NAG C 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

```c
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 \( j = 0, \ldots, l \) and \( b_k \), for \( k = 0, \ldots, m \), 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 (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


5 Parameters

1: \( a[\text{ia}] \) – const double

On entry: \( a[j] \), for \( j = 1, 2, \ldots, l + 1 \), must contain the value of the coefficient \( a_j \) in the numerator of the rational function.

2: \( \text{ia} \) – Integer

On entry: the value of \( l + 1 \), where \( l \) is the degree of the numerator.

Constraint: \( \text{ia} \geq 1 \).

3: \( b[\text{ib}] \) – const double

On entry: \( b[k] \), for \( k = 1, 2, \ldots, m + 1 \), must contain the value of the coefficient \( b_k \) in the denominator of the rational function.

Constraint: if \( \text{ib} = 1 \), \( b[0] \) must be non-zero.
4:   ib – Integer  
     \textit{Input}
     
     On entry: the value of $m + 1$, where $m$ is the degree of the denominator.
     
     Constraint: $ib \geq 1$.

5:   x – double  
     \textit{Input}
     
     On entry: the point $x$ at which the rational function is to be evaluated.

6:   ans – double *  
     \textit{Output}
     
     On exit: the result of evaluating the rational function at the given point $x$.

7:   fail – NagError *  
     \textit{Input/Output}
     
     The NAG error parameter (see the Essential Introduction).

6  Error Indicators and Warnings

\textbf{NE_INT}

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

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

\textbf{NE_INT_ARRAY}

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

\textbf{NE_POLE_PRESENT}

Evaluation at or near a pole.

\textbf{NE_BAD_PARAM}

On entry, parameter $\langle value \rangle$ had an illegal value.

\textbf{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 consult NAG for assistance.

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 the user is attempting to evaluate the approximant at or near a pole.

8  Further Comments

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

9  Example

The example program first calls \texttt{nag_1d_pade (e02rac)} to calculate the 4/4 Padé approximant to $e^x$, and then uses \texttt{nag_1d_pade_eval (e02rbc)} to evaluate the approximant at $x = 0.1, 0.2, \ldots, 1.0$. 
9.1 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("e02rbc Example Program Results\n");
    l = 4;
    m = 4;
    ia = l + 1;
    ib = m + 1;
    ic = ia + ib - 1;
    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;
        e02rac(ia, ib, cc, aa, bb, &fail);
        if (fail.code != NE_NOERROR)
        {
            printf("Error from 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;
        e02rbc(aa, ia, bb, ib, x, &ans, &fail);
        if (fail.code != NE_NOERROR)
        {
            printf("Error from e02rbc.\n%s\n", fail.message);
            exit_status = 1;
            goto END;
        }
    }
    return exit_status;
}
tval = exp(x);
Vprintf("%6.1f%15.5e%15.5e\n", x, ans, tval);
}

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

return exit_status;
}

9.2 Program Data

None.

9.3 Program Results

e02rbc Example Program Results

<table>
<thead>
<tr>
<th>x</th>
<th>Pade</th>
<th>True</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>1.10517e+00</td>
<td>1.10517e+00</td>
</tr>
<tr>
<td>0.2</td>
<td>1.22140e+00</td>
<td>1.22140e+00</td>
</tr>
<tr>
<td>0.3</td>
<td>1.34986e+00</td>
<td>1.34986e+00</td>
</tr>
<tr>
<td>0.4</td>
<td>1.49182e+00</td>
<td>1.49182e+00</td>
</tr>
<tr>
<td>0.5</td>
<td>1.64872e+00</td>
<td>1.64872e+00</td>
</tr>
<tr>
<td>0.6</td>
<td>1.82212e+00</td>
<td>1.82212e+00</td>
</tr>
<tr>
<td>0.7</td>
<td>2.01375e+00</td>
<td>2.01375e+00</td>
</tr>
<tr>
<td>0.8</td>
<td>2.22554e+00</td>
<td>2.22554e+00</td>
</tr>
<tr>
<td>0.9</td>
<td>2.45960e+00</td>
<td>2.45960e+00</td>
</tr>
<tr>
<td>1.0</td>
<td>2.71828e+00</td>
<td>2.71828e+00</td>
</tr>
</tbody>
</table>