skpr

R Package for Generation and Evaluation of Experimental Designs

Tyler Morgan-Wall

Outline

1. Demo
2. Introduction
3. Features of skpr
4. skprGUI

Live Demo

Outline

1. Demo
2. Introduction
3. Features of skpr
4. skprGUI

Motivation for skpr

- Create an open-source design of experiments package to allow for the creation and analysis of designs **all within R**.
- Implement optimal design algorithms using modern linear algebra libraries to make the underlying code **simple, reliable, and auditable**.
- Integrate Monte Carlo power evaluation as a standard feature and make it **flexible, extensible, and easy to use**.

Preparation matching execution

Plan your experiment using the same models and tools that you plan to use when analyzing them.

Shareable and repeatable

With skpr, there is never a question about *how* you ran an analysis—it's all there in the script.

Reproducing your work is as simple as re-running the code.

Outline

1. Demo
2. Introduction
3. Features of skpr
4. skprGUI

Five main functions + plotting

- **gen_design:** Optimal design generation
- **eval_design:** Parametric power evaluation
- **eval_design_mc:** Monte Carlo power evaluation
- **eval_design_survival_mc:** Censored Monte Carlo power evaluation
- **eval_design_custom_mc:** Custom library Monte Carlo power evaluation
- **plot_correlations & plot_fds:** Plot correlation map and fraction of design space plots

Five main functions + plotting

- **gen_design:** Optimal design generation
- **eval_design:** Parametric power evaluation
- **eval_design_mc:** Monte Carlo power evaluation
- **eval_design_survival_mc:** Censored Monte Carlo power evaluation
- **eval_design_custom_mc:** Custom library Monte Carlo power evaluation
- **plot_correlations & plot_fds:** Plot correlation map and fraction of design space plots

Workflow in skpr

Step 1 → Step 2 → Step 3

Step 1	Step 2	Step 3
Generate test points	gen_design	eval_design
		eval_design_mc
		eval_design_survival_mc
		eval_design_custom_mc

Example: How much caffeine extracted from coffee?

Factors:

Bean size: Large, Small

Bean type: Kona, Java

Water temperature: 80˚C, 90˚C, 100˚C

Roast Darkness (hard to change): Light, Medium, Dark

Main functions in skpr

- **gen_design:** Optimal design generation
- **eval_design:** Parametric power evaluation
- **eval_design_mc:** Monte Carlo power evaluation

gen_design – Basic Functionality

Candidate Set

size	type	temp
Large	Kona	80
Large	Kona	90
Large	Kona	100
Large	Java	80
Large	Java	90
Large	Java	100
Small	Kona	80
Small	Kona	90
Small	Kona	100
Small	Java	80
Small	Java	90
Small	Java	100

`coffeepoints`

Model

`Y ~ size + type + temp`

Trials

`10`

Inputs

```
gen_design(candidateset = coffeepoints,
           model = ~size + type + temp,
           trials = 10)
```

Output

Default is D-Optimal

Run	Large	Kona	Temp
1	Large	Java	80
2	Small	Java	100
3	Small	Kona	80
4	Large	Kona	100
5	Small	Kona	100
6	Small	Java	80
7	Large	Kona	80
8	Small	Java	80
9	Large	Kona	80
10	Large	Java	100

Creation of the candidate set is simple

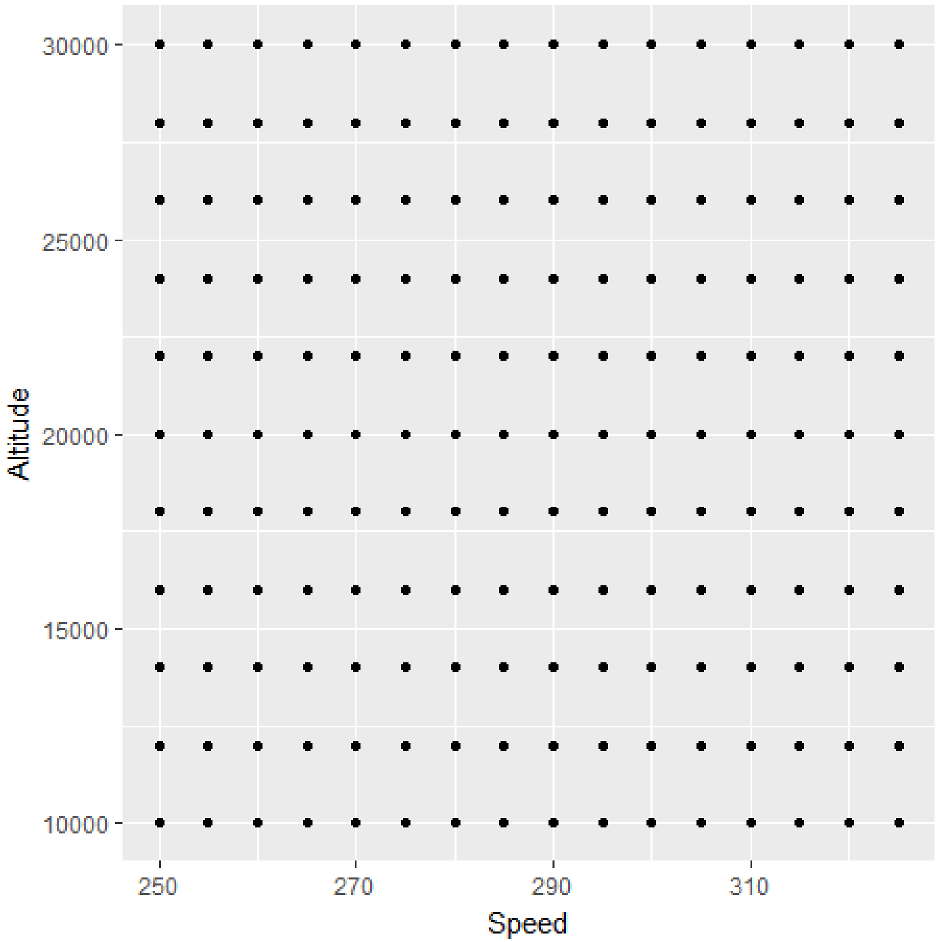
expand.grid function in R automatically generates all combinations of input parameters

```
> expand.grid(size=c("Large","Small"),type=c("Java","Kona"),temp=c(80,90,100))
    size type temp
1  Large Java   80
2  Small Java   80
3  Large Kona   80
4  Small Kona   80
5  Large Java   90
6  Small Java   90
7  Large Kona   90
8  Small Kona   90
9  Large Java  100
10 Small Java  100
11 Large Kona  100
12 Small Kona  100
```

Easily constrain your design space

Simple filtering of disallowed combinations to constrain the design space.

Example:
Testing within a flight envelope

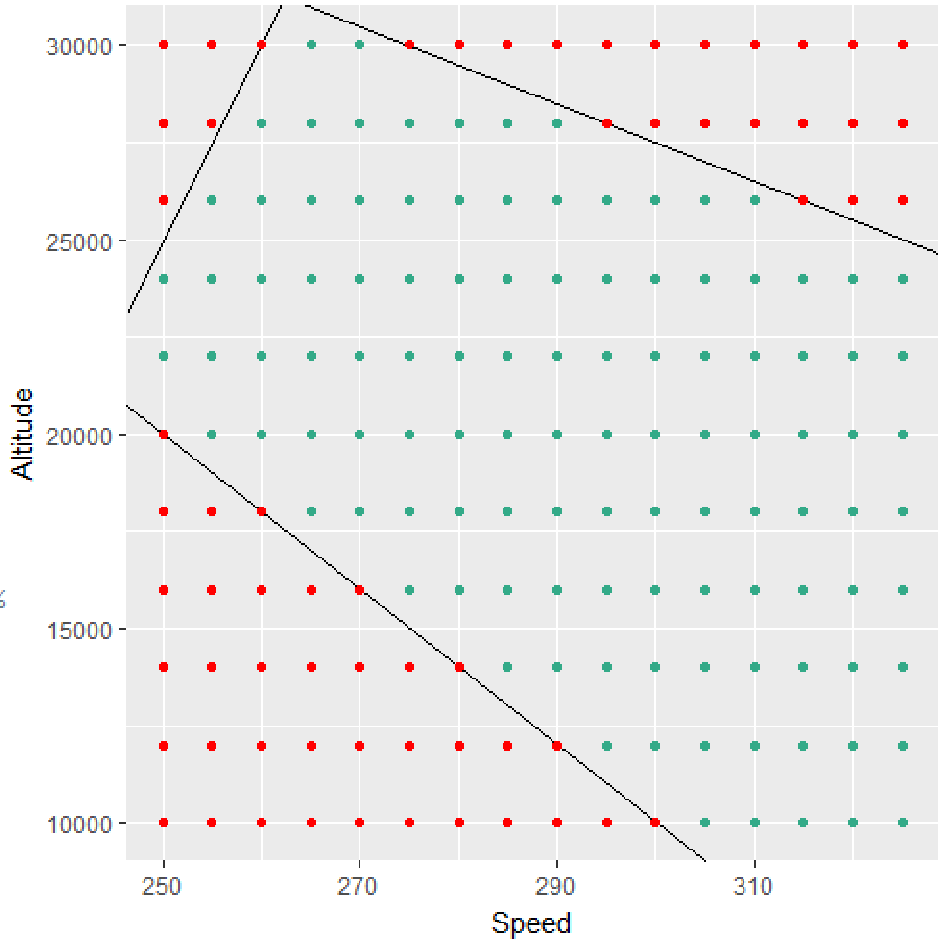


Easily constrain your design space

Simple filtering of disallowed combinations to constrain the design space.

Filter out unwanted test points

```
filteredcandidateset = candidateset %>%
  filter(70000 - speed * 200  < altitude) %>%
  filter(57500 - speed * 100 > altitude) %>%
  filter(-100000 + speed * 500 > altitude)
```

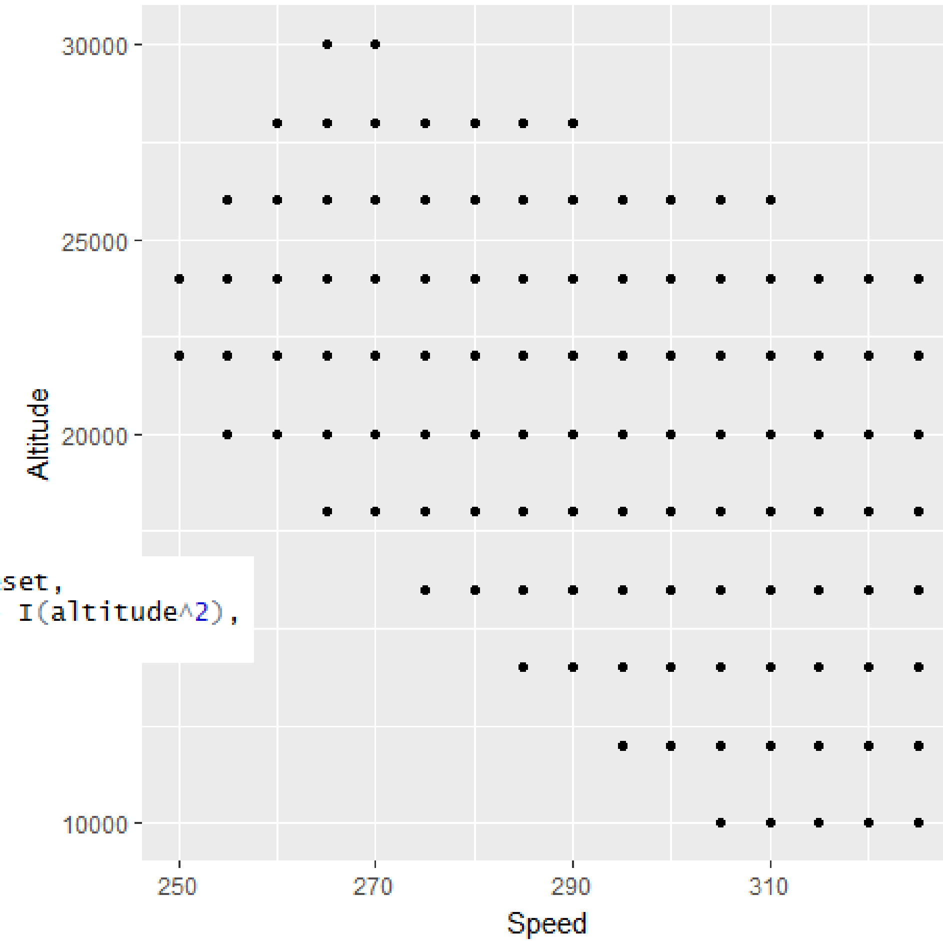


Easily constrain your design space

Simple filtering of disallowed combinations to constrain the design space.

Use new candidate set to generate design

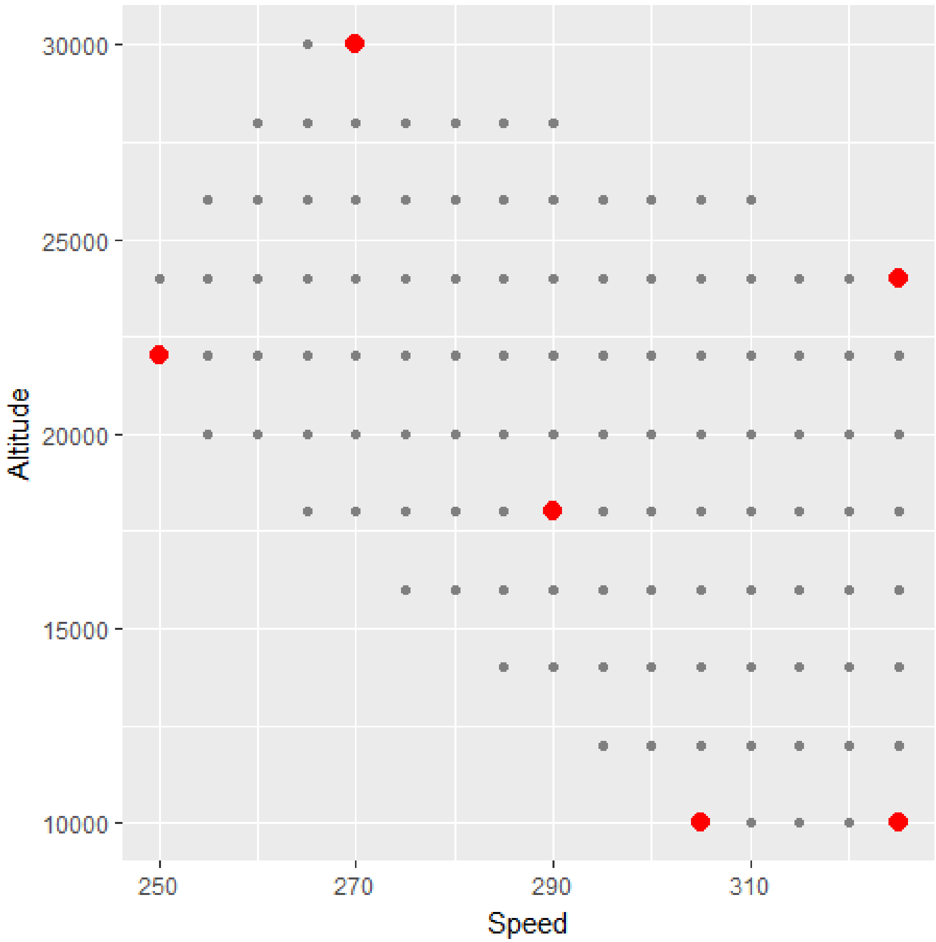
```
gen_design(candidateset = filteredcandidateset,
           ~speed + altitude + I(speed^2) + I(altitude^2),
           trials = 18)
```



Easily constrain your design space

Simple filtering of disallowed combinations to constrain the design space.

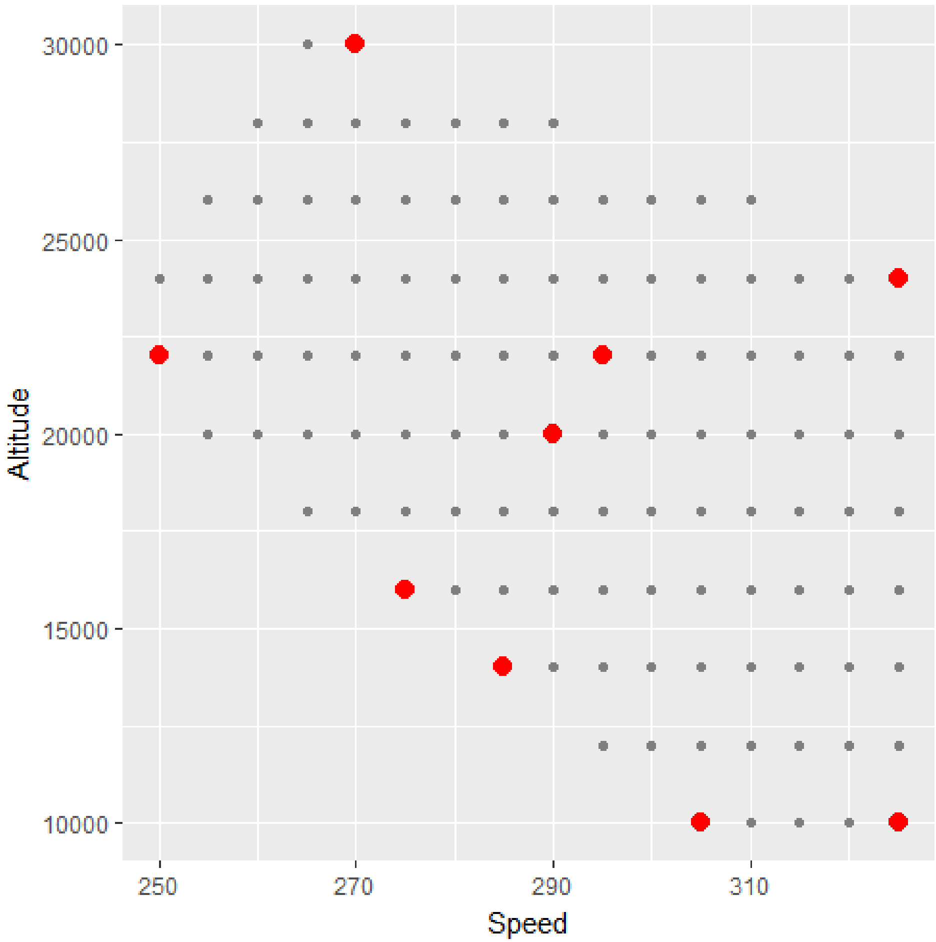
Use new candidate set to generate design



Easily constrain your design space

Simple filtering of disallowed combinations to constrain the design space.

Explore different options (e.g. I-optimal design)



Sequential construction of split-plot designs

```r
candidatelist = expand.grid(size=as.factor(c("Large","Small")),
            type=as.factor(c("Kona","Java")),
            temp=c(80,100),
            roastdarkness=c("Light","Medium","Dark"))

gen_design(candidateset = candidatelist, ~roastdarkness, trials=6) -> htcdesign

gen_design(candidateset = candidatelist, ~size+temp+type, trials=18,
           splitplotdesign = htcdesign, splitplotsizes=3)
```

```
  roastdarkness
1          Dark
2         Light
3          Dark
4        Medium
5         Light
6        Medium
```

3
3
3
3
3
3

```
    roastdarkness  size temp type
1.1          Dark Small  100 Java
1.2          Dark Large   80 Kona
1.3          Dark Large  100 Kona
2.1         Light Small   80 Kona
2.2         Light Large  100 Kona
2.3         Light Large  100 Java
3.1          Dark Small   80 Kona
3.2          Dark Small   80 Java
3.3          Dark Large  100 Java
4.1        Medium Large   80 Java
4.2        Medium Small  100 Kona
4.3        Medium Small   80 Java
5.1         Light Large   80 Kona
5.2         Light Small   80 Java
5.3         Light Small  100 Java
6.1        Medium Large   80 Java
6.2        Medium Large  100 Kona
6.3        Medium Small  100 Kona
```

Supports N-depth split-plot designs

```
gen_design(candidateset = candidatelist, ~roastdarkness, trials=4) -> vhtcdesign

gen_design(candidateset = candidatelist, ~type, trials=8,
           splitplotdesign = vhtcdesign, splitplotsizes = 2) -> htcdesign

gen_design(candidateset = candidatelist, ~size+temp, trials=24,
           splitplotdesign = htcdesign,  splitplotsizes = 3)
```

```
      roastdarkness type  size temp
1.1.1          Dark Kona Large  100
1.1.2          Dark Kona Small   80
1.1.3          Dark Kona Small  100
1.2.1          Dark Java Large   80
1.2.2          Dark Java Small  100
1.2.3          Dark Java Large   80
2.1.1        Medium Java Small   80
2.1.2        Medium Java Small   80
2.1.3        Medium Java Large  100
2.2.1        Medium Kona Small  100
2.2.2        Medium Kona Large   80
2.2.3        Medium Kona Large  100
3.1.1         Light Kona Small   80
3.1.2         Light Kona Large   80
3.1.3         Light Kona Large  100
3.2.1         Light Java Large   80
3.2.2         Light Java Small  100
3.2.3         Light Java Small  100
4.1.1         Light Kona Large  100
4.1.2         Light Kona Small   80
4.1.3         Light Kona Small   80
4.2.1         Light Java Small  100
4.2.2         Light Java Large   80
4.2.3         Light Java Large  100
```

Main functions in skpr

- **gen_design:** Optimal design generation
- **eval_design:** Parametric power evaluation
- **eval_design_mc:** Monte Carlo power evaluation

eval_design – Basic Functionality

Design

Size	Type	Temp
Medium	Kona	100
Large	Java	100
Large	Java	80
Small	Kona	80
Small	Java	80
Medium	Java	80
Large	Kona	100
Medium	Java	100
Large	Kona	80
Small	Java	100

`coffeedesign`

Model

`Y ~ size + type + temp`

Alpha

`0.2`

Inputs

```
eval_design(RunMatrix = coffeedesign,
            model = ~size + type + temp,
            alpha = 0.2)
```

Output

```
  parameters            type     power
(Intercept)     effect.power 0.9329381
       size     effect.power 0.7439050
       type     effect.power 0.9329381
       temp     effect.power 0.9329381
(Intercept) parameter.power 0.9329381
      size1 parameter.power 0.7910980
      size2 parameter.power 0.7389917
      type1 parameter.power 0.9329381
       temp parameter.power 0.9329381
```

Change effect size

Design

Size	Type	Temp
Medium	Kona	100
Large	Java	100
Large	Java	80
Small	Kona	80
Small	Java	80
Medium	Java	80
Large	Kona	100
Medium	Java	100
Large	Kona	80
Small	Java	100

`coffeedesign`

Model

`Y ~ size + type + temp`

Alpha

`0.2`

Effect size

`1.2`

Inputs

```
eval_design(RunMatrix = coffeedesign,
            model = ~size + type + temp,
            alpha = 0.2
            effectsize = 1.2)
```

Output

```
  parameters            type     power
(Intercept)     effect.power 0.6546176
       size     effect.power 0.4523370
       type     effect.power 0.6546176
       temp     effect.power 0.6546176
(Intercept)  parameter.power 0.6546176
      size1  parameter.power 0.4962631
      size2  parameter.power 0.4570906
      type1  parameter.power 0.6546176
       temp  parameter.power 0.6546176
```

Import and evaluate external designs

Simply import csv/excel file and skpr can evaluate it.

skpr automatically detects and converts JMP-generated split-plot designs.

```
> jmp_design <- data.frame(read_csv("U:/R/jmp_design.csv"))
Parsed with column specification:
cols(
  X1 = col_integer(),
  X2 = col_integer(),
  X3 = col_integer(),
  X4 = col_integer(),
  X5 = col_integer(),
  X6 = col_integer(),
  Y = col_character()
)
> jmp_design$Y = NULL
> jmp_design
   X1 X2 X3 X4 X5 X6
1  -1 -1  1  1  1  1
2   1  1 -1 -1 -1 -1
3  -1  1 -1  1  1 -1
4   1  1 -1  1  1  1
5  -1 -1 -1  1 -1  1
6   1 -1  1  1 -1 -1
7  -1 -1  1 -1 -1 -1
8   1 -1  1 -1 -1  1
9  -1  1 -1 -1  1  1
10  1 -1 -1 -1  1 -1
11 -1  1  1  1 -1  1
12  1  1  1 -1  1 -1
> eval_design(jmp_design,~.,0.2)
   parameters         type     power
1 (Intercept) effect.power 0.9689267
2          X1 effect.power 0.9445757
3          X2 effect.power 0.9445757
4          X3 effect.power 0.9445757
5          X4 effect.power 0.9445757
6          X5 effect.power 0.9445757
7          X6 effect.power 0.9445757
```

Main functions in skpr

- **gen_design:** Optimal design generation
- **eval_design:** Parametric power evaluation
- **eval_design_mc:** Monte Carlo power evaluation

IDA

No need to approximate when you can simulate

Calculating power for anything more complex than a non-blocked design with a normal response requires making approximations.

Those approximations tend to fail at low numbers of runs (where power matters most).

In many cases, no approximations exist.

eval_design_mc: Simple Monte Carlo power

Parametric

```
> expand.grid(size=as.factor(c("Large","Medium","Small")),
+             type=as.factor(c("Kona","Java")),
+             temp=c(80,100))         %>%
+   gen_design(~size+type+temp, 18)   %>%
+   eval_design(~size+type+temp, 0.2)
   parameters            type     power
1 (Intercept)    effect.power 0.9975907
2        size    effect.power 0.9521264
3        type    effect.power 0.9973936
4        temp    effect.power 0.9973936
5 (Intercept) parameter.power 0.9975907
6       size1 parameter.power 0.9473817
7       size2 parameter.power 0.9473817
8       type1 parameter.power 0.9973936
9        temp parameter.power 0.9973936
```

Monte Carlo

```
> expand.grid(size=as.factor(c("Large","Medium","Small")),
+             type=as.factor(c("Kona","Java")),
+             temp=c(80,100))         %>%
+   gen_design(~size+type+temp, 18)   %>%
+   eval_design_mc(~size+type+temp, 0.2)
   parameters               type power
1 (Intercept) parameter.power.mc 0.998
2       size1 parameter.power.mc 0.943
3       size2 parameter.power.mc 0.949
4       type1 parameter.power.mc 0.999
5        temp parameter.power.mc 0.999
```

eval_design_mc – Basic Functionality

Design

Size	Type	Temp
Medium	Kona	100
Large	Java	100
Large	Java	80
Small	Kona	80
Small	Java	80
Medium	Java	80
Large	Kona	100
Medium	Java	100
Large	Kona	80
Small	Java	100

`coffeedesign`

Model

`Y ~ size + type + temp`

Alpha

`0.2`

nsim

`1000`

glmfamily

`gaussian`

Inputs

```
eval_design_mc(RunMatrix = coffeedesign,
          model = ~size + type + temp,
          alpha = 0.2,
          nsim = 1000,
          glmfamily = "gaussian")
```

Output

```
   parameters                type power
(Intercept) parameter.power.mc 0.998
      size1 parameter.power.mc 0.943
      size2 parameter.power.mc 0.949
      type1 parameter.power.mc 0.999
       temp parameter.power.mc 0.999
```

eval_design_mc – Four built-in GLM families

Design

Size	Type	Temp
Medium	Kona	100
Large	Java	100
Large	Java	80
Small	Kona	80
Small	Java	80
Medium	Java	80
Large	Kona	100
Medium	Java	100
Large	Kona	80
Small	Java	100

`coffeedesign`

Model

`Y ~ size + type + temp`

Alpha

`0.2`

Effect

`2`

glmfamily

`poisson`

Inputs

```
eval_design_mc(RunMatrix = coffeedesign,
               model = ~size + type + temp,
               alpha = 0.2,
               effectsize = 2,
               glmfamily = “poisson”)
```

Output

```
 parameters                  type power
(Intercept) parameter.power.mc 0.480
      size1 parameter.power.mc 0.650
      size2 parameter.power.mc 0.471
      type1 parameter.power.mc 0.797
       temp parameter.power.mc 0.814
```

gaussian, binomial, poisson, exponential

eval_design_mc – Easy binomial power calculations

Design
coffeedesign_50runs

Model
Y ~ size + type + temp

Alpha
0.2

glmfamily
binomial

Inputs

Binomial Probabilities
Range: 50%-80%

```
eval_design_mc(RunMatrix = coffeedesign_50runs,
          model = ~size + type + temp,
          alpha = 0.2,
          glmfamily = "binomial"
          effectsize = c(0.5,0.8))
```

Output

parameters	type	power
(Intercept)	parameter.power.mc	0.673
size1	parameter.power.mc	0.577
size2	parameter.power.mc	0.627
type1	parameter.power.mc	0.805
temp	parameter.power.mc	0.810

eval_design_mc – Split-plot power w/ REML

Design
`splitplot_coffee`

Model
`Y ~ size + type + temp + roastdarkness`

Alpha
`0.2`

Blocking
`TRUE`

Inputs

```
eval_design_mc(RunMatrix = splitplot_coffee,
               model = ~size + type + temp + roastdarkness,
               alpha = 0.2,
               blocking=TRUE)
```

Output

```
     parameters               type power
1   (Intercept) parameter.power.mc 0.781
2         size1 parameter.power.mc 0.990
3         type1 parameter.power.mc 0.999
4          temp parameter.power.mc 0.995
5 roastdarkness1 parameter.power.mc 0.612
6 roastdarkness2 parameter.power.mc 0.616
```

eval_design_mc - Non-normal split-plot power

Model

`Y ~ size + type + temp + roastdarkness`

Alpha

`0.2`

glmfamily

`binomial`

Blocking

`TRUE`

Design

`spcoffeedesign_50runs`

Binomial Probabilities

`Range: 50%-80%`

Inputs

```
eval_design_mc(RunMatrix = spcoffeedesign_50runs,
          model = ~size + type + temp + roastdarkness,
          alpha = 0.2,
          blocking = TRUE,
          glmfamily = "binomial"
          effectsize = c(0.5,0.8))
```

Output

	parameters	type	power
1	(Intercept)	parameter.power.mc	0.498
2	size1	parameter.power.mc	0.723
3	type1	parameter.power.mc	0.737
4	temp	parameter.power.mc	0.763
5	roastdarkness1	parameter.power.mc	0.464
6	roastdarkness2	parameter.power.mc	0.466

No approximate way to calculate this.

skprGUI

Summary & Questions

Quickly execute your entire design of experiments workflow in one short script. Seamlessly integrate advanced statistical techniques into your planning process. Know exactly how your results were calculated.

Extra Slides

gen_design – Change optimality criterion

Candidate Set

size	type	temp
Large	Kona	80
Large	Kona	90
Large	Kona	100
Large	Java	80
Large	Java	90
Large	Java	100
Small	Kona	80
Small	Kona	90
Small	Kona	100
Small	Java	80
Small	Java	90
Small	Java	100

`coffeepoints`

Model

`Y ~ size + type + temp + temp^2`

Trials

`10`

Criterion

`"I"`

Inputs

```
gen_design(candidateset = coffeepoints,
           model = ~size + type + temp + I(temp^2),
           trials = 10,
           optimality = "I")
```

Output

Run	Large	Kona	Temp
1	Large	Kona	90
2	Large	Java	90
3	Large	Kona	80
4	Large	Java	100
5	Small	Java	80
6	Large	Kona	90
7	Small	Kona	100
8	Small	Java	90
9	Small	Java	90
10	Small	Kona	90

Five main functions

- **gen_design:** Optimal design generation
- **eval_design:** Parametric power evaluation
- **eval_design_mc:** Monte Carlo power evaluation
- **eval_design_survival_mc:** Censored Monte Carlo power evaluation
- **eval_design_custom_mc:** Custom library Monte Carlo power evaluation

eval_design_survival_mc: Censored power calculations

Design
coffeedesign

Model
Y ~ size + type + temp

Alpha
0.2

distribution
exponential

Inputs

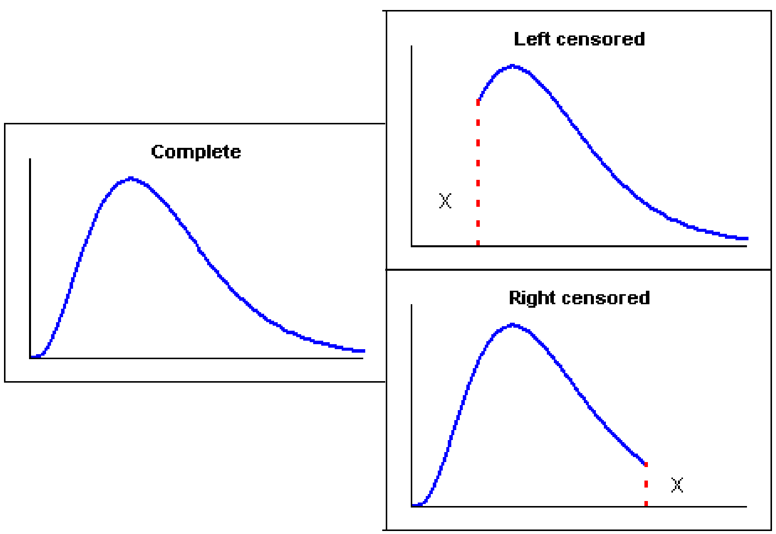
Censor Point
5

Censor Type
right

```
eval_design_survival_mc(RunMatrix = coffeedesign,
          model = ~size + type + temp,
          alpha = 0.2,
          censorpoint=5,
          censortype="right",
          distribution= "exponential")
```

Output

```
   parameters                type power
1 (Intercept) parameter.power.mc 0.930
2       size1 parameter.power.mc 0.702
3       size2 parameter.power.mc 0.267
4       type1 parameter.power.mc 0.780
5        temp parameter.power.mc 0.799
```



eval_design_custom_mc: Extensible framework

Recreate eval_design_survival_mc

```
rsurvival = function(X,b) {
  Y = rexp(n=nrow(X),rate=exp(-(X %*% b)))
  censored = Y > 1
  Y[censored] = 1
  return(Surv(time=Y,event=!censored,type="right"))
}

fitsurv = function(formula, X, contrastlist=NULL) {
  return(survreg(formula, data=X,dist="exponential"))
}

pvalsurv = function(fit) {
  return(summary(fit)$table[,4])
}

eval_design_custom_mc(RunMatrix=design,model=~a,alpha=0.05,nsim=100,
                      fitfunction=fitsurv, pvalfunction=pvalsurv, rfunction=rsurvival, delta=1)
```

Same output

```
> eval_design_custom_mc(RunMatrix=design,model=~a,alpha=0.05,nsim=10000,
+                       fitfunction=fitsurv, pvalfunction=pvalsurv, rfunction=rsurvival, delta=1)
   parameters               type  power
1 (Intercept) parameter.power.mc 0.9388
2           a parameter.power.mc 0.9222
> eval_design_survival_mc(RunMatrix=design,model=~a,alpha=0.05,nsim=10000,
+                       censortype="right",distribution="exponential",censorpoint=1,delta=1)
   parameters               type  power
1 (Intercept) parameter.power.mc 0.9351
2           a parameter.power.mc 0.9146
```

Type-I error increase in non-normal split-plot designs

Model

`Y ~ size + type + temp + roastdarkness`

Alpha

`0.2`

glmfamily

`binomial`

Design

`spcoffeedesign_50runs`

`10 blocks of 5 runs`

Inputs

Binomial Probabilities

`Range: 50%-80%`

```
eval_design_mc(RunMatrix = spcoffeedesign_50runs,
               model = ~size + type + temp + roastdarkness,
               alpha = 0.2,
               glmfamily = "binomial"
               binomialprobs = c(0.5,0.8))
```

Output

	parameters	type	power
1	(Intercept)	parameter.power.mc	0.498
2	size1	parameter.power.mc	0.723
3	type1	parameter.power.mc	0.737
4	temp	parameter.power.mc	0.763
5	roastdarkness1	parameter.power.mc	0.464
6	roastdarkness2	parameter.power.mc	0.466

Type-I error no blocking

	parameters	type	power
1	(Intercept)	parameter.power.mc	0.206
2	size1	parameter.power.mc	0.199
3	type1	parameter.power.mc	0.216
4	temp	parameter.power.mc	0.191
5	roastdarkness1	parameter.power.mc	0.187
6	roastdarkness2	parameter.power.mc	0.205

Type-I error blocking

	parameters	type	power
1	(Intercept)	parameter.power.mc	0.317
2	size1	parameter.power.mc	0.228
3	type1	parameter.power.mc	0.237
4	temp	parameter.power.mc	0.202
5	roastdarkness1	parameter.power.mc	0.307
6	roastdarkness2	parameter.power.mc	0.286