Module 11.2: nag_quad_1d_inf

Numerical Integration over an Infinite Interval

This module provides procedures for computing the value of a one-dimensional definite integral over a *semi-infinite* or *infinite* interval.

Contents

Introduction

The procedures in this module are designed to estimate the value of a one-dimensional definite integral of the form

$$\int_a^b f(x)\, dx,$$

where the function $f(x)$ is defined by the user and the limits of integration a and/or b are infinite.

This module also provides a procedure for integrands of the form

$$f(x) = w(x)g(x),$$

which contain a factor $w(x)$, called the *weight function*. The weight function is of the form $\cos(\omega x)$ or $\sin(\omega x)$ and the interval of integration is semi-infinite (lower limit finite). The procedure takes full account of any peculiar behaviour attributable to the $w(x)$ factor. For further details see the Mathematical Background section of module `nag_quad_1d` (11.1).

However, if $f(x)$ is defined numerically at four or more points, and the portion of the integral lying outside the range of the points supplied may be neglected, then the Gill–Miller finite difference method `nag_quad_1d_data` in module `nag_quad_1d` (11.1) should be used.

Procedure: nag_quad_1d_inf_gen

1 Description

nag_quad_1d_inf_gen computes an approximation to the integral of a function $f(x)$ over a semi-infinite or infinite interval $[a, b]$:

$$I = \int_a^b f(x)\,dx.$$

The entire infinite integration interval is first transformed to (0,1] (see Section 6.1). An adaptive algorithm, based on the Gauss 7-point and Kronrod 15-point rules, is then employed on the transformed integral. See Section 6.1 for more details.

This procedure requires a vector valued user-supplied function to evaluate the integrand at an array of points, and will therefore be more efficient if the evaluation can be performed in vector mode on a vector-processing machine.

2 Usage

```
USE nag_quad_1d_inf

CALL nag_quad_1d_inf_gen(f, a, inf_limit, result  [, optional arguments])
```

3 Arguments

3.1 Mandatory Arguments

f — function

f must return the values of the integrand f at a set of points.

```
function f(x)

real(kind=wp), intent(in) ::  x(:)
```

Input: the points at which the integrand f must be evaluated.

```
real(kind=wp) ::  f(SIZE(x))
```

Result: f(i) must contain the value of f at x(i), for $i = 1, 2, \ldots,$SIZE(x).

a — real(kind=wp), intent(in)

Input: the finite limit of the integration interval.

Note: a is not used if the interval is doubly infinite.

inf_limit — character(len=1), intent(in)

Input: indicates the infinite limit of integration:

if inf_limit = 'u' or 'U', the interval is $[a, \infty]$;
if inf_limit = 'l' or 'L', the interval is $[-\infty, a]$;
if inf_limit = 'b' or 'B', the interval is $[-\infty, \infty]$.

Constraints: inf_limit = 'u', 'U', 'l', 'L', 'b' or 'B' .

result — real(kind=wp), intent(out)

Output: the approximation to the integral I.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

abs_acc — real(kind=wp), intent(in), optional

Input: the absolute accuracy required.

Default: `abs_acc` = `SQRT(EPSILON(1.0_`wp`))`.

Constraints: `abs_acc` ≥ 0.0. Both `rel_acc` and `abs_acc` cannot be zero.

rel_acc — real(kind=wp), intent(in), optional

Input: the relative accuracy required.

Default: `rel_acc` $= 10^{-4}$.

Constraints: `rel_acc` ≥ 0.0. Both `rel_acc` and `abs_acc` cannot be zero.

abs_err — real(kind=wp), intent(out), optional

Output: an estimate of the modulus of the absolute error, which should be an upper bound for $\mid I - \texttt{result} \mid$.

max_num_subint — integer, intent(in), optional

Input: the maximum number of subintervals to be used in the subdivision strategy.

Default: `max_num_subint` = 500.

Constraints: `max_num_subint` ≥ 1.

num_subint_used — integer, intent(out), optional

Output: the final number of subintervals used in the subdivision strategy.

num_fun_eval — integer, intent(out), optional

Output: the actual number of integrand evaluations.

subint_info(:, :) — real(kind=wp), pointer, optional

Output: details of the computation which may be examined in the event of a failure. This array contains the end-points of the subinterval used by the procedure, along with the integral contribution and error estimates over these subintervals. Specifically, for $i = 1, 2, \ldots, m$, let r_i denote the approximation to the value of the integral over the subinterval $[x_i, x_{i+1}]$ in the partition of $[a, b]$, and let e_i be the corresponding absolute error estimate. Then

$$\int_{x_i}^{x_{i+1}} f(x)\, dx \simeq r_i \quad \text{and} \quad \texttt{result} = \sum_{i=1}^{m} r_i,$$

unless this procedure terminates while testing for divergence of the integral. In this case `result` (and `abs_err`) are taken to be the values returned from the extrapolation process. The value of m is returned in `num_subint_used`, and the values of x_i, r_i and e_i are stored in the array `subint_info`, that is:

$$x_i = \texttt{subint\_info}(i,1),\ i \neq m+1,\ x_{m+1} = b,$$
$$r_i = \texttt{subint\_info}(i,2),$$
$$e_i = \texttt{subint\_info}(i,3).$$

Note that this information applies to the integral transformed to (0,1] as described in Section 6.1, not to the original integral.

Note: this array is allocated by `nag_quad_1d_inf_gen`. It should be deallocated when no longer required.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
320	The procedure was unable to allocate enough memory.

Failures (error%level = 2):

error%code	Description
201	Maximum number of subdivisions allowed has been reached. The accuracy requirements have not been achieved. Look at the integrand in order to determine the integration difficulties. If the position of a local difficulty within the interval can be determined (e.g., a singularity of the integrand or its derivative, a peak, a discontinuity) you will probably gain from splitting up the interval at this point and calling this procedure on the infinite interval and an appropriate integrator on the finite subinterval. Alternatively, consider relaxing the accuracy requirements specified by the optional arguments `abs_acc` and `rel_acc`, or increasing the value of the optional argument `max_num_subint`.
202	Round-off error prevents the requested accuracy from being achieved. The error may be under-estimated. Consider requesting less accuracy.
203	Extremely bad local integrand behaviour. This causes a very strong subdivision around one (or more) points of the interval. The same advice applies as in the case `error%code` = 201.
204	The requested accuracy cannot be achieved. The extrapolation does not increase the accuracy satisfactorily; the result returned is the best which can be obtained. The same advice applies as in the case `error%code` = 201.
205	The integral is probably divergent, or slowly convergent. Note that divergence can also occur with any other value of `error%code` when `error%level` = 2.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 1 of this module document.

6 Further Comments

6.1 Algorithmic Detail

This procedure is a modified version of the QUADPACK procedure QAGI (Piessens *et al.* [5]). The entire infinite integration interval is first transformed to (0,1] using one of the identities:

$$\int_{-\infty}^{a} f(x)\,dx = \int_{0}^{1} f\left(a - \frac{1-t}{t}\right)\frac{1}{t^2}\,dt,$$

$$\int_a^\infty f(x)\,dx = \int_0^1 f\left(a + \frac{1-t}{t}\right)\frac{1}{t^2}\,dt,$$

$$\int_{-\infty}^\infty f(x)\,dx = \int_0^\infty (f(x) + f(-x))\,dx = \int_0^1 \left[f\left(\frac{1-t}{t}\right) + f\left(\frac{-1+t}{t}\right)\right]\frac{1}{t^2}\,dt,$$

where a represents a finite integration limit. An adaptive algorithm, based on the Gauss 7-point and Kronrod 15-point rules, is then employed on the transformed integral. The algorithm, described by De Doncker [1], incorporates a global acceptance criterion (as defined by Malcolm and Simpson [2]) together with the ϵ-algorithm (Wynn [6]) to perform extrapolation. The local error estimation is described in Piessens *et al.* [5].

This procedure is not suitable for integrands which decay slowly towards an infinite end-point, and oscillate in sign over the entire interval. For this class of functions it may be possible to calculate the integral by integrating between the zeros and invoking some extrapolation process.

6.2 Accuracy

The procedure cannot guarantee, but in practice usually achieves, the following accuracy:

$$|\, I - \texttt{result} \,| \le tol,$$

where

$$tol = \max\left(\texttt{abs\_acc},\ \texttt{rel\_acc} \times |\, I \,|\right)$$

and `abs_acc` and `rel_acc` are the specified absolute and relative accuracies, or their default values. The optional output argument `abs_err` normally satisfies

$$|\, I - \texttt{result} \,| \le \texttt{abs\_err} \le tol.$$

Procedure: nag_quad_1d_inf_wt_trig

1 Description

nag_quad_1d_inf_wt_trig computes an approximation to the integral (Fourier transform)

$$I = \int_a^\infty w(x)\, g(x)\, dx,$$

where $w(x) = \sin(\omega x)$ or $\cos(\omega x)$ for a given value of ω.

It is an adaptive procedure and over successive intervals

$$C_k = [a + (k-1)c, a + kc], \quad k = 1, 2, \ldots, n$$

integration is performed by the same algorithm as is used by nag_quad_1d_wt_trig. The intervals C_k are of constant length c which depends on the value of ω. See Section 6.1 for more details.

If $\omega = 0$ and trig_wt = 'c', this procedure uses the same algorithm as the procedure nag_quad_1d_inf_gen (with rel_acc = 0.0).

In contrast to the other procedures in this chapter, this procedure works with a specified *absolute* accuracy. See Section 6.1 for more details.

This procedure requires a vector valued user-supplied function to evaluate g at an array of points, and will therefore be more efficient if the evaluation can be performed in vector mode on a vector-processing machine.

2 Usage

```
USE nag_quad_1d_inf
CALL nag_quad_1d_inf_wt_trig(g, a, omega, trig_wt, result  [, optional arguments])
```

3 Arguments

3.1 Mandatory Arguments

g — function

g must return the values of the function g at a set of points.

```
function g(x)

real(kind=wp), intent(in) ::  x(:)
```
Input: the points at which g must be evaluated.

```
real(kind=wp) ::  g(SIZE(x))
```
Result: g(i) must contain the value of g at x(i), for $i = 1, 2, \ldots$,SIZE(x).

a — real(kind=*wp*), intent(in)

Input: the lower limit of integration, a.

omega — real(kind=*wp*), intent(in)

Input: the parameter ω in the weight function of the transform.

trig_wt — character(len=1), intent(in)

Input: indicates which transform to be computed.

If trig_wt = 'c' or 'C', $w(x) = \cos(\omega x)$;

if trig_wt = 's' or 'S', $w(x) = \sin(\omega x)$.

Constraints: trig_wt = 'c', 'C', 's' or 'S'.

result — real(kind=*wp*), intent(out)

Output: the approximation to the integral I.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

abs_acc — real(kind=*wp*), intent(in), optional

Input: the absolute accuracy required.

Default: abs_acc = SQRT(EPSILON(1.0_*wp*)).

Constraints: abs_acc ≥ 0.0.

abs_err — real(kind=*wp*), intent(out), optional

Output: an estimate of the modulus of the absolute error, which should be an upper bound for $|\, I -$ result $|$.

max_num_intvl — integer, intent(in), optional

Input: an upper bound on the number of intervals C_k needed for integration.

Default: max_num_intvl = 100.

Constraints: max_num_intvl ≥ 3.

max_num_subint — integer, intent(in), optional

Input: maximum number of subintervals allowed over each C_k.

Default: max_num_subint = 500.

Constraints: max_num_subint ≥ 1.

num_intvl_used — integer, intent(out), optional

Output: the total number of intervals C_k used for the integration; n.

num_subint_used — integer, intent(out), optional

Output: the final maximum number of subintervals actually used for integrating over any of the intervals C_k.

num_fun_eval — integer, intent(out), optional

Output: the actual number of integrand evaluations.

intvl_info(:, :) — real(kind=*wp*), pointer, optional

Output: details of the computation which may be examined in the event of a failure.
The elements intvl_info(k, 1) and intvl_info(k, 2), respectively, contain the values of the integral and its corresponding error estimate contribution over the interval C_k, for $k = 1, 2, \ldots,$num_intvl_used.

Note: this array is allocated by nag_quad_1d_inf_wt_trig. It should be deallocated when no longer required.

intvl_error_code(:) — integer, pointer, optional

Output: `intvl_error_code(k)` contains the error flag corresponding to `intvl_info(k,1)`, for $k = 1, 2, \ldots,$`num_intvl_used`. See Section 4.

Note: this array is allocated by `nag_quad_1d_inf_wt_trig`. It should be deallocated when no longer required.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
320	The procedure was unable to allocate enough memory.

Failures (error%level = 2):

error%code	Description
201	Maximum number of subdivisions allowed has been reached.
	The accuracy requirements have not been achieved. Look at the integrand in order to determine the integration difficulties. If the position of a local difficulty within the interval can be determined (e.g., a singularity of the integrand or its derivative, a peak, a discontinuity) you will probably gain from splitting up the interval at this point and calling this procedure on the infinite subinterval and an appropriate integrator on the finite subinterval. Alternatively, consider relaxing the accuracy requirements specified by the optional argument `abs_acc`, or increasing the value of the optional argument `max_num_subint`.
202	Round-off error prevents the requested accuracy from being achieved.
	The error may be under-estimated. Consider requesting less accuracy.
203	Extremely bad local integrand behaviour.
	This causes a very strong subdivision around one (or more) points of the interval. The same advice applies as in the case `error%code` = 201.
204	The requested accuracy cannot be achieved.
	The extrapolation does not increase the accuracy satisfactorily; the result returned is the best which can be obtained. The same advice applies as in the case `error%code` = 201.
205	The integral is probably divergent, or slowly convergent.
	Note that divergence can also occur with any other value of `error%code` when `error%level` = 2.
206	Bad integration behaviour occurs within one or more of the intervals C_k.
	The location and type of the difficulty involved can be determined from the optional array argument `intvl_error_code` (see below).

207 Maximum number of intervals C_k (= max_num_intvl) allowed has been achieved.

Increase the value of the optional argument max_num_intvl to allow more cycles (see below).

208 The extrapolation table does not converge to the required accuracy over the intervals C_k.

The extrapolation table constructed for convergence acceleration of the series formed by the integral contribution over the interval C_k, does not converge to the required accuracy.

In the cases error%code = 206, 207, or 208, the following values of intvl_error_code(k) give additional information about the cause of the error.

201 Maximum number of subdivisions allowed has been achieved on the kth interval.

202 Occurrence of round-off error prevents the requested accuracy imposed on the kth interval from being achieved.

203 Extremely bad integrand behaviour occurs at some points of the kth interval.

204 The integration procedure over the kth interval does not converge (to within the required accuracy) due to the round-off in the extrapolation procedure invoked on this interval. It is assumed that the result on this interval is the best which can be obtained.

205 The integral over kth interval is probably divergent or slowly convergent. It must be noted that divergence can occur with any other value of intvl_error_code(k).

5 Examples of Usage

A complete example of the use of this procedure appears in Example 2 of this module document.

6 Further Comments

6.1 Algorithmic Detail

The procedure is a modified version of the QUADPACK procedure QAWFE (Piessens *et al.* [5]). It is an adaptive procedure, designed to integrate a function of the form $w(x)g(x)$ over a semi-infinite interval, where $w(x)$ is either $\sin(\omega x)$ or $\cos(\omega x)$. Over successive intervals

$$C_k = [a + (k-1)c, a + kc], \quad k = 1, 2, \ldots, n$$

integration is performed by the same algorithm as is used by nag_quad_1d_wt_trig. The intervals C_k are of constant length

$$c = \frac{(2[|\ \omega\ |])\pi}{|\ \omega\ |},$$

where $[|\ \omega\ |]$ represents the largest integer less than or equal to $|\ \omega\ |$. Since c equals an odd number of half periods, the integral contributions over succeeding intervals will alternate in sign when the function g is positive and monotonically decreasing over $[a, \infty)$. The algorithm, described in Piessens *et al.* [5], incorporates a global acceptance criterion (as defined by Malcolm and Simpson [2]) together with the ϵ-algorithm (Wynn [6]) to perform extrapolation. The local error estimation is described in Piessens *et al.* [5].

If $\omega = 0$ and trig_wt = 'c', the procedure uses the same algorithm as nag_quad_1d_inf_gen (with rel_acc = 0.0).

In contrast to the other procedures in this chapter, this procedure works with a specified *absolute* accuracy (abs_acc). Over the interval C_k it attempts to satisfy the absolute accuracy requirement

$$e_k = U_k \times \texttt{abs\_acc},$$

where $U_k = (1-p)p^{k-1}$, for $k = 1, 2, \ldots$ and $p = 0.9$.

However, when difficulties occur during the integration over the kth subinterval C_k such that the error flag intvl_error_code(k) is non-zero, the accuracy requirement over subsequent intervals is relaxed. See Piessens *et al.* [5] for more details.

6.2 Accuracy

The procedure cannot guarantee, but in practice usually achieves, the following accuracy:

$$| I - \texttt{result} | \leq \texttt{abs\_acc},$$

where abs_acc is the specified absolute accuracy, or its default value. The optional output argument abs_err normally satisfies

$$| I - \texttt{result} | \leq \texttt{abs\_err} \leq \texttt{abs\_acc}.$$

Example 1: Integration of a function over a semi-infinite interval

The integral

$$\int_0^\infty \frac{1}{(x+1)\sqrt{x}}\,dx$$

is computed using the procedure `nag_quad_1d_inf_gen`.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
MODULE quad_1d_inf_ex01_mod

  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)

CONTAINS

  FUNCTION f(x)

    ! .. Implicit None Statement ..
    IMPLICIT NONE
    ! .. Intrinsic Functions ..
    INTRINSIC SIZE, SQRT
    ! .. Array Arguments ..
    REAL (wp), INTENT (IN) :: x(:)
    ! .. Function Return Value ..
    REAL (wp) :: f(SIZE(x))
    ! .. Executable Statements ..

    f = 1.0_wp/((x+1.0_wp)*SQRT(x))

  END FUNCTION f

END MODULE quad_1d_inf_ex01_mod

PROGRAM nag_quad_1d_inf_ex01

  ! Example Program Text for nag_quad_1d_inf
  ! NAG fl90, Release 3. NAG Copyright 1997.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_out
  USE nag_quad_1d_inf, ONLY : nag_quad_1d_inf_gen
  USE quad_1d_inf_ex01_mod, ONLY : wp, f
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Local Scalars ..
  REAL (wp) :: a, result
  CHARACTER (1) :: inf_limit
  ! .. Executable Statements ..

  WRITE (nag_std_out,*) 'Example Program Results for nag_quad_1d_inf_ex01'
```

```
    a = 0.0_wp
    inf_limit = 'upper'

    CALL nag_quad_1d_inf_gen(f,a,inf_limit,result)

    WRITE (nag_std_out,'(/,1X,A,F10.4)') &
     'a  - lower limit of integration = ', a
    WRITE (nag_std_out,'(1X,A,A)') &
     'inf_limit  - infinite limit of integration = ', inf_limit
    WRITE (nag_std_out,'(1X,A,F9.5)') &
     'result - approximation to the integral =', result

  END PROGRAM nag_quad_1d_inf_ex01
```

2 Program Data

None.

3 Program Results

```
Example Program Results for nag_quad_1d_inf_ex01

a  - lower limit of integration =     0.0000
inf_limit  - infinite limit of integration = u
result - approximation to the integral =  3.14159
```

Example 2: Computation of a cosine transform over a semi-infinite interval

The integral

$$\int_0^\infty \frac{1}{\sqrt{x}} \cos(\pi x/2)\, dx$$

is computed using the procedure `nag_quad_1d_inf_wt_trig`.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
MODULE quad_1d_inf_ex02_mod

  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)

CONTAINS

  FUNCTION g(x)

    ! .. Implicit None Statement ..
    IMPLICIT NONE
    ! .. Intrinsic Functions ..
    INTRINSIC SIZE, SQRT
    ! .. Array Arguments ..
    REAL (wp), INTENT (IN) :: x(:)
    ! .. Function Return Value ..
    REAL (wp) :: g(SIZE(x))
    ! .. Executable Statements ..

    WHERE (x>0.0_wp)
      g = 1.0_wp/SQRT(x)
    ELSEWHERE
      g = 0.0_wp
    END WHERE

  END FUNCTION g

END MODULE quad_1d_inf_ex02_mod

PROGRAM nag_quad_1d_inf_ex02

  ! Example Program Text for nag_quad_1d_inf
  ! NAG fl90, Release 3. NAG Copyright 1997.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_out
  USE nag_math_constants, ONLY : nag_pi
  USE nag_quad_1d_inf, ONLY : nag_quad_1d_inf_wt_trig
  USE quad_1d_inf_ex02_mod, ONLY : wp, g
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Local Scalars ..
  REAL (wp) :: a, omega, pi, result
```

```
    CHARACTER (1) :: trig_wt
    ! .. Executable Statements ..

    WRITE (nag_std_out,*) 'Example Program Results for nag_quad_1d_inf_ex02'

    pi = nag_pi(0.0_wp)
    a = 0.0_wp
    omega = 0.5_wp*pi
    trig_wt = 'cosine'

    CALL nag_quad_1d_inf_wt_trig(g,a,omega,trig_wt,result)

    WRITE (nag_std_out,'(/,1X,A,F10.4)') &
     'a  - lower limit of integration = ', a
    WRITE (nag_std_out,*) 'b  - upper limit of integration = infinity'
    WRITE (nag_std_out,'(1X,A,F9.5)') &
     'result - approximation to the integral =', result

  END PROGRAM nag_quad_1d_inf_ex02
```

2 Program Data

None.

3 Program Results

```
 Example Program Results for nag_quad_1d_inf_ex02

 a  - lower limit of integration =     0.0000
 b  - upper limit of integration = infinity
 result - approximation to the integral =  1.00000
```

Additional Examples

Not all example programs supplied with NAG *fl*90 appear in full in this module document. The following additional examples, associated with this module, are available.

`nag_quad_1d_inf_ex03`

Computation of a cosine transform over a semi-infinite interval with specified accuracy.

References

[1] De Doncker E (1978) An adaptive extrapolation algorithm for automatic integration *SIGNUM Newsl.* **13 (2)** 12–18

[2] Malcolm M A and Simpson R B (1976) Local versus global strategies for adaptive quadrature *ACM Trans. Math. Software* **1** 129–146

[3] Piessens R (1973) An algorithm for automatic integration *Angew. Inf.* **15** 399–401

[4] Piessens R and Branders M (1975) Algorithm 002. Computation of oscillating integrals *J. Comput. Appl. Math.* **1** 153–164

[5] Piessens R, De Doncker-Kapenga E, Überhuber C and Kahaner D (1983) *QUADPACK, A Subroutine Package for Automatic Integration* Springer-Verlag

[6] Wynn P (1956) On a device for computing the $e_m(S_n)$ transformation *Math. Tables Aids Comput.* **10** 91–96