```
python title="codecell" from sympy import Matrix, Symbol, derive_by_array,
Lambda, Function, MatrixSymbol, Derivative, diff, symbols from
sympy import var from sympy.abc import x, i, j, a, b
```

“‘python title=“codecell” def myvar(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 : myvar(‘x’, i, j)); X

```
```python title="codecell"
W = Matrix(m, p, lambda i,j : myvar('w', i, j)); W
```

```
python title="codecell" A = MatrixSymbol('X',3,3); Matrix(A) B =
MatrixSymbol('W',3,2)
```

“‘python title=“codecell”

```
```python title="codecell"
```

“‘python title=“codecell” v = lambda a,b: a*b

vL = Lambda((a,b), a*b)

n = Function(‘v’) #, Lambda((a,b), a*b))

vN = lambda mat1, mat2: Matrix(mat1.shape[0], mat2.shape[1], lambda i, j: Symbol(“n_{}{}”.format(i+1, j+1))); vN

Nelem = vN(X, W); Nelem

```
```python title="codecell"
n(X,W)
```

```
python title="codecell" n(A,B) python title="codecell" n(X,W).replace(n,
v) # replace works when v = python lambda  python title="codecell"
n(X,W).subs({n: vL}) # subs works when v = sympy lambda     python
title="codecell" n(X,W).replace(n, vL)     python title="codecell"
n(X,W).subs({n: v})# subs() doesn't work when v is python lambda
```

```
python title="codecell" Matrix(n(A,B).subs({n: vL}))
```

```
python title="codecell" #N = v(X, W); N N = n(A,B); N       python
title="codecell" N.replace(n, v) python title="codecell" N.replace(n,
v).subs({A: X, B:W}) # replacing ariable values after doing
function doesn't make the function apply directly on the values
(matrices), need to replace values before the function is replaced,
so that the function can act on them while they are given/alive.
```

```
python title="codecell" N.subs({n: vL, A:X, B:W}) python title="codecell"
Nspec = N.subs({A:X, B:W}).replace(n, v); Nspec
```
“‘python title=“codecell”
```

```python title="codecell"
N.diff(N)
```

python title="codecell" N.diff(X) “‘python title=“codecell”

```python title="codecell"
```

“‘python title=“codecell” # way 2 of declaring S (better way) sigma = Function(‘sigma’)

sigmaApply = Function(“sigma_apply”) #lambda matrix: matrix.applyfunc(sigma)

sigmaApply_ = lambda matrix: matrix.applyfunc(sigma)

sigmaApply(A)

```python title="codecell"
sigmaApply(A).subs({A: X})
```

python title="codecell" sigmaApply_(A)

python title="codecell" sigmaApply(A).subs({A: X}).replace(sigmaApply, sigmaApply_) # NOTE: subs of functions doesn't work, replace actually evaluates the replaced function!

python title="codecell" S = sigmaApply(N); S python title="codecell" Derivative(S, S) python title="codecell" Derivative(S, S).doit() python title="codecell" Derivative(S, n(A,B)).doit() “‘python title=“codecell” #lambd = Function(“lambda”) #Lagain = lambd(sigmaApply(n(A))); Lagain

diff(Lagain, A) # never execute

```python title="codecell"
S.replace(A,X).replace(B,W)
```

python title="codecell" S.replace(n, v) python title="codecell" S.subs({A:X, B:W}).replace(n, v) python title="codecell" Sspec = S.subs({A:X, B:W}).replace(n, v).replace(sigmaApply, sigmaApply_) Sspec python title="codecell" S.replace(n, vN) #.replace(sigmaApply, sigmaApply_) python title="codecell" Selem = S.replace(n, vN).replace(sigmaApply, sigmaApply_); Selem

“‘python title=“codecell” import itertools

elemToSpecD = dict(itertools.chain(*[[(Nelem[i, j], Nspec[i, j]) for j in range(2)]
for i in range(3)]))

elemToSpec = list(elemToSpecD.items())

Matrix(elemToSpec)

```python title="codecell"
elemToSpecFuncD = dict(itertools.chain(*[[(Nelem[i, j], Function("n_{}{}".format(i + 1, j +

elemToSpecFunc = list(elemToSpecFuncD.items())

Matrix(elemToSpecFunc)
```

“‘python title=“codecell” elemToSpecFuncArgsD = dict(itertools.chain(*[[(Nelem[i,
j], Function(“n_{}{}”.format(i + 1, j + 1))(*X*,W)) for j in range(2)] for i in
range(3)]))

elemToSpecFuncArgs = list(elemToSpecFuncArgsD.items())

Matrix(elemToSpecFuncArgs)

```python title="codecell"
Selem

python title="codecell" Selem.subs(elemToSpecD) python title="codecell"
Selem[0,1].diff(Nelem[0,1]) python title="codecell" Selem[0,1].diff(Nelem[0,1]).subs({Nelem[
: Nspec[0,1]}) #Selem[0,1].diff(Nelem[0,1]).subs(dict([{Nelem[0,1]
: Nspec[0,1]}]))

python title="codecell" Selem[0,1].diff(Nelem[0,1]).subs({Nelem[0,1]
: Nspec[0,1]}).subs({Nspec[0,1] : 23})     python title="codecell"
Selem[0,1].diff(Nelem[0,1]).subs({Nelem[0,1] : Nspec[0,1]}).replace(sigma,
lambda x: 8*x**3) python title="codecell" Selem[0,1].diff(Nelem[0,1]).replace(sigma,
lambda x: 8*x**3) python title="codecell" Selem[0,1].diff(Nelem[0,1]).replace(sigma,
lambda x: 8*x**3).doit() python title="codecell" # ### GOT IT: can
replace now with expression and do derivative with respect to
that expression. Selem[0,1].diff(Nelem[0,1]).subs({Nelem[0,1]
: Nspec[0,1]}).replace(sigma, lambda x: 8*x**3).doit()      python
title="codecell" Selem[0,1].subs({Nelem[0,1] : Nspec[0,1]}).diff(X[0,1])#.subs({Nelem[0,1]
: Nspec[0,1]}) python title="codecell" Selem

python title="codecell" nt = Nelem.subs(elemToSpecFunc); nt
python title="codecell" st = Selem.subs(elemToSpecFunc); st
python title="codecell" st.diff(nt)        python title="codecell"
st[0,0].diff(st[0,0].args[0]) python title="codecell" st[0,0].diff(X[0,0])
python title="codecell" st[0,0].diff(st[1,0].args[0])       python
title="codecell" Selem.diff(Nelem) python title="codecell" Selem.diff(Nelem).subs(elemToSpec
```

“‘python title=“codecell” # CAN even replace elements after have done an
operation on them!!! replacing n_21 * 2 with the number 4. Sspec.subs({Nspec[0,
0]: 3}).replace(sigma, lambda x: 2 * x).replace(Nspec[2, 1] * 2, 4)

```python title="codecell"
lambd = Function("lambda")
lambd_ = lambda matrix : sum(matrix)


vN(X, W)
```

python title="codecell" vN(A, B) python title="codecell" L = lambd(S); L python title="codecell" Nelem python title="codecell" L.replace(n, vN) python title="codecell" L.replace(n, vN).replace(sigmaApply, sigmaApply_) python title="codecell" L.replace(n, v) “‘python title=“codecell”

L.replace(n, v).replace(sigmaApply, sigmaApply_)

```python title="codecell"
L.subs({A:X, B:W}).replace(n, vL).replace(sigmaApply, sigmaApply_)
```

python title="codecell" L.replace(n, vN)

“‘python title=“codecell” L.replace(n, vN).subs({A:X, B:W}).replace(sigmaApply, sigmaApply_).replace(lambd, lambd_)

```python title="codecell"
from sympy import symbols, Derivative

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()
```

python title="codecell" # Never do this gives recursion ERROR (max depth exceeded) # h = g(f(A)) # Derivative(h, A).doit()

“‘python title=“codecell”

```python title="codecell"
from sympy.abc import a, b

Llower = lambd(sigmaApply(n(a, b)))
Llower
```

python title="codecell" Derivative(Llower, a).doit() “‘python title=“codecell”

```python title="codecell"
# ### WAY 1: of substituting to differentiate with respect to expression:
n_ij = Function('n_ij')
n_ij(A,B) # (N[0,0]); n_ij
```

python title="codecell" n_ij(A,B).args

“‘python title=“codecell” # sigma(n_ij).diff(n_ij).replace(n_ij, N[0,0]) # ERROR cannot deriv wi.r.t to the expression w11*x11 + ...

sigma(n_ij(A,B)).diff(n_ij(A,B))

```python title="codecell"
sigma(n_ij(*X,*W)).diff(X[0,0])
```

python title="codecell" nab_ij = n_ij(A,B) sigma(nab_ij).diff(nab_ij)#.subs({nab_ij : Nspec[0, 0]}) python title="codecell" sigma(nab_ij).diff(nab_ij).subs({nab_ij : Nspec[2, 1]})

python title="codecell" sigma(nab_ij).diff(nab_ij).subs({nab_ij : Nspec[2,1]}).subs({X[2,1]:77777}) python title="codecell" sigma(nab_ij).diff(nab_ij).subs({nab_ij : 23}) # ERROR if using replace() since it says can't calc derivs w.r.t to the x_11*w_11 + ...

python title="codecell" sigma(nab_ij).diff(nab_ij).subs({nab_ij : Nspec[2,1]}).doit() python title="codecell" sigma(nab_ij).subs({nab_ij : Nspec[2,1]})#.diff(X[2,1]) python title="codecell" # Substituting the value of the function n_ij first, and THEN differentiating with respect to something in that substitution. (X_21) sigma(nab_ij).subs({nab_ij : Nspec[2,1]}).diff(X[2,1])

“‘python title=“codecell” Selem[2,1].subs({Nelem[2,1] : Nspec[2,1]}).diff(X[2,1])

```python title="codecell"
# ### WAY 2:
n_11 = Function('n_11')(Nspec[0, 0]); n_11
```

python title="codecell" sigma(n_11) “‘python title=“codecell” assert Nspec[0,0] == n_11.args[0]

sigma(n_11).subs({n_11 : n_11.args[0]})

```python title="codecell"
sigma(n_11).diff(n_11) #.replace(n_ij, n_ij.args[0])
```

python title="codecell" sigma(n_11).diff(n_11).subs({n_11 : n_11.args[0]}).subs({X[0,0]:77777}) python title="codecell" sigma(n_11).diff(n_11).subs({n_11 : n_11.args[0]}).replace(n_11.args[0], 23) # same as subs in this case

python title="codecell" sigma(n_11).diff(X[0,0]) “‘python title=“codecell” id = Lambda(x, x)

sigma(n_11).diff(X[0,0]).subs({n_11 : id})

```python title="codecell"
# NOTE: so I don't think WAY 2 is correct because here it doesn't simplify the derivative d
```

```
sigma(n_11).diff(X[0,0]).subs({n_11 : Nspec[0,0]})
```

```
python title="codecell" # CORRECT WAY 1 sigma(n_11).subs({n_11 :
Nspec[0,0]}).diff(X[0,0])
```
“‘python title=“codecell” # CORRECT WAY 2

sigma(nab_ij).subs({nab_ij : Nspec[0,0]}).diff(X[0,0])

```
```python title="codecell"
# CORRECT WAY 3
Selem[2,1].subs({Nelem[2,1] : Nspec[2,1]}).diff(X[2,1])
```

```
python title="codecell" sigma(n_11) # WAY 1: sigma argument is
already hardcoded  python title="codecell" sigma(nab_ij) # Way 2:
sigma argument is function of matrixsymbol (better than 1)
```
“‘python title=“codecell” Selem[2,1] # WAY 3: sigma argument is just symbol and we replace it as function with argument hardcoded only later. (better than 2)

```
```python title="codecell"
L
```

“‘python title=“codecell” assert Selem == S.replace(n, vN).replace(sigmaApply, sigmaApply_)

Selem

```
```python title="codecell"
L.replace(n, vN).replace(sigmaApply, sigmaApply_)
```

```
python title="codecell" #L.replace(n, vN).replace(sigmaApply,
sigmaApply_).diff(Nelem[0,0])
```

```
python title="codecell" Lsum = L.replace(n, vN).replace(sigmaApply,
sigmaApply_).replace(lambd, lambd_) Lsum  python title="codecell"
Lsum.diff(Nelem) python title="codecell" Lsum.subs(elemToSpec)#.diff(X[2,1])
python title="codecell" Lsum.subs(elemToSpec).diff(X)
```
“‘python title=“codecell”

specToElemD = {v : k for k, v in elemToSpecD.items()}

Lsum.subs(elemToSpecD).diff(X).subs(specToElemD) “‘
```