CAD Pseudo Code

```
*********************************************************
        CAD:
                    A function that finds the anomalies of a time series of random deviates from
                    various normal distributions. (I need to clarify this further)

        Calls:
                    anomaly_finder

        Called by:
                    None

        Input Parameters:
                    time_series - the time series being searched for the anomalies
                    delta - the value to be added to the value to the CUSUM parameter k; its
                           default value is 3
                    lambda - the minimum length of the anomalous subsequence the code should
                           detect; the default is 5
                    type - the type of the anomaly “upper” or “lower”
                    number_of_windows - the number of windows used
                    step_size - the distance between two consecutive windows
                    training_set_window_length - the starting length of the training set searched
                           for within each window
                    training_set_step_size - the size of the shift by which the
                           training_set_windows are moved down the sequence.
                    starting_k_value - the starting value of the CUSUM parameter k
                    starting_H_value - the starting value of the  CUSUM parameter H

        Returns:
                    anomaly_indices - The indices of the anomalies
        *****************************************************************
```

```
FUNCTION    CAD(time_series, delta, lambda, type,
            number_of_windows, step_size, training_set_window_length,
training_set_step_size)

            n <-LEN(time_series)
            window_length <- n - (number_of_windows*step_size) + 1
            INIT anomaly_indices <- NULL

            FOR i FROM 1 TO (number_of_windows*step_size) BY      step_size

                  temp_var <- time_series[i TO (i+window_length -1)]
                  temp_indices <- anomaly_finder(temp_var, delta, lambda,
                        type, i, step_size,
                        training_set_window_length,
                                    training_set_step_size)
                  anomaly_indices <- CONCATENATE(anomaly_indices,
                                    temp_indices)

            ENDFOR

            RETURN(anomaly_indices)

ENDFUNCTION
```

```
************************************************************
        anomaly_finder:
                     A function that finds the anomalies for a specific window
                     of a time series
        Calls:
                     training_subsequence_finder
                     evaluate_CUSUM_results

        Called by:
                     CAD

        Input Parameters:
                     subsqnce  - a window length subsequence of the original
                            time series.
                     delta - the value to be added to the value to the CUSUM parameter k; its
                            default value is 3
                     lambda - the minimum length of the anomalous subsequence the code should
                            detect; the default is 5
                     type - the type of the anomaly “upper” or “lower”
                     indx - the index of the first element of subsqnce within time_series.
                     step_size - the distance between two consecutive windows
                     training_set_window_length - the starting length of the training set searched
                            for within each window
                     training_set_step_size - the size of the shift by which the
                            training_set_windows are moved down the sequence.
                     starting_k_value - the starting value of the CUSUM parameter k
                     starting_H_value - the starting value of the  CUSUM parameter H

        Returns:
                     global_indices - The indices of the anomalies within the original time_series.
**********************************************************************
```

```
FUNCTION anomaly_finder(subsqnce, delta, lambda, type, indx, step_size,
            training_window_length, training_set_step_size,
            starting_k_value, starting_H_value)

            training_set_obj <- training_subsequence_finder(subsqnce,
            training_window_length, training_set_step_size, starting_k_value,
                  starting_H_value)

            IF  training_set_obj[1] = NULL THEN
                  WARNING(“No training set was found.”)
                  RETURN(NULL)
            ELSE
                  x_bar <- MEAN(training_set_obj[1])
                  sigma <- STANDARD_DEVIATION(training_set_obj[1])
                  k <- training_set_obj[2] + delta
                  H <- training_set_obj[3]
                  local_indices <- evaluate_CUSUM_results(subsqnce, x_bar, sigma, H, k,
                        type, lambda)
                  IF LEN(local_indices)!= 0 THEN
                        global_indices <- local_indices + indx -1
                  ELSE
                        global_indices <- NULL
                  ENDIF
                  RETURN(global_indices)
           ENDIF
ENDFUNCTION
```

```
********************************************************
        training_subsequence_finder:
                    A function that searches for a subsequence of a time series that is both (close
                    to) normal and in statistical control.  This subsequence must have a minimal
                    length of 30 and maximal length of training_set_window_length.
        Calls:
                    normality_finder
                    parameter_finder

        Called by:
                    anomaly_finder

        Input Parameters:
                    x - a time series; it must have length greater than 30.
\
                    training_set_window_length - the starting length of the training set searched
                           for within each window
                    training_set_step_size - the size of the shift by which the
                           training_set_windows are moved down the sequence.
                    starting_k_value - the starting value of the CUSUM parameter k
                    starting_H_value - the starting value of the  CUSUM parameter H

        Returns:
                    training_set_obj - a list/object of length 3, where
                           training_set_obj[1] - the subsequence of x that is to be used
                                  as the training data set
                           training_set_obj[2] - the CUSUM k value of training_set_obj[1]
                           training_set_obj[3] - the CUSUM H value of training_set_obj[1]

****************************************************************

FUNCTION training_subsequence_finder(x, training_set_window_length,
training_set_step_size, starting_k_value, starting_H_value)

      INIT segment_found <- FALSE
      INIT cant_be_done <- FALSE
      INIT  first_time_through <- TRUE
      INIT parameters_plus <- NULL
      n <- LEN(x)

      WHILE (NOT segment_found) AND ( NOT cant_be_done)
            FOR i FROM 1 TO (n - training_set_window_length +1) BY
      training_set_step_size

                  temp <- x[i TO (i+training_set_window_length-1)]
                  temp_obj <- normality_finder(temp)

                  IF temp_obj[1]> 0.05 THEN
                        p_value <- temp_obj[1]
                        skew <- temp_obj[2]
                        kurt <- temp_obj[3]
                        parameters_obj <- parameter_finder(temp, starting_k_value,
                        starting_H_value)

                        IF (parameters_obj[1] ! = NULL) THEN
                              the_J <- ABS(skew) + ABS(3 - kurt)+ ABS(1-p_value)+
                        (100*parameters_obj[1])+(100*parameters_obj[2])
                              dummy_var <- [temp, p_value, skew, parameters_obj[1],
                                    parameters_obj[2],the_J]

                              IF first_time_through THEN
                                    parameters_plus <- dummy_var
                                    first_time_through <- FALSE
                              ELSE
                                    IF parameters_plus[6] > the_J
                                    parameters_plus <- dummy_var
                                    ENDIF
                              ENDIF
                        ENDIF
                  ENDIF
            ENDFOR

            IF parameters_plus != NULL THEN
                  segment_found <- TRUE
            ELSE
                  IF training_set_window_length > 30 THEN
                        training_set_window_length <-
                        training_set_window_length -1
                  ELSE
                        cant_be_done <- TRUE
                  ENDIF
            ENDIF
      ENDWHILE

      IF cant_be_done THEN
            return_obj <- [NULL, NULL, NULL]
      ELSE
            return_obj <- [parameters_plus[1], parameters_plus[4],
      parameters_plus[5]]
      ENDIF

      RETURN(return_obj)

ENDFUNCTION
```

```
*******************************************************
       normality_finder:
                  A function that finds Skewness, Kurtosis and Shapiro-Wilk normality test p-
                  value for a time series (the input.)
       Calls:
                  SKEWNESS
                  KURTOSIS
                  SHAPIRO_WILK_P_VALUE

       Called by:
                  training_subsequence_finder

       Input Parameters:
                  x - a time series; it must have length of at least 30.
\

       Returns:
                  normality_obj - a list/obj of length 3, where
                        normality_obj[1] - skewness of x
                        normality_obj[2] - kurtosis of x
                        normality_obj[3] - Shapiro-Wilk p-value of x

***************************************************************
```

```
FUNCTION normality_finder(x)

      temp1 <- SKEWNESS(x)
      temp2 <- KURTOSIS(x)
      temp3 <- SHAPIRO_WILK_P_VALUE(x)

      return([temp1, temp2, temp3])

ENDFUNCTION
```

```
*******************************************************
       parameter_finder:
                   A function takes a time series x of length at least 30 that has a (nearly)
                   normal distribution and selects the smallest CUSUM parameters k and H for which
                   the time series is in statistical control
       Calls:
                   CUSUM

       Called by:
                   training_subsequence_finder

       Input Parameters:
                   x - a time series of length of at least 30 with (nearly) normal distribution
                   starting_k_value - the starting value of the CUSUM parameter k, default is 3
                   starting_H_value - the starting value of the  CUSUM parameter H, default is 5


       Returns:
                   parameter_obj - a list/object of length 2, where
                          parameter_obj[1] - the smallest CUSUM k value for which x is in
                          statistical control
                          parameter_obj[2] - the smallest CUSUM H value for which x is in
                          statistical control
***************************************************************
```

OOPS **NOTE: In the current version of the code, H stays fixed at 5 and it is not modified at all. It might need to be messed with in future versions of CAD. But, H, as it stands now, can be left out of the code.**

```
FUNCTION parameter_finder(x, starting_k_value = 3, starting_H_value = 5)
      INIT parameters_found <- FALSE
      x_bar <- MEAN(x)
      sigma <- STANDARD_DEVIATION(x)

      WHILE (NOT parameters_found)
            temp_obj <- CUSUM(x, x_bar, sigma, H, k)
            IF (temp_obj[3] = NULL AND temp_obj[4] = NULL ) THEN
                  parameters_found <- TRUE
            ELSE
                  IF k > 20 THEN
                        WARNING("Something is off in CUSUM land")
                        k <- NULL
                        parameters_found <- TRUE
                  ELSE
                        k <- k +1
                  ENDIF
            ENDIF
      ENDWHILE

      RETURN([k, H])
ENDFUNCTION
```

```
***************************************************************************
       evaluate_CUSUM_results:
                    A function that finds the indices of anomalies, if there are any, of the time
                    series x, given its mean, standard deviation, and the minimum CUSUM parameters
                    H and k for which x should be in statistical control.

       Calls:
                    turning_points_finder
                    interval_finder
                    CUSUM

       Called by:
                    anomaly_finder

       Input Parameters:
                    x  - a time series length at least 30
                    x_bar - the mean of x
                    sigma - the standard deviation of x
                    k - CUSUM parameter k
                    H - CUSUM parameter H
                    type - the type of the anomaly "upper" or "lower"
                    lambda - the minimum length of the anomalous subsequence the code should
                           detect; the default is 5


       Returns:
                    indices - The indices of the anomalies within x.
****************************************************************


FUNCTION evaluate_CUSUM_results(x, x_bar, sigma, H=5, k, type, lambda)

      INIT hi_sum_indices <- NULL
      INIT low_sum_indices <- NULL
      INIT indices <- NULL

      CUSUM_obj <- CUSUM(x, x_bar, sigma, H, k)
      low_sums<-CUSUM_obj[1]
      hi_sums<-CUSUM_obj[2]
      upper_viol_index<-CUSUM_obj[3]
      lower_viol_index<-CUSUM_obj[4]
      lower_viol<-low_sums[lower_viol_index]
      upper_viol<-hi_sums[upper_viol_index]

      IF (LEN(upper_viol_index) > 0)THEN
            high_sum_turning_pts <- turning_points_finder(hi_sums)
            IF (high_sum_turning_pts[1]!= 1) THEN
                  high_sum_turning_pts <- CONCATENATE(1, high_sum_turning_pts)
            ENDIF
            hi_sum_df <- interval_finder(hi_sums, high_sum_turning_pts, type)
      ENDIF

      IF ( LEN(lower_viol_index) > 0) THEN
           low_sum_turning_pts <- turning_points_finder(low_sums)
           IF (low_sum_turning_pts[1] !=  1) THEN
                  low_sum_turning_pts <- CONCAT(1, low_sum_turning_pts)
            ENDIF
           low_sum_df <- interval_finder(low_sums, low_sum_turning_pts, type)
      ENDIF

      IF (type = "lower") THEN
      ############# Finding "lower" anomalies ##############
            #finding the indices of the decreasing terms for the upper violations seq.
            IF(LEN(upper_viol_index) > 0)THEN
                  dummy_df <-
                  SELECT
                        “left_index”,”right_index”
                  FROM
                        hi_sum_df
                  WHERE
                        (sign = “decreasing”) AND ((left_index-right_index)>lambda)


                  IF dummy_df != NULL THEN
                        n <- NUMBER_OF_ROWS(dummy_df)
                        FOR i FROM 1 TO n
                              IF (left_index[i] != NULL AND right_index != NULL) THEN
                                    interval <- [FROM left_index[i] TO right_index[i]]
                                    interval <- interval INTERSECT upper_viol_index
                                    IF (LEN(interval) > lambda) THEN
                                          hi_sum_indices <- CONCATENATE(hi_sum_indices,
                                                            interval)
                                    ENDIF
                              ENDIF
                        ENDFOR
                  ENDIF
            ENDIF

            #finding the indices of the decreasing terms for the lower violations seq.
            IF (LEN(lower_viol_index) > 0) THEN
                  dummy_df <-

                  SELECT
                        “left_index”,”right_index”
                  FROM
                        low_sum_df
                  WHERE
                        (sign = “decreasing”) AND ((right_index - left_index) > lambda)

                  IF dummy_df != NULL THEN
                        n <- NUMBER_OF_ROWS(dummy_df)
                        FOR i FROM 1 TO n
                              IF (left_index[i] != NULL AND right_index != NULL) THEN
                                    interval <- [FROM left_index[i] TO right_index[i]]
                                    interval <- interval INTERSECT lower_viol_index
                                    IF (LEN(interval) > lambda) THEN
                                          low_sum_indices  <-  CONCATENATE(low_sum_indices,
                                                                  interval)
                                    ENDIF
                              ENDIF
                        ENDFOR
                  ENDIF
            ENDIF

            indices <- CONCATENATE(hi_sum_indices, low_sum_indices)

      ELSE
      ############ Finding “upper” anomalies ##############
```

```
            #finding the indices of the increasing terms for the upper violations seq.
            IF (LEN(upper_viol_index) > 0) THEN
                  dummy_df <-

                  SELECT
                        “left_index”,”right_index”
                  FROM
                        hi_sum_df
                  WHERE
                        (sign = “increasing”) AND ((right_index - left_index) > lambda)

                  IF dummy_df != NULL THEN
                        n <- NUMBER_OF_ROWS(dummy_df)
                        FOR i FROM 1 TO n
                              IF (left_index[i] != NULL AND right_index != NULL) THEN
                                    interval <- [FROM left_index[i] TO right_index[i]]
                                    interval <- interval INTERSECT upper_viol_index
                                    IF (LEN(interval) > lambda) THEN
                                          hi_sum_indices <- CONCATENATE(hi_sum_indices,
                                                            interval)
                                    ENDIF
                              ENDIF
                        ENDFOR
                  ENDIF
            ENDIF

            #finding the indices of the increasing terms for the lower violations seq.
            IF (LEN(lower_viol_index) > 0) THEN
                  dummy_df <-

                  SELECT
                        “left_index”,”right_index”
                  FROM
                        low_sum_df
                  WHERE
                        (sign = “increasing”) AND ((right_index - left_index) > lambda)

                  IF dummy_df != NULL THEN
                        n <- NUMBER_OF_ROWS(dummy_df)
                        FOR i FROM 1 TO n
                              IF (left_index[i] != NULL AND right_index != NULL) THEN
                                    interval <- [FROM left_index[i] TO right_index[i]]
                                    interval <- interval INTERSECT lower_viol_index
                                    IF (LEN(interval) > lambda) THEN
                                          low_sum_indices  <-  CONCATENATE(low_sum_indices,
                                                                  interval)
                                    ENDIF
                              ENDIF
                        ENDFOR
                  ENDIF
            ENDIF

            indices <- CONCATENATE(hi_sum_indices, low_sum_indices)
      ENDIF

      RETURN(indices)

ENDFUNCTION
```

```
******************************************************
       turning_points_finder:
                   A recursively defined function that takes a sequence of numbers, x,  and finds
                   the indices of the values at which the sequence turns from increasing to
                   decreasing or vice-versa
       Calls:
                   SIGN

       Called by:
                   evaluate_CUSUM_results

       Input Parameters:
                   x  - a sequence of numbers longer than 1
                   turning_points - recursive variable, set to NULL when calling the function
                   sgn - recursive variable, set to 0 when calling the function
                   index - recursive variable, set to 1 when calling the function



       Returns:
                   turning_points - The indices of the turning points of x
*****************************************************************


FUNCTION turning_points_finder(x, turning_points = NULL, sgn = 0, index = 1)
      n <- LEN(turning_points)

      IF ((LEN(x) = 1) OR (LEN(x)  = 0)) THEN
            IF (LEN(turning_points) > 0)THEN
                  turning_points <- CONCATENATE(turning_points, index)
            ENDIF
            RETURN(turning_points)
      ENDIF

      diff <- x[2] - x[1]

      IF (diff = 0) THEN
            IF (sgn != 0) THEN
                  turning_points <- c(turning_points, index)
            ENDIF
            sgn <- 0
            index <- index + 1
            RETURN(turning_points_finder( x[FROM 2 TO LEN(x)], turning_points, sgn,
                  index)

      ELSE
            IF (SIGN(diff) = sgn) THEN
                  index <- index + 1
                  RETURN(turning_points_finder( x[FROM 2 TO LEN(x)], turning_points, sgn,
                        index)
            ELSE
                  sgn <- SIGN(diff)
                  turning_points <- CONCATENATE( turning_points, index)
                  index <- index + 1
                  RETURN(turning_points_finder( x[FROM 2 TO LEN(x)], turning_points, sgn,
                        index)
            ENDIF
      ENDIF
ENDFUNCTION
```

```
*****************************************************
       interval_finder:
                   Given a sequence, x, this function finds the endpoints of the monotone
                   increasing, monotone decreasing and constant subsequences it is composed of.

       Calls:
                   sign_finder

       Called by:
                   evaluate_CUSUM_results

       Input Parameters:
                   x  - a sequence of numbers longer than 1
                   tp_x - the indices of the turning points of x

       Returns:
                   df - A table with the following schema:
                          right_index: right indices of the intervals
                          left_index: left indices of the intervals
                          right_value: right endpoint values
                          left_value: left endpoint values
                          sign: "increasing", "decreasing", "constant"

****************************************************************************


FUNCTION interval_finder(x, tp_x)

      IF ((LEN(x) < = 1) OR LEN(tp_x) < = 1) THEN
            RETURN(NULL)
      ENDIF

      INIT sign <- NULL

      n <- LEN(tp_x)
      left_endpt_indices <- tp_x[ FROM 1 TO (n - 1)]
      right_endpt_indices <- tp_x[ FROM 2 TO n]

      tp_values1 <- x[left_endpt_indices]
      shifted <- left_endpt_indices + 1   #1 is added to each element of left_endpt_indices
      pt_value_after_tp <- x[shifted]



      FOR i FROM 1 TO (n-1)
            sign[i] <- sign_finder(tp_values1[i], pt_value_after_tp[i])
      ENDFOR

      CREATE TABLE df
            left_endpt_indices
            right_endpt_indices
            sign
      END CREATE TABLE

      RETURN(df)

ENDFUNCTION
```

I had some safety checks ensuring that the values of the input sequence between the turning points were indeed monotone increasing, monotone decreasing or constant. I excluded that from the code here.

```
**************************************************************
        sign_finder:
                    Given two values, a1 and a2,  it returns "increasing" if (a2 - a1) >0
                    returns "decreasing" if (a2 - a1) < 0, return "constant" if a2 = a1.

        Calls:
                    none

        Called by:
                    interval_finder

        Input Parameters:
                    a1  - a numeric value
                    a2 - a numeric value

        Returns:
                    sign - a factor variable with "increasing", "decreasing" or "constant" as
                    possible values

*************************************************************************************
```

```
FUNCTION sign_finder(x1, x2)
       IF (x2 - x1) > 0 THEN
              RETURN("increasing")
       ELSE
              IF (x2 - x1) < 0 THEN
                     RETURN("decreasing")
              ELSE
                     RETURN("constant")
              ENDIF
       ENDIF
ENDFUNCTION
```

```
********************************************************
       SIGN:
                    Given a value x, it returns 1 if x is positive, -1 if x is negative and 0 if
                    x =0.
       Calls:
                    none

       Called by:
                    turning_points_finder

       Input Parameters:
                    x  - a numeric value

       Returns:
                    s - a numeric value of -1, 0 or 1

*****************************************************************************
```

```
FUNCTION SIGN(x)
      IF (x < 0) THEN
            RETURN(-1)
      ELSE
            IF (x > 0) THEN
                  RETURN(1)
            ELSE
                  RETURN(0)
            ENDIF
      ENDIF
ENDFUNCTION
```