NAG Library Function Document

nag_sum_fast_gauss (c06sac)

1 Purpose

nag_sum_fast_gauss (c06sac) calculates the multidimensional fast Gauss transform.

2 Specification

```
#include <nag.h>
#include <nagc06.h>
```

```
void nag_sum_fast_gauss (Integer d, const double srcs[], Integer n,
     const double trgs[], Integer m, const double q[], Integer *p1,
     Integer *p2, Integer *k, const double hin[], Integer lhin, double tol,
     double v[], double term[], NagError *fail)
```

3 Description

nag_sum_fast_gauss (c06sac) calculates the d-dimensional fast Gauss transform (FGT), $\hat{G}(y)$, that approximates the discrete Gauss transform (DGT), $G(y)$, evaluated at a set of target points y_j, for $j = 1, 2, \dots, m \in R^d$. The DGT is defined as:

$$G(y_j) = \sum_{i=1}^{n} q_i e^{-\|y_j - x_i\|_2^2/h_i^2}, \quad j = 1, \dots, m$$

where x_i, for $i = 1, 2, \dots, n \in R^d$, are the Gaussian source points, q_i, for $i = 1, 2, \dots, n \in R^+$, are the source weights and h_i, for $i = 1, 2, \dots, n \in R^+$, are the source standard deviations (alternatively source scales or source bandwidths).

This function implements the improved FGT algorithm presented in Raykar and Duraiswami (2005). The algorithm clusters the sources into k distinct clusters and then computes two Taylor series approximations per cluster with p_1 and p_2 terms respectively. You must provide p_1, p_2 and k when calling the function. See Section 7 below for a further discussion on accuracy when choosing their values.

The input array **hin** of this function is designed to allow maximum flexibility in the supply of the standard deviation arguments by reusing, in a cyclic manner, elements of the array when it is less than n elements long. For example, if all Gaussian sources have the same standard deviation then it is only necessary to set **lhin** to 1 and to provide the value of the standard deviation in **hin**(1); the function will then automatically expand **hin** to be of length n. For further details please see Section 2.6 in the g01 Chapter Introduction.

4 References

Greengard L and Strain J (1991) The Fast Gauss Transform *SIAM J. Sci. Statist. Comput.* **12(1)** 79–94

Raykar V C and Duraiswami R (2005) Improved Fast Gauss Transform With Variable Source Scales *University of Maryland Technical Report CS-TR-4727/UMIACS-TR-2005-34*

5 Arguments

1: **d** – Integer *Input*

On entry: d, the number of dimensions.

Constraint: $\mathbf{d} > 0$.

2: **srcs**[**d** $\times$ **n**] – const double *Input*

Note: the (i,j)th element of the matrix is stored in **srcs**[$(j-1) \times$ **d** $+ i - 1$].

On entry: x, the locations of the Gaussian sources.

3: **n** – Integer *Input*

On entry: n, the number of Gaussian sources.

Constraint: **n** > 0.

4: **trgs**[**d** $\times$ **m**] – const double *Input*

Note: the (i,j)th element of the matrix is stored in **trgs**[$(j-1) \times$ **d** $+ i - 1$].

On entry: y, the locations of the target points at which the FGT will be evaluated.

5: **m** – Integer *Input*

On entry: m, the number of target points.

Constraint: **m** > 0.

6: **q**[**n**] – const double *Input*

On entry: q, the weights of the Gaussian sources.

7: **p1** – Integer * *Input/Output*

On entry: p_1, the number of terms of the first Taylor series to be evaluated.

On exit: **p1** is unchanged.

Constraint: **p1** > 0.

8: **p2** – Integer * *Input/Output*

On entry: p_2, the number of terms of the second Taylor series to be evaluated.

On exit: **p2** is unchanged.

Constraint: **p2** > 0.

9: **k** – Integer * *Input/Output*

On entry: k, the number of clusters into which the source points will be aggregated.

On exit: **k** is unchanged.

Constraint: $1 \leq$ **k** $\leq$ **n**.

10: **hin**[**lhin**] – const double *Input*

On entry: h, the standard deviations of the Gaussian sources. If **lhin** $<$ **n**, the array will be expanded automatically by repeating **hin** until it is of length **n**. See Section 2.6 in the g01 Chapter Introduction for further information.

Constraint: **hin**[i] > 0.0, for $i = 0, 1, \ldots,$ **lhin** $- 1$.

11: **lhin** – Integer *Input*

On entry: the length of the array **hin**.

Constraint: $1 \leq$ **lhin** $\leq$ **n**.

12: **tol** – double *Input*

On entry: ϵ, the desired accuracy of the FGT approximation of the DGT. Determines the radius of the source clusters: the contribution of a source point to the FGT approximation at a target point is disregarded if the source is outside the corresponding cluster radius.

Constraint: **tol** > 0.0.

13: **v**[**m**] – double *Output*

On exit: $\hat{G}(y)$, the value of the FGT evaluated at y.

14: **term**[**m**] – double *Output*

On exit: **term**[$j-1$] contains the absolute value of the final Taylor series term that is largest, relative to the size of the sum of the corresponding series, across all clusters that contribute to the FGT at target point **v**[$j-1$].

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

The NAG error argument (see Section 3.7 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, **lhin** = ⟨*value*⟩.
Constraint: $1 \leq$ **lhin** $\leq$ **n**.

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_INT

On entry, **d** = ⟨*value*⟩.
Constraint: **d** > 0.

On entry, **m** = ⟨*value*⟩.
Constraint: **m** > 0.

On entry, **n** = ⟨*value*⟩.
Constraint: **n** > 0.

On entry, **p1** = ⟨*value*⟩.
Constraint: **p1** > 0.

On entry, **p2** = ⟨*value*⟩.
Constraint: **p2** > 0.

NE_INT_2

On entry, **k** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: $1 \leq$ **k** $\leq$ **n**.

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.
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, **tol** $= \langle value \rangle$.
Constraint: **tol** > 0.0.

NE_REAL_ARRAY_INPUT

On entry, **hin**$[\langle value \rangle] = \langle value \rangle$.
Constraint: **hin**$[i] > 0.0$, for $i = 0, 1, \ldots,$ **lhin** $- 1$.

NW_TOO_FEW_TERMS

On exit, **p1** $= \langle value \rangle$, **p2** $= \langle value \rangle$ and **k** $= \langle value \rangle$.
p1, **p2** or **k** may have been too small to calculate **v**[**m** $- 1$] to the required accuracy **tol**.

7 Accuracy

The function does not currently implement the procedure described in Raykar and Duraiswami (2005) for automatically determining values for **p1**, **p2** and **k**. Non-zero values must therefore be provided for these parameters when calling the function.

For a given set of source and target points and a specified tolerance, there is an interaction between the number of clusters, **k**, and the number of Taylor series terms, **p1** and **p2**: if the sources are clustered together in fewer clusters (small **k**) then more terms will be needed in each cluster's Taylor series (large **p1** and **p2**) to capture the effect of the source points further from the cluster centres. Increasing the number of clusters reduces their individual radii and requires fewer terms in their Taylor series, but increases the number of Taylor series that must be evaluated overall.

If the source and target points are uniformly distributed in a unit hypercube, Raykar and Duraiswami (2005) advise users to select **k** $\sim \left\lceil (h_{\max} + h_{\min}/2)^{-d} \right\rceil$. If the points are not uniformly distributed then more clusters than this will be needed to calculate the FGT to within the specified **tol** without requiring prohibitively large values for **p1** and **p2**.

There is less guidance available for selecting good values for **p1** and **p2**. As the number of Taylor series terms is a major factor on the computation time taken by this function, you are advised to initially try a small number, e.g. 20 or so, and then tune **p1** and **p2** up or down based on the values returned. Note that **p1** and **p2** are not required to have identical values.

To aid the selection of values for **p1**, **p2** and **k**, the function returns in **term**$[j - 1]$ the absolute value of the final Taylor series term that is largest, relative to the size of the sum of the corresponding series, across all clusters that contribute to the FGT at target point j. If this value is larger than the requested **tol**, the function will additionally return a non-zero **fail** value and you are advised to re-run the function with larger **p1**, **p2** or **k**.

8 Parallelism and Performance

nag_sum_fast_gauss (c06sac) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_sum_fast_gauss (c06sac) 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 time complexity of the algorithm implemented by this function is $O(M+N)$, versus the $O(MN)$ time complexity of evaluating the DGT directly.

10 Example

In this example values for x, y, p_1, p_2, k and ϵ are read in, $\hat{G}(y)$ calculated and the results displayed.

10.1 Program Text

```
/* nag_sum_fast_gauss (c06sac) Example Program.
 *
 * Copyright 2017 Numerical Algorithms Group.
 *
 * Mark 26.1, 2017.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagc06.h>

int main(void) {

  /* Scalars */
  Integer  d, n, m, p1, p2, k, i, j;
  Integer  exit_status = 0;
  double   tol;

  /* Arrays */
  double   *srcs = 0, *trgs = 0, *q = 0, *hin= 0, *v = 0, *term = 0;

  /* Nag Types */
  NagError fail;

  INIT_FAIL(fail);

  printf("nag_sum_fast_gauss (c06sac) Example Program Results\n");

  /* Read dimensions of arrays from data file. */
#ifdef _WIN32
  scanf_s("%*[^\n] ");
  scanf_s("%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%*[^\n]", &d, &n, &m);
  scanf_s("%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%*[^\n]", &p1, &p2, &k);
  scanf_s("%lf %*[^\n]", &tol);
#else
  scanf("%*[^\n] ");
  scanf("%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%*[^\n]", &d, &n, &m);
  scanf("%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%*[^\n]", &p1, &p2, &k);
  scanf("%lf %*[^\n]", &tol);
#endif

  /* Allocate arrays accordingly */
  if (!(srcs = NAG_ALLOC((d * n), double)) ||
      !(trgs = NAG_ALLOC((d * m), double)) ||
      !(q = NAG_ALLOC((n), double)) ||
      !(hin = NAG_ALLOC((n), double)) ||
      !(v = NAG_ALLOC((m), double)) ||
      !(term = NAG_ALLOC((m), double))
    )
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  /* Read array values from data file */
  for(i=0;i<n;i++){
#ifdef _WIN32
    scanf_s("%lf", &q[i]);
#else
    scanf("%lf", &q[i]);
#endif
  }
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  for(i=0;i<n;i++){
#ifdef _WIN32
    scanf_s("%lf", &hin[i]);
#else
    scanf("%lf", &hin[i]);
#endif
  }
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  /* Data in file is stored by column but it is read into rows here
   * First for sources
   */
#define SRCS(I, J) srcs[(J-1)*d + I - 1]
  for(j=1; j<=n; j++) {
    for(i=1; i<=d; i++) {
#ifdef _WIN32
      scanf_s("%lf", &SRCS(i,j));
#else
      scanf("%lf", &SRCS(i,j));
#endif
    }
  }
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

#define TRGS(I, J) trgs[(J-1)*d + I - 1]
  /* And then for targets */
  for(j=1; j<=m; j++){
    for(i=1; i<=d; i++){
#ifdef _WIN32
      scanf_s("%lf", &TRGS(i,j));
#else
      scanf("%lf", &TRGS(i,j));
#endif
    }
  }

  nag_sum_fast_gauss(d, srcs, n, trgs, m, q, &p1, &p2,
                     &k, hin, n, tol, v, term, &fail);

  if (fail.code != NE_NOERROR ) {
   printf("Error from nag_sum_fast_gauss (c06sac).\n%s\n", fail.message);
```

```
    exit_status = 1;
    goto END;
  }

  printf("\n      y        FGT(y)     term\n\n");

  for(i=0; i<m; i++){
    for (j=0; j<d; j++)
      printf(" %4.1f",trgs[d*i+j]);
    printf("   %8.3f    %8.6f\n",v[i],term[i]);
  }

END:
  NAG_FREE(srcs);
  NAG_FREE(trgs);
  NAG_FREE(q);
  NAG_FREE(hin);
  NAG_FREE(v);
  NAG_FREE(term);
  return exit_status;
}
```

10.2 Program Data

```
nag_sum_fast_gauss (c06sac) Example Program Data
   2   5   5                                    : d, n, m
  10  10   1                                    : p1, p2, k
   0.001                                        : tol
   0.65 0.7  0.75 0.8  0.85                     : q
   0.9  1.0  1.1  1.2  1.3                      : hin
   0.0  0.0
   0.2  0.2
   0.4  0.4
   0.6  0.6
   0.8  0.8                                     : srcs
   0.1  0.0
   0.3  0.2
   0.5  0.4
   0.7  0.6
   0.9  0.8                                     : trgs
```

10.3 Program Results

```
nag_sum_fast_gauss (c06sac) Example Program Results

      y        FGT(y)     term

  0.1  0.0      2.877    0.000000
  0.3  0.2      3.231    0.000000
  0.5  0.4      3.256    0.000000
  0.7  0.6      2.985    0.000004
  0.9  0.8      2.518    0.000470
```