Visualizing Complex Functions with the Cardano3 Application

Murray Eisenberg
David J. M. Park, Jr.

Visualization is an invaluable companion to symbolic computation in understanding the complex plane and complex-valued functions of a complex variable. The Cardano3 application, an add-on to *Mathematica,* provides a rich set of tools for assisting such visualization. This paper demonstrates some capabilities of the application, especially those relevant to an introduction to complex analysis, and it indicates some teaching and learning issues that arise from using the application. Included are examples of how complex functions map objects in the complex plane and on the Riemann sphere, and of how complex functions behave near singularities and at branch points.

■ Introduction

Visualization and symbolic computation are both essential to understanding how functions behave. Visualizing the behavior of a real-valued function of a real variable is often easy, because the function's graph may be plotted in the plane—a space with just two real dimensions. Visualizing the behavior of a complex-valued function of a complex variable, by contrast, is more difficult, because the graph lives in a space with four real dimensions. Whereas *Mathematica* is replete with resources for symbolic computation with complex functions, out-of-the-box it provides only a meager set of tools for visualizing such functions. To do much more than what is provided by the standard add-on package `Graphics‘ComplexMap‘`—even to plot the image of a simple curve in the complex plane—requires the effort of constructing the image set and its graphical representation and then of combining graphics objects. The Cardano3 application [1], written by the second author, is an add-on to *Mathematica* that provides a rich set of tools for complex function plotting, along with some utilities for complex symbolic computation. Cardano3 includes full documentation, with individual HelpBrowser pages and examples for each command.

In a *Mathematica*-enriched introductory complex analysis course, the first author prepared demonstration and exercise notebooks for Cardano3 (see [2]) and experimented with students' using it; experience there led to some enhancement of the application. The course used the textbook by Mathews and Howell [3] which is fairly traditional in approach, although it does go a bit further than some introductions in emphasizing mapping properties of complex functions. Cardano3 would be especially suitable for use with a less traditional, more visually oriented text, such as the one by Needham [4], which inspired and helped guide development of the application.

In this paper we demonstrate some capabilities of the Cardano3 packages `ComplexRoutines` and `ComplexGraphics`, especially those relevant to an introduction to complex analysis. The examples include a geometric problem solved by complex means, representations of complex functions as mappings, depiction of singularities, and analysis of branch points. Typically the examples include embellishments that might be inappropriate for somebody to program who is a novice at both *Mathematica* and complex analysis, but these embellishments illustrate some of the advanced functionality of the packages.

Various graphical representations of complex functions are often needed to emphasize their various features. `MappingGraphic` plots and `TwoPanelPlot`s can be used to emphasize mapping properties. Riemann sphere plots can be used to better depict behavior near the point at infinity, or to emphasize periodic behavior. `CodedDensity` (domain coloring) plots give an overall representation of a function in the plane that emphasizes singular points and branch lines. `ComplexVector` plots emphasize how a function varies on a grid or along a path in the complex plane. Animation is especially effective in making various properties come alive. Cardano3 can give an overall picture or zoom in on particular portions of the domain. Multifunctions can be used with several of the plot types. All these techniques increase our ability to understand complex function behavior.

Top–level Cardano3 functions ultimately call upon routines in the DrawGraphics add–on application [5], which is discussed in Park [6]. To reproduce all the results in this notebook, you will therefore need not just Cardano3, but DrawGraphics as well; you will also need Ersek's RootSearch package [7]. Cardano3 and DrawGraphics are at present compatible with *Mathematica* versions 4.2-5.2.

■ Initialization

To begin, we initialize the two packages that constitute the Cardano3 application.

```
Needs["Cardano3‘ComplexGraphics‘"]
Needs["Cardano3‘ComplexRoutines‘"]
```

The DrawGraphics packages are automatically loaded by `ComplexGraphics`.

■ Geometry in the complex plane

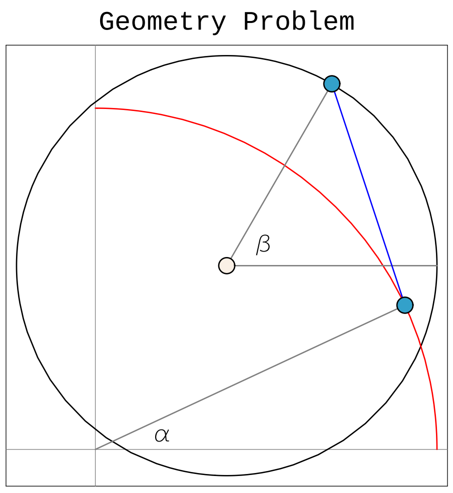
Cardano3 contains a complete set of graphics primitives that directly use complex numbers for points. Geometrical diagrams are an important part of mathematical discussions, yet students find it difficult to draw such diagrams. The power of complex algebra is that many such diagrams are more easily constructed and drawn in the complex plane. Such diagrams are not only an excellent introduction to Cardano3 and complex algebra but a valuable technique for mathematical work.

The problem considered here is from a posting on MathGroup [8] (with Greek letters used for the angle variables of the original):

> "I am trying to find the angle β corresponding to the points on the circumference of the circle (5 + 8 Cos[β], 7 + 8 Sin[β]) that are a distance of 7 units from a point on the circumference of the circle (13 Cos[α], 13 Sin[α]) for angles α in the first quadrant."

The following creates a diagram for the problem.

```
With[{c = 5 + 7 ⅈ, r1 = 8, r2 = 13, β1 = 60 °, α1 = 25 °},
  ComplexGraphics[
   {ComplexCurve[c + r1 ⅇ^(ⅈ β), {β, 0, 2 π}],
    Red, ComplexCurve[r2 ⅇ^(ⅈ α), {α, 0, π / 2}],
    Gray, ComplexLine[{c, c + r1}], ComplexLine[{c, c + r1 ⅇ^(ⅈ β1)}],
    ComplexLine[{0, r2 ⅇ^(ⅈ α1)}],
    Blue, ComplexLine[{r2 ⅇ^(ⅈ α1), c + r1 ⅇ^(ⅈ β1)}],
    Black, ComplexText["β", c + r1 / 5 ⅇ^(ⅈ β1/2)],
    ComplexText["α", r2 / 5 ⅇ^(ⅈ α1/2)],
    ComplexCirclePoint[c, 3, Black, Linen],
    ComplexCirclePoint[#, 3, Black, Peacock] & /@
     {c + r1 ⅇ^(ⅈ β1), r2 ⅇ^(ⅈ α1)}},
   PlotLabel → "Geometry Problem", ImageSize → 2.5 * 72] ];
```



Given a value of angle α, find angles β such that the blue line joining the two circled points has a length 7. To solve this problem, we simplify things by writing the *complex* equation `Abs[(5 + 7 ⅈ) + 8 ⅇ^(ⅈ β) − 13 ⅇ^(ⅈ α)] == 7` for β with α as parameter.

```
βeqn[α_] = ComplexExpand[Abs[(5 + 7 ⅈ) + 8 ⅇ^(ⅈ β) − 13 ⅇ^(ⅈ α)] == 7]
```

$$\sqrt{(5 - 13\,\mathrm{Cos}[\alpha] + 8\,\mathrm{Cos}[\beta])^2 + (7 - 13\,\mathrm{Sin}[\alpha] + 8\,\mathrm{Sin}[\beta])^2} == 7$$

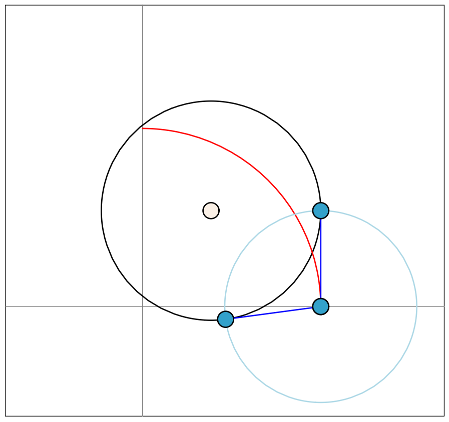
An additional geometrical diagram (not shown) indicates that the equation for β always has two roots. The roots are most easily found by using Ersek's RootSearch package [7].

```
Needs["Ersek`RootSearch`"]
```

For example, evaluating `RootSearch[βeqn[π/4],{β,0,2π}]` gives the result `{{β→1.53612},{β→5.71073}}`. Then the following animation shows the solutions as α is varied from `0` to π / 2.

```
frame2[α_] := Module[ {c = 5 + 7 ⅈ, r1 = 8, r2 = 13, β1, β2, θ},
   {β1, β2} = Flatten[β /. RootSearch[βeqn[α], {β, 0, 2 π}]];
   ComplexGraphics[
    {ComplexCurve[c + r1 ⅇ^(ⅈ β) , {β, 0, 2 π}],
     Red, ComplexCurve[r2 ⅇ^(ⅈ θ) , {θ, 0, π / 2}],
     Blue, ComplexLine[{r2 ⅇ^(ⅈ α) , c + r1 ⅇ^(ⅈ β1) }],
     ComplexLine[{r2 ⅇ^(ⅈ α) , c + r1 ⅇ^(ⅈ β2) }], LightBlue,
     ComplexCurve[r2 ⅇ^(ⅈ α) + 7 ⅇ^(ⅈ β) , {β, 0, 2 π}],
     Black, ComplexCirclePoint[c, 3, Black, Linen],
     ComplexCirclePoint[#, 3, Black, Peacock] & /@
      {r2 ⅇ^(ⅈ α) , c + r1 ⅇ^(ⅈ β1) , c + r1 ⅇ^(ⅈ β2) , r2 ⅇ^(ⅈ α) }},
    PlotLabel → "Geometry Problem Solved",
    PlotRange → {{-10, 22}, {-8, 22}}, ImageSize → 2.5 * 72] ];
Animate[frame2[ϕ], {ϕ, 0, π / 2, π / 2 / 24}]
SelectionMove[EvaluationNotebook[], All, GeneratedCell]
FrontEndTokenExecute["OpenCloseGroup"]; Pause[0.5];
FrontEndExecute[
 {FrontEnd`SelectionAnimate[200, AnimationDisplayTime → 0.1,
   AnimationDirection → ForwardBackward]}]
```

Geometry Problem Solved



Evidently it takes only a trivial amount of extra work to go from a static diagram to an animation. Producing a calculated animation of a solution is an excellent check on the method. The final statements in the code above close up and start the animation; they were pasted in by using a button on the DrawGraphics palette.

■ Complex functions as mappings

The most direct way to represent a function $f : \mathbb{C} \to \mathbb{C}$ as a mapping is to display side–by–side the domain and codomain planes (or their Riemann sphere compactifications), to place in the domain some objects of interest, and to display the corresponding images of these objects under f in the codomain. Usually we locate the objects with reference to some grid in the domain, and then display their images with reference to the image of that grid.

To represent such a function as a mapping in `Cardano3`, we use a `TwoPanelPlot`. The form of the call is as follows.

```
TwoPanelPlot[
  { funcs1, plottype1, scale1, plotargs1, extraprimitives1:{},
     plotoptions1:{}, paneloptions1:{} },
  { funcs2, plottype2, scale2, plotargs2, extraprimitives2:{},
     plotoptions2:{}, paneloptions2:{} },
  sizeopt]
```

In our immediate use, the lists `funcs1` and `func2` will consist of the identity function and the function f, respectively.

Our first example is an affine linear function, of the form $f(z) = a z + b$.

```
f[z_] := (2 + i) z - 3 i
```

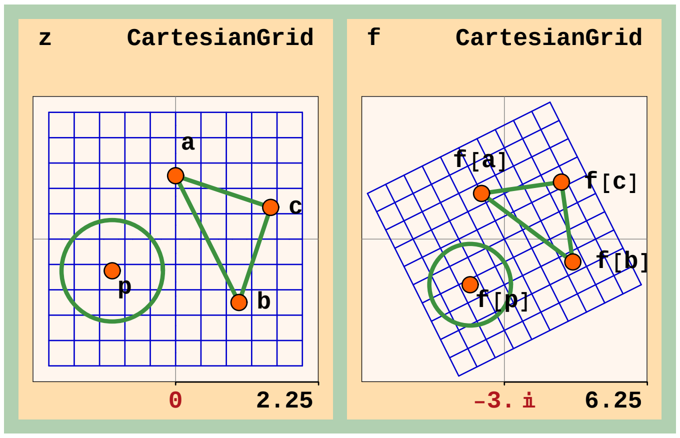
This example is a good place to start because it is so easy to calculate directly (even with paper and pencil!) the images of lines and circles, and hence to understand the very concept of a function $f : \mathbb{C} \to \mathbb{C}$ as a mapping—a concept that many beginners at first find difficult. The following `TwoPanelPlot` shows how a triangle and a circle upon a cartesian grid map under f.

First we specify the objects that we want to draw.

```
grid = {{{MediumBlue}, {-2, 2, 11}, {-2, 2, 11}}};
a = i; b = 1 - i; c = 3 / 2 + i / 2; p = -1 - i / 2;
pts = {ComplexCirclePoint[#, 3, Black, CadmiumOrange] & /@
     {a, b, c, p}};
triangle = {CobaltGreen, Thickness[0.015],
   ComplexLine[{a, b, c, a}]};
circle = {CobaltGreen, Thickness[0.015],
   ComplexCircle[-1 - i / 2, 4 / 5]};
zLabels = {Black, ComplexText["a", a, {-1, -2}],
   ComplexText["b", b, {-2, 0}], ComplexText["c", c, {-2, 0}],
   ComplexText["p", p, {-1, 1}]};
wLabels = {Black, ComplexText["f[a]", a, {0, -2}],
   ComplexText["f[b]", b, {-1.5, 0}], ComplexText["f[c]",
    c, {-1.5, 0}], ComplexText["f[p]", p, {-1, 1}]};
```

Then we insert them into each side of a `TwoPanelPlot`.

```
TwoPanelPlot[
  {{# &}, CartesianGrid, {0, 2.25},
   {grid}, {triangle, circle, pts, zLabels}},
  {{f}, CartesianGrid, {0, 6.25}, {grid},
   {triangle, circle, pts, wLabels}},
  ImageSize → 3.5 * 72];
```



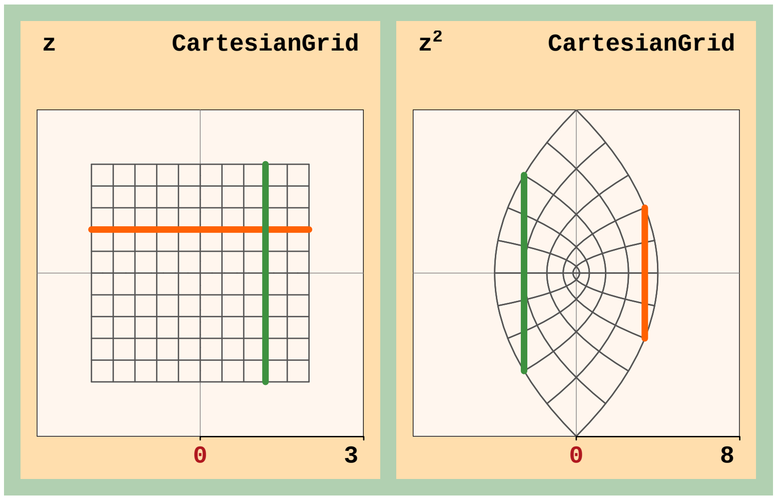
In panel plots the red label at the bottom gives the center of the plot as a complex number; the black label gives the half–width of the plot in each direction.

The plot suggests that f maps lines to lines, and circles to circles; it further suggests that rotation, stretching, and translation are involved. To verify analytically that this is so, one need only write f as the composite of a rotation around the origin, a dilation, and a translation. The HelpBrowser documentation for Cardano3 illustrates the actions of rotations, dilations, and translations by means of animations in the complex plane and their lifts to the Riemann sphere.

For a novice, such a first example should surely be simpler; for example, it might involve mapping just a single line segment and forego labeling the points. And producing such a graphic may either precede the analysis, to suggest what the analysis will reveal, or else follow the analysis, to confirm visually what the analysis predicts.

The simplest nonlinear polynomial is the squaring function. Here is a naive student's attempt to show how it maps.

```
grid = {{{DimGray}, {-2, 2, 11}, {-2, 2, 11}}};
horiz = {CadmiumOrange, Thickness[0.02],
   ComplexLine[{-2 + 4 ⅈ / 5, 2 + 4 ⅈ / 5}]};
vert = {CobaltGreen, Thickness[0.02],
   ComplexLine[{6 / 5 - 2 ⅈ, 6 / 5 + 2 ⅈ}]};

TwoPanelPlot[
  {{# &}, CartesianGrid, {0, 3}, {grid}, {horiz, vert}},
  {{#^2 &}, CartesianGrid, {0, 8}, {grid}, {horiz, vert}},
  ImageSize → 4 * 72];
```

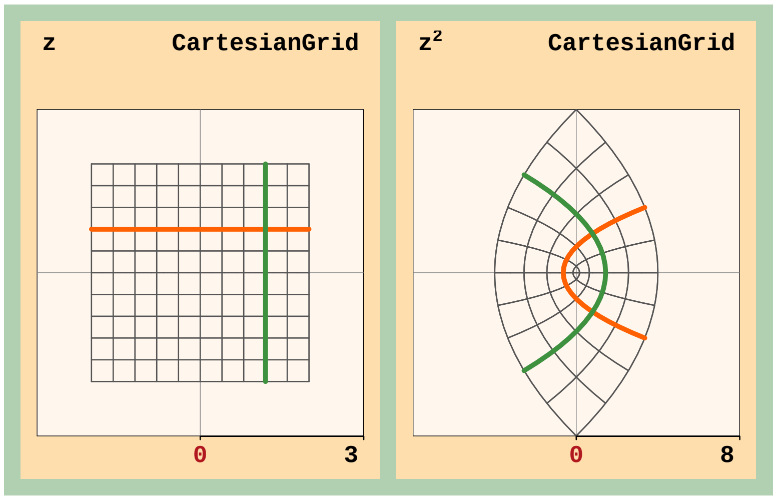


The parabolic arcs shown as the images of the horizontal and vertical segments of the grid are correct, but the images of the highlighted horizontal and vertical segments are *wrong*. The student wrongly generalized from the case of affine linear mapings and had unreasonable expectations as to what the application would do.

What went wrong is this. When forming the image of the grid itself, Cardano3 applies the target function (here $z \mapsto z^2$) to points along the lines of the grid and then connects the resulting image points in the codomain. For a primitive graphics object such as `ComplexLine`, however, it merely applies the function to distinguished points of the object—for `ComplexLine`, its vertices—and then forms the corresponding object in the codomain based upon the images of the distinguished points.

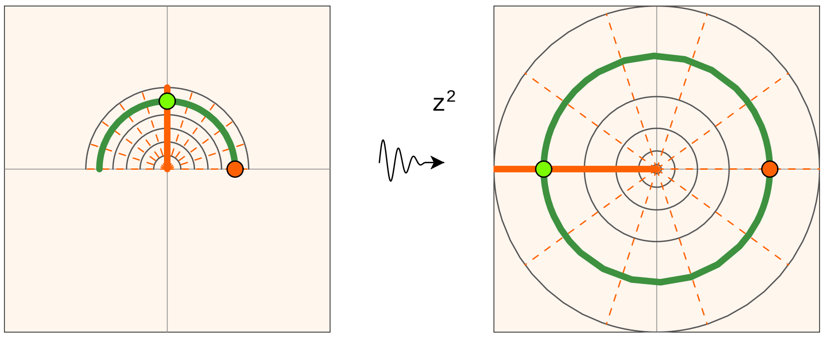
It was the encounter with this misunderstanding by students that led to the new Cardano3 primitive `ComplexCurve` to represent a curve in the complex plane parameterized by a real variable. The Cardano3 routines find the image of such a curve in the same way as for the lines in a grid—by sampling points along the curve, calculating their images, and then connecting the image points. The modified curve below, employing `ComplexCurve` objects, now correctly represents the mapping.

```
grid = {{{DimGray}, {-2, 2, 11}, {-2, 2, 11}}};
horizontal = {CadmiumOrange, Thickness[0.015],
   ComplexCurve[(1 - t) (-2 + 4 ⅈ / 5) + t (2 + 4 ⅈ / 5), {t, 0, 1}]};
vertical = {CobaltGreen, Thickness[0.015],
   ComplexCurve[(1 - s) (6 / 5 - 2 ⅈ) + s (6 / 5 + 2 ⅈ), {s, 0, 1}]};

TwoPanelPlot[
  {{# &}, CartesianGrid,
   {0, 3}, {grid}, {horizontal, vertical}},
  {{#^2 &}, CartesianGrid, {0, 8}, {grid},
   {horizontal, vertical}}, ImageSize → 4 * 72];
```



Of course a rectangular grid is hardly the best way to understand how the squaring function maps. Much better is a polar grid, as shown below. Instead of a `TwoPanelPlot`, this time we use a `MappingGraphic` plot frame.

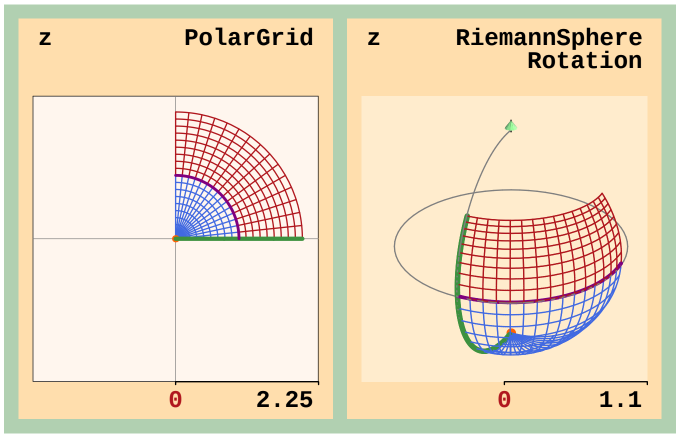
```
grids = {{{DimGray}, {0, 2, 7}, {0, π, 0}},
   {{Dashing[{0.02, 0.025}], CadmiumOrange},
    {0 , 2, 0}, {0, π, 11}}};
ray = {CadmiumOrange, Thickness[0.02], ComplexLine[{0, 2 ⅈ}]};
arc = {CobaltGreen, Thickness[0.02],
   ComplexCurve[5 Exp[ⅈ θ] / 3, {θ, 0, π}]};
pts = {ComplexCirclePoint[5 / 3, 3, Black, CadmiumOrange],
   ComplexCirclePoint[5 ⅈ / 3, 3, Black, LawnGreen]};
domainPlot = PolarGrid[{# &}, {0, 4}, {grids}, {ray, arc, pts}];
imagePlot =
  PolarGrid[{#^2 &}, {0, 4, False}, {grids}, {ray, arc, pts}];

MappingGraphic[{{0, 0.5, 0}, {0, 0}}][{domainPlot, imagePlot},
  {NeedhamMappingSymbol[1.15 + 0.52 ⅈ, 0.15],
   ComplexText[StyleForm[z^2 , FontSize → 9], 1.35 + 0.7 ⅈ]},
  ImageSize → 4.25 * 72];
```



Mapping properties of some complex functions may be nicely visualized by considering them as mappings of the Riemann sphere Ω. Let $\hat{\mathbb{C}} = \mathbb{C} \bigcup \{\infty\}$ be the extended complex plane, and let $p : \Omega \to \hat{\mathbb{C}}$ be the stereographic projection onto the equatorial plane, with the north pole going to the point at infinity. To visualize the map $p^{-1} : \mathbb{C} \to \Omega$, consider the part A of the closed disk $\overline{D}_2(0)$ in the closed first quadrant. Through a polar grid, the following TwoPanelAnimation depicts both A and its image $p^{-1}(A)$; parts of the boundary of A are highlighted.

```
grid = {{{RoyalBlue}, {0., 1, 10}, {0., π / 2, 16}},
   {{IndianRed}, {1, 2., 10}, {0., π / 2, 16}}};
gridmap = FineGrainLines[0.02, 8][
   First[PolarGrid[{#1 &}, {0, 8}, {grid}]]];
origin = {CadmiumOrange, PointSize[0.025], ComplexPoint[0]};
xAxis = {CobaltGreen, Thickness[0.015],
   FineGrainLines[0.02, 8][ComplexLine[{0, 2}]]};
quarterCircle = {Purple, Thickness[0.01],
   ComplexCurve[ⅇ^ⅈ θ, {θ, 0, π / 2}]};
```

```
TwoPanelAnimation[
   {{#1 &}, PolarGrid, static,
    {0, 2.25}, {grid}, {origin, xAxis, quarterCircle}},
   {{#1 &}, RiemannSphere, rotation, {0, 1.1},
    {ColoredRiemannSphere[{None}, {}, {}], {gridmap}},
    {{}, {StereographicMap[{origin, xAxis, quarterCircle}]}},
    {ViewPoint → {2.486, 1.127, 1.5},
     PlotRegion → {{-0.2, 1.15}, {-0.25, 1.1}},
     Background → BlanchedAlmond}},
   ImageSize → 3.5 * 72][18];
```

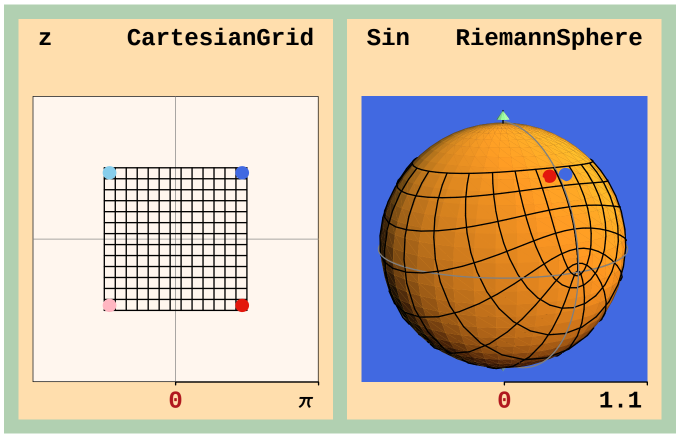


In the first author’s course, stereographic projection had been described geometrically. When this graphic was demonstrated, students were curious to learn how Cardano3 implemented the projection. The explicit formula appears in the ComplexGraphics package code, but students could not readily ferret that out. So using Cardano3 motivated discovering the formula and thereby presented an opportunity for students to exercise 3–dimensional vector methods.

Our final example of visualizing mappings is the complex sine. The following shows how Sin maps a square grid in the z–plane, with the image lifted to the Riemann sphere.

```
grid = {{{Black}, {-π / 2, π / 2, 14}, {-π / 2, π / 2, 14}}};
points = MapThread[{#1, ComplexPoint[#2]} &,
   {{RoyalBlue, SkyBlue, LightPink, DeepCadmiumRed},
    0.93 π / 2 {1 + ⅈ, -1 + ⅈ, -1 - ⅈ, 1 - ⅈ}}];
primitives = {AbsolutePointSize[5], points,
   First[CartesianGrid[{# &}, {0, π / 2}, {grid}]]};
```

```
TwoPanelPlot[{{# &}, CartesianGrid, {0, π},
   {grid}, {AbsolutePointSize[5], points}}, {{Sin},
   RiemannSphere, {0, 1.1},
   {ColoredRiemannSphere[], primitives}, {},
   {PlotRegion → {{-0.15, 1.2}, {-0.25, 1}}}},
  ImageSize → 3.5 * 72];
```



The grid is embellished with four colored points. Sine is periodic in the imaginary direction, and nothing better illustrates this than the way the image wraps around the Riemann sphere bringing the blue and red colored points together.

■ Singularities

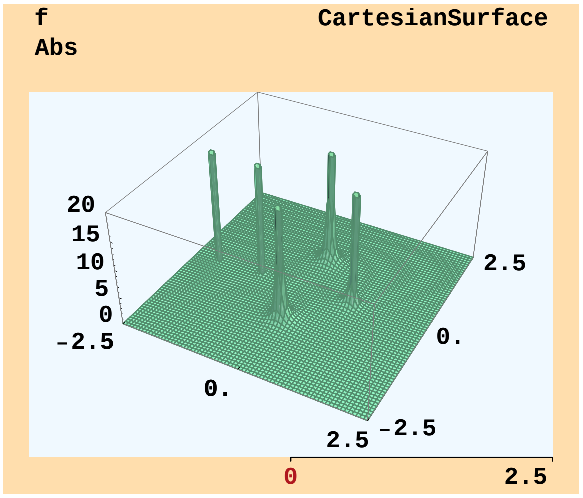
One way to visualize the behavior of a function $f : \mathbb{C} \to \mathbb{C}$ at singularities is to plot in 3–space the modulus $|f| : \mathbb{C} \to \mathbb{R}$. This can be realized in a `OnePanelPlot` of type `CartesianSurface` with first argument `{f,Abs}`. (More generally, an argument `{f,σ}` for a function $\sigma : \mathbb{C} \to \mathbb{R}$ will provide a plot of the composite $\sigma \circ f : \mathbb{C} \to \mathbb{R}$. Other instructive cases are σ = Re and σ = Im.)

The following function has poles at $z = -2$, $z = -1$, $z = 1$, $z = i$, and $z = -i$.

```
f[z_] := z Cos[z] / ((z - 1)^2 (z^2 + 1)^2 (z^2 + 3 z + 2))
```

The following plot depicts that $\lim_{z \to z_0} |f(z)| = \infty$ at each pole z_0 . (We could have used animation to zoom in on each pole.)

```
OnePanelPlot[{f, Abs},
  CartesianSurface, {0, 2.5},
  {ColorMix[MediumSeaGreen, White][0.3]}, {},
  {PlotRange → {0, 20}, PlotPoints → {71, 71},
   BoxRatios → {1, 1, 0.5}, Background → AliceBlue},
  AspectRatio → 0.85, ImageSize → 3 * 72];
```



■ Branch points

The next example, suggested by a problem in Needham [4, p. 117] uses functionality of Cardano3 that is more advanced than what might be introduced in a first course in complex analysis. It concerns the following complex function.

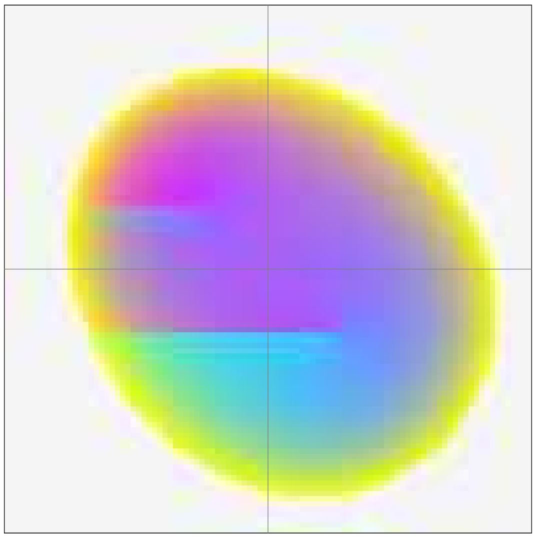
```
f[z_] := √(z - 1) ∛(z - ⅈ)
```

The objective is to determine the nature of the branch points $z = 1$ and $z = i$. To do so we shall employ a different representation of a complex function that uses a single complex plane: at each point z in that plane form a vector pointing from z to $f(z)$.

First we make a background plot on which to superimpose complex vector plots.

```
colfun0[{absmin_, absmax_}, {argmin_, argmax_}][z_] :=
 AbsArg3Color[Cyan, Magenta, Yellow, Gray, Smoke, 5.0][
   {absmin, absmax}, {argmin, argmax}][z]
```

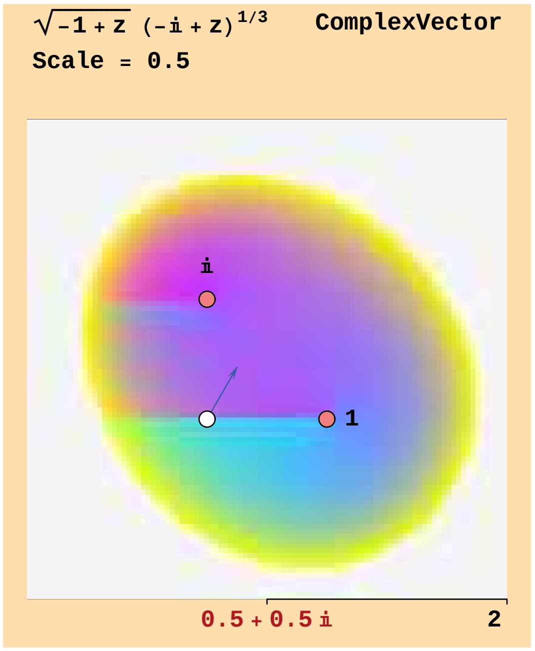
```
backplot = DoShow[CodedDensity[{f}, {(1 + ⅈ) 0.5, 2},
    {colfun0[{0, 1.5}, {-π, π}]}, {}, {ImageSize → 2.75 * 72}]];
```



Such a `CodedDensity` plot is an analog for a complex–valued function of a primitive *Mathematica* `DensityPlot` for a real–valued function of two real variables. But instead of representing how a real value varies over the plane, it represents how both the argument and modulus together vary over the complex plane. In our example, the color varies from cyan to magenta as the argument of `f[z]` varies from $-\pi$ to π, and shades to yellow as the modulus of `f[z]` varies from `0` to `1.5`. Cutting off at `1.5` gives us a slight indication of the modulus contours.

Next, we plot the value `f[z]` of the function at `z` as a vector laid over the domain coloring plot. But *Mathematica* gives us only one of the possible values of what is, after all, a multifunction.

```
OnePanelPlot[{f[#] &}, ComplexVector, {(1 + ⅈ) 0.5, 2, 0.5},
  {{0}, CirclePoint[#, 3, Black, White] &},
  {Prolog[First@backplot],
   ComplexCirclePoint[#, 3, Black, LightCoral] & /@ {1, ⅈ},
   ComplexText[1, 1, {-2, 0}], ComplexText[ⅈ, ⅈ, {0, -2}]},
  {}, ImageSize → 2.75 * 72];
```



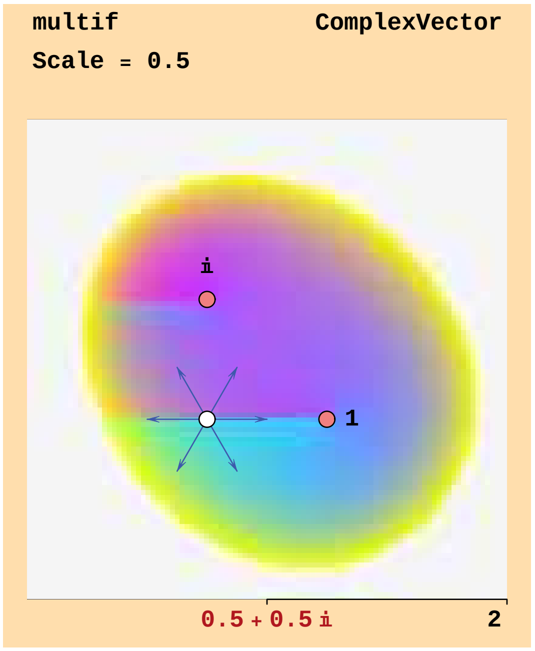
We have embellished the plot by adding circle points at the two branch points and labeling them.

In order to find all branches of the multifunction, we solve `f[z] = w` for `w` by taking 6th powers to clear the radical, so that the equation to be solved becomes $(-1+z)^3\,(-i+z)^2 == w^6$. Solving for the 6 values of the multifunction is encapsulated in `multif[z]`, which uses the Cardano3 routine `Multifunction`.

```
multif[z_] = Multifunction[w /. Solve[f[z]^6 == w^6, w]];
```

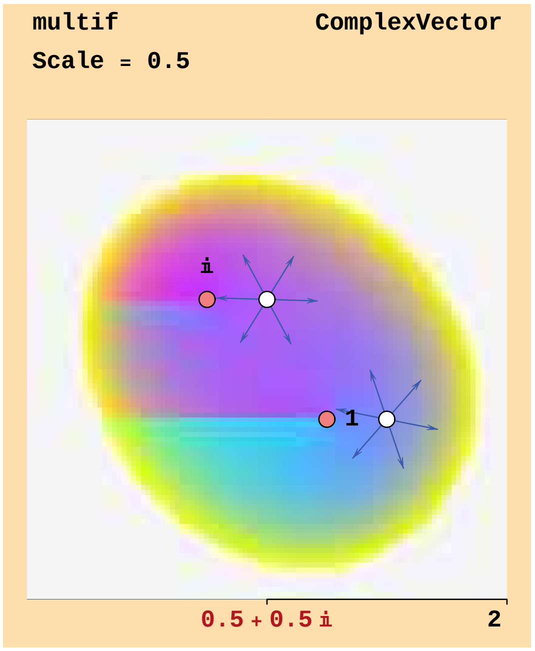
The result simply contains a list of the values to be represented graphically. We can now more accurately depict the function.

```
OnePanelPlot[{multif}, ComplexVector, {(1 + i) 0.5, 2, 0.5},
  {{0}, CirclePoint[#, 3, Black, White] &},
  {Prolog[First@backplot],
   ComplexCirclePoint[1, 3, Black, LightCoral],
   ComplexCirclePoint[i, 3, Black, LightCoral],
   ComplexText[1, 1, {-2, 0}], ComplexText[i, i, {0, -2}]},
  {}, ImageSize → 2.75 * 72];
```



Next, we shall create an animation showing what happens to the vectors as a moving point circles each of the two branch points.

```
frame1[θ_] :=
  OnePanelPlot[{multif}, ComplexVector, {(1 + i) 0.5, 2, 0.5},
   {{{1 + 1 / 2 e^(i θ), i + 1 / 2 e^(i θ)}},
    CirclePoint[#, 3, Black, White] &},
   {Prolog[First@backplot], ComplexCirclePoint[1, 3, Black,
     LightCoral], ComplexCirclePoint[i, 3, Black, LightCoral],
    ComplexText[1, 1, {-2, 0}], ComplexText[i, i, {0, -2}]},
   {}, ImageSize → 2.75 * 72];

Animate[frame1[θ], {θ, 0, 2 π - 2 π / 48, 2 π / 48}]
SelectionMove[EvaluationNotebook[], All, GeneratedCell]
FrontEndTokenExecute["OpenCloseGroup"]; Pause[0.5];
FrontEndTokenExecute["SelectionAnimate"];
```



Thus a vector returns to its original position after circulating twice around the branch point 1, whereas a three–fold circuit of the branch point i is necessary for a vector to return to its original position. Moreover, even though there are discontinuities in the principal Arg values on the horizontal lines ending at the branch points, there is no discontinuity of the multifunction as it crosses these lines.

■ Concluding remarks

Ideally, students coming to a complex analysis course where Cardano3 is used would already be experienced with *Mathematica*. In reality, unfortunately, this is seldom the case: students must learn fundamentals about *Mathematica* along with specifics about Cardano3 as they are learning about complex numbers and complex functions. In the first–named author's course, two days' class time was spent in a laboratory setting with a hands–on, rapid introduction to *Mathematica*, including a first glimpse of some functionality of Cardano3. Although that arrangement is hardly optimal, it sufficed to get them started.

To a *Mathematica* novice, the syntax of even a OnePanelPlot, with its nested lists, can be daunting. In the first author's course, few students succeeded in constructing a syntactically correct TwoPanelPlot without direct access to the documentation; they were therefore encouraged to use instructor–provided templates for their own work. For such reasons, and because of the more general pedagogical overload of introducing both *Mathematica* and Cardano3, it might be useful if Cardano3 were provided as a web*Mathematica* application. Of course users of such an on–line system would not have direct access to other capabilities of *Mathematica,* and they would be confined to the particular functionality of the on–line application.

■ References

[1] Cardano3, D. J.M. Park, Jr., (Dec. 2005) home.earthlink.net/~djmp/Cardano3Page.html.

[2] M. Eisenberg, Math 421 *Mathematica* notebooks, (Fall 2004) www.math.umass.edu/Courses/Math_421, Files page.

[3] J. H. Mathews and R. W. Howell, *Complex Analysis for Mathematics and Engineering,* 5 ed., Sudbury, Mass.: Jones and Bartlett, 2006.

[4] T. Needham, *Visual Complex Analysis,* New York: Oxford University Press, 1997.

[5] DrawGraphics, D. J.M. Park, Jr., (March 2006) home.earthlink.net/~djmp/DrawGraphicsPage.html.

[6] D. J. M. Park, Jr., "DrawGraphics," *Mathematica in Education and Research,* **10**, no. 1, 2005 pp. 44-73.

[7] RootSearch, T. Ersek, (July 2005) http://library.wolfram.com/infocenter/MathSource/4482/.

[8] Anon., posting to MathGroup e−mail list (Feb. 2006) http://forums.wolfram.com/mathgroup/archive/2006/Feb/msg00336.html.

About the Authors

Murray Eisenberg is Professor of Mathematics and Statistics at the University of Massachusetts Amherst. His principal mathematical interest is the topology of dynamical systems. He has published articles on topological dynamics, the APL and J programming languages, and the use of computers in teaching undergraduate mathematics; he is the author of three undergraduate textbooks. He received his A.B. and A.M. from the University of Pennsylvania and his Ph.D. from Wesleyan University.

David J. M. Park, Jr., received a B.S. and M.S. in electrical engineering from M.I.T. He worked on microwave and beam design elements of early cesium beam tubes for atomic clocks and on masers. While working as a computer consultant he became involved in biochemistry and developmental biology, published a number of articles in the field, and worked for a period at the Laboratory for Theoretical Biology at N.I.H. In his retirement he has used *Mathematica* to renew an interest in mathematical physics and in the process has developed packages used by many *Mathematica* users. Most recently he has been collaborating with Renan Cabrera and Jean−François Gouyet to design Tensorial, a *Mathematica* package for tensor calculus.

Murray Eisenberg
Department of Mathematics and Statistics
University of Massachusetts
Lederle Graduate Research Tower
710 North Pleasant Street
Amherst, MA 01003−9305 USA
murray@math.umass.edu
www.math.umass.edu/~murray

David J. M. Park, Jr.
1429 Searchlight Way
Mount Airy, MD 21771 USA
djmp@earthlink.net
home.earthlink.net/~djmp/