МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Факультет “Информатика и системы управления”
Кафедра “Системы обработки информации и управления”

Дисциплина “Парадигмы и конструкции языков программирования”

Отчет по домашнему заданию

Выполнил:
Студент группы ИУ5-36Б
Гаврилик Я. С.
Преподаватель:
Гапанюк Ю. Е.

Москва 2025

1. Задание

1.1 Подготовить краткую информацию о языке программирования Fortran.

1.2 Разработать проект на языке программирования Fortran.

Известны два представления числа π:

$$\pi = 3 + 4 * \left(\frac{1}{2*3*4} - \frac{1}{4*5*6} + \frac{1}{6*7*8} - \ldots \right)$$

$$\pi = \sqrt{6 * \left(1 + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \ldots \right)}$$

Сравнить скорость сходимости каждого представления. Полученные результаты расчета оформить в виде таблицы: точность, количество членов ряда.

2. Краткая информация о языке программирования Fortran

Fortran (от *FORmula TRANslation*) — это не просто старейший язык программирования высокого уровня, созданный в 1957 году под руководством Джона Бэкуса в IBM. Это живая технологическая легенда, которая продолжает активно развиваться и использоваться в самых требовательных областях современной науки и инженерии. За более чем 65 лет существования Fortran прошел путь от примитивного языка для программирования мейнфреймов до мощного инструмента для суперкомпьютерных вычислений XXI века.

Fortran появился в эпоху, когда программирование велось на машинных языках и ассемблере, и каждая новая задача требовала огромных усилий по оптимизации. Целью создателей было разработать язык, который бы позволил ученым и инженерам записывать математические формулы в естественной форме, близкой к алгебраической нотации. Первые версии Fortran (I, II, III) были революционными — они ввели понятия переменных, циклов, условных операторов и подпрограмм.

Настоящий прорыв произошел с Fortran IV (1966), который стал стандартом де-факто и использовался на таких исторических проектах, как программа "Аполлон". Fortran 77 добавил структурированное программирование, а Fortran 90 (на примере которого написан приведенный код) стал кардинальной модернизацией, включив динамическое выделение памяти, модули, операции над массивами и современный синтаксис.

Fortran создавался с четкой прагматичной целью — эффективно решать вычислительные задачи. Его дизайн отражает несколько ключевых принципов:

1. **Производительность превыше всего** — компиляторы Fortran исторически генерируют исключительно оптимизированный машинный код для математических операций.
2. **Минимализм синтаксиса** — язык содержит ровно те конструкции, которые необходимы для численных расчетов.
3. **Четкое разделение данных и кода** — Fortran строго разграничивает секции объявлений и исполняемых операторов.
4. **Естественная запись формул** — математические выражения записываются максимально близко к их алгебраической форме.

Сегодня Fortran остается доминирующим языком в областях, где важна скорость обработки больших массивов данных:

- **Климатическое и метеорологическое моделирование** (модели общей циркуляции атмосферы)
- **Вычислительная гидродинамика** (расчеты течений жидкости и газа)

- **Квантовая химия и физика** (программы Gaussian, VASP, Quantum ESPRESSO)
- **Астрофизика и космология** (симуляции галактик и звездной динамики)
- **Машинное обучение научных данных** (нейросети для анализа экспериментальных данных)

Стандарты Fortran 2003, 2008 и 2018 добавили поддержку объектно-ориентированного программирования, параллельных вычислений (Coarrays, DO CONCURRENT), интероперабельность с языком C и улучшенную обработку исключений.

3. Разработка проекта на языке программирования Fortran.

3.1 Среда разработки проекта

Разработка проекта обеспечена посредством использования утилиты GFortran.

Инсталляция утилиты выполнена в терминале Linux в VSCode в результате выполнения команды: sudo apt install gfortran.

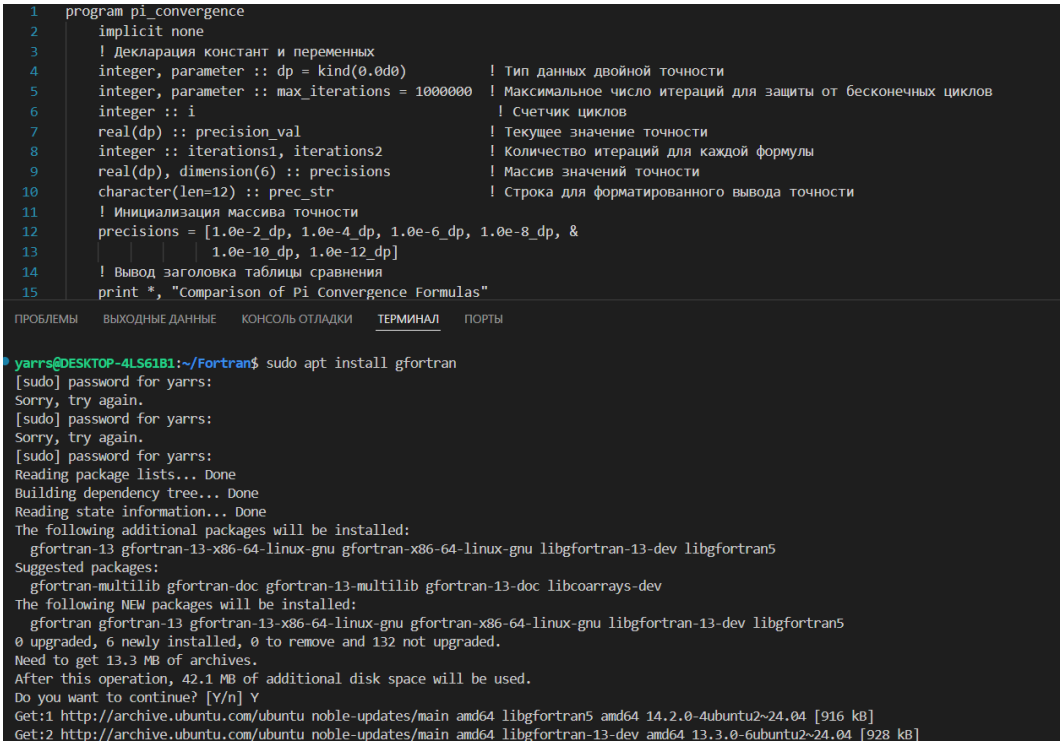

Рис. 1 – Среда разработки проекта на языке программирования Fortran

3.2 Листинг программного кода проекта

Следующий код демонстрирует типичный стиль программирования на современном Fortran — структурированный, модульный и ориентированный на численные методы:

```
program pi_convergence
  implicit none
  ! Декларация констант и переменных
  integer, parameter :: dp = kind(0.0d0)         ! Тип данных двойной точности
  integer, parameter :: max_iterations = 1000000  ! Максимальное число итераций для защиты от бесконечных циклов
  integer :: i                                    ! Счетчик циклов
```

```
    real(dp) :: precision_val                  ! Текущее значение точности
    integer :: iterations1, iterations2        ! Количество итераций для каждой формулы
    real(dp), dimension(6) :: precisions       ! Массив значений точности
    character(len=12) :: prec_str              ! Строка для форматированного вывода точности
    ! Инициализация массива точности
    precisions = [1.0e-2_dp, 1.0e-4_dp, 1.0e-6_dp, 1.0e-8_dp, &
                  1.0e-10_dp, 1.0e-12_dp]
    ! Вывод заголовка таблицы сравнения
    print *, "Comparison of Pi Convergence Formulas"
    print *, "====================================="
    print *, "Precision     Formula 1     Formula 2"
    print *, "-------------------------------------"
    ! Основной цикл: вычисление для каждого уровня точности
    do i = 1, 6
       precision_val = precisions(i)  ! Текущая точность вычислений
       ! Вычисление количества итераций по первой формуле
       call formula1_iterations(precision_val, iterations1)
       ! Вычисление количества итераций по второй формуле
       call formula2_iterations(precision_val, iterations2)
       ! Форматирование строки с точностью для красивого вывода
       if (abs(precision_val - 1.0e-2_dp) < 1.0e-10_dp) then
          prec_str = "1e-02"
       else if (abs(precision_val - 1.0e-4_dp) < 1.0e-10_dp) then
          prec_str = "1e-04"
       else if (abs(precision_val - 1.0e-6_dp) < 1.0e-10_dp) then
          prec_str = "1e-06"
       else if (abs(precision_val - 1.0e-8_dp) < 1.0e-10_dp) then
          prec_str = "1e-08"
       else if (abs(precision_val - 1.0e-10_dp) < 1.0e-10_dp) then
          prec_str = "1e-10"
       else if (abs(precision_val - 1.0e-12_dp) < 1.0e-10_dp) then
          prec_str = "1e-12"
       else
          write(prec_str, '(ES8.1)') precision_val
          prec_str = adjustl(prec_str)
       end if
       ! Вывод строки результатов с выравниванием колонок
       print '(A12, I12, I15)', trim(prec_str), iterations1, iterations2
    end do
    ! Вывод заключительной информации
    print *, "-------------------------------------"
    print *, "Formula 1: pi = 3 + 4*(1/(2*3*4) - 1/(4*5*6) + ...)"
    print *, "Formula 2: pi = sqrt(6*(1 + 1/2^2 + 1/3^2 + ...))"
    print *, "Conclusion: Formula 1 converges faster!"
contains
    ! Подпрограмма для вычисления итераций по формуле 1
    ! Формула: π = 3 + 4 * (1/(2×3×4) - 1/(4×5×6) + 1/(6×7×8) - ...)
```

```
  subroutine formula1_iterations(precision, iterations)
    real(dp), intent(in) :: precision    ! Входная точность вычислений
    integer, intent(out) :: iterations   ! Выходное количество итераций
    real(dp) :: term                     ! Текущее слагаемое ряда
    real(dp) :: sign_val                 ! Знак текущего слагаемого (+1 или -1)
    real(dp) :: denominator              ! Знаменатель текущего слагаемого
    integer :: n                         ! Номер текущего слагаемого
    ! Инициализация переменных
    sign_val = 1.0_dp                    ! Первое слагаемое положительное
    n = 2                                ! Начинаем с n = 2
    iterations = 0                       ! Счетчик итераций
    ! Основной цикл суммирования ряда
    do while (iterations < max_iterations)
      ! Вычисление знаменателя: n × (n+1) × (n+2)
      denominator = real(n, dp) * real(n + 1, dp) * real(n + 2, dp)
      ! Вычисление текущего слагаемого
      term = sign_val / denominator
      ! Критерий остановки: когда вклад текущего слагаемого становится меньше заданной точности
      ! Умножаем на 4, так как в формуле каждое слагаемое умножается на 4
      if (abs(4.0_dp * term) < precision) then
        exit  ! Выход из цикла при достижении требуемой точности
      end if
      ! Подготовка к следующей итерации
      sign_val = -sign_val   ! Чередование знаков
      n = n + 2              ! Переход к следующему набору трех последовательных чисел
      iterations = iterations + 1  ! Увеличение счетчика итераций
    end do
    ! Защита от бесконечного цикла: если достигнут предел итераций
    if (iterations >= max_iterations) then
      iterations = max_iterations
    end if
  end subroutine formula1_iterations
  ! Подпрограмма для вычисления итераций по формуле 2
  ! Формула: π = √(6 × (1 + 1/2² + 1/3² + 1/4² + ...))
  subroutine formula2_iterations(precision, iterations)
    real(dp), intent(in) :: precision    ! Входная точность вычислений
    integer, intent(out) :: iterations   ! Выходное количество итераций
    real(dp) :: term                     ! Текущее слагаемое ряда
    real(dp) :: sum_val                  ! Накопленная сумма ряда
    integer :: n                         ! Номер текущего слагаемого
    ! Инициализация переменных
    sum_val = 0.0_dp                     ! Начальная сумма ряда
    n = 1                                ! Начинаем с n = 1
    iterations = 0                       ! Счетчик итераций
    ! Основной цикл суммирования ряда
    do while (iterations < max_iterations)
```

```
        ! Вычисление текущего слагаемого: 1/n²
        term = 1.0_dp / (real(n, dp) ** 2)
        ! Добавление слагаемого к сумме
        sum_val = sum_val + term
        ! Критерий остановки: когда текущее слагаемое становится меньше заданной точности
        if (abs(term) < precision) then
           exit  ! Выход из цикла при достижении требуемой точности
        end if
        ! Подготовка к следующей итерации
        n = n + 1                    ! Переход к следующему целому числу
        iterations = iterations + 1     ! Увеличение счетчика итераций
      end do
      ! Защита от бесконечного цикла: если достигнут предел итераций
      if (iterations >= max_iterations) then
        iterations = max_iterations
      end if
   end subroutine formula2_iterations
end program pi_convergence
```

3.3 Результаты реализации проекта

```
Comparison of Pi Convergence Formulas
====================================
 Precision    Formula 1    Formula 2
------------------------------------
    1e-02         3           10
    1e-04        16          100
    1e-06        78         1000
    1e-08       367        10000
    1e-10      1709       100000
    1e-10      7936      1000000
------------------------------------
Formula 1: pi = 3 + 4*(1/(2*3*4) - 1/(4*5*6) + ...)
Formula 2: pi = sqrt(6*(1 + 1/2^2 + 1/3^2 + ...))
Conclusion: Formula 1 converges faster!
```

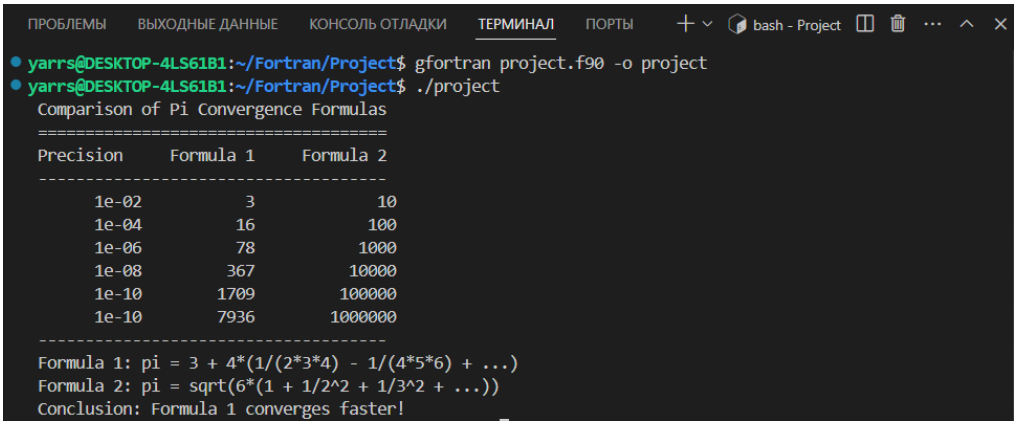

Рис. 2 – Снимок экрана с результатами выполнения программы

4. Заключение

Результатами реализации проекта являются:

1. Краткая информация о языке программирования Fortran.
2. Программа на языке Fortran обеспечивающая сравнение скорости сходимости представлений числа π.