1 Definition of metrics

1.1 Distance metric

The essential features of the distance metric for swarms, presented in (Eliot et al. 2018), can be summarised as follows:

$$\psi_d(S) = \mu_d(S) \pm \sigma_d(S)$$

where $\mu_d(S)$ is the mean distance over all agents $b \in S$, between b and its cohesion neighbours, given by:

$$\mu_d(S) = \frac{\sum_{b\in S} \sum_{b' \in n_c(b)} \|\vec{bb'}\|}{\sum_{b\in S} \left|n_c(b)\right|}$$

and $\sigma_d(S)$ is the standard deviation from the mean:

$$\sigma_d(S) = \sqrt{\frac{\sum_{b\in S} \sum_{b' \in n_c(b)} \left(\|\vec{bb'}\| - \mu_d(S)\right)^2}{\sum_{b\in S} \left|n_c(b)\right|}}$$

1.2 Cohesion/repulsion metric

The essential features of the cohesion/repulsion metric for swarms, adapted from (Eliot et al. 2018), can be summarised as follows:

$$\psi_p(S) = \mu_p(S) \pm \sigma_p(S)$$

where $\mu_p(S)$ is the mean of the adjusted magnitude values of the weighted cohesion/repulsion vectors of all agents, induced by their cohesion/repulsion neighbours. For each cohesion/repulsion vector, a positive value is derived from the magnitude of the vector if the cohesion component of the vector dominates, but a negative value is derived if the repulsion component dominates.

We define some helper functions, v_{cr} and P, to aid the specifications of μ_p and σ_p:

$$v_{cr}(b) = k_c v_c(b) + k_r v_r(b)$$

$$P(b) = \begin{cases} \|v_{cr}(b)\| & k_c v_c(b) > k_r v_r(b) \\ -\|v_{cr}(b)\| & \text{otherwise} \end{cases}$$

$v_{cr}(b)$ gives the weighted cohesion/repulsion vector for b and $P(b)$ gives the value derived from the magnitude of this vector. Now we can define the mean and standard deviation.

$$\mu_p(S) = \frac{\sum_{b\in S} P(b)}{D}$$

and $\sigma_p(S)$ is the standard deviation from the mean:

$$\sigma_p(S) = \sqrt{\frac{\sum_{b \in S} (P(b) - \mu_p(S))^2}{D}}$$

We still need to consider the definition of the denominator, D, here. (Eliot et al. 2018) defines D like this:

$$D = \sum_{b \in S} \left| n_c(b) \right| + \left| n_r(b) \right|$$

This seems to me to be over-counting agents. Remember that each agent $b \in S$ has at most one cohesion/repulsion vector as defined above. This has been scaled already by the reciprocal of the number of its cohesion and repulsion neighbours (see the definitions of $v_c(b)$ and $v_r(b)$ earlier). In calculating $\mu_p(S)$ and $\sigma_p(S)$, we should be dividing by at most $|S|$ but the value of D, as defined above, may be as big as $2(|S|^2 - |S|)$, clearly too big!. It might be argued that even $|S|$ may be too big, since S may include agents that are isolated and not participating in the cohesion/repulsion structure of the swarm and, therefore, should not be counted. In this case, we could define D as

$$D = \left| \bigcup_{b \in S} (n_c(b) \cup n_r(b)) \right|$$

I think it's reasonable to consider that the cohesion/repulsion structure is a property of the whole swarm S, whether or not it contains isolated agents, and, in the following, I take D to be

$$D = |S|$$

2 Implementation of metrics

2.1 Distance metric

```
@jit(nopython=True, fastmath=True)
def mu_sigma_d(mag, ecb):
    n_agents = mag.shape[0]
    msum = 0; msum_sq = 0; nsum = 0
    for i in prange(n_agents):
        for j in range(i):
            if mag[j, i] <= ecb[j, i]:
                msum += mag[j, i]
                msum_sq += mag[j, i] **2
                nsum += 1
            if mag[i, j] <= ecb[i, j]:
                msum += mag[i, j]
                msum_sq += mag[i, j] **2
                nsum += 1
```

```
    mu_d = msum / nsum
    mu_d_sq = msum_sq / nsum
    var_d = mu_d_sq - mu_d ** 2
    sigma_d = np.sqrt(var_d)
    return mu_d, sigma_d
```

2.2 Cohesion/repulsion metric

```
def mu_sigma_p(b):
    vcr_x = b[COH_X] + b[REP_X]                                # the weighted cohesion/
    vcr_y = b[COH_Y] + b[REP_Y]
    vcr_mag = np.hypot(vcr_x, vcr_y)                           # the magnitude of the w
    vc_mag = np.hypot(b[COH_X], b[COH_Y])                      # the magnitude of the c
    vr_mag = np.hypot(b[REP_X], b[REP_Y])                      # the magnitude of the r
    P = np.where(vc_mag > vr_mag, vcr_mag, -vcr_mag)           # the implementation of
    n_agents = b.shape[1]                                      # the total number of ag
    mu_p = np.sum(P) / n_agents                                # the mean
    sigma_p = np.sqrt(np.sum((P - mu_p) ** 2) / n_agents)      # the standard deviation
    return mu_p, sigma_p
```

3 Examples

3.1 Distance metric

```
b, step_args = load_swarm()
n_steps = 400                                                          # set the nu
step_ids = [i for i in range(n_steps)]                                 # create a l
mu = []                                                                # create a l
sigma = []                                                             # create a l
for i in range(n_steps):
    xv,yv,mag,ang,ecf,erf,ekc,ekr = compute_step(b, **step_args)    # take a step
    m, s = mu_sigma_d(mag, ecf)                                        # compute th
    mu += [m]                                                          # add to lis
    sigma += [s]
    apply_step(b)
step_ids = np.array(step_ids)                                          # convert li
mu = np.array(mu)
sigma = np.array(sigma)
fig, ax = plt.subplots(figsize=(4,4))                                  # create a g
ax.set(xlim=(0, n_steps), ylim=(0, 4))                              # set the limit
ax.set_title('Distance metric (not perimeter-directed, stability_factor=0.0)')
ax.set_xlabel('Simulation step number')
ax.set_ylabel('$\psi_d(S)$')
ax.grid(True)                                                          # show a gri
ax.plot(step_ids, mu, 'k-')                                            # plot the m
```

```
ax.fill_between(step_ids, mu + sigma, mu - sigma, facecolor='red', alpha=0.5)   # plot th
```

3.2 Cohesion/repulsion metric

```
b, step_args = load_swarm()
n_steps = 400                                                                         #
step_ids = [i for i in range(n_steps)]
mu_p = []
sigma_p = []
for i in range(n_steps):
    compute_step(b, **step_args)
    m, s = mu_sigma_p(b)
    mu_p += [m]
    sigma_p += [s]
    apply_step(b)
step_ids = np.array(step_ids)
mu_p = np.array(mu_p)
sigma_p = np.array(sigma_p)
fig, ax = plt.subplots(figsize=(4,4))
ax.set(xlim=(0, n_steps), ylim=(-20, 0))
ax.set_title('Cohesion/repulsion metric (not perimeter-directed, stability_factor=0.0)')
ax.set_xlabel('Simulation step number')
ax.set_ylabel('$\psi_P(S)$')
ax.grid(True)
ax.plot(step_ids, mu_p, 'k-')
ax.fill_between(step_ids, mu_p + sigma_p, mu_p - sigma_p, facecolor='blue', alpha=0.5)
```