

NAG Library Function Document

nag_1d_pade_eval (e02rbc)

1 Purpose

nag_1d_pade_eval (e02rbc) evaluates a rational function at a user-supplied point, given the numerator and denominator coefficients.

2 Specification

```
#include <nag.h>
#include <nage02.h>

void nag_1d_pade_eval (const double a[], Integer ia, const double b[],
                      Integer ib, double x, double *ans, NagError *fail)
```

3 Description

Given a real value x and the coefficients a_j , for $j = 0, 1, \dots, l$ and b_k , for $k = 0, 1, \dots, m$, nag_1d_pade_eval (e02rbc) evaluates the rational function

$$\frac{\sum_{j=0}^l a_j x^j}{\sum_{k=0}^m b_k x^k}.$$

using nested multiplication (see Conte and de Boor (1965)).

A particular use of nag_1d_pade_eval (e02rbc) is to compute values of the Padé approximants determined by nag_1d_pade (e02rac).

4 References

Conte S D and de Boor C (1965) *Elementary Numerical Analysis* McGraw–Hill

Peters G and Wilkinson J H (1971) Practical problems arising in the solution of polynomial equations *J. Inst. Maths. Applics.* **8** 16–35

5 Arguments

- 1: **a[ia]** – const double *Input*
On entry: **a[j]**, for $j = 1, 2, \dots, l + 1$, must contain the value of the coefficient a_j in the numerator of the rational function.
- 2: **ia** – Integer *Input*
On entry: the value of $l + 1$, where l is the degree of the numerator.
Constraint: **ia** ≥ 1 .
- 3: **b[ib]** – const double *Input*
On entry: **b[k]**, for $k = 1, 2, \dots, m + 1$, must contain the value of the coefficient b_k in the denominator of the rational function.
Constraint: if **ib** = 1, **b[0]** $\neq 0.0$.

- 4: **ib** – Integer *Input*
On entry: the value of $m + 1$, where m is the degree of the denominator.
Constraint: **ib** ≥ 1 .
- 5: **x** – double *Input*
On entry: the point x at which the rational function is to be evaluated.
- 6: **ans** – double * *Output*
On exit: the result of evaluating the rational function at the given point x .
- 7: **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 $\langle value \rangle$ had an illegal value.

NE_INT

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

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

NE_INT_ARRAY

The first **ib** entries in **b** are zero: **ib** = $\langle value \rangle$.

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_POLE_PRESENT

Evaluation at or near a pole.

7 Accuracy

A running error analysis for polynomial evaluation by nested multiplication using the recurrence suggested by Kahan (see Peters and Wilkinson (1971)) is used to detect whether you are attempting to evaluate the approximant at or near a pole.

8 Parallelism and Performance

Not applicable.

9 Further Comments

The time taken is approximately proportional to $l + m$.

10 Example

This example first calls `nag_1d_pade` (e02rac) to calculate the 4/4 Padé approximant to e^x , and then uses `nag_1d_pade_eval` (e02rbc) to evaluate the approximant at $x = 0.1, 0.2, \dots, 1.0$.

10.1 Program Text

```

/* nag_1d_pade_eval (e02rbc) Example Program.
 *
 * Copyright 2001 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 */

#include <stdio.h>
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nage02.h>

int main(void)
{
    /* Scalars */
    double ans, tval, x;
    Integer exit_status, i, l, m, ia, ib, ic;
    NagError fail;

    /* Arrays */
    double *aa = 0, *bb = 0, *cc = 0;

    INIT_FAIL(fail);

    exit_status = 0;
    printf("nag_1d_pade_eval (e02rbc) Example Program Results\n");

    l = 4;
    m = 4;
    ia = l + 1;
    ib = m + 1;
    ic = ia + ib - 1;

    /* Allocate memory */
    if (!(aa = NAG_ALLOC(ia, double)) ||
        !(bb = NAG_ALLOC(ib, double)) ||
        !(cc = NAG_ALLOC(ic, double)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }

    cc[0] = 1.0;
    for (i = 1; i <= ic - 1; ++i)
        cc[i] = cc[i-1] / (double) i;

    /* nag_1d_pade (e02rac).
     * Pade-approximants
     */
    nag_1d_pade(ia, ib, cc, aa, bb, &fail);
    if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_1d_pade (e02rac).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }

    printf("\n");
    printf("    x          Pade          True\n");
}

```

```

for (i = 0; i < 10; ++i)
{
  x = (double)(i + 1) / 10.0;
  /* nag_ld_pade_eval (e02rbc).
  * Evaluation of fitted rational function as computed by
  * nag_ld_pade (e02rac)
  */
  nag_ld_pade_eval(aa, ia, bb, ib, x, &ans, &fail);
  if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_ld_pade_eval (e02rbc).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }

  tval = exp(x);
  printf("%6.1f%15.5e%15.5e\n", x, ans, tval);
}

END:
NAG_FREE(aa);
NAG_FREE(bb);
NAG_FREE(cc);

return exit_status;
}

```

10.2 Program Data

None.

10.3 Program Results

nag_ld_pade_eval (e02rbc) Example Program Results

x	Pade	True
0.1	1.10517e+00	1.10517e+00
0.2	1.22140e+00	1.22140e+00
0.3	1.34986e+00	1.34986e+00
0.4	1.49182e+00	1.49182e+00
0.5	1.64872e+00	1.64872e+00
0.6	1.82212e+00	1.82212e+00
0.7	2.01375e+00	2.01375e+00
0.8	2.22554e+00	2.22554e+00
0.9	2.45960e+00	2.45960e+00
1.0	2.71828e+00	2.71828e+00

