4.1 Теңдеулердің түбірлері

Ньютон Рафсон әдісі

Ньютон-Рафсон алгоритмі түбірлерді табудың ең танымал әдісі болып табылады және де әдіс қарапайым және жылдам. Әдістің бірден-бір кемшілігі - ол функцияның $f'(x)$ туындысын, сондай-ақ $f(x)$ функциясының өзін қолданады. Сондықтан Ньютон-Рафсон әдісі $f'(x)$ оңай есептелетін есептерде ғана қолданылады.

Ньютон-Рафсон формуласы $f(x)$-тің x-ке қатысты Тейлор қатарына жіктелуінен алынуы мүмкін:

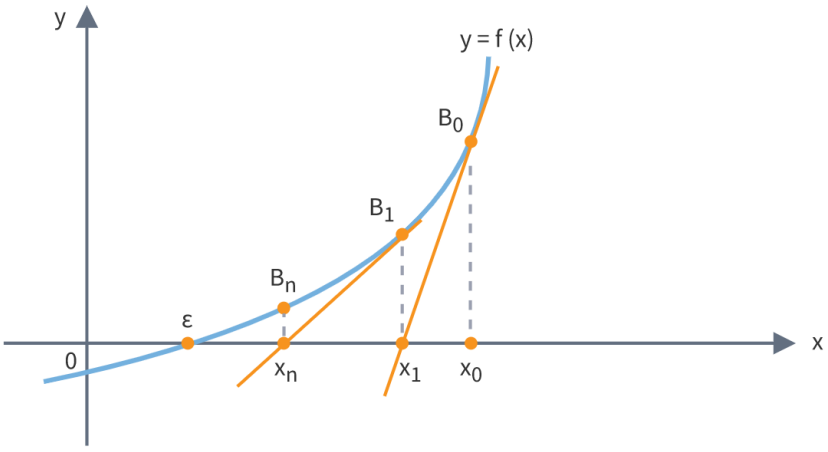
$$f(x_{i+1}) = f(x_i) + f'(x_i)(x_{i+1} - x_i) + O(x_{i+1} - x_i)^2 \quad (a)$$

мұндағы $O(z)$ «z ретімен» деп оқылады. Егерx_{i+1}мәні $f(x) = 0$ түбірі болса, онда (a) теңдеуі

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

осы түрде болады. x_i мәні x_{i+1} ге жақын деп есептесек, (b) теңдеудің соңғы мүшесін алып тастай аламыз және x_{i+1} қатысты шешеміз. Нәтижесінде Ньютон-Рафсон формуласы шығады:

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



1-сурет. Ньютон-Рафсон формуласының графикалық интерпретациясы.

Ньютон-Рафсон формуласының графикалық бейнесі 1-суретте көрсетілген. Формула $f(x)$ мәніне x_i нүктесіндегі қисыққа жанама болатын түзу арқылы жуықтады. Осылайша x_{i+1} нүктесі Ox осі мен жанама түзудің қиылысында болады. Ньютон-Рафсон әдісінің алгоритмі қарапайым: ол (1) теңдеуді қайталап қолданады. x_0 бастапқы мәнінен бастап, жинақтылық критерийіне дейін

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

орындалады, ε мүмкін болатын қателік мәні болып табылады. Есептеу барысында тек соңғы x мәнін сақтау керек. Алгоритмі:

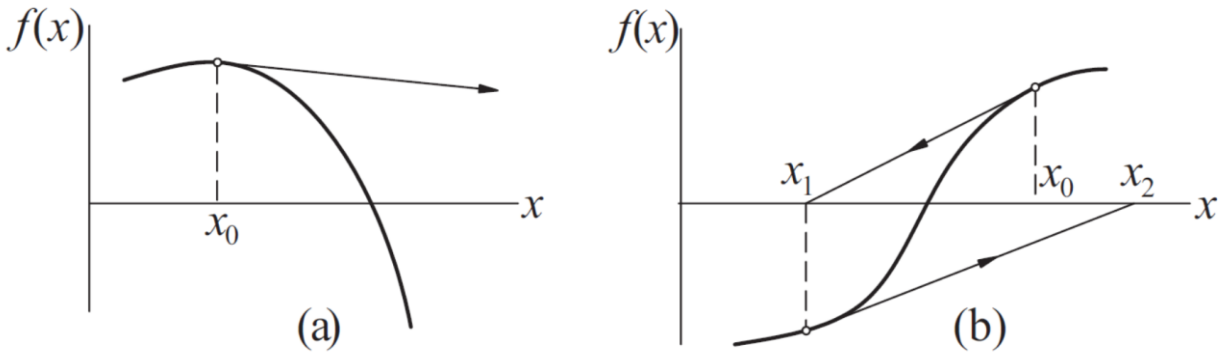
- x мәні $f(x) = 0$ түбірінің шамасы болсын.
- $|\Delta x| < \varepsilon$ дейін орындаңыз:
- $\Delta x = f(x)/f'(x)$ есептеңіз.
- $x \leftarrow x + \Delta x$ меншіктейміз.

Ньютон-Рафсон формуласындағы E кесу қатесін

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

ретінде көрсетуге болады, мұнда x – түбір. Бұл әдістің квадраттық жинақталатынын көрсетеді. Демек, әрбір итерацияда мәнді сандар саны шамамен екі есе артады.

Ньютон-Рафсон әдісі түбірге жақын жерде тез жинақталғанымен, оның жаһандық жинақталу сипаттамалары нашар. Себебі 2-суреттегі екі мысалда көрсетілгендей жанама сызық әрқашан функцияның қолайлы жуықтауы бола бермейді. Дегенмен, әдісті екіге бөлу әдісімен біріктіру арқылы әрқашанда жинақталататындай етуге болады.



2-сурет. Ньютон-Рафсон әдісінің жинақталмайтын мысалдары.

newtonRaphson модулі

Ньютон-Рафсон әдісінің келесі қауіпсіз нұсқасы есептелетін түбір бастапқыда (a, b) жақшаға алынады деп болжайды. Түбірдің бастапқы болжамы ретінде жақшаның ортаңғы нүктесі пайдаланылады. Жақшалар әрбір итерациядан кейін жаңартылады. Егер Ньютон-Рафсон итерациясы жақшаның ішінде қалмаса, ол ескерілмейді және екіге бөлу әдісімен ауыстырылады. newtonRaphson $f(x)$ функциясын, сондай-ақ оның туындысын пайдаланатындықтан, екеуін де (листингте f және df арқылы белгіленген) пайдаланушы енгізуі керек.

module newtonRaphson

```
''' түбір = newtonRaphson(f,df,a,b,tol=1,0e-9).
    Ньютон-Рафсонды екіге бөлу әдісімен
    біріктіру арқылы f(x) = 0 түбірін табады.
    Түбір (a,b) аралықта болуы керек
    Пайдаланушы ұсынған f(x) функциясын
    және оның df(x) туындысын шақырады.
'''
def newtonRaphson(f,df,a,b,tol=1.0e-9):
    import error
    from numpy import sign
    fa = f(a)
    if fa == 0.0: return a
    fb = f(b)
    if fb == 0.0: return b
    if sign(fa) == sign(fb): error.err('Түбір аралықта емес')
    x = 0.5*(a + b)
    for i in range(30):
        fx = f(x)
        if fx == 0.0: return x
      # Түбір жақшасын тексеру
        if sign(fa) != sign(fx): b = x
        else: a = x
        # Ньютон-Рафсон қадамын қолдану
        dfx = df(x)
        # Егер нөлге бөлінсе, x ті шегінен шығарыңыз
        try: dx = -fx/dfx
        except ZeroDivisionError: dx = b - a
        x = x + dx
        # Егер нәтиже жақшадан тыс болса, екіге бөлуді пайдаланыңыз
        if (b - x)*(x - a) < 0.0:
            dx = 0.5*(b - a)
            x = a + dx
        # Жинақтылыққа тексеріңіз
        if abs(dx) < tol*max(abs(b),1.0): return x
    print('Ньютон-Рафсондағы итерациялар тым көп')
```

МЫСАЛ 1

Екі бүтін санның қатынасы ретінде $\sqrt{2}$ тізбектес жуықтауларын алу үшін Ньютон-Рафсон әдісін қолданыңыз.

Шешуі. Есеп $f(x) = x^2 - 2 = 0$ түбірін табумен бірдей. Мұнда Ньютон-Рафсон формуласы

$$x \leftarrow x - \frac{f(x)}{f'(x)} = x - \frac{x^2 - 2}{2x} = \frac{x^2 + 2}{2x}$$

$x = 1$-ден бастап, тізбектес қайталаулары

$$x \leftarrow \frac{(1)^2 + 2}{2 \cdot 1} = \frac{3}{2}$$

$$x \leftarrow \frac{(3/2)^2 + 2}{2 \cdot (3/2)} = \frac{17}{12}$$

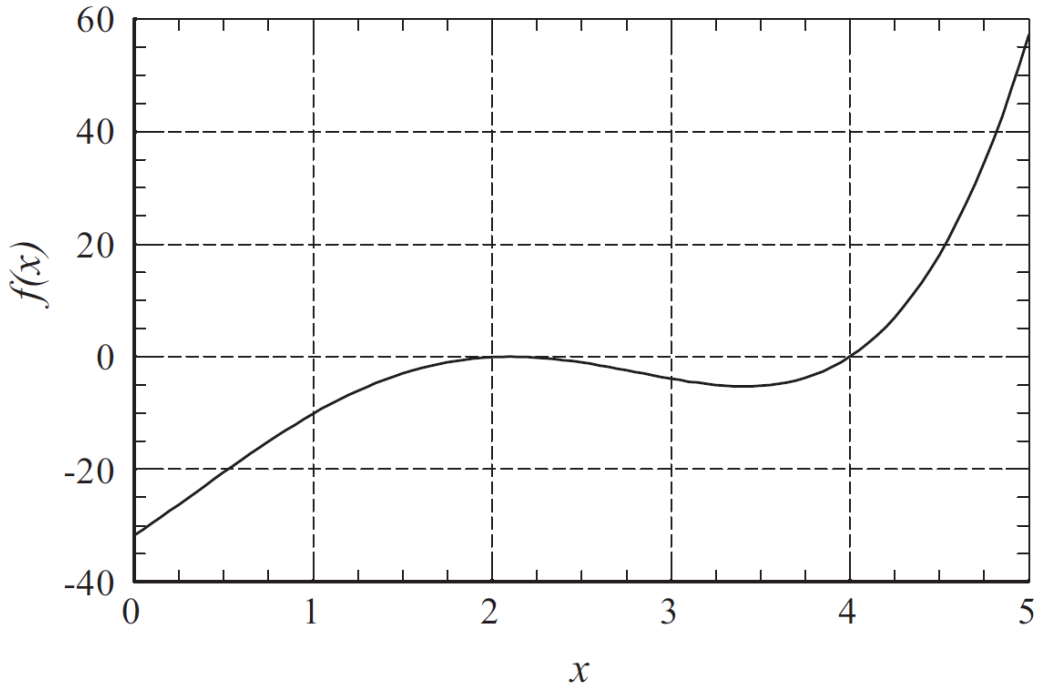
$$x \leftarrow \frac{(17/12)^2 + 2}{2 \cdot (17/12)} = \frac{577}{408}$$

Назар аударыңыз, $x = 577/408 = 1.1414216$ мәні $\sqrt{2} = 1.1414214$ мәніне өте жақын. Нәтижелер x-тің бастапқы мәніне байланысты. Мысалы, x=2 қатынастың басқа тізбегін береді.

МЫСАЛ 2

$$f(x) = x^4 - 6.4x^3 + 6.45x^2 + 20.538x - 31.752$$

функциясының ең кіші оң нөлін табыңыз.



Шешуі. Функцияның графигін тексере отырып, біз ең кіші оң нөл шамамен x = 2 кезінде қос түбір болатынын байқаймыз. Екіге бөлу және Риддер әдістері бұл

жерде жұмыс істемейді, өйткені олар функцияның түбірдегі таңбасын өзгертуіне байланысты. Бірдей аргументте newtonRaphson функциясына қолданылады. Дегенмен Ньютон-Рафсон әдісінің тұрпайы нұсқасының сәтті болмауына ешқандай себеп жоқ. Біз түбір мен бірге итерация санын басып шығаратын келесі бағдарламаны қолдандық:

```
#!/usr/bin/python
## example2
def f(x): return x**4 - 6.4*x**3 + 6.45*x**2 + 20.538*x - 31.752
def df(x): return 4.0*x**3 - 19.2*x**2 + 12.9*x + 20.538
def newtonRaphson(x,tol=1.0e-9):
    for i in range(30):
        dx = -f(x)/df(x)
        x = x + dx
        if abs(dx) < tol: return x,i
root,numIter = newtonRaphson(2.0)
print 'Түбір =',root
print 'Итерация саны =',numIter
```

Нәтижесі

```
Root = 2.0999999786199406
Number of iterations = 22
```

Түбірдің шын мәні $x = 2.1$. Еселі түбірдің жанында Ньютон-Рафсон әдісінің жинақтылығы квадраттық емес, сызықтық болатынын көрсетуге болады, бұл итерация санының көптігін түсіндіреді. Еселі түбірге жинақтылықты Ньютон-Рафсон формуласын (1) теңдеудегі мына ауыстыру арқылы тездетуге болады

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

болғанда, мұнда m – түбірдің еселігі (бұл есепте m=2). Бағдарламаға өзгеріс енгізгеннен кейін біз нәтижені тек бес итерацияда алдық.

Теңдеулер жүйесі

Кіріспе

Осы уақытқа дейін біз жалғыз $f(x) = 0$ теңдеуін шешуге назар аудардық. Енді сол есептің n өлшемді нұсқасын қарастырайық, атап айтқанда,

$$\boldsymbol{f}(\boldsymbol{x}) = \mathbf{0}$$

немесе скаляр белгісін қолданып

$$f_1(x_1, x_2, \ldots, x_n) = 0$$
$$f_2(x_1, x_2, \ldots, x_n) = 0$$
$$\vdots$$
$$f_n(x_1, x_2, \ldots, x_n) = 0$$

n, сызықты емес теңдеулер жүйесін шешу бір теңдеудің түбірін табудан әлдеқайда күрделі есеп. Мәселе мынада, шешім векторы x аралықта жатудың сенімді әдісі жоқ. Сондықтан, егер мұндай мәндер есептің физикасымен ұсынылмаса, біз әрқашан шешім алгоритмін x-тің жақсы бастапқы мәнімен қамтамасыз ете алмаймыз.

x-ті есептеудің ең қарапайым және тиімді құралы Ньютон-Рафсон әдісі болып табылады. Ол жақсы бастапқы нүктемен қамтамасыз етілсе, бір уақыттағы теңдеулермен жақсы жұмыс істейді. Жаһандық жинақталу сипаттамалары жақсырақ басқа әдістер бар, бірақ олардың барлығы Ньютон-Рафсон әдісінің түрлері болып табылады.

Ньютон-Рафсон әдісі

Теңдеулер жүйесі үшін Ньютон-Рафсон әдісін шығару үшін, x нүктесіне қатысты $f_i(x) = 0$ дің Тейлор қатарына жіктеуден бастаймыз:

$$f_j(x + \Delta x) = f_j(x) + \sum_{j=1}^{n} \frac{\partial f_i}{\partial x_j} \Delta x_j + O(\Delta x^2) \qquad (2a)$$

Δx^2 аз мүшесін түсіріп, (2a) теңдеуін мына түрде жаза аламыз.

$$f(x + \Delta x) = f(x) + J(x)\Delta x \qquad (2b)$$

мұнда $J(x)$ – дербес туындылардан құралған якобиандық матрица ($n \times n$ өлшемді)

$$J_{ij} = \frac{\partial f_i}{\partial x_j} \qquad (3)$$

(2b) теңдеуі x нүктесінің маңайында векторлық мәнді f функциясы сызықтық жуықтауы (x векторы айнымалы) болып табылады.

Енді x мәні $f(x) = 0$ шешімінің ағымдағы жуықтауы деп алайық, ал $x + \Delta x$ жақсартылған шешім болсын. Δx түзетуін табу үшін (2b)теңдеуге $f(x + \Delta x) = 0$ мәнін қоямыз. Нәтиже Δx үшін сызықтық теңдеулер жинағы:

$$J(x)\Delta x = -f(x) \qquad (4)$$

Әрбір $\partial f_i/\partial x_j$ мәнін аналитикалық шығару қиын немесе мүмкін емес болғандықтан, компьютерде олардың ақырлы айырымдық жуықтауынан есептегені дұрыс.

$$\frac{\partial f_i}{\partial x_j} \approx \frac{f_i\left(x + e_j h\right) - f_i(x)}{h} \qquad (5)$$

мұндағы $h - x_j$дің кіші қадамы және $e_j - x_j$ бағыты бойынша бірлік векторды білдіреді. Бұл формуланы (2a) теңдеуден Δx^2 шамасын алып тастағаннан кейін және $\Delta x = e_j h$ орнатқаннан кейін алуға болады. Біз (5) теңдеудегі жуықтауды пайдаланамыз, өйткені Ньютон-Рафсон әдісі $J(x)$ қателеріне өте сезімтал емес. Бұл жуықтауды қолдану арқылы біз компьютерлік кодқа $\partial f_i/\partial x_j$ өрнектерін теруден де аулақ боламыз.

Келесі қадамдар, сызықты емес теңдеулер жүйесі үшін Ньютон-Рафсон әдісін құрайды:

Шешім x векторын бағалаңыз.

$|\Delta x| < \varepsilon$ дейін орындаңыз:

(5) теңдеуден $J(x)$ матрицасын есептеңіз.

Δx үшін $J(x)\Delta x = -f(x)$ шешімін табыңыз.

$x \leftarrow x + \Delta x$ болсын.

мұндағы ε – мүмкін қателік. Бір өлшемді жағдайдағыдай, Ньютон-Рафсон үрдісінің сәттілігі толығымен x-тің бастапқы жуықтауына байланысты. Егер жақсы бастапқы нүкте қолданылса, шешімге жақындау өте жылдам болады. Әйтпесе, нәтижелерді болжау мүмкін емес.

newtonRaphson2 модулі

Бұл модуль Ньютон-Рафсон әдісін жүзеге асыру болып табылады. Ішкі jacobian функциясы якобиандық матрицаны (5) теңдеудегі ақырлы айырымдық жуықтауынан есептейді. (4) теңдеулер жүйесі Гаусстың біртіндеп жою арқылы шешіледі. $f(x)$ массивін қайтаратын f функциясының ішкі бағдарламасы пайдаланушы енгізу керек.

```
## module newtonRaphson2
```

```
''' soln = newtonRaphson2(f,x,tol=1.0e-9).
f(x)=0 теңдеулер жүйесін шешеді
'''
import numpy as np
from gaussPivot import *
import math
def newtonRaphson2(f,x,tol=1.0e-9):
    def jacobian(f,x):
        h = 1.0e-4
        n = len(x)
        jac = np.zeros((n,n))
        f0 = f(x)
        for i in range(n):
            temp = x[i]
            x[i] = temp + h
            f1 = f(x)
            x[i] = temp
            jac[:,i] = (f1 - f0)/h
        return jac,f0
    for i in range(30):
        jac,f0 = jacobian(f,x)
        if math.sqrt(np.dot(f0,f0)/len(x)) < tol: return x
        dx = gaussPivot(jac,-f0)
        x = x + dx
        if math.sqrt(np.dot(dx,dx)) < tol*max(max(abs(x)),1.0):
            return x
```

Якобиандық J(x) матрицасы әрбір итерациялық циклде қайта есептелетінін ескеріңіз. J(x) әрбір есебі $f(x)\ n+1$ есептеуін қамтитындықтан (n - теңдеулер саны), есептеу шығыны n өлшеміне және f(x) күрделілігіне байланысты жоғары болуы мүмкін. Итерациялар арасындағы якобиандық матрицаның өзгерістерін ескермеу арқылы компьютер уақытын үнемдеуге жиі болады, осылайша J(x) тек бір рет есептеледі. Бұл тәсіл бастапқы мән x шешімге жеткілікті жақын болған жағдайда жұмыс істейді.

Қолданылған әдебиеттер тізімі

1) Шакенов Қ.Қ. Есептеу математикасы әдістері лекциялар курсы.Алматы,2019. – 193б
2) P. Dechaumphai, N. Wansophark. Numerical Methods in Science and Engineering Theories with MATLAB, Mathematica, Fortran, C and Python Programs. Alpha Science International Ltd. 2022
3) Н. С. Бахвалов, Н. П. Жидков, Г. М. Кобельков Численные методы: Классический университетский учебник. —М.: Издательство «Бином. Лаб. знаний»,2020. — 636 с.
4) Вабищевич П.Н. Численные методы: Вычислительный практикум. — М.: Книжный дом «ЛИБРОКОМ», 2020. — 320 с.

Интернет ресурстар

1) https://docs.python.org/3/
2) http://math-hse.info/f/2018-19/py-polit/instruction_JN.pdf
3) https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/execute.html
4) https://colab.research.google.com/
5) https://planetcalc.ru/search/?tag=2874