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)(const Nag_Pointer a, const 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 \(\text{stride}\). vec must have previously been sorted into the specified order.

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

4 References
None.

5 Arguments
1: key – Pointer
\textit{Input}
\textit{On entry}: the object to search for.

2: vec[n] – const Pointer
\textit{Input}
\textit{On entry}: the array of objects to be searched.

3: n – size_t
\textit{Input}
\textit{On entry}: the number \(n\) of objects to be searched.
\textit{Constraint}: \(n \geq 0\).

4: stride – ptrdiff_t
\textit{Input}
\textit{On entry}: the increment between data items in vec to be searched.
\textit{Note}: if \text{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 $|\text{stride}|$ must be greater than or equal to $\text{size} \_\text{of} (\text{data objects})$, for the search to be performed successfully. However, the code performs no check for violation of this constraint.

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

5: compare – function, supplied by the user

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:

```c
Integer compare (const Nag_Pointer a, const Nag_Pointer b)
```

1: a – const Nag_Pointer

$On entry$: the first data field.

2: b – const Nag_Pointer

$On entry$: the second data field.

6: order – Nag_SortOrder

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

$Constraint: order = \text{NagAscending} \text{ or NagDescending}.$

7: final – Nag_SearchMatch

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

$Constraint: final = \text{NagFirst} \text{ or NagLast}.$

8: match – Pointer *

$On exit$: 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 $final = \text{NagFirst}$ this is the next later element, otherwise the next earlier element.

9: fail – NagError *

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 = (value).
Constraint: stride ≠ 0.

NE_INT_ARG_GT
On entry, n = (value).
Constraint: n ≤ (value).

On entry, stride = (value).
Constraint: |stride| ≤ (value).
These arguments are limited to an implementation-dependent size which is printed in the error message.

NE_INT_ARG_LT
On entry, n = (value).
Constraint: n ≥ 0.

7 Accuracy
Not applicable.

8 Parallelism and Performance
Not applicable.

9 Further Comments
The maximum time taken by the function is approximately proportional to \( \log_2 n \).

10 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.

10.1 Program Text
/* nag_search_vector (m01fsc) Example Program. */
/* Copyright 2014 Numerical Algorithms Group. */
/* Mark 2 revised, 1992. */
/* Mark 5 revised, 1998. */
/* Mark 7 revised, 2001. */
/* Mark 8 revised, 2004. */

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

#ifdef __cplusplus
extern "C" {
#endef
#endif

static Integer NAG_CALL compare(const Nag_Pointer a, const Nag_Pointer b);

#ifdef __cplusplus
}
#endif
int main(void)
{
    Integer  exit_status = 0;
    NagError fail;
    Pointer  match;
    double   key, *vec = 0;
    size_t   i, n;

    INIT_FAIL(fail);

    /* Skip heading in data file */
    #ifdef _WIN32
        scanf_s("%*[\n");
    #else
        scanf("%*[\n");
    #endif
    printf("nag_search_vector (m01fsc) Example Program Results
");
    /* Read number of points and number to search for */
    #ifdef _WIN32
        scanf_s("\%" NAG_UFMT "\lf", &n, &key);
    #else
        scanf("\%" NAG_UFMT "\lf", &n, &key);
    #endif
    if (n >= 1)
    {
        if (!(vec = NAG_ALLOC(50, double)))
        {
            printf("Allocation failure
");
            exit_status = -1;
            goto END;
        }
    }
    else
    {
        printf("Invalid n.
");
        exit_status = 1;
        return exit_status;
    }
    for (i = 0; i < n; ++i)
    #ifdef _WIN32
        scanf_s("\lf", &vec[i]);
    #else
        scanf("\lf", &vec[i]);
    #endif
    /* 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))
    {
        printf("Exact match found: ");
        if (fail.code != NE_NOERROR)
        {
            printf("Error from nag_search_vector (m01fsc).\n", fail.message);
            exit_status = 1;
            goto END;
        }
        printf("First match index: \"NAG_UFMT\n",
            (size_t)((double *) match - vec));
    }
    else
    {
        printf("No exact match found: ");
        if (match != NULL)
            printf("Nag_First nearest match index = \"NAG_UFMT\n",
                (size_t)((double *) match - vec));
        else
            printf("No match in the input array\n");
    }
}

m01fsc

m01fsc.4

Mark 25
END:
NAG_FREE(vec);
return exit_status;
}

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));
}

10.2 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

10.3 Program Results

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