**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики и информатики**

**Кафедра информационных систем управления**

**Отчет**

**По методам вычислений**

Выполнил студент группы № 12

*Шишлянников Иван Викторович*

Минск 2020

# Оглавление

[Условие согласно варианту 3](#_Toc41242372)

[Генерация матрицы 3](#_Toc41242373)

[Степенной метод 4](#_Toc41242374)

[Первый случай: 4](#_Toc41242375)

[Второй случай: 5](#_Toc41242376)

[Третий случай 5](#_Toc41242377)

[Выбор индекса j. 6](#_Toc41242378)

[Критерий остановки: 6](#_Toc41242379)

[Метод обратной итерации 7](#_Toc41242380)

[QR – Алгоритм 7](#_Toc41242381)

[Выводы по полученным результатам 8](#_Toc41242382)

[Решение уравнений 8](#_Toc41242383)

[Графическое отделение корней. 8](#_Toc41242384)

[Сужение отрезка методом бисекции 11](#_Toc41242385)

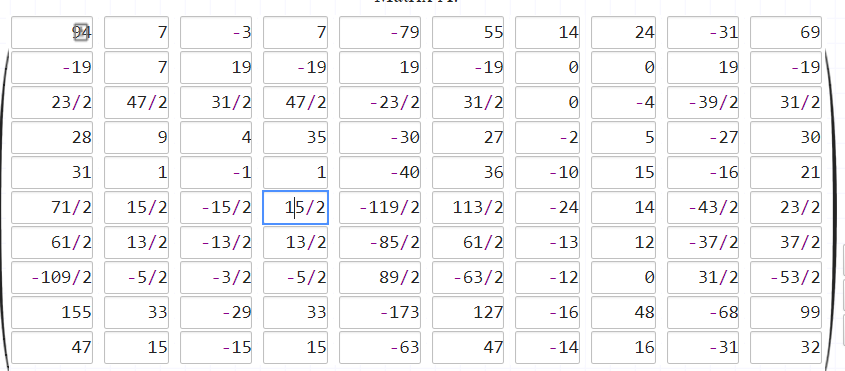
[Анализ полученных результатов. Сравнение сходимостей. Выводы 12](#_Toc41242386)

[Комментарии по поводу выполненной работы. 14](#_Toc41242387)

# Условие согласно варианту

N = 10;

Матрица



Вычисляемая матрица записывается в файле, котором выведена вся информация по ее решению.

Функция:



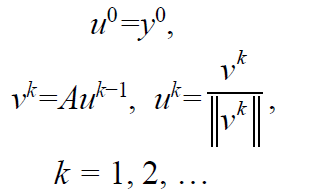
# Генерация матрицы

Для генерации чисел использовал стандартную библиотеку C# Random.

Для получения необходимой точности вся работа велась с типом double так как данный тип данных дает нам точность 15-17 цифр.

# Степенной метод

Для нормированных вычислений будем параллельно каждую итерацию заполнять последовательности v и u по следующим формулам:

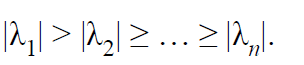


В качестве нормы возьмем кубическую норму.

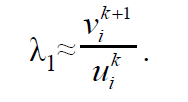
Изначально я вычислял без нормирования, но тогда матрицы 10x10 уже не сходились из-за переполнения при высокой точности (e > 0.001).

При вычислении минимальных/максимальных собственных значений мы можем встретить 3 различных случая сходимости.

## Первый случай:

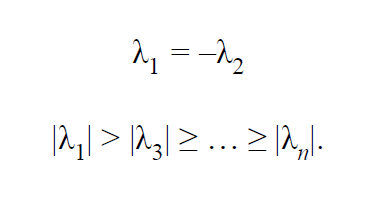


В данном случае собственное значение и соответствующий ему собственный вектор находятся по следующим формулам соответственно:

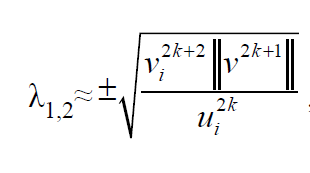


Собственный вектор = u^k;

## Второй случай:



В данном случае наша последовательность будет разбита на две последовательности (yk и y2k) каждая из которых стремится к своему собственному значению. Сходимость к одному из двух собственных значений в таком случае происходит через одну итерацию. Тогда собственные значения вычисляется по формуле:

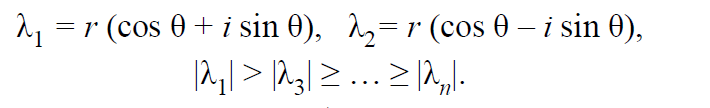


Собственный вектор, соответствующий собственному значению λ1, приближенно равен 

Собственный вектор, соответствующий собственному значению (*−*λ1), приближенно равен

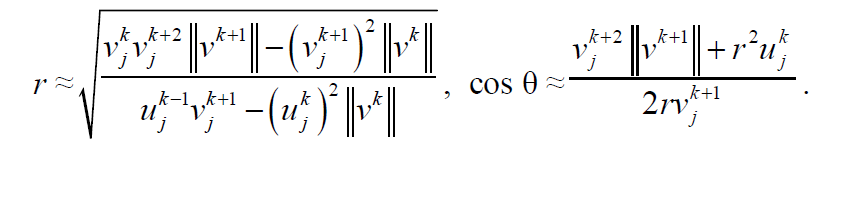


## Третий случай



Здесь для нахождения собственных значений нам понадобится последовательность из 4 векторов.

Собственные значения считаются по следующей формуле:



Если собственные значения вещественной матрицы комплексно-сопряженные, то и отвечающие им собственные векторы имеют комплексно-сопряженные координаты.

Собственный вектор, соответствующий собственному значению λ1, приближенно равен:



Собственный вектор, соответствующий собственному значению λ2, приближенно равен

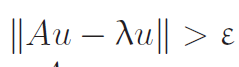


### Выбор индекса j.

При выборе индекса j для выбора элемента векторов последовательностей v и u, я всегда беру индекс максимального элемента текущего вектора. Делаю я это для того чтобы избежать работы с близкими или равными нулю элементами, тем самым повышая точность вычислений. (если взятый элемент окажется равен нулю, тогда мы вообще можем получить NaN при вычислении последующих элементов последовательностей и собственных значений, тем самым поставив крест на сходимости наших итераций). Я с такой проблемой столкнулся только когда в матрице были чистые нули, но высока вероятность, что данная проблема появится и при близких к нулю значениях.

## Критерий остановки:

Критерием итерации служит проверка:



Данный критерий вытекает из основного свойства собственного значения и собственного вектора.



То есть мы будем выполнять итерации пока разность между левой и правой частью данного равенства не будет меньше определенного эпсилон.

На каждой итерации я, так как заранее не знаю с каким случаем столкнулся, получаю по соответствующим формулам (описанных в описании случаев) получаю пары собственное значение – вектор. И проверяю по критерию остановки какие пары соответственно подходят, и тогда завершаю итерацию (в реализации алгоритма я проверяю соответствие критерию у пар значение-вектор в том порядке, в котором я написал список случаев. То есть случай 3 проверится в самую последнюю очередь. Хотя при случайной генерации матриц часто преобладал именно 3 случай, поэтому его проверку следовало бы поставить в начало, но так как все операции в проверках не требуют дополнительного перебора (то есть выполняются за O(1)), то данная модификация слабо бы повлияла на скорость алгоритма).

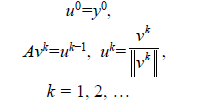
Точность подбиралась экспериментально. Проанализировав процесс сходимости моего алгоритма, я заметил, что значение точнее чем при эпсилон равном 10^-10 получить не получается. Процесс просто не сходится. Либо сходится, но очень медленно.

## Метод обратной итерации

Если нам нужно найти минимальное по модулю собственное значение матрицы A, то нам достаточно применить степенной метод к обратной матрице A^-1. Так как по свойству собственных значений собственные значения обратных матриц так же обратны, а значит 1/ λмакс матрицы A^-1 соответствует λмин матрицы A. Для того чтобы алгоритм был более эффективным, я избавился от необходимости искать обратную матрицу. Достаточно каждую итерацию решать СЛАУ



То есть нормированные вычисления выглядят вот так:



Так же для более быстрого решения СЛАУ я сделал LUP разложение матрицы A.

С таким алгоритмом мы не только понижаем сложность с точностью до константы, но и избавляем себя от риска сильно неточных вычислений обратной матрицы A при наличии у A большого числа обусловленности.

Так же есть вероятность работать с вырожденной матрицей. В таком случае при попытке найти минимальное собственное значение, сразу после LU разложения утвердить, что минимальное значение равно нулю и решить СЛАУ вида Ax = zeros для нахождения собственного вектора. То есть за одну итерацию сразу получаем ответ. В моем случае я нахожу сначала LUP – разложение, а потом только на первой итерации решаю СЛАУ, но можно делать эти две операции сразу тем самым сразу после разложения LU возвращать ответ.

# QR – Алгоритм

Для нахождения всех собственных значений одновременно используется QR алгоритм. Одним из способов его ускорения является начальное представление матрицы A в форме Хесенберга. Тогда сложность очередного QR разложения становится на порядок ниже ( O(n^2)). Но так как входные матрицы не обладают большими размерами, то я пренебрег данным ускорением и работал сразу с “грязной” матрицей. С матрицами большего размера работать так как я ни в коем случае нельзя.

Точность для QR алгоритма оставил такой же как и для степенного, но стал замечать что в среднем каждая 500-ая матрица вызывает у qr алгоритма затруднения, поэтому точность пришлось уменьшить на 1 порядок ( 1\*10^-9).

В качестве критерия остановки я выбрал норму разности векторов полученных собственных значений шагов k и k-1.

Алгоритм возвращает вектор собственных значений.

# Выводы по полученным результатам

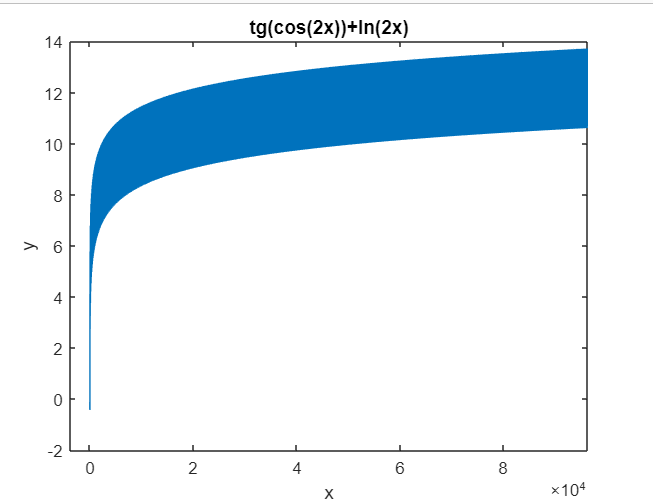
Проанализировав полученную статистику по 1000 сгенерированным матрицам делаю следующие выводы.

1. Среднее время нахождения корня степенным методов примерно в 50 раз меньше чем время нахождения всех корней QR алгоритмом без ускорений. То есть нахождение отдельного корня степенным методов в 5 раз быстрее чем QR алгоритмом. Если положить, что среднее количество итераций qr алгоритма не меняется для большой выборки сгенерированных матриц, то ускорение при помощи формы Хесенберга как раз бы сократило один порядок и мы бы получили равную скорость нахождения отдельных собственных значений независимо от метода.
2. Количество итераций необходимое на нахождение максимальных и минимальные собственных значений очень рознятся на несколько порядков. То есть если на нахождение минимального значения нужно около 200 итераций, то на нахождение максимального значения потребуется не меньше 10. И наоборот. Это может говорить о том, что разница между максимальным и минимальным значениями достаточно велика, и что начальное приближение всегда значительно ближе к какому – то из крайних собственных значений.
3. Минимальное и максимальное время нахождения собственных значений степенным методом сильно разнится, хотя среднее время намного ближе к минимальному. Это значит, что крайне редко у сгенерированной матрицы собственные значения почти равны. Данный случай сходится так же, как и [случай 1](#_Первый_случай:) без изменений алгоритма. Но сходиться данный случай может очень долго

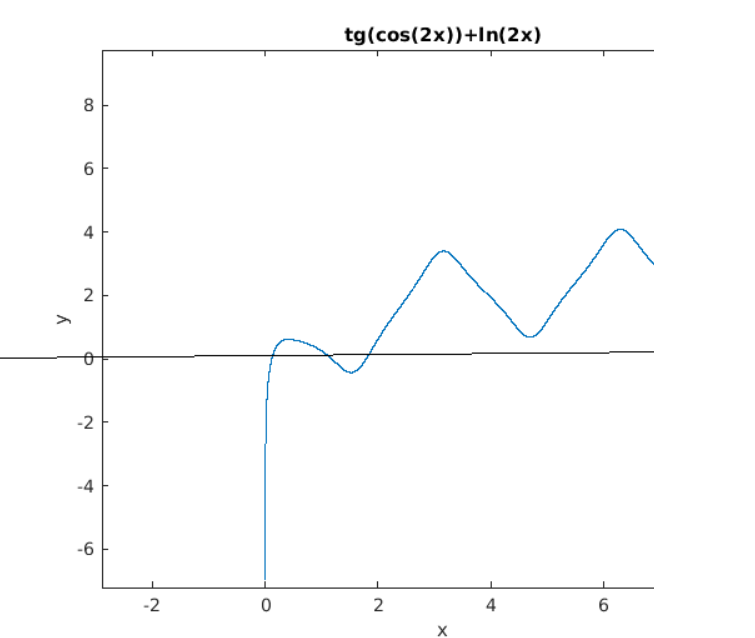
# Решение уравнений

## Графическое отделение корней.

Для начала посмотрим на большую часть графика целиком



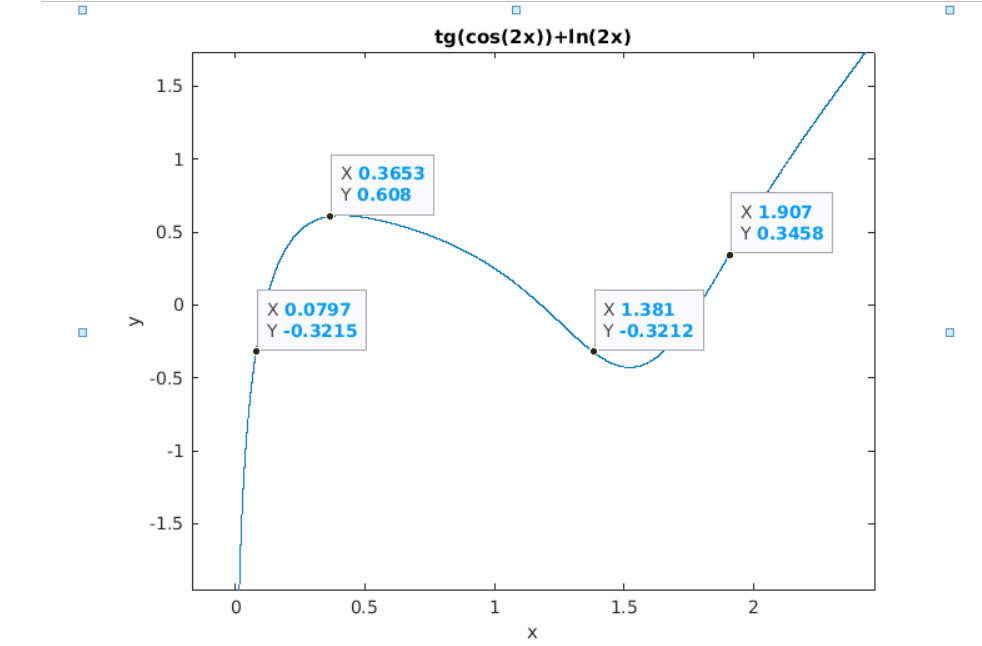
Думаю, здесь стало очевидно, что наши решения находятся где-то в самом начале. Изменим маштаб чтобы рассмотреть корни повнимательнее.



Как видим. У нашей функции 3 корня.

Их ровно 3 так как наша прямая пересекает график только 3 раза. А на предыдущем рисунке было четко видно, что график идет всегда вверх, поэтому кроме этих корней больше быть не может.

Для уточнения координат приблизим еще больше.



Таким образом мы получили примерные границы допустимых значений наших корней.

Как видим, никакие две точку подряд не обладают одним и тем же знаком значения y. Значит данными точками мы покрыли все зоны, где наш график пересекает ось OY, и на каждом промежутке корень единственный.

Для удобства округлим координаты так, чтобы минимально отдалиться от установленных границ (чтобы случайно не потерять корень из зоны).

x1 – [0.08; 0.37]

x2 – [0.37; 1.38]

x3 – [1.38; 1.9]

## Сужение отрезка методом бисекции

Сужать будем до размера 10^-5.

Алгоритм стандартный.

Полученные результаты:

x1 = [ 0,1148516845703125 0,11486053466796875 ]

x2 = [ 1,1796829223632812 1,1796906280517576 ]

x3 = [ 1,7896142578124996 1,7896221923828122 ]

Далее вся работа будет вестись с прибилижениями выбранными с левых краев данных промежутков.

Для проверки на сходимость при определенном начальном приближении проверялось следующее условие:

f(x0) \* f''(x0) > 0

Все три приближения прошли проверку, значит метод Ньютона сходится при данных приближениях.

В качестве критерия остановки любого из методов Ньютона я использовал неравенство

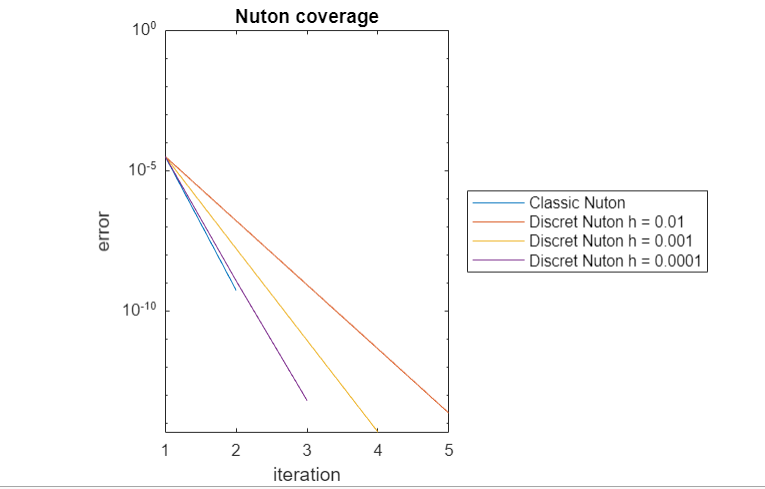
|xk+1-xk|< = eps

## Анализ полученных результатов. Сравнение сходимостей. Выводы

Проанализировав полученные результаты можно сделать следующие выводы:

1. Метод ньютона сходится всегда намного быстрее чем дискретный (при допустимом начальном приближении). Но получить результат точнее чем при eps = 10^-17 в моем случае не получается. Точнее, после достижения данного предела, разность между xk+1 и xk равна нулю (f(x) возвращает 0). Поэтому более точной картины для сравнения сходимости я не получил.
2. Рассмотрим диаграмму сходимости различных методов Ньютона.

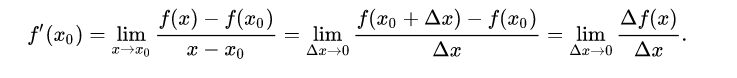
!!! На всех графиках будем делать 5 итераций. Если метод на какой-то итерации сошелся к нулю, то прямая соответствующая данному методу на графике просто закончит свое рисование. То есть это значит, что на данной итерации ошибка стала равно 0. (Пример: На данном графике Classic Nuton сошелся за 2 итерации).



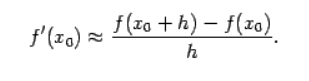
Как видим, Классический метод Ньютона сходится всегда быстрее чем его Дискретная модификация. Чем меньше (по модулю) шаг мы возьмем тем быстрее будет сходиться метод. Но в лучшем случае он Дискретный метод просто приблизится максимально по скорости к Классическому, так как классический метод Ньютона регулирует точность вычисляя производную каждую итерацию, а дискретный метод использует только некоторое приближение.

Ускорение вычислений при уменьшении h следует из смысла числового дифференцирования.

То есть так как производная вычисляется всегда по следующей формуле:

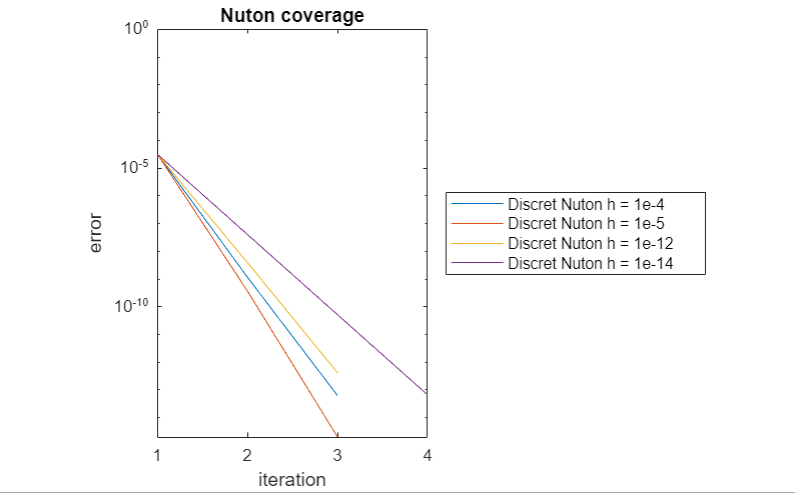


То если взять h -> 0, то мы как раз и получим близкое к реальному значение производной в нужной точке и формула примет вид:



Но есть один важный момент. При числовом дифференцировании важно понимать, что компьютер может сильно ухудшить результаты вычислений в силу своих возможностей при сильно низком значении h. То есть станут возможны делания на 0 и другие эффекты работы с очень маленькими числами.

Вот пример как будет выглядеть сходимость при сильно маленьких h:



Как видим, при уменьшении h до определенной границы (1e-5) скорость сходимости все еще растет, но если мы еще уменьшим h (1e-12), то мы видим, что скорость сходимости снова начинает падать, хотя должна была бы еще вырасти. То есть для получения достаточно точных результатов надо внимательно изучать данную нелинейную функцию и следить за потенциальными переполнениями при сильно малых значениях h.

Данная проблема наблюдалась и при h>e-12, но для наглядности были выбраны именно такие значения.

# Комментарии по поводу выполненной работы.

Графики были построены в matlab. Данные для графиков получались из моей программы.

Для проверки правильности вычислений использовались matlab и wolfram alpha.

К отчету приложены два txt файла. LabResults и MatrixSolve. В первом файле хранится отчет о проделанных задачах. Во втором хранится результат работы с матрицей из условия. Там же и выведена матрица с которой я работаю.

Весь рабочий код программы написан на C#. Для работы с комплексными числами использовался готовый класс Complex. Но для правильной работы программы в силу строгой типизации языка, многие методы пришлось продублировать для комплексного поля. Для удобства, работа степенного метода всегда идет над комплексным полем. Данное решение заставляет нас всегда хранить комплексную пару, но позволяет избежать постоянного конвертирования из одного поля в другое (R->C).

Результаты всех тестов были получены при сборке программы в режиме Release.