Preklad optimalizačných metód z jazyka Matlab do jazyka Python

Patrik Grman

24.6.2020

Problém

Na katedre aplikovanej matematiky a štatistiky sa pri výučbe predmetu Metódy voľnej optimalizácie na praktické ukážky používa jazyk Matlab. Ukazuje sa ale, že je vhodné použiť radšej niečo voľne dostupné, čo napríklad umožní študentom ľahšie experimentovať na vlastných počítačoch. Za týmto účelom vznikla požiadavka preložiť existujúci kód do jazyka Python verzie 3, keďže verzia 2 je EOL od začiatku roka 2020.

Zimný semester

Najprv bolo treba zistiť, či nejdeme riešiť už vyriešený problém. Ukázalo sa ale, že veľa nástrojov robí s Pythonom 2, a často Matlab skôr emulujú ako prekladajú. Bolo teda treba zistiť čo a ako sa má preložiť, aby to fungovalo pod Python verziou 3.

Časť 1: Zistenie množiny používanej funkcionality

V prvej fáze som sa zameral na existujúci kód. Potreboval som zistiť, aká funkcionalita sa reálne používa, aby som vedel čo treba preložiť. Bolo nájdených okolo 30 rôznych relevantných vecí v niekoľkých kategóriách, napríklad syntax a vstavané funkcie. Všetky zistenia boli zapísané do súboru pre neskoršie použitie.

Časť 2: Nájdenie ekvivalentov pre jazyk Python

Následne bolo nutné nájsť pythonovské ekvivalenty k veciam najdeným v prvej časti. Niektoré funkcie buď priamo existujú alebo sú ľahko emulovateľné v základnom Pythone.

Na veľa pokročilých matematickych operacií som zvolil knižnicu numpy, ktorá je de-facto štandardom pre náročnejšiu matematiku v Pythone. Na vykresľovanie grafov bola zvolená knižnica matplotlib, ktorej modul pyplot má rozhranie veľmi podobné matlabovým grafovým funkciám, čo umožnilo relatívne jednoduchú adaptáciu.

Samozrejme ani jedna knižnica neposkytuje funkcinalitu úplne identickú matlabovskému ekvivalentu, ale skombinovaním malého počtu funkcií sa vždy dala emulovať.

Letný semester

V letnom semestri bolo treba sformalizovať ako sa má robiť preklad samotný, a podľa možnosti aj preložiť čo treba.

Časť 3: Spísanie manuálu pre ručný preklad

Na základe znalostí oboch jazykov a použitim vedomostí z druhej časti som napísal manuál pre ručný preklad. Manuál je napísaný v jednoduchom jazyku Markdown, a automaticky je z neho generovaná pdf verzia.

Následne som ho priebežne upravoval a vylepšoval, keď sa počas práce na nasledúcich častiach ukázali niektoré dovtedy nespozorované skutočnosti.

Časť 4: Automatizácia prekladu

Už od začiatku bolo pomerne jasné, že preklad použitím čiste regulárnych jazykových prostriedkov, prípadne vlastného zložitejšieho mechanizmu by bol problematický a výrazne náchylný na chyby. Preto som sa rozhodol použiť niektorú voľne dostupnú technológiu na parsovanie bezkontextových jazykov.

Konkrétne som použil nástroj ANTLR, ktorý na základe formálne definovanej gramatiky vygeneruje kód ktorý realizuje samotné parsovanie. Ďalšia výhoda použitia existujúceho nástroja bola možnosť použiť predpripravenú gramatiku pre Matlab, určenú pre tento nástroj, aj ked sa nakoniec ukázala potreba gramatiku opakovane upravovať a rozširovať, vzhľadom na chábajúcu funkcionalitu.

Na generovanie výstupu som zvolil knižnicu StringTemplate, ktorá umožňuje veľmi pohodlné generovanie Pythonového kódu vrátane interného riešenia odsadenia.

Na preklad samotný som s výhodou použil návrhový vzor visitor, keďže parser generuje zakorenený strom reprezentujúici program rozdelený podľa pravidiel gramatiky.

Prekladač implementuje len nutné minimum funkcionality, keďže som nechcel riešiť neexistujúci problém. Je schopný prekladať buď jednotlivé súbory alebo (nerekurzívne) priečinky. Nástroj má momentálne len rozhranie použiteľné z príkazového riadku, na vytvorenie uživatelskeho rozhrania som využil knižnicu Apache Commons CLI.

Časť 5: Testovanie rýchlosti behu

Následne som pre zaujímavosť realizoval meranie na porovnanie rýchlosti vykonávania týchto dvoch jazykov. Oba jazyky su interpretované, preto ide v skutočnosti o porovnanie rýchlosti implementacií interpretera, prípadne knižničného kódu, ktorý napríklad numpy realizuje ako natívny kód.

Konkrétne som zvolil implementácie, ktoré su zadarmo verejne dostupné a štandardne sa používaj, teda pre Python je to CPython a pre Matlab je to GNU Octave.

Rýchlosť sa testovala vo viacerých disciplínach:

- Jednoduchý cyklus, ktorý sčítaval čísla od 0 po n - na ňom sa ukáže ako rýchlo beží interpreter pre jednoduché operácie
- Rekurzívny výpočet n-tého čísla Fibonacciho postupnosti - ukazuje výkon pri volaní funkcií
- Násobenie štvorcových matíc v cykle neefektívne počítajúc n-tú mocninu matice - rýchlosť matematického kódu

Kódy pre Matlab

```
% disciplina 1
function total= mark0(n)
    total=0;
    for i=0:n
        total=total+i;
```

```
    end
end
```

```
% disciplina 2
function x = mark1(n)
   if(n <= 1)
      x=1;
   else
      x=mark1(n-1)+mark1(n-2);
   end
end
```

```
% disciplina 3
function result=mark2(mat,n)
    result=mat;
    for i=0:n
        result=result*mat;
    end
end
```

Kódy pre Python

Kód bol preložený automatickým prekladačom a následne odstránené v tomto prípade nepotrebné zátvorky a voľné riadky

```
# disciplina 1
def mark0(n):
    total = 0
    for i in range(0, n + 1):
        total = total + i
    return total
```

```
# disciplina 2
def mark1(n):
    if n <= 1:
        x = 1
    else:
        x = mark1(n - 1) + mark1(n - 2)
    return x
```

```
# disciplina 3
def mark2(mat, n):
    result = mat
    for i in range(0 , n + 1):
        result = result * mat
    return result
```

Postup testovania

Postup pre oba jazyky bol rovnaký: použitím interných hodiniek sa odmeralo trvanie 10 spustení jedneho testu pre daný argument a vypočítala sa priemerná doba jedného spustenia. Ak táto doba