Zadania do poćwiczenia

Paweł Czachorowski

31 stycznia 2025

Poniższy zestaw zadań jest przeznaczony dla osób, które chciałyby poćwiczyć coś ponad to co oferują zadania na Wiki przedmiotu. Nie są one w żaden sposób oceniane ani sprawdzane. Oczywiście zachęcam do kontaktu jeżeli pojawiłyby się jakieś trudności, pytania czy uwagi.

Ostatnie dwa zadania są bardziej złożone i mają postać projektów do samodzielnego podłubania. Są to dość „akademickie" problemy i można znaleźć w sieci jakieś ich rozwiązania, niemniej przerabia się je zwykle na przedmiotach bardziej dedykowanych metodom numerycznym niż nasz kurs. Myślę, że ich przerobienie mogłoby być ciekawe dla osób głębiej zainteresowanych tematem i stanowić okazję do poszerzenia wiedzy.

1 Analiza listy

Dana jest lista: `lista=["Nagłówek",0,1,-2,-3.4,5.6,7.8,"Koniec"]`. Wiemy, że pierwszy i ostatni element zawierają nie interesujące nas dane. Napisz program, który stworzy i wyświetli nową listę, zawierającą jedynie te spośród istotnych elementów, które są większe od zera. Nie używaj pętli.

2 Liczenie do dziesięciu

Spróbuj uruchomić poniższy kod. Co się dzieje? Skąd ten błąd?

```
x=[]
for i in range(1,10):
    x.append(i)
    print(x[i])
```

3 Sklej-fu

Napisz kod w Pythonie, który wygeneruje listę zawierającą wartości funkcji $x^3 + 2x^2 - 1$ dla liczb całkowitych od 1 do 10. Spróbuj zrobić to bez użycia pętli, jedynie wykorzystując listy sklejane (*list comprehension*, np. `https://www.w3schools.com/python/python_lists_comprehension.asp`).

4 Sklej-fu 2: powrót floata

Jak wyżej, tym razem niech wartości x nie będą liczbami całkowitymi, tylko 10 liczbami zmiennoprzecinkowymi wylosowanymi z przedziału $[0, 2]$ za pomocą `random.random()` z zaimportowanego modułu `random` (`https://docs.python.org/3/library/random.html`). Dasz radę zapisać to w jednej linii (nie licząc importu oczywiście)?

5 Sklej-fu 3: Numpy kontratakuje

Jak w poprzednim zadaniu, tyle że wszystko robimy w `numpy` – jednowymiarowe tablice `numpy` zamiast zwykłych list, `random` też już nie ma po co importować, bo `numpy` ma przecież swoje własne procedury do losowania.

6 Losowy obrazek

Napisz program, który wygeneruje tablicę Numpy o wielkości 100×100, zawierającą losowe liczby z zakresu $[0, 1)$ (np. używając `numpy.random.random()`), podniesie jej elementy do kwadratu, a następnie wyświetli wynik jako obrazek za pomocą funkcji `matplotlib.pyplot.matshow()`. Dla czytelności, w tej ostatniej funkcji użyj opcjonalnego argumentu `cmap="Greys"`. Testując, warto pamiętać o istnieniu `numpy.random.seed()`, by zapewnić powtarzalność „losowych" wyników.

7 Negatyw

Zmodyfikuj program z poprzedniego zadania tak, by wygenerował negatyw obrazka. Następnie stwórz jedną tablicę 100×200, przechowującą obok siebie oba obrazki, i ją wyświetl. Może przydać się funkcja `numpy.hstack()`. Czy wyświetlone razem w ten sposób tablice wyglądają tak samo jak oddzielnie? Jeśli nie, spróbuj przesunąć wartości jednej z tablic do tego samego zakresu co druga.

8 Opłaty za media

W pliku `https://www.fuw.edu.pl/~pczachorowski/data/teaching/Python2024/media.dat` umieściłem przykładowe stany liczników prądu i wody spisywane w ok. miesięcznych odstępach. Napisz program, który wygeneruje z nich tabelę opłat za ostatni okres (miesiąc), podobną do tej pokazanej poniżej.

```
Dzień       Godzina |Energia[kWh] Koszt[zł]  |Ciepła[m3] Koszt[zł]  |Zimna[m3] Koszt[zł] |Suma[zł]
-------------------------------------------------------------------------------------------------
01.02.2022  11:00   | 44.800      47.67      | 1.556     27.75       | 1.451     5.17     | 80.59
05.03.2022  21:35   | 60.700      57.85      | 1.558     27.85       | 1.846     9.70     | 95.39
03.04.2022  12:00   | 47.000      49.08      | 1.457     22.81       | 1.818     9.37     | 81.26
02.05.2022  15:59   | 57.900      56.06      | 1.059      2.94       | 1.804     9.21     | 68.21
03.06.2022  9:00    | 58.200      56.25      | 1.382     19.07       | 1.971    11.13     | 86.44
03.07.2022  9:50    | 54.000      53.56      | 1.733     36.58       | 1.942    10.80     |100.94
-------------------------------------------------------------------------------------------------
TEN MIESIĄC         | 54.000      53.56      | 1.733     36.58       | 1.942    10.80     |100.94
```

Przyjmujemy, że:

- Koszt prądu to 0.64zł/kWh plus stał opłata za licznik w wysokości 19zł
- Ciepła woda kosztuje 49.91zł/m^3, ale pierwszy $1m^3$ w każdym okresie nie jest uwzględniany
- Zimna woda kosztuje 11.46zł/m^3, ale pierwszy $1m^3$ w każdym okresie nie jest uwzględniany

9 Opłaty za media – wizualizacja

Przedstaw dane z poprzedniego zadania w formie wykresów:

1. Pierwszy wykres (liniowy) niech przedstawia sumaryczny koszt w kolejnych miesiącach, z dodatkowo zaznaczoną (jako pozioma linia) wartością średnią, zaznaczonym na czerwono punktem odpowiadającym największej wartości oraz niebieskim punktem odpowiadającym wartości najniższej.

2. Trzy wykresy dla trzech pozostałych kolumn z kosztami niech znajdą się jeden pod drugim na jednym płótnie (`numpy.subplots`). Również, jak poprzednio, należy oznaczyć średnią i punkty największej i najmniejszej wartości.

3. Wykresy kołowe dla poszczególnych miesięcy (na jednym płótnie, po trzy wykresy w rzędzie), ilustrujące wkład energii oraz wody ciepłej i zimnej w całkowitym koszcie.

10 Wielokąty foremne

Napisz program, który dla podanego `N` będzie rysował N-kąt foremny (*wskazówka*: jak za pomocą `numpy.linspace` narysować okrąg?). Następnie spróbuj zmodyfikować go tak by:

- figury były wypełnione
- program rysował wielokąty gwiaździste foremne (pentagram i o większej liczbie wierzchołków)

11 Rzut ukośny

Na zajęciach przeanalizowaliśmy przypadek spadku swobodnego/rzutu pionowego (`https://www.fuw.edu.pl/~pczachorowski/data/teaching/Python2024/freefall.py`). Spróbuj uogólnić przypadek na dwa wymiary – tzn. w pionie sytuacja jest dokładnie taka sama jak poprzednio, ale pojawia się dodatkowa składowa pozioma.

12 Ukryty obrazek

Na zajęciach wyekstrahowaliśmy z jednego obrazka (`https://www.fuw.edu.pl/~pczachorowski/data/teaching/Python2024/obrazek.png`) drugi, ukryty w nim za pomocą wartości przezroczystości nieznacznie mniejszej od 1 (`https://www.fuw.edu.pl/~pczachorowski/data/teaching/Python2024/extract.py`). Spróbuj przeprowadzić operację odwrotną – samodzielnie ukryć jeden obrazek w drugim. Możesz użyć tych samych obrazków co ja (wstępnie już przekonwertowałem pierwszy na `.png`, a drugi nieco obrobiłem):
`https://www.fuw.edu.pl/~pczachorowski/data/teaching/Python2024/computer.png`[1],
`https://www.fuw.edu.pl/~pczachorowski/data/teaching/Python2024/feynman.png`[2]).
Wskazówka: spróbowałbym dodać wartości RGB ukrywanego obrazka i zaokrąglić je do 1 lub 0 (może się przydać polecenie `numpy.rint`).

13 Współczynnik wirialny i rysowanie wykresu z pliku

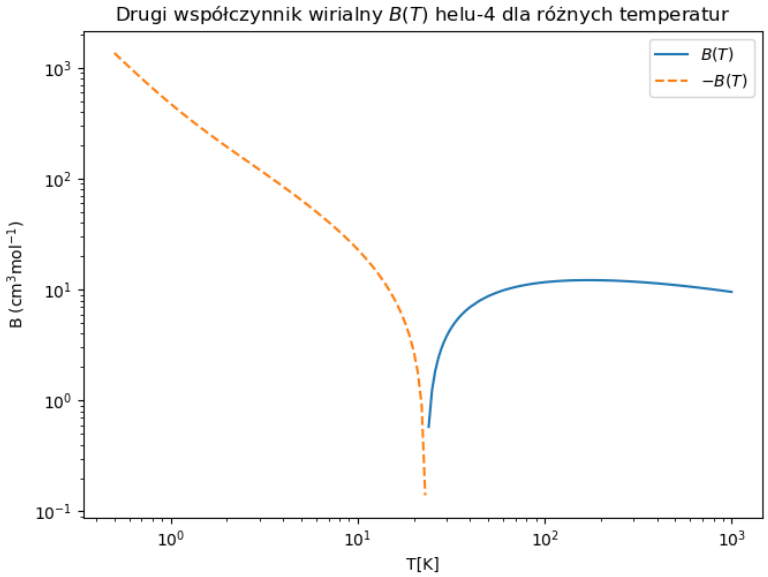
Z pewnością z lekcji chemii znasz równanie stanu gazu doskonałego (czasem nazywane równaniem Clapeyrona), wiążące ze sobą parametry termodynamiczne gazu w układzie. Jak sama nazwa wskazuje, czyni ono pewne wyidealizowane założenia na temat gazu, w szczególności zaniedbując oddziaływania między jego cząsteczkami. Pewnym „urealnieniem" tego modelu jest równanie Van der Waalsa, aczkolwiek nadal uwzględnia ono tylko pewne efekty. Potencjalnie najdokładniejszym, a przy tym dość zdroworozsądkowym podejściem jest tzw. równanie wirialne, które można zapisać między innymi tak:

$$p = k_\mathrm{B} T \left[\rho + B(T)\rho^2 + C(T)\rho^3 + \cdots\right],$$

gdzie p – ciśnienie, k_B – stała Boltzmanna, T – temperatura, ρ – gęstość liczbowa (liczba cząsteczek na jednostkę objętości). Równanie to wychodzi z równania stanu gazu doskonałego jako pierwszego przybliżenia i dodaje do niego kolejne poprawki, proporcjonalne do kolejnych potęg gęstości. Poprawki te zawierają tzw *współczynniki wirialne* ($B(T)$ – drugi wsp. wirialny, $C(T)$ – trzeci etc.), zależne od oddziaływań międzycząsteczkowych ($B(T)$ zależy od oddziaływań par cząsteczek, $C(T)$ od trójek itd.). Co ciekawe, współczynniki te można obliczyć teoretycznie (wg. reguł mechaniki kwantowej, np. `https://arxiv.org/abs/2007.09767`).

Pobierz plik `https://arxiv.org/src/2007.09767v2/anc/S4_virial_helium-4.txt`, zawierający wartości drugiego współczynnika wirialnego dla helu-4. Napisz program, który wczyta z niego temperatury i odpowiadające im wartości $B(T)$, a następnie stworzy wykres taki jak poniższy – tzn. obie osie będą w skali logarytmicznej oraz oddzielnie wyrysowane będą wartości ujemne i dodatnie[3]. Pomóc może funkcja `matplotlib.pyplot.xscale("log")` oraz analogiczna dla osi y. Wczytanie danych można oczywiście zrobić tak jak robiliśmy to już niegdyś, niemniej polecam też zerknąć na funkcję `numpy.loadtxt()` oraz jej opcjonalny argument `skiprows`. Jeśli chcesz, możesz się zastanowić jak połączyć obie części wykresu (dla ujemnych i dodatnich wartości $B(T)$) tak, by łączyły się ze sobą.

[1] Źródło: `https://commons.wikimedia.org/wiki/File:Computer_hacked_eighth_day_event_proposal.jpg`
[2] Źródło: `https://commons.wikimedia.org/wiki/File:Feynman.png`
[3] Z oczywistej przyczyny – logarytmy liczb ujemnych nie są rzeczywiste!



14 Metoda złotego podziału

Na ćwiczeniach spotkaliśmy się już z metodą bisekcji. Służyła ona do znajdowania miejsc zerowych funkcji w sposób numeryczny – a więc taki, który wymagał od nas jedynie podania wartości tej funkcji dla wybranych wartości zmiennej, bez potrzeby wchodzenia w badanie jej własności matematycznych (czyli rozwiązania problemu w sposób analityczny). W podobny sposób można poszukiwać lokalnych ekstremów funkcji. Bywa to szczególnie przydatne w sytuacjach, gdy wartość funkcji zależy od zmiennej w jakiś wyjątkowo zawikłany sposób[4]. Odpowiednikiem bisekcji jest tu metoda złotego podziału (`https://pl.wikipedia.org/wiki/Metoda_z%C5%82otego_podzia%C5%82u`, także polecam Numerical Recipes (nieważne w jakiej wersji języka programowania, w naszej bibliotece jest kilka, np. `https://omnis-buw.primo.exlibrisgroup.com/permalink/48OMNIS_UOW/1tfbgh2/alma991003672349708831`).

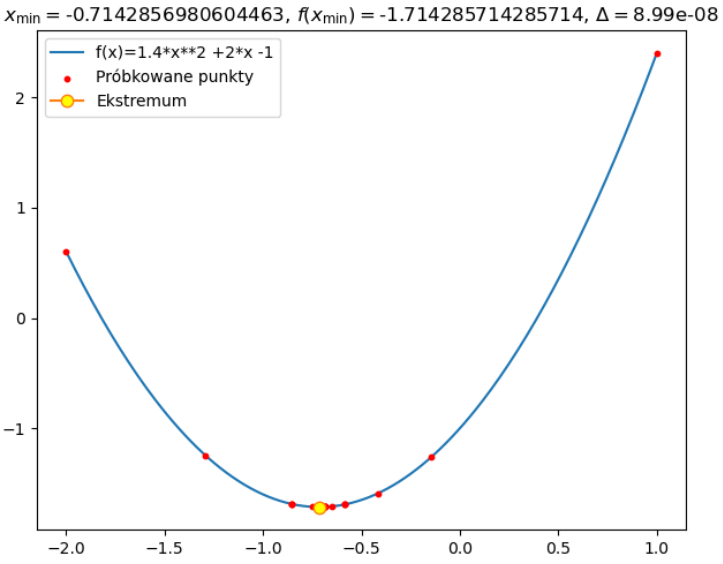
W przypadku bisekcji potrzebowaliśmy w danym kroku wartości funkcji w dwóch punktach, by znaleźć miejsce zerowe. Wyznaczanie nowego punktu dla kolejnego kroku w połowie między nimi również było dość naturalne. Gdyby podzielić przedział inaczej niż symetrycznie na pół, ryzykowalibyśmy, że poszukiwane miejsce zerowe będzie w większym kawałku (a więc mniej dokładnie je obejmiemy). W przypadku poszukiwania ekstremum, dwa punkty nie wystarczą, bo przecież musimy znaleźć taki kawałek funkcji, w którym najpierw ona rośnie, a potem maleje (bądź odwrotnie). Potrzebujemy więc trzech punktów, zaś sprawa optymalnego próbkowania (wybierania gdzie policzyć wartość funkcji) staje się nieco mniej oczywista. Co interesujące, w zaskakujący sposób powraca tu złoty podział związany z liczbą Fibonacciego, z którą już się zetknęliśmy w zadaniu z szeregiem[5].

Celem zadania jest napisanie kodu w Pythonie, który dla zadanego przedziału $x \in [a, b]$, dokładności (rozumianej jako $\varepsilon > |b - a|$ w ostatnim kroku) oraz funkcji $f(x)$ poszukuje metodą złotego podziału jej minimum w tym przedziale i wypisuje odpowiadające mu wartości $x_{\min}$ oraz $f(x_{\min})$. Niech program umożliwia również wybór poszukiwania maksimum (co można łatwo zrealizować jako poszukiwanie minimum funkcji z przeciwnym znakiem) oraz prezentuje wynik na wykresie, na którym będą zaznaczone wszystkie punkty przebadane w trakcie poszukiwania oraz (innym kolorem) samo ekstremum.

Gwoli porównania, dla $f(x) = 1.4x^2 + 2x - 1$ na przedziale $x \in [-2, 1]$ minimum to $x_{\min} = -0.7142857142857143$, $f(x_{\min}) = -1.7142857142857144$. Wynik przykładowego programu zaprezentowany jest poniżej (przyjęto $\varepsilon = 1 \cdot 10^{-7}$). Na wykresie $\Delta = b - a$, czyli faktyczna rozpiętość analizowanego przedziału w ostatnim kroku (jak widać jest mniejsza niż ε, czyli tak jak miało być).

[4] Np. w chemii kwantowej często poszukuje się najlepiej opisującej układ funkcji falowej poprzez manipulowanie jej parametrami, powtarzanie z tak zmienioną funkcją złożonych i czasochłonnych obliczeń i sprawdzanie dla jakich parametrów uzyska się najniższą energię. Ciężko tu podać bezpośredni wzór na minimum energii jako funkcji wspomnianych parametrów, więc stosuje się podejście numeryczne.

[5] W skrócie chodzi o to samo co w bisekcji – żeby zminimalizować (sic!) ryzyko związane z tym, że poszukiwane ekstremum jest w mniej korzystnym kawałku przedziału.



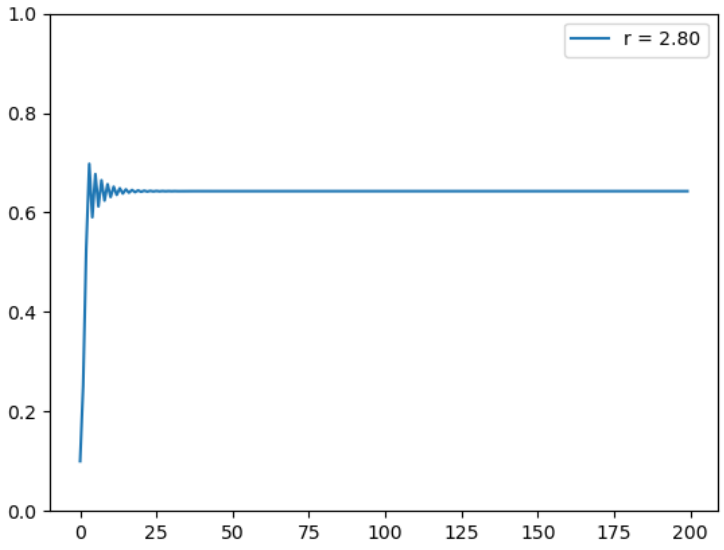
15 Odwzorowanie logistyczne

Odwzorowanie logistyczne (`https://pl.wikipedia.org/wiki/Odwzorowanie_logistyczne`, nieco więcej również w angielskiej wersji) to stosunkowo prosta rekurencja:

$$x_{n+1} = r x_n (1 - x_n).$$

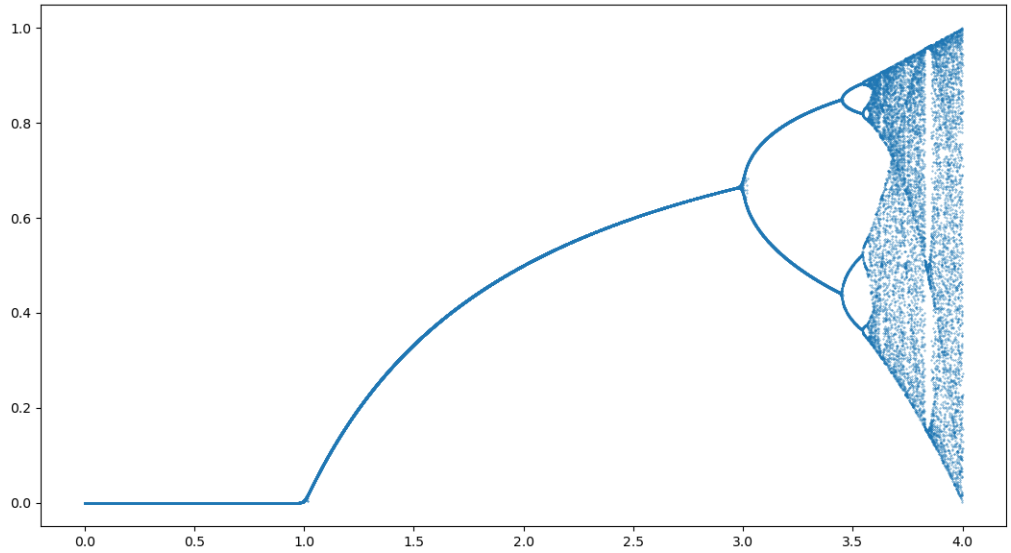
Mimo mało skomplikowanej postaci, posiada ona interesujące własności. Można jej użyć do opisu dynamiki populacji, gdzie $r \in [0, 4]$ określa się mianem współczynnika rozrodczości, który mówi o tempie przyrostu populacji, zaś $x \in [0, 1]$ opisuje stosunek obecnej liczności w populacji do maksymalnej możliwej. Posługując się tym prostym wzorem, można zobrazować jak przyrastać i wymierać (np. ze względu na brak pożywienia) będzie populacja jakichś organizmów.

Problemem do rozwiązania jest implementacja powyższego modelu. Program ma przyjmować (jako parametry wywołania) wartość współczynnika r, liczbę kroków N (a więc jakie wartości ma przebiegać n) oraz początkową wartość x (x_0). Na tej podstawie powinien wyznaczyć kolejne wartości x_n i narysować je na wykresie. Np. dla zestawu parametrów $r = 2.8, N = 200, x_0 = 0.1$, wykres powinien wyglądać mniej więcej tak (populacja rośnie, trochę oscyluje, po czym ustala się na stabilnym poziomie):



Warto przyjrzeć się jak zmienia się przebieg wykresu dla różnych kombinacji parametrów. Przy wartościach $r \in 1, 2, 3, 3.56995$ powinny zachodzić jakościowe zmiany w zachowaniu się wykresu. Istotne jest tu pojęcie atraktora – stanu (w naszym przypadku wartości x), do którego układ będzie dążył. Wraz ze wzrostem r powinny pojawiać się kolejne atraktory, pomiędzy którymi układ będzie przeskakiwał. Początkowa wartość x_0 – mało istotna dla małych wartości r, dla dużych r zacznie mieć ogromny wpływ na przebieg wykresu, co jest charakterystyczne dla układów chaotycznych.

Można to jeszcze lepiej zilustrować na kilka sposobów. Po pierwsze, program powinien mieć opcję animacji wykresu – tak, by r stopniowo zwiększało się od 0 do 4. Przydadzą się tu komendy `plt.pause(T)`, a także `plt.clf()`. Po drugie, powinien umieć stworzyć tzw. diagram bifurkacyjny - na którym dla każdej wartości r zaznaczone są wartości atraktorów. Można to osiągnąć w dość zwięzły sposób wykorzystując Numpy i dotychczasowy kod programu. Wzór na rekurencję jest dokładnie ten sam, jedynie r to teraz jednowymiarowa tablica numpy (z wartościami od 0 do 4), podobnie jak zresztą x. Jeżeli weźmiemy wystarczająco duże tablice (np. 100000 elementów) oraz x zbudujemy z losowych wartości z przedziału $[0, 1]$ (`np.random.random()`), końcowe wartości tablicy x powinny (ze względu na chaotyczność układu) zawierać wszystkie interesujące nas atraktory. Do wyrysowania takiego wykresu przyda się `plt.scatter()`. Przykładowy diagram bifurkacyjny wygląda tak:


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0