The Cumulative Normal Distribution using qfloat floating-point Library from LCC-WIN32

This implementation gives exactness over almost the 104 digits which the library provides.

For 'small' arguments an approach of Marsaglia is used (Taylor series in integer points and some pre-computed values using Maple). For 'large' arguments an asymptotic series from Abramowitz & Stegun is taken.

The library qfloat.dll should be in Window's system directory and the concurrent cdfn_mpl.dll should be places in the directory of this worksheet.

The C source for the DLL is given at the end.

AVt, Dec 2005

```
> restart;
  kernelopts(version);
```

Maple 10.02, IBM INTEL NT, Nov 8 2005 Build ID 208934

```
> Digits_lcc:=105;
```

$$\text{Digits\_lcc} := 105$$

```
> Digits:=2*Digits_lcc; # greater precision to check results
```

$$\text{Digits} := 210$$

Define the cumulative normal distribution within Maple

```
> cdfN := x -> 1/2+1/2*erf(1/2*x*2^(1/2));
  pdfN := x -> 1/2*1/Pi^(1/2)*exp(-1/2*x^2)*2^(1/2);
```

$$\text{cdfN} := x \to \frac{1}{2} + \frac{1}{2}\,\text{erf}\left(\frac{1}{2}\,x\,\sqrt{2}\right)$$

$$\text{pdfN} := x \to \frac{1}{2}\,\frac{\mathbf{e}^{(-1/2\,x^2)}\,\sqrt{2}}{\sqrt{\pi}}$$

For using the cdfn.dll locate its directory and call the external functions from there

```
> currentdir(): myDLL:=cat(%,`\\cdfn.dll`);
```

myDLL := "C:_Work\other\LCC_Work\cdfN\lcc\cdfn.dll"

A quick and dirty way is through strings:

```
> fct1 := define_external(
    'str_cdfN_str_Maple',
    'C',
    'x_str'::string[],
    'y_str'::string[],
    RETURN::integer[4],
    LIB=myDLL):

  cdfN_lcc:=proc(x)
  local X::string, Y;

  Y:=StringTools:-Fill( `0` , Digits_lcc+10);
  X:=convert(x,string);

  if type(parse(X),numeric) then
    fct1(X,Y);
    return parse(Y);
```

```
  else
    return 'x';
  end if;
  end proc:
```

This will provide the DLL with memory space (given as a string y_str) to store the results in the DLL. Since update is 'inplace' this will modify the string and its length. As Maple is a symbolic system one should never call this result directly, since this inconsistency for the same object will crash it (just try it and restart ...).

But a simple procedure above does the job.

Test that for inputs (first display yL from DLL, then Maple's result yM):

```
> xTst:= 1.0 - 1e-10;
  yL:= cdfN_lcc(xTst):
  evalf(cdfN(xTst)): yM:=evalf(%,105):
  yL; yM;
  `absolute error`=yL-yM;
  evalf((yL-yM)/yM,105): `relative error`=evalf(%,16);
```

$$xTst := 0.9999999999$$

$$0.84134474604434587613210835702965918643536976670176375716254332093295881667246875791300147\backslash$$
$$12743876885157318$$

$$0.84134474604434587613210835702965918643536976670176375716254332093295881667246875791300147\backslash$$
$$12743876885157317$$

$$\text{absolute error} = 0.1\ 10^{-104}$$

$$\text{relative error} = 0.1188573417379244\ 10^{-104}$$

... which is the exactness qfloat will provide directly ...

```
> xTst:= -4.2;
  yL:= cdfN_lcc(xTst):
  evalf(cdfN(xTst)): yM:=evalf(%,105):
  yL; yM;
  `absolute error`=yL-yM;
  evalf((yL-yM)/yM,105): `relative error`=evalf(%,16);
```

$$xTst := -4.2$$

$$0.00001334574901590633835309211778562737025071273916797644362072086788051355309341445686582\backslash$$
$$09720285026030765360$$

$$0.00001334574901590633835309211778562737025071273916797644362072086788051355309341445686582\backslash$$
$$09720285026030765359$$

$$\text{absolute error} = 0.1\ 10^{-108}$$

$$\text{relative error} = 0.7493022675670991\ 10^{-104}$$

```
> xTst:= -12.2;
  yL:=cdfN_lcc(xTst):
  evalf(cdfN(xTst)): yM:=evalf(%,105):
  yL; yM;
  `absolute error`=yL-yM;
  evalf((yL-yM)/yM,105): `relative error`=evalf(%,16);
```

$$xTst := -12.2$$

$$0.15541197863895935096114585573572951291478647716305510575010525401903160941684312751825282\backslash$$
$$19880316783407361\ 10^{-33}$$

0.155411978638959350961145855735729512914786477163055105750105254019031609416843127518252\
8 19880316783407360 10^{-33}

absolute error = 0.1 10^{-137}

relative error = 0.6434510446090644 10^{-104}

For the asymptotics:

```
> remDigits:=Digits: Digits:=1000:
  xTst:= -25.0 - 1e-4;
  yL:=cdfN_lcc(xTst):
  evalf(cdfN(xTst)): evalf(%): yM:=evalf(%,105):
  yL; yM;
  `absolute error`= yL-yM;
  evalf((yL-yM)/yM,105): `relative error`=evalf(%,16);
  Digits:=remDigits:
```

xTst := -25.0001

0.304905233610318176415648507799577085187316258592947937753137961951719488589802733602276\
4 50429538402490384 10^{-137}

0.304905233610318176415648507799577085187316258592947937753137961951719488589802733602276\
4 50429538402490391 10^{-137}

absolute error = -0.7 10^{-241}

relative error = -0.2295795292561720 10^{-103}

```
> remDigits:=Digits: Digits:=1000:
  xTst:= -32;
  yL:=cdfN_lcc(xTst):
  evalf(cdfN(xTst)): evalf(%): yM:=evalf(%,105):
  yL; yM;
  `absolute error`= yL-yM;
  evalf((yL-yM)/yM,105): `relative error`=evalf(%,16);
  #rhs(%): identify(%);
  Digits:=remDigits:
```

xTst := -32

0.545208060351239609196235250386970807887357546904723749404913718524812142903326769728959\
5 7221693762208378 9 10^{-224}

0.545208060351239609196235250386970807887357546904723749404913718524812142903326769728959\
5 7221693762208379 6 10^{-224}

absolute error = -0.7 10^{-328}

relative error = -0.1283913520187208 10^{-103}

```
> remDigits:=Digits: Digits:=2000:
  xTst:= -65;
  yL:=cdfN_lcc(xTst):
  evalf(cdfN(xTst)): evalf(%): yM:=evalf(%,105):
  yL; yM;
  `absolute error`= yL-yM;
  evalf((yL-yM)/yM,105): `relative error`=evalf(%,16);
  #rhs(%): identify(%);
  Digits:=remDigits:
```

xTst := -65

0.219180017725502893510973845400740893190140398055782708938133419826565640132483757789988\
5 1493109570454507 1 10^{-919}

0.219180017725502893510973845400740893190140398055782708938133419826565640132483757789988\
5 1493109570454507 3 10^{-919}

$$\text{absolute error} = -0.2\ 10^{-1023}$$

$$\text{relative error} = -0.9124919418999062\ 10^{-104}$$

>

>

The Algorithm

The method for 'small' x is given by George Marsaglia, Evaluating the Normal Distribution, Journal of Statistical Software (2004), http://www.jstatsoft.org/counter.php?id=100&url=v11/i04/

For 'large' arguments an asymptotic series from Abramowitz & Stegun is taken:

For x beyond 23.0 the value of cdfN(x) starts with 115 leading 9's after the decimal point

```
> remDigits:=Digits: Digits:=2000:
  xTst:= 23.0;
  evalf(cdfN(xTst)): evalf(%):
  evalf(%,117);
  Digits:=remDigits:
```

$$xTst := 23.0$$

$$0.99999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999\backslash$$
$$99999999999999999999999999999998$$

Since qfloat has 104 digits of exactness with QFLT_EPSILON around .109e-105 one can not destinguish cdfN(x) from 1.0 in qfloat for 23.0 < x. So set it to 1 for those cases.

Now look for very small arguments (using very high precision in Maple):

```
> remDigits:=Digits: Digits:=10000:
  xTst:= -213.1;
  yL:=cdfN_lcc(xTst):
  evalf(cdfN(xTst)): evalf(%): yM:=evalf(%,105):
  yL; yM;
  `absolute error`= yL-yM;
  evalf((yL-yM)/yM,105): `relative error`=evalf(%,16);
  #rhs(%): identify(%);
  Digits:=remDigits:
```

$$xTst := -213.1$$

$$0.18471745862110889328015324162355348247134275284562890655552593884134793292610332185 7\backslash$$
$$01063829207647706230 8\ 10^{-9863}$$

$$0.18471339126350836726235770068155654689622635366405542908938884474865911227926193196 5\backslash$$
$$80809876272182174070 9\ 10^{-9863}$$

$$\text{absolute error} = 0.40673576005260177955409419969355751163991815734774661370940926888206468\backslash$$
$$41389891202539529354655321599\ 10^{-9868}$$

$$\text{relative error} = 0.00002201983068311278$$

Since QFLT_MIN is about .706e-9864 one can set N(x) = for x < 213.1.

For the remaining range an asymtotic series from Abramowitz & Stegun is used (7.1.23, p 298), here 1 << x so x is *positive*:

>

```
> # A&S 7.1.23, p. 298: asympt for x -> oo
  sqrt(Pi)*exp(z^2)*z*erfc(z) = '(1+Sum( (-1)^m* doublefact(m)/(2*z^2)^m,
  m=1..infinity))',
  `   abs(argument(z))` < 3*Pi/4;
  ``;
  doublefact:= n -> (2*n)!/(n!*2^n);
```

$$\sqrt{\pi}\, e^{(z^2)}\, z\, \mathrm{erfc}(z) = 1 + \left(\sum_{m=1}^{\infty} \frac{(-1)^m\, \mathrm{doublefact}(m)}{(2\, z^2)^m} \right), \quad \mathrm{abs}(\mathrm{argument}(z)) < \frac{3\,\pi}{4}$$

$$\mathrm{doublefact} := n \to \frac{(2\,n)!}{n!\, 2^n}$$

Using $\mathrm{erfc}(z) = 1 + \mathrm{erf}(-z)$ and $z = \frac{1\,x}{\sqrt{2}}$ this reads as

```
> '(1+erf(-x/sqrt(2)))/2' = '1/2*exp(-x^2/2)/(x/sqrt(2))/sqrt(Pi)*
    (1+Sum( (-1)^m* doublefact(m)/(x)^m, m=1..infinity))';
```

$$\frac{1}{2} + \frac{1}{2}\,\mathrm{erf}\left(-\frac{x}{\sqrt{2}}\right) = \frac{1}{2}\,\frac{e^{\left(-\frac{x^2}{2}\right)}\sqrt{2}\left(1 + \left(\sum_{m=1}^{\infty} \frac{(-1)^m\, \mathrm{doublefact}(m)}{x^m}\right)\right)}{x\,\sqrt{\pi}}$$

or in a more compact form

```
> 'N(-x)' = '1/x*pdfN(x) * (1+asymptSeries(x))';
  'asymptSeries(x) = Sum( (-1)^m* doublefact(m)/(x^2)^m, m=1..infinity)';
```

$$N(-x) = \frac{\mathrm{pdfN}(x)\,(1 + \mathrm{asymptSeries}(x))}{x}$$

$$\mathrm{asymptSeries}(x) = \sum_{m=1}^{\infty} \frac{(-1)^m\, \mathrm{doublefact}(m)}{(x^2)^m}$$

This series can be (approximately) calculated very fast through recursion using

```
> 'doublefact(1) = 1, doublefact(n+1)/doublefact(n) = (2*n+1)';
  #simplify(%): is(%);
```

$$\mathrm{doublefact}(1) = 1,\ \frac{\mathrm{doublefact}(n+1)}{\mathrm{doublefact}(n)} = 2\,n + 1$$

The series is alternating with decreasing summands, so a cut off is estimated by the first term being ignored.

But since the C code stops through available precision just look how many steps are performed:

```
> fct2 := define_external(
    'asymptSeries_length',
    'C',
    'x_str'::string[],
    'max_iter'::integer[4],
    RETURN::integer[4],
    LIB=myDLL):

  asymptSeries_length_lcc:=proc(x,max_iter)
```

```
  local X::string;
  X:=convert(x,string);
  if type(parse(X),numeric) then
    return fct2(X,max_iter);
  else
    return 'x';
  end if;
  end proc:
> `summation steps for the asymptotic series:`;
  for xTst in [25,31,80,213] do
   cat(`x: `,xTst,`, steps: `, asymptSeries_length_lcc(xTst,800));
  end do;
```

summation steps for the asymptotic series:

x: 25, steps: 139

x: 31, steps: 97

x: 80, steps: 49

x: 213, steps: 35

So practically the algorithm terminates quite fast.

```
>
>
>
```

C Sources

```
#include <math.h>
#include <qfloat.h>
#include <float.h>

qfloat asymptSeries(qfloat x, long max_iter)
{
 qfloat xi, s, t, a;
 long m;
 qfloat M;

 xi = 1/(x*x);
 a = - xi;
 s = a;

 for (m = 2; m <= max_iter; m++)  {
  ltoq(&m, &M);
    a = (-a * (2 * M - 1) * xi);
    t = (s + a);
    if (s == t)
      break;
    s = t;
  }

 return s;
}

qfloat asymptotic_cdfN(qfloat x)
// use: x negative
{
 qfloat inv2Pi =
  -1.59154943091895335768883763372514362034459645740456448747667344058896797634226535090113802766253085956072 8e-01q;
 qfloat LnSqrt2Pi =
  -9.18938533204672741780329736405617639861397473637783412817151540482765695927260397694743298635954197622005 6e-01q;
 qfloat pdfN_x;
 qfloat s;
 long max_iter;
 qfloat result;

 pdfN_x = expq( - x*x * 0.5q + LnSqrt2Pi );
```

```
 max_iter = (long)qtof(x*x/2);   // x^2/2
 max_iter = min(max_iter,400);

 s = asymptSeries(-x, max_iter);

 result = - pdfN_x / x ;
 result = result + result * s;

 if ( 0 < x){
  result = 1.0q - result;}

 return result;
}


qfloat cPhi(qfloat x)
// Marsaglia: Taylor series for (cdfN(x) - 1/2)/pdfN(x) around j = 0, ... , 25
{
 qfloat LnSqrt2Pi =
  -9.189385332046727417803297364056176398613974736377834128171515404827656959272603976947432986
359541976220056e-01q;
 qfloat v[] = {
    0.0q,

6.556795424187984715438712307308112833992823328704620280536861587341971657663105890658509564491
215012600982e-01q,

4.213692292880544732249343335423849787175989742468528301923547371893501920034390301063309931288
962320701227e-01q,

3.045902987101032957336125465157222019433208678573168488430658892071832212001930708032943678654
800826582081e-01q,

2.366523829135606706239859364358435477228059571994490412689618815487443066070118263484772190932
998252281611e-01q,

1.928081047153157648774657279175162514903027552850364922863857669482312511129223557450673889612
379356458957e-01q,

1.623776608968674618156821028189930010128542994863279730774141190295700788930702164042716215621
489111505075e-01q,

1.401041834530502415995345217963609882417902970134247752690267552643763159531694967288779220384
174732990035e-01q,

1.231319632579322962821807435171990762805541840630742108560590716686185238723021623245091920958
724115757457e-01q,

1.097872825783082912306378330560974402496327243214612809906067679249376671255826556711961745874
457584507795e-01q,

9.902859647173192139533718859531057834552235180489260921918698553467839262810816271560099557113
360476126506e-02q,

9.017567550106468227978035618587343537644767214221298015345528715728327957147306339022124943007
058881597300e-02q,

8.276628650136917725226505004190590186017464793697802713229186842248914936669251788942168000332
491725546423e-02q,

7.647576101624850299349519422151429783949526792767011418177424839831605837844981759771438310497
651373158119e-02q,

7.106958053885210709059684157805762144125098476868856935945753939801804487480360700926988575759
518515026233e-02q,

6.637423582325017359132388042985626393854873030872072151400440383054522249921796280344162047668
378045704438e-02q,

6.225866599502619577668594501385982403428460225871392927628466500564465541728323497894384286537
523236716191e-02q,

5.862206498001594387545342445861346105169066482303107727730444249305009602971555715651737977485
705254569518e-02q,

5.538565147010073446724673895523119101736235630550723518489211095351226878416187508893350898465
```

```
711868475986e-02q,

5.248698021967636423680460423147858175066935164468776932882560880066077078739928490385238434816
973069201891e-02q,

4.987592598183678365824056147354767742198420093918533070409054911537778916406088527153185734887
896783991959e-02q,

4.751179427627811302655541642239351474253636254344250051841169910995220397146195007429004920809
163017587744e-02q,

4.536120728999310087585168767321315277143983606034941045363135773233129414385284785058609647333
631784882632e-02q,

4.339653309551270406429944660727825229200146242050321012610122788124098481232007086626497681213
090902737556e-02q,

4.159470223257550541965761627938910863039370818248539195089614560773623792335240289924133854038
685735255716e-02q,

3.993630476953559252877870153820891350739023983558383620387256714088695710457904262719419208606
424367409706e-02q
  };

 qfloat          h, a, b, z, t, sum, pwr;
 int             i, j;
 qfloat          I0, I1;
 qfloat & tmp1 = 0.0q;

 qabs(absx);

 if ( absx <= QFLT_EPSILON)
  return 0.5q;

 tmp1 = absx + 1.0q;

 j = qtoi(tmp1); // j=fabs(x)+1.;

 z = 1.0q * j;

 h = absx - z; // h=fabs(x)-z;
 a = v[j];
 b = z * a - 1.0q;
 pwr = 1.0q;
 sum = a + h * b;

 for (i = 2; i < 200; i += 2)
 {
  ltoq(&i, &I0);
  I1 = I0 + 1.0q;
  a = (a + z * b) / I0; // a=(a+z*b)/i;
  b = (b + z * a) / I1; // b=(b+z*a)/(i+1);

  pwr = pwr * h * h;
  t = sum;
  sum = sum + pwr * (a + h * b);
  if (sum == t)
   break;
 }

 sum = sum * expq(-0.5q * x * x + LnSqrt2Pi);
 if (x < 0.0q){
  sum = 1.0q - sum;}

 return sum;
}


qfloat cdfN(qfloat x)
// the main application
{
 if ( x==0.0q){
  return 0.5q;}

 if ( -25q <= x && x <= 0.0q ){
  return cPhi(x);}
```

```
 if ( 0.0q <= x && x <= 25.0q){
  return 1.0q - cPhi(-x);}

 if (x < -213.1q){
  return 0.0q;}
 if (x < -25.0q){
  return asymptotic_cdfN(x);}
 if (23.0q < x){
  return 1.0q;}

 return 0.0q;
}


extern __declspec(dllexport) long __stdcall
str_cdfN_str_Maple(char * x_str, char * y_str)
// Maple interface
{
 qfloat          result;
 qfloat          X;

 asctoq(x_str, &X);

 result = cdfN(X);

 qtoasc(&result, y_str, min(strlen(y_str), 115));

 return strlen(y_str);
}


extern __declspec(dllexport) long __stdcall
asymptSeries_length(char * x_str, long max_iter)
// for tests
{
 qfloat xi, s, t, a;
 long m;
 qfloat M;

 qfloat          result;
 qfloat          x;

 asctoq(x_str, &x);

 xi = 1/(x*x);
 a = - xi;
 s = a;

 for (m = 2; m <= max_iter; m++)  {
  ltoq(&m, &M);
    a = (-a * (2 * M - 1) * xi);
    t = (s + a);
    if (s == t)
      break;
    s = t;
  }

 return m;
}
```

[>