Math 4010/5160 [Complex Variables]

Some Plots to supplement our discussion of...

Conformal Mappings

Also, a construction (and plots related to)...

Linear Fractional Transformations

(i.e. Mobius Transformations)

```
> restart;
  with(LinearAlgebra):
  with(plots):

  assume(x,'real',y,'real',t,'real');
```

Our conformal map (note that g'(z) = 2z so this is not conformal at z=0)...

```
> g := z -> z^2:
  'g(z)' = g(z);
```

$$g(z) = z^2 \tag{1}$$

This is a parameterization of the circle: $(x-1)^2 + y^2 = 1$ (i.e. radius 1 centered at (1,0)).
We have that $0 = tA \le t \le tB = 2\pi$.

B1 plots the circle before g(z) is applied (i.e. in the domain) and A1 plots the image of the circle under the mapping g(z) (i.e. in the codomain).

```
> xp := t -> cos(t)+1:
  yp := t -> sin(t):
  tA := 0:  tB := 2*Pi:

  'r(t)' = [xp(t),yp(t)];

  B1 := plot([xp(t),yp(t), t=tA..tB], scaling=constrained, color=red):
  A1 := plot([Re(g(xp(t)+I*yp(t))), Im(g(xp(t)+I*yp(t))), t=tA..tB], scaling=
  constrained, color=red):
```

$$r(t) = [\cos(t\sim) + 1, \sin(t\sim)] \tag{2}$$

This parameterizes a line segment with $x = 1$ and $y = t$ for $-2 \le t \le 2$.

B2 and A2 are then plots before and after g(z) is applied.

```
> xp := t -> 1:
  yp := t -> t:
  tA := -2:  tB := 2:
```

```
'r(t)' = [xp(t),yp(t)];

B2 := plot([xp(t),yp(t), t=tA..tB], scaling=constrained, color=green):
A2 := plot([Re(g(xp(t)+I*yp(t))), Im(g(xp(t)+I*yp(t))), t=tA..tB], scaling=
constrained, color=green):
```

$$r(t) = [1, t\sim] \quad \textbf{(3)}$$

This parameterizes a line segment with $x = 2t$ and $y = -t - 1$ for $-1 \le t \le 1$.

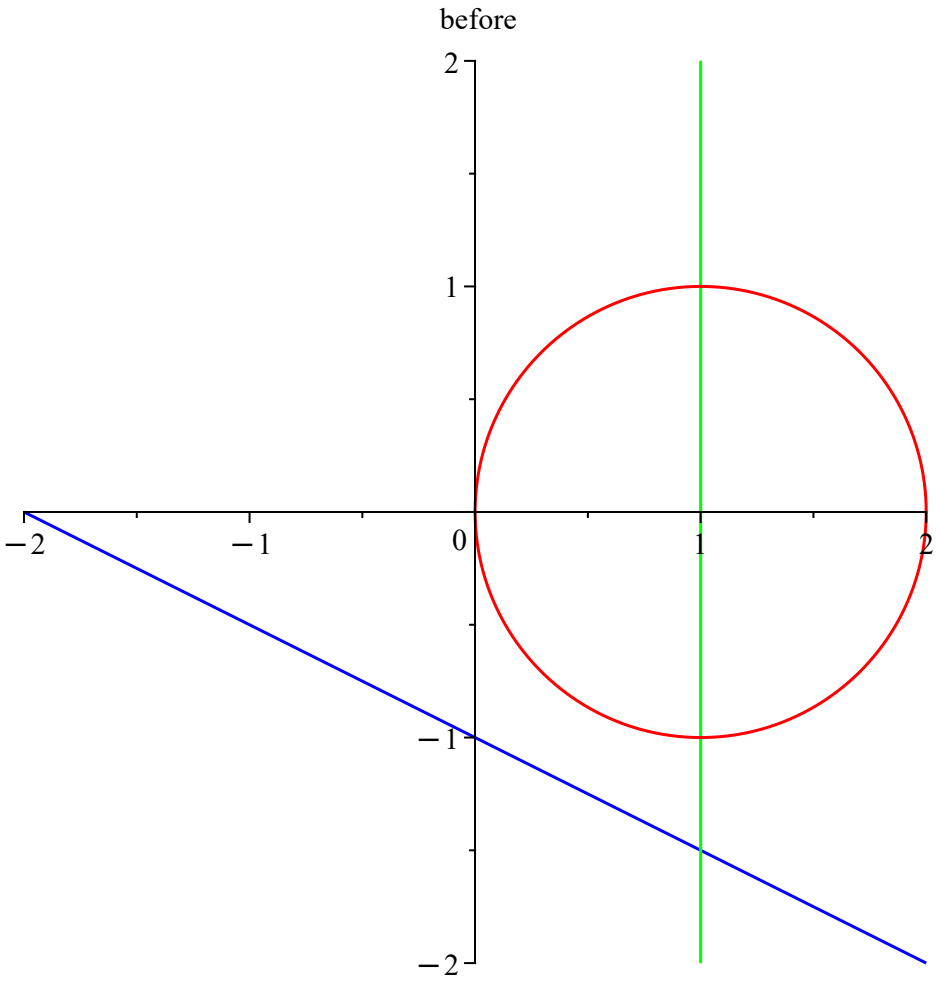
Again, B3 and A3 are the before and after plots.

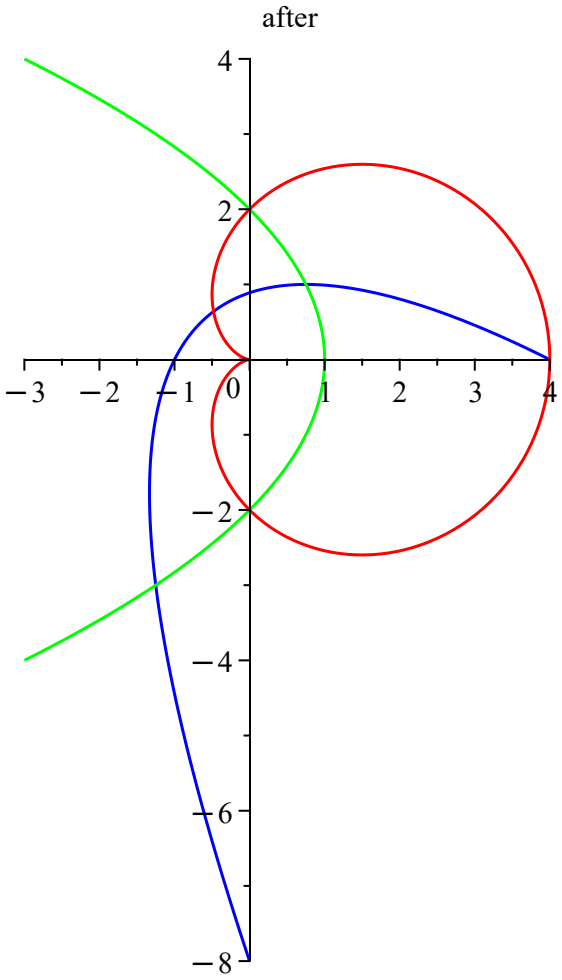
```
> xp := t -> 2*t:
  yp := t -> -1-t:
  tA := -1:  tB := 1:

  'r(t)' = [xp(t),yp(t)];

  B3 := plot([xp(t),yp(t), t=tA..tB], scaling=constrained, color=blue):
  A3 := plot([Re(g(xp(t)+I*yp(t))), Im(g(xp(t)+I*yp(t))), t=tA..tB], scaling=
  constrained, color=blue):
```

$$r(t) = [2t\sim, -1 - t\sim] \quad \textbf{(4)}$$

These commands display the before plots together (in the domain of g(z)) and the after plots together (in the codomain of g (z))...

```
> display({B1,B2,B3},scaling=constrained,title="before");

  display({A1,A2,A3},scaling=constrained,title="after");
```


before
2
1
−2
−1
0
1
2
−1
−2



Linear Fractional Transformations

This code associates a linear fractional transformation with a 2x2 matrix A...

```
> f := (A,z) -> (A[1,1]*z+A[1,2])/(A[2,1]*z+A[2,2]):
  'f_A(z)' = f(A,z);
```

$$f\_A(z) = \frac{A_{1,1}\,z + A_{1,2}}{A_{2,1}\,z + A_{2,2}} \tag{5}$$

This is code to automate the "existence" proof creating a linear fractional transformation sending z0,z1,z2 to w0,w1,w2.

matrixToLFT(A) outputs a function (= linear fractional transformation) associated with a 2x2 matrix A.

ptsToMatrix(z1,z2,z3) creates a matrix which is associated with a linear fractional transformation which sends z1,z2,z3 to $0, 1, \infty$.

ptsTopts(z1,z2,z3,w1,w2,w3) creates function (= linear fractional transformation) which sends z1,z2,z3 to w1,w2,w3.

Note: ∞ is an allowed value of for a z or w point. On the other hand, this code does not check that the user supplied distinct points.

```
> matrixToLFT := proc(A)
      return z -> simplify((A[1,1]*z+A[1,2])/(A[2,1]*z+A[2,2]));
```

```
  end proc:

  ptsToMatrix := proc(z1,z2,z3)
     if z1=z2 or z1=z3 or z2=z3 then return "Use Distinct Points"; end if;

     if z1=infinity then
        return <<0,1>|<z2-z3,-z3>>;
     end if;

     if z2=infinity then
        return <<1,1>|<-z1,-z3>>;
     end if;

     if z3=infinity then
        return <<1,0>|<-z1,z2-z1>>;
     end if;

     return <<z2-z3,z2-z1>|<-z1*(z2-z3),-z3*(z2-z1)>>;
  end proc:

  ptsToPts := proc(z1,z2,z3,w1,w2,w3)
     local A,B;

     A := ptsToMatrix(z1,z2,z3);
     B := ptsToMatrix(w1,w2,w3);

     return matrixToLFT(B^(-1).A);
  end proc:
```

Example: Create a map g(z) which sends 1,i,∞ to 0,-1,i

Maple can verify that 1 maps to 0 and i maps to -1 (it has trouble with ∞ although the formula is actually correct).

```
> g:= ptsToPts(1,I,infinity,0,-1,I):
  'g(z)' = g(z);
```

$$g(z) = \frac{-\mathrm{I}\,(1-z)}{1+z} \tag{6}$$

```
> g(1),g(I),g(infinity);
```

$$0,\ -1,\ \text{undefined} + \text{undefined}\,\mathrm{I} \tag{7}$$

This code replicates what was done for the conformal map above. Now we apply our linear fractional transformation.

```
> xp := t -> cos(t)+1:
  yp := t -> sin(t):
  tA := 0:  tB := 2*Pi:

  'r(t)' = [xp(t),yp(t)];

  B1 := plot([xp(t),yp(t), t=tA..tB], scaling=constrained, color=red):
```

```
A1 := plot([Re(g(xp(t)+I*yp(t))), Im(g(xp(t)+I*yp(t))), t=tA..tB], scaling=
constrained, color=red):
```

$$r(t) = [\cos(t\sim) + 1, \sin(t\sim)] \quad \textbf{(8)}$$

```
> xp := t -> 1:
  yp := t -> t:
  tA := -2:  tB := 2:

  'r(t)' = [xp(t),yp(t)];

  B2 := plot([xp(t),yp(t), t=tA..tB], scaling=constrained, color=green):
  A2 := plot([Re(g(xp(t)+I*yp(t))), Im(g(xp(t)+I*yp(t))), t=tA..tB], scaling=
  constrained, color=green):
```

$$r(t) = [1, t\sim] \quad \textbf{(9)}$$

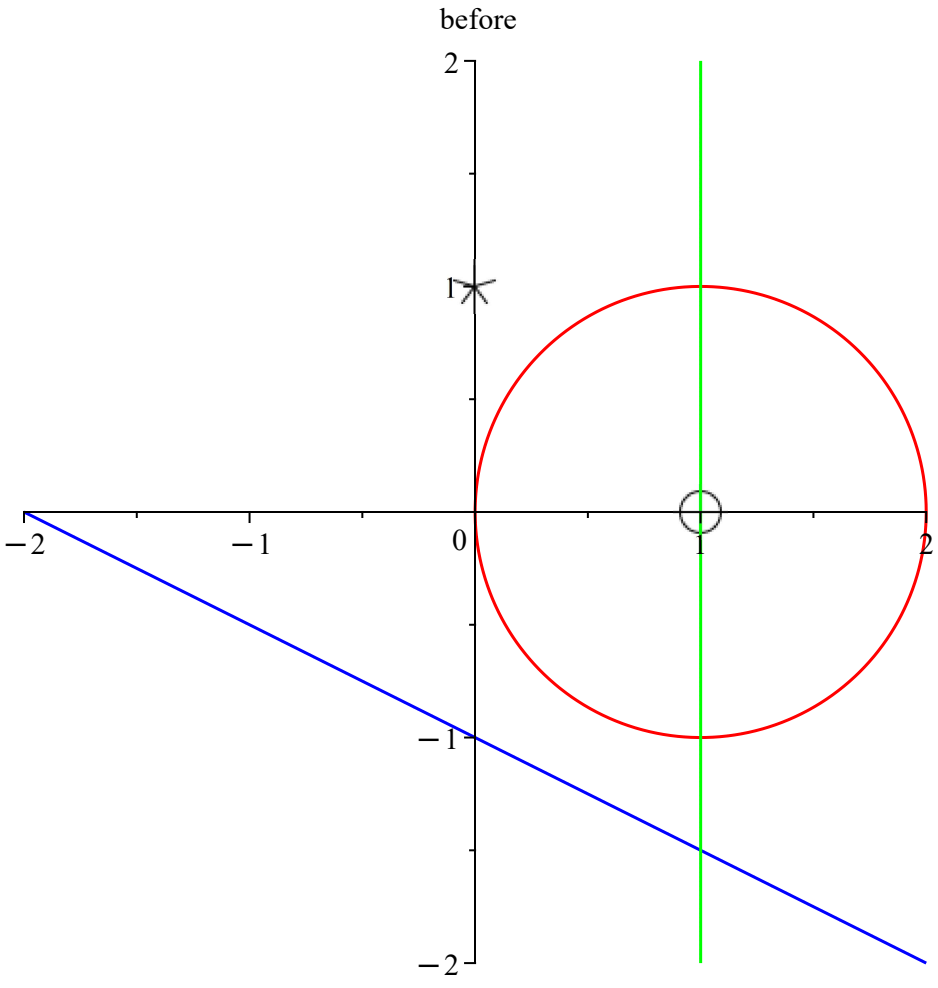
```
> xp := t -> 2*t:
  yp := t -> -1-t:
  tA := -1:  tB := 1:

  'r(t)' = [xp(t),yp(t)];

  B3 := plot([xp(t),yp(t), t=tA..tB], scaling=constrained, color=blue):
  A3 := plot([Re(g(xp(t)+I*yp(t))), Im(g(xp(t)+I*yp(t))), t=tA..tB], scaling=
  constrained, color=blue):
```

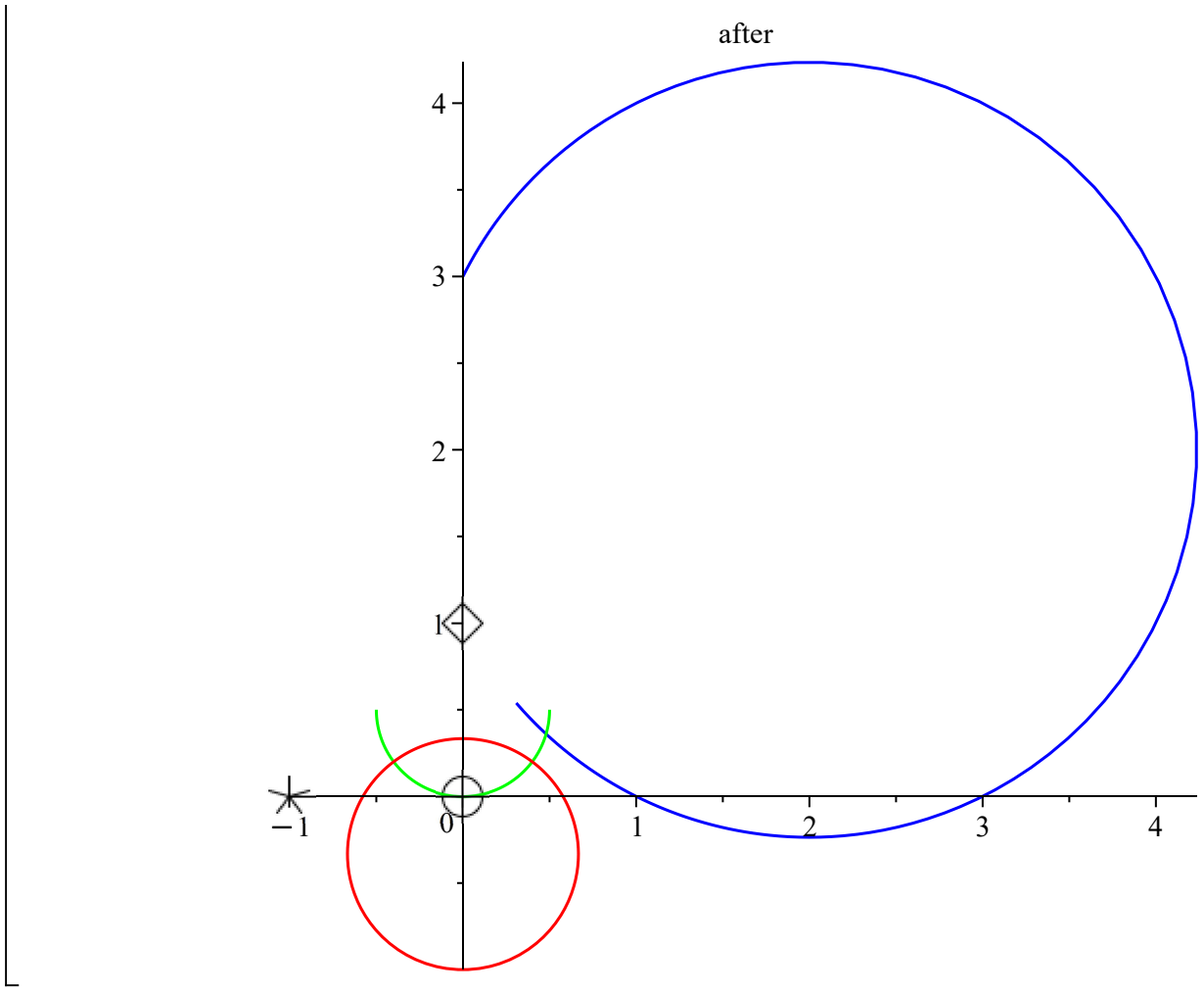
$$r(t) = [2\,t\sim, -1 - t\sim] \quad \textbf{(10)}$$

Notice that the circle and lines transform to circles (since only line segments are plotted in the domain, only arc of circles show up in the codomain).

Also, 1 and i are plotted with a circle and asterisk in the domain. Their images 0 and -1 are plotted with the same symbols in the codomain as well
as the image of ∞ (which is "i" and is plotted with a diamond).

```
> display({B1,B2,B3,
           pointplot([[1,0]],color=black,symbol=circle,symbolsize=30),
           pointplot([[0,1]],color=black,symbol=asterisk,symbolsize=30)},
           scaling=constrained,title="before");

  display({A1,A2,A3,
           pointplot([[0,0]],color=black,symbol=circle,symbolsize=30),
           pointplot([[-1,0]],color=black,symbol=asterisk,symbolsize=30),
           pointplot([[0,1]],color=black,symbol=diamond,symbolsize=30)},
           scaling=constrained,title="after");
```


before
2
1
−2
−1
0
1
2
−1
−2


after
4
3
2
1
−1
0
1
2
3
4