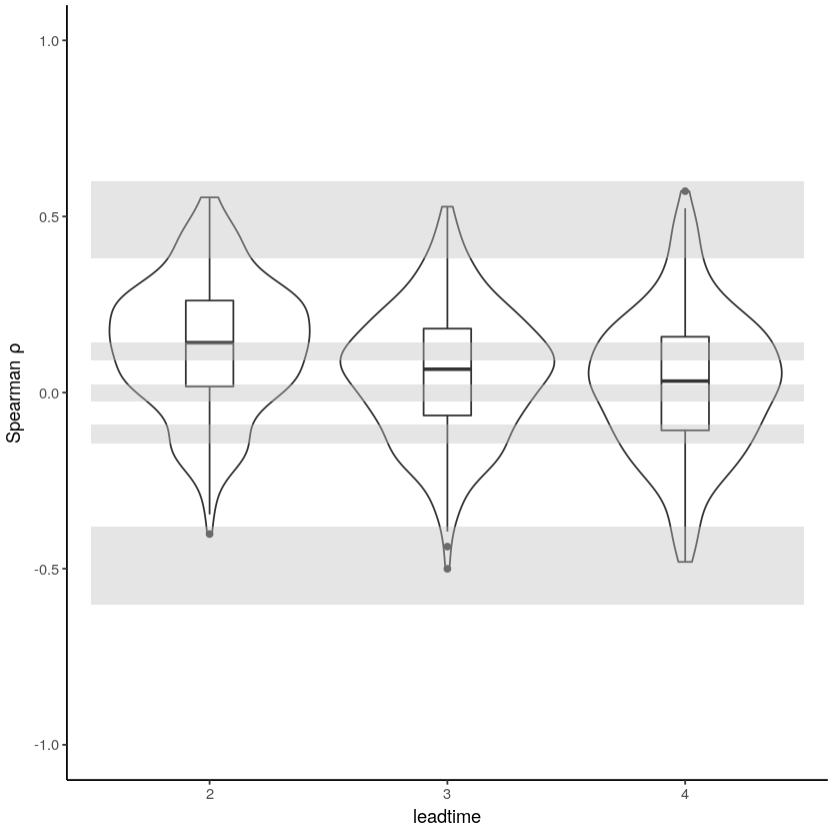
[17]:
```
independence_test(
    ensemble = SEAS5_Siberia_events,
    n_lds = 3,
    var_name = "t2m",
)
```

```
Warning message:
"Removed 975 rows containing non-finite values (stat_ydensity)."
Warning message:
"Removed 975 rows containing non-finite values (stat_boxplot)."
```



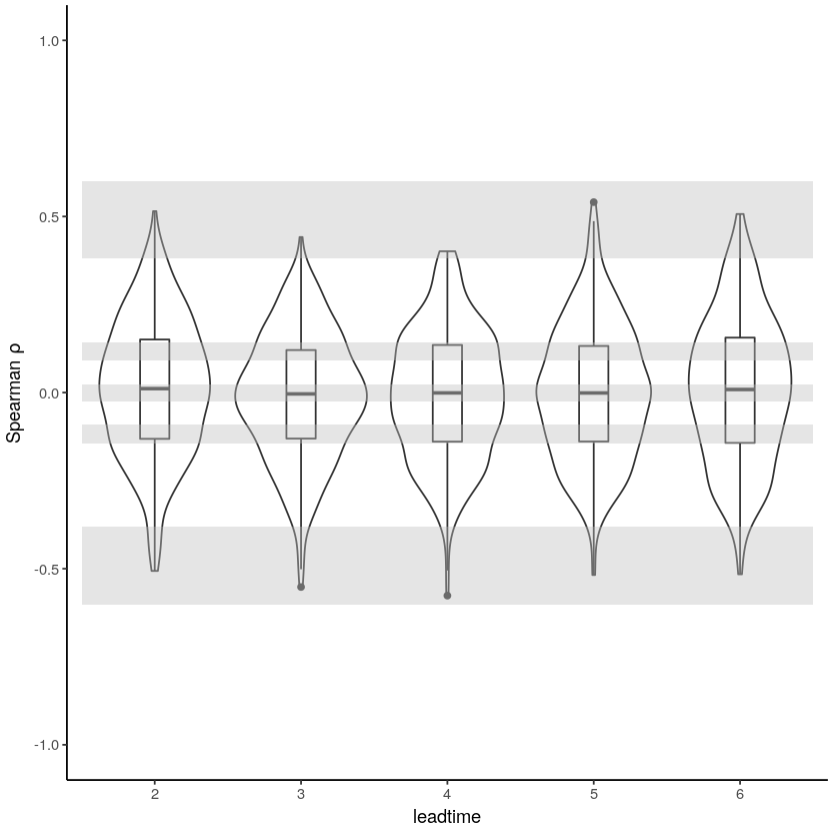
[18]:
```
independence_test(
    ensemble = SEAS5_Siberia_events_zoomed,
    n_lds = 3,
    var_name = "t2m",
)
```

```
Warning message:
“Removed 975 rows containing non-finite values (stat_ydensity).”
Warning message:
“Removed 975 rows containing non-finite values (stat_boxplot).”
```



[19]:
```
independence_test(ensemble = SEAS5_UK_weighted_df)
```

```
Warning message:
"Removed 1625 rows containing non-finite values (stat_ydensity)."
Warning message:
"Removed 1625 rows containing non-finite values (stat_boxplot)."
```

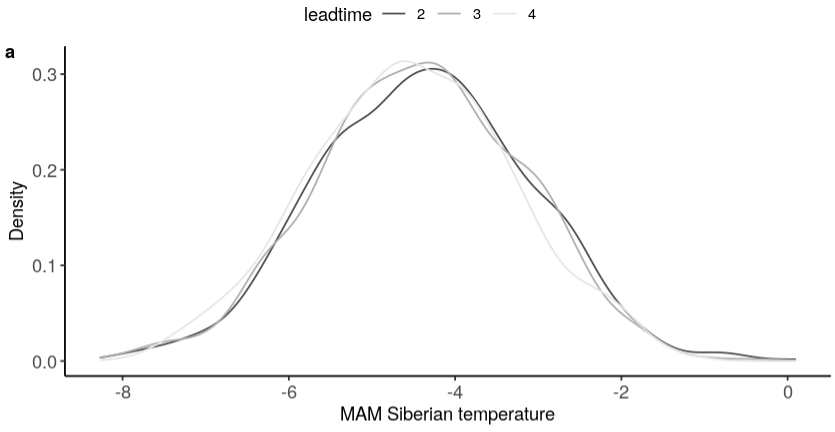


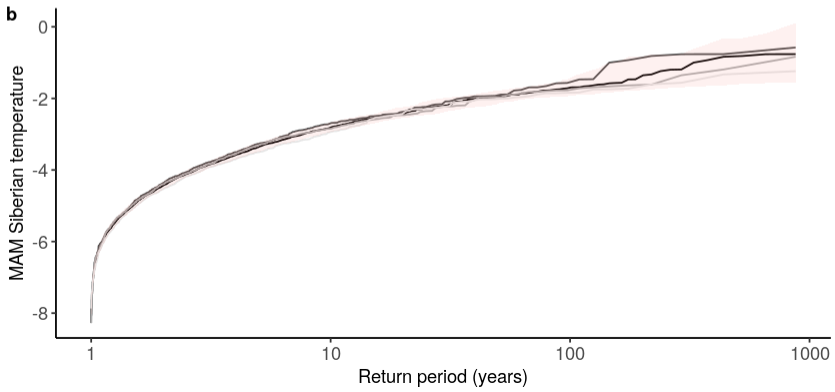
1.6.3 Stability

For the stability test we assess whether the events get more severe with leadtime, due to a potential 'drift' in the model. We need to use the consistent hindcast dataset for this.

[20]:
```
stability_test(
    ensemble = SEAS5_Siberia_events_zoomed_hindcast,
    lab = 'MAM Siberian temperature',
    var_name = 't2m'
)
```

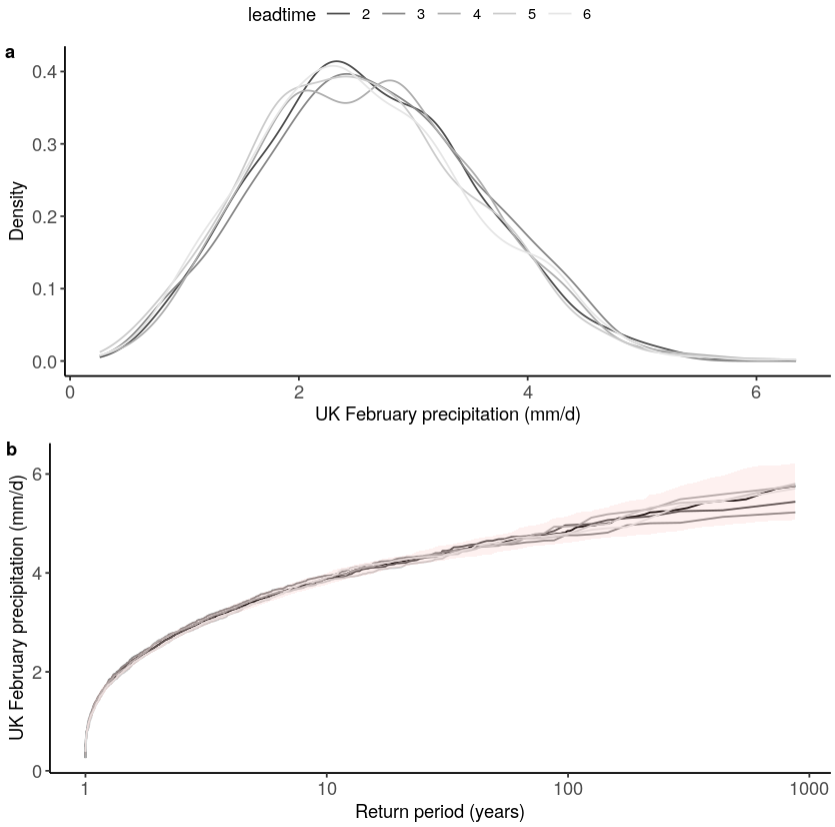
```
Warning message:
"Removed 2 row(s) containing missing values (geom_path)."
```





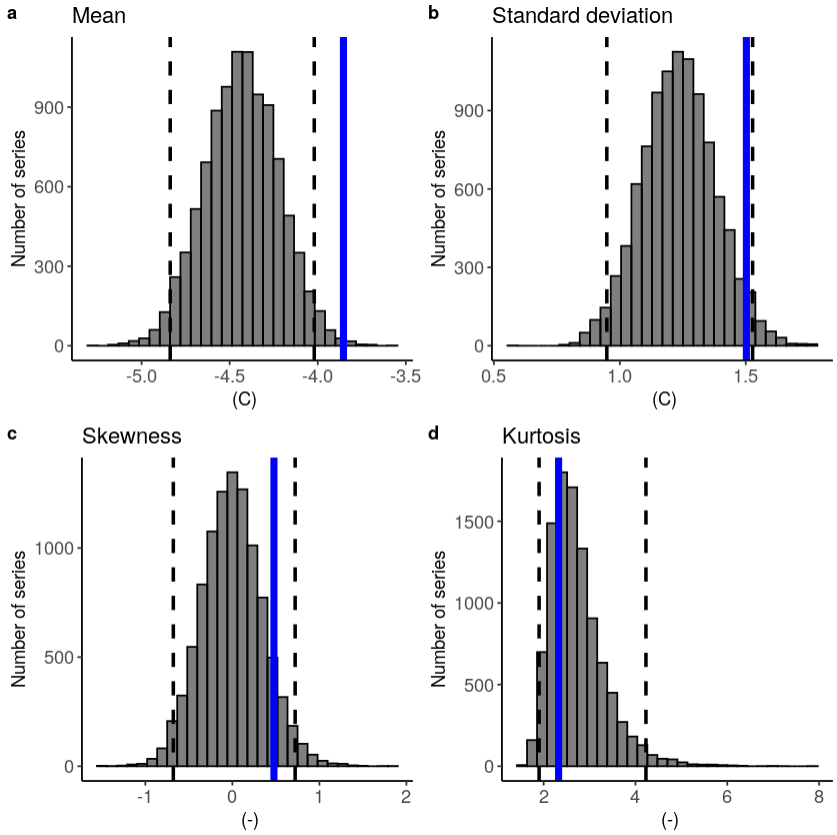
[21]:
```
stability_test(ensemble = SEAS5_UK, lab = 'UK February precipitation (mm/d)')
```

```
Warning message:
“Removed 4 row(s) containing missing values (geom_path).”
```



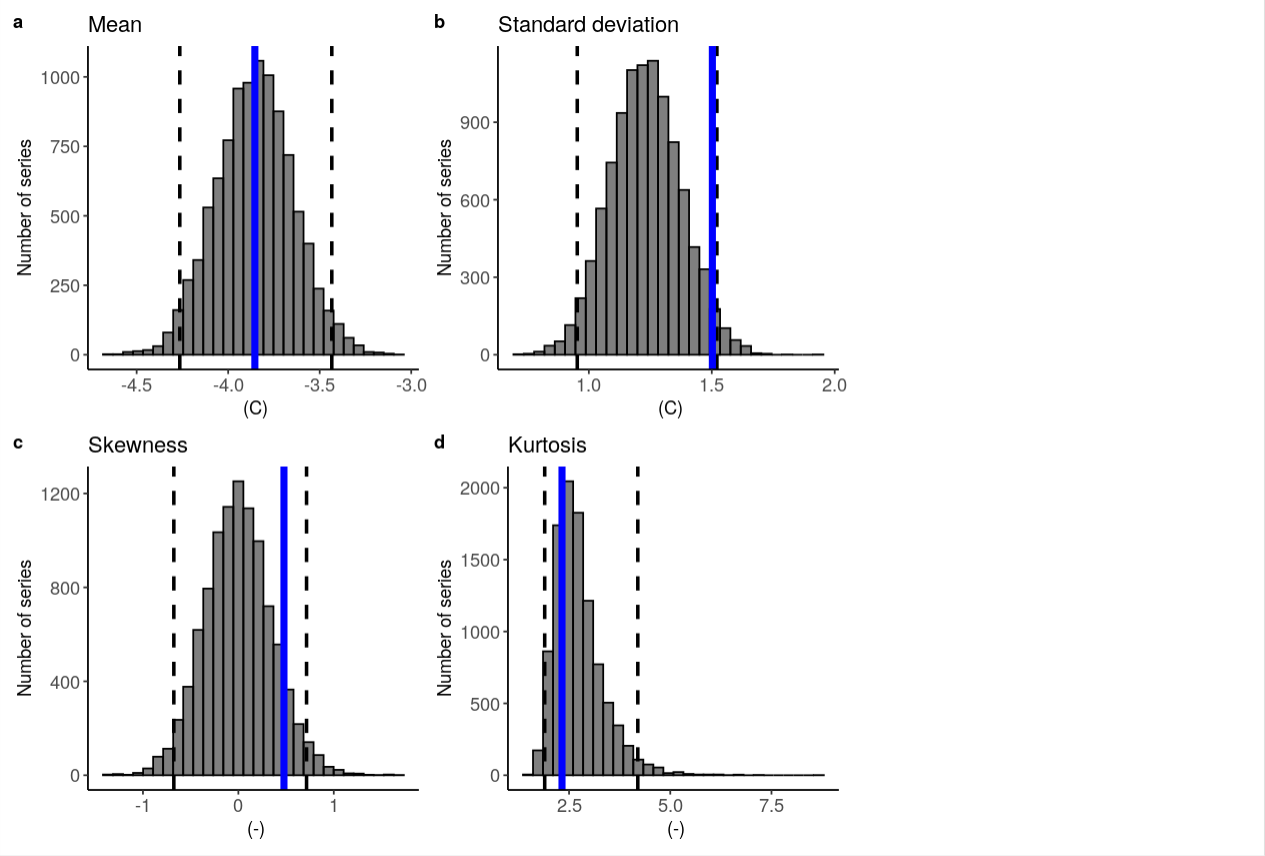
1.6.4 Fidelity

[22]:
```
fidelity_test(
    obs = ERA5_Siberia_events_zoomed_hindcast$t2m,
    ensemble = SEAS5_Siberia_events_zoomed_hindcast$t2m,
    units = 'C',
    biascor = FALSE
)
```



Lets apply a additive biascor

[23]:
```
#Lets apply a additive biascor
obs = ERA5_Siberia_events_zoomed_hindcast$t2m
ensemble = SEAS5_Siberia_events_zoomed_hindcast$t2m
ensemble_biascor = ensemble + (mean(obs) - mean(ensemble))

fidelity_test(
    obs = obs,
    ensemble = ensemble_biascor,
    units = 'C',
    biascor = FALSE
)
```



1.7 Global monthly temperature records in ERA5

Where have monthly average temperatures broken records across the world in 2020?

In this first section, we load required packages and modules

[29]:
```
##This is so variables get printed within jupyter
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

[2]:
```
##import packages
import os
import xarray as xr
import numpy as np
import matplotlib.pyplot as plt
import cartopy
import cartopy.crs as ccrs
import matplotlib.ticker as mticker

#for rank calculation
import bottleneck
```

[3]:
```
## this is to load our own function to retrieve ERA5,
## which is located in ../src/CDSretrieve.py
import sys
sys.path.append('../')
```

[4]:
```
##And here we load the module
import src.CDSretrieve as retrieve
```

[5]:
```
##We want the working directory to be the UNSEEN-open directory
pwd = os.getcwd() ##current working directory is UNSEEN-open/Notebooks/1.Download
pwd #print the present working directory
os.chdir(pwd+'/../') # Change the working directory to UNSEEN-open
os.getcwd() #print the working directory
```

[5]:
```
'/lustre/soge1/projects/ls/personal/timo/UNSEEN-open/Notebooks'
```

[5]:
```
'/lustre/soge1/projects/ls/personal/timo/UNSEEN-open'
```

1.7.1 Download ERA5

This section describes the retrieval of ERA5. We retrieve netcdf files of global monthly 2m temperature and 2m dewpoint temperature for each year over 1979-2020.

[39]:
```
retrieve.retrieve_ERA5(variables = ['2m_temperature','2m_dewpoint_temperature'],␣
↪folder = '../Siberia_example/')
;
```

[39]:
```
''
```

We load all files with xarray `open_mfdataset`. The latest 3 months in this dataset are made available through ERA5T, which might be slightly different to ERA5. In the downloaded file, an extra dimenions 'expver' indicates which data is ERA5 (`expver = 1`) and which is ERA5T (`expver = 5`). After retrieving and loading, I combine both ERA5 and ERA5T to create a dataset that runs until August 2020.

[10]:
```
ERA5 = xr.open_mfdataset('../Siberia_example/ERA5_????.nc',combine='by_coords') ##␣
↪open the data
ERA5#
```

[10]:
```
<xarray.Dataset>
Dimensions:    (expver: 2, latitude: 181, longitude: 360, time: 500)
Coordinates:
  * latitude   (latitude) float32 90.0 89.0 88.0 87.0 ... -88.0 -89.0 -90.0
  * longitude  (longitude) float32 -180.0 -179.0 -178.0 ... 177.0 178.0 179.0
  * expver     (expver) int32 1 5
  * time       (time) datetime64[ns] 1979-01-01 1979-02-01 ... 2020-08-01
Data variables:
    t2m        (time, latitude, longitude, expver) float32 dask.array<chunksize=(12,␣
↪181, 360, 2), meta=np.ndarray>
    d2m        (time, latitude, longitude, expver) float32 dask.array<chunksize=(12,␣
↪181, 360, 2), meta=np.ndarray>
Attributes:
    Conventions:  CF-1.6
    history:      2020-09-07 10:14:42 GMT by grib_to_netcdf-2.16.0: /opt/ecmw...
```

[14]:
```
ERA5_combine =ERA5.sel(expver=1).combine_first(ERA5.sel(expver=5))
ERA5_combine.load()
```

[14]:
```
<xarray.Dataset>
Dimensions:    (latitude: 181, longitude: 360, time: 500)
Coordinates:
  * latitude   (latitude) float32 90.0 89.0 88.0 87.0 ... -88.0 -89.0 -90.0
  * longitude  (longitude) float32 -180.0 -179.0 -178.0 ... 177.0 178.0 179.0
  * time       (time) datetime64[ns] 1979-01-01 1979-02-01 ... 2020-08-01
Data variables:
    t2m        (time, latitude, longitude) float32 244.7074 ... 214.79857
    d2m        (time, latitude, longitude) float32 241.76836 ... 211.0198
Attributes:
    Conventions:  CF-1.6
    history:      2020-09-07 10:14:42 GMT by grib_to_netcdf-2.16.0: /opt/ecmw...
```

1.7.2 Calculating the rank

We want to show for each month whether the recorded monthly average temperature for 2020 is the highest since 1979 (or second highest, etc.).

We first select only January months.

[15]:
```
ERA5_jan = ERA5_combine.sel(time=ERA5_combine['time.month'] == 1) ## Select only for␣
↪the i month
```

Then we calculate the rank of January average temperatures over the years. We rename the variable 't2m' into 'Temperature rank'.

[16]:
```
ERA5_jan_rank = ERA5_jan['t2m'].rank(dim = 'time')
ERA5_jan_rank = ERA5_jan_rank.rename('Temperature rank')
```

We now have calculated the rank in increasing order, i.e. the highest values has the highest rank. However, we want to show the highest rank being number 1, the second highest being number 2. Therefore, we invert the ranks and then we select the inverted rank of January 2020 average temperature within the January average temperatures of the other years. If January 2020 average temperature would be highest on record, the inverted rank will be 1. Second highest will be 2.

[17]:
```
ERA5_jan_rank_inverted = (len(ERA5_jan_rank.time) - ERA5_jan_rank + 1).sel(time='2020
↪')
ERA5_jan_rank_inverted
```

[17]:
```
<xarray.DataArray 'Temperature rank' (time: 1, latitude: 181, longitude: 360)>
array([[[25., 25., 25., ..., 25., 25., 25.],
        [23., 23., 23., ..., 23., 23., 23.],
        [25., 25., 25., ..., 25., 25., 25.],
        ...,
        [23., 23., 23., ..., 23., 23., 23.],
        [24., 24., 24., ..., 24., 24., 24.],
        [24., 24., 24., ..., 24., 24., 24.]]])
Coordinates:
  * latitude   (latitude) float32 90.0 89.0 88.0 87.0 ... -88.0 -89.0 -90.0
  * longitude  (longitude) float32 -180.0 -179.0 -178.0 ... 177.0 178.0 179.0
  * time       (time) datetime64[ns] 2020-01-01
```

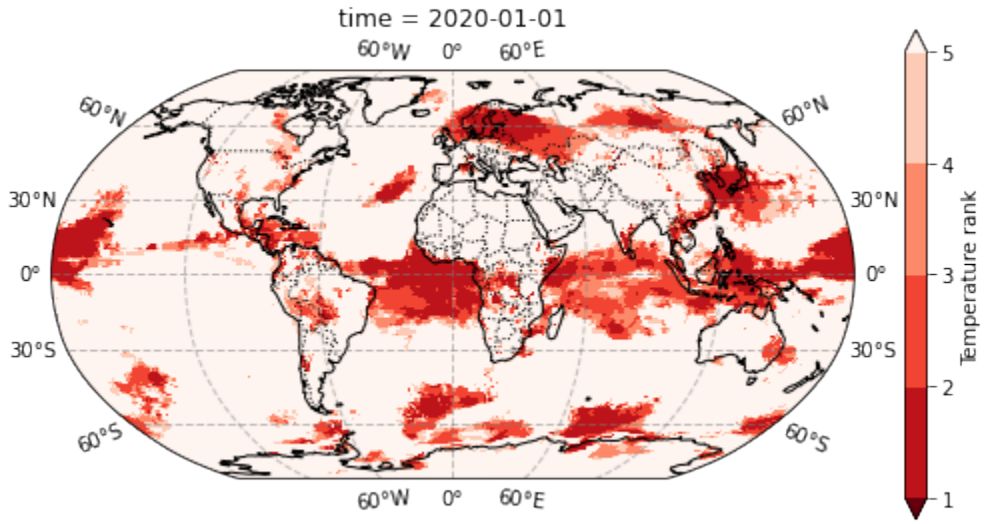
1.7.3 Plotting

We define a function to plot the data on a global map:

[18]:
```
def Global_plot(ERA5_i_rank_inverted):
    fig, ax = plt.subplots(figsize=(9, 4.5))
    ax = plt.axes(projection=ccrs.Robinson())
    ERA5_i_rank_inverted.plot(
        ax=ax,
        transform=ccrs.PlateCarree(),
        levels=[1, 2, 3, 4, 5],
        extend='both',
        colors=plt.cm.Reds_r)

    ax.add_feature(cartopy.feature.BORDERS, linestyle=':')
    ax.coastlines(
        resolution='110m')  #Currently can be one of "110m", "50m", and "10m".
    gl = ax.gridlines(crs=ccrs.PlateCarree(),
                      draw_labels=True,
                      linewidth=1,
                      color='gray',
                      alpha=0.5,
                      linestyle='--')
#     gl.top_labels = False
#     gl.right_labels = False
```

And plot!

[19]:
```
Global_plot(ERA5_jan_rank_inverted)
```



And zoom in for Siberia. We define a new plot:

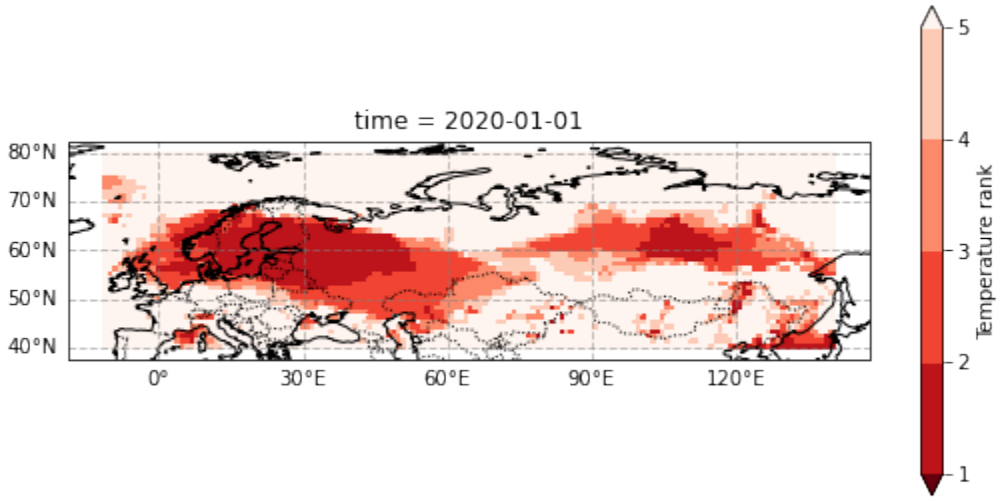
[55]:
```
def Siberia_plot(ERA5_i_rank_inverted):
    fig, ax = plt.subplots(figsize=(9, 4.5))
    ax = plt.axes(projection=ccrs.PlateCarree(central_longitude=50.0))
    ERA5_i_rank_inverted.plot(
        ax=ax,
        transform=ccrs.PlateCarree(),
        levels=[1, 2, 3, 4, 5],
```

(continues on next page)

(continued from previous page)

```
        extend='both',
        colors=plt.cm.Reds_r)

    ax.add_feature(cartopy.feature.BORDERS, linestyle=':')
    ax.coastlines(resolution='50m')
    gl = ax.gridlines(crs=ccrs.PlateCarree(),
                      draw_labels=True,
                      linewidth=1,
                      color='gray',
                      alpha=0.5,
                      linestyle='--')
    gl.top_labels = False
    gl.right_labels = False
```

[56]:
```
Siberia_plot(ERA5_jan_rank_inverted.sel(longitude = slice(-11,140), latitude =␣
↪slice(80,40)))
```



1.7.4 Loop over Jan-Aug

1.7.5 Create the gif

We use ImageMagick and run it from the command line. See this CMS notebook for more info on creating gifs.

[58]:
```
!convert -delay 60 ../Siberia_example/plots/Global*png graphs/Global_Animation_01.gif
```

[60]:
```
!convert -delay 60 ../Siberia_example/plots/Siberia*png graphs/Siberia_Animation_01.
↪gif
```

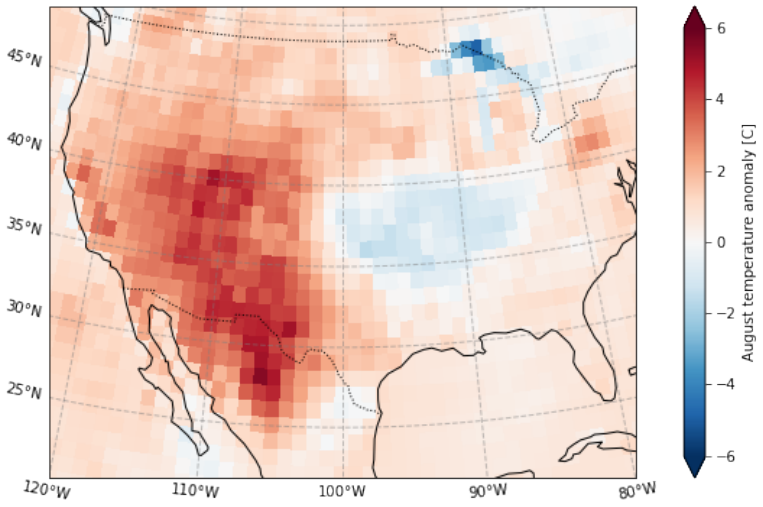
And show the gif in jupyter notebook with

```
![Global Temperature records 2020](../graphs/Global_Animation_01.gif
"Records2020")
```

Same for the Siberian temperature records: `![Siberian Temperature records 2020](../graphs/Siberia_Animation_01.gif "Records2020")`

1.8 California august temperature anomaly

How anomalous was the August 2020 average temperature?



In this first section, we load required packages and modules

[1]:
```
##This is so variables get printed within jupyter
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

[2]:
```
##import packages
import os
import xarray as xr
import numpy as np
import matplotlib.pyplot as plt
import cartopy
import cartopy.crs as ccrs
import matplotlib.ticker as mticker
import cdsapi


#for rank calculation
# import bottleneck
```

[3]:
```
os.chdir(os.path.abspath('../../'))
```

1.8.1 Load ERA5

We have retrieved netcdf files of global monthly 2m temperature and 2m dewpoint temperature for each year over 1979-2020.

We load all files with xarray `open_mfdataset`.

[4]:
```
ERA5 = xr.open_mfdataset('../California_example/ERA5/ERA5_????.nc',combine='by_coords
↪') ## open the data
ERA5#
```

[4]:
```
<xarray.Dataset>
Dimensions:    (latitude: 51, longitude: 61, time: 42)
Coordinates:
  * longitude  (longitude) float32 -130.0 -129.0 -128.0 ... -72.0 -71.0 -70.0
  * latitude   (latitude) float32 70.0 69.0 68.0 67.0 ... 23.0 22.0 21.0 20.0
  * time       (time) datetime64[ns] 1979-08-01 1980-08-01 ... 2020-08-01
Data variables:
    t2m        (time, latitude, longitude) float32 dask.array<chunksize=(1, 51, 61),␣
↪meta=np.ndarray>
    d2m        (time, latitude, longitude) float32 dask.array<chunksize=(1, 51, 61),␣
↪meta=np.ndarray>
Attributes:
    Conventions:  CF-1.6
    history:      2021-02-09 15:47:30 GMT by grib_to_netcdf-2.16.0: /opt/ecmw...
```

1.8.2 Retrieve the land-sea mask

We retrieve the land-sea mask for ERA5 from CDS.

[5]:
```
c = cdsapi.Client()

c.retrieve(
    'reanalysis-era5-single-levels-monthly-means',
    {
        'format': 'netcdf',
        'product_type': 'monthly_averaged_reanalysis',
        'variable': 'land_sea_mask',
        'grid': [1.0, 1.0],
        'year': '1979',
        'month': '01',
        'time': '00:00',
    },
    '../California_example/ERA_landsea_mask.nc')
```

```
2021-08-12 10:14:27,929 INFO Welcome to the CDS
2021-08-12 10:14:27,931 INFO Sending request to https://cds.climate.copernicus.eu/api/
↪v2/resources/reanalysis-era5-single-levels-monthly-means
2021-08-12 10:14:28,432 INFO Request is completed
2021-08-12 10:14:28,433 INFO Downloading https://download-0011.copernicus-climate.eu/
↪cache-compute-0011/cache/data5/adaptor.mars.internal-1628605177.1251855-440-12-
↪4553cc0b-4a6e-4405-85fa-b9d5396ac1f1.nc to ../California_example/ERA_landsea_mask.
↪nc (130.5K)
2021-08-12 10:14:28,662 INFO Download rate 574.8K/s
```

[5]:
```
Result(content_length=133596,content_type=application/x-netcdf,location=https://
↪download-0011.copernicus-climate.eu/cache-compute-0011/cache/data5/adaptor.mars.
↪internal-1628605177.1251855-440-12-4553cc0b-4a6e-4405-85fa-b9d5396ac1f1.nc)
```

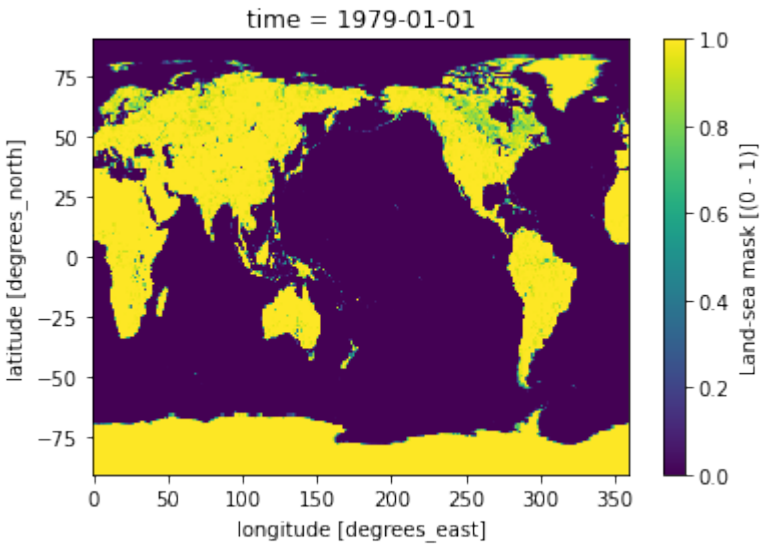
And here we open the dataset. It contains dimensionless values from 0 to 1. From CDS: "Grid boxes where this parameter has a value above 0.5 can be comprised of a mixture of land and inland water but not ocean. Grid boxes with a value of 0.5 and below can only be comprised of a water surface."

[5]:
```
LSMask = xr.open_dataset('../California_example/ERA_landsea_mask.nc')
LSMask.load()
```

[5]:
```
<xarray.Dataset>
Dimensions:    (latitude: 181, longitude: 360, time: 1)
Coordinates:
  * longitude  (longitude) float32 0.0 1.0 2.0 3.0 ... 356.0 357.0 358.0 359.0
  * latitude   (latitude) float32 90.0 89.0 88.0 87.0 ... -88.0 -89.0 -90.0
  * time       (time) datetime64[ns] 1979-01-01
Data variables:
    lsm        (time, latitude, longitude) float32 0.0 0.0 0.0 ... 1.0 1.0 1.0
Attributes:
    Conventions:  CF-1.6
    history:      2021-08-10 14:19:37 GMT by grib_to_netcdf-2.20.0: /opt/ecmw...
```

[6]:
```
LSMask['lsm'].plot()
```

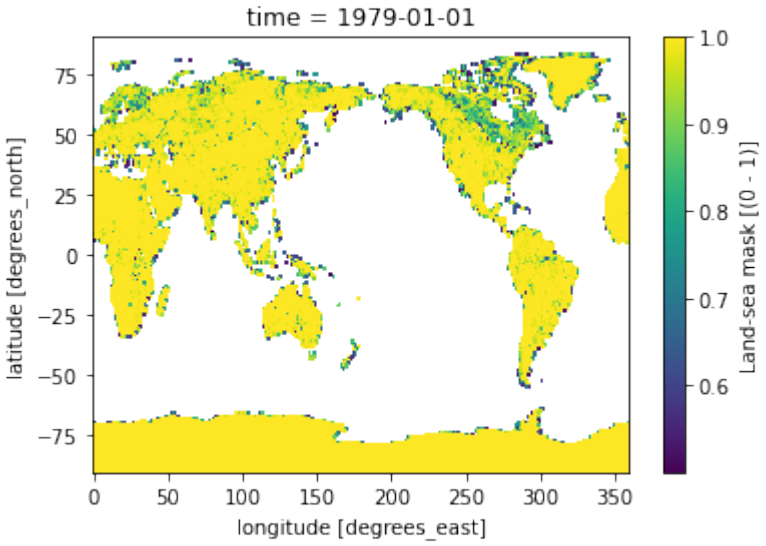
[6]:
```
<matplotlib.collections.QuadMesh at 0x7fbcb9f389a0>
```



We can select all gridcelss where the land-sea mask values are > 0.5 in order to remove ocean gridcells:

[7]:
```
LSMask['lsm'].where(LSMask['lsm'] > 0.5).plot()
```

[7]:
```
<matplotlib.collections.QuadMesh at 0x7fbcb9d95d00>
```



The longitude values in this dataset run from 0:360. We need to convert this to -180:180:

[8]:
```
# convert the longitude from 0:360 to -180:180
LSMask['longitude'] = (((LSMask['longitude'] + 180) % 360) - 180)
```

1.8.3 Calculating the anomaly

We want to show how anomalous the recorded monthly average temperature for 2020 is compared to the 1979-2010 average. We first calculate the temperature anomaly from the 1979-2010 mean and then calculate the standardardized anomaly by dividing the anomaly by the standard deviation:

[9]:
```
ERA5_anomaly = ERA5['t2m'] - ERA5['t2m'].sel(time=slice('1979','2010')).mean('time')
ERA5_anomaly.attrs = {
    'long_name': 'August temperature anomaly',
    'units': 'C'
}
ERA5_sd_anomaly = ERA5_anomaly / ERA5['t2m'].sel(time=slice('1979','2010')).std('time
↪')
ERA5_sd_anomaly.attrs = {
    'long_name': 'August temperature standardized anomaly',
    'units': '-'
}
```

1.8.4 Plotting the 2020 temperature anomaly

We define a function to plot the data on a global map:

[10]:
```
def plot_California(ERA5_input):

    extent = [-120, -80, 20, 50]
    central_lon = np.mean(extent[:2])
    central_lat = np.mean(extent[2:])

    plt.figure(figsize=(12, 6))
```

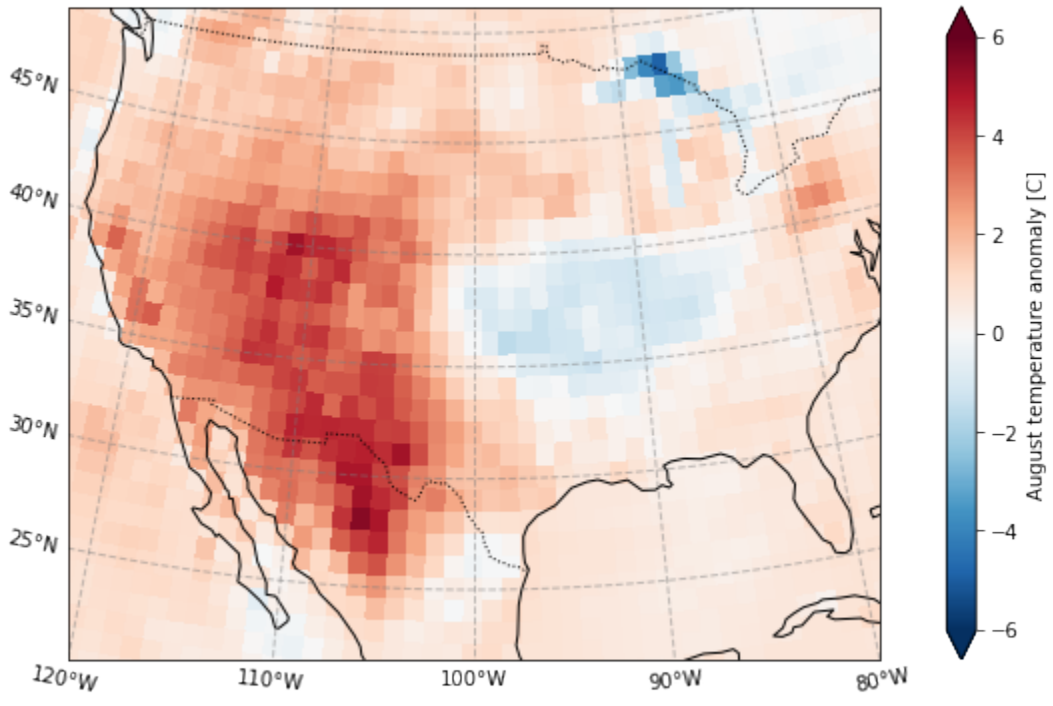
(continues on next page)

(continued from previous page)

```
    ax = plt.axes(projection=ccrs.AlbersEqualArea(central_lon, central_lat))
    ax.set_extent(extent)

    ERA5_input.plot(
        ax=ax,
        transform=ccrs.PlateCarree(),
        extend='both')

    ax.add_feature(cartopy.feature.BORDERS, linestyle=':')
    ax.coastlines(
        resolution='110m')  #Currently can be one of "110m", "50m", and "10m".
    ax.set_title('')
    gl = ax.gridlines(crs=ccrs.PlateCarree(),
                      draw_labels=True,
                      linewidth=1,
                      color='gray',
                      alpha=0.5,
                      linestyle='--')
    gl.top_labels = False
    gl.right_labels = False
```

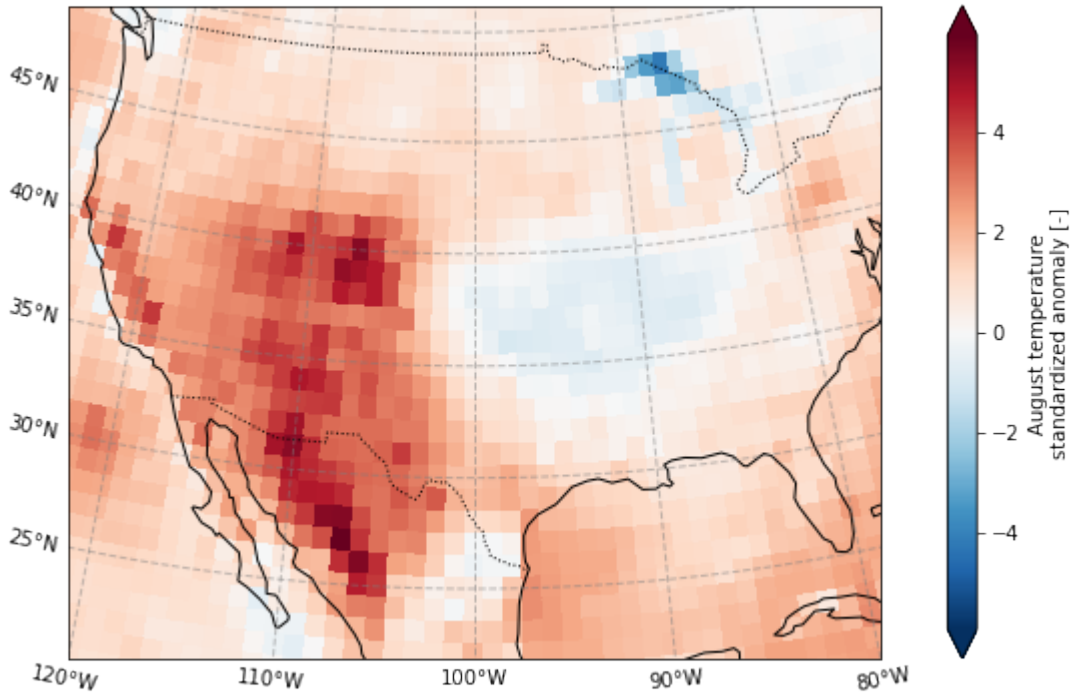
And plot the ERA5 2020 temperature anomaly

[11]:
```
plot_California(ERA5_anomaly.sel(time = '2020'))
# plt.savefig('graphs/California_anomaly.png')
```



Plot the standardized anomaly

[12]:
```
plot_California(ERA5_sd_anomaly.sel(time = '2020'))
```



1.8.5 Selecting a domain for further analysis

We define the domain as a contiguous, land-only region within the box 125:100W, 20:45N with temperature anomalies above 2 standard deviation.

[17]:
```
ERA5_sd_anomaly_masked = (ERA5_sd_anomaly.
                sel(longitude = slice(-125,-100), #select the domain
                   latitude = slice(45,20)).      #select the domain
                where(ERA5_sd_anomaly.sel(time = '2020').squeeze('time')>2).  #Select␣
↪the region where sd>2
                where(LSMask['lsm'].sel(time = '1979').squeeze('time') > 0.5) #Select␣
↪land-only gridcells
               )
ERA5_sd_anomaly_masked.load()
```

[17]:
```
<xarray.DataArray 't2m' (time: 42, latitude: 26, longitude: 26)>
array([[[nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        ...,
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan]],

       [[nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        ...,
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan]],
```

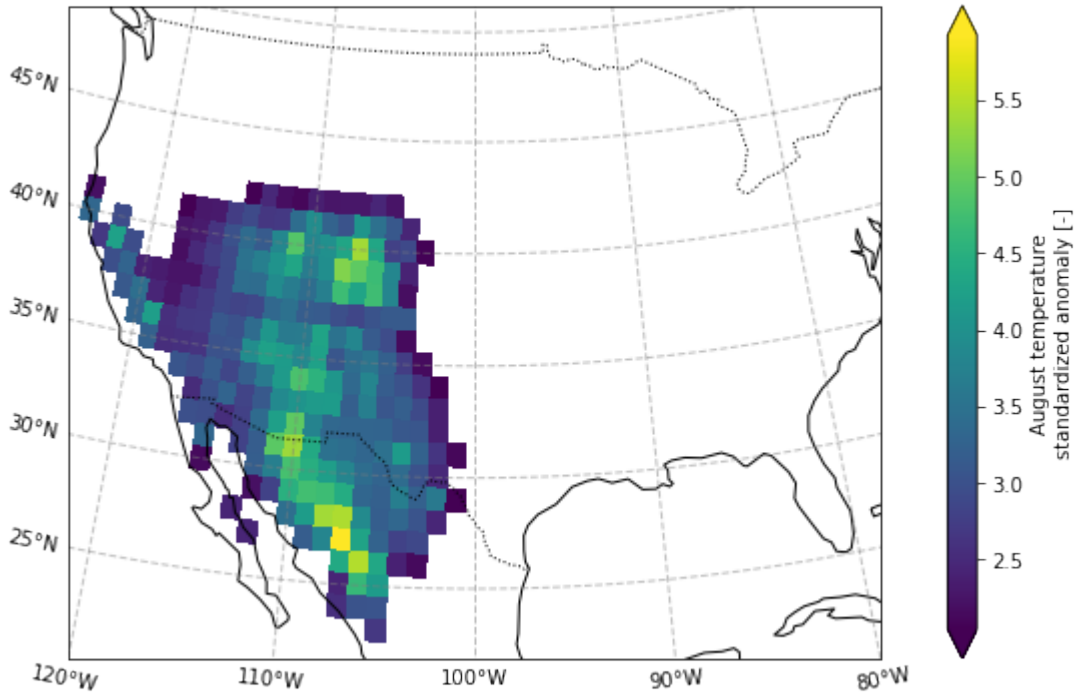
(continues on next page)

(continued from previous page)

```
       [[nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        ...,
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan]],

       ...,

       [[nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        ...,
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan]],

       [[nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        ...,
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan]],

       [[nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        ...,
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan]]], dtype=float32)
Coordinates:
  * latitude   (latitude) float64 45.0 44.0 43.0 42.0 ... 23.0 22.0 21.0 20.0
  * longitude  (longitude) float64 -125.0 -124.0 -123.0 ... -102.0 -101.0 -100.0
  * time       (time) datetime64[ns] 1979-08-01 1980-08-01 ... 2020-08-01
Attributes:
    long_name:  August temperature standardized anomaly
    units:      -
```

The resulting domain looks as follows:

[18]:
```
plot_California(ERA5_sd_anomaly_masked.sel(time = '2020'))
```



Let's make a nicer plot, using the previously defined function but adding an outline for the domain

[21]:
```
def plot_California(ERA5_input, ERA5_masked):

    extent = [-120, -80, 20, 50]
    central_lon = np.mean(extent[:2])
    central_lat = np.mean(extent[2:])

    plt.figure(frameon=False, figsize=(90 / 25.4, 60 / 25.4))
    ax = plt.axes(projection=ccrs.AlbersEqualArea(central_lon, central_lat))
    ax.set_extent(extent)

    ERA5_input.plot(
        ax=ax,
        transform=ccrs.PlateCarree(),
        extend='both')

    (ERA5_masked.fillna(0).sel(time = '2020').
     squeeze('time').
        plot.contour(levels = [0],
                     colors = 'black',
                      transform=ccrs.PlateCarree(),
                      ax = ax)
    )

    ax.add_feature(cartopy.feature.BORDERS, linestyle=':')
    ax.coastlines(
        resolution='110m')  #Currently can be one of "110m", "50m", and "10m".
    ax.set_title('')
    gl = ax.gridlines(crs=ccrs.PlateCarree(),
                      draw_labels=True,
                      linewidth=1,
```

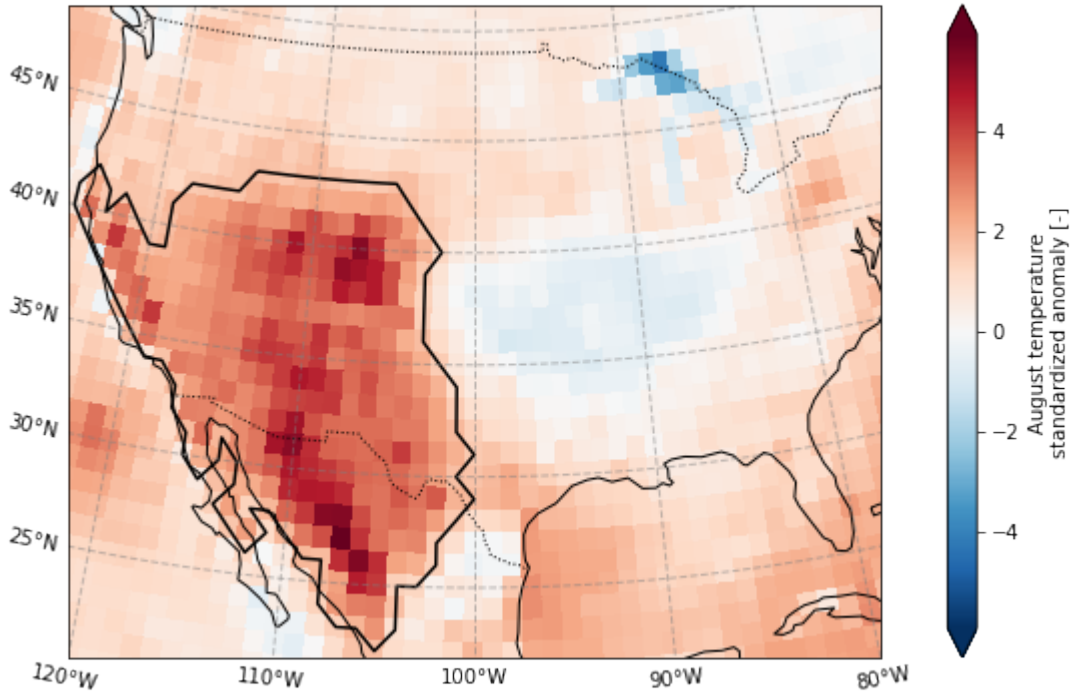
(continues on next page)

(continued from previous page)

```
                    color='gray',
                    alpha=0.5,
                    linestyle='--')
    gl.top_labels = False
    gl.right_labels = False
```

[20]:
```
plot_California(ERA5_sd_anomaly.sel(time = '2020'),ERA5_sd_anomaly_masked)
```

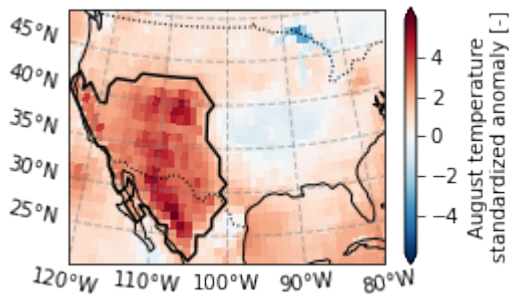
```
/soge-home/users/cenv0732/.conda/envs/UNSEEN-open/lib/python3.8/site-packages/cartopy/
↪mpl/geoaxes.py:1478: UserWarning: No contour levels were found within the data␣
↪range.
  result = matplotlib.axes.Axes.contour(self, *args, **kwargs)
```



Some functions that were used to create the figure for publication: - Set figure size as 90 by 60mm and remove frame: `plt.figure(frameon=False, figsize=(90 / 25.4, 60 / 25.4))` - Set the font: `plt.rcParams["font.family"] = "sans-serif"` - Set the font size: `plt.rcParams['font.size'] = 10` - Set the font type so editing software (e.g. inkscape) can recognize text for svg graphics: `plt.rcParams['svg.fonttype'] = 'none'` - Same but for pdf files: `plt.rcParams['pdf.fonttype'] = 42`

[22]:
```
plt.rcParams["font.family"] = "sans-serif" ##change font
plt.rcParams['font.size'] = 10  ## change font size
# plt.rcParams['svg.fonttype'] = 'none' ## so inkscape recognized texts in svg file
plt.rcParams['pdf.fonttype'] = 42 ## so illustrator can recognize text
plot_California(ERA5_sd_anomaly.sel(time = '2020'),ERA5_sd_anomaly_masked)
plt.savefig('graphs/California_sd_anomaly_contour.pdf')
```

```
/soge-home/users/cenv0732/.conda/envs/UNSEEN-open/lib/python3.8/site-packages/cartopy/
↪mpl/geoaxes.py:1478: UserWarning: No contour levels were found within the data␣
↪range.
  result = matplotlib.axes.Axes.contour(self, *args, **kwargs)
```



Timeseries for the selected domain

Here we take the areal average over the selected domain and plot the resulting timeseries. Gridcell need to be weighted with their cell area when taking the spatial mean. We first calculate the area weights and then use these to average.

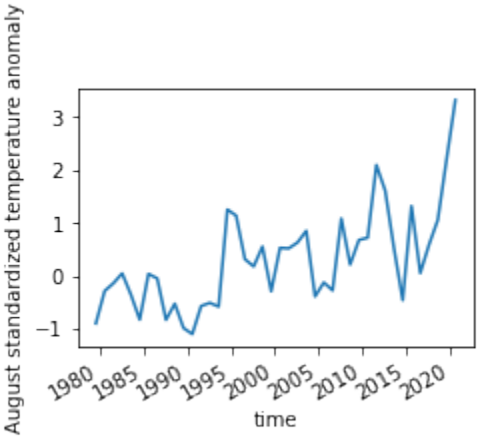
[23]:
```
area_weights = np.cos(np.deg2rad(ERA5_sd_anomaly.latitude))

ERA5_std_anomaly_timeseries = ERA5_sd_anomaly_masked.weighted(area_weights).mean([
↪'longitude','latitude'])
```

[28]:
```
plt.figure(frameon=False, figsize=(90 / 25.4, 60 / 25.4))
ERA5_std_anomaly_timeseries.plot()
plt.ylabel('August standardized temperature anomaly')
plt.savefig('graphs/California_anomaly_timeseries.pdf')
```

[28]:
```
<Figure size 255.118x170.079 with 0 Axes>
```

[28]:
```
[<matplotlib.lines.Line2D at 0x7fbcb0643f40>]
```

[28]:
```
Text(0, 0.5, 'August standardized temperature anomaly')
```



Another option would be to use the Californian domain using regionmask: `states = regionmask.defined_regions.natural_earth.us_states_50`

1.9 February and April 2020 precipitation anomalies

In this notebook, we will analyze precipitation anomalies of February and April 2020, which seemed to be very contrasting in weather. We use the EOBS dataset.

1.9.1 Import packages

[1]:
```
##This is so variables get printed within jupyter
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

[2]:
```
##import packages
import os
import xarray as xr
import numpy as np
import matplotlib.pyplot as plt
import cartopy
import cartopy.crs as ccrs
import matplotlib.ticker as mticker
```

[3]:
```
os.chdir(os.path.abspath('../../')) # Change the working directory to UNSEEN-open
os.getcwd() #print the working directory
```

[3]:
```
'/lustre/soge1/projects/ls/personal/timo/UNSEEN-open'
```

[4]:
```
### Set plot font size
plt.rcParams['font.size'] = 10  ## change font size
```

1.9.2 Load EOBS

I downloaded EOBS (from 1950 - 2019) and the most recent EOBS data (2020) here. Note, you have to register as E-OBS user.

The data has a daily timestep. I resample the data into monthly average mm/day. I chose not to use the total monthly precipitation because of leap days.

[5]:
```
EOBS = xr.open_dataset('../UK_example/EOBS/rr_ens_mean_0.25deg_reg_v20.0e.nc') ##␣
↪open the data
EOBS = EOBS.resample(time='1m').mean() ## Monthly averages
# EOBS = EOBS.sel(time=EOBS['time.month'] == 2) ## Select only February
EOBS
```

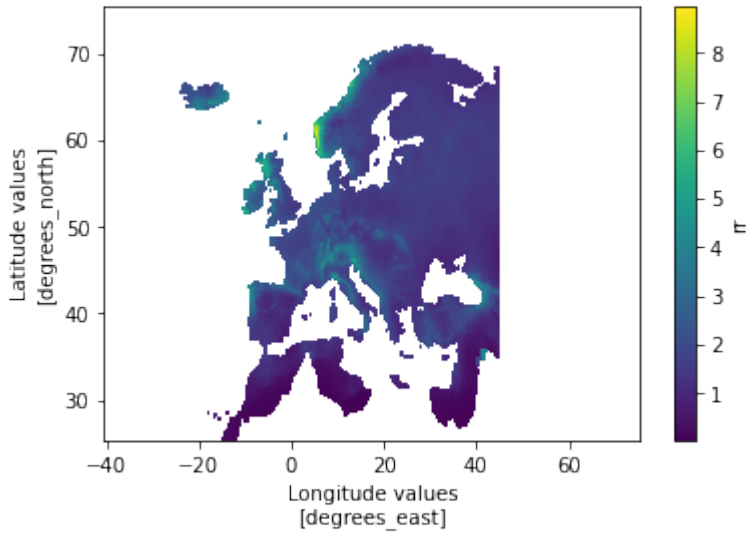
```
/soge-home/users/cenv0732/.conda/envs/UNSEEN-open/lib/python3.8/site-packages/xarray/
↪core/nanops.py:142: RuntimeWarning: Mean of empty slice
  return np.nanmean(a, axis=axis, dtype=dtype)
```

[5]:
```
<xarray.Dataset>
Dimensions:    (latitude: 201, longitude: 464, time: 835)
Coordinates:
  * time       (time) datetime64[ns] 1950-01-31 1950-02-28 ... 2019-07-31
  * longitude  (longitude) float64 -40.38 -40.12 -39.88 ... 74.88 75.12 75.38
  * latitude   (latitude) float64 25.38 25.62 25.88 26.12 ... 74.88 75.12 75.38
Data variables:
    rr         (time, latitude, longitude) float32 nan nan nan ... nan nan nan
```

Here I define the attributes, that xarray uses when plotting

[6]:
```
EOBS['rr'].attrs = {'long_name': 'rainfall',  ##Define the name
 'units': 'mm/day', ## unit
 'standard_name': 'thickness_of_rainfall_amount'} ## original name, not used
EOBS['rr'].mean('time').plot() ## and show the 1950-2019 average February␣
↪precipitation
```

[6]:
```
<matplotlib.collections.QuadMesh at 0x7f5f44952610>
```



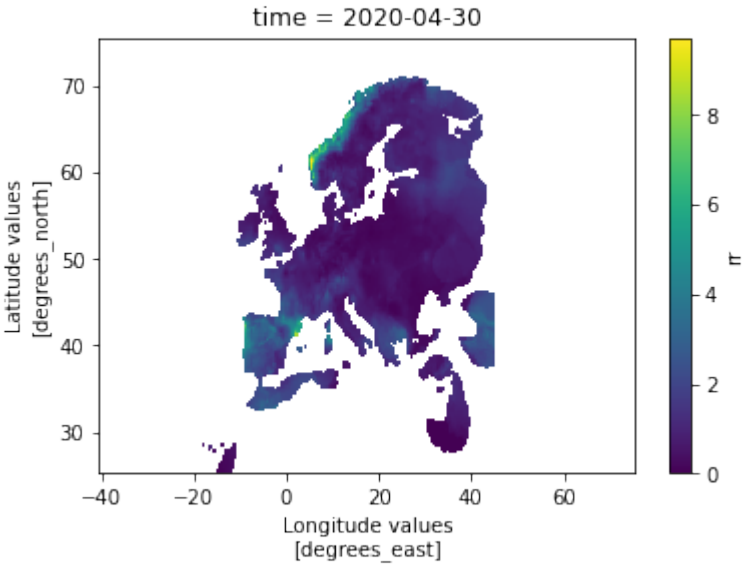
The 2020 data file is separate and needs the same preprocessing:

[7]:
```
EOBS2020 = xr.open_dataset('../UK_example/EOBS/rr_0.25deg_day_2020_grid_ensmean.nc.1
↪') #open
EOBS2020 = EOBS2020.resample(time='1m').mean() #Monthly mean
EOBS2020['rr'].sel(time='2020-04').plot() #show map
EOBS2020 ## display dataset
```

```
/soge-home/users/cenv0732/.conda/envs/UNSEEN-open/lib/python3.8/site-packages/xarray/
↪core/nanops.py:142: RuntimeWarning: Mean of empty slice
  return np.nanmean(a, axis=axis, dtype=dtype)
```

[7]:
```
<matplotlib.collections.QuadMesh at 0x7f5f448a0e50>
```

[7]:
```
<xarray.Dataset>
Dimensions:    (latitude: 201, longitude: 464, time: 12)
Coordinates:
  * time       (time) datetime64[ns] 2020-01-31 2020-02-29 ... 2020-12-31
  * longitude  (longitude) float64 -40.38 -40.12 -39.88 ... 74.88 75.12 75.38
  * latitude   (latitude) float64 25.38 25.62 25.88 26.12 ... 74.88 75.12 75.38
Data variables:
    rr         (time, latitude, longitude) float32 nan nan nan ... nan nan nan
```



1.9.3 Plot the 2020 event

I calculate the anomaly (deviation from the mean in mm/d) and divide this by the standard deviation to obtain the standardized anomalies.

[8]:
```
EOBS2020_anomaly = EOBS2020['rr'].groupby('time.month') - EOBS['rr'].groupby('time.
↪month').mean('time')
EOBS2020_anomaly

EOBS2020_sd_anomaly = EOBS2020_anomaly.groupby('time.month') / EOBS['rr'].groupby(
↪'time.month').std('time')

EOBS2020_sd_anomaly.attrs = {
    'long_name': 'Monthly precipitation standardized anomaly',
    'units': '-'
}

EOBS2020_sd_anomaly
```

[8]:
```
<xarray.DataArray 'rr' (time: 12, latitude: 201, longitude: 464)>
array([[[nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        ...,
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan]],

       [[nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        ...,
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan]],
```

(continues on next page)

(continued from previous page)

```
       [[nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        ...,
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan]],

       ...,

       [[nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        ...,
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan]],

       [[nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        ...,
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan]],

       [[nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        ...,
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan]]], dtype=float32)
Coordinates:
  * longitude  (longitude) float64 -40.38 -40.12 -39.88 ... 74.88 75.12 75.38
  * latitude   (latitude) float64 25.38 25.62 25.88 26.12 ... 74.88 75.12 75.38
  * time       (time) datetime64[ns] 2020-01-31 2020-02-29 ... 2020-12-31
    month      (time) int64 1 2 3 4 5 6 7 8 9 10 11 12
```

[8]:
```
<xarray.DataArray 'rr' (time: 12, latitude: 201, longitude: 464)>
array([[[nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        ...,
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan]],

       [[nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        ...,
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan]],
```

(continues on next page)

(continued from previous page)

```
       [[nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        ...,
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan]],

       ...,

       [[nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        ...,
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan]],

       [[nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        ...,
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan]],

       [[nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        ...,
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan]]], dtype=float32)
Coordinates:
  * longitude  (longitude) float64 -40.38 -40.12 -39.88 ... 74.88 75.12 75.38
  * latitude   (latitude) float64 25.38 25.62 25.88 26.12 ... 74.88 75.12 75.38
  * time       (time) datetime64[ns] 2020-01-31 2020-02-29 ... 2020-12-31
    month      (time) int64 1 2 3 4 5 6 7 8 9 10 11 12
Attributes:
    long_name:  Monthly precipitation standardized anomaly
    units:      -
```

I select February and April (tips on how to select this are appreciated)

[9]:
```
EOBS2020_sd_anomaly
# EOBS2020_sd_anomaly.sel(time = ['2020-02','2020-04']) ## Dont know how to select␣
↪this by label?
EOBS2020_sd_anomaly[[1,3],:,:] ## Dont know how to select this by label?
```

[9]:
```
<xarray.DataArray 'rr' (time: 12, latitude: 201, longitude: 464)>
array([[[nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        ...,
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
```

(continues on next page)

(continued from previous page)

```
        [nan, nan, nan, ..., nan, nan, nan]],

       [[nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        ...,
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan]],

       [[nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        ...,
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan]],

       ...,

       [[nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        ...,
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan]],

       [[nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        ...,
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan]],

       [[nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        ...,
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan]]], dtype=float32)
Coordinates:
  * longitude  (longitude) float64 -40.38 -40.12 -39.88 ... 74.88 75.12 75.38
  * latitude   (latitude) float64 25.38 25.62 25.88 26.12 ... 74.88 75.12 75.38
  * time       (time) datetime64[ns] 2020-01-31 2020-02-29 ... 2020-12-31
    month      (time) int64 1 2 3 4 5 6 7 8 9 10 11 12
Attributes:
    long_name:  Monthly precipitation standardized anomaly
    units:      -
```

[9]:

```
<xarray.DataArray 'rr' (time: 2, latitude: 201, longitude: 464)>
array([[[nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        ...,
```

(continues on next page)

(continued from previous page)

```
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan]],

       [[nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        ...,
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan]]], dtype=float32)
Coordinates:
  * longitude  (longitude) float64 -40.38 -40.12 -39.88 ... 74.88 75.12 75.38
  * latitude   (latitude) float64 25.38 25.62 25.88 26.12 ... 74.88 75.12 75.38
  * time       (time) datetime64[ns] 2020-02-29 2020-04-30
    month      (time) int64 2 4
Attributes:
    long_name:  Monthly precipitation standardized anomaly
    units:      -
```

And plot using cartopy!

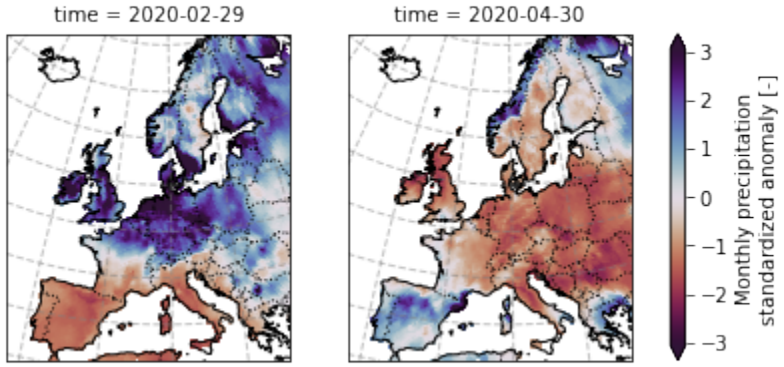
[11]:
```
EOBS_plots = EOBS2020_sd_anomaly[[1, 3], :, :].plot(
    transform=ccrs.PlateCarree(),
    robust=True,
    extend = 'both',
    col='time',
    cmap=plt.cm.twilight_shifted_r,
    subplot_kws={'projection': ccrs.EuroPP()})

for ax in EOBS_plots.axes.flat:
    ax.add_feature(cartopy.feature.BORDERS, linestyle=':')
    ax.coastlines(resolution='50m')
    gl = ax.gridlines(crs=ccrs.PlateCarree(),
                      draw_labels=False,
                      linewidth=1,
                      color='gray',
                      alpha=0.5,
                      linestyle='--')

# plt.savefig('graphs/February_April_2020_precipAnomaly.png', dpi=300)
```

[11]:
```
<cartopy.mpl.feature_artist.FeatureArtist at 0x7f5f4483cb20>
```

[11]:
```
<cartopy.mpl.feature_artist.FeatureArtist at 0x7f5f44847850>
```

[11]:
```
<cartopy.mpl.feature_artist.FeatureArtist at 0x7f5f449a6610>
```

[11]:
```
<cartopy.mpl.feature_artist.FeatureArtist at 0x7f5f403af2b0>
```



1.10 Using EOBS + upscaling

Here we explore how to best extract areal averaged precipitation and test this for UK precipitation within SEAS5 and EOBS. The code is inspired on Matteo De Felice's blog – credits to him!

We create a mask for all 241 countries within Regionmask, that has predefined countries from Natural Earth datasets (shapefiles). We use the mask to go from gridded precipitation to country-averaged timeseries. We regrid EOBS to the SEAS5 grid so we can select the same grid cells in calculating the UK average for both datasets. The country outline would not be perfect, but the masks would be the same so the comparison would be fair.

I use the xesmf package for upscaling, a good example can be found in this notebook.

1.10.1 Import packages

We need the packages regionmask for masking and xesmf for regridding. I cannot install xesmf into the UNSEEN-open environment without breaking my environment, so in this notebook I use a separate 'upscale' environment, as suggested by this issue. I use the packages esmpy=7.1.0 xesmf=0.2.1 regionmask cartopy matplotlib xarray numpy netcdf4.

[2]:
```
##This is so variables get printed within jupyter
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

[3]:
```
##import packages
import os
import sys
sys.path.insert(0, os.path.abspath('../../../'))
os.chdir(os.path.abspath('../../../'))
```

[4]:
```
import xarray as xr
import numpy as np
import matplotlib.pyplot as plt
import cartopy
import cartopy.crs as ccrs
import matplotlib.ticker as mticker

import regionmask        # Masking
import xesmf as xe       # Regridding
```

1.10.2 Load SEAS5 and EOBS

From CDS, we retrieve SEAS5 and here we merge the retrieved files (see more in preprocessing). We create a netcdf file containing the dimensions lat, lon, time (35 years), number (25 ensembles) and leadtime (5 initialization months).

[5]:
```
SEAS5 = xr.open_dataset('../UK_example/SEAS5/SEAS5_UK.nc')
```

And load EOBS netcdf with only February precipitation, resulting in 71 values, one for each year within 1950 - 2020 over the European domain (25N-75N x 40W-75E).

[6]:
```
EOBS = xr.open_dataset('../UK_example/EOBS/EOBS_UK.nc')
EOBS
```

[6]:
```
<xarray.Dataset>
Dimensions:    (latitude: 201, longitude: 464, time: 71)
Coordinates:
  * latitude   (latitude) float64 25.38 25.62 25.88 26.12 ... 74.88 75.12 75.38
  * longitude  (longitude) float64 -40.38 -40.12 -39.88 ... 74.88 75.12 75.38
  * time       (time) datetime64[ns] 1950-02-28 1951-02-28 ... 2020-02-29
Data variables:
    rr         (time, latitude, longitude) float32 ...
```

1.10.3 Masking

Here we load the countries and create a mask for SEAS5 and for EOBS.

Regionmask has predefined countries from Natural Earth datasets (shapefiles).

[7]:
```
countries = regionmask.defined_regions.natural_earth.countries_50
countries
```

[7]:
```
241 'Natural Earth Countries: 50m' Regions (http://www.naturalearthdata.com)
ZW ZM YE VN VE V VU UZ UY FSM MH MP VI GU AS PR US GS IO SH PN AI FK KY BM VG TC MS␣
↪JE GG IM GB AE UA UG TM TR TN TT TO TG TL TH TZ TJ TW SYR CH S SW SR SS SD LK E KR␣
↪ZA SO SL SB SK SLO SG SL SC RS SN SA ST RSM WS VC LC KN RW RUS RO QA P PL PH PE PY␣
↪PG PA PW PK OM N KP NG NE NI NZ NU CK NL AW CW NP NR NA MZ MA WS ME MN MD MC MX MU␣
↪MR M ML MV MY MW MG MK L LT FL LY LR LS LB LV LA KG KW KO KI KE KZ J J J I IS PAL␣
↪IRL IRQ IRN INDO IND IS HU HN HT GY GW GN GT GD GR GH D GE GM GA F PM WF MF BL PF␣
↪NC TF AI FIN FJ ET EST ER GQ SV EG EC DO DM DJ GL FO DK CZ CN CY CU HR CI CR DRC CG␣
↪KM CO CN MO HK CL TD CF CV CA CM KH MM BI BF BG BN BR BW BiH BO BT BJ BZ B BY BB BD␣
↪BH BS AZ A AU IOT HM NF AU ARM AR AG AO AND DZ AL AF SG AQ SX
```

Now we create the mask for the SEAS5 grid. Only one timestep is needed to create the mask. This mask will lateron be used to mask all the timesteps.

[8]:
```
SEAS5_mask = countries.mask(SEAS5.sel(leadtime=2, number=0, time='1982'),
                            lon_name='longitude',
                            lat_name='latitude')
```

And create a plot to illustrate what the mask looks like. The mask just indicates for each gridcell what country the gridcell belongs to.

[9]:
```
SEAS5_mask
SEAS5_mask.plot()
```

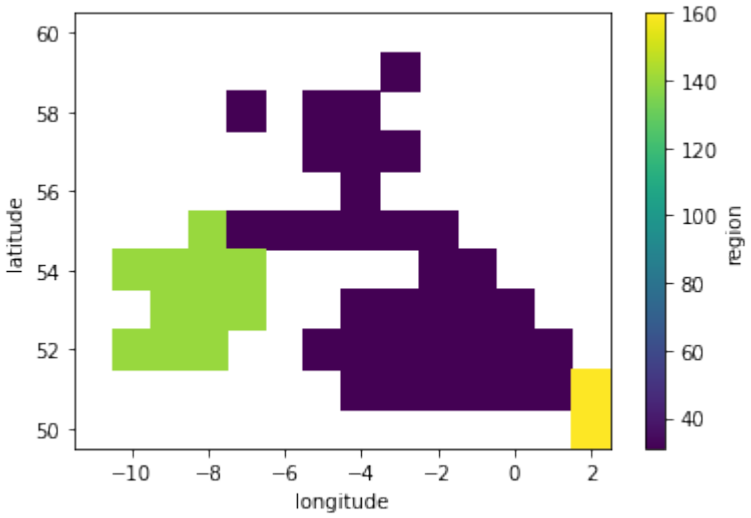
[9]:
```
<xarray.DataArray 'region' (latitude: 11, longitude: 14)>
array([[ nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,
```

(continues on next page)

(continued from previous page)

```
         nan,  nan,  nan],
       [ nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  31.,  nan,  nan,
         nan,  nan,  nan],
       [ nan,  nan,  nan,  nan,  31.,  nan,  31.,  31.,  nan,  nan,  nan,
         nan,  nan,  nan],
       [ nan,  nan,  nan,  nan,  nan,  nan,  31.,  31.,  31.,  nan,  nan,
         nan,  nan,  nan],
       [ nan,  nan,  nan,  nan,  nan,  nan,  nan,  31.,  nan,  nan,  nan,
         nan,  nan,  nan],
       [ nan,  nan,  nan, 140.,  31.,  31.,  31.,  31.,  31.,  31.,  nan,
         nan,  nan,  nan],
       [ nan, 140., 140., 140., 140.,  nan,  nan,  nan,  nan,  31.,  31.,
         nan,  nan,  nan],
       [ nan,  nan, 140., 140., 140.,  nan,  nan,  31.,  31.,  31.,  31.,
         31.,  nan,  nan],
       [ nan, 140., 140., 140.,  nan,  nan,  31.,  31.,  31.,  31.,  31.,
         31.,  31.,  nan],
       [ nan,  nan,  nan,  nan,  nan,  nan,  nan,  31.,  31.,  31.,  31.,
         31.,  31., 160.],
       [ nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,  nan,
         nan,  nan, 160.]])
Coordinates:
  * latitude   (latitude) float32 60.0 59.0 58.0 57.0 ... 53.0 52.0 51.0 50.0
  * longitude  (longitude) float32 -11.0 -10.0 -9.0 -8.0 ... -1.0 0.0 1.0 2.0
```

[9]:
```
<matplotlib.collections.QuadMesh at 0x2b58d909bd50>
```



And now we can extract the UK averaged precipitation within SEAS5 by using the mask index of the UK: `where(SEAS5_mask == UK_index)`. So we need to find the index of one of the 241 abbreviations. In this case for the UK use 'GB'. Additionally, if you can't find a country, use `countries.regions` to get the full names of the countries.

[10]:
```
countries.abbrevs.index('GB')
```

[10]:
```
31
```

To select the UK average, we select SEAS5 precipitation (tprate), select the gridcells that are within the UK and take the mean over those gridcells. This results in a dataset of February precipitation for 35 years (1981-2016), with 5 leadtimes and 25 ensemble members.

[11]:
```
SEAS5_UK = (SEAS5['tprate']
            .where(SEAS5_mask == 31)
            .mean(dim=['latitude', 'longitude']))
SEAS5_UK
```

```
/soge-home/users/cenv0732/.conda/envs/upscale/lib/python3.7/site-packages/xarray/core/
↪nanops.py:142: RuntimeWarning: Mean of empty slice
  return np.nanmean(a, axis=axis, dtype=dtype)
```

[11]:
```
<xarray.DataArray 'tprate' (leadtime: 5, time: 39, number: 51)>
array([[[1.7730116 , 1.9548205 , 3.7803986 , ...,        nan,
                nan,        nan],
        [3.040877  , 1.8855734 , 4.2009687 , ...,        nan,
                nan,        nan],
        [3.556001  , 3.6879914 , 3.184576  , ...,        nan,
                nan,        nan],
        ...,
        [2.9507504 , 2.789885  , 3.252184  , ..., 1.8537003 ,
         3.002799  , 3.8576229 ],
        [2.9951687 , 3.872034  , 3.8536534 , ..., 3.5534801 ,
         2.4795628 , 3.5001822 ],
        [1.6970206 , 1.3571059 , 2.7251225 , ..., 3.535101  ,
         3.3363144 , 3.8510854 ]],

       [[1.0868028 , 1.5332695 , 3.2461395 , ...,        nan,
                nan,        nan],
        [0.99898285, 2.9119303 , 2.1601522 , ...,        nan,
                nan,        nan],
        [3.6581304 , 3.5088263 , 2.0143754 , ...,        nan,
                nan,        nan],
        ...,
        [2.2972794 , 2.6159883 , 1.2061493 , ..., 2.6729386 ,
         2.8370278 , 3.2695346 ],
        [5.094759  , 2.7528167 , 1.9417399 , ..., 3.5889919 ,
         1.525842  , 3.0035791 ],
        [3.240109  , 4.146352  , 2.9840755 , ..., 3.131607  ,
         3.634821  , 2.8452704 ]],

       [[2.0119116 , 2.4435556 , 1.3927166 , ...,        nan,
                nan,        nan],
        [2.62523   , 3.6218376 , 3.003645  , ...,        nan,
                nan,        nan],
        [4.071685  , 2.6880858 , 3.8181992 , ...,        nan,
                nan,        nan],
        ...,
        [4.18686   , 2.4341922 , 2.3860729 , ..., 3.6107152 ,
         2.654895  , 1.8162413 ],
        [1.2847987 , 2.8927827 , 2.3829966 , ..., 4.846281  ,
         2.2673166 , 2.598539  ],
        [2.4176307 , 2.826758  , 1.9144063 , ..., 2.3856838 ,
         2.0960681 , 1.6105822 ]],

       [[2.9105136 , 3.6938024 , 1.1343408 , ...,        nan,
                nan,        nan],
        [4.02007   , 1.8249133 , 3.099     , ...,        nan,
                nan,        nan],
        [3.1248841 , 2.219241  , 3.6903172 , ...,        nan,
                nan,        nan],
```

(continues on next page)

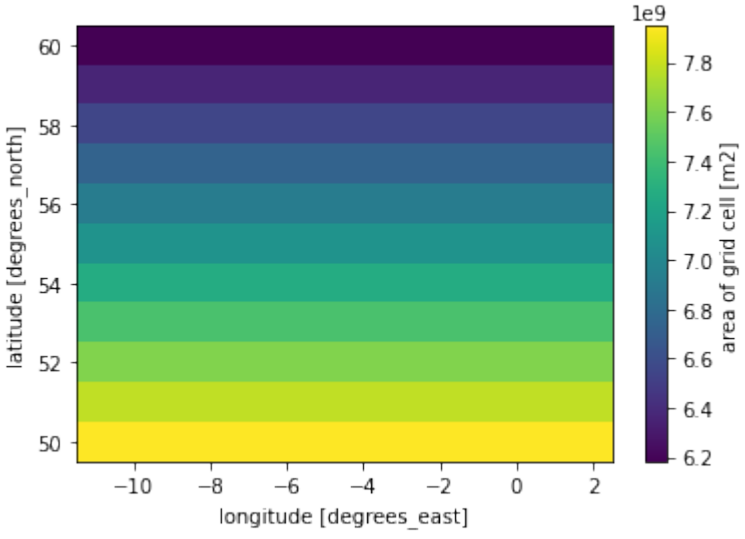
(continued from previous page)

```
        ...,
        [3.1467083 , 5.082951  , 2.9249673 , ..., 2.0092194 ,
         2.544652  , 3.8257258 ],
        [2.3694625 , 3.578296  , 3.527209  , ..., 3.950293  ,
         2.9967482 , 1.6948671 ],
        [4.3424473 , 5.037247  , 2.4635391 , ..., 2.5078914 ,
         2.767472  , 2.4778244 ]],

       [[3.1285267 , 3.269652  , 2.5995293 , ...,        nan,
                nan,        nan],
        [2.262867  , 3.3503478 , 2.4287066 , ...,        nan,
                nan,        nan],
        [4.0569496 , 2.156282  , 1.781804  , ...,        nan,
                nan,        nan],
        ...,
        [2.1076744 , 1.7262052 , 1.8901306 , ..., 3.2577527 ,
         3.3160198 , 1.7766333 ],
        [2.7879143 , 3.5520785 , 1.695757  , ..., 2.852018  ,
         3.3634171 , 2.9967682 ],
        [3.836647  , 2.7460904 , 4.5292573 , ..., 2.7118914 ,
         2.7603571 , 4.0256233 ]]], dtype=float32)
Coordinates:
  * number    (number) int64 0 1 2 3 4 5 6 7 8 9 ... 42 43 44 45 46 47 48 49 50
  * time      (time) datetime64[ns] 1982-02-01 1983-02-01 ... 2020-02-01
  * leadtime  (leadtime) int64 2 3 4 5 6
```

However, xarray does not take into account the area of the gridcells in taking the average. Therefore, we have to calculate the area-weighted mean of the gridcells. To calculate the area of each gridcell, I use cdo `cdo gridarea infile outfile`. Here I load the generated file:

[12]:
```
Gridarea_SEAS5 = xr.open_dataset('../UK_example/Gridarea_SEAS5.nc')
Gridarea_SEAS5['cell_area'].plot()
```

[12]:
```
<matplotlib.collections.QuadMesh at 0x2b58d91655d0>
```



[13]:
```
SEAS5_UK_weighted = (SEAS5['tprate']
                     .where(SEAS5_mask == 31)
```

(continues on next page)

(continued from previous page)

```
                  .weighted(Gridarea_SEAS5['cell_area'])
                  .mean(dim=['latitude', 'longitude'])
                 )
SEAS5_UK_weighted
```

[13]:
```
<xarray.DataArray (leadtime: 5, time: 39, number: 51)>
array([[[1.74715784, 1.91625164, 3.74246331, ...,        nan,
                nan,        nan],
        [3.01557164, 1.86355946, 4.23964218, ...,        nan,
                nan,        nan],
        [3.45037457, 3.67373672, 3.19124952, ...,        nan,
                nan,        nan],
        ...,
        [2.93410386, 2.74606084, 3.18639043, ..., 1.75603451,
         2.92185771, 3.83075713],
        [2.99583296, 3.88464775, 3.8476675 , ..., 3.51473106,
         2.43278432, 3.47741487],
        [1.70198039, 1.34639466, 2.70610296, ..., 3.45445812,
         3.2937839 , 3.80084943]],

       [[1.08925258, 1.502868  , 3.23383862, ...,        nan,
                nan,        nan],
        [0.96385251, 2.9144073 , 2.14176199, ...,        nan,
                nan,        nan],
        [3.64189637, 3.44186084, 1.96817031, ...,        nan,
                nan,        nan],
        ...,
        [2.26289577, 2.64050615, 1.18109141, ..., 2.64159823,
         2.78090027, 3.29229504],
        [5.05480192, 2.7228239 , 1.9085107 , ..., 3.56878426,
         1.46244825, 2.97974057],
        [3.19406732, 4.0754389 , 2.89935002, ..., 3.16283376,
         3.65486179, 2.7700864 ]],

       [[1.94362083, 2.4160058 , 1.36431312, ...,        nan,
                nan,        nan],
        [2.57294707, 3.55756557, 2.96458594, ...,        nan,
                nan,        nan],
        [4.13926899, 2.61380816, 3.76440713, ...,        nan,
                nan,        nan],
        ...,
        [4.12634415, 2.40580538, 2.30931212, ..., 3.60437091,
         2.65663573, 1.78742804],
        [1.26374643, 2.86376533, 2.36735188, ..., 4.80009495,
         2.22897541, 2.58805634],
        [2.37179549, 2.86106518, 1.90401998, ..., 2.40591114,
         2.08595829, 1.55529216]],

       [[2.83876303, 3.61651907, 1.0950032 , ...,        nan,
                nan,        nan],
        [3.95351432, 1.78778573, 3.08959013, ...,        nan,
                nan,        nan],
        [3.13152664, 2.19419128, 3.64975772, ...,        nan,
                nan,        nan],
        ...,
        [3.06984433, 4.99797376, 2.88955225, ..., 1.97087261,
         2.52861605, 3.75363435],
```

(continues on next page)

(continued from previous page)

```
        [2.36128612, 3.52506141, 3.50087731, ..., 3.93962213,
         2.94645673, 1.69376439],
        [4.35700042, 5.02027928, 2.46636484, ..., 2.46297193,
         2.74433285, 2.45057078]],

       [[3.15063505, 3.23490175, 2.60923731, ...,        nan,
                nan,        nan],
        [2.21017692, 3.32458317, 2.3819878 , ...,        nan,
                nan,        nan],
        [4.07655988, 2.07606666, 1.75961194, ...,        nan,
                nan,        nan],
        ...,
        [2.11429562, 1.68735339, 1.84325489, ..., 3.26149299,
         3.32371077, 1.78717855],
        [2.75385495, 3.4979577 , 1.66987709, ..., 2.85354534,
         3.40446786, 3.01953669],
        [3.81460036, 2.70650167, 4.54162104, ..., 2.69608025,
         2.73558576, 4.04264194]]])
Coordinates:
  * number    (number) int64 0 1 2 3 4 5 6 7 8 9 ... 42 43 44 45 46 47 48 49 50
  * time      (time) datetime64[ns] 1982-02-01 1983-02-01 ... 2020-02-01
  * leadtime  (leadtime) int64 2 3 4 5 6
```

Another solution is to take the cosine of the latitude, which is proportional to the grid cell area for regular latitude/longitude grids (xarray example). This should be the same as the previous example, but easier to reproduce.

[14]:
```
area_weights = np.cos(np.deg2rad(SEAS5.latitude))
SEAS5_UK_weighted_latcos = (SEAS5['tprate']
                  .where(SEAS5_mask == 31)
                  .weighted(area_weights)
                  .mean(dim=['latitude', 'longitude'])
                 )
```

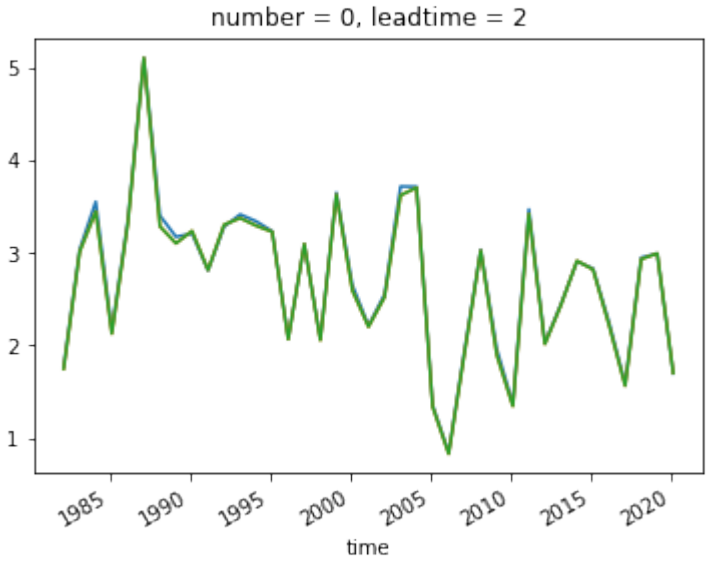
I plot the UK average for ensemble member 0 and leadtime 2 to show that the the two methods for taking an average are the same. Furthermore, the difference between the weighted and non-weighted average is very small in this case. The difference would be greater for larger domains and further towards to poles.

[15]:
```
SEAS5_UK.sel(leadtime=2,number=0).plot()
SEAS5_UK_weighted.sel(leadtime=2,number=0).plot()
SEAS5_UK_weighted_latcos.sel(leadtime=2,number=0).plot()
```

[15]:
```
[<matplotlib.lines.Line2D at 0x2b58d902ce90>]
```

[15]:
```
[<matplotlib.lines.Line2D at 0x2b58d9037c10>]
```

[15]:
```
[<matplotlib.lines.Line2D at 0x2b58d9014c10>]
```



1.10.4 Upscale

For EOBS, we want to upscale the dataset to the SEAS5 grid. We use the function `regridder(ds_in,ds_out, function)`, see the docs. We have to rename the lat lon dimensions so the function can read them.

We use bilinear interpolation first (i.e. function = 'bilinear'), because of its ease in implementation. However, the use of conservative areal average (function = 'conservative') for upscaling is preferred (Kopparla, 2013).

[16]:
```
regridder = xe.Regridder(EOBS.rename({'longitude': 'lon', 'latitude': 'lat'}), SEAS5.
↪rename({'longitude': 'lon', 'latitude': 'lat'}),'bilinear')
```

```
Overwrite existing file: bilinear_201x464_11x14.nc
 You can set reuse_weights=True to save computing time.
```

Now that we have the regridder, we can apply the regridder to our EOBS dataarray:

[17]:
```
EOBS_upscaled = regridder(EOBS)
EOBS_upscaled
```

```
using dimensions ('latitude', 'longitude') from data variable rr as the horizontal␣
↪dimensions for this dataset.
```

[17]:
```
<xarray.Dataset>
Dimensions:  (lat: 11, lon: 14, time: 71)
Coordinates:
  * time     (time) datetime64[ns] 1950-02-28 1951-02-28 ... 2020-02-29
  * lon      (lon) float32 -11.0 -10.0 -9.0 -8.0 -7.0 ... -2.0 -1.0 0.0 1.0 2.0
  * lat      (lat) float32 60.0 59.0 58.0 57.0 56.0 ... 54.0 53.0 52.0 51.0 50.0
Data variables:
    rr       (time, lat, lon) float64 nan nan nan nan nan ... nan nan nan 4.243
Attributes:
    regrid_method:  bilinear
```

And set the latlon dimension names back to their long name. This is so both SEAS5 and EOBS have the same latlon dimension names which is necessary when using the same mask.

[18]:
```
EOBS_upscaled = EOBS_upscaled.rename({'lon' : 'longitude', 'lat' : 'latitude'})
```

1.10.5 Illustrate the SEAS5 and EOBS masks for the UK

Here I plot the masked mean SEAS5 and upscaled EOBS precipitation. This shows that upscaled EOBS does not contain data for all gridcells within the UK mask (the difference between SEAS5 gridcells and EOBS gridcells with data). We can apply an additional mask for SEAS5 that masks the grid cells that do not contain data in EOBS.

[19]:
```
fig, axs = plt.subplots(1, 2, subplot_kw={'projection': ccrs.OSGB()})

SEAS5['tprate'].where(SEAS5_mask == 31).mean(
    dim=['time', 'leadtime', 'number']).plot(
    transform=ccrs.PlateCarree(),
    vmin=0,
    vmax=8,
    cmap=plt.cm.Blues,
    ax=axs[0])

EOBS_upscaled['rr'].where(SEAS5_mask == 31).mean(dim='time').plot(
    transform=ccrs.PlateCarree(),
    vmin=0,
    vmax=8,
    cmap=plt.cm.Blues,
    ax=axs[1])

for ax in axs.flat:
    ax.coastlines(resolution='10m')

axs[0].set_title('SEAS5')
axs[1].set_title('EOBS')
```

```
/soge-home/users/cenv0732/.conda/envs/upscale/lib/python3.7/site-packages/xarray/core/
↪nanops.py:142: RuntimeWarning: Mean of empty slice
  return np.nanmean(a, axis=axis, dtype=dtype)
```

[19]:
```
<matplotlib.collections.QuadMesh at 0x2b58d91fa710>
```

```
/soge-home/users/cenv0732/.conda/envs/upscale/lib/python3.7/site-packages/xarray/core/
↪nanops.py:142: RuntimeWarning: Mean of empty slice
  return np.nanmean(a, axis=axis, dtype=dtype)
```

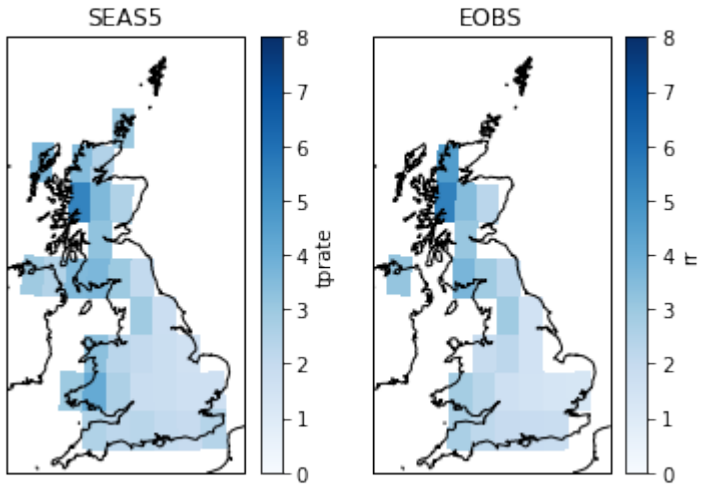
[19]:
```
<matplotlib.collections.QuadMesh at 0x2b58d9204f50>
```

[19]:
```
<cartopy.mpl.feature_artist.FeatureArtist at 0x2b58d9214fd0>
```

[19]:
```
<cartopy.mpl.feature_artist.FeatureArtist at 0x2b58d920a050>
```

[19]:
```
Text(0.5, 1.0, 'SEAS5')
```

[19]:
```
Text(0.5, 1.0, 'EOBS')
```



The additional mask of SEAS5 is where EOBS is not null:

[20]:
```
fig, axs = plt.subplots(1, 2, subplot_kw={'projection': ccrs.OSGB()})

(SEAS5['tprate']
 .where(SEAS5_mask == 31)
 .where(EOBS_upscaled['rr'].sel(time='1950').squeeze('time').notnull()) ## mask␣
↪values that are nan in EOBS
 .mean(dim=['time', 'leadtime', 'number'])
 .plot(
    transform=ccrs.PlateCarree(),
    vmin=0,
    vmax=8,
    cmap=plt.cm.Blues,
    ax=axs[0])
)

EOBS_upscaled['rr'].where(SEAS5_mask == 31).mean(dim='time').plot(
    transform=ccrs.PlateCarree(),
    vmin=0,
    vmax=8,
    cmap=plt.cm.Blues,
    ax=axs[1])

for ax in axs.flat:
    ax.coastlines(resolution='10m')

axs[0].set_title('SEAS5')
axs[1].set_title('EOBS')
```

```
/soge-home/users/cenv0732/.conda/envs/upscale/lib/python3.7/site-packages/xarray/core/
↪nanops.py:142: RuntimeWarning: Mean of empty slice
  return np.nanmean(a, axis=axis, dtype=dtype)
```

[20]:
```
<matplotlib.collections.QuadMesh at 0x2b58d9381a10>
```

```
/soge-home/users/cenv0732/.conda/envs/upscale/lib/python3.7/site-packages/xarray/core/
↪nanops.py:142: RuntimeWarning: Mean of empty slice
  return np.nanmean(a, axis=axis, dtype=dtype)
```

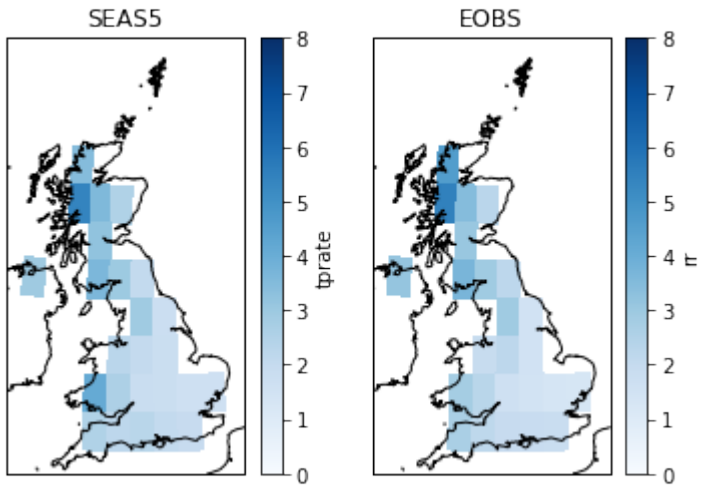
[20]:
```
<matplotlib.collections.QuadMesh at 0x2b58d9229f90>
```

[20]:
```
<cartopy.mpl.feature_artist.FeatureArtist at 0x2b58d9242390>
```

[20]:
```
<cartopy.mpl.feature_artist.FeatureArtist at 0x2b58d9242fd0>
```

[20]:
```
Text(0.5, 1.0, 'SEAS5')
```

[20]:
```
Text(0.5, 1.0, 'EOBS')
```



Let's include the 2020 event

[21]:
```
EOBS2020_sd_anomaly = EOBS_upscaled['rr'].sel(time='2020') - EOBS_upscaled['rr'].mean(
↪'time') / EOBS_upscaled['rr'].std('time')
EOBS2020_sd_anomaly.attrs = {
    'long_name': 'Precipitation anomaly',
    'units': '-'
}
EOBS2020_sd_anomaly
```

```
/soge-home/users/cenv0732/.conda/envs/upscale/lib/python3.7/site-packages/numpy/lib/
↪nanfunctions.py:1667: RuntimeWarning: Degrees of freedom <= 0 for slice.
  keepdims=keepdims)
```

[21]:
```
<xarray.DataArray 'rr' (time: 1, latitude: 11, longitude: 14)>
array([[[       nan,        nan,        nan,        nan,        nan,
                nan,        nan,        nan,        nan,        nan,
                nan,        nan,        nan,        nan],
        [       nan,        nan,        nan,        nan,        nan,
                nan,        nan,        nan,        nan,        nan,
                nan,        nan,        nan,        nan],
        [       nan,        nan,        nan,        nan,        nan,
                nan, 3.95092072,        nan,        nan,        nan,
                nan,        nan,        nan,        nan],
        [       nan,        nan,        nan,        nan,        nan,
                nan, 4.59428102, 3.79702769, 1.96181425,        nan,
                nan,        nan,        nan,        nan],
        [       nan,        nan,        nan,        nan,        nan,
                nan, 4.71781348, 5.2826348 ,        nan,        nan,
                nan,        nan,        nan,        nan],
        [       nan,        nan,        nan,        nan, 4.40361314,
                nan,        nan, 5.47361673, 6.10292013, 3.78611734,
```

(continues on next page)

(continued from previous page)

```
                nan,         nan,         nan,         nan],
        [       nan,         nan, 6.8501372 , 6.14007146, 4.30546062,
                nan,         nan,         nan,         nan, 3.68873396,
         2.38060983,         nan,         nan,         nan],
        [       nan,         nan,         nan, 5.73786063, 4.37045386,
                nan,         nan,         nan, 2.70572296, 3.11985962,
         3.02861986,         nan,         nan,         nan],
        [       nan, 5.81172535, 5.30645502,         nan,         nan,
                nan,         nan, 3.13110123, 3.03169002, 2.42073797,
         2.30607803, 1.0535416 , 0.16921805,         nan],
        [       nan,         nan,         nan,         nan,         nan,
                nan,         nan, 3.97998605, 3.35993006, 2.88487988,
         3.06793855, 1.90991066,         nan,         nan],
        [       nan,         nan,         nan,         nan,         nan,
                nan,         nan,         nan,         nan,         nan,
                nan,         nan,         nan, 2.58264812]]])
Coordinates:
  * time       (time) datetime64[ns] 2020-02-29
  * longitude  (longitude) float32 -11.0 -10.0 -9.0 -8.0 ... -1.0 0.0 1.0 2.0
  * latitude   (latitude) float32 60.0 59.0 58.0 57.0 ... 53.0 52.0 51.0 50.0
Attributes:
    long_name:  Precipitation anomaly
    units:      -
```

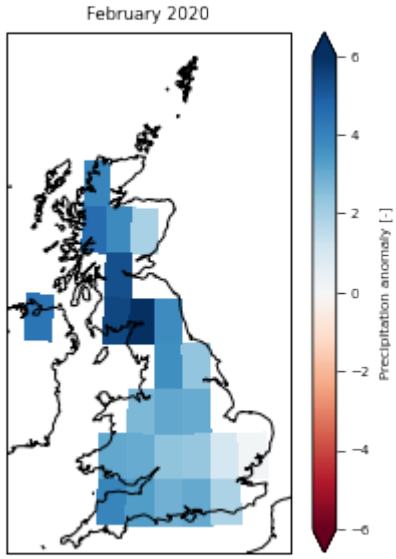
[50]:
```
plt.figure(figsize=(3.3, 4))
plt.rc('font', size=7) #controls default text size

ax = plt.axes(projection=ccrs.OSGB())

EOBS2020_sd_anomaly.where(SEAS5_mask == 31).plot(
    transform=ccrs.PlateCarree(),
    vmin = -6,
    vmax = 6,
    extend = 'both',
    cmap = plt.cm.RdBu,#twilight_shifted_r,#plt.cm.Blues,#
    ax=ax)

ax.coastlines(resolution='10m')
# gl = ax.gridlines(crs=ccrs.PlateCarree(),
#                   draw_labels=False,     # cannot label OSGB projection..
#                   linewidth=1,
#                   color='gray',
#                   alpha=0.5,
#                   linestyle='--')

ax.set_title('February 2020')
plt.tight_layout()
plt.savefig('graphs/UK_event_selection2.png', dpi=300)
```

[50]:
```
<Figure size 237.6x288 with 0 Axes>
```

[50]:
```
<matplotlib.collections.QuadMesh at 0x2b58f1ae4ed0>
```

[50]:
```
<cartopy.mpl.feature_artist.FeatureArtist at 0x2b58f1afbd50>
```

[50]:
```
Text(0.5, 1.0, 'February 2020')
```



[101]:
```
EOBS2020_sd_anomaly = EOBS['rr'].sel(time='2020') - EOBS['rr'].mean('time') / EOBS['rr
↪'].std('time')
EOBS2020_sd_anomaly.attrs = {
    'long_name': 'Precipitation anomaly',
    'units': '-'
}
EOBS2020_sd_anomaly


fig, axs = plt.subplots(1, 3, figsize=(6.7,2.5),subplot_kw={'projection': ccrs.OSGB()}
↪) #figsize=(10.,6.),

EOBS2020_sd_anomaly.plot(
    transform=ccrs.PlateCarree(),
    robust=True,
    extend = 'both',
    cmap=plt.cm.twilight_shifted_r,
    ax=axs[0])

(SEAS5['tprate']
 .where(SEAS5_mask == 31)
 .where(EOBS_upscaled['rr'].sel(time='1950').squeeze('time').notnull()) ## mask␣
↪values that are nan in EOBS
 .mean(dim=['time', 'leadtime', 'number'])
 .plot(
    transform=ccrs.PlateCarree(),
    vmin=0,
    vmax=8,
    cmap=plt.cm.Blues,
    ax=axs[1],
    cbar_kwargs={'label': 'February precipitation [mm/d]'}
 )
)

EOBS_upscaled['rr'].where(SEAS5_mask == 31).mean(dim='time').plot(
    transform=ccrs.PlateCarree(),
```

(continues on next page)

(continued from previous page)

```
    vmin=0,
    vmax=8,
    cmap=plt.cm.Blues,
    ax=axs[2],
    cbar_kwargs={'label': 'February precipitation [mm/d]'}
)


for ax in axs.flat:
    ax.coastlines(resolution='10m')
#     ax.set_aspect('auto')


axs[0].set_title('February 2020')
axs[1].set_title('SEAS5 average')
axs[2].set_title('EOBS average')

# fig.set_figwidth(180/26)
# plt.savefig('graphs/UK_event_selection.pdf', dpi=300)
```

```
/soge-home/users/cenv0732/.conda/envs/upscale/lib/python3.7/site-packages/numpy/lib/
↪nanfunctions.py:1667: RuntimeWarning: Degrees of freedom <= 0 for slice.
  keepdims=keepdims)
```

[101]:
```
<xarray.DataArray 'rr' (time: 1, latitude: 201, longitude: 464)>
array([[[nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        ...,
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan]]], dtype=float32)
Coordinates:
  * latitude   (latitude) float64 25.38 25.62 25.88 26.12 ... 74.88 75.12 75.38
  * longitude  (longitude) float64 -40.38 -40.12 -39.88 ... 74.88 75.12 75.38
  * time       (time) datetime64[ns] 2020-02-29
Attributes:
    long_name:  Precipitation anomaly
    units:      -
```

[101]:
```
<matplotlib.collections.QuadMesh at 0x7f16beef9790>
```

```
/soge-home/users/cenv0732/.conda/envs/upscale/lib/python3.7/site-packages/xarray/core/
↪nanops.py:142: RuntimeWarning: Mean of empty slice
  return np.nanmean(a, axis=axis, dtype=dtype)
```

[101]:
```
<matplotlib.collections.QuadMesh at 0x7f16beeb7e10>
```

```
/soge-home/users/cenv0732/.conda/envs/upscale/lib/python3.7/site-packages/xarray/core/
↪nanops.py:142: RuntimeWarning: Mean of empty slice
  return np.nanmean(a, axis=axis, dtype=dtype)
```

[101]:
```
<matplotlib.collections.QuadMesh at 0x7f16bee05ed0>
```

[101]:
```
<cartopy.mpl.feature_artist.FeatureArtist at 0x7f16bedd64d0>
```

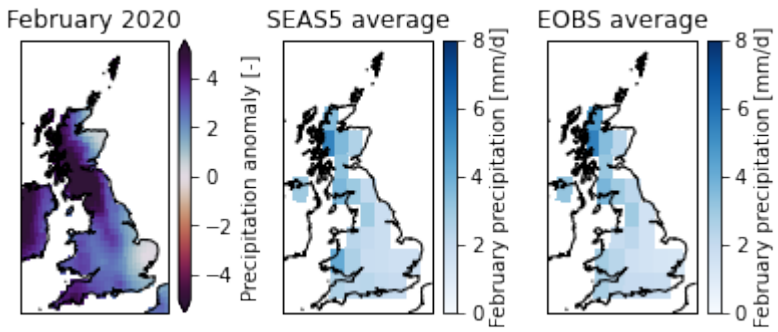
[101]:
```
<cartopy.mpl.feature_artist.FeatureArtist at 0x7f16bee162d0>
```

[101]:
```
<cartopy.mpl.feature_artist.FeatureArtist at 0x7f16bedd6910>
```

[101]:
```
Text(0.5, 1.0, 'February 2020')
```

[101]:
```
Text(0.5, 1.0, 'SEAS5 average')
```

[101]:
```
Text(0.5, 1.0, 'EOBS average')
```



1.10.6 Extract the spatial average

To select the UK average, we select SEAS5 precipitation (tprate), select the gridcells that are within the UK and take the area-weighted mean over those gridcells. This results in a dataset of February precipitation for 35 years (1981-2016), with 5 leadtimes and 25 ensemble members.

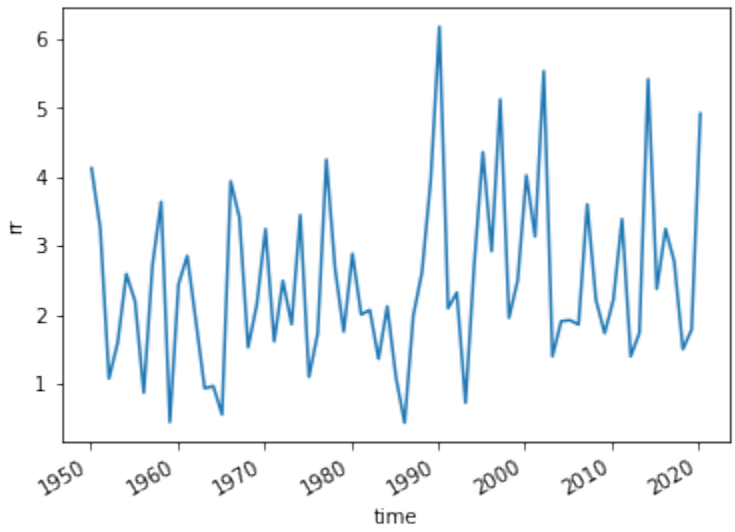
[31]:
```
SEAS5_UK_weighted = (SEAS5
                     .where(SEAS5_mask == 31)
                     .where(EOBS_upscaled['rr'].sel(time='1950').squeeze('time').
↪notnull())
                     .weighted(Gridarea_SEAS5['cell_area'])
                     .mean(dim=['latitude', 'longitude'])
                    )
SEAS5_UK_weighted
```

[31]:
```
<xarray.Dataset>
Dimensions:    (leadtime: 5, number: 51, time: 39)
Coordinates:
  * number     (number) int64 0 1 2 3 4 5 6 7 8 9 ... 42 43 44 45 46 47 48 49 50
  * time       (time) datetime64[ns] 1982-02-01 1983-02-01 ... 2020-02-01
  * leadtime   (leadtime) int64 2 3 4 5 6
Data variables:
    tprate     (leadtime, time, number) float64 1.62 1.803 3.715 ... 2.564 4.138
```

[32]:
```
EOBS_UK_weighted = (EOBS_upscaled
                    .where(SEAS5_mask == 31) ## EOBS is now on the SEAS5 grid, so use␣
↪the SEAS5 mask and gridcell area
                    .weighted(Gridarea_SEAS5['cell_area'])
                    .mean(dim=['latitude', 'longitude'])
                   )
EOBS_UK_weighted
EOBS_UK_weighted['rr'].plot()
```

[32]:
```
<xarray.Dataset>
Dimensions:  (time: 71)
Coordinates:
  * time     (time) datetime64[ns] 1950-02-28 1951-02-28 ... 2020-02-29
Data variables:
    rr       (time) float64 4.127 3.251 1.072 1.593 ... 2.774 1.498 1.782 4.92
```

[32]:
```
[<matplotlib.lines.Line2D at 0x7f16cc470b90>]
```



1.10.7 Illustrate the SEAS5 and EOBS UK average

And the area-weighted average UK precipitation for SEAS5 and EOBS I plot here. For SEAS5 I plot the range, both min/max and the 2.5/97.5 % percentile of all ensemble members and leadtimes for each year.

[33]:
```
ax = plt.axes()

Quantiles = (SEAS5_UK_weighted['tprate']
             .quantile([0,2.5/100, 0.5, 97.5/100,1],
                       dim=['number','leadtime']
                      )
            )
ax.plot(Quantiles.time, Quantiles.sel(quantile=0.5),
        color='orange',
        label = 'SEAS5 median')
ax.fill_between(Quantiles.time.values, Quantiles.sel(quantile=0.025), Quantiles.
↪sel(quantile=0.975),
                color='orange',
                alpha=0.2,
                label = '95% / min max')
ax.fill_between(Quantiles.time.values, Quantiles.sel(quantile=0), Quantiles.
↪sel(quantile=1),
                color='orange',
                alpha=0.2)

EOBS_UK_weighted['rr'].plot(ax=ax,
                            x='time',
                            label = 'E-OBS')
plt.legend(loc = 'lower left',
           ncol=2 ) #loc = (0.1, 0) upper left
```

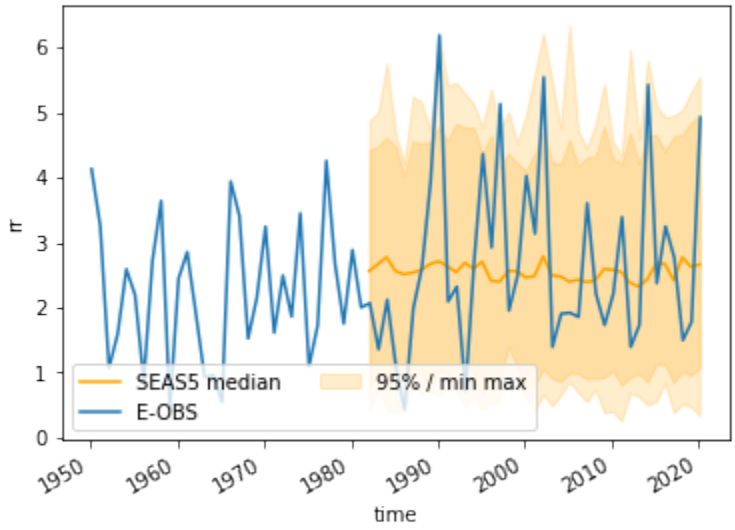
[33]:
```
[<matplotlib.lines.Line2D at 0x7f16cc417410>]
```

[33]:
```
<matplotlib.collections.PolyCollection at 0x7f16d5497610>
```

[33]:
```
<matplotlib.collections.PolyCollection at 0x7f16cc2ad7d0>
```

[33]:
```
[<matplotlib.lines.Line2D at 0x7f16cc50cb10>]
```

[33]:
```
<matplotlib.legend.Legend at 0x7f16cc3d18d0>
```



1.10.8 And save the UK weighted average datasets

[34]:
```
SEAS5_UK_weighted.to_netcdf('Data/SEAS5_UK_weighted_masked.nc')
SEAS5_UK_weighted.to_dataframe().to_csv('Data/SEAS5_UK_weighted_masked.csv')
EOBS_UK_weighted.to_netcdf('Data/EOBS_UK_weighted_upscaled.nc') ## save as netcdf
EOBS_UK_weighted.to_dataframe().to_csv('Data/EOBS_UK_weighted_upscaled.csv') ## and␣
↪save as csv.
```

[35]:
```
SEAS5_UK_weighted.close()
EOBS_UK_weighted.close()
```

1.10.9 Other methods

There are many different sources and methods available for extracting areal-averages from shapefiles. Here I have used shapely / masking in xarray. Something that lacks with this method is the weighted extraction from a shapefile, that is more precise on the boundaries. In R, raster:extract can use the percentage of the area that falls within the country for each grid cell to use as weight in averaging. For more information on this method, see the EGU 2018 course. For SEAS5, with its coarse resolution, this might make a difference. However, for it's speed and reproducibility, we have chosen to stick to xarray.

We have used xarray where you can apply weights yourself to a dataset and then calculate the weighted mean. Sources I have used: * xarray weighted reductions * Matteo's blog * regionmask package * Arctic weighted average example * area weighted temperature example.

And this pretty awesome colab notebook on seasonal forecasting regrids seasonal forecasts and reanalysis on the same grid before calculating skill scores.

CHAPTER

TWO

LICENSE

All code and example data are available under the open source MIT License.

CHAPTER

THREE

CITATION

When using the code or example data, please cite this project. If any questions may arise, please don't hesitate to get in touch t.kelder@lboro.ac.uk.