## implement notears

### need put this file under /notear

```
In [ ]: import numpy as np
        import utils
        import linear
        import pandas as pd
        utils.set_random_seed(1)
        import scipy
```

```
In [ ]: n, d, s0, graph_type, sem_type = 100, 20, 20, 'ER', 'gauss'
        # d (int): num of nodes
        # s0 (int): expected num of edges
        # graph_type (str): ER, SF, BP
        # n (int): num of samples, n=inf mimics population risk
        # sem_type (str): gauss, exp, gumbel, uniform, logistic, poisson

        B_true = utils.simulate_dag(d, s0, graph_type)
        W_true = utils.simulate_parameter(B_true)
        np.savetxt('W_true.csv', W_true, delimiter=',')

        X = utils.simulate_linear_sem(W_true, n, sem_type)
        #np.savetxt('X.csv', X, delimiter=',')

        W_est_notears = linear.notears_linear(X, lambda1=0.1, loss_type='l2')
        assert utils.is_dag(W_est_notears)
        np.savetxt('W_est_notears.csv', W_est_notears, delimiter=',')
```

## implement glasso

### Here I also use 0.3 as threshold and 1 as multiplier for penalty.

### Problem: probably not DAG

```
In [ ]: import rpy2.robjects as robjects
        r_script = '''
        library(glasso)
        X = read.csv('X.csv', header=FALSE)
        X = data.matrix(X)
        w = matrix(0, nrow=ncol(X), ncol=ncol(X))
        for (i in c(1:ncol(X))){
          a = glasso(t(X[,-c(i)]) %*% X[,-c(i)], rho=1)
          w[-c(i), i] = a$wi %*% t(X[,-c(i)]) %*% X[, i]
        }
        w = (abs(w)>0.3) * w
        write.csv(w, 'W_est_glasso.csv')
        '''
        robjects.r(r_script)
```

```
In [ ]: W_est_glasso = pd.read_csv('W_est_glasso.csv', index_col=0)
        W_est_glasso = np.array(W_est_glasso)
```

## evaluation

In [ ]:

```
# Hamming distance
print("Use Hamming distance as evaluation: [smaller is better]")
print("NOTEARS: {:.4f}".format(scipy.spatial.distance.hamming((W_est_notears!=(
print("glasso: {:.4f}".format(scipy.spatial.distance.hamming((W_est_glasso!=0).
print("")

# Jaccard distance
print("Use Jaccard-Needham dissimilarity as evaluation: [smaller is better]")
print("NOTEARS: {:.4f}".format(scipy.spatial.distance.jaccard((W_est_notears!=(
print("glasso: {:.4f}".format(scipy.spatial.distance.jaccard((W_est_glasso!=0).
print("")

# 2-norm
print("Use 2-norm as evaluation: [smaller is better]")
print("NOTEARS: {:.4f}".format(np.linalg.norm(W_est_notears-W_true, ord=2)))
print("glasso: {:.4f}".format(np.linalg.norm(W_est_glasso-W_true, ord=2)))
print("")

# accuracy
print("Use built-in evaluation function as evaluation: [fdr, tpr, fpr, shd smal
print("NOTEARS: ", utils.count_accuracy(B_true, W_est_notears!=0))
print("glasso: ", utils.count_accuracy(B_true, W_est_glasso!=0))
print("")
```

```
Use Hamming distance as evaluation: [smaller is better]
NOTEARS: 0.0025
glasso: 0.0925

Use Jaccard-Needham dissimilarity as evaluation: [smaller is better]
NOTEARS: 0.0500
glasso: 0.6852

Use 2-norm as evaluation: [smaller is better]
NOTEARS: 0.7233
glasso: 2.0822

Use built-in evaluation function as evaluation: [fdr, tpr, fpr, shd smaller is
better]
NOTEARS:  {'fdr': 0.0, 'tpr': 0.95, 'fpr': 0.0, 'shd': 1, 'nnz': 19}
glasso:  {'fdr': 0.6666666666666666, 'tpr': 0.85, 'fpr': 0.2, 'shd': 31, 'nn
z': 51}
```