MAE 259B Group 2 Progress Report

Siyuan Chen, Xiangzhou Kong, Long Chen

https://github.com/kmxz/mae259b-project

What we did - Starting point

Start from the homework code

Adapted from the MATLAB sample code, translated into Python, with minor changes and optimizations

Build utilities

Command line interface, 3D visualization tool, code snapshot tool, etc.

(Video: hw-render)

(Video: hw-nodes)

What we did - Performance optimization

Profiling shows 90% of time is spent on calculating F and J

```python
def gradEbAndHessEb(xkm1, ykm1, xk, yk, xkp1, ykp1, φk0):
    itm1 = 2 * tan(0.5 * φk0)
    itm2 = ((-xk + xkp1) * (xk - xkm1) + (-yk + ykp1) * (yk - ykm1))
    itm3 = ((-xk + xkp1) * (yk - ykm1) - (xk - xkm1) * (-yk + ykp1))
    itm4 = itm3 / itm2 ** 2
    itm5 = tan(0.5 * atan(itm3 / itm2))
    itm6 = (1 + itm3 ** 2 / itm2 ** 2)
    itm7 = ((ykm1 - ykp1) / itm2 + itm3 * (2 * xk - xkm1 - xkp1) / itm2 ** 2)
    itm8 = ((-xkm1 + xkp1) / itm2 + itm3 * (2 * yk - ykm1 - ykp1) / itm2 ** 2)
    itm9 = ((-xk + xkm1) / itm2 + (-yk + ykm1) * itm4)
    itm10 = ((-xk + xkm1) * itm4 + (yk - ykm1) / itm2)
    itm11 = (-itm1 + 2 * itm5) * (itm5 ** 2 + 1) * itm5 / itm6 ** 2
    itm12 = (-itm1 + 2 * itm5) * (itm5 ** 2 + 1) / itm6
    itm13 = itm12 / itm6
    itm14 = itm3 ** 2 / itm2 ** 3

    F = np.empty(6)
    F[0] = 2 * ((-xk + xkp1) * itm4 + (-yk + ykp1) / itm2) * itm12
    F[1] = 2 * ((xk - xkp1) / itm2 + (-yk + ykp1) * itm4) * itm12
    F[2] = 2 * itm7 * itm12
    F[3] = 2 * itm8 * itm12
    F[4] = 2 * itm10 * itm12
    F[5] = 2 * itm9 * itm12

    J11 = 2 * ((-2 * xk + 2 * xkp1) * (-xk + xkp1) * itm3 / itm2 ** 3 + 2 * (-xk + xkp1) * (-yk +
  ykp1) / itm2 ** 2) * itm12 + 2 * (-(-2 * xk + 2 * xkp1) * itm14 - (-2 * yk + 2 * ykp1) * itm4) *
  ((-xk + xkp1) * itm4 + (-yk + ykp1) / itm2) * itm13 + 2 * ((-xk + xkp1) * itm4 + (-yk + ykp1) /
  itm2) ** 2 * itm11 + 2 * ((-xk + xkp1) * itm4 + (-yk + ykp1) / itm2) ** 2 * (itm5 ** 2 + 1) ** 2
  / itm6 ** 2
    J12 = 2 * (-itm1 + 2 * itm5) * (itm5 ** 2 + 1) * ((-xk + xkp1) * (xk - xkp1) / itm2 ** 2 +
  (-xk + xkp1) * (-2 * yk + 2 * ykp1) * itm3 / itm2 ** 3 + (-yk + ykp1) ** 2 / itm2 ** 2) / itm6 +
  2 * ((xk - xkp1) / itm2 + (-yk + ykp1) * itm4) * ((-xk + xkp1) * itm4 + (-yk + ykp1) / itm2) *
  itm11 + 2 * ((xk - xkp1) / itm2 + (-yk + ykp1) * itm4) * ((-xk + xkp1) * itm4 + (-yk + ykp1) /
  itm2) * (itm5 ** 2 + 1) ** 2 / itm6 ** 2 + 2 * ((-xk + xkp1) * itm4 + (-yk + ykp1) / itm2) *
  (-(2 * xk - 2 * xkp1) * itm4 - (-2 * yk + 2 * ykp1) * itm14) * itm13
    J13 = 2 * (-itm1 + 2 * itm5) * (itm5 ** 2 + 1) * ((-xk + xkp1) * (ykm1 - ykp1) / itm2 ** 2 +
  (-xk + xkp1) * itm3 * (4 * xk - 2 * xkm1 - 2 * xkp1) / itm2 ** 3 + (-yk + ykp1) * (2 * xk - xkm1
  - xkp1) / itm2 ** 2 - itm4) / itm6 + 2 * itm7 * ((-xk + xkp1) * itm4 + (-yk + ykp1) / itm2) *
```

30.6 s

$\Downarrow$

7.6 s

4x faster

What we did - Initial curvature

When calculating bending energy, replace $(\phi_k)^2$ with $(\phi_k - \phi_{k0})^2$

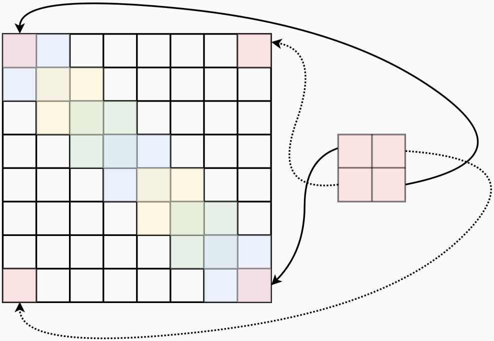
The formulas for calculating F and J need to be changed (do differentiation again)

```
from sympy import *

def φk(xkm, xk, xkp, ykm, yk, ykp):
    return atan(((xkp - xk) * (yk - ykm) - (xk - xkm) * (ykp - yk)) / ((xkp - xk) * (xk - xkm) + (ykp - yk) * (yk - ykm)))

def Ebk(xkm, xk, xkp, ykm, yk, ykp, φko):
    return (2 * tan(φk(xkm, xk, xkp, ykm, yk, ykp) / 2.0) - 2 * tan(φko / 2.0)) ** 2

xkm, xk, xkp, ykm, yk, ykp, φko = symbols('xkm1 xk xkp1 ykm1 yk ykp1 φk0')

Eb = Ebk(xkm, xk, xkp, ykm, yk, ykp, φko)

F1 = diff(Eb, xkm)
F2 = diff(Eb, ykm)
F3 = diff(Eb, xk)
F4 = diff(Eb, yk)
F5 = diff(Eb, xkp)
F6 = diff(Eb, ykp)

J11 = diff(F1, xkm)
J12 = diff(F1, ykm)
J13 = diff(F1, xk)
J14 = diff(F1, yk)
J15 = diff(F1, xkp)
J16 = diff(F1, ykp)
```

What we did - Circular structure

Instead of $nv - 1$ edges, we have nv edges.
For bending, instead of $nv - 2$ components, we have nv components.
For stretching, instead of $nv - 1$ components, we have nv components.

When compositing the Jacobians, new components added to connect two ends together:



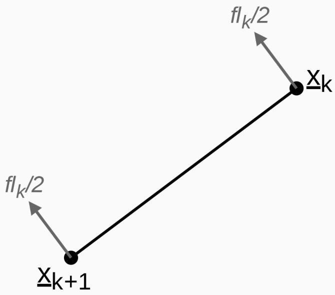
What we did - Circular structure

Verify our code by running the “hanging circle”

$Y = 10^6\,\mathrm{Pa}$ $\quad$ $Y = 10^7\,\mathrm{Pa}$ $\quad$ $Y = 10^8\,\mathrm{Pa}$

(Video: 1e6) (Video: 1e7) (Video: 1e8)

What we did - Inflation pressure



With $\underline{x}_k$ and $\underline{x}_{k+1}$, we can easily calculate force exerted on those two points. Taking derivatives on the forces, we have the corresponding Jabobian matrix.