Is it just me - or it is getting HOT in here?

Implementation and comparison of different 'offline' unsupervised methods of change-point detection.

Itay Gonnen

2022-07-18

Abstract

So I was wondering whatever it is really getting hot in here or it's just me. Luckily, I learned in the last semester about Change of Point Model, and I have this felling that this is a great opportunity to test it hands on real data.

Contents

Preparing the Data

read the data:

```
raw_data <- read.csv("./data4R/GlobalLandTemperaturesByCountry.csv")
only_israel = subset(raw_data, raw_data$Country == 'Israel')
summary(only_israel)
```

```
##       dt            AverageTemperature AverageTemperatureUncertainty
##  Length:2460        Min.   : 7.166     Min.   :0.0820
##  Class :character   1st Qu.:13.922     1st Qu.:0.3010
##  Mode  :character   Median :20.190     Median :0.4805
##                     Mean   :19.627     Mean   :0.9030
##                     3rd Qu.:25.234     3rd Qu.:1.5760
##                     Max.   :30.461     Max.   :4.8710
##                     NA's   :28         NA's   :28
##    Country
##  Length:2460
##  Class :character
##  Mode  :character
##
##
##
##
```

Taking care of mixture date formats as char and Na's:

```
no_na <- na.omit(only_israel)
origin_dates = no_na$dt
bar_dates   =  as.Date(origin_dates,format="%Y-%m-%d") # Produces NA when format is not "%Y-%m-%d"
slash_dates =  as.Date(origin_dates,format="%d/%m/%d") # Produces NA when format is not "%d/%m/%d"
bar_dates[is.na(bar_dates)] = slash_dates[!is.na(slash_dates)] # Combine both while keeping their ranks
dates = bar_dates
```

Note that we want to examine temperature, which means we need to dividing the data into seasons:

```
season <- function(date, ind){
  season =0
  year = as.integer(strsplit(as.character(date),"-")[[1]][1])
  month = as.integer(strsplit(as.character(date),"-")[[1]][2])
  if(month == 1)  { season = 1  }
  if(month == 4)  { season = 2  }
  if(month == 7)  { season = 3  }
  if(month == 10) { season = 4  }
  return (c(year, season))
}
```

Organizing by seasons:

```
df = data.frame(matrix(ncol = 4, nrow = 0))
colnames(df) <- c('Year', 'Season', 'AvgTemp','AvgTempUncertainty')
ind = 1
```

```
n = nrow(no_na)
while(ind < n){
  season_temp = no_na[ind:(ind+2),]
  year_month = season(dates[ind],ind)
  df[nrow(df) + 1,] = list(year_month[1],year_month[2],
                           mean(season_temp$AverageTemperature ),
                           mean(season_temp$AverageTemperatureUncertainty))

 ind = ind+3
}
df=df[1:810,]
```

Plotting basic data:

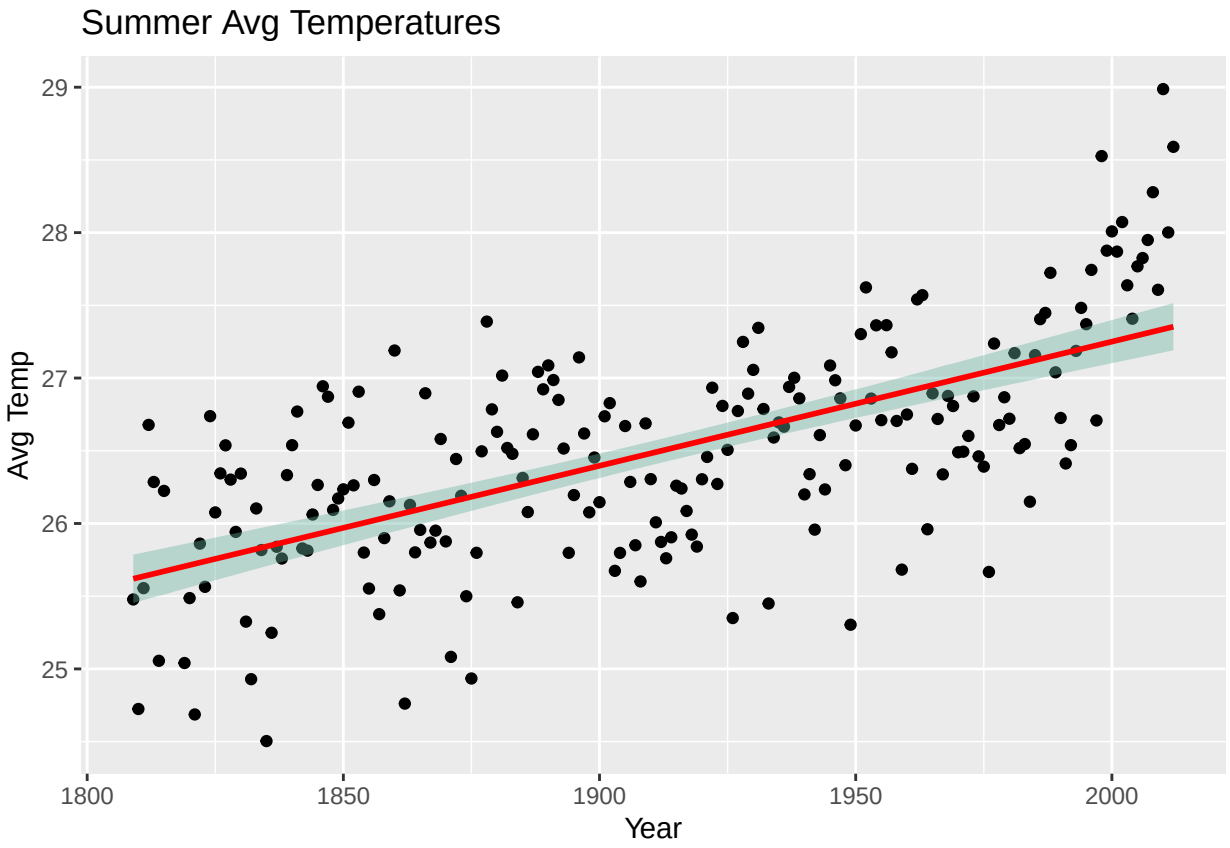
```
winter = subset(df, Season == 1)
spring = subset(df, Season == 2)
summer = subset(df, Season == 3)
autumn = subset(df, Season == 4)
```

1 Intuition

Change at first glance

Let's have a look on the summer temperatures:

```
ggplot(summer, aes(x = Year, y = AvgTemp)) + geom_point()+
       ggtitle("Summer Avg Temperatures")+ xlab("Year")+ylab("Avg Temp")+
       geom_smooth(method=lm , color="red", fill="#69b3a2", se=TRUE)
```

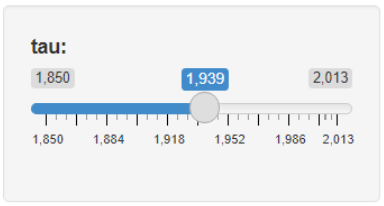


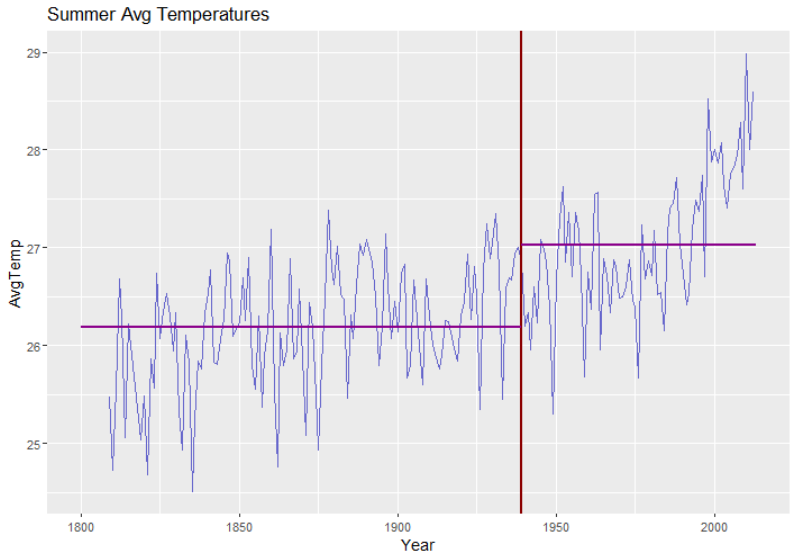
We clearly "see'' positive trend of the average temperatures. How can we automatically identify the time of such change? As said, the change we are looking for is change in the distribution - which can detect by observing the value's itselves.

The Postprior distribution

Change point, as said - is basically a time stamp which from now on we observed from a different distribution. Postprior distribution = the distribution *according* to the observed data.

So, **I encourage you to find the change point** by your self. Meaning, find the τ which balance the pink lines to best "fit" both sides of the graph (separably). Those lines are the expected value of the post-prior distribution. Meaning, the left line in μ_1 based on $y_{1:\tau} \sim f(\mu_1)$, similarly - right line represent μ_2 based on





$y_{\tau:2015} \sim f(\mu_2)$:

So far, we examine what is the idea of CPD. Now, let's examine the showcase - an implementation of a three detection algorithms for the early detection of rising trend of temperatures at in the Israel's summers.

2 - Model Comparison

Method I - MCP packege

The mcp is the most common package for change detection. One of it's advantages is the abilty to chose *the number of points* we want to find. At first try - let's guess two change points:

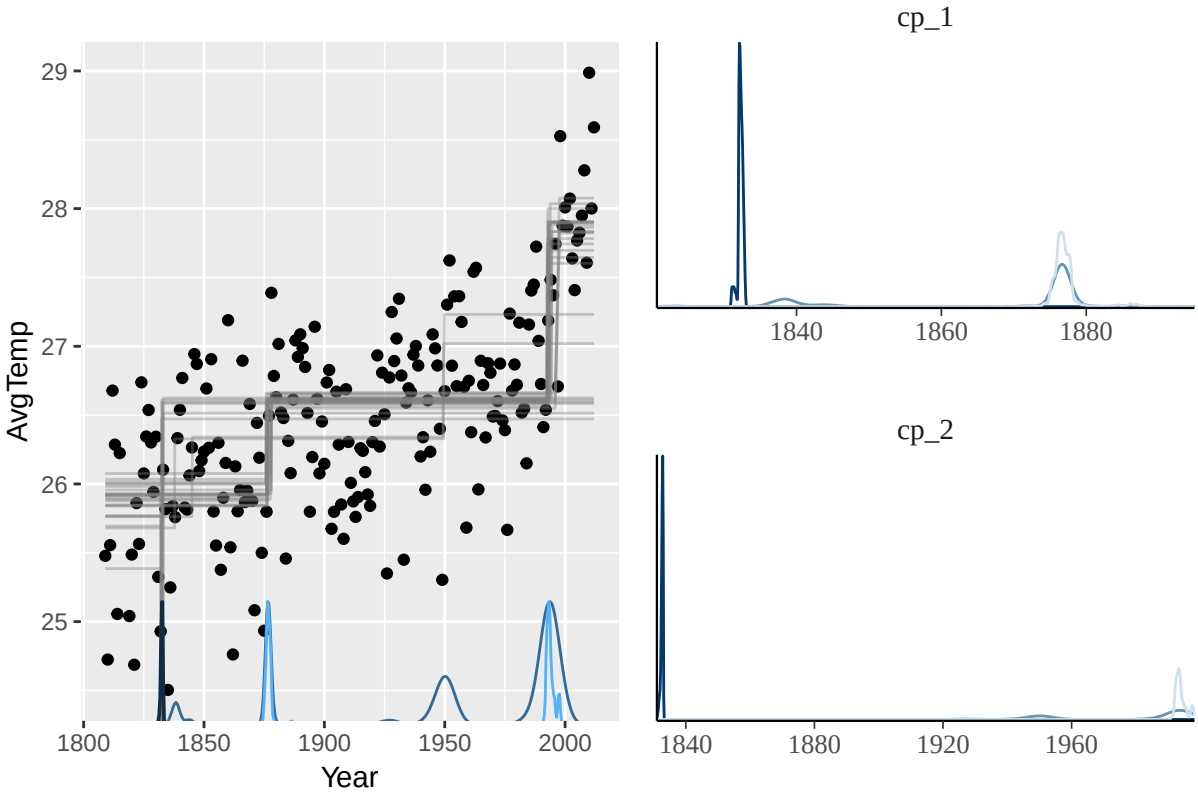
```
library ('mcp')
summer_model = list(AvgTemp~1, 1~1, 1~1)  # two intercept-only segments
fit_mcp_summer = mcp(summer_model, data = summer, par_x = "Year")
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 201
##    Unobserved stochastic nodes: 6
##    Total graph size: 3445
##
## Initializing model
```

```
summary(fit_mcp_summer)
```

```
## Family: gaussian(link = 'identity')
## Iterations: 9000 from 3 chains.
## Segments:
##   1: AvgTemp ~ 1
##   2: AvgTemp ~ 1 ~ 1
##   3: AvgTemp ~ 1 ~ 1
##
## Population-level parameters:
##     name    mean   lower   upper Rhat n.eff
##     cp_1 1858.96 1831.01 1878.15  8.2   321
##     cp_2 1935.43 1832.13 1997.70 16.4   208
##    int_1   25.87   25.55   26.10  1.7  2847
##    int_2   17.64   -3.05   26.75 10.3  2920
##    int_3   27.37   26.47   28.10  6.4  2395
##  sigma_1    0.62    0.51    0.78  4.5  3600
```

```
library('patchwork')
plot(fit_mcp_summer) + plot_pars(fit_mcp_summer, pars = c("cp_1","cp_2"), type = "dens_overlay")
```



The summary shows good parameter recovery, though the second change point is detected a bit early. This is understandable if you look at the data, and the true change point is still within the highest density interval.

Cross Validation for MCP

Wait, who said there are only 2 cpd?

We can test that assumption with Cross-Validation (*'leave one out'* method):

```
model1 = list(AvgTemp~1,1~1)            # two intercept-only segments
model2 = list(AvgTemp~1,1~1,1~1)        # three intercept-only segments
##
fit_mcp1 = mcp(model1, data = summer, par_x = "Year")
```

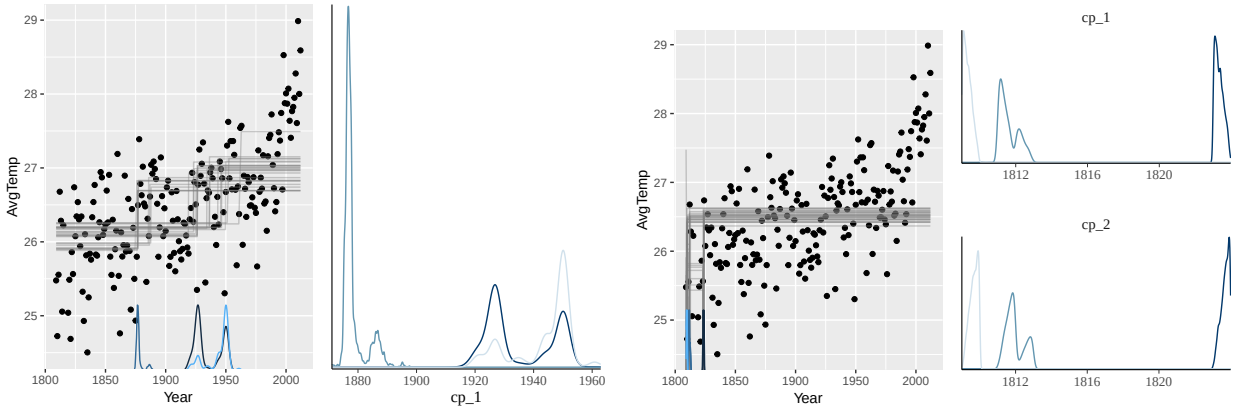
```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 201
##    Unobserved stochastic nodes: 4
##    Total graph size: 2432
##
## Initializing model
```

```
fit_mcp2 = mcp(model2, data = summer, par_x = "Year")
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 201
##    Unobserved stochastic nodes: 6
##    Total graph size: 3445
##
## Initializing model
```

compare distributions:

```
plot(fit_mcp1) + plot_pars(fit_mcp1, pars = c("cp_1"), type = "dens_overlay")
plot(fit_mcp2) + plot_pars(fit_mcp2, pars = c("cp_1", "cp_2"), type = "dens_overlay")
```



Observing the plots, we can see that when we ask from MCP to find two change points, it's dividing the one into two points at the edges of the change segment (form 1890 to 1960). Preforming comparison with CV:

```
fit_mcp1$loo = loo(fit_mcp1)
fit_mcp2$loo = loo(fit_mcp2)

loo::loo_compare(fit_mcp1$loo, fit_mcp2$loo)
```

```
##        elpd_diff se_diff
## model1   0.0       0.0
## model2 -24.0       5.7
```

Seems that the model with only one point's fit's better! Meaning, we need to decide which of cp_1, cp_2 is the "real" change point. For this task, we can use another CP methods.

Method II - The Changepoint packege

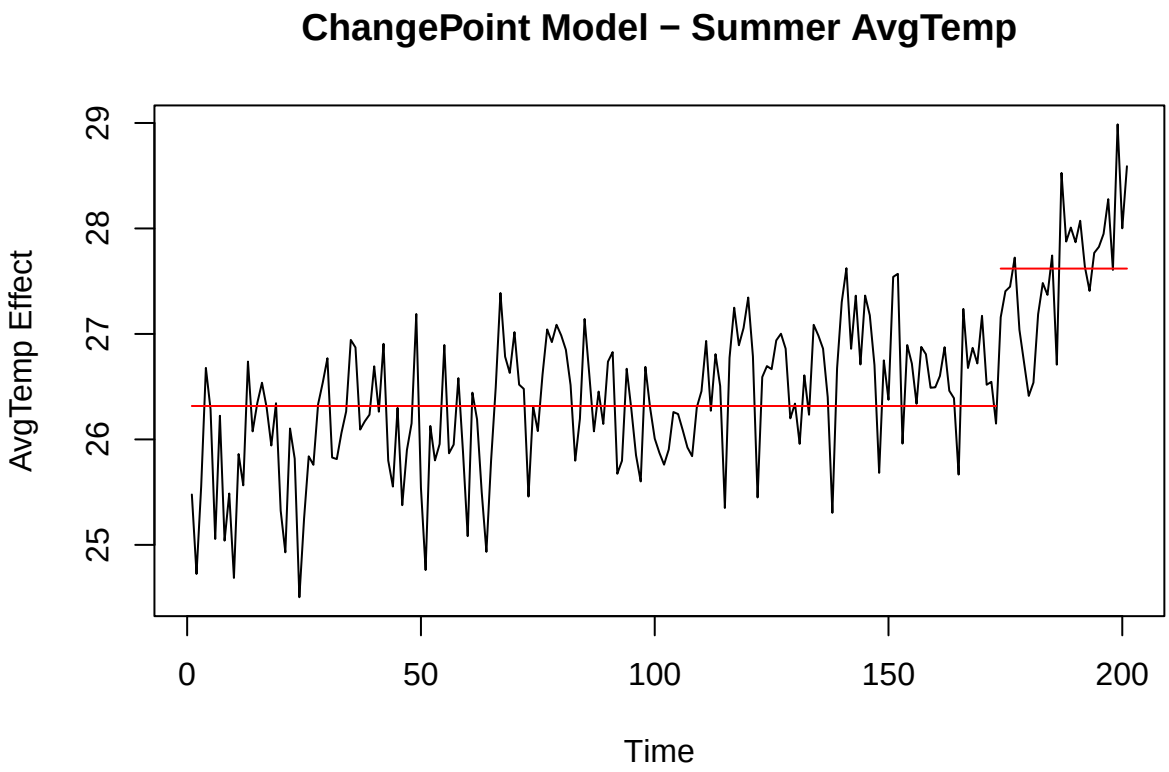
The next model is ChangePoint package. It's advantage is it's non-parametric approach. Meaning, it it concentrate on fitting the data by it's trend (very similar to the interactive example):

```
library(changepoint)
fit_changepoint_summer = cpt.mean(summer$AvgTemp)

# Return estimates
c(ints = param.est(fit_changepoint_summer)$mean,
cp = cpts(fit_changepoint_summer))
```

```
##     ints1     ints2        cp
##  26.31691  27.61955 173.00000
```

```
plot(fit_changepoint_summer, main= "ChangePoint Model - Summer AvgTemp", ylab = "AvgTemp Effect")
```

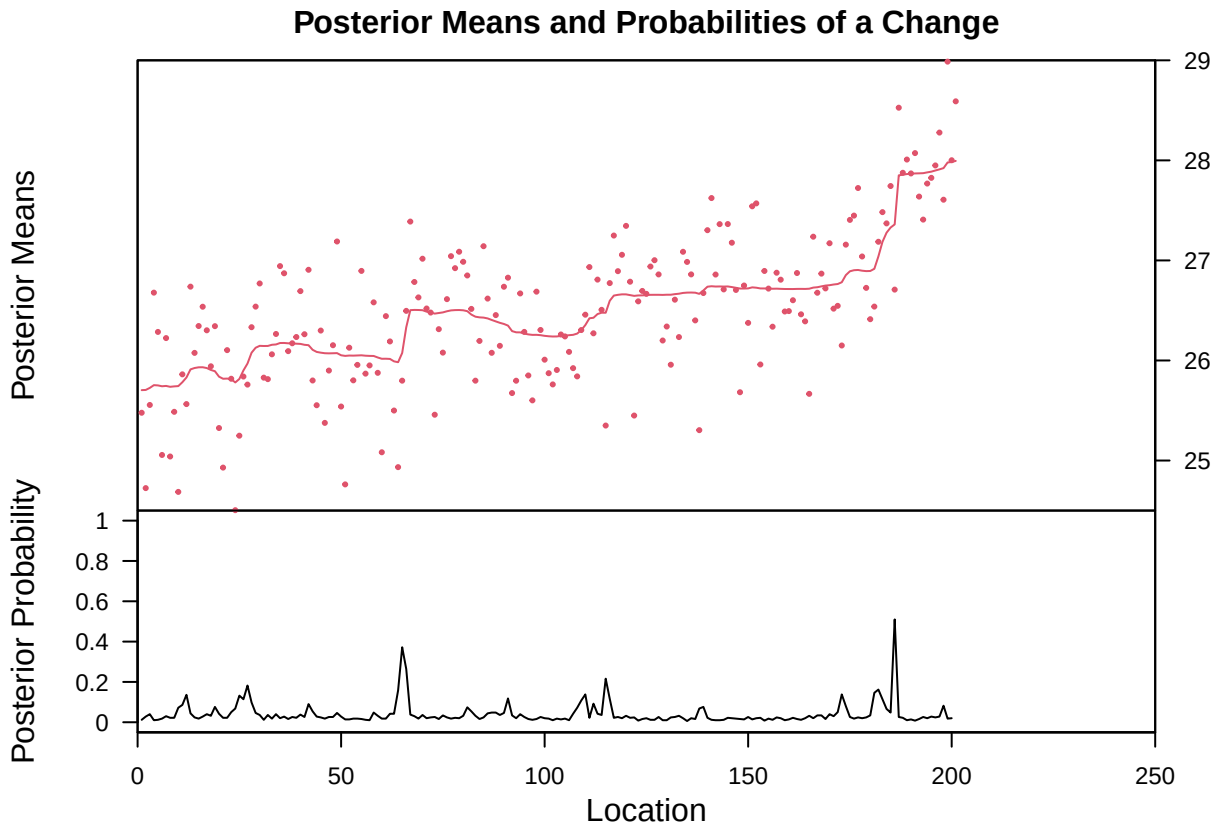


Interesting! CP detect only one change point - $cp_2 \sim 1970$. Indeed, if we get back to MCP plot, we can see that the "jump" at the cp_2 (the grey trajectorys) is higher.As said, CP detect changes by looking at the trend slop at each point.

Method III - The BCP packege

What's unique in BCP it is is pure Bayesian approach. It's gives you the Posterior density of your data. The upper plot is the estimated mean (the purple line from above!) at each point.

```
library(bcp)
fit_bcp = bcp(summer$AvgTemp, d = 1000)
plot(fit_bcp)
```

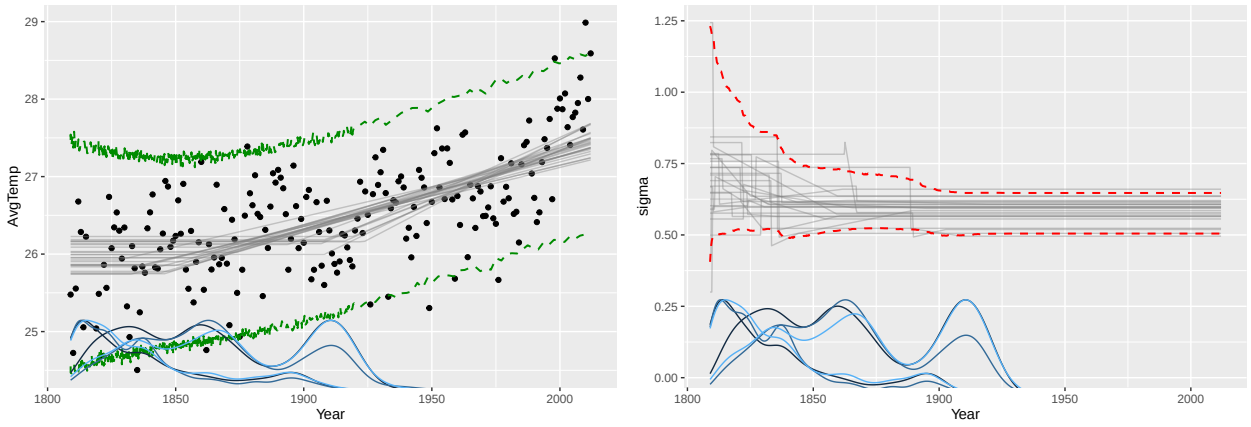


We see that it vaguely captures the change point at x = 60, and has some smeared-out probability around x = 170 (which is similar to Changepoint findings!).

3- One last observation

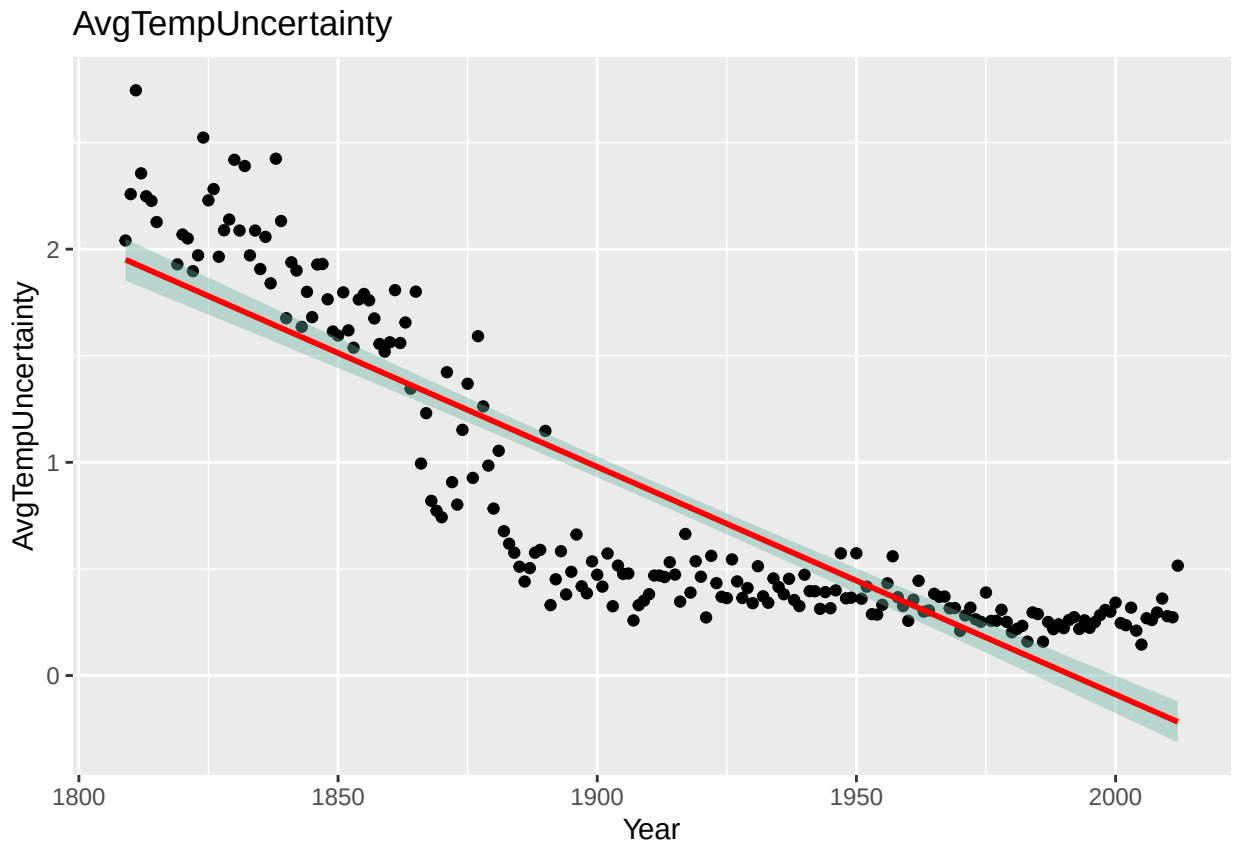
At least, we did not mention another aspect of changing - changing of the **variance of the distribution**. We can model variance by adding a sigma() term to the formula. Plotting the prediction interval is an intuitive way to to see how the variance is estimated. Small example upon our data:

```
var_model = list(
  AvgTemp ~ 1,
   ~ 0 + sigma(1 + Year),
   ~ 0 + Year
)
fit = mcp(var_model, data = summer,cores = 3, adapt = 5000, iter = 5000)
plot(fit, q_predict = TRUE)
plot(fit, which_y = "sigma", q_fit = TRUE)
```



The left plot is our data with it's variance interval at each year. The right plot is the variance interval at each year. At first observation, the plot is not informative. But note that the grey trajectories does have the same tendency. indeed - if we take a look on the Uncertainty of the data we have decreasing trend, and even a significant one around cp_2.

```
ggplot(summer, aes(x = Year, y = AvgTempUncertainty)) + geom_point()+
       ggtitle("AvgTempUncertainty")+
       geom_smooth(method=lm , color="red", fill="#69b3a2", se=TRUE)
```



4 - Summary

So, cp_2 is Definitely our winner for this contest. All three algorithms had detected this point as an interval of changing of the Israeli's summer temperatures distribution. Note, that we can pay attention to the change *interval*. MCP predicted 70-80 years interval around each point. CP predicted almost 60 years interval but symmetrically around cp_2 (BCP doesn't provide point).