μ Bias Note

Contents

1 Origin of FBMP's Bias

1.1 Model

$$
\begin{aligned}
\begin{bmatrix} \boldsymbol{w} \\ \boldsymbol{q}' \end{bmatrix} \Bigg| \boldsymbol{z} &\sim \mathrm{Normal}\left(\begin{bmatrix} \boldsymbol{V}_{\mathrm{PE}}\boldsymbol{z} \\ \boldsymbol{z} \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma} & \boldsymbol{V}_{\mathrm{PE}}\boldsymbol{Z} \\ \boldsymbol{Z}\boldsymbol{V}_{\mathrm{PE}}^{\intercal} & \boldsymbol{Z} \end{bmatrix} \right) \\
\boldsymbol{\Sigma} &= \boldsymbol{V}_{\mathrm{PE}}\boldsymbol{Z}\boldsymbol{V}_{\mathrm{PE}}^{\intercal} + \sigma_\epsilon^2 \boldsymbol{I}_M
\end{aligned} \tag{1}
$$

$$
\begin{aligned}
\nu &= \log[p(\boldsymbol{w}, \boldsymbol{z})] = \log[p(\boldsymbol{w}|\boldsymbol{z})p(\boldsymbol{z})] \\
&= -\frac{1}{2}(\boldsymbol{w} - \boldsymbol{V}_{\mathrm{PE}}\boldsymbol{z})^{\intercal}\boldsymbol{\Sigma}^{-1}(\boldsymbol{w} - \boldsymbol{V}_{\mathrm{PE}}\boldsymbol{z}) - \frac{1}{2}\log\det\boldsymbol{\Sigma} - \frac{N}{2}\log 2\pi \\
&\quad + \sum_i \log \mathrm{Poisson}(z_i, p_i)
\end{aligned} \tag{2}
$$

1.2 Approximation

- Poisson $\rightarrow$ Bernoulli, $z_i = 0, \boldsymbol{Z}_{ii} = 0$ or $z_i = 1, \boldsymbol{Z}_{ii} = \sigma^2$
- SPE approximation
  - For delta approximation, SPE $\rightarrow$ delta SPE, $\boldsymbol{V}_{PE} = \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{O} \end{bmatrix}$, $w_i \sim \mathrm{Normal}(1, \sigma^2 + \sigma_\epsilon^2)$
  - For flat approximation, SPE $\rightarrow$ flat SPE, $\boldsymbol{V}_{PE} = \frac{1}{\sqrt{M}}\boldsymbol{J}_{M,N}$, $w_i \sim \mathrm{Normal}(\frac{N}{\sqrt{M}}, \frac{N}{\sqrt{M}}\sigma^2 + \sigma_\epsilon^2)$, $\sum_i^M w_i \sim \mathrm{Normal}(N\sqrt{M}, M^{\frac{3}{2}}N\sigma^2 + M\sigma_\epsilon^2)$
- flat prior, $\begin{cases} \lambda & \text{if } z_i = 1 \\ 1 - \lambda & \text{if } z_i = 0 \end{cases}$

1.3 Evaluation

for a certain $(\boldsymbol{w}, \boldsymbol{z})$, let $K = \{i | z_i = 1\}$, $|K| = n$

1.3.1 Delta Approximation

$$
\boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{Z} & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{O} \end{bmatrix} + \sigma_\epsilon^2 \boldsymbol{I}_M \tag{3}
$$

$$
\det\boldsymbol{\Sigma} = \Pi_i(z_i\sigma^2 + \sigma_\epsilon^2) \tag{4}
$$

$$
\boldsymbol{\Sigma}_{ij}^{-1} = \begin{cases} \frac{1}{z_i\sigma^2 + \sigma_\epsilon^2} & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{5}
$$

Thus,

$$
\begin{aligned}
-2\nu = & \sum_{i \in K} \frac{(w_i - 1)^2}{\sigma^2 + \sigma_\epsilon^2} + \sum_{i \in \bar{K}} \frac{w_i^2}{\sigma_\epsilon^2} && (6) \\
& + n \log(\sigma^2 + \sigma_\epsilon^2) + (N - n) \log \sigma_\epsilon^2 && (7) \\
& - 2n \log \lambda - 2(N - n) \log(1 - \lambda) + \text{const} && (8)
\end{aligned}
$$

Additionally,

$$
\begin{aligned}
\sum_{i \in K} \frac{(w_i - 1)^2}{\sigma^2 + \sigma_\epsilon^2} & \sim \chi^2(n) && (9) \\
\frac{w_i^2}{\sigma_\epsilon^2} & = \frac{1}{\sigma_\epsilon^2}((\sigma^2 + \sigma_\epsilon^2)\frac{(w_i - 1)^2}{\sigma^2 + \sigma_\epsilon^2} + 2(w_i - 1) + 1) && (10)
\end{aligned}
$$

Calculate expectation on distribution of $\boldsymbol{w}$ and $\boldsymbol{z}$ with $|K| = |\boldsymbol{z}| = n$,

$$
\begin{aligned}
-2E(\nu)_n = & n + (N - n)\frac{\sigma^2 + \sigma_\epsilon^2}{\sigma_\epsilon^2} + (N - n)\frac{1}{\sigma_\epsilon^2} && (11) \\
& + n \log(\sigma^2 + \sigma_\epsilon^2) + (N - n) \log \sigma_\epsilon^2 && (12) \\
& - 2n \log \lambda - 2(N - n) \log(1 - \lambda) + \text{const} && (13)
\end{aligned}
$$

Thus,

$$
\begin{aligned}
-2D(\nu)_n = & - 2E(\nu)_n - (-2E(\nu)_{n-1}) && (14) \\
= & 1 - \frac{\sigma^2 + \sigma_\epsilon^2}{\sigma_\epsilon^2} - \frac{1}{\sigma_\epsilon^2} && (15) \\
& + \log(\sigma^2 + \sigma_\epsilon^2) - \log \sigma_\epsilon^2 && (16) \\
& - 2 \log \lambda + 2 \log(1 - \lambda) && (17)
\end{aligned}
$$

Remember that $\sigma^2 \gg \sigma_\epsilon^2$ and $\lambda \ll 1$, set $x = \sigma^2/\sigma_\epsilon^2$, then,

$$
D(\nu)_n \approx \frac{1}{2}x - \frac{1}{2} \log x + \log \lambda \tag{18}
$$

while $x \gg 1$, $\frac{1}{2}x - \frac{1}{2} \log x \gg 1$, the sign of $D(\nu)_n$ is defined by the relative magnitude of $\frac{1}{2}x - \frac{1}{2} \log x$ and $\log \lambda$.

1.3.2 Flat Approximation

$$\Sigma = \frac{1}{M}\sigma^2 \sum_i^N z_i \boldsymbol{J}_M + \sigma_\epsilon^2 \boldsymbol{I}_M \tag{19}$$

$$= \frac{\sigma^2 n}{M}\boldsymbol{J}_M + \sigma_\epsilon^2 \boldsymbol{I}_M \tag{20}$$

$$\det \Sigma = (\sigma_\epsilon^2 + \sigma^2 \sum_i^N z_i)\sigma_\epsilon^{2(M-1)} \tag{21}$$

$$= (\sigma_\epsilon^2 + \sigma^2 n)\sigma_\epsilon^{2(M-1)} \tag{22}$$

$$\Sigma^{-1} = \frac{1}{\sigma_\epsilon^2}(\boldsymbol{I}_M - \frac{1}{M}\frac{\sigma^2 \sum_i^N z_i}{\sigma^2 \sum_i^N z_i + \sigma_\epsilon^2}\boldsymbol{J}_M) \tag{23}$$

$$= \frac{1}{\sigma_\epsilon^2}(\boldsymbol{I}_M - \frac{1}{M}\frac{1}{1+\frac{\sigma_\epsilon^2}{\sigma^2 n}}\boldsymbol{J}_M) \tag{24}$$

set $M^* = M(1+\frac{\sigma_\epsilon^2}{\sigma^2 n})$, regarding $\sigma \gg \sigma_\epsilon$, thus $M^* \approx M$, and

$$\Sigma^{-1} = \frac{1}{\sigma_\epsilon^2}(\boldsymbol{I}_M - \frac{1}{M^*}\boldsymbol{J}_M) \tag{25}$$

Thus,

$$-2\nu = \frac{1}{\sigma_\epsilon^2}[\sum_i^M (w_i - \frac{n}{\sqrt{M}})^2 - \frac{1}{M^*}(\sum_i^M w_i - n\sqrt{M})^2] \tag{26}$$

$$+ \log(\sigma^2 n + \sigma_\epsilon^2) \tag{27}$$

$$- 2n \log\lambda - 2(N-n)\log(1-\lambda) + \text{const} \tag{28}$$

Additionally,

$$
\begin{aligned}
(w_i - \frac{n}{\sqrt{M}})^2 =& (w_i - \frac{N}{\sqrt{M}})^2 && (29)\\
&+ 2(w_i - \frac{N}{\sqrt{M}})\frac{N-n}{\sqrt{M}} + \frac{(N-n)^2}{M} && (30)\\
(\sum_i^M w_i - n\sqrt{M})^2 =& (\sum_i^M w_i - N\sqrt{M})^2 && (31)\\
&+ 2(\sum_i^M w_i - N\sqrt{M})(N-n)\sqrt{M} + (N-n)^2 M && (32)\\
\sum_i^M \frac{(w_i - \frac{N}{\sqrt{M}})^2}{\frac{N}{\sqrt{M}}\sigma^2 + \sigma_\epsilon^2} &\sim \chi^2(M) && (33)\\
\frac{(\sum_i^M w_i - N\sqrt{M})^2}{M^{\frac{3}{2}}N\sigma^2 + M\sigma_\epsilon^2} &\sim \chi^2(1) && (34)
\end{aligned}
$$

Calculate expectation on distribution of $\boldsymbol{w}$ and $\boldsymbol{z}$ with $|n| = |\boldsymbol{z}| = n$,

$$
\begin{aligned}
-2E(\nu)_n =& \frac{1}{\sigma_\epsilon^2}[(\frac{N}{\sqrt{M}}\sigma^2 + \sigma_\epsilon^2)M + (N-n)^2 && (35)\\
&- \frac{1}{M^*}(M^{\frac{3}{2}}N\sigma^2 + M\sigma_\epsilon^2 + (N-n)^2 M)] && (36)\\
&+ \log(\sigma^2 n + \sigma_\epsilon^2) && (37)\\
&- 2n\log\lambda - 2(N-n)\log(1-\lambda) + \text{const} && (38)\\
=& \frac{1}{\sigma_\epsilon^2}[(N\sqrt{M}\sigma^2 + (N-n)^2)(1 - \frac{M}{M^*}) + (M - \frac{M}{M^*})\sigma_\epsilon^2] && (39)\\
&+ \log(\frac{\sigma^2}{\sigma_\epsilon^2}n + 1) && (40)\\
&- 2n\log\lambda - 2(N-n)\log(1-\lambda) + \text{const} && (41)
\end{aligned}
$$

Thus,

$$
\begin{aligned}
E(\nu)'_n =& -\frac{1}{2}\{\frac{1}{\sigma_\epsilon^2}[2(n-N)(1-\frac{M}{M^*}) && (42)\\
&+ (N\sqrt{M}\sigma^2 + (N-n)^2 + \sigma_\epsilon^2)\frac{M}{M^{*2}}M^{*'}] && (43)\\
&+ \frac{1}{n + \frac{\sigma_\epsilon^2}{\sigma^2}}\} + \log\lambda - \log(1-\lambda) && (44)\\
M^{*'} =& -\frac{M\sigma_\epsilon^2}{\sigma^2 n^2} && (45)
\end{aligned}
$$

when $n = N$,

$$
\begin{aligned}
E(\nu)'_n|_N = & -\frac{1}{2}[\frac{1}{\sigma_\epsilon^2}(N\sqrt{M}\sigma^2 + \sigma_\epsilon^2)\frac{M}{M^{*2}}M^{*'} && (46) \\
& + \frac{1}{N + \frac{\sigma_\epsilon^2}{\sigma^2}}] + \log\lambda - \log(1-\lambda) && (47) \\
= & \frac{1}{2}\frac{N}{(N + \frac{\sigma_\epsilon^2}{\sigma^2})^2}(\sqrt{M} - 1) && (48) \\
& + \log\lambda - \log(1-\lambda) && (49)
\end{aligned}
$$

Remember that $M \gg 1$, then,

$$
C = \frac{1}{2}\frac{N}{(N + \frac{\sigma_\epsilon^2}{\sigma^2})^2}(\sqrt{M} - 1) > 0 \tag{50}
$$

the sign of $E(\nu)'_n|_N$ is defined by the relative magnitude of C and $\log\lambda$.

1.4 Discussion

When the time bin is very thin, $\log\lambda \ll 0$, $D(\nu)_n < 0$. When Bernoulli approximation fails, $D(\nu)_n > 0$.

While we may reasonably assume $E(\nu)_n$ will be its maximum when $n = N_{PE}$, considering the fact that RGS in FBMP has imperfect ergodicity, the bias will be positive given $D(\nu)_n > 0$ (or $E(\nu)'_n|_N > 0$) as RGS will **not** stop immediately when $n = N_{PE}$. Given $D(\nu)_n < 0$ (or $E(\nu)'_n|_N < 0$), the bias will be negative.

2 Phenomenon

2.1 P-FBMP with magic time bin

```
mu_t = abs(wave.sum() / gmu)
if Tau > 10:
    n = min(math.ceil(20 / mu_t), 5)
else:
    n = min(math.ceil(4 / mu_t), 3)
A, wave_r, tlist, t0_t, t0_delta, cha, left_wave, right_wave = wff.
    initial_params(wave[::wff.nshannon], spe_pre[ent[i]['ChannelID'
    ]], Mu, Tau, Sigma, gmu, Thres['lucyddM^\ast], p, nsp, nstd,
    is_t0=True, is_delta=False, n=n, nshannon=1)
mu_t = abs(wave_r.sum() / gmu)
def optit0mu(t0, mu, n, xmmse_star, psy_star, c_star, la):
    ys = np.log(psy_star) - np.log(poisson.pmf(c_star, la)).sum(
        axis=1)
    ys = np.exp(ys - ys.max()) / np.sum(np.exp(ys - ys.max()))
    t0list = np.arange(t0 - 3 * Sigma, t0 + 3 * Sigma + 1e-6, 0.2)
```

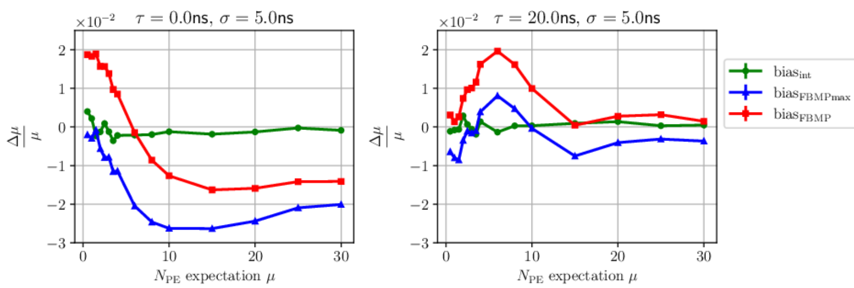
```
    mulist = np.arange(max(1e-8, mu - 3 * np.sqrt(mu)), mu + 3 * np.sqrt(mu), 0.1)
    b_mu = [max(1e-8, mu - 5 * np.sqrt(mu)), mu + 5 * np.sqrt(mu)]
    tlist_pan = np.sort(np.unique(np.hstack(np.arange(0, window)[:, None] + np.arange(0, 1, 1 / n))))
    As = np.zeros((len(xmmse_star), len(tlist_pan)))
    As[:, np.isin(tlist_pan, tlist)] = c_star
    assert sum(np.sum(As, axis=0) > 0) > 0

    def likelihood(x):
        a = x[0] * wff.convolve_exp_norm(tlist_pan - x[1], Tau, Sigma) / n + 1e-8 # use tlist_pan not tlist
        li = -special.logsumexp(np.log(poisson.pmf(As, a)).sum(axis=1), b=ys)
        return li

    likemu = np.array([likelihood([mulist[j], t0]) for j in range(len(mulist))])
    liket0 = np.array([likelihood([mu, t0list[j]]) for j in range(len(t0list))])
    mu, t0 = opti.fmin_l_bfgs_b(likelihood, x0=[mulist[likemu.argmin()], t0list[liket0.argmin()]], approx_grad=True, bounds=[b_mu, b_t0], maxfun=50000)[0]
    return mu, t0

truth = pelist[pelist['TriggerNo'] == ent[i]['TriggerNo']]
time_fbmp_start = time.time()
factor = np.sqrt(np.diag(np.matmul(A.T, A)).mean())
A = np.matmul(A, np.diag(1. / np.sqrt(np.diag(np.matmul(A.T, A)))))
la = mu_t * wff.convolve_exp_norm(tlist - t0_t, Tau, Sigma) / n + 1e-8
xmmse, xmmse_star, psy_star, nu_star, nu_star_bk, T_star, d_tot_i, d_max_i, num_i = wff.fbmpr_fxn_reduced(wave_r, A, la, spe_pre[cid]['std'] ** 2, (gsigma * factor / gmu) ** 2, factor, len(la), stop=5, truth=truth, i=i, left=left_wave, right=right_wave, tlist=tlist, gmu=gmu, para=p)
time_fbmp = time_fbmp + time.time() - time_fbmp_start
c_star = np.zeros_like(xmmse_star).astype(int)
for k in range(len(T_star)):
    t, c = np.unique(T_star[k][xmmse_star[k][T_star[k]] > 0], return_counts=True)
    c_star[k, t] = c
maxindex = 0

xmmse_most = xmmse_star[maxindex]
pet = np.repeat(tlist[xmmse_most > 0], c_star[maxindex][xmmse_most > 0])
cha = np.repeat(xmmse_most[xmmse_most > 0] / factor / c_star[maxindex][xmmse_most > 0], c_star[maxindex][xmmse_most > 0])

mu, t0 = optit0mu(t0_t, mu_t, n, xmmse_star, psy_star, c_star, la)
mu_i, t0_i = optit0mu(t0_t, mu_t, n, xmmse_most[None, :], np.array([1]), c_star[maxindex][None, :], la)
```

Motivation: demonstrate magic time binning on μ bias

Progress: magic time bin is

```python
mu_t = abs(wave.sum() / gmu)
if Tau > 10:
    n = min(math.ceil(20 / mu_t), 5)
else:
    n = min(math.ceil(4 / mu_t), 3)
```

Result: relative μ bias are less than 2.5%



2.2 P-FBMP only fitting

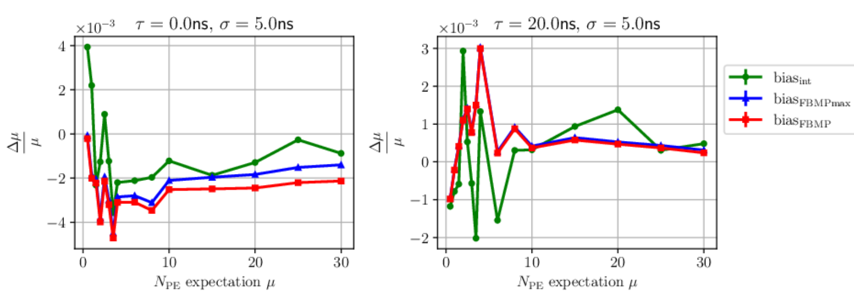
```python
def optit0mu(t0, mu, n, xmmse_star, psy_star, c_star, la):
    ys = np.log(psy_star) - np.log(poisson.pmf(c_star, la)).sum(axis=1)
    ys = np.exp(ys - ys.max()) / np.sum(np.exp(ys - ys.max()))
    t0list = np.arange(t0 - 3 * Sigma, t0 + 3 * Sigma + 1e-6, 0.2)
    mulist = np.arange(max(1e-8, mu - 3 * np.sqrt(mu)), mu + 3 * np.sqrt(mu), 0.1)
    b_mu = [max(1e-8, mu - 5 * np.sqrt(mu)), mu + 5 * np.sqrt(mu)]
    tlist_pan = np.sort(np.unique(np.concatenate([np.unique(np.hstack(np.arange(1, window-1)[:, None] + np.arange(0, 1, 1 / n))), tlist])))
    As = np.zeros((len(xmmse_star), len(tlist_pan)))
    As[:, np.isin(tlist_pan, tlist)] = c_star
    tlist_edge = np.concatenate([[tlist_pan[0] - np.diff(tlist_pan)[0] / 2], (tlist_pan[1:] + tlist_pan[:-1]) / 2, [tlist_pan[-1] + np.diff(tlist_pan)[-1] / 2]])
    diff = np.diff(tlist_edge)
    assert As.sum() == c_star.sum()
    assert sum(np.sum(As, axis=0) > 0) > 0

    def likelihood(x):
        a = x[0] * wff.convolve_exp_norm(tlist_pan - x[1], Tau, Sigma) * diff + 1e-8 # use tlist_pan not tlist
        li = -special.logsumexp(np.log(poisson.pmf(As, a)).sum(axis=1), b=ys)
        return li

    likemu = np.array([likelihood([mulist[j], t0]) for j in range(len(mulist))])
```

```
    liket0 = np.array([likelihood([mu, t0list[j]]) for j in range(len(t0list))])
    mu, t0 = opti.fmin_l_bfgs_b(likelihood, x0=[mulist[likemu.argmin()], t0list[liket0.argmin()]], approx_grad=True, bounds=[b_mu, b_t0], maxfun=50000)[0]
    return mu, t0

truth = pelist[pelist['TriggerNo'] == ent[i]['TriggerNo']]

tlist = truth['HitPosInWindow'][truth['HitPosInWindow'] < right_wave - 1]
if len(tlist) == 1:
    tlist_edge = np.array([tlist[0] - 0.5, tlist[0] + 0.5])
else:
    tlist_edge = np.concatenate([[tlist[0] - np.diff(tlist)[0] / 2], (tlist[1:] + tlist[:-1]) / 2, [tlist[-1] + np.diff(tlist)[-1] / 2]])
t_auto = (np.arange(left_wave, right_wave) / wff.nshannon)[:, None] - tlist
A = p[2] * np.exp(-1 / 2 * (np.log((t_auto + np.abs(t_auto)) / p[0] / 2) / p[1]) ** 2)

t0_t = t0_truth[i]['T0']
mu_t = len(truth)
la = mu_t * np.array([integrate.quad(lambda t : wff.convolve_exp_norm(t - t0_t, Tau, Sigma), tlist_edge[i], tlist_edge[i+1])[0] for i in range(len(tlist))]) + 1e-8

c_star_truth = np.ones_like(tlist)
wav_ans = np.sum([np.where(pp > truth['HitPosInWindow'][j], wff.spe(pp - truth['HitPosInWindow'][j], tau=p[0], sigma=p[1], A=p[2]) * truth['Charge'][j] / gmu, 0) for j in range(len(truth))], axis=0)

mu, t0 = optit0mu(t0_t, mu_t, n, np.empty(len(la))[None, :], np.array([1]), c_star_truth[None, :], la)
```

Motivation: to validate fitting process

Progress: Use true `HitPosInWindow` as `tlist`, true `T0` as `t0_t`, length of `HitPosInWindow` as `mu_t`. Without FBMP RGS, directly **fit** μ and t_0.

Result: fitting process is not related to bias of μ.



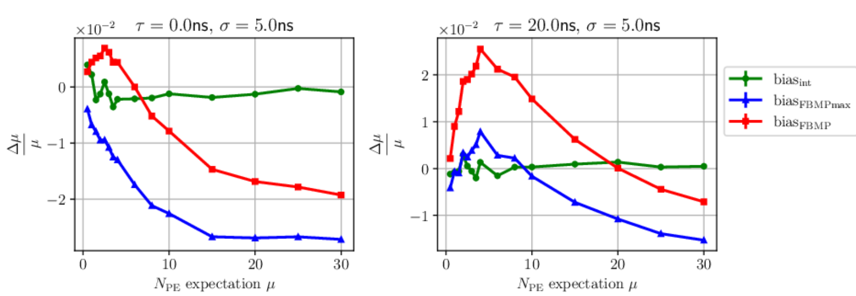
2.3 P-FBMP with truth prior

```
tlist = truth['HitPosInWindow'][truth['HitPosInWindow'] <
    right_wave - 1]
if len(tlist) == 1:
    tlist_edge = np.array([tlist[0] - 0.5, tlist[0] + 0.5])
else:
    tlist_edge = np.concatenate([[tlist[0] - np.diff(tlist)[0] /
        2], (tlist[1:] + tlist[:-1]) / 2, [tlist[-1] + np.diff(
        tlist)[-1] / 2]])
t_auto = (np.arange(left_wave, right_wave) / wff.nshannon)[:, None]
     - tlist
A = p[2] * np.exp(-1 / 2 * (np.log((t_auto + np.abs(t_auto)) / p[0]
     / 2) / p[1]) ** 2)
t0_t = t0_truth[i]['T0']
mu_t = len(truth)
factor = np.sqrt(np.diag(np.matmul(A.T, A)))
A = np.matmul(A, np.diag(1. / np.sqrt(np.diag(np.matmul(A.T, A)))))
la = mu_t * np.array([integrate.quad(lambda t : wff.
    convolve_exp_norm(t - t0_t, Tau, Sigma), tlist_edge[i],
    tlist_edge[i+1])[0] for i in range(len(tlist))]) + 1e-8
xmmse, xmmse_star, psy_star, nu_star, nu_star_bk, T_star, d_tot_i,
    d_max_i, num_i = wff.fbmpr_fxn_reduced(wave_r, A, la, spe_pre[
    cid]['std'] ** 2, (gsigma * factor / gmu) ** 2, factor, len(la)
    , stop=5, truth=truth, i=i, left=left_wave, right=right_wave,
    tlist=tlist, gmu=gmu, para=p)
c_star = np.zeros_like(xmmse_star).astype(int)
for k in range(len(T_star)):
    t, c = np.unique(T_star[k][xmmse_star[k][T_star[k]] > 0],
        return_counts=True)
    c_star[k, t] = c
mu = np.average(c_star.sum(axis=1), weights=psy_star)
t0 = t0_t
```

Motivation: to prove difference between artificial prior and truth prior (derived from simulation) used in FBMP is not the origin of μ bias.

Progress: Use true `HitPosInWindow` as `tlist`, true `T0` as `t0_t`, length of `HitPosInWindow` as `mu_t`. `la` is vector of integral in each time bins.

Result: prior used in FBMP is not the origin of μ bias.

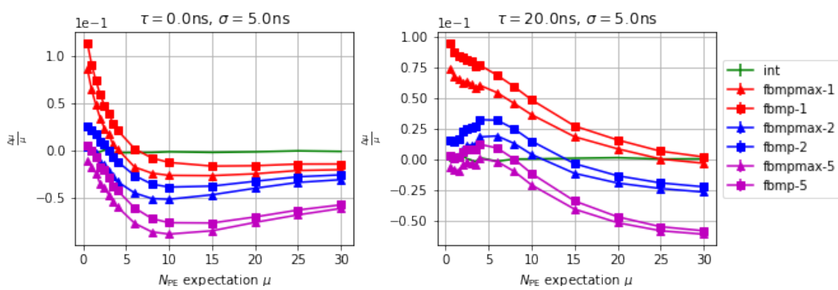


2.4 P-FBMP using different bin width

```python
mu_t = abs(wave.sum() / gmu)
A, wave_r, tlist, t0_t, t0_delta, cha, left_wave, right_wave = wff.initial_params(wave[::wff.nshannon], spe_pre[ent[i]['ChannelID']], Tau, Sigma, gmu, Thres['lucyddM^\ast], p, nsp, nstd, is_t0=True, is_delta=False, n=n, nshannon=1)
mu_t = abs(wave_r.sum() / gmu)
def optit0mu(t0, mu, n, xmmse_star, psy_star, c_star, la):
    ys = np.log(psy_star) - np.log(poisson.pmf(c_star, la)).sum(axis=1)
    ys = np.exp(ys - ys.max()) / np.sum(np.exp(ys - ys.max()))
    t0list = np.arange(t0 - 3 * Sigma, t0 + 3 * Sigma + 1e-6, 0.2)
    mulist = np.arange(max(1e-8, mu - 3 * np.sqrt(mu)), mu + 3 * np.sqrt(mu), 0.1)
    b_mu = [max(1e-8, mu - 5 * np.sqrt(mu)), mu + 5 * np.sqrt(mu)]
    tlist_pan = np.sort(np.unique(np.hstack(np.arange(0, window)[:, None] + np.arange(0, 1, 1 / n))))
    As = np.zeros((len(xmmse_star), len(tlist_pan)))
    As[:, np.isin(tlist_pan, tlist)] = c_star
    assert sum(np.sum(As, axis=0) > 0) > 0

    def likelihood(x):
        a = x[0] * wff.convolve_exp_norm(tlist_pan - x[1], Tau, Sigma) / n + 1e-8 # use tlist_pan not tlist
        li = -special.logsumexp(np.log(poisson.pmf(As, a)).sum(axis=1), b=ys)
        return li

    likemu = np.array([likelihood([mulist[j], t0]) for j in range(len(mulist))])
    liket0 = np.array([likelihood([mu, t0list[j]]) for j in range(len(t0list))])
    mu, t0 = opti.fmin_l_bfgs_b(likelihood, x0=[mulist[likemu.argmin()], t0list[liket0.argmin()]], approx_grad=True, bounds=[b_mu, b_t0], maxfun=50000)[0]
    return mu, t0

truth = pelist[pelist['TriggerNo'] == ent[i]['TriggerNo']]

factor = np.sqrt(np.diag(np.matmul(A.T, A)))
A = np.matmul(A, np.diag(1. / np.sqrt(np.diag(np.matmul(A.T, A)))))
la = mu_t * wff.convolve_exp_norm(tlist - t0_t, Tau, Sigma) / n + 1e-8
xmmse, xmmse_star, psy_star, nu_star, nu_star_bk, T_star, d_tot_i, d_max_i, num_i = wff.fbmpr_fxn_reduced(wave_r, A, la, spe_pre[cid]['std'] ** 2, (gsigma * factor / gmu) ** 2, factor, len(la), stop=5, truth=truth, i=i, left=left_wave, right=right_wave, tlist=tlist, gmu=gmu, para=p)
time_fbmp = time_fbmp + time.time() - time_fbmp_start
c_star = np.zeros_like(xmmse_star).astype(int)
for k in range(len(T_star)):
    t, c = np.unique(T_star[k][xmmse_star[k][T_star[k]] > 0], return_counts=True)
    c_star[k, t] = c
maxindex = 0

```

```
xmmse_most = xmmse_star[maxindex]
pet = np.repeat(tlist[xmmse_most > 0], c_star[maxindex][xmmse_most
    > 0])
cha = np.repeat(xmmse_most[xmmse_most > 0] / factor[xmmse_most > 0]
     / c_star[maxindex][xmmse_most > 0], c_star[maxindex][
    xmmse_most > 0])

mu, t0 = optit0mu(t0_t, mu_t, n, xmmse_star, psy_star, c_star, la)
mu_i, t0_i = optit0mu(t0_t, mu_t, n, xmmse_most[None, :], np.array
    ([1]), c_star[maxindex][None, :], la)
```

Motivation: to demonstrate the bin width's influence on μ bias.
Progress: $1/n$ is the bin width, $n = \{1, 2, 5\}$



2.5 N_{PE} estimation in one bin

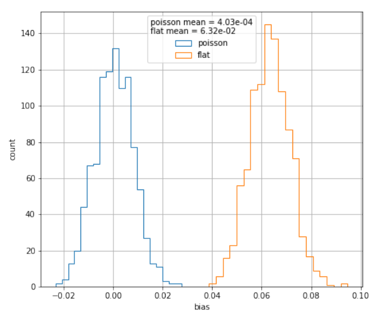
```
def getm(i, p):
    N = 10000
    bias = np.empty(N)
    np.random.seed(i)
    Mu = 10
    gmu = 160
    gsigma = 40
    npan = np.arange(1, 50)
    for i in range(N):
        n = poisson.ppf(1 - uniform.rvs(scale=1-poisson.cdf(0, Mu),
             size=1), Mu).astype(int)
        cha = norm.ppf(1 - uniform.rvs(scale=1-norm.cdf(0, loc=gmu,
             scale=gsigma), size=n), loc=gmu, scale=gsigma)
        cha = cha.sum()
        if p:
            weight = poisson.pmf(npan, Mu) * norm.pdf(cha, loc=gmu
                 * npan, scale=gsigma * np.sqrt(npan))
        else:
            weight = norm.pdf(cha, loc=gmu * npan, scale=gsigma *
                 np.sqrt(npan))
        weight = weight / weight.sum()
        bias[i] = np.sum(npan * weight) - n
    m = bias.mean()
    return m
slices = np.arange(1000).T.tolist()
```

```
with Pool(100) as pool:
    resultpoisson = np.hstack([pool.starmap(getm, zip(slices, [True
        ] * len(slices)))])
with Pool(100) as pool:
    resultflat = np.hstack([pool.starmap(getm, zip(slices, [False]
        * len(slices)))])
```

Motivation: to estimate bias of μ based on toy simulation in one bin

Progress: set μ. Sampling: $n \sim \mathrm{Poisson}(n|\mu)$, $q_i \sim \mathrm{Gaussian}(q_i|1, \sigma_q^2), i = \{1, 2, \cdots, n\}$, $Q = \sum_i q_i$. Estimation: $p(n|Q) = \frac{\mathrm{Poisson}(n|\mu)\cdot\mathrm{Gaussian}(Q|n, n\sigma_q^2)}{\sum_{j=1}^{n}\mathrm{Poisson}(j|\mu)\cdot\mathrm{Gaussian}(Q|j, j\sigma_q^2)}$ or $p(n|Q) = \frac{\mathrm{Gaussian}(Q|n, n\sigma_q^2)}{\sum_{j=1}^{n}\mathrm{Gaussian}(Q|j, j\sigma_q^2)}$

Result: when using Poisson prior, there is **no** bias, when using flat prior, there is bias.



2.6 N_{PE} estimation based on toy simulation

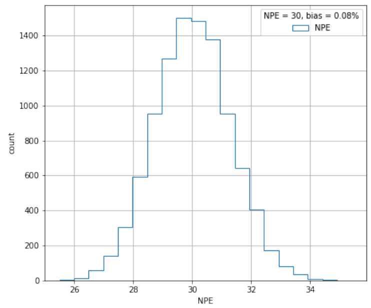
```
n = 10000
npe = 30
noi = True
if noi:
    std = 1.
else:
    std = 1e-4
window = 200
np.random.seed(0)
wdtp = np.dtype([('No', np.uint32), ('WaveforM^\ast, np.float,
    window)])
waves = np.empty(n).astype(wdtp)
tlist = np.repeat(np.array([0, 1, 1]), npe / 3)
sams = [np.vstack((tlist, wff.charge(npe, gmu=gmu, gsigma=gsigma,
    thres=0))).T for i in range(n)]
pan = np.arange(0, window)
t0 = 0.5
for i in tqdm(range(n)):
    wave = np.sum([np.where(pan > sams[i][j, 0], wff.spe(pan - sams
        [i][j, 0], tau=p[0], sigma=p[1], A=p[2]) * sams[i][j, 1] /
        gmu, 0) for j in range(len(sams[i]))], axis=0)
```

```
    if noi:
        wave = wave + np.random.normal(0, std, size=window)
    waves[i]['WaveforM^\ast] = wave
waves['No'] = np.arange(n).astype(np.uint32)
sdtp = np.dtype([('No', np.uint32), ('HitPosInWindow', np.float64),
    ('Charge', np.float64)])
pelist = np.empty(sum([len(sams[i]) for i in range(n)])).astype(
    sdtp)
pelist['No'] = np.repeat(np.arange(n), [len(sams[i]) for i in range
    (n)]).astype(np.uint32)
pelist['HitPosInWindow'] = np.hstack([sams[i][:, 0] for i in range(
    n)])
pelist['Charge'] = np.hstack([sams[i][:, 1] for i in range(n)])
mu = np.empty(n)
for i in tqdm(range(n)):
    wave = waves[i]['WaveforM^\ast].astype(np.float64) * spe_pre
        [0]['epulse']
    truth = pelist[pelist['No'] == waves[i]['No']]
    tlist = np.unique(truth['HitPosInWindow'])

    t_auto = np.arange(0, window)[:, None] - tlist
    A = p[2] * np.exp(-1 / 2 * (np.log((t_auto + np.abs(t_auto)) /
        p[0] / 2) / p[1]) ** 2)
    mu_t = npe
    factor = np.sqrt(np.diag(np.matmul(A.T, A)))
    A = np.matmul(A, np.diag(1. / np.sqrt(np.diag(np.matmul(A.T, A)
        ))))
    la = mu_t * np.array([1, 2]) / 3

    xmmse, xmmse_star, psy_star, nu_star, nu_star_bk, T_star,
        d_tot_i, d_max_i, num_i = wff.fbmpr_fxn_reduced(wave, A, la
        , std ** 2, (gsigma * factor / gmu) ** 2, factor, len(la),
        stop=5)

    c_star = np.zeros_like(xmmse_star).astype(int)
    for k in range(len(T_star)):
        t, c = np.unique(T_star[k][xmmse_star[k][T_star[k]] > 0],
            return_counts=True)
        c_star[k, t] = c

    mu[i] = np.average(c_star.sum(axis=1), weights=psy_star)
```

Motivation: try to understand the relation between convolution process and bias

Progress: there is no μ, set N_{PE} as 30, set *HitPosInWindow* to $[0, 1]$, set time profile prior as $[1/3, 2/3]$, sample *Charge*. Use FBMP to generate posterior distribution of N_{PE}.

Result: there is **no** bias on this configuration



2.7 N_{PE} ergodic estimation based on toy simulation

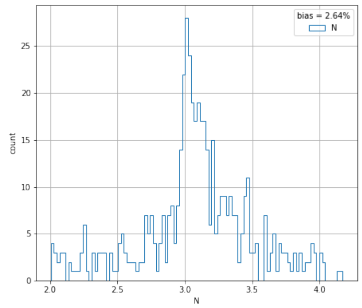
```
spemode = 'spe'
# spemode = 'delta'
# spemode = 'flat'
prior = False
plot = False

n = 500
gmu = 160.
gsigma = gmu / 4
window = 200
npe = 3
assert window > npe

def spe(t, mode='spe'):
    if mode == 'spe':
        return wff.spe((t + np.abs(t)) / 2, p[0], p[1], p[2])
    elif mode == 'delta':
        return np.where(t == 0, gmu, 0)
    elif mode == 'flat':
        return np.ones_like(t) / window * gmu

noi = True
if noi:
    std = 1
else:
    std = 1e-4
np.random.seed(0)
wdtp = np.dtype([('No', np.uint32), ('Waveform', np.float, window)
    ])
waves = np.empty(n).astype(wdtp)

tlist = np.arange(npe)

p = spe_pre[0]['parameters']
sams = [np.vstack((tlist, wff.charge(npe, gmu=gmu, gsigma=gsigma,
    thres=0))).T for i in range(n)]
pan = np.arange(window)
for i in tqdm(range(n)):
#     wave = np.sum([np.where(pan >= sams[i][j, 0], spe(pan - sams[
    i][j, 0], mode=spemode) * sams[i][j, 1] / gmu, 0) for j in
```

```
    range(len(sams[i]))], axis=0)
    wave = np.sum([spe(pan - sams[i][j, 0], mode=spemode) * sams[i
        ][j, 1] / gmu for j in range(len(sams[i]))], axis=0)
    if noi:
        wave = wave + np.random.normal(0, std, size=window)
    waves[i]['Waveform'] = wave
waves['No'] = np.arange(n).astype(np.uint32)
sdtp = np.dtype([('No', np.uint32), ('HitPosInWindow', np.float64),
     ('Charge', np.float64)])
pelist = np.empty(sum([len(sams[i]) for i in range(n)])).astype(
    sdtp)
pelist['No'] = np.repeat(np.arange(n), [len(sams[i]) for i in range
    (n)]).astype(np.uint32)
pelist['HitPosInWindow'] = np.hstack([sams[i][:, 0] for i in range(
    n)])
pelist['Charge'] = np.hstack([sams[i][:, 1] for i in range(n)])

# npe = 3
aver_npe = np.empty(n)
for i in tqdm(range(n)):
    wave = waves[i]['Waveform'].astype(np.float64)[:window] *
        spe_pre[0]['epulse']
    truth = pelist[pelist['No'] == waves[i]['No']]
    tlist = np.unique(truth['HitPosInWindow'])

    t_auto = np.arange(window)[:, None] - tlist
    A = spe(t_auto, spemode)
    mu_t = npe
    factor = np.sqrt(np.diag(np.matmul(A.T, A)))
    A = A / factor
    la = mu_t * np.ones_like(tlist) / len(tlist)

    samnum = int(10 ** 3)
    ind = np.array([[i // 100, i // 10 % 10, i % 10] for i in range
        (samnum)])
    nu = np.empty(samnum)
    for j in range(samnum):
        nu[j] = wff.nu_direct(wave, A, ind[j], factor, (gsigma *
            factor / gmu) ** 2, std ** 2, la, prior=prior)
        nu[j] = nu[j] + poisson.logpmf(ind[j], mu=la).sum()
    prob = np.exp(nu - nu.max()) / np.sum(np.exp(nu - nu.max()))
    aver_npe[i] = np.average(ind.sum(axis=1), weights=prob)
```

Motivation: see the estimation of N_{PE} when ergodic in model space

Progress: there is no μ, set N_{PE} as 3, set *tlist* to $[0, 1, 2]$. Use ergodic process to generate posterior distribution of N_{PE}.

Result: there is **still** bias on this configuration



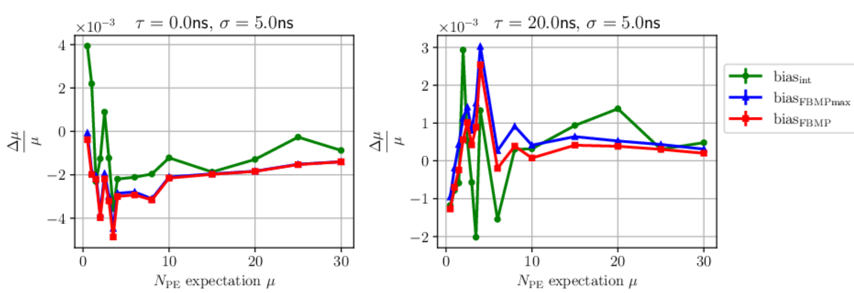
2.8 Influence of lucyddm `tlist` initialization on μ bias

```
n = 2
A, wave_r, tlist, t0_t, t0_delta, cha, left_wave, right_wave = wff.
    initial_params(wave[::wff.nshannon], spe_pre[ent[i]['ChannelID'
    ]], Tau, Sigma, gmu, Thres['lucyddM^\ast], p, nsp, nstd, is_t0=
    True, is_delta=False, n=n, nshannon=1)

truth = pelist[pelist['TriggerNo'] == ent[i]['TriggerNo']]

c_star_truth = np.sum([np.where(tlist - 0.5 / n < truth['
    HitPosInWindow'][j], 1, 0) * np.where(tlist + 0.5 / n > truth['
    HitPosInWindow'][j], 1, 0) for j in range(len(truth))], axis=0)

mu = c_star_truth.sum()
t0 = t0_t
```

Motivation: to test the influence of lucyddm time bin initialization on μ bias

Progress: lucyddm initialization, estimate μ as number of truth *HitPosInWindow* in `tlist`

Result: there is **no** bias from lucyddm `tlist` initialization



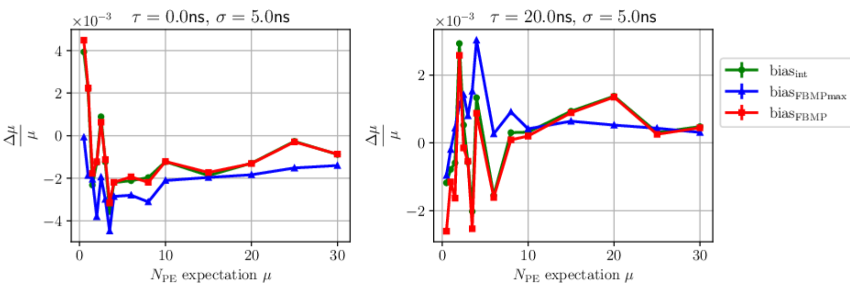
2.9 Influence of lucyddm `wave_r` initialization on μ bias

```
mu_t = abs(wave.sum() / gmu)
n = 2
A, wave_r, tlist, t0_t, t0_delta, cha, left_wave, right_wave = wff.initial_params(wave[::wff.nshannon], spe_pre[ent[i]['ChannelID']], Tau, Sigma, gmu, Thres['lucyddM^\ast], p, nsp, nstd, is_t0=True, is_delta=False, n=n, nshannon=1)
mu_t = abs(wave_r.sum() / gmu)
mu = mu_t
t0 = t0_t
```

Motivation: to test the influence of waveform cut in lucyddm initialization on μ bias

Progress: lucyddm initialization, estimate μ as number of integral of truncated waveform `wave_r`

Result: there is **no** bias from lucyddm waveform cut initialization



2.10 Iterative P-FBMP

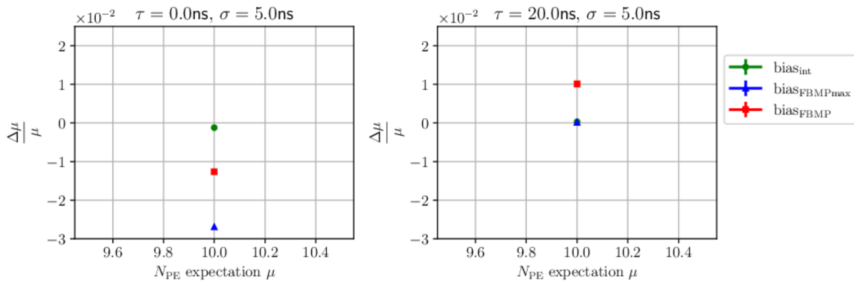
```
mu_t = abs(wave.sum() / gmu)
if Tau > 10:
    n = min(math.ceil(20 / mu_t), 5)
else:
    n = min(math.ceil(4 / mu_t), 3)
A, wave_r, tlist, t0_t, t0_delta, cha, left_wave, right_wave = wff.initial_params(wave[::wff.nshannon], spe_pre[ent[i]['ChannelID']], Tau, Sigma, gmu, Thres['lucyddM^\ast], p, nsp, nstd, is_t0=True, is_delta=False, n=n, nshannon=1)
mu_t = abs(wave_r.sum() / gmu)
def optit0mu(t0, mu, n, xmmse_star, psy_star, c_star, la):
    ys = np.log(psy_star) - np.log(poisson.pmf(c_star, la)).sum(axis=1)
    ys = np.exp(ys - ys.max()) / np.sum(np.exp(ys - ys.max()))
    t0list = np.arange(t0 - 3 * Sigma, t0 + 3 * Sigma + 1e-6, 0.2)
    mulist = np.arange(max(1e-8, mu - 3 * np.sqrt(mu)), mu + 3 * np.sqrt(mu), 0.1)
    b_mu = [max(1e-8, mu - 5 * np.sqrt(mu)), mu + 5 * np.sqrt(mu)]
    tlist_pan = np.sort(np.unique(np.hstack(np.arange(0, window)[:, None] + np.arange(0, 1, 1 / n))))
    As = np.zeros((len(xmmse_star), len(tlist_pan)))
    As[:, np.isin(tlist_pan, tlist)] = c_star
```

```
    assert sum(np.sum(As, axis=0) > 0) > 0

    def likelihood(x):
        a = x[0] * wff.convolve_exp_norm(tlist_pan - x[1], Tau, Sigma) / n + 1e-8 # use tlist_pan not tlist
        li = -special.logsumexp(np.log(poisson.pmf(As, a)).sum(axis=1), b=ys)
        return li

    likemu = np.array([likelihood([mulist[j], t0]) for j in range(len(mulist))])
    liket0 = np.array([likelihood([mu, t0list[j]]) for j in range(len(t0list))])
    mu, t0 = opti.fmin_l_bfgs_b(likelihood, x0=[mulist[likemu.argmin()], t0list[liket0.argmin()]], approx_grad=True, bounds=[b_mu, b_t0], maxfun=50000)[0]
    return mu, t0

truth = pelist[pelist['TriggerNo'] == ent[i]['TriggerNo']]
time_fbmp_start = time.time()
factor = np.sqrt(np.diag(np.matmul(A.T, A)).mean())
A = np.matmul(A, np.diag(1. / np.sqrt(np.diag(np.matmul(A.T, A)))))

t0_bk = np.inf
mu_bk = np.inf
iterc_i = 0
while abs(mu_bk - mu_t) > 1e-3 or abs(t0_bk - t0_t) > 1e-3:
    iterc_i += 1
    t0_bk = t0_t
    mu_bk = mu_t
    la = mu_t * wff.convolve_exp_norm(tlist - t0_t, Tau, Sigma) / n + 1e-8
    xmmse, xmmse_star, psy_star, nu_star, nu_star_bk, T_star, d_tot_i, d_max_i, num_i = wff.fbmpr_fxn_reduced(wave_r, A, la, spe_pre[cid]['std'] ** 2, (gsigma * factor / gmu) ** 2, factor, len(la), stop=5, truth=truth, i=i, left=left_wave, right=right_wave, tlist=tlist, gmu=gmu, para=p)
    time_fbmp = time_fbmp + time.time() - time_fbmp_start
    c_star = np.zeros_like(xmmse_star).astype(int)
    for k in range(len(T_star)):
        t, c = np.unique(T_star[k][xmmse_star[k][T_star[k]] > 0], return_counts=True)
        c_star[k, t] = c

    mu_t, t0_t = optit0mu(t0_t, mu_t, n, xmmse_star, psy_star, c_star, la)
    if iterc_i >= 10:
        break

mu = mu_t
t0 = t0_t
mu_i, t0_i = optit0mu(t0_t, mu_t, n, xmmse_most[None, :], np.array([1]), c_star[maxindex][None, :], la)
```

Motivation: demonstrate whether iteration can reduce μ bias

Progress: set $\mu = 10$ to save time, iteration after FBMP sampling and fitting until μ and t_0 are stable

Result: iteration can **not** reduce μ bias



2.11 Waveform simulation with fixed *HitPosInWindow*

Waveform simulation:

```
np.random.seed(a0 + round(Tau + Sigma))
npe = poisson.ppf(1 - uniform.rvs(scale=1-poisson.cdf(0, mu), size=
    a1 - a0), mu).astype(int)
t0 = np.random.uniform(100., 500., size=a1 - a0)
sams = [np.vstack((np.arange(npe[i]) + t0[i], wff.charge(npe[i],
    gmu=gmu, gsigma=gsigma, thres=0))).T for i in range(a1 - a0)]
wdtp = np.dtype([('TriggerNo', np.uint32), ('ChannelID', np.uint32)
    , ('WaveforM^\ast, np.float, window * wff.nshannon)])
waves = np.empty(a1 - a0).astype(wdtp)
pan = np.arange(0, window, 1 / wff.nshannon)
for i in range(a1 - a0):
    wave = np.sum([np.where(pan > sams[i][j, 0], wff.spe(pan - sams
        [i][j, 0], tau=p[0], sigma=p[1], A=p[2]) * sams[i][j, 1] /
        gmu, 0) for j in range(len(sams[i]))], axis=0)
    if args.noi:
        wave = wave + np.random.normal(0, std, size=window * wff.
            nshannon)
    waves[i]['WaveforM^\ast] = wave
tdtp = np.dtype([('TriggerNo', np.uint32), ('ChannelID', np.uint32)
    , ('T0', np.float64)])
t = np.empty(a1 - a0).astype(tdtp)
t['TriggerNo'] = np.arange(a0, a1).astype(np.uint32)
t['T0'] = t0
t['ChannelID'] = 0
waves['TriggerNo'] = np.arange(a0, a1).astype(np.uint32)
waves['ChannelID'] = 0
sdtp = np.dtype([('TriggerNo', np.uint32), ('PMTId', np.uint32), ('
    HitPosInWindow', np.float64), ('Charge', np.float64)])
pelist = np.empty(sum([len(sams[i]) for i in range(a1 - a0)])).
    astype(sdtp)
pelist['TriggerNo'] = np.repeat(np.arange(a0, a1), [len(sams[i])
    for i in range(a1 - a0)]).astype(np.uint32)
pelist['PMTId'] = 0
pelist['HitPosInWindow'] = np.hstack([sams[i][:, 0] for i in range(
    a1 - a0)])
```

```
pelist['Charge'] = np.hstack([sams[i][:, 1] for i in range(a1 - a0)
    ])
```

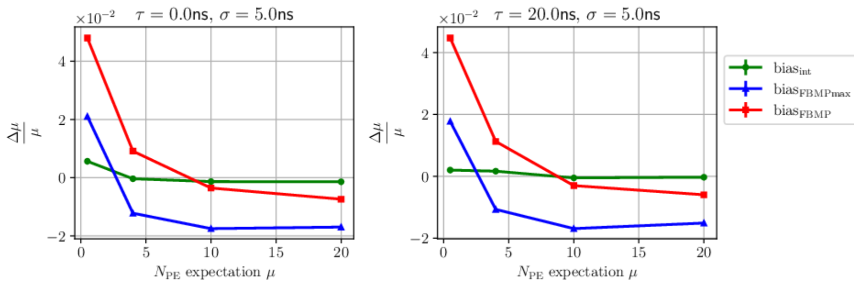
P-FBMP:

```
mu_t = abs(wave.sum() / gmu)
if Tau > 10:
    n = min(math.ceil(20 / mu_t), 5)
else:
    n = min(math.ceil(4 / mu_t), 3)
A, wave_r, tlist, t0_t, t0_delta, cha, left_wave, right_wave = wff.
    initial_params(wave[::wff.nshannon], spe_pre[ent[i]['ChannelID'
    ]], Tau, Sigma, gmu, Thres['lucyddM^\ast], p, nsp, nstd, is_t0=
    True, is_delta=False, n=n, nshannon=1)

truth = pelist[pelist['TriggerNo'] == ent[i]['TriggerNo']]

tlist = truth['HitPosInWindow'][truth['HitPosInWindow'] <
    right_wave - 1]
if len(tlist) == 0:
    tlist = np.array([300])
t_auto = (np.arange(left_wave, right_wave) / wff.nshannon)[:, None]
     - tlist
A = p[2] * np.exp(-1 / 2 * (np.log((t_auto + np.abs(t_auto)) / p[0]
     / 2) / p[1]) ** 2)

t0_t = t0_truth[i]['T0']
mu_t = len(truth)
factor = np.sqrt(np.diag(np.matmul(A.T, A)))
A = np.matmul(A, np.diag(1. / np.sqrt(np.diag(np.matmul(A.T, A)))))
la = mu_t * np.ones(len(tlist)) / len(tlist)
xmmse, xmmse_star, psy_star, nu_star, nu_star_bk, T_star, d_tot_i,
    d_max_i, num_i = wff.fbmpr_fxn_reduced(wave_r, A, la, spe_pre[
    cid]['std'] ** 2, (gsigma * factor / gmu) ** 2, factor, len(la)
    , stop=5, truth=truth, i=i, left=left_wave, right=right_wave,
    tlist=tlist, gmu=gmu, para=p)
time_fbmp = time_fbmp + time.time() - time_fbmp_start
c_star = np.zeros_like(xmmse_star).astype(int)
for k in range(len(T_star)):
    t, c = np.unique(T_star[k][xmmse_star[k][T_star[k]] > 0],
        return_counts=True)
    c_star[k, t] = c
maxindex = 0

xmmse_most = xmmse_star[maxindex]
pet = np.repeat(tlist[xmmse_most > 0], c_star[maxindex][xmmse_most
    > 0])
cha = np.repeat(xmmse_most[xmmse_most > 0] / factor[xmmse_most > 0]
     / c_star[maxindex][xmmse_most > 0], c_star[maxindex][
    xmmse_most > 0])

mu = np.average(c_star.sum(axis=1), weights=psy_star)
t0 = t0_t
mu_i = len(cha)
t0_i = t0_t
```

Motivation: to further reduce the influence of binning

Progress: when PE number is N_{PE}, set *HitPosInWindow* relatively (to t_0) $\{0, 1, \cdots, N_{PE} - 1\}$, sample *Charge* then P-FBMP process with truth prior.

Result: there is still bias with fixed *HitPosInWindow*



2.12 Waveform simulation with fixed *HitPosInWindow*, without prior

Waveform simulation:

```
sams = [np.vstack((np.arange(npe[i]) + t0[i], wff.charge(npe[i],
    gmu=gmu, gsigma=gsigma, thres=0))).T for i in range(a1 - a0)]
```

P-FBMP:

```
def fbmpr_fxn_reduced(y, A, p1, sig2w, sig2s, mus, D, stop=0, truth
    =None, i=None, left=None, right=None, tlist=None, gmu=None,
    para=None, prior=True):
    '''
    p1: prior probability for each bin.
    sig2w: variance of white noise.
    sig2s: variance of signal x_i.
    mus: mean of signal x_i.
    '''
    # Only for multi-gaussian with arithmetic sequence of mu and
        sigma
    M, N = A.shape

    p = 1 - poisson.pmf(0, p1).mean()
    # Eq. (25)
    if prior:
        nu_true_mean = -M / 2 - M / 2 * np.log(sig2w) - p * N / 2 *
            np.log(sig2s / sig2w + 1) - M / 2 * np.log(2 * np.pi)
          + N * np.log(1 - p) + p * N * np.log(p / (1 - p))
        nu_true_stdv = np.sqrt(M / 2 + N * p * (1 - p) * (np.log(p
          / (1 - p)) - np.log(sig2s / sig2w + 1) / 2) ** 2)
    else:
```

```python
        nu_true_mean = -M / 2 - M / 2 * np.log(sig2w) - p * N / 2 * np.log(sig2s / sig2w + 1) - M / 2 * np.log(2 * np.pi)
        nu_true_stdv = np.sqrt(M / 2 + N * p * (1 - p) * (np.log(sig2s / sig2w + 1) / 2) ** 2)
    nu_stop = (nu_true_mean + stop * nu_true_stdv).max()

    psy_thresh = 1e-4
    # upper limit of number of PEs.
    P = max(math.ceil(min(M, p1.sum() + 3 * np.sqrt(p1.sum()))), 1)

    T = np.full((P, D), 0)
    nu = np.full((P, D), -np.inf)
    xmmse = np.zeros((P, D, N))
    cc = np.zeros((P, D, N))
    d_tot = D

    # nu_root: nu for all s_n=0.
    if prior:
        nu_root = -0.5 * np.linalg.norm(y) ** 2 / sig2w - 0.5 * M * np.log(2 * np.pi) - 0.5 * M * np.log(sig2w) + np.log(poisson.pmf(0, p1)).sum()
    else:
        nu_root = -0.5 * np.linalg.norm(y) ** 2 / sig2w - 0.5 * M * np.log(2 * np.pi) - 0.5 * M * np.log(sig2w)
    # Eq. (29)
    cx_root = A / sig2w
    # Eq. (30) sig2s = 1 sigma^2 - 0 sigma^2
    betaxt_root = sig2s / (1 + sig2s * np.einsum('ij,ij->j', A, cx_root, optimize=True))
    # Eq. (31)
    if prior:
        nuxt_root = nu_root + 0.5 * (betaxt_root * (y @ cx_root + mus / sig2s) ** 2  - mus ** 2 / sig2s + np.log(betaxt_root / sig2s)) + np.log(poisson.pmf(1, p1) / poisson.pmf(0, p1))
    else:
        nuxt_root = nu_root + 0.5 * (betaxt_root * (y @ cx_root + mus / sig2s) ** 2  - mus ** 2 / sig2s + np.log(betaxt_root / sig2s))
    pan_root = np.zeros(N)

    # Repeated Greedy Search
    for d in range(D):
        nuxt = nuxt_root.copy()
        z = y.copy()
        cx = cx_root.copy()
        betaxt = betaxt_root.copy()
        pan = pan_root.copy()
        for p in range(P):
            # look for duplicates of nu and nuxt, set to -inf.
            # only inspect the same number of PEs in Row p.
            nuxtshadow = np.where(np.sum(np.abs(nuxt - nu[p:(p+1), :d].T) < 1e-4, axis=0), -np.inf, nuxt)
            nustar = max(nuxtshadow)
            istar = np.argmax(nuxtshadow)
            nu[p, d] = nustar
            T[p, d] = istar
```

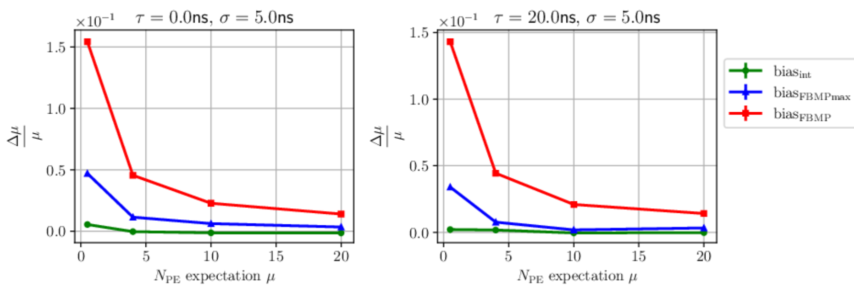
```
            pan[istar] += 1
            # Eq. (33)
            cx -= np.einsum('n,m,mp->np', betaxt[istar] * cx[:, istar], cx[:, istar], A, optimize=True)

            # Eq. (34)
            z -= A[:, istar] * mus[istar]
            assist = np.zeros(N)
            t, c = np.unique(T[:p+1, d], return_counts=True)
            assist[t] = mus[t] * c + sig2s[t] * c * np.dot(z, cx[:, t])
            cc[p, d][t] = c
            xmmse[p, d] = assist

            # Eq. (30)
            betaxt = sig2s / (1 + sig2s * np.sum(A * cx, axis=0))
            # Eq. (31)
            if prior:
                nuxt = nustar + 0.5 * (betaxt * (z @ cx + mus / sig2s) ** 2 - mus ** 2 / sig2s + np.log(betaxt / sig2s)) + np.log(poisson.pmf(pan + 1, mu=p1) / poisson.pmf(pan, mu=p1))
            else:
                nuxt = nustar + 0.5 * (betaxt * (z @ cx + mus / sig2s) ** 2 - mus ** 2 / sig2s + np.log(betaxt / sig2s))
            # nuxt[t] = -np.inf

        if max(nu[:, d]) > nu_stop:
            d_tot = d + 1
            break
    nu_bk = nu[:, :d_tot].copy()
    nu = nu[:, :d_tot].T.flatten()

    indx = np.argsort(nu)[::-1]
    d_max = math.floor(indx[0] // P) + 1
    num = min(math.ceil(np.sum(nu > nu.max() + np.log(psy_thresh))), d_tot * P)
    nu_star = nu[indx[:num]]
    nu_star_bk = nu_bk.T.flatten()[indx[:num]]
    # psy_star = np.exp(nu_star - nu.max()) / np.sum(np.exp(nu_star - nu.max()))
    T_star = [np.sort(T[:(indx[k] % P) + 1, indx[k] // P]) for k in range(num)]
    xmmse_star = np.empty((num, N))
    for k in range(num):
        xmmse_star[k] = xmmse[indx[k] % P, indx[k] // P]

    psy_star = np.exp(nu_star - nu.max()) / np.sum(np.exp(nu_star - nu.max()))

    xmmse = np.average(xmmse_star, weights=psy_star, axis=0)

    return xmmse, xmmse_star, psy_star, nu_star, nu_star_bk, T_star, d_tot, d_max, num
```

```python
prior = False
truth = pelist[pelist['TriggerNo'] == ent[i]['TriggerNo']]

tlist = truth['HitPosInWindow'][truth['HitPosInWindow'] <
    right_wave - 1]
if len(tlist) == 0:
    tlist = np.array([300])
t_auto = (np.arange(left_wave, right_wave) / wff.nshannon)[:, None]
    - tlist
A = p[2] * np.exp(-1 / 2 * (np.log((t_auto + np.abs(t_auto)) / p[0]
    / 2) / p[1]) ** 2)

t0_t = t0_truth[i]['T0']
mu_t = len(truth)
factor = np.sqrt(np.diag(np.matmul(A.T, A)))
A = A / factor
la = mu_t * np.ones(len(tlist)) / len(tlist)
xmmse, xmmse_star, psy_star, nu_star, nu_star_bk, T_star, d_tot_i,
    d_max_i, num_i = wff.fbmpr_fxn_reduced(wave_r, A, la, spe_pre[
    cid]['std'] ** 2, (gsigma * factor / gmu) ** 2, factor, len(la)
    , stop=5, truth=truth, i=i, left=left_wave, right=right_wave,
    tlist=tlist, gmu=gmu, para=p, prior=prior)

mu = np.average(c_star.sum(axis=1), weights=psy_star)
t0 = t0_t
mu_i = len(cha)
t0_i = t0_t
```

Motivation: to further reduce the influence of prior

Progress: set *HitPosInWindow* relatively (to t_0) $\{0, 1, \cdots, N_{PE} - 1\}$, P-FBMP process **without** truth prior.

Result: all biases **are larger** than 0



2.13 P-FBMP without prior and with Gaussian normalization term

```python
prior = False
space = True
n = 2
```

```python
tlist_pan = np.sort(np.unique(np.hstack(np.arange(0, window)[:, None] + np.linspace(0, 1, n, endpoint=False) - (n // 2) / n)))
b_t0 = [0., 600.]

def likelihood(mu, t0, As_k, nu_star_k):
    a = wff.convolve_exp_norm(tlist_pan - t0, Tau, Sigma) / n + 1e-8 # use tlist_pan not tlist
    # a *= mu / a.sum()
    a *= mu
    li = -special.logsumexp(np.log(poisson.pmf(As_k, mu=a)).sum(axis=1) + nu_star_k)
    return li

def sum_mu_likelihood(mu, t0_list, As_list, nu_star_list):
    return np.sum([likelihood(mu, t0_list[k], As_list[k], nu_star_list[k]) for k in range(len(t0_list))])

def optit0mu(mu, t0, nu_star, As, mu_init=None):
    l = len(t0)
    mulist = np.arange(max(1e-8, mu - 2 * np.sqrt(mu)), mu + 2 * np.sqrt(mu), 1e-1)
    b_mu = [max(1e-8, mu - 5 * np.sqrt(mu)), mu + 5 * np.sqrt(mu)]
    # psy_star = [np.exp(nu_star[k] - nu_star[k].max()) / np.sum(np.exp(nu_star[k] - nu_star[k].max())) for k in range(l)]
    t0list = [np.arange(t0[k] - 3 * Sigma, t0[k] + 3 * Sigma + 1e-6, 0.2) for k in range(l)]
    sigmamu = None
    logLv_mu = None
    if mu_init is None:
        mu_init = np.empty(l)
        for k in range(l):
            mu_init[k] = mulist[np.array([likelihood(mulist[j], t0[k], As[k], nu_star[k]) for j in range(len(mulist))]).argmin()]
            t0_init = t0list[k][np.array([likelihood(mu_init[k], t0list[k][j], As[k], nu_star[k]) for j in range(len(t0list[k]))]).argmin()]
            likelihood_x = lambda x, As, nu_star: likelihood(x[0], x[1], As, nu_star)
            t0[k] = opti.fmin_l_bfgs_b(likelihood_x, args=(As[k], nu_star[k]), x0=[mu_init[k], t0_init], approx_grad=True, bounds=[b_mu, b_t0], maxfun=50000)[0][1]
        Likelihood = lambda mu: np.sum([likelihood(mu, t0[k], As[k], nu_star[k]) for k in range(l)])
        mu, fval, _ = opti.fmin_l_bfgs_b(Likelihood, x0=[np.mean(mu_init)], approx_grad=True, bounds=[b_mu], maxfun=50000)
    else:
        def Likelihood(mu, t0_list, As_list, nu_star_list):
            with Pool(min(args.Ncpu // 3, cpu_count())) as pool:
                result = np.sum(pool.starmap(likelihood, zip([mu] * l, t0_list, As_list, nu_star_list)))
            return result
        mu, fval, _ = opti.fmin_l_bfgs_b(Likelihood, args=[t0, As, nu_star], x0=[np.mean(mu_init)], approx_grad=True, bounds=[b_mu], maxfun=50000, factr=100.0, pgtol=1e-10)
        print('Mu fitting info is', _)
```

```python
        sigmamu_est = np.sqrt(mu / N)
        mulist = np.sort(np.append(np.arange(max(1e-8, mu - 2 * sigmamu_est), mu + 2 * sigmamu_est, sigmamu_est / 50), mu))
        partial_sum_mu_likelihood = partial(sum_mu_likelihood, t0_list=t0, As_list=As, nu_star_list=nu_star)
        with Pool(min(args.Ncpu // 3, cpu_count())) as pool:
            logLv_mu = np.array(pool.starmap(partial_sum_mu_likelihood, zip(mulist)))

        mu_func = interp1d(mulist, logLv_mu, bounds_error=False, fill_value='extrapolate')
        logLvdelta = np.vectorize(lambda mu_t: np.abs(mu_func(mu_t) - fval - 0.5))
        # sigmamu = abs(opti.fmin_l_bfgs_b(logLvdelta, x0=[mulist[np.abs(logLv_mu - fval - 0.5).argmin()]], approx_grad=True, bounds=[[mulist[0], mulist[-1]]], maxfun=500000)[0] - mu) * np.sqrt(l)
        sigmamu_l = abs(opti.fmin_l_bfgs_b(logLvdelta, x0=[mulist[mulist <= mu][np.abs(logLv_mu[mulist <= mu] - fval - 0.5).argmin()]], approx_grad=True, bounds=[[mulist[0], mulist[-1]]], maxfun=500000)[0] - mu) * np.sqrt(l)
        sigmamu_r = abs(opti.fmin_l_bfgs_b(logLvdelta, x0=[mulist[mulist > mu][np.abs(logLv_mu[mulist > mu] - fval - 0.5).argmin()]], approx_grad=True, bounds=[[mulist[0], mulist[-1]]], maxfun=500000)[0] - mu) * np.sqrt(l)
        sigmamu = (sigmamu_l + sigmamu_r) / 2
        print('Finite difference sigmamu is {:.4}'.format(sigmamu.item()))

        # derivative_mu2 = nd.Derivative(Likelihood, step=1e-10, n=2, full_output=True)
        # s, info = derivative_mu2(mu, t0_list=t0, As_list=As, nu_star_list=nu_star)
        # print('Mu derivative info is', info)
        # sigmamu = 1 / np.sqrt(s) * np.sqrt(l)
    return mu, t0, [sigmamu, mulist, logLv_mu, fval]

time_fbmp_start = time.time()
cid = ent[i]['ChannelID']
assert cid == 0
wave = ent[i]['Waveform'].astype(np.float64) * spe_pre[cid]['epulse']

# initialization
mu_t = abs(wave.sum() / gmu)
A, y, tlist, t0_t, t0_delta, cha, left_wave, right_wave = wff.initial_params(wave[::wff.nshannon], spe_pre[ent[i]['ChannelID']], Tau, Sigma, gmu, Thres['lucyddm'], p, is_t0=True, is_delta=False, n=n, nshannon=1)
mu_t = abs(y.sum() / gmu)

truth = pelist[pelist['TriggerNo'] == ent[i]['TriggerNo']]

# Eq. (9) where the columns of A are taken to be unit-norm.
mus = np.sqrt(np.diag(np.matmul(A.T, A)))
A = A / mus
```

```
p1 = mu_t * wff.convolve_exp_norm(tlist - t0_t, Tau, Sigma) / n + 1e-8
# p1 = cha / cha.sum() * mu_t + 1e-8
p1 = p1 / p1.sum() * mu_t
sig2w = spe_pre[cid]['std'] ** 2
sig2s = (gsigma * mus / gmu) ** 2
xmmse_star, nu_star, T_star, c_star, d_max_i, num_i = wff.fbmpr_fxn_reduced(y, A, sig2w, sig2s, mus, len(p1), p1=p1, truth=truth, i=i, left=left_wave, right=right_wave, tlist=tlist, gmu=gmu, para=p, prior=prior, space=space)
time_fbmp[i - a0] = time.time() - time_fbmp_start

la_truth = Mu * wff.convolve_exp_norm(tlist - t0_truth[i]['T0'], Tau, Sigma) / n + 1e-8
nu_space_prior = np.array([wff.nu_direct(y, A, c_star[j], mus, sig2s, sig2w, la_truth, prior=True, space=True) for j in range(num_i)])

maxindex = nu_star.argmax()
xmmse_most = np.clip(xmmse_star[maxindex], 0, np.inf)
pet = np.repeat(tlist[xmmse_most > 0], c_star[maxindex][xmmse_most > 0])
cha = np.repeat(xmmse_most[xmmse_most > 0] / mus[xmmse_most > 0] / c_star[maxindex][xmmse_most > 0], c_star[maxindex][xmmse_most > 0])

As = np.zeros((num_i, len(tlist_pan)))
As[:, np.isin(tlist_pan, tlist)] = c_star
assert sum(np.sum(As, axis=0) > 0) > 0

spacefactor = 0
priorfactor = 0
if not space:
    spacefactor = np.array([-0.5 * np.log(np.linalg.det(wff.Phi(y, A, c_star[j], mus, sig2s, sig2w, p1))) for j in range(num_i)])
if prior:
    priorfactor = -poisson.logpmf(c_star, p1).sum(axis=1)
nu_star = nu_star + priorfactor + spacefactor
# nu_star = np.log(np.exp(nu_star - nu_star.max()) / np.sum(np.exp(nu_star - nu_star.max())))
mu, t0, _ = optit0mu(mu_t, [t0_t], [nu_star], [As])
mu_i, t0_i, _ = optit0mu(mu_t, [t0_t], [np.array([0.])], [As[maxindex][None, :]])
```

```
def fbmpr_fxn_reduced(y, A, sig2w, sig2s, mus, D, p1, truth=None, i=None, left=None, right=None, tlist=None, gmu=None, para=None, prior=False, space=True, plot=False):
    '''
    p1: prior probability for each bin.
    sig2w: variance of white noise.
    sig2s: variance of signal x_i.
    mus: mean of signal x_i.
    '''
    # Only for multi-gaussian with arithmetic sequence of mu and
```

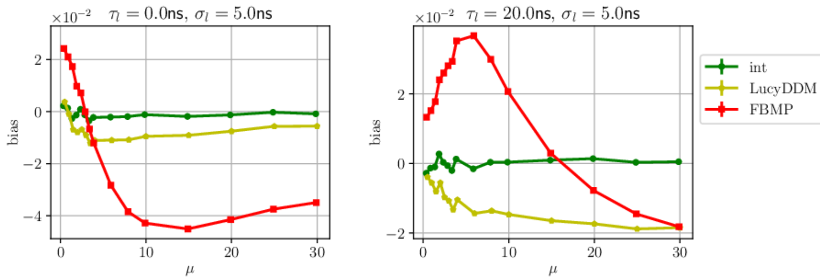
```
        sigma
    M, N = A.shape

    psy_thresh = 1e-1
    # upper limit of number of PEs.
    P = max(math.ceil(min(M, p1.sum() + 3 * np.sqrt(p1.sum()))), 1)

    T = np.full((P, D), 0)
    nu = np.full((P, D), -np.inf)
    xmmse = np.zeros((P, D, N))
    cc = np.zeros((P, D, N))

    # nu_root: nu for all s_n=0.
    # no Gaussian space factor
    nu_root = -0.5 * np.linalg.norm(y) ** 2 / sig2w - 0.5 * M * np.log(2 * np.pi)
    if space:
        nu_root -= 0.5 * M * np.log(sig2w)
    if prior:
        nu_root += poisson.logpmf(0, p1).sum()
    # Eq. (29)
    cx_root = A / sig2w
    # Eq. (30) sig2s = 1 sigma^2 - 0 sigma^2
    betaxt_root = sig2s / (1 + sig2s * np.einsum('ij,ij->j', A, cx_root, optimize=True))
    # Eq. (31)
    # no Gaussian space factor
    nuxt_root = nu_root + 0.5 * (betaxt_root * (y @ cx_root + mus / sig2s) ** 2 - mus ** 2 / sig2s)
    if space:
        nuxt_root += 0.5 * np.log(betaxt_root / sig2s)
    if prior:
        nuxt_root += poisson.logpmf(1, p1) - poisson.logpmf(0, p1)
    pan_root = np.zeros(N)

    # Repeated Greedy Search
    for d in range(D):
        nuxt = nuxt_root.copy()
        z = y.copy()
        cx = cx_root.copy()
        betaxt = betaxt_root.copy()
        pan = pan_root.copy()
        for p in range(P):
            # look for duplicates of nu and nuxt, set to -inf.
            # only inspect the same number of PEs in Row p.
            nuxtshadow = np.where(np.sum(np.abs(nuxt - nu[p:(p+1), :d].T) < 1e-4, axis=0), -np.inf, nuxt)
            nustar = max(nuxtshadow)
            istar = np.argmax(nuxtshadow)
            nu[p, d] = nustar
            T[p, d] = istar
            pan[istar] += 1
            # Eq. (33)
            cx -= np.einsum('n,m,mp->np', betaxt[istar] * cx[:, istar], cx[:, istar], A, optimize=True)

            # Eq. (34)
```

```
            z -= A[:, istar] * mus[istar]
            assist = np.zeros(N)
            t, c = np.unique(T[:p+1, d], return_counts=True)
            assist[t] = mus[t] * c + sig2s[t] * c * np.dot(z, cx[:, t])
            cc[p, d][t] = c
            xmmse[p, d] = assist

            # Eq. (30)
            betaxt = sig2s / (1 + sig2s * np.sum(A * cx, axis=0))
            # Eq. (31)
            # no Gaussian space factor
            nuxt = nustar + 0.5 * (betaxt * (z @ cx + mus / sig2s) ** 2 - mus ** 2 / sig2s)
            if space:
                nuxt += 0.5 * np.log(betaxt / sig2s)
            if prior:
                nuxt += poisson.logpmf(pan + 1, mu=p1) - poisson.logpmf(pan, mu=p1)
            nuxt[np.isnan(nuxt)] = -np.inf
            # nuxt[t] = -np.inf
    nu = nu.T.flatten()

    indx = np.argsort(nu)[::-1]
    d_max = math.floor(indx[0] // P) + 1
    num = min(math.ceil(np.sum(nu > nu.max() + np.log(psy_thresh))), D * P)
    nu_star = nu[indx[:num]]
    psy_star = np.exp(nu_star - nu_star.max()) / np.sum(np.exp(nu_star - nu_star.max()))
    T_star = [np.sort(T[:(indx[k] % P) + 1, indx[k] // P]) for k in range(num)]
    xmmse_star = np.empty((num, N))
    for k in range(num):
        xmmse_star[k] = xmmse[indx[k] % P, indx[k] // P]

    c_star = np.zeros_like(xmmse_star, dtype=int)
    for k in range(num):
        t, c = np.unique(T_star[k], return_counts=True)
        c_star[k, t] = c
```

Motivation: to reduce the influence of prior

Progress: *HitPosInWindow* is sampled from time profile, P-FBMP process **without** prior.

Result: μ bias persists.



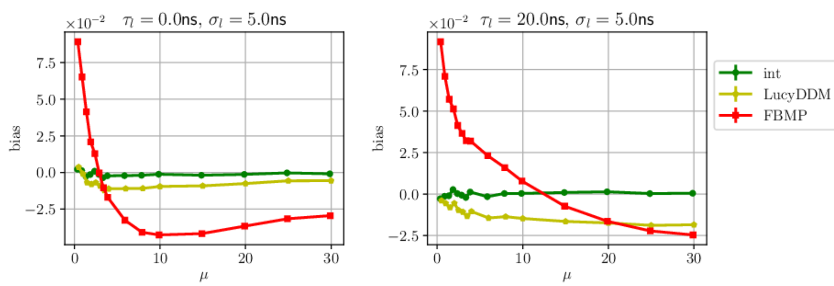
2.14 P-FBMP with lucy prior and with Gaussian

```
prior = True
space = True
la = cha / cha.sum() * mu_t + 1e-8
```

Motivation: to show the relation between lucy prior and μ bias.

Progress: Use `cha` (derived from lucyddm) as `la` and the bin width is fixed to 1/2.

Result: bias still exists when using lucy prior.



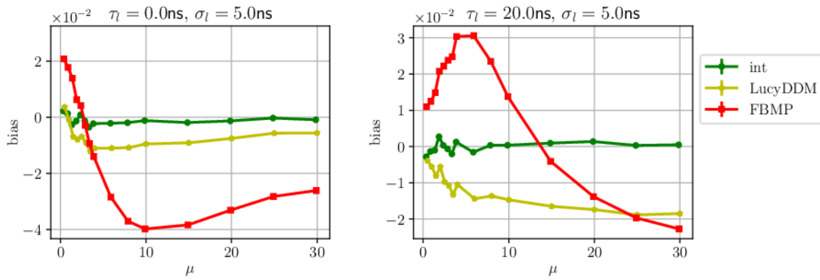
2.15 P-FBMP with profile prior and with Gaussian

```
prior = True
space = True
la = mu_t * wff.convolve_exp_norm(tlist - t0_t, Tau, Sigma) / n + 1
    e-8
```

Motivation: to show the relation between profile prior and μ bias.

Progress: Use time profile as `la` and the bin width is fixed to 1/2.

Result: bias still exists when using profile prior.



2.16 P-FBMP without prior and without Gaussian normalization term

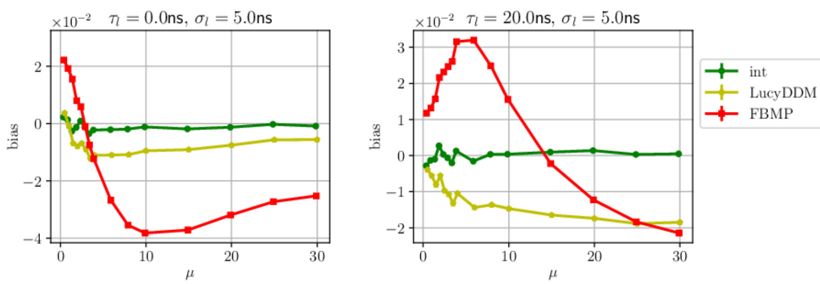
P-FBMP:

```
prior = False
space = False
```

Motivation: to show the μ bias when Gaussian normalization term $-\frac{1}{2}\log\det\mathbf{\Sigma}$ in model selecting metric ν is removed, additionally, without prior.

Progress: not calculate $-\frac{1}{2}\log\det\mathbf{\Sigma}$ in ν when RGS.

Result: μ bias persists.



2.17 P-FBMP with lucy prior and without Gaussian normalization term

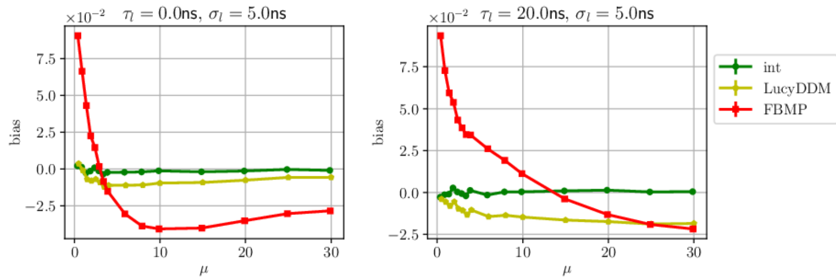
P-FBMP:

```
prior = True
space = False
la = cha / cha.sum() * mu_t + 1e-8
```

Motivation: to show the μ bias withour Gaussian normalization term and with lucy prior.

Progress: not calculate $-\frac{1}{2}\log\det\mathbf{\Sigma}$ in ν when RGS.

Result: μ bias persists.



2.18 P-FBMP with profile prior and without Gaussian normalization term

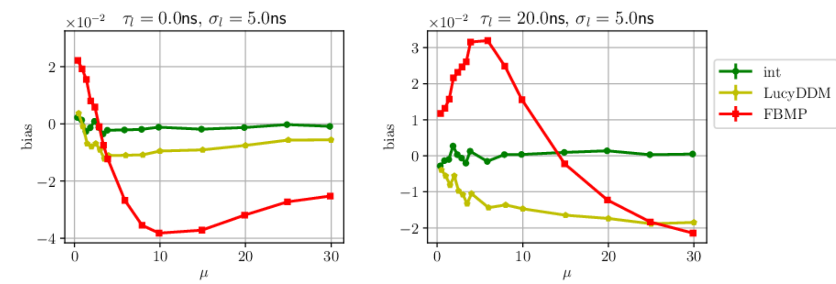
P-FBMP:

```
prior = True
space = False
la = mu_t * wff.convolve_exp_norm(tlist - t0_t, Tau, Sigma) / n + 1
    e-8
```

Motivation: to show the μ bias withour Gaussian normalization term and with profile prior.

Progress: not calculate $-\frac{1}{2}\log\det\mathbf{\Sigma}$ in ν when RGS.

Result: μ bias persists.



2.19 P-FBMP μ estimation with α correction

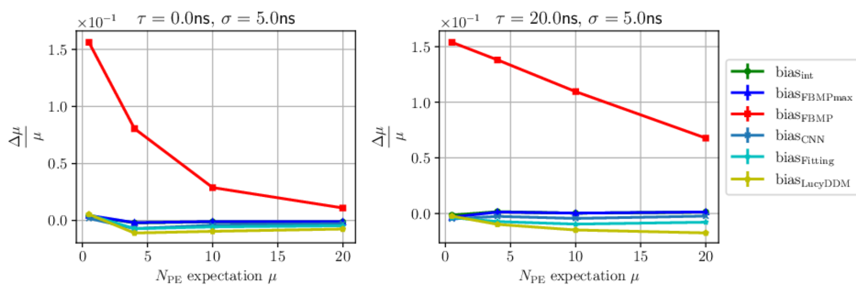
```
prior = False
nsp = 4
nstd = 3
mu_t = abs(wave.sum() / gmu)
n = 5
A, wave_r, tlist, t0_t, t0_delta, cha, left_wave, right_wave = wff.
    initial_params(wave[::wff.nshannon], spe_pre[ent[i]['ChannelID'
    ]], Tau, Sigma, gmu, Thres['lucyddm'], p, nsp, nstd, is_t0=True
    , is_delta=False, n=n, nshannon=1)
```

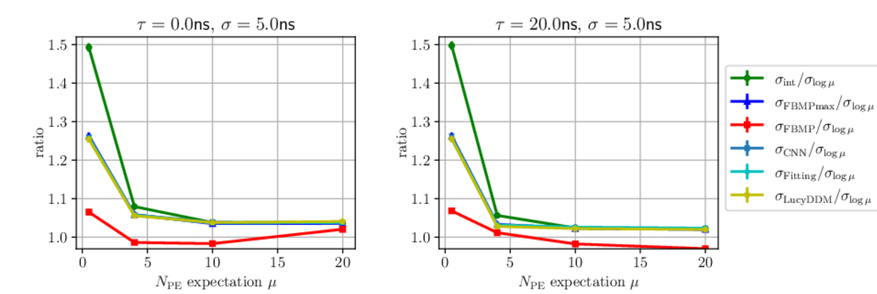
```python
mu_t = abs(wave_r.sum() / gmu)
def optit0mu(t0, mu, n, psy_star, c_star, la):
    ys = np.log(psy_star)
    if prior:
        ys = ys - poisson.logpmf(c_star, la).sum(axis=1)
    ys = np.exp(ys - ys.max()) / np.sum(np.exp(ys - ys.max()))
    t0list = np.arange(t0 - 3 * Sigma, t0 + 3 * Sigma + 1e-6, 0.2)
    mulist = np.arange(max(1e-8, mu - 3 * np.sqrt(mu)), mu + 3 * np.sqrt(mu), 0.1)
    b_mu = [max(1e-8, mu - 5 * np.sqrt(mu)), mu + 5 * np.sqrt(mu)]
    tlist_pan = np.sort(np.unique(np.hstack(np.arange(0, window)[:, None] + np.linspace(0, 1, n, endpoint=False) - (n // 2) / n)))
    As = np.zeros((len(psy_star), len(tlist_pan)))
    As[:, np.isin(tlist_pan, tlist)] = c_star
    assert sum(np.sum(As, axis=0) > 0) > 0
    def likelihood(x):
        a = wff.convolve_exp_norm(tlist_pan - x[1], Tau, Sigma) / n + 1e-8 # use tlist_pan not tlist
        a = a / a.sum() * x[0]
        li = -special.logsumexp(np.log(poisson.pmf(As, a)).sum(axis=1), b=ys)
        return li
    likemu = np.array([likelihood([mulist[j], t0]) for j in range(len(mulist))])
    liket0 = np.array([likelihood([mu, t0list[j]]) for j in range(len(t0list))])
    mu, t0 = opti.fmin_l_bfgs_b(likelihood, x0=[mulist[likemu.argmin()], t0list[liket0.argmin()]], approx_grad=True, bounds=[b_mu, b_t0], maxfun=50000)[0]
    return mu, t0
# 1st FBMP
time_fbmp_start = time.time()
# Eq. (9) where the columns of A are taken to be unit-norm.
factor = np.sqrt(np.diag(np.matmul(A.T, A)))
A = A / factor
# la = mu_t * wff.convolve_exp_norm(tlist - t0_t, Tau, Sigma) / n + 1e-8
la = cha / cha.sum() * mu_t + 1e-8
la = la / la.sum() * mu_t
xmmse, xmmse_star, psy_star, nu_star, nu_star_bk, T_star, d_max_i, num_i = wff.fbmpr_fxn_reduced(wave_r, A, spe_pre[cid]['std'] ** 2, (gsigma * factor / gmu) ** 2, factor, len(la), p1=la, stop=5, truth=truth, i=i, left=left_wave, right=right_wave, tlist=tlist, gmu=gmu, para=p, prior=prior)
time_fbmp = time_fbmp + time.time() - time_fbmp_start
c_star = np.zeros_like(xmmse_star).astype(int)
for k in range(len(T_star)):
    t, c = np.unique(T_star[k], return_counts=True)
    c_star[k, t] = c
la_truth = len(truth) * wff.convolve_exp_norm(tlist - t0_truth[i]['T0'], Tau, Sigma) / n + 1e-8
if prior:
    nu_star_prior = nu_star - poisson.logpmf(c_star, mu=la).sum(axis=1)
nu_star_prior = nu_star + poisson.logpmf(c_star, mu=la_truth).sum(axis=1)
```

```python
maxindex = psy_star.argmax()

xmmse_most = np.clip(xmmse_star[maxindex], 0, np.inf)
pet = np.repeat(tlist[xmmse_most > 0], c_star[maxindex][xmmse_most > 0])
cha = np.repeat(xmmse_most[xmmse_most > 0] / factor[xmmse_most > 0] / c_star[maxindex][xmmse_most > 0], c_star[maxindex][xmmse_most > 0])

mu = np.average(c_star.sum(axis=1), weights=psy_star)
t0 = t0_t
output = np.array([(xmmse_most / factor)[(tlist > t - 0.5) & (tlist < t + 0.5)].sum() for t in range(WindowSize)])
alpha = opti.fmin_l_bfgs_b(lambda alpha: wff.rss_alpha(alpha, output, wave, mnecpu), x0=[0.01], approx_grad=True, bounds=[[1e-20, np.inf]], maxfun=50000)[0]
cha = cha * alpha
mu_i = cha.sum()
t0_i = t0_t
```

Motivation: show the performance of α on FBMP.

Progress: $\mathrm{FBMP}_{\max}$ is μ estimation with α correction, FBMP is simple weighted average of N_{PE} of each $\boldsymbol{z}$.





3 Correction

3.1 ELBO

Induce **ELBO**(Evidence lower bound). We want to minimize $\mathrm{KL}_q(q(\boldsymbol{z})|p(\boldsymbol{z}|\boldsymbol{w}))$,

$$\begin{aligned}
\mathrm{ELBO} &= \log p(\boldsymbol{w}) - \mathrm{KL}_q(q(\boldsymbol{z})|p(\boldsymbol{z}|\boldsymbol{w})) && (51)\\
&= E_q(\log(p(\boldsymbol{w}|\boldsymbol{z}))) - \mathrm{KL}_q(q(\boldsymbol{z})|p(\boldsymbol{z})) && (52)\\
&= E_q(\log(p(\boldsymbol{w}|\boldsymbol{z}))) - E_q(\log \frac{q(\boldsymbol{z})}{p(\boldsymbol{z})}) && (53)\\
&= E_q(\log(p(\boldsymbol{w}|\boldsymbol{z})p(\boldsymbol{z}))) - E_q(\log q(\boldsymbol{z})) && (54)\\
&= E_q(\nu) + S_q && (55)
\end{aligned}$$

where S is entropy.
Let

$$q(\boldsymbol{z})_{\boldsymbol{\theta}} = \begin{cases} \frac{\exp \nu(\boldsymbol{z})}{\sum_{\boldsymbol{z}\in\mathcal{Z}'} \exp \nu(\boldsymbol{z})} & \text{if } \boldsymbol{z} \in \mathcal{Z}' \\ 0 & \text{if } \boldsymbol{z} \notin \mathcal{Z}' \end{cases} \tag{56}$$

Additionally,

$$\mathrm{ELBO} = \log \sum_{\boldsymbol{z}\in\mathcal{Z}'} \exp \nu \tag{57}$$

On the other hand, let

$$q(\boldsymbol{z})_{\boldsymbol{\theta}} = \begin{cases} \frac{f(\boldsymbol{\theta},\boldsymbol{z})}{\sum_{\boldsymbol{z}\in\mathcal{Z}'} f(\boldsymbol{\theta},\boldsymbol{z})} & \text{if } \boldsymbol{z} \in \mathcal{Z}' \\ 0 & \text{if } \boldsymbol{z} \notin \mathcal{Z}' \end{cases} \tag{58}$$

$$\forall \boldsymbol{z} \in \mathcal{Z}', f(\boldsymbol{\theta}, \boldsymbol{z}) > 0 \tag{59}$$

But, $f(\boldsymbol{\theta}, \boldsymbol{z}) = \exp(\nu + g(\boldsymbol{\theta}, |\boldsymbol{z}|))$ are bad.

$$
\begin{aligned}
\mathrm{ELBO}'_{\boldsymbol{\theta}} =&(\sum_{\boldsymbol{z}\in\mathcal{Z}'} \frac{\mathrm{e}^{\nu+g(\boldsymbol{\theta},|\boldsymbol{z}|)}}{\sum_{\boldsymbol{z}\in\mathcal{Z}'}\mathrm{e}^{\nu+g(\boldsymbol{\theta},|\boldsymbol{z}|)}}\nu && (60)\\
&-\sum_{\boldsymbol{z}\in\mathcal{Z}'} \frac{\mathrm{e}^{\nu+g(\boldsymbol{\theta},|\boldsymbol{z}|)}}{\sum_{\boldsymbol{z}\in\mathcal{Z}'}\mathrm{e}^{\nu+g(\boldsymbol{\theta},|\boldsymbol{z}|)}}\log\frac{\mathrm{e}^{\nu+g(\boldsymbol{\theta},|\boldsymbol{z}|)}}{\sum_{\boldsymbol{z}\in\mathcal{Z}'}\mathrm{e}^{\nu+g(\boldsymbol{\theta},|\boldsymbol{z}|)}})' && (61)\\
=&(-\sum_{\boldsymbol{z}\in\mathcal{Z}'} \frac{\mathrm{e}^{\nu+g(\boldsymbol{\theta},|\boldsymbol{z}|)}}{\sum_{\boldsymbol{z}\in\mathcal{Z}'}\mathrm{e}^{\nu+g(\boldsymbol{\theta},|\boldsymbol{z}|)}}g(\boldsymbol{\theta},|\boldsymbol{z}|) && (62)\\
&+\sum_{\boldsymbol{z}\in\mathcal{Z}'} \frac{\mathrm{e}^{\nu+g(\boldsymbol{\theta},|\boldsymbol{z}|)}}{\sum_{\boldsymbol{z}\in\mathcal{Z}'}\mathrm{e}^{\nu+g(\boldsymbol{\theta},|\boldsymbol{z}|)}}\log\sum_{\boldsymbol{z}\in\mathcal{Z}'}\mathrm{e}^{\nu+g(\boldsymbol{\theta},|\boldsymbol{z}|)})' && (63)\\
=&-\sum_{\boldsymbol{z}\in\mathcal{Z}'} \frac{\mathrm{e}^{\nu+g(\boldsymbol{\theta},|\boldsymbol{z}|)}}{\sum_{\boldsymbol{z}\in\mathcal{Z}'}\mathrm{e}^{\nu+g(\boldsymbol{\theta},|\boldsymbol{z}|)}}g'(\boldsymbol{\theta},|\boldsymbol{z}|) && (64)\\
&-\sum_{\boldsymbol{z}\in\mathcal{Z}'} (\frac{\mathrm{e}^{\nu+g(\boldsymbol{\theta},|\boldsymbol{z}|)}}{\sum_{\boldsymbol{z}\in\mathcal{Z}'}\mathrm{e}^{\nu+g(\boldsymbol{\theta},|\boldsymbol{z}|)}})'g(\boldsymbol{\theta},|\boldsymbol{z}|) && (65)\\
&+\frac{(\sum_{\boldsymbol{z}\in\mathcal{Z}'}\mathrm{e}^{\nu+g(\boldsymbol{\theta},|\boldsymbol{z}|)})'}{\sum_{\boldsymbol{z}\in\mathcal{Z}'}\mathrm{e}^{\nu+g(\boldsymbol{\theta},|\boldsymbol{z}|)}} && (66)\\
=&-\sum_{\boldsymbol{z}\in\mathcal{Z}'} (\frac{\mathrm{e}^{\nu+g(\boldsymbol{\theta},|\boldsymbol{z}|)}}{\sum_{\boldsymbol{z}\in\mathcal{Z}'}\mathrm{e}^{\nu+g(\boldsymbol{\theta},|\boldsymbol{z}|)}})'g(\boldsymbol{\theta},|\boldsymbol{z}|) && (67)
\end{aligned}
$$

when $\forall\boldsymbol{z}, g(\boldsymbol{\theta},|\boldsymbol{z}|)=0$, $\mathrm{ELBO}'_{\boldsymbol{\theta}}=0$.
It is hard to use **ELBO** to correct the bias of μ.