Tune-Up: Interpolation

Let's say we want to draw a curve passing through four points. There are (at least) two ways to deribe a curve in flat space. $f(x,y)=0$ or possibly $t \mapsto (f(t),g(t))$. It could be some work to show that one description is a good as the other.

- a curve passes through $(a_1,b_1),(a_2,b_2),(a_3,b_3),(a_4,b_4)$, so that $f(a_n,b_n)=0$ for $n=1,2,3,4$.
- if we do the parameterization, then we have $t_n \mapsto (f(t_n),g(t_n))=(a_n,b_n)$ where we have specified that the curve passes through (a,b) at time t.

If we also specify tangents to the curve, I think it's easier to specify using patermized version.

- the curve passes though point (a,b) at time t with tangent vector (a',b'). Then $(f'(t),g'(t))=(a',b')$.
- If we use equations we have that $(a,b) \in C$ is in our curve and $(a+\epsilon,b+\epsilon) \in C$ is also in our curve so that $f(a,b)=0$ and $f(a+\epsilon,b+\epsilon)=0$ with $\epsilon \ll 1$.

$$\begin{aligned} f(a+\epsilon,b+\epsilon) &= f(a,b)+\epsilon\left(\frac{\partial f}{\partial x}(a,b),\frac{\partial f}{\partial y}(a,b)\right)\cdot(a',b')+O(\epsilon^2)=0 \\ (\nabla f)(a,b)\cdot(a',b') &= 0 \end{aligned}$$

  The **gradient** ∇f of the curve at the point (a,b) is parallel to the tangent vector (a',b').

  Since there are four points and four tangent vectors there are $4+4=8$ equations. If we consider the vector space $\mathbb{R}[x,y]$, the subspace spanned by $x^m y^n$ with $0 \le m+n \le 4$ there are $5\times 5=25$ degrees of freedom.

These considerations will make more sense if we draw the pictures. Let's take as our four points $(a,b)=(1,0),(0,1),(-1,0),(0,-1)$. Let's try a quadratic curve $f(x,y)=\cdot x^2+\cdot xy+\cdot y^2+\cdot x+\cdot y+\cdot=0$. We are only counting six degrees of freedom, because . . . I don't remember combinatorics. $1+2+3=6=\frac{1}{2}\times 4\times 3=\binom{4}{2}$.

There's not enough "room" in our space of conic sections to specify four points and their tangents. On the right side, if we consider polynomials up to degree 100 there's certainly more than enough room. In fact, we could show there's lots of curves in this space that fit this criterion. Let's try $N=4$ we want $\langle x^m y^n : 0 \le m+n \le 4\rangle \subseteq \mathbb{R}[x,y]$.

We have this is a $1+2+3+4+5=15$ dimensional space also $15=\frac{1}{2}\times 6\times 5=\binom{6}{2}$.

Let's try solving one. We have that a circle passes through the points $0 \mapsto (1,0), 1 \mapsto (0,1), 2 \mapsto (-1,0), 3 \mapsto (0,-1)$ and the tangent vectors are $(0,1), (-1,0), (0,-1), (1,0)$. And we could try to switch the first tangent vector from $(0,1)$ to $(0,-1)$ or $(0,1) + \epsilon\,(\cos\theta, \sin\theta)$. And we take a guess for our curve:

$$f(t) = \sum_{n=0}^{7} a_n e^{2\pi i n t} \text{ and } f'(t) = \sum_{n=0}^{7} n\, a_n e^{2\pi i n t}$$

and our time intervals are not $t = 0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}$.

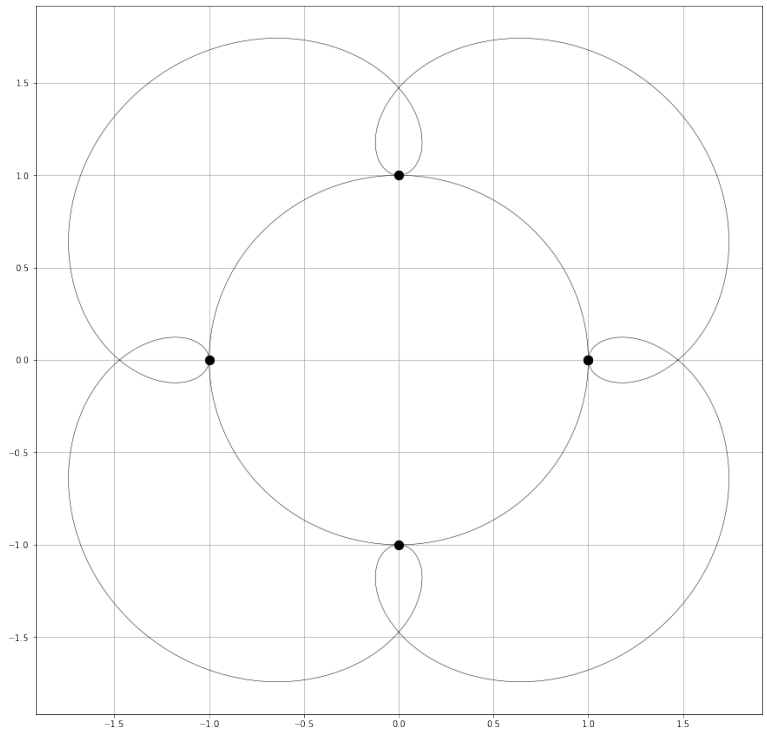
$$\begin{aligned}
f(0) &= a_0 + a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7 = 1 \\
f(\tfrac{1}{4}) &= a_0 + i\,a_1 - a_2 - i\,a_3 + a_4 + i\,a_5 - a_6 - i\,a_7 = i \\
f(\tfrac{1}{2}) &= a_0 + a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7 = -1 \\
f(\tfrac{3}{4}) &= a_0 - ia_1 - a_2 + ia_3 + a_4 - ia_5 - a_6 + ia_7 = -i
\end{aligned}$$

$$\begin{aligned}
f'(0) &= 0\cdot a_0 + a_1 + 2\cdot a_2 + 3\cdot a_3 + 4\cdot a_4 + 5\cdot a_5 + 6\cdot a_6 + 7\cdot a_7 = 1 \\
f'(\tfrac{1}{4}) &= 0\cdot a_0 + i\,a_1 - 2\cdot a_2 - 3\cdot i\,a_3 + 4\cdot a_4 + 5\cdot i\,a_5 - 6\cdot a_6 - 7\cdot i\,a_7 = i \\
f'(\tfrac{1}{2}) &= 0\cdot a_0 + a_1 + 2\cdot a_2 + 3\cdot a_3 + 4\cdot a_4 + 5\cdot a_5 + 6\cdot a_6 + 7\cdot a_7 = -1 \\
f'(\tfrac{3}{4}) &= 0\cdot a_0 - ia_1 - 2\cdot a_2 + 3\cdot ia_3 + 4\cdot a_4 - 5\cdot ia_5 - 6\cdot a_6 + 7\cdot ia_7 = -i
\end{aligned}$$

Linear algebra just turns this into an 8 dimensional space of possible curves.

$$\left[\begin{array}{cccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & i & -1 & -i & 1 & i & -1 & -i \\
1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\
1 & -i & -1 & i & 1 & -i & -1 & i \\
\hline
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
0 & i & -2 & -3i & 4 & 5i & -6 & -7i \\
0 & -1 & 2 & -3 & 4 & -5 & 6 & -7 \\
0 & -i & -2 & 3i & 4 & -5i & -6 & 7i
\end{array}\right]
\left[\begin{array}{c} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{array}\right]
=
\left[\begin{array}{c} 1 \\ i \\ -1 \\ -i \\ \hline i \\ -1 \\ -i \\ 1 \end{array}\right]$$

The answer should be a circle. Let's hope that our matrix math still works here.



The circle is a conic section, the outer curve which is also tangent at $(\pm 1, 0), (0, \pm 1)$ now has tangents pointing

in the opposite direction, has degree ≥ 6. Easy examples like these – exercises in Linear Algebra – can motivate Algebraic Geometry and the Hilbert Nullstellensatz (theorem on whether curves and equations are the same thing). The Python code here is written in a tedious way and could be more elegant:

```
import numpy as np
import matplotlib.pyplot as plt

M = 2
for m in np.arange(M):
    A = np.zeros((8,8))*0j
// curve passes through four points
    A[:,0] = np.exp(2j*np.pi*(1.0/8)*np.arange(8)*0)
    A[:,1] = np.exp(2j*np.pi*(1.0/8)*np.arange(8)*2)
    A[:,2] = np.exp(2j*np.pi*(1.0/8)*np.arange(8)*4)
    A[:,3] = np.exp(2j*np.pi*(1.0/8)*np.arange(8)*6)

// tangents at four points
    A[:,4] = np.arange(8)*(2*np.pi*1j)
    A[:,5] = np.arange(8)*(2*np.pi*1j)
    A[:,6] = np.arange(8)*(2*np.pi*1j)
    A[:,7] = np.arange(8)*(2*np.pi*1j)

    A[:,4] *= np.exp(2j*np.pi*(1.0/8)*np.arange(8)*0)
    A[:,5] *= np.exp(2j*np.pi*(1.0/8)*np.arange(8)*2)
    A[:,6] *= np.exp(2j*np.pi*(1.0/8)*np.arange(8)*4)
    A[:,7] *= np.exp(2j*np.pi*(1.0/8)*np.arange(8)*6)

  // specify tangents at four points
    a = np.array([1, 1j, -1, -1j,
     1*(2*np.pi*1j)*np.exp(2j*np.pi*(1.0/M)*m),
     1j*(2*np.pi*1j)*np.exp(2j*np.pi*(1.0/M)*m),
     -1*(2*np.pi*1j)*np.exp(2j*np.pi*(1.0/M)*m),
     -1j*(2*np.pi*1j)*np.exp(2j*np.pi*(1.0/M)*m) ])

  // solve for coefficents
    b = np.dot( np.linalg.inv(A).T , a)

  // draw curve
    dt = 10**-3
    t  = np.arange(0,1+dt,10**-3)

    z = np.exp(2j*np.pi*t)

    w = 0
    for n in range(8):
        w += z**n*b[n]

    plt.plot(w.real, w.imag, 'k-', linewidth=0.5)
    plt.plot(w[::250].real, w[::250].imag, 'k.', markersize=20)
plt.grid(True)
```

This is good enough for numerical methods. More interesting examples await.

References

[1] ...