Joanna Szałkowska
Michał Humiński
Bioinformatyka rok 3

Sprawozdanie z 3. części zajęć

Ostatnia część naszych zajęć dotyczyła problematyki algorytmów algebry liniowej. Zajmowaliśmy się obliczeniami przeprowadzanymi na macierzach oraz wektorach opartych o pakiet Numpy. Na ich podstawie staraliśmy się znaleźć uwarunkowania i rozwiązania układów równań nieliniowych. Poruszaliśmy tematykę: metody mnożenia macierzy, obliczania normy wektora, otrzymywania macierzy trójkątnych, rozwiązywania równań metodą eliminacji, problemów macierzy źle uwarunkowanych oraz aproksymacji metodą najmniejszych kwadratów.

Niniejsze sprawozdanie stanowi podsumowanie naszej pracy na ćwiczeniach. Chcemy w nim opisać przykładowe zagadnienie kosztu wykonania metody w zależności od podanych danych wejściowych. Do tego zadania wybraliśmy napisany przez nas kod dotyczący metody eliminacji. Zbiorami testowymi będą zestawy macierzy o różnych wymiarach.

Po otrzymaniu danych przystępujemy do ich analizy. Wykorzystujemy w niej metodę najmniejszych kwadratów celem wyznaczenia wielomianu, który w najlepszy sposób opisze badane zależności. Dobranie odpowiedniego wzoru zachodzi na podstawie porównania wyliczonych parametrów opisujących każdy z otrzymanych modeli.

Chcielibyśmy również zaznaczyć, że wszystkie wykonane badania i napisane skrypty, zostały przygotowanie i przeanalizowane wyłącznie przez nas.

Otrzymanie danych do analizy

Tak jak już zostało wspomniane we wstępie, analizowanym przez nas algorytmem jest metoda eliminacji. Polega ona na macierzowym rozwiązaniu układu równań, który to najpierw jest przekształcany do formy macierzy trójkątnej górnej. Modyfikując stopniowo elementy na przekątnej w kolejnych wierszach, zerujemy elementy macierzy, które znajdują się pod nią. Umożliwia to rozwiązanie otrzymanego układu w bardzo prosty sposób, wykonując odczytywanie i obliczanie szukanego wektora „od tyłu".

Kod - metoda eliminacji:

```
def metoda_eliminacji(A, B):
    wiersze_a, kolumny_a = np.shape(A)
    wiersze_b = np.shape(B)[0]
    if kolumny_a != wiersze_b:
        return(-1, None)
    kroki = 0
    wiersz = 0
    while wiersz < (wiersze_a-1):
        if A[wiersz][wiersz] == 0:
            return(-2, None)
        for wartosc_1 in range(wiersz+1, wiersze_a):
            czynnik = -(A[wartosc_1][wiersz] / A[wiersz][wiersz])
            kroki += 2
            for wartosc_2 in range(wiersz, kolumny_a):
                A[wartosc_1][wartosc_2] += czynnik * A[wiersz][wartosc_2]
                kroki += 2
            B[wartosc_1] += czynnik * B[wiersz]
            kroki += 2
        wiersz += 1
    X = np.zeros(shape=(kolumny_a,))
    licznik = wiersze_a - 1
    while licznik >= 0:
        suma = 0
        for wartosc_3 in range(licznik+1, kolumny_a):
            suma += A[licznik, wartosc_3] * X[wartosc_3]
            kroki += 2
        if A[licznik, licznik] == 0:
            return(-3, None)
        X[licznik] = (B[licznik] - suma) / A[licznik, licznik]
        kroki += 2
        licznik -= 1
    return(X, kroki)
```

Powyższa funkcja ma za zadanie wyznaczyć wektor X z układu równań: $A * X = B$, w którym to macierz A pomnożona przez wektor X daje macierz B. Parametrami wejściowymi są dwie macierze A (czynnik w równaniu) oraz B (wynik równania) o odpowiadających sobie wymiarach. Liczba kolumn w macierzy A musi równać się liczbie wierszy w macierzy B, aby można było wykonać mnożenie macierzowe. Jest to sprawdzane na samym początku kodu, kiedy to przy niespełnieniu naszego założenia zwracana jest informacja o niezgodności wymiarów w postaci: -1.

Następnie wykonujemy właściwe działania pozostawiając kopie naszych macierzy do sprawdzenia obliczeń i otrzymania wektora błędu. Zaczynamy od odpowiedniego przekształcenia macierzy A do macierzy trójkątnej, co zawarte jest w pierwszej pętli *while*. Najpierw badamy czy analizowany element na przekątnej nie jest równy 0. W przeciwnym przypadku kończymy funkcję zwracając wartość: -2. Problem ten zostanie poruszony na zakończenie opisu działania niniejszej funkcji.

Na podstawie wyznaczonego elementu na przekątnej staramy się wyzerować element leżący bezpośrednio pod nim (w wierszu poniżej). Znajdujemy odpowiednią liczbę (w kodzie pod nazwą *czynnik*), która po przemnożeniu przez wiersz z analizowanym elementem i dodaniu otrzymanej w ten sposób liczby do elementu da nam 0. Opisywane zachowanie wykonujemy analogicznie dla części wiersza na prawo względem analizowanej pozycji, mnożąc elementy w odpowiednich kolumnach przez wyliczony czynnik i dodając je do elementów poniżej nich. Następuje w ten sposób przekształcenie całego rzędu. Ponowne wyliczenia wykonujemy dla wszystkich linii poniżej badanego wiersza.

Po wyzerowaniu wartości w danej kolumnie, przechodzimy o jedną kolumnę do przodu i jeden wiersz poniżej, do kolejnego elementu na przekątnej. Powtarzamy opisany powyżej mechanizm do czasu dojścia do ostatniego elementu na przekątnej (elementu znajdującego się maksymalnie na prawym dole macierzy).

Należy pamiętać, że przy zerowaniu kolejnych elementów wierszy znajdujących się w jednej kolumnie, przemnożenie i dodanie określonych elementów dotyczy także macierzy wynikowej B - ona także ulega ewolucji.

Po przekształceniu początkowej macierzy do typu trójkątnego, jesteśmy gotowi na jej właściwe rozwiązanie, które jest zawarte w drugiej pętli *while*. Przed jej wykonaniem tworzymy jednak pusty wektor X o odpowiednich wymiarach - jego liczba wierszy musi odpowiadać liczbie kolumn w macierzy A, a liczby kolumn nie musimy definiować, ponieważ wynosi 1.

Zaczynamy obliczenia od ostatniego wiersza, w którym mamy tylko jeden element. Dzieląc odpowiednią wartość w wektorze B przez przez wybrany element w wektorze A otrzymujemy ostatnią wartość w wektorze X. Kiedy natrafimy na wartość elementu wynoszącą 0, ponownie przerywamy działanie funkcji i otrzymujemy wynik w postaci informacji: -3.

Dalej, w kolejnych wierszach, które zawierają więcej niewyzerowanych elementów, wyliczamy kolejne elementy w wektorze X. Sumujemy iloczyny odpowiadających sobie elementów z macierzy A i wektora X (w wierszach poniżej analizowanego), które to odejmujemy od wartości wektora B, a następnie dzielimy powstały wynik przez wartość analizowanego elementu - dostajemy w ten sposób kolejny czynnik wektora X.

Po przejściu od ostatniego do pierwszego wiersza macierzy, otrzymujemy wynik naszych obliczeń - wektor X jako typ *array*, który zawiera niewiadome w analizowanym przez nas układzie równań. Drugą zwracaną wartością jest liczba kroków, która jest sumowana podczas kolejnych obliczeń wykonywanych w naszym algorytmie. Przy opisanych

powyżej wynikach funkcji, które określają brak możliwości jej dalszego rozwiązania, zmienna *kroki* przyjmuje wartość $None$.

Należy zauważyć, że analizowany kod dotyczy podstawowej metody eliminacji, która nie dokonuje zamiany wierszy w przypadku natrafienia 0 na przekątnej lub też zamiany wierszy celem znalezienia największej możliwej wartości bezwzględnej dla wybranego elementu. Analiza którejkolwiek możliwości „rozwijającej" i usprawniającej nasz kod podczas dalszych analiz mogłaby dawać nam niewspółmierne dane. Przykładowo liczba kroków czy też czas działania funkcji w tych przypadkach zależałyby od określonego uporządkowania równań w naszym układzie, a sama zamiana wierszy końcowo nie zawsze dawałaby nam poprawniejsze rozwiązanie naszego układu.

Analizowane wielkości

Nasze badania skupiają się na 3 wielkościach, które możemy uzyskać z działania naszego kodu. Są to:

- czas obliczeń - czas działania naszej funkcji *metoda_eliminacji* dla wprowadzonych macierzy,
- ilość obliczeń - liczba naszych kroków (jako właściwych obliczeń), którą musimy wykonać, aby otrzymać wynik,
- wartość błędu wektora - błąd wektora wynikowego obliczany według podanej normy (różnica między rzeczywistą wartością a obliczoną poprzez mnożenie macierzy $A * X$).

Przygotowanie danych

Do naszych pomiarów przygotowujemy 11 zestawów danych. Są to zestawy macierzy A i B o odpowiednich wymiarach (opisanych już wcześniej). Zakładając wielkości dotyczące macierzy A, badamy macierze o wymiarach: 2x2, 3x3, 4x4, 5x5, 6x6, 7x7, 8x8, 9x9, 10x10, 20x20 oraz 50x50.

Dla każdego zestawu wykonujemy 1000 par losowo wygenerowanych macierzy A i B zawierających liczby z przedziału od 1 do 100, co zachodzi za pomocą funkcji: *np.random.randint()*. Należy pamiętać, że po właściwej nazwie komendy musimy dodać odpowiednie przypisanie typu: *astype("float")*, aby wyeliminować możliwy błąd w obliczeniach naszej funkcji.

Liczbę kroków w naszej funkcji otrzymujemy w jej wyniku i jest zawsze taka sama dla zestawu macierzy o tych samych wymiarach.

Czas obliczamy za pomocą funkcji *perf_counter()* z pakietu *Time*.

Wartość wektora błędu obliczamy za pomocą napisanej przez nas funkcji, która wylicza normę euklidesową:

```
def norma_euklidesowa(wektor):
    suma = 0.0
    for element in wektor:
        suma = suma + element**2
    return(suma**0.5)
```

Podczas każdej iteracji wyliczającej wynik z wygenerowanej pary macierzy interesujące nas wielkości dodajemy do przygotowanych wcześniej list. Na sam koniec (po 1000 powtórzeń naszej funkcji) wyliczamy średnie z list (za pomocą funkcji *mean* z pakietu *Statistics*), które są poszukiwanymi przez nas wartościami do dalszej analizy. Takich wartości otrzymujemy końcowo 11 dla każdej z analizowanych wielkości (tyle ile mamy zestawów danych).

Trzeba także wspomnieć, że w związku z niedoskonałością naszego kodu *metoda_eliminacji*, podczas wyliczeń musimy wziąć pod uwagę mogące zaistnieć problemy. Początkowy zakres liczb w macierzach A i B rozpoczyna się od 1, celem uniknięcia problematycznych wartości 0. Jednak podczas wykonywania naszego skryptu nadal możemy uzyskać wyniki -1, -2 lub -3 oznaczające brak możliwości wyliczenia wektora współczynników X. Takich przypadków nie bierzemy pod uwagę podczas uzyskiwania interesujących nas wielkości (czasu czy liczby kroków). Doświadczalnie okazało się, że stanowią one jedynie pojedyncze przypadki, których łączna liczba, względem liczby par macierzy wynoszącej 1000 dla każdego zestawu danych, jest znikoma.

Uzyskane dane

Na koniec uzyskujemy 4 listy z naszymi danymi, każda o długości wynoszącej 11 (równej liczbie zestawów danych):

- *x* (zmienne niezależne) - wielkości macierzy,
- *y1* (zmienne zależne 1) - czasy obliczeń,
- *y2* (zmienne zależne 2) - liczby kroków,
- *y3* (zmienne zależne 3) - wartości błędu wektorów.

Joanna Szałkowska
Michał Humiński
Bioinformatyka rok 3

Dane do analizy

Wartości badane/Wielkość macierzy	2	3	4	5	6	7	8	9	10	20	50
czas obliczeń	1.39195919999 99998e-05	2.7218426426 42595e-05	4.5849125125 124354e-05	7.0021245000 00017e-05	0.0001022366 6966966769	0.0001426919 7300000069	0.0001922962 1199999647	0.0002512722 442442443	0.0003219795 650000017	0.0017776352 862862875	0.0212734122 08625872
liczba kroków	14	40	84	150	242	364	520	714	950	6500	90750
wartości błędu	4.5268465248 73364e-15	1.9802719475 849116e-14	9.0816300431 34478e-14	3.2075746905 062784e-13	2.1732805487 24011e-13	4.1771021789 19595e-13	1.6176279954 795788e-12	6.7367498679 69288e-13	1.6645439661 20848e-12	1.0408019150 338877e-11	2.5069010654 414576e-10

Analiza otrzymanych danych

Kolejnym procesem, po uzyskaniu interesujących nas danych, jest ich analiza. Celem naszych działań jest uzyskanie wzorów matematycznych, będących wielomianami, które w najlepszy możliwy sposób będą opisywać badane przez nas zależności.

Dla każdego zestawu danych ukazanych na poprzedniej stronie prowadzimy obliczenia, dzięki czemu otrzymamy wielomiany różnego stopnia. Analizy będą przeprowadzane dla wielomianów w zakresie stopni 1-9. Dlatego wszystkie opisane poniżej kroki oraz funkcje będą wykonywane dziewięciokrotnie dla każdej analizowanej przez nas wielkości.

Otrzymanie współczynników równania szukanych przez nas funkcji, następuje poprzez rozwiązanie równania macierzowego metodą najmniejszych kwadratów. Analizowany przez nas zbiór punktów (x_i, y_i) staramy się przybliżyć najlepiej opisującą ich funkcją w postaci:

$f(x) = a_0 + a_1 x + \ldots + a_n x^n$

Stopień analizowanych wielomianów równy jest wartości najwyższej potęgi przy zmiennej x w powyższym wzorze.

Przygotowanie i rozwiązanie macierzowego układu równań

Najpierw w prawidłowy sposób formułujemy nasz układ równań. Na jego podstawie jesteśmy w stanie stworzyć równanie dotyczące mnożenia macierzy i wektora, w którym to wyznaczamy wektor X, zawierający współczynniki wielomianu aproksymacyjnego, ze znanego już nam wcześniej układu równań: $A * X = B$. Wykonujemy to za pomocą napisanej przez nas funkcji:

```
def metoda_najmniejszych_kwadratow(wartosci_x, wartosci_y, st_wielomianu):
    if len(wartosci_x) != len(wartosci_y):
        return(-1)
    st_macierzy = st_wielomianu + 1
    macierz = np.zeros(shape=(st_macierzy, st_macierzy))
    wektor = np.zeros(shape=(st_macierzy,))
    wspolczynniki = [1 for liczba in wartosci_x]
    macierz[0, 0] = sum(wspolczynniki)
    wyniki = [liczba for liczba in wartosci_y]
    wektor[0] = sum(wyniki)
    powtorzenie = 1
    koniec = 2*st_wielomianu
    while powtorzenie <= st_wielomianu:
        wspolczynniki = [liczba_1*liczba_2 for liczba_1,
                    liczba_2 in zip(wspolczynniki, wartosci_x)]
        suma_wspolczynnikow = sum(wspolczynniki)
        krok = powtorzenie
        wiersz = 0
        while krok >= 0:
            macierz[wiersz, krok] = suma_wspolczynnikow
            wiersz += 1
            krok -= 1
        wyniki = [liczba_1*liczba_2 for liczba_1,
                liczba_2 in zip(wyniki, wartosci_x)]
        wektor[powtorzenie] = sum(wyniki)
        powtorzenie += 1
    while powtorzenie <= koniec:
        wspolczynniki = [liczba_1*liczba_2 for liczba_1,
                    liczba_2 in zip(wspolczynniki, wartosci_x)]
        suma_wspolczynnikow = sum(wspolczynniki)
        krok = koniec - powtorzenie
        wiersz = st_wielomianu - krok
        kolumna = st_wielomianu
        while krok >= 0:
            macierz[wiersz, kolumna] = suma_wspolczynnikow
            wiersz += 1
            kolumna -= 1
            krok -= 1
        powtorzenie += 1
    return(macierz, wektor)
```

Przyjmuje ona 3 parametry, są to kolejno:

- *wartosci_x* - lista z wartościami zmiennej niezależnej (wielkości macierzy),
- *wartości_y* - lista z wartościami zmiennej zależnej (czasy obliczeń, liczby kroków lub wartości błędu wektorów),
- st_wielomianu - liczba, która będzie wyznaczać wielkość rozwiązywanych przez nas macierzy i wektorów, determinowana szukanym przez nas stopniem wielomianu (w naszym przypadku w zakresie 1-9).

Wynikiem powyżej funkcji jest macierz oraz wektor. Zgodnie z przedstawianym powyżej równaniem: $A * X = B$ są to macierz A oraz wektor B w formatach zaimplementowanych w pakiecie *Numpy*. Równanie jest gotowe, zatem musimy teraz znaleźć interesujący nas wektor współczynników X. Celem rozwiązania otrzymanego równania macierzowego, stosujemy znaną nam już wcześniej metodę eliminacji.

```
def metoda_eliminacji(A, B):
    wiersze_a, kolumny_a = np.shape(A)
    A1 = np.copy(A)
    B1 = np.copy(B)
    wiersz = 0
    while wiersz < (wiersze_a-1):
        for liczba in range(wiersz+1, wiersze_a):
            if abs(A1[liczba][wiersz]) > abs(A1[wiersz][wiersz]):
                A1[[wiersz, liczba]] = A1[[liczba, wiersz]]
                B1[wiersz], B1[liczba] = B1[liczba], B1[wiersz]
        for wartosc_1 in range(wiersz+1, wiersze_a):
            czynnik = -(A1[wartosc_1][wiersz] / A1[wiersz][wiersz])
            for wartosc_2 in range(wiersz, kolumny_a):
                A1[wartosc_1][wartosc_2] += czynnik * A1[wiersz][wartosc_2]
            B1[wartosc_1] += czynnik * B1[wiersz]
        wiersz += 1
    X = np.zeros(shape=(kolumny_a,))
    licznik = wiersze_a - 1
    while licznik >= 0:
        suma = 0
        for wartosc_3 in range(licznik+1, kolumny_a):
            suma += A1[licznik, wartosc_3] * X[wartosc_3]
        X[licznik] = (B1[licznik] - suma) / A1[licznik, licznik]
        licznik -= 1
    return(X)
```

Jako wejściowe parametry podajemy otrzymane w poprzednim etapie macierz oraz wektor wynikowy, dzięki czemu na koniec otrzymamy wektor zawierający interesujące nas elementy - współczynniki dla funkcji aproksymującej badaną zależność.

Należy również zauważyć, że nie jest to dokładna kopia analizowanej przez nas funkcji. Powyższy kod stanowi ulepszenie poprzedniego skryptu. Podczas stopniowego przekształcania równania do macierzy trójkątnej górnej, w każdym kroku staramy się, aby element, względem którego prowadzimy eliminację, miał możliwie największą wartość bezwzględną (biorąc pod uwagę wartości znajdujące się poniżej niego w kolumnie). Decyduje to zazwyczaj o wyższej dokładności i mniejszym błędzie końcowo otrzymanego wektora.

<u>Opracowanie i wybór wyników</u>

W trakcie wykonanych powyżej obliczeń dla każdej analizowanej zależności otrzymujemy 9 wektorów o różnych długościach (odpowiednio o długościach wynoszących wartość stopnia wielomianu powiększoną o 1). O wyborze najlepszego wzoru do aproksymacji

decydujemy na podstawie wyznaczonych wartości błędu. Dla każdego przypadku wyliczyliśmy zatem błąd sumaryczny dopasowania wszystkich obserwacji, błąd średni przypadający na jedną obserwację oraz najbardziej miarodajny wskaźnik - błąd przypadający na jeden „stopień swobody" dopasowania parametrów. Otrzymywaliśmy je na podstawie poniższych funkcji:

```
def blad_sumaryczny_dopasowania(war_oczekiwane, war_rzeczywiste):
    bledy = [(liczba_1-liczba_2)**2 for liczba_1,
            liczba_2 in zip(war_oczekiwane, war_rzeczywiste)]
    suma_bledow = sum(bledy)
    return(suma_bledow)

def blad_sredni_na_jedna_obserwacje(suma_bledow, lista):
    blad_na_obserwacje = suma_bledow/len(lista)
    return(blad_na_obserwacje)

def blad_na_jeden_stopien_swobody(suma_bledow, lista, stopien):
    blad_na_stopien = suma_bledow/(len(lista) - (stopien+1))
    return(blad_na_stopien)
```

Pomocne okazały się również dwie funkcje - przekształcająca otrzymany wektor współczynników w matematyczny wzór funkcji oraz obliczająca wartości zmiennej zależnej na podstawie opisywanego wektora.

```
def wzor_funkcji(wspolczynniki):
    pierwszy_skladnik = f"{wspolczynniki[0]}"
    wzor = "".join(pierwszy_skladnik)
    wykladnik = 1
    for element in wspolczynniki[1:]:
        kolejny_skladnik = " "
        if element > 0:
            kolejny_skladnik += "+"
        kolejny_skladnik += f"{element}*x"
        if wykladnik > 1:
            kolejny_skladnik += f"**{wykladnik}"
        wzor += kolejny_skladnik
        wykladnik += 1
    return(wzor)

def obliczenie_wartosci(x, wspolczynniki):
    wyniki = []
    for wartosc in x:
        wykladnik = 0
        wynik_czastkowy = 0
        for element in wspolczynniki:
            wynik_czastkowy += element*(wartosc**wykladnik)
            wykladnik += 1
        wyniki.append(wynik_czastkowy)
    return(wyniki)
```

Wydaje się jednak, że drugim najważniejszym parametrem jest współczynnik determinacji naszej funkcji wielomianowej. Określa nam w jakim stopniu zmienna zależna objaśniana jest za pomocą wyznaczonego wzoru. Obliczamy go za pomocą napisanej przez nas funkcji:

```
def wspolczynnik_determinacji(war_rzeczywiste, war_oczekiwane):
    srednia = mean(war_rzeczywiste)
    licznik = 0
    mianownik = 0
    for liczba_1, liczba_2 in zip(war_rzeczywiste, war_oczekiwane):
        licznik += (liczba_1 - liczba_2)**2
        mianownik += (liczba_1 - srednia)**2
    wspolczynnik = 1 - (licznik/mianownik)
    return(wspolczynnik)
```

Końcowo otrzymujemy wypis dla każdego stopnia wielomianu z obliczonymi współczynnikami, wzorem funkcji aproksymującej, wartościami błędów i współczynnika determinacji. Przykładowy jego fragment wygląda w następujący sposób:

```
...
==========
Stopien wielomianu: 6
Wspolczynniki: [-3.33813948e-06  7.12966029e-06  1.51792071e-07  3.19714324e-07
 -1.05552269e-08  2.69859429e-10 -2.41820202e-12]
Wzor: -3.3381394785371096e-06 +7.1296602921522915e-06*x
+1.5179207142435303e-07*x**2 +3.197143235787508e-07*x**3
-1.055522692410383e-08*x**4 +2.6985942888026404e-10*x**5
-2.418202018684911e-12*x**6
Blad sumaryczny dopasowania: 2.8840413612143417e-13
Blad sredni na jedna obserwacje: 2.6218557829221287e-14
Blad na jeden stopien swobody: 7.210103403035854e-14
Wspolczynnik determinacji: 0.9999999992837154
==========
...
```

Porównując wypisane wyniki dla wszystkich badanych stopni wielomianu, możemy określić, który z nich najlepiej aproksymuje analizowaną przez nas zależność.

Na sam koniec korzystając z pakietu *Matplotlib*, rysujemy wykres dla wszystkich określonych wcześniej funkcji wielomianowych. W niektórych przypadkach odpowiedni zakres osi X i Y musimy wyregulować za pomocą funkcji *plt.xlim* oraz *plt.ylim*.

```
x = np.linspace(0, 10, 1000000)
stopien_funkcji = 1
for element in czas_wzor_funkcji:
    plt.plot(x, wyrazenie_funkcji(element),
             label=f"stopien funkcji: {stopien_funkcji}")
    stopien_funkcji += 1
plt.legend()
plt.show()
```

Do powyższej funkcji pobieramy gotowe wyrażenie ze wzorem, który wcześniej zapisany był w postaci tekstowej. Przed wyrysowaniem wykresu przekształcamy zatem wzór do wyrażenia matematycznego używając kodu:

```
def wyrazenie_funkcji(wzor):
    funkcja = eval(wzor)
    return(funkcja)
```

Wyniki analizy działania metody eliminacji

W ostatnim rozdziale naszego sprawozdania postaramy się wybrać interesujące nas funkcje aproksymujące badane zależności.

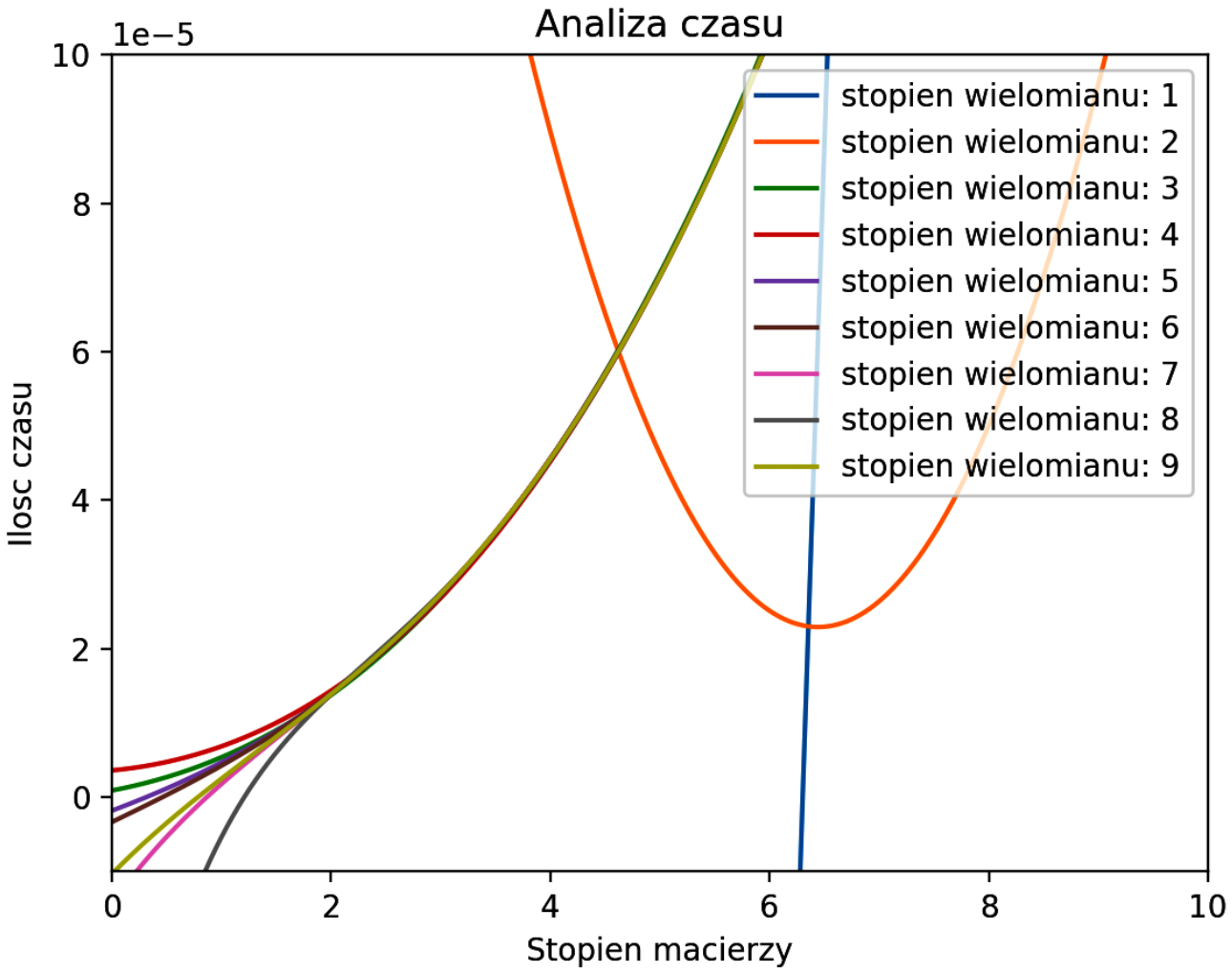
Analiza czasu obliczeń

Współczynnik determinacji dla wszystkich otrzymanych wyników był wysoki. Stopnie wielomianów, przy których praktycznie nie obserwowaliśmy jego istotnej zmiany, mieściły się w zakresie 3-9. W takim razie niezbędne było przeanalizowanie błędu przypadającego na jeden stopień swobody. Po porównaniu wszystkich wyników, najmniejszą wartość wspomnianego błędu miała funkcja wielomianowa stopnia 5. Należy jednak wspomnieć, że w rzeczywistości wszystkimi wielomianami stopni 4-7 jesteśmy w stanie aproksymować otrzymaną zależność niemalże z identyczną dokładnością - różnice w analizowanym błędzie dotyczą 14. miejsca po przecinku (większego o 10 pozycji względem średniego miejsca po przecinku dla otrzymanych współczynników wielomianu).

Wydruk dla najlepszego wyniku - **wielomianu stopnia 5.**:

Stopien wielomianu: 5
Wspolczynniki: [-1.80908211e-06 5.58206636e-06 7.03278597e-07 2.30722079e-07
-3.75997180e-09 4.44719533e-11]
*Wzor: -1.8090821061396692e-06 +5.582066355007854e-06*x*
*+7.032785965258617e-07*x**2 +2.3072207895452983e-07*x**3*
*-3.759971804401305e-09*x**4 +4.447195325048847e-11*x**5*
Blad sumaryczny dopasowania: 3.007511473490913e-13
Blad sredni na jedna obserwacje: 2.7341013395371934e-14
Blad na jeden stopien swobody: 6.015022946981826e-14
Wspolczynnik determinacji: 0.9999999992530502

Otrzymany wykres zależności ilości czasu (w sekundach) do stopnia obliczanej macierzy:



Zauważenie jakichkolwiek różnic w przebiegu analizowanych funkcji wymaga odpowiedniego wyskalowania wykresu. Widzimy, że funkcje wielomianowe wyższego stopnia bardzo szybko schodzą się do jednej krzywej.

Na podstawie powyższych wyników jesteśmy w stanie określić funkcję aproksymującą zależność ilości czasu do stopnia obliczanej macierzy metodą eliminacji:

$$f(x) = -1{,}8090821061396692 * 10^{-6} + (5{,}582066355007854 * 10^{-6})x +$$
$$+(7{,}032785965258617 * 10^{-7})x^2 + (2{,}3072207895452983 * 10^{-7})x^3 -$$
$$-(3{,}759971804401305 * 10^{-9})x^4 + (4{,}447195325048847 * 10^{-11})x^5$$

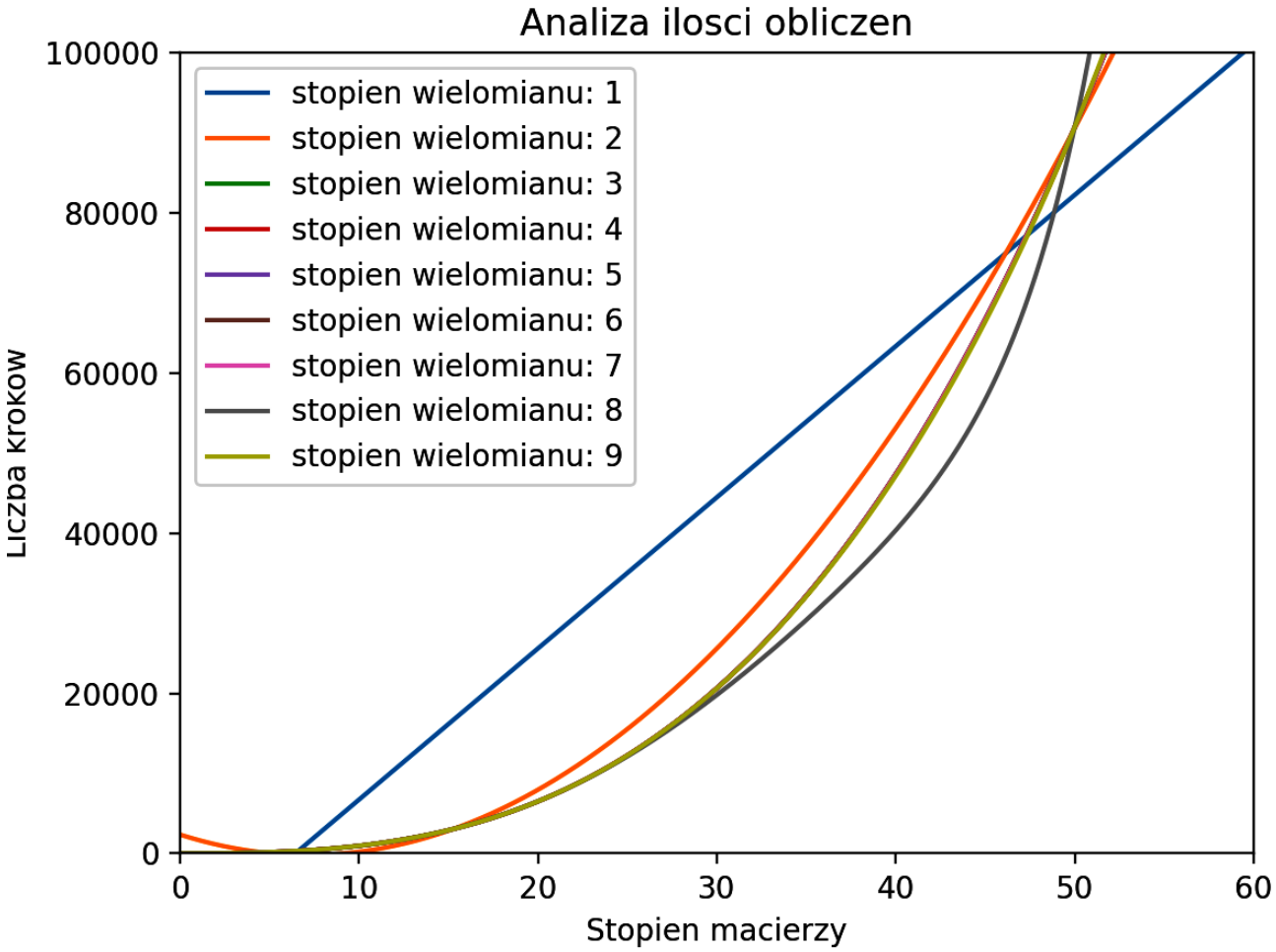
Analiza liczby wykonanych kroków

W opisywanym punkcie warto również wspomnieć, że analizowany przez nas „krok" nie odpowiada jednej operacji. W rzeczywistości nie wiemy co kryje się pod nawet tak błahymi działaniami jak dodawanie czy odejmowanie. Zatem za nasz „krok" przyjęliśmy wykonanie jednego działania matematycznego, niezależnie od jego stopnia skomplikowania.

Ponownie obserwowaliśmy wysokie wartości współczynnika determinacji dla wszystkich stopni wielomianu. Dalej analizować będziemy jednak tylko stopnie 3-7 oraz 9, ponieważ w tych przypadkach wynosił on dokładnie 1. Tutaj obserwujemy jednak znaczne zróżnicowanie dotyczące błędu na jeden stopień swobody. Ich różnica dotyczy aż 12 miejsc znaczących po przecinku. W tym przypadku dobieramy wielomian stopnia 3., którego wartość wspomnianego błędu dochodzi do rzędu wielkości 10^{-20}.

Wydruk dla najlepszego wyniku - **wielomianu stopnia 3.**:
Stopien wielomianu: 3
Wspolczynniki: [3.08487252e-10 -1.66666667e+00 3.00000000e+00 6.66666667e-01]
*Wzor: 3.0848725167525585e-10 -1.666666666766172*x +3.0000000000064615*x**2 +0.6666666666665747*x**3*
Blad sumaryczny dopasowania: 1.1787202853932948e-19
Blad sredni na jedna obserwacje: 1.0715638958120862e-20
Blad na jeden stopien swobody: 1.6838861219904213e-20
Wspolczynnik determinacji: 1.0

Otrzymany wykres zależności liczby kroków do stopnia obliczanej macierzy:



Tym razem tendencja zbiegania funkcji wielomianowych wyższych stopni do jednej krzywej nie występuje w każdym przypadku.

Na podstawie powyższych wyników jesteśmy w stanie określić funkcję aproksymującą zależność liczby wykonanych kroków do stopnia obliczanej macierzy metodą eliminacji:

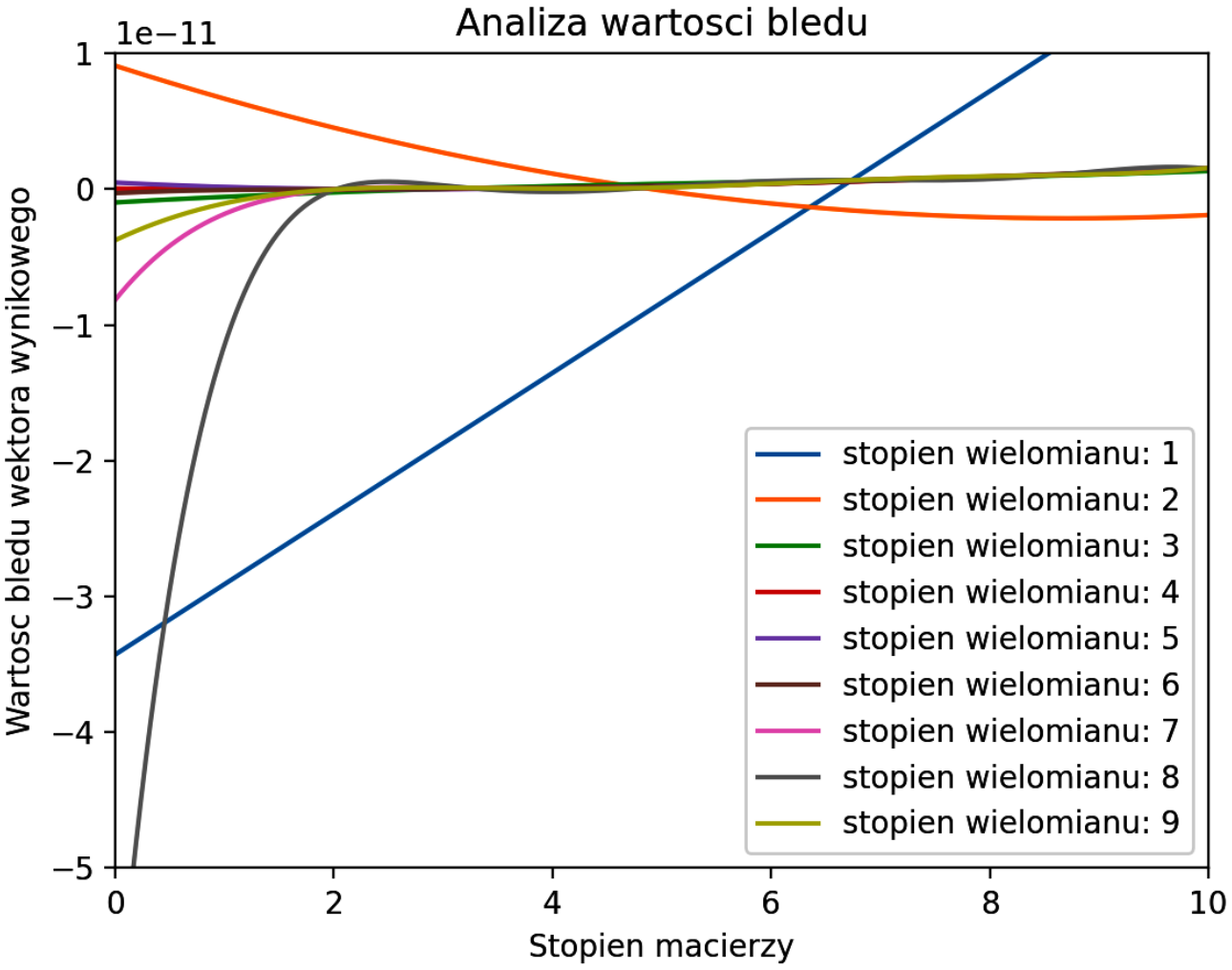
$$f(x) = 3{,}0848725167525585 * 10^{-10} - 1{,}6666666666766172x + \\ +3{,}0000000000064615x^2 + 0{,}6666666666665747x^3$$

Analiza wartości błędu wektora wynikowego

W ostatniej analizie ponownie za pożądane możemy uznać wartości współczynnika determinacji dla stopni wielomianów w zakresie 3-9. Tutaj różnice dotyczą piątego (stutysięcznego) miejsca po przecinku. Wybór najlepszej funkcji dokonujemy zatem znajdując najniższą wartość błędu przypadającego na jeden stopień swobody. Tak jak w poprzedniej analizie wybieramy wielomian stopnia 3. Błędy ponownie są niezwykle niskie, ponieważ mają rząd wielkości 10^{-25}, przez co rozbieżności poszczególnych wyników wydają się być praktycznie nieznaczące.

Wydruk dla najlepszego wyniku - **wielomianu stopnia 3.**:
Stopien wielomianu: 3
Wspolczynniki: [-9.77780409e-13 4.63560652e-13 -5.18880055e-14 2.86566480e-15]
*Wzor: -9.777804085173244e-13 +4.635606515742584e-13*x*
*-5.188800552364083e-14*x**2 +2.8656647951041573e-15*x**3*
Blad sumaryczny dopasowania: 1.1067172916118403e-24
Blad sredni na jedna obserwacje: 1.0061066287380367e-25
Blad na jeden stopien swobody: 1.581024702302629e-25
Wspolczynnik determinacji: 0.9999804195774528

Otrzymany wykres zależności błędu wektora wynikowego do stopnia obliczanej macierzy:



Ponownie obserwujemy tendencję zbiegania funkcji wielomianowych wyższych stopni do jednej krzywej.

Na podstawie powyższych wyników jesteśmy w stanie określić funkcję aproksymującą zależność wartości błędu wektora wynikowego (obliczanego jako norma euklidesowa) do stopnia obliczanej macierzy metodą eliminacji:

$$f(x) = -9{,}777804085173244 * 10^{-13} + (4{,}635606515742584 * 10^{-13})x -$$
$$-(5{,}188800552364083 * 10^{-14})x^2 + (2{,}8656647951041573 * 10^{-15})x^3$$