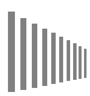
Fortran 90 Code Chapters B1–B20

Fortran 90 versions of all the Numerical Recipes routines appear in the following Chapters B1 through B20, numbered in correspondence with Chapters 1 through 20 in Volume 1. Within each chapter, the routines appear in the same order as in Volume 1, but not broken out separately by section number within Volume 1's chapters.

There are commentaries accompanying many of the routines, generally following the printed listing of the routine to which they apply. These are of two kinds: issues related to parallelizing the algorithm in question, and issues related to the Fortran 90 implementation. To distinguish between these two, rather different, kinds of discussions, we use the two icons,



the left icon (above) indicating a "parallel note," and the right icon denoting a "Fortran 90 tip." Specific code segments of the routine that are discussed in these commentaries are singled out by reproducing some of the code as an "index line" next to the icon, or at the beginning of subsequent paragraphs if there are several items that are commented on.

`d=merge(FPMIN,d,abs(d)<FPMIN)` This would be the start of a discussion of code that begins at the line in the listing containing the indicated code fragment.

⋆ ⋆ ⋆

A row of stars, like the above, is used between unrelated routines, or at the beginning and end of related groups of routines.

Some chapters contain discussions that are more general than commentary on individual routines, but that were deemed too specific for inclusion in Chapters 21 through 23. Here are some highlights of this additional material:

- Approximations to roots of orthogonal polynomials for parallel computation of Gaussian quadrature formulas (Chapter B4)
- Difficulty of, and tricks for, parallel calculation of special function values in a SIMD model of computation (Chapter B6)
- Parallel random number generation (Chapter B7)
- Fortran 90 tricks for dealing with ties in sorted arrays, counting things in boxes, etc. (Chapter B14)
- Use of recursion in implementing multigrid elliptic PDE solvers (Chapter B19)

Chapter B1. Preliminaries

```
SUBROUTINE flmoon(n,nph,jd,frac)
USE nrtype; USE nrutil, ONLY : nrerror
IMPLICIT NONE
INTEGER(I4B), INTENT(IN) :: n,nph
INTEGER(I4B), INTENT(OUT) :: jd
REAL(SP), INTENT(OUT) :: frac
    Our programs begin with an introductory comment summarizing their purpose and explain-
    ing their calling sequence. This routine calculates the phases of the moon. Given an integer
    n and a code nph for the phase desired (nph = 0 for new moon, 1 for first quarter, 2 for
    full, 3 for last quarter), the routine returns the Julian Day Number jd, and the fractional
    part of a day frac to be added to it, of the nth such phase since January, 1900. Greenwich
    Mean Time is assumed.
REAL(SP), PARAMETER :: RAD=PI/180.0_sp
INTEGER(I4B) :: i
REAL(SP) :: am,as,c,t,t2,xtra
c=n+nph/4.0_sp                          This is how we comment an individual line.
t=c/1236.85_sp
t2=t**2
as=359.2242_sp+29.105356_sp*c           You aren't really intended to understand this al-
am=306.0253_sp+385.816918_sp*c+0.010730_sp*t2         gorithm, but it does work!
jd=2415020+28*n+7*nph
xtra=0.75933_sp+1.53058868_sp*c+(1.178e-4_sp-1.55e-7_sp*t)*t2
select case(nph)
    case(0,2)
        xtra=xtra+(0.1734_sp-3.93e-4_sp*t)*sin(RAD*as)-0.4068_sp*sin(RAD*am)
    case(1,3)
        xtra=xtra+(0.1721_sp-4.0e-4_sp*t)*sin(RAD*as)-0.6280_sp*sin(RAD*am)
    case default
        call nrerror('flmoon: nph is unknown')        This is how we will indicate error
end select                                                conditions.
i=int(merge(xtra,xtra-1.0_sp, xtra >= 0.0))
jd=jd+i
frac=xtra-i
END SUBROUTINE flmoon
```

f90 `select case(nph)...case(0,2)...end select` Fortran 90 includes a `case` construction that executes at most one of several blocks of code, depending on the value of an integer, logical, or character expression. Ideally, the `case` construction will execute more efficiently than a long sequence of cascaded `if...else if...else if...` constructions. `C` programmers should note that the Fortran 90 construction, perhaps mercifully, does not have `C`'s "drop-through" feature.

`merge(xtra,xtra-1.0_sp, xtra >= 0.0)` The `merge` construction in Fortran 90, while intended primarily for use with vector arguments, is also a convenient way of generating conditional scalar expressions, that is, expressions with one value, or another, depending on the result of a logical test.

When the arguments of a merge are vectors, parallelization by the compiler is straightforward as an array parallel operation (see p. 964). Less obvious is how the scalar case, as above, is handled. For small-scale parallel (SSP) machines, the natural gain is via speculative evaluation of both of the first two arguments simultaneously with evaluation of the test.

A good compiler should not penalize a scalar machine for use of either the scalar or vector merge construction. The Fortran 90 standard states that "it is not necessary for a processor to evaluate all of the operands of an expression, or to evaluate entirely each operand, if the value of the expression can be determined otherwise." Therefore, for each test on a scalar machine, only one or the other of the first two argument components need be evaluated.

⋆ ⋆ ⋆

```
FUNCTION julday(mm,id,iyyy)
USE nrtype; USE nrutil, ONLY : nrerror
IMPLICIT NONE
INTEGER(I4B), INTENT(IN) :: mm,id,iyyy
INTEGER(I4B) :: julday
    In this routine julday returns the Julian Day Number that begins at noon of the calendar
    date specified by month mm, day id, and year iyyy, all integer variables. Positive year
    signifies A.D.; negative, B.C. Remember that the year after 1 B.C. was 1 A.D.
INTEGER(I4B), PARAMETER :: IGREG=15+31*(10+12*1582)      Gregorian Calendar adopted
INTEGER(I4B) :: ja,jm,jy                                     Oct. 15, 1582.
jy=iyyy
if (jy == 0) call nrerror('julday: there is no year zero')
if (jy < 0) jy=jy+1
if (mm > 2) then                              Here is an example of a block IF-structure.
    jm=mm+1
else
    jy=jy-1
    jm=mm+13
end if
julday=int(365.25_sp*jy)+int(30.6001_sp*jm)+id+1720995
if (id+31*(mm+12*iyyy) >= IGREG) then         Test whether to change to Gregorian Cal-
    ja=int(0.01_sp*jy)                            endar.
    julday=julday+2-ja+int(0.25_sp*ja)
end if
END FUNCTION julday
```

⋆ ⋆ ⋆

```
PROGRAM badluk
USE nrtype
USE nr, ONLY : flmoon,julday
IMPLICIT NONE
INTEGER(I4B) :: ic,icon,idwk,ifrac,im,iyyy,jd,jday,n
INTEGER(I4B) :: iybeg=1900,iyend=2000      The range of dates to be searched.
REAL(SP) :: frac
REAL(SP), PARAMETER :: TIMZON=-5.0_sp/24.0_sp
  Time zone -5 is Eastern Standard Time.
write (*,'(1x,a,i5,a,i5)') 'Full moons on Friday the 13th from',&
    iybeg,' to',iyend
do iyyy=iybeg,iyend                        Loop over each year,
    do im=1,12                             and each month.
        jday=julday(im,13,iyyy)            Is the 13th a Friday?
        idwk=mod(jday+1,7)
```

```
        if (idwk == 5) then
            n=12.37_sp*(iyyy-1900+(im-0.5_sp)/12.0_sp)
              This value n is a first approximation to how many full moons have occurred
              since 1900. We will feed it into the phase routine and adjust it up or down until
              we determine that our desired 13th was or was not a full moon. The variable
              icon signals the direction of adjustment.
            icon=0
            do
                call flmoon(n,2,jd,frac)       Get date of full moon n.
                ifrac=nint(24.0_sp*(frac+TIMZON))      Convert to hours in correct time
                if (ifrac < 0) then                        zone.
                    jd=jd-1                    Convert from Julian Days beginning at noon
                    ifrac=ifrac+24                 to civil days beginning at midnight.
                end if
                if (ifrac > 12) then
                    jd=jd+1
                    ifrac=ifrac-12
                else
                    ifrac=ifrac+12
                end if
                if (jd == jday) then           Did we hit our target day?
                    write (*,'(/1x,i2,a,i2,a,i4)') im,'/',13,'/',iyyy
                    write (*,'(1x,a,i2,a)') 'Full moon ',ifrac,&
                        ' hrs after midnight (EST).'
                      Don't worry if you are unfamiliar with FORTRAN's esoteric input/output
                      statements; very few programs in this book do any input/output.
                    exit                       Part of the break-structure, case of a match.
                else                           Didn't hit it.
                    ic=isign(1,jday-jd)
                    if (ic == -icon) exit          Another break, case of no match.
                    icon=ic
                    n=n+ic
                end if
            end do
        end if
    end do
end do
END PROGRAM badluk
```

f90 `...IGREG=15+31*(10+12*1582)` (in `julday`), `...TIMZON=-5.0_sp/24.0_sp` (in `badluk`) These are two examples of initialization expressions for "named constants" (that is, `PARAMETER`s). Because the initialization expressions will generally be evaluated at compile time, Fortran 90 puts some restrictions on what kinds of intrinsic functions they can contain. Although the evaluation of a real expression like `-5.0_sp/24.0_sp` *ought* to give identical results at compile time and at execution time, all the way down to the least significant bit, in our opinion the conservative programmer shouldn't count on strict identity at the level of floating-point roundoff error. (In the special case of *cross*-compilers, such roundoff-level discrepancies between compile time and run time are almost inevitable.)

⋆ ⋆ ⋆

```
SUBROUTINE caldat(julian,mm,id,iyyy)
USE nrtype
IMPLICIT NONE
INTEGER(I4B), INTENT(IN) :: julian
INTEGER(I4B), INTENT(OUT) :: mm,id,iyyy
    Inverse of the function julday given above. Here julian is input as a Julian Day Number,
    and the routine outputs mm,id, and iyyy as the month, day, and year on which the specified
    Julian Day started at noon.
INTEGER(I4B) :: ja,jalpha,jb,jc,jd,je
INTEGER(I4B), PARAMETER :: IGREG=2299161
if (julian >= IGREG) then                Cross-over to Gregorian Calendar produces this
    jalpha=int(((julian-1867216)-0.25_sp)/36524.25_sp)       correction.
    ja=julian+1+jalpha-int(0.25_sp*jalpha)
else if (julian < 0) then                Make day number positive by adding integer num-
    ja=julian+36525*(1-julian/36525)         ber of Julian centuries, then subtract them
else                                         off at the end.
    ja=julian
end if
jb=ja+1524
jc=int(6680.0_sp+((jb-2439870)-122.1_sp)/365.25_sp)
jd=365*jc+int(0.25_sp*jc)
je=int((jb-jd)/30.6001_sp)
id=jb-jd-int(30.6001_sp*je)
mm=je-1
if (mm > 12) mm=mm-12
iyyy=jc-4715
if (mm > 2) iyyy=iyyy-1
if (iyyy <= 0) iyyy=iyyy-1
if (julian < 0) iyyy=iyyy-100*(1-julian/36525)
END SUBROUTINE caldat
```