SENSITIVITY ANALYSIS

WHAT YOU CAN KNOW ABOUT SENSITIVITY BEFORE INFORMAL/FORMAL SENSITIVITY ANALYSIS

- Parameter interactions are *evident* in the equations

Recall our reservoir model

- **Input**: Reservoir height and flow rate
- **Output**: Instantaneous power generation (W/s)
- **Parameters**: K (efficiency) , ρ (density of water), g (acceleration due to gravity)

P = ρ * h * r * g * K

- P is Power in watts,
- ρ is the density of water (~1000 kg/m3)
- h is height in meters,
- r is flow rate in cubic meters per second,
- g is acceleration due to gravity of 9.8 m/s2,
- K Efficiency is a coefficient of efficiency ranging from 0 to 1.

The “effect” of **K** (efficiency) will depend on the other parameters
A 1% change in **K** will lead to a 1% change in Power
How much Power (magnitude) will change with an increase **K** will depend on value of other parameters and inputs
So you could say that Power is more sensitive to efficiency for larger flow rates or larger heights, We know this because the are multiplied together!

SOME CODE TO CONVINCE YOU OF THIS

```
# lets look at the power generation function
source(here("R/power_gen.R"))
Power_K1 <- power_gen(height = 2, flow = 4, K = 0.1, g = 9.8)
Power_K1
```

```
[1] 7840
```

```
Power_K2 <- power_gen(height = 2, flow = 4, K = 0.2, g = 9.8)
Power_K2
```

```
[1] 15680
```

```
# if you double K, you will double power (linear response)
# relative change in power
0.1 * 2
```

```
[1] 0.2
```

```
Power_K1 * 2
```

```
[1] 15680
```

```
Power_K2 / Power_K1
```

```
[1] 2
```

Because the equation is just multiplying factors

- Relative change is the same as the change in efficiency (K)
- Absolute change in power with change in efficiency depends on other parameters

So what happens to sensitivity if height is 3 instead of 2

```
# absolute change in power depends on other parameters/inputs
# absolute change in pawer or magnitude of the change in power with
Power_K2 - Power_K1
```

```
[1] 7840
```

```
# if height is 3 instead of 2
Power_K1 <- power_gen(height = 3, flow = 4, K = 0.1, g = 9.8)
Power_K1
```

```
[1] 11760
```

```
Power_K2 <- power_gen(height = 3, flow = 4, K = 0.2, g = 9.8)
Power_K2
```

```
[1] 23520
```

```
# Power still doubles (relative change is the same)
Power_K1 * 2
```

```
[1] 23520
```

```
# But absolute change is greater
Power_K2 - Power_K1
```

```
[1] 11760
```

WHAT YOU CAN KNOW ABOUT SENSITIVITY BEFORE "SENSITIVITY ANALYSIS"

- when transfer function is linear (equation *tells* you what the sensitivity is)
- Imagine that habitat suitability for parrot can be modelled with the following equations, where

$$PS = k_P * P + k_T * T$$

- **PS** probability that parrots will be found in a given location
- **P** is mean annual precipitation
- **T** is mean annual temperature
- k_P and k_T are empirically derived coefficients

Sensitivity to k_P and k_T will be linear
Which coefficient will matter more?
IF **P** and **T** where in the same units (e.g percent deviation from a mean value), then you could use the coefficients to tell you if parrot suitability would be more sensitive to a 10% change in precipitation versus a 10% change in temperature

WHAT YOU CAN NOT KNOW ABOUT SENSITIVITY BEFORE "SENSITIVITY ANALYSIS"

- Looking at the form of the equations in the box (transfer function) often tells you something about sensitivity
- BUT if you don't know what's in the box (transfer function) or if it has multiple boxes its difficult to tell
- any equation/transfer function with thresholds (if's, max, min's) hard to tell, sometimes parameters may matter sometimes they will not
- the first derivative of the equation with respect to the parameter IS the sensitivity; so if you can take the derivative you can *know* something about the sensitivity

Type text here

FORMAL 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

Type text here

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

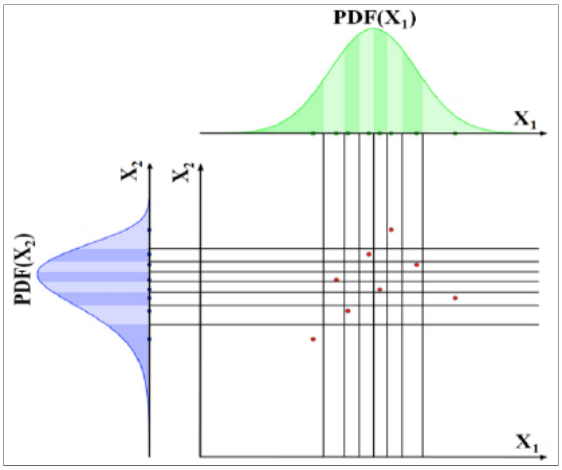
APPROACHES TO SAMPLING PARAMETER SPACE

- 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 what is their probability density funciton (i.e. normal, uniform)

  “qnorm”,”qunif”, “qlnorm”,”qgamma” many others]
  More Distributions and Parameters

STEPS FOR LATIN HYPERCUBE SAMPLING

- create a *lhs* matrix (essentially our hyper cube) which defines quantiles for each parameter
  - 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 i
# several summary statistics - how about mean, max and min yield
source(here("R/compute_almond_yield.R"))


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

# which parameters
pnames <- c("Tmincoeff1", "Tmincoeff2", "Pcoeff1", "Pcoeff2", "inte

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

# create our latin hyper cube of quantiles for each parameter
parm_quant <- randomLHS(nsample, npar)
colnames(parm_quant) <- pnames
```

```
   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

```
1 #
2 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(here("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 n

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

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

```
[1] -2006.212

[[1]]$meanyield
[1] -199.0907


[[2]]
[[2]]$maxyield
[1] 0.1124944

[[2]]$minyield
[1] -2180.083

[[2]]$meanyield
[1] -216.4686
```

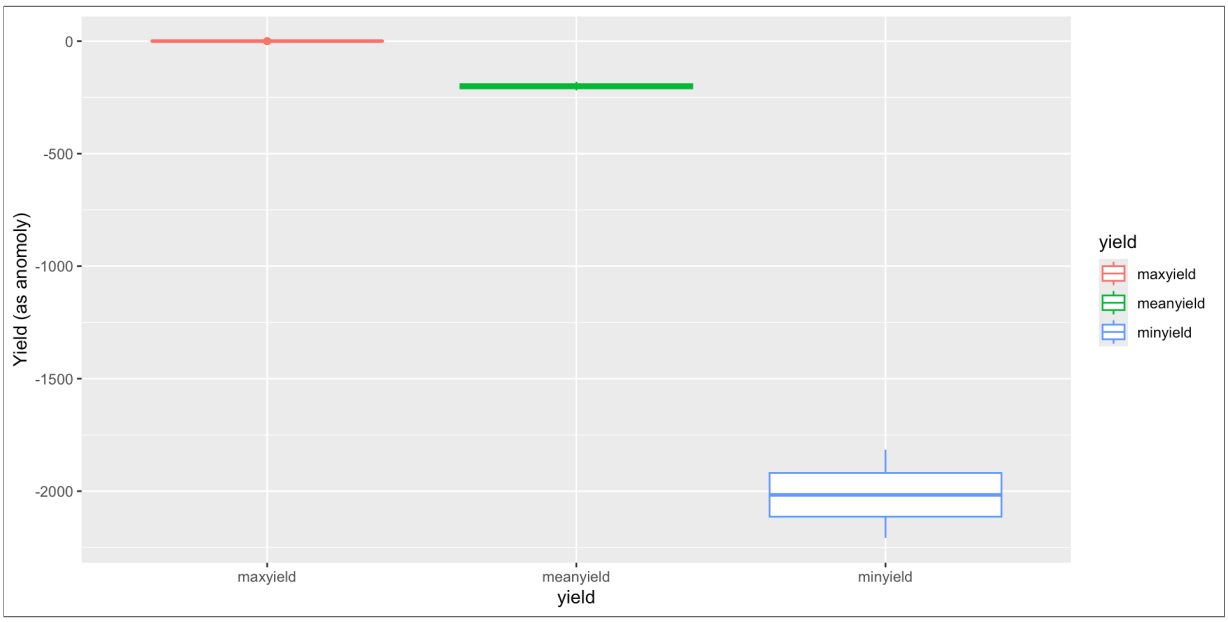
```
# turn results in to a dataframe for easy display/analysis
yieldsd <- yields %>% map_dfr(`[`, c("maxyield", "minyield", "meany
```

PLOTTING

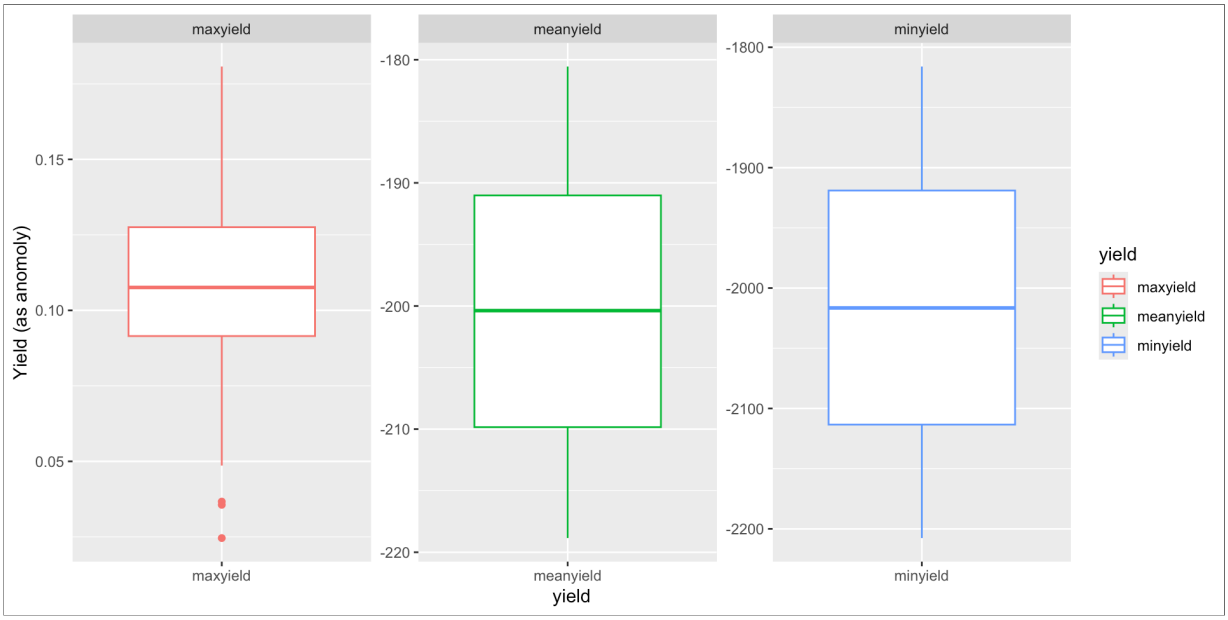
Plot relationship between parameter and output to understand how uncertainty in parameter impacts the output to determine over what ranges of the parameter uncertainty is most important (biggest effect)

- 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

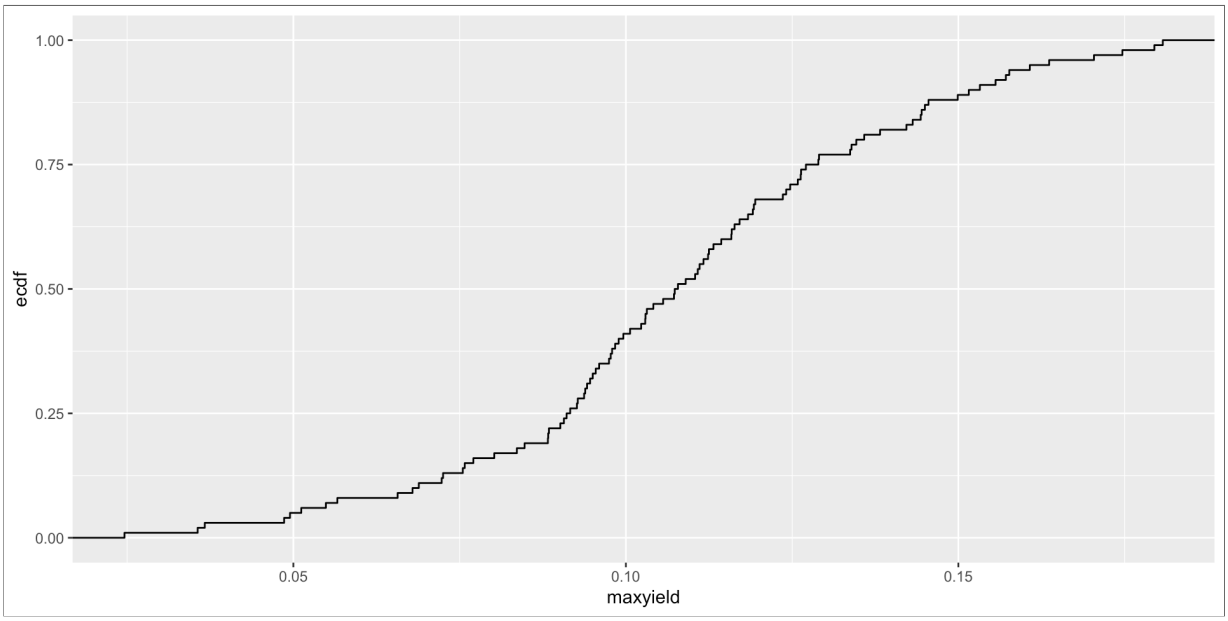
```
# add uncertainty bounds on our estimates
tmp <- yieldsd %>% gather(value = "value", key = "yield")
ggplot(tmp, aes(yield, value, col = yield)) +
  geom_boxplot() +
  labs(y = "Yield (as anomoly)")
```



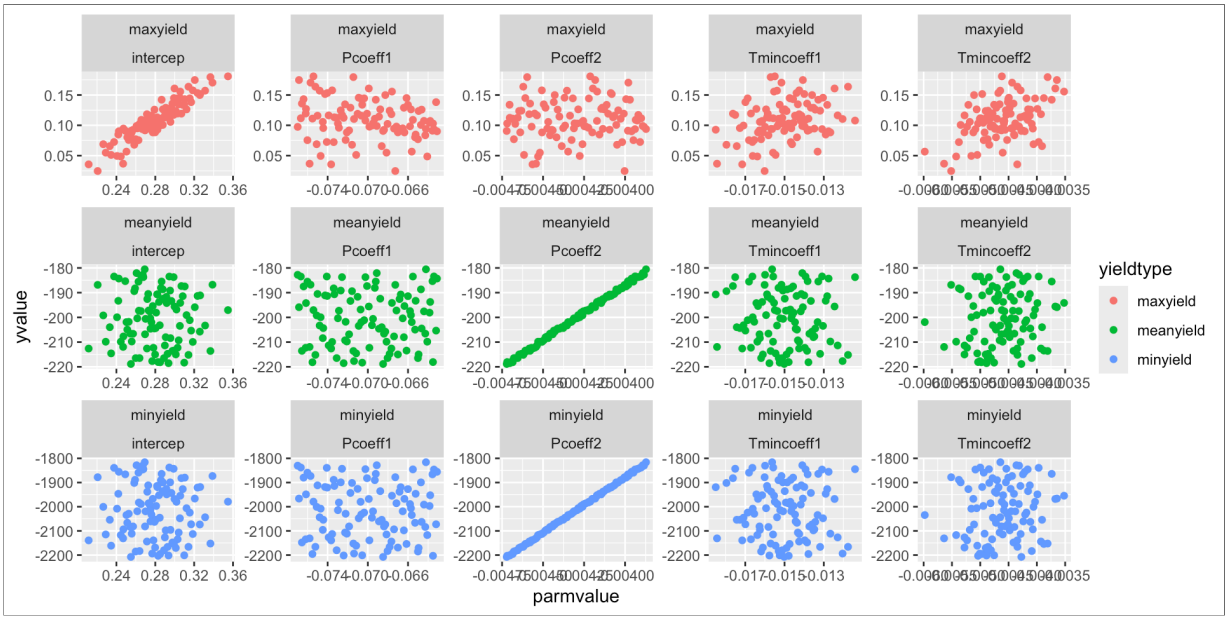

```
# note that you don't see the ranges because of the scale (min yiel
ggplot(tmp, aes(yield, value, col = yield)) +
  geom_boxplot() +
  labs(y = "Yield (as anomoly)") +
  facet_wrap(~yield, scales = "free")
```




```
# 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
tmp <- cbind.data.frame(yieldsd, parm)
tmp2 <- tmp %>% gather(maxyield, minyield, meanyield, value = "yval
tmp3 <- tmp2 %>% gather(-yvalue, -yieldtype, key = "parm", value =
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

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

If we have multiple parameters and inputs

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 accounting for Z (all other parameters)
  - Computed as the correlation of the *residuals* from linear regression of
    - X with Z
    - Y with Z `
- for *rank* correlation use ranks instead of values
  ```

QUANTIFYING SENSITIVITY

- Linear and monotonic relationship between x and y
- Correlation Coefficient - linear monotonic monotonic = variables are all going in the same direction (i.e. x and y are both going in the same direciton
- Non-linear relationship, but monotonic between x and y
  - you transform x and y (e.g log transform)
  - 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 FOR MAXIMUM YIELD

```
# 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$Pcoeff10
```

```
NULL
```

```
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
```

```
# extract just the correlation from results list
justR2 <- result %>% map_df("estimate")
justR2$pname = names(parm)
justR2
```

```
# A tibble: 5 × 2
      cor pname
    <dbl> <chr>
1  0.330  Tmincoeff1
2  0.422  Tmincoeff2
3 -0.208  Pcoeff1
4  0.0170 Pcoeff2
5  0.920  intercep
```

```
# extract just the confidence interval from results list
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
```

PARITAL RANK CORRELATION COEFFICIENTS FOR MAXIMUM YIELD

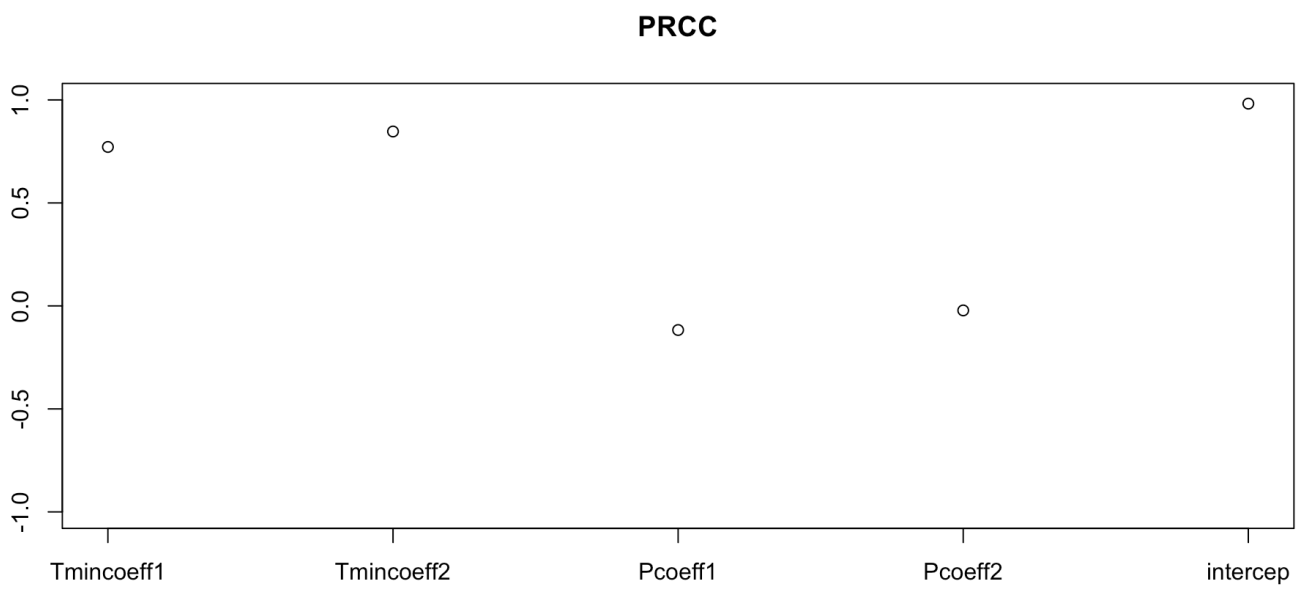
we can use *pcc* to compute these for us

```
# 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 RESULTS

```
# plot results
plot(senresult_rank)
```



TRY ON YOUR OWN

- Partial Rank Correlation Coefficients for Minimum Yield
- how are they different from results for maximum yield

CODE

```r
# 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
```

```r
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
```

```r
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
```

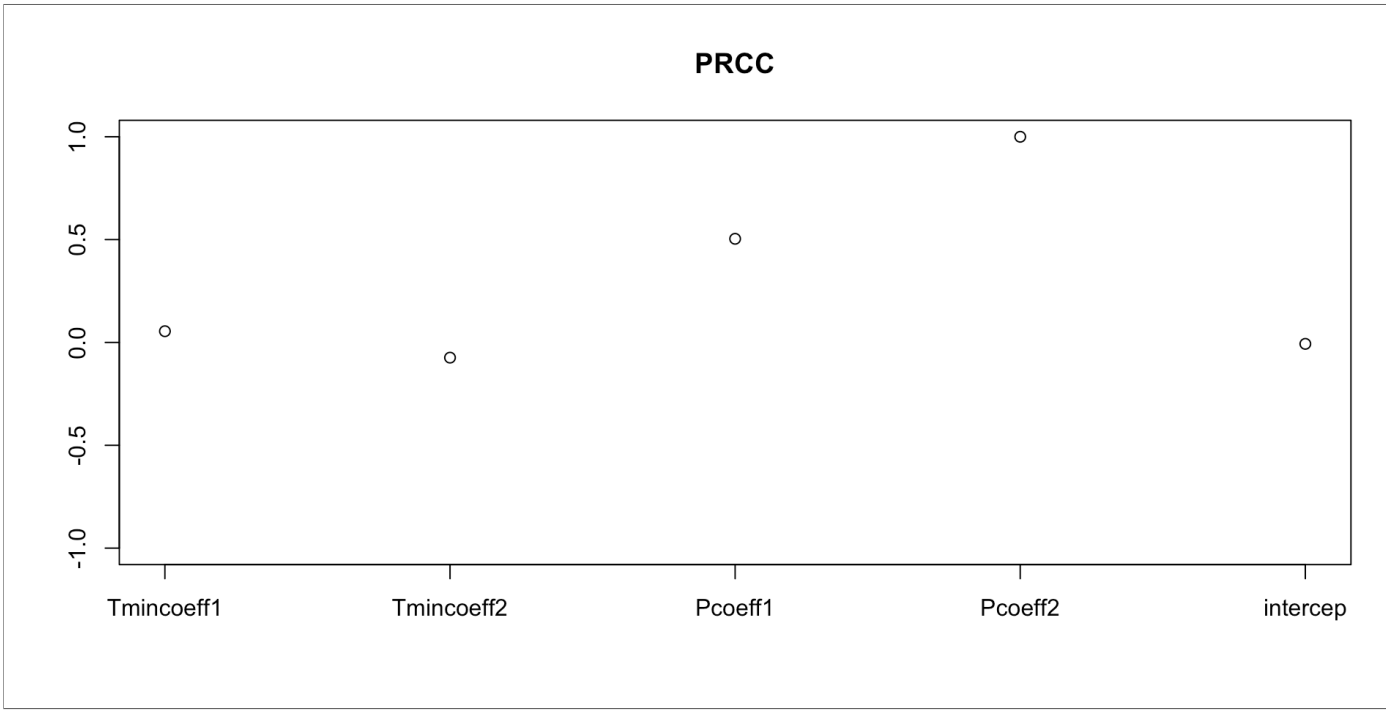
```
# A tibble: 2 × 5
  Tmincoeff1 Tmincoeff2 Pcoeff1 Pcoeff2 intercep
       <dbl>      <dbl>   <dbl>   <dbl>    <dbl>
1     -0.245    -0.0803  -0.220    1.00   -0.212
2      0.147     0.307    0.172    1.00    0.181
```

```
# 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
```

```
1 plot(senresult_rank)
```



PLOT PARAMETER SENSITIVITY

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