

# Package: baskoptr (via r-universe)

June 6, 2026

**Title** Utility-Based Optimization for Basket Trial Designs

**Version** 1.0.4

**Description** A unified framework for optimizing basket trial designs. To this end, the package supplies several utility functions and also a function for executing optimization algorithms on basket trial designs. The considered utility functions are discussed in Sauer et al. (2025) <[doi:10.1371/journal.pone.0323097](https://doi.org/10.1371/journal.pone.0323097)>.

**License** MIT + file LICENSE

**Imports** baskwrap (>= 1.0.1), future.apply

**Suggests** optimizr (>= 1.0.0), baskexact (>= 1.0.0), testthat (>= 3.0.0), progressr, basksim (>= 2.0.0)

**URL** <https://github.com/LukasDSauer/baskoptr>

**BugReports** <https://github.com/LukasDSauer/baskoptr/issues>

**Encoding** UTF-8

**RoxygenNote** 7.3.3

**Config/testthat/edition** 3

**NeedsCompilation** no

**Author** Lukas D Sauer [aut, cre] (ORCID:  
<<https://orcid.org/0000-0002-1340-9994>>)

**Maintainer** Lukas D Sauer <sauer@imbi.uni-heidelberg.de>

**Depends** R (>= 3.5.0)

**Repository** <https://cran.r-universe.dev>

**Date/Publication** 2026-04-02 20:57:38 UTC

**RemoteUrl** <https://github.com/cran/baskoptr>

**RemoteRef** HEAD

**RemoteSha** 10a010f543bc54f765e3fcbcd1bd52c253b8439d

## Contents

append_details	2
epsilon_extreme	3
get_details_for_two_scenarios	4
get_pls	5
get_scenarios	6
get_trace_info	6
io_val_pl	7
opt_design_gen	7
params_main	10
params_utility_caller	10
u_2ewp	10
u_avg	12
u_bnd	14
u_ewp	16
<b>Index</b>	<b>19</b>

---

append_details	<i>Internal helper function: Append and set elements of a details list</i>
----------------	--

---

### Description

append\_details takes the element details[["index"]] and appends it with the list further.  
 set\_details takes the element details[["index"]] and sets it to value.

### Usage

```
append_details(details, index, further)
```

```
set_details(details, index, value)
```

### Arguments

details	A list of details to be requested from baskwrap:get_details.fujikawa_x (will be supplied to the function as the which_details argument).
index	Indicates which element of details should be appended.
further	A list of further parameters to be appended.
value	A character string, details[[index]] will be assigned value.

### Value

The updated list of details.

---

epsilon_extreme	<i>Extreme borrowing cutoffs for borrowing in Fujikawa's design</i>
-----------------	---

---

### Description

Calculate the extreme borrowing cutoffs for the tuning parameters  $\tau$  and  $\epsilon$  in Fujikawa's basket trial design.

### Usage

```
epsilon_extreme(design, tau, detail_params)
```

```
tau_extreme(design, epsilon, detail_params)
```

### Arguments

design	An object of class <code>fujikawa</code> created by the function <code>baskwrap::setup_fujikawa_x</code> or <code>basksim::setup_fujikawa</code> .
tau	The optimization parameter $\tau$ of Fujikawa's basket trial design.
detail_params	A named list of parameters containing per-stratum sizes $n$ and the base of the logarithm for calculating Jensen-Shannon divergence <code>logbase</code> .
epsilon	The optimization parameter $\epsilon$ of Fujikawa's basket trial design.

### Details

The weights for the borrowing posterior in Fujikawa's design are calculated as

$$\omega_{ij} = \mathbf{1}(\tilde{\omega}_{ij}^\epsilon > \tau) \cdot \tilde{\omega}_{ij}^\epsilon$$

with  $\tau \in [0, 1]$ ,  $\epsilon \geq 0$ , and  $\tilde{\omega}_{ij} = 1 - JS(\text{Beta}_i^{\text{post}}, \text{Beta}_j^{\text{post}})$ . Here,  $JS(\cdot, \cdot)$  is the Jensen-Shannon divergence, and

$$\text{Beta}_i^{\text{post}} = \text{Beta}(a_i + r_i, b_i + n_i - r_i)$$

is the beta-binomial conjugated posterior distribution in the  $i$ -th stratum for prior Beta parameters  $a_i$  and  $b_i$ , number of responses  $r_i$  and number of patients  $n_i$ . In particular for

$$\tilde{\omega}_{ij}^\epsilon \leq \tau,$$

borrowing will only take place for baskets with completely identical response rates. We call this boundary the *extreme borrowing cutoff*. The functions `epsilon_extreme` and `tau_extreme` calculate the extreme borrowing cutoffs  $\tau_{\text{extreme}}(\epsilon)$  and  $\epsilon_{\text{extreme}}(\tau)$ , which are the boundaries of

$$\tau \geq \left( \max_{r_k \neq r_l} \tilde{\omega}_{kl} \right)^\epsilon$$

and

$$\epsilon \geq \log_{\max_{r_k \neq r_l} \tilde{\omega}_{kl}}(\tau).$$

The extreme borrowing cutoffs depend neither on  $p_0$  nor on  $p_1$ . They only depend on the number of baskets  $k$  and the number of patients per basket  $n$ . The extreme borrowing cutoff is discussed in Sauer et al. (2025).

**Value**

A numeric, the extreme borrowing cutoff for  $\varepsilon$ .

**References**

Sauer LD, Ritz A, Kieser M. Utility-based optimization of Fujikawa's basket trial design – Pre-specified protocol of a comparison study. PLOS ONE. 2025;20(5):e0323097. doi:10.1371/journal.pone.0323097

**Examples**

```
design <- basksim::setup_fujikawa(k = 3,
                               shape1 = 1,
                               shape2 = 1,
                               p0 = 0.2)

detail_params <- list(n = 20,
                     logbase = exp(1))
epsilon_extreme(design, tau = 0.5, detail_params)
tau_extreme(design, epsilon = 2, detail_params)
```

---

```
get_details_for_two_scenarios
```

*Internal helper function: Get details for two response scenarios*

---

**Description**

Internal helper function: Get details for two response scenarios

**Usage**

```
get_details_for_two_scenarios(
  design,
  x,
  detail_params,
  p1,
  p2,
  which_details_list = NULL
)
```

**Arguments**

design	An object of class fujikawa created by the function baskwrap::setup_fujikawa_x or basksim::setup_fujikawa.
x	A named list, the design's tuning parameters to be optimized.
detail_params	A named list of parameters that need to be supplied to get_details(). If p1 exists, detail_params\$p1 is replaced by p1 for calculating power resp. ECD and by p2 for calculating FWER.

p1	A numeric, response scenario for calculating power resp. ECD. This can also be left NULL, then the function goes looking in detail_params\$p1.
p2	A numeric, response scenario for calculating FWER, default is the global null scenario.
which_details_list	A list of two lists, which_details_list[["p1"]] and which_details_list[["p2"]], containing the details which are to be requested for p1 and p2, respectively. NULL means that all will be requested.

**Value**

A list of two lists containing return values of get\_details calls.

---

get_p1s	<i>Generate an array of possible outcome scenarios</i>
---------	--

---

**Description**

This functions generates an array of possible outcome scenarios. Each row of the array is a scenario vector. The array starts with k inactive baskets and 0 active baskets and then increments in steps of by up to k active baskets (or the next smaller multiple of by).

**Usage**

```
get_p1s(k, p0, p1, by = 1)
```

**Arguments**

k	The number of baskets.
p0	A common probability under the null hypothesis.
p1	A numeric, the true response probability for the active baskets
by	A numeric, increment of the scenario sequence, default: 1.

**Value**

A numeric array.

**Examples**

```
p1s <- get_p1s(k = 3, p0 = 0.2, p1 = 0.5)
```

---

get_scenarios	<i>Generate a scenario of possible outcomes</i>
---------------	---

---

**Description**

A wrapper of get\_p1s() that also returns the input values k and p0 as part of a list.

**Usage**

```
get_scenarios(k, p0, p1, by = 1)
```

**Arguments**

k	The number of baskets.
p0	A common probability under the null hypothesis.
p1	A numeric, the true response probability for the active baskets
by	A numeric, increment of the scenario sequence, default: 1.

**Value**

A list with components k, p0 and p1s, where p1s is generated by the the function get\_p1s.

**Examples**

```
scenario <- get_scenarios(k = 3, p0 = 0.2, p1 = 0.5)
```

---

get_trace_info	<i>Internal helper function: Decide whether to record a trace and what path to use</i>
----------------	--

---

**Description**

Internal helper function: Decide whether to record a trace and what path to use

**Usage**

```
get_trace_info(trace, type = "trace")
```

**Arguments**

trace	A parameter that can be FALSE, TRUE or a file path.
type	A character that designates the name of the input.

**Value**

A list with components rec and path.

---

io_val_p1	<i>Internal helper function: Input validation for p1</i>
-----------	--

---

**Description**

Returns detail\_params with checked element detail\_params\$p1 <- p1, depending on whether p1 is not NULL or detail\_params\$p1 is not NULL. Returns an error message if both are NULL.

**Usage**

```
io_val_p1(detail_params, p1)
```

**Arguments**

detail_params	A named list of parameters that need to be supplied to get_details(). If p1 exists, detail_params\$p1 is replaced by p1 for calculating power resp. ECD and by p2 for calculating FWER.
p1	A numeric, response scenario for calculating power resp. ECD. This can also be left NULL, then the function goes looking in detail_params\$p1.

**Value**

The updated list of detail\_params.

---

opt_design_gen	<i>Optimize a Basket Trial Design</i>
----------------	---------------------------------------

---

**Description**

Optimize the parameters of a basket trial design using a utility-based approach with a simulation algorithm of your choice.

**Usage**

```
opt_design_gen(  
  design,  
  utility,  
  algorithm,  
  detail_params,  
  utility_params,  
  algorithm_params,  
  x_names = NULL,  
  fn_name = "fn",  
  trace = FALSE,  
  trace_options = list(robust = FALSE),
```

```

format_result = NULL,
final_details = FALSE,
final_details_utility_params = utility_params,
progress_bar = NULL
)

```

## Arguments

design	An object of class <code>fujikawa</code> created by the function <code>baskwrap::setup_fujikawa_x</code> or <code>basksim::setup_fujikawa</code> .
utility	A function returning the utility of a parameter combination <code>x</code> of the form <code>utility(design, x, detail_params, ...)</code> , where <code>...</code> may further parameters to be supplied to the utility.
algorithm	A function returning the optimization algorithm's results, should have the form <code>algorithm(fn, ...)</code> , where <code>fn</code> is the function to be optimized and <code>...</code> may be further parameters of the optimization algorithm.
detail_params	A named list of parameters that need to be supplied to <code>get_details()</code> .
utility_params	A named list of further parameters that need to be supplied to the utility function.
algorithm_params	A named list of further parameters that need to be supplied to the optimization algorithm.
x_names	A character vector containing the names of the utility functions tuning parameters, automatically retrieved from <code>algorithm_params\$par</code> or <code>algorithm_params\$lower</code> if either is present. Default is <code>NULL</code> .
fn_name	The name of the function argument of <code>algorithm</code> . The default is <code>"fn"</code> .
trace	A logical or a character string, should the trace of the optimization algorithm be recorded (utility values, parameters vectors, and random number seeds)? Default is <code>FALSE</code> . If <code>TRUE</code> , recording is done by dynamic appending of a data frame (which may not be very efficient). If <code>trace</code> is a character vector specifying a file path (ending with <code>".RDS"</code> ), the trace will be dynamically saved to that RDS file. Some optimization algorithms automatically return their trace. In that case, you can switch off using <code>trace = FALSE</code> (or <code>""</code> or <code>NULL</code> ) and request the trace directly from your algorithm using <code>algorithm_params</code> .
trace_options	A list, if <code>trace_options = list(robust = TRUE)</code> , a robust recording method for saving the optimization steps is used that works well with parallelized algorithms. Caveat: the robust version is not sorted in chronological order.
format_result	(Optional:) A function <code>function(res)</code> for formatting the final output of the optimization algorithm. If you want <code>final_details = TRUE</code> , then the formatted result must have a non-null element <code>res\$par</code> containing the optimal parameters.
final_details	A logical, if <code>TRUE</code> , the function runs the utility function one more time on the optimization result and returns the output of the implicit call to <code>baskwrap::get_details()</code> as <code>attr("final_details")</code> and the result of the repeated utility function call as <code>attr("final_res_repeated")</code> . The latter should be identical to the result for deterministic calculations, but may differ for stochastic calculations.

- `final_details_utility_params` A list, only takes effect if `final_details==TRUE`. This last run of the utility function will use these list of parameters instead of `utility_params`. However, the default is `final_details_utility_params = utility_params`.
- `progress_bar` Do you want a progress bar? This argument either takes a numeric containing the number of steps or a `progressr::progressor()`. Only works if `trace` is activated. The number of steps is proportional to the number of algorithm iterations, but the exact number depends on the algorithm's implementation.

## Value

a list consisting of the algorithm's output (usually the optimal parameter vector and the resulting optimal utility value and some meta information). If `trace == TRUE`, the trace of the optimization algorithm can be found in the `[["trace"]]` entry of the list. (If the algorithm function also outputs a trace, this will be saved in an additional `[["trace_alg"]]` entry.)

## Examples

```
# Optimizing a three-basket trial design using Fujikawa's beta-binomial
# sharing approach
design <- baskwrap::setup_fujikawa_x(k = 3, shape1 = 1, shape2 = 1,
                                   p0 = 0.2, backend = "exact")
detail_params <- list(p1 = c(0.5, 0.2, 0.2),
                     n = 20,
                     weight_fun = baskwrap::weights_jsd,
                     logbase = exp(1))
utility_params <- list(penalty = 1, thresh = 0.1)
# Bounded simulated annealing with progress bar
progressr::handlers(global = TRUE)
opt_design_gen(design = design,
              utility = u_ewp,
              algorithm = optimizr::simann,
              detail_params = detail_params,
              utility_params = utility_params,
              algorithm_params = list(par = c(lambda = 0.99,
                                              epsilon = 2,
                                              tau = 0.5),
                                     lower = c(lambda = 0.001,
                                              epsilon = 1,
                                              tau = 0.001),
                                     upper = c(lambda = 0.999,
                                              epsilon = 10,
                                              tau = 0.999),
                                     control = list(maxit = 10,
                                                  temp = 10,
                                                  fnscale = -1)),
              progress_bar = 10 + 2)
```

---

params\_main *List of parameters used across the package*

---

### Description

List of parameters used across the package

### Arguments

design An object of class fujikawa created by the function `baskwrap::setup_fujikawa_x` or `basksim::setup_fujikawa`.

x A named list, the design's tuning parameters to be optimized.

detail\_params A named list of parameters that need to be supplied to `get_details()`.

report\_details A logical, if TRUE, the function returns the output of the implicit call to `baskwrap::get_details()` as `attr("details")`.

---

params\_utility\_caller *List of parameters used for functions calling utility functions*

---

### Description

List of parameters used for functions calling utility functions

### Arguments

utility A function returning the utility of a parameter combination `x` of the form `utility(design, x, detail_params, ...)`, where `...` may further parameters to be supplied to the utility.

utility\_params A named list of further parameters that need to be supplied to the utility function.

---

u\_2ewp *Utility functions: Two-level power-error combination functions*

---

### Description

These utility functions combine rewarding power and penalizing TOER by subtracting TOER from power. In addition, unacceptably high FWER above a certain threshold receives harsher penalty.

**Usage**

```

u_2ewp(
  design,
  x,
  detail_params,
  p1 = NULL,
  penalty1,
  penalty2,
  threshold,
  report_details = FALSE
)

u_2pow(
  design,
  x,
  detail_params,
  p1 = NULL,
  penalty1,
  penalty2,
  threshold,
  report_details = FALSE
)

```

**Arguments**

design	An object of class fujikawa created by the function <code>baskwrap::setup_fujikawa_x</code> or <code>basksim::setup_fujikawa</code> .
x	A named list, the design's tuning parameters to be optimized.
detail_params	A named list of parameters that need to be supplied to <code>get_details()</code> .
p1	A numeric, response scenario for calculating power and error rate.
penalty1	A numeric, penalty1 is the penalty for low FWER,
penalty2	A numeric, penalty1 + penalty2 is the penalty for high FWER.
threshold	A numeric, for high FWER above this threshold we impose a harsher penalty.
report_details	A logical, if TRUE, the function returns the output of the implicit call to <code>baskwrap::get_details()</code> as <code>attr("details")</code> .

**Details**

The utility function  $u_{2ewp}$  is defined as

$$u_{2ewp}(\boldsymbol{\phi}, \mathbf{p}) = ewp(\boldsymbol{\phi}, \mathbf{p}) - (\xi_1 fwer(\boldsymbol{\phi}, \mathbf{p}) + \xi_2 (fwer(\boldsymbol{\phi}, \mathbf{p}) - \eta) \mathbf{1}(fwer(\boldsymbol{\phi}, \mathbf{p}) - \eta)),$$

where  $\eta \in [0, 1]$  is a threshold for imposing harsher FWER penalty.

The utility function  $u_{2pow}$  is defined analogously as

$$u_{2pow}(\boldsymbol{\phi}, \mathbf{p}) = \sum_{i \in R} pow_i(\boldsymbol{\phi}, \mathbf{p}) - \sum_{j \in R^c} (\xi_1 toer_j(\boldsymbol{\phi}, \mathbf{p}) + \xi_2 (toer_j(\boldsymbol{\phi}, \mathbf{p}) - \eta) \mathbf{1}(toer_j(\boldsymbol{\phi}, \mathbf{p}) - \eta)),$$

where  $R$  and  $R^c$  are the sets of active and inactive strata, respectively.

The (averaged)  $u_{2\text{pow}}$  is defined in Jiang et al. (2021).

### Value

A numeric, the parameter combination's utility.

### References

Jiang L, Nie L, Yan F, Yuan Y. Optimal Bayesian hierarchical model to accelerate the development of tissue-agnostic drugs and basket trials. *Contemporary Clinical Trials*. 2021;107:106460. doi:10.1016/j.cct.2021.106460

### Examples

```
design <- baskwrap::setup_fujikawa_x(k = 3, shape1 = 1, shape2 = 1,
                                  p0 = 0.2, backend = "exact")
u_2ewp(design,
        x = list(lambda = 0.99, epsilon = 2, tau = 0.5),
        detail_params = list(p1 = c(0.5, 0.2, 0.2),
                              n = 20,
                              weight_fun = baskwrap::weights_jsd,
                              logbase = exp(1)),
        penalty1 = 1, penalty2 = 2,
        threshold = 0.1)
u_2pow(design,
        x = list(lambda = 0.99, epsilon = 2, tau = 0.5),
        detail_params = list(p1 = c(0.5, 0.2, 0.2),
                              n = 20,
                              weight_fun = baskwrap::weights_jsd,
                              logbase = exp(1)),
        penalty1 = 1, penalty2 = 2,
        threshold = 0.1)
```

---

u\_avg

*Utility function: Scenario-averaged utility function*

---

### Description

For a utility function  $u(\cdot, \mathbf{p})$  and a set of true scenarios  $\{\mathbf{p}_i, \dots\}$ , calculate the weighted average utility function

$$\bar{u}(x) = \sum_i w_i u(x, \mathbf{p}_i)$$

for a set of weights with  $\sum_i w_i = 1$ . By default,  $w_i = \frac{1}{|\{\mathbf{p}_i, \dots\}|}$  for all  $i$ . The idea of averaging utility functions across a set of scenarios is taken from Jiang et al. (2021).

**Usage**

```

u_avg(
  design,
  x,
  detail_params,
  utility,
  utility_params,
  p1s,
  weights_u = rep(1/nrow(p1s), nrow(p1s)),
  report_details = FALSE,
  penalty_maxtoer = NULL,
  threshold_maxtoer = NULL,
  use_future = FALSE
)

```

**Arguments**

design	An object of class <code>fujikawa</code> created by the function <code>baskwrap::setup_fujikawa_x</code> or <code>basksim::setup_fujikawa</code> .
x	A named list, the design's tuning parameters to be optimized.
detail_params	A named list of parameters that need to be supplied to <code>get_details()</code> . It must not contain <code>p1</code> , as this is supplied separately.
utility	A function returning the utility of a parameter combination <code>x</code> of the form <code>utility(design, x, detail_params, ...)</code> , where <code>...</code> may further parameters to be supplied to the utility.
utility_params	A named list of further parameters that need to be supplied to the utility function.
p1s	A numeric array in which each row defines a scenario of true response rates under the alternative hypothesis.
weights_u	A numeric vector of weights for calculating the weighted average.
report_details	A logical, if <code>TRUE</code> , the function returns the output of the implicit call to <code>baskwrap::get_details()</code> as <code>attr("details")</code> .
penalty_maxtoer	A numeric, the penalty for punishing the maximal TOER across all considered scenarios and all strata.
threshold_maxtoer	A numeric, above this threshold maximal TOER is punished by returning <code>-penalty_maxtoer</code> times the maximal TOER. Default is <code>NULL</code> , which means no penalty.
use_future	A logical, should <code>future_apply()</code> instead of <code>apply()</code> be used for calculating the utilities to be averaged over. You must still activate a <i>future</i> backend for this option to take any effect, default is <code>FALSE</code> .

**Value**

A numeric, the parameter combination's utility.

## References

Jiang L, Nie L, Yan F, Yuan Y. Optimal Bayesian hierarchical model to accelerate the development of tissue-agnostic drugs and basket trials. *Contemporary Clinical Trials*. 2021;107:106460. doi:10.1016/j.cct.2021.106460

## Examples

```
design <- baskwrap::setup_fujikawa_x(k = 3, shape1 = 1, shape2 = 1,
                                  p0 = 0.2, backend = "exact")
x <- list(lambda = 0.99, epsilon = 2, tau = 0.5)
detail_params <- list(n = 20,
                     weight_fun = baskwrap::weights_jsd,
                     logbase = exp(1))
p1s <- rbind(c(0.2,0.2,0.2), c(0.2,0.2,0.5), c(0.2,0.5,0.5), c(0.5,0.5,0.5))
# Averaging over u_ewp()
u_avg(design,
      x = x,
      detail_params = detail_params,
      utility = u_ewp,
      utility_params = list(penalty = 1, threshold = 0.1),
      p1s = p1s
    )

# Averaging over u_2ewp()
utility_params_2ewp <- list(penalty1 = 1, penalty2 = 2, threshold = 0.1)
u_avg(design,
      x = x,
      detail_params = detail_params,
      utility = u_2ewp,
      utility_params = utility_params_2ewp,
      p1s = p1s
    )

# Punishing maximal TOER in all scenarios and all strata
u_avg(design,
      x = x,
      detail_params = detail_params,
      utility = u_2ewp,
      utility_params = utility_params_2ewp,
      p1s = p1s,
      penalty_maxtoer = 1, threshold_maxtoer = 0.1
    )
```

---

u\_bnd

*Utility function with boundaries on the parameters*


---

## Description

This function manually implements boundaries for a given utility function `utility`. If the vector `x` lies out of the lower and upper bounds, the function returns `NA_real_`. Else it returns the utility functions value.

**Usage**

```
u_bnd(
  design,
  x,
  detail_params,
  utility,
  utility_params,
  lower,
  upper,
  report_details = FALSE
)
```

**Arguments**

design	An object of class fujikawa created by the function <code>baskwrap::setup_fujikawa_x</code> or <code>basksim::setup_fujikawa</code> .
x	A named list, the design's tuning parameters to be optimized.
detail_params	A named list of parameters that need to be supplied to <code>get_details()</code> .
utility	A function returning the utility of a parameter combination <code>x</code> of the form <code>utility(design, x, detail_params, ...)</code> , where <code>...</code> may further parameters to be supplied to the utility.
utility_params	A named list of further parameters that need to be supplied to the utility function.
lower	numerical, a vector of lower bounds of the parameters.
upper	numerical, a vector of upper bounds of the parameters.
report_details	A logical, if TRUE, the function returns the output of the implicit call to <code>baskwrap::get_details()</code> as <code>attr(, "details")</code> .

**Value**

A numeric, the parameter combination's utility.

**Examples**

```
design <- baskwrap::setup_fujikawa_x(k = 3, shape1 = 1, shape2 = 1,
                                   p0 = 0.2, backend = "exact")
lower <- list(lambda = 0, epsilon = 1, tau = 0)
upper <- list(lambda = 1, epsilon = 10, tau = 1)
utility_params <- list(penalty = 1, threshold = 0.1)
detail_params <- list(p1 = c(0.5, 0.2, 0.2),
                     n = 20,
                     weight_fun = baskwrap::weights_jsd,
                     logbase = exp(1))

# Out of bounds
u_bnd(design = design,
      x = list(lambda = 1.3, epsilon = 2, tau = 0.5),
      detail_params = detail_params,
      utility = u_ewp,
      utility_params = utility_params,
```

```

        lower = lower,
        upper = upper)
# Inside bounds, this is the same as u_ewp
x <- list(lambda = 0.99, epsilon = 2, tau = 0.5)
u_bnd(design = design,
      x = x,
      detail_params = detail_params,
      utility = u_ewp,
      utility_params = utility_params,
      lower = lower,
      upper = upper)
u_ewp(design = design,
      x = x,
      detail_params = detail_params, penalty = 1, threshold = 0.1)

```

---

u\_ewp

*Utility functions: Discontinuous power/ECD functions with type-I error penalty*


---

### Description

These utility functions return experiment-wise power (EWP) or expected number of correct decisions (ECD) if the family-wise error rate (FWER) is low and the negative FWER multiplied by a penalty parameter if the FWER is high.

### Usage

```

u_ewp(
  design,
  x,
  detail_params,
  p1 = NULL,
  p2 = rep(design$p0, design$k),
  threshold,
  penalty,
  report_details = FALSE,
  reduce_calculations = ifelse(design$backend == "exact", TRUE, FALSE)
)

u_ecd(
  design,
  x,
  detail_params,
  p1 = NULL,
  p2 = rep(design$p0, design$k),
  penalty,
  threshold,
  report_details = FALSE,

```

```

    reduce_calculations = ifelse(design$backend == "exact", TRUE, FALSE)
  )

```

### Arguments

design	An object of class fujikawa created by the function <code>baskwrap::setup_fujikawa_x</code> or <code>basksim::setup_fujikawa</code> .
x	A named list, the design's tuning parameters to be optimized.
detail_params	A named list of parameters that need to be supplied to <code>get_details()</code> . If <code>p1</code> exists, <code>detail_params\$p1</code> is replaced by <code>p1</code> for calculating power resp. ECD and by <code>p2</code> for calculating FWER.
p1	A numeric, response scenario for calculating power resp. ECD. This can also be left NULL, then the function goes looking in <code>detail_params\$p1</code> .
p2	A numeric, response scenario for calculating FWER, default is the global null scenario.
threshold	A numeric, for high FWER above this threshold we impose a penalty, default: 0.1.
penalty	A numeric, the scaling factor for FWER penalty, default: 1.
report_details	A logical, if TRUE, the function returns the output of the implicit call to <code>baskwrap::get_details()</code> as <code>attr("details")</code> .
reduce_calculations	A logical, only takes effect for the "exact" backend. If TRUE, the function will only execute the <code>get_details()</code> function for <code>p1</code> if the FWER for <code>p2</code> turned out to be low enough. This may speed up function execution. Default is TRUE for the "exact" backend and FALSE otherwise.

### Details

The utility function `u_ewp` is defined as defined as

$$u_{ewp}(x, \mathbf{p}_1, \mathbf{p}_2) = ewp(x, \mathbf{p}_1),$$

if the FWER fulfills  $fwer(x, \mathbf{p}_2) < \eta_1$ , and

$$u_{ewp}(x, \mathbf{p}_1, \mathbf{p}_2) = -\xi_1 \cdot fwer(x, \mathbf{p}_2),$$

if  $fwer(x, \mathbf{p}_2) \geq \eta_1$ . The parameter  $\eta_1$  is called the threshold, the parameter  $\xi_1$  is called the penalty.

The utility function `u_eed` is defined analogously with the expected number of correct decisions (ECD) instead of the experiment-wise power. The use of ECD together with FWER constraints in the context of basket trials stems from Broglio et al. (2020).

### Value

A numeric, the parameter combination's utility.

### References

Broglio KR, Zhang F, Yu B, et al. A Comparison of Different Approaches to Bayesian Hierarchical Models in a Basket Trial to Evaluate the Benefits of Increasing Complexity. *Statistics in Biopharmaceutical Research*. 2022;14(3):324-333. doi:10.1080/19466315.2021.2008484

**Examples**

```
# Calculating the EWP utility using basksim as a backend
design <- baskwrap::setup_fujikawa_x(k = 3, shape1 = 1, shape2 = 1, p0 = 0.2)
u_ewp(design,
      x = list(lambda = 0.99, epsilon = 2, tau = 0.5),
      detail_params = list(p1 = c(0.5, 0.2, 0.2),
                           n = 20,
                           iter = 100,
                           logbase = exp(1)),
      penalty = 1, threshold = 0.1)
# Calculating the ECD utility using baskexact as a backend
design_x <- baskwrap::setup_fujikawa_x(k = 3, shape1 = 1, shape2 = 1,
                                     p0 = 0.2, backend = "exact")
u_ecd(design_x,
      x = list(lambda = 0.99, epsilon = 2, tau = 0.5),
      detail_params = list(p1 = c(0.5, 0.2, 0.2),
                           n = 20,
                           weight_fun = baskwrap::weights_jsd,
                           logbase = exp(1)),
      penalty = 1, threshold = 0.1)
```

# Index

`append_details`, 2

`epsilon_extreme`, 3

`get_details_for_two_scenarios`, 4

`get_p1s`, 5

`get_scenarios`, 6

`get_trace_info`, 6

`io_val_p1`, 7

`opt_design_gen`, 7

`params_main`, 10

`params_utility_caller`, 10

`set_details (append_details)`, 2

`tau_extreme (epsilon_extreme)`, 3

`u_2ewp`, 10

`u_2pow (u_2ewp)`, 10

`u_avg`, 12

`u_bnd`, 14

`u_ecd (u_ewp)`, 16

`u_ewp`, 16