Providing test values and correctly rounded results for Excel VBA

```
> restart; Digits:=18: #interface(version);
```

The usual way to write the cumulative normal through the error function erf is $N(x) = \left(1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right) / 2$.

Numerically (and of course depending on the libraries involved) it is more efficient to use erfc, since it may avoid cancellation errors through the symmetry $N(-x) = 1 - N(x)$.

```
> cdfN:= x -> piecewise(x < 0, erfc(-x/sqrt(2))/2, 0 <= x,
  (1+erf(x/sqrt(2)))/2):
  'cdfN(x)': '%'=%;
```

$$\mathrm{cdfN}(x) = \begin{cases} \frac{1}{2}\,\mathrm{erfc}\left(-\frac{x\sqrt{2}}{2}\right) & x < 0 \\ \frac{1}{2} + \frac{1}{2}\,\mathrm{erf}\left(\frac{x\sqrt{2}}{2}\right) & 0 \le x \end{cases}$$

To judge numerical results done in double precision one needs to find the unique nearest number in IEEE 754, for which a library is loaded and used.

```
> read cat(myLib, "nearest.mpl"):
```

For example π is represented by .785398163397448279 * 4 = 3.14159265358979312:

```
> 'nearest(Pi)': '%'=%; ``=evalf(rhs(%));
```

$$\mathrm{nearest}(\pi) = \frac{884279719003555}{1125899906842624}\,\mathrm{pow}(2, 2)$$
$$= 3.14159265358979312$$

For testing floating point results in double precision now proceed as follows:

for an input x determine the nearest IEEE number X (which is a rational number), feed the function cdfN with X, numerical evaluate with increased precison (of 80 decimal digits) storing it in y - and finally find the very IEEE number Y, which is nearest to y.

```
> IEEE_result:=proc(x)
    local X,y,Y;
    X:=eval( nearest(x) );
    y:=evalf[80]( cdfN(X) );
    Y:=nearest(y);
  end proc;
```

IEEE_result :=

proc(x) **local** X, y, Y; X := eval(nearest(x)); y := evalf[80](cdfN(X)); Y := nearest(y) **end proc**

For example cdfN(−π) has the following representation:

```
> 'IEEE_result(-Pi)': '%'=%;  ``=evalf(rhs(%));

  rhs(%):
  'nearest(%)': '%'=%;
```

$$\mathrm{IEEE\_result}(-\pi) = \frac{3874545677847771}{4503599627370496}\,\mathrm{pow}(2, -10)$$
$$= 0.00084015816826337494 2$$

$$\text{nearest}(0.000840158168263374942) = \frac{3874545677847771}{4503599627370496}\,\text{pow}(2, -10)$$

The command nearest will recover the IEEE value from its approximate decimal presentation *exactly*.

To use it in Excel one needs a twist, since the I/O routines in VBA allow only 15 decimal places.

For that decompose a float x into its "head" having 15 decimals and its "tail" = x - head (written with 15 decimals as well), so x = head + tail ONE and setting ONE = 1.0 as global variable in VBA that overcomes the ugly "15 significant digits limitation".

```
> toVBA:=proc(x)
    local head, tail;
    Digits:=36;
    head:=evalf(x);
    head:=evalf[15](head);
    tail:= evalf(x) - head;
    tail:=evalf[15](tail);
    head + tail*ONE;
  end proc:
```

In the running example this is

```
> 'toVBA(eval( IEEE_result(-Pi) ))': '%'=%;
```

$$\text{toVBA}(\text{eval}(\text{IEEE\_result}(-\pi))) = 0.000840158168263375 - 0.580163766770170\ 10^{-19}\ \text{ONE}$$

This means:

The correct value of an implementation in VBA should be equal to .840158168263375e-3 – .580163766770170e-19 ONE and as we work with IEEE 754 this is exactly the value.

Let me provide some more explicit test values and their expected results. Below they are given as pairs in the presentation needed for Excel's VBA, the inputs x are the first entries and the correct results are the second ones:

```
> testValues:=[-36.181640625, -9.521484375, -6.73828125, -3.1494140625,
  -1.46484375, -0.36181640625];

  map(nearest, %):                      # write them as IEEE
  testValues:= map(toVBA, %):           # the way VBA can eat it
  ``;
  S1:=convert(testValues, Vector):      # inputs as vector
  eval(S1, ONE=1):                      # just in case ONE would appear ...
  S2:=map(IEEE_result, %):              # results as vector
  S3:=map(toVBA, S2):                   # results prepared for VBA
  Matrix([S1, S3]):

  convert(%, listlist): map(lprint,%); # ready for copy + paste
```

testValues := [-36.181640625, -9.521484375, -6.73828125, -3.1494140625, -1.46484375, -0.36181640625]

```
[-36.1816406250000, .591865808779869e-286+.293609510798825e-301*ONE]
 [-9.52148437500000, .853613924339736e-21+.112354473630801e-36*ONE]
 [-6.73828125000000, .801355578082585e-11+.246146033423138e-26*ONE]
  [-3.14941406250000, .8179910854034l1e-3-.853676172334019e-19*ONE]
  [-1.46484375000000, .714817762146660e-1+.456989977205922e-16*ONE]
   [-.361816406250000, .358744615604279-.494961151165626e-15*ONE]
```

[ ]