Supplementary Material Thesis

Contents

The purpose of this folder is to make the experiments presented in my thesis reproducible.

Head over to the main README for more information and instructions of how to install the package.

Study Performance of CBREE for Low Dimensional Problems

```
from os import path
import rareeventestimation as ree
import pandas as pd
import plotly.express as px
import numpy as np
from rareeventestimation.evaluation.constants import INDICATOR_APPROX_LATEX_NAME, BM_SOLVER_SCATTER_STYLI
import plotly.graph_objects as go
from IPython.display import display, Markdown
# recommended: use autoreload for development: https://ipython.readthedocs.io/en/stable/config/extensions
%load_ext autoreload
%autoreload 2
```

Load Data

Option 1: Get precomputed data online

```
# data is here: https://archive.org/details/konstantinalthaus-rareeventestimation-data
# you can got to this link and inspect the files before loading
df_agg = pd.read_json("https://archive.org/download/konstantinalthaus-rareeventestimation-data/cbree_toy_
df_agg_all = pd.read_json("https://archive.org/download/konstantinalthaus-rareeventestimation-data/cbree_
df_bm_agg = pd.read_json("https://archive.org/download/konstantinalthaus-rareeventestimation-data/benchma
```

Skip to main content

Option 2: Aggregate locally precomputed data

```
## uncomment to load existing data
## or to compile data after computing it yourself:
# data_dir ="docs/benchmarking/data/cbree_sim/toy_problems"
# path_df= path.join(data_dir, "cbree_toy_problems_processed.json")
# path_df_agg = path.join(data_dir, "cbree_toy_problems_aggregated.json")
# path_df_agg_all = path.join(data_dir, "cbree_toy_problems_aggregated_all.json")
# if not  (path.exists(path_df) and path.exists(path_df_agg) and path.exists(path_df_agg_all)):
#     # load and clean
#     df = ree.load_data(data_dir, "*")
#     df.drop(columns=["index", "Unnamed: 0"], inplace=True)
#     df.drop_duplicates(inplace=True)
#     df.reset_index(drop=True, inplace=True)
#     # Round parameters to compare floats safely
#     for col in DF_COLUMNS_TO_LATEX.keys():
#         if isinstance(df[col].values[0], float):
#             df[col] = df[col].round(5)
#     # process data: add obs_window and callback to solver name
#     df = df.apply(expand_cbree_name, axis=1, columns= ['observation_window', 'callback'])
#     # pretty names
#     to_drop = ["mixture_model"] # info is redundant as resample = False and callback exists
#     replace_values = {"Method": {"False": "CBREE", "vMFN Resample": "CBREE (vMFN, resampled)"}}
#     df = df.drop(columns=to_drop) \
#         .rename(columns=DF_COLUMNS_TO_LATEX) \
#         .replace(replace_values)
#     # rocess data: add evaluations
#     df_success = ree.add_evaluations(df.copy(),  only_success=True)
#     df_all = ree.add_evaluations(df.copy())
#     # aggregate
#     df_agg = ree.aggregate_df(df_success)
#     df_agg_all = ree.aggregate_df(df_all)
#     # Round parameters to compare floats safely. Has to be tone twice :(
#     for col in DF_COLUMNS_TO_LATEX.values():
#         if isinstance(df_agg[col].values[0], float):
#             df_agg[col] = df_agg[col].round(5)
#     for col in DF_COLUMNS_TO_LATEX.values():
#             if isinstance(df_agg_all[col].values[0], float):
#                 df_agg_all[col] = df_agg_all[col].round(5)
#     # save
#     df_success.to_json(path_df)
#     df_agg.to_json(path_df_agg)
#     df_agg_all.to_json(path_df_agg_all)
# else:
#     df_success = pd.read_json(path_df)
#     df_agg = pd.read_json(path_df_agg)
#     df_agg_all = pd.read_json(path_df_agg_all)
# # load benchmark
# df_bm, df_bm_agg = ree.get_benchmark_df()
```

Analyze Data

Make table with parameters used

```
paras = ["Sample Size"] + [v for v in DF_COLUMNS_TO_LATEX.values() if v!= "Method"]
tbl_params = df_agg.loc[:, tuple(paras)] \
    .replace({"Smoothing Function": INDICATOR_APPROX_LATEX_NAME}) \
    .apply(lambda col: col.sort_values().unique()) \
    .apply(ree.list_to_latex)
tbl_params = tbl_params.to_frame(name="Values")
tbl_params.index.name = "Parameter"
#save and show
tbl_params.style.to_latex("parameters_used_toy_problems.tex", clines="all;data")
display(tbl_params)
```

Skip to main content

Parameter	Values
Sample Size	$\{ 1000, 2000, 3000, 4000, 5000, 6000 \}$
$\epsilon_{{\text{{Target}}}}$	$\{ 0.10, 0.50 \}$
$\Delta_{{\text{{Target}}}}$	$\{ 1, 2, 5, 7, 10 \}$
Lip(σ)	1
Smoothing Function	$I_\text{{alg}}$, $I_\text{{arctan}}$
$N_{{ \text{{obs}} }}$	$\{ 0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...

Decide which smoothing function is best for lower dimensional problems

```
low_dim_probs= ["Convex Problem", "Linear Problem (d=2)", "Fujita Rackwitz Problem (d=2)"]
def decide(grp,par):
    """Custom function to decide betweeen to parameters."""
    best = grp.sort_values("Relative Root MSE")[par].values[0]
    return(pd.Series([best], index =[f"{par}"]))
# pull data
df_tgt_fun = df_agg.query("Problem in @low_dim_probs & Method == 'CBREE'")\
    .loc[:,("Problem",
            "Sample Size",
            "$\\Delta_{{\\text{{Target}}}}$",
            "Method",
            "Smoothing Function",
            "Relative Root MSE",
            "$N_{{ \\text{{obs}} }}$",
            "$\\epsilon_{{\\text{{Target}}}}$")] \
    .groupby(["Problem",
              "Sample Size",
              "$\\Delta_{{\\text{{Target}}}}$",
              "Method",
              "$N_{{ \\text{{obs}} }}$",
              "$\\epsilon_{{\\text{{Target}}}}$"]) \
    .apply(decide, "Smoothing Function")
tbl = df_tgt_fun.reset_index().value_counts(subset=["Smoothing Function", "Problem"], normalize=False)\
    .to_frame()\
    .unstack(level=1)
tbl.columns = tbl.columns.droplevel(0)
totals = tbl.sum()
tbl = tbl/totals
tbl["Total"] = tbl.mean(numeric_only=True, axis=1)
# style and save
tbl = tbl.sort_values("Total", ascending=False)
best_approximation = tbl.index.values[0]
tbl = tbl.applymap(lambda x: f"{x*100:.2f}\%")
tbl = tbl.rename(index = INDICATOR_APPROX_LATEX_NAME)
tbl = tbl.rename(columns={c:ree.squeeze_problem_names(c) for c in tbl.columns})
tbl.style.to_latex("performance_approximations.tex", clines="all;data")
display(tbl)
# make and save caption
tbl_description = f"Comparing the estimates of $\\textup{{relRootMSE}}(\\hat{{P}}_f)$ for different smoo
with open(f"performance_approximations_desc.tex", "w") as file:
    file.write(tbl_description)
display(Markdown(tbl_description))
```

Smoothing Function \ Problem	CP	FRP (d=2)	LP (d=2)	Total
$I_\text{{alg}}$	69.88\%	91.55\%	78.21\%	79.88\%
$I_\text{{arctan}}$	30.12\%	8.45\%	21.79\%	20.12\%

Comparing the estimates of $\textup{relRootMSE}(\hat{P}_f)$ for different smoothing functions averaged over all other parameter choices. The values denote the relative number of cases the corresponding smoothing function performed best for the given problem (in total 840 per problem).

Skip to main content

Decide which stepsize tolerance is best for lower dimensional problems

```
# pull data
df_tol = df_agg.query("Problem in @low_dim_probs & Method == 'CBREE'")\
    .loc[:,("Problem",
            "Sample Size",
            "$\\Delta_{{\\text{{Target}}}}$",
            "Method",
            "$\\epsilon_{{\\text{{Target}}}}$",
            "Relative Root MSE",
            "$N_{{ \\text{{obs}} }}$",
            "Smoothing Function")] \
    .groupby(["Problem",
              "Sample Size",
              "$\\Delta_{{\\text{{Target}}}}$",
              "Method",
              "$N_{{ \\text{{obs}} }}$",
              "Smoothing Function"]) \
    .apply(decide,  "$\\epsilon_{{\\text{{Target}}}}$")
tbl = df_tol.reset_index().value_counts(subset=["$\\epsilon_{{\\text{{Target}}}}$", "Problem"], normalize
    .unstack(level=1)
totals = tbl.sum()
tbl = tbl/totals
tbl["Total"] = tbl.mean(numeric_only=True, axis=1)
# style and save
tbl = tbl.sort_values("Total", ascending=False)
best_tolerance = tbl.index.values[0]
tbl = tbl.applymap(lambda x: f"{x*100:.2f}\%")
tbl = tbl.rename(columns={c:ree.squeeze_problem_names(c) for c in tbl.columns}).\
    rename(index={idx:str(idx) for idx in tbl.index })
tbl.style.to_latex("performance_stepsize_tolerance.tex", clines="all;data")
display(tbl)
# make and save caption
tbl_description = f"Comparing the estimates of $\\textup{{relRootMSE}}(\\hat{{P}}_f)$ for different valu
with open(f"performance_stepsize_tolerance_desc.tex", "w") as file:
    file.write(tbl_description)
display(Markdown(tbl_description))
```

Problem	CP	FRP (d=2)	LP (d=2)	Total
$\epsilon_{{\text{{Target}}}}$				
0.5	53.10\%	67.86\%	76.43\%	65.79\%
0.1	46.90\%	32.14\%	23.57\%	34.21\%

Comparing the estimates of $\textup{relRootMSE}(\hat{P}*f)$ for different values of $\epsilon*{\text{Target}}$ averaged over all other parameter choices. The values denote the relative number of cases (in total 840 per problem) the corresponding value performed best for the given problem.

Study effect of divergence check on success rate

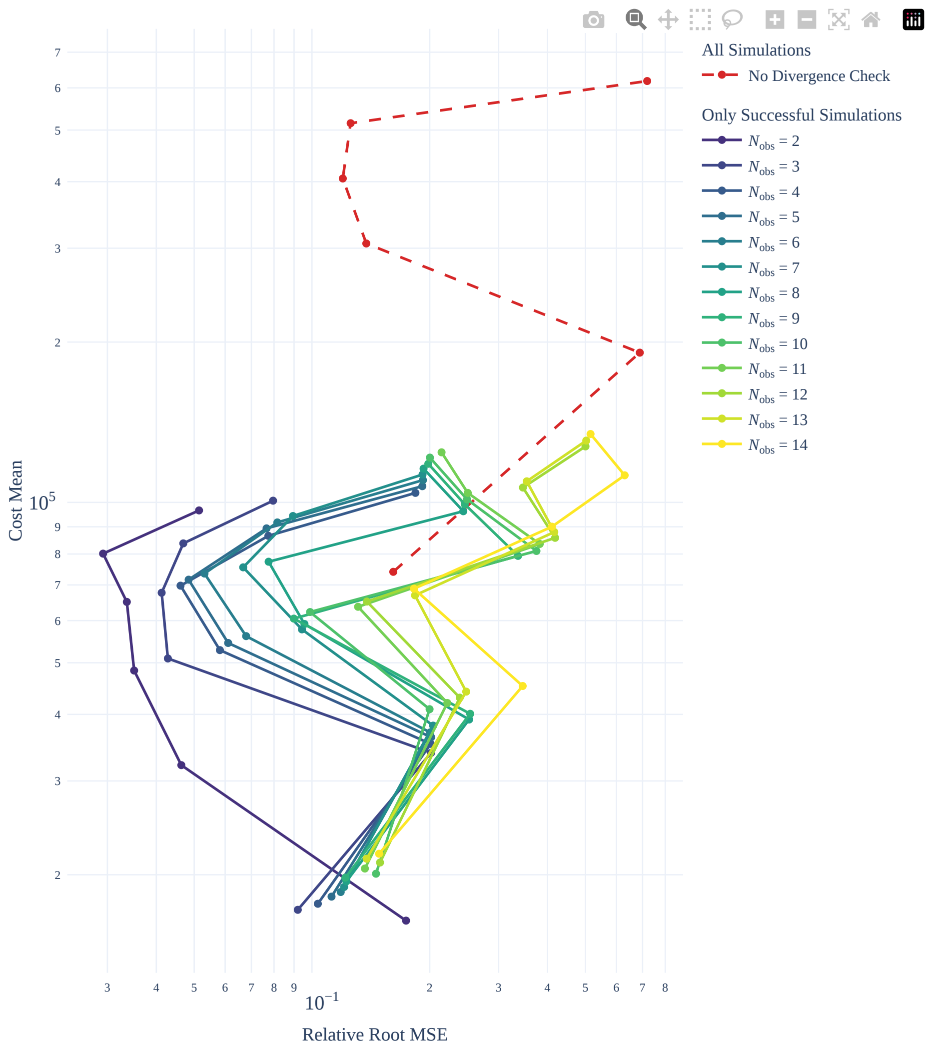
```
# pull data
df_rates = df_agg.query("Problem in @low_dim_probs & Method == 'CBREE'")
df_rates = df_rates[df_rates["$\\epsilon_{{\\text{{Target}}}}$"]==best_tolerance]
df_rates = df_rates[df_rates["Smoothing Function"] == best_approximation]
df_rates = df_rates[df_rates["$\\Delta_{{\\text{{Target}}}}$"]==1]
tbl_success_rates = pd.pivot_table(df_rates,
                                   values="Success Rate",
                                   index='$N_{{ \\text{{obs}} }}$',
                                   columns="Problem",
                                   aggfunc=np.mean)
tbl_success_rates = tbl_success_rates.groupby(list(tbl_success_rates))\
    .apply(lambda x: ree.vec_to_latex_set(x.index.values))\
    .to_frame(name='$N_{{ \\text{{obs}} }}$')\
    .reset_index()\
    .set_index('$N_{{ \\text{{obs}} }}$') \
    .applymap(lambda x: f"{x*100:.2f}\%") \
    .rename(index={"0":'No div. check'})
# style and save
tbl_success_rates.rename(columns = {c: ree.squeeze_problem_names(c) for c in tbl_success_rates.columns},
tbl_success_rates.style.to_latex("success_obs_window.tex", clines="all;data")
display(tbl_success_rates)
# make and save caption
tbl_description = f"Comparing the success rates of the CBREE  method for different values of $N_{{ \\tex
The parameters $\\Delta_{{\\text{{Target}}}}=1$, \
$\\epsilon_{{\\text{{Target}}}} = {best_tolerance}$ \
and the choice of the indicator approximation {INDICATOR_APPROX_LATEX_NAME[best_approximation]} \
are fixed. \
The values denote the relative number of cases the CBREE method converged successfully for the particula
with open(f"success_obs_window_desc.tex", "w") as file:
    file.write(tbl_description)
display(Markdown(tbl_description))
```

	CP	FRP (d=2)	LP (d=2)
$N_{{ \text{{obs}} }}$			
No div. check	98.92\%	7.83\%	99.67\%
\{2, 3, \ldots, 14\}	100.00\%	100.00\%	100.00\%

Comparing the success rates of the CBREE method for different values of $N_{ \text{obs} }$ averaged over all sample sizes $J = {1000, 2000, \ldots, 6000}$. The parameters $\Delta_{\text{Target}}=1$, $\epsilon_{\text{Target}} = 0.5$ and the choice of the indicator approximation $I_\text{{alg}}$ are fixed. The values denote the relative number of cases the CBREE method converged successfully for the particular combination of problem and paramter setting (in total 1200 per setting).

Study effect of divergence check on performance of Fujita Rackwitz Problem

```
# pull data
fr_all = df_agg_all.query("Problem == 'Fujita Rackwitz Problem (d=2)' & Method =='CBREE'")
fr_all = fr_all[(fr_all[DF_COLUMNS_TO_LATEX['observation_window']]==0) & \
                (fr_all[DF_COLUMNS_TO_LATEX['stepsize_tolerance']]==best_tolerance) & \
                (fr_all[DF_COLUMNS_TO_LATEX['tgt_fun']]==best_approximation) & \
                (fr_all[DF_COLUMNS_TO_LATEX['cvar_tgt']]==1)]
fr_all = fr_all.assign(Portion="All Simulations")
fr_success = df_agg.query("Problem == 'Fujita Rackwitz Problem (d=2)' & Method =='CBREE'")
fr_success = fr_success[(fr_success[DF_COLUMNS_TO_LATEX['observation_window']]>0) & \
                (fr_success[DF_COLUMNS_TO_LATEX['stepsize_tolerance']]==best_tolerance) & \
                (fr_success[DF_COLUMNS_TO_LATEX['tgt_fun']]==best_approximation) & \
                (fr_success[DF_COLUMNS_TO_LATEX['cvar_tgt']]==1)]
fr_success = fr_success.assign(Portion="Only Successful Simulations")
df_fr = pd.concat([fr_all, fr_success], axis=0)
# make plot
fig_fr =go.Figure()
n_obs_col_dict = ree.sr_to_color_dict(df_fr[DF_COLUMNS_TO_LATEX['observation_window']])
for n in df_fr[DF_COLUMNS_TO_LATEX['observation_window']].sort_values().unique():
    this_df = df_fr[df_fr[DF_COLUMNS_TO_LATEX['observation_window']] ==n]\
        .sort_values("Sample Size")
    tr = go.Scatter(
        x = this_df["Relative Root MSE"],
        y = this_df["Cost Mean"],
        mode="markers+lines",
        marker_symbol = "circle",
        legendgroup = str(n==0),
        marker_color = CMAP[3] if n==0 else n_obs_col_dict[str(n)],
        legendgrouptitle_text = "All Simulations" if n==0 else "Only Successful Simulations",
        name= "No Divergence Check" if n==0 else f"{LATEX_TO_HTML[DF_COLUMNS_TO_LATEX['observation_windo
        line_dash = "dash" if n==0 else "solid")
    fig_fr.add_trace(tr)
# style and save plot
fig_fr.update_layout(**MY_LAYOUT)
fig_fr.update_layout(height=800)
fig_fr.update_xaxes(title_text="Relative Root MSE", type="log")
fig_fr.update_yaxes(title_text="Cost Mean", type="log",  title_standoff=0)
fig_fr.write_image("divergence_fujita_rackwitz.png", scale=WRITE_SCALE)
fig_fr.show()
# make and save caption
fig_description = f"The effect of the divergence check on the Fujita Rackwitz Problem (d=2). \
We show the empirical error and cost estimates based on all 200 simulations (successful or not) \
if the CBREE methods runs with no divergence check. \
The same quantities  based on the successful portion of the 200 simulations \
are plotted for different values of $N_\\text{{obs}}$ \
if the divergence check is active.\
The parameters $\\Delta_{{\\text{{Target}}}}=1$,  $\\epsilon_{{\\text{{Target}}}} = {best_tolerance}$ \
and the choice of the indicator approximation {INDICATOR_APPROX_LATEX_NAME[best_approximation]} \
are fixed."
with open("divergence_fujita_rackwitz_desc.tex", "w")  as f:
    f.write(fig_description)
display(Markdown(fig_description))
```



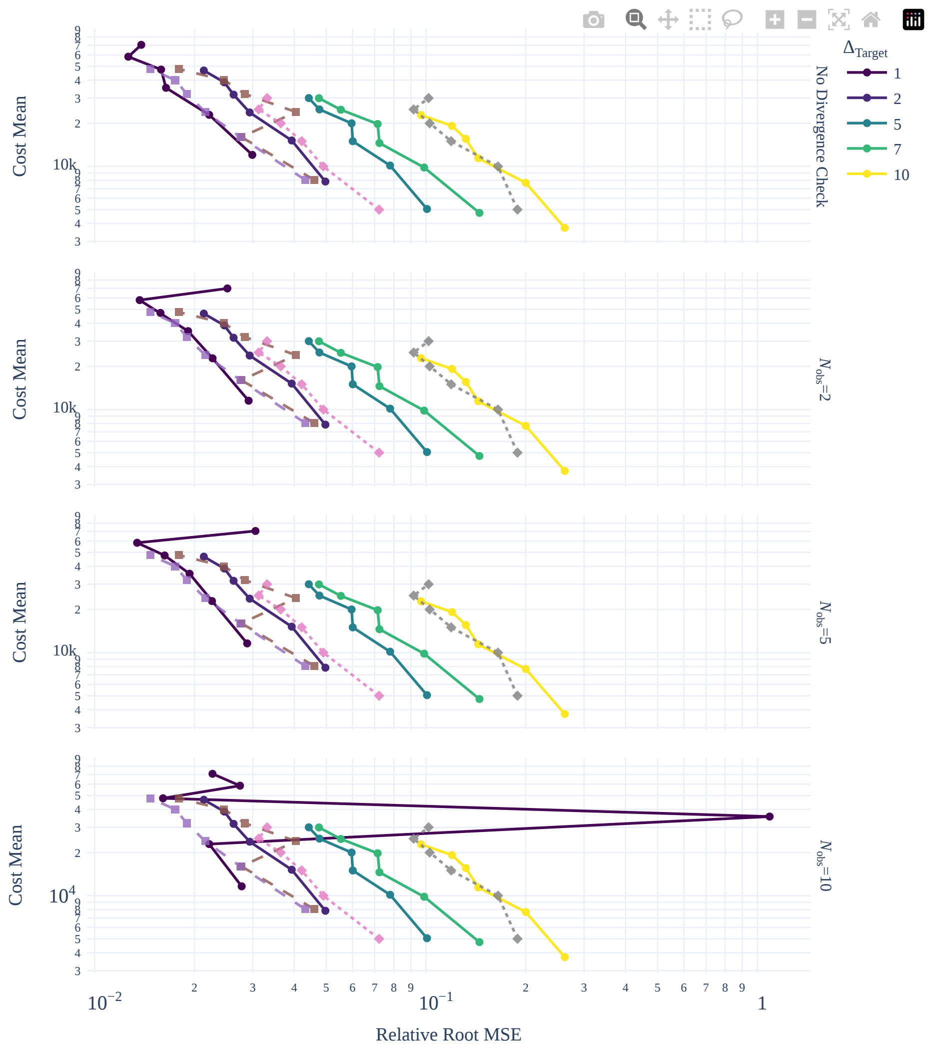
The effect of the divergence check on the Fujita Rackwitz Problem (d=2). We show the empirical error and cost estimates based on all 200 simulations (successful or not) if the CBREE methods runs with no divergence check. The same quantities based on the successful portion of the 200 simulations are plotted for different values of N_obs if the divergence check is active.The parameters $\Delta_{\text{Target}}=1$, $\epsilon_{\text{Target}} = 0.5$ and the choice of the indicator approximation $I_\text{{alg}}$ are fixed.

Skip to main content

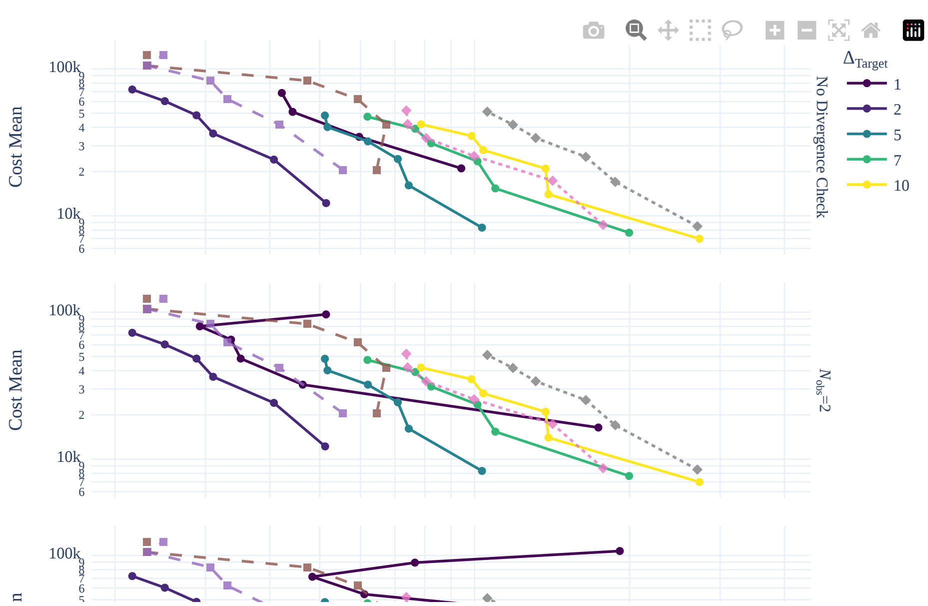
Study peformance

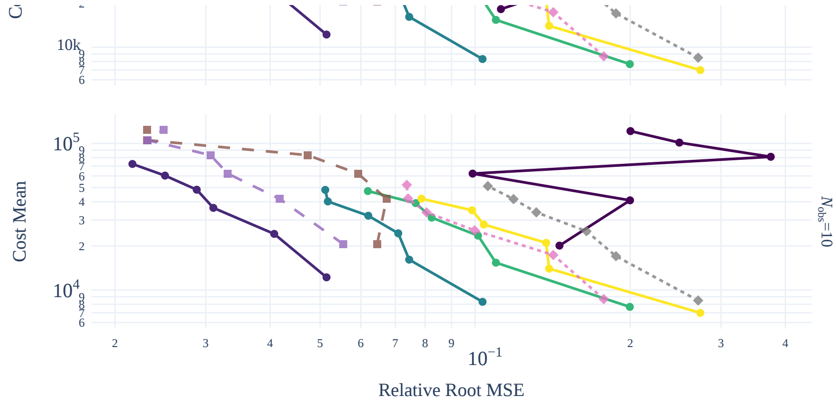
```
for prob in df_agg.Problem.unique():
    # filter
    this_df = df_agg.query("Problem == @prob & Method=='CBREE'")
    this_df = this_df[this_df["$\\epsilon_{{\\text{{Target}}}}$"]==best_tolerance]
    this_df = this_df[this_df["Smoothing Function"] == best_approximation]
    this_df = this_df[this_df['$N_{{ \\text{{obs}} }}$'].isin([0, 2,5,10])]
    cmap = ree.sr_to_color_dict(this_df["$\\Delta_{{\\text{{Target}}}}$"])
    this_df["cvar_tgt_str"] = this_df["$\\Delta_{{\\text{{Target}}}}$"].apply(str)
    this_df = this_df.sort_values(["$\\Delta_{{\\text{{Target}}}}$", "$N_{{ \\text{{obs}} }}$"])
    # plot
    fig = px.line(
        this_df,
        x = "Relative Root MSE",
        y="Cost Mean",
        facet_col="Method",
        facet_row="$N_{{ \\text{{obs}} }}$",
        color_discrete_map=cmap,
        color="cvar_tgt_str",
        hover_name='Success Rate',
        log_x=True,
        log_y=True,
        markers=True,
        labels=LATEX_TO_HTML | {"cvar_tgt_str": LATEX_TO_HTML[DF_COLUMNS_TO_LATEX["cvar_tgt"]]})
    # add benchmark
    this_df_bm = df_bm_agg.query("Problem == @prob & cvar_tgt == 1")
    num_rows = len(this_df["$N_{{ \\text{{obs}} }}$"].unique())
    num_cols = len(this_df["Method"].unique())
    for bm_solver in this_df_bm.Solver.unique():
        dat =this_df_bm.query("Solver == @bm_solver")
        trace_dict = {
            "x" : dat["Relative Root MSE"],
            "y" : dat["Cost Mean"],
            "legendgrouptitle_text": "Benchmark Methods",
            "name": bm_solver,
            "legendgroup": "group",
            "mode": "markers+lines",
            "opacity": 0.8
        }
        trace_dict = trace_dict | BM_SOLVER_SCATTER_STYLE[bm_solver]
        fig = ree.add_scatter_to_subplots(fig, num_rows, num_cols, **trace_dict)
    # style figure
    fig.update_layout(**MY_LAYOUT)
    fig.update_layout(**{"width":700,
    "height":800})
    # remove column heading
    fig.for_each_annotation(
        lambda a: a.update(text="" if a.text.startswith("Method") else a.text))
    # overwrite N_obs = 0
    old_a = LATEX_TO_HTML[DF_COLUMNS_TO_LATEX["observation_window"]] + "=0"
    new_a = "No Divergence Check"
    fig.for_each_annotation(
        lambda a: a.update(text = new_a if a.text.startswith(old_a) else a.text))
    # save and show
    fig.write_image(f"{prob} stopping criterion.png".replace(" ", "_").lower(), scale=WRITE_SCALE)
    fig.show()
    # make and save caption
    fig_description = f"Solving the {prob} with the CBREE method using  \
different parameters. \
We vary the stopping criterion $\\Delta_{{\\text{{Target}}}}$ (color) and \
the length of the observation window $N_\\text{{obs}}$ (row). \
The parameter $\\epsilon_{{\\text{{Target}}}} = {best_tolerance}$ \
and the choice of the indicator approximation {INDICATOR_APPROX_LATEX_NAME[best_approximation]} \
are fixed. \
Furthermore we plot also the performance of the benchmark methods EnKF \
and SiS. \
We used the sample sizes $J \\in {ree.vec_to_latex_set(df_agg['Sample Size'].unique())}$. \
Each marker represents the empirical estimates based the successful portion of $200$ simulations."
    with open(f"{prob} criterion_desc.tex".replace(" ", "_").lower(), "w") as file:
        file.write(fig_description)
    display(Markdown(fig_description))
```

Skip to main content

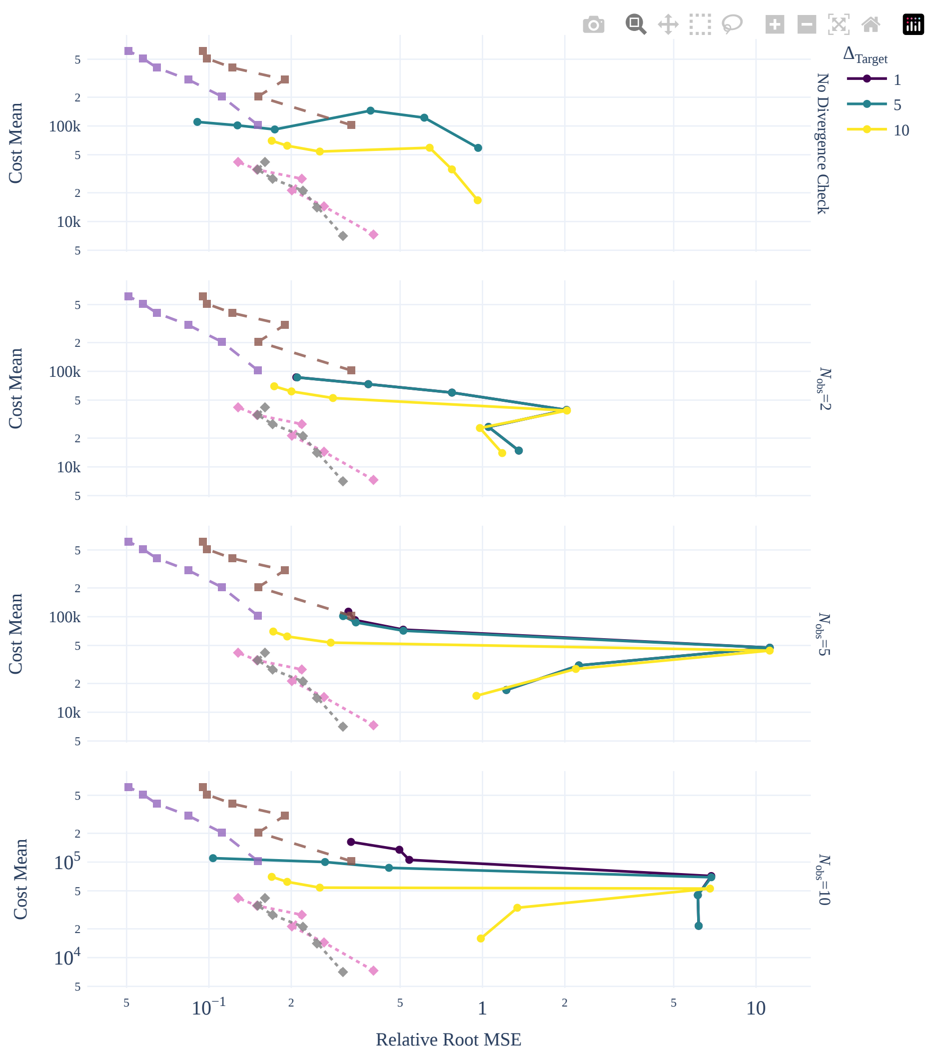


Solving the Convex Problem with the CBREE method using different parameters. We vary the stopping criterion Δ_{Target} (color) and the length of the observation window N_obs (row). The parameter $\epsilon_{\text{Target}} = 0.5$ and the choice of the indicator approximation $I_\text{{alg}}$ are fixed. Furthermore we plot also the performance of the benchmark methods EnKF and SiS. We used the sample sizes $J \in {1000, 2000, \ldots, 6000}$. Each marker represents the empirical estimates based the successful portion of 200 simulations.

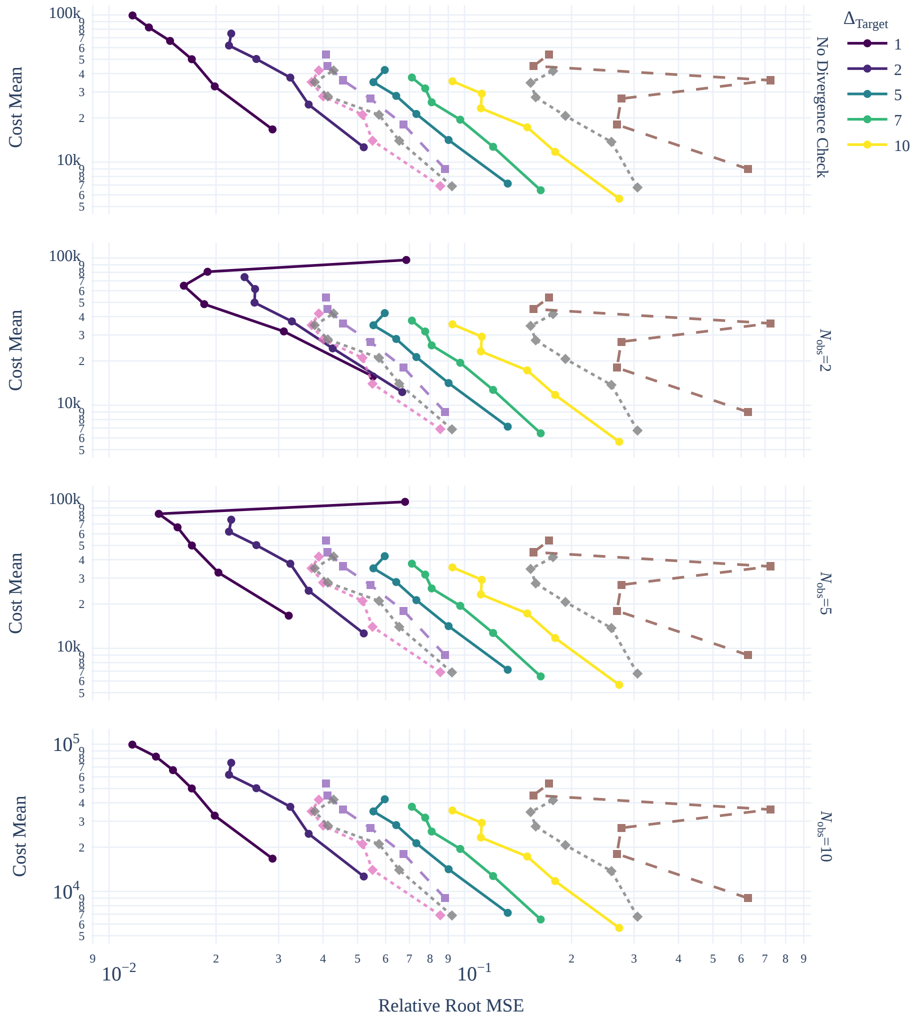




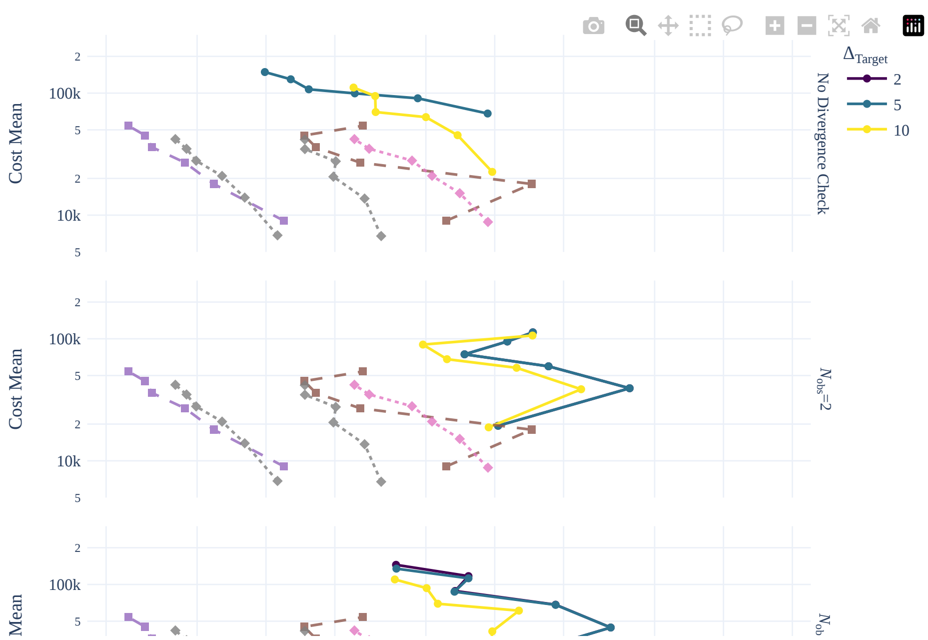
Solving the Fujita Rackwitz Problem (d=2) with the CBREE method using different parameters. We vary the stopping criterion Δ_{Target} (color) and the length of the observation window N_obs (row). The parameter $\epsilon_{\text{Target}} = 0.5$ and the choice of the indicator approximation $I_\text{{alg}}$ are fixed. Furthermore we plot also the performance of the benchmark methods EnKF and SiS. We used the sample sizes $J \in {1000, 2000, \ldots, 6000}$. Each marker represents the empirical estimates based the successful portion of 200 simulations.

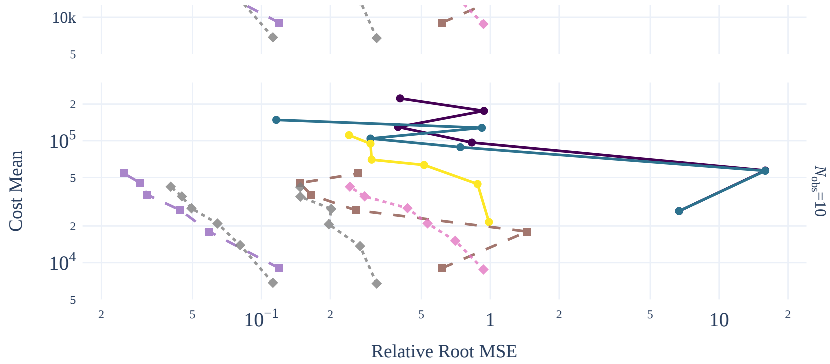


Solving the Fujita Rackwitz Problem (d=50) with the CBREE method using different parameters. We vary the stopping criterion Δ_{Target} (color) and the length of the observation window N_obs (row). The parameter $\epsilon_{\text{Target}} = 0.5$ and the choice of the indicator approximation $I_\text{{alg}}$ are fixed. Furthermore we plot also the performance of the benchmark methods EnKF and SiS. We used the sample sizes $J \in {1000, 2000, \ldots, 6000}$. Each marker represents the empirical estimates based the successful portion of 200 simulations.



Solving the Linear Problem (d=2) with the CBREE method using different parameters. We vary the stopping criterion Δ_{Target} (color) and the length of the observation window N_obs (row). The parameter $\epsilon_{\text{Target}} = 0.5$ and the choice of the indicator approximation $I_\text{{alg}}$ are fixed. Furthermore we plot also the performance of the benchmark methods EnKF and SiS. We used the sample sizes $J \in {1000, 2000, \ldots, 6000}$. Each marker represents the empirical estimates based the successful portion of 200 simulations.





Solving the Linear Problem (d=50) with the CBREE method using different parameters. We vary the stopping criterion Δ_{Target} (color) and the length of the observation window N_obs (row). The parameter $\epsilon_{\text{Target}} = 0.5$ and the choice of the indicator approximation $I_\text{{alg}}$ are fixed. Furthermore we plot also the performance of the benchmark methods EnKF and SiS. We used the sample sizes $J \in {1000, 2000, \ldots, 6000}$. Each marker represents the empirical estimates based the successful portion of 200 simulations.

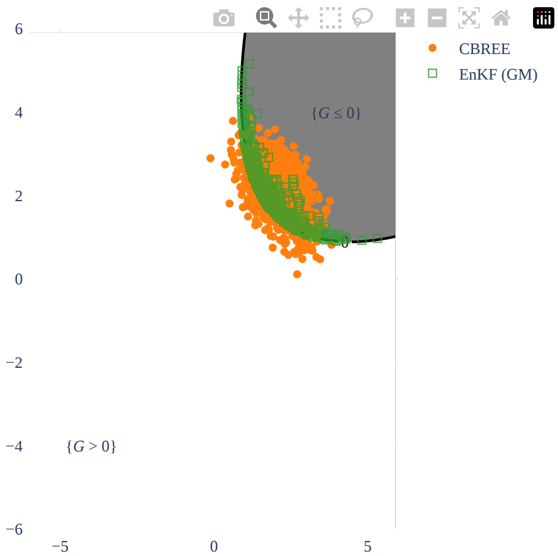
Plot failure domain for toy problems

```
from os import path
import rareeventestimation as ree
import numpy as np
import plotly.express as px
from rareeventestimation.evaluation.constants import INDICATOR_APPROX_LATEX_NAME, BM_SOLVER_SCATTER_STYLI
import plotly.graph_objects as go
from IPython.display import display, Markdown
# recommended: use autoreload for development: https://ipython.readthedocs.io/en/stable/config/extensions
%load_ext autoreload
%autoreload 2
```

Skip to main content

```
# problem and solver stuff
cvar_tgt= 1
plot_fitted_enkf_sample = False
sample_size = 1000
problem_list = [ree.prob_convex]
methods = [ree.CBREE(seed=1, cvar_tgt=cvar_tgt, divergence_check=False),  ree.ENKF(seed=1, cvar_tgt=cvar_
marker_shape_list = ["circle", "square-open", "cross-open"]
#figure stuff
annotation_anchors = [[4,4], [4,4], [-4,-4]]
delta = 0.1
x0 = -6
xx = np.arange(x0,-x0, delta)
yy = np.arange(x0,-x0, delta)
col_scale = [[0, "grey"], [1, "white"]]
contour_style = {"start": 0, "end": 0, "size": 0, "showlabels": True}
for i, prob in enumerate(problem_list):
    fig = go.Figure()
    # contour plot
    zz_lsf = np.zeros((len(yy), len(xx)))
    for (xi, x) in enumerate(xx):
        for(yi, y) in enumerate(yy):
            z = np.array([x, y])
            zz_lsf[yi, xi] = prob.lsf(z)
    c_lsf = go.Contour(z=zz_lsf, x=xx, y=yy, colorscale=col_scale,
                       contours=contour_style, line_width=2, showscale=False, showlegend=False)
    fig.add_trace(c_lsf)

    # scatter
    for j, solver in enumerate(methods):
        prob.set_sample(sample_size, seed=1)
        sol = solver.solve(prob)
        normal_sample = (plot_fitted_enkf_sample and str(solver) == "EnKF (GM)") or str(solver) != "EnKF
        xx = sol.ensemble_hist[-1,:,0] if normal_sample else sol.other["Final Iteration"][-1,:,0]
        yy = sol.ensemble_hist[-1,:,1] if normal_sample else sol.other["Final Iteration"][-1,:,1]
        sc = go.Scatter(
            x = xx,
            y= yy,
            name= str(solver),
            mode="markers",
            opacity=1 if j==0 else 0.8,
            marker_symbol = marker_shape_list[j],
        )
        fig.add_trace(sc)
    # style
    fig.update_layout(**MY_LAYOUT)
    fig.update_layout(height=450, width=450,
                      xaxis_range = [x0, -x0],
                      yaxis_range = [x0, -x0])
    fig.add_annotation(x=annotation_anchors[i][0],
                       y=annotation_anchors[i][1],
                       ay=0,
                       ax=0,
                       text="{<i>G</i> \u2264 0}")
    fig.add_annotation(x=-annotation_anchors[i][0],
                       y=-annotation_anchors[i][1],
                       ay=0,
                       ax=0,
                       text="{<i>G</i> > 0}")
    # save and show figure
    fig_name = f"{prob.name} scatter plot{'' if plot_fitted_enkf_sample else 'no enkf fit '}".replace(" 
    fig.write_image(fig_name +".png", scale=WRITE_SCALE)
    fig.show()
    # make and save caption
    fig_description = f"Failure domain of the {prob.name}. \
Also the final ensembles of the CBREE, SiS and EnKF methods respectively are plotted. " +\
("" if plot_fitted_enkf_sample else "Note that for the EnKF method this is not the sample fitted to last
f"Each method used $J={sample_size}$ samples and the stopping criterion $\\Delta_{{\\text{{Target}}}}$ =
The CBREE method performed no divergence check, used the approximation $I_\\text{{alg}}$, the stepsize c
    with open(fig_name + "_desc.tex", "w") as file:
        file.write(fig_description)
    display(Markdown(fig_description))
```

```
adaptive: True
```



Failure domain of the Convex Problem. Also the final ensembles of the CBREE, SiS and EnKF methods respectively are plotted. Note that for the EnKF method this is not the sample fitted to last particle ensemble of the internal iteration, which is used for importance sampling. Each method used $J=1000$ samples and the stopping criterion Δ_{Target} = 1. The CBREE method performed no divergence check, used the approximation I_alg, the stepsize control $\epsilon_{\text{Target}}=0.5$ and controled the increase of σ with $\text{Lip}(\sigma) = 1$.

Visualize the divergence check

```
from copy import deepcopy
import rareeventestimation as ree
import numpy as np
from rareeventestimation.evaluation.constants import *
import plotly.graph_objects as go
from plotly.subplots import  make_subplots
from IPython.display import display, Markdown
# recommended: use autoreload for development: https://ipython.readthedocs.io/en/stable/config/extension
%load_ext autoreload
%autoreload 2
```

Solve toy problem

```
cvar_tgt = .1
stepsize_tolerance = 4
J = 2500
lip_sigma = 2
prob = ree.make_linear_problem(10)
prob.set_sample(J, seed=J)
solver = ree.CBREE(seed=1,
             divergence_check=False,
             cvar_tgt=cvar_tgt,
             num_steps=250,
             save_history=True,
             lip_sigma=2,
             return_caches=True,
             return_other=True)
sol = solver.solve(prob)
```

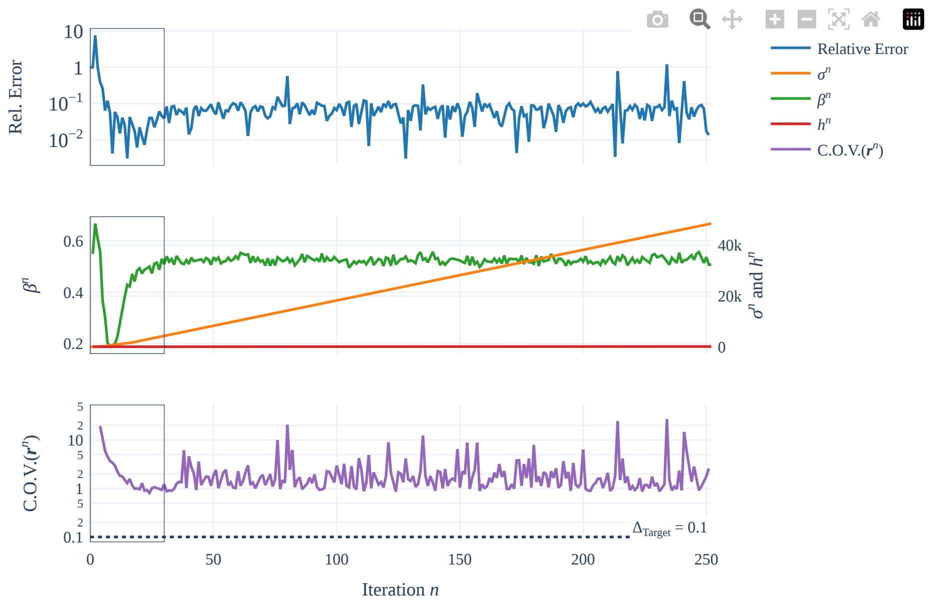
```
adaptive: True
```

Skip to main content

Plot results

```
n_cut = 30
# make figure
fig = make_subplots(rows=3,
                    cols=1,
                    shared_xaxes=True,
                    specs=[[{"secondary_y": False}],
                           [{"secondary_y": True}],
                           [{"secondary_y": False}]])
fig_name = "divergence_check"
# add error
fig.add_trace(
    go.Scatter(
        y = sol.get_rel_err(prob = prob),
        name="Relative Error"
    ),
    row=1,
    col=1)
# add parameters
params ={
    "sigma": STR_SIGMA_N,
    "beta": STR_BETA_N,
    "t_step": STR_H_N
}
for p, p_name in params.items():
    secondary = p != "beta"
    if p=="t_step":
        yy = -np.log(sol.other[p])
    else:
        yy = sol.other[p]
    fig.add_trace(
        go.Scatter(
            y=yy,
            name = p_name
        ),
        row = 2,
        col = 1,
        secondary_y=secondary)

# add cvar + goal
fig.add_trace(
    go.Scatter(
        y = sol.other["cvar_is_weights"],
        name = "C.O.V.(<i><b>r</b><sup>n</sup></i>)"
    ),
    row = 3,
    col = 1,)
fig.add_hline(y=cvar_tgt,
              line_dash="dot",
              annotation_text=f"\u0394<sub>Target</sub> = {cvar_tgt}",
              annotation_position="bottom right",
              annotation_y=cvar_tgt-0.65,
              annotation_bgcolor="white",
              row=3,

              col=1)

# Style figure
fig.update_yaxes(title_text="Rel. Error", type="log", row=1, col=1)
fig.update_yaxes(title_text="C.O.V.(<i><b>r</b><sup>n</sup></i>)", row=3, col=1, type="log")
fig.update_yaxes(title_text=f"{STR_SIGMA_N} and {STR_H_N}", title_standoff=10, row=2, col=1, secondary_y=
fig.update_yaxes(title_text=STR_BETA_N, row=2, col=1, secondary_y=False)
fig.update_xaxes(title_text="Iteration <i>n<i>", row=3, col=1)
fig.update_layout(**MY_LAYOUT)
fig2 = deepcopy(fig)
fig.add_vrect(0,n_cut, line_width=0.5)
# save
fig.write_image(fig_name + ".png", scale =WRITE_SCALE)
fig.show()
# make and save caption
fig_description = f"Solving the {prob.name} with the CBREE method using  \
$J = {J}$ particles, \
the stopping criterion $\\Delta_{{\\text{{Target}}}} = {cvar_tgt}$, \
the stepsize tolerance $\\epsilon_{{\\text{{Target}}}} = {solver.stepsize_tolerance}$, \
controlling the increase of $\\sigma$ with $\\text{{Lin}}(\\sigma) = {solver.lin_sigma}$ \
un]}."
```

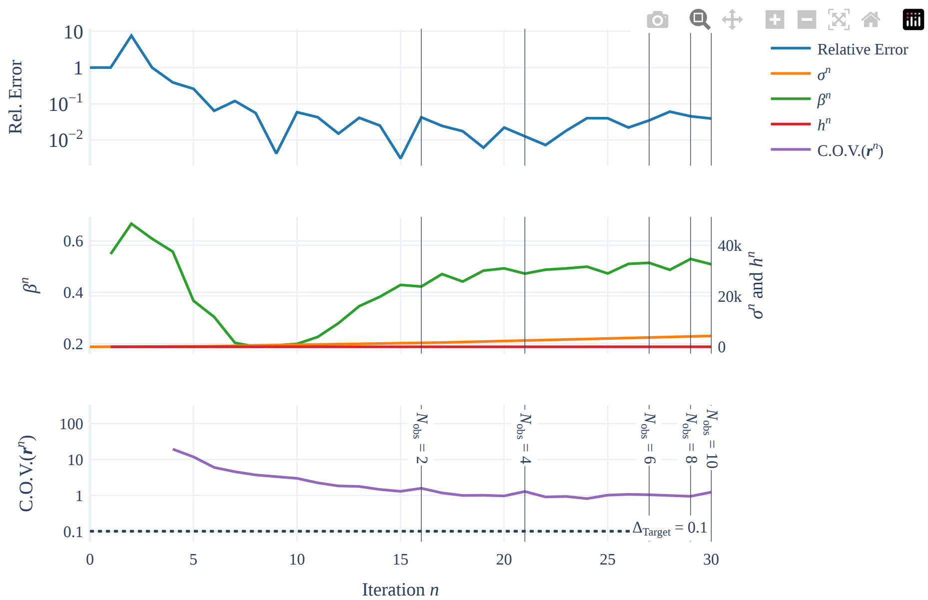
```
    file.write(fig_description)
display(Markdown(fig_description))
```



Solving the Linear Problem (d=10) with the CBREE method using $J = 2500$ particles, the stopping criterion $\Delta_{\text{Target}} = 0.1$, the stepsize tolerance $\epsilon_{\text{Target}} = 0.5$, controlling the increase of σ with $\text{Lip}(\sigma) = 2$ and approximating the indicator function with $I_\text{{alg}}$.

Zoom in

```
fig2.update_xaxes(range=[0,n_cut])
# add stops of divergence check
kk = range(2,12,2)
label_xx = np.zeros(0)
label_yy = np.zeros(0)
label_text = np.zeros(0)
# mark points where divergence check is triggered
for i,k in enumerate(kk):
    solver.divergence_check = True
    solver.observation_window = k
    sol_ref = solver.solve_from_caches(deepcopy(sol.other["cache_list"]))
    n = sol_ref.num_steps
    fig2.add_vline(
        x = n,
        line_width = 0.5)
    if n not in label_xx:
        label_xx = np.append(label_xx, n)
        label_yy = np.append(label_yy, 10)
        label_text =  np.append(label_text, f"<i>N</i><sub>obs</sub> = {k}")
    else:
        for (idx, x) in enumerate(label_xx):
            if n==x:
                label_text[idx] = f"{label_text[idx]}, {k}"
# style
for i, txt in enumerate(label_text):
    fig2.add_annotation(
        text=txt,
        x=label_xx[i],
        y=np.log(5),
        bgcolor="rgba(2550,255,255,1)",
        textangle=90,
        xref="x",
        yref="y4",
        showarrow=False,
        align="right"
    )
fig2.update_yaxes(range = [0.05, 1.2*np.amax(sol.other["cvar_is_weights"])], row=3, col=1)
fig2.show()
fig2.write_image(fig_name + "_zoom.png", scale =WRITE_SCALE)
```



Visualize effect of resampling final ensemble

```
from os import path
import rareeventestimation as ree
import numpy as np
import pandas as pd
import plotly.express as px
from rareeventestimation.evaluation.constants import INDICATOR_APPROX_LATEX_NAME, BM_SOLVER_SCATTER_STYLI
import plotly.graph_objects as go
from IPython.display import display, Markdown
# recommended: use autoreload for development: https://ipython.readthedocs.io/en/stable/config/extension:
%load_ext autoreload
%autoreload 2
```

Load Data

Option 1: Get precomputed data online

```
# data is here: https://archive.org/details/konstantinalthaus-rareeventestimation-data
# you can got to this link and inspect the files before loading
df_agg = pd.read_json("https://archive.org/download/konstantinalthaus-rareeventestimation-data/resampling
df_ess= pd.read_json("https://archive.org/download/konstantinalthaus-rareeventestimation-data/effective_:
```

Skip to main content

Option 2: Aggregate locally precomputed data

```
## uncomment to load existing data
## or to compile data after computing it yourself:
# if not path.exists(path.join(out_dir, "processed_data.json")):
#     df = ree.load_data(out_dir, pattern)
#     # Nice solver names
#     df.loc[df["callback"].isna(),"Solver"] = "CBREE"
#     df.loc[df["callback"].isna(),"callback"] = "None"
#     df.loc[df["callback"].str.contains("gm"), "Solver"] = "CBREE (G)"
#     df = df.loc[~df["callback"].str.contains("vmfnm"),:].reset_index()
#     df = ree.add_evaluations(df)
#     df_agg = ree.aggregate_df(df)
#     df_agg.to_json(path.join(out_dir, "processed_data.json"))
# else:
#     df_agg = pd.read_json(path.join(out_dir, "processed_data.json"))

# if not  path.exists(path.join(out_dir, "ess_data.json")):
#     df = ree.load_data(out_dir, pattern)
#     # Nice solver names
#     df.loc[df["callback"].isna(),"Solver"] = "CBREE"
#     df.loc[df["callback"].isna(),"callback"] = "None"
#     df.loc[df["callback"].str.contains("gm"), "Solver"] = "CBREE (G)"
#     df = df.loc[~df["callback"].str.contains("vmfnm"),:].reset_index()
#     df = ree.add_evaluations(df)
#     df["VAR IS Weights"] = (df["Estimate"] * df["cvar_is_weights"] )**2
#     df["J_ESS"] = df["VAR IS Weights"] / df["Estimate Variance"]
#     df["J_ESS"] = df.apply(lambda x: x["J_ESS"][-1], axis=1)
#     df_ess = df[["Problem", "Solver", "Sample Size", "J_ESS"]]
#     df_ess.to_json(path.join(out_dir, "ess_data.json"))
# else:
#     df_ess = pd.read_json(path.join(out_dir, "ess_data.json"))
```

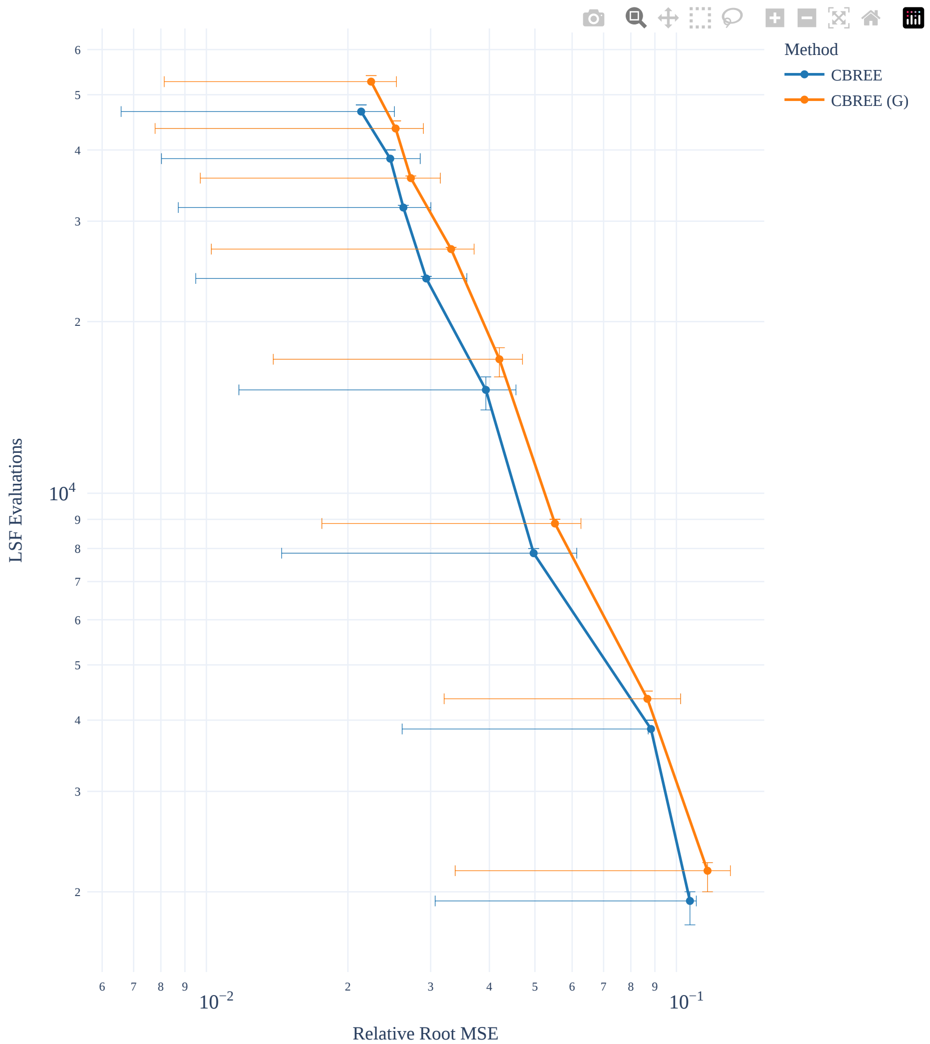
Make figures

Error-Cost plot

```
# data from creation
solver = ree.CBREE()
# plot
figs = ree.make_accuracy_plots(df_agg, layout=MY_LAYOUT, CMAP=CMAP)
fig = figs[0]
fig_name="resampling_in_final_step"
fig.update_yaxes(title_text = "LSF Evaluations")
fig.update_layout(title_text = "", height=800)
fig.write_image(fig_name + ".png",scale=WRITE_SCALE)
fig.show()

# make and save caption
fig_description = f"Solving the {df_agg.Problem.unique()[0]} with two CBREE methods using  \
$J \\in \\{{{', '.join(map(str, df_agg['Sample Size'].unique()))}\\}}$ particles, \
the stopping criterion $\\Delta_{{\\text{{Target}}}} = {solver.cvar_tgt}$, \
the stepsize tolerance $\\epsilon_{{\\text{{Target}}}} = {solver.stepsize_tolerance}$, \
controlling the increase of $\\sigma$ with $\\text{{Lip}}(\\sigma) = {solver.lip_sigma}$ \
and approximating the indicator function with {INDICATOR_APPROX_LATEX_NAME[solver.tgt_fun]}. \
No divergence check has been performed. \
Each simulation was repeated 200 times. \
While the markers present the empirical means of the visualized quantities, the error bars are drawn fro
with open(fig_name + "_desc.tex", "w") as file:
    file.write(fig_description)
display(Markdown(fig_description))
```

Skip to main content

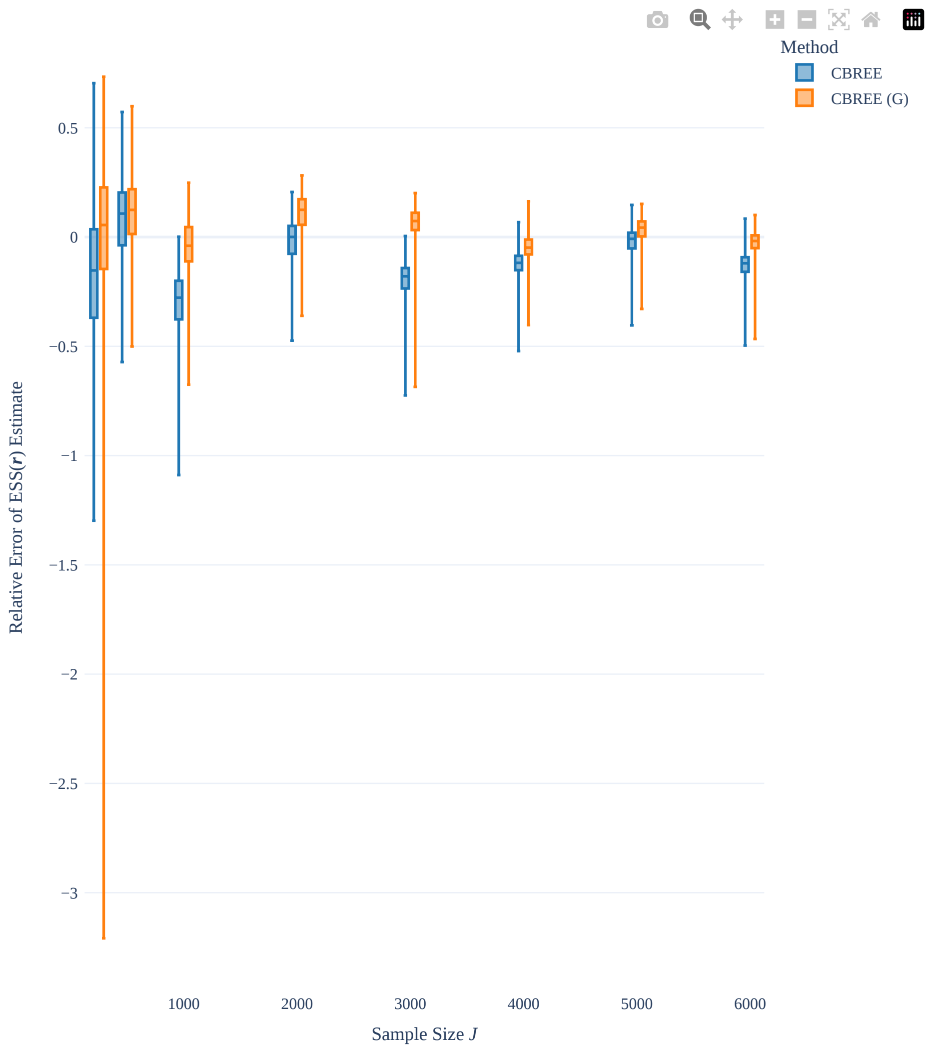
```
adaptive: True
```



Solving the Convex Problem with two CBREE methods using $J \in {250, 500, 1000, 2000, 3000, 4000, 5000, 6000}$ particles, the stopping criterion $\Delta_{\text{Target}} = 2$, the stepsize tolerance $\epsilon_{\text{Target}} = 0.5$, controlling the increase of σ with $\text{Lip}(\sigma) = 1$ and approximating the indicator function with $I_\text{{alg}}$. No divergence check has been performed. Each simulation was repeated 200 times. While the markers present the empirical means of the visualized quantities, the error bars are drawn from first to the third quartile.

Study correlation of importace function evaluations

```
# sort
df_ess["J_ESS"] =(df_ess["Sample Size"] -  df_ess.J_ESS) / df_ess["Sample Size"]
df_ess.sort_values(by="Solver", inplace = True)
# plot
fig_hist = px.box(df_ess,
                  x = "Sample Size",
                        y = "J_ESS",
                        color="Solver",
                        points=False,
                        color_discrete_sequence = CMAP,
                        labels={"Solver": "Method"})
# style and save
fig_hist.update_layout(**MY_LAYOUT)
fig_hist.update_layout(height=800)
fig_hist.update_xaxes(title_text = "Sample Size <i>J</i>")
fig_hist.update_yaxes(title_text = f"Relative Error of ESS(<b><i>r</b></i>) Estimate")
fig_hist.write_image(fig_name + "_boxplot.png",scale=WRITE_SCALE)
```



Visualize the Performance of the CBREE (vMFN) Method

```
from os import path
import rareeventestimation as ree
import pandas as pd
import plotly.express as px
from rareeventestimation.evaluation.constants import INDICATOR_APPROX_LATEX_NAME, BM_SOLVER_SCATTER_STYLI
import plotly.graph_objects as go
from IPython.display import display, Markdown
# recommended: use autoreload for development: https://ipython.readthedocs.io/en/stable/config/extension
%load_ext autoreload
%autoreload 2
```

Load Data

Option 1: Get precomputed data online

```
# data is here: https://archive.org/details/konstantinalthaus-rareeventestimation-data
# you can got to this link and inspect the files before loading
df_agg = pd.read_json("https://ia801504.us.archive.org/23/items/konstantinalthaus-rareeventestimation-dat
df_bm_agg = pd.read_json("https://archive.org/download/konstantinalthaus-rareeventestimation-data/benchma
```

Skip to main content

Option 2: Aggregate locally precomputed data

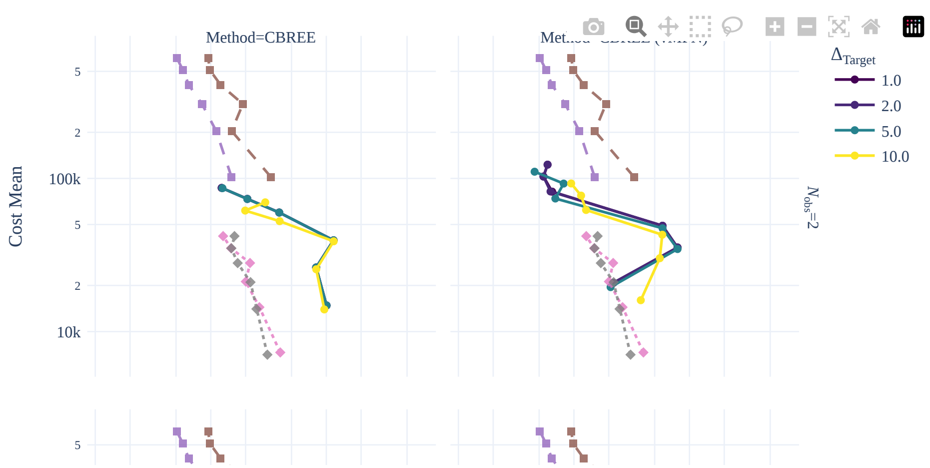
```
## uncomment to load existing data
## or to compile data after computing it yourself:
# data_dir ="docs/benchmarking/data/cbree_sim/toy_problems_resampled"
# df_path =path.join(data_dir, "resampled_toy_problems.json")
# if not path.exists(df_path):
#     # load and clean
#     df = ree.load_data(data_dir, "*")
#     df.drop(columns=["index", "Unnamed: 0", "VAR Weighted Average Estimate","CVAR"], inplace=True, erro
#     df.drop_duplicates(inplace=True)
#     df.reset_index(drop=True, inplace=True)
#     # Pretty names
#     to_drop = ["callback"] # no callbacks here
#     df.rename(columns={"mixture_model": "Method"}, inplace=True)
#     replace_values = {"Method": {"GM": "CBREE", "vMFNM": "CBREE (vMFN)"}}
#     df = df.drop(columns=to_drop) \
#         .rename(columns=DF_COLUMNS_TO_LATEX) \
#         .replace(replace_values)
#     # melt aggregated estimates into long format
#     df = df.rename(columns={"Estimate": "Last Estimate"})\
#         .melt(id_vars = [c for c in df.columns if not "Estimate" in c],
#               var_name="Averaging Method",
#               value_name="Estimate")
#     # make solver column with unique names wrt all options.
#     df = df.apply(expand_cbree_name, axis=1, columns= [DF_COLUMNS_TO_LATEX['observation_window'], "Ave
#     df = ree.add_evaluations(df,  only_success=True)
#     # %% aggregate
#     df_agg = ree.aggregate_df(df)
#     # save
#     df_agg.to_json(df_path)
# else:
#    df_agg = pd.read_json(df_path)
# # load benchmark
# df_bm, df_bm_agg = ree.get_benchmark_df()
```

Make Figures

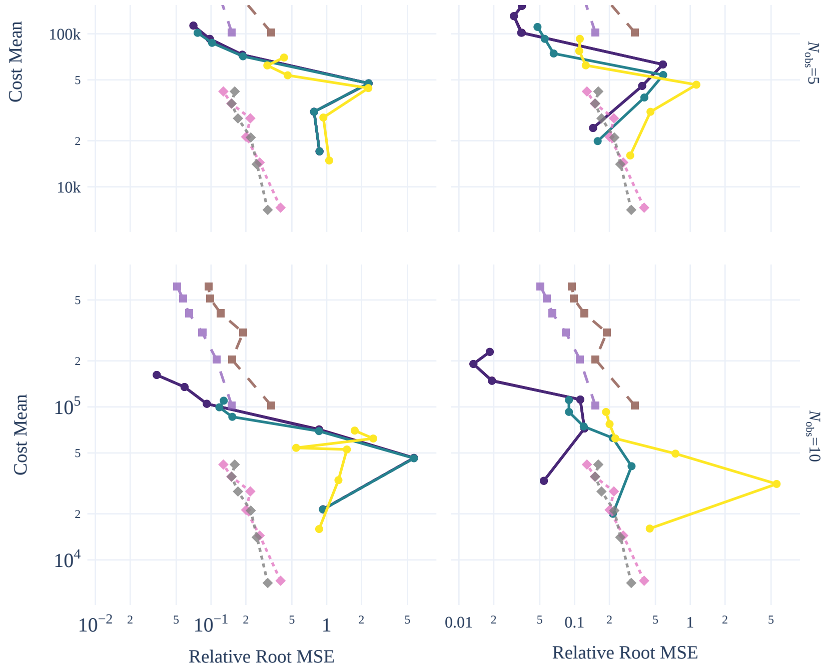
```
for prob in df_agg.Problem.unique():
    # filter
    this_df = df_agg.query("Problem == @prob & `Averaging Method`=='Average Estimate'")
    this_df = this_df[this_df['$N_{{ \\text{{obs}} }}$'].isin([2,5,10])]
    cmap = ree.sr_to_color_dict(this_df["$\\Delta_{{\\text{{Target}}}}$"].astype(float))
    this_df["cvar_tgt_str"] = this_df["$\\Delta_{{\\text{{Target}}}}$"].astype(float).apply(str)
    this_df = this_df.sort_values(["$\\Delta_{{\\text{{Target}}}}$", "$N_{{ \\text{{obs}} }}$", "Sample S
    # plot
    fig = px.line(
        this_df,
        x = "Relative Root MSE",
        y="Cost Mean",
        facet_col="Method",
        facet_row="$N_{{ \\text{{obs}} }}$",
        color_discrete_map=cmap,
        color="cvar_tgt_str",
        log_x=True,
        log_y=True,
        markers=True,
        hover_name="Sample Size",
        labels=LATEX_TO_HTML | {"cvar_tgt_str": LATEX_TO_HTML[DF_COLUMNS_TO_LATEX["cvar_tgt"]]})
    # add benchmark
    this_df_bm = df_bm_agg.query("Problem == @prob & cvar_tgt == 1")
    num_rows = len(this_df["$N_{{ \\text{{obs}} }}$"].unique())
    num_cols = len(this_df["Method"].unique())
    for bm_solver in this_df_bm.Solver.unique():
        dat =this_df_bm.query("Solver == @bm_solver")
        trace_dict = {
            "x" : dat["Relative Root MSE"],
            "y" : dat["Cost Mean"],
            "legendgrouptitle_text": "Benchmark Methods",
            "name": bm_solver,
            "legendgroup": "group",
            "mode": "markers+lines",
            "opacity": 0.8
        }
        trace_dict = trace_dict | BM_SOLVER_SCATTER_STYLE[bm_solver]
        fig = ree.add_scatter_to_subplots(fig, num_rows, num_cols, **trace_dict)
    # style
    fig.update_layout(**MY_LAYOUT)
    fig.update_layout(**{"width":700,
    "height":900})
    fig.for_each_annotation(
        lambda a: a.update(yshift =  -10 if a.text.startswith("Method") else 0)) # adjust column name pos
    # show and save
    fig.show()
    fig.write_image(f"{prob} resampled stopping criterion.png".replace(" ", "_").lower(), scale=WRITE_SCA
    # make and save caption
    fig_description = f"Solving the {prob} with the CBREE methods using  \
    different parameters. \
    We vary the stopping criterion $\\Delta_{{\\text{{Target}}}}$ (color), \
    the divergence criterion $N_\\text{{obs}}$ (row) and \
    the method  (column). \
    The parameter $\\epsilon_{{\\text{{Target}}}} = {0.5}$ \
    and the choice of the indicator approximation {INDICATOR_APPROX_LATEX_NAME['algebraic']} \
    are fixed. \
    Furthermore we plot also the performance of the benchmark methods EnKF \
    and SiS. \
    Each marker represents the empirical estimates based the successful portion of $200$ simulations."
    with open(f"{prob} resampled stopping criterion desc.tex".replace(" ", "_").lower(), "w") as file:
        file.write(fig_description)
    display(Markdown(fig_description))
```

Skip to main content

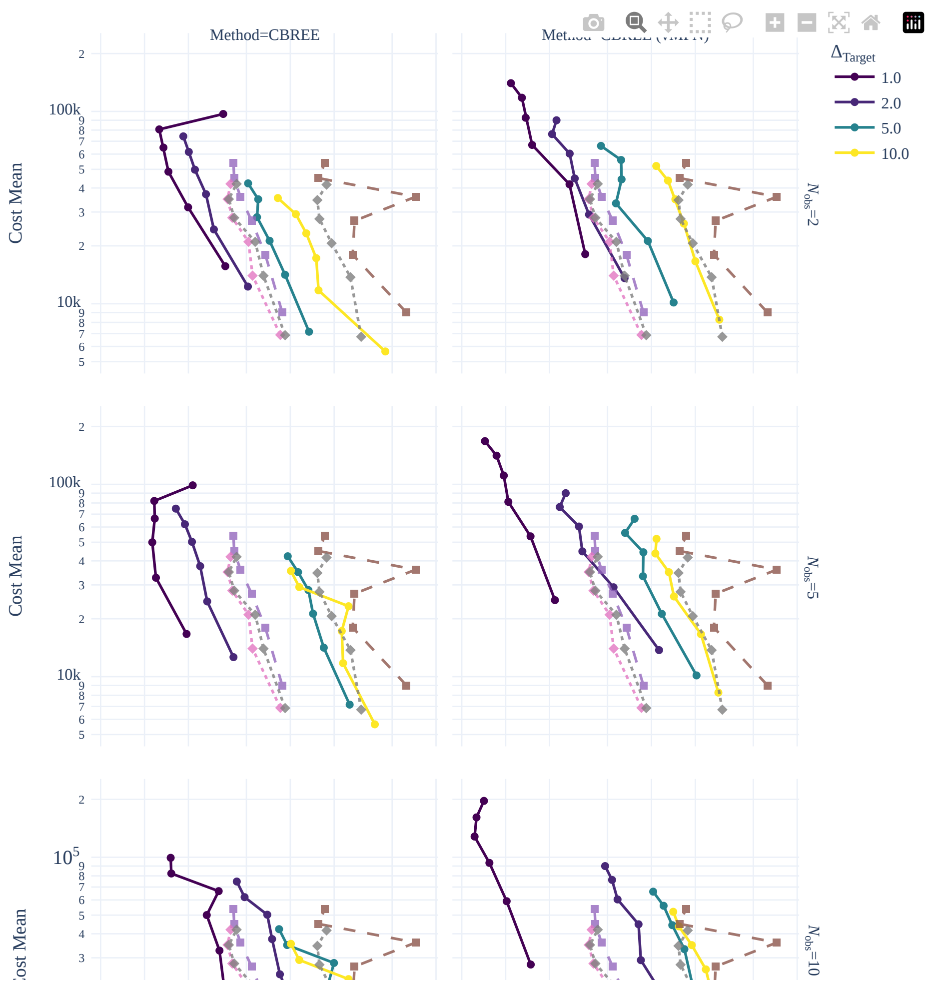
Solving the Fujita Rackwitz Problem (d=2) with the CBREE methods using different parameters. We vary the stopping criterion Δ_{Target} (color), the divergence criterion N_obs (row) and the method (column). The parameter $\epsilon_{\text{Target}} = 0.5$ and the choice of the indicator approximation $I_\text{{alg}}$ are fixed. Furthermore we plot also the performance of the benchmark methods EnKF and SiS. Each marker represents the empirical estimates based the successful portion of 200 simulations.

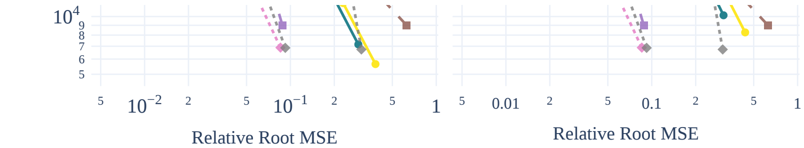


Skip to main content



Solving the Fujita Rackwitz Problem (d=50) with the CBREE methods using different parameters. We vary the stopping criterion Δ_{Target} (color), the divergence criterion N_obs (row) and the method (column). The parameter $\epsilon_{\text{Target}} = 0.5$ and the choice of the indicator approximation $I_\text{{alg}}$ are fixed. Furthermore we plot also the performance of the benchmark methods EnKF and SiS. Each marker represents the empirical estimates based the successful portion of 200 simulations.





Solving the Linear Problem (d=2) with the CBREE methods using different parameters. We vary the stopping criterion Δ_{Target} (color), the divergence criterion N_obs (row) and the method (column). The parameter $\epsilon_{\text{Target}} = 0.5$ and the choice of the indicator approximation $I_\text{{alg}}$ are fixed. Furthermore we plot also the performance of the benchmark methods EnKF and SiS. Each marker represents the empirical estimates based the successful portion of 200 simulations.

Solving the Linear Problem (d=50) with the CBREE methods using different parameters. We vary the stopping criterion Δ_{Target} (color), the divergence criterion N_obs (row) and the method (column). The parameter $\epsilon_{\text{Target}} = 0.5$ and the choice of the indicator approximation $I_\text{{alg}}$ are fixed. Furthermore we plot also the performance of the benchmark methods EnKF and SiS. Each marker represents the empirical estimates based the successful portion of 200 simulations.

Skip to main content

Visualize effect of β-adaptivity

```
import rareeventestimation as ree
import numpy as np
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import scipy as sp
from rareeventestimation.evaluation.constants import *
from IPython.display import display, Markdown
# recommended: use autoreload for development: https://ipython.readthedocs.io/en/stable/config/extensions
%load_ext autoreload
%autoreload 2
```

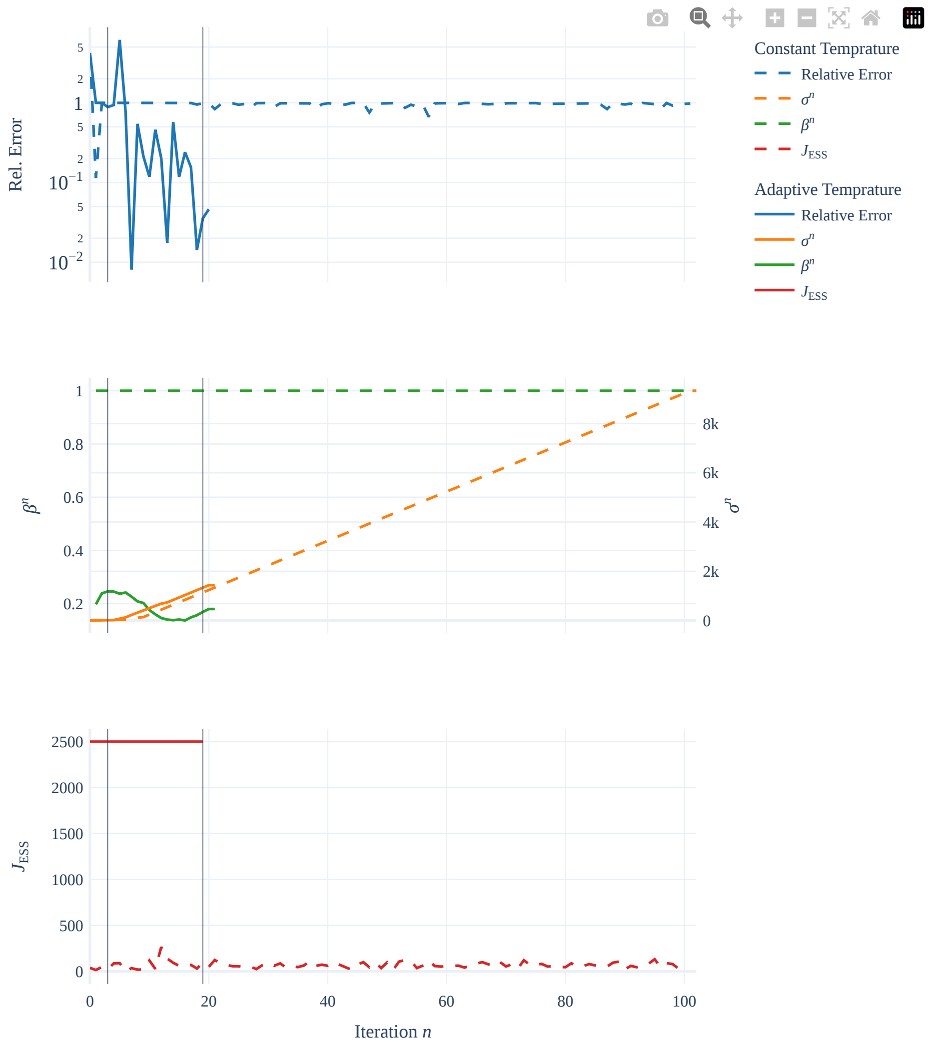
Plot error and parameters during iterations

```
# solvers
N = 5000
d=50
seed=1
cvar_tgt=5
cbree_const = ree.CBREE(beta_adaptivity=1.0,
                      save_history=True,
                      stepsize_tolerance=1,
                      seed=seed,
                      return_other = True,
                      divergence_check=False,
                      cvar_tgt=cvar_tgt,
                      return_caches = True,
                      name = "Constant Temprature",
                      )
cbree = ree.CBREE(save_history=True,
                seed=seed,
                stepsize_tolerance=1,
                return_other=True,
                divergence_check=False,
                return_caches = True,
                name = "Adaptive Temprature",
                cvar_tgt=cvar_tgt)
solver_list = [cbree_const, cbree]

# prepare figure
solver_dash = dict(zip([str(s) for s in solver_list], ["dash", "solid"]))
solver_marker = dict(zip([str(s) for s in solver_list], [1,3]))
fig = make_subplots(rows=3,
                    cols=1,
                    shared_xaxes=True,
                    specs=[[{"secondary_y": False}],
                           [{"secondary_y": True}],
                           [{"secondary_y": False}]])
fig_name = "constant_beta_iterations"

# populate figure
solution_list = []
for solver in solver_list:
    # solve
    prob = ree.make_linear_problem(d)
    prob.set_sample(N, seed = seed)
    sol = solver.solve(prob)
    sol.other["beta"][0] = np.nan
    solution_list.append(sol)
    # plot error
    fig.add_trace(
        go.Scatter(
            y=sol.get_rel_err(prob),
            name = f"Relative Error",
            line_dash=solver_dash[str(solver)],
            marker_color=CMAP[0],
            marker_symbol=solver_marker[str(solver)],
            legendgroup=str(solver),
            legendgrouptitle_text=str(solver)
        ),
        row=1,
        col=1)
    # plot parameter
    params = {
        "sigma": STR_SIGMA_N,
        "beta" : STR_BETA_N,
    }
    for i,(param,name) in enumerate(params.items()):
        fig.add_trace(
        go.Scatter(
            y=sol.other[param],
            name = name,
            line_dash=solver_dash[str(solver)],
            marker_color=CMAP[i+1],
            marker_symbol=solver_marker[str(solver)],
            legendgroup=str(solver),
        ),
        row=2
```

Skip to main content

```
    # plot effective sample size
    fig.add_trace(
        go.Scatter(
            y=sol.other["ess"],
            name = STR_J_ESS,
            line_dash=solver_dash[str(solver)],
            marker_color=CMAP[3],
            marker_symbol=solver_marker[str(solver)],
            legendgroup=str(solver),
        ),
        row=3,
        col=1)
# Add vertical lines
iters = [3,19]
for i in iters:
    fig.add_vline(i,line_width=0.5)

# style and save fig
fig.update_yaxes(title_text="Rel. Error", type="log", row=1, col=1)
fig.update_yaxes(title_text=STR_J_ESS, row=3, col=1)
fig.update_yaxes(title_text=STR_SIGMA_N, title_standoff=0, row=2, col=1, secondary_y=True)
fig.update_yaxes(title_text=STR_BETA_N, row=2, col=1, secondary_y=False)
fig.update_xaxes(title_text="Iteration <i>n<i>", row=3, col=1)
fig.update_layout(**MY_LAYOUT)
fig.update_layout(height=800)
fig.write_image(fig_name + ".png",scale=7)
fig.show()
# Make and save caption
fig_description = f"Solving the {prob.name} with the CBREE method using  \
$J = {N}$ particles, \
stopping criterion $\\Delta_{{\\text{{Target}}}} = {cvar_tgt}$, \
stepsize tolerance $\\epsilon_{{\\text{{Target}}}} = {cbree.stepsize_tolerance}$, \
controlling the increase of $\\sigma$ with $\\text{{Lip}}(\\sigma) = {cbree.lip_sigma}$ \
and approximating the indicator function with {INDICATOR_APPROX_LATEX_NAME[cbree.tgt_fun]}. \
No divergence check has been performed and the solver was able to run for at most {cbree.num_steps} itera
The vertical lines mark iterations whose weights we investigate."
with open("desc_constant_beta.tex", "w") as file:
    file.write(fig_description)
display(Markdown(fig_description))
```

```
adaptive: True
adaptive: True
```

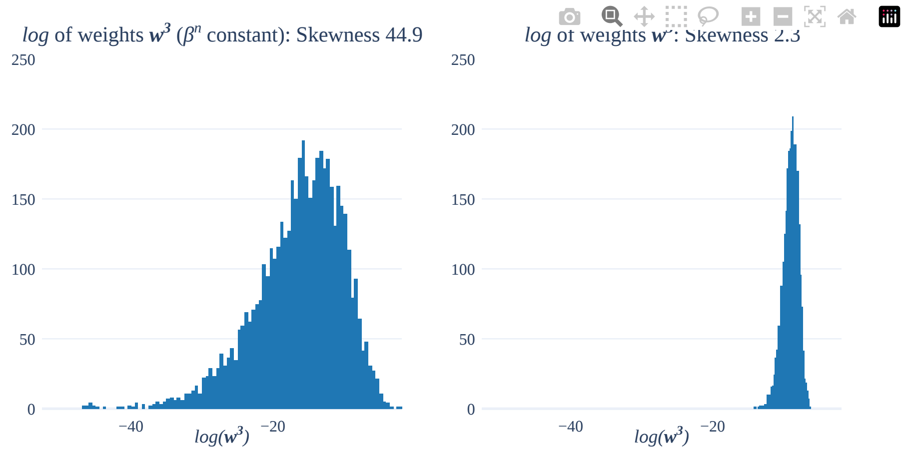


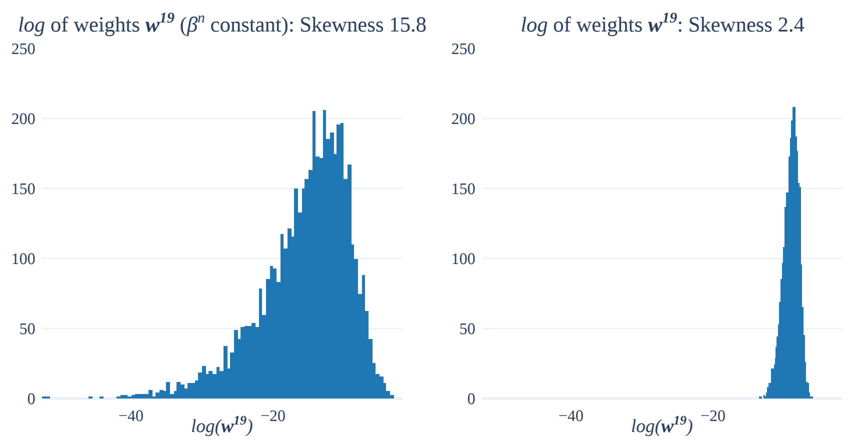
Solving the Linear Problem (d=50) with the CBREE method using $J = 5000$ particles, stopping criterion $\Delta_{\text{Target}} = 5$, stepsize tolerance $\epsilon_{\text{Target}} = 1$, controlling the increase of σ with $\text{Lip}(\sigma) = 1$ and approximating the indicator function with $I_\text{{alg}}$. No divergence check has been performed and the solver was able to run for at most 100 iterations. The vertical lines mark iterations whose weights we investigate.

Make histogram of weights for selected iterations

```
# set up figure
hist_fig_name="constant_beta_histograms"
hist_fig = make_subplots(cols=2, rows = 2,subplot_titles=("Plot 1", "Plot 2", "plot3", "plot4"))
x_lims = None
# populate
plot_counter=0
for i_n, i in enumerate(iters):
    for soln, sol in enumerate(solution_list):
        w = solver_list[soln]._CBREE__compute_weights(sol.other["cache_list"][i], return_weights=True)
        hist_name = f"<i>log</i> of weights <i><b>w<sup>{i}</sup></b></i>" +(  f" ({STR_BETA_N} constant
            if not solver_list[soln].beta_adaptivity else "") + \
            f": Skewness {sp.stats.skew(np.exp(w)):.1f}"
        hist = go.Histogram(
            x=np.log(w),
            showlegend=False,
            marker_color = CMAP[0])
        hist_fig.add_trace(hist, row=i_n+1, col=soln+1)
        # get x limits
        hist_fig.layout.annotations[plot_counter].update(text= hist_name)
        this_x_lims = np.array([np.min(np.log(w)), np.max(np.log(w))])
        if x_lims is None:
            x_lims=this_x_lims
        else:
            x_lims[0] = min(x_lims[0], this_x_lims[0])
            x_lims[1] = max(x_lims[1], this_x_lims[1])
        plot_counter += 1
# set x limits
hist_fig.update_xaxes(range=x_lims*1.01)

# add subplot title
for soln, sol in enumerate(solution_list):
    for i_n, i in enumerate(iters):
        hist_fig.update_xaxes(title=f"<i>log(<b>w<sup>{i}</sup></b>)<i>", row=i_n+1, col = soln+1,title_s
# style and save
hist_fig.update_yaxes(range=[0,250])
hist_fig.update_layout(**MY_LAYOUT)
hist_fig.update_layout(height=800)
hist_fig.update_layout(margin_t=50)
hist_fig.for_each_annotation(lambda a: a.update(y=a.y+0.01))
hist_fig.write_image(hist_fig_name + ".png",scale=WRITE_SCALE)
hist_fig.show()
```

Skip to main content





Visualize effect of problem-dimension on CBREE methods

```
from os import path
from re import sub
import rareeventestimation as ree
import numpy as np
import pandas as pd
import plotly.express as px
from rareeventestimation.evaluation.constants import INDICATOR_APPROX_LATEX_NAME, BM_SOLVER_SCATTER_STYLI
import plotly.graph_objects as go
from IPython.display import display, Markdown
# recommended: use autoreload for development: https://ipython.readthedocs.io/en/stable/config/extension
%load_ext autoreload
%autoreload 2
```

Load Data

Option 1: Get precomputed data online

```
# data is here: https://archive.org/details/konstantinalthaus-rareeventestimation-data
# you can got to this link and inspect the files before loading
df_agg= pd.read_json("https://archive.org/download/konstantinalthaus-rareeventestimation-data/dimension_
```

Skip to main content

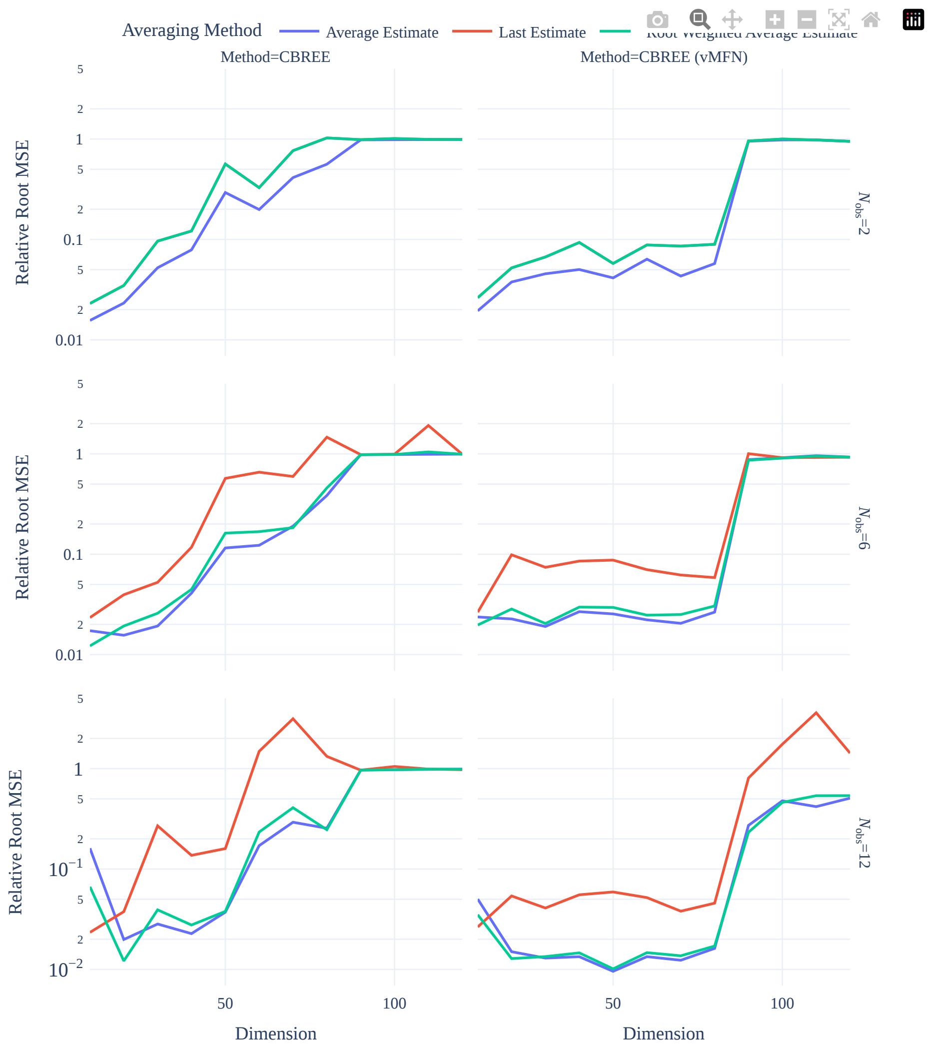
Option 2: Aggregate locally precomputed data

```
## uncomment to load existing data
## or to compile data after computing it yourself:
# data_dir ="docs/benchmarking/data/cbree_sim/dimension_study"
# df_path =path.join(data_dir, "dimension_study_data.json")
# file_pattern = "fujita_rackwitz_problem*"
# if not  path.exists(df_path):
#     df = ree.load_data(data_dir, file_pattern)
#     df.drop(columns=["index", "Unnamed: 0", "VAR Weighted Average Estimate","CVAR"], inplace=True, erro
#     df.drop_duplicates(inplace=True)
#     df.reset_index(drop=True, inplace=True)
#     df.rename(columns={"Solver":"Method"}, inplace=True)
#     # CBREE (VMFNM) does not have valid averaged estimates by definition (resample keyword has not been
#     # it mixes the last vmfnm with preceeding gaussian denisities
#     df = df.query("Method != 'CBREE (vMFNM)'")
#     # rename methods to comply with thesis notation
#     new_method_names = {"CBREE (GM)": "CBREE",
#                         "CBREE (vMFNM, resampled)": "CBREE (vMFN)"}
#     df = df.replace({"Method": new_method_names})
#     # melt aggregated estimates
#     df = df.rename(columns={"Estimate": "Last Estimate"})\
#         .melt(id_vars = [c for c in df.columns if not "Estimate" in c],
#               var_name="Averaging Method",
#               value_name="Estimate")
#     col_list = ['callback', 'observation_window', 'resample', 'mixture_model','divergence_check', "Ave
#     # make solver column with unique names wrt all options.
#     df["Solver"] = df.apply(lambda row: row["Method"] + f" ({', '.join([str(row[col]) for col in col_li
#     df = ree.add_evaluations(df,  only_success=True)
#     # %% aggregate
#     df_agg = ree.aggregate_df(df)
#     # add dimension of problem
#     df_agg["Dimension"] = df_agg["Problem"].apply(lambda x: int(sub(r"\D", "", x)))

#     #%% save
#     df_agg.to_json(df_path)
# else:
#    df_agg = pd.read_json(df_path)
```

Skip to main content

Make figure

```
# filter
this_df =df_agg[df_agg["observation_window"].isin([2,6,12])]
this_df = this_df.query("Dimension <=120 & `Sample Size` == 5000")
this_df = this_df.sort_values(["observation_window","Dimension"])
# plot
fig = px.line(this_df,
              y = "Relative Root MSE",
                x="Dimension",
                facet_col="Method",
                facet_row="observation_window",
                color="Averaging Method",
                labels={k: LATEX_TO_HTML[DF_COLUMNS_TO_LATEX[k]] for k in ["observation_window"] },
                log_y=True)
# style and save
fig.update_layout(**MY_LAYOUT)
fig.update_layout(height=800)
fig.update_layout(legend=dict(
    orientation="h",
    yanchor="bottom",
    y=1.02,
    xanchor="right",
    x=1
))
fig.write_image("dimension_study.png", scale=WRITE_SCALE)
fig.show()
# make and save caption
description = f"Solving the Fujita Rackwitz Problem in dimensions $d \\in {ree.vec_to_latex_set(this_df.
for sample size $J = {this_df['Sample Size'].unique()[0]}$  \
with different values for $N_{{ \\textup{{ obs }} }}$  (row), \
two variants of the CBREE method (column) and different averaging methods of the last \
$N_{{ \\textup{{ obs }} }}$ probability of failure estimates (color). \
Other parameters are fixed. \
Namely, we use the stopping criterion $\\Delta_{{\\text{{Target}}}} = 2$, \
the stepsize tolerance $\\epsilon_{{\\text{{Target}}}} = 0.5$, \
the increase control  of $\\sigma$ with $\\text{{Lip}}(\\sigma) = 1$ \
and approximate the indicator function with {INDICATOR_APPROX_LATEX_NAME['algebraic']}."
with open("dimension_study_desc.tex", "w") as f:
    f.write(description)
display(Markdown(description))
```



Solving the Fujita Rackwitz Problem in dimensions $d \in {10, 20, \ldots, 120}$ 200 timesfor sample size $J = 5000$ with different values for $N_{ \textup{ obs } }$ (row), two variants of the CBREE method (column) and different averaging methods of the last $N_{ \textup{ obs } }$ probability of failure estimates (color). Other parameters are fixed. Namely, we use the stopping criterion $\Delta_{\text{Target}} = 2$, the stepsize tolerance $\epsilon_{\text{Target}} = 0.5$, the increase control of σ with $\text{Lip}(\sigma) = 1$ and approximate the indicator function with $I_\text{{alg}}$.

Evaluate Performance for Diffusion Problem

```
from os import path
import rareeventestimation as ree
import pandas as pd
import plotly.express as px
from rareeventestimation.evaluation.constants import INDICATOR_APPROX_LATEX_NAME, BM_SOLVER_SCATTER_STYLI
import plotly.graph_objects as go
from IPython.display import display, Markdown
# recommended: use autoreload for development: https://ipython.readthedocs.io/en/stable/config/extension
%load_ext autoreload
%autoreload 2
```

Load Data

Option 1: Get precomputed data online

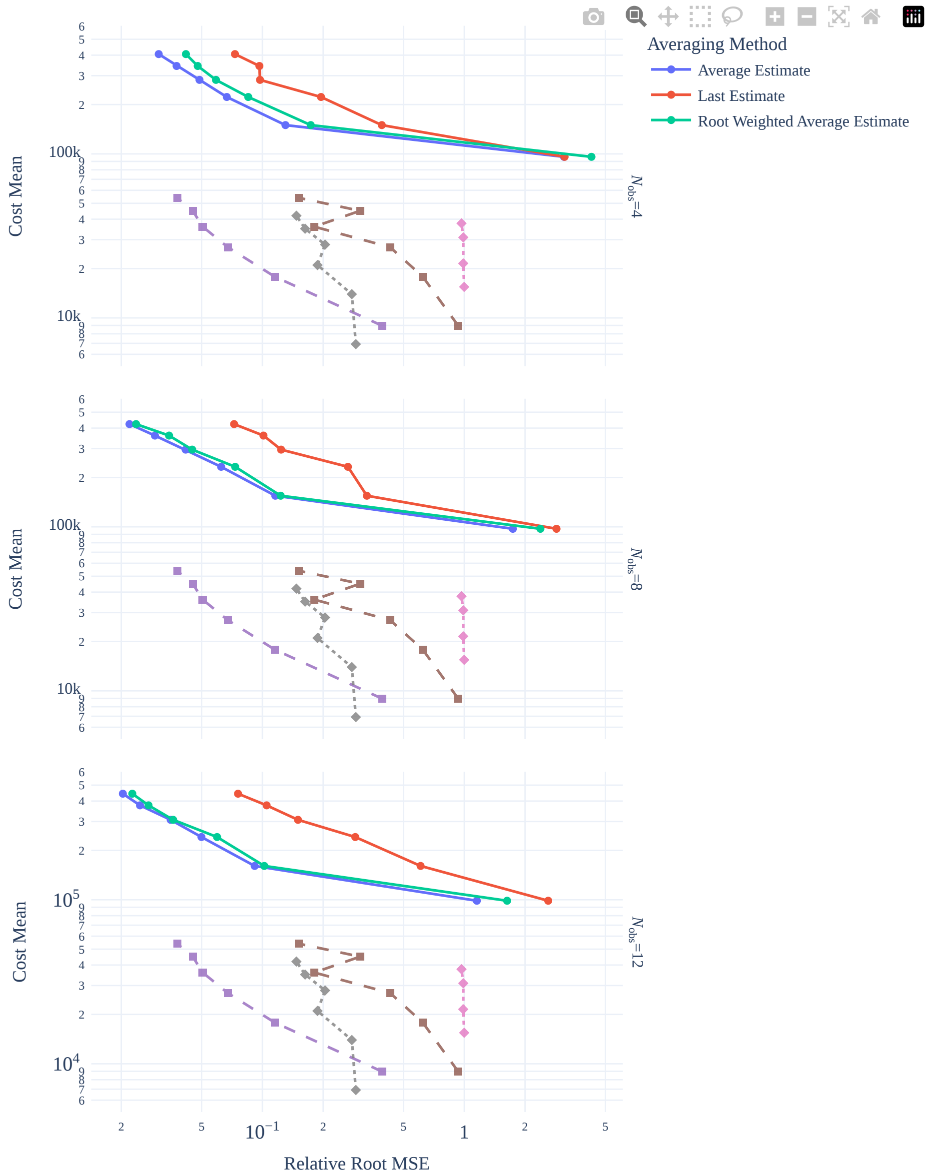
```
# data is here: https://archive.org/details/konstantinalthaus-rareeventestimation-data
# you can got to this link and inspect the files before loading
df_agg = pd.read_json("https://ia801504.us.archive.org/23/items/konstantinalthaus-rareeventestimation-dat
df_bm_agg = pd.read_json("https://ia801504.us.archive.org/23/items/konstantinalthaus-rareeventestimation
```

Option 2: Aggregate locally precomputed data

```
## uncomment to load existing data
## or to compile data after computing it yourself:
# data_dir = "docs/benchmarking/data/cbree_sim/diffusion_sim"
# path_df = path.join(data_dir, "cbree_diffusion_problem_processed.json")
# path_df_agg = path.join(data_dir, "cbree_diffusion_problem_aggregated.json")
# if not (path.exists(path_df) and path.exists(path_df_agg)):
#     df = ree.load_data(data_dir, "*vmfnm*")
#     df.drop(columns=["index", "Unnamed: 0",  "VAR Weighted Average Estimate","CVAR", "callback"], inpla
#     df.drop_duplicates(inplace=True)
#     df.reset_index(drop=True, inplace=True)
#     # Round parameters to compare floats safely
#     for col in [c for c in df.columns if c in DF_COLUMNS_TO_LATEX.keys()]:
#         if isinstance(df[col].values[0], float):
#             df[col] = df[col].round(5)
#     # melt aggregated estimates
#     df = df.rename(columns={"Estimate": "Last Estimate"})\
#         .melt(id_vars = [c for c in df.columns if not "Estimate" in c],
#               var_name="Averaging Method",
#               value_name="Estimate")
#     df = df.apply(expand_cbree_name, axis=1, columns = ["Averaging Method", "observation_window"])
#     # pretty names
#     df = df.rename(columns=DF_COLUMNS_TO_LATEX)
#     #process data: add evaluations etc
#     df = ree.add_evaluations(df)
#     df_agg = ree.aggregate_df(df)
#     #save
#     df.to_json(path_df)
#     df_agg.to_json(path_df_agg)
# else:
#     df = pd.read_json(path_df)
#     df_agg = pd.read_json(path_df_agg)
# load benchmarks
# bm_data_dirs = {
#     "enkf":"docs/benchmarking/data/enkf_sim_diffusion",
#     "sis": "docs/benchmarking/data/sis_sim_diffusion"
# }
# bm_df_names ={"df": "benchmark_diffusion_problems_processed.json",
#               "df_agg": "benchmark_diffusion_problems_aggregated.json"}
# df_bm, df_bm_agg = ree.get_benchmark_df(data_dirs=bm_data_dirs,
#                                         df_names=bm_df_names,
#                                         df_dir="docs/benchmarking/data")
```

Make Figure

```
for prob in df_agg["Problem"].unique():
    #filter
    this_df = df_agg.query("Problem == @prob & `Smoothing Function` == 'algebraic'")
    this_df = this_df[this_df["$\\epsilon_{{\\text{{Target}}}}$"]==1]
    this_df = this_df[this_df['$N_{{ \\text{{obs}} }}$'].isin([4,8,12])]
    this_df = this_df.sort_values(["$\\Delta_{{\\text{{Target}}}}$", "$N_{{ \\text{{obs}} }}$"])
    this_df_bm = df_bm_agg.query("Problem == @prob & cvar_tgt == 1")
    #plot
    fig = px.line(
        this_df,
        x = "Relative Root MSE",
        y="Cost Mean",
        facet_col=r'$\epsilon_{{\text{{Target}}}}$',
        facet_row="$N_{{ \\text{{obs}} }}$",
        color="Averaging Method",
        log_x=True,
        log_y=True,
        markers=True,
        labels=LATEX_TO_HTML | {"cvar_tgt_str": LATEX_TO_HTML[DF_COLUMNS_TO_LATEX["cvar_tgt"]]})
    # add benchmark
    num_rows = len(this_df["$N_{{ \\text{{obs}} }}$"].unique())
    num_cols = len(this_df[r'$\epsilon_{{\text{{Target}}}}$'].unique())
    for bm_solver in this_df_bm.Solver.unique():
        dat =this_df_bm.query("Solver == @bm_solver")
        dat = dat.sort_values(["Solver", "Sample Size"])
        trace_dict = {
            "x" : dat["Relative Root MSE"],
            "y" : dat["Cost Mean"],
            "legendgrouptitle_text": "Benchmark Methods",
            "name": bm_solver,
            "legendgroup": "group",
            "mode": "markers+lines",
            "opacity": 0.8,
            "text":dat["Sample Size"],
            "hoverinfo":"text"
        }
        trace_dict = trace_dict  | BM_SOLVER_SCATTER_STYLE[bm_solver]
        fig = ree.add_scatter_to_subplots(fig, num_rows, num_cols, **trace_dict)
    #style
    fig.update_layout(**MY_LAYOUT)
    fig.update_layout(height=900)
    fig.for_each_annotation(
        lambda a: a.update(text =  "" if a.text.startswith(LATEX_TO_HTML[DF_COLUMNS_TO_LATEX["stepsize_t
    # save
    fig.write_image(f"diffusion problem.png".replace(" ", "_").lower(), scale=WRITE_SCALE)
    fig.show()
    # make and save caption
    fig_description = f"Solving the {prob} with the CBREE (vMFN) method using  \
different parameters. \
We vary the averaging method (color) and \
the divergence check $N_\\text{{obs}}$ (row). \
The choice of the stopping criterion $\\Delta_{{\\text{{Target}}}} = 2$, \
the stepsize tolerance $\\epsilon_{{\\text{{Target}}}}=1$ \
and indicator approximation {INDICATOR_APPROX_LATEX_NAME['algebraic']} \
are fixed. \
Furthermore we plot also the performance of the benchmark methods EnKF \
and SIS.  \
We used the sample sizes $J \\in {ree.vec_to_latex_set(df_agg['Sample Size'].unique())}$. \
Each marker represents the empirical estimates based the successful portion of $200$ simulations."
    display(Markdown(fig_description))
    with open(f"diffusion problem desc.tex".replace(" ", "_").lower(), "w") as file:
        file.write(fig_description)
```



Solving the Diffusion Problem (d=150) with the CBREE (vMFN) method using different parameters. We vary the averaging method (color) and the divergence check N_obs (row). The choice of the stopping criterion $\Delta_{\text{Target}} = 2$, the stepsize tolerance $\epsilon_{\text{Target}}=1$ and indicator approximation $I_\text{{alg}}$ are fixed. Furthermore we plot also the performance of the benchmark methods EnKF and SIS. We used the sample sizes $J \in {1000, 2000, \ldots, 6000}$. Each marker represents the empirical estimates based the successful portion of 200 simulations.

Compare Perfromance of Different Smoothing Functions

```
from os import path
import rareeventestimation as ree
import pandas as pd
import plotly.express as px
from rareeventestimation.evaluation.constants import INDICATOR_APPROX_LATEX_NAME, BM_SOLVER_SCATTER_STYLI
import plotly.graph_objects as go
from IPython.display import display, Markdown
# recommended: use autoreload for development: https://ipython.readthedocs.io/en/stable/config/extensions
%load_ext autoreload
%autoreload 2
```

Skip to main content

Load Data

Option 1: Get precomputed data online

```
# data is here: https://archive.org/details/konstantinalthaus-rareeventestimation-data
# you can got to this link and inspect the files before loading
df= pd.read_json("https://ia801504.us.archive.org/23/items/konstantinalthaus-rareeventestimation-data/in
```

Option 2: Aggregate locally precomputed data

```
## uncomment to load existing data
## or to compile data after computing it yourself:
# data_dir = "docs/benchmarking/data/cbree_sim/indicator_functions_performance"
# df = ree.load_data(data_dir, "*")
# df.drop(columns=["index", "Unnamed: 0"], inplace=True)
# df.drop_duplicates(inplace=True)
# df = df.query("observation_window==0") \
#     .reset_index()
# df = ree.add_evaluations(df, only_success=True)
# df.to_json(path.join(data_dir, "indicator_function_performance.json"))
```

```
for tgt in df.cvar_tgt.unique():
    # Count proportion of unsuccessful exit status
    df_success=df.query("cvar_tgt==@tgt")\
        .groupby(["tgt_fun", "Problem"])["Message"].apply(pd.value_counts)
    df_success = pd.DataFrame(df_success)
    df_success = df_success[df_success.index.get_level_values(2)!="Success"]
    df_success["Message"] = df_success["Message"]/200
    df_success.reset_index(inplace=True)
    # Compute order of tgt_funs form best to worst
    lvl_1_order = df[["cvar_tgt", "tgt_fun", "Success Rate"]].query("cvar_tgt==@tgt") \
        .groupby("tgt_fun") \
        .mean() \
        .loc[:,"Success Rate"] \
        .sort_values(ascending=False) \
        .index \
        .values
    lvl_1 = [idx for idx in lvl_1_order if idx in df_success["tgt_fun"].values]
    # arange
    tbl = pd.pivot_table(df_success,
                        values="Message",
                        columns=["level_2"],
                        index=["tgt_fun", "Problem"],
                        fill_value="0 \%",
                        aggfunc= lambda x: f"{100*x.values.item():.1f}\\%")
    tbl = tbl.reindex(lvl_1, level=0)
    tbl.index  = tbl.index.set_levels([*map(ree.squeeze_problem_names, tbl.index.levels[1])], level=1) #
    # style and save
    tbl.columns.name=None
    tbl.index = tbl.index.set_names(names={"tgt_fun": "Approximation"}, )
    tbl = tbl.rename(columns={
            "Not Converged.":"Not Converged",
            "attempt to get argmax of an empty sequence": "No finite weights $\\bm{{w}}^n$",
            "singular matrix":"Singular $c^n$"}, index=INDICATOR_APPROX_LATEX_NAME)
    tbl.style.to_latex(f"success_rates_tgt_{tgt}.tex",
                       multirow_align="naive",
                       #column_format="ccrRP",
                       clines="skip-last;data")
    display(tbl)# no latex display: https://github.com/mathjax/mathjax-docs/wiki/LaTeX-Tabular-environme
    # write caption
    tbl_desc = f"Exit Messages of unsuccessful runs with stopping criterion $\\Delta_{{\\text{{Target}}}
    display(Markdown(tbl_desc))
    with open(f"success_rates_tgt_{tgt}_desc.tex", "w") as file:
        file.write(tbl_desc)
```

Approximation	Problem	Not Converged	No finite weights $\bm{{w}}^n$	Singular c^n
$I_\text{{arctan}}$	FRP (d=2)	100.0\%	0 \%	0 \%
	FRP (d=50)	100.0\%	0 \%	0 \%
	LP (d=50)	100.0\%	0 \%	0 \%
$I_\text{{alg}}$	CP	2.0\%	0 \%	0 \%
	FRP (d=2)	99.0\%	0 \%	0 \%
	FRP (d=50)	100.0\%	0 \%	0 \%
	LP (d=50)	100.0\%	0 \%	0 \%
$I_\text{{sig}}$	CP	4.0\%	0 \%	0 \%
	FRP (d=2)	100.0\%	0 \%	0 \%
	FRP (d=50)	100.0\%	0 \%	0 \%
	LP (d=50)	100.0\%	0 \%	0 \%
$I_\text{{tanh}}$	CP	28.0\%	0 \%	0 \%
	FRP (d=2)	91.0\%	0.5\%	0 \%
	FRP (d=50)	100.0\%	0 \%	0 \%
	LP (d=2)	2.5\%	7.0\%	0 \%
	LP (d=50)	47.0\%	0 \%	39.0\%
$I_\text{{erf}}$	CP	100.0\%	0 \%	0 \%
	FRP (d=2)	100.0\%	0 \%	0 \%
	FRP (d=50)	100.0\%	0 \%	0 \%
	LP (d=2)	40.5\%	0 \%	0 \%
	LP (d=50)	86.0\%	0 \%	14.0\%
$I_\text{{ReLU}}$	CP	100.0\%	0 \%	0 \%
	FRP (d=2)	100.0\%	0 \%	0 \%
	FRP (d=50)	100.0\%	0 \%	0 \%
	LP (d=2)	99.5\%	0 \%	0 \%
	LP (d=50)	100.0\%	0 \%	0 \%

Exit Messages of unsuccessful runs with stopping criterion $\Delta_{\text{Target}} = 1$. Values are proportional to 200 sample runs.

Approximation	Problem	Not Converged	No finite weights $\bm{{w}}^n$	Singular c^n
$I_\text{{erf}}$	LP (d=50)	0 \%	0 \%	14.0\%
$I_\text{{tanh}}$	FRP (d=2)	0 \%	0.5\%	0 \%
	LP (d=2)	0 \%	7.0\%	0 \%
	LP (d=50)	0 \%	0 \%	39.0\%
$I_\text{{ReLU}}$	FRP (d=2)	81.5\%	0 \%	0 \%

Exit Messages of unsuccessful runs with stopping criterion $\Delta_{\text{Target}} = 10$. Values are proportional to 200 sample runs.

```
for cvar_tgt in df.cvar_tgt.unique():
    df_agg = ree.aggregate_df(df.query(f"cvar_tgt==@cvar_tgt"))
    for op in ["==", "<"]:
        tgt_fun_list = df_agg[["tgt_fun", "Success Rate"]].groupby("tgt_fun") \
            .mean() \
            .reset_index() \
            .query(f"`Success Rate` {op} 1.0")["tgt_fun"].unique()
        if len(tgt_fun_list) > 0:
            # arrange functions
            df_acc = pd.pivot_table(df_agg.query("tgt_fun in @tgt_fun_list"),
                                    values="Relative Root MSE",
                                    columns=["tgt_fun"],
                                    index=["Problem"])
            # order functions
            df_acc = df_acc.reindex(df_acc.mean().sort_values().index, axis=1)
            # style and save
            df_acc = df_acc.rename(columns=INDICATOR_APPROX_LATEX_NAME)
            df_acc.columns.name="Approximation"
            tbl = df_acc.rename(index={p: ree.squeeze_problem_names(p) for p in df_acc.index})\
                .style.format(precision=2)
            tbl.to_latex(f"accuracy_tgt_{cvar_tgt}{'_success_only' if op=='==' else '' }.tex",
                         clines="all;data")
            display(tbl) # no latex display: https://github.com/mathjax/mathjax-docs/wiki/LaTeX-Tabular-(
            # write caption
            tbl_desc = f"Relative root mean squared error of {'successful runs with indicator function ap
            display(Markdown(tbl_desc))
            with open(f"accuracy_tgt_{cvar_tgt}{'_success_only' if op=='==' else '' }_desc.tex", "w") as
                file.write(tbl_desc)
```

Approximation	$I_\text{{arctan}}$	$I_\text{{sig}}$	$I_\text{{alg}}$	$I_\text{{tanh}}$	$I_\text{{erf}}$	$I_\text{{Re
Problem						
CP	0.02	0.02	0.02	0.03	nan	
FRP (d=2)	nan	nan	0.04	nan	nan	
LP (d=2)	0.02	0.02	0.02	0.05	0.14	

Relative root mean squared error of successful runs with indicator function approximations that have not always converged using the stopping criterion $\Delta_{\text{Target}} = 1$.

Approximation	$I_\text{{arctan}}$	$I_\text{{alg}}$	$I_\text{{sig}}$
Problem			
CP	0.12	0.14	0.11
FRP (d=2)	0.16	0.14	0.26
FRP (d=50)	0.38	0.64	0.73
LP (d=2)	0.14	0.15	0.07
LP (d=50)	0.42	0.50	0.64

Relative root mean squared error of successful runs with indicator function approximations that always led to convergence using the stopping criterion $\Delta_{\text{Target}} = 10$.

Approximation	$I_\text{{tanh}}$	$I_\text{{erf}}$	$I_\text{{ReLU}}$
Problem			
CP	0.10	0.15	0.54
FRP (d=2)	0.48	0.94	0.22
FRP (d=50)	0.93	0.95	0.94
LP (d=2)	0.11	0.14	0.59
LP (d=50)	0.69	0.80	0.80

Relative root mean squared error of successful runs with indicator function approximations that have not always converged using the stopping criterion $\Delta_{\text{Target}} = 10$.