**Комп’ютерний практикум №1**

**Розв’язання нелінійних рівнянь**

**Виконав:**

Студент 3 курсу ФТІ

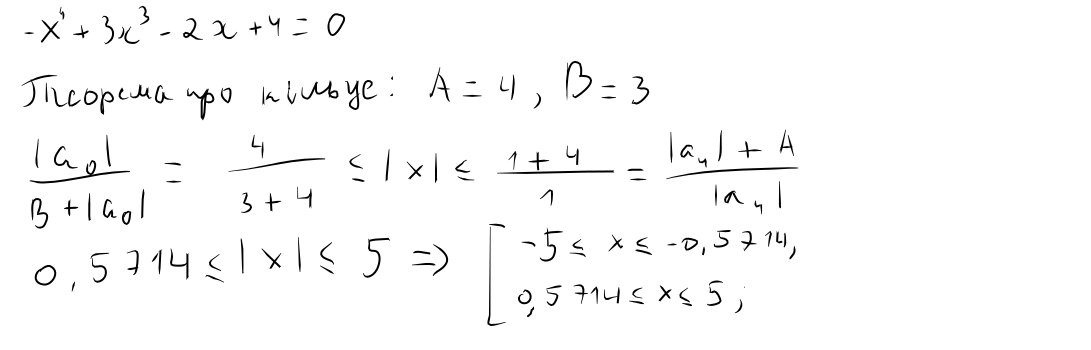
групи ФІ-92

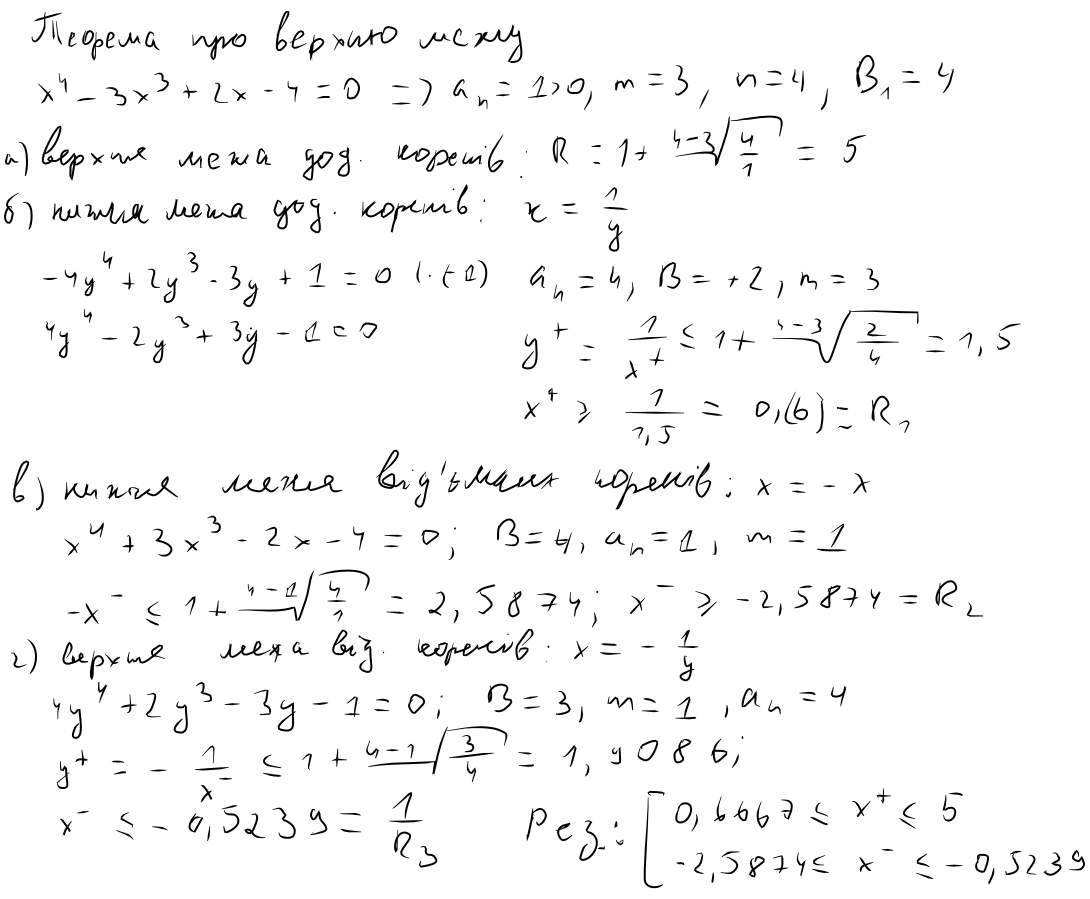
Поночевний Назар Юрійович

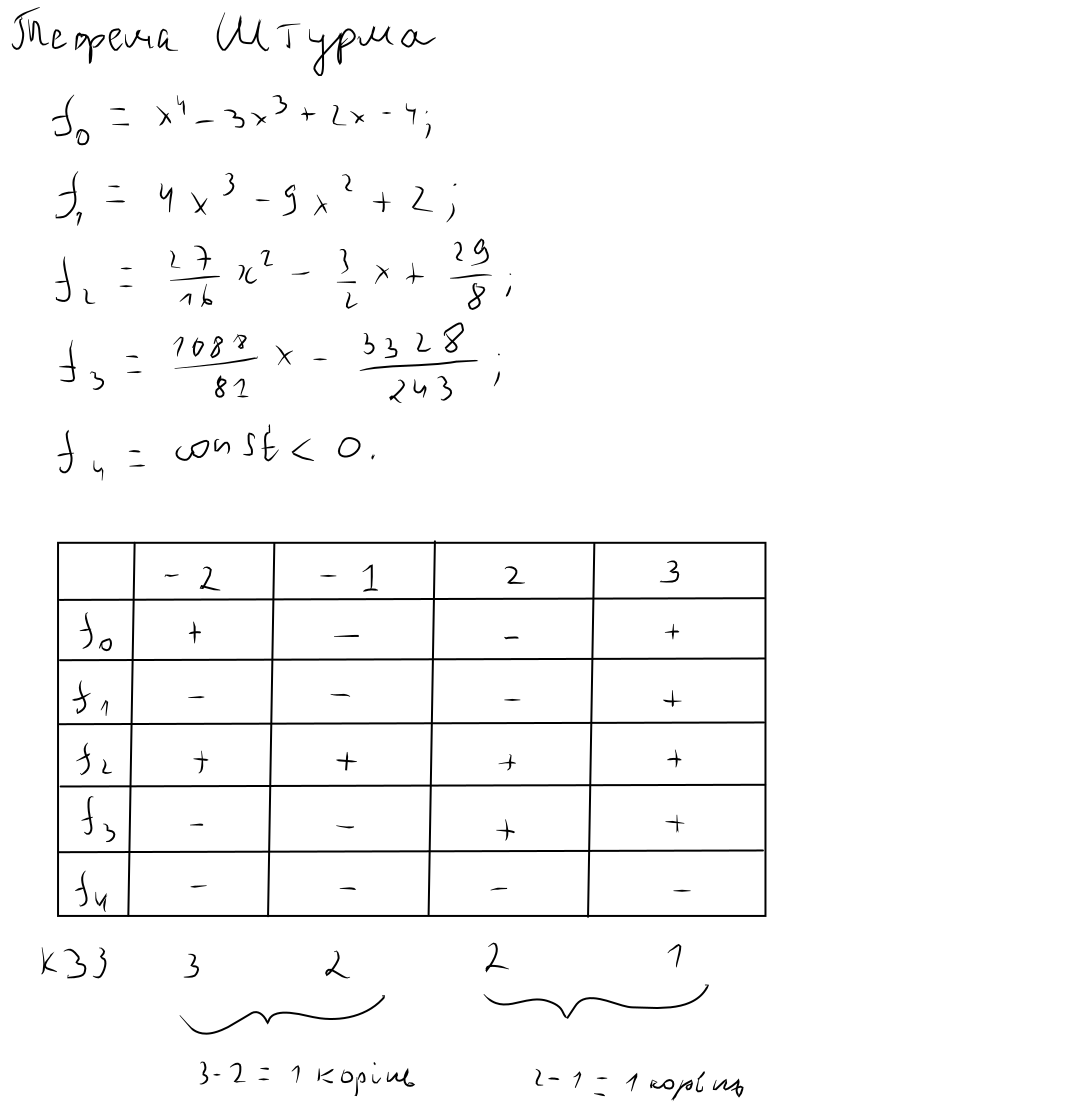
Варіант 14

**Завдання 1:**

Здійснити в якості допрограмового етапу аналіз та відокремлення коренів за допомогою теорем. Зокрема, визначити кількість дійсних коренів рівняння (теорема Гюа, теорема Штурма), відокремити дійсні корені рівняння (теорема про верхню межу). До аналізу комплексних коренів застосувати теорему про кільце. Результатом цього етапу повинна бути послідовність проміжків, кожен із яких містить лише один дійсний корінь рівняння;



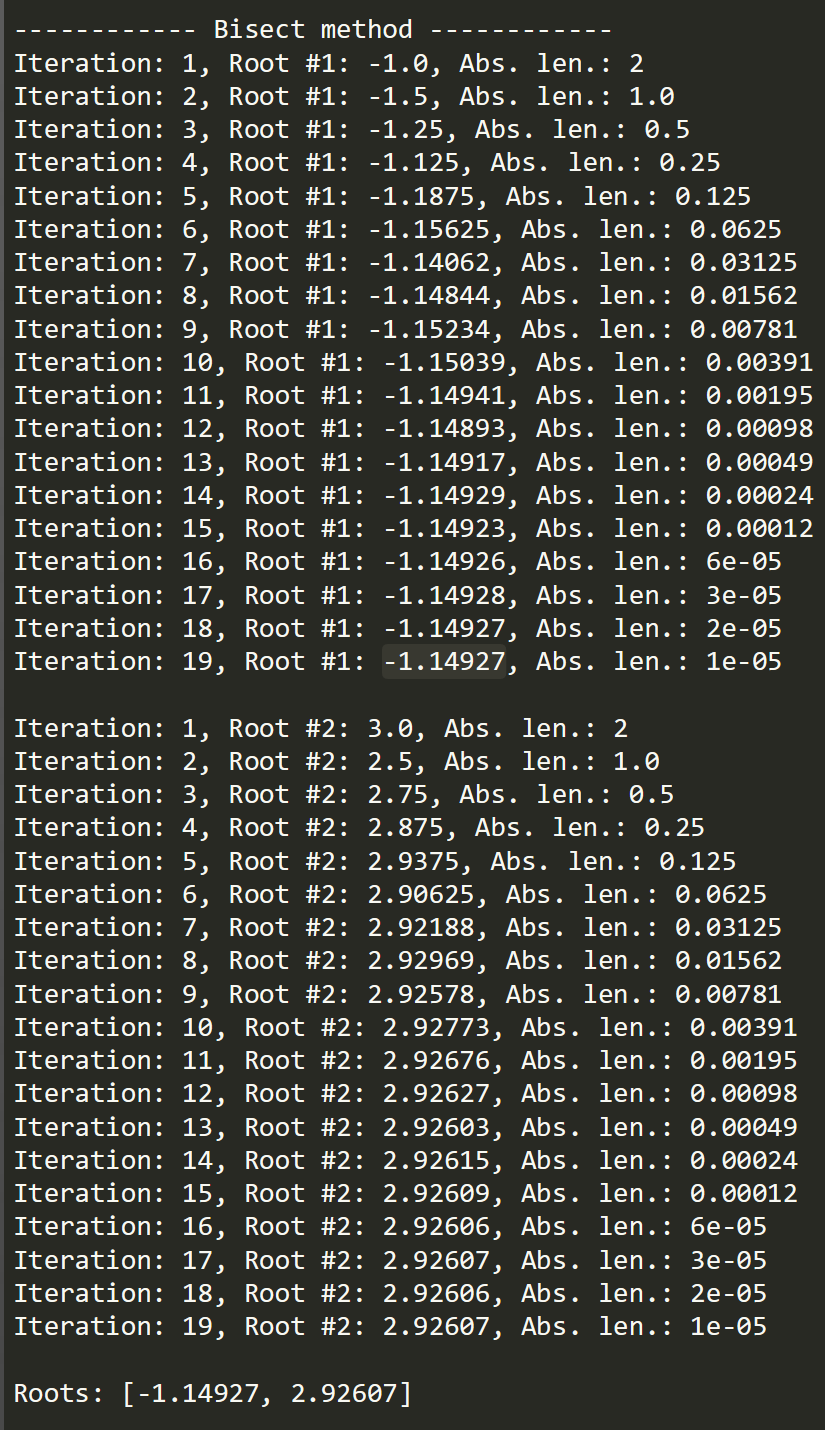


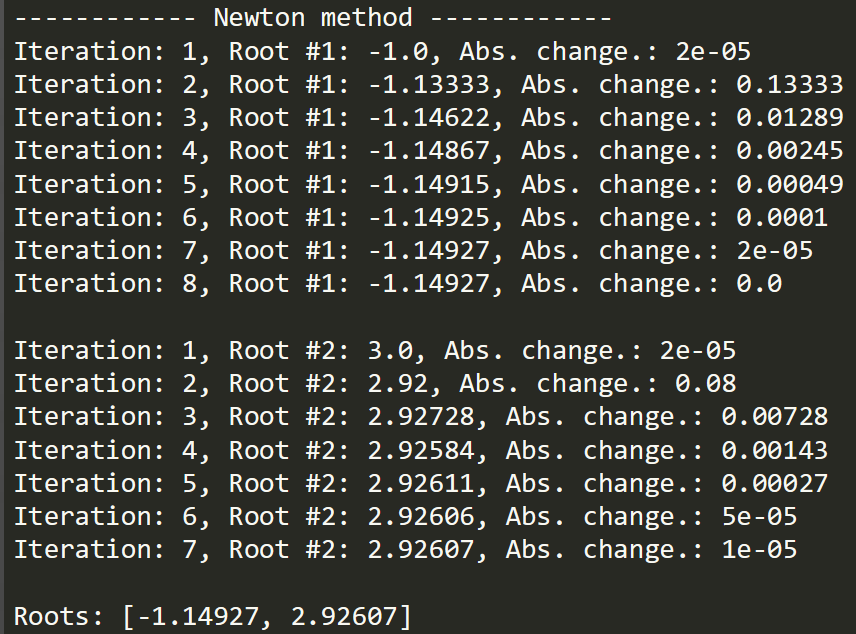
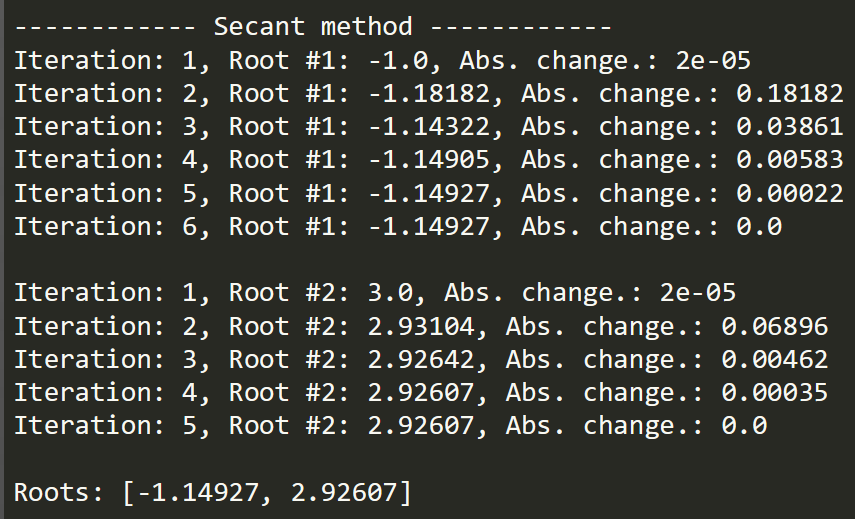


**Завдання 2:**

Програмний етап полягає в тому, щоб уточнити корені рівняння методом бісекції, методом хорд, методом дотичних;

| """ Solving polynoms  """  # ------------ Input ------------   def f(x):  return -x\*\*4 + 3 \* x\*\*3 - 2 \* x + 4   def df(x):  return -4 \* x\*\*3 + 9 \* x\*\*2 + 2   INTERVALS = [(-2, 0), (2, 4)] ACCURACY = 5   # ------------ Code ------------   def bisect(function, derivative, number, a, b, acc, counter=1, debug=True) -> float:  e = 10\*\*-acc  mid = (a + b) / 2  if debug:  print(f"Iteration: {counter}, Root #{number}: {round(mid, acc)}, Abs. len.: {round(abs(a - b), acc)}")  while abs(a - b) >= e:  if function(a) \* function(mid) <= 0:  b = mid  elif function(mid) \* function(b) <= 0:  a = mid  mid = (a + b) / 2  counter += 1  if debug:  print(f"Iteration: {counter}, Root #{number}: {round(mid, acc)}, Abs. len.: {round(abs(a - b), acc)}")  return mid   def secant(function, derivative, number, a, b, acc, counter=1, debug=True) -> float:  e = 10 \*\* -acc  mid = (a + b) / 2  x, x\_prev = mid, mid + 2 \* e  if debug:  print(f"Iteration: {counter}, Root #{number}: {round(x, acc)}, Abs. change.: {round(abs(x - x\_prev), acc)}")  while abs(x - x\_prev) >= e:  x, x\_prev = x - function(x) / (function(x) - function(x\_prev)) \* (x - x\_prev), x  counter += 1  if debug:  print(f"Iteration: {counter}, Root #{number}: {round(x, acc)}, Abs. change.: {round(abs(x - x\_prev), acc)}")  return x   def newton(function, derivative, number, a, b, acc, counter=1, debug=True) -> float:  e = 10 \*\* -acc  mid = (a + b) / 2  x, x\_prev = mid, mid + 2 \* e  if debug:  print(f"Iteration: {counter}, Root #{number}: {round(x, acc)}, Abs. change.: {round(abs(x - x\_prev), acc)}")  while abs(x - x\_prev) >= e:  x, x\_prev = x - function(x) / derivative(x), x  counter += 1  if debug:  print(f"Iteration: {counter}, Root #{number}: {round(x, acc)}, Abs. change.: {round(abs(x - x\_prev), acc)}")  return x   def find\_roots(function, derivative, intervals, method, accuracy=5, debug=True) -> list:  roots = []  for i, (a, b) in enumerate(intervals):  root = round(method(function, derivative,  i + 1, a, b, accuracy), accuracy)  roots.append(root)  if debug:  print()  return roots   def main():  print("\n------------ Bisect method ------------")  roots = find\_roots(f, df, INTERVALS, bisect, ACCURACY, True)  print("Roots:", roots)   print("\n------------ Secant method ------------")  roots = find\_roots(f, df, INTERVALS, secant, ACCURACY, True)  print("Roots:", roots)   print("\n------------ Newton method ------------")  roots = find\_roots(f, df, INTERVALS, newton, ACCURACY, True)  print("Roots:", roots)   if \_\_name\_\_ == "\_\_main\_\_":  main() |
| --- |





**Завдання 3:**

Порівняти отримані результати, зробити висновки, який метод приводить до меншої кількості ітерацій і чим це зумовлено.

1. Теорема про верхню границю + уточнення способом Лагранжа дало більш точні проміжки для пошуку коренів у моєму варіанті, ніж теорема про кільце, а теорема Штурма з кроком 1 змогла гарно розділити корені у цих проміжках;
2. Метод хорд у моєму варіанті справився за найменшу кількість ітерацій. Думаю, що це зумовлено виглядом функції у даних проміжках. На майже вертикальній прямій метод хорд трохи швидше ніж метод Ньютона і значно швидше ніж метод бісекцій зійшовся до кореня.