2-D tomography MATLAB toolbox

Chu, H., Liang, L., Toh, K. C., & Yang, L. (2020). An efficient implementable inexact entropic proximal point algorithm for a class of linear programming problems. arXiv preprint arXiv:2011.14312.

This toolbox solves a class of specially structured linear programming (LP) problems in the following form:

$$\begin{aligned} &\min \quad \langle C, X\rangle \\ &\text{s.t.}\;\; X \in \Omega := \Big\{ X \in \mathbb{R}^{n_1\times n_2} \;:\; \mathcal{A}^{(i)}(X) \;=\; \boldsymbol{b}^{(i)}, \quad i=1,\ldots,N, \quad 0 \le X \le U \Big\}, \end{aligned} \tag{1}$$

where $\langle \cdot,\cdot\rangle$ denotes the standard inner product in $\mathbb{R}^{n_1\times n_2}$, $\mathcal{A}^{(i)} : \mathbb{R}^{n_1\times n_2} \to \mathbb{R}^{m_i}$ is a given linear mapping defined by

$$\mathcal{A}^{(i)}(X) := \begin{bmatrix} \left\langle A_1^{(i)}, X \right\rangle \\ \vdots \\ \left\langle A_{m_i}^{(i)}, X \right\rangle \end{bmatrix}, \quad A_j^{(i)} \in \mathbb{R}^{n_1\times n_2}, \quad 1 \le j \le m_i, \quad 1 \le i \le N, \tag{2}$$

$\boldsymbol{b}^{(i)} = (b_1^{(i)}, \ldots, b_{m_i}^{(i)})^\top \in \mathbb{R}^{m_i}$ $(i = 1,\ldots,N)$, $C \in \mathbb{R}^{n_1\times n_2}$ and $U \in \mathbb{R}_+^{n_1\times n_2} \cup \{\infty\}^{n_1\times n_2}$ are given data.

1. `genData2D` *this function generates data for the experiments*

```
[P0,U0,Const] = genData2D(filename,...
'UpperBound',U,'Size',[256,256],'FlagNormalize',true);
```

input arguments `filename` (name of the file containing image, or a matrix), `U` (setting of the upper bound, name of a file or a matrix), `Size` (1-by-2 vector to resize image), `FlagNormalize` (normalize input if it is true).
output arguments `P0` (target image of size `Size`), `U0` (upper bound of size `Size`), `Const` (the scale constant).

2. `genCollectionA` *this function generates the linear constraint*

```
[Opers,OpersAdj,MatAlins] = genCollectionA(Size,Theta,...
'Input','degree','Level',[P,Q]);
```

input arguments: `Size` (2-by-1 vector indicates the size of X, e.g., `Size = [256,256]`), `Theta` (N-by-1 vector whose each entry θ indicates the angle in degree of the projection, e.g., `Theta = [0;45;-45;90]`), `Level` (1-by-2 vector of level, default `[P,Q]=[32,32]`).
processing: for each $\theta \neq 90 + k180$, it seeks the minimum

$$\frac{p}{q} := \min\left\{ \left|\tan(\theta) - \frac{\tilde p}{\tilde q}\right| \;\text{ such that } \tilde p \in \{0,1,\ldots,P\}, 0 \neq \tilde q \in \{-Q,-Q+1\ldots,Q\} \right\}$$

and set $\frac{p}{q} := \frac{1}{0}$ when $\theta = 90 + k180$. Then, it generates a set of binary matrices $A_j^{(i)} \in \{0,1\}^{n_1 \times n_2}$ such that $\sum_{j=1}^{m_j} A_j^{(i)} = \{1\}^{n_1 \times n_2}$ where in each $A_j^{(i)}$, its support set $T := \left\{(r,s) \mid \left[A_j^{(i)}\right]_{rs} = 1\right\}$ satisfies that

$$\frac{s - s_0}{r - r_0} = \frac{q}{p} \text{ for any } (r,s) \in T, \tag{3}$$

where, $r_0 := \min_r\{(r,s) \in T\}$. Then, it generates $\mathcal{A}^{(i)}$ and $\mathcal{A}^{(i,*)}$ as in (2).
output arguments: `Opers` (N-by-1 cell of all linear map $\mathcal{A}^{(i)}$), `OpersAdj` (N-by-1 cell of $\mathcal{A}^{(i,*)}$), `MatAlins` (N-by-1 cell of matrices representation of $\mathcal{A}^{(i)}$).

```
[Opers,OpersAdj,MatAlins] = genCollectionA(Size,Dir,...
'Input','Ratio');
```

input arguments: it can takes the direction $\frac{p}{q}$ directly by input `Dir` (2-by-N array whose each column $[p, q]$ indicates the angle θ of the projection so that $\tan(\theta) = \frac{p}{q}$, e.g., `Dir = [1;4]`).
To generate the set of projections $\boldsymbol{b}^{(i)}$,

```
Proj = cellfun(@(cel) cel(P0),Opers,'Uni',false);
```

3. `EPPA` *this is the main EPPA function solving* (1)

```
[X,info] = EPPA(C,Opers,OpersAdj,Proj,...
'Tolerance',1e-5,'MaxIter',1e3,'MaxTime',3600,...
'epsilon',0.05,'FlagEpsilon',false,...
'FlagPrint',2,'FlagPlot',1,'FlagDphi',false);
```

Note that `'FlagPrint'`,`'FlagPlot'`,`'FlagDphi'` control how much output are displayed in each iteration, hence larger values might slow down the performance.