Hydrodynamics

February 27, 2025

1 Hydrodynamics of a disk

in a thin film of weakly nematic fluid subject to linear friction, Abdallah Daddi-Moussa-Ider et al

[34]:
```
import sympy as sp
```

1.1 Section 2

For flows at low Reynolds numbers, viscous contributions dominate inertial ones. Under these conditions, the dynam- ics of the surrounding fluid are predominantly governed by the Stokes equation. We here study an extension of this equation to take into account possible friction with the surrounding.

In reality, such friction may arise from a sub- strate and possibly by an additional cover slide confining a thin, basically two-dimensional fluid layer, or, possibly and to some approximation, from the friction with a surrounding fluid. Consequently, the flow equation can be generally expressed as

We attempt to derive the following flow equation,

$$\nabla_j \sigma_{ij}(\mathbf{r}) + \mathbf{M}_{ij}\, v_j(\mathbf{r}) = f_i(\mathbf{r}) \tag{1}$$

1.1.1 Momentum conservation in a fluid

For a fluid of density ρ, the momentum balance is given by

$$\rho\left(\partial_t v_i + v_j\, \nabla_j v_i\right) = \nabla_j \sigma_{ij} + f_i^{\text{ext}}$$

where:

- $v_i(\mathbf{r})$ is the velocity field,
- σ_{ij} is the stress tensor,
- f_i^{ext} represents any external force density.

At low Reynolds numbers, the inertial terms (those involving ρ) become negligible compared to viscous effects.

This simplifies our momentum balance to,

$$\nabla_j \sigma_{ij} + f_i^{\text{ext}} = 0$$

Components and functions depend on both position and time

[35]:
```
# Define symbols for space and time
x, y, t = sp.symbols('x y t')

# Define functions for the velocity field components v1 and v2
v1 = sp.Function('v1')(x, y, t)
v2 = sp.Function('v2')(x, y, t)
# Velocity vector
v = sp.Matrix([v1, v2])
```

```
# Define functions for the external force components f1 and f2 (f^ext)
f1 = sp.Function('f1')(x, y, t)
f2 = sp.Function('f2')(x, y, t)
# External force vector
f_ext = sp.Matrix([f1, f2])

# Define functions for the stress tensor components sigma_ij
sigma11 = sp.Function('sigma11')(x, y, t)
sigma12 = sp.Function('sigma12')(x, y, t)
sigma21 = sp.Function('sigma21')(x, y, t)
sigma22 = sp.Function('sigma22')(x, y, t)
```

The stress tensor is a second-order tensor, i.e., **perhaps treating as a 2×2 matrix in 2D is sufficient?**

[36]:
```
# Build the stress tensor as a 2x2 matrix
sigma_tensor = sp.Matrix([[sigma11, sigma12],
                          [sigma21, sigma22]])
display(sigma_tensor)
```

$$\begin{bmatrix} \sigma_{11}(x,y,t) & \sigma_{12}(x,y,t) \\ \sigma_{21}(x,y,t) & \sigma_{22}(x,y,t) \end{bmatrix}$$

1.1.2 Index notation

Index i:

- This is a free index that specifies the component of the vector equation considered.
- In the term f_i^{ext}, it indicates the i-th component of the external force per unit volume acting on the fluid.

Index j:

- This index is summed over (according to Einstein's summation convention), meaning we sum the contributions from all spatial directions.
- In the term $\nabla_j \sigma_{ij}$, j indicates the coordinate direction along which you take the derivative of the stress tensor component σ_{ij}.
- Essentially, it sums the changes in the stress components in all directions to give the net force per unit volume in the i-th direction.

[37]:
```
# The momentum conservation equation in index notation is:
#   del_j sigma_{ij} + f_i^ext = 0.

# For i=1 (the x-component), this becomes:
eq1 = sp.Eq(sp.diff(sigma11, x) + sp.diff(sigma12, y) + f1, 0)

# For i=2 (the y-component), it is:
eq2 = sp.Eq(sp.diff(sigma21, x) + sp.diff(sigma22, y) + f2, 0)
```

[38]:
```
# Display the equations
print("Momentum conservation (x-component):")
display(eq1)
print("\nMomentum conservation (y-component):")
display(eq2)
```

```
Momentum conservation (x-component):
```

$$f_1(x,y,t) + \frac{\partial}{\partial x}\sigma_{11}(x,y,t) + \frac{\partial}{\partial y}\sigma_{12}(x,y,t) = 0$$

```
Momentum conservation (y-component):
```

$$f_2(x,y,t) + \frac{\partial}{\partial x}\sigma_{21}(x,y,t) + \frac{\partial}{\partial y}\sigma_{22}(x,y,t) = 0$$

1.1.3 Incorporating Frictional Effects

In many realistic situations, such as a thin film confined by a substrate or cover slide, or even friction with a surrounding fluid—there is an additional frictional force. Microscopically, this friction arises because the moving fluid exerts a force on the substrate, which in turn reacts with a force proportional to the local velocity. To model this effect, we introduce a friction term:

$$\text{Friction force per unit area} = \mathbf{M}_{ij}\, v_j$$

where $\mathbf{M}_{ij}$ is a friction tensor. For isotropic friction, this might simply be $\Gamma\,\delta_{ij}$ (with Γ being a friction coefficient), but in a more general case the friction can be anisotropic.

[39]:
```
# Define the friction tensor M_ij as a 2x2 matrix.
# For the most general case, we allow each component to be a function.
M11 = sp.Function('M11')(x, y, t)
M12 = sp.Function('M12')(x, y, t)
M21 = sp.Function('M21')(x, y, t)
M22 = sp.Function('M22')(x, y, t)

M_tensor = sp.Matrix([[M11, M12],
                      [M21, M22]])

print("Friction tensor")
display(M_tensor)

# Alternatively, if friction is isotropic, we might use a constant friction␣
 ↪coefficient Gamma:
Gamma = sp.symbols('Gamma', real=True, positive=True)
M_constant = Gamma * sp.eye(2)
print("\nIsotropic friction tensor")
display(M_constant)
```

Friction tensor

$$\begin{bmatrix} M_{11}(x,y,t) & M_{12}(x,y,t) \\ M_{21}(x,y,t) & M_{22}(x,y,t) \end{bmatrix}$$

Isotropic friction tensor

$$\begin{bmatrix} \Gamma & 0 \\ 0 & \Gamma \end{bmatrix}$$

Adding either of these friction terms to our force balance, we have

$$\nabla_j \sigma_{ij} + \mathbf{M}_{ij}\, v_j + f_i^{\text{ext}} = 0$$

[40]:
```
# Compute the divergence of the stress tensor as a vector:
# (dee_x sigma_11 + dee_y sigma_12,  dee_x sigma_21 + dee_y sigma_22)
div_sigma = sp.Matrix([
    sp.diff(sigma11, x) + sp.diff(sigma12, y),
    sp.diff(sigma21, x) + sp.diff(sigma22, y)
])

# Calculate the friction force per unit area as M_ij * v_j (matrix-vector␣
 ↪multiplication)
friction_force = M_tensor * v

# Combine them into the momentum conservation equation: div(sigma) + f_ext = 0
momentum_eq = sp.Eq(div_sigma + friction_force + f_ext, sp.Matrix([0, 0]))
display(momentum_eq)
```

$$\begin{bmatrix} M_{11}(x,y,t)v_1(x,y,t) + M_{12}(x,y,t)v_2(x,y,t) + f_1(x,y,t) + \frac{\partial}{\partial x}\sigma_{11}(x,y,t) + \frac{\partial}{\partial y}\sigma_{12}(x,y,t) \\ M_{21}(x,y,t)v_1(x,y,t) + M_{22}(x,y,t)v_2(x,y,t) + f_2(x,y,t) + \frac{\partial}{\partial x}\sigma_{21}(x,y,t) + \frac{\partial}{\partial y}\sigma_{22}(x,y,t) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

If we now define the net external force as

$$f_i(\mathbf{r}) \equiv -f_i^{\text{ext}}$$

the equation becomes

$$\nabla_j\, \sigma_{ij}(\mathbf{r}) + \mathbf{M}_{ij}\, v_j(\mathbf{r}) = f_i(\mathbf{r})$$

which is the form of equation (1).

($f_i(\mathbf{r})$ in two dimensions signifies a force density acting on the fluid, and summation over repeated indices is implied following Einstein's notation).

1.1.4 Incorporating the total stress tensor $\sigma_{ij}(\mathbf{r})$

Assuming $i, j \in \{x, y\}$, equation (1),

$$\sigma_{ij}(\mathbf{r}) = p(\mathbf{r})\,\delta_{ij} + \tilde{\sigma}_{ij}(\mathbf{r})$$

represents the components of the stress tensor where,

- $p(\mathbf{r})$ is the pressure field
- The Kronecker delta $\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$

Dead End - Including for fidelity To have the isotropic part carry the same overall weight as the full stress, we equate,

$$2p = \sigma_{xx} + \sigma_{yy}$$

NOTE: Here we see a variation in sign convention in the paper - Ultimately this doesn't appear to matter

[41]:
```
# Trace of sigma
trace_sigma = sigma_tensor.trace()

p = sp.Rational(1,2)*trace_sigma
I2 = sp.eye(2)

# Decompose:
# note the plus sign because p was defined with a minus
sigma_tilde = sigma_tensor.copy() - p*I2
# Then sigma = p I + sigma_tilde

display(sigma_tilde)
print(f"Type: {type(sigma_tilde)}")
assert sigma_tilde.trace() == 0, "isotropic pressure not separated as required"
```

$$\begin{bmatrix} \frac{\sigma_{11}(x,y,t)}{2} - \frac{\sigma_{22}(x,y,t)}{2} & \sigma_{12}(x,y,t) \\ \sigma_{21}(x,y,t) & -\frac{\sigma_{11}(x,y,t)}{2} + \frac{\sigma_{22}(x,y,t)}{2} \end{bmatrix}$$

```
Type: <class 'sympy.matrices.dense.MutableDenseMatrix'>
```

$\tilde{\sigma}_{ij}$ is traceless meaning the isotropic pressure part has been successfully separated

We substitute component wise into the `momentum_eq`

[42]:
```
momentum_eq_subst = momentum_eq.subs({
    sigma11: sigma_tilde[0,0],
    sigma12: sigma_tilde[0,1],
    sigma21: sigma_tilde[1,0],
    sigma22: sigma_tilde[1,1]
})
display(momentum_eq_subst)
```

$$\begin{bmatrix} M_{11}(x,y,t)v_1(x,y,t) + M_{12}(x,y,t)v_2(x,y,t) + f_1(x,y,t) + \frac{\partial}{\partial x}\left(\frac{3\sigma_{11}(x,y,t)}{4} - \frac{\sigma_{22}(x,y,t)}{4}\right) + \frac{\partial}{\partial y}\sigma_{12}(x,y,t) \\ M_{21}(x,y,t)v_1(x,y,t) + M_{22}(x,y,t)v_2(x,y,t) + f_2(x,y,t) + \frac{\partial}{\partial y}\left(-\frac{\sigma_{11}(x,y,t)}{2} + \frac{\sigma_{22}(x,y,t)}{2}\right) + \frac{\partial}{\partial x}\sigma_{21}(x,y,t) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

The $\frac{\partial}{\partial x}\sigma_{11}$ and $\frac{\partial}{\partial y}\sigma_{22}$ coefficients are wrong

The problem here is our definition of the derived stress,

$$\tilde{\sigma}_{ij} = \sigma_{ij} - p\,\delta_{ij}, \quad \text{with } p = \frac{1}{2}\text{tr}(\sigma)$$

creates circular dependency. The pressure p is defined in terms of the very components σ_{11} and σ_{22} that we try to substitute out in favour of $\tilde{\sigma}_{ij}$.

Amendment Instead of computing p from the trace of σ, we define it as a function

[43]:
```
# Define pressure field as an independent function
p_field = sp.Function('p')(x, y, t)
```

Define the independent components functions for the deviatoric stress

[44]:
```
# We define three functions: tilde_sigma_11, tilde_sigma_12, tilde_sigma_21,
# and set tilde_sigma_22 to enforce tracelessness.
sigmatilde_11 = sp.Function('sigmatilde_11')(x, y, t)
sigmatilde_12 = sp.Function('sigmatilde_12')(x, y, t)
sigmatilde_21 = sp.Function('sigmatilde_21')(x, y, t)
sigmatilde_22 = -sigmatilde_11  # Enforce tilde_sigma.trace() = 0

# Build the deviatoric stress tensor using the tilde notation
sigma_tilde = sp.Matrix([
    [sigmatilde_11, sigmatilde_12],
    [sigmatilde_21, sigmatilde_22]
])
```

Reconstruct the full stress tensor

[32]:
```
I2 = sp.eye(2)
sigma_tensor_p = p_field * I2 + sigma_tilde
display(sigma_tensor_p)
```

$$\begin{bmatrix} p(x,y,t) + \tilde{\sigma}_{11}(x,y,t) & \tilde{\sigma}_{12}(x,y,t) \\ \tilde{\sigma}_{21}(x,y,t) & p(x,y,t) - \tilde{\sigma}_{11}(x,y,t) \end{bmatrix}$$

[46]:
```
# Recompute p from the full stress tensor:
p_from_sigma = sp.Rational(1, 2) * sigma_tensor_p.trace()
assert sp.simplify(p_from_sigma - p_field) == 0, "Pressure mismatch!"

# Now, compute sigma - p*I2:
sigma_decomp = sp.simplify(sigma_tensor_p - p_from_sigma * I2)
```

```
assert sp.simplify(sigma_decomp - sigma_tilde) == sp.zeros(2), "Decomposition␣
 ↪does not hold!"
```

Now we can substitute component wise into the `momentum_eq`

[47]:
```
momentum_eq_tilde = momentum_eq.subs({
    sigma11: sigma_tensor_p[0,0],
    sigma12: sigma_tensor_p[0,1],
    sigma21: sigma_tensor_p[1,0],
    sigma22: sigma_tensor_p[1,1]
})
display(momentum_eq_tilde)
```

$$\begin{bmatrix} M_{11}(x,y,t)v_1(x,y,t) + M_{12}(x,y,t)v_2(x,y,t) + f_1(x,y,t) + \frac{\partial}{\partial x}\left(p(x,y,t) + \tilde{\sigma}_{11}(x,y,t)\right) + \frac{\partial}{\partial y}\tilde{\sigma}_{12}(x,y,t) \\ M_{21}(x,y,t)v_1(x,y,t) + M_{22}(x,y,t)v_2(x,y,t) + f_2(x,y,t) + \frac{\partial}{\partial y}\left(p(x,y,t) - \tilde{\sigma}_{11}(x,y,t)\right) + \frac{\partial}{\partial x}\tilde{\sigma}_{21}(x,y,t) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

1.1.5 4 Simplify stress tensor to 'pure dissipative stress'

We now need to show that,

$$\tilde{\sigma}_{ij} = \sigma^{\mathrm{D}}_{ij} = -\nu_{ijkl}\nabla_l\, v_k \tag{2}$$

`NOTE`: assuming the D subscript means " dissipative stress - *further definition required* "

`NOTE`: assumed that the expression ν_{ijkl} is " some rank-4 tensor "

In the Appendix of `Reference 80`: "Pleiner H and Brand H R 1996 Hydrodynamics and electrohydrodynamics of liquid crystals Pattern Formation in Liquid Crystals ed A Buka and L Kramer (Springer) p 15–67" I found,

> while the fourth rank viscosity tensor contains five (dissipative) viscosities [181], [182]
>
> $$\begin{aligned} \nu_{ijkl} =& \nu_2\left(\delta_{jl}\delta_{ik} + \delta_{il}\delta_{jk}\right) + 2(\nu_1 + \nu_2 - 2\nu_3)\, n_i n_j n_k n_l \\ &+(\nu_3 - \nu_2)(n_j n_l \delta_{ik} + n_j n_k \delta_{il} + n_i n_k \delta_{jl} + n_i n_l \delta_{jk}) \\ &+(\nu_4 - \nu_2)\,\delta_{ij}\delta_{kl} + (\nu_5 - \nu_4 + \nu_2)(\delta_{ij} n_k n_l + \delta_{kl} n_i n_j) \end{aligned} \tag{A.15}$$

Which is equation (3), simplified by equation (4). Why do we have 5 viscosity components?

- "The paper by Pleiner and Brand demonstrates that 5 are sufficient".
- 5 distinct viscosities defines an anisotropic fluid medium.
- We examine scenario of the director being consistently and uniformly orientated along a single global axis.

1.1.6 4b Derivation

We build ν_{ijkl} as a linear combination of all distinct tensor contractions allowed by the uniaxial symmetry around some director $\mathbf{n} = \begin{pmatrix} n_x & n_y \end{pmatrix}^T$

`Note`: we probably need to normalise as a unit vector

[48]:
```
n_x, n_y = sp.symbols('n_x n_y', real=True)
# Optionally enforce n_x^2 + n_y^2 = 1:
# constraints = [sp.Eq(n_x**2 + n_y**2, 1)]
n = (n_x, n_y)
```

Define multiple viscosity coefficients

[49]:
```
# Five viscosity parameters
nu1, nu2, nu3, nu4, nu5 = sp.symbols('nu1 nu2 nu3 nu4 nu5', real=True)
```

Define a Kronecker delta for 2D

[50]:
```
def kronecker(i, j):
    return 1 if i == j else 0
```

1.1.7 Einstein summation in the tensor construction

We have a tensor expression of the form,

$$\nu_{ijkl} = \nu_2, \left(\delta_{ik}, \delta_{jl} + \delta_{il}, \delta_{jk}\right) + [\dots], \delta_{ik}, n_k, n_l + \dots$$

In this expression, the term $\delta_{ik}\, n_k$ has a repeated index k. By the Einstein summation convention, a repeated index implies a sum over that index. That is,

$$\delta_{ik}, n_k = \sum_{k=0}^{1} \delta_{ik}, n_k = n_i, .$$

Thus, whenever we see $\delta_{ik}\, n_k$, it simplifies to n_i, meaning it selects the ith component of the vector $\mathbf{n}$

[53]:
```
nu_tensor = {}
for i in range(2):
    for j in range(2):
        for k in range(2):
            for l in range(2):
                val = 0

                # 1) "term1": Isotropic-like part
                #    nu2 * (delta_{ik} delta_{jl} + delta_{il} delta_{jk})
                val += nu2 * (
                    kronecker(i,k)*kronecker(j,l) +
                    kronecker(i,l)*kronecker(j,k)
```

```
                )

                # 2) "term2": bar_nu * n[i] n[j] n[k] n[l]
                bar_nu = 2*(nu1 + nu2 - 2*nu3)
                val += bar_nu * n[i]*n[j]*n[k]*n[l]

                # 3) "term3": (nu4 - nu2)* (delta_{ij} delta_{kl}), etc.
                #    Just an example if your reference has that piece:
                val += (nu4 - nu2)*(
                    kronecker(i,j)*kronecker(k,l)
                )

                # 4) "term4": (nu3 - nu2)*[n[i] n[k] delta(j,l) + ...]
                #    No repeated indices here, so we can code it directly:
                val += (nu3 - nu2)*(
                    n[i]*n[k]*kronecker(j,l)
                  + n[i]*n[l]*kronecker(j,k)
                  + n[j]*n[k]*kronecker(i,l)
                  + n[j]*n[l]*kronecker(i,k)
                )

                # 5) "term5": (nu5 - nu4 + nu2)* (delta_{i,k} n_k n_l +␣
↪delta_{k,l} n_i n_j + ...)
                #    If the reference has something like delta_{i,k} n_k => n_i,
                #    we do that by summation over a dummy index "a".
                sum_over_a = 0
                for a in range(2):
                    sum_over_a += kronecker(i,a)*n[a]  # => n_i
                sum_over_b = 0
                for b in range(2):
                    sum_over_b += kronecker(k,b)*n[b]  # => n_k
                sum_over_c = 0
                for c in range(2):
                    sum_over_c += kronecker(l,c)*n[c]  # => n_l

                # Then combine them carefully.
                val += (nu5 - nu4 + nu2)*(
                    sum_over_a * sum_over_c
                    + kronecker(l,k)*n[i]*n[j]
                )

                # Save
                nu_tensor[(i,j,k,l)] = sp.simplify(val)
```

[54]:
```
dv = sp.zeros(2,2)
# Indices: j=0 -> derivative w.r.t x, j=1 -> derivative w.r.t y
#          k=0 -> v1, k=1 -> v2
```

```
dv[0,0] = v1.diff(x)  # dee(v1)/x
dv[0,1] = v2.diff(x)  # dee(v2)/x
dv[1,0] = v1.diff(y)  # dee(v1)/y
dv[1,1] = v2.diff(y)  # dee(v2)/y

print("Velocity gradients dv")
display(dv)
```

```
Velocity gradients dv
```

$$\begin{bmatrix} \frac{\partial}{\partial x} v_1(x,y,t) & \frac{\partial}{\partial x} v_2(x,y,t) \\ \frac{\partial}{\partial y} v_1(x,y,t) & \frac{\partial}{\partial y} v_2(x,y,t) \end{bmatrix}$$

Once we have ν_{ijkl}, the dissipative (deviatoric) stress $\sigma^{\mathrm{D}}_{i\ell}$ is given by

$$\sigma^{\mathrm{D}}_{i\ell} = -\nu_{ijk\ell}\, \nabla_j v_k$$

[55]:
```
# Suppose dv is a 2x2 matrix: dv[j,k] = partial_j(v_k)
sigmaD = sp.zeros(2,2)
for i in range(2):
    for ell in range(2):
        expr = 0
        for j2 in range(2):
            for k2 in range(2):
                expr += - nu_tensor[(i,j2,k2,ell)] * dv[j2, k2]
        sigmaD[i, ell] = expr
```

[59]:
```
print("Dissipative Stress Tensor Components:")
for i in range(2):
    for j in range(2):
        # Construct a label for the (i,j) component using standard notation
        label = sp.symbols(f"\\tilde{{\\sigma}}^D_{{{i+1}{j+1}}}")
        eq = sp.Eq(label, sigmaD[i,j])
        display(eq)
```

```
Dissipative Stress Tensor Components:
```

$$\tilde{\sigma}^D_{11} = -n_x \left(2n_x^2 n_y \left(\nu_1+\nu_2-2\nu_3\right)+n_x\left(\nu_2-\nu_4+\nu_5\right)-2n_y\left(\nu_2-\nu_3\right)\right)\frac{\partial}{\partial x}v_2(x,y,t) -$$
$$n_x\left(2n_x^2 n_y\left(\nu_1+\nu_2-2\nu_3\right)-2n_y\left(\nu_2-\nu_3\right)+\left(n_x+n_y\right)\left(\nu_2-\nu_4+\nu_5\right)\right)\frac{\partial}{\partial y}v_1(x,y,t) +$$
$$\left(-2n_x^2 n_y^2\left(\nu_1+\nu_2-2\nu_3\right)-n_x^2\left(\nu_2-\nu_4+\nu_5\right)-\nu_2+\left(n_x^2+n_y^2\right)\left(\nu_2-\nu_3\right)\right)\frac{\partial}{\partial y}v_2(x,y,t) +$$
$$\left(-2n_x^4\left(\nu_1+\nu_2-2\nu_3\right)-4n_x^2\left(-\nu_2+\nu_3\right)-2n_x^2\left(\nu_2-\nu_4+\nu_5\right)-\nu_2-\nu_4\right)\frac{\partial}{\partial x}v_1(x,y,t)$$

$$\tilde{\sigma}^D_{12} = -2n_x n_y\left(n_y^2\left(\nu_1+\nu_2-2\nu_3\right)+\nu_3-\nu_4+\nu_5\right)\frac{\partial}{\partial y}v_2(x,y,t) -$$
$$n_x n_y\left(2n_x^2\left(\nu_1+\nu_2-2\nu_3\right)-\nu_2+2\nu_3-\nu_4+\nu_5\right)\frac{\partial}{\partial x}v_1(x,y,t)+\left(-2n_x^2 n_y^2\left(\nu_1+\nu_2-2\nu_3\right)-n_x n_y\left(\nu_2-\nu_4+\nu_5\right)-\nu_2\right.$$

$$\left(-2n_x^2n_y^2\left(\nu_1+\nu_2-2\nu_3\right)-n_x\left(n_x+n_y\right)\left(\nu_2-\nu_4+\nu_5\right)+\nu_2-\nu_4\right)\frac{\partial}{\partial x}v_2(x,y,t)$$

$$\tilde{\sigma}_{21}^D = -2n_xn_y\left(n_x^2\left(\nu_1+\nu_2-2\nu_3\right)+\nu_3-\nu_4+\nu_5\right)\frac{\partial}{\partial x}v_1(x,y,t) -$$

$$n_xn_y\left(2n_y^2\left(\nu_1+\nu_2-2\nu_3\right)-\nu_2+2\nu_3-\nu_4+\nu_5\right)\frac{\partial}{\partial y}v_2(x,y,t)+\left(-2n_x^2n_y^2\left(\nu_1+\nu_2-2\nu_3\right)-n_xn_y\left(\nu_2-\nu_4+\nu_5\right)-\nu_2\right.$$

$$\left(-2n_x^2n_y^2\left(\nu_1+\nu_2-2\nu_3\right)-n_y\left(n_x+n_y\right)\left(\nu_2-\nu_4+\nu_5\right)+\nu_2-\nu_4\right)\frac{\partial}{\partial y}v_1(x,y,t)$$

$$\tilde{\sigma}_{22}^D = -n_y\left(2n_xn_y^2\left(\nu_1+\nu_2-2\nu_3\right)-2n_x\left(\nu_2-\nu_3\right)+n_y\left(\nu_2-\nu_4+\nu_5\right)\right)\frac{\partial}{\partial y}v_1(x,y,t) -$$

$$n_y\left(2n_xn_y^2\left(\nu_1+\nu_2-2\nu_3\right)-2n_x\left(\nu_2-\nu_3\right)+\left(n_x+n_y\right)\left(\nu_2-\nu_4+\nu_5\right)\right)\frac{\partial}{\partial x}v_2(x,y,t) +$$

$$\left(-2n_x^2n_y^2\left(\nu_1+\nu_2-2\nu_3\right)-n_y^2\left(\nu_2-\nu_4+\nu_5\right)-\nu_2+\left(n_x^2+n_y^2\right)\left(\nu_2-\nu_3\right)\right)\frac{\partial}{\partial x}v_1(x,y,t) +$$

$$\left(-2n_y^4\left(\nu_1+\nu_2-2\nu_3\right)-4n_y^2\left(-\nu_2+\nu_3\right)-2n_y^2\left(\nu_2-\nu_4+\nu_5\right)-\nu_2-\nu_4\right)\frac{\partial}{\partial y}v_2(x,y,t)$$

Which are fairly monstrous components…

1.1.8 5 Simplification by continuity equation

We incorporate the common assumption made in similar analyses of fluid flows, namely, local volume conservation and constant density of the fluid. As a result, the continuity equation simplifies to

$$\nabla\cdot\mathbf{v}=0 \tag{5}$$

Under the given circumstances, the director $\hat{\mathbf{n}}$ introduces uniaxial anisotropy to the viscosity tensor. Without loss of generality, we choose $\hat{\mathbf{n}}\parallel\hat{\mathbf{x}} \implies \hat{\mathbf{n}}=(1,0)$.

Effect on the Viscosity Tensor

In our uniaxial tensor ν_{ijkl}, any factor like `n[i]` or `n[j]` becomes zero unless $i=0$ (the "x" index). Concretely;

- `n[0]` $=1$
- `n[1]` $=0$

So any product `n[i]*n[j]*...` will vanish unless $i=j=0$. Concretely: - `n[i]*n[j]` $=1$ only if $i=j=0$; otherwise 0. - `n[i]*n[j]*n[k]*n[l]` is 1 only if $i=j=k=l=0$; else 0.

Incompressibility

Using the original notation we set (which will be amended later) we have,

$$v_1\equiv v_x,\quad v_2\equiv v_y$$

Therefore

$$\frac{\partial v_1}{\partial x}=-\frac{\partial v_2}{\partial y}$$

[60]:
```
final_sigmaD = sigmaD.subs({n_x: 1, n_y: 0, dv[0,0]: -dv[1,1]})
final_sigmaD_simpl = sp.simplify(final_sigmaD)

print("Simplified Dissipative Stress Tensor Components:")
for i in range(2):
    for j in range(2):
        # Construct a label for the (i,j) component using standard notation
        label = sp.symbols(f"\\tilde{{\\sigma}}^D_{{{i+1}{j+1}}}")
        eq = sp.Eq(label, final_sigmaD_simpl[i,j])
        display(eq)
```

```
Simplified Dissipative Stress Tensor Components:
```

$$\tilde{\sigma}^D_{11} = -(\nu_2 - \nu_4 + \nu_5)\frac{\partial}{\partial y}v_1(x,y,t) - (\nu_2 - \nu_4 + \nu_5)\frac{\partial}{\partial x}v_2(x,y,t) + (2\nu_1 + \nu_2 - \nu_4 + 2\nu_5)\frac{\partial}{\partial y}v_2(x,y,t) - (\nu_2 + \nu_3 - \nu_4 + \nu_5)\frac{\partial}{\partial y}v_2(x,y,t)$$

$$\tilde{\sigma}^D_{12} = -\nu_3\frac{\partial}{\partial y}v_1(x,y,t) - \nu_5\frac{\partial}{\partial x}v_2(x,y,t)$$

$$\tilde{\sigma}^D_{21} = -\nu_3\frac{\partial}{\partial x}v_2(x,y,t) + (\nu_2 - \nu_4)\frac{\partial}{\partial y}v_1(x,y,t)$$

$$\tilde{\sigma}^D_{22} = (-\nu_2 + \nu_3 - \nu_4)\frac{\partial}{\partial y}v_2(x,y,t)$$