

# Estudio de la propagación y dinámica de precios a partir de la Matriz de Insumo-Producto (MIP)

Miguel De Lillo, Manuel Fernandez, Augusto Kielbowicz, Mariano Oca

2024-07-01

# Contents

<b>1</b>	<b>Datos</b>	<b>2</b>
<b>2</b>	<b>Motivación</b>	<b>3</b>
<b>3</b>	<b>Introducción</b>	<b>4</b>
3.1	Preguntas a Responder . . . . .	4
3.2	Descripción del Modelo . . . . .	4
3.3	Suposiciones (y limitaciones) del Modelo . . . . .	4
<b>4</b>	<b>Matriz Insumo Producto</b>	<b>6</b>
<b>5</b>	<b>Análisis de la MIP</b>	<b>10</b>
5.1	Grafo . . . . .	10
5.2	Distribución de grado . . . . .	12
5.3	Centralidad de autovectores . . . . .	13
<b>6</b>	<b>Los experimentos</b>	<b>14</b>
6.0.1	Dinámicas . . . . .	15
6.1	Experimentación con la dinámica local. . . . .	15
6.2	Experimentos con las dinámicas. . . . .	18
6.2.1	Caso particular 1. . . . .	18
6.2.2	Caso particular 2 . . . . .	20
6.2.3	Alpha = 0 . . . . .	21
6.2.4	Alpha = 0.2 . . . . .	23
6.2.5	Alpha 0.3 . . . . .	25
6.2.6	Alpha 0.6 . . . . .	28
<b>7</b>	<b>Conclusiones</b>	<b>31</b>
7.1	Inflación Estructural . . . . .	31
7.2	Sobre los experimentos y las preguntas respondidas . . . . .	31
7.2.1	Preguntas a responder . . . . .	31
7.3	Próximos pasos . . . . .	32
<b>8</b>	<b>Apéndices</b>	<b>33</b>
8.1	Imports . . . . .	33
8.2	Código Fuente . . . . .	33
8.3	Dependencias . . . . .	45

# Chapter 1

## Datos

**Modelado y Simulación de Sistemas Complejos con Aplicaciones en Economía**

Departamento de Computación, FCyN, UBA

Grupo: 6

Integrantes:

- Miguel De Lillo, miguelangeldelillo@gmail.com, LU:622/23
- Manuel Fernandez, manufernandezbur@gmail.com, LU:1700/21
- Augusto Kielbowicz, augusto.kiel@gmail.com, LU:738/11
- Mariano Oca, marianoagoca@gmail.com, LU:206/20

**Repositorio**

---

## **Chapter 2**

# **Motivación**

La idea de este modelo es estudiar, ante un shock de precios en un sector dado, cómo afectan las dinámicas propuestas entre los sectores productivos de la Argentina en la inflación global calculada a partir de las variaciones en los precios de los mismos. (Utilizando las relaciones dadas por la matriz de Insumo-Producto).

# Chapter 3

## Introducción

**Inflación estructural:** La inflación estructural se refiere a un tipo de inflación causada por desajustes sectoriales que afectan a productos específicos, lo que resulta en aumentos de precios individuales que luego se generalizan. Estos desajustes pueden ser el resultado de la rigidez de la estructura productiva y la imperfección de los mercados, y son ajenos a las decisiones de las autoridades monetarias. La inflación estructural se origina en desequilibrios sectoriales que afectan a productos específicos y no en un desajuste global entre la oferta y la demanda monetaria.

*Olivera, Julio H. G. (1965), “Inflación estructural y política financiera”.*

### 3.1 Preguntas a Responder

- ¿Cómo se propaga el aumento de precios a través de la red definida por la MIP?
- ¿Cómo impacta el aumento de precio en un producto/sector sobre otros productos/sectores? ¿Se mantiene en la misma cadena productiva?
- ¿Cómo influyen la dinámica de comportamiento de los agentes en la variación de la inflación global?
- ¿Cuál es la sensibilidad del sistema respecto a variaciones de precio en nodos específicos? ¿Cuáles son los nodos que propagan de mayor forma la variación de precios?

### 3.2 Descripción del Modelo

- Cada sector productivo de la Argentina es representado por un nodo del digrafo. El mismo representaría a todos los productos (y productores) del sector.
- Cada arista del digrafo representa la relación “le vende a” donde el peso de la arista es el porcentaje de la producción total que es comparada.
- Tanto los sectores productivos como los pesos que le corresponden a cada arista son extraídos de la Matriz Inzumo Producto del 1997 publicada por el INDEC. Los valores de dicha matriz son normalizados para que representen porcentajes, afines a la experimentación que se quiere realizar sobre el modelo.

### 3.3 Suposiciones (y limitaciones) del Modelo

- Un cambio de precios  $>0$  en los insumos provoca un cambio de precios saliente del agente (producción). Sólo vamos a estudiar variaciones positivas en los precios para atenernos a las preguntas a investigar con el modelo.

- La economía es cerrada. Esto es, no se traen productos de otros países (importaciones) ni se vende nada a ellos (exportaciones) durante la evolución del sistema.
- Rige la Ley de Say: la oferta es igual a la demanda.

# Chapter 4

## Matriz Insumo Producto

Cargamos la matriz insumo producto normalizada por demanda saliente.

```
In [3]: mip = pd.read_csv("../resources/MIP_normalizada.csv", index_col=0)
mip.shape
```

```
Out[3]: (123, 123)
```

```
In [4]: mip.head()
```

```
Out[4]:
```

Cultivo de cereales, oleaginosas y forrajeras	Cultivo de cereales, oleaginosas y forrajeras	0.0103
Cultivo de hortalizas, legumbres, flores y plan...	Cultivo de hortalizas, legumbres, flores y plan...	0.000000
Cultivo de frutas y nueces	Cultivo de frutas y nueces	0.000000
Cultivos industriales	Cultivos industriales	0.000000
Producción de semillas	Producción de semillas	0.7831
Cultivo de cereales, oleaginosas y forrajeras	Cultivo de hortalizas, legumbres, flores y plan...	0.0160
Cultivo de hortalizas, legumbres, flores y plan...	Cultivo de hortalizas, legumbres, flores y plan...	0.1600
Cultivo de frutas y nueces	Cultivo de frutas y nueces	0.000000
Cultivos industriales	Cultivos industriales	0.000000
Producción de semillas	Producción de semillas	0.000000
Cultivo de cereales, oleaginosas y forrajeras	Cultivo de frutas y nueces \	0.001542
Cultivo de hortalizas, legumbres, flores y plan...	Cultivo de frutas y nueces \	0.009565
Cultivo de frutas y nueces	Cultivo de frutas y nueces \	0.000000
Cultivos industriales	Cultivos industriales \	0.000000
Producción de semillas	Cultivos industriales \	0.029820
Cultivo de cereales, oleaginosas y forrajeras	Cultivos industriales \	0.001601
Cultivo de hortalizas, legumbres, flores y plan...	Cultivos industriales \	0.000000
Cultivo de frutas y nueces	Cultivos industriales \	0.000000
Cultivos industriales	Cultivos industriales \	0.020212
Producción de semillas	Cultivos industriales \	0.017254
Cultivo de cereales, oleaginosas y forrajeras	Producción de semillas \	0.000000
Cultivo de hortalizas, legumbres, flores y plan...	Producción de semillas \	0.000000

Cultivo de frutas y nueces	0.00000
Cultivos industriales	0.00000
Producción de semillas	0.02174
Cultivo de cereales, oleaginosas y forrajeras	Cría de ganado y producción de leche, lana ...
Cultivo de hortalizas, legumbres, flores y plan...	0.00000
Cultivo de frutas y nueces	0.00000
Cultivos industriales	0.00000
Producción de semillas	0.00000
	Producción de granja \
Cultivo de cereales, oleaginosas y forrajeras	0.012263
Cultivo de hortalizas, legumbres, flores y plan...	0.000000
Cultivo de frutas y nueces	0.000000
Cultivos industriales	0.000000
Producción de semillas	0.002291
	Servicios agropecuarios \
Cultivo de cereales, oleaginosas y forrajeras	0.0
Cultivo de hortalizas, legumbres, flores y plan...	0.0
Cultivo de frutas y nueces	0.0
Cultivos industriales	0.0
Producción de semillas	0.0
	Caza \
Cultivo de cereales, oleaginosas y forrajeras	0.0
Cultivo de hortalizas, legumbres, flores y plan...	0.0
Cultivo de frutas y nueces	0.0
Cultivos industriales	0.0
Producción de semillas	0.0
	Silvicultura y extracción de madera \
Cultivo de cereales, oleaginosas y forrajeras	0.0
Cultivo de hortalizas, legumbres, flores y plan...	0.0
Cultivo de frutas y nueces	0.0
Cultivos industriales	0.0
Producción de semillas	0.0
	... Enseñanza pública \
Cultivo de cereales, oleaginosas y forrajeras	... 0.000038
Cultivo de hortalizas, legumbres, flores y plan...	... 0.005679
Cultivo de frutas y nueces	... 0.003827
Cultivos industriales	... 0.000000
Producción de semillas	... 0.000000
	Enseñanza privada \
Cultivo de cereales, oleaginosas y forrajeras	0.000080
Cultivo de hortalizas, legumbres, flores y plan...	0.012030
Cultivo de frutas y nueces	0.008099
Cultivos industriales	0.000000
Producción de semillas	0.000000
	Salud humana pública \

Cultivo de cereales, oleaginosas y forrajeras	0.000007
Cultivo de hortalizas, legumbres, flores y plan...	0.002192
Cultivo de frutas y nueces	0.011067
Cultivos industriales	0.000000
Producción de semillas	0.000000
Salud humana privada \	
Cultivo de cereales, oleaginosas y forrajeras	0.000042
Cultivo de hortalizas, legumbres, flores y plan...	0.008913
Cultivo de frutas y nueces	0.005904
Cultivos industriales	0.000000
Producción de semillas	0.000000
Servicios veterinarios \	
Cultivo de cereales, oleaginosas y forrajeras	0.000012
Cultivo de hortalizas, legumbres, flores y plan...	0.000000
Cultivo de frutas y nueces	0.000000
Cultivos industriales	0.000000
Producción de semillas	0.000000
Servicios sociales \	
Cultivo de cereales, oleaginosas y forrajeras	0.000000
Cultivo de hortalizas, legumbres, flores y plan...	0.001238
Cultivo de frutas y nueces	0.001638
Cultivos industriales	0.000000
Producción de semillas	0.000000
Servicios de saneamiento \	
Cultivo de cereales, oleaginosas y forrajeras	0.0
Cultivo de hortalizas, legumbres, flores y plan...	0.0
Cultivo de frutas y nueces	0.0
Cultivos industriales	0.0
Producción de semillas	0.0
Actividad de asociaciones \	
Cultivo de cereales, oleaginosas y forrajeras	0.000000
Cultivo de hortalizas, legumbres, flores y plan...	0.007999
Cultivo de frutas y nueces	0.000000
Cultivos industriales	0.000000
Producción de semillas	0.000000
Servicios de cine, radio y televisión \	
Cultivo de cereales, oleaginosas y forrajeras	1.735054e-09
Cultivo de hortalizas, legumbres, flores y plan...	1.237709e-06
Cultivo de frutas y nueces	1.472092e-07
Cultivos industriales	0.000000e+00
Producción de semillas	0.000000e+00
Servicios personales, de reparación, activi...	
Cultivo de cereales, oleaginosas y forrajeras	0.0
Cultivo de hortalizas, legumbres, flores y plan...	0.073
Cultivo de frutas y nueces	0.0
Cultivos industriales	0.0
Producción de semillas	0.0

[5 rows x 123 columns]

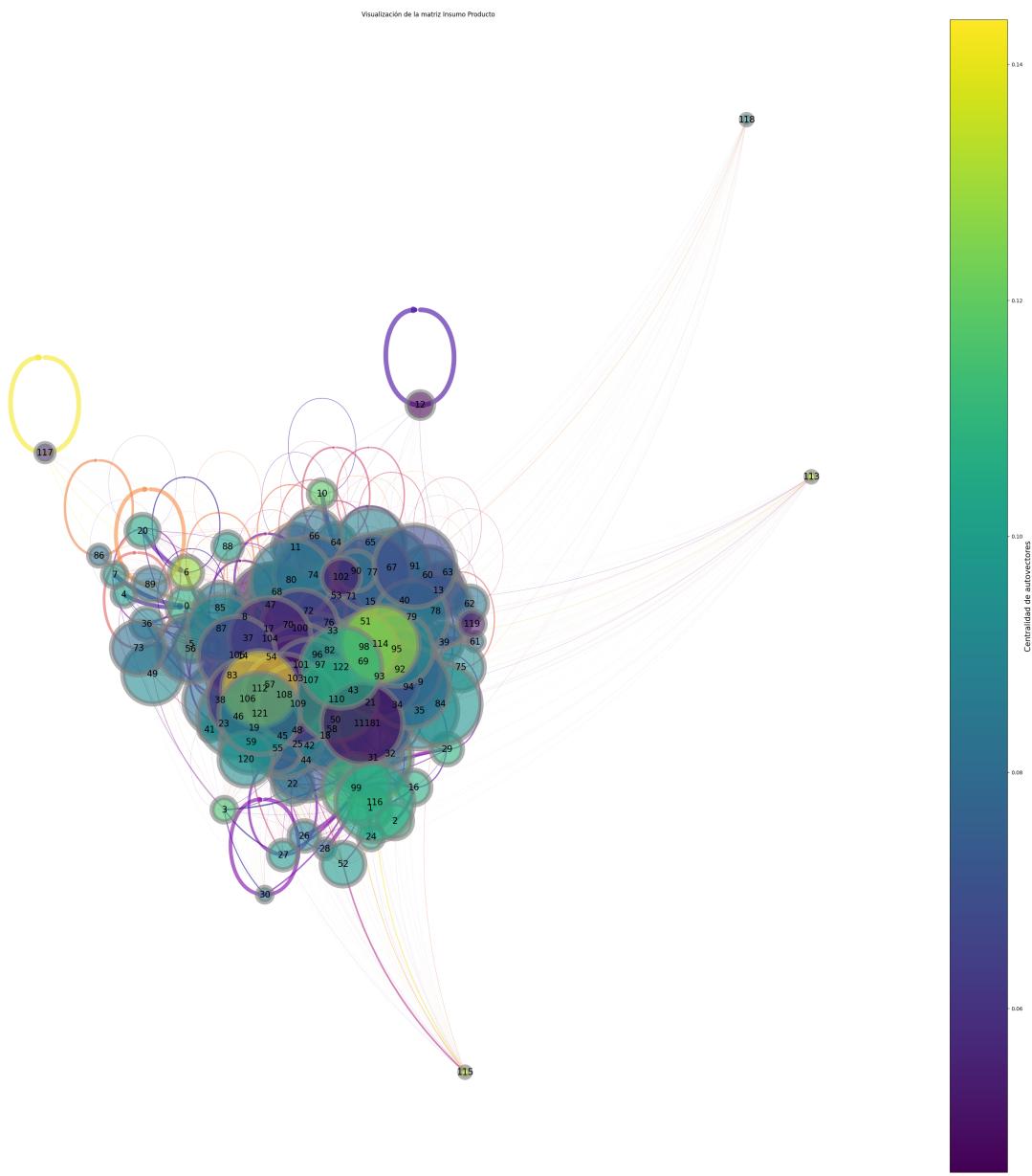
# Chapter 5

## Análisis de la MIP

### 5.1 Grafo

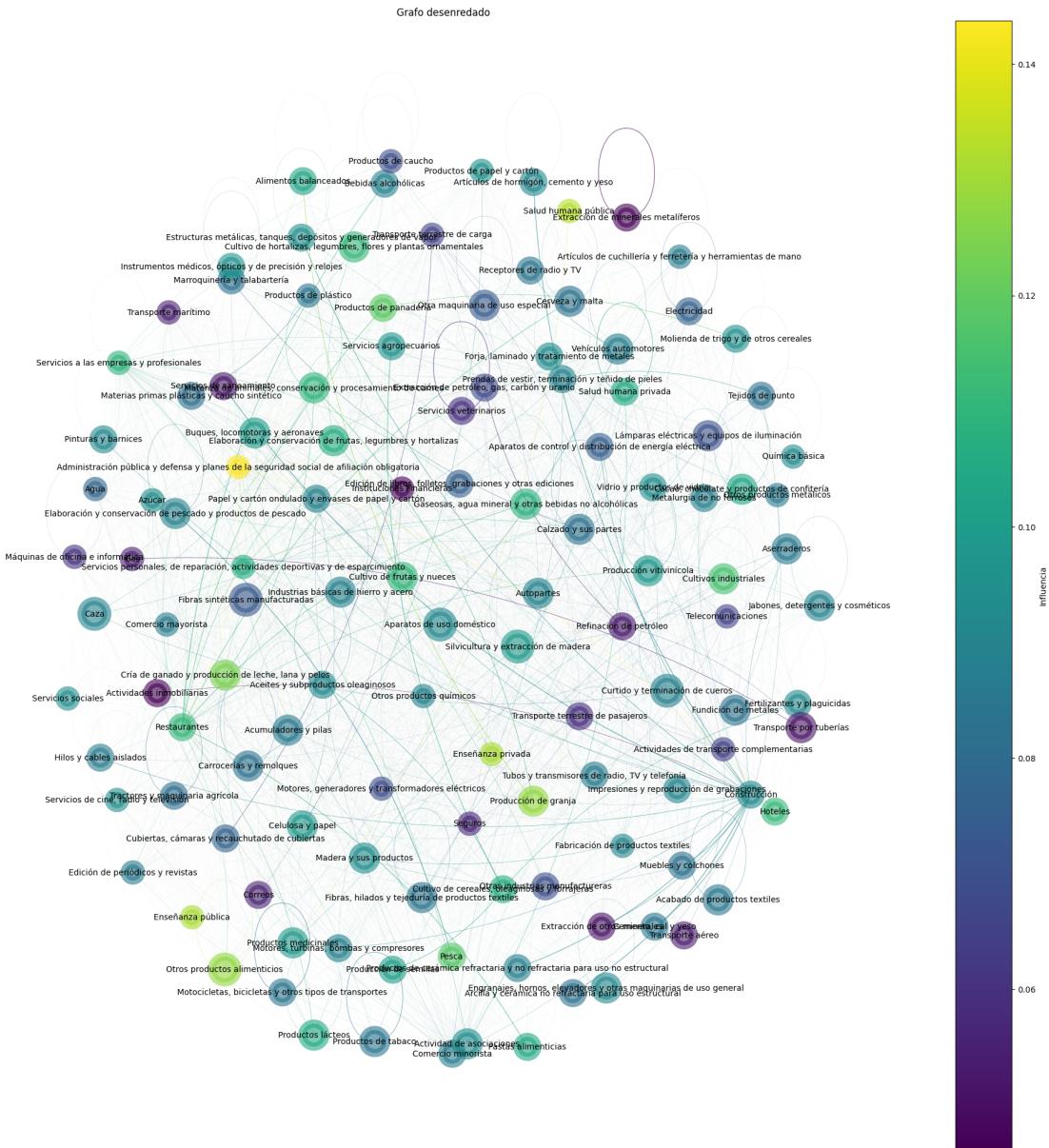
Veamos cómo se ve esta red.

```
In [5]: fig_gr_centralizado = grafo.verGrafoCentralizado(mip.to_numpy())
```



Intentemos separar los sectores para poder tener una mirada amplia.

In [6]: `fig_gr_abierto = grafo.verGrafoAbierto(mip)`



## 5.2 Distribución de grado

Veremos cómo se distribuye el grado en la red. Para tener una noción de qué está pasando.

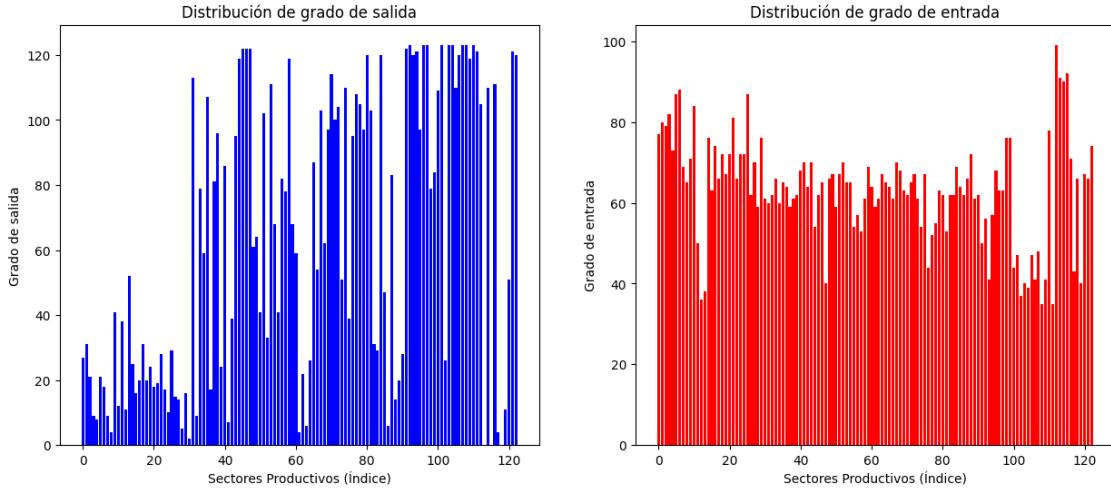
```
In [8]: G = nx.DiGraph(mip)
fig_dist_grados = grafo.graficar_dist_grados(G, mip.columns)
```

Media del Grado de salida:63.31707317073171

Desvío del Grado de salida:1961.1108467182228

Media del Grado de entrada:63.31707317073171

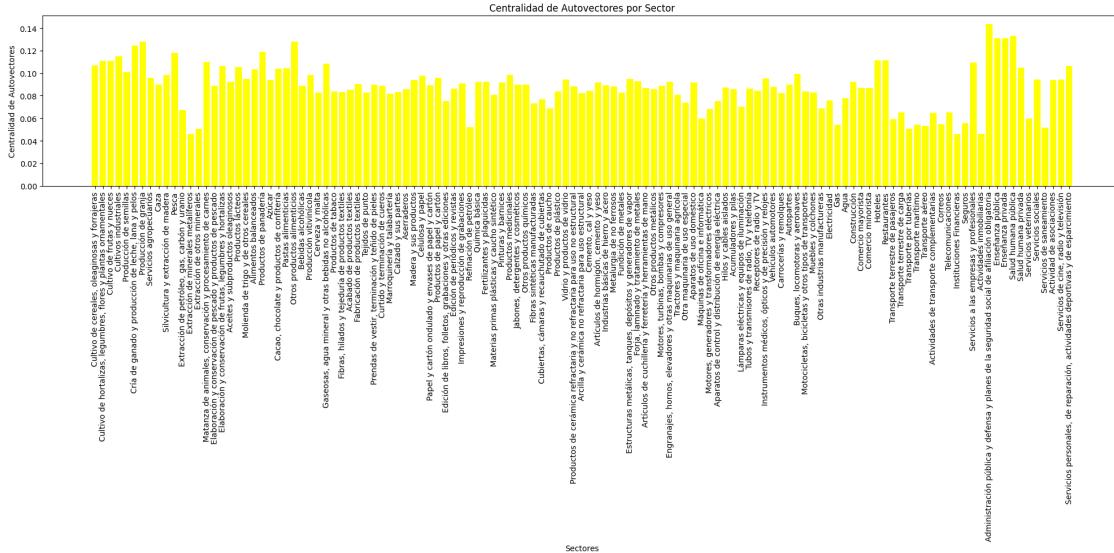
Desvío del Grado de entrada:162.3791394011501



### 5.3 Centralidad de autovectores

La centralidad de autovalores mide la influencia de un nodo sobre la red basada en la influencia de sus vecinos. Elegimos esta métrica para analizar el grafo pues los nodos que poseen un valor alto de esta medida de centralidad están conectados a otros nodos que a su vez son muy relevantes, en el sentido de la misma medida, o bien a muchos otros nodos, quizás menos relevantes. Por el contrario, los nodos conectados a otros nodos periféricos o poco relevantes, tendrán una baja centralidad de autovectores.

In [9]: `fig_dist_centralidad = grafo.graficar_dist_centralidad(G, mip.columns)`



# Chapter 6

## Los experimentos

En esta etapa, haremos un análisis de la divergencia y convergencia de la inflación. Para 4 valores de alfa distintos, vamos a hacer un shock inflacionario en cada uno de los 123 sectores productivos y computaremos la media móvil de la inflación entre estas 123 simulaciones para cada alfa. Además, mostraremos el grafo resultante luego de la evolución del sistema para dos sectores representativos de niveles de centralidad alto y bajo, respectivamente.

```
In [10]: alfa0 = pd.read_csv("../resources/alpha00.csv", index_col=0)
        alfa02 = pd.read_csv("../resources/alpha02.csv", index_col=0)
        alfa03 = pd.read_csv("../resources/alpha03.csv", index_col=0)
        alfa04 = pd.read_csv("../resources/alpha04.csv", index_col=0)
```

```
In [11]: exp.Experimento?
```

```
Init signature:
exp.Experimento(    grafo,      dinamica,      duracion_periodo=10,      metricas=None,      calcular_inflacion)
Docstring:      <no docstring>
Init docstring:
Inicializa una instancia de la clase Experimento.

Parámetros
-----
grafo : DiGraph
    Grafo que representa el MIP
dinamica : Callable
    Función con la siguiente firma : (aumento, peso_arista, inflacion, alpha)
duracion_periodo : int, opcional
    La duración de cada período en pasos para actualizar la inflacion, por defecto 10 pasos.
metricas : Dict(string,Callable), opcional
    Diccionario de con metricas a computar sobre el grafo, ej, {"cantidad_nodos", lambda grafo: len(grafo)}
calcular_inflacion : Callable, opcional
    Función para calcular la inflación en un período, recibe dos arrays de precios. ej, calc(precios_actuales, precios_anteriores)
alpha : float, opcional
    Parámetro de ponderación
    El alpha es CUANTO ve la inflación.
    Alpha == 1 es lo mismo que la dinamica 2.
    Alpha == 0 es lo mismo que la dinamica 1.
File:          c:\users\augus\dev\gh\akielbowicz\tp-msscae-2024\src\experimento.py
Type:          type
Subclasses:
```

### 6.0.1 Dinámicas

```
$dinamica_{local}(aumento, peso_{arista}, inflacion) = aumento * peso_{arista} $  
dinamica_global(aumento, peso_arista, inflacion) = inflacion  
$dinamica_{mixta}(aumento, peso_{arista}, inflacion, ) = * inflacion + (1 - ) * aumento *  
peso_{arista} $
```

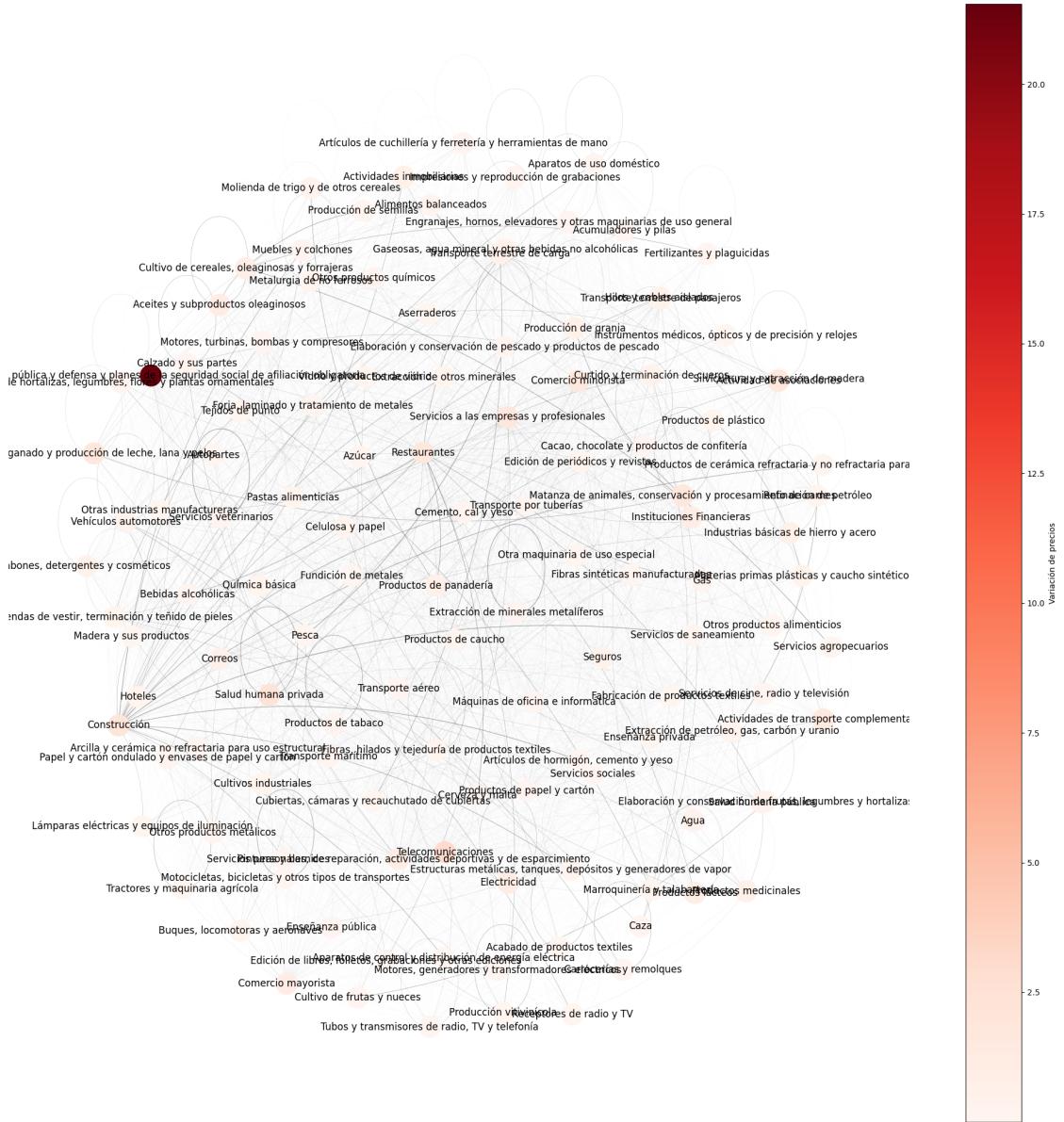
## 6.1 Experimentación con la dinámica local.

Veamos qué sucede con:

- un sector muy central: *Administración pública y defensa y planes de la seguridad social de afiliación obligatoria*
- otro poco central: *Instituciones Financieras*

```
In [12]: def calcular_precios(mip, sector, dinamica, alpha=0.0):  
    mip_grafo = grafo.armar_grafo(mip, precios_random=False)  
    experimento = exp.Experimento(  
        mip_grafo,  
        dinamica=dinamica,  
        calcular_inflacion=modelo.calcular_inflacion,  
        duracion_periodo=20,  
        alpha=alpha,  
    )  
    experimento.shock(sector, 20)  
    experimento.step(600)  
    precios_finales = []  
    for sector in mip_grafo.nodes():  
        precio = experimento.precios_periodo_pasado[sector]  
        precios_finales.append(precio)  
  
    precios_iniciales = [100 for _ in precios_finales]  
    return precios_iniciales, precios_finales
```

```
In [13]: sector = "Administración pública y defensa y planes de la seguridad social de afiliación obligatoria"  
precios_iniciales, precios_finales = calcular_precios(  
    mip, sector, modelo.dinamica_local  
)  
fig_inf = grafo.verInflacion(mip, precios_finales, precios_iniciales)
```



```
In [14]: sector = "Instituciones Financieras"
precios_iniciales, precios_finales = calcular_precios(
    mip, sector, modelo.dinamica_local
)
fig_inf = grafo.verInflacion(mip, precios_finales, precios_iniciales)
```



```
In [15]: def graficar_inflacion_para(mip_grafo, aumento, sectores=[]):
    inflaciones = []
    for sector in sectores:
        experimento = exp.Experimento(
            mip_grafo,
            dinamica=modelo.dinamica_local,
            calcular_inflacion=modelo.calcular_inflacion,
            duracion_periodo=20,
        )
        experimento.shock(sector, aumento)
        experimento.step(600)
        inflacion = experimento.metricas_evaluadas["inflacion"]
    return inflaciones
```

```

        inflaciones.append(inflacion)
    plot_inflaciones(inflaciones, sectores, aumento, 3)

```

In [16]: aumentos = [5, 20, 50]

In [17]: mip\_grafo = grafo.armar\_grafo(mip, precios\_random=False)

## 6.2 Experimentos con las dinámicas.

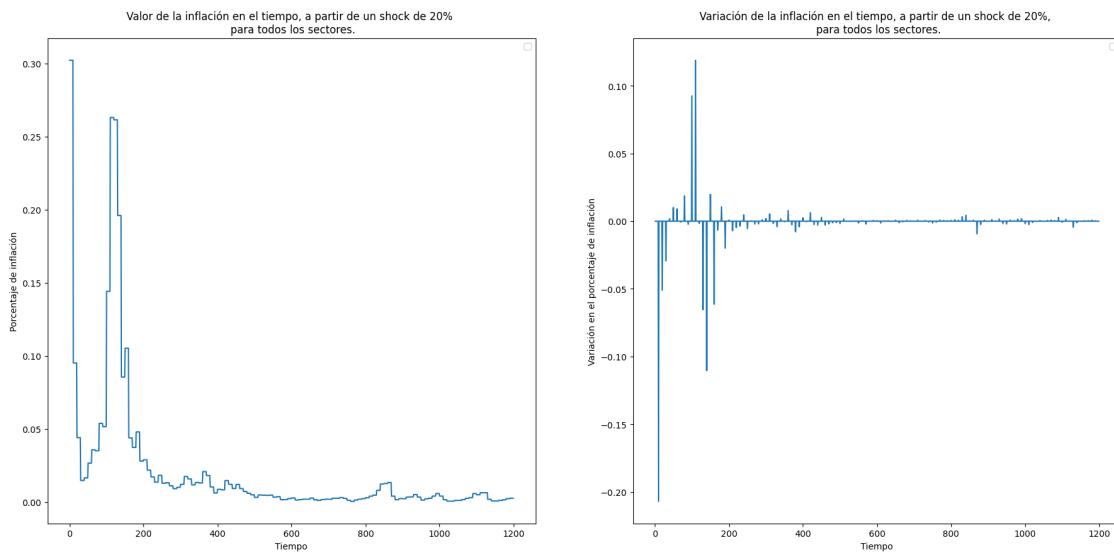
### 6.2.1 Caso particular 1.

Shock con a “Electricidad” con Alpha 0.2

```

In [18]: experimento = exp.Experimento(
            mip_grafo,
            dinamica=modelo.dinamica_mixta,
            calcular_inflacion=modelo.calcular_inflacion,
            alpha=0.2,
        )
        experimento.shock("Electricidad", 20)
        experimento.step(1200)
        inflacion = experimento.metricas_evaluadas["inflacion"]
        inflaciones = []
        inflaciones.append(inflacion)
        fig = plot_inflaciones(inflaciones, "Electricidad", 20, 3000)

```



Shock a “Electricidad” con Alpha 0.248

```

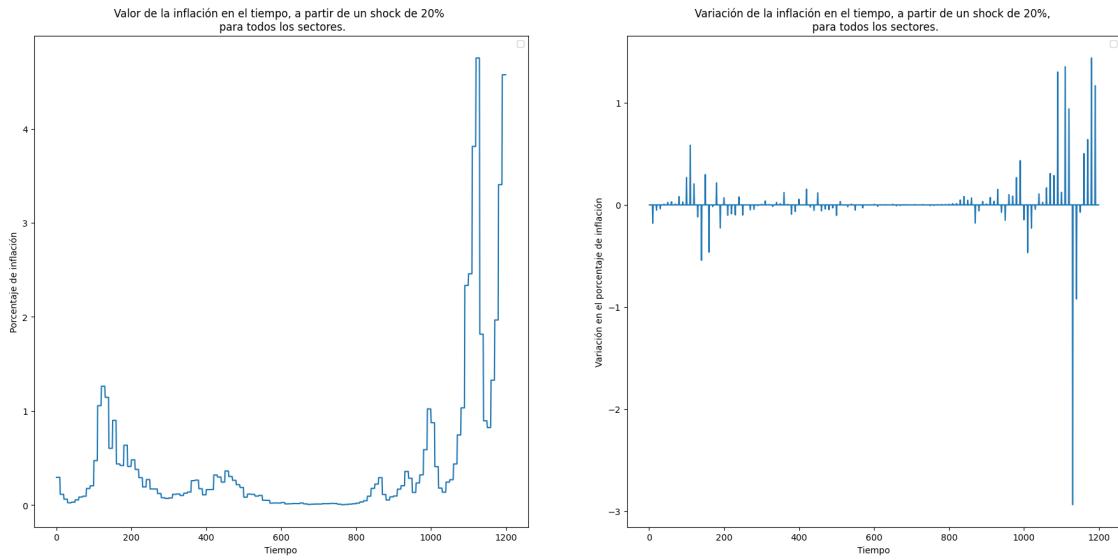
In [19]: experimento = exp.Experimento(
            mip_grafo,
            dinamica=modelo.dinamica_mixta,
            calcular_inflacion=modelo.calcular_inflacion,
            alpha=0.248,
        )

```

```

experimento.shock("Electricidad", 20)
experimento.step(1200)
inflacion = experimento.metricas_evaluadas["inflacion"]
inflaciones = []
inflaciones.append(inflacion)
fig = plot_inflaciones(inflaciones, "Electricidad", 20, 3000)

```

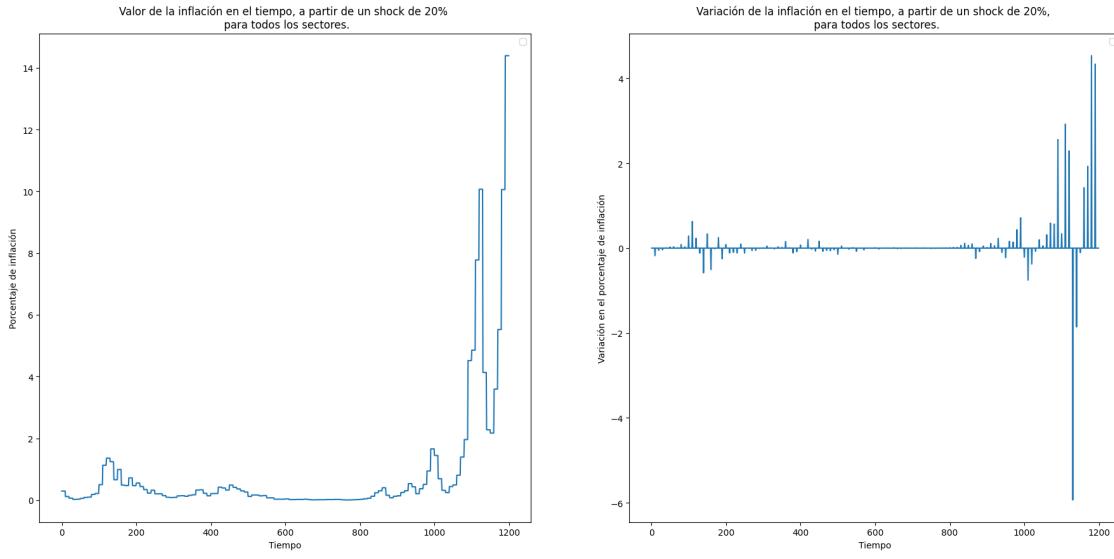


### Shock con a “Electricidad” con Alpha 0.25

```

In [20]: experimento = exp.Experimento(
    mip_grafo,
    dinamica=modelo.dinamica_mixta,
    calcular_inflacion=modelo.calcular_inflacion,
    alpha=0.25,
)
experimento.shock("Electricidad", 20)
experimento.step(1200)
inflacion = experimento.metricas_evaluadas["inflacion"]
inflaciones = []
inflaciones.append(inflacion)
fig = plot_inflaciones(inflaciones, "Electricidad", 20, 3000)

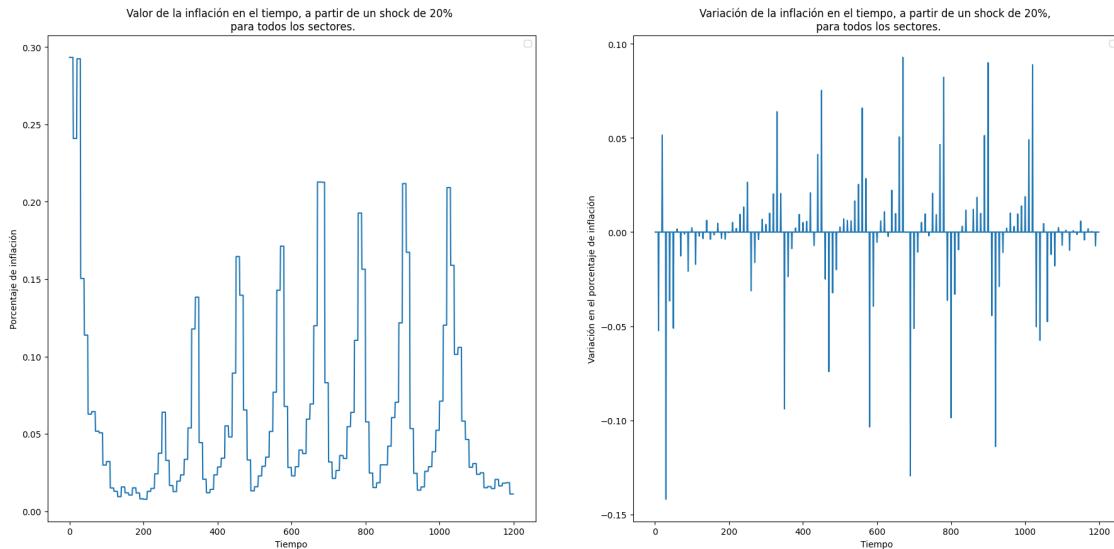
```



### 6.2.2 Caso particular 2

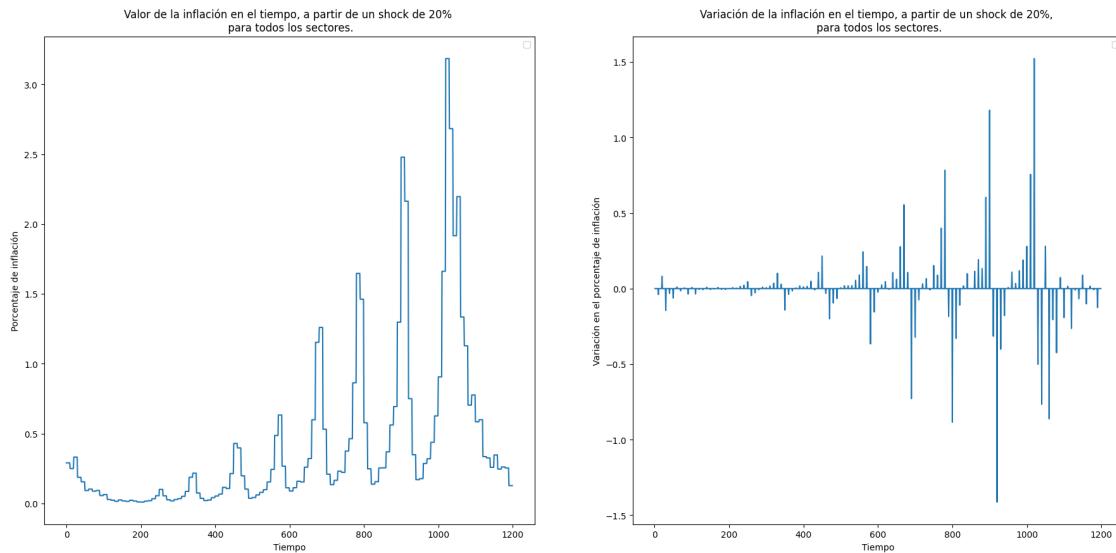
Shock a “Productos de panadería” con Alpha 0.20

```
In [21]: experimento = exp.Experimento(
    mip_grafo,
    dinamica=modelo.dinamica_mixta,
    calcular_inflacion=modelo.calcular_inflacion,
    alpha=0.20,
)
experimento.shock("Productos de panadería", 20)
experimento.step(1200)
inflacion = experimento.metricas_evaluadas["inflacion"]
inflaciones = []
inflaciones.append(inflacion)
fig = plot_inflaciones(inflaciones, "Productos de panadería", 20, 3000)
```



### Shock a “Productos de panadería” con Alpha 0.22

```
In [22]: experimento = exp.Experimento(
    mip_grafo,
    dinamica=modelo.dinamica_mixta,
    calcular_inflacion=modelo.calcular_inflacion,
    alpha=0.22,
)
experimento.shock("Productos de panadería", 20)
experimento.step(1200)
inflacion = experimento.metricas_evaluadas["inflacion"]
inflaciones = []
inflaciones.append(inflacion)
fig = plot_inflaciones(inflaciones, "Productos de panadería", 20, 3000)
```



### 6.2.3 Alpha = 0

```
In [23]: alpha = 0
```

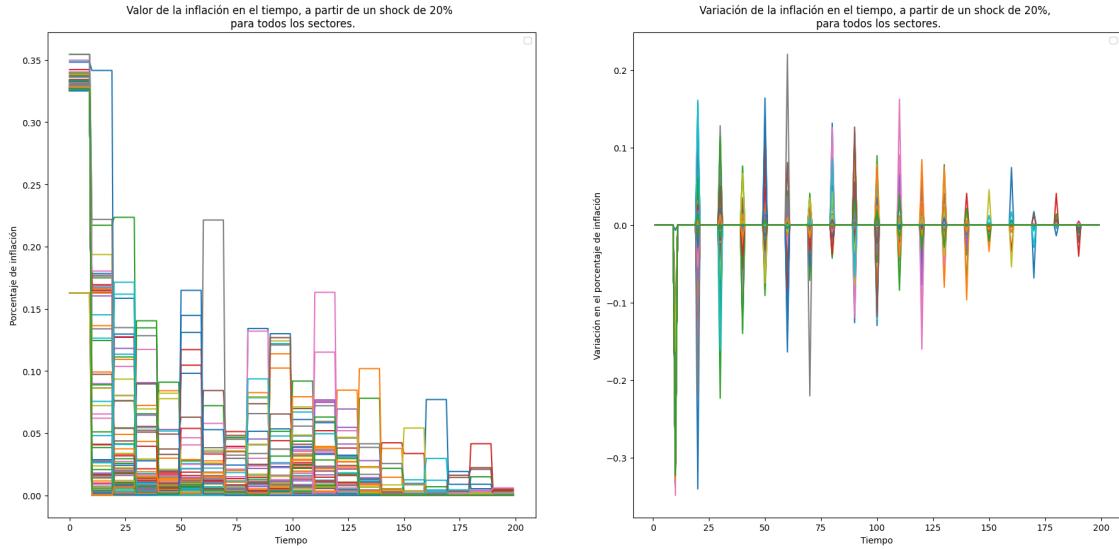
```
In [24]: aumentos = [20]
```

```
sectores = list(mip_grafo.nodes)
for j, aumento in enumerate(aumentos):
    inflaciones = []
    for sector in sectores:
        experimento = exp.Experimento(
            mip_grafo,
            dinamica=modelo.dinamica_mixta,
            calcular_inflacion=modelo.calcular_inflacion,
            alpha=alpha,
        )
```

```

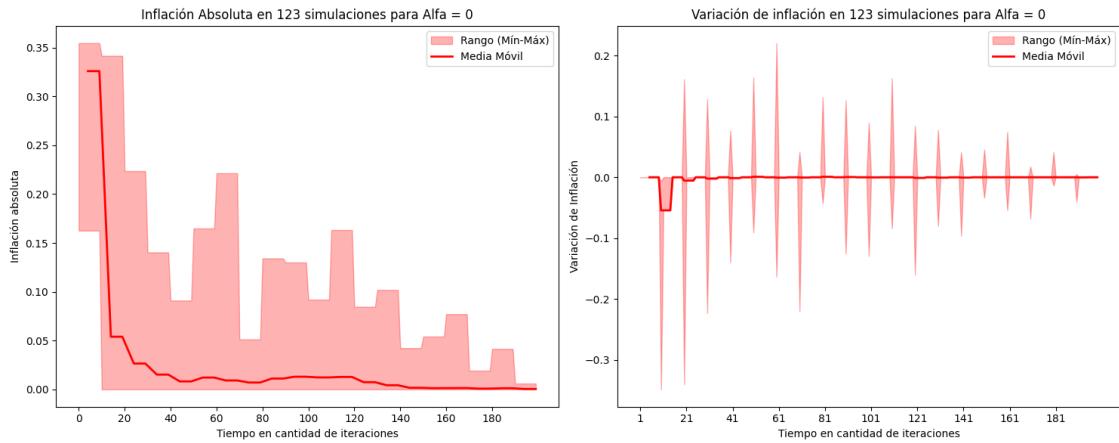
experimento.shock(sector, aumento)
experimento.step(200)
inflacion = experimento.metricas_evaluadas["inflacion"]
inflaciones.append(inflacion)
plot_inflaciones(inflaciones, sectores, aumento)

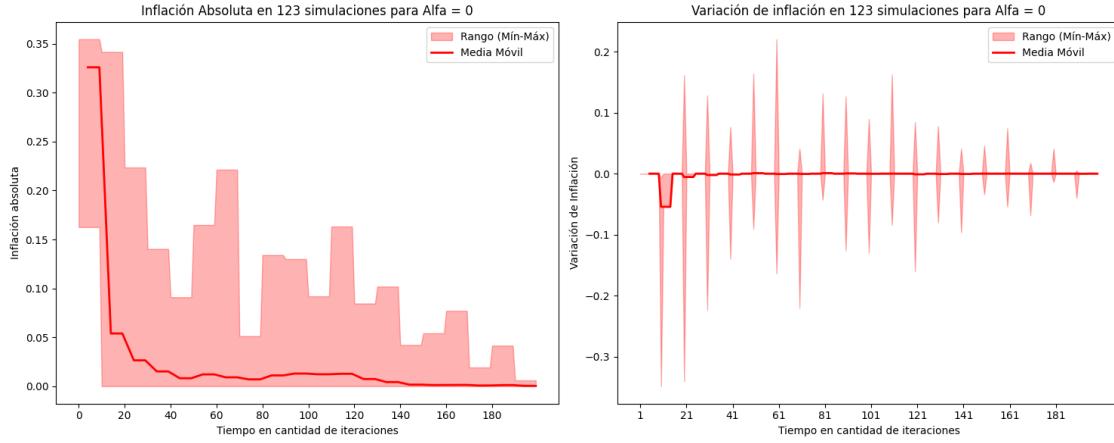
```



In [25]: verEvolucion(alfa0, "Alfa = 0")

Out[25]:





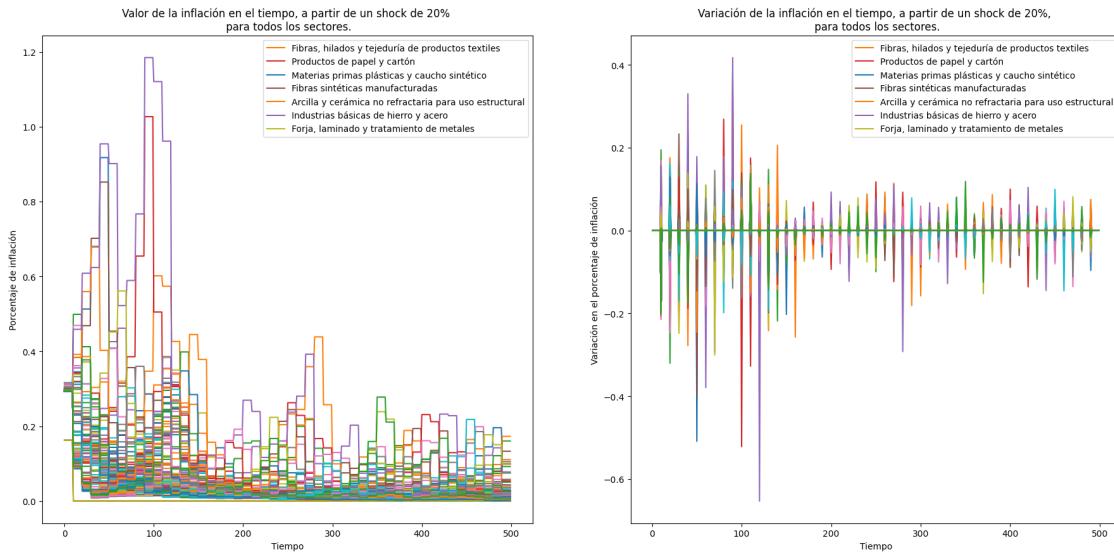
#### 6.2.4 Alpha = 0.2

```
In [26]: alpha = 0.2

In [27]: aumentos = [20]
          umbral_visualizacion = [1000, 2000, 2000]

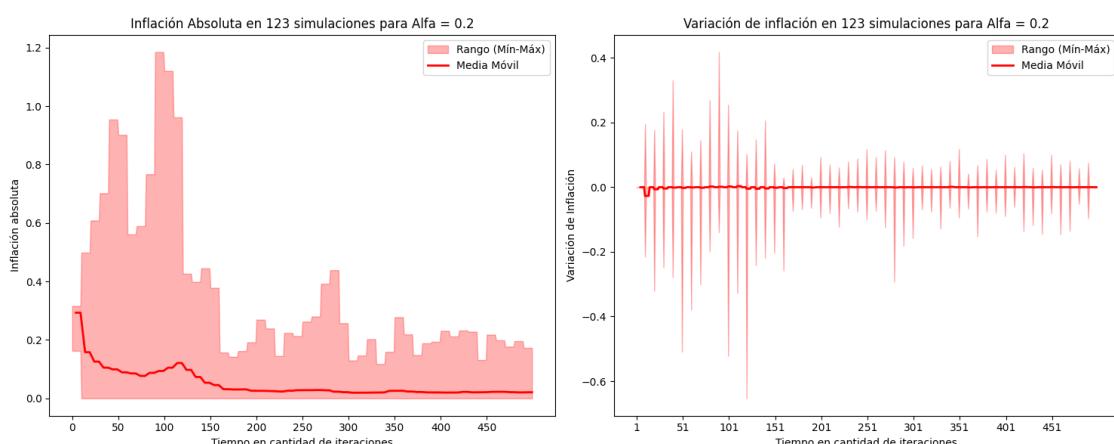
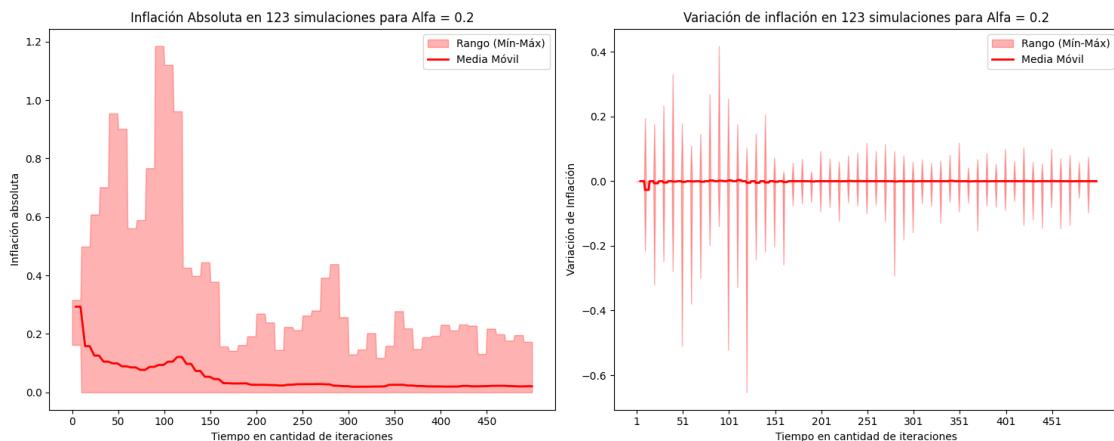
          sectores = list(mip_grafo.nodes)
          for j, aumento in enumerate(aumentos):
              inflaciones = []
              for sector in sectores:
                  experimento = exp.Experimento(
                      mip_grafo,
                      dinamica=modelo.dinamica_mixta,
                      calcular_inflacion=modelo.calcular_inflacion,
                      alpha=alpha,
                  )
                  experimento.shock(sector, aumento)
                  experimento.step(500)
                  inflacion = experimento.metricas_evaluadas["inflacion"]
                  inflaciones.append(inflacion)

          plot_inflaciones(inflaciones, sectores, aumento, 0.5)
```



```
In [28]: verEvolucion(alfa02, f"Alfa = {alpha}")
```

```
Out[28]:
```



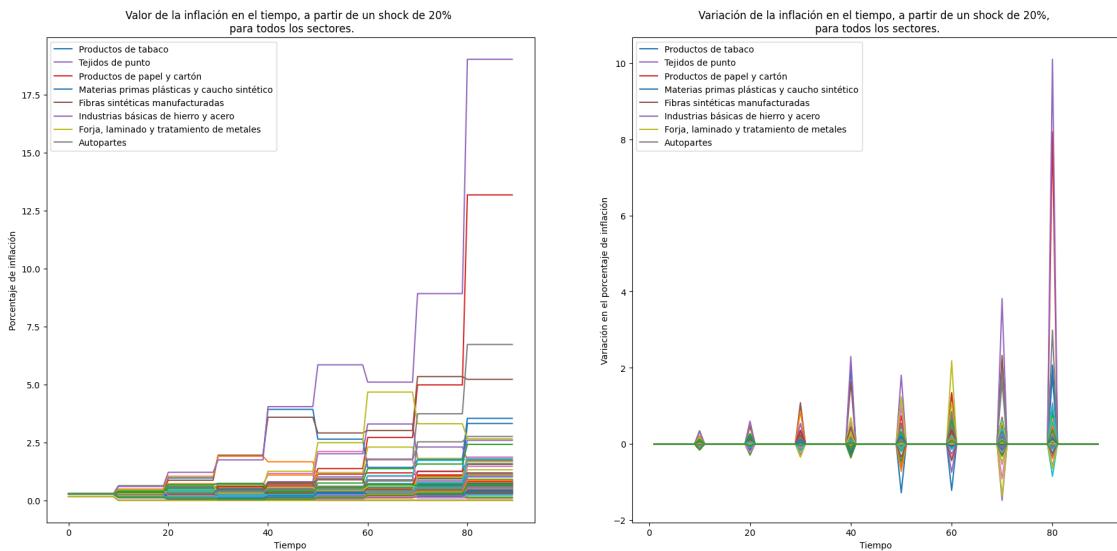
### 6.2.5 Alpha 0.3

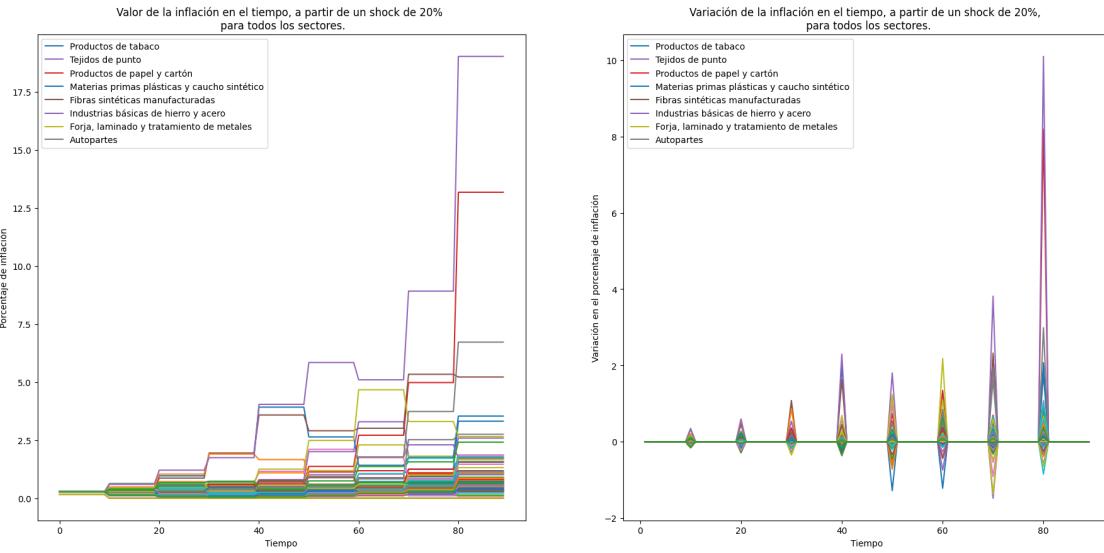
In [29]: `alpha = 0.3`

In [30]: `aumento = 20`  
`# umbral_visualizacion = [1000]`

```
sectores = list(mip_grafo.nodes)
inflaciones = []
for sector in sectores:
    experimento = exp.Experimento(
        mip_grafo,
        dinamica=modelo.dinamica_3,
        calcular_inflacion=modelo.calcular_inflacion,
        alpha=alpha,
    )
    experimento.shock(sector, aumento)
    experimento.step(90)
    inflacion = experimento.metricas_evaluadas["inflacion"]
    inflaciones.append(inflacion)
plot_inflaciones(inflaciones, sectores, aumento, 3)
```

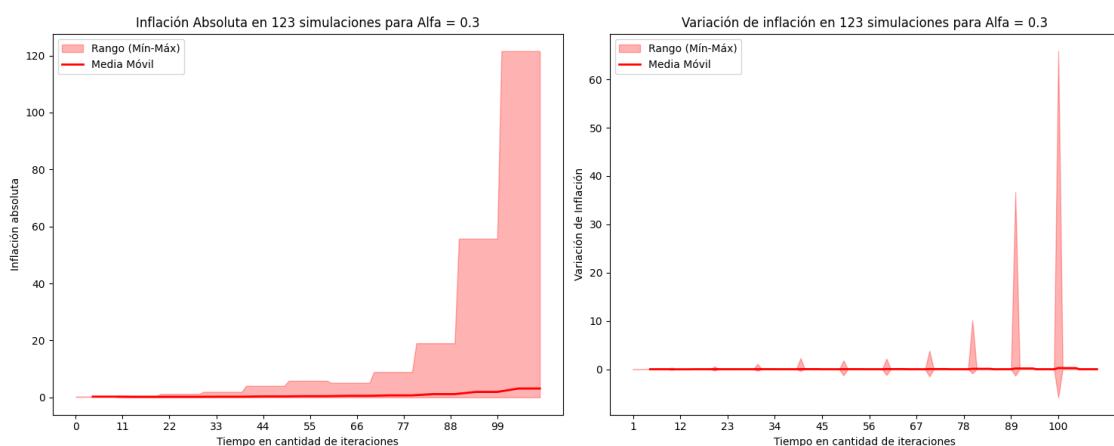
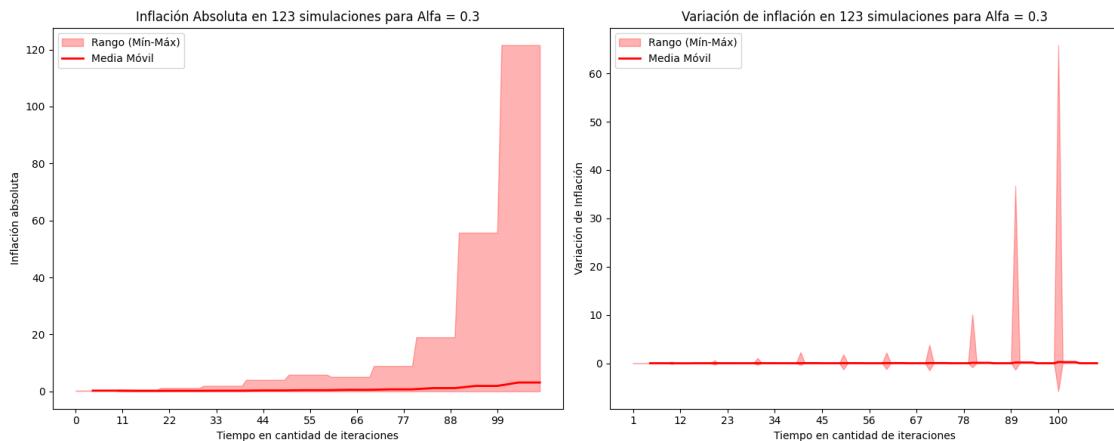
Out[30]:





```
In [31]: verEvolucion(alfa03, f"Alfa = {alpha}")
```

```
Out[31]:
```

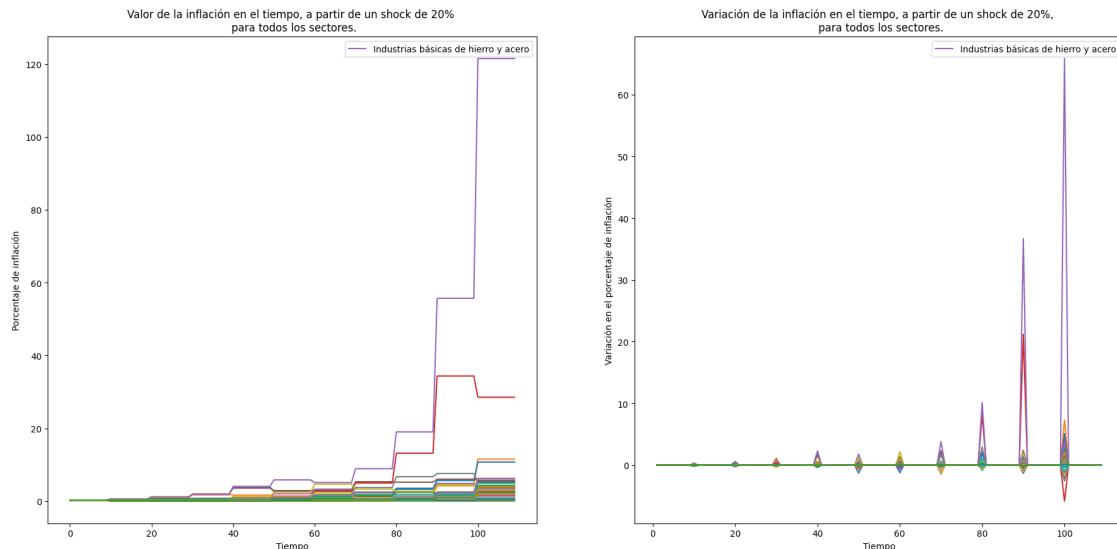


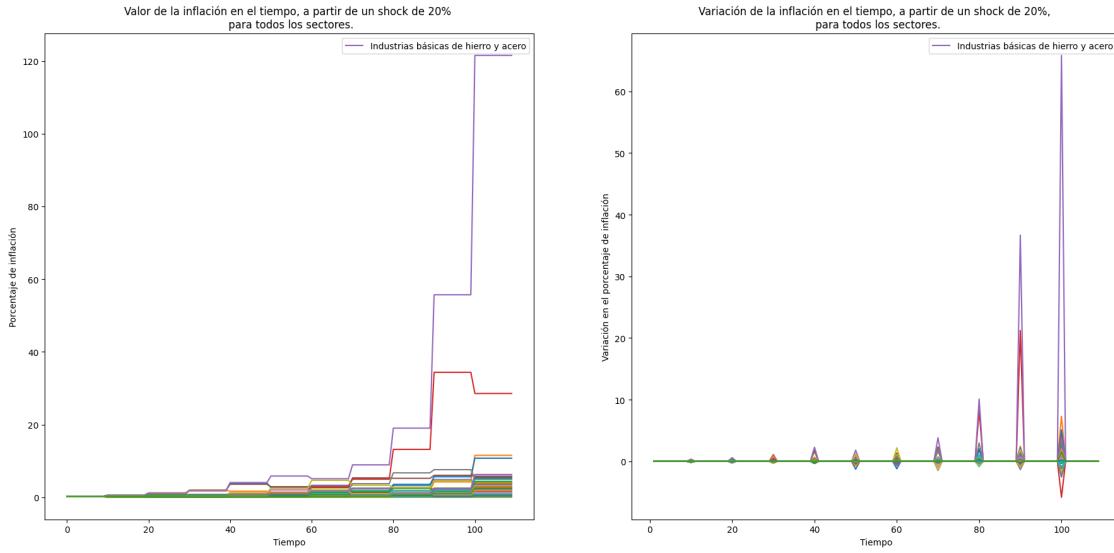
El mismo experimento pero con más pasos.

In [32]: aumento = 20

```
sectores = list(mip_grafo.nodes)
inflaciones = []
for sector in sectores:
    experimento = exp.Experimento(
        mip_grafo,
        dinamica=modelo.dinamica_3,
        calcular_inflacion=modelo.calcular_inflacion,
        alpha=alpha,
    )
    experimento.shock(sector, aumento)
    experimento.step(110)
    inflacion = experimento.metricas_evaluadas["inflacion"]
    inflaciones.append(inflacion)
plot_inflaciones(inflaciones, sectores, aumento, 50)
```

Out[32] :





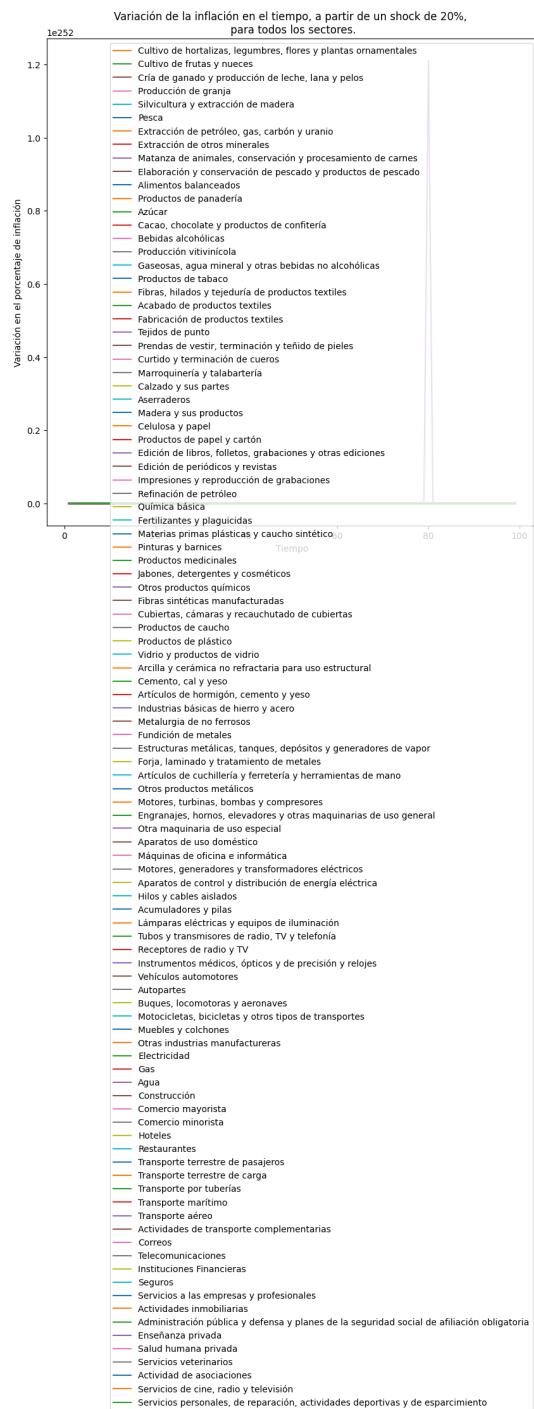
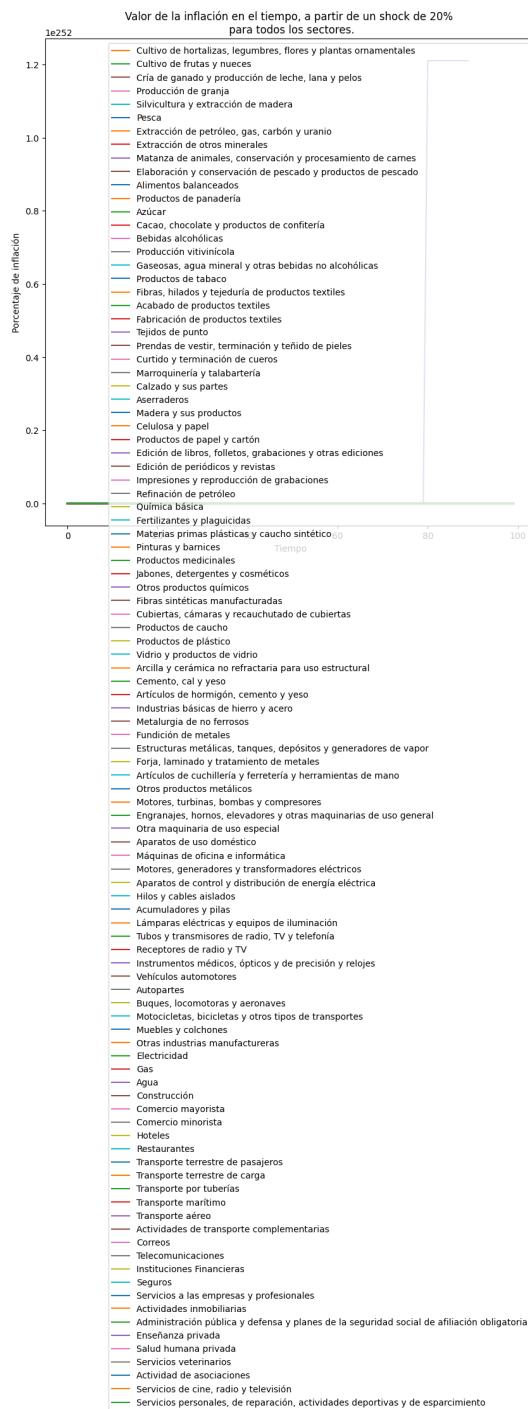
### 6.2.6 Alpha 0.6

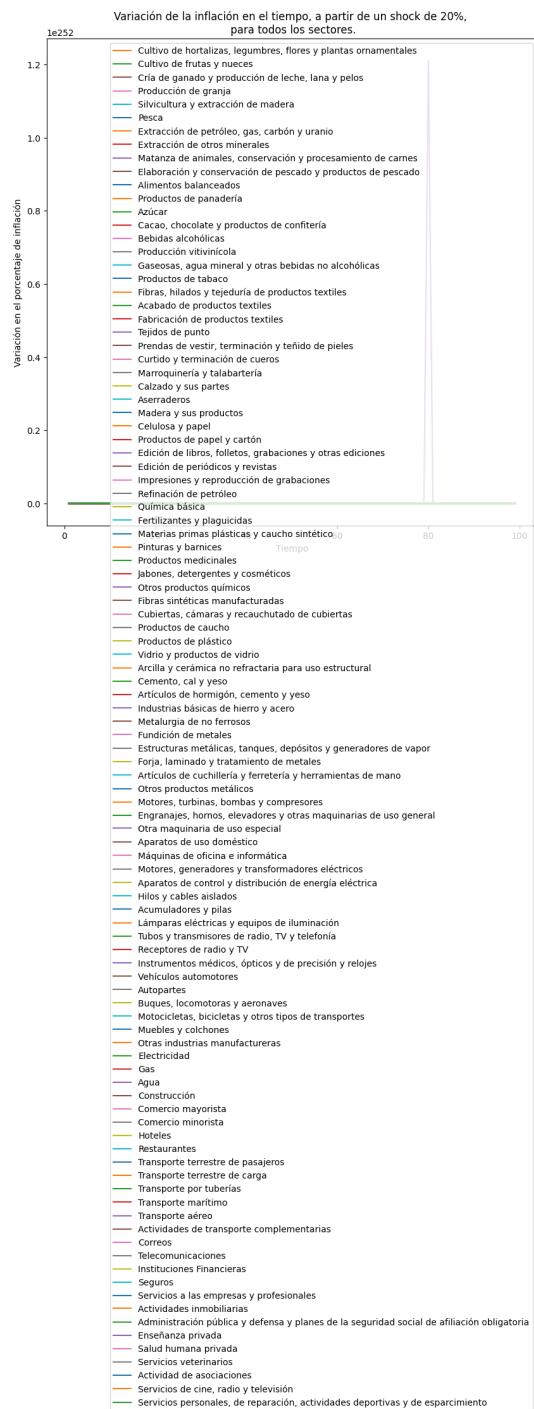
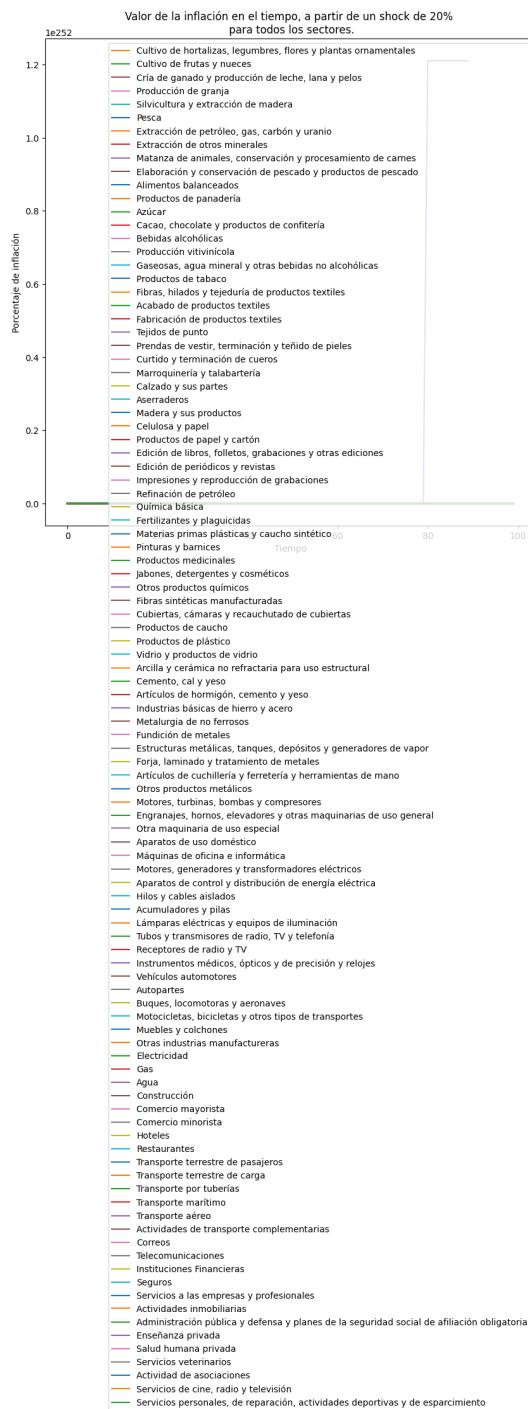
In [33]: `alpha = 0.6`

In [34]: `aumento = 20`

```
# umbral_visualizacion = [1000]
sectores = list(mip_grafo.nodes)
inflaciones = []
for sector in sectores:
    experimento = exp.Experimento(
        mip_grafo,
        dinamica=modelo.dinamica_mixta,
        calcular_inflacion=modelo.calcular_inflacion,
        alpha=alpha,
    )
    experimento.shock(sector, aumento)
    experimento.step(100)
    inflacion = experimento.metricas_evaluadas["inflacion"]
    inflaciones.append(inflacion)
plot_inflaciones(inflaciones, sectores, aumento, 3000)
```

Out[34]:





# Chapter 7

## Conclusiones

### 7.1 Inflación Estructural

Podemos notar que este modelo apoya la noción de que la inflación tiene un origen estructural. El hecho de que exista un alfa tal que si el aumento local de precios en un sector productivo estuviese dado por  $\alpha * \text{inflación\_global}$ , podría ser interpretado como una descoordinación entre los agentes. Estos hitos en la red estarían dando lugar a una espiralización de la inflación del modelo, que podría ser interpretada como la generalización de aumentos de precios individuales.

### 7.2 Sobre los experimentos y las preguntas respondidas

- ¿Cómo se propaga el aumento de precios a través de la red definida por la MIP?

La propagación de un shock en el sistema productivo varía dependiendo tanto del sector a cual se realiza el shock como del alpha del experimento. En nodos que están fuertemente conectados con otros, que propagan más un aumento, los shocks tienen más impacto que en aquellos con grados o pesos menores en sus aristas.

- ¿Cómo impacta el aumento de precio en un producto/sector sobre otros productos/sectores? ¿Se mantiene en la misma cadena productiva? ¿Cómo influyen la dinámica de comportamiento de los agentes en la variación de la inflación global?

Si el alpha es 0 el shock a un sector sólo genera cambios de precios en las aristas alcanzables por ese nodo. Además, como todos los pesos en las aristas son menores a 1, existe un decaimiento en los aumentos que se van pasando de sector a sector. Por otro lado, cuando el alpha aumenta y los agentes se ven cada vez más influenciados por la inflación global a la hora de aumentar sus precios, la respuesta ya no es tan clara. Para que un agente tome la decisión de aumentar su precio, le tiene que llegar un aumento, es decir, no tienen la capacidad de actualizarse si no les “avisan”. Por eso, es técnicamente cierto que el aumento se propaga sólo en la cadena productiva del sector que es shockeado. Lo interesante del modelo es que, al no ser nosotros los que definimos esa cadena sino que generamos la topografía de la red a partir del MIP, estas cadenas productivas son muy complejas y logran capturar todas las dependencias que existen entre sectores. A partir de eso, cuando el alpha aumenta, sectores a los que le hubiese llegado un aumento casi residual en un primer momento, toman la decisión de aumentar sus precios viendo la variable macro y así comienzan a generar aumentos generalizados en toda la red, es decir, inflación.

#### 7.2.1 Preguntas a responder

- ¿Cuál es la sensibilidad del sistema respecto a variaciones de precio en nodos específicos? ¿Cuáles son los nodos que propagan de mayor forma la variación de precios?

### 7.3 Próximos pasos

- El modelo en su estado actual calcula la inflación a partir de un IPC basado en una canasta básica sin ponderar. Como primer paso sería interesante agregar ponderaciones a los sectores para lograr un mejor modelo.
- La MIP también tiene la información de cuánto de lo producido por un sector se vende al consumidor final. Con eso se puede generar un nuevo agente que represente ese sector, el del consumo final, y que tenga ciertos sectores más prioritarios que otros a la hora de consumirlo. Con eso, se podría eliminar el supuesto de la Ley de Say, agregando el consumo a las dinámicas. Eso podría dar lugar a deflación.
- Clusterizar ciertos nodos del grado en rubros, y estudiar los shocks dentro de esos micro-sectores.
- A partir de identificar los nodos en los cuales el grafo es mas vulnerable a un shock (Un shock en ellos generan significativamente más inflación que en el resto), plantear una reestructuración del grafo para hacerlo mas estable y seguro, teniendo en cuenta las restricciones pertinentes para que el grafo resultante siga siendo un esquema productivo.
- Si se logra eliminar la Ley de Say, sería interesante reemplazar los sectores por agentes individuales de aquel sector, para poder modelar dinámicas de competencia en los precios.

# Chapter 8

## Apéndices

### 8.1 Imports

```
In [2]: import init
import warnings

warnings.filterwarnings("ignore")

In [3]: import grafo
import experimento as exp
from experimento import plot_inflaciones, verEvolucion
import modelo
import matplotlib.pyplot as plt
import numpy as np
import networkx as nx
import pandas as pd
import quantecon_book_networks
import quantecon_book_networks.input_output as qbn_io
import quantecon_book_networks.plotting as qbn_plt
import quantecon_book_networks.data as qbn_data
```

### 8.2 Código Fuente

```
In [35]: exp.Experimento??

Init signature:
exp.Experimento(    grafo,    dinamica,    duracion_periodo=10,    metricas=None,    calcular_inflacion
Docstring:      <no docstring>
Source:
class Experimento: def __init__(self, grafo, dinamica, duracion_periodo = 10, metricas=None, calcular_...
File:          c:\users\augus\dev\gh\akielbowicz\tp-msscae-2024\src\experimento.py
Type:          type
Subclasses:

In [9]: # %load ../src/experimento.py
        import queue
        import networkx as nx
        import matplotlib.pyplot as plt
        import numpy as np
```

```

def ejemplo(str):
    return f"Esto es una función de ejemplo: {str}"


class Experimento:
    def __init__(
        self,
        grafo,
        dinamica,
        duracion_periodo=10,
        metricas=None,
        calcular_inflacion=None,
        alpha=0.5,
    ):
        """
        Inicializa una instancia de la clase Experimento.

        Parámetros
        -----
        grafo : DiGraph
            Grafo que representa el MIP
        dinamica : Callable
            Función con la siguiente firma : (aumento, peso_arista, inflacion, alpha)
        duracion_periodo : int, opcional
            La duración de cada período en pasos para actualizar la inflacion, por defecto 10 p
        metricas : Dict(string,Callable), opcional
            Diccionario de con metricas a computar sobre el grafo, ej, {"cantidad_nodos", lambda
        calcular_inflacion : Callable, opcional
            Función para calcular la inflación en un período, recibe dos arrays de precios. ej,
        alpha : float, opcional
            Parametro de ponderacion
            El alpha es CUANTO ve la inflación.
            Alpha == 1 es lo mismo que la dinamica 2.
            Alpha == 0 es lo mismo que la dinamica 1.
        """
        self.grafo = grafo.copy()
        self.dinamica = dinamica
        self.metricas = metricas or {}
        self._curr_step = 0
        self.metricas_evaluadas = {nombre: [] for nombre in self.metricas}
        self.metricas_evaluadas["inflacion"] = []
        self.queue = queue.Queue() # Fila de tuplas (Nodo, aumento)
        self.duracion_periodo = duracion_periodo
        self.precios_periodo_pasado = {}
        for nodo in self.grafo.nodes:
            self.precios_periodo_pasado[nodo] = self.grafo.nodes[nodo]["precio"]

        self._calc_inflacion = calcular_inflacion or (lambda actual, pasado: 0.0)
        self.inflacion = 0 # Expresada en %. (50 para 50%, 12 para 12%)
        assert 0.0 <= alpha <= 1.0
        self.alpha = alpha

    def shock(self, nodo, aumento):

```

```

precio_actual = self.grafo.nodes[nodo]["precio"]
nuevo_precio = {nodo: {"precio": (precio_actual * (1 + (aumento / 100)))}}
nx.set_node_attributes(self.grafo, nuevo_precio)
vecinos = self.grafo[nodo].items()
aumentos_a_pasar = []
for vecino in vecinos:
    aumento_vecino = self.actualizar(vecino, aumento)
    aumentos_a_pasar.append(aumento_vecino)
for i, vecino in enumerate(vecinos):
    self.queue.put((vecino[0], aumentos_a_pasar[i]))

def step(self, n=1): # aumento pasado en %.
    for _ in range(n):
        self._single_step()
        self._calcular_metricas()
        self._curr_step += 1

def actualizar(self, vecino, aumento):
    nodo_vecino = vecino[0]
    precio_actual = self.grafo.nodes[nodo_vecino]["precio"]
    peso_arista = vecino[1]["w"]
    aumento_vecino = self.dinamica(aumento, peso_arista, self.inflacion, self.alpha)
    nuevo_precio = {
        nodo_vecino: {"precio": (precio_actual * (1 + (aumento_vecino / 100)))}
    }
    nx.set_node_attributes(self.grafo, nuevo_precio)
    return aumento_vecino

def _single_step(self):
    if self._curr_step % self.duracion_periodo == 0:
        self.actualizar_inflacion()

    if not self.queue.empty():
        nodo_actual, aumento = self.queue.get()
        vecinos = self.grafo[nodo_actual].items()
        aumentos_a_pasar = []
        for vecino in vecinos:
            aumento_vecino = self.actualizar(
                vecino, aumento
            ) # Side effect actualiza ese vecino.
            aumentos_a_pasar.append(aumento_vecino)
        for i, vecino in enumerate(vecinos):
            self.queue.put((vecino[0], aumentos_a_pasar[i]))

def actualizar_inflacion(self):
    precios_período_actual = {}
    for nodo in self.grafo.nodes:
        precios_período_actual[nodo] = self.grafo.nodes[nodo]["precio"]
    self.inflacion = self._calc_inflacion(
        precios_período_actual.values(), self.precios_período_pasado.values()
    )
    self.precios_período_pasado = precios_período_actual.copy()

def _calcular_metricas(self):

```

```

        self.metricas_evaluadas["inflacion"].append(self.inflacion)
    for nombre, metrica in self.metricas.items():
        self.metricas_evaluadas[nombre].append(metrica(self.grafo))

    def __str__(self):
        return str(self.to_dict())

    def __repr__(self):
        return str(self)

    def to_dict(self):
        return {
            "id_grafo": hash(self.grafo),
            "step": self._curr_step,
            "metricas": self._metricas_evaluadas,
        }

def plot_inflaciones(inflaciones, sectores, aumento, umbral_label=0.5):
    fig, (ax, ax2) = plt.subplots(1, 2, figsize=(22, 10))
    ax.set_title(
        f"Valor de la inflación en el tiempo, a partir de un shock de {aumento}%, \n para todos")
    )
    ax.set_xlabel("Tiempo")
    ax.set_ylabel("Porcentaje de inflación")
    ax2.set_title(
        f"Variación de la inflación en el tiempo, a partir de un shock de {aumento}%, \n para todos")
    )
    ax2.set_xlabel("Tiempo")
    ax2.set_ylabel("Variación en el porcentaje de inflación")
    for i, inflacion in enumerate(inflaciones):
        tiempo = range(0, len(inflacion))
        if max(inflacion) > umbral_label:
            ax.plot(tiempo, inflacion, label=sectores[i])
            ax2.plot(tiempo[1:], np.diff(inflacion), label=sectores[i])
        else:
            ax.plot(tiempo, inflacion)
            ax2.plot(tiempo[1:], np.diff(inflacion))
    ax.legend()
    ax2.legend()
    # plt.show()
    return fig

def verEvolucion(df, alfa):
    df = df.T
    df_diff = df.diff().dropna()
    window_size = 5 # Tamaño de la ventana para la media móvil
    df["mean"] = df.mean(axis=1).rolling(window=window_size).mean()
    df["max"] = df.max(axis=1)
    df["min"] = df.min(axis=1)
    df_diff["mean_diff"] = df_diff.mean(axis=1).rolling(window=window_size).mean()
    df_diff["max_diff"] = df_diff.max(axis=1)
    df_diff["min_diff"] = df_diff.min(axis=1)

```

```

fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(15, 6))
a = len(df) // 10

# Gráfico para datos absolutos
axes[0].fill_between(
    df.index, df["min"], df["max"], color="red", alpha=0.3, label="Rango (Mín-Máx)"
)
axes[0].plot(df.index, df["mean"], color="red", linewidth=2, label="Media Móvil")
axes[0].set_title(f"Inflación Absoluta en 123 simulaciones para {alfa}")
axes[0].set_xlabel("Tiempo en cantidad de iteraciones")
axes[0].set_ylabel("Inflación absoluta")
axes[0].set_xticks(df.index[::a])
axes[0].legend()

# Gráfico para diferencias de inflación
axes[1].fill_between(
    df_diff.index,
    df_diff["min_diff"],
    df_diff["max_diff"],
    color="red",
    alpha=0.3,
    label="Rango (Mín-Máx)",
)
axes[1].plot(
    df_diff.index,
    df_diff["mean_diff"],
    color="red",
    linewidth=2,
    label="Media Móvil",
)
axes[1].set_title(f"Variación de inflación en 123 simulaciones para {alfa}")
axes[1].set_xlabel("Tiempo en cantidad de iteraciones")
axes[1].set_ylabel("Variación de Inflación")
axes[1].set_xticks(df_diff.index[::a])
axes[1].legend()

# Ajustar el layout
plt.tight_layout()

# Mostrar los gráficos
# plt.show()
return fig

```

```
In [7]: # %load ../src/modelo.py
# Definimos las dinamicas
def dinamica_1(aumento, peso_arista, inflacion=0.0, alpha=1.0):
    aumento_vecino = aumento * peso_arista
    return aumento_vecino
```

```
def dinamica_2(aumento, peso_arista, inflacion, alpha=1.0):
    aumento_vecino = inflacion
    return aumento_vecino
```

```

def dinamica_3(aumento, peso_arista, inflacion, alpha):
    aumento_vecino = (alpha * inflacion) + ((1 - alpha) * ((aumento) * peso_arista))
    return aumento_vecino

## Defino como se calcula la inflacion
def calcular_inflacion(precios_período_actual, precios_período_pasado):
    # Asumimos una inflación no ponderada
    # donde la "canasta" sea un producto de cada sector. Entonces,
    # la canasta es la sumatoria de los precios de todos los sectores,
    # y con eso calculamos IPC e inflación.
    # calculo los precios actuales

    ipc_actual = sum(precios_período_actual) / len(precios_período_actual)
    ipc_pasado = sum(precios_período_pasado) / len(precios_período_pasado)
    inflacion = ((ipc_actual / ipc_pasado) - 1) * 100
    return inflacion

# Para refactor
dinamica_local = dinamica_1
dinamica_global = dinamica_2
dinamica_mixta = dinamica_3

```

In [8]: # %load ../src/grafos.py

```

import itertools as it
import networkx as nx
import numpy as np
import matplotlib.pyplot as plt
import quantecon_book_networks.input_output as qbn_io

```

```

def armar_grafo(dataframe, precios_random=True):
    graph = {}
    for index, row in dataframe.iterrows():
        d = {}
        for k, v in row.items():
            d[k] = {"w": v}
        graph[index] = d
    G_base = nx.DiGraph(graph) # Cambio a digrafo para los plots
    G = nx.DiGraph(graph)
    for n1, n2, d in G_base.edges(data=True):
        if d["w"] == 0:
            G.remove_edge(n1, n2)

    nodos = list(G.nodes())
    precios = {}

    nuevo_precio = (
        (lambda: np.random.randint(0, 100)) if precios_random else (lambda: 100)
    )

```

```

    for nodo in nodos:
        precios[nodo] = {"precio": nuevo_precio()}

    nx.set_node_attributes(G, precios)
    return G

def plot_grafo(
    grafo,
    labels=True,
    grande=False,
    node_data=True,
):
    connectionstyle = [f"arc3,rad={r}" for r in it.accumulate([0.15] * 4)]
    tamaños = list(grafo.degree())
    for i in range(len(tamaños)):
        tamaños[i] = (
            (tamaños[i][1] ** 2) / 9 if ((tamaños[i][1] ** 2) / 9) > 100 else 100
        )
    pos = nx.kamada_kawai_layout(grafo)
    if grande:
        fig, ax = plt.subplots(figsize=(30, 30))
    else:
        fig, ax = plt.subplots(figsize=(8, 8))
    plt.axis("off")
    nx.draw_networkx_nodes(grafo, pos, node_size=tamaños, ax=ax)
    nx.draw_networkx_labels(grafo, pos, font_size=10, ax=ax)
    nx.draw_networkx_edges(
        grafo,
        pos,
        edge_color="grey",
        connectionstyle=connectionstyle,
        ax=ax,
        arrows=True,
        arrowstyle="->",
    )
    if labels:
        labels = {
            tuple(edge): f"{'w'}={round(attrs['w'],3)}"
            for *edge, attrs in grafo.edges(keys=True, data=True)
        }
        nx.draw_networkx_edge_labels(
            grafo,
            pos,
            labels,
            connectionstyle=connectionstyle,
            label_pos=0.3,
            font_color="blue",
            bbox={"alpha": 0},
            ax=ax,
        )
    labels = dict(grafo.nodes(data=True))
    for key in labels:
        labels[key] = labels[key]["precio"]

```

```

        for key in pos:
            pos[key] = pos[key] + 0.05
        nx.draw_networkx_labels(
            grafo,
            pos,
            labels,
            font_size=10,
            font_color="red",
            ax=ax,
        )
    return fig

# %%

def verGrafoCentralizado(df):
    fig, ax = plt.subplots(figsize=(40, 40))
    plt.axis("off")
    G = nx.DiGraph(df)

    centrality = nx.eigenvector_centrality(G, max_iter=1000)
    values = np.array(list(centrality.values()))
    norm_values = (values - values.min()) / (values.max() - values.min())

    colores_nodo = plt.cm.viridis(norm_values)

    grado_nodo = np.array([G.out_degree[i] for i in range(len(G))])
    tamaños_nodo = 400 + (grado_nodo * 200)
    edge_widths = qbn_io.normalise_weights(qbn_io.edge_weights(G), 10)

    node_colors = qbn_io.colorise_weights(list(centrality), beta=False)
    node_to_color = dict(zip(G.nodes, node_colors))
    edge_colors = []
    for src, _ in G.edges:
        edge_colors.append(node_to_color[src])

    pos_nodos = nx.spring_layout(G)

    nx.draw_networkx_nodes(
        G,
        pos_nodos,
        node_color=colores_nodo,
        node_size=tamaños_nodo,
        edgecolors="grey",
        linewidths=10,
        alpha=0.6,
        ax=ax,
    )

    nx.draw_networkx_edges(
        G,
        pos_nodos,
        edge_color=edge_colors,
    )

```

```

        width=edge_widths,
        arrows=True,
        arrowsize=edge_widths,
        alpha=0.6,
        ax=ax,
        arrowstyle="->",
        node_size=10,
        connectionstyle="arc3,rad=0.15",
    )
)

nx.draw_networkx_labels(G, pos_nodos, font_size=17, ax=ax, labels=None)

sm = plt.cm.ScalarMappable(
    cmap=plt.cm.viridis, norm=plt.Normalize(vmin=values.min(), vmax=values.max())
)
sm.set_array([])
cbar = plt.colorbar(sm, ax=ax)
cbar.set_label("Centralidad de autovectores", fontsize=15)
plt.title("Visualización de la matriz Insumo Producto")

plt.show()
return fig
}

def verGrafoAbierto(df):
    fig, ax = plt.subplots(figsize=(25, 25))
    plt.axis("off")
    G = nx.DiGraph()
    N = len(df)

    # Agrego nodos.
    for i, col in enumerate(df.columns):
        G.add_node(col)

    grado_nodo = np.zeros(len(df))
    for i in range(len(df)):
        aristas = 0
        for j in range(len(df)):
            if df.iloc[i, j] > 0.2:
                aristas = aristas + 1
        grado_nodo[i] = aristas
    tamaños_nodo = 400 + (grado_nodo * 200)
    edge_widths = []
    for i in range(N):
        for j in range(N):
            a = df.iloc[i, j]
            G.add_edge(df.columns[i], df.columns[j])
            width = a
            edge_widths.append(width)

    pos_nodos = nx.spring_layout(G)
    H = nx.DiGraph(df.to_numpy())
    centrality = nx.eigenvector_centrality(H, max_iter=1000)
    values = np.array(list(centrality.values()))

```

```

norm_values = (values - values.min()) / (values.max() - values.min())
colores_nodo = plt.cm.viridis(norm_values)

nx.draw_networkx_nodes(
    G,
    pos_nodos,
    node_color=colores_nodo,
    node_size=tamaños_nodo,
    edgecolors=colores_nodo,
    linewidths=10,
    alpha=0.6,
    ax=ax,
)
nx.draw_networkx_edges(
    G,
    pos_nodos,
    edge_color=colores_nodo,
    width=edge_widths,
    arrows=True,
    arrowsize=edge_widths,
    alpha=0.6,
    ax=ax,
    arrowstyle="->",
    node_size=10,
    connectionstyle="arc3,rad=0.15",
)
nx.draw_networkx_labels(G, pos_nodos, font_size=10, ax=ax)

sm = plt.cm.ScalarMappable(
    cmap=plt.cm.viridis, norm=plt.Normalize(vmin=values.min(), vmax=values.max())
)
sm.set_array([])
cbar = plt.colorbar(sm, ax=ax)
cbar.set_label("Influencia", fontsize=10)
plt.title("Grafo desenredado")

plt.show()
return fig

def verInflacion(df, precios_finales, precios_iniciales, figsize=(25, 25)):
    fig, ax = plt.subplots(figsize=figsize)
    plt.axis("off")

    G = nx.DiGraph()
    N = len(df)

    # Agrego nodos.
    for i, col in enumerate(df.columns):
        G.add_node(col)
    edge_colors = []
    edge_widths = []

```

```

for i in range(N):
    for j in range(N):
        a = df.iloc[i, j]
        G.add_edge(df.columns[i], df.columns[j])
        width = a
        edge_widths.append(width)

grado_nodo = np.array([G.out_degree[(sector)] for sector in df.columns])
price_changes = np.array(precios_finales) - np.array(precios_iniciales)

max_change = max(price_changes)
min_change = min(price_changes)
norm_changes = (price_changes - min_change) / (max_change - min_change)
tamanos_nodo = 50 + (grado_nodo * 2)
tamanos_nodo.tolist()
colors = plt.cm.Reds(norm_changes)

pos_nodos = nx.spring_layout(G)

nx.draw_networkx_nodes(
    G,
    pos_nodos,
    node_color=colors,
    node_size=tamanos_nodo,
    edgecolors=colors,
    linewidths=10,
    alpha=1,
    ax=ax,
)
nx.draw_networkx_edges(
    G,
    pos_nodos,
    edge_color="gray",
    width=edge_widths,
    arrows=True,
    arrowsize=edge_widths,
    alpha=0.6,
    ax=ax,
    arrowstyle="->",
    node_size=10,
    connectionstyle="arc3,rad=0.15",
)
nx.draw_networkx_labels(G, pos_nodos, font_size=12, ax=ax)

sm = plt.cm.ScalarMappable(
    cmap=plt.cm.Reds, norm=plt.Normalize(vmin=min_change, vmax=max_change)
)
sm.set_array([])
cbar = plt.colorbar(sm, ax=ax)
cbar.set_label("Variación de precios", fontsize=10)
plt.show()

```

```

    return fig

def graficar_dist_grados(G, columns):
    grados_in = G.in_degree
    grados_out = G.out_degree
    nodos = []
    grado_salida = []
    grado_entrada = []
    j = 0
    for sector in columns:
        nodos.append(j)
        j += 1
        grado_salida.append(grados_out[sector])
        grado_entrada.append(grados_in[sector])

    fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(15, 6))

    axes[0].bar(nodos, grado_salida, color="blue")
    axes[0].set_xlabel("Sectores Productivos (Índice)")
    axes[0].set_ylabel("Grado de salida")
    axes[0].set_title("Distribución de grado de salida")
    print(f"Media del Grado de salida:" + str(np.mean(grado_salida)))
    print(f"Desvio del Grado de salida:" + str(np.var(grado_salida)))

    axes[1].bar(nodos, grado_entrada, color="red")
    axes[1].set_xlabel("Sectores Productivos (Índice)")
    axes[1].set_ylabel("Grado de entrada")
    axes[1].set_title("Distribución de grado de entrada")

    print(f"Media del Grado de entrada:" + str(np.mean(grado_entrada)))
    print(f"Desvio del Grado de entrada:" + str(np.var(grado_entrada)))
    return fig

def graficar_dist_centralidad(G, columns):
    centrality = nx.eigenvector_centrality(G, max_iter=1000)
    values = list(centrality.values())

    centralidad = []
    nodos_nombre = []
    j = 0
    for sector in columns:
        nodos_nombre.append(sector)
        centralidad.append(values[j])
        j += 1

    fig = plt.figure(figsize=(20, 10)) # Ancho 20, Alto 10
    plt.bar(nodos_nombre, centralidad, color="yellow")
    plt.xticks(rotation=90)
    plt.title("Centralidad de Autovectores por Sector")
    plt.xlabel("Sectores")
    plt.ylabel("Centralidad de Autovectores")
    plt.tight_layout()

```

```
    return fig
```

## 8.3 Dependencias

In [1]: %pip freeze

```
anyio==4.4.0
argon2-cffi==23.1.0
argon2-cffi-bindings==21.2.0
arrow==1.3.0
asttokens==2.4.1
async-lru==2.0.4
attrs==23.2.0
Babel==2.15.0
beautifulsoup4==4.12.3
bleach==6.1.0
certifi==2024.6.2
cffi==1.16.0
charset-normalizer==3.3.2
colorama==0.4.6
comm==0.2.2
contourpy==1.2.1
cycler==0.12.1
debugpy==1.8.2
decorator==5.1.1
defusedxml==0.7.1
exceptiongroup==1.2.1
executing==2.0.1
fastjsonschema==2.20.0
fonttools==4.53.0
fqdn==1.5.1
h11==0.14.0
httpcore==1.0.5
httpx==0.27.0
idna==3.7
ipykernel==6.29.4
ipython==8.26.0
isoduration==20.11.0
jedi==0.19.1
Jinja2==3.1.4
json5==0.9.25
jsonpointer==3.0.0
jsonschema==4.22.0
jsonschema-specifications==2023.12.1
jupyter-events==0.10.0
jupyter-lsp==2.2.5
jupyter_client==8.6.2
jupyter_core==5.7.2
jupyter_server==2.14.1
jupyter_server_terminals==0.5.3
jupyterlab==4.2.3
jupyterlab_pygments==0.3.0
jupyterlab_rise==0.42.0
jupyterlab_server==2.27.2
```

```
kiwisolver==1.4.5
llvmlite==0.43.0
lxml==5.2.2
MarkupSafe==2.1.5
matplotlib==3.9.0
matplotlib-inline==0.1.7
mistune==3.0.2
mpmath==1.3.0
nbclient==0.10.0
nbconvert==7.16.4
nbformat==5.10.4
nest-asyncio==1.6.0
networkx==3.3
notebook_shim==0.2.4
numba==0.60.0
numpy==2.0.0
overrides==7.7.0
packaging==24.1
pandas==2.2.2
pandas-datareader==0.10.0
pandocfilters==1.5.1
parso==0.8.4
pillow==10.4.0
platformdirs==4.2.2
POT==0.9.4
prometheus_client==0.20.0
prompt_toolkit==3.0.47
psutil==6.0.0
pure-eval==0.2.2
pycparser==2.22
Pygments==2.18.0
pyparsing==3.1.2
python-dateutil==2.9.0.post0
python-json-logger==2.0.7
pytz==2024.1
pywin32==306
pywinpty==2.0.13
PyYAML==6.0.1
pyzmq==26.0.3
quantecon==0.7.2
quantecon_book_networks==1.1
referencing==0.35.1
requests==2.32.3
rfc3339-validator==0.1.4
rfc3986-validator==0.1.1
rpds-py==0.18.1
scipy==1.14.0
Send2Trash==1.8.3
six==1.16.0
sniffio==1.3.1
soupsieve==2.5
stack-data==0.6.3
sympy==1.12.1
terminado==0.18.1
```

```
tinycss2==1.3.0
tomli==2.0.1
tornado==6.4.1
traitlets==5.14.3
types-python-dateutil==2.9.0.20240316
typing_extensions==4.12.2
tzdata==2024.1
uri-template==1.3.0
urllib3==2.2.2
wcwidth==0.2.13
webcolors==24.6.0
webencodings==0.5.1
websocket-client==1.8.0
Note: you may need to restart the kernel to use updated packages.
```