Второе задание. Вариант 10. Заяц Артур.

In [470]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import scipy as sps
import matplotlib.pyplot as plt
from scipy.optimize import fsolve
from mpl_toolkits import mplot3d
from IPython.display import Image
sns.set_theme(style="whitegrid", palette="pastel")
```

Задача 1

Используя систему компьютерной алгебры [Wolfram Mathematica]() решим систему в явном виде. Полученное решение запишем в виде функции.

In [471]:

```
Image("/home/stevejobs/Pictures/1.png")
```

Out[471]:

```
In[3]:= eqn = {v'[t] == -1/t*((1-t)*v[t]+4*w[t]), w'[t] == v[t], w[0] == 1, v[0] == -4};
        sol = DSolve[eqn, {v, w}, t]
```

Solve : Inconsistent or redundant transcendental equation . After reduction , the bad equation is $-c_2 == 0$.

Solve : Inverse functions are being used by Solve , so some solutions may not be found ; use Reduce for complete solution information .

Out[4]= $\left\{\left\{v \to \text{Function}\left[\{t\}, \frac{1}{6}(-24+36\,t-12\,t^2+t^3)\right], w \to \text{Function}\left[\{t\}, \frac{1}{24}(24-96\,t+72\,t^2-16\,t^3+t^4)\right]\right\}\right\}$

In [472]:

```
def solution(t):
    return np.array([1/6*(-24+36*t-12*t**2+t**3),1/24*(24-96*t+72*t**2-16*
t**3+t**4)])
```

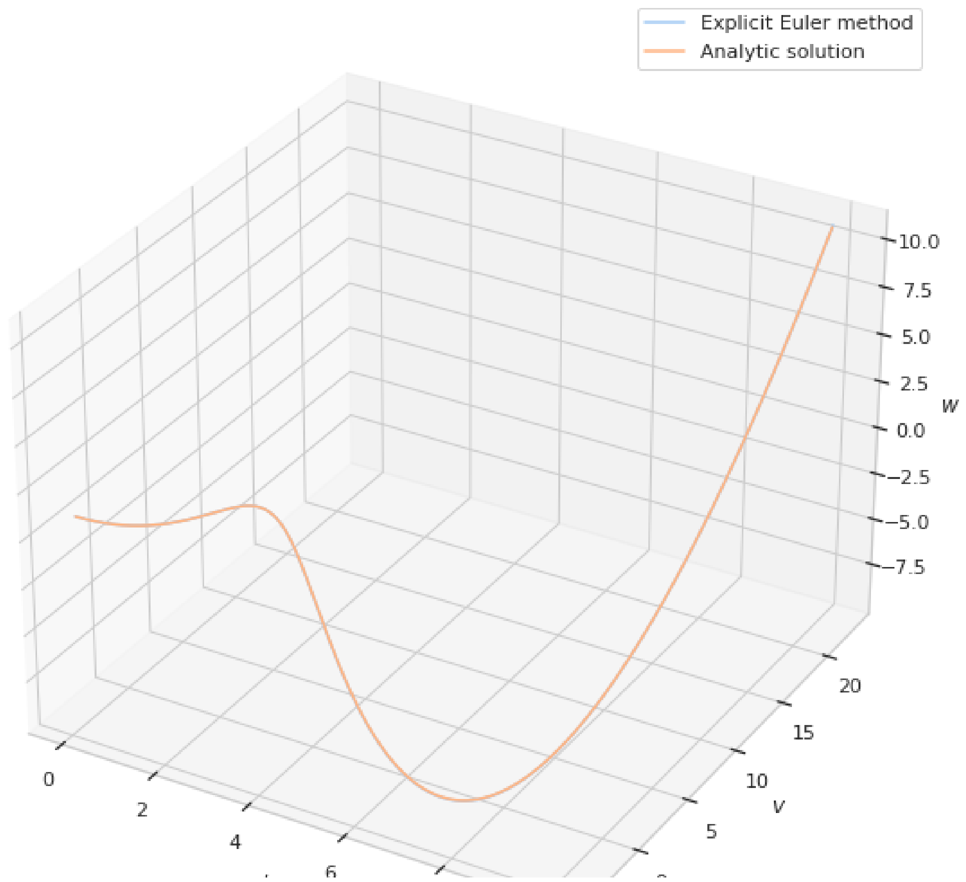
Решение методом Эйлера

С начала реализуем **явный метод Эйлера** на отрезке [0,10] с нашими начальными условиями. Возможность изменять оставим только step

In [475]:

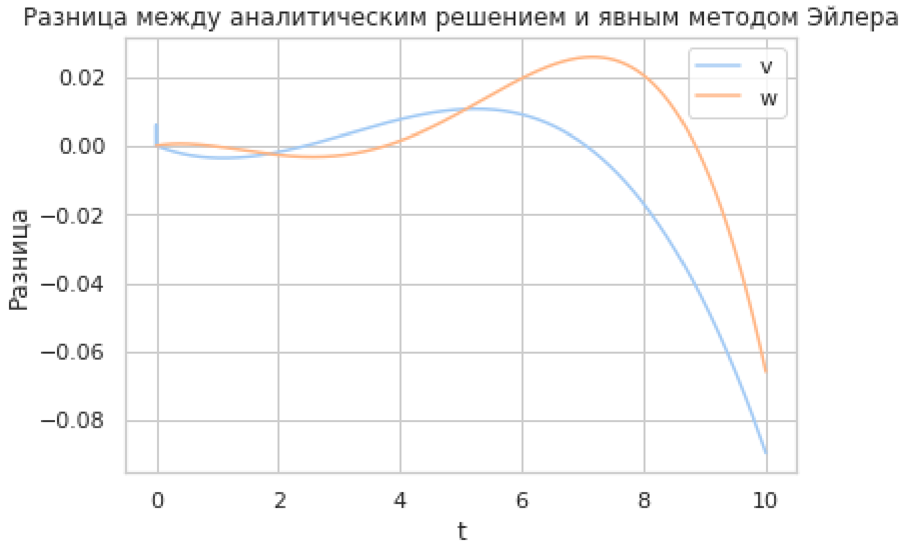
```
def f(v,w,t): #Правая часть
    if t == 0: #Т.к. в нуле проблемы
```

```
        return np.array([0,-4])
    return np.array([-1/t*((1-t)*v+4*w),v])

step = 0.001 #Длина шага
time = np.array([i * step for i in range(int(10/step) + 1)]) #Наши шаги по
времени

v_explicit, w_explicit = np.zeros(len(time)), np.zeros(len(time)) #Массив
 с ответами
v_explicit[0], w_explicit[0] = -4, 1 #Начальные условия

for i in range(1,len(time)): #Для каждого времени ищем ответ
    v_explicit[i] = step * f(v_explicit[i-1],w_explicit[i-1],time[i-1])[0]
+ v_explicit[i-1]
    w_explicit[i] = step * f(v_explicit[i-1],w_explicit[i-1],time[i-1])[1]
+ w_explicit[i-1]

solution_v = np.array([solution(i)[0] for i in time]) #Точные решения
solution_w = np.array([solution(i)[1] for i in time])

#Рисуем графики
plt.figure(figsize = (15,10))
ax = plt.axes(projection='3d')
ax.plot(time, v_explicit, w_explicit, label = 'Explicit Euler method')
ax.plot(time, solution_v, solution_w, label = 'Analytic solution')
ax.set_xlabel('$t$')
ax.set_ylabel('$v$')
ax.set_zlabel('$w$')
ax.legend();
```



In [476]:

```
plt.plot(time, solution_v-v_explicit,  label = 'v')
plt.plot(time, solution_w - w_explicit, label = 'w')
plt.title('Разница между аналитическим решением и явным методом Эйлера')
plt.xlabel('t')
plt.ylabel('Разница')
plt.legend();
```

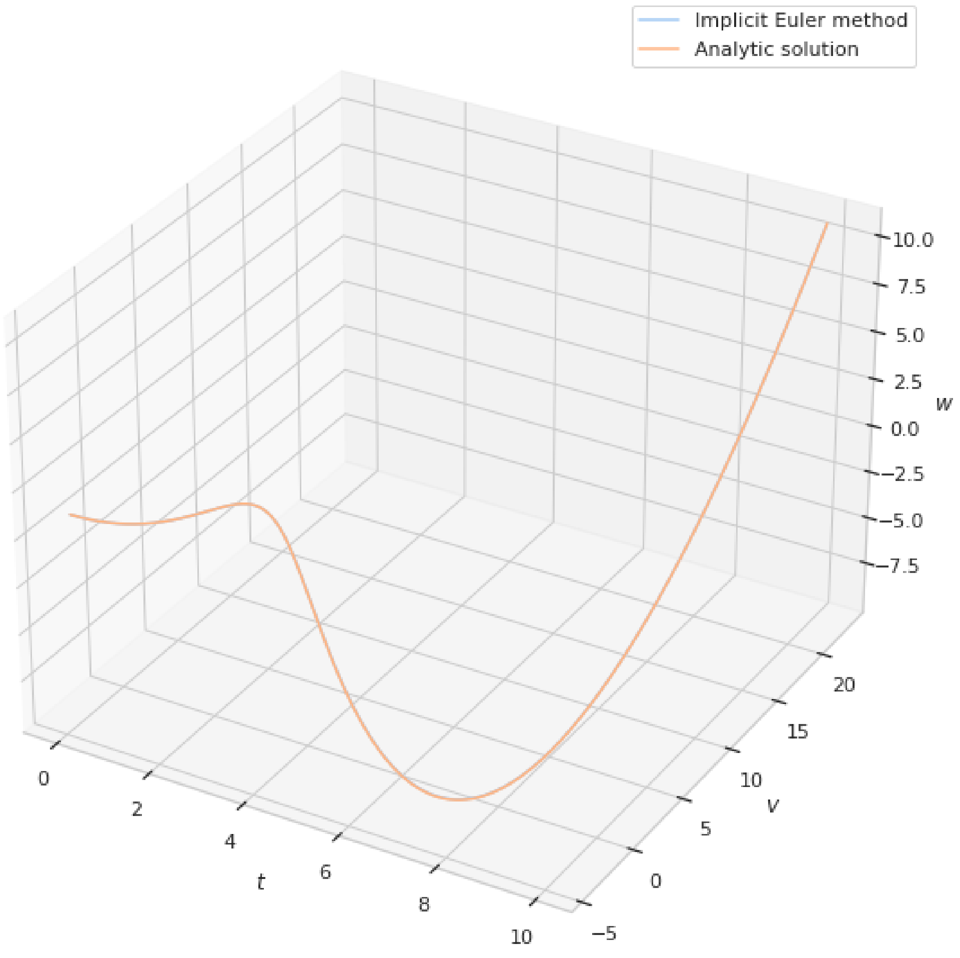


Теперь посмотрим на **неявный метод Эйлера**. В данном случае v^{k+1} и w^{k+1} элементарно аналитически выражаются.

In [477]:

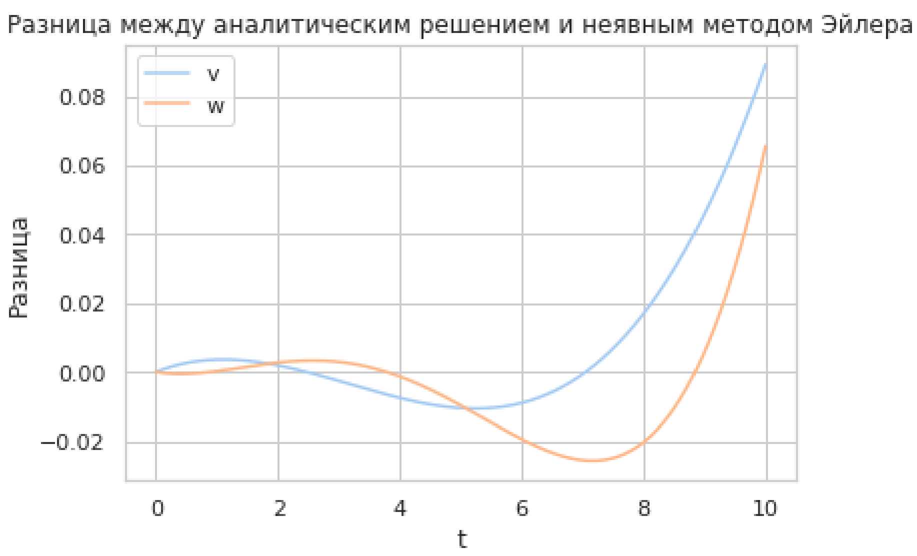
```
def next_iter_v(v,w,t): #V_n+1 через v_n, w_n, t_n+1 полученно аналитически.
    return (v - step * 4 * w / t)/ (1 + (step * (1-t) + 4 * step ** 2)/ t )

def next_iter_w(v,w,t): #Аналогично w через v_n+1!
    return (w+step*v)


v_implicit, w_implicit = np.zeros(len(time)), np.zeros((len(time))) #Массив ответов
v_implicit[0], w_implicit[0] = -4, 1 #Начальные условия

for i in range(1,len(time)):
    v_implicit[i] = next_iter_v(v_implicit[i-1],w_implicit[i-1],time[i])
    w_implicit[i] = next_iter_w(v_implicit[i],w_implicit[i-1],time[i])

plt.figure(figsize = (15,10))
ax = plt.axes(projection='3d')
ax.plot(time, v_implicit, w_implicit, label = 'Implicit Euler method')
ax.plot(time, solution_v, solution_w, label = 'Analytic solution')
ax.set_xlabel('$t$')
ax.set_ylabel('$v$')
```

```
ax.set_zlabel('$w$')
ax.legend();
```



In [347]:

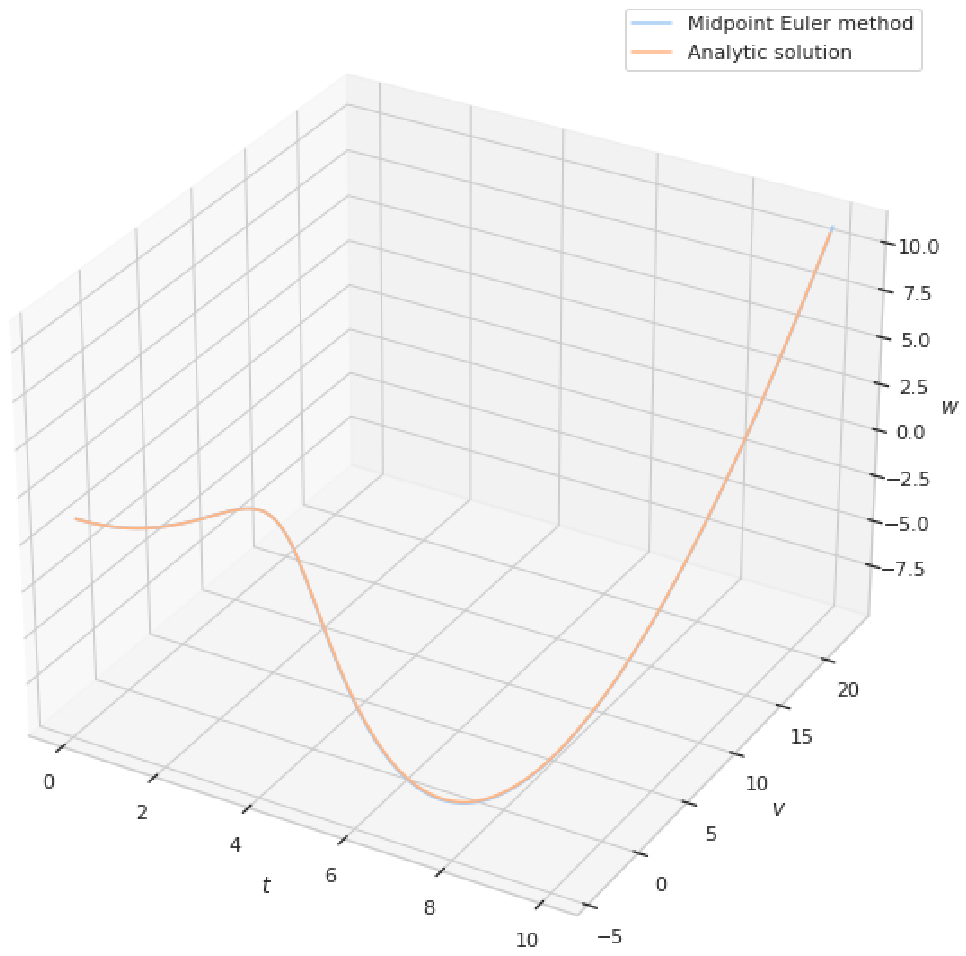
```
plt.plot(time, solution_v-v_implicit,  label = 'v')
plt.plot(time, solution_w - w_implicit, label = 'w')
plt.title('Разница между аналитическим решением и неявным методом Эйлера')
plt.xlabel('t')
plt.ylabel('Разница')
plt.legend();
```



Теперь посмотрим на **метод Эйлера с центральной точкой**. Первое приближение возьмём из явного метода Эйлера.

In [478]:

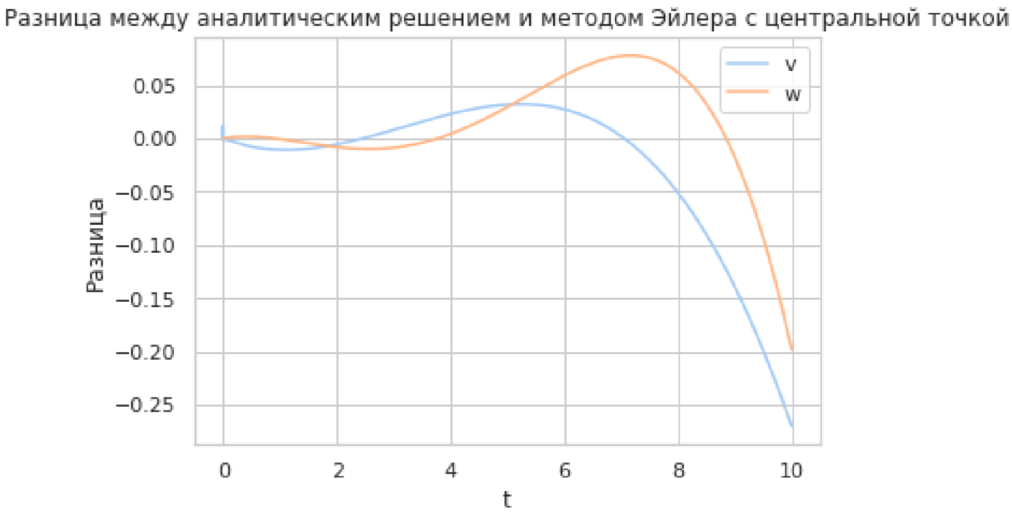
```
v_central, w_central = np.zeros(len(time)), np.zeros(len(time)) #Массив от
ветов
v_central[0], w_central[0] = -4, 1 #Начальные условия
v_central[1], w_central[1] = v_explicit[1], w_explicit[1] #Первое возьмём
 из явного метода

for i in range(2, len(time)):
    v_central[i] = step * f(v_central[i-2], w_central[i-2], time[i-2])[0]
+ v_central[i-1]
    w_central[i] = step * f(v_central[i-2], w_central[i-2], time[i-2])[1]
+ w_central[i-1]



plt.figure(figsize = (15,10))
ax = plt.axes(projection='3d')
ax.plot(time, v_central, w_central, label = 'Midpoint Euler method')
ax.plot(time, solution_v, solution_w, label = 'Analytic solution')
ax.set_xlabel('$t$')
ax.set_ylabel('$v$')
ax.set_zlabel('$w$')
ax.legend();
```



In [349]:

```
plt.plot(time, solution_v-v_central,  label = 'v')
plt.plot(time, solution_w - w_central, label = 'w')
plt.title('Разница между аналитическим решением и методом Эйлера с централ
```

```
ьной точкой')
plt.xlabel('t')
plt.ylabel('Разница')
plt.legend();
```



Анализ

Как видно из последних графиков для каждого метода Эйлера, неявный метод Эйлера оказался самым точным(расхождение с точным решением минимально в течение всего времени интегрирования).

Шаг был выбран из соображений "достаточно маленький, но чтобы долго не считало". На практике, согласно, например, данному источнику, он выбирается из требуемой точности, какой в условии задачи не было.

Вообще, для метода Рунге-Кутты произвольного порядка известно достаточное условие устойчивости: $c \cdot \tau \leq 1$, где τ - шаг интегрирования, c - константа Липшица правой части уравнения. Для данной задачи, поиск\оценивание константы Липшица правой части - проблема, труднее всей задачи в целом. Исследовать на устойчивость с помощью фазовых траекторий - тоже непосильная задача, т.к. система неавтономна.

Так же важно отметить, что ни в курсе лекций, ни в "ближайшом интернете" не упоминается словосочетание "строго устойчивый метод". Было лишь найдено понятие A-устойчивого метода.

Согласно литературе, явный метод Эйлера является A устойчивым на отрезке [0,2], а неявный - на всем вещественном интервале, кроме данного отрезка.

Решение методом Рунге-Кутты

Согласно литературе, наименьшая погрешность двухстадийного метода Рунге-Кутты достигается при следующих параметрах.

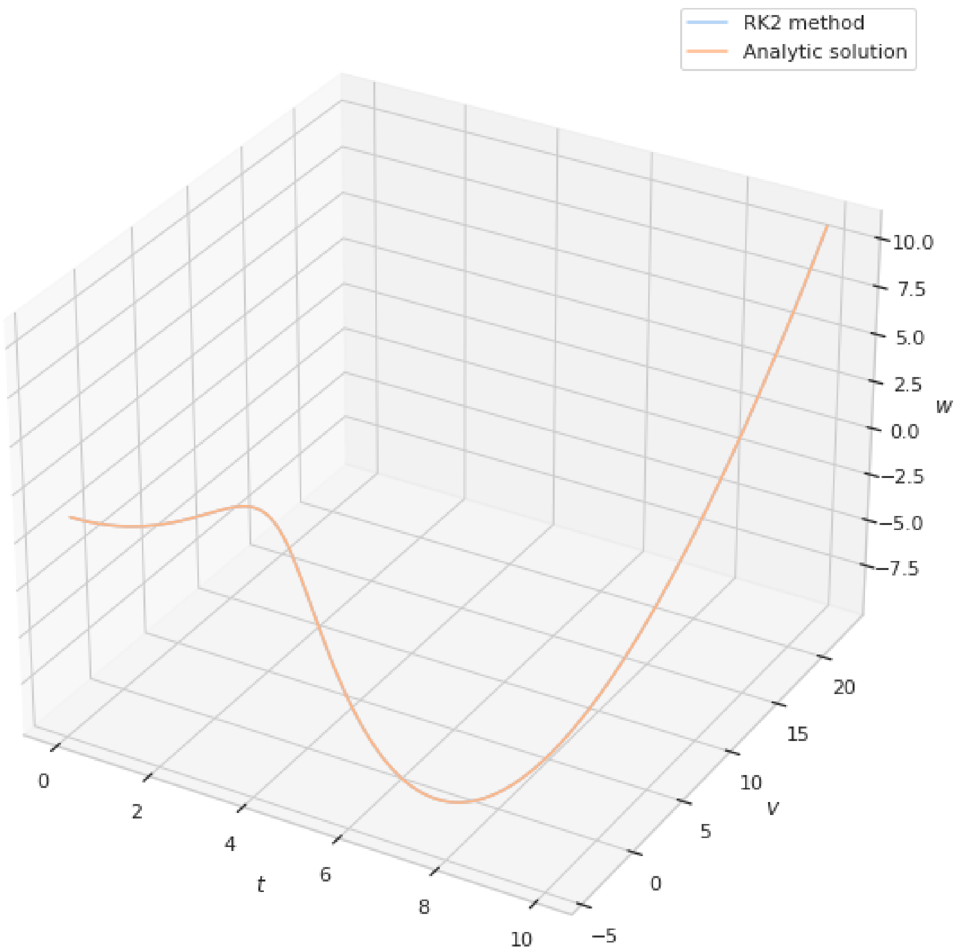
$$b_1 = 1/4, \quad b_2 = 3/4, \quad a_{21} = 2/3, \quad c_1 = 0, \quad c_2 = 2/3$$

Данная информация была получена в процессе подготовки к самостоятельной реализации.

Однако,реализовывать самому такое не обязательно, даже избыточно. Библиотка *scipy* имеет свою готовую реализацию. Для решения будем использовать следующую функцию. Один из главных параметров - rtol, т.е. мы сами можем контроллировать **точность** полученного **решения**. Данный метод имеет встроенный алгоритм автоматического выбора шага.

In [350]:

```
def func(t,y): #Определим функцию так, чтобы scipy мог с ей работать
    if t == 0:
        return np.array([0,-4])
    else:
        return np.array([-1/t*((1-t)*y[0]+4*y[1]),y[0]])

y_rk = sps.integrate.solve_ivp(func, (0,10), rtol = 0.00001, y0 = np.array
([-4,1]), method = 'RK23')

plt.figure(figsize = (15,10))
ax = plt.axes(projection='3d')
ax.plot(y_rk['t'], y_rk['y'][0], y_rk['y'][1], label = 'RK2 method')
ax.plot(time, solution_v, solution_w, label = 'Analytic solution')
ax.set_xlabel('$t$')
ax.set_ylabel('$v$')
ax.set_zlabel('$w$')
ax.legend();
```



In [463]:

```
solution_v, solution_w = [solution(t)[0] for t in y_rk['t']], [solution(t)
[1] for t in y_rk['t']]
plt.plot(y_rk['t'], y_rk['y'][0] - solution_v, label = 'v')
plt.plot(y_rk['t'], solution_w -  y_rk['y'][1], label = 'w')
plt.title('Разница между аналитическим решением и методом Рунге-Кутты трет
ьего порядка ')
plt.xlabel('t')
plt.ylabel('Разница')
plt.legend();
```

Но требуется проверить работу метода второго порядка. Будем использовать метод NSolve из *Wolfram Mathematica*. Т.к. в отличии от *Python*, в этой системе нельзя задать функцию, самостоятельно её доопределив в нуле, возьмём решение метода Эйлера на 10 шаге и будем решать задачу Коши с полученной точке на интервале времени [0.01,10]. Один из параметров функции - количество правильных знаков после запятой.

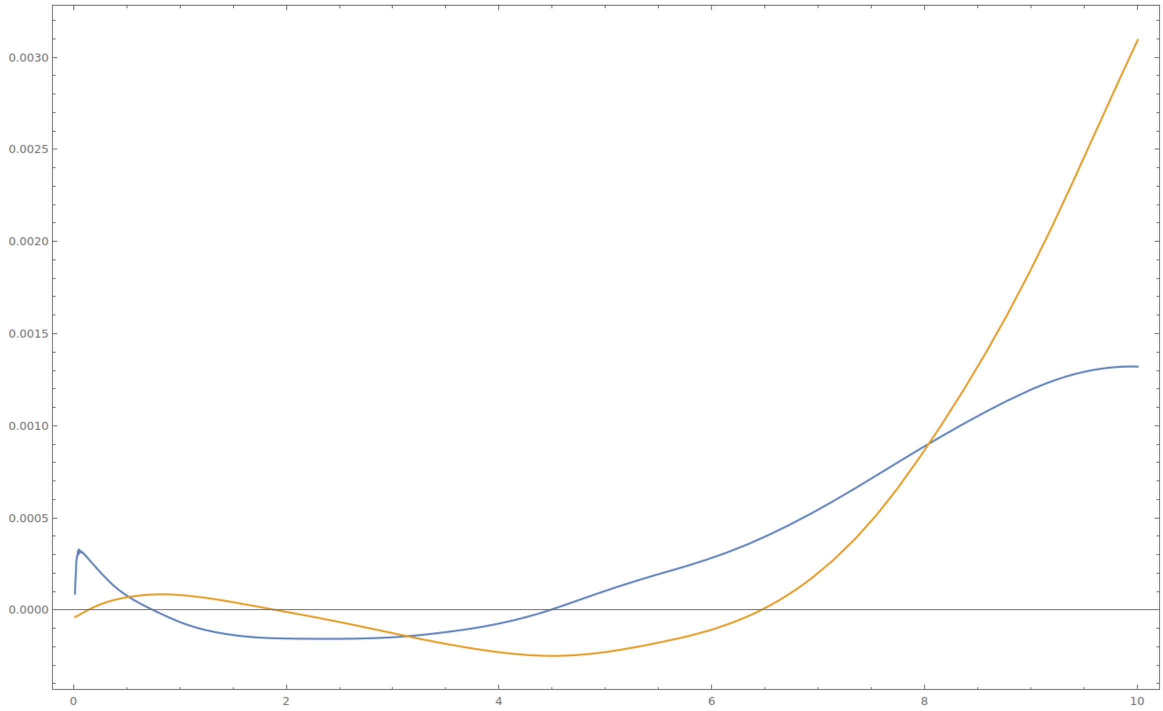
```
s = NDSolve[{v'[t] == -((1 - t) v[t] + 4 w[t])/t, w'[t] == v[t],
   w[0.01] == 9602638502660776/10^16,
   v[0.01] == -39401094717506693/10^16}, {v, w}, {t, 0.01, 10},
  Method →
   {"TimeIntegration" → {"ExplicitRungeKutta", "DifferenceOrder" → 2}},
  WorkingPrecision → 8]
```

```
Out[41]= {{v → InterpolatingFunction[ Domain: {{0.010000000, 10.000000}}
                                      Output: scalar ],
  w → InterpolatingFunction[ Domain: {{0.010000000, 10.000000}}
                             Output: scalar ]}}
```

Посмотрим на разницу с точным решением(по оси x - время, y - разница):

```
In[50]:= Plot[{Evaluate[{v[t] - 1/6 (-24 + 36 t - 12 t^2 + t^3), w[t] - 1/24 (24 + -96 t + 72 t^2 - 16 t^3 + t^4)} /. s]}, {t, 0.01, 10}, Frame → True]
```

Out[50]=



Выводы

Как видно, все методы показывают достаточно большую точность, сравнивать попарно их не имеет смысла. Также не имеет смысла искать оптимальный метод для этой задачи, в таком случае стоило бы делать поиск по более широкому классу методов.

Задача 2

Задача 3

Для начала определим фукнцию

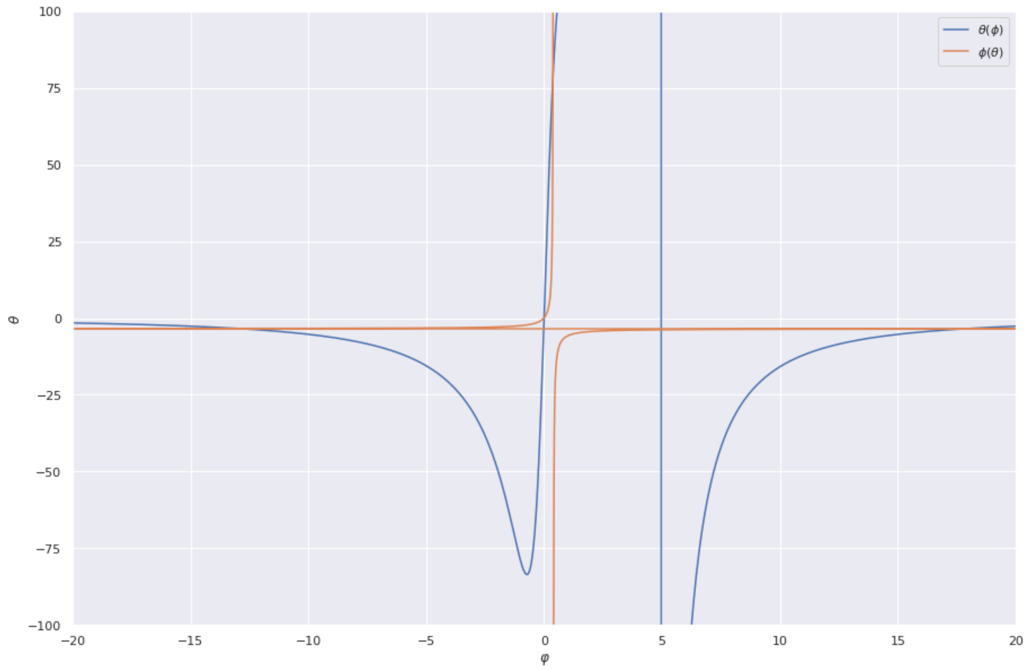
In [362]:

```
#Определим наши параметры
alpha, beta, gamma, theta_0, phi_0, C, k_1, k_2 = 2.0, 0.0015, 5.0, 3.5, 0.8275, 5.0, 0.05, 0.35

#Фунция уравнения
def kinetic(t,y):
    global alpha, beta, gamma, theta_0, phi_0, C, k_1, k_2

    theta, phi = y[0], y[1]

    dtheta = alpha * theta ** 2 / (theta + theta_0) - k_1 * theta - gamma * theta * phi
    dphi = beta * theta * (1 - phi/C) * (1 + (phi/phi_0) ** 2) - k_2 * phi

    return np.array([dtheta, dphi])
```

Для поиска особых точек системы построим графики функций $\theta(\varphi)$ и $\varphi(\theta)$, которые легко получаются, если приравнять левую часть системы к нулю.

In [402]:

```
phi, theta = np.linspace(-30,30,1000), np.linspace(-100,100,10000)

plt.figure(figsize = (15,10))

plt.plot(phi, k_2 * phi / (beta * (1 - phi / C) * (1 + (phi/phi_0)**2)), l
abel = r'$\theta(\phi)$')
plt.plot(1 / gamma * (alpha * theta/(theta + theta_0) - k_1), theta, label
= r'$\phi(\theta)$')
plt.xlim(-20,20)
plt.ylim(-100,100)
plt.xlabel(r'$\varphi$')
plt.ylabel(r'$\theta$')
plt.legend();
```



Видно, что есть всего 6 пересечений, а значи, и 6 решений.

Найдём особые точки системы. Для решения системы применим метод [NSolve] из уже упомянутой системы компьютерной алгебры *Wolfram*.

```
In[21]:= NSolve[
          {2 * theta ^ 2 / (theta + 3.5) - 0.05 * theta -
             5 * theta * phi == 0 ,
           -0.35 * phi + theta * 0.0015 * (1 - phi / 5) *
               (1 + (phi / 0.8275) ^ 2) == 0}, {theta, phi},
          Reals, WorkingPrecision -> 10]

Out[21]= {{theta -> 78.1444, phi -> 0.372852},
          {theta -> -3.39553, phi -> -13.0105},
          {theta -> -3.58122, phi -> 17.6273},
          {theta -> 0.0934328, phi -> 0.000400394},
          {theta -> 0., phi -> 0.}}
```

Всего пять особых точек. Внесем их в массив, оставив по 1 знакому после запятой. Т.к. мы хотим исследовать фазовые траектории вблизи особой точки, то точное значение нам незачем. Четвертую точку, т.к. она слишком близка к нулю, мы рассматривать не будем. Из графика выше видно, что мы действительно нашли точки пересечения данных функций. Не была лишь(почему-то) найдена точка около (5,-1). Рассмотрим её попозже.

In [418]:

```
dotes = np.array([[78,0.4],
         [-3.4, -13],
         [-3.6, 17.6],
         [0,0]])
```

Согласно другому методу той же системы компьютерной алгебры, который решает уравнения аналитически, всего действительно 5 особых точек у системы:

```
In[96]:= Solve[{2 * x ^ 2 / (x + 7 / 2) - 5 / 100 * x - 5 * x * y == 0 , -35 / 100 * y + x * (15 / 1000) * (1 - y / 5) * (1 + (y / (8275 / 10 000)) ^ 2) == 0}, {x, y}, Reals]
```

Out[96]=

$$\left\{\{x\to 0,\ y\to 0\},\ \left\{x\to -3.69\ldots,\ y\to \frac{338\,022\,323\,977\ (-3.69\ldots)}{736\,249\,920\,000}-\frac{1\,127\,450\,225\,441\ (-3.69\ldots)^2}{412\,299\,955\,200}-\frac{1\,997\,166\,059\,209\ (-3.69\ldots)^3}{6\,012\,707\,680\,000}+\frac{6\,810\,224\,894\,093\ (-3.69\ldots)^4}{42\,088\,953\,760\,000}\right\},\right.$$

$$\left\{x\to -3.09\ldots,\ y\to \frac{338\,022\,323\,977\ (-3.09\ldots)}{736\,249\,920\,000}-\frac{1\,127\,450\,225\,441\ (-3.09\ldots)^2}{412\,299\,955\,200}-\frac{1\,997\,166\,059\,209\ (-3.09\ldots)^3}{6\,012\,707\,680\,000}+\frac{6\,810\,224\,894\,093\ (-3.09\ldots)^4}{42\,088\,953\,760\,000}\right\},$$

$$\left\{x\to 0.150\ldots,\ y\to \frac{338\,022\,323\,977\ (0.150\ldots)}{736\,249\,920\,000}-\frac{1\,127\,450\,225\,441\ (0.150\ldots)^2}{412\,299\,955\,200}-\frac{1\,997\,166\,059\,209\ (0.150\ldots)^3}{6\,012\,707\,680\,000}+\frac{6\,810\,224\,894\,093\ (0.150\ldots)^4}{42\,088\,953\,760\,000}\right\},$$

$$\left.\left\{x\to 5.21\ldots,\ y\to \frac{338\,022\,323\,977\ (5.21\ldots)}{736\,249\,920\,000}-\frac{1\,127\,450\,225\,441\ (5.21\ldots)^2}{412\,299\,955\,200}-\frac{1\,997\,166\,059\,209\ (5.21\ldots)^3}{6\,012\,707\,680\,000}+\frac{6\,810\,224\,894\,093\ (5.21\ldots)^4}{42\,088\,953\,760\,000}\right\}\right\}$$

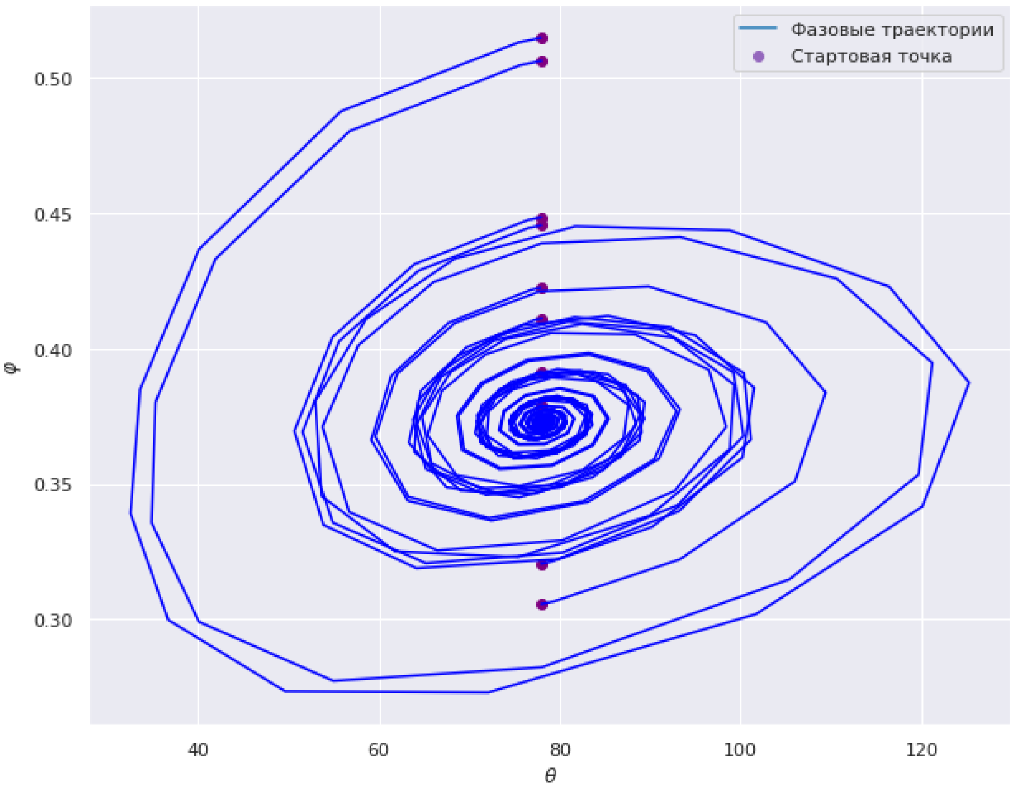
Когда мы убедились, что нашли все 5 точек правильно, исследуем поведение фазовых траекторий вдоль этих точек. Сделаем это с помощью готовой реализации метода Дорманда-Принца с адаптивным выбором шага, т.е. объеденим два требуемых метода: Рунге-Кутту с адаптивным методом шага и метод Дорманда-Принца, которые требуются в условии задачи.

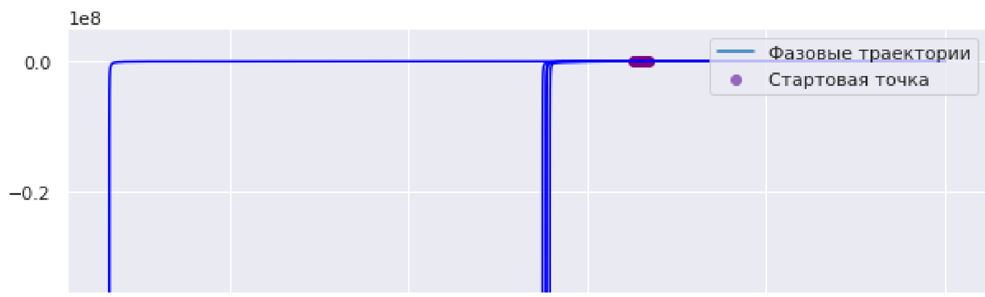
Будем брать нашу особую точку и брать по 10 случайных точек из её окрестности. В каждой точке будем строить решение на интервале времени от 0 до 100.

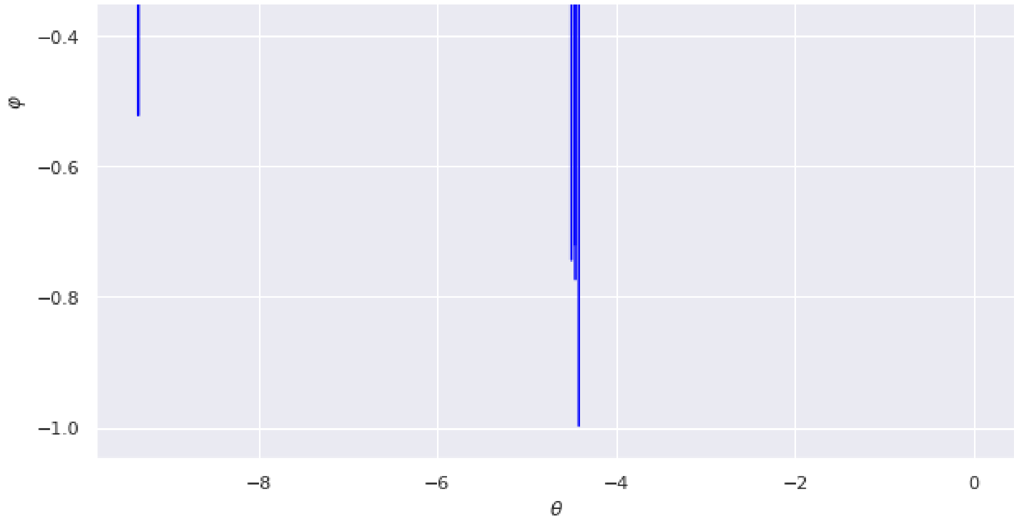
In [438]:

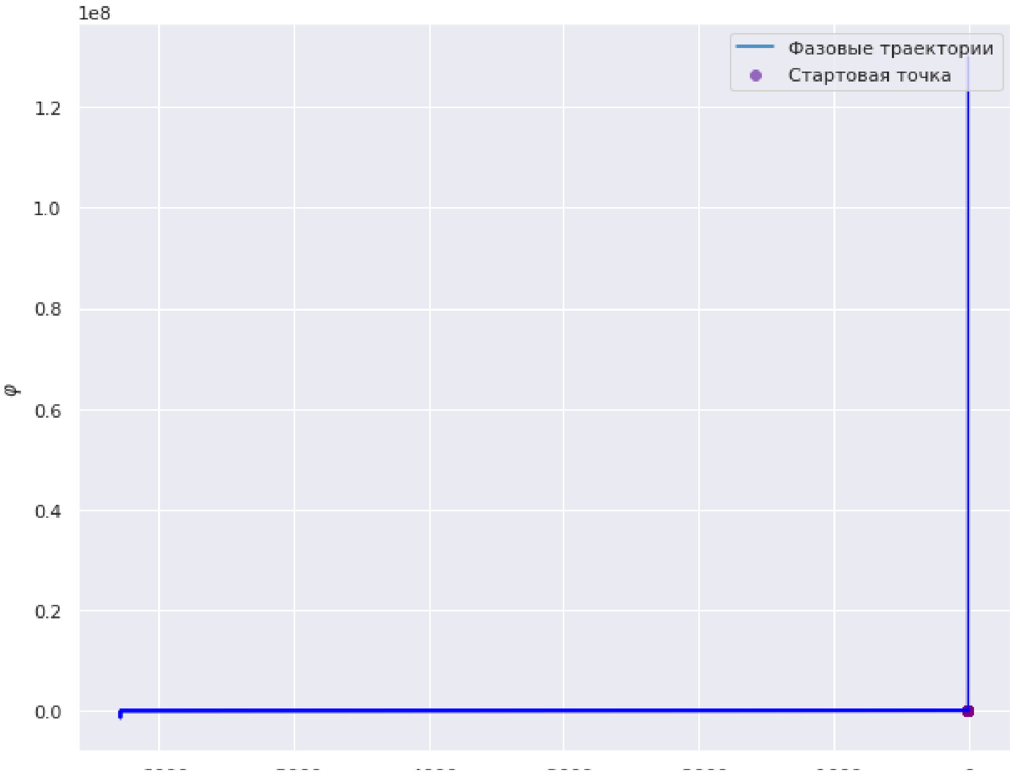
```
def bias(y): #Для исследований решений вблизи! особых точек
    return np.array([y[0] + np.random.normal(scale=0.2), y[1] + np.random.
```

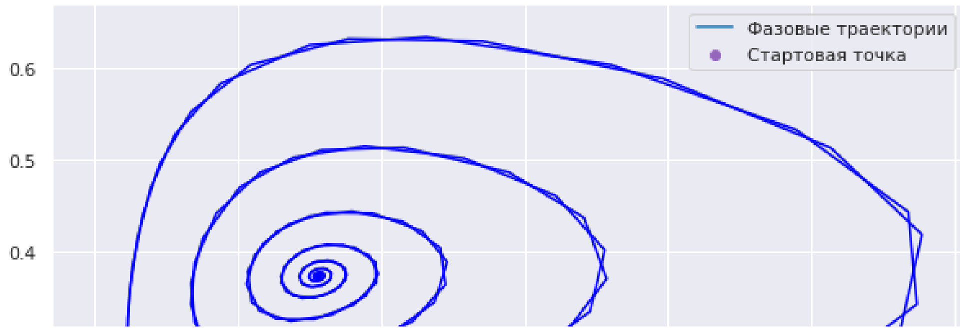
```
normal(scale=0.2)])

for i in range(4):
    plt.figure(figsize = (10,8))

    for _ in range(10):

        rk_res = sps.integrate.solve_ivp(kinetic,(0,100), bias(dotes[i]),
rtol = 1e-5) #Считаем

        plt.plot(rk_res['y'][0],rk_res['y'][1], color = 'blue') #Рисуем тр
аекторию

        plt.scatter(rk_res['y'][0][0],rk_res['y'][1][0], color = 'purple')

    plt.plot([],[],c ='tab:blue', label='Фазовые траектории')
    plt.scatter([],[],c='tab:purple', label='Стартовая точка')

    plt.xlabel(r'$\theta$')
    plt.ylabel(r'$\varphi$')

    plt.legend(loc = 'upper right')
```

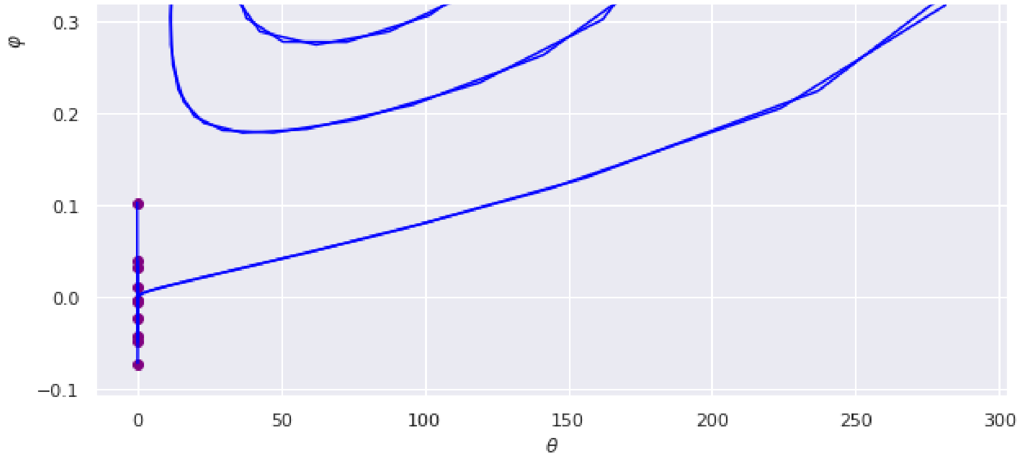





−0.4
−0.6
−0.8
−1.0
φ
−8
−6
−4
−2
0
θ


1e8
Фазовые траектории
Стартовая точка
1.2
1.0
0.8
0.6
0.4
0.2
0.0
φ

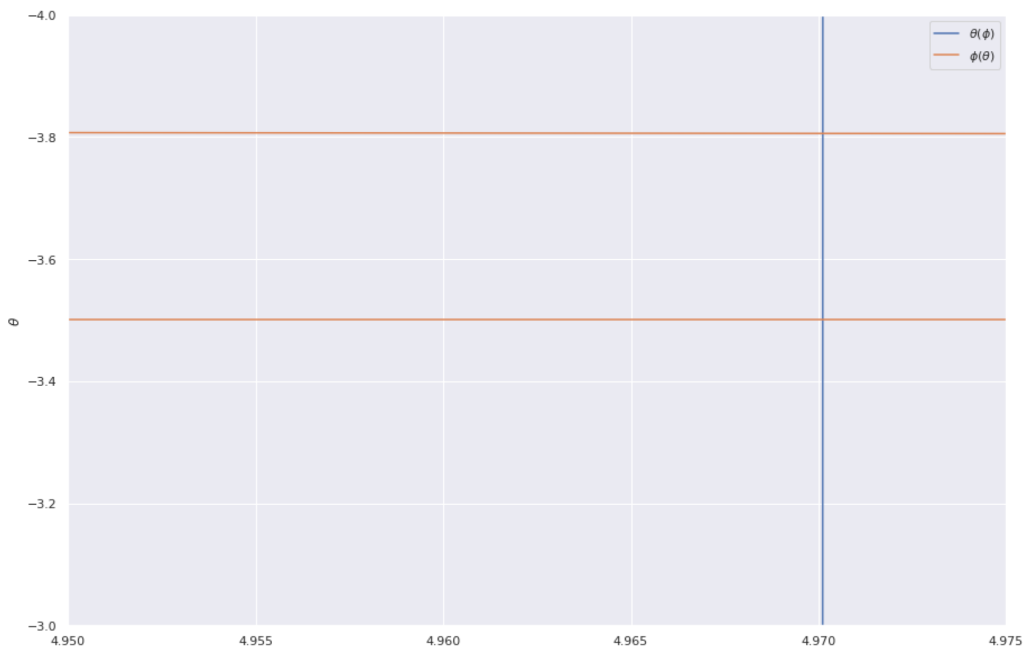

Фазовые траектории
Стартовая точка
0.6
0.5
0.4



Не был найден ни один цикл. Как видно, 2-4 точки не дают фазовых траекторий, а первая точка - устойчивый фокус. Рассмотрим последнюю точку, которую до этого не изучали. Т.к. вычислительные методы не помогают, определим её координаты графически(всё равно нужна только приблизительная точность).

In [446]:

```
phi, theta = np.linspace(-30,30,1000), np.linspace(-100,100,10000)

plt.figure(figsize = (15,10))

plt.plot(phi, k_2 * phi / (beta * (1 - phi / C) * (1 + (phi/phi_0)**2)), label = r'$\theta(\phi)$')
plt.plot(1 / gamma * (alpha * theta/(theta + theta_0) - k_1), theta, label = r'$\phi(\theta)$')
plt.xlim(4.950,4.975)
plt.ylim(-3,-4)
plt.xlabel(r'$\varphi$')
plt.ylabel(r'$\theta$')
plt.legend();
```

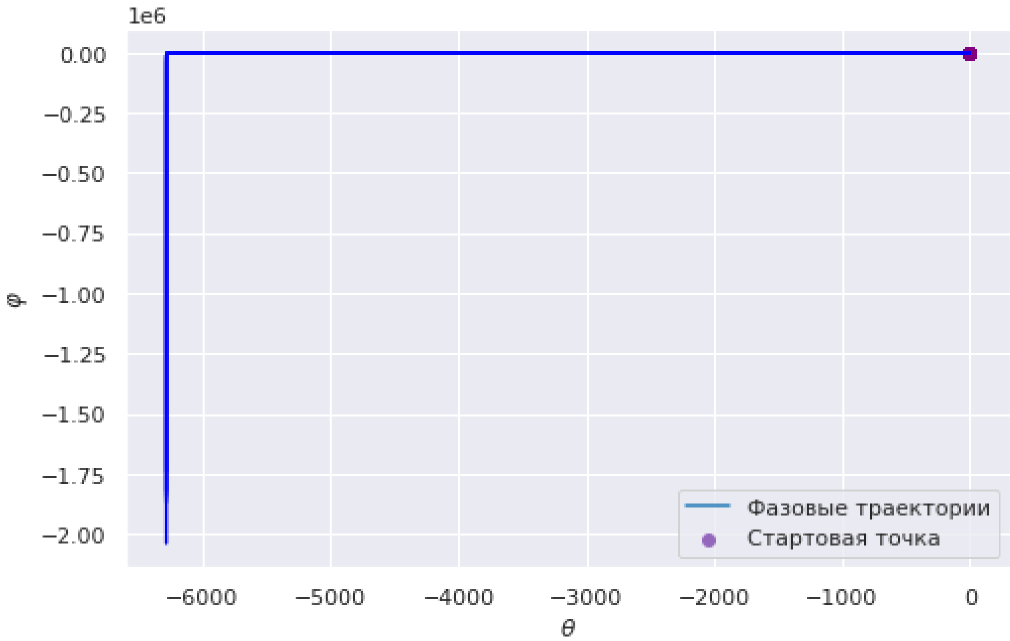


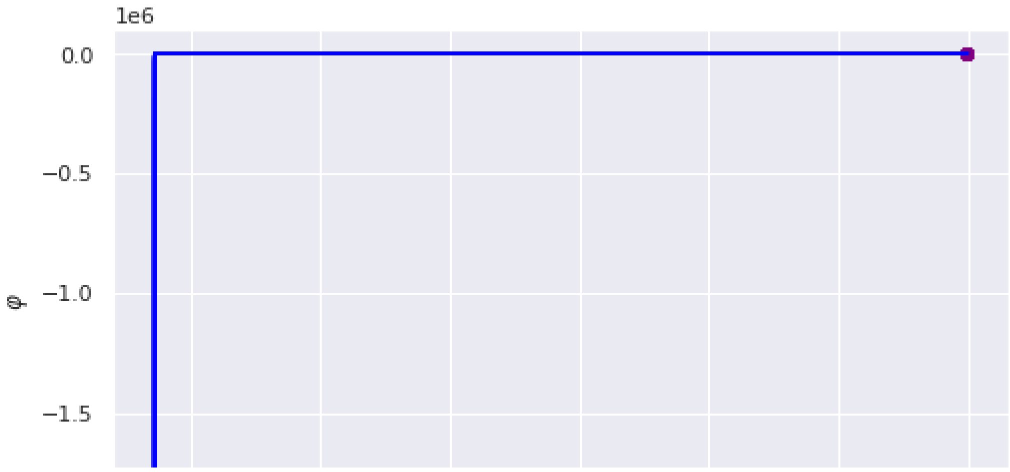
φ

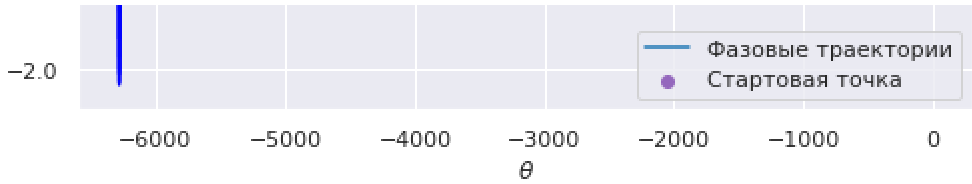
Оказывается, там целых 2 точки. Посмотрим на фазовые траектории.

In [456]:

```
for y in [[-3.5,4.970],[-3.8,4.970]]:
    plt.figure(figsize = (8,5))
    for _ in range(10):

        rk_res = sps.integrate.solve_ivp(kinetic,(0,100), bias(y), rtol = 
1e-5) #Считаем

        plt.plot(rk_res['y'][0],rk_res['y'][1], color = 'blue') #Рисуем тр
аекторию

        plt.scatter(rk_res['y'][0][0],rk_res['y'][1][0], color = 'purple')

    plt.plot([],[],c ='tab:blue', label='Фазовые траектории')
    plt.scatter([],[],c='tab:purple', label='Стартовая точка')

    plt.xlabel(r'$\theta$')
    plt.ylabel(r'$\varphi$')

    plt.legend(loc = 'lower right');
```







Опять ничего не нашли. Попробуй сделать это теоритически. Найдём особые точки для линеаризованной системы и исследуем характер поведения фазовых траекторий(файл с линеаризацией приложен к письму с решением). Линеаризовав систему, получем те же самые особые точки, что и нашли ранее.