NAG Library Function Document

nag_moving_average (g01wac)

1 Purpose

nag_moving_average (g01wac) calculates the mean and, optionally, the standard deviation using a rolling window for an arbitrary sized data stream.

2 Specification

```
#include <nag.h>
#include <nagg01.h>
```

```
void nag_moving_average (Integer m, Integer nb, const double x[],
     Nag_Weightstype iwt, const double wt[], Integer *pn, double rmean[],
     double rsd[], double rcomm[], NagError *fail)
```

3 Description

Given a sample of n observations, denoted by $x = \{x_i : i = 1, 2, \ldots, n\}$ and a set of weights, $w = \{w_j : j = 1, 2, \ldots, m\}$, nag_moving_average (g01wac) calculates the mean and, optionally, the standard deviation, in a rolling window of length m.

For the ith window the mean is defined as

$$\mu_i = \frac{\sum_{j=1}^{m} w_j x_{i+j-1}}{W} \tag{1}$$

and the standard deviation as

$$\sigma_i = \sqrt{\frac{\sum_{j=1}^{m} w_j \left(x_{i+j-1} - \mu_i\right)^2}{W - \frac{\sum_{j=1}^{m} w_j^2}{W}}} \tag{2}$$

with $W = \sum_{j=1}^{m} w_j$.

Four different types of weighting are possible:

(i) **No weights ($w_j = 1$)**

When no weights are required both the mean and standard deviations can be calculated in an iterative manner, with

$$\begin{aligned} \mu_{i+1} &= \mu_i + \tfrac{(x_{i+m} - x_i)}{m} \\ \sigma_{i+1}^2 &= (m-1)\sigma_i^2 + (x_{i+m} - \mu_i)^2 - (x_i - \mu_i)^2 - \tfrac{(x_{i+m} - x_i)^2}{m} \end{aligned}$$

where the initial values μ_1 and σ_1 are obtained using the one pass algorithm of West (1979).

(ii) **Each observation has its own weight**

In this case, rather than supplying a vector of m weights a vector of n weights is supplied instead, $v = \{v_j : j = 1, 2, \ldots, n\}$ and $w_j = v_{i+j-1}$ in (1) and (2).

If the standard deviations are not required then the mean is calculated using the iterative formula:

$$\begin{aligned} W_{i+1} &= W_i + (v_{i+m} - v_i) \\ \mu_{i+1} &= \mu_i + W_i^{-1}(v_{i+m}x_{i+m} - v_i x_i) \end{aligned}$$

where $W_1 = \sum_{i=1}^{m} v_i$ and $\mu_1 = W_1^{-1} \sum_{i=1}^{m} v_i x_i$.

If both the mean and standard deviation are required then the one pass algorithm of West (1979) is used in each window.

(iii) **Each position in the window has its own weight**

This is the case as described in (1) and (2), where the weight given to each observation differs depending on which summary is being produced. When these types of weights are specified both the mean and standard deviation are calculated by applying the one pass algorithm of West (1979) multiple times.

(iv) **Each position in the window has a weight equal to its position number ($w_j = j$)**

This is a special case of (iii).

If the standard deviations are not required then the mean is calculated using the iterative formula:

$$\begin{aligned} S_{i+1} &= S_i + (x_{i+m} - x_i) \\ \mu_{i+1} &= \mu_i + \tfrac{2(mx_{i+m} - S_i)}{m(m+1)} \end{aligned}$$

where $S_1 = \sum_{i=1}^{m} x_i$ and $\mu_1 = 2(m^2 + m)^{-1} S_1$.

If both the mean and standard deviation are required then the one pass algorithm of West is applied multiple times.

For large datasets, or where all the data is not available at the same time, x (and if each observation has its own weight, v) can be split into arbitrary sized blocks and nag_moving_average (g01wac) called multiple times.

4 References

Chan T F, Golub G H and Leveque R J (1982) *Updating Formulae and a Pairwise Algorithm for Computing Sample Variances* Compstat, Physica-Verlag

West D H D (1979) Updating mean and variance estimates: An improved method *Comm. ACM* **22** 532–555

5 Arguments

1: **m** – Integer *Input*

On entry: m, the length of the rolling window.

If **pn** $\neq 0$, **m** must be unchanged since the last call to nag_moving_average (g01wac).

Constraint: **m** ≥ 1.

2: **nb** – Integer *Input*

On entry: b, the number of observations in the current block of data. The size of the block of data supplied in **x** (and when **iwt** = Nag_WeightObs, **wt**) can vary; therefore **nb** can change between calls to nag_moving_average (g01wac).

Constraints:

nb ≥ 0;
if **rcomm** is **NULL**, **nb** $\geq$ **m**.

3: **x**[**nb**] – const double *Input*

On entry: the current block of observations, corresponding to x_i, for $i = k + 1, \ldots, k + b$, where k is the number of observations processed so far and b is the size of the current block of data.

4: **iwt** – Nag_Weightstype *Input*

On entry: the type of weighting to use.

iwt = Nag_NoWeights
No weights are used.

iwt = Nag_WeightObs
Each observation has its own weight.

iwt = Nag_WeightWindow
Each position in the window has its own weight.

iwt = Nag_WeightWindowPos
Each position in the window has a weight equal to its position number.

If **pn** $\neq 0$, **iwt** must be unchanged since the last call to nag_moving_average (g01wac).

Constraint: **iwt** = Nag_NoWeights, Nag_WeightObs, Nag_WeightWindow or Nag_WeightWindowPos.

5: **wt**[*dim*] – const double *Input*

Note: the dimension, *dim*, of the array **wt** must be at least

nb when **iwt** = Nag_WeightObs;
m when **iwt** = Nag_WeightWindow;
otherwise **wt** may be **NULL**.

On entry: the user-supplied weights.

If **iwt** = Nag_WeightObs, **wt**$[i - 1] = \nu_{i+k}$, for $i = 1, 2, \ldots, b$.

If **iwt** = Nag_WeightWindow, **wt**$[j - 1] = w_j$, for $j = 1, 2, \ldots, m$.

Otherwise, **wt** is not referenced and may be **NULL**.

Constraints:

if **iwt** = Nag_WeightObs, **wt**$[i - 1] \geq 0$, for $i = 1, 2, \ldots,$ **nb**;
if **iwt** = Nag_WeightWindow, **wt**$[0] \neq 0$ and $\sum_{j=1}^{m}$**wt**$[j - 1] > 0$;
if **iwt** = Nag_WeightWindow and **rsd** is not **NULL**, **wt**$[j - 1] \geq 0$, for $j = 1, 2, \ldots,$ **m**.

6: **pn** – Integer * *Input/Output*

On entry: k, the number of observations processed so far. On the first call to nag_moving_average (g01wac), or when starting to summarise a new dataset, **pn** must be set to 0.

If **pn** $\neq 0$, it must be the same value as returned by the last call to nag_moving_average (g01wac).

On exit: $k + b$, the updated number of observations processed so far.

Constraint: **pn** ≥ 0.

7: **rmean**[*dim*] – double *Output*

Note: the dimension, *dim*, of the array **rmean** must be at least max(0, **nb** + min(0, **pn** − **m** + 1)).

On exit: μ_l, the (weighted) moving averages, for $l = 1, 2, \ldots, b + \min(0, k - m + 1)$. Therefore, μ_l is the mean of the data in the window that ends on **x**$[l + m - \min(k, m - 1) - 2]$.

If, on entry, **pn** $\geq$ **m** − 1, i.e., at least one windows worth of data has been previously processed, then **rmean**$[l - 1]$ is the summary corresponding to the window that ends on **x**$[l - 1]$. On the other hand, if, on entry, **pn** = 0, i.e., no data has been previously processed, then **rmean**$[l - 1]$ is the summary corresponding to the window that ends on $\mathbf{x}[\mathbf{m}+l-2]$ (or, equivalently, starts on $\mathbf{x}[l-1]$).

8: **rsd**[*dim*] – double *Output*

Note: the dimension, *dim*, of the array **rsd** must be at least $\max(0, \mathbf{nb}+\min(0, \mathbf{pn}-\mathbf{m}+1))$.

Note: if standard deviations are not required then **rsd** must be **NULL**.

On exit: if **rsd** is not **NULL** then σ_l, the (weighted) standard deviation. The ordering of **rsd** is the same as the ordering of **rmean**.

9: **rcomm**[**2m + 20**] – double *Communication Array*

On entry: communication array, used to store information between calls to nag_moving_average (g01wac). If **rcomm** is **NULL** then **pn** must be set to zero and all the data must be supplied in one go.

10: **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 ⟨*value*⟩ had an illegal value.

NE_ILLEGAL_COMM

rcomm has been corrupted between calls.

NE_INT

On entry, **m** = ⟨*value*⟩.
Constraint: $\mathbf{m} \geq 1$.

On entry, **nb** = ⟨*value*⟩.
Constraint: $\mathbf{nb} \geq 0$.

On entry, **nb** = ⟨*value*⟩, **m** = ⟨*value*⟩.
Constraint: if **rcomm** is **NULL**, $\mathbf{nb} \geq \mathbf{m}$.

On entry, **pn** = ⟨*value*⟩.
Constraint: $\mathbf{pn} \geq 0$.

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_NEG_WEIGHT

On entry, **wt**[⟨*value*⟩] = ⟨*value*⟩.
Constraint: $\mathbf{wt}[i-1] \geq 0$.

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_PREV_CALL

if **pn** > 0, **iwt** must be unchanged since previous call.

On entry, **m** $= \langle value \rangle$.
On entry at previous call, **m** $= \langle value \rangle$.
Constraint: if **pn** > 0, **m** must be unchanged since previous call.

On entry, **pn** $= \langle value \rangle$.
On exit from previous call, **pn** $= \langle value \rangle$.
Constraint: if **pn** > 0, **pn** must be unchanged since previous call.

NE_SUM_WEIGHT

On entry, sum of weights supplied in **wt** is $\langle value \rangle$.
Constraint: if **iwt** = Nag_WeightWindow, the sum of the weights > 0.

NE_WEIGHT_ZERO

On entry, **wt**[0] $= \langle value \rangle$.
Constraint: if **iwt** = Nag_WeightWindow, **wt**[0] > 0.

NW_POTENTIAL_PROBLEM

On entry, at least one window had all zero weights.

On entry, unable to calculate at least one standard deviation due to the weights supplied.

7 Accuracy

Not applicable.

8 Parallelism and Performance

nag_moving_average (g01wac) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_moving_average (g01wac) makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

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

9 Further Comments

The more data that is supplied to nag_moving_average (g01wac) in one call, i.e., the larger **nb** is, the more efficient the function will be.

10 Example

This example calculates Spencer's 15-point moving average for the change in rate of the Earth's rotation between 1821 and 1850. The data is supplied in three chunks, the first consisting of five observations, the second 10 observations and the last 15 observations.

10.1 Program Text

```
/* nag_moving_average (g01wac) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 23, 2011.
 */
/* Pre-processor includes */
#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg01.h>

int main(void)
{
  /* Integer scalar and array declarations */
  Integer i, ierr, lrcomm, m, nb, offset, pn, nsummaries;
  Integer exit_status = 0;

  /* NAG structures and types */
  NagError fail;
  Nag_Weightstype iwt;
  Nag_Boolean want_sd;

  /* Double scalar and array declarations */
  double *rcomm = 0, *rmean = 0, *rsd = 0, *x = 0, *wt = 0;

  /* Character scalar and array declarations */
  char                 ciwt[40], cwant_sd[40];

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

  printf("nag_moving_average (g01wac) 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("%39s%"NAG_IFMT"%*[^\n] ",ciwt, _countof(ciwt),&m);
#else
  scanf("%39s%"NAG_IFMT"%*[^\n] ",ciwt,&m);
#endif
  iwt = (Nag_Weightstype) nag_enum_name_to_value(ciwt);

  /* Read in a flag indicating whether we want the standard deviations */
#ifdef _WIN32
  scanf_s("%39s%*[^\n] ",cwant_sd, _countof(cwant_sd));
#else
  scanf("%39s%*[^\n] ",cwant_sd);
#endif
  want_sd = (Nag_Boolean) nag_enum_name_to_value(cwant_sd);

  /* Initial handling of weights */
  if (iwt == Nag_WeightWindow) {
    /* Each observation in the rolling window has its own weight */
    if (!(wt = NAG_ALLOC(m, double)))
      {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
      }
    for (i = 0; i < m; i++)
      {
#ifdef _WIN32
```

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

  /* Allocate memory for the communication array */
  lrcomm = 2*m + 20;
  if (!(rcomm = NAG_ALLOC(lrcomm, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Print some titles */
  if (want_sd)
    {
      printf("                              Standard\n");
      printf("  Interval         Mean        Deviation\n");
      printf(" ---------------------------------------\n");
    }
  else
    {
      printf("  Interval         Mean  \n");
      printf(" ------------------------\n");
    }

  /* Loop over each block of data */
  for (pn = 0;;)
    {
      /* Read in the number of observations in this block */
#ifdef _WIN32
      ierr = scanf_s("%"NAG_IFMT, &nb);
#else
      ierr = scanf("%"NAG_IFMT, &nb);
#endif
      if (ierr == EOF || ierr < 1) break;
#ifdef _WIN32
      scanf_s("%*[^\n] ");
#else
      scanf("%*[^\n] ");
#endif

      /* Reallocate X to the required size */
      NAG_FREE(x);
      if (!(x = NAG_ALLOC(nb, double)))
        {
          printf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }

      /* Read in the data for this block */
      for (i = 0; i < nb; i++)
        {
#ifdef _WIN32
          scanf_s("%lf", &x[i]);
#else
          scanf("%lf", &x[i]);
#endif
        }
#ifdef _WIN32
      scanf_s("%*[^\n] ");
#else
```

```
      scanf("%*[^\n] ");
#endif

      if (iwt == Nag_WeightObs)
        {
          /* User supplied weights are present */

          /* Reallocate WT to the required size */
          NAG_FREE(wt);
          if (!(wt = NAG_ALLOC(nb, double)))
            {
              printf("Allocation failure\n");
              exit_status = -1;
              goto END;
            }

          /* Read in the weights for this block */
          for (i = 0; i < nb; i++)
            {
#ifdef _WIN32
              scanf_s("%lf", &wt[i]);
#else
              scanf("%lf", &wt[i]);
#endif
            }
#ifdef _WIN32
          scanf_s("%*[^\n] ");
#else
          scanf("%*[^\n] ");
#endif
        }

      /* Calculate the number of summaries we can produce */
      nsummaries = MAX(0,nb + MIN(0,pn-m+1));

      /* Reallocate the output arrays */
      NAG_FREE(rmean);
      if (!(rmean = NAG_ALLOC(nsummaries, double)))
        {
          printf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }
      if (want_sd)
        {
          NAG_FREE(rsd);
          if (!(rsd = NAG_ALLOC(nsummaries, double)))
            {
              printf("Allocation failure\n");
              exit_status = -1;
              goto END;
            }
        }

      /* nag_moving_average (g01wac):
         Calculate the moving average (and optionally the standard deviation)
         for this block of data
      */
      nag_moving_average(m, nb, x, iwt, wt, &pn, rmean, rsd, rcomm, &fail);
      if (fail.code != NE_NOERROR)
        {
          printf("Error from nag_moving_average (g01wac).\n%s\n",
                 fail.message);
          exit_status = -1;
          if (fail.code != NW_POTENTIAL_PROBLEM) goto END;
        }

      /* Number of results printed so far */
      offset = MAX(1,pn-nb-m+2);

      /* Display the results for this block of data */
      if (want_sd)
        {
          for (i = 0; i < nsummaries; i++)
            {
              printf(" [%3"NAG_IFMT",%3"NAG_IFMT"]     "
                     "%10.1f    %10.1f\n",
                     i + offset, i + m + offset - 1, rmean[i], rsd[i]);
            }
        }
      else
        {
          for (i = 0; i < nsummaries; i++)
            {
              printf(" [%3"NAG_IFMT",%3"NAG_IFMT"]    %10.1f\n",
                     i + offset, i + m + offset - 1, rmean[i]);
            }
        }
    }

  printf("\n");
  printf(" Total number of observations : %3"NAG_IFMT"\n", pn);
  printf(" Length of window             : %3"NAG_IFMT"\n", m);

 END:
  NAG_FREE(x);
  NAG_FREE(wt);
  NAG_FREE(rmean);
  NAG_FREE(rsd);
  NAG_FREE(rcomm);

  return(exit_status);
}
```

10.2 Program Data

```
nag_moving_average (g01wac) Example Program Data
Nag_WeightWindow 15                          :: iwt,m
Nag_FALSE                                    :: If Nag_TRUE sd's are calculated
-3.0 -6.0 -5.0 3.0 21.0 46.0 67.0
74.0 67.0 46.0 21.0 3.0 -5.0 -6.0 -3.0   :: wt
5                                            :: nb
 -2170.0 -1770.0 -1660.0 -1360.0 -1100.0 :: End of x for first block
10                                           :: nb
  -950.0  -640.0  -370.0  -140.0  -250.0
  -510.0  -620.0  -730.0  -880.0 -1130.0 :: End of x for second block
15                                           :: nb
 -1200.0  -830.0  -330.0  -190.0   210.0
   170.0   440.0   440.0   780.0   880.0
  1220.0  1260.0  1140.0   850.0   640.0 :: End of x for third block
```

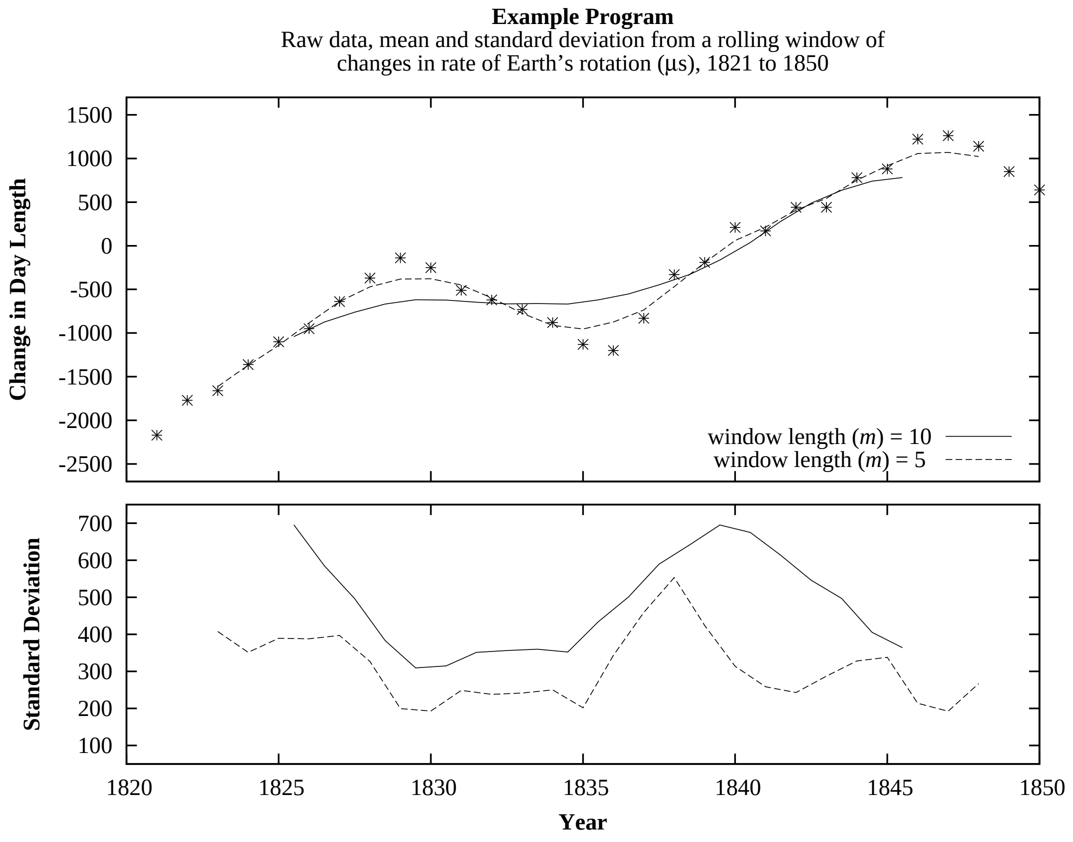
10.3 Program Results

```
nag_moving_average (g01wac) Example Program Results

  Interval         Mean
 ------------------------
 [  1, 15]       -427.6
 [  2, 16]       -332.5
 [  3, 17]       -337.1
 [  4, 18]       -438.2
 [  5, 19]       -604.4
 [  6, 20]       -789.4
 [  7, 21]       -935.4
 [  8, 22]       -990.6
 [  9, 23]       -927.1
 [ 10, 24]       -752.1
 [ 11, 25]       -501.3
 [ 12, 26]       -227.2
 [ 13, 27]         23.2
 [ 14, 28]        236.2
 [ 15, 29]        422.4
 [ 16, 30]        604.2

 Total number of observations :  30
 Length of window             :  15
```

This example plot shows the smoothing effect of using different length rolling windows on the mean and standard deviation. Two different window lengths, $m = 5$ and 10, are used to produce the unweighted rolling mean and standard deviations for the change in rate of the Earth's rotation between 1821 and 1850. The values of the rolling mean and standard deviations are plotted at the centre points of their respective windows.


Example Program
Raw data, mean and standard deviation from a rolling window of
changes in rate of Earth's rotation (µs), 1821 to 1850
Change in Day Length
Standard Deviation
window length (m) = 10
window length (m) = 5
Year