УДК 621.397

В.Ф. Одиноков

АВТОМАТИЗИРОВАННЫЙ ПОИСК ДОПОЛНЕНИЙ ТОПОЛОГИЧЕСКИХ ПУТЕЙ В LCR-СХЕМАХ

Рассмотрены методика и программа поиска дополнений путей связи между двумя узлами линейных LCR-схем на основе их топологии.

Ключевые слова: *передаточная функция, линейные цепи, топологическое моделирование*.

Введение. В работах [1-3] рассмотрены методика [1] и три программных этапа [2, 3] автоматизированного расчета передаточной функции (PerFun) сигналов от идеального источника (Ist) к ненагружающему приемнику (Pri) в линейных LCR-схемах на основе их топологии [4].

Исходное выражение для PerFun (1) [2], подлежащее автоматизированному расчету, представлено дробью с числителем ChiPerFun и знаменателем ZnPerFun. ChiPerFun определяется путями (Pu) передачи сигналов от источника к приемнику [активные или сигнальные пути Pu (SiPu)], а ZnPerFun – путями, не связанными с источником и приемником [пассивные или структурные (St) пути Pu (StPu)].

Используемые в ChiPerFun сигнальные пути SiPu подразделяются на положительные (Po) и отрицательные (Ot) пути Pu (PoPu и OtPu). Эти пути начинаются в меньшем по номеру узле Uz1 источника Ist [Ist.Uz1 – начальный (Na) узел Uz (NaUz) путей SiPu], проходят через приемник и заканчиваются в большем по номеру узле Uz2 источника [Ist.Uz2 – конечный (Ko) узел Uz (KoUz) путей SiPu]. Таким образом, PoPu и OtPu по отношению к источнику являются внешними путями.

Относительно приемника пути PoPu и OtPu являются как внешними, так и внутренними: PoPu направлены от узла Uz1 приемника Pri (Pri.Uz1) к его узлу Uz2 (Pri.Uz2), а пути OtPu – от Pri.Uz2 к Pri.Uz1. Условно узлы Uz1 источника и приемника можно отнести к положительным, а Uz2 – к отрицательным узлам.

Поиск путей PoPu и OtPu занимает все три упомянутые выше этапа [2, 3]. На первом этапе [2] определяются начальные элементы путей SiPu, присоединенные к узлу NaUz. На втором этапе [3] пути SiPu, не достигшие узла KoUz [незавершенные (Nz) пути Pu (NzPu)] удлиняются – за счет подключения к NzPu других проводимостей схемы – до узла KoUz [завершенные (Za) пути Pu (ZaPu)]. На третьем этапе [3] производится отбор PoPu и OtPu из всех найденных путей ZaPu.

Используемые в ZnPerFun структурные пути StPu рассматриваются между любыми произвольно выбранными узлами NaUz и KoUz схем, образованных при замкнутом источнике и изъятом приемнике. Так как при этом полярность (направление) путей StPu не имеет физического смысла, для их поиска достаточно двух первых этапов [2, 3].

Операции, осуществляющие поиск любых путей Pu, назовем Pu-этапом расчета PerFun, а исследуемые при этом LCR-соединения – Pu-схемами; Pu-схемы, содержащие источник, являются активными, а Pu-схемы, не содержащие его, – пассивными. Pu-этапы разбиваются на Pu-шаги, каждый из которых соответствует поиску одного пути Pu; последовательность Pu-шагов образует Pu-цикл.

Для нахождения ChiPerFun и ZnPerFun кроме самих путей Pu требуется также знание их дополнений Do, определяемых на последующем (четвертом) Do-этапе расчета PerFun при замыкании в Pu-схемах соответствующих путей Pu с образованием производных LCR-конструкций (Do-схем). Как и Pu-, Do-этапы разбиваются на Do-шаги, образующие Do-циклы.

В известных [1-7] (и им подобных) опубликованных материалах инженерная реализация автоматизированного расчета указанных дополнений не представлена.

Целью данной работы является методическая и программная проработка процедуры автоматизированного поиска дополнений Do путей Pu в LCR-схемах на основе их топологии.

Методическая часть. Визуальное (эвристическое) определение дополнений Do любых путей Pu для простых схем тривиально. Для примера рассмотрим активную топологическую Pu-схему по рисунку 1 [2], где рядом с узлами и внутри элементов (кроме источника Ist) даны их номера (при топологических расчетах функции PerFun приемник представляет собой виртуальную проводимость с единичным весом [4]); u, y – напряжения сигналов.

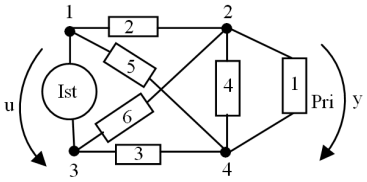

Рисунок 1 – Схема с идеальным источником и виртуальной проводимостью-приемником

Из рисунка 1 видно, что, например, путь Pu (по проводимостям) 2-1-3 от положительного (меньшего) начального узла NaUz=Ist.Uz1=1 источника до его отрицательного (большего) конечного узла KoUz=Ist.Uz2=3 через приемник в направлении от его меньшего (положительного) узла Pri.Uz1=2 к большему (отрицательному) узлу Pri.Uz2=4, является завершенным положительным активным (сигнальным) путем PoPu [2] (пример соответствует одному Pu-шагу Pu-этапа анализа активной Pu-схемы).

Для расчета дополнения Do пути PoPu (DoPoPu) этот путь (элементы 2-1-3) на активной Pu-схеме надо замкнуть (начало Do-шага Do-этапа). Получим производную Do-схему, содержащую замкнутые путем PoPu проводимости 1, 2, 3, 4, 5, 6 [исходная (I) Do-(IDo-)схема]. После отбрасывания замкнутых в IDo-схеме элементов (узлы как указатели соединений пока сохраним) образуется упрощенная (U) Do-(UDo-)схема, в данном случае не содержащая никаких проводимостей (пустая схема). При таком исходе искомая величина DoPoPu пути 2-1-3 по определению [4] равна 1.

Аналогично можно описать Pu- и Do-шаги для сигнального пути 5-1-6 (OtPu) рисунка 1.

В ряде случаев при анализе схем удобно сначала найти все требуемые пути Pu (выполнить Pu-цикл), затем их дополнения Do (Do-цикл). Любую связанную последовательность Pu- и Do-шагов назовем комбинированным Pu-Do-этапом (циклом).

Достаточно сложная (активная или пассивная) Pu-схема при замыкании на ней некоторого пути Pu может породить непустую UDo-схему, которую имеет смысл назвать производной вложенной (внутренней) Pu-схемой, поскольку она предполагает поиск очередных путей Pu, имеющихся в ней [4]. Эта LCR-схема содержит узлы, соединенные между собой путем Pu (производная исходная Pu-(IPu-)схема). После объединения в IPu-схеме соединенных узлов получим упрощенную Pu-(UPu-)схему. Выберем в ней произвольные узлы NaUz и KoUz и найдем все пути Pu1, Pu2,... (Pu-шаги вложенного Pu-этапа) между ними, после чего проведем поиск дополнений DoPu1 (для пути Pu1), DoPu2 (для Pu2),... (Do-шаги вложенного Do-этапа). Замыкание любого из путей Pu1, Pu2,... в последней UPu-схеме может дать очередную непустую UDo-схему и соответственно новую UPu-схему с её собственными путями и их дополнениями для выбранной в этой UPu-схеме пары узлов NaUz, KoUz и т.д. Таким образом, каждая производная непустая UDo-схема прерывают текущий Do-шаг вложенным (В) итерационным (И) Pu-Do-циклом (Ц) – ВИЦ. Последний (в серии) ВИЦ неизбежно (вследствие упрощения производных Pu- и Do-схем) заканчивается нахождением аксиоматических дополнений Do=1 [4] для соответствующих им путей Pu конечной (в последовательности ВИЦ) UPu-схеме. Найденные дополнения Do используются для завершения анализа прерванных предыдущих Do-шагов в обратном порядке – от завершающих простейших производных итерационных схем к предыдущим (более сложным) схемам.

Для многоконтурных и многозвенных LCR-схем эвристический анализ становится трудоемким и длительным процессом, требующим регистрации всех выполняемых операций. Программирование же эвристических методов – тоже не простой и далеко не очевидный процесс. Поэтому перед автоматизацией поиска дополнений Do путей Pu отметим некоторые важные методические аспекты.

Во-первых, любые Do-схемы не содержат источника сигналов. Поэтому они и соответственно вложенные Pu-Do-циклы связаны только со структурными (пассивными) путями.

Во-вторых, если любой текущий Do-шаг приводит к непустой (содержащей проводимости) производной Do-схеме, она становится вложенной Pu-схемой, порождающей свой Pu-Do-цикл, т.е. ВИЦ.

В-третьих, на любом Do-шаге обнаружение ВИЦ должно сопровождаться сохранением результатов всех уже сделанных операций для их завершения при возврате из ВИЦ.

В-четвертых, самым простым способом запоминания результатов прерванного (при обнаружении ВИЦ) Do-шага является сохранение предварительно назначаемого ему автономного программного рабочего (R) модуля (Mo) с индивидуальным итерационным номером N (вариант N=1, формально не соответствующий внутренней итерации, назовем внешней итерацией) при одновременном формировании для прерывающего ВИЦ нового автономного рабочего (R-) модуля с номером N+1. Тогда завершение прерванного Do-шага может быть осуществлено при возврате к его N-у R-модулю после окончания (N+1)-го вложенного итерационного R-модуля ВИЦ.

В-пятых, учитывая назначение R-модулей –

поиск (Poisk) N-го (в итерационной последовательности) дополнения Do пути Pu – с целью унификации их программного кода, целесообразно создание одного трафарета-оригинала (PoiskNDoPu), копируемого для создания любых R-модулей с конкретным значением N.

В-шестых, после выполнения любого автономного рабочего модуля (если в нем не обнаружен ВИЦ) он может быть ликвидирован без потери основного трафарета PoiskNDoPu. Это позволит экономить программную память, не накапливая ненужные, уже отработанные R-модули.

В-седьмых, формирование (Fo) и уничтожение (Un) любых рабочих модулей Mo итерационного поиска дополнений Do путей Pu логично поручить самостоятельной процедуре (FoUnMoDoPu), отделив её функции от операций в самих R-модулях.

Следует подчеркнуть, что пути Pu и их дополнения Do на любом итерационном шаге относятся только к текущим (итерационным) LCR-схемам; получаемые на любом итерационном шаге дополнения используются в расчетах прерванных предыдущих итераций.

Процедура FoUnMoDoPu. С учетом приведенных методических нотаций рассмотрим комментируемый вариант функции FoUnMoDoPu (ниже жирным шрифтом выделены *исполняемые* в редакторе Editor [5] программные коды; многоточие в них является *оператором* продолжения выражения на очередной программной строке; комментарии выполнены обычным шрифтом в виде текста и/или краткого пояснения в круглых скобках и *не могут* быть помещены в редактор Editor без символов %).

Программное определение функции FoUnMoDoPu в одноименном файле имеет следующий вид:

function Do=FoUnMoDoPu(nIt,Pu,MaUzPu,...
MaProSx,NaUz)

Здесь: nIt – итерационный (It) номер (n) создаваемого N-го R-модуля; Pu – путь на текущей итерационной LCR-схеме, для которого находится дополнение Do; MaUzPu – массив (Ma) узлов Uz пути Pu; MaProSx – массив проводимостей (Pro) текущей итерационной схемы (Sx).

Для формирования любого автономного рабочего R-модуля назначаем вспомогательную символьную переменную A –

A='PoiskNDoPu';

Преобразуем (функция num2str) номер nIt в символы и *поставим* (strrep) их на место N в A для конкретизации номера R-модуля –

A = strrep(A,'N',num2str(nIt));

Копируем (copyfile) трафаретный файл PoiskNDoPu.m (m – тип файла) как итерационный рабочий R-модуль A.m с номером N=nIt –

copyfile('PoiskNDoPu.m',[A,'.m']); (здесь [] – оператор объединения или конкатенации)

Преобразуем (str2func) A в handle-функцию (указатель на A) и передаем её переменной d с целью дальнейшего использования:

d=str2func(A);

Вызываем полученную handle-функцию d с необходимыми для её работы фактическими аргументами и передаем результаты её выполнения переменной Do –

Do=d(nIt,Pu,MaUzPu,MaProSx,NaUz);

Уничтожаем (delete) отработанный без образования ВИЦ модуль PoiskNDoPu с N=nIt –

delete([A,'.m']);

end (конец процедуры FoUnMoDoPu)

После выполнения процедуры FoUnMoDoPu происходит автоматический переход в точку её вызова для дальнейшего использования найденного значения Do (выходного аргумента функции FoUnMoDoPu).

Процедура PoiskNDoPu. Программное определение трафаретной функции PoiskNDoPu в одноименном файле представляется как

function Do=PoiskNDoPu(nIt,Pu,...
MaUzPu,MaProSx,NaUz)

При вызове PoiskNDoPu (из строки Do=d(nIt,Pu,MaUzPu,MaProSx,NaUz) функции FoUnMoDoPu) будут известны все фактические входные аргументы nIt, Pu, MaUzPu, MaProSx, NaUz. Для реализации созданного автономного R-модуля PoiskNDoPu с номером N=nIt (модуль Poisk(nIt)DoPu) дополнительно задаем начальное значение его выходного аргумента Do –

Do=0;

Для упрощения производной исходной (IDo-) схемы, получаемой после замыкания пути Pu, сформируем вспомогательный массив MaNoKzElMaProSx номеров (No) короткозамкнутых (Kz) путем Pu элементов (El) массива MaProSx с исходным пустым наполнением –

MaNoKzElMaProSx=[];

Для нахождения номеров nElMaProSx (короткозомкнутых) элементов массива MaProSx у всех их в диапазоне от 1 до numel(MaProSx) (число элементов в MaProSx) –

for nElMaProSx=1:numel(MaProSx)

проверяем принадлежность узлов соответствующих проводимостей массиву MaUzPu. Если узлы Uz1 *и* (оператор &&) Uz2 элемента с номером nElMaProSx *принадлежат* (оператор ==) *каким-либо* (оператор any) узлам в MaUzPu –

if any(MaProSx(nElMaProSx).Uz1==...
MaUzPu)&&...
any(MaProSx(nElMaProSx).Uz2==...
MaUzPu)

эта проводимость будет закорочена путем Pu, и, следовательно, искомый массив MaNoKzElMaProSx пополнится новым nElMaProSx-м элементом массива MaProSx –

MaNoKzElMaProSx=[MaNoKzElMaProSx,...
nElMaProSx];
end (конец оператора
if any(MaProSx(nElMaProSx).Uz1==...
MaUzPu)&&...
any(MaProSx(nElMaProSx).Uz2==...
MaUzPu))
end (конец оператора
for nElMaProSx=1:numel(MaProSx))

С целью корректного исключения короткозамкнутых проводимостей из производной IDo-схемы *сортируем* (sort) массив MaNoKzElMaProSx по *убыванию* (descend) его содержимого (номеров nElMaProSx элементов массива MaProSx):

MaNoKzElMaProSx=...
sort(MaNoKzElMaProSx,'descend');

Из каждого nElMaNoKzElMaProSx-го элемента массива MaNoKzElMaProSx (в каждом nElMaNoKzElMaProSx-м элементе массива MaNoKzElMaProSx хранится номер nElMaProSx короткозамкнутой проводимости массива MaProSx) –

for nElMaNoKzElMaProSx=...
1:numel(MaNoKzElMaProSx)

выделяем его содержимое (упомянутый номер nElMaProSx, обозначенный ниже с целью привязки к массиву MaNoKzElMaProSx как NoKzElMaProSx) –

NoKzElMaProSx=...
MaNoKzElMaProSx(nElMaNoKzElMaProSx);

Последнее позволяет удалить замкнутую проводимость MaProSx(NoKzElMaProSx) из IDo-схемы –

MaProSx(NoKzElMaProSx)=[]; (при этом, так как номера NoKzElMaProSx в MaNoKzElMaProSx сортированы по убыванию, начиная с NoKzElMaProSx, последующие элементы оставшегося итерационного массива MaProSx автоматически перенумеровываются без изменения порядка предыдущих элементов)
end (конец оператора
for nElMaKzElMaProSx=...
1:numel(MaKzElMaProSx))

Если после удаления закороченных путем Pu проводимостей итерационный массив MaProSx в полученной UDo-схеме оказывается пустым –

if numel(MaProSx)==0

искомое дополнение принимается равным 1 [4] –

Do=1;

и работа R-модуля Poisk(nIt)DoPu заканчивается. Полученная величина Do=1 в форме выходного аргумента этого модуля возвращается в точку его вызова – правую часть строки Do=d(nIt,Pu,MaUzPu,MaProSx,NaUz) программы FoUnMoDoPu, выполненный автономный R-модуль Poisk(nIt)DoPu уничтожается оператором delete([A,'.m']), а левая переменная Do указанной строки передается как аргумент функции FoUnMoDoPu в точку её вызова.

Иначе (если не все элементы массива MaProSx закорочены) –

else

обнаружен ВИЦ, и данная UDo-схема становится производной IPu-схемой.

Вначале упростим производную IPu-схему – объединим и перенумеруем соединенные путем Pu узлы схемы. Для этого к каждому nElMaProSx-у элементу итерационного массива MaProSx –

for nElMaProSx=1:numel(MaProSx)

применяем следующее правило: если узел Uz1 этого элемента принадлежит какому-либо узлу массива MaUzPu –

if any(MaProSx(nElMaProSx).Uz1==MaUzPu)

данный узел Uz1 нужно перенумеровать в NaUz, поскольку именно с этого узла начинается любой путь Pu –

MaProSx(nElMaProSx).Uz1=NaUz;
end (конец оператора
if any(MaProSx(nElMaProSx).Uz1==MaUzPu))

Аналогичные рассуждения применяются к узлу Uz2 проводимости MaProSx(nElMaProSx): если –

if any(MaProSx(nElMaProSx).Uz2==MaUzPu)

то

MaProSx(nElMaProSx).Uz2=NaUz;
end (конец оператора
if any(MaProSx(nElMaProSx).Uz2==MaUzPu))
end (конец оператора
for nElMaProSx=1:numel(MaProSx))

Теперь все соединенные путем Pu узлы IPu-схемы перенумерованы в один – NaUz. Получили производную упрощенную IPu-(UPu-)схему. Для её анализа назначаем в ней новые узлы NaUz и KoUz, например, как узлы Uz1, Uz2 первой проводимости MaProSx(1) в итерационном массиве MaProSx –

NaUz=MaProSx(1).Uz1;
KoUz=MaProSx(1).Uz2;

В полученной производной UPu-схеме находим все пути Pu между выбранными узлами NaUz, KoUz (Pu-этап ВИЦ). Для этого вызываем процедуру PoiskNaElPu поиска начальных элементов путей Pu [2] –

[MaNzPu,MaVeNzPu,MaMaUzNzPu,...
MaZaPu,MaVeZaPu,MaMaUzZaPu]=...
PoiskNaElPu(MaProSx,NaUz,KoUz);

Здесь: MaNzPu – массив путей Pu, не заканчивающихся узлом KoUz (незавершенные пути NzPu); MaVeNzPu – массив весов (Ve) путей NzPu (VeNzPu) (вес пути – произведение значений проводимостей его элементов); MaMaUzNzPu – общий массив всех частных массивов (субмассивов) MaUzNzPu узлов путей NzPu; MaZaPu – массив путей Pu, заканчивающихся узлом KoUz (завершенные пути ZaPu); MaVeZaPu – массив весов Ve путей ZaPu (VeZaPu).

Если найденный с помощью PoiskNaElPu массив MaNzPu незавершенных путей NzPu производной UPu-схемы не пустой –

if numel(MaNzPu)>0

вызываем процедуру PoiskPrNzPu поиска продолжений (Pr) этих путей [3]:

```
[MaNzPu,MaVeNzPu,MaMaUzNzPu,...
MaZaPu,MaVeZaPu,MaMaUzZaPu]=...
PoiskPrNzPu(MaNzPu,...
MaVeNzPu,MaMaUzNzPu,MaZaPu,...
MaVeZaPu,MaMaUzZaPu,MaProSx,KoUz);
```

end (конец оператора if numel(MaNzPu)>0)

Если процедурами PoiskNaElPu и PoiskPrNzPu в UPu-схеме найдены завершенные пути ZaPu –

if numel(MaZaPu)>0

необходимо осуществить поиск дополнений Do этих путей (DoZaPu), т.е. поочередно замыкать пути ZaPu и проводить анализ получаемых Do-схем (Do-этап ВИЦ).

Обнаружение ВИЦ соответствует увеличению текущего итерационного шага nIt:

nIt=nIt+1;

Так как путей ZaPu в последнем массиве MaZaPu может быть несколько, при поиске их дополнений каждый nZaPu-й путь –

for nZaPu=1:numel(MaZaPu)

выделяем из MaZaPu как временно рассматриваемый путь Pu=ZaPu для расчета его дополнения DoPu (т.е. DoZaPu) –

Pu=MaZaPu{nZaPu};

Для работы с производными Do- и Pu-схемами (после замыкания Pu=ZaPu) выделяем из общего массива MaMaUzZaPu субмассив узлов этого пути –

MaUzPu=MaMaUzZaPu{nZaPu};

Так как дополнение любого (исходного) пути Pu всегда выражается по формату знаменателя ZnPerFun функции PerFun (1) [2] как сумма произведений весов новых частных путей Pu полученной производной схемы (после замыкания исходного пути Pu) на их дополнения, то применительно к MaZaPu поиск Do в рассматриваемом R-модуле Poisk(nIt)DoPu можно представить последовательно наращиваемым от шага к шагу (его последнего (P) цикла for – Pfor-цикла) результатом Do=Do+Slag2 –

```
Do=Do+MaVeZaPu{nZaPu}*...
FoUnMoDoPu(nIt,Pu,MaUzPu,MaProSx,NaUz);
```

Здесь: ***** – программный оператор арифметического умножения; правая переменная Do символизирует сумму произведений весов VeZaPu уже рассмотренных (в Pfor-цикле модуля Poisk(nIt)DoPu) путей Pu=ZaPu на их дополнения DoZaPu; второе слагаемое (Slag2) в правой части оператора Do=Do+Slag2 является произведением веса следующего пути Pu=ZaPu на его дополнение, рассчитываемое при новом вызове функции FoUnMoDoPu на очередном шаге Pfor-цикла модуля Poisk(nIt)DoPu.

В соответствии с выражением nIt=nIt+1 (левое число nIt является фактическим параметром оператора вызова FoUnMoDoPu) при обращении к процедуре FoUnMoDoPu из Slag2 уже на первом шаге Pfor-цикла модуля Poisk(nIt)DoPu программа FoUnMoDoPu создает (строка copyfile('PoiskNDoPu.m',[A,'.m']) в FoUnMoDoP) и вызывает (строка Do=d(nIt,Pu,MaUzPu,MaProSx,NaUz)) рабочий R-модуль как копию трафарета PoiskNDoPu с номером N=nIt+1 (модуль Poisk(nIt+1)DoPu, где nIt соответствует рассматриваемой, еще не завершенной процедуре Poisk(nIt)DoPu), повторяя эту операцию на всех последующих шагах Pfor-цикла программы Poisk(nIt)DoPu (если эти шаги не прерываются новыми ВИЦ).

Если в процессе выполнения любого рабочего модуля Poisk(nIt+1)DoPu окажется, что numel(MaZaPu)=0, работа его, как было оговорено ранее для Poisk(nIt)DoPu, завершается и значение Do=1 в форме выходного аргумента функции Poisk(nIt+1)DoPu передается в точку её вызова – правую часть строки Do=d(nIt,Pu,MaUzPu,MaProSx,NaUz в FoUnMoDoPu, отработаный модуль Poisk(nIt+1)DoPu уничтожается оператором delete([A,'.m']), а полученный в левой части указанной строки результат Do в виде выходного аргумента функции FoUnMoDoPu передается в точку её вызова – часть Slag2 оператора Do=Do+Slag2 вызывающего модуля Poisk(nIt)DoPu, увеличивая его суммарный результат Do=Do+Slag2, после чего Poisk(nIt)DoPu переходит к выполнению очередного шага своего Pfor-цикла (если не обнаружен новый ВИЦ).

Если в процессе выполнения любого рабочего модуля Poisk(nIt+1)DoPu окажется, что numel(MaZaPu)>0, это означает обнаружение нового ВИЦ, что приводит к повторению всех операций предыдущего вложенного цикла для обнаруженного ВИЦ; при этом формируются

новые вызываемые модули Poisk(nIt+2)DoPu, а вызывающий их модуль Poisk(nIt+1)DoPu сохранится и будет ожидать завершения всех вызываемых им процедур Poisk(nIt+2)DoPu и т.д.

После окончания всех шагов Pfor-цикла модуля Poisk(nIt)DoPu (включая вложенные в него ВИЦ) суммарное (левое) значение Do=Do+Slag2 в виде выходного аргумента функции Poisk(nIt)DoPu через процедуру FoUnMoDoPu передается в точку её вызова, предшествующую рассмотренным операциям.

В любой ситуации (Do=1 или Do=Do+Slag2) указанной предшествующей точкой вызова программы FoUnMoDoPu может быть часть Slag2 в операторе Do=Do+Slag2 Pfor-цикла предыдущего (по отношению к последнему выполненному модулю PoiskNDoPu) R-модуля Poisk(N-1)DoPu, где ранее был обнаружен ВИЦ. При этом результирующее значение Do копии PoiskNDoPu с любым N=nIt через FoUnMoDoPu передается в Slag2 оператора Do=Do+Slag2 модуля Poisk(N-1)DoPu, и все следующие действия будут происходить уже в соответствии с программой Poisk(N-1)DoPu.

Если создание модуля Poisk(N-1)DoPu ранее было вызвано обнаружением ВИЦ в предыдущем по отношению к Poisk(N-1)DoPu модуле Poisk(N-2)DoPu, возврат к Poisk(N-2)DoPu из Poisk(N-1)DoPu и т.д. позволит выполнить все предшествующие аналогичные R-модули, работа которых была прервана обнаруженными вложенными в них итерационными циклами.

После завершения всех R-модулей результат Do из FoUnMoDoPu должен быть передан в точку исходного целевого вызова процедуры FoUnMoDoPu, например в главный управляющий модуль программы вычисления передаточной функции схемы.

end (конец оператора
for nZaPu=1:numel(MaZaPu))
end (конец оператора if numel(MaZaPu)>0)
end (конец оператора if numel(MaProSx)==0)
end (конец процедуры PoiskNDoPu)

Программные испытания. Как известно, многие программы содержат скрытые дефекты, проявляющие себя только в определенных ситуациях. Поэтому проверим модуль FoUnMoDoPu и трафарет PoiskNDoPu для нескольких схем с постепенным их усложнением.

Сначала рассмотрим относительно простую Pu-схему по рисунку 1. Для нее в командном MATLAB-окне (Command Window) заказываем фактические параметры для специализированной функции syms [1-3] – Лаплас-оператор *s* и требуемый ряд символических (тип sym) элементов схемы (их LCR-обозначения и номера 1, 2,... в схеме; источник Ist и приемник Pri не имеют символических параметров). Пример:

syms s R2 L3 C4 R5 L6;

Кроме этого, формируем компоненты-структуры исходной (внешней) активной Pu-схемы [2] [для структуры-источника сохраним имя Ist с указанием узлов; приемник как виртуальная схемная проводимость с номером 1 в массиве MaProSx вместо Pri получит имя MaProSx(1)]:

Ist.Uz1=[1]; (меньший узел источника)
Ist.Uz2=[3]; (больший узел источника)
MaProSx(1).Nom=[1]; (номер в схеме для приемника)
MaProSx(1).Ves=[1]; (вес проводимости приемника)
MaProSx(1).Uz1=[2]; (меньший узел приемника)
MaProSx(1).Uz2=[4]; (больший узел приемника)
MaProSx(1).NeUz=[]; (условный неприсоединенный – пока пустой – узел приемника, к которому можно подключить другой элемент схемы)

Аналогично инициализируются поля для проводимостей MaProSx(2)-MaProSx(6) с учетом их узлов связи и операторных символических весов 1/R2, 1/(s*L3), s*C4, 1/R5, 1/(s*L6).

Для работы рассматриваемых модулей FoUnMoDoPu и PoiskNDoPu необходимо задать какие-либо пути Pu (так как указанные модули находят только дополнения этих путей) между узлами NaUz, KoUz схемы. Для этого проще всего использовать уже известные программы PoiskNaElPu, PoiskPrNzPu и PoiskPuIstPri (поиск путей Pu от источника Ist к приемнику Pri) [2, 3], в совокупности определяющие пути PoPu и OtPu, начинающиеся в положительном узле Ist.Uz1 источника и заканчивающиеся в его отрицательном узле Ist.Uz2, что соответствует очевидным операторам присваивания:

NaUz=Ist.Uz1;
KoUz=Ist.Uz2;

Первые элементы любых путей между узлами NaUz и KoUz, как было сказано ранее, находятся процедурой PoiskNaElPu [2], которую можно вызвать из окна Command Window:

[MaNzPu,MaVeNzPu,MaMaUzNzPu,...
MaZaPu,MaVeZaPu,MaMaUzZaPu]=...
PoiskNaElPu(MaProSx,NaUz,KoUz)

Результаты работы PoiskNaElPu (её выходные аргументы) для схемы рисунка 1 в сокращенной форме имеют следующий вид:

MaNzPu – 1x2 cell (массив двух ячеек cell – два незавершенных пути NzPu) – [1x1 struct], [1x1 struct] (ячейки-пути содержат данные типа struct – проводимости 2 и 5);

MaZaPu – 0x0 cell (массив MaZaPu пустой, завершенных путей ZaPu нет).

Поскольку найденный массив MaNzPu – не

пустой, из Command Window вызываем процедуру PoiskPrNzPu [3] поиска продолжений Pr незавершенных путей NzPu –

[MaNzPu,MaVeNzPu,MaMaUzNzPu,...
MaZaPu,MaVeZaPu,MaMaUzZaPu]=...
PoiskPrNzPu(MaNzPu,MaVeNzPu,...
MaMaUzNzPu,MaZaPu,MaVeZaPu,...
MaMaUzZaPu,MaProSx,KoUz)

Наиболее значимые выходные аргументы модуля PoiskPrNzPu после его выполнения даны ниже:

MaNzPu – 0x0 cell (пустой массив – завершены все пути NzPu);

MaZaPu – 1x6 cell – [1x2 struct], [1x2 struct], [1x3 struct], [1x3 struct], [1x3 struct], [1x3 struct] (массив из шести ячеек cell с данными типа struct; каждая ячейка содержит элементы путей ZaPu: 2-6, 5-3, 2-1-2, 2-4-3, 5-1-6, 5-4-6).

Для извлечения из массива MaZaPu сигнальных путей PoPu и OtPu в окне Command Window задаем фактический аргумент Uz2Pri процедуры PoiskPuIstPri –

Uz2Pri=MaProSx(1).Uz2;

и вызываем сам модуль PoiskPuIstPri –

[MaPoPu,MaVePoPu,MaMaUzPoPu,...
MaOtPu,MaVeOtPu,MaMaUzOtPu]=...
PoiskPuIstPri(MaZaPu,MaVeZaPu,...
MaMaUzZaPu,Uz2Pri)

Ниже представлены некоторые результирующие выходные аргументы функции PoiskPuIstPri для рисунка 1:

MaPoPu – 1x1 cell (одна ячейка – один положительный путь PoPu – 2-1-3);

MaOtPu – 1x1 cell (один отрицательный путь 5-1-6).

Массивы MaPoPu, MaOtPu – не пустые, поэтому можно определять дополнения путей PoPu, OtPu (DoPoPu и DoOtPu). Например, для всех путей PoPu с номерами nPoPu –

for nPoPu=1:numel(MaPoPu)

сначала инициализируем фактические входные аргументы функции FoUnMoDoPu:

– путь Pu –

Pu=MaPoPu{nPoPu};

– номер первого (внешнего) итерационного Do-этапа расчетов дополнения DoPoPu пути Pu=PoPu –

nIt=1;

– массив узлов пути Pu –

MaUzPu=MaMaUzPoPu{nPoPu};

Поскольку каждому из путей PoPu соответствует свое дополнение DoPoPu, для поиска нужного элемента массива MaDoPoPu (дополнения всех путей в MaPoPu) используем номер nPoPu –

MaDoPoPu(nPoPu)=FoUnMoDoPu...
(nIt,Pu,MaUzPu,MaProSx,NaUz);
end (конец оператора
for nPoPu=1:numel(MaPoPu)

Аналогично задаем команды для поиска дополнений DoOtPu путей OtPu:

for nOtPu=1:numel(MaOtPu)
Pu=MaOtPu{nOtPu};
nIt=1;
MaUzPu=MaMaUzOtPu{nOtPu};
MaDoOtPu(nOtPu)= FoUnMoDoPu...
(nIt,Pu,MaUzPu,MaProSx,NaUz);
end (конец оператора
for nOtPu=1:numel(MaOtPu))

В итоге для определенных выше (с помощью процедуры PoiskPuIstPri) единственных путей в MaPoPu (проводимости 2-1-3) и MaOtPu (проводимости 5-1-6) Pu-схемы рисунка 1 получаем их дополнения с равными значениями –
MaDoPoPu(1)=MaDoOtPu(1)=1.

Последнее очевидно из визуального топологического анализа рисунка 1, так как при замыкании любого из путей 2-1-3 (MaPoPu) и 5-1-6 (MaOtPu) и упрощения получаемых IDo-схем остается одна и та же UDo-схема без проводимостей, когда дополнения Do [в данном случае DoPoPu=MaDoPoPu (1) и DoOtPu=MaDoOtPu (1)] равны 1.

Усложним рисунок 1 индуктивными компонентами 7, 8 и 9 – рисунок 2 [2].

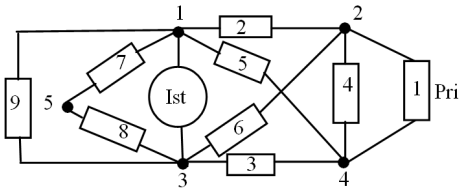

Рисунок 2 – Усложненная схема рисунка 1

Для него в функцию syms необходимо ввести дополнительные фактические аргументы L7, L8 и L9, а в массив MaProSx – соответствующие структуры MaProSx(7)-MaProSx(9) [2]. После вызова и исполнения процедур PoiskNaElPu, PoiskPrNzPu, PoiskPuIstPri и FoUnMoDoPu для рисунка 2 имеем такие результаты:

MaPoPu – 1x1 cell (путь 2-1-3);
MaOtPu – 1x1 cell (путь 5-1-6);
MaDoPoPu(1)=MaDoOtPu(1)=1/(L7*s)+1/(L8*s).

Первые два выражения для MaPoPu и MaOtPu ожидаемы, поскольку введенные элементы 7-9 не передают сигналы от источника к приемнику, т.е. не создают дополнительных (по сравнению с рисунком 1) путей PoPu, OtPu. Последнее выражение тривиально следует из рисунка 2, так как при замыкании пути 2-1-3 (MaPoPu) или 5-1-6 (MaOtPu) и изъятия замкнутых проводимостей остаются однотипные двухэлементные UDo-схемы (т.е. производные IPu-

схемы). После объединения соединенных узлов в IPu-схемах получаем две UPu-схемы в виде параллельных проводимостей 7 (с весом 1/(L7*s)) и 8 (с весом 1/(L8*s)). Из каждой такой UPu-схемы после замыкания соответствующих проводимостей 7 или 8 формируются пустые Do-схемы с Do=1, сумма произведений которых на веса 1/(L7*s) и 1/(L8*s) дают полученный выше результат MaDoPoPu(1) = MaDoOtPu(1) = =1/(L7*s)+1/(L8*s).

Дополним рисунок 2 резистивными элементами 10 и 11– рисунок 3.

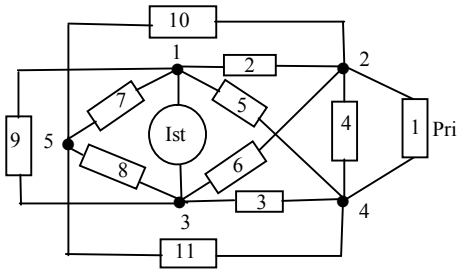

Рисунок 3 – Рисунок 2 с дополнительными проводимостями 10 и 11

Для рисунка 3 к предыдущему выражению syms рисунка 2 необходимо добавить переменные R10 и R11, а к MaProSx – структуры MaProSx(10) и MaProSx(11).

Автоматизированный поиск путей PoPu, OtPu и их дополнений для рисунка 3 по аналогии с вышеизложенным дает:

MaPoPu – 1x3 cell (пути 2-1-3, 2-1-11-8, 7-10-1-3);

MaOtPu – 1x3 cell (пути 5-1-6, 5-1-10-8, 7-11-1-6);

MaDoPoPu(1)=MaDoOtPu(1)=1/R10 +1/R11+ +1/(L7*s) +1/(L8*s);

MaDoPoPu(2)=MaDoPoPu(3)=MaDoOtPu(2)= =MaDoOtPu(3) = 1.

Все последние выражения подтверждаются визуальным топологическим анализом [4] Pu-схемы рисунка 3. В-частности, после замыкания на рисунке 3 любого из путей 2-1-3 (MaPoPu(1)) и 5-1-6 (MaOtPu(1)) остаются параллельно включенные проводимости 10, 11, 7, 8, а при замыкании путей 2-1-11-8 (MaPoPu(2)), 7-10-1-3 (MaPoPu(3)), 5-1-10-8 (MaOtPu(2)) и 7-11-1-6 (MaOtPu(3)) – пустые Do-схемы (т.е. MaDoPoPu(2) = MaDoPoPu(3) = MaDoOtPu(2) = =MaDoOtPu(3) = 1); замыкания в двух производных Pu-схемах (для путей MaPoPu(1) и MaOtPu(1)) любых проводимостей 7, 8, 10, 11 образуют пустые Do-схемы с Do=1, произведения которых на веса 1/R10, 1/R11, 1/(L7*s) и 1/(L8*s) дают предпоследний результат MaDoPoPu(1) = MaDoOtPu(1) = 1/R10 + 1/R11 + +1/(L7*s)+1/(L8*s).

Выводы. Предложенные методика и программа автоматизированного поиска дополнений завершенных топологических путей между любыми двумя узлами LCR-схем произвольной сложности основаны на их традиционном изображении и не требуют от пользователя каких-либо трудоемких ручных математических преобразований.

Научная новизна методики и программы поиска дополнений топологических путей состоит в следующем:

– разработан трафаретный алгоритм (PoiskNaElPu) для поиска дополнений, что позволило сократить количество требуемых программных модулей;

– предложен циклический оператор (Do = =Do+Slag2) для вызова процедур поиска дополнений всех путей;

– создан способ уничтожения использованных программных модулей, позволяющий уменьшить объем требуемой памяти вычислительной техники.

Областью применения изложенного материала является исследование LRC-аналогов любой физической природы, в которых соотношения между сигналами эквивалентны таковым для традиционных LRC-структур [1].

Примером прикладного использования разработанных программных модулей FoUnMoDoPu и PoiskNDoPu является поиск LRC-передаточных функций блоков Transfer Fcn для исследования Simulink-моделей [6].

Библиографический список

1. *Одиноков В.Ф.* Топологическое моделирование линейных цепей информационно-измерительных систем // Вестник Рязанского государственного радиотехнического университета. 2013. № 4-2 (46). С. 33-40.

2. *Одиноков В.Ф.* Автоматизированный поиск начальных элементов путей связи в LRC-схемах // Вестник Рязанского государственного радиотехнического университета. 2014. № 50-2. С. 51-57.

3. *Одиноков В.Ф.* Автоматизированный поиск путей связи между двумя узлами в LRC-схемах // Вестник Рязанского государственного радиотехнического университета. 2015. № 52. С.130-138.

4. *Мэзон С. и Циммерман Г.* Электронные цепи, сигналы и системы / пер. с англ. А. А. Соколова и И. В. Соловьева; под ред. проф. П. А. Ионкина. – М.: Издательство иностранной литературы, 1963.

5. *Дьяконов В. П.* MATLAB.*/R20006/R2007: самоучитель. – М.: ДМК Пресс, 2008.

6. *Дьяконов В. П.* Simulink 5/6/7: самоучитель. – М.: ДМК-Пресс, 2008.

7. *Евдокимов Ф.Е.* Теоретические основы электротехники: учеб. для студ. образоват. учреждений сред. проф. образования / Ф. Е. Евдокимов. – 9-е изд. стереотип. – М.: Издательский центр "Академия", 2004.