ch1_phase1

September 24, 2020

[1]:
```python
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 $W \in \mathbb{R}^{m \times p}$:

[2]:
```python
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
```

[2]:
$$\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix}$$

[3]:
```python
W = Matrix(m, p, lambda i,j : var('w', i, j)); W
```

[3]:
$$\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix}$$

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

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

[4]:
```python
v = Lambda((a,b), a*b); v
```

[4]:
$$((a,\ b) \mapsto ab)$$

[5]:
```python
N = v(X, W); N
```

[5]:
$$\begin{bmatrix} w_{11}x_{11} + w_{21}x_{12} + w_{31}x_{13} & w_{12}x_{11} + w_{22}x_{12} + w_{32}x_{13} \\ w_{11}x_{21} + w_{21}x_{22} + w_{31}x_{23} & w_{12}x_{21} + w_{22}x_{22} + w_{32}x_{23} \\ w_{11}x_{31} + w_{21}x_{32} + w_{31}x_{33} & w_{12}x_{31} + w_{22}x_{32} + w_{32}x_{33} \end{bmatrix}$$

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}.

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

[6]:
```python
from sympy import Function
# Nvec = Symbol('N', commutative=False)
sigma = Function('sigma')
sigma(N[0,0])
```

[6]:

$$\sigma(w_{11}x_{11} + w_{21}x_{12} + w_{31}x_{13})$$

[7]:
```python
# way 1 of declaring S
S = N.applyfunc(sigma); S
#type(S)
#Matrix(3, 2, lambda i, j: sigma(N[i,j]))
```

[7]:

$$\begin{bmatrix} \sigma(w_{11}x_{11} + w_{21}x_{12} + w_{31}x_{13}) & \sigma(w_{12}x_{11} + w_{22}x_{12} + w_{32}x_{13}) \\ \sigma(w_{11}x_{21} + w_{21}x_{22} + w_{31}x_{23}) & \sigma(w_{12}x_{21} + w_{22}x_{22} + w_{32}x_{23}) \\ \sigma(w_{11}x_{31} + w_{21}x_{32} + w_{31}x_{33}) & \sigma(w_{12}x_{31} + w_{22}x_{32} + w_{32}x_{33}) \end{bmatrix}$$

[8]:
```python
# way 2 of declaring S (better way)
sigmaApply = lambda matrix:  matrix.applyfunc(sigma)
sigmaApply(N)
```

[8]:

$$\begin{bmatrix} \sigma(w_{11}x_{11} + w_{21}x_{12} + w_{31}x_{13}) & \sigma(w_{12}x_{11} + w_{22}x_{12} + w_{32}x_{13}) \\ \sigma(w_{11}x_{21} + w_{21}x_{22} + w_{31}x_{23}) & \sigma(w_{12}x_{21} + w_{22}x_{22} + w_{32}x_{23}) \\ \sigma(w_{11}x_{31} + w_{21}x_{32} + w_{31}x_{33}) & \sigma(w_{12}x_{31} + w_{22}x_{32} + w_{32}x_{33}) \end{bmatrix}$$

[9]:
```python
sigmaApply(X**2) # can apply this function to any matrix argument.
```

[9]:

$$\begin{bmatrix} \sigma(x_{11}^2 + x_{12}x_{21} + x_{13}x_{31}) & \sigma(x_{11}x_{12} + x_{12}x_{22} + x_{13}x_{32}) & \sigma(x_{11}x_{13} + x_{12}x_{23} + x_{13}x_{33}) \\ \sigma(x_{11}x_{21} + x_{21}x_{22} + x_{23}x_{31}) & \sigma(x_{12}x_{21} + x_{22}^2 + x_{23}x_{32}) & \sigma(x_{13}x_{21} + x_{22}x_{23} + x_{23}x_{33}) \\ \sigma(x_{11}x_{31} + x_{21}x_{32} + x_{31}x_{33}) & \sigma(x_{12}x_{31} + x_{22}x_{32} + x_{32}x_{33}) & \sigma(x_{13}x_{31} + x_{23}x_{32} + x_{33}^2) \end{bmatrix}$$

[10]:
```python
S = sigmaApply(v(X,W)) # composing
S
```

[10]:

$$\begin{bmatrix} \sigma(w_{11}x_{11} + w_{21}x_{12} + w_{31}x_{13}) & \sigma(w_{12}x_{11} + w_{22}x_{12} + w_{32}x_{13}) \\ \sigma(w_{11}x_{21} + w_{21}x_{22} + w_{31}x_{23}) & \sigma(w_{12}x_{21} + w_{22}x_{22} + w_{32}x_{23}) \\ \sigma(w_{11}x_{31} + w_{21}x_{32} + w_{31}x_{33}) & \sigma(w_{12}x_{31} + w_{22}x_{32} + w_{32}x_{33}) \end{bmatrix}$$

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: $$

$$\begin{align}
L = \Lambda \begin{pmatrix} \sigma(XW_{11}) & \sigma(XW_{12}) & \ldots & \sigma(XW_{1p}) \\ \sigma(XW_{21}) & \sigma(XW_{22}) & \ldots & \sigma(XW_{2p}) \\ \vdots & \vdots & & \vdots \\ \sigma(XW_{n1}) & \sigma(XW_{n2}) & \ldots & \sigma(XW_{np}) \end{pmatrix} \tag{1} \\
= \sum_{i=1}^{p} \sum_{j=1}^{n} \sigma(XW_{ij}) \tag{2} \\
= \sigma(XW_{11}) + \sigma XW_{12} + \ldots + \sigma(XW_{np}) \tag{3}
\end{align}$$

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

[11]:
```
lambdaF = lambda matrix : sum(matrix)
lambdaF(S)
```

[11]:

$$\sigma(w_{11}x_{11} + w_{21}x_{12} + w_{31}x_{13}) + \sigma(w_{11}x_{21} + w_{21}x_{22} + w_{31}x_{23}) + \sigma(w_{11}x_{31} + w_{21}x_{32} + w_{31}x_{33}) + \sigma(w_{12}x_{11} + w_{22}x_{12} + w_{32}x_{13}) + \sigma(w_{12}x_{21} + w_{22}x_{22} + w_{32}x_{23}) + \sigma(w_{12}x_{31} + w_{22}x_{32} + w_{32}x_{33})$$

[12]:
```
L = lambdaF(sigmaApply(v(X, W)))
L
#L = lambda mat1, mat2: lambdaF(sigmaApply(v(mat1, mat2)))
#L(X, W)
```

[12]:

$$\sigma(w_{11}x_{11} + w_{21}x_{12} + w_{31}x_{13}) + \sigma(w_{11}x_{21} + w_{21}x_{22} + w_{31}x_{23}) + \sigma(w_{11}x_{31} + w_{21}x_{32} + w_{31}x_{33}) + \sigma(w_{12}x_{11} + w_{22}x_{12} + w_{32}x_{13}) + \sigma(w_{12}x_{21} + w_{22}x_{22} + w_{32}x_{23}) + \sigma(w_{12}x_{31} + w_{22}x_{32} + w_{32}x_{33})$$

[13]:
```
#derive_by_array(L, X)
```

[14]:
```
derive_by_array(L, S)
```

[14]:

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}$$

[15]:
```
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)
```

[15]: 16

[16]:
```
f = Function('f') #(sympify(N[0,0]))
f(N[0,0])
```

[16]:

$$f(w_{11}x_{11} + w_{21}x_{12} + w_{31}x_{13})$$

[17]:
```
f(N[0,0]).diff(X[0,0])
```

[17]:

$$w_{11} \frac{d}{d\xi_1} f(\xi_1) \bigg|_{\xi_1 = w_{11}x_{11} + w_{21}x_{12} + w_{31}x_{13}}$$

[18]:
```
n = v(X,W); n
n11 = Function('{}'.format(n[0,0]))
n11
```

[18]: w_11*x_11 + w_21*x_12 + w_31*x_13

[19]:
```
s_ij = Function('s_ij')
sig = Function('sig')(x)
```

[20]:
```
# 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]))))
```

[21]:
```
#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})
```

[21]:

$$3 * x * y$$

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

[22]:
```
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
```

[22]:

$$\frac{\partial}{\partial \xi_2} F\left(t, \xi_2, \frac{d}{dt} f(t)\right)\Big|_{\xi_2=f(t)}$$

[23]:
```
F(t,U,V)
```

[23]:

$$F\left(t, f(t), \frac{d}{dt} f(t)\right)$$

[24]:
```
F(t,U,V).subs(U,x)
```

[24]:

$$F\left(t, x, \frac{d}{dt} x\right)$$

[25]:
```
F(t,U,V).subs(U,x).diff(x)
```

[25]:

$$\frac{\partial}{\partial \xi_2} F\left(t, \xi_2, \frac{d}{dt} x\right)\Big|_{\xi_2=x}$$

[26]:
```
F(t,U,V).subs(U,x).diff(x).subs(x, U)
```

[26]:

$$\frac{\partial}{\partial \xi_2} F\left(t, \xi_2, \frac{d}{dt} f(t)\right)\Big|_{\xi_2 = f(t)}$$

[27]:

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

[27]:

$$\frac{\partial}{\partial \xi_2} F\left(t, \xi_2, \frac{d}{dt} f(t)\right)\Big|_{\xi_2 = f(t)}$$

[28]:

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

[28]: 6

[29]:

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

[29]: $3 * x * y$

[30]:

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

[30]: 0

[31]:

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

[32]:

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

[32]: $\sigma(w_{11}x_{11} + w_{21}x_{12} + w_{31}x_{13})$

[33]:

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

[34]:

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

[34]:

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}$$

[35]:

```
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()
```

[35]:

$$\frac{d}{df(x)} g(f(x))$$

[36]:

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

[36]:

$$\frac{d}{df(x)} g(f(x))$$

[37]:
```
S = sigmaApply(v(X,W)); S
```

[37]:

$$\begin{bmatrix} \sigma(w_{11}x_{11} + w_{21}x_{12} + w_{31}x_{13}) & \sigma(w_{12}x_{11} + w_{22}x_{12} + w_{32}x_{13}) \\ \sigma(w_{11}x_{21} + w_{21}x_{22} + w_{31}x_{23}) & \sigma(w_{12}x_{21} + w_{22}x_{22} + w_{32}x_{23}) \\ \sigma(w_{11}x_{31} + w_{21}x_{32} + w_{31}x_{33}) & \sigma(w_{12}x_{31} + w_{22}x_{32} + w_{32}x_{33}) \end{bmatrix}$$

[38]:
```
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
```

[38]:

$$\sigma(w_{11*x11+w21*x12+w31*x13})$$

[39]:
```
s.subs({W[0,0]: 14}) # doesn't work to substitute into an undefined function
```

[39]:

$$\sigma(w_{11*x11+w21*x12+w31*x13})$$

[40]:
```
Derivative(s, m(X,W)).doit()
```

[40]:

$$\frac{d}{dw_{11*x11+w21*x12+w31*x13}} \sigma(w_{11*x11+w21*x12+w31*x13})$$

[41]:
```
#s11 = Function('s_{11}')(n11); s11
#sigma(n11).diff(n11)
#s11.diff(n11)
sigma(n11)
```

[41]:

$$\sigma(w_{11}x_{11} + w_{21}x_{12} + w_{31}x_{13})$$

[42]:
```
# ERROR HERE TOO
type(sigma(n11).args[0])
```

[42]:
```
sympy.core.add.Add
```

[43]:
```
type(n11)
```

[43]:
```
sympy.core.add.Add
```

[44]:
```
#sigma(n11).diff(sigma(n11).args[0]) ## ERROR
```

[45]:
```

```

[45]:
```
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})
```

[45]: w_11*x_11 + w_21*x_12 + w_31*x_13

[46]:
```
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)
```

[46]: 0

[47]:
```
# TODO SEEM to have got the expression but it is not working since can't␣
↪substitute anything .... ???
f(xy).diff(xy).subs({x:2})
```

[47]: $\frac{d}{dx*y}f(x*y)$

[48]:
```
Function("x*y")(x,y)
xyf = lambdify([x,y],xy)
xyf(3,4)
f(g(xy)).diff(xy)
#
```

[48]: $\frac{d}{dg(x*y)}f(g(x*y))\frac{d}{dx*y}g(x*y)$

[49]:
```
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
```

[49]: xy

[50]:
```
#derive_by_array(S, N)
```

[51]:
```

```

[51]:
```

```