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

```
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

⇓

7.6 s

4x faster

What we did - Natural curvature

When calculating bending energy, replace $\frac{1}{2}EI(\phi_k)^2$ with $\frac{1}{2}EI(\phi_k - \phi_{k0})^2$

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

To verify the result:

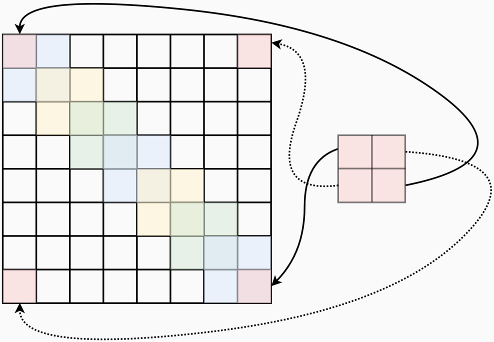
- Expect same result as previous code when $\phi_0 = 0$
- Expect a straight beam to recover natural curvature when no external force applied

```python
from sympy import symbols, diff, tan, atan

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)
```

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:



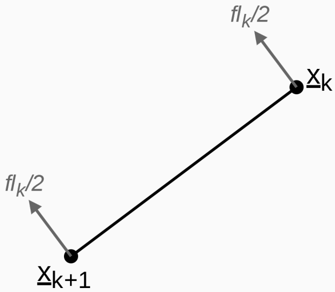
What we did - Circular structure

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.

As force is dependent on actual positions of nodes, we need to add the variation into the Jabobian matrix. We simply take derivatives on the forces.

What we did - Surface contact

Predictor-corrector method is used

Assume a surface at $y = 0$, When doing time-marching, on each frame:

1. Compute $\underline{q}(t)$ as before
2. Check if there exists any node whose $y < 0$. If there is any, set it as a temporarily constrained DOF with $y = 0$, and recompute current frame
3. Check if there exists any temporarily constrained DOF, such that the normal *force* between the surface and the node is negative. Remove such temporary constraint, and recompute current frame

```
Fb, Jb = getFb(qCurrentIterate, EI, nv, voronoiRefLen, -2 * pi / nv, isCircular=True)
Fs, Js = getFs(qCurrentIterate, EA, nv, refLen, isCircular=True)
Fg = m * garr
Fp, Jp = getFp(qCurrentIterate, nv, refLen, InflationPressure)

Forces = Fb + Fs + Fg + Fp

# Equation of motion
f = m * (qCurrentIterate - q0) / dt ** 2 - m * u / dt - Forces
fUncons = dofHelper.unconstrained_v(f)
```

What we did - Surface contact

In each frame:

```python
        qNew, reactionForces = objfun(q0)

        # inspect reactionForces to see if any one is negative, which should be UNCONSTRAINED
        needToFree = [unconsInd for unconsInd in dofHelper._constrained if (reactionForces[
unconsInd] < 0)]
        if needToFree:
            dofHelper.unconstraint(needToFree)
            print('Contact condition updated. Remove constraints and recompute')
            qNew, reactionForces = objfun(q0)

        # inspect qNew to see if any one falls below ground, which should be CONSTRAINED
        while True:
            q0Effective = None
            for c in range(nv):
                index = 2 * c + 1
                if qNew[index] < 0:  # y < 0: bad!
                    if q0Effective is None:
                        q0Effective = q0.copy()
                    q0Effective[index] = 0
                    dofHelper.constraint([index])
            if q0Effective is None:
                break
            else:
                print('Contact condition violated. Add constraints and recompute')
                qNew, reactionForces = objfun(q0Effective)
```

All set!

Low pressure

High pressure

(Video: l-pressure)

(Video: h-pressure)

Project on GitHub now!

All code, results (and these slides) uploaded to

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