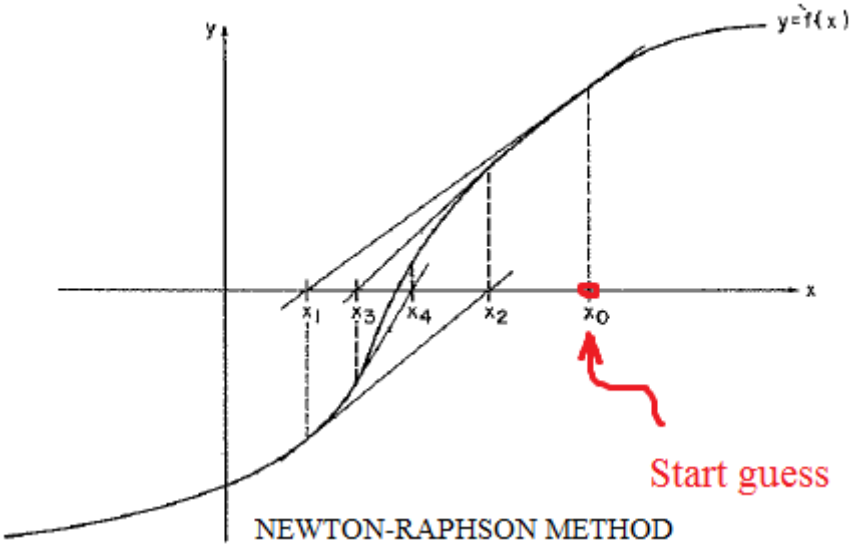
گزارش فارسی پیدا کردن ریشه های معادلات به کمک کدنویسی با متلب:

متد نیوتون-رافسون (Newton-Raphson Method)



پیدا کردن ریشه معادلات، پیدا کردن حل معادله $f(x)=0$ است در صورتی که تابع f داده شده باشد.

معرفی

یکی از عمومی ترین مسائل که در آنالیز مهندسی با آن مواجه میشویم بدین صورت است: یک تابع $f(x)$ داده شده است، مقداری از x را تعیین کنید که در آن $f(x)=0$. حل این معادلات (مقدار x) به عنوان ریشه های معادله $f(x) = 0$، یا صفر های تابع $f(x)$ شناخته میشوند.

قبل از اینکه در این قضیه جلوتر برویم، بهتر است در ابتدا مفهوم یک تابع را بیان کنیم. معادله:

$$y = f(x)$$

شامل سه المان می باشد: یک مقدار ورودی x، یک مقدار خروجی y و ضابطه f برای محاسبه y. اگر ضابطه f مشخص باشد میگوییم که تابع داده شده است. در محاسبات عددی ضابطه همیشه یک الگوریتم کامپیوتری است. ممکن است آن یک تابع به صورت زیر باشد:

$$f(x) = \cosh(x)\cos(x) - 1$$

یا یک روند پیچیده شامل هزاران یا صدها خط کد باشد. تا زمانی که الگوریتم ما برای هر مقدار مشخص x، یک خروجی برای y داشته باشد، به عنوان یک تابع شناخته میشود.

ریشه های معادلات ممکن است حقیقی یا مختلط باشند. ریشه های مختلط به ندرت محاسبه میشوند زیرا این ریشه ها از لحاظ اهمیت فیزیکی اهمیت کمی دارند. یک استثنا معادله چندجمله ایست.

$$a_1x^n + a_2x^{n-1} + \cdots + a_nx + a_{n+1} = 0$$

که در آنها ریشه های مختلط ممکن است دارای معنی و مفهوم باشند (به عنوان مثال در آنالیز دمپینگ ارتعاشات). در حال حاضر، ما فعلا روی پیدا کردن ریشه های حقیقی معادلات تمرکز خواهیم کرد.

عموما، ممکن است دارای تعداد زیادی ریشه حقیقی باشد و یا اصلا ریشه نداشته باشد. به عنوان مثال:

$$\sin x - x = 0$$

یک ریشه به صورت $x=0$ دارد، در حالی که:

$$\tan x - x = 0$$

تعداد بی نهایت ریشه دارد ($x=0,\ \pm4.493, \pm7.725, \ldots$).

همه متدهای پیدا کردن ریشه های معادلات جزو متدهای مبتنی بر تکرار هستند که نیاز به یک نقطه آغازگر دارند، یا به عبارتی به یک تخمین ریشه نیاز دارند. این تخمین میتواند خیلی حیاتی باشد. یک تخمین بد و دور از جواب ممکن است سبب شود که حل اصلا همگرا نگردد یا حتی ممکن است به جواب های نادرست و اشتباه همگرا گردد (ریشه بدست آمده با ریشه حقیقی متفاوت باشد). هیچ دستورالعمل جامع و عمومی برای تخمین مقدار ریشه های وجود ندارد. اگر مساله یا تابع ما مربوط به یک مساله فیزیکی باشد، محتوای مساله (دید فیزیکی) ممکن است بتواند مکان تقریبی ریشه ها را پیشنهاد کند. در غیر این صورت، تابع باید رسم شود، یا اینکه باید یک جستجوی عددی سیستماتیک برای پیدا کردن ریشه ها انجام گیرد.

این مساله شدیدا پیشنهاد میگردد که قبل از اینکه یک الگوریتم برای پیدا کردن ریشه ایجاد کنید و تابع را در این الگوریتم قرار دهید، یک گام جلوتر بروید و محدوده های بالایی و پایینی که ریشه در آن قرار دارد را تعیین کنید. در یک سری از متدهای ریشه یابی، پیدا کردن محدوده های بالاو پایین اجباری است.

متد نیوتون-رافسون

الگوریتم نیوتون-رافسون شناخته شده ترین متد برای پیدا کردن ریشه هاست آن هم به دلایل خوبی: این متد ساده و سریع است. تنها عیب این متد این است که این متد همانطور که از f(x) استفاده میکند، از مشتق آن نیز که $f'(x)$ است، استفاده میکند، بنابراین، متد نیوتون رافسون فقط در مسائلی قابل استفاده است که $f'(x)$ در آن به راحتی قابل محاسبه باشد.

فرمول نیوتون-رافسون میتواند از سری تیلور گسترش یافته f(x) حول x ، استخراج شود:

$$f(x_{i+1}) = f(x_i) + f'(x_i)(x_{i+1} - x_i) + O(x_{i+1} - x_i)^2 \qquad (a)$$

اگر x_{i+1} یک ریشه برای تابع f(x)=0 باشد، معادله a به صورت زیر در می آید:

$$0 = f(x_i) + f'(x_i)(x_{i+1} - x_i) + O(x_{i+1} - x_i)^2 \qquad (b)$$

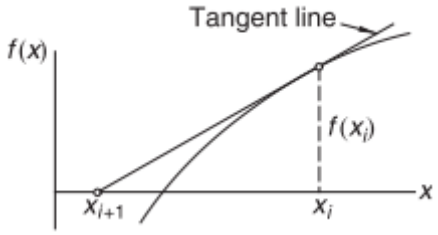
با فرض اینکه x_i نزدیک به x_{i+1} است، ما میتوانیم از ترم آخر معادله صرفنظر کنیم و معادله را برای x_{i+1} حل کنیم. نتیجه فرمولاسیون نیوتون-رافسون است.

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \qquad (۱)$$

اگر x جواب های درست ریشه را نشان دهد، خطا برای x_i، برابر با $E_i=x-x_i$ است. میتوان نشان داد که اگر x_{i+1} از معادله ۱ محاسبه شود، خطای متناظر به صورت زیر است:

$$E_{i+1} = -\frac{f''(x_i)}{2f'(x_i)}E_i^2$$

معادله بالا نشان میدهد که متد نیوتون-رافسون به صورت درجه دوم (خطا جزر خطای بدست آمده در گام قبلی است) همگرا میگردد. به عنوان یک نتیجه، تعداد ارقام قابل ملاحظه به صورت تقریبا در هر تکرار دو برابر میگردد. این نشان میدهد که x_i به ریشه نزدیک شده است.



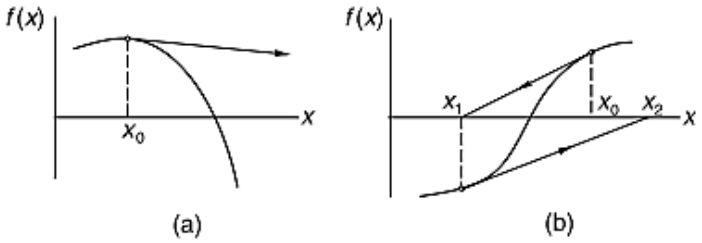
شکل۱: توصیف گرافیکی روش نیوتون رافسون

توصیف گرافیکی متد نیوتون-رافسون در شکل ۱ نشان داده شده است. این فرمولاسیون $f(x)$ را به وسیله یک خط مستقیم که در نقطه x_i بر منحنی مماس است، تقریب میزند. بنابراین x_{i+1} نقطه برخورد محور x و خط مماس است. الگوریتم روش نیوتون-رافسون ساده است: این الگوریتم معادله ۱ را به صورت تکراری اعمال میکند. بدین صورت که از یک مقدار اولیه x_0 آغاز میکند، تا زمانی که شرط همگرایی زیر بدست آید:

$$|x_{i+1} - x_1| < \varepsilon$$

ε تلورانس خطا است. در اینجا فقط آخرین مقدار بدست آمده برای x باید ذخیره شود. الگوریتم به صورت زیر است:

۱. بگذارید که x یک حدس برای ریشه $f(x)=0$ باشد.
۲. $\Delta x = -f(x)/f'(x)$ را محاسبه کنید.
۳. $x \to x + \Delta x$ و گام های ۲و۳ را تا زمانی که $|\Delta x| < \varepsilon$.



شکل۲: حالت هایی که متد نیوتون-رافسون در آنها واگرا میشود.

هرچند که متد نیوتون رافسون به سرعت در نزدیکی ریشه های همگرا میگردد، مشخصه های همگرایی کلی آن ضعیف هستند. دلیل این مساله آن است که خط مماسی همیشه یک تقریب قابل قبول از تابع نیست، همانطور که در دو مثال شکل ۲ نشان داده شده است. اما متد به وسیله ترکیب کردن با روش نصف کردن تقریبا از واگرایی نجات پیدا میکند.

- **متد نیوتون-رافسون**

ورژن ایمن (safe version) از متد نیوتون رافسون (نسخه بهبود یافته و ایمن از لحاظ همگرایی) فرض میکند که ریشه هایی که باید محاسبه شود در ابتدا در بازه (a,b) قرار دارد. نقطه میانی فاصله به عنوان حدس اولیه ریشه مورد استفاده قرار میگیرد. بازه ریشه پس از هر تکرار آپدیت میشود. اگر یک تکرار نیوتون رافسون در بازه مورد نظر باقی نماند، این روش نادیده گرفته خواهد شد و با روش نصف کردن جایگزین خواهد شد. چون newtonRaphson از تابع f(x) و مشتق آن استفاده میکند، کدهای تابع برای هر دوتای f(x) و مشتق آن (که به صورت func و dfunc نشان داده شده است) باید توسط کاربر فراهم گردد.

```
function root = newtonRaphson(func,dfunc,a,b,tol)
% Newton-Raphson method combined with bisection for
% finding a root of f(x) = 0.
% USAGE: root = newtonRaphson(func,dfunc,a,b,tol)
% INPUT:
% func = handle of function that returns f(x).
% dfunc = handle of function that returns f'(x).
% a,b = brackets (limits) of the root.
% tol = error tolerance (default is 1.0e6*eps).
% OUTPUT:
% root = zero of f(x) (root = NaN if no convergence).
if nargin < 5; tol = 1.0e6*eps; end
fa = feval(func,a); fb = feval(func,b);
if fa == 0; root = a; return; end
if fb == 0; root = b; return; end
if fa*fb > 0.0
        error('Root is not bracketed in (a,b)')
end
x = (a + b)/2.0;
for i = 1:30
```