

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

```
Boolean nag_search_vector(Pointer key, Pointer vec, size_t n,
    ptrdiff_t stride, Integer (*compare)(const Pointer, const Pointer),
    Nag_SortOrder order, Nag_SearchMatch final, Pointer *match, NagError *fail)
```

3. Description

nag_search_vector 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 **TRUE** if an exact match is found and **match** is set to point at that object. If there is no exact match then **FALSE** is returned and **match** is set to point to either the next later element, if **final** is equal to **Nag_First**, or the next earlier element, if **final** is **Nag_Last**.

4. Parameters

key

Input: the object to search for.

vec[]

Input: the array of objects to be searched.

n

Input: the number n of objects to be searched.

Constraint: $n \geq 0$.

stride

Input: 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 **|stride|** must be greater than or equal to **size_of** (data objects), for the search to be performed successfully. However, the code performs no check for violation of this constraint.

Constraint: **|stride|** > 0.

compare

User-supplied function: this function 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.

order

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

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

final

Input: 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**.

match

Output: 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** is **Nag_First** this is the next later element, otherwise the next earlier element.

fail

The NAG error parameter, see the Essential Introduction to the NAG C Library.

5. Error Indications and Warnings**NE_INT_ARG_LT**

On entry, **n** must not be less than 0: **n** = $\langle value \rangle$.

NE_INT_ARG_GT

On entry, **n** must not be greater than $\langle value \rangle$: **n** = $\langle value \rangle$.

On entry, $|\mathbf{stride}|$ must not be greater than $\langle value \rangle$: **stride** = $\langle value \rangle$.

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

NE_INT_ARG_EQ

On entry, **stride** must not be equal to 0: **stride** = $\langle value \rangle$.

NE_BAD_PARAM

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

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

6. Further Comments

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

7. See Also

nag_quicksort (m01esc)
 nag_rank_sort (m01dsc)
 nag_reorder_vector (m01esc)
 nag_make_indices (m01zac)

8. 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.

8.1. Program Text

```
/* nag_search_vector(m01fsc) Example Program
 *
 * Copyright 1990 Numerical Algorithms Group.
 *
 * Mark 2 revised, 1992.
 */

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

#ifdef NAG_PROTO
static Integer compare(const Pointer a,const Pointer b)
```

```

#else
    static Integer compare(a,b)
        Pointer a, b;
#endif
{
    double x = *((double *)a);
    double y = *((double *)b);
    return (x<y ? -1 : (x==y ? 0 : 1));
}

main()
{
    double key, vec[50];
    size_t i, n;
    Pointer match;

    /* Skip heading in data file */
    Vscanf("%*[^\\n]");
    Vprintf("m01fsc Example Program Results\\n");
    /* Read number of points and number to search for */
    Vscanf("%d%lf", &n, &key);
    if (n>=0)
    {
        for (i=0; i<n; ++i)
            Vscanf("%lf",&vec[i]);
        if (m01fsc((Pointer) &key, (Pointer) vec, n, (ptrdiff_t)(sizeof(double)),
            compare, Nag_Ascending, Nag_First, &match, NAGERR_DEFAULT))
        {
            Vprintf("Exact match found: ");
            Vprintf("First match index: %d\\n", (double *) match-vec);
        }
        else
        {
            Vprintf("No exact match found: ");
            if (match!=NULL)
                Vprintf("Nag_First nearest match index = %d\\n", (double *) match-vec);
            else
                Vprintf("No match in the input array\\n");
        }
        exit(EXIT_SUCCESS);
    }
    else
    {
        Vfprintf(stderr, "Data error: program terminated\\n");
        exit(EXIT_FAILURE);
    }
}

```

8.2. Program Data

```

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

```

8.3. Program Results

```

m01fsc Example Program Results
Exact match found: First match index: 8

```