

## Approximating Surfaces Numerically in 3 Dimensions with Differential Equations

**Purpose:** We endeavor to extend Euler iterations into n-space, and develop techniques to approximate both a single n-space point and an entire 3-space surface given a starting value and all first-degree differential equations of the function. Provided with the two partial differential equations, we want to be able to draw a picture of a 3-dimensional surface over any rectangular region using any Euler iteration, with a constant density of calculations over the entire region.

**The Math:** Our first task, then, is to generalize the process of approximating one point a distance of  $\Delta u$  away from a starting point, what we call point-to-point Euler. Any Euler iteration is simply an application of a numerical integration approximation method, used to integrate the first derivative of the function we wish to approximate. In all the examples below, then, we will illustrate using Euler's method, the application of the left-endpoint integral approximation. The other three techniques we investigated in class, and therefore include in our project, are Huen's Method, the Midpoint Method, and Runge-Kutta. Extending Euler's method,

$$y_{i+1} = y_i + f'(x_i, y_i) \Delta x$$

to n-space, we arrive at:

$$f_{i+1} = f_i + (f_x(x_i, y_i, \dots, n_i) u_x + \dots + f_n(x_i, \dots, n_i) u_n) \Delta u$$

The first element of our project, then, is a MathCAD document, easily scalable to any chosen dimension, that can calculate a point-to-point Euler approximation. Approximating from only one point to another, in fact, is a process nearly identical to that used in 2-space. The directional derivative,  $f_u$ , is analogous to  $f'$ , and  $\hat{u}$  is analogous to  $x$ . Hence, in an even more generalized form,

$$f_{i+1} = f_i + (f \cdot \hat{u}) \Delta u$$

$\hat{u}$ , the unit vector in the direction of the derivative (in this case, the direction of the line segment between the starting point and

the point to be estimated, in n-1 dimensions) can be calculated, in general, as

$$\hat{u} = \frac{\Delta x}{\sqrt{\Delta x^2 + \Delta y^2 + \dots}}, \frac{\Delta y}{\sqrt{\Delta x^2 + \Delta y^2 + \dots}}, \dots$$

Our next task, then, is to develop a scheme implementing this point-to-point method to approximately describe, or draw a picture of, a 3-space function over a given rectangular interval.

Implementation: The techniques we develop to approximate a surface should calculate points exclusively at the intersections of a grid in the rectangular coordinate system, to keep the density of points equal over the entire region. A radial calculation would have a high density near the given initial value, and a very low density near the edges of the region. Our techniques, therefore, work on the principle of filling a matrix, representing a regularly subdivided x-y plane, with z-values.

The first technique, as described below, could easily be extended into n-space. Starting with a single point, we approximate the surrounding points in a square (or an n-1-dimensional cube, for n-space), and then continue expanding this square outwards. In this way, each point over a region is given one approximation, with each corner point being used to approximate three points, and every side point approximating only one, as shown below:

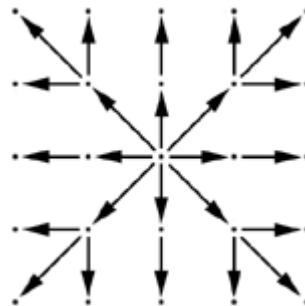


Figure 1. Simple calculation expansion pattern

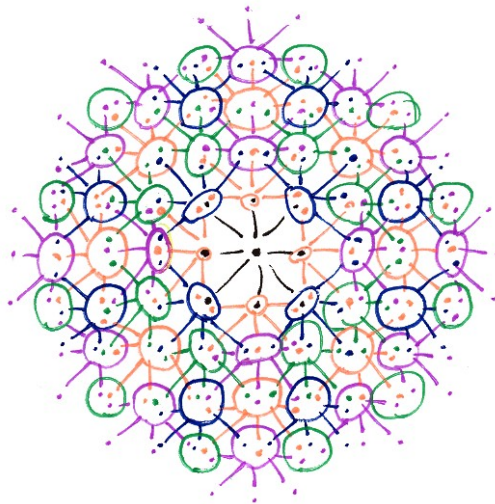


Figure 2. Sophisticated calculation expansion pattern

In any dimension three or higher, a point may be approached from many different directions. This is one reason that the directional derivative exists for higher dimensions, but not in 2-space. Therefore, we believe a better approximation of a surface could be achieved if each point were approximated from many different directions. The calculation pattern we developed (Figure 2), is quite complex and therefore an expansion into n-space is difficult.

The growth of this pattern is approximately circular. Many coordinates are grouped together in "calculation bands," sets of points used to approximate points further out from the central, known starting point, which we call the seed point. After a point has been used to approximate other points further out, its z-value is "protected," i.e. it cannot be re-approximated. The multiple guesses for the z-value at a given (x,y) coordinate provided by applying the Euler technique from different directions are weighted equally in a mean. When a point is used to approximate other points around it, it is this mean that we use for the starting z-value. Hopefully, approximating each z multiple times, along different paths, will give a value closer to the actual for any given Euler technique than implementing the same technique with the simple calculation expansion pattern.

To compare the two techniques in 3-space, we wrote a program that approximates a surface using both of them, and then finds the average absolute error for each point approximated. Below is a table of mean absolute errors:

Function	Minimum	Maximum	Seed	x-divs	y-divs	Soph. error	Simp. error	Method
$z=xy+3y^2-1.5\sin(10xy)$	(-.68,-.5)	(1.2,1.5)	(0,0,0)	50	50	0.436873	1.75026	Euler
$z=x^5+e^{\sin(y+e^y)}-xy^2$	(-1,-.5)	(1,4.5)	(0,0,2.320)	50	60	0.342205	1.37471	R-K
$z=\cos(e^{\sin(x)}x)/(y^2+.1)$	(-2,-3)	(1,0)	(-1,0,1.817)	80	80	8.14146	8.14906	Huen
$z=\sin(x^y)$	(1,1)	(3.5,3.5)	(1,1,0.8415)	80	80	0.0253561	0.62369	Midpt

The functions in the table are of a type that we expect would throw off most Euler approximations. As we see, even under very trying conditions, the more complex technique can do as much as 30 times better than the simple technique. There are conditions, though, under which the advantage is not as evident, and even some when the technique does much worse. Note, however, that there are relatively few divisions. With 200 divisions in each direction, approximating  $z=\sin(x^y)$  from (1,1) to (8,8), the sophisticated technique had a mean error half as large as the simple technique. While this puts a large strain on the computer's resources (40,000 cells to be calculated, between one and four times each), it shows that the technique we have developed is superior to the simpler technique under most conditions.

Both of these techniques have a clear advantage over using an Euler iteration to independently calculate each point. Even though the sophisticated technique recalculates some points as many as four times, no calculations go to waste. By making each point's calculation depend on the calculation of previous points, every calculation may be used to approximate the z-value of a point we care about, i.e. a point at the intersection of the grid, drastically decreasing the number of calculations that must be made.