NAG Library Routine Document

G13NEF

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

G13NEF 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 binary segmentation for a user-supplied cost function.

2 Specification

```
SUBROUTINE G13NEF (N, BETA, MINSS, MDEPTH, CHGPFN, NTAU, TAU, Y, IUSER,    &
                   RUSER, IFAIL)

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

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 cost function, $C(y_{\tau_{i-1}+1:\tau_i})$, G13NEF gives an approximate solution to

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

where β is a penalty term used to control the number of change points. The solution is obtained in an iterative manner as follows:

1. Set $u = 1$, $w = n$ and $k = 0$
2. Set $k = k + 1$. If $k > K$, where K is a user-supplied control parameter, then terminate the process for this segment.
3. Find v that minimizes
$$C(y_{u:v}) + C(y_{v+1:w})$$
4. Test
$$C(y_{u:v}) + C(y_{v+1:w}) + \beta < C(y_{u:w}) \qquad (1)$$
5. If inequality (1) is false then the process is terminated for this segment.
6. If inequality (1) is true, then v is added to the set of change points, and the segment is split into two subsegments, $y_{u:v}$ and $y_{v+1:w}$. The whole process is repeated from step 2 independently on each subsegment, with the relevant changes to the definition of u and w (i.e., w is set to v when processing the left hand subsegment and u is set to $v + 1$ when processing the right hand subsegment.

The change points are ordered to give τ.

4 References

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

5 Arguments

1: N – INTEGER *Input*

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

Constraint: $\text{N} \geq 2$.

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 $\tau_i - \tau_{i-1} \geq \text{MINSS}$.

Constraint: $\text{MINSS} \geq 2$.

4: MDEPTH – INTEGER *Input*

On entry: K, the maximum depth for the iterative process, which in turn puts an upper limit on the number of change points with $m \leq 2^K$.

If $K \leq 0$ then no limit is put on the depth of the iterative process and no upper limit is put on the number of change points, other than that inherent in the length of the series and the value of MINSS.

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

CHGPFN must calculate a proposed change point, and the associated costs, within a specified segment.

The specification of CHGPFN is:

```
SUBROUTINE CHGPFN (SIDE, U, W, MINSS, V, COST, Y, IUSER, RUSER,     &
                   INFO)

INTEGER            SIDE, U, W, MINSS, V, IUSER(*), INFO
REAL (KIND=nag_wp) COST(3), Y(*), RUSER(*)
```

1: SIDE – INTEGER *Input*

On entry: flag indicating what CHGPFN must calculate and at which point of the Binary Segmentation it has been called.

SIDE $= -1$
: only $C(y_{u:w})$ need be calculated and returned in COST(1), neither V nor the other elements of COST need be set. In this case, $u = 1$ and $w = $ n.

SIDE $= 0$
: all elements of COST and V must be set. In this case, $u = 1$ and $w = $ n.

SIDE $= 1$
: the segment, $y_{u:w}$, is a left hand side subsegment from a previous iteration of the Binary Segmentation algorithm. All elements of COST and V must be set.

SIDE $= 2$
: the segment, $y_{u:w}$, is a right hand side subsegment from a previous iteration of the Binary Segmentation algorithm. All elements of COST and V must be set.

The distinction between SIDE $= 1$ and 2 may allow for CHGPFN to be implemented in a more efficient manner. See section Section 10 for one such example.

The first call to CHGPFN will always have SIDE $= -1$ and the second call will always have SIDE $= 0$. All subsequent calls will be made with SIDE $= 1$ or 2.

2: U – INTEGER *Input*

On entry: u, the start of the segment of interest.

3: W – INTEGER *Input*

On entry: w, the end of the segment of interest.

4: MINSS – INTEGER *Input*

On entry: the minimum distance between two change points, as passed to G13NEF.

5: V – INTEGER *Output*

On exit: if SIDE $= -1$ then V need not be set.

if SIDE $\neq -1$ then v, the proposed change point. That is, the value which minimizes

$$\underset{v}{\text{minimize}} C(y_{u:v}) + C(y_{v+1:w})$$

for $v = u + \text{MINSS} - 1$ to $w - \text{MINSS}$.

6: COST(3) – REAL (KIND=nag_wp) array *Output*

On exit: costs associated with the proposed change point, v.

If SIDE $= -1$ then COST$(1) = C(y_{u:w})$ and the remaining two elements of COST need not be set.

If SIDE $\neq -1$ then

COST$(1) = C(y_{u:v}) + C(y_{v+1:w})$.

COST$(2) = C(y_{u:v})$.

COST$(3) = C(y_{v+1:w})$.

7: Y$(*)$ – REAL (KIND=nag_wp) array *User Data*

CHGPFN is called with Y as supplied to G13NEF. You are free to use the array Y to supply information to CHGPFN.

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

8: IUSER($*$) – INTEGER array *User Workspace*
9: RUSER($*$) – REAL (KIND=nag_wp) array *User Workspace*

CHGPFN is called with the arguments IUSER and RUSER as supplied to G13NEF. You should use the arrays IUSER and RUSER to supply information to CHGPFN.

10: INFO – INTEGER *Input/Output*

On entry: INFO $= 0$.

On exit: in most circumstances INFO should remain unchanged.

If INFO is set to a strictly positive value then G13NEF terminates with IFAIL $= 51$.

If INFO is set to a strictly negative value the current segment is skipped (i.e., no change points are considered in this segment) and G13NEF continues as normal. If INFO was set to a strictly negative value at any point and no other errors occur then G13NEF will terminate with IFAIL $= 52$.

CHGPFN must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which G13NEF 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($*$) – INTEGER array *Output*

Note: the dimension of the array TAU must be at least $\min\left(\text{ceiling}\frac{\text{N}}{\text{MINSS}}, 2^{\text{MDEPTH}}\right)$ if MDEPTH > 0, and at least $\text{ceiling}\frac{\text{N}}{\text{MINSS}}$ otherwise.

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 $\tau_i = \text{TAU}(i), 1 \leq i \leq m$.

The remainder of TAU is used as workspace.

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

Y is not used by G13NEF, but is passed directly to CHGPFN 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 G13NEF, but are passed directly to CHGPFN 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 by setting INFO $= \langle value \rangle$.

IFAIL $= 52$

User requested a segment to be skipped by setting INFO $= \langle value \rangle$.

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

G13NEF is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

Please consult the X06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this routine. Please also consult the Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

G13NDF 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 G13NDF 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

```
!   G13NEF Example Program Text
!   Mark 26 Release. NAG Copyright 2016.
    Module g13nefe_mod

!     G13NEF 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                           :: chgpfn, get_data
    Contains
      Subroutine chgpfn(side,u,w,minss,v,cost,y,iuser,ruser,info)
!       Routine to calculate a proposed change point and associated cost
!       The cost is based on the likelihood of the gamma distribution

!       .. Use Statements ..
        Use nag_library, Only: x07caf, x07cbf
!       .. Scalar Arguments ..
        Integer, Intent (Inout)        :: info
        Integer, Intent (In)           :: minss, side, u, w
        Integer, Intent (Out)          :: v
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out) :: cost(3)
        Real (Kind=nag_wp), Intent (Inout) :: ruser(0:*), y(*)
        Integer, Intent (Inout)        :: iuser(*)
!       .. Local Scalars ..
        Real (Kind=nag_wp)             :: dn, shape, this_cost, tmp, ys
        Integer                        :: floc, i, li, lloc
!       .. Local Arrays ..
        Integer                        :: cexmode(3), texmode(3)
!       .. Intrinsic Procedures ..
        Intrinsic                      :: log
!       .. Executable Statements ..
        Continue

!       The gamma cost function used below can result in log(0) being taken
!       (if there is a segment of zeros in Y), this leads to a cost of -Inf
!       (which is correct), but we need to make sure that the compiler
!       doesn't stop at the creation of the -Inf

!       Save the current IEEE exception mode
        Call x07caf(cexmode)

!       Set the IEEE exception mode to not trap division by zero
        texmode(:) = cexmode(:)
        texmode(2) = 0
        Call x07cbf(texmode)

!       Extract shape from RUSER
        shape = ruser(0)

!       Calculate the first and last positions for potential change
```

```
!       points, conditional on the fact that each sub-segment must be
!       at least MINSS wide
        floc = u + minss - 1
        lloc = w - minss

!       In order to calculate the cost of having a change point at I, we
!       need to calculate C(Y(FLOC:I)) and C(Y(I+1:LLOC)), where C(.) is
!       the cost function (based on the gamma distribution in this example).
!       Rather than calculate these values at each call to CHGPFN we store
!       the values for later use

!       If SIDE = 1 (i.e. we are working with a left hand sub-segment),
!       we already have C(Y(FLOC:I)) for this value of FLOC, so only need
!       to calculate C(Y(I+1:LLOC)), similarly when SIDE = 2 we only need
!       to calculate C(Y(FLOC:I))
!       When SIDE = -1, we need the cost of the full segment, which we do
!       in a forwards manner (calculating C(Y(FLOC:I)) in the process), so
!       when SIDE = 0 we only need to calculate C(Y(I:1:LLOC))

!       Get the intermediate costs
        ys = 0.0_nag_wp
        dn = 0.0_nag_wp
        If (side==0 .Or. side==1) Then
!         RUSER(2*I) = C(Y(I+1:W))
          Do i = w, floc + 1, -1
            dn = dn + 1.0_nag_wp
            tmp = dn*shape
            ys = ys + y(i)
            ruser(2*i-2) = 2.0_nag_wp*tmp*(log(ys)-log(tmp))
          End Do

        Else
!         RUSER(2*I-1) = C(Y(U:I))
          If (side==-1) Then
            li = w
          Else
            li = lloc
          End If
          Do i = u, li
            dn = dn + 1.0_nag_wp
            tmp = dn*shape
            ys = ys + y(i)
            ruser(2*i-1) = 2.0_nag_wp*tmp*(log(ys)-log(tmp))
          End Do
        End If

        If (side>=0) Then
!         Need to find a potential change point
          v = 0
          cost(1) = 0.0_nag_wp

!         Loop over all possible change point locations
          Do i = floc, lloc
            this_cost = ruser(2*i-1) + ruser(2*i)

            If (this_cost<cost(1) .Or. v==0) Then
!             Update the proposed change point location
              v = i
              cost(1) = this_cost
              cost(2) = ruser(2*i-1)
              cost(3) = ruser(2*i)
            End If
          End Do
        Else
!         Need to calculate the cost for the full segment
          cost(1) = ruser(2*w-1)
!         No need to populate the rest of COST or V
        End If

!       Reset the IEEE exception mode back to what it was
        Call x07cbf(cexmode)
```

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

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

!       .. Scalar Arguments ..
        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
!       .. Executable Statements ..
        Continue

!       Read in the series of interest
        Allocate (y(1:n))
        Read (nin,*) y(1:n)

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

!       We are going to use RUSER for two purposes, firstly to store the shape
!       parameter, and we also need an additional 2*N elements of workspace
!       we reference from 0 to make the coding easier later

!       IUSER is not going to be used
        Allocate (iuser(0),ruser(0:2*n))

!       Store the shape parameter in the 0th element of RUSER
        ruser(0) = shape

!       We will be populating the other elements of RUSER in the first
!       call to CHGPFN

        Return
      End Subroutine get_data
    End Module g13nefe_mod

    Program g13nefe

!     .. Use Statements ..
      Use nag_library, Only: g13nef, nag_wp
      Use g13nefe_mod, Only: chgpfn, get_data
!     .. Implicit None Statement ..
      Implicit None
!     .. Parameters ..
      Integer, Parameter               :: nin = 5, nout = 6
!     .. Local Scalars ..
      Real (Kind=nag_wp)               :: beta
      Integer                          :: i, ifail, mdepth, 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,*) 'G13NEF Example Program Results'
      Write (nout,*)

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

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

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

!     Allocate output arrays
      Allocate (tau(n))

!     Call routine to detect change points
      ifail = 0
      Call g13nef(n,beta,minss,mdepth,chgpfn,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 g13nefe
```

10.2 Program Data

```
G13NEF Example Program Data
100    3.4   3   0 :: N,BETA,MINSS,MDEPTH
 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

```
 G13NEF 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)*