**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

**(СПбГУТ)**

Лабораторная работа №7

по курсу

«Логическое и функциональное программирование»

Выполнил:

студент группы ИКПИ-14

Хохлов Т.В.

Принял:

доцент кафедры ПИиВТ

Ерофеев С.А.

Санкт-Петербург

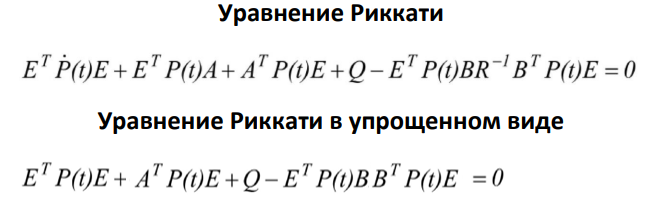
2023 г.

**Цель работы**

Разработать программу на языке Haskell для следующей задачи:

решения матричного уравнения Риккати методами Нюстрема 2-го, 3-го и 4-го порядков с визуализацией результатов и подсчётом времени выполнения алгоритмов.

**Описание алгоритм**



Уравнение Риккати – дифференциальное матричное уравнение 1-ого порядка. Решением данного уравнения является нахождение приближенного значения матрицы P(t) на отрезке времени от t0­ до tmaxс заданным шагом h и точностью **ε**.

Сначала на вход алгоритму подаются матрицы:

A – матрица коэффициентов размером NxN

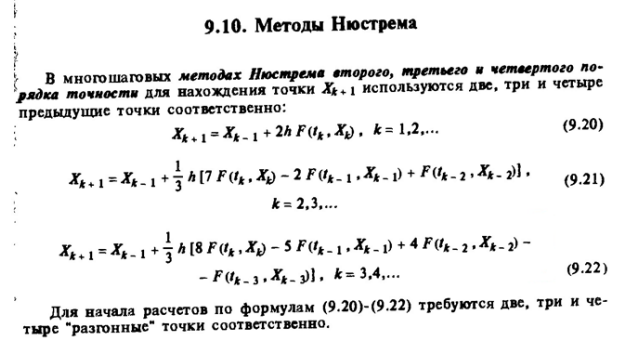
B – матрица коэффициентов размером MxN

E – единичная матрица с возможным заполнением нулями в нижнем правом секторе размером NxN

Q – единичная матрица с возможным заполнением нулями в нижнем правом секторе размером NxN

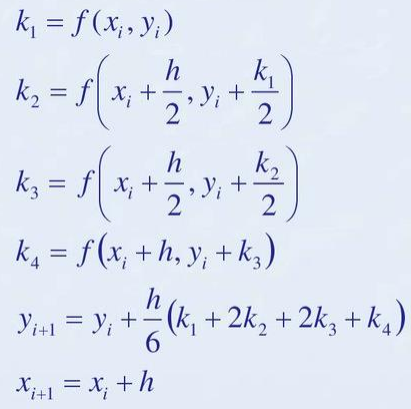
Размерности матриц имеют значение для их правильного произведения.

Методы Нюстрема представляют собой схемы численного интегрирования повышенного порядка точности, основанные на использовании предыдущих значений P и значений производной R(P). В данной работе реализованы схемы второго, третьего и четвёртого порядка, которые требуют соответственно одной, двух и трёх предыдущих точек для вычисления новой.



Чтобы применить метод Нюстрема порядка n, нужны предыдущие значения матрицы P в точках t₀, t₀ + h, ..., t₀ + (n-1) · h. Так как начальное значение P₀ задано (обычно нулевая матрица), оставшиеся значения генерируются с помощью метода Рунге-Кутты 4-го порядка, так как он является устойчивым и достаточно точным.

Рунге-Кутта 4-го порядка рассчитывает следующее значение по формулам:



На каждом шаге выполнения проверяется, что полученная матрица P не содержит NaN или Inf (например, при численных переполнениях) и вычисляется максимальная ошибка: error = ||P\_{n+1} − P\_n||\_∞

Если ошибка меньше заданного порога ε, итерации останавливаются. Также введена защита от зацикливания: если количество шагов превышает максимальное значение (например, 300 000 заданное параметрически), выполнение прекращается.

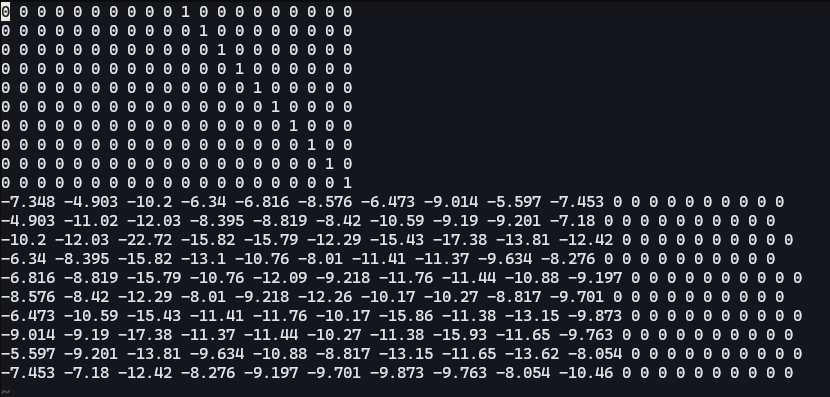
Также на каждом этапе метод сохраняет промежуточные значения в лог-файлы (logs/log2.txt, log3.txt, log4.txt) и записывает итоговое значение матрицы P, параметры запуска и время выполнения алгоритма в файл Results/outN.txt, где N — порядок метода.

**Используемые функции**

* riccatiEquation — формулирует уравнение Риккати.
* rungeKutta4 — классический метод Рунге-Кутты 4-го порядка.
* nystrom2, nystrom3, nystrom4 — методы Нюстрема соответствующих порядков.
* generateInitialPoints — генерация начальных точек для Нюстрема.
* isValidMatrix — проверка матрицы на NaN/∞.
* writeLog, logProgress, writeResults — логирование и сохранение результатов.
* runNystromMethod — обёртка, запускающая соответствующий метод.
* main — основная точка входа, запускает три метода Нюстрема.
* LA.Matrix, LA.Vector  
  Типы для хранения матриц и векторов.
* LA.fromLists :: [[Double]] -> Matrix  
  Преобразует список списков чисел в матрицу.
* LA.toLists :: Matrix -> [[Double]]  
  Преобразует матрицу обратно в список списков (для логирования и проверок).
* LA.<>  
  Оператор матричного умножения.
* LA.tr :: Matrix -> Matrix  
  Транспонирование матрицы.
* LA.scale :: Double -> Matrix -> Matrix  
  Умножение всей матрицы на скаляр.
* LA.norm\_Inf :: Matrix -> Double  
  Норма матрицы (максимальное абсолютное значение по строкам).
* LA.konst :: Double -> (Int, Int) -> Matrix  
  Создаёт матрицу заданного размера, заполненную константой.
* LA.subMatrix :: (Int, Int) -> (Int, Int) -> Matrix -> Matrix  
  Извлекает подматрицу из заданной позиции и размера.

**Тестирование**

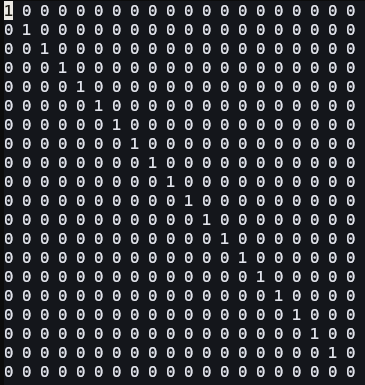
Возможный ввод матриц:

Матрица A.dat:

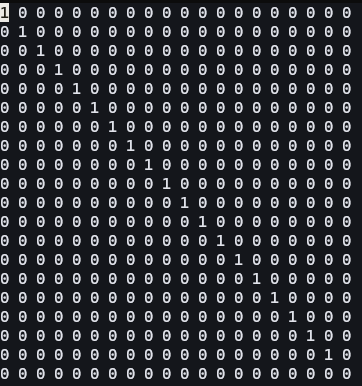
Матрица B.dat:



Матрица E.dat:

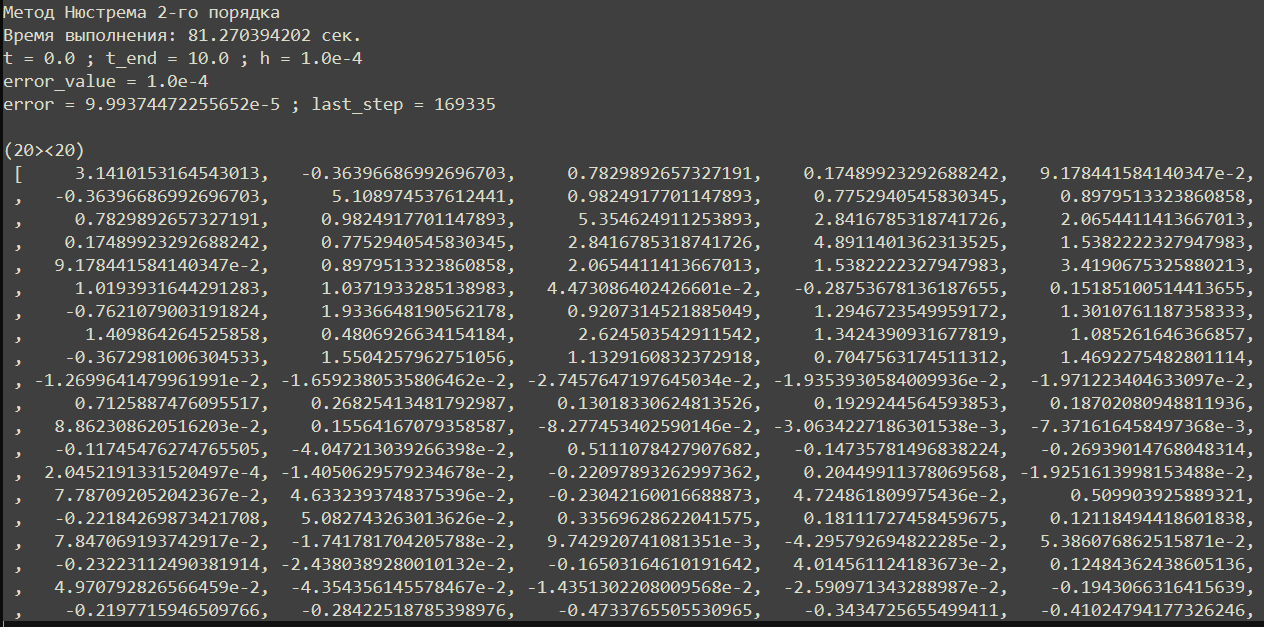


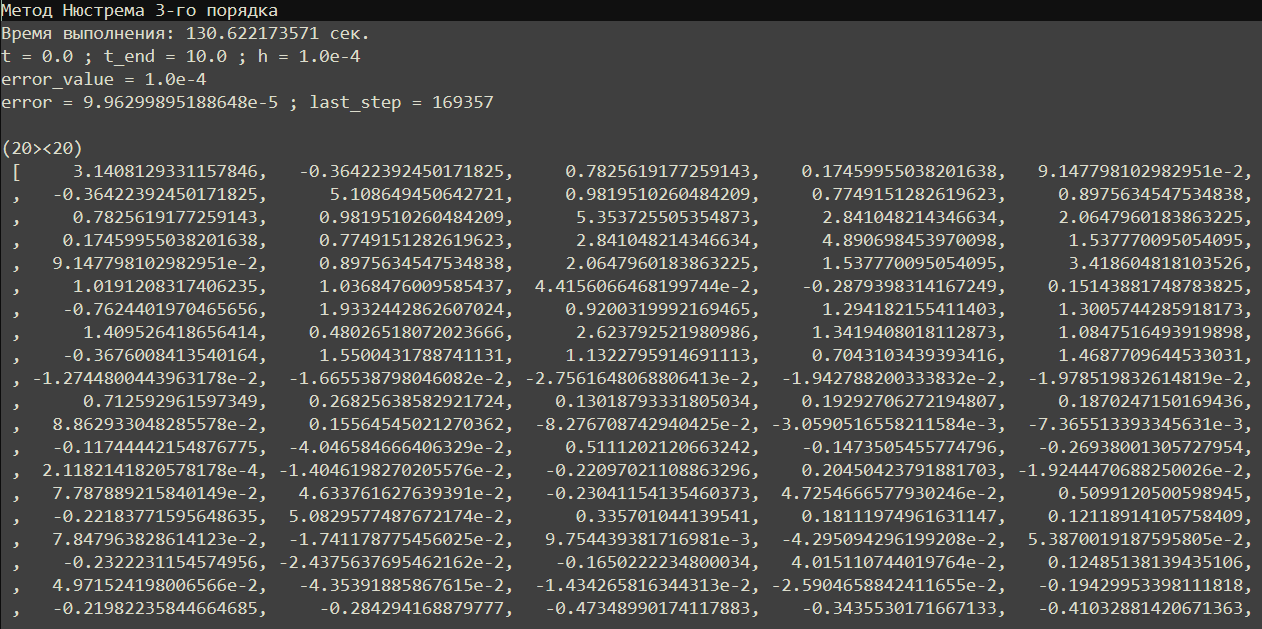
Матрица Q.dat:



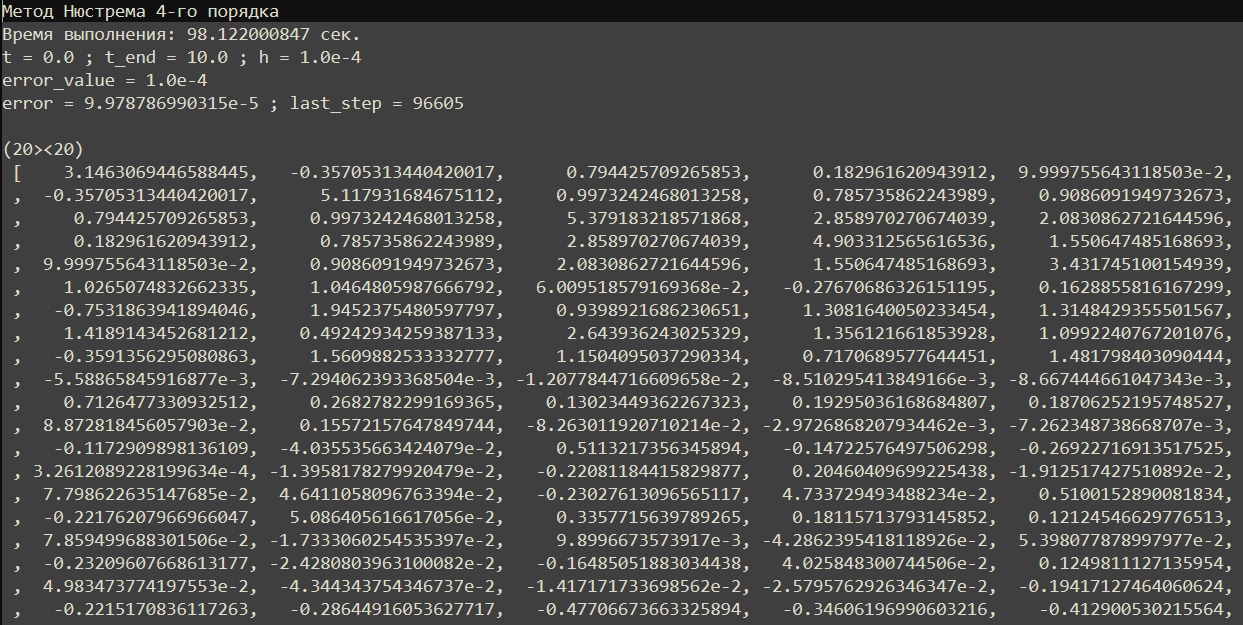
Результат работы по заданным матрицам:

Out2.txt:



Out3.txt:

Out4.txt:



**Выводы**

В ходе проведенной лабораторной работы я разработал программу на языке Haskell для реализации решения матричного уравнения Риккати методом Нюстрема 2-ого, 3-его и 4-ого порядков. Было проведено тестирование отдельных функций и всей программы в целом. Программа полностью удовлетворяет заданным требованиям.

# Исходный код программы

**{-# LANGUAGE OverloadedStrings #-}**

**import qualified Numeric.LinearAlgebra as LA**

**import System.IO**

**import Control.Monad**

**import Data.Time.Clock**

**import Text.Printf**

**import System.Directory (createDirectoryIfMissing)**

**import Prelude hiding (isNaN)**

**-- Типы данных**

**type Matrix = LA.Matrix Double**

**type Vector = LA.Vector Double**

**-- Чтение матрицы из файла**

**readMatrixFromFile :: FilePath -> IO Matrix**

**readMatrixFromFile filename = do**

**contents <- readFile filename**

**let rows = map (map read . words) $ lines contents**

**return $ LA.fromLists rows**

**-- Уравнение Риккати**

**riccatiEquation :: Matrix -> Matrix -> Matrix -> Matrix -> Matrix -> Matrix**

**riccatiEquation e a b q p =**

**(e LA.<> p LA.<> a) + (LA.tr a LA.<> p LA.<> e) + q - (e LA.<> p LA.<> b LA.<> LA.tr b LA.<> p LA.<> e)**

**-- Метод Рунге-Кутты 4-го порядка**

**rungeKutta4 :: (Matrix -> Matrix) -> Double -> Matrix -> Matrix**

**rungeKutta4 f h p = p + LA.scale (h / 6) (k1 + k2 + k2 + k3 + k3 + k4)**

**where**

**k1 = f p**

**k2 = f (p + LA.scale (h / 2) k1)**

**k3 = f (p + LA.scale (h / 2) k2)**

**k4 = f (p + LA.scale h k3)**

**-- Метод Нюстрема 2-го порядка с проверкой на NaN**

**nystrom2 :: (Matrix -> Matrix) -> Double -> Matrix -> Matrix -> Maybe Matrix**

**nystrom2 f h x1 x =**

**let result = x1 + LA.scale (2 \* h) (f x)**

**in if any (any isNaN) (LA.toLists result) then Nothing else Just result**

**where isNaN x = x /= x**

**-- Метод Нюстрема 3-го порядка с проверкой на NaN**

**nystrom3 :: (Matrix -> Matrix) -> Double -> Matrix -> Matrix -> Matrix -> Maybe Matrix**

**nystrom3 f h x2 x1 x =**

**let result = x2 + LA.scale (h/3) (LA.scale 7 (f x) - LA.scale 2 (f x1) + f x2)**

**in if any (any isNaN) (LA.toLists result) then Nothing else Just result**

**where isNaN x = x /= x**

**-- Метод Нюстрема 4-го порядка с проверкой на NaN**

**nystrom4 :: (Matrix -> Matrix) -> Double -> Matrix -> Matrix -> Matrix -> Matrix -> Maybe Matrix**

**nystrom4 f h x3 x2 x1 x =**

**let result = x1 + LA.scale (h/3) (LA.scale 8 (f x) - LA.scale 5 (f x1) + LA.scale 4 (f x2) - f x3)**

**in if any (any isNaN) (LA.toLists result) then Nothing else Just result**

**where isNaN x = x /= x**

**-- Генерация начальных точек с проверкой**

**generateInitialPoints :: (Matrix -> Matrix) -> Double -> Matrix -> Int -> Maybe [Matrix]**

**generateInitialPoints f h p count =**

**foldM (\acc \_ -> let next = rungeKutta4 f h (last acc)**

**in if any (any isNaN) (LA.toLists next)**

**then Nothing**

**else Just (acc ++ [next])) [p] [1..count-1]**

**where isNaN x = x /= x**

**-- Проверка матрицы на NaN и Inf**

**isValidMatrix :: Matrix -> Bool**

**isValidMatrix m = not (any (any isNaN) (LA.toLists m)) && not (any (any isInfinite) (LA.toLists m))**

**where isNaN x = x /= x**

**isInfinite x = abs x > 1e100**

**-- Запись в лог-файл**

**writeLog :: String -> String -> IO ()**

**writeLog method msg = do**

**createDirectoryIfMissing True "logs"**

**let filename = "logs/log" ++ method ++ ".txt"**

**appendFile filename (msg ++ "\n")**

**-- Запись прогресса в лог**

**logProgress :: String -> Double -> Double -> Double -> Double -> Int -> Double -> Matrix -> IO ()**

**logProgress method targetError error t h step tMax p = do**

**let msg = unlines [**

**"\nМетод: " ++ method,**

**"Требуемая ошибка: " ++ show targetError ++ " | Ошибка: " ++ show error,**

**"h: " ++ show h ++ " | Шаг: " ++ show step ++ " | t: " ++ show t ++ " из " ++ show tMax,**

**"P (первые 8x8 элементов):",**

**show (LA.subMatrix (0,0) (min 8 (LA.rows p), min 8 (LA.cols p)) p)**

**]**

**writeLog method msg**

**-- Запись результатов**

**writeResults :: String -> Double -> Double -> Double -> Double -> Double -> Int -> Double -> Matrix -> IO ()**

**writeResults method t0 tMax h targetError error steps execTime p = do**

**createDirectoryIfMissing True "Results"**

**let filename = "Results/out" ++ method ++ ".txt"**

**writeFile filename $ unlines [**

**"Метод Нюстрема " ++ method ++ "-го порядка",**

**"Время выполнения: " ++ show execTime ++ " сек.",**

**"t = " ++ show t0 ++ " ; t\_end = " ++ show tMax ++ " ; h = " ++ show h,**

**"error\_value = " ++ show targetError,**

**"error = " ++ show error ++ " ; last\_step = " ++ show steps ++ "\n",**

**show p**

**]**

**putStrLn $ "\ESC[32mРезультаты метода " ++ method ++ " записаны в " ++ filename ++ "\ESC[0m"**

**-- Измененная функция runNystromMethod (преобразуем время в секунды)**

**runNystromMethod :: Int -> (Matrix -> Matrix) -> Double -> Double -> Double -> Double -> Int -> Matrix -> IO ()**

**runNystromMethod order f t0 tMax h targetError maxSteps p0 = do**

**let method = show order**

**writeLog method $ "Запуск метода Нюстрема " ++ method ++ "-го порядка"**

**startTime <- getCurrentTime**

**let loop step pPrev p = do**

**case generateInitialPoints f h p (order-1) of**

**Nothing -> do**

**writeLog method "Ошибка: получены недопустимые значения (NaN или Inf) при генерации начальных точек"**

**endTime <- getCurrentTime**

**let execTime = realToFrac (diffUTCTime endTime startTime) :: Double**

**writeResults method t0 tMax h targetError (1/0) step execTime pPrev**

**Just initPoints ->**

**let methodResult = case order of**

**2 -> case initPoints of**

**[x1] -> nystrom2 f h x1 p**

**\_ -> error "Недостаточно точек для метода 2-го порядка"**

**3 -> case initPoints of**

**[x2, x1] -> nystrom3 f h x2 x1 p**

**\_ -> error "Недостаточно точек для метода 3-го порядка"**

**4 -> case initPoints of**

**[x3, x2, x1] -> nystrom4 f h x3 x2 x1 p**

**\_ -> error "Недостаточно точек для метода 4-го порядка"**

**\_ -> error "Неподдерживаемый порядок метода"**

**in**

**case methodResult of**

**Nothing -> do**

**writeLog method "Ошибка: получены недопустимые значения (NaN или Inf)"**

**endTime <- getCurrentTime**

**let execTime = realToFrac (diffUTCTime endTime startTime) :: Double**

**writeResults method t0 tMax h targetError (1/0) step execTime pPrev**

**Just newP ->**

**if not (isValidMatrix newP)**

**then do**

**writeLog method "Ошибка: получены недопустимые значения (NaN или Inf)"**

**endTime <- getCurrentTime**

**let execTime = realToFrac (diffUTCTime endTime startTime) :: Double**

**writeResults method t0 tMax h targetError (1/0) step execTime pPrev**

**else do**

**let error = LA.norm\_Inf (newP - pPrev)**

**-- Логирование прогресса**

**logProgress method targetError error t0 h step tMax newP**

**-- Условия выхода**

**if step >= maxSteps || error <= targetError || error > 1e10**

**then do**

**endTime <- getCurrentTime**

**let execTime = realToFrac (diffUTCTime endTime startTime) :: Double**

**writeResults method t0 tMax h targetError error step execTime newP**

**else loop (step+1) p newP**

**loop 0 p0 p0**

**-- Основная функция**

**main :: IO ()**

**main = do**

**-- Чтение матриц**

**e <- readMatrixFromFile "input/E.dat"**

**a <- readMatrixFromFile "input/A.dat"**

**b <- readMatrixFromFile "input/B.dat"**

**q <- readMatrixFromFile "input/Q.dat"**

**-- params**

**let t0 = 0.0**

**tMax = 10.0**

**h = 0.0001**

**targetError = 1e-4**

**maxSteps = 300000**

**f = riccatiEquation e a b q**

**p0 = LA.konst 0 (LA.rows e, LA.cols e)**

**-- Создаем папки если их нет**

**createDirectoryIfMissing True "logs"**

**createDirectoryIfMissing True "Results"**

**-- Запуск всех методов**

**runNystromMethod 2 f t0 tMax h targetError maxSteps p0**

**runNystromMethod 3 f t0 tMax h targetError maxSteps p0**

**runNystromMethod 4 f t0 tMax h targetError maxSteps p0**

**putStrLn "\nВсе методы выполнены. Результаты сохранены в папках logs/ и Results/"**