In [1]:
```
%cd ../..
```

```
/data0/jang/masters
```

Get hists pipeline

In [2]:
```
saved = "ml_hep_sim/analysis/results/pulls/"
```

In [3]:
```
from ml_hep_sim.analysis.hists_pipeline import get_hists_pipeline
from ml_hep_sim.analysis.hists_pipeline import MakeHistsFromSamples
```

In [4]:
```
use_class = True
mc_test = False

if use_class and not mc_test:
    saved += "class_"
elif use_class and mc_test:
    saved += "class_mc_"
elif mc_test:
    saved += "mbb_mc_"
else:
    saved += "mbb_"

hists_pipeline = get_hists_pipeline(use_classifier=use_class, N_gen=10**6)
hists_pipeline.pipes = hists_pipeline.pipes[:-1] # this one only makes 1 hist
```

In [5]:
```
saved
```

Out[5]:
```
'ml_hep_sim/analysis/results/pulls/class_'
```

In [6]:
```
b_sig_bkg = hists_pipeline.pipes[-1]
```

Do a scan for different N values

In [7]:
```
import numpy as np

from ml_hep_sim.analysis.ul_pipeline import PullBlock
from ml_hep_sim.pipeline.pipes import Pipeline

from ml_hep_sim.analysis.ul_pipeline import pull_plot
```

In [8]:
```
Ns = np.linspace(10**3, 10**5, 40).astype(int)
```

In [9]:
```
pull_pipelines = []

bonly = False
bins = 22

sys_err = 0.1

for N in Ns:
    if use_class:
        bin_range = (0.5, 1.1)
        b_hists = MakeHistsFromSamples(bin_range=bin_range, bins=bins, N_sig=N / 10, N_bkg=N,
                                       N_gen=10**6, bonly=bonly)(b_sig_bkg)
    else:
        bin_range = (0.01, 3.0)
        b_hists = MakeHistsFromSamples(bin_range=bin_range, bins=bins, N_sig=N / 10, N_bkg=N,
                                       N_gen=10**6, bonly=bonly)(b_sig_bkg)

    b_pull = PullBlock(bkg_err=sys_err, mc_test=mc_test)(b_hists)
    pipe = Pipeline()
    pipe.compose(hists_pipeline, b_hists, b_pull)

    pull_pipelines.append(pipe)
```

In [10]:
```
for pull_pipeline in pull_pipelines:
    pull_pipeline.fit()
```

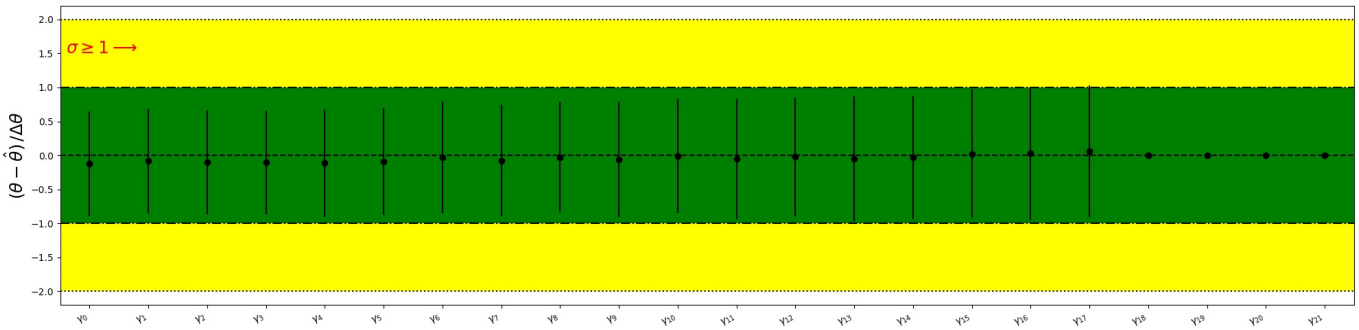
```
100%|| 10/10 [00:09<00:00,  1.04it/s]
100%|| 10/10 [00:11<00:00,  1.11s/it]
100%|| 10/10 [00:00<00:00, 3731.26it/s]
100%|| 10/10 [00:00<00:00, 9815.83it/s]
100%|| 10/10 [00:00<00:00, 879.46it/s]
100%|| 10/10 [00:00<00:00, 902.39it/s]
```

In [34]:
```
pull_pipelines[-1].draw_pipeline_tree(to_graphviz_file="ml_hep_sim/analysis/results/pulls/pull_pipe", block_idx=
```

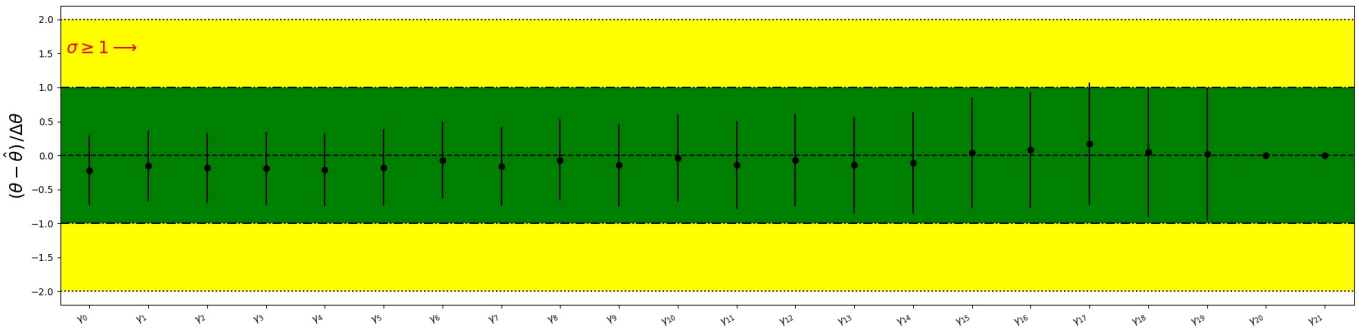
Out[34]:
```
<treelib.tree.Tree at 0x7f1fc25643d0>
```

In [12]:
```
pulls_lst, pullerr_lst, mus, mus_err, twice_nlls = [], [], [], [], []

for i, N in enumerate(Ns):
    print(N)
    p = pull_pipelines[i].pipes[-1]
    pulls, pullerr, errors, labels = p.results

    pull_plot(pulls, pullerr, errors, labels) #, save=saved + f"pull_{N}.pdf")
    # pull_plot(pulls, pullerr, errors, labels)

    pulls_lst.append(pulls)
    pullerr_lst.append(pullerr)
    mus.append(p.bestfit[0][0])
    mus_err.append(errors[0])
    twice_nlls.append(p.bestfit[1])
```

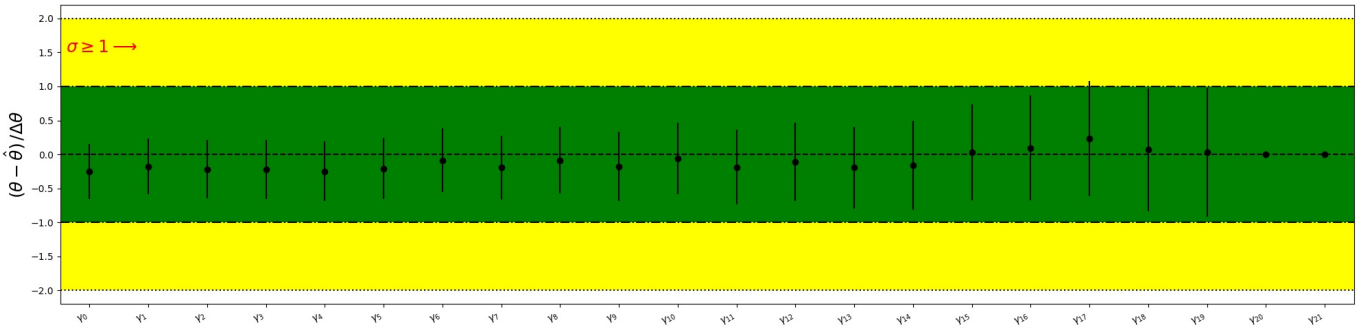
```
1000
```



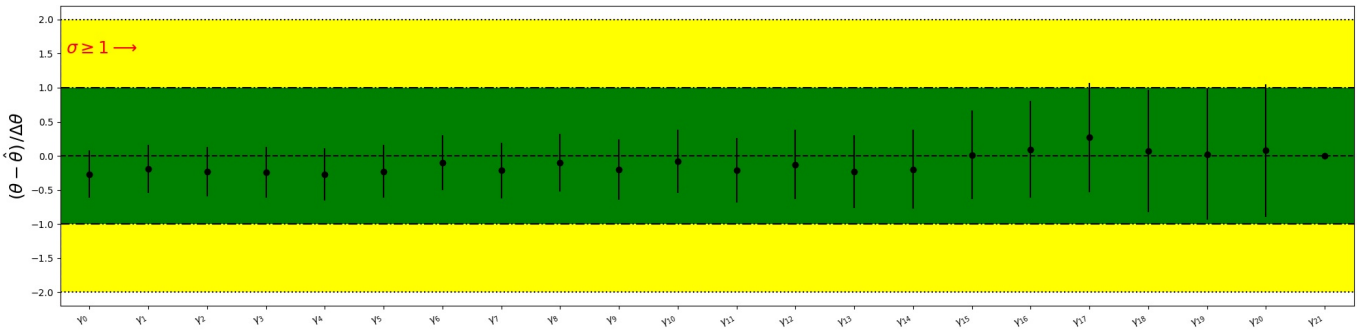

```
3538
```



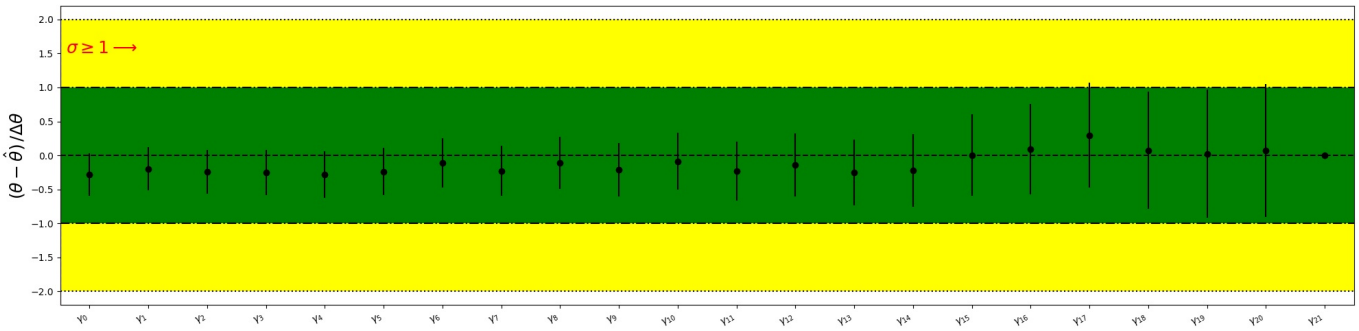

```
6076
```



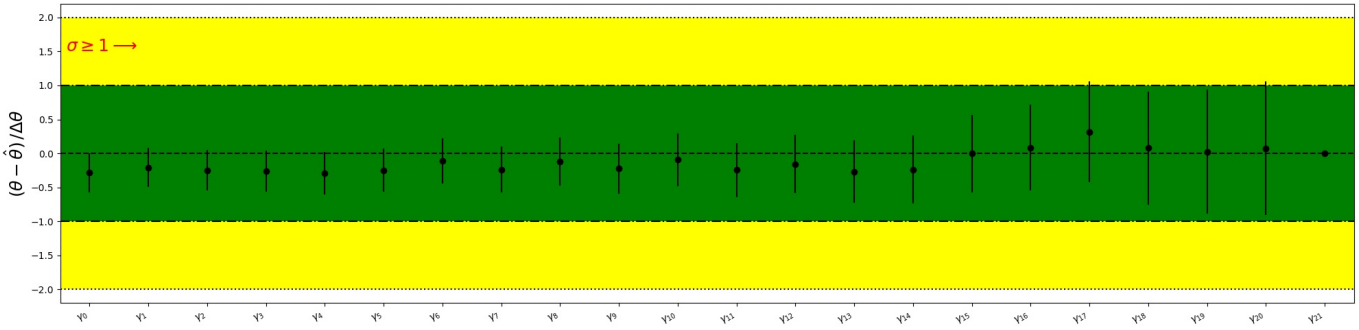
8615



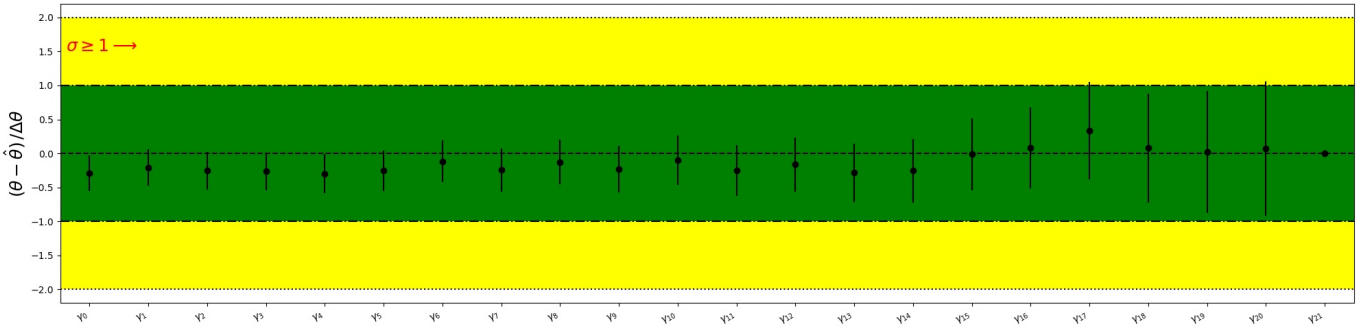
11153



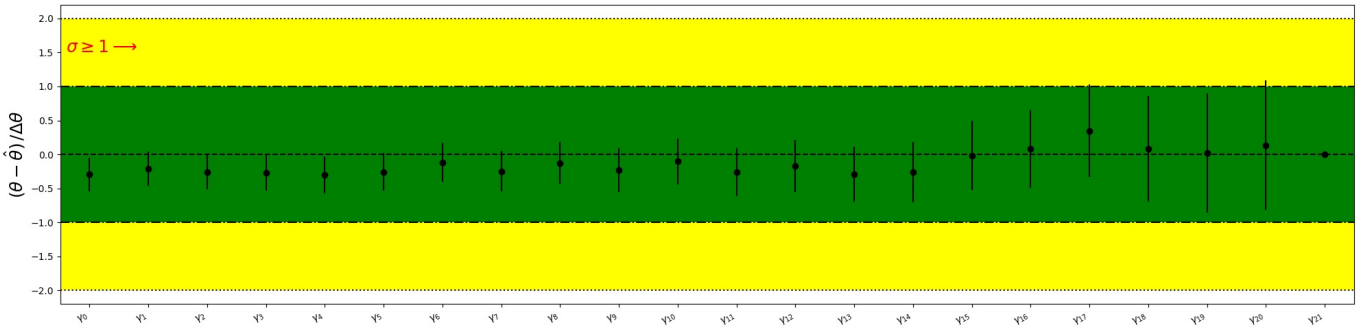
13692



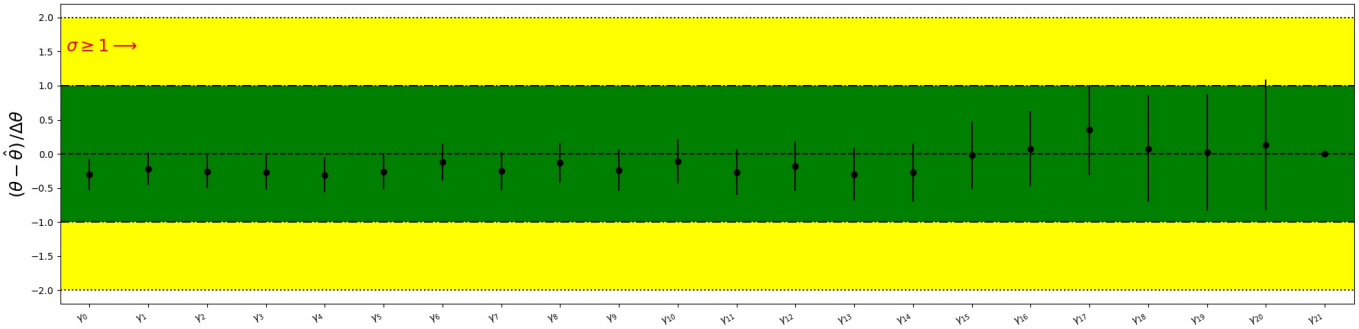
16230



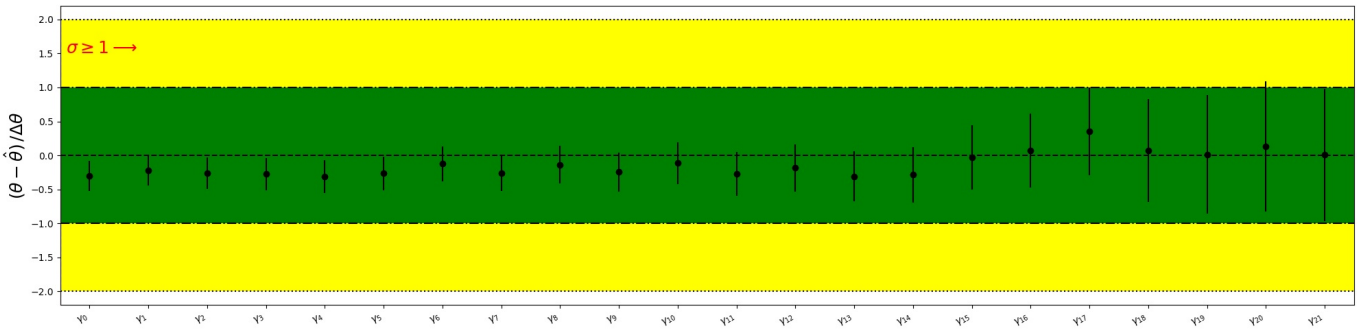
18769



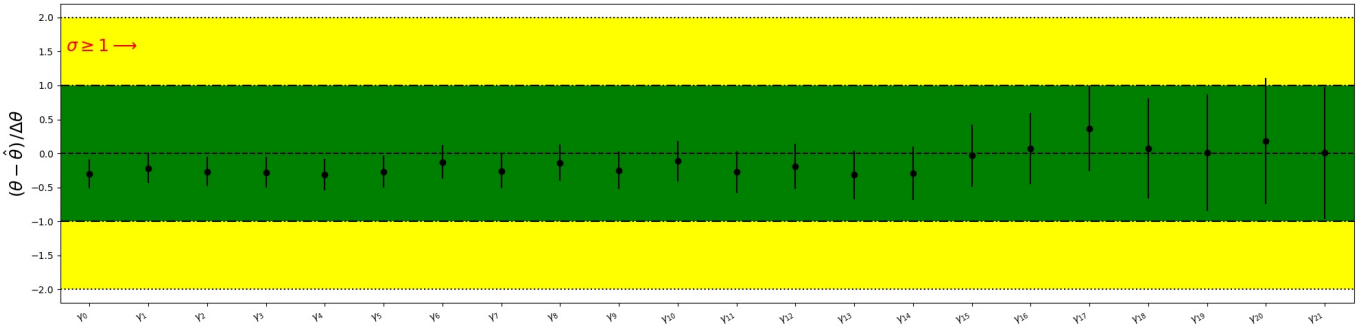
21307



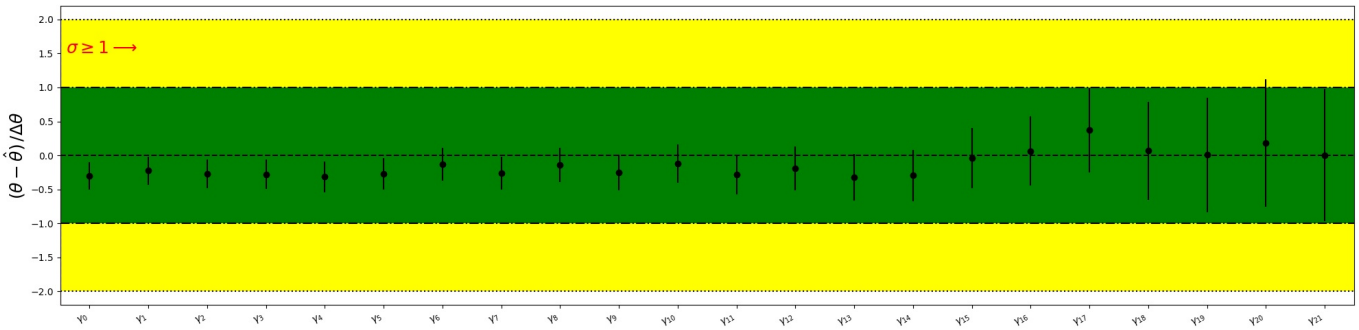
23846



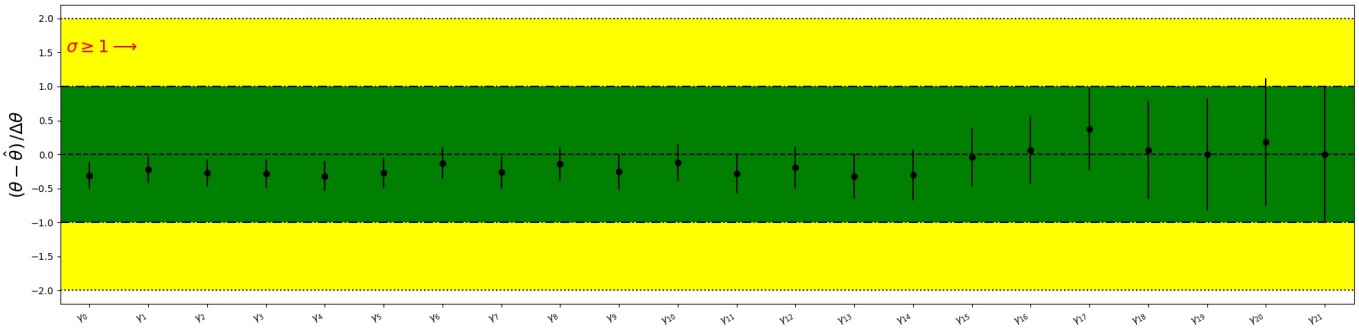
26384



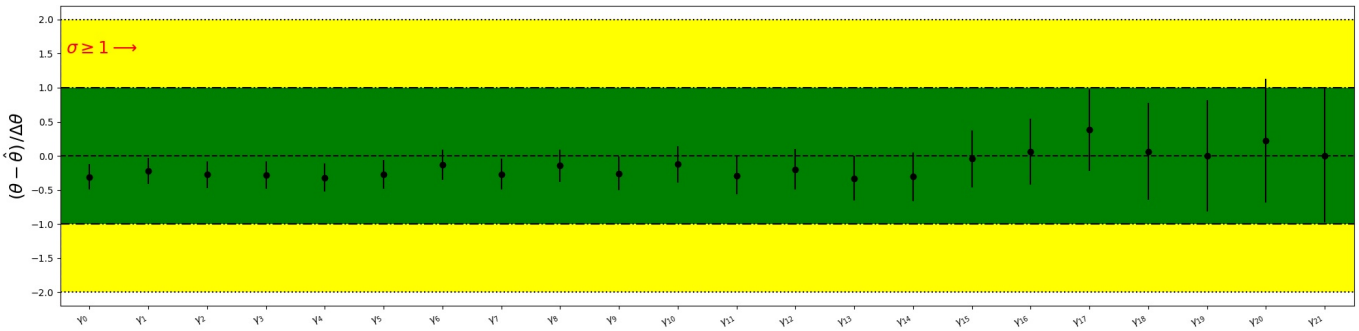
28923



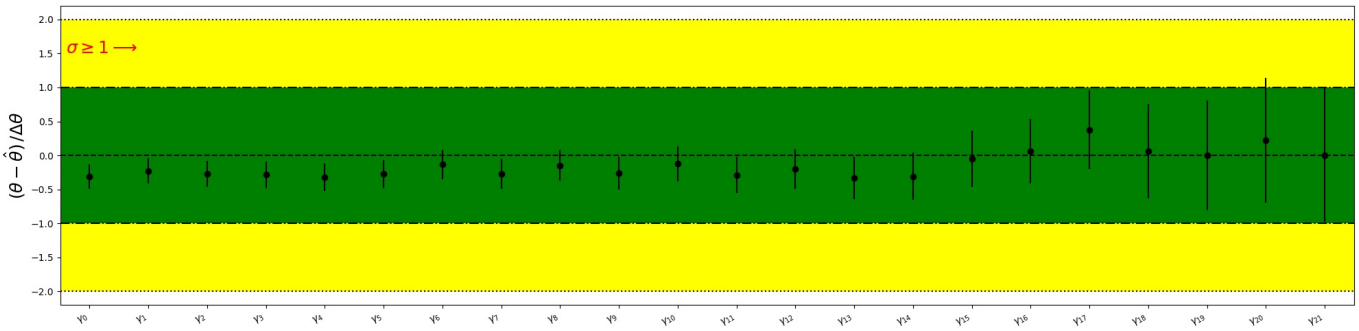
31461



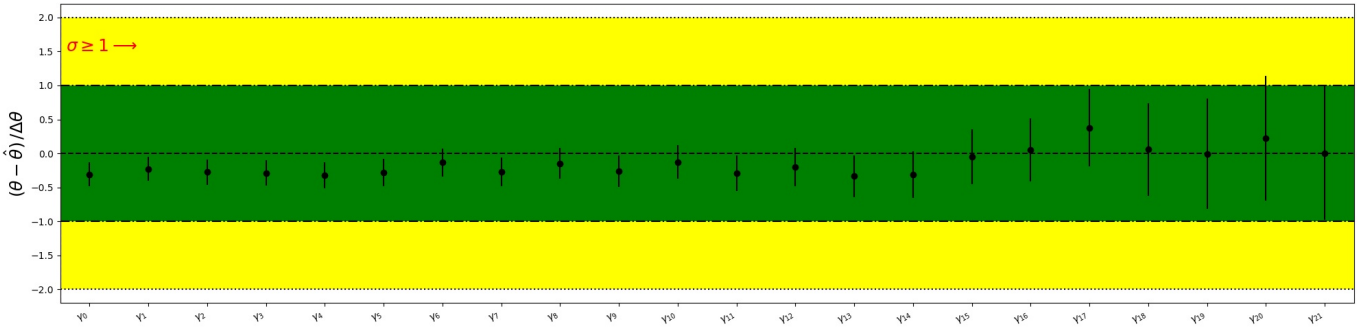
34000



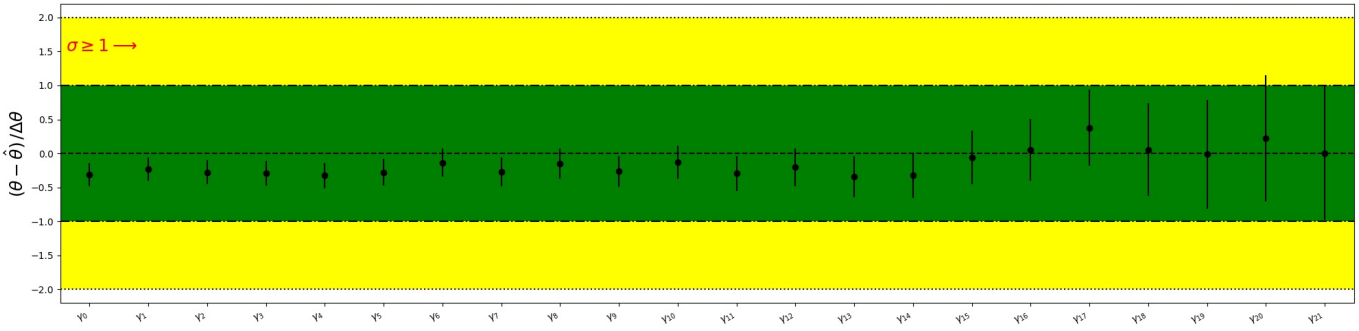
36538



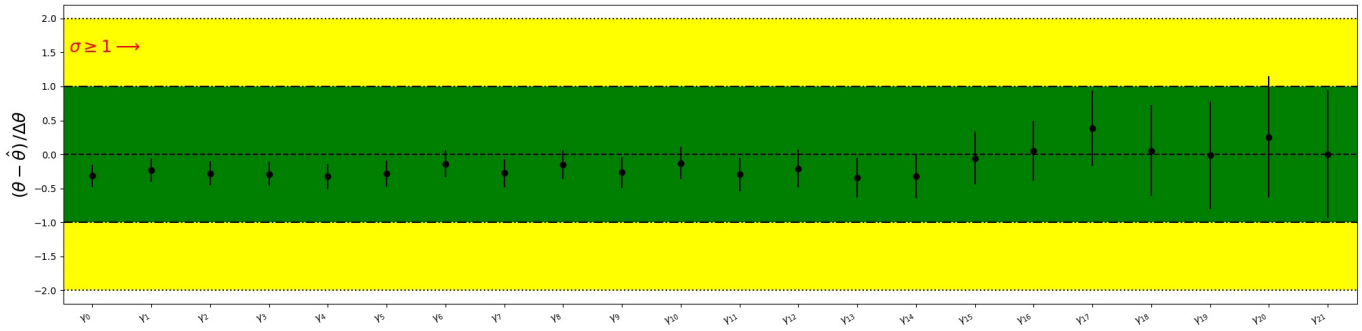
39076



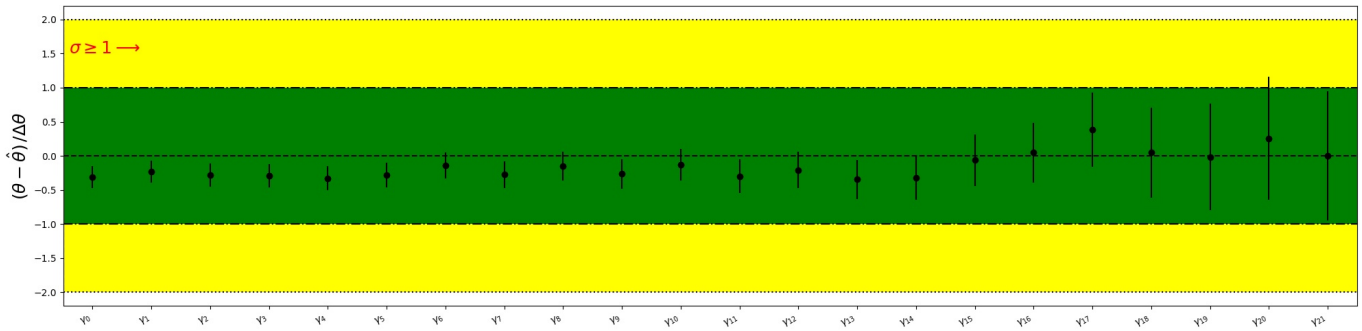
41615



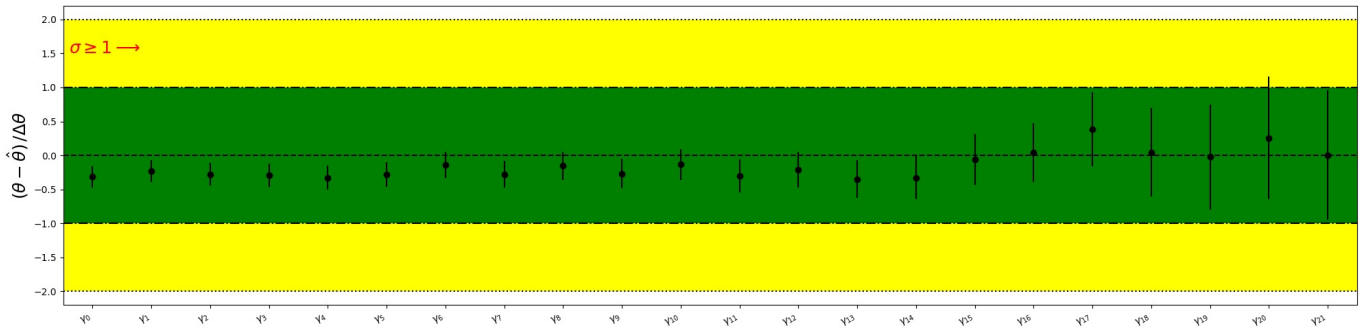
44153



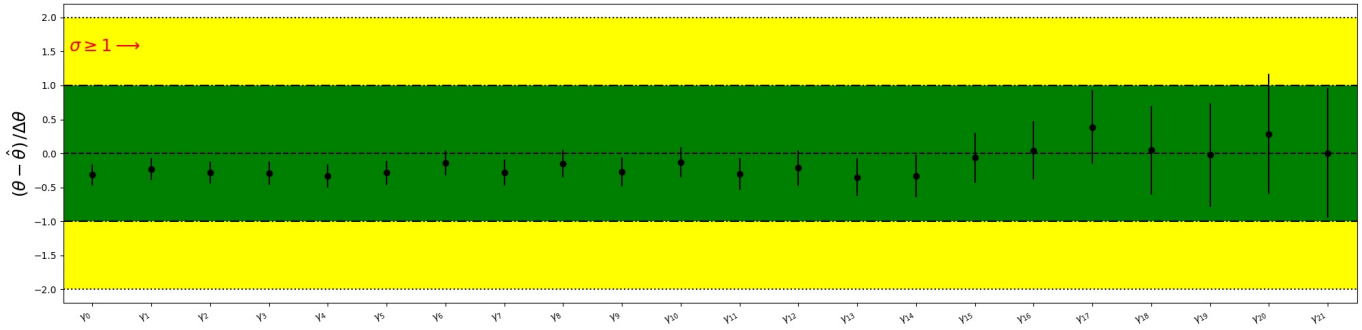
46692



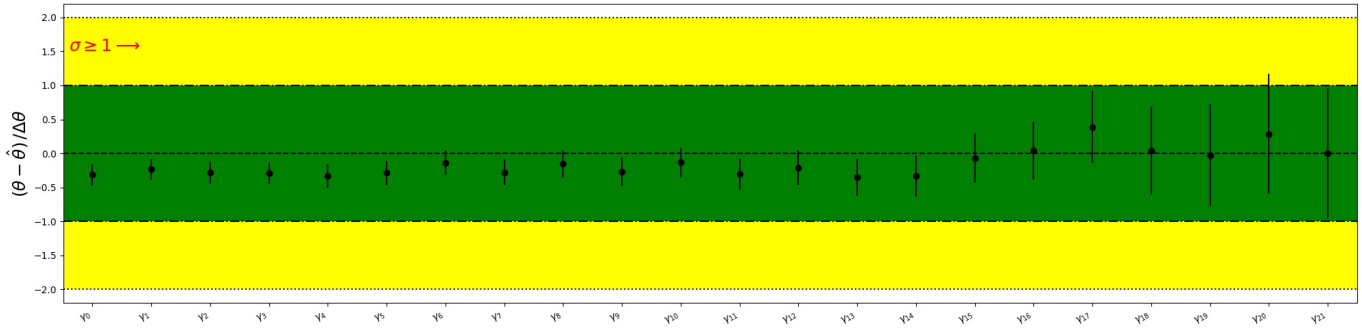
49230



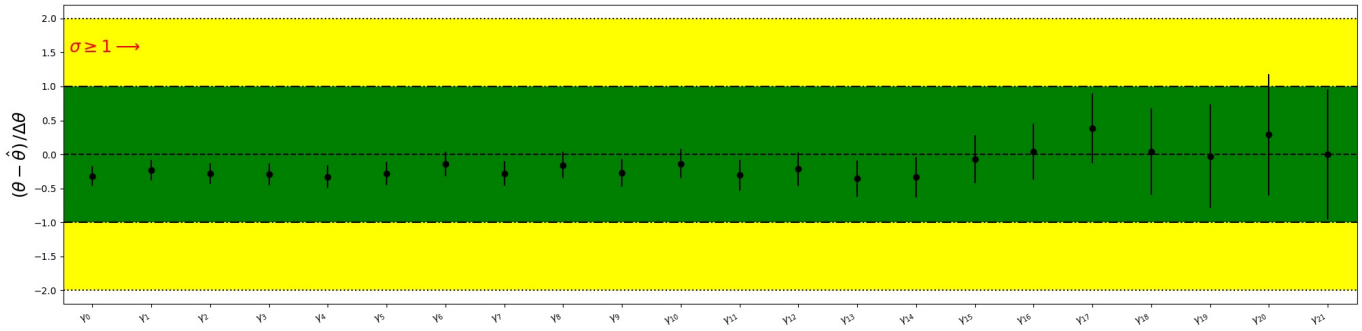
51769



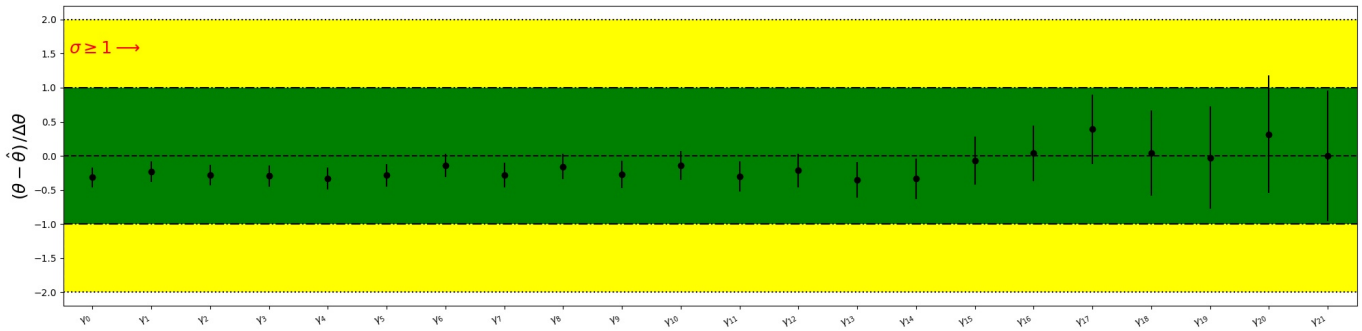
54307



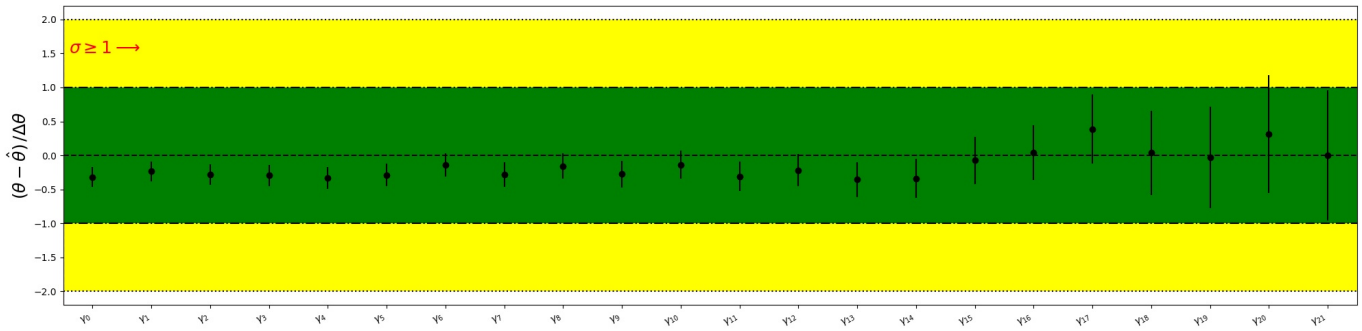
56846



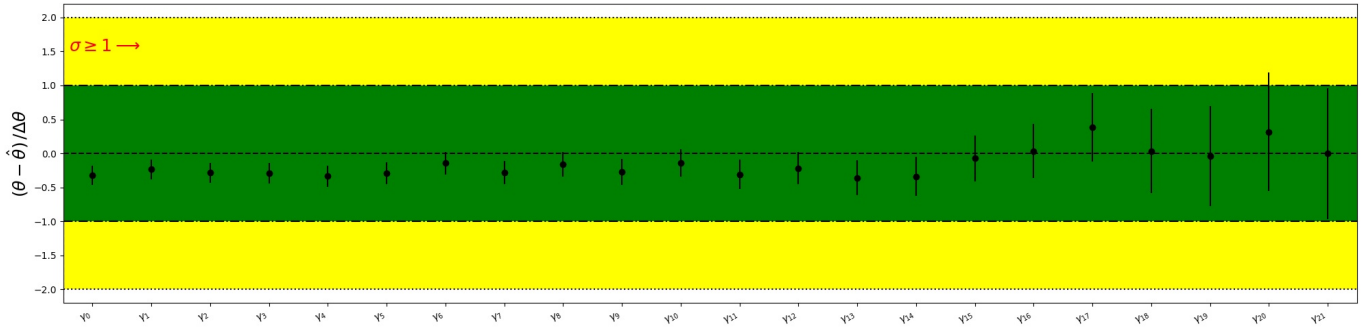
59384



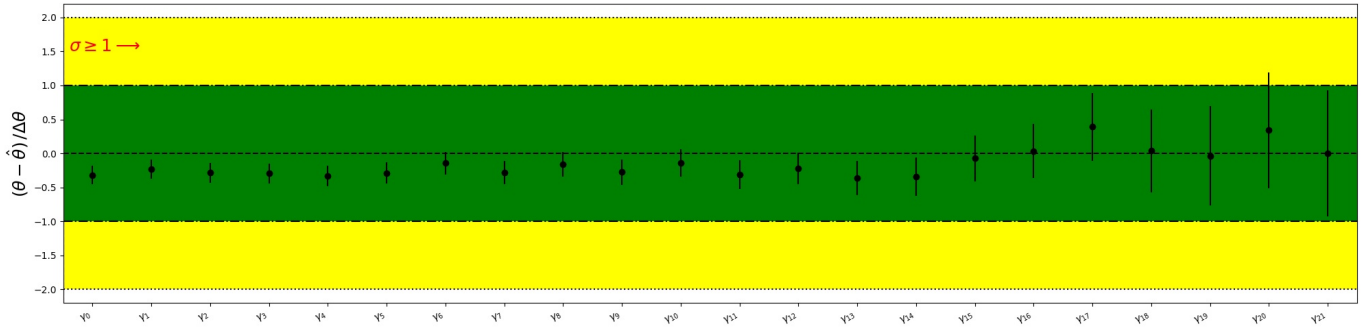
61923



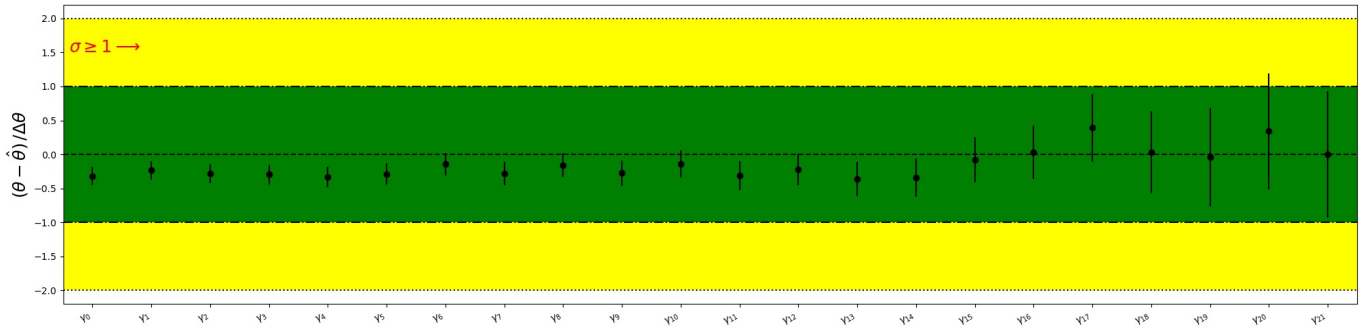
64461



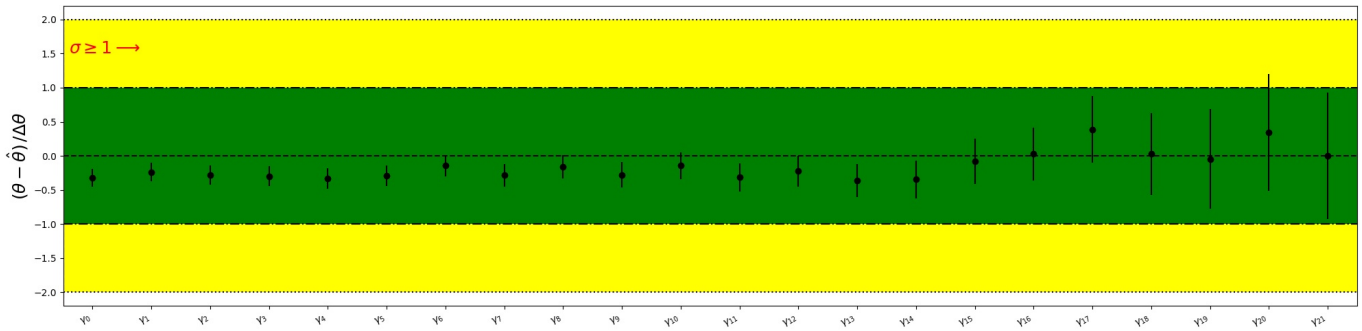
67000



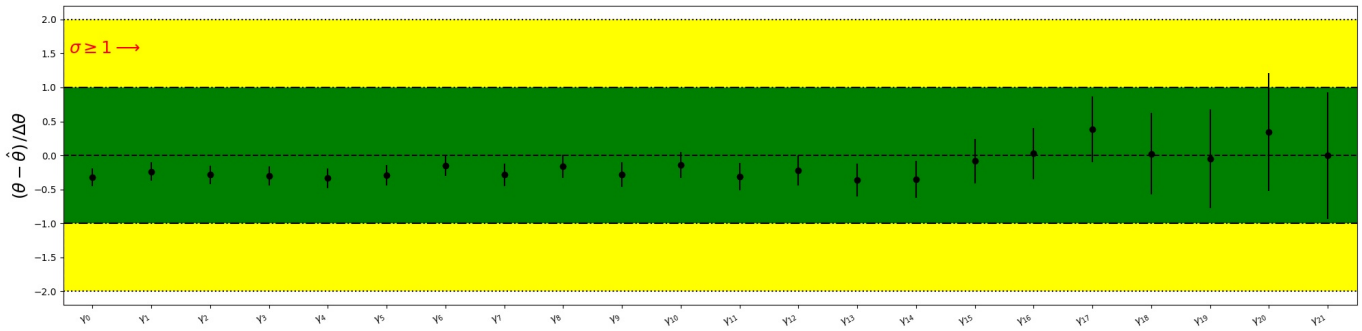
69538



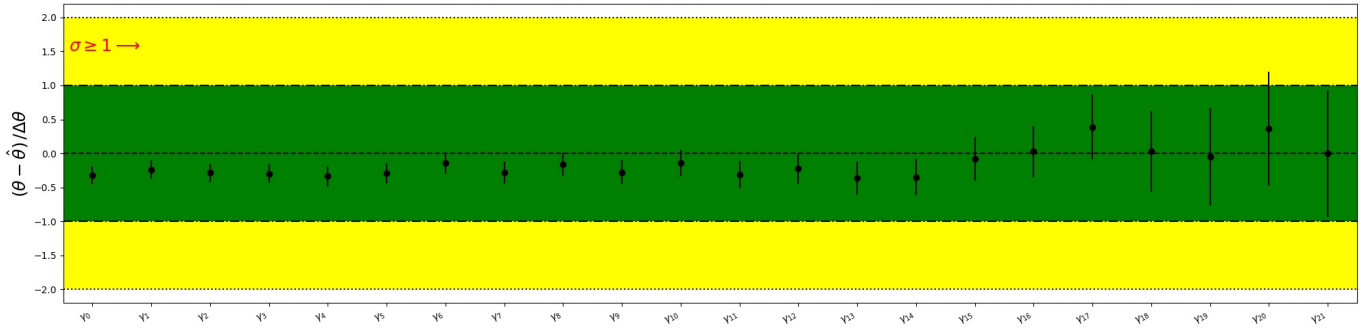
72076



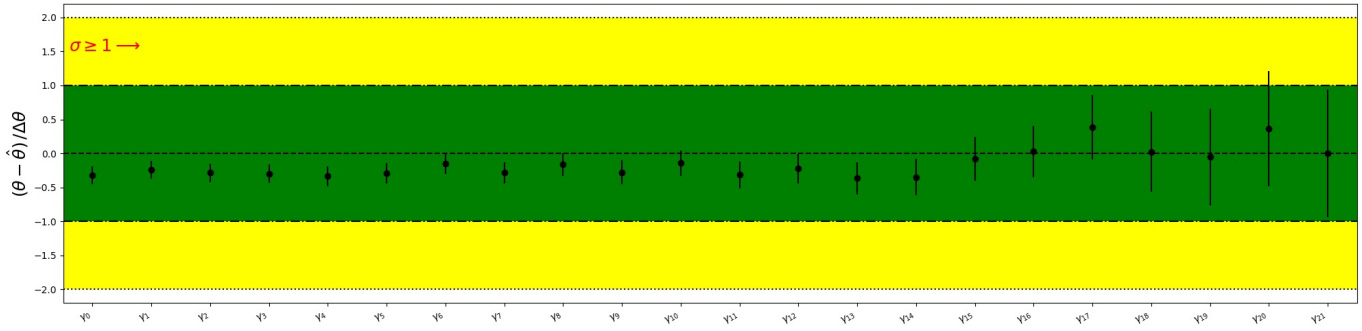
74615



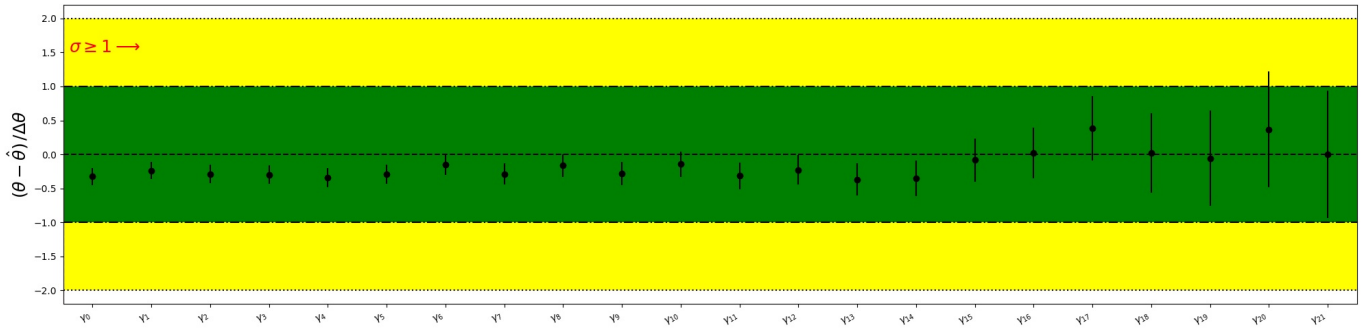
77153



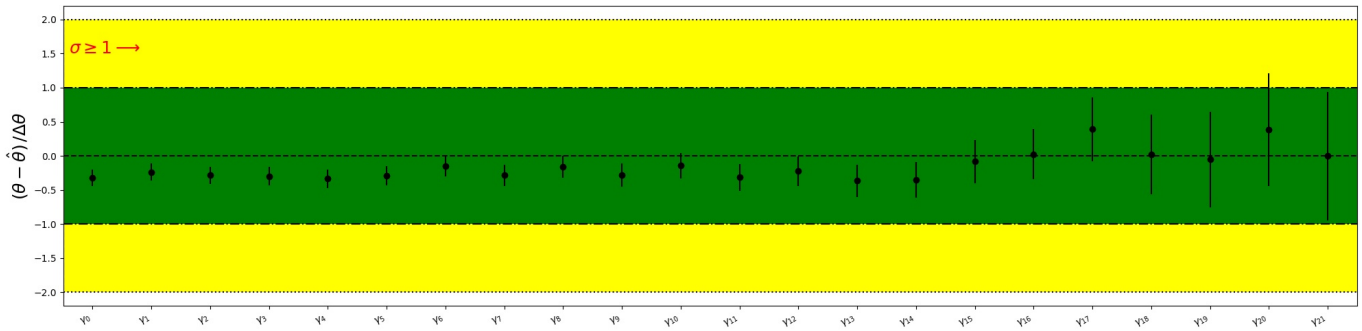
79692



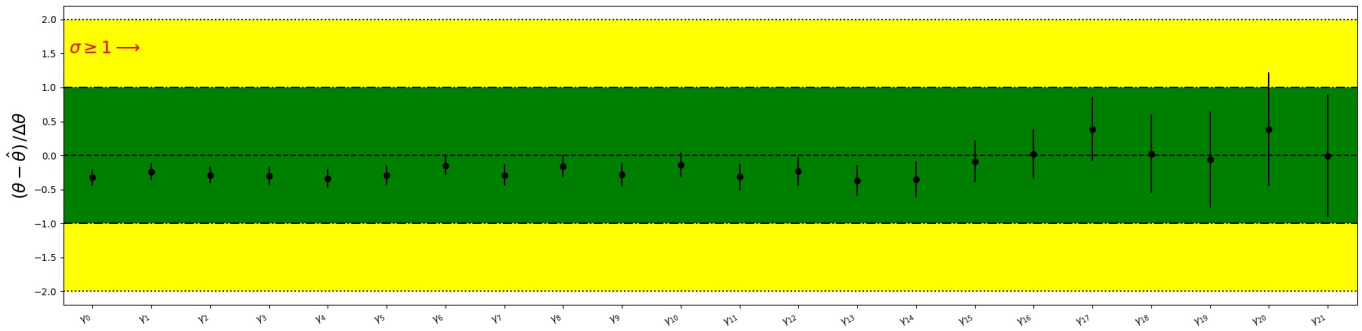
82230



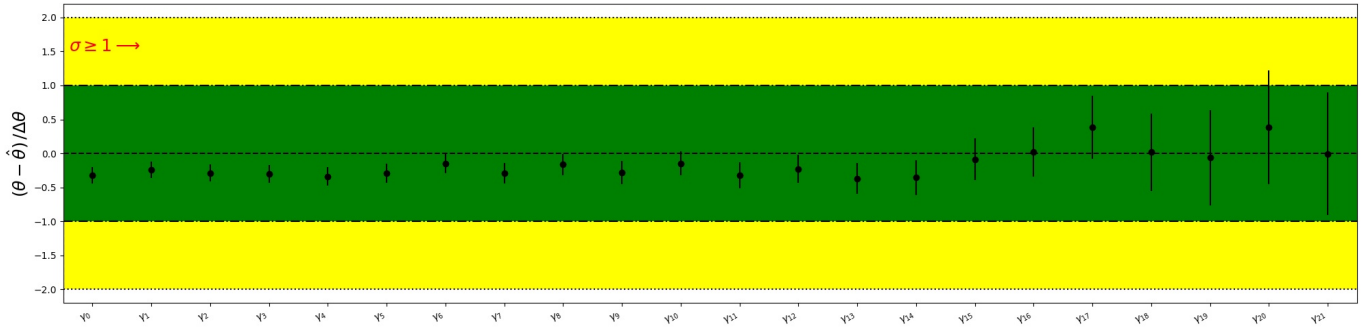
84769



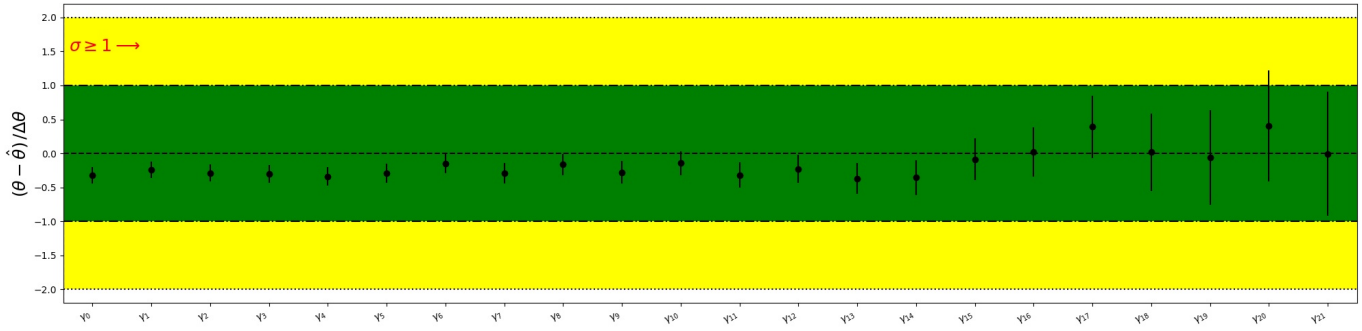
87307



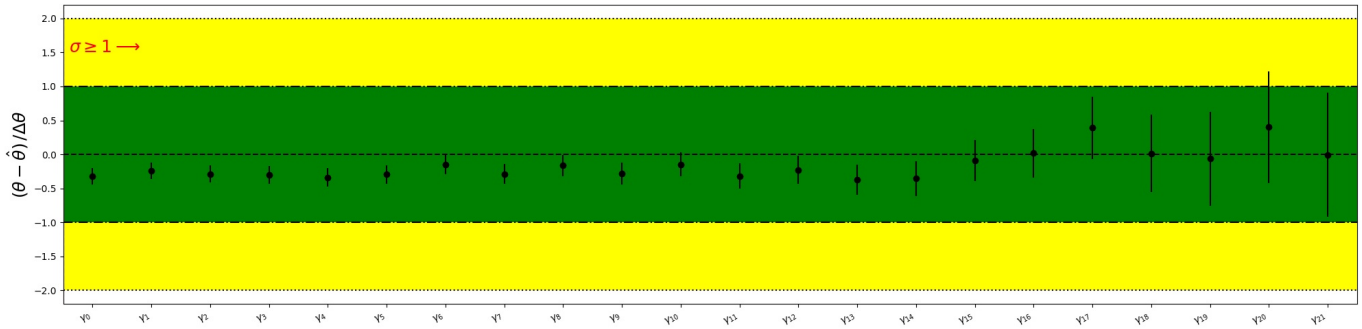
89846



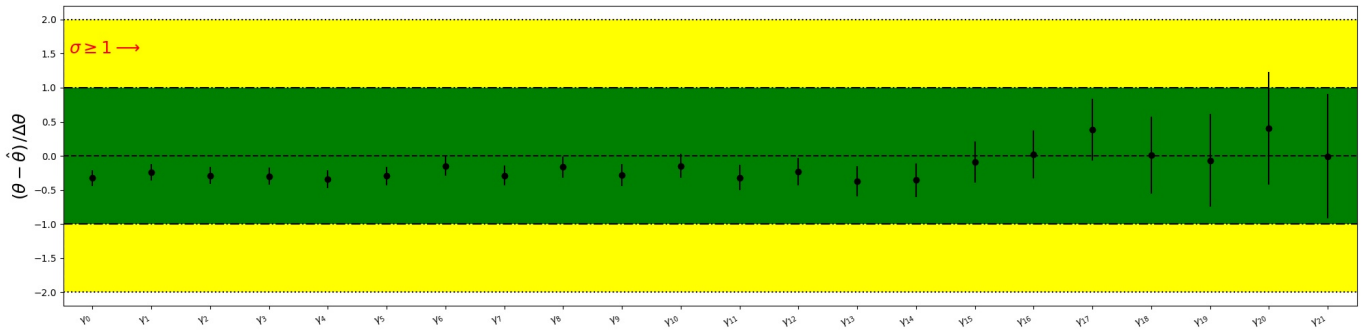
92384



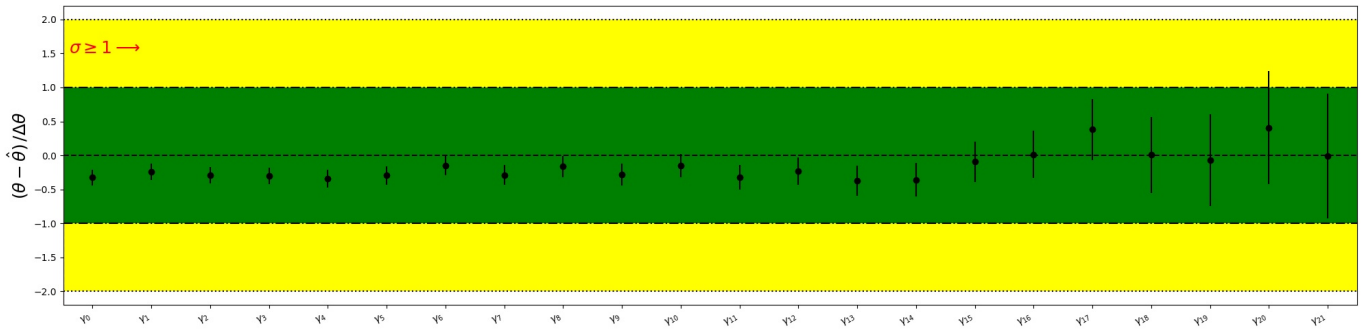
94923



97461



100000



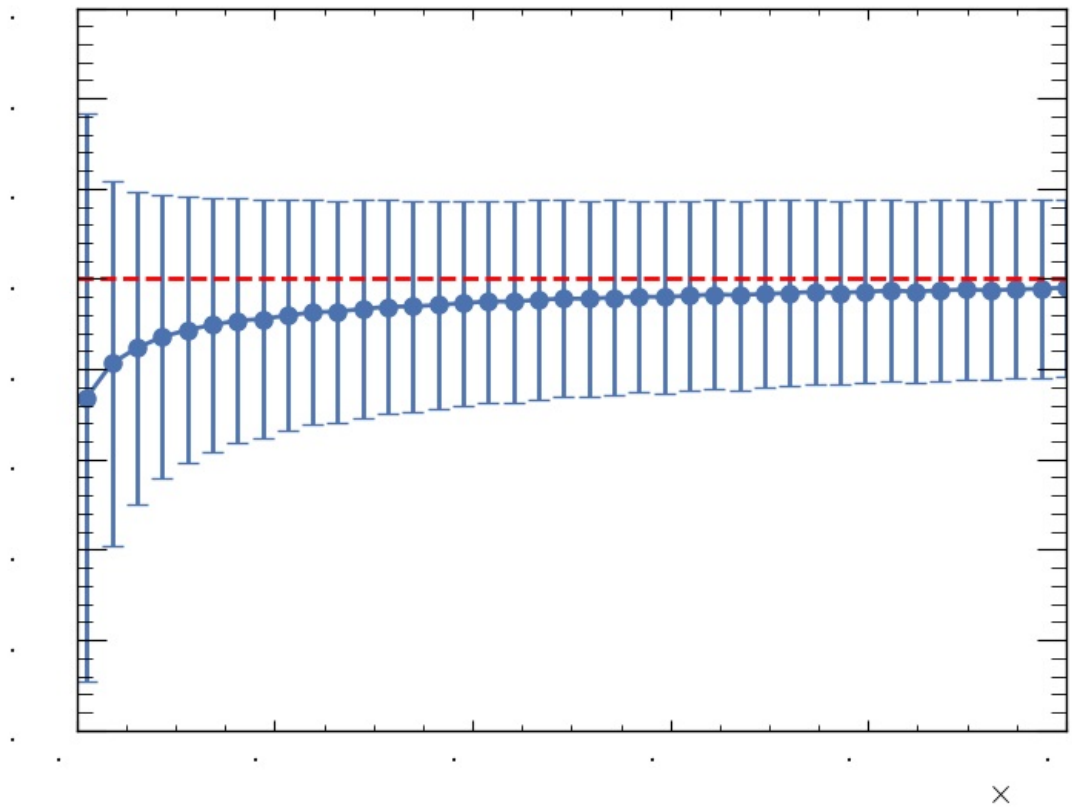
Mu

In [13]:
```
from uncertainties import unumpy, ufloat

import matplotlib.pyplot as plt
from ml_hep_sim.plotting.style import style_setup, set_size

set_size()
style_setup(seaborn_pallete=True)
```

In [14]:
```
eff_s_mc, eff_b_mc, eff_s_ml, eff_b_ml = pull_pipelines[0].pipes[-2].histograms["eff"]
```

In [15]:
```
plt.errorbar(Ns, mus, mus_err, capsize=5)
plt.scatter(Ns, mus)
plt.xlabel(r"$\nu_B$", loc="center")
plt.ylabel(r"$\mu$")
plt.tight_layout()
plt.axhline(0.1, c='r', ls='--')
# plt.savefig(saved + "mus.pdf")
```

Out[15]:
```
<matplotlib.lines.Line2D at 0x7f1e84523410>
```



In [16]:
```
unumpy.uarray(mus, mus_err)
```

Out[16]:
```
array([0.0868552230316927+/-0.031523943186854794,
       0.0906278877251446+/-0.020196045781404347,
       0.09234619857754008+/-0.017344574360954643,
       0.09356040240492665+/-0.015683756020434668,
       0.09432478662358411+/-0.014748094599927847,
       0.09489634431494473+/-0.014055823429678226,
       0.09536907187028927+/-0.013553504395749524,
       0.09553958937006372+/-0.013262347680793977,
       0.0959448729633969+/-0.012827994637787082,
       0.09627956572292737+/-0.01244965831459003,
       0.09638739148159993+/-0.012273286705746143,
       0.09664869306657116+/-0.01206112353317458,
       0.09691619046143095+/-0.011806901467328892,
       0.09694150920067544+/-0.011713170673038745,
       0.09710089941446651+/-0.011584572636974419,
       0.09729929448374142+/-0.01136046733896872,
       0.09750036517422414+/-0.011184680833506697,
       0.09747748481575892+/-0.01116852950581719,
       0.09765606370847477+/-0.01104891131800579,
       0.09782237615566304+/-0.010942069125049449,
       0.09778295011235062+/-0.010874494977934412,
       0.09793875129211295+/-0.010780041475932298,
       0.09803700614459473+/-0.010643193686658933,
       0.09797667560663442+/-0.010638728626541256,
       0.09812028005259472+/-0.010553260029147318,
       0.09826918002227407+/-0.0104476470183874,
       0.09818767183444169+/-0.010475691649701004,
       0.0983229429778925+/-0.010400917509520956,
       0.09843563058242626+/-0.010278776704867182,
       0.09856159816396327+/-0.010186393316034499,
       0.09845331441015336+/-0.010206231495271424,
       0.09857174149039272+/-0.010144149772709478,
       0.09868758621699901+/-0.010087932631673888,
       0.09859994618987608+/-0.010072098606681124,
       0.0987006632061702+/-0.010009269287779503,
       0.0988108996631145+/-0.009952368741579724,
       0.0987210584215069+/-0.009966380273643938,
       0.09882202047914865+/-0.009894727476696646,
       0.09890856790835513+/-0.009854140146806668,
       0.09900919406379671+/-0.009806017386215879], dtype=object)
```

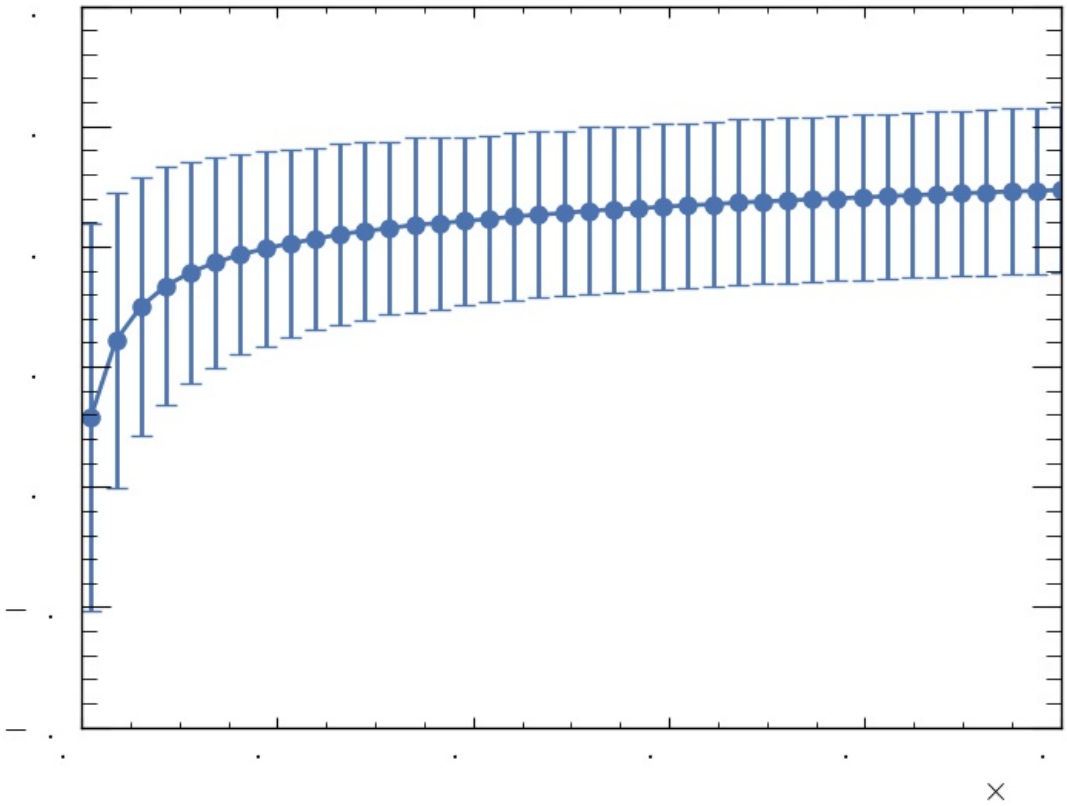
Gamma means and stds

In [17]:
```
gammas = [] # list of gamma and error for each N

for g_val, g_err in zip(pulls_lst, pullerr_lst):
    lst = []
    for v, err in zip(g_val, g_err):
        lst.append(ufloat(v, err))

    gammas.append(np.array(lst))
```

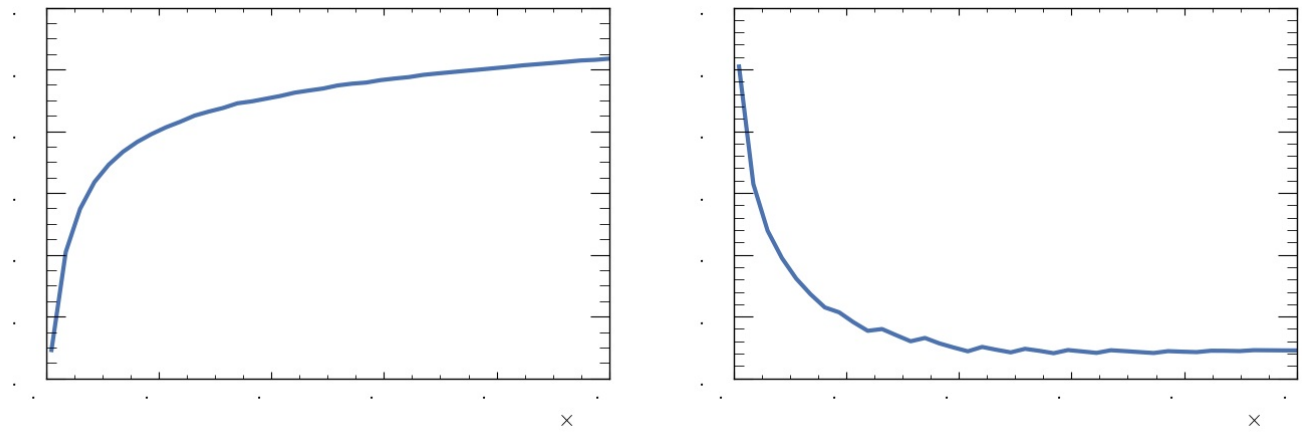
In [18]:
```
g_res = []

for g in gammas:
    g_res.append(np.sum(g**2)**0.5)
```

In [19]:
```
g_nom, g_err = [], [] # split to nominal and std

for g in g_res:
    g_nom.append(g.nominal_value)
    g_err.append(g.std_dev)
```

In [20]:
```
plt.scatter(Ns, g_nom)
plt.errorbar(Ns, g_nom, g_err, capsize=5)
plt.xlabel(r"$\nu_B$", loc='center')
plt.ylabel(r"$\gamma$")
plt.tight_layout()
# plt.savefig(saved + "gamma_N.pdf")
```



In [21]:
```
fig, axs = plt.subplots(1, 2, figsize=(14, 5))
axs = axs.flatten()

axs[0].plot(Ns, g_nom, lw=3)
axs[1].plot(Ns, g_err, lw=3)

axs[0].set_ylabel(r"$\sum_b \gamma_b^2$")
axs[1].set_ylabel(r"$\sum_b \gamma_b^2$ errors")

axs[0].set_xlabel(r"$\nu_B$", loc='center')
axs[1].set_xlabel(r"$\nu_B$", loc='center')
plt.tight_layout()
# plt.savefig(saved + "gamma_N_err_class.pdf")
```



Hists

In [22]:
```
if mc_test is True:
    raise ValueError
```

In [23]:
```
idx = 0
N_idx = Ns[idx]

pull_pipe = pull_pipelines[idx].pipes[-1] # idx depends on N

hists = pull_pipe.histograms
mu = pull_pipe.bestfit[0][0]
gamma = pull_pipe.bestfit[0][1:]
mu_err, gamma_err = pull_pipe.results[-2][0], pull_pipe.results[-2][1:]
```

```
hist_pipe = pull_pipelines[idx].pipes[-2]
bkg_gen = hist_pipe.bkg_generated_data
```

In [24]:
```
alpha = hist_pipe.alpha
```

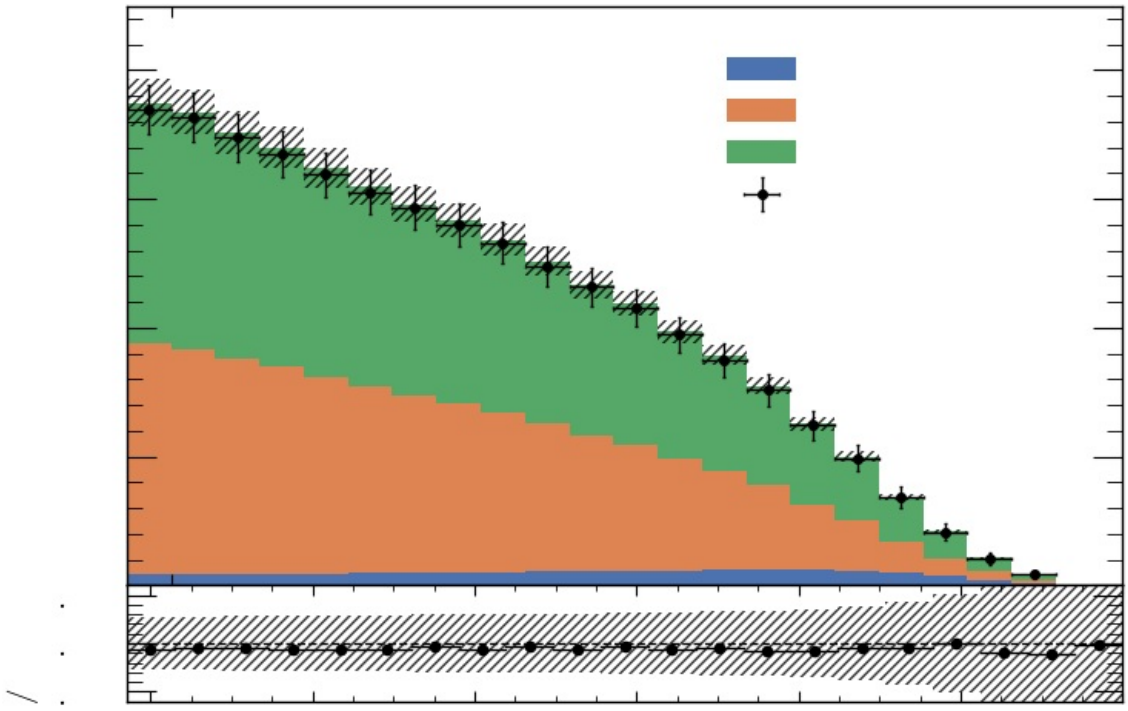
Errors

In [25]:
```
from ml_hep_sim.plotting.hep_plots import StackPlot
```

In [26]:
```
errors = pull_pipe.errors

data_err = errors["data_mc"]
bkg_err = np.sqrt(errors["nu_b_ml"] ** 2 + (hists["bkg_gen"] * sys_err)**2)
```

Prefit

In [27]:
```
x = np.arange(0, bins, 1)

sp = StackPlot(
    x,
    hists_lst=[alpha * hists["sig_mc"], hists["bkg_gen"], alpha * hists["sig_mc"] + hists["bkg_gen"]],
    data_hist=alpha * hists["sig_mc"] + hists["bkg_gen"] + hists["data_mc"],
)

sp.plot_stack(labels=["MC sig", "ML bkg", "MC sig + ML bkg"])

sp.plot_data(label="MC data", err=data_err, fmt='.', capsize=1, lw=1)

sp.plot_mc_errors(bkg_err)

counts_num, counts_den = hists["data_mc"], alpha* hists["sig_mc"] + hists["bkg_gen"]

counts_num_err = data_err
counts_den_err = bkg_err

sp.plot_lower_panel(counts_num, counts_den, counts_num_err, counts_den_err, ylabel="data$/$ML",
                    label_x_start=bin_range[0],
                    label_x_end=bin_range[1])

ax = sp.ax
ax_lower = sp.ax_lower

if use_class:
    ax_lower.set_xlabel("class. output")
else:
    ax_lower.set_xlabel("$m_{bb}$")

ax.set_ylabel("$N$")

# ax.set_ylim(0, 250)
ax_lower.set_ylim(0.7, 1.3)

plt.legend(loc='upper right')
plt.tight_layout()

# plt.savefig(saved + f"stacked_prefit_{N_idx}.pdf")
```

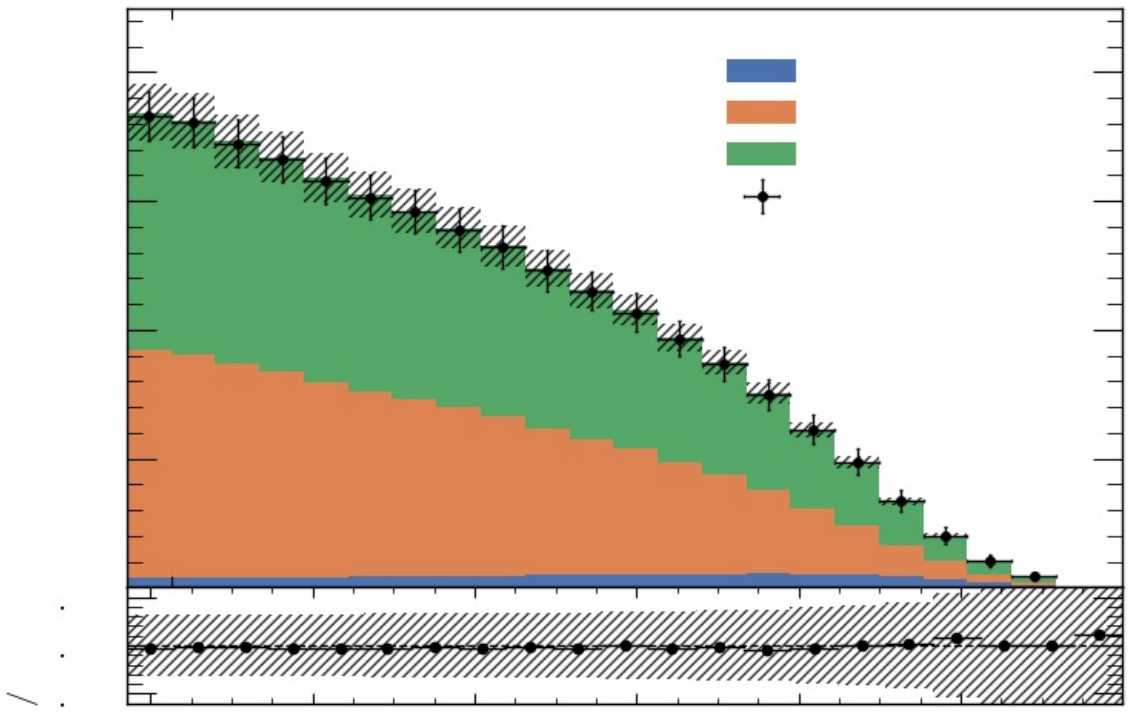


Postfit

In [28]:
```
gammas = unumpy.uarray(gamma, gamma_err)
bkg = hists["bkg_gen"] # unumpy.uarray(hists["bkg_gen"], bkg_err)

bkg_postfit = bkg * gammas
sig_postfit = ufloat(mu, mu_err) * hists["sig_mc"]

bkg_postfit_err = np.sqrt(unumpy.std_devs(bkg_postfit) ** 2 + (bkg * sys_err) ** 2) # unumpy.std_devs(bkg_postf
sig_postfit_err = unumpy.std_devs(sig_postfit)

bkg_postfit_val = unumpy.nominal_values(bkg_postfit)
sig_postfit_val = unumpy.nominal_values(sig_postfit)
```

In [29]:
```
sp = StackPlot(
    x,
    hists_lst=[sig_postfit_val, bkg_postfit_val, sig_postfit_val + bkg_postfit_val],
    data_hist=sig_postfit_val + bkg_postfit_val + hists["data_mc"],
)

sp.plot_stack(labels=["MC sig", "ML bkg", "MC sig + ML bkg"])

sp.plot_data(label="MC data", err=data_err, fmt='.', capsize=1, lw=1)

sp.plot_mc_errors(bkg_postfit_err)

counts_num, counts_den = hists["data_mc"], sig_postfit_val + bkg_postfit_val

counts_num_err = data_err
counts_den_err = bkg_postfit_err

sp.plot_lower_panel(counts_num, counts_den, counts_num_err, counts_den_err, ylabel="data$/$ML",
                    label_x_start=bin_range[0],
                    label_x_end=bin_range[1])

ax = sp.ax
ax_lower = sp.ax_lower

if use_class:
    ax_lower.set_xlabel("class. output")
else:
    ax_lower.set_xlabel("$m_{bb}$")

ax.set_ylabel("$N$")

# ax.set_ylim(0, 250)
ax_lower.set_ylim(0.7, 1.3)
```

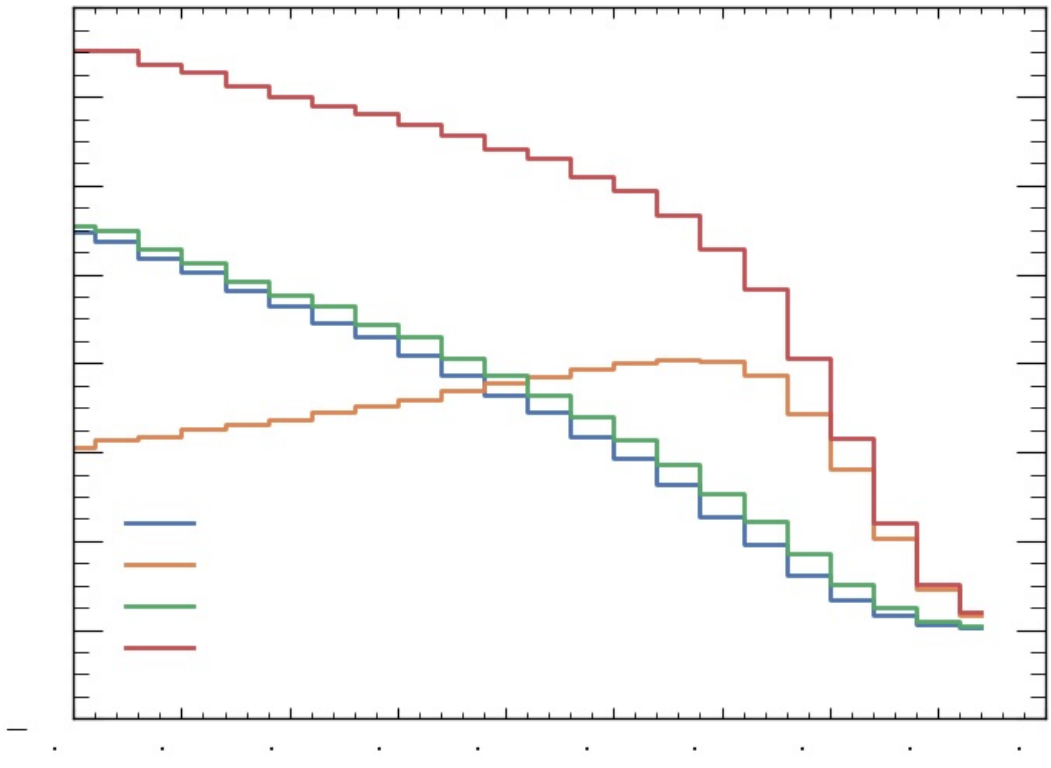
```
plt.legend(loc='upper right')
plt.tight_layout()

# plt.savefig(saved + f"stacked_postfit_{N_idx}.pdf")
```



Ratio

In [30]:
```
plt.step(x, hists["bkg_gen"], where="mid", label="bkg gen")
plt.step(x, hists["sig_mc"], where="mid", label="sig mc") # scale by mu from pyhf
plt.step(x, hists["data_mc"], where="mid", label="data mc")
plt.step(x, hists["sig_mc"] + hists["bkg_gen"], where="mid", label="sig mc + bkg gen")
plt.legend()
```

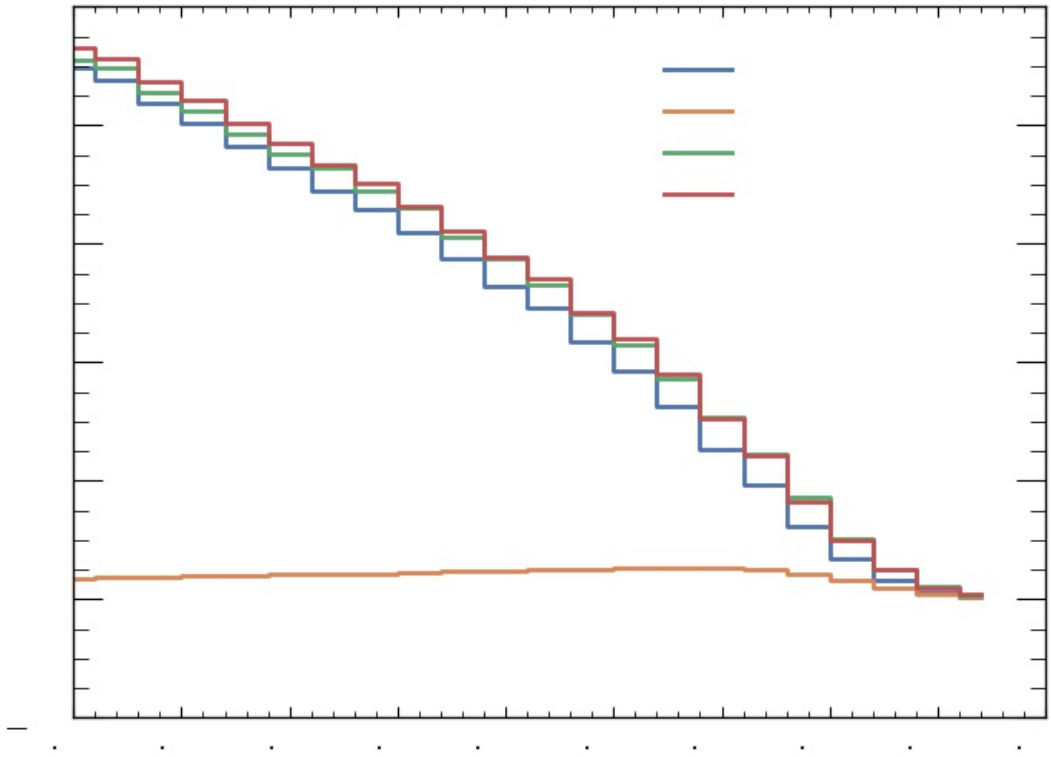
Out[30]:
```
<matplotlib.legend.Legend at 0x7f1e86d89910>
```



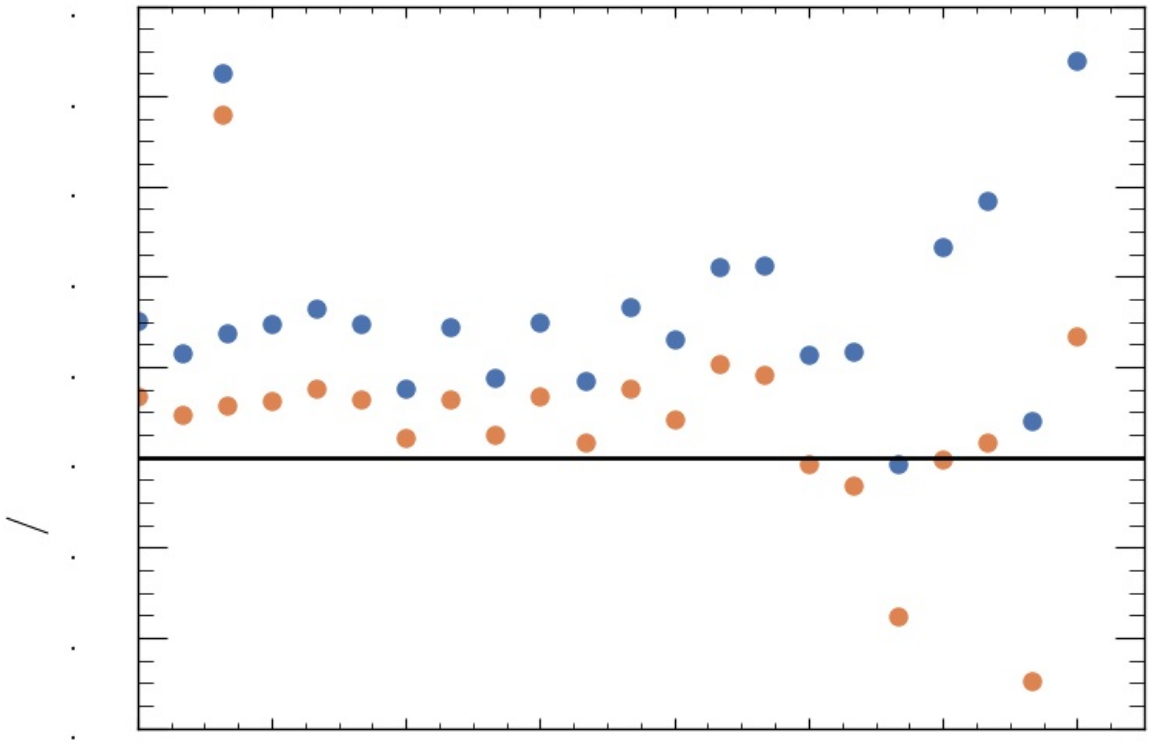
In [31]:
```
plt.step(x, hists["bkg_gen"], where="mid", label="bkg gen")
plt.step(x, mu * hists["sig_mc"], where="mid", label="sig mc") # scale by mu from pyhf
plt.step(x, hists["data_mc"], where="mid", label="data mc")
```

```
plt.step(x, mu * hists["sig_mc"] + hists["bkg_gen"], where="mid", label="sig mc + bkg gen")
plt.legend()
```

Out[31]: <matplotlib.legend.Legend at 0x7f1e844b9910>

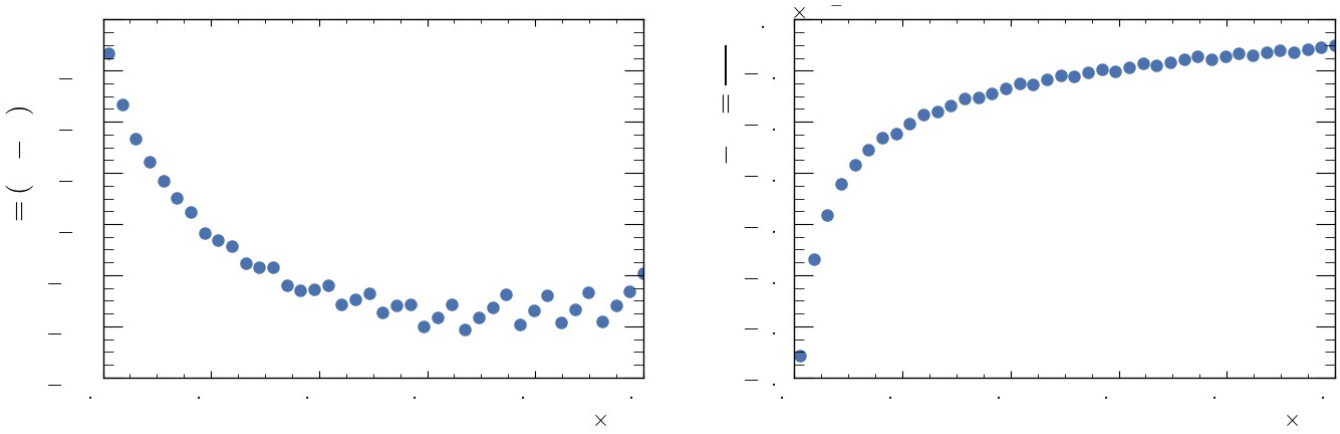


In [32]:
```
plt.scatter(x, (alpha * hists["sig_mc"] + hists["bkg_gen"]) / hists["data_mc"], label="prefit")
plt.scatter(x, (mu * hists["sig_mc"] + gamma * hists["bkg_gen"]) / hists["data_mc"], label="postfit")
plt.xticks(x[::3], x[::3])
plt.ylabel("(MC sig + ML bkg) $/$ MC data")
plt.xlabel("bins")
plt.axhline(1, c='k')
plt.legend()
plt.tight_layout()
# plt.savefig(saved + f"ratio_pre_post_{N_idx}.pdf")
```



In [33]:
```
fig, axs = plt.subplots(1, 2, figsize=(14, 5))
axs = axs.flatten()

q = unumpy.uarray(mus, mus_err)
```

```
spur = q * Ns - alpha * Ns
y = unumpy.nominal_values(spur)
yerr = unumpy.std_devs(spur)

axs[0].scatter(Ns, y)
# axs[0].errorbar(Ns, y, yerr, capsize=5)

spur_ratio = (q * Ns - alpha * Ns) / Ns
y_ratio = unumpy.nominal_values(spur_ratio)
yerr_ratio = unumpy.std_devs(spur_ratio)

axs[1].scatter(Ns, y_ratio)
# axs[1].errorbar(Ns, y_ratio, yerr_ratio, capsize=5)

axs[0].set_ylabel(r"$S_\text{spur}=(\mu-\alpha)\nu_B$")
axs[1].set_ylabel(r"$\mu - \alpha = \frac{S_\text{spur}}{\nu_B}$")

axs[0].set_xlabel(r"$\nu_B$", loc='center')
axs[1].set_xlabel(r"$\nu_B$", loc='center')

plt.tight_layout()
# plt.savefig(saved + f"spur_expected.pdf")
```



In [ ]:

In [ ]:

In [ ]: