SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR

Vizualizacija algoritama optimizacije

Marija Kalebota

Voditelj: *Izv. prof. dr. sc. Domagoj Jakobović*

Zagreb, svibanj, 2017.

Sadržaj

1. Uvod 3

2. Algoritmi optimizacije 4

2.1 Algoritam pronalaženja unimodalnog intervala funkcije 4

2.2 Algoritam zlatnog reza 5

2.3 Algoritam Hooke-Jeeves 5

3. Jupyter bilježnice 6

3.1 IPython Widgets 6

4. Implementacija 8

4.1 Primjer korištenja 10

5. Zaključak 13

6. Sažetak 14

7. Literatura 15

1. Uvod

Algoritmi optimizacije postupci su koji su vrlo široko primjenjivi u inženjerskoj struci. Pri procesu učenja načina na koji oni funkcioniraju, studentima mnogo pomažu vizualni alati. Kroz ovaj se seminarski rad pokušava implementirati vizualizacija koja prikazuje funkciju cilja i kretanje trenutno pronađenog rješenja kroz vrijeme, tj. tijekom izvođenja algoritma, a također i moduli koji bi učinili zadavanje i mijenjanje parametara algoritma interaktivnima, kako bi korisnik u stvarnom vremenu mogao vidjeti utjecaj različitih parametara na rezultate algoritama.

2. Algoritmi optimizacije

Algoritmi optimizacije algoritmi su koji se koriste za pronalaženje globalnog optimuma (minimuma ili maksimuma) matematičkih funkcija. Primjena tih algoritama učestala je kada se matematičkom funkcijom opisuju svojstva ili pojave iz stvarnog svijeta koje je cilj što više pojačati ili što više umanjiti. Primjerice, opišemo li matematičkom funkcijom odstupanje nekog rješenja od zadanih kriterija, pronalaženje minimuma te funkcije omogućit će nam pronalazak najmanjeg mogućeg odstupanja, koje nam je i najpoželjnije.

2.1 Algoritam pronalaženja unimodalnog intervala funkcije

Mnogi algoritmi optimizacije primjenjivi su samo na unimodalne funkcije, tj. na funkcije koje imaju samo jedan optimum na području definicije. Nisu sve funkcije takve, no ako je barem neki njihov dio unimodalan, algoritam optimizacije možemo provesti nad tim dijelom. Kako bismo ga pronašli, koristimo algoritam za pronalaženje unimodalnog intervala funkcije. U nastavku je opisana inačica algoritma koja traži unimodalni interval funkcije jedne varijable na kojem se nalazi minimum.

Koraci tog algoritma su sljedeći:

1. Odaberi točku iz područja definicije funkcije – to je x, početna točka algoritma
2. Pomakni se u smjeru apscise ulijevo i udesno od točke x za $2^1 * h$, gdje je h parametar algoritma (obično se uzima $h = 1$). Očitaj vrijednosti funkcije cilja u tim točkama.
3. Ako očitane vrijednosti pokazuju pad funkcije u jednom smjeru, povećavaj eksponent nad dvojkom koja množi h u tom smjeru dok vrijednost funkcije u promatranoj točki ne postane veća od one u prethodnom koraku.
4. Točke $(x \pm 2^{i-1} * h)$ i $(x \pm 2^i * h)$ iz zadnja dva koraka algoritma (gdje i označava redni broj koraka algoritma) granice su unimodalnog intervala.

2.2 Algoritam zlatnog reza

Algoritam zlatnog reza algoritam je optimizacije za unimodalne funkcije jedne varijable. Kao rješenje daje interval proizvoljno male duljine $\mathcal{E}$ (parametar algoritma, obično se uzima $\mathcal{E} = 10^{-6}$), pa kao optimum funkcije možemo uzeti bilo koju točku iz tog intervala.

Ovaj algoritam radi tako da u svakom koraku smanji duljinu intervala u kojem zna da se nalazi optimum funkcije. To će činiti tako da u svakom koraku pogleda dvije točke unutar intervala, i onu u kojoj je vrijednost funkcije lošija (kod minimizacije – onu u kojoj je viša) uzeti kao novi rub intervala. To bi se moglo učiniti, primjerice, dijeleći interval svaki put na tri dijela, no tada bi se u svakom koraku trebale računati dvije nove točke. Izaberemo li takav omjer, pak, da se točka iz prethodnog intervala koja nije izabrana za novi rub može opet iskoristiti kao jedna od dvije točke iz novog koraka, dobijemo upravo omjer zlatnog reza – k ≈ 0.618. Algoritam tako smanjuje interval u kojem se nalazi optimum dok on ne postane manji ili jednak $\mathcal{E}$.

2.3 Algoritam Hooke-Jeeves

Algoritam Hooke-Jeeves postupak je optimizacije za unimodalne funkcije više varijabli. Krene od početne točke x_0, a parametri su mu duljina pomaka Δx, faktor smanjenja pomaka k ($0 < k < 1$) i tolerancija točnosti $\mathcal{E}$ (obično $\mathcal{E} = 10^{-6}$). Radi tako da u svakom koraku izračuna tri točke: baznu točku x_b, početnu točku x_p (na početku su obje jednake x_0) i na temelju njih izračuna novu točku x_n. Nju dobije tako da se po svakoj dimenziji točke x_p pomakne u smjeru koordinatne osi za Δx ulijevo i udesno, odabere „najbolju" točku po iznosu funkcije od te tri (kod minimizacije, odabere točku u kojoj je vrijednost funkcije najniža). U sljedećem koraku točke x_b i x_p izračunava po formulama $x_{p(i)} = 2* x_{n(i-1)} - x_{b(i-1)}$ i $x_{b(i)} = x_{n(i-1)}$. To ponavlja dok god je vrijednost funkcije u točki x_n „bolja" (kod minimizacije – niža) od vrijednosti funkcije u točki x_b. Kada se to dogodi, smanjimo duljinu pomaka Δx za faktor k i ponavljamo algoritam (postavivši točke x_b i x_p na x_b iz prethodnog koraka), sve dok duljina pomaka Δx ne postane manja ili jednaka $\mathcal{E}$. Kao rješenje uzima se točka x_b ili x_n iz zadnjeg koraka algoritma.

3. Jupyter bilježnice

Jupyter bilježnice interaktivan su alat koji se sastoji od sljedeće tri komponente:

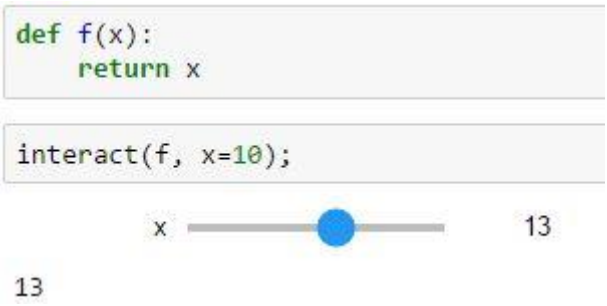
1. Mrežna aplikacija bilježnice omogućuje korisnicima da u internetskom pregledniku mijenjaju i pokreću programski kod, prikazuju interaktivne grafičke elemente i dodaju popratni tekst koristeći jezik za označavanje Markdown
2. Jezgra pokreće programski kod zapisan u bilježnici. Postoje jezgre za izvođenje raznih programskih jezika, a početna je jezgra ona za programski jezik Python
3. Datoteka bilježnice sadrži potpunu povijest jedne sjednice računanja. Njen sadržaj zapisan je u formatu JSON, a neke su vrijednosti zapisane u binarnom formatu, što znači da se može pročitati i obraditi velikim brojem različitih programskih jezika.

Uzimajući u obzir širok opseg mogućnosti koje Jupyter bilježnice nude i prirodu optimizacijskih algoritama da u svakom koraku postoji nešto što se može nazvati „trenutnim rješenjem”, te su se bilježnice pokazale kao prikladan alat za interaktivnu vizualizaciju koraka tih algoritama kroz vrijeme.

3.1 IPython Widgets

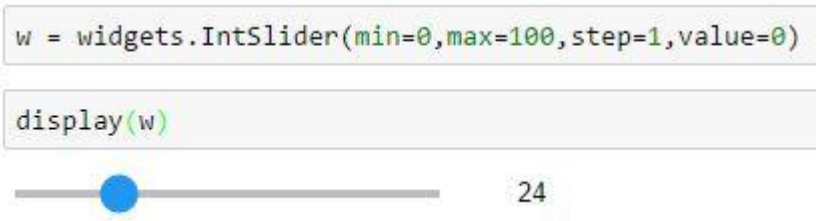
Glavni modul korišten u sklopu ovog rada modul je IPython Widgets. To je proširenje za Jupyter bilježnice koji omogućuje jednostavnu implementaciju vizualnih i interaktivnih elemenata. Neke njegove mogućnosti opisane su u nastavku.

Funkcija *interact()* kao argment prima metodu, a služi tome da argumente te metode, tj. ulaze metode koji utječu na njen izlaz, učini interaktivnima. Tako, primjerice, ako metoda kao argument prima podatak tipa *int*, funkcija *interact()* za taj podatak izgenerira klizač na kojem je moguće kao argument za metodu izabrati neki broj iz intervala. Također, ako je argument metode podatak tipa *boolean*, funkcija *interact()* će automatski korisniku ponuditi kvadratić za izbor (engl. *tick box*).



Slika 1. – Primjer korištenja funkcije *interact()*

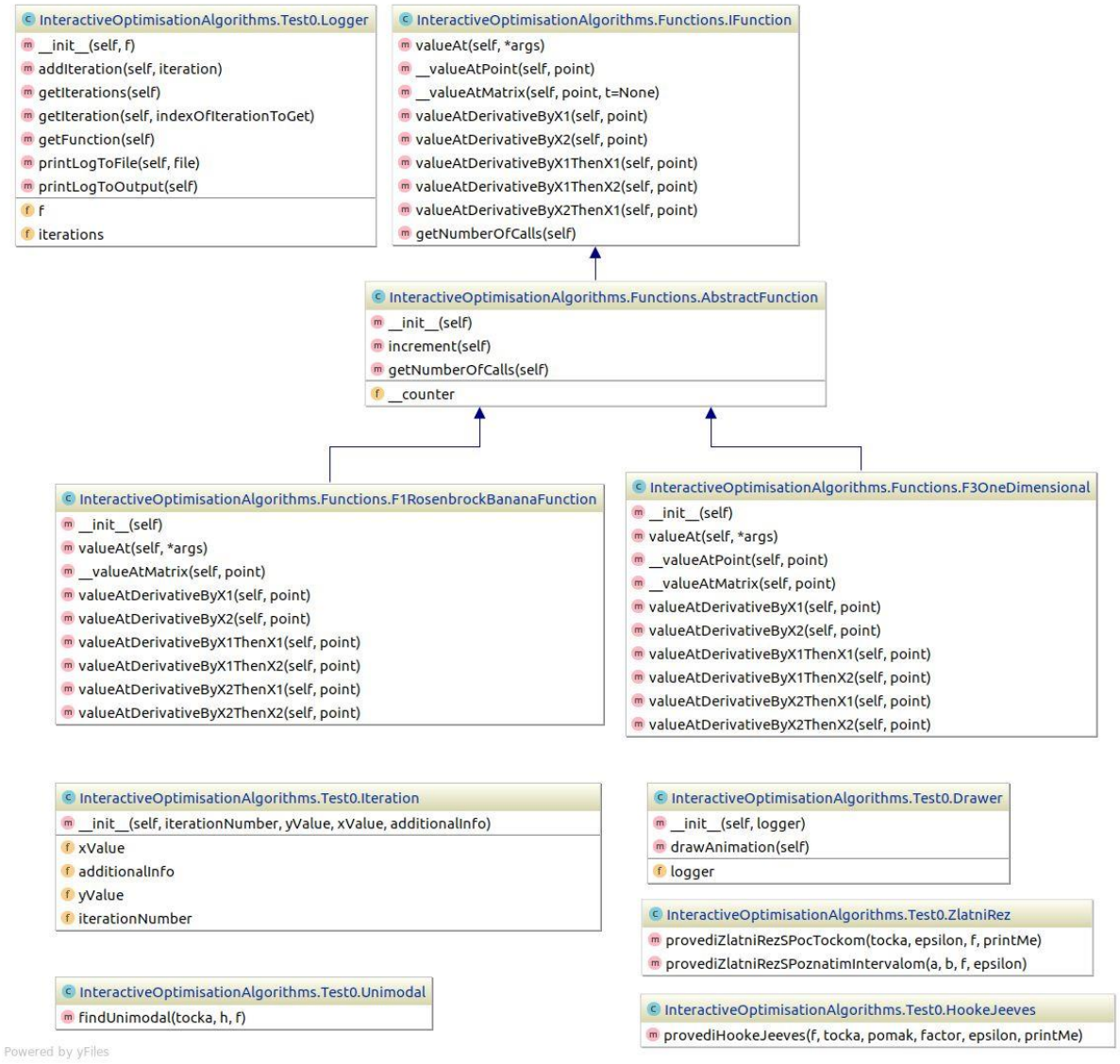
Raznovrsne komponente koje funkcija *interact()* može izgenerirati zapravo su objekti tipa *widget* (razred *widgets*). Te je objekte moguće stvoriti i spremiti u varijablu baš kao i objekte bilo kojeg drugog razreda. To nam omogućuje da im namještamo parametre i prikazujemo ih na mjestima gdje nam odgovara.



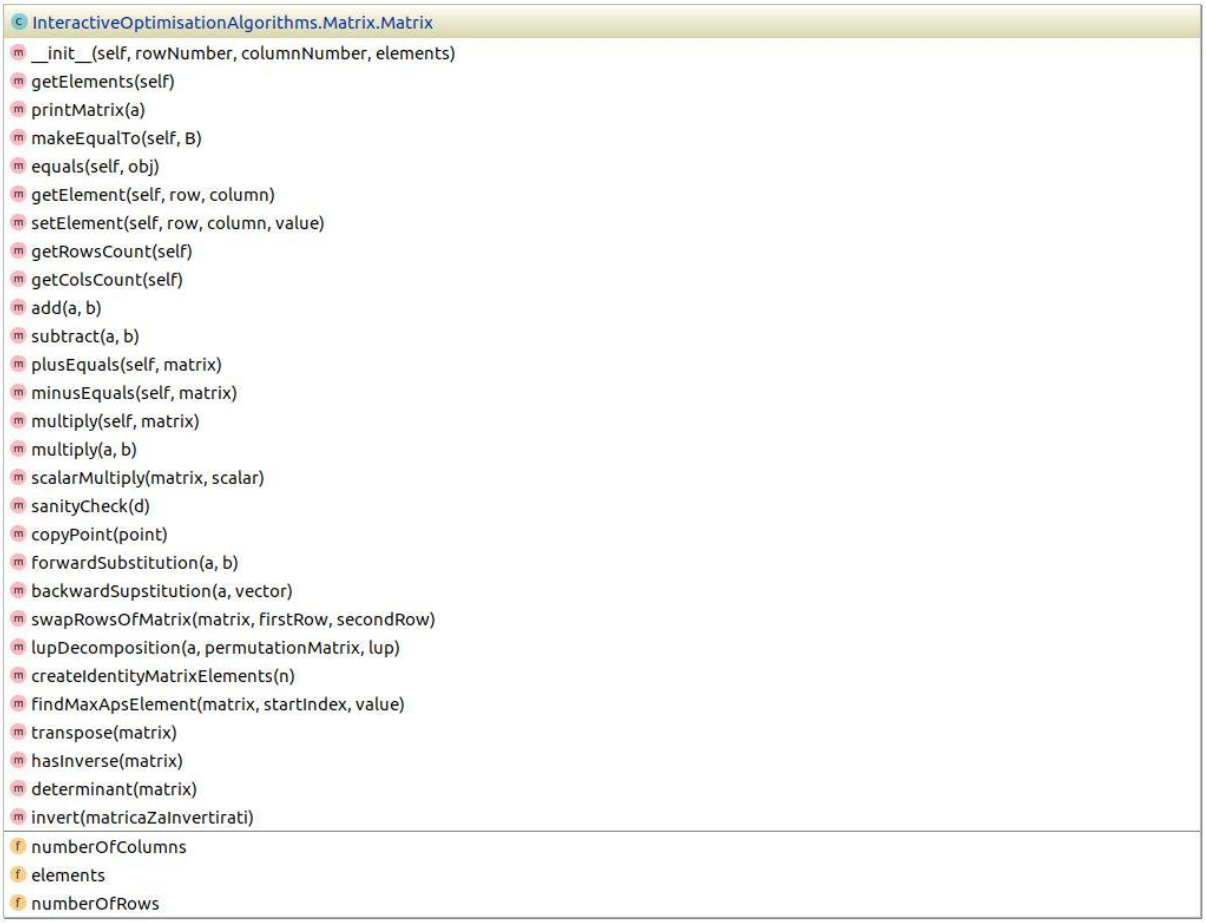
Slika 2. – Primjer korištenja objekta tipa *widget*

4. Implementacija

Za implementaciju vizualizacije algoritama optimizacije i popratne interaktivnosti korištene su Jupyter bilježnice i elementi tipa IntSlider i Play widget iz modula Ipython Widgets. Za implementaciju rada algoritama optimizacije napisani su popratni razredi u programskom jeziku Python. U nastavku je okviran opis korištenih razreda i dijagram razreda.



Slika 3.a – Dio dijagrama razreda



Slika 3.b – Dio dijagama razreda

Razred Matrix – razred koji sadrži većinu osnovnih funkcionalnosti za rad s matricama, poput njihovog zbrajanja, oduzimanja, množenja, usporedbe i sličnog.

Razredi IFunction i AbstractFunction – razredi koji služe tome da ih naslijede konkretne implementacije funkcija.

Razredi Unimodal, ZlatniRez i HookeJeeves – razredi u kojima su ostvarene implementacije algoritama optimizacije podrobnije opisani u poglavlju 2.

Razred Iteration – razred koji služi za spremanje jedne iteracije algoritma optimizacije. Sadrži trenutno pronađenu najbolju točku, vrijednost funkcije cilja u toj točki, redni broj iteracije i strukturu za spremanje popratnih informacija koje se mogu razlikovati od algoritma do algoritma.

Razred Logger – razred koji služi za spremanje iteracija algoritma.

Razred Drawer – razred koji služi za iscrtavanje interaktivnih elemenata koji služe za vizualizaciju rada algoritma. Kao argument prima objekt tipa Logger iz kojeg sazna sve podatke koji su mu za to potrebni.

Važno svojstvo implementacije koja je opisana u ovom radu je modularnost. Potrebne funkcionalnosti podijeljene su na ovaj način na različite razrede kako bi se u budućnosti lakše moglo implementirati nove algoritme optimizacije i nove načine vizualizacije.

4.1 Primjer korištenja

```
f3 = F3OneDimensional()
```

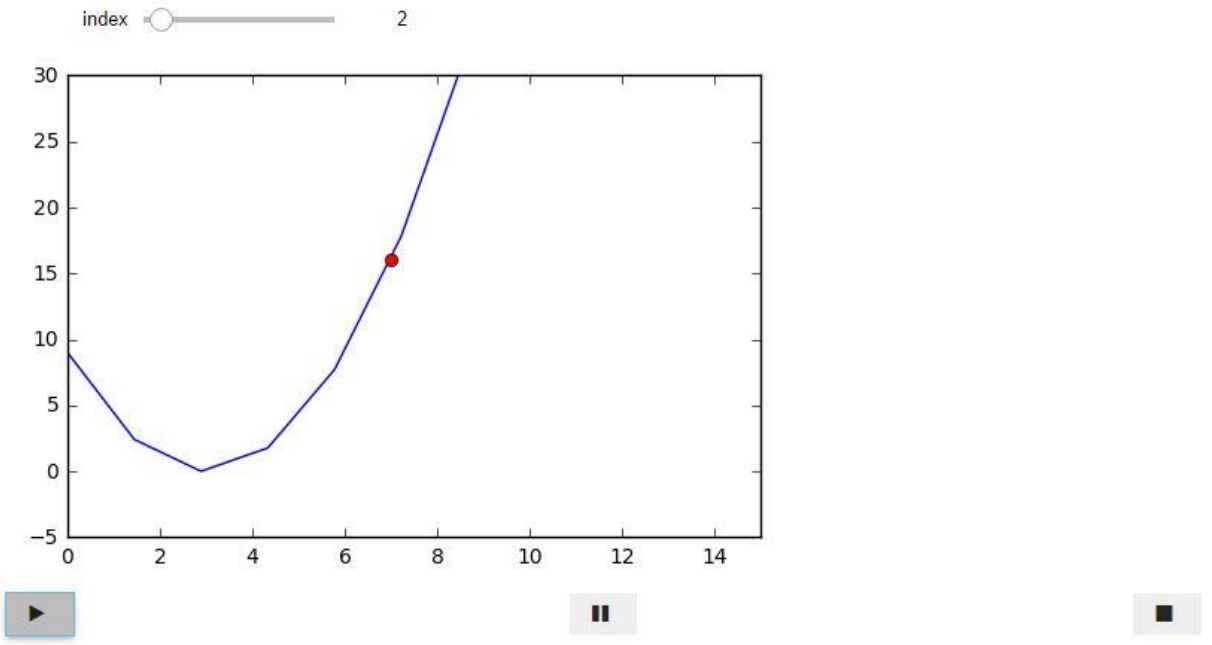
```
elements = np.array([[10]])
tocka = Matrix(1, 1, elements)
```

```
rjesenjeHookeJeeves, loggerHookeJeeves = HookeJeeves.provediHookeJeeves(f3, tocka, 1, 0.5, 1E-6, False)
```

```
Konacno rjesenje Hooke-Jeevesa za pocetnu tocku 10 je [[ 3.]]
```

```
drawer = Drawer(loggerHookeJeeves)
```

```
drawer.drawAnimation()
```



Slika 4. – Primjer interaktivne vizualizacije algoritma Hooke-Jeeves

Na slici 4. prikazan je primjer interaktivne vizualizacije algoritma Hooke-Jeeves. Prvo stvorimo varijablu *f3* tipa *F3OneDimensional*, razreda koji nasljeđuje razrede *AbstractFunction* i *IFunction*. Nakon toga stvorimo varijablu *tocka* tipa *Matrix*, i te dvije varijable pošaljemo u statičku metodu *provediHookeJeeves* razreda *HookeJeeves*, koja uspješno provede algoritam i ispiše konačan rezultat. Ta je metoda također vratila to konačno rješenje u varijablu *rjesenjeHookeJeeves*, i zapis o iteracijama u varijablu *loggerHookeJeeves* tipa *Logger*. Nakon toga stvorimo varijablu dawer tipa *Drawer*, kojoj u konsruktor predamo upravo izgeneriranu varijablu *loggerHookeJeeves*. Potom pozovemo metodu *drawAnimation()*, i ona nam prikaže graf funkcije *f3* i trenutno pronađenu najbolju točku ovisno o odabranom broju iteracije. Broj iteracije može se izabrati pomicanjem klizača, ili se može pokrenuti animacija pritiskom na gumb „*play*”.

Pogledamo li podrobnije tekst razreda Drawer (slika 5.), bit će nam jasnije na koji način radi metoda drawAnimation. Ona prvo stvori polja koordinata točaka iz iteracija algoritma koje će se iscrtavati na grafu. Potom stvori dva „*widgeta*” – klizač (eng. *slider*) *w* i modul Play imena *play* i spoji ih metodom widgets.jslink. Na taj način, kada se pritisne gumb „play” na modulu Play, pomiče se i klizač. Potom se definira funkcija za iscrtavanje *f*, i konačno se pozovu dvije metode koje prikazuju ove elemente na ekranu – *interact()* i *display()*.

```
class Drawer:
    def __init__(self, logger):
        self.logger = logger

    def drawAnimation(self):
        playMaxOfInterval = 0
        poljeX = []
        poljeY = []
        for iteration in self.logger.getIterations():
            playMaxOfInterval = playMaxOfInterval + 1
            # stvori polje ovih brojeva
            poljeX.append(iteration.xValue)
            poljeY.append(iteration.yValue)

        w = widgets.IntSlider(min=0, max=playMaxOfInterval - 1, step=1, value=0)
        play = widgets.Play(
            value=0,
            min=0,
            max=playMaxOfInterval,
            step=1,
            description="Press play",
            disabled=False
        )
        widgets.jslink((play, 'value'), (w, 'value'))
        def f(poljeX, poljeY, index):
            funkcija = self.logger.getFunction()
            plt.clf()
            plt.close('all')
            plt.figure(index)
            plt.axis([0.0, 15.0, -5.0, 30.0])
            ax = plt.gca()
            ax.set_autoscale_on(False)

            X = np.linspace(0.0, 13.0, num=10)
            Y = [funkcija.valueAt(x) for x in X]

            plt.plot(X, Y, 'b')
            plt.plot(poljeX[index], poljeY[index], 'ro')
            plt.show()

        interact(f, poljeX=fixed(poljeX), poljeY = fixed(poljeY), index=w)
        display(play)
```

Slika 5. – Tekst razreda *Drawer*

5. Zaključak

Optimizacijski algoritmi postupci su pronalaženja optimuma neke funkcije. Ima ih mnogo i ponekad ih je samo čitanjem matematičkih formula teško pojmiti, a ponekad i razumjeti. Interaktivnost i vizualizacija koje pružaju Jupyter bilježnice može riješiti upravo taj problem – može omogućiti lakše shvaćanje i poimanje algoritama optimizacije, pogotovo zbog toga jer većina tih algoritama radi iterativno.

U sklopu ovog rada implementirani su elementi koji omogućuju vizualizaciju optimizacijskih algoritama, ali, još važnije, ostvaren je cilj implementacije strukture koja se lako može nadograđivati, shodno dobroj programerskoj praksi.

6. Sažetak

U sklopu ovog seminarskog rada razvijen je okvir koji omogućuje interaktivnu vizualizaciju algoritama optimizacije. Cilj mu je omogućiti korisniku da pomicanjem klizača i/ili pokretanjem, pauziranjem i zaustavljanjem animacije djeluje na vizualni prikaz iteracija algoritama optimizacije kroz vrijeme. Pisan je u programskom jeziku Python, a kao alat za vizualizaciju korištene su Jupyter bilježnice i modul IPython Widgets.

7. Literatura

[1] Budin, L., Analiza i projektiranje računalom - skripta s predavanjima, Zagreb

[2] Jupyter Team, What is the Jupyter Notebook?, http://jupyter-notebook.readthedocs.io/en/latest/examples/Notebook/What%20is%20the%20Jupyter%20Notebook.html, 9.5.2017.

[3] Jupyter Team, ipywidgets: User Guide, https://ipywidgets.readthedocs.io/en/latest/index.html, 9.5.2017.