Implementing Chib (1998) multiple change-point model

Anh Le

April 30, 2015

```
The line_profiler extension is already loaded. To reload it, use:
  %reload_ext line_profiler
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

Github repo: https://github.com/LaDilettante/changepoint_model

1 Introduction

This project implements Chib (1998) multiple change-point model. This algorithm comprises of two parts:

Given a number of change-points 1. Detect the location of change-points and the parameters of the DGP in the different regimes 2. Calculate the marginal likelihood of the model, which can be used to compare models with different change-points

The innovation of the model is "a formulation of the change-point modle in terms of a latent discrete statement variable that indicates the regime from which a particular observation has been drawn." Formulating the model in this way, we can then use the forward-backward algorithm to sample the latent state vector (and Gibbs sampler to sample the other parameters).

This method requires only $2n$ passes through the data, *regardless of the number of change-points*. This is a big improvement over previous change-point models, in which we have to sample the change-point one at a time.

2 Model Parameterization

Consider a time series Y_n with an unknown m change-points.

Introduce the latent state vector $s_t = \{1, 2, \ldots, m+1\}$, indicating which regime the observation y_t is drawn from. In other words, if $s_t = k$, then $y_t \sim f(y_t|Y_{t-1}, \theta_k)$

The variable s_t is modeled as a discrete Markov process with the transition probability matrix designed to fit the change-point model. Specifically,

$$P = \begin{pmatrix} p_{11} & p_{22} & 0 & \ldots & 0 \\ 0 & p_{22} & p_{23} & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ & \vdots & & & \\ \ldots & \vdots & 0 & p_{mm} & p_{m,m+1} \\ 0 & 0 & 0 & \ldots & 1 \end{pmatrix}$$

Given this transition matrix, the latent state variable s_t can only jump forward, conforming to the change-point model.

In sum, we have 3 following parameters S_n, Θ, P. The next section describe the MCMC scheme to sample each parameter.

3 MCMC scheme

With a prior density $\pi(\Theta, P)$ and data Y_n, we want to estimate the posterior $\pi(\Theta, P|Y_n)$. With the data augmentation using the latent variable s_t, we can use the following full conditionals:

- $S_n|Y_n, \Theta, P$
- $P|S_n$
- $\Theta|Y_n, S_n$

Note how the introduction of S_n simplifies the full conditional of P and Θ

3.1 Simulation of s_t (forward-backward)

Denote S_t the state history up to time t, and S^{t+1} the future from $t+1$ to n. We now sample $p(S_n|Y_n, \Theta, P)$

$$\begin{aligned} p(S_n|Y_n, \Theta, P) =& p(s_{n-1}|Y_n, s_n, \Theta, P)\times & (1)\\ & p(s_{n-2}|Y_n, S^{n-1}, \Theta, P)\times & (2)\\ & \dots & (3)\\ & p(s_1|Y_n, S^2, \Theta, P) & (4)\end{aligned}$$

A typical term from above is in the form $p(s_t|Y_n, S^{t+1}, \Theta, P)$, and can be calculated as following:

$$p(s_t|Y_n, S^{t+1}, \Theta, P) \propto p(s_t|Y_t, \Theta, P)p(s_{t+1}|s_t, P) \quad (5)$$

Key idea - Notice how given s_{t+1} and $p(s_t|Y_t, \Theta, P)$, we can calculate $p(s_t|Y_n, S^{t+1}, \Theta, P)$. This is how we go **backward** after initializing $s_n = m + 1$

We still need $p(s_t|Y_t, \Theta, P)$, which is calculated as:

$$p(s_t = k|Y_t, \Theta, P) = \frac{p(s_t = k|Y_{t-1}, \Theta, P) \times f(y_t|Y_{t-1}, \theta_k)}{\sum_{l=\text{all k's}} p(s_t = l|Y_{t-1}, \Theta, P) \times f(y_t|Y_{t-1}, \theta_l)}$$

and

$$p(s_t = k|Y_{t-1}, \Theta, P) = \sum_{l=k-1}^{k} p_{lk} \times p(s_{t-1} = l|Y_{t-1}, \Theta, P)$$

Key idea: - Notice how given $p(s_{t-1} = l|Y_{t-1}, \Theta, P)$, we can calculate $p(s_t = k|Y_t, \Theta, P)$. This is how we go **forward**, starting by setting $p(s_1|Y_0, \theta) = 1$.

3.2 Simulation of P

Since $P|Y_n, S_n, \Theta$ is independent of Y_n, Θ given S_n, sampling from the full conditional is easy.

Given the prior $p_{ii} \sim Beta(a, b)$, we have the posterior $p_i i|S_n \sim Beta(a + n_{ii}, b + 1)$, with n_{ii} denote the number of one-step transitions from state i to state i in the sequence S_n

3.3 Simulation of Θ

The full conditional $\Theta|Y_n, S_n$ depends on each data model. Given the latent state vector S_n, we have a complete data likelihood, and the posterior update is routine.

4 MCEM and calculating marginal likelihood

To decide how many change points there are in the data, we can fit models with different change points the compare the marginal likelihood of each model.

The marginal likelihood of model M_r is:

$$m(Y_n|M_r) = \frac{f(Y_n|M_r, \Theta^*, P^*)\pi(\Theta^*, P^*|M_r)}{\pi(\Theta^*, P^*|Y_n, M_r)}$$

where Θ^*, P^* could be any point in the parameter space. However, for numerical stability, we choose a high posterior density point, which is the MLE, estimated via Monte Carlo EM.

The posterior ordinate $\pi(\Theta^*, P^*|M_r)$ is separated into two terms:

$$\pi(\Theta^*, P^*|M_r) = \pi(\Theta * |Y_n)\pi(P^*|Y_n, \Theta^*)$$

both of which are estimated from MCMC samples.

5 Implementation

- Each full conditional in a MCMC iteration is a function, grouped into a module ("full_conditionals.py"). This makes it easy to re-use the sampling step for the MCMC and MCEM
- The update for each parameter in the MCEM iteration is a function, grouped into a module ("mcem.py")
- The calculation of each term in the marginal likelihood is a function (in file "ordinate.py")

The MCMC and MCEM samplers then call these modules. To facilitate speed comparison, the sampler can choose to call the regular module or the optimized version (suffixed with "_opt.py"). The sampler function can also specify the DGP (binary or poisson) and the number of change points.

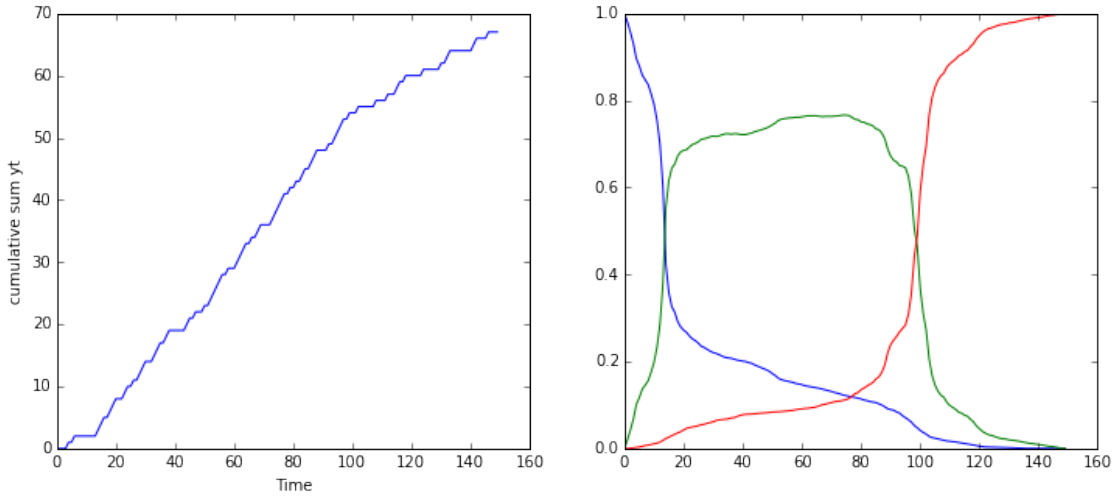
6 Result

We use the algorithm on binary data (simulated) and poisson data (British coal-mining disasters)

6.1 Binary data

We simulate binary data $y_t \sim Bernoulli(\xi_t)$, where

$$f(n) = \begin{cases} \theta_1 = 0.5 & \text{if } t \leq 50 \\ \theta_2 = 0.75 & \text{if } 50 < t \leq 100 \\ \theta_3 = 0.25 & \text{if } 100 < t \leq 150 \end{cases}$$

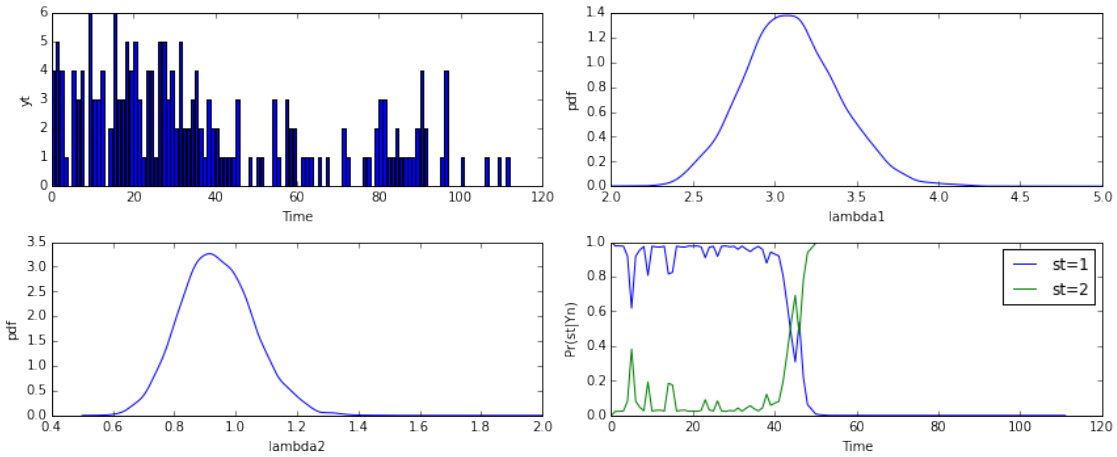


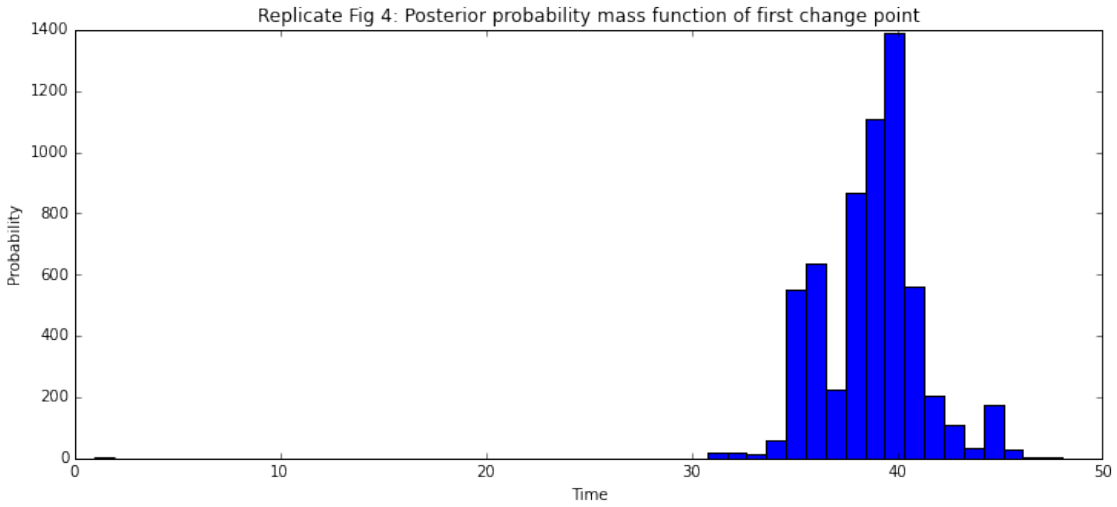
6.2 Poisson data

We use the method on the British coal mining disasters by year. Model the count y_t from a hierarchical Poisson model

$$f(y_t|\xi_t) = \frac{\xi_t^{y_t} \mathrm{e}^{-\xi_t}}{y_t!}$$

Our implementation successfully replicates the author's result (Figure 3 and Figure 4) for the model with one break.



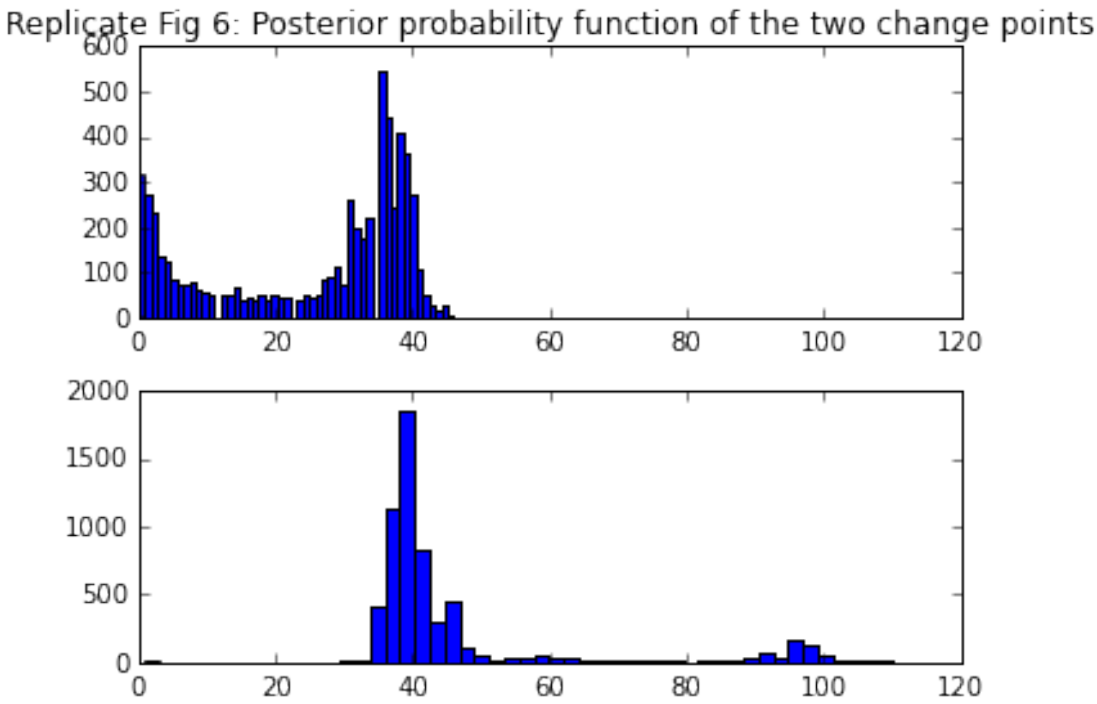


6.3 Marginal likelihood calculation

The result of our Monte Carlo EM and the marginal likelihood function for Poisson data also matches the author's result exactly. Comparing the marginal likelihood of the one break (-178.37) vs two breaks (-179.59), we reach the same conclusion that one break model is a better fit (See Table 2 in the paper for the same conclusion). The location of change-points in the two-break models also matches the author's exactly.

```
Theta_MLE = [ 3.12513647  0.9260591 ]
log-likelihood = -172.181385853
marginal-likelihood = -178.378503257

Theta_MLE = [ 3.14525235  1.09034126  0.30937427]
log-likelihood = -170.63154491
marginal-likelihood = -179.592209806
```



7 Code optimization

Due to the sequential updating in MCMC, we cannot vectorize the entire MCMC step. Instead, we can only vectorize within the sampling of Θ and P. The most time consuming step in each MCMC iteration is the sampling of S_n – however, due to the nature of the forward-backward algorithm, we cannot speed this part up either.

The following demonstration shows that the vectorize version of *Theta_sampling* cuts the time in half (from 0.005s to 0.002s). It is important to point out that sampling Theta takes up only a small part of the total time in the MCMC, however.

- Unoptimized

```
Timer unit: 1e-06 s

Total time: 0.005355 s
File: changepoint/full_conditionals.py
Function: Theta_sampling at line 33

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    33                                           def Theta_sampling(Yn, Sn, model):
    34                                               '''
    35                                               Sample Theta from its full conditional distributio
    36
    37                                               Args:
    38                                                   Yn: n x 1 data vector
    39                                                   Sn: n x 1 latent state vector
```

```
    40
    41                                                 Returns:
    42                                                     a (m + 1) x 1 vector of parameter, one param fo
    43                                                 '''
    44         1            48     48.0      0.9      number_of_regimes = len(np.unique(Sn))
    45         1             5      5.0      0.1      thetas = np.empty(number_of_regimes)
    46
    47         3             8      2.7      0.1      for k in range(1, number_of_regimes + 1):
    48         2          5293   2646.5     98.8          thetas[k - 1] = Theta_conditional(k, Yn, Sn, mo
    49         1             1      1.0      0.0      return thetas
```

- Optimized

```
Timer unit: 1e-06 s

Total time: 0.002913 s
File: changepoint/full_conditionals_opt.py
Function: Theta_sampling at line 32

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    32                                           def Theta_sampling(Yn, Sn, model):
    33                                               '''
    34                                               Sample Theta from its full conditional distributio
    35
    36                                               Args:
    37                                                   Yn: n x 1 data vector
    38                                                   Sn: n x 1 latent state vector
    39
    40                                               Returns:
    41                                                   a (m + 1) x 1 vector of parameter, one param fo
    42                                               '''
    43         1            52     52.0      1.8      number_of_regimes = len(np.unique(Sn))
    44         1             1      1.0      0.0      m = number_of_regimes - 1
    45
    46         1            74     74.0      2.5      Nks = Nk(Sn)
    47         1            71     71.0      2.4      Uks = Uk(Yn, Sn)
    48
    49         1             2      2.0      0.1      if model == "binary":
    50                                                   # Prior
    51                                                   a = 2 ; b = 2
    52                                                   return st.beta(a + Uks, b + Nks - Uks).rvs()
    53         1             1      1.0      0.0      elif model == "poisson":
    54                                                   # Different priors based on number of breaks
    55         1             0      0.0      0.0          if m == 1:
    56         1             1      1.0      0.0              a = 2 ; b = 1
    57                                                   elif m == 2:
    58                                                       a = 3 ; b = 1
    59         1          2711   2711.0     93.1          return st.gamma(a + Uks, scale=(1.0 / (b + Nks)
```

Similar speed-up is achieved in the updating of Theta and P in the Monte Carlo EM. Total time (73.6s -> 67.3s), Theta-M-step (0.63s -> 0.03s). Note how sampling S_n in the E-step takes up the lion-share of computation time ($> 98\%$)

- Unoptimized

```
Timer unit: 1e-06 s

Total time: 0.627406 s
File: changepoint/mcem.py
Function: theta_mstep at line 20

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    20                                           def theta_mstep(k, Yn, Sns, model):
    21                                               '''
    22                                               Calculate theta_k, using N samples of Sn
    23                                               '''
    24       200          292      1.5      0.0      N = Sns.shape[1]
    25
    26       200          532      2.7      0.1      Uks = np.zeros(N)
    27       200          294      1.5      0.0      Nks = np.zeros(N)
    28
    29     30220        19346      0.6      3.1      for N_ in range(N):
    30     30020       308828     10.3     49.2          Uks[N_] = Uk(k, Yn, Sns[:, N_])
    31     30020       296273      9.9     47.2          Nks[N_] = Nk(k, Sns[:, N_])
    32
    33       200          129      0.6      0.0      if model == "binary":
    34                                                   return 1.0 * Uks.sum() / Nks.sum()
    35       200          116      0.6      0.0      if model == "poisson":
    36       200         1596      8.0      0.3          return 1.0 * Uks.sum() / Nks.sum()

Total time: 73.585 s
File: changepoint/mcem_sampler.py
Function: mcem_sampler at line 7

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     7                                           def mcem_sampler(Yn, model, m, mcem_module, tol=None):
     8                                               '''
     9                                               Calculate the MLE of Theta and P, using Monte Carl
    10
    11                                               We use EM because the latent state vector is not ob
    12                                               We use Monte Carlo method because the Q function in
    13
    14                                               Args
    15                                                   Yn: array, time series data
    16                                                   model: string, e.g. "binary", "poisson"
    17                                                   m: int, number of change points
    18                                                   mcem: module, e.g. mcem, mcem_opt
    19                                                       Use the regular of optimized version
    20                                               '''
    21                                               # Initialize
    22         1            5      5.0      0.0      n = len(Yn) ; m = m
    23         1          162    162.0      0.0      P = init.P(m + 1)
    24
    25         1            4      4.0      0.0      P[0, 0] = 0.8 # Paper
    26         1            2      2.0      0.0      P[0, 1] = 1 - 0.8
    27
    28         1            2      2.0      0.0      if (m == 2) and (model == "poisson"):
```

```
    29                                                #     Without giving a high chance of staying at s
    30                                                P[1, 1] = 0.9
    31                                                P[1, 2] = 1 - 0.9
    32
    33                                            # Initialize Theta
    34         1            2      2.0      0.0    if model == "binary":
    35                                                Theta = np.repeat(0.5, m + 1)
    36         1            2      2.0      0.0    elif model == "poisson":
    37                                                #Theta = np.repeat(2, m + 1)
    38         1           31     31.0      0.0        Theta = 2 + np.random.rand(m + 1)
    39
    40                                            # N is the number of Sn sample
    41                                            # According to paper, start N = 1 and increases ov
    42         1           52     52.0      0.0    Ns = np.linspace(1, 300, 10).astype(np.int64)
    43         1            6      6.0      0.0    Thetas = np.empty((m + 1, 100))
    44         1            4      4.0      0.0    Ps = np.empty((m + 1, m + 1, 100))
    45
    46                                            # Start 100 MCEM steps
    47         1            2      2.0      0.0    i = 0
    48       101          125      1.2      0.0    while i < 100:
    49       100          174      1.7      0.0        N = Ns[i / 10]
    50
    51                                                # E-step
    52       100     72758913 727589.1     98.9        Sns = mcem_module.S_estep(N, Yn, Theta, P, mode
    53                                                #if i <= 3:
    54                                                #    print Sns
    55
    56                                                # M-step
    57       100          254      2.5      0.0        Theta_old = Theta
    58       100       682949   6829.5      0.9        Theta = mcem_module.Theta_mstep(Yn, Sns, model=
    59       100       141408   1414.1      0.2        P = mcem_module.P_mstep(Sns)
    60
    61                                                # Store result across iterations
    62       100          312      3.1      0.0        Thetas[:, i] = Theta
    63       100          306      3.1      0.0        Ps[:, :, i] = P
    64
    65
    66                                                #Stop condition based on convergence
    67       100          119      1.2      0.0        if (tol is not None) and (np.allclose(Theta, T
    68                                                    break
    69
    70       100          120      1.2      0.0        i += 1
    71
    72         1            1      1.0      0.0    return Thetas, Ps
```

- Optimized

```
Timer unit: 1e-06 s

Total time: 0.029493 s
File: changepoint/mcem_opt.py
Function: theta_mstep at line 15

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
```

```
    15                                           def theta_mstep(k, Yn, Sns, model):
    16                                               '''
    17                                               Calculate theta_k, using N samples of Sn. Vectorize
    18                                               '''
    19       200        17869     89.3     60.6      Uks = np.sum(Yn[:, np.newaxis] * (Sns == k), axis=0
    20       200         9773     48.9     33.1      Nks = np.sum(Sns == k, axis=0)
    21
    22       200          159      0.8      0.5      if model == "binary":
    23                                                   return 1.0 * Uks.sum() / Nks.sum()
    24       200           97      0.5      0.3      if model == "poisson":
    25       200         1595      8.0      5.4          return 1.0 * Uks.sum() / Nks.sum()

Total time: 67.3543 s
File: changepoint/mcem_sampler.py
Function: mcem_sampler at line 7

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     7                                           def mcem_sampler(Yn, model, m, mcem_module, tol=None):
     8                                               '''
     9                                               Calculate the MLE of Theta and P, using Monte Carlo
    10
    11                                               We use EM because the latent state vector is not ob
    12                                               We use Monte Carlo method because the Q function in
    13
    14                                               Args
    15                                                   Yn: array, time series data
    16                                                   model: string, e.g. "binary", "poisson"
    17                                                   m: int, number of change points
    18                                                   mcem: module, e.g. mcem, mcem_opt
    19                                                       Use the regular of optimized version
    20                                               '''
    21                                               # Initialize
    22         1            4      4.0      0.0      n = len(Yn) ; m = m
    23         1          151    151.0      0.0      P = init.P(m + 1)
    24
    25         1            5      5.0      0.0      P[0, 0] = 0.8 # Paper
    26         1            2      2.0      0.0      P[0, 1] = 1 - 0.8
    27
    28         1            2      2.0      0.0      if (m == 2) and (model == "poisson"):
    29                                                   # Without giving a high chance of staying at st
    30                                                   P[1, 1] = 0.9
    31                                                   P[1, 2] = 1 - 0.9
    32
    33                                               # Initialize Theta
    34         1            2      2.0      0.0      if model == "binary":
    35                                                   Theta = np.repeat(0.5, m + 1)
    36         1            2      2.0      0.0      elif model == "poisson":
    37                                                   #Theta = np.repeat(2, m + 1)
    38         1           34     34.0      0.0          Theta = 2 + np.random.rand(m + 1)
    39
    40                                               # N is the number of Sn sample
    41                                               # According to paper, start N = 1 and increases ove
    42         1           52     52.0      0.0      Ns = np.linspace(1, 300, 10).astype(np.int64)
```

43	1	7	7.0	0.0	`Thetas = np.empty((m + 1, 100))`
44	1	3	3.0	0.0	`Ps = np.empty((m + 1, m + 1, 100))`
45					
46					`# Start 100 MCEM steps`
47	1	1	1.0	0.0	`i = 0`
48	101	125	1.2	0.0	`while i < 100:`
49	100	184	1.8	0.0	`    N = Ns[i / 10]`
50					
51					`    # E-step`
52	100	67308444	673084.4	99.9	`    Sns = mcem_module.S_estep(N, Yn, Theta, P, mode`
53					`    #if i <= 3:`
54					`    #    print Sns`
55					
56					`    # M-step`
57	100	270	2.7	0.0	`    Theta_old = Theta`
58	100	34774	347.7	0.1	`    Theta = mcem_module.Theta_mstep(Yn, Sns, model=`
59	100	9408	94.1	0.0	`    P = mcem_module.P_mstep(Sns)`
60					
61					`    # Store result across iterations`
62	100	309	3.1	0.0	`    Thetas[:, i] = Theta`
63	100	294	2.9	0.0	`    Ps[:, :, i] = P`
64					
65					
66					`    #Stop condition based on convergence`
67	100	120	1.2	0.0	`    if (tol is not None) and (np.allclose(Theta, T`
68					`        break`
69					
70	100	124	1.2	0.0	`    i += 1`
71					
72	1	1	1.0	0.0	`return Thetas, Ps`

8 Comparison with existing packages

We compare our algorithm with:

- The R implementation in package **MCMCpack**, which produces identical result to ours (See Figures below). (In fact, the marginal likelihood calculation is more similar to ours than to the original paper).
- Another Bayesian change point model, by Barry and Hartigan (1993), which gives similar conclusion. This model detect the number of regimes in the data, their means and variances. The result also suggests one break point, with two regimes having means $= [3, 1]$, similar to our Poisson model (See Figures below).

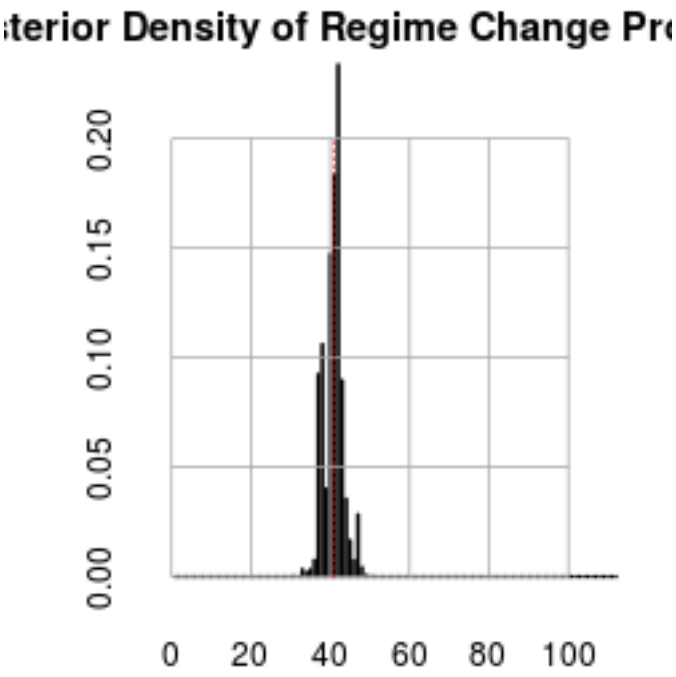
These two packages in R / C++ implementation run a lot faster than our implementation however (2.83s and 1.86s, compared to > 1 min in our implementation).

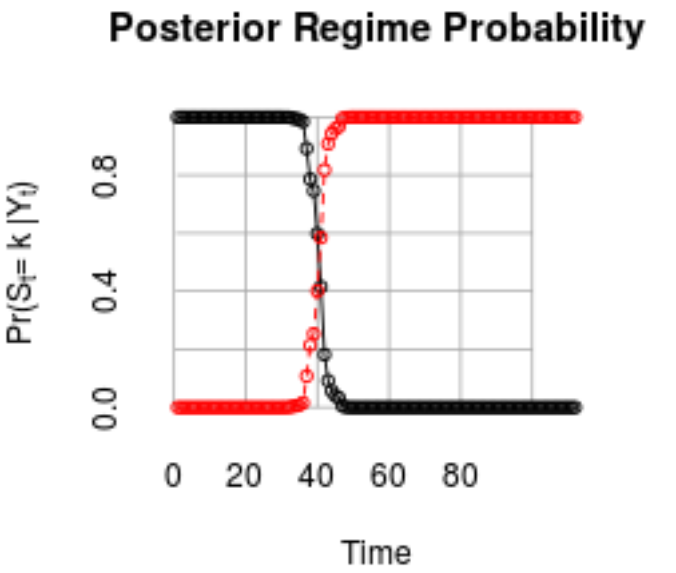
```
1 loops, best of 3: 2.82 s per loop
```

```
1 loops, best of 3: 1.86 s per loop
```

8.0.1 MCMCpack implementation of Chib (1998)

```
[1] "log-likelihood -172.194701317687"
[1] "log-marginal-likelihood -178.354928104687"
```


sterior Density of Regime Change Pr
0.00
0.05
0.10
0.15
0.20
0
20
40
60
80
100


Posterior Regime Probability
Pr(S_t= k |Y_t)
0.0
0.4
0.8
0
20
40
60
80
Time

8.0.2 bcp Implementation of Barry and Hartigan (1993)