PRAKTYCZNE ASPEKTY MODELOWANIA RZECZYWISTOŚCI

Daniel Wójcik
Instytut Biologii Doświadczalnej PAN

d.wojcik@nencki.gov.pl

tel. 022 5892 424

http://www.neuroinf.pl/Members/danek/swps/

Wprowadzenie do Octave

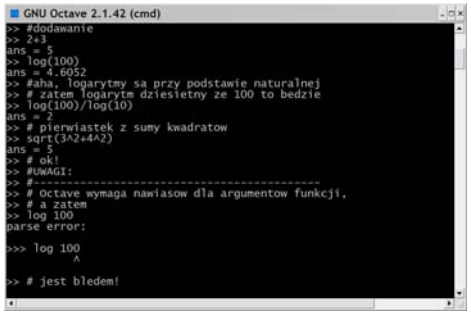
- Uruchamianie
- Octave jako kalkulator
- Podstawowe komendy
- Pisanie skryptu
- Uruchamianie skryptu

Uruchamianie Octave

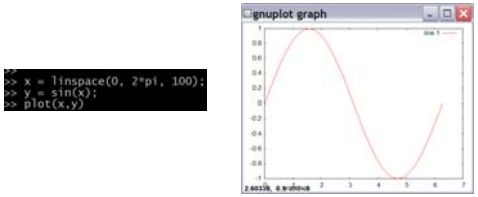
Klikamy ikonę

lub uruchamiamy Octave z menu

Po znaku zachęty w ostatniej linijce podajemy komendy



Rysowanie

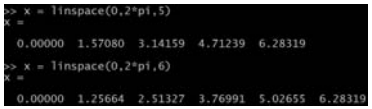


Komenda

```
linspace(pocz,koniec,ile)
```

tworzy wektor o 'ile' elementach, którego pierwszy element to 'pocz', ostatni to 'koniec', a pozostałe są rozłożone równomiernie.

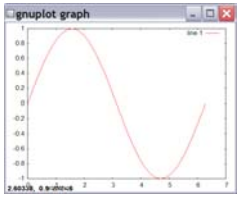
Przykład:



Komenda `plot(x,y)` tworzy dwuwymiarowy wykres z par punktów, których współrzędne x są podane w pierwszym wektorze, a współrzędne y w drugim. Oba wektory muszą mieć tą samą długość.

Spójrzmy jeszcze raz:

```
>> x = linspace(0, 2*pi, 100);
>> y = sin(x);
>> plot(x,y)
```

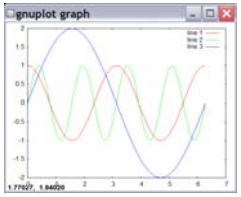


Co ciekawe, komenda `sin` liczy sinus *wszystkich* elementów macierzy x jednocześnie!

Kolejny przykład rysowania

Komenda `hold on` służy do tego, żeby kilka linii wydrukować na jednym rysunku:

```
>> x = linspace(0,2*pi);
>> a = cos(2*x);
>> b = sin(4*x);
>> c = 2*sin(x);
>> plot(x,a);
>> hold on
>> plot(x,b);
>> plot(x,c);
```



A teraz sprawdźmy:

```
hold off;
plot(x,a+b+c);
```

Elementy składni

Zmienne zaczynają się od litery, potem mogą być cyfry, podkreślenia

Pewne stałe są wstępnie zdefiniowane w Octave, na przykład

```
pi, e, i
```

Średnik powoduje, że wynik działania komendy nie jest wypisywany na ekranie. Porównaj

```
linspace(0,2*pi);
linspace(0,2*pi)
```

Skrypty

Możemy zachować grupę komend na później w tak zwanym skrypcie. Żeby stworzyć skrypt o nazwie 'mojskrypt' piszemy

```
edit mojskrypt
```

Taki skrypt może zawierać definicję naszej funkcji, albo po prostu zbiór komend, które chcemy uruchamiać po kolei

Ćwiczenie

Narysuj wykres okręgu o środku w zerze i promieniu 1

Narysuj wykres swojej ulubionej funkcji

UWAGA: Żeby narysować wykres iloczynu funkcji, np. sin x * cos x, musimy użyć operatora “.*”

```
plot(x, sin(x) .* cos(x));
```

Macierze, albo tablice, to uporządkowane zbiory liczb. Najprostsze z nich, to wektory – ciągi liczb. Najczęściej używane – macierze prostokątne.

Tworzymy wektor wierszowy

Tworzymy wektor kolumnowy

Tworzymy macierz prostokątną

```
>> x = [1, 3, 2]
x =
   1   3   2
>> y = [1; 2; 3]
y =
   1
   2
   3
>> A = [1, 2, 3; 4, 5, 6]
A =
   1   2   3
   4   5   6
```

Take it or leave it

Pierwsza symulacja:

Kiedy wszystko inne zawiedzie
obniż standardy

Przypuśćmy, że aplikujesz o pracę w wielu miejscach. Po kilku dniach otrzymujesz pierwszą ofertę. Czy przyjmiesz ją czy odrzucisz?

Jeżeli ją przyjmiesz, może się okazać, że kolejne oferty będą znacznie lepsze i będziesz żałował, że nie poczekałeś.

Jeżeli ją odrzucisz, może się okazać, że tak dobra oferta już się nie powtórzy, albo że będziesz musiał czekać na lepszą bardzo długo.

Podejmujemy takie decyzje w oparciu o pewien zestaw kryteriów, które każdy z nas ma. W przypadku pracy mogą to być pewne oczekiwania:

1. Pensja co najmniej 5000 zł
2. Praca w miejscu zamieszkania
3. Samodzielny pokój
4. Duże biurko
5. Przynajmniej miesiąc urlopu w ciągu roku
6. Umowa o pracę na czas niekreślony
7. Elastyczne godziny pracy
8. Twórcze zadania
9. Dużo możliwości podróży służbowych
10. Częste podwyżki

Powiedzmy, że pierwsza oferta spełnia tylko 6 tych kryteriów. Czy przyjmiesz ją, czy odrzucisz? Druga może spełniać tylko 4 kryteria. Przyjmiesz ją? Jeżeli nie, czy dostaniesz kolejną ofertę?

Ogólniej, jaki jest związek między minimalną liczbą kryteriów, której wymagamy, a liczbą ofert, które musimy odrzucić, zanim taka oferta się pojawi?

Aby to zbadać utwórzmy w Octave wiele list potencjalnych ofert, powiedzmy po 1000 ofert, opisywanych przez liczbę spełnianych przez nie naszych kryteriów.

Początkowo przyjmijmy, że oferty są losowane z rozkładu normalnego o średniej 6 i odchyleniu standardowym 2. Zatem około 68% wszystkich ofert będzie spełniało od 4 do 8 cech, 95% pomiędzy 2 a 10, a 99% pomiędzy 0 a 12.

```
>> jobs = ceil(6+2*randn(1,1000));
>> jobs(1:5)
ans =
     6     5     8     8     7

>> jobs = ceil(6+2*randn(1,1000));
>> jobs(1:5)
ans =
     3     2     9     8     6
```

Omówić elementy

Widzimy, że za każdym razem otrzymujemy inną listę ofert. Zapiszmy to w edytorze:

```
>> edit takeit
```

```
% Tworzymy wektor 1000 ofert, z ktorych kazda
% jest opisana liczba pozadanych przez nas
% cech. Oferty losujemy z rozkladu normalnego
% o sredniej 6 i odchyleniu standardowym 2
oferty = ceil (6 + 2*randn(1,1000));
```

Załóżmy, że oczekujemy oferty, spełniającej przynajmniej 8 naszych kryteriów.

```
>> jobs = ceil(6+2*randn(1,1000));
>> jobs(1:5)
ans =
       6       5       8       8       7

>> jobs = ceil(6+2*randn(1,1000));
>> jobs(1:5)
ans =
       3       2       9       8       6
```

Jeżeli pierwsza oferta spełnia 3 kryteria, druga 2, a trzecia 9, to przyjmujemy trzecią.

Jak znaleźć dobre oferty?
Użyjemy komendy `find`

```
>> jobs = ceil(6+2*randn(1,1000));
>> jobs(1:14)
ans =

 Columns 1 through 7:

       7       6       2       6       6       7       5

 Columns 8 through 14:

       8       6       5       7       9       9       6

>> good_offers = find(jobs >= 8 );
>> good_offers(1:7)
ans =
       8      12      13      17      18      19      25
```

Zatem komenda find znajduje wszystkie oferty spełniające nasze oczekiwania. W tym przypadku znajduje te oferty, które spełniają 8 naszych kryteriów.

Wpiszmy to do naszego programu!

```
% Tworzymy wektor 1000 ofert, z ktorych kazda
% jest opisana liczba pozadanych przez nas
% cech. Oferty losujemy z rozkladu normalnego
% o sredniej 6 i odchyleniu standardowym 2
oferty = ceil (6 + 2*randn(1,1000));

% znajdujemy teraz numery ofert spelniajacych
% nasze kryteria (minimum 8)
dobre_oferty = find(oferty >= 8 );
pierwsza_dobra = dobre_oferty(1)
```

Napisaliśmy naszą pierwszą symulację.Zobaczmy jak działa:

```
>> takeit
first_good = 7
>> takeit
first_good = 4
>> takeit
first_good = 2
>> takeit
first_good = 2
```

Ponieważ każda symulacja daje inny wynik, żeby powiedzieć coś o typowym zachowaniu, musimy uruchomić nasz program kilkakrotnie i policzyć średnią:

```
>> takeit
first_good = 7
>> takeit
first_good = 4
>> takeit
first_good = 2
>> takeit
first_good = 2
```

```
>> mean([7 4 2 2])
ans =
  3.7500
```

Oczywiście, cztery wyniki to trochę mało, żeby liczyć średnie. Z drugiej strony, nie będziemy przecież uruchamiać ręcznie naszego programu 1000 razy!

Skorzystamy z pętli `for`

```
>> for proba = 1:10
proba/2
endfor
```

Co ten kod robi?

Skorzystajmy z pętli `for` w naszym programie:

```
>> for proba = 1:100
% wykonaj dotychczasowy program
endfor
```

Musimy jeszcze gdzieś zapamiętać wyniki. W tym celu utwórzmy macierz `wyniki`

```
wyniki = zeros(1,1000);
for proba = 1:1000
  % tu dotychczasowy program

  wyniki(proba) = dobre_oferty(1);
endfor

mean(wyniki)
```

Żeby móc eksperymentować z programem zamieńmy liczbę oczekiwanych cech na zmienną, której wartość będziemy wczytywać na początku.

Zatem zamiast 8 napiszmy `ile_cech`, a na początku programu dodajmy linijkę:

```
ile_cech = input('ile cech tym razem? ')
```

```
>> takeit
ile cech tym razem? 5
ile_cech = 5
ans = 1.2090
>> takeit
ile cech tym razem? 6
ile_cech = 6
ans = 1.4350
>> takeit
ile cech tym razem? 7
ile_cech = 7
ans = 2.0140
>> takeit
ile cech tym razem? 8
ile_cech = 8
ans = 3.2720
>> takeit
ile cech tym razem? 9
ile_cech = 9
ans = 6.3660
>> takeit
ile cech tym razem? 10
ile_cech = 10
ans = 15.176
>> takeit
ile cech tym razem? 11
ile_cech = 11
ans = 42.389
>> x = [5 6 7 8 9 10 11];
>> y = [1.2090 1.4350 2.0140 3.2720 6.3660 15.176 47.389];
>> plot(x,y)
```

Widzimy więc, że kiedy liczba cech, których oczekujemy, przekracza dwa odchylenia standardowe od średniej, czas oczekiwania na dobrą ofertę rośnie bardzo szybko.

Zatem jeżeli żadna z ofert, które otrzymujemy, nie spełnia naszych oczekiwań, może powinniśmy je obniżyć?

Mamy teraz gotowy program, który jest dobrym punktem wyjścia do dalszych eksperymentów. Możemy zmieniać

- parametry rozkładu, z którego wybieramy oferty,
- nasze oczekiwania,
- sposób wyboru oferty, itd.

Państwa zadaniem będzie teraz:

- Poeksperymentować z programem tak, żeby wszystkie jego elementy były dla Państwa zrozumiałe
- Napisać sprawozdanie z eksperymentów. Sprawozdanie powinno zawierać:
  - Przedstawienie problemu
  - Kod programu z opisem, co on robi
  - Opis przeprowadzonych eksperymentów z rysunkami przedstawiającymi wyniki
  - Podsumowanie zawierające wnioski z przeprowadzonych symulacji

Przykładowe warianty eksperymentów

Jak zmieniają się wyniki dla różnych rozkładów? Różnych wartości średniej i odchylenia standardowego?

Czy można sformułować ogólną zależność pomiędzy czasem oczekiwania, liczbą ofert odrzuconych, a średnią i wariancją rozkładu?

A gdyby odrzucać pierwszą, a przyjmować drugą. Czy wtedy zmienia się wartość oczekiwana jakości ofert?

A gdyby tak mieć podwójne standardy: np. 8 i 10. Jeżeli oferta jest >= 10, bierzemy; jeżeli jest <10 ale >=8, bierzesz drugą, czy coś zyskujemy w ten sposób? Na przykład nieznaczne wydłużenie czasu oczekiwania przy poprawie średniego wyniku?

Jak wykorzystać rysunek z gnuplota?

Naciskamy prawy klawisz myszki i wybieramy
Options -> Copy to clipboard