روش های لاگرانژ

مریم محمدی ملیکا شیخی

۱۵ خرداد ۱۴۰۳

نمایه

چکیده

در بهینه‌سازی ریاضی، روش ضرایب لاگرانژ یک استراتژی برای یافتن ماکسیمم‌ها و مینیمم‌های محلی یک تابع تحت محدودیت‌های معادله‌ای است (یعنی با توجه به شرطی که یک یا چند معادله باید دقیقاً با مقادیر انتخابی متغیرها ارضا شوند). این روش به نام ریاضیدان ژوزف-لویی لاگرانژ نامگذاری شده است.

پیش گفتار

قبلا در توابع یک متغیره ما با صفر قرار دادن مشتق تابع min یا max را پیدا می کردیم . ایده این بود که تابع در جایی به max یا $localmin$ می رسد که خط مماس آن صاف است. همین مفهوم را برای توابع چند متغیره نیز داریم. به گونه ای که گرادیان را برابر صفر قرار داده که معادل برابر قرار دادن تمامی مشتقات جزئی با صفر است و سپس معادلات را حل می‌کنیم. این روش تا حدودی خوب کار می‌کند، اما غالباً در شرایط دنیای واقعی ما محدودیت‌های اضافی داریم که مقادیر متغیرهای ما را محدود می‌کنند. این به‌این معناست که متغیرهای مستقل x و y آزاد نیستند که در هر جایی از صفحه حرکت کنند و به ناحیه یا منحنی خاصی محدود می‌شوند و اجازه ندارند مقادیری خارج از آن بگیرند. حال برعکس به همین دلیل، گراف تابع ما اکنون سطحی با مرزهایی است و بنابراین ماکسیمم یا مینیمم مطلق تابع می‌تواند روی آن اتفاق بیفتد. این می‌تواند مشکل‌ساز باشد زیرا اگر ماکسیمم یا مینیمم روی مرز رخ دهد، گرادیان لازم نیست که در آنجا صفر باشد، یعنی سطح لازم نیست که در آنجا صاف باشد. این درست مانند زمانی است که در حساب دیفرانسیل یک متغیره برای پیدا کردن ماکسیمم یا مینیمم تابع در یک بازه بسته، باید نقاط انتهایی یا مرزهای بازه را نیز علاوه بر نقاط صافبررسی می‌کردیم زیرا منحنی لازم نبود که در نقاط انتهایی صاف باشد. در حساب دیفرانسیل یک متغیره، بررسی یک مرز نسبتاً ساده بود زیرا با توابع یک متغیره ما فقط دو نقطه مرزی برای بررسی داشتند، نقطه انتهایی چپ و نقطه انتهایی راست، بنابراین می‌توانستیم تابع را در هر دو ارزیابی کنیم تا ببینیم آیا هیچ‌کدام از آن‌ها ماکسیمم یا مینیمم مطلق هستند یا خیر. اما با توابع چندمتغیره، مرز یک منحنی است و یک منحنی حاوی بی‌نهایت نقطه است که به این معناست که نمی‌توانیم همه آن‌ها را یکی یکی وارد کنیم تا ببینیم کدام یک بالاترین و کدام یک پایین‌ترین است. به یاد داشته باشید که هر زمان که یک مشکل ماکسیمم یا مینیمم را حل می‌کنیم، از یک معیار استفاده می‌کنیم که نشان می‌دهد ماکسیمم یا مینیمم کجا می‌تواند رخ دهد. در حساب دیفرانسیل یک متغیره یا در مورد پارامتری‌کردن منحنی مرزی، آن معیار برابر صفر قرار دادن مشتق منحنی بود. منطق این بود که اگر یک تابع در وسط یک منحنی ماکسیمم یا مینیمم داشته باشد، باید در آنجا صاف باشد. بنابراین آنچه ما به دنبال آن هستیم، یک معیار دیگر است که نشان‌دهنده ماکسیمم یا مینیمم روی منحنی مرزی است و نیازی به گرفتن مشتق در طول خود منحنی ندارد که نیازمند پارامتری‌کردن منحنی است. چگونه می‌توانیم این کار را انجام دهیم؟ ماکسیمم یا مینیمم یک تابع $f(x,y)$ با توجه به محدودیت $g(x,y)=k$ باید در جایی رخ دهد که گرادیان f موازی با گرادیان g باشد. در واقع، یک تغییر کوچک می‌توانیم در این بیان ایجاد کنیم که آن را برای حل یک

مسئله عینی قابل استفاده‌تر کند. به یاد داشته باشید که اگر دو بردار موازی باشند، به این معناست که یکی یک ضریب اسکالر از دیگری است. بنابراین، راه دیگری که می‌توانیم این معیار را بیان کنیم این است که بگوییم گرادیان f برابر است با یک ضریب اسکالر ثابت λ ضربدر گرادیان g. این ضریب ثابت λ در واقع ضریب لاگرانژ نامیده می‌شود و از اینجاست که این تکنیک نام خود را گرفته است. خب، حالا که این معیار را داریم، چگونه از آن برای حل یک مسئله واقعی استفاده کنیم؟ ایده این است که این معیار که گرادیان fبرابر است با یک ضریب ثابت λ ضربدر گرادیان g را بگیریم و آن را به یک دستگاه معادلات تبدیل کنیم با مساوی کردن اجزای مشابه. این را با معادله$g(x,y)=k$ برای خود محدودیت ترکیب کنیم و یک دستگاه سه معادله و سه مجهول خواهیم داشت. راه‌حل‌های این دستگاه، برای جایی که تابع f در طول منحنی مرزی آن به حداکثر یا حداقل می‌رسد، خواهند بود و همانطور که معمولاً انجام می‌دهیم، جوابی که بالاترین مقدار f را می‌دهد ماکسیمم است و جوابی که کمترین مقدار f را می‌دهد مینیمم است.

مقدمه

ایده اصلی این است که یک مسئله محدود شده را به شکلی تبدیل کنیم که بتوان آزمون مشتق مسئله نامحدود را همچنان اعمال کرد. رابطه بین گرادیان تابع و گرادیان محدودیت‌ها به طور طبیعی به بازفرمول‌بندی مسئله اصلی منجر می‌شود که به عنوان تابع لاگرانژ یا لاگرانژ شناخته می‌شود. در حالت کلی، لاگرانژی به صورت زیر تعریف می‌شود:

$$L\left(x,\lambda\right)\equiv f\left(x\right)+\left(\lambda,g\left(x\right)\right)$$

برای توابع $f(x),g(x),\lambda$ ضریب لاگرانژ است.در موارد ساده، جایی که ضرب نقطه‌ای به عنوان ضرب داخلی تعریف شده است، لاگرانژ به صورت زیر است:

$$L(x,\lambda)\equiv f(x)+\lambda.g(x)$$

این روش به صورت زیر خلاصه می‌شود: برای پیدا کردن حداکثر یا حداقل یک تابع$f(x)$ که تحت محدودیت تساوی $g(x)=0$ قرار دارد، نقاط ایستای L را به عنوان تابعی از x و ضریب لاگرانژ λ پیدا کنید. این به این معناست که تمام مشتقات جزئی باید صفر باشند، از جمله مشتق جزئی نسبت بهλ

$$\frac{\partial L}{\partial x}=0,\qquad \frac{\partial L}{\partial \lambda}=0$$

ویا :

$$\frac{\partial L}{\partial x}+\lambda\frac{\partial g}{\partial x}=0,\qquad g(x)=0$$

راه حل مربوط به بهینه‌سازی مقید اصلی همیشه یک نقطه زینی از تابع لاگرانژ است که می‌توان آن را از میان نقاط ثابت با تعیین قطعیت ماتریس هسیان مرزی شناسایی کرد. مزیت بزرگ این روش این است که بهینه‌سازی را بدون پارامتری‌سازی صریح در قالب قیود ممکن می‌سازد. به همین دلیل، روش ضرایب لاگرانژ به طور گسترده‌ای برای حل مسائل چالش‌برانگیز بهینه‌سازی مقید استفاده می‌شود. علاوه بر این، روش ضرایب لاگرانژ با شرایط کاروش-کان-تاکر تعمیم داده می‌شود که می‌تواند قیود نامساوی به شکل $h(x) \leq c$ برای یک ثابت داده‌شده c را نیز در نظر بگیرد.

توضیح مسئله

این مطلب به عنوان قضیه چندجمله‌ای لاگرانژ شناخته می‌شود.اگر تابع هدف را $f : R^n \to R$ تعریف کنیم و $g : R^n \to R^c$ تابع محدودیت‌ها باشد و هر دو به C^1 تعلق داشته باشند (به‌این معنا که اشتقاق‌های اولیه شان پیوسته باشند) فرض کرده یک راه‌حل بهینه برای مسئله بهینه‌سازی زیر باشد به‌طوری که، برای ماتریس اشتقاق‌های جزئی $[Dg(x\star)]_{j,k} = \frac{\partial g_j}{\partial x_k}$

$$rank(Dg(x\star) = c \leq n : \qquad maximize f(x) \qquad s.t g(x) = 0$$

سپس یک ضریب لاگرانژ منحصربفرد $\lambda\star \in R^c$ وجود دارد به‌طوری که $Df(x\star) = \lambda \star^t$ $Dg(x\star)$ (توجه شود که $\lambda\star$ به‌وضوح به‌عنوان یک بردار ستونی در نظر گرفته می‌شود تا اطمینان حاصل شود که ابعاد مطابقت دارند. اما، می‌توانیم آن را به‌عنوان یک بردار ردیفی در نظرگرفته و معکوس کنیم).

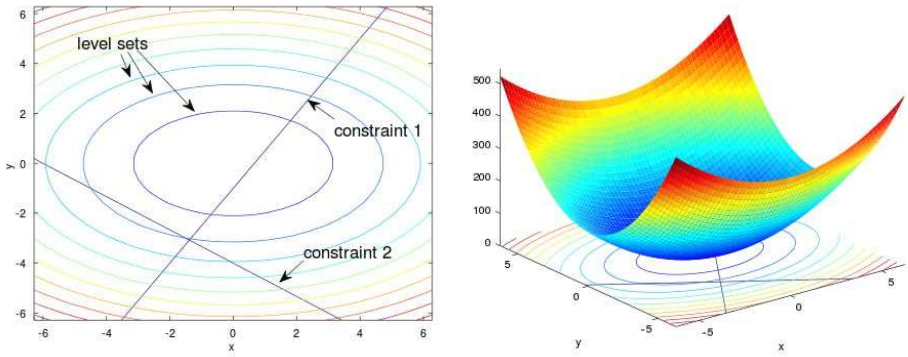
قضیه چند جمله‌ای لاگرانژ

این قضیه بیان می‌کند که در هر بیشینه محلی (یا کمینه) از تابع مقداردهی شده تحت محدودیت‌های برابری، اگر شرایط محدودیت اعمال شود، آنگاه گرادیان تابع (در آن نقطه) می‌تواند به‌عنوان یک ترکیب خطی از گرادیان محدودیت‌ها (در آن نقطه) بیان شود، با ضرایب عمل کننده به عنوان ضرایب لاگرانژ. این معادل است با گفتن اینکه هر جهت عمود بر همه گرادیان محدودیت‌ها نیز عمود بر گرادیان تابع است. یا هنوز، گفتن اینکه مشتق جهتی تابع در هر جهت قابل اجرا است

مسئله با چند محدودیت :

روش ضریب‌های لاگرانژ می‌تواند گسترش یابد تا مسائل با چندین محدودیت با استفاده از یک استدلال مشابه حل شود. یک زیرمجموعه مکعبی را در نظر بگیرید که متعلق به دو محدودیت خطی است که در یک نقطه تقاطع می‌کنند. به عنوان تنها راه‌حل ممکن، این نقطه به‌وضوح یک اکسترمم محدود شده است. با این حال، مجموعه سطحی از به‌وضوح موازی با هر یک از محدودیت‌ها در نقطه تقاطع نیست (مشاهده

شکل ۱)؛ به‌جای آن، این یک ترکیب خطی از گرادیان دو محدودیت است. در صورت محدودیت‌های چندگانه، به طور کلی همان چیزی است که ما به دنبالش هستیم: روش لاگرانژ به دنبال نقاطی می‌گردد که گرادیان لزوماً ضربی از گرادیان هر محدودیت تکراری نیست، بلکه در آن ترکیب خطی از گرادیان‌های تمامی محدودیت‌ها است.



به‌طور مشخص، فرض کنید محدودیت داشته باشیم و در حال پیمایش مجموعه نقاطی که $g_i(x) = 0$ $i = 1, .., M$ را برآورده می‌کنند. هر نقطه در حاشیه تابع محدودیت داده شده g_i یک فضای جهت قابل قبول دارد: فضایی از بردارهای عمود بر ∇g_i . مجموعه جهت‌هایی که توسط تمام محدودیت‌ها قابل قبول است بنابراین فضای جهت‌هایی است که عمود بر تمامی گرادیان‌های محدودیت‌ها است. این فضا را با نشان می‌دهیم و اسپن گرادیان‌های محدودیت‌ها را با □ نشان می‌دهیم. پس ، فضایی از بردارهای عمود بر هر عنصر از $A = S^{\perp}S$. هنوز به دنبال یافتن نقاطی هستیم که هنگام حرکت ما تغییر نمی‌کند، زیرا این نقاط ممکن است اکسترمم‌های (محدود شده) باشند. بنابراین، ما به دنبال هستیم به گونه‌ای که هر جهت قابل قبول حرکت دوری از عمود بر $\nabla f(x)$ باشد (در غیر این صورت می‌توانیم با حرکت در آن جهت ممکن را افزایش دهیم). به عبارت دیگر، $f(x) \in A^{\perp} = \nabla S$. بنابراین مقادیر $\lambda 1, \lambda 2, ..., \lambda M$ وجود دارد که

$$\nabla f(x) = \sum_{k=1}^{M} \lambda_k \nabla g_k(x\star) \Leftrightarrow \nabla f(x) - \sum_{k=1}^{M} \lambda_k \nabla g_k(X) = 0$$

این اعداد متغیرها ضرایب لاگرانژ هستند. اکنون از آن‌ها داریم، یکی برای هر محدودیت.

مانند قبل ، یک تابع کمکی معرفی می‌کنیم

$$L(x_1, ..., \lambda_1, ..., \lambda_M) = f(x_x, ..., x_x) - \sum_{k=1}^{M} \lambda_k g_k(x_1, ..., x_n)$$

$$\nabla_{x_1,...,x_n,\lambda_1,...,\lambda_M} L(x_1, ..., x_n, \lambda_1, ..., \lambda_M) = 0 \Leftrightarrow$$
$$\nabla f(x) - \sum_{k=1}^{M} \lambda_k \nabla g_k(x) = 0 \qquad g_1(x) = ... = g_M(x) = 0$$

که معادل حل کردن + معادله در + مجهول است. فرضیه محدودیت-کیفیت در صورت وجود محدودیت‌های چندگانه این است که گرادیان‌های محدودیت در نقطه مربوطه مستقل خطی هستند.

مسائل کاربردی

الگوریتم

l1.py > dfunc

```python
import numpy as np
from scipy.optimize import fsolve

def func(X) :
    x = X[0]
    y = X[1]
    l = X[2] # This is the multiplier

    return x + y + l * (x**2 + y**2 - 1)

def dfunc(X) :
    dLambda = np.zeros(len(X))
    h = 1e-3 # This is the step size used in the finite difference
    for i in range(len(X)):
        dX = np.zeros(len(X))
        dX[i] = h
        dLambda[i] = (func(X+dX)-func(X-dX))/(2*h)
    return dLambda

# This is the max
X1 = fsolve(dfunc, [1, 1, 0])
print (X1, func(X1))

# This is the min
X2 = fsolve(dfunc, [-1, -1, 0])
print (X2, func(X2))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS D:\Melika\Uni\Term 6\Behinesazai Gheire Khati\erae>  & 'c:\Users\ASuS\anaco
024.6.0-win32-x64\bundled\libs\debugpy\adapter/../..\debugpy\launcher' '2942'
[ 0.70710678  0.70710678 -0.70710678] 1.4142135623730951
[-0.70710678 -0.70710678  0.70710678] -1.414213562373095
```

برای پیاده سازی کد این روش از دو کتابخانه $numpy$ و $scipy$ استفاده می‌کنیم.

تابع $func$ یک آرایه X به عنوان ورودی می‌گیرد که شامل متغیرها و ضرایب لاگرانژ است تابع مقدار بازگشتی ترکیبی از این متغیرها است که به نوعی معادله هدف را تشکیل می‌دهد. تابع $dfunc$ ، مشتق عددی $func$ نسبت به هر یک از متغیرهای X را محاسبه می‌کند. $dLambda$ یک آرایه صفر به اندازه X ایجاد می‌کند که برای ذخیره مشتقات استفاده می‌شود. h گام کوچک برای محاسبه مشتق به روش تفاضل مرکزی ($finite \quad difference$) است. درون حلقه for، به ترتیب برای هر عنصر X - dX یک آرایه صفر به اندازه X است. مقدار h به i مین عنصر dX اضافه می‌شود.

مشتق نسبت به i امین متغیر با استفاده از فرمول تفاضل مرکزی محاسبه می‌شود :

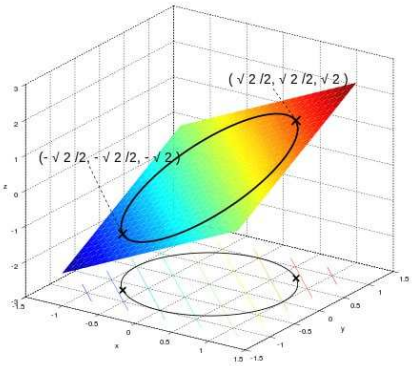
$$f(x) = f(x + \frac{1}{2}dx) - f(x - \frac{1}{2}dx)$$

در نهایت $dLambda$ که شامل مشتقات است، بازگردانده می‌شود. $fsolve$ از $scipy.optimize$ برای حل معادلات غیرخطی استفاده می‌شود. $dfunc$ به عنوان تابعی که باید حل شود به $fsolve$ داده می‌شود.

$fsolve$ تلاش می‌کند تا مجموعه‌ای از متغیرها را پیدا کند که در آن$dfunc$ صفر باشد (یعنی مشتقات تابع لاگرانژیان نسبت به هر متغیر صفر شود، که نشان‌دهنده نقاط بحرانی یا نقاط حداقل/حداکثر تابع است).

- در نهایت، $X1$ که راه‌حل به دست آمده از $fsolve$ است، به همراه مقدار تابع$func$ در آن نقطه چاپ می‌شود.

این کد در کل به دنبال یافتن نقاط بحرانی تابع لاگرانژیان با استفاده از مشتقات عددی و حل معادلات غیرخطی است.



پیچیدگی زمانی این الگوریتم$O(n^2)$است

```python
import numpy as np
from scipy.optimize import fsolve

def func(X) :
    x = X[0]
    y = X[1]
    l = X[2] # This is the multiplier

    return 2*x + x*y + 3*y + l * (x**2 + y - 3)

def dfunc(X) :
    dLambda = np.zeros(len(X))
    h = 1e-3 # This is the step size used in the finite difference
    for i in range(len(X)):
        dX = np.zeros(len(X))
        dX[i] = h
        dLambda[i] = (func(X+dX)-func(X-dX))/(2*h)
    return dLambda

# This is the max
X1 = fsolve(dfunc, [1, 1, 0])
print (X1, func(X1))

# This is the min
X2 = fsolve(dfunc, [-1, -1, 0])
print (X2, func(X2))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS D:\Melika\Uni\Term 6\Behinesazai Gheire Khati\erae>  & 'c:\Users\ASuS\anaco
\bundled\libs\debugpy\adapter/../..\debugpy\launcher' '2956' '--' 'd:\Melika\U
[ 0.63299316  2.59931966 -3.63299316] 10.709296863229078
[-2.63299316 -3.93265299 -0.36700684] -6.709296863229079
```

نیوتون-رافسون

```python
x1, x2, x3, lambda1, lambda2 = sp.symbols('x1 x2 x3 lambda1 lambda2')


# Lagrange func
L = x1 + x2 + x3 + lambda1 * (x1**2 + x2 - 3) + lambda2 * (x1 + 3*x2 + 2*x3 - 7)

# Gradient
grad_L = [sp.diff(L, var) for var in (x1, x2, x3, lambda1, lambda2)]

# Hessian matrix
jacobian_L = sp.Matrix(grad_L).jacobian([x1, x2, x3, lambda1, lambda2])


grad_L_func = sp.lambdify((x1, x2, x3, lambda1, lambda2), grad_L, 'numpy')
jacobian_L_func = sp.lambdify((x1, x2, x3, lambda1, lambda2), jacobian_L, 'numpy')


def newton_raphson(x0, tolerance=1e-3, max_iterations=3):
    x_n = np.array(x0, dtype=float)
    for i in range(max_iterations):
        grad = np.array(grad_L_func(*x_n), dtype=float).flatten()
        H = np.array(jacobian_L_func(*x_n), dtype=float)
        if np.linalg.norm(grad, ord=2) < tolerance:
            print(f"Converged to: {x_n}")
            L_opt = lagrangian(*x_n)
            print(f"Optimal Lagrangian value: {L_opt}")
            return x_n
        delta_x = np.linalg.solve(H, -grad)
        x_n = x_n + delta_x
        print(f"Iteration {i + 1}: {x_n}")
    L_opt = lagrangian(*x_n)
    print("Did not fully converge after the maximum number of iterations")
    print(f"Optimal Lagrangian value: {L_opt}")
    return x_n


lagrangian = sp.lambdify((x1, x2, x3, lambda1, lambda2), L, 'numpy')

# Initial guess
x0 = [0.5, 0.5, 0.5, 0.5, 0.5]


result = newton_raphson(x0)
print("Result:", result)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS D:\Melika\Uni\Term 6\Behinesazai Gheire Khati\erae> & 'c:\Users\ASuS\anaconda3\python.exe' 'c:\l
re Khati\erae\16.py'
Iteration 1: [-0.5    3.75  -1.875  0.5   -0.5  ]
Iteration 2: [-0.5    2.75  -0.375  0.5   -0.5  ]
Converged to: [-0.5    2.75  -0.375  0.5   -0.5  ]
Optimal Lagrangian value: 1.875
Result: [-0.5    2.75  -0.375  0.5   -0.5  ]
```

این کد مربوط به حل مسئله لاگرانژ به کمک روش نیوتن-رافسون است.

در اینجا، کتابخانه‌های $numpy$ و $sympy$ برای انجام محاسبات عددی و سمبلیک وارد می‌شوند. $numpy$

برای محاسبات عددی و $sympy$ برای محاسبات سمبلیک استفاده می‌شود. ابتدا ، متغیرهای سمبلیک با استفاده از تابع $symbols$از کتابخانه $sympy$ تعریف می‌شوند. این متغیرها در تعریف تابع لاگرانژ استفاده خواهند شد. سپس ، تابع لاگرانژ □ تعریف می‌شود. این تابع ترکیبی از تابع هدف و محدودیت‌ها است. بعد از آن گرادیان تابع لاگرانژ Lرا محاسبه می کند. گرادیان مشتقات جزئی L نسبت به هر یک از متغیرها است. همچنین، ماتریس ژاکوبین گرادیان Lمحاسبه می‌شود که شامل مشتقات جزئی مرتبه دوم (هسیان) تابع Lنسبت به تمامی متغیرها است. سپس، توابع گرادیان و ژاکوبین به توابع عددی تبدیل می‌شوند که با استفاده از$numpy$ قابل محاسبه هستند. این تبدیل به کمک تابع $lambdify$از کتابخانه$sympy$انجام می‌شود.بخش بعد، تابع نیوتن-رافسون را تعریف می‌کند که برای یافتن نقطه بهینه تابع لاگرانژ استفاده می‌شود: $x0$ مقدار اولیه حدس برای متغیرها است. $tolerance$ میزان دقت مورد نظر برای همگرایی است. $max_iterations$ حداکثر تعداد تکرارها را مشخص می‌کند. x_n به عنوان مقدار فعلی متغیرها در هر تکرار نگه‌داری می‌شود. در هر تکرار، گرادیان و ژاکوبین در مقدار فعلیx_n محاسبه می‌شوند. اگر مقدار گرادیان کمتر از$tolerance$ باشد، روش متوقف شده و مقدار بهینه‌ی لاگرانژ چاپ و برگردانده می‌شود. در غیر این صورت، با استفاده از حل دستگاه معادلات خطی، به روز رسانیx_n انجام می‌شود. بعد از اتمام تکرارها، اگر همگرایی حاصل نشد، مقدار فعلی و لاگرانژین محاسبه و چاپ می‌شود. در آخر، تابع لاگرانژ L به تابع عددی تبدیل می‌شود که با استفاده از$numpy$ قابل محاسبه است. مقدار اولیه برای هر متغیر تعیین می‌شود و در نهایت، روش نیوتن-رافسون با مقدار اولیه $x0$ اجرا می‌شود و نتیجه بهینه به دست آمده چاپ می‌شود. پیچیدگی زمانی این الگوریتم $O(n^3)$ است.

مثالی دیگر

```python
import numpy as np
import sympy as sp

x1, x2, lambda1 = sp.symbols('x1 x2 lambda1')


# Lagrange func
L = 2*x1 + x1*x2 + 3*x2 + lambda1 * (x1**2 + x2 - 3)

# Gradient
grad_L = [sp.diff(L, var) for var in (x1, x2, lambda1)]

# Hessian matrix
jacobian_L = sp.Matrix(grad_L).jacobian([x1, x2, lambda1])


grad_L_func = sp.lambdify((x1, x2, lambda1), grad_L, 'numpy')
jacobian_L_func = sp.lambdify((x1, x2, lambda1), jacobian_L, 'numpy')


def newton_raphson(x0, tolerance=1e-3, max_iterations=3):
    x_n = np.array(x0, dtype=float)
    for i in range(max_iterations):
        grad = np.array(grad_L_func(*x_n), dtype=float).flatten()
        H = np.array(jacobian_L_func(*x_n), dtype=float)
        if np.linalg.norm(grad, ord=2) < tolerance:
            print(f"Converged to: {x_n}")
            L_opt = lagrangian(*x_n)
            print(f"Optimal Lagrangian value: {L_opt}")
            return x_n
        delta_x = np.linalg.solve(H, -grad)
        x_n = x_n + delta_x
        print(f"Iteration {i + 1}: {x_n}")
    L_opt = lagrangian(*x_n)
    #print("Did not fully converge after the maximum number of iterations")
    print(f"Optimal Lagrangian value: {L_opt}")
    return x_n


lagrangian = sp.lambdify((x1, x2, lambda1), L, 'numpy')

# Initial guess
x0 = [0.5, 0.5, 0.5]


```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS D:\Melika\Uni\Term 6\Behinesazai Gheire Khati\erae>  & 'c:\Users\ASuS\anaconda3\python.exe
re Khati\erae\17.py'
Iteration 1: [ 1.75  1.5  -4.75]
Iteration 2: [ 0.85984848  3.0530303  -3.85984848]
Iteration 3: [ 0.64682852  2.62699037 -3.64682852]
Optimal Lagrangian value: 10.708356466367288
Result: [ 0.64682852  2.62699037 -3.64682852]
```

مقایسه با الگوریتم های دیگر

روش نیوتون رافسون

مزایا: سرعت همگرایی بالا: اگر نقاط اولیه خوب انتخاب شوند، روش نیوتن-رافسون دارای سرعت همگرایی بالایی است و می‌تواند در چند تکرار به جواب بهینه نزدیک شود. استفاده از اطلاعات مشتقات دوم: استفاده از ماتریس ژاکوبین (مشتقات دوم) به بهبود دقت و سرعت همگرایی کمک می‌کند.
معایب: نیاز به مشتقات دقیق: این روش به مشتقات دقیق (گرادیان و ماتریس ژاکوبین) نیاز دارد که محاسبه آن‌ها می‌تواند پیچیده و زمان‌بر باشد. نیاز به نقطه اولیه خوب: روش نیوتن-رافسون حساس به نقطه اولیه است و انتخاب نامناسب نقطه اولیه می‌تواند منجر به همگرایی به نقطه غیر بهینه یا عدم همگرایی شود.

روش $fsolve$

مزایا: ساده‌سازی فرآیند محاسباتی: با استفاده از$fsolve$ ،نیازی به محاسبه دستی مشتقات نیست و$fsolve$از روش‌های عددی برای تقریب مشتقات استفاده می‌کند. انعطاف پذیری بالا:$fsolve$ به طور خودکار مشتقات را محاسبه می‌کند و می‌تواند در مسائل پیچیده با مشتقات غیرخطی عملکرد خوبی داشته باشد.
معایب: سرعت همگرایی پایین‌تر: به دلیل استفاده از روش‌های عددی برای تقریب مشتقات، سرعت همگرایی ممکن است کمتر از روش نیوتن-رافسون باشد. حساسیت به نقاط اولیه: مشابه نیوتن-رافسون،$fsolve$ نیز به نقاط اولیه حساس است و انتخاب نقاط اولیه نامناسب می‌تواند منجر به نتایج نامناسب شود.

نتیجه گیری

انتخاب بهترین روش بستگی به ماهیت مسئله و نیازهای خاص شما دارد. اگر مشتقات را می‌توانید به دقت محاسبه کنید و نیاز به سرعت همگرایی بالا دارید، روش نیوتن-رافسون مناسب‌تر است. اما اگر محاسبه مشتقات پیچیده است و می‌خواهید از ابزارهای آماده استفاده کنید، روش $fsolve$ مناسب‌تر است.

لاگرانژ

روش لاگرانژ برای بهینه‌سازی مسئله‌هایی که دارای قیود هستند استفاده می‌شود. در این روش، از تابع لاگرانژ استفاده می‌شود که ترکیبی از تابع هدف و قیود است. سپس گرادیان این تابع را محاسبه کرده و

با حل معادلات گرادیان به دست آمده، نقطه بهینه را پیدا می‌کنیم.

نیوتون-رافسون

روش نیوتن-رافسون یک روش تکراری برای پیدا کردن ریشه‌های معادلات است که با استفاده از مشتقات مرتبه اول و دوم تابع، به سمت جواب همگرا می‌شود. در این مثال، ما گرادیان و ژاکوبین تابع لاگرانژ را محاسبه کرده و با استفاده از آن‌ها، هر بار مقدار جدیدی برای متغیرها به دست می‌آوریم تا زمانی که به همگرایی برسیم. این روش به صورت مستقیم به دنبال نقطه‌ای می‌گردد که گرادیان صفر شود. استفاده از $fsolve$ در اینجا نشان می‌دهد که از یک روش مبتنی بر نیوتن برای یافتن ریشه‌های معادلات استفاده شده است. روش نیوتن-رافسون به صورت تکراری و با استفاده از گرادیان و ژاکوبین به سمت جواب همگرا می‌شود.

سرعت همگرایی

روش لاگرانژ (با استفاده از $fsolve$): این روش معمولا سریع‌تر همگرا می‌شود و در صورتی که تابع به خوبی تعریف شده باشد، نتایج دقیقی ارائه می‌دهد.
روش نیوتن-رافسون: این روش نیز دقیق است، اما ممکن است در برخی موارد نیاز به تعداد بیشتری از تکرارها داشته باشد تا به همگرایی برسد.

پایداری

روش لاگرانژ: به دلیل استفاده از $fsolve$ که خود از یک روش نیوتن بهبود یافته استفاده می‌کند، معمولا پایدارتر است.
روش نیوتن-رافسون: ممکن است در برخی موارد ناپایدار باشد و به یک نقطه بهینه محلی همگرا شود.

سرعت

روش لاگرانژ: بسته به تابع و قیود ممکن است سریع‌تر همگرا شود.
روش نیوتن-رافسون: سرعت همگرایی بستگی به تابع و انتخاب نقاط اولیه دارد، اما معمولا نیاز به تعداد بیشتری از تکرارها دارد.

در این مثال خاص، هر دو روش می‌توانند جواب بهینه را پیدا کنند، اما روش استفاده شده در $fsolve$ (که از یک روش نیوتن-رافسون بهبود یافته استفاده می‌کند) به دلیل پایداری و دقت بیشتر، ممکن است ترجیح داده شود. در عین حال، روش نیوتن-رافسون خالص نیز دقت بالایی دارد اما نیاز به تنظیمات دقیق‌تر و توجه بیشتری به همگرایی دارد.

مراجع

[۱] $https://youtu.be/5A39Ht9Wcu0?si=NO5L865CzNRJb5dy$

[۲] $https://nasseralkmim.github.io/notes/lagrange-multiplier/$

[۳] $https://machinelearningmastery.com/lagrange-multiplier-approach-with-inequality-constraints/$

[۴] $https://en.wikipedia.org/wiki/Lagrange_multiplier$