

This algorithm implemented in C++ approximates the value of any triple integral numerically. For example, the approximation of the following triple integral is,

$$\int_{-1}^1 \int_{-\sqrt{1-x^2}}^{\sqrt{1-x^2}} \int_{-\sqrt{1-x^2-y^2}}^{\sqrt{1-x^2-y^2}} e^{(x^x+y^2+z^2)^{\frac{3}{2}}} dz dy dx \approx 7.24177$$

Below is the algorithm.

```
//Approximation of triple integral using Gaussian Quadrature

#include<iostream>
#include<math.h>
#include <assert.h>

using namespace std;

double PI = 3.14159265358979323846264338327950288;

double (*func)(double, double, double); //integrand
double (*_c)(double); //middle lower limit
double (*_d)(double); //middle upper limit
double (*_alpha)(double, double); //inner lower limit
double (*_beta)(double, double); //inner upper limit

double GaussianTripleIntegral(double f, double a, double b, double c, double d,
double alpha, double beta, int m, int n, int p)
{
//need to specify: f=integrand, a=outer lower limit, b=outer upper limit,
//c=middle lower limit, d=middle upper limit, alpha=inner lower limit,
//beta=inner upper limit, m,n=number of iterations
    int steps = n;
    int steps2 = m;
    int steps3 = p;
    double begin = a;
    double end = b;
    double h1, h2, B, __x, __y, __z;
    int i, j, k;
    double answer;

    //roots
    double r_2 [2] = {0.5773502692, -0.5773502692};
    double r_3 [3] = {0.7745966692, 0.0000000000, -0.7745966692};
    double r_4 [4] = {0.8611363116, 0.3399810436, -0.3399810436, -0.8611363116};
    double r_5 [5] = {0.9061798459, 0.5384693101, 0.0000000000, -0.5384693101,
-0.9061798459};
```

```

//coefficients
double c_2 [2] = {1.0000000000, 1.0000000000};
double c_3 [3] = {0.5555555556, 0.8888888889, 0.5555555556};
double c_4 [4] = {0.3478548451, 0.6521451549, 0.6521451549, 0.3478548451};
double c_5 [5] = {0.2369268850, 0.4786286705, 0.5688888889, 0.4786286705,
0.2369268850};

h1 = (end - begin)/2;
h2 = (end + begin)/2;
B = 0;

for (i=0; i<steps2; i++)
{
    double BX = 0;
    __x = h1*r_5[i]+h2;
    double d1 = _d(__x);
    double c1 = _c(__x);
    double k1 = (d1-c1)/2;
    double k2 = (d1+c1)/2;
    for (j=0; j<steps; j++)
    {
        double BY = 0;
        __y = k1*r_5[j]+k2;
        double beta_ = _beta(__x, __y);
        double alpha_ = _alpha(__x, __y);
        double l1 = (beta_ - alpha_)/2;
        double l2 = (beta_ + alpha_)/2;
        for (k=0; k<steps3; k++)
        {
            __z = l1*r_5[k]+l2;
            double Q = func(__x, __y, __z);
            BY = BY + c_5[k]*Q;
        }
        BX = BX + c_5[j]*l1*BY;
    }
    B = B + c_5[i]*k1*BX;
}

answer = h1*B;

return answer;
}

```