Modeling field trials using RAP

Bart-Jan van Rossum

2019-03-04

The RAP Package

The RAP package is developed as an easy to use package for analyzing data of plant breeding experiments with many options for plotting and reporting the results of the analyses.

The package has three main components:

- Modeling trial data for single trials and extracting results
- Genotype by Environment (GxE) analysis
- QTL analysis

This vignette deals with the modeling part of the package and describes in detail how to prepare data for analysis, perform analysis using different modeling engines and extract the results from the models.

Data preparation

The first step when modeling field trial data or performing a GxE analysis with the RAP package is creating an object of class `TD`. This object is used by the RAP package for performing most analyses.

Creating a `TD` object

A `TD` object can be created from a `data.frame` with the function `createTD`. This function does a number of things:

- Rename columns to default column names used by the functions in the RAP package. For example, the column in the data containing variety/accession/genotype is renamed to "genotype". Original column names are stored as an attribute of the `TD` object.
- Convert column types to the default column types. For example, the column "genotype" is converted to a factor and "rowCoord" to a numeric column.
- Split the data into separate data.frames by trial. A `TD` object is a `list` of `data.frames` where each `data.frame` contains the data for a single trial. If there is only one trial or no column trial is defined, the output will be a `list` with only one item.
- Add meta data to the separate `data.frames` in the `TD` object. The meta data consists of location, date of the experiment, longitude, latitude, trial design, plot width and plot length. None of these is strictly neccessary for any analysis but the meta data is used for plotting field layouts, plotting trials on a map and naming plots. Meta data for a single trial can simply be added when creating the `TD` object using the parameters in `createTD`. However if the data consists of multiple trials with different meta data per trial it is more convenient to first create a `TD` object without adding meta data. This will still add a location based on the name of the trial. After creating the `TD` object, the meta data can then be extracted using `getMeta`. This function creates a `data.frame` with the meta data per trial. The content of this `data.frame` can then be modified and added back to the `TD` object using `setMeta`.

After creating a `TD` object, data for new trials can be added to it using `addTD`. This function works in exactly the same way as `createTD` except that it adds data to an existing `TD` object instead of creating a new one.

Dropping one or more trials from a TD object can be done using the function dropTD specifying the trials to be dropped in the parameter rmTrials.

Example

Field data from an experiment with wheat in Chili described in detail by Lado et al. (2013) will be used as an example throughout this vignette. The experiment was performed in two locations in Chili, Santa Rosa and Cauquenes, with two different drought regimes in 2011 and 2012 for the Santa Rosa and one trial in 2012 for Cauquenes. For 384 genotypes four traits were measured in 2011, but in 2012 only grain yield (GY) was measured and the examples will mainly focus on this trait.

For the example first a TD object is created for the first location and the data for the second location is then added later on. In practice all this could be done in one go.

```
## Create a TD object containing the data from Santa Rosa.
data("wheatChl")
wheatTD <- createTD(data = wheatChl[wheatChl$trial != "C_SWS_12", ],
                    genotype = "trt", repId = "rep", subBlock = "bl",
                    rowCoord = "row", colCoord = "col")
```

The TD object just created is a list with four items, one for each trial (combination of location, drought regime and year) in the original data.frame. The column "trt" in the original data is renamed to "genotype" and converted to a factor. The columns "rep" and "bl" are renamed and converted likewise. The columns "row" and "col" are renamed to "rowCoord" and "colCoord" respectively. Simultaneously two columns "rowId" and "colId" are created containing the same information converted to a factor. This seemingly duplicate information is needed for spatial analysis. It is possible to define different columns as "rowId" and "colId" than the ones used as "rowCoord" and "colCoord". The information about which columns have been renamed when creating a TD object is stored as an attribute of each individual data.frame in the object.

Meta data

The meta data will be a data.frame with four rows, one for each item in wheatTD.

```
## Extract meta data from the TD object.
(wheatMeta <- getMeta(TD = wheatTD))
#>           trLocation trDate trDesign trLat trLong trPlWidth trPlLength
#> SR_FI_11    SR_FI_11   <NA>       NA    NA     NA         NA          NA
#> SR_FI_12    SR_FI_12   <NA>       NA    NA     NA         NA          NA
#> SR_MWS_11  SR_MWS_11   <NA>       NA    NA     NA         NA          NA
#> SR_MWS_12  SR_MWS_12   <NA>       NA    NA     NA         NA          NA
```

After extracting the meta data, it can be modified and then added back to the original TD object.

```
## Fill in meta data and add back to the TD object.
wheatMeta$trLocation <- "Santa Rosa"
wheatMeta$trDate <- as.Date(rep(c("310811", "310812"), times = 2), "%d%m%y")
wheatMeta$trLat <- -36.32
wheatMeta$trLong <- -71.55
wheatMeta$trPlWidth = 2
wheatMeta$trPlLength = 1
wheatTD <- setMeta(TD = wheatTD, meta = wheatMeta)
```

Add extra data to a TD object

To add the data for the final trial to the TD object the function addTD can be used. Since now only one new trial is added, it makes sense to immediately add the meta data for this trial as well, using the appropriate parameters in addTD.

```
## Add the data for Cauquenes to the TD object.
wheatTD <- addTD(TD = wheatTD, data = wheatChl[wheatChl$trial == "C_SWS_12", ],
                 genotype = "trt", repId = "rep", subBlock = "bl",
                 rowCoord = "row", colCoord = "col", trLocation = "Cauquenes",
                 trDate = as.Date("070912", "%d%m%y"), trLat = -35.58,
                 trLong = -72.17, trPlWidth = 2, trPlLength = 1)
## Inspect the meta data after the extra trial was added.
getMeta(TD = wheatTD)
#>           trLocation     trDate trDesign  trLat trLong trPlWidth trPlLength
#> SR_FI_11  Santa Rosa 2011-08-31       NA -36.32 -71.55         2          1
#> SR_FI_12  Santa Rosa 2012-08-31       NA -36.32 -71.55         2          1
#> SR_MWS_11 Santa Rosa 2011-08-31       NA -36.32 -71.55         2          1
#> SR_MWS_12 Santa Rosa 2012-08-31       NA -36.32 -71.55         2          1
#> C_SWS_12   Cauquenes 2012-09-07       NA -35.58 -72.17         2          1
```

Summarizing a TD object

The summary function can be used to get an idea of the content of the data in the TD object. Multiple traits may be summarized at once but for clarity here a summary is only made for GY.

```
## Create a summary for grain yield in SR_FI_11.
summary(wheatTD, trial = "SR_FI_11", traits = "GY")
#>
#> Summary statistics for GY in SR_FI_11
#>
#>                                GY
#> Number of observations         794
#> Number of missing values         6
#> Mean                      8015.60
#> Median                    8021.00
#> Min                       2239.50
#> Max                      12992.00
#> Lower quartile            6728.50
#> Upper quartile            9321.75
#> Variance                3080361.436
```

Using the default options nine summary statistics are printed, but many more are available. These can be accessed using the parameter what in the summary function. For a full list of available statistics, use help(summary.TD). It is also possible to output all statistics using what = "all".

It is possible to summarize the data in a TD object for different groups. This can be done using the parameter groupBy. It will display three main summary statistics per group. Again, more statistics can be displayed using the parameter what.

```
## Create a summary for the two replicates in SR_FI_11.
summary(wheatTD, trial = "SR_FI_11", traits = "GY", groupBy = "repId")
#>
#> Summary statistics for GY in SR_FI_11 grouped by repId
#>
```

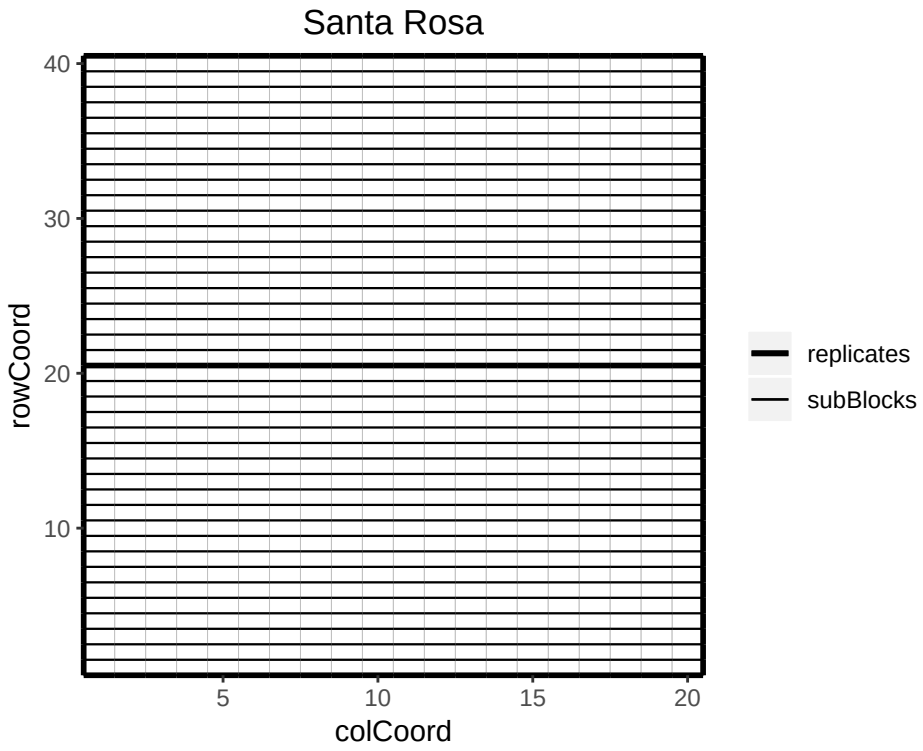
```
#>                                1        2
#> Number of observations       397      397
#> Mean                      7966.98  8064.22
#> Standard deviation        1788.353 1722.078
```

Plotting a TD object

Several plots can be made to further investigate the contents of a TD object.

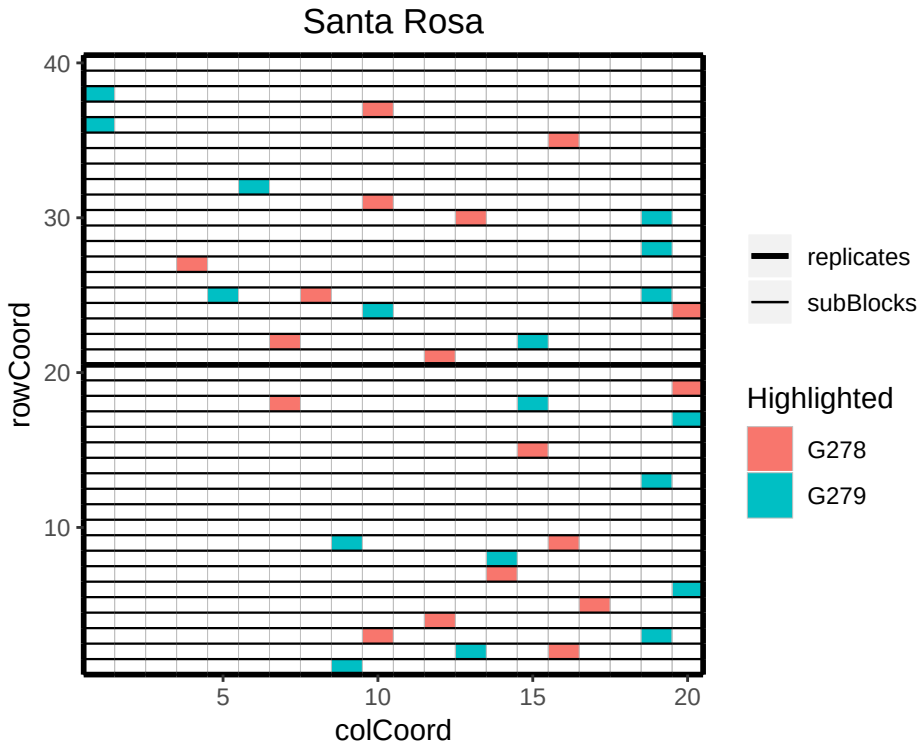
The default plot creates plots for the layout of all trials in the TD object. This can be restricted to selected trials using the trials parameter. The width and length of the plot are derived from "trPlWidth" and "trPlLength" in the meta data if these are available.

```
plot(wheatTD, trials = "SR_FI_11")
```



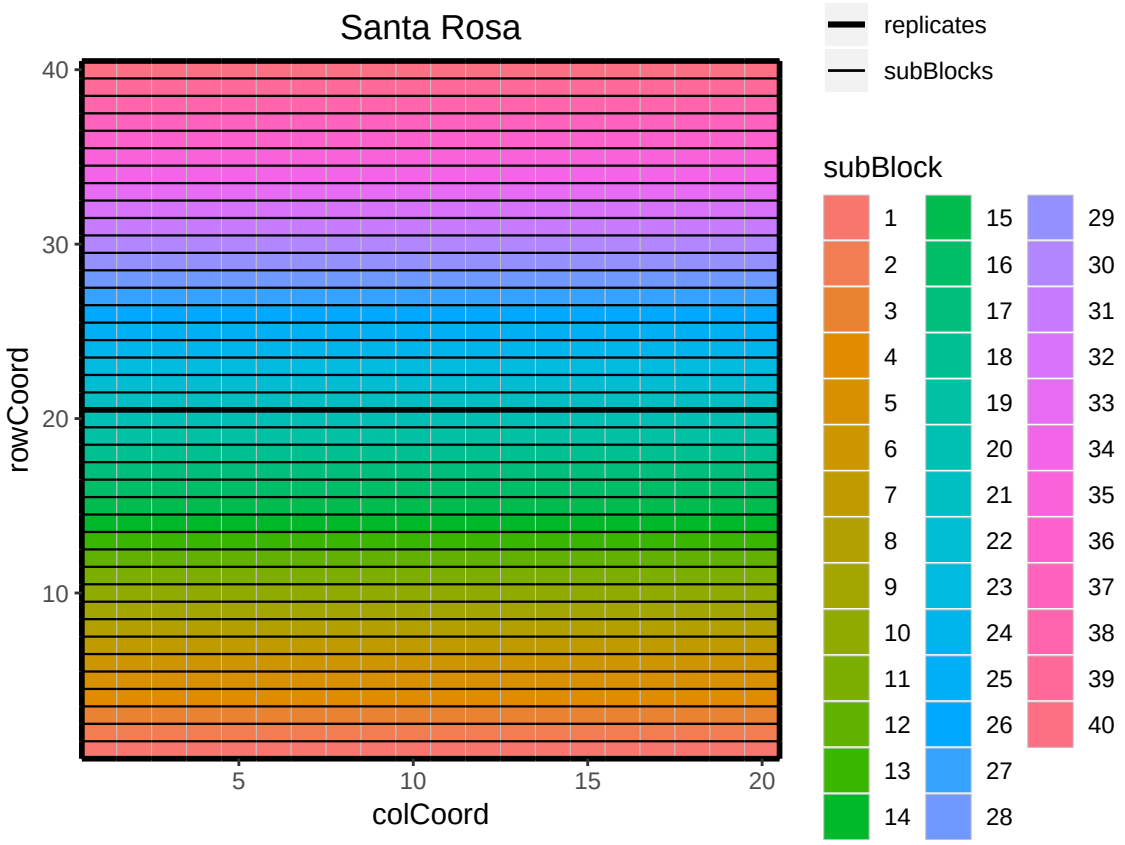
This plot can be extended by highlighting interesting genotypes in the layout.

```
## Plot the layout for SR_FI_11 with genotypes G278 and G279 highlighted.
plot(wheatTD, trials = "SR_FI_11", highlight = c("G278", "G279"))
```



It is also possible to color the subBlocks within the field.

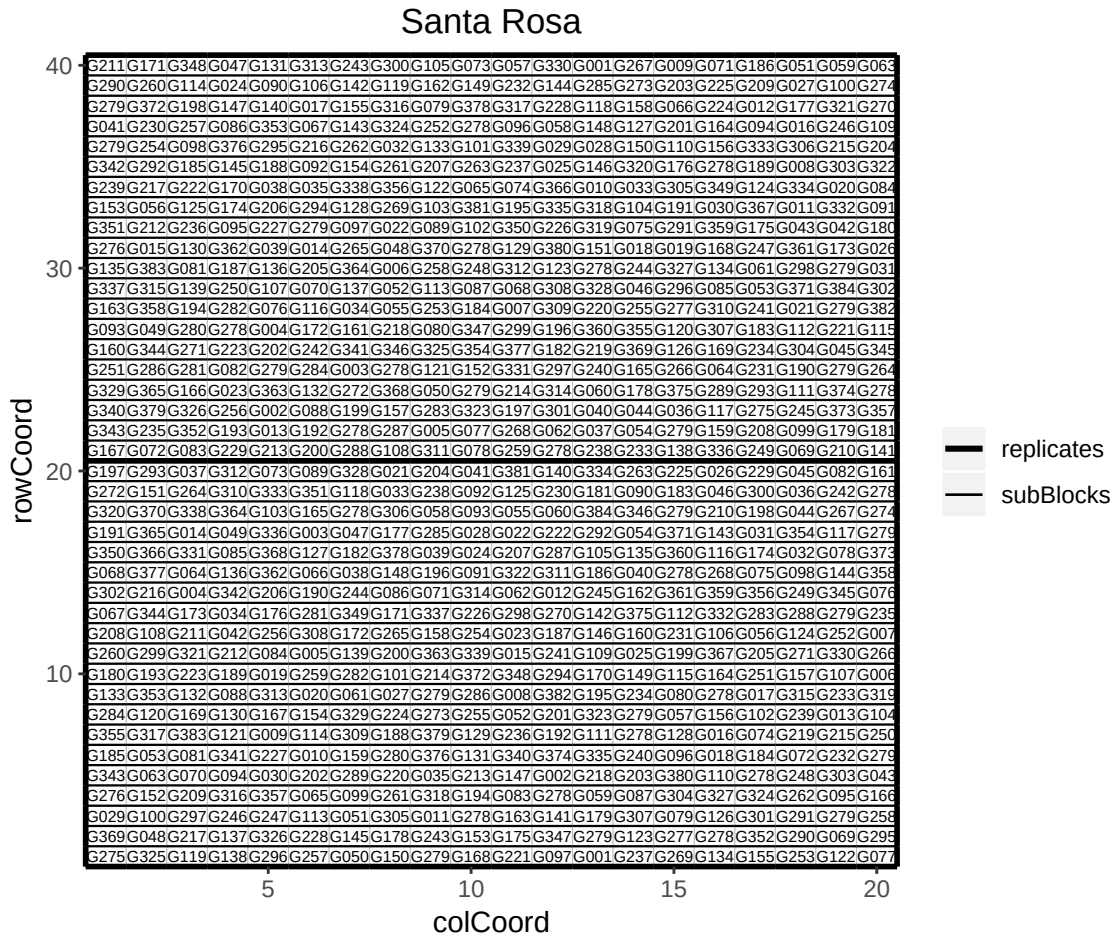
```
## Plot the layout for SR_FI_11, color subBlocks.
plot(wheatTD, trials = "SR_FI_11", colorSubBlock = TRUE)
```



Highlighting genotypes and coloring subBlocks cannot be done simultaneously. If both options are specified, only highlighting is done.

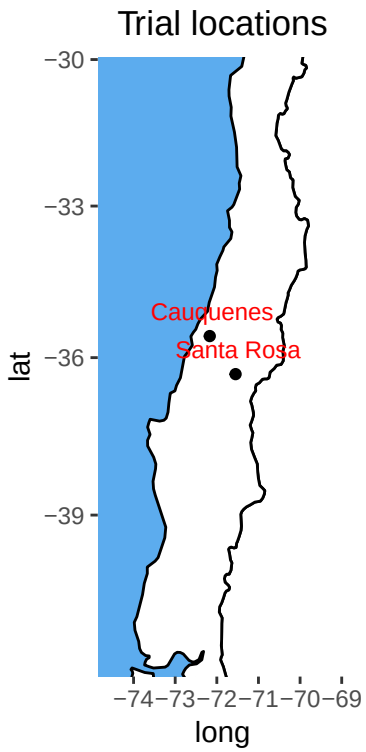
Finally it is possible to add the names of the genotypes to the layout.

```
## Plot the layout for SR_FI_11, color subBlocks.
plot(wheatTD, trials = "SR_FI_11", showGeno = TRUE)
```



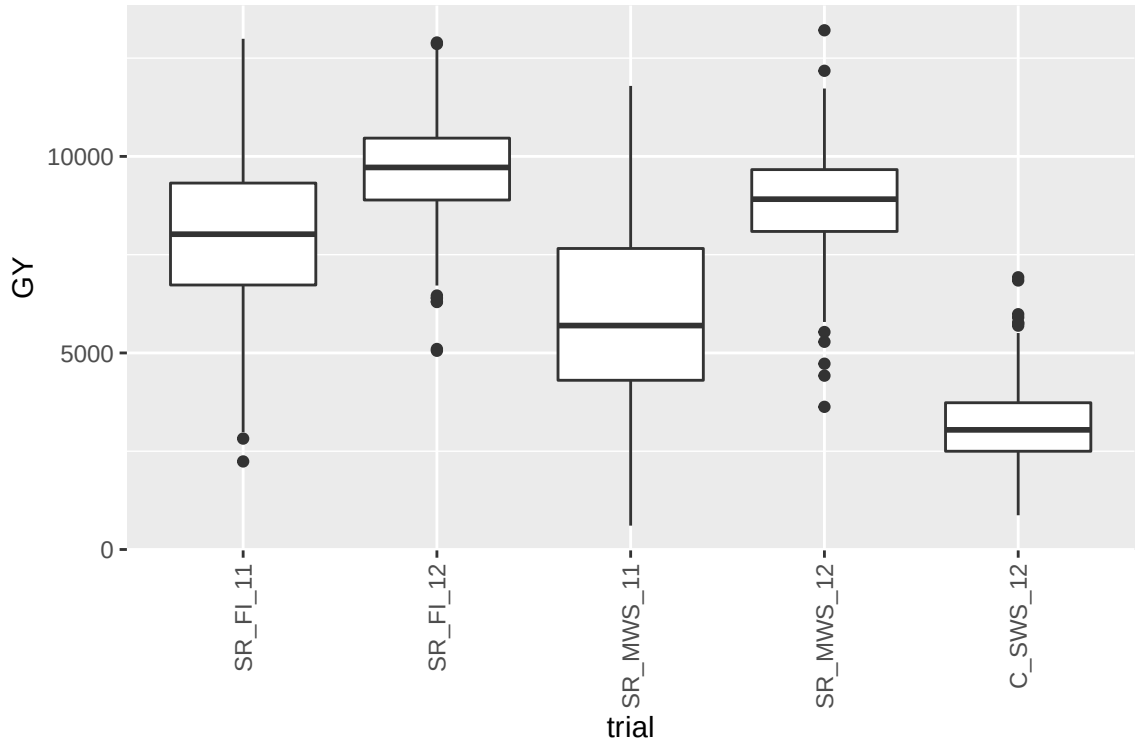
A second type of plot displays the trial locations on a map. This plot is made based on trLat and trLong in the meta data. If latitude or longitude is not available for a certain location, then this location is not plotted. If the locations are very close together, the resulting map can become quite small. The parameters `minLatRange` and `minLongRange` can be used to extend the minimum range of latitude and longitude respectively to address this issue.

```
## Plot the locations of the trials on a map.
plot(wheatTD, plotType = "map")
```



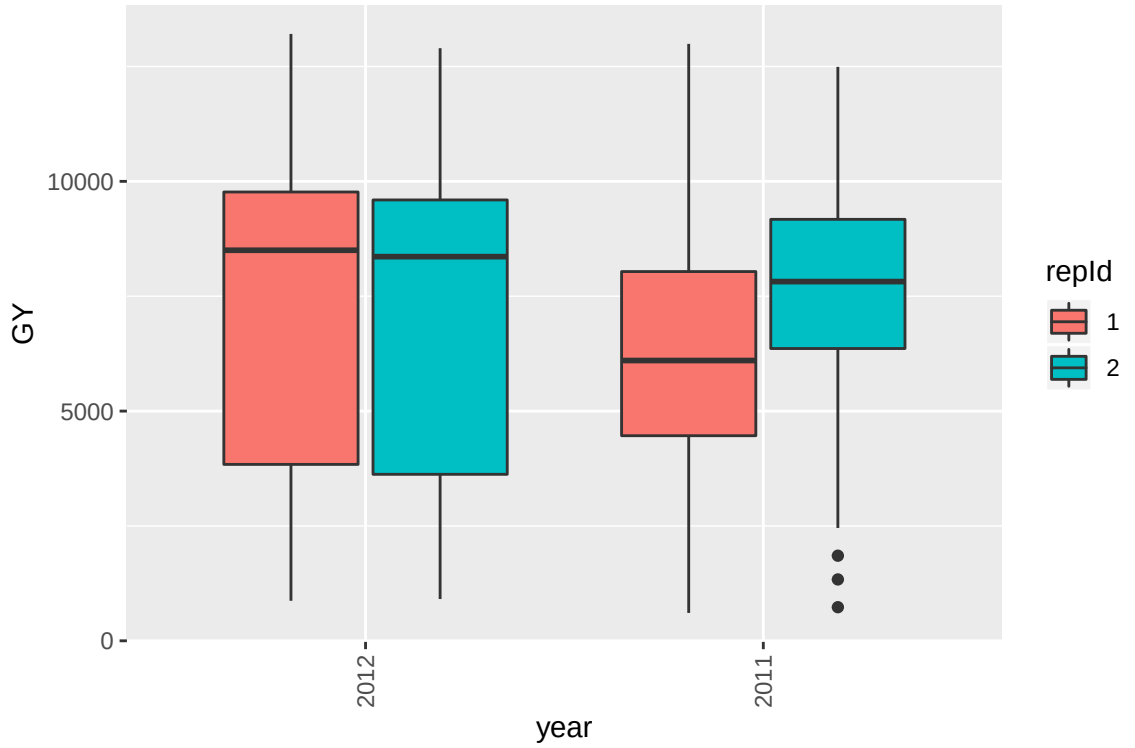
Boxplots can be made to get an idea of the contents of the data in the `TD` object. By default a box is plotted per trial in the data for the specified traits. Boxplots for multiple traits can be made at once.

```
## Create a boxplot for grain yield.
plot(wheatTD, plotType = "box", traits = "GY")
```



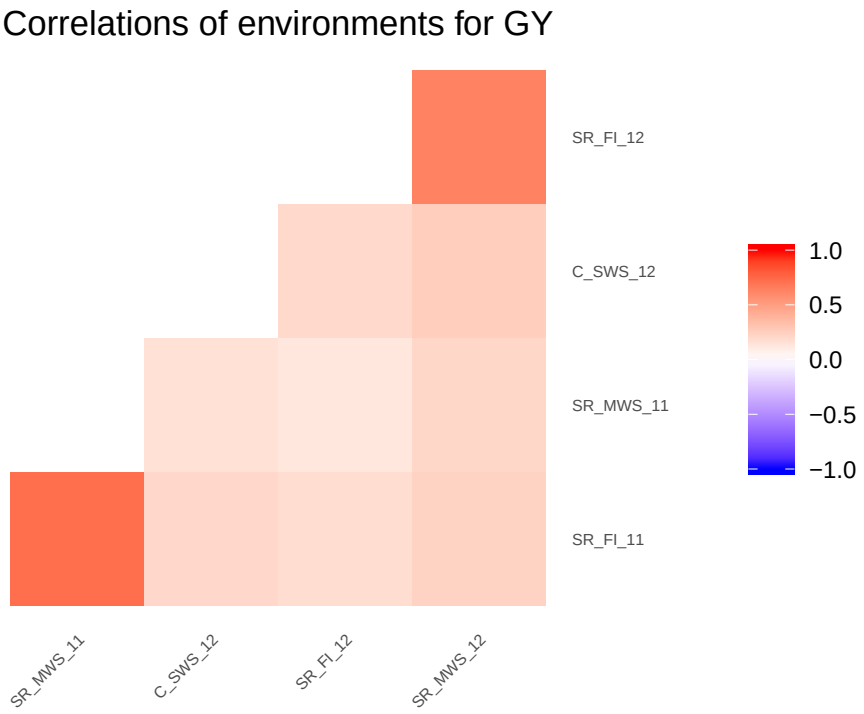
The trials in the boxplot can be grouped using the parameter `groupBy`. Colors can be applied to groups within trials using the parameter `colorBy`. The boxes for the (groups of) trials can be ordered using `orderBy`.

```
## Create a boxplot for grain yield with boxes grouped by year and repIds within
## years colored.
plot(wheatTD, plotType = "box", traits = "GY", groupBy = "year",
     colorBy = "repId", orderBy = "descending")
```



The last plot that can be made is a plot of the correlations between the fields for a specified trait. The order of the plotted fields is determined by clustering them and plotting closely related field close to each other.

```
## Create a correlation plot for grain yield.
plot(wheatTD, plotType = "cor", traits = "GY")
```



Modeling

After creating a `TD` object, a model can be fitted on the trial data. This is done using the function `STRunModel`. Depending on the design of the trial several different models can be fitted. The design can be given as a parameter in the function or included in the meta data of the `TD` object. In the former case the same model will be fitted for all trials, in the latter different models can be fitted for different trials depending on the design. If both are available the function parameter will always be leading.

The output of `STRunModel` is an object of class `SSA` (Single Site Analysis), a `list` of fitted models with one item for each trial the model was fitted for.

Model types

`STRunModel` uses three different engines for fitting the models, namely SpATS (Rodríguez-Álvarez et al. 2017), lme4 (Bates et al. 2015) and asreml (Gilmour et al. 2009). For models with row column or resolvable row column design, SpATS is the default engine, for the other models lme4. This can always be overruled by specifying the function parameter `engine`.

Models can be fitted for five different trial designs. These are listed in the following table with their respective model specifications.

design	model fitted
incomplete block design (ibd)	trait = genotype + *subBlock* + ϵ
resovable incomplete block design (res.ibd)	trait = genotype + **repId** + **repId:subBlock** + ϵ
randomized complete block design (rcbd)	trait = genotype + **repId** + ϵ

design	model fitted
row column design (rowcol)	trait = genotype + *rowId* + *colId* + ϵ
resolvable row column design (res.rowcol)	trait = genotype + **repId** + *repId:rowId* + *repId:colId* + ϵ

In the models above fixed effects are indicated in *italics* whereas random effects are indicated in **bold**. genotype can be fitted as fixed or as random effect depending on the value of the parameter `what`. Extra fixed effects may be fitted using the parameter `covariates`.
If SpATS is used as modeling engine, an extra spatial term is always included in the model. The same is done when the engine is asreml and the function parameter `trySpatial` is set to `TRUE`.

Using the `TD` object wheatTD from the previous section, a model for the trial SR_FI_11 and trait GY can now be fitted on the data as follows. For the design of the trial a resolvable row column design is assumed. Since `engine` is not supplied as a parameter SpATS is used for fitting the model.

```
## Fit a single trial model.
modWheatSp <- STRunModel(TD = wheatTD, trials = "SR_FI_11", traits = "GY",
                         design = "res.rowcol")
```

Note that by not supplying the `what` argument to the function, two models are fitted, one with genotype as a fixed effect and one with genotype as a random effect. The results of both these models are stored in the `SSA` object `modWheatSp`. This is very useful for extracting different results from the model later on. A trade off is that fitting two models takes more time than fitting only one so when fitting models on large datasets it is sensible to use `what` if only a subset of the results are needed as output.

```
## Fit a single trial model with genotype as random effect.
modWheatSp2 <- STRunModel(TD = wheatTD, trials = "SR_FI_11", traits = "GY",
                          what = "random", design = "res.rowcol")
```

Spatial models

When using SpATS as a modeling engine for fitting a model, an extra spatial component is always included in the model. This spatial component is composed using the `PSANOVA` function in the SpATS package which uses 2-dimensional smoothing with P-splines as described in Lee, Durbán, and Eilers (2013). See `help(PSANOVA, SpATS)` for a detailed description. The parameters `nseg` and `nest.div` of `PSANOVA` can be modified using the `control` parameter in `STRunModel`.
Refitting the model in the previous section specifying the number of segments for both rows and columns as 20 works as follows

```
## Fit a spatial single trial model using SpATS.
## Manually specify the number of segments for rows and columns.
modWheatSp3 <- STRunModel(TD = wheatTD, trials = "SR_FI_11", traits = "GY",
                          design = "res.rowcol",
                          control = list(nSeg = c(20, 20)))
```

Alternatively, spatial models can be fitted using asreml as modeling engine and setting the parameter `trySpatial = TRUE`. In this case seven models are fitted and the best model, based on a goodness-of-fit criterion, either AIC or BIC, is chosen. For a full specification of the models fitted see `help(STRunModel)`. The criterion to be used can be specified using the `control` parameter in `STRunModel`.
Fitting a model similar to the one above using asreml with BIC as goodness-of-fit criterion works as follows

```
if (requireNamespace("asreml", quietly = TRUE)) {
  ## Fit a spatial single trial model using asreml.
  modWheatAs <- STRunModel(TD = wheatTD, trials = "SR_FI_11", traits = "GY",
                           design = "res.rowcol", trySpatial = TRUE,
```

```
                           engine = "asreml", control = list(criterion = "BIC"))
}
```

The fitted models and the best model are stored in the output together with a summary table with details on the fitted models.

```
if (requireNamespace("asreml", quietly = TRUE)) {
  ## Overview of fitted models
  print(modWheatAs$SR_FI_11$sumTab$GY, digits = 2)
}
#>     spatial random   AIC   BIC  row  col   error correlated error converge
#> 1      none   NULL 12241 12260   NA   NA 1019615                NA      TRUE
#> 2  AR1(x)id   NULL 12170 12194 0.44   NA 1072417                NA      TRUE
#> 3  id(x)AR1   NULL 12033 12057   NA 0.73 1420739                NA      TRUE
#> 4 AR1(x)AR1   NULL 11992 12021 0.43 0.76 1700929                NA      TRUE
#> 5  AR1(x)id  units 12172 12200 0.45   NA   30969           1041452      TRUE
#> 6  id(x)AR1  units 12018 12046   NA 0.88  217227           1583179      TRUE
#> 7 AR1(x)AR1  units 11935 11968 0.63 0.87  277864           1373645      TRUE
```

As the overview shows the best model, the model with the lowest BIC, is AR1(x)AR1.

Model output

Since genotype has been fitted both as fixed and as random factor in `modWheatSp` it is possible to calculate both the Best Lineair Unbiased Estimators (BLUEs) and the Best Lineair Unbiased Predictors (BLUPs). Therefore both are printed in the summary of the model together with their respective standard errors.

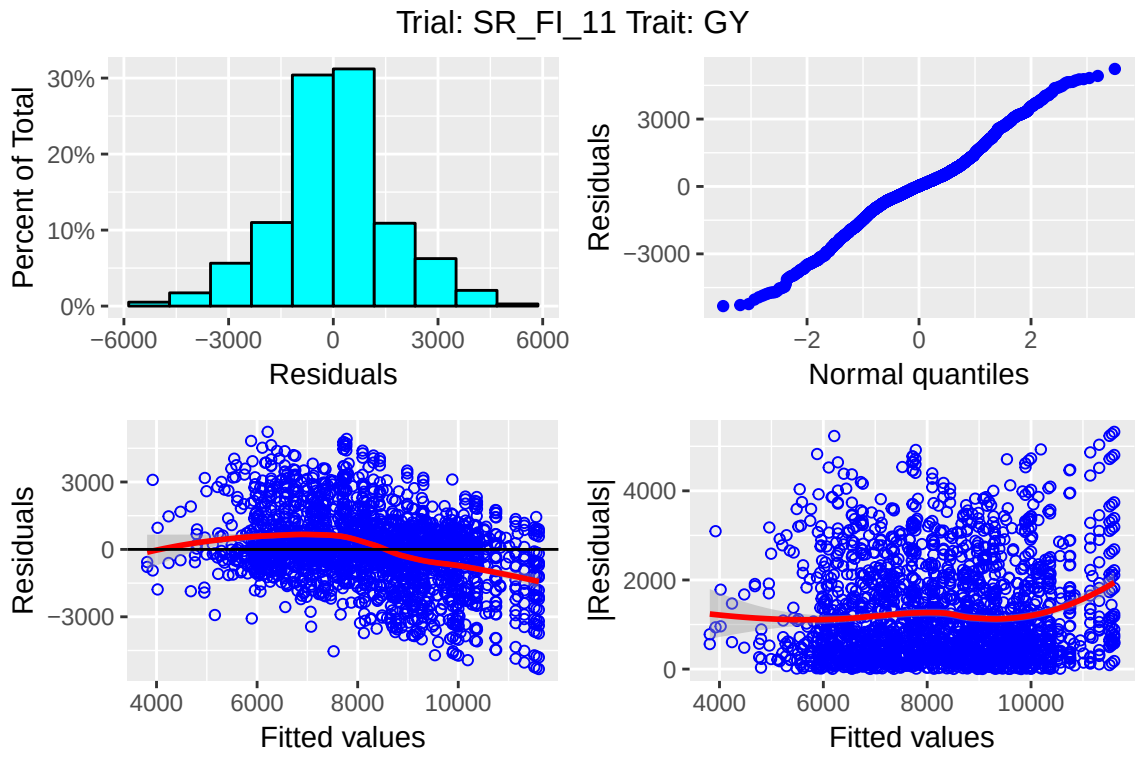
```
## Set nBest to 5 to decrease size of output.
summary(modWheatSp, nBest = 5)
#> Summary statistics
#> ==================
#>
#> Summary statistics for GY in SR_FI_11
#>
#>                                  GY
#> Number of observations          794
#> Number of missing values          6
#> Mean                         8015.60
#> Median                       8021.00
#> Min                          2239.50
#> Max                         12992.00
#> Lower quartile               6728.50
#> Upper quartile               9321.75
#> Variance                  3080361.436
#>
#>
#> Estimated heritability
#> ======================
#>
#> Heritability: 0.65
#>
#> Predicted means (BLUEs & BLUPs)
#> ===============================
#> Best 5 genotypes
```

```
#>          BLUEs       SE    BLUPs     SE
#> G369 10649.75   681.05  9643.18 542.30
#> G349 10351.81   668.15  9563.50 537.49
#> G341 10294.38   665.01  9468.74 536.49
#> G227 10043.79   673.05  9351.96 540.84
#> G271 10010.28   663.48  9398.66 536.11
```

Model plots

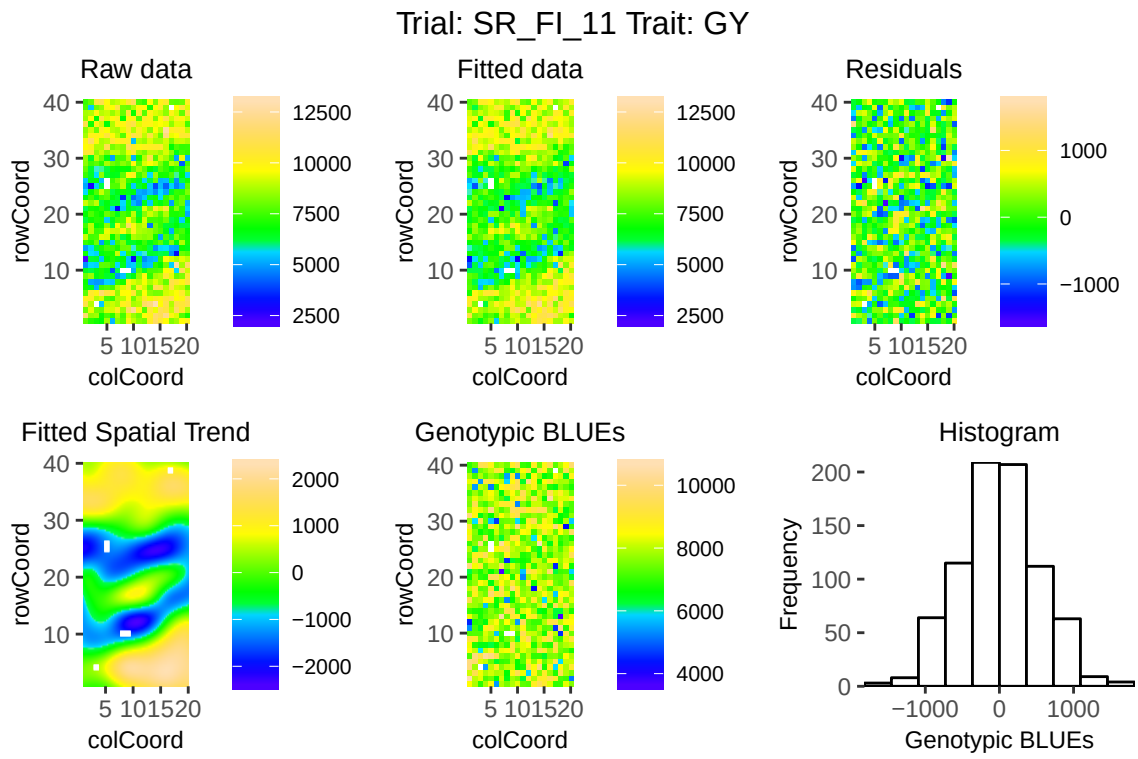
Two types of plots can be made for fitted models. The first is a series of four plots, a histogram of the residuals, normal quantiles of the residuals, a scatter plot of residuals against fitted values and a scatter plot of absolute value of residuals against fitted values. These plots can be made by calling `plot` on the `SSA` object. Plots can be made for multiple trials and multiple traits simultaneously, either for the model with genotype as fixed effect or for the model with genotype as random effect. By default plots are made for all trials and all traits, but this can be restricted using the parameters `trials` and `traits`. If only one model is fitted, i.e. only the model with genotype as fixed effect or only the model with genotype as random effect, the results of this model will be plotted. In case both models were fitted, as a default the results will be plotted for the model with genotype fixed. This can be changed using the parameter `what`.

```
## Base plots for the model with genotype fitted as random effect.
plot(modWheatSp, what = "random")
```



The second type of plot consists of five plots, spatial plots of the raw data, fitted values, residuals and either BLUEs or BLUPs, and a histogram of the BLUEs or BLUPs. If SpATS was used for modeling an extra plot with the fitted spatial trend is included. The call for creating these plots differs from the base plots only by an extra parameter `plotType = "spatial"`. Note that spatial plots can only be made if spatial information, i.e. `rowCoord` and `colCoord`, is available in the `TD` object.

```
## Spatial plot for the model with genotype fitted as fixed effect.
plot(modWheatSp, plotType = "spatial")
```



Model reports

As with many objects in the RAP package there is a `report` function available for `SSA` objects. This `report` function creates a pdf report describing the main results of the fitted model. Also the tex file and figures used for generating the pdf report are saved. By editing the tex file it is possible to modify the report to ones needs, creating high flexibility.

When no outfile is specified, a report will be created with a default name, i.e. "modelReport.pdf", in the current working directory. The parameter `outfile` can be used to change the location. This value of this parameter should be a valid location and name for a **pdf** file, i.e. including the postfix ".pdf". Non-existing directories are created by the `report` function.

The report contains general information on the model fitted, a summary of the results, the plots described in the previous section, a `list` of best (highest BLUEs or BLUPs) genotypes and a scatter plot of all genotypes and their BLUEs or BLUPs. If, for the modeled trait, a low value means the genotype is performing well, then the selection on the best genotypes can be based on the lowest BLUEs or BLUPs by setting the parameter `descending` to `TRUE` in the report function.

```
## Create a report in the current working directory
report(modWheatSp)
## Create a report for the model with genotype fitted as random.
report(modWheatSp, outfile = "./myReports/wheatReport.pdf", what = "random")
```

Reporting for an `SSA` object is always done for one trait in one trial for one fitted model. Parameters `trait`, `trial` and `what` can be used for selecting the model for which the report should be made.

Extracting model results

After fitting a model, various results can be extracted or calculated from the fitted model using the function `STExtract`. This can be anything from a single result for one trait and one trial to a `list` of different results for all models in an `SSA` object. The results that can be extracted depend on the type of model fitted and sometimes on the modeling engine as well. For example, BLUEs can only extracted if genotype was fitted as fixed effect whereas BLUPs and heritabilities can only be extracted and calculated if genotype was fitted as random effect.

Possible results to extract

All results that can be extracted are shown in the table below. The first column shows what model needs to be fitted in order to be able to extract the result. Here F stands for genotype as fixed effect and R for genotype as random effect. In the second column is the result. This is also the value to be used for the parameter `what` in `STExtract` needed to extract the corresponding result. The last column gives a short description of the result that will be extracted and, where needed, also states for which modeling engines it can be extracted.

model	result	description
F	BLUEs	Best Lineair Unbiased Estimators
F	seBLUES	standard errors of the BLUEs
R	BLUPs	Best Lineair Unbiased Predictors
R	seBLUPs	standard errors of the BLUPs
F	ue	unit errors - only for `lme4` and `asreml`
R	heritability	broad sense heritability
F	varCompF	variance components for the model with genotype as fixed component
R	varCompR	variance components for the model with genotype as random component
R	varGen	genetic variance component
R	varErr	residual variance component - only for `lme4` and `asreml`
R	varSpat	spatial variance components - only for `SpATS`
F	fitted	fitted values for the model with genotype as fixed component
F	resid	residuals for the model with genotype as fixed component
F	stdRes	standardized residuals for the model with genotype as fixed component
R	rMeans	fitted values for the model with genotype as random component
R	ranEf	random genetic effects
F	wald	results of the wald test - only for `lme4` and `asreml`
F	CV	Coefficient of Variation - only for `lme4` and `asreml`
F	rDf	residual degrees of freedom
R	effDim	effective dimensions - only for `SpATS`
R	ratEffDim	ratios of the effective dimensions - only for `SpATS`
F	sed	standard error of difference - only for `asreml`
F	lsd	least significant difference - only for `asreml`

Using `what = "all"` in the function call, extracts all results possible for the fitted model. This is also the default.

Below are some examples of extracting results from a fitted model. Recall that `modWheatSp` contains two fitted models, one with genotype as fixed effect and one with genotype as random effect.

```
## Extract BLUEs
BLUEsWheat <- STExtract(SSA = modWheatSp, what = "BLUEs")
## Extract BLUEs and BLUPs
predWheat <- STExtract(SSA = modWheatSp, what = c("BLUEs", "BLUPs"))
```

Both `BLUEsWheat` and `predWheat` are a `list` with one item, the trial used for modeling. In general, when extracting results, there will be an item in the `list` for every trial for which results were extracted. These items are a `list` themselves with an item for every statistic that has been extracted, so one item for `BLUEsWheat`, a `data.frame` containing the BLUEs, and two for `predWheat`, the `data.frames` containing BLUEs and BLUPs respectively.

Adding extra variables

The `data.frame` BLUEs in either of the `lists` consists of only two columns, genotype and GY. If the model would have been fitted for multiple traits all these traits would become columns in the `data.frame`. It might be useful to add extra columns from the data used to fit the model to the output. This can be achieved using the paramater `keep` in `STExtract`. To include the trial in the output, useful for easily combining several `data.frames` with BLUEs and using them for a GxE analysis, use the following command

```
## Extract BLUEs from the fitted model.
BLUEsWheat2 <- STExtract(SSA = modWheatSp, what = "BLUEs", keep = "trial")
head(BLUEsWheat2[["SR_FI_11"]]$BLUEs)
#>   genotype    trial       GY
#> 1     G001 SR_FI_11 8828.678
#> 2     G002 SR_FI_11 8592.090
#> 3     G003 SR_FI_11 8942.543
#> 4     G004 SR_FI_11 6804.189
#> 5     G005 SR_FI_11 8224.345
#> 6     G006 SR_FI_11 7443.286
```

Not every column from the original `TD` object can be included in the extracted data in this way. Only columns for which the values are uniform per genotype can be included. For example, the column `repId` containing replicates, that has several different values for a single genotype, cannot be included. When trying to do so it will be dropped with a warning.
It is however possible to include `repId` when extracting fitted values, since for each observation in the original data a fitted value is computed.

```
## Extract fitted values from the model.
fitVals <- STExtract(SSA = modWheatSp, what = "fitted",
                     keep = c("trial", "repId"))
head(fitVals[["SR_FI_11"]]$fitted)
#>   repId genotype    trial       GY
#> 1     1     G275 SR_FI_11 8234.353
#> 2     1     G325 SR_FI_11 7660.984
#> 3     1     G119 SR_FI_11 6675.174
#> 4     1     G138 SR_FI_11 7638.940
#> 5     1     G296 SR_FI_11 8409.450
#> 6     1     G257 SR_FI_11 5556.333
```

Prepare data for GxE analysis

To use the BLUEs or BLUPs from the fitted model in a GxE analysis they have to be converted again into a `TD` object. For this the function `SSAtoTD` can be used. It creates a `TD` object from a fitted model outputting one or more of the following: BLUEs, standard errors of BLUEs, BLUPs and standard errors of BLUPs. Optionally a column `wt` with weights (calculated as $(1/seBLUEs)^2$) can be added as well. In the same way as described in the previous section extra columns can be added to the output using the parameter `keep`.

```
## Fit a model for all trials with genotype as fixed factor.
modWheatSpTot <- STRunModel(TD = wheatTD, traits = "GY", what = "fixed",
                            design = "res.rowcol")
## Create a TD object containing BLUEs and standard errors of BLUEs.
TDGxE <- SSAtoTD(SSA = modWheatSpTot, what = c("BLUEs", "seBLUEs"))
## Add weights to the output.
TDGxE2 <- SSAtoTD(SSA = modWheatSpTot, what = c("BLUEs", "seBLUEs"), addWt = TRUE)
```

See the vignette Genotype by Environment analysis using RAP for an overview of how to use the input just created for performing GxE analysis.

References

Bates, Douglas, Martin Mächler, Ben Bolker, and Steve Walker. 2015. "Fitting Linear Mixed-Effects Models Using Lme4." *Journal of Statistical Software* 67 (1). https://doi.org/10.18637/jss.v067.i01.

Gilmour, A.R., B.J. Gogel, B.R. Cullis, and R. Thompson. 2009. "ASReml User Guide Release 3.0." https://www.vsni.co.uk/.

Lado, Bettina, Ivan Matus, Alejandra Rodríguez, Luis Inostroza, Jesse Poland, François Belzile, Alejandro del Pozo, Martín Quincke, Marina Castro, and Jarislav von Zitzewitz. 2013. "Increased Genomic Prediction Accuracy in Wheat Breeding Through Spatial Adjustment of Field Trial Data." *G3: Genes|Genomes|Genetics* 3 (12): 2105–14. https://doi.org/10.1534/g3.113.007807.

Lee, Dae-Jin, María Durbán, and Paul Eilers. 2013. "Efficient Two-Dimensional Smoothing with P-Spline Anova Mixed Models and Nested Bases." *Computational Statistics & Data Analysis* 61 (May): 22–37. https://doi.org/10.1016/j.csda.2012.11.013.

Rodríguez-Álvarez, María, Martin P. Boer, Fred van Eeuwijk, and Paul H.C. Eilers. 2017. "Correcting for Spatial Heterogeneity in Plant Breeding Experiments with P-Splines." *Spatial Statistics* 23 (October): 52–71. https://doi.org/10.1016/j.spasta.2017.10.003.