УДК 004.021

Никитин Д.А.

ПРИЛОЖЕНИЯ АЛГОРИТМА СИНТЕЗА РЕКУРСИВНЫХ ЦИФРОВЫХ ФИЛЬТРОВ ПО ИМПУЛЬСНОЙ ХАРАКТЕРИСТИКЕ

В статье описано, как можно использовать алгоритм для поиска как точных, так и приближённых функциональных зависимостей, обнаружения сигналов, сжатия данных, упрощения математических выражений, быстрой генерации массива значений заданной функции. Также приведены вероятные пути развития алгоритма.

Введение. В статье [1] описан алгоритм синтеза БИХ-фильтров по импульсной характеристике, работоспособный при условии, что она определяется элементарной математической функцией или линейной комбинацией таких функций. В данной статье будут даны дополнения, поясняющие алгоритм и расширяющие область его применения, а также описаны возможные приложения алгоритма, о которых в вышеупомянутой статье было сказано совсем мало.

Указанный алгоритм успешно работает в случае, когда исходная импульсная характеристика, по которой следует отыскивать цифровой фильтр, определяется любой функцией из множества так называемых функций, известных алгоритму. «Определяется» означает, что отсчёты импульсной характеристики H являются значениями такой функции f, взятыми с равномерным шагом: $h_i = f(i)$, $i = 0, 1, 2, \ldots, n$. Хотя импульсная характеристика БИХ-фильтра бесконечна, алгоритму достаточно лишь некоторое количество её начальных отсчётов, которое зависит от сложности функции. Грубым критерием сложности функции в данном случае может считаться количество её параметров. Найденный фильтр называется фильтром, соответствующим исходной последовательности.

Множество известных алгоритму функций было описано следующим образом:

- полиномиальные функции;
- показательные функции;
- синусоиды (косинусоиды);
- произвольные периодические функции, при условии, что период функции кратен частоте дискретизации;
- любая линейная комбинация (конечная сумма) указанных функций.

Поскольку любая числовая последовательность, полученная табуляцией любой из этих функций, не будет являться сходящейся к нулю последовательностью, то фильтр с такой импульсной характеристикой не будет устойчивым. Поэтому использование алгоритма именно в приложениях цифровой фильтрации представляется сомнительным, пока во множество известных алгоритму функций не войдут какие-либо сходящиеся функции. Однако алгоритм может найти применение и в приложениях, не требующих устойчивости отыскиваемых фильтров. Ниже будет описано несколько таких приложений, в которых алгоритм отыскания БИХ-фильтра, соответствующего некоторой числовой последовательности, является лишь одним из первых шагов методики.

Поиск точных функциональных зависимостей в числовых последовательностях. Это приложение уже описано в статье [2]. Суть предлагаемой методики сводится следующему. Система поиска обладает некоторой базой знаний о функциональных зависимостях, которые он может обнаруживать. Такая база знаний должна содержать:

1. информацию, необходимую для идентификации определённой зависимости;
2. правила для нахождения значений параметров функции из значений коэффициентов соответствующего фильтра.

В качестве идентифицирующей информации для «простых» функций выступают точные значения коэффициентов обратной связи соответствующего фильтра, а для «сложных» функций — значения и соотношения между коэффициентами ЦФ. Примеры таких правил и конкретных значений коэффициентов ЦФ приведены в [1, 2].

На первом этапе методики выполняется алгоритм из [1]. Т.е. для начала исходной последовательности выполняется синтез соответствующего фильтра. Поскольку алгоритм может использоваться для поиска разных функциональных зависимостей, им могут соответствовать фильтры разных порядков. Таким образом, для того, чтобы проверить в последовательности наличие всех функций из базы знаний, надо в худшем случае совершить N попыток синтеза БИХ-фильтра, где N — количество разных порядков ЦФ, соответствующих функциям из базы знаний алгоритма. Для нахождения каждого такого фильтра необходимо $2k$ первых элементов исходной последовательности, где k — порядок фильтра. К счастью, алгоритм из [1], названный А1, построен так, что позволяет избежать непосредственного вычисления БИХ-фильтра каждого порядка. При этом он находит минимальный порядок ЦФ, соответствующего данной последовательности, при котором он является единственным.

После вычисления фильтра некоторого порядка k нужно по его коэффициентам проверить наличие всех функций, которым соответствует фильтр такого порядка. Если для текущего порядка фильтра проверка не дала положительного результата (то есть полученные коэффициенты не соответствуют ни одному правилу из базы знаний), следует проверить фильтры меньших порядков.

Если в результате проверки оказалось, что начальные элементы последовательности не подчиняются ни одной функциональной зависимости из базы знаний, надо отбросить первый рассматриваемый элемент и повторить проверку для оставшейся части последовательности.

Когда найден БИХ-фильтр, коэффициенты которого говорят о том, что элементы последовательности, используемые при его вычислении, описываются функцией известной алгоритму, следует определить, на сколько следующих членов последовательности распространяется найденная функциональная зависимость. Суть такой проверки сводится к вычислению импульсной характеристики найденного фильтра и её поэлементному сравнению с рассматриваемой последовательностью. Первые $2k$ элементов совпадают по определению, так как они использовались при вычислении ЦФ как первые отсчёты импульсной характеристики.

После того, как длина участка последовательности, подчиняющегося найденной зависимости, определена, можно сохранить всю полученную информацию об этом участке и повторить проверку для оставшейся части последовательности. В качестве сохраняемой информации могут выступать индекс первого элемента участка, длина участка (количество членов последовательности), идентификатор типа найденной функции, коэффициенты соответствующего фильтра. Коэффициенты фильтра нужны для вычисления значений параметров функции, поэтому вместо них можно хранить сами параметры.

Таким образом, после подобного анализа будет известны участки исходной последовательности, в точности подчиняющиеся полностью определённым функциям одного переменного.

Далее обсуждаются некоторые моменты приведённого алгоритма: оптимизация перебора порядков фильтра, настраиваемость алгоритма на нужные виды функций и др.

Процесс перебора фильтров всех порядков можно реализовать по-разному. Наиболее разумным представляется перебор от больших порядков к меньшим, т.к. фильтры более высоких позволяют обнаружить более сложные функции, охватывая при этом на этапе вычисления коэффициентов ЦФ большее количество элементов исходной последовательности. Именно так действует алгоритм А1. Если для текущего порядка N фильтр не является единственным, или он является единственным, но его коэффициенты не удовлетворяют ни одному правилу из базы знаний, то следующим вычисляется фильтр порядка N–1. И так до минимального рассматриваемого порядка или пока не будет найден ЦФ, указывающий на наличие некоторой функциональной зависимости, известной алгоритму. Может показаться, что такой перебор не совсем оптимален, т.к. согласно ему первыми будут выполняться вычислительно более сложные действия (решения систем уравнений бóльших порядков), и если на самом деле элементы последовательности

подчиняются одной из наиболее простых функций, либо вообще никакой не подчиняются, то большое количество вычислений будет произведено впустую. Однако была найдена следующая закономерность, позволяющая ускорить переход к фильтру нужного порядка. Если при вычислении ЦФ по алгоритму A1 оказалось, что решаемая система уравнений имеет неполный ранг и он отличается от полного ранга на число $r > 2$, то следующий ЦФ, который следует вычислять, должен иметь порядок на $r/2$ меньше, чем текущий. Вычисление промежуточных ЦФ можно пропустить, так как все соответствующие СЛАУ будут также иметь неполный ранг.

Кроме того, при переборе порядков фильтров от высших к низшим, при формировании каждой следующей системы уравнений не используется новых элементов исходной последовательности (используется только часть тех, что используются на текущем шаге). Поэтому — с учётом архитектуры большинства существующих вычислительных систем — можно утверждать, что с большой долей вероятности все данные, требуемые для формирования новой СЛАУ, уже будут в кэш-памяти. А значит, матрицы всех систем уравнений, кроме первой, будут формироваться очень быстро.

Разумеется, возможны различные способы оптимизации, зависящие от специфики конкретного приложения. Например, если известно, что в анализируемых данных будет преобладать известная функциональная зависимость, то имеет смысл всегда начинать поиск с вычисления фильтра того порядка, который соответствует этой функции.

Стремление найти ЦФ минимального порядка обусловлено желанием найти функцию с наименьшим числом параметров, описывающую элементы последовательности. При этом поиск не предлагается начинать с меньших порядков ЦФ из-за вероятности обнаружения более простых функций, которые будут действовать на коротких участках исходной последовательности, и при этом они не будут давать обнаружить «более сложную» функцию, действующую на более длинном участке. Например вместо длинного участка некоторой периодической функции будет обнаружено много коротких участков, подчиняющихся различным показательным или полиномиальным зависимостям. Если же никакой «сложной» функции на данном участке нет, а действительно имеются только более простые функции (т.е. с меньшим числом параметров), то при спуске от больших порядков ЦФ к меньшим они всё равно будут найдены.

Важным свойством алгоритма является возможность изменения множества искомых функций. Таким образом, можно модифицировать алгоритм для конкретной решаемой задачи. Процесс выбора конкретной модификации выглядит примерно так:

1. определяется множество искомых функциональных зависимостей;
2. в базу знаний вносятся правила для определения каждой функции из множества искомых;
3. определяется множество порядков фильтров, соответствующих искомым функциям;
4. далее в процессе работы алгоритм перебирает порядки ЦФ только из заданного множества.

Например, при помощи ЦФ третьего порядка можно обнаружить параболы, синусоиды и все периодические функции, у которых период функции в 3 раза превосходит период дискретизации (другими словами, в исходной последовательности им соответствуют подпоследовательности с периодом в три отсчёта). Однако не все эти функции могут быть интересны для конкретного приложения. Допустим, нужно искать только синусоиды. Тогда в базе знаний для фильтров третьего порядка будет присутствовать только правило (условия), которому должны удовлетворять коэффициенты ЦФ, соответствующего синусоиде. Если бы правил было больше, то фильтр такого порядка для каждого участка исходной последовательности всё равно вычислялся бы только один раз. Но после вычисления его коэффициенты проверялись бы на соответствие нескольким правилам. Вообще, такие проверки очень просты по сравнению с затратами на само вычисление фильтра, а значит,

количество искомых функций не сильно влияет на скорость поиска. Гораздо сильнее на производительность влияет мощность множества порядков фильтров. Например, заметно ускорить работу можно, отказавшись от поиска всех функций, которым соответствуют ЦФ одного определённого порядка. Тогда фильтры этого порядка уже не нужно вычислять. Разумеется, больший эффект увеличения производительности даёт отказ от фильтров более высоких порядков. Эти рассуждения приводят нас к интересному свойству алгоритма: вычислительные затраты на поиск нескольких различных функций в исходных данных не зависят линейно от количества искомых функций (как это было бы, например, при использовании МНК). Линейной зависимость будет в худшем (очень редком) случае, а чаще же рост вычислительных затрат при росте числа искомых функций будет гораздо медленнее.

В соответствии с описанным алгоритмом — для его проверки и демонстрации возможностей — была разработана реализация [3]. В ней осуществлялся поиск синусоид, показательных и линейных функций, полиномов 2-й, 3-й и 4-й степеней, константных зависимостей (т.е. последовательностей одинаковых элементов), суммы двух синусоид, а также периодических подпоследовательностей с периодами 2, 3, 4 и 5 отсчётов. Соответственно, для этого вычислялись фильтры 2-го, 3-го, 4-го и 5-го порядков.

Поиск приближенных функциональных зависимостей. Исследования показали, что метод пригоден не только для обнаружения функции, проходящих точно через заданные точки (фактически интерполяция), но и для обнаружения аппроксимирующей функции. Это очень важно, так как на практике в любом физическом сигнале присутствуют помехи. Обнаружение аппроксимирующей функции отличается от описанного выше алгоритма тем, что правила используемые после вычисления коэффициентов фильтра немного модифицируются. Модификация основана на том, что если форма сигнала близка к некоторой идеальной форме, то полученные коэффициенты фильтра тоже будут близки к коэффициентам, соответствующим функции, описывающей эту идеальную форму. Например, сигналу с формой параболы соответствует фильтр с коэффициентами обратной связи, равными –3, 3, –1 (это, фактически, соответствующее правило из базы знаний). Таким образом, если для имеющейся последовательности получен фильтр с коэффициентами ОС, равными –3.04, 3.11, –0.97, то можно утверждать, что она аппроксимируется квадратичной функцией.

В соответствии с этим, если требуется обнаружение приближённых функций, все правила из базы знаний модифицируются с учётом того, что все коэффициенты теперь должны принимать не фиксированные значения, а удовлетворять некоторым неравенствам. Также изменятся и условия отражающие взаимосвязь между коэффициентами ЦФ, соответствующих «сложным» функциям.

К сожалению, пока не установлена закономерность связывающая, например, отклонения значений коэффициентов ЦФ от идеальных и среднеквадратичное отклонение членов последовательности на отрезке от идеальной функции. Поэтому значения допустимых отклонений коэффициентов в зависимости от требуемой точности пока представляется возможным получать только на практике, опытным путём.

Возможность поиска приближённых функциональных зависимостей была опробована в реализации приложения метода — алгоритме демодуляции телеметрической информации (ТМИ) с коррекцией искажений. При передаче ТМИ в стандартах РФ сигнал модулируется синусоидой. В зависимости от скорости передачи при модуляции нулей и единиц скачкообразно может изменяться либо только фаза, либо и частота и фаза. Различной может быть и длительность передачи одного бита. Но в любом случае модулированный сигнал состоит из участков синусоид некоторой фазы и частоты. Причины неидеальности сигнала на принимающей стороне при этом известны: несущая на передающей стороне генерируется не идеально, всевозможные помехи в канале передачи, шумы квантования при оцифровке сигнала на принимающей стороне и др.

При этом в случае невозможности 100%-ной корректной демодуляции, бывает

необходимо сохранять все записи сеансов связи в исходном виде. Хранение необработанной информации весьма накладно из-за больших размеров (например, запись минутного сеанса одного канала может занимать десятки мегабайт).

Созданный программный продукт воплощает такие преимущества алгоритма как возможность настройки на обнаружение конкретных функциональных зависимостей, а также возможность использовать алгоритм для обнаружения зашумлённых функциональных зависимостей и, как следствие, для сжатия с потерей информации.

Реализация была модифицирована для обнаружения только синусоид. Таким образом, вычислялись ЦФ только третьего порядка, что сильно ускорило работу программы. Для обнаружения зашумлённой синусоиды использовалось следующее правило:

$$0 < a_1 < 3,$$
$$|a_1 + a_2| \le e_1,$$
$$|a_3 + 1| \le e_2,$$

где a_1, a_2, a_3 — коэффициенты обратной связи ЦФ. Значения $e_1 = e_2 = 0.2$ были получены опытным путём (при обнаружении идеальной синусоиды они должны быть равны нулю). Чем они больше, тем больше исходные данные могут отличаться от идеальной синусоиды.

После обнаружения участка зашумлённой синусоиды её можно заменить на идеальную синусоиду, а значит хранить вместо неё только 4 коэффициента (a_1, b_0, b_1, b_2). Сохранение обработанной (демодулированной) информации сделано в двух вариантах:

1. Сохраняется информация о распознанных участках синусоид, фазовых скачках и нераспознанные участки. Это может пригодиться, если имеются нераспознанные участки. В этом случае этой информации будет достаточно для отображения в программе визуализации, чтобы её обработку мог закончить специалист (в части нераспознанных участков).
2. Если нераспознанных участков нет, значит, найдены все фазовые скачки и все нули и единицы модулированного цифрового потока могут быть однозначно идентифицированы и подвергнуты окончательному декодированию в соответствии с применяемой на передающей стороне схемой кодирования.

Обнаружение сигналов известной формы и интеллектуальный анализ данных. Два предыдущих приложения создают возможность использования алгоритма для обнаружения сигналов известной формы. Существуют области техники, где есть необходимость обнаружения коротких сигналов известной формы в непрерывно поступающих зашумлённых данных. Например, в приложениях радиолокации необходимо обнаруживать искажённые копии лоцирующих сигналов. Такие сигналы, как правило, имеют несложную форму, которую можно описать функцией (функциями), известной предлагаемому алгоритму.

Преимуществами данного алгоритма перед аналогами являются:

- лёгкая настраиваемость на одновременное обнаружение нескольких видов сигналов или сигналов, описываемых некоторой кусочной функцией;
- возможность быстрой аппаратной реализации.

Второе преимущество вытекает из того, что в алгоритме используется цифровая фильтрация, для которой есть эффективные аппаратные решения в виде DSP-процессоров или реализаций на ПЛИС. При этом вне зависимости от сложности искомых функций, при работе алгоритма выполняются только самые простые операции (сложение, вычитание, умножение, сравнительно небольшое количество делений, сдвиговые регистры небольшой ёмкости). Например, при обнаружении синусоидальной функции вычисление синуса не требуется, равно как и при обнаружении полинома высокой степени ничего возводить в степень не надо. Ещё большее преимущество в быстродействии можно получить при одновременном обнаружении нескольких видов сигналов, т.к. выше было указано, что вычислительные затраты растут медленно при увеличении количества различных обнаруживаемых функций.

Другой большой и ещё достаточно новой областью, где также требуется поиск приближённых функциональных зависимостей и обнаружение интересующих зависимостей, является интеллектуальный анализ данных (Data Mining). В настоящее время в таких приложениях для обнаружения и прогнозирования часто используются такие методы как нейросети и генетические алгоритмы. При этом известны такие недостатки этих методов как долгое время работы и необходимость начальной настройки, требующей специальных знаний. Поэтому предложенный алгоритм может удачно дополнить арсенал методов, используемых в приложениях Data Mining.

Сжатие данных. Выше было сказано, что вместо обнаруженного участка некоторой функции можно хранить коэффициенты соответствующего фильтра (да и то не все). Этот факт можно использовать для сжатия данных. Причём, как вытекает из вышесказанного, предложенный алгоритм можно использовать для сжатия, как с потерями информации, так и без них. При сжатии без потерь информации должны искаться только функции, точно описывающие отрезки сжимаемой последовательности. А при сжатии с потерями можно искать и аппроксимирующие функции.

В общем случае метод сжатия, основанный предложенном алгоритме, будет действовать по такой схеме. Просматривая с начала сжимаемые данные, архиватор ищет участки, описываемые (точно или приближенно) какой-нибудь из известных ему функций. Вместо каждого такого участка в архиве будет сохраняться его длина, идентификатор обнаруженной функциональной зависимости и некоторое количество коэффициентов цифрового фильтра. Возможны и другие сочетания, необходимых для восстановления сжатого блока данных, однако данное сочетание было выбрано как самое оптимальное (с точки зрения как занимаемого размера, так и скорости восстановления данных).

Объясним, например, почему было решено сохранять коэффициенты фильтра вместо параметров самой функции. Цель, ради которой они хранятся, одна — это восстановление отсчётов найденной функции. Т.е. например, для участка данных, описываемого полиномом второй степени $y_i = \sum_{k=0}^{2} s_k \cdot i^k$, $i = 0, 1, \ldots, L$, надо сохранить 3 параметра s_k. При этом ему соответствует БИХ-фильтр третьего порядка с шестью коэффициентами (три из них — коэффициенты обратной связи). На первый взгляд хранить коэффициенты невыгоднее. Однако выше уже упоминалось, что определённому типу функций соответствуют конкретные коэффициенты обратной связи ЦФ (для «простых» функций) или некоторые соотношения между коэффициентами ЦФ (для «сложных» функций). Именно на этом и основана возможность обнаружения функциональных зависимостей. Поэтому часть коэффициентов можно не хранить, так как потом их можно будет восстановить, зная тип функциональной зависимости (в частности для этого сохраняется идентификатор функциональной зависимости). Например, для того же полинома второй степени не нужно хранить коэффициенты ОС, так как они будут иметь для такой функции фиксированное значение: –3, 3, –1. Таким образом, необходимо сохранять только оставшиеся три коэффициента — столько же, сколько и параметров функции. Для других видов функций, в том числе и «сложных», получается аналогично — количество коэффициентов фильтра, которые необходимо сохранить, равно количеству параметров функции с учётом свободного члена.

Для восстановления данных, можно использовать два разных способа: восстановление как отсчётов некоторой функции взятой с равным шагом и восстановление исходных данных как импульсной характеристики ЦФ. При этом второй способ, использующий именно коэффициенты соответствующего фильтра, вычислительно эффективнее. Поясним это на примере. Для полинома 4-й степени $f(i) = a \cdot i^4 + b \cdot i^3 + c \cdot i^2 + d \cdot i + e$ для восстановления каждого члена исходной последовательности требуется 10 умножений и 4 сложения.

Если же восстанавливать данные как импульсную характеристику ЦФ, то вычислительные затраты будут следующими. Полиному 4-й степени соответствует ЦФ 5-го порядка с 5-ю коэффициентами a_k и 5-ю коэффициентами b_k:

$$y(n) = \sum_{k=0}^{4} b_k x(n-k) - \sum_{k=1}^{5} a_k y(n-k)$$

Поскольку $x(n)$ в нашем случае является единичным импульсом, то первая сумма имеет ненулевое значение только при вычислении первых пяти значений $y(n)$ и равно по сути b_n, то есть для вычисления не требует даже умножения. Таким образом, восстановление каждого члена исходной последовательности в худшем случае потребует 5 умножений и 4 сложения. При этом не происходит возведения в большую степень, поэтому вероятность переполнения или потери значимости гораздо меньше, чем в классическом случае. Поэтому, если не требуется знать точных выражений функций, описывающих участки исходных данных, (а сжатие — именно такой случай), то не стоит тратить время на вычисление их параметров.

Вышесказанное говорит о том, что при сжатии можно вообще не определять, какой зависимости соответствует найденный ЦФ и не вычислять её параметры. Это хорошо с точки зрения скорости алгоритма, так как эти действия могут быть достаточно трудоёмки, особенно для «сложных» функций. Основная задача алгоритма при сжатии — отыскание БИХ-фильтра, импульсная характеристика которого совпадает (при сжатии без потерь информации) или описывает с допустимой ошибкой (при сжатии с потерями) некоторый участок исходных данных. Определение вида соответствующей функции и её параметров при этом становится задачей вторичной и даже необязательной.

Этот приём использовался при создании пробной реализации архиватора [4], использующего данный алгоритм синтеза БИХ-фильтров по импульсной характеристике. Чтобы эффективно сжимать данные архиватор должен иметь как можно большую базу знаний коэффициентов обратной связи для «простых» функций и базу знаний правил определения «сложных» функций по коэффициентам ЦФ. Эти знания определяют список функций, известных алгоритму. Каждой такой функции поставлен в соответствие уникальный идентификатор. Размер идентификатора был принят равным 1 байту. Т.е. всего максимально такой архиватор мог бы «знать» 256 функций. При обнаружении использовались порядки ЦФ от 2 до 7. С учётом перечисленного в начале статьи стандартного множества известных алгоритму функций (без учёта последнего пункта), при помощи фильтров таких порядков можно обнаружить гораздо меньше функций, чем 256. Поэтому было решено расширить множество известных архиватору функций за счёт линейных комбинаций уже известных функций. Однако таких комбинаций очень много, и вычислять для каждой правила, по которым её можно обнаружить, нерационально. Вместо этого было сделано следующее. Алгоритмом обрабатывались наборы тестовых данных, и затем анализировались коэффициенты всех найденных фильтров, импульсные характеристики которых описывали достаточно длинные участки. Фильтры с наиболее часто встречающимися коэффициентами ОС или некоторыми зависимостями, связывающими коэффициенты фильтра, заносились в базу знаний алгоритма. Некоторые из этих фильтров были позже проанализированы, и оказалось, что их импульсные характеристики действительно точно совпадают со значениями функций, являющихся линейными комбинациями других известных алгоритму функций. Делать это же для всех найденных фильтров не было надобности. Главное, что в исходных данных такие фильтры описывали участки данных такого размера, что при их замене на коэффициенты фильтра достигается сжатие. Подобным образом было найдено несколько десятков функций (фильтров). Тем не менее, в авторской реализации задействовано менее половины идентификаторов. При этом было видно, что при внесении в базу знаний новых правил для обнаружения, тестовые файлы, по которым были обнаружены эти правила, после этого сжимаются лучше.

Так как при сжатии может встретиться функция, не входящая в список известных алгоритму, но, тем не менее, описывающая достаточно длинный участок исходной последовательности, то предусмотрена возможность сохранения и блока неизвестной зависимости. Разумеется, при этом приходится сохранять все коэффициенты фильтра.

Данный архиватор был сделан как универсальный, то есть он выполняет сжатие без

потерь информации. Получившийся коэффициент сжатия хуже, чем у распространённых архиваторов. В скорости сжатия данная реализация также уступает аналогам, а скорость декомпрессии получилась очень хорошей, т.к. при этом в соответствии с уравнением цифровой фильтрации выполняются только самые простые команды — сложение и умножение. Основные пути улучшения сжатия:

1. Использование ЦФ более высоких порядков.
2. Расширение множества функций, известных алгоритму (приоритетными являются нерасходящиеся и медленно расходящиеся функции).
3. Использование сжатия с потерями информации.

Вообще использование данного метода сжатия для универсального архиватора на практике является сомнительным. Данная реализация была сделана, прежде всего, для проверки и отработки алгоритма. Более перспективным представляется использование алгоритма в областях, где, во-первых, в данных ожидается наличие зависимостей, выражаемых математической формулой, а во-вторых, допустимо сжатие с потерей информации. Такой пример был представлен выше — это алгоритм демодуляции телеметрической информации с коррекцией искажений. Полученная в том случае степень сжатия была гораздо выше, чем при сжатии без потерь. Кроме того, было использовано другое свойство алгоритма: возможность настройки на поиск только определённых функций, что послужило значительному ускорению работы.

Упрощение математических выражений. Другим интересным возможным приложением для предлагаемого алгоритма синтеза БИХ-фильтров является упрощение математических выражений. Как известно, современные математические пакеты справляются с этой задачей не очень успешно, притом, что такая функция бывает очень полезна.

Решение такой задачи может быть выполнено по довольно простой методике. Пусть имеется некая математическая функция от одного переменного $f(x)$. Приложение вычисляет последовательность значений функции с постоянным шагом, а затем выполняет действия, описанные в первом пункте данной статьи — «Поиск точных функциональных зависимостей в числовых последовательностях». Если окажется, что полученной последовательности соответствует некоторая функция из базы знаний, задачу упрощения можно считать решённой. Вероятность того, что функция, найденная алгоритмом будет сложнее, чем исходная, практически отсутствует, так как алгоритм построен так, что ищет самую простую функцию, проходящую через заданный набор точек (в отличие от распространённых способов отыскания интерполирующих функций). А точнее — функцию с наименьшим количеством параметров. Например, для последовательности {1, 1, 1, 1, 1, 1, …} будет найдена функция $y=1$, а не синусоида, или многочлен степени, равной длине последовательности минус один. Аналогично, для последовательности {0, –1, 0, 1, 3, 8, …} будет найдена парабола $y=x^2-1$. А не какая-либо функция с бóльшим числом параметров, которых на самом деле бесконечно много.

При этом надо учитывать тот факт, что алгоритм может ошибиться в случае, если исходная функция (а значит и искомая) — периодическая, а шаг, с которым брали её значения — больше половины периода (следует из теоремы Котельникова). Если шаг недостаточно мал, или более того, кратен периоду исходной функции, то будет найдена неверная функция. Однако такой ошибки несложно избежать. Надо всего лишь после того, как найдена некоторая функция, протабулировать исходную функцию с меньшим шагом и ещё раз осуществить поиск. Если найдена другая функция, значит, на предыдущем шаге значения функции $f(x)$ были взяты со слишком большим шагом. Более того, если упрощение выполняется в полуавтоматическом режиме, когда пользователь может указывать некоторые параметры для функции упрощения, то ему часто уже заранее может быть известно о наличии периодичности и значении периода анализируемой функции, и он сможет его указать, дабы ускорить поиск.

Разумеется, такой алгоритм будет успешно упрощать выражения только до известных

ему функций. Поэтому, как и в вышеприведённых приложениях, можно указать на необходимость расширения множества функций, известных алгоритму. Для решения задач упрощения выражений и поиска функциональных зависимостей будет весьма полезно внесение в базу знаний не только линейных комбинаций уже известных функций, но и более сложных комбинаций, например произведений таких функций.

Быстрая генерация подряд идущих значений заданной функции. В разных приложениях ЦОС могут потребоваться массивы значений различных функций, например синуса. Предложенный алгоритм может стать основой для процедуры быстрой генерации значений заданной функции одного переменного, равноотстоящих по оси абсцисс. Для этого до генерации (один раз) должен быть найден БИХ-фильтр, соответствующий нужной функции. Затем, вычисляя L первых отсчётов импульсной характеристики фильтра, получаем значения требуемой функции для значений аргумента 0, 1, 2, 3, …, L–1. Если требуются значения функции для другого диапазона и с другим, но постоянным, шагом, эту задачу легко можно привести к решаемой при помощи замены переменной. При этом функция не изменит свой вид, а изменятся только её параметры (по крайней мере, это выполняется для всех функций, входящих сейчас во множество функций, известных алгоритму, а значит, также выполняется и для их любой линейной комбинации). Например, значения функции $y_1(x) = k_1\sin(w_1 x + \varphi_1) + c$, взятые с равным шагом s на интервале $[t_1, t_2]$, можно вычислить как значения другой функции $y_2(x) = k_2\sin(w_2 x + \varphi_2) + c$, взятые на интервале $[0, t_3]$ с единичным шагом.

Объяснение, почему вычисление значений как отсчётов импульсной характеристики имеет преимущество в скорости перед непосредственным вычислением значений некоторой функции, было дано выше в пункте о сжатии. Причина кроется в разнице количества выполняемых операций и в том, что при вычислении импульсной характеристики используются только простейшие операции сложения и умножения.

В частности был проведён вычислительный опыт для вычисления массива значений синуса. Измерялась скорость заполнения массива тысячи значений синуса в диапазоне $(0, 2\pi)$ с равномерным шагом. Один массив заполнялся с использованием стандартной функции вычисления синуса на языке Си. Второй — при помощи предложенного метода. Вычисления производились на процессорах популярной архитектуры x86. На вычислительных системах разных конфигураций (с процессорами разных поколений — от Pentium II до Pentium-4) скорость авторского метода превосходила скорость вычисления стандартным путем в 5-25 раз. Из-за того, что при генерации каждого члена импульсной характеристики БИХ-фильтра используются значения предыдущих членов, в вычислениях с плавающей запятой имеется накапливаемая погрешность. В данном случае погрешность могла быть оценена сравнением со значениями первого массива. Она составила для первых членов массива — порядка 10^{-18}, для последних — порядка 10^{-12}. В подавляющем большинстве приложений это можно считать приемлемой точностью. К тому же некоторыми алгоритмическими методами можно уменьшить погрешность для последних членов, и таким образом уменьшить среднюю ошибку. Кроме того, погрешность имеется только для определённых целевых функций, когда фильтр имеет нецелые коэффициенты, а в остальных случаях фильтр и его импульсная характеристика получаются целочисленными. Об этом также будет ещё сказано ниже.

Сравнение скорости с вычислением при помощи расширений команд SSE2 и SSE3 не проводилось, так как, во-первых, в данном случае уже будет возможна различная реализация алгоритма заполнения массива, а во-вторых, простота авторского алгоритма позволяет его реализовать практически на любых архитектурах и не имеющих подобных расширений.

Кроме того, стоит упомянуть, что если для работы приложения необходим только один массив, то не имеет большого смысла оптимизировать этот процесс. В этих случаях обычно такой массив обычно вычисляется в начале работы (или просто прошивается в постоянной памяти), а затем используется выборка из массива значений. Предложенный алгоритм стоит использовать при наличии каких-то из указанных ниже условий: таких массивов может быть

несколько, целевые функции заранее точно неизвестны и массивы будут генерироваться во время работы приложения, памяти недостаточно для хранения предвычисленных массивов значений, либо значения целевой функции будут нужны всего один-два раза и при этом в порядке возрастания аргумента.

Перспективы развития алгоритма. Наиболее востребованными направлениями развития предложенного алгоритма синтеза БИХ-фильтров и его приложений, пожалуй, являются следующие:

1. *Расширение на многомерный случай*. На данный момент все сделанные реализации приложений алгоритма являются одномерными. При сжатии данные сжимаются как одномерный поток чисел, при обнаружении — ищутся только одномерные сигналы (функции) и т.д. Однако в приложениях сжатия, обнаружения и Data Mining может найти применение многомерная модификация алгоритма. Например, двухмерная — для сжатия изображений.
2. *Поиск новых функций, для которых существуют соответствующие фильтры*. Выше уже не раз упоминалось о необходимости расширения множества известных алгоритму функций. Это положительно повлияет на потенциальные возможности всех приложений алгоритма. Кроме линейных комбинаций было бы очень полезно определить применимость алгоритма к более сложным комбинациям известных алгоритму функций, например, произведениям функций.
3. *Расширение множества известных алгоритму функций при помощи разложения в ряд Тэйлора*. Тот факт, что при помощи алгоритма можно обнаруживать многочлены любых порядков, наталкивает на мысль, что можно использовать разложения в ряд Тэйлора других функций, неизвестных алгоритму, для их обнаружения с некоторой допустимой ошибкой. Например, для функций логарифма или обратных тригонометрических функций не удалось аналитическим путём доказать существование соответствующего БИХ-фильтра или определить возможный порядок такого фильтра. Однако известны разложения этих функций в бесконечный степенной ряд Тэйлора, являющиеся, по сути, полиномами бесконечной степени. Использование рядов Тэйлора для приближённого вычисления значений функций известно давно. Точность вычисления зависит от количества взятых первых членов разложения функции в ряд. При этом, чем больше слагаемых, тем меньше ошибка вычисления, но больше вычислительных затрат. Поэтому в ресурсоёмких приложениях и системах реального времени следует использовать настолько короткий ряд, насколько позволяет значение допустимой ошибки вычисления. Допустимая ошибка может быть определена из требуемой точности приложения или предельной точности используемых вычислительных средств. После того, как выбрано допустимое значение ошибки, можно, основываясь на разложении интересующей функции, виде остаточного члена ряда, наличии/отсутствии знакопеременности членов ряда, определить минимальное количество членов ряда, которые необходимо вычислять. Также следует учитывать область сходимости ряда. В нашем случае функцию вычислять не надо, а надо её обнаруживать. Поэтому значение ошибки будет означать, насколько обнаруженный участок, описываемый некоторым полиномом (началом ряда Тэйлора), может в каждом отсчёте отличаться от изначально искомой функции. Хотя сам полином здесь вычислять не нужно, уменьшение его степени всё равно будет положительно влиять на время работы алгоритма, так как в работе алгоритма надо будет использовать меньший порядок фильтра.
4. *Синтез целочисленных фильтров*. Известно, что использование целочисленных фильтров выгоднее, чем фильтров с вещественными коэффициентами. Операции с целыми числами выполняются быстрее, целые числа часто занимают меньше

памяти, у DSP-процессоров, поддерживающих операции с плавающей точкой, выше стоимость. Поэтому в практике ЦОС востребованы способы синтеза таких фильтров. Если будут найдены классы сходящихся к нулю функций, для которых предложенным алгоритмом можно найти соответствующий фильтр, либо будут найдены методы отыскания таких функций, то алгоритм можно будет использовать для синтеза устойчивых БИХ-фильтров. При этом было обнаружено, что если члены исходной импульсной характеристики являются целыми числами, то соответствующий ей фильтр в большинстве случаев тоже целочисленный. Исключение составляют случаи, когда в функции, описывающей члены импульсной характеристики, в качестве слагаемого присутствует синус (косинус). Возможность алгоритма работать в целых числах была использована в пробной реализации архиватора.

Так как в приложениях предложенного алгоритма одним из этапов является вычисление импульсной характеристики (т.е. фактически — процесс фильтрации единичного импульса), то хорошим известием можно считать объявление фирмы Intel, о том, что в 2009 году процессоры этой фирмы будут дополнены расширением команд AVX (Advanced Vector eXtensions), содержащим команды для реализации БИХ-фильтрации [5]. Это сделает возможным создание ещё более эффективной реализации алгоритма на ПК.

Заключение. Множество возможных приложений алгоритма синтеза БИХ-фильтров по импульсной характеристике говорит о том, что алгоритм скорее всего будет развиваться и использоваться, даже если он не найдёт применения в ЦОС для своего прямого назначения — синтеза цифровых фильтров.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Никитин Д.А., Ханов В.Х. Синтез рекурсивных цифровых фильтров по импульсной характеристике, определяемой элементарной математической функцией // Цифровая обработка сигналов. 2008, № 3, С. 10-14.
2. Ханов В.Х., Никитин Д.А. Алгоритм анализа числовых последовательностей // Вестник СибГАУ. — Красноярск, 2006. — № 6 (13). — С. 10-14.
3. Никитин Д.А. Программа поиска функциональных зависимостей в числовых последовательностях «FunSearch 1.0» — М.: ВНТИЦ, 2007. — № 50200700388, рег. № ОФАП 7729.
4. Никитин Д.А. Сжатие временных рядов с использованием блочной интерполяции // Информационные технологии моделирования и управления. Науч.-технич. журнал. — Воронеж: Научная книга, 2007. — № 1 (35). — С. 85–89.
5. Астахов И. Реализация БИХ-фильтров с помощью набора инструкций Intel AVX для комплексных чисел с плавающей точкой. — http://softwarecommunity-rus.intel.com/articles/rus/3783.htm.

Nikitin D. A.

AN ALGORITHM OF IIR-FILTERS SYNTHESIS ON THE PULSE RESPONSE APPLICATIONS

How algorithm may be used for searching of exact or approximate functional dependencies, signals detection, data compression, simplification of mathematical expressions, fast generation of prescribed function values array is described in the article. Also possible ways of algorithm evolution are given.