سید علیرضا میرعابدینی کچومثقالی

۴۰۱۴۱۴۰۲۴ ([گیت‌هاب](https://github.com/salirezamir/))

خلاصه

توضیحاتی در رابطه با این الگوریتم و کد عمل کرد آن

درون یاب اسپلاین خطی

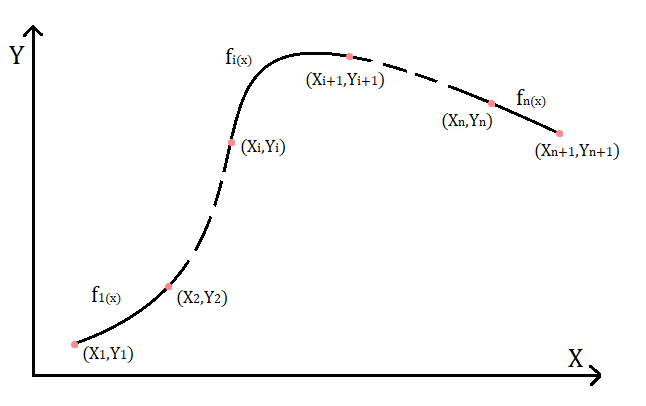
پروژه کلاس محاسبات عددی استاد نیکان

# مقدمه

درون یابی اسپلاین مکعبی یک روش ریاضی مرسوم برای ساخت معادله نقاطی هست که درون مرز نقاط قرار داده شده بگنجد که این نقاط جدید از تابع درون یاب تکه ای که به آنها اسپلاین گفته می‌شود استخراج شده‌اند.

# شروع

با توجه به تعریف داریم که اگر (n+1) نقطه به عنوان ورودی داشته باشیم آنگاه تابع درون یابمان به فرم زیر می‌شود :



و بنا بر اینکه انحنا در نقاط نباید به هم بخورد داریم :

و دو سر بازه باید مشتق دوم صفر بشود پس داریم :

# الگوریتم محاسبه

1. به دست آوردن معادلات نقاط داده شده :

با توجه به معادلات زیر :

که 2n معادله به ما می‌دهد.

1. به دست آوردن معادلات مشتق اول :

با استفاده از معادله (۱) داریم :

که n-1 معادله به ما می‌دهد.

1. به دست آوردن معادلات مشتق دوم:

با استفاده از معادله (۲) داریم :

که n-1 معادله به ما می‌دهد.

1. معادلات شرایط مرزی ابتدا و انتها بازه :

با استفاده از معادله (۳) و کمک از معادله (۶) داریم :

1. تشکیل دستگاه برای حل معادلات :

به دلیل بزرگی ماتریس دستگاه ، دستگاهی با n=2 برای مثال آورده شده است:

که ردیف ۰ تا ۳ مربوط به معادلات (۳) ، ردیف ۴ مربوط به معادلات (۵) ،ردیف ۵ مربوط به معادلات (۷) و ردیف ۸ تا ۹ مربوط به معادلات (۸) است.

برای حل دستگاه ابتدا برای سادگی در نوشتار آن را به صورت زیر می‌نویسیم:

در نتیجه اگر دو طرف را در معکوس ماتریس A ضرب کنیم داریم:

در نتیجه ثوابت که در ماتریس C هستند به دست می‌آیند.

# توضیحات کد

1. خط اول کد:

import numpy as np

با استفاده از کتابخانه numpy می‌توانیم از متغییر های آرایه ای و راه کار های جبر خطی برای آنها استفاده کنیم .

1. تابع Cube\_Spline

این تابع به عنوان ورودی نقاط داده شده را گرفته و بعنوان خروجی ماتریسی که ثوابت تکه خم ها هست را ارئه می‌دهد.

def cube\_spline(x, y):

    n = len(x) - 1

    A = np.zeros((4 \* n, 4 \* n))

    b = np.zeros(4 \* n)

در ابتدای تعریف این تابع اول n که تعداد نقاط منهای یک هست را با استفاده از تابع len() که خروجی آن تعداد اعضا داخل آرایه هست بدست می‌آوریم سپس ماتریس های A و b که در مرحله پنجم از الگوریتم تعریف کردیم را تعریف و تمامی خانه های آن را برابر ۰ قرار می‌دهیم.

    # First's and End's of S(i)'s=Yi's

    for i in range(n):

        for j in range(4):

            A[2 \* i, 4 \* i + j] = x[i] \*\* (3 - j)

            A[2 \* i + 1, 4 \* i + j] = x[i + 1] \*\* (3 - j)

        b[2 \* i] = y[i]

        b[2 \* i + 1] = y[i + 1]

سپس شروع به پرکردن ماتریس مرحله پنج می‌شویم ابتدا از مرحله اول الگوریتم شروع کرده و مطابق دستورالعمل ارئه شده عمل می‌کنیم.

    # First derivation

    for i in range(n - 1):

        A[i + 2 \* n, 4 \* i] = 3 \* x[i + 1] \*\* 2

        A[i + 2 \* n, 4 \* i + 1] = 2 \* x[i + 1]

        A[i + 2 \* n, 4 \* i + 2] = 1

        A[i + 2 \* n, 4 \* (i + 1)] = -3 \* x[i + 1] \*\* 2

        A[i + 2 \* n, 4 \* (i + 1) + 1] = -2 \* x[i + 1]

        A[i + 2 \* n, 4 \* (i + 1) + 2] = -1

حال به مرحله دوم از الگوریتم رفته و شروع به پر کدن خانه های لازم طبق دستورالعمل می‌شویم.

    for i in range(n - 1):

        A[i + 3 \* n - 1, 4 \* i] = 6 \* x[i + 1]

        A[i + 3 \* n - 1, 4 \* i + 1] = 2

        A[i + 3 \* n - 1, 4 \* (i + 1)] = -6 \* x[i + 1]

        A[i + 3 \* n - 1, 4 \* (i + 1) + 1] = -2

    A[4 \* n - 2, 0] = 6 \* x[0]

    A[4 \* n - 1, 4 \* (n - 1)] = 6 \* x[n]

    A[4 \* n - 2, 1] = 2

    A[4 \* n - 1, 4 \* (n - 1) + 1] = 2

اکنون به مرحله سوم رفته و شروع به پر کدن خانه های لازم طبق دستورالعمل می‌شویم.

return np.dot(np.linalg.inv(A), b)

و در مرحله آخر با استفاده توابع جبر خطی و مطابق مرحله ۶ ام دستورالعمل معکوس ماتریس A را بدست آورده و در ماتریس b ضرب می‌کنیم.

1. تابع درون یاب

def interpolation(x\_0, x, y):

    n = len(x)

    j = 0

    for i in range(n):

        if x[i] >= x\_0:

            j = i - 1

            break

ابتدا تابع درون یاب را تعریف و سپس این تابع سعی میکند که بخشی که نقطه مورد نظر در آن قرار دارد را پیدا و شماره بخش ان را در متغیر j قرار می‌دهد

    C = cube\_spline(x, y)

    return (

        C[4 \* j] \* x\_0\*\*3

        + C[4 \* j + 1] \* x\_0\*\*2

        + C[4 \* j + 2] \* x\_0

        + C[4 \* j + 3]

    )

سپس با تابع cube\_spline که در بالا توضیح داده شد ثوابت را پیدا و با استفاده از ثوابت بخش مورد نظر و جاگذاری آد در معادله y نقطه داده شده را بدست می‌آوریم

1. بخش تعاریف

x = np.array([0, 1, 2, 3, 4])

y = np.array([21, 24, 24, 18, 16])

print(interpolation(x\_0, x, y))

‌x : محل محور افقی نقاط ورودی

‌y : محل محور عمودی نقاط ورودی

X\_0 : نقطه ای که نقطه درون یاب آن را میخواهیم بیابیم (این عدد به عنوان مثال قرار داده شده است)

# چند مثال عددی

1. سهمی به معادله y=x2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 4 | 3 | 2 | 1 | 0 | x |
| 16 | 9 | 4 | 1 | 0 | y |

به عنوان ورودی به این الگوریتم می‌دهید و خروجی درون یاب نقطه 3.5 را از آن میگیریم که به صورت ایده‌آل باید برار 12.25 بشود و تا چهار رقم اعشار خروجی برابر 12.3393 می‌شود.

1. Y=3x3+2x

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 5 | 4 | 3 | 2 | 1 | 0 | x |
| 385 | 200 | 87 | 28 | 5 | 0 | y |

به عنوان ورودی به این الگوریتم می‌دهید و خروجی درون یاب نقطه 2.5 را از آن میگیریم که به صورت ایده‌آل باید برار 51.875بشود و تا چهار رقم اعشار خروجی برابر 52.1711 می‌شود.

1. Y=5+4x4+3x2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 5 | 4 | 3 | 2 | 1 | 0 | x |
| 2580 | 1077 | 356 | 81 | 12 | 5 | y |

به عنوان ورودی به این الگوریتم می‌دهید و خروجی درون یاب نقطه 4.5 را از آن میگیریم که به صورت ایده‌آل باید برار 1706 بشود و تا چهار رقم اعشار خروجی برابر 1760.5622 می شود.

# مقایسه این روش با روش نیوتون رافسون

برای مقایسه ابتدا همان ورودی های مثال های بخش قبلی را به همان الگوریتم می‌دهیم.

|  |  |  |  |
| --- | --- | --- | --- |
| جواب واقعی | نتیجه اسپلاین مکعبی | نتیجه نیوتون رافسون | شماره مثال |
| 12.25 | 12.3393 | 12.0547 | ۱ |
| 51.875 | 52.1711 | 51.875 | ۲ |
| 1706 | 1760.5622 | 1706.0 | ۳ |

از لحاظ سرعت قابل رویت هست که این روش بهتر از نیوتون رافسون عمل کرد ولی از لحاظ جواب آخر به نظر می آید نیوتون رافسون نتایج قابل قبول تری دارد

ولی دو نکته وجود دارد که کاربرد این دو روش را از هم جدا می‌کند

1. اینکه نیوتون رافسون نیاز مند حدس اول هست در صورتی که این روش خم مورد نظر را به صورت تکه ای بدون حدس اولیه اعلام میکند
2. این روش به ما معاله خم می‌دهد ولی روش نیوتون رافسون معادله ای به ما
3. نمیدهد و صرفا نقطه ای را به ما اعلام می‌کند

# نتیجه گیری

به نظر برای مواقعی که خم نیاز داشته باشیم این روش بهتر است و زمانی که صرفا یک نقطه را نیاز داشته باشیم نیوتون رافسون.