1 Functional graphics

```python
import numpy as np
from matplotlib import pyplot as plt
from functools import partial, reduce
from operator import add, mul
```

```python
I = lambda t: t
const = lambda c: lambda _: c
compose = lambda *fs: reduce(lambda f, g: lambda *x: f(g(*x)), fs)
pure = lambda t: [t]
iterate = lambda n: lambda f: lambda x: iterate(n-1)(f)(f(x)) if n > 0 else x
mcompose = lambda *fls: reduce(lambda fl, gl: [compose(f, g) for f in fl for g in gl], fls)
```

```python
def apply(f, *args, **kwargs):
    return f(*args, **kwargs)
```

```python
lapply = lambda *fs: lambda t: np.array([f(t) for f in fs])
```

```python
origin = lapply(const(0), const(0))
```

```python
line = lapply(lambda t: 2*t - 1, const(0))
```

```python
ltrans = lambda T: lambda x: T @ x
rot = lambda θ: ltrans(np.array([[np.cos(θ), np.sin(θ)], [-np.sin(θ), np.cos(θ)]]))
rotcw = rot(np.pi / 2)
rotccw = rot(-np.pi / 2)
```

```python
rotccw(np.array([-1, 1]))
```

```
array([-1., -1.])
```

```python
split = lambda *fs: lambda t: fs[len(fs) - 1 if t == 1 else int(t * len(fs))](t*len(fs) % 1)
```

```python
curve = split(compose(rot(1), line), line)
```

```python
plt.figure(figsize=(5, 5))
plt.plot(*zip(*map(curve, np.linspace(0, 1))), 'wo')
plt.axis('off');
```

1.1 2D regions and fractals

```
pair = lambda *fs: lambda a: np.array([f(*a) for f in fs])
constv = lambda a, b: pair(const(a), const(b))
square = pair(lambda u, v: 2*u - 1, lambda u, v: 2*v - 1)
circle = pair(lambda u, v: v * np.cos(2*np.pi*u), lambda u, v: v * np.sin(2*np.pi*u))
henon = lambda a, b: pair(lambda u, v: 1 - a*u**2 + v, lambda u, v: b*u)
ikedat = lambda u, v: 0.4 - 6 / (1 + u**2 + v**2)
ikeda = lambda a: pair(
    lambda u, v: 1 + a*(u*np.cos(ikedat(u, v)) - v*np.sin(ikedat(u, v))),
    lambda u, v: a*(u*np.sin(ikedat(u, v)) - v*np.cos(ikedat(u, v)))
)
rapply = lambda g: lambda f: lambda *args: g(*f(*args))
```

```
trans = lambda *cs: partial(add, np.array(cs))
resize = lambda *cs: partial(mul, np.array(cs))
scale = lambda c: resize(c, c)
```

```
transforms = [
    circle,
    scale(0.5),
    trans(1/2, 1/2),
] * 2
transforms.reverse()
shape = compose(*transforms)
shape([1/2, 1/2])
```

```
array([0.5 , 0.75])
```

```python
n = 100
uniform = np.linspace(0, 1, n)
```

Grid

```python
us, vs = np.meshgrid(uniform, uniform)
```

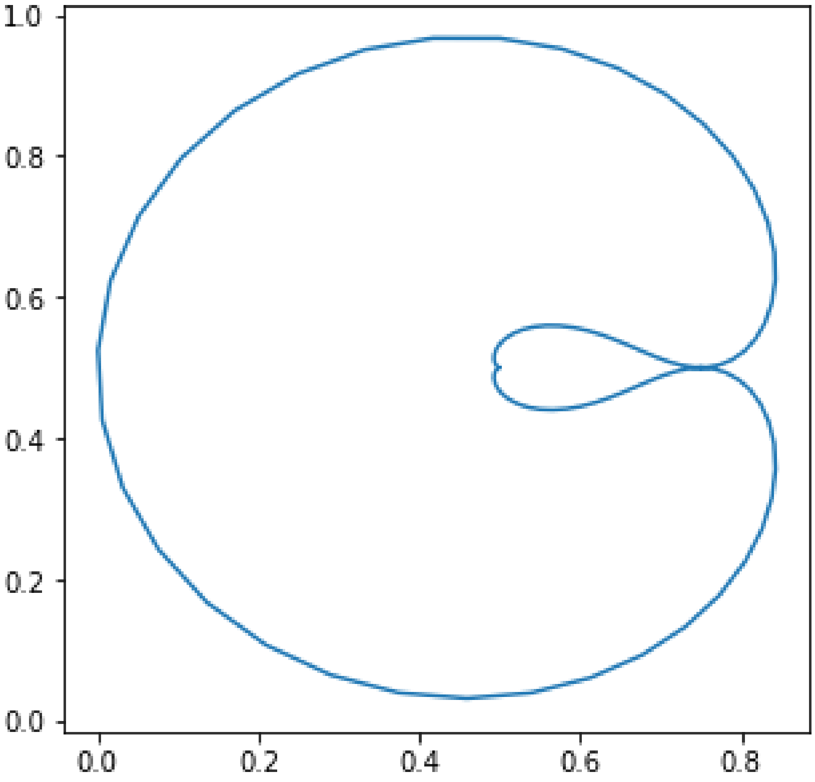
Boundary

```python
us = np.hstack([uniform, np.ones(n), 1 - uniform, np.zeros(n)])
vs = np.hstack([np.zeros(n), uniform, np.ones(n), 1 - uniform])
```

```python
us, vs = uniform, np.ones(n)
```

Draw the shape

```python
xs, ys = zip(*map(shape, zip(us.flat, vs.flat)))
plt.figure(figsize=(5, 5))
plt.plot(xs, ys, '-');
# plt.axis('off');
```

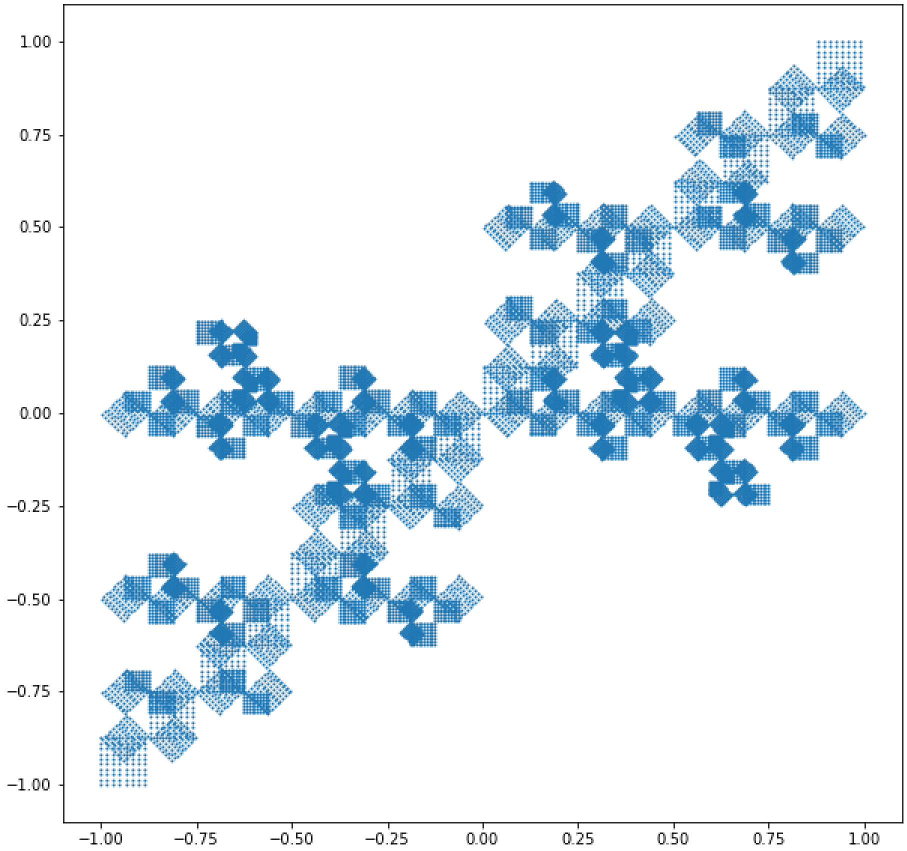


Now fractals

```python
double = lambda f, g: lambda a: compose(f, resize(2, 1))(a) if a[0] < 1/2 else compose(g, resize(2, 1), trans(-1/2, 0))(a)
```

```python
frac = lambda f: (lambda g: compose(scale(1/2), double(g, compose(rot(np.pi/4), scale(1/np.sqrt(2)), g))))(double(compose(trans(-1, -1), f), compose(trans(1, 1), f)))
shape = iterate(4)(frac)(square)
```

```python
us, vs = np.meshgrid(np.linspace(0, 1, 2000), np.linspace(0, 1, 10))
xs, ys = zip(*map(shape, zip(us.flat, vs.flat)))
plt.figure(figsize=(10, 10))
plt.plot(xs, ys, 'o', markersize=1);
```



Now use a graphics library like a regular person.

```python
import cairocffi as cairo
from PIL import Image
```

```python
def draw_point(ctx, p, r=0.005):
    ctx.arc(*p, r, 0, 2*np.pi)
    ctx.fill()
    ctx.stroke()

def draw_line(ctx, line):
    p1, p2 = line
    ctx.move_to(*p1)
    ctx.line_to(*p2)
    ctx.stroke()
```

```
frac = lambda geom: lambda *fs: lambda ls: [geom(f, l) for l in ls for f in fs]
linegeom = lambda f, line: (f(line[0]), f(line[1]))
pointgeom = lambda f, point: f(point)
```

Draw a mandala-like fractal

```
# drawgeom, mapgeom, initgeom = draw_line, linegeom, [([-1, 0], [1, 0])]
drawgeom, mapgeom, initgeom = draw_point, pointgeom, [[0, 0]]

langle = np.pi / 8
n = 4
geoms = iterate(n)(frac(mapgeom)(*mcompose(
    [I, rot(np.pi / 2)],
    [I, resize(-1, 1)],
    [trans(1, 0)],
    [I, resize(-0.5, 1)],
    [I, resize(1, -1)],
    [compose(trans(0, np.tan(langle)), rot(-langle), scale(1 / (2*np.cos(langle))), trans(-1, 0))]
)))(initgeom)
```

```
width, height = 300, 500
surface = cairo.PDFSurface('mandala.pdf', width, height)
ctx = cairo.Context(surface)
ctx.translate(width / 2, height / 2)
side = 0.5 * min(width, height)
ctx.scale(side, side)
ctx.move_to(0, 0)

ctx.set_source_rgba(0, 0, 1, 0.25)
ctx.set_line_width(1e-4)
ctx.scale(0.5, 0.5)
for geom in geoms:
    drawgeom(ctx, geom)

ctx.set_source_rgba(1, 0, 0)
ctx.arc(0, 0, 0.01, 0, 2*np.pi)
ctx.fill()
ctx.stroke()

surface.finish()
```

Distort a region. Note: a better implementation of iterated function systems would be the usual chaos game.

```
def draw_poly(ctx, poly):
    ctx.move_to(*poly[0])
    for p in poly[1:]:
        ctx.line_to(*p)
    ctx.set_source_rgba(1, 0, 1, 0.5)
#     ctx.fill()
    ctx.fill_preserve()
    ctx.set_source_rgba(0, 1, 0, 0.5)
    ctx.stroke()

polygeom = lambda f, poly: [f(p) for p in poly]
```

```
# drawgeom, mapgeom, initgeom = (
#     draw_poly,
#     polygeom,
# #     [[circle([a, 1]) for a in np.linspace(0, 1, 4)]]
# #     [
# #         [[0,0], [1,0], [1/2,1]],
# #         [[1,1], [1,0], [1/2,1]],
# #         [[0,1], [1/2,1], [0,0]]
# #     ]
#     [
#         [[-1,-1], [1,-1], [-1,1]],
#         [[-1,1], [1,1], [1,-1]]
#     ]
# )
drawgeom, mapgeom, initgeom = draw_point, pointgeom, [[0, 0]]

geoms = iterate(8)(frac(mapgeom)(*mcompose(
#     [resize(1, 1/2), compose(scale(1/2), trans(0, 1)), compose(scale(1/2), trans(1, 1))]
    [scale(0.5)],
    [trans(-1, -1), trans(1, -1), trans(-1, 1), trans(1, 1)]
)))(initgeom)
geoms = iterate(12)(frac(mapgeom)(*mcompose(
#     [henon(1.4, 0.3)]
    [ikeda(-1.2)]
)))(geoms)
```

```
width, height = 300, 500
surface = cairo.PDFSurface('distort.pdf', width, height)
ctx = cairo.Context(surface)
ctx.translate(width / 2, height / 2)
side = 0.5 * min(width, height)
ctx.scale(side, -side)

ctx.set_line_width(1e-4)
ctx.scale(0.3, 0.3)
ctx.translate(-1/2, -1)
ctx.set_source_rgba(0, 0, 1, 0.5)
for geom in geoms:
    drawgeom(ctx, geom)

ctx.set_source_rgb(1, 0, 0)
ctx.arc(0, 0, 0.01, 0, 2*np.pi)
ctx.fill()

surface.finish()
```

A line fractal

```
drawgeom, mapgeom, initgeom = (
    draw_line,
    linegeom,
    [([-1, 0], [1, 0])]
)
```

```
m = 3
geoms = iterate(4)(frac(mapgeom)(*mcompose(
    [I, *mcompose(
        [compose(scale(1), rot(np.pi/m))],
        [rot(2*np.pi*i/m) for i in range(m)],
        [compose(scale(-0.5), trans(1, 0))]
    )],
    mcompose(
        [rot(2*np.pi*i/m) for i in range(m)],
        [compose(scale(-0.5), trans(1, 0))]
    )
)))(initgeom)
```

```
width, height = 300, 500
surface = cairo.PDFSurface('linefractal.pdf', width, height)
ctx = cairo.Context(surface)
ctx.translate(width / 2, height / 2)
side = 0.5 * min(width, height)
ctx.scale(side, -side)

ctx.scale(0.75, 0.75)

# ctx.set_source_rgba(0, 0, 0, 0.02)
# draw_poly(ctx, [[0,0],[0,1],[1,1],[1,0]])
# ctx.set_source_rgba(0, 1, 0, 0.5)
# ctx.arc(1/2, 1/2, 0.01, 0, 2*np.pi)
# ctx.fill()

ctx.set_line_width(1e-4)
ctx.set_source_rgba(0, 0, 1, 0.75)
for geom in geoms:
    drawgeom(ctx, geom)

ctx.set_source_rgb(1, 0, 0)
ctx.arc(0, 0, 0.01, 0, 2*np.pi)
ctx.fill()

surface.finish()
```

1.2 Turtle graphics

```
class Turtle:
    def __init__(self, ctx):
        self.x = 0
        self.y = 0
        self.θ = 0
        self.states = []

    def dr(self, r):
        return r*np.cos(self.θ), r*np.sin(self.θ)

    def move(self, r):
        dx, dy = self.dr(r)
        self.x += dx
        self.y += dy
        return self

    def rotate(self, dθ):
        self.θ = (self.θ + dθ) % (2*np.pi)
        return self

    def draw(self, r):
        dx, dy = self.dr(r)
        ctx.move_to(self.x, self.y)
        ctx.rel_line_to(dx, dy)
        ctx.stroke()
        return self.move(r)

    def push(self):
        self.states.append((self.x, self.y, self.θ))
        return self

    def pop(self):
        self.x, self.y, self.θ = self.states.pop()
        return self
```

```
width, height = 300, 500
surface = cairo.PDFSurface('turtle.pdf', width, height)
ctx = cairo.Context(surface)
ctx.translate(width / 2, height / 2)
side = 0.5 * min(width, height)
ctx.scale(side, -side)

ctx.save()
ctx.scale(0.1, 0.1)
ctx.set_line_width(1e-4)
ctx.set_source_rgba(0, 0, 1, 0.5)

t = Turtle(ctx)
tr = 3e-2
for _ in range(10000):
#     t.rotate(2*np.pi*np.random.randint(4)/4)
    t.rotate((np.random.rand() - 0.5) * 2*np.pi / 8)
    t.draw(tr)

ctx.restore()
ctx.set_source_rgb(1, 0, 0)
ctx.arc(0, 0, 0.01, 0, 2*np.pi)
ctx.fill()

surface.finish()
```

1.3 L-systems

```
class LSystem:
    def __init__(self, rules, actions, state):
        self.rules = rules
        self.actions = actions
        self.state = state

    def run(self, syms, n):
        for s in syms:
            if n > 0:
                self.run(self.rules[s], n-1)
```

```python
            else:
                self.state = self.actions[s](self.state)
```

```python
width, height = 300, 500
surface = cairo.PDFSurface('lsystem.pdf', width, height)
ctx = cairo.Context(surface)
ctx.translate(width / 2, height / 2)
side = 0.5 * min(width, height)
ctx.scale(side, -side)

ctx.translate(0, -1.5)
ctx.save()
ctx.scale(0.004, 0.004)
ctx.rotate(-0.1 + np.pi / 2)
ctx.set_line_width(1e-4)
ctx.set_source_rgb(0, 0.25, 0)

LSystem([
    [1,2,4,4,0,5,3,0,5,3,1,4,3,1,0,5,2,0], # X: 0
    [1, 1], # F: 1
    [2], # +: 2
    [3], # -: 3
    [4], # [: 4
    [5], # ]: 5
], [
    I,
    lambda t: t.draw(1),
    lambda t: t.rotate(0.1*(np.random.rand() - 0.5) - np.pi / 7),
    lambda t: t.rotate(0.1*(np.random.rand() - 0.5) + np.pi / 7),
    lambda t: t.push(),
    lambda t: t.pop()
], Turtle(ctx)).run([0], 8)

ctx.restore()
ctx.set_source_rgb(1, 0, 0)
ctx.arc(0, 0, 0.01, 0, 2*np.pi)
ctx.fill()

surface.finish()
```

```python
import itertools
```

```python
def partitions(n, k=0):
    if n ≤ 0:
        return [[]]
    return ([(n-i, q)] + p for i in range(n-k) for p in partitions(i) for q in partitions(n-i-1))
```

```python
def draw_partition(ctx, p, fill=True):
    ctx.set_fill_rule(cairo.FILL_RULE_EVEN_ODD)
    for (i, q) in p:
        if i > 0:
            ctx.save()
            ctx.translate(-2, 0)
            draw_partition(ctx, q)
            ctx.restore()
        r0 = 2*i - 1
        ctx.arc(-r0, 0, r0, 0, np.pi)
#         a = r0 / np.sqrt(3)
#         r = 2*a
#         ctx.arc(-r0, -a, r, np.pi/6, 5*np.pi/6)
        ctx.translate(-4*i, 0)
```

```
width, height = 300, 300
m = 5
surface = cairo.PDFSurface('intpartitions_{}_grid.pdf'.format(m), width, height)
ctx = cairo.Context(surface)
ctx.translate(width / 2, height / 2)
side = 0.5 * min(width, height)
ctx.scale(side, -side)

ctx.set_line_width(1e-2)
ctx.save()

ctx.set_source_rgb(0, 0, 1)

userscale = 9e-4
ctx.translate(-0.95, 0.95)
ctx.scale(userscale, userscale)
ctx.set_fill_rule(cairo.FILL_RULE_EVEN_ODD)
for (j, p) in enumerate(partitions(m)):
    op = np.array([i for (i, _) in p])
    ctx.save()
    for (k, q) in enumerate(partitions(m)):
        if j <= k:
            oq = np.array([i for (i, _) in q])
            olen = min(len(op), len(oq))
            if not (np.any(op[:olen] == oq[:olen]) or np.any(op[-olen:] == oq[-olen:])):
                ctx.save()
#                 ctx.translate(m+2, (m+2) / np.tan(np.pi / 3))
                draw_partition(ctx, p)
                ctx.fill()
                ctx.restore()

                ctx.save()
                ctx.scale(1, -1)
#                 ctx.rotate(2*np.pi / 3)
#                 ctx.translate(m+2, (m+2) / np.tan(np.pi / 3))
                draw_partition(ctx, q)
                ctx.fill()
                ctx.restore()

#                 ctx.save()
#                 ctx.rotate(-2*np.pi / 3)
#                 ctx.translate(m+2, (m+2) / np.tan(np.pi / 3))
#                 draw_partition(ctx, q)
#                 ctx.fill()
#                 ctx.restore()
                ctx.translate(0, -4*m)
#                 ctx.show_page()
    ctx.restore()
    ctx.translate(4*m, 0)
ctx.set_fill_rule(cairo.FILL_RULE_WINDING)

ctx.restore()

surface.finish()
```