Supplementary Material Paper

Contents

The purpose of this folder is to make the experiments presented in the paper **K. Althaus I. Papaioannou, and E. Ullmann, Consensus-based rare event estimation, (2023), https://doi.org/10.48550/arXiv.2304.09077** reproducible.

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

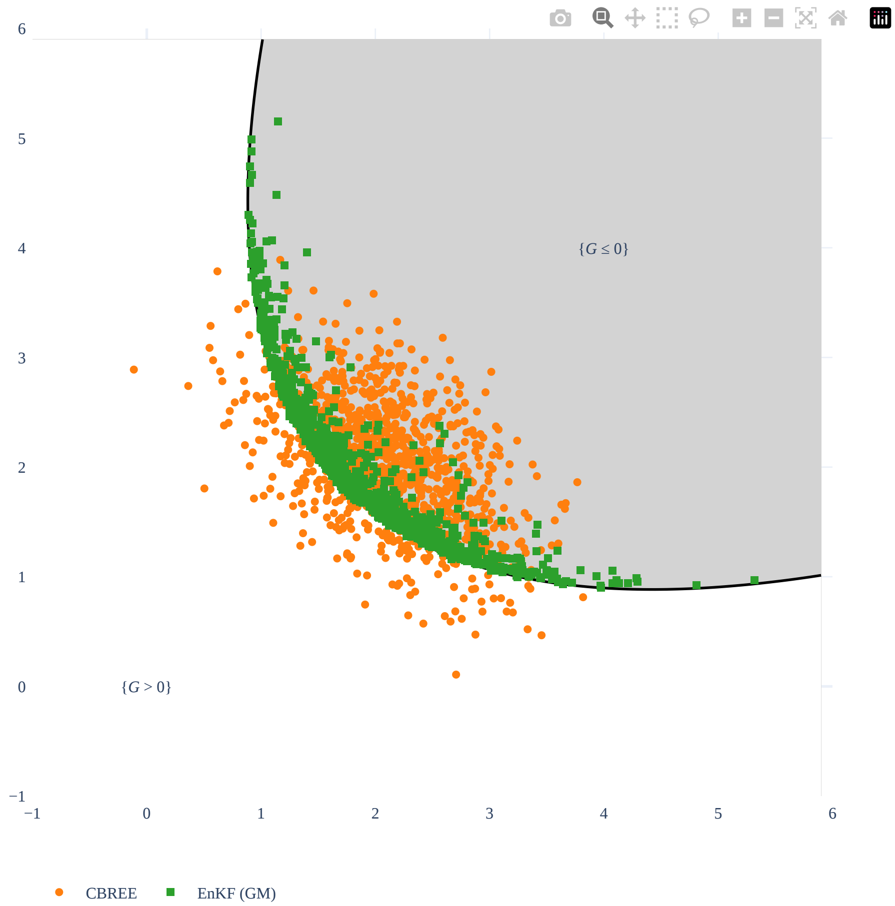
Plot failure domain for toy problems

```
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/extension:
%load_ext autoreload
%autoreload 2
```

Skip to main content

```
# problem and solver stuff
import plotly.io as pio
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", "cross-open"]
#figure stuff
safe_annotation_anchor = [0, 0]
failure_annotation_anchor = [4 , 4]
delta = 0.1
x0 = -1
x1 = 6
xx = np.arange(x0,x1, delta)
yy = np.arange(x0,x1, delta)
col_scale = [[0, "lightgrey"], [1, "white"]]
contour_style = {"start": 0, "end": 0, "size": 0, "showlabels": False}
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,
            marker_symbol = marker_shape_list[j],
        )
        fig.add_trace(sc)
    # style
    fig.update_layout(**MY_LAYOUT)
    fig.update_layout(height=WIDTH,
                     width=WIDTH,
                     legend_orientation="h",
                     xaxis_range = [x0, x1],
                     yaxis_range = [x0, x1])
    fig.add_annotation(x=failure_annotation_anchor[0],
                      y=failure_annotation_anchor[1],
                      ay=0,
                      ax=0,
                      text="{<i>G</i> \u2264 0}")
    fig.add_annotation(x=safe_annotation_anchor[0],
                      y=safe_annotation_anchor[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 +".pdf")
    fig.show()
    # make and save caption
    fig_description = f"Failure domain of the {prob.name}. \
The final ensembles of the EnKF and CBREE methods applied to the convex limit state function $G(x)= \\fra
("" 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 co
    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. The final ensembles of the EnKF and CBREE methods applied to the convex limit state function $G(x)= \frac{(x_1-x_2)^2}{10} - \frac{x_1 +x_2}{\sqrt{2}} + \frac{2}{5}$. 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 $\text{Lip}(\varsigma) = 1$.

Evaluate Performance for Flowrate 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
```

Skip to main content

Load Data

Option 1: Get precomputed data online

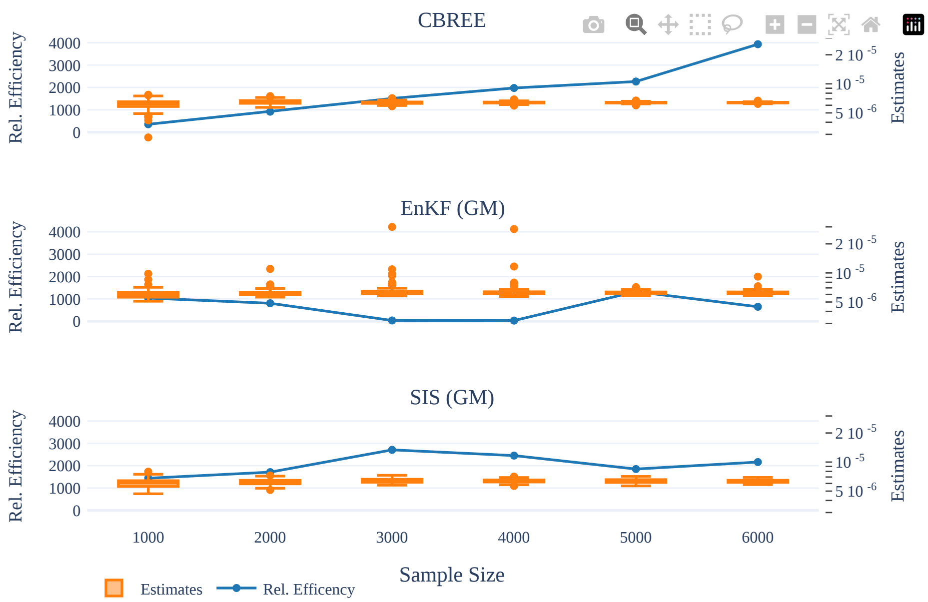
```
# data is here: https://archive.org/details/konstantinalthaus-rareeventestimation-data
# you can got to this link and inspect the files pefore loading
df_bm_agg = pd.read_json("https://ia801504.us.archive.org/23/items/konstantinalthaus-rareeventestimation
df_bm = pd.read_json("https://ia801504.us.archive.org/23/items/konstantinalthaus-rareeventestimation-data
df_agg = pd.read_json("https://ia801504.us.archive.org/23/items/konstantinalthaus-rareeventestimation-da
df = pd.read_json("https://ia601504.us.archive.org/23/items/konstantinalthaus-rareeventestimation-data/cl
```

Option 2: Aggregate locally precomputed data

```
# # uncomment to load existing data
# # or to compile data after computing it yourself:

# data_dir ="/Users/konstantinalthaus/Documents/Master-TUM/Masterthesis/data/cbree_sim/nonlinear_oscilla

# path_df = path.join(data_dir, "cbree_oscillator_problem_processed.json")
# path_df_agg = path.join(data_dir, "cbree_oscilltor_problem_aggregated.json")
# if not (path.exists(path_df) and path.exists(path_df_agg)):
#     df = ree.load_data(data_dir, "*")
#     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(ree.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, double_precision = DOUBLE_PRECISION)
#     df_agg.to_json(path_df_agg, double_precision=DOUBLE_PRECISION)
# else:
#     df = pd.read_json(path_df)
#     df_agg = pd.read_json(path_df_agg)
# # load benchmarks
# data_dirs_bm = {
#     "enkf": "/Users/konstantinalthaus/Documents/Master-TUM/Masterthesis/data/enkf_sim_oscillator",
#     "sis": "/Users/konstantinalthaus/Documents/Master-TUM/Masterthesis/data/sis_sim_oscillator"
# }
# df_names_bm = {
#     "df": "oscillator_problem_benchmark_processed.json",
#     "df_agg": "oscillator_problem_benchmark_aggregate.json"
# }
# df_bm, df_bm_agg = ree.get_benchmark_df(data_dirs=data_dirs_bm,
#                                          df_names=df_names_bm,
#                                          df_dir="/Users/konstantinalthaus/Documents/Master-TUM/Masterthe
#                                          force_reload=True,
#                                          remove_outliers=False)
```

Compare relative Efficiency

```
# filter
my_mixture_model = "GM"
my_obs_windows = 2
my_epsilon = 1
my_bm_cvar_tgt = 1
prob = df_agg.Problem.unique().item()
this_df_agg = df_agg.query(
    "Problem == @prob & `Averaging Method`=='Average Estimate' & mixture_model==@my_mixture_model")
this_df_agg = this_df_agg[this_df_agg['$N_{{ \\text{{obs}} }}$']
                          == my_obs_windows]
this_df_agg = this_df_agg[this_df_agg['$\\epsilon_{{\\text{{Target}}}}$'] == my_epsilon]
this_df_agg = this_df_agg[this_df_agg['$\\Delta_{{\\text{{Target}}}}$'] == my_bm_cvar_tgt]
this_df = df.query("Problem == @prob & `Averaging Method`=='Average Estimate' & mixture_model==@my_mixtu
this_df = this_df[this_df['$N_{{ \\text{{obs}} }}$']
                          == my_obs_windows]
this_df = this_df[this_df['$\\epsilon_{{\\text{{Target}}}}$'] == my_epsilon]
this_df = this_df[this_df['$\\Delta_{{\\text{{Target}}}}$'] == my_bm_cvar_tgt]
this_df_agg = pd.concat(
    [this_df_agg, df_bm_agg[df_bm_agg.Solver.str.contains("GM")]])
this_df = pd.concat(
    [this_df, df_bm[df_bm.Solver.str.contains("GM")]])
this_df_agg["Relative Efficiency"] = (
    1 - this_df_agg["Truth"]) * this_df_agg["Truth"]
this_df_agg["Relative Efficiency"] = this_df_agg["Relative Efficiency"] / \
    (this_df_agg["Cost Mean"] * this_df_agg["MSE"])

this_df_agg.loc[this_df_agg.Solver.str.startswith("CBREE"), "Solver"] = "CBREE"
this_df.loc[this_df.Solver.str.startswith("CBREE"), "Solver"] = "CBREE"
fig = ree.make_efficiency_plot(this_df_agg,
                               this_df,
                               "Sample Size",
                           "Relative Efficiency",
                           "Estimate",
                           facet_row="Solver",
                           shared_secondary_y_axes=True)
fig.show()
fig_title = "eficiency plot " + prob + " solver vs sample size"
fig.write_image(f"{fig_title}.pdf".replace(" ", "_").lower())
fig_description = f"Solving the {prob} with the CBREE method and two benchmark methods (row). \
We vary the sample size ($x$-axis) and show for each sample size two quantities. \
There is an estimate of the relative efficiency (left $y$-axis) and \
a boxplot of the corresponding ${int(2*this_df_agg.Seed.unique()[0]+1)}$ empirical estimates of the failu
The other parameters of the CBREE method are $\\Delta_{{\\text{{Target}}}} = {my_bm_cvar_tgt}$, \
$N_\\text{{obs}} = {my_obs_windows}$ and \
$\\epsilon_{{\\text{{Target}}}} = {my_epsilon}$."
with open(f"{fig_title} desc.tex".replace(" ", "_").lower(), "w") as file:
    file.write(fig_description)
display(Markdown(fig_description))
```



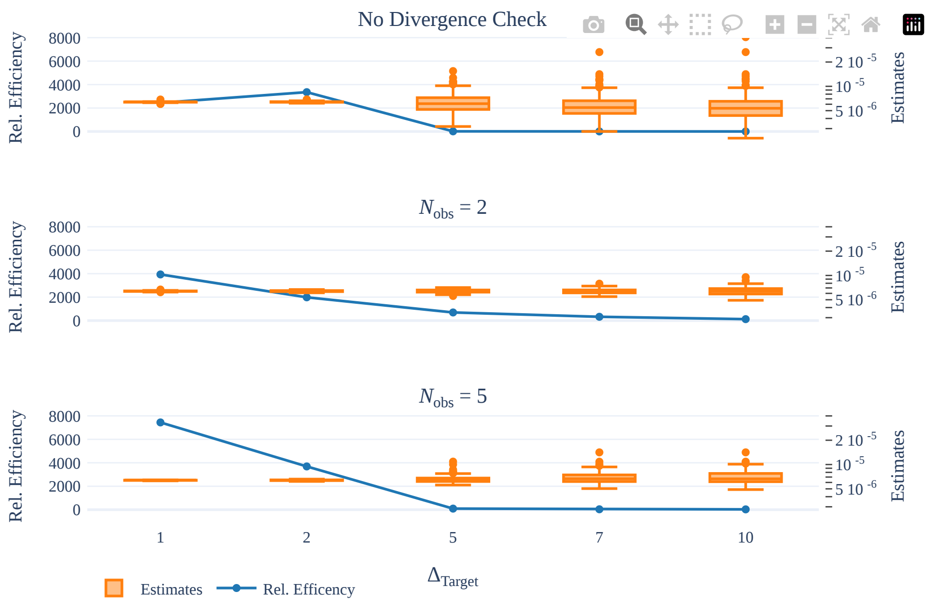
Solving the Nonlinear Oscillator Problem with the CBREE method and two benchmark methods (row). We vary the sample size (x-axis) and show for each sample size two quantities. There is an estimate of the relative efficiency (left y-axis) and a boxplot of the corresponding 100 empirical estimates of the failure probability (righ y-axis). The other parameters of the CBREE method are $\Delta_{\text{Target}} = 1$, $N_\text{obs} = 2$ and $\epsilon_{\text{Target}} = 1$.

Parameter impact on rel. Efficiency

The observation window

```
# filter
my_mixture_model = "GM"
my_obs_windows = [0,2,5]
my_epsilon = 1
my_bm_cvar_tgt = [1,2,5]
my_sample_size=6000
this_df_agg = df_agg.query("Problem == @prob & `Averaging Method`=='Average Estimate' & mixture_model==@m
this_df_agg = this_df_agg[this_df_agg['$N_{{ \\text{{obs}} }}$'].isin(my_obs_windows)]
this_df_agg = this_df_agg[this_df_agg['$\\epsilon_{{\\text{{Target}}}}$'] == my_epsilon]
this_df = df.query("Problem == @prob & `Averaging Method`=='Average Estimate' & mixture_model==@my_mixtu
this_df = this_df[this_df['$N_{{ \\text{{obs}} }}$'].isin(my_obs_windows)]
this_df = this_df[this_df['$\\epsilon_{{\\text{{Target}}}}$'] == my_epsilon]
this_df_agg.loc[:,"Relative Efficiency"] = (
    1 - this_df_agg["Truth"]) * this_df_agg["Truth"]
this_df_agg.loc[:,"Relative Efficiency"] = this_df_agg["Relative Efficiency"] / \
    (this_df_agg["Cost Mean"] * this_df_agg["MSE"])

# nice columns names
columns = ['$\\Delta_{{\\text{{Target}}}}$']
# new_names = {k:LATEX_TO_HTML[v] for v in columns}
for dat in [this_df, this_df_agg]:
    for c in columns:
        dat.loc[:,c] = dat[c].astype("int").astype("string")

fig = ree.make_efficiency_plot(this_df_agg,
                               this_df,
'$\\Delta_{{\\text{{Target}}}}$',
                           "Relative Efficiency",
                           "Estimate",
                           facet_row='$N_{{ \\text{{obs}} }}$',
                           facet_row_prefix = LATEX_TO_HTML['$N_{{ \\text{{obs}} }}$'],
                           shared_secondary_y_axes=True,
                           facet_name_sep = ' = ',
                           x_axis_sorting_key=lambda x: x.astype("int"),
                           labels = LATEX_TO_HTML)
# 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))
fig.show()
fig_title = "eficiency plot " + prob + " delta_tgt vs obs_window"
fig.write_image(f"{fig_title}.pdf".replace(" ", "_").lower())
fig_description = f"Solving the {prob} with the CBREE method with ${my_sample_size}$ samples. \
We vary the parameter $\\Delta_{{\\text{{Target}}}}$ ($x$-axis) and \
the length of the observation window $N_\\text{{obs}}$ (row). \
For each parameter choice we plot an estimate of the relative efficiency (left $y$-axis) and \
a boxplot of the corresponding ${int(2*this_df_agg.Seed.unique()[0]+1)}$ empirical estimates of the failu
The parameter $\\epsilon_{{\\text{{Target}}}} = {my_epsilon}$ is fixed."
with open(f"{fig_title} desc.tex".replace(" ", "_").lower(), "w") as file:
    file.write(fig_description)
display(Markdown(fig_description))
```

Skip to main content



Solving the Nonlinear Oscillator Problem with the CBREE method with 6000 samples. We vary the parameter Δ_{Target} (x-axis) and the length of the observation window N_obs (row). For each parameter choice we plot an estimate of the relative efficiency (left y-axis) and a boxplot of the corresponding 100 empirical estimates of the failure probability (righ y-axis). The parameter $\epsilon_{\text{Target}} = 1$ is fixed.

Stepsize Control

```
# data is here: https://archive.org/details/konstantinalthaus-rareeventestimation-data
# you can got to this link and inspect the files pefore loading
df_stepsize = pd.read_json("https://ia601504.us.archive.org/23/items/konstantinalthaus-rareeventestimati
df_stepsize_agg = pd.read_json("https://ia601504.us.archive.org/23/items/konstantinalthaus-rareeventesti
```

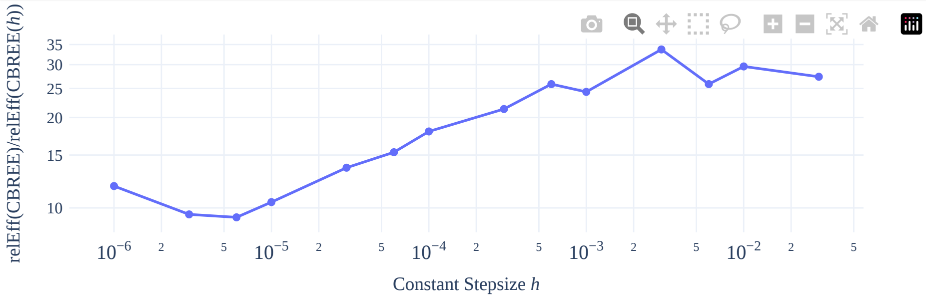
```
# # uncomment to load existing data
# # or to compile data after computing it yourself:

# data_dir ="/Users/konstantinalthaus/Documents/Master-TUM/Masterthesis/rareeventestimation/docs/benchma

# path_df_stepsize = path.join(data_dir, "cbree_fixed_stepsize_processed.json")
# path_df_stepsize_agg = path.join(data_dir, "cbree_fixed_stepsize_aggregated.json")
# if not (path.exists(path_df_stepsize) and path.exists(path_df_stepsize_agg)):
#     df_stepsize = ree.load_data(data_dir, "*")
#     df_stepsize.drop(columns=["index", "Unnamed: 0",  "VAR Weighted Average Estimate","CVAR", "callbac
#             inplace=True,
#             errors='ignore')
#     df_stepsize.drop_duplicates(inplace=True)
#     df_stepsize.reset_index(drop=True, inplace=True)
#     # Round parameters to compare floats safely
#     for col in [c for c in df_stepsize.columns if c in DF_COLUMNS_TO_LATEX.keys()]:
#         if isinstance(df_stepsize[col].values[0], float):
#             df_stepsize[col] = df_stepsize[col].round(5)
#     # melt aggregated estimates
#     df_stepsize = df_stepsize.rename(columns={"Estimate": "Last Estimate"})\
#         .melt(id_vars = [c for c in df_stepsize.columns if not "Estimate" in c],
#              var_name="Averaging Method",
#              value_name="Estimate")
#     df_stepsize = df_stepsize.apply(ree.expand_cbree_name, axis=1, columns = ["Averaging Method", "obs
#     # pretty names
#     df_stepsize = df_stepsize.rename(columns=DF_COLUMNS_TO_LATEX)
#     #process data: add evaluations etc
#     df_stepsize = ree.add_evaluations(df_stepsize)
#     df_stepsize_agg = ree.aggregate_df(df_stepsize)
#     #save
#     df_stepsize.to_json(path_df_stepsize, double_precision = DOUBLE_PRECISION)
#     df_stepsize_agg.to_json(path_df_stepsize_agg, double_precision=DOUBLE_PRECISION)
# else:
#     df_stepsize = pd.read_json(path_df_stepsize)
```

[Skip to main content](#)

Comapre Efficiency of Adaptive and Constant Stepsize Method

```
# filter adaptive stepsize df
my_mixture_model = "GM"
my_obs_windows = 2
my_epsilon = 1
my_bm_cvar_tgt = 1
prob = df_agg.Problem.unique().item()
this_df_agg = df_agg.query(
    "Problem == @prob & `Averaging Method`=='Average Estimate' & mixture_model==@my_mixture_model")
this_df_agg = this_df_agg[this_df_agg['$N_{{ \\text{{obs}} }}$']
                          == my_obs_windows]
this_df_agg = this_df_agg[this_df_agg['$\\epsilon_{{\\text{{Target}}}}$'] == my_epsilon]
this_df_agg = this_df_agg[this_df_agg['$\\Delta_{{\\text{{Target}}}}$'] == my_bm_cvar_tgt]
this_df = df.query("Problem == @prob & `Averaging Method`=='Average Estimate' & mixture_model==@my_mixtu
this_df = this_df[this_df['$N_{{ \\text{{obs}} }}$']
                          == my_obs_windows]
this_df = this_df[this_df['$\\epsilon_{{\\text{{Target}}}}$'] == my_epsilon]
this_df = this_df[this_df['$\\Delta_{{\\text{{Target}}}}$'] == my_bm_cvar_tgt]
this_df["t_step"] = 'adaptive'
this_df_agg["t_step"] = 'adaptive'
# filter constant stepsize df
this_df_stepsize_agg = df_stepsize_agg.query("`Averaging Method`=='Average Estimate'")
this_df_stepsize= df_stepsize.query("`Averaging Method`=='Average Estimate'")
# concat dfs, compute efficiency
this_df_agg = pd.concat(
    [this_df_agg, this_df_stepsize_agg])
this_df = pd.concat(
    [this_df, this_df_stepsize])
this_df_agg["Relative Efficiency"] = (
    1 - this_df_agg["Truth"]) * this_df_agg["Truth"]
this_df_agg["Relative Efficiency"] = this_df_agg["Relative Efficiency"] / \
    (this_df_agg["Cost Mean"] * this_df_agg["MSE"])
# plot
this_df_agg.loc[this_df_agg.Solver.str.startswith("CBREE"), "Solver"] = "CBREE"
this_df.loc[this_df.Solver.str.startswith("CBREE"), "Solver"] = "CBREE"
df_eff = this_df_agg.query("`Sample Size` == 6000")
efficiency_adaptive_method =df_eff.loc[df_eff.t_step == 'adaptive', 'Relative Efficiency'].values.item()
df_eff['Relative Efficiency'] = efficiency_adaptive_method / df_eff['Relative Efficiency']
df_eff = df_eff[df_eff.t_step != 'adaptive']
df_eff['t_step'] = df_eff['t_step'].apply(lambda x: float(x))
df_eff = df_eff.sort_values("t_step")
df_eff = df_eff.query("t_step <= 0.03")
labels = {'t_step': 'Constant Stepsize <i>h</i>',
          'Relative Efficiency': 'relEff(CBREE)/relEff(CBREE(<i>h</i>))'}
fig = px.line(df_eff,
        x='t_step',
        y='Relative Efficiency',
        log_x=True,
        log_y=True,
        markers=True,
        labels=labels)
fig.update_layout(**MY_LAYOUT)
fig.update_layout(height = 0.5 * MY_LAYOUT['height'])

fig.show()
fig_title = "eficiency plot " + prob + "adaptive vs constant stepsize"
fig.write_image(f"{fig_title}.pdf".replace(" ", "_").lower())
fig_description = f"Results of comparing the efficiency of the CBREE method with constant stepsize, CBREE
We fix the sample size to 6000 and compute the relative efficiency of the original CBREE method as well a
Here we plot the ratio of the relative Efficiency of the CBREE method and of the CBREE($h$) method ($y$-a
Each estimate of the relative efficiency is based on 100 samples. \
The other parameters of the CBREE method are $\\Delta_{{\\text{{Target}}}} = {my_bm_cvar_tgt}$, \
$N_\\text{{obs}} = {my_obs_windows}$ and \
$\\epsilon_{{\\text{{Target}}}} = {my_epsilon}$."
with open(f"{fig_title} desc.tex".replace(" ", "_").lower(), "w") as file:
    file.write(fig_description)
display(Markdown(fig_description))
```

```
/tmp/ipykernel_2970/5112171.py:37: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.ht
```



Results of comparing the efficiency of the CBREE method with constant stepsize, CBREE(h), and CBREE method with adaptive stepsize, CBREE, for the Nonlinear Oscillator Problem. We fix the sample size to 6000 and compute the relative efficiency of the original CBREE method as well as the CBREE(h) method with a fixed stepsize h. Here we plot the ratio of the relative Efficiency of the CBREE method and of the CBREE(h) method (y-axis) for several choices of h (x-axis). Each estimate of the relative efficiency is based on 100 samples. The other parameters of the CBREE method are $\Delta_{\text{Target}} = 1$, $N_\text{obs} = 2$ and $\epsilon_{\text{Target}} = 1$.

Evaluate Performance for Flowrate 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 pefore loading
df_bm_agg = pd.read_json("https://ia801504.us.archive.org/23/items/konstantinalthaus-rareeventestimation
df_bm = pd.read_json("https://ia801504.us.archive.org/23/items/konstantinalthaus-rareeventestimation-data
df_agg = pd.read_json("https://ia801504.us.archive.org/23/items/konstantinalthaus-rareeventestimation-dat
df = pd.read_json("https://ia601504.us.archive.org/23/items/konstantinalthaus-rareeventestimation-data/cl
```

Option 2: Aggregate locally precomputed data

```
# # uncomment to load existing data
# # or to compile data after computing it yourself:

# data_dir ="/Users/konstantinalthaus/Documents/Master-TUM/Masterthesis/data/cbree_sim/cbree_sim_flowrat

# path_df = path.join(data_dir, "cbree_flowrate_problem_processed.json")
# path_df_agg = path.join(data_dir, "cbree_flowrate_problem_aggregated.json")
# if not (path.exists(path_df) and path.exists(path_df_agg)):
#     df = ree.load_data(data_dir, "*")
#     df.drop(columns=["index", "Unnamed: 0",  "VAR Weighted Average Estimate","CVAR", "callback"], inpl
#     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(ree.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, double_precision = DOUBLE_PRECISION)
#     df_agg.to_json(path_df_agg, double_precision=DOUBLE_PRECISION)
# else:
#     df = pd.read_json(path_df)
#     df_agg = pd.read_json(path_df_agg)
# # load benchmarks
# data_dirs_bm = {
#     "enkf": "/Users/konstantinalthaus/Documents/Master-TUM/Masterthesis/data/enkf_flow_rate",
#     "sis": "/Users/konstantinalthaus/Documents/Master-TUM/Masterthesis/data/sis_flow_rate"
# }
# df_names_bm = {
#     "df": "flow_rate_benchmark_processed.json",
#     "df_agg": "flow_rate_benchmark_aggregate.json"
# }
# df_bm, df_bm_agg = ree.get_benchmark_df(data_dirs=data_dirs_bm,
#                                          df_names=df_names_bm,
#                                          df_dir="/Users/konstantinalthaus/Documents/Master-TUM/Masterth
#                                          force_reload=True,
#                                          remove_outliers=False)
```

Compare relative Efficiency

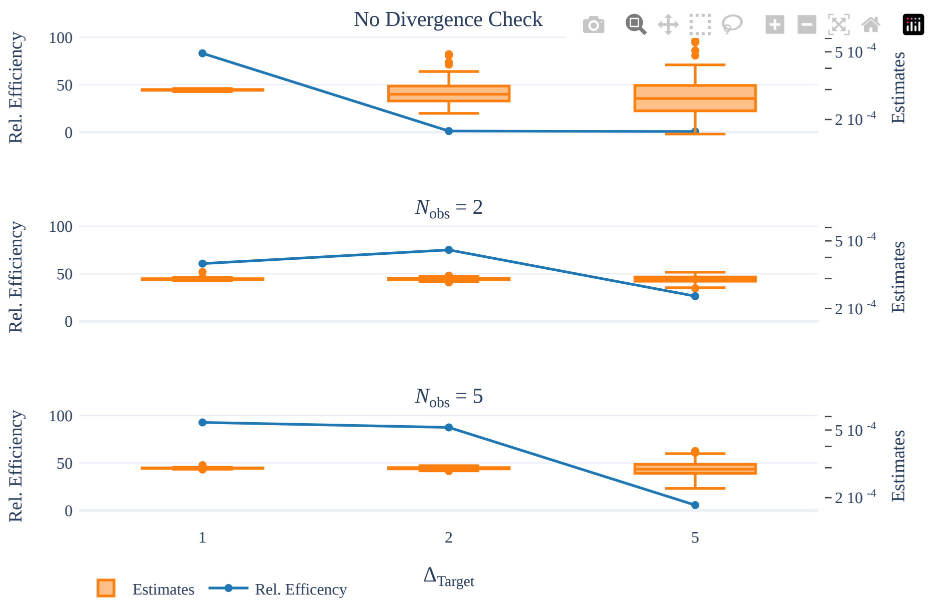
```
# filter
my_mixture_model = "GM"
my_obs_windows = 2
my_epsilon = 1
my_bm_cvar_tgt = 1
prob = df_agg.Problem.unique().item()
this_df_agg = df_agg.query(
    "Problem == @prob & `Averaging Method`=='Average Estimate' & mixture_model==@my_mixture_model")
this_df_agg = this_df_agg[this_df_agg['$N_{{ \\text{{obs}} }}$']
                          == my_obs_windows]
this_df_agg = this_df_agg[this_df_agg['$\\epsilon_{{\\text{{Target}}}}$'] == my_epsilon]
this_df_agg = this_df_agg[this_df_agg['$\\Delta_{{\\text{{Target}}}}$'] == my_bm_cvar_tgt]
this_df = df.query("Problem == @prob & `Averaging Method`=='Average Estimate' & mixture_model==@my_mixtu
this_df = this_df[this_df['$N_{{ \\text{{obs}} }}$']
                          == my_obs_windows]
this_df = this_df[this_df['$\\epsilon_{{\\text{{Target}}}}$'] == my_epsilon]
this_df = this_df[this_df['$\\Delta_{{\\text{{Target}}}}$'] == my_bm_cvar_tgt]
this_df_agg = pd.concat(
    [this_df_agg, df_bm_agg[df_bm_agg.Solver.str.contains("GM")]])
this_df = pd.concat(
    [this_df, df_bm[df_bm.Solver.str.contains("GM")]])
this_df_agg["Relative Efficiency"] = (
    1 - this_df_agg["Truth"]) * this_df_agg["Truth"]
this_df_agg["Relative Efficiency"] = this_df_agg["Relative Efficiency"] / \
    (this_df_agg["Cost Mean"] * this_df_agg["MSE"])

this_df_agg.loc[this_df_agg.Solver.str.startswith("CBREE"), "Solver"] = "CBREE"
this_df.loc[this_df.Solver.str.startswith("CBREE"), "Solver"] = "CBREE"
fig = ree.make_efficiency_plot(this_df_agg,
                               this_df,
                               "Sample Size",
                           "Relative Efficiency",
                           "Estimate",
                           facet_row="Solver",
                           shared_secondary_y_axes=True)
fig.show()
fig_title = "eficiency plot " + prob + " solver vs sample size"
fig.write_image(f"{fig_title}.pdf".replace(" ", "_").lower())
fig_description = f"Solving the {prob} with the CBREE method and two benchmark methods (row). \
We vary the sample size ($x$-axis) and show for each sample size two quantities. \
There is an estimate of the relative efficiency (left $y$-axis) and \
a boxplot of the corresponding ${int(2*this_df_agg.Seed.unique()[0]+1)}$ empirical estimates of the failu
The other parameters of the CBREE method are $\\Delta_{{\\text{{Target}}}} = {my_bm_cvar_tgt}$, \
$N_\\text{{obs}} = {my_obs_windows}$ and \
$\\epsilon_{{\\text{{Target}}}} = {my_epsilon}$."
with open(f"{fig_title} desc.tex".replace(" ", "_").lower(), "w") as file:
    file.write(fig_description)
display(Markdown(fig_description))
```

Skip to main content

Solving the Flow-rate Problem (d=10) with the CBREE method and two benchmark methods (row). We vary the sample size (x-axis) and show for each sample size two quantities. There is an estimate of the relative efficiency (left y-axis) and a boxplot of the corresponding 100 empirical estimates of the failure probability (righ y-axis). The other parameters of the CBREE method are $\Delta_{\text{Target}} = 1$, $N_\text{obs} = 2$ and $\epsilon_{\text{Target}} = 1$.

Parameter impact on rel. Efficiency

```
# filter
my_mixture_model = "GM"
my_obs_windows = [0,2,5]
my_epsilon = 1
my_bm_cvar_tgt = [1,2,5]
my_sample_size=6000
this_df_agg = df_agg.query("Problem == @prob & `Averaging Method`=='Average Estimate' & mixture_model==@m
this_df_agg = this_df_agg[this_df_agg['$N_{{ \\text{{obs}} }}$'].isin(my_obs_windows)]
this_df_agg = this_df_agg[this_df_agg['$\\epsilon_{{\\text{{Target}}}}$'] == my_epsilon]
this_df_agg = this_df_agg[this_df_agg['$\\Delta_{{\\text{{Target}}}}$'].isin(my_bm_cvar_tgt)]
this_df = df.query("Problem == @prob & `Averaging Method`=='Average Estimate' & mixture_model==@my_mixtu
this_df = this_df[this_df['$N_{{ \\text{{obs}} }}$'].isin(my_obs_windows)]
this_df = this_df[this_df['$\\epsilon_{{\\text{{Target}}}}$'] == my_epsilon]
this_df = this_df[this_df['$\\Delta_{{\\text{{Target}}}}$'].isin(my_bm_cvar_tgt)]
this_df_agg.loc[:,"Relative Efficiency"] = (
    1 - this_df_agg["Truth"]) * this_df_agg["Truth"]
this_df_agg.loc[:,"Relative Efficiency"] = this_df_agg["Relative Efficiency"] / \
    (this_df_agg["Cost Mean"] * this_df_agg["MSE"])

# nice columns names
columns = ['$\\Delta_{{\\text{{Target}}}}$']
# new_names = {k:LATEX_TO_HTML[v] for v in columns}
for dat in [this_df, this_df_agg]:
    for c in columns:
        dat.loc[:,c] = dat[c].astype("string")

fig = ree.make_efficiency_plot(this_df_agg,
                               this_df,
'$\\Delta_{{\\text{{Target}}}}$',
                           "Relative Efficiency",
                           "Estimate",
                           facet_row='$N_{{ \\text{{obs}} }}$',
                           facet_row_prefix = LATEX_TO_HTML['$N_{{ \\text{{obs}} }}$'],
                           shared_secondary_y_axes=True,
                           facet_name_sep = ' = ',
                           labels = LATEX_TO_HTML)
# 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))
fig.show()
fig_title = "eficiency plot " + prob + " delta_tgt vs obs_window"
fig.write_image(f"{fig_title}.pdf".replace(" ", "_").lower(),  engine="kaleido")
fig_description = f"Solving the {prob} with the CBREE method with ${my_sample_size}$ samples. \
We vary the parameter $\\Delta_{{\\text{{Target}}}}$ ($x$-axis) and \
the length of the observation window $N_\\text{{obs}}$ (row). \
For each parameter choice we plot an estimate of the relative efficiency (left $y$-axis) and \
a boxplot of the corresponding ${int(2*this_df_agg.Seed.unique()[0]+1)}$ empirical estimates of the failu
The parameter $\\epsilon_{{\\text{{Target}}}} = {my_epsilon}$ is fixed."
with open(f"{fig_title} desc.tex".replace(" ", "_").lower(), "w") as file:
    file.write(fig_description)
display(Markdown(fig_description))
```



Solving the Flow-rate Problem (d=10) with the CBREE method with 6000 samples. We vary the parameter Δ_{Target} (x-axis) and the length of the observation window N_obs (row). For each parameter choice we plot an estimate of the relative efficiency (left y-axis) and a boxplot of the corresponding 100 empirical estimates of the failure probability (righ y-axis). The parameter $\epsilon_{\text{Target}} = 1$ is fixed.

Highlight Loss of Performance in Higher Dimensions

```
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_STYLE
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

```
# Add new benchmark simulations to existing df
df_bm_agg = pd.read_json("https://archive.org/download/konstantinalthaus-rareeventestimation-data/benchma
df_agg =pd.read_json("https://ia801504.us.archive.org/23/items/konstantinalthaus-rareeventestimation-data
```

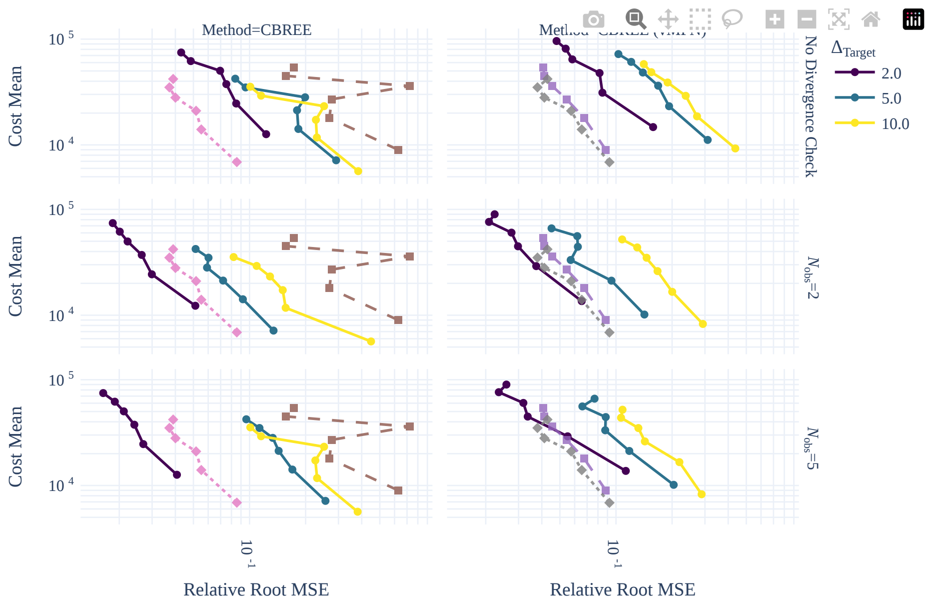
Skip to main content

Make Figures

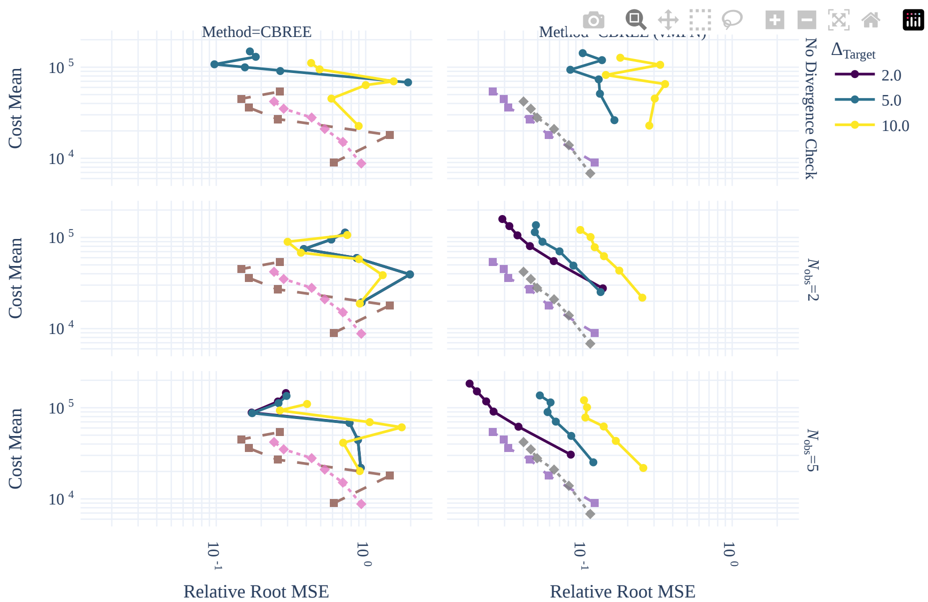
```
for prob in ["Linear Problem (d=2)", "Linear Problem (d=50)"]:
    # filter
    my_mixture_model = "CBREE"
    my_obs_windows = 2
    my_epsilon = 1
    my_bm_cvar_tgt = 1
    this_df = df_agg.query("Problem == @prob & `Averaging Method`=='Average Estimate'")
    this_df = this_df[this_df['$N_{{ \\text{{obs}} }}$'].isin([0,2,5])]
    this_df = this_df[this_df['$\\Delta_{{\\text{{Target}}}}$'].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")
    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
        }
        num_rows = len(this_df["$N_{{ \\text{{obs}} }}$"].unique())
        cols_idx = []
        for i, method in enumerate(this_df["Method"].unique()):
            if method == "CBREE" and "GM" in bm_solver:
                cols_idx.append(i)
            if "MFN" in method and ("MFN" in bm_solver):
                cols_idx.append(i)

        trace_dict = trace_dict | BM_SOLVER_SCATTER_STYLE[bm_solver]
        fig = ree.add_scatter_to_subplots(fig, num_rows, cols_idx, **trace_dict)
    # style
    fig.update_layout(**MY_LAYOUT)
    if  "yaxis_exponentformat" in MY_LAYOUT.keys():
        fig = ree.update_axes_format(fig,  MY_LAYOUT["xaxis_exponentformat"], MY_LAYOUT["yaxis_exponentfo
    # 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))

    # adjust column name position
    fig.for_each_annotation(
        lambda a: a.update(yshift =  -10 if a.text.startswith("Method") else 0))
    # show and save
    fig.show()
    fig_title = "convergence plot" + prob + "gm vs nongm"
    fig.write_image(f"{fig_title}.png".replace(" ", "_").lower(), scale=WRITE_SCALE)# make and save capti
    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 $\\epsilon_{{\\text{{Target}}}} = {0.5}$ is fixed. \
Furthermore we plot also the performance of the benchmark methods EnKF \
and SiS  \
                                                                         this_df.Seed.uniq
```

Skip to main content

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



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 $\epsilon_{\text{Target}} = 0.5$ is 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.



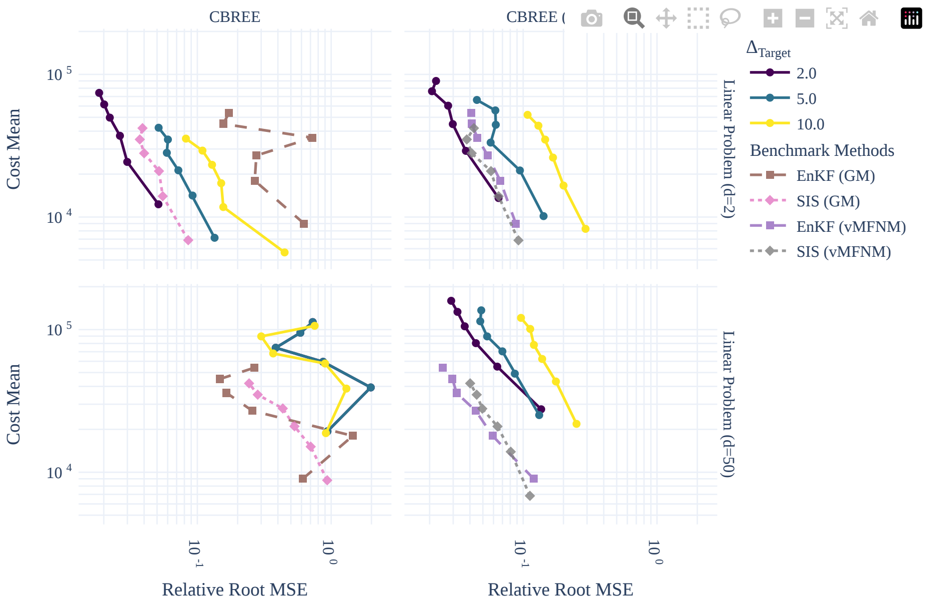
Solving the Linear Problem (d=50) with the CBREE methods using different parameters. We vary the stopping criterion Δ_{Target} (color), the divergence criterion N_obs (row) and the method (column). The parameter $\epsilon_{\text{Target}} = 0.5$ is fixed. Furthermore we plot also the performance of the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

Alternative Figure

```
# filter
my_mixture_model = "CBREE"
my_obs_windows = 2
my_epsilon = 1
my_bm_cvar_tgt = 1
problems = ["Linear Problem (d=2)", "Linear Problem (d=50)"]
this_df = df_agg.query(" `Averaging Method`=='Average Estimate'")
this_df = this_df[this_df['Problem'].isin(problems)]
this_df = this_df[this_df['$N_{{ \\text{{obs}} }}$'].isin([2])]
this_df = this_df[this_df['$\\Delta_{{\\text{{Target}}}}$'].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 Size
# plot
fig = px.line(
    this_df,
    x = "Relative Root MSE",
    y="Cost Mean",
    facet_col="Method",
    facet_row="Problem",
    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("cvar_tgt == 1")
methods = this_df.Method.unique()
for prob in problems:
    for method in this_df.Method.unique():
        if "CBREE" == method:
            bm_solvers = [s for s in this_df_bm.Solver.unique() if "GM" in s]
        if "MFN" in method:
            bm_solvers = [s for s in this_df_bm.Solver.unique() if "MFN" in s]
        for bm_solver in bm_solvers:
            dat =this_df_bm.query("Problem == @prob & Solver == @bm_solver")
            trace_dict = {
                "x" : dat["Relative Root MSE"],
                "y" : dat["Cost Mean"],
                "legendgrouptitle_text": "Benchmark Methods",
                "showlegend" : prob == problems[1], # avoids duplicated entries
                "name": bm_solver,
                "legendgroup": "group",
                "mode": "markers+lines",
                "opacity": 0.8
            }
            num_rows = [1-i   for i,p in enumerate(problems) if prob == p] # cave flipped y-axis by px
            cols_idx = [i  for i, m in enumerate(methods) if method == m]
            trace_dict = trace_dict | BM_SOLVER_SCATTER_STYLE[bm_solver]
            fig = ree.add_scatter_to_subplots(fig, num_rows, cols_idx, **trace_dict)

# style
if  "yaxis_exponentformat" in MY_LAYOUT.keys():
    fig = ree.update_axes_format(fig,  MY_LAYOUT["xaxis_exponentformat"], MY_LAYOUT["yaxis_exponentforma
# overwrite N_obs = 0 etc
new_labels = {LATEX_TO_HTML[DF_COLUMNS_TO_LATEX["observation_window"]] + "=0": "No Divergence Check",
              "Method=":"",
              "Problem=":""}
for old, new in new_labels.items():
    fig.for_each_annotation(
        lambda a: a.update(text = a.text.replace(old,new) if a.text.startswith(old) else a.text))

fig.update_layout(**MY_LAYOUT)
# show and save
fig.show()
fig_title = "Linear problems lower and higher dimensions"
fig.write_image(f"{fig_title}.pdf".replace(" ", "_").lower(),
                engine="kaleido")
fig_description = f"Solving the Linear Problem with the CBREE methods using  \
different parameters. \
We vary the stopping criterion $\\Delta_{{\\text{{Target}}}}$ (color), \
the problem's dimension $d$ (row) and \
```

Skip to main content

```
and $N_\\text{{obs}}=2$ 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 ${int(2*this_df.Seed.uniqu
with open(f"{fig_title} desc.tex".replace(" ", "_").lower(), "w") as file:
    file.write(fig_description)
display(Markdown(fig_description))
```



Solving the Linear Problem with the CBREE methods using different parameters. We vary the stopping criterion Δ_{Target} (color), the problem's dimension d (row) and the method (column). The parameters $\epsilon_{\text{Target}} = 0.5$ and $N_\text{obs}=2$ 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.