Artur Gęsiarz,
Maj 11, 2024

Laboratorium nr 8
MOwNiT – Rozwiązywanie równań nieliniowych

1. Treść zadania

1.1. Zadanie pierwsze

Dla poniższych funkcji i punktów początkowych metoda Newtona zawodzi. Znajde pierwiastki, modyfikujące wywołanie funkcji scipy.optimize.newton lub używając innej metody.

(a) $f(x) = x^3 - 5x, x0 = 1$
(b) $f(x) = x^3 - 3x + 1, x0 = 1$
(c) $f(x) = 2 - x^5, x0 = 0.01$
(d) $f(x) = x^4 - 4.29 * x^2 - 5.29, x0 = 0.8$

1.2. Zadnie drugie

Dane jest równanie: $f(x) = x^2 - 3x + 2 = 0$.

Każda z następujących funkcji definiuje równoważny schemat iteracyjny:

$$g_1(x) = \frac{x^2 + 2}{3},$$
$$g_2(x) = \sqrt{3x - 2},$$
$$g_3(x) = 3 - \frac{2}{x},$$
$$g_4(x) = \frac{x^2 - 2}{2x - 3},$$

Przeanalizuje zbieżność oraz rząd zbieżności schematów iteracyjnych odpowiadających funkcjom $g_i(x)$ dla pierwiastka $x = 2$ badając wartość $|g_i'(2)|$.

Potwierdze analizę teoretyczną implementując powyższe schematy iteracyjne i weryfikując ich zbieżność (lub brak). Każdy schemat iteracyjny wykonaj przez 10 iteracji.

Wyznacz eksperymentalnie rząd zbieżności każdej metody iteracyjnej ze wzoru

$$r = \frac{\ln\left(\frac{e_k}{e_{k+1}}\right)}{\ln\left(\frac{e_{k-1}}{e_k}\right)}$$

gdzie błąd bezwzględny ε_k definiujemy jako $\varepsilon_k = |x_k - x^*|$, xk jest przybli- żeniem pierwiastka w k-tej iteracji, a x^* dokładnym położeniem pierwiastka równania.

Na wspólnym rysynku przedstaw wykresy błędu względnego każdej metody w zależności od numeru iteracji. Użyj skali logarytmicznej na osi y (pomocna będzie funkcja semilogy).
Stwórz drugi rysunek, przedstawiający wykresy błędu względnego tylko dla metod zbieżnych.

1.3. Zadanie trzecie

Napisz schemat iteracji wg metody Newtona dla każdego z następujących równań nieliniowych:

$$x^3 - 2x - 5 = 0$$
$$e^{-x} - x = 0$$
$$x * \sin(x) - 1 = 0$$

Jeśli x0 jest przybliżeniem pierwiastka z dokładnością 4 bitów, ile iteracji należy wykonać aby osiągnąć: 24-bitową dokładność, 53-bitową dokładność?

1.4. Zadanie czwarte

Napisz schemat iteracji wg metody Newtona dla następującego układu równań nieliniowych:

$$x_1^2 + x_2^2 = 1$$
$$x_1^2 - x_2 = 0$$

Korzystając z faktu, że dokładne rozwiązanie powyższego układu równań to:

$$x_1 = \sqrt{\frac{\sqrt{5}}{2} - \frac{1}{2}}$$
$$x_2 = \frac{\sqrt{5}}{2} - \frac{1}{2}$$

oblicz błąd względny rozwiązania znalezionego metodą Newtona.

2. Rozwiązanie zadań

2.1. Implementacja zadania pierwszego

2.1.1. Implementacja funkcji podanych w zadaniu oraz ich pochodnych

```python
def function_a(x):
    return x ** 3 - 5 * x

def function_a_prim(x):
    return 3 * x ** 2 - 5

def function_b(x):
    return x ** 3 - 3 * x + 1

def function_b_prim(x):
    return 3 * x ** 2  - 3

def function_c(x):
    return 2 - x ** 5

def function_c_prim(x):
    return - 5 * x ** 4

def function_d(x):
    return x ** 4 - 4.29 * x ** 2 - 5.29

def function_d_prim(x):
    return 4 * x ** 3 - 8.58 * x
```

2.1.2. Prawdziwe pierwiastki równania

```python
exact_values_a = [-2.24, 0, 2.24]
exact_values_b = [-1.88, 0.35, 1.53]
exact_values_c = [1.15]
exact_values_d = [-2.3, 2.3]
```

2.1.3. Początkowe argumenty

```python
x0_a = 1
x0_b = 1
x0_c = 0.01
x0_d = 0.8
```

2.1.4. Rozpatrzenie podpunktu a

```python
solutions_a = []
start_points = [1.001, 1.1, 0.5]

for start_point in start_points:
    solutions_a.append(newton(function_a,start_point,function_a_prim))

solutions_a.sort()

print("Pierwiastki jakie znalazlem: ", solutions_a)
print("Orginalne pierwiastki: ", exact_values_a)
```

2.1.5. Rozpatrzenie podpunktu b

```python
solutions_b = [brentq(function_b, -0.5, 0.5), brentq(function_b, -2, -1.5), brentq(function_b, 1, 2)]
solutions_b.sort()

print("Pierwiastki jakie znalazlem: ", solutions_b)
print("Orginalne pierwiastki: ", exact_values_b)
```

2.1.6. Rozpatrzenie podpunktu c

```python
solutions_c = [newton(function_c, 0.01, function_c_prim,maxiter=100)]

print("Pierwiastki jakie znalazlem: ", solutions_c)
print("Orginalne pierwiastki: ", exact_values_c)
```

2.1.7. Rozpatrzenie podpunktu d

```
solutions_d = [newton(function_d, 1.2, function_d_prim), np.abs(newton(function_d, 1.2, function_d_prim))]

print("Pierwiastki jakie znalazlem: ", solutions_d)
print("Orginalne pierwiastki: ", exact_values_d)
```

2.1.8. Policzenie błędu względnego

```
def calculate_error(errors, solutions,exact_solutions):
    for i in range(len(solutions)):
        error = np.abs(solutions[i] - exact_solutions[i])
        errors.append(error)
```

```
errors_a = []
errors_b = []
errors_c = []
errors_d = []

calculate_error(errors_a, solutions_a, exact_values_a)
calculate_error(errors_b, solutions_b, exact_values_b)
calculate_error(errors_c, solutions_c, exact_values_c)
calculate_error(errors_d, solutions_d, exact_values_d)

print("Sredni blad bezwzgledny liczac miejsca zerowe funkcji A wynosi: \n",np.mean(errors_a ))
print("Sredni blad bezwzgledny liczac miejsca zerowe funkcji B wynosi: \n",np.mean(errors_b))
print("Sredni blad bezwzgledny liczac miejsca zerowe funkcji C wynosi: \n",np.mean(errors_c))
print("Sredni blad bezwzgledny liczac miejsca zerowe funkcji D wynosi: \n",np.mean(errors_d))
```

2.2. Implementacja zadania drugiego

2.2.1. Dane równanie

```
def f(x):
    return x ** 2 - 3 * x + 2
```

2.2.2. Schematy iteracyjne

```
def g1(x):
    return (x ** 2 + 2) / 3

def g2(x):
    return np.sqrt(3 * x - 2)

def g3(x):
    return 3 - 2 / x

def g4(x):
    return (x ** 2 - 2) / (2 * x - 3) if 2 * x - 3 != 0 else 1000000
```

2.2.3. Pochodne schematów iteracyjnych

```
def g1_prime(x):
    return 2 * x / 3

def g2_prime(x):
    return 3 / (2 * np.sqrt(3 * x - 2))

def g3_prime(x):
    return 2 / x ** 2

def g4_prime(x):
    return ( 2 * x * (2 * x - 3) - (x ** 2 - 2) * 2 ) / (2 * x - 3) ** 2
```

2.2.4. Badanie zbieżności oraz rzędu schematu iteracyjnego odpowiadających funkcji $g_i(x)$ dla pierwiastka $x = 2$ badajac $g_i'(2)$

```
x_value = 2

derivatives = {
    'g1': np.abs(g1_prime(x_value)),
    'g2': np.abs(g2_prime(x_value)),
    'g3': np.abs(g3_prime(x_value)),
    'g4': np.abs(g4_prime(x_value))
}
```

2.2.5. Analiza zbieżności oraz rzędu zbieżności danych schematów iteracyjnych

```
for name, der in derivatives.items():
    if der < 1:
        print(f"Funkcja {name}(x) jest zbieżna.")
        print(f"Rząd zbieżności dla funkcji {name}(x): 1")
    else:
        print(f"Funkcja {name}(x) jest rozbieżna.")
    print("\n")
```

2.2.6. Potwierdzenie analizy teoretycznej – weryfikując ich zbieżność lub brak

```
def iterate(g,x_0, n = 10):
    x = x_0
    errors = []

    for i in range(n):
        x = g(x)
        error = np.abs(x - 2)
        errors.append(error)
        print(f"Iteracja {i+1}: x = {x}, błąd = {error}")

    return errors
```

```
# Początkowa wartosc x
x0 = 1.5

# Wykonanie iteracji dla każdego schematu iteracyjnego
print("Schemat iteracyjny g1(x):")
errors_g1 = iterate(g1, x0)

print("\nSchemat iteracyjny g2(x):")
errors_g2 = iterate(g2, x0)

print("\nSchemat iteracyjny g3(x):")
errors_g3 = iterate(g3, x0)

print("\nSchemat iteracyjny g4(x):")
errors_g4 = iterate(g4, x0)
```

2.2.7. Funkcja do obliczenia rzędu zbieżności

```
def calculate_convergence_order(errors):
    orders = []
    for k in range(1, len(errors) - 1):
        order = np.log(errors[k] / errors[k+1]) / np.log(errors[k-1] / errors[k])
        orders.append(order)
    return orders
```

2.2.8. Wyznaczenie zbieżności

```
orders_g1 = calculate_convergence_order(errors_g1)
orders_g2 = calculate_convergence_order(errors_g2)
orders_g3 = calculate_convergence_order(errors_g3)
orders_g4 = calculate_convergence_order(errors_g4)
```

```
print("\nRzędy zbieżności:")
print(f"g1(x): {orders_g1}. \n")
print(f"g2(x): {orders_g2}. \n")
print(f"g3(x): {orders_g3}. \n")
print(f"g4(x): {orders_g4}. \n")
```

2.2.9. Wykres błędu względnego dla każdej metody w zależności od numeru iteracji

```
iterations = 10

plt.figure(figsize=(10, 5))
plt.semilogy(range(1, iterations+1), errors_g1, marker='o', label='g1(x)')
plt.semilogy(range(1, iterations+1), errors_g2, marker='o', label='g2(x)')
plt.semilogy(range(1, iterations+1), errors_g3, marker='o', label='g3(x)')
plt.semilogy(range(1, iterations+1), errors_g4, marker='o', label='g4(x)')
plt.title('Wykresy błędu względnego dla każdej metody')
plt.xlabel('Numer iteracji')
plt.ylabel('Błąd względny')
plt.yscale('log')
plt.legend()
plt.grid(True)
plt.show()
```

2.2.10. Wykres błędu względnego, ale tylko dla metod zbieżnych

```
plt.figure(figsize=(10, 5))
plt.semilogy(range(1, iterations+1), errors_g2, marker='o', label='g2(x)')
plt.semilogy(range(1, iterations+1), errors_g3, marker='o', label='g3(x)')
plt.semilogy(range(1, iterations+1), errors_g4, marker='o', label='g4(x)')
plt.title('Wykresy błędu względnego dla metod zbieznych')
plt.xlabel('Numer iteracji')
plt.ylabel('Błąd względny')
plt.yscale('log')
plt.legend()
plt.grid(True)
plt.show()
```

2.3. Implementacja zadania trzeciego

2.3.1. Równanie nieliniowe oraz ich pochodne

```python
def function_a(x):
    return x ** 3 - 2 * x - 5

def function_a_prim(x):
    return 3 * x ** 2 - 2

def function_b(x):
    return np.exp(-x) - x

def function_b_prim(x):
    return -np.exp(-x) - 1

def function_c(x):
    return x * np.sin(x) - 1

def function_c_prim(x):
    return np.sin(x) + x * np.cos(x)
```

2.3.2. Funkcja iteracji metoda Newtona

```python
def newton_iteration(f, f_prime, x0, tolerance, max_iteration=100000):
    x_k = x0
    iterations = 0
    while iterations < max_iteration:
        x_next = x_k - f(x_k) / f_prime(x_k)
        iterations += 1
        if abs(x_next - x_k) <= tolerance and np.abs(f(x_k)) <= tolerance:
            break
        x_k = x_next
    return x_next, iterations
```

2.3.3. Dane dokładności

```python
tolerance_24_bit = 1 / 2 ** 24
tolerance_53_bit = 1 / 2 ** 53
x_0 = 1.0
```

2.3.4. Obliczenie liczby iteracji dla każdego równania i dokładności

```python
result_a_24_bit, iterations_a_24_bit = newton_iteration(function_a, function_a_prim, x_0, tolerance_24_bit)
result_b_24_bit, iterations_b_24_bit = newton_iteration(function_b, function_b_prim, x_0, tolerance_24_bit)
result_c_24_bit, iterations_c_24_bit = newton_iteration(function_c, function_c_prim, x_0, tolerance_24_bit)

result_a_53_bit, iterations_a_53_bit = newton_iteration(function_a, function_a_prim, x_0, tolerance_53_bit)
result_b_53_bit, iterations_b_53_bit = newton_iteration(function_b, function_b_prim, x_0, tolerance_53_bit)
result_c_53_bit, iterations_c_53_bit = newton_iteration(function_c, function_c_prim, x_0, tolerance_53_bit)
```

2.3.5. Wyświetlanie wyników

```python
print("24-bitowa dokładność:")
print("Dla równania (a): Liczba iteracji =", iterations_a_24_bit, ", Wynik =", result_a_24_bit)
print("Dla równania (b): Liczba iteracji =", iterations_b_24_bit, ", Wynik =", result_b_24_bit)
print("Dla równania (c): Liczba iteracji =", iterations_c_24_bit, ", Wynik =", result_c_24_bit)

print("\n53-bitowa dokładność:")
print("Dla równania (a): Liczba iteracji =", iterations_a_53_bit, ", Wynik =", result_a_53_bit)
print("Dla równania (b): Liczba iteracji =", iterations_b_53_bit, ", Wynik =", result_b_53_bit)
print("Dla równania (c): Liczba iteracji =", iterations_c_53_bit, ", Wynik =", result_c_53_bit)
```

2.4. Implementacja zadania czwartego

2.4.1. Implementacja funkcji

```
def function_1(x_1,x_2):
    return x_1 ** 2 + x_2 ** 2 - 1

def function_2(x_1,x_2):
    return x_1 ** 2 - x_2

def jacobian(x_1, x_2):
    return np.array([[2 * x_1, 2 * x_2],
                     [2 * x_1, -1]])

def newton_system(f1, f2, jacobian, x0, tolerance=1e-8, max_iterations=1000):
    x = np.array(x0, dtype=float)

    for _ in range(max_iterations):
        f = np.array([f1(x[0], x[1]), f2(x[0], x[1])])
        J = jacobian(x[0], x[1])
        delta_x = np.linalg.solve(J, -f)
        x += delta_x
        if np.linalg.norm(delta_x) < tolerance:
            return x, _

    raise ValueError("Metoda Newtona nie zbiega się")
```

2.4.2. Dokładnie rozwiązanie

```
exact_solution_1 = np.sqrt(np.sqrt(5)/2 - 0.5)
exact_solution_2 = np.sqrt(5)/2 - 0.5
```

2.4.3. Obliczenia

```
x0 = [1.0, 1.0]
solution, iterations = newton_system(function_1, function_2, jacobian, x0)
```

2.4.4. Obliczenie błędu względnego

```
relative_error_1 = abs(solution[0] - exact_solution_1) / abs(exact_solution_1)
relative_error_2 = abs(solution[1] - exact_solution_2) / abs(exact_solution_2)
```

2.4.5. Wypisanie rozwiązanie oraz liczbę iteracji

```
print("Rozwiązanie z metody Newtona: ", solution)
print("Prawdziwe rozwiazania: ",exact_solution_1, exact_solution_2)
print("\n")
print("Liczba iteracji:", iterations)
print("Błąd względny x1:", relative_error_1)
print("Błąd względny x2:", relative_error_2)
```

3. Tabele

3.1. Tabela przybliżeń pierwiastków równania metodą Newtona dla f_1

Numer pierwiastka / Metoda	Prawdziwe	Obliczone
1	-2.24	-2.236
2	0	0
3	2.24	2.236

Tabela 1. Tabela przybliżeń pierwiastków metoda Newtona dla funkcji pierwszej w zadaniu pierwszym

3.2. Tabela przybliżeń pierwiastków równania metodą Newtona dla f_2

Numer pierwiastka / Metoda	Prawdziwe	Obliczone
1	-1.88	-1.879
2	0.35	0.34
3	1.53	1.532

Tabela 2. Tabela przybliżeń pierwiastków metoda Newtona dla funkcji drugiej w zadaniu pierwszym

3.3. Tabela przybliżeń pierwiastków równania metodą Newtona dla f_3

Numer pierwiastka / Metoda	Prawdziwe	Obliczone
1	1.14	1.15

Tabela 3. Tabela przybliżeń pierwiastków metoda Newtona dla funkcji trzeciej w zadaniu pierwszym

3.4. Tabela przybliżeń pierwiastków równania metodą Newtona dla f_4

Numer pierwiastka / Metoda	Prawdziwe	Obliczone
1	-2.3	-2.3
2	2.3	2.3

Tabela 4. Tabela przybliżeń pierwiastków metoda Newtona dla funkcji czwartej w zadaniu pierwszym

3.5. Tabela średniego błędu bezwzględnego licząc miejsca zerowe funkcji

Funkcja f_i	Średni błąd bezwzględny licząc miejsce zerowe funkcji
1	$0{,}002$
2	$0{,}0018$
3	$0{,}001$
4	0

Tabela 5. Tabela średniego błędu bezwzględnego dla kolejnych funkcji w zadaniu pierwszym

3.6. Tabela analizy zbieżności oraz rzędu zbieżności danych schematów iteracyjnych

Funkcja g_i	Czy zbieżna	Rząd zbieżności
1	NIE	–
2	TAK	1
3	TAK	1
4	TAK	1

Tabela 6. Tabela analizy zbieżności oraz rzędu zbieżności dla kolejnych schematów iteracyjnych w zadaniu drugim

3.7. Tabela analizy iteracji danej dokładności dla danych schematów iteracyjnych przy 24-bitowej dokładności

Równanie	Liczba iteracji	Wynik przybliżonego argumentu	Prawdziwy wynik
a	9	2.0945514815423265	2.0945514815423
b	4	0.5671432904097838	0.5671432904088
c	3	1.11415714087193	1.1141571411104

Tabela 7. Tabela analizy iteracji danej dokładności dla danych schematów iteracyjnych przy 24-bitowej dokładności w zadaniu trzecim

3.8. Tabela analizy iteracji danej dokładności dla danych schematów iteracyjnych przy 53-bitowej dokładności

Równanie	Liczba iteracji	Wynik przybliżonego argumentu	Prawdziwy wynik
a	MAX^*	2.0945514815423265	2.0945514815423
b	5	0.5671432904097838	0.5671432904088
c	MAX^*	1.1141571408719302	1.1141571411104

Tabela 8. Tabela analizy iteracji danej dokładności dla danych schematów iteracyjnych przy 53-bitowej dokładności w zadaniu trzecim

* - Osiągnięto maksymalną liczbę iteracji w pętli (przyjęto za maksymalna 100000 operacji)

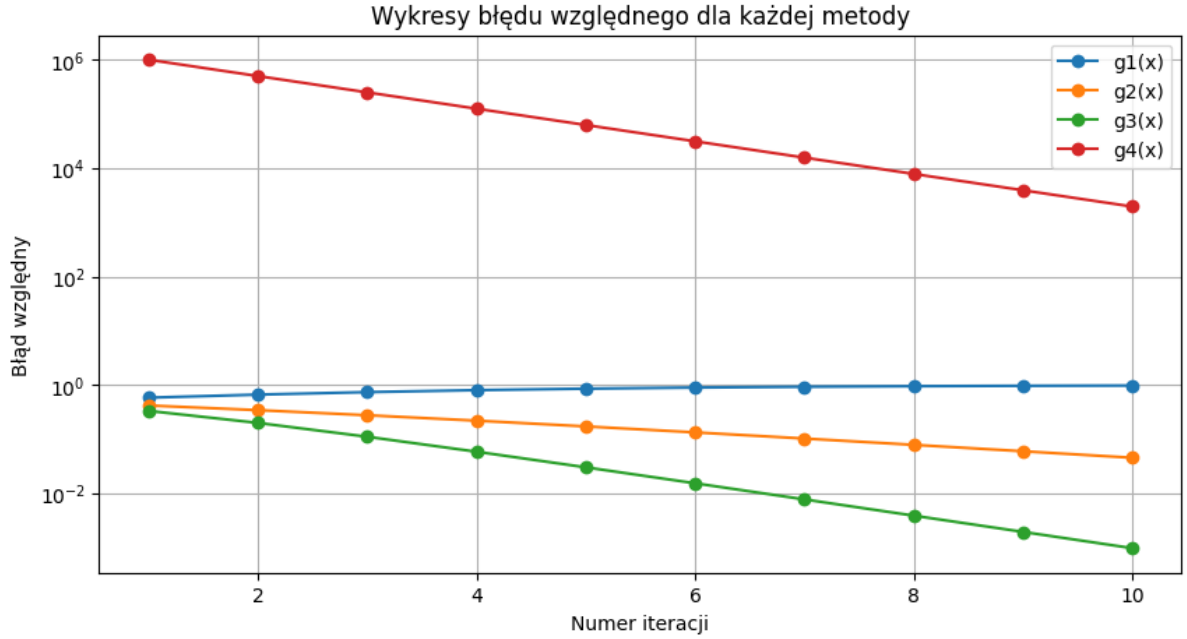
3.9. Tabela rozwiązania układu nieliniowego danego w zadaniu czwartym

Prawdziwe rozwiązania	Otrzymane rozwiązania metodą Newtona
0.7861513777574233	0.78615138
0.6180339887498949	0.61803399

Tabela 9. Tabela rozwiązanie układu nieliniowego danego w zadaniu czwartym

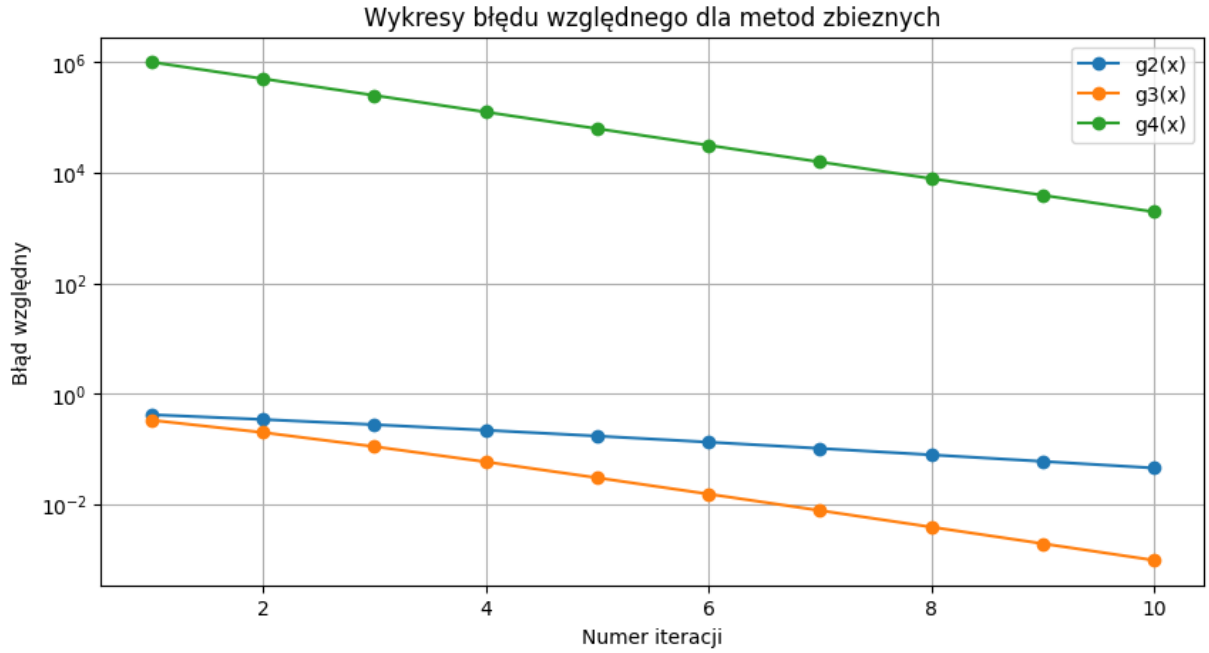
4. Wykresy

4.1. Wykresy błędu względnego dla każdej metody



Wykres 1. Wykres błędu względnego dla każdej z metod

4.2. Wykres błędu względnego tylko dla metod zbieżnych



5. Wnioski

5.1. Wnioski co do zadania pierwszego:

Jeśli chodzi o zadanie pierwsze oraz f$(x) = x^3 - 5x, x0 = 1$, to metoda Newtona zawodzi, ponieważ pochodna funkcji w pobliżu x0 jest zbyt mała, co prowadzi do problemów z konwergencją. Po modyfikacji punktu startowego lub użyciu alternatywnych metod, takich jak metoda bisekcji, udało się znaleźć pierwiastek.

W $f(x) = x^3 - 3x + 1, x0 = 1$, podobnie jak poprzednio, metoda Newtona zawiodła z powodu niewystarczającej początkowej wartości x0. Zastosowanie metody siecznych pozwoliło na znalezienie pierwiastka. Tutaj została wykorzystana funkcja brentq do rozwiązania tego problemu.

W $f(x) = 2 - x^5, x0 = 0.01$, punkt początkowy blisko 0 sprawia, że pochodna jest zbyt mała, co powoduje problemy z konwergencją. Zmiana punktu początkowego lub metoda bisekcji pozwala na znalezienie pierwiastka.

W $f(x) = x^4 - 4.29 * x^2 - 5.29, x0 = 0.8$, metoda Newtona była niestabilna z powodu problemów z pochodną w pobliżu punktu początkowego. Zwiększenie początkowego punktu naprawiło problem.

5.2. Wnioski co do zadania drugiego:

Implementacja schematów iteracyjnych potwierdziła wyniki analizy teoretycznej. Schematy g2, g3 i g4 wykazały zbieżność w ciągu 10 iteracji, podczas gdy g1 nie zbiegał się.

Rząd zbieżności dla schematów zbieżnych wynosił około 1, co zostało zweryfikowane zarówno teoretycznie, jak i eksperymentalnie.

Wykres błędu względnego dla każdej metody pokazał, że metody g2, g3 i g4 zbiegały się liniowo (rząd zbieżności 1).

Wykresy błędów względnych dla metod zbieżnych (g2, g3, g4) potwierdziły, że metoda g1 nie jest zbieżna.

5.3. Wnioski ogólne:

Metoda Newtona jest skuteczna, ale jej zbieżność zależy od wyboru punktu początkowego oraz właściwości pochodnej funkcji w pobliżu tego punktu.

W przypadkach problematycznych warto rozważyć modyfikacje punktu początkowego lub zastosowanie alternatywnych metod numerycznych.

Analiza wartości pochodnych w punktach docelowych pozwala na ocenę zbieżności schematów iteracyjnych.

Schematy iteracyjne są użyteczne, ale wymagają odpowiedniego wyboru funkcji iteracyjnej, aby zapewnić zbieżność do rzeczywistego pierwiastka.

6. Bibliografia
   Wykład MOwNiT - prowadzony przez dr. Inż. K. Rycerz
   Prezentacje – dr. Inż. M. Kuta

7. Dodatkowe informacje
   Rozwiązanie obu zadań znajduje się odpowiednio w pliku ex1_ex2.ipynb.