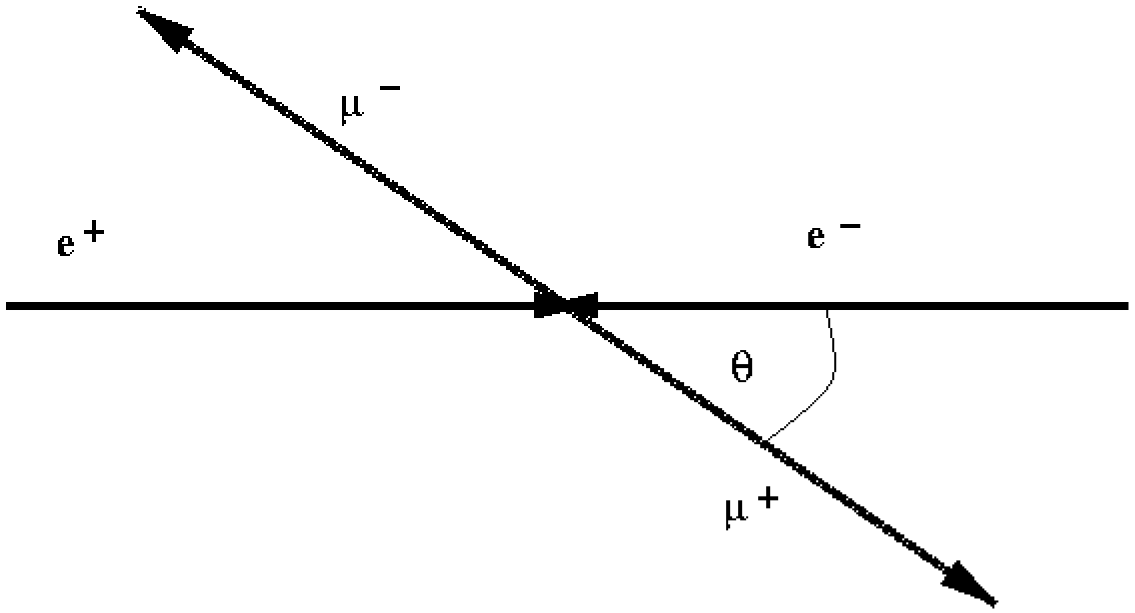
Assessment 1

This is the first assessed mini-project. When electrons (e^-) and positrons (e^+) collide together they sometimes produce two muons (μ^-, μ^+) that come out of the collision back-to-back. The angle between the incoming e^+ and the outgoing μ^+ is defined to be θ. See the figure below:



If the energy of the e^-e^+ collision is a long way below the mass of the Z boson then θ has a distribution of $1 + \cos^2\theta$. However, as the centre of mass energy of the collision approaches that of the Z boson an asymmetry appears and the distribution becomes $\mathcal{A}(1 + \cos^2\theta) + \mathcal{B}\cos\theta$ which can be written $(1 + \cos^2\theta) + \dfrac{\mathcal{B}}{\mathcal{A}}\cos\theta$. The ratio $\dfrac{\mathcal{B}}{\mathcal{A}} = \kappa$ varies greatly (up to a maximum of ~10%) around the collision energies near the centre of mass of the Z boson and even changes sign. In the 1990s measuring these assymetries was an important scientific goal as they told us much about electroweak unification.

In the early 1990s the LEP collider at CERN was one of the first e^-e^+ colliders to run near the Z mass. The aim of the mini-project is to determine how well the experiments could measure κ in three different scenarios:

1. When LEP started running each experiment would collect a few 10s of these events/day and (say) would run for 100 days a year.
2. A couple of years later each experiment would collect a few 100s of these events/day and (say) would run for 100 days a year.
3. A couple of years later again each experiment would collect a few 1000s of these events/day and (say) would run for 100 days a year.

You should consider (and simulate) three different values for $\kappa = \pm 0.07$ and 0. You should fit your simulation to consider how well you can tell them apart and what precision can you make on the individual measurements.

Assessment Criteria

You will submit a jupyter-notebook that is both a record of your analysis (so that we can run your code to verify your findings) and a description of what you have found out. You have an overall mark out of 20 that will have two components (each out of 10). These are:

Technical achievement

This will be a judge of: how well you have modelled the situation and the techiques that you have used, the quality of your code, how well you have analysed the data etc

Communicating your results

This will be a judge of: how well you have understood what you have seen, how well you have communicated your understanding (this includes clarity of explanation and presentational aspects of the plots) etc

My submission - Mihir Koka

Analytical Approach

To start, I first attempted an analytical approach to try solve the problem and see if the cumulative distribution function (CDF) could be inverted to sample from the given probability density function (PDF).

The PDF is given as:

$$f(\theta) = C \cdot \left(1 + \cos^2(\theta) + \kappa \cos(\theta)\right),$$

where (C) is the normalizing constant. To find (C), I computed the definite integral of $(f(\theta))$ over the range $([0, \pi])$:

$$\int_0^\pi f(\theta)\, d\theta = 1.$$

Expanding $f(\theta)$, we have:

$$\int_0^\pi \left(1 + \cos^2(\theta) + \kappa \cos(\theta)\right)\, d\theta.$$

Breaking this into separate terms:

1. The integral of 1 over $[0, \pi]$ is:

$$\int_0^\pi 1 \, d\theta = \pi.$$

2. The integral of $\cos^2(\theta)$ over $[0, \pi]$ can be simplified using a trigonometric identity to:

$$\int_0^\pi \cos^2(\theta) \, d\theta = \int_0^\pi \frac{1 + \cos(2\theta)}{2} \, d\theta = \frac{\pi}{2}.$$

3. The integral of $\kappa \cos(\theta)$ over $[0, \pi]$ is:

$$\int_0^\pi \kappa \cos(\theta) \, d\theta = \kappa \sin(\theta)\Big|_0^\pi = 0.$$

Adding these results together:

$$\int_0^\pi \left(1 + \cos^2(\theta) + \kappa \cos(\theta)\right) \, d\theta = \pi + \frac{\pi}{2} + 0 = \frac{3\pi}{2}.$$

Thus, the normalizing constant is:

$$C = \frac{2}{3\pi}.$$

Cumulative Distribution Function (CDF)

The cumulative distribution function (CDF) is obtained by integrating the normalized PDF from 0 to θ:

$$F(\theta) = \int_0^\theta f(\theta') \, d\theta' = \frac{2}{3\pi} \int_0^\theta \left(1 + \cos^2(\theta') + \kappa \cos(\theta')\right) \, d\theta'.$$

Carrying out the integration term by term:

1. The integral of 1 is:

$$\int_0^\theta 1 \, d\theta' = \theta.$$

2. The integral of $\cos^2(\theta')$ is:

$$\int_0^\theta \cos^2(\theta') \, d\theta' = \frac{1}{2} \int_0^\theta \left(1 + \cos(2\theta')\right) \, d\theta' = \frac{\theta}{2} + \frac{\sin(2\theta)}{4}.$$

3. The integral of $\kappa \cos(\theta')$ is:

$$\int_0^\theta \kappa \cos(\theta') \, d\theta' = \kappa \sin(\theta).$$

Combining these results:

$$F(\theta) = \frac{2}{3\pi}\left(\frac{3\theta}{2} + \frac{\sin(2\theta)}{4} + \kappa\sin(\theta)\right).$$

Invertibility of the CDF

The CDF is a transcendental function. That means it can't be inverted analytically to solve for θ in terms of $F(\theta)$.

Substitution with $u = \cos(\theta)$

So I tried using a substitution instead, as with this we get something that looks like a quadtratic To simplify the integral, I substituted $u = \cos(\theta)$, which gives:

$$\sin(\theta)\,d\theta = -\frac{du}{\sqrt{1-u^2}}.$$

The PDF in terms of u becomes:

$$f(u) = C \cdot \frac{1 + u^2 + \kappa u}{\sqrt{1-u^2}}.$$

However, this integral also cannot be inverted analytically due to the square root term in the denominator.

Conclusion

So with all this I wasn't able to solve the question analytically. Therefore I will now try doing it with a **numerical approach** . This will involve the methods we went through in lecture, those being either a **numerical inverse transform sampling** or the **accept-reject method**.

Numerical approach: plan

So I will generate data for κ ∈ {−0.07, 0, +0.07} and event rates {10/day, 100/day, 1000/day} over 100 days (N ∈ {1000, 10000, 100000}):

- Step 1: Set up the normalized model, CDF, and two samplers (numerical inverse CDF and accept-reject).
- Step 2: Generate datasets for all 9 cases and store them.
- Step 3: Unbinned MLE with iminuit (use HESSE and MINOS errors).
- Step 4: Binned likelihood (Poisson) fits for multiple binning choices and compare to unbinned.
- Step 5: Do some Least-squares fits
- Step 6: For each method, study κ uncertainty vs event rate and compare HESSE vs MINOS, Plot distributions.
- Step 7: Summarize results

Step 1 — Numerical inverse transform

Plan

- Build a grid in $\theta \in [0, \pi]$.
- Compute the unnormalised shape $g(\theta|\kappa) = 1 + \cos^2\theta + \kappa\cos\theta$.
- Normalize numerically to get the pdf by dividing by the area under g. (like in lecture)
- Compute the CDF by cumulative sum over the grid and normalise to 1.
- Construct an inverse map with a spline: $\theta = F^{-1}(u)$.
- Draw $u \sim \mathcal{U}(0, 1)$ and map to θ; check with a histogram vs the pdf.

In [ ]:
```python
import numpy as np
import numpy.random as npr
import scipy.interpolate as spi
import matplotlib.pyplot as plt

# fixed random seed so we can reproduce results
rng = np.random.default_rng(6067948) #My CID !!

# constants
PI = np.pi

# temporary kappa
kappa = 0.07

# build a theta grid on [0, π]
theta = np.linspace(0.0, PI, 200_001)
dtheta = theta[1] - theta[0]

g = 1.0 + np.cos(theta)**2 + kappa * np.cos(theta)

# CDF via cumulative sum
F = np.cumsum(g)
F /= F[-1]
F[0]  = 0.0
F[-1] = 1.0

pdf = g / (g.sum() * dtheta)

print("PDF min on grid:", pdf.min())
print("Normalization check (sum*dθ):", (pdf*dtheta).sum())
print("CDF endpoints:", F[0], F[-1])
```

```
PDF min on grid: 0.21194522475469074
Normalization check (sum*dθ): 0.9999999999999998
CDF endpoints: 0.0 1.0
```

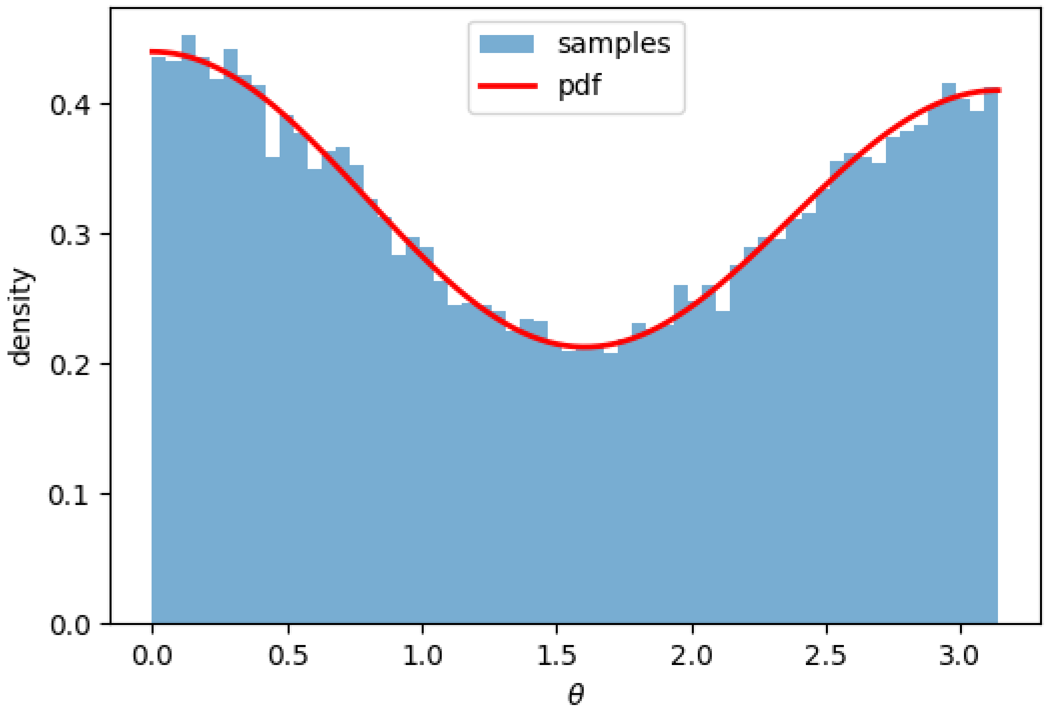
Step 2 — Inverse-CDF sampling and make the 9 datasets

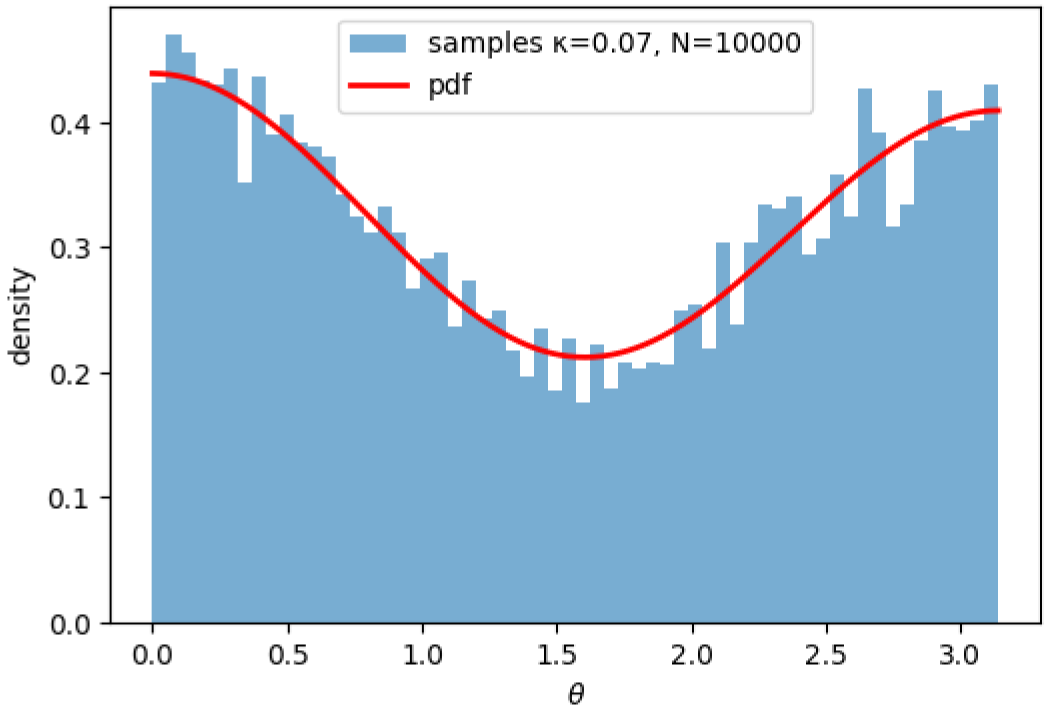
- Build θ = F^{-1}(u) with a linear spline on the CDF.
- Sample u ~ U(0,1), map to θ, and compare a histogram to the target pdf.

- Loop over κ ∈ {−0.07, 0, +0.07} and N ∈ {1000, 10000, 100000} to make 9 datasets.

In [ ]:
```python
invF = spi.interp1d(F, theta, kind="linear", bounds_error=False,
                    fill_value=(theta[0], theta[-1]))

N_demo = 50_000
u = rng.uniform(0.0, 1.0, size=N_demo)
theta_samples = invF(u)

# compare histogram to target pdf
plt.figure(figsize=(6,4))
plt.hist(theta_samples, bins=60, range=(0, PI), density=True,
         alpha=0.6, label="samples")
plt.plot(theta[::400], pdf[::400], "r-", lw=2, label="pdf")
plt.xlabel(r"$\theta$")
plt.ylabel("density")
plt.legend()
plt.show()

#-----------------------------------------------------------------------------------
# Generate the 9 datasets

kappa_vals = [-0.07, 0.0, 0.07]
Ns = [1_000, 10_000, 100_000]

datasets = {}  # where key is kappa, value is the data itself

for kappa in kappa_vals:
    g_k = 1.0 + np.cos(theta)**2 + kappa * np.cos(theta)
    F_k = np.cumsum(g_k)
    F_k /= F_k[-1]
    F_k[0]  = 0.0
    F_k[-1] = 1.0

    invF_k = spi.interp1d(F_k, theta, kind="linear", bounds_error=False,
                          fill_value=(theta[0], theta[-1]))

    for N in Ns:
        u = rng.uniform(0.0, 1.0, size=N)
        datasets[(kappa, N)] = invF_k(u)

# plotting one example
k_plot, N_plot = 0.07, 10_000
g_plot = 1.0 + np.cos(theta)**2 + k_plot * np.cos(theta)
pdf_plot = g_plot / (g_plot.sum() * dtheta)

plt.figure(figsize=(6,4))
plt.hist(datasets[(k_plot, N_plot)], bins=60, range=(0, PI), density=True,
         alpha=0.6, label=f"samples κ={k_plot}, N={N_plot}")
plt.plot(theta[::400], pdf_plot[::400], "r-", lw=2, label="pdf")
plt.xlabel(r"$\theta$")
plt.ylabel("density")
plt.legend()
plt.show()
```

```
print("Datasets generated:", sorted(datasets.keys()))
```





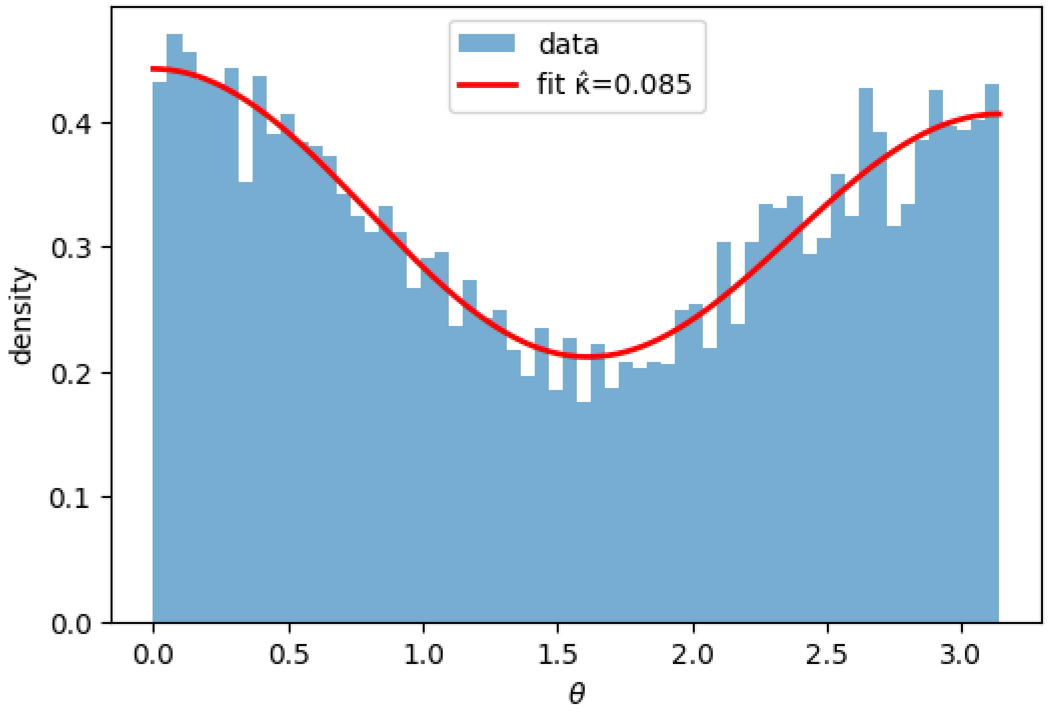
```
Datasets generated: [(-0.07, 1000), (-0.07, 10000), (-0.07, 100000), (0.0, 1000),
(0.0, 10000), (0.0, 100000), (0.07, 1000), (0.07, 10000), (0.07, 100000)]
```

Step 3 - Unbinned MLE for Kappa (inminuit)

In [ ]:
```
import numpy as np
from iminuit import Minuit

# pick a dataset to fit
```

```python
k_true, N_sel = 0.07, 10_000
theta_data = datasets[(k_true, N_sel)]
cos_th = np.cos(theta_data)

# negative log-likelihood
def nll(kappa):
    g = 1.0 + cos_th**2 + kappa * cos_th
    if np.any(g <= 0.0):
        return np.inf
    return -np.sum(np.log(g))

m = Minuit(nll, kappa=0.0)
m.limits["kappa"] = (-1.9, 1.9)
m.errordef = Minuit.LIKELIHOOD

m.migrad()
m.hesse()
m.minos("kappa")

k_hat = m.values["kappa"]
k_hesse = m.errors["kappa"]
k_minos = m.merrors["kappa"]

print(f"Example fit (true κ={k_true}, N={N_sel})")
print(f"  κ̂= {k_hat:.5f}  (HESSE σ = {k_hesse:.5f})")
print(f"  MINOS: -{abs(k_minos.lower):.5f}  +{abs(k_minos.upper):.5f}")

# overlay fit on histogram
theta_grid = theta
g_fit = 1.0 + np.cos(theta_grid)**2 + k_hat * np.cos(theta_grid)
pdf_fit = g_fit / (g_fit.sum() * (theta_grid[1] - theta_grid[0]))

import matplotlib.pyplot as plt
PI = np.pi
plt.figure(figsize=(6,4))
plt.hist(theta_data, bins=60, range=(0, PI), density=True, alpha=0.6, label="data")
plt.plot(theta_grid[::400], pdf_fit[::400], "r-", lw=2, label=f"fit κ̂={k_hat:.3f}")
plt.xlabel(r"$\theta$")
plt.ylabel("density")
plt.legend()
plt.show()
```

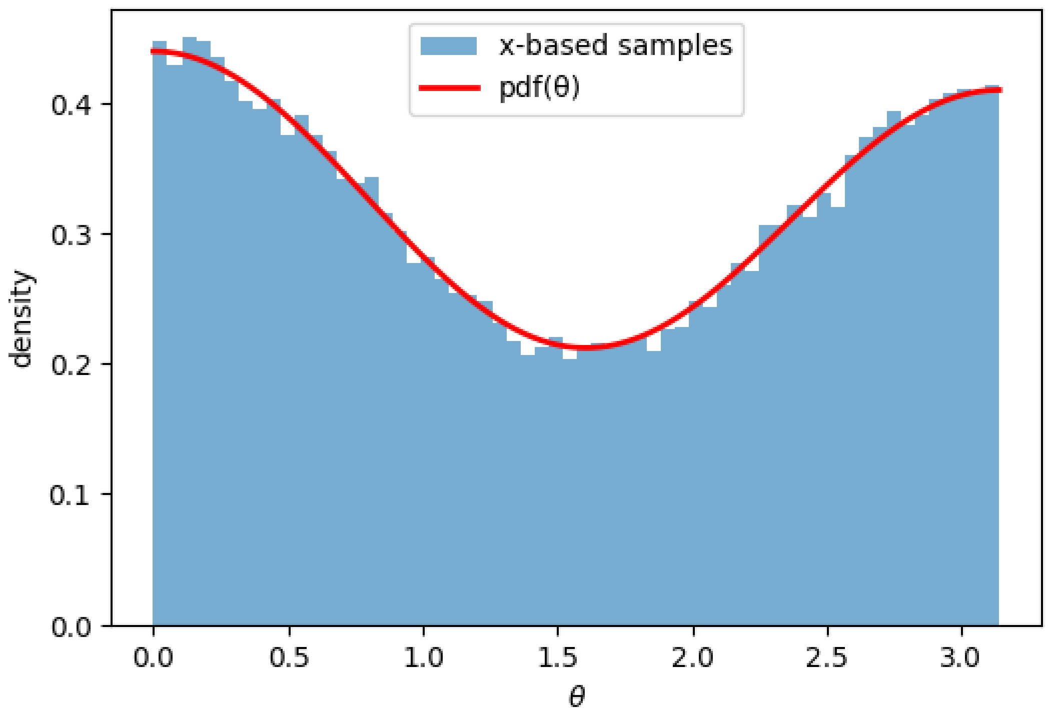
```
Example fit (true κ=0.07, N=10000)
  κ̂= 0.08535  (HESSE σ = 0.02255)
  MINOS: -0.02253  +0.02256
```



In [5]:
```
from iminuit import Minuit

summary = []
for (k_true, N_sel), theta_data in sorted(datasets.items()):
    cos_th = np.cos(theta_data)

    def nll(kappa):
        g = 1.0 + cos_th**2 + kappa * cos_th
        if np.any(g <= 0.0):
            return np.inf
        return -np.sum(np.log(g))

    m = Minuit(nll, kappa=0.0)
    m.limits["kappa"] = (-1.9, 1.9)
    m.errordef = Minuit.LIKELIHOOD
    m.migrad()
    m.hesse()

    k_hat = m.values["kappa"]
    k_err = m.errors["kappa"]
    summary.append((k_true, N_sel, k_hat, k_err))

print("Unbinned MLE summary (HESSE errors):")
for k_true, N_sel, k_hat, k_err in summary:
    print(f"  κ_true={k_true:+.2f}, N={N_sel:6d}  ->  κ̂={k_hat:+.5f} ± {k_err:.5f}"
Unbinned MLE summary (HESSE errors):
  κ_true=-0.07, N=  1000  ->  κ̂=+0.00216 ± 0.07184
  κ_true=-0.07, N= 10000  ->  κ̂=-0.04826 ± 0.02255
  κ_true=-0.07, N=100000  ->  κ̂=-0.06670 ± 0.00716
  κ_true=+0.00, N=  1000  ->  κ̂=-0.10801 ± 0.07165
  κ_true=+0.00, N= 10000  ->  κ̂=-0.02021 ± 0.02264
  κ_true=+0.00, N=100000  ->  κ̂=+0.00341 ± 0.00716
  κ_true=+0.07, N=  1000  ->  κ̂=+0.12196 ± 0.07172
  κ_true=+0.07, N= 10000  ->  κ̂=+0.08535 ± 0.02255
  κ_true=+0.07, N=100000  ->  κ̂=+0.06387 ± 0.00715
```

trying using the same substitution we used in the analytical method

In [ ]:
```python
# x = cos(theta) inverse-CDF sampler

from scipy.integrate import cumulative_trapezoid
from scipy.interpolate import PchipInterpolator

eps = 1e-6  # back off from ±1 (avoid spikes at the ends, so we cut of just before
x = np.linspace(-1.0 + eps, 1.0 - eps, 200_001)

# unnormalised shape in x
g_x = (1.0 + x**2 + kappa * x) / np.sqrt(1.0 - x**2)

# CDF
F_x = cumulative_trapezoid(g_x, x, initial=0.0)
F_x /= F_x[-1]

# monotone inverse
invF_x = PchipInterpolator(F_x, x, extrapolate=False)

# sample u inside (0,1)
ueps = 1e-12
u = rng.random(50_000)
u = u * (1.0 - 2*ueps) + ueps

x_samples = invF_x(u)
x_samples = np.clip(x_samples, -1.0, 1.0)
theta_samples_x = np.arccos(x_samples)

# compare to ϑ-pdf from Step 1
plt.figure(figsize=(6,4))
plt.hist(theta_samples_x, bins=60, range=(0, PI), density=True, alpha=0.6, label="x
plt.plot(theta[::400], pdf[::400], "r-", lw=2, label="pdf(θ)")
plt.xlabel(r"$\theta$")
plt.ylabel("density")
plt.legend()
plt.show()
```

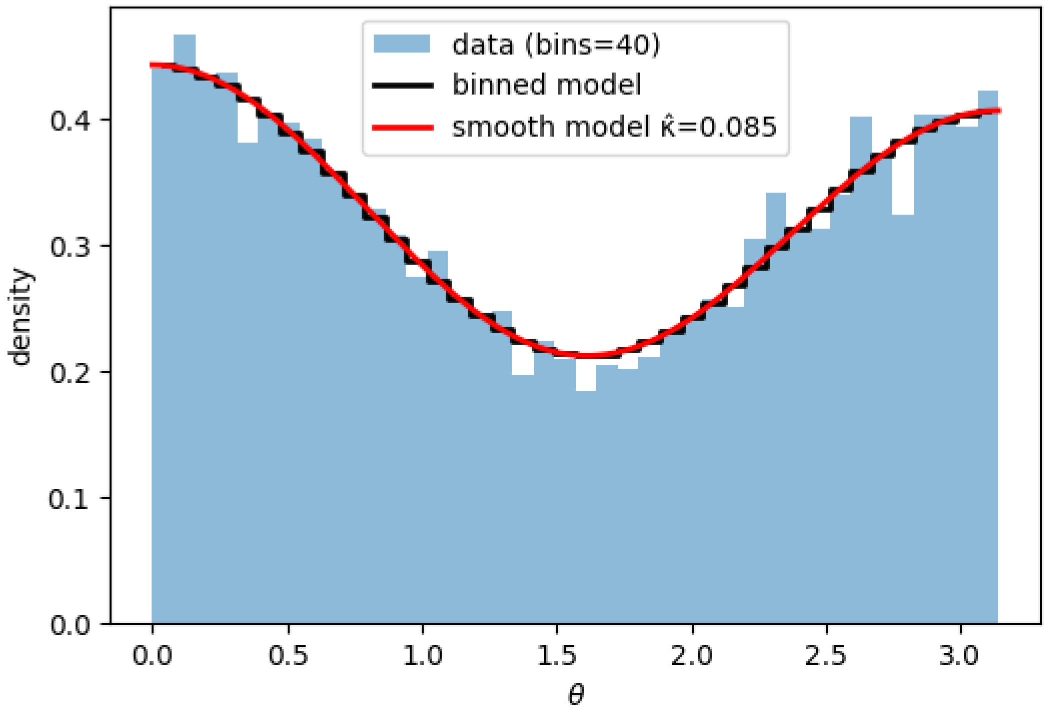


Step 4 - Binned poisson likelihood

In [ ]:
```
# Step 4 — Binned Poisson likelihood fits (multiple binnings) and comparison to unb

import numpy as np
from iminuit import Minuit
from scipy.integrate import cumulative_trapezoid
import matplotlib.pyplot as plt

# a helper to keep track of expected bin counts for given kappa and bin edges
def expected_counts(kappa, edges, N):
    g = 1.0 + np.cos(theta)**2 + kappa * np.cos(theta)
    G = cumulative_trapezoid(g, theta, initial=0.0)  # integral from 0 to ϑ (same a
    total = G[-1]

    G_edges = np.interp(edges, theta, G)
    bin_area = np.diff(G_edges)
    probs = bin_area / total
    mu = N * probs
    return mu

# negative log-likelihood for Poisson-binned counts
def make_nll_binned(n_obs, edges):
    N = int(np.sum(n_obs))
    def nll(kappa):
        mu = expected_counts(kappa, edges, N)
        if np.any(mu <= 0.0):
            return np.inf
        return np.sum(mu - n_obs * np.log(mu))
    return nll

# Example-1: we try and fit one dataset with several bin choices and compare to unb
```

```
k_true, N_sel = 0.07, 10_000
theta_data = datasets[(k_true, N_sel)]

bin_choices = [10, 20, 40, 60]
results = []

for nb in bin_choices:
    edges = np.linspace(0.0, PI, nb + 1)
    n_obs, _ = np.histogram(theta_data, bins=edges)

    nll = make_nll_binned(n_obs, edges)
    m = Minuit(nll, kappa=0.0)
    m.limits["kappa"] = (-1.9, 1.9)
    m.errordef = Minuit.LIKELIHOOD
    m.migrad()
    m.hesse()

    k_hat_b = m.values["kappa"]
    k_err_b = m.errors["kappa"]
    results.append((nb, k_hat_b, k_err_b))

# print summary and compare to unbinned result from Step 3 for the same data
print(f"Binned Poisson MLE (k_true={k_true:+.2f}, N={N_sel})")
for nb, k_hat_b, k_err_b in results:
    print(f"  bins={nb:2d}  ->  κ̂={k_hat_b:+.5f} ± {k_err_b:.5f} (HESSE)")

# try to plot one binned fit overlay, 40 in this case, against the model density
nb_plot = 40
edges = np.linspace(0.0, PI, nb_plot + 1)
n_obs, _ = np.histogram(theta_data, bins=edges)
nll = make_nll_binned(n_obs, edges)
m = Minuit(nll, kappa=0.0); m.limits["kappa"]=(-1.9,1.9); m.errordef=Minuit.LIKELIH
m.migrad(); m.hesse()
k_hat_b = m.values["kappa"]

# model density
g_fit = 1.0 + np.cos(theta)**2 + k_hat_b * np.cos(theta)
pdf_fit = g_fit / (cumulative_trapezoid(g_fit, theta, initial=0.0)[-1])

# binned model as a step-density: μ_bin / (N * bin_width)
mu = expected_counts(k_hat_b, edges, N_sel)
bin_w = np.diff(edges)
dens_bin = mu / (N_sel * bin_w)

plt.figure(figsize=(6,4))
plt.hist(theta_data, bins=edges, density=True, alpha=0.5, label=f"data (bins={nb_pl
plt.step((edges[:-1]+edges[1:])/2, dens_bin, where="mid", color="k", lw=2, label="b
plt.plot(theta[::400], pdf_fit[::400], "r-", lw=2, label=f"smooth model κ̂={k_hat_b:
plt.xlabel(r"$\theta$")
plt.ylabel("density")
plt.legend()
plt.show()
```

```
Binned Poisson MLE (k_true=+0.07, N=10000)
  bins=10  ->  κ̂=+0.08524 ± 0.02261 (HESSE)
  bins=20  ->  κ̂=+0.08536 ± 0.02257 (HESSE)
  bins=40  ->  κ̂=+0.08547 ± 0.02255 (HESSE)
  bins=60  ->  κ̂=+0.08507 ± 0.02255 (HESSE)
```



In [19]:
```python
# Step 4.5? — Runnning the same binned fits for all datasets (fixed binning)

from iminuit import Minuit
from scipy.integrate import cumulative_trapezoid

nbins = 40
edges = np.linspace(0.0, PI, nbins + 1)

summary_binned = []
for (k_true, N_sel), theta_data in sorted(datasets.items()):
    n_obs, _ = np.histogram(theta_data, bins=edges)
    nll = make_nll_binned(n_obs, edges)
    m = Minuit(nll, kappa=0.0)
    m.limits["kappa"] = (-1.9, 1.9)
    m.errordef = Minuit.LIKELIHOOD
    m.migrad()
    m.hesse()
    summary_binned.append((k_true, N_sel, m.values["kappa"], m.errors["kappa"]))

print(f"Binned Poisson MLE summary (bins={nbins}, HESSE errors):")
for k_true, N_sel, k_hat, k_err in summary_binned:
    print(f"  κ_true={k_true:+.2f}, N={N_sel:6d} -> κ̂={k_hat:+.5f} ± {k_err:.5f}")
```

```
Binned Poisson MLE summary (bins=40, HESSE errors):
  κ_true=-0.07, N=  1000 -> κ̂=+0.00525 ± 0.07186
  κ_true=-0.07, N= 10000 -> κ̂=-0.04778 ± 0.02255
  κ_true=-0.07, N=100000 -> κ̂=-0.06644 ± 0.00716
  κ_true=+0.00, N=  1000 -> κ̂=-0.10781 ± 0.07165
  κ_true=+0.00, N= 10000 -> κ̂=-0.02065 ± 0.02264
  κ_true=+0.00, N=100000 -> κ̂=+0.00347 ± 0.00716
  κ_true=+0.07, N=  1000 -> κ̂=+0.12293 ± 0.07176
  κ_true=+0.07, N= 10000 -> κ̂=+0.08547 ± 0.02255
  κ_true=+0.07, N=100000 -> κ̂=+0.06359 ± 0.00715
```

Estimates seem to converge to κ_true as N increases.

Errors scale ≈ 1/√N: 0.072 → 0.0226 → 0.00716.

Step 5 - Least squares fits

In [14]:
```python
# Step 5 — Least-squares (chi-square) fits

import numpy as np
from iminuit import Minuit
from scipy.integrate import cumulative_trapezoid
import matplotlib.pyplot as plt

# reuse expected_counts from Step 4
def expected_counts(kappa, edges, N):
    g = 1.0 + np.cos(theta)**2 + kappa * np.cos(theta)
    G = cumulative_trapezoid(g, theta, initial=0.0)
    total = G[-1]
    G_edges = np.interp(edges, theta, G)
    probs = np.diff(G_edges) / total
    return N * probs

# chi-square
def make_chi2(n_obs, edges):
    N = int(np.sum(n_obs))
    def chi2(kappa):
        mu = expected_counts(kappa, edges, N)
        mu = np.clip(mu, 1e-9, None)
        return np.sum((n_obs - mu)**2 / mu)
    return chi2

# Example: compare chi2 vs Poisson for one dataset
k_true, N_sel = 0.07, 10_000
theta_data = datasets[(k_true, N_sel)]
edges = np.linspace(0.0, PI, 40 + 1)
n_obs, _ = np.histogram(theta_data, bins=edges)

# Chi-square fit
chi2 = make_chi2(n_obs, edges)
m_chi2 = Minuit(chi2, kappa=0.0)
m_chi2.limits["kappa"] = (-1.9, 1.9)
m_chi2.errordef = Minuit.LEAST_SQUARES  # = 1
m_chi2.migrad(); m_chi2.hesse()
k_hat_chi2, k_err_chi2 = m_chi2.values["kappa"], m_chi2.errors["kappa"]
```

```python
# Poisson fit (from Step 4) for comparison
def make_nll_binned(n_obs, edges):
    N = int(np.sum(n_obs))
    def nll(kappa):
        mu = expected_counts(kappa, edges, N)
        if np.any(mu <= 0.0):
            return np.inf
        return np.sum(mu - n_obs * np.log(mu))
    return nll

nll = make_nll_binned(n_obs, edges)
m_pois = Minuit(nll, kappa=0.0)
m_pois.limits["kappa"] = (-1.9, 1.9)
m_pois.errordef = Minuit.LIKELIHOOD
m_pois.migrad(); m_pois.hesse()
k_hat_pois, k_err_pois = m_pois.values["kappa"], m_pois.errors["kappa"]

print(f"Example (k_true={k_true:+.2f}, N={N_sel}):")
print(f"  Chi2:   κ̂={k_hat_chi2:+.5f} ± {k_err_chi2:.5f}")
print(f"  Poisson κ̂={k_hat_pois:+.5f} ± {k_err_pois:.5f}")

# Run chi-square fits for all datasets (same 40-bin scheme)
nbins = 40
edges = np.linspace(0.0, PI, nbins + 1)
summary_chi2 = []
for (k_true, N_sel), theta_data in sorted(datasets.items()):
    n_obs, _ = np.histogram(theta_data, bins=edges)
    chi2 = make_chi2(n_obs, edges)
    m = Minuit(chi2, kappa=0.0)
    m.limits["kappa"] = (-1.9, 1.9)
    m.errordef = Minuit.LEAST_SQUARES
    m.migrad(); m.hesse()
    summary_chi2.append((k_true, N_sel, m.values["kappa"], m.errors["kappa"]))

print(f"Chi-square summary (bins={nbins}, HESSE errors):")
for k_true, N_sel, k_hat, k_err in summary_chi2:
    print(f"  κ_true={k_true:+.2f}, N={N_sel:6d} -> κ̂={k_hat:+.5f} ± {k_err:.5f}")
```

```
Example (k_true=+0.07, N=10000):
  Chi2:   κ̂=+0.08359 ± 0.02245
  Poisson κ̂=+0.08547 ± 0.02255
Chi-square summary (bins=40, HESSE errors):
  κ_true=-0.07, N=  1000 -> κ̂=+0.00775 ± 0.07104
  κ_true=-0.07, N= 10000 -> κ̂=-0.04578 ± 0.02244
  κ_true=-0.07, N=100000 -> κ̂=-0.06646 ± 0.00715
  κ_true=+0.00, N=  1000 -> κ̂=-0.11461 ± 0.07102
  κ_true=+0.00, N= 10000 -> κ̂=-0.02025 ± 0.02262
  κ_true=+0.00, N=100000 -> κ̂=+0.00356 ± 0.00716
  κ_true=+0.07, N=  1000 -> κ̂=+0.12813 ± 0.07068
  κ_true=+0.07, N= 10000 -> κ̂=+0.08359 ± 0.02245
  κ_true=+0.07, N=100000 -> κ̂=+0.06381 ± 0.00715
```

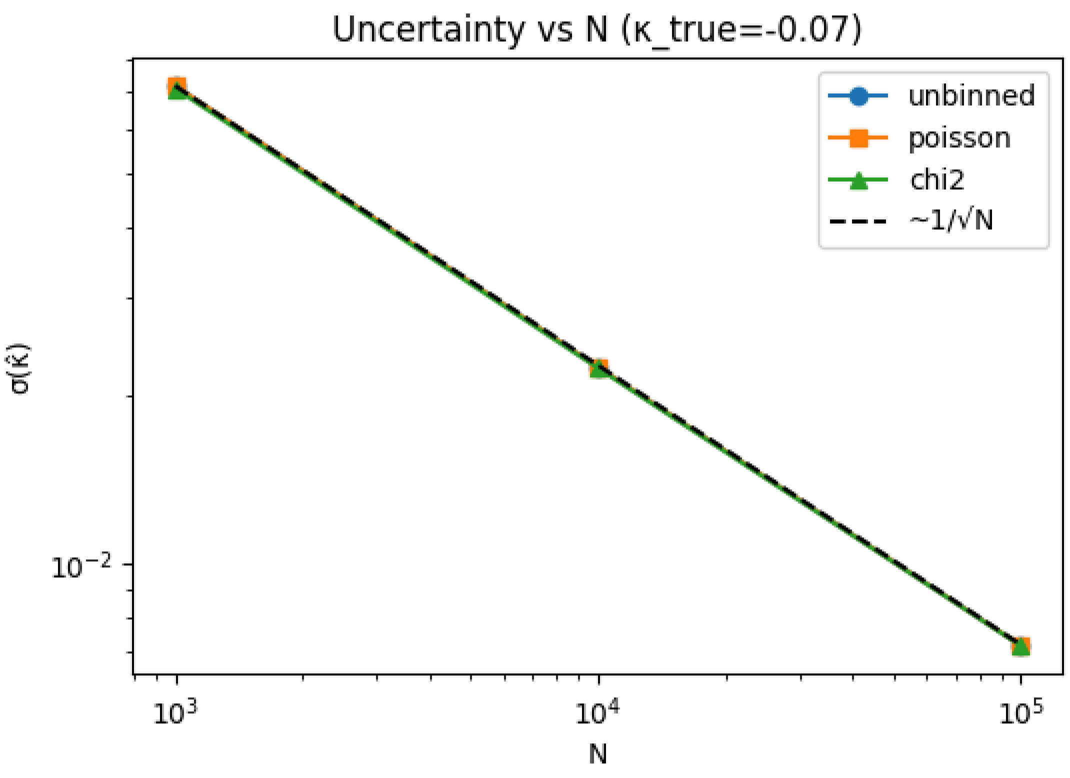
Step 6: compare methods to see how uncertainty scales with N

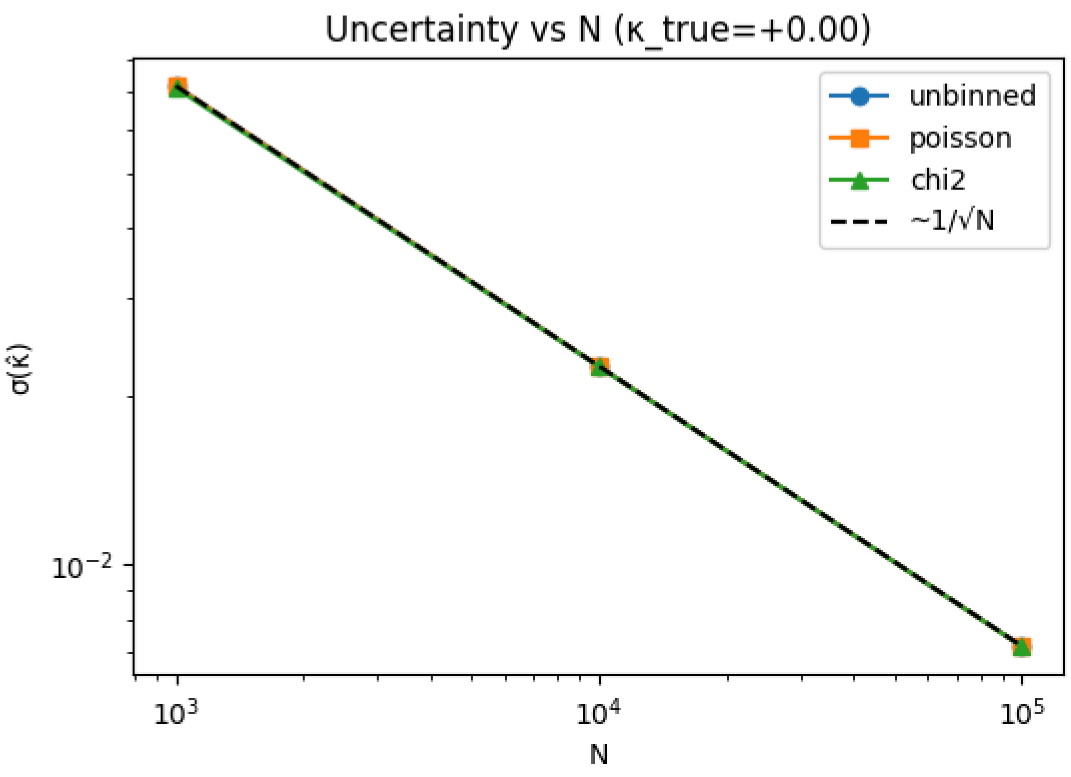
In [ ]:

```python
# Step 6 — Uncertainty vs N, compare methods (unbinned, Poisson-binned, chi-square)
```

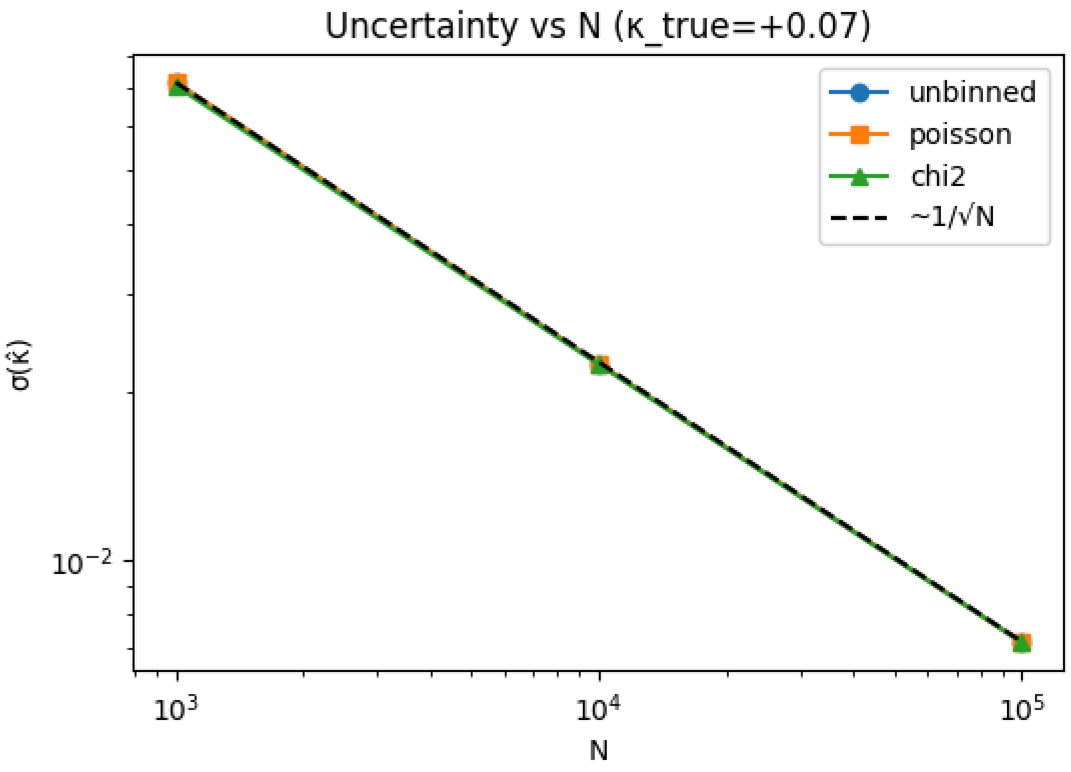
```python
import numpy as np
import matplotlib.pyplot as plt
from iminuit import Minuit
from scipy.integrate import cumulative_trapezoid

# 40-bin scheme for binned methods
nbins = 40
edges = np.linspace(0.0, PI, nbins + 1)

def expected_counts(kappa, edges, N):
    g = 1.0 + np.cos(theta)**2 + kappa * np.cos(theta)
    G = cumulative_trapezoid(g, theta, initial=0.0)
    total = G[-1]
    G_edges = np.interp(edges, theta, G)
    probs = np.diff(G_edges) / total
    return N * probs

def make_nll_binned(n_obs, edges):
    N = int(np.sum(n_obs))
    def nll(kappa):
        mu = expected_counts(kappa, edges, N)
        if np.any(mu <= 0.0):
            return np.inf
        return np.sum(mu - n_obs * np.log(mu))
    return nll

def make_chi2(n_obs, edges):
    N = int(np.sum(n_obs))
    def chi2(kappa):
        mu = expected_counts(kappa, edges, N)
        mu = np.clip(mu, 1e-9, None)
        return np.sum((n_obs - mu)**2 / mu)
    return chi2

def fit_unbinned(theta_data):
    cos_th = np.cos(theta_data)
    def nll(kappa):
        g = 1.0 + cos_th**2 + kappa * cos_th
        if np.any(g <= 0.0):
            return np.inf
        return -np.sum(np.log(g))
    m = Minuit(nll, kappa=0.0)
    m.limits["kappa"] = (-1.9, 1.9)
    m.errordef = Minuit.LIKELIHOOD
    m.migrad(); m.hesse()
    k, e = m.values["kappa"], m.errors["kappa"]
    try:
        m.minos("kappa")
        me = m.merrors["kappa"]
        lo, hi = float(me.lower), float(me.upper)
    except Exception:
        lo, hi = np.nan, np.nan
    return k, e, lo, hi

# Fit all datasets with 3 methods
records = []
```

```
for (k_true, N_sel), theta_data in sorted(datasets.items()):
    # unbinned
    k_u, e_u, lo_u, hi_u = fit_unbinned(theta_data)

    # Poisson-binned
    n_obs, _ = np.histogram(theta_data, bins=edges)
    nll = make_nll_binned(n_obs, edges)
    mp = Minuit(nll, kappa=0.0); mp.limits["kappa"]=(-1.9,1.9); mp.errordef=Minuit.
    mp.migrad(); mp.hesse()
    k_p, e_p = mp.values["kappa"], mp.errors["kappa"]

    # chi-square
    chi2 = make_chi2(n_obs, edges)
    mc = Minuit(chi2, kappa=0.0); mc.limits["kappa"]=(-1.9,1.9); mc.errordef=Minuit
    mc.migrad(); mc.hesse()
    k_c, e_c = mc.values["kappa"], mc.errors["kappa"]

    records.append((k_true, N_sel, "unbinned", k_u, e_u))
    records.append((k_true, N_sel, "poisson",  k_p, e_p))
    records.append((k_true, N_sel, "chi2",     k_c, e_c))

# Helper to slice results into smth readable
def get_series(k_true, method):
    Ns, khat, err = [], [], []
    for kt, N, m, k, e in records:
        if kt == k_true and m == method:
            Ns.append(N); khat.append(k); err.append(e)
    idx = np.argsort(Ns)
    return np.array(Ns)[idx], np.array(khat)[idx], np.array(err)[idx]

methods = ["unbinned", "poisson", "chi2"]
colors  = {"unbinned":"tab:blue","poisson":"tab:orange","chi2":"tab:green"}
markers = {"unbinned":"o","poisson":"s","chi2":"^"}
k_vals  = [-0.07, 0.0, 0.07]

# 1) Error vs N with ~1/sqrt(N)
for kt in k_vals:
    plt.figure(figsize=(6,4))
    for m in methods:
        Ns, kh, er = get_series(kt, m)
        plt.loglog(Ns, er, marker=markers[m], color=colors[m], label=m)
    # 1/sqrt(N) line anchored to unbinned at smallest N
    Ns_ref, _, er_ref = get_series(kt, "unbinned")
    c = er_ref[0] * np.sqrt(Ns_ref[0])
    plt.loglog(Ns_ref, c/np.sqrt(Ns_ref), "k--", label="~1/√N")
    plt.xlabel("N"); plt.ylabel("σ(κ̂)")
    plt.title(f"Uncertainty vs N (κ_true={kt:+.2f})")
    plt.legend()
    plt.show()

# 2) Bias (κ̂− κ_true) vs N
for kt in k_vals:
    plt.figure(figsize=(6,4))
    for m in methods:
        Ns, kh, er = get_series(kt, m)
        plt.semilogx(Ns, kh-kt, marker=markers[m], color=colors[m], label=m)
    plt.axhline(0.0, color="k", lw=1)
    plt.xlabel("N"); plt.ylabel("bias (κ̂−κ_true)")
    plt.title(f"Bias vs N (κ_true={kt:+.2f})")
    plt.legend()
    plt.show()

# 3) Separation power between κ=-0.07 and +0.07
plt.figure(figsize=(6,4))
for m in methods:
    Ns_minus, _, e_minus = get_series(-0.07, m)
    Ns_plus,  _, e_plus  = get_series(+0.07, m)
    # assume same N sets
    Z = 0.14 / np.sqrt(e_minus**2 + e_plus**2)  # Δκ / combined σ
    plt.semilogx(Ns_minus, Z, marker=markers[m], color=colors[m], label=m)
plt.xlabel("N"); plt.ylabel("Z-separation (±0.07)")
plt.title("Separation power vs N")
plt.legend()
plt.show()

#comparing HESSE vs MINOS for unbinned but just for N=10k
print("Unbinned MINOS vs HESSE, N=10k:")
for kt in k_vals:
    th = datasets[(kt, 10_000)]
    k_u, e_u, lo_u, hi_u = fit_unbinned(th)
    print(f"  κ_true={kt:+.2f}: κ̂={k_u:+.5f}, HESSE={e_u:.5f}, MINOS=(-{abs(lo_u):.
```

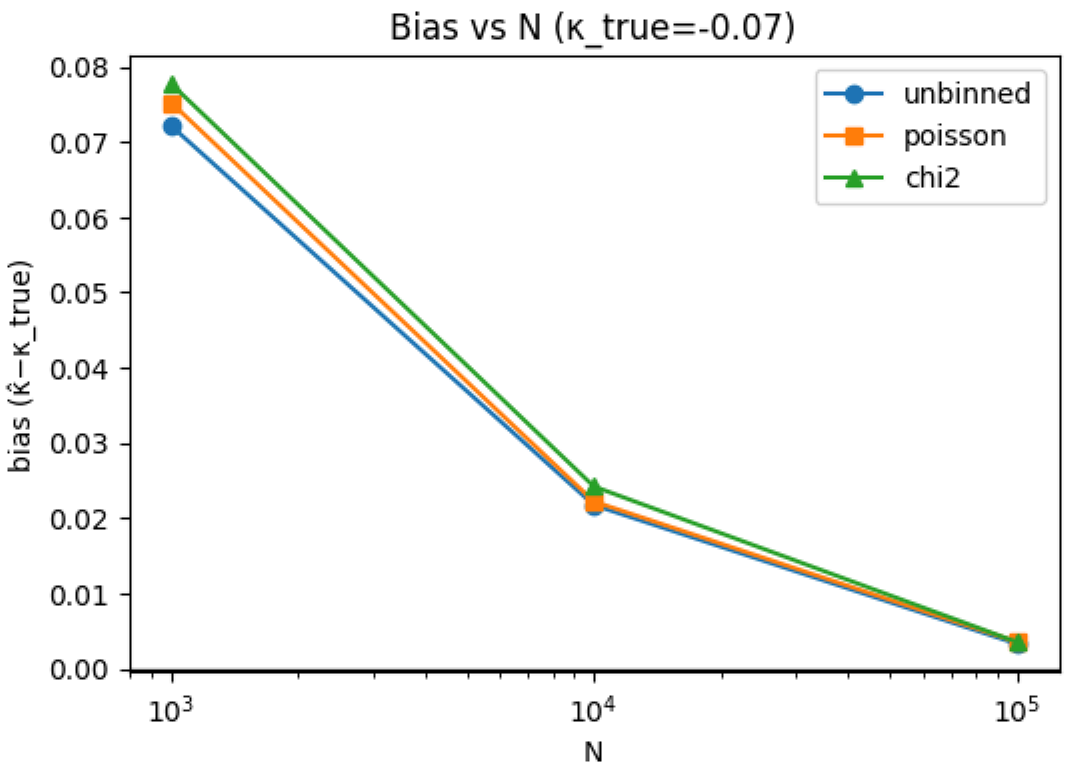


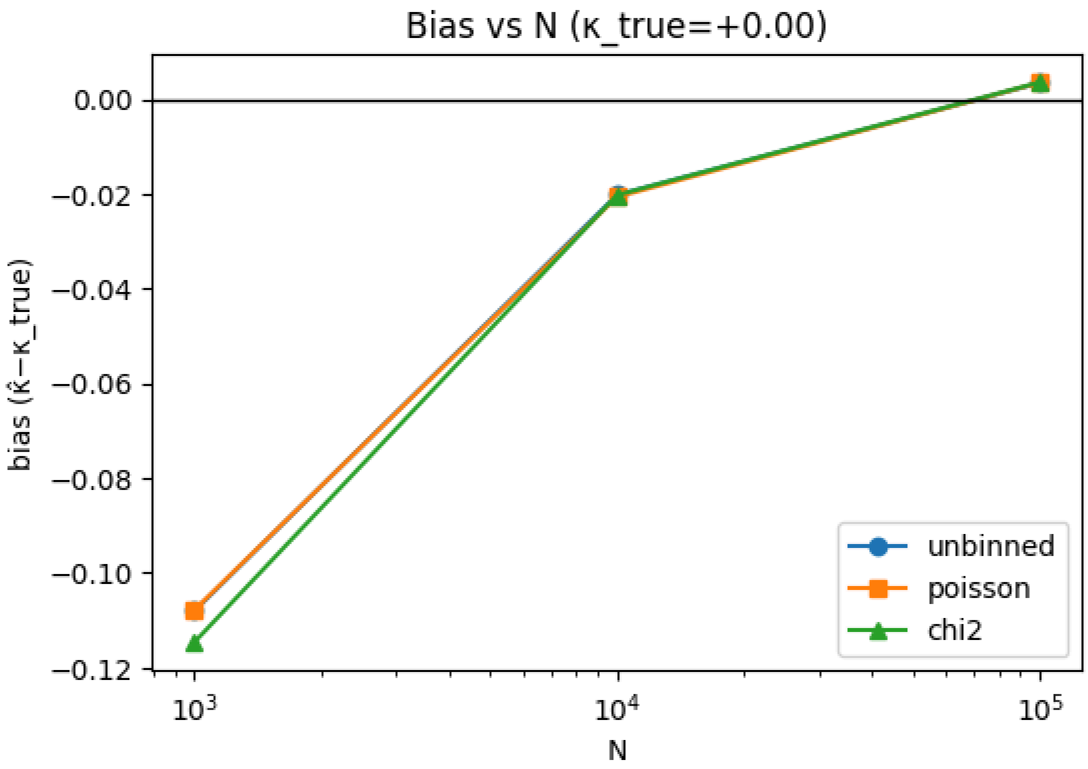


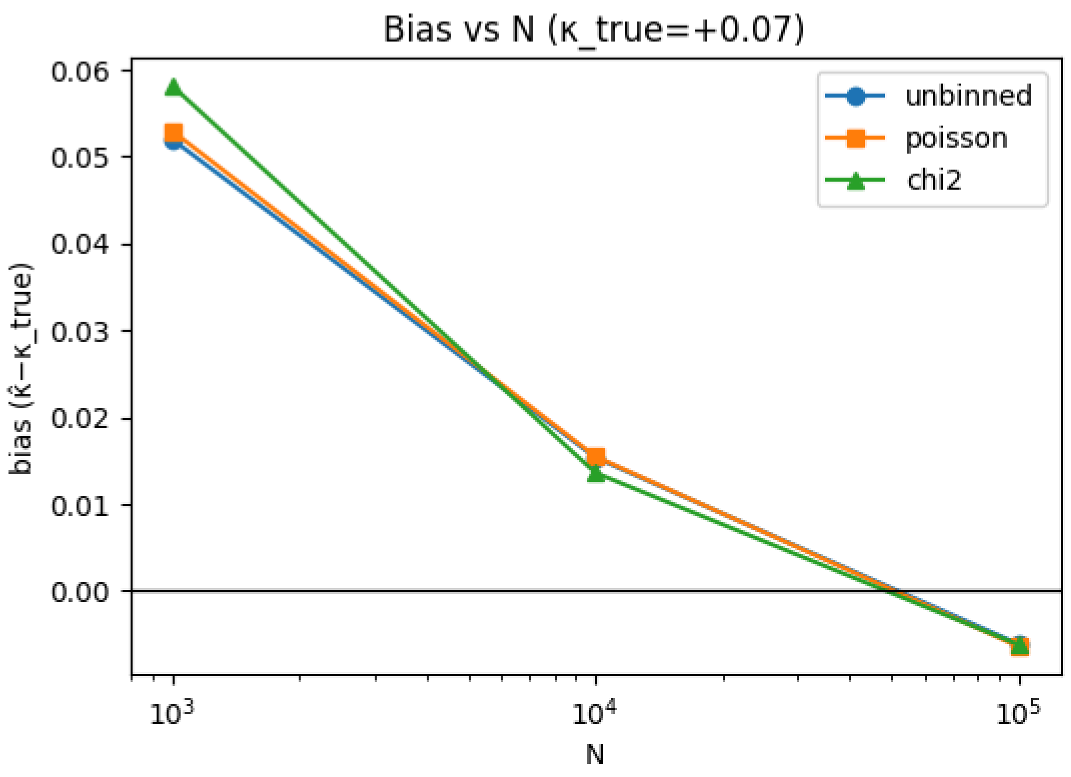
Uncertainty vs N (κ_true=+0.00)
unbinned
poisson
chi2
~1/√N
σ(κ̂)
10^{-2}
10^3
10^4
10^5
N

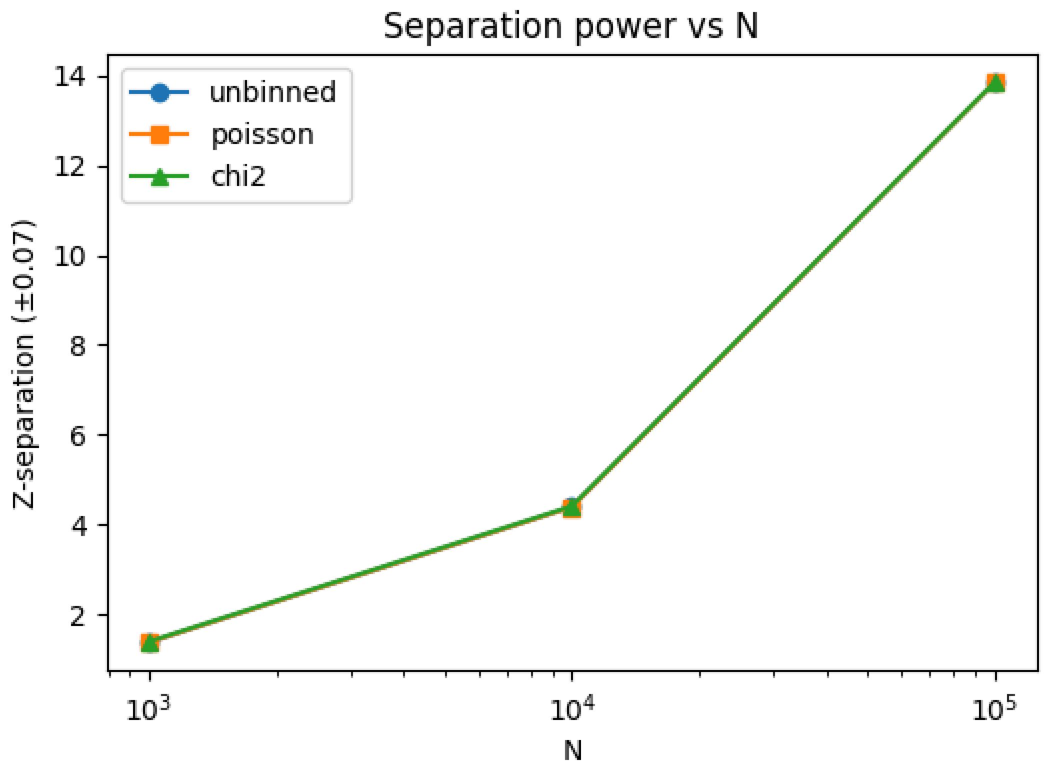


Uncertainty vs N (κ_true=+0.07)
unbinned
poisson
chi2
~1/√N
σ(κ̂)
10^{-2}
10^3
10^4
10^5
N


Bias vs N (κ_true=-0.07)
bias (κ̂−κ_true)
N
unbinned
poisson
chi2
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
10^3
10^4
10^5


Bias vs N (κ_true=+0.00)
bias (κ̂−κ_true)
N
unbinned
poisson
chi2
0.00
−0.02
−0.04
−0.06
−0.08
−0.10
−0.12
10^3
10^4
10^5





```
Unbinned MINOS vs HESSE, N=10k:
  κ_true=-0.07: κ̂=-0.04826, HESSE=0.02255, MINOS=(-0.02255, +0.02255)
  κ_true=+0.00: κ̂=-0.02021, HESSE=0.02264, MINOS=(-0.02263, +0.02264)
  κ_true=+0.07: κ̂=+0.08535, HESSE=0.02255, MINOS=(-0.02253, +0.02256)
```

Step 7 — Summary and conclusions

What I see from the 9 datasets and 3 fit methods:

- Precision follows ~1/√N: σ ≈ 0.0717 → 0.0226 → 0.00715 (for N=1k, 10k, 100k).
- Bias is small and shrinks with N:
- All the methods we used provide consistent results. χ^2 gives slightly smaller σ at low N Poisson and unbinned are more reliable when we have just a few bins. Unbinned seems to be the way to go.

Recommendation:

- Use unbinned MLE as a good baseline.
- χ^2 is fine with ≥40 bins but not at lower bins

In [18]:
```
# Step 7 — Summary, just doing what we did on step 6 again for everything to see at

import numpy as np
PI = np.pi
nbins = 40
edges = np.linspace(0.0, PI, nbins + 1)

def expected_counts(kappa, edges, N):
    g = 1.0 + np.cos(theta)**2 + kappa * np.cos(theta)
    G = cumulative_trapezoid(g, theta, initial=0.0)
    total = G[-1]
    G_edges = np.interp(edges, theta, G)
    probs = np.diff(G_edges) / total
    return N * probs

def make_nll_binned(n_obs, edges):
    N = int(np.sum(n_obs))
    def nll(kappa):
        mu = expected_counts(kappa, edges, N)
        if np.any(mu <= 0.0):
            return np.inf
        return np.sum(mu - n_obs * np.log(mu))
    return nll

def make_chi2(n_obs, edges):
    N = int(np.sum(n_obs))
    def chi2(kappa):
        mu = expected_counts(kappa, edges, N)
        mu = np.clip(mu, 1e-9, None)
        return np.sum((n_obs - mu)**2 / mu)
    return chi2

def fit_unbinned(theta_data):
    cos_th = np.cos(theta_data)
    def nll(kappa):
        g = 1.0 + cos_th**2 + kappa * cos_th
        if np.any(g <= 0.0):
            return np.inf
        return -np.sum(np.log(g))
    m = Minuit(nll, kappa=0.0)
    m.limits["kappa"] = (-1.9, 1.9)
    m.errordef = Minuit.LIKELIHOOD
    m.migrad(); m.hesse()
    return m.values["kappa"], m.errors["kappa"]

records = []
for (k_true, N_sel), theta_data in sorted(datasets.items()):
    # unbinned
    k_u, e_u = fit_unbinned(theta_data)
    # poisson-binned
    n_obs, _ = np.histogram(theta_data, bins=edges)
    nll = make_nll_binned(n_obs, edges)
    mp = Minuit(nll, kappa=0.0); mp.limits["kappa"]=(-1.9,1.9); mp.errordef=Minuit.
    mp.migrad(); mp.hesse()
    k_p, e_p = mp.values["kappa"], mp.errors["kappa"]
    # chi2
    chi2 = make_chi2(n_obs, edges)
    mc = Minuit(chi2, kappa=0.0); mc.limits["kappa"]=(-1.9,1.9); mc.errordef=Minuit
    mc.migrad(); mc.hesse()
    k_c, e_c = mc.values["kappa"], mc.errors["kappa"]

    records.append((k_true, N_sel, "unbinned", k_u, e_u))
    records.append((k_true, N_sel, "poisson",  k_p, e_p))
    records.append((k_true, N_sel, "chi2",     k_c, e_c))

methods = ["unbinned", "poisson", "chi2"]
Ns_all = sorted({N for _, N, _, _, _ in records})

def subset(method, N):
    return [(k_true, k_hat, err) for (k_true, N_sel, m, k_hat, err) in records if m

print("Step 7 — Summary (aggregated across κ_true)")
for m in methods:
    print(f"\nMethod: {m}")
    for N in Ns_all:
        rows = subset(m, N)
        errs = np.array([e for _,_,e in rows])
        kh   = np.array([khat for _,khat,_ in rows])
        kts  = np.array([kt for kt,_,_ in rows])
        bias = kh - kts
        mean_sig = errs.mean()
        max_pull = np.max(np.abs(bias/errs))
        rmse = np.sqrt(np.mean(bias**2))
        print(f"  N={N:6d}: mean σ={mean_sig:.5f}, RMSE(bias)={rmse:.5f}, max |bias

# Separation for ±0.07
for m in methods:
    print(f"\nSeparation Z (κ=±0.07), method={m}")
    for N in Ns_all:
        e_minus = [e for (kt, N_sel, mm, _, e) in records if mm==m and N_sel==N and
        e_plus  = [e for (kt, N_sel, mm, _, e) in records if mm==m and N_sel==N and
        Z = 0.14/np.sqrt(e_minus**2 + e_plus**2)
        print(f"  N={N:6d}: Z={Z:.2f}")
```

```
Step 7 — Summary (aggregated across κ_true)

Method: unbinned
  N=  1000: mean σ=0.07174, RMSE(bias)=0.08077, max |bias|/σ=1.51
  N= 10000: mean σ=0.02258, RMSE(bias)=0.01930, max |bias|/σ=0.96
  N=100000: mean σ=0.00715, RMSE(bias)=0.00448, max |bias|/σ=0.86

Method: poisson
  N=  1000: mean σ=0.07175, RMSE(bias)=0.08183, max |bias|/σ=1.50
  N= 10000: mean σ=0.02258, RMSE(bias)=0.01966, max |bias|/σ=0.98
  N=100000: mean σ=0.00716, RMSE(bias)=0.00468, max |bias|/σ=0.90

Method: chi2
  N=  1000: mean σ=0.07092, RMSE(bias)=0.08672, max |bias|/σ=1.61
  N= 10000: mean σ=0.02250, RMSE(bias)=0.01984, max |bias|/σ=1.08
  N=100000: mean σ=0.00715, RMSE(bias)=0.00460, max |bias|/σ=0.87

Separation Z (κ=±0.07), method=unbinned
  N=  1000: Z=1.38
  N= 10000: Z=4.39
  N=100000: Z=13.84

Separation Z (κ=±0.07), method=poisson
  N=  1000: Z=1.38
  N= 10000: Z=4.39
  N=100000: Z=13.83

Separation Z (κ=±0.07), method=chi2
  N=  1000: Z=1.40
  N= 10000: Z=4.41
  N=100000: Z=13.85
```