VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMACINIŲ SISTEMŲ INŽINERIJOS STUDIJŲ PROGRAMA

Vienmatis optimizavimas

Laboratorinis darbas

Atliko: 2 kurso 1 grupės studentas
Igor Repkin

Darbo vadovas: prof. dr. Julius Žilinskas

Vilnius - 2023

TURINYS

1. Įvadas

Laboratorinio darbo tikslas yra taikant intervalo dalijimo pusiau, auksinio pjūvio ir Niutono metodus surasti funkcijos minimumą, nustatyti funkcijos reikšmę minimumo taške, skaičiuoti iteracijų skaičių ir apskaičiuoti funkcijų skaičių, palyginti gautus rezultatus ir nustatyti, kuris metodas yra efektyviausias sprendžiant šio tipo uždavinius.

2. Užduotys

2.1. Tikslo funkcija

Šis kodas aprašo tris funkcijas, kurių pagalba galima skaičiuoti tikslo funkcijos reikšmę bei jos pirmąją ir antrąją išvestines. Funkcija funw(x) apskaičiuoja funkcijos reikšmę naudojant įvestą x reikšmę. Tai daro pagal šią formulę: $(x^2 - 6)^2/9 - 1$. Funkcija f(x, stats) naudoja funkciją funw(x) apskaičiuoti funkcijos reikšmę, o funkcija fder(x, stats) apskaičiuoja pirmąją funkcijos išvestinę pagal x. Funkcija fsecder(x, stats) apskaičiuoja antrąją funkcijos išvestinę pagal x ir taip pat įrašo iškvietimų skaičių į žodyną stats.

```
def funw(x):
  return (x**2 - 6)**2 / 9 - 1


def f(x, stats):
  stats['call_count'] += 1
  return funw(x)


def fder(x, stats):
  stats['call_count'] += 1
  return (4 / 9) * x * (x**2 - 6)


def fsecder(x, stats):
  stats['call_count'] += 1
  return (4 / 3) * (x**2 - 2)
```

2.2. Intervalo dalijimo pusiau algoritmas

Šis kodas aprašo intervalo dalijimo pusiau algoritmą, kuris naudojamas funkcijos reikšmių minimizavimui. Funkcija bisection(l, r, deltax) yra pagrindinė funkcija, kuri atlieka intervalo dalijimo pusiau algoritmą su pradiniu intervalu [l, r] ir deltax tikslumo lygiu. Funkcija naudoja žodyną stats, kad sektų intervalo dalijimo pusiau proceso eigos duomenis, tokius kaip žingsnių skaičius, funkcijos kvietimų skaičius ir kiekvieno taško reikšmė. Funkcija susideda iš šešių žingsnių:

1. Nustatyti pradinį tašką, tada apskaičiuoti intervalo ilgį ir funkcijos reikšmę šiame taške.
2. Nustatyti du naujus taškus, vieną viduryje tarp l ir xm, kitą viduryje tarp xm ir r, ir apskaičiuoti jų funkcijos reikšmes.
3. Jei fx1 yra mažesnis nei fxm, tai reiškia, kad intervalas su mažesne funkcijos reikšme yra kairėje, todėl reikia pakeisti tašką r į xm ir xm į x1.
4. Jei fx2 yra mažesnis nei fxm, tai reiškia, kad intervalas su mažesne funkcijos reikšme yra dešinėje, todėl reikia pakeisti tašką l į xm ir xm į x2.
5. Jei fx1 ir fx2 yra didesni nei fxm, tai reiškia, kad intervalas su mažesne funkcijos reikšme yra tarp x1 ir x2, todėl reikia pakeisti intervalo ribas l ir r į x1 ir x2.

6. Atnaujinti intervalo ilgį ir patikrinti, ar jis yra mažesnis už deltax tikslumo lygį.
7. Pabaigoje funkcija grąžina veikimo rezultatus.

```python
def bisection(l, r, deltax):
  stats = {'steps': 0, 'call_count': 0, 'points': [], 'interval': []}

  # Step 1
  xm = (l + r) / 2
  L = r - l
  fxm = f(xm, stats)
  stats['points'].append(xm)  # Save a point
  stats['interval'].append(L)  # Save interval

  while True:
      stats['steps'] += 1
      # Step 2
      x1 = l + L / 4
      fx1 = f(x1, stats)
      x2 = r - L / 4
      fx2 = f(x2, stats)

      # Step 3
      if fx1 < fxm:
          r = xm
          xm = x1
          fxm = fx1
      # Step 4
      elif fx2 < fxm:
          l = xm
          xm = x2
          fxm = fx2
      # Step 5
      else:
          l = x1
          r = x2

      stats['points'].append(xm)  # Save a point
      stats['interval'].append(L)  # Save interval

      # Step 6
      L = r - l
      if L < deltax:
          return (xm, stats, funw(xm))
```

2.3. Auksinio pjūvio algoritmas

Šis kodas aprašo auksinio pjūvio algoritmą. Kodas atlieka šiuos veiksmus:

1. Apskaičiuoja ”t” reikšmę.

2. Toliau, pagal aukso dalijimo metodą, apskaičiuojami pradiniai taškai (x1 ir x2), funkcijos reikšmės šiuose taškuose (fx1 ir fx2).
3. Tada vykdomas ciklas, kuriame apskaičiuojamas funkcijos reikšmė taške x1 ir x2, ir priklausomai nuo to, kurio taško funkcijos reikšmė yra mažesnė, keičiamas intervalo kairysis (jei fx2<fx1) arba dešinysis galas (jei fx1<=fx2).
4. Ciklas tęsiasi, kol intervalo ilgis yra didesnis nei nustatytasis tikslumas (deltax).
5. Pabaigoje funkcija grąžina veikimo rezultatus.

```python
def golden_section(l, r, deltax):
  stats = {'steps': 0, 'call_count': 0, 'points': [], 'interval': []}

  t = (-1 + math.sqrt(5)) / 2

  # Step 1
  L = r - l
  x1 = r - t * L
  fx1 = f(x1, stats)
  x2 = l + t * L
  fx2 = f(x2, stats)
  stats['points'].append((x1 + x2) / 2)  # Save a point
  stats['interval'].append(L)  # Save interval

  while True:
      stats['steps'] += 1
      # Step 2
      if fx2 < fx1:
          l = x1
          L = r - l
          x1 = x2
          fx1 = fx2

          x2 = l + t * L
          fx2 = f(x2, stats)
      # Step 3
      else:
          r = x2
          L = r - l
          x2 = x1
          fx2 = fx1

          x1 = r - t * L
          fx1 = f(x1, stats)

      stats['points'].append((x1 + x2) / 2)  # Save a point
      stats['interval'].append(L)  # Save interval

      # Step 4
      if L < deltax:
```

```python
        return ((x1 + x2) / 2, stats, funw((x1 + x2) / 2))
```

2.4. Niutono metodo algoritmas

Šis kodas aprašo Niutono metodo algoritmą. Kintamasis x0 yra pradinis spėjimas, o deltax yra reikalingas tikslumas. Kiekvienoje iteracijoje yra skaičiuojama nauja reikšmė xinext pagal ankstesnę reikšmę xi. fder ir fsecder yra funkcijos f pirmos ir antros išvestinės atitinkamai. Taip pat skaičiuojamos ir saugomos reikšmės kaip punktai sąraše stats['points']. Jeigu skirtumas tarp xi ir xinext yra mažesnis nei deltax, tada grąžinamas xinext ir stats.

```python
def newtons(x0, deltax):
  stats = {'steps': 0, 'call_count': 0, 'points': [x0], 'interval': []}

  xinext = x0
  while True:
      stats['steps'] += 1
      xi = xinext
      xinext = xi - (fder(xi, stats) / fsecder(xi, stats))

      stats['points'].append(xinext)  # Save a point
      stats['interval'].append(abs(xi - xinext))  # Save interval

      if abs(xi - xinext) < deltax:
          return (xinext, stats, funw(xinext))
```

3. Rezultatai, palyginimas ir išvados

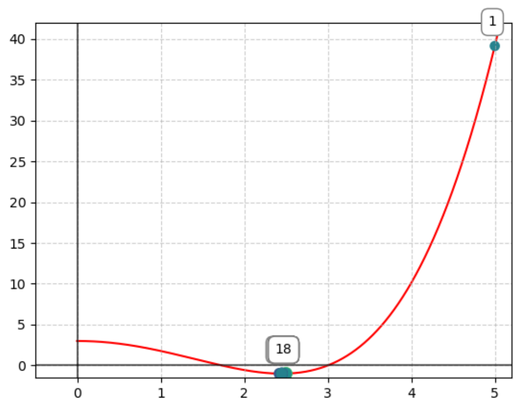
Mažiausiai žingsnių atliko Niutono metodo algoritmas (žr. **??** lent.) kadangi pasirinktas $x_0 = 5$ buvo netoli funkcijos minimumo $x_{min} = 2.5$. Niutono metodo algoritmas iškarto nusileido į funkcijos minimumą (žr. **??** pav.), o pvz. auksinio pjūvio algoritmas kelis kartus pasikeitė pusę iš kurios jis ieško funkcijos minimumą (žr. **??** pav.). Auksinio pjūvio algoritmas padarė daugiausiai iteracijų (žingsnių skaičius) (žr. **??** lent.), bet jis mažiau kartu skaičiavo funkcijos reikšmę negu Intervalo dalijimo pusiau algoritmas (žr. **??** pav.). Mano konkrečiam atvejui efektyviausias skaičiavimo metodas – Niutono metodo algoritmas, nes jis atliko mažiausiai funkcijos F skaičiavimų bei iteracijų.

1 lentelė. Algoritmų žingsnių skaičiaus ir F skaičiavimų sk. palyginimas.

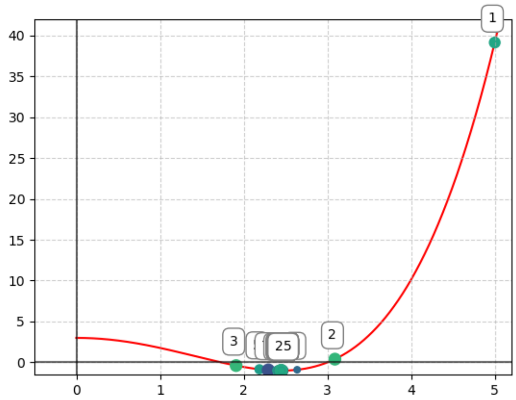
Pavadinimas	Žingsnių sk.	F skaičiavimų sk.
Intervalo dalijimo pusiau alg.	17	35
Auksinio pjūvio alg.	24	26
Niutono metodo alg.	6	12

2 lentelė. Gauti sprendiniai, rastas funkcijos minimumo įvertis.

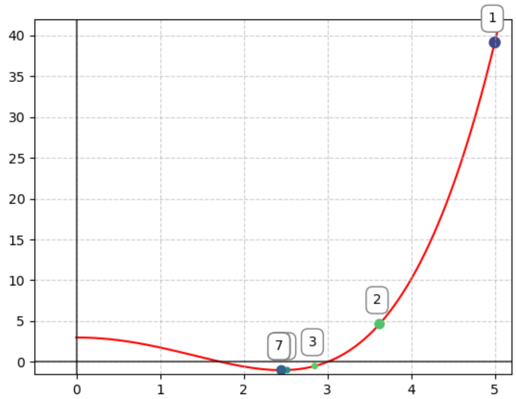
Pavadinimas	Sprendimas	Minimumo įvertis
Intervalo dalijimo pusiau alg.	2.449493408203125	-0.9999999999641724
Auksinio pjūvio alg.	2.449462010049481	-0.9999999979490778
Niutono metodo alg.	2.449489742799229	-1.0



1 pav. Intervalo dalijimo pusiau algoritmo veikimo rezultatai.



2 pav. Auksinio pjūvio algoritmo veikimo rezultatai.



3 pav. Niutono metodo algoritmo veikimo rezultatai.