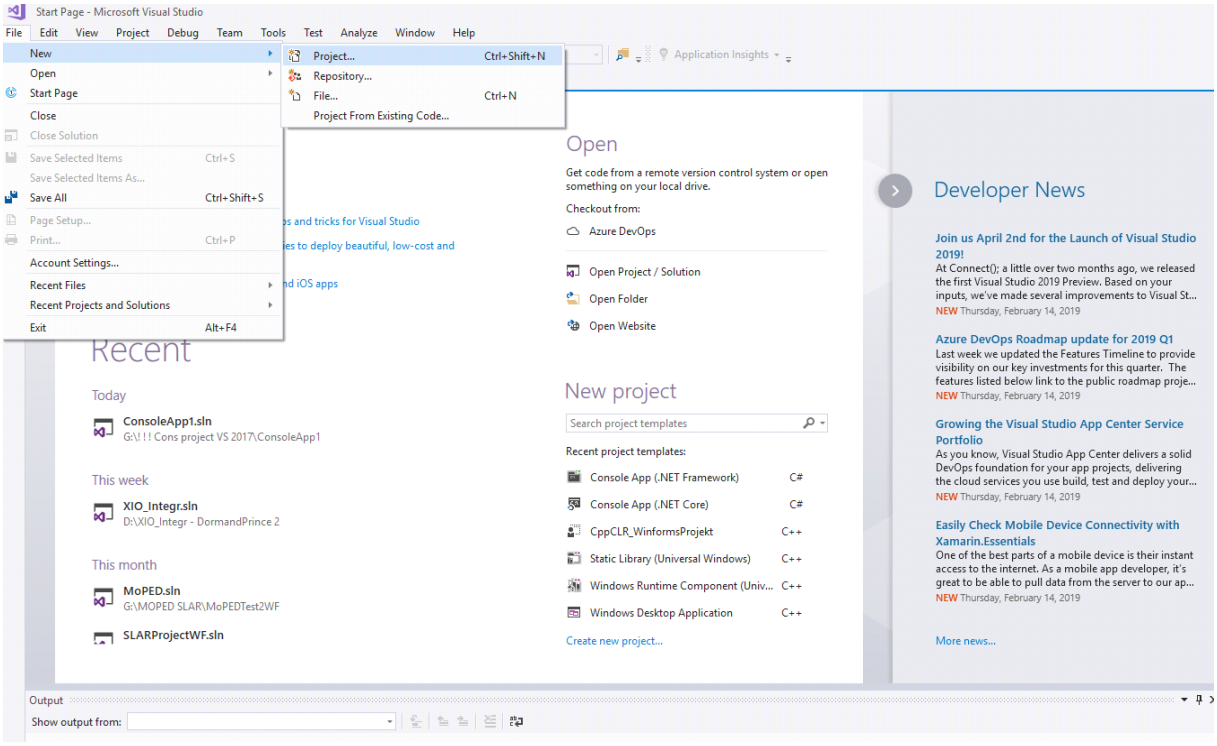
Консольний режим роботи середовища Visual Studio 2017 мовою C#

Мета роботи:

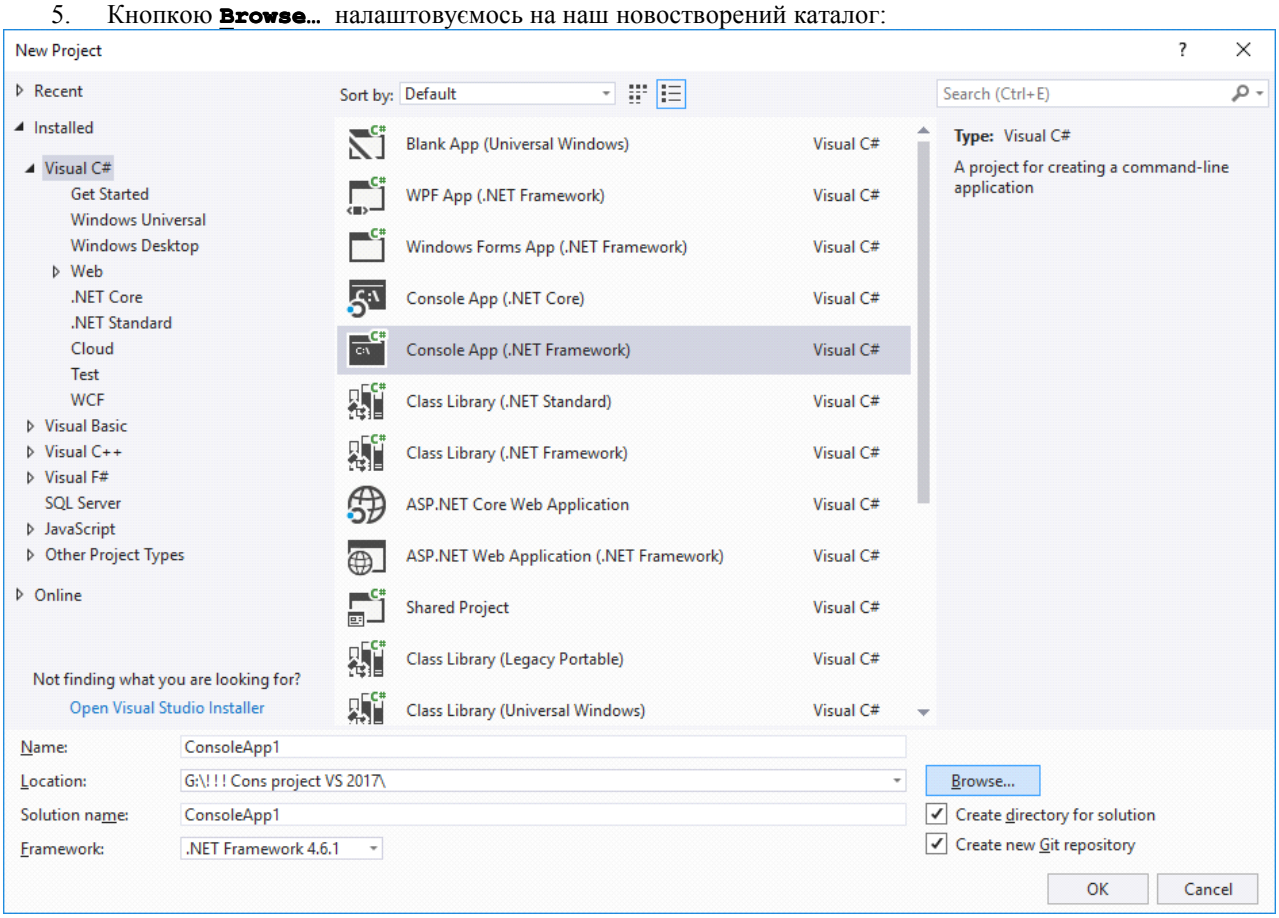
1. Освоїти роботу в консольному режимі середовища Visual Studio 2017 мовою C#.
2. Вивчити оператори введення/виведення даних для консольного режиму.
3. Повторити тему: “Методи розв’язання нелінійних рівнянь”.

Порядок виконання роботи

1. Створюємо у своєму каталозі папку, наприклад, **`! ! ! Cons project VS 2017.`**
2. Запускаємо на роботу Visual Studio 2012. Одержимо стартове вікно.
3. У його текстовому меню обираємо**: `FILE/New/Project`**…



4. У вікні **`New Project`** обираємо режим **`Console Application(.NET Framework)`**.
5. Кнопкою **`Browse…`** налаштовуємось на наш новостворений каталог:



6. У вікні, що відкриється, **програмуємо алгоритм методу ділення навпіл** для розв’язання нелінійних рівнянь:

- Описуємо функцію *f(x)*, що відповідає нашому нелінійному рівнянню.
- Задаємо інтервал $[a, b]$, на якому існує тільки один корінь рівняння, і бажану точність його обчислення Eps. За необхідності, табулюємо *f(x)*.
- Перевіряємо, чи виконується умова: $f(a) * f(b) < 0$, тобто чи шуканий корінь справді міститься в середині заданого інтервалу.
- Якщо ця умова не виконується, програма видає повідомлення про відсутність кореня на зазначеному інтервалі і завершує роботу.
- Якщо корінь на інтервалі $[a, b]$ – присутній, то визначаємо середину інтервалу, тобто точку $c = \frac{a+b}{2}$, і перевіряємо умову: $f(a) * f(c) < 0$.
- Якщо вона виконується, то праву межу інтервалу b переносимо в середню точку c – шляхом присвоєння **`b = c;`**. У протилежному випадку в точку c переносимо ліву межу (**`a = c;`**).
  - Поділ уточненого інтервалу $[a, b]$ навпіл і перенесення однієї із меж інтервалу продовжуємо доти, доки не виконається умова $b - a < Eps$.
  - Друкуємо обчислене наближене значення кореня і значення лічильника $Lich$ кількості поділів інтервалу.
  - У програмі також перевіряємо, чи точка a, або точка b, або точка не знаходиться поблизу шуканого кореня (у межах заданої нами точності Eps).

7. Повний текст одного із варіантів програмної реалізації алгоритму МДН наведено нижче

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace ConsoleApplication2
{
 class Program
 {
  public static double f(double x)
    { return x * x - 4;
    }
        static void Main(string[] args)
       {
        double a, b, c, Eps;
        int Lich = 0;
        Console.WriteLine("Input a, b, eps");
        a = Convert.ToDouble(Console.ReadLine());
        b = Convert.ToDouble(Console.ReadLine());
        Eps = Convert.ToDouble(Console.ReadLine());
        if (f(a) * f(b) > 0)                                    // Перевіряємо, чи є корінь на інтервалі [a, b]
        {     Console.WriteLine("No root in the interval");
              Console.ReadLine();                               // Затримка показу повідомлення
              return;
        }
        if (Math.Abs(f(a)) < Eps)                               // Перевіряємо, чи ліва межа не є коренем
             {Console.WriteLine("x = " + a + "      Lich = " + Lich);
              Console.ReadLine();                               // Затримка показу повідомлення
              return;
         }
         else
         if (Math.Abs(f(b)) < Eps)                              // Перевіряємо, чи права межа не є коренем
                  { Console.WriteLine("x = " + b + "    Lich = " + Lich);
                    Console.ReadLine();                         // Затримка показу повідомлення
                      return;
                  }
                  else
                      {
                          while (Math.Abs(b - a) > Eps)         // Цикл Методу ділення навпіл
                          {
                              c = (a + b) / 2;
                              Lich++;
                              if (Math.Abs(f(c)) < Eps)         // Перевіряємо, чи точка с не є коренем
                              {
                                  Console.WriteLine("x = " + c + "    Lich = " + Lich);
                                  Console.ReadLine();           // Затримка показу повідомлення
```

```
                    return;
                }
                else if (f(a) * f(c) < 0)   b = c;        // Звуження інтервалу пошуку кореня
                                            else    a = c;
            }
            Console.WriteLine("x = " + (a + b) / 2 + " Lich = " + Lich);
            Console.ReadLine();                           // Затримка показу повідомлення
            return;
        }
    }
  }
}
```

Завдання:

1. Запустити програму МДН у консольному режимі і дослідити її роботу.
2. Запустити цю програму в автономному копіляторі **csc.exe** і порівняти розміри **exe**-файлу для обох варіантів програми.
3. У консольному виконанні самостійно запрограмувати ще один популярний алгоритм знаходження, із точністю ε, кореня нелінійного рівняння: $f(x) = 0$ (1) – **метод Ньютона** (дотичних). Нагадаєто його теорію:

Задамо де-яке початкове наближення $x_0 \in [a,\ b]$ і лінеаризуємо функцію $f(x)$ в околі x_0 за допомогою частини ряду Тейлора: $f(x) = f(x_0) + f'(x_0)\cdot(x - x_0)$.

Замість рівняння (1) розв’яжемо лінеаризоване рівняння $f(x_0) + f'(x_0)\cdot(x - x_0) = 0$, трактуючи його розв’язок x як перше наближення до кореня $x_1 = x_0 - \dfrac{f(x_0)}{f'(x_0)}$. Продовжуючи цей процес m раз, приходимо до **формули Ньютона**: $x_{m+1} = x_m - \dfrac{f(x_m)}{f'(x_m)}$, (2)

яка є ітераційним процесом з ітераційною функцією $s(x) = x - \dfrac{f(x)}{f'(x)}$.

Геометричну інтерпретацію цього процесу проілюстровано на рис. 1. Рівняння (2) є рівнянням прямої, дотичної до кривої $f(x)$ у точці x_0, отож, цей метод називають **методом дотичних**.

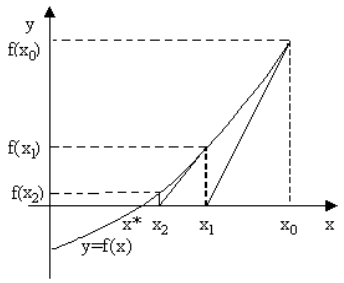


Рис. 1. Геометричне трактування методу Ньютона (дотичних)

Ітераційну формулу методу Ньютона можна також отримати, використовуючи означення похідної функції $f(x)$ у де-якій точці x_0, як тангенсу кута α дотичної до графіку функції у точці x_0 з віссю ox (рис. 2.)

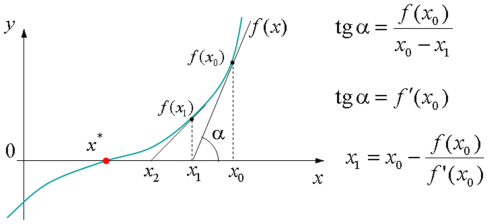


Рис. 2. Виведення ітераційної формули методу Ньютона

Головною характеристикою ітераційних процесів є їхня **збіжність**. Збіжність методу Ньютона оцінюється нерівністю $|x_{m+1} - x^*| \le |x_m - x^*|^2 \cdot \dfrac{M_2}{m_1}$, де $M_2 = max|f''(x)|$, $m_1 = 2min|f'(x)|$, $x \in [a, b]$. Таку збіжність називають **квадратичною**, оскільки похибка кожного кроку обчислень є пропорційна до квадрату попередньої.

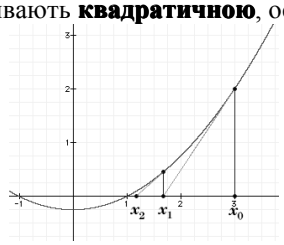


Рис. 3. Метод збігається

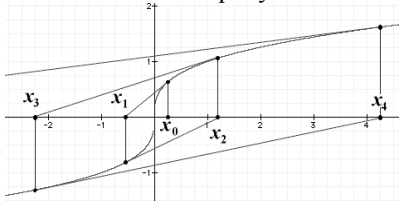


Рис. 4. Метод не збігається

Зазначимо, що поліпшення збіжності методу Ньютона, порівняно з іншими методами, досягається збільшенням витрат на виконання кожного кроку, оскільки на кожному кроці треба обчислювати не тільки значення функції *f(x)*, але й значення її похідної $f'(x)$.

Щоб позбутися необхідності обчислювати похідну на кожно-му кроці, використовують **модифікований метод Ньютона**, у якому похідну обчислюють тільки один раз: $x_{m+1} = x_m - \frac{f(x_m)}{f'(x_0)}$. (3)

Цей метод можна також вважати методом релаксацій із параметром $\tau = -1/f'(x_0)$. Його збіжність є лінійною.

Можна довести, що початкове наближення x_0 для початку ітераційного процесу необхідно обирати таким, щоб виконувалась умова $f(x_0) * f''(x_0) > 0$. В іншому випадку ітерації методу Ньютона можуть і не збігатись до розв’язку.

Алгоритм методу Ньютона

Якщо локалізовано проміжок $[a, b]$, на якому розташований корінь рівняння (1), його уточнюємо алгоритмом Ньютона, який програмуємо так:

В **описовій** частині консольної програми описуємо:

1. Три функції дійсного типу від аргументу *x* з іменами *f*, *fp* і *f2p*, які, для заданого значення *x*, обчислюють, відповідно: значення функції *f*, її першу та другу похідні.
2. Змінні дійсного типу: *a*, *b*, точність обчислення кореня *Eps*, робочі змінні *x*, *D*, *Dx*.
3. Змінні цілого типу: *i* – параметр циклу; *Kmax* – максималь-но допустиму кількість ітерацій.

У **виконуваній** частині цієї програми:

1. Вводимо межі *a*, *b* проміжку локалізації кореня, значеннння точності *Eps* та кількості ітерацій *Kmax*.
2. Надаємо змінній *x* значення *b*.
3. Якщо *f(x)*f2p(x)<0*, то надаємо змінній *x* значення *a*, інакше – переходимо до кроку 6;
4. Якщо *f(x)*f2p(x)>0*, – переходимо до кроку 6, інакше
5. Виводимо на екран повідомлення: “**Для заданого рівняння збіжність методу Ньютона не гарантується**” і потім:
6. Починаємо цикл по *i* від *1* до *Kmax*, у тілі якого:
   - обчислюємо *Dx = f(x) / fp(x)*;
   - виконуємо ітерацію Ньютона: *x = x – Dx*;
   - якщо **модуль** зачення *Dx* є більшим за *Eps* – продовжуємо виконувати цей ітераційний цикл, якщо – ні, то:
   - виводимо на екран шукане наближене значення кореня *x*;
   - завершуємо роботу програми.
7. У випадку успішного завершення цього циклу, тобто, якщо корінь не знайдено, виводимо повідомлення: “**За задану кількість ітерацій корінь з точністю Eps не знайдено**”.
8. Завершуємо роботу програми.

Примітка: Значення виразів для першої та другої похідних можна обчислити “вручну” за правилами математичного аналізу. Програма буде універсальнішою, якщо похідні знаходити чисельно.

Першу похідну визначають за означенням похідної: *fp(x) =(f(x + D) - f(x)) / D* .

Другу похідну визначають одним із двох способів:

- аналогічно до першої: *f2p(x) =(fp(x + D) - fp(x)) / D* , або
- за чисельною триточковою формулою: *f2p(x) =(f(x + D) + f(x - D) - 2*f(x)) / D²* .

Значення *D* доцільно обирати таким: *D = Eps / 100.0*.

Після кожного оператора виведення на екран повідомлень чи результату необхідно затримувати зображення екрана, наприклад, оператором `Console.ReadLine();`.

4. Захистити алгоритм і програму методу Ньютона.

Додаткове завдання:

Дослідити форматоване виведення на текстовий екран дійсного числа:

```
using System;
namespace Float2String
{
 class Class1
  {
   [STAThread]
   static void Main(string[] args)
    { double d = 3.1415926;
      Console.WriteLine("{0}", d);                 // 3,1415926
      Console.WriteLine("{0:F2}", d);              // 3,14
      Console.WriteLine(string.Format("{0}", d));  // 3,1415926
      Console.WriteLine(string.Format("{0:F3}", d)); // 3,142
      Console.WriteLine(d.ToString("F2"));         // 3,14
      Console.WriteLine(d.ToString("E"));          // 3,141593E+000
      Console.WriteLine(d.ToString("C"));          // 3,14 грн.
      Console.WriteLine(d.ToString("G"));          // 3,1415926
      Console.WriteLine(d.ToString("R"));          // 3,1415926
      Console.ReadLine();
    }
  } }
```