NAG Library Routine Document

G13NBF

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details.

1 Purpose

G13NBF detects change points in a univariate time series, that is, the time points at which some feature of the data, for example the mean, changes. Change points are detected using the PELT (Pruned Exact Linear Time) algorithm and a user-supplied cost function.

2 Specification

```
SUBROUTINE G13NBF (N, BETA, MINSS, K, COSTFN, NTAU, TAU, Y, IUSER,          &
                   RUSER, IFAIL)

INTEGER             N, MINSS, NTAU, TAU(N), IUSER(*), IFAIL
REAL (KIND=nag_wp) BETA, K, Y(*), RUSER(*)
EXTERNAL            COSTFN
```

3 Description

Let $y_{1:n} = \{y_j : j = 1, 2, \ldots, n\}$ denote a series of data and $\tau = \{\tau_i : i = 1, 2, \ldots, m\}$ denote a set of m ordered (strictly monotonic increasing) indices known as change points with $1 \leq \tau_i \leq n$ and $\tau_m = n$. For ease of notation we also define $\tau_0 = 0$. The m change points, τ, split the data into m segments, with the ith segment being of length n_i and containing $y_{\tau_{i-1}+1:\tau_i}$.

Given a user-supplied cost function, $C(y_{\tau_{i-1}+1:\tau_i})$ G13NBF solves

$$\underset{m,\tau}{\text{minimize}} \sum_{i=1}^{m} (C(y_{\tau_{i-1}+1:\tau_i}) + \beta) \tag{1}$$

where β is a penalty term used to control the number of change points. This minimization is performed using the PELT algorithm of Killick *et al.* (2012). The PELT algorithm is guaranteed to return the optimal solution to (1) if there exists a constant K such that

$$C\big(y_{(u+1):v}\big) + C\big(y_{(v+1):w}\big) + K \leq C\big(y_{(u+1):w}\big) \tag{2}$$

for all $u < v < w$

4 References

Chen J and Gupta A K (2010) *Parametric Statistical Change Point Analysis With Applications to Genetics Medicine and Finance* **Second Edition** Birkhl̀user

Killick R, Fearnhead P and Eckely I A (2012) Optimal detection of changepoints with a linear computational cost *Journal of the American Statistical Association* **107:500** 1590–1598

5 Arguments

1: N – INTEGER *Input*

On entry: n, the length of the time series.

Constraint: N $\geq$ 2.

2: BETA – REAL (KIND=nag_wp) *Input*

On entry: β, the penalty term.

There are a number of standard ways of setting β, including:

SIC or BIC
$$\beta = p \times \log(n)$$

AIC
$$\beta = 2p$$

Hannan-Quinn
$$\beta = 2p \times \log(\log(n))$$

where p is the number of parameters being treated as estimated in each segment. The value of p will depend on the cost function being used.

If no penalty is required then set $\beta = 0$. Generally, the smaller the value of β the larger the number of suggested change points.

3: MINSS – INTEGER *Input*

On entry: the minimum distance between two change points, that is $\tau_i - \tau_{i-1} \geq$ MINSS.

Constraint: MINSS ≥ 2.

4: K – REAL (KIND=nag_wp) *Input*

On entry: K, the constant value that satisfies equation (2). If K exists, it is unlikely to be unique in such cases, it is recommened that the largest value of K, that satisfies equation (2), is chosen. No check is made that K is the correct value for the supplied cost function.

5: COSTFN – SUBROUTINE, supplied by the user. *External Procedure*

The cost function, C. COSTFN must calculate a vector of costs for a number of segments.

The specification of COSTFN is:

```
SUBROUTINE COSTFN (TS, NR, R, C, Y, IUSER, RUSER, INFO)
INTEGER            TS, NR, R(NR), IUSER(*), INFO
REAL (KIND=nag_wp) C(NR), Y(*), RUSER(*)
```

1: TS – INTEGER *Input*

On entry: a reference time point.

2: NR – INTEGER *Input*

On entry: number of segments being considered.

3: R(NR) – INTEGER array *Input*

On entry: time points which, along with TS, define the segments being considered, $0 \leq \mathrm{R}(i) \leq n$ for $i = 1, 2, \ldots \mathrm{NR}$.

4: C(NR) – REAL (KIND=nag_wp) array *Output*

On exit: the cost function, C, with

$$\mathrm{C}(i) = \begin{cases} C\left(y_{r_i:t}\right) & \text{if } t > r_i, \\ C\left(y_{t:r_i}\right) & \text{otherwise.} \end{cases}$$

where $t =$ TS and $r_i = \mathrm{R}(i)$.

It should be noted that if $t > r_i$ for any value of i then it will be true for all values of i. Therefore the inequality need only be tested once per call to COSTFN.

5: Y(∗) – REAL (KIND=nag_wp) array *User Data*

COSTFN is called with Y as supplied to G13NBF. You are free to use the array Y to supply information to COSTFN.

Y is supplied in addition to IUSER and RUSER for ease of coding as in most cases COSTFN will require (functions of) the time series, y.

6: IUSER(∗) – INTEGER array *User Workspace*

7: RUSER(∗) – REAL (KIND=nag_wp) array *User Workspace*

COSTFN is called with the arguments IUSER and RUSER as supplied to G13NBF. You should use the arrays IUSER and RUSER to supply information to COSTFN.

8: INFO – INTEGER *Input/Output*

On entry: INFO $= 0$.

On exit: set INFO to a nonzero value if you wish G13NBF to terminate with IFAIL $= 51$.

COSTFN must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which G13NBF is called. Arguments denoted as *Input* must **not** be changed by this procedure.

6: NTAU – INTEGER *Output*

On exit: m, the number of change points detected.

7: TAU(N) – INTEGER array *Output*

On exit: the first m elements of TAU hold the location of the change points. The ith segment is defined by $y_{(\tau_{i-1}+1)}$ to y_{τ_i}, where $\tau_0 = 0$ and $\tau_i = \mathrm{TAU}(i), 1 \leq i \leq m$.

The remainder of TAU is used as workspace.

8: Y(∗) – REAL (KIND=nag_wp) array *User Data*

Y is not used by G13NBF, but is passed directly to COSTFN and may be used to pass information to this routine. Y will usually be used to pass (functions of) the time series, y of interest.

9: IUSER(∗) – INTEGER array *User Workspace*

10: RUSER(∗) – REAL (KIND=nag_wp) array *User Workspace*

IUSER and RUSER are not used by G13NBF, but are passed directly to COSTFN and should be used to pass information to this routine.

11: IFAIL – INTEGER *Input/Output*

On entry: IFAIL must be set to 0, -1 or 1. If you are unfamiliar with this argument you should refer to Section 3.4 in How to Use the NAG Library and its Documentation for details.

For environments where it might be inappropriate to halt program execution when an error is detected, the value -1 or 1 is recommended. If the output of error messages is undesirable, then the value 1 is recommended. Otherwise, if you are not familiar with this argument, the recommended value is 0. **When the value -1 or 1 is used it is essential to test the value of IFAIL on exit.**

On exit: IFAIL $= 0$ unless the routine detects an error or a warning has been flagged (see Section 6).

6 Error Indicators and Warnings

If on entry IFAIL $= 0$ or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

Errors or warnings detected by the routine:

IFAIL $= 11$

On entry, N $= \langle value \rangle$.
Constraint: N ≥ 2.

IFAIL $= 31$

On entry, MINSS $= \langle value \rangle$.
Constraint: MINSS ≥ 2.

IFAIL $= 51$

User requested termination.

IFAIL $= -99$

An unexpected error has been triggered by this routine. Please contact NAG.

See Section 3.9 in How to Use the NAG Library and its Documentation for further information.

IFAIL $= -399$

Your licence key may have expired or may not have been installed correctly.

See Section 3.8 in How to Use the NAG Library and its Documentation for further information.

IFAIL $= -999$

Dynamic memory allocation failed.

See Section 3.7 in How to Use the NAG Library and its Documentation for further information.

7 Accuracy

Not applicable.

8 Parallelism and Performance

G13NBF is not threaded in any implementation.

9 Further Comments

G13NAF performs the same calculations for a cost function selected from a provided set of cost functions. If the required cost function belongs to this provided set then G13NAF can be used without the need to provide a cost function routine.

10 Example

This example identifies changes in the scale parameter, under the assumption that the data has a gamma distribution, for a simulated dataset with 100 observations. A penalty, β of 3.6 is used and the minimum segment size is set to 3. The shape parameter is fixed at 2.1 across the whole input series.

The cost function used is

$$C(y_{\tau_{i-1}+1:\tau_i}) = 2an_i(\log S_i - \log(an_i))$$

where a is a shape parameter that is fixed for all segments and $n_i = \tau_i - \tau_{i-1} + 1$.

10.1 Program Text

```
!   G13NBF Example Program Text
!   Mark 26 Release. NAG Copyright 2016.

    Module g13nbfe_mod

!     G13NBF Example Program Module:
!            Parameters and User-defined Routines

!     .. Use Statements ..
      Use nag_library, Only: nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Accessibility Statements ..
      Private
      Public                           :: costfn, get_data
    Contains
      Subroutine costfn(ts,nr,r,c,y,iuser,ruser,info)
!       Cost function, C. This cost function is based on the likelihood of
!       the gamma distribution

!       .. Scalar Arguments ..
        Integer, Intent (Inout)        :: info
        Integer, Intent (In)           :: nr, ts
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out) :: c(nr)
        Real (Kind=nag_wp), Intent (Inout) :: ruser(*), y(0:*)
        Integer, Intent (Inout)        :: iuser(*)
        Integer, Intent (In)           :: r(nr)
!       .. Local Scalars ..
        Real (Kind=nag_wp)             :: dn, shape, si
        Integer                        :: i
!       .. Intrinsic Procedures ..
        Intrinsic                      :: log, real
!       .. Executable Statements ..
        Continue

!       RUSER(1) holds the shape parameter (a) for the gamma distribution
        shape = ruser(1)

!       Test which way around TS and R are (only needs to be done once)
        If (ts<r(1)) Then
          Do i = 1, nr
            si = y(r(i)) - y(ts)
            dn = real(r(i)-ts,kind=nag_wp)
            c(i) = 2.0_nag_wp*dn*shape*(log(si)-log(dn*shape))
          End Do

        Else
          Do i = 1, nr
            si = y(ts) - y(r(i))
            dn = real(ts-r(i),kind=nag_wp)
            c(i) = 2.0_nag_wp*dn*shape*(log(si)-log(dn*shape))
          End Do
        End If

!       Set info nonzero to terminate execution for any reason
        info = 0
      End Subroutine costfn

      Subroutine get_data(nin,n,k,y,iuser,ruser)
!       Read in data that is specific to the cost function

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (Out) :: k
        Integer, Intent (In)           :: n, nin
!       .. Array Arguments ..
        Real (Kind=nag_wp), Allocatable, Intent (Out) :: ruser(:), y(:)
        Integer, Allocatable, Intent (Out) :: iuser(:)
!       .. Local Scalars ..
```

```
        Real (Kind=nag_wp)             :: shape
        Integer                        :: i
!       .. Executable Statements ..
        Continue

!       Read in the series of interest
!       NB: we are starting Y allocation at 0 as we manipulate
!       the data in Y in a moment
        Allocate (y(0:n))
        Read (nin,*) y(1:n)

!       Read in the shape parameter for the Gamma distribution
        Read (nin,*) shape

!       Store the shape parameter in RUSER. IUSER is not used
        Allocate (ruser(1),iuser(0))
        ruser(1) = shape

!       The cost function is a function of the sum of Y, so for
!       efficiency we will calculate the cumulative sum
!       It should be noted that this may introduce some rounding issues
!       with very extreme data
        y(0) = 0.0_nag_wp
        Do i = 1, n
          y(i) = y(i-1) + y(i)
        End Do

!       The value of K is defined by the cost function being used
!       in this example a value of 0.0 is the required value
        k = 0.0_nag_wp

        Return
      End Subroutine get_data
    End Module g13nbfe_mod

    Program g13nbfe

!     .. Use Statements ..
      Use nag_library, Only: g13nbf, nag_wp
      Use g13nbfe_mod, Only: costfn, get_data
!     .. Implicit None Statement ..
      Implicit None
!     .. Parameters ..
      Integer, Parameter               :: nin = 5, nout = 6
!     .. Local Scalars ..
      Real (Kind=nag_wp)               :: beta, k
      Integer                          :: i, ifail, minss, n, ntau
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable  :: ruser(:), y(:)
      Integer, Allocatable             :: iuser(:), tau(:)
!     .. Intrinsic Procedures ..
      Intrinsic                        :: repeat
!     .. Executable Statements ..
      Continue
      Write (nout,*) 'G13NBF Example Program Results'
      Write (nout,*)

!     Skip heading in data file
      Read (nin,*)

!     Read in the problem size, penalty and minimum segment size
      Read (nin,*) n, beta, minss

!     Read in the rest of the data, that (may be) dependent on the cost
!     function
      Call get_data(nin,n,k,y,iuser,ruser)

!     Allocate output arrays
      Allocate (tau(n))

!     Call routine to detect change points
```

```
      ifail = 0
      Call g13nbf(n,beta,minss,k,costfn,ntau,tau,y,iuser,ruser,ifail)

!     Display the results
      Write (nout,99999) ' -- Change Points --'
      Write (nout,99999) ' Number     Position'
      Write (nout,99999) repeat('=',21)
      Do i = 1, ntau
        Write (nout,99998) i, tau(i)
      End Do

99999 Format (1X,A)
99998 Format (1X,I4,7X,I6)
    End Program g13nbfe
```

10.2 Program Data

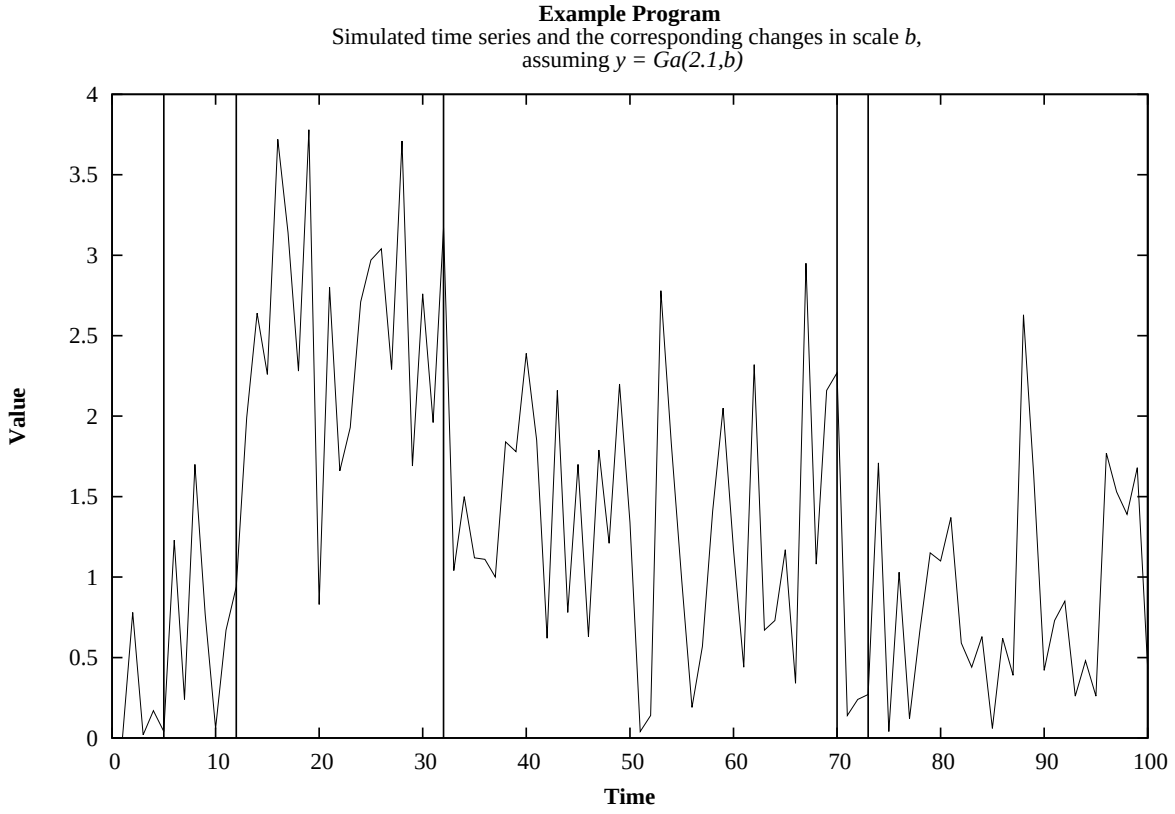
```
G13NBF Example Program Data
100    3.4   3 :: N,BETA,MINSS
 0.00  0.78  0.02  0.17  0.04  1.23  0.24  1.70  0.77  0.06
 0.67  0.94  1.99  2.64  2.26  3.72  3.14  2.28  3.78  0.83
 2.80  1.66  1.93  2.71  2.97  3.04  2.29  3.71  1.69  2.76
 1.96  3.17  1.04  1.50  1.12  1.11  1.00  1.84  1.78  2.39
 1.85  0.62  2.16  0.78  1.70  0.63  1.79  1.21  2.20  1.34
 0.04  0.14  2.78  1.83  0.98  0.19  0.57  1.41  2.05  1.17
 0.44  2.32  0.67  0.73  1.17  0.34  2.95  1.08  2.16  2.27
 0.14  0.24  0.27  1.71  0.04  1.03  0.12  0.67  1.15  1.10
 1.37  0.59  0.44  0.63  0.06  0.62  0.39  2.63  1.63  0.42
 0.73  0.85  0.26  0.48  0.26  1.77  1.53  1.39  1.68  0.43 :: End of Y
2.1            :: shape parameter used in COSTFN
```

10.3 Program Results

```
 G13NBF Example Program Results

  -- Change Points --
  Number     Position
 ====================
    1            5
    2           12
    3           32
    4           70
    5           73
    6          100
```

This example plot shows the original data series and the estimated change points.


Example Program
Simulated time series and the corresponding changes in scale b,
assuming y = Ga(2.1,b)
Value
Time
0
0.5
1
1.5
2
2.5
3
3.5
4
0
10
20
30
40
50
60
70
80
90
100