# Лабораторная работа №2 по предмету Методы Вычислений

Автор: Томашевич Константин, Вариант 7.

## Часть 1. Генерация матрицы.

Для генерации матрицы со случайным элементами я использовал тот же генератор Mersenne Twister, что и в первой лабораторной работе. Использовал я его по тем же причинам и добавил при генерации тот же костыль, что и в первой лабораторной работе. Копировать всё слово в слово из предыдущего отчёта мне не хочется, а перефразировать лень.

## Часть 2. Степенной метод.

Сразу отмечу, что далеко не для всех матриц получилось удачно применить степенной метод, так как в некоторых случаях сходимость оказывалась настолько медленной, что мне надоедало ждать (больше секунд). Поэтому я ограничил максимальное число итераций как . При этом как эпсилон сходимости использовалось . При прогоне метода на запусков со случайными матрицами были получены следующие результаты:

|  |  |
| --- | --- |
| Минимальная евклидова норма ошибки | 0.0000000000000003 |
| Максимальная евклидова норма ошибки | 75.6064786021455150 |
| Средняя евклидова норма ошибки | 11.3263001962471800 |
| Среднее время выполнения | 9мс |

Под евклидовой нормой ошибки понимается .

Как видно из данной таблицы, в некоторых случаях случился откровенный \_\_\_\_\_\_ (впишите свой вариант). Все случаи с настолько большой нормой ошибки останавливаются на максимальном числе итераций, то есть не успевают сойтись в собственное значение. Точно анализировать запусков мне было лень, поэтому просто предположу, что приблизительно в случаев алгоритм нормально сходится, но вот оставшиеся матрицы оказываются камнем преткновения. Впрочем, ничего необычного для «наивных» алгоритмов с простым математическим аппаратом.

Второе собственное значение я получал следующим образом:

1. Вычислял матрицу .
2. Вычислял максимальное по модулю собственное значение матрицы степенным методом.
3. .
4. Полученный при вычислении собственный вектор .

Вычисление второго собственного значения таким образом нестрого подтвердило гипотезу выше: в большинстве случаев, когда первое собственное значение сходится плохо, второе собственное значение сходится с довольно хорошей нормой ошибки и не достигая максимального числа итераций. Это происходит из-за того, что у матриц, на которых метод сходится медленно, часто несколько собственных значений лежат около максимального. Тогда когда мы отнимем первое найденное (хоть и криво) собственное значение, эти собственные значения будут околонулевыми и не будут мешать методу. Тогда поиск второго собственного значения не будет сходится хорошо только тогда, когда около его также будут находиться несколько других собственных значений, что довольно маловероятно.

## Часть 3. QR-алгоритм.

При реализации QR-алгоритма были использованы следующие константы:

* Барьер дефляции – числа меньше по модулю этой константы будут считаться нулями.Взят как .
* Барьер сходимости : . Где – собственные значения, полученные на новой итерации, а – полученные на предыдущей. Был взят как .

Ограничение максимального количества итераций не применялось. При прогоне метода на запусков со случайными матрицами были получены следующие результаты:

|  |  |
| --- | --- |
| Среднее количество итераций | 1918 |
| Среднее время выполнения | 25мс |

Как можно заметить, среднее количество итераций и время выполнения великовато. Конечно результат всё ещё значительно лучше, чем в степенном методе. Всё-таки мы ищем все значения, да и с точностью тут у нас проблем нет.

Попробуем разобраться в чём проблема. Прокрутив лог с результатов я заметил, что многие матрицы решались QR-алгоритмом за 300-500 итераций. Также при выполнении были заметны «подвисания» на несколько секунд на некоторых матрицах. Решив простую систему линейных уравнений мы получим, что присутствие матриц, решение которых заняло итераций, среди матриц, решаемых за итераций, приведёт к отклонению среднего количества итераций как раз к близкому к полученному значению. То есть в большинстве случаев QR-алгоритм будет сходиться в раз быстрее, чем в среднем.

Также хочу отметить огромную зависимость времени выполнения алгоритма от значения барьера сходимости. Так при на запусков со случайными матрицами выполнение QR-алгоритма в среднем занимало всего 6мс, что в раза быстрее степенного алгоритма. Из всего этого можно сделать вывод, что QR-алгоритм позволяет достаточно быстро найти все собственные матрицы с заданной точностью. Если её, конечно, не задавать как астрономические . В любом случае, QR-алгоритм находит все собственные значения, однозначно сходиться за приемлемое время (меньше секунд) на любых случайных матрицах и даёт достаточно точные результаты. Всё это делает его однозначно более удобным для большинства задач, чем степенной метод.

## Часть 4. Анализ графика функции.

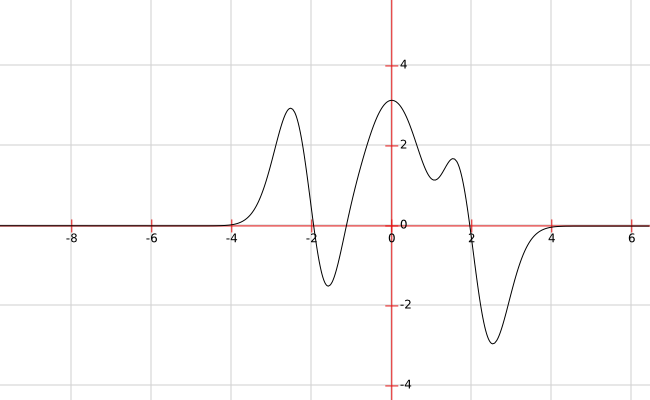


Рисунок 1. Общий вид графика.

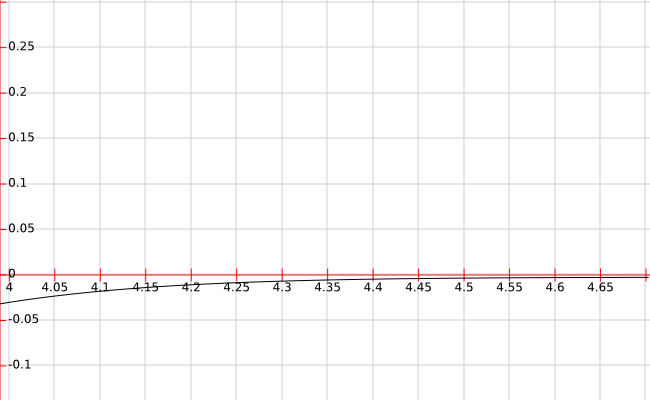


Рисунок 2. Правая ветвь.

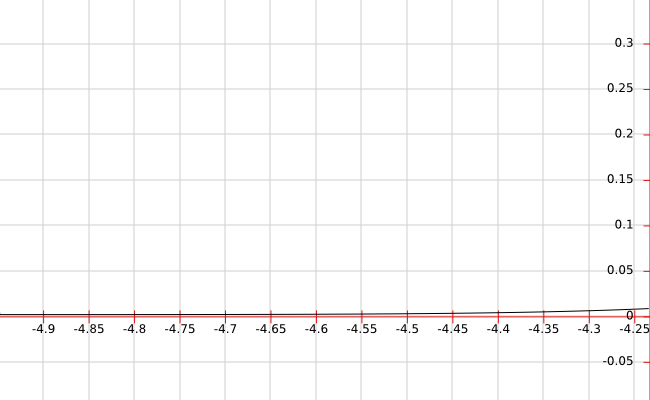


Рисунок 3. Левая ветвь.

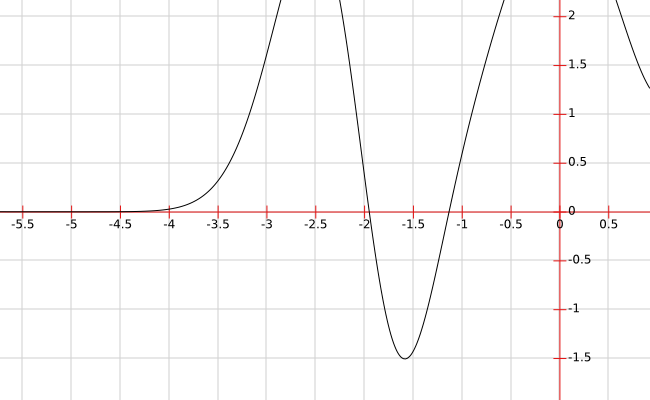


Рисунок 4. Ещё одно приближение.

Из следующих графиков явно видны следующие отрезки корней:

Заметим также, что функция не имеет точек разрыва, поэтому данный график позволяет с уверенностью заявлять об отсутствии других корней на промежутке На приближениях левой и правой ветви чёрная линия (график) переходит в прямую немного выше (слева) и немного ниже (справа) оси Х – это условное обозначение построителя для чрезвычайно малых, но не пересекающих ось, прямых. Так нарисовали, короче.

Убедимся, что отсутствуют корни на промежутке . Не трудно заметить, что первое слагаемое функции стремится к нулю и очень мало по модулю уже при , из чего не сложно сделать вывод, что на исследуемом бесконечном промежутке функция сходится к , которая не имеет корней на исследуемом промежутке. Profit.

## Часть 5. Метод биекций.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Левая граница до | Левая граница после | Правая граница до | Правая граница после |
| 1 | -2 | -1.9519531250000002 | -1.6 | -1.9518554687500003 |
| 2 | -1.4 | -1.1358398437499999 | -1 | -1.1357421874999998 |
| 3 | 1.7 | 1.9518554687500003 | 2.1 | 1.9519531250000002 |

Как можно легко подтвердить, все три полученных отрезка теперь имеют ширину меньше . Во всех трёх случаях метод бисекций сделал 12 шагов. С математической точки зрения так и должно быть, так как длина каждого отрезка в начале была , а . Максимальная точность, которую мы может «выжать» из double в данном случае . Для достижения этой точности нам сделать 52 шага методом биекций, то есть на 40 шагов больше. Не так уж и плохо, но теоретически можно лучше. Получив отрезок длины можно уже использовать метод с более сложными условиями, который сходился бы быстрее. Метод Ньютона, короче.

## Часть 6. Дискретный метод Ньютона.

|  |  |  |
| --- | --- | --- |
| # | Количество шагов | Получившийся корень |
| 1 | 5 | -1.9519125752999189 |
| 2 | 4 | -1.1357567044695389 |
| 3 | 5 | 1.9519145740760813 |

Во всех трёх случаях дискретный метод Ньютона принимал за правую границу отрезка. Также осуществлялась проверка на выход за границы отрезка, но, к счастью, во всех случаях этого не происходило. Константа бралась равной , алгоритм останавливался на шаге при условии .

Как видно из таблицы, дискретному методу Ньютона понадобилось сделать в среднем в 8 раз меньше шагов, чем пришлось бы методу бисекций, для достижения точности . Конечно следует заметить, что ограничения метода Ньютона делают его применимым только на достаточно малых отрезках. Из этого можно сделать вывод, что наиболее эффективным методом поиска корня является сжатие отрезка методом бисекций до размера с последующим применением дискретного метода Ньютона для достижения точности

## Часть 7. Точный метод Ньютона.

Сразу отмечу, что улучшить с помощью точного метода Ньютона полученные дискретным метод Ньютона корни у меня не получилось. Я брал за полученный дискретным методом Ньютона корень, считал точным методом Ньютона и в результате получал, что . То есть для точности ничего уже не менялось, что означает невозможность улучшения корня при хранении переменной в формате double. Следуя из этого результата я решил отдельно провести подсчёт корней из полученных методом бисекций отрезков точным методом Ньютона:

|  |  |  |
| --- | --- | --- |
| # | Количество шагов | Получившийся корень |
| 1 | 4 | -1.9519125752999189 |
| 2 | 3 | -1.1357567044695389 |
| 3 | 3 | 1.9519145740760813 |

Как не трудно заметить, корни у нас получились те же самые. Но вот количество шагов заметно уменьшилось. Из этого следует, что, как минимум в арифметике с плавающей точкой С, дискретный метод Ньютона с большой вероятностью даст те же корни, что и точный метод Ньютона. Но точный метод Ньютона, благодаря использованию точной производной, сойдётся быстрее.

## Использованная литература

