NAG Library Function Document

nag_approx_quantiles_fixed (g01anc)

1 Purpose

nag_approx_quantiles_fixed (g01anc) finds approximate quantiles from a data stream of known size using an out-of-core algorithm.

2 Specification

```
#include <nag.h>
#include <nagg01.h>
void nag_approx_quantiles_fixed (Integer *ind, Integer n, const double rv[],
     Integer nb, double eps, Integer *np, const double q[], double qv[],
     Integer nq, double rcomm[], Integer lrcomm, Integer icomm[],
     Integer licomm, NagError *fail)
```

3 Description

A quantile is a value which divides a frequency distribution such that there is a given proportion of data values below the quantile. For example, the median of a dataset is the 0.5 quantile because half the values are less than or equal to it.

nag_approx_quantiles_fixed (g01anc) uses a slightly modified version of an algorithm described in a paper by Zhang and Wang (2007) to determine ϵ-approximate quantiles of a data stream of n real values, where n is known. Given any quantile $q \in [0.0, 1.0]$, an ϵ-approximate quantile is defined as an element in the data stream whose rank falls within $[(q-\epsilon)n, (q+\epsilon)n]$. In case of more than one ϵ-approximate quantile being available, the one closest to qn is returned.

4 References

Zhang Q and Wang W (2007) A fast algorithm for approximate quantiles in high speed data streams *Proceedings of the 19th International Conference on Scientific and Statistical Database Management* IEEE Computer Society 29

5 Arguments

1: **ind** – Integer * *Input/Output*

On entry: indicates the action required in the current call to nag_approx_quantiles_fixed (g01anc).

ind $= 0$
Return the required length of **rcomm** and **icomm** in **icomm**$[0]$ and **icomm**$[1]$ respectively. **n** and **eps** must be set and **licomm** must be at least 2.

ind $= 1$
Initialise the communication arrays and process the first **nb** values from the data stream as supplied in **rv**.

ind $= 2$
Process the next block of **nb** values from the data stream. The calling program must update **rv** and (if required) **nb**, and re-enter nag_approx_quantiles_fixed (g01anc) with all other parameters unchanged.

ind $= 3$
Calculate the **nq** ϵ-approximate quantiles specified in **q**. The calling program must set **q** and **nq** and re-enter nag_approx_quantiles_fixed (g01anc) with all other parameters unchanged. This option can be chosen only when **np** $\geq \lceil \exp(1.0)/\textbf{eps} \rceil$.

On exit: indicates output from a successful call.

ind $= 1$
Lengths of **rcomm** and **icomm** have been returned in **icomm**$[0]$ and **icomm**$[1]$ respectively.

ind $= 2$
nag_approx_quantiles_fixed (g01anc) has processed **np** data points and expects to be called again with additional data (i.e., **np** $<$ **n**).

ind $= 3$
nag_approx_quantiles_fixed (g01anc) has returned the requested ϵ-approximate quantiles in **qv**. These quantiles are based on **np** data points.

ind $= 4$
Routine has processed all **n** data points (i.e., **np** $=$ **n**).

Constraint: on entry **ind** $= 0$, 1, 2 or 3.

2: **n** – Integer *Input*

On entry: n, the total number of values in the data stream.

Constraint: **n** > 0.

3: **rv**$[dim]$ – const double *Input*

Note: the dimension, *dim*, of the array **rv** must be at least **nb** when **ind** $= 1$ or 2.

On entry: if **ind** $= 1$ or 2, the vector containing the current block of data, otherwise **rv** is not referenced.

4: **nb** – Integer *Input*

On entry: if **ind** $= 1$ or 2, the size of the current block of data. The size of blocks of data in array **rv** can vary; therefore **nb** can change between calls to nag_approx_quantiles_fixed (g01anc).

Constraint: if **ind** $= 1$ or 2, **nb** > 0.

5: **eps** – double *Input*

On entry: approximation factor ϵ.

Constraint: **eps** $\geq \exp(1.0)/$**n** and **eps** ≤ 1.0.

6: **np** – Integer * *Output*

On exit: the number of elements processed so far.

7: **q**$[dim]$ – const double *Input*

Note: the dimension, *dim*, of the array **q** must be at least **nq** when **ind** $= 3$.

On entry: if **ind** $= 3$, the quantiles to be calculated, otherwise **q** is not referenced. Note that **q**$[i] = 0.0$, corresponds to the minimum value and **q**$[i] = 1.0$ to the maximum value.

Constraint: if **ind** $= 3$, $0.0 \leq$ **q**$[i-1] \leq 1.0$, for $i = 1, 2, \ldots,$ **nq**.

8: **qv**$[dim]$ – double *Output*

Note: the dimension, *dim*, of the array **qv** must be at least **nq** when **ind** $= 3$.

On exit: if **ind** $= 3$, **qv**$[i]$ contains the ϵ-approximate quantiles specified by the value provided in **q**$[i]$.

9: **nq** – Integer *Input*

On entry: if **ind** $= 3$, the number of quantiles requested, otherwise **nq** is not referenced.

Constraint: if **ind** $= 3$, **nq** > 0.

10: **rcomm**[**lrcomm**] – double *Communication Array*

11: **lrcomm** – Integer *Input*

On entry: the dimension of the array **rcomm**.

Constraint: if **ind** $\neq 0$, **lrcomm** must be at least equal to the value returned in **icomm**[0] by a call to nag_approx_quantiles_fixed (g01anc) with **ind** $= 0$. This will not be more than $x + 2 \times \min(x, \lceil x/2.0 \rceil + 1) \times \log_2(\mathbf{n}/x + 1.0) + 1$, where $x = \max(1, \lfloor \log(\mathbf{eps} \times \mathbf{n})/\mathbf{eps} \rfloor)$.

12: **icomm**[**licomm**] – Integer *Communication Array*

13: **licomm** – Integer *Input*

On entry: the dimension of the array **icomm**.

Constraints:

if **ind** $= 0$, **licomm** ≥ 2;
otherwise **licomm** must be at least equal to the value returned in **icomm**[1] by a call to nag_approx_quantiles_fixed (g01anc) with **ind** $= 0$. This will not be more than $2 \times (x + 2 \times \min(x, \lceil x/2.0 \rceil + 1) \times y) + y + 6$, where $x = \max(1, \lfloor \log(\mathbf{eps} \times \mathbf{n})/\mathbf{eps} \rfloor)$ and $y = \log_2(\mathbf{n}/x + 1.0) + 1$.

14: **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_ARRAY_SIZE

On entry, **licomm** is too small: **licomm** $= \langle value \rangle$.

On entry, **lrcomm** is too small: **lrcomm** $= \langle value \rangle$.

NE_BAD_PARAM

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

NE_INT

On entry, **ind** $= 1$ or 2 and **nb** $= \langle value \rangle$.
Constraint: if **ind** $= 1$ or 2 then **nb** > 0.

On entry, **ind** $= 3$ and **nq** $= \langle value \rangle$.
Constraint: if **ind** $= 3$ then **nq** > 0.

On entry, **ind** $= \langle value \rangle$.
Constraint: **ind** $= 0$, 1, 2 or 3.

On entry, **n** $= \langle value \rangle$.
Constraint: **n** > 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 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_Q_OUT_OF_RANGE

On entry, **ind** $= 3$ and $\mathbf{q}[\langle value \rangle] = \langle value \rangle$.
Constraint: if **ind** $= 3$ then $0.0 \leq \mathbf{q}[i] \leq 1.0$ for all i.

NE_REAL

On entry, **eps** $= \langle value \rangle$.
Constraint: $\exp(1.0)/\mathbf{n} \leq \mathbf{eps} \leq 1.0$.

NE_TOO_SMALL

Number of data elements streamed, $\langle value \rangle$ is not sufficient for a quantile query when **eps** $= \langle value \rangle$.
Supply more data or reprocess the data with a higher **eps** value.

7 Accuracy

Not applicable.

8 Parallelism and Performance

nag_approx_quantiles_fixed (g01anc) 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 function. Please also consult the Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

The average time taken by nag_approx_quantiles_fixed (g01anc) is $\mathbf{n}\log(1/\epsilon\log(\epsilon\mathbf{n}))$.

10 Example

This example calculates ϵ-approximate quantile for $q = 0.25$, 0.5 and 1.0 for a data stream of 60 values. The stream is read in four blocks of varying size.

10.1 Program Text

```
/* nag_approx_quantiles_fixed (g01anc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#include <nag.h>
```

```
#include <stdio.h>
#include <nag_stdlib.h>
#include <nagg01.h>

int main(void)
{
  /* Scalars */
  Integer exit_status = 0;
  Integer i, ind, j, licomm, lrcomm, n, nb, np, nq, nrv;
  double eps;
  /* Arrays */
  double *q = 0, *qv = 0, *rcomm = 0, *rv = 0, trcomm[1], trv[1];
  Integer *icomm = 0, ticomm[2];
  /* Nag Types */
  NagError fail;

  INIT_FAIL(fail);

  printf("nag_approx_quantiles_fixed (g01anc) Example Program Results\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
#ifdef _WIN32
  scanf_s("%lf%*[^\n] ", &eps);
#else
  scanf("%lf%*[^\n] ", &eps);
#endif
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%*[^\n] ", &nq);
#else
  scanf("%" NAG_IFMT "%*[^\n] ", &nq);
#endif

  if (!(qv = NAG_ALLOC(nq, double)) || !(q = NAG_ALLOC(nq, double)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  /* Read in the quantiles that are required */
  for (i = 0; i < nq; ++i)
#ifdef _WIN32
    scanf_s("%lf", &q[i]);
#else
    scanf("%lf", &q[i]);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  /* Call the routine for the first time to obtain lrcomm and licomm */
  nb = lrcomm = 1;
  licomm = 2;
  ind = 0;
  nag_approx_quantiles_fixed(&ind, n, trv, nb, eps, &np, q, qv,
                             nq, trcomm, lrcomm, ticomm, licomm, &fail);
  if (fail.code != NE_NOERROR) {
```

```
    printf("Error from nag_approx_quantiles_fixed (g01anc).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }

  /* Use calculated array sizes to allocate the communication arrays */
  lrcomm = ticomm[0];
  licomm = ticomm[1];
  if (!(rcomm = NAG_ALLOC(lrcomm, double)) ||
      !(icomm = NAG_ALLOC(licomm, Integer)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  /* Read in the number of blocks of data */
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%*[^\n] ", &nrv);
#else
  scanf("%" NAG_IFMT "%*[^\n] ", &nrv);
#endif

  /* Loop over each block of data */
  for (i = 0; i < nrv; ++i) {

    /* Read in the size of the i'th block of data */
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "%*[^\n] ", &nb);
#else
    scanf("%" NAG_IFMT "%*[^\n] ", &nb);
#endif

    /* Reallocate rv */
    NAG_FREE(rv);
    if (!(rv = NAG_ALLOC(nb, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

    /* Read in the data for the i'th block */
    for (j = 0; j < nb; ++j)
#ifdef _WIN32
      scanf_s("%lf", &rv[j]);
#else
      scanf("%lf", &rv[j]);
#endif
#ifdef _WIN32
    scanf_s("%*[^\n] ");
#else
    scanf("%*[^\n] ");
#endif

    /* Update the summaries based on the i'th block of data */
    nag_approx_quantiles_fixed(&ind, n, rv, nb, eps, &np, q, qv, nq,
                               rcomm, lrcomm, icomm, licomm, &fail);
    if (fail.code != NE_NOERROR) {
      printf("Error from nag_approx_quantiles_fixed (g01anc).\n%s\n",
             fail.message);
      exit_status = 1;
      goto END;
    }
    if (ind == 4)
      break;
  }

  /* Call the routine again to calculate quantiles specified in vector q */
  ind = 3;
```

```
  nag_approx_quantiles_fixed(&ind, n, rv, nb, eps, &np, q, qv, nq, rcomm,
                             lrcomm, icomm, licomm, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_approx_quantiles_fixed (g01anc).\n%s\n",
            fail.message);
    exit_status = 1;
    goto END;
  }

  /* Print the results */
  printf("\n    Input data:\n");
  printf("    %" NAG_IFMT " observations\n", n);
  printf("    eps = %5.2f\n", eps);
  printf("    Quantile   Result\n\n");
  for (i = 0; i < nq; ++i) {
    printf("  %7.2f    %7.2f\n", q[i], qv[i]);
  }

END:
  NAG_FREE(rv);
  NAG_FREE(q);
  NAG_FREE(qv);
  NAG_FREE(rcomm);
  NAG_FREE(icomm);

  return exit_status;
}
```

10.2 Program Data

```
nag_approx_quantiles_fixed (g01anc) Example Program Data
60                                          :: n
0.2                                         :: eps
3                                           :: nq
0.25 0.5 1.0                                :: qv
4                                           :: number of blocks of data
16                                          :: nb (1st of block data)
34.01  57.95  44.88  22.04  28.84   4.43
 0.32  20.82  20.53  13.08  7.99   54.03
23.21  26.73  39.72   0.97                 :: end of rv (1st block of data)
24                                          :: nb (2nd of block data)
39.05  38.78  19.38  51.34  24.08  12.41
58.11  35.90  40.38  27.41  19.80   6.02
45.33  36.34  43.14  53.84  39.49   9.04
36.74  58.72  59.95  15.41  33.05  39.54   :: end of rv (2nd block of data)
8                                           :: nb (3rd block of data)
33.24  58.67  54.12  39.48  43.73  24.15
55.72   8.87                               :: end of rv (3rd block of data)
12                                          :: nb (4th block of data)
40.47  46.18  20.36   6.95  36.86  49.24
56.83  43.87  29.86  22.49  25.29  33.17   :: end of rv (4th block of data)
```

10.3 Program Results

```
nag_approx_quantiles_fixed (g01anc) Example Program Results

    Input data:
    60 observations
    eps =  0.20
    Quantile   Result

     0.25      22.49
     0.50      36.86
     1.00      59.95
```