c subroutine binco(x,b)
c computes all the binary coefficients for upper index x
c uses recursion starting from lower index 0
    real b(0:10)
    real x
    integer n, i
c common/coefs/b

n = 4

    b(0) = 1.

do i = 1,n
    b(i) = (n-i+1.)/(real(i))*b(i-1)
    print *,i,b ,i,b(i)
enddo

end

program fdiff
xc computes different orders of forward finite differences of
c given function f
set arbitrary maximum order to 10
real f, fk(0:10), x(0:10), del(0:10), b(0:10)
    integer n,i

open(unit = 8,file = 'fdiff.dat')
c select the order
write(*,*)'enter number of points & diffs'
read(*,*)nm
    n = nm-1
c set the range of x
write(*,*)'enter x0,xn'
read(*,*)x(0),x(n)
    dx = (x(n) - x(0))/real(n)
c call function f, defined below in order to set the vector fk
c used in constructing the differences
    fk(0) = f(x(0))
    fk(n) = f(x(n))
c finish setting the vector fk
if(n .gt. 1)then
doi = 1,n-1
\[ x(i) = x(i-1) + dx \]
\[ f_k(i) = f(x(i)) \]

enddo
endif

c use binomial coefficients defined below to construct forward differences
do i = 1,n
call binco(real(i),b)
del(i) = 0.
do j = 0,i
del(i) = del(i) + (-1.)**j*b(j)*fk(i-j)
enddo
endo

write(8,*)'fk ',(fk(i),i = 0,n)
write(8,*)'dn ',(del(i),i = 1,n)
end

function f(x)
real f,x
c f = x*x
f = sin(x)
subroutine binco(x,b)
  c computes all the binary coefficients for upper index x
  c uses recursion starting from lower index 0
  c routine designed for real x. Modified here for integer
real b(0:10)
  real x
  integer n, i
  c replace x with integer
  n = x

  b(0) = 1.

  do i = 1,n
    b(ii) = (n-i+1.)/(real(ii))*b(i-1)
  enddo

return
end

program sbess
c estimates spherical bessel function using recursion
uses ascending recursion for small x and descending for large x

real*8 x, f0, fac, f(0:100)
integer lm,l,n, lmx

open(unit = 8,file = 'sbess.dat')
write(8,'(*1x,5f18.10)')
write(8,*)'x, f(n), n = 0, 1, 2,...'

set range of values of x to be used to compute j(x)
x_min = 1.e-3
x_max = 12.

write(*,*) 'enter lmx, max Bessel function order desired to use'
read(*,*) lmx

nstps = 200

dx = (x_max - x_min)/nstps

x = x_min
c  ****************************************************************
c  loop through x values
c  ****************************************************************

do 200 i = 1,nstps

c  ****************************************************************
c  select method depending on x; divide range at 0.1
c  for x < divide use descending recursion
c  for x >= divide use ascending recursion

c  ****************************************************************

if(abs(x) .lt. 0.1)go to 99

c  ****************************************************************
c  descending recursion for larger x

c  ****************************************************************

c  ****************************************************************
c  for descending recursion start with a large lm

c  ****************************************************************

lm = 100

c  ****************************************************************
c  pick some arbitrary values to initiate descending recursion
c  Need values for two adjacent large orders

c  ****************************************************************
f(lm) = 1.d0
f(lm-1) = 5.d0

c ****************************************************************
c     execute descending recursion of Bessel functions
c     down to zero order
c ****************************************************************

do n = lm,2,-1
    f(n-2) = ((2*n-1.d0)/x)*f(n-1) - f(n)
endo  
c ****************************************************************
c     renormalize using known value for f0

c Start by finding ratio of known zero order value to computed

c value by recursion

c ****************************************************************

f0 = sin(x)/x

fac = f0/f(0)

c ****************************************************************
c     Then renormalize each of the previously computed orders
c ****************************************************************

do n = 0,lmx
\[ f(n) = \text{fac} \times f(n) \]
enddo

c  ****************************************************************
c  Done, so go to end
c  ****************************************************************
goto 199

99      continue

c  ****************************************************************
c  Or do ascending recursion for small x
c  ****************************************************************
c  ****************************************************************
c  ascending recursion relation
c  ****************************************************************
c  ****************************************************************
c  to avoid numerical problems use expansions for f0 and f1 at small x
c  ****************************************************************

if(abs(x) .lt. 1.e-6) then
    f(0) = 1. - x^2/6.
    f(1) = x/3.
else
    f(0) = \sin(x)/x
    f(1) = (\sin(x) - x\cos(x))/x^2

endif

c ****************************************************************
c Provided we want orders beyond 1 need to use recursion again
c ****************************************************************

if(lmx .gt. 1)then

c ****************************************************************
c execute ascending recursion for orders higher than one in

c small argument range

c ****************************************************************

do n = 2,lmx
    f(n) = ((2.*n - 1)/x)*f(n-1) - f(n-2)
enddo

endif

199 continue

c ****************************************************************
c set for maximum 5 spherical Bessel fcns

c ****************************************************************

c ****************************************************************
c write results for later plotting
write(8,'(1p7e12.3)')x,(f(n),n = 0,lmx)

c *****************************************

c increment x for next pass

c *****************************************

x = x + dx

200 continue

end

c tests execution speed of array arithmetic

c in Fortran cycling rows and columns first

real a(10000,10000), t1, t2

c initialize timing

call cpu_time(t1)

c define elements by row; cycle 2nd index fastest
do i = 1, 10000
  do j = 1, 10000
    a(i,j) = real(i+j)
  enddo
enddo

call cpu_time(t2)

print *, ' defining array by row , delta t = ', t2 - t1
print *
print *
array element a(10000,10000)

call cpu_time(t1)

c define elements by column; cycle 1st index fastest

do j = 1, 10000
  do i = 1, 10000
    a(i,j) = real(i+j)
  enddo
enddo
enddo

call cpu_time(t2)

print *
print *
print *, ' defining array by column , delta t = ', t2 - t1
print *
print *
array element a(10000,10000)
program deriv

  c      computes central lowest order numerical derivatives
  c      and forward differences to second order

  c define arrays for data sampling
  c set data type for user defined function f and its derivative

  real*8 y(0:5000),dy(0:5000),f,df,x(0:5000),dx
  real*8 dff,dfc,ef,ec,efmx,ecmx

  c open output file
  open(unit = 8,file = 'deriv.dat')

  print *,'enter initial number of sample points < 5000'
  read *,n

  c set up loop to refine the differences to look at convergence
  c find derivative estimates for 6 different resolutions

  do k = 1,6
c set the range and difference interval for calculations

\[ x(0) = -2.0 \]
\[ x(n) = 2.0 \]

c set difference interval and grid points

\[ dx = \frac{x(n) - x(0)}{\text{real}(n)} \]

do i = 1,n-1
\[ x(i) = x(i-1) + dx \]
enddo

c evaluate f and its analytic derivative using function calls

c on grid points

do i = 0,n
\[ y(i) = f(x(i)) \]
\[ dy(i) = df(x(i)) \]
enddo

c find lowest order forward and central difference derivatives

c initialize routine to find max values

c initialize search for max values by negative (impossible) values

\[ efmx = -1.0 \times 10^{37} \]
\[ ecmx = -1.0 \times 10^{37} \]
c compare numerical to analytic derivatives and find the max
c dff is forward difference, dfc is central difference
c ef is forward error, ec is central error, both compared to
c analytical value
c efmax and ecmax are the respective max error values

do i = 1,n-1
    if(i .lt. n-1)then
        dff = 0.5*(4.*y(i+1) -y(i+2) -3*y(i))/dx
        ef = abs((dff - dy(i))/dy(i))
        efmx = max(ef,efmx)
    endif
    dfc = 0.5*(y(i+1) - y(i-1))/dx
    ec = abs((dfc - dy(i))/dy(i))
    ecmx = max(ec,ecmx)
enddo

write(8,'(i5,1p2e12.3)')n,efmx,ecmx

c now double the resolution and do it again

    n = 2*n

enddo

end

function f(x)
c user function to evaluate

real*8 f,x
f = tanh(x)
return
end

function df(x)
c user derivative function to evaluate
real*8 df,x
df = (cosh(x))**(2)
return
end

!##################################################################################
!##################################################################################
program hunt
c for a sequence of points find the indices that span a value
c use bisection method

real x(0:5000), xt
integer n,i,i0,inc,imn,imx

open(unit = 8,file = 'hunt.dat')

set number of bins
n = 200
write(8,'*')'number of bins = ',n

c  fix end points of full interval

x(0) = 0.
x(n) = 2.

c  set up intervals with a geometrically increasing size
c  ratio of bin sizes is r

r = 1.05

write(8,'*')'geometric bin size ratio = ',r

write(8,*)''

dx = (x(n) - x(0))*(r - 1.)/(r**n - 1.)

do i = 1,n-1
  x(i) = x(i-1) + dx
  dx = r*dx
  write(99,*),i,x(i)
enddo
xtest = x(n-1) + dx - x(n)

c  select trial xs by random number generator
c  iseed = 1764
  print *, 'enter random seed '
  read*, iseed

c  loop though the target values to bin

do kk = 1, 10

c    code picks 10 values randomly between x(0) and x(n)
x_t = x(0) + (x(n) - x(0)) * ran(iseed)

c    Set upper limit to the number of iterations

  jmx = 2 * n

c    first guess is the middle index

c    set upper bound at the top of grid

c    set lower bound at the bottom of grid

  i = n/2
    imx = n+1
    imn = -1

c    Create index j that tracks the number of tries.

  j = 1

c    statement 300 is the escape from the algorithm
statement 100 is the top of upper bisection loop
statement 200 is the top of lower bisection loop

100  continue

if(imx - imn .le. 1)goto 300

c  bound index from above by bisecting upper interval

    if(x(i) .lt. xt)then
        imn = i
        i = 0.5*(i + imx)
        j = j + 1
        if(j .gt. jmx)goto 300
        goto 100
    else
    imx = i
    i = 0.5*(i + imn)
    goto 200
    endif

200  continue

    if(imx - imn .le. 1)goto 300

c  bound from below by bisecting lower interval
if(x(i) .ge. xt)then
    imx = i
    i = 0.5*(i + imn)
    j = j+1
    if(j .gt. jmx)goto 300
    goto 200
else
    imn = i
    i = 0.5*(i + imx)
    goto 100
endif

300    continue

    idif = imx - imn

    write(8,*),'niter ', j
    write(8,'(a24,1pe12.3,2i5,1p2e12.3)')
    + 'x,imn,imx,x(imn),x(imx) ',xt,imn,imx,x(imn),x(imx)
    write(8,*)
    write(8,*)
    write(*,'(a39,1pe12.3,5x,2i5)')'xt, n bins & num of iterations,'
    + 'x,imn,imx,x(imn),x(imx) ',xt,imn,imx,x(imn),x(imx)
    print *,"
enddo

end