Emergent System Lab

PYHSMM読解 1

LST勉強会第三回

@ryo_huhuhu
RYO NAKASHIMA

2015/8/28

はじめに

- NPB-DAA（dahsmm）の理解のため，まずはpyhsmmのコードリーディングから行っていく
  - pyhsmm/example にあるhsmm.py（チュートリアルコード）の流れに沿って理解していく
  - 今回は右部分まで読解

    依存先で使用されている部分も理解すること

hsmm.py

```
from __future__ import division
import numpy as np
np.seterr(divide='ignore') # these warnings are usually harmless for this code
from matplotlib import pyplot as plt
import copy

import pyhsmm
from pyhsmm.util.text import progprint_xrange

SAVE_FIGURES = False

print ¥
'''
This demo shows the HDP-HSMM in action. Its iterations are slower than those for
the (Sticky-)HDP-HMM, but explicit duration modeling can be a big advantage for
conditioning the prior or for discovering structure in data.
'''

#####################
#  data generation  #
#####################

N = 4
T = 500
obs_dim = 2

obs_hypparams = {'mu_0':np.zeros(obs_dim),
                'sigma_0':np.eye(obs_dim),
                'kappa_0':0.3,
                'nu_0':obs_dim+5}

dur_hypparams = {'alpha_0':2*30,
                 'beta_0':2}

# Construct the true observation and duration distributions
true_obs_distns = [pyhsmm.distributions.Gaussian(**obs_hypparams) for state in range(N)]
true_dur_distns = [pyhsmm.distributions.PoissonDuration(**dur_hypparams) for state in range(N)]
```

hsmm.py>> 1～37行目

```python
N = 4
T = 500
obs_dim = 2

obs_hypparams = {'mu_0':np.zeros(obs_dim),
                'sigma_0':np.eye(obs_dim),
                'kappa_0':0.3,
                'nu_0':obs_dim+5}

dur_hypparams = {'alpha_0':2*30,
                 'beta_0':2}

# Construct the true observation and duration distributions
true_obs_distns = [pyhsmm.distributions.Gaussian(**obs_hypparams) for state in range(N)]
true_dur_distns = [pyhsmm.distributions.PoissonDuration(**dur_hypparams) for state in range(N)]
```

- **推定するための真のデータ生成を行う**
  - 状態数，フレーム数，データ次元設定
  - 観測分布パラメータ設定

    多次元ガウスを仮定
  - 持続時間分布パラメータ設定

    ポアソン分布を過程
  - ガウス分布オブジェクト生成
  - ポアソン分布オブジェクト生成

basic/pybasicbayes/distributions.pyのGaussian クラスに注目

```python
class Gaussian(_GaussianBase, GibbsSampling, MeanField, Collapsed, MAP, MaxLikelihood):
    '''
    Multivariate Gaussian distribution class.

    NOTE: Only works for 2 or more dimensions. For a scalar Gaussian, use one of
    the scalar classes. Uses a conjugate Normal/Inverse-Wishart prior.

    Hyperparameters mostly follow Gelman et al.'s notation in Bayesian Data
    Analysis, except sigma_0 is proportional to expected covariance matrix:
        nu_0, sigma_0
        mu_0, kappa_0

    Parameters are mean and covariance matrix:
        mu, sigma
    '''

    def __init__(self,mu=None,sigma=None,
            mu_0=None,sigma_0=None,kappa_0=None,nu_0=None,
            kappa_mf=None,nu_mf=None):
        self.mu    = mu
        self.sigma = sigma

        self.mu_0    = mu_0
        self.sigma_0 = sigma_0
        self.kappa_0 = kappa_0
        self.nu_0    = nu_0

        self.kappa_mf = kappa_mf if kappa_mf is not None else kappa_0
        self.nu_mf    = nu_mf if nu_mf is not None else nu_0
        self.mu_mf    = mu
        self.sigma_mf = sigma

        if (mu,sigma) == (None,None) and None not in (mu_0,sigma_0,kappa_0,nu_0):
            self.resample() # initialize from prior

    @property
    def hypparams(self):
        return dict(mu_0=self.mu_0,sigma_0=self.sigma_0,kappa_0=self.kappa_0,nu_0=self.nu_0)

    @property
    def num_parameters(self):
        D = len(self.mu)
        return D*(D+1)/2
```

Slide Section

Gaussian class

Emergent System Lab

basic/pybasicbayes/distributions.py>> Gaussian class 144~171

```python
144     def __init__(self,mu=None,sigma=None,
145             mu_0=None,sigma_0=None,kappa_0=None,nu_0=None,
146             kappa_mf=None,nu_mf=None):
147         self.mu    = mu
148         self.sigma = sigma
149 
150         self.mu_0    = mu_0
151         self.sigma_0 = sigma_0
152         self.kappa_0 = kappa_0
153         self.nu_0    = nu_0
154 
155         self.kappa_mf = kappa_mf if kappa_mf is not None else kappa_0
156         self.nu_mf    = nu_mf if nu_mf is not None else nu_0
157         self.mu_mf    = mu
158         self.sigma_mf = sigma
159 
160         if (mu,sigma) == (None,None) and None not in (mu_0,sigma_0,kappa_0,nu_0):
161             self.resample() # initialize from prior
162 
163     @property
164     def hypparams(self):
165         return dict(mu_0=self.mu_0,sigma_0=self.sigma_0,kappa_0=self.kappa_0,nu_0=self.nu_0)
166 
167     @property
168     def num_parameters(self):
169         D = len(self.mu)
170         return D*(D+1)/2
171 
```

- 多変量ガウス分布（多次元ガウス

  __init__：コンストラクタ（厳密には違うらしい

  - mu：平均，sigma：分散（Noneならdef resample() で取得
  - mu_0： muに関する（共役）事前分布のハイパーパラメータ
  - sigma_0：sigmaに関する（共役）事前分布である逆ウィシャート分布用のハイパーパラメータ

    必ず対称半正定値行列
  - kappa_0：逆ウィシャートから得た分散パラの調整用ハイパーパラメータ

$$\Sigma \sim \text{Inv-Wishart}_{\nu_0}(\Lambda_0^{-1})$$
$$\mu|\Sigma \sim \text{N}(\mu_0, \Sigma/\kappa_0),$$

  - nu_0：逆ウィシャートの自由度ハイパーパラメータ（小さいと変動が大きくなる）

def hypparams():ハイパーパラメータ参照
def num_parameters():EMで使用？（gibbsで用いてない？

mean field: 変分ベイズ推論用(ギブスサンプリング)では必要ない？

Emergent System Lab

basic/pybasicbayes/distributions.py> > Gaussian class 172~212

```python
    @staticmethod
    def _get_statistics(data,D=None):
        n = getdatasize(data)
        if n > 0:
            D = getdatadimension(data) if D is None else D
            if isinstance(data,np.ndarray):
                xbar = np.reshape(data,(-1,D)).mean(0)
                centered = data - xbar
                sumsq = np.dot(centered.T,centered)
            else:
                xbar = sum(np.reshape(d,(-1,D)).sum(0) for d in data) / n
                sumsq = sum(np.dot((np.reshape(d,(-1,D))-xbar).T,(np.reshape(d,(-1,D))-xbar))
                        for d in data)
        else:
            xbar, sumsq = None, None
        return n, xbar, sumsq

```

```python
    @staticmethod
    def _get_weighted_statistics(data,weights,D=None):
        # NOTE: _get_statistics is special case with all weights being 1
        # this is kept as a separate method for speed and modularity
        if isinstance(data,np.ndarray):
            neff = weights.sum()
            if neff > 0:
                D = getdatadimension(data) if D is None else D
                xbar = np.dot(weights,np.reshape(data,(-1,D))) / neff
                centered = np.reshape(data,(-1,D)) - xbar
                sumsq = np.dot(centered.T,(weights[:,na] * centered))
            else:
                xbar, sumsq = None, None
        else:
            neff = sum(w.sum() for w in weights)
            if neff > 0:
                D = getdatadimension(data) if D is None else D
                xbar = sum(np.dot(w,np.reshape(d,(-1,D))) for w,d in zip(weights,data)) / neff
                sumsq = sum(np.dot((np.reshape(d,(-1,D))-xbar).T,w[:,na]*(np.reshape(d,(-1,D))-xbar))
                        for w,d in zip(weights,data))
            else:
                xbar, sumsq = None, None

        return neff, xbar, sumsq
```

def _get_statistics(data,D=none)

観測データ？（未確認）から統計量を計算

- n:データサイズ
- xbar：データ平均
- sumsq：S

オブジェクト呼び出し時は n=0,xbar=None,sumsq=None

where S is the sum of squares matrix about the sample mean,

$$S = \sum_{i=1}^{n} (y_i - \overline{y})(y_i - \overline{y})^T.$$

_get_statisticsはすべての重みが1の場合の特殊例（何の重みかはしらん）
resample()には_get_statisticsを用いているので_get_weighted_statistics**は省略**

basic/pybasicbayes/distributions.py>> Gaussian class 214~252

```python
214     def _posterior_hypparams(self,n,xbar,sumsq):
215         mu_0, sigma_0, kappa_0, nu_0 = self.mu_0, self.sigma_0, self.kappa_0, self.nu_0
216         if n > 0:
217             mu_n = self.kappa_0 / (self.kappa_0 + n) * self.mu_0 + n / (self.kappa_0 + n) * xbar
218             kappa_n = self.kappa_0 + n
219             nu_n = self.nu_0 + n
220             sigma_n = self.sigma_0 + sumsq + ¥
221                     self.kappa_0*n/(self.kappa_0+n) * np.outer(xbar-self.mu_0,xbar-self.mu_0)
222 
223             return mu_n, sigma_n, kappa_n, nu_n
224         else:
225             return mu_0, sigma_0, kappa_0, nu_0
226 
227     def empirical_bayes(self,data):
228         D = getdatadimension(data)
229         self.kappa_0 = 0
230         self.nu_0 = 0
231         self.mu_0 = np.zeros(D)
232         self.sigma_0 = np.zeros((D,D))
233         self.mu_0, self.sigma_0, self.kappa_0, self.nu_0 = ¥
234                 self._posterior_hypparams(*self._get_statistics(data))
235         if (self.mu,self.sigma) == (None,None):
236             self.resample() # intialize from prior
237 
238         return self
239 
240     ### Gibbs sampling
241 
242     def resample(self,data=[]):
243         D = len(self.mu_0)
244         self.mu_mf, self.sigma_mf = self.mu, self.sigma = ¥
245                 sample_niw(*self._posterior_hypparams(*self._get_statistics(data,D)))
246         return self
247 
248     def copy_sample(self):
249         new = copy.copy(self)
250         new.mu = self.mu.copy()
251         new.sigma = self.sigma.copy()
252         return new
```

□ def _posterior_hypparams(self,n,xbar,sumsq)

事後分布用のパラメータ取得：def resample()で用いる

先程の_get_statisticsで得た変数を用いる（n, xbar, sumsq）

- mu_0, kappa_0, nu_0, sigma_0 からmu_n, kappa_n, nu_n, sigma_nを得る（n=0：オブジェクト生成時はmu_0, kappa_0, nu_0, sigma_0 を返す）

$$
\begin{aligned}
\mu_n &= \frac{\kappa_0}{\kappa_0 + n}\mu_0 + \frac{n}{\kappa_0 + n}\overline{y} \\
\kappa_n &= \kappa_0 + n \\
\nu_n &= \nu_0 + n \\
\Lambda_n &= \Lambda_0 + S + \frac{\kappa_0 n}{\kappa_0 + n}(\overline{y} - \mu_0)(\overline{y} - \mu_0)^T,
\end{aligned}
$$

Emergent System Lab

basic/pybasicbayes/distributions.py>> Gaussian class 214~252

```python
214     def _posterior_hypparams(self,n,xbar,sumsq):
215         mu_0, sigma_0, kappa_0, nu_0 = self.mu_0, self.sigma_0, self.kappa_0, self.nu_0
216         if n > 0:
217             mu_n = self.kappa_0 / (self.kappa_0 + n) * self.mu_0 + n / (self.kappa_0 + n) * xbar
218             kappa_n = self.kappa_0 + n
219             nu_n = self.nu_0 + n
220             sigma_n = self.sigma_0 + sumsq + ¥
221                     self.kappa_0*n/(self.kappa_0+n) * np.outer(xbar-self.mu_0,xbar-self.mu_0)
222 
223             return mu_n, sigma_n, kappa_n, nu_n
224         else:
225             return mu_0, sigma_0, kappa_0, nu_0
226 
227     def empirical_bayes(self,data):
228         D = getdatadimension(data)
229         self.kappa_0 = 0
230         self.nu_0 = 0
231         self.mu_0 = np.zeros(D)
232         self.sigma_0 = np.zeros((D,D))
233         self.mu_0, self.sigma_0, self.kappa_0, self.nu_0 = ¥
234                 self._posterior_hypparams(*self._get_statistics(data))
235         if (self.mu,self.sigma) == (None,None):
236             self.resample() # intialize from prior
237 
238         return self
239 
240     ### Gibbs sampling
241 
242     def resample(self,data=[]):
243         D = len(self.mu_0)
244         self.mu_mf, self.sigma_mf = self.mu, self.sigma = ¥
245                 sample_niw(*self._posterior_hypparams(*self._get_statistics(data,D)))
246         return self
247 
248     def copy_sample(self):
249         new = copy.copy(self)
250         new.mu = self.mu.copy()
251         new.sigma = self.sigma.copy()
252         return new
```

- **def emprical_bayes(self,data)**

  _posterior_hypparamsから得たmu_n, kappa_n, nu_n, sigma_nでmu_0, kappa_0, nu_0, sigma_0をそれぞれ更新

- **def resample(self, data=[])**
  - mu, sigmaを，求めた事後分布用パラメータからサンプリング（その際mu_mf, sigma_mfにも代入
  - sample_niw()関数を用いる（次ページ参照）

- **def copy_sample(self)**
  - オブジェクトコピー用？

basic/pybasicbayes/util/stats.py>> sample_niw, sample_invwishart function 80~110

```python
def sample_niw(mu,lmbda,kappa,nu):
    '''
    Returns a sample from the normal/inverse-wishart distribution, conjugate
    prior for (simultaneously) unknown mean and unknown covariance in a
    Gaussian likelihood model. Returns covariance.
    '''
    # code is based on Matlab's method
    # reference: p. 87 in Gelman's Bayesian Data Analysis

    # first sample Sigma ~ IW(lmbda,nu)
    lmbda = sample_invwishart(lmbda,nu)
    # then sample mu | Lambda ~ N(mu, Lambda/kappa)
    mu = np.random.multivariate_normal(mu,lmbda / kappa)

    return mu, lmbda

def sample_invwishart(lmbda,dof):
    # TODO make a version that returns the cholesky
    # TODO allow passing in chol/cholinv of matrix parameter lmbda
    # TODO lowmem! memoize! dchud (eigen?)
    n = lmbda.shape[0]
    chol = np.linalg.cholesky(lmbda)

    if (dof <= 81+n) and (dof == np.round(dof)):
        x = np.random.randn(dof,n)
    else:
        x = np.diag(np.sqrt(stats.chi2.rvs(dof-np.arange(n))))
        x[np.triu_indices_from(x,1)] = np.random.randn(n*(n-1)/2)
    R = np.linalg.qr(x,'r')
    T = scipy.linalg.solve_triangular(R.T,chol.T,lower=True).T
    return np.dot(T,T.T)
```

- **def sample_niw(mu,lmbda,kappa,nu)**

  _posterior_hypparamsから得たパラメータを用いてmu, sigma を得る

  - def sample_invwishart(lmbda,nu)

    $$\Sigma \sim \text{Inv-Wishart}_{\nu_0}(\Lambda_0^{-1})$$

  - mu = np.random.multivariate_normal(mu,lmbda / kappa)

    $$\mu|\Sigma \sim \mathrm{N}(\mu_0, \Sigma/\kappa_0)$$

basic/pybasicbayes/distributions.py>> Gaussian class 253~378

```
254     ### Mean Field
255
256     # NOTE my sumsq is Bishop's Nk*Sk
257
258     def _get_sigma_mf(self):
259         return self._sigma_mf
260
261     def _set_sigma_mf(self,val):
262         self._sigma_mf = val
263         self._sigma_mf_chol = None
264
265     sigma_mf = property(_get_sigma_mf,_set_sigma_mf)
266
267     @property
268     def sigma_mf_chol(self):
269         if self._sigma_mf_chol is None:
270             self._sigma_mf_chol = np.linalg.cholesky(self.sigma_mf)
271         return self._sigma_mf_chol
272
273     def meanfieldupdate(self,data,weights):
274         # update
275         D = len(self.mu_0)
276         self.mu_mf, self.sigma_mf, self.kappa_mf, self.nu_mf = ¥
277                 self._posterior_hypparams(*self._get_weighted_statistics(data,weights,D))
278         self.mu, self.sigma = self.mu_mf, self.sigma_mf/(self.nu_mf - D - 1) # for plotting
279
280     def get_vlb(self):
281         # return avg energy plus entropy, our contribution to the mean field
282         # variational lower bound
283         D = len(self.mu_0)
284         loglmbdatilde = self._loglmbdatilde()
285
286         # see Eq. 10.77 in Bishop
287         q_entropy = -0.5 * (loglmbdatilde + D * (np.log(self.kappa_mf/(2*np.pi))-1)) ¥
288                 + invwishart_entropy(self.sigma_mf,self.nu_mf)
289         # see Eq. 10.74 in Bishop, we aren't summing over K
290         p_avgengy = 0.5 * (D * np.log(self.kappa_0/(2*np.pi)) + loglmbdatilde ¥
291                 - D*self.kappa_0/self.kappa_mf - self.kappa_0*self.nu_mf*¥
292                 np.dot(self.mu_mf -
293                     self.mu_0,np.linalg.solve(self.sigma_mf,self.mu_mf - self.mu_0))) ¥
294                 + invwishart_log_partitionfunction(self.sigma_0,self.nu_0) ¥
295                 + (self.nu_0 - D - 1)/2*loglmbdatilde - 1/2*self.nu_mf*¥
296                 np.linalg.solve(self.sigma_mf,self.sigma_0).trace()
297
298         return p_avgengy + q_entropy
299
300     def expected_log_likelihood(self,x):
301         mu_n, sigma_n, kappa_n, nu_n = self.mu_mf, self.sigma_mf, self.kappa_mf, self.nu_mf
302         D = len(mu_n)
303         x = np.reshape(x,(-1,D)) - mu_n # x is now centered
304         xs = np.linalg.solve(self.sigma_mf_chol,x.T)
305
306         # see Eqs. 10.64, 10.67, and 10.71 in Bishop
307         return self._loglmbdatilde()/2 - D/(2*kappa_n) - nu_n/2 * ¥
308                 inner1d(xs.T,xs.T) - D/2*np.log(2*np.pi)
```

```
310     def _loglmbdatilde(self):
311         # see Eq. 10.65 in Bishop
312         D = len(self.mu_0)
313         chol = self.sigma_mf_chol
314         return special.digamma((self.nu_mf-np.arange(D))/2).sum() ¥
315                 + D*np.log(2) - 2*np.log(chol.diagonal()).sum()
316 #----------------------------------not used in HDP-HSMM end----------------------------------#
317     ### Collapsed
318
319     def log_marginal_likelihood(self,data):
320         n, D = getdatasize(data), len(self.mu_0)
321         return self._log_partition_function(*self._posterior_hypparams(*self._get_statistics(data))) ¥
322                 - self._log_partition_function(self.mu_0,self.sigma_0,self.kappa_0,self.nu_0) ¥
323                 - n*D/2 * np.log(2*np.pi)
324
325     def _log_partition_function(self,mu,sigma,kappa,nu):
326         D = len(mu)
327         chol = np.linalg.cholesky(sigma)
328         return nu*D/2*np.log(2) + special.multigammaln(nu/2,D) + D/2*np.log(2*np.pi/kappa) ¥
329                 - nu*np.log(chol.diagonal()).sum()
330
331     def log_predictive_studentt_datapoints(self,datapoints,olddata):
332         D = len(self.mu_0)
333         mu_n, sigma_n, kappa_n, nu_n = self._posterior_hypparams(*self._get_statistics(olddata,D))
334         return multivariate_t_loglik(datapoints,nu_n-D+1,mu_n,(kappa_n+1)/(kappa_n*(nu_n-D+1))*sigma_n)
335
336     def log_predictive_studentt(self,newdata,olddata):
337         # an alternative computation to the generic log_predictive, which is implemented
338         # in terms of log_marginal_likelihood. mostly for testing, I think
339         newdata = np.atleast_2d(newdata)
340         return sum(self.log_predictive_studentt_datapoints(d,combinedata((olddata,newdata[:i])))[0]
341                         for i,d in enumerate(newdata))
342
343     ### Max likelihood
344
345     # NOTE: could also use sumsq/(n-1) as the covariance estimate, which would
346     # be unbiased but not max likelihood, but if we're in the regime where that
347     # matters we've got bigger problems!
348
349     def max_likelihood(self,data,weights=None):
350         D = getdatadimension(data)
351         if weights is None:
352             n, muhat, sumsq = self._get_statistics(data)
353         else:
354             n, muhat, sumsq = self._get_weighted_statistics(data,weights)
355
356         # this SVD is necessary to check if the max likelihood solution is
357         # degenerate, which can happen in the EM algorithm
358         if n < D or (np.linalg.svd(sumsq,compute_uv=False) > 1e-6).sum() < D:
359             # broken!
360             self.mu = 99999999*np.ones(D)
361             self.sigma = np.eye(D)
362             self.broken = True
363         else:
364             self.mu = muhat
365             self.sigma = sumsq/n
366
367         return self
368
369     def MAP(self,data,weights=None):
370         # max likelihood with prior pseudocounts included in data
371         if weights is None:
372             n, muhat, sumsq = self._get_statistics(data)
373         else:
374             n, muhat, sumsq = self._get_weighted_statistics(data,weights)
375
376         self.mu, self.sigma, _, _ = self._posterior_hypparams(n,muhat,sumsq)
377         return self
```

253~317は変分ベイズ用

317~342は周辺化

343~378はMAP用

今のところ使わないので省略

basic/pybasicbayes/distributions.py>>_GaussianBase class 48~125

```python
class _GaussianBase(object):
    @property
    def params(self):
        return dict(mu=self.mu,sigma=self.sigma)

    ### internals

    def getsigma(self):
        return self._sigma

    def setsigma(self,sigma):
        self._sigma = sigma
        self._sigma_chol = None

    sigma = property(getsigma,setsigma)

    @property
    def sigma_chol(self):
        if self._sigma_chol is None:
            self._sigma_chol = np.linalg.cholesky(self._sigma)
        return self._sigma_chol

    ### distribution stuff

    def rvs(self,size=None):
        size = 1 if size is None else size
        size = size + (self.mu.shape[0],) if isinstance(size,tuple) else (size,self.mu.shape[0])
        return self.mu + np.random.normal(size=size).dot(self.sigma_chol.T)

    def log_likelihood(self,x):
        mu, sigma, D = self.mu, self.sigma, self.mu.shape[0]
        sigma_chol = self.sigma_chol
        bads = np.isnan(np.atleast_2d(x)).any(axis=1)
        x = np.nan_to_num(x).reshape((-1,D)) - mu
        xs = scipy.linalg.solve_triangular(sigma_chol,x.T,lower=True)
        out = -1./2. * inner1d(xs.T,xs.T) - D/2*np.log(2*np.pi) ¥
                - np.log(sigma_chol.diagonal()).sum()
        out[bads] = 0
        return out

```

```python
    ### plotting

    def plot(self,data=None,indices=None,color='b',plot_params=True,label=''):
        from util.plot import project_data, plot_gaussian_projection, plot_gaussian_2D
        if data is not None:
            data = flattendata(data)

        D = self.mu.shape[0]

        if D > 2 and ((not hasattr(self,'plotting_subspace_basis'))
                or (self.plotting_subspace_basis.shape[1] != D)):
            # TODO improve this bookkeeping. need a notion of collection. it's
            # totally potentially broken and confusing to set class members like
            # this!

            subspace = np.random.randn(D,2)
            self.__class__.plotting_subspace_basis = np.linalg.qr(subspace)[0].T.copy()

        if data is not None:
            if D > 2:
                data = project_data(data,self.plotting_subspace_basis)
            plt.plot(data[:,0],data[:,1],marker='.',linestyle=' ',color=color)

        if plot_params:
            if D > 2:
                plot_gaussian_projection(self.mu,self.sigma,self.plotting_subspace_basis,
                        color=color,label=label)
            else:
                plot_gaussian_2D(self.mu,self.sigma,color=color,label=label)

    def to_json_dict(self):
        D = self.mu.shape[0]
        assert D == 2
        U,s,_ = np.linalg.svd(self.sigma)
        U /= np.linalg.det(U)
        theta = np.arctan2(U[0,0],U[0,1])*180/np.pi
        return {'x':self.mu[0],'y':self.mu[1],'rx':np.sqrt(s[0]),'ry':np.sqrt(s[1]),
                'theta':theta}
```

- パラメータ取得部分
- パラメータ材料部分
  - rvs：ランダムにNサンプル
  - loglikelihood
- プロット部分
- データ変換（JSON）

などなど…

Slide Section

Poisson class

basic/distributions.py>> Poisson Duration class 49

```
42 ##########################
43 # Distribution classes #
44 ##########################
45
46 class GeometricDuration(Geometric,DurationDistribution):
47     pass
48
49 class PoissonDuration(_StartAtOneMixin,Poisson,DurationDistribution):
50     pass
51
52 class NegativeBinomialDuration(_StartAtOneMixin,NegativeBinomial,DurationDistribu
53     pass
54
55 class NegativeBinomialFixedRDuration(_StartAtOneMixin,NegativeBinomialFixedR,
56         DurationDistribution):
57     pass
58
59 class NegativeBinomialIntegerRDuration(_StartAtOneMixin,NegativeBinomialIntegerR,
60         DurationDistribution):
61     pass
62
63 class NegativeBinomialFixedRVariantDuration(NegativeBinomialFixedRVariant,
64         DurationDistribution):
65     pass
66
67 class NegativeBinomialIntegerRVariantDuration(NegativeBinomialIntegerRVariant,
68         DurationDistribution):
69     pass
70
```

依存先のPoisson,
DurationDistribution
classを見ていく

basic/pybasicbayes/distributions.py>> Poisson class 1308~1339

```python
1308 class Poisson(GibbsSampling, Collapsed):
1309     '''
1310     Poisson distribution with a conjugate Gamma prior.
1311 
1312     NOTE: the support is {0,1,2,...}
1313 
1314     Hyperparameters (following Wikipedia's notation):
1315         alpha_0, beta_0
1316 
1317     Parameter is the mean/variance parameter:
1318         lmbda
1319     '''
1320     def __init__(self,lmbda=None,alpha_0=None,beta_0=None):
1321         self.lmbda = lmbda
1322 
1323         self.alpha_0 = alpha_0
1324         self.beta_0 = beta_0
1325 
1326         if lmbda is None and None not in (alpha_0,beta_0):
1327             self.resample() # intialize from prior
1328 
1329     @property
1330     def params(self):
1331         return dict(lmbda=self.lmbda)
1332 
1333     @property
1334     def hypparams(self):
1335         return dict(alpha_0=self.alpha_0,beta_0=self.beta_0)
1336 
1337     def log_sf(self,x):
1338         return stats.poisson.logsf(x,self.lmbda)
1339 
1340     def _posterior_hypparams(self,n,tot):
1341         return self.alpha_0 + tot, self.beta_0 + n
1342 
1343     def rvs(self,size=None):
1344         return np.random.poisson(self.lmbda,size=size)
1345 
1346     def log_likelihood(self,x):
1347         lmbda = self.lmbda
1348         x = np.array(x,ndmin=1)
1349         raw = np.empty(x.shape)
1350         raw[x>=0] = -lmbda + x[x>=0]*np.log(lmbda) - special.gammaln(x[x>=0]+1)
1351         raw[x<0] = -np.inf
1352         return raw if isinstance(x,np.ndarray) else raw[0]
```

□ __init__()

- lmbda: ポアソン分布の平均，分散（オブジェクト生成時（None）はresample()
- alpha_0, bata_0:共役事前分布のガンマ分布のハイパーパラメータ

これらは更新されない？（Gaussian classではemprical_bayes()で更新できる）

mean: $\mathbf{E}[X]=\frac{\alpha}{\beta}$

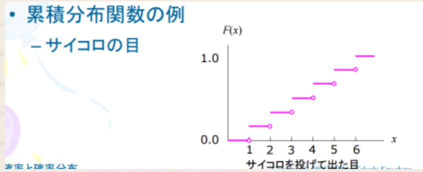
□ def params(): 辞書型パラメータ取得（lmbda）

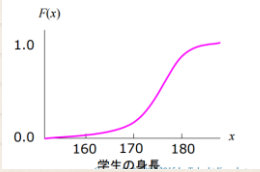
□ def hypparams(): 辞書型ハイパラ取得（alpha_0, beta_0）

□ def log_sf(x)

- survival function(残存時間関数)←CDF:累積分布関数から得る

おそらく隠れ状態がどれだけ継続するか…の根幹部分（今は不使用）





Emergent System Lab

basic/pybasicbayes/distributions.py>> Poisson class 1339~1352

```python
1308 class Poisson(GibbsSampling, Collapsed):
1309     '''
1310     Poisson distribution with a conjugate Gamma prior.
1311
1312     NOTE: the support is {0,1,2,...}
1313
1314     Hyperparameters (following Wikipedia's notation):
1315         alpha_0, beta_0
1316
1317     Parameter is the mean/variance parameter:
1318         lmbda
1319     '''
1320     def __init__(self,lmbda=None,alpha_0=None,beta_0=None):
1321         self.lmbda = lmbda
1322
1323         self.alpha_0 = alpha_0
1324         self.beta_0 = beta_0
1325
1326         if lmbda is None and None not in (alpha_0,beta_0):
1327             self.resample() # intialize from prior
1328
1329     @property
1330     def params(self):
1331         return dict(lmbda=self.lmbda)
1332
1333     @property
1334     def hypparams(self):
1335         return dict(alpha_0=self.alpha_0,beta_0=self.beta_0)
1336
1337     def log_sf(self,x):
1338         return stats.poisson.logsf(x,self.lmbda)
1339
1340     def _posterior_hypparams(self,n,tot):
1341         return self.alpha_0 + tot, self.beta_0 + n
1342
1343     def rvs(self,size=None):
1344         return np.random.poisson(self.lmbda,size=size)
1345
1346     def log_likelihood(self,x):
1347         lmbda = self.lmbda
1348         x = np.array(x,ndmin=1)
1349         raw = np.empty(x.shape)
1350         raw[x>=0] = -lmbda + x[x>=0]*np.log(lmbda) - special.gammaln(x[x>=0]+1)
1351         raw[x<0] = -np.inf
1352         return raw if isinstance(x,np.ndarray) else raw[0]
```

- **def _posterior_hypparams(self,n,tot)**
  - n, tot: 統計量（_get_statistics()から取得：次ページ）

Poisson	λ (rate)	Gamma	$\alpha,\ \beta$ [note 3]	$\alpha+\sum_{i=1}^{n} x_i,\ \beta+n$	α total occurrences in β intervals

tot n

  - 事後分布用パラメータを返す
- **def rvs(self,size=None)**
  - size個ランダムにポアソン分布からサンプル
- **def log_likelihood(self,x)**
  - 尤度取得？（今は不使用）

Emergent System Lab

basic/pybasicbayes/distributions.py>> Poisson class 1354~1404

```python
1354    ### Gibbs Sampling
1355
1356    def resample(self,data=[]):
1357        alpha_n, beta_n = self._posterior_hypparams(*self._get_statistics(data))
1358        self.lmbda = np.random.gamma(alpha_n,1/beta_n)
1359        return self
1360
1361    def _get_statistics(self,data):
1362        if isinstance(data,np.ndarray):
1363            n = data.shape[0]
1364            tot = data.sum()
1365        elif isinstance(data,list):
1366            n = sum(d.shape[0] for d in data)
1367            tot = sum(d.sum() for d in data)
1368        else:
1369            assert isinstance(data,int)
1370            n = 1
1371            tot = data
1372        return n, tot
1373
1374    def _get_weighted_statistics(self,data,weights):
1375        pass # TODO
1376
1377    ### Collapsed
1378
1379    def log_marginal_likelihood(self,data):
1380        return self._log_partition_function(*self._posterior_hypparams(*self._get_statistics(data))) ¥
1381                - self._log_partition_function(self.alpha_0,self.beta_0) ¥
1382                - self._get_sum_of_gammas(data)
1383
1384    def _log_partition_function(self,alpha,beta):
1385        return special.gammaln(alpha) - alpha * np.log(beta)
1386
1387    def _get_sum_of_gammas(self,data):
1388        if isinstance(data,np.ndarray):
1389            return special.gammaln(data+1).sum()
1390        elif isinstance(data,list):
1391            return sum(special.gammaln(d+1).sum() for d in data)
1392        else:
1393            assert isinstance(data,int)
1394            return special.gammaln(data+1)
1395
1396    ### Max likelihood
1397
1398    def max_likelihood(self,data,weights=None):
1399        if weights is None:
1400            n, tot = self._get_statistics(data)
1401        else:
1402            n, tot = self._get_weighted_statistics(data,weights)
1403
1404        self.lmbda = tot/n
1405
```

- **def resample(self,data=[])**
  - _posterior_hypparams()から得たパラメータをalpha_n,beta_nへ格納
  - 共役事前分布のガンマ分布からポアソン分布の平均，分散であるlambdaをサンプル
- **def _get_statistics(self,data)**
  - _posterior_hypparams()で用いる統計量を計算

Poisson	λ (rate)	Gamma	$\alpha,\ \beta$ [note 3]	$\alpha + \sum_{i=1}^{n} x_i,\ \beta + n$	α total occurrences in β intervals

tot n

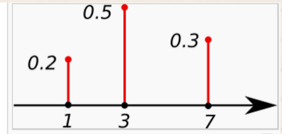
- **_get_weighted_statistics()**
  - 未実装
- **1377~1395：周辺化部分（現在未使用なので省略）**
- **1395~1404：Max likelihood （現在未使用で省略）**

basic/abstractions.py>> DurationDistribution class 12~78

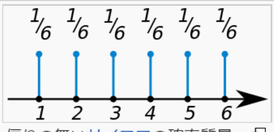
Emergent System Lab

```python
12  class DurationDistribution(Distribution):
13      __metaclass__ = abc.ABCMeta
14
15      # in addition to the methods required by Distribution, we also require a
16      # log_sf implementation
17
18      @abc.abstractmethod
19      def log_sf(self,x):
20          '''
21          log survival function, defined by log_sf(x) = log(P[X ¥gt x]) =
22          log(1-cdf(x)) where cdf(x) = P[X ¥leq x]
23          '''
24          pass
25
26      def log_pmf(self,x):
27          return self.log_likelihood(x)
28
29      # default implementations below
30
31      def pmf(self,x):
32          return np.exp(self.log_pmf(x))
33
34      def rvs_given_greater_than(self,x):
35          tail = self.log_sf(x)
36          trunc = 500
37          while self.log_sf(x+trunc) - tail > -20:
38              trunc *= 1.1
39          logprobs = self.log_pmf(np.arange(x+1,x+trunc+1)) - tail
40          return sample_discrete_from_log(logprobs)+x+1
```

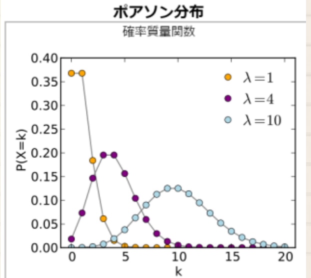
- def log_pmf(),def pmf()
  - （対数）確率質量関数



確率質量関数の例。全ての値は非負であり、その和は1になる。



偏りの無いサイコロの確率質量関数。サイコロの目が出る確率は何れも同じである。



横軸はkパラメータで、確率質量関数は整数値kにおいてのみ定義される。

- def rvs_given_greater_than(): ?

```python
42      def resample_with_truncations(self,data=[],truncated_data=[]):
43          '''
44          truncated_data is full of observations that were truncated, so this
45          method samples them out to be at least that large
46          '''
47          if not isinstance(truncated_data,list):
48              filled_in = np.asarray([self.rvs_given_greater_than(x-1) for x in truncated_data])
49          else:
50              filled_in = np.asarray([self.rvs_given_greater_than(x-1)
51                  for xx in truncated_data for x in xx])
52          self.resample(data=combinedata((data,filled_in)))
53
54      @property
55      def mean(self):
56          trunc = 500
57          while self.log_sf(trunc) > -20:
58              trunc *= 1.5
59          return np.arange(1,trunc+1).dot(self.pmf(np.arange(1,trunc+1)))
60
61      def plot(self,data=None,color='b'):
62          data = flattendata(data) if data is not None else None
63
64          try:
65              tmax = np.where(np.exp(self.log_sf(np.arange(1,1000))) < 1e-3)[0][0]
66          except IndexError:
67              tmax = 2*self.rvs(1000).mean()
68          tmax = max(tmax,data.max()) if data is not None else tmax
69
70          t = np.arange(1,tmax+1)
71          plt.plot(t,self.pmf(t),color=color)
72
73          if data is not None:
74              if len(data) > 1:
75                  plt.hist(data,bins=t-0.5,color=color,normed=len(set(data)) > 1)
76              else:
77                  plt.hist(data,bins=t-0.5,color=color)
```

- def resample_with_truncations()
  - 今は不使用だが今後使用する可能性高いので覚えておく
- def mean()
  - pmfの平均？
- def plot()
  - プロット