BRIEF DESCRIPTION OF MULTI-SCALE LOCALIZATION ALGORITHM

TRAVIS HARTY

1. Overview

The purpose of this algorithm is to construct the matrices $\mathbf{U}$, $\mathbf{\Sigma}$, and $\mathbf{V}$ such that

$$\mathbf{U\Sigma V} = \mathbf{R}^{-1/2}\mathbf{H}\mathbf{P}^{1/2}$$

is a singular value decomposition where $\mathbf{P}$ is the background covariance matrix, $\mathbf{R}$ is the observation error covariance matrix, and $\mathbf{H}$ is a forward observation matrix. We will assume we have perfect knowledge of $\mathbf{R}$ and for simplicity will take $\mathbf{R}$ to be equal to the identity matrix.

We begin with an ensemble, X, of N_e samples drawn from the distribution $N(0, \mathbf{P})$ so that $\dim X = n \times N_e$ with n being the state space dimension. In order to estimate errors in our leading singular vectors, we require an ensemble of ensembles. In order to generate an ensemble of ensembles, we resample our original ensemble with replacement as is done with the bootstrap method. We can do this an arbitrary number of times so that we will end with N_{ee} ensembles.

This ensemble of ensembles allows us to generate an ensemble of sample covariance matrices. We can then localize each of these covariance matrices and calculate which localization radius minimizes the difference between the leading right singular vectors. By difference between the leading singular vectors, we mean the average angle between all the ensemble members is minimized.

After this optimal localization length is determined, the process is repeated this time for P which has all previous optimal leading right singular vectors in it's null space. This can be repeated until all desired singular vectors are determined.

2. Details

(1) $\mathbf{X}$ is an ensemble of dimension $n \times N_e$ drawn from $N(0, \mathbf{P})$ with ensemble size N_e and state dimension of n.
(2) Resample $\mathbf{X}$ with replacement N_{ee} times to generate N_{ee} ensembles each of dimension $n \times N_e$. Call these resampled ensembles $\tilde{\mathbf{X}}_i$.
(3) Generate an ensemble of sample covariance matrices $\tilde{\mathbf{P}}_i$. After these covariance matrices are generated, the $\tilde{\mathbf{X}}_i$'s are no longer needed.
(4) For every length scale in a set of reasonable length scales, generate a localization matrix $\mathbf{L}$ and localize each of the $\tilde{\mathbf{P}}_i$'s. Call these localized sample covariance matrices $\tilde{\mathbf{P}}_i^{loc} = \mathbf{L} \circ \tilde{\mathbf{P}}_i$.

(5) For each $\tilde{\mathbf{P}}_i^{loc}$ calculate the leading m right singular vectors V_i of $\mathbf{H}\left(\tilde{\mathbf{P}}_i^{loc}\right)^{1/2}$
(6) Calculate the angle between the spans of each pair of V_i's. By angle, we mean $\arccos(\sigma_m)$ where σ_m is the smallest singular value of $V_i^T V_j$, $i \neq j$.
(7) Find the localization length which minimizes the average angle between all of the ensemble members.
(8) For each of the N_{ee} ensembles, calculate the projection matrix $\tilde{\mathbf{P}}_i^{\mathbf{roj}} = \mathbf{I} - V_i V_i^T$ corresponding to this optimal length scale.
(9) Similarly, calculate and save the leading m singular values and vectors of $\mathbf{H}\left(\mathbf{L}^{opt} \circ \mathbf{P}^{sam}\right)^{1/2}$ for the original ensemble's sample covariance and the optimal length scale as well as the corresponding projection matrix $\mathbf{P}^{\mathbf{roj}}$.
(10) Repeat steps 5-7 replacing $\tilde{\mathbf{P}}_i^{loc}$ with $\tilde{\mathbf{P}}_i^{loc}\tilde{\mathbf{P}}_i^{\mathbf{roj}}$ and $\mathbf{P}^{sam}$ with $\mathbf{P}^{sam}\mathbf{P}^{\mathbf{roj}}$ and steps 8 and 9 in which the projection matrices are calculated using all previously found right singular vectors.

3. Bad Pseudocode

```
#Used in Step (2)
def resample(ensemble):
    resample_size = ensemble.shape[1]
    max_int = ensemble.shape[1]
    random_indexes = np.random.randint(0, max_int, resample_size)

    return ensemble[:, random_indexes]

# Used in Step (6)
def angle(V0, V1):
    IPs = V0.T @ V1
    norm0 = (V0 ** 2).sum(axis=0)
    norm1 = (V1 ** 2).sum(axis=0)
    norm0 = norm0[:, None]
    norm1 = norm1[None, :]
    sigmas = sp.linalg.svd(
        IPs / np.sqrt(norm0 * norm1),
        compute_uv=False)
    dist = np.arccos(sigmas.min())
    return dist

# Step (1)
ensemble = generate_ensemble(
    N_e, mu, P_sqrt)
P_sample = np.cov(ensemble)
```

```
P_sample_array = np.ones([dimension, dimension, ens_ens_size])
for ens_ens_num in range(N_ee):
    # Step (2)
    temp_ens = resample(ensemble=ensemble)
    # Step (3)
    P_sample_array[:, :, ens_ens_num] = np.cov(temp_ens)


for sig_count, sig_slice in enumerate(sig_slice_array):
    for rho_count, rho_loc in enumerate(rho_array):
        loc = generate_loc_matrix(
            dimension, dx, rho_loc, covariance.fft_sqd_exp_1d)
        for ens_count in range(N_ee):

            # Step (4)
            P_loc = P_sample_array[:, :, ens_count] * loc
            this_P_sqrt = covariance.matrix_sqrt(P_loc).real

            # Step (5)
            aU, aS, aVT = sp.linalg.svd(
                H @ this_P_sqrt @ proj_array[:, :, ens_count])
            U_array[:, sig_slice, rho_count, ens_count] = aU[:, :sig_num]
            s_array[sig_slice, rho_count, ens_count] = aS[:sig_num]
            V_array[:, sig_slice, rho_count, ens_count] = aVT[:sig_num, :].T

        # Step (6)
        comb_count = 0
        angles = np.ones(comb_num) * np.nan
        for ens_count in range(ens_ens_size):
            aV = V_array[:, sig_slice, rho_count, ens_count]

            for other_ens_count in range(ens_count + 1, ens_ens_size):
                oV = V_array[:, sig_slice, rho_count, other_ens_count]
                this_angle = angle(aV, oV)
                angles[comb_count] = this_angle
                comb_count += 1
        V_average_angle[sig_count, rho_count] = angles.mean()

    # Step (7)
    opt_rho_index =  V_average_angle[sig_count].argmin()
    opt_rho_array[sig_count] = rho_array[opt_rho_index]
    opt_U_array_ens[:, sig_slice] = U_array[:, sig_slice, opt_rho_index, :]
    opt_s_array_ens[sig_slice] = s_array[sig_slice, opt_rho_index, :]
```

```
    opt_V_array_ens[:, sig_slice] = V_array[:, sig_slice, opt_rho_index, :]

    # Step (8)
    proj_array = eye_array - np.einsum(
        'ij...,kj...->ik...',
        opt_V_array_ens[:, :sig_slice.stop],
        opt_V_array_ens[:, :sig_slice.stop])

    # Step (9)
    loc = generate_loc_matrix(
        dimension, dx, opt_rho_array[sig_count],
        covariance.fft_sqd_exp_1d)
    P_loc = P_sample * loc
    this_P_sqrt = covariance.matrix_sqrt(P_loc).real
    aU, aS, aVT = sp.linalg.svd(
        H @ this_P_sqrt @ proj)

    opt_U_array[:, sig_slice] = aU[:, :sig_num]
    opt_s_array[sig_slice] = aS.diagonal()[:sig_num]
    V_opt = aVT[:sig_num, :].T
    opt_V_array[:, sig_slice] = V_opt

    proj = np.eye(dimension) - (opt_V_array[:, :sig_slice.stop]
                                @ opt_V_array[:, :sig_slice.stop].T)
```