NAG Library Function Document

nag_approx_quantiles_arbitrary (g01apc)

1 Purpose

nag_approx_quantiles_arbitrary (g01apc) finds approximate quantiles from a large arbitrary-sized data stream using an out-of-core algorithm.

2 Specification

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

```
void nag_approx_quantiles_arbitrary (Integer *ind, 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_arbitrary (g01apc) uses a slightly modified version of an algorithm described in a paper by Zhang and Wang (2007) to determine ϵ-approximate quantiles of a large arbitrary-sized data stream of real values, where ϵ is a user-defined approximation factor. Let m denote the number of data elements processed so far then, 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 $[(q - \epsilon)m, (q + \epsilon)m]$. In case of more than one ϵ-approximate quantile being available, the one closest to qm is used.

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 initial entry: must be set to 0.

On entry: indicates the action required in the current call to nag_approx_quantiles_arbitrary (g01apc).

ind $= 0$
Initialize the communication arrays and attempt to process the first **nb** values from the data stream. **eps**, **rv** and **nb** must be set and **licomm** must be at least 10.

ind $= 1$
Attempt to 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_arbitrary (g01apc) with all other parameters unchanged.

ind $= 2$
Continue calculation following the reallocation of either or both of the communication arrays **rcomm** and **icomm**.

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_arbitrary (g01apc) with all other parameters unchanged. This option can be chosen only when **np** $\geq \lceil \exp(1.0)/$**eps**$\rceil$.

On exit: indicates output from the call.

ind $= 1$

nag_approx_quantiles_arbitrary (g01apc) has processed **np** data points and expects to be called again with additional data.

ind $= 2$

Either one or more of the communication arrays **rcomm** and **icomm** is too small. The new minimum lengths of **rcomm** and **icomm** have been returned in **icomm**[0] and **icomm**[1] respectively. If the new minimum length is greater than the current length then the corresponding communication array needs to be reallocated, its contents preserved and nag_approx_quantiles_arbitrary (g01apc) called again with all other parameters unchanged.

If there is more data to be processed, it is recommended that **lrcomm** and **licomm** are made significantly bigger than the minimum to limit the number of reallocations.

ind $= 3$

nag_approx_quantiles_arbitrary (g01apc) has returned the requested ϵ-approximate quantiles in **qv**. These quantiles are based on **np** data points.

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

2: **rv**[*dim*] – const double *Input*

Note: the dimension, *dim*, of the array **rv** must be at least **nb** when **ind** $= 0$, 1 or 2.

On entry: if **ind** $= 0$, 1 or 2, the vector containing the current block of data, otherwise **rv** is not referenced.

3: **nb** – Integer *Input*

On entry: if **ind** $= 0$, 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_arbitrary (g01apc).

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

4: **eps** – double *Input*

On entry: approximation factor ϵ.

Constraint: **eps** > 0.0 and **eps** ≤ 1.0.

5: **np** – Integer * *Output*

On exit: m, the number of elements processed so far.

6: **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**.

7: **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**$[i]$.

8: **nq** – Integer *Input*

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

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

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

On entry: if **ind** $= 1$ or 2 then the first l elements of **rcomm** as supplied to nag_approx_quantiles_arbitrary (g01apc) must be identical to the first l elements of **rcomm** returned from the last call to nag_approx_quantiles_arbitrary (g01apc), where l is the value of **lrcomm** used in the last call. In other words, the contents of **rcomm** must not be altered between calls to this function. If **rcomm** needs to be reallocated then its contents must be preserved. If **ind** $= 0$ then **rcomm** need not be set.

On exit: **rcomm** holds information required by subsequent calls to nag_approx_quantiles_arbitrary (g01apc)

10: **lrcomm** – Integer *Input*

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

Constraints:

if **ind** $= 0$, **lrcomm** ≥ 1;
otherwise **lrcomm** $\geq$ **icomm**$[0]$.

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

On entry: if **ind** $= 1$ or 2 then the first l elements of **icomm** as supplied to nag_approx_quantiles_arbitrary (g01apc) must be identical to the first l elements of **icomm** returned from the last call to nag_approx_quantiles_arbitrary (g01apc), where l is the value of **licomm** used in the last call. In other words, the contents of **icomm** must not be altered between calls to this function. If **icomm** needs to be reallocated then its contents must be preserved. If **ind** $= 0$ then **icomm** need not be set.

On exit: **icomm**$[0]$ holds the minimum required length for **rcomm** and **icomm**$[1]$ holds the minimum required length for **icomm**. The remaining elements of **icomm** are used for communication between subsequent calls to nag_approx_quantiles_arbitrary (g01apc).

12: **licomm** – Integer *Input*

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

Constraints:

if **ind** $= 0$, **licomm** ≥ 10;
otherwise **licomm** $\geq$ **icomm**$[1]$.

13: **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.

NE_ARRAY_SIZE

On entry, **licomm** $= \langle value \rangle$.
Constraint: **licomm** ≥ 10.

On entry, **lrcomm** $= \langle value \rangle$.
Constraint: **lrcomm** ≥ 1.

NE_BAD_PARAM

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

NE_ILLEGAL_COMM

The contents of **icomm** have been altered between calls to this function.

The contents of **rcomm** have been altered between calls to this function.

NE_INT

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

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.

NE_Q_OUT_OF_RANGE

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

NE_REAL

On entry, **eps** $= \langle value \rangle$.
Constraint: $0.0 <$ **eps** ≤ 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_arbitrary (g01apc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

Please consult the Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

The average time taken by nag_approx_quantiles_arbitrary (g01apc) scales as $\mathbf{np}\log(1/\epsilon\log(\epsilon\mathbf{np}))$.

It is not possible to determine in advance the final size of the communication arrays **rcomm** and **icomm** without knowing the size of the dataset. However, if a rough size (n) is known, the speed of the computation can be increased if the sizes of the communication arrays are not smaller than

$$\begin{aligned}\textbf{lrcomm} &= (\log_2{(n \times \textbf{eps} + 1.0)} - 2) \times \lceil 1.0/\textbf{eps} \rceil + 1 + x + 2 \times \min(x, \lceil x/2.0 \rceil + 1) \times y + 1 \\ \textbf{licomm} &= (\log_2{(n \times \textbf{eps} + 1.0)} - 2) \times (2 \times (\lceil 1.0/\textbf{eps} \rceil + 1) + 1) + \\ & \quad 2 \times (x + 2 \times \min(x, \lceil x/2.0 \rceil + 1) \times y) + y + 11\end{aligned}$$

where

$$\begin{aligned}x &= \max(1, \lfloor \log{(\textbf{eps} \times n)}/\textbf{eps} \rfloor) \\ y &= \log_2{(n/x + 1.0)} + 1.\end{aligned}$$

10 Example

This example computes a list of ϵ-approximate quantiles. The data is processed in blocks of 20 observations at a time to simulate a situation in which the data is made available in a piecemeal fashion.

10.1 Program Text

```
/* nag_approx_quantiles_arbitrary (g01apc) Example Program.
 *
 * Copyright 2011 Numerical Algorithms Group.
 *
 * Mark 23, 2011.
 */

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

int main(void)
{
  /* Scalars */
  Integer     exit_status = 0;
  Integer     i, ind, licomm, lrcomm, nb, np, nq, ierr;
  double      eps;
  Nag_Boolean repeat;
  /* Arrays */
  double      *q = 0, *qv = 0, *rcomm = 0, *trcomm = 0, *rv = 0;
  Integer     *icomm = 0, *ticomm = 0;
  /* Nag Types */
  NagError    fail;

  INIT_FAIL(fail);

  printf("nag_approx_quantiles_arbitrary (g01apc) Example Program Results\n");

  /* Skip heading in data file */
  scanf("%*[^\n]");

  /* Read in the problem size */
  scanf("%lf%*[^\n] ", &eps);
  scanf("%ld%*[^\n] ", &nq);

  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)
    scanf("%lf", &q[i]);
  scanf("%*[^\n]");

  /* Going to be reading in the data in blocks of size 20 */
  nb = 20;
  /* Make an initial allocation to the communication arrays */
  lrcomm = 100;
  licomm = 400;
  if (!(rcomm = NAG_ALLOC(lrcomm, double)) ||
      !(icomm = NAG_ALLOC(licomm, Integer)) ||
      !(rv = NAG_ALLOC(nb, double))) {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  /* Start looping across the data */
  ind = 0;
  repeat = Nag_TRUE;

  while(repeat) {
    /* Read in the blocks of data, each of size nb */
    for (i = 0; i < nb; ++i) {
      ierr = scanf("%lf", &rv[i]);
      if (ierr == EOF || ierr == 0) {
        /* We've read in the last block of data */
        repeat = Nag_FALSE;

        /* Set nb to the size of the last block of data */
        nb = i;
        break;
      }
    }

    /* No data read in, so stop */
    if (nb == 0) break;

    do {
      /* Update the summaries based on the current block of data */
      nag_approx_quantiles_arbitrary(&ind, rv, nb, eps, &np, q, qv,
                                     nq, rcomm, lrcomm, icomm, licomm, &fail);
      if (fail.code != NE_NOERROR) {
        printf(
               "Error from nag_approx_quantiles_arbitrary (g01apc).\n%s\n",
               fail.message);
        exit_status = 1;
        goto END;
      }

      if (ind==2){
        /* At least one of the communication arrays are too small */

        if (lrcomm < icomm[0]) {
          /* Need to make rcomm larger */

          /* Allocate memory a real communication array of the new
             size (held in icomm[0]) */
          if (!(trcomm = NAG_ALLOC(icomm[0], double))) {
            printf("Allocation failure\n");
            exit_status = -1;
            goto END;
          }

          /* Copy the old information into the new array */
          for (i = 0; i < lrcomm; ++i)
            trcomm[i] = rcomm[i];

          /* Set lrcomm to the new size */
          lrcomm = icomm[0];

          /* Free up the old communication array */
          NAG_FREE(rcomm);

          /* Set rcomm to the new array */
          rcomm = trcomm;
        }

        if (licomm < icomm[1]) {
          /* Need to make icomm larger */

          /* Allocate memory to an integer communication array of the new
             size (held in icomm[1]) */
          if (!(ticomm = NAG_ALLOC(icomm[1], Integer))) {
            printf("Allocation failure\n");
            exit_status = -1;
            goto END;
          }

          /* Copy the old information into the new array */
          for (i = 0; i < licomm; ++i)
            ticomm[i] = icomm[i];

          /* Set lrcomm to the new size */
          licomm = icomm[1];

          /* Free up the old communication array */
          NAG_FREE(icomm);

          /* Set icomm to the new array */
          icomm = ticomm;
        }
      }

      /* If ind == 2 then we want to call the routine again, with the same
         block of data */
    } while (ind==2);
  }

  /* Call the routine again to calculate quantiles specified in vector q */
  ind = 3;
  nag_approx_quantiles_arbitrary(&ind, rv, nb, eps, &np, q, qv,
                                 nq, rcomm, lrcomm, icomm, licomm, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_approx_quantiles_arbitrary (g01apc).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }

  /* Print the results */
  printf("\n    Input data:\n");
  printf("    %ld observations\n", np);
  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_arbitrary (g01apc) Example Program Data
0.2                                        :: eps
3                                          :: nq
0.25 0.5 1.0                               :: q
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 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
33.24 58.67 54.12 39.48 43.73
24.15 55.72  8.87 40.47 46.18
20.36  6.95 36.86 49.24 56.83
43.87 29.86 22.49 25.29 33.17
```

10.3 Program Results

```
nag_approx_quantiles_arbitrary (g01apc) Example Program Results

    Input data:
    60 observations
    eps =  0.20
    Quantile   Result

     0.25      22.49
     0.50      39.54
     1.00      59.95
```