Sensitivity Analysis

Sensitivity Analysis

Many approaches (entire text books)

Two main classes

- global simultaneous: vary all parameters over possible ranges
- local parameter specific: hold all other parameters content and then vary

Challenge is sampling parameter uncertainty space and balancing this against computational limits

Optimization - special type of sensitivity analysis

Single Parameter Sensitivity

Vary one parameter but hold the others constant

Useful for focused analysis but..

- sensitivity to a parameter may change as a function of other parameters

Consider the sensitivity of a seasonal snow accumulation and melt model to both temperature and radiation

- for high temperatures, radiation may not impact rates substantially since there is less snow
- for lower temperature, radiation may matter much more - so temperatures related to radiation absorption (e.g albedo) will have a greater impact on output

Steps in Sensitivity Analysis

Define the range (pdf) of input parameters

Define the outputs to be considered (e.g if you had streamflow are you looking at daily, max, min, annual)

Sample the pdf of input parameters and use this sample to run the model - repeat many times

Graph the results

Quantify sensitivity

Sensitivity Analysis: What ranges should I vary my parameters over!

- range (min, max, middle values)
- pdf (probability density function)
- If parameters are physically based this can come from literature (e.g range of snow albedo's, range of hydraulic conductivity in a soil)
- If parameter's are estimated (e.g a regression slope, coefficients from another model) then statistics such as confidence bounds can help you to define the ranges

Approaches to sampling parameter space

Random - Monte Carlo

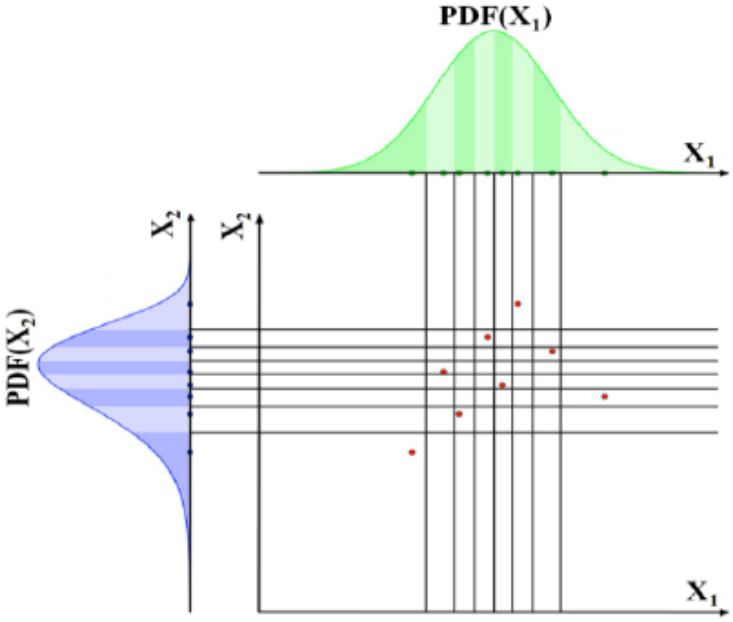
Latin Hypercube

Sobel

Difference is in how you sample parameter space - tradeoffs with efficiency, likelihood of capturing responses from rarer parameters

Latin Hyper Cube

generating random samples from equally probability



intervals

Tools in R

lhs (Parameter Space Exploration with Latin Hypercubes) Library for Latin Hypercube Sampling and Sensitivity Analysis

Install lhs library

What we need to use the LHS function

- number of parameters that you want to perform sensitivity analysis on
- number of samples
- distributions of the parameters

"qnorm","qunif","qlnorm","qgamma" many others]

More Distributions and Parameters

Steps for Latin HyperCube Sampling

- create a matrix (essentially our hypter cube) with random samples for the number of parameters that you have
  - *these are essentially quantiles that will let us fully sample the parameter space*
- use these quantiles to sample from actual parameter distributions
  - *we do this by adding in the characteristics of the distributions*
    - type (e.g normal, uniform)
    - parameters (e.g mean, variance)

Example of using Latin Hypercube sampling for sensitivity analysis

Lets look at our almond yield example

```
# for formal sensitivity analysis it is useful to describe output in
# several summary statistics - how about mean, max and min yield
source("../R/compute_almond_yield.R")


# set a random seed to make things 'random'
set.seed(1)

# which parameters
pnames = c("Tmincoeff1", "Tmincoeff2", "Pcoeff1", "Pcoeff2", "intercep")

# how many parameters
npar =  length(pnames)
# how many samples
nsample = 100

parm_quant = randomLHS(nsample, npar)
colnames(parm_quant)=pnames
# choose distributions for parameters - this would come from
# what you know about the likely range of variation
# then use our random samples to pick the quantiles

parm = as.data.frame(matrix(nrow=nrow(parm_quant), ncol=ncol(parm_quant)))
colnames(parm) = pnames
# for each parameter pick samples
# I'm using several examples normal distribution (with 10% standard deviation) and
        uniform with +- 10%
# in reality I should pick distribution from knowledge about uncertainty in parameters

# to make it easy to change i'm setting standard deviation / range variation to a
        variable
pvar = 10

parm[,"Tmincoeff1"] = qnorm(parm_quant[,"Tmincoeff1"], mean=-0.015, sd=0.015/pvar)
parm[,"Tmincoeff2"] = qnorm(parm_quant[,"Tmincoeff2"], mean=-0.0046, sd=0.0046/pvar)

# for uniform I'm using +- 10%
```

```
parm[,"Pcoeff1"] = qunif(parm_quant[,"Pcoeff1"], min=-0.07-0.07/pvar,
        max=-0.07+0.07/pvar)
parm[,"Pcoeff2"] = qunif(parm_quant[,"Pcoeff2"], min=-0.0043-0.0043/pvar,
        max=-0.0043+0.0043/pvar)


parm[,"intercep"] = qnorm(parm_quant[,"intercep"], mean=0.28, sd=0.28/pvar)
# note I could also index by column number by names keep things more clear, fewer
        mistakes
#parm[,5] = qnorm(parm_quant[,5], mean=0.28, sd=0.28/pvar)


head(parm)
```

```
##    Tmincoeff1   Tmincoeff2     Pcoeff1      Pcoeff2  intercep
## 1 -0.01593953 -0.005063382 -0.06570787 -0.004285695 0.3121720
## 2 -0.01511405 -0.004809981 -0.07310106 -0.004654693 0.2889460
## 3 -0.01290502 -0.004404387 -0.06523886 -0.004156141 0.2419644
## 4 -0.01699395 -0.004511290 -0.06320412 -0.004434716 0.3169259
## 5 -0.01481798 -0.004818892 -0.06715120 -0.004238696 0.2731872
## 6 -0.01425928 -0.003927767 -0.06765378 -0.004584168 0.2714421
```

Run model for parameter sets

- We will do this in R
- We can use **pmap** to efficiently run our model for all of our parameter sets

but first

Examining Output

Sensitivity of What?

If your model is estimating a single value, you are done

- long term mean almond yield anomoly
- mean profit from solar

But models are often estimating multiple values

- streamflow
- almond yield anomoly for multiple years

In that case to quantify sensitivity you need summary metrics

- mean
- max
- min
- variance

Which one depends on what you care about

Example: Almond Yield

- I'm going to return some summary information from my model
- You could do this in a separate function!

I'll return

- max yield
- min yield
- average yield

often helpful when doing sensitivity analysis to place w

```
#
compute_almond_yield
```

```
## function (clim, Tmincoeff1 = -0.015, Tmincoeff2 = -0.0046,
Pcoeff1 = -0.07,
##     Pcoeff2 = 0.0043, intercep = 0.28)
## {
##     tmp = clim %>% group_by(month, year) %>%
dplyr::summarize(tmin_c = min(tmin_c),
##         .groups = "drop")
##     Feb_minT = (tmp %>% subset(month == 2))$tmin_c
##     tmp = clim %>% group_by(month, year) %>%
dplyr::summarize(precip = sum(precip),
##         .groups = "drop")
##     Jan_P = (tmp %>% subset(month == 1))$precip
##     yield = Tmincoeff1 * Feb_minT + Tmincoeff2 * Feb_minT^2 +
##         Pcoeff1 * Jan_P + Pcoeff2 * Jan_P^2 + intercep
##     return(list(maxyield = max(yield), minyield = min(yield),
##         meanyield = mean(yield)))
## }
```

now lets run for our parameters and climate data

```
# read in the input data
clim=read.table("../data/clim.txt", header=T)




# lets now run our model for all of the parameters generated by LHS
# pmap is useful here - it is a map function that uses the actual names of input
        parameters

yields = parm %>% pmap(compute_almond_yield,clim=clim)

# notice that what pmap returns is a list
head(yields)
```

```
## [[1]]
## [[1]]$maxyield
## [1] 0.1262876
##
## [[1]]$minyield
## [1] -2006.212
##
## [[1]]$meanyield
## [1] -199.0907
##
##
## [[2]]
## [[2]]$maxyield
## [1] 0.1124944
##
## [[2]]$minyield
## [1] -2180.083
##
## [[2]]$meanyield
## [1] -216.4686
##
##
## [[3]]
```

```
## [[3]]$maxyield
## [1] 0.08477128
##
## [[3]]$minyield
## [1] -1946.579
##
## [[3]]$meanyield
## [1] -193.296
##
##
## [[4]]
## [[4]]$maxyield
## [1] 0.1382241
##
## [[4]]$minyield
## [1] -2072.673
##
## [[4]]$meanyield
## [1] -205.3581
##
##
## [[5]]
## [[5]]$maxyield
## [1] 0.09792737
##
## [[5]]$minyield
## [1] -1985.683
##
## [[5]]$meanyield
## [1] -197.2151
##
##
## [[6]]
## [[6]]$maxyield
## [1] 0.1183692
##
## [[6]]$minyield
## [1] -2144.044
##
## [[6]]$meanyield
## [1] -212.5893
```

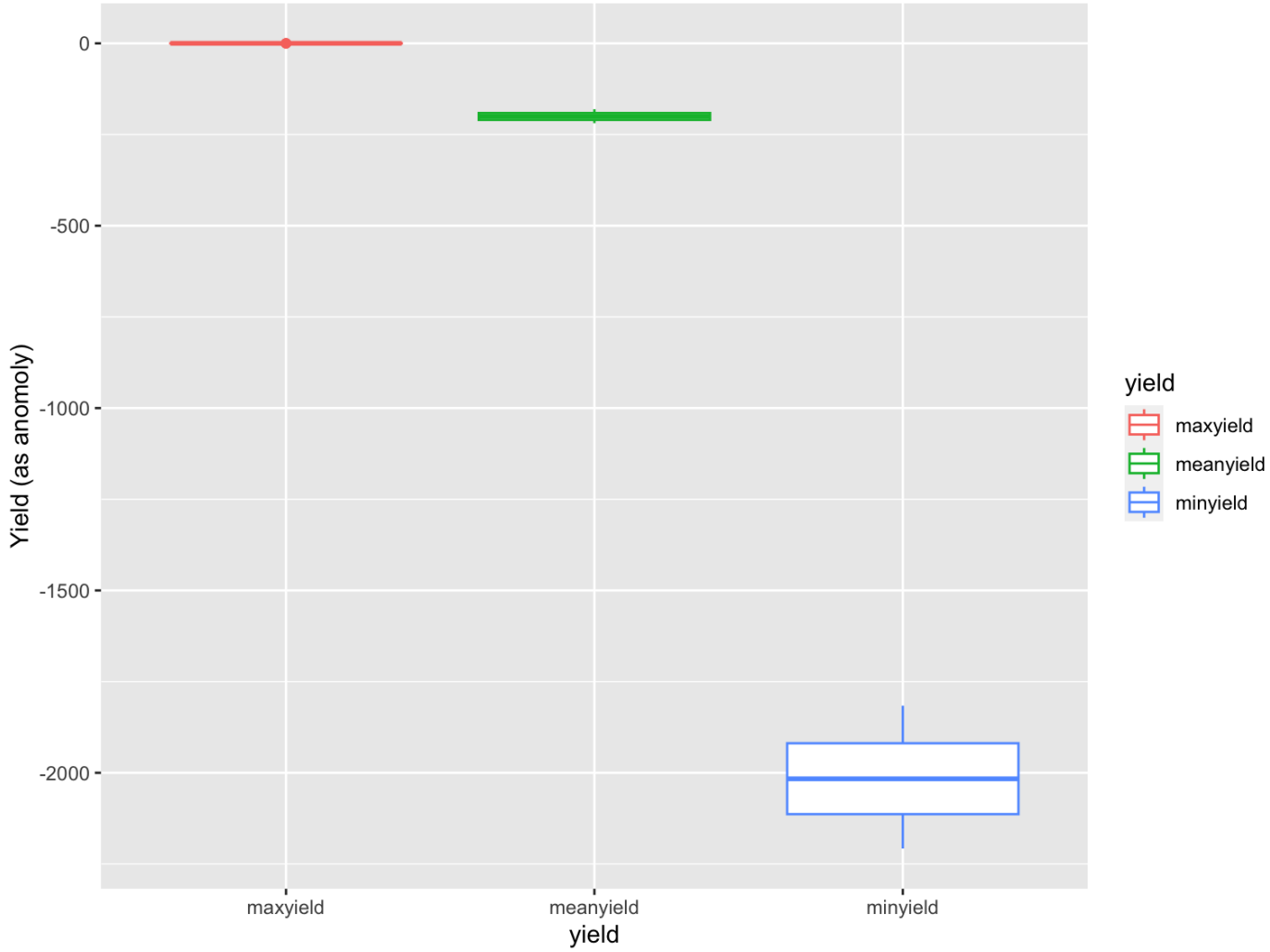
```
# turn results in to a dataframe for easy display/analysis
yieldsd = yields %>% map_dfr(`[`,c("maxyield","minyield","meanyield"))
```

Plotting

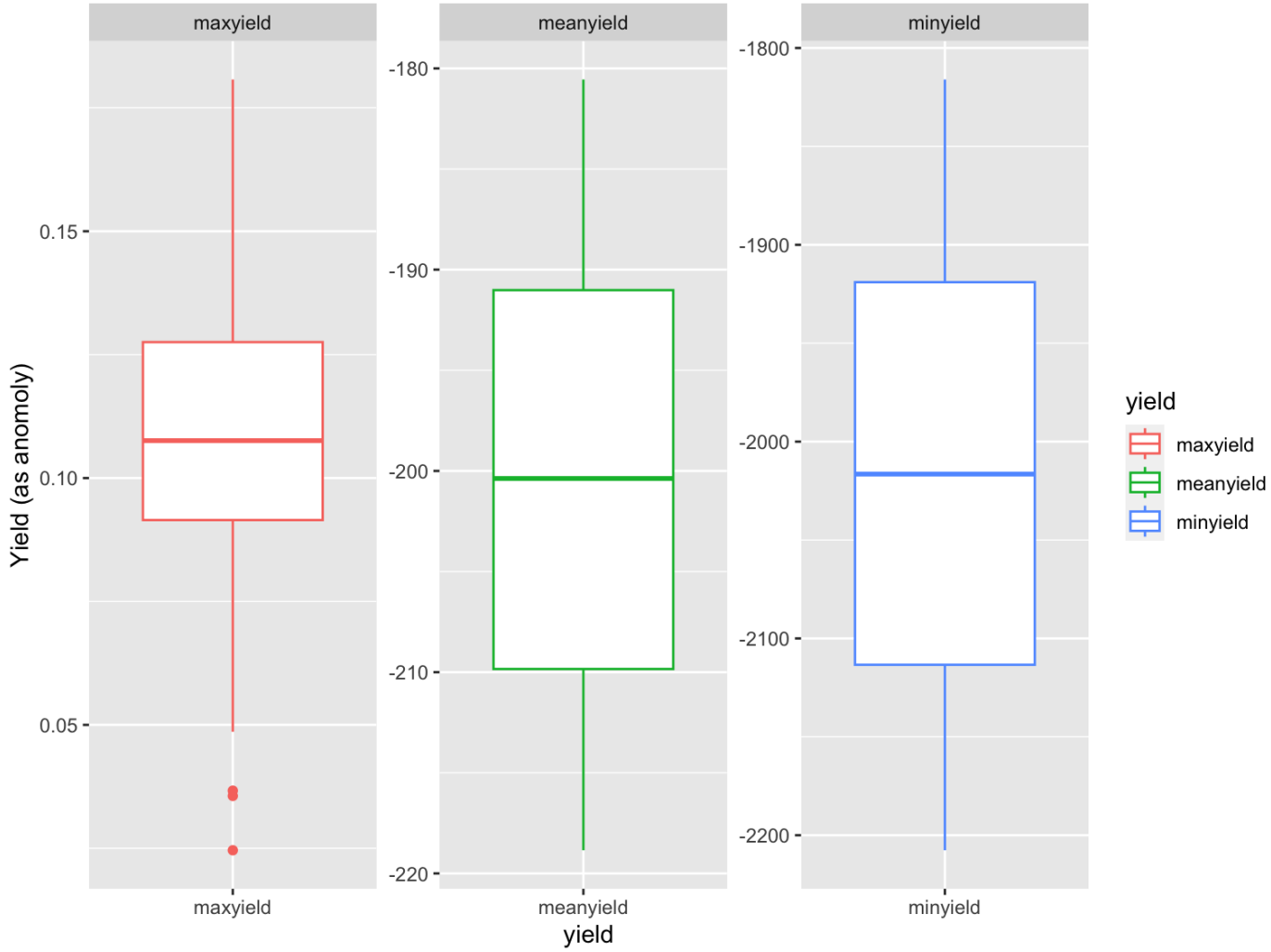
Plot relationship between parameter and output to understand how uncertainty in parameter impacts the output to determine over what ranges of the parameter uncertainty is most important (biggest effect)

- Use a box plot (of output) to graphically show the impact of uncertainty on output of interest
- To see more of the distribution - graph the cumulative distribution
  - *high slope, many values in that range*
  - *low slope, few values in that range*
  - *constant slope, even distribution*
- Scatterplots against parameter values

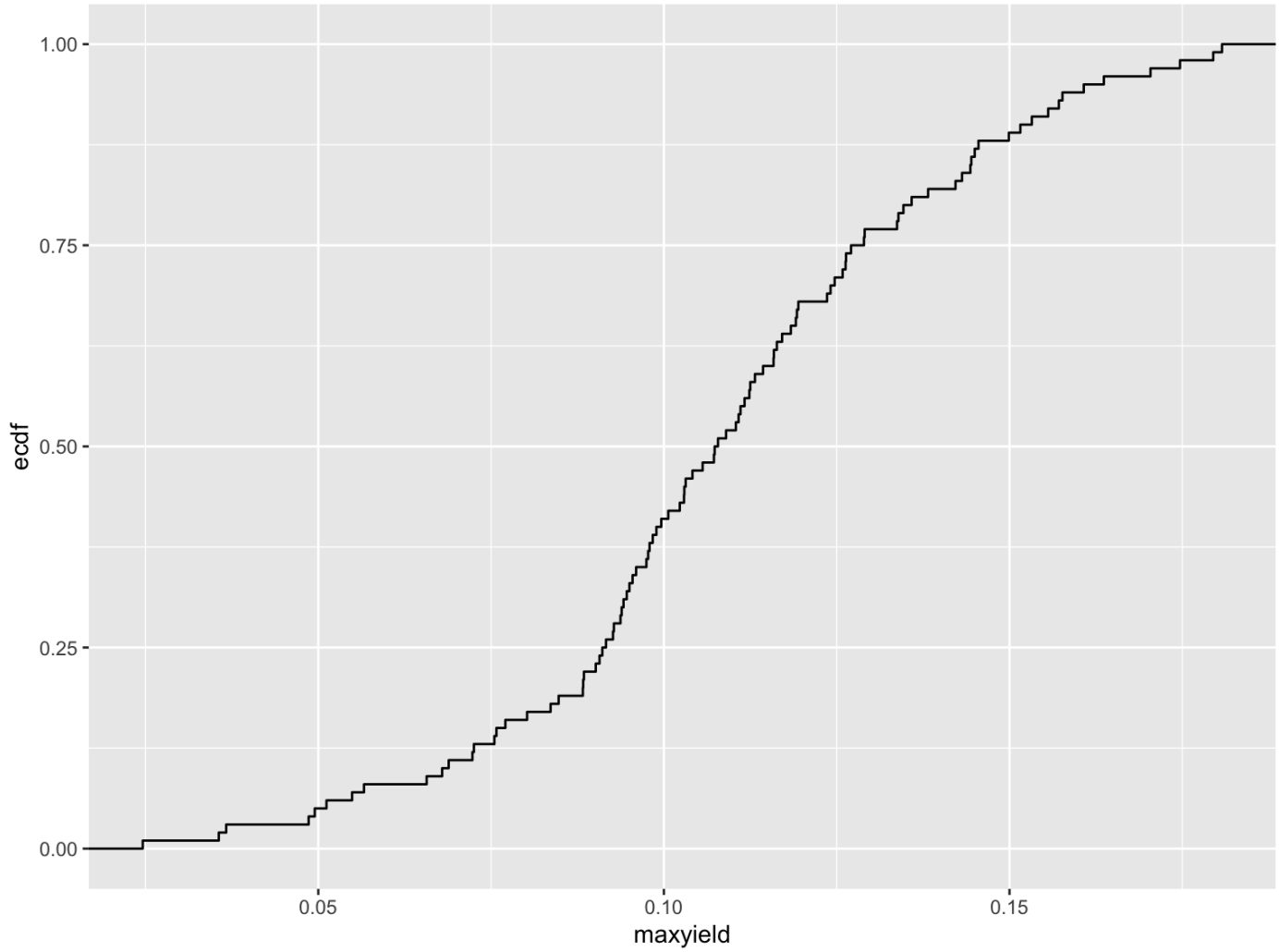
```
# add uncertainty bounds on our estimates
tmp = yieldsd %>% gather(value="value", key="yield")
ggplot(tmp, aes(yield, value, col=yield))+geom_boxplot()+
  labs(y="Yield (as anomoly)")
```



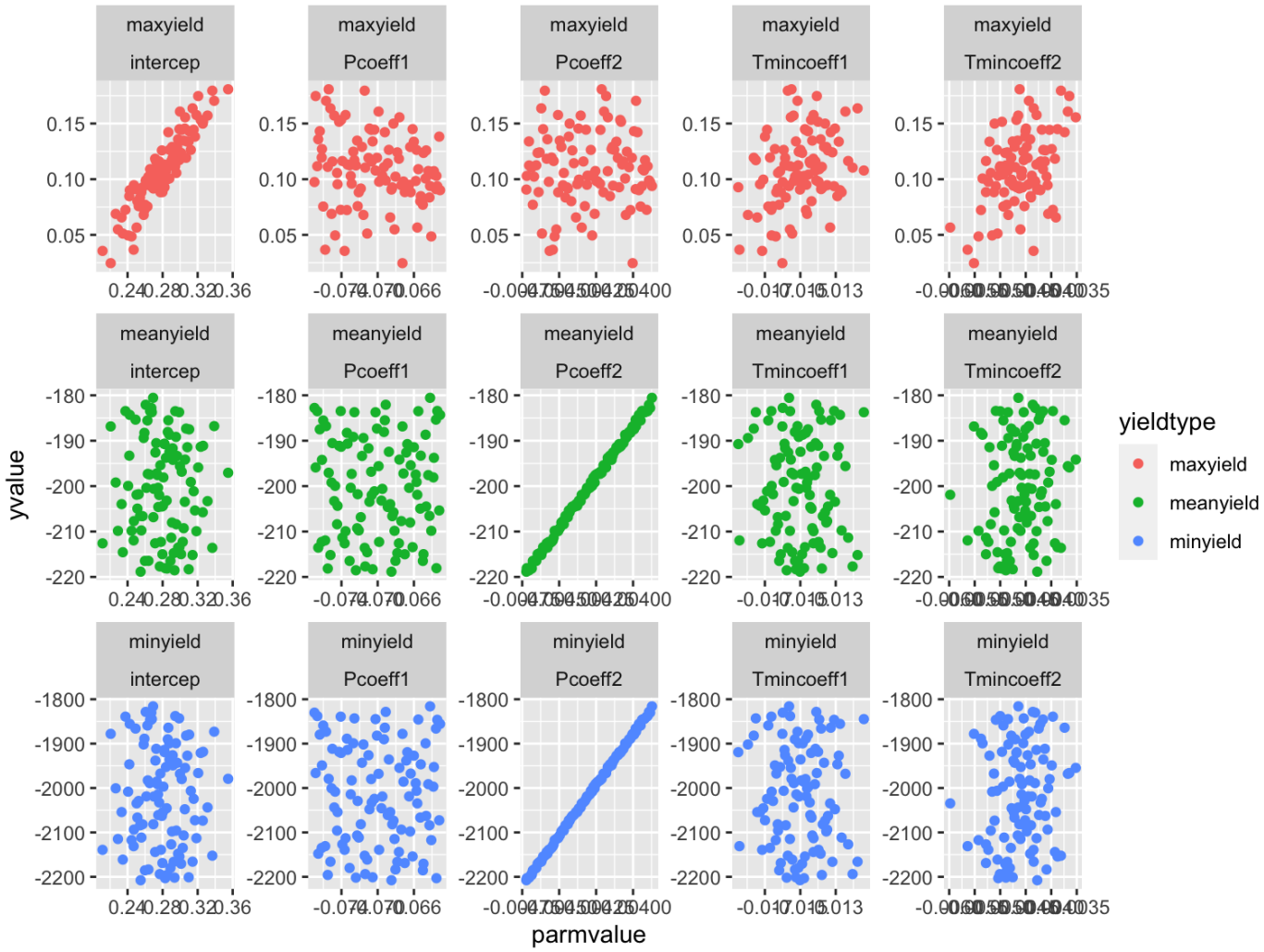

```
# note that you don't see the ranges because of the scale (min yield anomoly much
        smaller than max) - here's a more informative way to graph
ggplot(tmp, aes(yield, value, col=yield))+
  geom_boxplot()+labs(y="Yield (as anomoly)")+
  facet_wrap(~yield, scales="free" )
```




```
# cumulative distribution
ggplot(yieldsd, aes(maxyield))+stat_ecdf()
```




```
# plot parameter sensitivity
# a bit tricky but nice way to make it easy to plot all parameters against all values
tmp = cbind.data.frame(yieldsd, parm)
tmp2 = tmp %>% gather(maxyield, minyield, meanyield, value="yvalue",key="yieldtype")
tmp3 = tmp2 %>% gather(-yvalue, -yieldtype, key="parm", value="parmvalue")
ggplot(tmp3, aes(parmvalue, yvalue,
        col=yieldtype))+geom_point()+facet_wrap(~yieldtype*parm, scales="free",
        ncol=5)
```


maxyield
intercep
maxyield
Pcoeff1
maxyield
Pcoeff2
maxyield
Tmincoeff1
maxyield
Tmincoeff2
meanyield
intercep
meanyield
Pcoeff1
meanyield
Pcoeff2
meanyield
Tmincoeff1
meanyield
Tmincoeff2
minyield
intercep
minyield
Pcoeff1
minyield
Pcoeff2
minyield
Tmincoeff1
minyield
Tmincoeff2
yvalue
parmvalue
yieldtype
maxyield
meanyield
minyield

Quantifying Sensitivity

If relationships between output and input are close to linear,

- Correlation coefficient between output and each input (separately)
- Partial correlation coefficient (PCC) for multiple parameters;

```
* Correlation after effects of other parameters accounted
for

* Correlation of X and Y given Z

* Computed as the correlation of the *residuals* from
linear regression of

    * X with Z

    * Y with Z
```

We can essentially fit a regression relationship between input and output - and slope is a measure of sensitivity (standardized regression coefficients)

$$y = \beta * x + \alpha$$

$$\beta_{std} = \beta * \frac{s_x}{s_y}$$

where s_x and s_y are standard deviations of x and y

Quantifying Sensitivity

- Correlation Coefficient - linear monotonic

Non-linear relationship, but monotonic between x and y - you can rank transform x and y to developed

- Partial Rank Correlation Coefficient - non-linear but monotonic

R function *pcc* in *sensitivity* package can compute these for you AND makes it easy to graph the results

It can be used for both partial correlation coefficients and partial rank correlation coefficients

Correlation and Pcc for Maximum Yield

```
# combine parameter sets with output


# simple correlation coefficients
result = map(parm, cor.test, y=yieldsd$maxyield)
result$Tmincoeff1
```

```
##
##  Pearson's product-moment correlation
##
## data:  .x[[i]] and yieldsd$maxyield
## t = 3.4665, df = 98, p-value = 0.0007842
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.1433856 0.4947942
## sample estimates:
##       cor
## 0.3304958
```

```
result$Tmincoeff2
```

```
##
##  Pearson's product-moment correlation
##
## data:  .x[[i]] and yieldsd$maxyield
## t = 4.6048, df = 98, p-value = 1.241e-05
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.2456988 0.5708875
## sample estimates:
##       cor
## 0.4217628
```

```
result$Pcoeff1
```

```
##
##  Pearson's product-moment correlation
##
## data:  .x[[i]] and yieldsd$maxyield
```

```
## t = -2.1013, df = 98, p-value = 0.03818
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.38821821 -0.01169142
## sample estimates:
##        cor
## -0.2076327
```

```
result$Pcoeff2
```

```
##
##  Pearson's product-moment correlation
##
## data:  .x[[i]] and yieldsd$maxyield
## t = 0.16865, df = 98, p-value = 0.8664
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.1799868  0.2127399
## sample estimates:
##       cor
## 0.0170335
```

```
result$intercep
```

```
##
##  Pearson's product-moment correlation
##
## data:  .x[[i]] and yieldsd$maxyield
## t = 23.213, df = 98, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.8829586 0.9454478
## sample estimates:
##       cor
## 0.9198483
```

```
# just the confidence interval
justconf = result %>% map_df("conf.int")
justconf
```

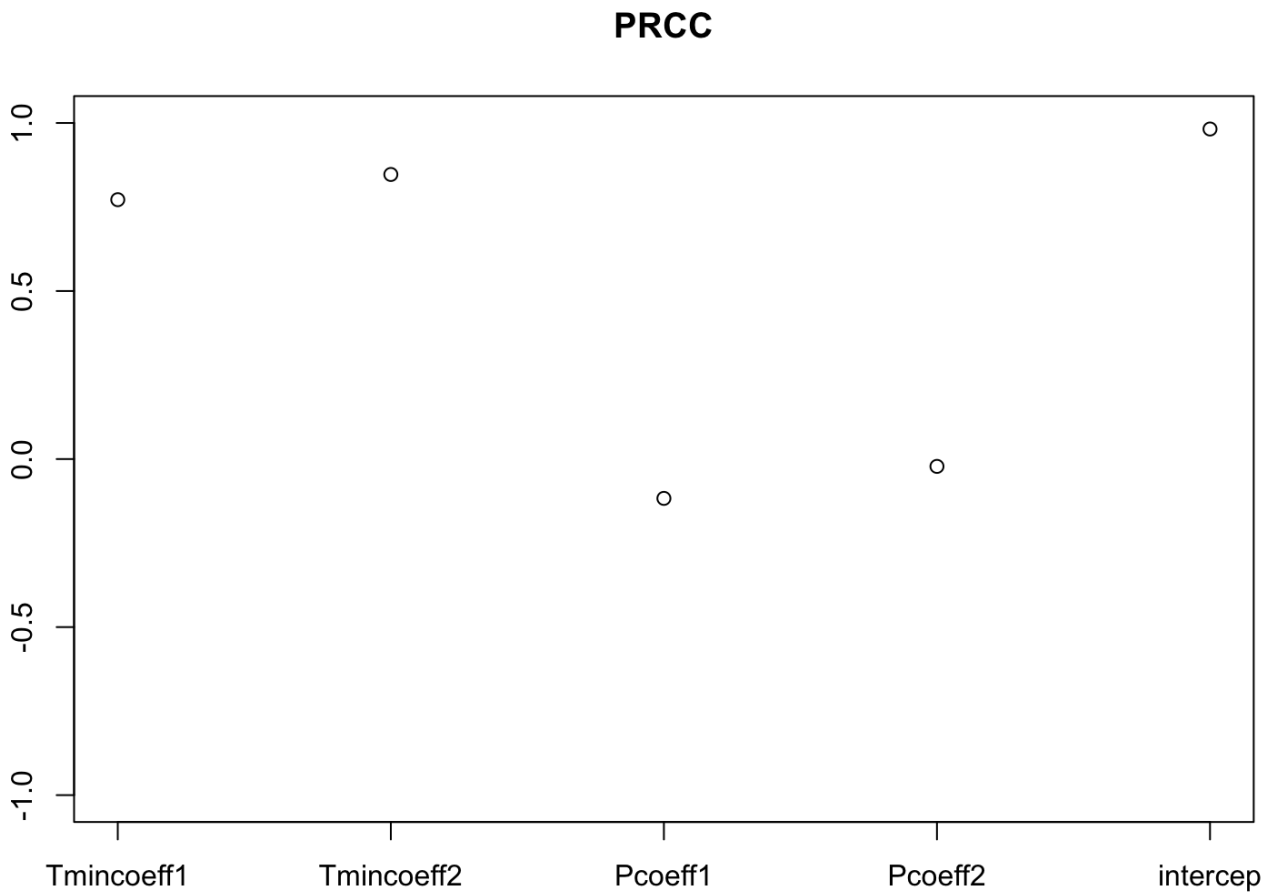
```
## # A tibble: 2 × 5
##   Tmincoeff1 Tmincoeff2 Pcoeff1 Pcoeff2 intercep
##        <dbl>      <dbl>   <dbl>   <dbl>    <dbl>
## 1      0.143      0.246 -0.388   -0.180    0.883
## 2      0.495      0.571 -0.0117   0.213    0.945
```

```
# partial regression rank coefficients

senresult_rank = pcc(parm, yieldsd$maxyield, rank=TRUE )
senresult_rank
```

```
##
## Call:
## pcc(X = parm, y = yieldsd$maxyield, rank = TRUE)
##
## Partial Rank Correlation Coefficients (PRCC):
##               original
## Tmincoeff1  0.77176139
## Tmincoeff2  0.84679080
## Pcoeff1    -0.11701497
## Pcoeff2    -0.02183301
## intercep    0.98205207
```

```
plot(senresult_rank)
```



Try again for minimum yield

```
# combine parameter sets with output


# simple correlation coefficients
result = map(parm, cor.test, y=yieldsd$minyield)
result$Tmincoeff1
```

```
##
##  Pearson's product-moment correlation
##
## data:  .x[[i]] and yieldsd$minyield
## t = -0.50348, df = 98, p-value = 0.6158
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.2447696  0.1470921
## sample estimates:
##         cor
## -0.05079351
```

```
result$Tmincoeff2
```

```
##
##  Pearson's product-moment correlation
##
## data:  .x[[i]] and yieldsd$minyield
## t = 1.1764, df = 98, p-value = 0.2423
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.08027224  0.30730177
## sample estimates:
##       cor
## 0.1180065
```

```
result$Pcoeff1
```

```
##
##  Pearson's product-moment correlation
##
## data:  .x[[i]] and yieldsd$minyield
```

```
## t = -0.24734, df = 98, p-value = 0.8052
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.2203150  0.1722856
## sample estimates:
##         cor
## -0.02497779
```

```
result$Pcoeff2
```

```
##
##  Pearson's product-moment correlation
##
## data:  .x[[i]] and yieldsd$minyield
## t = 412.48, df = 98, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.9995714 0.9998066
## sample estimates:
##       cor
## 0.9997121
```

```
result$intercep
```

```
##
##  Pearson's product-moment correlation
##
## data:  .x[[i]] and yieldsd$minyield
## t = -0.15835, df = 98, p-value = 0.8745
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.2117470  0.1809927
## sample estimates:
##         cor
## -0.01599406
```

```
# just the confidence interval
justconf = result %>% map_df("conf.int")
justconf
```

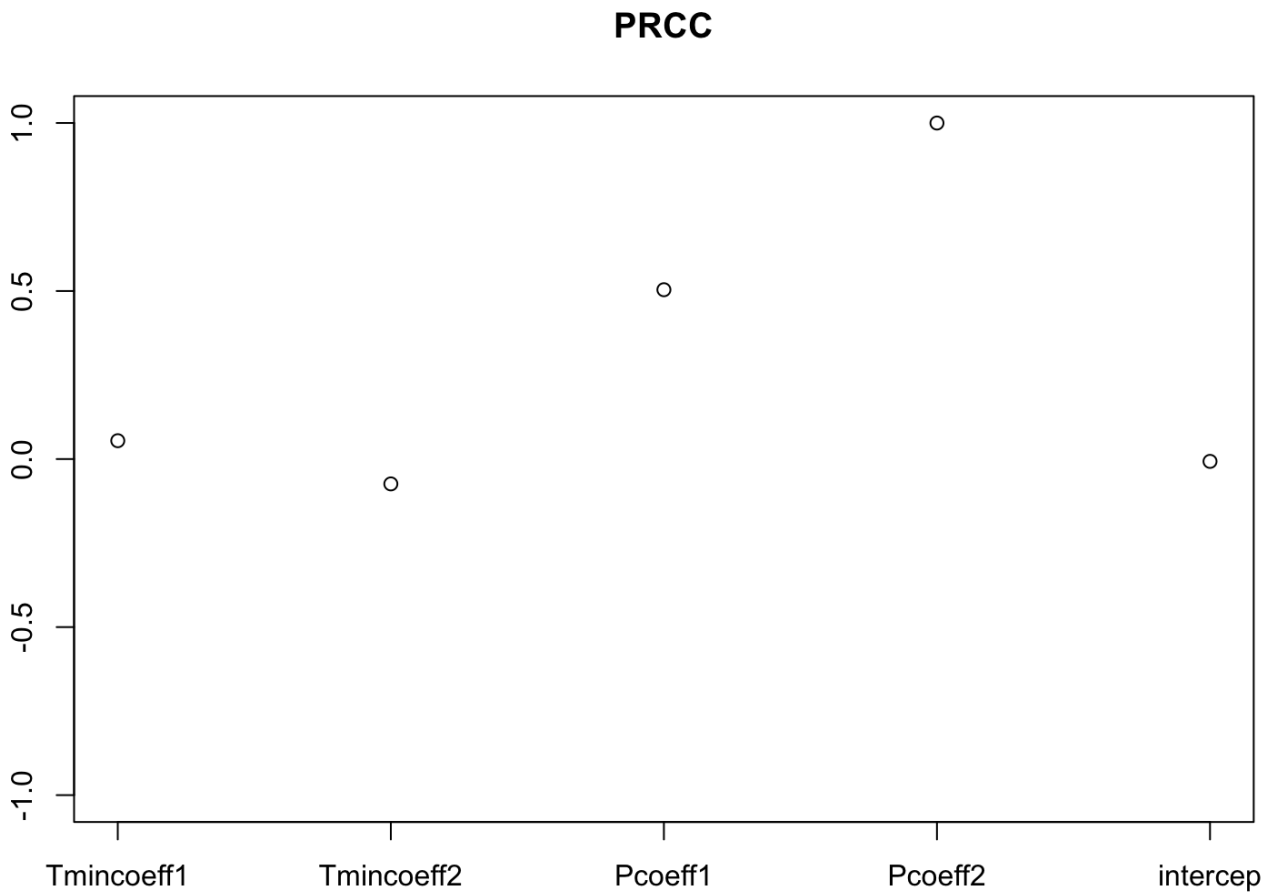
```
## # A tibble: 2 × 5
##   Tmincoeff1 Tmincoeff2 Pcoeff1 Pcoeff2 intercep
##        <dbl>      <dbl>   <dbl>   <dbl>    <dbl>
## 1     -0.245    -0.0803  -0.220    1.00   -0.212
## 2      0.147     0.307    0.172    1.00    0.181
```

```r
# try again using rank coefficients

senresult_rank = pcc(parm, yieldsd$minyield, rank=TRUE )
senresult_rank
```

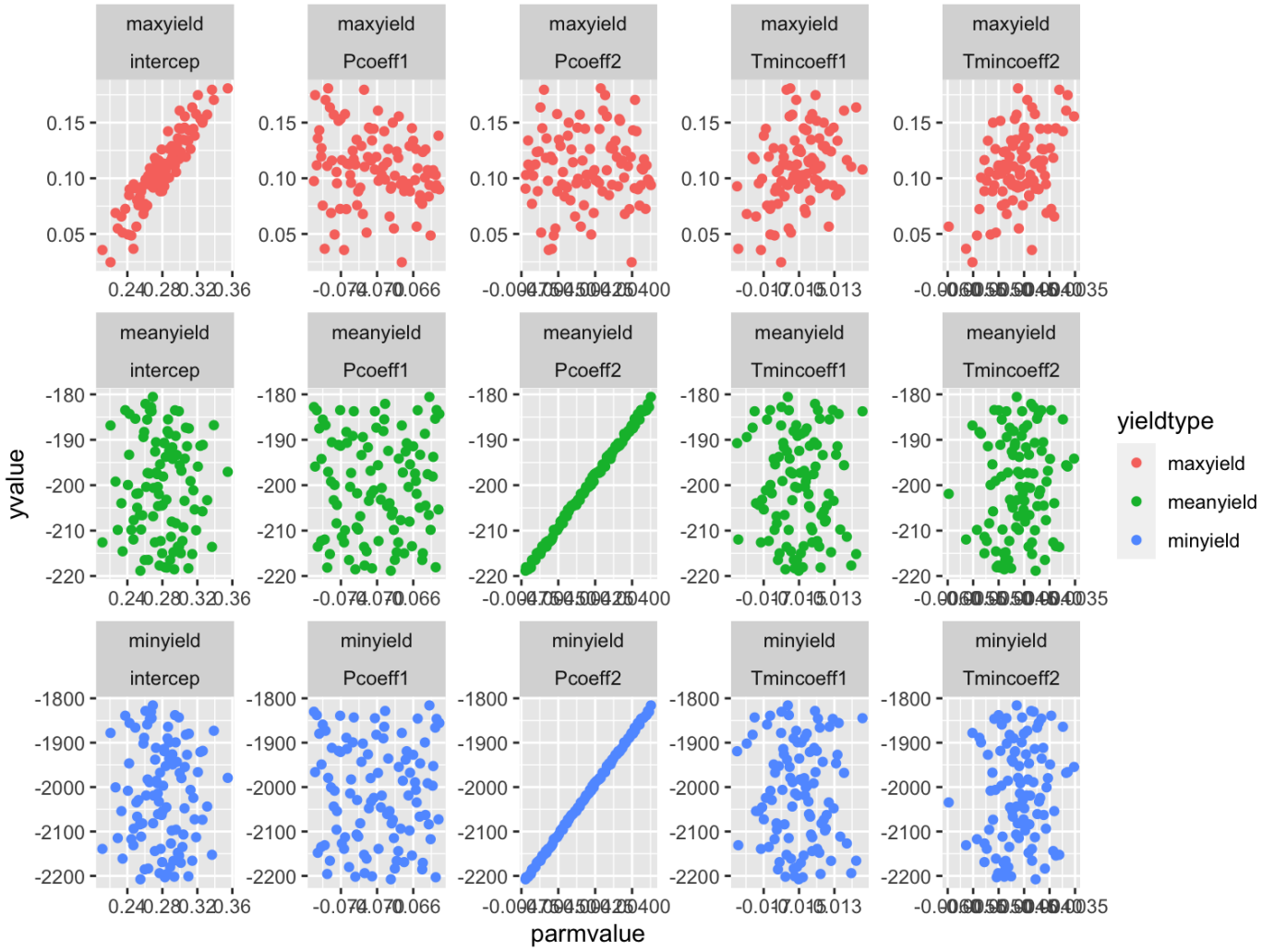
```
##
## Call:
## pcc(X = parm, y = yieldsd$minyield, rank = TRUE)
##
## Partial Rank Correlation Coefficients (PRCC):
##                original
## Tmincoeff1  0.054577499
## Tmincoeff2 -0.073950217
## Pcoeff1     0.503826237
## Pcoeff2     0.999842547
## intercep   -0.006924748
```

```r
plot(senresult_rank)
```



plot parameter sensitivity

```
# a bit tricky but nice way to make it easy to plot all parameters against all values
tmp = cbind.data.frame(yieldsd, parm)
tmp2 = tmp %>% gather(maxyield, minyield, meanyield, value="yvalue",key="yieldtype")
tmp3 = tmp2 %>% gather(-yvalue, -yieldtype, key="parm", value="parmvalue")
ggplot(tmp3, aes(parmvalue, yvalue,
        col=yieldtype))+geom_point()+facet_wrap(~yieldtype*parm, scales="free",
        ncol=5)
```



Other Formal Sensitivity Approaches

Sobol method

Fourier Amplitude Sensitivity Test

- more efficient parameter space sampling
- estimates of parameter contributions that can be more informative

Larger Complex Models

Only do sensitivity on some inputs/parameters

Use Latin Hypercube to generate samples

Run model for samples

Generate Summary Statistics and graph

More Information on Sensitivity Analysis

[Applying Sensitivity Analysis in Biology] (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2570191/)

Nice paper on sensitivity analysis for environmental modeling Click to Download

Saltelli, Andrea, and Paola Annoni. "How to avoid a perfunctory sensitivity analysis." Environmental Modelling & Software 25, no. 12 (2010): 1508-1517

Steps in Sensitivity Analysis

Define the distribution of input parameters (e.g normal with its mean and standard deviation)

Define the outputs to be considered (e.g if your output is time series are you looking at daily, max, min, annual ?)

Sample from pdf of input parameters - we use *randomLHS* for this

Run the model for each sample

Graph the results

Quantify sensitivity * we used *pcc*

In class : Repeat for a Different Example

Often when we are estimating vegetation or crop water use we need to know the atmospheric conductance - which is essentially how easily water diffuses into the air and depends largely on windspeed (you get more evaporation in windier conditions) Atmospheric conductance is also influenced by the vegetation itself and the turbulence it creates

I've provided a function to compute atmospheric conductance C_{at} (how easily vapor diffuses from vegetation surfaces)

The function *Catm.R* is provided

Atmospheric conductance (simple model)

So that you know what it does - here's some background on the function

$$C_{at} = \frac{v_m}{6.25 * \ln(\frac{z_m - z_d}{z_0})^2}$$

$$z_d = k_d * h$$

$$z_0 = k_0 * h$$

z_m is the height at which windspeed is measured - must be higher than the vegetation (cm), it is usually measured 200 cm above the vegetation

h is vegetation height (cm)

v is windspeed (cm/s)

Typical values if k_d and k_o are 0.7 and 0.1 respectively (so use those as defaults)

Your task

For a given forest, perform a sensitivity analysis of model predictions of conductance Consider the sensitivity of your estimate to uncertainty in the following parameters and inputs

- height
- k_d
- k_0
- v

Windspeeds v are normally distributed with a mean of 250 cm/s with a standard deviation of 30 cm/s

For vegetation height assume that height is somewhere between 9.5 and 10.5 m (but any value in that range is equally likely)

For the k_d and k_0 parameters you can assume that they are normally distributed with standard deviation of 1% of their default values

1. Use the Latin hypercube approach to generate parameter values for the 4 parameters

2. Run the atmospheric conductance model for these parameters
3. Plot conductance estimates in a way that accounts for parameter uncertainty
4. Plot conductance estimates against each of your parameters
5. Estimate the Partial Rank Correlation Coefficients

Submit partial rank correlation coefficience to discussion