J.-F. Bercher

September 30, 2020

Contents

```
from sympy import Matrix, Symbol, derive_by_array, Lambda,
    symbols, Derivative, diff
from sympy.abc import x, y, i, j, a, b
```

Defining variable-element matrices $X \in \mathbb{R}^{n\times m}$ and $W \in \mathbb{R}^{m\times p}$:

```
def var(letter: str, i: int, j: int) -> Symbol:
    letter_ij = Symbol('{}_{}{}'.format(letter, i+1, j+1),
        is_commutative=True)
    return letter_ij


n,m,p = 3,3,2

X = Matrix(n, m, lambda i,j : var('x', i, j)); X
```

```
W = Matrix(m, p, lambda i,j : var('w', i, j)); W
```

Defining $N = \nu(X, W) = X \times W$

- $\nu : \mathbb{R}^{(n\times m)\times(m\times p)} \to \mathbb{R}^{n\times p}$
- $N \in \mathbb{R}^{n\times p}$

```
v = Lambda((a,b), a*b); v
```

```
N = v(X, W); N
```

Defining $S = \sigma_{\text{apply}}(N) = \sigma_{\text{apply}}(\nu(X, W)) = \sigma_{\text{apply}}(X \times W) = \left\{\sigma(XW_{ij})\right\}$.

Assume that $\sigma_{\text{apply}} : \mathbb{R}^{n\times p} \to \mathbb{R}^{n\times p}$ while $\sigma : \mathbb{R} \to \mathbb{R}$, so the function σ_{apply} takes in a matrix and returns a matrix while the simple σ acts on the individual elements $N_{ij} = XW_{ij}$ in the matrix argument N of σ_{apply}.

- $\sigma : \mathbb{R} \to \mathbb{R}$
- $\sigma_{\text{apply}} : \mathbb{R}^{n\times p} \to \mathbb{R}^{n\times p}$
- $S \in \mathbb{R}^{n\times p}$

```
from sympy import Function

# Nvec = Symbol('N', commutative=False)

sigma = Function('sigma')
sigma(N[0,0])
```

```
# way 1 of declaring S
S = N.applyfunc(sigma); S
#type(S)
#Matrix(3, 2, lambda i, j: sigma(N[i,j]))
```

```
# way 2 of declaring S (better way)
sigmaApply = lambda matrix:  matrix.applyfunc(sigma)

sigmaApply(N)
```

```
sigmaApply(X**2) # can apply this function to any matrix
    argument.
```

```
S = sigmaApply(v(X,W)) # composing
S
```

Defining $L = \Lambda(S) = \Lambda(\sigma_{\text{apply}}(\nu(X, W))) = \Lambda\left(\left\{\sigma(XW_{ij})\right\}\right)$. In general, let the function be defined as:

$$
\begin{aligned}
L = \Lambda \begin{pmatrix}
\sigma(XW_{11}) & \sigma(XW_{12}) & \dots & \sigma(XW_{1p}) \\
\sigma(XW_{21}) & \sigma(XW_{22}) & \dots & \sigma(XW_{2p}) \\
\vdots & \vdots & & \vdots \\
\sigma(XW_{n1}) & \sigma(XW_{n2}) & \dots & \sigma(XW_{np})
\end{pmatrix} \\
= \sum_{i=1}^{p} \sum_{j=1}^{n} \sigma(XW_{ij}) \\
= \sigma(XW_{11}) + \sigma XW_{12} + \dots + \sigma(XW_{np})
\end{aligned}
$$

NOTE HERE: * $\Lambda : \mathbb{R}^{n\times p} \to \mathbb{R}$ * $L \in \mathbb{R}$

```
lambdaF = lambda matrix : sum(matrix)
lambdaF(S)
```

```
L = lambdaF(sigmaApply(v(X, W)))
L
#L = lambda mat1, mat2: lambdaF(sigmaApply(v(mat1, mat2)))
#L(X, W)
```

```
#derive_by_array(L, X)
```

```
derive_by_array(L, S)
```

```
from sympy import sympify, lambdify
n = lambdify((X[0,0],X[0,1],X[0,2],W[0,0],W[1,0],W[2,0]), N
    [0,0])
n(1,2,3,4,3,2)
```

```
16
```

```
f = Function('f') #(sympify(N[0,0]))
f(N[0,0])
```

```
f(N[0,0]).diff(X[0,0])
```

```
n = v(X,W); n
n11 = Function('{}'.format(n[0,0]))
n11
```

```
w_11*x_11 + w_21*x_12 + w_31*x_13
```

```
s_ij = Function('s_ij')
sig = Function('sig')(x)
```

```
# KEY: got not expecting UndefinedFunction error again here
    too
#S_ij = Matrix(3, 2, lambda i,j: Function('s_{}{}'.format(i+1,
    j+1))(Function('{}'.format(N[i,j]))))
```

```
#S_ij[0,0](sympify(N[0,0])).diff(sympify(N[0,0]))
F = 3*x*y

xy = Symbol('{}'.format(F))
xy.subs({x:3})
sympify(xy).subs({x:3})
```

Sympy Example of trying to differentiate with respect to an **expression** not just a variable.

```
from sympy.abc import t

F = Function('F')
f = Function('f')
U = f(t)
V = U.diff(t)

direct = F(t, U, V).diff(U); direct
```

```
F(t,U,V)
```

```
F(t,U,V).subs(U,x)
```

```
F(t,U,V).subs(U,x).diff(x)
```

```
F(t,U,V).subs(U,x).diff(x).subs(x, U)
```

```
indirect = F(t,U,V).subs(U, x).diff(x).subs(x,U); indirect
```

```
F = Lambda((x,y), 3*x* y)
F(1,2)
```

```
U = x*y
G = 3*x*y
xy
```

```
F.diff(xy)
```

```
# derive_by_array(S, N) # ERROR
```

```
s11 = S[0,0]
s11
```

```


#s11.diff(n11)
```

```
derive_by_array(L, S)
```

```


x, y, r, t = symbols('x y r t') # r (radius), t (angle theta)
f, g, h = symbols('f g h', cls=Function)
h = g(f(x))
Derivative(h, f(x)).doit()
```

```
h.args[0]
h.diff(h.args[0])
```

```
S = sigmaApply(v(X,W)); S
```

```
from sympy.abc import n

n11 = (X*W)[0,0]
m = lambda mat1, mat2: sympify(Symbol('{}'.format((mat1 * mat2
    )[0,0] )))
s = sigma(m(X,W)); s
```

```
s.subs({W[0,0]: 14}) # doesn't work to substitute into an
    undefined function
```

```
Derivative(s, m(X,W)).doit()
```

```


#s11 = Function('s_{11}')(n11); s11
#sigma(n11).diff(n11)

#s11.diff(n11)
sigma(n11)
```

```
# ERROR HERE TOO
type(sigma(n11).args[0])
```

sympy.core.add.Add

```
type(n11)
```

sympy.core.add.Add

```
#sigma(n11).diff(sigma(n11).args[0]) ## ERROR
```

```

```

```
b = Symbol('{}'.format(n11))
ns_11 = Function(b, real=True)
ns_11


# ERROR cannot diff wi.r. to undefinedfunction
# sigma(n11).diff(ns_11)


#
#sigma(b).diff(b).subs({b:1})
```

```
w_11*x_11 + w_21*x_12 + w_31*x_13
```

```
f, g = symbols('f g', cls=Function)
xy = Symbol('x*y'); xy
#sympify(xy).subs({x:2, y:4})
f(g(x,y)).diff(xy)
```

```
# TODO SEEM to have got the expression but it is not working
    since can't substitute anything .... ???
f(xy).diff(xy).subs({x:2})
```

```
Function("x*y")(x,y)
xyf = lambdify([x,y],xy)
xyf(3,4)
f(g(xy)).diff(xy)
#
```

```
xyd = Derivative(x*y, x*y,0).doit();xyd

#Derivative(3*xyd, xyd, 1).doit() ### ERROR can't calc deriv w
    .r.t to x*y
```

```
#derive_by_array(S, N)
```