УДК 536.532.001

В. Г. ЧЕРНОВ

(Владимир)

АНАЛИЗ АЛГОРИТМОВ КУСОЧНО-ЛИНЕЙНОЙ АППРОКСИМАЦИИ СТАТИЧЕСКИХ ХАРАКТЕРИСТИК ДАТЧИКОВ

Задача линеаризации статических характеристик датчиков информационно-измерительных систем имеет большое значение, так как уровень ее решения может оказать существенное влияние на параметры системы.

В литературе известно большое число различных методов линеаризации статических характеристик датчиков [1—3]. Однако в условиях большого разнообразия типов датчиков и их функциональных зависимостей наиболее универсальным следует считать метод кусочно-линейной аппроксимации. Именно этот метод, если точность вычислительных операций ограничить погрешностью 0,01%, позволяет вести сложную функциональную обработку исходной информации в реальном масштабе времени. Различные варианты реализации этого метода устройствами с «жесткой» аппаратной логикой описаны в [4—7]. Главным недостатком такого подхода является то, что каждое устройство ориентировано только на один тип (номинал) датчика и погрешность линеаризации. Более удобным оказывается применение устройств с программируемой логикой — микропроцессоров. Появляется возможность создания универсальных приборов, у которых замена датчика или изменение характеристик устройства осуществляются сменой запоминающего устройства, последние могут быть заранее изготовлены по заказам потребителей.

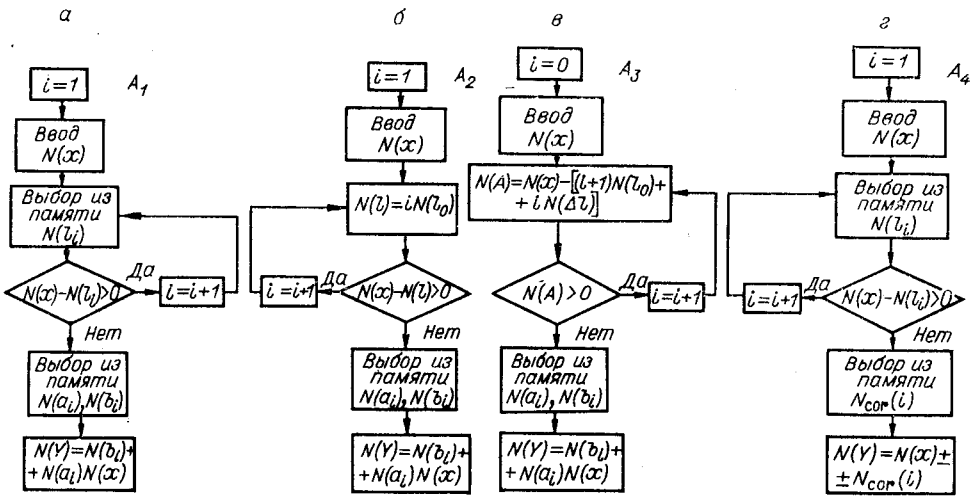
Однако применение микропроцессоров требует выбора алгоритма линеаризации, удовлетворяющего следующим требованиям: количество запоминаемой исходной информации должно быть минимально, а программа линеаризации максимально проста. Последнее требование связано не только с объемом запоминающего устройства, но и с тем, что программирование микропроцессоров остается пока достаточно трудной задачей.

В соответствии с этим для алгоритмов должно быть найдено описание, отражающее как программные, так и аппаратные затраты. Алгоритмические языки высокого уровня не позволяют решить эту задачу в связи с тем, что аппаратные затраты в них не описываются. Графические представления алгоритмов становятся громоздкими и трудноанализируемыми.

Выходом из подобной ситуации является применение формальных языков, используемых для описания организации ЦВМ или любых цифровых схем, представленных на регистровом уровне [8—10]. Наиболее удобным является язык CDL [9, 10].

Рассмотрим четыре варианта алгоритма кусочно-линейной аппроксимации (рисунок, *а—г*).

A_1 — линеаризация с неравномерными промежутками. Для каждого участка аппроксимации необходимо хранить три числа: величину участка l_i и коэффициенты аппроксимации a_i, b_i $(i = 1, 2, \ldots, n)$.



Варианты алгоритма кусочно-линейной аппроксимации:
$N(x)$ — код сигнала датчика, $N(l_i)$ — код длины участка аппроксимации, $N(a_i)$, $N(b_i)$ — коды коэффициентов аппроксимации, $N(\Delta l)$ — код приращения участка аппроксимации.

A_2 — линеаризация с равномерными промежутками. Запоминаются длина интервала $l_0 = \text{const}$ и коэффициенты аппроксимации a_i, b_i $(i = 1, 2, \ldots, n)$.

A_3 — линеаризация для датчиков с убывающей производной статической характеристикой. Запоминаются начальная длина интервала аппроксимации $l_0 = \text{const}$, приращение $\Delta l = \text{const}$ и коэффициенты аппроксимации.

A_4 — линеаризация путем добавления или вычитания определенного числа на соответствующем участке аппроксимации. В память заносятся длина участка аппроксимации l_i, характер операции «+», «—» и корректирующие числа.

Возможно комбинирование алгоритмов A_4, A_2 или A_3. Рассмотрение приведенных блок-схем не дает полного представления о преимуществах и недостатках того или иного алгоритма. Для этого требуется значительная детализация, которую обеспечивает язык CDL.

Прежде чем переходить непосредственно к описанию алгоритмов на языке CDL, сделаем ряд предварительных замечаний.

1. Для сокращения изложения отдельные разъяснения и определения CDL будем приводить в комментариях.

2. Числа представляются с фиксированной запятой «знак + модуль».

3. Ввод информации с АЦП рассматривать не будем, так как эта операция общая для всех алгоритмов.

4. В нашем распоряжении имеется параллельный сумматор, формальное описание которого на CDL дано в [10]. Поскольку этот элемент общий для всех алгоритмов линеаризации, отдельно описывать его не будем.

5. Все операции осуществляются по сигналам генератора синхросигналов, который отдельно не рассматривается.

6. Умножение будем выполнять по методу многократного сложения с использованием на каждом шаге двух битов множителя [10]. Условия 4,6, вообще говоря, не обязательны.

Алгоритм A_1. Согласно правилам CDL, сначала описывается состав устройства, затем алгоритм функционирования:

```
Register   R(S, 1—M)              —  $ буферный регистр, содержащий выходной
                                     код АЦП,
           AC(S, R, Q, 1—M)       —  $ накапливающий регистр,
           MQ(S, 1—M)             —  $ регистр множителя-частного,
           SR(S, 1—M)             —  $ запоминающий регистр,
           SC(1—m)                —  $ счетчик сдвигов m = ent (M/2),
           C(0—k)                 —  $ адресный регистр.
Memory     M(C) = M(0—L, 1—M)     —  $ L объем памяти, M — длина слова.
```

Для каждого узла аппроксимации в памяти отводятся три строки для хранения кодов $N(l_i)$, $N(a_i)$, $N(b_i)$.

```
Subregister R(M) = R(1—M)          —  $ биты в значащей части R,
            AC(R, Q, M) =
            = AC(R, Q, 1 — M)      —  $      »        AC,
            MQ(M) = MQ(1—M)        —  $      »        MQ,
            SR(M) = SR(1—M)        —  $      »        SR.
Casregister   ACMQ(R, Q, M) =
              = ACMQ(R, Q,1—2M) —  $ биты в значащей части каскадного
                                     регистра ACMQ, образованного объединением регистров AC, MQ.

Begin.
|B0|C(k) ← [count up C(k)]; (count up —   увеличение на 1 содержимого регистра C(k)).
SR(M) ← M(C); (comment —                  в регистр SR(M) заносится содержимое памяти по адресу, определяемому адресным регистром C(k). В нашем случае код N(l1)).
SR(M) ← [R(M) sub SR(M)]; (sub —          операция вычитания).
IF SR(M) > 0 THEN
C(k) ← [count up [C(k) ← count up
        C(k)]]; (comment —                увеличение содержимого C(k) подряд на две единицы соответствует переходу на следующий участок аппроксимации).
Go to |B0|;
ELSE C(k) ← count up C(k);
MQ(M) ← M(C); (comment — в регистр множителя заносится из памяти код (N(a1)).
⌈SC(m) ← m;
IF R(S) = MQ(S) THEN AC(S) ← 0, ELSE AC(S) ← 1, AC(R, Q, M) ← 0;
|B1| IF ACMQ(2M — 1,2M) = 11 THEN
AC(R, Q, M) ← AC(R, Q, M) add [shl2R(M) add 1 — 1 — [R(M)]′];
```

([R(M)]′ — дополнительный код содержимого регистра, R(M), add — операция суммирования, shl — операция сдвига влево).

```
IF ACMQ(2M — 1,2M) = 10 THEN
AC(R, Q, M) ← AC(R, Q, M) add [shl R (M)],
IF ACMQ(2M — 1,2M) = 01 THEN
AC(R, Q, M) ← AC(R, Q, M) add R(M),
IF ACMQ(2M — 1,2M) = 00 THEN
AC(R, Q, M) ← AC(R, Q, M);
ACMQ(R, Q, M) ← shr 2 ACMQ(R, Q, M); (shr — операция сдвига вправо).
SC(m) ← count dn SC(m); (count dn — операция уменьшения содержимого регистра
                                                    на 1).
IF SC(m) ≠ 0 THEN Go to |B1|,
ELSE⌋
C(k) ← count up C(k);
SR(M) ← M(C);  (comment — ввод второго коэффициента аппроксимации).
ACMQ(R, Q, M) ← [ACMQ(R, Q, M) add SR (M)].
Terminal ACMQ(R, Q, M). (comment — выходной код снимается с триггеров регистра
                                                    ACMQ(R, Q, M)).
```

Знаками ⌊ ⌉ отмечена процедура умножения, которую для остальных алгоритмов раскрывать не будем, ограничимся лишь записью ⌊Procedure ««multiply»⌉.

Алгоритм A_2. В этом случае необходимо ввести в состав устройства дополнительный регистр для хранения кода длины отрезка аппроксимации MR(M). Информация в регистр MR может заноситься в дополнительном коде. Таким образом, в описание состава надо добавить MR(1 — M) — $ запоминающий регистр кода длины отрезка аппроксимации. Программа работы в этом случае имеет вид

```
Begin.
      |B1| C(k) ← [count up C(k)];
           AC(M) ← [R(M) sub MR(M)];
           IF AC(M) > 0 THEN Go to |B1|,
           ELSE MQ(M) ← M(C);
             ⌈Procedure «multiply»⌋;
           C(k) ← [count up C(k)];
           SR(M) ← M(C);
       ACMQ (R, Q, M) ← [ACMQ(R, Q, M) add SR(M)].
      Terminal ACMQ (R, Q, M).
```

Из представленного описания видно, что число команд, предшествующих Procedure «multiply», почти в два раза меньше, чем в алгоритме A_1. Однако постоянная длина участков аппроксимации может потребовать большого числа этих участков, что может снизить эффект от экономии памяти за счет уменьшения числа запоминаемых величин и сокращения программы.

Алгоритм A_3 для датчиков с монотонно убывающей производной статической характеристики датчиков. При использовании этого алгоритма необходимо организовать хранение не только кода первоначальной длины отрезка аппроксимации l_0, но и приращения Δl. Однако можно запоминать лишь длину отрезка аппроксимации l_0, а приращение определять по младшим разрядам непосредственно либо несложными логическими преобразованиями, сводящимися к инвертированию соответствующих выходов. Для упрощения будем полагать, что приращение представляется H младшими разрядами регистра MR(M), т. е. в описание Subregister добавляется MR(INC) = MR(1 — H) — $ с 1 по H младшие разряды определяются как регистр приращений INC, первые буквы increment — приращение. Программа для алгоритма A_3 запишется следующим образом:

```
Begin.
    C(k) ← [count up C(k)];
    SR(M) ← MR(M);
    |B1|AC(M) ← [R(M) sub SR(M)];
    IF AC(M) > 0 THEN [SR(M) add MR(INC)] Go to |B1|;
    ELSE MQ (M) ← M(C);
    ⌈Procedure «multiply»⌋;
    C(k) ← [count up C(k)];
    R(M) ← M(C);
    ACMQ(R, Q, M) ← [ACMQ(R, Q, M) add R(M)].
    Terminal ACMQ (R, Q, M).
```

Алгоритм A_3 близок к алгоритму A_2 по числу выполняемых операций, но позволяет сократить объем запоминаемой информации за счет более оптимального расположения участков аппроксимации. Недостатком этого алгоритма является его применимость только для датчиков с убывающей производной статической характеристики.

Алгоритм A_4. Этот алгоритм получил широкое распространение [5—7]. Однако при «жесткой логике» к его неуниверсальности относительно типов датчиков добавляется еще один недостаток, связанный с тем, что на практике не удается получить действительно оптимальное разбиение на отрезки аппроксимации из-за невозможности добавить или отнять дробное число импульсов. Поэтому естественным ограничением при разбиении на отрезки является целочисленность корректирующих воздействий. При организации памяти необходимо запоминать характер операции (суммирование или вычитание) на соответствующих участках аппроксимации. Для этой цели в старшем разряде надо хранить «0» (операция суммирования), «1» (операция вычитания). Для рассматриваемого алгоритма описание устройства запишется несколько иначе, чем для всех предыдущих.

```
Register
          R(S, 1 — M)          —  $    буферный регистр,
          AC(S, R, 1 — M)      —  $    накапливающий регистр,
          SR(S, 1 — M)         —  $    запоминающий регистр,
          C(0 — k)             —  $    адресный регистр,
Subregister SR(S)              —  $    бит для записи кода операции.
Memory    M(C) = M(0 — L, 1 — M).
Begin.
          |B0|C(k) ← [count up C(k)];
          AC(M) ← M(C);
          AC(M) ← [R(M) sub AC(M)];
          IF AC(M) > 0 THEN Go to |B0|;
          ELSE C(k) ← [count up C (k)];
          SR(S, M) ← M(C);
          IF SR(S) = 1 THEN AC(R, M) ← [R(M) sub SR(M)];
          ELSE AC(R, M) ← [R(M) add SR(M)].
Terminal  AC(R, M).
```

В пользу последнего алгоритма можно высказать следующие соображения.

1. Обеспечивается оптимальное разбиение на отрезки аппроксимации с точки зрения минимизации их числа, соответственно сокращается объем информации, хранимой в памяти.
2. Сокращается объем оборудования (примерно в полтора раза).
3. Упрощается программа, что приводит к сокращению объема соответствующего запоминающего устройства, а также упрощается программирование и ускоряется сам процесс обработки.

Однако следует отметить, что при выборе алгоритма линеаризации необходимо учитывать, какой тип АЦП используется и каковы его характеристики. Если по техническим требованиям необходима коррекция погрешностей АЦП, которая возлагается на тот же процессор, то в этом случае алгоритм коррекции включает множительно-делительные операции, и тогда алгоритм A_4 сохраняет только одно преимущество — ускорение процесса обработки, поскольку не потребуется обращения к программам выполнения множительно-делительных операций, которые по отношению к программе линеаризации будут выступать как подпрограммы.

В заключение отметим, что линеаризация характеристик датчиков относится к рутинным операциям, которые должны выполняться на нижнем уровне иерархии системы. В этом случае целесообразно ввести распределенный принцип построения, когда наряду с центральным процессором имеются периферийные процессоры — сателлиты, решающие описанную выше задачу. Программа периферийного процессора может быть реализована в заранее отработанном и испытанном приборе. Такая программа не является большей частью основного системного программного обеспечения. А это, естественно, означает, что нужно отлаживать меньший объем программы.

ЛИТЕРАТУРА

1. **Новицкий П. В., Киорринг В. Г., Гутников В. С.** Цифровые приборы с частотными датчиками. Л.: Энергия, 1970.
2. **Воронов А. А. и др.** Цифровые аналоги для систем автоматического управления. М.: Изд-во АН СССР, 1960.
3. **Персин С. М.** Функциональные кодирующие преобразователи с использованием принципа управления частотой следования импульсов.— Труды Главной геофиз. обсерватории им. А. М. Воейкова, 1967, вып. 112.
4. **Методы** и средства измерения температуры: Тематическая подборка.— Приборы и системы управления, 1971, № 9, 10, 11.
5. **Кокорев Е. Н., Чернов В. Г.** Многоканальный цифровой измеритель температуры.— В кн.: Элементы и технические средства управления и регулирования. Новочеркасск: изд. Новочеркасского политех. ин-та, 1977.
6. **Браго Е. Н.** Методы и устройства цифрового преобразования информации в измерительных системах нефтяной промышленности. М.: Недра, 1976.

7. **Коллонтай, Харконен.** Цифровая линеаризация результатов измерений.— Электроника, 1968, № 5.
8. **Schlaeppi H. P.** A Formal Language for Describing Machine Logic Timing and Sequensing (LCTIS).— IEEE Trans on Elec. Comp. Eng., 1964, p. 439—448.
9. **Chu Y.** An Algol-Like Computer Design Language.— Comm. of ACM., 1967, p. 607—615.
10. **Чу Я.** Организация ЭВМ и микропрограммирование. М.: Мир, 1975.

Поступило в редакцию 27 ноября 1979 г.

УДК 681.325 : 621.376

Р. Р. ХАМИТОВ
(Москва)

ОБ ОПТИМАЛЬНОМ ПРЕОБРАЗОВАНИИ ПРИ ВЫБОРЕ ПРИЗНАКОВ В ЗАДАЧАХ РАСПОЗНАВАНИЯ ОБРАЗОВ

При реализации на ЭВМ системы распознавания с большой размерностью вектора наблюдений N_0 желательно иметь возможно меньшее число признаков. Поэтому важное значение имеет такое преобразование исходного пространства наблюдений, которое дало бы возможность проводить классификацию в пространстве признаков меньшей размерности $N \ll N_0$ без существенной потери в точности системы.

Пусть X есть последовательность $1 \times N_0$ случайных векторов наблюдений, искаженных аддитивным гауссовым шумом и принадлежащих к одному из классов (эталонных образов) ω_i $(i = 1, 2, \ldots, m)$. При этом математическое ожидание

$$\overline{X} = \omega_i, \quad X \in \omega_i, \tag{1}$$

и ковариационная матрица

$$\overline{(X - \omega_i)(X - \omega_i)^{\mathrm{T}}} = K_i, \quad X \in \omega_i \tag{2}$$

$(i = 1, 2, \ldots, m)$; т — операция транспонирования.

В [1] показано, что при байесовом классификаторе и равных ковариационных матрицах $K_i = K$ $(i = 1, 2, \ldots, m)$ оптимальной с точки зрения минимизации верхней границы вероятности ошибочной классификации является модификация известного преобразования Карунена — Лоэва, базис которого строится из собственных векторов матрицы $K^{-1}A$, где матрица

$$A = \sum_{i=1}^{m} \sum_{j=1}^{m} p(\omega_i)\, p(\omega_j)\, (\omega_i - \omega_j)(\omega_i - \omega_j)^{\mathrm{T}} \tag{3}$$

определяется расположением средних векторов классов, $p(\omega_i)$ — априорная вероятность классов.

Ясно, что расчет собственных векторов матрицы $K^{-1}A$ — трудоемкая задача даже при не слишком значительных N_0. Поэтому реализация указанного преобразования весьма затруднительна. Этим объясняется то, что на практике чаще применяются не оптимальные, но быстрые преобразования Фурье, Адамара — Уолша, Хаара и т. п.

Однако, проделав несложные линейные преобразования, можно показать, что оптимальное в указанном смысле преобразование Q, которое формируется из собственных векторов $K^{-1}A$, представляется в таком виде:

$$Q = CS^{-1/2}B. \tag{4}$$

В этом выражении строки матрицы B являются собственными векторами матрицы K; S — диагональная матрица, элементы которой есть собственные значения K, а строки матрицы C — собственные векторы матрицы

$$D = \sum_{i=1}^{m} \sum_{j=1}^{m} p(\omega_i)\, p(\omega_j)\, (\mu_i - \mu_j)(\mu_i - \mu_j)^{\mathrm{T}}, \tag{5}$$

где $\mu_i = S^{-1/2}B\omega_i$ $(i = 1, 2, \ldots, m)$.

Таким образом, преобразование Q можно осуществить в следующей последовательности:

1) определяются векторы $X_i - \omega_i$;

2) в соответствии с выражениями (П4), (П5) рассчитываются собственные зна-