NAG Library Function Document

nag_1d_pade (e02rac)

1 Purpose

nag_1d_pade (e02rac) calculates the coefficients in a Padé approximant to a function from its user-supplied Maclaurin expansion.

2 Specification

```
#include <nag.h>
#include <nage02.h>
void nag_1d_pade (Integer ia, Integer ib, const double c[], double a[],
     double b[], NagError *fail)
```

3 Description

Given a power series

$$c_0 + c_1x + c_2x^2 + \cdots + c_{l+m}x^{l+m} + \cdots$$

nag_1d_pade (e02rac) uses the coefficients c_i, for $i = 0, 1, \ldots, l+m$, to form the $[l/m]$ Padé approximant of the form

$$\frac{a_0 + a_1x + a_2x^2 + \cdots + a_lx^l}{b_0 + b_1x + b_2x^2 + \cdots + b_mx^m}$$

with b_0 defined to be unity. The two sets of coefficients a_j, for $j = 0, 1, \ldots, l$, and b_k, for $k = 0, 1, \ldots, m$, in the numerator and denominator are calculated by direct solution of the Padé equations (see Graves–Morris (1979)); these values are returned through the argument list unless the approximant is degenerate.

Padé approximation is a useful technique when values of a function are to be obtained from its Maclaurin expansion but convergence of the series is unacceptably slow or even nonexistent. It is based on the hypothesis of the existence of a sequence of convergent rational approximations, as described in Baker and Graves–Morris (1981) and Graves–Morris (1979).

Unless there are reasons to the contrary (as discussed in Chapter 4, Section 2, Chapters 5 and 6 of Baker and Graves–Morris (1981)), one normally uses the diagonal sequence of Padé approximants, namely

$$\{[m/m], m = 0, 1, 2, \ldots\}.$$

Subsequent evaluation of the approximant at a given value of x may be carried out using nag_1d_pade_eval (e02rbc).

4 References

Baker G A Jr and Graves–Morris P R (1981) Padé approximants, Part 1: Basic theory *encyclopaedia of Mathematics and its Applications* Addison–Wesley

Graves–Morris P R (1979) The numerical calculation of Padé approximants *Padé Approximation and its Applications. Lecture Notes in Mathematics* (ed L Wuytack) **765** 231–245 Adison–Wesley

5 Arguments

1: **ia** – Integer *Input*
2: **ib** – Integer *Input*

On entry: **ia** must specify $l+1$ and **ib** must specify $m+1$, where l and m are the degrees of the numerator and denominator of the approximant, respectively.

Constraint: **ia** ≥ 1 and **ib** ≥ 1.

3: **c**[(**ia** + **ib** − **1**)] – const double *Input*

On entry: **c**$[i-1]$ must specify, for $i = 1, 2, \ldots, l+m+1$, the coefficient of x^{i-1} in the given power series.

4: **a**[**ia**] – double *Output*

On exit: **a**$[j]$, for $j = 1, 2, \ldots, l+1$, contains the coefficient a_j in the numerator of the approximant.

5: **b**[**ib**] – double *Output*

On exit: **b**$[k]$, for $k = 1, 2, \ldots, m+1$, contains the coefficient b_k in the denominator of the approximant.

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

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

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_DEGENERATE

The Pade approximant is degenerate.

NE_INT_2

On entry, **ib** = ⟨*value*⟩ and **ia** = ⟨*value*⟩.
Constraint: **ia** ≥ 1 and **ib** ≥ 1.

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.

An unexpected error has been triggered by this function. Please contact NAG.
See Section 3.6.6 in the Essential Introduction for further information.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in the Essential Introduction for further information.

7 Accuracy

The solution should be the best possible to the extent to which the solution is determined by the input coefficients. It is recommended that you determine the locations of the zeros of the numerator and denominator polynomials, both to examine compatibility with the analytic structure of the given function and to detect defects. (Defects are nearby pole-zero pairs; defects close to $x = 0.0$ characterise ill-conditioning in the construction of the approximant.) Defects occur in regions where the approximation is necessarily inaccurate. The example program calls nag_zeros_real_poly (c02agc) to determine the above zeros.

It is easy to test the stability of the computed numerator and denominator coefficients by making small perturbations of the original Maclaurin series coefficients (e.g., c_l or c_{l+m}). These questions of intrinsic error of the approximants and computational error in their calculation are discussed in Chapter 2 of Baker and Graves–Morris (1981).

8 Parallelism and Performance

nag_1d_pade (e02rac) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_1d_pade (e02rac) makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

Please consult the X06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also consult the Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

The time taken is approximately proportional to m^3.

10 Example

This example calculates the $[4/4]$ Padé approximant of e^x (whose power-series coefficients are first stored in the array **c**). The poles and zeros are then calculated to check the character of the $[4/4]$ Padé approximant.

10.1 Program Text

```
/* nag_1d_pade (e02rac) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 * Mark 7b revised, 2004.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagc02.h>
#include <nage02.h>

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

  /* Arrays */
  double   *aa = 0, *bb = 0, *cc = 0, *dd = 0;
  Complex  *z = 0;
```

```
  INIT_FAIL(fail);

  exit_status = 0;
  printf("nag_1d_pade (e02rac) 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)) ||
      !(dd = NAG_ALLOC(ia + ib, double)) ||
      !(z = NAG_ALLOC(l+m, Complex)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Power series coefficients in cc */
  cc[0] = 1.0;
  for (i = 1; i <= 8; ++i)
    cc[i] = cc[i-1] / (double) i;

  printf("\n");

  printf("The given series coefficients are\n");

  for (i = 1; i <= ic; ++i)
    {
      printf("%13.4e", cc[i-1]);
      printf(i%5 == 0 || i == ic?"\n":" ");
    }

  /* 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("Numerator coefficients\n");

  for (i = 1; i <= ia; ++i)
    {
      printf("%13.4e", aa[i-1]);
      printf(i%5 == 0 || i == ia?"\n":" ");
    }

  printf("\n");
  printf("Denominator coefficients\n");

  for (i = 1; i <= ib; ++i)
    {
      printf("%13.4e", bb[i-1]);
      printf(i%5 == 0 || i == ib?"\n":" ");
    }

  /* Calculate zeros of the approximant using nag_zeros_real_poly (c02agc) */
  /* First need to reverse order of coefficients */
  for (i = 1; i <= ia; ++i)
```

```
    dd[ia-i] = aa[i-1];

  /* nag_zeros_real_poly (c02agc).
   * Zeros of a polynomial with real coefficients
   */
  nag_zeros_real_poly(l, dd, Nag_TRUE, z, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf(
              "Error from nag_zeros_real_poly (c02agc), 1st call.\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }

  printf("\n");
  printf("Zeros of approximant are at\n");
  printf("    Real part    Imag part\n");
  for (i = 1; i <= l; ++i)
    printf("%13.4e%13.4e\n", z[i-1].re, z[i-1].im);

  /* Calculate poles of the approximant using nag_zeros_real_poly (c02agc) */
  /* Reverse order of coefficients */
  for (i = 1; i <= ib; ++i)
    dd[ib-i] = bb[i-1];

  /* nag_zeros_real_poly (c02agc), see above. */
  nag_zeros_real_poly(m, dd, Nag_TRUE, z, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf(
              "Error from nag_zeros_real_poly (c02agc), 2nd call.\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }

  printf("\n");
  printf("Poles of approximant are at\n");
  printf("    Real part    Imag part\n");
  for (i = 1; i <= m; ++i)
    printf("%13.4e%13.4e\n", z[i-1].re, z[i-1].im);

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

  return exit_status;
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_1d_pade (e02rac) Example Program Results

The given series coefficients are
   1.0000e+00    1.0000e+00    5.0000e-01    1.6667e-01    4.1667e-02
   8.3333e-03    1.3889e-03    1.9841e-04    2.4802e-05

Numerator coefficients
   1.0000e+00    5.0000e-01    1.0714e-01    1.1905e-02    5.9524e-04

Denominator coefficients
   1.0000e+00   -5.0000e-01    1.0714e-01   -1.1905e-02    5.9524e-04
```

```
Zeros of approximant are at
    Real part    Imag part
  -5.7924e+00   1.7345e+00
  -5.7924e+00  -1.7345e+00
  -4.2076e+00   5.3148e+00
  -4.2076e+00  -5.3148e+00

Poles of approximant are at
    Real part    Imag part
   5.7924e+00   1.7345e+00
   5.7924e+00  -1.7345e+00
   4.2076e+00   5.3148e+00
   4.2076e+00  -5.3148e+00
```