Gamma-Poisson Model

General Principles

To model the relationship between a count outcome variable and one or more independent variables with overdispersion , we can use the *Negative Binomial model*.

Considerations

> Caution
>
> - We have the same considerations as for the Poisson model.
> - Overdispersion is handled because the Gamma-Poisson model assumes that each Poisson count observation has its own rate. This is an additional parameter specified in the model (in the code, it is `log_days`).

Example

Below is an example code snippet demonstrating a Bayesian Gamma-Poisson model using the BayesForge (BF) package.

Python

```
from BayesForge import bf
import jax.numpy as jnp

# Setup device ---------------------------------------------
m = bf(platform='cpu') # Import

# Import Data & Data Manipulation ------------------------------------------------
```

```python
# Import
from importlib.resources import files
data_path =  m.load.sim_gamma_poisson(only_path=True)
m.data(data_path, sep=',')


# Define model ------------------------------------------------
def model(log_days, monastery, y):
    a = m.dist.normal(0, 1, name = 'a', shape=(1,))
    b = m.dist.normal(0, 1, name = 'b', shape=(1,))
    phi = m.dist.exponential(1, name = 'phi', shape=(1,))
    mu = jnp.exp(log_days + a + b * monastery)
    Lambda =  m.dist.gamma(rate = mu*phi, concentration = phi, name = 'Lambda')
    m.dist.poisson(rate = Lambda, obs=y)
# Run MCMC ----------------------------------------------------
m.fit(model, progress_bar=False) # Optimize model parameters through MCMC sampling

# Summary -----------------------------------------------------
m.summary() # Get posterior distributions
```

```
bf v 0.0.48 package loaded
```

```
E0526 16:09:15.546243 1051694 cuda_dnn.cc:523] Loaded runtime CuDNN library: 9.1.0 but source
E0526 16:09:15.551444 1051694 cuda_dnn.cc:523] Loaded runtime CuDNN library: 9.1.0 but source
```

```
jax.local_device_count 32
```

	mean	sd	hdi_5.5%	hdi_94.5%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
Lambda[0]	1.46	0.34	0.92	2.01	0.00	0.00	7637.67	2212.89	1.00
Lambda[1]	1.47	0.34	0.89	1.96	0.00	0.00	7582.21	2835.22	1.00
Lambda[2]	1.63	0.36	1.04	2.17	0.00	0.00	6870.91	2699.35	1.00
Lambda[3]	1.55	0.35	0.99	2.06	0.00	0.00	7910.03	2574.77	1.00
Lambda[4]	1.46	0.35	0.92	2.01	0.00	0.00	8511.18	2837.00	1.00
...	...	...	...	...	...	...	...	...	...
Lambda[3398]	3.13	0.72	2.01	4.26	0.01	0.01	6698.62	2692.70	1.00
Lambda[3399]	2.97	0.71	1.94	4.17	0.01	0.01	5646.74	2587.29	1.00
a[0]	-0.41	0.02	-0.44	-0.38	0.00	0.00	593.86	1081.71	1.01
b[0]	-2.75	0.03	-2.80	-2.69	0.00	0.00	1049.48	1921.53	1.00
phi[0]	17.39	2.28	14.09	21.36	0.22	0.15	111.96	238.76	1.03

R

```r
library(BayesForge)

# Setup platform-----------------------------------------------
m=importBF(platform='cpu')

# Import data -------------------------------------------------
m$data(m$load$sim_gamma_poisson(only_path=T), sep = '')
m$data_to_model(list('log_days', 'monastery', 'y' )) # Send to model (convert to jax array)

# Define model ------------------------------------------------
model <- function(log_days, monastery, y){
  # Parameter prior distributions
  alpha = bf.dist.normal(0, 1, name='alpha', shape=c(1))
  beta = bf.dist.normal(0, 1, name='beta', shape=c(1))
  phi = bf.dist.exponential(1, name='phi', shape=c(1))
  mu = jnp$exp(log_days + alpha + beta * monastery)
  Lambda =  bf.dist.gamma(rate = mu*phi, concentration = phi, name = 'Lambda')
  # Likelihood
  bf.dist.poisson(rate=Lambda, obs=y)
}
# Run MCMC ----------------------------------------------------
m$fit(model) # Optimize model parameters through MCMC sampling

# Summary -----------------------------------------------------
m$summary() # Get posterior distributions
```

Julia

```julia
using BayesForge

# Setup device------------------------------------------------
m = importBF(platform="cpu")

# Import Data & Data Manipulation -----------------------------------------------
# Import
data_path = m.load.sim_gamma_poisson(only_path = true)
m.data(data_path, sep=',')

# Define model ------------------------------------------------
```

```
@BF function model(log_days, monastery, y)
    a = m.dist.normal(0, 1, name = "a", shape=(1,))
    b = m.dist.normal(0, 1, name = "b", shape=(1,))
    phi = m.dist.exponential(1, name = "phi", shape=(1,))
    mu = jnp.exp(log_days + a + b * monastery)
    Lambda =  m.dist.gamma(rate = mu*phi, concentration = phi, name = "Lambda")
    m.dist.poisson(rate = Lambda, obs=y)
end

# Run mcmc -----------------------------------------------
m.fit(model)  # Optimize model parameters through MCMC sampling

# Summary ------------------------------------------------
m.summary() # Get posterior distributions
```

Mathematical Details

Bayesian model

In the Bayesian formulation, we define each parameter with priors . We can express the Bayesian regression model accounting for prior distributions as follows:

$$Y_i \sim \text{Poisson}(\lambda_i)$$

$$\lambda_i \sim \text{Gamma}(\mu_i\phi, \phi)$$

$$\log(\mu_i) = \text{rates}_i + \alpha + \beta X_i$$

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

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

$$\phi \sim \text{Exponential}(1)$$

Where:

- Y_i is the dependent variable for observation i.
- λ_i is the rate parameter of the Poisson distribution for observation i, assuming that each Poisson count observation has its own $rate_i$.
- μ_i is the mean rate parameter.
-
- ϕ controls the level of overdispersion in the rates.
- α is the intercept term.
- β is the regression coefficient.
- X_i is the value of the predictor variable for observation i.

Notes

> **Note**
>
> - We can apply multiple variables similarly as in chapter 2.
> - We can apply interaction terms similarly as in 3. Interaction between continuous variables.
> - We can apply categorical variables similarly as in chapter 4.

Reference(s)