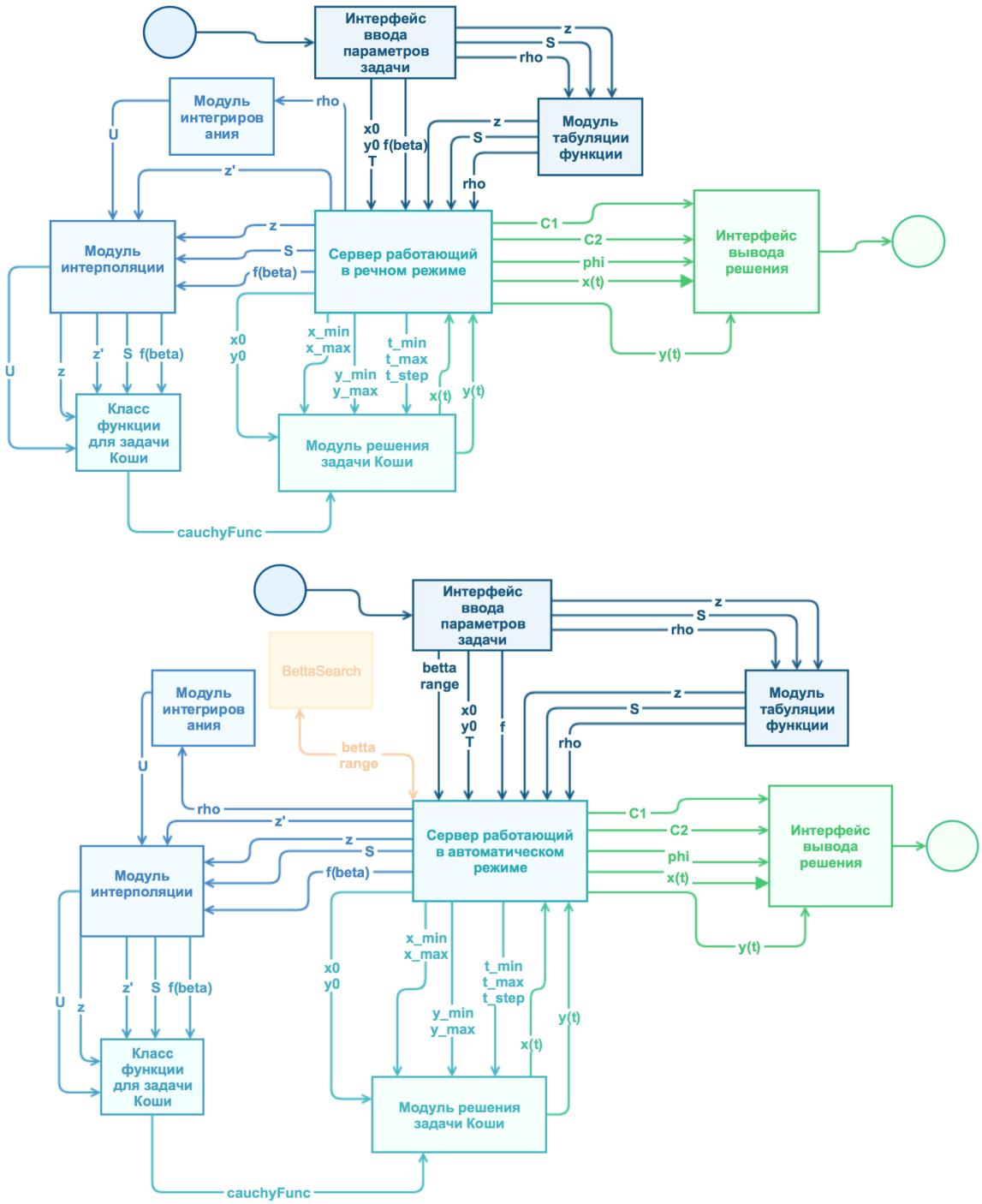
Финальный отчет по проекту “Рекламный сервер”.

Содержание

Структура проекта.

Проект “Рекламный Сервер” реализован с помощью двух языков программирования – C++ и Python. C++ые исходники с численными методами динамически импортятся в код интерфейса написаного на питоне. Для написания интерфейса использоватся flask-фреймфорк.

Схема проекта



Сигнатуры функций

В рамках проекта были реализованы следующий модули и классы:

- класс TabularFunction – класс табулированной функции – хранит в себе мап x->y.
  Подкласс TabularFunctionConstIterator – итератор по хранящимся значениям.
  Функции:
  Дефолтный конструктор

```
TabularFunction();
```

Конструктор копирования

```
TabularFunction(const TabularFunction& function);
```

Функция получения всех известных значений данной функции

```
std::map<double, double> GetValues() const;
```

Функция получения значения функции по значению аргумента

```
double GetValue(const double argument) const;
```

Функция добавления нового значения

```
void AddValue(const double argument, const double value);
```

Функция получения итератора на начало мапа со значениями

```
TabularFunctionConstIterator Begin() const;
```

Функция получения количества известных значений функции

```
size_t Size() const;
```

Функция дифференцирования табулированной функции

```
TabularFunction Diff() const;
```

Функция умножения значений данной функции на какое то число

```
TabularFunction MultiplyBy(const double value) const;
```

Функция смещения значения данной функции в 0 в указаной точке точке

```
TabularFunction MakeValueZero(const double zero_arg) const;
```

Функция уснозения значений данной функции на значения аргумента

```
TabularFunction MultiplyByArgument() const;
```

Функция преобразования f(x) в f(y(x))

```
TabularFunction Combine(const TabularFunction& inner) const;
```

Функция домножения данной функции на другую

```
TabularFunction MultiplyByAnotherFunction(const TabularFunction& another_function)
    const;
```

Функция вывода на экран всех известных значений данной функции

```
void Print() const;
```

• класс TabulateIntegration – модуль позволяющий вычислять интеграл табулированной функции методом трапеций.

Функция вычисления интеграла табулированной функции

```
TabularFunction Integrate(const TabularFunction& function) const;
```

• класс LinearSystemSolution – класс решения линейной системы алгебраических уравнений.

Подкласс ConstIterator итератор по полученным значениям x

Функции:

Функция получения итератора на начало ответов

```
ConstIterator begin() const;
```

Функция получения итератора на конец ответов

```
ConstIterator end() const;
```

Конструктор

```
LinearSystemSolution(const std::vector<double>& solution);
```

Функция для определения наличия ответа в данном классе

```
bool IsThereSolution() const;
```

Функция получения вектора ответов из данного класса

```
std::vector<double> GetSolution() const;
```

• класс LinearSystem – класс вычисления решения линейной системы алгебраических уравнений.

Функции:

Функция очищения системы

```
LinearSystem& clear();
```

Функция добавления линейного уравнения в систему

```
LinearSystem& AddEquation(const std::vector<double>& new_left_part, const double
    right_part);
```

Функция свапа двух уравнений в системе

```
LinearSystem& SwapEquations(const size_t first_index, const size_t second_index);
```

Функция деления коэффициентов данного уравнения из системы на какое то число

```
LinearSystem& Transformation(const size_t first_index, const size_t second_index,
    const double coefficient);
```

Функция решения системы уравнений

```
LinearSystemSolution Solve();
```

• класс CubeSpline – класс кубического сплайна.
Функции:
Конструктор

```
CubeSpline(const double a, const double b, const double c, const double d, const
    double x)
```

Функция получения значения сплайна в данной точке

```
double GetValue(const double x) const;
```

• класс Interpolator.
Функции:
Конструктор

```
Interpolator(const TabularFunction& function);
```

Функция получения значения интерполированной функции по аргументу

```
double GetValue(const double x) const;
```

Функция получения табулированной функции из интерполированной

```
TabularFunction GetTabularFunction(const double precise_step) const;
```

• класс CauchySolution – класс решения задачи Коши – две табулированные функции от t.
Функции:
Функция получения значений первой функции

```
TabularFunction GetX() const;
```

Функция получения значений второй функции

```
TabularFunction GetY() const;
```

• класс BaseCauchyFunction – базовый класс для функции из задачи Коши.
• класс CircleCauchyFunction – функция для которой решения задачи Коши – круг.
Функции:
Функция получения значения X

```
double GetValueX(const double x, const double y, const double t) const;
```

Функция получения значения Y

```
double GetValueY(const double x, const double y, const double t) const;
```

• класс SpiralCauchyFunction – функция для которой решения задачи Коши – спираль.
Функции:
Функция получения значения X

```
double GetValueX(const double x, const double y, const double t) const;
```

Функция получения значения Y

```
double GetValueY(const double x, const double y, const double t) const;
```

• класс CauchyProblem – модуль для решения задачи Коши методом Эйлера.
Функции:
Конструктор принимающий ограничения на значения аргументов и векторфункцию f.

```
CauchyProblem(const double x0, const double y0, const double x_min, const double
    x_max, const double y_min, const double y_max, const double t_min, const double
    t_max, const double t_step, BaseCauchyFunction * f);
```

• класс ParametrisedAdBaseFunction – базовый класс функции f.
Функции:
Функция присваивания значения бете

```
virtual void SetBeta(const double new_beta);
```

Функция получения значения беты

```
virtual double GetBeta() const;
```

Функция получения значения данной функции

```
virtual double GetValue(const double z, const double x, const double S) const;
```

• класс SimpleDiffAdFunction – beta * (S - x).
• класс AdCauchyFunction – функция для задачи коши соответствующая ParametrisedAdBaseFunction.
Функции:
Конструктор

```
AdCauchyFunction(const Interpolator& z_diff, const Interpolator& U, const
    Interpolator& z, const Interpolator& S, ParametrisedAdBaseFunction* f_beta);
```

Функция получения значения беты из

```
f_beta double GetBeta() const;
```

Функция получения значения X (первой функции z_diff.GetValue(t) * U.GetValue(y))

```
double GetValueX(const double x, const double y, const double t) const;
```

Функция поучения занчения(f_beta->GetValue(z.GetValue(t), x, S.GetValue(t))) Y

```
double GetValueY(const double x, const double y, const double t) const;
```

Функция записи беты

```
void SetBeta(const double beta);
```

• класс AdServer – базовый класс сервера являющийся интерфейсом для ручного режима.
Функции:
Конструктор

```
AdServer(const TabularFunction& rho, const TabularFunction& S,
    ParametrisedAdBaseFunction * f_beta, const TabularFunction& z, const double x0,
    const double y0, const double T, const double step)
```

Сеттеры

```
void SetBeta(const double beta); void SetX0(const double x0); void SetY0(const double
    y0);
```

Геттеры полученных значений функций

```
TabularFunction GetS() const; TabularFunction GetX() const; TabularFunction GetY()
    const; double GetBeta() const;
```

Функция для решения задачи Коши относительно z.Diff(), Interpolator(integration.Integrate(rho).MultiplyBy(-1).MakeValueZero(1.)), Interpolator(z), Interpolator(S), f_beta

```
void SolveCauchyProblem();
```

Функции для получения значений функционалов качества

```
double C1() const;
double C2() const;
double Phi() const;
```

• класс AdServerAutoSetup – базовый класс сервера являющийся интерфейсом для автоматического режима.

Функции:

Конструктор

```
AdServerAutoSetup(const AdServer& ad_server, const double beta_min, const double
    beta_max, const double beta_precision);
```

Сеттеры

```
void SetX0(const double x0); void SetY0(const double y0);
```

Функция поиска оптимального значения ветта с помощью тернарного поиска

```
void SetupOptimalBeta();
```

Геттер сервера который делает вычисления все функционалов при заданной бете

```
AdServer GetAdServer() const;
```

Метод трапеций

Простая и составная формула метода трапеций

Простая формула метода трапеций:

$$\int_a^b f(x)dx \approx \frac{f(a)+f(b)}{2}(b-a)$$

Составная формула метода трапеций:

$$\int_a^b f(x)dx \approx \sum_{i=0}^{n-1}\left(\frac{f(x_i)+f(x_{i+1})}{2}(x_{i+1}-x_i)\right)$$

Оценка аналитического порядка точности

Рассмотрим функцию f на отрезке от x_i до $x_i + h$. Разложим ее в ряд Тейлора с остаточным членом в форме Лагранжа

$$f(x) = f(x_i) + (x - x_i)f'(x_i) + \frac{(x - x_i)^2}{2}f''(\xi),$$

где $\xi \in [x_i, x_i + h]$.

Проинтегрируем это разложение:

$$\int_{x_i}^{x_i+h} f(x)dx = hf(x_i) + \frac{h^2}{2}f'(x_i) + \frac{h^3}{6}f''(\xi).$$

Также подставим $x = x_i + h$ в это разложение

$$f(x_i + h) = f(x_i) + hf'(x_i) + \frac{h^2}{2}f''(\xi).$$

Преобразуем последние два соотношения: вычтем из первого второе, домноженое на $h/2$:

$$\int_{x_i}^{x_i+h} f(x)dx - \frac{h}{2}f(x_i + h) = \frac{h}{2}f(x_i) - \frac{h^3}{12}f''(\xi).$$

$$\left| \int_{x_i}^{x_i+h} f(x)dx - \frac{f(x_{i+1}) - f(x_i)}{2}(x_{i+1} - x_i) \right| \leq ||f''||\frac{h^3}{12}$$

Просуммировав эти неравенства по всем отрезкам $[x_i, x_{i+1}]$ получим, что общий уровень ошибки меньше, чем

$$||f''||\frac{h^3}{12} * \frac{b - a}{h} = ||f''||\frac{h^2(b - a)}{12}.$$

Это означает, что порядок точности второй.

Реализация на sage для последующих оценок метода

```
def get_segments(a, b, step):
	x = a
	while (x + step <= b):
		yield (x, x + step)
		x += step

def trapesoid_integrate(func, a, b, step):
	integral = 0
	for (x, y) in get_segments(a, b, step):
		integral += (func(x) + func(y)) / 2 * (y - x)

	return integral
```

Сравнение методов с встроенными методами sage для примеров

Сравнивающий код:

```
from sage.symbolic.integration.integral import definite_integral

for (func, a, b) in (
   (x^3, 1, 2),
   (sgn(x), -1, 1),
   (sin(1/x), 0.000001, 1),
):
   print "Func {func} on segment [{a}, {b}]: trapesoid method value : {trapesoid_value},
       real value : {real_value}".format(
      func=func, a=a, b=b, trapesoid_value = trapesoid_integrate(func,a, b, 0.001),
          real_value = float(definite_integral(func, x, a, b))
   )
```

Вывод кода:

```
Func x^3 on segment [1, 2]: trapesoid method value : 3.75000074999912, real value : 3.75
Func sgn(x) on segment [-1, 1]: trapesoid method value : -5.55111512312578e-17, real
    value : 0.0
Func sin(1/x) on segment [1.00000000000000e-6, 1]: trapesoid method value :
    0.500258204182386, real value : 0.504067061906
```

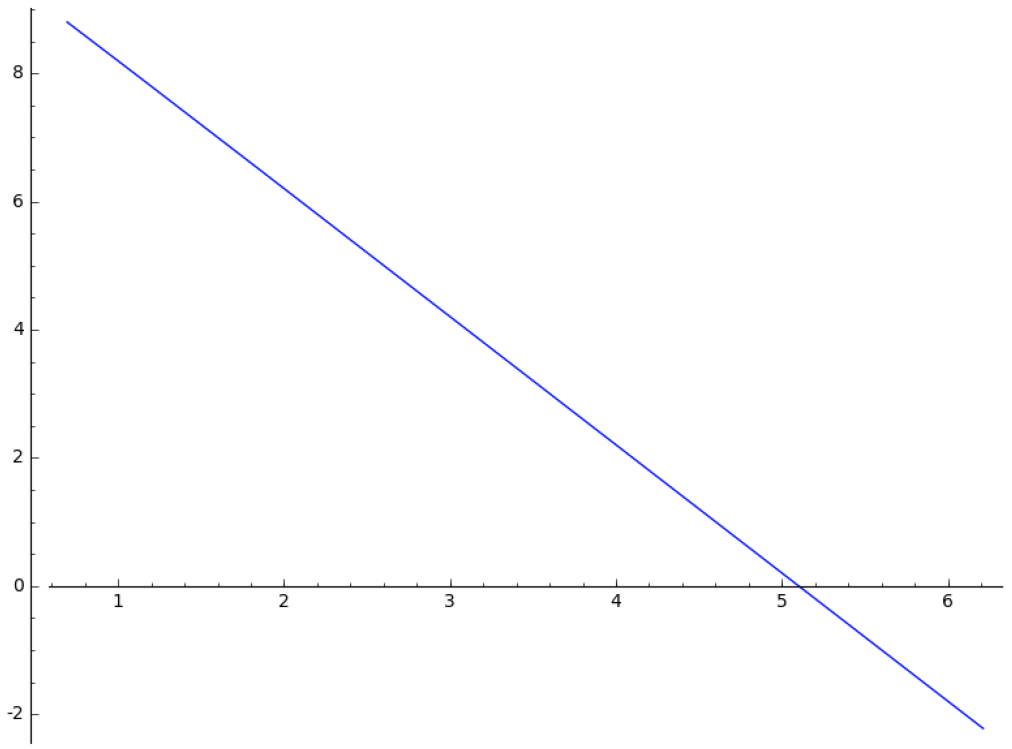
Видно, что для гладкой и разрывной функции удалось довольно хорошо вычислить значение интеграла. С осциллирующей функцией получилось не так хорошо, есть ошибка уже в тысячных долях. Это произошло, потому что ее значение второй производной велико.

Экспериментальный анализ точности метода

Код для анализа:

```
func = x^4
a = 1.
b = 2.
points = []
for segments_count in xrange(2, 500):
   step = (b - a) / segments_count
   error = abs(float(definite_integral(func, x, a, b)) - trapesoid_integrate(func, a, b,
       step))
   points.append((log(segments_count), log(error)))
line(points)
```

Полученный график



На нем видим, что изначально, когда сегментов мало, ошибка большая, а потом она уменьшается и логарифм ошибки уже становится отрицательным. Наклон кривой как раз равен -2, что подтверждает теоретические заключения о втором порядке погрешности.

Метод Гаусса

Описание метода Гаусса решения системы линейных уравнений

Метод Гаусса заключается в том, что сначала нужно выписать матрицу линейной системы уравнений и столбец свободных коэффициентов справа от нее. Далее, нужно совершать элементарные преобразования строк с матрицей и такие же преобразования строк со столбцом свободных коэффициентов (элементарное преобразование – это вычитание из одной строки другой строки, домноженной на любой коэффициент).

Метод работает тогда и только тогда, когда определитель матрицы уравнения отличен от нуля. Пусть это так. Тогда в первом столбце матрицы есть ненулевое число. Перенесем строку с этим числом на первое место, а затем из всех остальных строк, начиная со второй, вычтем первую строку с таким коэффициентом, чтобы все элементы в первом столбце, кроме первого, стали равны нулю. Далее рассмотрим подматрицу исходной матрицы, начиная со второй строки и второго столбца и применим к ней тот же алгоритм.

В итоге, матрица будет иметь верхнетреугольный вид с ненулевыми коэффициентами на диагонали. После этого, наоборот, вычитаем из всех строк последнюю с таким коэффициентом, чтобы занулить все числа в последнем столбце кроме последнего. Затем, рассматриваем подматрицу исходной матрицы без последнего столбца и последней строки и применяем к ней тот же алгоритм.

В итоге, матрица будет иметь диагональный вид. Остается лишь решить систему уравнений, в которой уравнения имеют вид $k_i x_i = b_i$, и решение будет иметь вид $x_i = \dfrac{b_i}{k_i}$.

Обоснование корректности метода Гаусса решения системы линейных уравнений

Метод Гаусса использует элементарные преобразования строк матрицы и корректен, потому что соблюдение условий $f = 0, g = 0$ равносильно соблюдению условий $f + \lambda g = 0, g = 0$.

Система уравнений преобразуется к равносильной ей системе, которая затем решается.

Условия применимости метода Гаусса для решения линейной системы уравнений

Как уже было сказано ранее, для конструкции необходимо, чтобы определитель матрицы был отличен от нуля, иначе на очередном шаге мы не можем гарантировать, что найдем ненулевой элемент для того, чтобы за его счет обнулить все остальные в столбце.

Вычисления с тремя матрицами

Для вычисления определителей, матричных норм и неувязок используем sage

Код программы:

```
def get_modules_sum(numbers):
   return sum(abs(x) for x in numbers)

def get_line_modules_sum(cur_matrix):
   for line in cur_matrix:
      yield get_modules_sum(line)

def get_infty_norm(cur_matrix):
   return max(get_line_modules_sum(cur_matrix))

def get_conditionality_number(cur_matrix):
   return get_infty_norm(cur_matrix) * get_infty_norm(cur_matrix^(-1))

def elementary_transformation(cur_matrix, coefs, i, j, la):
   cur_matrix[i] -= cur_matrix[j] * float(la)
   coefs[i] -= coefs[j] * float(la)

def swap_rows(cur_matrix, coefs, i, j):
   (cur_matrix[i], cur_matrix[j]) = (cur_matrix[j], cur_matrix[i])
   (coefs[i], coefs[j]) = (coefs[j], coefs[i])

def gauss_solve(cur_matrix, coefs):
   for start_row_index in xrange(cur_matrix.nrows()):
      max_row_index = None
      max_col_value = None
      for row_index in xrange(start_row_index, cur_matrix.nrows()):
         if max_col_value == None or max_col_value <
             abs(cur_matrix[row_index][start_row_index]):
            max_col_value = abs(cur_matrix[row_index][start_row_index])
            max_row_index = row_index

      swap_rows(cur_matrix, coefs, start_row_index, max_row_index)

      for row_index in xrange(start_row_index + 1, cur_matrix.nrows()):
         elementary_transformation(
            cur_matrix,
            coefs,
            row_index,
```

```
            start_row_index,
            cur_matrix[row_index][start_row_index]
               / cur_matrix[start_row_index][start_row_index]
         )

   for start_row_index in xrange(cur_matrix.nrows() - 1, -1, -1):
      for row_index in xrange(start_row_index - 1, -1, -1):
         elementary_transformation(
            cur_matrix,
            coefs,
            row_index,
            start_row_index,
            cur_matrix[row_index][start_row_index]
               / cur_matrix[start_row_index][start_row_index]
         )

   for row_index in xrange(cur_matrix.nrows()):
      yield coefs[row_index] / cur_matrix[row_index][row_index]

def make_column(vector):
   return matrix([[x] for x in vector])

def calc_error_vector(cur_matrix, coefs, solution):
   return make_column(coefs) - cur_matrix * make_column(solution)

def calc_error(cur_matrix, coefs, solution):
   return max(line[0] for line in calc_error_vector(cur_matrix, coefs, solution))

A = matrix([[1,2,3],[2.0001,3.999,6],[15,3,6]])
B = matrix(QQ, 8, 8, lambda i, j: 1./(i + j + 1))
C = matrix([[float(10^6),float(2)],[float(10^13),float(2)]])

rows = [["det","matrix_norm","conditionality_number","error"]]
for cur_matrix in (A, B, C):
   infty_norm = get_infty_norm(cur_matrix)
   conditionality_number = get_conditionality_number(cur_matrix)
   b = list(1 for i in xrange(cur_matrix.nrows()))
   solution = list(gauss_solve(cur_matrix, b))
   error = calc_error(cur_matrix, b, solution)
   rows.append([
      float(det(cur_matrix)),
      float(infty_norm),
      float(conditionality_number),
      error,
   ])
print table(rows)
```

Вывод программы:

```
det               matrix_norm   conditionality_number error
-0.0387           24.0          72557.9534884         0.0
2.73705011549e-33 2.71785714286 33872791095.0         9.04844199567e-09
1.9999998e+13     1e+13         5.000001e+12          0.0
```

Вывод: точный метод Гаусса ошибается при большом числе обусловленности, а при большой норме – не обязательно.

Интерполяция кубическими сплайнами

Описание интерполяции кубическими сплайнами

Дан отрезок, разбитый на много маленьких отрезков. Даны значения некоторой неизвестной функции в узлах сетки. В алгоритме аппроксимации, на каждом маленьком отрезочке мы пытаемся найти кубическую функцию, которая совпадает с исходной функцией в точках сетки, при этом добавляем условия равенства первой и второй производной соседних сплайнов в точке сетки, а также добавляем условия на то, что вторая производная в концах отрезка равна нулю.

Все условия являются линейными относительно коэффициентов всех сплайнов, поэтому формируют систему линейных уравнений, которую можно решить, и получить результирующие коэффициенты сплайнов.

Вычисления

В качестве гладкой функции взята x^3, в качестве осциллирующей $\sin\left(\frac{1}{x}\right)$, а в качестве разрывной – $\operatorname{sgn}(x)$.

Вот вычисляющий код:

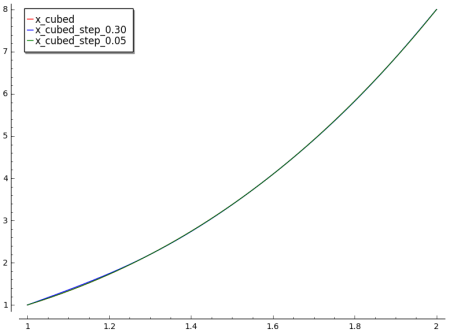
```
def get_segments(a, b, step):
   x = a
   while (x + step <= b):
      yield (x, x + step)
      x += step
   if x < b:
      yield (x, b)

class CubeSpline(object):
   def __init__(self, a, b, c, d, x1):
      self.a = a
      self.b = b
      self.c = c
      self.d = d
      self.x1 = x1

   def get_value(self, x):
      return self.a + self.b * (x - self.x1) + self.c/2 * (x - self.x1)**2 + self.d / 6 *
          (x - self.x1) ** 3

class SplineConvex(object):
   def __init__(self, func, a, b, step):
      self.splines = []
      self.segments = list(get_segments(a, b, step))
      n = len(self.segments)

      equations = [
         [1] + [0] * n,
         [0] * n + [1],
      ]
      b = [
         0,
         0,
      ]

      for i in xrange(1, n):
         (x_i_prev, x_i) = self.segments[i - 1]
```

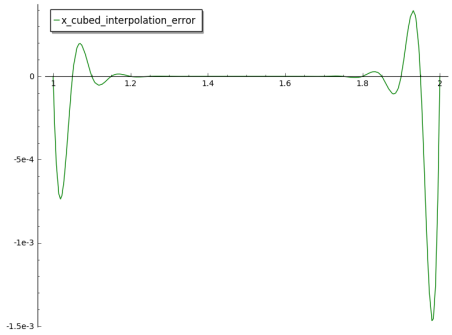
```
            (x_i, x_i_post) = self.segments[i]
            f_i = func(x_i)
            f_i_post = func(x_i_post)
            f_i_prev = func(x_i_prev)
            h_i = x_i - x_i_prev
            h_i_post = x_i_post - x_i
            equations.append(
               [0] * (i - 1) + [h_i, 2 * (h_i + h_i_post), h_i_post] + [0] * (n - i - 1),
            )
            b.append(6 * ((f_i_post - f_i) / h_i_post - (f_i - f_i_prev) / h_i))

         c_coefs = list(gauss_solve(matrix(equations), b))

         for i in xrange(1, n + 1):
            c_i = c_coefs[i]
            c_i_prev = c_coefs[i - 1]
            (x_i_prev, x_i) = self.segments[i - 1]
            f_i = func(x_i)
            f_i_prev = func(x_i_prev)
            h_i = x_i - x_i_prev
            a_i = f_i
            d_i = (c_i - c_i_prev) / h_i
            b_i = (f_i - f_i_prev) / h_i + h_i * (2 * c_i + c_i_prev) / 6
            self.splines.append(CubeSpline(a_i, b_i, c_i, d_i, x_i))


      def get_value(self, x):
         for index in xrange(len(self.segments)):
            (x1, x2) = self.segments[index]
            spline = self.splines[index]
            if x1 <= x and x <= x2:
               return spline.get_value(x)



for (func, a, b, func_name) in (
   (x^3, 1., 2., "x_cubed"),
   (sgn(x), -1., 1., "sign_x"),
   (sin(1/x), 0.000001, 1., "sin_reverse_x"),
):

   g = Graphics()
   g += plot(func, (x, a, b), color='red', legend_label=func_name)
   for (step, color) in ((0.3, 'blue'), (0.05, 'green')):
      spline_convex = SplineConvex(func, a, b, step)
      g += plot(
         lambda t: spline_convex.get_value(t),
         (x, a, b), color=color,
         legend_label="{func_name}_step_{step:.2f}".format(func_name=func_name,
             step=float(step))
      )

   g.save("{func_name}_splines_image.png".format(func_name=func_name))

   g = Graphics()
   step = 0.05
   spline_convex = SplineConvex(func, a, b, step)
   g += plot(
```

```
    lambda t: func(t) - spline_convex.get_value(t),
    (x, a, b), color=color,
    legend_label="{func_name}_interpolation_error".format(func_name=func_name)
)
g.save("{func_name}_interpolation_error.png".format(func_name=func_name))
```

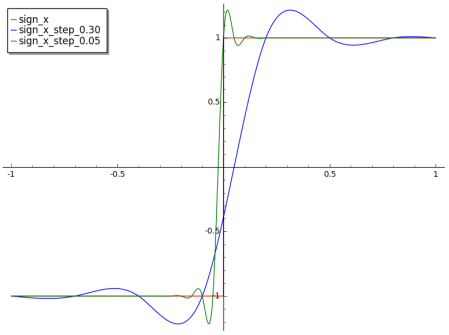
Картинки

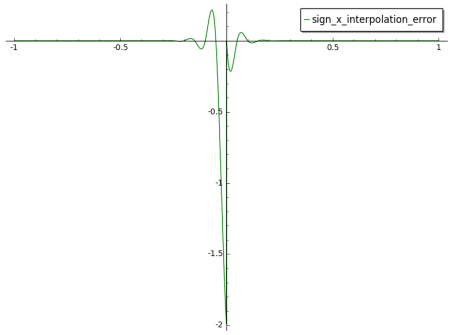
Для функции x^3



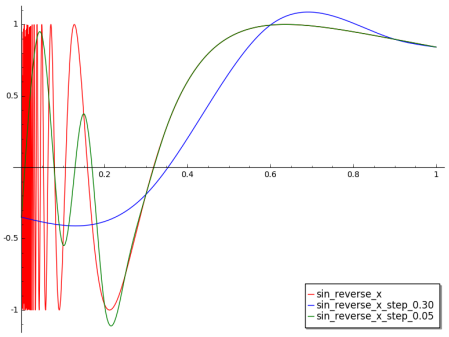


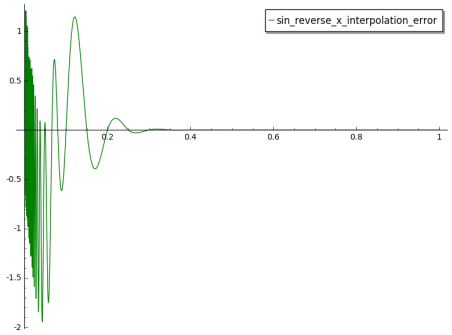
Для функции $\mathrm{sgn}(x)$





Для функции $\sin\left(\frac{1}{x}\right)$





Выводы про аппроксимацию

Гладкую функцию приблизили почти идеально, но на границах приближения есть небольшие артифакты из-за того, что мы приравняли вторую производную на краях нулем, а это в реальной модели не так.

У разрывной функции наблюдаем образование лишних изгибов и экстремумов в районе разрыва.

С оцсиллирующей функцией сплайны справляются очень хорошо до тех пор, пока осцилляции не становятся в несколько раз чаще, чем шаг сетки. Но это логично, тут и невозможно ожидать более хорошего результата.

Решение задачи Коши методом Эйлера

Описание метода решения задачи Коши

Для решения задачи Коши от двух переменных выбран метод Эйлера. Он заключается в том, что мы строим точки x_n, y_n, t_n по следующему индуктивному правилу:

- x_0, y_0 – начальная точка из условия задачи Коши, $t_0 = 0$,
- $x_{n+1} = x_n + t * f_x(x_n, y_n, t_n), \quad y_{n+1} = y_n + t * f_y(x_n, y_n, t_n), \quad t_{n+1} = t_n + t,$

где t – значение временного шага. Чем оно меньше, тем точнее решение, полученное в результате применения метода.

Полученные точки (x_n, y_n) образуют табулированное численное решение.

Единственность решения

Точное решение задачи Коши единственно по теореме о единственности.

Наше численное решение для заданного заранее параметра t также обладает свойством единственности, так как если бы фазовые кривые пересекались, возникло бы противоречие с тем, что для точки пересечения алгоритм построения следующей точки решения детерминирован.

Применение решения для конкретной задачи Коши

Рассмотрим задачу $x' = y, \quad y' = -x, \quad x_0 = 1, \quad y_0 = 0.$

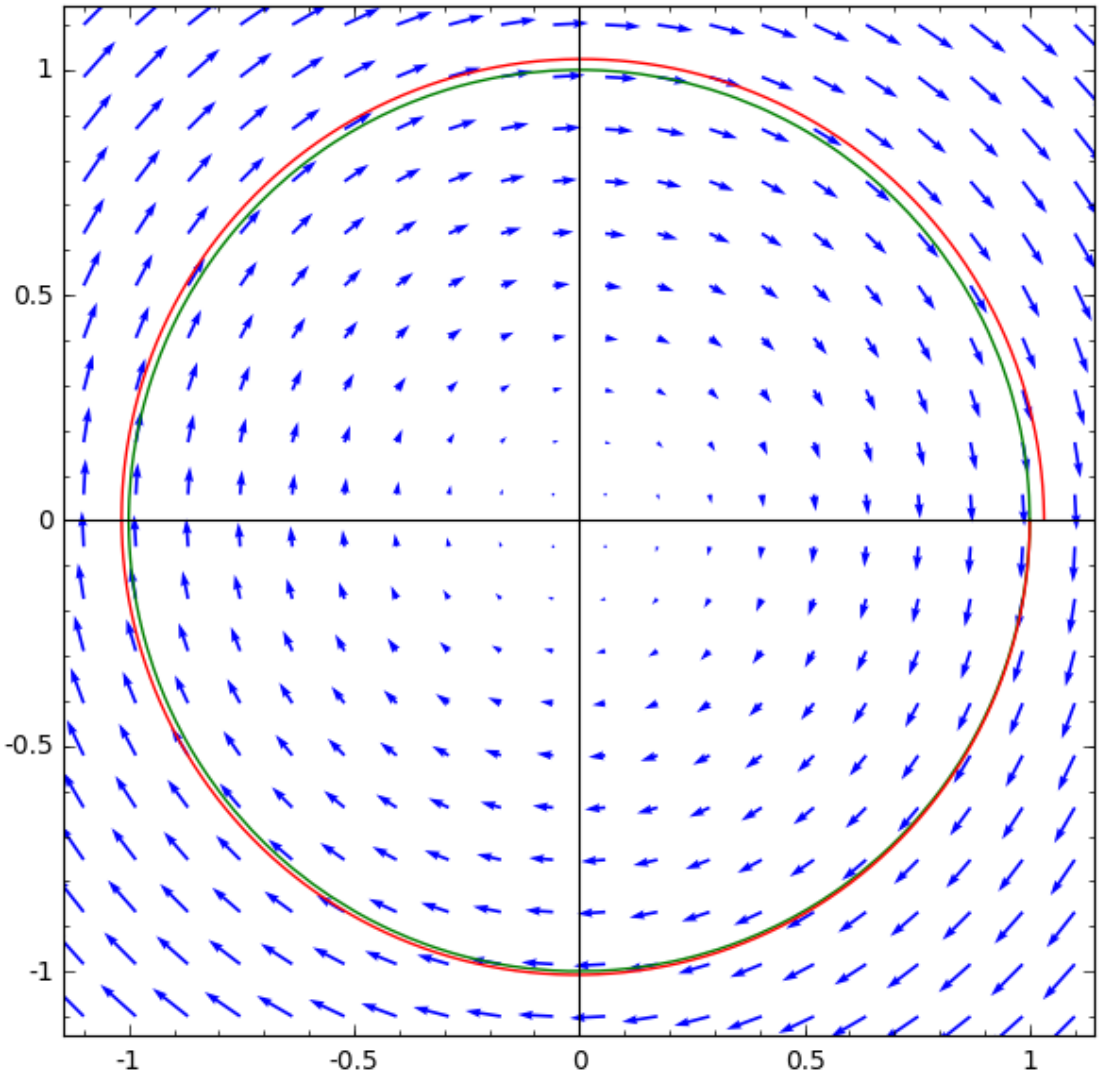
Ее точное решение: $x = cos(t), \quad y = -sin(t)$. Без заданного начального условия, фазовые кривые имеют вид $x = C\cos(t), \quad y = -C\sin(t)$, являются окружностями всех радиусов вокруг начала координат.

Код, строящий графики

```
class DiffEq(object):
  def __init__(self, fx, fy, start_t, end_t, step_t, x0, y0):
    self.fx = fx
    self.fy = fy
    self.start_t = start_t
    self.end_t = end_t
    self.step_t = step_t
    self.x0 = x0
    self.y0 = y0

  def build_solution(self):
    x = []
    y = []
    x_y_data = []
    cur_t = self.start_t
    prev_x = self.x0
    prev_y = self.y0
    prev_t = cur_t
    while cur_t < self.end_t:
      cur_x = prev_x + (cur_t - prev_t) * self.fx(x = prev_x, y = prev_y, t = prev_t)
      cur_y = prev_y + (cur_t - prev_t) * self.fy(x = prev_x, y = prev_y, t = prev_t)
      x.append((cur_t, cur_x))
      y.append((cur_t, cur_y))
      x_y_data.append((cur_x, cur_y))
      prev_x = cur_x
      prev_y = cur_y
      prev_t = cur_t

      cur_t += self.step_t
```

```
        self.x = x
        self.y = y
        self.x_y_data = x_y_data


x = var('x')
y = var('y')
t = var('t')
g = Graphics()
g += plot_vector_field((y, -x), (x,-1.1,1.1), (y,-1.1,1.1), color="blue")
g += parametric_plot((cos(t),sin(t)),(t, 0, 2*pi),color="green")
diff_eq = DiffEq(y, -x, 0, 2*pi, 0.01, 1, 0)
diff_eq.build_solution()
g += list_plot(diff_eq.x_y_data, plotjoined=True,color="red")
g.save("diff_eq_solution.png")
```

Синим цветом обозначено векторное поле, зеленым – идеальное аналитическое решение, красным – приближенное решение. Видно, что оно отличается от идеального решения.



Оценка погрешности метода

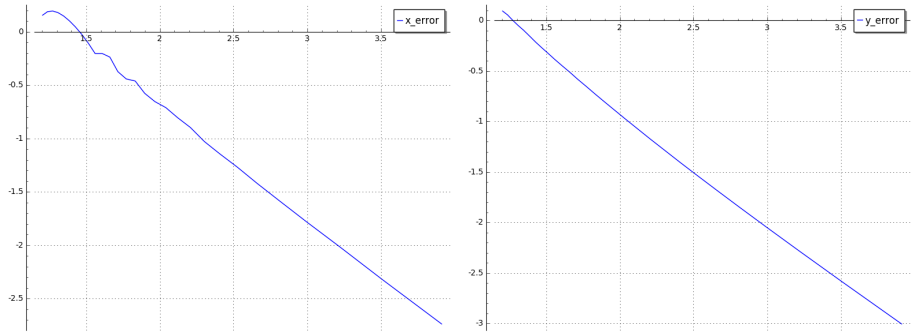
Для выше рассмотренной задачи построим графики логарифма ошибки по норме Чебышева от логарифма шага алгоритма.

Код, который строит график ошибки

```
x = var('x')
y = var('y')
t = var('t')
h = 0.3
log_error_points = {}
while h > 0.01:
   diff_eq = DiffEq(y, -x, 0, 2*pi, h, 1, 0)
   diff_eq.build_solution()

   print "for h={h}".format(h=h)
   for (real_func, approx_func, var_name) in ((cos(t), diff_eq.x, 'x'), (-sin(t),
```

```
            diff_eq.y, 'y')):
        max_norm = 0
        cur_t = 0
        for (t_, x_) in approx_func:
            if max_norm < abs(real_func(t_) - x_):
                max_norm = abs(real_func(t_) - x_)

        log_error_points.setdefault(var_name, []).append((-log(h), log(max_norm)))

    h -= 0.01

for var_name in ('x', 'y'):
    g = Graphics()
    g += list_plot(log_error_points[var_name], plotjoined=True,
         legend_label="{var_name}_error".format(var_name=var_name), gridlines=True)
    g.save("diff_eq_{var_name}_error.png".format(var_name=var_name))
```

Графики ошибок:



Из них не затруднительно видеть, что метод имеет первый порядок ошибки.