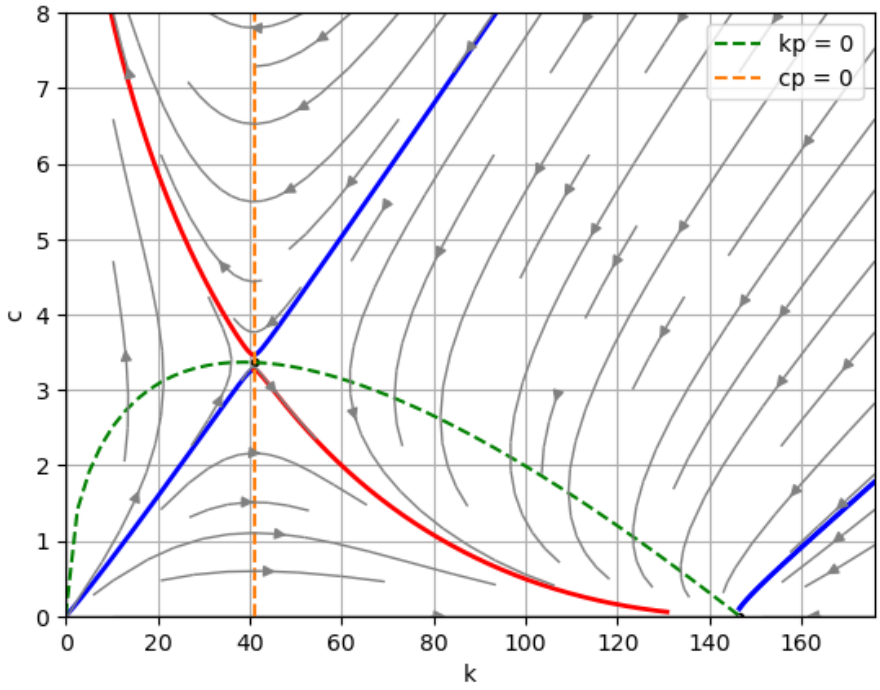
Ramsey-Cass-Koopmans

Runge Kutta 4 2D



Integrants:

Joaquin Flores Ruiz

Genís Ruiz Menárguez

Pol Abadia Conejos

Oliver Einard Tarragó Boardman

Francesco Tedesco

Índex

1 Introducció al model Ramsey-Cass-Koopmans

El model de Ramsey-Cass-Koopmans es un model econòmic d'optimització intertemporal que descriu com un agent econòmic, com un individu, família, o inclús una economia sencera, pren decisions sobre el consum i l'estalvi, tenint en compte com aquestes decisions afectaran al seu benestar futur. L'objectiu del model és trobar un equilibri entre el consum actual i el futur per a maximitzar la utilitat de l'agent al llarg del temps, subjecte a certes restriccions, com per exemple, la depreciació del capital, la preferència temporal dels consumidors o les millores tecnològiques. En aquest context, la utilitat mesura el nivell de benefici que un agent econòmic obté al consumir un bé o un servei (capital).

2 Descripció del sistema

2.1 Descripció de les variables

- $k(t)$: **Capital per capita.** Notar que en economia del creixement, el capital és qualsevol bé, material o servei, que serveix per a produir altres bens i serveis. Poden ser màquines, edificis, vehicles, software, patents, etc
- $c(t)$: **Consum per capita,** és a dir, quant de capital es consumeix per capita.

2.2 Descripció dels paràmetres

- α : **Elasticitat de la producció amb respecte al capital.** Indica quant augmenta la producció si augmentes el capital en un 1%. Per exemple, $\alpha = 0.5$ indica que si el capital augmenta en un 1%, la producció augmenta en un 0.5%. Usualment, $0 < \alpha < 1$

- δ : **Taxa de depreciació del capital** És la pèrdua de valor del capital per desgast, obsolescència, manteniment, renovació de llicències, etc., és a dir, el capital, com per exemple una màquina, un vehicle o un edifici, es degrada amb el pas del temps i de l'ús i perd valor.

- ϕ : **Taxa de creixement de la productivitat de la força de treball.** Indica quant creix la productivitat per força de treball degut a millores tecnològiques, millores en l'eficiència, etc.

- ρ : **Taxa de preferència temporal, o aversió al risc.** Expressa quant prefereixen els agents econòmics el consum present en comparació al consum futur. Un rho més alt, que seria una aversió al risc baixa, indica una major preferència pel consum present respecte del futur.

- ξ : **Taxa de creixement de la força de treball.** Indica quant creix la força de treball degut al creixement poblacional, immigració, etc.

- θ : **Paràmetre de suavització del consum** Informa quant valoren els consumidors la suavització del consum. Un valor alt indica que els consumidors prefereixen mantenir el consum constant al llarg del temps, sense daltabaixos; i un valor baix mostra que els consumidors toleren fluctuacions en el nivell de consum.

2.3 Sistema d'equacions

$$\begin{cases} \dfrac{dk}{dt} = k^{\alpha} - c - (\phi + \xi + \delta) \cdot k \\ \dfrac{dc}{dt} = c \cdot \left(\dfrac{\alpha \cdot k^{\alpha-1} - \theta - \xi - \delta}{\rho} - \phi \right) \end{cases}$$

On k^{α} és la funció d'utilitat i $\alpha \cdot k^{\alpha-1}$ la seva derivada.

2.4 Aplicacions en la industria

Aquest model és utilitzat en empresa per poder prendre decisions respecte

- **Gestió d'inversions**: Aquest model pot ser útil als negocis per analitzar les oportunitats d'inversió i també per poder gestionar el posicionament del recursos.
- **Plans d'expansió**: El model RCK pot ser utilitzat per avaluar com de factible és un pla d'expansió considerant factors com Elasticitat de la producció amb respecte al capital, Taxa de depreciació del capital...
- **Activitats de Recerca i Desenvolupament (R&D)**: Les companyies poden utilitzar aquest model per avaluar quins riscos i quins beneficis pot tenir un determinat projecte relacionat amb R&D.

Com que es tracta d'un model teòric, per poder aplicar els resultat hem de fer una anàlisi de la situació avançada, ja que hem de tenir en compte:

- **La simplificació per part del model**: Aquest model assumeix el comportament de diversos factors per poder fer la predicció, és per això que hem de tenir clar en quin context industrial estem aplicant el model.
- **Possibles fenòmens no considerats pel model** : Com acabem de mencionar, aquest model no té perquè considerar tots el fenòmens relacionats amb el àmbit industrial en concret, és per això que hem de fer una anàlisi més acurada per poder determinar el context en el qual volem aplicar el model, per evitar que els factors no considerats pel model no acabin espatllant les prediccions.

3 Resolució del sistema

3.1 Per què Runge Kutta 4 2D?

Resoldre analíticament un sistema d'equacions diferencials ordinàries no lineals com aquest pot ser molt complicat, inclús impossible. Per això es fan servir mètodes numèrics per a la seva resolució. En el nostre cas utilitzarem el Runge-Kutta 4 en 2D, un mètode utilitzat en la indústria per les següents raons:

- **Precisió**: Es tracta d'un mètode que pot aconseguir una major precisió degut a les diferents avaluacions del pendent per a poder aproximar el següent pas
- **Simplicitat**: És un mètode fàcilment implementable i ofereix una considerable eficiència, en el nostre cas ho farem en Python.
- **Versatilitat**: Aquest mètode pot ser utilitzat per a resoldre un nombre ampli d'equacions diferencials ordinàries, fàcilment aplicable a sistemes de 2 dimensions.

3.2 Implementació Runge-Kutta 4 2D en Python

```
# Función de resolución del sistema de ecuaciones diferenciales
def runge_kutta_2D(xp, yp, x0, y0, t0, tf, h):
    N = int((tf - t0) / abs(h))
    t = np.linspace(t0, tf, N+1)
    x = np.zeros(N+1)
    y = np.zeros(N+1)
    x[0] = x0
    y[0] = y0
    for i in range(N):
```

```
        # Cálculo de las pendientes en los puntos intermedios
        k1 = xp(x[i], y[i])
        l1 = yp(x[i], y[i])
        k2 = xp(x[i] + h * k1 / 2, y[i] + h * l1 / 2)
        l2 = yp(x[i] + h * k1 / 2, y[i] + h * l1 / 2)
        k3 = xp(x[i] + h * k2 / 2, y[i] + h * l2 / 2)
        l3 = yp(x[i] + h * k2 / 2, y[i] + h * l2 / 2)
        k4 = xp(x[i] + h * k3, y[i] + h * l3)
        l4 = yp(x[i] + h * k3, y[i] + h * l3)
        # Cálculo de los nuevos valores de x e y utilizando el método
        # de Runge-Kutta de cuarto orden
        x[i+1] = x[i] + h * (k1 + 2 * k2 + 2 * k3 + k4) / 6
        y[i+1] = y[i] + h * (l1 + 2 * l2 + 2 * l3 + l4) / 6
    return t, x, y
```

4 Exemples i interpretació

En primer lloc, s'han graficat dues funcions clau. $\frac{dk}{dt} = 0$, anotada com a kp i corresponent a la situació on el capital k és estable, i $\frac{dc}{dt} = 0$, recta vertical cp corresponent als valors que estabilitzen el consum. D'aquesta manera, considerarem com a punt estacionari de gran interès analític la intersecció entre la corba kp i la recta cp. No obstant, s'ha de tenir en compte la presència de punts estacionaris amb un sentit econòmic nul. Aquest cas seria el del punt on tendeix la corba kp, el qual tindrà un valor k=140 però amb un consum c=0. Estimant els paràmetres, en concret α, aquest mateix punt estacionari mateix un desplaçament lateral, però roman amb absència de sentit interpretatiu.

La finalitat del Runge-Kutta 4 2D en aquest cas és trobar la **trajectòria estable**, és a dir la trajectòria que per cada nivell de capital ens determina el corresponent nivell de consum per poder convergir a un punt estacionari. Per tant, gràcies al Runge-Kutta 4 2D, serem capaços de determinar una una quantitat de capital i consum inicials (condició d'estabilitat) respecte dels paràmetres que descriuen el context. És important situar-nos sobre la trajectòria estable, ja que en qualsevol altre punt, a llarg termini aquestes trajectòries acabaran en dues possibles situacions. En la primera situació, el capital desapareixerà ja que haurem consumit massa (capital tendint a infinit), en l'altre cas, el consum baixarà a 0, la qual cosa voldrà dir que haurem invertit massa (consum tendint a infinit).

En les següents tres gràfiques, s'ha fixat els paràmetres α, δ, ϕ i ξ i estudiat com afecten ρ i θ al model. S'ha representat les separatrius on la direcció del camp apunta cap a un equilibri de color blau i les altres de vermell, les isoclines de camp vertical de color verd i les de camp horitzontal taronja.

α : 0.55

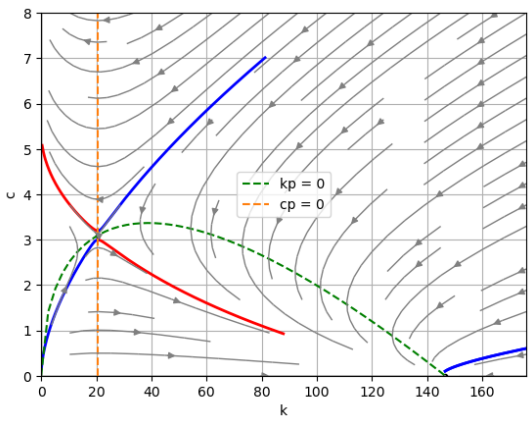
δ : 0.08

ϕ : 0.025

ρ : 2

ξ : 0.001

θ : 0.01



En la primera gràfica, s'ha pres una ρ=2, que implica una aversió al risc baixa, és a dir, una preferència pel consum present respecte el futur. Aquest tret és notable ja que la trajectòria que s'acosta al punt d'equilibri determinat per *kp* i *cp* és força pronunciada, fet que implica un gran consum c per a un capital k.

α : 0.55

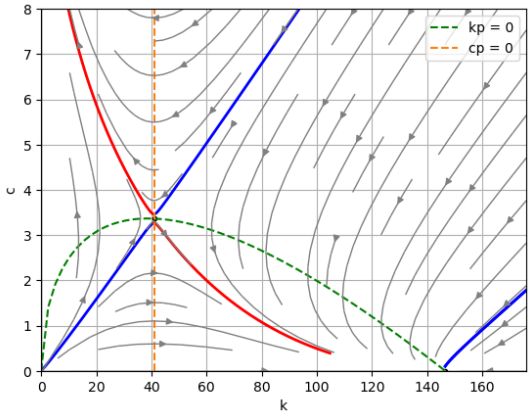
δ : 0.08

ϕ : 0.025

ρ : 0.5

ξ : 0.001

θ : 0.01



En la segona gràfica s'ha fet servir una $\rho = 0.5$, que indica una aversió al risc més elevada que en l'anterior cas, i això es tradueix en una preferència pel consum futur respecte el present. Gràficament això s'aprecia amb la trajectòria que porta a l'equilibri, que té un pendent menys pronunciat que en l'anterior cas. D'aquesta manera, la trajectòria per arribar al mateix nivell de consum que l'anterior és més llarga.

α : 0.55

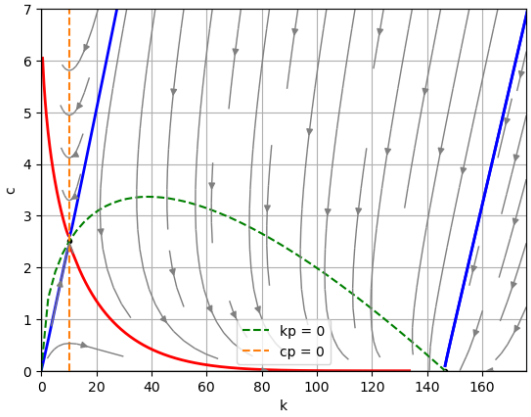
δ : 0.08

ϕ : 0.025

ρ : 0.5

ξ : 0.001

θ : 0.1



En la tercera gràfica, s'ha volgut estudiar l'impacte de la variable θ assignant θ=0.1. Aquest paràmetre representa la suavització del consum. Al incrementar per 10 el valor de θ respecte la gràfica anterior, aquest model correspon a un cas on el consum es manté més constant al llarg del temps, sense cap daltabaix. Conseqüentment, tenir un consum estable implica una variació baixa del capital. En aquest cas, veiem com la trajectòria òptima blava pràcticament sobreposa la corba *kp*, que marca l'estabilització del capital.

5 Conclusions

Per a recapitular:

- El mètode **Runge-Kutta 4 2D** ens permet determinar numèricament la **condició d'estabilitat**, és a dir el consum inicial respecte el capital inicial i el paràmetres que defineixen el context.
- Es tracta d'un mètode numèric **simple**, però que a la vegada ens proporciona la **precisió suficient** per fer una anàlisi amb prou seguretat.
- Gràcies a la seva **simplicitat** i als resultats als quals es poden arribar utiltizant RK4 2D per resoldre les equacions d'aquest model, trobem coherent que sigui utilitzat en la **industria**.

6 Apèndix

```
import numpy as np
import matplotlib.pyplot as plt
from sympy import Eq, solve, Matrix, lambdify, symbols
from scipy.optimize import fsolve


k, c = symbols('k c')
ph, x, d, th, r, a = symbols('ph x d th r a')


def initialize_system():
    kps = k**a - c - k * (ph + x + d)
    cps = (c *(a*k**(a-1) - th - x - d - r * ph)) / r
```

```
    return kps, cps


def get_system(kps, cps, alpha, phi, xi, delta, theta, rho):
    ka = kps.subs([(a,alpha),(ph,phi),(x,xi),(d,delta),(th,theta),(r,rho)])
    ca = cps.subs([(a,alpha),(ph,phi),(x,xi),(d,delta),(th,theta),(r,rho)])
    return ka, ca


def lambdify_system(ka, ca):
    kp = lambdify((k, c), ka)
    cp = lambdify((k, c), ca)
    return kp, cp


def get_partial_derivatives(ka, ca):
    dkpdk = lambdify((k, c), ka.diff(k))
    dcpdk = lambdify((k, c), ca.diff(k))
    dcpdc = lambdify((k, c), ca.diff(c))
    dkpdc = lambdify((k, c), ka.diff(c))
    return dkpdk, dcpdk, dcpdc, dkpdc


def get_equilibrium_points(kp, cp, kps, cps,  alpha, phi, xi, delta, theta, rho):
    if alpha == 0.5:
        alpha += 0.1
        ka, ca = get_system(kps, cps, alpha, phi, xi, delta, theta, rho)
        kp, cp = lambdify_system(ka, ca)
        [eq1, eq2] = [Eq(kp(k,c),0), Eq(cp(k,c),0)]
```

```
        initial_guesses = solve((eq1, eq2), (k, c))
        solutions = []
        for guess in initial_guesses:
            sol = fsolve([eq1, eq2], guess)
            solutions.append(sol)
    else:
        [eq1, eq2] = [Eq(kp(k,c),0), Eq(cp(k,c),0)]
        solutions = solve((eq1, eq2), (k, c))
    return solutions


def plot_orbits(kp, cp, ax):
    _, k1, c1 = runge_kutta_2D(xp = kp, yp = cp, x0=100, y0=2, t0=0, tf=40, h=0.1)
    _, k2, c2 = runge_kutta_2D(xp = kp, yp = cp, x0=30, y0=1, t0=0, tf=40, h=0.1)
    _, k3, c3 = runge_kutta_2D(xp = kp, yp = cp, x0=15, y0=0.5, t0=0, tf=40, h=0.1)
    _, k4, c4 = runge_kutta_2D(xp = kp, yp = cp, x0=5, y0=0., t0=0, tf=40, h=0.1)
    _, k5, c5 = runge_kutta_2D(xp = kp, yp = cp, x0=20, y0=5, t0=0, tf=40, h=0.1)
    _, k6, c6 = runge_kutta_2D(xp = kp, yp = cp, x0=100, y0=6, t0=0, tf=40, h=0.1)
    _, k7, c7 = runge_kutta_2D(xp = kp, yp = cp, x0=200, y0=4, t0=0, tf=40, h=0.1)
    _, k8, c8 = runge_kutta_2D(xp = kp, yp = cp, x0=270, y0=4, t0=0, tf=40, h=0.1)
    _, k9, c9 = runge_kutta_2D(xp = kp, yp = cp, x0=300, y0=4, t0=0, tf=40, h=0.1)
    _, k10, c10 = runge_kutta_2D(xp = kp, yp = cp, x0=50, y0=6, t0=0, tf=40, h=0.1)


    ax.plot(k1, c1, 'c', lw=1)
    ax.plot(k2, c2, 'c', lw=1)
    ax.plot(k3, c3, 'c', lw=1)
```

```
    ax.plot(k4, c4, 'c', lw=2)
    ax.plot(k5, c5, 'c', lw=1)
    ax.plot(k6, c6, 'c', lw=1)
    ax.plot(k7, c7, 'c', lw=1)
    ax.plot(k8, c8, 'c', lw=1)
    ax.plot(k9, c9, 'c', lw=1)
    ax.plot(k10, c10, 'c', lw=1)

def plot_isoclines(kp, cp, ax, xmax, ymax):
    sep1 = lambdify(k, solve(kp(k,c) , c)[0])
    sep2 = lambdify(c, solve(cp(k,c) , k)[0])
    ax.axhline(0, lw=1, color='k')
    ax.axvline(0, lw=1, color='k')
    ax.grid()
    x1 = np.linspace(0, 400 , int(xmax + 10))
    y2 = np.linspace(0, 100 , int(ymax + 5))
    x2 = np.full(int(ymax + 5) , sep2(c))
    ax.plot(x1,sep1(x1), 'g--',  label = 'kp = 0')
    ax.plot(x2,y2, 'C1--',  label = 'cp = 0')

def main_plot(kp, cp, interval, eq_pts, dkpdk, dcpdk, dcpdc, dkpdc, h):
    col = ['r--', 'b--']
    xmax, ymax = np.max(eq_pts, axis = 0)
    fig, ax = plt.subplots()
    ax.set_xlabel('k')
```

```
    ax.set_ylabel('c')
    for p in eq_pts:
        Ap= Matrix([[dkpdk(p[0],p[1]), dcpdk(p[0],p[1])],
        [dkpdc(p[0],p[1]), dcpdc(p[0],p[1])]])
        for i , (eiva, _,  eigenvector) in enumerate(Ap.eigenvects()):
            if eiva > 0:
                col = "red"
                sign = 1
            else:
                col = "blue"
                sign =- 1
            _, pk1, pc1 = runge_kutta_2D(xp = kp, yp = cp,
                                         x0=p[0] + h*eigenvector[0][0],
                                         y0=p[1] + h*eigenvector[0][1],
                                         t0=0, tf=75, h=sign*0.1)
            _, nk1, nc1 = runge_kutta_2D(xp = kp, yp = cp,
                                x0=p[0] + (-h)*eigenvector[0][0],
                                y0=p[1] + (-h)*eigenvector[0][1],
                                t0=0, tf=75, h=sign*0.1)


            ax.plot(pk1, pc1, color = col, lw=2)
            ax.plot(nk1, nc1, color = col, lw=2)
```

```
        ax.plot(p[0], p[1], 'ko', markersize=3)

    # plot_orbits(kp, cp, ax)
    plot_isoclines(kp, cp, ax, xmax, ymax)



    k_vals = np.linspace(0, int(xmax + 30), 100)
    c_vals = np.linspace(0,int(ymax + 5), 100)
    K, C = np.meshgrid(k_vals, c_vals)
    U = kp(K, C)
    V = cp(K, C)

    ax.streamplot(K, C, U, V, density=0.6, color='gray', linewidth=1)
    ax.set_xlim(0, int(xmax + 30))
    ax.set_ylim(0, int(ymax + 5))
    ax.legend()


alpha = 0.55
phi = 0.025
xi = 0.001
delta = 0.08
theta = 0.1
rho = 0.5
interval = [[-100, 100], [0, 0.000005]]
h = 0.1
```

```
kps, cps = initialize_system()

ka, ca = get_system(kps, cps, alpha, phi, xi, delta, theta, rho)

kp, cp = lambdify_system(ka, ca)

dkpdk, dcpdk, dcpdc, dkpdc = get_partial_derivatives(ka, ca)

eq_pts = get_equilibrium_points(kp, cp, kps, cps,  alpha, phi, xi, delta, theta, rho)


main_plot(kp, cp, interval, eq_pts, dkpdk, dcpdk, dcpdc, dkpdc, h)
```