Криворожский государственный педагогический институт
Кафедра информатики и прикладной математики

А.П. Полищук
С.А. Семериков

ЧИСЛЕННЫЕ МЕТОДЫ В ОБЪЕКТНО-ОРИЕНТИРОВАННОЙ МЕТОДОЛОГИИ

Учебное пособие

Раздел 3: Аналитическое приближение функций
Раздел 4: Численное интегрирование и дифференцирование

Кривой Рог

1998

Полищук А.П., Семериков С.А. Численные методы в объектно-ориентированной методологии. Разделы «Аналитическое приближение функций», «Численное интегрирование и дифференцирование». Учебное пособие. – Кривой Рог: КГПИ, 1998. – 29 с.

Авторы:

Полищук А.П.	к. т. н., с. н. с., доцент кафедры информатики и прикладной математики.
Семериков С.А.	магистр математики.

Рецензенты:

Рашевский Н.А.	к. ф.-м. н., доцент кафедры математики (КГПИ)
Теплицкий И.А.	учитель-методист, зам. директора по научной работе (Центрально-Городская гимназия)

Под общей редакцией доктора физико-математических наук, профессора В.Н. Соловьёва.

Рекомендовано к печати на заседании кафедры информатики КГПИ, протокол №2 от 17.09.98 г.

Підп. до друку 29.10.98
Друк №3. Друк офсетний
Умовн. фарбо-відб. 1,45
Тираж 300

Формат 80x84 1/16.
Умовн. друк. арк. 1,6
Зам. №10-1564

КД

ПІ, 324086, Кривий Ріг-86, пр. Гагаріна, 54

Криворізька міська друкарня
324050, Кривий Ріг-50, пр. Металургів, 28.

Оглавление

3. Аналитическое приближение функций, заданных таблично

3.1. Общая постановка задачи

Представление функции в виде дискретной последовательности значений аргумента (так называемых узлов) и соответствующих значений функции в узловых точках - обычное и естественное представление при обработке данных на цифровой вычислительной машине.

Например, подключенные к машине цифровые преобразователи измерительных устройств поставляют последовательность значений параметра (температуры, давления, расхода, расстояния, скорости и т.п.) в дискретные моменты времени. Мы можем использовать также и дискретные представления аналитических функций достаточно сложной структуры. Если при этом возникает потребность в выполнении таких операций, как интегрирование, дифференцирование функции или просто необходимость оценить ее значения в неузловых точках, то желание осуществить аналитическую замену дискретной последовательности тоже выглядит достаточно естественным. Для упрощения будем считать вначале, что значения функции заданы при равноотстоящих друг от друга значениях аргумента - т.е. если не оговорено альтернативное представление, будем рассматривать задачу приближения при равноотстоящих узлах; это дает возможность представить аргумент функции в виде, например, последовательности целых чисел с выбранным постоянным шагом между ними.

Постановка задачи аналитического приближения (***аппроксимации***) функций и методы ее решения определяются следующими основными факторами, которые необходимо проанализировать в самом начале:

1) В нашем распоряжении числовой ряд значений функции, полученный на некотором интервале значений аргумента - назовем его интервалом наблюдения. Если аналитическая замена этого ряда необходима для последующей работы внутри интервала наблюдения, то соответствующая задача называется ***задачей интерполяции***. Может оказаться, что нас интересуют оценки значений

функции правее интервала наблюдения, т.е. в той области значений аргумента, где значения функции не заданы - в этом случае говорят о ***задаче экстраполяции или прогнозирования***. Наконец, может стоять задача получения достоверной оценки значения функции непосредственно на правой границе интервала наблюдения или в любом его узле - эта задача возникает при наличии существенных ошибок в определении значений функции (так называемых шумов в наблюдениях) и называется ***задачей фильтрации или сглаживания***. Таким образом, задача аналитического приближения табличных функций - это комплексная задача интерполяции, экстраполяции и сглаживания. Её «направленность» в значительной мере определяет методы ее решения. Когда аргументом функции является время, можно говорить о направленности в прошлое, будущее или оценках настоящего.

2) Второй существенный аспект проблемы состоит в определении, что называть «хорошим» или «наилучшим» приближением, т.е. в выборе ***критерия оптимальности приближения***. Классические методы приближения функций используют в качестве критерия оптимальности приближения требование точного совпадения значений приближающей функции с табличными значениями в узлах (приближения по критерию Лагранжа). Преимущество такого подхода - в простоте теории приближения и вычислительных процедур; недостаток - в игнорировании неизбежного в реальных условиях наличия шумов в наблюдениях. Общая задача интерполяции, сглаживания и экстраполяции была впервые математически сформулирована А.Н. Колмогоровым, в начале второй мировой войны ею занимался Норберт Винер в приложении к управлению зенитным огнём, связанным с необходимостью прогнозировать координаты цели на время полета снаряда. Задача слежения за целью всегда связана с ее непредвиденными маневрами и ошибками наблюдения, поэтому сглаживание и упреждение траектории цели - актуальная задача, не вписывающаяся в классическую постановку с точным отслеживанием значений в узлах. В дальнейшем необходимость в сглаживании и упреждении при различных условиях и различных требованиях к качеству и области допустимых прогнозов возникла в различных задачах экономики, метеорологии, автоматическом регулировании и пр.

При необходимости учета ошибок наблюдений чаще всего используют классический критерий минимума суммы квадратов разностей между расчетными по приближающей функции и наблюдаемыми значениями в узлах (***критерий Гаусса***) и его современные модификации.

Другой критерий связывают с именем Чебышева; он состоит в требовании минимизировать максимальную разность между наблюдаемыми и расчетными значениями функции в узлах - так называемый минимаксный критерий. Существуют и другие подходы к оценке качества приближения.

3) Третий ключевой момент состоит в выборе класса приближающих аналитических функций. Основное требование, которое у нас возникает к этим функциям - это независимость результатов аппроксимации от начала отсчета, т.е. от сдвига по последовательности значений аргумента. Другими словами, необходимо, чтобы конечное множество функций выбранного для аппроксимации класса переходило само в себя при замене x на $x+k$. Таким свойством обладают:

- линейные комбинации степенных функций $1, x, x^2, \ldots, x^n$;
- тригонометрические функции $\cos(a_i x)$, $\sin(a_i x)$;
- экспоненциальные функции вида $e^{-a_i z}$, часто встречающиеся в реальных задачах.

Использование любого другого конечного множества аппроксимирующих функций (кроме трех перечисленных) подразумевает наличие естественного начала отсчета, так как выбор начала повлияет на результат - но в большинстве задач нет естественного начала и мы вынуждены выбирать из перечисленных классов функций.

В соответствии с перечисленными классами различают ***полиномиальную*** аппроксимацию, Фурье или ***тригонометрическую*** аппроксимацию и ***экспоненциальную*** аппроксимацию.

Еще одно важное свойство, которое хотелось бы придать множеству аппроксимирующих функций - это инвариантность к изменению масштаба, т.е. чтобы множество не изменялось при замене x на kx. Из рассмотренных множеств таким свойством обладает только множество

$$1, x, x^2, \ldots, x^n,$$

поэтому при отсутствии в задаче естественного масштаба мы вынуждены либо выбирать полиномы, либо воздействовать на результат волевым выбором масштаба. Правда, большинство практических задач, в частности, связанных с функциями времени, имеют естественный масштаб и преобладание полиномиальной аппроксимации над другими видами объясняется большей продвинутостью полиномиальной алгебры и простотой вычислительных процедур с полиномами.

3.2. Общая методика решения задач аппроксимации

Обычно в качестве аппроксимирующей функции выбирают линейную комбинацию функций, выбранных из рассмотренных классов, вида:

$$t(x)=\sum_{i=0}^{n} c_i\varphi_i(x)$$

с действительными коэффициентами c_i. Если такой обобщенный многочлен сформирован и выбран критерий оптимальности приближения, то можно составить и решить систему алгебраических уравнений, линейных относительно коэффициентов c_i:

$$\sum_{i=0}^{n} c_i\varphi_i(x_k)=f(x_k),$$

где k=0, 1, 2, … - порядковый номер узла, $f(x_k)$ - заданное табличное значение функции в k-м узле.

Очевидно, что количество таких уравнений должно быть не меньше n+1, а способ ее решения зависит от критерия приближения.

Если требуется точное проведение аппроксимирующей функции через узловые точки, то есть решается задача интерполяции в постановке Лагранжа, то $k=n+1$, а к системе функций $\varphi_i(x)$ предъявляются следующее требование:

Чтобы для заданной функции f(x), определенной на отрезке [a,b], и набора n+1 узлов $x_0, x_1, \ldots, x_n$ ($x\subset[a,b]$) *существовал и был единственным обобщенный интерполяционный многочлен* $t(x)=\sum_{i=0}^{n} c_i\varphi_i(x)$, *необходимо и достаточно, чтобы любой обобщен-*

ный многочлен (с хотя бы одним ненулевым коэффициентом) по выбранной системе функций $\varphi_i(x)$ *имел на отрезке [a,b] не более n корней.*

В настоящем пособии рассмотрены только основные методы алгебраической аппроксимации в разрезе задач интерполяции функций и сглаживания по методу наименьших квадратов. Специфичные вопросы прогнозирования и фильтрации требуют рассмотрения стохастических подходов и не укладываются в короткий вводный курс.

Алгебраическая интерполяция

Классический интерполяционный полином

Использование в качестве $\varphi_i(x)$ последовательности степенных функций приводит к аппроксимирующей функции в виде классического полинома $\sum_{i=0}^{n} c_i x^i$ и к системе n+1 линейных уравнений вида: $\sum_{i=0}^{n} c_i x_k^i = f(x_k)$ относительно c_i. Определитель этой системы есть определитель Вандермонда, который не равен нулю, если мы не используем повторяющихся узлов. Решение СЛАУ любым из известных методов относительно c_i полностью определяет интерполяционный полином и оценка значения $f(x)$ в любой неузловой точке может быть получена вычислением значения полинома в этой точке. Этот полином можно также использовать в дальнейших аналитических процедурах дифференцирования, интегрирования и пр.

В следующих разделах мы рассмотрим другие формы записи интерполяционных полиномов, но необходимо понимать, что независимо от формы записи два интерполяционных полинома степени n, проведенных через одни и те же n+1 узловые точки, тождественно равны, так как их разность есть полином степени не выше n и его значение обращается в нуль в этих узловых точках, т.е. тождественно равна нулю.

Метод интерполяции Лагранжа

При использовании классического полинома мы сначала получали полином (его коэффициенты), а затем использовали его для интерполяции. Метод Лагранжа предполагает строить интерполяционный полином для каждого вычисляемого значения, объединяя его построение с вычислением. Основная идея этого метода состоит в том, что сначала ищется многочлен, значение которого равно 1 в одной узловой точке и 0 - в остальных. Функция

$$L_j(x)=\frac{\prod_{i=0}^{n}(x-x_i)}{\prod_{i=0}^{n}(x_j-x_i)}$$

при $i\neq j$ есть многочлен степени n, значение которого 1 при $x=x_j$ и 0 если $x=x_i$, $i\neq j$.

Тогда многочлен $f(x_i)L_j(x)$ будет принимать значение $f(x_i)$ в i-й узловой точке и 0 в остальных узлах. Тогда многочлен $f(x)=\sum_{j=0}^{n} f(x_j)L_j(x)$ степени n проходит через n+1 точку (x_i,y_i).

Эта запись неудобна для вычислений и ее приводят к так называемой барицентрической форме. Пусть все $f(x_i)=1$, тогда $\sum_{j=1}^{n} L_j(x)=1$.

Положим $A_j=\frac{1}{\prod_{i}(x_j-x_i)}$ и разделим числитель и знаменатель на $\prod_{j}(x-x_j)$. Получим расчетную формулу:

$$f(x)=\frac{\sum_{j=0}^{n}[A_j f(x_j)/(x-x_j)]}{\sum_{j=0}^{n}[A_j/(x-x_j)]}.$$

Так как метод Лагранжа требует вычисления интерполяционного полинома при каждом определении значения в межузловых промежутках, используют его только при интерполяции на небольшом количестве точек.

Интерполяционный полином Ньютона

Полином Ньютона для интерполяции имеет вид:

$$N_n(x)=A_0+A_1(x-x_0)+A_2(x-x_0)(x-x_1)+\ldots+A_n(x-x_0)(x-x_1)\ldots(x-x_{n-1})=$$

$$= A_0 + \sum_{i=1}^{n} A_i \prod_{j=1}^{i} (x - x_{j-1}). \qquad (1)$$

Равносильный вариант полинома можно получить при симметричной перенумерации узлов и значений функции в исходной таблице от n до 0:

$$N_n(x)=B_n+B_{n-1}(x-x_n)+B_{n-2}(x-x_n)(x-x_{n-1})+\ldots+B_0(x-x_n)(x-x_{n-1})\ldots(x-x_1)=$$

$$= B_n + \sum_{i=1}^{n} B_{n-i} \prod_{j=1}^{i} (x - x_{n-j+1}).$$

Коэффициенты полиномов определяются из условий Лагранжа

$$N_n(x_j)=f(x_j),\ 0\le j\le n$$

Полагаем $x=x_0$, тогда в формуле (1) все слагаемые, кроме A_0, обращаются в нуль и $A_0=f(x_0)$.

Затем полагаем $x=x_1$, тогда $f_0+A_1(x_1-x_0)=f_1$, откуда находим коэффициент

$$A_1=(f(x_0)-f(x_1))/(x_0-x_1)=f_{01},$$

который называется разделенной разностью первого порядка. Величина этой разности близка к первой производной функции $f(x)$ при малом расстоянии между узлами x_0 и x_1.

При $x=x_2$ полином принимает значение

$$N_n(x_2)=f(x_0)+A_1(x_2-x_0)+A_2(x_2-x_0)(x_2-x_1),$$

из условия Лагранжа определим искомый коэффициент

$$A_2=(f_{01}-f_{02})/(x_1-x_2)=f_{012},\ \text{где}\ f_{02}=(f(x_0)-f(x_2))/(x_0-x_2).$$

Величина f_{012} называется разделенной разностью второго порядка, которая при близком расположении x_0, x_1, x_2 будет пропорциональна второй производной функции $f(x)$.

Аналогично при $x=x_3$ находим коэффициент A_3:

$$A_3=(f_{012}-f_{013})/(x_2-x_3=f_{0123}),$$

$$\text{где}\ f_{013}=(f_{01}-f_{03})/(x_1-x_3),\ f_{03}=(f(x_0)-f(x_3))/(x_0-x_3).$$

Для коэффициента A_k методом математической индукции запишем следующее выражение:

$$A_k=(f_{01\ldots k-1}-f_{01\ldots k})/(x_{k-1}-x_k).$$

Полученные результаты сведем в таблицу:

x	$f(x)$	*1*	*2*	*3*	*4*

x_0	$f(x_0)$				
x_1	$f(x_1)$	$f_{01}=\dfrac{f(x_0)-f(x_1)}{x_0-x_1}$			
x_2	$f(x_2)$	$f_{02}=\dfrac{f(x_0)-f(x_2)}{x_0-x_2}$	$f_{012}=\dfrac{f_{01}-f_{02}}{x_1-x_2}$		
x_3	$f(x_3)$	$f_{03}=\dfrac{f(x_0)-f(x_3)}{x_0-x_3}$	$f_{013}=\dfrac{f_{01}-f_{03}}{x_1-x_3}$	$f_{0123}=\dfrac{f_{012}-f_{013}}{x_2-x_3}$	
x_4	$f(x_4)$	$f_{04}=\dfrac{f(x_0)-f(x_3)}{x_0-x_3}$	$f_{014}=\dfrac{f_{01}-f_{04}}{x_1-x_4}$	$f_{0124}=\dfrac{f_{012}-f_{014}}{x_2-x_4}$	$f_{01234}=\dfrac{f_{0123}-f_{0124}}{x_3-x_4}$

Для построения интерполяционного полинома Ньютона используются только диагональные элементы приведенной таблицы, остальные являются промежуточными данными.

Добавление новых узлов в таблицу не изменит уже вычисленных коэффициентов, таблица будет дополнена новыми строками и столбцами разделенных разностей.

Погрешность полиномиальной аппроксимации функции $f(x)$ определяется соотношением:

$$|f(x)-N_n(x)|<\frac{M_{n+1}}{(n+1)!}\prod_{i=1}^{n}|x-x_i|,\ \text{где}\ M_{n+1}=\max|f^{(n+1)}(x)|$$

Так как нам обычно неизвестны заранее производные функции $f(x)$, то по приведенной формуле можно провести апостериорную оценку погрешности, основанную на том, что при достаточно близком расположении узлов разделенные разности являются приближенными оценками производных соответствующего порядка k, деленными на $k!$

Поэтому правая часть приведенного неравенства приближенно совпадает по модулю с новым членом полинома Ньютона, появляющимся при добавлении $(n+1)$-го узла. Таким образом, вычис-

ление модуля каждого из членов суммы (1) позволяет установить, сколько узлов следует использовать для аппроксимации исходной функции с заданной точностью. Если узлы расположены равномерно с шагом h, то наименьшая погрешность будет в интервалах, примыкающих к центральному узлу, за счет минимальной величины произведения в правой части оценки погрешности. Особенно резко возрастает погрешность при попытках экстраполяции.
В центральном интервале оценка погрешности может быть получена по:

$$|f(x)-N_n(x)|< M_{n+1}(h/2)^{n+1}\sqrt{2/\pi n}$$

После определения коэффициентов полинома Ньютона вычисление его значений при конкретных аргументах x наиболее экономично проводить по схеме Горнера, получаемой последовательным вынесением за скобки множителей $(x-x_j)$ в формуле (1):

$$N_n(x)=f_0+(x-x_0)(f_{01}+(x-x_1)(f_{012}+(x-x_2)(f_{0123}+...)...).$$

В отличие от алгоритма вычисления полинома Лагранжа, при интерполяции полиномом Ньютона удается разделить задачи определения коэффициентов и вычисления значений полинома при различных значениях x.

Интерполяция сплайнами

Может показаться, что алгебраической интерполяции всегда сопутствует порядок полинома, равный количеству заданных узлов. На самом деле это не так; представим, что необходимо построить график заданной таблично функции с числом узлов в несколько десятков - нецелесообразность использования столь высоких степеней интерполяционного полинома становится очевидной. Но есть и другая причина для снижения степени интерполяционного полинома: несмотря на выполнение условий Лагранжа в узлах интерполяции, интерполяционная функция может иметь значительное отклонение от аппроксимируемой кривой между узлами - возникает так называемый ***эффект волнистости***. Поэтому осуществляют, как правило, кусочную аппроксимацию заданной функции, разбивая весь набор узлов на группы по 2-4 узла и аппроксимируя функцию на отрезках полиномами степеней не выше третьей. Для задач типа численного интегрирования функций этого вполне достаточно, но, скажем, для упомянутой задачи построения графиков надо позаботиться о «сшивании» соседних полино-

мов в точках соприкосновения, иначе кривые на отдельных участках в этих точках будут иметь различные наклоны (вплоть до разных знаков первых производных) и результирующая кривая не будет гладкой.

Для проведения гладких кривых через узловые точки, помимо условий Лагранжа, накладывают дополнительные требования к интерполяционной кривой.

Метод интерполяции, получивший свое название от упругой металлической линейки, накладываемой на узловые точки аппроксимируемой кривой, называется ***методом сплайновой интерполяции***. По законам упругости металлическая линейка, опирающаяся на ряд точек, проходит между ними по линии с нулевой четвертой производной $\varphi^{(4)}(x)=0$. Если в качестве функции $\varphi(x)$ выбрать полином, то его степень должна быть не выше третьей - этот полином и носит название кубического сплайна, имеющего на интервале $[x_{j-1}, x_j]$ вид:

$$\varphi_i=a_i+b_i(x-x_{i-1})+c_i(x-x_{i-1})^2+d_i(x-x_{i-1})^3,$$

где a_i, b_i, c_i, d_i - коэффициенты сплайна, определяемые из дополнительных условий, $i=1, 2, ..., n$ - номера сплайнов. При сплайн-интерполяции на каждом интервале $[x_{j-1}, x_j]$ строится отдельный полином 3-й степени со своими коэффициентами, которые определяются из условия сшивания соседних сплайнов в узловых точках:

1) выполнение условия Лагранжа $\varphi_i(x_{i-1})=f(x_{i-1})$, $\varphi_i(x_i)=f(x_i)$;
2) непрерывность первой и второй производной в узлах $\varphi_i^{(1)}(x_i)=\varphi_{i+1}^{(1)}(x_i), \varphi_i^{(2)}(x_i)=\varphi_{i+1}^{(2)}(x_i)$.
3) условия на концах могут быть различными - в том числе со свободными концами, т.е. описываться уравнениями прямых и в этом случае иметь нулевые вторые производные:

$$\varphi_1^{(2)}(x_0)=0, \varphi_n^{(2)}(x_n)=0.$$

Алгоритм определения коэффициентов кубических сплайнов с учетом оговоренных условий выглядит так:

Условия Лагранжа в узлах x_{i-1} после подстановки i-го сплайна принимают вид:

$a_i=f(x_{i-1})$, $a_i+b_ih_i+c_ih_i^2+d_ih_i^3=f(x_i)$, где $h_i=x_i-x_{i-1}$, $1 \le i \le n$. (*)

Продифференцировав дважды сплайн по x, получим:

$$\varphi_i^{(1)}=b_i+2c_i(x-x_{i-1})+3d_i(x-x_{i-1})^2,$$

$$\varphi_i^{(2)}=2c_i+6d_i(x-x_{i-1}).$$

Из условий непрерывности производных при переходе в точке x_i от i-го к (i+1)-му сплайну с учетом предыдущих выражений получим:

$$b_i+2c_ih_i+3d_ih_i^2=b_{i+1},\ c_i+3d_ih_i=c_{i+1}.\ (**)$$

Из граничных условий на основании последнего выражения для второй производной получим

$$c_1=0,\ c_n+3d_nh_n=0.\ (***)$$

Вышеприведенные соотношения (*), (**), (***) представляют собой СЛАУ относительно коэффициентов сплайнов a_i, b_i, c_i, d_i. Преобразуем ее так, чтобы неизвестными была только одна группа коэффициентов c_i:

$$d_i=(c_{i+1}-c_i)/3h_j,\ b_i=(f(x_i)-f(x_{i-1})/h_i-(c_{i+1}+2c_i)h_j/3.$$

После подстановки этих выражений в (**) получим уравнение относительно c_i. Для симметрии записи в полученном уравнении уменьшим значение индекса i на единицу

$$h_{i-1}c_{i-1}+2(h_{i-1}+h_i)c_i+h_ic_{i+1}=3[(f(x_i)-f(x_{i-1}))/h_i-(f(x_{i-1})-f(x_{i-2}))/h_{i-1}],$$

где $2\le i\le n$.

При $i=n$ для свободных концов $c_{n+1}=0$. Таким образом, n-1 уравнение для c_i вместе с $c_1=0$ и $c_{n+1}=0$ образуют СЛАУ для вычисления c_i, а b_i и d_i мы уже выразили через c_i. Коэффициенты a_j равны значениям аппроксимируемой функции в узлах $a_i=f(x_{i-1})$.

В каждое из уравнений системы входят только 3 неизвестных - c_{i-1}, c_i, c_{i+1}, поэтому матрица СЛАУ является трёхдиагональной. Для решения таких систем наиболее эффективен метод прогонки (частный случай метода Гаусса). Рассмотрим один из его вариантов применительно к нашей задаче. Чтобы сократить запись, представим последнее уравнение в виде:

$$h_{i-1}c_{i-1}+s_ic_i+h_ic_{i+1}=r_i,$$

где $s_i=2(h_{i-1}+h_i)$, $r_i=3[(f(x_i)-f(x_{i-1}))/h_i-(f(x_{i-1})-f(x_{i-2}))/h_{i-1}]$.

Как и в методе Гаусса, работаем в 2 этапа: в прямом ходе вычисляем значения коэффициентов линейной связи каждого предыдущего неизвестного c_i с последующим c_{i+1}.

Из последнего уравнения при i=2 с учетом граничного условия (***) установим связь коэффициента c_2 с коэффициентом c_3:

$c_2=k_2-l_2c_3$, где $k_2=r_2/s_2$, $l_2=h_2/s_2$ - прогоночные коэффициенты.

Затем, подставляя c_2 в последнее уравнение при i=3, получим линейную связь c_3 с c_4:

$$c_3=k_3-l_3c_4.$$

Аналогично для любых соседних коэффициентов с номерами i, i+1 можно установить линейную связь в виде $c_i=k_i-l_ic_{i+1}$.

В процессе выполнения прямого хода метода прогонки необходимо вычислить значения всех прогоночных коэффициентов k_i, l_i, для которых получены рекуррентные соотношения. Подставим формулу связи (i-1)-го и i-го коэффициентов $c_{i-1}=k_{i-1}-l_{i-1}c_i$ в уравнение и в результате получим:

$$c_i=\frac{r_i-h_{i-1}k_{i-1}}{s_i-h_{i-1}l_{i-1}}-\frac{h_i}{s_i-h_{i-1}l_{i-1}}c_{i+1}.$$

Сравнение этого соотношения с формулой для c_i позволяет записать рекуррентные формулы для определения прогоночных коэффициентов l_i, k_i:

$$k_i=\frac{r_i-h_{i-1}k_{i-1}}{s_i-h_{i-1}h_{i-1}},\ l_i=\frac{h_i}{s_i-h_{i-1}l_{i-1}}.$$

Учитывая граничное условие (***) и соотношение $c_1=k_1-l_1c_2$, а также полагая $c_2\neq0$, задаем начальные коэффициенты $k_1=0$, $l_1=0$. Затем по формуле для k_i, l_i вычислим все n пар прогоночных коэффициентов k_i, l_i. На основании соотношения $c_n=k_n-l_nc_{n+1}$ и граничного условия $c_{n+1}=0$ получим $c_n=k_n$. Далее, последовательно применяя формулу $c_i=k_i-l_ic_{i+1}$ при $i=n$-1, n-2, …, 2 вычислим значения искомых c_{n-1}, c_{n-2}, …, c_2. Эта процедура называется обратным ходом метода прогонки.

После определения всех c_i другие коэффициенты сплайнов вычисляются с помощью уже приведенных их представлений через c_i.

Аппроксимация функций по методу наименьших квадратов

Если мы снимаем требование обязательного прохождения аппроксимирующей функции через узловые точки и заменяем его требованием минимума суммы квадратов разностей между значениями аппроксимирующей и аппроксимируемой функций в узлах, то приходим к ***методу наименьших квадратов***, который не игнорирует наличие ошибок в значения аппроксимируемой функции, а пытается усреднить их влияние на результат аппроксимации. Теория метода и его программная реализация в матричном классе нами уже рассмотрена, а его включение в класс аппроксимирую-

щих функций - дело техники объектно-ориентированного программирования. Поэтому мы не будем повторять уже изложенные материал, отметим только, что МНК используется обычно не для кусочной аппроксимации на отрезках общего интервала (хотя и такой подход возможен), а для аппроксимации на всем интервале задания исходной функции. При использовании ортогональных функций в качестве базисных значительно упрощается вычисление коэффициентов разложения исходной функции по базисным функциям, что позволяет последовательным наращиванием количества членов разложения определить минимальную степень полинома для достижения приемлемой точности аппроксимации.

Программная реализация класса аппроксимирующих функций

Класс аппроксимирующих функций должен, по-видимому, содержать в качестве членов-данных два вектора одинаковой размерности для хранения последовательности узлов и значений заданной функции в этих узлах. Набор функций-членов, помимо обязательного набора конструкторов, должен включать методы реализации рассмотренных алгоритмов и, возможно, подпрограммы для интерфейса пользователя. Ниже приведено содержание включаемого файла с определением класса аппроксимаций, содержащего указанные компоненты с сопровождающими комментариями.

```
#include "polynom.h"
#include "matrix.h"

/*Класс полиномиальных аппроксимаций.*/

template <class YourOwnFloatType>
class Approximate
{
/*Для решения задачи аппроксимации нам понадобится массивы узлов и
значений функции в узлах, т.е. объекты класса vector */
  vector<YourOwnFloatType> x,y;//Узлы и значения в узлах
 public:/*Общедоступные методы */
/*Основной конструктор в качестве параметра принимает два вектора -
вектор узлов и вектор значений в узлах */
```

```
  Approximate(vector<YourOwnFloatType>    _x,    vec-
tor<YourOwnFloatType> _y): x(_x), y(_y)
  {
    /*Проверка размерностей на корректность */
    if(x.getm()!=y.getm())
      throw xmsg("Количество узлов не совпадает с
количеством значений в них");
  /*Наверное, в этом случае задача аппроксимации теряет смысл: */
    if(x.getm()<2)
      throw xmsg("Слишком мало точек");
  }
  /*Конструктор копирования */
  Approximate(Approximate  &_):  x(_.x),  y(_.y)  {}
/*Результатом решения должен быть, естественно, объект класса
polynom*/
  //Прототипы методов класса аппроксимаций
  //Интерполяция каноническим полиномом
  polynom<YourOwnFloatType> classic(),
  //Интерполяция полиномом Ньютона
          newton();
/*Интерполяция кубическими сплайнами. Возвращаемое значение - мас-
сив полиномов, организуемый динамически. По окончании работы па-
мять из под него в обязательном порядке необходимо освободить */
  polynom<YourOwnFloatType>* spline();
/*Аппроксимация функции одной переменной по методу наименьших
квадратов при степенном базисе. Аргумент - порядок полинома */
  polynom<YourOwnFloatType> MNK(int);
};


/*Теперь определения методов класса полиномиальных аппроксимаций */
//Интерполяция каноническим полиномом
template <class YourOwnFloatType>
polynom<YourOwnFloatType>                    Approxi-
mate<YourOwnFloatType>::classic()
{
  /*Матрицы для хранения левой и правой частей системы уравнений */
  matrix<YourOwnFloatType>    mtr(x.getm(),x.getm()),
f(x.getm(),1);

  //Формирование матрицы СЛАУ
  for(int i=0;i<x.getm();i++)
  {
```

```
    /*Правая часть системы - значения функции в узлах */
    f[i][0]=y[i];
    /*Левая часть системы - вектор степеней узлов */
    for(int j=0;j<x.getm();j++)
      mtr[i][j]=pow(x[i],j);
  }
  matrix<YourOwnFloatType> sol=SLAE_Gauss(mtr,f);
/*Вызываем метод Гаусса из матричного класса для решения СЛАУ.
/*Результат решения построенной системы - коэффициенты полинома */
  polynom<YourOwnFloatType> res=x.getm();
  /*формируем полином из коэффициентов */
  for(int i=0;i<x.getm();i++)
    res[i]=sol[i][0];
  return res;/*Возвращаем результат */
}


//Интерполяция полиномом Ньютона
template <class YourOwnFloatType>
polynom<YourOwnFloatType>                    Approxi-
mate<YourOwnFloatType>::newton()
{
  //Формирование коэффициентов полинома
  matrix<YourOwnFloatType> table(x.getm(),x.getm());
/*Первый столбец таблицы - значения в узлах, во всех последующих -
разделённые разности соответствующего порядка */
  for(int i=0;i<x.getm();i++)
    table[i][0]=y[i];
  for(int j=1;j<x.getm();j++)
    for(int i=j;i<x.getm();i++)
      table[i][j]=
   (table[j-1][j-1]-table[i][j-1])/(x[j-1]-x[i]);
/*В диагонали сформированной матрицы находятся коэффициенты полинома в Ньютоновой форме. Для того, чтобы перейти к канонической записи полинома, проделаем следующее: */
/*Объявим и инициализируем два вспомогательных полинома - полиномиальный X и полиномиальную единицу */
  polynom<YourOwnFloatType> Odin,X=2,p;
  Odin[0]=1,X[1]=1; /*инициализируем их */
/*сформируем искомый полином как сумму произведений одночленов соответствующих степеней */
  for(int i=0;i<x.getm();i++)
  {
```

```
  polynom<YourOwnFloatType> temp=table[i][i]*Odin;
    for(int j=0;j<i;j++)
      temp*=(X-x[j]*Odin);
    p+=temp;
  }
  return p;/*возвращаем результат */
}
```

//Интерполяция кубическими сплайнами.
/*Общее количество сплайнов равно количеству межузловых промежутков, т.е. x.getm()-1. В связи с этим результатом выполнения данной функции будет не один полином, а x.getm()-1*/

```
template <class YourOwnFloatType>
polynom<YourOwnFloatType>*                    Approxi-
mate<YourOwnFloatType>::spline()
{
```

/*создаём две матрицы для левой и правой частей СЛАУ, результатом решения которой будут производные второго порядка каждого сплайна */

```
  matrix<YourOwnFloatType>
mtr(x.getm(),x.getm()),right(x.getm(),1);
  //0,n-1 - в этих точках вторая производная равна нулю
  mtr[0][0]=1,right[0][0]=0;
  mtr[x.getm()-1][x.getm()-1]=1,
  right[x.getm()-1][0]=0;
  for(long i=1;i<mtr.getm()-1;i++)
  {
    YourOwnFloatType hi=x[i+1]-x[i],him1=x[i]-x[i-1];
    mtr[i][i-1]=him1;
    mtr[i][i]=2*(him1+hi);
    mtr[i][i+1]=hi;
    right[i][0]=
         6*(-(y[i]-y[i-1])/him1+(y[i+1]-y[i])/hi);
  }
```

/*Находим вторые производные и выражаем через них и исходные данные коэффициенты сплайна в форме Тейлора */

```
  matrix<YourOwnFloatType> d2y=SLAE_Orto(mtr,right);
  vector<YourOwnFloatType> a=x.getm()-1;
  vector<YourOwnFloatType> b=a,c=a,d=a;
  for(long i=0;i<a.getm();i++)
  {
    YourOwnFloatType hi=x[i+1]-x[i];
    a[i]=y[i];
```

```
    b[i]=(y[i+1]-y[i])/hi-
hi*(d2y[i+1][0]+2*d2y[i][0])/6;
    c[i]=d2y[i][0]/2;
    d[i]=(d2y[i+1][0]-d2y[i][0])/(6*hi);
  }
  /*Выделяем память под массив полиномов */
  polynom<YourOwnFloatType>      *arr=new      poly-
nom<YourOwnFloatType>[a.getm()];
/*Составляем два служебных полинома - полиномиальный X и полиномиальную единицу */
  polynom<YourOwnFloatType> Odin,X=2;
  Odin[0]=1,X[1]=1;
  /*Формируем полиномы, описывающие элементарные сплайны */
  for(int i=0;i<a.getm();i++)
    arr[i]=a[i]*Odin+b[i]*(X-x[i]*Odin)+c[i]*((X-
x[i]*Odin)^2)+d[i]*((X-x[i]*Odin)^3);
  return arr;/*Возвращаем указатель на заполненный массив - не забудьте его освободить! */
}


//Аппроксимация по методу наименьших квадратов
/*Аргумент - степень полинома, которым будем аппроксимировать экспериментально полученные данные */
template <class YourOwnFloatType>
polynom<YourOwnFloatType>                    Approxi-
mate<YourOwnFloatType>::MNK(int rng)
{
  matrix<YourOwnFloatType>
mtr(x.getm(),rng+1),/*Прямоугольная матрица измерений */
          f(x.getm(),1);  //Столбец свободных членов

  //Формирование матрицы измерений
  for(int i=0;i<mtr.getm();i++)
  {
    f[i][0]=y[i];
    for(int j=0;j<mtr.getn();j++)
      mtr[i][j]=pow(x[i],j);
  }
  //Реализуем формулу a=((mtr*mtrT)^-1) * (mtrT * f)
  matrix<YourOwnFloatType>
sol=SLAE_Orto((~mtr)*mtr,(~mtr)*f);
```

```
  polynom<YourOwnFloatType> a(sol.getm()); //Полином - решение
  for(long i=0;i<sol.getm();i++)
    a[i]=sol[i][0];
return a;//Возвращаем полином - решение задачи аппроксимации
}

/*Определяем типы данных "действительный полином" и "действительный вектор" */
typedef polynom<double> dpolynom;
typedef vector<double> dvector;

#include <conio.h>

/*Тестирующая программа */
void main()
{
  /*Узлы и значения в узлах */
  double                          xdata[5]={1,2,3,4,5},
ydata[5]={0.5,1,2,3,3.5};
  dvector x(5,xdata),y(5,ydata);
  Approximate<double> test(x,y);/*Объявляем тестовый объект аппроксимационного класса */
  dpolynom what=test.classic();/*Канонический полином */
  cout<<what<<endl<<what(2)<<endl;
  what=test.newton();        /*Ньютоновский полином */
  cout<<what<<endl<<what(2)<<endl;
  what=test.MNK(1);               /*МНК, прямая */
  cout<<what<<endl<<what(2)<<endl;
  what=test.MNK(2);               /*МНК, парабола */
  cout<<what<<endl<<what(2)<<endl;
  what=test.MNK(3);          /*МНК, кубическая парабола */
  cout<<what<<endl<<what(2)<<endl;
  dpolynom  *arr=test.spline();    /*получаем указатель на массив сплайнов */
  cout<<endl<<endl;
  for(long  i=0;i<x.getm()-1;i++)  /*выводим сплайны с попутным тестом их в узлах */
    cout<<arr[i]<<"      "<<arr[i](x[i])
        <<"      "<<arr[i](x[i+1])<<endl;
  delete[] arr;/*освобождаем память из под массива сплайнов */
  getch();
```

}

4. Численное интегрирование и дифференцирование табличных функций

4.1. Численное интегрирование

Методы вычисления определенных интегралов $\int_a^b f(x)dx$ отличаются от методов вычисления неопределенных интегралов $\int_a^x f(x)dx$. В первом случае результатом вычислений будет число, во втором - функция, заданная последовательностью значений для заданной последовательности значений аргумента *x*.

Вычисление определенных интегралов

Классификация методов

Ставится задача вычислить интеграл вида $\int_a^b f(x)dx$, где *a* и *b* - нижний и верхний пределы интегрирования; *f*(*x*) - функция, заданная на отрезке [*a*,*b*] последовательностью значений в узлах, в простейшем случае - равноотстоящих.

К численным методам обращаются также при невозможности записать первообразную функцию аналитически через элементарные функции или когда такая запись имеет слишком сложный вид. В любом случае материалом для вычисления интеграла служит конечная последовательность дискретных значений, но при интегрировании вычисляемых аналитических функций есть возможность изменять шаг между узлами для повышения точности вычислений.

Простейший метод вычисления определенных интегралов состоит в замене подынтегральной функции *f*(*x*) аппроксимирующей

функцией $\upsilon(x)$, для которой первообразная легко выражается в элементарных функциях.

Используемые на практике методы численного интегрирования можно классифицировать по способу аппроксимации подынтегральной функции.

Методы Ньютона-Котеса основаны на полиномиальной аппроксимации и отличаются друг от друга степенью аппроксимирующего полинома. Эти алгоритмы просты и легко программируются.

Сплайновые методы базируются на сплайновой аппроксимации подынтегральной функции, различаются по типу выбранных сплайнов и применяются в задачах обработки данных с использованием сплайнов.

Методы наивысшей алгебраической точности (Гаусса-Кристофеля и др.) применимы в основном к аналитическим функциям, используют неравноотстоящие узлы, расположенные по алгоритму, обеспечивающему минимальную погрешность интегрирования для наиболее сложных функций при заданном количестве узлов.

Методы Монте-Карло используют случайное расположение узлов и дает результат вероятностного смысла.

Независимо от метода, в процессе численного интегрирования необходимо вычислить приближенное значение интеграла и оценить погрешность, которая зависит от числа участков разбиения интервала интегрирования - уменьшается с ростом числа участков за счет более точной аппроксимации и одновременно растет за счет погрешности суммирования частичных интегралов. Эта составляющая с некоторого значения N начинает преобладать, что препятствует чрезмерному дроблению интервала интегрирования.

Мы рассмотрим только методы вычисления определенных интегралов из семейства методов Ньютона-Котеса.

Методы прямоугольников. Это простейшие из класса Ньютона-Котеса методы основаны на аппроксимации подынтегральной функции полиномом нулевой степени - константой на заданном отрезке интервала. Для такой аппроксимации достаточно одной точки - любого значения подынтегральной функции в любом узле; это значение считается постоянным на всем промежутке между соседними узлами. Осмысленный выбор левой или правой границ

шага разбиения в качестве значения на всем частичном интервале сделать трудно даже при наличии априорной информации о поведении функции в интервале интегрирования - метод дает только грубую оценку значения определенного интеграла - априорная нижняя оценка погрешности интегрирования $R_{APRIOR}=\frac{h}{2}\int_{x_0}^{x_n} f^{(1)}(x)dx$.

Для главного члена оценки априорных погрешностей использую конструкцию вида $R_{APRIOR}=Ah^p$, где A - коэффициент, зависящий от метода интегрирования, h - шаг интегрирования, p - порядок метода. Эту зависимость можно использовать для большинства методов численного интегрирования.

Для оценки апостериорной погрешности используют так называемую первую формулу Рунге $R_{APOSTER}=(w_h-w_{kh})/(k^p-1)$, где w_h, w_{kh} - значения некоторой переменной w, вычисленной с шагом h и kh, k - коэффициент увеличения шага. Эта формула позволяет получить главную составляющую апостериорной оценки погрешности двойным просчетом с различными шагами.

Для случая, когда порядок метода неизвестен, используют формулу Эйткена для вычисления k^p для подстановки в первую формулу Рунге:

$$k^p=(w_{kh}-w_{kh}^2)(w_h-w_{kh}).$$

Чтобы использовать этот метод, надо третий раз вычислить w с шагом k^2h.

Метод трапеций получается при замене подынтегральной функции полиномом 1-й степени $P_1(x)$, для чего приходится использовать обе границы частичного интервала. На каждом элементарном отрезке аргумента x участок кривой интегрирования представляет собой отрезок прямой - две ординаты и отрезок оси абсцисс вместе с этой прямой ограничивают фигуру трапецеидальной формы, что и дает название этому методу кусочно-линейной аппроксимации подынтегральной функции. Приближенное значение интеграла определяется площадью трапеции:

$$\int_{x_i}^{x_i+h} f(x)dx = h[f(x_i)+f(x_i+h)]/2+R\,.$$

Оценка априорной погрешности равна $R_{APRIOR}=-\frac{h^2}{12}\int_{x_0}^{x_n} f^{(2)}(x)dx$.

Метод Симпсона получается при замене подынтегральной функции полиномом 2-й степени, при записи в Ньютоновской форме для 3-х узлов

$$P_2(x)=f_0+f_{01}(x-x_0)+f_{012}(x-x_0)(x-x_1),$$

где разделенные разности

$$f_{01}=(f_0-f_1)/(x_0-x_1)=(f_1-f_0)/h,$$
$$f_{012}=(f_{01}-f_{02})/(x_1-x_2)=(f_{02}f_1+f_2)/2h^2$$

при шаге h между узлами.

Заменяя $z=x-x_0$ или $x=z-x_0$, получим полином в виде

$$P_2(z)=f_0+(f_{01}-f_{012}h)z+f_{012}z^2$$

Интеграл от полинома

$$\int_{x_0}^{x_2} P_2(x)dx=\int_0^{2h} P_2(z)dz=(f_0+4f_1+f2)h/3.$$

называют квадратурной формулой Симпсона.

Полная априорная погрешность оценивается величиной $R_{APRIOR}=-\frac{h^4}{180}\int_{x_0}^{x_n} f^{(4)}(x)dx$, если невелико значение четвертой производной, иначе можно получить большую погрешность, чем у методов второго порядка точности. Например, для функции

$$f(x)=-25x^4+45x^2-7$$

при $n=2$ для интервала [-1, 1] метод трапеций дает точный результат, равный 4, а формула Симпсона неправильно определяет даже знак (-8/3).

4.1.1. Программная реализация методов численного интегрирования

Нам не обойтись без класса векторов

```
#include "vector.h"

/*Параметризованный класс для вычисления интегральных сумм */
template <class YourOwnFloatType>
class Integrate
{
  vector<YourOwnFloatType> x,y; //Узлы и значения в узлах
 public:
 /*Конструктор, в качестве параметра принимающий два вектора, содержащих соответственно узлы и значения в них */
```

```
  Integrate(vector<YourOwnFloatType>      _x,      vec-
tor<YourOwnFloatType> _y): x(_x), y(_y)
  {
    /*Проверяем входные данные на корректность */
    if(x.getm()!=y.getm())
      throw xmsg("Количество узлов не совпадает с
количеством значений в них");
    if(x.getm()<2)
      throw xmsg("Слишком мало точек");
  }
  Integrate(Integrate   &_):   x(_.x),   y(_.y)   {}
/*Конструктор копирования */
  //Прототипы методов численного интегрирования
  //Метод прямоугольников
  YourOwnFloatType rectangle_method(),
  //Метод трапеций
                   trapecion_method(),
  //Метод Симпсона - желательно четное число интервалов
                   simpson_method();
};


//Определения методов численного интегрирования
//Метод прямоугольников
template <class YourOwnFloatType>
YourOwnFloatType                                  Inte-
grate<YourOwnFloatType>::rectangle_method()
{
  YourOwnFloatType sum=0;/*Интегральная сумма */
/*В связи с возможностью наличия во входных данных неравноотстоя-
щих узлов шаг интегрирования в этом и последующих методах будем
вычислять как разность между текущим узлом и ближайшим к нему */
  for(int i=0;i<x.getm()-1;i++)
    sum+=(x[i+1]-x[i])*y[i];
  return sum;/*Возвращаем результат */
}


//Метод трапеций
template <class YourOwnFloatType>
YourOwnFloatType                                  Inte-
grate<YourOwnFloatType>::trapecion_method()
{
```

```
  YourOwnFloatType sum=0;
/*Суммируем предполагаемые значения функции в междоузлиях */
  for(int i=0;i<x.getm()-1;i++)
    sum+=(x[i+1]-x[i])*(y[i]+y[i+1])/2;
  return sum;
}


//Метод Симпсона - желательно четное число одинаковых интервалов
template <class YourOwnFloatType>
YourOwnFloatType                                    Inte-
grate<YourOwnFloatType>::simpson_method()
{
  YourOwnFloatType sum=0;
  /*В этом методе на каждом шаге интегрирования интегральная сумма
вычисляется на двух интервалах (по трём точкам)*/
  for(int i=0;i<x.getm()-2;i+=2)
    sum+=(x[i+1]-x[i])*(y[i]+4*y[i+1]+y[i+2])/3;
  return sum;
}

/*Подставляя в шаблон конкретный тип, получаем действительный век-
тор */
typedef vector<double> dvector;

#include <conio.h>

/*Тестирующая функция */
void main()
{
  double limits[2]={.7,1.3};/*Пределы интегрирования */
  /*Количество интервалов */
  #define POINTCOUNT 20
  /*Вектора, содержащие узлы и значения в них */
  dvector x=POINTCOUNT+1,y=POINTCOUNT+1;
  /*Шаг интегрирования */
  double step=(limits[1]-limits[0])/POINTCOUNT;
  for(int i=0;i<=POINTCOUNT;i++)
    x[i]=limits[0]+i*step,
    y[i]=1/sqrt(2*x[i]*x[i]+.3);
  /*Создаём объект, в качестве параметров используя только что запол-
ненные векторы */
  Integrate<double> test(x,y);
```

```
  /*Выводим на экран результаты интегрирования различными методами для сравнения и анализа */
  cout<<test.rectangle_method()<<endl
      <<test.trapecion_method()<<endl
      <<test.simpson_method()<<endl;
  getch();
}
```

4.2. Численное дифференцирование

Первое совершенно тривиальное решение, которое приходит в голову при необходимости вычислить производную табличной функции в той или иной точке (в узле или межузловом промежутке) - это выполнить аналитическую замену легко дифференцируемой функцией (например, полиномом), продифференцировать полученную аппроксимирующую функцию и вычислить значение функции-производной при заданном значении аргумента. Но мы столкнемся при этом с рядом проблем – «волнистостью» аналитической замены из-за слишком высокой степени интерполяционного полинома, построенного по большому количеству узлов или по узлам со слишком большим шагом, влиянием погрешностей различного происхождения. Это может привести (и, как правило, приводит) к тому, что даже первая производная в заданной точке для исходной и аппроксимирующей функций будут отличаться очень сильно, вплоть до несовпадения их знаков.

Дифференцирование табличных функций - в принципе некорректная задача, так как погрешность вычисления производных может многократно превышать погрешность интерполяции самой функции - погрешность вычисления производной равна производной погрешности интерполяции. Поэтому, если есть возможность избежать численного дифференцирования табличных функций - это надо сделать. Если же выхода нет - применяйте диктуемые здравым смыслом приемы для уменьшения потенциальных погрешностей дифференцирования.

К таким приемам относятся методы фильтрации аппроксимирующей функции - удаления из нее высоких частот незначительной амплитуды, близкой к допустимой погрешности аппроксимации. К такому эффекту осреднения приводит использование метода наименьших квадратов со степенью аппроксимирующего полинома значительно ниже количества используемых узлов. При ис-

пользовании в качестве базисных функций ортогональных полиномов Чебышева можно постепенно добавлять члены с более высокими степенями с вычислением для каждой степени величины остаточной дисперсии - когда она уменьшится до удовлетворительного значения или темп ее уменьшения станет малым, наращивание степени полинома следует остановить. Попутно можно вычислять производную в заданной точке и, если ее значения подвержены при наращивании степени полинома значительным изменениям, к получаемому результату надо относиться с недоверием.

Другой способ фильтрации высоких частот состоит в том, что полученную тем или иным способом аппроксимирующую функцию пропускают через инерционное звено - математически это означает использование аппроксимирующей функции в качестве правой части неоднородного линейного дифференциального уравнения и получения в результате решения уравнения функции, используемой для последующего дифференцирования. Если программа для решения дифференциальных уравнений снабжена удобным пользовательским интерфейсом, позволяющим плавно менять коэффициенты уравнения и выводить графики исходной функции, аппроксимирующей функции и функции решения дифуравнения, то можно визуально наблюдать результаты аппроксимации и фильтрации и оценивать допустимость производимых при этом искажений. В следующей главе мы рассмотрим методы численного решения дифференциальных уравнений с примером пользовательского интерфейса с выводом графиков функций для операционной системы Windows и вы сможете воспользоваться ими для проверки возможности использования высказанных рекомендаций.

На этом мы завершим рассмотрение методов численного дифференцирования табличных функций - занятия малоэффективного в части доверия к получаемым результатам.