Imports

Packages

In [1]:
```
import os
import cvxpy as cp
import numpy as np
import statsmodels.api as sm
from statsmodels.tsa.api import VAR
from numpy import linalg
import pandas as pd
import astropy
import datetime as dt
from astropy.coordinates import EarthLocation
import matplotlib.pyplot as plt
import matplotlib as mpl
from matplotlib import animation
import seaborn as sns
import scipy.stats as stats
import random
import cartopy.crs as ccrs
import cartopy
import cartopy.feature as cfeature
from matplotlib import animation, rc
import plotly.graph_objects as go
from IPython.display import HTML
from sklearn import linear_model
import plotly.graph_objects as go
import networkx as nx
import plotly.express as px
plt.rcParams.update({'font.size': 13})
```

executed in 2.53s, finished 15:59:56 2020-04-12

Global Variables

In [2]:
```
from AirLib import data_accessories
# , AIRLINE_DATA

global CODES, NAMES, CLRS, AIRLINE_DATA
CODES = ['WN','DL','AA','OO','UA','YX',
         'B6','MQ','OH','AS','9E','YV',
         'NK','EV','F9','G4','HA']
NAMES = ['Southwest', 'Delta', 'American',
         'SkyWest', 'United', 'Midwest',
         'Jet Blue', 'Am. Eagle', 'PSA',
         'Alaska', 'Endeavor','Mesa', 'Spirit',
         'ExpressJet', 'Frontier', 'Allegiant',
         'Hawaiian']
CLRS = sns.color_palette("tab10", len(CODES))
```

executed in 43ms, finished 15:59:57 2020-04-12

Classes

In [3]:
```
class clean_raw_data:

    # reading from data_accessories
    ap_info = data_accessories.ap_info

    def __init__(self, airports, routes, fares, delays):

        # reading/cleaning data
        self.airports = airports
        self.routes = routes
        self.fares = fares
        self.delays = self.process_delays(delays)
        self.make_airport_coords()

    def make_airport_coords(self):

        '''
        Merging additional information with given airport
        info. Adding airport name as well as lat/lon data.
        '''
        airport_name = [] ; lat = [] ; lon = []
        x = [] ; y = [] ; z = []
        for AP in self.airports['Airport'].unique():

            ap_i = self.ap_info[self.ap_info['AIRPORT'] == AP].iloc[-1]
            airport_name.append(ap_i['DISPLAY_AIRPORT_NAME'])

            # gettinbg airport latitude and longitude
            lon_i = ap_i['LONGITUDE']
            lat_i = ap_i['LATITUDE']


            # converts to geocentric coordinates meters
            geo_centr = EarthLocation(lon_i, lat_i).value

            lat.append(lat_i)
            lon.append(lon_i)
            x.append(geo_centr[0])
            y.append(geo_centr[1])
            z.append(geo_centr[2])

        # lat / long data
        self.airports['airport_name'] = airport_name
        self.airports['lon'] = lon
        self.airports['lat'] = lat

        # geocentric cartesian coordinates (meters)
        self.airports['x'] = x
        self.airports['y'] = y
        self.airports['z'] = z

    def process_delays(self, data):
        # getting rid of nan
        self.delays = data.dropna(subset=['ARR_DELAY','DEP_DELAY'])
        data['datetime'] = pd.to_datetime(data['FL_DATE'])
        data = data[data['CANCELED'] != 1]
        return data

class prep_delay_data:

    def __init__(self, data, carrier, N_airports):
        self.carrier = carrier
        self.N_airports = N_airports
        self.top_airports = self.find_top_aps(data)
        self.delay_matr = self.clean_and_format(data)

    def summarize(self):
        pass

    def find_top_aps(self, data):
        top_airports = {}
        for i in range(len(CODES)):
            carr = CODES[i]
            sub = AIRLINE_DATA.delays[AIRLINE_DATA.delays['CARRIER'] == carr]
            ap_counts = sub['ORIGIN'].value_counts()
```

```
            top_airports[carr] = ap_counts.index[:self.N_airports]
        return list(top_airports[self.carrier])

    def clean_and_format(self, data):
        data['datetime'] = pd.to_datetime(data['datetime'])
        subset_0 = data[((data['airport']).isin(self.top_airports)) &
                        (data['carrier'] == self.carrier)]

        '''~ dropping bad dates ~'''
        # if a given airport has 0 flights on some,
        # remove all flights on that day
        bad_index = []
        for ap in subset_0['airport'].unique():
            sub = subset_0[subset_0['airport'] == ap]['avg_dep']
            nan_rows = pd.isna(sub)
            nan_inds = nan_rows[nan_rows == True].index
            bad_index.extend(list(nan_inds))

        bad_dates = subset_0['datetime'].loc[bad_index]
        inds_to_drop = subset_0[subset_0['datetime'].isin(bad_dates)].index
        subset_1 = subset_0.drop(inds_to_drop)
        #print('Number of rows dropped:',len(subset_0)-len(subset_1))

        '''~ log transforming ~ & ~ making delay matrix ~'''
        trans_rows = []
        delay_matr = pd.DataFrame()
        diff_matr = pd.DataFrame()
        matr = []
        for ap in self.top_airports:
            sub = subset_1[subset_1['airport'] == ap]
            # airport wise transformation, each airport X_i ~ N(0,sigma**2)
            delays = sub['avg_dep'].values
            shift = abs(delays.min())+1
            delays += shift
            delays = np.log(delays)
            mu_log = np.mean(delays)
            delays -= mu_log

            # making matr object (cleaned and transformed version of original input)
            # note that delay_matr is just a table of delays with no info about time etc
            sub['avg_dep'] = np.log(sub['avg_dep']+shift)-mu_log

            # making matrs
            delay_matr[ap] = delays
            diff_matr[ap] = pd.Series(delays).diff(1)[1:]
            # appending data
            matr.append(sub)
            trans_rows.append({'airport': ap,
                               'shift': shift,
                               'mu_log': mu_log})

        self.trans_df = pd.DataFrame(trans_rows)
        self.matr = pd.concat(matr)
        self.matr['raw_avg_dep'] = subset_1['avg_dep']
        self.diff_matr = diff_matr
        '''~ sorting based on distances ~'''
        unsorted_aps = delay_matr.columns
        rltv_dists = []
        rfrnce_ap = unsorted_aps[0]
        for ap_i in unsorted_aps:
            if ap_i != rfrnce_ap:
                # trying to see if there is an actual flight, if not, set 1e6 distance
                try:
                    subset = AIRLINE_DATA.routes[
                        (AIRLINE_DATA.routes['ORIGIN'] == rfrnce_ap) &
                        (AIRLINE_DATA.routes['DEST'] == ap_i)
                        ]
                    d = subset['DISTANCE'].values[0]
                    rltv_dists.append({'airport': ap_i, 'distance': d})
                except:
                    rltv_dists.append({'airport': ap_i, 'distance': 1e6})
            else:
                rltv_dists.append({'airport': rfrnce_ap, 'distance': 0})

        dist_df = pd.DataFrame(rltv_dists).sort_values(by = ['distance'])
```

```
        delay_matr = delay_matr[dist_df['airport']]

        return delay_matr

class visual:
    '''
    Note that this class takes multiple different kinds of
    data. It will not execute if you call a plotting method
    that doesnt math the data type (dataframe with correct columns)
    '''
    def __init__(self, data, carrier):
        self.data = data
        self.carrier = carrier
        self.index = CODES.index(carrier)

    def flight_frequency(self):
        dates = self.data['FL_DATE'].unique()
        counts = self.data['ORIGIN'].value_counts()/len(dates)
        aps = counts.index

        fig = go.Figure(data=[go.Bar(
                x=aps, y=counts,
                marker_color=['rgb{}'.format(CLRS[self.index]) for i in range(len(aps))])
                      ])
        fig.update_layout(
            title="{} - Flights Per Day ({}, {})".format(NAMES[self.index], dates[0], dates[-1]),
            plot_bgcolor="white",
            width = 950, height = 500)

        color = 'rgb(209, 209, 209)'
        fig.update_xaxes(showgrid=True, gridwidth=1, gridcolor=color)
        fig.update_yaxes(showgrid=True, gridwidth=1, gridcolor=color)
        fig.show()

    def time_series(self):
        fig = go.Figure()
        for c in self.data['airport'].unique():
            sub = self.data[self.data['airport'] == c]
            fig.add_trace(go.Scatter(
                           x = sub['datetime'],
                           y = sub['avg_dep'],
                           name = c,
                           opacity=.9))

        fig.update_layout(title = '{} - Mean Delay'.format(NAMES[self.index]),
                          xaxis_range=[
                              self.data['datetime'].min(),
                              self.data['datetime'].max()
                          ],
                         xaxis_rangeslider_visible=True,
                         width = 950, height = 600,
                         plot_bgcolor = "white")
        color = 'rgb(209, 209, 209)'
        fig.update_xaxes(showgrid=True, gridwidth=1, gridcolor=color)
        fig.update_yaxes(showgrid=True, gridwidth=1, gridcolor=color)
        fig.show()

    def cov_matr(self):
        # airpots, covariance & precision matrix
        aps = self.data.columns
        cov_matr = np.corrcoef(self.data.T)
        prec_matr = linalg.inv(cov_matr)

        fig, (ax0, ax1) = plt.subplots(1, 2, figsize = (18,7))
        sns.heatmap(cov_matr, ax = ax0, center=0,
                    xticklabels = aps, yticklabels = aps)
        sns.heatmap(prec_matr, ax = ax1, center=0,
                    xticklabels = aps, yticklabels = aps)
        plt.show()

    def dist_corr(self):
        aps = list(self.data.columns)
        cov_matr = np.corrcoef(self.data.T)
        prec_matr = linalg.inv(cov_matr)
        pairs = []
```

```
        for i in aps:
            for j in aps:
                if i!=j and (j, i) not in pairs:
                    pairs.append((i,j))

        values = []
        for p in pairs:
            subset = AIRLINE_DATA.routes[
                (AIRLINE_DATA.routes['ORIGIN'] == p[0]) &
                (AIRLINE_DATA.routes['DEST'] == p[1])]
            d = subset['DISTANCE'].values

            if len(d) > 0:
                values.append({'distance': d[0],
                               'corr': cov_matr[aps.index(p[0])][aps.index(p[1])],
                               'origin': p[0],
                               'dest': p[1]})

        dist_corr = pd.DataFrame(values)
        plt.figure(figsize = (10,6))
        plt.grid(b=True, which='major', color='grey', linewidth=0.3)
        plt.grid(b=True, which='minor', color='grey', linewidth=0.3)
        plt.scatter(dist_corr['distance'], dist_corr['corr'], s = 40, color = 'k')
        plt.xlabel('Distance', fontsize = 17)
        plt.ylabel('Correlation', fontsize = 17)
        plt.show()
```

executed in 30ms, finished 16:00:39 2020-04-12

In [19]:
```
class meinhausen_bulman:
    def __init__(self, name, delay_matr, carrier, lambd):
        self.name = name
        self.delay_matr = delay_matr
        self.carrier = carrier
        self.lambd = lambd
        self.nodes = list(delay_matr.columns)
        self.params = self.estimate_parmeters()
        self.edges = self.stitch_edges()

    def estimate_parmeters(self):
        params = {}
        for node_i in self.nodes:
            data_i = self.delay_matr[node_i]
            data_rest = self.delay_matr[[i for i in self.nodes if i != node_i]]
            lassoreg_i = linear_model.Lasso(self.lambd)
            lassoreg_i.fit(data_rest, data_i)
            params[node_i] = lassoreg_i.set_params()
        return params

    def stitch_edges(self):
        delta = 0.01
        edges = pd.DataFrame()
        for node_i, model_i in self.params.items():
            ind = list(self.params.keys()).index(node_i)
            coeffs = [i if abs(i) >= delta else 0 for i in model_i.coef_]
            coeffs.insert(ind, 1)
            edges[node_i] = coeffs

        for node_i in self.nodes:
            for node_j in range(len(self.nodes)):
                edge_pair = [edges[node_i].iloc[node_j],
                             edges[self.nodes[node_j]].iloc[self.nodes.index(node_i)]]
                max_edge = max(edge_pair)
                edges[node_i].iloc[node_j] = max_edge
                edges[self.nodes[node_j]].iloc[self.nodes.index(node_i)] = max_edge
        return edges

    def view_graph(self, g_type):
        N = self.nodes

        G=nx.Graph()
        G.add_nodes_from(N)
        routes = MY_DATA[['airport', 'lat', 'lon']].drop_duplicates()
        plotly_rows = []
        for n_i in N:
            for n_j in N:
                if n_i != n_j:
                    weight = self.edges[n_i].iloc[N.index(n_j)]
                    if weight != 0:
                        edge = (n_i, n_j)
                        G.add_edge(n_i, n_j, weight = weight)

                    org = routes[routes['airport'] == n_i]
                    dest = routes[routes['airport'] == n_j]
                    plotly_rows.append({
                        'weight': weight,
                        'start_lon': org['lon'].values[0],
                        'end_lon': dest['lon'].values[0],
                        'start_lat': org['lat'].values[0],
                        'end_lat': dest['lat'].values[0],
                        'start_airport': n_i,
                        'end_airport': n_j
                    })
        plotly_df = pd.DataFrame(plotly_rows).drop_duplicates()

        if g_type == 'circular':
            plt.figure(figsize = (2.5,2.5))

            nx.draw_circular(G, node_size = 125,
                             node_color = [CLRS[CODES.index(self.carrier)]],
                             with_labels=False)
            plt.show(G)

        elif g_type == 'geographic':
```

```
            sub = plotly_df[['start_airport','start_lat','start_lon']].drop_duplicates()
            inds = plotly_df[[i for i in plotly_df.columns if 'end' not in i]].drop_duplicates().index
            plotly_df = plotly_df.loc[inds]

            fig = go.Figure()
            for ap in sub['start_airport'].unique():
                ap_i = sub[sub['start_airport'] == ap]
                fig.add_trace(
                    go.Scattergeo(
                        locationmode = 'USA-states',
                        lon = ap_i['start_lon'],
                        lat = ap_i['start_lat'],
                        hoverinfo = 'text',
                        name = ap,
                        mode = 'markers',
                        marker = dict(
                            size = 2,
                            color = 'rgb(0,0,0)',
                            line = dict(width = 3,color = 'rgb(0,0,0)'))
                    )
                )
            line_colors = ['red','green']
            def sign_color(weight):
                if weight >= 0:
                    return 'green'
                else:
                    return 'red'

            flight_paths = []
            for i in range(len(plotly_df)):
                fig.add_trace(
                    go.Scattergeo(
                        locationmode = 'USA-states',
                        lon = [plotly_df['start_lon'].iloc[i], plotly_df['end_lon'].iloc[i]],
                        lat = [plotly_df['start_lat'].iloc[i], plotly_df['end_lat'].iloc[i]],
                        mode = 'lines',
                        name = plotly_df['end_airport'].iloc[i],
                        opacity = 1,
                        line = dict(
                            width = abs(plotly_df['weight'].iloc[i])*5,
                            color = sign_color(plotly_df['weight'].iloc[i]))
                    )
                )
            fig.update_layout(
                title = 'Carrier - {}'.format(self.name),
                showlegend = False,
                geo = dict(
                    scope = 'north america',
                    projection_type = 'azimuthal equal area',
                    showland = True,
                    landcolor = 'rgb(230,230,230)',
                    countrycolor = 'rgb(204, 204, 204)'),
                width = 960, height = 700)
            fig.show()
```

executed in 18ms, finished 16:03:21 2020-04-12

In [28]:
```
class fuse_lasso:

    DELTA = 0.01
    def __init__(self, flight_data, M, lambd_grid, alpha_grid):
        self.flight_data = flight_data
        self.M = M
        self.nodes = list(M.columns)
        self.T = M.shape[0]
        self.d = M.shape[1]
        self.diff_matr = self.make_diff_matr()
        self.lambd_grid = lambd_grid
        self.alpha_grid = alpha_grid
        self.edge_list = self.edge_estimation()
        self.graphs = self.precision_matricies()

    def make_diff_matr(self):
        D = (np.identity(self.T) + np.diag([-1]*(self.T-1),k=1))[:-1]
        return D

    def MSE(self, X, y, beta):
        return (.5)*sum([(y[t] - sum([beta[t][i]*X[t][i] for i in range(self.d-1)]))**2 for t in range(self.T)])

    def l1_norm(self, beta):
        return sum([cp.norm1(beta[t]) for t in range(self.T)])

    def fusion(self, beta):
        return sum([cp.norm1(vec) for vec in [self.diff_matr@beta.T[i] for i in range(self.d-1)]])

    def obj_func(self, X, y, beta_matr, _lamda_, _alpha_):
        return self.MSE(X, y, beta_matr) + _lamda_*self.l1_norm(beta_matr) + _alpha_*self.fusion(beta_matr)

    def edge_estimation(self):
        estimates = []
        for node_i in self.nodes:
            y = self.M[node_i].to_numpy()
            X = self.M[[j for j in self.nodes if j != node_i]].to_numpy()

            # node_i optimization
            _lambd_ = cp.Parameter(nonneg=True)
            _alpha_ = cp.Parameter(nonneg=True)
            beta_matr = cp.Variable(shape = (self.T, self.d-1))
            optim_problem = cp.Problem(
                cp.Minimize(self.obj_func(X, y, beta_matr, _lambd_, _alpha_))
            )

            for l in self.lambd_grid:
                _lambd_.value = l
                for a in self.alpha_grid:
                    _alpha_.value = a
                    optim_problem.solve(solver='ECOS')
                    estimates.append({'node': node_i,
                                      'l': l, 'a': a,
                                      'error': optim_problem.value,
                                      'beta_matrix': beta_matr.value})
        return pd.DataFrame(estimates)

    def precision_matricies(self):
        omega_list = []
        for t in range(self.T):

            # creaing precision_matrix(t)
            edges_t = pd.DataFrame()
            for i in range(len(self.edge_list['node'])):
                node_i = self.nodes[i]
                beta_t_i = list(
                    self.edge_list[self.edge_list['node'] == node_i]['beta_matrix'].iloc[0][t]
                )
                beta_t_i.insert(i,1)
                edges_t[node_i] = beta_t_i

            # stitching the edges for each precision_matrix(t)
            for node_i in self.nodes:
                for node_j in self.nodes:
                    index_i = self.nodes.index(node_i)
                    index_j = self.nodes.index(node_j)
```

```
                max_edge = max([edges_t[node_i].iloc[index_j],
                                edges_t[node_j].iloc[index_i]])
                if max_edge < 0.01:
                    max_edge = 0
                edges_t[node_i].iloc[index_j] = max_edge
                edges_t[node_j].iloc[index_i] = max_edge
        omega_list.append(edges_t)
    return omega_list

def plot_paramaters(self, demo = False):
    dates = self.flight_data['datetime'].unique()

    num_nodes = len(self.nodes)
    if demo == True:
        num_nodes = 3
    for i in range(num_nodes):
        params = []
        for omega_t in self.graphs:
            indicies = [x for x in range(len(self.nodes)) if x != i]
            params.append(omega_t[self.nodes[i]].iloc[indicies])
        plt.figure(figsize = (13,4))
        plt.plot(dates, params)
        plt.title(self.nodes[i])
        plt.show()

def generate_segments(self, threshold, plot = True):
    omega = self.graphs
    dates = self.flight_data['datetime'].unique()
    omega_delta = [omega[i+1].to_numpy()-omega[i].to_numpy() for i in range(len(omega)-1)]

    rows = []
    for i in range(len(omega_delta)):
        Sum = 0
        omega_dt = omega_delta[i]
        for j in range(len(omega_dt)):
            Sum += omega_dt[j].max()
        rows.append({'dates': dates[i], 'value': Sum})
    df = pd.DataFrame(rows)

    if plot == True:
        plt.figure(figsize = (14, 6))
        plt.plot(df['dates'], df['value'], color = 'k')
        plt.title('lambda = {}'.format(self.alpha_grid[0]))
        #plt.hlines(y = threshold,
        #           xmin = dates[0], xmax= dates[-1],
         #          color = 'red',
          #         linestyles  = 'dashed',
           #        linewidth = 4)
        plt.show()


    good = df[(df['value'] >= threshold)]
    dates = good['dates']
    bad_dates = []
    for i in range(len(dates)-1):
        if (dates.iloc[i+1]-dates.iloc[i] < dt.timedelta(days = 30)):
            bad_dates.append(dates.iloc[i+1])

    return good[~(good['dates']).isin(bad_dates)]
```

executed in 23ms, finished 16:06:56 2020-04-12

Data

In [6]:
```
path = os.getcwd() + '\\'

airports0 = pd.read_csv(path + 'Airports.csv')
routes0 = pd.read_csv(path + 'Routes.csv')
fares0 = pd.read_csv(path + 'AirFares.csv')
delays0 = pd.read_csv(path + 'FlightDelays.csv')

AIRLINE_DATA = clean_raw_data(airports0, routes0, fares0, delays0)
```
executed in 37.7s, finished 16:01:27 2020-04-12

```
C:\Users\Landon\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py:3057: DtypeWarning:

Columns (32) have mixed types. Specify dtype option on import or set low_memory=False.
```

In [7]:
```
MY_DATA = pd.read_csv(path + '\correct_MATR.csv')
MY_DATA['datetime'] = pd.to_datetime(MY_DATA['datetime'])
```
executed in 419ms, finished 16:01:29 2020-04-12

First look at the data

In [10]:
```
print(CODES)
```
executed in 4ms, finished 16:01:50 2020-04-12

```
['WN', 'DL', 'AA', 'OO', 'UA', 'YX', 'B6', 'MQ', 'OH', 'AS', '9E', 'YV', 'NK', 'EV', 'F9', 'G4', 'HA']
```

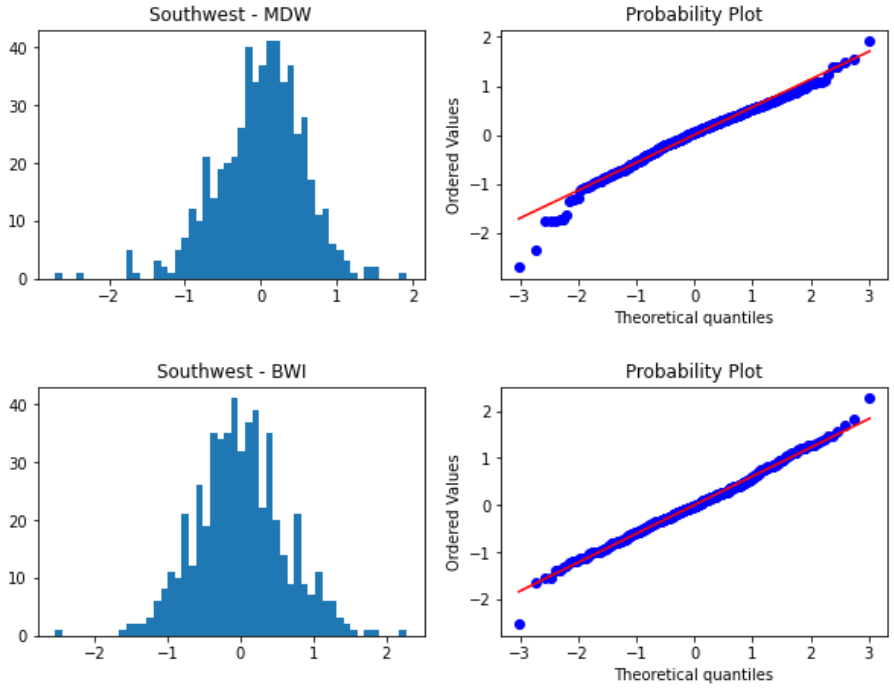
In [163]:
```
num_airports = 30
carrier = 'WN'
```
executed in 4ms, finished 17:13:28 2020-04-12

In [164]:
```
X = prep_delay_data(MY_DATA, carrier, num_airports)

matr = X.matr
delay_matr = X.delay_matr
```
executed in 12.5s, finished 17:13:41 2020-04-12

```
C:\Users\Landon\Anaconda3\lib\site-packages\ipykernel_launcher.py:116: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
```

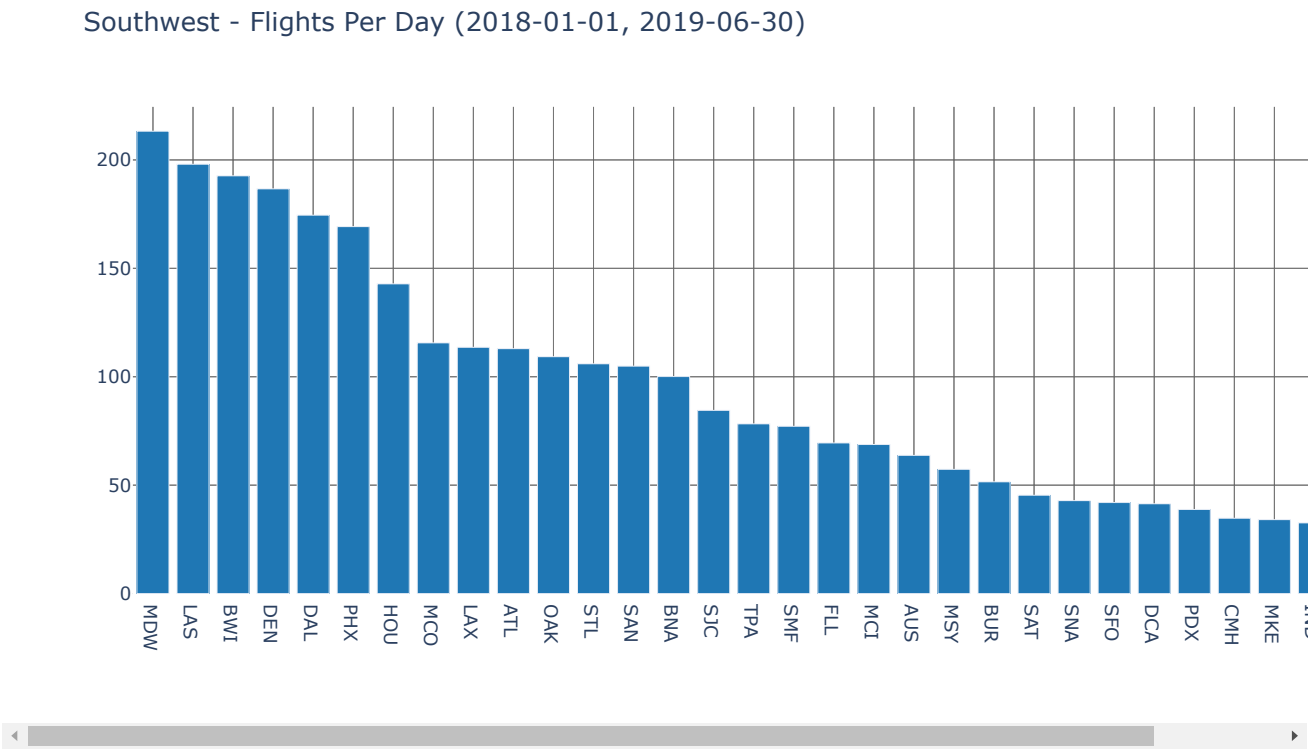
In [165]:
```
top_3 = X.top_airports

y = delay_matr[top_3[0]]
fig, (ax0, ax1) = plt.subplots(1,2, figsize = (10,3))
ax0.hist(y, bins = 50, color = CLRS[CODES.index(carrier)])
ax0.set_title('{} - {}'.format(f'{NAMES[CODES.index(carrier)]}', top_3[0]))
stats.probplot(y, dist=stats.norm, plot=ax1)
plt.show()

y = delay_matr[top_3[2]]
fig, (ax0, ax1) = plt.subplots(1,2, figsize = (10,3))
ax0.hist(y, bins = 50, color = CLRS[CODES.index(carrier)])
ax0.set_title('{} - {}'.format(f'{NAMES[CODES.index(carrier)]}',top_3[2]))
stats.probplot(y, dist=stats.norm, plot=ax1)
plt.show()
```

executed in 380ms, finished 17:13:42 2020-04-12

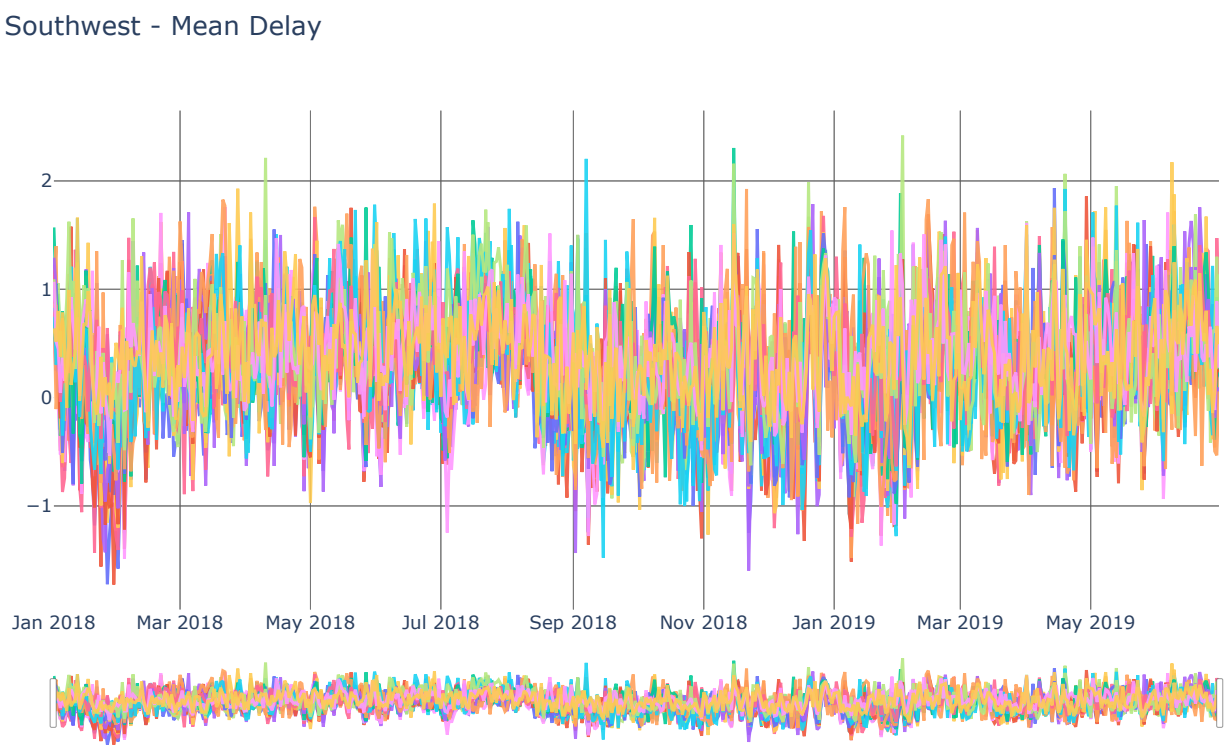


In [166]:
```
raw_subset = AIRLINE_DATA.delays[
    (AIRLINE_DATA.delays['CARRIER'] == carrier) &
    ((AIRLINE_DATA.delays['ORIGIN']).isin(X.top_airports))
]

visual(raw_subset, carrier).flight_frequency()
```

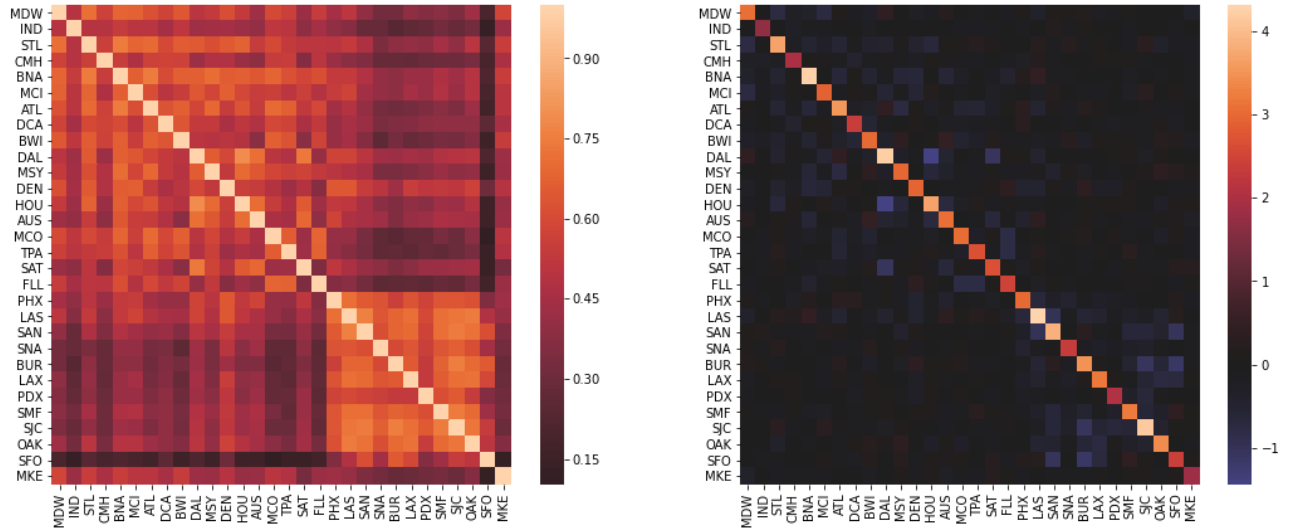
executed in 1.27s, finished 17:13:43 2020-04-12



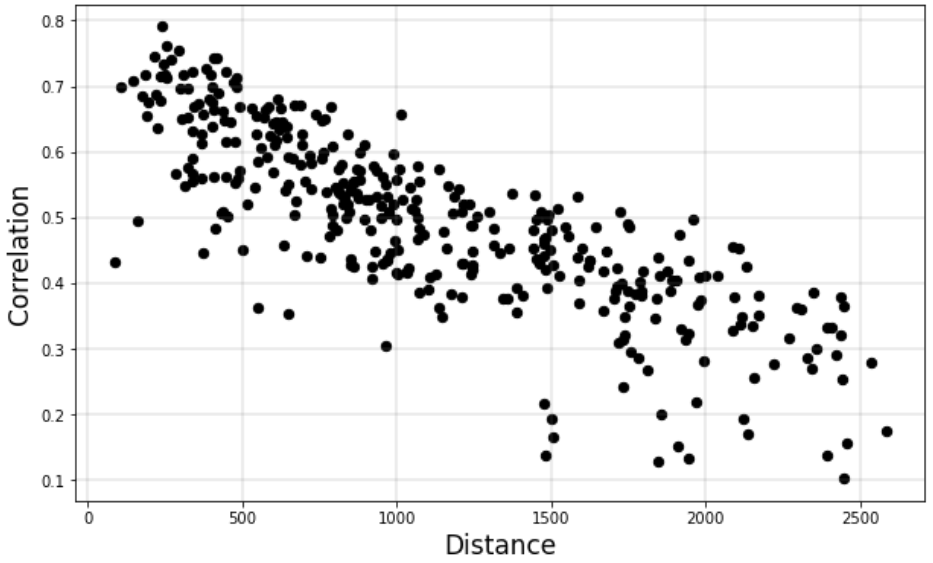
In [167]:
```
visual(matr, carrier).time_series()
```
executed in 1.57s, finished 17:13:44 2020-04-12



In [168]:
```
V = visual(delay_matr, carrier)
V.cov_matr()

# make sure to note that this is distance based from MDW
# the distance plot is relative distance
```
executed in 668ms, finished 17:13:45 2020-04-12



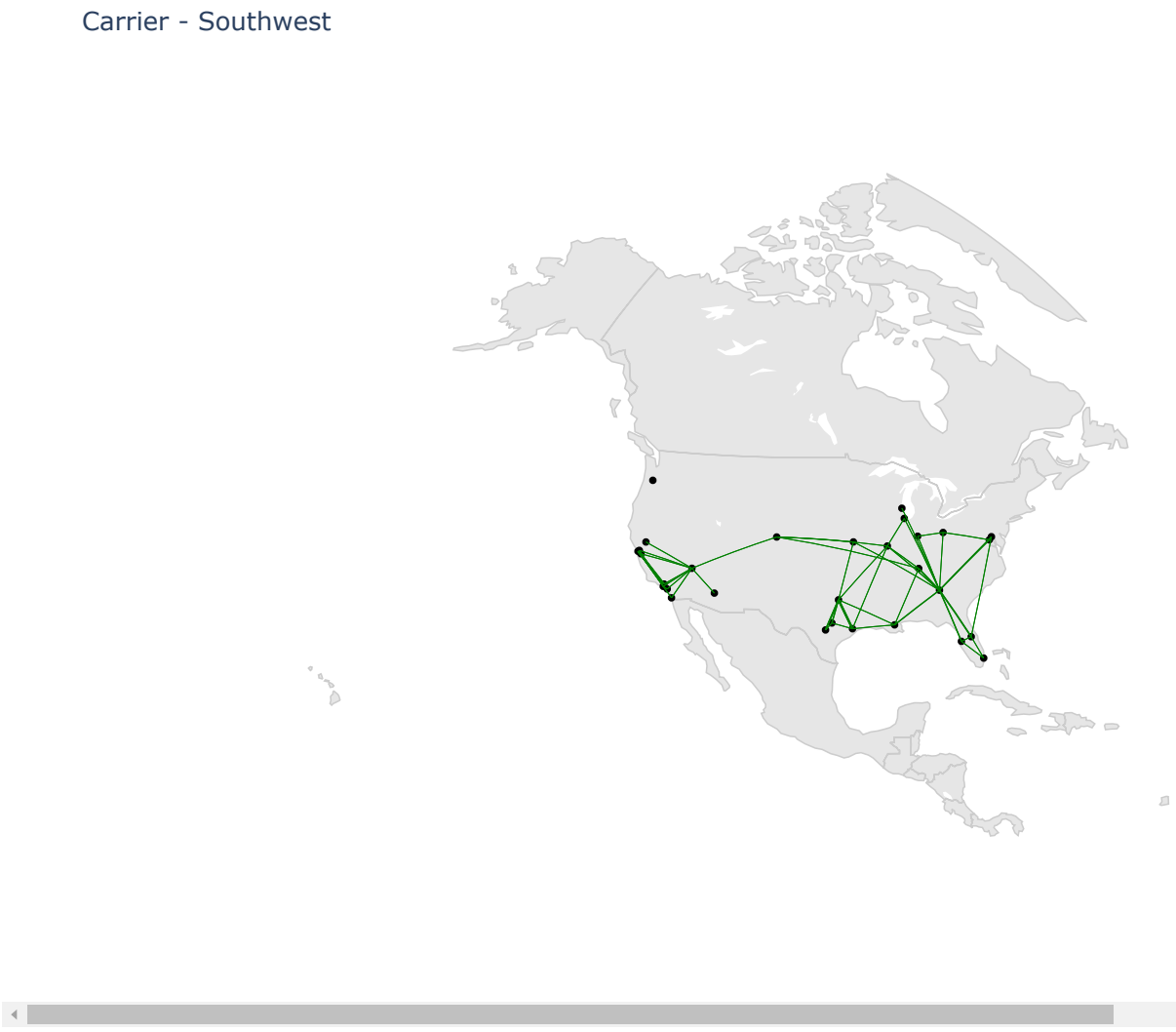
In [169]:
```
V.dist_corr()
```
executed in 686ms, finished 17:13:46 2020-04-12



In [170]:
```
MB = meinhausen_bulman(NAMES[CODES.index(carrier)], delay_matr, carrier, .2)
MB.view_graph('geographic')
```

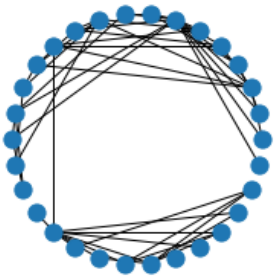
executed in 5.20s, finished 17:13:51 2020-04-12

```
C:\Users\Landon\Anaconda3\lib\site-packages\pandas\core\indexing.py:190: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
```



In [171]:
```
MB.view_graph('circular')
```
executed in 858ms, finished 17:13:52 2020-04-12

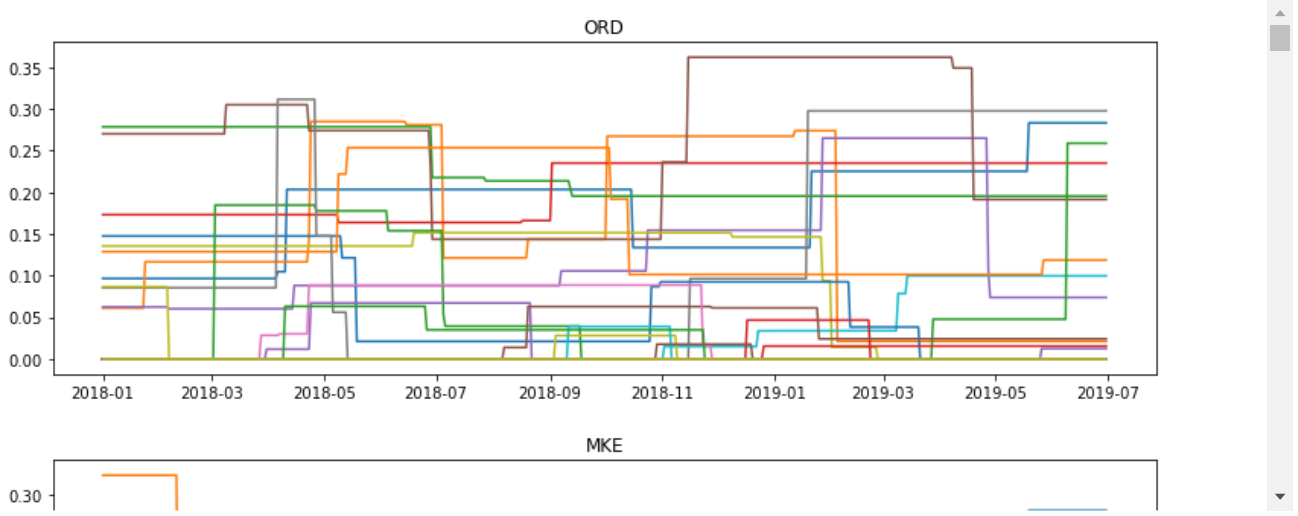
```
C:\Users\Landon\Anaconda3\lib\site-packages\networkx\drawing\nx_pylab.py:579: MatplotlibDeprecationWarning:

The iterable function was deprecated in Matplotlib 3.1 and will be removed in 3.3. Use np.iterable instead.
```



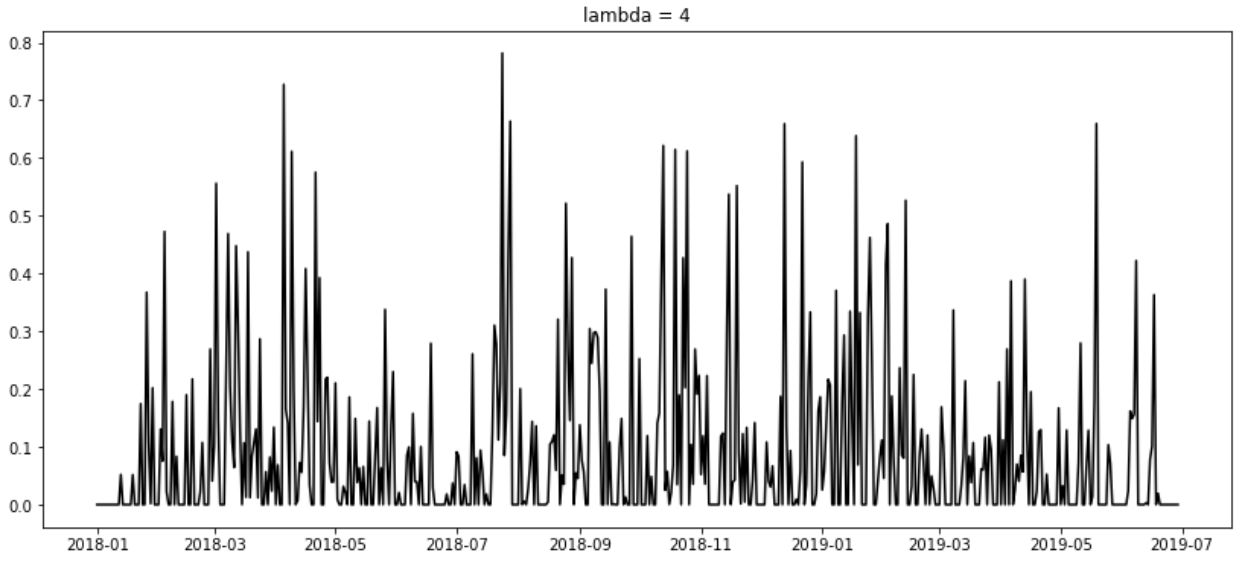
Segmentation

In [978]:
```
lambd = [.07]
alpha = [4]

model = fuse_lasso(matr, delay_matr, lambd, alpha)
```
executed in 11m 29s, finished 23:31:49 2020-04-08

In [981]:
```
omega_list = model.graphs
model.plot_paramaters(demo = False)
```
executed in 5.92s, finished 23:34:06 2020-04-08



In [983]:
```
segs = model.generate_segments(.5)['dates']
```
executed in 166ms, finished 23:34:26 2020-04-08



Vector Autoregressive Modelling

In [984]:
```
delay_matr.index = matr['datetime'].unique()

sub = matr[(matr['datetime'] >= pd.to_datetime('2018-07-01')) &
    (matr['datetime'] <  pd.to_datetime('2018-11-01'))
]

df = pd.DataFrame()
for ap in sub['airport'].unique():
    df[ap] = sub[sub['airport'] == ap]['avg_dep'].values
```
executed in 37ms, finished 23:34:32 2020-04-08

In [985]:
```
import statsmodels.api as sm
from statsmodels.tsa.api import VAR

lag = 1
vmodel = VAR(df)
results = vmodel.fit(lag)
```
executed in 7ms, finished 23:34:32 2020-04-08

Granger Causality

In [986]:
```
row = []
for ap in X.top_airports:
    other_aps = [i for i in X.top_airports if i != ap]
    if aps[0] != aps[1]:
        GC_test = results.test_causality(caused = ap, causing = other_aps).summary()
        pvalue = GC_test.data[1::2][0][-2]
        row.append({'caused': ap, 'pval': pvalue})
GC_df = pd.DataFrame(row).sort_values(by = ['pval'])
```

executed in 378ms, finished 23:34:37 2020-04-08

In [987]:
```
alpha = 0.05
signif = GC_df[GC_df['pval'] <= alpha]
signif
```

executed in 7ms, finished 23:34:37 2020-04-08

Out[987]:

	caused	pval
21	COS	0.010051
27	SBN	0.014456
29	RNO	0.022276
16	BNA	0.027764

2019 Q3 Predictions

In [143]:
```
data = pd.read_csv(path+'raw_q3.csv')
```

executed in 4.79s, finished 16:49:59 2020-04-12

In [144]:
```
Q3_raw = clean_raw_data(airports0, routes0, fares0, data)
```

executed in 2.66s, finished 16:50:01 2020-04-12

In [145]:
```
Q3_DATA = pd.read_csv(path + 'q3_delay_MATR.csv')
Q3_DATA['datetime'] = pd.to_datetime(Q3_DATA['datetime'])

Q3_DATA = Q3_DATA[
    (Q3_DATA['datetime'] >= pd.to_datetime('2019-07-01')) &
    (Q3_DATA['datetime'] <  pd.to_datetime('2019-10-01'))
]
```

executed in 202ms, finished 16:50:01 2020-04-12

In [997]:
```
num_airports = 10
carrier = 'OO'
```

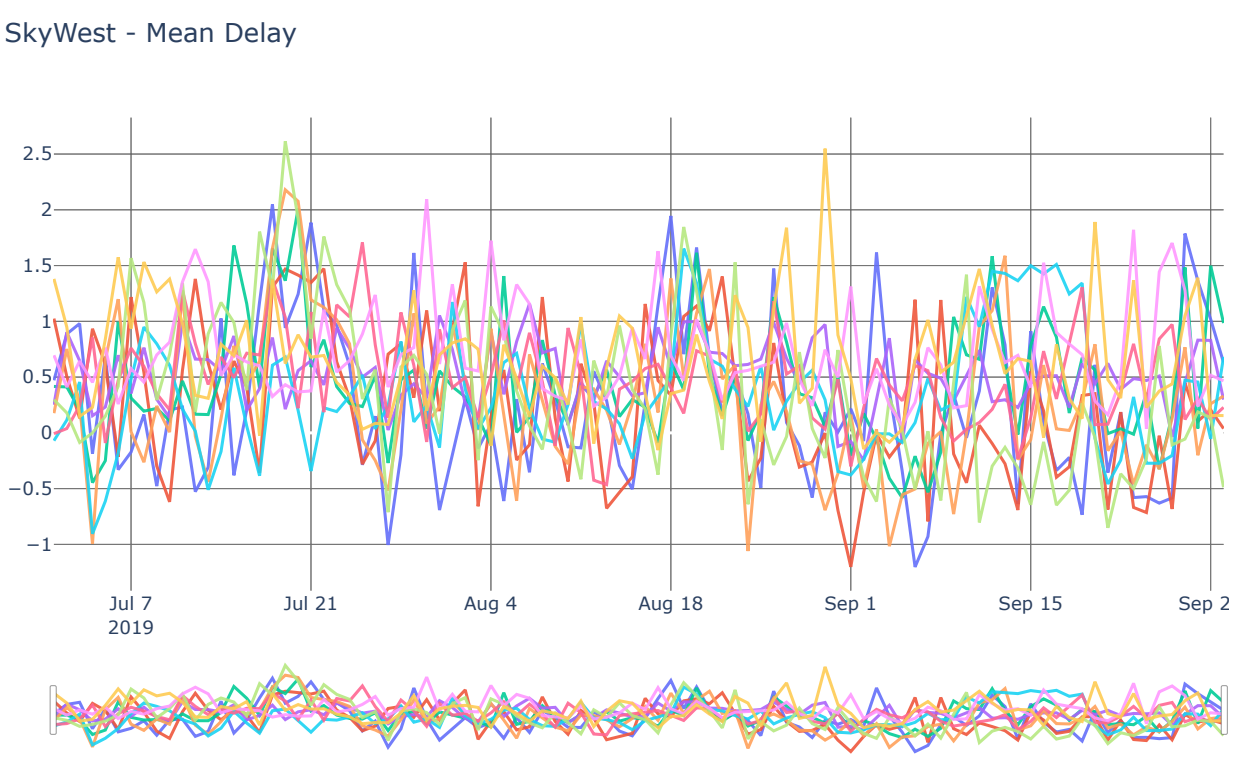
executed in 3ms, finished 23:35:30 2020-04-08

In [998]:
```
Y = prep_delay_data(Q3_DATA, carrier, num_airports)

matr_q3 = Y.matr
delay_matr_q3 = Y.delay_matr
```

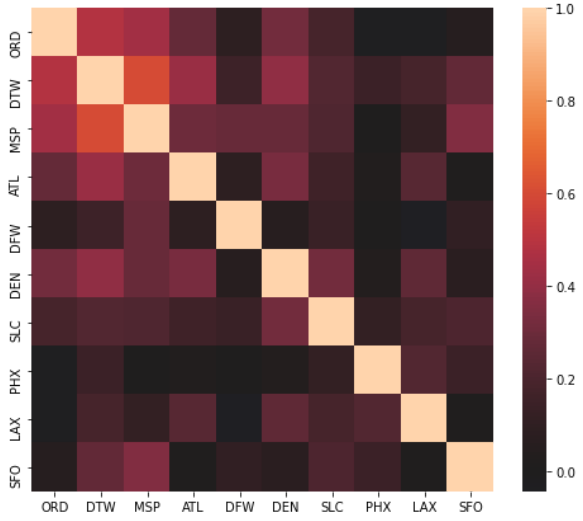
executed in 11.7s, finished 23:35:42 2020-04-08

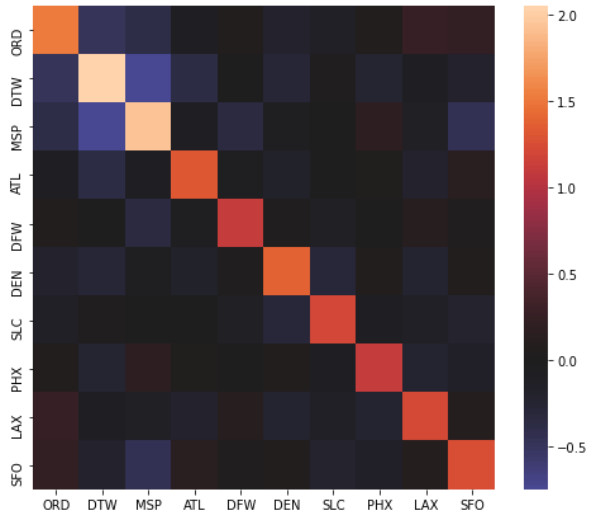
```
C:\Users\Landon\Anaconda3\lib\site-packages\ipykernel_launcher.py:117: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
```

In [999]:
```
visual(matr_q3, carrier).time_series()
```
executed in 262ms, finished 23:35:43 2020-04-08



In [1000]:
```
V = visual(delay_matr_q3, carrier)
V.cov_matr()
```
executed in 267ms, finished 23:35:43 2020-04-08





In [158]:
```
import statsmodels.api as sm
from statsmodels.tsa.api import VAR
from itertools import combinations

class VAR_model:
    def __init__(self, train, test, segments, carrier, lag = 1):
        self.trans_train = train.trans_df
        self.train_info = train.matr
        self.train_delays = train.delay_matr
        self.airports = train.delay_matr.columns

        self.trans_test = test.trans_df
        self.test_info = test.matr
        self.test_delays = test.delay_matr
        self.segments = segments

        self.carrier = carrier
        self.lag = lag


        self.train_diff = train.diff_matr
        self.test_diff = test.diff_matr

        self.predicted = self.generate_predictions()

    def estimate_VAR(self, data):
        return VAR(data).fit(self.lag)

    def transform(self, data, DIFF, Type, airport):

        if Type == 'train':

            df = self.trans_train
            params = df[df['airport'] == airport]
            new_data = np.exp(data+params['mu_log'].iloc[0])+params['shift'].iloc[0]

        elif Type == 'test':
            data = data[1:]
            DIFF = self.test_diff[airport]
            df = self.trans_test
            params = df[df['airport'] == airport]
            new_data = np.exp(DIFF+data+params['mu_log'].iloc[0])+params['shift'].iloc[0]

            #new_data = np.exp(DIFF+data+params['mu_log'].iloc[0])+params['shift'].iloc[0]
        return new_data

    def generate_predictions(self):

        X = self.train_delays
        X.index = self.train_info.datetime.unique()
        Y = self.test_delays
        Y.index = self.test_info.datetime.unique()
        time_delta = dt.timedelta(days = 365)

        diff_X = self.train_diff
        diff_X.index = self.train_info.datetime.unique()[1:]
        diff_Y = self.test_diff
        diff_Y.index = self.test_info.datetime.unique()[1:]

        rolling_preds = [] ; rolling_dates = []
        rolling_granger = []
        for i in range(len(self.segments)-1):
            dates = []
            t0 = self.segments[i]
            t1 = self.segments[i+1]
            sub_train = X[(X.index >= t0) & (X.index < t1)]
            sub_test = Y[(Y.index >= t0+time_delta) & (Y.index < t1+time_delta)]

            if len(sub_test) >= 2:
                predictions = pd.DataFrame()
                model_t = VAR(sub_train)
                results = model_t.fit(self.lag)
                for i in range(len(sub_test)-self.lag-1):
                    obs_i = sub_test.iloc[i:i+self.lag].to_numpy()
                    predictions[sub_test.index[i+self.lag]] = results.forecast(obs_i, 1)[0]
```

```
                    dates.extend(sub_test.index[i:i+self.lag])

                row = []
                for ap in self.airports:
                    other_aps = [i for i in self.airports if i != ap]
                    if ap[0] != ap[1]:
                        GC_test = results.test_causality(caused = ap, causing = other_aps).summary()
                        pvalue = GC_test.data[1::2][0][-2]
                        row.append({'caused': ap, 'pval': pvalue})
                GC_df = pd.DataFrame(row).sort_values(by = ['pval'])
                predictions.index = sub_test.columns
                predictions = predictions.T
                rolling_dates.extend(dates)
                rolling_preds.append(predictions)
                rolling_granger.append(GC_df)
        self.granger = rolling_granger
        predictions = pd.concat(rolling_preds)
        t0_new = rolling_dates[0]
        t1_new = rolling_dates[-1]
        self.observed = Y[(Y.index >= t0_new) & (Y.index < t1_new)]

        diff_X = diff_X[(diff_X.index > t0_new-time_delta)  & (diff_X.index <= t1_new-time_delta)]
        diff_Y = diff_Y[(diff_Y.index > t0_new)  & (diff_Y.index <= t1_new)]
        for ap in self.airports:
            self.observed[ap] = self.transform(self.observed[ap], None,  'train', ap)
            predictions[ap] = self.transform(predictions[ap], diff_X[ap], 'test', ap)
        predictions.index += dt.timedelta(days = 1)

        return predictions.iloc[1:]

    def plot_results(self):
        dates0 = self.observed.index
        dates1 = self.predicted.index
        for ap in self.airports:
            print(ap)
            o = self.observed[ap]
            p = self.predicted[ap]
            plt.figure(figsize = (15,7))
            plt.plot(dates0, o, c = 'k')
            plt.plot(dates1, p, c = 'r')
            plt.savefig(f'pred+{ap}.png')
            plt.show()
```

executed in 17ms, finished 17:01:52 2020-04-12

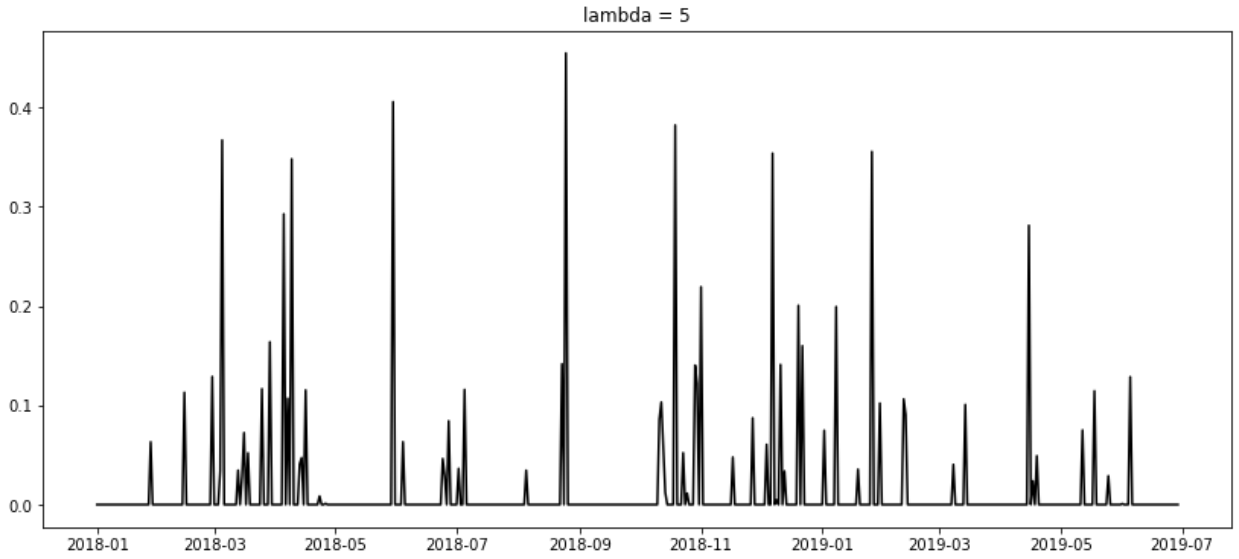
In [147]:

```
num_airports = 10
carrier = 'OO'

# training data
A = prep_delay_data(MY_DATA, carrier, num_airports)
# test data
B = prep_delay_data(Q3_DATA, carrier, num_airports)
```

executed in 22.0s, finished 16:50:24 2020-04-12

```
C:\Users\Landon\Anaconda3\lib\site-packages\ipykernel_launcher.py:116: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy)

C:\Users\Landon\Anaconda3\lib\site-packages\ipykernel_launcher.py:116: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
```

In [1004]:
```
lambd = [.05]
alpha = [5]

# for results do OO and used the fuse lasso as previously found

model = fuse_lasso(A.matr, A.delay_matr, lambd, alpha)
segments = list(model.generate_segments(.2, True)['dates'])
```
executed in 1m 8.83s, finished 23:37:34 2020-04-08

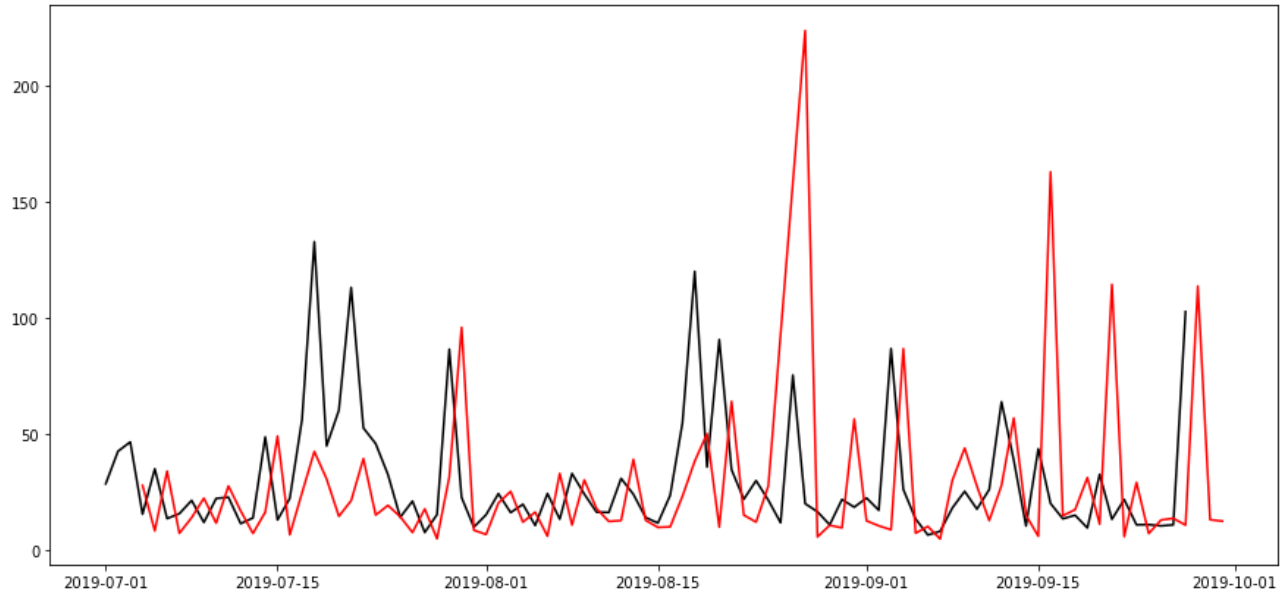


In [1005]:
```
lag = 1
var_model = VAR_model(A, B, segments, carrier, lag)
```
executed in 1.46s, finished 23:37:35 2020-04-08

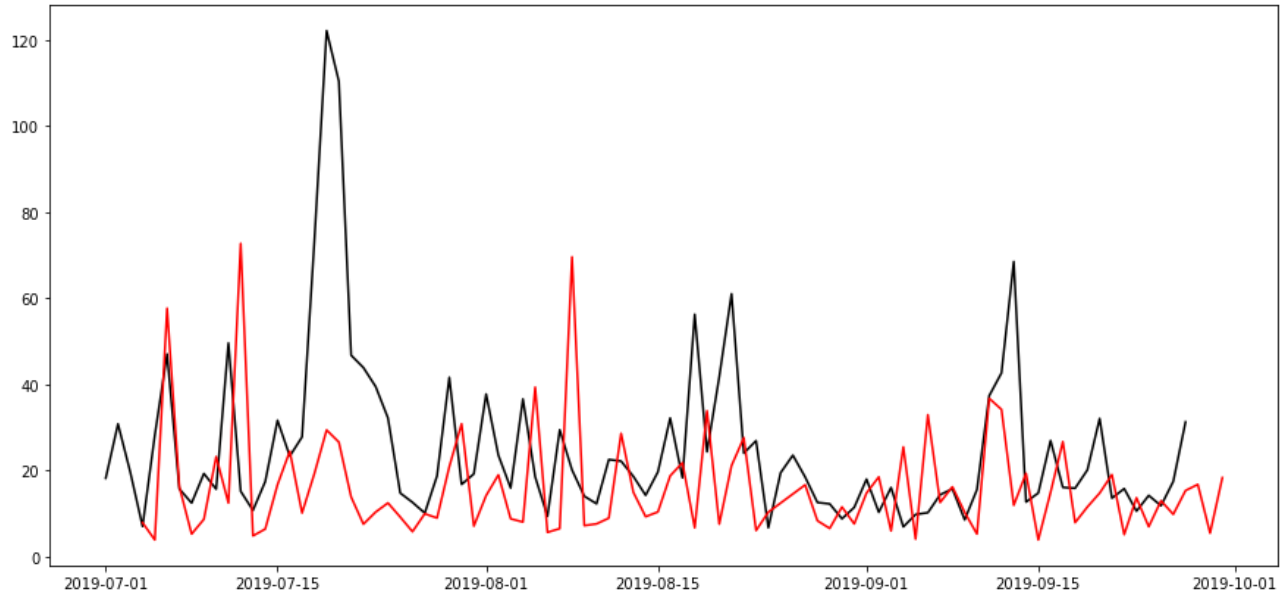
```
C:\Users\Landon\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:162: ValueWarning:

No frequency information was provided, so inferred frequency D will be used.

C:\Users\Landon\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:162: ValueWarning:

No frequency information was provided, so inferred frequency D will be used.

C:\Users\Landon\Anaconda3\lib\site-packages\ipykernel_launcher.py:104: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
```

In [1006]:
```
var_model.plot_results()
```
executed in 1.57s, finished 23:37:37 2020-04-08

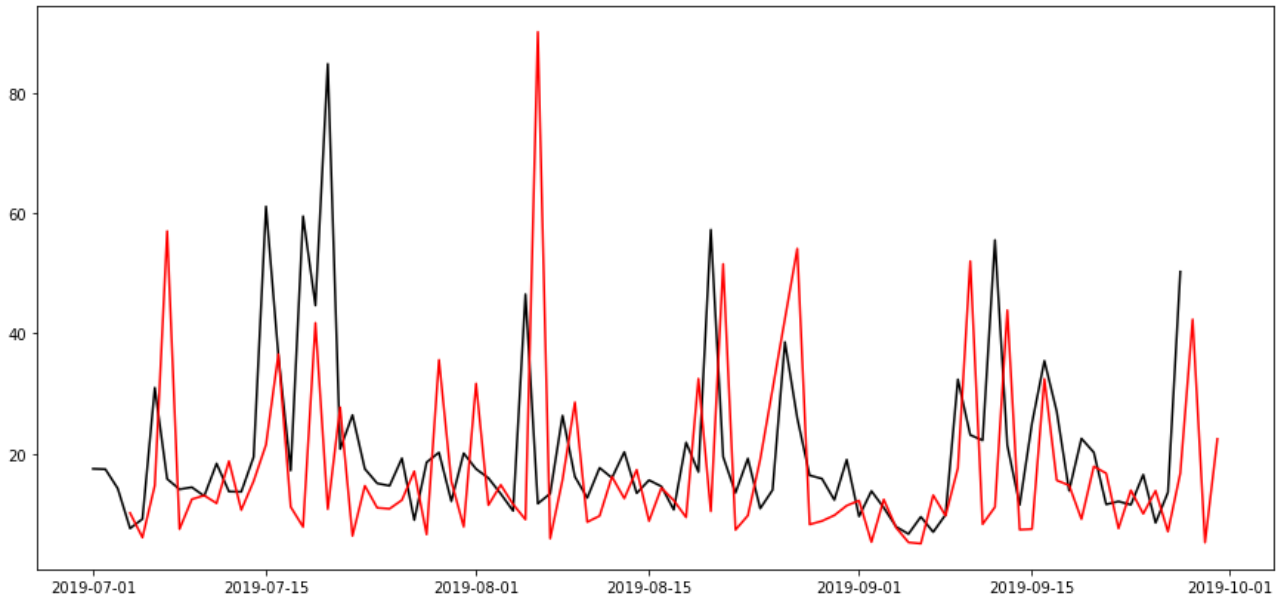
ORD



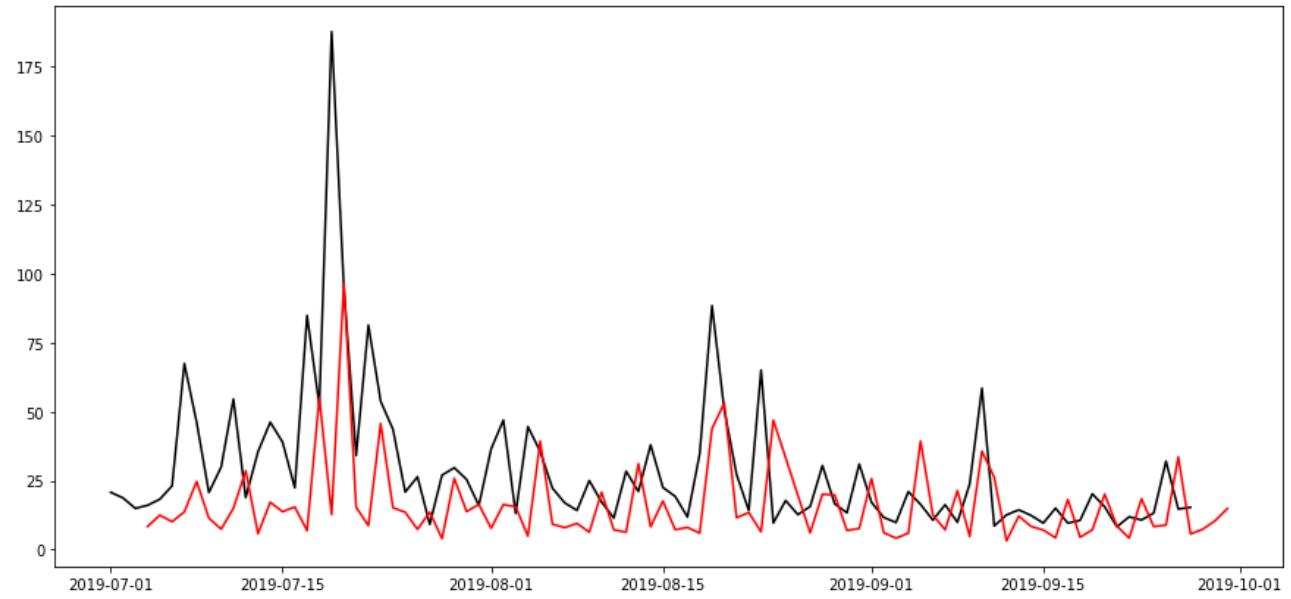
DTW



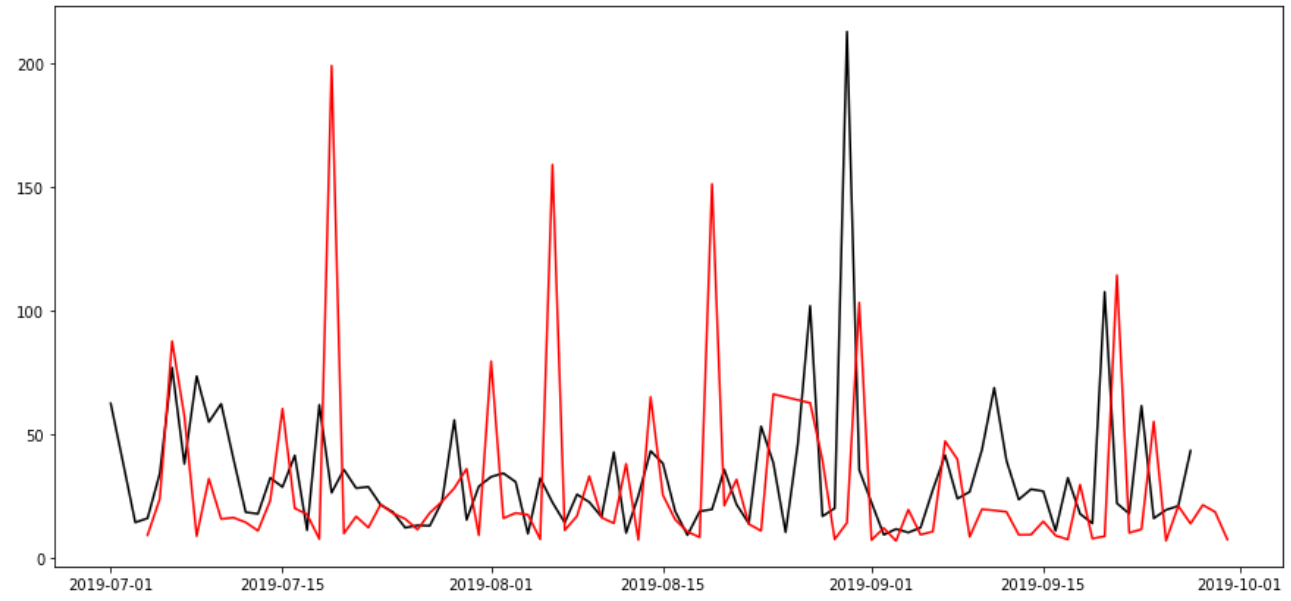
MSP


80
60
40
20
2019-07-01
2019-07-15
2019-08-01
2019-08-15
2019-09-01
2019-09-15
2019-10-01

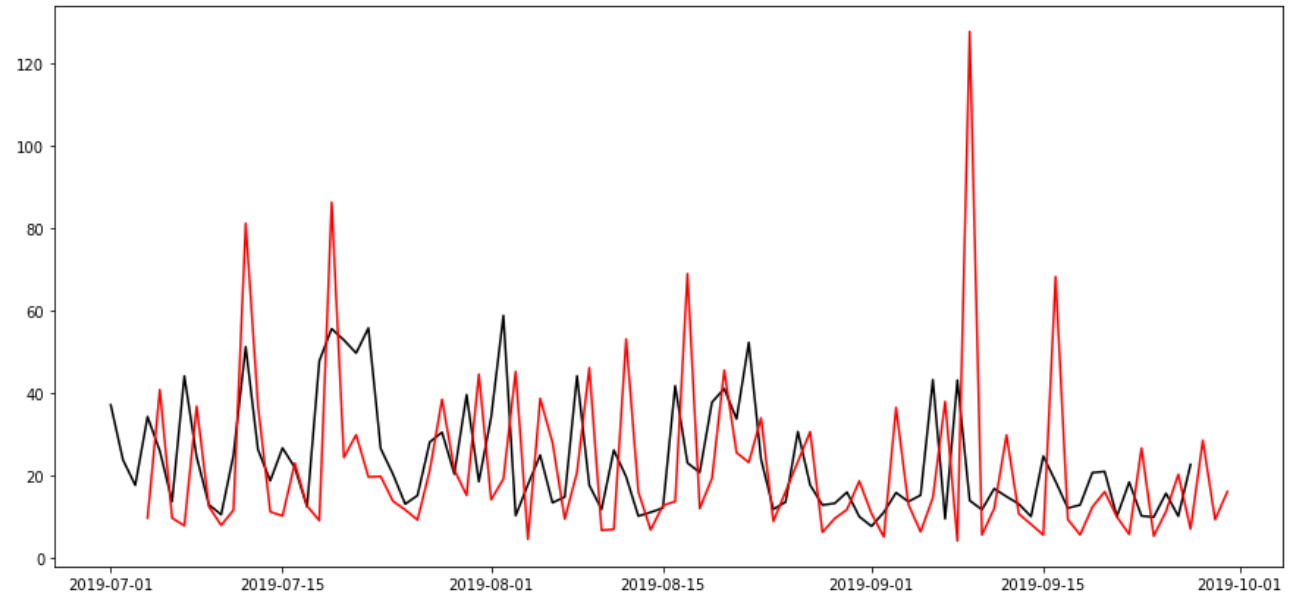
ATL


175
150
125
100
75
50
25
0
2019-07-01
2019-07-15
2019-08-01
2019-08-15
2019-09-01
2019-09-15
2019-10-01

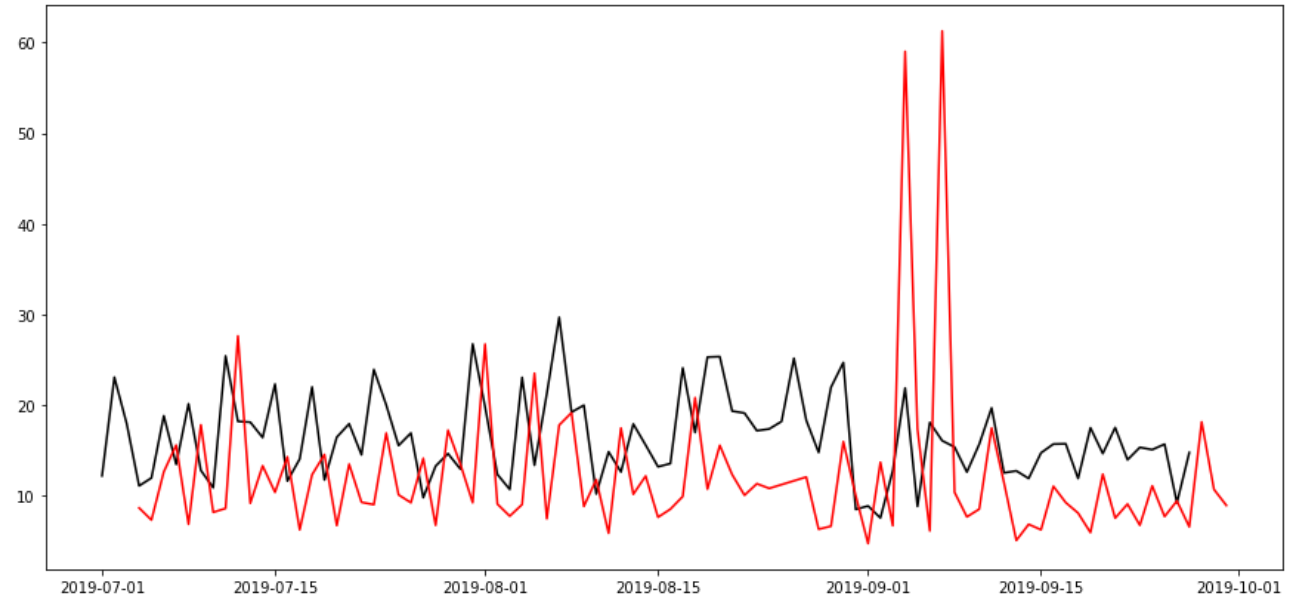
DFW


200
150
100
50
0
2019-07-01
2019-07-15
2019-08-01
2019-08-15
2019-09-01
2019-09-15
2019-10-01

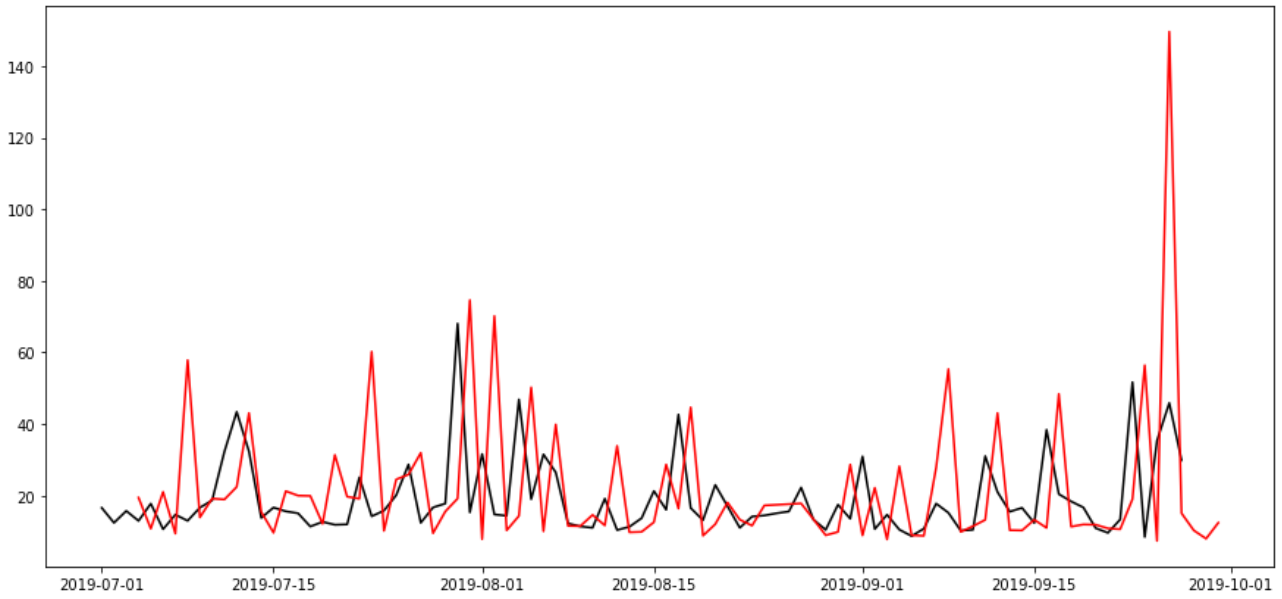
DEN


120
100
80
60
40
20
0
2019-07-01
2019-07-15
2019-08-01
2019-08-15
2019-09-01
2019-09-15
2019-10-01

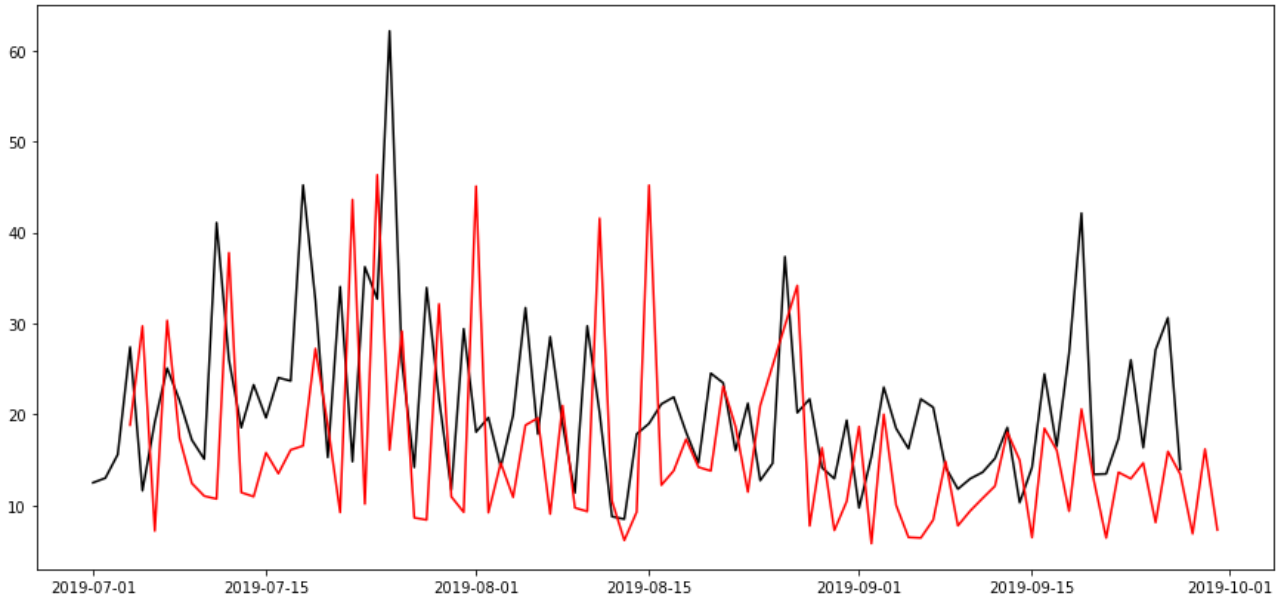
SLC


60
50
40
30
20
10
2019-07-01
2019-07-15
2019-08-01
2019-08-15
2019-09-01
2019-09-15
2019-10-01

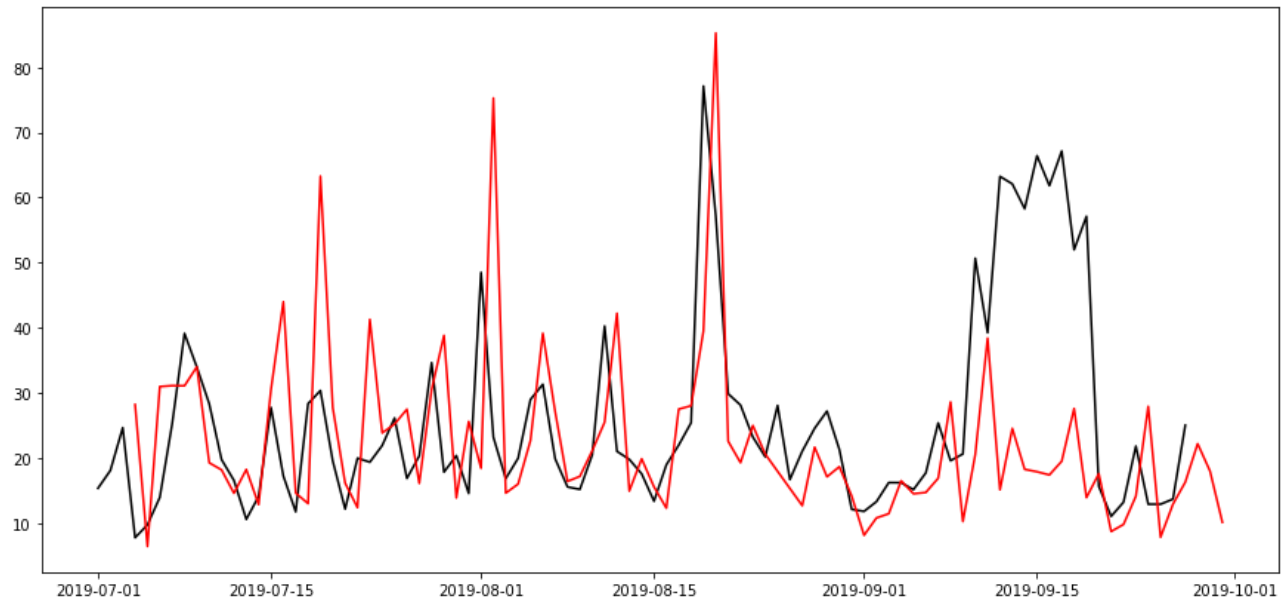
PHX


140
120
100
80
60
40
20
2019-07-01
2019-07-15
2019-08-01
2019-08-15
2019-09-01
2019-09-15
2019-10-01

LAX


60
50
40
30
20
10
2019-07-01
2019-07-15
2019-08-01
2019-08-15
2019-09-01
2019-09-15
2019-10-01

SFO


80
70
60
50
40
30
20
10
2019-07-01
2019-07-15
2019-08-01
2019-08-15
2019-09-01
2019-09-15
2019-10-01

In [162]:
```
"""df = []
for CARRIER in CODES:
    num_airports = 10
    carrier = CARRIER

    # training data
    A = prep_delay_data(MY_DATA, carrier, num_airports)
    # test data
    B = prep_delay_data(Q3_DATA, carrier, num_airports)
""
    lambd = [.05]
    alpha = [5]
    model = fuse_lasso(A.matr, A.delay_matr, lambd, alpha)
    segments = list(model.generate_segments(.02, True)['dates'])
    lag = 1
    var_model = VAR_model(A, B, segments, carrier, lag)
    pred_df = var_model.observed.iloc[3:]
    obs_df = var_model.predicted.iloc[:-3]

    print('-'*50)

    aps = A.delay_matr.columns
    for air in aps:
        k = pred_df[air].to_numpy()
        o = obs_df[air].to_numpy()

        print(air, np.square(np.subtract(k, o)).mean())
    print('-'*50)
    title = 'pred_{}.csv'.format(carrier)
    #pred_df.to_csv(title, encoding='utf-8', index=False)

    q  = var_model.observed.iloc[3:].to_numpy().flatten()
    s = var_model.predicted.iloc[:-3].to_numpy().flatten()
    mse = np.square(np.subtract(q, s)).mean()
    df.append({'ap': CARRIER, 'mse': mse})
    print(CARRIER, mse)"""
```

executed in 5ms, finished 17:13:00 2020-04-12

Out[162]:
```
'df = []\nfor CARRIER in CODES:\n    num_airports = 10\n    carrier = CARRIER\n\n    # training data\n    A = prep_delay_data(MY_DATA, carrier, num_airports)\n    # test data\n    B = prep_delay_data(Q3_DATA, carrier, num_airports)\n""\n    lambd = [.05]\n    alpha = [5]\n    model = fuse_lasso(A.matr, A.delay_matr, lambd, alpha)\n    segments = list(model.generate_segments(.02, True)[\'dates\'])\n    lag = 1\n    var_model = VAR_model(A, B, segments, carrier, lag)\n    pred_df = var_model.observed.iloc[3:]\n    obs_df = var_model.predicted.iloc[:-3]\n    \n    print(\'-\'*50)\n\n    aps = A.delay_matr.columns\n    for air in aps:\n        k = pred_df[air].to_numpy()\n        o = obs_df[air].to_numpy()\n        \n        print(air, np.square(np.subtract(k, o)).mean())\n    print(\'-\'*50)\n    title = \'pred_{}.csv\'.format(carrier)\n    #pred_df.to_csv(title, encoding=\'utf-8\', index=False)\n    \n    q  = var_model.observed.iloc[3:].to_numpy().flatten()\n    s = var_model.predicted.iloc[:-3].to_numpy().flatten()\n    mse = np.square(np.subtract(q, s)).mean()\n    df.append({\'ap\': CARRIER, \'mse\': mse})\n    print(CARRIER, mse)'
```