A tracking algorithm by E. Danicki

Abstract

A simple algorithm is presented that can find applications in tracing of zero-level lines of two dimensional functions or in controlling of a mobile robot movement along the given path. *John, this is a funny algorithm, due to its simplicity and usfulness, that may interest your students: making some changes - may result in nice figures: see last figures.*

Tracing of a curve of given level (level zero, for instance) of a complicated function $f(x, y)$ which computation is excessively time consuming, requires clever choice of evaluation points $z_i = x_i + jy_i$ (complex plane is convenient here). Similar problem arises with controlling of a mobile robot that should follow the prescribed path.

The idea is the following (Fig. 1). Using a straight move over a chosen constant distance d in certain direction, the direction of next step is made at an angle $\theta = 2\pi/3$ clockwise with respect to the previous step. This clockwise rotation is maintained in subsequent steps until the sign of the function changes, that is until the zero-level line is crossed; in this case the rotation changes its direction to opposite (clockwise to counter-clockwise or vice versa). In other words: 'keep turning right until you cross the line (zero-level of f), then change to left...' Naturally, the first step should be chosen such that the line crossing occurs in initial two steps, otherwise we return to the starting point. On a complex plane, the subsequent step direction corresponds to multiplication of complex-valued step d by $c = \exp(-j\theta)$; changing the value of c to complex conjugate means changing the rotation (clockwise, counter-clockwise).

An example code in MATLAB is the following:

```
d=.09;n=200;c=exp(2i*pi/3);z=[];zn=d/3i;fn=-1;
for i=1:n;zo=zn;fo=fn;d=d*c;zn=zo+d;
fn=imag(zn)-sin(2*pi*real(zn));
z=[z;real(zn) imag(zn)];
if fo*fn<0;c=conj(c);end;
end;plot(z(:,1),z(:,2),'k-');
```

In this example, f is defined in 3rd line as 'fn,' 'n' is set to limit the program run-time (it can be run as presented in a MATLAB command window). Other initial values are set in 1st line. Symbols ending with 'n' gets new values, while ending by 'o' are for storing 'old' values for use in next step: the condition 'fn*fo<0' indicates the zero line-crossing. The plotted result is presented in Fig. 1a).

An interesting feature of the above algorithm is that it carries us back towards the starting point along the line that terminates blindly. Such termination takes place if we define $f = 0$ outside certain area of x, y. (Also note that in the above algorithm, any integer fraction of 2π can be chosen for θ.)

The code can be improved in order to avoid occasional redundant evaluation of f at the same point. Let the line-crossing occurred in previous step between points 0 and 1 (see inset to Fig. 1b). If the line crossing does not occur in the current step from point 1 to point 2, it will certainly occur in the subsequent step that would be from 2 to 0 (as governed by the rotation direction; it would be changed after this step). The position of further point 3 is also known, it is 3, with rotation direction changed again (to the same as at point 2 - counter-clockwise in the figure). This point 3 can be reached without making steps 2-0-3, simple by retarding the step's rotation by $\theta/2$. This is performed by division of d by $\sqrt{c}$ in the line 7 of the code below. This way, the redundant evaluation of f at point 0 is avoided.

The corrected algorithm reads (note another f defined by 'fn')

```
d=.09;n=150;c=exp(-pi/1.5i);z=[];p=[];
zn=d/2i;fn=-1;fr=nan;zr=fr; for i=1:n;
a=(fn*zr-fr*zn)/(fn-fr);p=[p;a];
z=[z;real(zn) imag(zn)];zo=zn;fo=fn;d=d*c;
zn=zo+d;fn=imag(zn)-sin(pi*real(zn)^2);
if fo*fn<0;c=conj(c);
zr=zn-d;fr=fo;else;d=d/sqrt(c);end;end;
plot(z(:,1),z(:,2),'k-');hold on;
plot(real(p),imag(p),'k.');hold off;
```

The above code is appended by a Newton formula to evaluate the estimated zeros of f between the evaluated points (although the function is not analytical in the example presented, the Newton rule works well). To this end we need to store the values obtained just before the last line-crossing occurred ('zr' and 'fr'). Fig. 1b) presents the resulting steps and the estimated points (dots). This corrected algorithm can however, go into infinite loop in rare cases of blindly terminated lines.

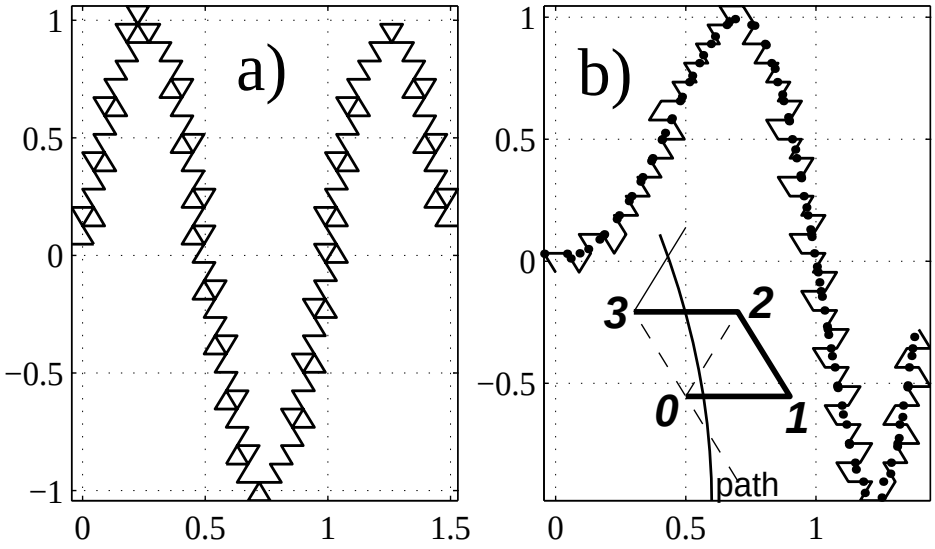

Fig. 1. Resulting path produced a) by an original algorithm, and b) by a corrected one; inset explains the correction.

The modification (note use of 'rand'-om numbers making fig. unique)

```
d=.02;n=2200;c=exp(2i*pi/6.1);z=[];zn=d/3i;fn=-1; for
i=1:n;zo=zn;fo=fn;d=d*c;zn=zo+d;
fn=imag(zn)-sin(2*pi*real(zn));z=[z;real(zn) imag(zn)]; if
fo*fn<0;c=conj(c*exp(1.7i*rand(1)));end;
end;plot(z(:,1),z(:,2),'k-')
```