The Cumulative Normal Distribution for Dimensions up to 3 using the qfloat floating-point Library from LCC-WIN32: Some Tests for Dimension = 3

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

The system library `qfloat.dll` should be in Window's system directory and the concurrent `cdfn123.dll` should be placed in the directory of this worksheet.

AVt, Jan 2006

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

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

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

Digits_lcc := 105

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

Digits := 210

For using the DLL locate its directory to call external functions from there:

```
> currentdir(): myDLL:=cat(%,`\\cdfn123.dll`);
```

myDLL := "C:_Work\Maple_Work_allTheStuff\Statistik\Verteilungen\TrivariateNormal\cdfn123.dll"

Accessing the DLL functions is through strings:

```
> lccstr:=proc(var)
    convert( evalf(parse(convert(var,string)),Digits_lcc + 10), string);
  end proc:
```

The cumulative normal distribution

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);
```

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

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

```
> fct_cdfN := define_external(
    'cdfN_string_Maple',
    'C',
    'x_str'::string[],
    'ret_str'::string[],
    RETURN::integer[4],
    LIB=myDLL):

  cdfN_lcc:=proc(x)
    local X::string, result::string;
    result:=StringTools:-Fill( `0` , Digits_lcc+10);
    X:=lccstr(x):
    if type(parse(X),numeric) then
      fct_cdfN(X,result);
      return parse(result);
```

```
    else
      return 'cdfN_lcc(x)';
    end if;
  end proc: #maplemint(%);
```

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 ...). So a simple procedure is used as interface.

The Bivariate Case

The bivariate normal distribution can be written as:

```
> pdfN2:= (x,y,rho) ->
    1/sqrt(1-rho^2)/(2*Pi)*exp(-(x^2-2*rho*x*y+y^2)/(2*(1-rho^2)));
  ``;
  cdfN2:= (x,y,rho) ->
    Int(Int( pdfN2(xi,eta,rho),   eta=-infinity..y),xi=-infinity..x);
```

$$\text{pdfN2} := (x, y, \rho) \to \frac{1}{2} \frac{e^{\left(-\frac{x^2 - 2\rho x y + y^2}{2 - 2\rho^2}\right)}}{\sqrt{1 - \rho^2}\,\pi}$$

$$\text{cdfN2} := (x, y, \rho) \to \int_{-\infty}^{x} \int_{-\infty}^{y} \text{pdfN2}(\xi, \eta, \rho)\, d\eta\, d\xi$$

It is coded within the DLL giving 104 decimal points of precision and can be accessed as follows:

```
> fct_cdfN2 := define_external(
    'cdfN2_string_Maple',
    'C',
    'x_str'::string[],
    'y_str'::string[],
    'r_str'::string[],
    'ret_str'::string[],
    RETURN::integer[4],
    LIB=myDLL):

  cdfN2_lcc:=proc(x,y,r)
    local X::string, Y::string, R::string, result;

    result:=StringTools:-Fill( `0` , Digits_lcc+10);
    X:=lccstr(x);
    Y:=lccstr(y);
    R:=lccstr(r);

    if type(parse(X),numeric) and type(parse(Y),numeric) and
  type(parse(R),numeric) then
      fct_cdfN2(X,Y,R,result);
      return parse(result);
    else
      return 'cdfN2_lcc(x,y,r)';
    end if;
  end proc:
```

This is fast, about 10 milli seconds.

But difficult to test. For that use a re-formulation:

```
> 'cdfN2(x,y,rho)'= 'Int(pdfN(tau)*cdfN((y-rho*tau)/sqrt(1-rho^2)),tau =
  -infinity .. x)';
```

$$\mathrm{cdfN2}(x, y, \rho) = \int_{-\infty}^{x} \mathrm{pdfN}(\tau)\, \mathrm{cdfN}\left(\frac{y - \rho\,\tau}{\sqrt{1-\rho^2}}\right) d\tau$$

For dimension 3 a similar reduction is available and attributed to D B Owen.

```
>
```

Dimension = 3

The trivariate case can be handled through semi-definite integrals using bivariate normal distributions, cf Alan Genz, *Numerical Computation of Rectangular Bivariate and Trivariate ...* http://www.sci.wsu.edu/math/faculty/genz/homepage

To see that the used integration routine is highly exact as an example just compare what happens for integrating over the usual pdfN (that is coded in the DLL only for a test):

For directly inputting constants it not clear that they belong to a correlation matrix, so provide a check:

```
> check_coefficients:=proc(_r12,_r13,_r23)
  local R, det, ev, remDigits;

  remDigits:=Digits;
  Digits:=18;
  R := Matrix(
    [[1,_r12,_r13],
         [1,_r23],
            [1]],
    shape=symmetric,
    scan=triangular[upper]);

  if (LinearAlgebra:-IsDefinite(R, query=positive_definite)) then
    det:=evalf(LinearAlgebra:-Determinant(R), 2*Digits);
    det:=evalf(det,6);
    print(`valid correlation matrix`, `determinant `=det);
    Digits:=remDigits;
    return 0;
  else
    print(`not a correlation matrix!`);
    Digits:=remDigits;
    return -1;
  end if;
  Digits:=remDigits;
  return;
  end proc:
```

Provide an interface to the DLL ...

```
> fct3 := define_external(
    'cdfN3_string_Maple',
    'C',
    'x1_str'::string[], 'x2_str'::string[], 'x3_str'::string[],
```

```
    'r12_str'::string[], 'r13_str'::string[], 'r23_str'::string[],
    'ret_str'::string[],
    RETURN::integer[4],
    LIB=myDLL):

  cdfN3_lcc:=proc(x1,x2,x3,r12,r13,r23)
  #local X::string, Y::string, R::string, result;
  local X1, X2, X3, R12, R13, R23, result, lccstr;

  lccstr:=proc(var)
    convert( evalf(parse(convert(var,string)),105), string);
  end proc:

  result:=StringTools:-Fill( `0` , Digits_lcc+10);
  #X1:=convert(x1,string); X2:=convert(x2,string); X3:=convert(x3,string);
  #R12:=convert(r12,string); R13:=convert(r13,string);
  R23:=convert(r23,string);
  X1:=lccstr(x1): X2:=lccstr(x2): X3:=lccstr(x3):
  R12:=lccstr(r12): R13:=lccstr(r13): R23:=lccstr(r23):

  if type(parse(X1),numeric) then
    fct3(X1, X2, X3, R12, R13, R23,result);
    return parse(result);
  else
    return 'x';
  end if;
  end proc:
```

To cross check results one can use a formula, which is attributed to Owen:

```
> 'cdfN3'='Int(pdfN(xi)*F_Owen(xi), xi= -infinity .. x1)';
  F_Owen:= (xi,x2,x3, rho12,rho13,rho23) ->
    cdfN2(
    (x2 - rho12 * xi) / sqrt(1 - rho12*rho12),
    (x3 - rho13 * xi) / sqrt(1 - rho13*rho13),
    (rho23 - rho13 * rho12) / sqrt( (1 - rho12*rho12)*(1 - rho13*rho13) )
    );
```

$$\mathrm{cdfN3} = \int_{-\infty}^{x1} \mathrm{pdfN}(\xi)\ \mathrm{F\_Owen}(\xi)\ d\xi$$

$$\mathrm{F\_Owen} := (\xi, x2, x3, \rho12, \rho13, \rho23) \to \mathrm{cdfN2}\left(\frac{x2 - \rho12\,\xi}{\sqrt{1 - \rho12\,\rho12}}, \frac{x3 - \rho13\,\xi}{\sqrt{1 - \rho13\,\rho13}}, \frac{\rho23 - \rho13\,\rho12}{\sqrt{(1 - \rho12\,\rho12)\,(1 - \rho13\,\rho13)}}\right)$$

That reduces the problem to dimension 2. But it will be very slow for 100 Digits and for improving speed I would use an external implementation for cdfN2 - which would make the test depend on a repeated use my implementation in dim 2.

But one can reduce to dimension one through partial integration:

```
> 'Int(pdfN(x)*F(x),x=-infinity..x1)' =
  'F(x1)*cdfN(x1)-int(D(F)(x)*cdfN(x),x = -infinity .. x1)';
  ``;
  'diff(cdfN2(a1+b1*x, a2+b2*x,r),x)' =
    'b2*pdfN(b2*x+a2)*cdfN((a1-r*a2+(b1-r*b2)*x)/(1-r^2)^(1/2)) +
     b1*pdfN(b1*x+a1)*cdfN((a2-r*a1+(b2-r*b1)*x)/(1-r^2)^(1/2))';
```

$$\int_{-\infty}^{x1} \mathrm{pdfN}(x)\,F(x)\,dx = F(x1)\,\mathrm{cdfN}(x1) - \int_{-\infty}^{x1} D(F)(x)\,\mathrm{cdfN}(x)\,dx$$

$$\frac{\partial}{\partial x}\mathrm{cdfN2}(a1+b1\,x,\,a2+b2\,x,\,r) = b2\,\mathrm{pdfN}(a2+b2\,x)\,\mathrm{cdfN}\left(\frac{a1 - r\,a2 + (b1 - r\,b2)\,x}{\sqrt{1-r^2}}\right)$$
$$+ b1\,\mathrm{pdfN}(a1+b1\,x)\,\mathrm{cdfN}\left(\frac{a2 - r\,a1 + (b2 - r\,b1)\,x}{\sqrt{1-r^2}}\right)$$

where the constant have to be to evaluated as

```
> L:={
  b1 = -r12/(1-r12^2)^(1/2), a1 = x2/(1-r12^2)^(1/2),
  b2 = -r13/(1-r13^2)^(1/2), a2 = x3/(1-r13^2)^(1/2),
  r = (r23 - r13 * r12) / sqrt( (1 - r12*r12)*(1 - r13*r13) )};
```

$$L := \left\{b2 = -\frac{r13}{\sqrt{1-r13^2}},\ a2 = \frac{x3}{\sqrt{1-r13^2}},\ b1 = -\frac{r12}{\sqrt{1-r12^2}},\ a1 = \frac{x2}{\sqrt{1-r12^2}},\ r = \frac{r23 - r13\,r12}{\sqrt{(1-r12^2)\,(1-r13^2)}}\right\}$$

For short let us write it as

```
> theIntegrand:= 'cdfN(xi)' *
   '(b2*pdfN(b2*xi+a2)*cdfN((a1-r*a2+(b1-r*b2)*xi)/(1-r^2)^(1/2)) +
     b1*pdfN(b1*xi+a1)*cdfN((a2-r*a1+(b2-r*b1)*xi)/(1-r^2)^(1/2)))';
  ``;
  'cdfN3' =
  'F(x1)*cdfN(x1)-Int(theIntegrand,x = -infinity .. x1)';
  `F(x)` = 'cdfN2(a1+b1*x, a2+b2*x,r)';
```

$$\mathrm{theIntegrand} := \mathrm{cdfN}(\xi)\left(b2\,\mathrm{pdfN}(b2\,\xi+a2)\,\mathrm{cdfN}\left(\frac{a1 - r\,a2 + (b1 - r\,b2)\,\xi}{\sqrt{1-r^2}}\right)\right.$$
$$\left. + b1\,\mathrm{pdfN}(b1\,\xi+a1)\,\mathrm{cdfN}\left(\frac{a2 - r\,a1 + (b2 - r\,b1)\,\xi}{\sqrt{1-r^2}}\right)\right)$$

$$\mathrm{cdfN3} = F(x1)\,\mathrm{cdfN}(x1) - \int_{-\infty}^{x1} \mathrm{theIntegrand}\,dx$$

$$F(x) = \mathrm{cdfN2}(a1+b1\,x,\,a2+b2\,x,\,r)$$

```
>
>
```

Test 1

```
> x1 := 1.000000000;
  x2 := 0.330000000;
  x3 := -0.500000000;

  r12 := 0.9;
  r13 := 0.7;
  r23 := 0.8;

  st:=time():
  if 0 <= check_coefficients(r12,r13,r23) then
    cdfN3_lcc(x1,x2,x3,r12,r13,r23);
    #evalf(%,16);
  end if;
  `seconds`=time()-st;
```

x1 := 1.000000000

x2 := 0.330000000

x3 := -0.500000000

r12 := 0.9

r13 := 0.7

r23 := 0.8

valid correlation matrix, determinant = 0.068

0.29611483371358269602046893516892354569704844906801982803743386674282929601356874095\
2\
5431104416122689731844

seconds = 5.419

Test it with the modification of Owen's formula

```
> TheIntegrand:='eval(theIntegrand, L)';
  TheIntegral:= 'Int(theIntegrand,xi = -infinity .. x1)';
```

$$\text{TheIntegrand} := \text{eval}(\text{theIntegrand}, L)$$

$$\text{TheIntegral} := \int_{-\infty}^{x1} \text{theIntegrand}\, d\xi$$

Since the integrand contains $\text{cdfN}(\xi)$ as factor I will cut off against $-\infty$ (as it will be below 1E-105):

```
> TheIntegral:= 'Int(TheIntegrand,xi = -23.0 .. x1, method = _Gquad)';
```

$$\text{TheIntegral} := \text{Int}(\text{TheIntegrand}, \xi = -23.0 .. x1, \text{method} = \_Gquad)$$

Then Maple gives the same result as the DLL implementation:

```
> 'TheIntegral'=evalf[105](TheIntegral):
  `eval(cdfN2_lcc(a1+b1*x1, a2+b2*x1,r)*cdfN(x1), L) - TheIntegral =`;
  evalf(eval(cdfN2_lcc(a1+b1*x1, a2+b2*x1,r)*cdfN(x1), L) -TheIntegral,105);
```

eval(cdfN2_lcc(a1+b1*x1, a2+b2*x1,r)*cdfN(x1), L) - TheIntegral =

0.29611483371358269602046893516892354569704844906801982803743386674282929601356874095\
2\
5431104416122689731884

Test 2

If the determinant is close to zero the correlation coefficients are choosen from
a valid matrix R (cf the Genz paper):

```
> theta1 := 0.01;
  theta2 := 0.01; #1 - 1e-6;
  theta3 := -0.02;

  R:='R': A:='A': `R`= A*A^t;
  `A`=Matrix('[[1,0,0],
    [cos(theta1*Pi),sin(theta1*Pi),0],

  [cos(theta2*Pi)*cos(theta3*Pi),cos(theta2*Pi)*sin(theta3*Pi),sin(theta2*Pi
  )]]');
  rhs(%):
  A:=evalf(%): #evalf(evalhf(%),15);
  LinearAlgebra:-Multiply(A, LinearAlgebra:-Transpose(A)):
  R:=evalf(evalhf(%),15); A:='A':
  ``;
  r12:=R[1,2];
```

```
r13:=R[1,3];
r23:=R[2,3];

check_coefficients(r12,r13,r23):
```

$$\theta 1 := 0.01$$

$$\theta 2 := 0.01$$

$$\theta 3 := -0.02$$

$$R = A\,A^{t}$$

$$A = \begin{bmatrix} 1 & 0 & 0 \\ \cos(\theta 1\,\pi) & \sin(\theta 1\,\pi) & 0 \\ \cos(\theta 2\,\pi)\cos(\theta 3\,\pi) & \cos(\theta 2\,\pi)\sin(\theta 3\,\pi) & \sin(\theta 2\,\pi) \end{bmatrix}$$

$$R := \begin{bmatrix} 1. & 0.999506560365732 & 0.997534262484406 \\ 0.999506560365732 & 1. & 0.995070714871375 \\ 0.997534262484406 & 0.995070714871375 & 1. \end{bmatrix}$$

$$r12 := 0.999506560365732$$

$$r13 := 0.997534262484406$$

$$r23 := 0.995070714871375$$

valid correlation matrix, determinant $= 0.973450\ 10^{-6}$

```
> x1 := 1.000000000;
  x2 := 0.330000000;
  x3 := -0.400000000;
```

$$x1 := 1.000000000$$

$$x2 := 0.330000000$$

$$x3 := -0.400000000$$

Here the computation needs not much more time to achieve the desired accuracy:

```
> st:=time():
  if 0 <= check_coefficients(r12,r13,r23) then
    print(`cdfN3 =`);
    cdfN3_lcc(x1,x2,x3,r12,r13,r23);
    #evalf(%,16);
  end if;
  `seconds`=time()-st;
```

valid correlation matrix, determinant $= 0.973450\ 10^{-6}$

cdfN3 =

0.3445782583896753255040535543496512702040655159855600014161076910047810789311874264513809570222070725505 23

seconds = 6.795

```
> TheIntegrand:='eval(theIntegrand, L)';
  TheIntegral:= 'Int(TheIntegrand,xi = -23.0 .. x1, method = _Gquad)';
```

$$\text{TheIntegrand} := \text{eval}(\text{theIntegrand}, L)$$

$$\text{TheIntegral} := \int_{-23.0}^{x1} \text{TheIntegrand}\,d\xi \quad \text{method} = \text{\_Gquad}$$

```
> 'TheIntegral'=evalf[105](TheIntegral):
  `eval(cdfN2_lcc(a1+b1*x1, a2+b2*x1,r)*cdfN(x1), L) - TheIntegral =`;
  evalf(eval(cdfN2_lcc(a1+b1*x1, a2+b2*x1,r)*cdfN(x1), L) -TheIntegral,105);
```

eval(cdfN2_lcc(a1+b1*x1, a2+b2*x1,r)*cdfN(x1), L) - TheIntegral =

0.3445782583896753255040535543496512702040655159855600014161076910047810789311874264513809570222070725505 24

Again up to the last decimal place the results coincide.

Test 3

```
> theta1 := 0.01;
  theta2 := 1 - 1e-5; #1 - 1e-6;
  theta3 := -0.02;

  R:='R': A:='A': `R`= A*A^t;
  `A`=Matrix('[[1,0,0],
    [cos(theta1*Pi),sin(theta1*Pi),0],

  [cos(theta2*Pi)*cos(theta3*Pi),cos(theta2*Pi)*sin(theta3*Pi),sin(theta2*Pi
  )]]'):
  rhs(%):
  A:=evalf(%): #evalf(evalhf(%),15);
  LinearAlgebra:-Multiply(A, LinearAlgebra:-Transpose(A)):
  R:=evalf(evalhf(%),15); A:='A':
  ``;
  r12:=R[1,2];
  r13:=R[1,3];
  r23:=R[2,3];

  check_coefficients(r12,r13,r23):
```

$$\theta 1 := 0.01$$

$$\theta 2 := 0.99999$$

$$\theta 3 := -0.02$$

$$R = A\ A^{t}$$

$$R := \begin{bmatrix} 1. & 0.999506560365732 & -0.998026727935765 \\ 0.999506560365732 & 1. & -0.995561964111790 \\ -0.998026727935765 & -0.995561964111790 & 1. \end{bmatrix}$$

$$r12 := 0.999506560365732$$

$$r13 := -0.998026727935765$$

$$r23 := -0.995561964111790$$

valid correlation matrix, determinant $= 0.973766\ 10^{-12}$

```
> x1 := 1.000000000;
  x2 := 0.330000000;
  x3 := -0.400000000;
```

$$x1 := 1.000000000$$

$$x2 := 0.330000000$$

$$x3 := -0.400000000$$

```
> st:=time():
  if 0 <= check_coefficients(r12,r13,r23) then
    print(`cdfN3 =`);
    cdfN3_lcc(x1,x2,x3,r12,r13,r23);
    #evalf(%,16);
  end if;
  `seconds`=time()-st;
```

valid correlation matrix, determinant $= 0.973766\ 10^{-12}$

cdfN3 =

0.00467092420241752960787766136913876150807713123882978349949810853673416980367382085\
0\
6967835592791537634799

seconds = 4.786

For that case Owen's function is already difficult to use:

```
> theF := proc(xi,x2,x3, rho12,rho13,rho23)
  local a1,b1,a2,b2,corr, result,r;

  b1 := -rho12/(1-rho12^2)^(1/2); a1 := x2/(1-rho12^2)^(1/2);
  b2 := -rho13/(1-rho13^2)^(1/2); a2 := x3/(1-rho13^2)^(1/2);
  corr := (rho23 - rho13 * rho12) / sqrt( (1 - rho12*rho12)*(1 -
  rho13*rho13) );

  r:=corr;
  #[(a1-r*a2+(b1-r*b2)*x)/(1-r^2)^(1/2),(a2-r*a1+(b2-r*b1)*x)/(1-r^2)^(1/2)]
  ;

  return cdfN2_lcc(a1+b1*xi, a2+b2*xi,corr); #cdfN2(a1+b1*x, a2+b2*x,corr);
  end proc;
```

theF := **proc**(ξ, x2, x3, ρ12, ρ13, ρ23)

local a1, b1, a2, b2, corr, result, r;

b1 := –ρ12 / (1 – ρ12^2)^(1 / 2);

a1 := x2 / (1 – ρ12^2)^(1 / 2);

b2 := –ρ13 / (1 – ρ13^2)^(1 / 2);

a2 := x3 / (1 – ρ13^2)^(1 / 2);

corr := (ρ23 – ρ13*ρ12) / sqrt((1 – ρ12*ρ12)*(1 – ρ13*ρ13));

r := corr;

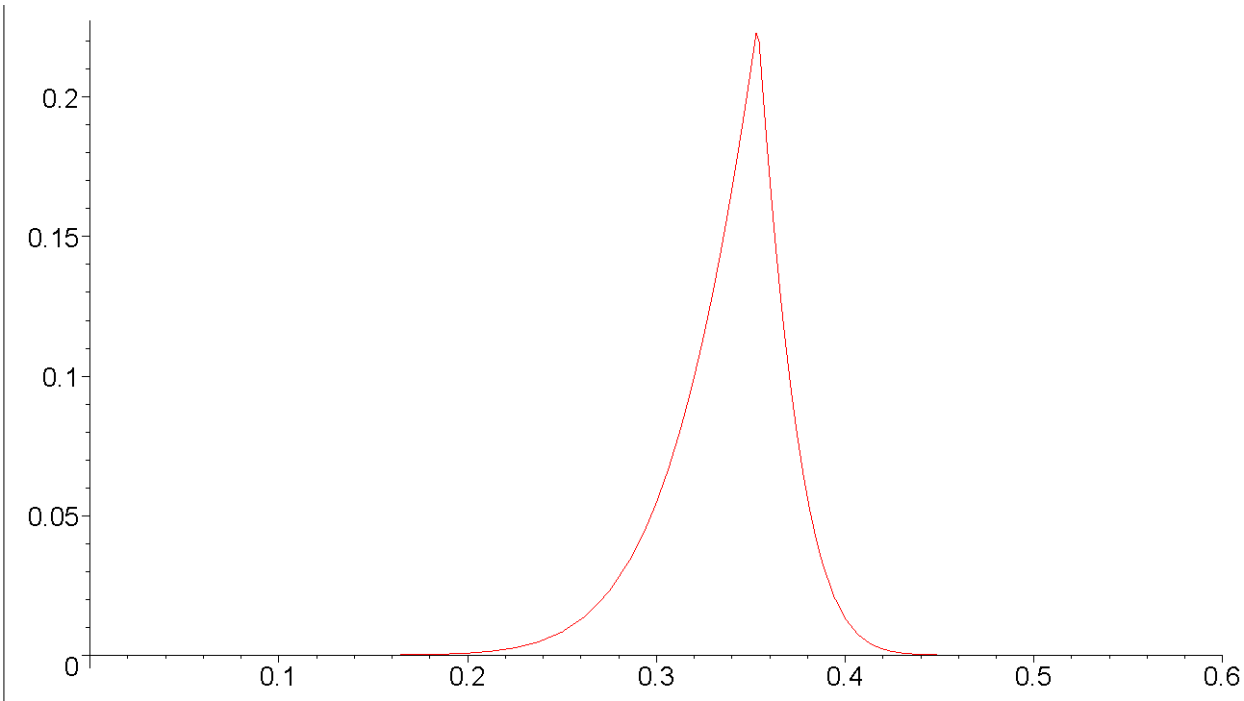
return cdfN2_lcc(a1 + b1*ξ, a2 + b2*ξ, corr)

end proc

```
> theF(xi,x2,x3, r12,r13,r23):
  tst:=unapply(%,xi): # Owen's function for the test data
```

```
> remDigits:=Digits: Digits:=14;
  plot(tst, 0..0.6); #plot(tst, 0.353685..0.353686);
  mid:=0.3536856;
  tst(mid): evalf(%);
  Digits:=remDigits:
```

Digits := 14



mid := 0.3536856

0.22698051973509

Even if the function is actually analytic it has a numerical peak.

To avoid problem in applying the modification one re-arranges the integrand to be positive (zero correspond to a peak through D(F)), i.e. rho12, rho13 have the same sign, which I do by hand here (and sketch the way):

```
> `sigma^(-1): 1 -> 3, 2 -> 1, 3 -> 2`;
  `sigma:      1 -> 2, 2 -> 3, 3 -> 1`;
```

sigma^(-1): 1 -> 3, 2 -> 1, 3 -> 2

sigma: 1 -> 2, 2 -> 3, 3 -> 1

```
> `sigma gives x_new` = 'x3, x1, x3';
```

sigma gives x_new = (x3, x1, x3)

```
> `tau` = `sigma^(-1)`;
  'r_new[i,j]' = r_old[tau(i),tau(j)];
  'r_new[i,j]' = r_old[tau(j),tau(i)];
```

$\tau = \text{sigma}^{(-1)}$

$\text{r\_new}_{i,j} = \text{r\_old}_{\tau(i),\tau(j)}$

$\text{r\_new}_{i,j} = \text{r\_old}_{\tau(j),\tau(i)}$

```
> 'r_new[1,2]' = r[1,3]; #rho[3,1];
  'r_new[1,3]' = r[2,3]; #rho[3,2];
  'r_new[2,3]' = r[1,2];
```

$\text{r\_new}_{1,2} = r_{1,3}$

$\text{r\_new}_{1,3} = r_{2,3}$

$\text{r\_new}_{2,3} = r_{1,2}$

```
> # new values
  r12 := -0.998026727935765;
  r13 := -0.995561964111790;
  r23 := 0.999506560365732;
```

r12 := -0.998026727935765

r13 := -0.995561964111790

r23 := 0.999506560365732

```
> # new values
  x1 := -0.400000000;
  x2 := 1.000000000;
  x3 := 0.330000000;
```

x1 := -0.400000000

x2 := 1.000000000

x3 := 0.330000000

For the DLL that change does not matter:

```
> st:=time():
  if 0 <= check_coefficients(r12,r13,r23) then
    print(`cdfN3 =`);
    cdfN3_lcc(x1,x2,x3,r12,r13,r23);
    #evalf(%,16);
  end if;
  new_result_from_DLL:=%:
  `seconds`=time()-st;
```

valid correlation matrix, determinant = $0.973766\ 10^{-12}$

cdfN3 =

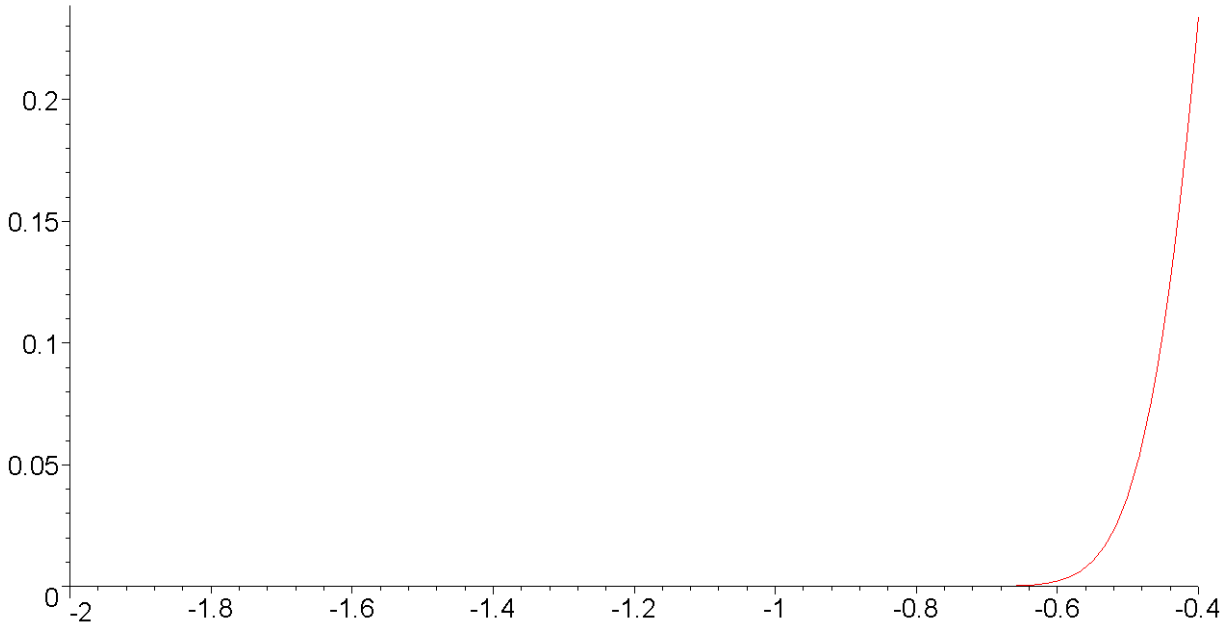
0.00467092420241752960787766136913876150807713123882978349949810853673416980367382085\
0\
6967835592791537634799 2

seconds = 4.786

but Owen's function now looks better to handle:

```
> theF(xi,x2,x3, r12,r13,r23):
  tst:=unapply(%,xi):
> remDigits:=Digits: Digits:=14;
  plot(tst, -2.. x1);
  Digits:=remDigits:
```

Digits := 14




```
> 
> TheIntegrand:='eval(theIntegrand, L)';
  TheIntegral:= 'Int(TheIntegrand,xi = -23.0 .. x1, method = _Gquad)';
```

$$\text{TheIntegrand} := \text{eval}(\text{theIntegrand}, L)$$

$$\text{TheIntegral} := \text{Int}(\text{TheIntegrand}, \xi = -23.0 \ldots x1, \text{method} = \_\text{Gquad})$$

```
> 'TheIntegral'=evalf[105](TheIntegral):
  `eval(cdfN2_lcc(a1+b1*x1, a2+b2*x1,r)*cdfN(x1), L) - TheIntegral =`;
  evalf(eval(cdfN2_lcc(a1+b1*x1, a2+b2*x1,r)*cdfN(x1), L) -TheIntegral,105);
```

eval(cdfN2_lcc(a1+b1*x1, a2+b2*x1,r)*cdfN(x1), L) - TheIntegral =

0.00467092420241752960787766136913876150807713123882978349949810853673416980367382085\
0\
69678355927915376347 70

and up to the last places that approach gives the same

Test 4

```
> theta1 := 1e-3;
  theta2 := 1 - 1e-6; #1 - 1e-6;
  theta3 := -1e-3;

  R:='R': A:='A': `R`= A*A^t;
  `A`=Matrix('[[1,0,0],
    [cos(theta1*Pi),sin(theta1*Pi),0],

  [cos(theta2*Pi)*cos(theta3*Pi),cos(theta2*Pi)*sin(theta3*Pi),sin(theta2*Pi
  )]]'):
  rhs(%):
  A:=evalf(%): #evalf(evalhf(%),15);
  LinearAlgebra:-Multiply(A, LinearAlgebra:-Transpose(A)):
  R:=evalf(evalhf(%),15); A:='A':
  ``;
  r12:=R[1,2];
  r13:=R[1,3];
  r23:=R[2,3];

  check_coefficients(r12,r13,r23):
```

$$\theta 1 := 0.001$$

$$\theta 2 := 0.999999$$

$$\theta 3 := -0.001$$

$$R = A\,A^{t}$$

$$R := \begin{bmatrix} 1. & 0.999995065201858 & -0.999995065196923 \\ 0.999995065201858 & 1. & -0.999980260851202 \\ -0.999995065196923 & -0.999980260851202 & 1. \end{bmatrix}$$

$$r12 := 0.999995065201858$$

$$r13 := -0.999995065196923$$

$$r23 := -0.999980260851202$$

$$\text{valid correlation matrix, determinant } = 0.974223\ 10^{-16}$$

```
> x1 := 1.000000000;
  x2 := 0.330000000;
  x3 := -0.400000000;
```

$$x1 := 1.000000000$$

$$x2 := 0.330000000$$

$$x3 := -0.400000000$$

For that case Owen's function can not be used, it would evaluate to zero:

```
> theF(xi,x2,x3, r12,r13,r23):
  tst:=unapply(%,xi):
  tst(xi): op(%): evalf(%,8): cdfN2_lcc(%);
```

$$\mathrm{cdfN2\_lcc}(105.04244 - 318.30884\,\xi,\ -127.32410 + 318.30868\,\xi,\ 0.99999950)$$

Generally the DLL uses sorting on the correlation coefficients and takes care that the integral is provided with a 'positive integrand' as follows:

```
> arrangeXR:=proc(x,r)
  local R,X,swapR,swapX;

  X := [x[1],x[2],x[3]];
  R := [r[1],r[2],r[3]];

  if R[2] < R[1] then # swapR them
    swapR:= R[2];
    R[2]:=R[1];
    R[1]:=swapR;

    swapX:= X[2];
    X[2]:=X[3];
    X[3]:=swapX;
  end if;
  #print(R);

  if R[3] < R[2] then # swapR them
    swapR:= R[3];
    R[3]:=R[2];
    R[2]:=swapR;

    swapX:=X[1];
    X[1]:=X[2];
    X[2]:=swapX;
  end if;
  #print(R);

  if R[2] < R[1] then # swapR them
    swapR:=R[2];
    R[2]:=R[1];
    R[1]:=swapR;

    swapX:= X[2];
    X[2]:=X[3];
    X[3]:=swapX;
  end if;
  #print([X,R]);

  if ( R[3] < 0 or 0 < R[1] or R[1] = 0 or R[2] = 0) then
  else
    if 0 < R[2] then
  print("we do it ", [X,R]);
      swapR:=R[1];
      R[1]:=R[2];
      R[2]:=R[3];
      R[3]:=swapR;

      swapX:=X[1];
      X[1]:=X[3];
      X[3]:=X[2];
      X[2]:=swapX;
    end if;
```

```
   end if;

   return [X,R];
   end proc;
arrangeXR := proc(x, r)
local R, X, swapR, swapX;
     X := [ x[ 1 ], x[ 2 ], x[ 3 ] ];
     R := [ r[ 1 ], r[ 2 ], r[ 3 ] ];
     if R[ 2 ] < R[ 1 ] then
          swapR := R[ 2 ]; R[ 2 ] := R[ 1 ]; R[ 1 ] := swapR; swapX := X[ 2 ]; X[ 2 ] := X[ 3 ]; X[ 3 ] := swapX
     end if;
     if R[ 3 ] < R[ 2 ] then
          swapR := R[ 3 ]; R[ 3 ] := R[ 2 ]; R[ 2 ] := swapR; swapX := X[ 1 ]; X[ 1 ] := X[ 2 ]; X[ 2 ] := swapX
     end if;
     if R[ 2 ] < R[ 1 ] then
          swapR := R[ 2 ]; R[ 2 ] := R[ 1 ]; R[ 1 ] := swapR; swapX := X[ 2 ]; X[ 2 ] := X[ 3 ]; X[ 3 ] := swapX
     end if;
     if R[ 3 ] < 0 or 0 < R[ 1 ] or R[ 1 ] = 0 or R[ 2 ] = 0 then
     else
          if 0 < R[ 2 ] then
               print( "we do it ", [ X, R ] );
               swapR := R[ 1 ];
               R[ 1 ] := R[ 2 ];
               R[ 2 ] := R[ 3 ];
               R[ 3 ] := swapR;
               swapX := X[ 1 ];
               X[ 1 ] := X[ 3 ];
               X[ 3 ] := X[ 2 ];
               X[ 2 ] := swapX
          end if
     end if;
     return [ X, R ]
end proc
```

I simply use that for the test

```
> X:=[x1,x2,x3]: R:=[r12,r13,r23]:
  LL:=arrangeXR(X,R):

  # new values
  r12 := LL[2][1];
  r13 := LL[2][2];
  r23 := LL[2][3];

  x1 := LL[1][1];
  x2 := LL[1][2];
  x3 := LL[1][3];
```

r12 := -0.999995065196923

r13 := -0.999980260851202

r23 := 0.999995065201858

x1 := -0.400000000

x2 := 1.000000000

x3 := 0.330000000

```
> st:=time():
  if 0 <= check_coefficients(r12,r13,r23) then
    print(`cdfN3 =`);
    cdfN3_lcc(x1,x2,x3,r12,r13,r23);
    #evalf(%,16);
  end if;
  new_result_from_DLL:=%:
  `seconds`=time()-st;
```

valid correlation matrix, determinant = $0.974223\ 10^{-16}$

cdfN3 =

0.8221621925275981967888479621309472494426606399606260516581369812774939422650677658745044067167190879386083 10^{-32}

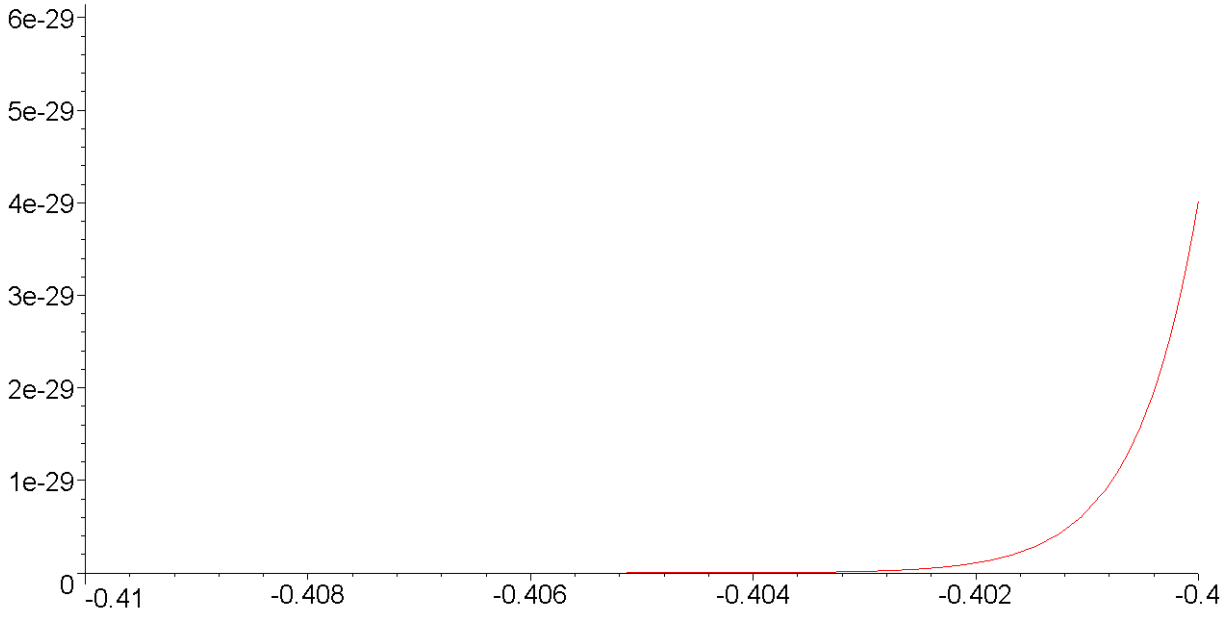
seconds = 4.376

where Owen's function becomes somewhat nicer at least:

```
> theF(xi,x2,x3, r12,r13,r23):
  tst:=unapply(%,xi):
  tst(xi): op(%): evalf(%,8): cdfN2_lcc(%);

  remDigits:=Digits: Digits:=14:
  plot(tst, -0.41.. x1);
  Digits:=remDigits:
```

cdfN2_lcc(318.31025 + 318.30868 ξ, 52.521470 + 159.15283 ξ, 0.99999987)



Again compare both results:

```
> TheIntegrand:='eval(theIntegrand, L)';
  TheIntegral:= 'Int(TheIntegrand,xi = -23.0 .. x1, method = _Gquad)';
  ``;
  'TheIntegral'=evalf[105](TheIntegral):
  `eval(cdfN2_lcc(a1+b1*x1, a2+b2*x1,r)*cdfN(x1), L) - TheIntegral =`;
  evalf(eval(cdfN2_lcc(a1+b1*x1, a2+b2*x1,r)*cdfN(x1), L)
```

```
-TheIntegral,105);
`computational error` = % -new_result_from_DLL;
```

$$\text{TheIntegrand} := \text{eval}(\text{theIntegrand}, L)$$

$$\text{TheIntegral} := \text{Int}(\text{TheIntegrand}, \xi = -23.0 \ldots x1, \text{method} = \_\text{Gquad})$$

$$\text{eval}(\text{cdfN2\_lcc}(a1+b1*x1, a2+b2*x1,r)*\text{cdfN}(x1), L) - \text{TheIntegral} = 0.82216219252759819678884796213094724944266063996062605165813698127749394199330233639784224873905886620\ 10^{-32}$$

$$\text{computational error} = -0.27176542947620181842813201318608 3\ 10^{-104}$$

The computational error is less than 1E-104 which is the exactness of qfloat (it is not in digits, but decimal places).

Test 5

The correlation coefficients need not be close to 1 to give an ugly matrix neither the determinant need to be extremly small to cause difficulties:

```
> r12:=1/2;
  r13:=1/2;
  r23:=-1/2 + 10^(-3); r23 := evalf(r23):

  check_coefficients(r12,r13,r23):
```

$$r12 := \frac{1}{2}$$

$$r13 := \frac{1}{2}$$

$$r23 := \frac{-499}{1000}$$

valid correlation matrix, determinant = 0.00149900

```
> x1 := 1.000000000;
  x2 := 0.330000000;
  x3 := -0.400000000;
```

$$x1 := 1.000000000$$

$$x2 := 0.330000000$$

$$x3 := -0.400000000$$

The DLL asserts an result in moderate time:

```
> st:=time():
  if 0 <= check_coefficients(r12,r13,r23) then
    print(`cdfN3 =`);
    cdfN3_lcc(x1,x2,x3,r12,r13,r23);
    #evalf(%,16);
  end if;
  new_result_from_DLL:=%:
  `seconds`=time()-st;
```

valid correlation matrix, determinant = 0.00149900

$$\text{cdfN3} = 0.14229639400313725770331360633704182047280310214336430236413564182082038407186632458973302887020297000327 5120$$

$$\text{seconds} = 8.416$$

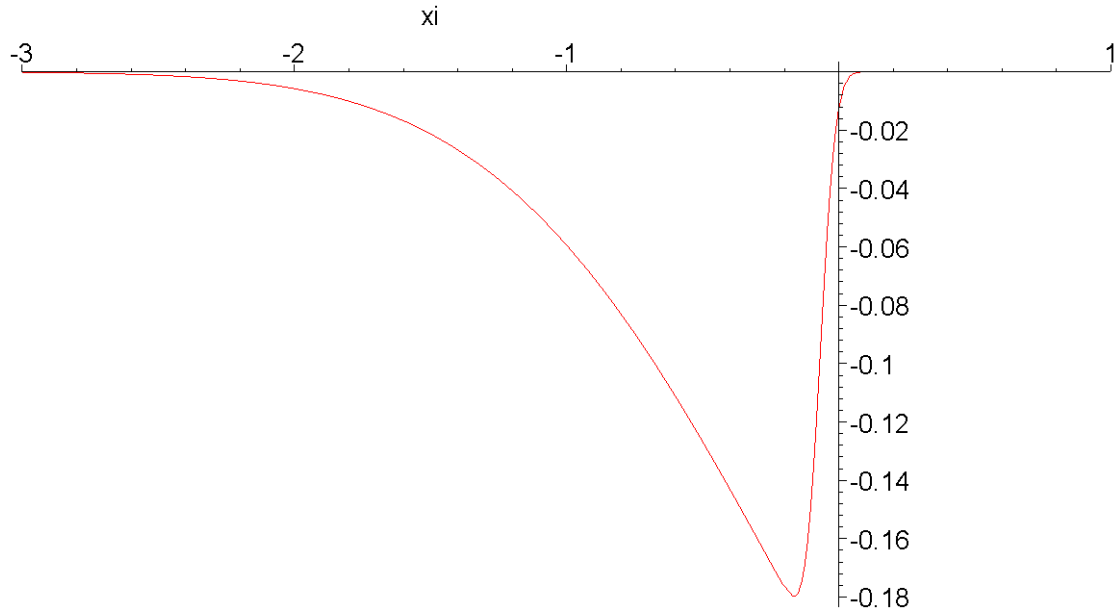
```
> TheIntegrand:='eval(theIntegrand, L)';
  TheIntegral:= 'Int(evalf(TheIntegrand),xi = -23.0 .. x1)';
```

$$\text{TheIntegrand} := \text{eval}(\text{theIntegrand}, L)$$

$$\text{TheIntegral} := \int_{-23.0}^{x1} \text{evalf}(\text{TheIntegrand})\, d\xi$$

And looking at the integrand Maple should be able to compute that (otherwise split it into pieces, for r23 towards -1/2 it becomes quite ugly ... I lost patients with Maple to integrate then and did not work out an example for that):

```
> remDigits:=Digits: Digits:=14:
  plot(TheIntegrand,xi = -3.0 .. x1);
  DI:=evalf(diff(TheIntegrand,xi)): expand(%): Tryhard(%): combine(%,exp):
  evalf(%,14):
  fsolve(%=0, xi=-1..0);
  Digits:=remDigits:
```



-0.16541693191509

Again have patients to let Maple compute it up to 105 Digits and then compare

```
> evalf(TheIntegrand): normal(%): evalf(%):
  TheIntegral:= 'Int(%,xi = -23.0 .. x1)':
  TheI_105:=evalf(%,105);
```

TheI_105 := -0.142296394003137257703313606337041820472803102143364302364135641820820384\0
718663245897302887029700032751 21

```
> `eval(cdfN2_lcc(a1+b1*x1, a2+b2*x1,r)*cdfN(x1), L) - TheIntegral =`;
  evalf(eval(cdfN2_lcc(a1+b1*x1, a2+b2*x1,r)*cdfN(x1), L) -TheI_105,105);
  `computational error` = % -new_result_from_DLL;
```

eval(cdfN2_lcc(a1+b1*x1, a2+b2*x1,r)*cdfN(x1), L) - TheIntegral =

0.142296394003137257703313606337041820472803102143364302364135641820820384071866324589\0
730288702970003275121

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

which shows that within the exactness of qfloat the result is correct.

```
[ >
```