Build in models

PCA

Python

```
from BI import bi,jnp
m=bi()
m.data('iris.csv', sep=',') # Data is already scaled
m.data_on_model = dict(
    X=jnp.array(m.df.iloc[:,0:-2].values)
)
m.fit(m.models.pca(type="ARD"), progress_bar=False) # or robust, sparse, classic, sparse_robu

m.models.pca.plot(
    X=m.df.iloc[:,0:-2].values,
    y=m.df.iloc[:,-2].values,
    feature_names=m.df.columns[0:-2],
    target_names=m.df.iloc[:,-1].unique(),
    color_var=m.df.iloc[:,0].values,
    shape_var=m.df.iloc[:,-2].values
)
```

Survival analysis

```
from BI import bi
import numpy as np
import jax.numpy as jnp
# Setup device----------------------------------------------------
m = bi(platform='cpu')
```

```
# Import Data & Data Manipulation ---------------------------------------------
# Import
from importlib.resources import files
data_path = files('BI.resources.data') / 'mastectomy.csv'
m.data(data_path, sep=',')

m.df.metastasized = (m.df.metastasized == "yes").astype(np.int64)
m.df.event = jnp.array(m.df.event.values, dtype=jnp.int32)

## Create survival object
m.models.survival.surv_object(time='time', event='event', cov='metastasized', interval_lengt

# Plot censoring ------------------------------------------------
m.models.survival.plot_censoring(cov='metastasized')

# Model ---------------------------------------------
def model(intervals, death, metastasized, exposure):
    # Parameter prior distributions-------------------------
    ## Base hazard distribution
    lambda0 = m.dist.gamma(0.01, 0.01, shape= intervals.shape, name = 'lambda0')
    ## Covariate effect distribution
    beta = m.dist.normal(0, 1000, shape = (1,),  name='beta')
    ### Likelihood
    #### Compute hazard rate based on covariate effect
    lambda_ = m.models.survival.hazard_rate(cov = metastasized, beta = beta, lambda0 = lambda
    #### Compute exposure rates
    mu = exposure * lambda_

    # Likelihood calculation
    y = m.dist.poisson(mu + jnp.finfo(mu.dtype).tiny, obs = death)

# Run mcmc ------------------------------------------------
m.fit(model, num_samples=500)

# Summary ---------------------------------------------
print(m.summary())

# Plot hazards and survival function ----------------------------------------------
m.models.survival.plot_surv()
```

Gaussian Mixture Models

Dirichlet Process Mixture Models

Python

```python
from BI import bi
from sklearn.datasets import make_blobs

# Generate synthetic data
data, true_labels = make_blobs(
    n_samples=500, centers=8, cluster_std=0.8,
    center_box=(-10,10), random_state=101
)


#  The model
def gmm(data, K, initial_means): # Here K is the *exact* number of clusters
    D = data.shape[1]  # Number of features
    alpha_prior = 0.5 * jnp.ones(K)
    w = dist.dirichlet(concentration=alpha_prior, name='weights')

    with dist.plate("components", K): # Use fixed K
        mu = dist.multivariatenormal(loc=initial_means, covariance_matrix=0.1*jnp.eye(D), nam
        sigma = dist.halfcauchy(1, shape=(D,), event=1, name='sigma')
        Lcorr = dist.lkjcholesky(dimension=D, concentration=1.0, name='Lcorr')

        scale_tril = sigma[..., None] * Lcorr

    dist.mixturesamefamily(
        mixing_distribution=dist.categorical(probs=w, create_obj=True),
        component_distribution=dist.multivariatenormal(loc=mu, scale_tril=scale_tril, create_
        name="obs",
        obs=data
    )

m.data_on_model = {"data": data,"K": 4 }
m.fit(gmm) # Optimize model parameters through MCMC sampling
m.plot(X=data,sampler=m.sampler) # Prebuild plot function for GMM
```

R

```
```

Network Models

Python

```
from BI import bi

# Setup device--------------------------------------------------
from BI import bi
# Setup device--------------------------------------------------
m = bi(platform='cpu')

m.data_on_model = dict(
    idx = idx,
    Any = Any-1,
    Merica = Merica-1,
    Quantum = Quantum-1,
    result_outcomes = m.net.mat_to_edgl(data['outcomes']),
    kinship = m.net.mat_to_edgl(kinship),
    focal_individual_predictors = data['individual_predictors'],
    target_individual_predictors = data['individual_predictors'],
    exposure_mat = data['exposure']
)


def model(idx, result_outcomes,
    exposure,
    kinship,
    focal_individual_predictors, target_individual_predictors,
    Any, Merica, Quantum):
      # Block ---------------------------------------
      B_any = m.net.block_model(Any,1)
      B_Merica = m.net.block_model(Merica,3)
      B_Quantum = m.net.block_model(Quantum,2)

      ## SR shape =  N individuals---------------------------------------
      sr =  m.net.sender_receiver(focal_individual_predictors,target_individual_predictors)
```

```
      # Dyadic shape = N dyads-------------------------------------
      dr = m.net.dyadic_effect(dyadic_predictors)

      m.dist.poisson(jnp.exp(B_any + B_Merica + B_Quantum + sender_receiver + dr), obs = resu



m.fit(model)
summary = m.summary()
summary.loc[['focal_effects[0]', 'target_effects[0]', 'dyad_effects[0]']]

m.fit(model2)
summary = m.summary()
summary
```

R

```
library(BI)
# Setup platform-----------------------------------------------
m=importbi(platform='cpu')

# Import data -------------------------------------------------

load(paste(system.file(package = "BI"),'/data/STRAND sim sr only.Rdata', sep = ''))

ids = 0:(model_dat$N_id-1)
idx = m$net$vec_node_to_edgle(jnp$stack(jnp$array(list(ids, ids)), axis = -as.integer(1)))

keys <- c("idx",
          'idxShape',
          "result_outcomes",
          'focal_individual_predictors',
          'target_individual_predictors')

values <- list(
  idx,
  idx$shape[[1]],
  m$net$mat_to_edgl(model_dat$outcomes[,,1]),
  jnp$array(model_dat$individual_predictors)$reshape(as.integer(1),as.integer(50)),
  jnp$array(model_dat$individual_predictors)$reshape(as.integer(1),as.integer(50))
)
```

```
data = py_dict(keys, values, convert = TRUE)
m$data_on_model=data

# Define model ------------------------------------------------
model <- function(idx, idxShape, result_outcomes,focal_individual_predictors, target_individu
  N_id = 50
  x=0.1/jnp$sqrt(N_id)
  tmp=jnp$log(x / (1 - x))

  ## Block ---------------------------------------
  B = bi.dist.normal(tmp, 2.5, shape=c(1), name = 'block')

  #SR ---------------------------------------
  sr =  m$net$sender_receiver(focal_individual_predictors,target_individual_predictors)

  ### Dyadic---------------------------------------
  #dr, dr_raw, dr_sigma, dr_L = m.net.dyadic_random_effects(idx.shape[0], cholesky_density =
  dr = m$net$dyadic_effect(shape = c(idxShape))

  ## SR ---------------------------------------
  m$poisson(jnp$exp(B + sr + dr), obs=result_outcomes)
}

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

# Summary ------------------------------------------------
summary =m$summary()

summary[rownames(summary) %in% c('focal_effects[0]', 'target_effects[0]', 'block[0]'),]
```