In [58]:
```
import datetime
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from datlib.FRED import *
from datlib.plots import *
import pandas_datareader.data as web

%matplotlib inline

# Import Statsmodels

from statsmodels.tsa.api import VAR
from statsmodels.tsa.stattools import adfuller
from statsmodels.tools.eval_measures import rmse, aic
```

In [59]:
```
#FRED.py
#. . .
def bil_to_mil(series):
    return series* 10**3
# . . .
#fedProject.py
# . . .
data_codes  = {# Assets
               "Balance Sheet: Total Assets ($ Mil)": "WALCL",
               "Balance Sheet Securities, Prem-Disc, Repos, and Loans ($ Mil)": '
               "Balance Sheet: Securities Held Outright ($ Mil)": "WSHOSHO",
               ### breakdown of securities holdings ###
               "Balance Sheet: U.S. Treasuries Held Outright ($ Mil)":"WSHOTSL",
               "Balance Sheet: Federal Agency Debt Securities ($ Mil)" : "WSHOFAD
               "Balance Sheet: Mortgage-Backed Securities ($ Mil)": "WSHOMCB",
               # other forms of lending
               "Balance Sheet: Repos ($ Mil)": "WORAL",
               "Balance Sheet: Central Bank Liquidity Swaps ($ Mil)" : "SWPT",
               "Balance Sheet: Direct Lending ($ Mil)" : "WLCFLL",
               # unamortized value of securities held (due to changes in interest
               "Balance Sheet: Unamortized Security Premiums ($ Mil)": "WUPSHO",
               # Liabilities
               "Balance Sheet: Total Liabilities ($ Mil)" : "WLTLECL",
               "Balance Sheet: Federal Reserve Notes Outstanding ($ Mil)" : "WLFN
               "Balance Sheet: Reverse Repos ($ Mil)": "WLRRAL",
               ### Major share of deposits
               "Balance Sheet: Deposits from Dep. Institutions ($ Mil)":"WLODLL",
               "Balance Sheet: U.S. Treasury General Account ($ Mil)": "WDTGAL",
               "Balance Sheet: Other Deposits ($ Mil)": "WOTHLB",
               "Balance Sheet: All Deposits ($ Mil)": "WLDLCL",
               # Capital
               "Balance Sheet: Total Capital": "WCTCL",
               # Interest Rates
               "Unemployment Rate": "UNRATE",
               "Nominal GDP ($ Bil)":"GDP",
               "Real GDP ($ Bil)":"GDPC1",
               "GDP Deflator":"GDPDEF",
               "CPI":"CPIAUCSL",
               "Core PCE":"PCEPILFE",
               "Private Investment":"GPDI",
               "Base: Total ($ Mil)": "BOGMBASE",
               "Base: Currency in Circulation ($ Bil)": "WCURCIR",
               "1 Month Treasury Rate (%)": "DGS1MO",
               "3 Month Treasury Rate (%)": "DGS3MO",
               "1 Year Treasury Rate (%)": "DGS1",
               "2 Year Treasury Rate (%)": "DGS2",
               "10 Year Treasury Rate (%)": "DGS10",
               "30 Year Treasury Rate (%)": "DGS30",
               "Effective Federal Funds Rate (%)": "DFF",
               "Federal Funds Target Rate (Pre-crisis)":"DFEDTAR",
               "Federal Funds Upper Target":"DFEDTARU",
               "Federal Funds Lower Target":"DFEDTARL",
               "Interest on Reserves (%)": "IOER",
               "VIX": "VIXCLS",
                "5 Year Forward Rate": "T5YIFR"
               }
```

```
inflation_target = 2

unemployment_target = 4
# Select start and end dates
start = datetime.datetime(2000, 1, 1)
end = datetime.datetime.today()

## year variable automatically adjusts the numper of periods
#   per year in light of data frequency
annual_div = {"Q":4,
              "W":52,
              "M":12}
### choose frequency
freq = "M"
### set periods per year
year = annual_div[freq]
```

In [60]:
```
#data cleaning, importing

d_parser = lambda x: pd.datetime.strptime(x, '%m/%d/%Y')
df = pd.read_csv('M4-3.csv', parse_dates=['Date'], date_parser=d_parser)
df
```

Out[60]:

	Date	M4	Log Total Assets	Effective Federal Funds Rate (%)	Log Currency in Circulation ($ Bil)	Unemployment Rate	Inflation Rate
0	2010-01-31	7.07	14.63	0.11	6.83	9.8	2.62
1	2010-02-28	7.06	14.63	0.13	6.83	9.8	2.15
2	2010-03-31	7.05	14.65	0.17	6.84	9.9	2.29
3	2010-04-30	7.06	14.66	0.20	6.84	9.9	2.21
4	2010-05-31	7.06	14.66	0.20	6.84	9.6	2.00
...	...	...	...	...	...	...	...
115	2019-08-31	7.40	15.14	2.13	7.47	3.7	1.74
116	2019-09-30	7.40	15.15	2.04	7.47	3.5	1.72
117	2019-10-31	7.41	15.19	1.83	7.48	3.6	1.77
118	2019-11-30	7.42	15.21	1.55	7.49	3.6	2.04
119	2019-12-31	7.42	15.23	1.55	7.49	3.6	2.26

120 rows × 7 columns

In [61]:
```
df['Date_at_year_month'] = df['Date'].dt.strftime('%Y-%m')
```

In [5]:
```
column_names = {'Date_at_year_month':'DATE',
                'M4':'M4',
                'Log Total Assets': 'TA',
                'Log Currency in Circulation ($ Bil)':'CC',
                'Effective Federal Funds Rate (%)':'FFR',
               'Unemployment Rate': 'U',
               'Inflation Rate': 'I'}

# rename columns
df = df.rename(columns = column_names)

df
```

Out[5]:

	Date	M4	TA	FFR	CC	U	I	DATE
0	2010-01-31	7.07	14.63	0.11	6.83	9.8	2.62	2010-01
1	2010-02-28	7.06	14.63	0.13	6.83	9.8	2.15	2010-02
2	2010-03-31	7.05	14.65	0.17	6.84	9.9	2.29	2010-03
3	2010-04-30	7.06	14.66	0.20	6.84	9.9	2.21	2010-04
4	2010-05-31	7.06	14.66	0.20	6.84	9.6	2.00	2010-05
...	...	...	...	...	...	...	...	...
115	2019-08-31	7.40	15.14	2.13	7.47	3.7	1.74	2019-08
116	2019-09-30	7.40	15.15	2.04	7.47	3.5	1.72	2019-09
117	2019-10-31	7.41	15.19	1.83	7.48	3.6	1.77	2019-10
118	2019-11-30	7.42	15.21	1.55	7.49	3.6	2.04	2019-11
119	2019-12-31	7.42	15.23	1.55	7.49	3.6	2.26	2019-12

120 rows × 8 columns

In [62]:
```
df = df.set_index('DATE')
df
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_15860/4232121041.py in <module>
----> 1 df = df.set_index('DATE')
      2 df

~\anaconda3\lib\site-packages\pandas\util\_decorators.py in wrapper(*args, **kw
args)
    309                     stacklevel=stacklevel,
    310                 )
--> 311             return func(*args, **kwargs)
    312
    313         return wrapper

~\anaconda3\lib\site-packages\pandas\core\frame.py in set_index(self, keys, dro
p, append, inplace, verify_integrity)
   5449
   5450         if missing:
-> 5451             raise KeyError(f"None of {missing} are in the columns")
   5452
   5453         if inplace:

KeyError: "None of ['DATE'] are in the columns"
```

In [ ]:
```
df = df.drop(['Date'], axis = 1)
df
```

In [ ]:
```
data = df
data
```

In [ ]:
```
data.isnull().sum()
```

In [ ]:
```
## 1st diff
data_diff = data.diff().dropna()
data_diff
```

In [ ]:
```
#ADF test

X = data_diff["M4"].values
result = adfuller(X)
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
print('Critical Values:')
for key, value in result[4].items():
    print('\t%s: %.3f' % (key, value))

if result[0] < result[4]["5%"]:
    print ("Reject Ho - Time Series is Stationary")
else:
    print ("Failed to Reject Ho - Time Series is Non-Stationary")


X = data_diff["FFR"].values
result = adfuller(X)
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
print('Critical Values:')
for key, value in result[4].items():
    print('\t%s: %.3f' % (key, value))

if result[0] < result[4]["5%"]:
    print ("Reject Ho - Time Series is Stationary")
else:
    print ("Failed to Reject Ho - Time Series is Non-Stationary")


X = data_diff["TA"].values
result = adfuller(X)
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
print('Critical Values:')
for key, value in result[4].items():
    print('\t%s: %.3f' % (key, value))

if result[0] < result[4]["5%"]:
    print ("Reject Ho - Time Series is Stationary")
else:
    print ("Failed to Reject Ho - Time Series is Non-Stationary")


X = data_diff["CC"].values
result = adfuller(X)
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
print('Critical Values:')
for key, value in result[4].items():
    print('\t%s: %.3f' % (key, value))

if result[0] < result[4]["5%"]:
    print ("Reject Ho - Time Series is Stationary")
else:
    print ("Failed to Reject Ho - Time Series is Non-Stationary")
```

```
X = data_diff["U"].values
result = adfuller(X)
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
print('Critical Values:')
for key, value in result[4].items():
    print('\t%s: %.3f' % (key, value))

if result[0] < result[4]["5%"]:
    print ("Reject Ho - Time Series is Stationary")
else:
    print ("Failed to Reject Ho - Time Series is Non-Stationary")

X = data_diff["I"].values
result = adfuller(X)
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
print('Critical Values:')
for key, value in result[4].items():
    print('\t%s: %.3f' % (key, value))

if result[0] < result[4]["5%"]:
    print ("Reject Ho - Time Series is Stationary")
else:
    print ("Failed to Reject Ho - Time Series is Non-Stationary")
```

In [ ]:
```
##2nd diff
data_new = data_diff.diff().dropna()
data_new
```

In [ ]:
```
#ADF test

X = data_new["M4"].values
result = adfuller(X)
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
print('Critical Values:')
for key, value in result[4].items():
    print('\t%s: %.3f' % (key, value))

if result[0] < result[4]["5%"]:
    print ("Reject Ho - Time Series is Stationary")
else:
    print ("Failed to Reject Ho - Time Series is Non-Stationary")


X = data_new["FFR"].values
result = adfuller(X)
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
print('Critical Values:')
for key, value in result[4].items():
    print('\t%s: %.3f' % (key, value))

if result[0] < result[4]["5%"]:
    print ("Reject Ho - Time Series is Stationary")
else:
    print ("Failed to Reject Ho - Time Series is Non-Stationary")


X = data_new["TA"].values
result = adfuller(X)
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
print('Critical Values:')
for key, value in result[4].items():
    print('\t%s: %.3f' % (key, value))

if result[0] < result[4]["5%"]:
    print ("Reject Ho - Time Series is Stationary")
else:
    print ("Failed to Reject Ho - Time Series is Non-Stationary")


X = data_new["CC"].values
result = adfuller(X)
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
print('Critical Values:')
for key, value in result[4].items():
    print('\t%s: %.3f' % (key, value))

if result[0] < result[4]["5%"]:
    print ("Reject Ho - Time Series is Stationary")
else:
    print ("Failed to Reject Ho - Time Series is Non-Stationary")
```

```
X = data_new["U"].values
result = adfuller(X)
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
print('Critical Values:')
for key, value in result[4].items():
    print('\t%s: %.3f' % (key, value))

if result[0] < result[4]["5%"]:
    print ("Reject Ho - Time Series is Stationary")
else:
    print ("Failed to Reject Ho - Time Series is Non-Stationary")

X = data_new["I"].values
result = adfuller(X)
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
print('Critical Values:')
for key, value in result[4].items():
    print('\t%s: %.3f' % (key, value))

if result[0] < result[4]["5%"]:
    print ("Reject Ho - Time Series is Stationary")
else:
    print ("Failed to Reject Ho - Time Series is Non-Stationary")
```

In [ ]:
```
df = data_new
```

In [26]:
```
from statsmodels.tsa.stattools import kpss

def kpss_test(df):
    statistic, p_value, n_lags, critical_values = kpss(df.values)

    print(f'KPSS Statistic: {statistic}')
    print(f'p-value: {p_value}')
    print(f'num lags: {n_lags}')
    print('Critial Values:')
    for key, value in critical_values.items():
        print(f'   {key} : {value}')

print('KPSS Test: M4')
kpss_test(df['M4'])
print('KPSS Test: TA')
kpss_test(df['TA'])
print('KPSS Test: FFR')
kpss_test(df['FFR'])
print('KPSS Test: U')
kpss_test(df['U'])
print('KPSS Test: CC')
kpss_test(df['CC'])
```

```
KPSS Test: M4
KPSS Statistic: 0.14964284945890138
p-value: 0.1
num lags: 15
Critial Values:
   10% : 0.347
   5% : 0.463
   2.5% : 0.574
   1% : 0.739
KPSS Test: TA
KPSS Statistic: 0.08493068113321174
p-value: 0.1
num lags: 13
Critial Values:
   10% : 0.347
   5% : 0.463
   2.5% : 0.574
   1% : 0.739
KPSS Test: FFR
KPSS Statistic: 0.07686568226713045
p-value: 0.1
num lags: 19
Critial Values:
   10% : 0.347
   5% : 0.463
   2.5% : 0.574
   1% : 0.739
KPSS Test: U
KPSS Statistic: 0.19969793589528864
p-value: 0.1
num lags: 34
Critial Values:
   10% : 0.347
   5% : 0.463
```

```
   2.5% : 0.574
   1% : 0.739
KPSS Test: CC
KPSS Statistic: 0.13001240181893342
p-value: 0.1
num lags: 16
Critial Values:
   10% : 0.347
   5% : 0.463
   2.5% : 0.574
   1% : 0.739
```

In [38]:
```
from arch.unitroot import engle_granger

eg_test = engle_granger(df.M4, df.TA, trend="n")
eg_test
```

Out[38]:

Engle-Granger Cointegration Test

Test Statistic	-8.810
P-value	0.000
ADF Lag length	4
Estimated Root ρ (γ+1)	-3.227

Trend: Constant
Critical Values: -2.50 (10%), -2.82 (5%), -3.44 (1%)
Null Hypothesis: No Cointegration
Alternative Hypothesis: Cointegration
Distribution Order: 1

In [39]:
```
eg_test.cointegrating_vector
```

Out[39]:
```
M4    1.00000
TA    0.12931
dtype: float64
```

In [40]:
```
eg_test = engle_granger(df.M4, df.CC, trend="n")
eg_test
```

Out[40]:

Engle-Granger Cointegration Test

Test Statistic	-9.118
P-value	0.000
ADF Lag length	4
Estimated Root ρ (γ+1)	-3.392

Trend: Constant
Critical Values: -2.50 (10%), -2.82 (5%), -3.44 (1%)
Null Hypothesis: No Cointegration
Alternative Hypothesis: Cointegration
Distribution Order: 1

In [41]:
```
eg_test.cointegrating_vector
```

Out[41]:
```
M4    1.000000
CC    0.173077
dtype: float64
```

In [42]:
```
eg_test = engle_granger(df.M4, df.FFR, trend="n")
eg_test
```

Out[42]:

Engle-Granger Cointegration Test

Test Statistic	-9.005
P-value	0.000
ADF Lag length	3
Estimated Root ρ (γ+1)	-2.379

Trend: Constant
Critical Values: -2.50 (10%), -2.82 (5%), -3.43 (1%)
Null Hypothesis: No Cointegration
Alternative Hypothesis: Cointegration
Distribution Order: 1

In [43]:
```
eg_test.cointegrating_vector
```

Out[43]:
```
M4     1.000000
FFR    0.034614
dtype: float64
```

In [44]:
```
eg_test = engle_granger(df.M4, df.U, trend="n")
eg_test
```

Out[44]:

Engle-Granger Cointegration Test

Test Statistic	-9.169
P-value	0.000
ADF Lag length	4
Estimated Root ρ (γ+1)	-3.576

Trend: Constant
Critical Values: -2.50 (10%), -2.82 (5%), -3.44 (1%)
Null Hypothesis: No Cointegration
Alternative Hypothesis: Cointegration
Distribution Order: 1

In [45]:
```
eg_test.cointegrating_vector
```

Out[45]:
```
M4    1.000000
U    -0.005612
dtype: float64
```

In [ ]:

In [46]:
```
eg_test = engle_granger(df.TA, df.U, trend="n")
eg_test
```

Out[46]:

Engle-Granger Cointegration Test

Test Statistic	-10.241
P-value	0.000
ADF Lag length	1
Estimated Root ρ (γ+1)	-0.511

Trend: Constant
Critical Values: -2.50 (10%), -2.82 (5%), -3.43 (1%)
Null Hypothesis: No Cointegration
Alternative Hypothesis: Cointegration
Distribution Order: 1

In [47]:
```
eg_test.cointegrating_vector
```

Out[47]:
```
TA    1.000000
U     0.001531
dtype: float64
```

In [48]:
```
eg_test = engle_granger(df.TA, df.CC, trend="n")
eg_test
```

Out[48]:

Engle-Granger Cointegration Test

Test Statistic	-14.804
P-value	0.000
ADF Lag length	0
Estimated Root ρ (γ+1)	-0.292

Trend: Constant
Critical Values: -2.50 (10%), -2.82 (5%), -3.43 (1%)
Null Hypothesis: No Cointegration
Alternative Hypothesis: Cointegration
Distribution Order: 1

In [49]:
```
eg_test.cointegrating_vector
```

Out[49]:
```
TA    1.000000
CC   -0.134615
dtype: float64
```

In [50]:
```
eg_test = engle_granger(df.TA, df.FFR, trend="n")
eg_test
```

Out[50]:

Engle-Granger Cointegration Test

Test Statistic	-14.499
P-value	0.000
ADF Lag length	0
Estimated Root ρ (γ+1)	-0.272

Trend: Constant
Critical Values: -2.50 (10%), -2.82 (5%), -3.43 (1%)
Null Hypothesis: No Cointegration
Alternative Hypothesis: Cointegration
Distribution Order: 1

In [51]:
```
eg_test.cointegrating_vector
```

Out[51]:
```
TA     1.000000
FFR   -0.006524
dtype: float64
```

In [52]:
```
eg_test = engle_granger(df.CC, df.U, trend="n")
eg_test
```

Out[52]:

Engle-Granger Cointegration Test

Test Statistic	-14.863
P-value	0.000
ADF Lag length	2
Estimated Root ρ (γ+1)	-2.476

Trend: Constant
Critical Values: -2.50 (10%), -2.82 (5%), -3.43 (1%)
Null Hypothesis: No Cointegration
Alternative Hypothesis: Cointegration
Distribution Order: 1

In [53]:
```
eg_test.cointegrating_vector
```

Out[53]:
```
CC    1.000000
U     0.002721
dtype: float64
```

In [35]:
```
eg_test = engle_granger(df.CC, df.FFR, trend="n")
eg_test
```

Out[35]:

Engle-Granger Cointegration Test

Test Statistic	-14.755
P-value	0.000
ADF Lag length	2
Estimated Root ρ (γ+1)	-2.454

Trend: Constant
Critical Values: -2.50 (10%), -2.82 (5%), -3.43 (1%)
Null Hypothesis: No Cointegration
Alternative Hypothesis: Cointegration
Distribution Order: 1

In [54]:
```
eg_test.cointegrating_vector
```

Out[54]:
```
CC    1.000000
U     0.002721
dtype: float64
```

In [36]:
```
eg_test = engle_granger(df.CC, df.TA, trend="n")
eg_test
```

Out[36]:

Engle-Granger Cointegration Test

Test Statistic	-14.654
P-value	0.000
ADF Lag length	2
Estimated Root ρ (γ+1)	-2.449

Trend: Constant
Critical Values: -2.50 (10%), -2.82 (5%), -3.43 (1%)
Null Hypothesis: No Cointegration
Alternative Hypothesis: Cointegration
Distribution Order: 1

In [55]:
```
eg_test.cointegrating_vector
```

Out[55]:
```
CC    1.000000
U     0.002721
dtype: float64
```

In [56]:
```
eg_test = engle_granger(df.FFR, df.U, trend="n")
eg_test
```

Out[56]:

Engle-Granger Cointegration Test

Test Statistic	-15.965
P-value	0.000
ADF Lag length	1
Estimated Root ρ (γ+1)	-1.303

Trend: Constant
Critical Values: -2.50 (10%), -2.82 (5%), -3.43 (1%)
Null Hypothesis: No Cointegration
Alternative Hypothesis: Cointegration
Distribution Order: 1

In [57]:
```
eg_test.cointegrating_vector
```

Out[57]:
```
FFR    1.00000
U      0.00102
dtype: float64
```

In [65]:
```
## Partial Correlation

import statsmodels.api as sm

residuals = {}
for y_var in df.keys():
    X_vars = list(df.keys())
    X_vars.remove(y_var)
    X = df[X_vars]
    # Initial estimate should include constant
    #   This won't be the case we regress the errors
    X["Constant"] = 1
    # pass y_var as list for consistent structure
    y = df[[y_var]]
    model = sm.OLS(y, X)
    results = model.fit()
    residuals[y_var] = results.resid
residuals = pd.DataFrame(residuals)
residuals
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_15860/4005149703.py in <module>
     13     # pass y_var as list for consistent structure
     14     y = df[[y_var]]
---> 15     model = sm.OLS(y, X)
     16     results = model.fit()
     17     residuals[y_var] = results.resid

~\anaconda3\lib\site-packages\statsmodels\regression\linear_model.py in __init_
_(self, endog, exog, missing, hasconst, **kwargs)
    888                     "An exception will be raised in the next version.")
    889             warnings.warn(msg, ValueWarning)
--> 890         super(OLS, self).__init__(endog, exog, missing=missing,
    891                                   hasconst=hasconst, **kwargs)
    892         if "weights" in self._init_keys:

~\anaconda3\lib\site-packages\statsmodels\regression\linear_model.py in __init_
_(self, endog, exog, weights, missing, hasconst, **kwargs)
    715         else:
    716             weights = weights.squeeze()
--> 717         super(WLS, self).__init__(endog, exog, missing=missing,
    718                                   weights=weights, hasconst=hasconst, *
*kwargs)
    719         nobs = self.exog.shape[0]

~\anaconda3\lib\site-packages\statsmodels\regression\linear_model.py in __init_
_(self, endog, exog, **kwargs)
    189     """
    190     def __init__(self, endog, exog, **kwargs):
--> 191         super(RegressionModel, self).__init__(endog, exog, **kwargs)
    192         self._data_attr.extend(['pinv_wexog', 'wendog', 'wexog', 'weigh
ts'])
    193

~\anaconda3\lib\site-packages\statsmodels\base\model.py in __init__(self, endo
```

```
g, exog, **kwargs)
    265
    266     def __init__(self, endog, exog=None, **kwargs):
--> 267         super().__init__(endog, exog, **kwargs)
    268         self.initialize()
    269

~\anaconda3\lib\site-packages\statsmodels\base\model.py in __init__(self, endo
g, exog, **kwargs)
     90         missing = kwargs.pop('missing', 'none')
     91         hasconst = kwargs.pop('hasconst', None)
---> 92         self.data = self._handle_data(endog, exog, missing, hasconst,
     93                                       **kwargs)
     94         self.k_constant = self.data.k_constant

~\anaconda3\lib\site-packages\statsmodels\base\model.py in _handle_data(self, e
ndog, exog, missing, hasconst, **kwargs)
    130
    131     def _handle_data(self, endog, exog, missing, hasconst, **kwargs):
--> 132         data = handle_data(endog, exog, missing, hasconst, **kwargs)
    133         # kwargs arrays could have changed, easier to just attach here
    134         for key in kwargs:

~\anaconda3\lib\site-packages\statsmodels\base\data.py in handle_data(endog, ex
og, missing, hasconst, **kwargs)
    671
    672     klass = handle_data_class_factory(endog, exog)
--> 673     return klass(endog, exog=exog, missing=missing, hasconst=hasconst,
    674                  **kwargs)

~\anaconda3\lib\site-packages\statsmodels\base\data.py in __init__(self, endog,
exog, missing, hasconst, **kwargs)
     80             self.orig_endog = endog
     81             self.orig_exog = exog
---> 82             self.endog, self.exog = self._convert_endog_exog(endog, exo
g)
     83
     84         self.const_idx = None

~\anaconda3\lib\site-packages\statsmodels\base\data.py in _convert_endog_exog(s
elf, endog, exog)
    505         exog = exog if exog is None else np.asarray(exog)
    506         if endog.dtype == object or exog is not None and exog.dtype ==
 object:
--> 507             raise ValueError("Pandas data cast to numpy dtype of objec
t. "
    508                              "Check input data with np.asarray(data).")
    509         return super(PandasData, self)._convert_endog_exog(endog, exog)

ValueError: Pandas data cast to numpy dtype of object. Check input data with n
p.asarray(data).
```

In [64]:
```
residuals.corr()[residuals.corr().abs() < 1].mul(-1).fillna(1).round(2)
```

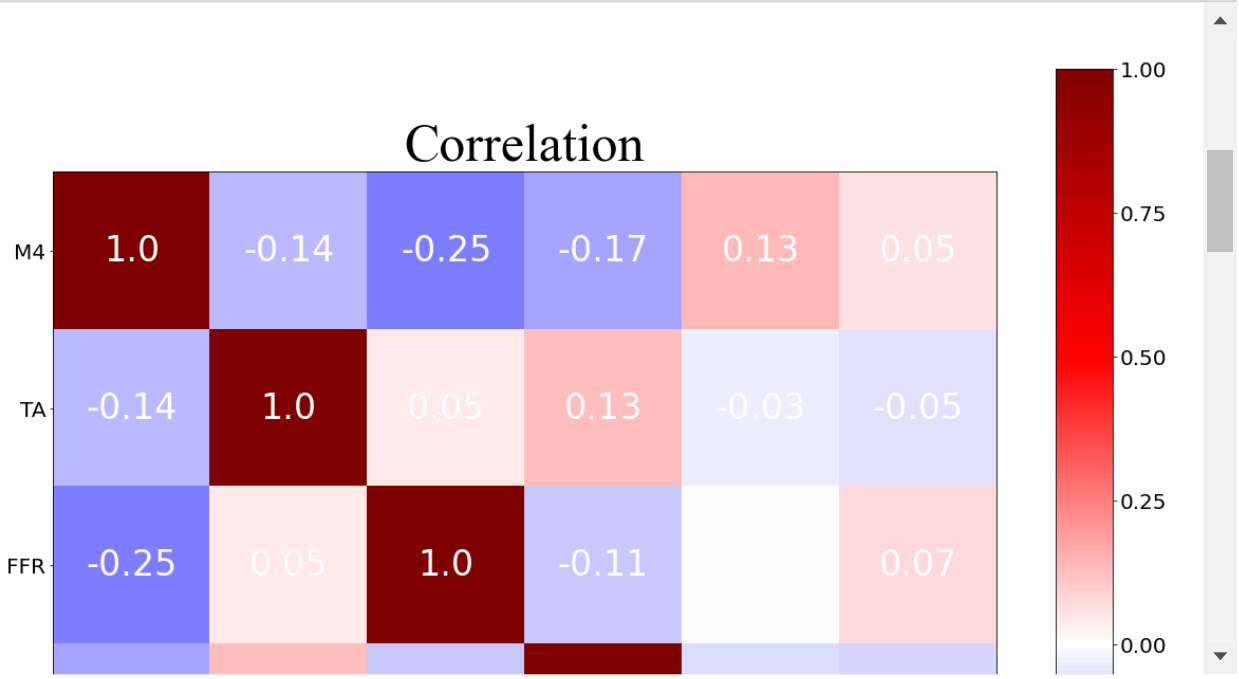
```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_15860/1867714976.py in <module>
----> 1 residuals.corr()[residuals.corr().abs() < 1].mul(-1).fillna(1).round(2)

AttributeError: 'dict' object has no attribute 'corr'
```

In [17]:
```
# !pip install pingouin
import pingouin
df.pcorr().round(2)
```

Out[17]:

	M4	TA	FFR	CC	U	I
M4	1.00	-0.10	-0.28	-0.18	0.13	0.07
TA	-0.10	1.00	0.03	0.11	-0.02	-0.04
FFR	-0.28	0.03	1.00	-0.15	0.03	0.08
CC	-0.18	0.11	-0.15	1.00	-0.04	-0.06
U	0.13	-0.02	0.03	-0.04	1.00	-0.12
I	0.07	-0.04	0.08	-0.06	-0.12	1.00

In [18]:
```
from datlib.plots import *
corr_matrix_heatmap(df.corr(),
                    save_fig = False,
                    pp = None)
corr_matrix_heatmap(df.pcorr(), save_fig = False, pp = None)
```



In [19]:
```
pcorr_pvalues = {}
for y, Y in residuals.items():
    pcorr_pvalues[y] = {}
    for x, X in residuals.items():
        if x != y:
            pcorr_pvalues[y][x] = sm.OLS(Y,X).fit().pvalues[x]

        else:
            pcorr_pvalues[y][x] = np.NaN
pd.DataFrame(pcorr_pvalues).round(2)
```

Out[19]:

	M4	TA	FFR	CC	U	I
M4	NaN	0.29	0.00	0.05	0.16	0.44
TA	0.29	NaN	0.73	0.25	0.84	0.63
FFR	0.00	0.73	NaN	0.10	0.71	0.37
CC	0.05	0.25	0.10	NaN	0.64	0.52
U	0.16	0.84	0.71	0.64	NaN	0.20
I	0.44	0.63	0.37	0.52	0.20	NaN

In [20]:
```
##DAG

import pingouin
from pgmpy.estimators import PC
import matplotlib.pyplot as plt
from matplotlib.patches import ArrowStyle
from networkx.drawing.nx_agraph import graphviz_layout
import warnings
warnings.filterwarnings("ignore")
from matplotlib.backends.backend_pdf import PdfPages
import networkx as nx
```

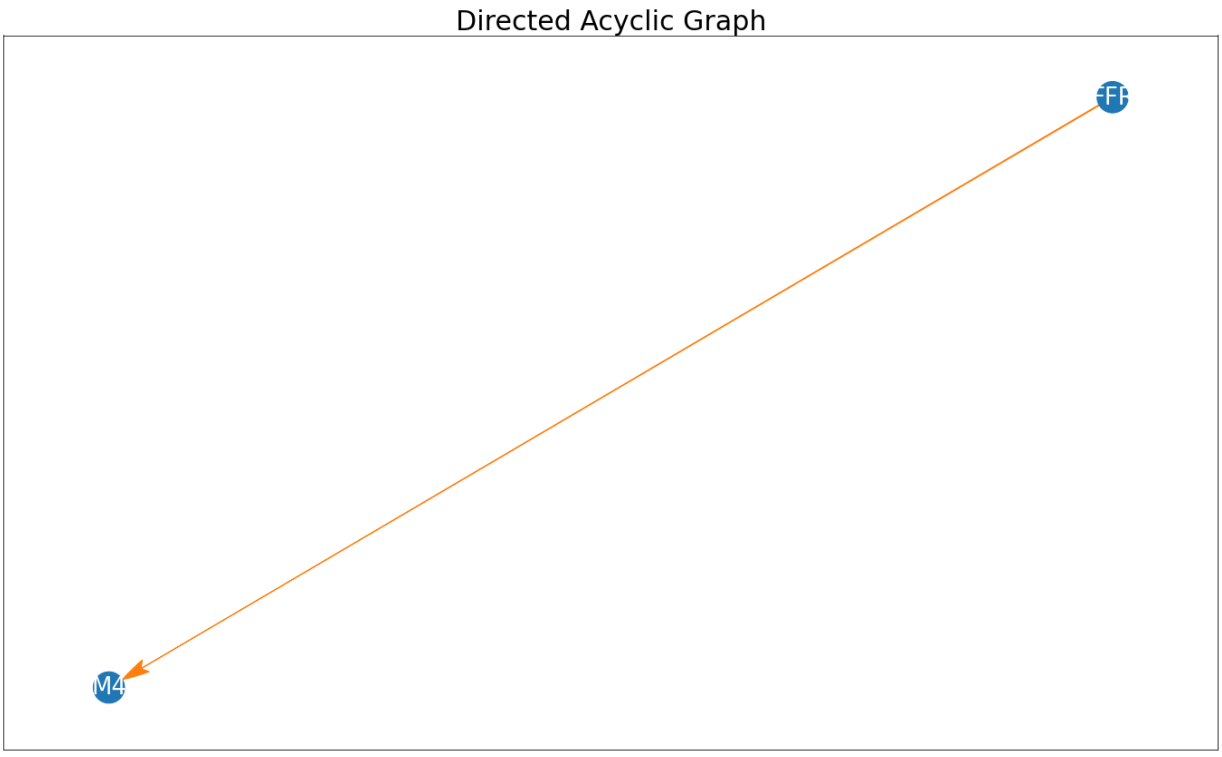
In [21]:
```
## Estimating a Directed Acyclic Graph
p_val = .01
from pgmpy.estimators import PC
c = PC(df)
max_cond_vars = len(df.keys())-2


model = c.estimate(return_type = "dag",variant= "parallel",#"orig", "stable"
                   significance_level = p_val,
                   max_cond_vars = max_cond_vars, ci_test = "pearsonr")
edges = model.edges()
```

Working for n conditional variables: 2/4 [00:00<00:00, 2.69it/s]

2: 50%

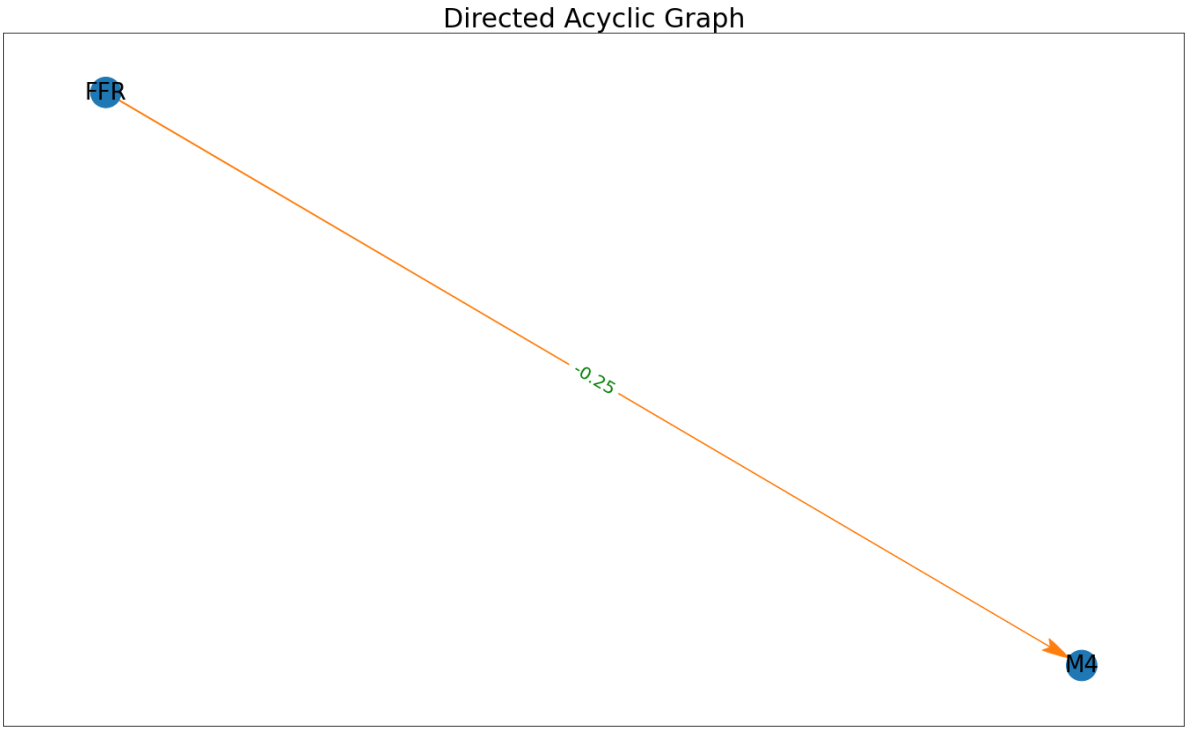
In [22]:
```
from matplotlib.patches import ArrowStyle

def graph_DAG(edges, df, title = ""):
    graph = nx.DiGraph()
    graph.add_edges_from(edges)
    color_map = ["C0" for g in graph]

    fig, ax = plt.subplots(figsize = (20,12))
    graph.nodes()
    plt.tight_layout()
    pos = nx.spring_layout(graph)#, k = 5/(len(sig_corr.keys())**.5))

    plt.title(title, fontsize = 30)
    nx.draw_networkx(graph, pos, node_color=color_map, node_size = 1200,
                     with_labels=True,  arrows=True,
                     font_color = "white",
                     font_size = 26, alpha = 1,
                     width = 1, edge_color = "C1",
                     arrowstyle=ArrowStyle("Fancy, head_length=3, head_width=1.5,

graph_DAG(edges, df, title = "Directed Acyclic Graph")
edges
```

Out[22]:
```
OutEdgeView([('FFR', 'M4')])
```



In [23]:
```
## D-separation

def graph_DAG(edges, df, title = ""):
    graph = nx.DiGraph()
    edge_labels = {}
    ############ Add ############
    for edge in edges:
        controls = [key for key in df.keys() if key not in edge]
        controls = list(set(controls))
        keep_controls = []
        for control in controls:
            control_edges = [ctrl_edge for ctrl_edge in edges if control == ctrl_
            if (control, edge[1]) in control_edges:
                print("keep control:", control)
                keep_controls.append(control)
        print(edge, keep_controls)
        pcorr = df[[edge[0], edge[1]]+keep_controls].pcorr()
#         corr_matrix_heatmap(pcorr, save_fig = False, pp = None, title = "Partic
        edge_labels[edge] = str(round(pcorr[edge[0]].loc[edge[1]],2))
    graph.add_edges_from(edges)
    color_map = ["C0" for g in graph]

    fig, ax = plt.subplots(figsize = (20,12))
    graph.nodes()
    plt.tight_layout()
    pos = nx.spring_layout(graph)#, k = 5/(len(sig_corr.keys())**.5))

    plt.title(title, fontsize = 30)
    nx.draw_networkx(graph, pos, node_color=color_map, node_size = 1200,
                     with_labels=True,  arrows=True,
                     # turn text black for larger variable names in homework
                     font_color = "k",
                     font_size = 26, alpha = 1,
                     width = 1, edge_color = "C1",
                     arrowstyle=ArrowStyle("Fancy, head_length=3, head_width=1.5,
    ############ Add ############
    nx.draw_networkx_edge_labels(graph,pos,
                                 edge_labels=edge_labels,
                                 font_color='green',
                                 font_size=20)

graph_DAG(edges, df, title = "Directed Acyclic Graph")
```

```
('FFR', 'M4') []
```



In [24]:
```
data = df
def firstLetterWord(str, num_chars = 3):

    result = ""

    # Traverse the string.
    v = True
    for i in range(len(str)):

        # If it is space, set v as true.
        if (str[i] == ' '):
            v = True

        # Else check if v is true or not.
        # If true, copy character in output
        # string and set v as false.
        elif (str[i] != ' ' and v == True):
            result += (str[i:i+num_chars])
            v = False

    return result
```

In [25]:
```
def graph_DAG(edges, data_reg, title = "",
              fig = False, ax = False,
              edge_labels = False,sig_vals = [0.05, 0.01, 0.001]):
    pcorr = data_reg.pcorr()
    graph = nx.DiGraph()
    def build_edge_labels(edges, df, sig_vals):
        edge_labels = {}
        for edge in edges:
            controls = [key for key in df.keys() if key not in edge]
            controls = list(set(controls))
            keep_controls = []
            for control in controls:
                control_edges = [ctrl_edge for ctrl_edge in edges if control == c
                if (control, edge[1]) in control_edges:
                    keep_controls.append(control)
#             print(edge, keep_controls)
            pcorr = df.partial_corr(x = edge[0], y = edge[1], covar=keep_controls
                                    method = "pearson")
            label = str(round(pcorr["r"][0],2))
            pvalue = pcorr["p-val"][0]
#             pcorr = df[[edge[0], edge[1]]+keep_controls].pcorr()
#             label = pcorr[edge[0]].loc[edge[1]]

            for sig_val in sig_vals:
                if pvalue < sig_val:
                    label = label + "*"

            edge_labels[edge] = label
        return edge_labels

    if edge_labels == False:
        edge_labels = build_edge_labels(edges,
                                        data_reg,
                                        sig_vals=sig_vals)
    graph.add_edges_from(edges)
    color_map = ["grey" for g in graph]

    if fig == False and ax == False: fig, ax = plt.subplots(figsize = (20,12))
    graph.nodes()
    plt.tight_layout()
    #pos = nx.spring_layout(graph)
    pos = graphviz_layout(graph)

    edge_labels2 = []
    for u, v, d in graph.edges(data=True):
        if pos[u][0] > pos[v][0]:
            if (v,u) in edge_labels.keys():
                edge_labels2.append(((u, v,), f'{edge_labels[u,v]}\n\n\n{edge_lab
        if (v,u) not in edge_labels.keys():
            edge_labels2.append(((u,v,), f'{edge_labels[(u,v)]}'))
    edge_labels = dict(edge_labels2)

    nx.draw_networkx(graph, pos, node_color=color_map, node_size = 2500,
                     with_labels=True,  arrows=True,
                     font_color = "black",
                     font_size = 26, alpha = 1,
```

```
                    width = 1, edge_color = "C1",
                    arrowstyle=ArrowStyle("Fancy, head_length=3, head_width=1.5,
                    connectionstyle='arc3, rad = 0.05',
                    ax = a)
    nx.draw_networkx_edge_labels(graph,pos,
                                 edge_labels=edge_labels,
                                 font_color='green',
                                 font_size=20,
                                ax = a)


DAG_models_vars = {0:["M4", "TA", "U","I"],
                   1:["M4", "CC", "TA","FFR"],
                   2:["M4", "FFR", "TA", "CC", "U"],
                   3:["TA", "U", "FFR", "M4"],
                   4:["TA", "U","I", "FFR", "CC"],
                   5:["TA", "U", "FFR", "CC","M4", "I"],}
# link_sigs = [0.05, 0.1, 0.2]
link_sigs = [0.05, .1, .2]
algorithms = ["orig", "stable", "parallel"]
for keys in DAG_models_vars.values():
    fig, ax = plt.subplots(len(algorithms), len(link_sigs), figsize = (30,30))
    max_cond_vars = len(keys) - 2
    data_reg = data[keys].dropna()
    data_reg.rename(columns = {col:firstLetterWord(col) for col in keys}, inplace
    keys = data_reg.keys()
    c = PC(data_reg[keys].dropna())
    max_cond_vars = len(keys) - 2
    i,j = 0,0
    for sig in link_sigs:
        for algorithm in algorithms:
            model = c.estimate(return_type = "pdag", variant = algorithm,
                               significance_level = sig,
                               max_cond_vars = max_cond_vars, ci_test = "pearsonr
            edges = model.edges()
            pcorr = data_reg.pcorr()
            weights = {}
            a = ax[i][j]
            graph_DAG(edges, data_reg, fig = fig, ax = a)

            if j == 0:
                a.set_ylabel(algorithm, fontsize = 20)
            if i == len(algorithms) - 1:
                a.set_xlabel("$p \leq$ "+ str(sig), fontsize = 20)
            i += 1
        j += 1
        i = 0
    plt.suptitle(str(list(keys)).replace("[","").replace("]",""), fontsize = 40,
    plt.show()
    plt.close()
edges
```

Working for n conditional variables: 1/2 [00:00<00:00,

1: 50% 32.00it/s]

Working for n conditional variables: 1: 50% 1/2 [00:00<00:00, 32.00it/s]

Working for n conditional variables: 1: 50% 1/2 [00:00<00:00, 21.34it/s]

Working for n conditional variables: 1: 50% 1/2 [00:00<00:00, 32.02it/s]

Working for n conditional variables: 1: 50% 1/2 [00:00<00:00, 21.34it/s]

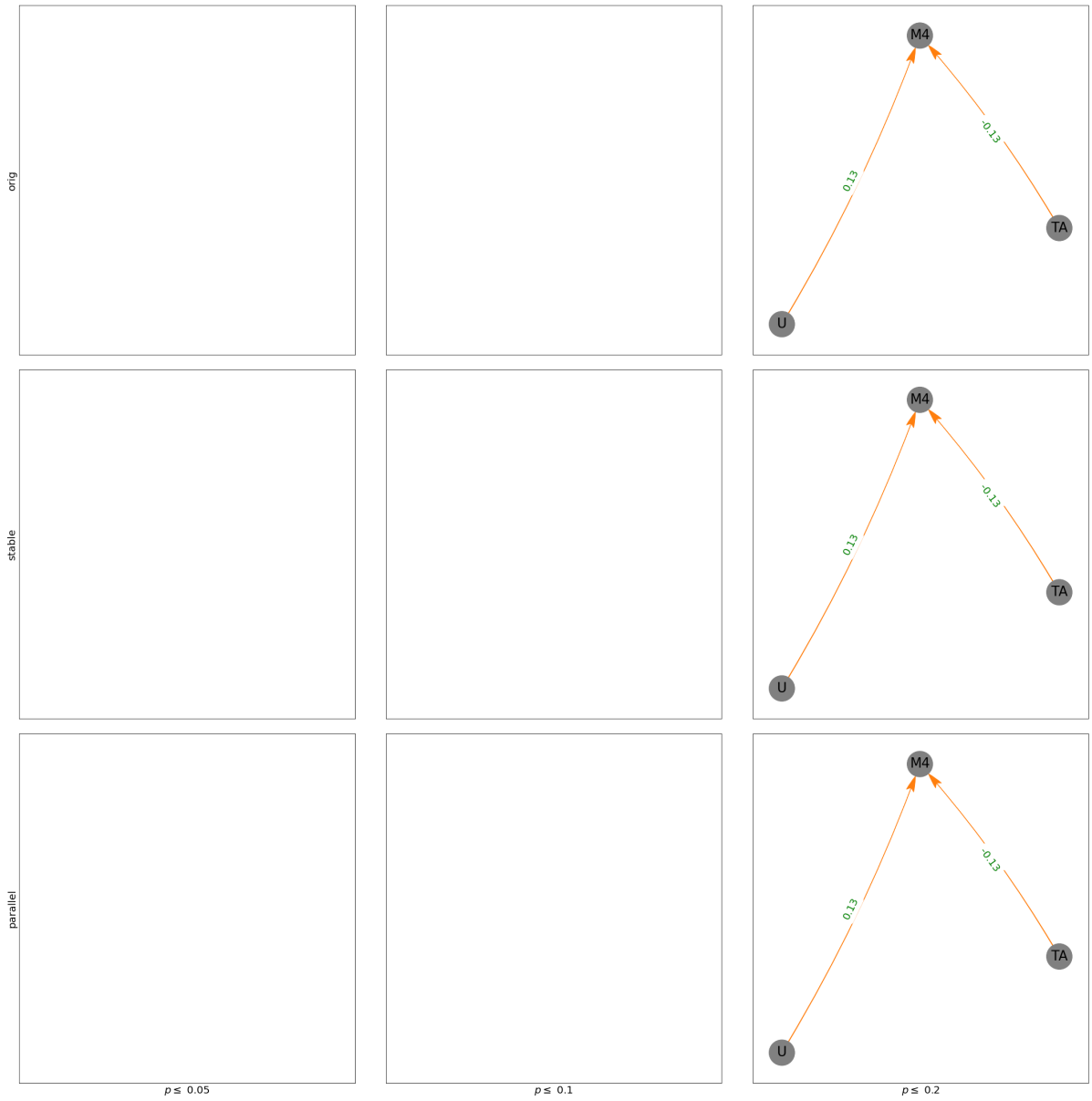
Working for n conditional variables: 1: 50% 1/2 [00:00<00:00, 32.03it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 8.59it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 42.67it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 18.38it/s]

'M4', 'TA', 'U', 'I'



Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 64.06it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 64.00it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 42.67it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 42.68it/s]

Working for n conditional variables: 2/2 [00:00<00:00,

2: 100% 42.67it/s]

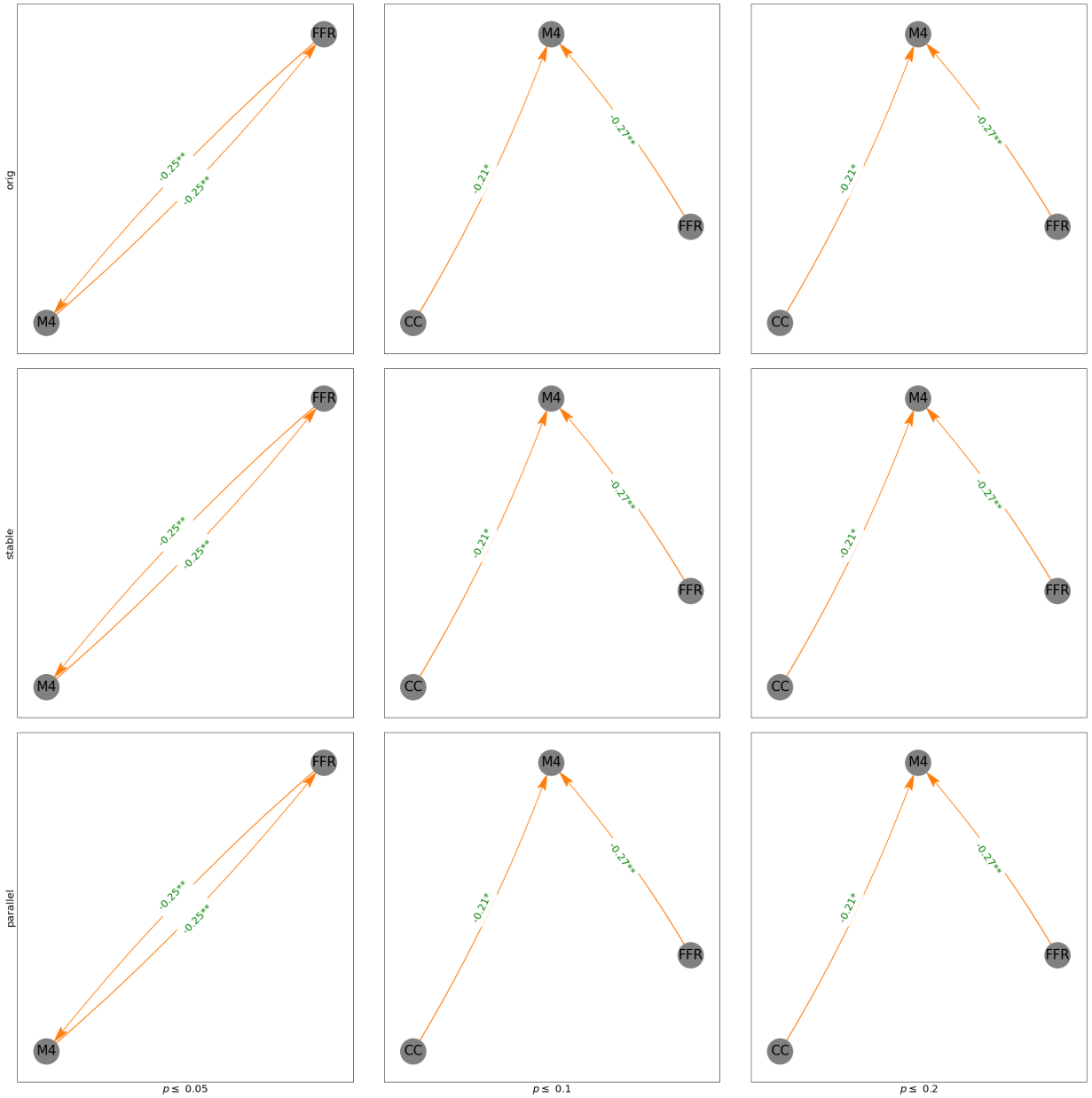
Working for n conditional variables: 2/2 [00:00<00:00,
2: 100% 32.00it/s]

Working for n conditional variables: 2/2 [00:00<00:00,
2: 100% 25.05it/s]

Working for n conditional variables: 2/2 [00:00<00:00,
2: 100% 20.36it/s]

Working for n conditional variables: 2/2 [00:00<00:00,
2: 100% 17.55it/s]

'M4', 'CC', 'TA', 'FFR'



Working for n conditional variables: 2: 67% 2/3 [00:00<00:00, 64.01it/s]

Working for n conditional variables: 2: 67% 2/3 [00:00<00:00, 42.67it/s]

Working for n conditional variables: 2: 67% 2/3 [00:00<00:00, 42.67it/s]

Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 48.00it/s]

Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 48.00it/s]

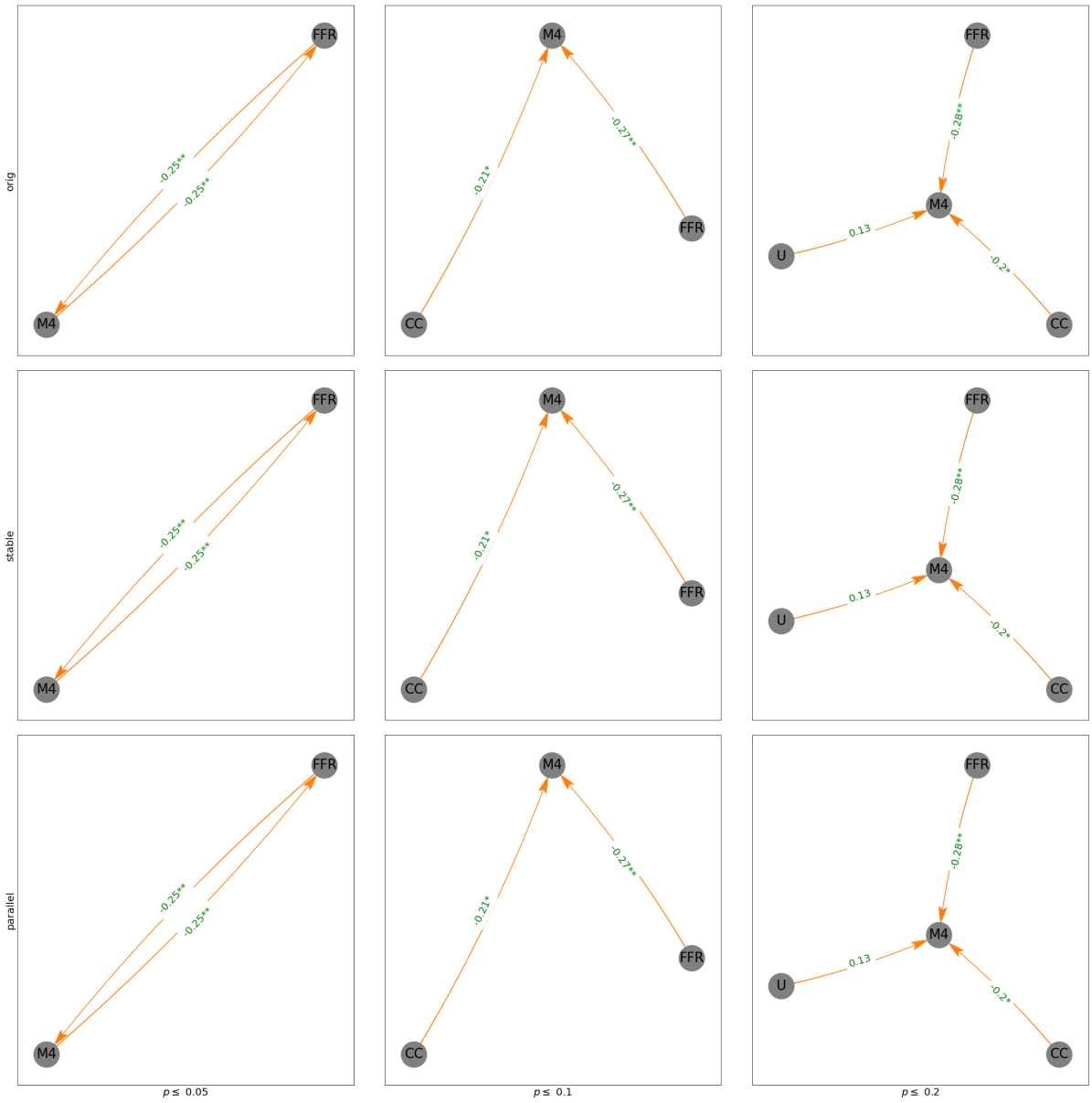
Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 4.16it/s]

Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 15.75it/s]

Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 15.51it/s]

Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 13.42it/s]

'M4', 'FFR', 'TA', 'CC', 'U'


orig
stable
parallel
FFR
M4
CC
U
-0.25**
-0.25**
-0.21*
-0.27**
-0.28**
0.13
-0.2*
$p \leq 0.05$
$p \leq 0.1$
$p \leq 0.2$

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 64.04it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 64.00it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 42.67it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 64.00it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 42.65it/s]

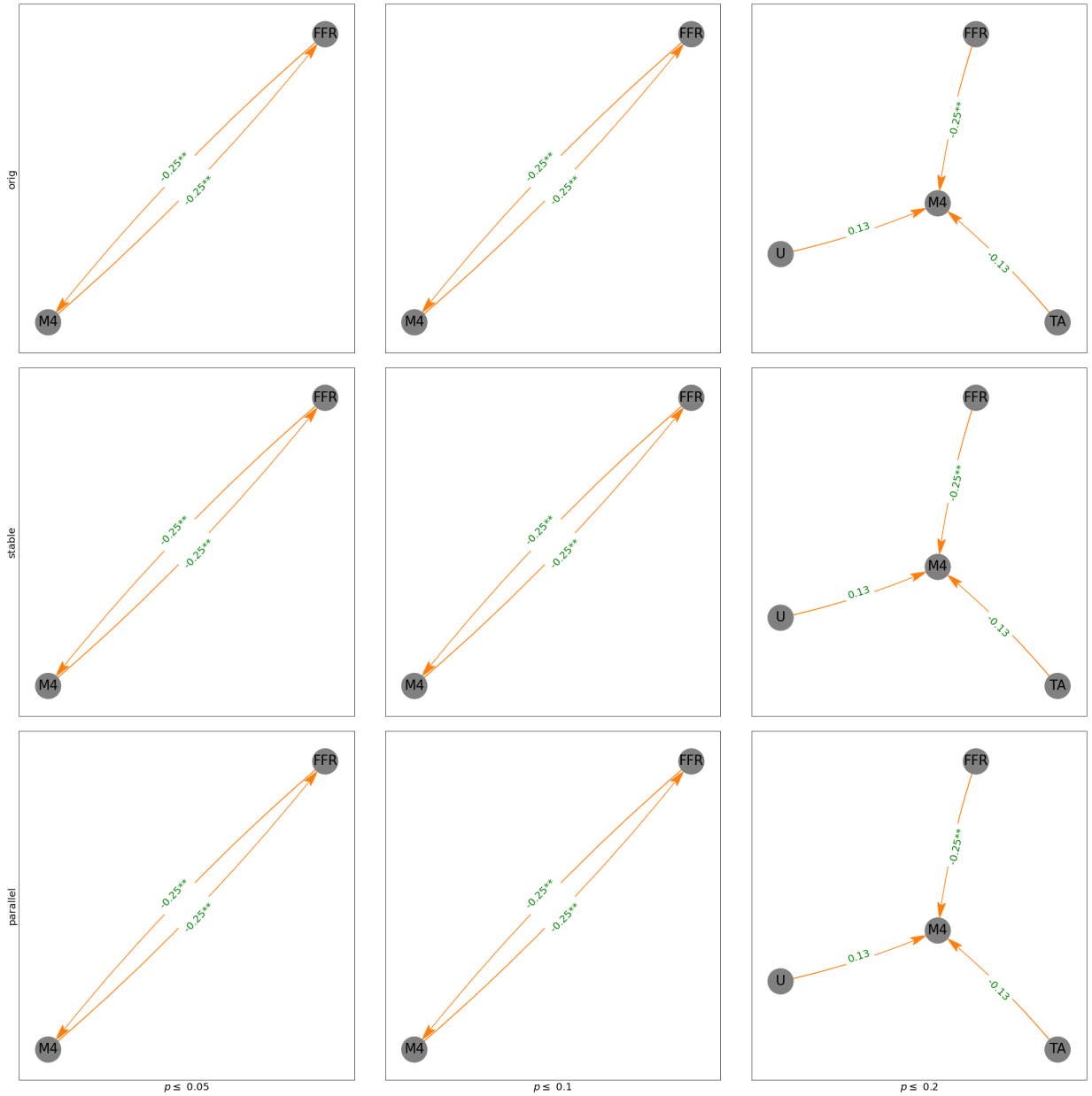
Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 32.00it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 9.70it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 16.29it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 19.13it/s]

'TA', 'U', 'FFR', 'M4'



Working for n conditional variables: 1/3 [00:00<00:00, 32.00it/s]

Working for n conditional variables: 1/3 [00:00<00:00, 32.00it/s]

Working for n conditional variables: 1/3 [00:00<00:00, 21.34it/s]

Working for n conditional variables: 1/3 [00:00<00:00, 21.33it/s]

Working for n conditional variables: 1/3 [00:00<00:00, 32.03it/s]
1: 33%

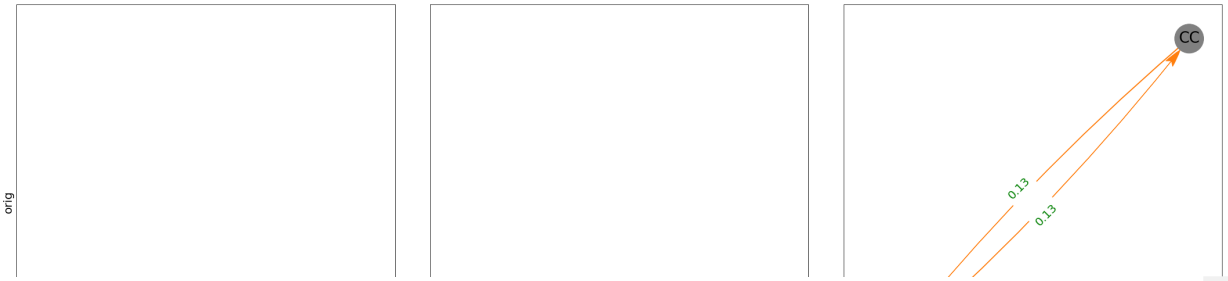
Working for n conditional variables: 1/3 [00:00<00:00, 32.00it/s]
1: 33%

Working for n conditional variables: 2/3 [00:00<00:00, 42.67it/s]
2: 67%

Working for n conditional variables: 2/3 [00:00<00:00, 64.01it/s]
2: 67%

Working for n conditional variables: 2/3 [00:00<00:00, 42.69it/s]
2: 67%

'TA', 'U', 'I', 'FFR', 'CC'



Working for n conditional variables: 2/4 [00:00<00:00, 42.67it/s]

2: 50%

Working for n conditional variables: 2/4 [00:00<00:00, 42.69it/s]

2: 50%

Working for n conditional variables: 2/4 [00:00<00:00, 32.00it/s]

2: 50%

Working for n conditional variables: 3/4 [00:00<00:00, 64.04it/s]

3: 75%

Working for n conditional variables: 3/4 [00:00<00:00, 64.03it/s]

3: 75%

Working for n conditional variables: 3/4 [00:00<00:00, 29.75it/s]

3: 75%

Working for n conditional variables: 4/4 [00:00<00:00, 18.20it/s]

4: 100%

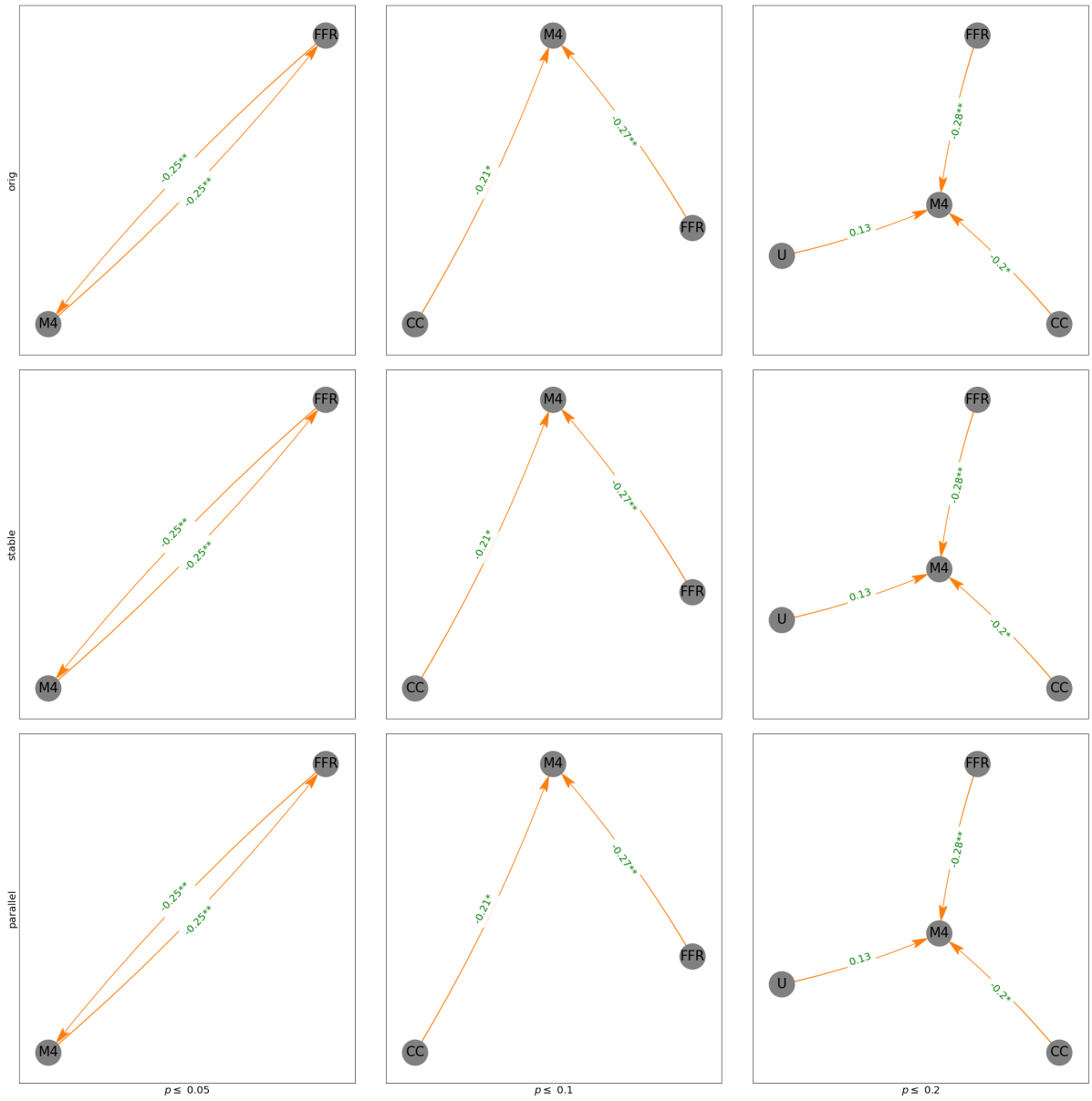
Working for n conditional variables: 4/4 [00:00<00:00, 16.99it/s]

4: 100%

Working for n conditional variables: 4/4 [00:00<00:00, 12.48it/s]

4: 100%

'TA', 'U', 'FFR', 'CC', 'M4', 'I'



Out[25]: OutEdgeView([('U', 'M4'), ('CC', 'M4'), ('FFR', 'M4')])

In [ ]:

In [ ]:

In [ ]: