Programowanie i metody numeryczne

Ćwiczenia 7.

Aproksymacja funkcji.

Zadanie 1. Interpolacja wielomianowa.

a) Napisz funkcję `int_lagrange`, przyjmującą jako argumenty dwie listy liczb, X oraz Y oraz liczbę rzeczywistą x. Funkcja powinna wyznaczać wartość wielomianu interpolacyjnego Lagrange'a w punkcie x, otrzymanego na podstawie punktów (X, Y).

b) Napisz funkcję `int_newton`, przyjmującą jako argumenty dwie listy liczb, X oraz Y oraz liczbę rzeczywistą x. Funkcja powinna wyznaczać wartość wielomianu interpolacyjnego Newtona w punkcie x, otrzymanego na podstawie punktów (X, Y). Wykorzystaj algorytm różnic dzielonych.

c) Napisz program `int`, który przyjmuje jako argumenty wywołania dwie listy liczb i rysuje wykres obu wielomianów interpolacyjnych. Czy widać między nimi jakąś różnicę?

d) Napisz program `intsin`, rysuje oba wielomiany interpolacyjne dla punktów uzyskanych z funkcji $\sin x$ dla $N + 1$ węzłów, gdzie liczba N jest argumentem wywołania programu, oraz wykresy błędu bezwzględnego i względnego w obu przypadkach.

Zadanie 2. Efekt Rungego.

Rozważmy następujące funkcje gładkie, określone na przedziale $[-1, 1]$:

$$f(x) = \sin(\pi x), \qquad g(x) = \sin(5\pi x), \qquad h(x) = \frac{1}{1 + 8x^2}.$$

a) Napisz program `rungef`, przyjmujący jako argument wywołania liczbę naturalną N. Program powinien dwukrotnie wyznaczyć wielomian interpolacyjny Newtona $p(x)$ dla punktów postaci $\big(x_n,\, f(x_n)\big)$, $n = 0, 1, \ldots, N$, jako węzły (x_n) wybierając: za pierwszym razem – węzły równoodległe, zaś za drugim – węzły Czebyszewa.

Wynikiem działania programu powinny być cztery wykresy: wielomianu interpolacyjnego z węzłami równoodległymi oraz – w tym samym układzie współrzędnych – funkcji $f(x)$; błędu bezwzględnego interpolacji z węzłami równoodległymi; wielomianu interpolacyjnego z węzłami Czebyszewa oraz – w tym samym układzie współrzędnych – funkcji $f(x)$; błędu bezwzględnego interpolacji z węzłami Czebyszewa. Umieść wykresy na jednym rysunku, obok siebie, tak, by łatwo było je porównywać.

W celu wykonania interpolacji wykorzystaj odpowiednie funkcje biblioteczne dostępne dla języka programowania albo pakietu obliczeń naukowych, z którego korzystasz.

b) Napisz programy `rungeg` i `rungeh`, działające tak samo, jak program `rungef`, jednak zamiast funkcji $f(x)$ interpolujące, odpowiednio, funkcje $g(x)$ i $h(x)$.

c) Przeanalizuj wyniki działania Twoich programów dla różnych wartości N, np. dla 7, 24, 21 i 28 węzłów (pamiętaj, że liczba węzłów to $N+1$). Jak zmienia się błąd interpolacji dla każdej z funkcji, gdy liczba węzłów rośnie?

Zadanie 3. Zachorowania na COVID-19.

Napisz program `covid`, wykreślający krzywą zachorowań na COVID–19 dla wybranej przez użytkownika lokalizacji. Program powinien pobierać z internetu plik z danymi dotyczącymi tygodniowej liczby zachorowań w formacie CSV, udostępniony pod adresem

`https://opendata.ecdc.europa.eu/covid19/nationalcasedeath/csv`

a następnie wypisywać listę lokalizacji, dla których dane są dostępne, prosić użytkownika o wskazanie jednej z nich i rysować krzywą przedstawiającą dzienną ilość nowych zachorowań na COVID–19 w tej lokalizacji.

W celu uzyskania informacji o dziennej liczbie zachorowań na podstawie dostępnych danych posłuż się interpolacją wielomianową, wykorzystując odpowiednie funkcje biblioteczne dostępne dla języka programowania albo pakietu obliczeń naukowych, z którego korzystasz.

Szkic rozwiązania w języku Python.

- Napisz kod pobierający plik z danymi i zapisujący go w bieżącym folderze. Możesz użyć do tego celu na przykład funkcji `get` z modułu `requests` (prościej, wymagana wcześniejsza instalacja pakietu `requests`) lub funkcji `urlopen` z modułu `urllib.request` (trudniej, jednak nie wymaga instalacji).
- Wczytaj plik z danymi, używając polecenia

  ```
  lines = open("data.csv").readlines()
  ```

  Jakiego typu jest obiekt `lines`? Możesz to sprawdzić korzystając z funkcji `type()`.
- Zapisz w zmiennej `data` trzeci element listy `lines`. Jakiego typu jest obiekt `data`?
- Podziel łańcuch `data` na fragmenty z przecinkiem jako separatorem, wywołując metodę `split()` z odpowiednimi argumentami. Zbadaj, jak wygląda wynik działania tej funkcji i wydobądź z niego informacje o tygodniu, kraju oraz ilości zachorowań. Wypisz te dane, by sprawdzić, czy kod działa poprawnie.
- Zmodyfikuj swój kod tak, by zamiast analizować pojedynczą linię pliku z danymi, badał wszystkie.

  Można w tym celu po prostu analizować kolejne elementy listy `lines` (np. za pomocą pętli `for`). Rozwiązanie takie byłoby poprawne, miałoby jednak istotną wadę. Polecenie `open("data.csv") .readlines()` wczytuje do pamięci od razu cały plik `data.csv` i tworzy ogromną listę `lines` zawierającą wszystkie jego linie. Proces ten w przypadku dużej ilości danych jest długotrwały i zużywa sporo pamięci operacyjnej; co więcej, gdyby plik z danymi był zbyt duży, pamięci mogłoby zabraknąć – zostałby wówczas zgłoszony wyjątek `MemoryError`, a program nie wykonałby swojego zadania.

  Bezpieczniej jest wczytywać plik z danymi linia po linii, program będzie wówczas działał poprawnie dla dowolnie dużej ilości danych. Można w tym celu posłużyć się na przykład następującym kodem (zamiast tworzyć listę `lines`):

  ```
  with open("data.csv") as datafile:
      for line in datafile:
          # Łańcuch line zawiera treść pojedynczej linii pliku.
  ```

  Zastosowanie polecenia `with` daje nam pewność, że po zakończeniu pracy z plikiem wszystkie związane z nim zasoby zostaną zwolnione (wcześniej nie przejmowaliśmy się tym problemem, co mogło skutkować pojawieniem się tzw. wycieku pamięci).

  Utwórz pusty słownik o nazwie `cases_pol`, a następnie przeanalizuj kolejne linie pliku z danymi, sprawdzając, czy dana linia dotyczy Polski, i – jeśli tak jest – dodaj do słownika `cases_pol` element, którego kluczem będzie rok i numer tygodnia (np. w postaci łańcucha znaków), zaś wartością – liczba zachorowań. Narysuj wykres danych zawartych w słowniku `cases_pol`, by sprawdzić, czy Twój kod działa poprawnie.

- Zmodyfikuj program tak, by analizował dane o zachorowaniach dla wybranej przez użytkownika lokalizacji.

  Najprostszym sposobem realizacji tego celu wydaje się być zastąpienie słownika `cases_pol` słownikiem `cases` mającym strukturę „słownika słowników": każdy jego klucz powinien być nazwą jednej z lokalizacji, dla których dane o zachorowaniach są dostępne, zaś wartość odpowiadająca danemu kluczowi – słownikiem zawierającym dane o zachorowaniach w lokalizacji, na którą wskazuje ten klucz, zbudowanym analogicznie, jak omówiony szczegółowo w poprzednim punkcie słownik `cases_pol`. Obiekt `cases["Italy"]` będzie wówczas słownikiem zawierającym dane o zachorowaniach we Włoszech, obiekt `cases["Poland"]` – słownikiem z danymi o zachorowaniach w Polsce (dokładnie tym samym, którym wcześniej był `cases_pol`) etc.

  Rozwiązanie to, choć poprawne, nie jest optymalne, w pamięci operacyjnej powstanie bowiem ogromna struktura danych, przechowująca wszystkie informacje zawarte w pliku z danymi. Gdy informacji tych będzie dużo, mogą wystąpić te same problemy, o których wspominaliśmy w poprzednim punkcie.

  Lepszym rozwiązaniem będzie przeprowadzenie analizy danych dwuetapowo. Na początku możemy wydobyć z pliku z danymi wyłącznie informację o tym, dla jakich lokalizacji dostępne są dane, następnie poprosić użytkownika o wskazanie interesującej go lokalizacji i ponownie przejrzeć plik, odczytując informacje o zachorowaniach wyłącznie dla tej lokalizacji.

  Kierując się tą ideą, zmodyfikuj kod programu w następujący sposób. Zamiast słownika `cases_pol` program powinien tworzyć pusty zbiór o nazwie `locations`, a potem analizować kolejne linie pliku z danymi, wydobywając z każdej z nich informację o tym, jakiej lokalizacji dotyczą zawarte w niej dane, i dodawać nazwę tej lokalizacji do zbioru `locations` (zastosowanie zbioru będzie tu sporym ułatwieniem – zbiór, w przeciwieństwie do np. listy, utożsamia ze sobą identyczne elementy, nie pozwalając na pojawienie się duplikatów). Program powinien następnie wypisać elementy zbioru `locations` i poprosić użytkownika o wybranie jednego z nich. Gdy użytkownik poda poprawną nazwę lokalizacji (np. `Poland`), program powinien utworzyć słownik o nazwie `cases` zawierający dane na temat zachorowań w tej lokalizacji (skonstruowany analogicznie, jak omówiony szczegółowo w poprzednim punkcie słownik `cases_pol`) i – na jego podstawie – narysować krzywą zachorowań dotyczącą tej lokalizacji.

- Na koniec zmodyfikuj program tak, by rysował dzienną krzywą zachorowań. W tym celu zagęść dane z pliku, posługując się interpolacją wielomianową. Teraz słownik `cases` powinien zawierać elementy, dla których kluczem będzie data (rok, miesiąc i dzień, np. w postaci łańcucha znaków), zaś wartością – liczba zachorowań danego dnia.

  Postaraj się tak skonfigurować wykres, by był on czytelny i estetyczny. W celu zapewnienia poprawnego wyświetlania etykiet na osi odciętych konieczne może się okazać przechowywanie dat w słowniku `cases` nie w postaci łańcuchów tekstowych, lecz za pomocą obiektów dedykowanego datom typu, np. typu `datetime` zdefiniowanego w module `datetime`.

Opracowanie: Bartłomiej Zglinicki.