

School of Engineering and Applied Science (SEAS), Ahmedabad University

Probability and Stochastic Processes (MAT 277)

Special Assignment Report

Group Name: hn-cli-12

Team Members:

- Name of Team Member-1(Roll no): AU2140029 – Kashish Jethmalani
- Name of Team Member-2(Roll no): AU2140112 - Vishwa Joshi
- Name of Team Member-3(Roll no): AU2140117 - Srushti Thakar
- Name of Team Member-4(Roll no): AU2140213- Vanaja Agarwal

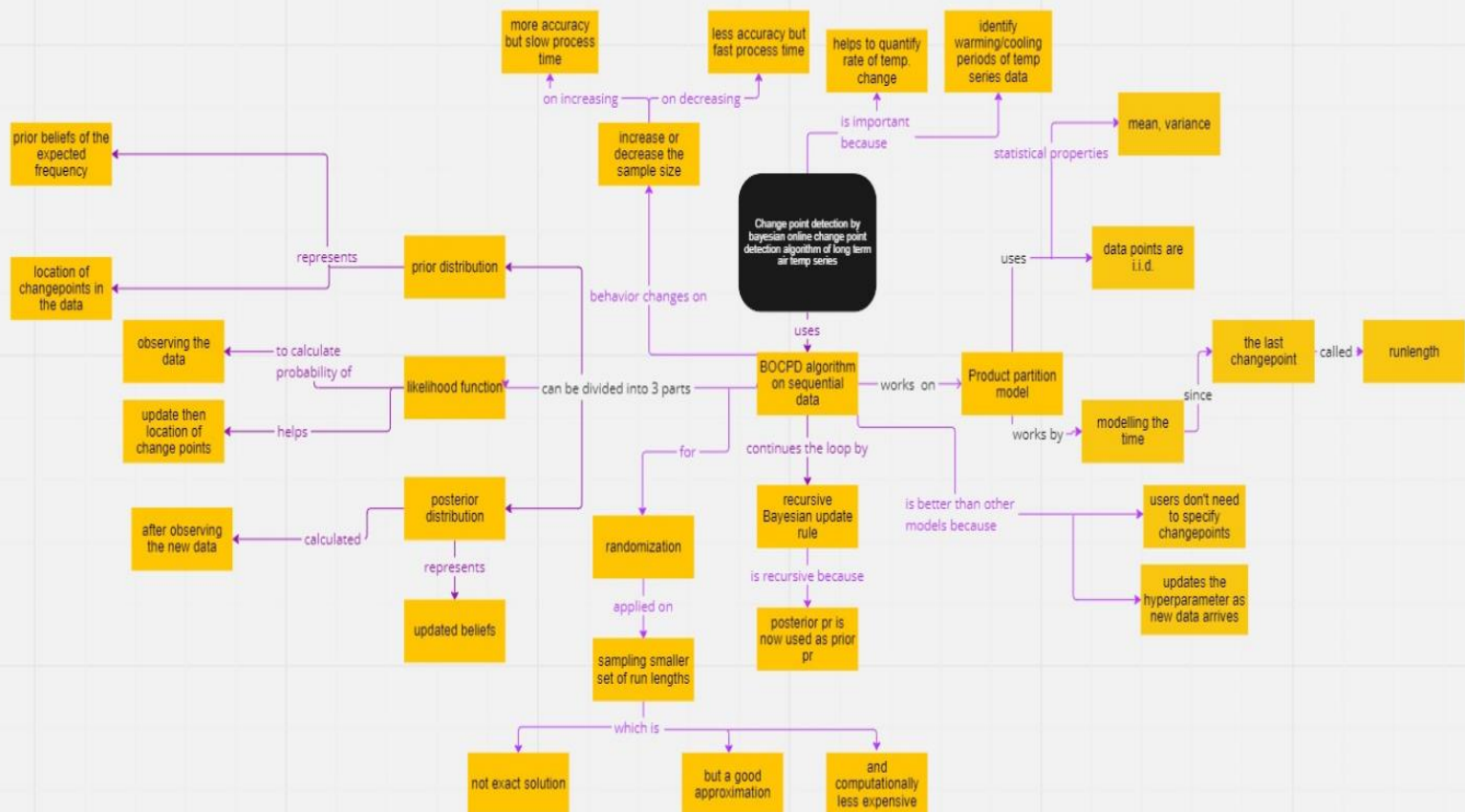
Algorithm: Bayesian Online Changepoint Detection(BOCPD)

Project Title: Changepoint detection in temperature series.

I. Team Activity Learning and Concept Map:

Working on changepoint detection in temperature series using BOCPD provided several valuable team activity learning opportunities. Working on a real-world problem like changepoint detection in temperature series required a team effort to come up with an effective solution. Our team brainstormed ideas, discussed different approaches, and collaborated to implement the solution. We had to work with data in order to make probabilistic inferences. Thus, the assignment developed our data analysis skills as we were working with real temperature data, exploring it, and making sense of it. A lot of statistical modeling was also involved in learning about different probability distributions, Bayesian inference, and hypothesis testing. Therefore, we gained hands-on experience with modeling and fitting statistical models to real-world data. We even had to learn how to use Python to implement Bayesian inference algorithms, visualize data, and perform data analysis. Later on, we had to present our findings and results while making the presentation videos, which helped us improve our communication and presentation skills. Overall, the project was an excellent opportunity for the team to develop

important skills that are applicable in many different fields, including statistics, data science, machine learning, and more.



II. Background and Motivation

Background:

Change-point detection is a method used to identify a point in a time series where the properties of the series change abruptly. Change-points can occur for various reasons, including changes in the underlying process generating the data, changes in the measurement instruments or methods, or changes in external factors that affect the data. Temperature series data can be analyzed using change-point detection to identify the points in time where the temperature trends change significantly. This can be useful for detecting climate change signals, identifying periods of warming or cooling, and assessing the impact of human activities on the climate.

There are various methods for performing change-point detection in temperature series data, including parametric methods such as the Bayesian Change-Point Analysis, non-parametric methods such as the Cumulative Sum (CUSUM) method, and model-based methods such as the Segment Neighbors approach. The choice of method depends on the nature of the data and the research question being addressed. We will be using the Bayesian Online Changepoint Detection method for identifying the changepoints.

Motivation:

One of the main motivations for change-point detection in temperature series is to identify the onset of global warming and to quantify the rate and magnitude of temperature changes over time. This is important for understanding the impact of human activities on the climate and for developing effective strategies for mitigating climate change. Temperature series data can exhibit natural variability due to factors such as El Niño and La Niña events, volcanic eruptions, and solar radiation. Change-point detection can help distinguish between natural variability and long-term trends in the data.

Change-point detection can be used to identify periods of warming or cooling in temperature series data, which can be useful for trend analysis and for predicting future temperature trends. Change-point detection can help identify changes in the frequency and intensity of extreme temperature events, such as heat waves and cold spells. This information is important for assessing the impacts of climate change on human health, agriculture, and natural ecosystems.

Overall, change-point detection in temperature series data is a valuable tool for understanding the dynamics of the climate system, for detecting changes in the climate due to natural and human factors, and for developing effective strategies for mitigating the impacts of climate change.

III. Algorithm Description

Algorithm: Bayesian Online Changepoint Detection

Inputs:

data: sequence of observed data points

prior_distribution: prior probability distribution over the number of changepoints

likelihood_function: function that computes the likelihood of a segment of data given the number of changepoints

hazard_function: function that computes the hazard rate, i.e., the probability of a changepoint at any given time point

Outputs:

changepoints: list of detected changepoints

Algorithm:

```
# Initialize the algorithm
```

```
t = 0
```

```
prior = prior_distribution()
```

```
posterior = prior
```

```
changepoints = []
```

```
# Loop through the data
```

```
for x in data:
```

```
    t += 1
```

```
    # Compute the predictive distribution
```

```
    predictive = 0
```

```
    for k in range(t):
```

```
        likelihood = likelihood_function(data[k:t])
```

```
        predictive += posterior[k] * likelihood
```

```
    predictive *= hazard_function(t)
```

```

# Compute the posterior distribution
posterior = [0] * t
for k in range(t):
    likelihood = likelihood_function(data[k:t])
    posterior[k] = posterior[k-1] + likelihood * prior[k-1]
posterior.append(predictive)
posterior /= sum(posterior)

# Update the prior distribution
prior = posterior

# Check for a changepoint
if max(posterior) > threshold:
    changepoint = argmax(posterior)
    changepoints.append(changepoint)

# Return the list of detected changepoints
return changepoints

```

Here's a breakdown of the steps in the BOCPD algorithm:

1. Initialize the algorithm by setting the current time t to 0, the prior distribution over the number of changepoints to the specified `prior_distribution`, the posterior distribution to the same prior, and an empty list of detected changepoints.
2. Loop through the data one point at a time, incrementing t by 1 each time.
3. Compute the predictive distribution, which is the distribution of the next data point given all the previous data points and the current posterior distribution over the number of changepoints.

4. Update the posterior distribution using the recursive Bayesian update rule, which multiplies the likelihood of each segment of the data by the previous posterior distribution and sums over all possible numbers of changepoints.
5. Update the prior distribution to be the same as the posterior distribution for the next iteration.
6. Check if the maximum value of the posterior distribution exceeds a specified threshold. If so, a changepoint is detected and its index is added to the list of detected changepoints.
7. Continue looping through the data until all points have been processed.
8. Return the list of detected changepoints.
9. Note that the `likelihood_function` and `hazard_function` are problem-specific and need to be defined based on the particular application of BOCPD. Also, the threshold parameter is a tuning parameter that needs to be set based on the desired trade-off between false positives and false negatives.

–

IV. Application

The Bayesian Online Change Point Detection (BOCPD) algorithm is a statistical technique that is used to identify change points in a data stream. This algorithm has a wide range of applications in fields such as finance, manufacturing, engineering, and environmental monitoring.

(i)Finance:One of the main applications of the BOCPD algorithm is in the field of finance. This algorithm can be used to identify changes in stock prices, bond yields, and other financial data. For example, the BOCPD algorithm can be used to identify changes in the volatility of stock prices, which can help investors make better decisions about when to buy or sell stocks.

(ii)Manufacturing: The BOCPD algorithm can also be used in the field of manufacturing to detect changes in the production process. For example, the algorithm can be used to identify changes in the rate of defects in a production line. By detecting these changes early, manufacturers can take corrective action to prevent further defects and improve the overall quality of their products.

(iii)Engineering: The BOCPD algorithm can be used in a variety of engineering applications, such as monitoring the performance of machines and detecting faults in complex systems. For example, the algorithm can be used to identify changes in the vibration patterns of a machine, which can be indicative of a fault. By detecting these changes early, engineers can take action to prevent further damage to the machine.

(iv)Environmental monitoring: The BOCPD algorithm can also be used to monitor environmental data, such as air pollution levels and water quality. For example, the algorithm can be used to identify changes in the concentration of pollutants in the air or water, which can help environmentalists and policymakers take action to address these issues.

(v)Medical applications: The BOCPD algorithm can be used in medical applications to identify changes in patient data, such as heart rate, blood pressure, and other vital signs. For example, the algorithm can be used to detect changes in a patient's heart rate, which can be indicative of a heart attack or other medical emergency. By detecting these changes early, doctors can take action to save the patient's life.

(vi)Internet of Things (IoT): The BOCPD algorithm can be used in the field of IoT to detect changes in sensor data. For example, the algorithm can be used to detect changes in temperature or humidity levels, which can be indicative of a problem with a heating or cooling system. By detecting these changes early, homeowners or building managers can take action to prevent further damage.

(vii)Traffic management: The BOCPD algorithm can also be used in the field of traffic management to detect changes in traffic patterns. For example, the algorithm can be used to identify changes in the flow of traffic on a highway, which can be indicative of an accident or other traffic problem. By detecting these changes early, traffic managers can take action to prevent further delays and improve traffic flow.

By detecting changes in data streams early, the BOCPD algorithm can help businesses, governments, and individuals take action to prevent further damage and improve overall performance. As more data becomes available through the Internet of Things and other sources, the BOCPD algorithm is likely to become even more important in the years ahead.

V. Mathematical Analysis:

BOCPD is based on Bayesian inference, which is a mathematical framework for updating our beliefs about an unknown quantity as we collect more data. In the case of BOCPD, the unknown quantity is the location of the changepoints in the data sequence.

To detect changepoints in real-time, BOCPD uses a sequential Bayesian model that updates its beliefs about the location of the changepoints as new data is observed. The model assumes that the data is generated from a set of probability distributions, and the goal is to estimate the parameters of these distributions as well as the location of the changepoints. The model is based on the following equations:

(i) Prior distribution:

The Prior distribution is a probability distribution that is specified at the beginning of the Bayesian Online Changepoint Detection (BOCPD) algorithm. The Prior distribution represents our prior beliefs about the expected frequency and location of changepoints in the data. In simpler terms, let's say you're trying to detect changes or anomalies in a sequence of data over time, and you have some prior knowledge about the expected location and frequency of these changes. The Prior distribution allows you to incorporate this prior knowledge into the BOCPD algorithm. The various variables that the prior distribution generally has are as follows:

(a) mean0 : This is the mean of the prior distribution over the unknown mean of the Gaussian. It represents our belief about the likely value of the mean before we have seen any data.

(b) var0 : This is the variance of the prior distribution over the unknown mean of the Gaussian. It represents our uncertainty about the true value of the mean before seeing any data.

(c) varx : This is the variance of the Gaussian distribution that generates the observed data. It represents the noise in the data and is assumed to be known.

mean_params : This is a NumPy array that contains the current estimate of the mean of the Gaussian. Initially, it contains the prior mean mean0 .

(d) prec_params : This is a NumPy array that contains the current estimate of the precision (inverse variance) of the Gaussian. Initially, it contains the inverse of the prior variance $1/\text{var0}$.

(ii) Likelihood function:

The Likelihood function is a mathematical function used in the BOCPD algorithm to calculate the probability of observing the data up to a certain point, given the location of the changepoints.

In simpler terms, the Likelihood function tells us how likely it is to observe the data we have collected so far, assuming that the data is generated from a set of probability distributions with a certain number of changepoints.

The Likelihood function is denoted by $p(X_{1:t}|C_1)$, where $X_{1:t}$ is the data observed up to time t , and C_1 is the location of the first changepoint. The Likelihood function takes into account the probability distributions that generate the data, and calculates the probability of the data we have collected so far given these distributions and the location of any changepoints. The Likelihood function is an important component of the BOCPD algorithm, as it helps update our beliefs about the location of the changepoints as new data is collected.

(iii) Posterior distribution:

The Posterior distribution is a probability distribution that is calculated by the BOCPD algorithm after observing new data. The Posterior distribution represents our updated beliefs about the location and number of changepoints, given the new data.

The Posterior distribution is denoted by $p(C_{1:t}|X_{1:t})$, where $C_{1:t}$ is the location of all changepoints up to time t , and $X_{1:t}$ is the data observed up to time t . The Posterior distribution takes into account the Prior distribution (our initial beliefs about the expected frequency and location of changepoints) and the Likelihood function (the probability of observing the data given the location of the changepoints) to calculate our updated beliefs about the location and number of changepoints.

In practical terms, the Posterior distribution tells us the probability that a changepoint has occurred at each time point, given the data we have observed so far. By updating our beliefs about the location and number of changepoints as new data is collected, we can make more accurate predictions and detect changes or anomalies in the data more effectively.

(iv) Recursive Bayesian update rule:

The Recursive Bayesian update rule tells us how to update our beliefs about the location and number of changepoints as new temperature measurements (or other data) are taken. This allows us to detect changes or anomalies in the data more effectively over time. The Recursive Bayesian Update Rule is "recursive" because we use the posterior probability distribution at the previous time step as our prior probability for the current time step. This allows us to track changes in the temperature over time.

To detect a change in the temperature, we look for "changepoints" in the data where the statistical properties (e.g., mean, variance) seem to change. Overall, the Recursive Bayesian Update Rule and BOCPD provide a way to analyze sequential data in real-time and detect changes in the underlying statistical properties over time.

(v) Detecting changepoints:

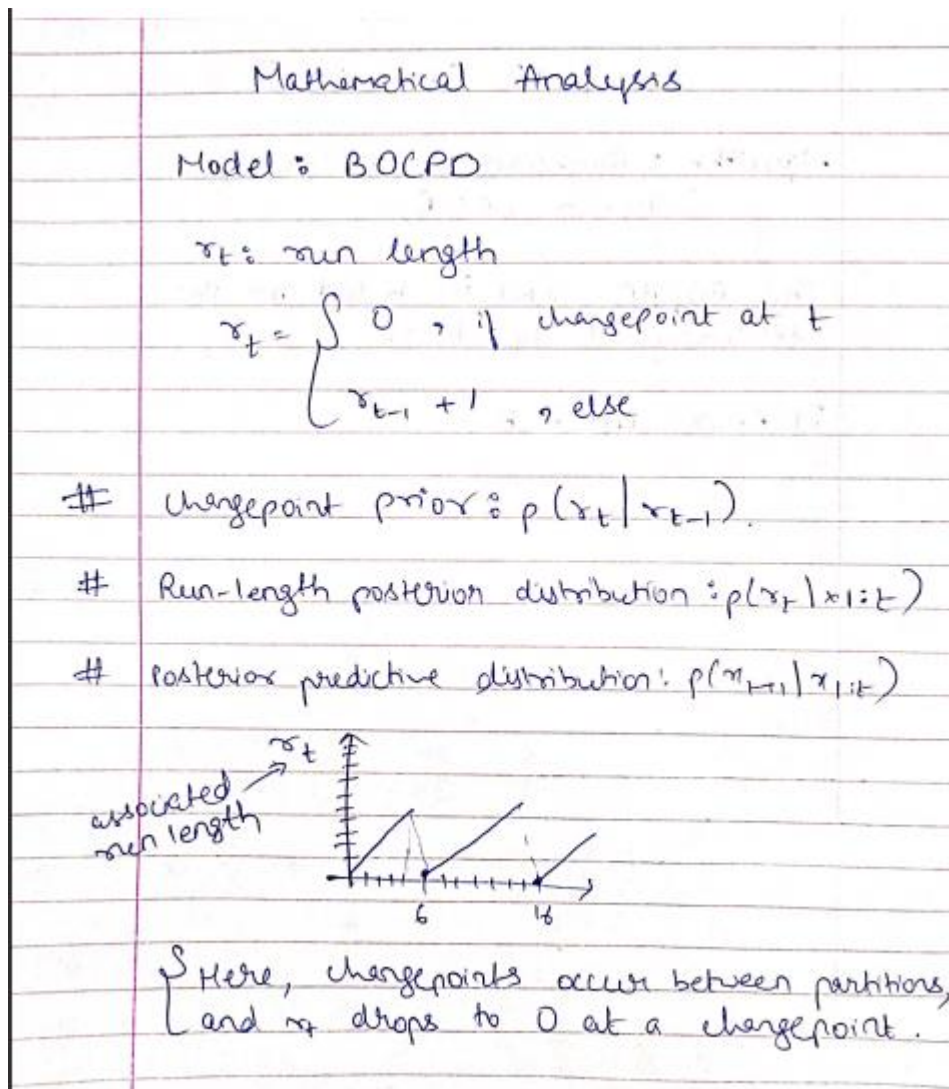
The algorithm detects a changepoint at time t if the posterior probability of the previous changepoint location falls below a certain threshold. If a changepoint is detected, the algorithm updates the prior distribution to reflect the new belief about the expected frequency and location of changepoints.

Randomization in the BOCPD algorithm:

In the context of Bayesian online change detection, a run length refers to the number of sequential data points before a change point occurs. For example, consider a time series of stock prices. The run length represents the number of consecutive time points where the stock prices have not changed significantly before a change occurs. If there is a change in the stock price, the run length will reset to 0, and the process will start again. In the above code, there is a step where the run length distribution is determined. This means that for each time step, the algorithm tries to determine how long the current run of data is. However, calculating the run length distribution can be computationally expensive, especially for large datasets.

To make this calculation more computationally efficient, the code uses a randomized algorithm. Instead of considering all possible run lengths, the algorithm randomly samples a smaller set of possible run lengths. It then calculates the run length distribution based on this smaller set of

samples, which is computationally faster. While this approach may not provide an exact solution, it can still provide a good approximation of the true distribution.



The run length, denoted as r_t , is a key variable in this method. It represents the number of observations since the last changepoint. At any given time, the run length can take one of two possible values: either it is zero, indicating that the most recent observation was generated from a new probability distribution due to a changepoint, or it is equal to the previous run length r_{t-1} plus one, indicating that the most recent observation was generated from the same distribution as the previous observations.

The transition probabilities between different run lengths are modelled by the changepoint prior $p(r_t | r_{t-1})$. This prior captures the assumption that the probability of a changepoint occurring at any given time is low, and that the probability of a changepoint occurring does not depend on

any of the observations that came before it, but only on the current run length. This means that the run length is conditionally independent of all the observations given the previous run length.

* Recursive RL posterior estimation \rightarrow

$$p(\hat{\pi} | D) = \int \overbrace{p(\hat{\pi} | \theta)}^{\text{model}} \overbrace{p(\theta | D)}^{\text{posterior}} d\theta$$

D : set,
 N : iid observations, θ : model parameters

Posterior predictive: (by marginalizing out the run length r_t)

$$p(x_{t+1} | x_{1:t}) = \sum_{r_t} p(x_{t+1}, r_t | x_{1:t})$$

$$= \sum_{r_t} \underbrace{p(x_{t+1} | r_t, x^{(r_t)})}_{\text{UPM predictive}} \overbrace{p(r_t | x_{1:t})}^{\text{RL posterior}}$$

$\therefore \text{Eq}^n$ now can be written as:

$$p(x_{t+1} | x_{1:t}) = \sum_{l=0}^t p(x_{t+1} | r_t=l, x_{(t-l):t}) p(r_t=l | x_{1:t})$$

In Bayesian online changepoint detection, we want to predict the next data point, x_{t+1} , given the observations up to time t , denoted as $x_{1:t}$. To do this, we compute the posterior predictive distribution, which is the probability of x_{t+1} given the observed data up to time t . This distribution is computed by marginalizing out the run length r_t , which is the time since the last changepoint occurred.

The UPM predictive term is the product of two probabilities. The first probability, $p(x_{t+1} | r_t, \{x\}_{1:t})$, is the probability of the next data point x_{t+1} given the run length r_t and the observations associated with that run length $\{x\}_{1:t}$. In other words, this term models the probability distribution of the next data point given that a changepoint occurred l time steps ago and the current run length is r_t .

The second probability, $p(r_t | \{x\}_{1:t})$, is the posterior probability of the run length r_t given all the observations up to time t . This term takes into account all possible run lengths up to time t and their corresponding probabilities based on the observations. In other words, this term models the probability of the current run length r_t given all the observations up to time t .

The summation over all possible run lengths r_t in the equation marginalizes the uncertainty in the run length, effectively averaging over all possible run lengths weighted by their posterior probabilities. This results in the posterior predictive probability of the next data point x_{t+1} , which takes into account the uncertainty in both the run length and the next data point given the observations up to time t .

General form for any member of the exponential family is:

$$p(x|\eta) = h(x)g(\eta)\exp\{\eta u(x)\}.$$

η : natural parameters
 $h(x)$: underlying measure..
 $u(x)$: sufficient statistic of the data
 $g(\eta)$: normalizer.

\Rightarrow Recursive RL posterior estimation

$$\begin{aligned} p(r_t | x_{1:t}) &= \frac{p(r_t, x_{1:t})}{\sum_{r_t} p(r_t, x_{1:t})} \\ &= \sum_{r_{t-1}} p(r_t, r_{t-1}, x_{1:t-1}) \\ &= \sum_{r_{t-1}} p(r_t, r_{t-1} | x_{1:t-1}) p(r_{t-1}, x_{1:t-1}) \\ &= \sum_{r_{t-1}} p(r_t | r_{t-1}, x_{1:t-1}) \frac{p(r_{t-1}, x_{1:t-1})}{p(r_{t-1}, x_{1:t-1})} \end{aligned}$$

Here, $p\{x\}_{1:t}$ given r_t) is the likelihood of observing the data up to time t given a particular run length r_t , and pr_t is the prior probability of the run length. The denominator is the marginal likelihood of the data, which ensures that the posterior distribution is properly normalized.

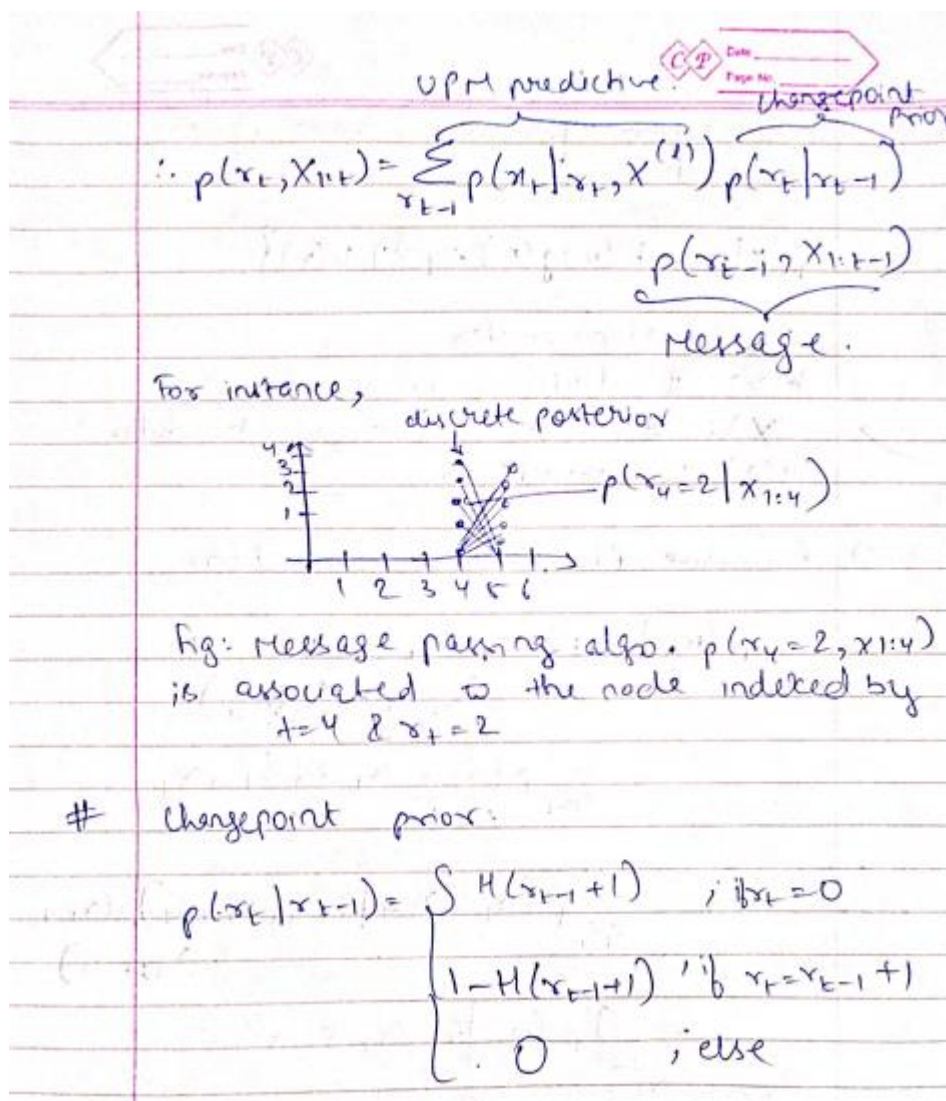
To compute the likelihood $p(\{x_{1:t} | r_t)$, we can use the chain rule of probability and marginalize over all possible run lengths at the previous time step, r_{t-1} :

$$p(\mathbf{x}_{1:t} | r_t) = \sum_{r_{t-1}} p(\mathbf{x}_{1:t}, r_{t-1} | r_t).$$

The last equation in the above picture shows that the BOCPD algorithm makes two assumptions. The first assumption is that the probability of the current distribution being in a particular state only depends on the previous state, not on the previous observations. The second assumption is

that the probability of observing a data point given the current state and some fixed parameters only depends on the current state, not on the previous observations.

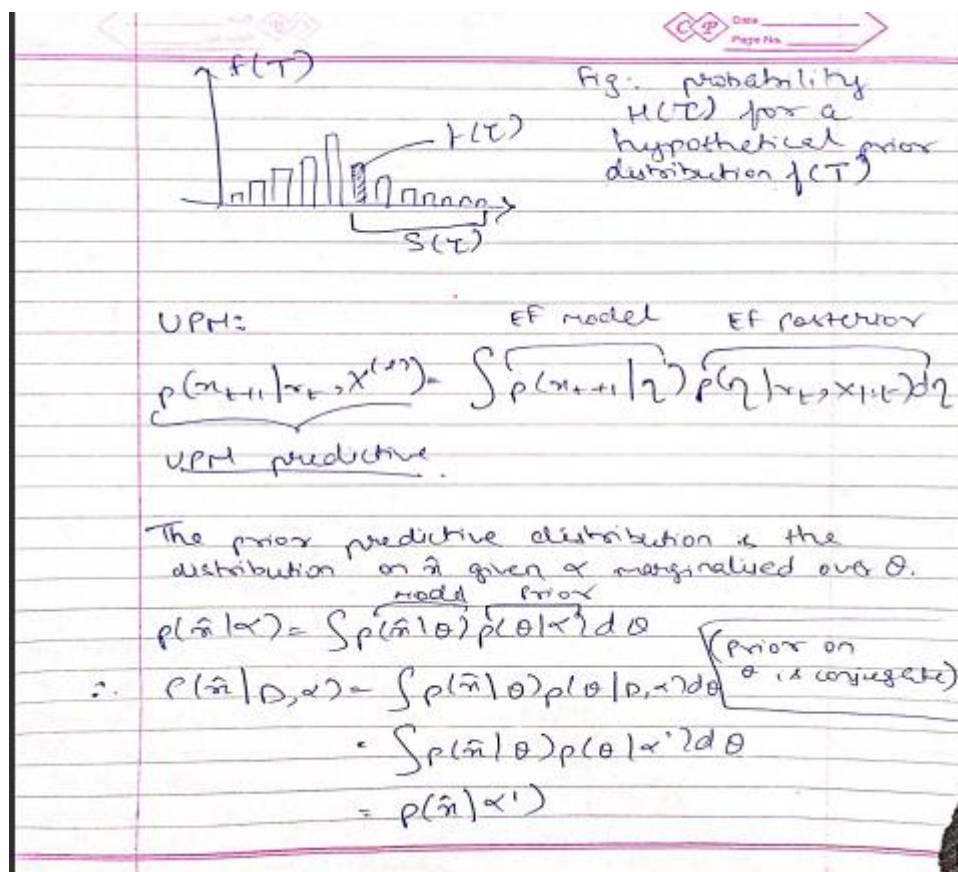
These assumptions allow the joint probability to be written recursively in terms of the conditional probabilities, which can then be used to compute the posterior probability of the state given the observations. The BOCPD algorithm monitors the posterior probability over time and detects changes in the distribution when the probability of being in a particular state exceeds a certain threshold.



This joint probability can be computed recursively using the UPM predictive, changepoint prior, and message from the previous time step.

The UPM predictive is the probability of the observed data given the current hidden state r_t and past observed data $x_{1:t}$. The changepoint prior is the probability of the current hidden state r_t given the previous hidden state r_{t-1} . The message is the joint probability of the previous hidden state and observed data $p(r_{t-1}, x_{1:t-1})$.

By computing the joint probability recursively, we can use a message-passing algorithm to update the posterior probability of the hidden states given the observed data at each time step. This allows us to monitor the probability of being in a particular state over time and detect changes in the distribution when this probability exceeds a certain threshold.



$$p(\hat{x} | \alpha) = \int p(\hat{x} | \theta) p(\theta | \alpha) d\theta.$$

This formula gives us a way to calculate the distribution of new observations before seeing any data.

The prior distribution of the parameters, denoted by $p(\theta | \alpha)$, is a conjugate prior. This means that it is chosen such that the posterior distribution of the parameters, given the data $\{D\}$ and

hyperparameters α , is in the same family as the prior distribution. In other words, the prior and posterior distributions belong to the same family of probability distributions.


Due to this conjugacy, we know that the posterior distribution of θ , given the data D and hyperparameters α , can be expressed as a new distribution with updated hyperparameters α' , such that

$$p(\theta \mid \mathcal{D}, \alpha) = p(\theta \mid \alpha').$$

This means that we can update our prior beliefs about the parameters by simply updating the hyperparameters to their new values α' after seeing the data. We can then use this updated prior distribution to make predictions about new observations, as shown in the prior predictive formula above.

VI. Code(with the description of each line)

(i) Bayesian Online Changepoint Detection(BOCPD):

```
4s  """ =====  
For Reference Purposes:  
  
Author: Gregory Gundersen  
  
Python implementation of Bayesian online changepoint detection for a normal  
model with unknown mean parameter. For algorithm details, see  
  
    Adams & MacKay 2007  
    "Bayesian Online Changepoint Detection"  
    https://arxiv.org/abs/0710.3742  
  
For Bayesian inference details about the Gaussian, see:  
  
    Murphy 2007  
    "Conjugate Bayesian analysis of the Gaussian distribution"  
    https://www.cs.ubc.ca/~murphyk/Papers/bayesGauss.pdf  
  
This code is associated with the following blog posts:  
  
    http://gregorygundersen.com/blog/2019/08/13/bocd/  
    http://gregorygundersen.com/blog/2020/10/20/implementing-bocd/  
===== """  
  
#-----Code Begins----->  
  
import matplotlib.pyplot as plt  
from matplotlib.colors import LogNorm  
import numpy as np  
from scipy.stats import norm  
from scipy.special import logsumexp  
import random  
  
...  
The Python code contains a function named bocd that implements the  
Bayesian Online Changepoint Detection (BOCD) algorithm for detecting  
changes in a time series dataset. The function takes in the following  
parameters: data which represents the time series data, model which  
is an object that represents the underlying statistical model used  
to generate the data, hazard which is the hazard rate for the changepoints,  
and num_samples which is the number of samples to use in the randomized algorithm.  
  
...
```

```

def bocc(data, model, hazard, num_samples=4500):
    # Step 1. Initializes a lower triangular matrix log_R representing the
    # posterior probability distribution as a function of time, and
    # initializes the model parameters.
    T = len(data)
    log_R = -np.inf * np.ones((T+1, T+1))
    log_R[0, 0] = 0 # log 0 == 1
    pmean = np.empty(T) # Model's predictive mean.
    pvar = np.empty(T) # Model's predictive variance.
    log_message = np.array([0]) # log 0 == 1
    log_H = np.log(hazard)
    log_1mH = np.log(1 - hazard)

    for t in range(1, T+1):
        # Step 2. Observes the new datum and makes model predictions for the predictive mean and variance
        x = data[t-1]
        pmean[t-1] = np.sum(np.exp(log_R[t-1, :t]) * model.mean_params[:t])
        pvar[t-1] = np.sum(np.exp(log_R[t-1, :t]) * model.var_params[:t])

        # Step 3. Evaluates the predictive probabilities using the statistical model.
        log_pis = model.log_pred_prob(t, x)

        # Step 4. Calculates the growth probabilities using the predictive probabilities, a message, and the hazard rate.
        log_growth_probs = log_pis + log_message + log_1mH

        # Step 5. Calculates the changepoint probabilities using the predictive probabilities, a message, and the complement of the hazard rate.
        log_cp_prob = logsumexp(log_pis + log_message + log_H)

        # Step 6. Calculates the evidence by appending the changepoint probability to the growth probabilities.
        new_log_joint = np.append(log_cp_prob, log_growth_probs)

        # Step 7. Determines the run length distribution by updating the log posterior matrix log_R with the evidence and normalizes it.
        log_R[t, :t+1] = new_log_joint
        log_R[t, :t+1] -= logsumexp(new_log_joint)

        # Step 8. Updates the sufficient statistics of the model using the observed data.
        model.update_params(t, x)

        # Step 9. Passes the message for use in the next iteration of the algorithm.
        log_message = new_log_joint

    R = np.exp(log_R)

```

```

# ----->
...
This is a Python class called GaussianUnknownMean which defines a model for a
Bayesian online changepoint detection algorithm. The model assumes that the
mean of a normal distribution is unknown but the variance is known. The model
has three parameters: mean0, var0, and varx. The mean0 and var0 are the mean and
variance of the prior distribution of the mean, respectively. Varx is the
known variance of the normal distribution.
...
class GaussianUnknownMean:

    def __init__(self, mean0, var0, varx):

        ...
        p(meanx) = N(mean0, var0)
        p(x) = N(meanx, varx)

        Initializes the class variables, mean_params and prec_params, which are
        arrays representing the mean and precision parameters, respectively. The
        first element of each array is set to mean0 and 1/var0, respectively.

        ...
        self.mean0 = mean0
        self.var0 = var0
        self.varx = varx
        self.mean_params = np.array([mean0])
        self.prec_params = np.array([1/var0])

    def log_pred_prob(self, t, x):
        """

        Computes the posterior predictive for each run length hypothesis. The
        method takes the current time step t and the current data point x as
        input and uses the current mean and variance to compute the predictive
        probabilities.

        """

```

```

    """
    # Posterior predictive: From eq. 40 in (Murphy 2007).
    post_means = self.mean_params[:t]
    post_stds = np.sqrt(self.var_params[:t])
    return norm(post_means, post_stds).logpdf(x)

def update_params(self, t, x):
    """
    Updates the mean and precision parameters upon observing a new data
    point x at time t. The method uses the current precision parameters
    and the observed data to compute the new precision and mean parameters.

    """
    # From eq. 19 in (Murphy 2007).
    new_prec_params = self.prec_params + (1/self.varx)
    self.prec_params = np.append([1/self.var0], new_prec_params)
    # From eq. 24 in (Murphy 2007).
    new_mean_params = (self.mean_params * self.prec_params[:-1] + \
                        (x / self.varx)) / new_prec_params
    self.mean_params = np.append([self.mean0], new_mean_params)

@property
def var_params(self):
    """
    A helper function for computing the posterior variance. The method
    returns an array of posterior variances for each run length hypothesis
    based on the current precision parameters and varx.

    """
    return 1./self.prec_params + self.varx

# ----->

```

```

# ----->
def generate_data(varx, mean0, var0, T, cp_prob):
    """
    The function generates data consisting of T observations, with a probability
    of a changepoint at each observation given by cp_prob. The mean of the data
    is determined by hyperpriors mean0 and var0. The generated data is returned
    as a list along with the list of changepoint indices.

    """
    data = []
    cps = []
    meanx = mean0
    for t in range(0, T):
        if np.random.random() < cp_prob:
            meanx = np.random.normal(mean0, var0)
            cps.append(t)
        data.append(np.random.normal(meanx, varx))
    return data, cps

# ----->
def plot_posterior(T, data, cps, R, pmean, pvar):
    """
    The function creates a plot with two subplots. The first subplot displays the
    generated data, with a line plot of the mean of the posterior predictive
    distribution over the data. The second subplot displays a heat map of the
    posterior probabilities for each run length and time step. The plot_posterior()
    function takes as input the number of observations T, the generated data and
    changepoint indices, the result of the Bayesian online changepoint detection
    algorithm (BOCD) R, the posterior mean pmean, and the posterior variance pvar.

    """
    fig, axes = plt.subplots(2, 1, figsize=(20,10))

    ax1, ax2 = axes

```

```

# ----->
def plot_posterior(T, data, cps, R, pmean, pvar):
    """
    The function creates a plot with two subplots. The first subplot displays the
    generated data, with a line plot of the mean of the posterior predictive
    distribution over the data. The second subplot displays a heat map of the
    posterior probabilities for each run length and time step. The plot_posterior()
    function takes as input the number of observations T, the generated data and
    changepoint indices, the result of the Bayesian online changepoint detection
    algorithm (BOCD) R, the posterior mean pmean, and the posterior variance pvar.

    """
    fig, axes = plt.subplots(2, 1, figsize=(20,10))

    ax1, ax2 = axes

    ax1.scatter(range(0, T), data)
    ax1.plot(range(0, T), data)
    ax1.set_xlim([0, T])
    ax1.margins(0)

    # Plot predictions.
    ax1.plot(range(0, T), pmean, c='k')
    _2std = 2 * np.sqrt(pvar)
    ax1.plot(range(0, T), pmean - _2std, c='k', ls='--')
    ax1.plot(range(0, T), pmean + _2std, c='k', ls='--')

    ax2.imshow(np.rot90(R), aspect='auto', cmap='gray_r',
               norm=LogNorm(vmin=0.0001, vmax=1))
    ax2.set_xlim([0, T])
    ax2.margins(0)

    for cp in cps:
        ax1.axvline(cp, c='red', ls='dotted')
        ax2.axvline(cp, c='red', ls='dotted')

    plt.tight_layout()
    plt.show()
# ----->

if __name__ == '__main__':
    """
    It checks whether the current module is the main program or a module being
    imported. If it is the main program, it sets several parameters and calls the
    generate_data() function to generate data. It then creates an instance of a
    GaussianUnknownMean object, passes it the generated data, and runs the BOCD
    algorithm to compute the posterior probabilities for each run length and time
    step. Finally, it calls the plot_posterior() function to display the results.

    """
    T = 1000 # Number of observations.
    hazard = 1/100 # Constant prior on changepoint probability.
    mean0 = 0 # The prior mean on the mean parameter.
    var0 = 2 # The prior variance for mean parameter.
    varx = 1 # The known variance of the data.

    data, cps = generate_data(varx, mean0, var0, T, hazard)
    model = GaussianUnknownMean(mean0, var0, varx)
    R, pmean, pvar = bocd(data, model, hazard)

    plot_posterior(T, data, cps, R, pmean, pvar)

```


(ii) Bayesian Online Changepoint Detection with randomized algorithm:

✓
4s



```
"""=====
For Reference Purposes:

Author: Gregory Gundersen

Python implementation of Bayesian online changepoint detection for a normal
model with unknown mean parameter. For algorithm details, see

    Adams & MacKay 2007
    "Bayesian Online Changepoint Detection"
    https://arxiv.org/abs/0710.3742

For Bayesian inference details about the Gaussian, see:

    Murphy 2007
    "Conjugate Bayesian analysis of the Gaussian distribution"
    https://www.cs.ubc.ca/~murphyk/Papers/bayesGauss.pdf

This code is associated with the following blog posts:

    http://gregorygundersen.com/blog/2019/08/13/bocd/
    http://gregorygundersen.com/blog/2020/10/20/implementing-bocd/
===== """
#-----Code Begins----->

import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm
import numpy as np
from scipy.stats import norm
from scipy.special import logsumexp
import random

...

The Python code contains a function named bocd that implements the
Bayesian Online Changepoint Detection (BOCD) algorithm for detecting
changes in a time series dataset. The function takes in the following
parameters: data which represents the time series data, model which
is an object that represents the underlying statistical model used
to generate the data, hazard which is the hazard rate for the changepoints,
and num_samples which is the number of samples to use in the randomized algorithm.

...
```

```

def bocd(data, model, hazard, num_samples=1000):
    # Step 1. Initializes a lower triangular matrix log_R representing the
    # posterior probability distribution as a function of time, and
    # initializes the model parameters.
    T = len(data)
    log_R = -np.inf * np.ones((T+1, T+1))
    log_R[0, 0] = 0 # log 0 == 1
    pmean = np.empty(T) # Model's predictive mean.
    pvar = np.empty(T) # Model's predictive variance.
    log_message = np.array([0]) # log 0 == 1
    log_H = np.log(hazard)
    log_1mH = np.log(1 - hazard)

    for t in range(1, T+1):
        # Step 2. Observes the new datum and makes model predictions for the predictive mean and variance
        x = data[t-1]
        pmean[t-1] = np.sum(np.exp(log_R[t-1, :t]) * model.mean_params[:t])
        pvar[t-1] = np.sum(np.exp(log_R[t-1, :t]) * model.var_params[:t])

        # Step 3. Evaluates the predictive probabilities using the statistical model.
        log_pis = model.log_pred_prob(t, x)

        # Step 4. Calculates the growth probabilities using the predictive probabilities, a message, and the hazard rate.
        log_growth_probs = log_pis + log_message + log_1mH

        # Step 5. Calculates the changepoint probabilities using the predictive probabilities, a message, and the complement of the hazard rate.
        log_cp_prob = logsumexp(log_pis + log_message + log_H)

        # Step 6. Calculates the evidence by appending the changepoint probability to the growth probabilities.
        new_log_joint = np.append(log_cp_prob, log_growth_probs)

        # Step 7. Determines the run length distribution by updating the log posterior matrix log_R with the evidence and normalizes it.
        log_R[t, :t+1] = new_log_joint
        log_R[t, :t+1] -= logsumexp(new_log_joint)

        # Step 8. Updates the sufficient statistics of the model using the observed data.
        model.update_params(t, x)

        # Step 9. Passes the message for use in the next iteration of the algorithm.
        log_message = new_log_joint

```

```

# Step 10. Uses a randomized algorithm to generate samples and count the number of times a changepoint occurs in the past, and updates the log posterior matrix log_R.
if t > 1:
    samples = []
    for i in range(num_samples):
        sample = random.choice(range(1, t))
        samples.append(sample)
    counts = np.bincount(samples, minlength=t)
    log_R[t, 1:t+1] = log_R[t, 1:t+1] + np.log(counts)
    log_R[t, 1:t+1] -= logsumexp(log_R[t, 1:t+1])

R = np.exp(log_R)
return R, pmean, pvar

```

----->

...

This is a Python class called GaussianUnknownMean which defines a model for a Bayesian online changepoint detection algorithm. The model assumes that the mean of a normal distribution is unknown but the variance is known. The model has three parameters: mean0, var0, and varx. The mean0 and var0 are the mean and variance of the prior distribution of the mean, respectively. Varx is the known variance of the normal distribution.

...

```

class GaussianUnknownMean:

    def __init__(self, mean0, var0, varx):
        ...

        p(meanx) = N(mean0, var0)
        p(x) = N(meanx, varx)

        Initializes the class variables, mean_params and prec_params, which are
        arrays representing the mean and precision parameters, respectively. The
        first element of each array is set to mean0 and 1/var0, respectively.

        ...

```

```

class GaussianUnknownMean:

    def __init__(self, mean0, var0, varx):
        """
        p(meanx) = N(mean0, var0)
        p(x) = N(meanx, varx)

        Initializes the class variables, mean_params and prec_params, which are
        arrays representing the mean and precision parameters, respectively. The
        first element of each array is set to mean0 and 1/var0, respectively.

        """
        self.mean0 = mean0
        self.var0 = var0
        self.varx = varx
        self.mean_params = np.array([mean0])
        self.prec_params = np.array([1/var0])

    def log_pred_prob(self, t, x):
        """
        Computes the posterior predictive for each run length hypothesis. The
        method takes the current time step t and the current data point x as
        input and uses the current mean and variance to compute the predictive
        probabilities.

        """
        # Posterior predictive: From eq. 40 in (Murphy 2007).
        post_means = self.mean_params[:t]
        post_stds = np.sqrt(self.var_params[:t])
        return norm(post_means, post_stds).logpdf(x)

    def update_params(self, t, x):
        """
        Updates the mean and precision parameters upon observing a new data
        point x at time t. The method uses the current precision parameters
        and the observed data to compute the new precision and mean parameters.

        """

```

```

# From eq. 19 in (Murphy 2007).
new_prec_params = self.prec_params + (1/self.varx)
self.prec_params = np.append([1/self.var0], new_prec_params)
# From eq. 24 in (Murphy 2007).
new_mean_params = (self.mean_params * self.prec_params[:-1] + \
                    (x / self.varx)) / new_prec_params
self.mean_params = np.append([self.mean0], new_mean_params)

@property
def var_params(self):
    """
    A helper function for computing the posterior variance. The method
    returns an array of posterior variances for each run length hypothesis
    based on the current precision parameters and varx.

    """
    return 1./self.prec_params + self.varx

# ----->

def generate_data(varx, mean0, var0, T, cp_prob):
    """
    The function generates data consisting of T observations, with a probability
    of a changepoint at each observation given by cp_prob. The mean of the data
    is determined by hyperpriors mean0 and var0. The generated data is returned
    as a list along with the list of changepoint indices.

    """
    data = []
    cps = []
    meanx = mean0
    for t in range(0, T):
        if np.random.random() < cp_prob:
            meanx = np.random.normal(mean0, var0)
            cps.append(t)
        data.append(np.random.normal(meanx, varx))
    return data, cps

```

```

# ----->
def plot_posterior(T, data, cps, R, pmean, pvar):
    """
    The function creates a plot with two subplots. The first subplot displays the
    generated data, with a line plot of the mean of the posterior predictive
    distribution over the data. The second subplot displays a heat map of the
    posterior probabilities for each run length and time step. The plot_posterior()
    function takes as input the number of observations T, the generated data and
    changepoint indices, the result of the Bayesian online changepoint detection
    algorithm (BOCD) R, the posterior mean pmean, and the posterior variance pvar.
    """
    fig, axes = plt.subplots(2, 1, figsize=(20,10))

    ax1, ax2 = axes

    ax1.scatter(range(0, T), data)
    ax1.plot(range(0, T), data)
    ax1.set_xlim([0, T])
    ax1.margins(0)

    # Plot predictions.
    ax1.plot(range(0, T), pmean, c='k')
    _2std = 2 * np.sqrt(pvar)
    ax1.plot(range(0, T), pmean - _2std, c='k', ls='--')
    ax1.plot(range(0, T), pmean + _2std, c='k', ls='--')

    ax2.imshow(np.rot90(R), aspect='auto', cmap='gray_r',
               norm=LogNorm(vmin=0.0001, vmax=1))
    ax2.set_xlim([0, T])
    ax2.margins(0)

    for cp in cps:
        ax1.axvline(cp, c='red', ls='dotted')
        ax2.axvline(cp, c='red', ls='dotted')

    plt.tight_layout()
    plt.show()

# ----->

```

```

# ----->
if __name__ == '__main__':
    ...
    It checks whether the current module is the main program or a module being
    imported. If it is the main program, it sets several parameters and calls the
    generate_data() function to generate data. It then creates an instance of a
    GaussianUnknownMean object, passes it the generated data, and runs the BOCB
    algorithm to compute the posterior probabilities for each run length and time
    step. Finally, it calls the plot_posterior() function to display the results.
    ...

    T      = 1000    # Number of observations.
    hazard = 1/100    # Constant prior on changepoint probability.
    mean0  = 0       # The prior mean on the mean parameter.
    var0   = 2       # The prior variance for mean parameter.
    varx   = 1       # The known variance of the data.

    data, cps      = generate_data(varx, mean0, var0, T, hazard)
    model          = GaussianUnknownMean(mean0, var0, varx)
    R, pmean, pvar = bocb(data, model, hazard)

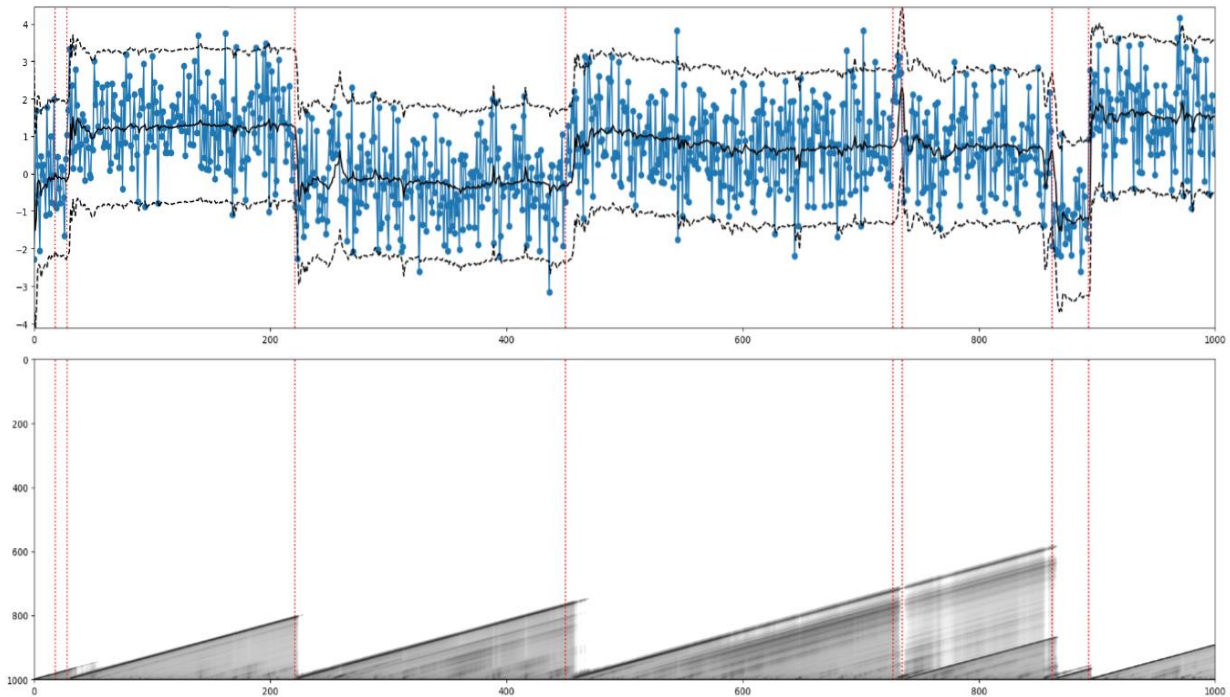
    plot_posterior(T, data, cps, R, pmean, pvar)

```

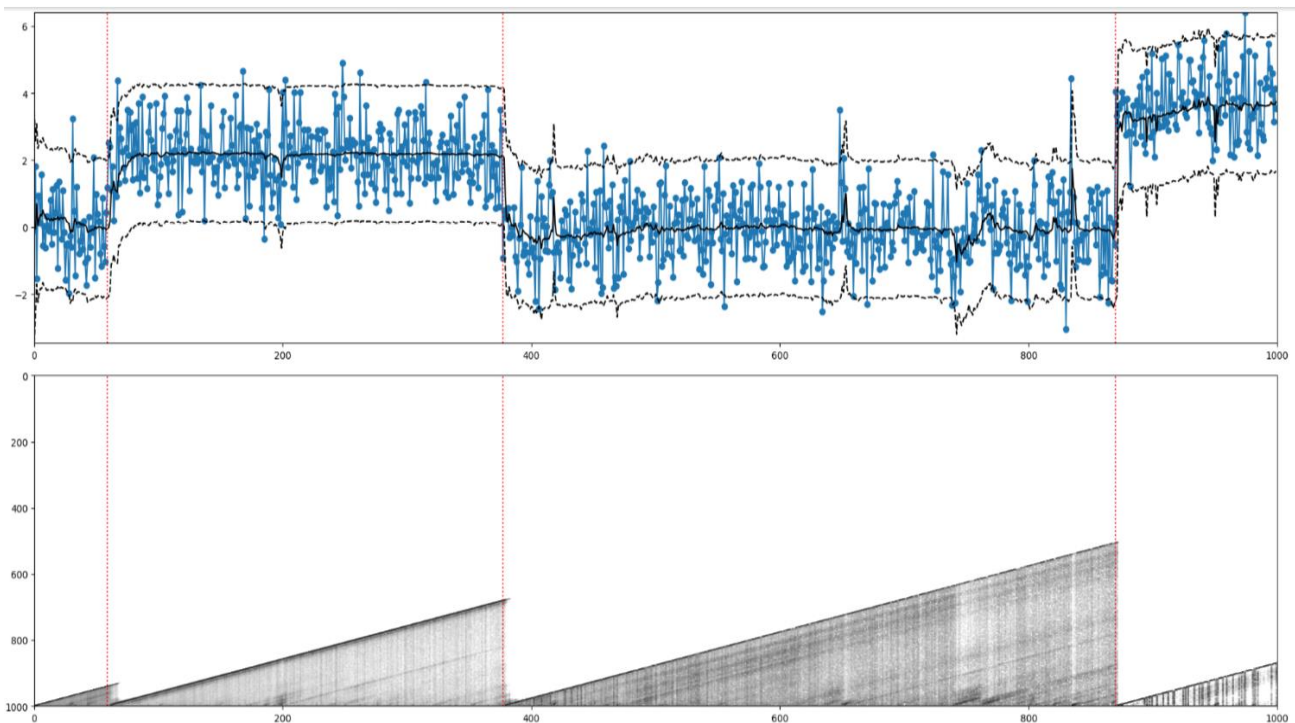
VII. Results and Inferences:

(i) Changepoint detection using Bayesian

Online Changepoint detection:



(ii) Changepoint detection using Bayesian Online Changepoint detection implemented with randomized algorithm for dividing the run length into smaller sample sets:



VIII. References:

1. Main Reference paper: Monteiro, M., & Costa, M. (2023, January 4). *Change point detection by state space modeling of long-term air temperature series in Europe*. MDPI. Retrieved April 9, 2023, from <https://www.mdpi.com/2571-905X/6/1/7>
2. Adams, R. P., & MacKay, D. J. C. (2007, October 19). *Bayesian Online Changepoint Detection*. arXiv.org. Retrieved April 9, 2023, from <https://arxiv.org/abs/0710.3742>
3. Gallagher, C., Lund, R., & Robbins, M. (2013, July 15). *Changepoint detection in climate time series with long-term trends*. AMETSOC. Retrieved April 9, 2023, from <https://journals.ametsoc.org/view/journals/clim/26/14/jcli-d-12-00704.1.xml>
4. *Quantifying the impact of land use and land cover change on moisture ...* (n.d.). Retrieved April 9, 2023, from <https://agupubs.onlinelibrary.wiley.com/doi/full/10.1029/2022JD038421>
- 5.
- 6.
7. <https://www.cs.ubc.ca/~murphyk/Papers/bayesGauss.pdf>
8. Gundersen, G. (2019, August 13). *Bayesian Online Changepoint Detection*. <http://gregorygundersen.com/blog/2019/08/13/bocd/>
9. Gundersen, G. (2020, October 20). *Implementing Bayesian Online Changepoint Detection*. <http://gregorygundersen.com/blog/2020/10/20/implementing-bocd/>
10. G. (n.d.). *GitHub - gwgundersen/bocd: Python implementation of Bayesian online changepoint detection*. GitHub. <https://github.com/gwgundersen/bocd>

