```
#The following function vectorises the function norminvcdf from the package StatsFuns
function normquant(x1::Number)
    return norminvcdf(x1)
end
function normquant(x1::Array{Float64, 1})
    return map(norminvcdf, x1)
end

#And this function vectorises norm pdf from the same package

function normpdfalt(x1::Number)
    return normpdf(x1)
end
function normpdfalt(x1::Array{Float64, 1})
    return map(normpdf, x1)
end

# This function is based on the method described by
#     Drezner, Z and G.O. Wesolowsky, (1989),
#     On the computation of the bivariate normal integral,
#     Journal of Statist. Comput. Simul. 35, pp. 101-107,
# with major modifications for double precision, and for |R| close to 1.
#It calculates the Prob(X>dh, y>DY) for standard bivariate normal with correlation coefficient return
#Source of code: https://github.com/JuliaStats/StatsFuns.jl/blob/e21bc26b1773aeb86bc8df4832db9c371d09ba2b

const bvncdf_w_array = [0.1713244923791705e+00 0.4717533638651177e-01 0.1761400713915212e-01;
                        0.3607615730481384e+00 0.1069393259953183e+00 0.4060142980038694e-01;
                        0.4679139345726904e+00 0.1600783285433464e+00 0.6267204833410906e-01;
                        0.0                    0.2031674267230659e+00 0.8327674157670475e-01;
                        0.0                    0.2334925365383547e+00 0.1019301198172404e+00;
                        0.0                    0.2491470458134029e+00 0.1181945319615184e+00;
                        0.0                    0.0                    0.1316886384491766e+00;
                        0.0                    0.0                    0.1420961093183821e+00;
                        0.0                    0.0                    0.1491729864726037e+00;
                        0.0                    0.0                    0.1527533871307259e+00]

const bvncdf_x_array = [-0.9324695142031522e+00 -0.9815606342467191e+00 -0.9931285991850949e+00;
                        -0.6612093864662647e+00 -0.9041172563704750e+00 -0.9639719272779138e+00;
                        -0.2386191860831970e+00 -0.7699026741943050e+00 -0.9122344282513259e+00;
                         0.0                    -0.5873179542866171e+00 -0.8391169718222188e+00;
                         0.0                    -0.3678314989981802e+00 -0.7463319064601508e+00;
                         0.0                    -0.1252334085114692e+00 -0.6360536807265150e+00;
                         0.0                    0.0                     -0.5108670019508271e+00;
                         0.0                    0.0                     -0.3737060887154196e+00;
                         0.0                    0.0                     -0.2277858511416451e+00;
                         0.0                    0.0                     -0.7652652113349733e-01]

function bvnuppercdf(dh::Float64, dk::Float64, r::Float64)
    if abs(r) < 0.3
       ng = 1
       lg = 3
    elseif abs(r) < 0.75
       ng = 2
       lg = 6
    else
       ng = 3
       lg = 10
    end
    h = dh
    k = dk
    hk = h*k
    bvn = 0.0
    if abs(r) < 0.925
        if abs(r) > 0
            hs = (h * h + k * k) * 0.5
            asr = asin(r)
            for i = 1:lg
                for j = -1:2:1
                    sn = sin(asr * (j * bvncdf_x_array[i, ng] + 1.0) * 0.5)
                    bvn += bvncdf_w_array[i, ng] * exp((sn * hk - hs) / (1.0 - sn*sn))
                end
            end
```

```
            bvn *= asr / (4.0pi)
        end
        bvn += cdf(Normal(), -h) * cdf(Normal(), -k)
    else
        if r < 0
            k = -k
            hk = -hk
        end
        if abs(r) < 1
            as = (1.0 - r) * (1.0 + r)
            a = sqrt(as)
            bs = (h - k)^2
            c = (4.0 - hk) * 0.125
            d = (12.0 - hk) * 0.0625
            asr = -(bs / as + hk) * 0.5
            if ( asr > -100 )
                bvn = a * exp(asr) * (1.0 - c * (bs - as) * (1.0 - d * bs / 5.0) / 3.0 + c * d * as * as
            end
            if -hk < 100
                b = sqrt(bs)
                bvn -= exp(-hk * 0.5) * sqrt(2.0pi) * cdf(Normal(), -b / a) * b * (1.0 - c * bs * (1.0 -
            end
            a /= 2.0
            for i = 1:lg
                for j = -1:2:1
                    xs = (a * (j*bvncdf_x_array[i, ng] + 1.0))^2
                    rs = sqrt(1.0 - xs)
                     asr = -(bs / xs + hk) * 0.5
                     if asr > -100
                         bvn += a * bvncdf_w_array[i, ng] * exp(asr) * (exp(-hk * (1.0 - rs) / (2.0 * (1.
                     end
                 end
             end
             bvn /= -2.0pi
         end
         if r > 0
             bvn += cdf(Normal(), -max(h, k))
         else
             bvn = -bvn
             if k > h
                 bvn += cdf(Normal(), k) - cdf(Normal(), h)
             end
         end
     end
     return bvn
end

```