Package fpop2D

Description

fpop2D is an R package written in Rcpp/C++ and developed to change-point detection in bivariate time-series. The package implements the Functional Pruning Optimal Partitioning algorithm with two types of pruning: "intersection of sets" and "difference of intersection and union of sets".

Package structure

- Part R
  - fpop2D.R

    The file contains the implementation of the following functions:

    data_gen2D - generation of data of dimension 2 with a given values of means and changepoints

    plotFPOP2D - plot of data with a values of means and changepoints.
  - RcppExports.R

    The file contains the implementation of the function **FPOP2D** that calls the function **FPOP2D** implemented in C++.
- Part C++
  - Cost.h, Cost.cpp

    The files contain the implementation code for the class **Cost** .
  - Disk.h, Disk.cpp

    The files contain the implementation code for the class **Disk**.
  - Rect.h, Rect.cpp

    The files contain the implementation code for the class **Rect**.
  - Geom.h, Geom.cpp

    The files contain the implementation code for the class **Geom**.
  - OP.h, OP.cpp

    The files contain the implementation code for the class **OP**.
  - fpop2D.cpp

    The file contains the code of the function **FPOP2D** that implements the change-point detection in bivariate time-series using the Functional Pruning Optimal Partitioning algorithm.
  - RcppExports.cpp

    The file contains the code that exports data R/C ++ .

We consider $(y_1, y_2) \in (R^2)^n$ - bivariate time-series when $y_1 = (y_1^1, .., y_n^1)$ and $y_2 = (y_1^2, .., y_n^2)$ - two vectors of univariate data size n.
We use the Gaussian cost of the segmented bivariate data when m_t is the value of the optimal cost, $m_0 = -\beta$. The cost function has the form:

$$q_t^i(\theta_1, \theta_2) = m_{i-1} + \beta + (t-i+1)((\theta_1 - E[y_{i:t}^1])^2 + (\theta_2 - E[y_{i:t}^2])^2) + (t-i+1)(V(y_{i:t}^1) + V(y_{i:t}^2)), \quad i = 1:t. \tag{1}$$

$$E[y_{i:t}^k] = \frac{1}{t-i+1} \sum_{l=i}^{t} (y_{i:t}^k), \quad k = 1, 2. \tag{2}$$

$$V(y_{i:t}^k) = \frac{1}{t-i+1} \sum_{l=i}^{t} (y_{i:t}^k)^2 - \left(\frac{1}{t-i+1} \sum_{l=i}^{t} (y_{i:t}^k) \right)^2, \quad k = 1, 2. \tag{3}$$

In the case of a Gaussian cost $q_t^i(\theta_1, \theta_2)$ is a paraboloid so that:

$$m_t = min_{i=1:t} \left\{ m_{i-1} + (t-i+1)(V(y_{i:t}^1) + V(y_{i:t}^2)) + \beta \right\}. \tag{4}$$

We introduce the notations:

$$\begin{aligned} mu1 &= E[y_{i:t}^1], \\ mu2 &= E[y_{i:t}^2], \\ coef &= (t-i+1), \\ mi\_1\_p &= m_{i-1} + \beta, \\ coef\_Var &= (t-i+1)(V(y_{i:t}^1) + V(y_{i:t}^2)). \end{aligned} \tag{5}$$

Then the cost function takes the form:

$$q_t^i(\theta_1, \theta_2) = mi\_1\_p + coef \cdot ((\theta_1 - mu1)^2 + (\theta_2 - mu1)^2) + coef\_Var, \quad i = 1:t. \tag{6}$$

Class Cost

We define the class characteristics (5):

- **coef**, **coef_Var**, **mi_1_p**, **mu1**, **mu2**

The class implements constructor:

- **Cost(unsigned int i, unsigned int t, double** vectS, double mi_1pen)** is the cost function at the time $[i, t]$.

We define the class methods:

- **get_coef()**, **get_coef_Var()**, **get_mi_1_p()**, **get_mu1()**, **get_mu2()**

  We use these methods to access the characteristics of the class.

- **get_min()**

  We use this method to get the minimum value of the cost function (4).

Class Disk

A class element is a circle that is defined by the center coordinates and the radius.

- We define the class characteristics:
  - **center1, center2** is the center of the circle.
  - **radius** is the radius of the circle.
- The class implements two constructors:
  - **Disk()** All characteristics are equal zero by default.
  - **Disk(double c1, double c2, double r)**
    $(c1, c2)$ are the center coordinates and r is the radius of the circle.
- We define the class methods:
  - **get_center1()**, **get_center2()**, **get_radius()**
    We use these methods to access the class characteristics.

Class Rect

A class element is a rectangle that is defined by the coordinates of the lower left-hand corner and the upper-right hand corner.

- We define the class characteristics:
  - **rectx0,recty0** are coordinates of the lower left-hand corner.
  - **rectx1, recty1** are coordinates of the upper-right hand corner.
- The class implements two constructors:
  - **Rect()** All characteristics are equal zero by default.
  - **Rect(double x0, double y0, double x1, double y1)**
    $(x0, y0)$ is the the lower left-hand corner and $(x1, y1)$ is the upper-right hand corner.
- We define the class methods:
  - **get_center1()**, **get_center2()**, **get_radius()**
    We use these methods to access the class characteristics.

Class Geom

- We define the class characteristics:
  - **label_t** is the time moment.
  - **rect_t** is the rectangle, the element of class **Rect**.
- The class implements two constructors:
  - **Geom(double c1, double c2, double r, double t)**
    * $label\_t = t$;
    * $rect\_t = Rect(c1 - r, c2 - r, c1 + r, c2 + r)$.
  - **Geom(double t, Cost cst, Rect rct)**
    * $label\_t = t$;
    * $rect\_t = rct$;
    * $cost\_t = cst$.
- We define the class methods:
  - **get_label_t()**, **get_rect_t()**
    We use these methods to access the class characteristics.
  - **min_ab(double a, double b)**, **max_ab(double a, double b)**
    We use these methods to find the minimum and maximum of two numbers.
  - **bool empty_set(Rect R)**
    The function checks the parameters of the rectangle **R**. If the parameters are not correct, this rectangle is empty.
  - **Rect intersection(Rect rect, Disk disk)**
    The function approximates a rectangle and a circle intersection area by horizontal and vertical lines. Basing on the intersection points of these lines, we construct a rectangle with a minimum area, which contains the intersection area of the rectangle and the circle.
  - **Rect difference(Rect rect, Disk disk)**
    The function approximates a rectangle and a circle difference area by horizontal and vertical lines. Basing on the intersection points of these lines, we construct a rectangle with a minimum area, which contains the difference area of the rectangle and the circle.

Class OP

- We define the class characteristics:
  - **n** is the number of data points.
  - **penalty** is a value of penalty (a non-negative real number).
  - **sy12** are sum vectors $\sum_{k=1}^{t} y_k^1$, $\sum_{k=1}^{t} y_k^2$, $\sum_{k=1}^{t} (y_k^1)^2$, $\sum_{k=1}^{t} (y_k^2)^2$, $t = 1 : n$.
  - **last_chpts** is the vector of candidates for the position of changepoints.
  - **m** is the vector of the optimal cost value.
  - **changepoints** is the vector of changepoints.
  - **means1** is the vector of successive means for data1.
  - **means2** is the vector of successive means for data2.
  - **globalCost** is the global cost.
- The class implements the constructor:
  - **OP(std::vector<double> y1, std::vector<double> y2, double beta)**
    - $penalty = beta$;
    - $n = y1.size()$;
    - we allocate memory for sy12;
- We define the class methods:
  - **get_changepoints()**, **get_means1()**, **get_means2()**, **get_globalCost()**, **get_n()**, **get_sy12()**
    We use these methods to access the class characteristics.
  - **vect_sy12(std::vector<double> y1, std::vector<double> y2)**
    We use this method to find the sum vectors $\sum_{k=1}^{t} y_k^1$, $\sum_{k=1}^{t} y_k^2$, $\sum_{k=1}^{t} (y_k^1)^2$, $\sum_{k=1}^{t} (y_k^2)^2$, $t = 1 : n$.
  - **backtracking(unsigned int ndata)**
  - **algoFPOP(std::vector<double> y1, std::vector<double> y2, int type)**
    The function implements the Functional Pruning Optimal Partitioning algorithm with two types of pruning: "intersection of sets"(*type* = 1) and "difference of intersection and union of sets"(*type* = 2).

Rect Geom::intersection(Rect rect, Disk disk)

Description

The function approximates a rectangle and a circle intersection area by horizontal and vertical lines. Basing on the intersection points of these lines, we construct a rectangle with a minimum area, which contains the intersection area of the rectangle and the circle.
If there is no intersection, the function returns the rectangle with parameters that correspond to the condition:

$$(rectx0 \geq rectx1)||(recty0 \geq recty1). \tag{7}$$

Input parameters:

The input of this function consists of two parameters:

- **rect** is the rectangle, the element of class **Rect** with characteristics:
  - **rectx0,recty0** are coordinates of the bottom left corner;
  - **rectx1, recty1** are coordinates of the top right corner.

  To access these characteristics we use the methods **get_rectx0()**, **get_recty0()**, **get_rectx1()**, **get_recty1()** implemented in the class **Rect**.

- **disk** is the circle, the element of class **Disk** with characteristics:
  - **center1, center2** is the center of the circle;
  - **radius** is the radius of the circle.

  To access these characteristics we use the methods **get_center1()**, **get_center2()**, **get_radius()** implemented in the class **Disk**.

Output parameters:

- The function returns new rectangle **rect_approx** (the element of class **Rect**) with a minimum area, which contains the intersection area of the rectangle **rect** and the circle **disk**. The rectangle is formed as a result of the intersection of horizontal and vertical lines that approximate the intersection area of the rectangle and the circle.

  If there is no intersection, the function returns a rectangle **rect_approx** with parameters that correspond to the condition (7).

Algorithm:

Preprocessing

Using the methods **get_rectx0()**, **get_recty0()**, **get_rectx1()**, **get_recty1()** implemented in the class **Rect** we define the parameters of the rectangle **rect**:

$$\begin{aligned} x0 &= rect.get\_rectx0(), \\ x1 &= rect.get\_rectx1(), \\ y0 &= rect.get\_recty0(), \\ y1 &= rect.get\_recty1(). \end{aligned} \tag{8}$$

Using the methods **get_center1()**, **get_center2()**, **get_radius()** implemented in the class **Disk** we define the parameters of the circle **disk**:

$$\begin{aligned} c1 &= disk.get\_center1(), \\ c2 &= disk.get\_center2(), \\ r &= disk.get\_radius(). \end{aligned} \tag{9}$$

Approximation

We consider two directions and update the characteristics of rectangle:

- horizontal direction(the characteristics $y0, y1$):
  If $x0 \leq c1 \leq x1$ then
  $$\begin{aligned} y0 &= \max\{y0, c2 - r\}, \\ y1 &= \min\{y1, c2 + r\}. \end{aligned} \tag{10}$$
  Otherwise, we consider the points of intersection of the circle with straight lines $x = x0$ and $x = x1$. The circle has two intersection points with a straight line if the discriminant is positive. We define the values dl, dr (13) as the value of a discriminant devided by 4 of each system (11, 12):
  $$\begin{cases} (x - c1)^2 + (y - c2)^2 = r^2, \\ x = x0. \end{cases} \tag{11}$$
  $$\begin{cases} (x - c1)^2 + (y - c2)^2 = r^2, \\ x = x1. \end{cases} \tag{12}$$
  $$\begin{aligned} dl &= r^2 - (x0 - c1)^2, \\ dr &= r^2 - (x1 - c1)^2, \end{aligned} \tag{13}$$
  Note: we define the default intersection points for the algorithm to work correctly as:
  $$\begin{aligned} l1 &= r1 = \infty, \\ l2 &= r2 = -\infty. \end{aligned} \tag{14}$$
  We check the sign of dl, dr and find the intersection points:
  $$\begin{cases} dl > 0, \\ l1 = c2 - \sqrt{dl}, \\ l2 = c2 + \sqrt{dl}. \end{cases} \tag{15}$$
  $$\begin{cases} dr > 0, \\ r1 = c2 - \sqrt{dr}, \\ r2 = c2 + \sqrt{dr}. \end{cases} \tag{16}$$
  We define the characteristics of rectangle as:
  $$\begin{aligned} y0 &= \max\{y0, \min\{l1, r1\}\}, \\ y1 &= \min\{y1, \max\{l2, r2\}\}. \end{aligned} \tag{17}$$
- vertical direction (the characteristics $x0, x1$)
  If $y0 \leq c2 \leq y1$ then
  $$\begin{aligned} x0 &= \max\{x0, c1 - r\}, \\ x1 &= \min\{x1, c1 + r\}. \end{aligned} \tag{18}$$
  Otherwise, we consider the points of intersection of the circle with straight lines $y = y0$ and $y = y1$. The circle has two intersection points with a straight line if the discriminant is positive. We define the values db, dt (21) as the value of a discriminant devided by 4 of each system (19, 20):

$$\begin{cases} (x - c1)^2 + (y - c2)^2 = r^2, \\ y = y0. \end{cases} \tag{19}$$

$$\begin{cases} (x - c1)^2 + (y - c2)^2 = r^2, \\ y = y1. \end{cases} \tag{20}$$

$$\begin{aligned} db &= r^2 - (y0 - c2)^2, \\ dt &= r^2 - (y1 - c2)^2. \end{aligned} \tag{21}$$

Note: we define the default intersection points for the algorithm to work correctly as:

$$\begin{aligned} b1 = t1 &= \infty, \\ b2 = t2 &= -\infty. \end{aligned} \tag{22}$$

We check the sign of db, dt and find the intersection points:

$$\begin{cases} db > 0, \\ b1 = c1 - \sqrt{db}, \\ b2 = c1 + \sqrt{db}. \end{cases} \tag{23}$$

$$\begin{cases} dt > 0, \\ t1 = c1 - \sqrt{dt}, \\ t2 = c1 + \sqrt{dt}. \end{cases} \tag{24}$$

We define the characteristics of rectangle as:

$$\begin{aligned} x0 &= \max\{x0, \min\{b1, t1\}\}, \\ x1 &= \min\{x1, \max\{b2, t2\}\}. \end{aligned} \tag{25}$$

- If all the values dl, dr, db, dt are non-positive, the rectangle has no intersections with the circle and we define the characteristics of rectangle as:

$$x0 = x1. \tag{26}$$

Output

Once all the parameters are updated we form the required rectangle **rect_approx** as:

$$rect\_approx = Rect(x0, y0, x1, y1); \tag{27}$$

Rect Geom::difference(Rect rect, Disk disk)

Description

The function approximates a rectangle and a circle difference area by horizontal and vertical lines. Basing on the intersection points of these lines, we construct a rectangle with a minimum area, which contains the difference area of the rectangle and the circle.
If the difference is the empty set, the function returns the rectangle with parameters that correspond to the condition (7).

Input parameters:

The input of this function consists of two parameters:

- **rect** is the rectangle, the element of class **Rect** with characteristics:
  - **rectx0,recty0** are coordinates of the bottom left corner;
  - **rectx1, recty1** are coordinates of the top right corner.

  To access these characteristics we use the methods **get_rectx0()**, **get_recty0()**, **get_rectx1()**, **get_recty1()** implemented in the class **Rect**.
- **disk** is the circle, the element of class **Disk** with characteristics:
  - **center1, center2** is the center of the circle;
  - **radius** is the radius of the circle.

  To access these characteristics we use the methods **get_center1()**, **get_center2()**, **get_radius()** implemented in the class **Disk**.

Output parameters:

- The function returns new rectangle **rect_approx** (the element of class **Rect**) with a minimum area, which contains the difference area of the rectangle **rect** and the circle **disk**. The rectangle is formed as a result of the intersection of horizontal and vertical lines that approximate the difference area of the rectangle and the circle.

  If the difference is the empty set, the function returns a rectangle **rect_approx** with parameters that correspond to the condition (7).

Algorithm:

Preprocessing

We define:

- the parameters of the rectangle **rect** (8),
- the parameters of the circle **disk** (9),
- the values dl, dr, db, dt (13,21).

Approximation

We consider two directions and update the characteristics of rectangle:

- horizontal direction(the characteristics $y0, y1$):

  We consider the points of intersection of the circle with straight lines $x = x0$ and $x = x1$. We update the characteristics $y0, y1$ if dl and dr (13) is positive.

  Note: we define the default intersection points for the algorithm to work correctly as (14).

We check the sign of dl, dr and find the intersection points (15) and (16).

We define the characteristics of rectangle as:

$$\begin{aligned} y0 &= \max\{y0, \min\{l2, r2\}\}, \\ y1 &= \min\{y1, \max\{l1, r1\}\}. \end{aligned} \tag{28}$$

- vertical direction (the characteristics $x0, x1$)

  We consider the points of intersection of the circle with straight lines $y = y0$ and $y = y1$. We update the characteristics $x0, x1$ if db and dt (21) is positive.

  Note: we define the default intersection points for the algorithm to work correctly as (22).

  We check the sign of db, dt and find the intersection points (23) and (24).

  We define the characteristics of rectangle as:

$$\begin{aligned} x0 &= \max\{x0, \min\{b2, t2\}\}, \\ x1 &= \min\{x1, \max\{b1, t1\}\}. \end{aligned} \tag{29}$$

Output

Once all the parameters are updated we form the required rectangle **rect_approx** as (27).

bool Geom::empty_set(Rect rect)

Description

The function checks the parameters of the rectangle. If the parameters are not correct, this rectangle is empty.

Input parameters:

- **rect** is the rectangle, the element of class **Rect** with characteristics:
  - **rectx0,recty0** are coordinates of the bottom left corner;
  - **rectx1, recty1** are coordinates of the top right corner.

To access these characteristics we use the methods **get_rectx0()**, **get_recty0()**, **get_rectx1()**, **get_recty1()**, implemented in the class **Rect**.checks the parameters of the rectangle.

Output parameters:

The function returns a boolean value **true** if the rectangle is empty, and **false** if it is not empty.

Algorithm:

If the parameters of the rectangle correspond to the condition (7) this rectangle is empty and the function returns a boolean value **true**, else **false**.

void OP::algoFPOP(std::vector<double> y1, std::vector<double> y2, int type)

Description

The function implements the Functional Pruning Optimal Partitioning algorithm with two types of pruning: "intersection of sets"($type = 1$) and "difference of intersection and union of sets"($type = 2$) for bivariate time series.

Input parameters:

- **y1** is the vector of data1(a univariate time series).
- **y2** is the vector of data2(a univariate time series).
- **type** is the value defined the type of pruning (1 = intersection set, 2 = difference of intersection and union set).

Output parameters:

The function forms the vectorS **changepoints**, **means1, means2** andthe value **globalCost**.

Algorithm:

Preprocessing

We define:

- $n = y1.size()$.
- $sy12 = vect_s y12(y1, y2), s = get_s y12()$ are the sum vectors.
- $geom\_activ$ is an element of class **Geom**.
- $cost\_activ$ is a cost function, an element of class **Cost**.
- $list\_geom$ is a list of class **Geom** elements.
- it_list is an itetator for $list\_geom$.
- $mus1, mus2$ are the vectors of means for $last\_chpts$.
- $m = 0, penalty$ are the variables for finding the minimum value of cost function.

Processing

For each $t = 0, .., n - 1$ we do:

- We create $geom\_activ$ for the point (y_t^1, y_t^2) of the bivariate time series as (30) and we add this element to the begining of the list $list\_geom$.

$$geom\_activ = Geom(y1[t], y2[t], sqrt(m[t + 1] - m[t]), t). \tag{30}$$

  For first list element we define:

  - $cost\_activ = Cost(t, t, s[t], s[t + 1], m[t])$;
  - i is a label of first element in $list\_geom$.
  - $min\_val\_cost = cost\_activ.get\_min()$ for the first element.
  - $mean1, mean2$ are the values of means for interval (i, t).

- <u>The first run: Search $m[t + 1]$</u>

  Starting from the second element of the list $list\_geom$ until we get to the end of the list:

- We define lbl is the $label\_t$ of the current element.
- We create new $cost\_activ$ for the interval (lbl, t).
- We find the minimum value of the cost function min_val_cost using the method **get_min()**.

- We put $\min min_v al_c ost + penalty$ to the vector m.
- We put the lbl that corresponds $m[t+1]$ to the vector $last\_chpts$ and put the values of means for the interval (lbl, t) to the vectors $mus1$ and $mus2$.
- The second run: Pruning
  - Type = 1: Pruning "intersection"
    * We define:
      · $disk\_new$ is an element of **Disk** class.
      · $rect\_new$ is an element of **Rect** class.
      · $geom\_update$ is an element of **Geom** class.
      · r_new is a value of the radius for $disk\_new$.
    * Starting from the second element of the list $list\_geom$ until we get to the end of the list:
      · We get the element of the list $list\_geom$ pointed to by the iterator it_list to the variable $geom\_activ$.
      · We define lbl is the $label\_t$ of the current element.
      · We create new $cost\_activ$ for the interval (lbl, t).
      · We calculate the radius r_new as (31) and we forms the new disk $disk\_new$ (32).

$$r\_new = \sqrt{\frac{m[t+1] - cost\_activ.get\_mi\_1\_p()}{cost\_activ.get\_coef()} - cost\_activ.get\_coef\_Var()}. \tag{31}$$

$$disk\_new = Disk(cost\_activ.get\_mu1(), cost\_activ.get\_mu2(), r\_new). \tag{32}$$

      · We put $rect\_t$ to the variable $rect\_new$ using the method **get_rect_t()** of **Geom** class.
      · We intersect the disk $disk\_new$ with the $rect\_new$ and forms new element $geom\_update$ of **Geom** class as (33).

$$\begin{gathered} Rectrect\_new = geom\_activ.intersection(geom\_activ.get\_rect\_t(), disk\_new), \\ geom\_update = Geom(geom\_activ.get\_label\_t(), rect\_new). \end{gathered} \tag{33}$$

      · Using the method **empty_set(geom_update.get_rect_t())** we check the correctness $rect\_t$ of new element $geom\_update$.
      · If it is not empty, we replace the element of list $list\_geom$ pointed to by the iterator it_list with $geom\_update$, otherwise we delete this element of list $list\_geom$.
  - Type = 2: Pruning "difference of intersection and union set"
    ....

Output:

Knowing the values of the vectors m, $last\_chpts$, $mus1$ and$mus2$, we forme the vectors $changepoints$, $means1$, $means2$ and the value $globalCost$.