

```

> restart;
> Digits:=20;
                                Digits := 20
> libname:="/nalib",libname;
    libname := "/nalib", "//Library/Frameworks/Maple.framework/Versions/13/lib"
> with(numanal);
[SOR, SOR_dir, adaptq, adaptq_dir, bezier, bezier_dir, bisection, bisection_dir, chop, chop_dir,
 clamped_spline, clamped_spline_dir, divided_diff, divided_diff_dir, extrap, extrap_dir,
 falseposition, falseposition_dir, fixedpoint, fixedpoint_dir, gaussseidel, gaussseidel_dir, hermite,
 hermite_dd, hermite_dd_dir, hermite_dir, horner, horner_dir, jacobi, jacobi_dir, muller,
 muller_dir, natural_spline, natural_spline_dir, newton, newton_dir, romberg, romberg_dir,
 secant, secant_dir, steffensen, steffensen_dir]
> numanal[extrap] := proc(f::algebraic,a::numeric,b::numeric,
 alpha::numeric,tol::positive,hmax::numeric,hmin::numeric)
 local F, OK, A, B, ALPHA, TOL, HMIN, HMAX, FLAG, NAME, NK, J, I,
 TO, WO, H, DONE, Q, K, NFLAG, HK, T, W2, W3, M, W1, Y, V;
> F := unapply(f,t,y);
> A := evalf(a);
> B := evalf(b);
> if A >= B then
> ERROR("Left endpoint must be less than right endpoint.");
fi;
> ALPHA := evalf(alpha);
> TOL := evalf(tol);
> if TOL <= 0 then
> ERROR("Tolerance must be positive.");
fi;
> HMIN := evalf(hmin);
> HMAX := evalf(hmax);
> if HMIN >= HMAX or HMIN <= 0 then
> ERROR("Minimum mesh spacing must be a positive real number and
less than the maximum mesh spacing.");
fi;
> printf(`GRAGG EXTRAPOLATION\n\n`);
> printf(`      t[i]          w[i]          h[i]          K\n`);
> # STEP 1 */
> NK[0] := 2;
> NK[1] := 4;
> for J from 1 to 3 do
> I := 2*J;
> NK[I] := 3*NK[I-1]/2;
> NK[I+1] := 2*NK[I-1];
> od;
> # STEP 2 */
> TO := A;
> WO := ALPHA;
> H := HMAX;
> # DONE is used in place of FLAG to exit the loop in Step 4 */
> DONE := FALSE;
> # STEP 3 */
> for I from 1 to 7 do
> for J from 1 to I do
> Q[I-1,J-1] := (NK[I]*1/NK[J-1])*(NK[I]*1/NK[J-1]);

```

```

> od;
> od;
> # STEP 4 */
> while DONE = FALSE do
> # STEP 5 */
> K := 1;
> # when desired accuracy achieved, NFLAG is set to 1 */
> NFLAG := 0;
> # STEP 6 */
> while K <= 8 and NFLAG = 0 do
> # STEP 7 */
> HK := H/NK[K-1];
> T := T0;
> W2 := W0;
> # Euler first step *
> W3 := W2+HK*F(T, W2);
> T := T0+HK;
> # STEP 8 */
> M := NK[K-1]-1;
> for J from 1 to M do
> W1 := W2;
> W2 := W3;
> # midpoint method */
> W3 := W1+2*HK*F(T, W2);
> T := T0+(J+1)*HK;
> od;
> # STEP 9 */
> # endpoint correction to compute Y(K,1) */
> Y[K] := (W3+W2+HK*F(T, W3))/2;
> # STEP 10 */
> # NOTE: Y(K-1)=Y(K-1,1), Y(K-2)=Y(K-2,2),..., */
> # Y(1)=Y(K-1,K-1) since only previous row of table */
> # is saved */
> if K >= 2 then
> # STEP 11 */
> J := K;
> # save Y(K-1,K-1) */
> V := Y[1];
> # STEP 12 */
> while J >= 2 do
> # extrapolation to compute */
> # Y(J-1) := Y(K,K-J+2) */
> Y[J-1] := Y[J]+(Y[J]-Y[J-1])/(Q[K-2,J-2]-1);
> J := J-1;
> od;
> # STEP 13 */
> if abs(Y[1] - V) <= TOL then
> NFLAG := 1;
> fi;
> # Y(1) accepted as new w */
> fi;
> # STEP 14 */
> K := K+1;
> od;
> # STEP 15 */
> K := K-1;
> # STEP 16 */
> if NFLAG = 0 then
> # STEP 17 */
> # new value for w rejected, decrease H */
> H := H/2;
> # STEP 18 */

```

```

> if H < HMIN then
> printf(`HMIN exceeded\n`);
> DONE := TRUE;
> fi;
> else
> # STEP 19 */
> # new value for w accepted */
> W0 := Y[1];
> T0 := T0 + H;
> printf(`%8.4f %15.10f %8.4f %6d\n`, T0, W0, H, K);
> # STEP 20 */
> # increase H if possible */
> if T0 >= B then
> DONE := TRUE;
> # Procedure completed successfully.
> else
> if T0 + H > B then
> H := B - T0;
> # Terminate at t = B.
> else
> if K <= 3 then
> H := 2*H;
> fi;
> fi;
> fi;
> if H > HMAX then
> H := H/2;
> fi;
> fi;
> od;
> # STEP 21 */
> end;

```

Warning, imaginary unit `I` used as a local variable in procedure numanal[extrap]

numanal_{extrap} := **proc**(*f*::algebraic, *a*::numeric, *b*::numeric, *alpha*::numeric, *tol*::positive, *hmax*::

numeric, *hmin*::numeric)

local *F*, *OK*, *A*, *B*, *ALPHA*, *TOL*, *HMIN*, *HMAX*, *FLAG*, *NAME*, *NK*, *J*, *I*, *T0*, *W0*, *H*, *DONE*, *Q*,
K, *NFLAG*, *HK*, *T*, *W2*, *W3*, *M*, *W1*, *Y*, *V*;

F := unapply(*f*, *t*, *y*);

A := evalf(*a*);

B := evalf(*b*);

if *B* <= *A* **then** ERROR("Left endpoint must be less than right endpoint.") **end if**;

ALPHA := evalf(*alpha*);

TOL := evalf(*tol*);

if *TOL* <= 0 **then** ERROR("Tolerance must be positive.") **end if**;

HMIN := evalf(*hmin*);

HMAX := evalf(*hmax*);

if *HMAX* <= *HMIN* **or** *HMIN* <= 0 **then**

```

        ERROR("Minimum mesh spacing must be a positive real number and less than the maximum
        mesh spacing.")
    end if;
    printf('GRAGG EXTRAPOLATION
           '); printf(' t[i]    w[i]    h[i]    K
);
NK[0] := 2;
NK[1] := 4;
for J to 3 do I:= 2 * J, NK [I] := 3/2 * NK [I - 1]; NK [I + 1] := 2 * NK [I - 1] end do;
T0 := A;
W0 := ALPHA;
H := HMAX;
DONE := FALSE;
for I to 7 do
    for J to I do
        Q[I - 1, J - 1] := NK [I] * 1 * NK [I] / (NK [J - 1] * NK [J - 1])
    end do
end do;
while DONE = FALSE do
    K := 1;
    NFLAG := 0;
    while K <= 8 and NFLAG = 0 do
        HK := H / NK [K - 1];
        T := T0;
        W2 := W0;
        W3 := W2 + HK * F (T, W2);
        T := T0 + HK;
        M := NK [K - 1] - 1;
        for J to M do
            W1 := W2; W2 := W3; W3 := W1 + 2 * HK * F (T, W2); T := T0 + (J + 1) * HK
        end do;

```

```

Y[K] := 1/2 * W3 + 1/2 * W2 + 1/2 * HK * F(T, W3);
if 2 <= K then
    J := K;
    V := Y[1];
    while 2 <= J do
        Y[J-1] := Y[J] + (Y[J] - Y[J-1]) / (Q[K-2, J-2] - 1); J :=
        J - 1
    end do;
    if abs(Y[1] - V) <= TOL then NFLAG := 1 end if
end if;
    K := K + 1
end do;
    K := K - 1;
    if NFLAG = 0 then H := 1/2 * H; if H < HMIN then printf( 'HMIN exceeded
); DONE := TRUE end if else W0 := Y[1]; T0 := T0 + H; printf( '%8.4f%15.10f%8.4f%6d
', T0, W0, H, K);
if B <= T0 then
    DONE := TRUE
else
    if B < T0 + H then H := B - T0 else if K <= 3 then H := 2 * H end if end if
end if;
if HMAX < H then H := 1/2 * H end if
end if
end do
end proc

```

```

> numanal[extrap](y-t^2+1,0,2,0.5,10^(-10),0.25,0.01);
GRAGG EXTRAPOLATION

```

t[i]	w[i]	h[i]	K
0.2500	0.9204872917	0.2500	5
0.5000	1.4256393646	0.2500	5
0.7500	2.0039999917	0.2500	5
1.0000	2.6408590858	0.2500	5
1.2500	3.3173285213	0.2500	4
1.5000	4.0091554648	0.2500	5
1.7500	4.6851986620	0.2500	5
2.0000	5.3054719505	0.2500	5

```

> numanal[extrap_dir]:=proc()
printf('extrap returns a table of values for the ODE.\n\n');
printf('t[i] is the t value, w[i] the approximation at y(t[i]),
\n\n');

```

```

printf(`h[i] the stepsize, and k is the level of extrapolation.
\n\n`);
printf(`The arguments for extrap are:\n`);
printf(`(1)function expression in t and y\n`);
printf(`(2)lefthand endpoint\n`);
printf(`(3)righthand endpoint\n`);
printf(`(4)initial value\n`);
printf(`(5)tolerance\n`);
printf(`(6)maximum stepsize\n`);
printf(`(7)maximum stepsize\n`);
end;

```

numanal_{extrap_dir} := proc () printf (`extrap returns a table of values for the ODE.

); printf (`t[i] is the t value, w[i] the approximation at y(t[i]),

); printf (`h[i] the stepsize, and k is the level of extrapolation.

); printf (`The arguments for extrap are:

); printf (`(1)function expression in t and y

); printf (`(2)lefthand endpoint

); printf (`(3)righthand endpoint

); printf (`(4)initial value

); printf (`(5)tolerance

); printf (`(6)maximum stepsize

); printf (`(7)maximum stepsize

) end proc

```
> numanal[extrap_dir]();
```

extrap returns a table of values for the ODE.

t[i] is the t value, w[i] the approximation at y(t[i]),

h[i] the stepsize, and k is the level of extrapolation.

The arguments for extrap are:

(1)function expression in t and y

(2)lefthand endpoint

(3)righthand endpoint

(4)initial value

(5)tolerance

(6)maximum stepsize

(7)maximum stepsize

```
> savelibName:="/nalib";
```

savelibName := "/nalib"

```
> savelib(numanal);
```

```
> restart;
```

```
> libname:="/nalib",libname;
```

libname := "/nalib", "//Library/Frameworks/Maple.framework/Versions/13/lib"

```
> with(numanal);  
[SOR, SOR_dir, adaptq, adaptq_dir, bezier, bezier_dir, bisection, bisection_dir, chop, chop_dir,  
clamped_spline, clamped_spline_dir, divided_diff, divided_diff_dir, extrap, extrap_dir,  
falseposition, falseposition_dir, fixedpoint, fixedpoint_dir, gaussseidel, gaussseidel_dir, hermite,  
hermite_dd, hermite_dd_dir, hermite_dir, horner, horner_dir, jacobi, jacobi_dir, muller,  
muller_dir, natural_spline, natural_spline_dir, newton, newton_dir, romberg, romberg_dir,  
secant, secant_dir, steffensen, steffensen_dir]
```