

NAG Library Function Document

nag_tsa_cp_pelt_user (g13nbc)

1 Purpose

nag_tsa_cp_pelt_user (g13nbc) 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 using one of a provided set of cost function.

2 Specification

```
#include <nag.h>
#include <naggl3.h>

void nag_tsa_cp_pelt_user (Integer n, double beta, Integer minss, double k,
    void (*costfn)(Integer ts, Integer nr, const Integer r[], double c[],
        Nag_Comm *comm, Integer *info),
    Integer *ntau, Integer tau[], Nag_Comm *comm, NagError *fail)
```

3 Description

Let $y_{1:n} = \{y_j : j = 1, 2, \dots, n\}$ denote a series of data and $\tau = \{\tau_i : i = 1, 2, \dots, m\}$ 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 i th segment being of length n_i and containing $y_{\tau_{i-1}+1:\tau_i}$.

Given a user-supplied cost function, $C(y_{\tau_{i-1}+1:\tau_i})$ nag_tsa_cp_pelt_user (g13nbc) solves

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

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

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

for all $u < v < w$

4 References

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

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** – double *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 $\tau_i - \tau_{i-1} \geq \mathbf{minss}$.
Constraint: $\mathbf{minss} \geq 2$.
- 4: **k** – double *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 recommended 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** – function, supplied by the user *External Function*
 The cost function, C . **costfn** must calculate a vector of costs for a number of segments.

The specification of **costfn** is:

```
void costfn (Integer ts, Integer nr, const Integer r[], double c[],
            Nag_Comm *comm, Integer *info)
```

1: **ts** – Integer *Input*

On entry: a reference time point.

2: **nr** – Integer *Input*

On entry: number of segments being considered.

3: **r[nr]** – const Integer *Input*

On entry: time points which, along with **ts**, define the segments being considered, $0 \leq \mathbf{r}[i-1] \leq n$ for $i = 1, 2, \dots, \mathbf{nr}$.

4: **c[nr]** – double *Output*

On exit: the cost function, C , with

$$\mathbf{c}[i-1] = \begin{cases} C(y_{r_i:t}) & \text{if } t > r_i, \\ C(y_{t:r_i}) & \text{otherwise.} \end{cases}$$

where $t = \mathbf{ts}$ and $r_i = \mathbf{r}[i-1]$.

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: **comm** – Nag_Comm *
 Pointer to structure of type Nag_Comm; the following members are relevant to **costfn**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be void *. Before calling nag_tsa_cp_pelt_user (g13nbc) you may allocate memory and initialize these pointers with various quantities for use by **costfn** when called from nag_tsa_cp_pelt_user (g13nbc) (see Section 3.2.1.1 in the Essential Introduction).

6: **info** – Integer * *Input/Output*

On entry: **info** = 0.

On exit: set **info** to a nonzero value if you wish nag_tsa_cp_pelt_user (g13nbc) to terminate with **fail.code** = NE_USER_STOP.

6: **ntau** – Integer * *Output*

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

7: **tau[n]** – Integer *Output*

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

The remainder of **tau** is used as workspace.

8: **comm** – Nag_Comm *
 The NAG communication argument (see Section 3.2.1.1 in the Essential Introduction).

9: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
 See Section 3.2.1.2 in the Essential Introduction for further information.

NE_BAD_PARAM

On entry, argument $\langle value \rangle$ had an illegal value.

NE_INT

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

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

NE_INTERNAL_ERROR

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.

An unexpected error has been triggered by this function. Please contact NAG.
 See Section 3.6.6 in the Essential Introduction for further information.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in the Essential Introduction for further information.

NE_USER_STOP

User requested termination.

7 Accuracy

Not applicable.

8 Parallelism and Performance

Not applicable.

9 Further Comments

nag_tsa_cp_pelt (g13nac) 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 nag_tsa_cp_pelt (g13nac) 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

```

/* nag_tsa_cp_pelt_user (g13nbc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 25, 2014.
 */
/* Pre-processor includes */
#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg13.h>
#include <nagx02.h>
#include <math.h>

/* Structure to hold extra information that the cost function requires */
typedef struct {
    Integer isinf;
    double shape;
    double *y;
} CostInfo;

/* Functions that are dependent on the cost function used */
void costfn(Integer ts, Integer nr, const Integer r[], double c[],
            Nag_Comm *comm, Integer *info);
Integer get_data(Integer n, double *k, Nag_Comm *comm);
void clean_data(Nag_Comm *comm);

int main(void)

```

```

{
/* Integer scalar and array declarations */
Integer i, minss, n, ntau;
Integer exit_status = 0;
Integer *tau = 0;

/* NAG structures and types */
NagError fail;
Nag_Comm comm;

/* Double scalar and array declarations */
double beta, k;

/* Initialise the error structure */
INIT_FAIL(fail);

printf("nag_tsa_cp_pelt_user (g13nbc) Example Program Results\n\n");

/* Skip heading in data file */
#ifdef _WIN32
scanf_s("%*[\n] ");
#else
scanf("%*[\n] ");
#endif

/* Read in the problem size, penalty and minimum segment size */
#ifdef _WIN32
scanf_s("%"NAG_IFMT"%lf%"NAG_IFMT"%*[\n] ", &n, &beta, &minss);
#else
scanf("%"NAG_IFMT"%lf%"NAG_IFMT"%*[\n] ", &n, &beta, &minss);
#endif

/* Read in other data, that (may be) dependent on the cost function */
get_data(n, &k, &comm);

/* Allocate output arrays */
if (!(tau = NAG_ALLOC(n, Integer)))
{
printf("Allocation failure\n");
exit_status = -1;
goto END;
}

/* Call nag_tsa_cp_pelt_user (g13nbc) to detect change points */
nag_tsa_cp_pelt_user(n, beta, minss, k, costfn, &ntau, tau, &comm, &fail);
if (fail.code != NE_NOERROR)
{
printf("Error from nag_tsa_cp_pelt_user (g13nbc).\n%s\n", fail.message);
exit_status = 1;
goto END;
}

/* Display the results */
printf(" -- Change Points --\n");
printf(" Number      Position\n");
printf(" =====\n");
for (i = 0; i < ntau; i++)
{
printf(" %4"NAG_IFMT"          %6"NAG_IFMT"\n", i+1, tau[i]);
}

END:
NAG_FREE(tau);
clean_data(&comm);

return(exit_status);
}

void costfn(Integer ts, Integer nr, const Integer r[], double c[],
Nag_Comm *comm, Integer *info) {
double dn, shape, si;

```

```

Integer i;
CostInfo *ci;

ci = (CostInfo *) comm->p;

/* Get the shape parameter for the gamma distribution from comm */
shape = ci->shape;

/* Test which way around TS and R are (only needs to be done once) */
if (ts < r[0])
{
    for (i = 0; i < nr; i++)
    {
        si = ci->y[r[i]] - ci->y[ts];

        if (si <= 0.0)
        {
            /* -Inf */
            ci->isinf = 1;
            c[i] = -X02ALC;
        }
        else
        {
            dn = (double) (r[i]-ts);
            c[i] = 2.0*dn*shape*(log(si)-log(dn*shape));
        }
    }
}
else
{
    for (i = 0; i < nr; i++)
    {
        si = ci->y[ts] - ci->y[r[i]];

        if (si <= 0.0)
        {
            /* -Inf */
            ci->isinf = 1;
            c[i] = -X02ALC;
        }
        else
        {
            dn = (double) (ts-r[i]);
            c[i] = 2.0*dn*shape*(log(si)-log(dn*shape));
        }
    }
}

/* Set info nonzero to terminate execution for any reason */
*info = 0;
}

Integer get_data(Integer n, double *k, Nag_Comm *comm) {
/* Read in data that is specific to the cost function */
double shape;
Integer i;
CostInfo *ci;

/* Allocate some memory for the additional information structure */
/* This will be pointed to by comm->p */
comm->p = 0;
if (!(ci = NAG_ALLOC(1, CostInfo)))
{
    printf("Allocation failure\n");
    return -1;
}

/* Read in the series of interest */
/* NB: we are allocating n+1 elements to y as we manipulate the data
in Y in a moment */
if (!(ci->y = NAG_ALLOC(n+1, double)))

```

```

    {
        printf("Allocation failure\n");
        return -1;
    }

    /* Referencing y from 1 here to aid manipulation later */
    for (i = 1; i <= n; i++)
#ifdef _WIN32
        scanf_s("%lf",&(ci->y)[i]);
#else
        scanf("%lf",&(ci->y)[i]);
#endif
#ifdef _WIN32
        scanf_s("%*[\n] ");
#else
        scanf("%*[\n] ");
#endif

    /* Read in the shape parameter for the Gamma distribution */
#ifdef _WIN32
        scanf_s("%lf%*[\n] ",&shape);
#else
        scanf("%lf%*[\n] ",&shape);
#endif

    /* Store the shape parameter in CostInfo structure */
    ci->shape = shape;

    /* Set the warning flag to 0 */
    ci->isinf = 0;

    /* 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 */
    (ci->y)[0] = 0.0;
    for (i = 1; i <= n; i++)
        (ci->y)[i] += (ci->y)[i-1];

    /* 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;

    /* Store pointer to CostInfo structure in Nag_Comm */
    comm->p = (void *) ci;

    return 0;
}

void clean_data(Nag_Comm *comm) {
    /* Free any memory allocated in get_data */
    CostInfo *ci;

    if (comm->p) {
        ci = (CostInfo *) comm->p;
        NAG_FREE(ci->y);
    }

    NAG_FREE(comm->p);
}

```

10.2 Program Data

```

nag_tsa_cp_pelt_user (g13nbc) 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

nag_tsa_cp_pelt_user (g13nbc) 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.

