Classroom Tips and Techniques: Steepest-Ascent Curves

Robert J. Lopez
Emeritus Professor of Mathematics and Maple Fellow
Maplesoft

Introduction

Recently, while flipping through an old calculus text, I spotted an exercise that called for finding the steepest-ascent curve on a surface described by a function $z = f(x, y)$. Ah, thought I, that should be easy enough to do in Maple. It was. Then I thought, what if the gradient field can't be integrated analytically. Can a numeric integration generate the steepest-ascent curve? Of course. And how about if the surface itself is given digitally, not analytically by a formula. Can the steepest-ascent curve still be determined? Again, yes. The details are all given below.

Steepest-Ascent Paths on a Surface: Analytic Case

Figure 1 is a graph of the surface be defined by

$$f := 5 - x^2 - 7y^2 :$$

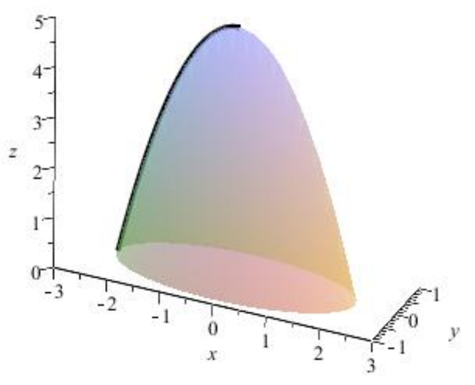
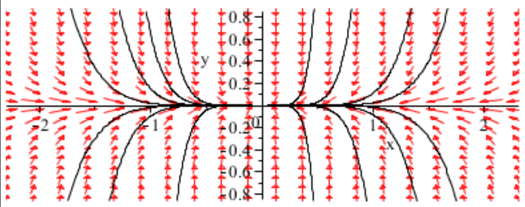


Figure 1 Surface for $z = 5 - x^2 - 7y^2$

Steepest-ascent curves from any point on the ellipse $f(x, y) = 0$ are integral curves of the gradient field ∇f. , itself everywhere orthogonal to the level curves for the function. With $\nabla f = -2\,x\,\mathbf{i} - 14\,y\,\mathbf{j}$, we can use the interactive task template in Table 1 to obtain a graph of the flow lines of the gradient field. After entering the appropriate data, simply click on a point through which an integral curve is to be drawn. Constrained (one-to-one) scaling is imposed with the Context Menu.

Tools≻Tasks≻Browse Calculus - Vector≻Vector Fields≻Integrate Planar Vector Field	
Integrate Planar Vector Field	
Plot Window Insert Defaults: $-2.3 \le x \le 2.3$, $-.85 \le y \le .85$ **Vector Field** Component 1: $-2\ x$ Component 2: $-14\ y$ **Coordinates** System: Cartesian Variables: x, y **Path Parameter** Insert Defaults: $-5 \le t \le 5$	 Erase Graph Clear All

Enter Data	
Cartesian coordinates of point clicked: [-1.5711688, -.41220784]	

Table 1 Interactive construction of flow lines for the gradient field of $f(x, y) = 5 - x^2 - 7y^2$

Figure 2 shows the level curves and the orthogonal trajectories, generated by the commands hidden in the Table holding the graph.

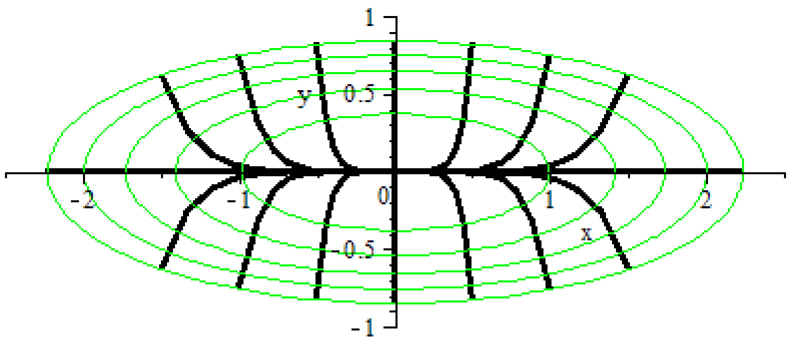


Figure 2 Level curves (green) for $f(x, y) = 5 - x^2 - 7y^2$ and flow lines (black) for ∇f

The integral curves of the gradient field can be found by solving the system

$$x'(p) = f_x(x(p), y(p))\, y'(p) = f_y(x(p), y(p))$$

in which case the steepest-ascent curves are given parametrically by $(x(p), y(p), f(x(p), y(p)))$, where p is the parameter along the curve. Alternatively, the gradient field can be integrated by solving the differential equation

$$\frac{d}{dx}(f(x,y(x)) = c) \Rightarrow f_x(x,y(x)) + f_y(x,y(x))\,y'(x) = 0$$

in which case the steepest-ascent curves are given by $(x, y(x), f(x, y(x)))$.

For the surface depicted in Figure 1, the latter differential equation is $y' = 7\,y/x$, with solution $y(x) = A\,x^7$. The apex of that surface is a critical point for the gradient field, that is, $\nabla f = \mathbf{0}$ at the top of the hill. Hence, it is necessary to find curves of steepest *ascent*, and for this, we select initial points along the level curve $z = 0$, that is, along the ellipse $x^2 + 7\,y^2 = 5$. Figure 3 shows the surface, along with a collection of steepest-ascent curves emanating from points along the base ellipse. The details are hidden behind the table containing the figure.

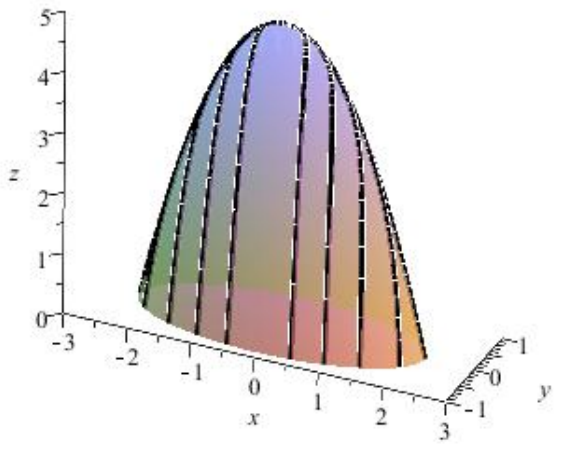


Figure 3 Steepest-ascent curves on the surface $z = 5 - x^2 - 7\,y^2$

Steepest-Ascent Paths on a Surface: Numeric

Integration

The function used for Figures 1 and 3 is sufficiently simple that the gradient field can be integrated analytically. Should this not be the case, numeric integration could be used. Code for this is illustrated in Table 2. The lists *X* and *Y* defining initial points on the base ellipse are defined behind Figure 3.

```
sys := [x'(t) =-2 x(t), y'(t) =-14 y(t), z'(t) =-2 x(t) x'(t)-14 y(t) y
      '(t), x(0) = x0, y(0) = y0, z(0) = z0] :
vars := [x(t), y(t), z(t)] :
R := dsolve(sys, vars, numeric, parameters = [x0, y0, z0]) :
 for k from 1 to 10 do
R(parameters = evalf([X[k],-Y[k], 0])) :
pp||k := plots[odeplot](R, [x(t), y(t), z(t)], t = 0 ..5, color = black,
      thickness = 3);
 end do:
```

Table 2 Maple code for generating steepest-ascent curves numerically

The first two differential equations in the list *sys* integrate the planar gradient field, while the third is

$$\frac{d}{dt}z(t) = \frac{d}{dt}f(x(t), y(t)) = -2\,x\,x' - 14\,y\,y'$$

The use of the *parameters* option in **dsolve**/**numeric** simplifies changing the initial conditions for each ascent curve. Before the resulting numerically determined curve can be graphed, the intervening line that defines a set of parameters must be executed. Surprisingly, this invocation must be in terms of floats - hence, the **evalf** command.

```
plots[display]([p1, pp||(1 ..10)], axes
      = frame, scaling = constrained)
```

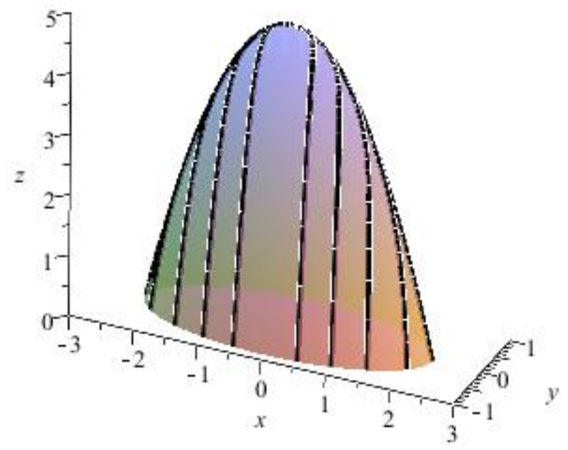


Figure 4 Numerically generated steepest-ascent curves

Steepest-Ascent Curves on a Digitally-Defined Surface

In this final phase of our investigation, we assume that the surface is given, not analytically by a formula, but digitally, by a collection of data coordinates. Because we will interpolate the data points to generate a smooth surface, we need to impose some structure on the grid supporting the heights of the surface. The grid does not have to be equispaced in the supporting *xy*-plane, but it has to be *regular*. By regular, we mean that heights are known at each point (x_i, y_j) , where x_i is a member of an ordered list of *x*-coordinates in the data set, and y_j is a member of the corresponding list of *y*-coordinates. In effect, the surface is known at each point of the direct product of the lists of *x*- and *y*-coordinates.

Thinking of the function used earlier, namely

$$F(x, y) = 5 - x^2 - 7\,y^2$$

we write the coordinate lists

$$X := evalf\left(\left[-\sqrt{5}, -2, -3/2, -1, -1/2, 1/2, 1, 3/2, 2, \sqrt{5}\right]\right):$$
$$Y := evalf\left(\left[-\sqrt{5/7}, -.1, 0, .1, \sqrt{5/7}\right]\right):$$

and generate a matrix of corresponding function values:

$$M := Matrix\left(10, 5, (a, b) \rightarrow F\left(X_a, Y_b\right)\right)$$

```
[[-5.000000001, -0.069999998, 2. 10^-9, -0.069999998, -5.000000001
    ],
    [-4.000000003, 0.93, 1., 0.93, -4.000000003],
    [-2.250000003, 2.680000000, 2.750000000, 2.680000000,
    -2.250000003],
    [-1.000000003, 3.93, 4., 3.93, -1.000000003],
    [-0.250000003, 4.680000000, 4.750000000, 4.680000000,
    -0.250000003],
    [-0.250000003, 4.680000000, 4.750000000, 4.680000000,
    -0.250000003],
    [-1.000000003, 3.93, 4., 3.93, -1.000000003],
    [-2.250000003, 2.680000000, 2.750000000, 2.680000000,
    -2.250000003],
    [-4.000000003, 0.93, 1., 0.93, -4.000000003],
    [-5.000000001, -0.069999998, 2. 10^-9, -0.069999998,
    -5.000000001]]
```

Next, we define $G(x, y)$, a function that interpolates the given data.

$$G := (a, b) \rightarrow CurveFitting{:}\text{-}ArrayInterpolation([X, Y], M, [[a], [b]], method = spline)[1, 1]:$$

The first argument to **ArrayInterpolation** is a list of the lists of the independent coordinates. The second argument is the matrix of known heights. The third argument defines the "target," that is, the points at which "new" function values are to be calculated. Ordinarily, this would be an Array with the coordinates of many points. Here, it consists of the coordinates of a single point, (a, b) . Thus, $G(u, v)$ is the interpolated height at the single point (u, v) .

The final argument to **ArrayInterpolation** is the interpolating method. Since this is merely an illustrative example, we've picked "spline" for no particular reason. Figure 5 displays the surface generated by the interpolating function $G(x, y)$.

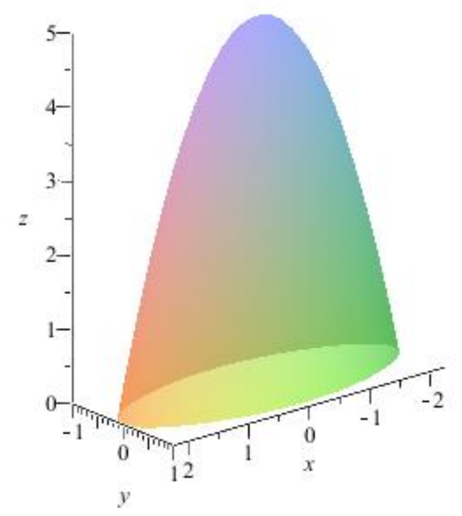


Figure 5 Surface generated by the interpolating function $G(x, y)$

Now we must devise a "steepest-ascent algorithm" that works strictly numerically, on just data. We don't have a function whose gradient field we can integrate. We can use only the given data points and the interpolating function G.

Here is our thinking. If we are at a point $P = (\alpha, \beta)$ where the height is $G(\alpha, \beta)$, we want to move to the "nearest highest point." Of course, for a continuous surface, this is meaningless. So instead, we ask for the nearest highest point in a neighborhood of *P*, and define that neighborhood as a small circle around *P*. But how do we find the highest point on $G(x, y)$ subject to a constraint such as $(x-\alpha)^2 + (y-\beta)^2 = (1/10)^2$? We use the **Search** command from Dr. Sergey Moiseev's *DirectSearch* package. A sequence of such optimizations generates the steepest-ascent curve as a set of points in $\mathbb{R}^3$.

For example, we start a steepest-ascent curve at the point on the base contour where $x = 1.5$, and compute the corresponding *y*-coordinate to be

$$y_0 := fsolve('G(1.5, y)', y, y = 0..1)$$

```
0.5909254764
```

and hence, the initial point on the steepest-ascent curve is

$$P_1 := [1.5, y_0, 0]:$$

Additional points are then obtained as per the calculations in Table 3.

```
for k from 1 to 20 do
temp := DirectSearch:-Search(G, variables = [x, y],
      [(x - P_k[1])^2 + (y - P_k[2])^2 = 0.01],
      maximize = true);
P_{k+1} := [convert(temp[2], list)[ ], temp[1]];
end do:
```

Table 3 A sequence of steepest-ascent points computed as constrained maxima

Figure 6 contains a graph of the connected points, superimposed on the surface in Figure 5.

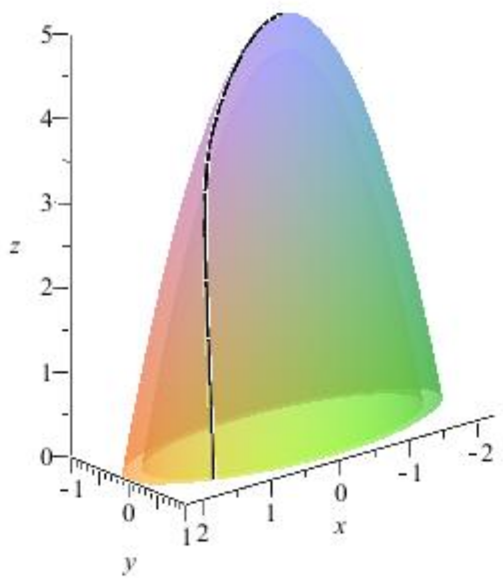


Figure 6 Discretely computed steepest-ascent curve on the surface in Figure 5

Of course, the calculations in Table 3 would have to be repeated for a number of different initial points if we wanted to reproduce the equivalent of Figure 4. Other considerations include the "stepsize," that is, the radius of the constraint circle defining the neighborhood of each point on the surface.