Exponential Map Fractal Viewer

Ioannis Galidakis

Agricultural University of Athens

Static fractal viewer for the Julia sets of the exponential map $\lambda \cdot e^z$ with variable λ. Allows adjustable viewing window, zoom-in and display of periodic attractors up to period 5. Dynamic version can generate animated views, like the Knaster explosion of period 3 shown in the image below.

```
> ####################################################
> # Tetrational Julia Sets Viewer v. 1.0, Ioannis Galidakis,
> # supplementary files to Ph.D. Thesis, Agricultural University of Athens
> # Final Draft: 14 February 2014
> # Project Euler
> ####################################################
> restart;
> with(plots) :
> ####################################################
> Ox := 0; Oy := 0;#Origin
```

$$Ox := 0$$
$$Oy := 0 \quad \textbf{(1)}$$

```
> M := 1;#Magnification
```

$$M := 1 \quad \textbf{(2)}$$

```
> w := 2/M :#window on complex plane
> R := 400 :#Resolution
> lim := -1 :#tentatively
> Niter := 1000 :#Newton's iterations
> Jiter := 300 :#Julia set iterations
> ####################################################
> W := LambertW :
> eps := 0.08 :#numerical resolution
> phi := z->exp(z/exp(z)) :#potential
> delta := -0.02341 :#eps vector for Devaney finger piercing unit circle
> k := 1;#n-th root index
```

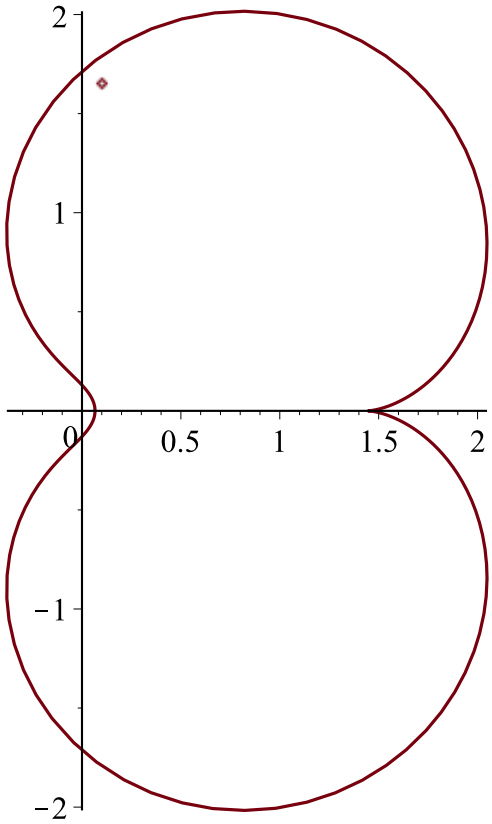
$$k := 1 \quad \textbf{(3)}$$

```
> p := 3;#pre-period
```

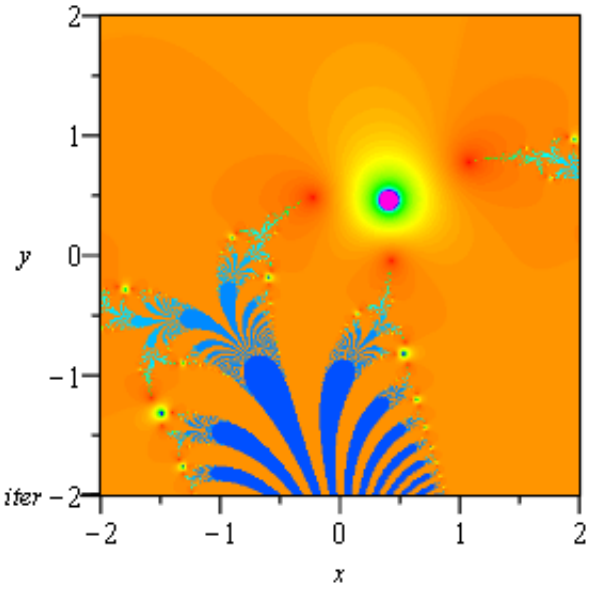
$$p := 3 \qquad (4)$$

```
> t := (1 + delta)·exp( 2·k·Pi·I / p ) :#multiplier+phase angle
> c := evalf(phi(t)) :#parameter for Julia Set
> g := z→c^z :#exponential's iterates
> bail := 1e30 :#bail value for g
> h := (k, z) → W(k, -log(z)) / -log(z) :#main feature of convergence
> ##################################################
> HW :=proc( )#secondary features of convergence
  local y, n, c, s, p, sol, i, aprx, dy, dist, r, newr, oldr, fun, dfun;
  if nargs < 2 then ERROR("At least two arguments required");fi;
  n := args[-1]; y := args[-2]; c := [args[1..-3]];
  if y = 0 then 0;
  else
  dist := infinity;
  fun := exp(z);
  for i from 1 to nargs - 2 do fun := exp(c[-i]·fun) od;
  fun := z·fun - y;
  dfun := diff(fun, z);
  s := series(fun, z, n);
  p := convert(s, polynom);
  sol := {fsolve(p = 0, z, complex)};
  for i from 1 to nops(sol) do
  aprx := evalf(subs(z = op(i, sol), fun));
  dy := evalf(abs(aprx));
  if dy ≤ dist then
  r := op(i, sol);
  dist := dy;
  fi;
  od;
  oldr := r;
  newr := r - evalf( subs(z = r, fun) / subs(z = r, dfun) );
  for i from 1 to Niter while abs( (oldr - newr) / oldr ) > eps do
  oldr := newr;
  newr := newr - evalf( subs(z = newr, fun) / subs(z = newr, dfun) );
  od;
  newr;
  fi;
  end:
> ppc :=proc(c)#find preperiod using multiplier
  local p, z0, at;global lim;
  p := -1; z0 := evalf(h(0, c));
  at := evalf(abs(subs(z = z0, diff((g@@1)(z), z))));
  if at ≤ 1 then p := 1; lim := z0;fi;
```

```
    if p =-1 then
    z0 := evalf( HW(-log(c), log(c), 15) / log(c) );
    at := evalf(abs(subs(z=z0, diff((g@@2)(z), z))));
    if at ≤ 1 then p := 2; lim := z0;fi;
    fi;
    if p =-1 then
     z0 := evalf( HW(-log(c), log(c), log(c), 15) / log(c) );
     at := evalf(abs(subs(z=z0, diff((g@@3)(z), z))));
     if at ≤ 1 then p := 3; lim := z0;fi;
     fi;
    if p =-1 then
     z0 := evalf( HW(-log(c), log(c), log(c), log(c), 15) / log(c) );
     at := evalf(abs(subs(z=z0, diff((g@@4)(z), z))));
     if at ≤ 1 then p := 4; lim := z0;fi;
     fi;
    if p =-1 then
     z0 := evalf( HW(-log(c), log(c), log(c), log(c), log(c), 25) / log(c) );
     at := evalf(abs(subs(z=z0, diff((g@@5)(z), z))));
     if at ≤ 1 then p := 5; lim := z0;fi;
     fi;
     p;
     end:
> #debug(ppc);
> lambda := evalf(log(c));#speed up Julia calculations
                        λ := 0.5038485943 + 1.509520643 I                (5)
> ppc(c), lim;
                        1, 0.4069680050 + 0.4593148563 I                 (6)
> ######################################################
> RST := complexplot(phi(exp(theta·I)), theta = 0..2·Pi, scaling = CONSTRAINED):
> cp := complexplot([c], style = POINT, symbol = DIAMOND):
> display(RST, cp);#map
```




```
> ##################################################
> J := proc(x, y)#Plot Julia Set for point x+yi in C
  local z, m; z := evalhf(x + y·I); m := 1;
  to Jiter while abs(z − lim) > eps and abs(z) < bail do
  z := evalhf(exp(lambda·z));
  m := m + 1;
  od;
  ln(m);
  end:
> ##################################################
> Jhf := (x, y) → evalhf(J(x, y)) :#Carl's patch for evalhf
> ##################################################
> t1 := time( ) :
> p1 := plot3d(Jhf, Ox − w..Ox + w, Oy − w..Oy + w, grid = [R, R], shading = zhue, lightmodel
      = none, style = PATCHNOGRID, orientation = [−90, 0], axes = BOX, labels = [x, y, iter]) :
> display(p1);
```




```
> t2 := time( ) :
> print("Julia Set for c=", c, " in time:", t2 − t1);
```

"Julia Set for c=", 0.1013526298 + 1.651972558 I, " in time:", 16.708 **(7)**

```
>
```