```
with respect to the parameters a at point xx. Also they
must define the types
TYPE
    glndata = ARRAY [1..ndata] OF real;
    glmma = ARRAY [1..mma] OF real;
    gllista = ARRAY [1..mma] OF integer;
    glnalbynal = ARRAY [1..nalp,1..nalp] OF real;
in the main routine *)
VAR
    k,j,i: integer; ymod,wt,sig2i,dy: real; dyda: glmma;
BEGIN
    FOR j := 1 TO mfit DO BEGIN
        FOR k := 1 TO j DO BEGIN
            alpha[j,k] := 0.0 END;
        beta[j] := 0.0 END;
    chisq := 0.0;
    FOR i := 1 TO ndata DO BEGIN
        funcs(x[i],a,ymod,dyda,mma); sig2i := 1.0/(sig[i]*sig[i]);
        dy := y[i]-ymod;
        FOR j := 1 TO mfit DO BEGIN
            wt := dyda[lista[j]]*sig2i;
            FOR k := 1 TO j DO BEGIN
                alpha[j,k] := alpha[j,k]+wt*dyda[lista[k]] END;
            beta[j] := beta[j]+dy*wt END;
        chisq := chisq+dy*dy*sig2i END;
    FOR j := 2 TO mfit DO BEGIN
        FOR k := 1 TO j-1 DO BEGIN
            alpha[k,j] := alpha[j,k] END END
END;

PROCEDURE fgauss(x: real; a: glnparam; VAR y: real;
        VAR dyda: glnparam; na: integer);
(* Programs using routine FGAUSS must define the type
TYPE
    glnparam = ARRAY [1..na] OF real;
in the main routine. *)
VAR
    i,ii: integer; fac,ex,arg: real;
BEGIN
    y := 0.0;
    FOR ii := 1 TO (na DIV 3) DO BEGIN
        i := 3*ii-2; arg := (x-a[i+1])/a[i+2]; ex := exp(-sqr(arg));
        fac := a[i]*ex*2.0*arg; y := y+a[i]*ex; dyda[i] := ex;
        dyda[i+1] := fac/a[i+2]; dyda[i+2] := fac*arg/a[i+2] END
END;
```

Robust Estimation

```
PROCEDURE medfit(x,y: glndata; ndata: integer; VAR a,b,abdev: real);
(* Programs using routine MEDFIT must define the type
TYPE
    glndata = ARRAY [1..ndata] OF real;
in the main routine. MEDFIT also assumes that the instructions
at the beginning of ROFUNC have been carried out so that arrays
x,y and arr, and real variables aa and abdevt exist as globally
defined variables. *)
LABEL 3;
VAR
    j: integer; sy,sxy,sxx,sx,sigb,f2,f1,f: real; del,chisq,bb,b2,b1: real;
BEGIN
    sx := 0.0; sy := 0.0; sxy := 0.0; sxx := 0.0;
    FOR j := 1 TO ndata DO BEGIN
```



```
        sx := sx+x[j]; sy := sy+y[j]; sxy := sxy+x[j]*y[j];
        sxx := sxx+sqr(x[j]) END;
    del := ndata*sxx-sx*sx; aa := (sxx*sy-sx*sxy)/del;
    bb := (ndata*sxy-sx*sy)/del; chisq := 0.0;
    FOR j := 1 TO ndata DO BEGIN
        chisq := chisq+sqr(y[j]-(aa+bb*x[j])) END;
    sigb := sqrt(chisq/del); b1 := bb; f1 := rofunc(b1);
    IF (f1 >= 0.0) THEN BEGIN
        b2 := bb+abs(3.0*sigb) END
    ELSE BEGIN
        b2 := bb-abs(3.0*sigb) END;
    f2 := rofunc(b2);
    WHILE ((f1*f2) > 0.0) DO BEGIN
        bb := 2.0*b2-b1; b1 := b2; f1 := f2; b2 := bb; f2 := rofunc(b2)
    END;
    sigb := 0.01*sigb;
    WHILE (abs(b2-b1) > sigb) DO BEGIN
        bb := 0.5*(b1+b2);
        IF ((bb = b1) OR (bb = b2)) THEN GOTO 3;
        f := rofunc(bb);
        IF ((f*f1) >= 0.0) THEN BEGIN
            f1 := f; b1 := bb END
        ELSE BEGIN
            f2 := f; b2 := bb END END;
3:  a := aa;
    b := bb; abdev := abdevt/ndata
END;

FUNCTION rofunc(b: real): real;
(* Programs using ROFUNC must declare
CONST
    ndata = [the number of data points]
VAR
    aa,abdevt: real;
    x,y,arr: ARRAY [1..ndata] OF real;
in the main routine. *)
VAR
    d,sum: real; j,n1,nmh,nml: integer;
BEGIN
    n1 := ndata+1;
    nml := n1 DIV 2;
    nmh := n1-nml;
    FOR j := 1 TO ndata DO arr[j] := y[j]-b*x[j];
    sort(ndata,arr); aa := 0.5*(arr[nml]+arr[nmh]); sum := 0.0;
    abdevt := 0.0;
    FOR j := 1 TO ndata DO BEGIN
        d := y[j]-(b*x[j]+aa); abdevt := abdevt+abs(d);
        IF (d > 0.0) THEN sum := sum+x[j] ELSE sum := sum-x[j]
    END;
    rofunc := sum
END;
```

Chapter 15. Integration of Ordinary Differential Equations

Runge-Kutta Method

```
PROCEDURE rk4(y,dydx: glnarray; n: integer; x,h: real; VAR yout: glnarray);
(* Programs using routine RK4 must provide a
PROCEDURE derivs(x:real; y:glnarray; VAR dydx:glnarray);
which returns the derivatives dydx at location x, given both x and the
```

```
function values y. The calling program must also define the types
TYPE
    glnarray = ARRAY [1..nvar] OF real;
where nvar is the number of variables y. *)
VAR
    i: integer; xh,hh,h6: real; dym,dyt,yt: glnarray;
BEGIN
    hh := h*0.5; h6 := h/6.0; xh := x+hh;
    FOR i := 1 TO n DO BEGIN
        yt[i] := y[i]+hh*dydx[i] END;
    derivs(xh,yt,dyt);
    FOR i := 1 TO n DO BEGIN
        yt[i] := y[i]+hh*dyt[i] END;
    derivs(xh,yt,dym);
    FOR i := 1 TO n DO BEGIN
        yt[i] := y[i]+h*dym[i]; dym[i] := dyt[i]+dym[i] END;
    derivs(x+h,yt,dyt);
    FOR i := 1 TO n DO BEGIN
        yout[i] := y[i]+h6*(dydx[i]+dyt[i]+2.0*dym[i]) END
END;

 PROCEDURE rkdumb(vstart: glnarray; nvar: integer; x1,x2: real; nstep: integer);
 (* Programs using routine RKDUMB must provide a
 PROCEDURE derivs(x:real; v:glnarray; VAR dvdx:glnarray);
 which returns the derivatives dvdx at location x, given both x and the
 function values v. They must also define the type
 TYPE
     glnarray = ARRAY [1..nvar] OF real;
 where nvar is the number of functions. They must also
 declare the variables
 VAR
     xx: ARRAY [1..200] OF real;
     y: ARRAY [1..nvar,1..200] OF real;
 in the main routine. *)
 VAR
     k,i: integer; x,h: real; v,vout,dv: glnarray;
 BEGIN
     FOR i := 1 TO nvar DO BEGIN
         v[i] := vstart[i]; y[i,1] := v[i] END;
     xx[1] := x1; x := x1; h := (x2-x1)/nstep;
     FOR k := 1 TO nstep DO BEGIN
         derivs(x,v,dv); rk4(v,dv,nvar,x,h,vout);
         IF (x+h = x) THEN BEGIN
             writeln('pause in routine RKDUMB');
             writeln('stepsize to small'); readln END;
         x := x+h; xx[k+1] := x;
         FOR i := 1 TO nvar DO BEGIN
             v[i] := vout[i]; y[i,k+1] := v[i] END END
 END;
```

Adaptive Stepsize Control for Runge-Kutta

```
PROCEDURE rkqc(VAR y,dydx: glarray; n: integer; VAR x: real;
        htry,eps: real; yscal: glarray; VAR hdid,hnext: real);
(* Programs using routine RKQC must provide a
PROCEDURE derivs(x:real; y:glnarray; VAR dydx:glnarray);
which returns the derivatives dydx at location x, given both x and the
function values y. They must also define the type
TYPE
    glarray = ARRAY [1..n] OF real;
in the main routine.    *)
LABEL 1;
```



```
CONST
    pgrow=-0.20; pshrnk=-0.25;
    fcor=0.06666666;    (* 1.0/15.0 *)
    one=1.0; safety=0.9; errcon=6.0e-4;
VAR
    i: integer; xsav,hh,h,temp,errmax: real; dysav,ysav,ytemp: glarray;
BEGIN
    xsav := x;
    FOR i := 1 TO n DO BEGIN
        ysav[i] := y[i]; dysav[i] := dydx[i] END;
    h := htry;
1:  hh := 0.5*h;
    rk4(ysav,dysav,n,xsav,hh,ytemp); x := xsav+hh; derivs(x,ytemp,dydx);
    rk4(ytemp,dydx,n,x,hh,y); x := xsav+h;
    IF (x = xsav) THEN BEGIN
        writeln('pause in routine RKQC');
        writeln('stepsize too small'); readln END;
    rk4(ysav,dysav,n,xsav,h,ytemp); errmax := 0.0;
    FOR i := 1 TO n DO BEGIN
        ytemp[i] := y[i]-ytemp[i]; temp := abs(ytemp[i]/yscal[i]);
        IF (errmax < temp) THEN errmax := temp
    END;
    errmax := errmax/eps;
    IF (errmax > one) THEN BEGIN
        h := safety*h*exp(pshrnk*ln(errmax));
        GOTO 1 END
    ELSE BEGIN
        hdid := h;
        IF (errmax > errcon) THEN BEGIN
            hnext := safety*h*exp(pgrow*ln(errmax)) END
        ELSE BEGIN
            hnext := 4.0*h END END;
    FOR i := 1 TO n DO BEGIN
        y[i] := y[i]+ytemp[i]*fcor END
END;

PROCEDURE odeint(VAR ystart: glnarray; nvar: integer;
        x1,x2,eps,h1,hmin: real; VAR nok,nbad: integer);
(* Programs using routine ODEINT must provide a
PROCEDURE derivs(x:real; y:glnarray; VAR dydx:glnarray);
which returns the derivatives dydx at location x, given both x
and the function values y. They must also define the type
TYPE
    glnarray = ARRAY [1..nvar] OF real;
and must declare the following parameters
VAR
    kmax,kount: integer; dxsav: real;
    xp: ARRAY [1..200] OF real;
    yp: ARRAY [1..10,1..200] OF real;
and must initialize kmax and dxsav in the main routine. *)
LABEL 99;
CONST
    maxstp=10000; two=2.0; zero=0.0; tiny=1.0e-30;
VAR
    nstp,i: integer; xsav,x,hnext,hdid,h: real; yscal,y,dydx: glnarray;
BEGIN
    x := x1;
    IF (x2 >= x1) THEN h := abs(h1) ELSE h := -abs(h1);
    nok := 0; nbad := 0; kount := 0;
    FOR i := 1 TO nvar DO BEGIN
        y[i] := ystart[i] END;
    IF kmax>0 THEN xsav := x-dxsav*two;
    FOR nstp := 1 TO maxstp DO BEGIN
        derivs(x,y,dydx);
```

```
        FOR i := 1 TO nvar DO BEGIN
            yscal[i] := abs(y[i])+abs(dydx[i]*h)+tiny END;
        IF (kmax > 0) THEN BEGIN
            IF (abs(x-xsav) > abs(dxsav)) THEN BEGIN
                IF (kount < kmax-1) THEN BEGIN
                    kount := kount+1; xp[kount] := x;
                    FOR i := 1 TO nvar DO BEGIN
                        yp[i,kount] := y[i] END;
                    xsav := x END END END;
        IF (((x+h-x2)*(x+h-x1)) > zero) THEN h := x2-x;
        rkqc(y,dydx,nvar,x,h,eps,yscal,hdid,hnext);
        IF (hdid = h) THEN BEGIN
            nok := nok+1 END
        ELSE BEGIN
            nbad := nbad+1 END;
        IF (((x-x2)*(x2-x1)) >= zero) THEN BEGIN
            FOR i := 1 TO nvar DO BEGIN
                ystart[i] := y[i] END;
            IF (kmax <> 0) THEN BEGIN
                kount := kount+1; xp[kount] := x;
                FOR i := 1 TO nvar DO BEGIN
                    yp[i,kount] := y[i] END END;
            GOTO 99
        END;
        IF (abs(hnext) < hmin) THEN BEGIN
            writeln('pause in routine ODEINT');
            writeln('stepsize too small'); readln END;
        h := hnext; END;
    writeln('pause in routine ODEINT - too many steps'); readln;
99: END;
```

Modified Midpoint Method

```
PROCEDURE mmid(y,dydx: glnarray; nvar: integer; xs,htot: real;
        nstep: integer; VAR yout: glnarray);
(* Programs using routine MMID must provide a
PROCEDURE derivs(x:real; y:glnarray; VAR dydx:glnarray);
which returns the derivatives dydx at location x, given both x and the
function values y. They must also define the type
TYPE
    glnarray = ARRAY [1..nvar] OF real;
in the main routine. Note that this routine is in
single precision, unlike the FORTRAN version. *)
VAR
    n,i: integer; x,swap,h2,h: real; ym,yn: glnarray;
BEGIN
    h := htot/nstep;
    FOR i := 1 TO nvar DO BEGIN
        ym[i] := y[i]; yn[i] := y[i]+h*dydx[i] END;
    x := xs+h; derivs(x,yn,yout); h2 := 2.0*h;
    FOR n := 2 TO nstep DO BEGIN
        FOR i := 1 TO nvar DO BEGIN
            swap := ym[i]+h2*yout[i]; ym[i] := yn[i]; yn[i] := swap END;
        x := x+h; derivs(x,yn,yout) END;
    FOR i := 1 TO nvar DO BEGIN
        yout[i] := 0.5*(ym[i]+yn[i]+h*yout[i]) END
END;
```


Richardson Extrapolation and the Bulirsch-Stoer Method

```
PROCEDURE bsstep(VAR y: glyarray; dydx: glyarray; nv: integer; VAR x: real;
        htry,eps: real; yscal: glyarray; VAR hdid,hnext: real);
(* Programs using routine BSSTEP must define the type
TYPE
    glyarray = ARRAY [1..nv] OF real;
in the main routine. *)
LABEL 99;
CONST
    imax=11; nuse=7; one=1.0e0; shrink=0.95e0; grow=1.2e0;
VAR
    j,i: integer; xsav,xest,h,errmax: real; ysav,dysav,yseq,yerr: glyarray;
    nseq: ARRAY [1..imax] OF integer;
BEGIN
    nseq[1] := 2; nseq[2] := 4; nseq[3] := 6;
    nseq[4] := 8; nseq[5] := 12; nseq[6] := 16;
    nseq[7] := 24; nseq[8] := 32; nseq[9] := 48;
    nseq[10] := 64; nseq[11] := 96; h := htry; xsav := x;
    FOR i := 1 TO nv DO BEGIN
        ysav[i] := y[i]; dysav[i] := dydx[i] END;
    WHILE true DO BEGIN
        FOR i := 1 TO imax DO BEGIN
            mmid(ysav,dysav,nv,xsav,h,nseq[i],yseq); xest := sqr(h/nseq[i]);
            rzextr(i,xest,yseq,y,yerr,nv,nuse); errmax := 0.0;
            FOR j := 1 TO nv DO BEGIN
                IF (errmax < abs(yerr[j]/yscal[j])) THEN
                    errmax := abs(yerr[j]/yscal[j]) END;
            errmax := errmax/eps;
            IF (errmax < one) THEN BEGIN
                x := x+h; hdid := h;
                IF (i = nuse) THEN hnext := h*shrink
                ELSE IF (i = nuse-1) THEN hnext := h*grow
                ELSE hnext := (h*nseq[nuse-1])/nseq[i];
                GOTO 99
            END END;
        h := 0.25*h;
        IF ((imax-nuse) DIV 2) > 0) THEN BEGIN
            FOR i := 1 TO ((imax-nuse) DIV 2) DO h := h/2
        END;
        IF ((x+h) = x) THEN BEGIN
            writeln('pause in routine BSSTEP');
            writeln('step size underflow'); readln END END;
99: END;


PROCEDURE rzextr(iest: integer; xest: real; yest: glyarray;
        VAR yz,dy: glyarray; nv,nuse: integer);
(* Programs using routine RZEXTR must declare
TYPE
    glyarray = ARRAY [1..nv] OF real;
CONST
    glimax=11; glnmax=10; glncol=7;
VAR
    glx: ARRAY [1..glimax] OF real;
    gld: ARRAY [1..glnmax,1..glncol] OF real;
in the main routine. *)
CONST
    ncol=7;
VAR
    m1,k,j: integer; yy,v,ddy,c,b1,b: real;
    fx: ARRAY [1..ncol] OF real;
BEGIN
    glx[iest] := xest;
```

```
    IF (iest = 1) THEN BEGIN
        FOR j := 1 TO nv DO BEGIN
            yz[j] := yest[j]; gld[j,1] := yest[j]; dy[j] := yest[j] END END
    ELSE BEGIN
        IF (iest < nuse) THEN m1 := iest ELSE m1 := nuse;
        FOR k := 1 TO m1-1 DO BEGIN
            fx[k+1] := glx[iest-k]/xest END;
        FOR j := 1 TO nv DO BEGIN
            yy := yest[j]; v := gld[j,1]; c := yy; gld[j,1] := yy;
            FOR k := 2 TO m1 DO BEGIN
                b1 := fx[k]*v; b := b1-c;
                IF (b <> 0.0) THEN BEGIN
                    b := (c-v)/b; ddy := c*b; c := b1*b END
                ELSE ddy := v;
                IF k<>m1 THEN v := gld[j,k]; gld[j,k] := ddy;
                yy := yy+ddy END;
            dy[j] := ddy; yz[j] := yy END END
END;
```

```
PROCEDURE pzextr(iest: integer; xest: real; yest: glyarray;
        VAR yz,dy: glyarray; nv,nuse: integer);
(* Programs using routine PZEXTR must declare
TYPE
    glyarray = ARRAY [1..nv] OF real;
CONST
    glimax=11; glnmax=10; glncol=7;
VAR
    glx: ARRAY [1..glimax] OF real;
    glqcol: ARRAY [1..glnmax,1..glncol] OF real;
in the main routine. *)
CONST
    nmax=10;
VAR
    m1,k1,j: integer; q,f2,f1,delta: real;
    d: ARRAY [1..nmax] OF real;
BEGIN
    glx[iest] := xest;
    FOR j := 1 TO nv DO BEGIN
        dy[j] := yest[j]; yz[j] := yest[j] END;
    IF (iest = 1) THEN BEGIN
        FOR j := 1 TO nv DO BEGIN
            glqcol[j,1] := yest[j] END END
    ELSE BEGIN
        IF (iest < nuse) THEN m1 := iest ELSE m1 := nuse;
        FOR j := 1 TO nv DO BEGIN
            d[j] := yest[j] END;
        FOR k1 := 1 TO m1-1 DO BEGIN
            delta := 1.0/(glx[iest-k1]-xest); f1 := xest*delta;
            f2 := glx[iest-k1]*delta;
            FOR j := 1 TO nv DO BEGIN
                q := glqcol[j,k1]; glqcol[j,k1] := dy[j]; delta := d[j]-q;
                dy[j] := f1*delta; d[j] := f2*delta; yz[j] := yz[j]+dy[j]
            END END;
        FOR j := 1 TO nv DO BEGIN
            glqcol[j,m1] := dy[j] END END
END;
```


Chapter 16. Two-Point Boundary Value Problems

The Shooting Method

```
PROCEDURE shoot(nvar: integer; VAR v: gln2array; delv: gln2array;
        n2: integer; x1,x2,eps,h1,hmin: real; VAR f,dv: glnvar);
(* Programs using routine SHOOT must define the types
TYPE
    gln2array = ARRAY [1..n2] OF real;
    glnvar = ARRAY [1..nvar] OF real;
    gln2byn2 = ARRAY [1..n2,1..n2] OF real;
    glinvar = ARRAY [1..nvar] OF integer;
    glnpbynp = gln2byn2;
in the main routine, and set the variable kmax of ODEINT to zero. *)
VAR
    nok,nbad,iv,i: integer; sav,det: real; y: glnvar; dfdv: gln2byn2;
    indx: glinvar;
BEGIN
    load(x1,v,y); odeint(y,nvar,x1,x2,eps,h1,hmin,nok,nbad); score(x2,y,f);
    FOR iv := 1 TO n2 DO BEGIN
        sav := v[iv]; v[iv] := v[iv]+delv[iv]; load(x1,v,y);
        odeint(y,nvar,x1,x2,eps,h1,hmin,nok,nbad); score(x2,y,dv);
        FOR i := 1 TO n2 DO BEGIN
            dfdv[i,iv] := (dv[i]-f[i])/delv[iv] END;
        v[iv] := sav END;
    FOR iv := 1 TO n2 DO BEGIN
        dv[iv] := -f[iv] END;
    ludcmp(dfdv,n2,nvar,indx,det); lubksb(dfdv,n2,nvar,indx,dv);
    FOR iv := 1 TO n2 DO BEGIN
        v[iv] := v[iv] + dv[iv] END
END;
```

Shooting to a Fitting Point

```
PROCEDURE shootf(nvar: integer; VAR v1: gln2array; VAR v2: gln1array;
        delv1: gln2array; delv2: gln1array; n1,n2: integer;
        x1,x2,xf,eps,h1,hmin: real; VAR f: glnvar;
        VAR dv1: gln2array; VAR dv2: gln1array);
(* Programs using routine SHOOTF must define the types
TYPE
    gln1array = ARRAY [1..n1] OF real;
    gln2array = ARRAY [1..n2] OF real;
    glnvar = ARRAY [1..nvar] OF real;
    glnvarbynvar = ARRAY [1..nvar,1..nvar];
    glindx = ARRAY [1..nvar] OF integer;
    glnpbynp = glnvarbynvar;
in the main routine, and set the variable kmax of ODEINT to zero. *)
VAR
    nok,nbad,j,iv,i: integer; sav,det: real; y,f1,f2: glnvar;
    dfdv: glnvarbynvar; indx: glindx;
BEGIN
    load1(x1,v1,y); odeint(y,nvar,x1,xf,eps,h1,hmin,nok,nbad);
    score(xf,y,f1); load2(x2,v2,y);
    odeint(y,nvar,x2,xf,eps,h1,hmin,nok,nbad); score(xf,y,f2); j := 0;
    FOR iv := 1 TO n2 DO BEGIN
        j := j+1; sav := v1[iv]; v1[iv] := v1[iv]+delv1[iv]; load1(x1,v1,y);
        odeint(y,nvar,x1,xf,eps,h1,hmin,nok,nbad); score(xf,y,f);
        FOR i := 1 TO nvar DO BEGIN
            dfdv[i,j] := (f[i]-f1[i])/delv1[iv] END;
```