DESIGN IDEAS

Simpson's Rule solves double integrals*

Alex Cameron
Adelaide,
Australia

Although one generally uses Simpson's rule to approximate single integrals, you can extend the technique for use in solving double integrals. The C language routine shown in **Listing 1** contains the required algorithm plus and example, demonstrating that you can apply Simpson's rule to certain complex double integrals that would normally require the application of complex numerical techniques. In additional, the technique applies equally well to integrals of a higher order.

A typical double integral is represented in mathematical form using the notation

$$S = \int\int f(x, y)dxdy$$

and Simpson's rule integrates a single integral using the following formula:

$$S_x = \frac{h_x}{3}[f(x_0) + f(x_n) + 4\sum_{i=1}^{n-2/2} f(x_{2i-1}) + 2\sum_{i=1}^{n-2/2} f(x_{2i})]$$

Where h_x represents the increments between the calculated steps:

$$h = \left(upper\ limit\ -\ lower\ limit\right)/\left(number\ of\ steps\right)$$

Modifying to allow the function $f(x, y)$ to be represented, we obtain equation 1,

$$S_x(y_i) = f(x_0, y_j) + f(x_n, y_j) + 4\sum_{i-1}^{\frac{n-2}{2}} f(x_{2i-1}, y_j) + 2\sum_{i=1}^{\frac{n-2}{2}} f(x_{2i}, y_j) \quad (1)$$

In equation 1, n represents the number of steps or strips in the evaluation of the integral in the x domain, and $S_x(y_n)$ represents Simpson's rule applied along the x-axis as a function of y_n on the *y-axis,* for the double integral's inner integral.

Re-applying Simpson's rule to compute the outer integral (*Equation 2*) allows you to express the original equation in algebraic terms. After further simplification it should be apparent that this technique consists of an integration along on axis for each interval on the orthogonal axis, followed by an application of Simpson's rule on the accumulated results along the orthogonal axis to obtain *Equation 3*. (Note, this potentially introduces and accumulated error which the user should be aware of).

$$S = \frac{h_x * h_y}{9} * (S_x(y_0) + S_x(y_n) + 4 * S_x(y_1) + 2 * S_x(y_2)....etc) \quad (2)$$

Or

$$S = \frac{h_x * h_y}{9} [(S_x(y_0) + S_x(y_n) + 4 \sum_{j=1}^{n-2/2} S_x(y_j) + 2 \sum_{j=1}^{n-2/2} S_x(y_j)] \quad (3)$$

The example in Listing 1, implements equations 1 and 2 and is a partial solution for the total radiated power through a hemispherical surface. Two quarter-wavelength monopole antennas, separated by a quarter wavelength and fed by signals that are out of phase by a quarter wavelength, are the source of the radiated power. (The result should be 3.829042)

(First published in Electronic Design News (EDN) 25th June, 1987)

Listing 1.

```
/*
 ============================================================================
 Name        : DounbleIntegralSimpsonsRule.c
 Author      : Alex Cameron
 Version     :
 Copyright   : None
 Description : Simpson integration technique for
               evaluating  double integrals
 ============================================================================
 */

#include "math.h"
float fxy[50][50], fy[50];
float pi;

main() {
    float f();
    float llx, lly, ulx, uly, x, y;
    float hx, hy, ef, of, simpson;
    int nosx, nosy, i, j;

    /*
    ============================================================================
     *
     *  Simpson integration constants.
     *      nos -> number of strips
     *      ul  -> upper limit of integration
     *      ll  -> lower limit of integration
     *      h   -> incremental value per strip
     ============================================================================
     */
    pi = 3.1415926;
    nosx = 30;
    nosy = 40;

    llx = 0.1e-8;
    lly = 0.1e-8;

    ulx = pi;
    uly = 2.0 * pi;
    hx = (ulx - llx) / nosx;
    hy = (uly - lly) / nosy;

    printf("\n\nDouble integration parameters: \n");
    printf(" Step size hx and hy:  %f,%f\n", hx, hy);
    printf(" Number of steps (x,y): %d,%d\n\n", nosx, nosy);
```

Continued next page

```
    /*
     Calculate all the points within the integration domain
     */

    for (j = 0; j <= nosy; j++) {
        y = j * hy + lly;
        for (i = 0; i <= nosx; i++) {
            x = i * hx + llx;
            fxy[i][j] = f(x, y);
        }
    }
    /*
     ============================================================
     Now perform a Simpson integration along
     the x axis and accumulate results using
     the y axis variable as an index
     ============================================================
     */

    for (j = 0; j <= nosy; j++) {
        of = fxy[1][j];
        ef = 0.0;
        for (i = 2; i <= nosx - 2; i += 2) {
            ef += fxy[i][j];
            of += fxy[i + 1][j];
        }
        fy[j] = fxy[0][j] + fxy[nosx][j] + 2.0 * ef + 4 * of;
    }
    /*
     * ============================================================
     * Lastly perform Simpson integration
     *  along the y axis.
     * ============================================================
     */
    of = fy[1];
    ef = 0.0;
    for (j = 2; j <= nosy - 2; j += 2) {
        ef += fy[j];
        of += fy[j + 1];
    }
    simpson = (hx * hy / 9.0) * (fy[0] + fy[nosy] + 2.0 * ef + 4.0
    printf("Result = %f\n\n", simpson);
}
```

Continued next page

```
/*
 * ================================================================
 * Enter the function to be integrated here.
 * ================================================================
 */

float f(x, y)
    float x, y; {
    double zd;

    zd = cos(pi * cos(x) / 2.0) * cos(pi * (1.0 - sin(x) * cos(y)) / 4.0);
    zd = pow(zd, 2.0) / sin(x);
    return (fabs(zd));
}
```

End of listing.