Making Gaussian Processes Useful

Bill Engels (PyMC Labs) and **Chris Fonnesbeck** (PyMC Labs)

Probabilistic Programming

Stochastic language "primitives"

Distribution over values:

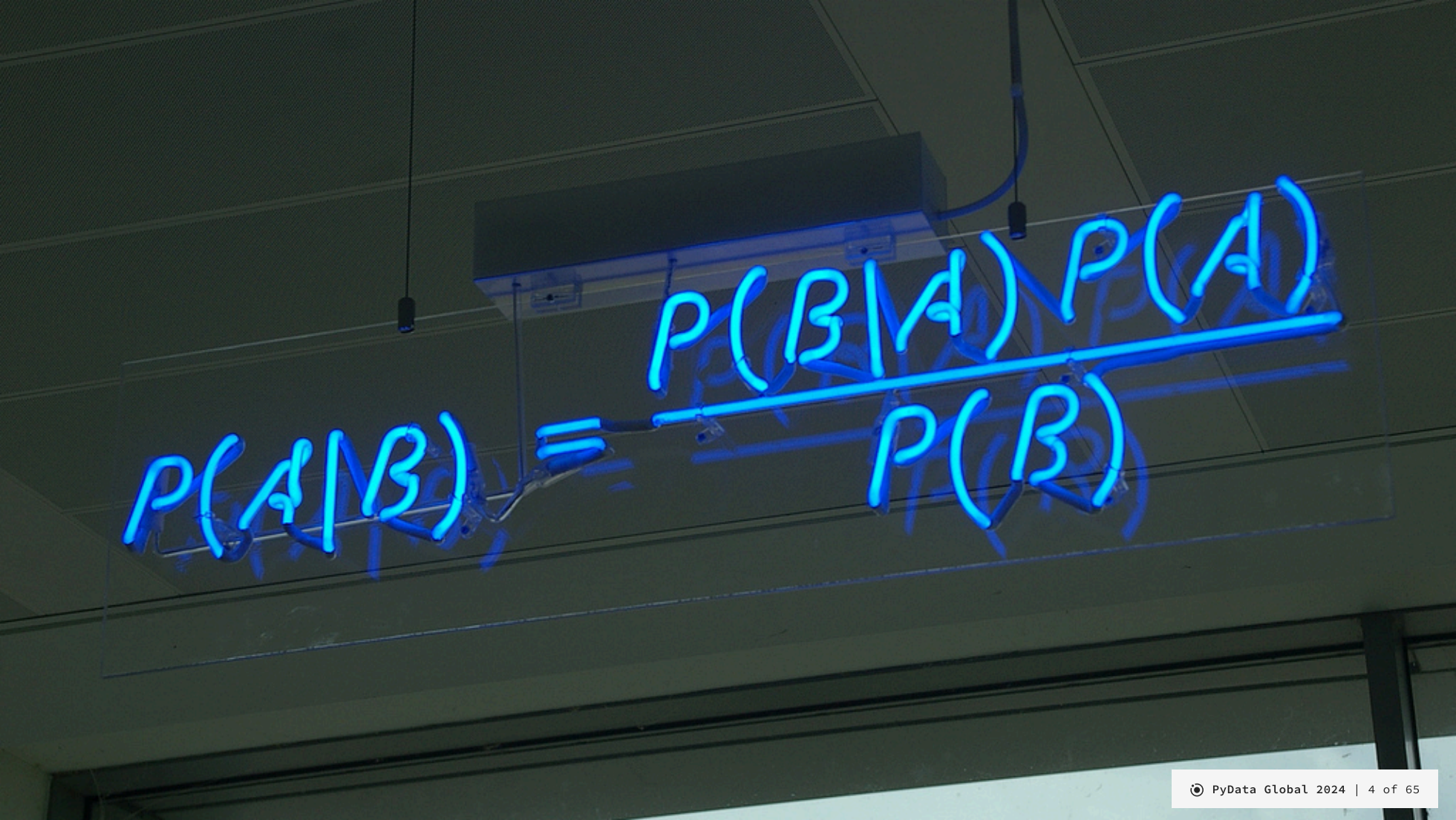
```
X ~ Normal(μ, σ)
x = X.random(n=100)
```

Distribution over functions:

```
Y ~ GaussianProcess(mean_func(x), cov_func(x))
y = Y.predict(x2)
```

Conditioning:

```
p ~ Beta(1, 1)
z ~ Bernoulli(p) # z|p
```


$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$

Bayesian Inference

How do we make a probability statement about θ given y?

Consider the **joint probability**

$$Pr(\theta, y)$$

Conditional Probability

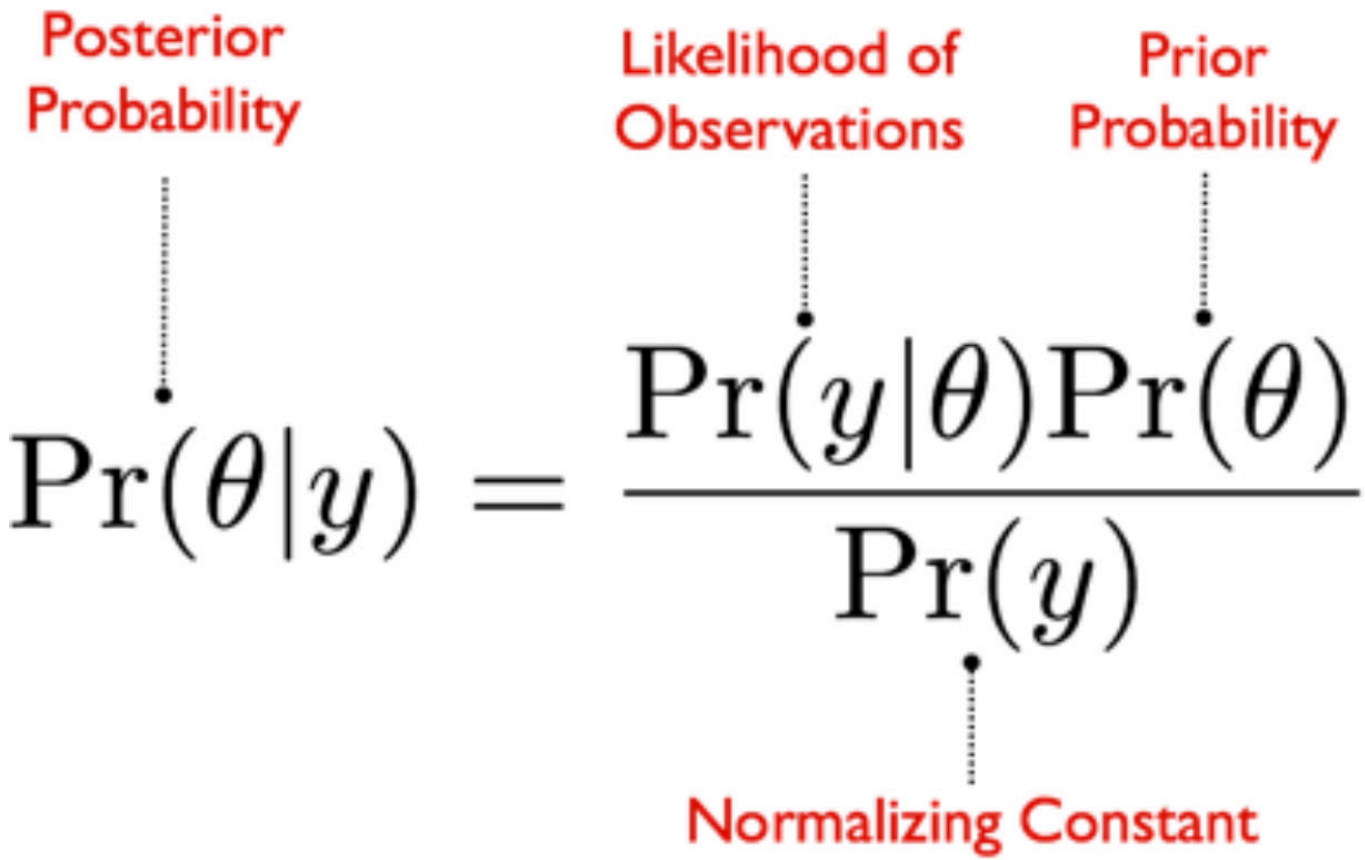
Let’s break this joint probability into two components:

$$Pr(\theta, y) = Pr(y|\theta)Pr(\theta)$$

Conditional Probability

Similarly, we can use the definition of conditional probability on $Pr(\theta, y)$:

$$Pr(\theta|y) = \frac{Pr(\theta, y)}{Pr(y)}$$


Posterior Probability
Likelihood of Observations
Prior Probability
$\Pr(\theta|y) = \frac{\Pr(y|\theta)\Pr(\theta)}{\Pr(y)}$
Normalizing Constant

Normalization constant

Denominator $Pr(y)$ is the probability of observing the data, **integrated** over all possible values of the parameters.

$$Pr(\theta|y) = \frac{Pr(y|\theta)Pr(\theta)}{\int_{\theta} Pr(y|\theta)Pr(\theta)d\theta}$$

Infer Values for Latent Variables

Calculate the posterior distribution

$$Pr(\theta|y) \propto Pr(y|\theta)Pr(\theta)$$

Stochastic program

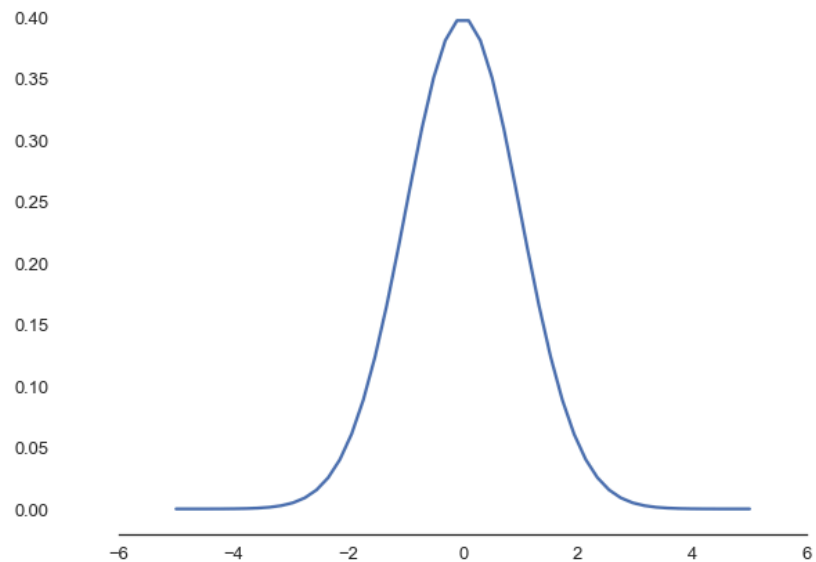
Joint distribution of latent variables and data

$$Pr(\theta, y) = Pr(y|\theta)Pr(\theta)$$

Prior distribution

Quantifies the **uncertainty** in latent variables

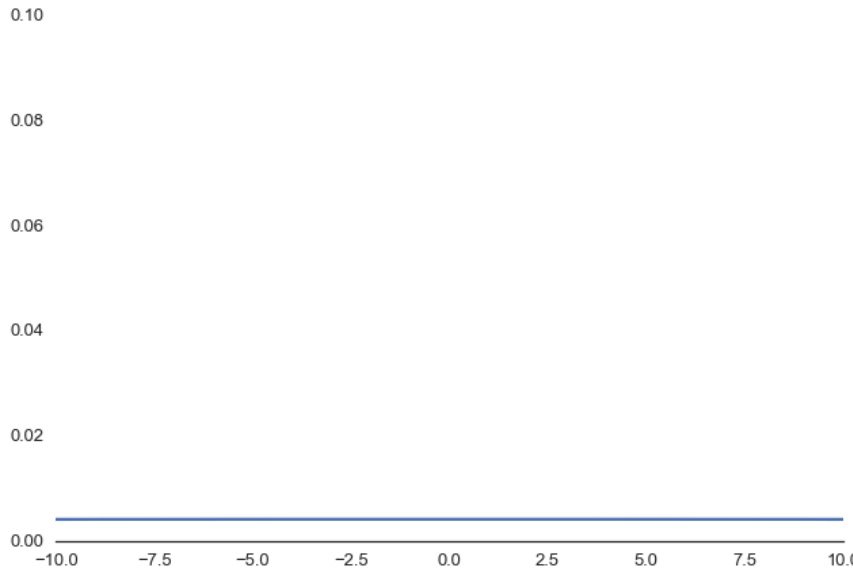
$$\theta \sim \text{Normal}(0, 1)$$



Prior distribution

Quantifies the **uncertainty** in latent variables

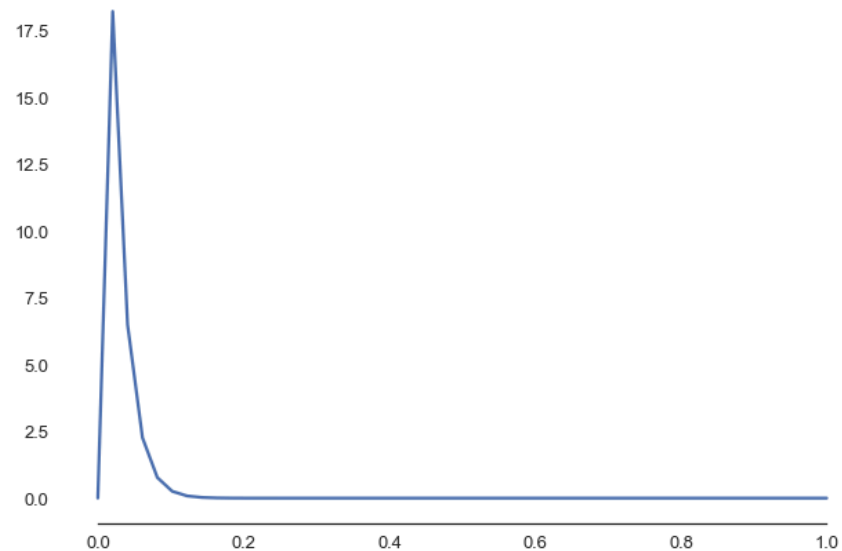
$$\theta \sim \text{Normal}(0, 100)$$



Prior distribution

Quantifies the **uncertainty** in latent variables

$$\theta \sim \text{Beta}(1, 50)$$



Likelihood function

Conditions our model on the observed data

$$Pr(y|\theta)$$

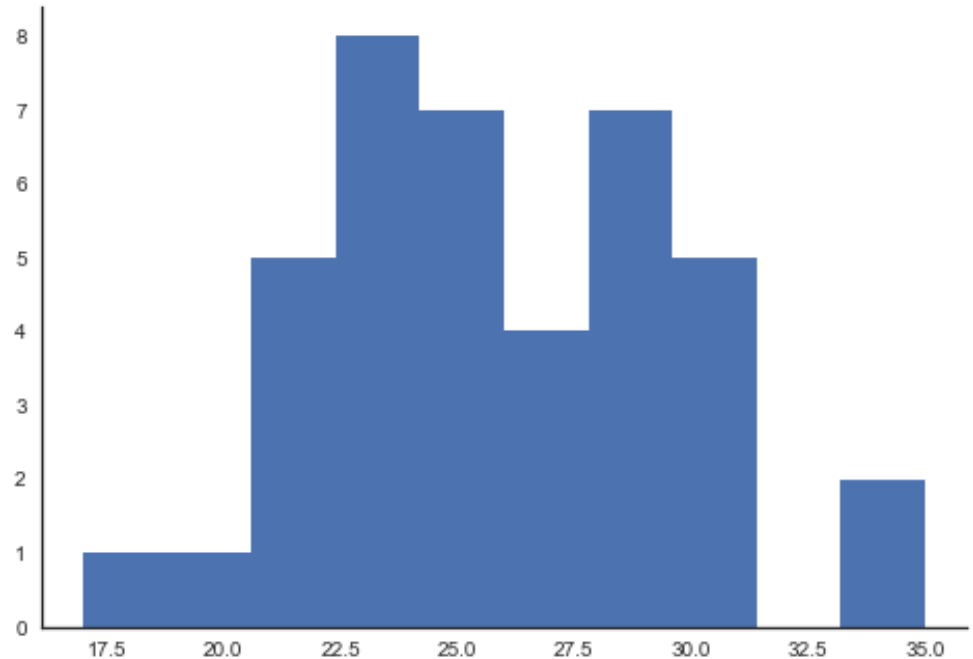
Likelihood function

Conditions our model on the observed data

$$x \sim \text{Normal}(\mu, \sigma^2)$$

$$x_K \sim \text{Binomial}(n_{AB}, p_H)$$

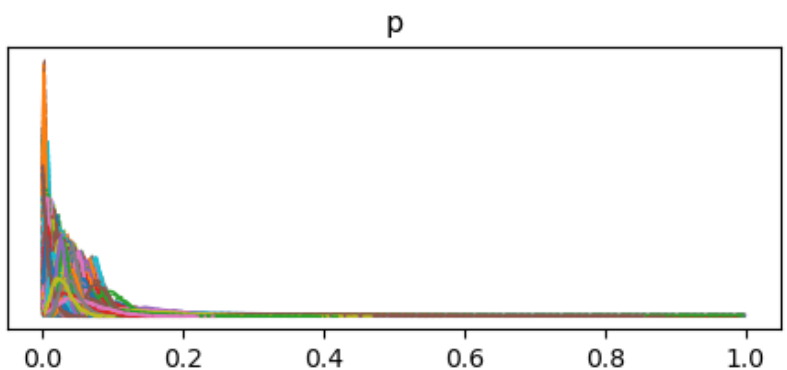
Models the distribution of x home runs observed from n plate appearances.

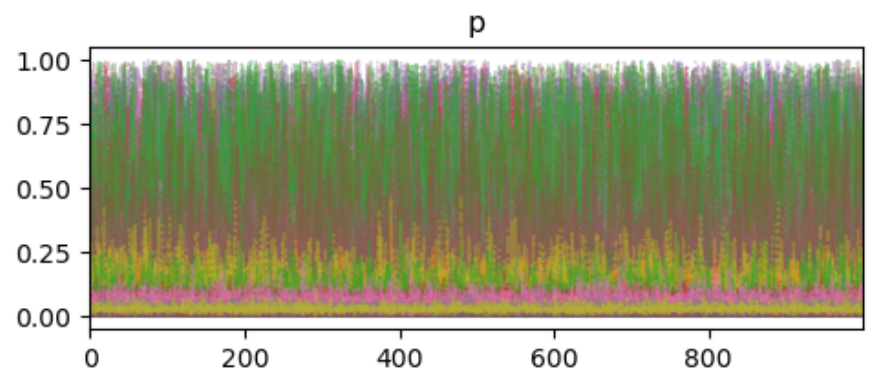


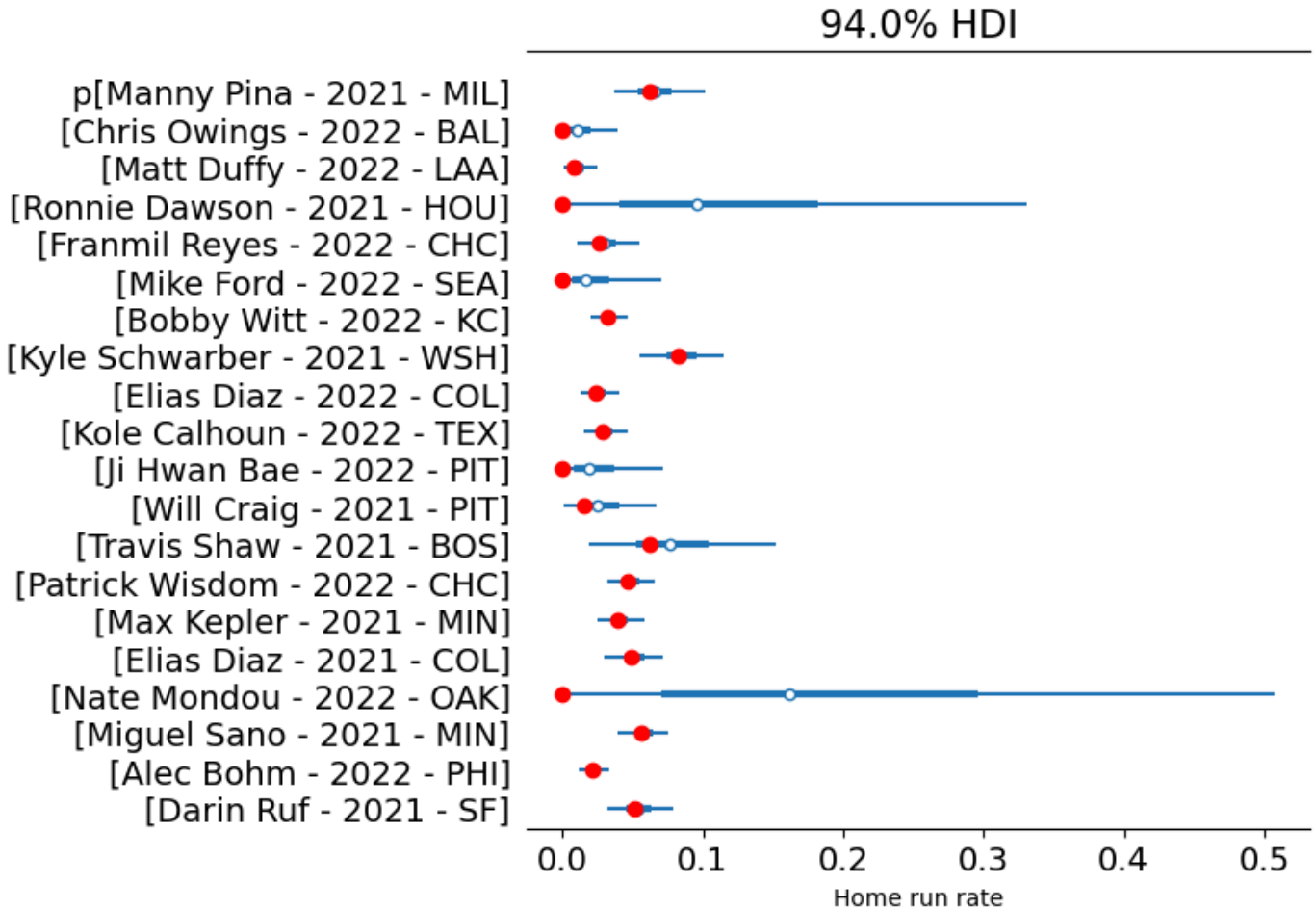
Name	Age	Team	Games	PA	H	HR
Mike Yastrzemski	32	San Francisco	105	379	77	15
Christian Yelich	31	Milwaukee	143	628	153	19
Juan Yepez	25	St. Louis	28	65	11	2
Masataka Yoshida	29	Boston	139	575	153	15
Jacob Young	23	Washington	33	121	27	0
Jared Young	27	Chicago	16	47	8	2
Seby Zavala	29	Arizona,Chicago	73	193	30	7
Mike Zunino	32	Cleveland	40	132	21	3

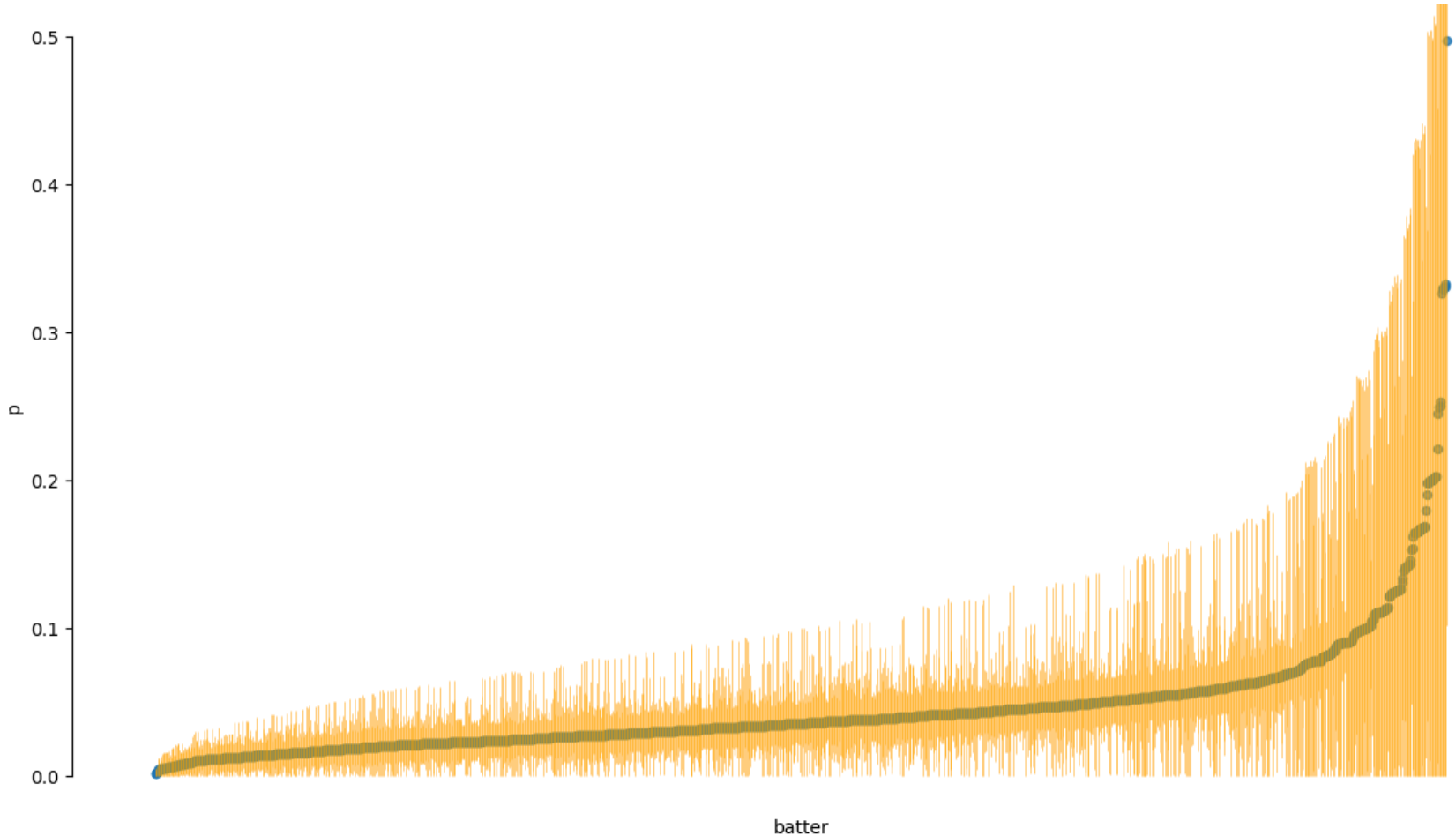
```
coords = {'player': batting_data.Name.values}
with pm.Model(coords=coords) as unpooled_model:

    p = pm.Uniform('p', 0, 1, dims='player')

    y = pm.Binomial('y', n=pa, p=p,
          observed=hr, dims='player')

    trace = pm.sample()
```

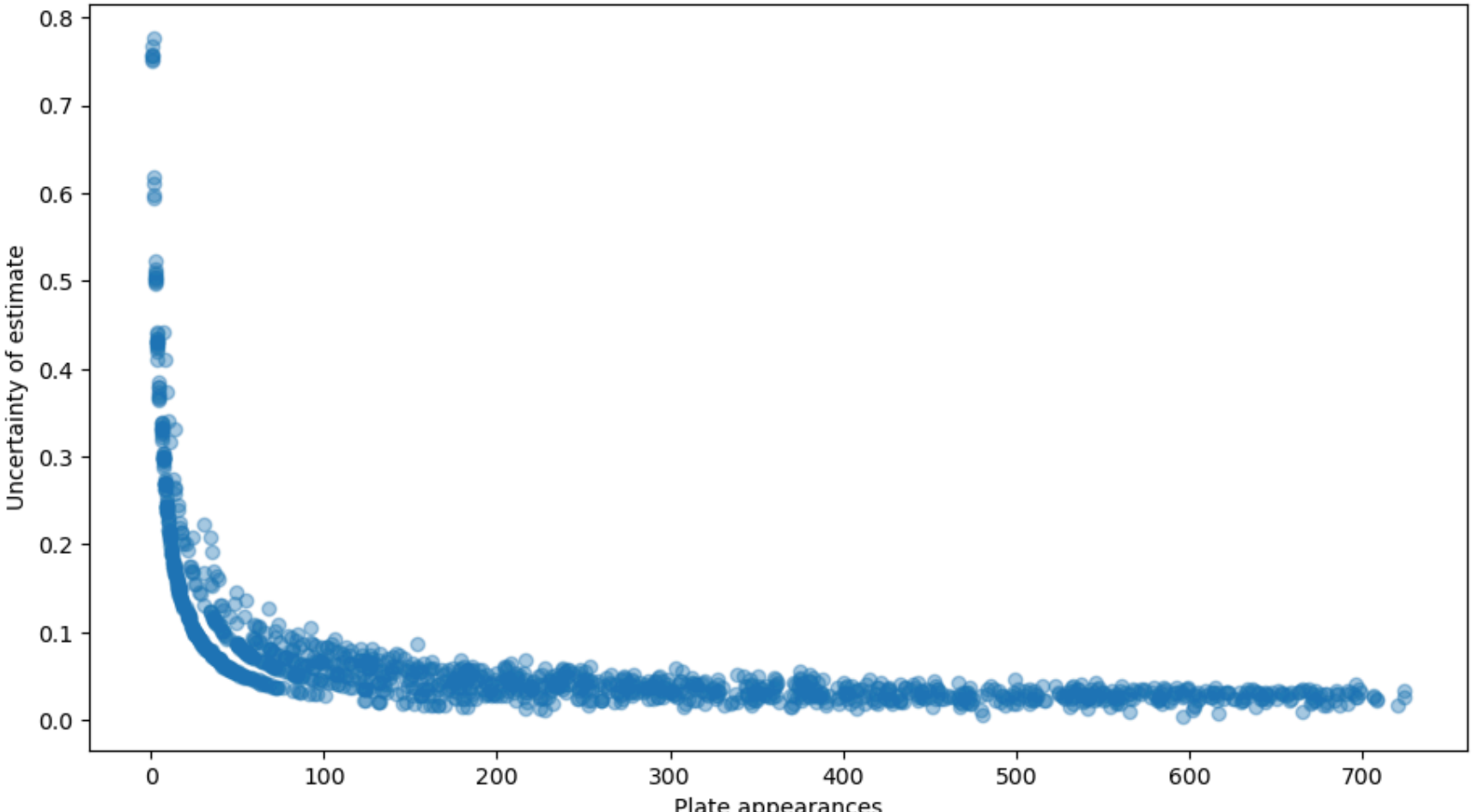
```
coords = {'player': batting_data.Name.values}
with pm.Model(coords=coords) as unpooled_model:

    p = pm.Uniform('p', 0, 1, dims='player')

    y = pm.Binomial('y', n=pa, p=p,
          observed=hr, dims='player')
```



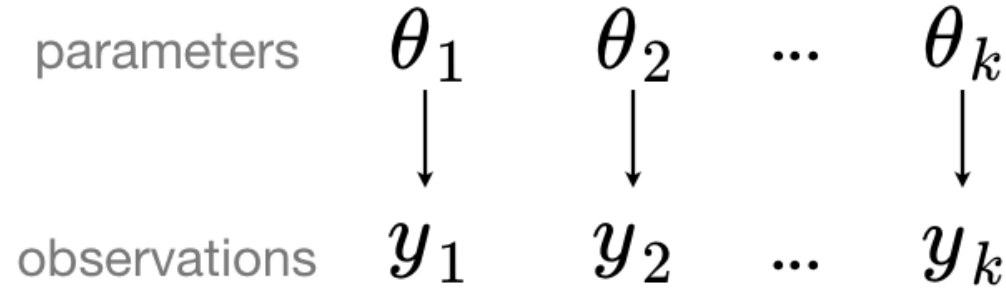



94.0% HDI
p[Manny Pina - 2021 - MIL]
[Chris Owings - 2022 - BAL]
[Matt Duffy - 2022 - LAA]
[Ronnie Dawson - 2021 - HOU]
[Franmil Reyes - 2022 - CHC]
[Mike Ford - 2022 - SEA]
[Bobby Witt - 2022 - KC]
[Kyle Schwarber - 2021 - WSH]
[Elias Diaz - 2022 - COL]
[Kole Calhoun - 2022 - TEX]
[Ji Hwan Bae - 2022 - PIT]
[Will Craig - 2021 - PIT]
[Travis Shaw - 2021 - BOS]
[Patrick Wisdom - 2022 - CHC]
[Max Kepler - 2021 - MIN]
[Elias Diaz - 2021 - COL]
[Nate Mondou - 2022 - OAK]
[Miguel Sano - 2021 - MIN]
[Alec Bohm - 2022 - PHI]
[Darin Ruf - 2021 - SF]
0.0
0.1
0.2
0.3
0.4
0.5
Home run rate

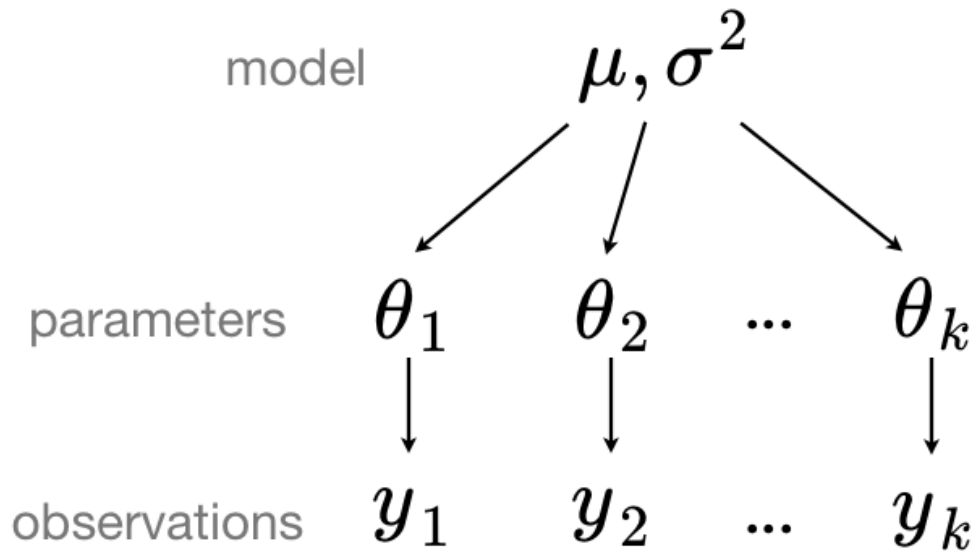

0.5
0.4
0.3
0.2
0.1
0.0
p
batter


0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Uncertainty of estimate
0
100
200
300
400
500
600
700
Plate appearances

Unpooled model

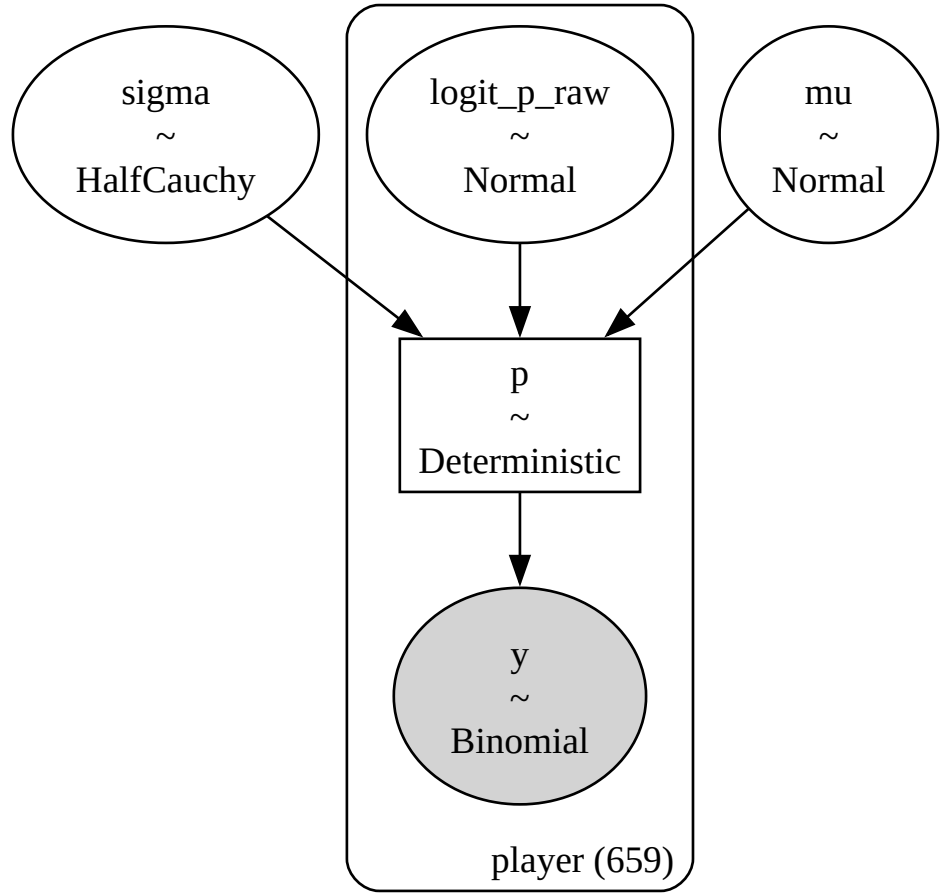


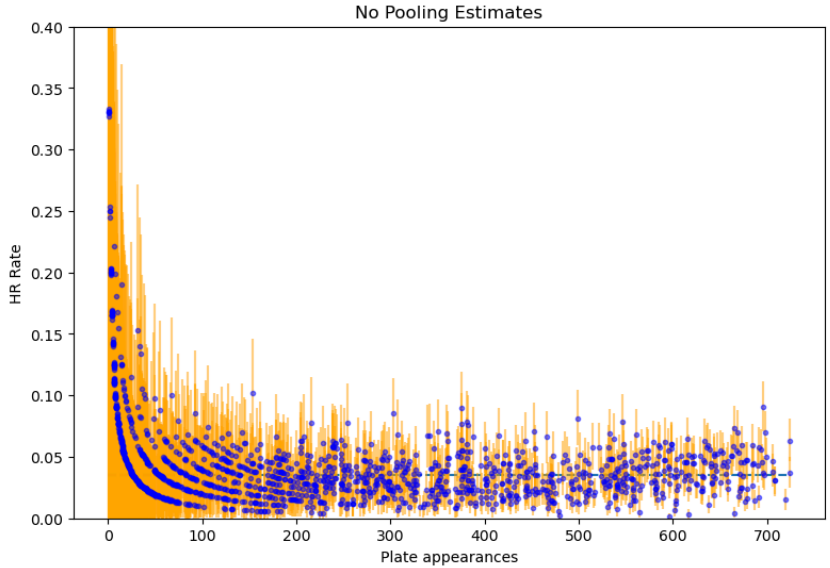
Partial pooling model

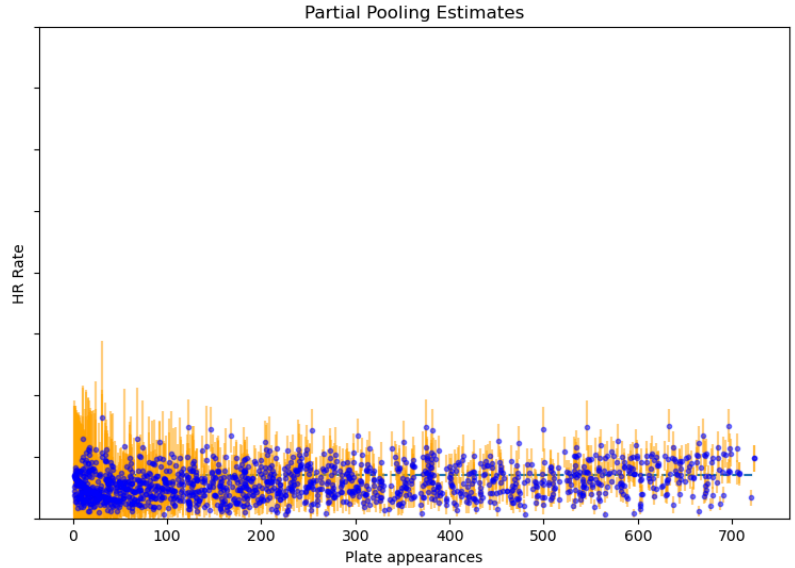


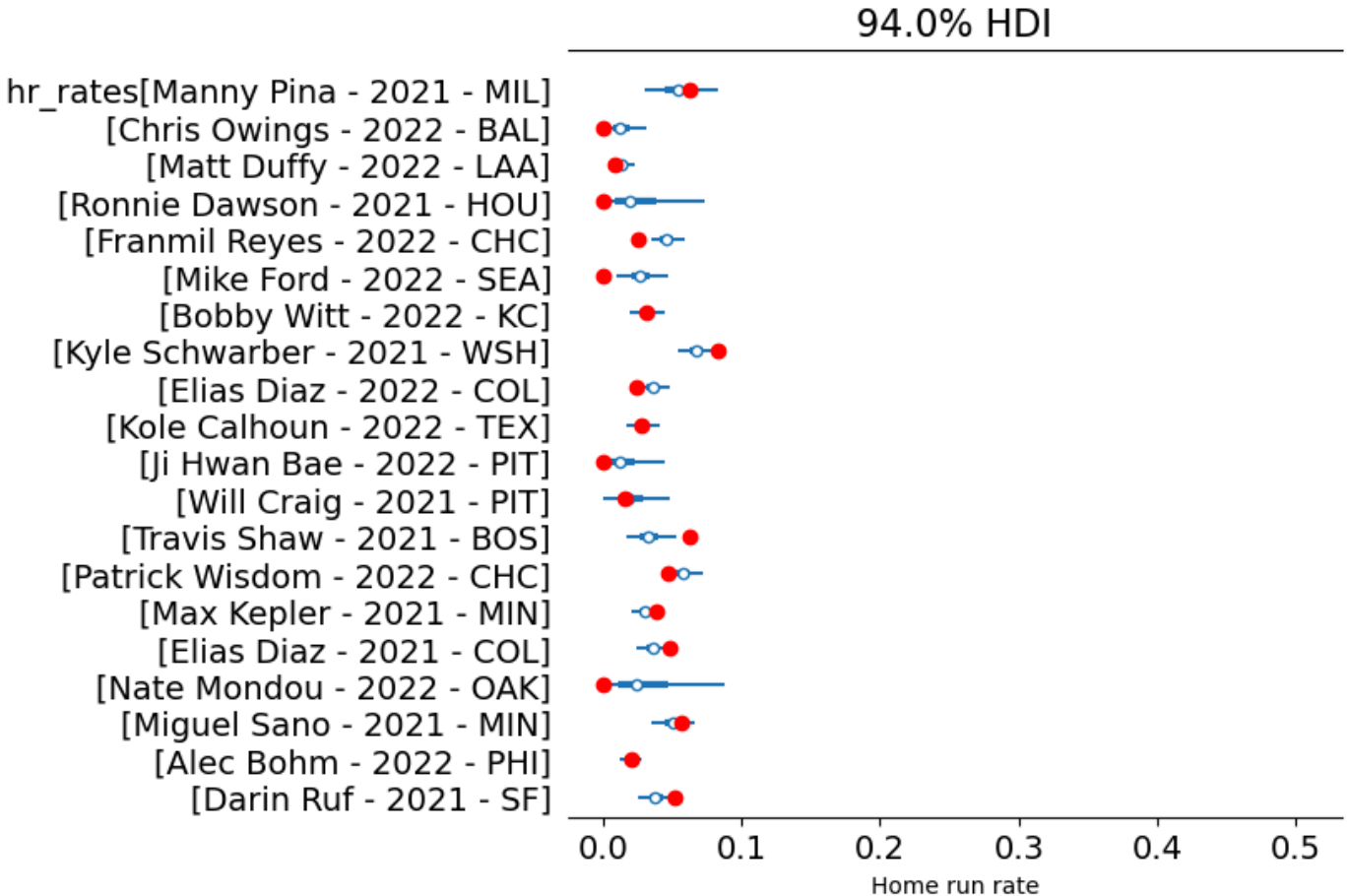
```
coords = {'player': batting_data.Name.values}
with pm.Model(coords=coords) as partial_pooling:

    mu = pm.Normal('mu', mu=-2, sigma=1)
    sigma = pm.HalfCauchy('sigma', 1)
    logit_p_raw = pm.Normal('logit_p_raw', 0, 1,
                            dims='player')

    p = pm.math.invlogit(logit_p_raw * sigma + mu)

    y = pm.Binomial('y', n=ab, p=p,
                    observed=h,
                    dims='player')
```

Partial Pooling Model

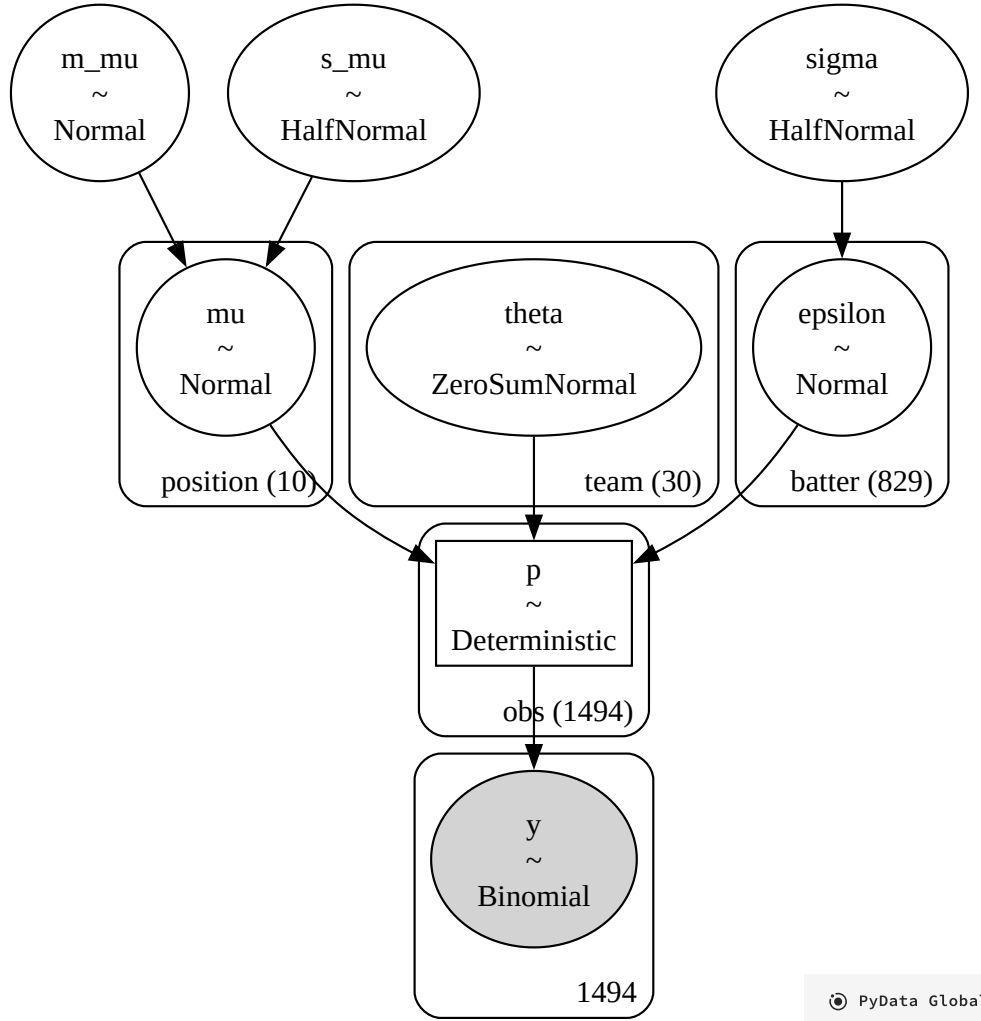



No Pooling Estimates
HR Rate
Plate appearances
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0
100
200
300
400
500
600
700

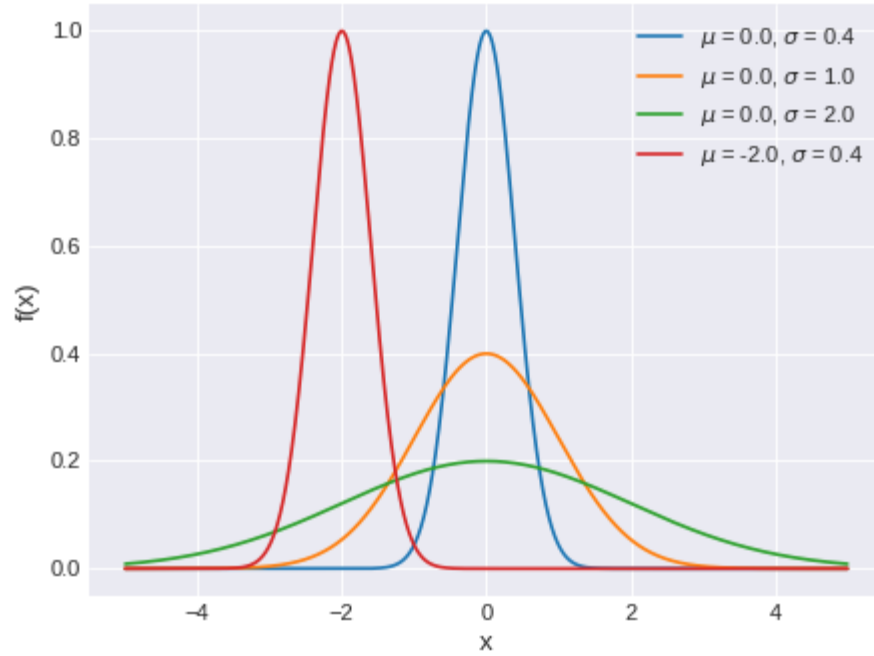

Partial Pooling Estimates
HR Rate
Plate appearances
0
100
200
300
400
500
600
700


94.0% HDI
hr_rates[Manny Pina - 2021 - MIL]
[Chris Owings - 2022 - BAL]
[Matt Duffy - 2022 - LAA]
[Ronnie Dawson - 2021 - HOU]
[Franmil Reyes - 2022 - CHC]
[Mike Ford - 2022 - SEA]
[Bobby Witt - 2022 - KC]
[Kyle Schwarber - 2021 - WSH]
[Elias Diaz - 2022 - COL]
[Kole Calhoun - 2022 - TEX]
[Ji Hwan Bae - 2022 - PIT]
[Will Craig - 2021 - PIT]
[Travis Shaw - 2021 - BOS]
[Patrick Wisdom - 2022 - CHC]
[Max Kepler - 2021 - MIN]
[Elias Diaz - 2021 - COL]
[Nate Mondou - 2022 - OAK]
[Miguel Sano - 2021 - MIN]
[Alec Bohm - 2022 - PHI]
[Darin Ruf - 2021 - SF]
0.0
0.1
0.2
0.3
0.4
0.5
Home run rate

Hierarchical model

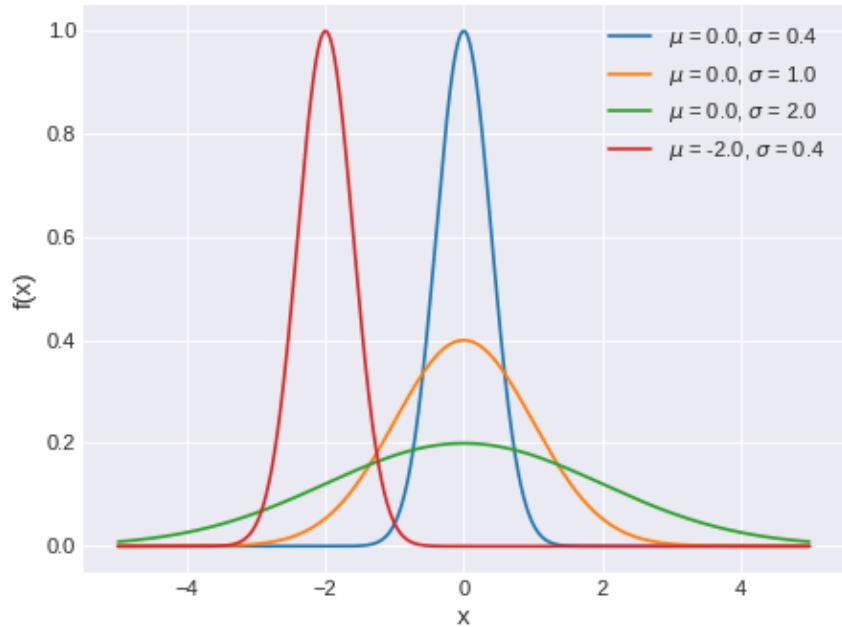


Building Models with
Gaussian distributions



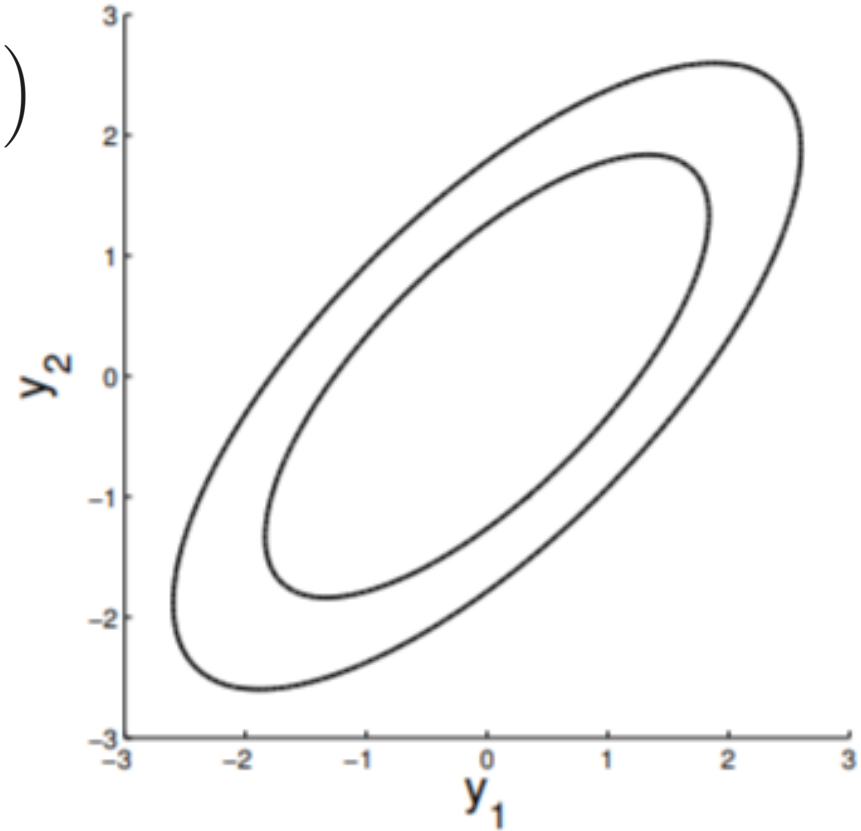
Gaussian (normal) distribution

$$y \sim N(\mu, \Sigma)$$



Gaussian distribution

$$Pr(y_1, y_2|\mu, \Sigma) = \mathcal{N}\left(\begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} \sigma_1^2 & \sigma_1\sigma_2\rho \\ \sigma_1\sigma_2\rho & \sigma_2^2 \end{bmatrix}\right)$$



Marginalization property

Info
The marginal distribution of some elements of a multivariate normal is also normal

$$p(x,y) = \mathcal{N}\left(\begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix}, \begin{bmatrix} \Sigma_x & \Sigma_{xy} \\ \Sigma_{xy}^T & \Sigma_y \end{bmatrix}\right)$$

$$p(x) = \int p(x,y)dy = \mathcal{N}(\mu_x, \Sigma_x)$$

Conditioning property

> **Info**
> The conditional distribution of some elements of a multivariate normal is also normal

$$p(x|y) = N(\underbrace{\mu_x + \Sigma_{xy}\Sigma_y^{-1}(y - \mu_y)}_{\text{conditional mean}}, \underbrace{\Sigma_x - \Sigma_{xy}\Sigma_y^{-1}\Sigma_{xy}^T}_{\text{conditional covariance}})$$

Gaussian processes

Gaussian distribution:

$$y \sim N(\mu, \Sigma)$$

Gaussian process:

$$f \sim GP(m(x), k(x, x'))$$

Gaussian Process:

An *infinite collection* of random variables, any finite subset of which have a Gaussian distribution.

Gaussian processes

$$f \sim GP(m(x), k(x, x'))$$

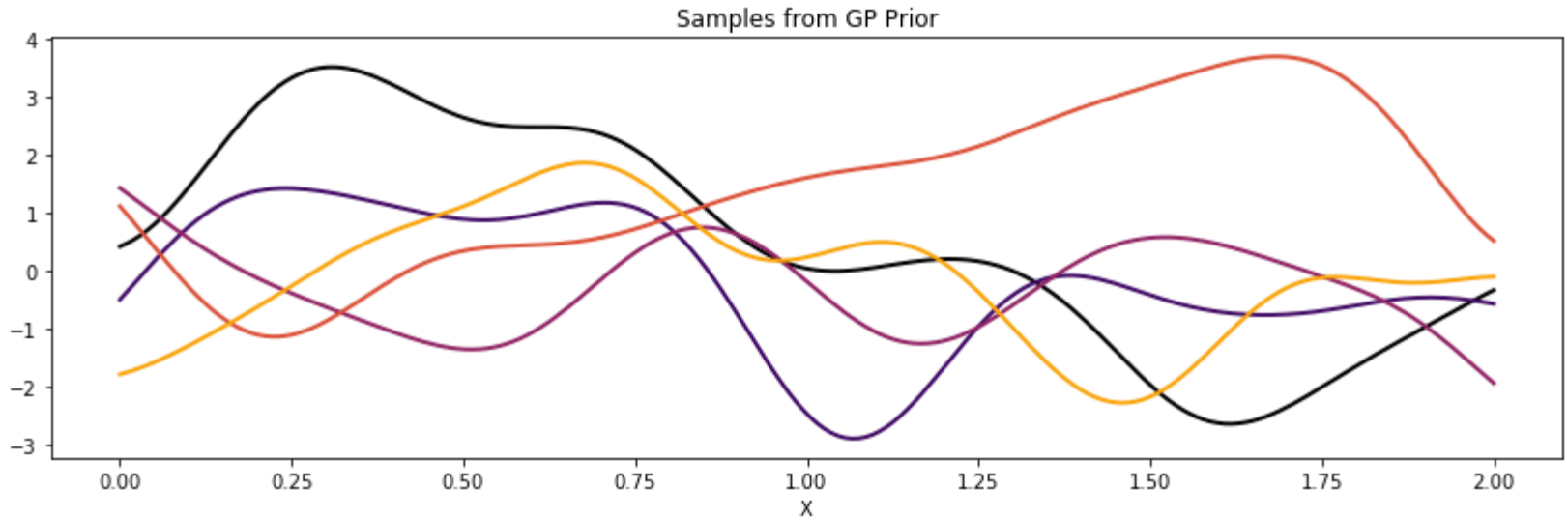
Covariance Functions

Functions that generate **covariance matrices**:

$$\Sigma \sim k(x, x' | \phi)$$

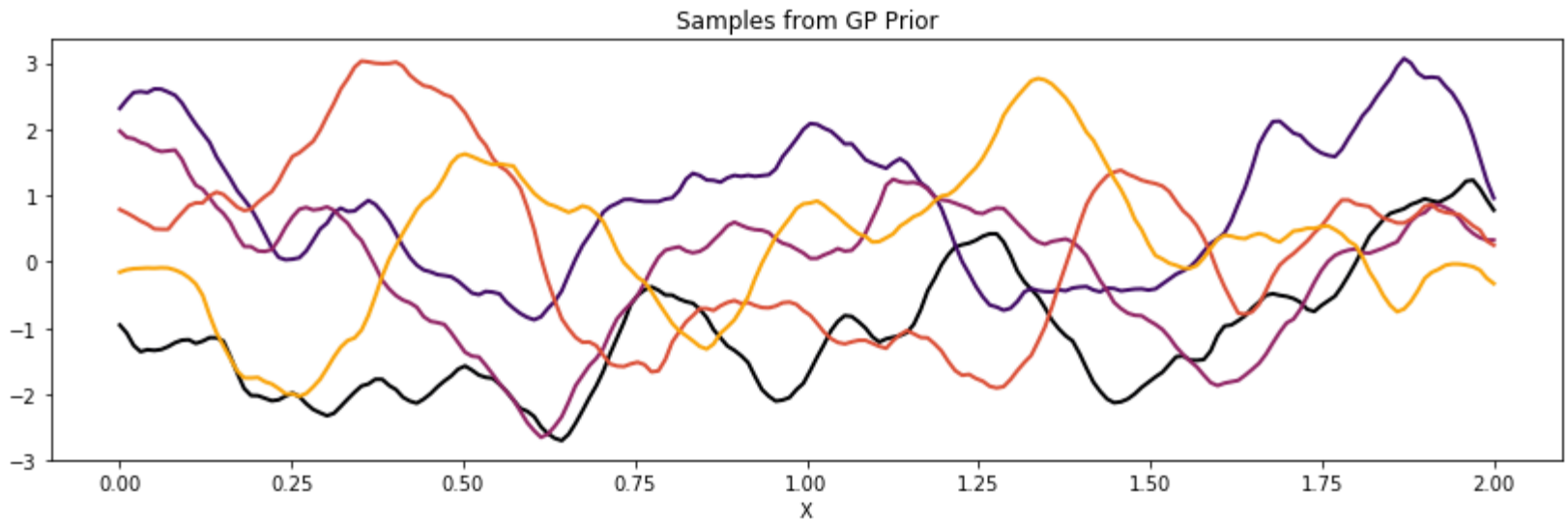
Exponential Quadratic

$$k(x, x') = \exp\left[-\frac{(x - x')^2}{2\ell^2}\right]$$



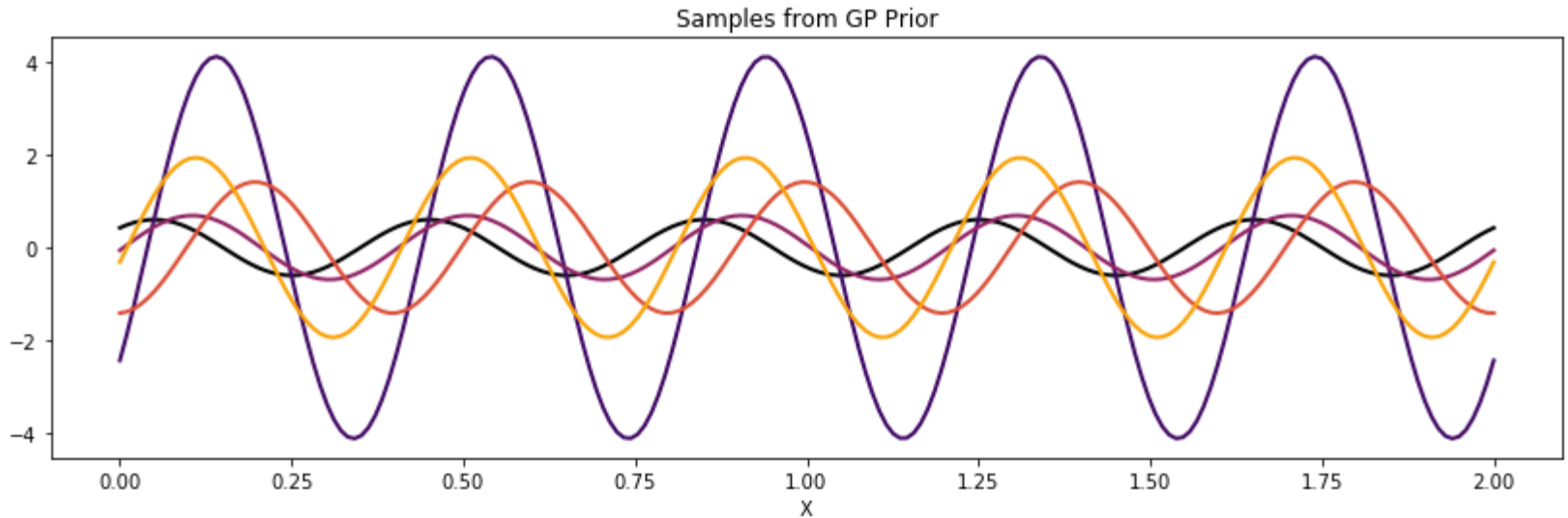
Matern(3/2)

$$k(x, x') = \left(1 + \frac{\sqrt{3(x - x')^2}}{\ell}\right) \exp\left[-\frac{\sqrt{3(x - x')^2}}{\ell}\right]$$



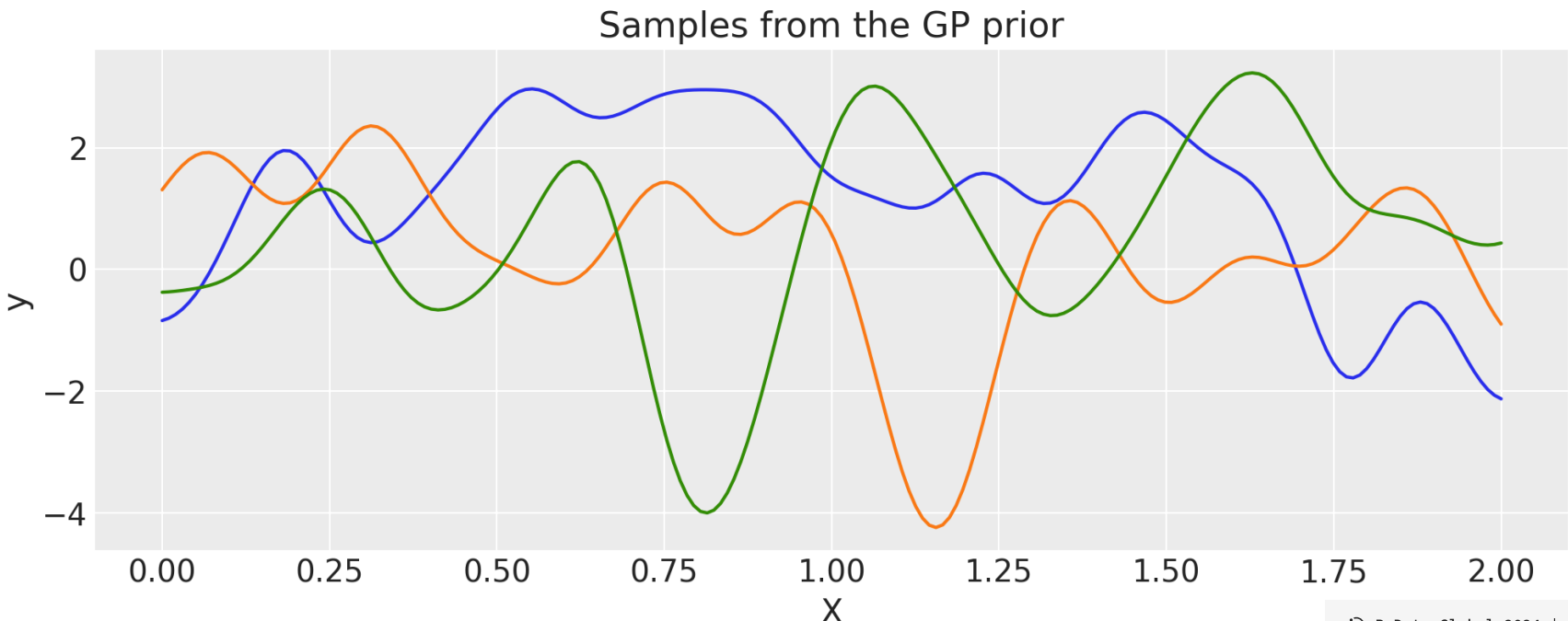
Cosine

$$k(x, x') = \cos\left(\frac{||x - x'||}{\ell^2}\right)$$



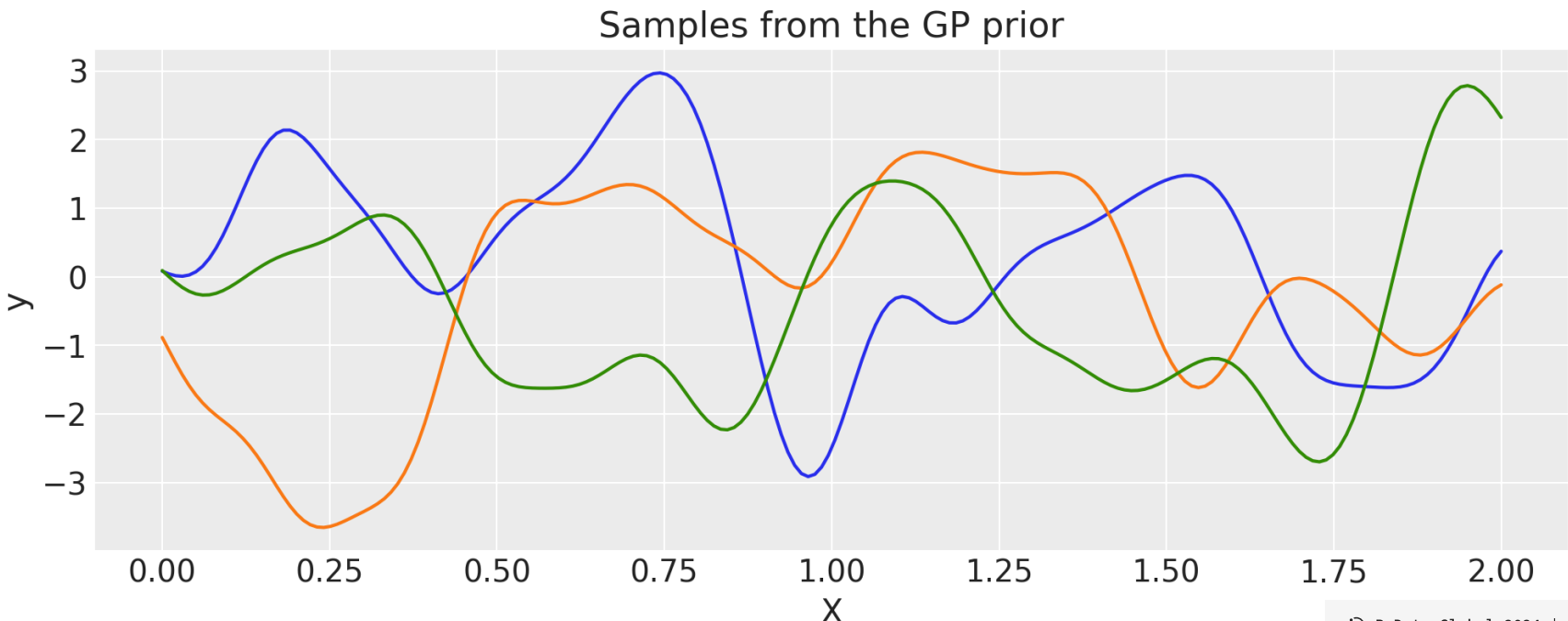
Covariance functions are additive

$$k(x, x') = \exp\left[-\frac{(x - x')^2}{2\ell_1^2}\right] + \exp\left[-\frac{(x - x')^2}{2\ell_2^2}\right]$$



Covariance functions are multiplicative

$$k(x, x') = \exp\left[-\frac{(x - x')^2}{2\ell_1^2}\right] * \exp\left[-\frac{(x - x')^2}{2\ell_2^2}\right]$$



Mean Functions

Functions that generate **mean vectors**:

$$\mu \sim m(x|\phi)$$

Examples:

- Zero: $m(x) = 0$
- Constant: $m(x) = C$
- Linear: $m(x) = Ax + b$

Example:

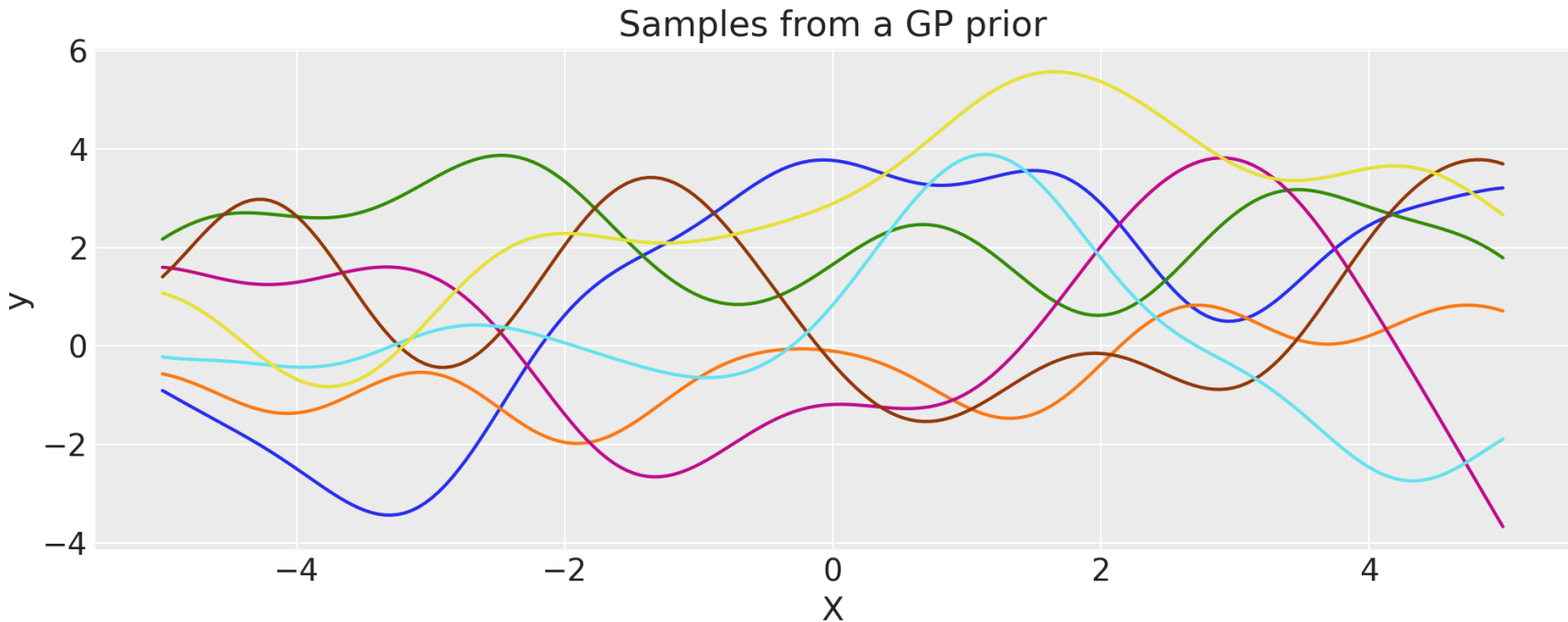
$$m(x) = 0$$

$$k(x, x') = \theta_1 \exp\left(-\frac{\theta_2}{2}(x - x')^2\right)$$

```
def exponential_cov(x, y, scale, length_scale):
    return scale * np.exp(
        -0.5 * length_scale * np.subtract.outer(x, y)**2
    )
```

Prior over functions

$$y \sim GP(m = 0, k = \text{ExpQuad}(1))$$



Marginal Gaussian Process

$$\underbrace{Pr(\theta|y)}_{\text{closed-form GP}} \propto \prod_{i=1}^{N} \overbrace{Pr(y_i|\theta)}^{\text{Gaussian data}} \underbrace{Pr(\theta)}_{\text{GP prior}}$$

Latent Gaussian Process

$$\underbrace{Pr(\theta|y)}_{\text{transformed GP}} \propto \prod_{i=1}^{N} \overbrace{Pr(y_i|\theta)}^{\text{Binomial data}} \underbrace{Pr(\theta)}_{\text{GP prior}}$$

Hyperparameter priors

$$k(x, x') = \eta \exp\left[-\frac{(x - x')^2}{2\rho^2}\right]$$

```
with hr_rate_model:

    lengthscale = pm.InverseGamma('lengthscale', 10, 25)
    amplitude = pm.HalfNormal('amplitude', 3)
```

Covariance function and marginal GP

```
with hr_rate_model:

    cov_func = amplitude**2 * pm.gp.cov.ExpQuad(1, lengthscale
    gp = pm.gp.Latent(cov_func=cov_func)
    f = gp.prior('f', X=age[:, None])

    p = Deterministic('p', invlogit(f))
```

Binomial likelihood

```
with salmon_model:

    hr = Binomial('hr', p=p, n=pa, observed=hr)
```

Posterior Covariance

$$k^*(x^*) = k(x^*, x^*) + \sigma^2 - k(x^*, x)^T \underbrace{[k(x, x) + \sigma^2 I]^{-1}}_{😢} k(x^*, x)$$

$$O(N^3) \text{ compute time}, O(N^2) \text{ memory}$$

Hilbert Space Gaussian Processes (HSGP)

A *low-rank*, *parametric* approximation based on a pre-computed set of *basis functions*.

For m basis functions ($m << n$):

$$O(n^3) \longrightarrow O(mn + m)$$

How does it work?

$$f \sim \mathcal{GP}\left(0, k\left(x, x'; \ell\right)\right)$$

is replaced with

$$f \approx \phi(x)\beta(\ell)$$

HSGP Hyperparameters

m

Number of basis vectors

L

Boundary of space

c

Extension factor

Choosing HSGP parameters

1. Approximation Accuracy

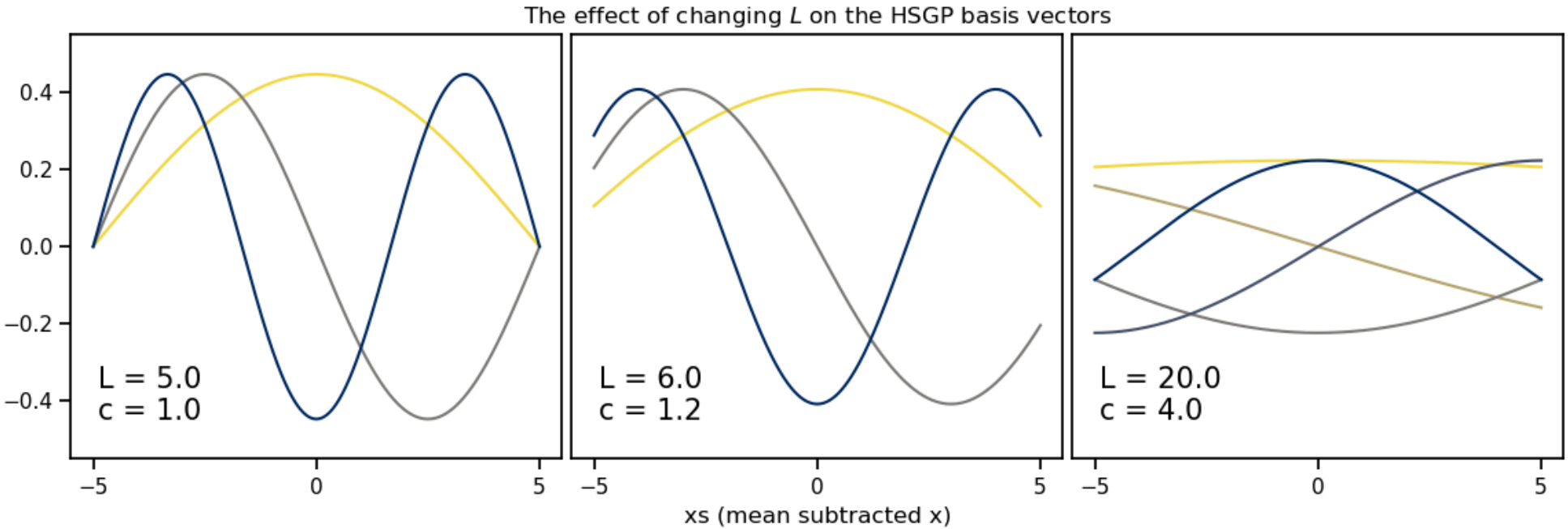
How well should the HSGP match the full GP?

2. Computational Efficiency

Balancing speed and memory requirements

3. Prediction Locations

Where in X space do we need to predict?


The effect of changing L on the HSGP basis vectors
0.4
0.2
0.0
−0.2
−0.4
L = 5.0
c = 1.0
L = 6.0
c = 1.2
L = 20.0
c = 4.0
−5
0
5
xs (mean subtracted x)

Covariance function

```
with hr_rate_model:

    amplitude = pm.HalfCauchy('amplitude', 1) + 1
    lengthscale = pm.InverseGamma('lengthscale',
        alpha=10, beta=25
    )

    cov_func = (amplitude**2
        * pm.gp.cov.ExpQuad(input_dim=1, ls=lengthscale)
    )
```

Latent HSGP

```
with hr_rate_model:

    gp_age = pm.gp.HSGP(
        m=[100],
        c=3,
        cov_func=cov_func,
        drop_first=True,
    )
    age_effect = gp_age.prior(
        "age_effect",
        X=np.unique(age)[:, None],
    )
```

Likelihood of Transformed GP

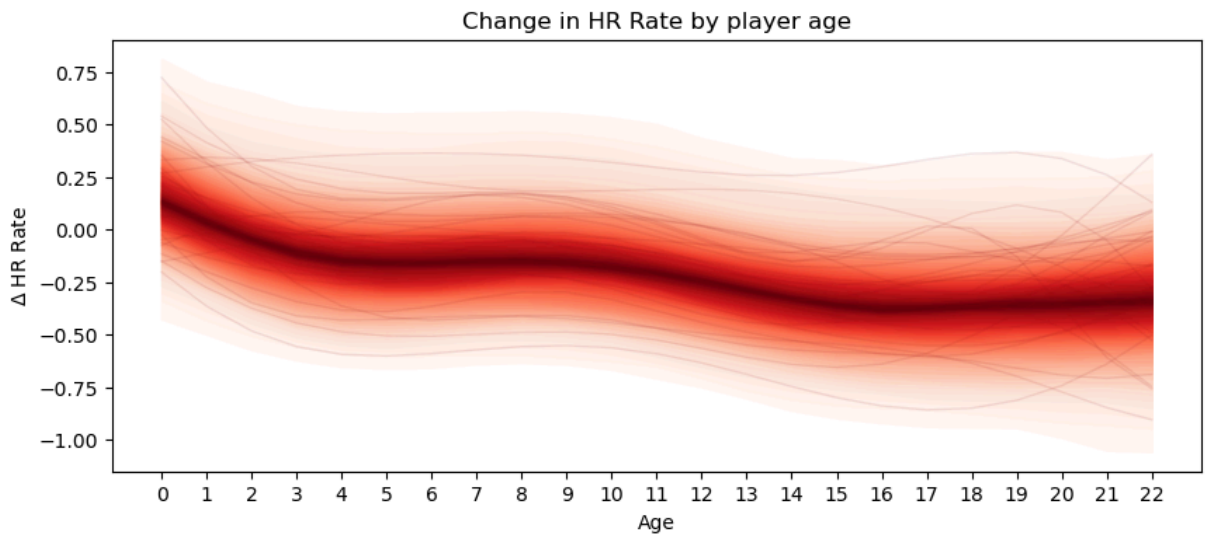
```
with hr_rate_model:

    logit_p = pm.Deterministic('logit_p',
        mu[position_idx] +
        theta[team_idx] +
        epsilon[batter_idx] +
        age_effect[age_idx],
    )
```

MCMC Sampling

```
with hr_rate_model:
    trace_gp = pm.sample(nuts_sampler='nutpie', chains=2)
```

```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (2 chains in 2 jobs)
NUTS: [m_mu, s_mu, mu, theta, sigma, epsilon, amplitude, lengthscale, f_hsgp_coeffs]
100%|██████████| 3000/3000 [04:04<00:00, 12.27it/s]
```

```
import arviz as az
from pymc.gp.util import plot_gp_dist
f = az.extract(trace_gp, var_names="age_effect")
plot_gp_dist(x=np.unique(age)[:, None], samples=f.T)
```



Caveats

1. Can only be used with **stationary** covariance kernels
2. Does not scale well with the **input dimension**
3. May struggle with more **rapidly varying** processes