Инженерный вестник Дона, №4 (2020)
ivdon.ru/ru/magazine/archive/n4y2020/6435

Автоматизация расчета передаточных функций АСУ методом некасающихся контуров

Б.К. Набиулин, И.М. Сафаров, Г.М. Сафиуллина

Казанский государственный энергетический университет

Аннотация: При представлении современных систем автоматического управления в виде структурных схем они становятся громоздкими и нечитаемыми, а их исследование требует большого количества сложных для человека расчетов. К тому же, многие студенты, обучающиеся по направлению «Автоматизация технологических процессов и производств» не понимают современных методов расчета передаточных функций систем управления. В данной статье рассмотрен способ представления структурных схем в виде ориентированных графов, что сильно упрощает читабельность схем, а также дает возможность написания алгоритмов обработки систем автоматического управления, представленных в виде графов. В статье предложены алгоритмы для нахождения передаточных функций, написанные на языке программирования C#, которые могут стать базой для собственной программы для исследования САУ, а также студентами для проверки корректности найденной передаточной функции.

Ключевые слова: граф, орграф, теория автоматического управления, система автоматического управления, передаточная функция.

Для исследования систем автоматического управления в ТАУ их принято схематично изображать в виде структурных схем, на которых располагают элементы, выполняющие функции, необходимые для осуществления процесса автоматического управления [1-2].

Однако, на практике схемы становятся слишком громоздкими и сложными для восприятия. К тому же, исследование САУ требует большого количества сложных для человека расчетов.

Использование вычислительных ресурсов компьютера упрощает задачу исследования систем автоматического управления [3]. Однако, структурное описание схемы, как типа данных для ЭВМ и алгоритмов работы с ними, является трудоемким процессом.

Представление схем в виде ориентированных графов решает все вышепоставленные проблемы:

1) Делает схему проще и читабельнее;

2) Орграф легко представить как тип данных;

3) Существует множество алгоритмов для работы с графами [4].

Задачей текущей работы является автоматизирование расчета передаточных функций путем создания программы с использованием правила некасающихся контуров, которое выражается следующей формулой:

$$W(s) = Y(s)/X(s) = \frac{\sum_{i=1}^{n} W_{\text{пр } i}(s)\Delta_i(s)}{\Delta s} \quad (1)$$

где Wпрi - передаточная функция i-го отдельного прямого пути от X(s) до Y(s), вычисленная как произведение передаточных функций дуг, входящих в этот путь;

Δs - определитель орграфа [5-6].

$$\Delta s = 1 - \sum_j W_j(s) + \sum_{jk} W_j(s)W_k(s) + \ldots \quad (2)$$

Wj(s) - передаточная функция j-го замкнутого контура, вычисленная как произведение передаточных функций дуг, входящих в этот контур;

Wj(s)Wk(s) - произведение передаточных функций пары (j-го и k-го) замкнутых контуров, не касающихся ни дугами, ни вершинами, суммирование осуществляется по всем парам некасающихся контуров;

$\Delta_i(s)$- определитель орграфа, полученного при удалении дуг и вершин i-го отдельного прямого пути, определяется по формуле (2) [7].

Описанная далее программа может быть использована студентами, изучающими теорию автоматического управления, для глубокого понимания методов расчета передаточных функций АСУ, для проверки правильности решениях задач по курсу «Теория автоматического управления», а также может стать основой для полноценной программы расчета передаточных функций для использования на предприятиях.

Для достижения этой цели нужно выполнить следующие задачи:

1) Представим ориентированный граф в удобном для обработки компьютером виде

2) Опишем алгоритм поиска прямых путей графа

3) Опишем алгоритм поиска замкнутых контуров графа.

Для работы с графом представляем его в удобном для обработки виде и реализуем три класса:

1) Класс для хранения пройденных вершин

2) Класс для хранения стека пути

3) Класс для хранения вершин, которые мы можем посетить из текущей, но еще не посетили [8].

Реализация всех классов и методов представленных в статье осуществлялась на языке программирования c# в среде разработки Visual Studio.

Для работы с графом реализуем класс Edge, представляющий его вершину, а также класс Neighbor для хранения списка соседних вершин и значения передаточной функции между ними. Пример реализации метода Edge представлен на рисунке 5 [9].

```
class Edge
{
    private List<Neighbor> AllNeaigbors = new List<Neighbor>();
    public Guid Id { get; private set; }

    public Edge() { Id = Guid.NewGuid(); }

    public List<Edge> GetNeighbors()
    {
        return AllNeaigbors
            .Select(neighbor => neighbor.EdgeNeighbor).ToList();
    }


    public void AddNeighbor(Edge  link, string Function)
    {
        var neighbor = AllNeaigbors
            .Where(n => n.EdgeNeighbor.Id == link.Id).Select(n => n);

        if (!neighbor.Any())
            AllNeaigbors.Add(new Neighbor(link, Function));
        else
            neighbor.First().Functions.Add(Function);
    }
    public List<string> GetFunctionsBetweenNeaighbor(Edge neaighbor)
    {
        return AllNeaigbors
            .Where(n => n.EdgeNeighbor.Id == neaighbor.Id)
            .SelectMany(n => n.Functions).ToList();
    }
}
```

Рис 5. Реализация класса Edge

В текущей реализации класс Edge каждой вершине класс создается уникальный идентификатор, а также реализован метод для добавления соседних вершин графа и назначения им передаточной функции, метод для получения всех соседних вершин и метод для получения значений передаточных функций между текущей и переданной на вход вершиной.

Для представления соседей реализован класс Neighbor, который хранит в себе вершину соседа и значения передаточной функции между текущей вершиной и ее соседом. Пример его реализации представлен на рисунке 6.

```
class Neighbor
{
    public List<string> Functions { get; private set; }
    public Edge EdgeNeighbor { get; private set; }
    public Neighbor(Edge Link, string function)
    {
        EdgeNeighbor = Link;
        Functions = new List<string> { function };
    }
}
```

Рис 6. Реализация класса Neighbor

Первый класс History, необходимый для работы с графом, имеет поле для хранения пройденных вершин графа и методов, с помощью которых можно проверять была ли посещена текущая вершина, а также менять состояние посещения на противоположное. Реализация данного класса может иметь вид изображенныйна рис. 7.

В текущей реализации класс History имеет публичное поле VisitedEdges, представляющее собой словарь, ключи которого – вершины графа, а значение является булевой величиной, показывающей, была ли посещена текущая вершина (true – посещена, false – не посещена).

Конструктор представленного класс принимает на вход сам граф, представленный перечислением вершин, преобразует его в словарь и запоминает в поле VisitedEdges, устанавливая всем вершинам значение false, так как ни одна вершина еще не была посещена.

```
class History
{
    public Dictionary <Edge, bool> VisitedEdges =
            new Dictionary<Edge, bool>();

    public History (IEnumerable<Edge> Graph)
    {
        VisitedEdges = Graph.ToDictionary(
                edge => edge, edge => false );
    }
    public bool IsVisited(Edge edge)
    {
        return VisitedEdges[edge];
    }
    public void SetVisited(Edge edge)
    {
        VisitedEdges[edge] = true;
    }
    public void SetUnvisited(Edge edge)
    {
        VisitedEdges [edge] = false;
    }
}
```

Рис 7. Реализация класса History

Внутри класса реализованы три вспомогательных метода IsVisited, VisitedEdges, SetUnvisited, каждый из которых принимает на вход вершину. Первый метод возвращает для текущей вершины булевое значение, которое показывает была ли она посещена, второй устанавливает вершине значение того, что она была посещена, а третий то, что вершина не была посещена.

Второй класс содержит поле для хранения стека пройденного пути, а также методы для получения текущего стека, добавления в него новой вершины, удаления из него вершины и тд. Пример реализация такого класса представлен на рисунке 8.

```
class StackWay
{
    public List<Edge> CurrentWayInfo = new List<Edge>();
    public void AddToWay(Edge edge)
    {
        CurrentWayInfo.Add(edge);
    }
    public Edge GetAndRemoveLastEdge()
    {
        var lastEdge = CurrentWayInfo[CurrentWayInfo.Count - 1];
        CurrentWayInfo.RemoveAt(CurrentWayInfo.Count - 1);
        return lastEdge;
    }
    public Edge GetLastEdge() { return CurrentWayInfo.Last(); }
    public List<Edge> GetWay() { return CurrentWayInfo; }
    public int GetNumberByElement(Edge edge)
        { return CurrentWayInfo.LastIndexOf(edge); }
    public Edge GetElementByNumber(int Number)
        { return CurrentWayInfo[Number]; }
    public bool IsEmpty() { return CurrentWayInfo.Count == 0; }
}
```

Рис 8. Реализация класса StackWay

Класс StackWay в текущей реализации содержит список вершин CurrentWayInfo, который служит для хранения самого стека пути, а также несколько вспомогательных методов для добавления новой вершины в стек, для удаления вершины из стека, получения последней вершины, индекса вершины и получения вершины по индексу, проверки стека на пустоту.

Третий класс служит для того, чтобы на этапе обработки текущей вершины хранить информацию о том, из какой вершины мы попали в текущую, и какие можем посетить, но еще не посещали. Класс должен содержать словарь, ключами которого являются вершины, а значения – списка вершин, которые можно посетить из текущей.

```
class ChildDependecy
{
    Dictionary<Edge, HashSet<Edge>> childDependecy =
        new Dictionary<Edge, HashSet<Edge>>();
    public ChildDependecy (IEnumerable<Edge> Graph)
    {
        childDependecy = Graph.ToDictionary(
            edge => edge, edge => new HashSet<Edge>());
    }
    public List<Edge> GetAllGrandchildren(Edge edge)
        { return childDependecy[edge].ToList<Edge>(); }
    public void AddDependence(Edge Root, Edge grandChild)
        { childDependecy[Root].Add(grandChild); }
    public void SetGrandChildUnvisitedForEdge(
        Edge edge, History history)
    {
        GetAllGrandchildren(edge).ForEach(child => history.SetUnvisited(child));
    }
}
```

Рис 9. Реализация класса ChildDependecy

Представленный класс ChildDependecy в текущей реализации имеет публичное поле childDependecy, которое является словарем и в качестве ключа хранит вершину, а в качестве значения – список ее вершин-детей. Конструктор такого класса принимает на вход в качестве аргумента весь граф, предоставленный как перечисление вершин, превращает его в словарь, ключи которого вершины, а значения – пустые списки, и запоминает его в поле childDependecy.

В текущем классе реализованы вспомогательные методы для получения всех зависимостей текущей вершины, для добавления к вершине новых зависимостей и для установки всем зависимостям вершины состояние «непосещенности».

Реализуем метод поиска всех простых путей графа между двумя заданными вершинами, который принимает на вход два параметра:

начальную и конечную вершины, а возвращает список путей. Реализация метода может иметь следующий вид:

```
public List<List<Edge>> FindWay(Edge X, Edge Y)
{
    var result = new List<List<Edge>> ();
    var history = new History(Graph);
    var stack = new StackWay();
    var childDependecy = new ChildDependecy (Graph);
    stack.AddToWay(X);
    while (stack.IsEmpty() != true)
    {
        var Current = stack.GetLastEdge ();
        SetGrandchildDependency(
            stack, childDependecy, Current);
        if (Current == Y)
        {
            result.Add(new List<Edge> (stack.GetWay()));
            history.SetVisited(Y);
            stack.GetAndRemoveLastEdge();
        }
        else
        {
    childDependecy
        .SetGrandChildUnvisitedForEdge(Current, history);
    var NeighborsNotVisited =
        FilterNeighbors(stack, history, Current);
            if (NeighborsNotVisited.Count > 0)
                stack.AddToWay(NeighborsNotVisited[0]);
            else
                stack.GetAndRemoveLastEdge();
            history.SetVisited(Current);
        }
    }
    return result;
}
```

Рис 10. Реализация метода FindWay

Метод FindWay возвращает все наборы вершин от переданной ему вершины X до вершины У. В его реализации был также использован

вспомогательный метод FilterNeighbors, который позволяет получить список непосещенных вершин-соседей переданной в него вершины.

```
public List<Edge> FilterNeighbors(
    StackWay way, History history, Edge edge)
{
    var Neighbors = edge.GetNeighbors();
    var NeighborsWithoutStack = new List<Edge>();
    foreach (var t in Neighbors)
    {
        if (way.GetNumberByElement(t) == -1)
            NeighborsWithoutStack.Add(t);
    }
    var NeighborsNotVisited = new List<Edge>();
    foreach (var t in NeighborsWithoutStack)
    {
        if (history.IsVisited(t) == false)
            NeighborsNotVisited.Add(t);
    }
    return NeighborsNotVisited;
}
```

Рис 11. Реализация метода FilterNeighbors

Замкнутые контуры являются также простыми путями, где начальная и конечная вершины совпадают, поэтому для их поиска можно также воспользоваться методом FindWay(Edge X, Edge Y). Однако его следует немного модифицировать, чтобы не вылетать из метода при первой итерации. Для этого достаточно добавить булевую переменную, благодаря которой будет определяться первая итерация для пропуска проверки if (Current == Y), а в конце ее значение будет меняться на противоположное. А если передавать в метод FindWay вместо самой вершины ее соседей и в получившиеся наборы вершин в начало вставить нужную нам вершину, то мы получим все замкнутые в нее пути, не модернизируя метод.

После нахождения всех простых путей и замкнутых контуров достаточно подставить получившиеся значения передаточных функций в формулу 1 для нахождения передаточной функции всей системы. Для ее вычисление следует реализовать отдельный метод.

Из полученного набора вершин методом FindWay можно найти передаточные функции, воспользовавшись методом GetFunctionsBetweenNeaighbor, реализованном в классе Edge. Однако, учитывая то, что между двумя одинаковыми вершинами могут быть более одного пути, нужно посчитать декартово произведение списков передаточных функций между вершинами. Реализация такого участка кода представлен на рисунке 12, где метод CartesianProducts реализует декартово произведение.

```
foreach(var way in result)
{
    var functionOfDirectWays =
        new List<List<string>>();
    for (int i = 0; i < way.Count() - 1; i++)
    {
        functionOfDirectWays
            .Add(way[i].GetFunctionsBetweenNeaighbor(way[i + 1]));
    }
    functions.Add( CartesianProducts(functionOfDirectWays));
}
public List<string> CartesianProducts(
    List<List<string>> functionOfDirectWays)
{
    var cartesianProducts = functionOfDirectWays.First();
    for(int i = 1; i < functionOfDirectWays.Count(); i++)
    {
        cartesianProducts = (from pr in cartesianProducts
                             from fun in functionOfDirectWays[i]
                             select pr + fun).ToList();
    }
    return cartesianProducts;
}
```

Рис 12. Поиск передаточных функций и реализация метода CartesianProducts

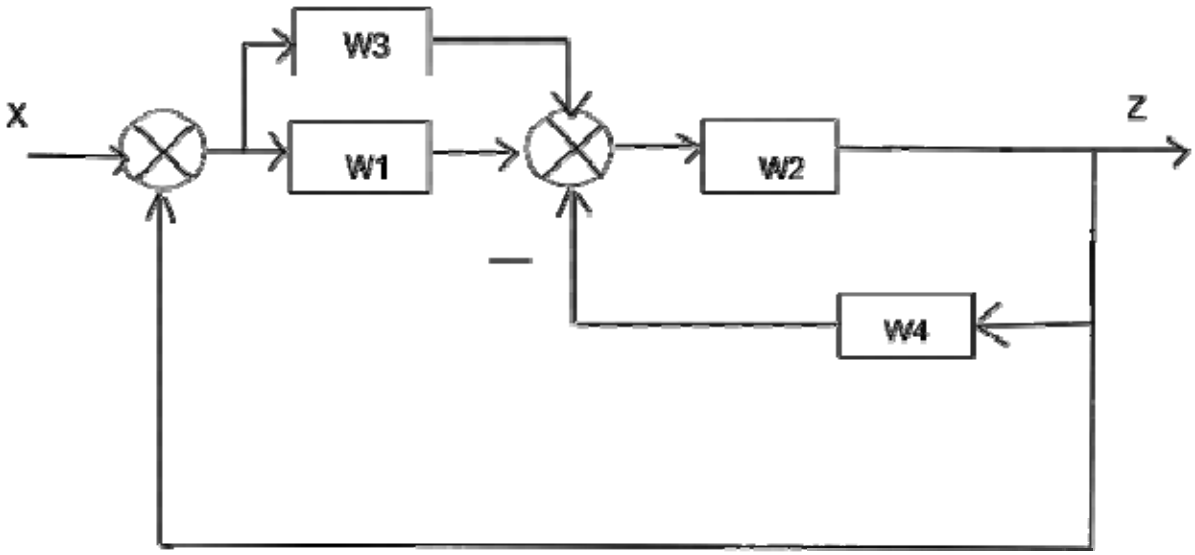

Рис 13. Пример структурной схемы САУ

```
var e = new Edge();
var u = new Edge();
var z = new Edge();

e.AddNeighbor(u, "w3");
e.AddNeighbor(u, "w1");
u.AddNeighbor(z, "w2");
z.AddNeighbor(u, "w4");
z.AddNeighbor(e, "w5");
var Graph = new List<Edge> {  e, u, z };
```

Рис 14. Инициализация структурной схемы

На рисунке 14 представлена инициализация вершин и самого графа для структурной схемы САУ, изображенной выше. При использовании на них методов, представленных в текущей работе, были получены передаточные функции прямых путей в общем виде (рис 15). Для поиска замкнутых путей

использовался также метод FindWay, в который передавались по очереди вершины в качестве первого аргумента, а вторым аргументом ее соседи, после чего в каждый полученный список вершин добавлялась в начало вершина, переданная первым аргументом (рис 16).

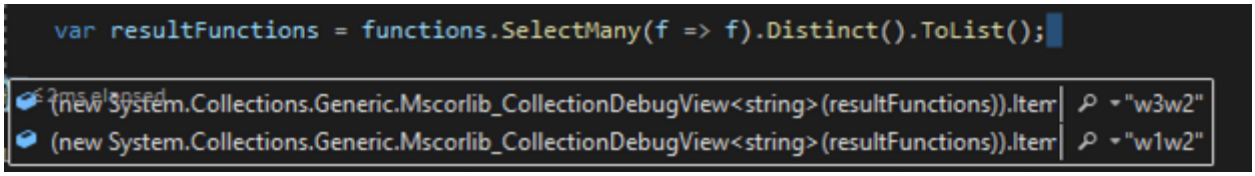

Рис 15. Результат поиска передаточных функций прямых путей

Рис 16. Результат поиска передаточных функций замкнутых путей

Таким образом можно рассчитать передаточные функции для любой схемы АСУ, потратив при этом минимум времени и усилий, что позволяет студентам проверять свои расчеты. Программа может использоваться для упрощения расчетов в научных работах и стать основой полноценной программы для использования на предприятиях, что позволит разгрузить высококвалифицированных инженеров от расчетов и нагрузить менее квалифицированных, так как использование программы требует минимум знаний [10].

Литература

1. Лебедев С.К. Математические основы теории автоматического управления. Иваново: ИЭИ, 1989. 149с.

2. Целигорова Е.Н. Современные информационные технологии и их использование для исследования систем автоматического управления// Инженерный вестник Дона, 2010, №3 URL: ivdon.ru/magazine/archive/n3y2013/1907.

3. Гинис Л.А. Развитие инструментария когнитивного моделирования для исследования сложных систем// Инженерный вестник Дона, 2013, №3 URL: ivdon.ru/ru/magazine/archive/n3y2013/1806.

4. Бесекерский В.А., Попов Е.П. Теория систем автоматического регулирования. М.: Наука, 1972. 768 с.

5. Kuo, Benjamin C. Automatic Control Systems. 2nd Edition. New Jersey: Prentice-Hall, 1967. 653 p.

6. Bolton, W. Newnes. Control Engineering Pocketbook. Oxford: Newnes, 1998. 304 p.

7. Ерофеев А.А. Теория автоматического управления. СПБ.: Политехника, 2013. 302 с.

8. Седжвик Р. Алгоритмы на С++. Фундаментальные алгоритмы и структуры данных. Киев: ДиаСофт, 2001. 1001 с.

9. Скиена С. Алгоритмы. Руководство по разработке. 2-е изд. СПБ.: БХВ – Петербург, 2011. 720 с.

10. Сафаров И.М., Хаматханов Д.И., Калимуллин А.А. Автоматизированная система управления параметрами теплоносителя с удаленным доступом// Инженерный вестник Дона, 2018, №2 URL: ivdon.ru/ru/magazine/archive/n2y2018/4912.

References

1. Lebedev S.K. Matematicheskie osnovy` teorii avtomaticheskogo upravleniya [Mathematical foundations of automatic control theory]. Ivanovo: IEI, 1989. 149p.

2. Celigorova E.H. Inzhenernyj vestnik Dona, 2010, №3 URL: ivdon.ru/magazine/archive/n3y2013/1907.

3. Ginis L.A. Inzhenernyj vestnik Dona, 2013, №3 URL: ivdon.ru/ru/magazine/archive/n3y2013/1806.

4. Besekerskij V.A., Popov E.P. Teoriya sistem avtomaticheskogo regulirovaniya [Theory of automatic control systems]. M.: Nauka, 1972. 768p.

5. Kuo, Benjamin C. Automatic Control Systems. 2nd Edition. New Jersey: Prentice-Hall, 1967. 653 p.

6. Bolton, W. Newnes. Control Engineering Pocketbook. Oxford: Newnes, 1998. 304 p.

7. Erofeev A.A. Teoriya avtomaticheskogo upravleniya. [Theory of Automatic Control]. SPB.: Politechnika, 2013. 302 p.

8. Sedzhvik R. Algoritmy` na S++. Fundamental`ny`e algoritmy` i struktury` danny`x [Algorithms in C++. Fundamental algorithms and data structures]. Kiev: Diasoft, 2001. 1001 p.

9. Skiena S. Algoritmy`. Rukovodstvo po razrabotke [Algorithms. Development guide]. vol 2. SPB.: BHV-Petersburg, 2011. 720 p.

10. Safarov I.M., Xamatxanov D.I., Kalimullin A.A. Inzhenernyj vestnik Dona, 2018, №2 URL: ivdon.ru/magazine/archive/n2y2018/4912.