Итерациялық әдістер

Кіріспе

Осы уақытқа дейін біз шешудің тура әдістерін ғана талқыладық. Бұл әдістердің ортақ сипаттамасы – олар шешімді операциялардың шектеулі санымен есептейді. Оның үстіне, егер компьютер шексіз дәлдікке қабілетті болса (дөңгелектеу қателері жоқ), шешім дәл болар еді.

Итерациялық немесе жанама әдістер x шешімін бастапқы болжаудан бастайды, содан кейін x өзгерісі елеусіз болғанша шешімді қайта-қайта жақсартады. Итерациялардың қажетті саны үлкен болуы мүмкін болғандықтан, жанама әдістер, жалпы алғанда, тура әдіске қарағанда баяу жұмыс жасайды. Дегенмен, итерациялық әдістер оларды белгілі бір есептер үшін тартымды ететін келесі екі артықшылыққа ие:

1. Коэффициенттік матрицаның нөлден басқа элементтерін ғана сақтау мүмкін. Бұл өте үлкен матрицалармен жұмыс істеуге мүмкіндік береді. Көптеген есептерде коэффициент матрицасын мүлде сақтаудың қажеті жоқ.

2. Итерациялық процедуралар өздігінен түзетіледі, яғни бір итерациялық циклдегі дөңгелектеу қателері (тіпті арифметикалық қателер) келесі циклдарда түзетіледі.

Итерациялық әдістердің елеулі кемшілігі – олар әрқашан шешімге жақындай бермейді. Коэффицент матрицасы диагональ бойынша басым болған жағдайда ғана жинақталуға кепілдік берілетінін көрсетуге болады. x үшін бастапқы жуықтау жинақталудың орын алуын анықтауда ешқандай рөл атқармайды — егер процедура бір бастапқы вектор үшін жинақталса, ол кез келген бастапқы вектор үшін солай жасайды. Бастапқы болжам жинақталуға қажетті итерациялар санына ғана әсер етеді.

Гаусс-Зейдель әдісі

$A \cdot x = b$ теңдеулер жүйесі скалярлық белгілеуде

$$\sum_{j=1}^{n} A_{ij} x_j = b_i, \qquad i = 1,2, \dots, n$$

Қосынды белгісінен x_i бар мүшені алу келесі нәтиже береді

$$A_{ii} x_i + \sum_{\substack{j=1 \\ j \neq i}}^{n} A_{ij} x_j = b_i, \qquad i = 1,2, \dots, n$$

x_i үшін шешеміз

$$x_i = \frac{1}{A_{ii}} \left(b_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} A_{ij} x_j \right), \qquad i = 1,2, \dots, n$$

Соңғы теңдеу келесі итерациялық сұлбаны ұсынады:

$$x_i \leftarrow \frac{1}{A_{ii}} \left(b_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} A_{ij} x_j \right), \qquad i = 1,2, \dots, n \tag{1}$$

Біз *x* бастапқы векторын таңдаудан бастаймыз. Егер шешімнің жақсы болжамы болмаса, *x* кездейсоқ таңдалуы мүмкін. Содан кейін (1) теңдеу *x*-тің әрбір элементін қайта есептеу үшін пайдаланылады, әрқашан x_j -ң ең соңғы қол жетімді мәндерін пайдаланады. Бұл бір итерация циклін аяқтайды. Процедура тізбектес итерация циклдері арасындағы *x* өзгерістері жеткілікті түрде аз болғанша қайталанады.

Гаусс-Зайдель әдісінің жинақталуын релаксациямен жақсартуға болады. Негізгі идея x_i-ң жаңа мәнін оның алдыңғы мәні мен (1) теңдеу бойынша болжанған мәнінің орташа өлшенген мәні ретінде қабылдау болып табылады. Сәйкес итерациялық формула келесідей болып табылады

$$x_i \leftarrow \frac{\omega}{A_{ii}}\left(b_i - \sum_{\substack{j=1 \\ j\neq i}}^{n} A_{ij}x_j\right) + (1-\omega)x_i, \qquad i = 1,2,\ldots,n \tag{2}$$

мұндағы ω салмақ релаксация коэффициенті деп аталады. Егер $\omega = 1$ болса, релаксация болмайтынын көруге болады, өйткені (1) мен (2) теңдеулер бірдей нәтиже береді. Егер $\omega < 1$ болса, (2) теңдеу ескі x_i мен (1) теңдеуде берілген мән арасындағы интерполяцияны көрсетеді. Бұл төменгі релаксация деп аталады. $\omega > 1$ болған жағдайда бізде экстраполяция немесе жоғары релаксация болады.

ω тиімді мәнін алдын ала анықтаудың практикалық әдісі жоқ; дегенмен, бағалауды орындау уақытында есептеуге болады. $\Delta x^{(k)} = \left|x^{(k-1)} - x^{(k)}\right|$ k-итерация кезіндегі x өзгерісінің шамасы болсын (релаксациясыз кезінде [яғни, ω = 1]). Егер *k* жеткілікті үлкен болса (айталық, $k \geq 5$), ω тиімді мәнінің жуықтауы болатынын көрсетуге болады

$$\omega_{opt} \approx \frac{2}{1 + \sqrt{1 - (\Delta x^{(k+p)}/\Delta x^{(k)})^{1/p}}} \tag{3}$$

мұндағы p – натурал сан.

Релаксациясы бар Гаусс-Зайдель алгоритмінің негізгі элементтері келесідей:

$\omega = 1$ болатын *k* итерацияны орындаймыз ($k = 10$ орынды).
$\Delta x^{(k)}$ жазып аламыз.
p итерация үшін қайта орындаймыз.
$\Delta x^{(k+p)}$ жазып аламыз.
(3) теңдеуінен ω_{opt} есептеңіз.
Барлық келесі итерацияларды $\omega = \omega_{opt}$ арқылы орындаңыз.

gaussSeidel

`gaussSeidel` функциясы Гаусс-Зайдель әдісінің релаксациямен жүзеге асырылуы болып табылады. Ол $k = 10$ және $p = 1$ көмегімен автоматты түрде (3) теңдеуден ω_{opt} есептейді. Пайдаланушы (1) теңдеудегі қайталанатын формулалардан жақсартылған *x* есептейтін `iterEqs` функциясын қамтамасыз етуі керек. `gaussSeidel` функциясы *x* шешім векторын, орындалған итерациялар санын және пайдаланылған ω_{opt} мәнін қайтарады.

```
## module gaussSeidel
"""
x,numIter,omega = gaussSeidel(iterEqs,x,tol = 1.0e-9)
```

```
[A]{x} = {b} шешуге арналған Гаусс-Зайдель әдісі.
[A] матрицасы сиретілген болуы керек. Пайдаланушы ағымдағы {x}
мәнін ескере отырып, жақсартылған {x} мәнін қайтаратын
iterEqs(x,omega)функциясын қамтамасыз етуі керек
(«omega» — релаксация коэффициенті).
"""
import numpy as np
import math
def gaussSeidel(iterEqs,x,tol = 1.0e-9):
    omega = 1.0
    k = 10
    p = 1
    for i in range(1,501):
        xOld = x.copy()
        x = iterEqs(x,omega)
        dx = math.sqrt(np.dot(x-xOld,x-xOld))
        if dx < tol: return x,i,omega
    #  k+p итерациясынан кейін релаксация коэффициентін есептеңіз
    if i == k: dx1 = dx
    if i == k + p:
        dx2 = dx
        omega = 2.0/(1.0 + math.sqrt(1.0 \
              - (dx2/dx1)**(1.0/p)))
    print('Гаусс-Зайдель жинақталмады')
```

Түйіндес градиент әдісі

$$f(x) = \frac{1}{2}x^T Ax - b^T x \tag{4}$$

скаляр функциясын минимизациялайтын x векторын табу есебін қарастырайық, мұндағы A матрицасы симметриялы және оң анықталған. Оның градиенті $\nabla f = Ax - b$ нөлге тең болғанда $f(x)$ минимизацияланатындықтан, минимизациялау есебі

$$Ax = b \tag{5}$$

шешіміне эквивалент екенін көреміз.

Градиенттік әдістер бастапқы вектор x_0 бастап итерация арқылы минимизациялауды жүзеге асырады. Әрбір қайталанатын цикл k нақтыланған шешімді есептейді

$$x_{k+1} = x_k + \alpha_k s_k \tag{6}$$

α_k қадам ұзындығы s_k іздеу бағытында x_{k+1} мәні $f(x_{k+1})$ минимизациялайтын етіп таңдалады. Яғни, x_{k+1} (5) теңдеуді қанағаттандыруы керек:

$$A(x_k + \alpha_k s_k) = b$$

Теңдеуінің қалдығын енгізіп,

$$r_k = b - Ax_k \tag{7}$$

$\alpha_k A s_k = r_k$ аламыз. Екі жағын s_k^T-ға алдын ала көбейтіп, α_k үшін шешсек, біз келесіні аламыз

$$\alpha_k = \frac{s_k^T r_k}{s_k^T A s_k} \tag{8}$$

Бізде әлі де s_k іздеу бағытын анықтау мәселесі қалды. Интуиция бізге $s_k = -\nabla f = r_k$ таңдау керектігін айтады, өйткені бұл $f(x)$ ішіндегі ең үлкен теріс өзгерістің бағыты. Осы процедура ең жылдам түсу әдісі ретінде белгілі. Бұл танымал алгоритм емес, өйткені оның жинақталуы баяу болуы мүмкін. Неғұрлым тиімді түйіндес градиент әдісі іздеу бағытын келесідей пайдаланады

$$s_{k+1} = r_{k+1} + \beta_k s_k \tag{9}$$

β_k тұрақтысы екі тізбектес іздеу бағыты бір-бірімен түйіндес етіп таңдалады, яғни

$$s_{k+1}^T A s_k = 0$$

Түйіндес градиенттердің үлкен тартымдылығы мынада: бір түйіндес бағытта минимизациялау бұрынғы минимизацияларды жоймайды (минимизациялар бір-біріне кедергі жасамайды).

Соңғы теңдеудегі s_{k+1}орнына (9) қойылады. Осыдан аламыз

$$(r_k^T + \beta_k s_k^T) A s_k = 0$$

ол береді

$$\beta_k = -\frac{r_{k+1}^T A s_k}{s_k^T A s_k}$$

Мұнда түйіндес градиент алгоритмінің құрылымы берілген:

x_0 векторын таңдаңыз(кез келген вектор).
$r_0 = b - Ax_0$ болсын.
$s_0 \leftarrow r_0$ болсын (алдыңғы іздеу бағыты жоқ, ең тік түсу бағытын таңдаңыз).
$k = 0, 1, 2, \ldots$ көмегімен орындаңыз:

$\alpha_k \leftarrow \frac{s_k^T r_k}{s_k^T A s_k}$
$x_{k+1} \leftarrow x_k + \alpha_k s_k$
$r_{k+1} \leftarrow b - Ax_{k+1}$
егер$|r_{k+1}| < \varepsilon$ шығу циклі (ε – қате ауытқуы).
$\beta_k \leftarrow -\frac{r_{k+1}^T A s_k}{s_k^T A s_k}$
$s_{k+1} \leftarrow r_{k+1} + \beta_k s_k$

Алгоритм бойынша алынған $r_1, r_2, r_3, \ldots$ қалдық векторлары өзара ортогональ болатынын көрсетуге болады, яғни $r_i \cdot r_j = 0, \ i \neq j$. Енді n қалдық векторлардың барлық жиынын есептеу үшін жеткілікті итерацияларды орындадық делік. Келесі итерация нәтижесінде пайда болатын қалдық шешімнің алынғанын көрсететін нөлдік вектор ($r_{n+1} =$

0) болуы керек. Осылайша, түйіндес градиент алгоритмі итерациялық әдіс емес сияқты көрінеді, өйткені ол n есептеу циклінен кейін нақты шешімге жетеді. Алайда іс жүзінде түйіндес әдіс n-нен аз итерацияда қол жеткізіледі.

Түйіндес градиент әдісі кішігірім теңдеулер жиынын шешуде тура әдістермен бәсекеге қабілетті емес. Оның күші үлкен, сирек жүйелермен жұмыс істеуде (A элементтерінің көпшілігі нөлге тең) көрінеді. Айта кету керек, A алгоритмге тек оны векторға көбейту арқылы кіреді; яғни Av түрінде, мұндағы v – вектор (x_{k+1}немесе s_k). Егер A сирек болса, көбейту үшін тиімді ішкі бағдарламаны жазуға болады және A-ның өзіне емес, оның көбейтіндісін түйіндес градиент алгоритміне беруге болады.

conjGrad

Келесі көрсетілген `conjGrad` функциясы түйіндес градиент алгоритмін жүзеге асырады. Итерациялардың максималды рұқсат етілген саны n (белгісіздер саны) болып тұр. Назар аударыңыз, `conjGrad` Av функциясын шақыратынын ескеріңіз. Бұл функцияны пайдаланушы қамтамасыз етуі керек. Пайдаланушы сонымен қатар x_0 бастапқы векторын және тұрақты (оң жақ) b векторын беруі керек. Функция x шешім векторын және итерациялар санын қайтарады.

```
##  conjGrad модулі
"""
x, numIter = conjGrad(Av,x,b,tol=1,0e-9)
[A]{x} = {b} шешуге арналған түйіндес градиент әдісі.
[A] матрицасы сирек болуы керек. Пайдаланушы жеткізуі керек
[A]{v} векторын қайтаратын Av(v) функциясын.
"""
import numpy as np
import math
def conjGrad(Av,x,b,tol=1.0e-9):
    n = len(b)
    r = b - Av(x)
    s = r.copy()
    for i in range(n):
        u = Av(s)
        alpha = np.dot(s,r)/np.dot(s,u)
        x = x + alpha*s
        r = b - Av(x)
        if(math.sqrt(np.dot(r,r))) < tol:
            break
        else:
            beta = -np.dot(r,u)/np.dot(s,u)
            s = r + beta*s
    return x,i
```

Мысал-1

$$\begin{bmatrix} 2 & -1 & 0 & 0 & \dots & 0 & 0 & 0 & 1 \\ -1 & 2 & -1 & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & \dots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & -1 & 2 & -1 \\ 1 & 0 & 0 & 0 & \dots & 0 & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-2} \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Берілген n сызықты алгебралық теңдеулер жүйесін Гаусс-Зайдельдің релаксация әдісімен шешуге арналған компьютерлік бағдарламаны жазыңыз. Бағдарлама кез келген n мәнімен жұмыс істеуі керек. Бұл үшдиагоналды теңдеулер жүйесі екінші ретті дифференциалдық теңдеулерді ақырлы айырымдық әдіспен шешу кезінде пайда болуы мүмкін. Бағдарламаны $n = 20$ үшін орындаңыз. Нақты шешімді $x_i = -n/4 + i/2, i = 1,2, \dots, n$ деп көрсетуге болады.

Шешуі. Берілген теңдеулер жүйесін (2) формулалар түріне келтіреміз:

$$x_1 = \omega(x_2 - x_n)/2 + (1 - \omega)x_1$$

$$x_i = \omega(x_{i-1} + x_{i+1})/2 + (1 - \omega)x_i, i = 2,3, \dots, n-1$$

$$x_n = \omega(1 - x_1 + x_{n-1})/2 + (1 - \omega)x_n$$

Бұл формулалар iterEqs функциясында есептелінеді.

```
# Мысал1
import numpy as np
import math

def iterEqs(x,omega):
    n = len(x)
    x[0] = omega*(x[1] - x[n-1])/2.0 + (1.0 - omega)*x[0]
    for i in range(1,n-1):
        x[i] = omega*(x[i-1] + x[i+1])/2.0 + (1.0 - omega)*x[i]
    x[n-1] = omega*(1.0 - x[0] + x[n-2])/2.0 + (1.0 - omega)*x[n-1]
    return x
def gaussSeidel(iterEqs,x,tol = 1.0e-9):
    omega = 1.0
    k = 10
    p = 1
    for i in range(1,501):
        xOld = x.copy()
        x = iterEqs(x,omega)
        dx = math.sqrt(np.dot(x-xOld,x-xOld))
        if dx < tol: return x,i,omega
        # k+p итерациясынан кейін релаксация коэффициентін есептеңіз
        if i == k: dx1 = dx
        if i == k + p:
            dx2 = dx
            omega = 2.0/(1.0 + math.sqrt(1.0 - (dx2/dx1)**(1.0/p)))
    print('Гаусс-Зайдель жинақталмады')
```

```
    n = eval(input("Теңдеулер саны ==> "))
    x = np.zeros(n)
    x,numIter,omega = gaussSeidel(iterEqs,x)
    print("\nИтерация саны =",numIter)
    print("\nРелаксация коэффициенті =",omega)
    print("\nШешім :\n",x)
```

Бағдарлама нәтижесі:

```
Теңдеулер саны ==> 20

Итерация саны = 259

Релаксация коэффициенті = 1.7054523107131407

Шешім :
 [-4.50000000e+00 -4.00000000e+00 -3.50000000e+00 -3.00000000e+00
 -2.50000000e+00 -2.00000000e+00 -1.50000000e+00 -9.99999997e-01
 -4.99999998e-01  2.14046747e-09  5.00000002e-01  1.00000000e+00
  1.50000000e+00  2.00000000e+00  2.50000000e+00  3.00000000e+00
  3.50000000e+00  4.00000000e+00  4.50000000e+00  5.00000000e+00]
```

Жинақталуы өте баяу, өйткені коэффициент матрицасында диагональдық үстемдік жоқ, диагональдық үстемдікте шарты

$$|A_{ii}| > \sum_{\substack{j=1 \\ j\neq i}}^{n} |A_{ij}| \quad i = 1,2,\dots,n$$

орындалу керек, ал коэффициенттік А матрица элементтері теңсіздіктен гөрі теңдікті шығарады. Егер коэффициенттің әрбір диагональ мүшесін 2-ден 4-ке өзгертетін болсақ, А диагональ бойынша басым болады және шешім тек 17 итерацияда жинақталады.

Мысал-2

Жоғарыдағы мысалды түйіндес градиенттер әдісімен шешіңіз. $n = 20$ үшін.

Шешуі. Келесі көрсетілген бағдарлама conjGrad функциясын пайдаланады. Шешім векторы x алғашқы мәні бағдарламада нөл деп алынады. $Ax(v)$ функциясы $A \cdot v$ көбейтіндісін қайтарады, мұндағы A – коэффициент матрицасы және v – вектор. Берілген А үшін $Ax(v)$ векторының құрамдастары келесідей

$$(Ax)_1 = 2v_1 - v_2 + v_n$$

$$(Ax)_i = -v_{i-1} + 2v_i - v_{i+1}, i = 2,3,\dots,n-1$$

$$(Ax)_n = -v_{n-1} + 2v_n + v_1$$

```
# Мысал2
import numpy as np
import math

def Ax(v):
    n = len(v)
    Ax = np.zeros(n)
    Ax[0] = 2.0*v[0] - v[1]+v[n-1]
    Ax[1:n-1] = -v[0:n-2] + 2.0*v[1:n-1] -v[2:n]
    Ax[n-1] = -v[n-2] + 2.0*v[n-1] + v[0]
    return Ax
def conjGrad(Av,x,b,tol=1.0e-9):
```

```
    n = len(b)
    r = b - Av(x)
    s = r.copy()
    for i in range(n):
        u = Av(s)
        alpha = np.dot(s,r)/np.dot(s,u)
        x = x + alpha*s
        r = b - Av(x)
        if(math.sqrt(np.dot(r,r))) < tol:
            break
        else:
            beta = -np.dot(r,u)/np.dot(s,u)
            s = r + beta*s
    return x,i
n = eval(input("Теңдеулер саны ==> "))
b = np.zeros(n)
b[n-1] = 1.0
x = np.zeros(n)
x,numIter = conjGrad(Ax,x,b)
print("\nШешім :\n",x)
print("\nИтерация саны =",numIter)
```

Бағдарлама нәтижесі:

```
Теңдеулер саны ==> 20

Шешім :
 [-4.5 -4.  -3.5 -3.  -2.5 -2.  -1.5 -1.  -0.5  0.   0.5  1.   1.5  2.
  2.5  3.   3.5  4.   4.5  5. ]

Итерация саны = 9
```

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