setup and data

In [3]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [4]:
```
# We'll choose the parameters of our synthetic data.
# The outlier probability will be 80%:
true_frac = 0.8

# The linear model has unit slope and zero intercept:
true_params = [1.0, 0.0]

# The outliers are drawn from a Gaussian with zero mean and unit variance
true_outliers = [0.0, 1.0]
```

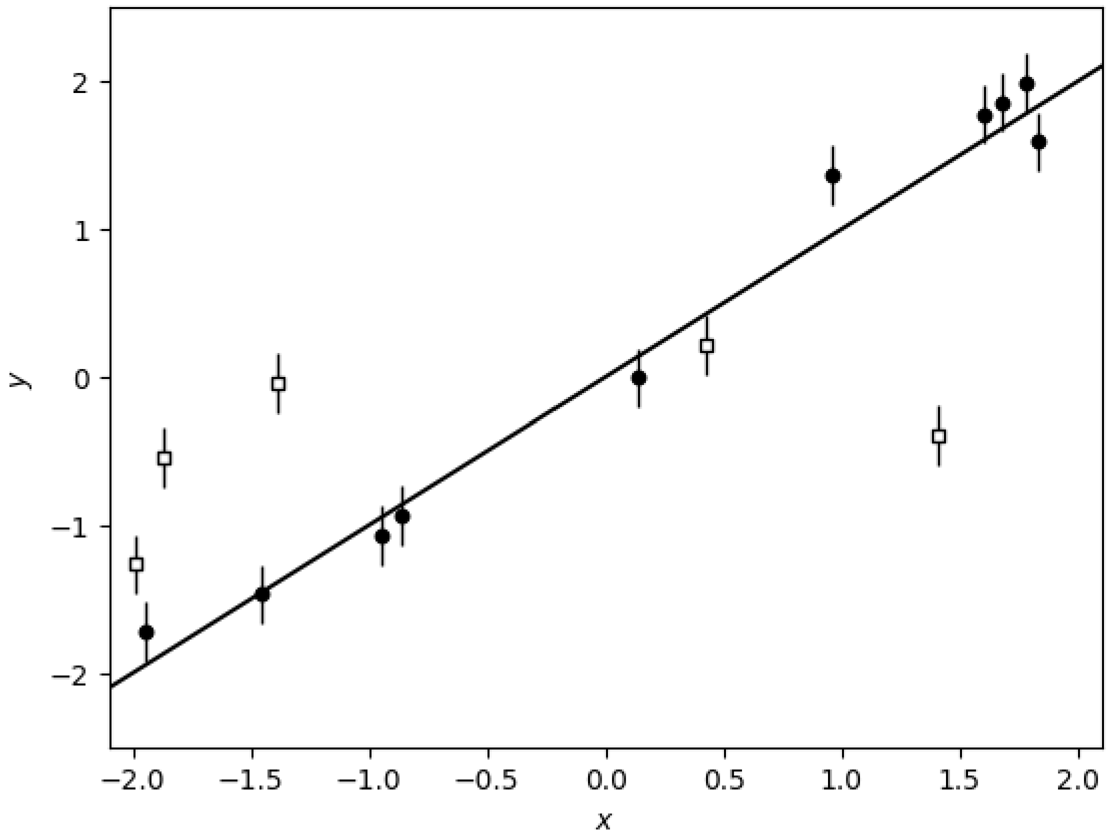
In [5]:
```
# For reproducibility, let's set the random number seed and generate the
np.random.seed(12)
x = np.sort(np.random.uniform(-2, 2, 15))
yerr = 0.2 * np.ones_like(x)
y = true_params[0] * x + true_params[1] + yerr * np.random.randn(len(x))

# Those points are all drawn from the correct model so let's replace some
# them with outliers.
m_bkg = np.random.rand(len(x)) > true_frac
y[m_bkg] = true_outliers[0]
y[m_bkg] += np.sqrt(true_outliers[1] + yerr[m_bkg] ** 2) * np.random.ranc

# Then save the *true* line.
x0 = np.linspace(-2.1, 2.1, 200)
y0 = np.dot(np.vander(x0, 2), true_params)
```

In [6]:
```
def plot_data():
    plt.errorbar(x, y, yerr=yerr, fmt=",k", ms=0, capsize=0, lw=1, zorder
    plt.scatter(x[m_bkg], y[m_bkg], marker="s", s=22, c="w", edgecolor="k
    plt.scatter(
        x[~m_bkg], y[~m_bkg], marker="o", s=22, c="k", zorder=1000, label
    )
    plt.plot(x0, y0, color="k", lw=1.5)
    plt.xlabel("$x$")
    plt.ylabel("$y$")
    plt.ylim(-2.5, 2.5)
    plt.xlim(-2.1, 2.1)
```

In [7]:
```
plot_data()
```



numpyro: simple model, with and without treatment of outliers

In [8]:
```
import jax
import jax.numpy as jnp

import numpyro
from numpyro import distributions as dist, infer
from numpyro_ext.distributions import MixtureGeneral
```

In [9]:
```
numpyro.set_host_device_count(2)
```

model formulation -->

In [10]:
```
def linear_model(x, yerr, y=None):
    # These are the parameters that we're fitting and we're required to d
    # priors using distributions from the numpyro.distributions module.
    theta = numpyro.sample("theta", dist.Uniform(-0.5 * jnp.pi, 0.5 * jnp
    b_perp = numpyro.sample("b_perp", dist.Normal(0, 1))

    # Transformed parameters (and other things!) can be tracked during sa
    # "deterministics" as follows:
    m = numpyro.deterministic("m", jnp.tan(theta))
    b = numpyro.deterministic("b", b_perp / jnp.cos(theta))

    # Then we specify the sampling distribution for the data, or the like
    # Here we're using a numpyro.plate to indicate that the data are inde
    # isn't actually necessary here and we could have equivalently omitte
    # the Normal distribution can already handle vector-valued inputs. Bu
    # get into the habit of using plates because some inference algorithm
    # can take advantage of knowing this structure.
```

```
    with numpyro.plate("data", len(x)):
        numpyro.sample("y", dist.Normal(m * x + b, yerr), obs=y)
```

In [11]:
```
# Using the model above, we can now sample from the posterior distributic
# U-Turn Sampler (NUTS).
sampler1 = infer.MCMC(
    infer.NUTS(linear_model),
    num_warmup=2000,
    num_samples=2000,
    num_chains=2,
    progress_bar=True,
)
%time sampler1.run(jax.random.PRNGKey(0), x, yerr, y=y)
```

```
  0%|          | 0/4000 [00:00<?, ?it/s]
  0%|          | 0/4000 [00:00<?, ?it/s]
CPU times: user 1.8 s, sys: 46.8 ms, total: 1.85 s
Wall time: 1.87 s
```

In [13]:
```
def linear_mixture_model(x, yerr, y=None):
    # Our "foreground" model is identical to the one we used previously:
    # parameterized by "theta" and "b_perp". Note that we don't wrap the
    # sampling distribution in a `numpyro.sample` here because we're goir
    # use it in the mixture distribution below.
    theta = numpyro.sample("theta", dist.Uniform(-0.5 * jnp.pi, 0.5 * jnp
    b_perp = numpyro.sample("b_perp", dist.Normal(0.0, 1.0))
    m = numpyro.deterministic("m", jnp.tan(theta))
    b = numpyro.deterministic("b", b_perp / jnp.cos(theta))
    fg_dist = dist.Normal(m * x + b, yerr)

    # Our outlier model is a Gaussian where we're fitting for the zero ar
    # standard deviation.
    bg_mean = numpyro.sample("bg_mean", dist.Normal(0.0, 1.0))
    bg_sigma = numpyro.sample("bg_sigma", dist.HalfNormal(3.0))
    bg_dist = dist.Normal(bg_mean, jnp.sqrt(bg_sigma**2 + yerr**2))

    # We use a `Catagorical` distribution to define the outlier probabili
    # fit for the parameter `Q` which specifies the probability that any
    # individual point is a member of the foreground model. Therefore, th
    # "prior" outlier probability is `1 - Q`.
    Q = numpyro.sample("Q", dist.Uniform(0.0, 1.0))
    mix = dist.Categorical(probs=jnp.array([Q, 1.0 - Q]))

    # As with the previous model, the use of a `plate` here is optional,
    # let's do it anyways.
    with numpyro.plate("data", len(x)):
        # The `numpyro.distributions` module doesn't yet implement a mixt
        # distribution that is flexible enough for our use cases, so we'l
        # one that I implemented in the `numpyro-ext` package. (This is s
        # a lie: the `MixtureSameFamily` distribution in NumPyro _would_
        # here, but not in our next example, so bear with me!)
        numpyro.sample("obs", MixtureGeneral(mix, [fg_dist, bg_dist]), ob
```

In [14]:
```
# Using the model above, we can now sample from the posterior distributic
# U-Turn Sampler (NUTS).
sampler2 = infer.MCMC(
    infer.NUTS(linear_mixture_model),
    num_warmup=2000,
    num_samples=2000,
    num_chains=2,
```

```
    progress_bar=True,
)
%time sampler2.run(jax.random.PRNGKey(3), x, yerr, y=y)
```

```
  0%|          | 0/4000 [00:00<?, ?it/s]
  0%|          | 0/4000 [00:00<?, ?it/s]
CPU times: user 3.01 s, sys: 46.4 ms, total: 3.05 s
Wall time: 3.01 s
```

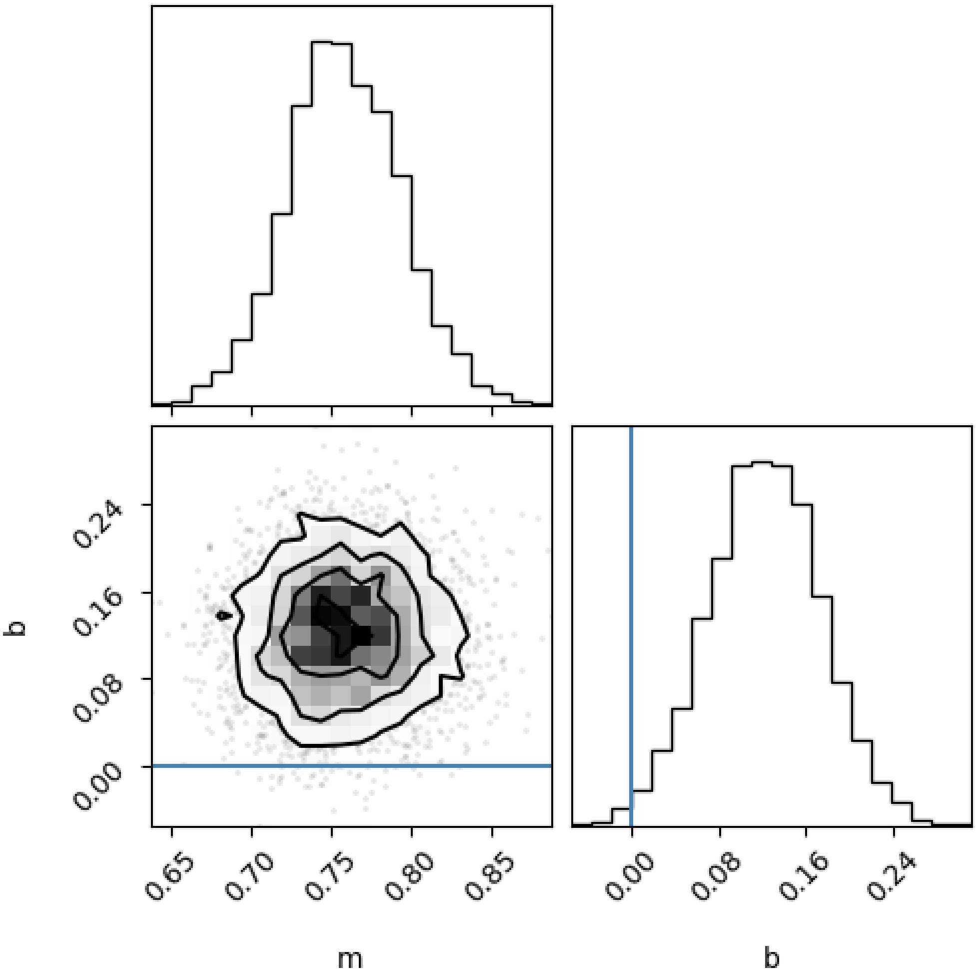
model evaluation -->

In [15]:
```
import arviz as az
import corner
```

In [16]:
```
inf_data1 = az.from_numpyro(sampler1)
corner.corner(inf_data1, var_names=["m", "b"], truths=true_params);
az.summary(inf_data1)
```

Out[16]:

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tai
b	0.122	0.052	0.018	0.214	0.001	0.001	3891.0	2685.(
b_perp	0.098	0.041	0.013	0.169	0.001	0.000	3894.0	2742.(
m	0.758	0.035	0.691	0.826	0.001	0.000	3461.0	2719.(
theta	0.648	0.023	0.608	0.693	0.000	0.000	3461.0	2719.(

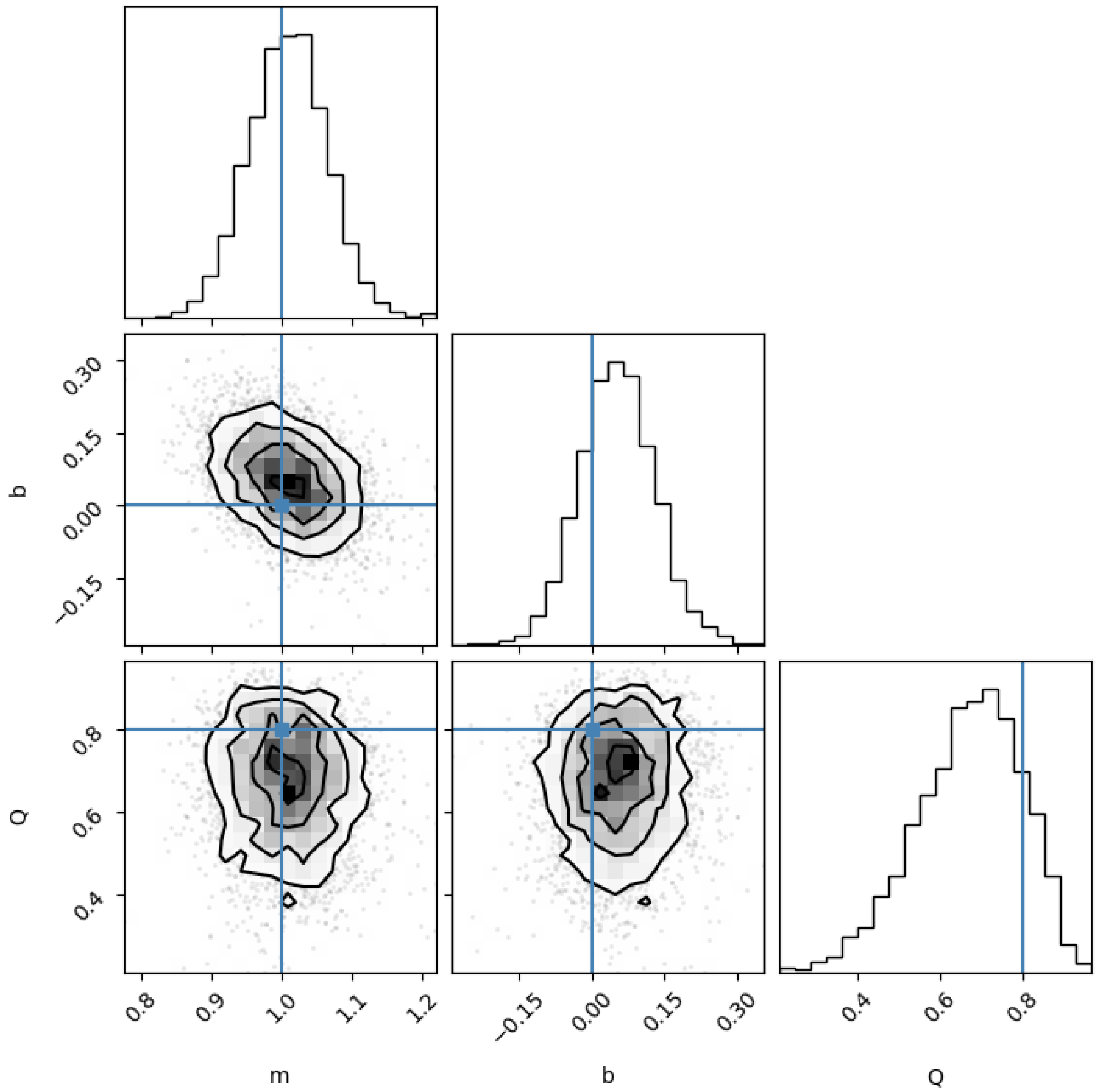


In [17]:
```
inf_data2 = az.from_numpyro(sampler2)
corner.corner(
```

```
    inf_data2,
    var_names=["m", "b", "Q"],
    truths={
        "m": true_params[0],
        "b": true_params[1],
        "Q": true_frac,
    },
);
az.summary(inf_data2)
```

Out[17]:

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_
Q	0.668	0.127	0.427	0.886	0.002	0.002	3513.0	29
b	0.052	0.078	-0.097	0.198	0.001	0.001	2954.0	26
b_perp	0.037	0.056	-0.069	0.140	0.001	0.001	2912.0	27
bg_mean	-0.412	0.403	-1.170	0.341	0.009	0.006	2264.0	22
bg_sigma	0.776	0.512	0.021	1.657	0.011	0.008	2195.0	15
m	1.008	0.056	0.897	1.106	0.001	0.001	2738.0	25
theta	0.789	0.028	0.738	0.842	0.001	0.000	2738.0	25



pymc: redo same models, with and without numpyro api

In [18]:
```
import pymc as pm
import numpy as np

from pymc import Uniform, Normal, HalfNormal, Model, Mixture, Categorical
```

model formulation -->

simple model, no outlier handling

In [19]:
```
with Model() as model:
    theta = Uniform("theta", -0.5*np.pi, 0.5*np.pi)
    b_perp = Normal("b_perp", 0, sigma=1)

    # transformed params
    m = pm.Deterministic("m", np.tan(theta))
    b = pm.Deterministic("b", b_perp / np.cos(theta))

    # likelihood
    likelihood = Normal("y", mu=m*x+b, sigma=yerr, observed=y)

    # inference
    %time idata1 = sample(2000, chains=2)
```

100.00% [6000/6000 00:00<00:00
Sampling 2 chains, 0 divergences]

```
CPU times: user 1.46 s, sys: 179 ms, total: 1.64 s
Wall time: 2.39 s
```

... and now with jax

In [20]:
```
with Model() as model:
    theta = Uniform("theta", -0.5*np.pi, 0.5*np.pi)
    b_perp = Normal("b_perp", 0, sigma=1)

    # transformed params
    m = pm.Deterministic("m", np.tan(theta))
    b = pm.Deterministic("b", b_perp / np.cos(theta))

    # likelihood
    likelihood = Normal("y", mu=m*x+b, sigma=yerr, observed=y)

    # inference
    %time idata2 = sampling_jax.sample_numpyro_nuts(2000, chains=2)
```

```
Compiling...
Compilation time =  0:00:00.280433
Sampling...
  0%|          | 0/3000 [00:00<?, ?it/s]
  0%|          | 0/3000 [00:00<?, ?it/s]
```

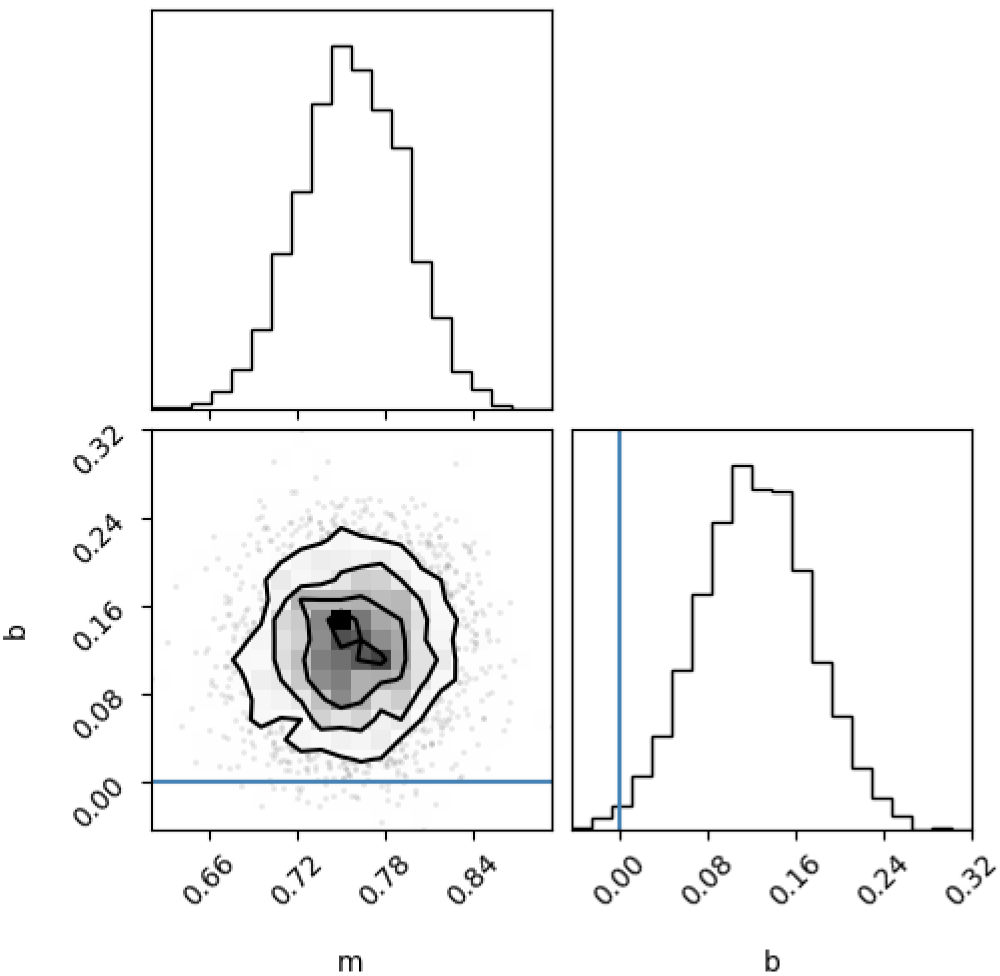
```
Sampling time =  0:00:01.486291
Transforming variables...
Transformation time =  0:00:00.079975
CPU times: user 1.82 s, sys: 54 ms, total: 1.87 s
Wall time: 1.87 s
```

model evaluation -->

In [21]:
```
corner.corner(idata1, var_names=["m", "b"], truths=true_params);
az.summary(idata1, var_names=["b_perp", "theta", "m", "b"])
```

Out[21]:

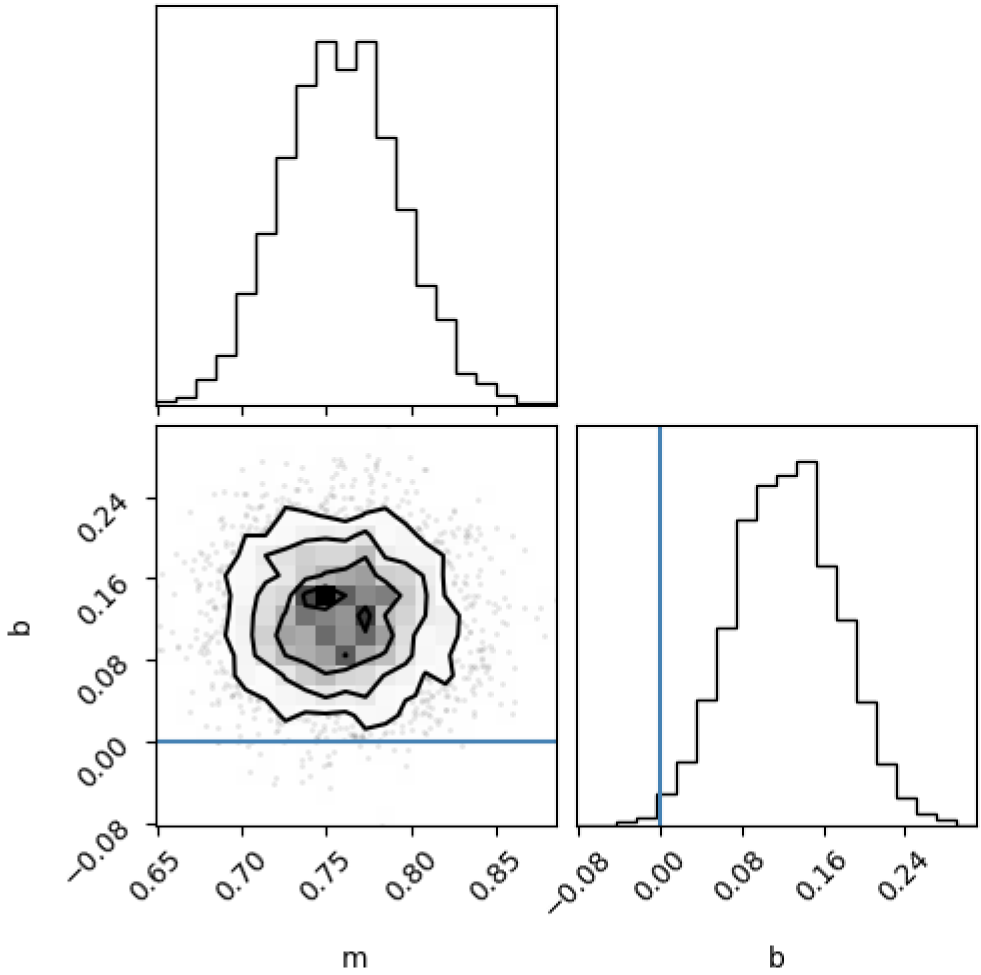
	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tai
b_perp	0.098	0.041	0.020	0.174	0.001	0.000	4486.0	2971.(
theta	0.648	0.023	0.606	0.690	0.000	0.000	4471.0	3046.(
m	0.757	0.036	0.693	0.825	0.001	0.000	4471.0	3046.(
b	0.123	0.051	0.028	0.222	0.001	0.001	4491.0	2992.(



In [22]:
```
corner.corner(idata2, var_names=["m", "b"], truths=true_params);
az.summary(idata2, var_names=["b_perp", "theta", "m", "b"])
```

Out[22]:

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tai
b_perp	0.098	0.042	0.021	0.174	0.001	0.001	3133.0	2385.(
theta	0.648	0.022	0.608	0.690	0.000	0.000	3119.0	2506.(
m	0.758	0.035	0.696	0.825	0.001	0.000	3119.0	2506.(
b	0.124	0.052	0.025	0.217	0.001	0.001	3130.0	2491.(



model formulation -->

model with some outlier handling

In [23]:

```
with Model() as model:
    theta = Uniform("theta", -0.5*np.pi, 0.5*np.pi)
    b_perp = Normal("b_perp", 0.0, sigma=1.0)
    m = pm.Deterministic("m", pm.math.tan(theta))
    b = pm.Deterministic("b", b_perp / pm.math.cos(theta))
    fg_dist = Normal.dist(m*x+b, yerr)

    bg_mean = Normal("bg_mean", 0.0, 1.0)
    bg_sigma = HalfNormal("bg_sigma", 3.0)
    bg_dist = Normal.dist(bg_mean, pm.math.sqrt(bg_sigma**2 + yerr**2))

    Q = Uniform("Q", 0.0, 1.0)
    mix = pm.Dirichlet("mix", pm.math.stack([Q, 1.0-Q]))

    # likelihood
```

```
    likelihood = Mixture("likelihood", w=mix, comp_dists=[fg_dist, bg_dis

    # inference
    %time idata3 = sample(2000, chains=2)
```

100.00% [6000/6000 00:03<00:00 Sampling 2 chains, 4 divergences]

```
CPU times: user 4.83 s, sys: 314 ms, total: 5.14 s
Wall time: 8.78 s
```

In [24]:

```
with Model() as model:
    theta = Uniform("theta", -0.5*np.pi, 0.5*np.pi)
    b_perp = Normal("b_perp", 0.0, sigma=1.0)
    m = pm.Deterministic("m", pm.math.tan(theta))
    b = pm.Deterministic("b", b_perp / pm.math.cos(theta))
    fg_dist = Normal.dist(m*x+b, yerr)

    bg_mean = Normal("bg_mean", 0.0, 1.0)
    bg_sigma = HalfNormal("bg_sigma", 3.0)
    bg_dist = Normal.dist(bg_mean, pm.math.sqrt(bg_sigma**2 + yerr**2))

    Q = Uniform("Q", 0.0, 1.0)
    mix = pm.Dirichlet("mix", pm.math.stack([Q, 1.0-Q]))

    # likelihood
    likelihood = Mixture("likelihood", w=mix, comp_dists=[fg_dist, bg_dis

    # inference
    %time idata4 = sampling_jax.sample_numpyro_nuts(2000, chains=2)
```

```
Compiling...
Compilation time =  0:00:01.186800
Sampling...
  0%|          | 0/3000 [00:00<?, ?it/s]
  0%|          | 0/3000 [00:00<?, ?it/s]
Sampling time =  0:00:02.443143
Transforming variables...
Transformation time =  0:00:00.110047
CPU times: user 3.64 s, sys: 101 ms, total: 3.74 s
Wall time: 3.75 s
```

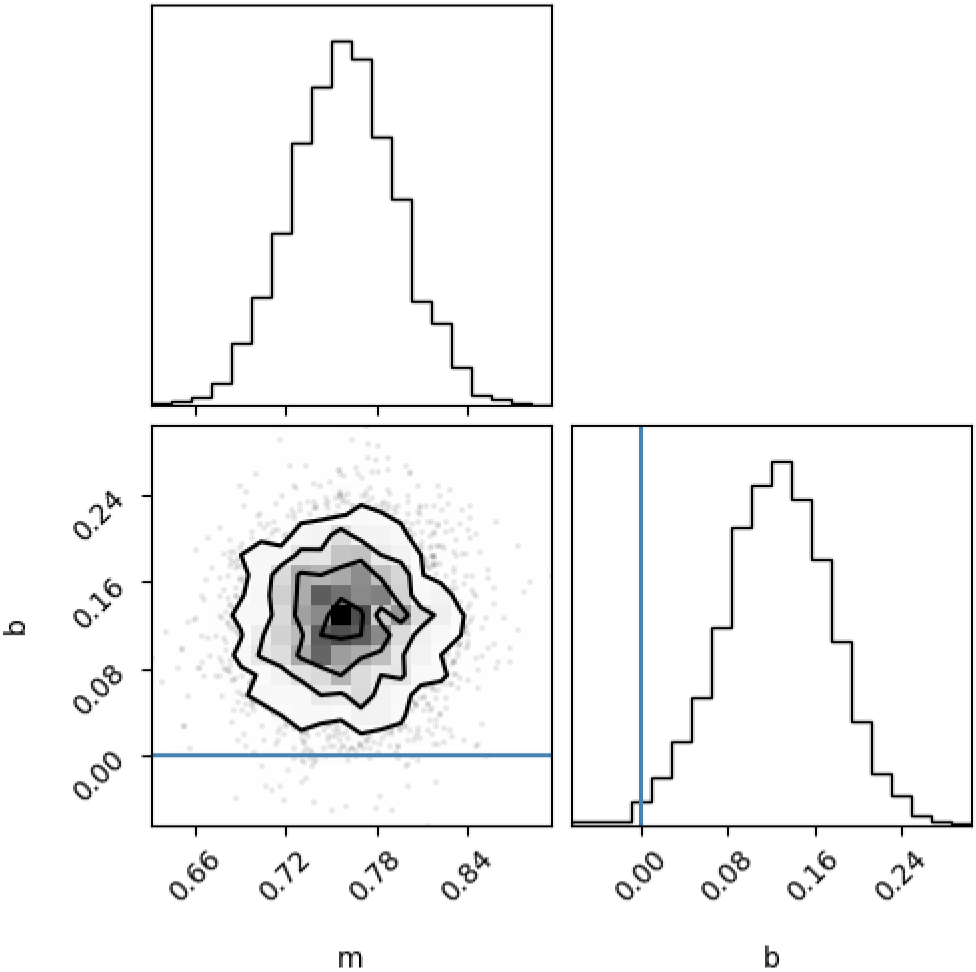
model evaluation -->

In [34]:

```
corner.corner(idata3, var_names=["m", "b"], truths=true_params);
az.summary(idata3, var_names=["b_perp", "theta", "m", "b"])
```

Out[34]:

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tai
b_perp	0.099	0.041	0.020	0.176	0.001	0.001	2783.0	2377.(
theta	0.649	0.023	0.605	0.690	0.000	0.000	3319.0	2452.(
m	0.759	0.036	0.692	0.825	0.001	0.000	3319.0	2452.(
b	0.125	0.051	0.024	0.220	0.001	0.001	2768.0	2622.(



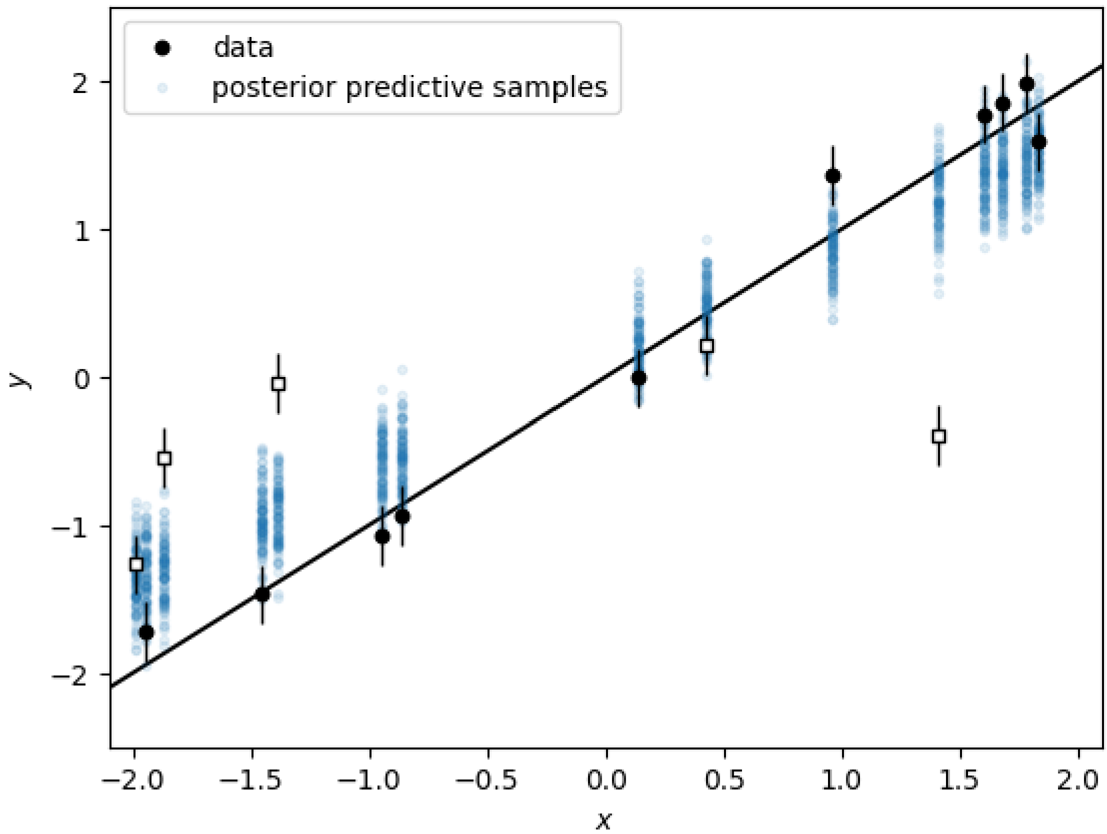
In [ ]:
```
corner.corner(idata4, var_names=["m", "b"], truths=true_params);
az.summary(idata4, var_names=["b_perp", "theta", "m", "b"])
```

---

simulate

In [15]:
```
post_pred_samples = infer.Predictive(linear_model, sampler.get_samples())(
    jax.random.PRNGKey(1), x, yerr
)
post_pred_y = post_pred_samples["y"]

plot_data()
label = "posterior predictive samples"
for n in np.random.default_rng(0).integers(len(post_pred_y), size=100):
    plt.plot(x, post_pred_y[n], ".", color="C0", alpha=0.1, label=label)
    label = None
plt.legend();
```


data
posterior predictive samples
2
1
0
−1
−2
y
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
x

```
In [ ]:
```