

NAG Library Function Document

nag_search_vector (m01fsc)

1 Purpose

nag_search_vector (m01fsc) searches a vector of arbitrary type data objects for the first or last match to a given value.

2 Specification

```
#include <nag.h>
#include <nagm01.h>
```

```
Nag_Boolean nag_search_vector (Pointer key, const Pointer vec[], size_t n,
    ptrdiff_t stride,
    Integer (*compare)(Nag_Pointer a, Nag_Pointer b),
    Nag_SortOrder order, Nag_SearchMatch final, Pointer match, NagError *fail)
```

3 Description

nag_search_vector (m01fsc) searches a sorted vector of n arbitrary type data objects, which are stored in the elements of an array at intervals of length **stride**. **vec** must have previously been sorted into the specified order.

The function searches for the first or last match depending on the value of **final**. It returns Nag_TRUE if an exact match is found and **match** is set to point at that object. If there is no exact match then Nag_FALSE is returned and **match** is set to point to either the next later element, if **final** = Nag_First, or the next earlier element, if **final** = Nag_Last.

4 References

None.

5 Arguments

- | | | |
|----|---|--------------|
| 1: | key – Pointer | <i>Input</i> |
| | <i>On entry:</i> the object to search for. | |
| 2: | vec[n] – const Pointer | <i>Input</i> |
| | <i>On entry:</i> the array of objects to be searched. | |
| 3: | n – size_t | <i>Input</i> |
| | <i>On entry:</i> the number n of objects to be searched. | |
| | <i>Constraint:</i> $n \geq 0$. | |
| 4: | stride – ptrdiff_t | <i>Input</i> |
| | <i>On entry:</i> the increment between data items in vec to be searched. | |
| | Note: if stride is positive, vec should point at the first data object; otherwise vec should point at the last data object. | |

It should be noted that $|\mathbf{stride}|$ must be greater than or equal to $\mathbf{size_of}$ (data objects), for the search to be performed successfully. However, the code performs no check for violation of this constraint.

Constraint: $|\mathbf{stride}| > 0$.

5: **compare** – function, supplied by the user *External Function*

nag_search_vector (m01fsc) compares two data objects. If its arguments are pointers to a structure, this function must allow for the offset of the data field in the structure (if it is not the first).

The function must return:

- 1
if the first data field is less than the second,
- 0
if the first data field is equal to the second,
- 1
if the first data field is greater than the second.

The specification of **compare** is:

Integer compare (Nag_Pointer a, Nag_Pointer b)

1:	a – Nag_Pointer <i>On entry:</i> the first data field.	<i>Input</i>
2:	b – Nag_Pointer <i>On entry:</i> the second data field.	<i>Input</i>

6: **order** – Nag_SortOrder *Input*

On entry: specifies whether the array will be sorted into ascending or descending order.

Constraint: **order** = Nag_Ascending or Nag_Descending.

7: **final** – Nag_SearchMatch *Input*

On entry: specifies whether to search for the first or last match. This also determines the pointer returned if an exact match cannot be found.

Constraint: **final** = Nag_First or Nag_Last.

8: **match** – Pointer *Output*

On exit: if an exact match is found this is a pointer to a pointer to the matching data object. If an exact match is not found this is set to point to the nearest object. If **final** = Nag_First this is the next later element, otherwise the next earlier element.

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

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_BAD_PARAM

On entry, argument **final** had an illegal value.

On entry, argument **order** had an illegal value.

NE_INT_ARG_EQ

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

Constraint: **stride** = 0.

NE_INT_ARG_GT

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

On entry, **stride** = $\langle value \rangle$.
 Constraint: **stride** \leq $\langle value \rangle$.

These arguments are limited to an implementation-dependent size which is printed in the error message.

NE_INT_ARG_LT

On entry, **n** = $\langle value \rangle$.
 Constraint: **n** \geq 0.

7 Accuracy

Not applicable.

8 Further Comments

The maximum time taken by the function is approximately proportional to $\log_2 n$.

9 Example

The example program reads a key and a list of real numbers, which have been sorted into ascending order. It then searches the list for the first number which matches the key.

9.1 Program Text

```

/* nag_search_vector (m01fsc) Example Program.
 *
 * Copyright 1990 Numerical Algorithms Group.
 *
 * Mark 2 revised, 1992.
 * Mark 5 revised, 1998.
 * Mark 7 revised, 2001.
 * Mark 8 revised, 2004.
 *
 */

#include <nag.h>
#include <nagx04.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <nag_stddef.h>
#include <nagm01.h>

#ifdef __cplusplus
extern "C" {
#endif
static Integer NAG_CALL compare(const Nag_Pointer a, const Nag_Pointer b)
{
  double x = *((const double *) a);
  double y = *((const double *) b);
  return(x < y?-1:(x == y?0:1));
}
#ifdef __cplusplus
}
#endif

int main(int argc, char *argv[])
{
  FILE      *fpin, *fpout;
  Integer   exit_status = 0;

```

```

NagError fail;
Pointer match;
double key, *vec = 0;
size_t i, n;

INIT_FAIL(fail);

/* Check for command-line IO options */
fpin = nag_example_file_io(argc, argv, "-data", NULL);
fpout = nag_example_file_io(argc, argv, "-results", NULL);

/* Skip heading in data file */
fscanf(fpin, "%*[\n]");
fprintf(fpout, "nag_search_vector (m01fsc) Example Program Results\n");
/* Read number of points and number to search for */
fscanf(fpin, "%" NAG_UFMT "%lf", &n, &key);
if (n >= 1)
{
    if (!(vec = NAG_ALLOC(50, double)))
    {
        fprintf(fpout, "Allocation failure\n");
        exit_status = -1;
        goto END;
    }
}
else
{
    fprintf(fpout, "Invalid n.\n");
    exit_status = 1;
    return exit_status;
}
for (i = 0; i < n; ++i)
    fscanf(fpin, "%lf", &vec[i]);
/* nag_search_vector (m01fsc).
 * Searches a vector for either the first or last match to a
 * given value
 */
if (nag_search_vector((Pointer) &key, (Pointer) vec, n,
                    (ptrdiff_t)(sizeof(double)), compare, Nag_Ascending,
                    Nag_First, &match, &fail))
{
    fprintf(fpout, "Exact match found: ");
    if (fail.code != NE_NOERROR)
    {
        fprintf(fpout, "Error from nag_search_vector (m01fsc).\n%s\n",
                fail.message);
        exit_status = 1;
        goto END;
    }
    fprintf(fpout, "First match index: %" NAG_UFMT "\n",
            (double *) match - vec);
}
else
{
    fprintf(fpout, "No exact match found: ");
    if (match != NULL)
        fprintf(fpout, "Nag_First nearest match index = %" NAG_UFMT "\n",
                (double *) match - vec);
    else
        fprintf(fpout, "No match in the input array\n");
}
END:
if (fpin != stdin) fclose(fpin);
if (fpout != stdout) fclose(fpout);
if (vec) NAG_FREE(vec);
return exit_status;
}

```

9.2 Program Data

```
nag_search_vector (m01fsc) Example Program Data
20
2.3
0.5 0.5 1.1 1.2 1.2 1.2 1.3 2.1 2.3 2.3
2.3 2.3 4.1 5.8 5.9 6.3 6.5 6.5 8.6 9.9
```

9.3 Program Results

```
nag_search_vector (m01fsc) Example Program Results
Exact match found: First match index: 8
```
