69

```
import numpy as np

def getdA(A, beta):
    rigidity_param = np.array(
        [0.14, 0.3, beta[0], 0.12])

    weight = np.array([beta[1], 28., beta[2]])

    dA = [np.zeros_like(A),
          np.zeros_like(A),
          np.zeros_like(A),
          np.zeros_like(A)]

    dA[1][3, 2] = -1. / weight[1]
    dA[1][3, 4] = 1. / weight[1]
    dA[1][5, 2] = 1. / weight[2]
    dA[1][5, 4] = -1. / weight[2]

    dA[2][1, 0] = (rigidity_param[1] + rigidity_param[0]) / (weight[0] * weight[0])
    dA[2][1, 2] = -(rigidity_param[1]) / (weight[0] * weight[0])

    dA[3][5, 2] = -(rigidity_param[2]) / (weight[2] * weight[2])
    dA[3][5, 4] = (rigidity_param[3] + rigidity_param[2]) / (weight[2] * weight[2])

    return dA
```

70

```
def getAwithBeta(beta):
    rigidity_param = np.array(
        [0.14, 0.3, beta[0], 0.12])

    weight = np.array([beta[1], 28., beta[2]])

    A = np.zeros((6, 6))
    A[0, 1] = 1.
    A[1, 0] = -(rigidity_param[0] + rigidity_param[1]) / weight[0]
    A[1, 2] = rigidity_param[1] / weight[0]
    A[2, 3] = 1.
    A[3, 0] = rigidity_param[1] / weight[1]
    A[3, 2] = -(rigidity_param[1] + rigidity_param[2]) / weight[1]
    A[3, 4] = rigidity_param[2] / weight[1]
```

```
    A[4, 5] = 1.
    A[5, 2] = rigidity_param[2] / weight[2]
    A[5, 4] = -(rigidity_param[3] + rigidity_param[2]) / weight[2]

    return A
```

71

```
def RungeKutt(f, x, delta):
    k1 = delta * f(x)
    k2 = delta * f(x + k1/2.)
    k3 = delta * f(x + k2/2.)
    k4 = delta * f(x + k3)
    return (k1 + 2. * k2 + 2. * k3 + k4)/6.
```

72

```
initialFunc = np.loadtxt('y1.txt')

current = 1.0
previous = 0

beta = np.array([0.1, 10., 21.])

delta = np.float64(0.2)
epsilon = 1e-9
n = initialFunc.shape[1]
while np.abs(previous - current) > epsilon:
    A = getAwithBeta(beta)

    dA = getdA(A, beta)

    leftIntPart = 0.
    rightIntPart = 0.
    newBeta = 0.

    U = np.zeros((6, 3))
    y_vec = np.copy(initialFunc[:, 0].reshape(-1, 1))

    for i in range(1, n):
        delta_U = np.column_stack(((dA[1] @ y_vec).reshape(-1),
                        (dA[2] @ y_vec).reshape(-1), (dA[3] @ y_vec).reshape(-1)))

        U += RungeKutt(lambda x: A@x + delta_U, U, delta)
```

```
        # Calculate new y
        y_vec += RungeKutt(lambda x: A@x, y_vec, delta)

        leftIntPart += U.T @ U
        rightIntPart += U.T @ (initialFunc[:, i].reshape(-1, 1) - y_vec)

        newBeta += (initialFunc[:, i].reshape(-1, 1) -
                                    y_vec).T @ (initialFunc[:, i] - y_vec.reshape(-1))

    dBeta = np.linalg.inv(leftIntPart * delta) @ (rightIntPart * delta)
    beta += dBeta.reshape(-1)

    previous = current
    current = newBeta * delta

print('Результат', beta)

Результат [ 0.19999989 11.99999485 17.9999918 ]
```