```
TYPE
    glindex = ARRAY [1..nyj] OF integer;
    glsarray = ARRAY [1..nsi,1..nsj] OF real;
    glyarray = ARRAY [1..nyj,1..nyk] OF real;
and also declare the global quantities
CONST
    m=41;
VAR
    h,c2,anorm: real; mm,n: integer;
    x: ARRAY [1..m] OF real;
in the main routine.    *)
VAR
    temp2,temp: real;
BEGIN
    IF (k = k1) THEN BEGIN
        IF (((n+mm) MOD 2) = 1) THEN BEGIN
            s[3,3+indexv[1]] := 1.0; s[3,3+indexv[2]] := 0.0;
            s[3,3+indexv[3]] := 0.0; s[3,jsf] := y[1,1] END
        ELSE BEGIN
            s[3,3+indexv[1]] := 0.0; s[3,3+indexv[2]] := 1.0;
            s[3,3+indexv[3]] := 0.0; s[3,jsf] := y[2,1] END END
    ELSE IF (k > k2) THEN BEGIN
        s[1,3+indexv[1]] := -(y[3,m]-c2)/(2.0*(mm+1.0));
        s[1,3+indexv[2]] := 1.0; s[1,3+indexv[3]] := -y[1,m]/(2.0*(mm+1.0));
        s[1,jsf] := y[2,m]-(y[3,m]-c2)*y[1,m]/(2.0*(mm+1.0));
        s[2,3+indexv[1]] := 1.0; s[2,3+indexv[2]] := 0.0;
        s[2,3+indexv[3]] := 0.0; s[2,jsf] := y[1,m]-anorm END
    ELSE BEGIN
        s[1,indexv[1]] := -1.0; s[1,indexv[2]] := -0.5*h;
        s[1,indexv[3]] := 0.0; s[1,3+indexv[1]] := 1.0;
        s[1,3+indexv[2]] := -0.5*h; s[1,3+indexv[3]] := 0.0;
        temp := h/(1.0-sqr(x[k]+x[k-1])*0.25);
        temp2 := 0.5*(y[3,k]+y[3,k-1])-c2*0.25*sqr(x[k]+x[k-1]);
        s[2,indexv[1]] := temp*temp2*0.5;
        s[2,indexv[2]] := -1.0-0.5*temp*(mm+1.0)*(x[k]+x[k-1]);
        s[2,indexv[3]] := 0.25*temp*(y[1,k]+y[1,k-1]);
        s[2,3+indexv[1]] := s[2,indexv[1]];
        s[2,3+indexv[2]] := 2.0+s[2,indexv[2]];
        s[2,3+indexv[3]] := s[2,indexv[3]]; s[3,indexv[1]] := 0.0;
        s[3,indexv[2]] := 0.0; s[3,indexv[3]] := -1.0;
        s[3,3+indexv[1]] := 0.0; s[3,3+indexv[2]] := 0.0;
        s[3,3+indexv[3]] := 1.0;
        s[1,jsf] := y[1,k]-y[1,k-1]-0.5*h*(y[2,k]+y[2,k-1]);
        s[2,jsf] := y[2,k]-y[2,k-1]-temp*((x[k]+x[k-1])
            *0.5*(mm+1.0)*(y[2,k]+y[2,k-1])-temp2 *0.5*(y[1,k]+y[1,k-1]));
        s[3,jsf] := y[3,k]-y[3,k-1] END
END;
```

Chapter 17. Introduction to Partial Differential Equations

Relaxation Methods for Boundary Value Problems

```
PROCEDURE sor(a,b,c,d,e,f: gljmax; VAR u: gljmax;
          jmax: integer; rjac: double);
(* Programs using routine SOR must define the type
TYPE
    gljmax = ARRAY [1..jmax,1..jmax] OF double;
in the main routine. *)
LABEL 99;
CONST
```



```
    maxits=1000; eps=1.0e-5; zero=0.0; half=0.5; qtr=0.25; one=1.0;
VAR
    n,l,j: integer; resid,omega,anormf,anorm: double;
BEGIN
    anormf := zero;
    FOR j := 2 TO jmax-1 DO BEGIN
        FOR l := 2 TO jmax-1 DO BEGIN
            anormf := anormf+abs(f[j,l]) END END;
    omega := one;
    FOR n := 1 TO maxits DO BEGIN
        anorm := zero;
        FOR j := 2 TO (jmax-1) DO BEGIN
            FOR l := 2 TO (jmax-1) DO BEGIN
                IF (((j+l) MOD 2) = (n MOD 2)) THEN BEGIN
                    resid := a[j,l]*u[j+1,l]+b[j,l]*u[j-1,l]
                        +c[j,l]*u[j,l+1]+d[j,l]*u[j,l-1]
                        +e[j,l]*u[j,l]-f[j,l]; anorm := anorm+abs(resid);
                    u[j,l] := u[j,l]-omega*resid/e[j,l] END END END;
        IF (n = 1) THEN BEGIN
            omega := one/(one-half*sqr(rjac)) END
        ELSE BEGIN
            omega := one/(one-qtr*sqr(rjac)*omega) END;
        IF ((n > 1) AND (anorm < (eps*anormf))) THEN GOTO 99
    END;
    writeln('pause in routine SOR');
    writeln('too many iterations'); readln;
99: END;
```

Operator Splitting Methods and ADI

```
PROCEDURE adi(a,b,c,d,e,f,g: gljmax; VAR u: gljmax;
        jmax,k: integer; alpha,beta,eps: double);
(* Programs using routine ADI must define the type
TYPE
    gljmax = ARRAY [1..jmax,1..jmax] OF double;
in the main routine. *)
LABEL 99;
CONST
    jj=50; kk=6;
    nrr=32;         (* nrr=2 to the power (kk-1) *)
    maxits=100; zero=0.0; two=2.0; half=0.5;
TYPE
    gljjarray = ARRAY [1..jj] OF double;
VAR
    i,nr,nits,next,n,l,kits,k1,j,twopwr: integer;
    rfact,resid,disc,anormg,anorm,ab: double; aa,bb,cc,rr,uu: gljjarray;
    psi: ARRAY [1..jj,1..jj] OF double;
    alph,bet: ARRAY [1..kk] OF double;
    r: ARRAY [1..nrr] OF double;
    s: ARRAY [1..nrr,1..kk] OF double;
PROCEDURE tridag(a,b,c,r: gljjarray; VAR u: gljjarray; n: integer);
    (* This is a double precision version of TRIDAG for use with ADI,
        which defines the constant jj and the type gljjarray.    *)
    VAR
        j: integer; bet: double; gam: gljjarray;
    BEGIN
        IF (b[1] = 0.0) THEN BEGIN
            writeln('pause 1 in TRIDAG'); readln END;
        bet := b[1]; u[1] := r[1]/bet;
        FOR j := 2 TO n DO BEGIN
            gam[j] := c[j-1]/bet; bet := b[j]-a[j]*gam[j];
            IF (bet = 0.0) THEN BEGIN
```

```
            writeln('pause 2 in TRIDAG'); readln END;
          u[j] := (r[j]-a[j]*u[j-1])/bet END;
      FOR j := n-1 DOWNTO 1 DO u[j] := u[j]-gam[j+1]*u[j+1]
    END;
BEGIN
    IF (jmax > jj) THEN BEGIN
        writeln('Pause in routine ADI - increase jj'); readln
    END;
    IF (k > (kk-1)) THEN BEGIN
        writeln('Pause in routine ADI - increase kk'); readln
    END;
    k1 := k+1; nr := 1;
    FOR i := 1 TO k DO nr := 2*nr;
    alph[1] := alpha; bet[1] := beta;
    FOR j := 1 TO k DO BEGIN
        alph[j+1] := sqrt(alph[j]*bet[j]); bet[j+1] := half*(alph[j]+bet[j])
    END;
    s[1,1] := sqrt(alph[k1]*bet[k1]);
    FOR j := 1 TO k DO BEGIN
        ab := alph[k1-j]*bet[k1-j]; twopwr := 1;
        FOR i := 1 TO (j-1) DO twopwr := 2*twopwr;
        FOR n := 1 TO twopwr DO BEGIN
            disc := sqrt(sqr(s[n,j])-ab); s[2*n,j+1] := s[n,j]+disc;
            s[2*n-1,j+1] := ab/s[2*n,j+1] END END;
    FOR n := 1 TO nr DO r[n] := s[n,k1];
    anormg := zero;
    FOR j := 2 TO (jmax-1) DO BEGIN
        FOR l := 2 TO (jmax-1) DO BEGIN
            anormg := anormg+abs(g[j,l]); psi[j,l] := -d[j,l]*u[j,l-1]
                +(r[1]-e[j,l])*u[j,l]-f[j,l]*u[j,l+1] END END;
    nits := maxits DIV nr;
    FOR kits := 1 TO nits DO BEGIN
        FOR n := 1 TO nr DO BEGIN
            IF (n = nr) THEN next := 1 ELSE next := n+1;
            rfact := r[n]+r[next];
            FOR l := 2 TO (jmax-1) DO BEGIN
                FOR j := 2 TO jmax-1 DO BEGIN
                    aa[j-1] := a[j,l]; bb[j-1] := b[j,l]+r[n];
                    cc[j-1] := c[j,l]; rr[j-1] := psi[j,l]-g[j,l] END;
                tridag(aa,bb,cc,rr,uu,jmax-2);
                FOR j := 2 TO (jmax-1) DO BEGIN
                    psi[j,l] := -psi[j,l] +two*r[n]*uu[j-1] END END;
            FOR j := 2 TO (jmax-1) DO BEGIN
                FOR l := 2 TO (jmax-1) DO BEGIN
                    aa[l-1] := d[j,l]; bb[l-1] := e[j,l]+r[n];
                    cc[l-1] := f[j,l]; rr[l-1] := psi[j,l] END;
                tridag(aa,bb,cc,rr,uu,jmax-2);
                FOR l := 2 TO (jmax-1) DO BEGIN
                    u[j,l] := uu[l-1]; psi[j,l] := -psi[j,l]+rfact*uu[l-1]
                END END END;
        anorm := zero;
        FOR j := 2 TO (jmax-1) DO BEGIN
            FOR l := 2 TO (jmax-1) DO BEGIN
                resid := a[j,l]*u[j-1,l]+(b[j,l]+e[j,l])*u[j,l]
                    +c[j,l]*u[j+1,l]+d[j,l]*u[j,l-1] +f[j,l]*u[j,l+1]+g[j,l];
                anorm := anorm+abs(resid) END END;
        IF (anorm < (eps*anormg)) THEN GOTO 99
    END;
    writeln('Pause in routine ADI - too many iterations');
99: END;
```

Table of Program Dependencies

The following table lists, in alphabetical order, all the routines in *Numerical Recipes*. The section number in which the routine is listed occurs next, followed by the list of routines (if any) that are necessary for the execution of the routine first named. Note that the dependency list is complete: you do not need to look at other table entries for the additional dependencies of routines listed on the right-hand side of a given entry.

ADI	(§17.6)	TRIDAG			
AMOEBA	(§10.4)				
ANNEAL	(§10.9)	LINK			
AVEVAR	(§13.4)				
BADLUK	(§1.1)	FLMOON	JULDAY		
BALANC	(§11.5)				
BCUCOF	(§3.6)				
BCUINT	(§3.6)	BCUCOF			
BESSI	(§6.5)	BESSIO			
BESSIO	(§6.5)				
BESSI1	(§6.5)				
BESSJ	(§6.4)	BESSJ0	BESSJ1		
BESSJ0	(§6.4)				
BESSJ1	(§6.4)				
BESSK	(§6.5)	BESSK0	BESSK1	BESSIO	BESSI1
BESSK0	(§6.5)	BESSIO			
BESSK1	(§6.5)	BESSI1			
BESSY	(§6.4)	BESSY0	BESSY1	BESSJ0	BESSJ1
BESSY0	(§6.4)	BESSJ0			
BESSY1	(§6.4)	BESSJ1			
BETA	(§6.1)	GAMMLN			
BETACF	(§6.3)				
BETAI	(§6.3)	BETACF	GAMMLN		
BICO	(§6.1)	FACTLN	GAMMLN		
BKSUB	(§16.3)				
BNLDEV	(§7.3)	RAN1	GAMMLN		
BRENT	(§10.2)				
BSSTEP	(§15.4)	MMID	RZEXTR		
CALDAT	(§1.1)				
CEL	(§6.7)				
CHDER	(§5.7)				
CHEBEV	(§5.6)				