

# NAG Library Function Document

## nag\_tsa\_cp\_pelt (g13nac)

### 1 Purpose

nag\_tsa\_cp\_pelt (g13nac) 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 functions.

### 2 Specification

```
#include <nag.h>
#include <nagg13.h>

void nag_tsa_cp_pelt (Nag_TS_ChangeType ctype, Integer n, const double y[],
    double beta, Integer minss, const double param[], Integer *ntau,
    Integer tau[], double sparam[], 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,  $\tau$ , 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 cost function,  $C(y_{\tau_{i-1}+1:\tau_i})$  nag\_tsa\_cp\_pelt (g13nac) solves

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

where  $\beta$  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  $u < v < w$ .

nag\_tsa\_cp\_pelt (g13nac) supplies four families of cost function. Each cost function assumes that the series,  $y$ , comes from some distribution,  $D(\Theta)$ . The parameter space,  $\Theta = \{\theta, \phi\}$  is subdivided into  $\theta$  containing those parameters allowed to differ in each segment and  $\phi$  those parameters treated as constant across all segments. All four cost functions can then be described in terms of the likelihood function,  $L$  and are given by:

$$C(y_{(\tau_{i-1}+1):\tau_i}) = -2 \log L(\hat{\theta}_i, \phi | y_{(\tau_{i-1}+1):\tau_i})$$

where  $\hat{\theta}_i$  is the maximum likelihood estimate of  $\theta$  within the  $i$ th segment. In all four cases setting  $K = 0$  satisfies equation (2). Four distributions are available: Normal, Gamma, Exponential and Poisson. Letting

$$S_i = \sum_{j=\tau_{i-1}}^{\tau_i} y_j$$

the log-likelihoods and cost functions for the four distributions, and the available subdivisions of the parameter space are:

**Normal distribution:**  $\Theta = \{\mu, \sigma^2\}$

$$-2\log L = \sum_{i=1}^m \sum_{j=\tau_{i-1}}^{\tau_i} \log(2\pi) + \log(\sigma_i^2) + \frac{(y_j - \mu_i)^2}{\sigma_i^2}$$

*Mean changes:*  $\theta = \{\mu\}$

$$C(y_{\tau_{i-1}+1:\tau_i}) = \sum_{j=\tau_{i-1}}^{\tau_i} \frac{(y_j - n_i^{-1}S_i)^2}{\sigma^2}$$

*Variance changes:*  $\theta = \{\sigma^2\}$

$$C(y_{\tau_{i-1}+1:\tau_i}) = n_i \left( \log \left( \sum_{j=\tau_{i-1}}^{\tau_i} (y_j - \mu)^2 \right) - \log n_i \right)$$

*Both mean and variance change:*  $\theta = \{\mu, \sigma^2\}$

$$C(y_{\tau_{i-1}+1:\tau_i}) = n_i \left( \log \left( \sum_{j=\tau_{i-1}}^{\tau_i} (y_j - n_i^{-1}S_i)^2 \right) - \log n_i \right)$$

**Gamma distribution:**  $\Theta = \{a, b\}$

$$-2\log L = 2 \times \sum_{i=1}^m \sum_{j=\tau_{i-1}}^{\tau_i} \log \Gamma(a_i) + a_i \log b_i + (1 - a_i) \log y_j + \frac{y_j}{b_i}$$

*Scale changes:*  $\theta = \{b\}$

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

**Exponential Distribution:**  $\Theta = \{\lambda\}$

$$-2\log L = 2 \times \sum_{i=1}^m \sum_{j=\tau_{i-1}}^{\tau_i} \log \lambda_i + \frac{y_j}{\lambda_i}$$

*Mean changes:*  $\theta = \{\lambda\}$

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

**Poisson distribution:**  $\Theta = \{\lambda\}$

$$-2\log L = 2 \times \sum_{i=1}^m \sum_{j=\tau_{i-1}}^{\tau_i} \lambda_i - \text{floor } y_j + 0.5 \log \lambda_i + \log \Gamma(\text{floor } y_j + 0.5 + 1)$$

*Mean changes:*  $\theta = \{\lambda\}$

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

when calculating  $S_i$  for the Poisson distribution, the sum is calculated for floor  $y_i + 0.5$  rather than  $y_i$ .

## 4 References

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

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: **ctype** – Nag\_TS\_ChangeType *Input*

*On entry:* a flag indicating the assumed distribution of the data and the type of change point being looked for.

**ctype** = Nag\_NormalMean

Data from a Normal distribution, looking for changes in the mean,  $\mu$ .

**ctype** = Nag\_NormalStd

Data from a Normal distribution, looking for changes in the standard deviation  $\sigma$ .

**ctype** = Nag\_NormalMeanStd

Data from a Normal distribution, looking for changes in the mean,  $\mu$  and standard deviation  $\sigma$ .

**ctype** = Nag\_GammaScale

Data from a Gamma distribution, looking for changes in the scale parameter  $b$ .

**ctype** = Nag\_ExponentialLambda

Data from an exponential distribution, looking for changes in  $\lambda$ .

**ctype** = Nag\_PoissonLambda

Data from a Poisson distribution, looking for changes in  $\lambda$ .

*C o n s t r a i n t :* **ctype** = Nag\_NormalMean, Nag\_NormalStd, Nag\_NormalMeanStd, Nag\_GammaScale, Nag\_ExponentialLambda or Nag\_PoissonLambda.

2: **n** – Integer *Input*

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

*Constraint:*  $n \geq 2$ .

3: **y[n]** – const double *Input*

*On entry:*  $y$ , the time series.

if **ctype** = Nag\_PoissonLambda, that is the data is assumed to come from a Poisson distribution, floor  $y + 0.5$  is used in all calculations.

*Constraints:*

if **ctype** = Nag\_GammaScale, Nag\_ExponentialLambda or Nag\_PoissonLambda,

$y[i - 1] \geq 0$ , for  $i = 1, 2, \dots, n$ ;

if **ctype** = Nag\_PoissonLambda, each value of  $y$  must be representable as an integer;

if **ctype**  $\neq$  Nag\_PoissonLambda, each value of  $y$  must be small enough such that  $y[i - 1]^2$ , for  $i = 1, 2, \dots, n$ , can be calculated without incurring overflow.

4: **beta** – double *Input*

*On entry:*  $\beta$ , the penalty term.

There are a number of standard ways of setting  $\beta$ , 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. This is usually set to 2 when **ctype** = Nag\_NormalMeanStd and 1 otherwise.

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

5: **minss** – Integer *Input*

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

*Constraint:* **minss**  $\geq 2$ .

6: **param**[1] – const double *Input*

*On entry:*  $\phi$ , values for the parameters that will be treated as fixed. If **ctype**  $\neq$  Nag\_GammaScale, **param** may be set to **NULL**.

If **ctype** = Nag\_NormalMean

if **param** is **NULL**,  $\sigma$ , the standard deviation of the Normal distribution, is estimated from the full input data. Otherwise  $\sigma = \mathbf{param}[0]$ .

If **ctype** = Nag\_NormalStd

If **param** is **NULL**,  $\mu$ , the mean of the Normal distribution, is estimated from the full input data. Otherwise  $\mu = \mathbf{param}[0]$ .

If **ctype** = Nag\_GammaScale, **param**[0] must hold the shape,  $a$ , for the Gamma distribution, otherwise **param** is not referenced.

*Constraint:* if **ctype** = Nag\_NormalMean or Nag\_GammaScale, **param**[0]  $> 0.0$ .

7: **ntau** – Integer \* *Output*

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

8: **tau**[*dim*] – 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  $y_{\tau_i}$ , where  $\tau_0 = 0$  and  $\tau_i = \mathbf{tau}[i - 1]$ ,  $1 \leq i \leq m$ .

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

9: **sparam**[*dim*] – double *Output*

*On exit:* the estimated values of the distribution parameters in each segment

**ctype** = Nag\_NormalMean, Nag\_NormalStd or Nag\_NormalMeanStd

**sparam**[ $2i - 2$ ] =  $\mu_i$  and **sparam**[ $2i - 1$ ] =  $\sigma_i$  for  $i = 1, 2, \dots, m$ , where  $\mu_i$  and  $\sigma_i$  is the mean and standard deviation, respectively, of the values of  $y$  in the  $i$ th segment.

It should be noted that  $\sigma_i = \sigma_j$  when **ctype** = Nag\_NormalMean and  $\mu_i = \mu_j$  when **ctype** = Nag\_NormalStd, for all  $i$  and  $j$ .

**ctype** = Nag\_GammaScale

**sparam**[ $2i - 2$ ] =  $a_i$  and **sparam**[ $2i - 1$ ] =  $b_i$  for  $i = 1, 2, \dots, m$ , where  $a_i$  and  $b_i$  are the shape and scale parameters, respectively, for the values of  $y$  in the  $i$ th segment. It should be noted that  $a_i = \mathbf{param}[0]$  for all  $i$ .

**ctype** = Nag\_ExponentialLambda or Nag\_PoissonLambda

**sparam**[ $i - 1$ ] =  $\lambda_i$  for  $i = 1, 2, \dots, m$ , where  $\lambda_i$  is the mean of the values of  $y$  in the  $i$ th segment.

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

10: **fail** – NagError \* *Input/Output*

The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

## 6 Error Indicators and Warnings

### NE\_ALLOC\_FAIL

Dynamic memory allocation failed.

See Section 2.3.1.2 in How to Use the NAG Library and its Documentation 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**  $\geq 2$ .

On entry, **n** =  $\langle value \rangle$ .

Constraint: **n**  $\geq 2$ .

### 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 2.7.6 in How to Use the NAG Library and its Documentation for further information.

### NE\_NO\_LICENCE

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

See Section 2.7.5 in How to Use the NAG Library and its Documentation for further information.

### NE\_REAL

On entry, **ctype** =  $\langle value \rangle$  and **param**[0] =  $\langle value \rangle$ .

Constraint: if **ctype** = Nag\_NormalMean or Nag\_GammaScale and **param** is not NULL, then **param**[0]  $> 0.0$ .

### NE\_REAL\_ARRAY

On entry, **ctype** =  $\langle value \rangle$  and **y**[ $\langle value \rangle$ ] =  $\langle value \rangle$ .

Constraint: if **ctype** = Nag\_GammaScale, Nag\_ExponentialLambda or Nag\_PoissonLambda then **y**[ $i - 1$ ]  $\geq 0.0$ , for  $i = 1, 2, \dots, \mathbf{n}$ .

On entry, **y**[ $\langle value \rangle$ ] =  $\langle value \rangle$ , is too large.

### NW\_TRUNCATED

To avoid overflow some truncation occurred when calculating the cost function,  $C$ . All output is returned as normal.

To avoid overflow some truncation occurred when calculating the parameter estimates returned in **sparam**. All output is returned as normal.

## 7 Accuracy

For efficiency reasons, when calculating the cost functions,  $C$  and the parameter estimates returned in **sparam**, this function makes use of the mathematical identities:

$$\sum_{j=u}^v y_j^2 = \sum_{j=1}^v y_j^2 - \sum_{j=1}^{u-1} y_j^2$$

and

$$\sum_{j=1}^n (y_j - \bar{y})^2 = \left( \sum_{j=1}^n y_j^2 \right) - n\bar{y}^2$$

where  $\bar{y} = n^{-1} \sum_{j=1}^n y_j$ .

The input data,  $y$ , is scaled in order to try and mitigate some of the known instabilities associated with using these formulations. The results returned by `nag_tsa_cp_pelt` (g13nac) should be sufficient for the majority of datasets. If a more stable method of calculating  $C$  is deemed necessary, `nag_tsa_cp_pelt_user` (g13nbc) can be used and the method chosen implemented in the user-supplied cost function.

## 8 Parallelism and Performance

`nag_tsa_cp_pelt` (g13nac) is not threaded in any implementation.

## 9 Further Comments

None.

## 10 Example

This example identifies changes in the mean, under the assumption that the data is normally distributed, for a simulated dataset with 100 observations. A BIC penalty is used, that is  $\beta = \log n \approx 4.6$ , the minimum segment size is set to 2 and the variance is fixed at 1 across the whole input series.

### 10.1 Program Text

```
/* nag_tsa_cp_pelt (g13nac) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */
/* Pre-processor includes */
#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg13.h>

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_TS_ChangeType ctype;
```

```

Nag_Boolean param_supplied;

/* Double scalar and array declarations */
double beta;
double *param = 0, *sparam = 0, *y = 0;

/* Character scalar and array declarations */
char cctype[40], cparam_supplied[40];

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

printf("nag_tsa_cp_pelt (g13nac) Example Program Results\n\n");

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

/* Read in the problem size */
#ifdef _WIN32
scanf_s("%" NAG_IFMT "%*[\n] ", &n);
#else
scanf("%" NAG_IFMT "%*[\n] ", &n);
#endif

/* Allocate memory to hold the input series */
if (!(y = NAG_ALLOC(n, double)))
{
printf("Allocation failure\n");
exit_status = -1;
goto END;
}

/* Read in the input series */
for (i = 0; i < n; i++) {
#ifdef _WIN32
scanf_s("%lf", &y[i]);
#else
scanf("%lf", &y[i]);
#endif
}
#ifdef _WIN32
scanf_s("%*[\n] ");
#else
scanf("%*[\n] ");
#endif

/* Read in the type of change point, penalty and minimum segment size */
#ifdef _WIN32
scanf_s("%39s%39s%lf%" NAG_IFMT "%*[\n] ", cctype,
(unsigned)_countof(cctype), cparam_supplied,
(unsigned)_countof(cparam_supplied), &beta, &minss);
#else
scanf("%39s%39s%lf%" NAG_IFMT "%*[\n] ", cctype, cparam_supplied, &beta,
&minss);
#endif
cctype = (Nag_TS_ChangeType) nag_enum_name_to_value(cctype);
param_supplied = (Nag_Boolean) nag_enum_name_to_value(cparam_supplied);

/* Read in the distribution parameter (if required) */
if (param_supplied) {
if (!(param = NAG_ALLOC(1, double)))
{
printf("Allocation failure\n");
exit_status = -1;
goto END;
}
}
#ifdef _WIN32

```

```

    scanf_s("%lf", &param[0]);
#else
    scanf("%lf", &param[0]);
#endif
#ifdef _WIN32
    scanf_s("%*[^\\n] ");
#else
    scanf("%*[^\\n] ");
#endif
}

/* Allocate output arrays */
if (!(tau = NAG_ALLOC(n, Integer)) ||
    !(sparam = NAG_ALLOC(2 * n + 2, double)))
{
    printf("Allocation failure\\n");
    exit_status = -1;
    goto END;
}

/* Call nag_tsa_cp_pelt (g13nac) to to detect change points */
nag_tsa_cp_pelt(ctype, n, y, beta, minss, param, &tau, tau, sparam, &fail);
if (fail.code != NE_NOERROR) {
    if (fail.code != NW_TRUNCATED) {
        printf("Error from nag_tsa_cp_pelt (g13nac).\\n%s\\n", fail.message);
        exit_status = 1;
        goto END;
    }
}

/* Display the results */
if (ctype == Nag_ExponentialLambda || ctype == Nag_PoissonLambda) {
    /* Exponential or Poisson distribution */
    printf(" -- Change Points --      Distribution\\n");
    printf("   Number      Position      Parameter\\n");
    printf(" =====\\n");
    for (i = 0; i < ntau; i++) {
        printf(" %4" NAG_IFMT "      %6" NAG_IFMT "      %12.2f\\n",
            i + 1, tau[i], sparam[i]);
    }
}
else {
    /* Normal or Gamma distribution */
    printf(" -- Change Points --      --- Distribution ---\\n");
    printf("   Number      Position      Parameters\\n");
    printf(" =====\\n");
    for (i = 0; i < ntau; i++) {
        printf(" %4" NAG_IFMT "      %6" NAG_IFMT "      %12.2f      %12.2f\\n",
            i + 1, tau[i], sparam[2 * i], sparam[2 * i + 1]);
    }
}
if (fail.code == NW_TRUNCATED) {
    printf("Some truncation occurred internally to avoid overflow.\\n");
}

END:
NAG_FREE(y);
NAG_FREE(param);
NAG_FREE(tau);
NAG_FREE(sparam);

return (exit_status);
}

```



## 10.2 Program Data

```
nag_tsa_cp_pelt (g13nac) Example Program Data
100
:: n
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
Nag_NormalMean Nag_TRUE 4.6 2 :: ctype,param_supplied,beta,minss
1.0 :: param[0]
```

## 10.3 Program Results

```
nag_tsa_cp_pelt (g13nac) Example Program Results
```

-- Change Points --		--- Distribution ---	
Number	Position	Parameters	
1	12	0.34	1.00
2	32	2.57	1.00
3	49	1.45	1.00
4	52	-0.48	1.00
5	70	1.20	1.00
6	100	-0.23	1.00

This example plot shows the original data series, the estimated change points and the estimated mean in each of the identified segments.

