Семинар 9

1. Модель мультипликатора-акселлератора Самюэльсона

Модель Самюэльсона-Хикса объединяет в себе

1. уравнение национального дохода $Y_t = \bar{G}_t + I_t + C_t$,
2. кейнсианскую функцию потребления $C_t = \alpha Y_{t-1} + \gamma$ и
3. инвестиционный акселератор $I_t = \beta(Y_{t-1} - Y_{t-2})$.

Подставим выражения (2) и (3) в уравнение национального дохода (1): Получили неоднородное разностное уравнение второго порядка.

Данное уравнение решается обычным способом. Решим, для начала, однородное разностное уравнение

$$Y_t = \rho_1 Y_{t-1} + \rho_2 Y_{t-2}$$

Общее решение этого уравнения имеет вид

$$Y_t = C_1 \lambda_1^t + C_2 \lambda_2^t$$

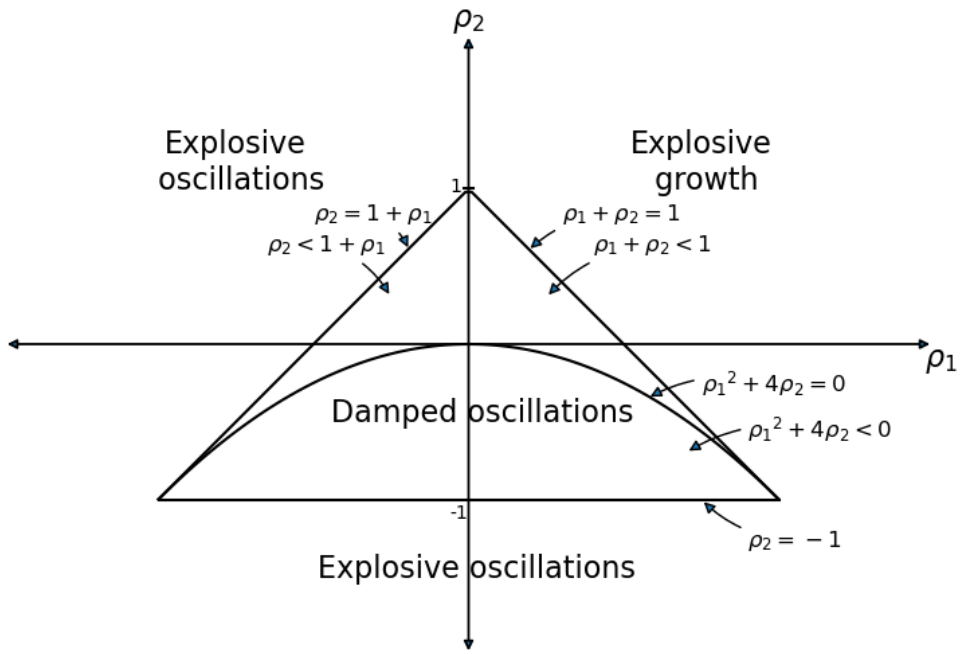
где $\lambda_{1,2}$ — корни характеристического уравнения

$$z^2 - \rho_1 z - \rho_2 = 0$$

Как известно, это уравнение имеет ровно два корня (возможно совпадающих):

- если $\lambda_1, \lambda_2 \in \mathbb{R}$ и $\lambda_1 \neq \lambda_2$, то
$$Y_t = C_1 \lambda_1^t + C_2 \lambda_2^t$$
Данное решение будет давать экспоненциальную сходимость, если $|\lambda_{1,2}| < 1$. Если один из корней строго больше 1, то получим экспоненциальный рост. Если же строго меньше -1 — взрывные осцилляции.
- если $\lambda_1, \lambda_2 \in \mathbb{R}$ и $\lambda_1 = \lambda_2$, то
$$Y_t = C_1 \lambda^t + C_2 t \lambda^t$$
- если $\lambda_1, \lambda_2 \in$, то $\lambda_1 = \lambda_2^*$! Решение также будет равно
$$Y_t = C_1 \lambda_1^t + C_2 \lambda_2^t$$
но оно всегда будет осциллирующим. Если $|\lambda_{1,2}| < 1$, то решение будет давать осциллирующую сходимость.

На рисунке можно увидеть, для каких значений $\rho_{1,2}$ будут получаться все описанные решения.

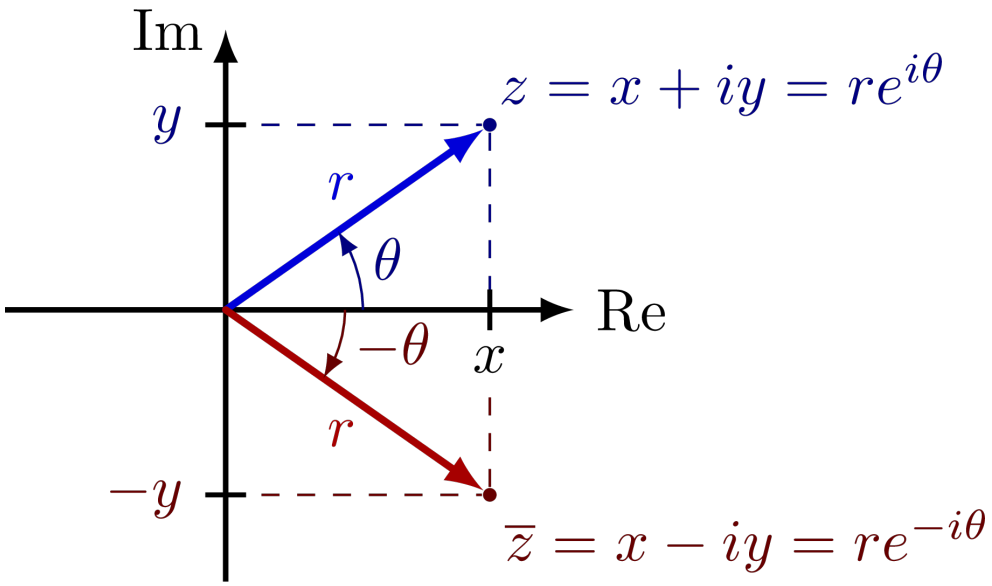


T.J. Sargent, J. Stachurski, CC BY-SA 4.0 https://creativecommons.org/licenses/by-sa/4.0/

Выведем решение для последнего случая. Вспомним, что комплексные числа имеют три эквивалентные записи

$$z = x + iy = re^{i\theta} = r[\cos(\theta) + i\sin(\theta)]$$

где $r = \sqrt{x^2 + y^2}$ — модуль числа.



TikZ.net, CC SA-BY 4.0 http://creativecommons.org/licenses/by-sa/4.0/

Тогда комплексно сопряженные корни имеют вид $\lambda_{1,2} = re^{\pm i\theta}$. Отсюда получим, что

$$\begin{aligned} Y_t &= C_1\lambda_1^t + C_2\lambda_2^t = \\ &= C_1 r^t e^{i\theta t} + C_2 r^t e^{-i\theta t} = \\ &= C_1 r^t \left[\cos(\theta t) + i\sin(\theta t)\right] + C_2 r^t \left[\cos(\theta t) - i\sin(\theta t)\right] = \\ &= (C_1 + C_2) r^t \cos(\theta t) + (C_1 - C_2) i \sin(\theta t) \end{aligned}$$

Внезапно вспомним, что Y_t должно быть **вещественным**! Для этого нужно, чтобы $C_1 + C_2 \in \mathbb{R}$ и $C_1 - C_2 \in \mathbb{C}$. Это возможно только если $C_1 = C_2^*$.

То есть $C_{1,2} = ve^{\pm i\xi}$. Получим

$$\begin{aligned} Y_t &= vr^t e^{i(\theta t+\xi)} + vr^t e^{-i(\theta t+\xi)} = \\ &= vr^t \left(e^{i(\theta t+\xi)} + e^{-i(\theta t+\xi)}\right) = \\ &= 2vr^t \cos(\theta t + \xi) \end{aligned}$$

Последнее следует из

$$\begin{aligned} e^{ix} + e^{-ix} &= \cos(x) + i\sin(x) + \cos(-x) + i\sin(-x) = \\ &= \cos(x) + i\sin(x) + \cos(x) - i\sin(x) = \\ &= 2\cos(x) \end{aligned}$$

Значение v зависит от начальных условий и неоднородного члена уравнения.

1.1. Представление в виде LSS

Те, кто внимательно изучил материалы предыдущей лекции, умеют записывать LSS для разностного уравнения второго порядка.

Вектор состояния запишем как

$$x_t = \begin{pmatrix} 1 \\ Y_t \\ Y_{t-1} \end{pmatrix}$$

Матрицу C — как

$$C = \begin{pmatrix} 0 \\ \sigma \\ 0 \end{pmatrix}$$

В описанном выше примере $\sigma = 0$, но вообще мы можем вводить в модель стохастику, давая σ положительное значение. Матрица G примет вид (это **НЕ** госрасходы)

$$G = \begin{pmatrix} 0 & 1 & 0 \end{pmatrix}$$

А матрица системы A

$$A = \begin{pmatrix} 1 & 0 & 0 \\ \bar{G} + \gamma & \alpha + \beta & -\beta \\ 0 & 1 & 0 \end{pmatrix}$$

Мы можем несколько усложнить матрицу G, получив в качестве наблюдений не только доход Y_{t+1}, но и потребление C_{t+1} и инвестиции I_{t+1}.

$$\begin{pmatrix} Y_{t+1} \\ C_{t+1} \\ I_{t+1} \end{pmatrix} = Gx_t$$

$$G = \begin{pmatrix} \bar{G} + \gamma & \rho_1 & \rho_2 \\ \gamma & \alpha & 0 \\ 0 & \beta & -\beta \end{pmatrix}$$

2. Эксперименты

```
import numpy as np
import quantecon as qe
import matplotlib.pyplot as plt

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

Построим модель для указанных далее значений параметров. Сначала сделаем все руками, затем сделаем класс, наследующий классу `LinearStateSpace`.

```
alpha = 0.8
beta = 0.9

r1 = alpha + beta
r2 = -beta

gamma = 10
sigma = 2
g = 10

n = 100
```

Посмотрим на корни характеристического многочлена $z^2 - \rho_1 z - \rho_2 = 0$.

```
roots = np.roots([1, -r1, -r2])
print(roots)
print([abs(r) for r in roots])
```

```
[0.85+0.42130749j 0.85-0.42130749j]
[np.float64(0.9486832980505138), np.float64(0.9486832980505138)]
```

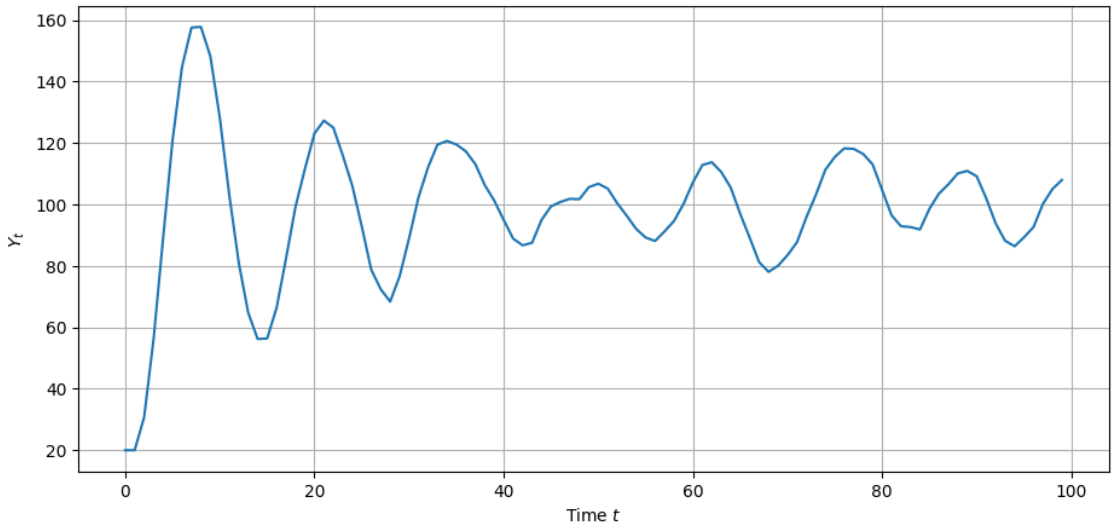
Создадим функцию для расчета **следующего** значения Y_{t+1}. На вход она принимает уже сгенерированный ряд `y`, из которого берутся два последних значения.

```
def transition(y, g=0, r1=r1, r2=r2, gamma=gamma, sigma=sigma):
    e = np.random.normal(0, 1) if sigma > 0 else 0
    return y[-1] * r1 + y[-2] * r2 + gamma + g + sigma * e
```

Смоделируем последовательность $\{Y_t\}$ для начальных значений $Y_{-1,-2} = 20$.

```
y_t = [20, 20]

for t in range(2, n):
    y_t.append(transition(y_t, g=g))

plt.clf()
fig, (ax) = plt.subplots()
ax.plot(y_t)
plt.grid()
plt.xlabel("Time $t$")
plt.ylabel("$Y_t$")
plt.show()
```

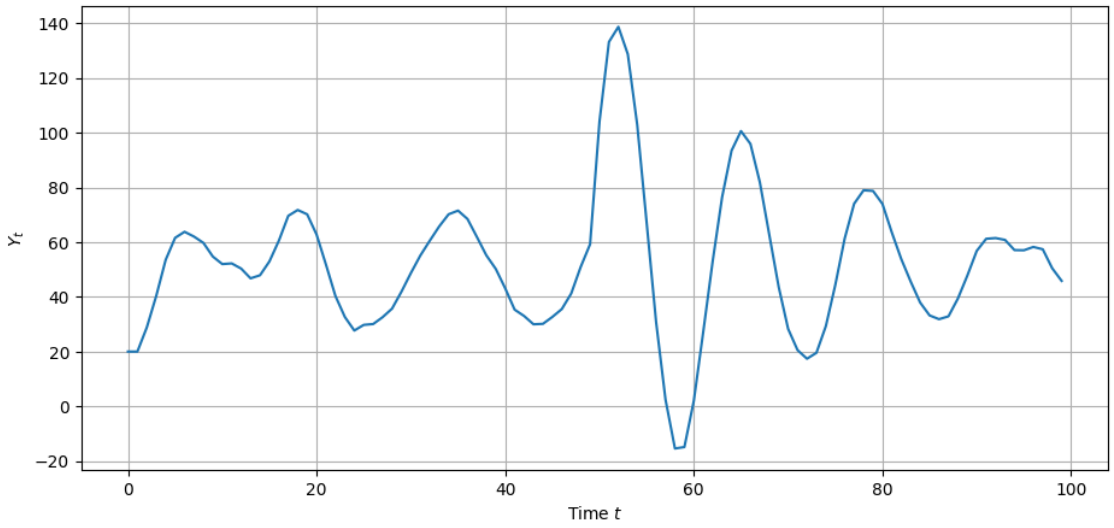
```
<Figure size 1100x500 with 0 Axes>
```



Посмотрим на результат единичного шока $G = 40$ в момент времени 50.

```
y_t = [20, 20]

for t in range(2, n):
    if t == 50:
        y_t.append(transition(y_t, g=40))
    else:
        y_t.append(transition(y_t))

plt.clf()
fig, (ax) = plt.subplots()
ax.plot(y_t)
plt.grid()
plt.xlabel("Time $t$")
plt.ylabel("$Y_t$")
plt.show()
```

```
<Figure size 1100x500 with 0 Axes>
```



2.1. Напишем собственный класс

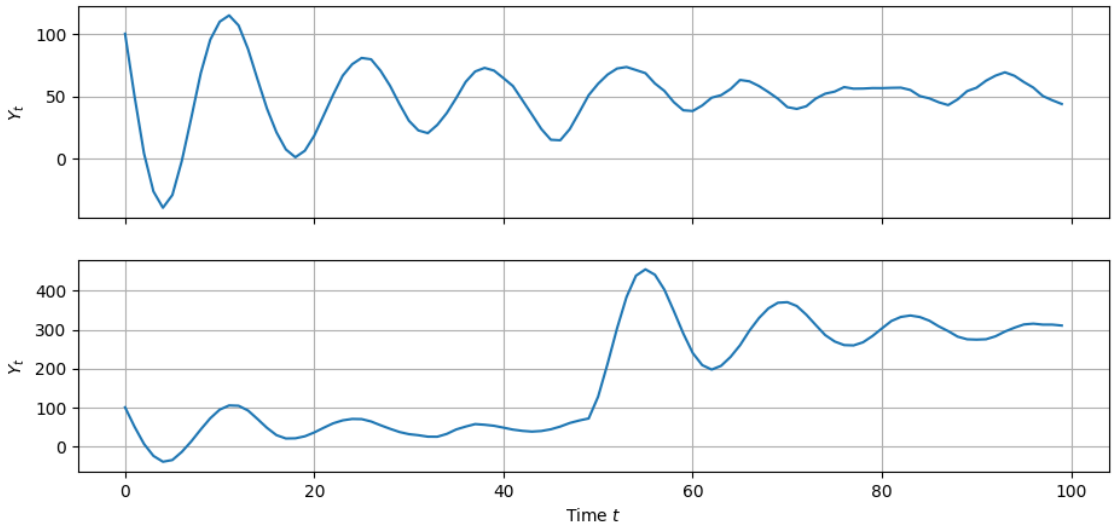
Постоянно возиться с пересозданием одного и того же при изменении каких-либо параметров неспортивно и неудобно. Давайте создадим свой собственный класс!

```
class Samuelson:
    def __init__(
        self,
        alpha,
        beta,
        gamma,
        G_shock=0,
        sigma=0,
        shock_time=0,
        shock_permanent=False,
        n=100,
        # Начальные значения
        y_0=100,
        y_1=50,
    ):
        self.alpha, self.beta = alpha, beta
        self.gamma = gamma
        self.sigma = sigma
        self.G_shock, self.shock_time = G_shock, shock_time
        self.shock_permanent = shock_permanent
        self.n = n
        self.y_0, self.y_1 = y_0, y_1

        self.r1, self.r2 = alpha + beta, -beta

    def _transition(self, y, G=0):
        e = np.random.normal(0, 1) if self.sigma > 0 else 0
```

```
        return self.r1 * y[-1] + self.r2 * y[-2] + self.gamma +␣
↪G + e * self.sigma

    def generate(self, n=None, shock=False):
        iters = self.n if n is None else n

        y_t = [self.y_0, self.y_1]

        for t in range(2, iters):
            # Если мы генерируем ряд с перманентным шоком G
            if shock and self.shock_permanent and t >= self.
↪shock_time:
                y_t.append(self._transition(y_t, G=self.G_shock))
            # Если мы генерируем ряд с одномоментным шоком G
            elif shock and not self.shock_permanent and t ==␣
↪self.shock_time:
                y_t.append(self._transition(y_t, G=self.G_shock))
            # В остальных случаях G=0
            else:
                y_t.append(self._transition(y_t))

        return y_t

    def plot(self):
        plt.clf()
        fig, ax = plt.subplots(2, 1, sharex=True)

        y = self.generate()
        ax[0].plot(y)
        ax[0].grid()
        ax[0].set_ylabel("$Y_t$")

        y = self.generate(shock=True)
        ax[1].plot(y)
        ax[1].grid()
        ax[1].set_ylabel("$Y_t$")
        ax[1].set_xlabel("Time $t$")

        plt.show()
```

```
sam = Samuelson(0.8, 0.9, 10, 50, 2, 50, True)
sam.plot()
```

```
<Figure size 1100x500 with 0 Axes>
```



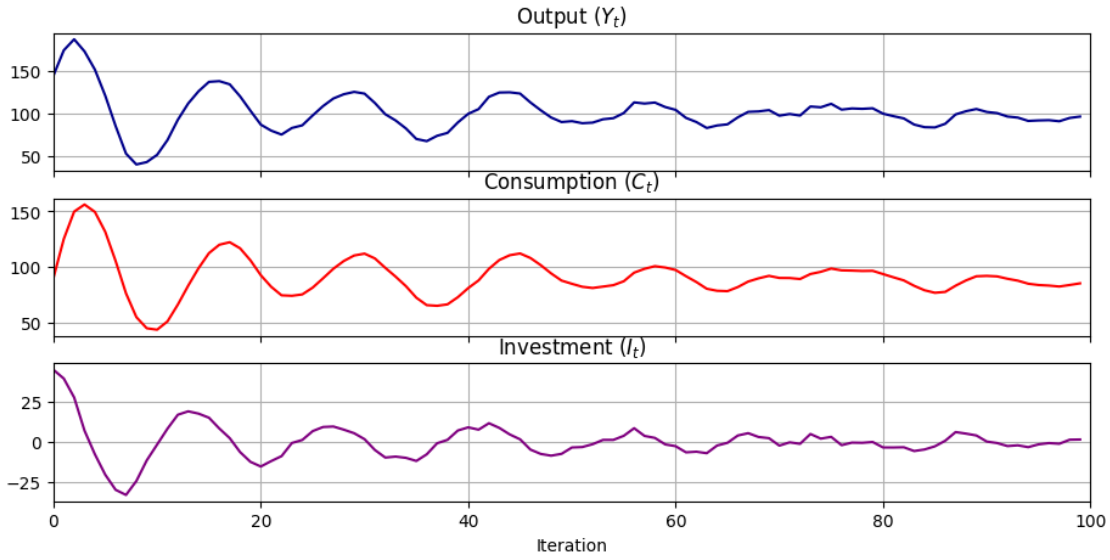
2.2. Воспользуемся классом LinearStateSpace

Разумеется, мы можем воспользоваться классом `LinearStateSpace`!

```
A = [
    [1, 0, 0],
    [gamma + g, r1, r2],
    [0, 1, 0],
]

C = np.zeros((3, 1))
C[1] = sigma

G = [
    [gamma + g, r1, r2],
    [gamma, alpha, 0],
    [0, beta, -beta],
]

mu_0 = [1, 100, 50]

sam2 = qe.LinearStateSpace(A, C, G, mu_0=mu_0)
```

```
x, y = sam2.simulate(ts_length=n)

fig, axes = plt.subplots(3, 1, sharex=True)
titles = ["Output ($Y_t$)", "Consumption ($C_t$)", "Investment
  ↪($I_t$)"]
colors = ["darkblue", "red", "purple"]
for ax, series, title, color in zip(axes, y, titles, colors):
    ax.plot(series, color=color)
    ax.set(title=title, xlim=(0, n))
```

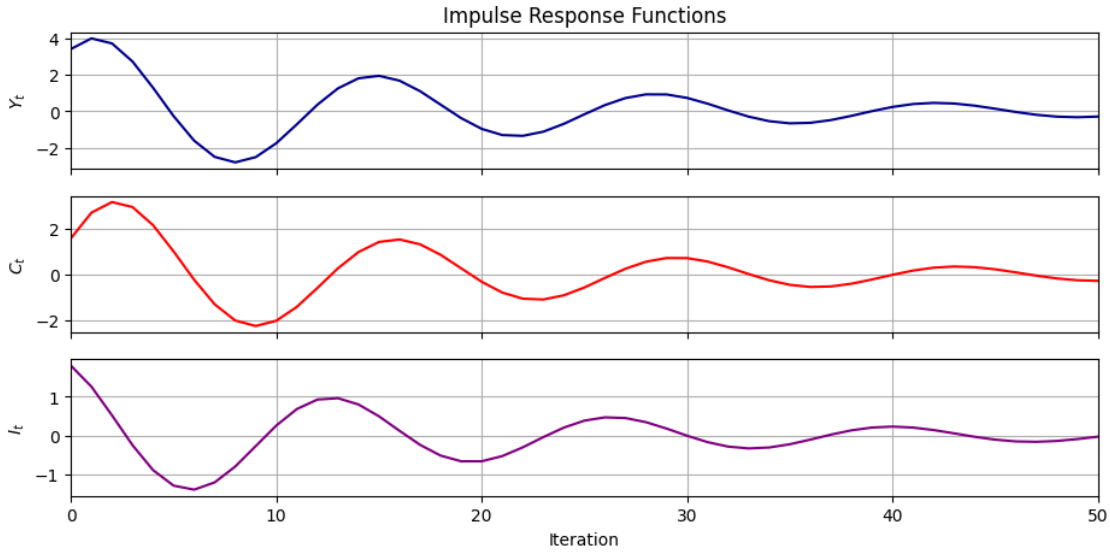
```
    ax.grid()

axes[-1].set_xlabel("Iteration")

plt.show()
```



Класс `LinearStateSpace` позволяет строить импульсные отклики на шоки.

```
x, y = sam2.impulse_response(50)
y = np.hstack(y)
```

```
fig, axes = plt.subplots(3, 1, sharex=True)
labels = ["$Y_t$", "$C_t$", "$I_t$"]
colors = ["darkblue", "red", "purple"]
for ax, series, label, color in zip(axes, y, labels, colors):
    ax.plot(series, color=color)
    ax.set(xlim=(0, 50))
    ax.set_ylabel(label)
    ax.grid()

axes[0].set_title("Impulse Response Functions")
axes[-1].set_xlabel("Iteration")
plt.show()
```



Создадим еще один класс, но с наследованием от класса `LinearStateSpace`. Велосипедостроение — вещь полезная, но ненужная. Наследование позволяет использовать все атрибуты и методы родительского класса как атрибуты и методы наследующего класса.

Обратите внимание на явный вызов конструктора родительского класса.

```
class SamuelsonLSS(qe.LinearStateSpace):
    def __init__(self, alpha, beta, gamma=10, G=10, sigma=0,␣
  ↪y_0=100, y_1=50):
        self.alpha, self.beta = alpha, beta
        self.gamma = gamma
        self.sigma = sigma
        self.G = G
        self.y_0, self.y_1 = y_0, y_1

        self.r1, self.r2 = alpha + beta, -beta

        self.A = [
            [1, 0, 0],
            [gamma + G, self.r1, self.r2],
            [0, 1, 0],
        ]

        self.C = np.zeros((3, 1))
        self.C[1] = sigma

        self.G = [
            [gamma + G, self.r1, self.r2],
            [gamma, alpha, 0],
            [0, beta, -beta],
        ]
```

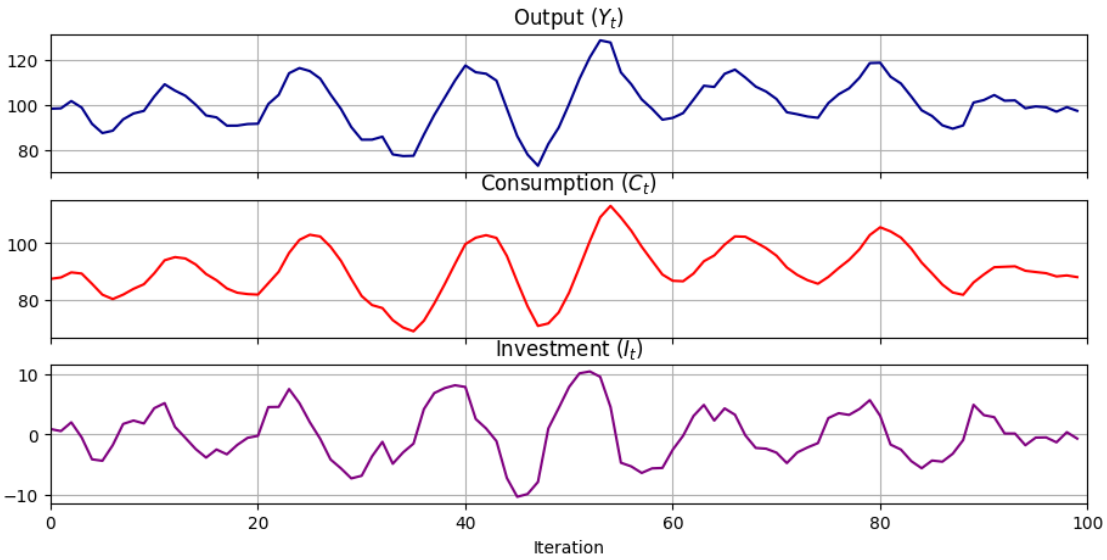
```
        self.mu_0 = [1, y_0, y_1]

        qe.LinearStateSpace.__init__(self, self.A, self.C, self.
↪G, mu_0=self.mu_0)

    def plot(self, ts_length=100, stationary=True):
        orig_mu = self.mu_0
        orig_Sigma = self.Sigma_0

        # Используем стационарное распределение в качестве␣
↪начальных значений рядов.
        if stationary:
            try:
                mu_x, mu_y, Sigma_x, Sigma_y, Sigma_xy = self.
↪stationary_distributions()
                self.mu_0 = mu_x
                self.Sigma_0 = Sigma_x
            except ValueError:
                print("There is no stationary distribution!")

        _, y = self.simulate(ts_length)

        fig, axes = plt.subplots(3, 1, sharex=True)
        titles = ["Output ($Y_t$)", "Consumption ($C_t$)",␣
↪"Investment ($I_t$)"]
        colors = ["darkblue", "red", "purple"]
        for ax, series, title, color in zip(axes, y, titles,␣
↪colors):
            ax.plot(series, color=color)
            ax.set(title=title, xlim=(0, ts_length))
            ax.grid()

        axes[-1].set_xlabel("Iteration")

        self.mu_0 = orig_mu
        self.Sigma_0 = orig_Sigma

        return fig

    def plot_irf(self, j=5):
        _, y = self.impulse_response(j)
        yirf = np.hstack(y)

        fig, axes = plt.subplots(3, 1, sharex=True)
        labels = ["$Y_t$", "$C_t$", "$I_t$"]
        colors = ["darkblue", "red", "purple"]
        for ax, series, label, color in zip(axes, yirf, labels,␣
↪colors):
```

```
            ax.plot(series, color=color)
            ax.set(xlim=(0, j))
            ax.set_ylabel(label)
            ax.grid()

        plt.suptitle("Impulse Response Functions")
        axes[-1].set_xlabel("Iteration")

        return fig
```

Построим графики со стационарными начальными значениями.

```
sam3 = SamuelsonLSS(0.8, 0.9, 10, sigma=2)
plt.clf()
sam3.plot()
plt.show()
```

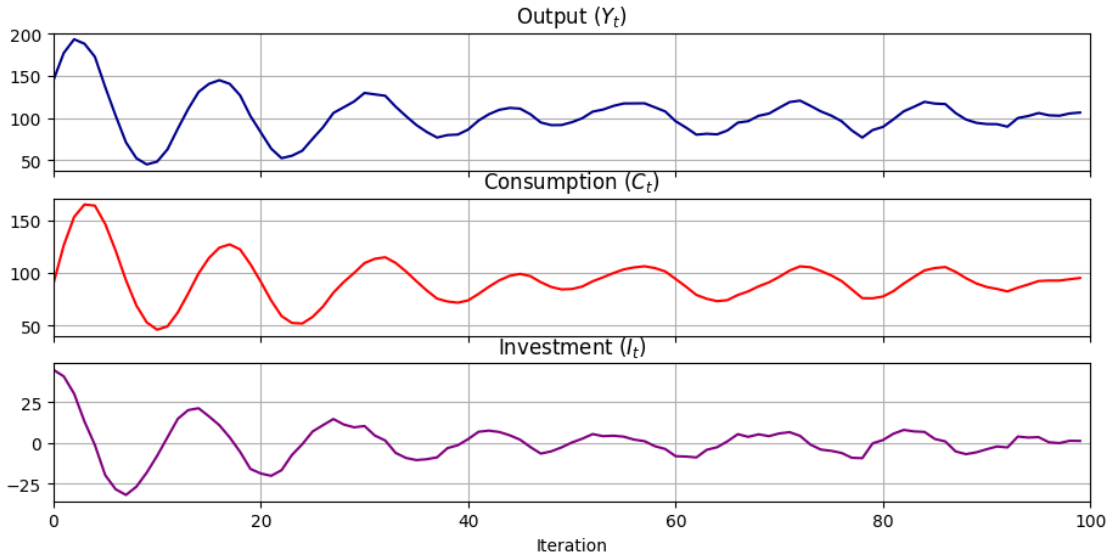
```
<Figure size 1100x500 with 0 Axes>
```



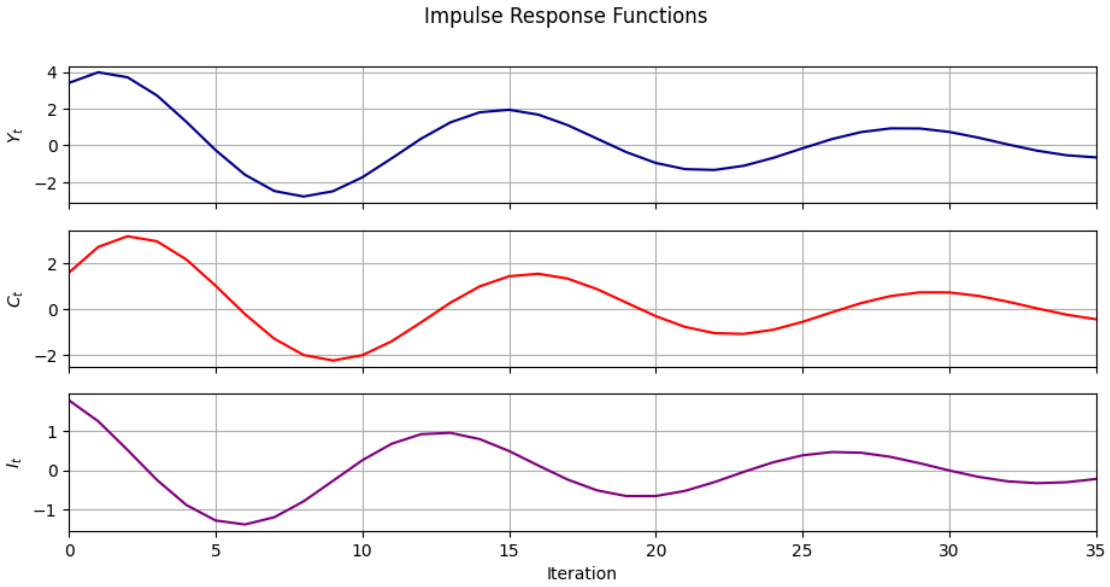
И с заданными вручную при создании объекта.

```
plt.clf()
sam3.plot(stationary=False)
plt.show()
```

```
<Figure size 1100x500 with 0 Axes>
```



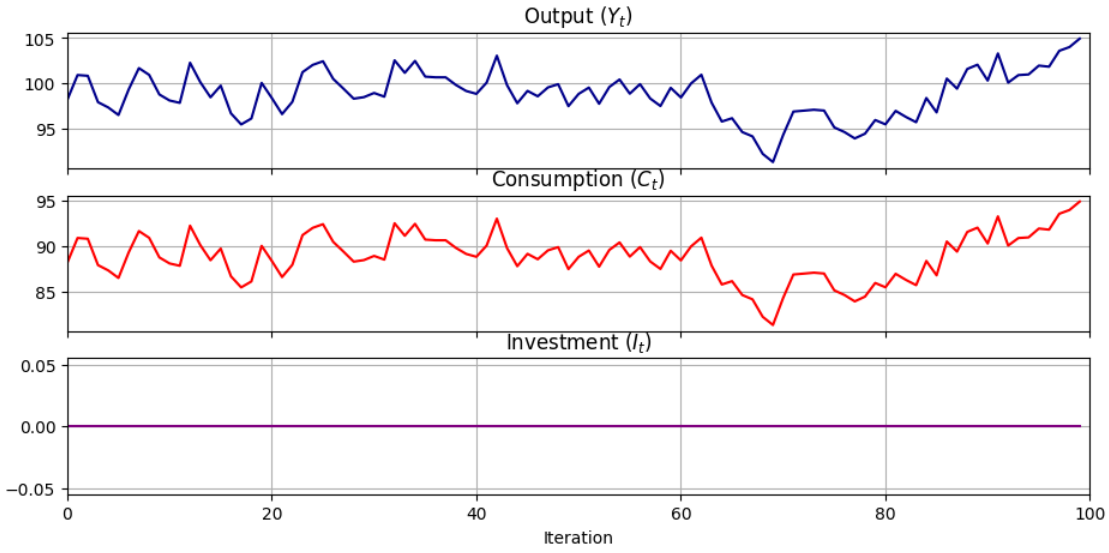

```
sam3.plot_irf(35)
plt.show()
```




```
sam4 = SamuelsonLSS(0.8, 0, sigma=2)
```

```
plt.clf()
sam4.plot()
plt.show()
```

```
<Figure size 1100x500 with 0 Axes>
```




```
plt.clf()
sam4.plot_irf(100)
plt.show()
```

```
<Figure size 1100x500 with 0 Axes>
```