Climarkplus Package

William Hughes

August 31, 2014

Abstract

A reimplimentation in R of some of the concepts and methods of R. Stern's Climate package in Instat extended to allow for more general modelling

1 Design

1.1 intro

The package is not a port of the Climate package for Instat and Genstat, written by R. Stern, however, in some ways it is quite similar

The major difference is that fitting is done using a generalized linear model on the unbinned data rather than to estimated probabilities. Modifications to the basic Markov model can be made (e.g. a time term) and evaluated.

Other differences include the fact that R does not have many of the limitations of Instat. and the "spreadsheet model", in which everything is more or less a matrix, is only partially used. Columns are referred to by name, not number.

2 Example

2.1 Data Set

We will work with a dataset of 83 years of data from the Zaza, Rawanda station. This has been put into R form using functions from Helen Greatrex.

```
> data(zaza)
> head(zaza)
```

```
        Date     Inputfile Station Year Day Month Rain TMax TMin
1 1930-10-01 Zaza mod1.txt    Zaza 1930  01    10    0   NA   NA
2 1930-10-02 Zaza mod1.txt    Zaza 1930  02    10    8   NA   NA
3 1930-10-03 Zaza mod1.txt    Zaza 1930  03    10   43   NA   NA
4 1930-10-04 Zaza mod1.txt    Zaza 1930  04    10    0   NA   NA
5 1930-10-05 Zaza mod1.txt    Zaza 1930  05    10    0   NA   NA
6 1930-10-06 Zaza mod1.txt    Zaza 1930  06    10   45   NA   NA
```

```
> tail(zaza)

            Date     Inputfile Station Year Day Month Rain TMax TMin
29732 2012-02-24 Zaza mod1.txt    Zaza 2012  24    02  0.0   NA   NA
29733 2012-02-25 Zaza mod1.txt    Zaza 2012  25    02  0.0   NA   NA
29734 2012-02-26 Zaza mod1.txt    Zaza 2012  26    02  0.0   NA   NA
29735 2012-02-27 Zaza mod1.txt    Zaza 2012  27    02  1.4   NA   NA
29736 2012-02-28 Zaza mod1.txt    Zaza 2012  28    02  0.0   NA   NA
29737 2012-02-29 Zaza mod1.txt    Zaza 2012  29    02  0.0   NA   NA
```

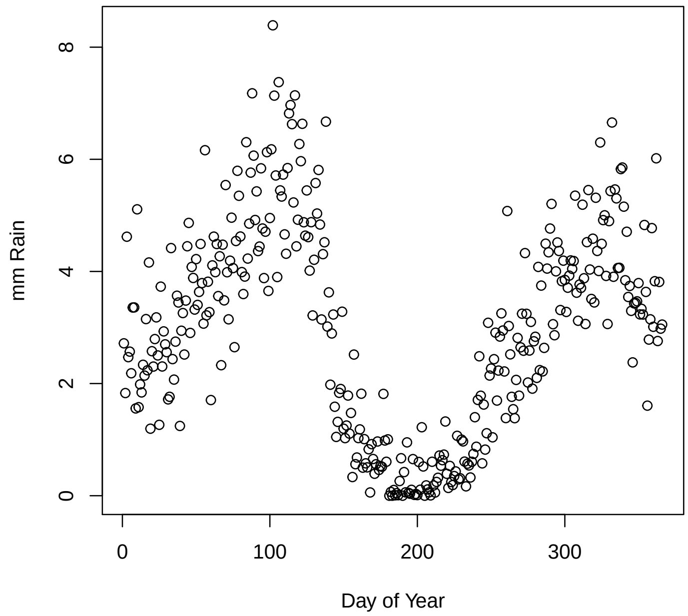
It is useful to have a dataset with day of year (consistent in that March 1 is day 61 for non leap years as well as leap years). The function `convert_data` does this.

```
> zaza_doy=convert_data(zaza)
> head(zaza_doy)

  Station       Date Rain DOY
1    Zaza 1930-10-01    0 275
2    Zaza 1930-10-02    8 276
3    Zaza 1930-10-03   43 277
4    Zaza 1930-10-04    0 278
5    Zaza 1930-10-05    0 279
6    Zaza 1930-10-06   45 280
```

We can plot the average rain over the year (more on the details of this later)

```
>  plot(sapply(split(zaza_doy$Rain,zaza_doy$DOY),mean,na.rm=T),
+       ylab="mm Rain",xlab="Day of Year")
```



Note the rainfall has two peaks, but does not fall to 0 in Dec/Jan.

2.2 Markov Model

The first thing we do is to add the Markov lags, up to order=2.

```
> zaza_wm=add_markov(zaza_doy)
> #load("zaza_wm")
> head(zaza_wm)
```

```
  Station       Date Rain DOY wet_or_dry lag_1 lag_2
1    Zaza 1930-10-01    0 275          d  <NA>  <NA>
2    Zaza 1930-10-02    8 276          w     d  <NA>
3    Zaza 1930-10-03   43 277          w     w    wd
4    Zaza 1930-10-04    0 278          d     w    ww
5    Zaza 1930-10-05    0 279          d     d    dw
6    Zaza 1930-10-06   45 280          w     d    dd
```

Note there are three new columns. ”d” means a dry day and ”w” means any day in which the amount of rain is more than some threshold (default 0.12 mm). Lag_n is the pattern of wet and dry days over the previous n days.

2.3 Philosophy of Model

The basic idea of the package is to model and analyze rainfall. To determine the amount of rainfall on a given day we use two steps.

1. Get the probability that there will be rain.
2. Get the distribution of the amount of rain on rainy days

2.3.1 Probability of Rain

We use a markov model of order k of the probability of rain, that is the chance of rain will depend on the pattern of wet and dry days over the previous k days. The order can be chosen (standard is an order of two, e.g in the given model)

If the order is k, then there are 2^k possible patterns of wet, w, and dry, d, days. For $k = 2$ we have ww, dw, wd, dd. Note for every pattern there is a column in the model called $P(w|\text{pattern})$ For each day (the rows 1–366) we have the probability applicable to that day. There is no restriction on where these values come from. They can be fitted values from raw data, however there are other possibities.

2.3.2 Amount of rain

We use a markov model of order k for both the mean of the rain. Hence, the mean ammount of rain will depend of the pattern of wet and dry days over the previous k days. (It is assumed that the amount of rain follows a Gamma distribution with a constant shape) The order can be chosen (standard is an order of 1, or 0 (do not take into account any pattern))

If the order is k, then there are 2^k possible patterns of wet, w, and dry, d, days. For $k = 1$ we have w, d Note for every pattern there is a column in the model called $<(r|\text{pattern}>)$. For each day (the rows 1–366) we have the mean. There is no restriction on where these values come from. Note we also have the $<\text{rain}>$ column This is the unconditional mean of the rain amount.

Note though there is a day of year dependence in the model there is (as yet) no year dependence.

2.3.3 Obtaining Values: shape, offset

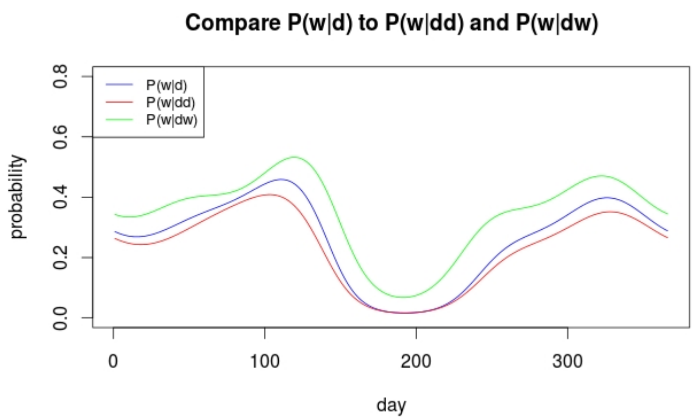
Each column, whether probability or amount is said to be a "curve". As indicated, the model does not know where the curve came from. However, it is often usefull to break the curve down into two parts. a "shape" and an "offset". The shape is any general curve, the offset is a single number applied to the shape to get the final curve. So

$$(curve) = \text{shape} + \text{offset} \tag{1}$$

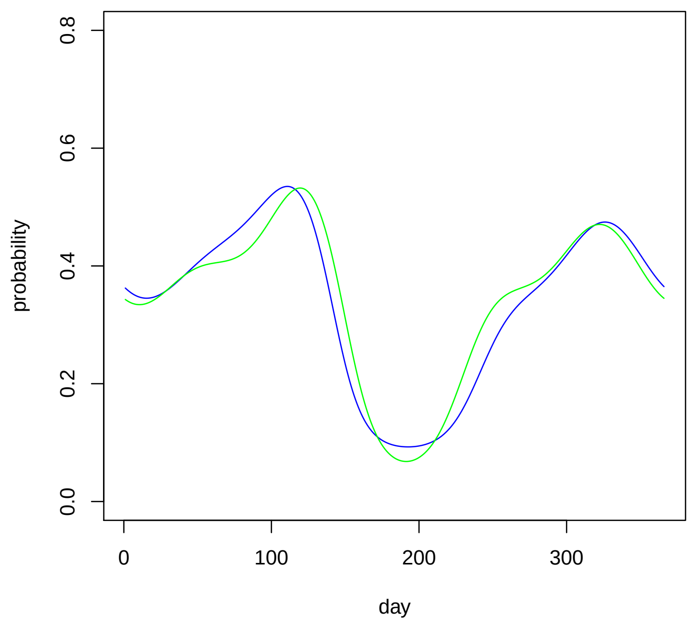
(We do not attempt to make a canonical offset for the shape. We need the weighted sum of squares fit to a constant to be zero, but in practice the weighting is not known)

The motivation here, is that it is often noted that higher order curves (eg $P(w|\text{dw})$) are often very similar to lower order curves (eg $P(w|\text{d})$) but with an offset. Thus, it makes sense to estimate the higher order curve by estimating only this offset, rather than the large number of parameters needed to directly estimate the higher order curve.

An example can help here. We look at the same site that was used to produce the above model.



The blue line is the probability of rain given that the previous day was dry, $P(w|\text{d})$. This can be broken down into $P(w|\text{dd})$, the red line, and $P(w|\text{dw})$, the green line. We note that the green line has much the same shape as the blue line but with a substantial offset.



Here we see $P(w|\mathrm{d})$ (the blue line) shifted to match $P(w|\mathrm{dw})$ (the green line). It seems that we can use the shifted blue line rather than the green line. Given that we have much more confidence in the shifted blue line (more data, fewer coefficients) that the green line, we may prefer to use the shifted blue line.

3 Parameter Files

On way of creating a model is to start with a set of raw probabilities, `raw_probs`, derived from a dataset associated (usually) with a station. Then each curve $P(w|\mathrm{lag})$ can be a fitted version of the corresponding data in the dataset, or a (possibly shifted) fitted version of some other data from the dataset.

The model is described by a `parameter (*.pl)` file. An example of the start of such a file is given.

```
<order> = 2
<dd> = dd
<dd_fit_order>= choose
<dd_offset> = NO
...
```

Every line has the form

```
<key> = value
```

When read in you get a list, with `list[key] = value` with value a string.

The first parameter, `order`, gives the order of the wet/dry part of the model. For each of the 2^{order} values for the `lag`, there are three parameters. The first `lag` is the column of the raw dataset used; the second `lag_fit_order` is the number of harmonics used to fit the raw data (if this value is `choose` then the fit order is determined automatically); the third `lag_offset` is the dataset from which the offset is to be determined, if the value is `NO` then there is no offset.

The part of the parameter file which deals with the amount of rain on wet days is similar, although in this case we have `lag` for the mean and standard deviation.

```
<rain_order>= 1
<rw>=w
<rw_fit_order>=4
<rw_offset> = NO
...
```

This is the "standard" parameter file, `order_2_0.pl`. It describes an order 2 model for the probability of rain, and an order zero model for the amount of rain.

```
<order> = 2
<dd> = dd
<dd_fit_order>= 4
<dd_offset> = NO
<dw> = dw
<dw_fit_order>= 4
<dw_offset> = NO
<wd>= wd
<wd_fit_order>= 4
<wd_offset> = NO
<ww> = ww
<ww_fit_order>= 4
<ww_offset> = NO

rain_order>= 0
<r0_fit_order>= 4
```

If we want an order 1 model for the amount of rain, we change only the rain section (last three line) to

```
<rain_order>= 1
<rw>=w
<rw_fit_order>=4
<rw_offset> = NO
<rd>=d
```

```
<rd_fit_order>=4
<rd_offset> = NO
```

We can get a mixed markov model by changing what is used to estimate higher order lags.

```
<order> = 2
<dd> = d
<dd_fit_order>= 4
<dd_offset> = NO
<dw> = d
<dw_fit_order>= 4
<dw_offset> = NO
<wd>= wd
<wd_fit_order>= 4
<wd_offset> = NO
<ww> = ww
<ww_fit_order>= 4
<ww_offset> = NO


<rain_order>= 0
<r0_fit_order>= 4
```

Note that the curves for the order two lags dd and dw are both derived from the order 1 curve lag d. However, the curves for the order 2 lags wd and ww are derived from order two curves. In other words we only use order 2 lags if the first day is wet.

We can add offsets if desired. Let us change the offset for the curve for lag dw.

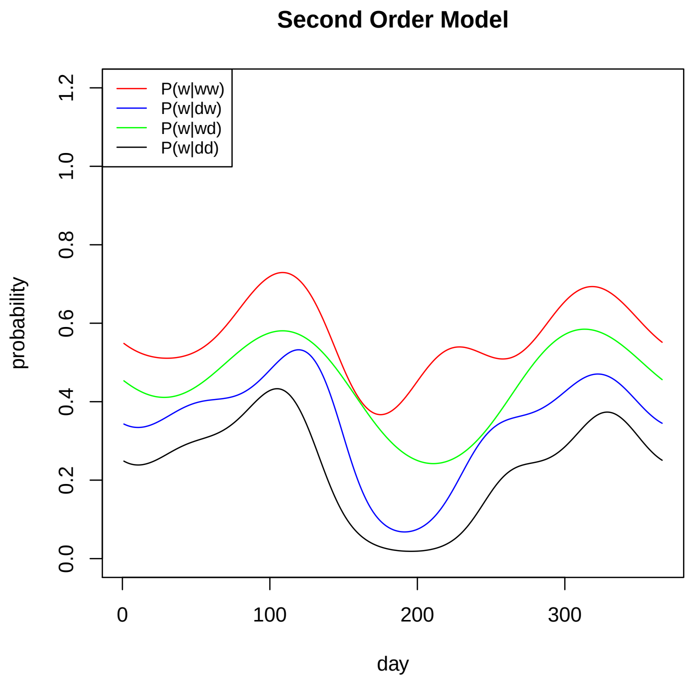
```
<dw> = d
<dw_fit_order>= 4
<dw_offset> = dw
```

Note that we still derive the curve from the first order lag, but now we add an offset derived from the second order lag.

We can illustrate this by using the above data set. (here we are looking a only the probability of rain)

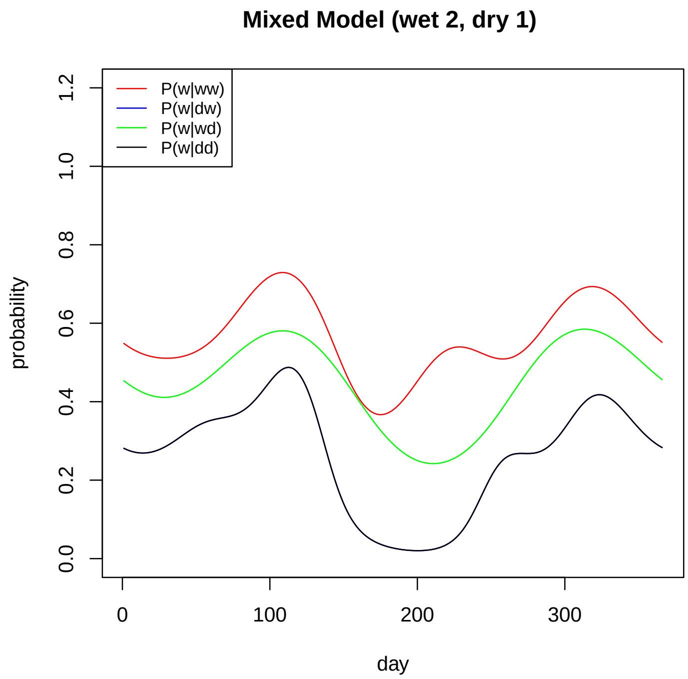
First we show a second order model

```
> mod=make_approx_model_pl(zaza_pbs,"/home/william/Reading/rstudio/Climarkplus/inst/paramete
> plot_model(mod,"Second Order Model")
```



As we have noted above, the wd, and ww, lines are not close in shape to the w line, but the dd and dw lines are close in shape to the d line. So we can try a mixed model, going to order 2 lags if the previous day was wet, but using only order 1 if the previous day was dry

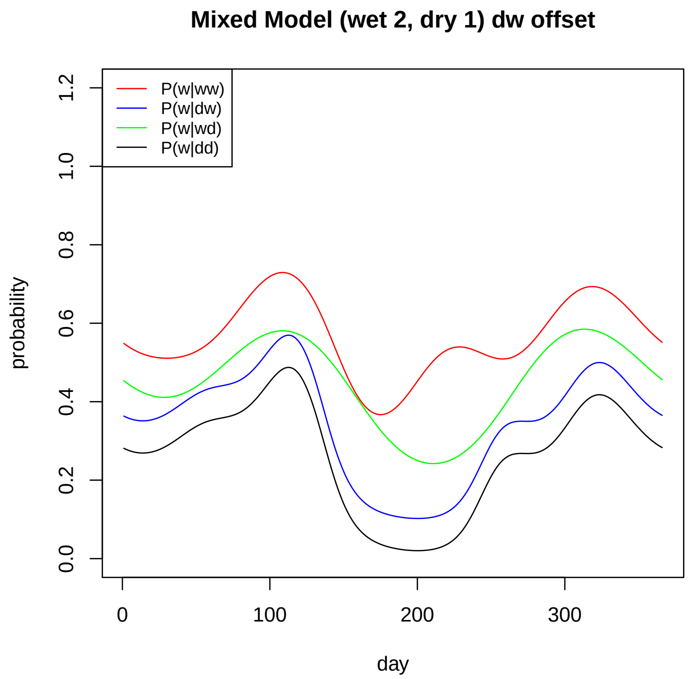
```
> mod=make_approx_model_pl(zaza_pbs,"/home/william/Reading/rstudio/climate/trunk/inst/parame
> plot_model(mod,"Mixed Model (wet 2, dry 1) ")
```



Note that `P(w|dw)` and `P(w|dd)` are identical. The latter obscures the former.

We note from above that the d line can be used for dd but while the dw line is similar in shape to dd there is an obvious offset. We can include this offset.

```
> mod=make_approx_model_pl(zaza_pbs,"/home/william/Reading/rstudio/climate/trunk/inst/parame
> plot_model(mod,"Mixed Model (wet 2, dry 1) dw offset")
```



Note that the dw line is now visible. It has the same shape as the dd (and hence d) line, but not the same offset.

3.1 Fitting

There are two functions that do the true fitting: `fit_rainy` for the probability of rain; and `fit_amount` for the amount of rain. Both functions take the same parameters

wms This is the raw data. It must have column DOY and it must have columns for the needed Markov lags

filename The name of the parameter file used to guide the fit. Standard parameter files can be founc in `inst/parameter`

others This is a vector of names of other predictors that should be used in the fit (no interactions). Anything in others should also be a column in the raw data.

other_model_string This can be anything you want. It is a string that is added verbatim to the fitting string. This can be used e.g. to study interactions. However, if this is used the fit object produced probably cannot be used to construct a model for synthesis

The output of both functions is a `fit_object`. This is a list of two items. The first is a list of information, the second is the fit, an R object.

An example

```
> fit_object_1=fit_rainy(zaza_wm,filename="/home/william/Reading/rstudio/Climarkplus/inst/pa
> fit_object_1[[1]]

[[1]]
          order              dd    dd_fit_order       dd_offset              dw
            "2"            "dd"        "choose"            "NO"            "dw"
   dw_fit_order       dw_offset              wd    wd_fit_order       wd_offset
       "choose"            "NO"            "wd"        "choose"            "NO"
             ww    ww_fit_order       ww_offset      rain_order    r0_fit_order
           "ww"        "choose"            "NO"             "0"             "4"
r0_sd_fit_order
            "4"

[[2]]
[1] "end"

> summary(fit_object_1[[2]])

Call:
glm(formula = fit_string, family = "binomial", data = wms)

Deviance Residuals:
   Min      1Q  Median      3Q     Max
-1.616  -0.919  -0.365   1.035   2.862

Coefficients:
                                  Estimate Std. Error z value Pr(>|z|)
ULAGSdd:cos(DOY * 0 * 2 * pi/366) -1.48832    0.02926  -50.87  < 2e-16 ***
ULAGSdw:cos(DOY * 0 * 2 * pi/366) -0.74892    0.06142  -12.19  < 2e-16 ***
ULAGSwd:cos(DOY * 0 * 2 * pi/366) -0.22048    0.04840   -4.56  5.2e-06 ***
ULAGSww:cos(DOY * 0 * 2 * pi/366)  0.26700    0.05441    4.91  9.2e-07 ***
ULAGSdd:cos(DOY * 1 * 2 * pi/366)  1.13404    0.04582   24.75  < 2e-16 ***
ULAGSdw:cos(DOY * 1 * 2 * pi/366)  0.64009    0.10575    6.05  1.4e-09 ***
ULAGSwd:cos(DOY * 1 * 2 * pi/366)  0.34052    0.08043    4.23  2.3e-05 ***
ULAGSww:cos(DOY * 1 * 2 * pi/366)  0.23568    0.09304    2.53  0.01131 *
ULAGSdd:sin(DOY * 1 * 2 * pi/366)  0.39057    0.03628   10.77  < 2e-16 ***
ULAGSdw:sin(DOY * 1 * 2 * pi/366)  0.23388    0.05558    4.21  2.6e-05 ***
ULAGSwd:sin(DOY * 1 * 2 * pi/366)  0.15623    0.05135    3.04  0.00234 **
ULAGSww:sin(DOY * 1 * 2 * pi/366)  0.07820    0.05172    1.51  0.13054
ULAGSdd:cos(DOY * 2 * 2 * pi/366) -0.84844    0.04301  -19.72  < 2e-16 ***
ULAGSdw:cos(DOY * 2 * 2 * pi/366) -0.60359    0.08805   -6.86  7.1e-12 ***
ULAGSwd:cos(DOY * 2 * 2 * pi/366) -0.30668    0.06923   -4.43  9.4e-06 ***
ULAGSww:cos(DOY * 2 * 2 * pi/366) -0.31485    0.07842   -4.01  5.9e-05 ***
```

```
ULAGSdd:sin(DOY * 2 * 2 * pi/366) -0.43209    0.03829  -11.29  < 2e-16 ***
ULAGSdw:sin(DOY * 2 * 2 * pi/366) -0.37117    0.06830   -5.43  5.5e-08 ***
ULAGSwd:sin(DOY * 2 * 2 * pi/366) -0.40672    0.06036   -6.74  1.6e-11 ***
ULAGSww:sin(DOY * 2 * 2 * pi/366) -0.24445    0.06400   -3.82  0.00013 ***
ULAGSdd:cos(DOY * 3 * 2 * pi/366)  0.33172    0.04113    8.06  7.3e-16 ***
ULAGSdw:cos(DOY * 3 * 2 * pi/366)  0.29888    0.07109    4.20  2.6e-05 ***
ULAGSwd:cos(DOY * 3 * 2 * pi/366)  0.00489    0.06047    0.08  0.93560
ULAGSww:cos(DOY * 3 * 2 * pi/366)  0.11722    0.06388    1.83  0.06651 .
ULAGSdd:sin(DOY * 3 * 2 * pi/366)  0.02232    0.03894    0.57  0.56647
ULAGSdw:sin(DOY * 3 * 2 * pi/366)  0.06524    0.06858    0.95  0.34149
ULAGSwd:sin(DOY * 3 * 2 * pi/366) -0.08162    0.06070   -1.34  0.17877
ULAGSww:sin(DOY * 3 * 2 * pi/366) -0.24936    0.06087   -4.10  4.2e-05 ***
ULAGSdd:cos(DOY * 4 * 2 * pi/366) -0.14488    0.03870   -3.74  0.00018 ***
ULAGSdw:cos(DOY * 4 * 2 * pi/366) -0.22765    0.06077   -3.75  0.00018 ***
ULAGSwd:cos(DOY * 4 * 2 * pi/366) -0.07142    0.05568   -1.28  0.19956
ULAGSww:cos(DOY * 4 * 2 * pi/366) -0.09873    0.05513   -1.79  0.07331 .
ULAGSdd:sin(DOY * 4 * 2 * pi/366) -0.07877    0.03889   -2.03  0.04280 *
ULAGSdw:sin(DOY * 4 * 2 * pi/366) -0.05261    0.06062   -0.87  0.38541
ULAGSwd:sin(DOY * 4 * 2 * pi/366) -0.06239    0.05553   -1.12  0.26122
ULAGSww:sin(DOY * 4 * 2 * pi/366)  0.11769    0.05409    2.18  0.02956 *
---
Signif. codes:  0 âĂŸ***âĂŹ 0.001 âĂŸ**âĂŹ 0.01 âĂŸ*âĂŹ 0.05 âĂŸ.âĂŹ 0.1 âĂŸ âĂŹ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 30758  on 22187  degrees of freedom
Residual deviance: 23633  on 22151  degrees of freedom
  (7550 observations deleted due to missingness)
AIC: 23705

Number of Fisher Scoring iterations: 6
```

We see that the first member of the information list is the contents of the parameter file. The fit is the second member, summary gives the coefficients in a nice form. Note that the coefficients are Fourier Coeffiencts for the levels of ULAGS. ULAGS is a new column added to the raw data. It is similar to lags_n (n the order of the Markov fit) but some lags may be combined (not in this example though)

We can experiment with adding something to the `others` parameter. First make a column of the Julian Day and add it to zaza_wm

```
> Julian = julian(as.Date(zaza_wm$Date))
> zaza_wm["Julian"]=Julian
> head(zaza_wm)

  Station       Date Rain DOY wet_or_dry lag_1 lag_2 Julian
1    Zaza 1930-10-01    0 275          d  <NA>  <NA> -14337
```

```
2    Zaza 1930-10-02    8 276          w     d  <NA> -14336
3    Zaza 1930-10-03   43 277          w     w    wd -14335
4    Zaza 1930-10-04    0 278          d     w    ww -14334
5    Zaza 1930-10-05    0 279          d     d    dw -14333
6    Zaza 1930-10-06   45 280          w     d    dd -14332
```

We can now add Julian as a predictor

```
> fit_object_1a=fit_rainy(zaza_wm,filename="/home/william/Reading/rstudio/Climarkplus/inst/p
> summary(fit_object_1a[[2]])

Call:
glm(formula = fit_string, family = "binomial", data = wms)

Deviance Residuals:
   Min      1Q  Median      3Q     Max
-1.655  -0.917  -0.366   1.034   2.871

Coefficients:
                                   Estimate Std. Error z value Pr(>|z|)
Julian                             4.75e-06   2.03e-06    2.34  0.01904 *
ULAGSdd:cos(DOY * 0 * 2 * pi/366) -1.48e+00   2.96e-02  -49.91  < 2e-16 ***
ULAGSdw:cos(DOY * 0 * 2 * pi/366) -7.41e-01   6.15e-02  -12.04  < 2e-16 ***
ULAGSwd:cos(DOY * 0 * 2 * pi/366) -2.12e-01   4.85e-02   -4.37  1.3e-05 ***
ULAGSww:cos(DOY * 0 * 2 * pi/366)  2.73e-01   5.45e-02    5.02  5.2e-07 ***
ULAGSdd:cos(DOY * 1 * 2 * pi/366)  1.13e+00   4.58e-02   24.77  < 2e-16 ***
ULAGSdw:cos(DOY * 1 * 2 * pi/366)  6.41e-01   1.06e-01    6.06  1.4e-09 ***
ULAGSwd:cos(DOY * 1 * 2 * pi/366)  3.41e-01   8.04e-02    4.24  2.2e-05 ***
ULAGSww:cos(DOY * 1 * 2 * pi/366)  2.41e-01   9.31e-02    2.59  0.00964 **
ULAGSdd:sin(DOY * 1 * 2 * pi/366)  3.91e-01   3.63e-02   10.76  < 2e-16 ***
ULAGSdw:sin(DOY * 1 * 2 * pi/366)  2.37e-01   5.56e-02    4.26  2.0e-05 ***
ULAGSwd:sin(DOY * 1 * 2 * pi/366)  1.59e-01   5.14e-02    3.10  0.00194 **
ULAGSww:sin(DOY * 1 * 2 * pi/366)  8.29e-02   5.18e-02    1.60  0.10908
ULAGSdd:cos(DOY * 2 * 2 * pi/366) -8.49e-01   4.30e-02  -19.74  < 2e-16 ***
ULAGSdw:cos(DOY * 2 * 2 * pi/366) -6.04e-01   8.81e-02   -6.86  7.0e-12 ***
ULAGSwd:cos(DOY * 2 * 2 * pi/366) -3.07e-01   6.92e-02   -4.43  9.3e-06 ***
ULAGSww:cos(DOY * 2 * 2 * pi/366) -3.18e-01   7.84e-02   -4.05  5.1e-05 ***
ULAGSdd:sin(DOY * 2 * 2 * pi/366) -4.31e-01   3.83e-02  -11.26  < 2e-16 ***
ULAGSdw:sin(DOY * 2 * 2 * pi/366) -3.74e-01   6.83e-02   -5.47  4.6e-08 ***
ULAGSwd:sin(DOY * 2 * 2 * pi/366) -4.09e-01   6.04e-02   -6.77  1.3e-11 ***
ULAGSww:sin(DOY * 2 * 2 * pi/366) -2.45e-01   6.40e-02   -3.83  0.00013 ***
ULAGSdd:cos(DOY * 3 * 2 * pi/366)  3.29e-01   4.11e-02    8.01  1.2e-15 ***
ULAGSdw:cos(DOY * 3 * 2 * pi/366)  2.98e-01   7.11e-02    4.20  2.7e-05 ***
ULAGSwd:cos(DOY * 3 * 2 * pi/366)  4.47e-03   6.05e-02    0.07  0.94103
ULAGSww:cos(DOY * 3 * 2 * pi/366)  1.22e-01   6.39e-02    1.91  0.05567 .
ULAGSdd:sin(DOY * 3 * 2 * pi/366)  2.10e-02   3.89e-02    0.54  0.58930
ULAGSdw:sin(DOY * 3 * 2 * pi/366)  6.62e-02   6.86e-02    0.96  0.33484
```

```
ULAGSwd:sin(DOY * 3 * 2 * pi/366) -8.11e-02   6.07e-02   -1.34  0.18184
ULAGSww:sin(DOY * 3 * 2 * pi/366) -2.50e-01   6.09e-02   -4.11  4.0e-05 ***
ULAGSdd:cos(DOY * 4 * 2 * pi/366) -1.44e-01   3.87e-02   -3.72  0.00020 ***
ULAGSdw:cos(DOY * 4 * 2 * pi/366) -2.27e-01   6.08e-02   -3.74  0.00019 ***
ULAGSwd:cos(DOY * 4 * 2 * pi/366) -7.11e-02   5.57e-02   -1.28  0.20180
ULAGSww:cos(DOY * 4 * 2 * pi/366) -1.02e-01   5.51e-02   -1.85  0.06502 .
ULAGSdd:sin(DOY * 4 * 2 * pi/366) -7.75e-02   3.89e-02   -1.99  0.04621 *
ULAGSdw:sin(DOY * 4 * 2 * pi/366) -5.24e-02   6.06e-02   -0.86  0.38767
ULAGSwd:sin(DOY * 4 * 2 * pi/366) -6.20e-02   5.55e-02   -1.12  0.26408
ULAGSww:sin(DOY * 4 * 2 * pi/366)  1.15e-01   5.41e-02    2.13  0.03331 *
---
Signif. codes:  0 âĂŸ***âĂŹ 0.001 âĂŸ**âĂŹ 0.01 âĂŸ*âĂŹ 0.05 âĂŸ.âĂŹ 0.1 âĂŸ âĂŹ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 30758  on 22187  degrees of freedom
Residual deviance: 23627  on 22150  degrees of freedom
  (7550 observations deleted due to missingness)
AIC: 23701

Number of Fisher Scoring iterations: 6
```

We note that the coefficient of "Julian" is significant at the 5% level. Furthermore, looking at the coefficient (4.75e-06) and the mininum and maximum values of Julian, we note that the net change over time is about 0.15. This is in logit space, the net change in probability is about .03, about a 6% change in probability.

We can also add something to the `other_model_string` parameter. Lets look at the interactions of Julian with the lags

```
> fit_object_1b=fit_rainy(zaza_wm,filename="/home/william/Reading/rstudio/Climarkplus/inst/p
> summary(fit_object_1b[[2]])

Call:
glm(formula = fit_string, family = "binomial", data = wms)

Deviance Residuals:
   Min      1Q  Median      3Q     Max
-1.636  -0.916  -0.365   1.033   2.873

Coefficients:
                                   Estimate Std. Error z value Pr(>|z|)
Julian                             6.00e-06   3.33e-06    1.80  0.07175 .
ULAGSdd:cos(DOY * 0 * 2 * pi/366) -1.47e+00   3.02e-02  -48.91  < 2e-16 ***
ULAGSdw:cos(DOY * 0 * 2 * pi/366) -7.32e-01   6.19e-02  -11.82  < 2e-16 ***
ULAGSwd:cos(DOY * 0 * 2 * pi/366) -2.08e-01   4.90e-02   -4.25  2.1e-05 ***
ULAGSww:cos(DOY * 0 * 2 * pi/366)  2.62e-01   5.47e-02    4.78  1.7e-06 ***
```

```
ULAGSdd:cos(DOY * 1 * 2 * pi/366)  1.13e+00   4.58e-02   24.77  < 2e-16 ***
ULAGSdw:cos(DOY * 1 * 2 * pi/366)  6.43e-01   1.06e-01    6.07  1.3e-09 ***
ULAGSwd:cos(DOY * 1 * 2 * pi/366)  3.42e-01   8.05e-02    4.25  2.2e-05 ***
ULAGSww:cos(DOY * 1 * 2 * pi/366)  2.31e-01   9.32e-02    2.48  0.01306 *
ULAGSdd:sin(DOY * 1 * 2 * pi/366)  3.91e-01   3.63e-02   10.76  < 2e-16 ***
ULAGSdw:sin(DOY * 1 * 2 * pi/366)  2.41e-01   5.57e-02    4.32  1.5e-05 ***
ULAGSwd:sin(DOY * 1 * 2 * pi/366)  1.61e-01   5.14e-02    3.12  0.00180 **
ULAGSww:sin(DOY * 1 * 2 * pi/366)  7.41e-02   5.19e-02    1.43  0.15339
ULAGSdd:cos(DOY * 2 * 2 * pi/366) -8.49e-01   4.30e-02  -19.75  < 2e-16 ***
ULAGSdw:cos(DOY * 2 * 2 * pi/366) -6.05e-01   8.81e-02   -6.86  6.8e-12 ***
ULAGSwd:cos(DOY * 2 * 2 * pi/366) -3.07e-01   6.93e-02   -4.43  9.3e-06 ***
ULAGSww:cos(DOY * 2 * 2 * pi/366) -3.12e-01   7.85e-02   -3.98  6.9e-05 ***
ULAGSdd:sin(DOY * 2 * 2 * pi/366) -4.31e-01   3.83e-02  -11.25  < 2e-16 ***
ULAGSdw:sin(DOY * 2 * 2 * pi/366) -3.77e-01   6.84e-02   -5.50  3.7e-08 ***
ULAGSwd:sin(DOY * 2 * 2 * pi/366) -4.10e-01   6.04e-02   -6.78  1.2e-11 ***
ULAGSww:sin(DOY * 2 * 2 * pi/366) -2.44e-01   6.40e-02   -3.81  0.00014 ***
ULAGSdd:cos(DOY * 3 * 2 * pi/366)  3.29e-01   4.12e-02    7.99  1.4e-15 ***
ULAGSdw:cos(DOY * 3 * 2 * pi/366)  2.98e-01   7.11e-02    4.19  2.8e-05 ***
ULAGSwd:cos(DOY * 3 * 2 * pi/366)  4.29e-03   6.05e-02    0.07  0.94347
ULAGSww:cos(DOY * 3 * 2 * pi/366)  1.13e-01   6.41e-02    1.76  0.07817 .
ULAGSdd:sin(DOY * 3 * 2 * pi/366)  2.07e-02   3.90e-02    0.53  0.59566
ULAGSdw:sin(DOY * 3 * 2 * pi/366)  6.73e-02   6.87e-02    0.98  0.32722
ULAGSwd:sin(DOY * 3 * 2 * pi/366) -8.09e-02   6.07e-02   -1.33  0.18313
ULAGSww:sin(DOY * 3 * 2 * pi/366) -2.49e-01   6.09e-02   -4.09  4.3e-05 ***
ULAGSdd:cos(DOY * 4 * 2 * pi/366) -1.44e-01   3.87e-02   -3.71  0.00021 ***
ULAGSdw:cos(DOY * 4 * 2 * pi/366) -2.27e-01   6.08e-02   -3.73  0.00019 ***
ULAGSwd:cos(DOY * 4 * 2 * pi/366) -7.09e-02   5.57e-02   -1.27  0.20273
ULAGSww:cos(DOY * 4 * 2 * pi/366) -9.61e-02   5.52e-02   -1.74  0.08158 .
ULAGSdd:sin(DOY * 4 * 2 * pi/366) -7.72e-02   3.89e-02   -1.98  0.04719 *
ULAGSdw:sin(DOY * 4 * 2 * pi/366) -5.21e-02   6.07e-02   -0.86  0.39048
ULAGSwd:sin(DOY * 4 * 2 * pi/366) -6.19e-02   5.56e-02   -1.11  0.26532
ULAGSww:sin(DOY * 4 * 2 * pi/366)  1.20e-01   5.41e-02    2.22  0.02673 *
ULAGSdw:Julian                     4.45e-06   5.65e-06    0.79  0.43123
ULAGSwd:Julian                     8.40e-07   5.57e-06    0.15  0.88010
ULAGSww:Julian                    -1.01e-05   5.40e-06   -1.87  0.06103 .
---
Signif. codes:  0 âĂŸ***âĂŹ 0.001 âĂŸ**âĂŹ 0.01 âĂŸ*âĂŹ 0.05 âĂŸ.âĂŹ 0.1 âĂŸ âĂŹ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 30758  on 22187  degrees of freedom
Residual deviance: 23621  on 22147  degrees of freedom
  (7550 observations deleted due to missingness)
AIC: 23701

Number of Fisher Scoring iterations: 6
```

There is no significant interaction of Julian with any of the lags. (Note that fit_object_1b is not suitable for making a model)

We can also fit the amounts. Note that we use Gamma regression (we used logistic regression for tt fit_rainy). As well we only fit on days wich are rainy

```
> fit_object_2=fit_amounts(zaza_wm,filename="/home/william/Reading/rstudio/Climarkplus/inst/
> summary(fit_object_2[[2]])

Call:
glm(formula = fit_string, family = "Gamma", data = subdata)

Deviance Residuals:
   Min      1Q  Median      3Q     Max
-2.419  -1.231  -0.480   0.354   4.160

Coefficients:
                           Estimate Std. Error t value Pr(>|t|)
cos(DOY * 0 * 2 * pi/366)  0.117709   0.002399   49.07  < 2e-16 ***
cos(DOY * 1 * 2 * pi/366) -0.000910   0.004038   -0.23    0.822
sin(DOY * 1 * 2 * pi/366) -0.010436   0.002451   -4.26  2.1e-05 ***
cos(DOY * 2 * 2 * pi/366)  0.002683   0.003394    0.79    0.429
sin(DOY * 2 * 2 * pi/366)  0.007182   0.003032    2.37    0.018 *
cos(DOY * 3 * 2 * pi/366)  0.003651   0.002824    1.29    0.196
sin(DOY * 3 * 2 * pi/366)  0.000197   0.002871    0.07    0.945
cos(DOY * 4 * 2 * pi/366) -0.003776   0.002528   -1.49    0.135
sin(DOY * 4 * 2 * pi/366)  0.000227   0.002546    0.09    0.929
---
Signif. codes:  0 âĂŸ***âĂŹ 0.001 âĂŸ**âĂŹ 0.01 âĂŸ*âĂŹ 0.05 âĂŸ.âĂŹ 0.1 âĂŸ âĂŹ 1

(Dispersion parameter for Gamma family taken to be 1.3)

    Null deviance:   NaN  on 7623  degrees of freedom
Residual deviance: 10476  on 7614  degrees of freedom
AIC: 48266

Number of Fisher Scoring iterations: 7
```

Note that as we are using a rain order of 0, there is no interaction of the Fourier coefficients and any lag.

3.2 The model

Central to the package is the `model` data set. There are functions to create models (e.g. from known data).

We can create a model from the fit objects produced above.

```
> zaza_mod=make_model_from_fit_objects(fit_object_1,fit_object_2)
> head(zaza_mod)
```

```
                        info P(w|ww) P(w|dw) P(w|wd) P(w|dd) <rain>
1                <order> = 2    0.55    0.34    0.43    0.26    8.4
2           <rain_order> = 0    0.55    0.34    0.43    0.26    8.4
3 <shape> = 0.856151042431432    0.54    0.34    0.42    0.26    8.4
4                               0.54    0.34    0.42    0.26    8.3
5                               0.54    0.34    0.41    0.26    8.3
6                               0.54    0.34    0.41    0.25    8.3
```

[Note that the model can and probably will change] The data set has 366 rows, only the first 6 are shown. The first column contains some information about the model. The <`rain`> column contains the mean of the Gamma distribution of the amount of rate. The shape is constant and is given in the firt column. Columns of the form `P(w|lag)` are the Markov probabilities of rain.

3.3 Simple Fitting For Model Choice

We count for each day of year:the number of "w" days following two "d" days; the number of "w" days following a "d" then a "w" day etc. Dividing by e.g. the number of times we have two consecutive "d" days give the estimated probability. The function `make_all_probs` does this, for all lags up to order (default 2). As well, for each day of year, we determine the mean and standard deviation of the rain of a "w" day both unconditional, and conditioned on lags up to `max_mean_rain_order` (default 1)

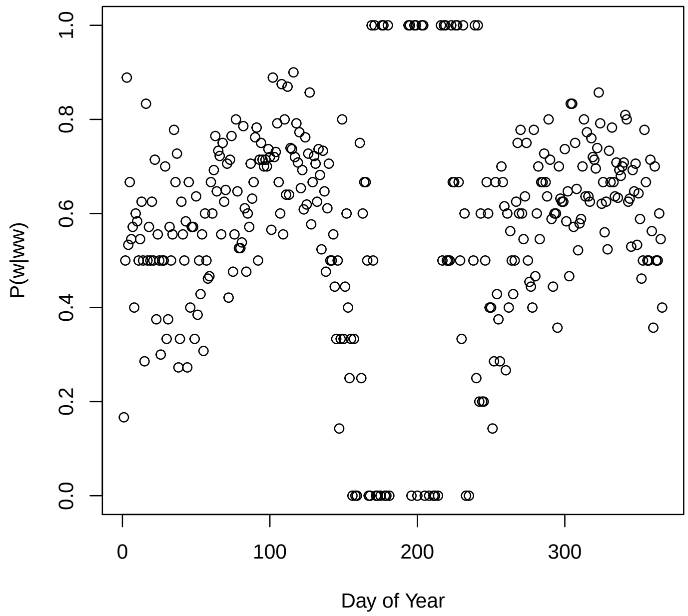
```
>   zaza_pbs=make_all_probs(zaza_wm)
```

```
> head(zaza_pbs)
```

```
  P(w) <rain> sd(rain) # days # wet days P(w|w) P(w|d) #w #d P(w|ww) P(w|dw)
1 0.29    9.4     16.0     62         18   0.30   0.27 20 41    0.17    0.38
2 0.34    5.4      7.9     62         21   0.50   0.27 18 44    0.50    0.43
3 0.52    8.9     13.0     62         32   0.71   0.41 21 41    0.89    0.56
4 0.27    9.0      8.2     62         17   0.38   0.17 32 30    0.53    0.33
5 0.39    6.6      7.1     62         24   0.65   0.29 17 45    0.67    0.35
6 0.40    5.4      7.9     62         25   0.58   0.29 24 38    0.55    0.33
  P(w|wd) P(w|dd) #ww #dw #wd #dd <(r|w)> <(r|d)> sd(r|w) sd(r|d) #rw #rd
1    0.50    0.20  12  16   8  25     2.0    12.7     2.4    19.5   6  11
2    0.55    0.20   6  14  11  30     4.3     6.2     5.3     9.6   9  12
3    0.58    0.38   9   9  12  32     7.7    10.1    13.1    13.2  15  17
4    0.24    0.12  15   6  17  24     9.7     7.4     9.3     4.8  12   5
5    0.60    0.24  12  20   5  25     3.8     9.0     3.7     8.5  11  13
6    0.62    0.28  11   6  13  32     3.2     8.2     3.8    10.6  14  11
```

As we can see from a plot, the probabilities are all over the map. In this form they will not help use to choose a good model

```
>   plot(zaza_pbs[,"P(w|ww)"],xlab="Day of Year",ylab="P(w|ww)")
```

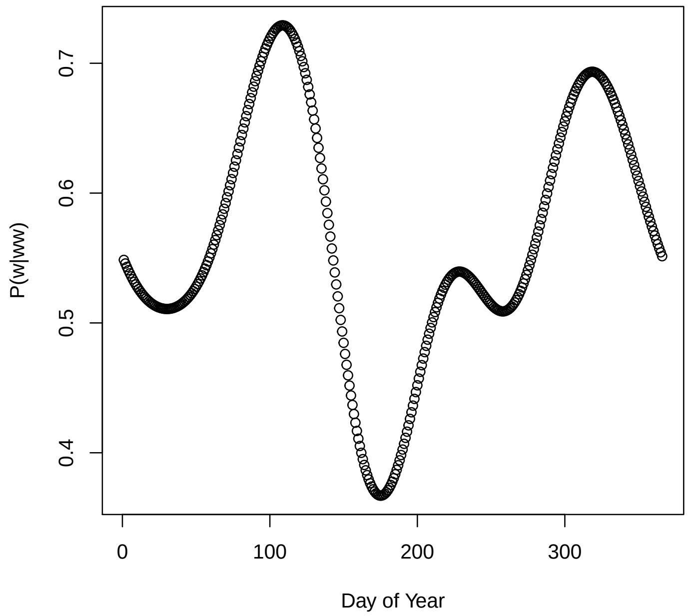


We need to smooth the probabilities. We fit a Fourier series (this has the advantage that we can make things periodic with period 1 year). The order of the fit can be determined before hand or determined interactively or automatically. The function used is `make_approx_model_pl`.

```
>   zaza_approx = make_approx_model_pl(zaza_pbs,"/home/william/Reading/rstudio/climate/trunk
```

When fitting to obtain the approximate model , we weight each estimate by the number of observations used to obtain the estimate. So estimates based on only a single day (e.g. two consecutive ”w” days during the dry season) are not given much weight.

```
>   plot(zaza_approx[,"P(w|ww)"],xlab="Day of Year",ylab="P(w|ww)")
```



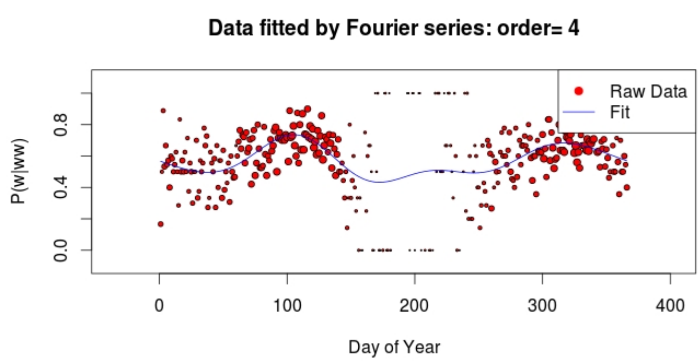
And things look much smoother.

3.4 Interactive

We can also do the fitting interactively. At the console, enter the command

```
>   zaza_mod = make_model_general(zaza_pbs,inter=TRUE)
```

We see a graph



The red circles represent data points. The are of each circle is proportional to the weighting the data point has in the fit. The smallest circles are data point calculated from a single line of the raw data (so for probabilities are 0 or 1).

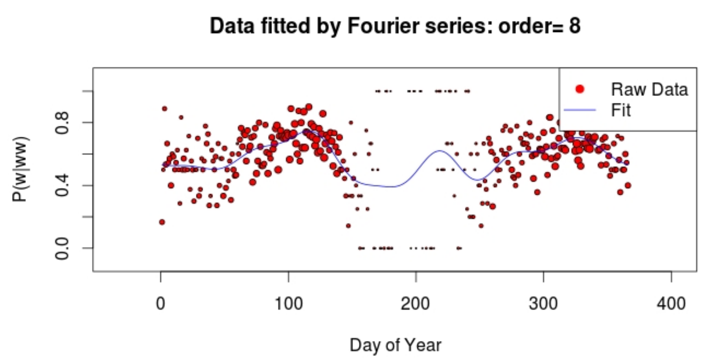
The blue line represents the fitted curve. It will change when we change the order of the Fourier fit.

On the console you will see:

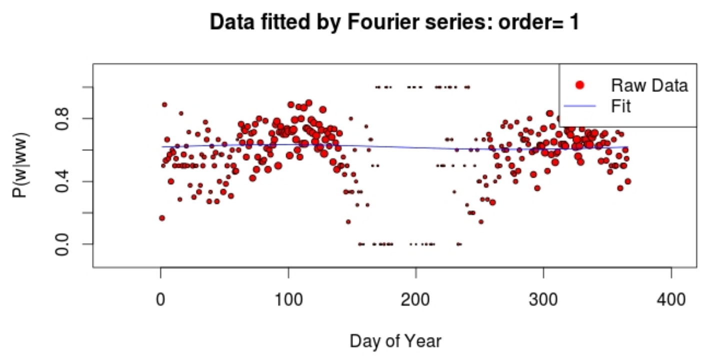
```
enter

a:  use this fit
b:  use previous order
f:  add one to order
k:  set order to k
```

To enter a value you must type the value into the console and then press return. Entering a number changes the fit order to that number. Try entering 8. You get:



The fit does not look that much better, especially when considering how much more wavy the line is. Try entering 1. You get:



Clearly we are now underfitting. Enter 4, to get back to the first graph, then enter `a` to accept this fit. You will then have 5 more graphs to fit. Play around with the `a,b` and `f` keys or enter numbers. Repeatedly pressing `a` will use the default.

3.5 Synthetic Data

Once we have a model, we can use it to synthesize data. The command is

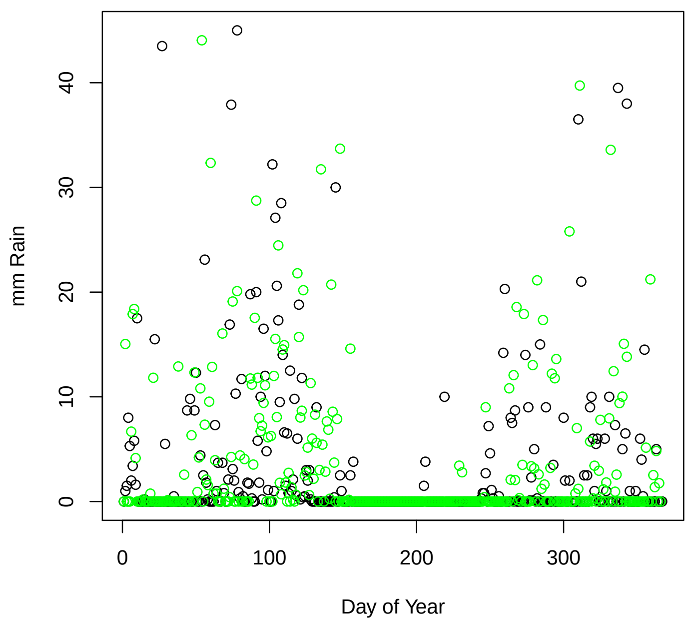
```
>   #load("zaza_synth")
>   zaza_synth=synth_data_set_mod(zaza_mod,num_years=83)
>   head(zaza_synth)
```

```
  Station       Date Rain DOY
1   synth 1970-01-01  0.0   1
2   synth 1970-01-02 15.0   2
3   synth 1970-01-03  0.0   3
4   synth 1970-01-04  0.0   4
5   synth 1970-01-05  0.0   5
6   synth 1970-01-06  6.7   6
```

The data will start from year 1970 by default. The number of years produced is limited by your patience (and by the address space of your machine, 200,000 years for a 32 bit machine; you need to have a **LOT** of patience to reach the limit if you have a 64 bit machine). As a rule of thumb 1000 years takes about a minute (your mileage will vary).

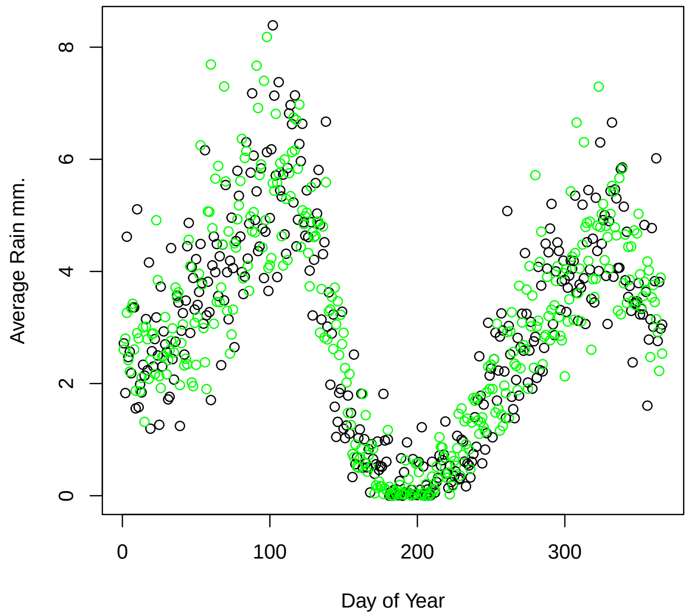
Let's compare the synthetic and the real data. Clearly we cannot expect day by day comparisons to be equal, indeed, compare the rain in 1931, to the first year of rain in the synthetic data.

```
> plot(zaza$Rain[93:459],xlab="Day of Year",ylab="mm Rain")
> points(zaza_synth$Rain[1:366],col="green")
```



The exact values are different, but the pattern is similar. We now take advantage of the r function `x_split=split(x,y)` where x and y are in the same dataframe. What this gives is a list indexed by the values of y. Each element contains every value of x with the given value of y. So zaza_split=split(zaza_doy$Rain, zaza_doy$DOY) is a list of 366 vectors.
tt zaza_split[[”120”]] is a vector of all rainfall on Day of Year 120. So `mean(zaza_split[["120"]],na.rm=TRUE)` is the mean rainfall on day 120 (the last arg means ignore NA’s). We take further advantage by using the r function `sapply(list, function,args)` which applies function to every element of list, passing args to function. So `sapply(zaza_split,mean,na.rm=TRUE)` is a vector of the average rainfalls. Comparing

```
>   zaza_split=split(zaza_doy$Rain,zaza_doy$DOY)
> synth_split=split(zaza_synth$Rain,zaza_synth$DOY)
> plot(sapply(zaza_split,mean,na.rm=TRUE),xlab="Day of Year",
+      ylab="Average Rain mm.")
> points(sapply(synth_split,mean,na.rm=TRUE),col="green")
```



The fit looks good on the mean, but we note that the variablity for the simulated data seems a bit lower that that of the real data.

3.6 Yearly Stats

If we split our real or synthetic data set by `mod_year` then we get a list of datasets, each covering a specific year. We can then find things such a the average spell length (over years), the maximum dry spell for each year etc. There is a simple function to split by `mod_year` adding the year it if it does not exist. The function `add_spell_info` calculates and adds the length of wet and dry spells. This should be applied before splitting if spells should cross year boundaries. Note that we need to add the Markov stuff before adding spell info.

```
>   synth_wm= add_markov(zaza_synth)
> synth_spell= add_spell_info(synth_wm)
> zaza_spell=add_spell_info(zaza_wm)
> synth_split=split_by_year(synth_spell,year_begins_in_july=TRUE)
> zaza_split=split_by_year(zaza_spell,year_begins_in_july=TRUE)
> length(synth_split)
```

```
[1] 84
```

```
> head(synth_split[["1981"]])
```

```
     Station       Date Rain DOY wet_or_dry lag_1 lag_2 first_DOY spell_length
3835   synth 1980-07-01    0 183          d     d    dd       165           57
3836   synth 1980-07-02    0 184          d     d    dd       165           57
3837   synth 1980-07-03    0 185          d     d    dd       165           57
3838   synth 1980-07-04    0 186          d     d    dd       165           57
3839   synth 1980-07-05    0 187          d     d    dd       165           57
3840   synth 1980-07-06    0 188          d     d    dd       165           57
     mod_year month day
3835     1981     7   1
3836     1981     7   2
3837     1981     7   3
3838     1981     7   4
3839     1981     7   5
3840     1981     7   6
```

```
> nrow(synth_split[["1981"]])
```

```
[1] 365
```

```
> length(zaza_split)
```

```
[1] 82
```

```
> head(zaza_split[["1951"]])
```

```
     Station       Date Rain DOY wet_or_dry lag_1 lag_2 Julian first_DOY
7214    Zaza 1950-07-01    0 183          d     d    dd  -7124       151
7215    Zaza 1950-07-02    0 184          d     d    dd  -7123       151
7216    Zaza 1950-07-03    0 185          d     d    dd  -7122       151
7217    Zaza 1950-07-04    0 186          d     d    dd  -7121       151
7218    Zaza 1950-07-05    0 187          d     d    dd  -7120       151
7219    Zaza 1950-07-06    0 188          d     d    dd  -7119       151
     spell_length mod_year month day
7214           89     1951     7   1
7215           89     1951     7   2
7216           89     1951     7   3
7217           89     1951     7   4
7218           89     1951     7   5
7219           89     1951     7   6
```

```
> nrow(zaza_split[["1951"]])
```

```
[1] 365
```

So `synth_split` consists of 84 data sets (there is a partial year at the beginning and end) and `zaza_split` consists of 82 data sets (there are no partial years.)

3.6.1 Spell Lengths

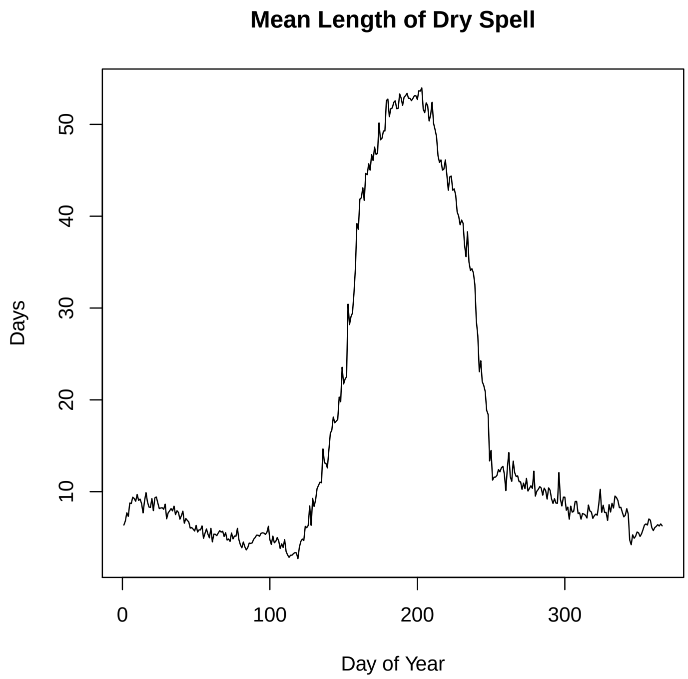
We can look at the average dry spell length. For this we use the data that has not been spit into years. First we take only dry spells. To do this, take only rows of `zaza_spell` that correspond to a dry day.

```
>   zaza_dry=zaza_spell[(zaza_spell$wet_or_dry=="d"),]
> head(zaza_dry)

  Station       Date Rain DOY wet_or_dry lag_1 lag_2 Julian first_DOY
1    Zaza 1930-10-01    0 275          d  <NA>  <NA> -14337       275
4    Zaza 1930-10-04    0 278          d     w    ww -14334       278
5    Zaza 1930-10-05    0 279          d     d    dw -14333       278
7    Zaza 1930-10-07    0 281          d     w    wd -14331       281
8    Zaza 1930-10-08    0 282          d     d    dw -14330       281
9    Zaza 1930-10-09    0 283          d     d    dd -14329       281
  spell_length
1            1
4            2
5            2
7            4
8            4
9            4
```

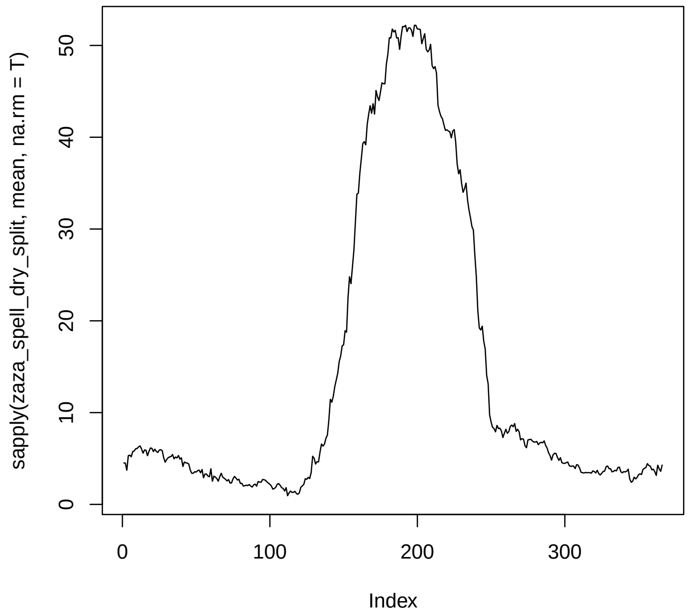
Now we split the spell lengths by day of year, then take the mean (ignoring NA's). Plot this

```
> dry_spell_split=split(zaza_dry$spell_length,zaza_dry$DOY)
> plot(sapply(dry_spell_split,mean,rm.na=T),type="l",xlab="Day of Year",
+                     ylab="Days",main="Mean Length of Dry Spell")
```



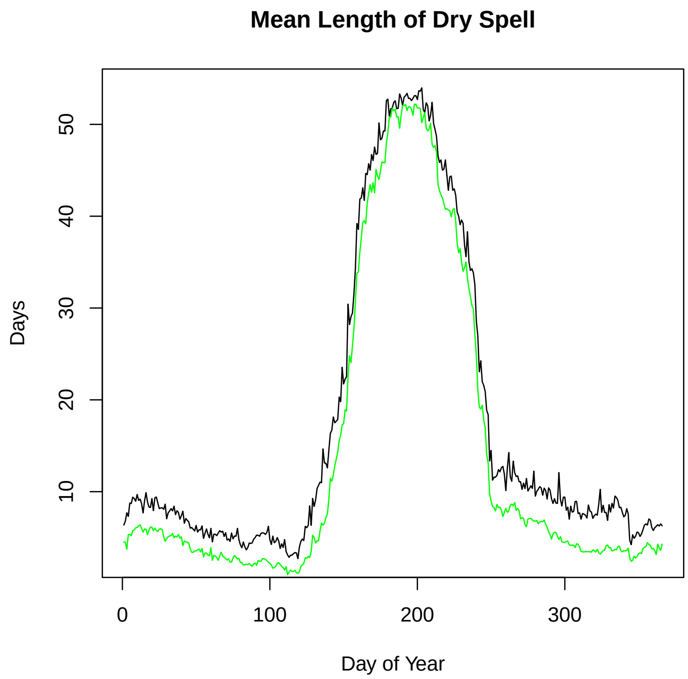
Note this is the average length of a dry spell, given that there is a dry spell. This may or may not be what you want. If you want the mean dry spell length, taking the dry spell length of a wet day to be 0 then this can be done by adding another column.

```
> zaza_spell$dry_spell_length=zaza_spell$spell_length
> zaza_spell$dry_spell_length[zaza_spell$wet_or_dry == "w"]=0
> zaza_spell$dry_spell_length[is.na(zaza_spell$wet_or_dry)]=NA
> zaza_spell_dry_split=split(zaza_spell$dry_spell_length,zaza_spell$DOY)
> plot(sapply(zaza_spell_dry_split,mean,na.rm=T),type="l")
```



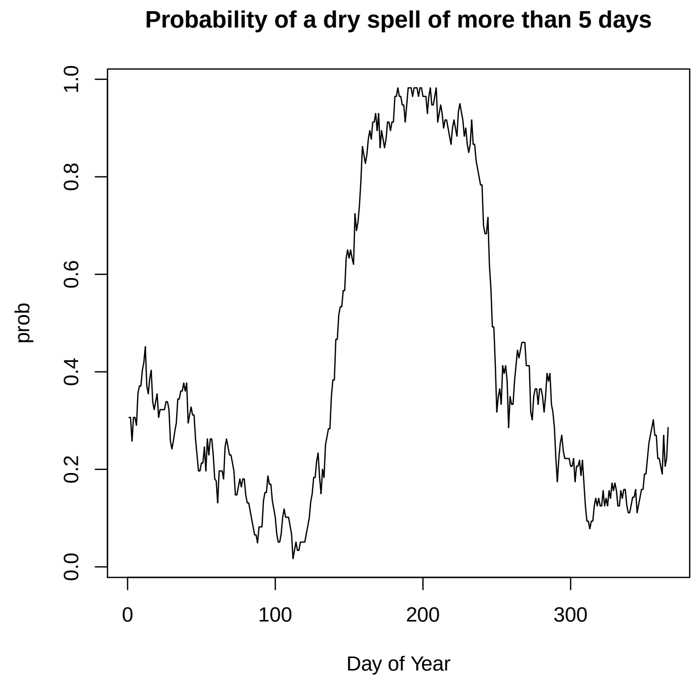
This looks similar. Plot them together for comparison. (note the use of the `lines` function which adds lines to a graph)

```
> plot(sapply(dry_spell_split,mean,rm.na=T),type="l",xlab="Day of Year",
+                      ylab="Days",main="Mean Length of Dry Spell")
>                      lines(sapply(zaza_spell_dry_split,mean,na.rm=T),col="green")
```



Assume that we know our crop can withstand dry spells of up to 5 days. Then we are interested in the probability of a dry spell of more than 5 days.

```
> zaza_dry_spell_over_5=zaza_spell[zaza_spell$wet_or_dry=="d" & zaza_spell$spell_length > 5,
> zaza_dry_spell_over_5_split=split(zaza_dry_spell_over_5,zaza_dry_spell_over_5$DOY)
>  num_of_dry_spells_over_5=sapply(zaza_dry_spell_over_5_split,nrow)
> zaza_spell_good=zaza_spell[!is.na(zaza_spell$wet_or_dry),]
> zaza_spell_good_split=split(zaza_spell_good,zaza_spell_good$DOY)
> num_good_days=sapply(zaza_spell_good_split,nrow)
> prob_of_dry_spell_over_5=num_of_dry_spells_over_5/num_good_days
> plot(prob_of_dry_spell_over_5,type="l",xlab="Day of Year",ylab="prob",
+ main="Probability of a dry spell of more than 5 days")
```



Naturally, all the above can be done with synthetic data as well. (Indeed, this is the whole point. Get your model from 20 years of real data, but get your probabilities from 100 or 1000 years of synthesized data. In some cases, the actual value can be obtained from the model by clever analysis (e.g. Markov model's and spell probabilities). However, R is cheap and fast; Statisticians capable of doing the analyses are expensive and slow. And what happens when you modify your model?)

3.6.2 First Day of Growing Season

Two common definitions of the "First Day of the Growing Season" are:

1. The first day of a period of n rainy days on which there is more than k millimetres of rain in total
2. As (1.) but additionally, there is no dry spell of k days in the next j days

The second definition may be defined as the time of successful planting (young seedlings may be injured or killed by a dry spell of k days) The function `fdgs` can calculate either of these for a year of data. So if we apply it to every year of data to find the distribution of the "First Day of the Growing Season" (we start our search in July if the growing season occurs in "winter"). So:

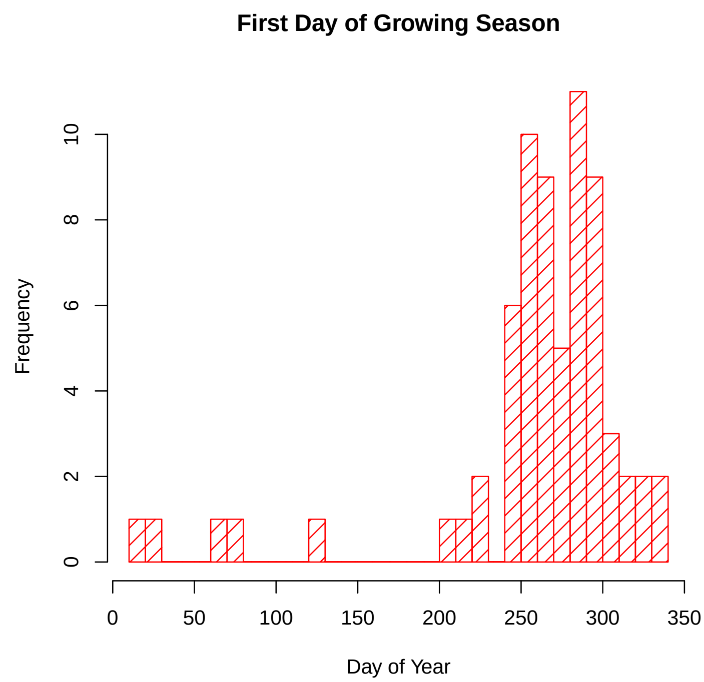
```
> first_days=sapply(zaza_split,fdgs)
> first_days_succ=sapply(zaza_split,fdgs,type=2)
```

and we can do the same thing with synthetic data

```
> first_days_synth=sapply(synth_split,fdgs)
> first_days_succ_synth=sapply(synth_split,fdgs,type=2)
```

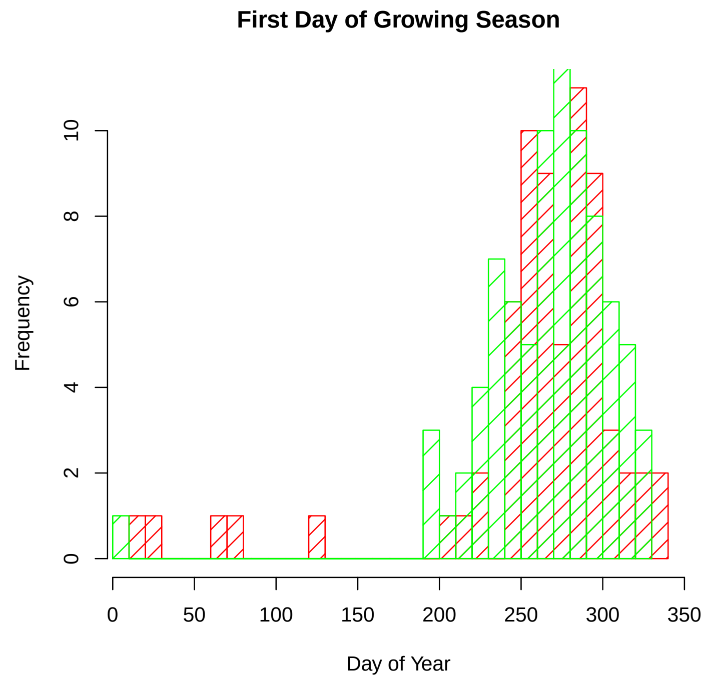
In this case a histogram is best for viewing a comparison.

```
>   hist(first_days,breaks=25,col="red",density=10,
+         xlab="Day of Year",main="First Day of Growing Season")
```



We can compare this with the calculations from the synthetic data (note for adding information to a histogram, use the `add=T`) parameter.

```
>   hist(first_days,breaks=25,col="red",density=10,
+         xlab="Day of Year",main="First Day of Growing Season")
> hist(first_days_synth,breaks=25,col="green",density=5,add=T)
```



Again, this is similar but not identical.

Another question we might ask is: ”In how many of the 82 years was planting by definition 1. successful?” To answer this we can compare, `first_days` and `first_days_succ`. We need to be careful to allow for NA’s and for modding by 366.

```
> diff=(first_days_succ-first_days) %% 366
> diff_no_na=diff[!is.na(diff)]
> diff_good=diff_no_na[diff_no_na==0]
> length(diff_good)/length(diff_no_na)
```

```
[1] 0.51
```

This suggests that the default values do not work well for this site. Perhaps a more drought tolerant crop would work better, or wait for more rain before saying the growing season has started.