$2q+2$. Romberg integration is normally given as the extrapolation to the limit of the trapezoid rule. Let

$$T_0^{(k)} = h\left(\frac{1}{2}f_0 + f_1 + \cdots + f_{2^k-1} + \frac{1}{2}f_{2^k}\right), \text{ and } T_m^{(k)} = \frac{4^m T_{m-1}^{(k+1)} - T_{m-1}^{(k)}}{4^m - 1},$$

then the Romberg quadrature rule gives

$$\int_{t_0}^{t_n} f(t)\,dt = T_m^{(k)} \approx \sum_{j=0}^{n} w_j f(t_j),$$

where $n = 2^q$, $m = (p-2)/2$, and $k = q-m$. The left and right endpoints must be in $t[0]$ and $t[n]$ respectively. The abscissas are returned in $t[0]$ to $t[n]$ and the corresponding weights in $w[0]$ to $w[n]$.

If $p = 2$ the Romberg rule is the same as the repeated trapezoid rule, and if $p = 4$, the same as the repeated Simpson rule.

For $n \leqq 128$ and $p \leqq 16$, the numerical integration of powers (less than p) of x on the interval [0, 1] gave answers correct to one round off error in an 11-digit mantissa.

REFERENCE: Bauer, F. L., Rutishauser, H., and Stiefel, E. New aspects in numerical quadrature. *Proc. of Symp. in Appl. Math.*, Vol. 15: High speed computing and experimental arithmetic. Amer. Math. Soc., Providence, R. I., 1963, pp. 199–218;

```
begin integer i, j, m, m1, m4, s;
  real h, ci; real array c[0: (p − 2)/2];
  h := (t[n] − t[0])/n; w[0] := w[n] := 0;
  for i := n−1 step −1 until 1 do
    begin w[i] := c[i] := 0; t[i] := i × h + t[0] end;
  m := (p − 2)/2; c[0] := 1.0; s := m4 := 1; c[n] := 0;
  if m > ln(n)/ln(2) then m := ln(n)/ln(2);
  for j := 1 step 1 until m do
  begin m4 := 4 × m4; m1 := m4 − 1;
    for i := j step −1 until 1 do
      c[i] := (m4 × c[i] − c[i − 1])/m1;
    c[0] := c[0] × (m4/m1);
  end;
  for i := 0 step 1 until m do
  begin ci := c[i] × s;
    for j := 0 step s until n do w[j] := w[j]+ci;
    s := 2 × s
  end;
  w[0] := w[n] := 0.5×w[0];
  for j := 0 step 1 until n do w[j] := w[j] × h;
end rombergrule
```

ALGORITHM 282
DERIVATIVES OF e^x/x, $\cos(x)/x$, AND $\sin(x)/x$* [S22]

WALTER GAUTSCHI (Recd. 19 Aug. 1965)
Argonne National Laboratory, Argonne, Ill.

* Work performed under the auspices of the U.S. Atomic Energy Commission. Author's present address is Purdue University.

```
procedure dsubn(x, nmax, d);
  value x, nmax; integer nmax; real x; array d;
comment This procedure generates the derivatives
```

$$d_n(x) = \frac{d^n}{dx^n}\left(\frac{e^x}{x}\right) \quad (n = 0, 1, 2, \cdots, nmax)$$

using the recurrence relation

$$d_n(x) = (e^x - n d_{n-1}(x))/x \qquad (n = 1, 2, 3, \cdots).$$

The results are stored in the array d. If $x = 0$, there is an error exit to a global label called *alarm*;

```
begin integer n; real e;
  if x = 0 then go to alarm;
  e := exp(x); d[0] := e/x;
  for n := 1 step 1 until nmax do
    d[n] := (e − n × d[n − 1])/x
end dsubn;
procedure csubn(x, nmax, c);
  value x, nmax; integer nmax; real x; array c;
comment This procedure obtains the derivatives
```

$$c_n(x) = \frac{d^n}{dx^n}\left(\frac{\cos x}{x}\right) \quad (n = 0, 1, 2, \cdots, nmax)$$

from the recurrence relation

$$c_n(x) = (\tau_n(x) - n c_{n-1}(x))/x \quad (n = 1, 2, 3, \cdots),$$

where $\{\tau_n(x)\}_{n=1}^{\infty} = \{-\sin x, -\cos x, \sin x, \cos x, -\sin x, \cdots\}$. The results are stored in the array c. If $x = 0$, there is an error exit to a global label called *alarm*;

```
begin integer n; array tau[1: 4];
  if x = 0 then go to alarm;
  tau[1] := −sin(x); tau[2] := −cos(x);
  tau[3] := −tau[1]; tau[4] := −tau[2];
  c[0] := tau[4]/x;
  for n := 1 step 1 until nmax do
    c[n] := (tau[n−4×((n−1) ÷ 4)] − n×c[n−1])/x
end csubn;
procedure ssubn(x, nmax, d, s);
  value x, nmax, d; integer nmax, d; real x; array s;
comment This procedure generates to d significant digits the
  derivatives
```

$$s_n(x) = \frac{d^n}{dx^n}\left(\frac{\sin x}{x}\right) \quad (n = 0, 1, 2, \cdots, nmax),$$

and stores the results in the array s. The method of computation is based on the recurrence relation

$$s_n(x) = (\sigma_n(x) - n s_{n-1}(x))/x \qquad (n = 1, 2, 3, \cdots),$$

where $\{\sigma_n(x)\}_{n=1}^{\infty} = \{\cos x, -\sin x, -\cos x, \sin x, \cos x, \cdots\}$. The recurrence relation is applied in forward direction as long as $n \leqq |x|$, and in backward direction for the remaining values of n, starting with an appropriately large $n = \nu$. A detailed discussion of the method will be published elsewhere. It is assumed that a global real procedure $t(y)$ is available, which evaluates the inverse function $t = t(y)$ of $y = t \ln t$ to low accuracy for $y \geqq 0$. (See W. Gautschi, Algorithm 236, Bessel functions of the first kind, *Comm. ACM 7* (Aug. 1964), 479 Gautschi, W. Computation of successive derivatives of $f(z)/z$, in press;

```
begin integer n, n0, nu; real x1, d1, s1; array sigma [1: 4];
  x1 := abs(x);
  sigma [1] := cos(x); sigma [2] := −sin(x);
  sigma [3] := −sigma [1]; sigma [4] := −sigma [2];
  n0 := entier (x1); s[0] := if x ≠ 0 then sigma [4]/x else 1;
  for n := 1 step 1 until if n0 ≦ nmax then n0 else nmax do
    s[n] := (sigma[n − 4 × ((n − 1) ÷ 4)] − n × s[n − 1])/x;
  if n0 < nmax then
  begin
    s1 := 0; d1 := 2.3026 × d + .6931;
    nu := if nmax ≦ 2.7183 × x1 0 then
      1 + entier (2.7183 × x1 × t(.36788 × d1/x1)) else
        1 + entier (nmax × t(d1/nmax));
    for n := nu step −1 until n0+2 do
    begin
      s1 := (sigma[n − 4 × ((n − 1) ÷ 4)] − x × s1)/n;
      if n ≦ nmax + 1 then s[n−1] := s1
    end
  end
end ssubn
```

ALGORITHM 283
SIMULTANEOUS DISPLACEMENT OF POLYNOMIAL ROOTS IF REAL AND SIMPLE [C2]

IMMO O. KERNER (Recd. 8 Sept. 1965 and 12 Nov. 1965)
Rechenzentrum Universitaet Rostock

```
procedure Prrs (A, X, n, eps); value n, eps;
  integer n; real eps; array A, X;
comment Prrs (polynomial roots real simple) computes the n
  roots X of the polynomial equation
```

$$A_n x^n + A_{n-1}x^{n-1} + \cdots + A_0 = 0$$

```
  simultaneously. On entry the array X contains the vector of
  initial approximations to the roots and on exit it contains the
  vector of improved approximations to the roots. The initial
  approximations must be distinct. Accuracy is specified by means
  of a parameter eps. Iteration is continued until the Euclidean
  norm of the correction vector does not exceed eps. The con-
  vergence is quadratic;
begin integer i, k; real x, P, Q;
      eps := eps ↑ 2;
W:  Q := 0;
      for i := 1 step 1 until n do
  begin  x := P := A[n];
    for k := 1 step 1 until n do
    begin x := x × X[i] + A[n − k];
      if k ≠ i then P := P × (X[i]−X[k])
    end;
    X[i] := X[i]−x/P;
    Q := Q + (x/P) ↑ 2
  end;
  if Q > eps then go to W
end
```

CERTIFICATION OF ALGORITHM 9 [D2]
RUNGE-KUTTA INTEGRATION [P. Naur et al., *Comm. ACM 3* (May 1960), 318]

HENRY C. THACHER, JR. (Recd. 28 July 1964 and 22 Nov. 1965)
Argonne National Laboratory, Argonne, Ill.

Algorithm 9 was transcribed into the hardware representation for CDC 3600 ALGOL and run successfully. The following procedure was used for the global procedure *comp*:

```
real procedure comp (a, b, c); value a, b, c; real a, b, c;
begin integer AE, BE, CE;
  integer procedure expon(x); real x;
  comment This function produces the base 10 exponent of x;
  expon := if x = 0 then −999 else
    entier (.4342944819 × ln(abs(x)) + 1);
  comment The number −999 may be replaced by any number
    less than the exponent of the smallest positive number handled
    by the particular machine used, for this algorithm assumes
    that true zero has an exponent smaller than any nonzero
    floating-point number. Users implementing real procedure
    comp by machine code should make sure that this condition
    is satisfied by their program;
  AE := expon(a); BE := expon(b); CE := expon(c);
  if AE < BE then AE := BE; if AE < CE then AE := CE;
  comp := abs(a − b)/10 ↑ AE
end
```

This has the advantage of machine independence, but is highly inefficient compared to machine code.

The procedure was tested using the two following procedures for *FKT*:

```
procedure FKT (X, Y, N, Z); real X; integer N; array
  Y, Z;
comment (dy1/dx) = z1 = y2 , (dy2/dx) = z2 = −y1 . With
  y1(0) = 0, y2(0) = 1, the solution is y1 = sin x, y2 = cos x;
begin Z [1] := Y [2]; Z [2] := −Y [1] end;
procedure FKT (X, Y, N, Z); real X; integer N; array
  Y, Z;
comment (dy1/dx) = 1 + y1². For y1(0) = 0, y(x) = tan x;
  Z [1] := 1+Y[1]↑2;
```

The *RK* procedure was used to integrate the differential equations represented by the first *FKT* procedure from $x = 0(0.5)7.0$, with $eps = eta = 10^{-6}$, and with $y_1(0) = 0$, $y_2(0) = 1$. The actual step size h was .0625 for most of the range, but was reduced to .03125 in the neighborhood of $x = k\pi/2$, where one or the other of the solutions is small.

The computed solutions at $x = 7.0$ were: $y_1 = 6.5698602746 \times 10^{-1}$, $y_2 = 7.5390270246 \times 10^{-1}$, with errors -5.71×10^{-7} and 4.48×10^{-7}, respectively.

Results for the second differential equation are summarized in Table I below.

The efficiency of the procedure would be increased slightly on most computers by changing the type of the **own** variable *s* from **real** to **integer.**

The error is estimated by comparing the results of successive pairs of steps with that of a single double step. This is somewhat more time-consuming than the Kutta-Merson process presented in Algorithm 218 [*Comm. ACM 6* (Dec. 1963) 737–8]. However, the criterion for step-size variation in Algorithm 9 which effectively applies an approximate relative error criterion, *eps*, for $|y| > eta$, and an absolute error criterion $eta \times eps$, for $|y| < eta$, appears superior when the solution fluctuates in magnitude.

REMARK ON ALGORITHM 218 [D2]
KUTTA-MERSON [Phyllis M. Lukehart, *Comm. ACM 6* (Dec. 1963), 737]

G. BAYER (Recd. 25 Oct. 1965)
Technische Hochschule, Braunschweig, Germany

Successive calls of *Kutta Merson* with *first* ≡ **false** do not reach the upper bound $t+h$ if the interval h is unequal to the interval h of the first call with *first* ≡ **true**.

Proposed correction:

1) declaration **real** *hc*, instead of **own real** *hc*;
2) **if** *first* **then begin for** *i* := **1 step 1 until** *n* **do** *y0*[*i*] := *y*[*i*];
   hc := *h*; *ploc* := 1; *first* := **false**
   end else *hc* := *h*/*ploc*;
   instead of **if** *first* **then begin** ⋯ **end;**

TABLE I [ALG. 9]

		$x = 0.5$			$x = 1.0$			$x = 1.5$		
	η	h_{min}	*Absolute error*	*Relative error*	h_{min}	*Absolute error*	*Relative error*	h_{min}	*Absolute error*	*Relative error*
10^{-7}	10^{-3}	.03125	-1×10^{-9}	-2×10^{-9}	.03125	9×10^{-8}	6×10^{-8}	.00390625	-1×10^{-6}	-8×10^{-8}
10^{-5}	10^{-3}	.125	-5×10^{-7}	-9×10^{-7}	.0625	8×10^{-7}	5×10^{-7}	.0078125	-2×10^{-4}	-1×10^{-5}
10^{-3}	10^{-3}	.25	-1×10^{-5}	-2×10^{-5}	.25	-2×10^{-4}	-1×10^{-4}	.03125	-3×10^{-2}	-2×10^{-3}