**Лабораторная работа №3**

1. Цель работы

Построить параметрический рациональный сплайн с четвертым типом краевых условий через параметризацию по суммарной длине хорд для циклоиды.

1. Задание

Интерполировать циклоиду

параметрическим рациональным сплайном с четвертым типом краевых условий.

1. Алгоритм выполнения задания

*Интерполяционный рациональный сплайн* есть совокупность двух рациональных сплайнов

,

интерполирующих соответственно координаты точек кривой

В качестве параметра берем суммарную длину хорд . В этом случае сплайн на участке между точками может быть записан в виде

где – заданные числа ,

Выразим через , определим коэффициенты так, чтобы были непрерывны первая и вторая производные, и найдем формулы для их вычисления

Четвертое краевое условие выглядит следующим образом

где

Выпишем системы уравнений относительно неизвестных для четвертого типа краевых условий

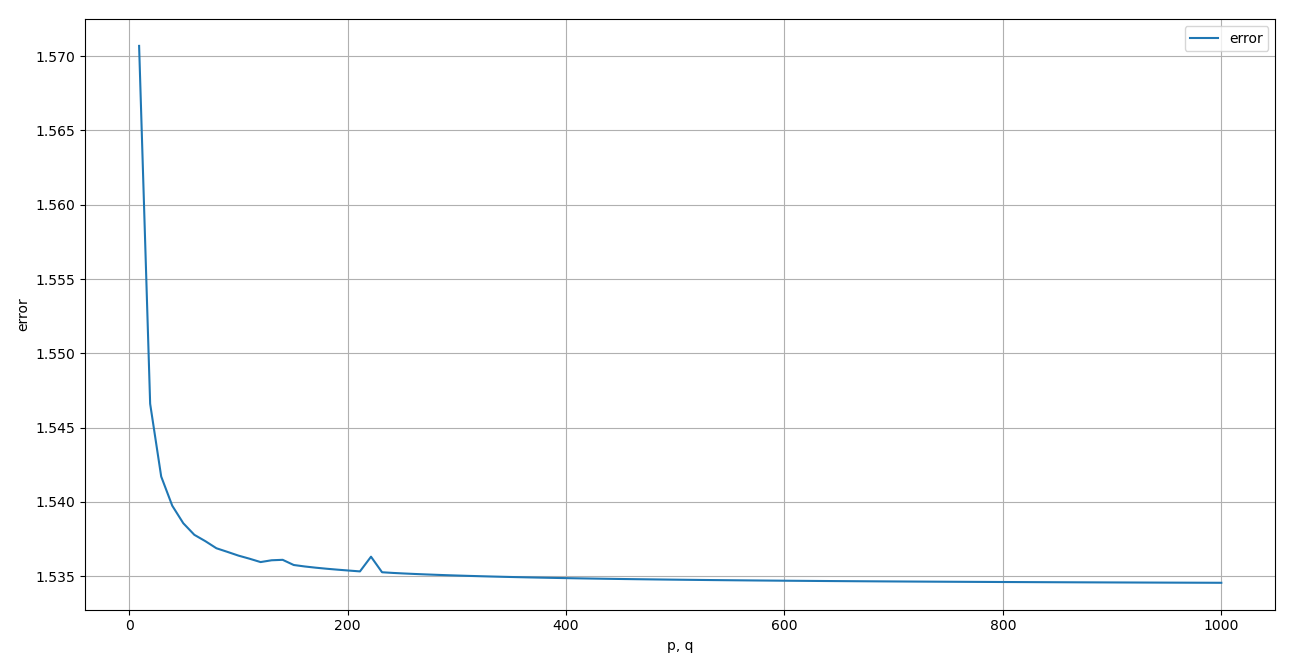
где

Сплайн строится заменой на .

1. Результат

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

Зависимость ошибки интерполяции от коэффициентов .



Ошибка интерполяции считалась по формуле:

1. Вывод

Таким образом, мы построили интерполяционный параметрический рациональный сплайн с четвертым краевым условием для циклоиды.   
Отрицательные коэффициенты ухудшают результаты интерполяции. При значении коэффициентов, равных нулю, получим интерполяцию кубическим сплайном. Для наилучшего результата интерполирования этой функции рекомендуется выбирать большие положительные коэффициенты .

1. Код программы

|  |
| --- |
| Вычисление параметрического рационального сплайна |
| def getIndex(s, s\_i):      for i in range(0, len(s\_i) - 1):          s1 = s\_i[i]          s2 = s\_i[i + 1]          if s1 <= s <= s2:              return i      assert False, "ERROR: 'i' not found!"  def rationalSpline(f, s, p, q, s\_i, d, m):      i = getIndex(s, s\_i)      coeff = (2 + q[i]) \* (2 + p[i]) - 1      C\_i = (-(3 + q[i])\*(f[i+1] - f[i]) + d[i]\*m[i] +\            (2+q[i])\*d[i]\*m[i+1]) / coeff      D\_i = ((3+p[i])\*(f[i+1]-f[i]) - d[i]\*m[i+1] -\            (2+p[i])\*d[i]\*m[i]) / coeff      A\_i = f[i+1] - C\_i      B\_i = f[i] - D\_i      t = (s - s\_i[i]) / d[i]        term1 = A\_i \* t      term2 = B\_i \* (1 - t)      term3 = (C\_i \* np.power(t, 3)) / (1 + p[i] \* (1 - t))      term4 = (D\_i \* np.power(1 - t, 3)) / (1 + q[i] \* t)      return term1 + term2 + term3 + term4 |

|  |
| --- |
| Вычисление коэффициентов рационального сплайна |
| def lam(i, d):      return d[i] / (d[i-1] + d[i])  def mu(i, d):      return 1 - lam(i, d)  def P\_i(i, p, q):      coeff = (2 + q[i]) \* (2 + p[i]) - 1      return (3 + 3 \* p[i] + pow(p[i], 2)) / coeff  def Q\_i(i, p, q):      coeff = (2 + q[i]) \* (2 + p[i]) - 1      return (3 + 3 \* q[i] + pow(q[i], 2)) / coeff  def c\_i(i, p, q, f, d):      term1 = lam(i,d) \* P\_i(i-1,p,q) \* (3+q[i-1])      term2 = (f[i] - f[i-1])/d[i-1]      term3 = mu(i,d) \* Q\_i(i,p,q) \* (3+p[i])      term4 = (f[i+1]-f[i])/d[i]      return term1 \* term2 + term3 \* term4  def getMCoeff(f, q, p, d):      n = len(f) - 1      m\_a = np.zeros((n + 1, n + 1))      m\_b = np.zeros(n + 1)      yy\_0\_2 = pow(d[0] / d[1], 2)      yy\_n\_2 = pow(d[n-1] / d[n-2], 2)      term1 = 2 \* (f[1] - f[0]) / d[0]      term2 = 2 \* yy\_0\_2      term3 = (f[2] - f[1]) / d[1]      c\_1\_\_ =  term1 - term2 \* term3        term1 = 2\*(f[n]-f[n-1])/d[n-1]      term2 = 2\*yy\_n\_2      term3 = (f[n-1]-f[n-2])/d[n-2]      c\_n\_\_ = term1 - term2 \* term3      m\_a[0][0] = 1      m\_a[0][1] = -(yy\_0\_2 - 1)      m\_a[0][2] = -yy\_0\_2      m\_b[0] = c\_1\_\_      m\_a[1][1] = lam(1,d) \* P\_i(0,p,q) \* (1 + q[0] + yy\_0\_2) +\                  mu(1,d) \* Q\_i(1,p,q) \* (2 + p[1])      m\_a[1][2] = mu(1,d) \* Q\_i(1,p,q) + lam(1,d) \* P\_i(0,p,q) \* yy\_0\_2      m\_b[1] = c\_i(1,p,q,f,d) - lam(1,d) \* P\_i(0,p,q) \* c\_1\_\_      for i in range(2, n - 1):          P\_i\_1 = P\_i(i - 1,p,q)          Q\_i\_ = Q\_i(i,p,q)          m\_a[i][i - 1] = lam(i,d) \* P\_i\_1          m\_a[i][i] = lam(i,d) \* P\_i\_1 \* (2 + q[i-1]) +\                      mu(i,d) \* Q\_i\_ \* (2 + p[i])          m\_a[i][i + 1] = mu(i,d) \* Q\_i\_          m\_b[i] = c\_i(i,p,q,f,d)        m\_a[n-1][n-2] = lam(n-1,d) \* P\_i(n-2,p,q) +\                      mu(n-1,d) \* yy\_n\_2 \* Q\_i(n-1,p,q)      m\_a[n-1][n-1] = lam(n-1,d) \* P\_i(n-2,p,q) \* (2 + q[n-2]) +\                      mu(n-1,d) \* Q\_i(n-1,p,q) \* (1 + p[n-1] + yy\_n\_2)      m\_b[n-1] = c\_i(n-1,p,q,f,d) - mu(n-1,d) \* Q\_i(n-1,p,q) \* c\_n\_\_      m\_a[n][n-2] = -yy\_n\_2      m\_a[n][n-1] = -(yy\_n\_2 - 1)      m\_a[n][n] = 1      m\_b[n] = c\_n\_\_      return np.linalg.solve(m\_a, m\_b) |

|  |
| --- |
| Интерполяция |

def d(i, x, y):

    return np.sqrt(pow(x[i+1] - x[i], 2) + pow(y[i+1] - y[i], 2))

def cycloid(r, t):

    x = r \* t - r \* np.sin(t)

    y = r - r \* np.cos(t)

    return x, y

def main():

    n = 10

    p = q = np.ones(n + 1) \* 50

    t = np.linspace(0, 2\*np.pi, n+1)

    x, y = cycloid(1, t)

    d\_ = np.zeros(n)

    for i in range(n):

        d\_[i] = d(i, x, y)

    s\_ = np.zeros(n + 1)

    for i in range(n + 1):

        s\_[i] = sum([d\_[j] for j in range(0, i - 1)])

    m\_x = getMCoeff(x, q, p, d\_)

    m\_y = getMCoeff(y, q, p, d\_)

    tt = np.linspace(0, s\_[-1], 10)

    x\_s = np.array([rationalSpline(x, si, p, q, s\_, d\_, m\_x) for si in tt])

    y\_s = np.array([rationalSpline(y, si, p, q, s\_, d\_, m\_y) for si in tt])

    plt.plot(x, y, label="func", lw=3)

    plt.plot(x\_s, y\_s, label="spline")

    plt.legend()

    plt.show()

main()