Example-01: AD - Stiffness

April 8, 2020

1 Example 1 - AD Stiffness

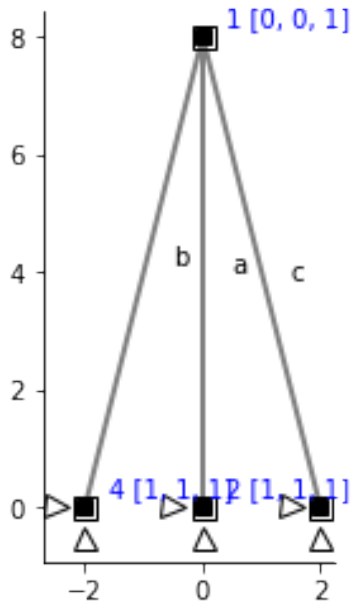
This short script is intended to provide a brief introduction to *algorithmic differentiation* (also known as *automatic differentiation* or AD), through an application in computational mechanics. This is not intended to suggest a practical use for AD, but rather highlight it's utility for those more familiar with concepts in finite element analysis than machine learning.

This script utilizes the `JAX` library for python, which is a research project under development by Google. It is of interest to note the warning produced from the first code cell below, which indicates how readily this framework might be extended for distributed computing.

[1]:
```
from imports import *
# %config InlineBackend.figure_format = 'svg'
```

```
/home/claudio/miniconda3/envs/linjax/lib/python3.8/site-
packages/jax/lib/xla_bridge.py:122: UserWarning: No GPU/TPU found, falling back
to CPU.
  warnings.warn('No GPU/TPU found, falling back to CPU.')
```

[2]:
```
Model = model_def3([1.0e3, 5.0, 10.0])
```



1.1 Newton-Raphson method

[5]:
```
def NewtonRaphson( Pf, Model, tol=1.0e-3):
    nf = Model.nf
    nt = Model.nt
    U = jnp.zeros(nt)

    # create pure function Pr = Pr_U( U ) from function Pr = Pr_vector(U, Model)
    Pr_U = partial(emx.Pr_vector, Model )

    # use 'forward mode' algorithmic differentiation
    # to obtain jacobian of Pr_U(U), Kt(U)
    Jac = jax.jacfwd(_Pr_of_U)
    Kt = lambda U: jnp.squeeze(Jac(U))

    res = tol+1
    while res > tol:
        Pr = Pr_U(U)[:nf]
        Pu = Pf - Pr
        Kf = Kt(U)[:nf,:nf]
        DUf = jnp.linalg.solve(Kf,Pu)

        # reshaping
        DUf = jnp.squeeze(DUf)
        DU = jnp.pad(DUf, (0, Model.nr), 'constant')

        # update displacement
        U += DU

        # residual
        res = jnp.linalg.norm(Pu)

    return U
```

[4]:
```
P = emx.P_vector(Model); # Define loading
```

[6]:
```
NewtonRaphson(P[:Model.nf], Model)
```

[6]:
```
DeviceArray([-0.35046396,  0.04217934,  0.        ,  0.        ,
              0.        ,  0.        ,  0.        ,  0.        ,
              0.        ,  0.        ,  0.        ,  0.        ],
dtype=float32)
```

1.2 Appendix - Resisting force function

[7]:
```
def Pr_vector(Model, U ):

    Qr = Qr_vector(Model,U)

    Pr = Be_tensor(Model)@Qr # Boolean selection array
    return Pr


def Qr_vector(Model, U ):
    Q = npx.concatenate([ElemData.pr_vector(npx.take(U,ElemData.dofs-1)) for␣
 ↪ElemData in Model.ElemData], axis=0)
    return Q


def pr_vector(ElemData, u):
    """Residual force vector for a Green-Lagrange truss"""

    L = ElemData.L0
    DX = ElemData.nodes[1].xyz[0] - ElemData.nodes[0].xyz[0]
    DY = ElemData.nodes[1].xyz[1] - ElemData.nodes[0].xyz[1]

    dU = npx.eye(4)

    v = (u[2]-u[0])*DX/L + (u[1]-u[3])*DY/L + ((u[2]-u[0])**2 + (u[1]-u[3])**2)/
 ↪(2*L)

    dv = npx.array([[-DX/L - (u[2]-u[0])/L],
                    [ DY/L + (u[1]-u[3])/L],
                    [ DX/L + (u[2]-u[0])/L],
                    [-DY/L - (u[1]-u[3])/L]])

    dv = dU@dv
    e = v/L
    q = ElemData.E * ElemData.A * e + ElemData.s0
    return dv*q
```