Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский Авиационный Институт»
(Национальный Исследовательский Университет)

Факультет №8 «Информационные технологии и прикладная математика»
Кафедра 805 «Математическая кибернетика»

Реферат

по теме

«Решение рекуррентных соотношений»

1 семестр

Автор работы:
студент 1 курса, гр. М8О-103Б-21
Тысячный В. В.
Руководитель проекта:
Севастьянов В. С.
Дата сдачи:

Москва 2021

Оглавление

Введение

Широкое внедрение компьютеров и математических пакетов в учебную и профессиональную деятельность математиков поставило перед научным сообществом задачу пересмотра не только методики преподавания математики, но и корректировки базового перечня математических дисциплин. Если в докомпьютерную эпоху основой математической подготовки как математиков, так и инженеров являлся математический анализ и дифференциальные уравнения, то на современном этапе все большую роль играют такие разделы, как дискретная математика и численные методы.

Этот крен в сторону большей значимости дискретных моделей приводит к тому, что многие курсы – в особенности прикладного характера – на самой ранней стадии изложения вводят не только дифференциальную модель, но и как минимум параллельно рассматривают соответствующую разностную.

Классическое дифференциальное исчисление как великое достижение математической мысли определило многие пути развития математики. Наука постоянно совершенствует свои достижения, и дифференциальное исчисление не должно быть исключением. Рекуррентное исчисление является новым этапом в развитии дифференциального исчисления. С его появлением стало видно, что дифференциальное исчисление было неполным, и включало только одно направление. В действительности, изначально существовали два направления развития исчисления – классическое интеллектуальное и рекуррентное алгоритмическое. Классическое исчисление, которое по своей природе было творческим, ориентированным на интеллект, единственным и бесспорным, пока не наступила компьютерная эра. Сохраняя свои достоинства при решении интеллектуальных классическое исчисление оказалось совершенно беспомощным при решении вычислительных задач. Частично с этим связано доминирование в настоящее время дискретных численных методов, которые сейчас кажутся бесспорными. Рекуррентное исчисление является настолько мощным вычислительным инструментом, что может изменить приоритеты.

Основой рекуррентного исчисления являются рекуррентные последовательности. Рекуррентные последовательности имеют важное значение в математике и ее приложениях. Часто при решении различных задач, как прикладных, так и теоретических, появляются последовательности. Это могут быть последовательности чисел или элементов какого-то другого множества.

Нередко при решении содержательных задач возникает такая ситуация: мы знаем, как определяются элементы последовательности a_n при каждом значении n. Такое описание последовательностей называется явным. Однако явное представление не всегда удобно, поэтому зачастую последовательность задается рекуррентным соотношением, а уже по нему иногда требуют найти явное описание. Это достаточно трудоемкий процесс, поэтому в данном реферате будет представлен алгоритмический вариант решения линейных рекуррентных уравнений на языке программирования Python в среде Jupyter Notebook с использованием библиотеки SymPy.

Теоретическое обоснование

Базовые определения

Последовательностью элементов заданного множества A называют закон, по которому каждому натуральному числу n сопоставляется элемент a_n множества A. Например, на множестве натуральных чисел последовательность квадратов натуральных чисел задается простым правилом, каждому n сопоставляется n^2.

Другой способ задания последовательности – с помощью указания связи между некоторыми членами последовательности. Так можно задать, например, арифметическую прогрессию: разность для арифметической прогрессии между любыми двумя соседними членами последовательности a_{n+1} и a_n есть величина постоянная, равная d – разности арифметической прогрессии. В таком способе задания могут участвовать и более двух членов последовательности. Таким образом, один из членов последовательности можно считать определенным с помощью других членов этой последовательности – это, так называемый, рекуррентный способ задания. В случае арифметической прогрессии имеем следующее соотношение для ее членов: $a_{n+1} - a_n = d$. Классическим является пример чисел Фибоначчи, данная последовательность задается уже не через один член, а два: $a_{n+2} - a_{n+1} - a_n = 0$.

Последовательность $a_0, a_1 \ldots\ a_n \ldots$ называется **рекуррентной** (возвратной) **последовательностью порядка k**, если ее члены при каждом $n \in \mathbb{N}$ удовлетворяют равенству (рекуррентному соотношению):

$$a_{n+k} + p_{k-1}a_{n+k-1} + p_{k-2}a_{n+k-2} + \cdots + p_0 a_n = f(n),$$

где k – фиксированное натуральное число, обозначающее порядок рекуррентного уравнения, p_i – коэффициенты рекуррентного соотношения, $f(n)$ – некоторая функция, натурального аргумента. При $f(n) = 0$ последовательность называется однородной, в противном случае – неоднородной. Последовательность порядка k однозначно задается первыми своими k элементами, эти элементы называются **начальными значениями** последовательности.

Линейным однородным рекуррентным уравнением (ЛОРУ) порядка k называется соотношение вида

$$x_{n+k} + p_{k-1}x_{n+k-1} + \cdots + p_0 x_n = 0,$$

где $p_0, p_1, \ldots, p_{k-1}$ – некоторые коэффициенты.

Если последовательность $\{x_n\}$ задается некоторым ЛОРУ, то данная последовательность называется **возвратной**. Например, последовательность чисел Фибоначчи является возвратной и задается ЛОРУ 2 порядка.

Частным решением рекуррентного соотношения называется такая последовательность $\{a_n\}$, что при подстановке в уравнение соответствующих элементов последовательности при каждом n получается верное равенство.

Лемма 1:

> Если последовательности $\{a_n\}$ и $\{b_n\}$ – частные решения ЛОРУ, а C_1 и C_2 – произвольные коэффициенты, то последовательность $\{C_1 a_n + C_2 b_n\}$ так же является частным решением данного ЛОРУ.

Доказательство:

Подставив данную последовательность в ЛОРУ и вынеся C_1 и C_2 за скобку как общие коэффициенты, в скобках получатся ЛОРУ с решениями $\{a_n\}$ и $\{b_n\}$, которые по определению равны 0, следовательно последовательность $\{C_1 a_n + C_2 b_n\}$ является частным решением.

Множество всех частных решений называется **общим решением** рекуррентного соотношения. Из леммы 1 следует, что общее решение ЛОРУ является линейной комбинацией частных решений.

Линейным неоднородным рекуррентным уравнением (ЛНРУ) порядка k называется соотношение вида:

$$x_{n+k} + p_{k-1}x_{n+k-1} + \cdots + p_0 x_n = g(n)$$

где $p_0, p_1, \ldots, p_{k-1}$ – некоторые коэффициенты, а $g(n)$ – некоторая числовая функция, не равная тождественно нулю.

Теорема 1:

Общим решением ЛНРУ является сумма частного решения $\{a_n\}$ ЛНРУ и общего решения, соответствующего ему ЛОРУ.

Доказательство:

Тривиально, что указанная сумма является решением исходного ЛНРУ. Пусть $\{b_n\}$ является еще одним частным решением ЛНРУ, тогда $\{a_n - b_n\}$ является частным решением соответствующего ЛОРУ, следовательно входит в его общее решение.

Решение рекуррентных соотношений

На данный момент существует два метода решения: метод производящих функций и метод характеристических уравнений. Первый метод является сугубо математическим, требует использования абстрактных понятий, таких как сама производящая функция $G(z)$, алгебраических преобразований, которые требуют некоторый творческий подход, и разложение функции $G(z)$ в ряд. Эти все действия требуют интеллектуального подхода со стороны человека, так что запрограммировать данный метод весьма нетривиальная задача. Второй метод более затратный по времени для человека, но имеет более четкую структуру, он и будет рассматриваться далее.

Решение ЛОРУ

Общий алгоритм:

1. Составить характеристический многочлен ЛОРУ
2. Решить полученный многочлен
3. Составить общее решение ЛОРУ
4. Составить систему из k уравнений относительно неизвестных коэффициентов C_i, подставляя начальные значения для последовательности
5. Найти явное выражение для рекуррентной последовательности

Характеристическим многочленом ЛОРУ

$$x_{n+k} + \sum_{i=1}^{k} p_{k-j}x_{n+k-j} = 0,$$

называется многочлен комплексной переменной (степень характеристического многочлена совпадает с порядком линейного рекуррентного уравнения)

$$P(x) = x^k + p_{k-1}x^{k-1} + \cdots + p_1 x + p_0,$$

Лемма 2:

Пусть λ – корень характеристического многочлена $P(x)$ ЛОРУ. Тогда последовательность $\{\lambda^n\}$ является частным решением этого ЛОРУ

Доказательство:

Если предположение верно, то при подстановке данной последовательности в ЛОРУ должен получиться тождественный 0. Выполним подстановку и преобразования:

$$\lambda^{n+k} + \sum_{i=1}^{k} p_{k-i}\lambda^{n+k-i} = \lambda^n \left(\lambda^k + \sum_{i=1}^{k} p_{k-i}\lambda^{k-i} \right) = \lambda^n P(\lambda) = 0$$

так как λ является корнем $P(\lambda)$, то по определению $P(\lambda) = 0$.

Теорема 2:

Если $\lambda_1, \dots, \lambda_k$ – **различные** корни характеристического многочлена $P(x)$ ЛОРУ, то общее решение этого уравнения имеет вид $C_1\lambda_1^n + \cdots + C_k\lambda_k^n$, где $C_1, \dots, C_k$ – произвольные (комплексные) числа.

Доказательство:

Из леммы 2 следует, что каждая из последовательностей $\{\lambda_i^n\}$ является частным решением ЛОРУ, а из леммы 1 следует, что линейная комбинация частных решений так же является частным решением. Из этого следует, что требуется доказать лишь отсутствие других решений ⇒ любое частное решение данного ЛОРУ представимо в таком виде.

Предположим противное, тогда $\exists\{a_n\}$, являющееся частным решением ЛОРУ, но непредставимая в виде $C_1\lambda_1^n + \cdots + C_k\lambda_k^n$. Тогда для данной системы невозможно подобрать коэффициенты $C_1, \dots, C_k$:

$$\begin{cases} c_1 + \cdots + c_k = a_0 \\ \lambda_1 c_1 + \cdots + \lambda_k c_k = a_1 \\ \dots \\ \lambda_1^{k-1} + \cdots + \lambda_k^{k-1} c_k^{k-1} = a_{k-1} \end{cases}$$

Рассмотрим матрицу для данной системы:

$$A = \begin{pmatrix} 1, & \dots, & 1 \\ \lambda_1, & \dots, & \lambda_k \\ \lambda_1^2, & \dots, & \lambda_k^2 \\ & \dots & \\ \lambda_1^{k-1}, & \dots, & \lambda_k^{k-1} \end{pmatrix}$$

По методу Крамера у данной системы существует решение, и оно единственно, если $\det(A) \neq 0$. $\det(A)$ является определителем Вандермонда, следовательно

$$\det(A) = \prod_{1 \le i < j \le k} (\lambda_i - \lambda_j)$$

Так как по условию корни попарно различны, то $\det(A) \neq 0$, следовательно $\{a_n\}$ выражается линейной комбинацией $\{\lambda_i^n\}$, противоречие.

Для явного выражения рекуррентной последовательности требуется решить описанную выше систему линейных уравнений, подставив вместо a_i i-ое начальное значение.

Но описанный выше алгоритм работает только при попарно различных корнях, в противном случае существуют другие частные решения.

Теорема 3:

Пусть λ – корень кратности r, $1 \leq r \leq k$ характеристического полинома ЛОРУ. Тогда последовательность $\{n^m\lambda^n\}$, где $0 \leq m < r$, является частным решением этого ЛОРУ.

Доказательство:

Подставим элементы проверяемого частного решения в ЛОРУ:

$$(n+k)^m\lambda_{n+k} + \sum_{i=1}^{k} p_{k-i}\,(n+k-i)^m\lambda^{n+k-i} = \lambda^n\left(\sum_{i=0}^{k} p_{k-i}(n+k-i)^m\lambda^{k-i}\right),$$

где p_0 будем считать равным 0. Тогда, выполнив замену выражения $(n+k-j)^m$ по биному Ньютона, получим:

$$\lambda^n\left(\sum_{i=0}^{k} p_{k-i}\sum_{j=0}^{m} C_m^j n^{m-j}(k-i)^j\,\lambda^{k-i}\right) = \lambda^n\left(\sum_{j=0}^{m} C_m^j n^{m-j}\sum_{i=0}^{k} p_{k-i}(k-i)^j\lambda^{k-i}\right) = \lambda^n n^m\left(\sum_{j=0}^{m} C_m^j n^{-j}P_j(x)\right).$$

Заметим, что $P_0(x) = P(x)$ – характеристический многочлен ЛОРУ. По условию λ – корень кратности r, следовательно по теореме Безу $P_0(x) = (x-\lambda)^r Q_0(x)$.

Возьмем первую производную по x от $P_j(x)$:

$$P_j'(x) = \sum_{i=0}^{k} p_{k-i}(k-i)^{j+1}x^{k-i-1}$$

Заметим, что тогда $P_{j+1}(x) = xP_j'(x)$.

Докажем по индукции, что при любом натуральном $m < r$: $P_m(x) \vdots (x-\lambda)^{r-m}$:

База: $P_0(x) = (x-\lambda)^r Q_0(x)$ – выполнено.

Переход: пусть при $m = k$ данное высказывание истинно, тогда $P_k(x) = (x-\lambda)^{r-k}A(x)$, где $A(x)$ – некоторый многочлен. Докажем для $m = k+1$:

Т.к. $P_{j+1}(x) = xP_j'(x)$, то $P_{k+1}(x) = xP_k'(x) = x\Big((r-k)(x-\lambda)^{r-k-1}A + (x-\lambda)^{r-k}A'\Big) = (x-\lambda)^{r-(k+1)}\Big(x\big((r-k)A + (x-\lambda)A'\big)\Big) = (x-\lambda)^{r-(k+1)}B$, где B – некоторый многочлен. Из этого следует, что $P_{k+1} \vdots (x-\lambda)^{r-(k+1)}$.

Из этого всего следует, что $P_j(\lambda) = 0\ (0 \leq j < r) \Rightarrow$ последовательность $\{n^m\lambda^n\}$ является частным решением исходного ЛОРУ.

Остается доказать, что каждое частное решение ЛОРУ имеет такой же вид. Доказательство аналогично доказательству теоремы 2.

Решение ЛНРУ

Есть два метода решения ЛНРУ: решение путем сведения к однородным и решение по виду правой части. Суть первого метода заключается в вычитании из ЛНРУ с заменой n на $n+1$ исходного уравнения до тех пор, пока правая часть не станет равна нулю. Данный метод может быть очень затратен по времени, так что будет использоваться второй.

Общий алгоритм:

1. Составить характеристический многочлен соответствующего ЛОРУ
2. Решить полученный многочлен для ЛОРУ
3. Выписать общее решение ЛОРУ
4. В зависимости от λ выбрать вид частного решения ЛНРУ
5. Подставить частное решение в исходное рекуррентное соотношение
6. Найти параметры для частного решения ЛНРУ
7. Сложить общее решение ЛОРУ и частное решение ЛНРУ

Во втором случае нельзя предложить универсальный метод для любой правой части. Но если она имеет вид λ^n, умноженное на полином переменной n, частное решение имеет определенный вид.

Лемма 3:

Если λ – корень кратности r характеристического многочлена, соответствующего ЛОРУ, то $\sum_{i=0}^{k} p_i \lambda^i i^n = 0$ тогда и только тогда, когда $n < r, n \in \mathbb{N}$.

Доказательство:

Если λ – корень кратности r функции $f(x)$, то для $f'(x)$ λ – корень кратности $r-1$.

Если взять первую производную от $f(x) = \sum_{i=0}^{k} p_i x^i$ и домножить ее на λ, то новая функция $\lambda f'(x)$, исходя из вышеописанного свойства будет иметь корень λ кратности $r-1$, следовательно функция $f_1(x) = \lambda f'(x)$ тоже. При этом $f_1(\lambda) = \sum_{i=0}^{k} p_i \lambda^i i = 0$.

Возьмем от функции еще одну производную и домножим на λ: $f_2(x) = \lambda f_1'(x)$. Данная функция имеет корень λ кратности $r-2$, при этом $f_2(\lambda) = \sum_{i=0}^{k} p_i \lambda^i i^2 = 0$.

Таким образом, если брать производную от прошлой функции и домножать ее на λ, то новая функция будет иметь корень λ. Так как с каждым взятием производной кратность уменьшается на единицу, то так будет пока индекс функции не станет равным $r-1$, тогда кратность λ будет ровно 1, а функция $f_{r-1}(\lambda) = \sum_{i=0}^{k} p_i \lambda^i i^{r-1} = 0$. Так как при следующих итерациях кратность λ станет равна 0, то функции $\sum_{i=0}^{k} p_i \lambda^i i^n$ при $\forall n > r-1$ не будут равны 0.

Теорема 4:

Для ЛНРУ $x_{n+k} + \sum_{j=1}^{k} p_{k-j} x_{n+k-j} = \left(\sum_{l=0}^{m} d_l n^l\right) \cdot \lambda^n$, в котором $p_{k-1}, \dots, p_0, d_0, d_1, \dots, d_m, \lambda$ – некоторые числа, $\lambda \neq 0, k \geq 1$, существует частное решение вида:

- $\{(\sum_{l=0}^{m} c_l\, n^l) \cdot \lambda^n\}$, где $c_0, c_1, \dots, c_m$ – некоторые параметры, если λ – не является корнем характеристического многочлена, соответствующего ЛОРУ;
- $\{n^r \cdot (\sum_{l=0}^{m} c_l\, n^l) \cdot \lambda^n\}$, где $c_0, c_1, \dots, c_m$ – некоторые параметры, если λ – корень кратности r характеристического многочлена, соответствующего ЛОРУ.

Доказательство:

- Докажем, что решение вида $(\sum_{l=0}^{m} c_l\, n^l) \cdot \lambda^n$ при λ, не являющимся корнем характеристического многочлена ЛОРУ, является частным решение ЛНРУ:

  Если это так, то должны существовать такие c_l, что $\forall n \in \mathbb{N}$:
  $\sum_{j=0}^{k} p_{k-j}\lambda^{n+k-j}\left(\sum_{l=0}^{m} c_l(n+k-j)^l\right) = \left(\sum_{l=0}^{m} d_l n^l\right)\lambda^n$ – тождественно истинно.

  Поделив на λ^n и разложив по биному Ньютона получим:
  $\sum_{j=0}^{k-1} p_{k-j}\lambda^{k-j}\left(\sum_{l=0}^{m} c_l\left(\sum_{i=0}^{l} C_l^i n^{l-i}(k-j)^i\right)\right) + p_0\left(\sum_{l=0}^{m} c_l n^l\right) = \sum_{l=0}^{m} d_l n^l$, где $p_k = 1$.

  Заметим, что левая и правая части – полиномы от n степени m. Приведем выражение в вид полинома:

  $$\sum_{l=0}^{m} n^l \left(\sum_{j=0}^{k-1} p_{k-j}\lambda^{k-j}\left(\sum_{i=0}^{m-l} c_{m-i} C_{m-i}^{m-l-i}(k-j)^{m-l-i}\right) + p_0 c_l\right) = \sum_{l=0}^{m} n^l d_l$$

  Таким образом, если коэффициенты перед n^l будут равны, утверждение будет доказано. Следовательно, необходимо доказать существование параметров c_{m-i}, чтобы выполнялось данное равенство коэффициентов.

  Рассмотрим равенство коэффициентов перед старшей степенью: $c_m\left(\sum_{j=0}^{k} p_j\lambda^j\right) = d_m$. В данном уравнении содержится только один параметр c_m, все остальное – известные числа, следовательно, можно найти $c_m = \frac{d_m}{\sum_{j=0}^{k} p_j\lambda^j}$.

  Уравнение для коэффициента перед степенью $m-1$:

  $$c_m\left(\sum_{j=0}^{k} p_j\lambda^j\, j\right) C_m^1 + c_{m-1}\left(\sum_{j=0}^{k} p_j\lambda^j\right) = d_{m-1}$$

  В нем уже два параметра: c_m и c_{m-1}, но c_m можно рассчитать по ранее выведенному уравнению. Тогда, зная c_m, можно выразить $c_{m-1} = \frac{d_{m-1} - c_m\left(\sum_{j=0}^{k} p_j\lambda^j j\right)C_m^1}{\sum_{j=0}^{k} p_j\lambda^j}$.

Таким образом, каждый коэффициент можно рассчитать, зная параметры для предыдущих уравнений: $c_l = \frac{d_l - \left(\sum_{i=0}^{m-l-1} c_{m-i}\left(\sum_{j=0}^{k} p_j \lambda^j j^{m-l-i}\right) C_{m-i}^{l}\right)}{\sum_{j=0}^{k} p_j \lambda^j}, l \in [m-1; 0]$.

Знаменатель не равен 0, иначе λ является корнем полинома $\sum_{j=0}^{k} p_j x^j$, а он является ничем иным, как характеристическим многочленом ЛОРУ, а это противоречит условию. Из этого следует, что всегда можно найти набор параметров c_l, чтобы равенства коэффициентов выполнялись.

- Если λ является корнем кратности r характеристического многочлена, то должно выполняться:

$$\sum_{j=0}^{k} p_{k-j} \lambda^{n+k-j} (n+k-j)^r \left(\sum_{l=0}^{m} c_l (n+k-j)^l\right) = \left(\sum_{l=0}^{m} d_l n^l\right) \lambda^n$$

Поделив на λ^n и разложив по биному Ньютона получим:

$$\sum_{j=0}^{k-1} p_{k-j} \lambda^{k-j} \left(\sum_{i=0}^{r} C_r^i n^i (k-j)^{r-i}\right) \left(\sum_{l=0}^{m} c_l \left(\sum_{i=0}^{l} C_l^i n^i (k-j)^{l-i}\right)\right) + p_0 \left(\sum_{l=0}^{m} c_l n^l\right) = \sum_{l=0}^{m} d_l n^l$$

Приведем в вид полинома от n и выполним некоторые преобразования. Если $r > m$ (в противном случае уравнение имеет схожий вид, это не отражается на доказательстве):

$$
\begin{aligned}
&n^{m+r}c_m\left(\sum_{j=0}^{k}p_{k-j}\lambda^{k-j}\right)+n^{m+r-1}\left(c_m(C_r^0C_m^1+C_r^1C_m^0)\left(\sum_{j=0}^{k}p_{k-j}\lambda^{k-j}(k-j)^1\right)+c_{m-1}\left(\sum_{j=0}^{k}p_{k-j}\lambda^{k-j}\right)\right)\\
&+n^{m+r-2}\left(c_m(C_r^0C_m^2+C_r^1C_m^1+C_r^2C_m^0)\left(\sum_{j=0}^{k}p_{k-j}\lambda^{k-j}(k-j)^2\right)\right.\\
&\left.+c_{m-1}(C_r^0C_{m-1}^1+C_r^1C_{m-1}^0)\left(\sum_{j=0}^{k}p_{k-j}\lambda^{k-j}(k-j)^1\right)+c_{m-2}\left(\sum_{j=0}^{k}p_{k-j}\lambda^{k-j}\right)\right)+\cdots\\
&+n^{r}\left(c_m(C_r^0C_m^m+\cdots+C_r^mC_m^0)\left(\sum_{j=0}^{k}p_{k-j}\lambda^{k-j}(k-j)^m\right)\right.\\
&\left.+c_{m-1}(C_r^0C_{m-1}^{m-1}+\cdots+C_r^{m-1}C_{m-1}^{m-1})\left(\sum_{j=0}^{k}p_{k-j}\lambda^{k-j}(k-j)^{m-1}\right)+\cdots+c_0\left(\sum_{j=0}^{k}p_{k-j}\lambda^{k-j}\right)\right)\\
&+n^{r-1}\left(c_m(C_r^1C_m^m+\cdots+C_r^{m+1}C_m^0)\left(\sum_{j=0}^{k}p_{k-j}\lambda^{k-j}(k-j)^{m+1}\right)+\cdots+c_0C_r^1\left(\sum_{j=0}^{k}p_{k-j}\lambda^{k-j}\,(k-j)^1\right)\right)+\cdots\\
&+n^{m}\left(c_m(C_r^{m-r}C_m^m+\cdots+C_r^rC_m^0)\left(\sum_{j=0}^{k}p_{k-j}\lambda^{k-j}(k-j)^r\right)+\cdots+c_0C_r^m\left(\sum_{j=0}^{k}p_{k-j}\lambda^{k-j}\,(k-j)^{r-m}\right)\right)\\
&+n^{m-1}\left(c_m(C_r^0C_m^{m-1}+\cdots+C_r^{m-1}C_m^0)\left(\sum_{j=0}^{k}p_{k-j}\lambda^{k-j}(k-j)^{r+1}\right)\right.\\
&\left.+c_{m-1}(C_r^0C_{m-1}^{m-1}+\cdots+C_r^{m-1}C_{m-1}^0)\left(\sum_{j=0}^{k}p_{k-j}\lambda^{k-j}(k-j)^r\right)\right.\\
&\left.+c_{m-2}(C_r^0C_{m-2}^{m-2}+\cdots+C_r^{m-2}C_{m-2}^0)\left(\sum_{j=0}^{k}p_{k-j}\lambda^{k-j}(k-j)^{r-1}\right)+\cdots+c_0C_r^{m-1}\left(\sum_{j=0}^{k}p_{k-j}\lambda^{k-j}\,(k-j)^{r-m+1}\right)\right)\\
&+n^{m-2}\left(c_m(C_r^0C_m^{m-2}+\cdots+C_r^{m-2}C_m^0)\left(\sum_{j=0}^{k}p_{k-j}\lambda^{k-j}(k-j)^{r+2}\right)\right.\\
&\left.+c_{m-1}(C_r^0C_{m-1}^{m-2}+\cdots+C_r^{m-2}C_{m-1}^0)\left(\sum_{j=0}^{k}p_{k-j}\lambda^{k-j}(k-j)^{r+1}\right)+\cdots+c_0C_r^{m-2}\left(\sum_{j=0}^{k}p_{k-j}\lambda^{k-j}(k-j)^{r-m+2}\right)\right)\\
&+\cdots+n\left(c_m(C_r^0C_m^1+C_m^1)\left(\sum_{j=0}^{k}p_{k-j}\lambda^{k-j}(k-j)^{m+r-1}\right)+\cdots+c_0C_r^1\left(\sum_{j=0}^{k}p_{k-j}\lambda^{k-j}(k-j)^{r-1}\right)\right)\\
&+c_m\left(\sum_{j=0}^{k}p_{k-j}\lambda^{k-j}(k-j)^{m+r}\right)+\cdots+c_0\left(\sum_{j=0}^{k}p_{k-j}\lambda^{k-j}\,(k-j)^r\right)=\sum_{l=0}^{m}d_ln^l
\end{aligned}
$$

Рассмотрим многочлен $\sum_{j=0}^{k}p_{k-j}\lambda^{k-j}\,(k-j)^x$, где $1\le x\le m+r$. Из леммы 3 следует, что данный многочлен при равен 0 тогда и только тогда, когда $x<r$, следовательно, уравнение принимает вид:

$$\sum_{l=0}^{m}n^l\left(\sum_{i=0}^{m-l}c_{m-i}C_{(m-i,l)}\left(\sum_{j=0}^{k}p_{k-j}\lambda^{k-j}\,(k-j)^{r+m-l-i}\right)\right)=\sum_{l=0}^{m}n^ld_l,$$

где $C_{(m-i,l)}$ – сумма произведений числа сочетании, соответствующая c_{m-i} для n^l. Таким образом, получается уравнение, похожее на то, что было доказано в прошлом пункте, только с немного другими коэффициентами. Так как $\sum_{j=0}^{k}p_{k-j}\lambda^{k-j}\,(k-$

$j)^{r+m-l-i}$ и $C_{(m-i,l)}$ не равны 0, то приравнивая коэффициенты, как в прошлом пункте, всегда можно найти набор параметров $\{c_{m-i}\}$.

Практическая часть

Выбор средств реализации

В качестве языка программирования был выбран Python в силу своей многофункциональности и наличия большого количества библиотек. В связи с тем, что при решении осуществляется большое количество математических преобразований, требующих анализ символьных выражений, было принято решение использовать библиотеку SymPy, предоставляющую возможности компьютерной алгебры. Для удобства используется вычислительная среда Jupyter Notebook.

Входные и выходные данные

На вход программе подается рекуррентное соотношение и k начальных значений, каждое из которых вводится с новой строки. В целях формализации входных данных, соотношения будут задаваться в следующем виде:

a(n+k)+p_{k-1}*a(n+k-1)+p_{k-2}*a(n+k-2)+ ... +p_0*a(n)=$f(n)$,

где $p_{k-1}, p_{k-2}, \dots, p_0$ – произвольные числа, $f(n)$ – функция от n, представимая в виде $\left(\sum_{l=0}^{m} d_l n^l\right) \cdot \lambda^n$.

На выходе получается явный вид заданного рекуррентного соотношения. Так как в качестве среды выбран Jupyter Notebook, то выражение на выходе будет в LaTex формате. Если же вывод осуществить через базовую функцию Python print(), то выражение будет в стандартном виде.

Пример:

Входные данные	Вывод
a(n+2)-a(n+1)-a(n)=0 1 1	$\frac{2^{-n}\sqrt{5}\left(-\left(1-\sqrt{5}\right)^n+\left(1+\sqrt{5}\right)^n\right)}{5}$
a(n+2)-4*a(n+1)+4*a(n)=(n**2+2*n+1)*2**n 1 1	$\frac{2^n\left(n^4-n^2-24n+48\right)}{48}$

Реализация

Так как в ходе решения повсеместно используются символьные выражения, то в программе преимущественно все действия выполняются в типе string и sympify (тип данных для выражений в библиотеке SymPy).

Из библиотеки SymPy используются функции:

- sympy.core.sympify.sympify(a, locals=None, convert_xor=True, strict=False, rational=False, evaluate=None) - используется для преобразования любого произвольного выражения, чтобы его можно было использовать как выражение SymPy
- sympy.core.symbol.symbols(names, *, cls=<class 'sympy.core.symbol.Symbol'>, **args) – определяет, как переменные, несколько символов за раз, строка, подающаяся на вход содержит названия переменных, разделенные запятыми или пробелами.
- sympy.solvers.solvers.solve(f, *symbols, **flags) – выдает список корней системы уравнений относительно заданных переменных
- sympy.simplify.simplify.simplify(expr, ratio=1.7, measure=<function count_ops>, rational=False, inverse=False, doit=True, **kwargs) – упрощает заданное выражение
- subs(*args, **kwargs) - заменяет все случаи первого параметра на второй
- sympy.core.relational.Eq(*lhs, rhs=None, **options*) – создает символьные уравнения

В первой ячейке производится ввод рекуррентного соотношения и его разделение на левую и правую части:

In [ ]:
```
import sympy as s

recurrence_relation = input()
index = recurrence_relation.find("=") + 1
right_part = recurrence_relation[index:]
left_part = recurrence_relation[:index - 1]

print(left_part)
right_part
```

Далее из левой части составляется характеристическое уравнение с помощью манипуляций со строковым типом данных и находится порядок соотношения k:

In [ ]:
```
left_part = left_part.replace("(n)", "**0")
left_part = left_part.replace("(n+", "**")
left_part = left_part.replace(")", "")
characteristic_polynomial = s.sympify(left_part)
k = int(left_part[left_part.find('*') + 2:(left_part.find('+') if ((left_part.find('+') < left_part.fir

print(characteristic_polynomial)
k
```

In [ ]:
```
part.find('-') or left_part.find('-') == -1) and left_part.find('+') != -1) else left_part.find('-'))])
```

С помощью функции solve() находятся корни характеристического уравнения:

In [ ]:
```
a = s.symbols('a')
non_repeating_roots = s.solve(characteristic_polynomial, a)

non_repeating_roots
```

Однако используемая функция выводит список неповторяющихся корней, поэтому необходимо по теореме Безу делить характеристическое уравнение до тех пор, пока количество корней не станет равным k:

In [ ]:
```
decision2 = []
if k > len(non_repeating_roots):
    func = f'(a-{non_repeating_roots[0]})'
    for i in range(1, len(non_repeating_roots)):
        func += f'*(a-{non_repeating_roots[i]})'
    print(func)
    s.simplify(characteristic_polynomial / s.sympify(func))
    decision2 = s.solve(characteristic_polynomial, a)
roots_LORU = non_repeating_roots + decision2

roots_LORU
```

Затем составляется общее решение ЛОРУ:

In [ ]:
```
general_solution_LORU = ''
k = 0
for i in range(len(Non-repeating_roots)):
    for j in range(roots_LORU.count(Non-repeating_roots[i])):
        general_solution_LORU += f'C{k}*n**{j}*({str(Non-repeating_roots[i])})**n+'
        k += 1
general_solution_LORU = general_solution_LORU[:-1]
general_solution_LORU = s.sympify(general_solution_LORU)

general_solution_LORU
```

В последующих 8 ячейках производится вычисление частного решения ЛНРУ, поэтому если правая часть соотношения равна 0, то частное решение тоже приравнивается сразу 0. В данной ячейке определяется степень полинома в правой части m:

In [ ]:
```
if right_part != '0':
    num = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
    ind_m = right_part.find('n**')
    if ind_m == -1:
        if 'n' not in right_part or (right_part.count('n') == 1 and '**n' in right_part):
            m = 0
        else:
            m = 1
    else:
        i = ind_m + 3
        m = ''
        try:
            while right_part[i] in num:
                m += right_part[i]
                i += 1
        except IndexError:
            pass
    m = int(m)

    print(m)
```

Далее определяется параметр λ. В зависимости от того, является ли λ корнем характеристического ЛОРУ, выбирается вид частного решения ЛНРУ и составляется параметрическое уравнение частного решения:

In [ ]:
```
if right_part != '0':
    lamb = 1
    if "**n" in right_part:
        lamb = ''
        i = right_part.find('**n') - 1
        while right_part[i] in num:
            lamb += right_part[i]
            i -= 1
        lamb = int(lamb[::-1])

    print(lamb)

    if lamb not in non_repeating_roots:
        private_solution_LNRU = '('
        for i in range(m + 1):
            private_solution_LNRU += f'C{i}*n**{i}+'
        private_solution_LNRU = private_solution_LNRU[:-1] + f")*{lamb}**n"
    else:
        r = roots_LORU.count(lamb)
        private_solution_LNRU = f'n**{r}*('
        for i in range(m + 1):
            private_solution_LNRU += f'C{i}*n**{i}+'
        private_solution_LNRU = private_solution_LNRU[:-1] + f")*{lamb}**n"
    private_solution_LNRU = s.sympify(private_solution_LNRU)

private_solution_LNRU
```

Чтобы найти параметры для частного решения ЛНРУ, частное решение подставляется в рекуррентное соотношение:

In [ ]:
```
if right_part != '0':
    n = s.symbols('n')
    while "a" in recurrence_relation:
        ind1 = recurrence_relation.find("a")
        ind2 = recurrence_relation[ind1:].find(")") + ind1
        ind_var = recurrence_relation[ind1 + 2:ind2]
        a = f"({str(private_solution_LNRU.subs(n, s.sympify(ind_var)))})"
        recurrence_relation = recurrence_relation.replace(recurrence_relation[ind1:ind2+1], a, 1)

recurrence_relation
```

Так как операции = и == в Python являются операторами присваивания и сравнения, то, если передать строку, содержащую эти операции, SymPy выдаст ошибку. Поэтому для составления уравнений используется функция Eq():

In [ ]:
```
if right_part != '0':
    f = s.Eq(s.simplify(s.sympify("(" + recurrence_relation[:recurrence_relation.find("=")] + ")" + f"/
f
```

In [ ]:
```
elation[:recurrence_relation.find("=")] + ")" + f"/{lamb}**n")), s.sympify(f"{right_part}/{lamb}**n"))
```

Чтобы найти параметры для частного решения ЛНРУ, частное решение подставляется в рекуррентное соотношение. Однако в SymPy нет средств, которые бы смогли подобрать параметры в данном уравнении. Но поделив обе части уравнения на λ^n, получается полином степени m с коэффициентами из параметров. Стоит заметить, что коэффициент при старшей степени зависит только от одного параметра, а все последующие зависят от предыдущего набора параметров и одного нового. Таким образом, приравнивая коэффициенты в левой и правой части, можно выразить каждый параметр. Данную задачу уже может решить функция solve():

In [ ]:
```
symbol = s.symbols(["C" + str(i) for i in range(m + 1)])
if right_part != '0':
    roots = s.solve(f, symbol)

roots
```

Если в прошлом пункте требовалось найти только один параметр, то функция solve() выводит просто число, иначе выводится словарь, в котором ключом является название параметра, а значением само его значение. Из-за разных вариантов типов данных требуется обработать возникающую ошибку с помощью try except:

In [ ]:
```
try:
    roots = list(roots.values())
except AttributeError:
    pass

roots
```

Чтобы получить готовое частное решение остается подставить найденные параметры. Из-за различных типов данных необходимо обработать ошибку и тут:

In [ ]:
```
if right_part != '0':
    for i in range(m + 1):
        try:
            private_solution_LNRU = private_solution_LNRU.subs(symbol[i], roots[i])
        except TypeError:
            private_solution_LNRU = private_solution_LNRU.subs(symbol[i], roots)
else:
    private_solution_LNRU = 0

private_solution_LNRU
```

Зная общее решение ЛОРУ и частное ЛНРУ, для получения общего решения ЛНРУ (для ЛОРУ частное решение просто равно 0) требуется сложить решения:

In [ ]:
```
general_solution_LNRU = str(general_solution_LORU) + "+" + str(private_solution_LNRU)
general_solution_LNRU = s.simplify(s.sympify(general_solution_LNRU))

general_solution_LNRU
```

Так как получения общего решения недостаточно, ибо требуется явный вид соотношения, необходимо найти параметры. Для этого составляется система линейных уравнений относительно параметров, количество уравнений и параметров в точности равно порядку соотношения k. Каждое уравнение составляется подстановкой общего решения в рекуррентное для n от 0 до $k-1$:

In [ ]:
```
n = s.symbols('n')
form_array = []
for i in range(1, k + 1):
    form_array.append(general_solution_LNRU.subs(n, i))

form_array
```

В прошлой ячейки составлялась не сама система уравнений, а только список ее левых частей, так что их требуется приравнять с помощью Eq() к начальным значениям, которые вводятся в данной ячейке. Полученная система линейных уравнений решается с помощью функции linsolve(). Так как функция возвращает множество из кортежа, чтобы получить просто кортеж корней, берется нулевой элемент множества, приведенного к типу список:

In [ ]:
```
symbol = s.symbols(["C"+str(i) for i in range(k)])
a = [s.sympify(input()) for i in range(k)]
d = list(s.linsolve([s.Eq(form_array[i], a[i]) for i in range(k)], symbol))
d = d[0]

d
```

Для получения явного вида остается подставить методом subs() найденные параметры:

In [ ]:
```
symbol2 = s.symbols(["C"+str(i) for i in range(len(d))])
for i in range(len(d)):
    explicit_view = explicit_view.subs(symbol2[i], d[i])
explicit_view = str(explicit_view)

explicit_view
```

Вид полученного уравнения может быть достаточно громоздким, поэтому выражение упрощается функцией simplify():

In [ ]:
```
explicit_view = s.simplify(s.sympify(explicit_view))
explicit_view
```

Заключение

С развитием компьютерных технологий и программирования математический мир все больше отдаляется от дифференциального исчисления. Дифференциальное исчисление хорошо подходит для обработки человеком, требует интеллектуального, творческого подхода. Компьютеру гораздо проще работать с дискретной информацией, поэтому будущее стоит за рекуррентным исчислением. Для того, чтобы можно было бы связать два метода исчисления, требуется умение решать рекуррентные соотношения. Это умение позволяет иметь возможность перевести рекуррентную формулу в функцию более понятную человеку. В интернете существуют различные решатели, но преимущественно они обрабатывают какие-то частные случаи. Есть также система Wolfram|Alpha, она удобна, ибо включает множество различных функций, но ей нужно уметь пользоваться, также многие функции являются платными. Таким образом, данное решение является актуальным и может быть использовано при решении прикладных задач.

Список литературы

1. И.С. Каширский Рекуррентное дифференциальное исчисление. Теория и практические приложения.
2. О.А. Монахова, Н.А. Осминина Рекуррентные последовательности. Алгебра формальных рядов.
3. С.Н. Селезнева Последовательности, определяемые рекуррентными соотношениями. Однородные и неоднородные линейные рекуррентные уравнения (ЛОРУ и ЛНРУ). Общие решения ЛОРУ и ЛНРУ.
4. А.Я. Султанов Дополнительные вопросы алгебры. Рекуррентные последовательности.
5. Документация библиотеки SymPy
6. Краткое описание функций и методов библиотеки SymPy (1)
7. Краткое описание функций и методов библиотеки SymPy (2)

Приложение

```
# Ввод рекуррентного соотношения

"""
    Формат ввода:
    a(n+k)+p(k-1)*a(n+k-1)+p(k-2)*a(n+k-2)+ ... +p(0)*a(n)=f(n),
    где p(k-1), p(k-2), ..., p(0) - коэффициенты, k - порядок ЛРУ, f(n) - функция, представимая в виде, указанном в документации
    Пример:
    a(n+2)-a(n+1)-a(n)=0 - числа Фибоначчи
    a(n+3)+4*a(n+2)-3*a(n)=(n**2+4*n+4)*6**n
"""

import sympy as s

recurrence_relation = input()
index = recurrence_relation.find("=") + 1
right_part = recurrence_relation[index:]
left_part = recurrence_relation[:index - 1]

# для проверки выводится левая и правая часть

# print(left_part)
# print(right_part)

# составляется характеристический многочлен ЛОРУ

left_part = left_part.replace("(n)", "**0")
left_part = left_part.replace("(n+", "**")
left_part = left_part.replace(")", "")
characteristic_polynomial = s.sympify(left_part)
k = int(left_part[left_part.find('*') + 2:(left_part.find('+') if ((left_part.find('+') < left_part.find('-') or left_part.find('-') == -1) and left_part.find('+') != -1) else left_part.find('-'))])

# для проверки выводится характеристический многочлен и порядок соотношения

# print(characteristic_polynomial)
# print(k)

# решение характеристического многочлена

# Замечание: функция solve находит НЕПОВТОРЯЮЩИЕСЯ КОРНИ

a = s.symbols('a')
non_repeating_roots = s.solve(characteristic_polynomial, a)

# вывод корней

# print(non_repeating_roots)
```

```
# составление списка ВСЕХ корней

decision2 = []
if k > len(non_repeating_roots):
    func = f'(a-{non_repeating_roots[0]})'
    for i in range(1, len(non_repeating_roots)):
        func += f'*(a-{non_repeating_roots[i]})'
    # print(func)
    s.simplify(characteristic_polynomial / s.sympify(func))
    decision2 = s.solve(characteristic_polynomial, a)
roots_LORU = non_repeating_roots + decision2

# вывод всех корней

# print(roots_LORU)

# составление общего решения ЛОРУ

general_solution_LORU = ''
k = 0
for i in range(len(Non-repeating_roots)):
    for j in range(roots_LORU.count(Non-repeating_roots[i])):
        general_solution_LORU += f'C{k}*n**{j}*({str(Non-repeating_roots[i])})**n+'
        k += 1
general_solution_LORU = general_solution_LORU[:-1]
general_solution_LORU = s.sympify(general_solution_LORU)

# вывод общего решения ЛОРУ

# print(general_solution_LORU)

# нахождение степени полинома в правой части

if right_part != '0':
    num = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
    ind_m = right_part.find('n**')
    if ind_m == -1:
        if 'n' not in right_part or (right_part.count('n') == 1 and '**n' in right_part):
            m = 0
        else:
            m = 1
    else:
        i = ind_m + 3
        m = ''
        try:
            while right_part[i] in num:
                m += right_part[i]
                i += 1
        except IndexError:
            pass
```

```
    m = int(m)

    # вывод степени

    # print(m)

# нахождение параметра lambda

    lamb = 1
    if "**n" in right_part:
        lamb = ''
        i = right_part.find('**n') - 1
        while right_part[i] in num:
            lamb += right_part[i]
            i -= 1
        lamb = int(lamb[::-1])

    # вывод lambda

    # print(lamb)

    # построение параметрического уравнения частного решения

    if lamb not in non_repeating_roots:
        private_solution_LNRU = '('
        for i in range(m + 1):
            private_solution_LNRU += f'C{i}*n**{i}+'
        private_solution_LNRU = private_solution_LNRU[:-1] + f')*{lamb}**n'
    else:
        r = roots_LORU.count(lamb)
        private_solution_LNRU = f'n**{r}*('
        for i in range(m + 1):
            private_solution_LNRU += f'C{i}*n**{i}+'
        private_solution_LNRU = private_solution_LNRU[:-1] + f')*{lamb}**n'
    private_solution_LNRU = s.sympify(private_solution_LNRU)

# вывод частного решения

    # print(private_solution_LNRU)

# подстановка частного решения в рекуррентное соотношение для поиска параметров

    n = s.symbols('n')
    while "a" in recurrence_relation:
        ind1 = recurrence_relation.find("a")
        ind2 = recurrence_relation[ind1:].find(")") + ind1
        ind_var = recurrence_relation[ind1 + 2:ind2]
        a = f"({str(private_solution_LNRU.subs(n, s.sympify(ind_var)))})"
        recurrence_relation = recurrence_relation.replace(recurrence_relation[ind1:ind2+1], a, 1)
    # print(recurrence_relation)
```

```
# перевод в тип, исполнимый библиотекой SymPy

    f = s.Eq(s.simplify(s.sympify("(" + recurrence_relation[:recurrence_relation.find("=")] + ")" +
f"/{lamb}**n")), s.sympify(f"{right_part}/{lamb}**n"))
    # print(f)

# задание списка параметров, относительно которых будет решаться уравнение

    symbol = s.symbols(["C" + str(i) for i in range(m + 1)])

# нахождение этих параметров

    roots = s.solve(f, symbol)

# их вывод

    # print(roots)

# обработка ошибок из-за разного представления типов

    try:
        roots = list(roots.values())
    except AttributeError:
        pass
    # print(roots)

# подстановка параметров в частное решение

    for i in range(m + 1):
        try:
            private_solution_LNRU = private_solution_LNRU.subs(symbol[i], roots[i])
        except TypeError:
            private_solution_LNRU = private_solution_LNRU.subs(symbol[i], roots)
else:
    private_solution_LNRU = 0

# вывод частного решения ЛНРУ

# print(private_solution_LNRU)

# составление общего решения

general_solution_LNRU = str(general_solution_LORU) + "+" + str(private_solution_LNRU)
general_solution_LNRU = s.simplify(s.sympify(general_solution_LNRU))

# вывод общего решения

# print(general_solution_LNRU)

# составление системы линейных уравнений из первых k значений для поиска параметров для
приведения к явному виду
```

```
# ЕСЛИ ПОСЛЕДОВАТЕЛЬНОСТЬ ЗАДАЕТСЯ ОТ 1 ЭЛЕМЕНТА, ТО НУЖНО В ЦИКЛЕ ПОМЕНЯТЬ РАМКИ НА (1, k + 1)

n = s.symbols('n')
form_array = []
for i in range(1, k + 1):
    form_array.append(general_solution_LNRU.subs(n, i))

# вывод полученных уравнений

# print(form_array)

# ввод начальных значений и нахождение параметров

symbol = s.symbols(["C"+str(i) for i in range(k)])
a = [s.sympify(input()) for i in range(k)]
d = list(s.linsolve([s.Eq(form_array[i], a[i]) for i in range(k)], symbol))
d = d[0]

# вывод параметров

# print(d)

# подстановка параметров в общее решение

symbol2 = s.symbols(["C"+str(i) for i in range(len(d))])
for i in range(len(d)):
    explicit_view = explicit_view.subs(symbol2[i], d[i])
explicit_view = str(explicit_view)

# вывод полученного явного вида

# print(explicit_view)

# упрощение

explicit_view = s.simplify(s.sympify(explicit_view))
print(explicit_view)
```