# Лабораторная работа №1 по предмету Методы Вычислений

Автор: Томашевич Константин.

## Часть 1. Анализ графика функции.

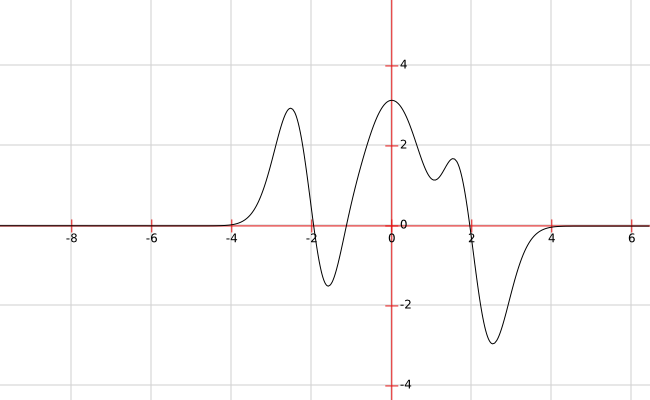


Рисунок 1. Общий вид графика.

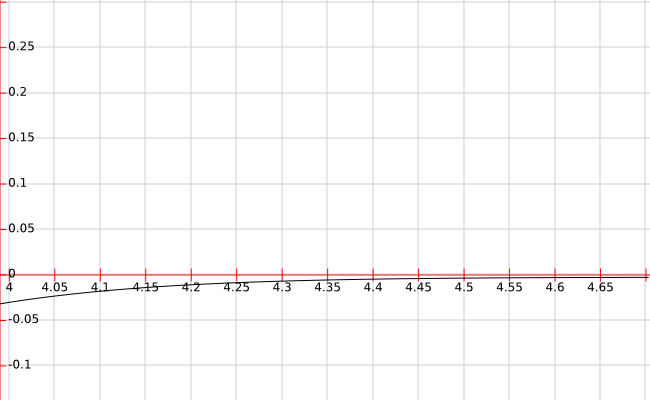


Рисунок 2. Правая ветвь.

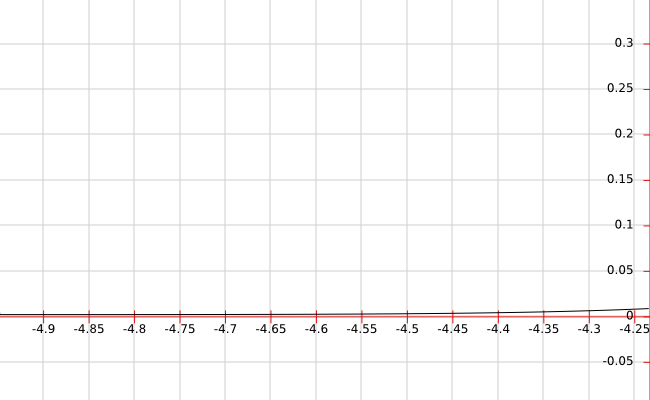


Рисунок 3. Левая ветвь.

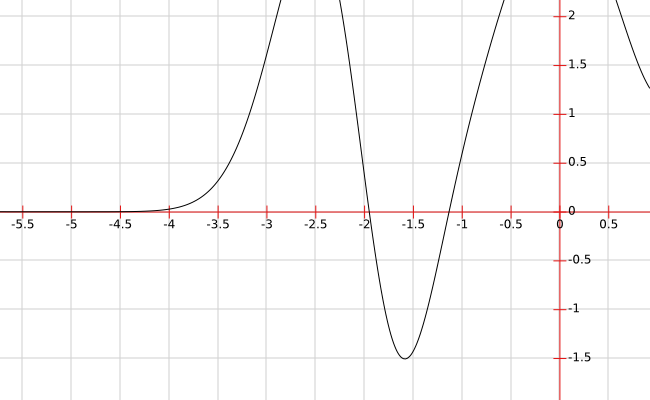


Рисунок 4. Ещё одно приближение.

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 2. 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 3. 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 4. Точный метод Ньютона.

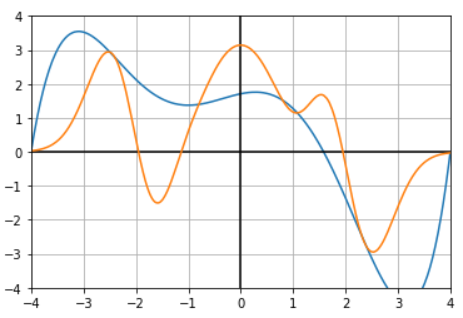
Сразу отмечу, что улучшить с помощью точного метода Ньютона полученные дискретным метод Ньютона корни у меня не получилось. Я брал за полученный дискретным методом Ньютона корень, считал точным методом Ньютона и в результате получал, что . То есть для точности ничего уже не менялось, что означает невозможность 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.

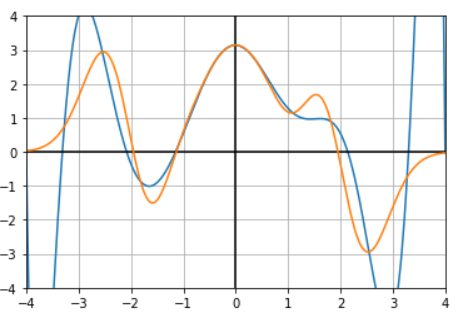
## Часть 5. Интерполяция многочленами Ньютона по равноотстоящим узлам.

График многочлена по 6 равноотстоящим точкам (синий) и функции (оранжевый):



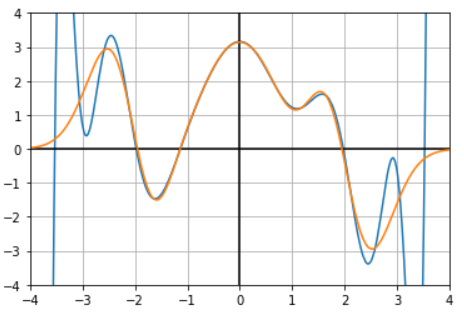
Выглядит немного не очень. Или совсем не очень. Но чего ждать от многочлена по 6 точкам? Видно, что как минимум в 6 точках он совпадает с функций. Большего ждать и не стоило, мне кажется.

График многочлена по 12 равноотстоящим точкам (синий) и функции (оранжевый):



Многочлен по 12 точкам уже намного точнее повторяет функцию, в центре полностью с ней совпадая, что очень даже и неплохо. Только вот по краям творится жесточайшая дичь, но надо же чем-то жертвовать для совпадения центра.

График многочлена по 18 равноотстоящим точкам (синий) и функции (оранжевый):

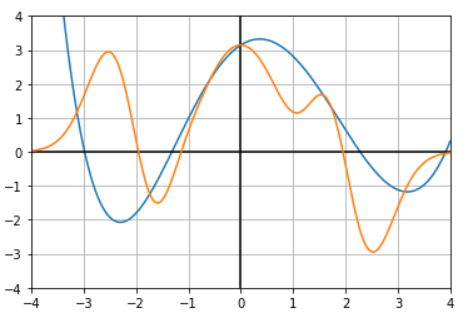


Многочлен по 18 точкам практически полностью совпадает с функцией на половине промежутка, что очень даже круто. На остальной части промежутка при этом творится трэш, угар и содомия. Чем-то напоминает шутки в стиле «Москва и остальная часть России».

Вывод: если нужно получить хорошее совпадение многочлена с исходной функцией, нужно брать достаточно много узлов и промежуток должен быть в раза в два больше нужного, так как по краям будет происходить адский ад.

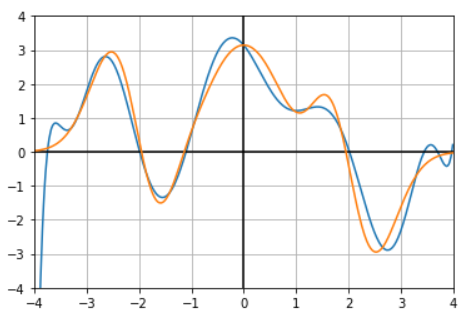
## Часть 6. Интерполяция многочленами Ньютона по узлам Чебышёва.

График многочлена по 6 узлам (синий) и функции (оранжевый):



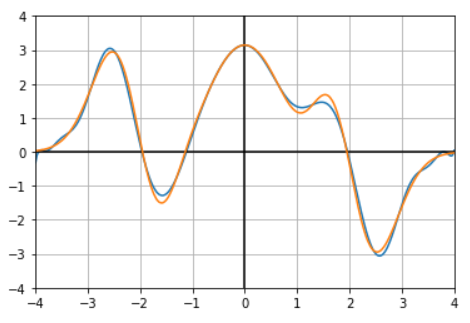
В отличии от многочлена по 6 равноотстоящим точкам, многочлен по 6 узлам Чебышёва даже немного напоминает функцию в центре, что однозначно является успехом.

График многочлена по 12 узлам (синий) и функции (оранжевый):



Многочлен по 12 узлам Чебышёва уже очень даже неплохо мимикрирует исходную функцию, только начинает немного беситься по краям. Тем не менее этот результат однозначно намного лучше многочлена по 12 равноотстоящим узлам.

График многочлена по 18 узлам (синий) и функции (оранжевый):

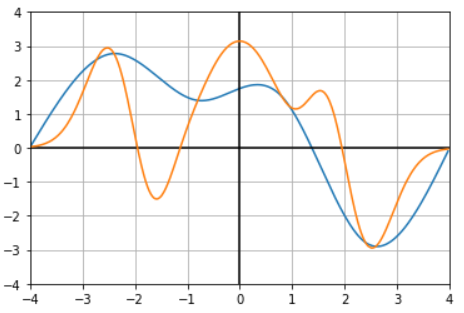


Многочлен по 18 узлам Чебышёва очень точно изображает исходную функцию. При этом по краям его колбасит совсем чуть-чуть, в отличии от такого же по равноотстоящим узлам.

Вывод: использование узлов Чебышёва сделало интерполирование многочленом Ньютона значительно более точным, чем оно было при использовании равноотстоящих точек. При использовании узлов Чебышёва можно делать интерполяцию на нужном промежутке (нет необходимости добавлять «уши»), и хороший результат при этом достигается быстрее. Но следует отметить один минус – узлы Чебышёва обычно делают рассчёт многочлена более сложным, так как операции с ними занимают значительно больше времени, чем операции с равноотстоящими точками. Это происходит из-за количества цифр после запятой. В зависимости от структуры компилятора и добавляемым им оптимизаций, рассчёт многочлена с узлами Чебышёва может занять больше на 10%-1000% (именно такой разброс замечен между уровнем оптимизаций O3 и O0 компилятора GCC на 2000 узлов) времени.

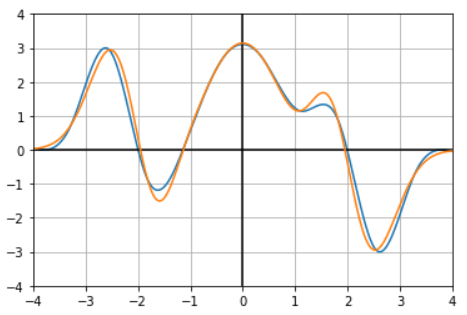
## Часть 7. Интерполяция кубическими сплайнами.

Кубический сплайн по 6 узлам (синий) и функция (оранжевый):



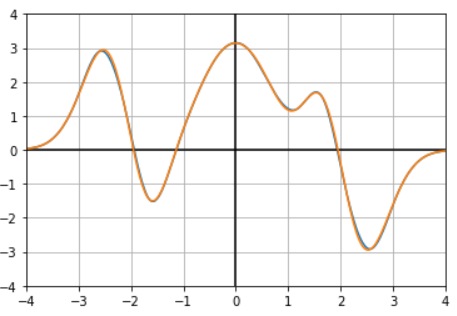
Как можно видеть из графика выше, 6 равноотстоящих точек оказалось явно недостаточно для хорошего совпадения сплайна с функцией. Но, как показали предыдущие методы, 6 точек это действительно мало.

Кубический сплайн по 12 узлам (синий) и функция (оранжевый):



На 12 точках сплайн уже почти полностью совпадает с функцией, обгоняя в этом даже многочлен Ньютона на узлах Чебышёва. Очень даже хороший результат.

Кубический сплайн по 18 узлам (синий, если его вообще видно) и функция (оранжевый):

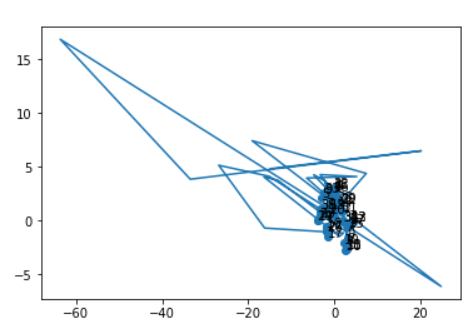


На 18 точках отличия вообще очень сложно заметить, совпадение почти 100%-ное.

Вывод: кубический сплайн даёт чрезвычайно хорошее совпадение при достаточном количестве узлом. При этом, как ни странно, он ещё значительно быстрее! Дело в том, что оптимизированный метод построения кубического сплайна работает за O(N), где N – количество узлов, а многочлен Ньютона строится за O(), что существенно медленнее. Для примера: многочлен Ньютона по 1000 узлов Чебышёва был построен за 3.1мс, а кубический сплайн по 1000 узлов всего за 0.2мс! Заметим, что в приведённом выше эксперименте сыграло роль также выделение памяти – построение многочлена Чебышёва требует её значительно большего количества и на её выделение уходит больше времени.

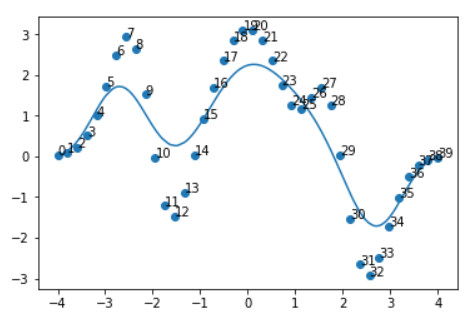
## Часть 8. Кривая Безье.

Кривая Безье по 40 случайным точкам функции, точки пронумерованы.



Собсвенно, а чего вы ждали? Точки случайные и неотсортированные. Кривая, чтобы к ним стремиться, поворачивает очень резко и эти повороты выглядят углами при таком масштабе графика.

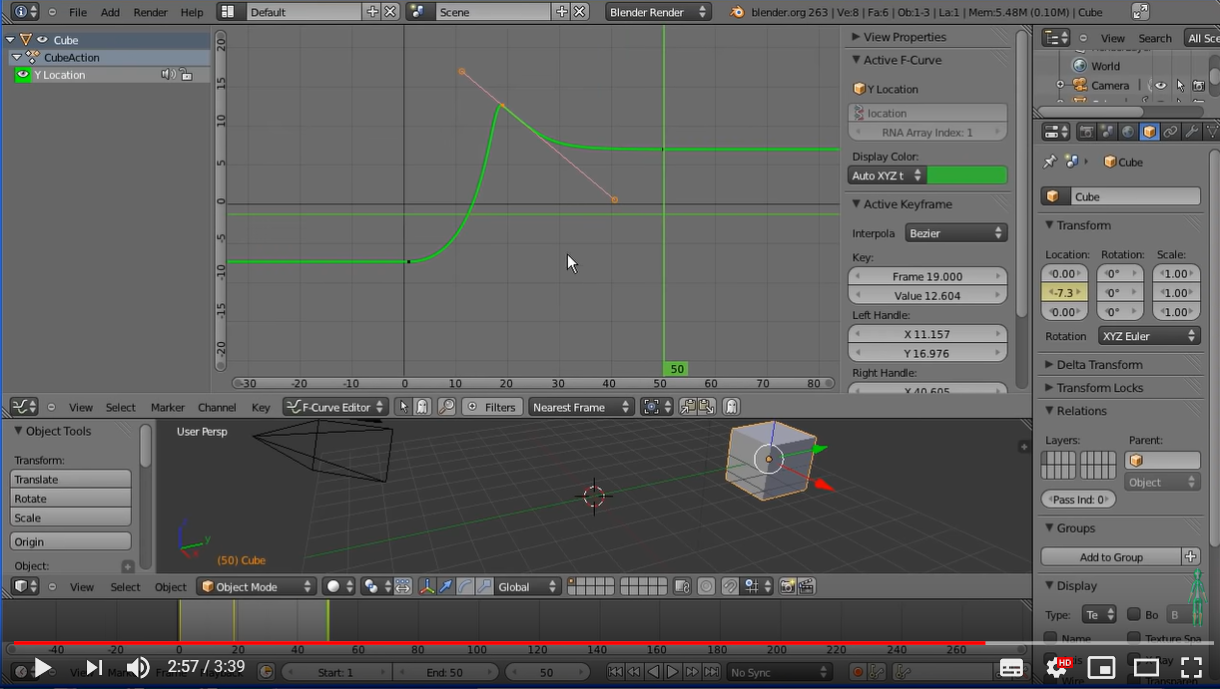
Кривая Безье по 40 равноотстоящим отсортированным точкам функции, точки пронумерованы.



А вот уже кривая после сортировки тех же точек. Выглядит намного лучше, в нужных места стремится к нужным точкам, как и должно быть.

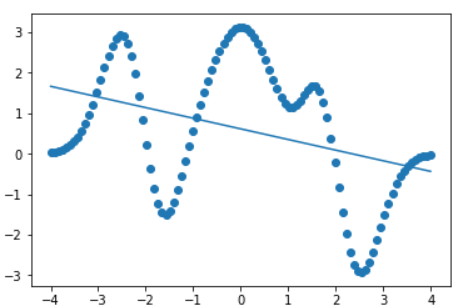
Вывод: кривая Безье штука очень важная и хорошая. Например в той же анимации без неё никак не обойтись. Но вот чтобы её делали по 40 точкам – первый раз слышу. 😊

Вот вам ещё скриншот из [обучающего видео по анимации](https://www.youtube.com/watch?v=R9gCHhJOjlQ), где используется кривая Безье:

[](https://www.youtube.com/watch?v=R9gCHhJOjlQ)

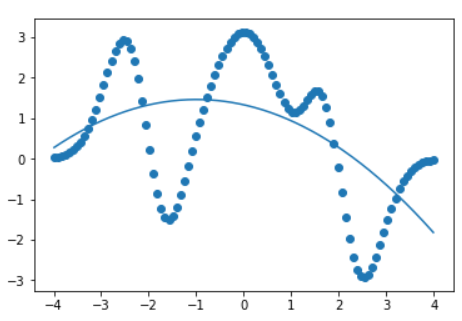
## Часть 9. Среднеквадратичное приближение.

Среднеквадратичное приближение 1-ой степени и его точки:



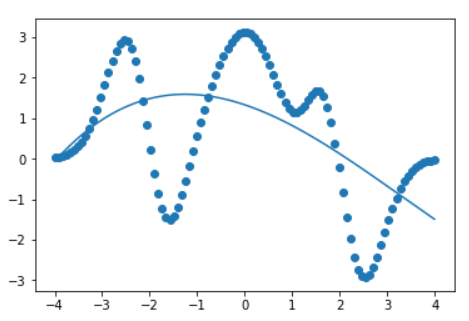
Линия и линия, больше тут и не прокоментируеш.

Среднеквадратичное приближение 2-ой степени и его точки:



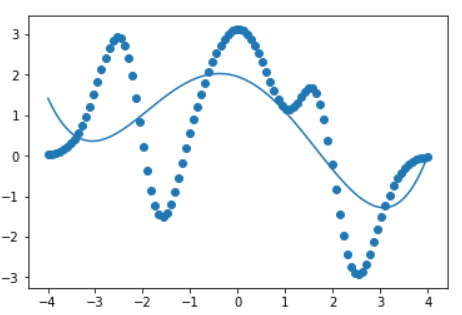
Тут уже более-менее понятно, что парабола стремится иметь наименьшую дистанцию до точек.

Среднеквадратичное приближение 3-ой степени и его точки:



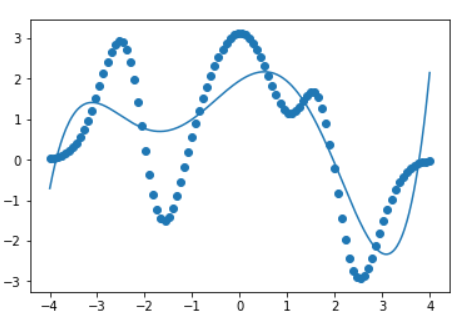
То же самое – начинает наблюдаться среднеквадратичная приближённость.

Среднеквадратичное приближение 4-ой степени и его точки:



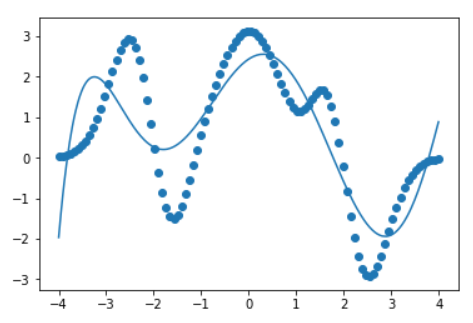
А тут ещё интересней стало!

Среднеквадратичное приближение 5-ой степени и его точки:



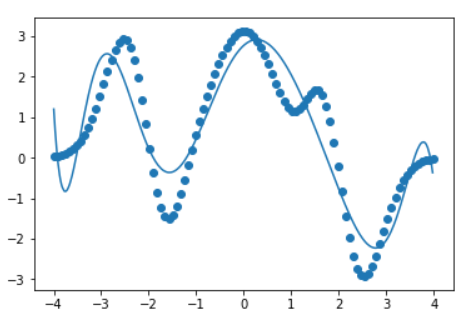
Даже функцией начинает пахнуть!

Среднеквадратичное приближение 6-ой степени и его точки:



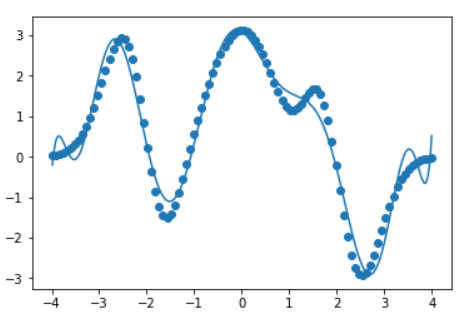
Мы всё ближе и ближе к функции.

Среднеквадратичное приближение 9-ой степени и его точки:



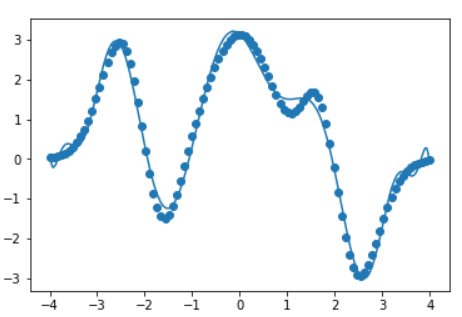
Уже почти наша функция получилась.

Среднеквадратичное приближение 12-ой степени и его точки:



Мы всё ближе и ближе к функции.

Среднеквадратичное приближение 15-ой степени и его точки:

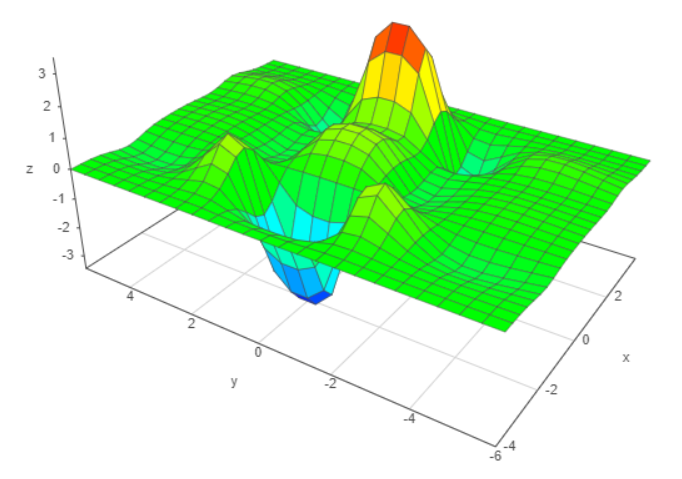


Мне одному кажется, что это очень похоже на многочлен Ньютона по 18 узлам Чебышёва?

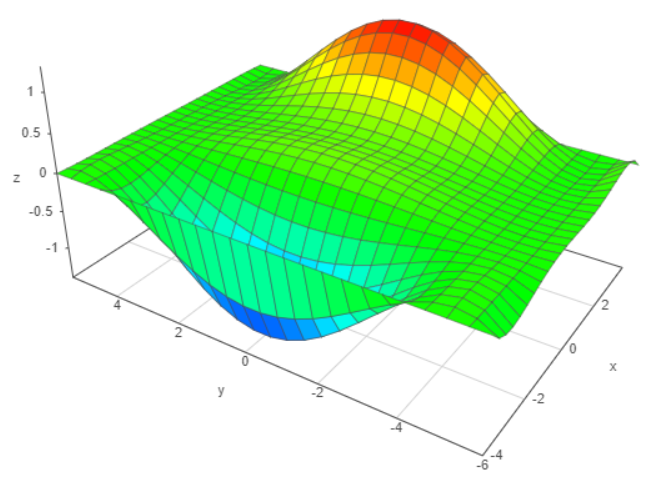
Вывод: выглядит как ещё один способ построения интерполяционного многочлена (я знаю, что на самом деле это не так). Только тут мы уже используем много узлов и подгоняем к ним многочлен какой-либо степени. При больших степенях (соразмерных с такими же степенями у Ньютона) результат получается довольно неплохим. Ясное дело, что при больших степенях за границами промежутка будет происходить дичь. Так как наиболее частая задача среднеквадратичного приближения (в той же физике) – найти вид функция за границами промежутка (или в какой-то пустой области без точек), то обычно используются малые степени, которые не сходят с ума при выходе из промежутка.

## Часть 10. Приближение поверхности многочленом Лагранжа.

Исходная функция:

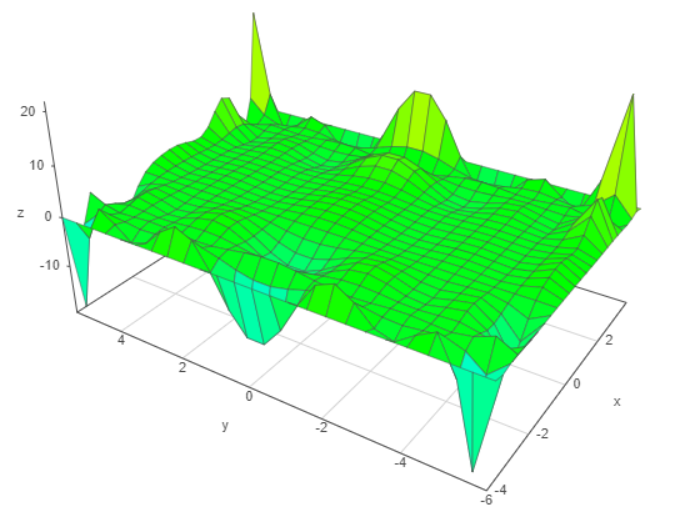


Поверхность многочлена Лагранжа, полученного из сетки 6х6:

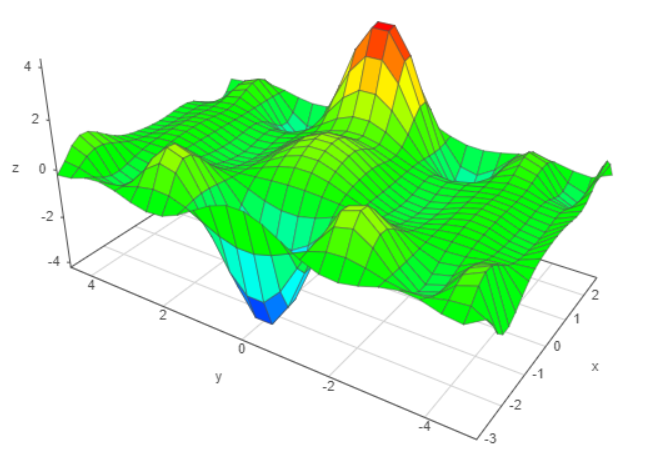


Совсем немного (то есть почти никак) похожа на исходную функцию. Но та же фигня была и в двумерном случае при интерполяции 6 точек, поэтому для 6х6 такой результат нормален.

Поверхность многочлена Лагранжа, полученного из сетки 12х12 на исходном множестве:

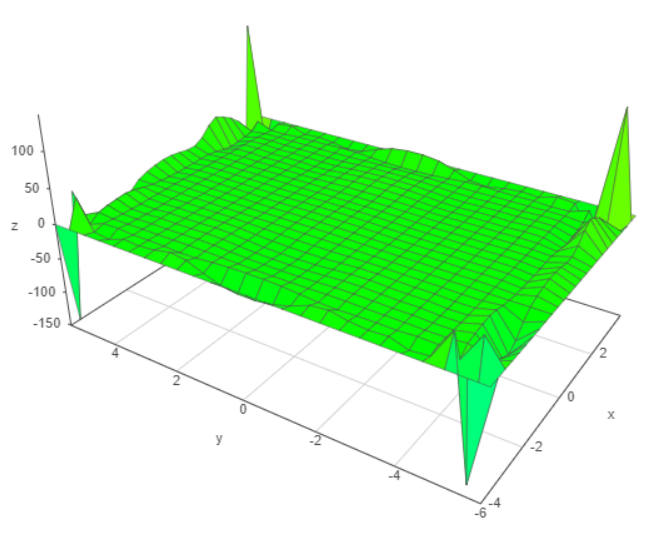


Поверхность многочлена Лагранжа, полученного из сетки 12х12 на усечённом множестве:

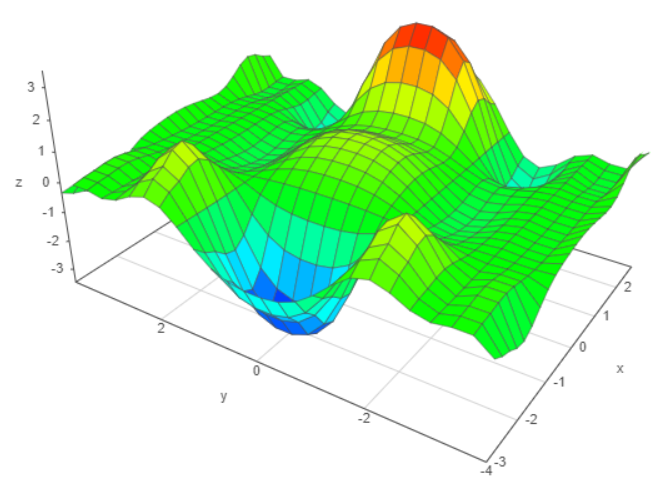


Как видно, по краям функция начала жесточайше скакать, как было и с интерполяцией в двумерном случае. Тем не менее при обрезке краёв мы получаем поверхность, которая довольно близка к искомой.

Поверхность многочлена Лагранжа, полученного из сетки 18х18 на исходном множестве:



Поверхность многочлена Лагранжа, полученного из сетки 18х18 на усечённом множестве:

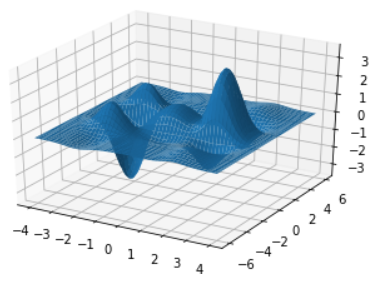


Результат довольно ожидаем и аналогичен двумерному случаю: по краям всё задралось, а по центру получилось очень даже хорошее совпадение.

Вывод: всё тоже, что и с интерполяционными многочленами в двумерном случае, а именно – надо брать достаточно плотную сетку и добавлять к нужному промежку «уши», так как на краях всегда творится какое-то безумие.

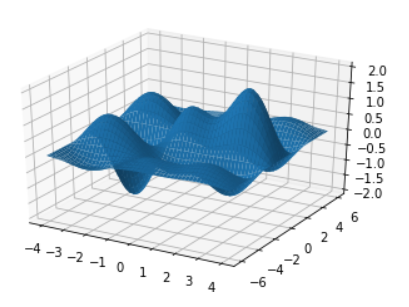
## Часть 11. Приближение поверхности бикубическим сплайном.

Исходная функция:



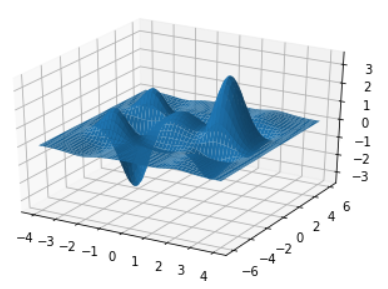
Замечу, что тут графики построены в другой программе, так как использованная в предыдущем пункте не умела строить поверхности по точкам. А для генерации поверхности из сплайнов явно удобнее забить просто матрицу 250х250 точек, чем вбивать по отдельности все ячейки решётки и собирать их в единую фигуру.

Бикубический сплайн 6х6:



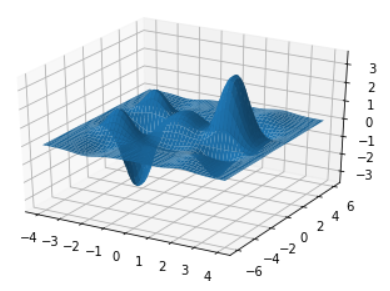
Даже в 6х6 уже видно некоторое совпадение поверхности с искомой. Но, так как некоторые ключевые выпуклости не попали на сетку, всё-таки видно существенное отклонение.

Бикубический сплайн 12х12:



Уже на 12х12 созданная сплайнами поверхность практически неотличима от исходной, что является очень хорошим результатом.

Бикубический сплайн 18х18:



Вы видите отличия 18х18 от 12х12? Я что-то не вижу. В любом случае совпадение с искомой поверхностью чрезвычайно высоко.

Вывод: в трёхмерном случае происходит всё тоже, что и в двумерном, а именно – при наличии достаточно плотной сетки (даже менее плотной, чем нужно для интерполяционного многочлена) бикубический сплайн намного точнее и без ошибок на краях поверхности повторяет искомую плоскость. При этом он обычно строится даже быстрее интерполяционного многочлена.

## Часть 12. Интегралы.

Полная информация про подсчёт интегралов представлена в пункте 11 сгенерированного программой отчёта report.txt. Он достаточно длинный, поэтому в документ я решил его не вставлять.

Отчёт формировался для значений , так как этот набор достаточен для обоснования основных тенденций, которые я изложу ниже, а значения считаются уж очень долго, при этом не добавляю ничего интересного к уже имеющимся наблюдениям.

Заметим, что для малых значений время в отчёте достаточно случайно, так как объём работы очень небольшой и даже при 10 запусках функции тяжело получить точное затраченное время. Также по неизвестной мне причине методы Ньютона и Гаусса при высоких степенях несмотря ни на что работают быстрее Симпсона и трапеций. Скорее всего тут что-то связано со спецификой работы внутреннего оптимизатора GCC. Ньютон и Гаусс используют статичные массивы коэффициентов, а у Симпсона и трапеций они прописаны вручную. Скорее всего компилятор нашёл какую-то интересную оптимизацию, но видит он её только при использовании статичных массивов.

Начнём с левых и правых прямоугольников. По сути эти методы одинаковы и имеют одинаковые свойства. Их линейная сходимости легко заметна по отчёту при . Вот так они медленно и идут к цели, уменьшая погрешность приблизительно в 4 раза при уменьшении шага в 4 раза. Даже если взять очень маленький шаг, эти методы дадут большую погрешность. Фигня, что сказать.

Средние прямоугольник начиная с демонстрируют стабильную квадратичную сходимость вплоть до . А уже на погрешность увеличивается, что свидетельствует о достижении максимально допустимого эффективного шага (то есть при больших шагах всё летит нафиг из-за погрешности чисел с плавающей точкой). Довольно хороший метод, учитывая его простоту и скорость.

Метод трапеций, так же как и метод средних прямоугольников, показывает стабильную квадратичную сходимость, но при этом отстаёт на константу от метода средних прямоугольников (его погрешность всегда где-то в 2.5 раз больше) На он достигает своей максимальной точности. На больших он, очевидно (после такой же точности начали расходиться среднии прямоугольники), начнёт также расходиться из-за погрешности вычислений с плавающей точкой. Метод трапеций не плох, но во всём отстаёт от средних прямоугольников.

Метод Сипмсона сходиться очень быстро и находит свой максимум погрешности (такой же, как и у средних прямоугольников) уже на , после чего так же начинает расходиться из-за погрешности вычислений с плавающей точкой. Таким образом при одинаковых значениях достигнутой погрешности Сипмсон оказывается почти в 5 раз быстрее (0.5мс против 2.7мс) средних прямоугольников, что очень хороший результат.

Методы Ньютона высших степеней показывают несравнимо лучший результат. Так метод 15 степени сошёлся даже к большей максимальной точности, чем Симпсон, всего на , потратив (благодаря компиляторной оптимизации) несравнимо меньше времени. Все методы Ньютона находят свои наилучшие погрешности на . Таким образом они значительно лучше методов Сипмсона и средних прямоугольников.

Методы Гаусса также сходятся к наилучшим результатам на , показывая почти такой же результат как и методы Ньютона. Но следует заметить, что на достигают наилучшего результата сразу методы 4, 5, 6 и 7 степеней. Возможно, что при адаптивном подборе шага методы Гаусса оказались бы даже лучше.

## Часть 13. Правило Рунге.

Полная информация про подсчёт интегралов представлена в пункте 12 сгенерированного программой отчёта report.txt. Он достаточно длинный, поэтому в документ я решил его не вставлять.

В ходе тестирования обнаружилось две интересные особенности:

* При слишком маленьком ε может происходить зануление шага из-за погрешности вычислений с плавающей точкой. Нам нужно проверять шаг и в случае зануления возвращать последний полученный до зануления интеграл.
* Иногда запрос правилом Рунге более мелкого шага приводит к увеличению погрешности. Поэтому мы сверяем погрешности текущего и предыдущего шагов и в случае увеличения погрешности возвращаем предыдущий.

Чтобы уменьшить частоту возникновения вышеописанных ошибок необходимо брать достаточно большие значения ε. Но, что логично, нельзя брать и слишком большие, так как мы всё равно получим низкую точность. Экспериментальным путём я выбрал ε=.

Наилучшим образом себя показал метод Гаусса 6-ой степени, получив точность около за 3 шага алгоритма Рунге с итоговым количеством отрезков равным 32. Выполнение алгоритма заняло меньше 0.1мс (средствами стандартной библиотеки С нельзя отловить промежуток времени меньше 0.1мс).

Также хорошо себя показали методы Гаусса 3, 4, 5, 7 степеней и Ньютона 9, 11, 13 и 15 степеней. Тем не менее методы Гаусса показали немного лучшую точность и скорость благодаря меньшему количеству операций с плавающей точкой. Так же было замечено, что методы относительно низких степеней (Гауса 3, 4 и Ньютона 9) требуют меньшего количества шагов, так как в них округление операций с плавающей точкой портит алгоритм не настолько сильно.

Вывод: алгоритм Рунге (правило Рунге) – очень удобной метод быстрого адаптивного подбора шага для выбранной точности. Тем не менее следует применять его аккуратно и не «закручивать» ε, чтобы избежать искажений из-за погрешности вычислений с плавающей точкой.