

Chapter B6. Special Functions

f90 A Fortran 90 intrinsic function such as `sin(x)` is both *generic* and *elemental*. Generic means that the argument `x` can be any of multiple intrinsic data types and kind values (in the case of `sin`, any real or complex kind). Elemental means that `x` need not be a scalar, but can be an array of any rank and shape, in which case the calculation of `sin` is performed independently for each element.

Ideally, when we implement more complicated special functions in Fortran 90, as we do in this chapter, we would make them, too, both generic and elemental. Unfortunately, the language standard does not completely allow this. User-defined elemental functions are prohibited in Fortran 90, though they will be allowed in Fortran 95. And, there is no fully automatic way of providing for a single routine to allow arguments of multiple data types or kinds — nothing like C++’s “class templates,” for example.

However, don’t give up hope! Fortran 90 does provide a powerful mechanism for overloading, which can be used (perhaps not always with a maximum of convenience) to *simulate* both generic and elemental function features. In most cases, when we implement a special function with a scalar argument, `gammln(x)` say, we will also implement a corresponding vector-valued function of vector argument that evaluates the special function for each component of the vector argument. We will then overload the scalar and vector version of the function onto the same function name. For example, within the `nr` module are the lines

```
INTERFACE gammln
  FUNCTION gammln_s(xx)
    USE nrtype
    REAL(SP), INTENT(IN) :: xx
    REAL(SP) :: gammln_s
  END FUNCTION gammln_s

  FUNCTION gammln_v(xx)
    USE nrtype
    REAL(SP), DIMENSION(:), INTENT(IN) :: xx
    REAL(SP), DIMENSION(size(xx)) :: gammln_v
  END FUNCTION gammln_v
END INTERFACE
```


which can be included by a statement like “`USE nr, ONLY: gammln,`” and then allow you to write `gammln(x)` without caring (or even thinking about) whether `x` is a scalar or a vector. If you want arguments of even higher rank (matrices, and so forth), you can provide these yourself, based on our models, and overload them, too.

That takes care of “elemental”; what about “generic”? Here, too, overloading provides an acceptable, if not perfect, solution. Where double-precision versions of special functions are needed, you can in many cases easily construct them from our provided routines by changing the variable kinds (and any necessary convergence

parameters), and then additionally overload them onto the same generic function names. (In general, in the interest of brevity, we will not ourselves do this for the functions in this chapter.)

At first meeting, Fortran 90's overloading capability may seem trivial, or merely cosmetic, to the Fortran 77 programmer; but one soon comes to rely on it as an important conceptual simplification. Programming at a "higher level of abstraction" is usually more productive than spending time "bogged down in the mud." Furthermore, the use of overloading is generally fail-safe: If you invoke a generic name with arguments of shapes or types for which a specific routine has not been defined, the compiler tells you about it.

We won't reprint the module `nr`'s interface blocks for all the routines in this chapter. When you see routines named `something_s` and `something_v`, below, you can safely assume that the generic name `something` is defined in the module `nr` and overloaded with the two specific routine names. A full alphabetical listing of all the interface blocks in `nr` is given in Appendix C2.



Given our heavy investment, in this chapter, in overloadable vector-valued special function routines, it is worth discussing whether this effort is simply a stopgap measure for Fortran 90, soon to be made obsolete by Fortran 95's provision of user-definable ELEMENTAL procedures. The answer is "not necessarily," and takes us into some speculation about the future of SIMD, versus MIMD, computing.

Elemental procedures, while applying the same executable code to each element, do not insist that it be feasible to perform all the parallel calculations in lockstep. That is, elemental procedures can have tests and branches (`if-then-else` constructions) that result in different elements being calculated by totally different pieces of code, in a fashion that can only be determined at run time. For true 100% MIMD (multiple instruction, multiple data) machines, this is not a problem: individual processors do the individual element calculations asynchronously.

However, virtually none of today's (and likely tomorrow's) largest-scale parallel supercomputers are 100% MIMD in this way. While modern parallel supercomputers increasingly have MIMD features, they continue to reward the use of SIMD (single instruction, multiple data) code with greater computational speed, often because of hardware pipelining or vector processing features within the individual processors. The use of Fortran 90 (or, for that matter Fortran 95) in a data-parallel or SIMD mode is thus by no means superfluous, or obviated by Fortran 95's ELEMENTAL construction.

The problem we face is that parallel calculation of special function values often doesn't fit well into the SIMD mold: Since the calculation of the value of a special function typically requires the convergence of an iterative process, as well as possible branches for different values of arguments, it cannot *in general* be done efficiently with "lockstep" SIMD programming.

Luckily, in particular cases, including most (but not all) of the functions in this chapter, one can in fact make reasonably good parallel implementations with the SIMD tools provided by the language. We will in fact see a number of different tricks for accomplishing this in the code that follows.

We are interested in demonstrating SIMD techniques, but we are not completely impractical. None of the data-parallel implementations given below are too inefficient on a scalar machine, and some may in fact be faster than Fortran 95's ELEMENTAL

alternative, or than do-loops over calls to the scalar version of the function. On a scalar machine, how can this be? We have already, above, hinted at the answer: (i) most modern scalar processors can overlap instructions to some degree, and data-parallel coding often provides compilers with the ability to accomplish this more efficiently; and (ii) data-parallel code can sometimes give better cache utilization.

* * *

```

FUNCTION gammln_s(xx)
USE nrtype; USE nrutil, ONLY : arth,assert
IMPLICIT NONE
REAL(SP), INTENT(IN) :: xx
REAL(SP) :: gammln_s
  Returns the value  $\ln[\Gamma(xx)]$  for  $xx > 0$ .
REAL(DP) :: tmp,x
  Internal arithmetic will be done in double precision, a nicety that you can omit if five-figure
  accuracy is good enough.
REAL(DP) :: stp = 2.5066282746310005_dp
REAL(DP), DIMENSION(6) :: coef = (/76.18009172947146_dp,&
  -86.50532032941677_dp,24.01409824083091_dp,&
  -1.231739572450155_dp,0.1208650973866179e-2_dp,&
  -0.5395239384953e-5_dp/)
call assert(xx > 0.0, 'gammln_s arg')
x=xx
tmp=x+5.5_dp
tmp=(x+0.5_dp)*log(tmp)-tmp
gammln_s=tmp+log(stp*(1.000000000190015_dp+&
  sum(coef(:)/arth(x+1.0_dp,1.0_dp,size(coef))))/x)
END FUNCTION gammln_s

```


```

FUNCTION gammln_v(xx)
USE nrtype; USE nrutil, ONLY: assert
IMPLICIT NONE
INTEGER(I4B) :: i
REAL(SP), DIMENSION(:), INTENT(IN) :: xx
REAL(SP), DIMENSION(size(xx)) :: gammln_v
REAL(DP), DIMENSION(size(xx)) :: ser,tmp,x,y
REAL(DP) :: stp = 2.5066282746310005_dp
REAL(DP), DIMENSION(6) :: coef = (/76.18009172947146_dp,&
  -86.50532032941677_dp,24.01409824083091_dp,&
  -1.231739572450155_dp,0.1208650973866179e-2_dp,&
  -0.5395239384953e-5_dp/)
if (size(xx) == 0) RETURN
call assert(all(xx > 0.0), 'gammln_v arg')
x=xx
tmp=x+5.5_dp
tmp=(x+0.5_dp)*log(tmp)-tmp
ser=1.000000000190015_dp
y=x
do i=1,size(coef)
  y=y+1.0_dp
  ser=ser+coef(i)/y
end do
gammln_v=tmp+log(stp*ser/x)
END FUNCTION gammln_v

```

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for Internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-
 readable files (including this one), to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs
 visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

f90 call assert(xx > 0.0, 'gammln_s arg') We use the `nrutil` routine `assert` for functions that have restrictions on the allowed range of arguments. One could instead have used an `if` statement with a call to `nrerror`; but we think that the uniformity of using `assert`, and the fact that its logical arguments read the “desired” way, not the “erroneous” way, make for a clearer programming style. In the vector version, the `assert` line is: `call assert(all(xx > 0.0), 'gammln_v arg')`

 Notice that the scalar and vector versions achieve parallelism in quite different ways, something that we will see many times in this chapter. In the scalar case, parallelism (at least small-scale) is achieved through constructions like

```
sum(coef(:)/arth(x+1.0_dp,1.0_dp,size(coef)))
```

Here vector utilities construct the series $x + 1, x + 2, \dots$ and then sum a series with these terms in the denominators and a vector of coefficients in the numerators. (This code may seem terse to Fortran 90 novices, but once you get used to it, it is quite clear to read.)

In the vector version, by contrast, parallelism is achieved across the components of the vector argument, and the above series is evaluated sequentially as a `do-loop`. Obviously the assumption is that the length of the vector argument is much longer than the very modest number (here, 6) of terms in the sum.

* * *

```
FUNCTION factrl_s(n)
USE nrtype; USE nrutil, ONLY : arth,assert,cumprod
USE nr, ONLY : gammln
IMPLICIT NONE
INTEGER(I4B), INTENT(IN) :: n
REAL(SP) :: factrl_s
  Returns the value n! as a floating-point number.
INTEGER(I4B), SAVE :: ntop=0
INTEGER(I4B), PARAMETER :: NMAX=32
REAL(SP), DIMENSION(NMAX), SAVE :: a      Table of stored values.
call assert(n >= 0, 'factrl_s arg')
if (n < ntop) then                          Already in table.
  factrl_s=a(n+1)
else if (n < NMAX) then                      Fill in table up to NMAX.
  ntop=NMAX
  a(1)=1.0
  a(2:NMAX)=cumprod(arth(1.0_sp,1.0_sp,NMAX-1))
  factrl_s=a(n+1)
else                                         Larger value than size of table is required.
  factrl_s=exp(gammln(n+1.0_sp))            Actually, this big a value is going to over-
end if                                       flow on many computers, but no harm in
END FUNCTION factrl_s                       trying.
```

f90 `cumprod(arth(1.0_sp,1.0_sp,NMAX-1))` By now you should recognize this as an idiom for generating a vector of consecutive factorials. The routines `cumprod` and `arth`, both in `nrutil`, are both capable of being parallelized, e.g., by recursion, so this idiom is potentially faster than an in-line `do-loop`.

```

FUNCTION factrl_v(n)
USE nrtype; USE nrutil, ONLY : arth,assert,cumprod
USE nr, ONLY : gammln
IMPLICIT NONE
INTEGER(I4B), DIMENSION(:), INTENT(IN) :: n
REAL(SP), DIMENSION(size(n)) :: factrl_v
LOGICAL(LGT), DIMENSION(size(n)) :: mask
INTEGER(I4B), SAVE :: ntop=0
INTEGER(I4B), PARAMETER :: NMAX=32
REAL(SP), DIMENSION(NMAX), SAVE :: a
call assert(all(n >= 0), 'factrl_v arg')
if (ntop == 0) then
  ntop=NMAX
  a(1)=1.0
  a(2:NMAX)=cumprod(arth(1.0_sp,1.0_sp,NMAX-1))
end if
mask = (n >= NMAX)
factrl_v=unpack(exp(gammln(pack(n,mask)+1.0_sp)),mask,0.0_sp)
where (.not. mask) factrl_v=a(n+1)
END FUNCTION factrl_v

```



`unpack(exp(gammln(pack(n,mask)+1.0_sp)),mask,0.0_sp)` Here we meet the first of several solutions to a common problem: How shall we get answers, from an external vector-valued function, for just a *subset* of vector arguments, those defined by a mask? Here we use what we call the “pack-unpack” solution: Pack up all the arguments using the mask, send them to the function, and unpack the answers that come back. This packing and unpacking is not without cost (highly dependent on machine architecture, to be sure), but we hope to “earn it back” in the parallelism of the external function.

`where (.not. mask) factrl_v=a(n+1)` In some cases we might take care of the `.not. mask` case directly within the unpack construction, using its third (“FIELD=”) argument to provide the not-unpacked values. However, there is no guarantee that the compiler won’t evaluate all components of the “FIELD=” array, if it finds it efficient to do so. Here, since the index of `a(n+1)` would be out of range, we can’t do it this way. Thus the separate `where` statement.

* * *

```

FUNCTION bico_s(n,k)
USE nrtype
USE nr, ONLY : factln
IMPLICIT NONE
INTEGER(I4B), INTENT(IN) :: n,k
REAL(SP) :: bico_s
  Returns the binomial coefficient  $\binom{n}{k}$  as a floating-point number.
bico_s=nint(exp(factln(n)-factln(k)-factln(n-k)))
  The nearest-integer function cleans up roundoff error for smaller values of n and k.
END FUNCTION bico_s

```

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one), to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

```

FUNCTION bico_v(n,k)
USE nrtype; USE nrutil, ONLY : assert_eq
USE nr, ONLY : factln
IMPLICIT NONE
INTEGER(I4B), DIMENSION(:), INTENT(IN) :: n,k
REAL(SP), DIMENSION(size(n)) :: bico_v
INTEGER(I4B) :: ndum
ndum=assert_eq(size(n),size(k),'bico_v')
bico_v=nint(exp(factln(n)-factln(k)-factln(n-k)))
END FUNCTION bico_v

```

* * *

```

FUNCTION factln_s(n)
USE nrtype; USE nrutil, ONLY : arth,assert
USE nr, ONLY : gammln
IMPLICIT NONE
INTEGER(I4B), INTENT(IN) :: n
REAL(SP) :: factln_s
  Returns ln(n!).
INTEGER(I4B), PARAMETER :: TMAX=100
REAL(SP), DIMENSION(TMAX), SAVE :: a
LOGICAL(LGT), SAVE :: init=.true.
if (init) then      Initialize the table.
  a(1:TMAX)=gammln(arth(1.0_sp,1.0_sp,TMAX))
  init=.false.
end if
call assert(n >= 0, 'factln_s arg')
if (n < TMAX) then  In range of the table.
  factln_s=a(n+1)
else                Out of range of the table.
  factln_s=gammln(n+1.0_sp)
end if
END FUNCTION factln_s

```

```

FUNCTION factln_v(n)
USE nrtype; USE nrutil, ONLY : arth,assert
USE nr, ONLY : gammln
IMPLICIT NONE
INTEGER(I4B), DIMENSION(:), INTENT(IN) :: n
REAL(SP), DIMENSION(size(n)) :: factln_v
LOGICAL(LGT), DIMENSION(size(n)) :: mask
INTEGER(I4B), PARAMETER :: TMAX=100
REAL(SP), DIMENSION(TMAX), SAVE :: a
LOGICAL(LGT), SAVE :: init=.true.
if (init) then
  a(1:TMAX)=gammln(arth(1.0_sp,1.0_sp,TMAX))
  init=.false.
end if
call assert(all(n >= 0), 'factln_v arg')
mask = (n >= TMAX)
factln_v=unpack(gammln(pack(n,mask)+1.0_sp),mask,0.0_sp)
where (.not. mask) factln_v=a(n+1)
END FUNCTION factln_v

```

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for Internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one) to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

f90 `gammln(arth(1.0_sp,1.0_sp,TMAX))` Another example of the programming convenience of combining a function returning a vector (here, `arth`) with a special function whose generic name (here, `gammln`) has an overloaded vector version.

* * *

```
FUNCTION beta_s(z,w)
USE nrtype
USE nr, ONLY : gammln
IMPLICIT NONE
REAL(SP), INTENT(IN) :: z,w
REAL(SP) :: beta_s
    Returns the value of the beta function  $B(z,w)$ .
beta_s=exp(gammln(z)+gammln(w)-gammln(z+w))
END FUNCTION beta_s
```

```
FUNCTION beta_v(z,w)
USE nrtype; USE nrutil, ONLY : assert_eq
USE nr, ONLY : gammln
IMPLICIT NONE
REAL(SP), DIMENSION(:), INTENT(IN) :: z,w
REAL(SP), DIMENSION(size(z)) :: beta_v
INTEGER(I4B) :: ndum
ndum=assert_eq(size(z),size(w),'beta_v')
beta_v=exp(gammln(z)+gammln(w)-gammln(z+w))
END FUNCTION beta_v
```

* * *

```
FUNCTION gammp_s(a,x)
USE nrtype; USE nrutil, ONLY : assert
USE nr, ONLY : gcf,gser
IMPLICIT NONE
REAL(SP), INTENT(IN) :: a,x
REAL(SP) :: gammp_s
    Returns the incomplete gamma function  $P(a,x)$ .
call assert( x >= 0.0, a > 0.0, 'gammp_s args')
if (x<a+1.0_sp) then
    gammp_s=gser(a,x)
    Use the series representation.
else
    gammp_s=1.0_sp-gcf(a,x)
    Use the continued fraction representation
    and take its complement.
end if
END FUNCTION gammp_s
```

```
FUNCTION gammp_v(a,x)
USE nrtype; USE nrutil, ONLY : assert,assert_eq
USE nr, ONLY : gcf,gser
IMPLICIT NONE
REAL(SP), DIMENSION(:), INTENT(IN) :: a,x
REAL(SP), DIMENSION(size(x)) :: gammp_v
LOGICAL(LGT), DIMENSION(size(x)) :: mask
INTEGER(I4B) :: ndum
ndum=assert_eq(size(a),size(x),'gammp_v')
call assert( all(x >= 0.0), all(a > 0.0), 'gammp_v args')
mask = (x<a+1.0_sp)
gammp_v=merge(gser(a,merge(x,0.0_sp,mask)), &
    1.0_sp-gcf(a,merge(x,0.0_sp,.not. mask)),mask)
END FUNCTION gammp_v
```

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for Internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one) to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).



call `assert(x >= 0.0, a > 0.0, 'gammq_s args')` The generic routine `assert` in `nrutil` is overloaded with variants for more than one logical assertion, so you can make more than one assertion about argument ranges.



`gammq_v=merge(gser(a,merge(x,0.0_sp,mask)), & 1.0_sp-gcf(a,merge(x,0.0_sp,.not. mask)),mask)` Here we meet the *second* solution to the problem of getting masked values from an external vector function. (For the first solution, see note to `factrl`, above.) We call this one “merge with dummy values”: Inappropriate values of the argument `x` (as determined by `mask`) are set to zero before `gser`, and later `gcf`, are called, and the supernumerary answers returned are discarded by a final merge. The assumption here is that the dummy value sent to the function (here, zero) is a special value that computes extremely fast, so that the overhead of computing and returning the supernumerary function values is outweighed by the parallelism achieved on the nontrivial components of `x`. Look at `gser_v` and `gcf_v` below to judge whether this assumption is realistic in this case.

```

FUNCTION gammq_s(a,x)
USE nrtype; USE nrutil, ONLY : assert
USE nr, ONLY : gcf,gser
IMPLICIT NONE
REAL(SP), INTENT(IN) :: a,x
REAL(SP) :: gammq_s
    Returns the incomplete gamma function  $Q(a,x) \equiv 1 - P(a,x)$ .
call assert(x >= 0.0, a > 0.0, 'gammq_s args')
if (x<a+1.0_sp) then           Use the series representation
    gammq_s=1.0_sp-gser(a,x)   and take its complement.
else                           Use the continued fraction representation.
    gammq_s=gcf(a,x)
end if
END FUNCTION gammq_s

```

```

FUNCTION gammq_v(a,x)
USE nrtype; USE nrutil, ONLY : assert,assert_eq
USE nr, ONLY : gcf,gser
IMPLICIT NONE
REAL(SP), DIMENSION(:), INTENT(IN) :: a,x
REAL(SP), DIMENSION(size(a)) :: gammq_v
LOGICAL(LGT), DIMENSION(size(x)) :: mask
INTEGER(I4B) :: ndum
ndum=assert_eq(size(a),size(x),'gammq_v')
call assert(all(x >= 0.0), all(a > 0.0), 'gammq_v args')
mask = (x<a+1.0_sp)
gammq_v=merge(1.0_sp-gser(a,merge(x,0.0_sp,mask)), &
    gcf(a,merge(x,0.0_sp,.not. mask)),mask)
END FUNCTION gammq_v

```

```

FUNCTION gser_s(a,x,gln)
USE nrtype; USE nrutil, ONLY : nrerror
USE nr, ONLY : gammln
IMPLICIT NONE
REAL(SP), INTENT(IN) :: a,x
REAL(SP), OPTIONAL, INTENT(OUT) :: gln
REAL(SP) :: gser_s
INTEGER(I4B), PARAMETER :: ITMAX=100
REAL(SP), PARAMETER :: EPS=epsilon(x)

```


Returns the incomplete gamma function $P(a, x)$ evaluated by its series representation as `gser`. Also optionally returns $\ln \Gamma(a)$ as `gln`.

```

INTEGER(I4B) :: n
REAL(SP) :: ap, del, summ
if (x == 0.0) then
    gser_s=0.0
    RETURN
end if
ap=a
summ=1.0_sp/a
del=summ
do n=1, ITMAX
    ap=ap+1.0_sp
    del=del*x/ap
    summ=summ+del
    if (abs(del) < abs(summ)*EPS) exit
end do
if (n > ITMAX) call nrerror('a too large, ITMAX too small in gser_s')
if (present(gln)) then
    gln=gammln(a)
    gser_s=summ*exp(-x+a*log(x)-gln)
else
    gser_s=summ*exp(-x+a*log(x)-gammln(a))
end if
END FUNCTION gser_s

```

```

FUNCTION gser_v(a,x,gln)
USE nrtype; USE nrutil, ONLY : assert_eq, nrerror
USE nr, ONLY : gammln
IMPLICIT NONE
REAL(SP), DIMENSION(:), INTENT(IN) :: a,x
REAL(SP), DIMENSION(:), OPTIONAL, INTENT(OUT) :: gln
REAL(SP), DIMENSION(size(a)) :: gser_v
INTEGER(I4B), PARAMETER :: ITMAX=100
REAL(SP), PARAMETER :: EPS=epsilon(x)
INTEGER(I4B) :: n
REAL(SP), DIMENSION(size(a)) :: ap, del, summ
LOGICAL(LGT), DIMENSION(size(a)) :: converged, zero
n=assert_eq(size(a), size(x), 'gser_v')
zero=(x == 0.0)
where (zero) gser_v=0.0
ap=a
summ=1.0_sp/a
del=summ
converged=zero
do n=1, ITMAX
    where (.not. converged)
        ap=ap+1.0_sp
        del=del*x/ap
        summ=summ+del
        converged = (abs(del) < abs(summ)*EPS)
    end where
    if (all(converged)) exit
end do
if (n > ITMAX) call nrerror('a too large, ITMAX too small in gser_v')
if (present(gln)) then
    if (size(gln) < size(a)) call &
        nrerror('gser: Not enough space for gln')
    gln=gammln(a)
    where (.not. zero) gser_v=summ*exp(-x+a*log(x)-gln)
else
    where (.not. zero) gser_v=summ*exp(-x+a*log(x)-gammln(a))

```


Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one), to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

```
end if
END FUNCTION gser_v
```

f90 REAL(SP), OPTIONAL, INTENT(OUT) :: gln Normally, an OPTIONAL argument will be INTENT(IN) and be used to provide a less-often-used extra input argument to a function. Here, the OPTIONAL argument is INTENT(OUT), used to provide a useful value that is a byproduct of the main calculation.

Also note that although $x \geq 0$ is required, we omit our usual call `assert` check for this, because `gser` is supposed to be called only by `gammp` or `gammq` — and these routines supply the argument checking themselves.

`do n=1,ITMAX...end do...if (n > ITMAX)...` This is typical code in Fortran 90 for a loop with a maximum number of iterations, relying on Fortran 90's guarantee that the index of the `do`-loop will be available after normal completion of the loop with a predictable value, greater by one than the upper limit of the loop. If the `exit` statement within the loop is ever taken, the `if` statement is guaranteed to fail; if the loop goes all the way through `ITMAX` cycles, the `if` statement is guaranteed to succeed.

 `zero=(x == 0.0)...where (zero) gser_v=0.0...converged=zero` This is the code that provides for very low overhead calculation of zero arguments, as is assumed by the merge-with-dummy-values strategy in `gammp` and `gammq`. Zero arguments are “pre-converged” and are never the holdouts in the convergence test.

```
FUNCTION gcf_s(a,x,gln)
USE nrtype; USE nrutil, ONLY : nrerror
USE nr, ONLY : gammln
IMPLICIT NONE
REAL(SP), INTENT(IN) :: a,x
REAL(SP), OPTIONAL, INTENT(OUT) :: gln
REAL(SP) :: gcf_s
INTEGER(I4B), PARAMETER :: ITMAX=100
REAL(SP), PARAMETER :: EPS=epsilon(x),FPMIN=tiny(x)/EPS
```

Returns the incomplete gamma function $Q(a, x)$ evaluated by its continued fraction representation as `gammcf`. Also optionally returns $\ln \Gamma(a)$ as `gln`.

Parameters: `ITMAX` is the maximum allowed number of iterations; `EPS` is the relative accuracy; `FPMIN` is a number near the smallest representable floating-point number.

```
INTEGER(I4B) :: i
REAL(SP) :: an,b,c,d,del,h
if (x == 0.0) then
  gcf_s=1.0
  RETURN
end if
b=x+1.0_sp-a
c=1.0_sp/FPMIN
d=1.0_sp/b
h=d
do i=1,ITMAX
  an=-i*(i-a)
  b=b+2.0_sp
  d=an*d+b
  if (abs(d) < FPMIN) d=FPMIN
  c=b+an/c
  if (abs(c) < FPMIN) c=FPMIN
```

Set up for evaluating continued fraction by modified Lentz's method (§5.2) with $b_0 = 0$.

Iterate to convergence.

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0) Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software. Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one), to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

```

    d=1.0_sp/d
    del=d*c
    h=h*del
    if (abs(del-1.0_sp) <= EPS) exit
end do
if (i > ITMAX) call nrerror('a too large, ITMAX too small in gcf_s')
if (present(gln)) then
    gln=gammln(a)
    gcf_s=exp(-x+a*log(x)-gln)*h      Put factors in front.
else
    gcf_s=exp(-x+a*log(x)-gammln(a))*h
end if
END FUNCTION gcf_s

FUNCTION gcf_v(a,x,gln)
USE nrtype; USE nrutil, ONLY : assert_eq,nrerror
USE nr, ONLY : gammln
IMPLICIT NONE
REAL(SP), DIMENSION(:), INTENT(IN) :: a,x
REAL(SP), DIMENSION(:), OPTIONAL, INTENT(OUT) :: gln
REAL(SP), DIMENSION(size(a)) :: gcf_v
INTEGER(I4B), PARAMETER :: ITMAX=100
REAL(SP), PARAMETER :: EPS=epsilon(x),FPMIN=tiny(x)/EPS
INTEGER(I4B) :: i
REAL(SP), DIMENSION(size(a)) :: an,b,c,d,del,h
LOGICAL(LGT), DIMENSION(size(a)) :: converged,zero
i=assert_eq(size(a),size(x),'gcf_v')
zero=(x == 0.0)
where (zero)
    gcf_v=1.0
elsewhere
    b=x+1.0_sp-a
    c=1.0_sp/FPMIN
    d=1.0_sp/b
    h=d
end where
converged=zero
do i=1,ITMAX
    where (.not. converged)
        an=-i*(i-a)
        b=b+2.0_sp
        d=an*d+b
        d=merge(FPMIN,d, abs(d)<FPMIN )
        c=b+an/c
        c=merge(FPMIN,c, abs(c)<FPMIN )
        d=1.0_sp/d
        del=d*c
        h=h*del
        converged = (abs(del-1.0_sp)<=EPS)
    end where
    if (all(converged)) exit
end do
if (i > ITMAX) call nrerror('a too large, ITMAX too small in gcf_v')
if (present(gln)) then
    if (size(gln) < size(a)) call &
        nrerror('gser: Not enough space for gln')
    gln=gammln(a)
    where (.not. zero) gcf_v=exp(-x+a*log(x)-gln)*h
else
    where (.not. zero) gcf_v=exp(-x+a*log(x)-gammln(a))*h
end if
END FUNCTION gcf_v

```

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for Internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-
 readable files (including this one), to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs
 visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).



zero=(x == 0.0)...where (zero) gcf_v=1.0...converged=zero See note on
gser. Here, too, we pre-converge the special value of zero.

* * *

```
FUNCTION erf_s(x)
USE nrtype
USE nr, ONLY : gammp
IMPLICIT NONE
REAL(SP), INTENT(IN) :: x
REAL(SP) :: erf_s
  Returns the error function erf(x).
erf_s=gammp(0.5_sp,x**2)
if (x < 0.0) erf_s=-erf_s
END FUNCTION erf_s
```

```
FUNCTION erf_v(x)
USE nrtype
USE nr, ONLY : gammp
IMPLICIT NONE
REAL(SP), DIMENSION(:), INTENT(IN) :: x
REAL(SP), DIMENSION(size(x)) :: erf_v
erf_v=gammp(spread(0.5_sp,1,size(x)),x**2)
where (x < 0.0) erf_v=-erf_v
END FUNCTION erf_v
```



erf_v=gammp(spread(0.5_sp,1,size(x)),x**2) Yes, we do have an over-
loaded vector version of gammp, but it is vectorized on *both* its arguments.

Thus, in a case where we want to vectorize on only *one* argument, we need a spread construction. In many contexts, Fortran 90 automatically makes scalars conformable with arrays (i.e., it automatically spreads them to the shape of the array); but the language does *not* do so when trying to match a generic function or subroutine call to a specific overloaded name. Perhaps this is wise; it is safer to prevent “accidental” invocations of vector-specific functions. Or, perhaps it is an area where the language could be improved.

```
FUNCTION erfc_s(x)
USE nrtype
USE nr, ONLY : gammp,gammq
IMPLICIT NONE
REAL(SP), INTENT(IN) :: x
REAL(SP) :: erfc_s
  Returns the complementary error function erfc(x).
erfc_s=merge(1.0_sp+gammp(0.5_sp,x**2),gammq(0.5_sp,x**2), x < 0.0)
END FUNCTION erfc_s
```



erfc_s=merge(1.0_sp+gammp(0.5_sp,x**2),gammq(0.5_sp,x**2), x < 0.0)

An example of our use of merge as an idiom for a conditional expression.
Once you get used to these, you’ll find them just as clear as the multiline
if...then...else alternative.

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-
readable files (including this one), to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs
visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

```

FUNCTION erfc_v(x)
USE nrtype
USE nr, ONLY : gammq,gammq
IMPLICIT NONE
REAL(SP), DIMENSION(:), INTENT(IN) :: x
REAL(SP), DIMENSION(size(x)) :: erfc_v
LOGICAL(LGT), DIMENSION(size(x)) :: mask
mask = (x < 0.0)
erfc_v=merge(1.0_sp+gammq(spread(0.5_sp,1,size(x)), &
    merge(x,0.0_sp,mask)**2),gammq(spread(0.5_sp,1,size(x)), &
    merge(x,0.0_sp,.not. mask)**2),mask)
END FUNCTION erfc_v

```

f90 `erfc_v=merge(1.0_sp+...)` Another example of the “merge with dummy values” idiom described on p. 1090. Here positive values of x in the call to `gammq`, and negative values in the call to `gammq`, are first set to the dummy value zero. The value zero is a special argument that computes very fast. The unwanted dummy function values are then discarded by the final outer merge.

* * *

```

FUNCTION erfcc_s(x)
USE nrtype; USE nrutil, ONLY : poly
IMPLICIT NONE
REAL(SP), INTENT(IN) :: x
REAL(SP) :: erfcc_s
    Returns the complementary error function erfc(x) with fractional error everywhere less than
     $1.2 \times 10^{-7}$ .
REAL(SP) :: t,z
REAL(SP), DIMENSION(10) :: coef = (/ -1.26551223_sp, 1.00002368_sp, &
    0.37409196_sp, 0.09678418_sp, -0.18628806_sp, 0.27886807_sp, &
    -1.13520398_sp, 1.48851587_sp, -0.82215223_sp, 0.17087277_sp /)
z=abs(x)
t=1.0_sp/(1.0_sp+0.5_sp*z)
erfcc_s=t*exp(-z*z+poly(t,coef))
if (x < 0.0) erfcc_s=2.0_sp-erfcc_s
END FUNCTION erfcc_s

```

```

FUNCTION erfcc_v(x)
USE nrtype; USE nrutil, ONLY : poly
IMPLICIT NONE
REAL(SP), DIMENSION(:), INTENT(IN) :: x
REAL(SP), DIMENSION(size(x)) :: erfcc_v,t,z
REAL(SP), DIMENSION(10) :: coef = (/ -1.26551223_sp, 1.00002368_sp, &
    0.37409196_sp, 0.09678418_sp, -0.18628806_sp, 0.27886807_sp, &
    -1.13520398_sp, 1.48851587_sp, -0.82215223_sp, 0.17087277_sp /)
z=abs(x)
t=1.0_sp/(1.0_sp+0.5_sp*z)
erfcc_v=t*exp(-z*z+poly(t,coef))
where (x < 0.0) erfcc_v=2.0_sp-erfcc_v
END FUNCTION erfcc_v

```

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one), to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

f90 `erfcc_v=t*exp(-z*z+poly(t,coef))` The vector code is identical to the scalar, because the `nrutil` routine `poly` has overloaded cases for the evaluation of a polynomial at a single value of the independent variable, and at multiple values. One *could* also overload a version with a matrix of coefficients whose columns could be used for the simultaneous evaluation of different polynomials at different values of independent variable. The point is that as long as there are differences in the shapes of at least one argument, the intended version of `poly` can be discerned by the compiler.

* * *

```

FUNCTION expint(n,x)
USE nrtype; USE nrutil, ONLY : arth,assert,nrerror
IMPLICIT NONE
INTEGER(I4B), INTENT(IN) :: n
REAL(SP), INTENT(IN) :: x
REAL(SP) :: expint
INTEGER(I4B), PARAMETER :: MAXIT=100
REAL(SP), PARAMETER :: EPS=epsilon(x),BIG=huge(x)*EPS
  Evaluates the exponential integral  $E_n(x)$ .
  Parameters: MAXIT is the maximum allowed number of iterations; EPS is the desired relative
  error, not smaller than the machine precision; BIG is a number near the largest representable
  floating-point number; EULER (in nrtype) is Euler's constant  $\gamma$ .
INTEGER(I4B) :: i,nm1
REAL(SP) :: a,b,c,d,del,fact,h
call assert(n >= 0, x >= 0.0, (x > 0.0 .or. n > 1), &
'expint args')
if (n == 0) then                               Special case.
  expint=exp(-x)/x
  RETURN
end if
nm1=n-1
if (x == 0.0) then                             Another special case.
  expint=1.0_sp/nm1
else if (x > 1.0) then                         Lentz's algorithm (§5.2).
  b=x+n
  c=BIG
  d=1.0_sp/b
  h=d
  do i=1,MAXIT
    a=-i*(nm1+i)
    b=b+2.0_sp
    d=1.0_sp/(a*d+b)           Denominators cannot be zero.
    c=b+a/c
    del=c*d
    h=h*del
    if (abs(del-1.0_sp) <= EPS) exit
  end do
  if (i > MAXIT) call nrerror('expint: continued fraction failed')
  expint=h*exp(-x)
else                                           Evaluate series.
  if (nm1 /= 0) then                          Set first term.
    expint=1.0_sp/nm1
  else
    expint=-log(x)-EULER
  end if
  fact=1.0
  do i=1,MAXIT
    fact=-fact*x/i
    if (i /= nm1) then
      del=-fact/(i-nm1)


```

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-
 readable files (including this one), to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs
 visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

```

else
     $\psi(n)$  appears here.
    del=fact*(-log(x)-EULER+sum(1.0_sp/arth(1,1,nm1)))
end if
expint=expint+del
if (abs(del) < abs(expint)*EPS) exit
end do
if (i > MAXIT) call nrerror('expint: series failed')
end if
END FUNCTION expint

```

 expint does not readily parallelize, and we thus don't provide a vector version. For syntactic convenience you could make a vector version with a do-loop over calls to this scalar version; or, in Fortran 95, you can of course make the function ELEMENTAL.

* * *

```

FUNCTION ei(x)
USE nrtype; USE nutil, ONLY : assert,nrerror
IMPLICIT NONE
REAL(SP), INTENT(IN) :: x
REAL(SP) :: ei
INTEGER(I4B), PARAMETER :: MAXIT=100
REAL(SP), PARAMETER :: EPS=epsilon(x),FPMIN=tiny(x)/EPS
    Computes the exponential integral  $Ei(x)$  for  $x > 0$ .
    Parameters: MAXIT is the maximum number of iterations allowed; EPS is the relative error,
    or absolute error near the zero of  $Ei$  at  $x = 0.3725$ ; FPMIN is a number near the smallest
    representable floating-point number; EULER (in nrtype) is Euler's constant  $\gamma$ .
INTEGER(I4B) :: k
REAL(SP) :: fact,prev,sm,term
call assert(x > 0.0, 'ei arg')
if (x < FPMIN) then
    ei=log(x)+EULER
else if (x <= -log(EPS)) then
    Use power series.
    sm=0.0
    fact=1.0
    do k=1,MAXIT
        fact=fact*x/k
        term=fact/k
        sm=sm+term
        if (term < EPS*sm) exit
    end do
    if (k > MAXIT) call nrerror('series failed in ei')
    ei=sm+log(x)+EULER
else
    Use asymptotic series.
    Start with second term.
    sm=0.0
    term=1.0
    do k=1,MAXIT
        prev=term
        term=term*k/x
        if (term < EPS) exit
        if (term < prev) then
            sm=sm+term
        else
            Diverging: subtract previous term and exit.
            sm=sm-prev
            exit
        end if
    end do
    if (k > MAXIT) call nrerror('asymptotic failed in ei')
    ei=exp(x)*(1.0_sp+sm)/x
end if
END FUNCTION ei

```

Special case: avoid failure of convergence test
because of underflow.

Since final sum is greater than one, term itself
approximates the relative error.
Still converging: add new term.

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for Internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one) to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).



ei does not readily parallelize, and we thus don't provide a vector version. For syntactic convenience you could make a vector version with a do-loop over calls to this scalar version; or, in Fortran 95, you can of course make the function ELEMENTAL.

* * *

```

FUNCTION betai_s(a,b,x)
USE nrtype; USE nrutil, ONLY : assert
USE nr, ONLY : betacf,gammln
IMPLICIT NONE
REAL(SP), INTENT(IN) :: a,b,x
REAL(SP) :: betai_s
    Returns the incomplete beta function  $I_x(a,b)$ .
REAL(SP) :: bt
call assert(x >= 0.0, x <= 1.0, 'betai_s arg')
if (x == 0.0 .or. x == 1.0) then
    bt=0.0
else
    bt=exp(gammln(a+b)-gammln(a)-gammln(b)&
+a*log(x)+b*log(1.0_sp-x))
    Factors in front of the continued frac-
    tion.
end if
if (x < (a+1.0_sp)/(a+b+2.0_sp)) then
    betai_s=bt*betacf(a,b,x)/a
    Use continued fraction directly.
else
    betai_s=1.0_sp-bt*betacf(b,a,1.0_sp-x)/b
    Use continued fraction after making the
    symmetry transformation.
end if
END FUNCTION betai_s

```

```

FUNCTION betai_v(a,b,x)
USE nrtype; USE nrutil, ONLY : assert,assert_eq
USE nr, ONLY : betacf,gammln
IMPLICIT NONE
REAL(SP), DIMENSION(:), INTENT(IN) :: a,b,x
REAL(SP), DIMENSION(size(a)) :: betai_v
REAL(SP), DIMENSION(size(a)) :: bt
LOGICAL(LGT), DIMENSION(size(a)) :: mask
INTEGER(I4B) :: ndum
ndum=assert_eq(size(a),size(b),size(x),'betai_v')
call assert(all(x >= 0.0), all(x <= 1.0), 'betai_v arg')
where (x == 0.0 .or. x == 1.0)
    bt=0.0
elsewhere
    bt=exp(gammln(a+b)-gammln(a)-gammln(b)&
+a*log(x)+b*log(1.0_sp-x))
end where
mask=(x < (a+1.0_sp)/(a+b+2.0_sp))
betai_v=bt*betacf(merge(a,b,mask),merge(b,a,mask),&
merge(x,1.0_sp-x,mask))/merge(a,b,mask)
where (.not. mask) betai_v=1.0_sp-betai_v
END FUNCTION betai_v

```

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-
 readable files (including this one), to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs
 visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).



Compare the scalar

```
if (x < (a+1.0_sp)/(a+b+2.0_sp)) then
  betai_s=bt*betacf(a,b,x)/a
else
  betai_s=1.0_sp-bt*betacf(b,a,1.0_sp-x)/b
end if
```

with the vector

```
mask=(x < (a+1.0_sp)/(a+b+2.0_sp))
betai_v=bt*betacf(merge(a,b,mask),merge(b,a,mask),&
  merge(x,1.0_sp-x,mask))/merge(a,b,mask)
where (.not. mask) betai_v=1.0_sp-betai_v
```

Here `merge` is used (several times) to evaluate all the required components in a single call to the vectorized `betacf`, notwithstanding that some components require one pattern of arguments, some a different pattern.

```
FUNCTION betacf_s(a,b,x)
USE nrtype; USE nrutil, ONLY : nrerror
IMPLICIT NONE
REAL(SP), INTENT(IN) :: a,b,x
REAL(SP) :: betacf_s
INTEGER(I4B), PARAMETER :: MAXIT=100
REAL(SP), PARAMETER :: EPS=epsilon(x), FPMIN=tiny(x)/EPS
  Used by betai: Evaluates continued fraction for incomplete beta function by modified
  Lentz's method (§5.2).
REAL(SP) :: aa,c,d,del,h,qab,qam,qap
INTEGER(I4B) :: m,m2
qab=a+b
qap=a+1.0_sp
qam=a-1.0_sp
c=1.0
d=1.0_sp-qab*x/qap
if (abs(d) < FPMIN) d=FPMIN
d=1.0_sp/d
h=d
do m=1,MAXIT
  m2=2*m
  aa=m*(b-m)*x/((qam+m2)*(a+m2))
  d=1.0_sp+aa*d
  if (abs(d) < FPMIN) d=FPMIN
  c=1.0_sp+aa/c
  if (abs(c) < FPMIN) c=FPMIN
  d=1.0_sp/d
  h=h*d*c
  aa=-(a+m)*(qab+m)*x/((a+m2)*(qap+m2))
  d=1.0_sp+aa*d
  if (abs(d) < FPMIN) d=FPMIN
  c=1.0_sp+aa/c
  if (abs(c) < FPMIN) c=FPMIN
  d=1.0_sp/d
  del=d*c
  h=h*del
  if (abs(del-1.0_sp) <= EPS) exit
end do
if (m > MAXIT)&
  call nrerror('a or b too big, or MAXIT too small in betacf_s')
betacf_s=h
END FUNCTION betacf_s
```

These q's will be used in factors that occur in the coefficients (6.4.6).

First step of Lentz's method.

One step (the even one) of the recurrence.

Next step of the recurrence (the odd one).

Are we done?

```

FUNCTION betacf_v(a,b,x)
USE nrtype; USE nrutil, ONLY : assert_eq,nrerror
IMPLICIT NONE
REAL(SP), DIMENSION(:), INTENT(IN) :: a,b,x
REAL(SP), DIMENSION(size(x)) :: betacf_v
INTEGER(I4B), PARAMETER :: MAXIT=100
REAL(SP), PARAMETER :: EPS=epsilon(x), FPMIN=tiny(x)/EPS
REAL(SP), DIMENSION(size(x)) :: aa,c,d,del,h,qab,qam,qap
LOGICAL(LGT), DIMENSION(size(x)) :: converged
INTEGER(I4B) :: m
INTEGER(I4B), DIMENSION(size(x)) :: m2
m=assert_eq(size(a),size(b),size(x),'betacf_v')
qab=a+b
qap=a+1.0_sp
qam=a-1.0_sp
c=1.0
d=1.0_sp-qab*x/qap
where (abs(d) < FPMIN) d=FPMIN
d=1.0_sp/d
h=d
converged=.false.
do m=1,MAXIT
  where (.not. converged)
    m2=2*m
    aa=m*(b-m)*x/((qam+m2)*(a+m2))
    d=1.0_sp+aa*d
    d=merge(FPMIN,d, abs(d)<FPMIN )
    c=1.0_sp+aa/c
    c=merge(FPMIN,c, abs(c)<FPMIN )
    d=1.0_sp/d
    h=h*d*c
    aa=-(a+m)*(qab+m)*x/((a+m2)*(qap+m2))
    d=1.0_sp+aa*d
    d=merge(FPMIN,d, abs(d)<FPMIN )
    c=1.0_sp+aa/c
    c=merge(FPMIN,c, abs(c)<FPMIN )
    d=1.0_sp/d
    del=d*c
    h=h*del
    converged = (abs(del-1.0_sp) <= EPS)
  end where
  if (all(converged)) exit
end do
if (m > MAXIT)&
  call nrerror('a or b too big, or MAXIT too small in betacf_v')
betacf_v=h
END FUNCTION betacf_v

```

f90 `d=merge(FPMIN,d, abs(d)<FPMIN)` The scalar version does this with an `if`. Why does it become a `merge` here in the vector version, rather than a `where`? Because we are already inside a “`where (.not. converged)`” block, and Fortran 90 doesn’t allow nested `where`’s! (Fortran 95 *will* allow nested `where`’s.)

* * *

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one), to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

```

FUNCTION bessj0_s(x)
USE nrtype; USE nrutil, ONLY : poly
IMPLICIT NONE
REAL(SP), INTENT(IN) :: x
REAL(SP) :: bessj0_s
  Returns the Bessel function  $J_0(x)$  for any real x.
REAL(SP) :: ax,xx,z
REAL(DP) :: y
  We'll accumulate polynomials in double precision.
REAL(DP), DIMENSION(5) :: p = (/1.0_dp,-0.1098628627e-2_dp,&
  0.2734510407e-4_dp,-0.2073370639e-5_dp,0.2093887211e-6_dp/)
REAL(DP), DIMENSION(5) :: q = (/ -0.1562499995e-1_dp,&
  0.1430488765e-3_dp,-0.6911147651e-5_dp,0.7621095161e-6_dp,&
  -0.934945152e-7_dp/)
REAL(DP), DIMENSION(6) :: r = (/57568490574.0_dp,-13362590354.0_dp,&
  651619640.7_dp,-11214424.18_dp,77392.33017_dp,&
  -184.9052456_dp/)
REAL(DP), DIMENSION(6) :: s = (/57568490411.0_dp,1029532985.0_dp,&
  9494680.718_dp,59272.64853_dp,267.8532712_dp,1.0_dp/)
if (abs(x) < 8.0) then
  Direct rational function fit.
  y=x**2
  bessj0_s=poly(y,r)/poly(y,s)
else
  Fitting function (6.5.9).
  ax=abs(x)
  z=8.0_sp/ax
  y=z**2
  xx=ax-0.785398164_sp
  bessj0_s=sqrt(0.636619772_sp/ax)*(cos(xx)*&
    poly(y,p)-z*sin(xx)*poly(y,q))
end if
END FUNCTION bessj0_s


```

```

FUNCTION bessj0_v(x)
USE nrtype; USE nrutil, ONLY : poly
IMPLICIT NONE
REAL(SP), DIMENSION(:), INTENT(IN) :: x
REAL(SP), DIMENSION(size(x)) :: bessj0_v
REAL(SP), DIMENSION(size(x)) :: ax,xx,z
REAL(DP), DIMENSION(size(x)) :: y
LOGICAL(LGT), DIMENSION(size(x)) :: mask
REAL(DP), DIMENSION(5) :: p = (/1.0_dp,-0.1098628627e-2_dp,&
  0.2734510407e-4_dp,-0.2073370639e-5_dp,0.2093887211e-6_dp/)
REAL(DP), DIMENSION(5) :: q = (/ -0.1562499995e-1_dp,&
  0.1430488765e-3_dp,-0.6911147651e-5_dp,0.7621095161e-6_dp,&
  -0.934945152e-7_dp/)
REAL(DP), DIMENSION(6) :: r = (/57568490574.0_dp,-13362590354.0_dp,&
  651619640.7_dp,-11214424.18_dp,77392.33017_dp,&
  -184.9052456_dp/)
REAL(DP), DIMENSION(6) :: s = (/57568490411.0_dp,1029532985.0_dp,&
  9494680.718_dp,59272.64853_dp,267.8532712_dp,1.0_dp/)
mask = (abs(x) < 8.0)
where (mask)
  y=x**2
  bessj0_v=poly(y,r,mask)/poly(y,s,mask)
elsewhere
  ax=abs(x)
  z=8.0_sp/ax
  y=z**2
  xx=ax-0.785398164_sp
  bessj0_v=sqrt(0.636619772_sp/ax)*(cos(xx)*&
    poly(y,p,.not. mask)-z*sin(xx)*poly(y,q,.not. mask))
end where
END FUNCTION bessj0_v

```

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-
 readable files (including this one), to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs
 visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).



where $(\text{mask}) \dots \text{bessj0}_v = \text{poly}(y, r, \text{mask}) / \text{poly}(y, s, \text{mask})$. Here we meet the *third* solution to the problem of getting masked values from an external vector function. (For the other two solutions, see notes to `factr1`, p. 1087, and `gammp`, p. 1090.) Here we simply evade all responsibility and pass the mask into every routine that is supposed to be masked. Let it be somebody else's problem! That works here because your hardworking authors have overloaded the `nrutil` routine `poly` with a masked vector version. More typically, of course, it becomes *your* problem, and you have to remember to write masked versions of all the vector routines that you call in this way. (We'll meet examples of this later.)

* * *

```

FUNCTION bessy0_s(x)
USE nrtype; USE nrutil, ONLY : assert,poly
USE nr, ONLY : bessj0
IMPLICIT NONE
REAL(SP), INTENT(IN) :: x
REAL(SP) :: bessy0_s
    Returns the Bessel function  $Y_0(x)$  for positive x.
REAL(SP) :: xx,z
REAL(DP) :: y
    We'll accumulate polynomials in double precision.
REAL(DP), DIMENSION(5) :: p = (/1.0_dp,-0.1098628627e-2_dp,&
    0.2734510407e-4_dp,-0.2073370639e-5_dp,0.2093887211e-6_dp/)
REAL(DP), DIMENSION(5) :: q = (/ -0.1562499995e-1_dp,&
    0.1430488765e-3_dp,-0.6911147651e-5_dp,0.7621095161e-6_dp,&
    -0.934945152e-7_dp/)
REAL(DP), DIMENSION(6) :: r = (/ -2957821389.0_dp,7062834065.0_dp,&
    -512359803.6_dp,10879881.29_dp,-86327.92757_dp,&
    228.4622733_dp/)
REAL(DP), DIMENSION(6) :: s = (/40076544269.0_dp,745249964.8_dp,&
    7189466.438_dp,47447.26470_dp,226.1030244_dp,1.0_dp/)
call assert(x > 0.0, 'bessy0_s arg')
if (abs(x) < 8.0) then
    Rational function approximation of (6.5.8).
    y=x**2
    bessy0_s=(poly(y,r)/poly(y,s))+&
        0.636619772_sp*bessj0(x)*log(x)
else
    Fitting function (6.5.10).
    z=8.0_sp/x
    y=z**2
    xx=x-0.785398164_sp
    bessy0_s=sqrt(0.636619772_sp/x)*(sin(xx)*&
        poly(y,p)+z*cos(xx)*poly(y,q))
end if
END FUNCTION bessy0_s

```

```

FUNCTION bessy0_v(x)
USE nrtype; USE nrutil, ONLY : assert,poly
USE nr, ONLY : bessj0
IMPLICIT NONE
REAL(SP), DIMENSION(:), INTENT(IN) :: x
REAL(SP), DIMENSION(size(x)) :: bessy0_v
REAL(SP), DIMENSION(size(x)) :: xx,z
REAL(DP), DIMENSION(size(x)) :: y
LOGICAL(LGT), DIMENSION(size(x)) :: mask
REAL(DP), DIMENSION(5) :: p = (/1.0_dp,-0.1098628627e-2_dp,&
    0.2734510407e-4_dp,-0.2073370639e-5_dp,0.2093887211e-6_dp/)
REAL(DP), DIMENSION(5) :: q = (/ -0.1562499995e-1_dp,&
    0.1430488765e-3_dp,-0.6911147651e-5_dp,0.7621095161e-6_dp,&

```

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one), to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

```

-0.934945152e-7_dp/)
REAL(DP), DIMENSION(6) :: r = (/ -2957821389.0_dp, 7062834065.0_dp, &
-512359803.6_dp, 10879881.29_dp, -86327.92757_dp, &
228.4622733_dp/)
REAL(DP), DIMENSION(6) :: s = (/ 40076544269.0_dp, 745249964.8_dp, &
7189466.438_dp, 47447.26470_dp, 226.1030244_dp, 1.0_dp/)
call assert(all(x > 0.0), 'bessy0_v arg')
mask = (abs(x) < 8.0)
where (mask)
  y=x**2
  bessy0_v=(poly(y,r,mask)/poly(y,s,mask))+&
0.636619772_sp*bessj0(x)*log(x)
elsewhere
  z=8.0_sp/x
  y=z**2
  xx=x-0.785398164_sp
  bessy0_v=sqrt(0.636619772_sp/x)*(sin(xx)*&
poly(y,p,.not. mask)+z*cos(xx)*poly(y,q,.not. mask))
end where
END FUNCTION bessy0_v

```

* * *

```

FUNCTION bessj1_s(x)
USE nrtype; USE nrutil, ONLY : poly
IMPLICIT NONE
REAL(SP), INTENT(IN) :: x
REAL(SP) :: bessj1_s
  Returns the Bessel function  $J_1(x)$  for any real x.
REAL(SP) :: ax,xx,z
REAL(DP) :: y
  We'll accumulate polynomials in double precision.
REAL(DP), DIMENSION(6) :: r = (/ 72362614232.0_dp, &
-7895059235.0_dp, 242396853.1_dp, -2972611.439_dp, &
15704.48260_dp, -30.16036606_dp/)
REAL(DP), DIMENSION(6) :: s = (/ 144725228442.0_dp, 2300535178.0_dp, &
18583304.74_dp, 99447.43394_dp, 376.9991397_dp, 1.0_dp/)
REAL(DP), DIMENSION(5) :: p = (/ 1.0_dp, 0.183105e-2_dp, &
-0.3516396496e-4_dp, 0.2457520174e-5_dp, -0.240337019e-6_dp/)
REAL(DP), DIMENSION(5) :: q = (/ 0.04687499995_dp, &
-0.2002690873e-3_dp, 0.8449199096e-5_dp, -0.88228987e-6_dp, &
0.105787412e-6_dp/)
if (abs(x) < 8.0) then
  Direct rational approximation.
  y=x**2
  bessj1_s=x*(poly(y,r)/poly(y,s))
else
  Fitting function (6.5.9).
  ax=abs(x)
  z=8.0_sp/ax
  y=z**2
  xx=ax-2.356194491_sp
  bessj1_s=sqrt(0.636619772_sp/ax)*(cos(xx)*&
poly(y,p)-z*sin(xx)*poly(y,q))*sign(1.0_sp,x)
end if
END FUNCTION bessj1_s

```

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one), to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

```

FUNCTION bessj1_v(x)
USE nrtype; USE nrutil, ONLY : poly
IMPLICIT NONE
REAL(SP), DIMENSION(:), INTENT(IN) :: x
REAL(SP), DIMENSION(size(x)) :: bessj1_v
REAL(SP), DIMENSION(size(x)) :: ax,xx,z
REAL(DP), DIMENSION(size(x)) :: y
LOGICAL(LGT), DIMENSION(size(x)) :: mask
REAL(DP), DIMENSION(6) :: r = (/72362614232.0_dp,&
-7895059235.0_dp,242396853.1_dp,-2972611.439_dp,&
15704.48260_dp,-30.16036606_dp/)
REAL(DP), DIMENSION(6) :: s = (/144725228442.0_dp,2300535178.0_dp,&
18583304.74_dp,99447.43394_dp,376.9991397_dp,1.0_dp/)
REAL(DP), DIMENSION(5) :: p = (/1.0_dp,0.183105e-2_dp,&
-0.3516396496e-4_dp,0.2457520174e-5_dp,-0.240337019e-6_dp/)
REAL(DP), DIMENSION(5) :: q = (/0.04687499995_dp,&
-0.2002690873e-3_dp,0.8449199096e-5_dp,-0.88228987e-6_dp,&
0.105787412e-6_dp/)
mask = (abs(x) < 8.0)
where (mask)
  y=x**2
  bessj1_v=x*(poly(y,r,mask)/poly(y,s,mask))
elsewhere
  ax=abs(x)
  z=8.0_sp/ax
  y=z**2
  xx=ax-2.356194491_sp
  bessj1_v=sqrt(0.636619772_sp/ax)*(cos(xx)*&
    poly(y,p,.not. mask)-z*sin(xx)*poly(y,q,.not. mask))*&
    sign(1.0_sp,x)
end where
END FUNCTION bessj1_v

```

* * *

```

FUNCTION bessy1_s(x)
USE nrtype; USE nrutil, ONLY : assert,poly
USE nr, ONLY : bessj1
IMPLICIT NONE
REAL(SP), INTENT(IN) :: x
REAL(SP) :: bessy1_s
Returns the Bessel function  $Y_1(x)$  for positive x.
REAL(SP) :: xx,z
REAL(DP) :: y
REAL(DP), DIMENSION(5) :: p = (/1.0_dp,0.183105e-2_dp,&
-0.3516396496e-4_dp,0.2457520174e-5_dp,-0.240337019e-6_dp/)
REAL(DP), DIMENSION(5) :: q = (/0.04687499995_dp,&
-0.2002690873e-3_dp,0.8449199096e-5_dp,-0.88228987e-6_dp,&
0.105787412e-6_dp/)
REAL(DP), DIMENSION(6) :: r = (/ -0.4900604943e13_dp,&
0.1275274390e13_dp,-0.5153438139e11_dp,0.7349264551e9_dp,&
-0.4237922726e7_dp,0.8511937935e4_dp/)
REAL(DP), DIMENSION(7) :: s = (/0.2499580570e14_dp,&
0.4244419664e12_dp,0.3733650367e10_dp,0.2245904002e8_dp,&
0.1020426050e6_dp,0.3549632885e3_dp,1.0_dp/)
call assert(x > 0.0, 'bessy1_s arg')
if (abs(x) < 8.0) then
  Rational function approximation of (6.5.8).
  y=x**2
  bessy1_s=x*(poly(y,r)/poly(y,s))+&
    0.636619772_sp*(bessj1(x)*log(x)-1.0_sp/x)
else
  Fitting function (6.5.10).

```

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one), to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

```

      z=8.0_sp/x
      y=z**2
      xx=x-2.356194491_sp
      bessy1_s=sqrt(0.636619772_sp/x)*(sin(xx)*&
        poly(y,p)+z*cos(xx)*poly(y,q))
end if
END FUNCTION bessy1_s

FUNCTION bessy1_v(x)
USE nrtype; USE nrutil, ONLY : assert,poly
USE nr, ONLY : bessj1
IMPLICIT NONE
REAL(SP), DIMENSION(:), INTENT(IN) :: x
REAL(SP), DIMENSION(size(x)) :: bessy1_v
REAL(SP), DIMENSION(size(x)) :: xx,z
REAL(DP), DIMENSION(size(x)) :: y
LOGICAL(LGT), DIMENSION(size(x)) :: mask
REAL(DP), DIMENSION(5) :: p = (/1.0_dp,0.183105e-2_dp,&
  -0.3516396496e-4_dp,0.2457520174e-5_dp,-0.240337019e-6_dp/)
REAL(DP), DIMENSION(5) :: q = (/0.04687499995_dp,&
  -0.2002690873e-3_dp,0.8449199096e-5_dp,-0.88228987e-6_dp,&
  0.105787412e-6_dp/)
REAL(DP), DIMENSION(6) :: r = (/ -0.4900604943e13_dp,&
  0.1275274390e13_dp,-0.5153438139e11_dp,0.7349264551e9_dp,&
  -0.4237922726e7_dp,0.8511937935e4_dp/)
REAL(DP), DIMENSION(7) :: s = (/0.2499580570e14_dp,&
  0.4244419664e12_dp,0.3733650367e10_dp,0.2245904002e8_dp,&
  0.1020426050e6_dp,0.3549632885e3_dp,1.0_dp/)
call assert(all(x > 0.0), 'bessy1_v arg')
mask = (abs(x) < 8.0)
where (mask)
  y=x**2
  bessy1_v=x*(poly(y,r,mask)/poly(y,s,mask))+&
    0.636619772_sp*(bessj1(x)*log(x)-1.0_sp/x)
elsewhere
  z=8.0_sp/x
  y=z**2
  xx=x-2.356194491_sp
  bessy1_v=sqrt(0.636619772_sp/x)*(sin(xx)*&
    poly(y,p,.not. mask)+z*cos(xx)*poly(y,q,.not. mask))
end where
END FUNCTION bessy1_v

```

* * *

```

FUNCTION bessy_s(n,x)
USE nrtype; USE nrutil, ONLY : assert
USE nr, ONLY : bessy0,bessy1
IMPLICIT NONE
INTEGER(I4B), INTENT(IN) :: n
REAL(SP), INTENT(IN) :: x
REAL(SP) :: bessy_s
  Returns the Bessel function  $Y_n(x)$  for positive  $x$  and  $n \geq 2$ .
INTEGER(I4B) :: j
REAL(SP) :: by,bym,byp,tox
call assert(n >= 2, x > 0.0, 'bessy_s args')
tox=2.0_sp/x
by=bessy1(x)           Starting values for the recurrence.
bym=bessy0(x)
do j=1,n-1             Recurrence (6.5.7).

```

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one), to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

```

    byp=j*tox*by-bym
    bym=by
    by=byp
end do
bessy_s=by
END FUNCTION bessy_s

```

```

FUNCTION bessy_v(n,x)
USE nrtype; USE nrutil, ONLY : assert
USE nr, ONLY : bessy0,bessy1
IMPLICIT NONE
INTEGER(I4B), INTENT(IN) :: n
REAL(SP), DIMENSION(:), INTENT(IN) :: x
REAL(SP), DIMENSION(size(x)) :: bessy_v
INTEGER(I4B) :: j
REAL(SP), DIMENSION(size(x)) :: by,bym,byp,tox
call assert(n >= 2, all(x > 0.0), 'bessy_v args')
tox=2.0_sp/x
by=bessy1(x)
bym=bessy0(x)
do j=1,n-1
    byp=j*tox*by-bym
    bym=by
    by=byp
end do
bessy_v=by
END FUNCTION bessy_v

```

f₉₀ Notice that the vector routine is *exactly* the same as the scalar routine, but operates only on vectors, and that nothing in the routine is specific to any level of precision or kind type of real variable. Cases like this make us wish that Fortran 90 provided for “template” types that could automatically take the type and shape of the actual arguments. (Such facilities are available in other, more object-oriented languages such as C++.)

* * *

```

FUNCTION bessj_s(n,x)
USE nrtype; USE nrutil, ONLY : assert
USE nr, ONLY : bessj0,bessj1
IMPLICIT NONE
INTEGER(I4B), INTENT(IN) :: n
REAL(SP), INTENT(IN) :: x
REAL(SP) :: bessj_s
INTEGER(I4B), PARAMETER :: IACC=40,IEXP=maxexponent(x)/2
    Returns the Bessel function  $J_n(x)$  for any real  $x$  and  $n \geq 2$ . Make the parameter IACC
    larger to increase accuracy.
INTEGER(I4B) :: j,jsum,m
REAL(SP) :: ax,bj,bjm,bjp,summ,tox
call assert(n >= 2, 'bessj_s args')
ax=abs(x)
if (ax*ax <= 8.0_sp*tiny(x)) then
    bessj_s=0.0
    Underflow limit.
else if (ax > real(n,sp)) then
    Upwards recurrence from  $J_0$  and  $J_1$ .
    tox=2.0_sp/ax
    bjm=bessj0(ax)
    bj=bessj1(ax)
    do j=1,n-1

```

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one), to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).


```

        bjp=j*tox*bj-bjm
        bjm=bj
        bj=bjp
    end do
    bessj_s=bj
else
    tox=2.0_sp/ax
    m=2*(n+int(sqrt(real(IACC*n,sp))))/2)
    bessj_s=0.0
    jsum=0
    summ=0.0
    bjp=0.0
    bj=1.0
    do j=m,1,-1
        bjm=j*tox*bj-bjp
        bjp=bj
        bj=bjm
        if (exponent(bj) > IEXP) then
            bj=scale(bj,-IEXP)
            bjp=scale(bjp,-IEXP)
            bessj_s=scale(bessj_s,-IEXP)
            summ=scale(summ,-IEXP)
        end if
        if (jsum /= 0) summ=summ+bj
        jsum=1-jsum
        if (j == n) bessj_s=bjp
    end do
    summ=2.0_sp*summ-bj
    bessj_s=bessj_s/summ
end if
if (x < 0.0 .and. mod(n,2) == 1) bessj_s=-bessj_s
END FUNCTION bessj_s

```

Downwards recurrence from an even m here computed.

j sum will alternate between 0 and 1; when it is 1, we accumulate in sum the even terms in (5.5.16).

The downward recurrence.

Renormalize to prevent overflows.

Accumulate the sum.
Change 0 to 1 or vice versa.
Save the unnormalized answer.

Compute (5.5.16)
and use it to normalize the answer.



The `bessj` routine does not conveniently parallelize with Fortran 90's language constructions, but Bessel functions are of sufficient importance that we feel the need for a parallel version nevertheless. The basic method adopted below is to encapsulate as contained vector functions two separate algorithms, one for the case $x \leq n$, the other for $x > n$. Both of these have masks as input arguments; within each routine, however, they immediately revert to the pack-unpack method. The choice to pack in the subsidiary routines, rather than in the main routine, is arbitrary; the main routine is supposed to be a little clearer this way.

f90 if (exponent(bj) > IEXP) then... In the Fortran 77 version of this routine, we scaled the variables by 10^{-10} whenever `bj` was bigger than 10^{10} . On a machine with a large exponent range, we could improve efficiency by scaling less often. In order to remain portable, however, we used the conservative value of 10^{10} . An elegant way of handling renormalization is provided by the Fortran 90 intrinsic functions that manipulate real numbers. We test with `if (exponent(bj) > IEXP)` and then if necessary renormalize with `bj=scale(bj,-IEXP)` and similarly for the other variables. Our conservative choice is to set `IEXP=maxexponent(x)/2`. Note that an added benefit of scaling this way is that only the exponent of each variable is modified; no roundoff error is introduced as it can be if we do a floating-point division instead.

```

FUNCTION bessj_v(n,xx)
USE nrtype; USE nrutil, ONLY : assert
USE nr, ONLY : bessj0,bessj1
IMPLICIT NONE
INTEGER(I4B), INTENT(IN) :: n
REAL(SP), DIMENSION(:), INTENT(IN) :: xx
REAL(SP), DIMENSION(size(xx)) :: bessj_v
INTEGER(I4B), PARAMETER :: IACC=40,IEXP=maxexponent(xx)/2
REAL(SP), DIMENSION(size(xx)) :: ax
LOGICAL(LGT), DIMENSION(size(xx)) :: mask,mask0
REAL(SP), DIMENSION(:), ALLOCATABLE :: x,bj,bjm,bjp,summ,tox,bessjle
LOGICAL(LGT), DIMENSION(:), ALLOCATABLE :: renorm
INTEGER(I4B) :: j,jsum,m,npak
call assert(n >= 2, 'bessj_v args')
ax=abs(xx)
mask = (ax <= real(n,sp))
mask0 = (ax*ax <= 8.0_sp*tiny(xx))
bessj_v=bessjle_v(n,ax,logical(mask .and. .not.mask0, kind=lgd))
bessj_v=merge(bessjgt_v(n,ax,.not. mask),bessj_v,.not. mask)
where (mask0) bessj_v=0.0
where (xx < 0.0 .and. mod(n,2) == 1) bessj_v=-bessj_v
CONTAINS

FUNCTION bessjgt_v(n,xx,mask)
IMPLICIT NONE
INTEGER(I4B), INTENT(IN) :: n
REAL(SP), DIMENSION(:), INTENT(IN) :: xx
LOGICAL(LGT), DIMENSION(size(xx)), INTENT(IN) :: mask
REAL(SP), DIMENSION(size(xx)) :: bessjgt_v
npak=count(mask)
if (npak == 0) RETURN
allocate(x(npak),bj(npak),bjm(npak),bjp(npak),tox(npak))
x=pack(xx,mask)
tox=2.0_sp/x
bjm=bessj0(x)
bj=bessj1(x)
do j=1,n-1
    bjp=j*tox*bj-bjm
    bjm=bj
    bj=bjp
end do
bessjgt_v=unpack(bj,mask,0.0_sp)
deallocate(x,bj,bjm,bjp,tox)
END FUNCTION bessjgt_v

FUNCTION bessjle_v(n,xx,mask)
IMPLICIT NONE
INTEGER(I4B), INTENT(IN) :: n
REAL(SP), DIMENSION(:), INTENT(IN) :: xx
LOGICAL(LGT), DIMENSION(size(xx)), INTENT(IN) :: mask
REAL(SP), DIMENSION(size(xx)) :: bessjle_v
npak=count(mask)
if (npak == 0) RETURN
allocate(x(npak),bj(npak),bjm(npak),bjp(npak),summ(npak), &
    bessjle(npak),tox(npak),renorm(npak))
x=pack(xx,mask)
tox=2.0_sp/x
m=2*((n+int(sqrt(real(IACC*n,sp)))))/2)
bessjle=0.0
jsum=0
summ=0.0
bjp=0.0
bj=1.0
do j=m,1,-1
    bjm=j*tox*bj-bjp

```

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one), to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

```

bjp=bj
bj=bjm
renorm = (exponent(bj)>IEXP)
bj=merge(scale(bj,-IEXP),bj,renorm)
bjp=merge(scale(bjp,-IEXP),bjp,renorm)
bessjle=merge(scale(bessjle,-IEXP),bessjle,renorm)
summ=merge(scale(summ,-IEXP),summ,renorm)
if (jsum /= 0) summ=summ+bj
jsum=1-jsum
if (j == n) bessjle=bjp
end do
summ=2.0_sp*summ-bj
bessjle=bessjle/summ
bessjle_v=unpack(bessjle,mask,0.0_sp)
deallocate(x,bj,bjm,bjp,summ,bessjle,tox,renorm)
END FUNCTION bessjle_v
END FUNCTION bessj_v

```

f90 `bessj_v=...` `bessj_v=merge(bessjgt_v(...),bessj_v,...)` The vector `bessj_v` is set once (with a mask) and then merged with *itself*, along with the vector result of the `bessjgt_v` call. Thus are the two evaluation methods combined. (A third case, where an argument is zero, is then handled by an immediately following where.)

* * *

```

FUNCTION bessj0_s(x)
USE nrtype; USE nrutil, ONLY : poly
IMPLICIT NONE
REAL(SP), INTENT(IN) :: x
REAL(SP) :: bessj0_s
Returns the modified Bessel function  $I_0(x)$  for any real x.
REAL(SP) :: ax
REAL(DP), DIMENSION(7) :: p = (/1.0_dp,3.5156229_dp,&
3.0899424_dp,1.2067492_dp,0.2659732_dp,0.360768e-1_dp,&
0.45813e-2_dp/) Accumulate polynomials in double precision.
REAL(DP), DIMENSION(9) :: q = (/0.39894228_dp,0.1328592e-1_dp,&
0.225319e-2_dp,-0.157565e-2_dp,0.916281e-2_dp,&
-0.2057706e-1_dp,0.2635537e-1_dp,-0.1647633e-1_dp,&
0.392377e-2_dp/)
ax=abs(x)
if (ax < 3.75) then Polynomial fit.
bessj0_s=poly(real((x/3.75_sp)**2,dp),p)
else
bessj0_s=(exp(ax)/sqrt(ax))*poly(real(3.75_sp/ax,dp),q)
end if
END FUNCTION bessj0_s

```

```

FUNCTION bessj0_v(x)
USE nrtype; USE nrutil, ONLY : poly
IMPLICIT NONE
REAL(SP), DIMENSION(:), INTENT(IN) :: x
REAL(SP), DIMENSION(size(x)) :: bessj0_v
REAL(SP), DIMENSION(size(x)) :: ax
REAL(DP), DIMENSION(size(x)) :: y
LOGICAL(LGT), DIMENSION(size(x)) :: mask
REAL(DP), DIMENSION(7) :: p = (/1.0_dp,3.5156229_dp,&
3.0899424_dp,1.2067492_dp,0.2659732_dp,0.360768e-1_dp,&

```

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
Permission is granted for Internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one) to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

```

    0.45813e-2_dp/)
REAL(DP), DIMENSION(9) :: q = (/0.39894228_dp,0.1328592e-1_dp,&
    0.225319e-2_dp,-0.157565e-2_dp,0.916281e-2_dp,&
    -0.2057706e-1_dp,0.2635537e-1_dp,-0.1647633e-1_dp,&
    0.392377e-2_dp/)
ax=abs(x)
mask = (ax < 3.75)
where (mask)
    bessio_v=poly(real((x/3.75_sp)**2,dp),p,mask)
elsewhere
    y=3.75_sp/ax
    bessio_v=(exp(ax)/sqrt(ax))*poly(real(y,dp),q,.not. mask)
end where
END FUNCTION bessio_v

```

* * *

```

FUNCTION bessk0_s(x)
USE nrtype; USE nrutil, ONLY : assert,poly
USE nr, ONLY : bessio
IMPLICIT NONE
REAL(SP), INTENT(IN) :: x
REAL(SP) :: bessk0_s
    Returns the modified Bessel function  $K_0(x)$  for positive real x.
REAL(DP) :: y
    Accumulate polynomials in double precision.
REAL(DP), DIMENSION(7) :: p = (/ -0.57721566_dp, 0.42278420_dp, &
    0.23069756_dp, 0.3488590e-1_dp, 0.262698e-2_dp, 0.10750e-3_dp, &
    0.74e-5_dp/)
REAL(DP), DIMENSION(7) :: q = (/ 1.25331414_dp, -0.7832358e-1_dp, &
    0.2189568e-1_dp, -0.1062446e-1_dp, 0.587872e-2_dp, &
    -0.251540e-2_dp, 0.53208e-3_dp/)
call assert(x > 0.0, 'bessk0_s arg')
if (x <= 2.0) then
    Polynomial fit.
    y=x*x/4.0_sp
    bessk0_s=(-log(x/2.0_sp)*bessio(x))+poly(y,p)
else
    y=(2.0_sp/x)
    bessk0_s=(exp(-x)/sqrt(x))*poly(y,q)
end if
END FUNCTION bessk0_s

```

```

FUNCTION bessk0_v(x)
USE nrtype; USE nrutil, ONLY : assert,poly
USE nr, ONLY : bessio
IMPLICIT NONE
REAL(SP), DIMENSION(:), INTENT(IN) :: x
REAL(SP), DIMENSION(size(x)) :: bessk0_v
REAL(DP), DIMENSION(size(x)) :: y
LOGICAL(LGT), DIMENSION(size(x)) :: mask
REAL(DP), DIMENSION(7) :: p = (/ -0.57721566_dp, 0.42278420_dp, &
    0.23069756_dp, 0.3488590e-1_dp, 0.262698e-2_dp, 0.10750e-3_dp, &
    0.74e-5_dp/)
REAL(DP), DIMENSION(7) :: q = (/ 1.25331414_dp, -0.7832358e-1_dp, &
    0.2189568e-1_dp, -0.1062446e-1_dp, 0.587872e-2_dp, &
    -0.251540e-2_dp, 0.53208e-3_dp/)
call assert(all(x > 0.0), 'bessk0_v arg')
mask = (x <= 2.0)
where (mask)
    y=x*x/4.0_sp
    bessk0_v=(-log(x/2.0_sp)*bessio(x))+poly(y,p,mask)

```

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for Internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one) to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

```

elsewhere
  y=(2.0_sp/x)
  bessk0_v=(exp(-x)/sqrt(x))*poly(y,q,.not. mask)
end where
END FUNCTION bessk0_v

```

* * *

```

FUNCTION bess1_s(x)
USE nrtype; USE nrutil, ONLY : poly
IMPLICIT NONE
REAL(SP), INTENT(IN) :: x
REAL(SP) :: bess1_s
  Returns the modified Bessel function  $I_1(x)$  for any real x.
REAL(SP) :: ax
REAL(DP), DIMENSION(7) :: p = (/0.5_dp,0.87890594_dp,&
  0.51498869_dp,0.15084934_dp,0.2658733e-1_dp,&
  0.301532e-2_dp,0.32411e-3_dp/)
  Accumulate polynomials in double precision.
REAL(DP), DIMENSION(9) :: q = (/0.39894228_dp,-0.3988024e-1_dp,&
  -0.362018e-2_dp,0.163801e-2_dp,-0.1031555e-1_dp,&
  0.2282967e-1_dp,-0.2895312e-1_dp,0.1787654e-1_dp,&
  -0.420059e-2_dp/)
ax=abs(x)
if (ax < 3.75) then      Polynomial fit.
  bess1_s=ax*poly(real((x/3.75_sp)**2,dp),p)
else
  bess1_s=(exp(ax)/sqrt(ax))*poly(real(3.75_sp/ax,dp),q)
end if
if (x < 0.0) bess1_s=-bess1_s
END FUNCTION bess1_s

```

```

FUNCTION bess1_v(x)
USE nrtype; USE nrutil, ONLY : poly
IMPLICIT NONE
REAL(SP), DIMENSION(:), INTENT(IN) :: x
REAL(SP), DIMENSION(size(x)) :: bess1_v
REAL(SP), DIMENSION(size(x)) :: ax
REAL(DP), DIMENSION(size(x)) :: y
LOGICAL(LGT), DIMENSION(size(x)) :: mask
REAL(DP), DIMENSION(7) :: p = (/0.5_dp,0.87890594_dp,&
  0.51498869_dp,0.15084934_dp,0.2658733e-1_dp,&
  0.301532e-2_dp,0.32411e-3_dp/)
REAL(DP), DIMENSION(9) :: q = (/0.39894228_dp,-0.3988024e-1_dp,&
  -0.362018e-2_dp,0.163801e-2_dp,-0.1031555e-1_dp,&
  0.2282967e-1_dp,-0.2895312e-1_dp,0.1787654e-1_dp,&
  -0.420059e-2_dp/)
ax=abs(x)
mask = (ax < 3.75)
where (mask)
  bess1_v=ax*poly(real((x/3.75_sp)**2,dp),p,mask)
elsewhere
  y=3.75_sp/ax
  bess1_v=(exp(ax)/sqrt(ax))*poly(real(y,dp),q,.not. mask)
end where
where (x < 0.0) bess1_v=-bess1_v
END FUNCTION bess1_v

```

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-
 readable files (including this one), to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs
 visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

* * *

```

FUNCTION bessk1_s(x)
USE nrtype; USE nrutil, ONLY : assert,poly
USE nr, ONLY : bess1
IMPLICIT NONE
REAL(SP), INTENT(IN) :: x
REAL(SP) :: bessk1_s
  Returns the modified Bessel function  $K_1(x)$  for positive real x.
REAL(DP) :: y          Accumulate polynomials in double precision.
REAL(DP), DIMENSION(7) :: p = (/1.0_dp,0.15443144_dp,&
-0.67278579_dp,-0.18156897_dp,-0.1919402e-1_dp,&
-0.110404e-2_dp,-0.4686e-4_dp/)
REAL(DP), DIMENSION(7) :: q = (/1.25331414_dp,0.23498619_dp,&
-0.3655620e-1_dp,0.1504268e-1_dp,-0.780353e-2_dp,&
0.325614e-2_dp,-0.68245e-3_dp/)
call assert(x > 0.0, 'bessk1_s arg')
if (x <= 2.0) then      Polynomial fit.
  y=x*x/4.0_sp
  bessk1_s=(log(x/2.0_sp)*bess1(x))+(1.0_sp/x)*poly(y,p)
else
  y=2.0_sp/x
  bessk1_s=(exp(-x)/sqrt(x))*poly(y,q)
end if
END FUNCTION bessk1_s

```

```

FUNCTION bessk1_v(x)
USE nrtype; USE nrutil, ONLY : assert,poly
USE nr, ONLY : bess1
IMPLICIT NONE
REAL(SP), DIMENSION(:), INTENT(IN) :: x
REAL(SP), DIMENSION(size(x)) :: bessk1_v
REAL(DP), DIMENSION(size(x)) :: y
LOGICAL(LGT), DIMENSION(size(x)) :: mask
REAL(DP), DIMENSION(7) :: p = (/1.0_dp,0.15443144_dp,&
-0.67278579_dp,-0.18156897_dp,-0.1919402e-1_dp,&
-0.110404e-2_dp,-0.4686e-4_dp/)
REAL(DP), DIMENSION(7) :: q = (/1.25331414_dp,0.23498619_dp,&
-0.3655620e-1_dp,0.1504268e-1_dp,-0.780353e-2_dp,&
0.325614e-2_dp,-0.68245e-3_dp/)
call assert(all(x > 0.0), 'bessk1_v arg')
mask = (x <= 2.0)
where (mask)
  y=x*x/4.0_sp
  bessk1_v=(log(x/2.0_sp)*bess1(x))+(1.0_sp/x)*poly(y,p,mask)
elsewhere
  y=2.0_sp/x
  bessk1_v=(exp(-x)/sqrt(x))*poly(y,q,.not. mask)
end where
END FUNCTION bessk1_v

```

* * *

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one), to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

```

FUNCTION bessk_s(n,x)
USE nrtype; USE nrutil, ONLY : assert
USE nr, ONLY : bessk0,bessk1
IMPLICIT NONE
INTEGER(I4B), INTENT(IN) :: n
REAL(SP), INTENT(IN) :: x
REAL(SP) :: bessk_s
  Returns the modified Bessel function  $K_n(x)$  for positive  $x$  and  $n \geq 2$ .
INTEGER(I4B) :: j
REAL(SP) :: bk,bkm,bkp,tox
call assert(n >= 2, x > 0.0, 'bessk_s args')
tox=2.0_sp/x
bkm=bessk0(x)           Upward recurrence for all x...
bk=bessk1(x)           ...and here it is.
do j=1,n-1
  bkp=bkm+j*tox*bk
  bkm=bk
  bk=bkp
end do
bessk_s=bk
END FUNCTION bessk_s

```

```

FUNCTION bessk_v(n,x)
USE nrtype; USE nrutil, ONLY : assert
USE nr, ONLY : bessk0,bessk1
IMPLICIT NONE
INTEGER(I4B), INTENT(IN) :: n
REAL(SP), DIMENSION(:), INTENT(IN) :: x
REAL(SP), DIMENSION(size(x)) :: bessk_v
INTEGER(I4B) :: j
REAL(SP), DIMENSION(size(x)) :: bk,bkm,bkp,tox
call assert(n >= 2, all(x > 0.0), 'bessk_v args')
tox=2.0_sp/x
bkm=bessk0(x)
bk=bessk1(x)
do j=1,n-1
  bkp=bkm+j*tox*bk
  bkm=bk
  bk=bkp
end do
bessk_v=bk
END FUNCTION bessk_v

```



The scalar and vector versions of `bessk` are identical, and have no precision-specific constants, another example of where we would like to define a generic “template” function if the language had this facility.

* * *

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for Internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one) to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

```

FUNCTION bessj_s(n,x)
USE nrtype; USE nrutil, ONLY : assert
USE nr, ONLY : bessj0
IMPLICIT NONE
INTEGER(I4B), INTENT(IN) :: n
REAL(SP), INTENT(IN) :: x
REAL(SP) :: bessj_s
INTEGER(I4B), PARAMETER :: IACC=40, IEXP=maxexponent(x)/2
  Returns the modified Bessel function  $I_n(x)$  for any real  $x$  and  $n \geq 2$ . Make the parameter
  IACC larger to increase accuracy.
INTEGER(I4B) :: j,m
REAL(SP) :: bi,bim,bip,tox
call assert(n >= 2, 'bessj_s args')
bessj_s=0.0
if (x*x <= 8.0_sp*tiny(x)) RETURN          Underflow limit.
tox=2.0_sp/abs(x)
bip=0.0
bi=1.0
m=2*((n+int(sqrt(real(IACC*n,sp))))))      Downward recurrence from even m.
do j=m,1,-1
  bim=bip+j*tox*bi                          The downward recurrence.
  bip=bi
  bi=bim
  if (exponent(bi) > IEXP) then              Renormalize to prevent overflows.
    bessj_s=scale(bessj_s,-IEXP)
    bi=scale(bi,-IEXP)
    bip=scale(bip,-IEXP)
  end if
  if (j == n) bessj_s=bip
end do
bessj_s=bessj_s*bessj0(x)/bi                 Normalize with bessj0.
if (x < 0.0 .and. mod(n,2) == 1) bessj_s=-bessj_s
END FUNCTION bessj_s

```



if (exponent(bi) > IEXP) then See discussion of scaling for bessj on p. 1107.

```

FUNCTION bessj_v(n,x)
USE nrtype; USE nrutil, ONLY : assert
USE nr, ONLY : bessj0
IMPLICIT NONE
INTEGER(I4B), INTENT(IN) :: n
REAL(SP), DIMENSION(:), INTENT(IN) :: x
REAL(SP), DIMENSION(size(x)) :: bessj_v
INTEGER(I4B), PARAMETER :: IACC=40, IEXP=maxexponent(x)/2
INTEGER(I4B) :: j,m
REAL(SP), DIMENSION(size(x)) :: bi,bim,bip,tox
LOGICAL(LGT), DIMENSION(size(x)) :: mask
call assert(n >= 2, 'bessj_v args')
bessj_v=0.0
mask = (x <= 8.0_sp*tiny(x))
tox=2.0_sp/merge(2.0_sp,abs(x),mask)
bip=0.0
bi=1.0_sp
m=2*((n+int(sqrt(real(IACC*n,sp))))))
do j=m,1,-1
  bim=bip+j*tox*bi
  bip=bi
  bi=bim
  where (exponent(bi) > IEXP)
    bessj_v=scale(bessj_v,-IEXP)

```



```

        bi=scale(bi,-IEXP)
        bip=scale(bip,-IEXP)
    end where
    if (j == n) bessj_v=bip
end do
bessj_v=bessj_v*bessj0(x)/bi
where (mask) bessj_v=0.0_sp
where (x < 0.0 .and. mod(n,2) == 1) bessj_v=-bessj_v
END FUNCTION bessj_v

```



```

mask = (x == 0.0)
tox=2.0_sp/merge(2.0_sp,abs(x),mask)

```

For the special case $x = 0$, the value of the returned function should be zero; however, the evaluation of `tox` will give a divide check. We substitute an innocuous value for the zero cases, then fix up their answers at the end.

* * *

```

SUBROUTINE bessjy_s(x,xnu,rj,ry,rjp,ryp)
USE nrtype; USE nrutil, ONLY : assert,nrerror
USE nr, ONLY : beschb
IMPLICIT NONE
REAL(SP), INTENT(IN) :: x,xnu
REAL(SP), INTENT(OUT) :: rj,ry,rjp,ryp
INTEGER(I4B), PARAMETER :: MAXIT=10000
REAL(DP), PARAMETER :: XMIN=2.0_dp, EPS=1.0e-10_dp, FPMIN=1.0e-30_dp
    Returns the Bessel functions  $rj = J_\nu$ ,  $ry = Y_\nu$  and their derivatives  $rjp = J'_\nu$ ,  $ryp = Y'_\nu$ ,
    for positive  $x$  and for  $xnu = \nu \geq 0$ . The relative accuracy is within one or two significant
    digits of EPS, except near a zero of one of the functions, where EPS controls its absolute
    accuracy. FPMIN is a number close to the machine's smallest floating-point number. All
    internal arithmetic is in double precision. To convert the entire routine to double precision,
    change the SP declaration above and decrease EPS to  $10^{-16}$ . Also convert the subroutine
    beschb.
INTEGER(I4B) :: i, isign, l, nl
REAL(DP) :: a, b, c, d, del, del1, e, f, fact, fact2, fact3, ff, gam, gam1, gam2, &
    gammi, gampl, h, p, pimu, pimu2, q, r, rj1, rj11, rjmu, rjpl, rjpl, rjtemp, &
    ry1, rymu, rymup, rytemp, sum, sum1, w, x2, xi, xi2, xmu, xmu2
COMPLEX(DPC) :: aa, bb, cc, dd, dl, pq
call assert(x > 0.0, xnu >= 0.0, 'bessjy args')
nl=merge(int(xnu+0.5_dp), max(0,int(xnu-x+1.5_dp)), x < XMIN)
    nl is the number of downward recurrences of the  $J$ 's and upward recurrences of  $Y$ 's. xmu
    lies between  $-1/2$  and  $1/2$  for  $x < XMIN$ , while it is chosen so that  $x$  is greater than the
    turning point for  $x \geq XMIN$ .
xmu=xnu-nl
xmu2=xmu*xmu
xi=1.0_dp/x
xi2=2.0_dp*xi
w=xi2/PI_D
isign=1
h=xnu*xi
if (h < FPMIN) h=FPMIN
b=xi2*xnu
d=0.0
c=h
do i=1,MAXIT
    b=b+xi2
    d=b-d
    if (abs(d) < FPMIN) d=FPMIN
    c=b-1.0_dp/c

```

The Wronskian.
Evaluate CF1 by modified Lentz's method (§5.2). `isign` keeps track of sign changes in the denominator.

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-
 readable files (including this one), to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs
 visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

```

    if (abs(c) < FPMIN) c=FPMIN
    d=1.0_dp/d
    del=c*d
    h=del*h
    if (d < 0.0) isign=-isign
    if (abs(del-1.0_dp) < EPS) exit
end do
if (i > MAXIT) call nrerror('x too large in bessjy; try asymptotic expansion')
rjl=isign*FPMIN           Initialize  $J_\nu$  and  $J'_\nu$  for downward recurrence.
rjpl=h*rjl
rjl1=rjl                 Store values for later rescaling.
rjp1=rjpl
fact=xnu*xi
do l=nl,1,-1
    rjtemp=fact*rjl+rjpl
    fact=fact-xi
    rjpl=fact*rjtemp-rjl
    rjl=rjtemp
end do
if (rjl == 0.0) rjl=EPS
f=rjpl/rjl              Now have unnormalized  $J_\mu$  and  $J'_\mu$ .
if (x < XMIN) then      Use series.
    x2=0.5_dp*x
    pimu=PI_D*xmu
    if (abs(pimu) < EPS) then
        fact=1.0
    else
        fact=pimu/sin(pimu)
    end if
    d=-log(x2)
    e=xmu*d
    if (abs(e) < EPS) then
        fact2=1.0
    else
        fact2=sinh(e)/e
    end if
    call beschb(xmu,gam1,gam2,gampl,gammi)    Chebyshev evaluation of  $\Gamma_1$  and  $\Gamma_2$ .
    ff=2.0_dp/PI_D*fact*(gam1*cosh(e)+gam2*fact2*d)     $f_0$ .
    e=exp(e)
    p=e/(gampl*PI_D)     $p_0$ .
    q=1.0_dp/(e*PI_D*gammi)     $q_0$ .
    pimu2=0.5_dp*pimu
    if (abs(pimu2) < EPS) then
        fact3=1.0
    else
        fact3=sin(pimu2)/pimu2
    end if
    r=PI_D*pimu2*fact3*fact3
    c=1.0
    d=-x2*x2
    sum=ff+r*q
    sum1=p
    do i=1,MAXIT
        ff=(i*ff+p*q)/(i*xmu2)
        c=c*d/i
        p=p/(i-xmu)
        q=q/(i+xmu)
        del=c*(ff+r*q)
        sum=sum+del
        del1=c*p-i*del
        sum1=sum1+del1
        if (abs(del) < (1.0_dp+abs(sum))*EPS) exit
    end do
if (i > MAXIT) call nrerror('bessy series failed to converge')

```

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one), to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

```

rymu=-sum
ry1=-sum1*xi2
rymup=xmu*xi*rymu-ry1
rjmu=w/(rymup-f*rymu)
else
a=0.25_dp-xmu2
pq=cplx(-0.5_dp*xi,1.0_dp,kind=dpc)
aa=cplx(0.0_dp,xi*a,kind=dpc)
bb=cplx(2.0_dp*x,2.0_dp,kind=dpc)
cc=bb+aa/pq
dd=1.0_dp/bb
pq=cc*dd*pq
do i=2,MAXIT
  a=a+2*(i-1)
  bb=bb+cplx(0.0_dp,2.0_dp,kind=dpc)
  dd=a*dd+bb
  if (absc(dd) < FPMIN) dd=FPMIN
  cc=bb+a/cc
  if (absc(cc) < FPMIN) cc=FPMIN
  dd=1.0_dp/dd
  dl=cc*dd
  pq=pq*dl
  if (absc(dl-1.0_dp) < EPS) exit
end do
if (i > MAXIT) call nrerror('cf2 failed in bessjy')
p=real(pq)
q=aimag(pq)
gam=(p-f)/q
rjmu=sqrt(w/((p-f)*gam+q))
rjmu=sign(rjmu,rj1)
rymu=rjmu*gam
rymup=rymu*(p+q/gam)
ry1=xmu*xi*rymu-rymup
end if
fact=rjmu/rj1
rj=rj1*fact
rjp=rjp1*fact
do i=1,nl
  rytemp=(xmu+i)*xi2*ry1-rymu
  rymu=ry1
  ry1=rytemp
end do
ry=rymu
ryp=xnu*xi*rymu-ry1
CONTAINS

FUNCTION absc(z)
IMPLICIT NONE
COMPLEX(DPC), INTENT(IN) :: z
REAL(DP) :: absc
absc=abs(real(z))+abs(aimag(z))
END FUNCTION absc
END SUBROUTINE bessjy_s

```

Equation (6.7.13).
Evaluate CF2 by modified Lentz's method (§5.2).

Equations (6.7.6) – (6.7.10).

Scale original J_ν and J'_ν .

Upward recurrence of Y_ν .



Yes there is a vector version `bessjy_v`. Its general scheme is to have a bunch of contained functions for various cases, and then combine their outputs (somewhat like `bessj_v`, above, but much more complicated). A listing runs to about four printed pages, and we judge it to be of not much interest, so we will not include it here. (It is included on the machine-readable media.)

* * *

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for Internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one) to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

```

SUBROUTINE beschb_s(x,gam1,gam2,gampl,gammi)
USE nrtype
USE nr, ONLY : chebev
IMPLICIT NONE
REAL(DP), INTENT(IN) :: x
REAL(DP), INTENT(OUT) :: gam1,gam2,gampl,gammi
INTEGER(I4B), PARAMETER :: NUSE1=5,NUSE2=5
  Evaluates  $\Gamma_1$  and  $\Gamma_2$  by Chebyshev expansion for  $|x| \leq 1/2$ . Also returns  $1/\Gamma(1+x)$  and
   $1/\Gamma(1-x)$ . If converting to double precision, set NUSE1 = 7, NUSE2 = 8.
REAL(SP) :: xx
REAL(SP), DIMENSION(7) :: c1=(-1.142022680371168_sp,&
  6.5165112670737e-3_sp,3.087090173086e-4_sp,-3.4706269649e-6_sp,&
  6.9437664e-9_sp,3.67795e-11_sp,-1.356e-13_sp/)
REAL(SP), DIMENSION(8) :: c2=(/1.843740587300905_sp,&
  -7.68528408447867e-2_sp,1.2719271366546e-3_sp,&
  -4.9717367042e-6_sp,-3.31261198e-8_sp,2.423096e-10_sp,&
  -1.702e-13_sp,-1.49e-15_sp/)
xx=8.0_dp*x*x-1.0_dp      Multiply x by 2 to make range be -1 to 1, and then apply
gam1=chebev(-1.0_sp,1.0_sp,c1(1:NUSE1),xx)      transformation for evaluating even Cheby-
gam2=chebev(-1.0_sp,1.0_sp,c2(1:NUSE2),xx)      shev series.
gampl=gam2-x*gam1
gammi=gam2+x*gam1
END SUBROUTINE beschb_s

```

```

SUBROUTINE beschb_v(x,gam1,gam2,gampl,gammi)
USE nrtype
USE nr, ONLY : chebev
IMPLICIT NONE
REAL(DP), DIMENSION(:), INTENT(IN) :: x
REAL(DP), DIMENSION(:), INTENT(OUT) :: gam1,gam2,gampl,gammi
INTEGER(I4B), PARAMETER :: NUSE1=5,NUSE2=5
REAL(SP), DIMENSION(size(x)) :: xx
REAL(SP), DIMENSION(7) :: c1=(-1.142022680371168_sp,&
  6.5165112670737e-3_sp,3.087090173086e-4_sp,-3.4706269649e-6_sp,&
  6.9437664e-9_sp,3.67795e-11_sp,-1.356e-13_sp/)
REAL(SP), DIMENSION(8) :: c2=(/1.843740587300905_sp,&
  -7.68528408447867e-2_sp,1.2719271366546e-3_sp,&
  -4.9717367042e-6_sp,-3.31261198e-8_sp,2.423096e-10_sp,&
  -1.702e-13_sp,-1.49e-15_sp/)
xx=8.0_dp*x*x-1.0_dp
gam1=chebev(-1.0_sp,1.0_sp,c1(1:NUSE1),xx)
gam2=chebev(-1.0_sp,1.0_sp,c2(1:NUSE2),xx)
gampl=gam2-x*gam1
gammi=gam2+x*gam1
END SUBROUTINE beschb_v

```

* * *

```

SUBROUTINE bessik(x,xnu,ri,rk,rip,rkp)
USE nrtype; USE nrutil, ONLY : assert,nrerror
USE nr, ONLY : beschb
IMPLICIT NONE
REAL(SP), INTENT(IN) :: x,xnu
REAL(SP), INTENT(OUT) :: ri,rk,rip,rkp
INTEGER(I4B), PARAMETER :: MAXIT=10000
REAL(SP), PARAMETER :: XMIN=2.0
REAL(DP), PARAMETER :: EPS=1.0e-10_dp,FPMIN=1.0e-30_dp
  Returns the modified Bessel functions  $ri = I_\nu$ ,  $rk = K_\nu$  and their derivatives  $rip = I'_\nu$ ,
   $rkp = K'_\nu$ , for positive x and for  $xnu = \nu \geq 0$ . The relative accuracy is within one or

```

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-
 readable files (including this one), to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs
 visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

two significant digits of EPS. FPMIN is a number close to the machine's smallest floating-point number. All internal arithmetic is in double precision. To convert the entire routine to double precision, change the REAL declaration above and decrease EPS to 10^{-16} . Also convert the subroutine beschb.

```

INTEGER(I4B) :: i,l,nl
REAL(DP) :: a,a1,b,c,d,del,del1,delh,dels,e,f,fact,fact2,ff,&
    gam1,gam2,gammi,gampl,h,p,pimu,q,q1,q2,qnew,&
    ril,ril1,rimu,ripl,ripl,ritemp,rk1,rkmu,rkmup,rktemp,&
    s,sum,sum1,x2,xi,xi2,xmu,xmu2
call assert(x > 0.0, xnu >= 0.0, 'bessik args')
nl=int(xnu+0.5_dp)
xmu=xnu-nl
xmu2=xmu*xmu
xi=1.0_dp/x
xi2=2.0_dp*xi
h=xnu*xi
if (h < FPMIN) h=FPMIN
b=xi2*xnu
d=0.0
c=h
do i=1,MAXIT
    b=b+xi2
    d=1.0_dp/(b+d)
    c=b+1.0_dp/c
    del=c*d
    h=del*h
    if (abs(del-1.0_dp) < EPS) exit
end do
if (i > MAXIT) call nrerror('x too large in bessik; try asymptotic expansion')
ril=FPMIN
ripl=h*ril
ril1=ril
ripl1=ripl
fact=xnu*xi
do l=nl,1,-1
    ritemp=fact*ril+ripl
    fact=fact-xi
    ripl=fact*ritemp+ril
    ril=ritemp
end do
f=ripl/ril
if (x < XMIN) then
    x2=0.5_dp*x
    pimiu=PI_D*xmu
    if (abs(pimiu) < EPS) then
        fact=1.0
    else
        fact=pimiu/sin(pimiu)
    end if
    d=-log(x2)
    e=xmu*d
    if (abs(e) < EPS) then
        fact2=1.0
    else
        fact2=sinh(e)/e
    end if
    call beschb(xmu,gam1,gam2,gampl,gammi)
    ff=fact*(gam1*cosh(e)+gam2*fact2*d)
    sum=ff
    e=exp(e)
    p=0.5_dp*e/gampl
    q=0.5_dp/(e*gammi)
    c=1.0
    d=x2*x2

```

nl is the number of downward recurrences of the I 's and upward recurrences of K 's. xmu lies between $-1/2$ and $1/2$.

Evaluate CF1 by modified Lentz's method (§5.2).

Denominators cannot be zero here, so no need for special precautions.

Initialize I_ν and I'_ν for downward recurrence.

Store values for later rescaling.

Now have unnormalized I_μ and I'_μ . Use series.

Chebyshev evaluation of Γ_1 and Γ_2 .

f_0 .

p_0 .

q_0 .

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for Internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one) to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

```

sum1=p
do i=1,MAXIT
  ff=(i*ff+p+q)/(i*i-xmu2)
  c=c*d/i
  p=p/(i-xmu)
  q=q/(i+xmu)
  del=c*ff
  sum=sum+del
  del1=c*(p-i*ff)
  sum1=sum1+del1
  if (abs(del) < abs(sum)*EPS) exit
end do
if (i > MAXIT) call nrerror('bessk series failed to converge')
rkmu=sum
rk1=sum1*xi2
else
  Evaluate CF2 by Steed's algorithm (§5.2),
  which is OK because there can be no
  zero denominators.

  b=2.0_dp*(1.0_dp+x)
  d=1.0_dp/b
  delh=d
  h=delh
  q1=0.0
  q2=1.0
  a1=0.25_dp-xmu2
  c=a1
  q=c
  a=-a1
  s=1.0_dp+q*delh
  do i=2,MAXIT
    a=a-2*(i-1)
    c=-a*c/i
    qnew=(q1-b*q2)/a
    q1=q2
    q2=qnew
    q=q+c*qnew
    b=b+2.0_dp
    d=1.0_dp/(b+a*d)
    delh=(b*d-1.0_dp)*delh
    h=h+delh
    dels=q*delh
    s=s+dels
    if (abs(dels/s) < EPS) exit
  end do
  if (i > MAXIT) call nrerror('bessik: failure to converge in cf2')
  h=a1*h
  rkmu=sqrt(PI_D/(2.0_dp*x))*exp(-x)/s
  rk1=rkmu*(xmu+x+0.5_dp-h)*xi
  Omit the factor  $\exp(-x)$  to scale all the
  returned functions by  $\exp(x)$  for  $x \geq$ 
  XMIN.

  end if
  rkmup=xmu*xi*rkmu-rk1
  rimu=xi/(f*rkmu-rkmup)
  Get  $I_\mu$  from Wronskian.
  Scale original  $I_\nu$  and  $I'_\nu$ .
  ri=(rimu*ril1)/ril
  rip=(rimu*rip1)/ril
  do i=1,nl
    rktemp=(xmu+i)*xi2*rk1+rkmu
    rkmu=rk1
    rk1=rktemp
  end do
  rkp=xnu*xi*rkmu-rk1
  Upward recurrence of  $K_\nu$ .
END SUBROUTINE bessik

```

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for Internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-
 readable files (including this one) to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs
 visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).



bessik does not readily parallelize, and we thus don't provide a vector version. Since airy, immediately following, requires bessik, we don't have a vector version of it, either.

* * *

```

SUBROUTINE airy(x,ai,bi,aip,bip)
USE nrtype
USE nr, ONLY : bessik,bessjy
IMPLICIT NONE
REAL(SP), INTENT(IN) :: x
REAL(SP), INTENT(OUT) :: ai,bi,aip,bip
  Returns Airy functions  $Ai(x)$ ,  $Bi(x)$ , and their derivatives  $Ai'(x)$ ,  $Bi'(x)$ .
REAL(SP) :: absx,ri,rip,rj,rjp,rk,rkp,rootx,ry,ryp,z
REAL(SP), PARAMETER :: THIRD=1.0_sp/3.0_sp,TWOTHR=2.0_sp/3.0_sp, &
  ONOVRT=0.5773502691896258_sp
absx=abs(x)
rootx=sqrt(absx)
z=TWOTHR*absx*rootx
if (x > 0.0) then
  call bessik(z,THIRD,ri,rk,rip,rkp)
  ai=rootx*ONOVRT*rk/PI
  bi=rootx*(rk/PI+2.0_sp*ONOVRT*ri)
  call bessik(z,TWOTHR,ri,rk,rip,rkp)
  aip=-x*ONOVRT*rk/PI
  bip=x*(rk/PI+2.0_sp*ONOVRT*ri)
else if (x < 0.0) then
  call bessjy(z,THIRD,rj,ry,rjp,ryp)
  ai=0.5_sp*rootx*(rj-ONOVRT*ry)
  bi=-0.5_sp*rootx*(ry+ONOVRT*rj)
  call bessjy(z,TWOTHR,rj,ry,rjp,ryp)
  aip=0.5_sp*absx*(ONOVRT*ry+rj)
  bip=0.5_sp*absx*(ONOVRT*rj-ry)
else
  Case  $x = 0$ .
  ai=0.3550280538878172_sp
  bi=ai/ONOVRT
  aip=-0.2588194037928068_sp
  bip=-aip/ONOVRT
end if
END SUBROUTINE airy

```

* * *

```

SUBROUTINE sphbes_s(n,x,sj,sy,sjp,syp)
USE nrtype; USE nrutil, ONLY : assert
USE nr, ONLY : bessjy
IMPLICIT NONE
INTEGER(I4B), INTENT(IN) :: n
REAL(SP), INTENT(IN) :: x
REAL(SP), INTENT(OUT) :: sj,sy,sjp,syp
  Returns spherical Bessel functions  $j_n(x)$ ,  $y_n(x)$ , and their derivatives  $j'_n(x)$ ,  $y'_n(x)$  for
  integer  $n \geq 0$  and  $x > 0$ .
REAL(SP), PARAMETER :: RTPIO2=1.253314137315500_sp
REAL(SP) :: factor,order,rj,rjp,ry,ryp
call assert(n >= 0, x > 0.0, 'sphbes_s args')
order=n+0.5_sp
call bessjy(x,order,rj,ry,rjp,ryp)
factor=RTPIO2/sqrt(x)
sj=factor*rj
sy=factor*ry

```

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one), to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

```

sjp=factor*rjp-sj/(2.0_sp*x)
syp=factor*ryp-sy/(2.0_sp*x)
END SUBROUTINE sphbes_s

SUBROUTINE sphbes_v(n,x,sj,sy,sjp,syp)
USE nrtype; USE nrutil, ONLY : assert
USE nr, ONLY : bessjy
IMPLICIT NONE
INTEGER(I4B), INTENT(IN) :: n
REAL(SP), DIMENSION(:), INTENT(IN) :: x
REAL(SP), DIMENSION(:), INTENT(OUT) :: sj,sy,sjp,syp
REAL(SP), PARAMETER :: RTPIO2=1.253314137315500_sp
REAL(SP) :: order
REAL(SP), DIMENSION(size(x)) :: factor,rj,rjp,ry,ryp
call assert(n >= 0, all(x > 0.0), 'sphbes_v args')
order=n+0.5_sp
call bessjy(x,order,rj,ry,rjp,ryp)
factor=RTPIO2/sqrt(x)
sj=factor*rj
sy=factor*ry
sjp=factor*rjp-sj/(2.0_sp*x)
syp=factor*ryp-sy/(2.0_sp*x)
END SUBROUTINE sphbes_v

```



Note that `sphbes_v` uses (through overloading) `bessjy_v`. The listing of that routine was omitted above, but it is on the machine-readable media.

* * *

```

FUNCTION plgndr_s(l,m,x)
USE nrtype; USE nrutil, ONLY : arth,assert
IMPLICIT NONE
INTEGER(I4B), INTENT(IN) :: l,m
REAL(SP), INTENT(IN) :: x
REAL(SP) :: plgndr_s
  Computes the associated Legendre polynomial  $P_l^m(x)$ . Here  $m$  and  $l$  are integers satisfying
   $0 \leq m \leq l$ , while  $x$  lies in the range  $-1 \leq x \leq 1$ .
INTEGER(I4B) :: ll
REAL(SP) :: pll,pmm,pmmp1,somx2
call assert(m >= 0, m <= l, abs(x) <= 1.0, 'plgndr_s args')
pmm=1.0          Compute  $P_m^m$ .
if (m > 0) then
  somx2=sqrt((1.0_sp-x)*(1.0_sp+x))
  pmm=product(arth(1.0_sp,2.0_sp,m))*somx2**m
  if (mod(m,2) == 1) pmm=-pmm
end if
if (l == m) then
  plgndr_s=pmm
else
  pmmp1=x*(2*m+1)*pmm          Compute  $P_{m+1}^m$ .
  if (l == m+1) then
    plgndr_s=pmmp1
  else
    Compute  $P_l^m, l > m + 1$ .
    do ll=m+2,l
      pll=(x*(2*ll-1)*pmmp1-(ll+m-1)*pmm)/(ll-m)
      pmm=pmmp1
      pmmp1=pll
    end do
    plgndr_s=pll
  end if
end if

```

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one), to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).


```

    end if
end if
END FUNCTION plgndr_s

```



product(arth(1.0_sp,2.0_sp,m))
That is, $(2m - 1)!!$

```

FUNCTION plgndr_v(l,m,x)
USE nrtype; USE nrutil, ONLY : arth,assert
IMPLICIT NONE
INTEGER(I4B), INTENT(IN) :: l,m
REAL(SP), DIMENSION(:), INTENT(IN) :: x
REAL(SP), DIMENSION(size(x)) :: plgndr_v
INTEGER(I4B) :: ll
REAL(SP), DIMENSION(size(x)) :: pll,pmm,pmmp1,somx2
call assert(m >= 0, m <= 1, all(abs(x) <= 1.0), 'plgndr_v args')
pmm=1.0
if (m > 0) then
    somx2=sqrt((1.0_sp-x)*(1.0_sp+x))
    pmm=product(arth(1.0_sp,2.0_sp,m))*somx2**m
    if (mod(m,2) == 1) pmm=-pmm
end if
if (l == m) then
    plgndr_v=pmm
else
    pmmp1=x*(2*m+1)*pmm
    if (l == m+1) then
        plgndr_v=pmmp1
    else
        do ll=m+2,l
            pll=(x*(2*ll-1)*pmmp1-(ll+m-1)*pmm)/(ll-m)
            pmm=pmmp1
            pmmp1=pll
        end do
        plgndr_v=pll
    end if
end if
END FUNCTION plgndr_v

```



All those if's (not where's) may strike you as odd in a vector routine, but it is vectorized only on x , the dependent variable, not on the scalar indices l and m . Much harder to write a routine that is parallel for a vector of arbitrary triplets (l, m, x) . Try it!

* * *

```

SUBROUTINE frenel(x,s,c)
USE nrtype; USE nrutil, ONLY : nrerror
IMPLICIT NONE
REAL(SP), INTENT(IN) :: x
REAL(SP), INTENT(OUT) :: s,c
INTEGER(I4B), PARAMETER :: MAXIT=100
REAL(SP), PARAMETER :: EPS=epsilon(x),FPMIN=tiny(x),BIG=huge(x)*EPS,&
    XMIN=1.5

```

Computes the Fresnel integrals $S(x)$ and $C(x)$ for all real x .

Parameters: MAXIT is the maximum number of iterations allowed; EPS is the relative error; FPMIN is a number near the smallest representable floating-point number; BIG is a number

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one), to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

```

near the machine overflow limit; XMIN is the dividing line between using the series and
continued fraction.
INTEGER(I4B) :: k,n
REAL(SP) :: a,ax, fact,pix2,sign,sum,sumc,sums,term,test
COMPLEX(SPC) :: b,cc,d,h,del,cs
LOGICAL(LGT) :: odd
ax=abs(x)
if (ax < sqrt(FPMIN)) then           Special case: avoid failure of convergence test be-
    s=0.0                             cause of underflow.
    c=ax
else if (ax <= XMIN) then           Evaluate both series simultaneously.
    sum=0.0
    sums=0.0
    sumc=ax
    sign=1.0
    fact=PI02*ax*ax
    odd=.true.
    term=ax
    n=3
    do k=1,MAXIT
        term=term*fact/k
        sum=sum+sign*term/n
        test=abs(sum)*EPS
        if (odd) then
            sign=-sign
            sums=sum
            sum=sumc
        else
            sumc=sum
            sum=sums
        end if
        if (term < test) exit
        odd=.not. odd
        n=n+2
    end do
    if (k > MAXIT) call nrerror('frenel: series failed')
    s=sums
    c=sumc
else                                 Evaluate continued fraction by modified Lentz's method
    pix2=PI*ax*ax                    (§5.2).
    b=cplx(1.0_sp,-pix2,kind=spc)
    cc=BIG
    d=1.0_sp/b
    h=d
    n=-1
    do k=2,MAXIT
        n=n+2
        a=-n*(n+1)
        b=b+4.0_sp
        d=1.0_sp/(a*d+b)             Denominators cannot be zero.
        cc=b+a/cc
        del=cc*d
        h=h*del
        if (absc(del-1.0_sp) <= EPS) exit
    end do
    if (k > MAXIT) call nrerror('cf failed in frenel')
    h=cplx(ax,-ax,kind=spc)
    cs=cplx(0.5_sp,0.5_sp,kind=spc)*(1.0_sp-&
        cplx(cos(0.5_sp*pix2),sin(0.5_sp*pix2),kind=spc)*h)
    c=real(cs)
    s=aimag(cs)
end if
if (x < 0.0) then                   Use antisymmetry.
    c=-c

```

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for Internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-
 readable files (including this one), to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs
 visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

```

      s=-s
end if
CONTAINS
FUNCTION absc(z)
IMPLICIT NONE
COMPLEX(SPC), INTENT(IN) :: z
REAL(SP) :: absc
absc=abs(real(z))+abs(aimag(z))
END FUNCTION absc
END SUBROUTINE frenel

```

f90 `b=cplx(1.0_sp,-pix2,kind=spc)` It's a good idea *always* to include the `kind=` parameter when you use the `cplx` intrinsic. The reason is that, perhaps counterintuitively, the result of `cplx` is not determined by the kind of its arguments, but is rather the “default complex kind.” Since that default may not be what you think it is (or what `spc` is defined to be), the desired kind should be specified explicitly.

`c=real(cs)` And why not specify a `kind=` parameter here, where it is also optionally allowed? Our answer is that the `real` intrinsic actually merges two different usages. When its argument is complex, it is the counterpart of `aimag` and returns a value whose kind is determined by the kind of its argument. In fact `aimag` doesn't even allow an optional kind parameter, so we never put one in the corresponding use of `real`. The other usage of `real` is for “casting,” that is, converting one real type to another (e.g., double precision to single precision, or vice versa). Here we *always* include a kind parameter, since otherwise the result is the default real kind, with the same dangers mentioned in the previous paragraph.

* * *

```

SUBROUTINE cisi(x,ci,si)
USE nrtype; USE nrutil, ONLY : nrerror
IMPLICIT NONE
REAL(SP), INTENT(IN) :: x
REAL(SP), INTENT(OUT) :: ci,si
INTEGER(I4B), PARAMETER :: MAXIT=100
REAL(SP), PARAMETER :: EPS=epsilon(x),FPMIN=4.0_sp*tiny(x),&
  BIG=huge(x)*EPS,TMIN=2.0
  Computes the cosine and sine integrals  $Ci(x)$  and  $Si(x)$ .  $Ci(0)$  is returned as a large negative
  number and no error message is generated. For  $x < 0$  the routine returns  $Ci(-x)$  and you
  must supply the  $-i\pi$  yourself.
  Parameters: MAXIT is the maximum number of iterations allowed; EPS is the relative error,
  or absolute error near a zero of  $Ci(x)$ ; FPMIN is a number near the smallest representable
  floating-point number; BIG is a number near the machine overflow limit; TMIN is the dividing
  line between using the series and continued fraction; EULER =  $\gamma$  (in nrtype).
INTEGER(I4B) :: i,k
REAL(SP) :: a,err,fact,sign,sum,sumc,sums,t,term
COMPLEX(SPC) :: h,b,c,d,del
LOGICAL(LGT) :: odd
t=abs(x)
if (t == 0.0) then                               Special case.
  si=0.0
  ci=-BIG
  RETURN
end if
if (t > TMIN) then                               Evaluate continued fraction by modified Lentz's
  b=cplx(1.0_sp,t,kind=spc)                       method (§5.2).

```

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-
 readable files (including this one), to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs
 visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

```

c=BIG
d=1.0_sp/b
h=d
do i=2,MAXIT
  a=-(i-1)**2
  b=b+2.0_sp
  d=1.0_sp/(a*d+b)          Denominators cannot be zero.
  c=b+a/c
  del=c*d
  h=h*del
  if (absc(del-1.0_sp) <= EPS) exit
end do
if (i > MAXIT) call nrerror('continued fraction failed in cisi')
h=cplx(cos(t),-sin(t),kind=spc)*h
ci=-real(h)
si=PI02+aimag(h)
else
  Evaluate both series simultaneously.
  Special case: avoid failure of convergence test
  because of underflow.
  if (t < sqrt(FPMIN)) then
    sumc=0.0
    sums=t
  else
    sum=0.0
    sums=0.0
    sumc=0.0
    sign=1.0
    fact=1.0
    odd=.true.
    do k=1,MAXIT
      fact=fact*t/k
      term=fact/k
      sum=sum+sign*term
      err=term/abs(sum)
      if (odd) then
        sign=-sign
        sums=sum
        sum=sumc
      else
        sumc=sum
        sum=sums
      end if
      if (err < EPS) exit
      odd=.not. odd
    end do
    if (k > MAXIT) call nrerror('MAXIT exceeded in cisi')
  end if
  si=sums
  ci=sumc+log(t)+EULER
end if
if (x < 0.0) si=-si
CONTAINS
FUNCTION absc(z)
IMPLICIT NONE
COMPLEX(SPC), INTENT(IN) :: z
REAL(SP) :: absc
absc=abs(real(z))+abs(aimag(z))
END FUNCTION absc
END SUBROUTINE cisi

```

* * *

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one), to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

```

FUNCTION dawson_s(x)
USE nrtype; USE nrutil, ONLY : arth,geop
IMPLICIT NONE
REAL(SP), INTENT(IN) :: x
REAL(SP) :: dawson_s
  Returns Dawson's integral  $F(x) = \exp(-x^2) \int_0^x \exp(t^2) dt$  for any real  $x$ .
INTEGER(I4B), PARAMETER :: NMAX=6
REAL(SP), PARAMETER :: H=0.4_sp, A1=2.0_sp/3.0_sp, A2=0.4_sp, &
  A3=2.0_sp/7.0_sp
INTEGER(I4B) :: i, n0
REAL(SP) :: ec, x2, xp, xx
REAL(SP), DIMENSION(NMAX) :: d1, d2, e1
REAL(SP), DIMENSION(NMAX), SAVE :: c=(/ (0.0_sp, i=1, NMAX) /)
if (c(1) == 0.0) c(1:NMAX)=exp(-(arth(1,2,NMAX)*H)**2)
  Initialize c on first call.
if (abs(x) < 0.2_sp) then                                Use series expansion.
  x2=x**2
  dawson_s=x*(1.0_sp-A1*x2*(1.0_sp-A2*x2*(1.0_sp-A3*x2)))
else                                                       Use sampling theorem representation.
  xx=abs(x)
  n0=2*nint(0.5_sp*xx/H)
  xp=xx-real(n0,sp)*H
  ec=exp(2.0_sp*xp*H)
  d1=arth(n0+1,2,NMAX)
  d2=arth(n0-1,-2,NMAX)
  e1=geop(ec,ec**2,NMAX)
  dawson_s=0.5641895835477563_sp*sign(exp(-xp**2),x)*&  Constant is  $1/\sqrt{\pi}$ .
    sum(c*(e1/d1+1.0_sp/(d2*e1)))
end if
END FUNCTION dawson_s

```

f90 REAL(SP), DIMENSION(NMAX), SAVE :: c=(/ (0.0_sp, i=1, NMAX) /) This is one way to give initial values to an array. Actually, we're somewhat nervous about using the “implied do-loop” form of the array constructor, as above, because our parallel compilers might not always be smart enough to execute the constructor in parallel. In this case, with NMAX=6, the damage potential is quite minimal. An alternative way to initialize the array would be with a data statement, “DATA c /NMAX*0.0_sp/”; however, this is not considered good Fortran 90 style, and there is no reason to think that it would be faster.

c(1:NMAX)=exp(-(arth(1,2,NMAX)*H)**2) Another example where the arth function of nrutil comes in handy. Otherwise, this would be

```

do i=1,NMAX
  c(i)=exp(-((2.0_sp*i-1.0_sp)*H)**2)
end do

```

arth(n0+1,2,NMAX) . . . arth(n0-1,-2,NMAX) . . . geop(ec,ec**2,NMAX) These are not just notationally convenient for generating the sequences $(n_0+1, n_0+3, n_0+5, \dots)$, $(n_0-1, n_0-3, n_0-5, \dots)$, and (ec, ec^3, ec^5, \dots) . They also may allow parallelization with parallel versions of arth and geop, such as those in nrutil.

```


FUNCTION dawson_v(x)
USE nrtype; USE nrutil, ONLY : arth
IMPLICIT NONE
REAL(SP), DIMENSION(:), INTENT(IN) :: x
REAL(SP), DIMENSION(size(x)) :: dawson_v
INTEGER(I4B), PARAMETER :: NMAX=6

```

```

REAL(SP), PARAMETER :: H=0.4_sp,A1=2.0_sp/3.0_sp,A2=0.4_sp,&
    A3=2.0_sp/7.0_sp
INTEGER(I4B) :: i,n
REAL(SP), DIMENSION(size(x)) :: x2
REAL(SP), DIMENSION(NMAX), SAVE :: c=(/ (0.0_sp,i=1,NMAX) /)
LOGICAL(LGT), DIMENSION(size(x)) :: mask
if (c(1) == 0.0) c(1:NMAX)=exp(-(arth(1,2,NMAX)*H)**2)
mask = (abs(x) >= 0.2_sp)
dawson_v=dawsonseries_v(x,mask)
where (.not. mask)
    x2=x**2
    dawson_v=x*(1.0_sp-A1*x2*(1.0_sp-A2*x2*(1.0_sp-A3*x2)))
end where
CONTAINS
FUNCTION dawsonseries_v(xin,mask)
IMPLICIT NONE
REAL(SP), DIMENSION(:), INTENT(IN) :: xin
LOGICAL(LGT), DIMENSION(size(xin)), INTENT(IN) :: mask
REAL(SP), DIMENSION(size(xin)) :: dawsonseries_v
INTEGER(I4B), DIMENSION(:), ALLOCATABLE :: n0
REAL(SP), DIMENSION(:), ALLOCATABLE :: d1,d2,e1,e2,sm,xp,xx,x
n=count(mask)
if (n == 0) RETURN
allocate(n0(n),d1(n),d2(n),e1(n),e2(n),sm(n),xp(n),xx(n),x(n))
x=pack(xin,mask)
xx=abs(x)
n0=2*nint(0.5_sp*xx/H)
xp=xx-real(n0,sp)*H
e1=exp(2.0_sp*xp*H)
e2=e1**2
d1=n0+1.0_sp
d2=d1-2.0_sp
sm=0.0
do i=1,NMAX
    sm=sm+c(i)*(e1/d1+1.0_sp/(d2*e1))
    d1=d1+2.0_sp
    d2=d2-2.0_sp
    e1=e2*e1
end do
sm=0.5641895835477563_sp*sign(exp(-xp**2),x)*sm
dawsonseries_v=unpack(sm,mask,0.0_sp)
deallocate(n0,d1,d2,e1,e2,sm,xp,xx)
END FUNCTION dawsonseries_v
END FUNCTION dawson_v

```

 dawson_v=dawsonseries_v(x,mask) Pass-the-buck method for getting masked values, see note to bessj0_v above, p. 1102. Within the contained dawsonseries, we use the pack-unpack method. Note that, unlike in dawson_s, the sums are done by do-loops, because the parallelization is already over the components of the vector argument.

* * *

```

FUNCTION rf_s(x,y,z)
USE nrtype; USE nrutil, ONLY : assert
IMPLICIT NONE
REAL(SP), INTENT(IN) :: x,y,z
REAL(SP) :: rf_s
REAL(SP), PARAMETER :: ERRTOL=0.08_sp,TINY=1.5e-38_sp,BIG=3.0e37_sp,&
    THIRD=1.0_sp/3.0_sp,&

```

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one), to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

```

C1=1.0_sp/24.0_sp,C2=0.1_sp,C3=3.0_sp/44.0_sp,C4=1.0_sp/14.0_sp
Computes Carlson's elliptic integral of the first kind,  $R_F(x,y,z)$ .  $x$ ,  $y$ , and  $z$  must be
nonnegative, and at most one can be zero. TINY must be at least 5 times the machine
underflow limit, BIG at most one-fifth the machine overflow limit.
REAL(SP) :: alamb,ave,delx,dely,delz,e2,e3,sqrtx,sqrty,sqrtz,xt,yt,zt
call assert(min(x,y,z) >= 0.0, min(x+y,x+z,y+z) >= TINY, &
  max(x,y,z) <= BIG, 'rf_s args')
xt=x
yt=y
zt=z
do
  sqrtx=sqrt(xt)
  sqrty=sqrt(yt)
  sqrtz=sqrt(zt)
  alamb=sqrtx*(sqrty+sqrtz)+sqrty*sqrtz
  xt=0.25_sp*(xt+alamb)
  yt=0.25_sp*(yt+alamb)
  zt=0.25_sp*(zt+alamb)
  ave=THIRD*(xt+yt+zt)
  delx=(ave-xt)/ave
  dely=(ave-yt)/ave
  delz=(ave-zt)/ave
  if (max(abs(delx),abs(dely),abs(delz)) <= ERRTOL) exit
end do
e2=delx*dely-delz**2
e3=delx*dely*delz
rf_s=(1.0_sp+(C1*e2-C2-C3*e3)*e2+C4*e3)/sqrt(ave)
END FUNCTION rf_s

```

```

FUNCTION rf_v(x,y,z)
USE nrtype; USE nrutil, ONLY : assert,assert_eq
IMPLICIT NONE
REAL(SP), DIMENSION(:), INTENT(IN) :: x,y,z
REAL(SP), DIMENSION(size(x)) :: rf_v
REAL(SP), PARAMETER :: ERRTOL=0.08_sp,TINY=1.5e-38_sp,BIG=3.0e37_sp,&
  THIRD=1.0_sp/3.0_sp,&
  C1=1.0_sp/24.0_sp,C2=0.1_sp,C3=3.0_sp/44.0_sp,C4=1.0_sp/14.0_sp
REAL(SP), DIMENSION(size(x)) :: alamb,ave,delx,dely,delz,e2,e3,&
  sqrtx,sqrty,sqrtz,xt,yt,zt
LOGICAL(LGT), DIMENSION(size(x)) :: converged
INTEGER(I4B) :: ndum
ndum=assert_eq(size(x),size(y),size(z),'rf_v')
call assert(all(min(x,y,z) >= 0.0), all(min(x+y,x+z,y+z) >= TINY), &
  all(max(x,y,z) <= BIG), 'rf_v args')
xt=x
yt=y
zt=z
converged=.false.
do
  where (.not. converged)
    sqrtx=sqrt(xt)
    sqrty=sqrt(yt)
    sqrtz=sqrt(zt)
    alamb=sqrtx*(sqrty+sqrtz)+sqrty*sqrtz
    xt=0.25_sp*(xt+alamb)
    yt=0.25_sp*(yt+alamb)
    zt=0.25_sp*(zt+alamb)
    ave=THIRD*(xt+yt+zt)
    delx=(ave-xt)/ave
    dely=(ave-yt)/ave
    delz=(ave-zt)/ave
    converged = (max(abs(delx),abs(dely),abs(delz)) <= ERRTOL)
  end where
end do

```

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for Internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-
 readable files (including this one) to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs
 visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

```

        end where
        if (all(converged)) exit
    end do
    e2=delx*dely-delz**2
    e3=delx*dely*delz
    rf_v=(1.0_sp+(C1*e2-C2-C3*e3)*e2+C4*e3)/sqrt(ave)
END FUNCTION rf_v

```

```

FUNCTION rd_s(x,y,z)
USE nrtype; USE nrutil, ONLY : assert
IMPLICIT NONE
REAL(SP), INTENT(IN) :: x,y,z
REAL(SP) :: rd_s
REAL(SP), PARAMETER :: ERRTOL=0.05_sp, TINY=1.0e-25_sp, BIG=4.5e21_sp, &
    C1=3.0_sp/14.0_sp, C2=1.0_sp/6.0_sp, C3=9.0_sp/22.0_sp, &
    C4=3.0_sp/26.0_sp, C5=0.25_sp*C3, C6=1.5_sp*C4
    Computes Carlson's elliptic integral of the second kind,  $R_D(x,y,z)$ .  $x$  and  $y$  must be
    nonnegative, and at most one can be zero.  $z$  must be positive. TINY must be at least twice
    the negative 2/3 power of the machine overflow limit. BIG must be at most  $0.1 \times \text{ERRTOL}$ 
    times the negative 2/3 power of the machine underflow limit.
REAL(SP) :: alamb, ave, delx, dely, delz, ea, eb, ec, ed, &
    ee, fac, sqrtx, sqrtz, sum, xt, yt, zt
call assert(min(x,y) >= 0.0, min(x+y,z) >= TINY, max(x,y,z) <= BIG, &
    'rd_s args')
xt=x
yt=y
zt=z
sum=0.0
fac=1.0
do
    sqrtx=sqrt(xt)
    sqrtz=sqrt(zt)
    alamb=sqrtx*(sqrtz+sqrtz)+sqrtz*sqrtx
    sum=sum+fac/(sqrtz*(zt+alamb))
    fac=0.25_sp*fac
    xt=0.25_sp*(xt+alamb)
    yt=0.25_sp*(yt+alamb)
    zt=0.25_sp*(zt+alamb)
    ave=0.2_sp*(xt+yt+3.0_sp*zt)
    delx=(ave-xt)/ave
    dely=(ave-yt)/ave
    delz=(ave-zt)/ave
    if (max(abs(delx),abs(dely),abs(delz)) <= ERRTOL) exit
end do
ea=delx*dely
eb=delz*delz
ec=ea-eb
ed=ea-6.0_sp*eb
ee=ed+ec+ec
rd_s=3.0_sp*sum+fac*(1.0_sp+ed*(-C1+C5*ed-C6*delz*ee)&
    +delz*(C2*ee+delz*(-C3*ec+delz*C4*ea)))/(ave*sqrt(ave))
END FUNCTION rd_s

```

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-
 readable files (including this one), to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs
 visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).


```

FUNCTION rd_v(x,y,z)
USE nrtype; USE nrutil, ONLY : assert,assert_eq
IMPLICIT NONE
REAL(SP), DIMENSION(:), INTENT(IN) :: x,y,z
REAL(SP), DIMENSION(size(x)) :: rd_v
REAL(SP), PARAMETER :: ERRTOL=0.05_sp,TINY=1.0e-25_sp,BIG=4.5e21_sp,&
  C1=3.0_sp/14.0_sp,C2=1.0_sp/6.0_sp,C3=9.0_sp/22.0_sp,&
  C4=3.0_sp/26.0_sp,C5=0.25_sp*C3,C6=1.5_sp*C4
REAL(SP), DIMENSION(size(x)) :: alamb,ave,delx,dely,delz,ea,eb,ec,ed,&
  ee,fac,sqrtx,sqrty,sqrtz,sum,xt,yt,zt
LOGICAL(LGT), DIMENSION(size(x)) :: converged
INTEGER(I4B) :: ndum
ndum=assert_eq(size(x),size(y),size(z),'rd_v')
call assert(all(min(x,y) >= 0.0), all(min(x+y,z) >= TINY), &
  all(max(x,y,z) <= BIG), 'rd_v args')
xt=x
yt=y
zt=z
sum=0.0
fac=1.0
converged=.false.
do
  where (.not. converged)
    sqrtx=sqrt(xt)
    sqrty=sqrt(yt)
    sqrtz=sqrt(zt)
    alamb=sqrtx*(sqrty+sqrtz)+sqrty*sqrtz
    sum=sum+fac/(sqrtz*(zt+alamb))
    fac=0.25_sp*fac
    xt=0.25_sp*(xt+alamb)
    yt=0.25_sp*(yt+alamb)
    zt=0.25_sp*(zt+alamb)
    ave=0.2_sp*(xt+yt+3.0_sp*zt)
    delx=(ave-xt)/ave
    dely=(ave-yt)/ave
    delz=(ave-zt)/ave
    converged = (all(max(abs(delx),abs(dely),abs(delz)) <= ERRTOL))
  end where
  if (all(converged)) exit
end do
ea=delx*dely
eb=delz*delz
ec=ea-eb
ed=ea-6.0_sp*eb
ee=ed+ec+ec
rd_v=3.0_sp*sum+fac*(1.0_sp+ed*(-C1+C5*ed-C6*delz*ee)&
  +delz*(C2*ee+delz*(-C3*ec+delz*C4*ea)))/(ave*sqrt(ave))
END FUNCTION rd_v

```

```

FUNCTION rj_s(x,y,z,p)
USE nrtype; USE nrutil, ONLY : assert
USE nr, ONLY : rc,rf
IMPLICIT NONE
REAL(SP), INTENT(IN) :: x,y,z,p
REAL(SP) :: rj_s
REAL(SP), PARAMETER :: ERRTOL=0.05_sp,TINY=2.5e-13_sp,BIG=9.0e11_sp,&
  C1=3.0_sp/14.0_sp,C2=1.0_sp/3.0_sp,C3=3.0_sp/22.0_sp,&
  C4=3.0_sp/26.0_sp,C5=0.75_sp*C3,C6=1.5_sp*C4,C7=0.5_sp*C2,&
  C8=C3+C3
  Computes Carlson's elliptic integral of the third kind,  $R_J(x,y,z,p)$ .  $x$ ,  $y$ , and  $z$  must be
  nonnegative, and at most one can be zero.  $p$  must be nonzero. If  $p < 0$ , the Cauchy

```

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for Internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-
 readable files (including this one) to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs
 visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

principal value is returned. TINY must be at least twice the cube root of the machine underflow limit, BIG at most one-fifth the cube root of the machine overflow limit.

```

REAL(SP) :: a,alamb,alpha,ave,b,bet,delp,delx,&
  dely,delz,ea,eb,ec,ed,ee,fac,pt,rho,sqrtx,sqrty,sqrtz,&
  sm,tau,xt,yt,zt
call assert(min(x,y,z) >= 0.0, min(x+y,x+z,y+z,abs(p)) >= TINY, &
  max(x,y,z,abs(p)) <= BIG, 'rj_s args')
sm=0.0
fac=1.0
if (p > 0.0) then
  xt=x
  yt=y
  zt=z
  pt=p
else
  xt=min(x,y,z)
  zt=max(x,y,z)
  yt=x+y+z-xt-zt
  a=1.0_sp/(yt-pt)
  b=a*(zt-yt)*(yt-xt)
  pt=yt+b
  rho=xt*zt/yt
  tau=p*pt/yt
end if
do
  sqrtx=sqrt(xt)
  sqrty=sqrt(yt)
  sqrtz=sqrt(zt)
  alamb=sqrtx*(sqrty+sqrtz)+sqrty*sqrtz
  alpha=(pt*(sqrtx+sqrty+sqrtz)+sqrtx*sqrty*sqrtz)**2
  bet=pt*(pt+alamb)**2
  sm=sm+fac*rc(alpha,bet)
  fac=0.25_sp*fac
  xt=0.25_sp*(xt+alamb)
  yt=0.25_sp*(yt+alamb)
  zt=0.25_sp*(zt+alamb)
  pt=0.25_sp*(pt+alamb)
  ave=0.2_sp*(xt+yt+zt+pt+pt)
  delx=(ave-xt)/ave
  dely=(ave-yt)/ave
  delz=(ave-zt)/ave
  delp=(ave-pt)/ave
  if (max(abs(delx),abs(dely),abs(delz),abs(delp)) <= ERRTOL) exit
end do
ea=delx*(dely+delz)+dely*delz
eb=delx*dely*delz
ec=delp**2
ed=ea-3.0_sp*ec
ee=eb+2.0_sp*delp*(ea-ec)
rj_s=3.0_sp*sm+fac*(1.0_sp+ed*(-C1+C5*ed-C6*ee)+eb*(C7+delp*(-C8&
  +delp*C4))+delp*ea*(C2-delp*C3)-C2*delp*ec)/(ave*sqrt(ave))
if (p <= 0.0) rj_s=a*(b*rj_s+3.0_sp*(rc(rho,tau)-rf(xt,yt,zt)))
END FUNCTION rj_s

```

```

FUNCTION rj_v(x,y,z,p)
USE nrtype; USE nrutil, ONLY : assert,assert_eq
USE nr, ONLY : rc,rf
IMPLICIT NONE
REAL(SP), DIMENSION(:), INTENT(IN) :: x,y,z,p
REAL(SP), DIMENSION(size(x)) :: rj_v
REAL(SP), PARAMETER :: ERRTOL=0.05_sp,TINY=2.5e-13_sp,BIG=9.0e11_sp,&
  C1=3.0_sp/14.0_sp,C2=1.0_sp/3.0_sp,C3=3.0_sp/22.0_sp,&

```

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for Internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one) to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

```

C4=3.0_sp/26.0_sp,C5=0.75_sp*C3,C6=1.5_sp*C4,C7=0.5_sp*C2,&
C8=C3+C3
REAL(SP), DIMENSION(size(x)) :: a,alamb,alpha,ave,b,bet,delp,delx,&
dely,delz,ea,eb,ec,ed,ee,fac,pt,rho,sqrtx,sqrty,sqrtz,&
sm,tau,xt,yt,zt
LOGICAL(LGT), DIMENSION(size(x)) :: mask
INTEGER(I4B) :: ndum
ndum=assert_eq(size(x),size(y),size(z),size(p),'rj_v')
call assert(all(min(x,y,z) >= 0.0), all(min(x+y,x+z,y+z,abs(p)) >= TINY), &
all(max(x,y,z,abs(p)) <= BIG), 'rj_v args')
sm=0.0
fac=1.0
where (p > 0.0)
  xt=x
  yt=y
  zt=z
  pt=p
elsewhere
  xt=min(x,y,z)
  zt=max(x,y,z)
  yt=x+y+z-xt-zt
  a=1.0_sp/(yt-pt)
  b=a*(zt-yt)*(yt-xt)
  pt=yt+b
  rho=xt*zt/yt
  tau=p*pt/yt
end where
mask=.false.
do
  where (.not. mask)
    sqrtx=sqrt(xt)
    sqrty=sqrt(yt)
    sqrtz=sqrt(zt)
    alamb=sqrtx*(sqrty+sqrtz)+sqrty*sqrtz
    alpha=(pt*(sqrtx+sqrty+sqrtz)+sqrtx*sqrty*sqrtz)**2
    bet=pt*(pt+alamb)**2
    sm=sm+fac*rc(alpha,bet)
    fac=0.25_sp*fac
    xt=0.25_sp*(xt+alamb)
    yt=0.25_sp*(yt+alamb)
    zt=0.25_sp*(zt+alamb)
    pt=0.25_sp*(pt+alamb)
    ave=0.2_sp*(xt+yt+zt+pt+pt)
    delx=(ave-xt)/ave
    dely=(ave-yt)/ave
    delz=(ave-zt)/ave
    delp=(ave-pt)/ave
    mask = (max(abs(delx),abs(dely),abs(delz),abs(delp)) <= ERRTOL)
  end where
  if (all(mask)) exit
end do
ea=delx*(dely+delz)+dely*delz
eb=delx*dely*delz
ec=delp**2
ed=ea-3.0_sp*ec
ee=eb+2.0_sp*delp*(ea-ec)
rj_v=3.0_sp*sm+fac*(1.0_sp+ed*(-C1+C5*ed-C6*ee)+eb*(C7+delp*(-C8&
+delp*C4))+delp*ea*(C2-delp*C3)-C2*delp*ec)/(ave*sqrt(ave))
mask = (p <= 0.0)
where (mask) rj_v=a*(b*rj_v+&
unpack(3.0_sp*(rc(pack(rho,mask),pack(tau,mask)))-&
rf(pack(xt,mask),pack(yt,mask),pack(zt,mask))),mask,0.0_sp))
END FUNCTION rj_v

```

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for Internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one) to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).



unpack(3.0_sp*(rc(pack(rho,mask),pack(tau,mask))...),mask,0.0_sp)

If you're willing to put up with fairly unreadable code, you can use the pack-unpack trick (for getting a masked subset of components out of a vector function) right in-line, as here. Of course the "outer level" that is seen by the enclosing where construction has to contain only objects that have the same shape as the mask that goes with the where. Because it is so hard to read, we don't like to do this very often. An alternative would be to use CONTAINS to incorporate short, masked "wrapper functions" for the functions used in this way.

```

FUNCTION rc_s(x,y)
USE nrtype; USE nrutil, ONLY : assert
IMPLICIT NONE
REAL(SP), INTENT(IN) :: x,y
REAL(SP) :: rc_s
REAL(SP), PARAMETER :: ERRTOL=0.04_sp,TINY=1.69e-38_sp,&
  SQRRTNY=1.3e-19_sp,BIG=3.0e37_sp,TNBG=TINY*BIG,&
  COMP1=2.236_sp/SQRRTNY,COMP2=TNBG*TNBG/25.0_sp,&
  THIRD=1.0_sp/3.0_sp,&
  C1=0.3_sp,C2=1.0_sp/7.0_sp,C3=0.375_sp,C4=9.0_sp/22.0_sp
  Computes Carlson's degenerate elliptic integral,  $R_C(x,y)$ .  $x$  must be nonnegative and  $y$ 
  must be nonzero. If  $y < 0$ , the Cauchy principal value is returned. TINY must be at least
  5 times the machine underflow limit, BIG at most one-fifth the machine maximum overflow
  limit.
REAL(SP) :: alamb,ave,s,w,xt,yt
call assert( (/x >= 0.0,y /= 0.0,x+abs(y) >= TINY,x+abs(y) <= BIG, &
  y >= -COMP1 .or. x <= 0.0 .or. x >= COMP2/), 'rc_s')
if (y > 0.0) then
  xt=x
  yt=y
  w=1.0
else
  xt=x-y
  yt=-y
  w=sqrt(x)/sqrt(xt)
end if
do
  alamb=2.0_sp*sqrt(xt)*sqrt(yt)+yt
  xt=0.25_sp*(xt+alamb)
  yt=0.25_sp*(yt+alamb)
  ave=THIRD*(xt+yt+yt)
  s=(yt-ave)/ave
  if (abs(s) <= ERRTOL) exit
end do
rc_s=w*(1.0_sp+s*s*(C1+s*(C2+s*(C3+s*C4)))/sqrt(ave)
END FUNCTION rc_s

```

```

FUNCTION rc_v(x,y)
USE nrtype; USE nrutil, ONLY : assert,assert_eq
IMPLICIT NONE
REAL(SP), DIMENSION(:), INTENT(IN) :: x,y
REAL(SP), DIMENSION(size(x)) :: rc_v
REAL(SP), PARAMETER :: ERRTOL=0.04_sp,TINY=1.69e-38_sp,&
  SQRRTNY=1.3e-19_sp,BIG=3.0e37_sp,TNBG=TINY*BIG,&
  COMP1=2.236_sp/SQRRTNY,COMP2=TNBG*TNBG/25.0_sp,&
  THIRD=1.0_sp/3.0_sp,&
  C1=0.3_sp,C2=1.0_sp/7.0_sp,C3=0.375_sp,C4=9.0_sp/22.0_sp
REAL(SP), DIMENSION(size(x)) :: alamb,ave,s,w,xt,yt
LOGICAL(LGT), DIMENSION(size(x)) :: converged
INTEGER(I4B) :: ndum

```

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-
 readable files (including this one), to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs
 visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

```

ndum=assert_eq(size(x),size(y),'rc_v')
call assert( (/all(x >= 0.0),all(y /= 0.0),all(x+abs(y) >= TINY), &
  all(x+abs(y) <= BIG),all(y >= -COMP1 .or. x <= 0.0 &
  .or. x >= COMP2) /),'rc_v')
where (y > 0.0)
  xt=x
  yt=y
  w=1.0
elsewhere
  xt=x-y
  yt=-y
  w=sqrt(x)/sqrt(xt)
end where
converged=.false.
do
  where (.not. converged)
    alamb=2.0_sp*sqrt(xt)*sqrt(yt)+yt
    xt=0.25_sp*(xt+alamb)
    yt=0.25_sp*(yt+alamb)
    ave=THIRD*(xt+yt+yt)
    s=(yt-ave)/ave
    converged = (abs(s) <= ERRTOL)
  end where
  if (all(converged)) exit
end do
rc_v=w*(1.0_sp+s*s*(C1+s*(C2+s*(C3+s*C4)))/sqrt(ave)
END FUNCTION rc_v

```

* * *

```

FUNCTION ellf_s(phi,ak)
USE nrtype
USE nr, ONLY : rf
IMPLICIT NONE
REAL(SP), INTENT(IN) :: phi,ak
REAL(SP) :: ellf_s
  Legendre elliptic integral of the 1st kind  $F(\phi, k)$ , evaluated using Carlson's function  $R_F$ .
  The argument ranges are  $0 \leq \phi \leq \pi/2$ ,  $0 \leq k \sin \phi \leq 1$ .
REAL(SP) :: s
s=sin(phi)
ellf_s=s*rf(cos(phi)**2,(1.0_sp-s*ak)*(1.0_sp+s*ak),1.0_sp)
END FUNCTION ellf_s

```

```

FUNCTION ellf_v(phi,ak)
USE nrtype; USE nrutil, ONLY : assert_eq
USE nr, ONLY : rf
IMPLICIT NONE
REAL(SP), DIMENSION(:), INTENT(IN) :: phi,ak
REAL(SP), DIMENSION(size(phi)) :: ellf_v
REAL(SP), DIMENSION(size(phi)) :: s
INTEGER(I4B) :: ndum
ndum=assert_eq(size(phi),size(ak),'ellf_v')
s=sin(phi)
ellf_v=s*rf(cos(phi)**2,(1.0_sp-s*ak)*(1.0_sp+s*ak),&
  spread(1.0_sp,1,size(phi)))
END FUNCTION ellf_v

```

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one), to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

```

FUNCTION elle_s(phi,ak)
USE nrtype
USE nr, ONLY : rd,rf
IMPLICIT NONE
REAL(SP), INTENT(IN) :: phi,ak
REAL(SP) :: elle_s
    Legendre elliptic integral of the 2nd kind  $E(\phi, k)$ , evaluated using Carlson's functions  $R_D$ 
    and  $R_F$ . The argument ranges are  $0 \leq \phi \leq \pi/2$ ,  $0 \leq k \sin \phi \leq 1$ .
REAL(SP) :: cc,q,s
s=sin(phi)
cc=cos(phi)**2
q=(1.0_sp-s*ak)*(1.0_sp+s*ak)
elle_s=s*(rf(cc,q,1.0_sp)-((s*ak)**2)*rd(cc,q,1.0_sp)/3.0_sp)
END FUNCTION elle_s

```

```

FUNCTION elle_v(phi,ak)
USE nrtype; USE nrutil, ONLY : assert_eq
USE nr, ONLY : rd,rf
IMPLICIT NONE
REAL(SP), DIMENSION(:), INTENT(IN) :: phi,ak
REAL(SP), DIMENSION(size(phi)) :: elle_v
REAL(SP), DIMENSION(size(phi)) :: cc,q,s
INTEGER(I4B) :: ndum
ndum=assert_eq(size(phi),size(ak),'elle_v')
s=sin(phi)
cc=cos(phi)**2
q=(1.0_sp-s*ak)*(1.0_sp+s*ak)
elle_v=s*(rf(cc,q,spread(1.0_sp,1,size(phi)))-((s*ak)**2)*&
rd(cc,q,spread(1.0_sp,1,size(phi)))/3.0_sp)
END FUNCTION elle_v

```



rd(cc,q,spread(1.0_sp,1,size(phi))) See note to erf_v, p. 1094 above.

```

FUNCTION ellpi_s(phi,en,ak)
USE nrtype
USE nr, ONLY : rf,rj
IMPLICIT NONE
REAL(SP), INTENT(IN) :: phi,en,ak
REAL(SP) :: ellpi_s
    Legendre elliptic integral of the 3rd kind  $\Pi(\phi, n, k)$ , evaluated using Carlson's functions  $R_J$ 
    and  $R_F$ . (Note that the sign convention on  $n$  is opposite that of Abramowitz and Stegun.)
    The ranges of  $\phi$  and  $k$  are  $0 \leq \phi \leq \pi/2$ ,  $0 \leq k \sin \phi \leq 1$ .
REAL(SP) :: cc,enss,q,s
s=sin(phi)
enss=en*s*s
cc=cos(phi)**2
q=(1.0_sp-s*ak)*(1.0_sp+s*ak)
ellpi_s=s*(rf(cc,q,1.0_sp)-enss*rj(cc,q,1.0_sp,1.0_sp+enss)/3.0_sp)
END FUNCTION ellpi_s

```

```

FUNCTION ellpi_v(phi,en,ak)
USE nrtype
USE nr, ONLY : rf,rj
IMPLICIT NONE
REAL(SP), DIMENSION(:), INTENT(IN) :: phi,en,ak
REAL(SP), DIMENSION(size(phi)) :: ellpi_v
REAL(SP), DIMENSION(size(phi)) :: cc,enss,q,s
s=sin(phi)
enss=en*s*s
cc=cos(phi)**2
q=(1.0_sp-s*ak)*(1.0_sp+s*ak)
ellpi_v=s*(rf(cc,q,spread(1.0_sp,1,size(phi)))-enss*&
  rj(cc,q,spread(1.0_sp,1,size(phi)),1.0_sp+enss)/3.0_sp)
END FUNCTION ellpi_v

```

* * *

```

SUBROUTINE sncndn(uu,emmc,sn,cn,dn)
USE nrtype; USE nrutil, ONLY : nrerror
IMPLICIT NONE
REAL(SP), INTENT(IN) :: uu,emmc
REAL(SP), INTENT(OUT) :: sn,cn,dn
  Returns the Jacobian elliptic functions  $sn(u, k_c)$ ,  $cn(u, k_c)$ , and  $dn(u, k_c)$ . Here  $uu = u$ ,
  while  $emmc = k_c^2$ .
REAL(SP), PARAMETER :: CA=0.0003_sp      The accuracy is the square of CA.
INTEGER(I4B), PARAMETER :: MAXIT=13
INTEGER(I4B) :: i,ii,l
REAL(SP) :: a,b,c,d,emc,u
REAL(SP), DIMENSION(MAXIT) :: em,en
LOGICAL(LGT) :: bo
emc=emmc
u=uu
if (emc /= 0.0) then
  bo=(emc < 0.0)
  if (bo) then
    d=1.0_sp-emc
    emc=-emc/d
    d=sqrt(d)
    u=d*u
  end if
  a=1.0
  dn=1.0
  do i=1,MAXIT
    l=i
    em(i)=a
    emc=sqrt(emc)
    en(i)=emc
    c=0.5_sp*(a+emc)
    if (abs(a-emc) <= CA*a) exit
    emc=a*emc
    a=c
  end do
  if (i > MAXIT) call nrerror('sncndn: convergence failed')
  u=c*u
  sn=sin(u)
  cn=cos(u)
  if (sn /= 0.0) then
    a=cn/sn
    c=a*c
    do ii=1,1,-1
      b=em(ii)

```

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-
 readable files (including this one), to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs
 visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

```

        a=c*a
        c=dn*c
        dn=(en(ii)+a)/(b+a)
        a=c/b
    end do
    a=1.0_sp/sqrt(c**2+1.0_sp)
    sn=sign(a,sn)
    cn=c*sn
end if
if (bo) then
    a=dn
    dn=cn
    cn=a
    sn=sn/d
end if
else
    cn=1.0_sp/cosh(u)
    dn=cn
    sn=tanh(u)
end if
END SUBROUTINE sncndn

```

* * *

MODULE hypgeo_info

```

USE nrtype
COMPLEX(SPC) :: hypgeo_aa,hypgeo_bb,hypgeo_cc,hypgeo_dz,hypgeo_z0
END MODULE hypgeo_info

```

FUNCTION hypgeo(a,b,c,z)

```

USE nrtype
USE hypgeo_info
USE nr, ONLY : bsstep,hypdrv,hypser,odeint
IMPLICIT NONE
COMPLEX(SPC), INTENT(IN) :: a,b,c,z
COMPLEX(SPC) :: hypgeo
REAL(SP), PARAMETER :: EPS=1.0e-6_sp
    Complex hypergeometric function  ${}_2F_1$  for complex  $a, b, c$ , and  $z$ , by direct integration of
    the hypergeometric equation in the complex plane. The branch cut is taken to lie along the
    real axis,  $\text{Re } z > 1$ .
    Parameter: EPS is an accuracy parameter.
COMPLEX(SPC), DIMENSION(2) :: y
REAL(SP), DIMENSION(4) :: ry
if (real(z)**2+aimag(z)**2 <= 0.25) then      Use series...
    call hypser(a,b,c,z,hypgeo,y(2))
    RETURN
else if (real(z) < 0.0) then                  ...or pick a starting point for the path
    hypgeo_z0=cplx(-0.5_sp,0.0_sp,kind=spc)    integration.
else if (real(z) <= 1.0) then
    hypgeo_z0=cplx(0.5_sp,0.0_sp,kind=spc)
else
    hypgeo_z0=cplx(0.0_sp,sign(0.5_sp,aimag(z)),kind=spc)
end if
hypgeo_aa=a                                  Load the module variables, used to pass
hypgeo_bb=b                                  parameters "over the head" of odeint
hypgeo_cc=c                                  to hypdrv.
hypgeo_dz=z-hypgeo_z0
call hypser(hypgeo_aa,hypgeo_bb,hypgeo_cc,hypgeo_z0,y(1),y(2))
    Get starting function and derivative.
ry(1:4:2)=real(y)

```

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for Internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-
 readable files (including this one), to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs
 visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).


```

ry(2:4:2)=aimag(y)
call odeint(ry,0.0_sp,1.0_sp,EPS,0.1_sp,0.0001_sp,hypdrv,bsstep)
  The arguments to odeint are the vector of independent variables, the starting and ending
  values of the dependent variable, the accuracy parameter, an initial guess for stepsize, a
  minimum stepsize, and the names of the derivative routine and the (here Bulirsch-Stoer)
  stepping routine.
y=cmplx(ry(1:4:2),ry(2:4:2),kind=spc)
hypgeo=y(1)
END FUNCTION hypgeo

```

```

SUBROUTINE hypser(a,b,c,z,series,deriv)
USE nrtype; USE nrutil, ONLY : nrerror
IMPLICIT NONE
COMPLEX(SPC), INTENT(IN) :: a,b,c,z
COMPLEX(SPC), INTENT(OUT) :: series,deriv
  Returns the hypergeometric series  ${}_2F_1$  and its derivative, iterating to machine accuracy.
  For  $\text{cabs}(z) \leq 1/2$  convergence is quite rapid.
INTEGER(I4B) :: n
INTEGER(I4B), PARAMETER :: MAXIT=1000
COMPLEX(SPC) :: aa,bb,cc,fac,temp
deriv=cmplx(0.0_sp,0.0_sp,kind=spc)
fac=cmplx(1.0_sp,0.0_sp,kind=spc)
temp=fac
aa=a
bb=b
cc=c
do n=1,MAXIT
  fac=((aa*bb)/cc)*fac
  deriv=deriv+fac
  fac=fac*z/n
  series=temp+fac
  if (series == temp) RETURN
  temp=series
  aa=aa+1.0
  bb=bb+1.0
  cc=cc+1.0
end do
call nrerror('hypser: convergence failure')
END SUBROUTINE hypser

```

```

SUBROUTINE hypdrv(s,ry,rdyds)
USE nrtype
USE hypgeo_info
IMPLICIT NONE
REAL(SP), INTENT(IN) :: s
REAL(SP), DIMENSION(:), INTENT(IN) :: ry
REAL(SP), DIMENSION(:), INTENT(OUT) :: rdyds
  Derivative subroutine for the hypergeometric equation; see text equation (5.14.4).
COMPLEX(SPC), DIMENSION(2) :: y,dyds
COMPLEX(SPC) :: z
y=cmplx(ry(1:4:2),ry(2:4:2),kind=spc)
z=hypgeo_z0+s*hypgeo_dz
dyds(1)=y(2)*hypgeo_dz
dyds(2)=(hypgeo_aa*hypgeo_bb)*y(1)-(hypgeo_cc-&
  (hypgeo_aa+hypgeo_bb)+1.0_sp)*z)*y(2))*hypgeo_dz/(z*(1.0_sp-z))
rdyds(1:4:2)=real(dyds)
rdyds(2:4:2)=aimag(dyds)
END SUBROUTINE hypdrv

```

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
 Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
 Permission is granted for Internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-
 readable files (including this one), to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs
 visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

f90 Notice that the real array (of length 4) `ry` is immediately mapped into a complex array of length 2, and that the process is reversed at the end of the routine with `rdyds`. In Fortran 77 no such mapping is necessary: the calling program sends real arguments, and the Fortran 77 `hypdrv` simply interprets what is sent as complex. Fortran 90's stronger typing does not encourage (and, practically, does not allow) this convenience; but it is a small price to pay for the vastly increased error-checking capabilities of a strongly typed language.

Sample page from NUMERICAL RECIPES IN FORTRAN 90: The Art of PARALLEL Scientific Computing (ISBN 0-521-57439-0)
Copyright (C) 1986-1996 by Cambridge University Press. Programs Copyright (C) 1986-1996 by Numerical Recipes Software.
Permission is granted for Internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one) to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs visit website <http://www.nr.com> or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).