Семинар 12

1. Модель Касса-Купмана

Модель Касса-Купмана является расширением модели Солоу, в которой норма сбережения является эндогенной переменной. Мы рассмотрим два варианта модели:

1. модель плановой экономики,
2. модель конкурентной экономики.

1.1. Плановая экономика

В плановой экономике нет

- цен,
- бюджетных ограничений.

Планировщик решает

- что производить,
- сколько инвестировать,
- кому и сколько потреблять.

Время дискретно: $t = 0, \dots, T$. T конечно.

Единственный товар используется и для потребления, и для инвестиций в капитал. Капитал может использоваться несколько периодов, но каждый период он уменьшается с нормой выбытия δ. Потребляемая часть товара не переходит в следующий период.

Пусть

$$\vec{C} = \{C_0, C_1, \dots, C_T\}$$

$$\vec{K} = \{K_0, K_1, \dots, K_T, K_{T+1}\}$$

Пусть **репрезентативный** потребитель в каждый период обладает единицей труда, а потребление для него является благом. Функция полезности имеет вид

$$U(\vec{C}) = \sum_{t=0}^{T} \beta^t u(C_t) = \sum_{t=0}^{T} \beta^t \frac{C_t^{1-\gamma}}{1-\gamma} \tag{1}$$

$$\beta \in (0, 1)$$

$$\gamma > 0$$

Заметим, что $u'(C) > 0$, $u''(C) < 0$.

Пусть $K_0 > 0$ — некоторый экзогенно заданный начальный уровень капитала. Производственная функция имеет вид

$$\begin{gathered} F(K_t, L_t) = AK_t^{\alpha} L_t^{1-\alpha} \\ A > 0 \\ \alpha \in (0,1) \\ L_t = 1, \ \forall t \end{gathered}$$

где L_t — неэластичное предложение труда; домохозяйство предлагает весь располагаемый труд.

Допустимое распределение потребление и капитала должно удовлетворять следующему условию

$$C_t + K_{t+1} \leqslant F(K_t, N_t) + (1-\delta)K_t, \ \forall t \tag{2}$$

Планировщик решает задачу максимизации (1) при ограничении (2). Для этого он формирует Лагранжиан вида

$$\mathscr{L}(\vec{C}, \vec{K}, \vec{\mu}) = \sum_{t=0}^{T} \beta^t \left\{ u(C_t) + \mu_t \left[F(K_t, 1) + (1-\delta)K_t - C_t - K_{t+1} \right] \right\} \tag{3}$$

и находит следующий экстремум

$$\min_{\vec{\mu}} \max_{\vec{C}, \vec{K}} \mathscr{L}(\vec{C}, \vec{K}, \vec{\mu})$$

для удобства введем производственную функцию на душу населения

$$\begin{aligned} \frac{F(K_t, N_t)}{N_t} &= A \left(\frac{K_t}{N_t} \right)^{\alpha} \Bigg|_{N_t=1} = AK_t^{\alpha} = f(K_t) \\ \frac{\partial F(K_t, N_t)}{\partial K_t} &= \frac{\partial N_t f\left(\frac{K_t}{N_t}\right)}{\partial K_t} = N_t f'\left(\frac{K_t}{N_t}\right) \frac{1}{N_t} \Bigg|_{N_t=1} = f'(K_t) \\ \frac{\partial F(K_t, N_t)}{\partial N_t} &= \frac{\partial N_t f\left(\frac{K_t}{N_t}\right)}{\partial N_t} = \cdots = f(K_t) - f'(K_t)K_t \end{aligned}$$

Выпишем условия первого порядка для Лагранжиана (3)

$$C_t: \quad u'(C_t) - \mu_t = 0 \tag{4}$$

$$K_t: \quad \beta\mu_t \left[(1-\delta) + f'(K_t) \right] - \mu_{t+1} = 0 \tag{5}$$

$$\mu_t: \quad F(K_t, 1) + (1-\delta)K_t - C_t - K_{t+1} = 0 \tag{6}$$

$$K_{T+1}: \quad -\mu_T \begin{cases} \leqslant 0 & K_{T+1} = 0 \\ = 0 & K_{T+1} > 0 \end{cases} \tag{7}$$

Ограничение (7) является следствием условия Куна-Таккера $\mu_T K_{T+1} = 0$. Из (4) и (5) следует

$$\beta u'(C_t) \left[(1-\delta) + f'(K_t) \right] = u'(C_{t-1}), \ \forall t = 1, \ldots, T+1 \tag{8}$$

Сдвинув (8) на 1 период и применив функцию, обратную к u', получим

$$C_{t+1} = u'^{-1}\left[\left(\frac{\beta}{u'(C_t)}\left[(1-\delta) + f'(K_{t+1})\right]\right)^{-1}\right]$$

Если учесть вид функции потребления, то получим

$$C_{t+1} = \left\{\beta C_t^{\gamma}\left[f'(K_{t+1}) + (1-\delta)\right]\right\}^{1/\gamma}$$
$$K_{t+1} = F(K_t, 1) + (1-\delta)K_t - C_t$$

систему нелинейных разностных уравнений первого порядка, которой должны удовлетворять оптимальные последовательности $\vec{C}$ и $\vec{K}$. Они также должны удовлетворять начальным условиям K_0 и $K_{T+1} = 0$.

1.2. Алгоритм для плановой экономики

Если бы мы имели некоторое начальное значение μ_0, то мы могли бы реализовать следующий алгоритм:

1. Возьмем начальные значения μ_0 и K_0.
2. Рассчитаем
   - C_0 при помощи уравнения (4);
   - K_1 при помощи уравнения (6);
   - μ_1 при помощи уравнения (5).
3. Будем повторять данную процедуру до тех пор, пока мы не рассчитаем все значения $\vec{C}$, $\vec{K}$ и $\vec{\mu}$.

Но тут мы столкнемся с двумя проблемами. Во-первых, мы не уверены, что в итоге условие Куна-Таккера $\mu_T K_{T+1} = 0$ будет выполнено. Во-вторых, мы не знаем μ_0.

Для решения этих проблем применим **алгоритм стрельбы** совместно с алгоритмом **бисекции**:

1. Выберем некоторое предположение относительно C_0. Также выберем минимальную и максимальную границы для C_0. Отметим, что μ_0 можно заменить на C_0, так как они взаимно однозначно связаны уравнением (4).
2. Выполним описанный выше алгоритм.
3. Если
   - $K_{T+1} = 0$, то мы нашли решение;
   - $K_{T+1} > 0$, то сдвинем **нижнюю** границу в C_0: мы начали со слишком маленького потребления и не потребили весь капитал;
   - $K_{T+1} < 0$, то сдвинем **верхнюю** границу в C_0: мы начали со слишком высокого потребления и потребили больше, чем у нас было.
4. Если на прошлом шаге решение не было найдено, то выберем новое значение C_0 как центр обновленного диапазона.

1.3. Конкурентная экономика

Рассмотрим случай экономики совершенной конкуренции. В данной экономике нет планировщика, зато есть не только потребители, но и фирмы. Как и ранее, будем рассматривать **репрезентативного** потребителя (домохозяйство) и фирму.

В каждый момент времени потребитель получает набор цен (w_t, η_t) — ставки заработной платы и аренды капитала, которые определяют доход потребителя. Потребитель может сберегать путем покупки капитала или заявок на повышенное потребление в будущих периодах. Потребитель предоставляет фирме свой труд и капитал, потребляет и инвестирует.

Фирма стремится максимизировать свою прибыль, используя технологию, аналогичную таковой в плановой экономике, и арендуя труд и капитал потребителя.

Фирма и потребитель не влияют на цены.

В экономике имеется последовательность цен $\{w_t, \eta_t, q_t^0\}_{t=0}^T$, где q_t^0 — межвременная цена потребления (цена Хикса-Эрроу), равная цене потребления в момент t, измеренная в момент 0. w_t и η_t измеряются в единицах товара периода t за единицу труда и капитала, соответственно. q_t^0 измеряется в единицах товара периода t за единицу товара периода 0.

1.3.1. Задача фирмы

В момент t фирма получает прибыль

$$F(\tilde{k}_t, \tilde{n}_t) - w_t \tilde{n}_t - \eta_t \tilde{k}_t$$

Условия получения нулевой прибыли

$$\begin{aligned} \frac{\partial F}{\partial k_t} &= \eta_t \\ \frac{\partial F}{\partial n_t} &= w_t \end{aligned} \tag{9}$$

1.3.2. Задача потребителя

В период t домохозяйство сдает в аренду свой имеющийся труд (1) и капитал (k_t), получая доход

$$w_t 1 + \eta_t k_t$$

который распределеяется между потреблением c_t и инвестициями $k_{t+1} - (1-\delta)k_t$.

Потребитель может суммарно потреблять и инвестировать больше, чем его доход. Обозначим разрыв как

$$e_t := [c_t + k_{t+1} - (1-\delta)k_t] - [w_t 1 + \eta_t k_t]$$

Тогда бюджетное ограничение потребителя равно

$$\sum_{t=0}^{T} q_t^0 e_t \leqslant 0$$

или

$$\sum_{t=0}^{T} q_t^0 \left[c_t + k_{t+1} - (1-\delta)k_t\right] \leqslant \sum_{t=0}^{T} q_t^0 \left[w_t 1 + \eta_t k_t\right]$$

Итоговая задача потребителя примет вид

$$\begin{aligned} &\max_{\vec{c},\vec{k}} \sum_{t=0}^{T} \beta^t u(c_t) \\ &\sum_{t=0}^{T} q_t^0 \left[c_t + k_{t+1} - (1-\delta)k_t - w_t - \eta_t k_t\right] \leqslant 0 \end{aligned} \tag{10}$$

1.3.3. Поиск решения

В поиске решения нам поможет то, что все агенты модели получают цены извне. Это позволяет применить трюк «big K - small k», то есть при поиске решения использовать цены как функции от параметров репрезентативного потребителя и фирмы, полагая их **независимыми** от индивидуальных параметров.

Из (9) и (10) мы получим

$$q_t^0 = \beta^t u'(C_t) \tag{11}$$
$$w_t = f(K_t) - K_t f'(K_t) \tag{12}$$
$$\eta_t = f'(K_t) \tag{13}$$

Так как эти предположения выполнены для **репрезентативных** потребителя и фирмы, то определяемые ими цены принимаются **всеми** агентами.

Равновесие характеризуется равенством предложения ресурсов домохозяйствами и спроса на них со стороны фирм

$$\begin{aligned} k_t^*(\vec{q},\vec{w},\vec{\eta}) &= \tilde{k}_t^*(\vec{q},\vec{w},\vec{\eta}) \\ 1 &= \tilde{n}_t^*(\vec{q},\vec{w},\vec{\eta}) \\ c_t^* + k_{t+1}^* - (1-\delta)k_t^* &= F(\tilde{k}_t^*, \tilde{n}_t^*) \end{aligned}$$

Рассмотрим лагранжиан потребителя

$$\mathscr{L}(\vec{c},\vec{k},\lambda) = \sum_{t=0}^{T} \beta^t u(c_t) + \lambda \left\{ \sum_{t=0}^{T} q_t^0 \left[(1-\delta)k_t + w_t + \eta_t k_t - c_t - k_{t-1}\right] \right\}$$

и задачу его оптимизации

$$\min_{\lambda} \max_{\vec{c},\vec{k}} \mathscr{L}(\vec{c},\vec{k},\lambda)$$

Получим условия первого порядка

$$c_t: \quad \beta^t u'(c_t) - \lambda q_t^0 = 0,\ t = 0,1,\dots,T \tag{14}$$
$$k_t: \quad -\lambda q_t^0 \left[(1-\delta) + \eta_t\right] + \lambda q_{t-1}^0 = 0,\ t = 1,\dots,T+1 \tag{15}$$
$$\lambda: \quad \sum_{t=0}^{T} q_t^0 \left[(1-\delta)k_t + w_t + \eta_t k_t - c_t - k_{t-1}\right] \leqslant 0 \tag{16}$$
$$k_{T+1}: \quad -\lambda q_{T+1}^0 \begin{cases} \leqslant 0 & k_{T+1} = 0 \\ = 0 & k_{T+1} > 0 \end{cases} \tag{17}$$
$$\tag{18}$$

- Комбинация (11) и (14) дает нам (4).
- Комбинация (11), (13) и (15) дает нам (5).
- Комбинация (11), (12), (13) и (16) дает нам (6).
- Комбинация (11) и (17) дает нам (7).

Для фирмы все еще проще: достаточно подставить условие равенства спроса и предложения в условия нулевой прибыли. Тем самым мы получим предположения цен для репрезентативных агентов.

Другими словами, оптимальное распределение потребления и капитала в конкурентной задаче равно таковому в задаче плановой экономики.

2. Эксперимент

```
%reset -f
```

```
from numba import njit, float64
from numba.experimental import jitclass
import numpy as np

import matplotlib.pyplot as plt

plt.rcParams["figure.figsize"] = (11, 5)
```

```
planning_data = [
    ("gamma", float64),
    ("beta", float64),
    ("delta", float64),
    ("alpha", float64),
    ("A", float64),
]
```

```
@jitclass(planning_data)
class PlanningProblem:
    def __init__(self, gamma=2, beta=0.95, delta=0.02, alpha=0.
  ↪33, A=1):
        self.gamma, self.beta = gamma, beta
        self.delta, self.alpha, self.A = delta, alpha, A

    def u(self, c):
        gamma = self.gamma
        return c ** (1 - gamma) / (1 - gamma) if gamma != 1 else␣
  ↪np.log(c)

    def u_prime(self, c):
        "Derivative of utility"
        gamma = self.gamma
        return c ** (-gamma)
```

```
    def u_prime_inv(self, c):
        "Inverse of derivative of utility"
        gamma = self.gamma
        return c ** (-1 / gamma)

    def f(self, k):
        "Production function"
        alpha, A = self.alpha, self.A
        return A * k**alpha

    def f_prime(self, k):
        "Derivative of production function"
        alpha, A = self.alpha, self.A
        return alpha * A * k ** (alpha - 1)

    def f_prime_inv(self, k):
        "Inverse of derivative of production function"
        alpha, A = self.alpha, self.A
        return (k / (A * alpha)) ** (1 / (alpha - 1))

    def next_k_c(self, k, c):
        beta, delta = self.beta, self.delta
        u_prime, u_prime_inv = self.u_prime, self.u_prime_inv
        f, f_prime = self.f, self.f_prime

        k_next = f(k) + (1 - delta) * k - c
        c_next = u_prime_inv(u_prime(c) / (beta *␣
↪(f_prime(k_next) + (1 - delta))))

        return k_next, c_next
```

```
pp = PlanningProblem()
```

2.1. Функция для «стрельбы»

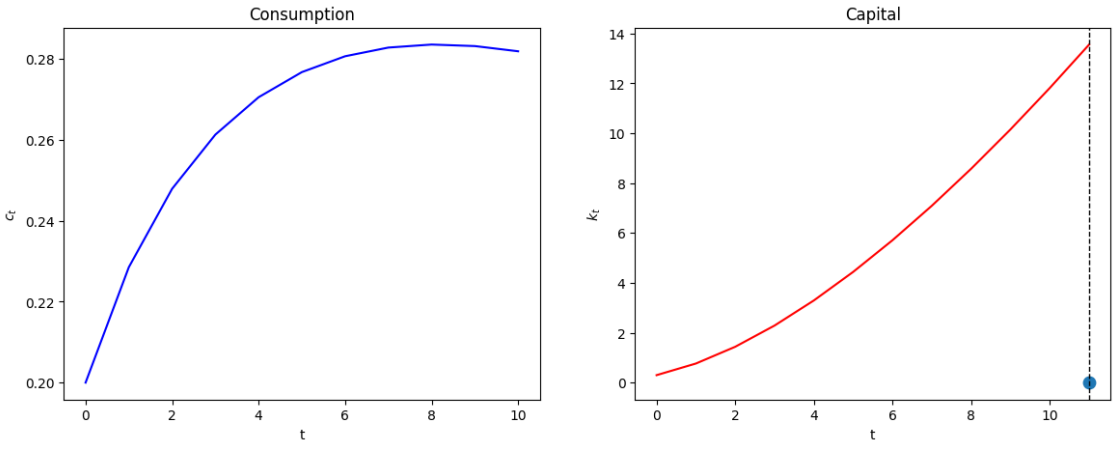
```
@njit
def shoot(pp, c0, k0, T=10):
    if c0 > pp.f(k0):
        print("initial consumption is not feasible")
        return None

    # initialize vectors of c and k
    c_vec = np.empty(T + 1)
    k_vec = np.empty(T + 2)

    c_vec[0] = c0
    k_vec[0] = k0

    for t in range(T):
```

```
        k_vec[t + 1], c_vec[t + 1] = pp.next_k_c(k_vec[t],␣
↪c_vec[t])

    k_vec[T + 1] = pp.f(k_vec[T]) + (1 - pp.delta) * k_vec[T] -␣
↪c_vec[T]

    return c_vec, k_vec
```

Сделаем неверное предположение.

```
paths = shoot(pp, 0.2, 0.3, T=10)
```

```
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

colors = ["blue", "red"]
titles = ["Consumption", "Capital"]
ylabels = ["$c_t$", "$k_t$"]

T = paths[0].size - 1

for i in range(2):
    axes[i].plot(paths[i], c=colors[i])
    axes[i].set(xlabel="t", ylabel=ylabels[i], title=titles[i])

axes[1].scatter(T + 1, 0, s=80)
axes[1].axvline(T + 1, color="k", ls="--", lw=1)

plt.show()
```



2.2. Функция для бисекции

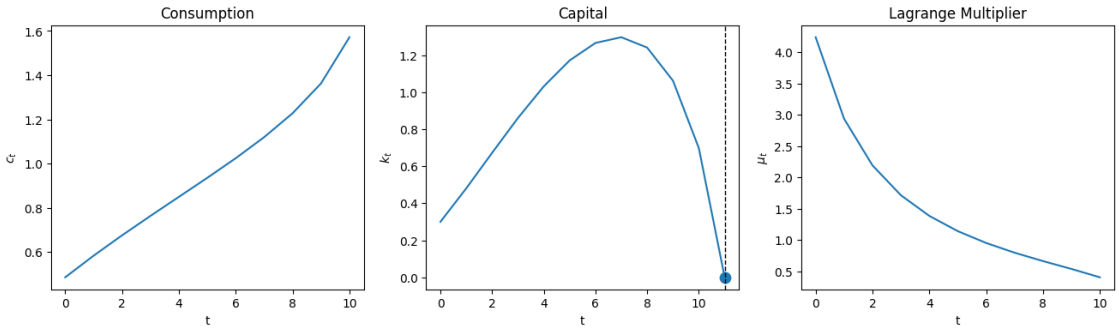
```
@njit
def bisection(pp, c0, k0, T=10, tol=1e-4, max_iter=500,␣
  ↪k_ter=0, verbose=True):
    # initial boundaries for guess c0
    c0_upper = pp.f(k0)
    c0_lower = 0

    for i in range(max_iter):
        c_vec, k_vec = shoot(pp, c0, k0, T)
        error = k_vec[-1] - k_ter

        # Проверяем попадание в цель (с некоторой точностью)
        if np.abs(error) < tol:
            if verbose:
                print("Converged successfully on iteration ", i␣
  ↪+ 1)
            return c_vec, k_vec

        # Если мы не попали в цель, то обновляем границы␣
  ↪интервала для C0
        if error > 0:
            c0_lower = c0
        else:
            c0_upper = c0

        c0 = (c0_lower + c0_upper) / 2

    if verbose:
        print("Convergence failed.")
    return c_vec, k_vec
```

2.3. Функция для изображения траекторий потребления, капитала и множителя Лагранжа

```
def plot_paths(pp, c0, k0, T_arr: list[int], k_ter=0,␣
  ↪k_ss=None, axes=None):
    if axes is None:
        _, axes = plt.subplots(1, 3, figsize=(16, 4))

    ylabels = ["$c_t$", "$k_t$", r"$\mu_t$"]
    titles = ["Consumption", "Capital", "Lagrange Multiplier"]

    c_paths = []
    k_paths = []

    for T in T_arr:
```

```
        c_vec, k_vec = bisection(pp, c0, k0, T, k_ter=k_ter,␣
↪verbose=False)

        c_paths.append(c_vec)
        k_paths.append(k_vec)

        mu_vec = pp.u_prime(c_vec)
        paths = [c_vec, k_vec, mu_vec]

        for i in range(3):
            axes[i].plot(paths[i])
            axes[i].set(xlabel="t", ylabel=ylabels[i],␣
↪title=titles[i])

        # Если мы передали в функцию стационарное состояние␣
↪капитала,
        # то отобразим его
        if k_ss is not None:
            axes[1].axhline(k_ss, c="k", ls="--", lw=1)

        axes[1].axvline(T + 1, c="k", ls="--", lw=1)
        axes[1].scatter(T + 1, paths[1][-1], s=80)

    return c_paths, k_paths
```

```
_ = plot_paths(pp, 0.3, 0.3, [10]);
```



2.4. Стационарное состояние

Если подставить **постоянные** значения капитала и потребления в (8), то можно получить

$$1 = \beta \frac{u'(\bar{C})}{u'(\bar{C})} \left[f'(\bar{K}) + (1 - \delta) \right]$$

Положим $\beta := \frac{1}{1+\rho}$ и получим, что

$$1 + \rho = f'(\bar{K}) + (1 - \delta)$$

$$\bar{K} = f'^{-1}(\rho + \delta)$$

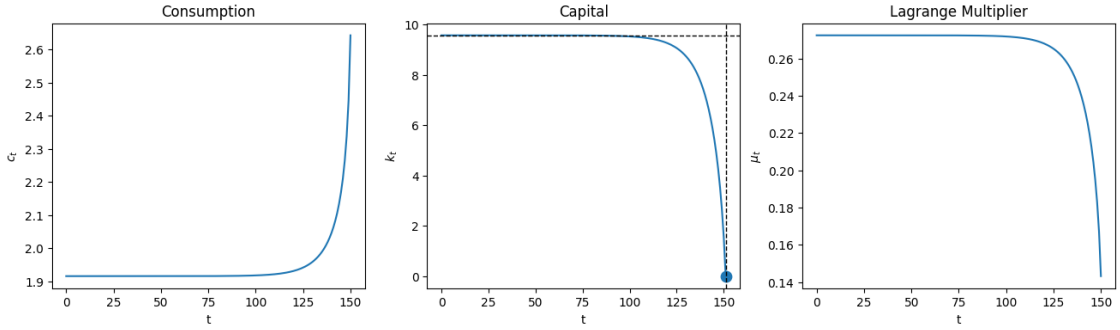
Аналогично, из (6) получим

$$f(\bar{K}) - \delta\bar{K} = \bar{C}$$

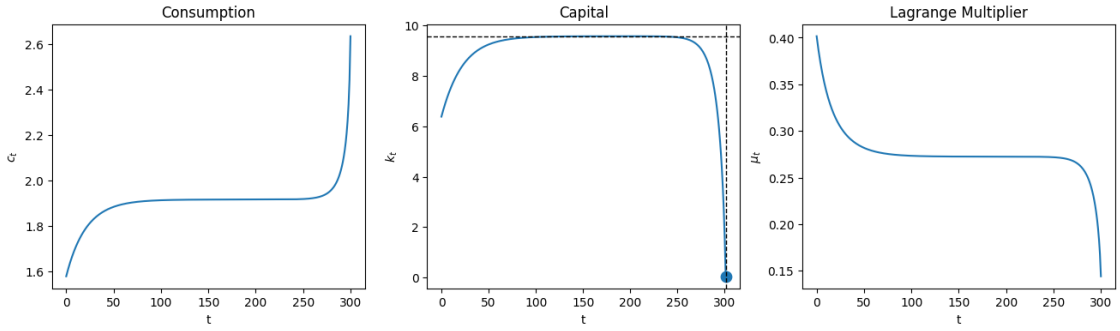
```
rho = 1 / pp.beta - 1
k_ss = pp.f_prime_inv(rho + pp.delta)
c_ss = pp.f(k_ss) - pp.delta * k_ss

print(f"Стационарное значение капитала: {k_ss}")
```

```
Стационарное значение капитала: 9.57583816331462
```

```
_ = plot_paths(pp, 0.3, k_ss, [150], k_ss=k_ss);
```




```
_ = plot_paths(pp, 0.3, k_ss * 2 / 3, [300], k_ss=k_ss)
```



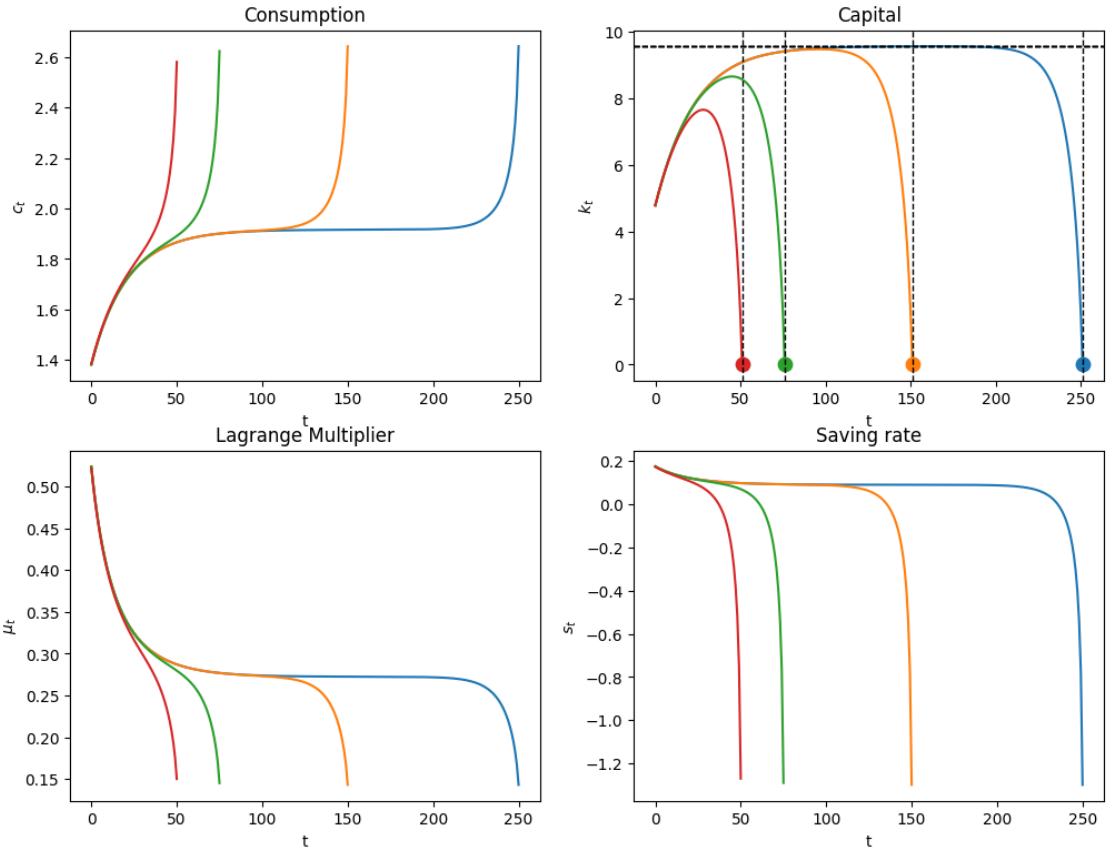
2.5. Норма сбережения

```
@njit
def saving_rate(pp, c_path, k_path):
    production = pp.f(k_path[:-1])
    return (production - c_path) / production
```

```
def plot_saving_rate(pp, c0, k0, T_arr, k_ter=0, k_ss=None,␣
  ↪s_ss=None):
```

```
    _, axes = plt.subplots(2, 2, figsize=(12, 9))

    c_paths, k_paths = plot_paths(
        pp, c0, k0, T_arr, k_ter=k_ter, k_ss=k_ss, axes=axes.
↪flatten()
    )

    for i, T in enumerate(T_arr):
        s_path = saving_rate(pp, c_paths[i], k_paths[i])
        axes[1, 1].plot(s_path)

    axes[1, 1].set(xlabel="t", ylabel="$s_t$", title="Saving␣
↪rate")

    if s_ss is not None:
        axes[1, 1].hlines(s_ss, 0, np.max(T_arr), linestyle="--")
```

```
plot_saving_rate(pp, 0.3, k_ss / 2, [250, 150, 75, 50],␣
↪k_ss=k_ss)
```



2.6. Дополнительный код для конкурентной экономики

```
@njit
def q(pp, c_path):
    T = len(c_path) - 1
    q_path = np.ones(T + 1)
    q_path[0] = 1
    for t in range(1, T + 1):
        q_path[t] = pp.beta**t * pp.u_prime(c_path[t])
    return q_path


@njit
def w(pp, k_path):
    w_path = pp.f(k_path) - k_path * pp.f_prime(k_path)
    return w_path


@njit
def η(pp, k_path):
    η_path = pp.f_prime(k_path)
    return η_path
```

```
T_arr = [250, 150, 75, 50]

fix, axs = plt.subplots(2, 3, figsize=(13, 6))
titles = [
    "Arrow-Hicks Prices",
    "Labor Rental Rate",
    "Capital Rental Rate",
    "Consumption",
    "Capital",
    "Lagrange Multiplier",
]
ylabels = ["$q_t^0$", "$w_t$", r"$\eta_t$", "$c_t$", "$k_t$",␣
  ↪r"$\mu_t$"]

for T in T_arr:
    c_path, k_path = bisection(pp, 0.3, k_ss / 3, T,␣
  ↪verbose=False)

    mu_path = pp.u_prime(c_path)

    q_path = q(pp, c_path)
    w_path = w(pp, k_path)[:-1]
    η_path = η(pp, k_path)[:-1]

    paths = [q_path, w_path, η_path, c_path, k_path, mu_path]
```

```
    for i, ax in enumerate(axs.flatten()):
        ax.plot(paths[i])
        ax.set(title=titles[i], ylabel=ylabels[i], xlabel="t")

        if titles[i] == "Capital":
            ax.axhline(k_ss, lw=1, ls="--", c="k")

        if titles[i] == "Consumption":
            ax.axhline(c_ss, lw=1, ls="--", c="k")

plt.tight_layout()
plt.show()
```