In [23]:
```
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 [3]:
```
#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]
```

TWICE DIFF WITH MONTHLY RATE: M4 TA FFR CC NGDP

In [25]:
```
#data cleaning, importing

d_parser = lambda x: pd.datetime.strptime(x, '%m/%d/%Y')
df = pd.read_csv('M4-11.csv', parse_dates=['Date'], date_parser=d_parser)
df
```

Out[25]:

	Date	Log M4 including Treasuries	Log Total Assets	Effective Federal Funds Rate (%)	Log Currency in Circulation ($ Bil)	Log Nominal GDP (Bln)	Unemployment Rate	Inflation Rate
0	2010-01-31	7.07	14.63	0.11	6.83	16.51	9.8	2.62
1	2010-02-28	7.06	14.63	0.13	6.83	16.70	9.8	2.15
2	2010-03-31	7.05	14.65	0.17	6.84	16.70	9.9	2.29
3	2010-04-30	7.06	14.66	0.20	6.84	16.52	9.9	2.21
4	2010-05-31	7.06	14.66	0.20	6.84	16.70	9.6	2.00
...	...	...	...	...	...	...	...	...
115	2019-08-31	7.40	15.14	2.13	7.47	16.70	3.7	1.74
116	2019-09-30	7.40	15.15	2.04	7.47	16.70	3.5	1.72
117	2019-10-31	7.41	15.19	1.83	7.48	16.89	3.6	1.77
118	2019-11-30	7.42	15.21	1.55	7.49	16.70	3.6	2.04
119	2019-12-31	7.42	15.23	1.55	7.49	16.70	3.6	2.26

120 rows × 8 columns

In [26]:
```
df['Date_at_year_month'] = df['Date'].dt.strftime('%Y-%m')
column_names = {'Date_at_year_month':'DATE',
                'Log M4 including Treasuries':'M4',
                'Log Total Assets': 'TA',
                'Log Currency in Circulation ($ Bil)':'CC',
                'Log Nominal GDP (Bln)':'NGDP',
               'Effective Federal Funds Rate (%)': 'FFR',
               'Unemployment Rate': 'U',
               'Inflation Rate': 'I'}


# rename columns
df = df.rename(columns = column_names)
df = df.set_index('DATE')
df = df.drop(['Date'], axis = 1)
df = df.drop(['U','I'], axis = 1)
df
```

Out[26]:

DATE	M4	TA	FFR	CC	NGDP
2010-01	7.07	14.63	0.11	6.83	16.51
2010-02	7.06	14.63	0.13	6.83	16.70
2010-03	7.05	14.65	0.17	6.84	16.70
2010-04	7.06	14.66	0.20	6.84	16.52
2010-05	7.06	14.66	0.20	6.84	16.70
...	...	...	...	...	...
2019-08	7.40	15.14	2.13	7.47	16.70
2019-09	7.40	15.15	2.04	7.47	16.70
2019-10	7.41	15.19	1.83	7.48	16.89
2019-11	7.42	15.21	1.55	7.49	16.70
2019-12	7.42	15.23	1.55	7.49	16.70

120 rows × 5 columns

In [27]:
```
df_new = df.diff().dropna()
```

In [29]:
```
df = df_new.diff().dropna()
df
```

Out[29]:

DATE	M4	TA	FFR	CC	NGDP
2010-03	8.881784e-16	2.000000e-02	0.02	1.000000e-02	-0.19
2010-04	2.000000e-02	-1.000000e-02	-0.01	-1.000000e-02	-0.18
2010-05	-1.000000e-02	-1.000000e-02	-0.03	0.000000e+00	0.36
2010-06	-1.000000e-02	1.000000e-02	-0.02	1.000000e-02	-0.18
2010-07	2.000000e-02	-2.000000e-02	0.02	-1.000000e-02	-0.17
...	...	...	...	...	...
2019-08	1.000000e-02	0.000000e+00	-0.29	-1.000000e-02	-0.36
2019-09	-1.000000e-02	2.000000e-02	0.18	0.000000e+00	0.18
2019-10	1.000000e-02	3.000000e-02	-0.12	1.000000e-02	0.19
2019-11	0.000000e+00	-2.000000e-02	-0.07	-8.881784e-16	-0.38
2019-12	-1.000000e-02	-1.776357e-15	0.28	-1.000000e-02	0.19

118 rows × 5 columns

In [30]:
```
#ADF test

X = df["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 = df["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 = df["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 = df["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 = df["NGDP"].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")
```

```
ADF Statistic: -8.381034
p-value: 0.000000
Critical Values:
        1%: -3.491
        5%: -2.888
        10%: -2.581
Reject Ho - Time Series is Stationary
ADF Statistic: -3.943879
p-value: 0.001735
Critical Values:
        1%: -3.492
        5%: -2.889
        10%: -2.581
Reject Ho - Time Series is Stationary
ADF Statistic: -6.016185
p-value: 0.000000
Critical Values:
        1%: -3.490
        5%: -2.887
        10%: -2.581
Reject Ho - Time Series is Stationary
ADF Statistic: -6.497273
p-value: 0.000000
Critical Values:
        1%: -3.494
        5%: -2.889
        10%: -2.582
Reject Ho - Time Series is Stationary
ADF Statistic: -9.041977
p-value: 0.000000
Critical Values:
        1%: -3.492
        5%: -2.889
        10%: -2.581
Reject Ho - Time Series is Stationary
```

In [6]:
```
## 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
```

Out[6]:

DATE	M4	TA	FFR	CC	NGDP
2010-03	0.002266	0.021385	0.073781	0.007450	-0.251514
2010-04	0.015494	-0.004844	0.039686	-0.005667	-0.114575
2010-05	-0.008633	-0.016333	-0.120451	-0.000121	0.399718
2010-06	-0.009484	0.009980	0.016095	0.005844	-0.193220
2010-07	0.015675	-0.014680	0.063929	-0.003976	-0.110353
...	...	...	...	...	...
2019-08	-0.003003	0.003953	-0.211302	-0.014544	-0.049576
2019-09	-0.002151	0.018271	0.137866	-0.000313	-0.071530
2019-10	0.012304	0.025590	-0.129247	0.007236	0.265340
2019-11	-0.006640	-0.015658	0.000942	-0.000440	-0.256128
2019-12	-0.002688	0.000384	0.221011	-0.006244	-0.121739

118 rows × 5 columns

In [7]:
```
residuals.corr()[residuals.corr().abs() < 1].mul(-1).fillna(1).round(2)
```

Out[7]:

	M4	TA	FFR	CC	NGDP
M4	1.00	-0.08	-0.18	-0.20	-0.12
TA	-0.08	1.00	-0.06	0.12	0.19
FFR	-0.18	-0.06	1.00	-0.12	0.46
CC	-0.20	0.12	-0.12	1.00	-0.05
NGDP	-0.12	0.19	0.46	-0.05	1.00

In [8]:
```
# !pip install pingouin
import pingouin
df.pcorr().round(2)
```

Out[8]:

	M4	TA	FFR	CC	NGDP
M4	1.00	-0.08	-0.18	-0.20	-0.12
TA	-0.08	1.00	-0.06	0.12	0.19
FFR	-0.18	-0.06	1.00	-0.12	0.46
CC	-0.20	0.12	-0.12	1.00	-0.05
NGDP	-0.12	0.19	0.46	-0.05	1.00

In [9]:
```
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[9]:

	M4	TA	FFR	CC	NGDP
M4	NaN	0.39	0.05	0.03	0.19
TA	0.39	NaN	0.51	0.20	0.04
FFR	0.05	0.51	NaN	0.21	0.00
CC	0.03	0.20	0.21	NaN	0.57
NGDP	0.19	0.04	0.00	0.57	NaN

In [10]:
```
residuals
```

Out[10]:

DATE	M4	TA	FFR	CC	NGDP
2010-03	0.002266	0.021385	0.073781	0.007450	-0.251514
2010-04	0.015494	-0.004844	0.039686	-0.005667	-0.114575
2010-05	-0.008633	-0.016333	-0.120451	-0.000121	0.399718
2010-06	-0.009484	0.009980	0.016095	0.005844	-0.193220
2010-07	0.015675	-0.014680	0.063929	-0.003976	-0.110353
...	...	...	...	...	...
2019-08	-0.003003	0.003953	-0.211302	-0.014544	-0.049576
2019-09	-0.002151	0.018271	0.137866	-0.000313	-0.071530
2019-10	0.012304	0.025590	-0.129247	0.007236	0.265340
2019-11	-0.006640	-0.015658	0.000942	-0.000440	-0.256128
2019-12	-0.002688	0.000384	0.221011	-0.006244	-0.121739

118 rows × 5 columns

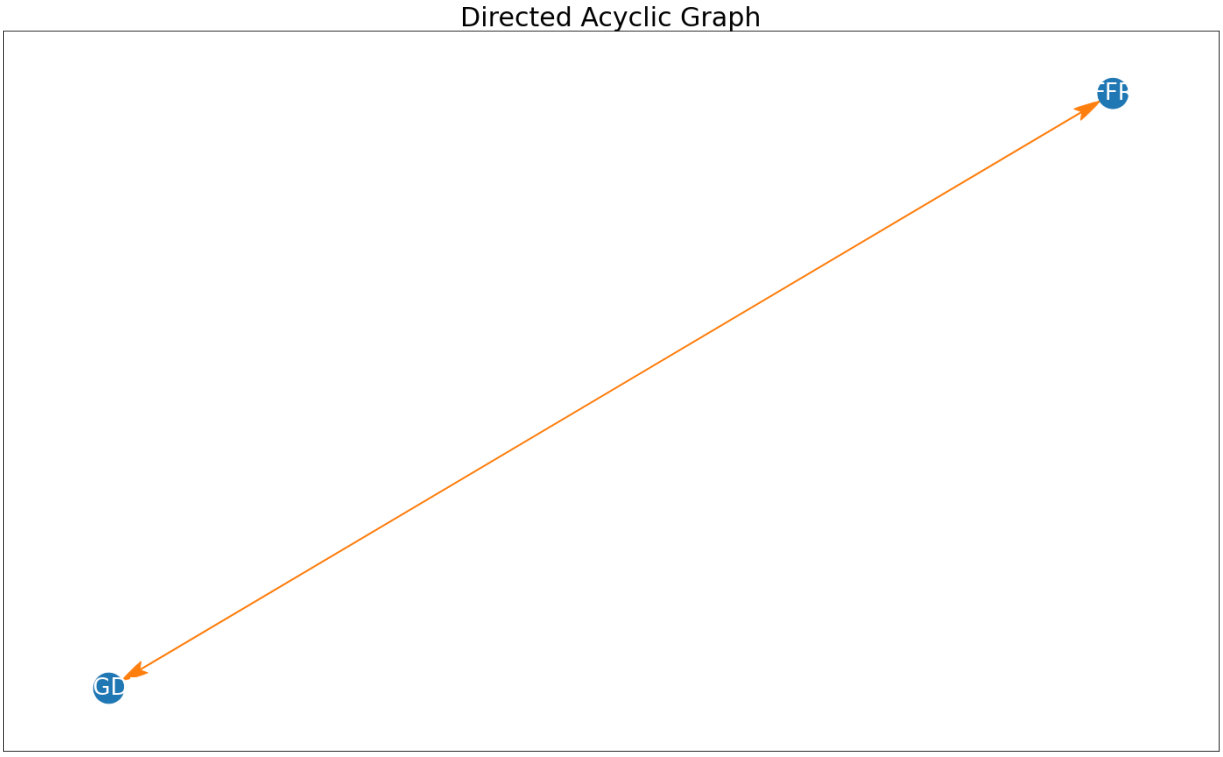
In [11]:
```
##DAG

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 [12]:
```
## 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 = "pdag",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: 67% 2/3 [00:00<00:00, 3.60it/s]

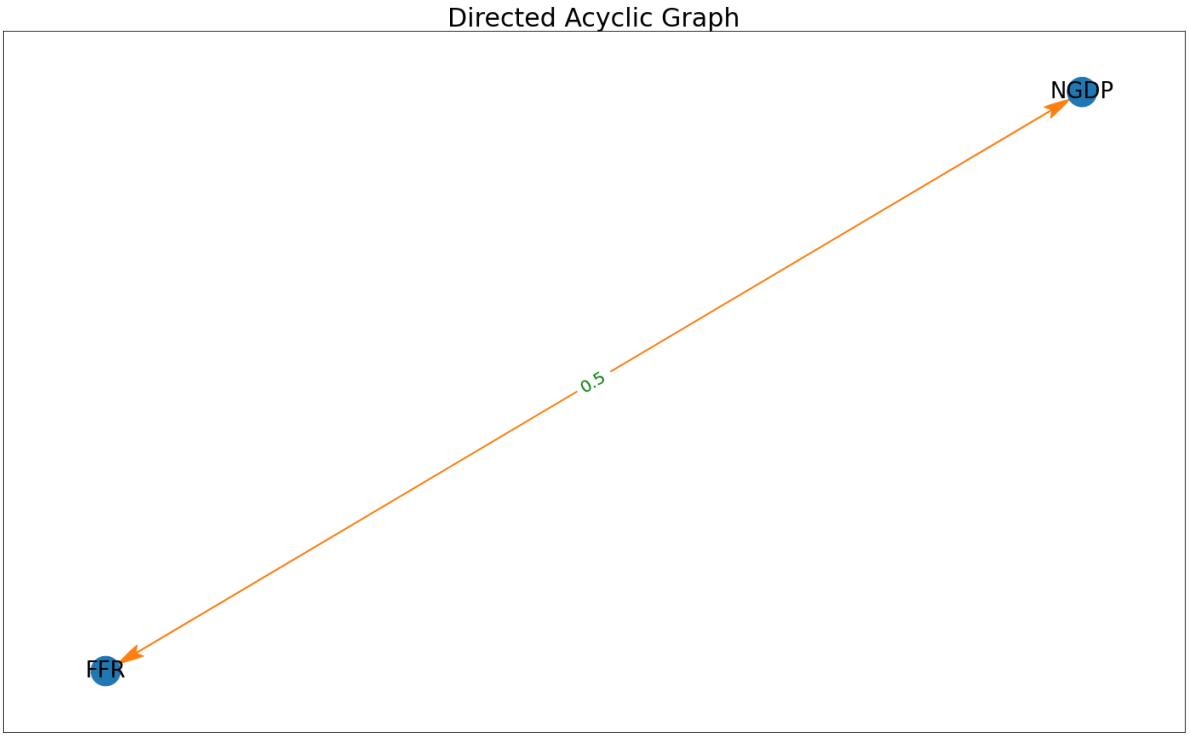
In [13]:
```
from matplotlib.patches import ArrowStyle

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[13]:
```
OutEdgeView([('FFR', 'NGDP'), ('NGDP', 'FFR')])
```



In [14]:
```
## 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', 'NGDP') []
('NGDP', 'FFR') []
```



In [15]:
```
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 [16]:
```
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,
```

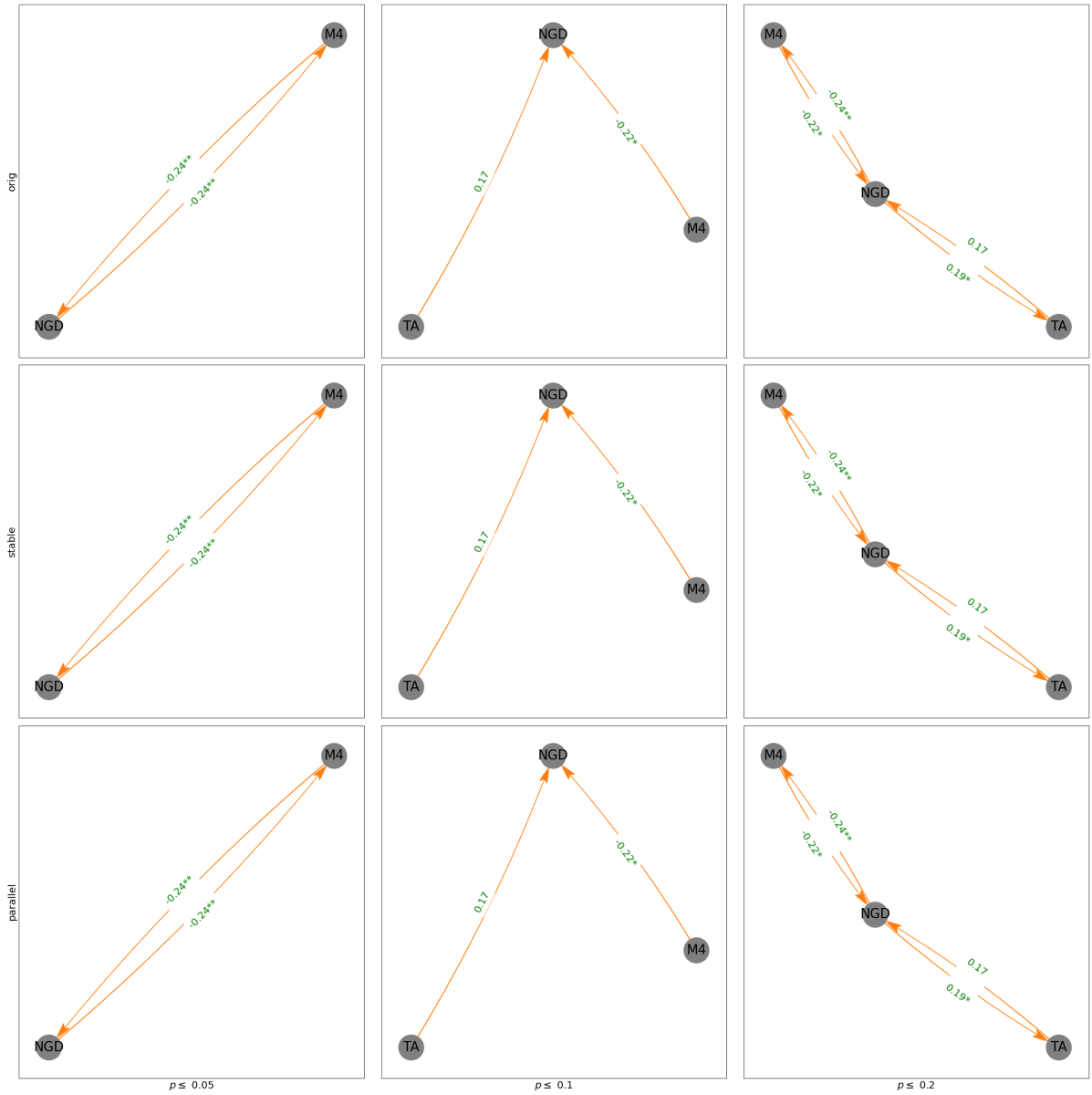
```
                    width = 1, edge_color = "C1",
                    arrowstyle=ArrowStyle("Fancy, head_length=3, head_width=1.5,
                    connectionstyle='arc3, rad = 0.05',
                    ax = a)
    nx.draw_networkx_edge_labels(graph,pos,
                                 edge_labels=edge_labels,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


DAG_models_vars = {0:["M4","TA", "NGDP"],
                   1:["TA", "CC", "FFR"],
                   2:["M4","CC", "FFR"],
                   3:["TA", "CC", "FFR", "M4"],
                   4:["TA", "NGDP", "FFR", "CC","M4"],
                   5:["CC", "NGDP","M4"]}
# 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: 100%

1/1 [00:00<00:00, 21.35it/s]

Working for n conditional variables: 1: 100%		1/1 [00:00<00:00, 21.33it/s]
Working for n conditional variables: 1: 100%		1/1 [00:00<00:00, 13.96it/s]
Working for n conditional variables: 1: 100%		1/1 [00:00<00:00, 9.14it/s]
Working for n conditional variables: 1: 100%		1/1 [00:00<00:00, 14.75it/s]
Working for n conditional variables: 1: 100%		1/1 [00:00<00:00, 14.49it/s]
Working for n conditional variables: 1: 100%		1/1 [00:00<00:00, 16.68it/s]
Working for n conditional variables: 1: 100%		1/1 [00:00<00:00, 16.78it/s]
Working for n conditional variables: 1: 100%		1/1 [00:00<00:00, 12.14it/s]

'M4', 'TA', 'NGD'



Working for n conditional variables: 1: 100% 1/1 [00:00<00:00, 32.00it/s]

Working for n conditional variables: 1: 100% 1/1 [00:00<00:00, 32.01it/s]

Working for n conditional variables: 1: 100% 1/1 [00:00<00:00, 16.84it/s]

Working for n conditional variables: 1: 100% 1/1 [00:00<00:00, 32.00it/s]

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

variables: 1: 100% 32.02it/s]

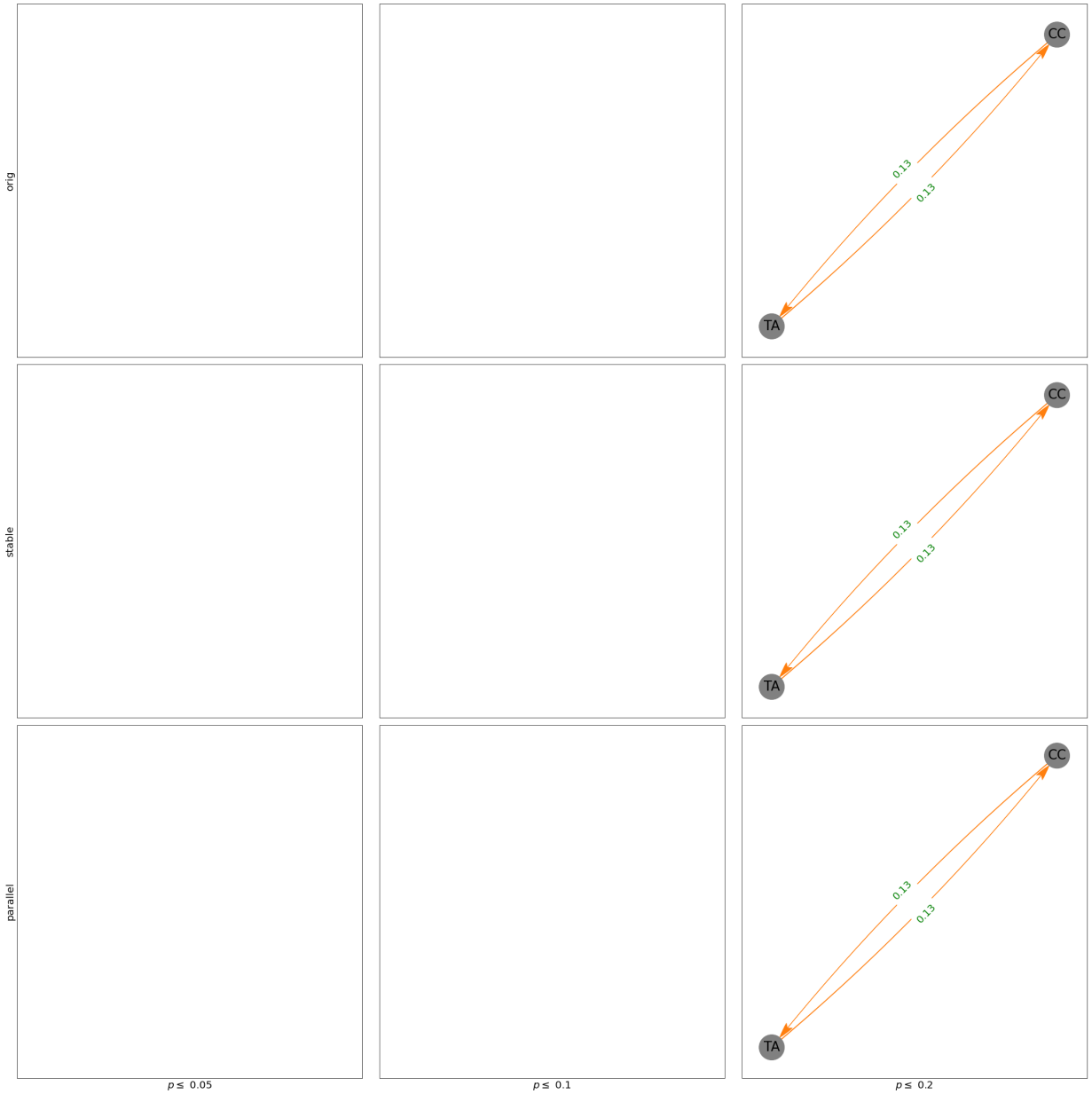
Working for n conditional variables: 1: 100% 1/1 [00:00<00:00, 21.34it/s]

Working for n conditional variables: 1: 100% 1/1 [00:00<00:00, 12.80it/s]

Working for n conditional variables: 1: 100% 1/1 [00:00<00:00, 31.98it/s]

Working for n conditional variables: 1: 100% 1/1 [00:00<00:00, 16.93it/s]

'TA', 'CC', 'FFR'



Working for n conditional variables: 1: 100% 1/1 [00:00<00:00, 32.00it/s]

Working for n conditional variables: 1: 100% 1/1 [00:00<00:00, 12.81it/s]

Working for n conditional variables: 1: 100% 1/1 [00:00<00:00, 14.68it/s]

Working for n conditional variables: 1: 100% 1/1 [00:00<00:00, 21.34it/s]

Working for n conditional variables: 1: 100% 1/1 [00:00<00:00, 17.93it/s]

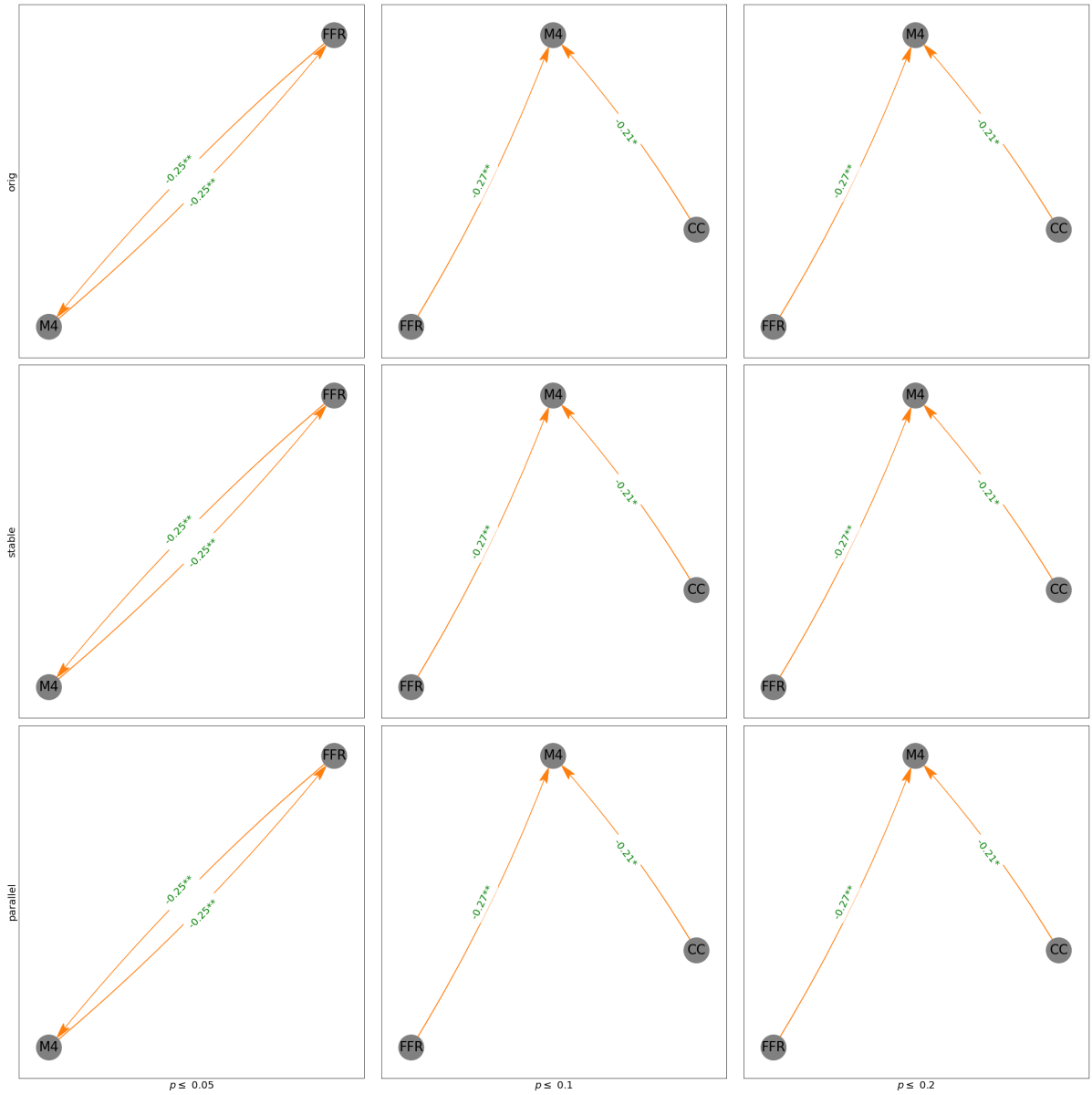
Working for n conditional variables: 1: 100% 1/1 [00:00<00:00, 15.76it/s]

Working for n conditional variables: 1: 100% 1/1 [00:00<00:00, 21.34it/s]

Working for n conditional variables: 1: 100% 1/1 [00:00<00:00, 21.34it/s]

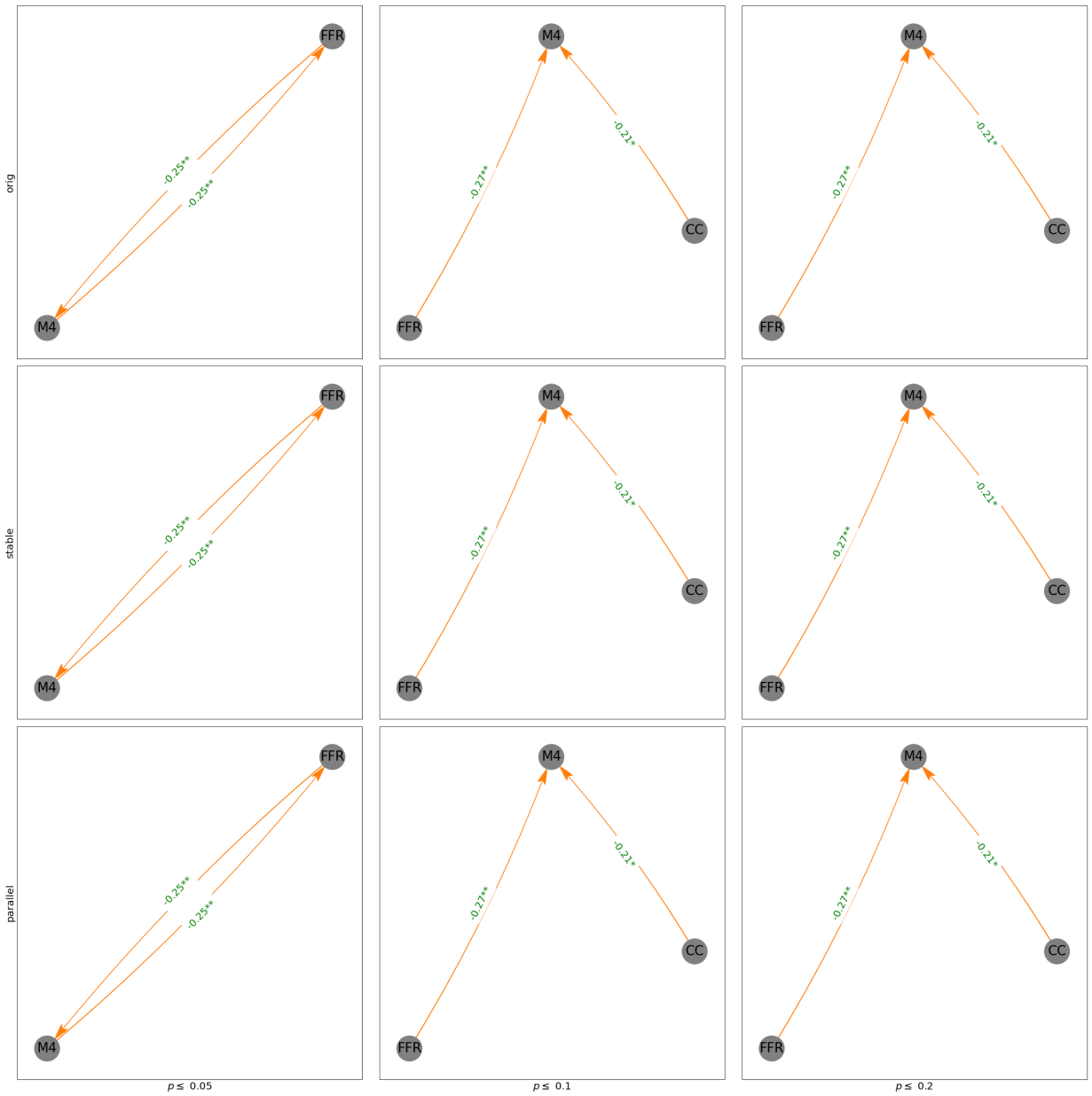
Working for n conditional variables: 1: 100% 1/1 [00:00<00:00, 13.97it/s]

'M4', 'CC', 'FFR'



Working for n conditional variables: 2: 100%		2/2 [00:00<00:00, 27.69it/s]
Working for n conditional variables: 2: 100%		2/2 [00:00<00:00, 64.05it/s]
Working for n conditional variables: 2: 100%		2/2 [00:00<00:00, 27.78it/s]
Working for n conditional variables: 2: 100%		2/2 [00:00<00:00, 29.82it/s]
Working for n conditional variables: 2: 100%		2/2 [00:00<00:00, 24.21it/s]
Working for n conditional variables: 2: 100%		2/2 [00:00<00:00, 23.94it/s]
Working for n conditional variables: 2: 100%		2/2 [00:00<00:00, 15.63it/s]
Working for n conditional variables: 2: 100%		2/2 [00:00<00:00, 27.78it/s]
Working for n conditional variables: 2: 100%		2/2 [00:00<00:00, 11.84it/s]

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



Working for n conditional variables: 2: 67% | 2/3 [00:00<00:00, 20.08it/s]

Working for n conditional variables: 2: 67% | 2/3 [00:00<00:00, 21.55it/s]

Working for n conditional variables: 2: 67% | 2/3 [00:00<00:00, 18.40it/s]

Working for n conditional variables: 3: 100% | 3/3 [00:00<00:00, 19.30it/s]

Working for n conditional | 3/3 [00:00<00:00,

variables: 3: 100% 15.39it/s]

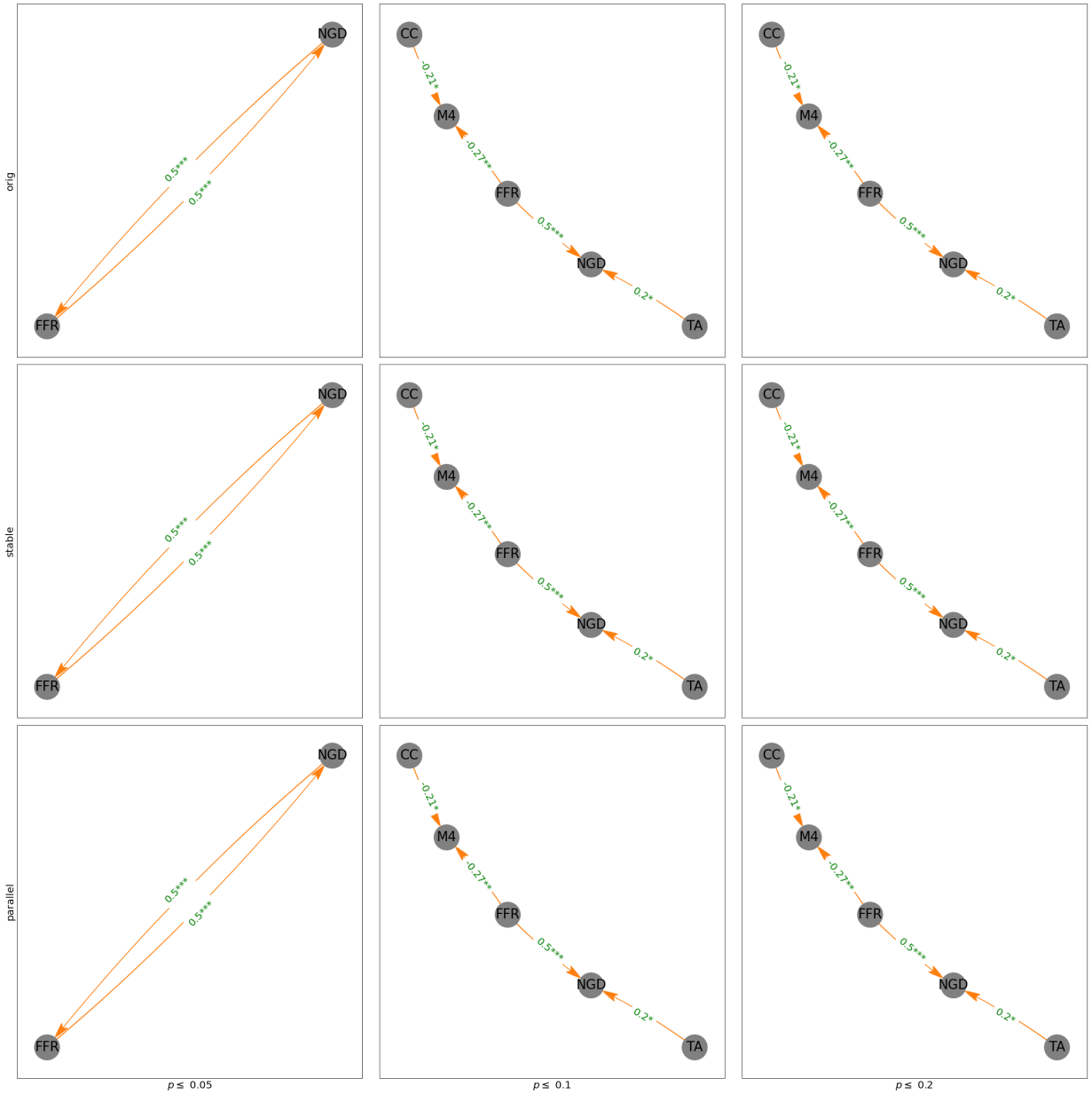
Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 12.58it/s]

Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 10.59it/s]

Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 12.48it/s]

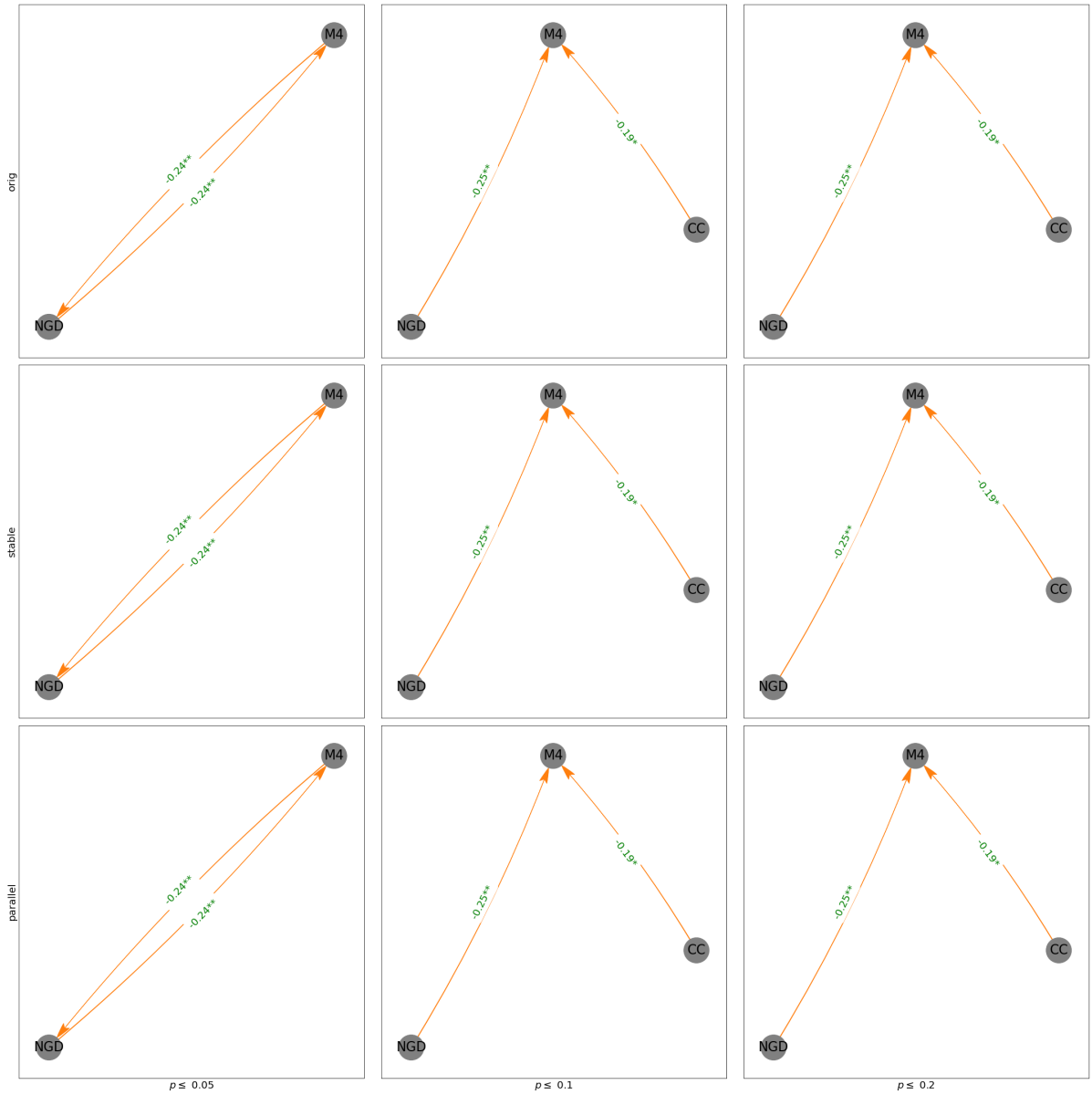
Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 9.28it/s]

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



Working for n conditional variables: 1: 100%		1/1 [00:00<00:00, 21.33it/s]
Working for n conditional variables: 1: 100%		1/1 [00:00<00:00, 63.92it/s]
Working for n conditional variables: 1: 100%		1/1 [00:00<00:00, 12.80it/s]
Working for n conditional variables: 1: 100%		1/1 [00:00<00:00, 12.41it/s]
Working for n conditional variables: 1: 100%		1/1 [00:00<00:00, 21.34it/s]
Working for n conditional variables: 1: 100%		1/1 [00:00<00:00, 14.82it/s]
Working for n conditional variables: 1: 100%		1/1 [00:00<00:00, 21.35it/s]
Working for n conditional variables: 1: 100%		1/1 [00:00<00:00, 18.05it/s]
Working for n conditional variables: 1: 100%		1/1 [00:00<00:00, 13.89it/s]

'CC', 'NGD', 'M4'



Out[16]: OutEdgeView([('NGD', 'M4'), ('CC', 'M4')])

TWICE DIFF WITH YEARLY RATE: M4 TA FFR CC NGDP

In [31]:
```
#data cleaning, importing

d_parser = lambda x: pd.datetime.strptime(x, '%m/%d/%Y')
df = pd.read_csv('M4-11.csv', parse_dates=['Date'], date_parser=d_parser)
df
```

Out[31]:

	Date	Log M4 including Treasuries	Log Total Assets	Effective Federal Funds Rate (%)	Log Currency in Circulation ($ Bil)	Log Nominal GDP (Bln)	Unemployment Rate	Inflation Rate
0	2010-01-31	7.07	14.63	0.11	6.83	16.51	9.8	2.62
1	2010-02-28	7.06	14.63	0.13	6.83	16.70	9.8	2.15
2	2010-03-31	7.05	14.65	0.17	6.84	16.70	9.9	2.29
3	2010-04-30	7.06	14.66	0.20	6.84	16.52	9.9	2.21
4	2010-05-31	7.06	14.66	0.20	6.84	16.70	9.6	2.00
...	...	...	...	...	...	...	...	...
115	2019-08-31	7.40	15.14	2.13	7.47	16.70	3.7	1.74
116	2019-09-30	7.40	15.15	2.04	7.47	16.70	3.5	1.72
117	2019-10-31	7.41	15.19	1.83	7.48	16.89	3.6	1.77
118	2019-11-30	7.42	15.21	1.55	7.49	16.70	3.6	2.04
119	2019-12-31	7.42	15.23	1.55	7.49	16.70	3.6	2.26

120 rows × 8 columns

In [32]:
```
df['Date_at_year_month'] = df['Date'].dt.strftime('%Y-%m')
column_names = {'Date_at_year_month':'DATE',
                'Log M4 including Treasuries':'M4',
                'Log Total Assets': 'TA',
                'Log Currency in Circulation ($ Bil)':'CC',
                'Log Nominal GDP (Bln)':'NGDP',
               'Effective Federal Funds Rate (%)': 'FFR',
               'Unemployment Rate': 'U',
               'Inflation Rate': 'I'}


# rename columns
df = df.rename(columns = column_names)
df = df.set_index('DATE')
df = df.drop(['Date'], axis = 1)
df = df.drop(['U','I'], axis = 1)
df
```

Out[32]:

DATE	M4	TA	FFR	CC	NGDP
2010-01	7.07	14.63	0.11	6.83	16.51
2010-02	7.06	14.63	0.13	6.83	16.70
2010-03	7.05	14.65	0.17	6.84	16.70
2010-04	7.06	14.66	0.20	6.84	16.52
2010-05	7.06	14.66	0.20	6.84	16.70
...	...	...	...	...	...
2019-08	7.40	15.14	2.13	7.47	16.70
2019-09	7.40	15.15	2.04	7.47	16.70
2019-10	7.41	15.19	1.83	7.48	16.89
2019-11	7.42	15.21	1.55	7.49	16.70
2019-12	7.42	15.23	1.55	7.49	16.70

120 rows × 5 columns

In [34]:
```
df_new = df.diff(year).dropna()
```

In [37]:
```
#ADF test

X = df_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 = df_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 = df_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 = df_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 = df_new["NGDP"].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")
```

```
ADF Statistic: -2.436319
p-value: 0.131747
Critical Values:
        1%: -3.501
        5%: -2.892
        10%: -2.583
Failed to Reject Ho - Time Series is Non-Stationary
ADF Statistic: -1.605273
p-value: 0.480944
Critical Values:
        1%: -3.502
        5%: -2.893
        10%: -2.583
Failed to Reject Ho - Time Series is Non-Stationary
ADF Statistic: -1.410472
p-value: 0.577246
Critical Values:
        1%: -3.502
        5%: -2.893
        10%: -2.583
Failed to Reject Ho - Time Series is Non-Stationary
ADF Statistic: -1.476699
p-value: 0.545005
Critical Values:
        1%: -3.502
        5%: -2.893
        10%: -2.583
Failed to Reject Ho - Time Series is Non-Stationary
ADF Statistic: -3.371436
p-value: 0.011972
Critical Values:
        1%: -3.500
        5%: -2.892
        10%: -2.583
Reject Ho - Time Series is Stationary
```

In [35]:
```
df = df_new.diff(year).dropna()
```

In [36]:
```
#ADF test

X = df["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 = df["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 = df["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 = df["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 = df["NGDP"].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")
```

```
ADF Statistic: -4.258588
p-value: 0.000524
Critical Values:
        1%: -3.512
        5%: -2.897
        10%: -2.586
Reject Ho - Time Series is Stationary
ADF Statistic: 0.454911
p-value: 0.983436
Critical Values:
        1%: -3.512
        5%: -2.897
        10%: -2.586
Failed to Reject Ho - Time Series is Non-Stationary
ADF Statistic: -0.741620
p-value: 0.835673
Critical Values:
        1%: -3.512
        5%: -2.897
        10%: -2.586
Failed to Reject Ho - Time Series is Non-Stationary
ADF Statistic: -4.414979
p-value: 0.000279
Critical Values:
        1%: -3.512
        5%: -2.897
        10%: -2.586
Reject Ho - Time Series is Stationary
ADF Statistic: -4.548336
p-value: 0.000161
Critical Values:
        1%: -3.511
        5%: -2.897
        10%: -2.585
Reject Ho - Time Series is Stationary
```

In [25]:
```
## 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
```

Out[25]:

DATE	M4	TA	FFR	CC	NGDP
2012-01	0.026877	0.217509	-0.263951	0.038479	-0.007866
2012-02	0.015046	0.154969	-0.280712	0.036645	-0.007444
2012-03	0.000253	0.048978	-0.200944	0.023904	-0.004733
2012-04	-0.005432	-0.035920	0.012115	0.001934	-0.000171
2012-05	-0.027124	-0.130555	-0.156182	0.009240	-0.001449
...	...	...	...	...	...
2019-08	-0.015245	-0.123489	-0.606620	-0.002582	0.002337
2019-09	-0.008763	-0.119885	-0.575575	-0.019963	0.005972
2019-10	0.000413	-0.061525	-1.294853	0.007059	0.001991
2019-11	0.017875	-0.011930	-1.393046	0.001401	0.003774
2019-12	0.002063	-0.010571	-1.518892	0.010720	0.001099

96 rows × 5 columns

In [26]:
```
residuals.corr()[residuals.corr().abs() < 1].mul(-1).fillna(1).round(2)
```

Out[26]:

	M4	TA	FFR	CC	NGDP
M4	1.00	-0.43	-0.23	-0.11	-0.02
TA	-0.43	1.00	-0.29	-0.18	0.19
FFR	-0.23	-0.29	1.00	0.24	0.03
CC	-0.11	-0.18	0.24	1.00	0.25
NGDP	-0.02	0.19	0.03	0.25	1.00

In [27]:
```
# !pip install pingouin
import pingouin
df.pcorr().round(2)
```

Out[27]:

	M4	TA	FFR	CC	NGDP
M4	1.00	-0.43	-0.23	-0.11	-0.02
TA	-0.43	1.00	-0.29	-0.18	0.19
FFR	-0.23	-0.29	1.00	0.24	0.03
CC	-0.11	-0.18	0.24	1.00	0.25
NGDP	-0.02	0.19	0.03	0.25	1.00

In [28]:
```
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[28]:

	M4	TA	FFR	CC	NGDP
M4	NaN	0.00	0.02	0.30	0.85
TA	0.00	NaN	0.00	0.08	0.06
FFR	0.02	0.00	NaN	0.02	0.74
CC	0.30	0.08	0.02	NaN	0.02
NGDP	0.85	0.06	0.74	0.02	NaN

In [29]:
```
residuals
```

Out[29]:

DATE	M4	TA	FFR	CC	NGDP
2012-01	0.026877	0.217509	-0.263951	0.038479	-0.007866
2012-02	0.015046	0.154969	-0.280712	0.036645	-0.007444
2012-03	0.000253	0.048978	-0.200944	0.023904	-0.004733
2012-04	-0.005432	-0.035920	0.012115	0.001934	-0.000171
2012-05	-0.027124	-0.130555	-0.156182	0.009240	-0.001449
...	...	...	...	...	...
2019-08	-0.015245	-0.123489	-0.606620	-0.002582	0.002337
2019-09	-0.008763	-0.119885	-0.575575	-0.019963	0.005972
2019-10	0.000413	-0.061525	-1.294853	0.007059	0.001991
2019-11	0.017875	-0.011930	-1.393046	0.001401	0.003774
2019-12	0.002063	-0.010571	-1.518892	0.010720	0.001099

96 rows × 5 columns

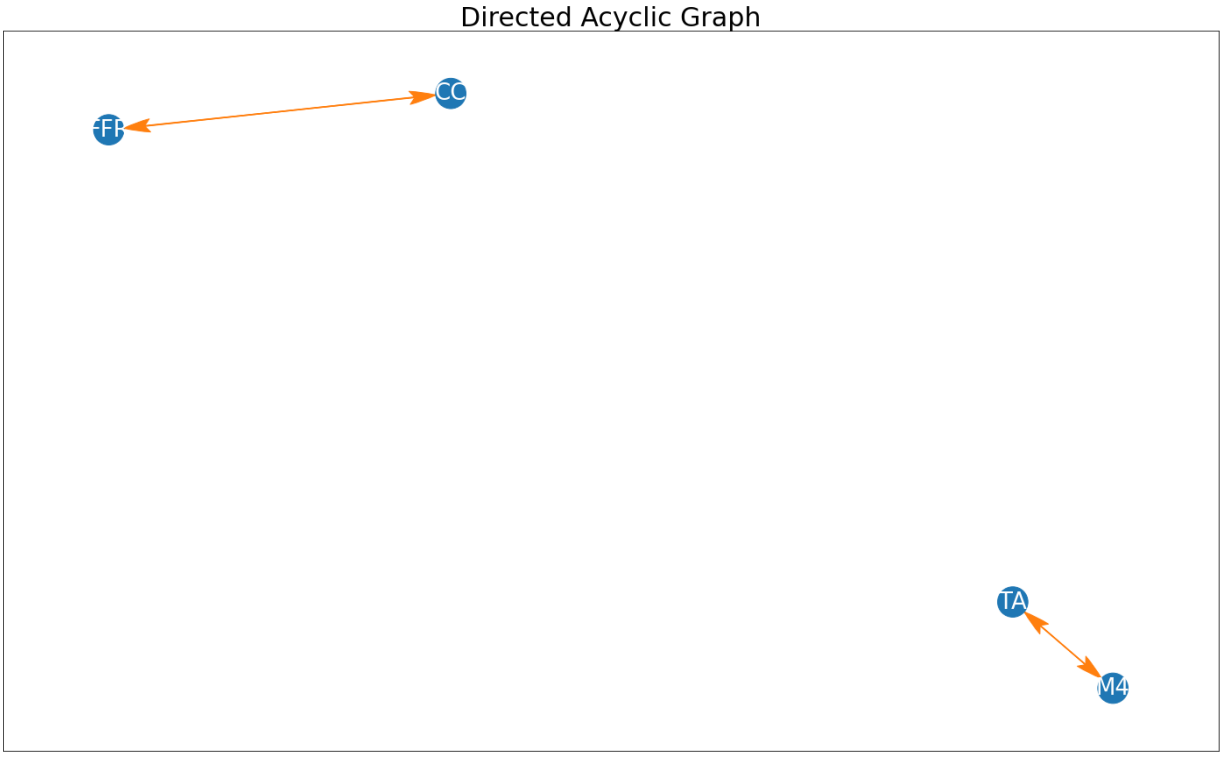
In [30]:
```
##DAG

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 [31]:
```
## 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 = "pdag",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: 67% 2/3 [00:00<00:00, 31.25it/s]

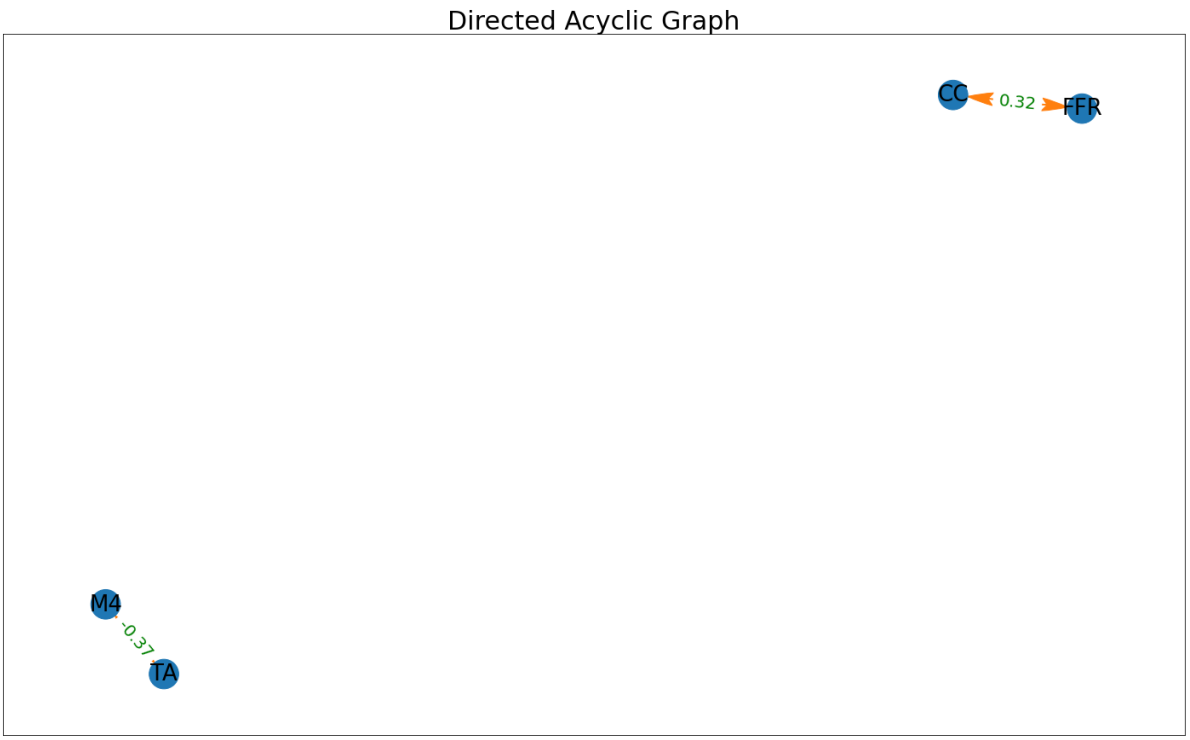
In [32]:
```
from matplotlib.patches import ArrowStyle

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[32]:
```
OutEdgeView([('FFR', 'CC'), ('CC', 'FFR'), ('TA', 'M4'), ('M4', 'TA')])
```



In [33]:
```
## 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', 'CC') []
('CC', 'FFR') []
('TA', 'M4') []
('M4', 'TA') []
```



In [34]:
```
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 [35]:
```
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", "NGDP"],
                   1:["TA", "CC", "FFR"],
                   2:["M4","CC", "FFR"],
                   3:["TA", "CC", "FFR", "M4"],
                   4:["TA", "NGDP", "FFR", "CC","M4"],
                   5:["CC", "NGDP","M4"]}
# 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: 100% 1/1 [00:00<00:00, 32.00it/s]

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

variables: 1: 100% 63.92it/s]

Working for n conditional variables: 1: 100% 1/1 [00:00<00:00, 16.18it/s]

Working for n conditional variables: 1: 100% 1/1 [00:00<00:00, 32.02it/s]

Working for n conditional variables: 1: 100% 1/1 [00:00<00:00, 21.35it/s]

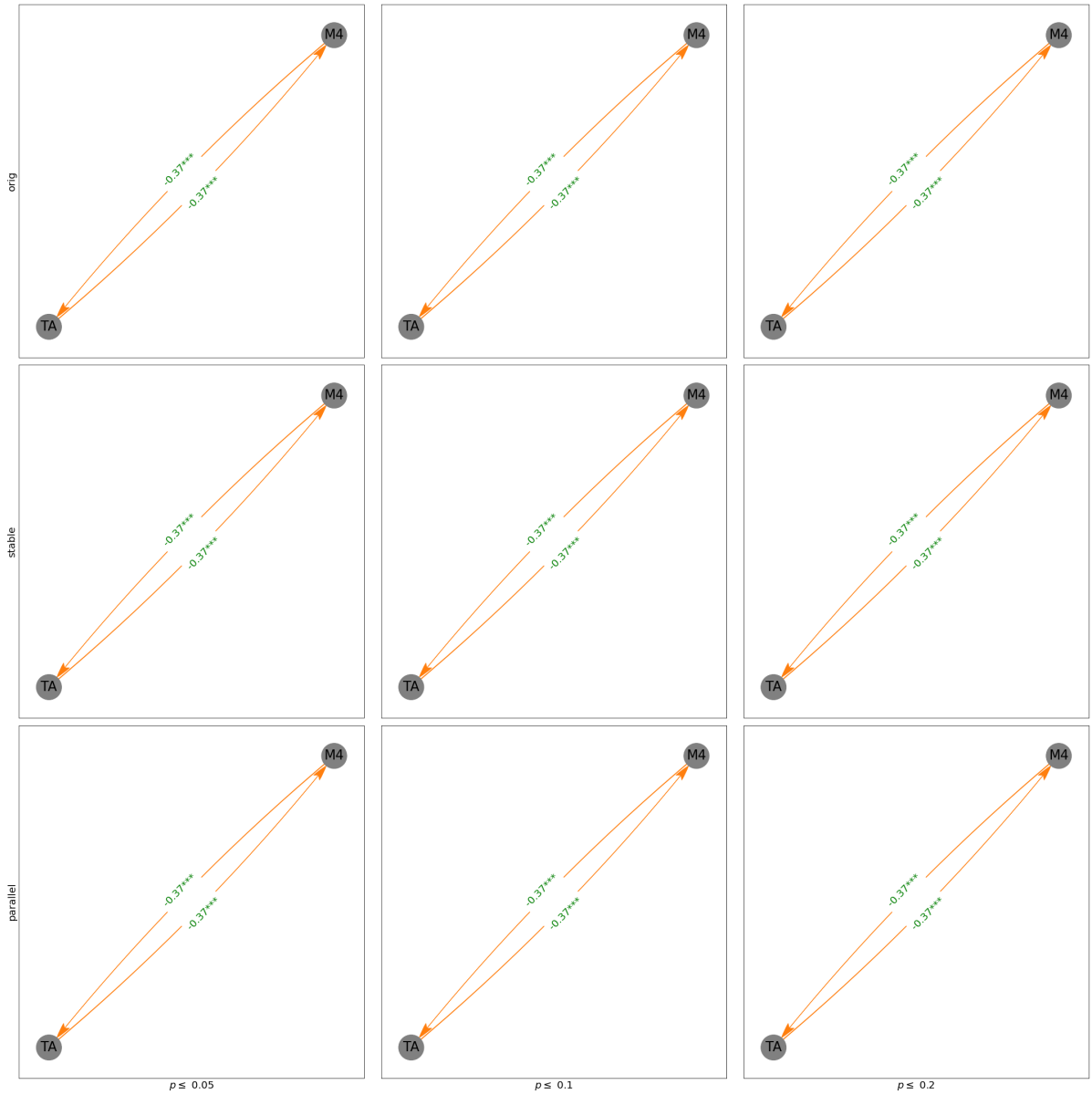
Working for n conditional variables: 1: 100% 1/1 [00:00<00:00, 13.98it/s]

Working for n conditional variables: 1: 100% 1/1 [00:00<00:00, 13.89it/s]

Working for n conditional variables: 1: 100% 1/1 [00:00<00:00, 9.82it/s]

Working for n conditional variables: 1: 100% 1/1 [00:00<00:00, 8.30it/s]

'M4', 'TA', 'NGD'



Working for n conditional variables: 1: 100% 1/1 [00:00<00:00, 15.98it/s]

Working for n conditional variables: 1: 100% 1/1 [00:00<00:00, 19.87it/s]

Working for n conditional variables: 1: 100% 1/1 [00:00<00:00, 12.30it/s]

Working for n conditional variables: 1: 100% 1/1 [00:00<00:00, 9.68it/s]

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

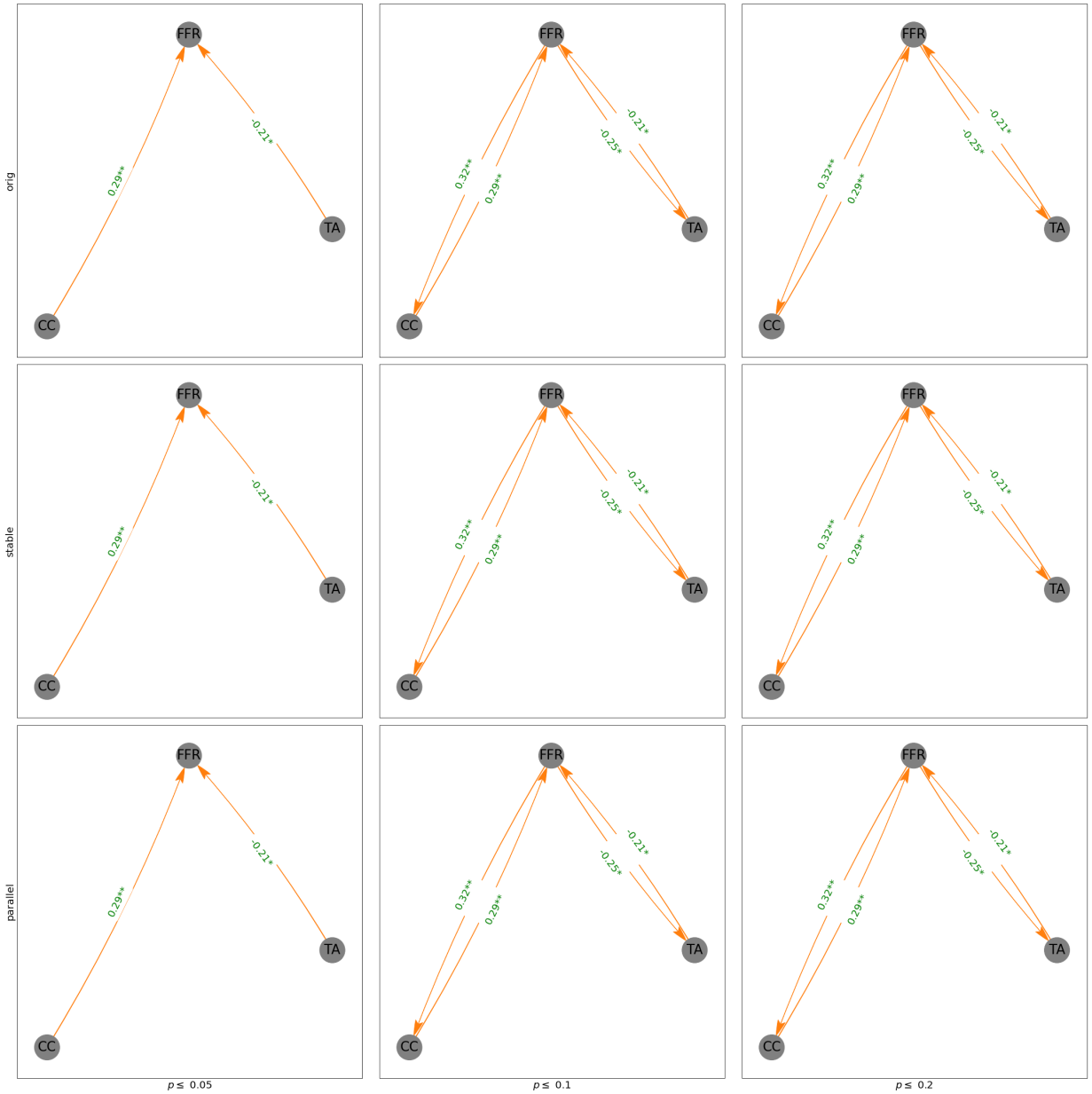
Working for n conditional variables: 1: 100% | 1/1 [00:00<00:00, 12.11it/s]

Working for n conditional variables: 1: 100% | 1/1 [00:00<00:00, 18.39it/s]

Working for n conditional variables: 1: 100% | 1/1 [00:00<00:00, 14.39it/s]

Working for n conditional variables: 1: 100% | 1/1 [00:00<00:00, 9.99it/s]

'TA', 'CC', 'FFR'



Working for n conditional variables: 1: 100% | 1/1 [00:00<00:00, 14.64it/s]

Working for n conditional variables: 1: 100% | 1/1 [00:00<00:00, 14.19it/s]

Working for n conditional variables: 1: 100% | 1/1 [00:00<00:00, 18.37it/s]

Working for n conditional variables: 1: 100% | 1/1 [00:00<00:00, 11.66it/s]

Working for n conditional variables: 1: 100% | 1/1 [00:00<00:00, 31.27it/s]

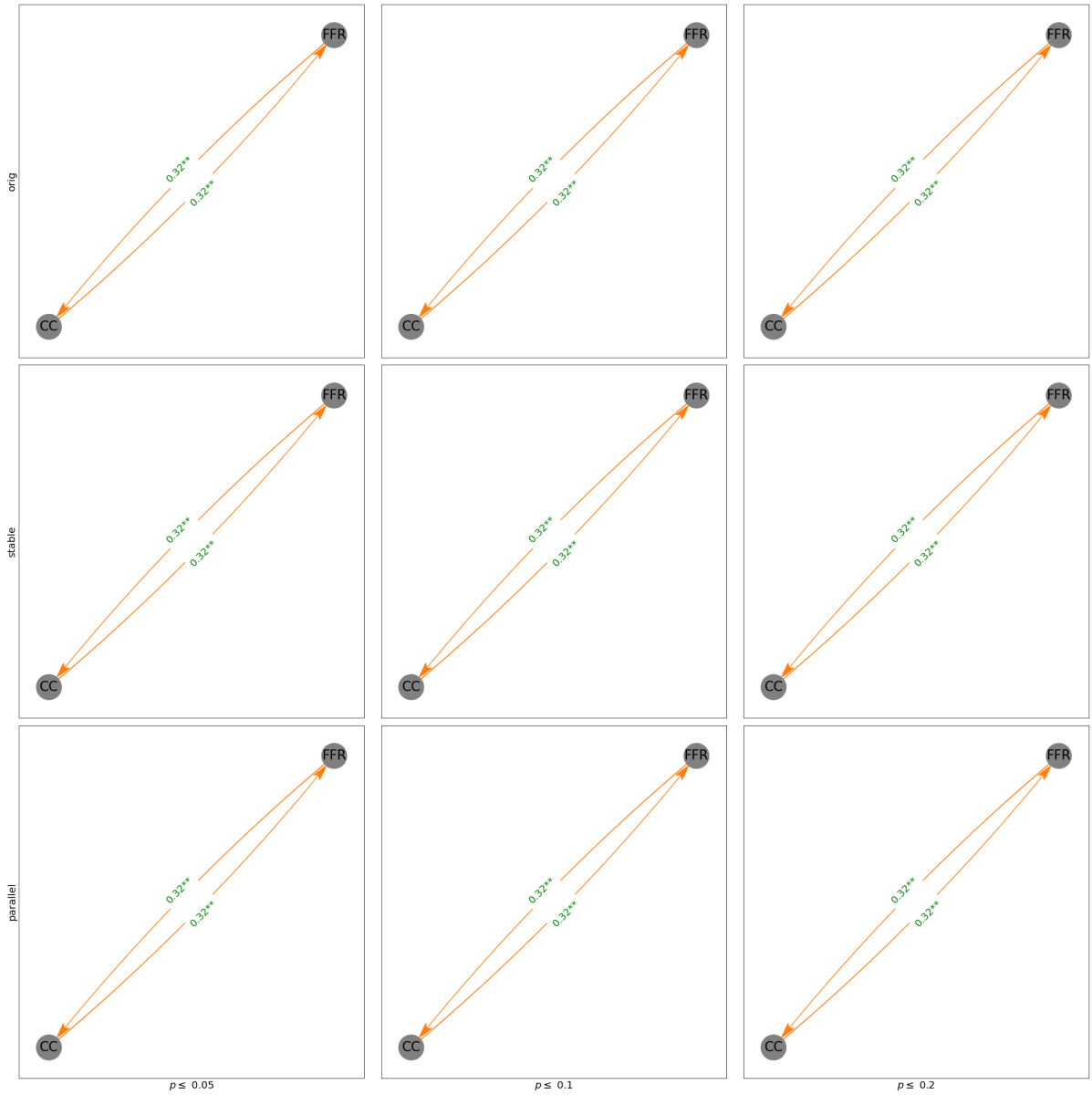
Working for n conditional variables: 1: 100% | 1/1 [00:00<00:00, 21.33it/s]

Working for n conditional variables: 1: 100% | 1/1 [00:00<00:00, 21.33it/s]

Working for n conditional variables: 1: 100% | 1/1 [00:00<00:00, 20.84it/s]

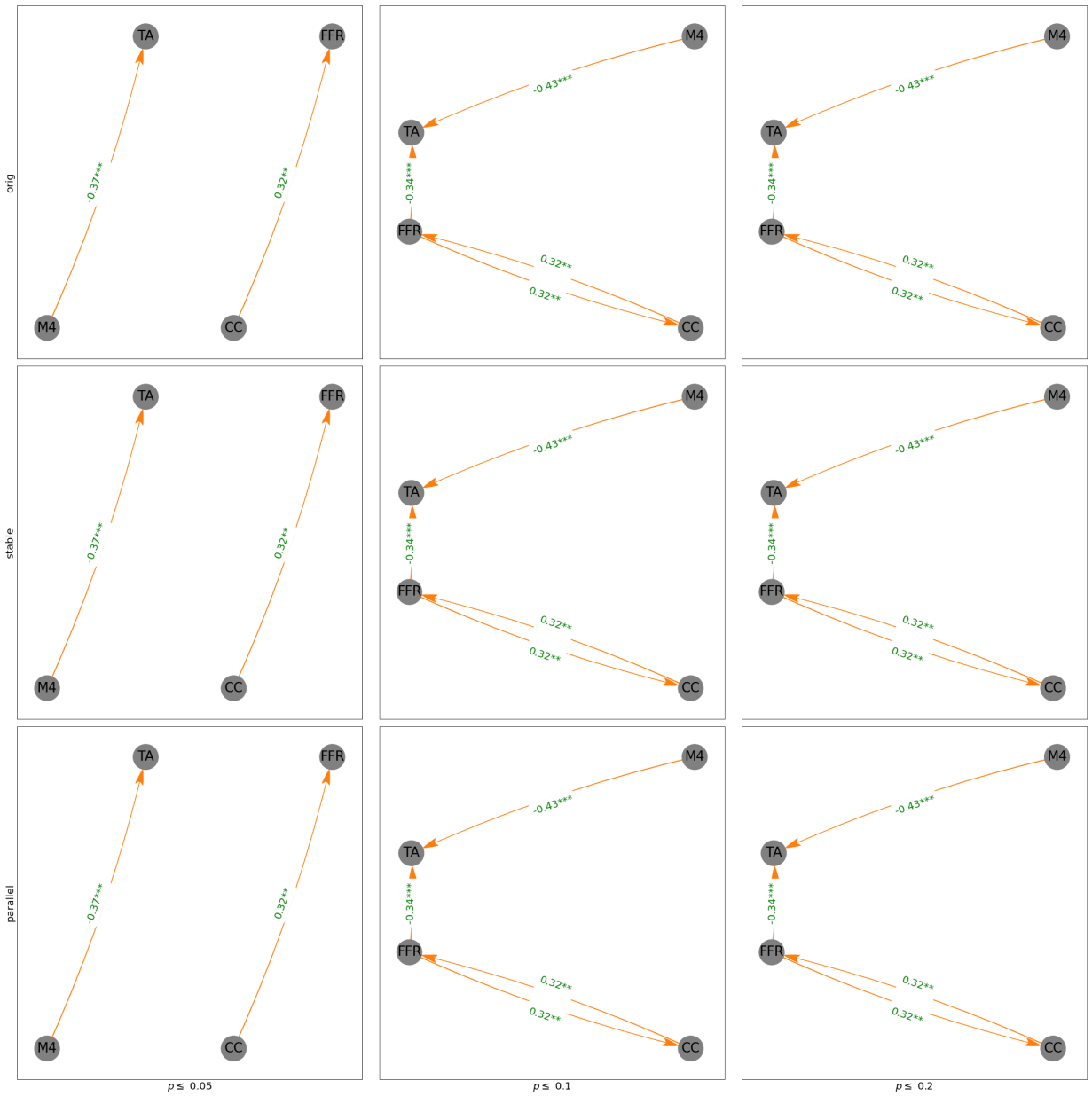
Working for n conditional variables: 1: 100% | 1/1 [00:00<00:00, 15.06it/s]

'M4', 'CC', 'FFR'



Working for n conditional variables: 2: 100%		2/2 [00:00<00:00, 26.57it/s]
Working for n conditional variables: 2: 100%		2/2 [00:00<00:00, 23.66it/s]
Working for n conditional variables: 2: 100%		2/2 [00:00<00:00, 16.51it/s]
Working for n conditional variables: 2: 100%		2/2 [00:00<00:00, 26.92it/s]
Working for n conditional variables: 2: 100%		2/2 [00:00<00:00, 24.86it/s]
Working for n conditional variables: 2: 100%		2/2 [00:00<00:00, 12.20it/s]
Working for n conditional variables: 2: 100%		2/2 [00:00<00:00, 11.86it/s]
Working for n conditional variables: 2: 100%		2/2 [00:00<00:00, 19.06it/s]
Working for n conditional variables: 2: 100%		2/2 [00:00<00:00, 12.68it/s]

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



Working for n conditional variables: 3: 100% | 3/3 [00:00<00:00, 39.49it/s]

Working for n conditional variables: 3: 100% | 3/3 [00:00<00:00, 32.85it/s]

Working for n conditional variables: 3: 100% | 3/3 [00:00<00:00, 27.79it/s]

Working for n conditional variables: 3: 100% | 3/3 [00:00<00:00, 29.85it/s]

Working for n conditional | 3/3 [00:00<00:00,

variables: 3: 100% 31.24it/s]

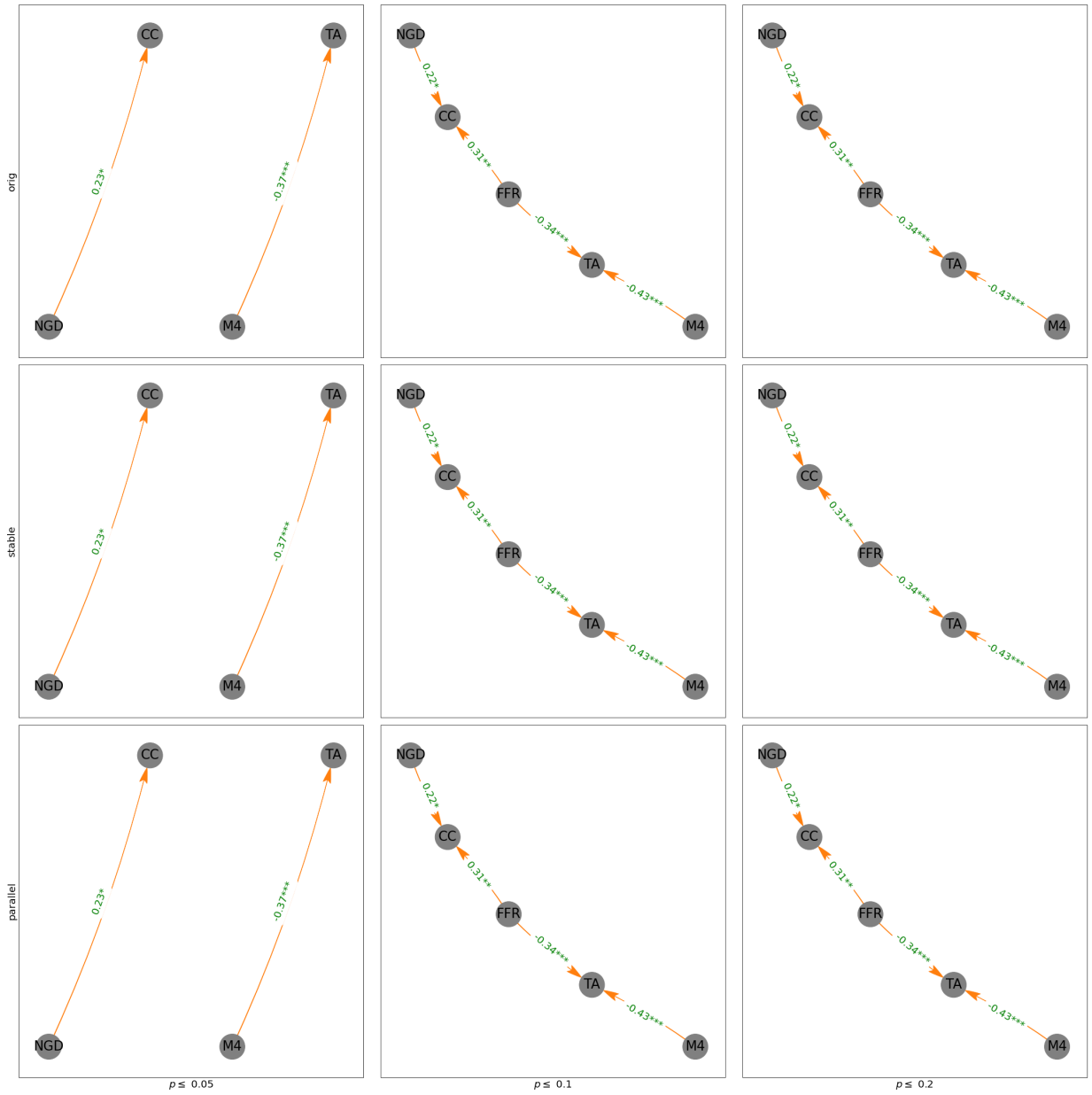
Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 12.73it/s]

Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 13.24it/s]

Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 13.51it/s]

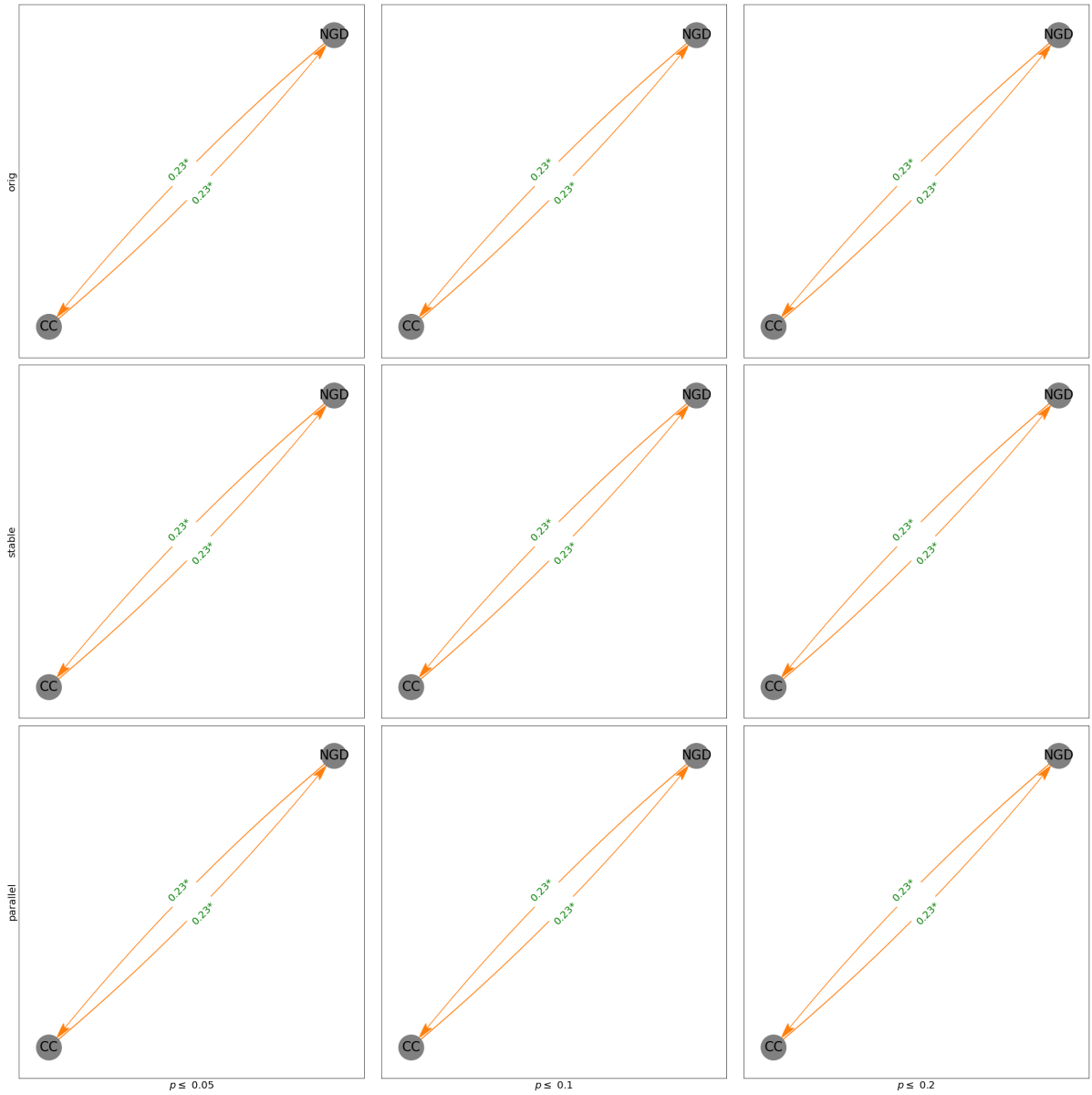
Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 9.32it/s]

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



Working for n conditional variables: 1: 100%		1/1 [00:00<00:00, 21.33it/s]
Working for n conditional variables: 1: 100%		1/1 [00:00<00:00, 32.00it/s]
Working for n conditional variables: 1: 100%		1/1 [00:00<00:00, 21.33it/s]
Working for n conditional variables: 1: 100%		1/1 [00:00<00:00, 32.00it/s]
Working for n conditional variables: 1: 100%		1/1 [00:00<00:00, 21.33it/s]
Working for n conditional variables: 1: 100%		1/1 [00:00<00:00, 18.88it/s]
Working for n conditional variables: 1: 100%		1/1 [00:00<00:00, 18.38it/s]
Working for n conditional variables: 1: 100%		1/1 [00:00<00:00, 21.34it/s]
Working for n conditional variables: 1: 100%		1/1 [00:00<00:00, 14.22it/s]

'CC', 'NGD', 'M4'



Out[35]: OutEdgeView([('CC', 'NGD'), ('NGD', 'CC')])

TWICE DIFF YEARLY DATA WITH M4 TA FFR CC NGDP I (WITH INFLATION)

In [44]:
```
#data cleaning, importing

d_parser = lambda x: pd.datetime.strptime(x, '%m/%d/%Y')
df = pd.read_csv('M4-11.csv', parse_dates=['Date'], date_parser=d_parser)
df
```

Out[44]:

	Date	Log M4 including Treasuries	Log Total Assets	Effective Federal Funds Rate (%)	Log Currency in Circulation ($ Bil)	Log Nominal GDP (Bln)	Unemployment Rate	Inflation Rate
0	2010-01-31	7.07	14.63	0.11	6.83	16.51	9.8	2.62
1	2010-02-28	7.06	14.63	0.13	6.83	16.70	9.8	2.15
2	2010-03-31	7.05	14.65	0.17	6.84	16.70	9.9	2.29
3	2010-04-30	7.06	14.66	0.20	6.84	16.52	9.9	2.21
4	2010-05-31	7.06	14.66	0.20	6.84	16.70	9.6	2.00
...	...	...	...	...	...	...	...	...
115	2019-08-31	7.40	15.14	2.13	7.47	16.70	3.7	1.74
116	2019-09-30	7.40	15.15	2.04	7.47	16.70	3.5	1.72
117	2019-10-31	7.41	15.19	1.83	7.48	16.89	3.6	1.77
118	2019-11-30	7.42	15.21	1.55	7.49	16.70	3.6	2.04
119	2019-12-31	7.42	15.23	1.55	7.49	16.70	3.6	2.26

120 rows × 8 columns

In [45]:
```
df['Date_at_year_month'] = df['Date'].dt.strftime('%Y-%m')
column_names = {'Date_at_year_month':'DATE',
                'Log M4 including Treasuries':'M4',
                'Log Total Assets': 'TA',
                'Log Currency in Circulation ($ Bil)':'CC',
                'Log Nominal GDP (Bln)':'NGDP',
               'Effective Federal Funds Rate (%)': 'FFR',
               'Unemployment Rate': 'U',
               'Inflation Rate': 'I'}


# rename columns
df = df.rename(columns = column_names)
df = df.set_index('DATE')
df = df.drop(['Date'], axis = 1)
df = df.drop(['U'], axis = 1)
df
```

Out[45]:

DATE	M4	TA	FFR	CC	NGDP	I
2010-01	7.07	14.63	0.11	6.83	16.51	2.62
2010-02	7.06	14.63	0.13	6.83	16.70	2.15
2010-03	7.05	14.65	0.17	6.84	16.70	2.29
2010-04	7.06	14.66	0.20	6.84	16.52	2.21
2010-05	7.06	14.66	0.20	6.84	16.70	2.00
...	...	...	...	...	...	...
2019-08	7.40	15.14	2.13	7.47	16.70	1.74
2019-09	7.40	15.15	2.04	7.47	16.70	1.72
2019-10	7.41	15.19	1.83	7.48	16.89	1.77
2019-11	7.42	15.21	1.55	7.49	16.70	2.04
2019-12	7.42	15.23	1.55	7.49	16.70	2.26

120 rows × 6 columns

In [ ]:
```
```

In [47]:
```
df_new = df.diff(year).dropna()
```

In [ ]:
```
#ADF test

X = data["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["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["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["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["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")
```

In [48]:
```
df = df_new.diff(year).dropna()
```

In [49]:
```
#ADF test

X = df["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 = df["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 = df["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 = df["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 = df["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")

X = df["NGDP"].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")
```

```
ADF Statistic: -4.258588
p-value: 0.000524
Critical Values:
        1%: -3.512
        5%: -2.897
        10%: -2.586
Reject Ho - Time Series is Stationary
ADF Statistic: 0.454911
p-value: 0.983436
Critical Values:
        1%: -3.512
        5%: -2.897
        10%: -2.586
Failed to Reject Ho - Time Series is Non-Stationary
ADF Statistic: -0.741620
p-value: 0.835673
Critical Values:
        1%: -3.512
        5%: -2.897
        10%: -2.586
Failed to Reject Ho - Time Series is Non-Stationary
ADF Statistic: -4.414979
p-value: 0.000279
Critical Values:
        1%: -3.512
        5%: -2.897
        10%: -2.586
Reject Ho - Time Series is Stationary
ADF Statistic: -1.902375
p-value: 0.330945
```

```
Critical Values:
        1%: -3.512
        5%: -2.897
        10%: -2.586
Failed to Reject Ho - Time Series is Non-Stationary
ADF Statistic: -4.548336
p-value: 0.000161
Critical Values:
        1%: -3.511
        5%: -2.897
        10%: -2.585
Reject Ho - Time Series is Stationary
```

In [ ]:

In [40]:
```
## 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
```

Out[40]:

DATE	M4	TA	FFR	CC	NGDP	I
2012-01	0.026612	0.109960	-0.302653	0.038684	-0.006649	1.601119
2012-02	0.014963	0.098759	-0.289010	0.036697	-0.006987	0.490945
2012-03	0.000284	0.042069	-0.192191	0.023868	-0.004783	-0.190379
2012-04	-0.005247	0.009220	0.042700	0.001776	-0.000938	-1.135857
2012-05	-0.026778	-0.030169	-0.096403	0.008946	-0.002849	-2.101518
...	...	...	...	...	...	...
2019-08	-0.015133	-0.069916	-0.577593	-0.002673	0.001844	-0.670710
2019-09	-0.008786	-0.093149	-0.569447	-0.019934	0.005977	0.153285
2019-10	0.000496	-0.029211	-1.257802	0.006985	0.001610	-0.513118
2019-11	0.017807	-0.021194	-1.378861	0.001455	0.003978	0.394810
2019-12	0.001935	-0.032541	-1.513115	0.010824	0.001615	0.787822

96 rows × 6 columns

In [41]:
```
residuals.corr()[residuals.corr().abs() < 1].mul(-1).fillna(1).round(2)
```

Out[41]:

	M4	TA	FFR	CC	NGDP	I
M4	1.00	-0.37	-0.23	-0.11	-0.02	0.02
TA	-0.37	1.00	-0.31	-0.14	0.23	0.51
FFR	-0.23	-0.31	1.00	0.24	0.05	0.13
CC	-0.11	-0.14	0.24	1.00	0.24	-0.02
NGDP	-0.02	0.23	0.05	0.24	1.00	-0.13
I	0.02	0.51	0.13	-0.02	-0.13	1.00

In [42]:
```
import pingouin
df.pcorr().round(2)
```

Out[42]:

	M4	TA	FFR	CC	NGDP	I
M4	1.00	-0.37	-0.23	-0.11	-0.02	0.02
TA	-0.37	1.00	-0.31	-0.14	0.23	0.51
FFR	-0.23	-0.31	1.00	0.24	0.05	0.13
CC	-0.11	-0.14	0.24	1.00	0.24	-0.02
NGDP	-0.02	0.23	0.05	0.24	1.00	-0.13
I	0.02	0.51	0.13	-0.02	-0.13	1.00

In [43]:
```
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[43]:

	M4	TA	FFR	CC	NGDP	I
M4	NaN	0.00	0.02	0.30	0.86	0.88
TA	0.00	NaN	0.00	0.17	0.02	0.00
FFR	0.02	0.00	NaN	0.02	0.62	0.19
CC	0.30	0.17	0.02	NaN	0.02	0.84
NGDP	0.86	0.02	0.62	0.02	NaN	0.21
I	0.88	0.00	0.19	0.84	0.21	NaN

In [44]:
```
residuals
```

Out[44]:

DATE	M4	TA	FFR	CC	NGDP	I
2012-01	0.026612	0.109960	-0.302653	0.038684	-0.006649	1.601119
2012-02	0.014963	0.098759	-0.289010	0.036697	-0.006987	0.490945
2012-03	0.000284	0.042069	-0.192191	0.023868	-0.004783	-0.190379
2012-04	-0.005247	0.009220	0.042700	0.001776	-0.000938	-1.135857
2012-05	-0.026778	-0.030169	-0.096403	0.008946	-0.002849	-2.101518
...	...	...	...	...	...	...
2019-08	-0.015133	-0.069916	-0.577593	-0.002673	0.001844	-0.670710
2019-09	-0.008786	-0.093149	-0.569447	-0.019934	0.005977	0.153285
2019-10	0.000496	-0.029211	-1.257802	0.006985	0.001610	-0.513118
2019-11	0.017807	-0.021194	-1.378861	0.001455	0.003978	0.394810
2019-12	0.001935	-0.032541	-1.513115	0.010824	0.001615	0.787822

96 rows × 6 columns

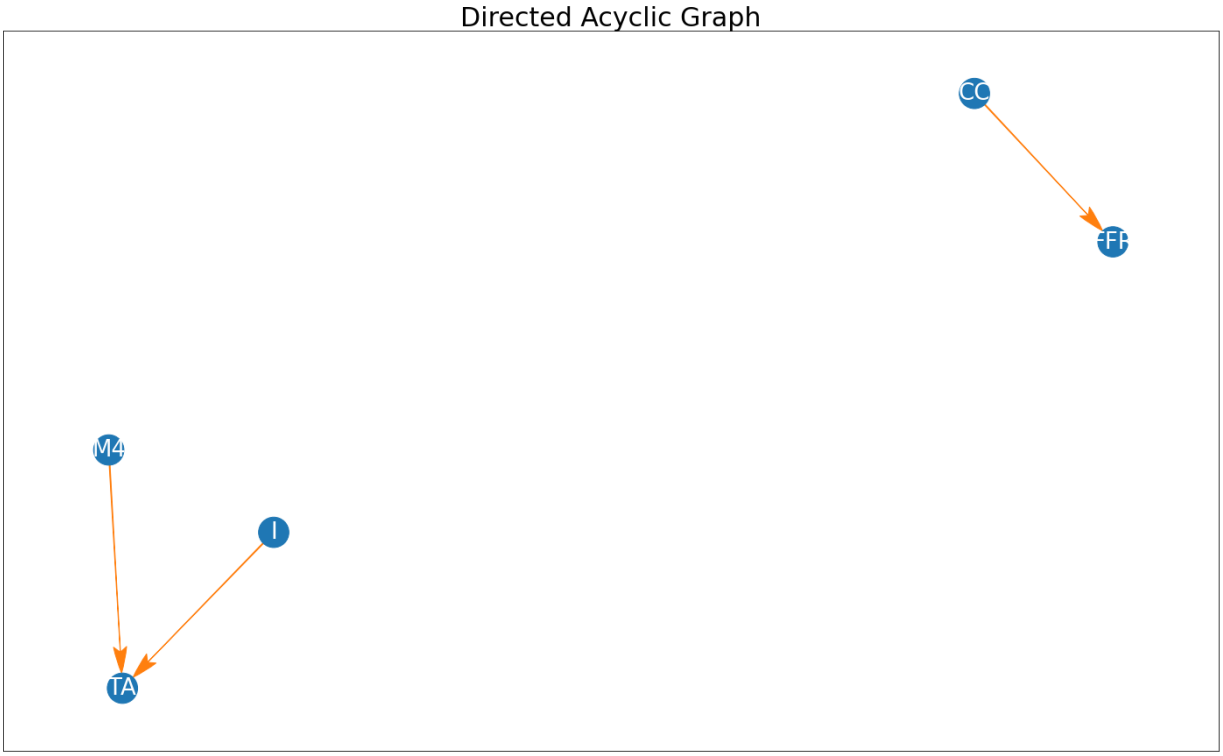
In [45]:
```
##DAG

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 [46]:
```
## 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: 3: 75% 3/4 [00:00<00:00, 28.86it/s]

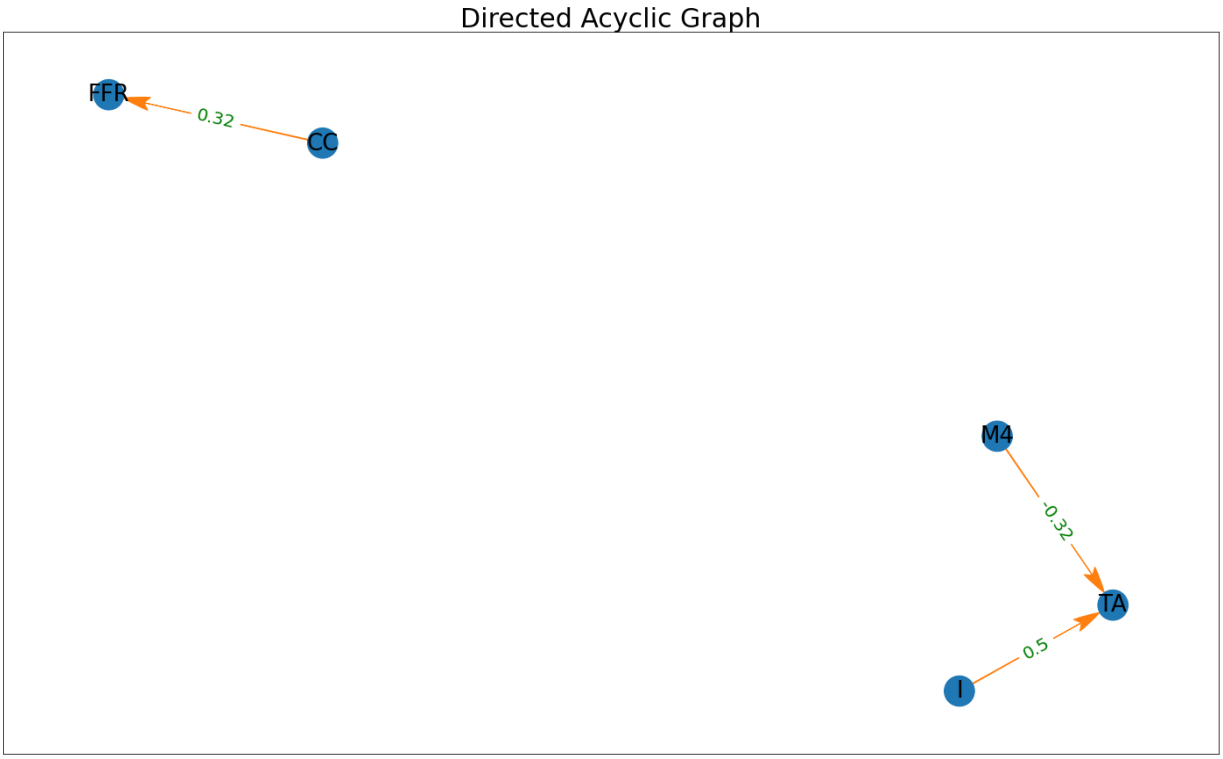
In [47]:
```
from matplotlib.patches import ArrowStyle

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[47]:
```
OutEdgeView([('I', 'TA'), ('M4', 'TA'), ('CC', 'FFR')])
```



In [48]:
```
## 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")
```

```
keep control: M4
('I', 'TA') ['M4']
keep control: I
('M4', 'TA') ['I']
('CC', 'FFR') []
```



In [49]:
```
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 [50]:
```
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:["NGDP", "TA", "M4", "I"],
                   1:["NGDP", "CC", "M4", "I"],
                   2:["TA", "CC", "FFR", "NGDP","I"],
                   3:["TA", "M4", "FFR", "NGDP","I"],
                   4:["TA", "M4", "FFR", "CC", "I"],
                   5:["TA", "M4", "FFR", "NGDP","CC", "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: 2: 100% 2/2 [00:00<00:00, 32.98it/s]

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

variables: 2: 100% 31.97it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 19.77it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 25.39it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 22.73it/s]

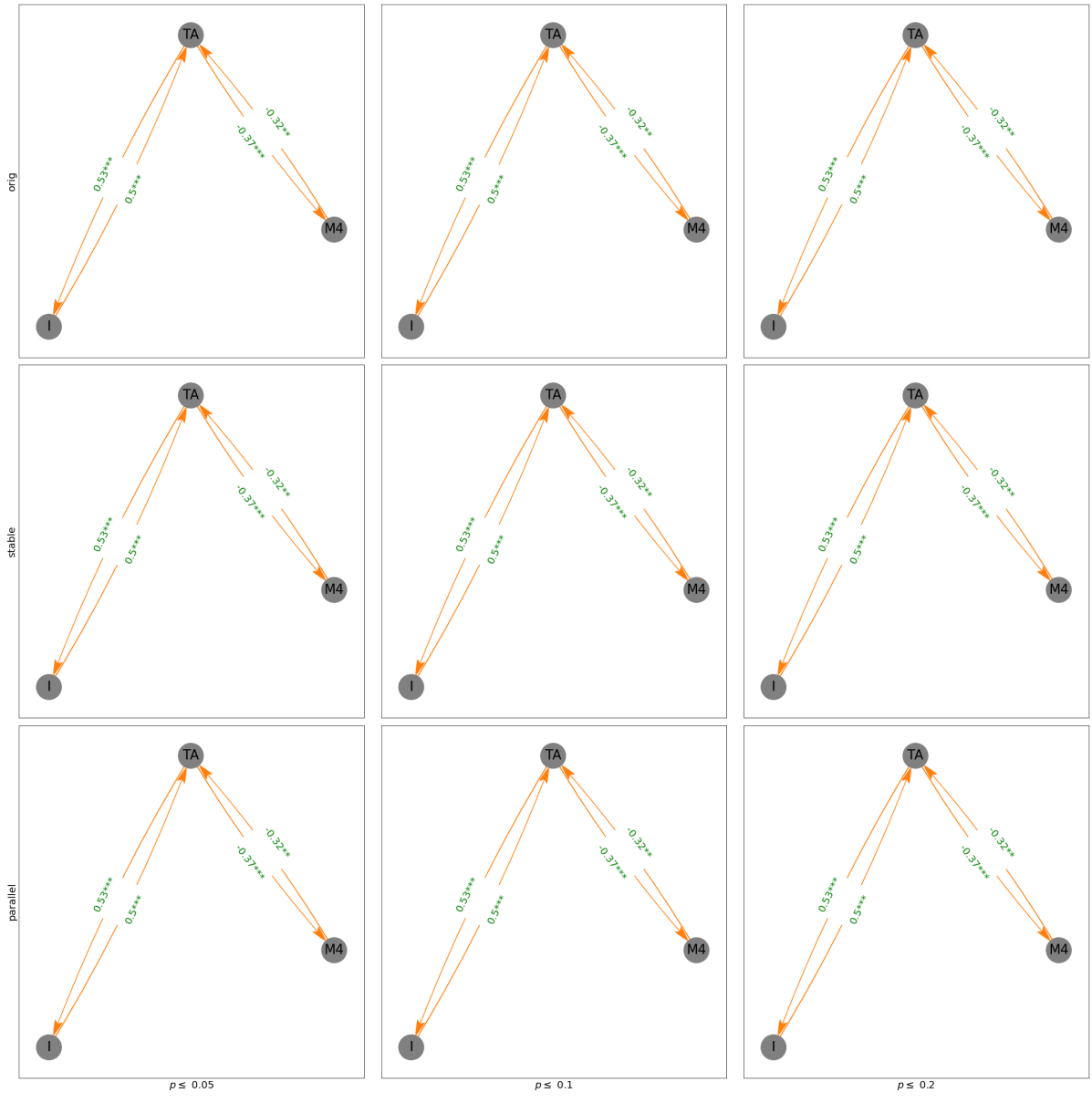
Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 16.57it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 20.83it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 21.36it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 15.62it/s]

'NGD', 'TA', 'M4', 'I'



Working for n conditional variables: 2: 100% | 2/2 [00:00<00:00, 3.58it/s]

Working for n conditional variables: 2: 100% | 2/2 [00:00<00:00, 64.01it/s]

Working for n conditional variables: 2: 100% | 2/2 [00:00<00:00, 39.53it/s]

Working for n conditional variables: 2: 100% | 2/2 [00:00<00:00, 64.01it/s]

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

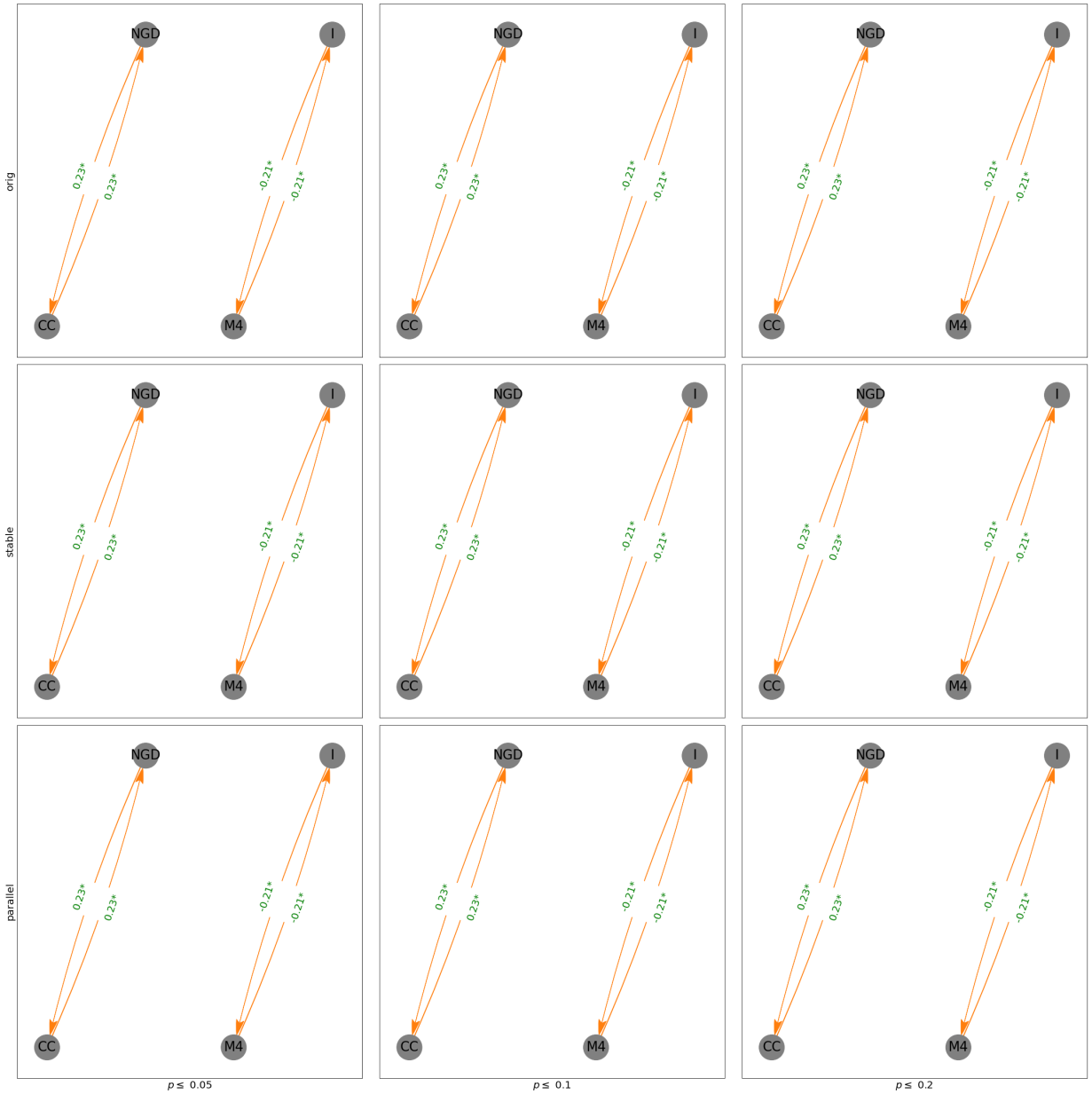
Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 35.90it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 31.35it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 39.86it/s]

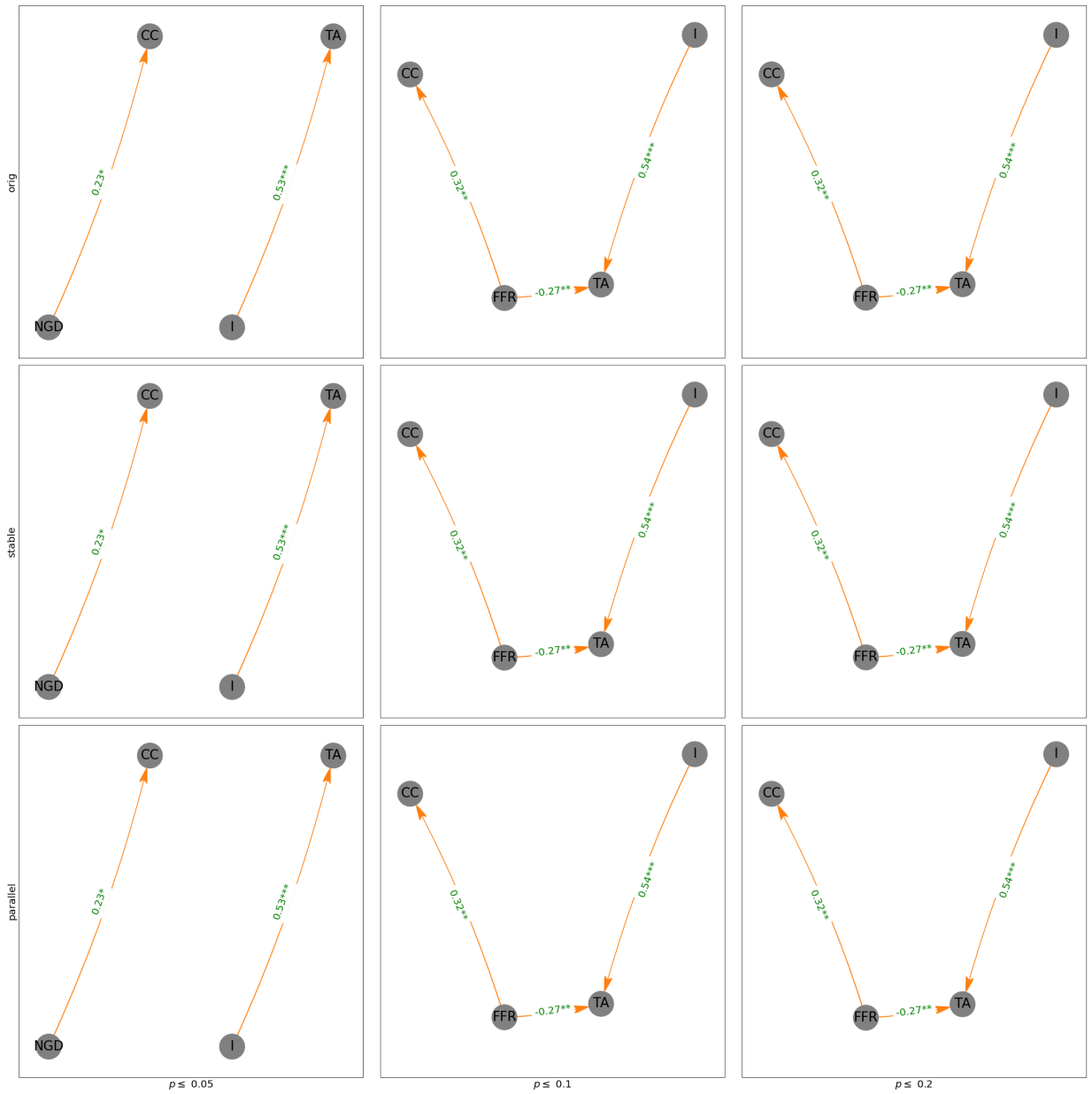
Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 26.54it/s]

'NGD', 'CC', 'M4', 'I'



Working for n conditional variables: 3: 100%		3/3 [00:00<00:00, 32.88it/s]
Working for n conditional variables: 3: 100%		3/3 [00:00<00:00, 37.43it/s]
Working for n conditional variables: 3: 100%		3/3 [00:00<00:00, 18.71it/s]
Working for n conditional variables: 3: 100%		3/3 [00:00<00:00, 17.18it/s]
Working for n conditional variables: 3: 100%		3/3 [00:00<00:00, 15.15it/s]
Working for n conditional variables: 3: 100%		3/3 [00:00<00:00, 10.49it/s]
Working for n conditional variables: 3: 100%		3/3 [00:00<00:00, 16.57it/s]
Working for n conditional variables: 3: 100%		3/3 [00:00<00:00, 15.24it/s]
Working for n conditional variables: 3: 100%		3/3 [00:00<00:00, 10.83it/s]

'TA', 'CC', 'FFR', 'NGD', 'I'



Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 18.64it/s]

Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 19.61it/s]

Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 16.64it/s]

Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 9.02it/s]

Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 14.88it/s]

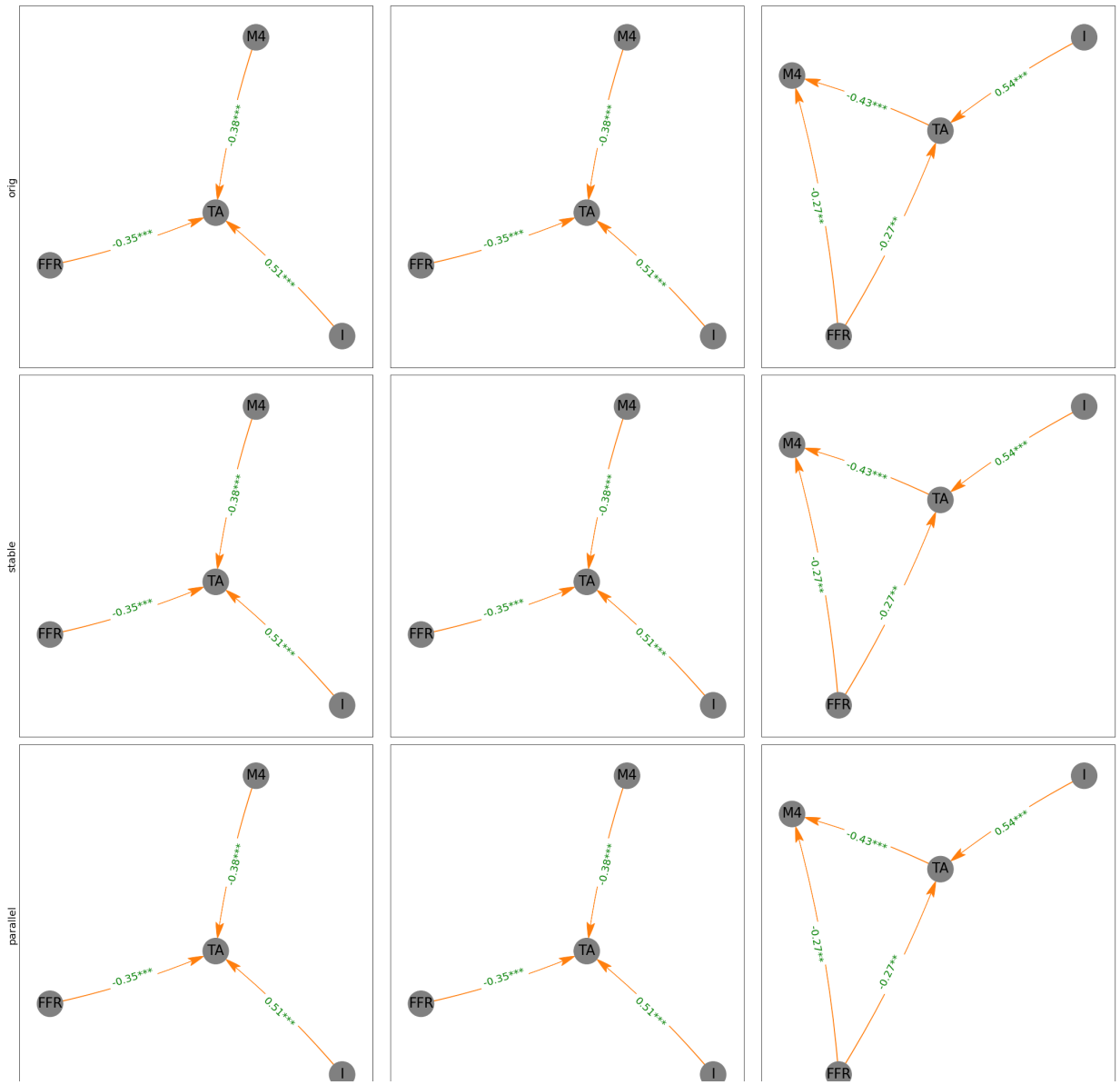
Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 11.30it/s]

Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 11.06it/s]

Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 12.22it/s]

Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 9.41it/s]

'TA', 'M4', 'FFR', 'NGD', 'I'



Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 17.54it/s]

Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 17.36it/s]

Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 13.45it/s]

Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 15.74it/s]

Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 16.82it/s]

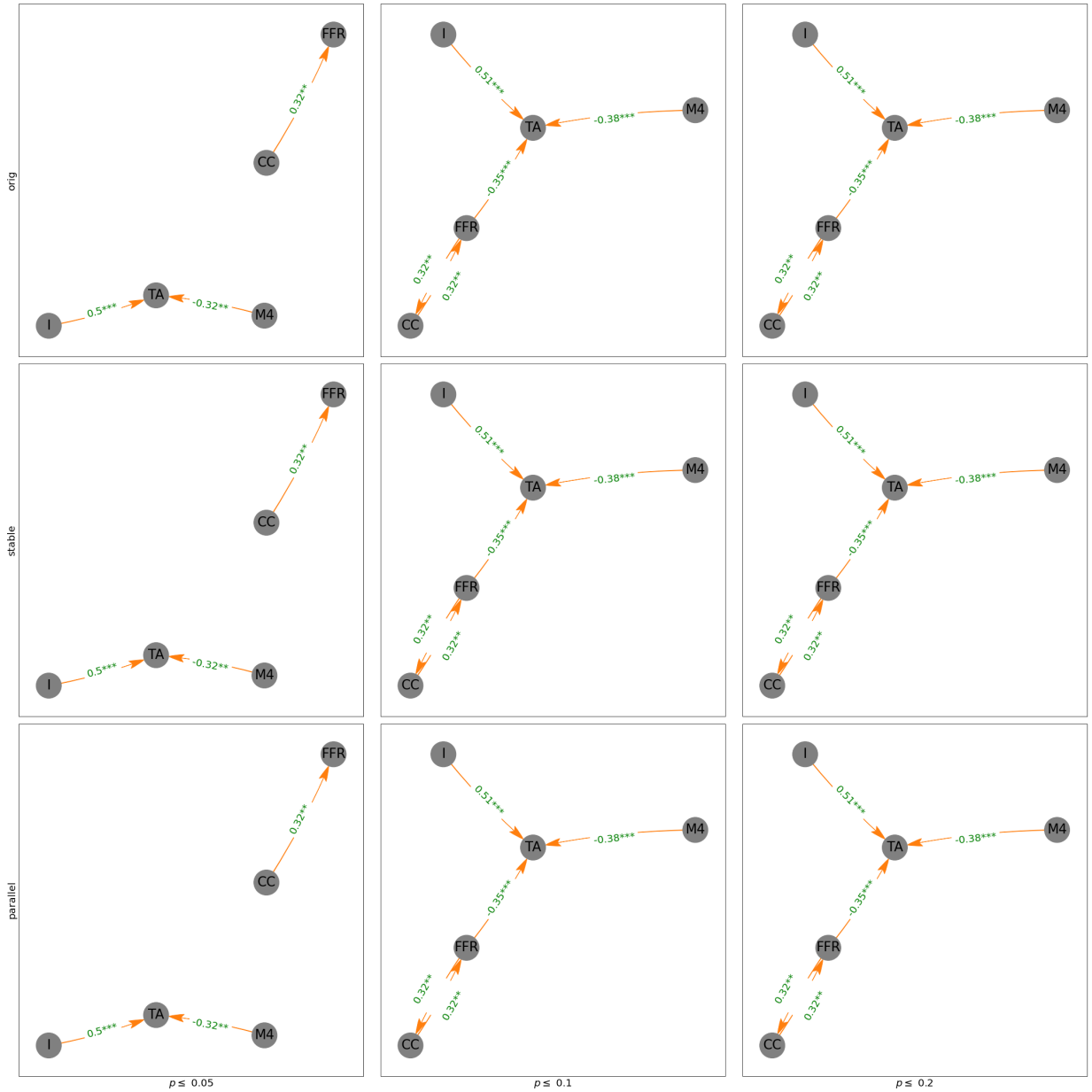
Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 11.44it/s]

Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 11.70it/s]

Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 11.85it/s]

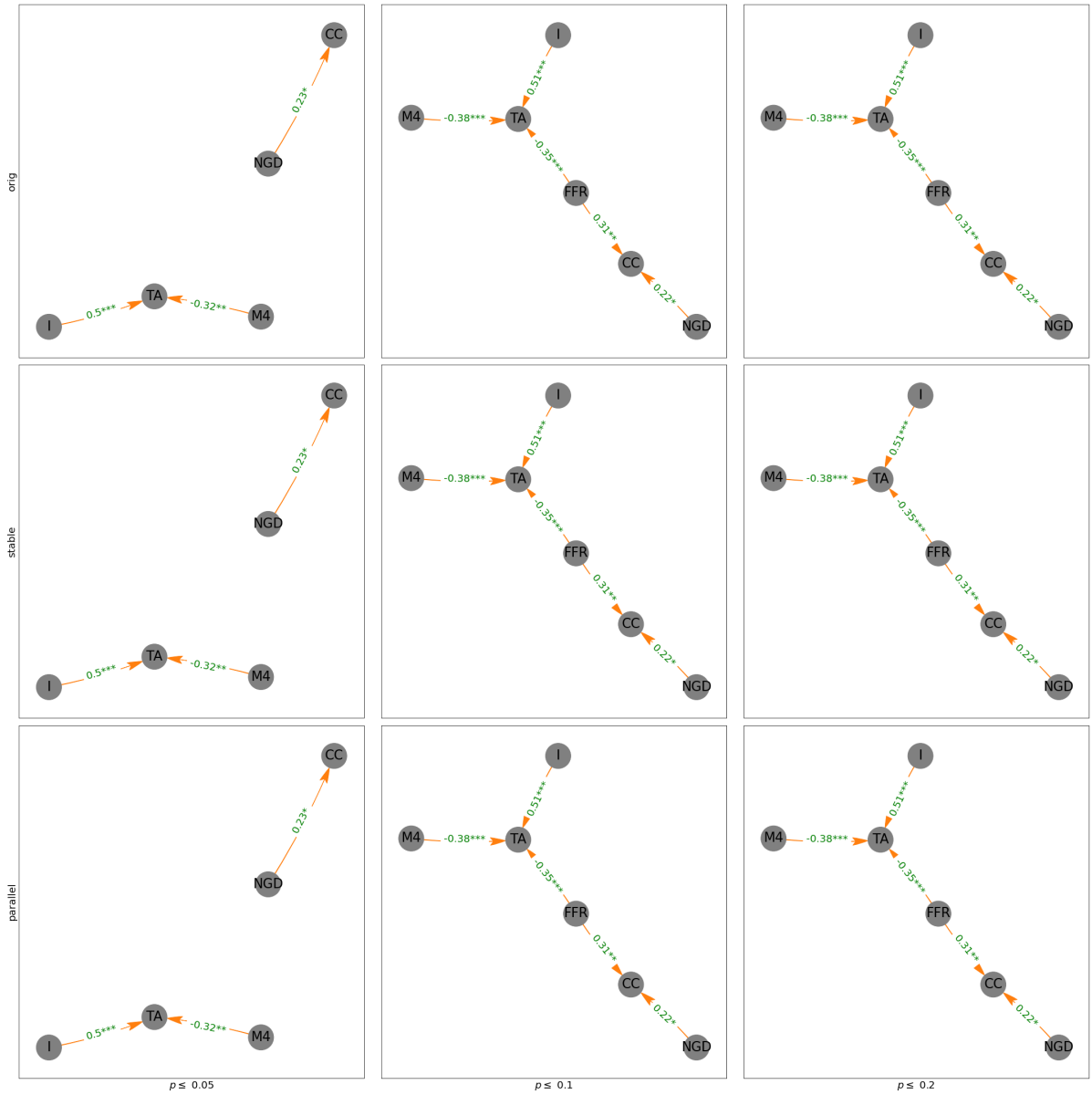
Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 8.81it/s]

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



Working for n conditional variables: 4: 100%		4/4 [00:00<00:00, 16.23it/s]
Working for n conditional variables: 4: 100%		4/4 [00:00<00:00, 18.13it/s]
Working for n conditional variables: 4: 100%		4/4 [00:00<00:00, 12.90it/s]
Working for n conditional variables: 4: 100%		4/4 [00:00<00:00, 11.92it/s]
Working for n conditional variables: 4: 100%		4/4 [00:00<00:00, 13.33it/s]
Working for n conditional variables: 4: 100%		4/4 [00:00<00:00, 8.57it/s]
Working for n conditional variables: 4: 100%		4/4 [00:00<00:00, 9.89it/s]
Working for n conditional variables: 4: 100%		4/4 [00:00<00:00, 9.36it/s]
Working for n conditional variables: 4: 100%		4/4 [00:00<00:00, 6.10it/s]

'TA', 'M4', 'FFR', 'NGD', 'CC', 'I'



Out[50]: OutEdgeView([('FFR', 'TA'), ('FFR', 'CC'), ('I', 'TA'), ('NGD', 'CC'), ('M4', 'TA')])

In [51]:
```
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:["NGDP", "TA", "M4", "I"],
                   1:["NGDP", "CC", "M4", "I"],
                   2:["TA", "CC", "FFR", "NGDP","I"],
                   3:["TA", "M4", "FFR", "NGDP","I"],
                   4:["TA", "M4", "FFR", "CC", "I"],
                   5:["TA", "M4", "FFR", "NGDP","CC", "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: 2: 100% 2/2 [00:00<00:00, 18.44it/s]

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

variables: 2: 100% 23.78it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 18.93it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 22.36it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 22.11it/s]

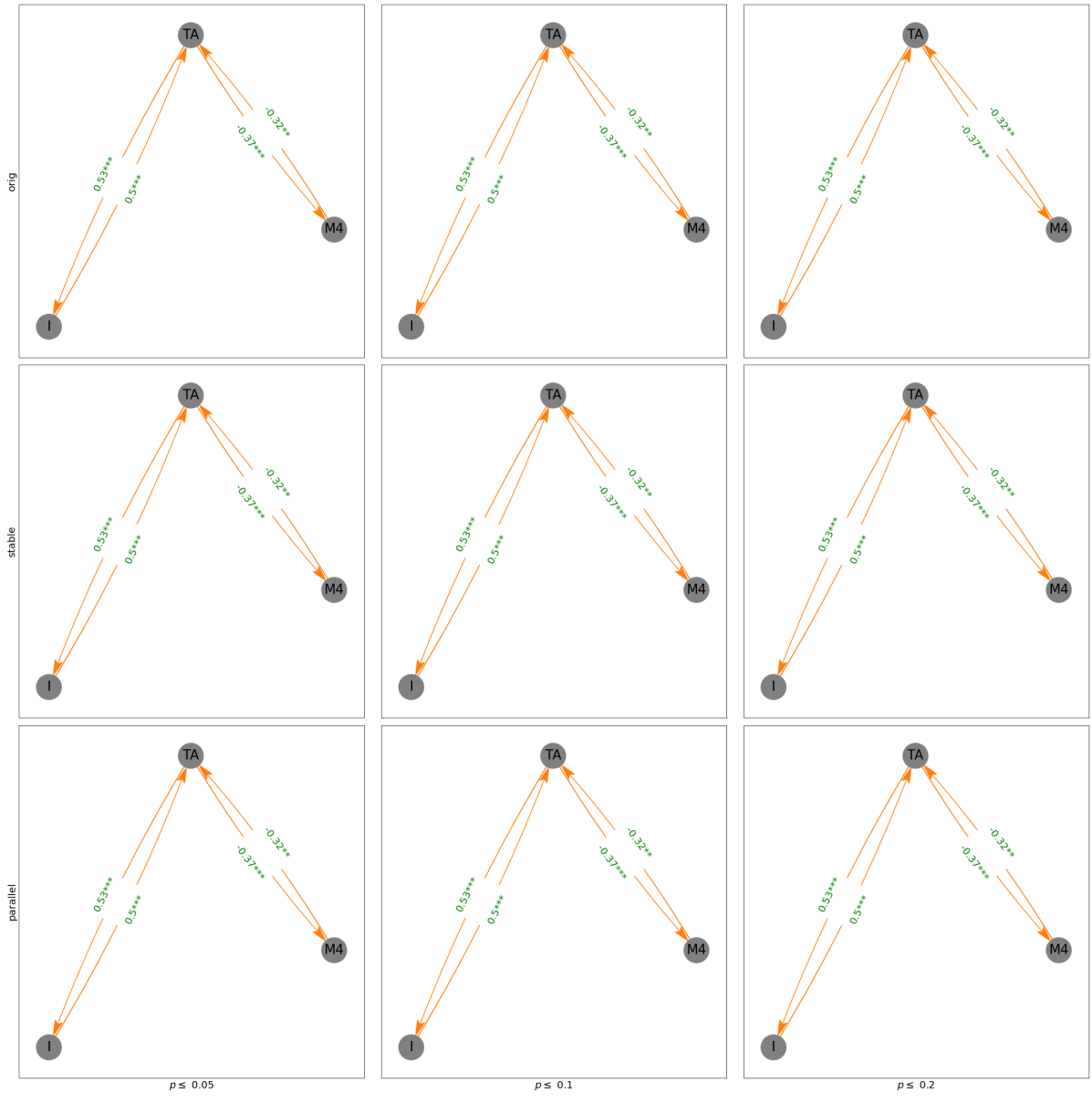
Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 16.85it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 25.16it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 23.59it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 14.92it/s]

'NGD', 'TA', 'M4', 'I'



Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 32.01it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 18.29it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 8.00it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 18.29it/s]

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

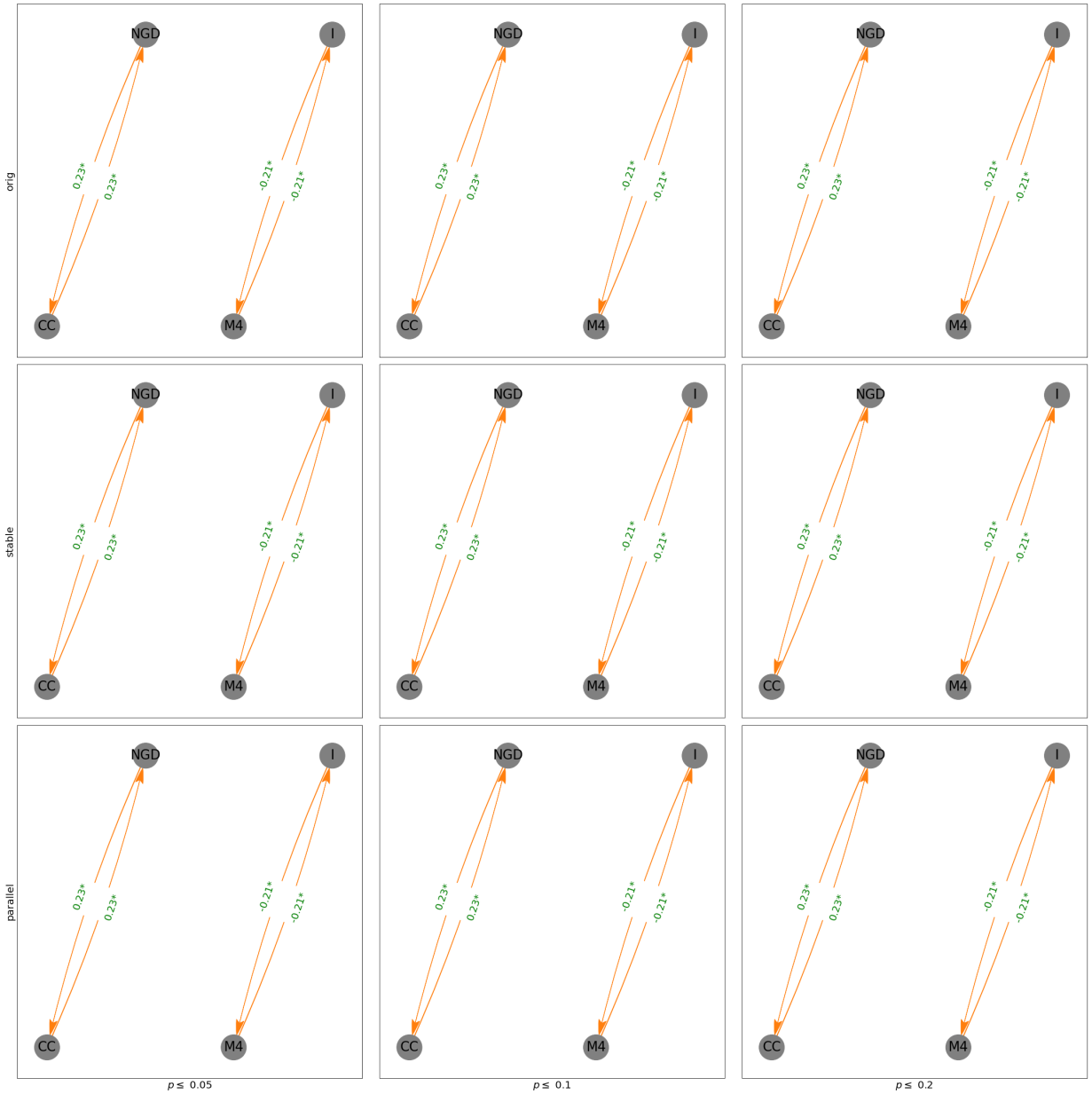
Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 9.15it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 7.11it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 9.15it/s]

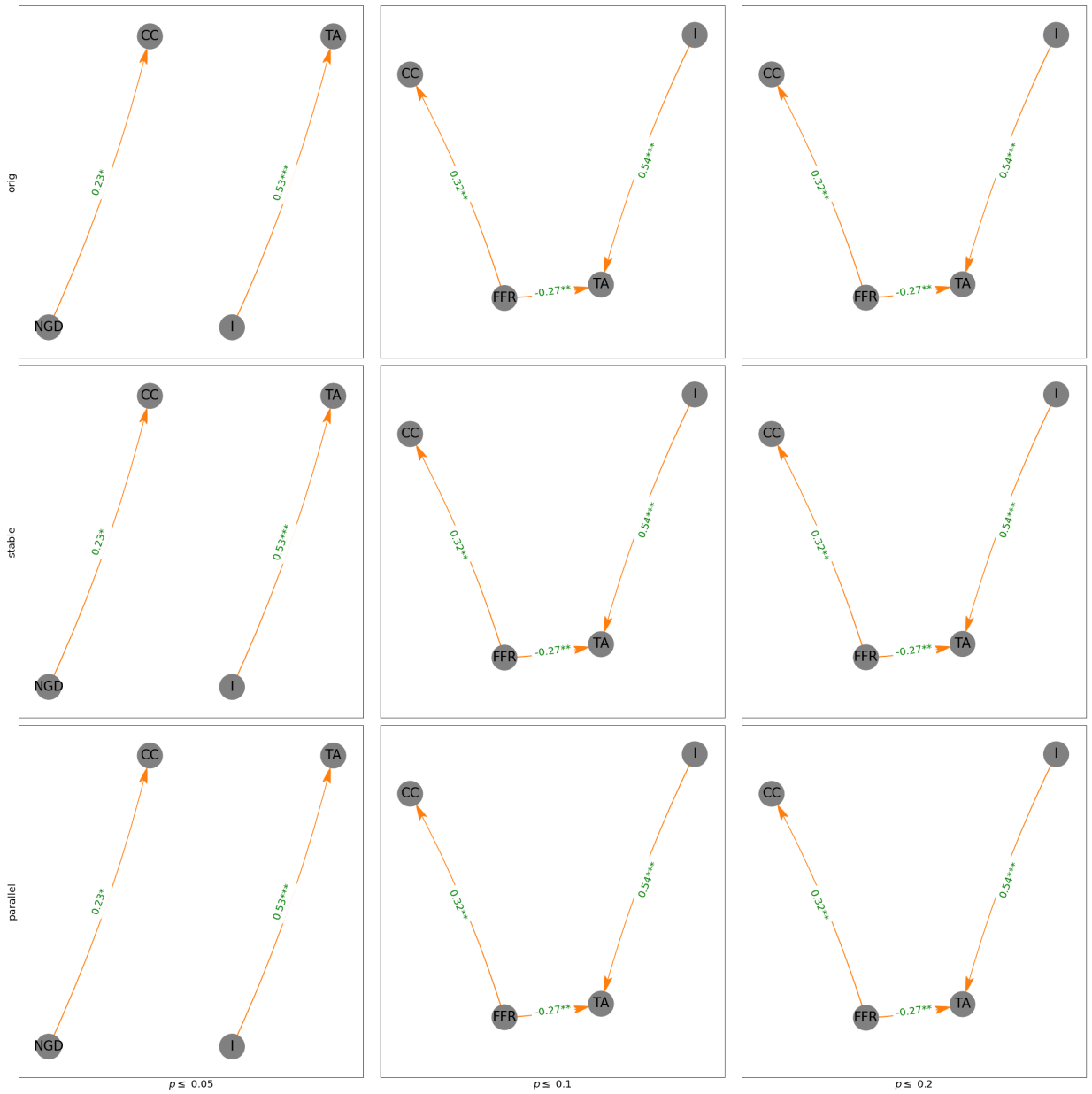
Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 9.14it/s]

'NGD', 'CC', 'M4', 'I'



Working for n conditional variables: 3: 100%		3/3 [00:00<00:00, 9.10it/s]
Working for n conditional variables: 3: 100%		3/3 [00:00<00:00, 4.19it/s]
Working for n conditional variables: 3: 100%		3/3 [00:00<00:00, 7.42it/s]
Working for n conditional variables: 3: 100%		3/3 [00:00<00:00, 5.41it/s]
Working for n conditional variables: 3: 100%		3/3 [00:00<00:00, 4.70it/s]
Working for n conditional variables: 3: 100%		3/3 [00:00<00:00, 3.81it/s]
Working for n conditional variables: 3: 100%		3/3 [00:00<00:00, 5.58it/s]
Working for n conditional variables: 3: 100%		3/3 [00:00<00:00, 3.14it/s]
Working for n conditional variables: 3: 100%		3/3 [00:00<00:00, 3.77it/s]

'TA', 'CC', 'FFR', 'NGD', 'I'



Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 7.67it/s]

Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 7.00it/s]

Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 5.96it/s]

Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 5.04it/s]

Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 5.57it/s]

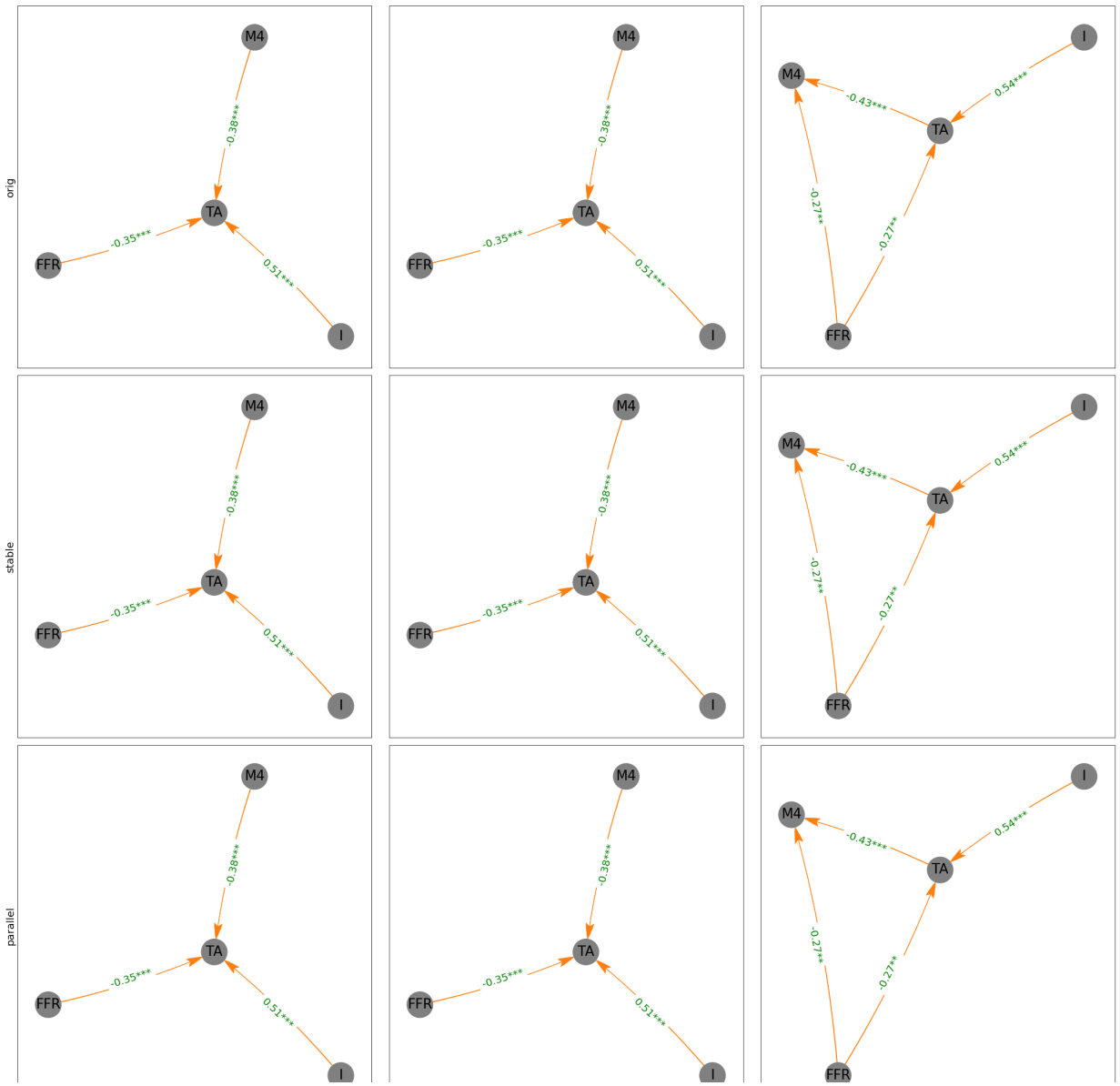
Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 4.29it/s]

Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 3.91it/s]

Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 4.12it/s]

Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 3.18it/s]

'TA', 'M4', 'FFR', 'NGD', 'I'



Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 6.58it/s]

Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 5.54it/s]

Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 5.33it/s]

Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 5.38it/s]

Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 5.25it/s]

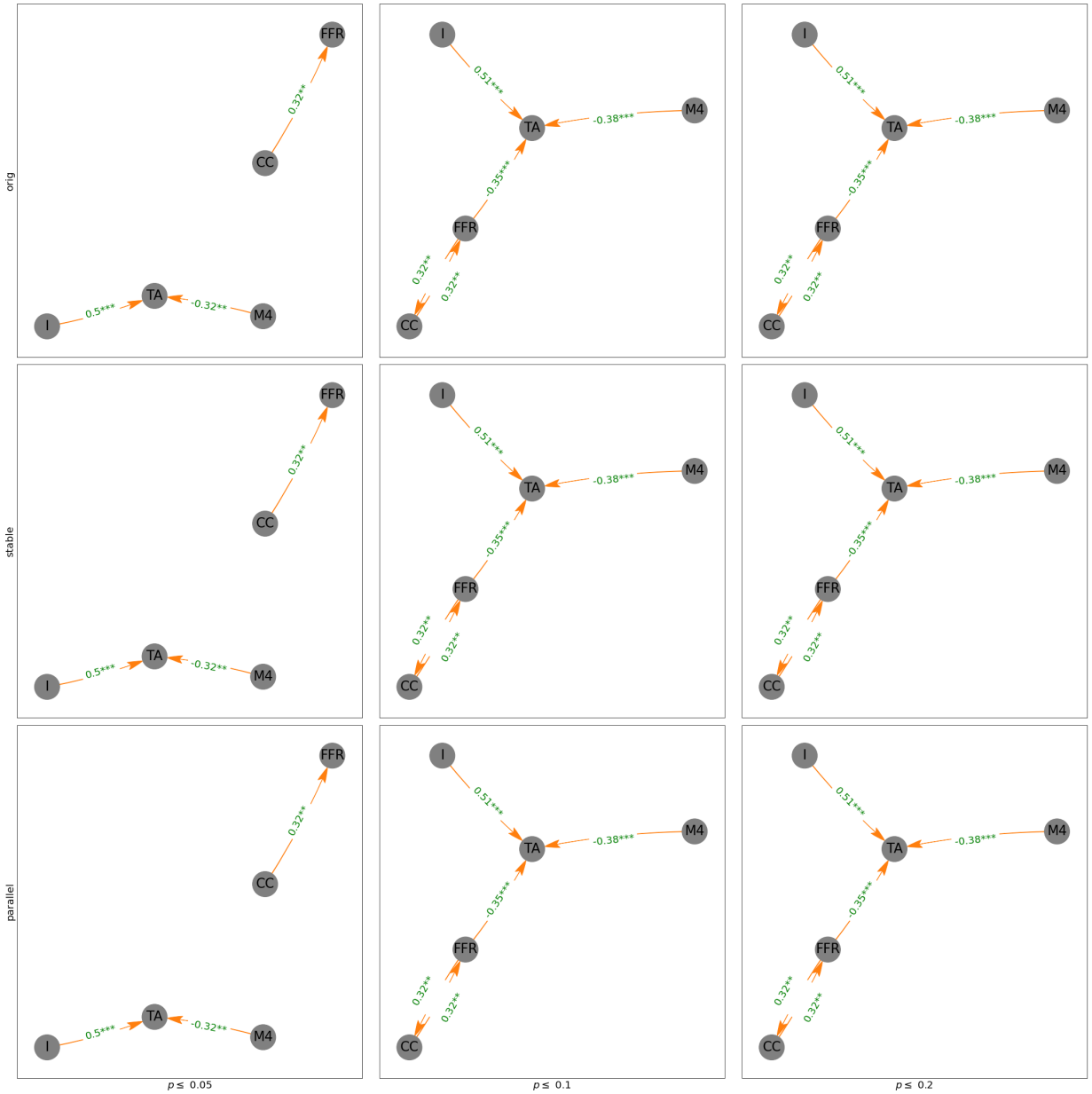
Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 4.10it/s]

Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 4.34it/s]

Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 4.07it/s]

Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 3.16it/s]

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



Working for n conditional variables: 4: 100% 4/4 [00:00<00:00, 5.45it/s]

Working for n conditional variables: 4: 100% 4/4 [00:00<00:00, 5.57it/s]

Working for n conditional variables: 4: 100% 4/4 [00:00<00:00, 5.73it/s]

Working for n conditional variables: 4: 100% 4/4 [00:00<00:00, 4.15it/s]

Working for n conditional variables: 4: 100% 4/4 [00:00<00:00, 3.47it/s]

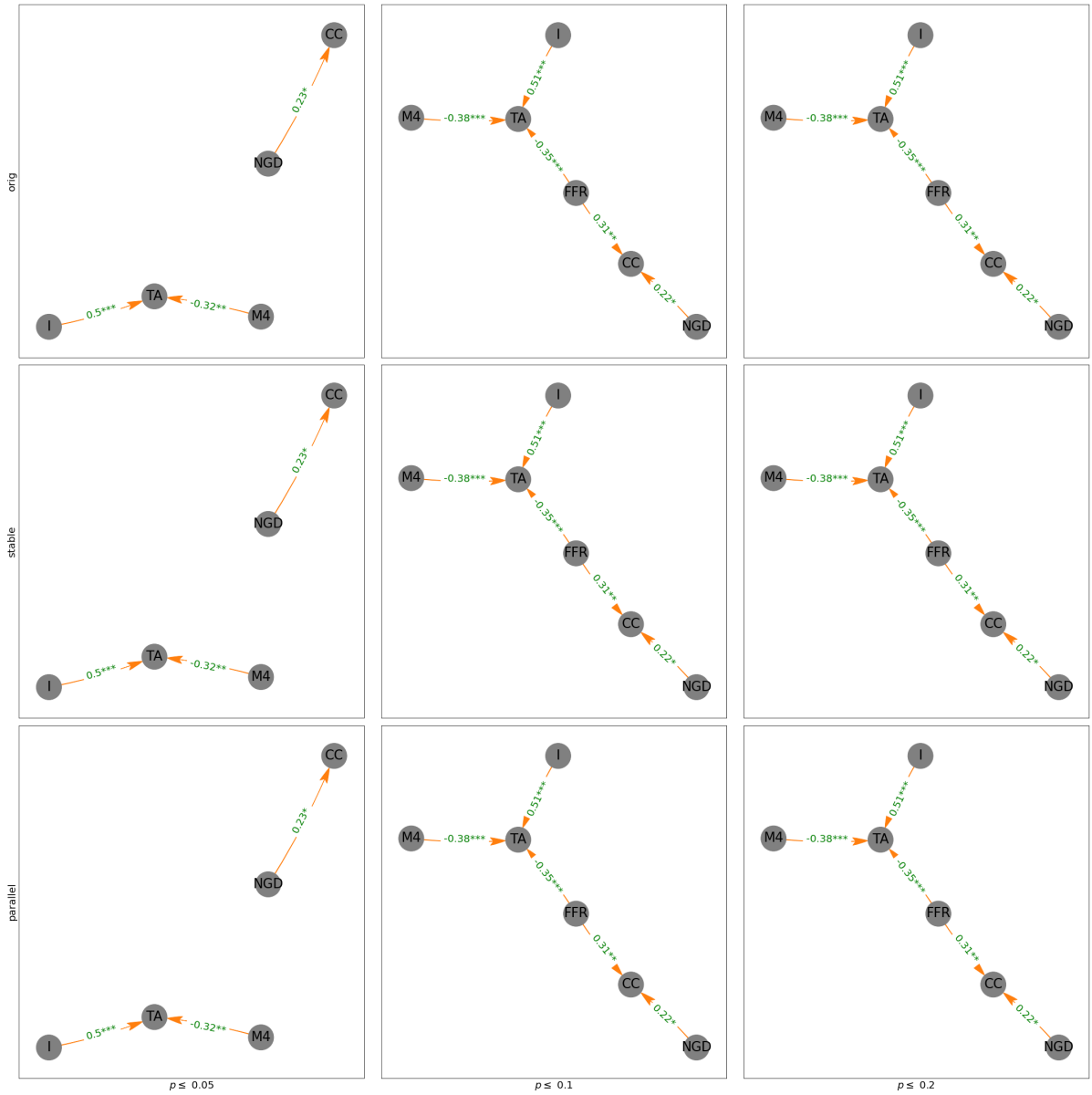
Working for n conditional variables: 4: 100% 4/4 [00:00<00:00, 5.94it/s]

Working for n conditional variables: 4: 100% 4/4 [00:00<00:00, 3.38it/s]

Working for n conditional variables: 4: 100% 4/4 [00:00<00:00, 3.23it/s]

Working for n conditional variables: 4: 100% 4/4 [00:01<00:00, 3.26it/s]

'TA', 'M4', 'FFR', 'NGD', 'CC', 'I'



Out[51]: OutEdgeView([('FFR', 'TA'), ('FFR', 'CC'), ('I', 'TA'), ('NGD', 'CC'), ('M4', 'TA')])

TWICE DIFF MONTHLY DATA WITH M4 TA FFR CC NGDP I (WITH INFLATION)

In [50]:
```
#data cleaning, importing

d_parser = lambda x: pd.datetime.strptime(x, '%m/%d/%Y')
df = pd.read_csv('M4-11.csv', parse_dates=['Date'], date_parser=d_parser)
df
```

Out[50]:

	Date	Log M4 including Treasuries	Log Total Assets	Effective Federal Funds Rate (%)	Log Currency in Circulation ($ Bil)	Log Nominal GDP (Bln)	Unemployment Rate	Inflation Rate
0	2010-01-31	7.07	14.63	0.11	6.83	16.51	9.8	2.62
1	2010-02-28	7.06	14.63	0.13	6.83	16.70	9.8	2.15
2	2010-03-31	7.05	14.65	0.17	6.84	16.70	9.9	2.29
3	2010-04-30	7.06	14.66	0.20	6.84	16.52	9.9	2.21
4	2010-05-31	7.06	14.66	0.20	6.84	16.70	9.6	2.00
...	...	...	...	...	...	...	...	...
115	2019-08-31	7.40	15.14	2.13	7.47	16.70	3.7	1.74
116	2019-09-30	7.40	15.15	2.04	7.47	16.70	3.5	1.72
117	2019-10-31	7.41	15.19	1.83	7.48	16.89	3.6	1.77
118	2019-11-30	7.42	15.21	1.55	7.49	16.70	3.6	2.04
119	2019-12-31	7.42	15.23	1.55	7.49	16.70	3.6	2.26

120 rows × 8 columns

In [51]:
```
df['Date_at_year_month'] = df['Date'].dt.strftime('%Y-%m')
column_names = {'Date_at_year_month':'DATE',
                'Log M4 including Treasuries':'M4',
                'Log Total Assets': 'TA',
                'Log Currency in Circulation ($ Bil)':'CC',
                'Log Nominal GDP (Bln)':'NGDP',
               'Effective Federal Funds Rate (%)': 'FFR',
               'Unemployment Rate': 'U',
               'Inflation Rate': 'I'}


# rename columns
df = df.rename(columns = column_names)
df = df.set_index('DATE')
df = df.drop(['Date'], axis = 1)
df = df.drop(['U'], axis = 1)
df
```

Out[51]:

DATE	M4	TA	FFR	CC	NGDP	I
2010-01	7.07	14.63	0.11	6.83	16.51	2.62
2010-02	7.06	14.63	0.13	6.83	16.70	2.15
2010-03	7.05	14.65	0.17	6.84	16.70	2.29
2010-04	7.06	14.66	0.20	6.84	16.52	2.21
2010-05	7.06	14.66	0.20	6.84	16.70	2.00
...	...	...	...	...	...	...
2019-08	7.40	15.14	2.13	7.47	16.70	1.74
2019-09	7.40	15.15	2.04	7.47	16.70	1.72
2019-10	7.41	15.19	1.83	7.48	16.89	1.77
2019-11	7.42	15.21	1.55	7.49	16.70	2.04
2019-12	7.42	15.23	1.55	7.49	16.70	2.26

120 rows × 6 columns

In [52]:
```
df_new = df.diff().dropna()
```

In [53]:
```
df = df_new.diff().dropna()
```

In [54]:
```
#ADF test

X = df["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 = df["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 = df["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 = df["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 = df["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")

X = df["NGDP"].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")
```

```
ADF Statistic: -8.381034
p-value: 0.000000
Critical Values:
        1%: -3.491
        5%: -2.888
        10%: -2.581
Reject Ho - Time Series is Stationary
ADF Statistic: -3.943879
p-value: 0.001735
Critical Values:
        1%: -3.492
        5%: -2.889
        10%: -2.581
Reject Ho - Time Series is Stationary
ADF Statistic: -6.016185
p-value: 0.000000
Critical Values:
        1%: -3.490
        5%: -2.887
        10%: -2.581
Reject Ho - Time Series is Stationary
ADF Statistic: -6.497273
p-value: 0.000000
Critical Values:
        1%: -3.494
        5%: -2.889
        10%: -2.582
Reject Ho - Time Series is Stationary
ADF Statistic: -4.617794
```

```
p-value: 0.000120
Critical Values:
        1%: -3.494
        5%: -2.889
        10%: -2.582
Reject Ho - Time Series is Stationary
ADF Statistic: -9.041977
p-value: 0.000000
Critical Values:
        1%: -3.492
        5%: -2.889
        10%: -2.581
Reject Ho - Time Series is Stationary
```

In [9]:
```
## 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
```

Out[9]:

DATE	M4	TA	FFR	CC	NGDP	I
2010-03	0.001410	0.021994	0.064050	0.008358	-0.243274	0.623866
2010-04	0.015875	-0.005155	0.043964	-0.006113	-0.118411	-0.310845
2010-05	-0.008498	-0.016395	-0.118352	-0.000243	0.398229	-0.081490
2010-06	-0.008602	0.009326	0.025208	0.004889	-0.201007	-0.627510
2010-07	0.014294	-0.013658	0.049001	-0.002497	-0.097702	0.982006
...	...	...	...	...	...	...
2019-08	-0.002816	0.003813	-0.207788	-0.014695	-0.051197	-0.131491
2019-09	-0.002179	0.018272	0.136477	-0.000276	-0.071138	0.024437
2019-10	0.012024	0.025741	-0.130945	0.007484	0.267350	0.180955
2019-11	-0.006882	-0.015442	-0.001870	-0.000154	-0.253409	0.190669
2019-12	-0.002433	0.000198	0.222038	-0.006496	-0.123925	-0.182035

118 rows × 6 columns

In [10]:
```
residuals.corr()[residuals.corr().abs() < 1].mul(-1).fillna(1).round(2)
```

Out[10]:

	M4	TA	FFR	CC	NGDP	I
M4	1.00	-0.08	-0.19	-0.19	-0.12	0.05
TA	-0.08	1.00	-0.06	0.12	0.18	-0.04
FFR	-0.19	-0.06	1.00	-0.11	0.46	0.09
CC	-0.19	0.12	-0.11	1.00	-0.05	-0.06
NGDP	-0.12	0.18	0.46	-0.05	1.00	-0.03
I	0.05	-0.04	0.09	-0.06	-0.03	1.00

In [11]:
```
import pingouin
df.pcorr().round(2)
```

Out[11]:

	M4	TA	FFR	CC	NGDP	I
M4	1.00	-0.08	-0.19	-0.19	-0.12	0.05
TA	-0.08	1.00	-0.06	0.12	0.18	-0.04
FFR	-0.19	-0.06	1.00	-0.11	0.46	0.09
CC	-0.19	0.12	-0.11	1.00	-0.05	-0.06
NGDP	-0.12	0.18	0.46	-0.05	1.00	-0.03
I	0.05	-0.04	0.09	-0.06	-0.03	1.00

In [12]:
```
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[12]:

	M4	TA	FFR	CC	NGDP	I
M4	NaN	0.40	0.04	0.03	0.19	0.57
TA	0.40	NaN	0.53	0.21	0.04	0.70
FFR	0.04	0.53	NaN	0.23	0.00	0.35
CC	0.03	0.21	0.23	NaN	0.56	0.55
NGDP	0.19	0.04	0.00	0.56	NaN	0.72
I	0.57	0.70	0.35	0.55	0.72	NaN

In [13]:
```
residuals
```

Out[13]:

DATE	M4	TA	FFR	CC	NGDP	I
2010-03	0.001410	0.021994	0.064050	0.008358	-0.243274	0.623866
2010-04	0.015875	-0.005155	0.043964	-0.006113	-0.118411	-0.310845
2010-05	-0.008498	-0.016395	-0.118352	-0.000243	0.398229	-0.081490
2010-06	-0.008602	0.009326	0.025208	0.004889	-0.201007	-0.627510
2010-07	0.014294	-0.013658	0.049001	-0.002497	-0.097702	0.982006
...	...	...	...	...	...	...
2019-08	-0.002816	0.003813	-0.207788	-0.014695	-0.051197	-0.131491
2019-09	-0.002179	0.018272	0.136477	-0.000276	-0.071138	0.024437
2019-10	0.012024	0.025741	-0.130945	0.007484	0.267350	0.180955
2019-11	-0.006882	-0.015442	-0.001870	-0.000154	-0.253409	0.190669
2019-12	-0.002433	0.000198	0.222038	-0.006496	-0.123925	-0.182035

118 rows × 6 columns

In [14]:
```
##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
```

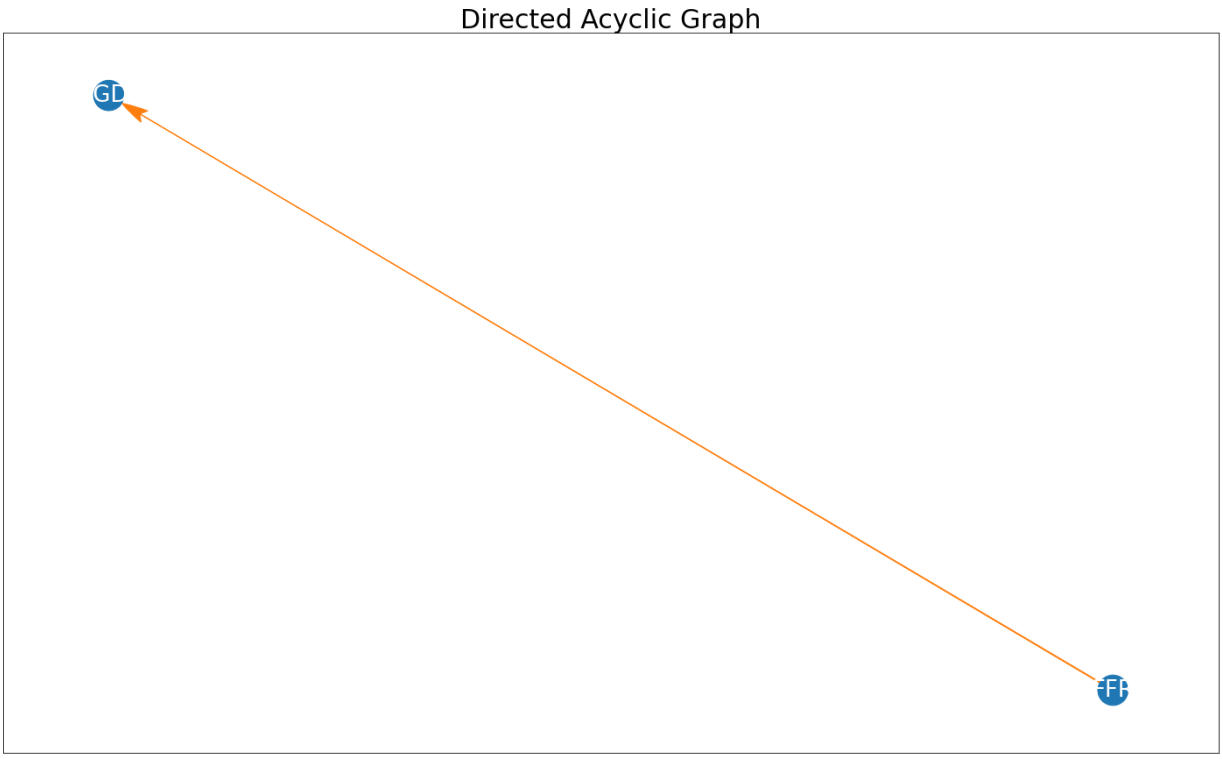
In [15]:
```
## Estimating a Directed Acyclic Graph
p_val = .01
from pgmpy.estimators import PC
c = PC(df)
max_cond_vars = len(df.keys())-2



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: 50% 7.62it/s]

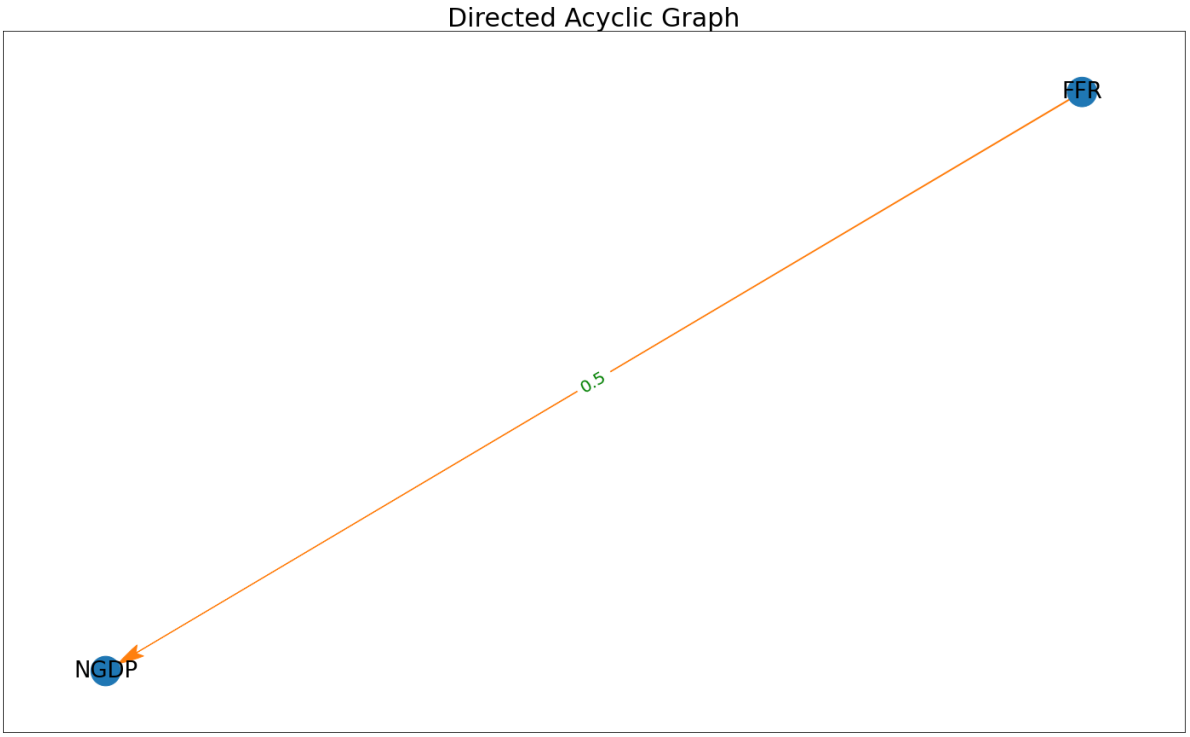
In [16]:
```
from matplotlib.patches import ArrowStyle

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[16]:
```
OutEdgeView([('FFR', 'NGDP')])
```



In [17]:
```
## 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', 'NGDP') []
```



In [18]:
```
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 [19]:
```
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:["NGDP", "TA", "M4", "I"],
                   1:["NGDP", "CC", "M4", "I"],
                   2:["NGDP", "FFR", "M4", "I"],
                   3:["TA", "M4", "FFR", "NGDP","I"],
                   4:["TA", "M4", "FFR", "CC", "I"],
                   5:["TA", "M4", "FFR", "NGDP","CC", "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:                    2/2 [00:00<00:00,
2: 100%                                                 42.67it/s]
```

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 36.32it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 26.73it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 30.91it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 27.13it/s]

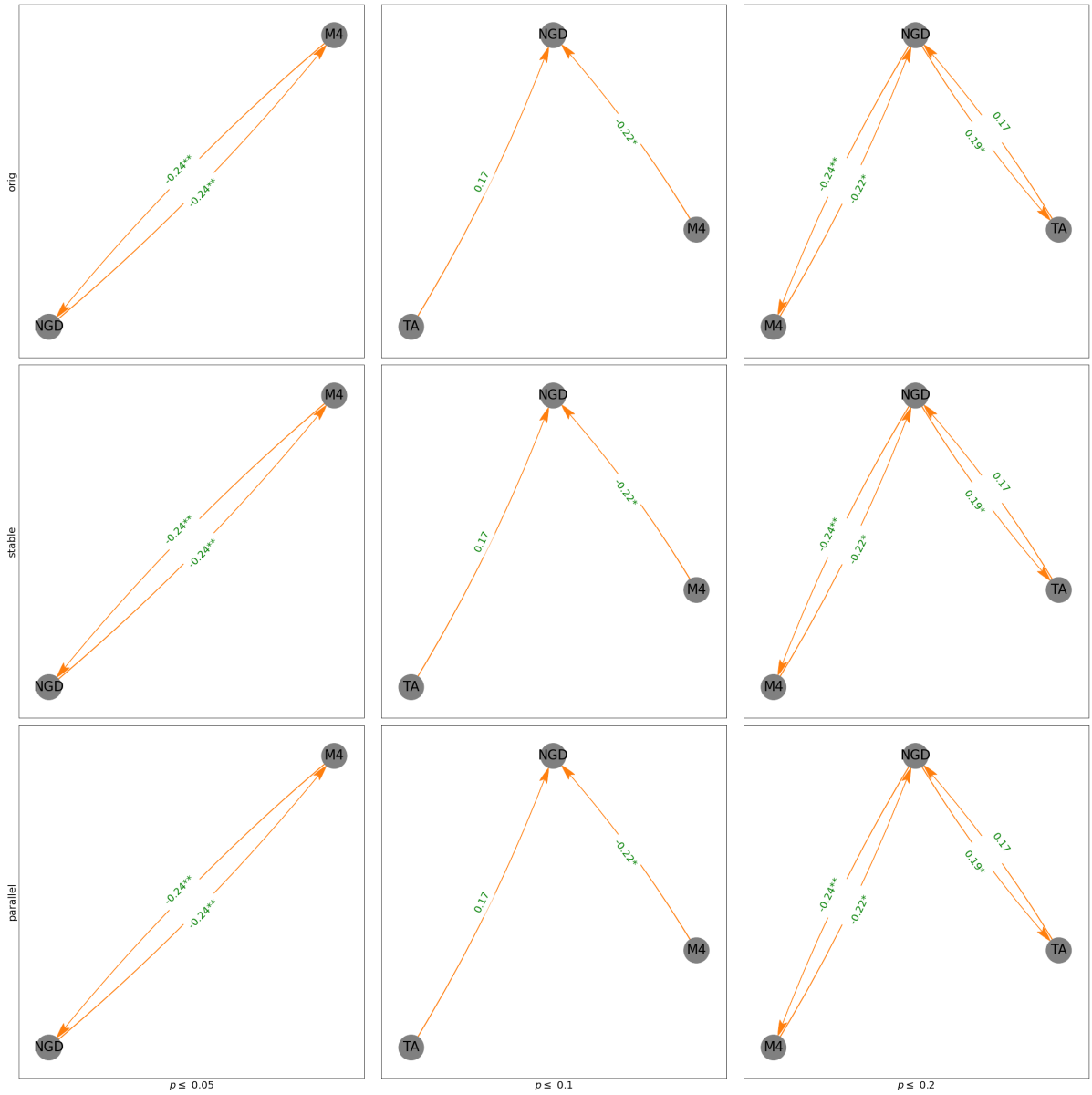
Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 23.98it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 27.29it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 19.33it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 19.56it/s]

'NGD', 'TA', 'M4', 'I'



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, 64.01it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 33.53it/s]

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

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

2: 100% 30.49it/s]

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

2: 100% 22.65it/s]

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

2: 100% 31.53it/s]

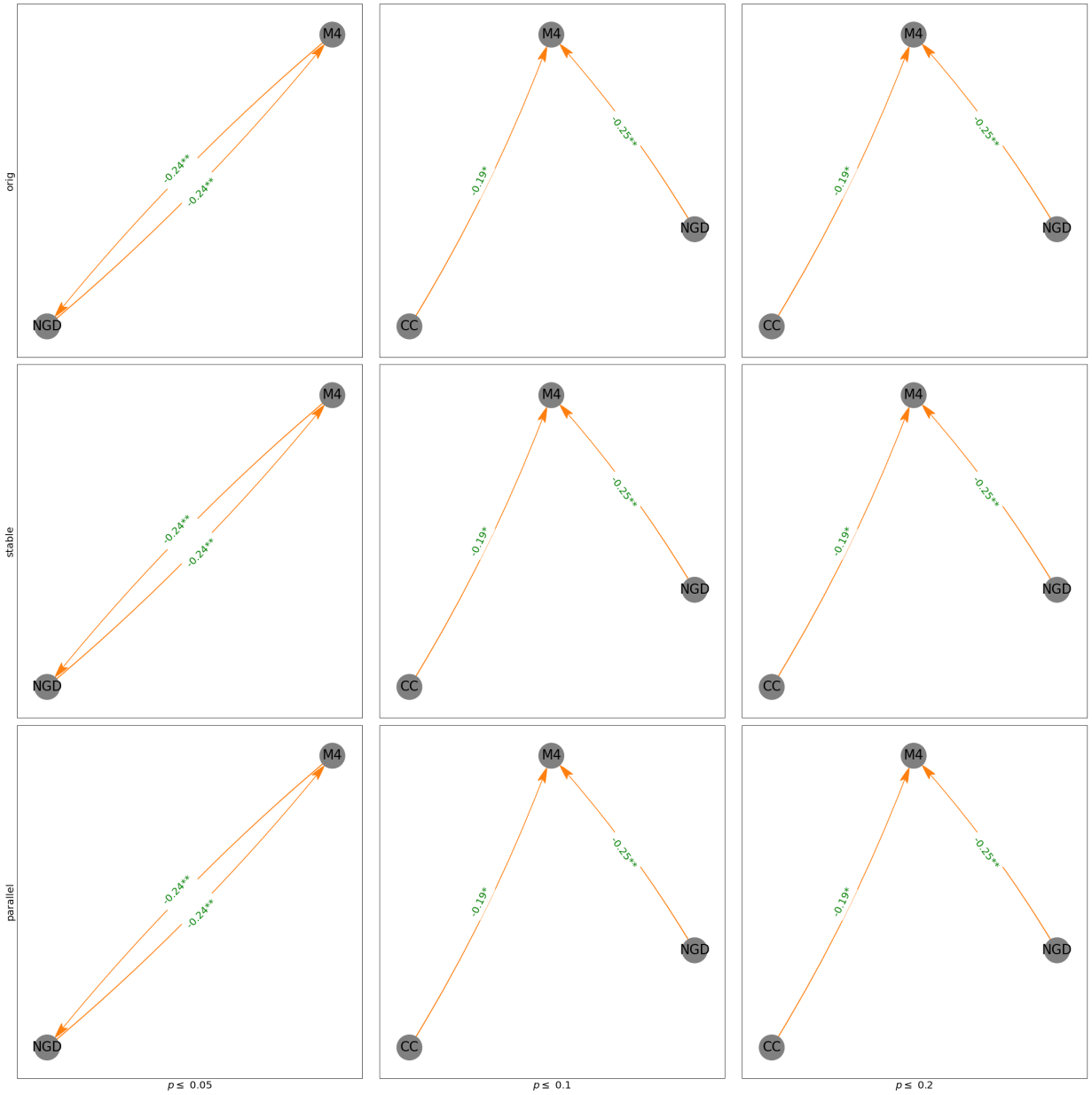
Working for n conditional variables: 2/2 [00:00<00:00,

2: 100% 38.78it/s]

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

2: 100% 23.89it/s]

'NGD', 'CC', 'M4', 'I'



Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 29.59it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 25.85it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 21.84it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 25.52it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 23.91it/s]

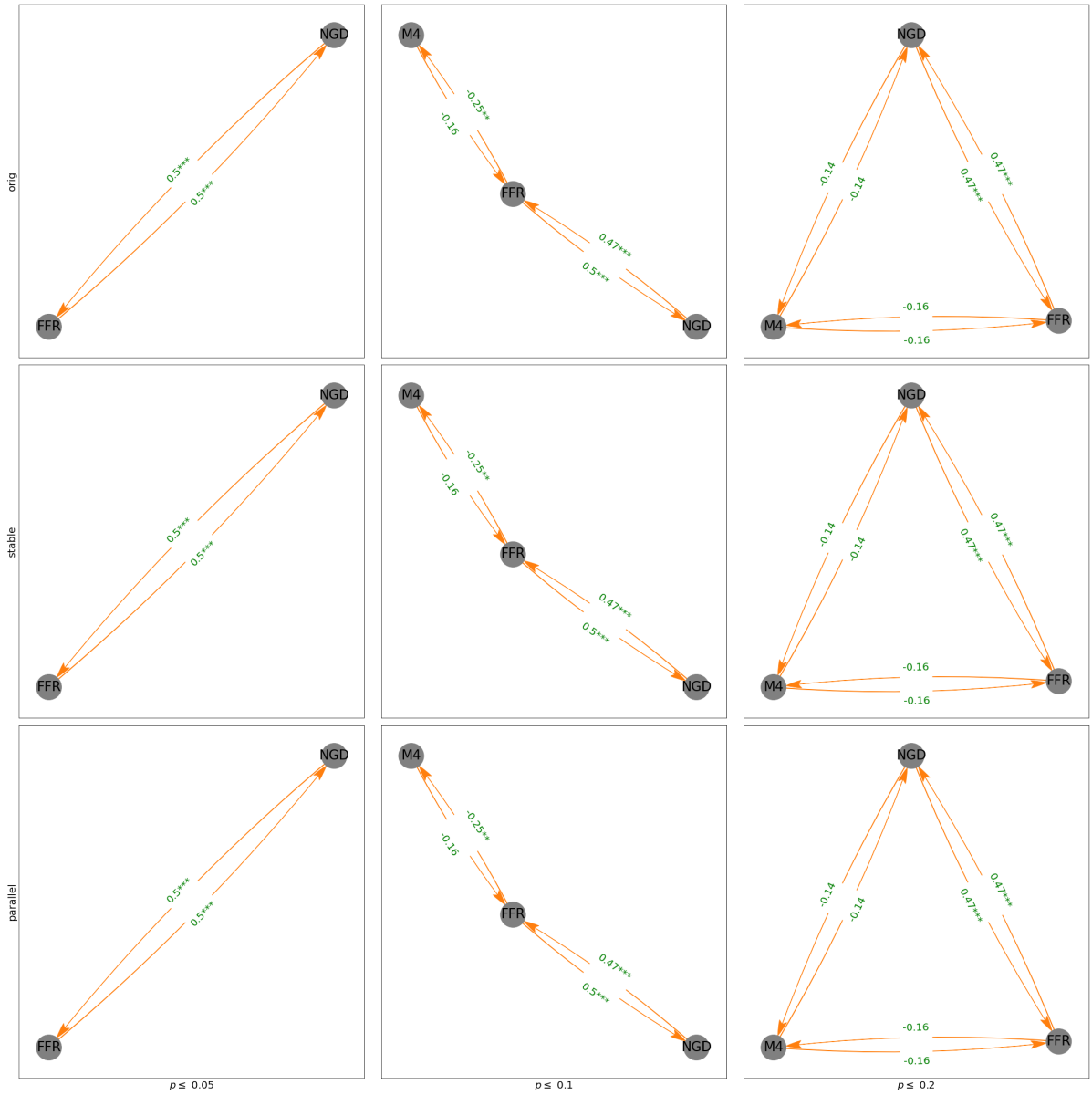
Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 19.90it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 23.88it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 22.41it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 5.85it/s]

'NGD', 'FFR', 'M4', 'I'


orig
stable
parallel
NGD
FFR
M4
0.5***
-0.25**
-0.16
0.47***
-0.14
-0.16
p ≤ 0.05
p ≤ 0.1
p ≤ 0.2

Working for n conditional variables: 2: 67% 2/3 [00:00<00:00, 30.20it/s]

Working for n conditional variables: 2: 67% 2/3 [00:00<00:00, 26.43it/s]

Working for n conditional variables: 2: 67% 2/3 [00:00<00:00, 19.58it/s]

Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 34.35it/s]

Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 32.45it/s]

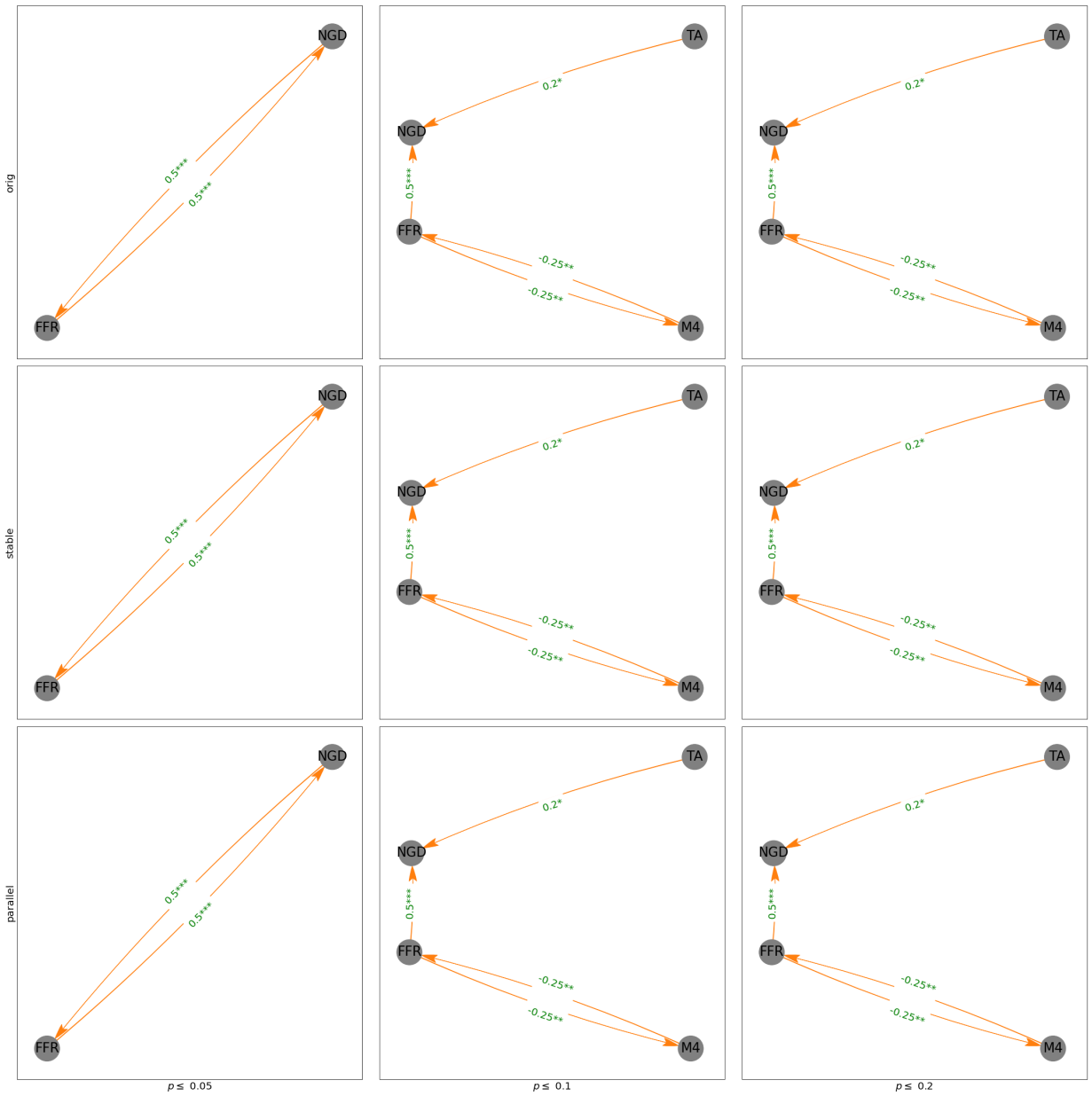
Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 32.00it/s]

Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 27.43it/s]

Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 32.00it/s]

Working for n conditional variables: 3: 100% 3/3 [00:00<00:00, 6.36it/s]

'TA', 'M4', 'FFR', 'NGD', 'I'



Working for n conditional variables: 2/3 [00:00<00:00, 64.00it/s]

Working for n conditional variables: 2/3 [00:00<00:00, 37.16it/s]

Working for n conditional variables: 2/3 [00:00<00:00, 28.35it/s]

Working for n conditional variables: 3/3 [00:00<00:00, 57.12it/s]

Working for n conditional variables: 3/3 [00:00<00:00,

3: 100% 44.13it/s]

Working for n conditional variables: 3/3 [00:00<00:00,

3: 100% 29.76it/s]

Working for n conditional variables: 3/3 [00:00<00:00,

3: 100% 39.75it/s]

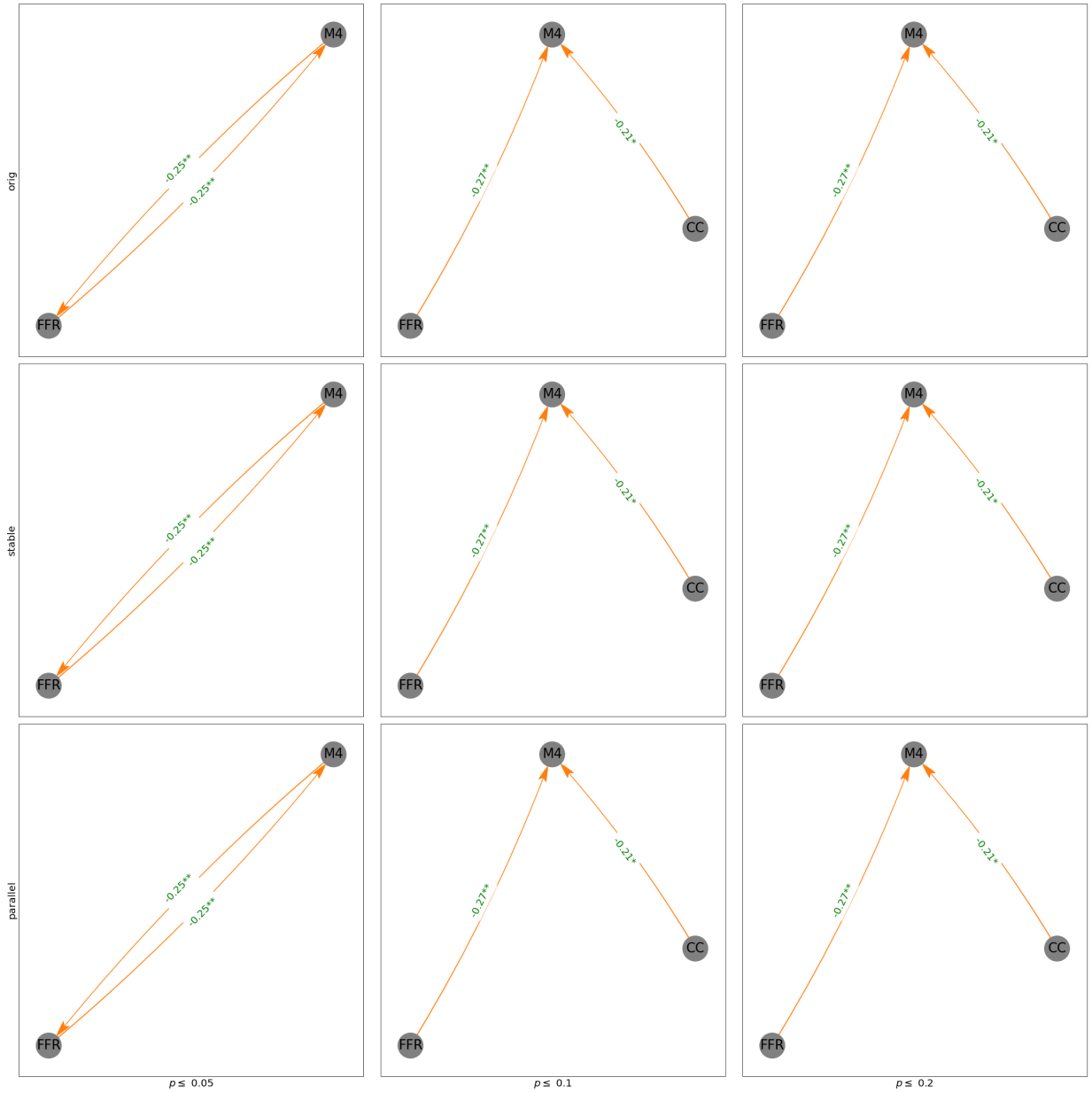
Working for n conditional variables: 3/3 [00:00<00:00,

3: 100% 41.10it/s]

Working for n conditional variables: 3/3 [00:00<00:00,

3: 100% 17.91it/s]

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



Working for n conditional variables: 2: 50% 2/4 [00:00<00:00, 25.70it/s]

Working for n conditional variables: 2: 50% 2/4 [00:00<00:00, 42.66it/s]

Working for n conditional variables: 2: 50% 2/4 [00:00<00:00, 19.62it/s]

Working for n conditional variables: 3: 75% 3/4 [00:00<00:00, 35.47it/s]

Working for n conditional variables: 3: 75% 3/4 [00:00<00:00, 15.94it/s]

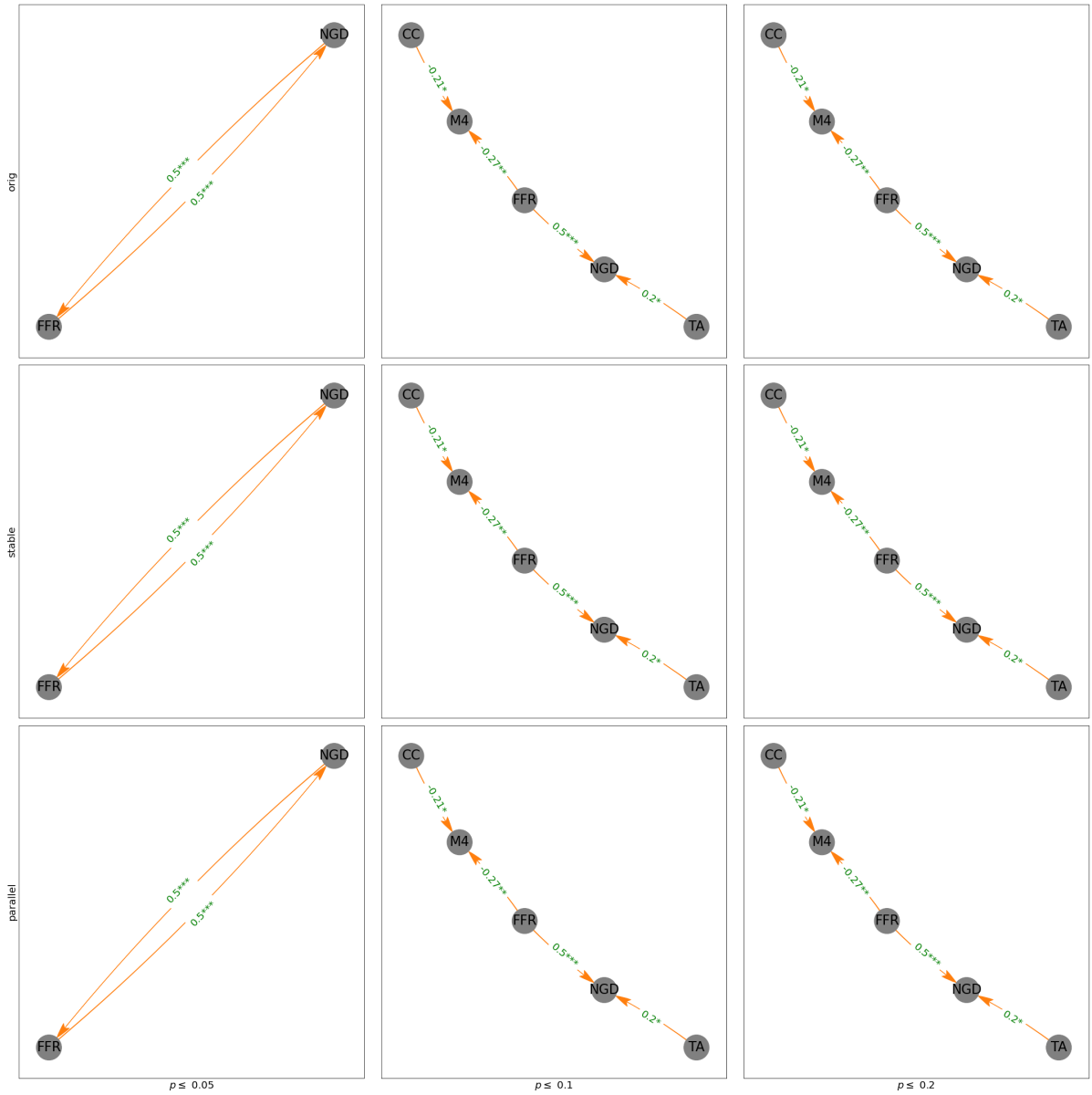
Working for n conditional variables: 3: 75% 3/4 [00:00<00:00, 13.74it/s]

Working for n conditional variables: 3: 75% 3/4 [00:00<00:00, 14.90it/s]

Working for n conditional variables: 3: 75% 3/4 [00:00<00:00, 15.48it/s]

Working for n conditional variables: 3: 75% 3/4 [00:00<00:00, 9.41it/s]

'TA', 'M4', 'FFR', 'NGD', 'CC', 'I'



Out[19]: OutEdgeView([('FFR', 'M4'), ('FFR', 'NGD'), ('CC', 'M4'), ('TA', 'NGD')])

In [20]:
```
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:["NGDP", "TA", "M4", "I"],
                   1:["NGDP", "CC", "M4", "I"],
                   2:["NGDP", "FFR", "M4", "I"],
                   3:["TA", "M4", "FFR", "NGDP","I"],
                   4:["TA", "NGDP", "FFR", "CC", "I"],
                   5:["TA", "M4", "FFR", "NGDP","CC", "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:                    2/2 [00:00<00:00,
2: 100%                                                 34.65it/s]

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

2: 100% 64.05it/s]

Working for n conditional variables: 2/2 [00:00<00:00,
2: 100% 25.02it/s]

Working for n conditional variables: 2/2 [00:00<00:00,
2: 100% 32.01it/s]

Working for n conditional variables: 2/2 [00:00<00:00,
2: 100% 33.30it/s]

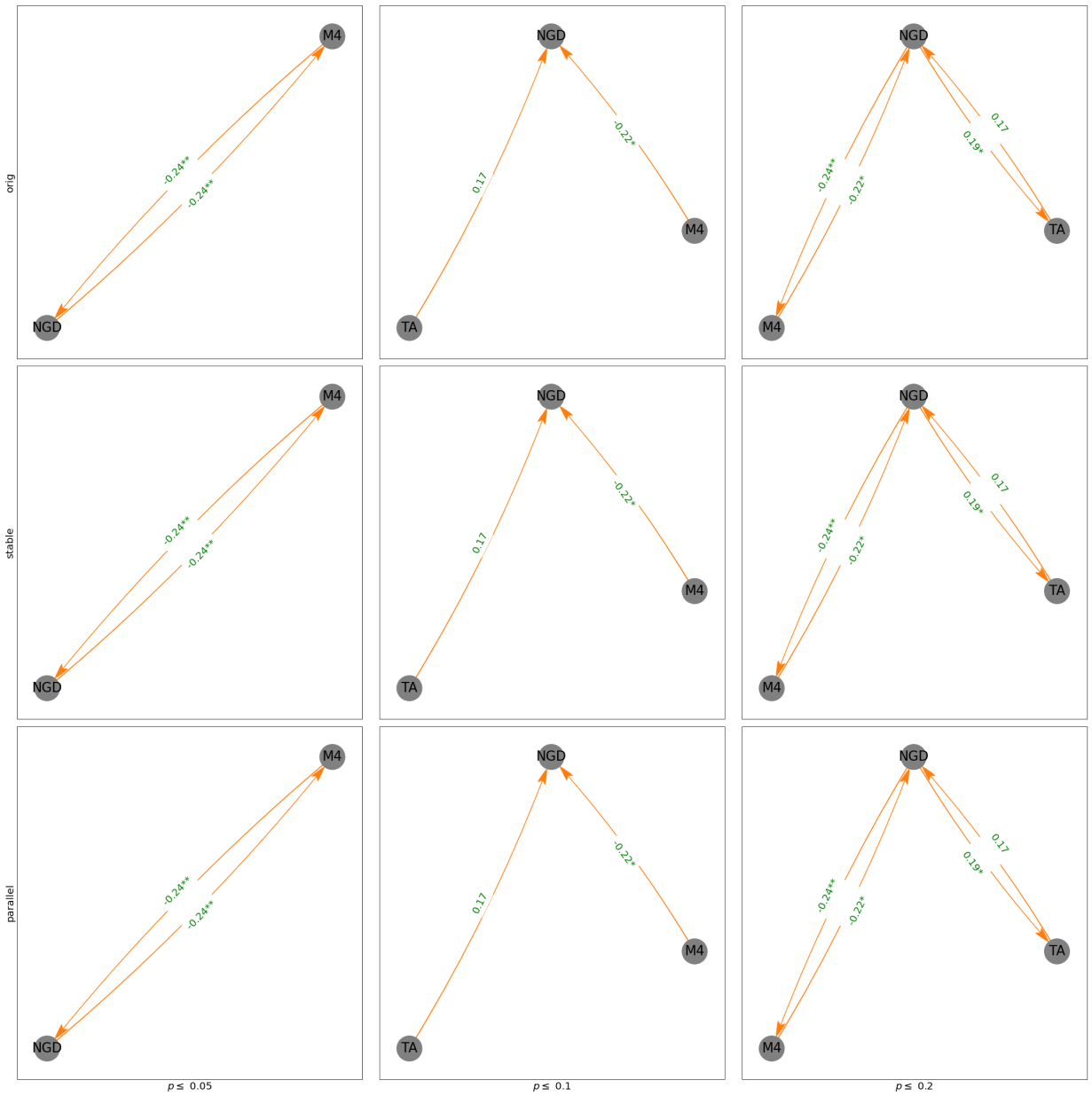
Working for n conditional variables: 2/2 [00:00<00:00,
2: 100% 26.21it/s]

Working for n conditional variables: 2/2 [00:00<00:00,
2: 100% 20.30it/s]

Working for n conditional variables: 2/2 [00:00<00:00,
2: 100% 17.70it/s]

Working for n conditional variables: 2/2 [00:00<00:00,
2: 100% 16.78it/s]

'NGD', 'TA', 'M4', 'I'



Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 38.22it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 64.01it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 23.00it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 24.25it/s]

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

2: 100% 25.52it/s]

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

2: 100% 22.91it/s]

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

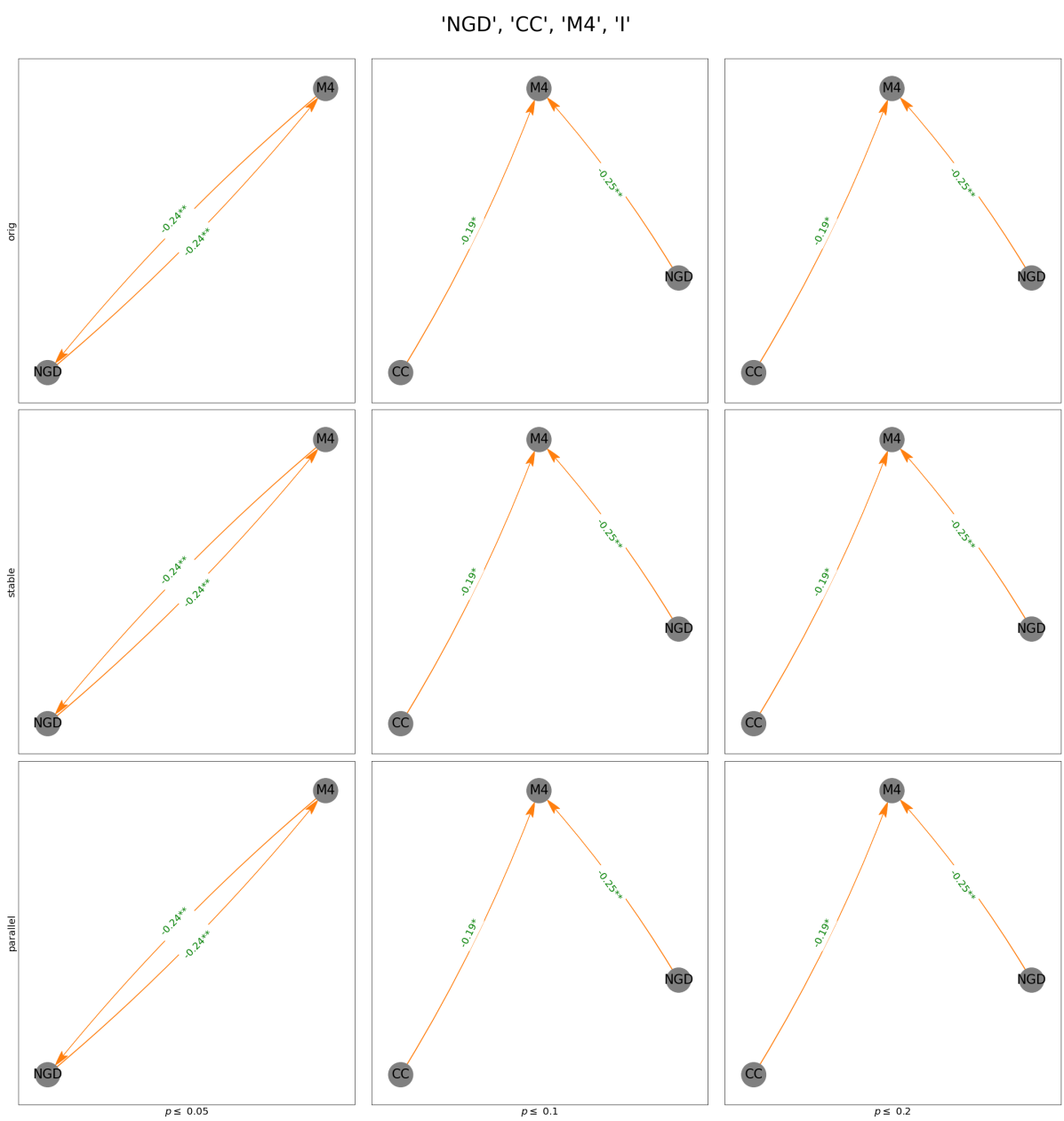
2: 100% 42.68it/s]

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

2: 100% 33.48it/s]

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

2: 100% 22.93it/s]



Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 24.73it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 25.26it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 19.04it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 23.71it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 31.76it/s]

Working for n conditional variables: 2: 100% 2/2 [00:00<00:00, 19.62it/s]

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

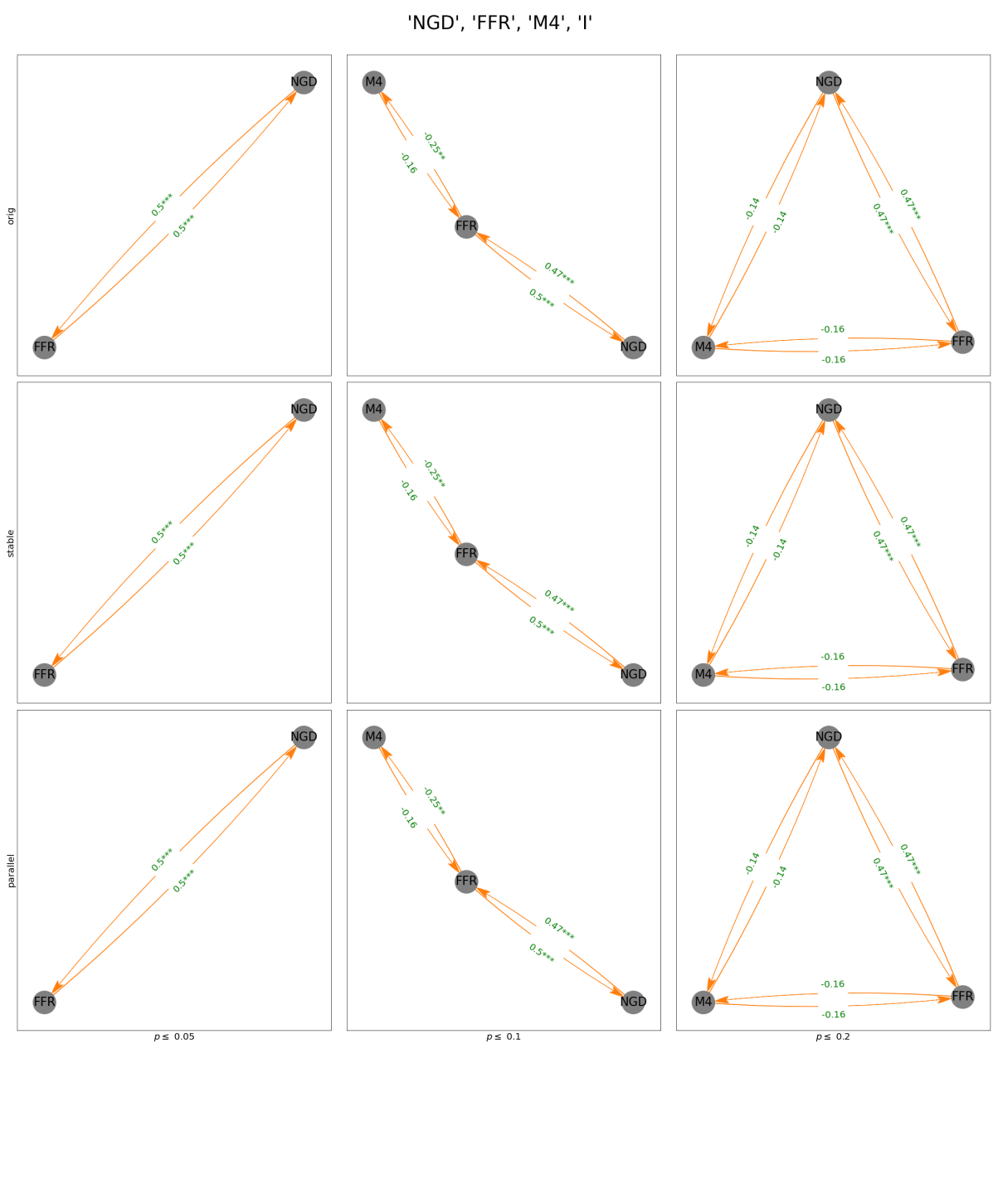
2: 100% 31.28it/s]

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

2: 100% 25.01it/s]

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

2: 100% 19.62it/s]



Working for n conditional variables: 2/3 [00:00<00:00,

2: 67% 29.87it/s]

Working for n conditional variables: 2/3 [00:00<00:00, 33.35it/s]
2: 67%

Working for n conditional variables: 2/3 [00:00<00:00, 14.82it/s]
2: 67%

Working for n conditional variables: 3/3 [00:00<00:00, 29.14it/s]
3: 100%

Working for n conditional variables: 3/3 [00:00<00:00, 32.99it/s]
3: 100%

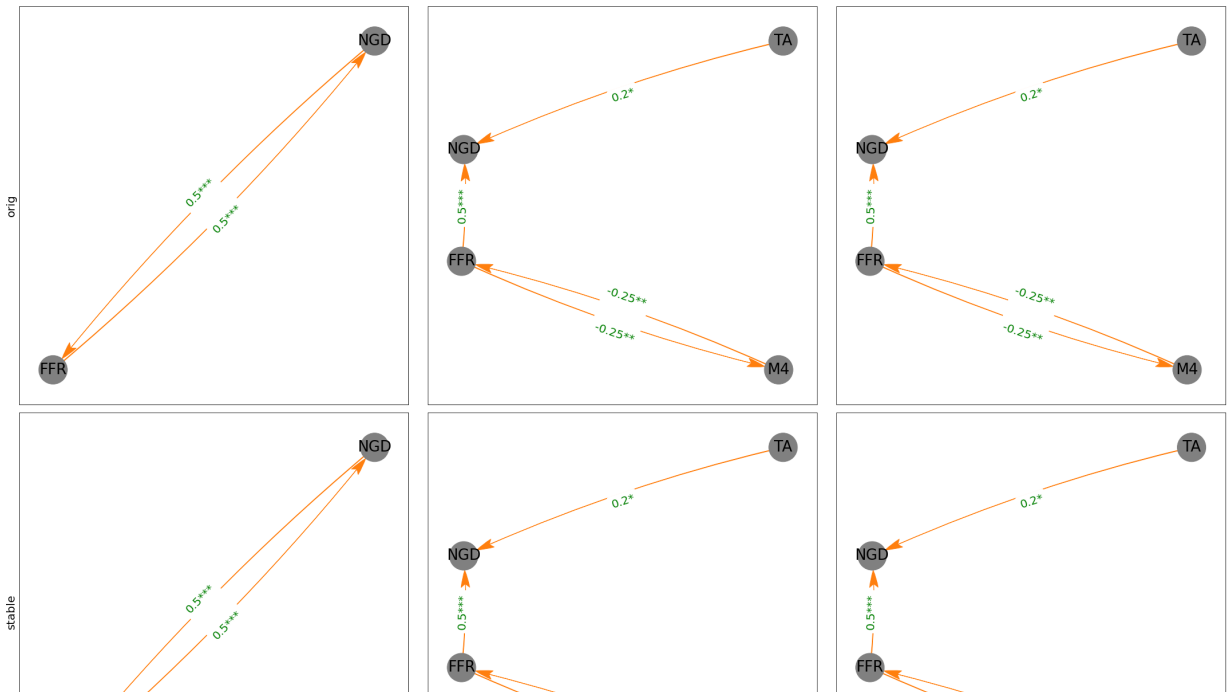
Working for n conditional variables: 3/3 [00:00<00:00, 16.82it/s]
3: 100%

Working for n conditional variables: 3/3 [00:00<00:00, 8.55it/s]
3: 100%

Working for n conditional variables: 3/3 [00:00<00:00, 19.06it/s]
3: 100%

Working for n conditional variables: 3/3 [00:00<00:00, 13.34it/s]
3: 100%

'TA', 'M4', 'FFR', 'NGD', 'I'



Working for n conditional variables: 3/3 [00:00<00:00, 3: 100% 35.09it/s]

Working for n conditional variables: 3/3 [00:00<00:00, 3: 100% 52.73it/s]

Working for n conditional variables: 3/3 [00:00<00:00, 3: 100% 29.85it/s]

Working for n conditional variables: 3/3 [00:00<00:00, 3: 100% 32.95it/s]

Working for n conditional variables: 3/3 [00:00<00:00, 3: 100% 36.92it/s]

Working for n conditional variables: 3/3 [00:00<00:00, 3: 100% 17.31it/s]

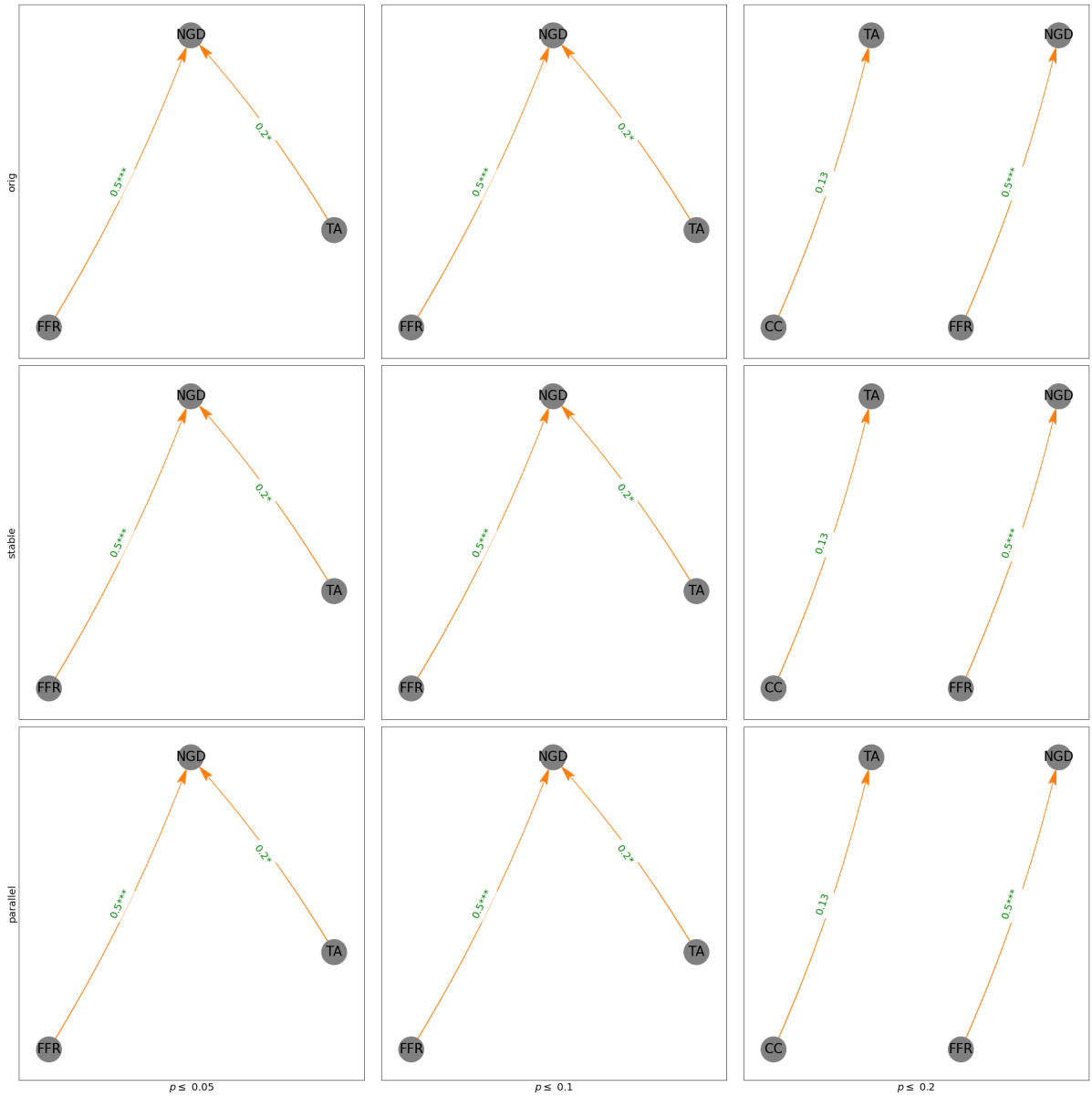
Working for n conditional variables: 3/3 [00:00<00:00, 3: 100% 41.20it/s]

Working for n conditional variables: 3/3 [00:00<00:00, 3: 100% 38.98it/s]

Working for n conditional variables: 3/3 [00:00<00:00,

3: 100% 28.44it/s]

'TA', 'NGD', 'FFR', 'CC', 'I'



Working for n conditional variables: 2/4 [00:00<00:00, 28.18it/s]
2: 50%

Working for n conditional variables: 2/4 [00:00<00:00, 19.60it/s]
2: 50%

Working for n conditional variables: 2/4 [00:00<00:00, 19.18it/s]
2: 50%

Working for n conditional variables: 3/4 [00:00<00:00, 19.80it/s]
3: 75%

Working for n conditional variables: 3/4 [00:00<00:00, 18.75it/s]
3: 75%

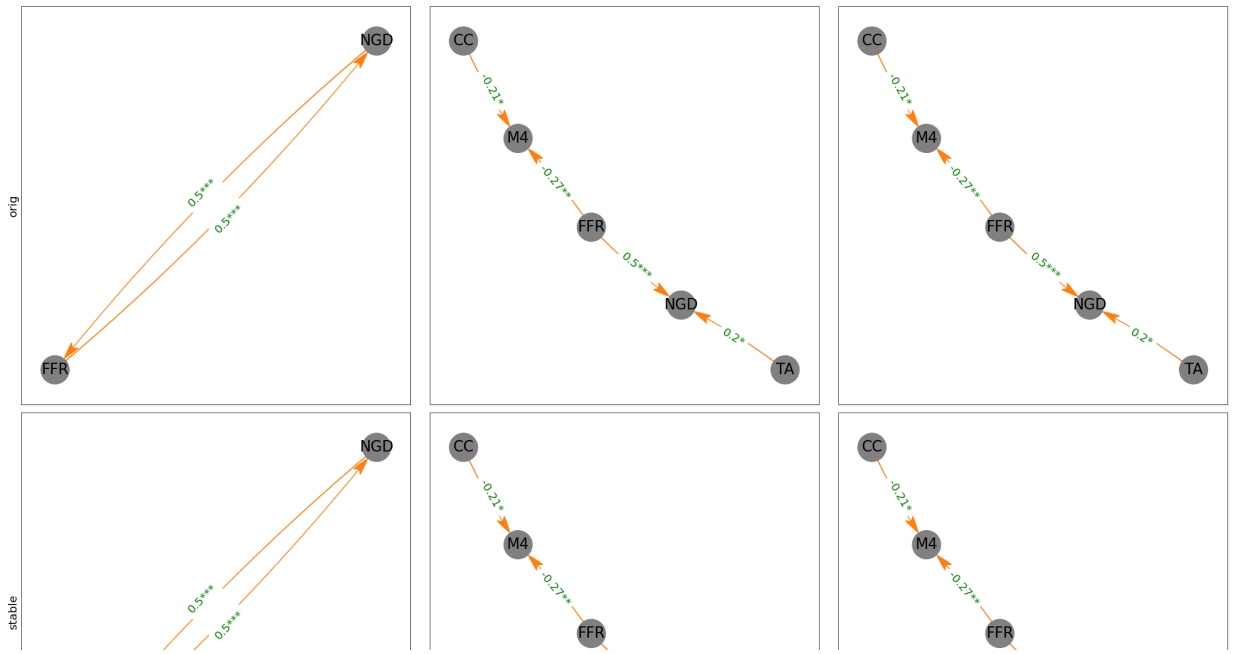
Working for n conditional variables: 3/4 [00:00<00:00, 15.10it/s]
3: 75%

Working for n conditional variables: 3/4 [00:00<00:00, 10.18it/s]
3: 75%

Working for n conditional variables: 3/4 [00:00<00:00, 9.08it/s]
3: 75%

Working for n conditional variables: 3/4 [00:00<00:00, 8.82it/s]
3: 75%

'TA', 'M4', 'FFR', 'NGD', 'CC', 'I'



Out[20]: OutEdgeView([('FFR', 'M4'), ('FFR', 'NGD'), ('CC', 'M4'), ('TA', 'NGD')])

In [ ]:

In [ ]: