Лекция 8. Реализация некоторых численных методов

8.1 Программирование методов решения нелинейных уравнений в Maxima

Необходимость отыскания корней нелинейных уравнений встречается в целом ряде задач: расчетах систем автоматического управления и регулирования, собственных колебаний машин и конструкций, в задачах кинематического анализа и синтеза, плоских и пространственных механизмов и других задачах.

Пусть дано нелинейное уравнение $f(x) = 0$, и необходимо решить это уравнение, т. е. найти его корень $\overline{x}$.

Если функция имеет вид многочлена степени m, $f(x) = a_0x^m + a_1x^{m-1} + a_2x^{m-2} + \ldots + a_{m-1}x + a_m$, где a_i — коэффициенты многочлена, $i = \overline{1, m}$, то уравнение $f(x) = 0$ имеет m корней (основная теорема алгебры).

Если функция $f(x)$ включает в себя тригонометрические или экспоненциальные функции от некоторого аргумента x, то уравнение $f(x) = 0$ называется трансцендентным уравнением. Такие уравнения обычно имеют бесконечное множество решений.

Как известно, не всякое уравнение может быть решено точно. В первую очередь это относится к большинству трансцендентных уравнений.

Доказано также, что нельзя построить формулу, по которой можно было бы решать произвольные алгебраические уравнения степени, выше четвертой.

Однако точное решение уравнения не всегда является необходимым. Задачу отыскания корней уравнения можно считать практически решенной, если мы сумеем найти корни уравнения с заданной степенью точности. Для этого используются приближенные (численные) методы решения.

Большинство употребляющихся приближенных методов решения уравнений являются, по существу, способами уточнения корней. Для их применения необходимо знание интервала изоляции $[a, b]$, в котором лежит уточняемый корень уравнения.

Процесс определения интервала изоляции $[a, b]$, содержащего только один из корней уравнения, называется отделением этого корня.

Процесс отделения корней проводят исходя из физического смысла прикладной задачи, графически, с помощью таблиц значений функции $f(x)$ или при помощи специальной программы отделения корней. Процедура отделения корней основана на известном свойстве непрерывных функций: если функция непрерывна на замкнутом интервале $[a, b]$ и на его концах имеет различные знаки, т.е. $f(a) \cdot f(b) < 0$, то между точками a и b имеется хотя бы один корень уравнения $f(x) = 0$. Если при этом функция $f(x)$ на отрезке $[a, b]$ монотонна, то указанный корень единственный.

Процесс определения корней алгебраических и трансцендентных уравнений состоит из двух

этапов:

- отделение корней, — т.е. определение интервалов изоляции $[a, b]$, внутри которого лежит каждый корень уравнения;
- уточнение корней, — т.е. сужение интервала $[a, b]$ до величины равной заданной степени точности ε.

Для алгебраических и трансцендентных уравнений пригодны одни и те же методы уточнения приближенных значений действительных корней:

- метод половинного деления (метод дихотомии);
- метод простых итераций;
- метод Ньютона (метод касательных);
- модифицированный метод Ньютона (метод секущих);
- метод хорд и др.

8.1.1 Метод половинного деления

Рассмотрим следующую задачу: дано нелинейное уравнение $f(x) = 0$, необходимо найти корень уравнения, принадлежащий интервалу $[a, b]$, с заданной точностью ε.

Для уточнения корня методом половинного деления последовательно осуществляем следующие операции:

- вычисляем значение функции $f(x)$ в точках a и $t = (a + b)/2$;
- если $f(a) \cdot f(t) < 0$, то корень находится в левой половине интервала $[a, b]$, поэтому отбрасываем правую половину интервала и принимаем $b = t$;
- если условие $f(a) \cdot f(t) < 0$ не выполняется, то корень находится в правой половине интервала $[a, b]$, поэтому отбрасываем левую половину интервала $[a, b]$ за счёт присваивания $a = t$.

В обоих случаях новый интервал $[a, b]$ в 2 раза меньший предыдущего.

Процесс сокращения длины интервала неопределённости циклически повторяется до тех пор, пока длина интервала $[a, b]$ не станет равной либо меньшей заданной точности, т.е. $|b - a| \leq \varepsilon$.

Реализация метода половинного деления в виде функции **Maxima** представлена в следующем примере:

- собственно функция $bisect$, в которую передаётся выражение f, определяющее уравнение, которое необходимо решить

```
(%i1)   bisect(f,sp,eps):=block([a,b],a:sp[1],b:sp[2],p:0,
while abs(b-a)>eps do (
        p:p+1, c:(a+b)/2,
        fa:float(subst(a,x,f)), fc:float(subst(c,x,f)),
        if fa*fc<0 then b:c else a:c
),
float(c));
```

$$(\%o1)\ \text{bisect}\,(f, sp, eps) := \text{block}([a, b], a : sp_1, b : sp_2, p : 0,\ \text{while}\,|b - a| > eps\ \text{do}(p : p+1, c : \tfrac{a+b}{2}, fa : \text{float}\,(\text{subst}\,(a, x, f))\,,\ fc : \text{float}\,(\text{subst}\,(c, x, f))\,,\ \text{if}\ fa\,fc < 0\ \text{then}\ b : c\ \text{else}\ a : c),\ \text{float}\,(c))$$

- последовательность команд, организующая обращение к bisect и результаты вычислений

```
(%i2)   f:exp(-x)-x$
a:-1$ b:2$ eps:0.000001$ xrez:bisect(f,[-2,2],eps)$
print("Решение ",xrez," Невязка ",subst(xrez,x,f))$
```

Решение 0.56714344024658 Невязка - 2.34815726529724610^{-7}

$$(\%o2)\ -2.3481572652972\,46\,10^{-7}$$

В представленном примере решается уравнение $e^{-x} - x = 0$. Поиск корня осуществляется на отрезке [-2, 2] с точностью 0.000001.

Следует отметить особенность программирования для **Maxima**, заключающуюся в том, что решаемое уравнение задаётся в виде математического выражения (т.е. фактически текстовой строки). Числовое значение невязки уравнения вычисляется при помощи функции subst, посредством которой выполняется подстановка значений x = a или $x = c$ в заданное выражение. Вычисление невязки после решения осуществляется также путём подстановки результата решения $xrez$ в выражение f.

8.1.2 Метод простых итераций

В ряде случаев весьма удобным приёмом уточнения корня уравнения является метод последовательных приближений (метод итераций).

Пусть с точностью ε необходимо найти корень уравнения $f(x) = 0$, принадлежащий интервалу изоляции $[a, b]$. Функция $f(x)$ и ее первая производная непрерывны на этом отрезке.

Для применения этого метода исходное уравнение $f(x) = 0$ должно быть приведено к виду $x = \varphi(x)$.

В качестве начального приближения может быть выбрана любая точка интервала $[a, b]$.

Далее итерационный процесс поиска корня строится по схеме:

$$x_1 = f(x_0),$$
$$x_2 = f(x_1),$$
$$\ldots$$
$$x_n = f(x_{n-1})$$

В результате итерационный процесс поиска реализуется рекуррентной формулой. Процесс поиска

прекращается, как только выполняется условие $|x_n - x_{n-1}| \le \varepsilon$ или число итераций превысит заданное число N.

Для того, чтобы последовательность $x_1, x_2, ..., x_n$ приближалась к искомому корню, необходимо, чтобы выполнялось условие сходимости $|\varphi'(x)| < 1$.

Пример реализации метода итераций представлен ниже:

```
(%i1)   f:exp(-x)-x$
beta:0.1$ x1:1$ x0:0$ eps:0.000001$ p:0$
while abs(x1-x0)>eps do
                (x0:x1, p:p+1, x1:float(x0+beta*(subst(x0,x,f))))$
print("Число итераций ",p," ","Решение ",float(x1),
" Невязка ",float(abs(x1-x0)))$
```

Число итераций 67 Решение 0.56714848327814
Невязка $9.650298036234517 10_{7}$

8.1.3 Метод Ньютона (метод касательных)

Рассмотренные ранее методы решения нелинейных уравнений являются методами прямого поиска. В них для нахождения корня используется нахождение значения функции в различных точках интервала $[a, b]$.

Метод Ньютона относится к градиентным методам, в которых для нахождения корня используется значение производной.

Рассмотрим нелинейное уравнение $f(x) = 0$, для которого необходимо найти корень на интервале $[a, b]$ с точностью ε.

Метод Ньютона основан на замене исходной функции $f(x)$, на каждом шаге поиска касательной, проведённой к этой функции. Пересечение касательной с осью X дает приближение корня.

Выберем начальную точку $x_0 = b$ (конец интервала изоляции). Находим значение функции в этой точке и проводим к ней касательную, пересечение которой с осью X дает первое приближение корня x_1:

$x_1 = x_0 - h_0,$ где
$h_0 = \frac{f(x_0)}{\mathrm{tg}(\alpha)} = \frac{f(x_0)}{f'(x_0)}.$

Поэтому $$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

В результате, итерационный процесс схождения к корню реализуется рекуррентной формулой

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Процесс поиска продолжаем до тех пор, пока не выполнится условие: $|x_{n+1} - x_n| \le \varepsilon$, откуда $|\frac{f(x_n)}{f'(x_n)}| \le \varepsilon$.

Метод обеспечивает быструю сходимость, если выполняется условие: $f(x_0) \cdot f''(x_0) > 0$, т.е. первую касательную рекомендуется проводить в той точке интервала $[a, b]$, где знаки функции $f(x_0)$ и ее кривизны $f''(x_0)$ совпадают.

Пример реализации метода Ньютона в **Maxima** представлен ниже:

```
(%i1)   newton(f,x0,eps):=block([df,xn,xn0,r,p],
        xn0:x0, df:diff(f,x),
        p:0, r:1,
        while abs(r)>eps do (
                p:p+1, xn:xn0-float(subst(xn0,x,f)/subst(xn0,x,df)),
                print("x0,x1 ",xn0,xn),r:xn-xn0, xn0:xn
        ),
        [xn,p])$
```

Последовательность команд для обращения к функции $newton$ и результаты вычислений представлены в следующем примере:

```
(%i2)   f:exp(-x)-x$
        eps:0.000001$ xrez:newton(f,1,eps)$
        print("Решение ",xrez[1]," Число итераций ",xrez[2],
        " Невязка ",subst(xrez[1],x,f))$
```

$x0, x1$ 1.53788284273999
x0,x1 0.53788284273999 0.56698699140541
$x0, x1$ 0.56698699140541 0.56714328598912
x0,x1 0.56714328598912 0.56714329040978
Решение 0.56714329040978 Число итераций 4 Невязка 0.0

Особенности приведённого примера — промежуточная печать результатов и возвращаемое значение в виде списка, что позволяет одновременно получить как значение корня, так и необходимое для достижения заданной точности число итераций. Существенному уменьшению числа итераций способствует и аналитическое вычисление производной.

8.1.4 Модифицированный метод Ньютона (метод секущих)

В этом методе для вычисления производных на каждом шаге поиска используется численное дифференцирование по формуле:

$$f'(x) = \frac{\Delta f(x)}{\Delta x}$$

Тогда рекуррентная формула метода Ньютона приобретёт вид:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = x_n - \frac{f(x_n)\Delta x}{\Delta f(x_n)} =$$
$$= x_n - \frac{f(x_n)\Delta x}{f(x_n+\Delta x)-f(x_n)},$$

где $\Delta x \approx \varepsilon$.

Реализация данного метода в **Maxima** представлено ниже:

```
(%i1)   secant(f,sp,eps):=block([x0,x1,d,y,r],
        x0:sp[1],x1:sp[2],
        p:0, r:x1-x0, d:float(subst(x0,x,f)),
        while abs(r)>eps do (
                p:p+1, y:float(subst(x1,x,f)), r:r/(d-y)*y,
                d:y, x1:x1+r
        ),
        x1)$
(%i2)   f:exp(-x)-x$
        eps:0.000001$ xrez:secant(f,[-2,2],eps)$
        print("Решение ",xrez," Невязка ",subst(xrez,x,f))$
```

$$\text{Решение} \quad 0.56714329040978 \quad \text{Невязка} \quad -1.1102230246251565\,10^{-16}$$

Особенности программирования для **Maxima**, использованные в этом примере, аналогичны приведённым выше в примере, касающемся метода половинного деления.

8.1.5 Метод хорд

Метод основан на замене функции $f(x)$ на каждом шаге поиска хордой, пересечение которой с осью X дает приближение корня.

При этом в процессе поиска семейство хорд может строится:

1. при фиксированном левом конце хорд, т.е. $z = a$, тогда начальная точка $x0 = b$;
2. при фиксированном правом конце хорд, т.е. $z = b$, тогда начальная точка $x0 = a$.

В результате итерационный процесс схождения к корню реализуется рекуррентной формулой:

$$x_{n+1} = x_n - \frac{f(x_n)}{f(x_n)-f(a)}(x_n - a) \quad \text{для случая а);}$$
$$x_{n+1} = x_n - \frac{f(x_n)}{f(x_n)-f(b)}(x_n - b) \quad \text{для случая б);}$$

Процесс поиска продолжается до тех пор, пока не выполнится условие

$$|x_{n+1} - x_n| \le \varepsilon \text{ или } |h| \le \varepsilon.$$

Метод обеспечивает быструю сходимость, если $f(z) \cdot f''(z) > 0$, т.е. хорды фиксируются в том конце интервала $[a, b]$, где знаки функции $f(z)$ и ее кривизны $f''(z)$ совпадают.

8.2 Численное интегрирование

Задача численного интегрирования состоит в замене исходной подинтегральной функции $f(x)$, для которой трудно или невозможно записать первообразную в аналитике, некоторой аппроксимирующей функцией $\varphi(x)$. Такой функцией обычно является полином (кусочный полином) $\varphi(x) = \sum_{i=1}^{n} c_i \phi_i(x)$.

Таким образом

$$\mathrm{I} = \int_a^b f(x)dx = \int_a^b \phi(x)dx + R,$$

$$R = \int_a^b r(x)dx$$

где a — априорная погрешность метода на интервале интегрирования, а $r(x)$ — априорная погрешность метода на отдельном шаге интегрирования.

8.2.1 Обзор методов интегрирования

Методы вычисления однократных интегралов называются квадратурными (для кратных интегралов — кубатурными), и делятся на следующие группы:

- Методы Ньютона-Котеса. Здесь $\varphi(x)$ — полином различных степеней. Сюда относятся метод прямоугольников, трапеций, Симпсона.
- Методы статистических испытаний (методы Монте-Карло). Здесь узлы сетки для квадратурного или кубатурного интегрирования выбираются с помощью датчика случайных чисел, ответ носит вероятностный характер. В основном применяются для вычисления кратных интегралов.
- Сплайновые методы. Здесь $\varphi(x)$ — кусочный полином с условиями связи между отдельными полиномами посредством системы коэффициентов.
- Методы наивысшей алгебраической точности. Обеспечивают оптимальную расстановку узлов сетки интегрирования и выбор весовых коэффициентов $\rho(x)$ в задаче $\int_a^b \phi(x)\rho(x)dx$ (характерный пример — метод Гаусса).

8.2.2 Метод прямоугольников

Различают метод левых, правых и средних прямоугольников. Суть метода ясна из рисунка. На каждом шаге интегрирования функция аппроксимируется полиномом нулевой степени — отрезком, параллельным оси абсцисс.

Формулы метода прямоугольников можно получить из анализа разложения функции $f(x)$ в ряд Тейлора вблизи некоторой точки $x = x_i$:

$$f(x)|_{x=x_i} = f(x_i) + (x - x_i)\, f'(x_i) + \frac{(x - x_i)^2}{2!} f''(x_i) + \dots.$$

Рассмотрим диапазон интегрирования от x_i до $x_i + h$, где h — шаг интегрирования. Вычислим интеграл от исследуемой функции на этом промежутке:

$$\int_{x_i}^{x_i+h} f(x)dx = x \cdot f(x_i)|_{x_i}^{x_i+h} + \frac{(x-x_i)^2}{2} f'(x_i)|_{x_i}^{x_i+h} +$$
$$+ \frac{(x-x_i)^3}{3\cdot 2!} f''(x_i)|_{x_i}^{x_i+h} + \dots =$$
$$= f(x_i)h + \frac{h^2}{2} f'(x_i) + O(h^3) = f(x_i)h + r_i.$$

таким образом, на базе анализа ряда Тейлора получена формула правых (или левых) прямоугольников и априорная оценка погрешности r на отдельном шаге интегрирования. Основной критерий, по которому судят о точности алгоритма — степень при величине шага в формуле априорной оценки погрешности. В случае равного шага h на всем диапазоне интегрирования общая формула имеет вид

$$\int_a^b f(x)dx = h\sum_{i=0}^{n-1} f(x_i) + R,$$

где n — число разбиений интервала интегрирования,

$R = \sum_{i=0}^{n-1} r_i = \frac{h}{2}\cdot h\sum_{i=0}^{n-1} f'(x_i) = \frac{h}{2}\int_a^b f'(x)dx$. Полученная оценка справедлива при наличии непрерывной производной подинтегральной функции $f'(x)$.

8.2.3 Метод средних прямоугольников

Здесь на каждом интервале значение функции считается в средней точке отрезка $[x_i, x_i + h]$, то есть $\int_{x_i}^{x_i+h} f(x)dx = hf(\bar{x}) + r_i$.

Разложение функции в ряд Тейлора показывает, что в случае средних прямоугольников точность метода существенно выше:

$$r = \frac{h^3}{24}f''(\bar{x}),\ R = \frac{h^2}{24}\int_a^b f''(x)dx.$$

Пример функции **Maxima**, реализующей метод средних прямоугольников, представлен ниже:

```
(%i1)   intpr(f,n,a,b):=block([h,i,s,_x],h:(b-a)/n, _x:a+h/2, s:0,
        for i:1 thru n do (s:s+float(subst(_x,x,f)),_x:_x+h),s:s*h)$
(%i2)   intpr(x^2,100,-1,1);
```

(%o2) 0.6666

8.2.4 Метод трапеций

Аппроксимация в этом методе осуществляется полиномом первой степени. На единичном интервале

$$\int_{x_i}^{x_i+h} f(x)dx = \frac{h}{2}\left(f(x_i) + f(x_i + h)\right) + r_i.$$

В случае равномерной сетки (h = const)

$$\int_a^b f(x)dx = h\left(\frac{1}{2}f(x_0) + \sum_{i=1}^{n-1} f(x_i) + \frac{1}{2}f(x_n)\right) + R$$

При этом $r_i = -\frac{h^3}{12}f''(x_i)$, а $R = -\frac{h^3}{12}\int_a^b f''(x)dx$.

Погрешность метода трапеций в два раза выше, чем у метода средних прямоугольников. Однако на практике найти среднее значение на элементарном интервале можно только у функций, заданных аналитически (а не таблично), поэтому использовать метод средних прямоугольников удаётся далеко не всегда. В силу разных знаков погрешности в формулах трапеций и средних прямоугольников истинное значение интеграла обычно лежит между двумя этими оценками.

Пример реализации метода трапеций в виде функции приведен ниже:

```
(%i1)   f:x^2;
```

$$(\%o1)\ x^2$$

```
(%i2)   inttrap(f,n,a,b):=block([h,i,s],h:(b-a)/n,
        s:(float(subst(a,x,f))+float(subst(b,x,f)))/2,
        for i:1 thru n-1 do (s:s+float(subst(a+i*h,x,f))), s:s*h)$
(%i3)   inttrap(f,100,-1,1);
```

$$(\%o3)\ 0.6668$$

Подинтегральная функция задаётся в виде выражения **Maxima.** Выражение, определяющее подинтегральную функцию, можно задавать и непосредственно при обращении к методу, как в следующем примере:

```
(%i4)   inttrap(x*sin(x),100,-1,1);
```

$$(\%o4)\ 0.60242947746101$$

8.2.5 Метод Симпсона

При использовании данного метода подинтегральная функция $f(x)$ заменяется интерполяционным полиномом второй степени $P(x)$ — параболой, проходящей через три соседних узла. Рассмотрим два шага интегрирования ($h = const = x_{i+1} - x_i$), то есть три узла x_0, x_1, x_2, через которые проведем параболу, воспользовавшись уравнением Ньютона:

$$P(x) = f_0 + \frac{x - x_0}{h}(f_1 - f_0) + \frac{(x - x_0)(x - x_1)}{2h^2}(f_0 - 2f_1 + f_2).$$

Пусть $z = x - x_0$, тогда

$$P(z) = f_0 + \frac{z}{h}(f_1 - f_0) + \frac{z(z-h)}{2h^2}(f_0 - 2f_1 + f_2) =$$
$$= f_0 + \frac{z}{2h}(-3f_0 + 4f_1 - f_2) + \frac{z^2}{2h^2}(f_0 - 2f_1 + f_2)$$

Воспользовавшись полученным соотношением, вычислим интеграл по данному интервалу:

$$\int_{x_0}^{x_2} P(x)dx = \int_0^{2h} P(z)dz =$$
$$= 2hf_0 + \frac{(2h)^2}{4h}(-3f_0 + 4f_1 - f_2) + \frac{(2h)^3}{6h^2}(f_0 - 2f_1 + f_2) =$$
$$= 2hf_0 + h(-3f_0 + 4f_1 - f_2) + \frac{4h}{3}(f_0 - 2f_1 + f_2) =$$
$$= \frac{h}{3}(6f_0 - 9f_0 + 12f_1 - 3f_2 + 4f_0 - 8f_1 + 4f_2).$$

$$\int_{x_0}^{x_2} f(x)dx = \frac{h}{3}(f_0 + 4f_1 + f_2) + r$$

В итоге .

Для равномерной сетки и чётного числа шагов n формула Симпсона принимает вид:

$$\int_a^b f(x)dx = \frac{h}{3}(f_0 + 4f_1 + 2f_2 + 4f_3 + \cdots + 2f_{n-2} + 4f_{n-1} + f_n) + R,$$

$r = -\frac{h^5}{90} f^{IV}(x_i)$ $R = -\frac{h^4}{180}\int_a^b f^{IV}(x)dx$

где , а в предположении непрерывности четвёртой производной подинтегральной функции.

Реализация метода Симпсона средствами **Maxima** представлена в следующем примере:

```
(%i1)   intsimp(f,n,a,b):=block([h,h2,i,_x,s],h:(b-a)/n, h2:h/2,
        s:(float(subst(a,x,f))+float(subst(b,x,f)))/2+
                                2*float(subst(a+h2,x,f)),
        _x:a,
        for i:1 thru n-1 do (_x:_x+h,
                s:s+2*float(subst(_x+h2,x,f))+float(subst(_x,x,f))),
        s:s*h/3)$
(%i2)   intsimp(x^2,100,-1,1);
```

$$(\%o2)\ 0.666666666666667$$

8.3 Методы решения систем линейных уравнений

8.3.1 Общая характеристика и классификация методов решения

Рассмотрим систему линейных алгебраических уравнений (сокращенно — СЛАУ):

$$A \cdot \bar{x} = \bar{f},\ (8.1)$$

где A — матрица $m \times m,\ \bar{x} = (x_1, x_2, \ldots, x_m)^T$ — искомый вектор, $\bar{f} = (f_1, f_2, \ldots, f_m)^T$ — заданный вектор. Будем предполагать, что определитель матрицы А отличен от нуля, т.е. решение системы (8.1) существует.

Методы численного решения системы (8.1) делятся на две группы: прямые методы ("точные") и итерационные методы.

Прямыми методами называются методы, позволяющие получить решение системы (8.1) за конечное число арифметических операций. К этим методам относятся метод Крамера, метод

Гаусса, LU-метод и т.д.

Итерационные методы (методы последовательных приближений) состоят в том, что решение системы (8.1) находится как предел последовательных приближений $\bar{x}^{(n)}$ при $n \to \infty$, где n — номер итерации. При использовании методов итерации обычно задается некоторое малое число ε и вычисления проводятся до тех пор, пока не будет выполнена оценка $\left\|\bar{x}^{(n)} - \bar{x}\right\| < \varepsilon$. К этим методам относятся метод Зейделя, Якоби, метод верхних релаксаций и т.д.

Следует заметить, что реализация прямых методов на компьютере приводит к решению с погрешностью, т.к. все арифметические операции над переменными с плавающей точкой выполняются с округлением. В зависимости от свойств матрицы исходной системы эти погрешности могут достигать значительных величин.

8.3.2 Метод Гаусса

Запишем систему (8.1), в развернутом виде

$$\begin{aligned}
&a_{11}x_1 + a_{12}x_2 + \cdots + a_{1m}x_m = f_1,\\
&a_{21}x_1 + a_{22}x_2 + \cdots + a_{2m}x_m = f_2,\\
&\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\\
&a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mm}x_m = f_m.
\end{aligned}$$

Метод Гаусса состоит в последовательном исключении неизвестных из этой системы.

Предположим, что $a_{11} \neq 0$. Последовательно умножая первое уравнение на $-\dfrac{a_{i1}}{a_{11}}$ и складывая с i-м уравнением, исключим x_1 из всех уравнений кроме первого. Получим систему

$$\begin{aligned}
&a_{11}x_1 + a_{12}x_2 + \cdots + a_{1m}x_m = f_1,\\
&a^{(1)}_{22}x_2 + \cdots + a^{(1)}_{2m}x_m = f^{(1)}_2,\\
&\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\\
&a^{(1)}_{m2}x_2 + \cdots + a^{(1)}_{mm}x_m = f^{(1)}_m.
\end{aligned}$$

где

$$a^{(1)}_{ij} = a_{ij} - \frac{a_{i1}a_{1j}}{a_{11}}, \quad f^{(1)}_i = f_i - \frac{a_{i1}f_1}{a_{11}}, \quad i,j = 2,3,\ldots,m.$$

Аналогичным образом из полученной системы исключим x_2. Последовательно, исключая все неизвестные, получим систему треугольного вида

$$\begin{aligned}
&a_{11}x_1 + a_{12}x_2 + \cdots + a_{1k}x_k + \cdots + a_{1m}x_m = f_1,\\
&a^{(1)}_{22}x_2 + \cdots + a^{(1)}_{2k}x_k + \cdots + a^{(1)}_{2m}x_m = f^{(1)}_2,\\
&\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots,\\
&a^{(m-1)}_{m-1,m-1}x_{m-1} + a^{(m-1)}_{m-1,m}x_m = f^{(m-1)}_{m-1},\\
&a^{(m-1)}_{m,m}x_m = f^{(m-1)}_m.
\end{aligned}$$

Описанная процедура называется прямым ходом метода Гаусса. Заметим, что ее выполнение было возможно при условии, что все $a_{i,i}^{(l)},\ l = 1, 2, \ldots, m-1$ не равны нулю. Выполняя последовательные подстановки в последней системе, (начиная с последнего уравнения) можно получить все значения неизвестных.

$$x_m = \frac{f_m^{(m-1)}}{a_{m,m}^{(m-1)}},$$
$$x_i = \frac{1}{a_{i,i}^{(i-1)}}(f_i^{(i-1)} - \sum_{j=i-1}^{m} a_{ij}^{(i-1)} x_j).$$

Эта процедура получила название обратный ход метода Гаусса.

Метод Гаусса может быть легко реализован на компьютере. При выполнении вычислений, как правило, промежуточные значения матрицы A не представляют интереса. Поэтому численная реализация метода сводится к преобразованию элементов массива размерности $m \times (m+1)$, где $m+1$-й столбец содержит элементы правой части системы.

Для контроля ошибки реализации метода используются, так называемые, контрольные суммы. Схема контроля основывается на следующем очевидном положении. Увеличение значения всех неизвестных на единицу равносильно замене данной системы контрольной системой, в которой свободные члены равны суммам всех коэффициентов соответствующей строки. Создадим дополнительный столбец, хранящий сумму элементов матрицы по строкам. На каждом шаге реализации прямого хода метода Гаусса будем выполнять преобразования и над элементами этого столбца, и сравнивать их значение с суммой по строке преобразованной матрицы. В случае не совпадения значений счет прерывается.

Один из основных недостатков метода Гаусса связан с тем, что при его реализации накапливается вычислительная погрешность.

Для систем порядка m число действий умножения и деления близко $\frac{m^3}{3}$ и быстро растет с величиной m.

Для того, чтобы уменьшить рост вычислительной погрешности применяются различные модификации метода Гаусса. Например, метод Гаусса с выбором главного элемента по столбцам, в этом случае на каждом этапе прямого хода строки матрицы переставляются таким образом, чтобы диагональный угловой элемент был максимальным. При исключении соответствующего неизвестного из других строк деление будет производиться на наибольший из возможных коэффициентов и, следовательно, относительная погрешность будет наименьшей.

Пример реализации метода Гаусса в **Maxima** приведен в функции ниже (применен метод без выбора главного элемента):

```
(%i1)   gauss(a0,b0,n):=block([a,b,i,j,k,d],
        a:copymatrix (a0), b:copymatrix(b0), x:copymatrix(b0),
        for i:1 thru n-1 do
        (
                for k:i+1 thru n do
                (
                        d:a[k,i]/a[i,i],
                        for j:i+1 thru n do (a[k,j]:a[k,j]-a[i,j]*d),
                        b[k,1]:b[k,1]-b[i,1]*d
                )
```

```
        ),
        for i:n thru 1 step -1 do
        (
                for j:i+1 thru n do (
                        b[i,1]:b[i,1]-a[i,j]*x[j,1]),
                x[i,1]:b[i,1]/a[i,i]
        ),
        x)$
```

Пример обращения к функции, реализующей метод Гаусса:

```
(%i2)   aa:matrix([3,1,1],[1,3,1],[1,1,3]); bb:matrix([6],[6],[8]);
        zz:gauss(aa,bb,3);
```

$$(\%o2)\ \begin{pmatrix}3 & 1 & 1\\ 1 & 3 & 1\\ 1 & 1 & 3\end{pmatrix}\ (\%o3)\ \begin{pmatrix}6\\ 6\\ 8\end{pmatrix}\ (\%o4)\ \begin{pmatrix}1\\ 1\\ 2\end{pmatrix}$$

Проверка вычислений показывает, что перемножение матрицы A на вектор решения zz дает вектор, совпадающий с вектором правых частей bb .

```
(%i5)   aa.zz;
```

$$(\%o5)\ \begin{pmatrix}6.0\\ 6.0\\ 8.0\end{pmatrix}$$

Существует метод Гаусса с выбором главного элемента по всей матрице. В этом случае переставляются не только строки, но и столбцы. Использование модификаций метода Гаусса приводит к усложнению алгоритма увеличению числа операций и соответственно к росту времени счета.

Выполняемые в методе Гаусса преобразования прямого хода, приведшие матрицу A системы к треугольному виду позволяют вычислить определитель матрицы

$$\det A = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ 0 & a_{22}^{(1)} & \dots & a_{2m}^{(1)} \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & a_{m,m}^{(m-1)} \end{vmatrix} = a_{11} \cdot a_{22}^{(1)} \dots a_{m,m}^{(m-1)}$$

Метод Гаусса позволяет найти и обратную матрицу. Для этого необходимо решить матричное уравнение

$$A \cdot X = E,$$

где E – единичная матрица. Его решение сводится к решению m систем

$$A\bar{x}^{(j)} = \bar{\delta}^{(j)}, \quad j = 1, 2, \dots, m,$$

у вектора $\bar{\delta}^{(j)}$ j –я компонента равна единице, а остальные компоненты равны нулю.

8.3.3 Метод квадратного корня

Метод квадратного корня основан на разложении матрицы A в произведение

$$A = S^T S,$$

где S — верхняя треугольная матрица с положительными элементами на главной диагонали, S^T — транспонированная к ней матрица.

Пусть A — матрица размером $m \times m$. Тогда

$$\left(S^T S\right)_{ij} = \sum_{k=1}^{m} s_{ik}^T s_{kj} \quad (8.2)$$

Из условия (8.2) получаются уравнения

$$\sum_{k=1}^{m} s_{ik}^T s_{kj} = a_{ij}, \qquad i, j = 1, 2, \ldots .., m \quad (8.3)$$

Так как матрица A симметричная, не ограничивая общности, можно считать, что в системе (8.3) выполняется неравенство $i \leq j$. Тогда (8.3) можно переписать в виде

$$\sum_{k=1}^{i-1} s_{ik}^T s_{kj} + s_{ii} s_{ij} + \sum_{k=i+1}^{m} s_{ik}^T s_{kj} = a_{ij}$$

$$s_{ii} s_{ij} + \sum_{k=1}^{i-1} s_{ik}^T s_{kj} = a_{ij}, \quad i \leqslant j.$$

В частности, при $i = j$ получится

$$|s_{ii}|^2 = a_{ii} - \sum_{k=1}^{i-1} |s_{ki}|^2$$

$$s_{ii} = \left(\left| a_{ii} - \sum_{k=1}^{i-1} |s_{ki}|^2 \right| \right)^{1/2}$$

Далее, при $i < j$ получим

$$s_{ij} = \frac{a_{ij} - \sum_{k=1}^{i-1} s_{ik}^T s_{kj}}{s_{ii}}$$

По приведённым формулам находятся рекуррентно все ненулевые элементы матрицы S.

Обратный ход метода квадратного корня состоит в последовательном решении двух систем уравнений с треугольными матрицами.

$$S^T y = f,$$
$$Sx = y.$$

Решения этих систем находятся по рекуррентным формулам:

$$\begin{cases} y_i = \dfrac{f_i - \sum\limits_{k=1}^{i-1} s_{ki} y_k}{s_{ii}}, \quad i = 2, 3, \ldots, m \\ y_1 = \dfrac{f_1}{s_{11}} \end{cases}$$

$$\begin{cases} x_i = \dfrac{y_i - \sum\limits_{k=i+1}^{m} s_{ik} x_k}{s_{ii}}, \quad i = m-1, m-2, \ldots, 1 \\ x_m = \dfrac{y_m}{s_{mm}} \end{cases}$$

Всего метод квадратного корня при факторизации $A = S^T S$ требует примерно $\dfrac{m^3}{3}$ операций умножения и деления и m операций извлечения квадратного корня.

Пример функции, реализующей метод квадратного корня:

```
(%i1)   holetsk(a0,b0):=block([L,Lt,x,y,i,j,k,n],
        n:length(a0), L:zeromatrix(n,n),
        for i:1 thru n do (
                for j:1 thru i-1 do (
                s:0,
                for k:1 thru j-1 do (s:s+L[i,k]*L[j,k]),
                L[i,j]:1/L[j,j]*(a0[i,j]-s)
                ),
                s:0,
                for k:1 thru i-1 do (s:s+L[i,k]^2),
                L[i,i]:sqrt(a0[i,i]-s)
                ),Lt:transpose(L),
        y:zeromatrix(n,1), x:zeromatrix(n,1),
        for i:1 thru n do (
                s:0,
                for k:1 thru i-1 do (s:s+L[i,k]*y[k,1]),
                        y[i,1]:(b0[i,1]-s)/L[i,i]
        ),
        for i:n thru 1 step -1 do (
                s:0,
                for k:n thru i+1 step -1 do (s:s+Lt[i,k]*x[k,1]),
                x[i,1]:(y[i,1]-s)/Lt[i,i]
        ),x
        )$
```

Тест данной функции (решение системы $Ax = B, B$ — матрица $n \times 1,\ A$ — квадратная симметричная матрица $n \times n$, результат решения — вектор $n \times 1$):

```
(%i2)   A:matrix([4,1,1],[1,4,1],[1,1,4])$ B:matrix([1],[1],[1])$
```

Результаты вычислений:

```
(%i4)   x:holetsk(A,B);
```

$$(\%o4)\ \begin{pmatrix} \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \end{pmatrix}$$

Проверка:

```
(%i5)   A.x;
```

$$(\%o5)\ \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

8.3.4 Корректность постановки задачи и понятие обусловленности

При использовании численных методов для решения тех или иных математических задач необходимо различать свойства самой задачи и свойства вычислительного алгоритма, предназначенного для ее решения.

Говорят, что задача поставлена корректно, если решение существует и единственно и если оно непрерывно зависит от входных данных. Последнее свойство называется также устойчивостью относительно входных данных.

Корректность исходной математической задачи еще не гарантирует хороших свойств численного метода ее решений и требует специального исследования.

Известно, что решение задачи (8.1) существует тогда и только тогда, когда $\det A \neq 0$. В этом случае можно определить обратную матрицу A^{-1} и решение записать в виде $\bar{x} = A^{-1}\bar{f}$.

Исследование устойчивость задачи (8.1) сводится к исследованию зависимости ее решения от правых частей $\bar{f}$ и элементов a_{ij} матрицы A. Для того чтобы можно было говорить о непрерывной зависимости вектора решений от некоторых параметров, необходимо на множестве m-мерных векторов принадлежащих линейному пространству $\mathbb{H}$, ввести метрику.

В линейной алгебре предлагается определение множества метрик l_p — норма

$$\|\bar{x}\|_p = \left(\sum_{i=1}^{m} |x_i|^p \right)^{1/p}$$

из которого легко получить наиболее часто используемые метрики

- при $p = 1,\ \|\bar{x}\|_1 = \sum_{i=1}^{m} |x_i|$,
- при $p = 2,\ \|\bar{x}\|_2 = \left(\sum_{i=1}^{m} |x_i|^2 \right)^{1/2}$,
- при

$$p \to \infty,\ \|\bar{x}\|_\infty = \lim_{p\to\infty} \left(\sum_{i=1}^{m} |x_i|^p \right)^{1/p}$$

.

Подчиненные нормы матриц определяемые как $\|A\| = \sup_{0\neq x\in\mathbb{H}} \frac{\|A\bar{x}\|}{\|\bar{x}\|}$, соответственно записываются в следующем виде:

$$\|A\|_1 = \max_{1\leqslant j\leqslant m} \sum_{i=1}^{m} |a_{ij}|,$$

$$\|A\|_\infty = \max_{1\leqslant i\leqslant m} \sum_{i=1}^{m} |a_{ij}|,$$

$$\|A\|_2 = \sqrt{\rho(A^T A)} = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{m} a_{ij}^2}.$$

Обычно рассматривают два вида устойчивости решения системы (8.1):

- по правым частям;
- по коэффициентам системы (8.1) и по правым частям.

Наряду с исходной системой (8.1) рассмотрим систему с "возмущенными" правыми частями

$$A \cdot \tilde{x} = \tilde{f},$$

где $\tilde{f} = \bar{f} + \delta\bar{f}$ возмущенная правая часть системы, а $\tilde{x} = \bar{x} + \delta\bar{x}$ возмущенное решение.

Можно получить оценку, выражающую зависимость относительной погрешности решения от относительной погрешности правых частей

$$\frac{\|\delta\bar{x}\|}{\|\bar{x}\|} \leqslant M_A \frac{\|\delta\bar{f}\|}{\|\bar{f}\|},$$

где $M_A = \|A\| \cdot \left\|A^{-1}\right\|$ — число обусловленности матрицы A (в современной литературе это число обозначают как $cond(A)$). Если число обусловленности велико ($M_A \sim 10^k,\ \ k > 2$), то говорят, что матрица A плохо обусловлена. В этом случае малые возмущения правых частей системы (8.1), вызванные либо неточностью задания исходных данных, либо вызванные погрешностями вычисления существенно влияют на решение системы.

Если возмущение внесено в матрицу A, то для относительных возмущений решения имеет место следующая оценка:

$$\frac{\|\delta\bar{x}\|}{\|\bar{x}\|} \leqslant \frac{M_A}{1 - M_A\dfrac{\|\delta A\|}{\|A\|}}\left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta\bar{f}\|}{\|\bar{f}\|}\right).$$

В **Maxima** матричные нормы вычисляются посредством функции mat_norm. Синтаксис вызова: $mat_norm(M, type)$, где M — матрица, $type$ — тип нормы, $type$ может быть равен 1 (норма $\|A\|_1$), inf (норма $\|A\|_\infty$), $frobenius$ (норма $\|A\|_2$).

Пример вычисления указанных видов нормы в **Maxima**:

```
(%i1)   A:matrix([1,2,3],[4,5,6],[7,8,9]);
```

$$(\%o1)\ \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

```
(%i2)   mat_norm(A,1);
```

$$(\%o2)\ 18$$

```
(%i3)   mat_norm(A,inf);
```

$$(\%o3)\ 24$$

```
(%i4)   mat_norm(A,frobenius);
```

$$(\%o4)\ \sqrt{285}$$

Вычислим число обусловленности для плохо и хорошо обусловленных матриц:

```
(%i1)   A:matrix([1,1],[0.99,1]);
```

$$(\%o1)\ \begin{pmatrix} 1 & 1 \\ 0.99 & 1 \end{pmatrix}$$

```
(%i2)   nrA:mat_norm(A,frobenius);
```

$$(\%o2)\ 1.995018796903929$$

```
(%i3)   A1:invert(A);
```

$$(\%o3)\ \begin{pmatrix} 99.99999999999992 & -99.99999999999992 \\ -98.99999999999992 & 99.99999999999992 \end{pmatrix}$$

```
(%i4)   nrA1:mat_norm(A1,frobenius);
```

$$(\%o4)\ 199.50187969039 27$$

```
(%i5)   MA:nrA*nrA1;
```

$$(\%o5)\ 398.0099999999997$$

Таким образом, для плохо обусловленной матрицы число обусловленности достигает почти 400.

Аналогичным путём (с использованием нормы Фробениуса) для матрицы $B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ число обусловленности составило $MB = 2$.

8.3.5 О вычислительных затратах

Один из важных факторов предопределяющих выбор того или иного метода при решении конкретных задач, является вычислительная эффективность метода. Учитывая, что операция сложения выполняется намного быстрее, чем операция умножения и деления, обычно ограничиваются подсчетом последних. Для решения СЛАУ методом Гаусса без выбора главного элемента требуется $\frac{m^3}{3} + m^2 - \frac{m}{3}$ умножений и делений, решение СЛАУ методом квадратного корня требует $\frac{m^3}{6} + \frac{3m^2}{2} + \frac{m}{3}$ и m операций извлечения корней. При больших значениях размерности m, можно сказать, что вычислительные затраты на операции умножения и деления в методе Гаусса составляют величину $O\left(\frac{m^3}{3}\right)$, в методе квадратных корней $O\left(\frac{m^3}{6}\right)$.

8.4 Итерационные методы

В приближенных или итерационных методах решение системы линейных алгебраических уравнений является пределом итерационной последовательности, получаемой с помощью этих методов. К ним относятся: метод простой итерации, метод Зейделя и др. Итерационные методы выгодны для системы специального вида, со слабо заполненной матрицей очень большого вида порядка $10^3 \ldots 10^5$.

Для итерационных методов характерно то, что они требуют начальных приближений значений неизвестных, решение ищется в виде последовательности, постепенно улучшающихся приближений, и кроме того, итерационный процесс должен быть сходящимся. В вычислительной практике процесс итерации обычно продолжается до тех пор, пока два последовательных приближения не совпадут в пределах заданной точности.

8.4.1 Матричная формулировка итерационных методов решения систем линейных уравнений

При использовании СКМ **Maxima** вполне обосновано использование и матричной формулировки итерационных методов.

Рассмотрим решение системы $Ax = f$ (A — квадратная матрица, f — вектор правых частей, x — вектор неизвестных). Обозначим $A = L + D + U$, где L — нижняя треугольная матрица с нулевыми диагональными элементами; D — диагональная матрица; U — верхняя треугольная матрица с нулевыми диагональными элементами.

Для решения этой системы рассмотрим итерационный процесс

$$x^{i+1} = x^i - H_i(Ax^i - f),$$

где $\det(H_i) \neq 0$, или

$$x^{i+1} = P_i x^i + d_i, \qquad x(0) = x_0,$$

где $P_i = I - P_i A$ — оператор i-го шага итерационного процесса; $d_i = P_i A$.

Итерационный процесс сходящийся, если последовательность $\{x_i\}$ сходится к решению x^* при любом x_0.

Если матрица не зависит от номера итерации, итерационный процесс называется стационарным:

$$x^{i+1} = Px^i + d. \quad (8.4)$$

Необходимым и достаточным условием сходимости стационарного процесса является выполнение условия $\rho(P) < 1$, где $\rho(P)$ — спектральный радиус матрицы P (наибольшее по модулю собственное число матрицы P).

С использованием введённых обозначений метод простой итерации (метод Якоби) даётся формулой:

$$P = I - D^{-1}A, \qquad D = diag(a_{ii}), \qquad H = D^{-1},$$

а метод Гаусса–Зейделя — формулой:

$$P = -(D + L)^{-1}U, \qquad H = (D + L)^{-1}.$$

Рассмотрим поэлементные расчетные соотношения для методов Якоби и Гаусса—Зейделя.

Все элементы главной диагонали матрицы $I = D^{-1}A$ равны нулю, остальные элементы равны $-\frac{a_{ij}}{a_{ii}}, \quad i, j = \overrightarrow{1, n}$. Свободный член уравнения (8.4) равен $\frac{f_i}{a_{ii}}$.

Таким образом, для метода Якоби итерационный процесс записывается в виде $x^{k+1} = Cx^k +$,

где $C_{ij} = -\frac{a_{ij}}{a_{ii}}; \ k = 0, 1, \ldots, \ i, j = \overrightarrow{1, n}; \ E_i = \frac{f_i}{a_{ii}}$.

Для метода Гаусса–Зейделя $x^{k+1} = -(D + L)^{-1}Ux^k + (D + L)^{-1}f$, или $(D + L)x^{k+1} = -Ux^k + b$, $x^{k+1} = Bx^{k+1} + Ex^k + e$, где

$$B = \begin{bmatrix} 0 & 0 & \ldots & 0 \\ c_{21} & 0 & \ldots & 0 \\ c_{31} & c_{32} & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots \\ c_{n-1,1} & c_{n-1,2} & \ldots & 0 \\ c_{n1} & c_{n2} & \ldots & 0 \end{bmatrix}, \text{ и } E = \begin{bmatrix} 0 & c_{21} & \ldots & \ldots & c_{1n} \\ 0 & 0 & \ldots & \ldots & c_{2n} \\ 0 & & \ddots & & \\ \vdots & & & \ddots & c_{n-1,n} \\ 0 & & & & 0 \end{bmatrix}.$$

Рассмотрим решение конкретной системы уравнений $Ax = b$ методом Якоби:

$$\begin{aligned} 8x_1 - x_2 + 2x_3 &= 8, \\ x_1 + 9x_2 + 3x_3 &= 18, \\ 2x_1 - 3x_2 + 10x_3 &= -5. \end{aligned}$$

Вычисляем элементы матрицы B и вектора e :

$$B = \begin{bmatrix} 0 & \frac{1}{8} & -\frac{2}{8} \\ -\frac{1}{9} & 0 & -\frac{3}{9} \\ -0,2 & 0,3 & 0 \end{bmatrix}, \quad e = \begin{bmatrix} 1 \\ 2 \\ 0,5 \end{bmatrix}.$$

Вычислим значения x по формуле $x^{k+1} = Bx^k + e$. Для решения использована следующая последовательность команд Maxima:

- преобразование заданных матриц

```
(%i1)   A:matrix([8,-1,2],[1,9,3],[2,-3,10])$
        b:matrix([8],[18],[5])$
        A0:matrix([A[1,1],A[1,1],A[1,1]],
                  [A[2,2],A[2,2],A[2,2]],
                  [A[3,3],A[3,3],A[3,3]])$
        B:-A/A0+diagmatrix(3,1)$
        e:b/matrix([A[1,1]],[A[2,2]],[A[3,3]])$ x:e$
```

- собственно вычисление решения

```
(%i7)   xt:float(B.x+e)$
        xt:float(B.xt+e)$
        xt:float(B.xt+e)$
        xt:float(B.xt+e)$
        xt:float(B.xt+e)$
        xt:float(B.xt+e)$
        xt:float(B.xt+e)$
        x0:xt$
        xt:float(B.xt+e)$
        x1:xt$
        r:x1-x0$
        float(r); /* оценка сходимости*/
        float(A.x1-b); /* оценка невязки*/
```

$$(\%o18) \quad \begin{pmatrix} -1.2272477756924971\,10^{-5} \\ -1.3018148195786949\,10^{-4} \\ 6.2047575160040225\,10^{-5} \end{pmatrix}$$

$$(\%o19) \quad \begin{pmatrix} 2.5427663227794994\,10^{-4} \\ 1.738702477211973\,10^{-4} \\ 3.6599949035931445\,10^{-4} \end{pmatrix}$$

8.4.2 Метод простой итерации

Для решения системы линейных алгебраических уравнений (8.1) итерационным методом её необходимо привести к нормальному виду:

$$\bar{x} = P\bar{x} + \bar{g} \quad (8.5)$$

Стационарное итерационное правило получаем, если матрица B и вектор $\bar{g}$ не зависят от номера итерации: $\bar{x}^{k+1} = P\bar{x}^k + \bar{g}$. Нестационарное итерационное правило получаем если матрица B или вектор $\bar{g}$ изменяются с ростом номера итерации: $\bar{x}^{k+1} = P_k\bar{x}^k + \bar{g}^k$.

Стационарное итерационное правило обычно называют методом простой итерации. Предел итерационной последовательности является точным решением системы (8.5) или (8.1).

Для того, чтобы метод простой итерации сходился при любом начальном приближении, необходимо и достаточно, чтобы все собственные значения матрицы B были по модулю меньше единицы.

В силу того, что проверить сформулированное выше условие достаточно сложно на практике применяют следующие достаточные признаки:

- для того чтобы метод простой итерации сходился, достаточно, чтобы какая-либо норма матрицы P была меньше единицы;
- для того чтобы метод простой итерации сходился, достаточно, чтобы выполнялось одно из следующих условий:
  - $\sum_{j=1}^{n} |P_{ij}| < 1,\ i = \overline{1,n}$;
  - $\sum_{i=1}^{n} |P_{ij}| < 1,\ j = \overline{1,n}$;
  - $\sum_{i,j=1}^{n} |P_{ij}|^2 < 1$.

Для определения скорости сходимости можно воспользоваться следующей теоремой: если какая-либо норма матрицы P, согласованная с данной нормой вектора, меньше единицы, то имеет место следующая оценка погрешности метода простой итерации:

$$\left\|\bar{x}^* - \bar{x}^k\right\| < \|P\|^k \cdot \left\|\bar{x}^k\right\| + \frac{\|P\|^k \cdot \|\bar{g}\|}{1 - \|P\|},$$

где $\bar{x}^*$ — точное решение системы (8.1).

Другими словами, условие сходимости выполняется, если выполняется условие доминирования диагональных элементов матрицы исходной системы A по строкам или столбцам:

$\sum_{i,j=1}^{n} |a_{ij}| < |a_{ii}|$ или $\sum_{i,j=1}^{n} |a_{ij}| \le |a_{jj}|$.

В этом случае легко можно перейти от системы вида (8.1) к системе (8.5). Для этого разделим i –ое уравнение системы на $a_{i,i}$ и выразим x_i :

$$x_i = \frac{f_i}{a_{ii}} - \frac{a_{11}}{a_{ii}}x_1 - \ldots - \frac{a_{1n}}{a_{ii}}x_n,$$

т.е. для матрицы P будет выполнено одно из условий сходимости, где

$$P = \begin{bmatrix} 0 & -\frac{a_{12}}{a_{11}} & \ldots & -\frac{a_{1n}}{a_{11}} \\ -\frac{a_{21}}{a_{22}} & 0 & \ldots & -\frac{a_{2n}}{a_{22}} \\ \ldots & \ldots & \ldots & \ldots \\ -\frac{a_{n1}}{a_{nn}} & -\frac{a_{n2}}{a_{nn}} & \ldots & 0 \end{bmatrix}.$$

Пример реализации метода простой итерации средствами **Maxima** с печатью промежуточных результатов представлен в скрипте ниже:

```
(%i1)   iterpr(a0,b0,x,n,eps):=block([a,b,x0,i,j,s,sum,p],
        a:copymatrix (a0), b:copymatrix(b0), x0:copymatrix(x),
        sum:1, p:0,
        while sum>eps do (
                sum:0, p:p+1, print("p= ",p," x= ",float(x)),
                for i:1 thru n do (
                        s:b[i,1],
                        for j:1 thru n do (s:s-a[i,j]*x0[j,1]),
                        s:s/a[i,i], x[i,1]:x0[i,1]+s, sum:sum+abs(s)
                ),
                x0:copymatrix(x)
        ),
        float(x))$
```

8.4.3 Метод Зейделя

В методе Зейделя система (8.1) также приводится к системе (8.5). Но при вычислении последующей компоненты вектора используются уже вычисленные компоненты этого вектора.

Итерационная формула метода в скалярной форме записывается следующим образом:

$$x_i^{(k+1)} = \sum_{j=1}^{i} p_{ij}x_i^{(k+1)} + \sum_{j=i+1}^{n} p_{ij}x_j^{(k)} + g_i.$$

Установим связь между методом Зейделя и методом простой итерации. Для этого матрицу B представим в виде суммы двух матриц: $P = H + F$, где

$$H = \begin{bmatrix} 0 & 0 & 0 & \ldots & 0 \\ p_{21} & 0 & 0 & \ldots & 0 \\ p_{31} & p_{32} & 0 & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ p_{n1} & p_{n2} & p_{n3} & \ldots & 0 \end{bmatrix}; \quad F = \begin{bmatrix} p_{11} & p_{12} & p_{13} & \ldots & p_{1n} \\ 0 & p_{22} & p_{23} & \ldots & p_{2n} \\ 0 & 0 & p_{33} & \ldots & p_{3n} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & \ldots & p_{nn} \end{bmatrix}.$$

Итерационная формула метода Зейделя в матричной форме записывается в виде:

$$\bar{x}^{(k+1)} = H\bar{x}^{(k+1)} + F\bar{x}^{(k)} + \bar{g}, \qquad \text{или}$$
$$(E-H)\,\bar{x}^{(k+1)} = F\bar{x}^{(k)} + \bar{g}, \qquad \text{откуда}$$
$$\bar{x}^{(k+1)} = (E-H)^{-1}F\bar{x}^{(k)} + (E-H)^{-1}\bar{g},$$

т.е. метод Зейделя эквивалентен методу простой итерации с матрицей $(E-H)^{-1}F$.

Исходя из полученной аналогии методов Зейделя и простой итерации, можно сформулировать следующий признак сходимости метода Зейделя: для того чтобы метод Зейделя сходился, необходимо и достаточно, чтобы все собственные значения матрицы $(E-H)^{-1}F$ по модулю были меньше единицы.

Другими словами, чтобы метод Зейделя сходился, необходимо и достаточно, чтобы все корни уравнения по модулю были меньше единицы, т.к.

$$\begin{aligned}
&\left|(E-H)^{-1}F - \lambda E\right| = \\
&\left|(E-H)^{-1}(E-H)\left[(E-H)^{-1}F - \lambda E\right]\right| = \\
&\left|(E-H)^{-1}\right|\,|F + \lambda H - \lambda E| = \\
&|F + \lambda H - \lambda E| = 0.
\end{aligned}$$

Сформулируем достаточный признак сходимости: для того, чтобы метод Зейделя сходился, достаточно, чтобы выполнилось одно из условий:

- $\|P\|_1 = \max\limits_i \sum\limits_{j=1}^{n} |p_{ij}| < 1$
- $\|P\|_2 = \max\limits_j \sum\limits_{i=1}^{n} |p_{ij}| < 1$
- $\|P\|_3 = \sqrt{\sum\limits_{i,j=1}^{n} |p_{ij}|^2} < 1$

При использовании метода Зейделя итерационный процесс сходится к единственному решению быстрее метода простых итераций.

Пример реализации метода Зейделя:

```
(%i1)   seidel(a0,b0,x,n,eps):=block([a,b,i,j,s,sum,p],
        a:copymatrix (a0), b:copymatrix(b0),
        sum:1, p:0,
        while sum>eps do (
                sum:0, p:p+1, print("p= ",p),
                for i:1 thru n do
                (
                        s:b[i,1],
                        for j:1 thru n do (s:s-a[i,j]*x[j,1]),
                        s:s/a[i,i], x[i,1]:x[i,1]+s, sum:sum+abs(s)
                )
        ),
        float(x))$
```

Пример решения простой системы методом Зейделя:

```
(%i2)   aa:matrix([3,1,1],[1,3,1],[1,1,3]); bb:matrix([6],[6],[8]);
        x:matrix([3],[3],[3]); zz:seidel(aa,bb,x,3,0.0000001);
```

$$(\%o2)\ \begin{pmatrix}3 & 1 & 1\\ 1 & 3 & 1\\ 1 & 1 & 3\end{pmatrix}\quad (\%o3)\ \begin{pmatrix}6\\ 6\\ 8\end{pmatrix}\quad (\%o4)\ \begin{pmatrix}3\\ 3\\ 3\end{pmatrix}$$

$$p=1\ p=2\ p=3\ p=4\ p=5\ p=6$$

$$p=7\ p=8\ p=9\ p=10\ p=11\ p=12$$

$$(\%o5)\ \begin{pmatrix}1.000000000753427\\ 0.99999999214211\\ 2.000000002368154\end{pmatrix}$$

8.5 Решение обыкновенных дифференциальных уравнений

8.5.1 Методы решения задачи Коши

Среди задач, с которыми приходится иметь дело в вычислительной практике, значительную часть составляют различные задачи, сводящиеся к решению обыкновенных дифференциальных уравнений. Обычно приходится прибегать к помощи приближенных методов решения подобных задач. В случае обыкновенных дифференциальных уравнений в зависимости от того, ставятся ли дополнительные условия в одной или нескольких точках отрезка изменения независимой переменной, задачи обычно подразделяются на одноточечные (задачи с начальными условиями или задачи Коши) и многоточечные. Среди многоточечных задач наиболее часто в прикладных вопросах встречаются так называемые граничные задачи, когда дополнительные условия ставятся на концах рассматриваемого отрезка.

В дальнейшем ограничимся рассмотрением численных методов решения задачи Коши. Для простоты изложения методов решения задачи будем рассматривать случай одного обыкновенного дифференциального уравнения первого порядка.

Пусть на отрезке $x_0 \le x \le L$ требуется найти решение $y(x)$ дифференциального уравнения

$$y' = f(x, y),\ (8.6)$$

удовлетворяющее при $x = x_0$ начальному условию $y(x_0) = y_0$.

Будем считать, что условия существования и единственности решения поставленной задачи Коши выполнены.

На практике найти общее либо частное решение задачи Коши удается для весьма ограниченного круга задач, поэтому приходится решать эту задачу приближенно.

Отрезок $[x_0, L]$ накрывается сеткой (разбивается на интервалы) чаще всего с постоянным шагом h ($h = x_{n+1} - x_n$), и по какому- то решающему правилу находится значение $y_{n+1} = y(x_{n+1})$. Таким образом, результатом решения задачи Коши численными методами является таблица, состоящую из двух векторов: $x = (x_0, x_1, \ldots, x_n)$ — вектора аргументов

и соответствующего ему вектора значений искомой функции $y = (y_0, y_1, \ldots, y_n)$.

Численные методы (правила), в которых для нахождения значения функции в новой точке используется информация только об одной (предыдущей) точке, называются одношаговыми.

Численные методы (правила), в которых для нахождения значения функции в новой точке используется информация о нескольких (предыдущих) точках, называются многошаговыми.

Из общего курса обыкновенных дифференциальных уравнений широкое распространение получил аналитический метод, основанный на идее разложения в ряд решения рассматриваемой задачи Коши. Особенно часто для этих целей используется ряд Тейлора. В этом случае вычислительные правила строятся особенно просто.

Приближенное решение $y_m(x)$ исходной задачи ищут в виде

$$y_m(x) = \sum_{i=0}^{m} \frac{(x - x_0)^i}{i!} y^{(i)}(x_0), \quad (8.7)$$
$$x_0 \leqslant x \leqslant b.$$

Здесь $y^{(0)}(x_0) = y(x_0), \quad y^{(1)}(x_0) = y'(x_0) = f(x_0, y_0)$, а значения $y^{(i)}(x_0), i = 2, 3, \ldots, m$ находят по формулам, полученным последовательным дифференцированием заданного уравнения:

$$y^{(2)}(x_0) = y''(x_0) = f_x(x_0, y_0) + f(x_0, y_0) f_y(x_0, y_0);$$
$$y^{(3)}(x_0) = y'''(x_0) = f_{x^2}(x_0, y_0) + 2f(x_0, y_0) f_y(x_0, y_0) +$$
$$+ f^2(x_0, y_0) f_{y^2}(x_0, y_0) + f_y(x_0, y_0)[f_x(x_0, y_0) + f(x_0, y_0) f_y(x_0, y_0)]; \quad (8.8)$$
$$\ldots$$
$$y^{(m)}(x_0) = F_m(f; f_x; f_{x^2}; f_{xy}; f_{y^2}; \ldots; f_{x^{m-1}}; f_{y^{m-1}})|_{x=x_0, y=y_0}.$$

Для значений x, близких к x_0, метод рядов (8.7) при достаточно большом значении m дает обычно хорошее приближение к точному решению $y(x)$ задачи (8.6). Однако с ростом расстояния $|x - x_0|$ погрешность приближения искомой функции рядом возрастает по абсолютной величине (при одном и том же количестве членов ряда), и правило (8.7) становится вовсе неприемлемым, когда x выходит из области сходимости соответствующего ряда (8.7) Тейлора.

Если в выражении (8.7) ограничиться $m = 1$, то для вычисления новых значений $y(x)$ нет необходимости пересчитывать значение производной, правда и точность решения будет невысока.

При использовании системы компьютерной алгебры более естественным выглядит метод последовательных приближений Пикара.

Рассмотрим интегрирование единичного дифференциального уравнения $\frac{dy}{dx} = f(x, y)$ на

отрезке $[x_0, x]$ с начальным условием $y(x_0) = y_0$. При формальном интегрировании получим:

$$\int_{x_0}^{x} \frac{dy}{dx} dx = \int_{x_0}^{x} f(x, y) dx$$

Процедура последовательных приближений метода Пикара реализуется согласно следующей схеме

$$y_{n+1}(x) - y(x_0) = \int_{x_0}^{x} f(x, y_n(x)) dx$$

В качестве примера рассмотрим решение уравнения $y' = -y$, при $y(0) = 1, x_0 = 0$:

```
(%i1)   rp:-y$ y0:1$ x0:0$ /*rp-правая часть уравнения */
(%i4)   y1:y0+integrate(subst(y0,y,rp),x,x0,x);
        /* y1 - первое приближение */
```

$$(\%o4)\ 1 - x$$

```
(%i5)   y2:y0+integrate(subst(y1,y,rp),x,x0,x);
        /* y2 - второе приближение */
```

$$(\%o5)\ \frac{x^2 - 2\,x}{2} + 1$$

```
(%i6)   y3:y0+integrate(subst(y2,y,rp),x,x0,x);
```

$$(\%o6)\ 1 - \frac{x^3 - 3\,x^2 + 6\,x}{6}$$

```
(%i7)   expand(%); /* Очевидное решение рассматриваемого ОДУ -
                экспонента y=exp(-x).
                В результате использование метода Пикара
                получаем решение в виде ряда Тейлора */
```

$$(\%o7)\ -\frac{x^3}{6} + \frac{x^2}{2} - x + 1$$

8.5.2 Метод рядов, не требующий вычисления производных правой части уравнения

Естественно поставить задачу о таком усовершенствовании приведенного выше одношагового метода, которое сохраняло бы основные его достоинства, но не было бы связано с нахождением значений производных правой части уравнения

$$y_m(x_{n+1}) \approx \sum_{i=0}^{m} \frac{h^i}{i!} y^{(i)}(x_n), \quad (8.9)$$

где $x_{n+1} = x_n + h$.

Чтобы выполнить это условие (последнее), производные у $y^{(i)}(x)$, $i = 2, 3, \ldots, m$,

входящие в правую часть уравнения (8.9), можно заменить по формулам численного дифференцирования их приближенными выражениями через значение функции y' и учесть, что $y'(x) = f[x, y(x)]$.

8.5.2.1 Метод Эйлера

В случае $m = 1$ приближенное равенство (8.9) не требует вычисления производных правой части уравнения и позволяет с погрешностью порядка h^2 находить значение $y(x_n + h)$ решения этого уравнения по известному его значению $y(x_n)$. Соответствующее одношаговое правило можно записать в виде

$$y_{n+1} = y_n + h f_n. \quad (8.10)$$

Это правило (8.10) впервые было построено Эйлером и носит его имя. Иногда его называют также правилом ломаных или методом касательных. Метод Эйлера имеет относительно низкий порядок точности — h^2 на одном шаге. Практическая оценка погрешности приближенного решения может быть получена по правилу Рунге.

Пример реализации метода Эйлера средствами **Maxima** приведён в следующем примере:

```
(%i1)   euler1(rp,fun,y0,x0,xend,h):=block([OK,_x,_y,_y1,rez],
                _x:x0, _y:y0, rez:[_y], OK:-1, eps:0.1e-7,
                while OK<0 do (
                        if ((_x+h>xend) or (abs(_x+h-xend)<eps))
                                then (h:xend-_x,_x:xend, OK:1)
                                else (_x:_x+h),
                        _y1:makelist(float(_y[i]+h*subst([fun[i]=_y[i],x=_x],
                                rp[i])),i,1,length(_y)),rez:append(rez,[_y1]),
                        _y:_y1
                ),
                rez
        )$
```

Правые части решаемых дифференциальных уравнений передаются в функцию $euler1$ в списке rp. По умолчанию предполагается, что список имён зависимых переменных — fun, имя независимой переменной — x. Начальные значения независимой и зависимых переменных — список y_0 и скалярная величина x_0, граница интервала интегрирования — величина $xend$, шаг интегрирования — h.

Следующий пример — обращение к функции $euler1$. Приведено решение системы из трёх дифференциальных уравнений на интервале [0, 1] с шагом $h = 0.1$:

$$\begin{cases} \dfrac{dy}{dx} = -2y, \\ \dfrac{dv}{dx} = -5v, \\ \dfrac{dz}{dx} = 3x. \end{cases}$$

С начальными условиями $y(0) = 1.0; v(0) = 1.0; z(0) = 0$, решением уравнений данной

системы будут функции:

$$\begin{cases} y(x) = e^{-2x}, \\ v(x) = e^{-5x}, \\ z(x) = \dfrac{3}{2} \cdot x^2. \end{cases}$$

```
(%i2)   euler1([-2*y,-5*v,3*x],[y,v,z],[1,1,0],0,1,0.1);
```

$$(\%o2)\ [[1,1,0],[0.8,0.5,0.03],[0.64,0.25,0.09],[0.512,0.125,0.18],$$
$$[0.4096,0.0625,0.3],[0.32768,0.03125,0.45],[0.262144,0.015625,0.63],$$
$$[0.2097152,0.0078125,0.84],[0.16777216,0.00390625,1.08],[0.134217728,$$
$$0.001953125,1.35],[0.1073741824,9.7656249999999913 * 10^-4,1.65]]$$

Проверить решение можно сравнивая графики точного решения и множества вычисленных приближенных значений. Пример последовательности команд, позволяющих выделить отдельные компоненты решения системы ОДУ и построить график точного и приближенного решения третьего уравнения системы, представлен ниже (точные решения — списки yf, vf, zf; приближенные решения — списки yr, vr, zr; список значений независимой переменной — xg).

```
(%i3)   rez:euler1([-2*y,-5*v,3*x],[y,v,z],[1,1,0],0,1.0,0.1)$
        n:length(rez)$
        yr:makelist(rez[k][1], k, 1, n)$
        vr:makelist(rez[k][2], k, 1, n)$
        zr:makelist(rez[k][3], k, 1, n)$
        xg:makelist(0.1*(k-1),k,1,n)$
        yf:makelist(exp(-2*xg[k]), k, 1, n)$
        vf:makelist(exp(-5*xg[k]), k, 1, n)$
        zf:makelist(3*xg[k]^2/2, k, 1, n)$
        plot2d ([[discrete, xg,zr],[discrete, xg,zf]],
                        [style, points, lines])$
```

Уменьшение шага h приводит к уменьшению погрешности решения (в данном примере — шаг 0.1).

8.5.2.2 Метод Рунге-Кутта

Изложим идею метода на примере задачи Коши:

$$y' = f(x, y);$$
$$x_0 \le x \le b;$$
$$y(x_0) = y_0.$$

Интегрируя это уравнение в пределах от x до $x + h$ ($0 < h < 1$), получим равенство

$$y(x+h) = y(x) + \int_x^{x+h} f\,[t, y(t)]dt, \quad (8.10)$$

которое посредством последнего интеграла связывает значения решения рассматриваемого уравнения в двух точках, удаленных друг от друга на расстояние шага h.

Для удобства записи выражения (8.11) используем обозначение $\Delta y = y(x+h) - y(x)$ и замену переменной интегрирования $t = x + h$. Окончательно получим:

$$\Delta y = h \int_0^1 f[x + \alpha\, h, y(x + \alpha\, h)] d\alpha \quad (8.12)$$

В зависимости от способа вычисления интеграла в выражении (8.12) получают различные методы численного интегрирования обыкновенных дифференциальных уравнений.

Рассмотрим линейную комбинацию величин $\phi_i,\ i = 0, 1, \ldots, q$, которая будет являться аналогом квадратурной суммы и позволит вычислить приближенное значение приращения Δy:

$$\Delta y \approx \sum_{i=0}^{q} a_i \phi_i,$$

где

$$\phi_0 = hf(x, y);$$
$$\phi_1 = hf(x + \alpha_1 h; y + \beta_{10}\phi_0);$$
$$\phi_2 = hf(x + \alpha_2 h; y + \beta_{20}\phi_0 + \beta_{21}\phi_1);$$
$$\ldots$$

Метод четвертого порядка для $q = 3$, являющийся аналогом широко известной в литературе четырехточечной квадратурной формулы "трех восьмых", имеет вид

$$\Delta y \approx \frac{1}{8}(\phi_0 + 3\phi_1 + 3\phi_2 + \phi_3),$$

где

$$\phi_0 = hf(x_n, y_n);$$
$$\phi_1 = hf(x_n + \frac{h}{3}, y_n + \frac{\phi_0}{3});$$
$$\phi_2 = hf(x_n + \frac{2}{3}h, y_n - \frac{\phi_0}{3} - \phi_1);$$
$$\phi_3 = hf(x_n + h, y_n + \phi_0 - \phi_1 + \phi_2).$$

Особо широко известно другое вычислительное правило типа Рунге-Кутта четвертого порядка точности:

$$\Delta y = \frac{1}{6}(\phi_0 + 2\phi_1 + 2\phi_2 + \phi_3),$$

где

$$\phi_0 = hf(x_n, y_n),$$
$$\phi_1 = hf(x_n + \frac{h}{2}, y_n + \frac{\phi_0}{2}),$$
$$\phi_2 = hf(x_n + \frac{h}{2}, y_n + \frac{\phi_1}{2}),$$
$$\phi_3 = hf(x_n + h, y_n + \phi_2).$$

Метод Рунге-Кутта имеет погрешность четвертого порядка ($\sim h^4$).

Функция **Maxima**, реализующая метод Рунге-Кутта 4-го порядка, приведена в следующем примере (с печатью промежуточных результатов):

```
(%i1)   rk4(rp,fun,y0,x0,xend,h):=block(
                [OK,n,h1,_x,_y,_k1,_k2,_k3,_k4,rez],
                _x:x0, _y:y0, rez:[_y], OK:-1, h1:h, n:length(_y),
                while OK<0 do (
                        if (_x+h1>=xend) then (h1:xend-_x, OK:1),
                        _k1:makelist(float(h1*subst([fun[i]=
                                float(_y[i]),x=float(_x)],rp[i])),i,1,n),
                        _k2:makelist(float(h1*subst([fun[i]=
                                float(_y[i]+_k1[i]/2),x=float(_x+h1/2)],
                                        rp[i])),i,1,n),
                        _k3:makelist(float(h1*subst([fun[i]=
                                float(_y[i]+_k2[i]/2),x=float(_x+h1/2)],
                                        rp[i])),i,1,n),
                        _k4:makelist(float(h1*subst([fun[i]=
                                float(_y[i]+_k3[i]),x=float(_x+h1)],rp[i])),
                                        i,1,n),
                        _y1:makelist(float(_y[i]+
                                (_k1[i]+2*_k2[i]+2*_k3[i]+_k4[i])/6),i,1,n),
                        rez:append(rez,[_y1]),
                        print("x= ",_x, " y= ",_y),
                        _x:_x+h1,
                        _y:_y1
                ), rez
        )$
```

Пример обращения к функции $rk4$ представлен следующей последовательностью команд (решалась та же система, что и при тестировании метода Эйлера):

```
(%i2)   rk4([-2*y,-5*v,3*x],[y,v,z],[1,1,1],0,1,0.1);
x= 0    y= [1,1,1]
x= 0.1  y= [0.81873333333333,0.60677083333333,1.015]
x= 0.2  y= [0.67032427111111,0.36817084418403,1.06]
x= 0.3  y= [0.54881682490104,0.22339532993458,1.135]
x= 0.4  y= [0.44933462844064,0.13554977050718,1.24]
x= 0.5  y= [0.3678852381253,0.082247647208783,1.375]
x= 0.6  y= [0.30119990729446,0.04990547343658,1.54]
x= 0.7  y= [0.24660240409888,0.030281185705008,1.735]
x= 0.8  y= [0.20190160831589,0.018373740284549,1.96]
x= 0.9  y= [0.16530357678183,0.011148649703906,2.215]
x= 1.0  y= [0.13533954843051,0.0067646754713805,2.5]
```