Part 1. We recap the line search problem and solution.

Notations:

Let $C^X \in \mathbb{R}^{n\times n}, C^Y \in \mathbb{R}^{m\times m}$ denote the cost matrices in source domain and target domains respectively.

Let $\mathcal{M} \in \mathbb{R}^{n\times n\times m\times m}$ be the cost tensor, defined in [29] as $\mathcal{M}_{i,i',j,j'} = \left|C^X_{i,i'} - C^Y_{j,j'}\right|^2$.

Let $T^{(k)} \in \mathbb{R}^{n\times m}$ denote the transportation plan at iteration k in the Frank-wolf algorithm.

Let $\tilde{T}^{(k)}$ denote the transportation plan obtained by the OT (or partial OT) solver at iteration $k+1$.

Let $\delta T = \tilde{T}^{(l)} - \tilde{T}^{(l)}$.

Let $M \circ T \in \mathbb{R}^{n\times m}$ denote the tensor product defined by $(\mathcal{M}\circ T)_{ij} = \sum_{ij} \mathcal{M}_{i,i',j,j'}\,\gamma_{i',j'}$

Problem setup:

We aim to solve $\gamma^* = \arg\min_{\gamma\in[0,1]} \left\langle \mathcal{M}\circ(1-\gamma)T^{(k)} + \gamma\tilde{T}^{(k)}, (1-\gamma)T^{(k)} + \gamma\tilde{T}^{(k)} \right\rangle$

$$= \arg\min_{\gamma\in[0,1]} \left\langle \mathcal{M}\circ T^{(k)} + \gamma\delta T, T^{(k)} + \gamma\delta T^{(k)} \right\rangle.$$

Simplifying the formula, we obtain a quadratic minimization problem:

$$\gamma^2\left\langle \mathcal{M}\circ\delta T^{(l)}, \delta T^{(l)}\right\rangle + \gamma\left\langle 2\mathcal{M}\circ T^{(l)}, \delta T^{(l)}\right\rangle + constant$$

By setting $a = \left\langle M\circ\delta T^{(l)}, \delta T^{(l)}\right\rangle, b = 2\left\langle M\circ T^{(l)}, \delta T^{(l)}\right\rangle,$ Eq (1)

we obtain the solution is the following:

$$\gamma = \begin{cases} 0 & if\ a \le 0, a+b \le 0 \\ 1 & if\ a \le 0, a+b > 0 \\ clip\left(-\frac{b}{2a}, [0,1]\right) & if\ a > 0 \end{cases}$$

Remark 1: If we use $\frac{1}{2}a, \frac{1}{2}b$ to replace a, b, the solution γ is unchanged. (Scaling does not change the line search problem.)

Remark 2: the above γ is consitant to the solution $\gamma^{(k)}$ in paper [29]

$\gamma \in [0,1]$

It can be shown that $\gamma^{(k)}$ can take the following values, with $\boldsymbol{E}^{(k)} = \widetilde{\boldsymbol{T}}^{(k)} - \boldsymbol{T}^{(k)}$:

- if $\langle \mathcal{M}(\boldsymbol{C}^s, \boldsymbol{C}^t) \circ \boldsymbol{E}^{(k)}, \boldsymbol{E}^{(k)} \rangle_F < 0$ we have

$$\gamma^{(k)} = \begin{cases} 0 & \text{if } \langle \mathcal{M}(\boldsymbol{C}^s, \boldsymbol{C}^t) \circ \boldsymbol{E}^{(k)}, \boldsymbol{E}^{(k)} \rangle_F + 2\langle \mathcal{M}(\boldsymbol{C}^s, \boldsymbol{C}^t) \circ \boldsymbol{E}^{(k)}, \boldsymbol{T}^{(k)} \rangle_F > 0 \\ 1 & \text{otherwise.} \end{cases}$$

- if $\langle \mathcal{M}(\boldsymbol{C}^s, \boldsymbol{C}^t) \circ \boldsymbol{E}^{(k)}, \boldsymbol{E}^{(k)} \rangle_F > 0$ we have

$$\gamma^{(k)} = \min\left(1, \max\left(0, -\frac{\langle \mathcal{M}(\boldsymbol{C}^s, \boldsymbol{C}^t) \circ \boldsymbol{E}^{(k)}, \boldsymbol{T}^{(k)} \rangle_F}{\langle \mathcal{M}(\boldsymbol{C}^s, \boldsymbol{C}^t) \circ \boldsymbol{E}^{(k)}, \boldsymbol{E}^{(k)} \rangle_F}\right)\right)$$

Part 2: We go through the code in linke ot.partial — POT Python Optimal Transport 0.9.3 documentation (pythonot.github.io).

Part 2.1. We verify the following (see the first two figures):

"gwgrad_partial" returns $\mathcal{M} \circ T$

"gw_loss_partial" returns $\frac{1}{2}\langle \mathcal{M} \circ T, T \rangle$

Part 2.2. We verify the following (see the third figures):

line "M=gw_gwgrad_partial(C1,C2,G0)", it returns $M = \mathcal{M} \circ \tilde{T}^{(k)}$ (where $G^0 = \tilde{T}^{(k)}$)

line "deltaG=G0-Gprev", it returns $\delta T = \tilde{T}^{(k)} - T^{(k)}$

line "a=gwloss_partial(C1,C2,deltaG)", it returns $a = \frac{1}{2}\langle \mathcal{M} \circ \delta T, \delta T \rangle$

line "b=2*np.sum(M*deltaG)", it returns $b = 2 \cdot \langle \mathcal{M} \circ \tilde{T}^{(k)}, \delta T \rangle$

Compare these two lines with equation (1), we observe they are not consistent.

```
                                                                    [docs]
def gwgrad_partial(C1, C2, T):
    """Compute the GW gradient. Note: we can not use the trick in :ref:`[12] <references-gwgrad
    as the marginals may not sum to 1.

    Parameters
    ----------
    C1: array of shape (n_p,n_p)
        intra-source (P) cost matrix

    C2: array of shape (n_u,n_u)
        intra-target (U) cost matrix

    T : array of shape(n_p+nb_dummies, n_u) (default: None)
        Transport matrix

    Returns
    -------
    numpy.array of shape (n_p+nb_dummies, n_u)
        gradient


    .. _references-gwgrad-partial:
    References
    ----------
    .. [12] Peyré, Gabriel, Marco Cuturi, and Justin Solomon,
        "Gromov-Wasserstein averaging of kernel and distance matrices."
        International Conference on Machine Learning (ICML). 2016.
    """
    cC1 = np.dot(C1 ** 2 / 2, np.dot(T, np.ones(C2.shape[0]).reshape(-1, 1)))
    cC2 = np.dot(np.dot(np.ones(C1.shape[0]).reshape(1, -1), T), C2 ** 2 / 2)
    constC = cC1 + cC2
    A = -np.dot(C1, T).dot(C2.T)
    tens = constC + A
    return tens * 2
```

```python
def gwloss_partial(C1, C2, T):
    """Compute the GW loss.

    Parameters
    ----------
    C1: array of shape (n_p,n_p)
        intra-source (P) cost matrix

    C2: array of shape (n_u,n_u)
        intra-target (U) cost matrix

    T : array of shape(n_p+nb_dummies, n_u) (default: None)
        Transport matrix

    Returns
    -------
    GW loss
    """
    g = gwgrad_partial(C1, C2, T) * 0.5
    return np.sum(g * T)
```

```
    while (err > tol and cpt < numItermax):

        Gprev = np.copy(G0)

        M = gwgrad_partial(C1, C2, G0)
        M_emd = np.zeros(dim_G_extended)
        M_emd[:len(p), :len(q)] = M
        M_emd[-nb_dummies:, -nb_dummies:] = np.max(M) * 1e2
        M_emd = np.asarray(M_emd, dtype=np.float64)

        Gc, logemd = emd(p_extended, q_extended, M_emd, log=True, **kwargs)

        if logemd['warning'] is not None:
            raise ValueError("Error in the EMD resolution: try to increase the"
                             " number of dummy points")

        G0 = Gc[:len(p), :len(q)]

        if cpt % 10 == 0:  # to speed up the computations
            err = np.linalg.norm(G0 - Gprev)
            if log:
                log['err'].append(err)
            if verbose:
                if cpt % 200 == 0:
                    print('{:5s}|{:12s}|{:12s}'.format(
                        'It.', 'Err', 'Loss') + '\n' + '-' * 31)
                print('{:5d}|{:8e}|{:8e}'.format(cpt, err,
                                                 gwloss_partial(C1, C2, G0)))

        deltaG = G0 - Gprev
        a = gwloss_partial(C1, C2, deltaG)
        b = 2 * np.sum(M * deltaG)
        if b > 0:  # due to numerical precision
            gamma = 0
```

```
        if logemd['warning'] is not None:
            raise ValueError("Error in the EMD resolution: try to increase the"
                             " number of dummy points")

        G0 = Gc[:len(p), :len(q)]

        if cpt % 10 == 0:  # to speed up the computations
            err = np.linalg.norm(G0 - Gprev)
            if log:
                log['err'].append(err)
            if verbose:
                if cpt % 200 == 0:
                    print('{:5s}|{:12s}|{:12s}'.format(
                        'It.', 'Err', 'Loss') + '\n' + '-' * 31)
                print('{:5d}|{:8e}|{:8e}'.format(cpt, err,
                                                 gwloss_partial(C1, C2, G0)))

        deltaG = G0 - Gprev
        a = gwloss_partial(C1, C2, deltaG)
        b = 2 * np.sum(M * deltaG)
        if b > 0:  # due to numerical precision
            gamma = 0
            cpt = numItermax
        elif a > 0:
            gamma = min(1, np.divide(-b, 2.0 * a))
        else:
            if (a + b) < 0:
                gamma = 1
            else:
                gamma = 0
                cpt = numItermax

        G0 = Gprev + gamma * deltaG
        cpt += 1
```