KARADENİZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

PADE YAKLAŞIMININ SİMGESEL MATEMATİK YÖNTEMLERİYLE UYGULANMASI

YÜKSEK LİSANS TEZİ

Esra ERDEN

EKİM 2024
TRABZON

KARADENİZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

PADE YAKLAŞIMININ SİMGESEL MATEMATİK YÖNTEMLERİYLE UYGULANMASI

Esra ERDEN

Karadeniz Teknik Üniversitesi Fen Bilimleri Enstitüsünce
"BİLGİSAYAR YÜKSEK MÜHENDİSİ
Unvanı Verilmesi İçin Kabul Edilen Tezdir.

Tezin Enstitüye Verildiği Tarih : 31.10.2024
Tezin Savunma Tarihi : 02.10.2024

Tez Danışmanı : Doç. Dr. Hüseyin PEHLİVAN

Trabzon 2024

ÖNSÖZ

"Pade Yaklaşımının Simgesel Matematik Yöntemleriyle Uygulanması" isimli bu tez Karadeniz Teknik Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı Yüksek Lisans Programı'nda hazırlanmıştır.

Tez çalışmam boyunca yardım, deneyim, tecrübe ve hiçbir desteğini esirgemeyerek kıymetli bilgileriyle yolumu aydınlatan değerli danışman hocam Sayın Doç. Dr. Hüseyin PEHLİVAN'a,

Hayatımın her aşamasında, her kararımda arkamda olan, maddi ve manevi desteklerini hiçbir zaman esirgemeyip bugünlere gelmemde en büyük paya sahip olan anneme, babama ve kardeşime,

Yüksek lisans eğitimimi bitirmemde her zaman destekçim olup, sonuna kadar yanımda duran Melike YÜKSEL'e ve yakın arkadaşlarıma,

Minnetlerimi ve sonsuz teşekkürlerimi sunarım.

Bu tez çalışmasının bundan sonraki çalışmalara katkı sağlamasını temenni ederim.

Esra ERDEN

Trabzon 2024

TEZ ETİK BEYANNAMESİ

Yüksek Lisans Tezi olarak sunduğum "Pade Yaklaşımının Simgesel Matematik Yöntemleriyle Uygulanması" başlıklı bu çalışmayı baştan sona kadar danışmanım Doç. Dr. Hüseyin PEHLİVAN'ın sorumluluğunda tamamladığımı, verileri/örnekleri kendim topladığımı, deneyleri/analizleri ilgili laboratuvarlarda yaptığımı/yaptırdığımı, başka kaynaklardan aldığım bilgileri metinde ve kaynakçada eksiksiz olarak gösterdiğimi, çalışma sürecinde bilimsel araştırma ve etik kurallara uygun olarak davrandığımı ve aksinin ortaya çıkması durumunda her türlü yasal sonucu kabul ettiğimi beyan ederim. 02/10/2024

Esra ERDEN

İÇİNDEKİLER

Sayfa No

Yüksek Lisans Tezi

ÖZET

PADE YAKLAŞIMININ
SİMGESEL MATEMATİK YÖNTEMLERİYLE UYGULANMASI

Esra ERDEN

Karadeniz Teknik Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı
Danışman: Doç. Dr. Hüseyin PEHLİVAN
2024, 70 Sayfa

Bu çalışmada, Pade yaklaşımı simgesel matematik yöntemleriyle birlikte kullanılarak rasyonel polinom ifadelerinin belirlenmesi ve hesaplama adımlarının bir PDF dokümanına aktarımını yapabilen bir yazılım uygulamasının geliştirilmesi amaçlanmıştır. Polinom ifadelerini temsil edebilmek için EBNF benzeri notasyonuna sahip bir gramer tasarlanmıştır. JavaCC ile tanımlanan ayrıştırıcılar yardımıyla polinom ifadelerinin taşıdıkları anlamlara göre soyut sözdizim ağaçları oluşturulmuştur. Değerlendirme aşamasında ise farklı işlem türlerini tekil bir sınıf altında toplama olanağı sağlayan Ziyaretçi tasarım deseninden yararlanılmıştır. Belirli bir matematiksel fonksiyonu temsil edecek rasyonel polinomları belirlemek için gerçekleştirilen hesaplamalar, işlem adımlarını içerecek biçimde hem bir terminal üzerinde gösterilebilmekte hem de bir PDF dokümanına aktarılabilmektedir. PDF dosyalarının üretimi, polinom ifadelerine ait sözdizim ağaçları üzerinden LaTex doküman formatlama dilinde yapılacak metinsel dönüşümlerle gerçekleştirilmiştir.

Anahtar Kelimeler: Pade yaklaşımı, Biçimsel gramerler, Ayrıştırıcılar, Simgesel hesaplama, Doküman formatlama.

Master Thesis

SUMMARY

PADE APPROXIMATION APPLICATION WITH SYMBOLICAL MATHEMATICS METHODS

Esra ERDEN

Karadeniz Technical University
The Graduate School of Natural and Applied Sciences
Computer Engineering Graduate Program
Supervisor: Asst. Prof. Dr. Hüseyin PEHLİVAN
2024, 70 Pages

The aim of this study is to develop a software application that can compute rational polynomial expressions by using the Pade approach in conjunction with symbolic mathematics methods, and export the computation steps to a PDF document. A grammar with EBNF-like notation is designed to represent polynomial expressions. JavaCC parsers is used to create abstract syntax trees based on the meanings of polynomial expressions. The Visitor design pattern is employed in the evaluation process, which allows collecting different types of operations contained in the syntax tree under a single class. The calculations involved in determining rational polynomials that should be used to describe a certain mathematical function can be displayed on a terminal and transferred to a PDF document, along with the process steps. The production of PDF files is carried out by textual transformations into the LaTex document formatting language via syntax trees of polynomial expressions.

Key Words: Pade approximation, Formal grammars, Parsers, Symbolic computation, Document formatting

ŞEKİLLER DİZİNİ

Sayfa No

TABLOLAR DİZİNİ

Sayfa No

SEMBOLLER VE KISALTMALAR DİZİNİ

GAMS	: General Algebraic Modeling System
AMPL	: A Mathematical Programming Language
CFG	: Context Free Grammer
BNF	: Backus Naur Form
ANTLR	: ANother Tool for Language Recognition
JavaCC	: Java Compiler Compiler
EBNF	:Extented Backus Naur Form
LL(k)	: Left to Left
LR(k)	: Left to right
YaCC	: Yet Another Compiler Compiler
JVM	:Java Virtual Machine
JNI	: Java Native Interface
XML	: Xtensible Markup Language
JSON	: JavaScript Object Notation
JDK	: Java Development Kit

1. GENEL BİLGİLER

1.1. Giriş

Bilgisayar bilimlerinde sayılar ve semboller gibi matematiksel nesneleri işleyecek algoritmaların analizi, tasarımı ve gerçeklemesi ile ilgilenen bilimsel alana simgesel hesaplama denilmektedir. Simgesel hesaplama, bilgisayar cebrini simgelere dayalı olarak ifadeler veya formüller gibi formal dil yapıları ile polinomlar veya matrisler gibi cebirsel yapıların yanı sıra eğriler veya yüzeyler gibi geometrik matematiksel yapılara uygulayan ileri düzey bir programlama alanıdır. Matematiğin herhangi bir alanındaki ifadelerin cebirsel bir şekilde gösterilmesi sonucunda, sonsuz matematiksel varlıkların sonlu ifadeleri olarak birleştirilmesi mümkün hale gelmektedir (Buchberger, 2021). Bu sayede sayısal yöntemlerin getirdiği karmaşık matematiksel problemlere daha kesin ve etkili çözümler üretilebilmektedir.

Uygulamalı matematik, mühendislik, fizik, kimya, bilgisayar bilimleri gibi alanlarda çözümü zor ve karmaşık birçok problem, simgesel hesaplama gibi güçlü bir hesaplama sistemi olmadan çözülememektedir (Borwein & Bailey, 2008). Simgesel hesaplama yöntemleri ile matematiksel problemler daha soyut ve genelleştirilmiş formlara dönüştürülerek analiz edilebilmesi, karmaşık ifadelerin basitleştirilmesi, denklem ve denklem sistemlerinin çözülmesi, sembolik ifadelere dönüşüm, işleme gibi temel işlevler sayesinde karmaşık problemlerin çözülmesi kolaylaşmaktadır. Problemin açık ifadesinin ardından çözüm algoritmasının uygulanması ile çözüm sağlanmaktadır (Gathen & Gerhard, 2013). Özellikle sayısal matematik alanında, daha yüksek verimlilik ve doğruluk sağlayan yeni algoritmaların geliştirilmesi ile birlikte bilgisayarlar, karmaşık matematiksel işlemleri hızlı, hatasız bir şekilde gerçekleştirip daha kesin sonuçlar elde edebilmektedir (Petkovic & Herceg, 2017).

Simgesel matematik, matematikteki temel unsurların ötesinde polinomsal denklem sistemlerinin çözümlenmesi, otomasyon teoremlerinin ispatlanması, sembolik diferansiyel denklemlerin çözümlenmesi, türev ve integral gibi birçok kompleks problem içeren ve hayati önem taşıyan daha karmaşık yöntemleri içerir. Denklemlerin ve ifadelerin sayısal sınırlamalar olmadan araştırılmasını sağlar. Teoremleri kanıtlamak, ifadeleri basitleştirmek, temel matematiksel yapıları ve ilişkileri anlamak için çok değerlidir.

Hesaplama sistemlerinin çok hızlı gelişimiyle matematiksel problemlerin çözümünde bilgisayarların kullanımı yaygınlaşmıştır. Simgesel hesaplama matematiği ve matematiksel yaklaşımları kapsamında bulunduran tüm bilim ve mühendislik alanlarının genel bir uygulama aracıdır. Kullanım alanları, elle yapılabilen ancak hesaplama sistemleri tarafından daha verimli ve doğru bir şekilde gerçekleştirilen hesaplamalardan, elle hesaplamanın ötesine geçen, genellikle rutin olarak yapılabilen işlemlerin makine tarafından tamamlanmasına ve bilgisayar kullanımına rağmen büyük çaba gerektiren hesaplamalara kadar geniş bir yelpazeyi kapsamaktadır (Boyle & Caviness, 1988).

Pade yaklaşımı çözümlenmesi zor ve uzun işlemler gerektiren matematiksel ifadelerden biridir. Sürekli kesirler ve Öklid algoritması temeline dayanan Pade yaklaşımı, 1800'lü yıllarda ortaya çıkmıştır. Yaklaşım sayısal analiz, diferansiyel denklemler, simülasyon, ısı iletimi ve sıvıların dinamiği gibi çeşitli uygulama alanlarına sahip olup birçok algoritma ve matematiksel denklemler ile birlikte, doğrusal olmayan matematiksel problemleri analiz etmek için kullanılmaktadır. Yaklaşımın amacı, polinom gibi araçları kullanarak karmaşık fonksiyonların daha basit fonksiyonlara yaklaşımını sağlamaktır.

1.2. Tezin Amacı

Geçmişten günümüze simgesel hesaplama yöntemleri, birçok matematiksel problemin çözüm aşamasında kullanılmaktadır. Simgesel hesaplama yöntemleri sayesinde matematiksel problemler hem sayısal hem de simgesel işlemlerden yararlanarak daha soyut ve genelleştirilmiş şekilde çözülebilmektedir. Çalışmada kullanılan Pade yaklaşımı, rasyonel polinom ifadelerinin hesaplanmasında simgesel cebir aritmetiğinin ve doğrusal denklem sistemlerinin kullanılmasını gerektiren önemli bir simgesel matematik yöntemidir. Bu sayede klasik yöntemlerle çözümleri zor olan problemler Pade yaklaşımı ile yakınsanarak daha kolay bir şekilde çözülebilmektedir.

Bu tez çalışmasında Pade yaklaşımı simgesel matematik yöntemleriyle birlikte kullanılarak rasyonel polinom ifadelerinin belirlenmesi ve hesaplama adımlarını hem terminal üzerinden gösterebilen hem de bir PDF dokümanına aktarımını yapabilen yazılım uygulamasının geliştirilmesi amaçlanmıştır.

1.3. Literatür Taraması

Simgesel hesaplamalarla ilgili olarak; 1953'te G. W. Leibniz tarafından matematiksel hesaplamalarda, matematiksel ifadeleri ve yöntemleri, algoritmalara ve formüllere dönüştürmeyi kolaylaştırmak için karakteristik sembolik bir dil oluşturulmaya çalışılmıştır (Boyle & Caviness, 1988). Bu çalışmanın sonucunda algoritmalar ve yazılımlarda önemli ilerlemeler kaydedilmiş olup dünya çapında, Altran, Macsyma, mu-Math vb. gibi önemli matematiksel hesaplama yazılımları bulunmaktadır (D'Inverno, 1975). Aynı zamanda matematiksel modelleme ve optimizasyon alanında hesaplama kolaylıkları içeren GAMS, AMPL, Python gibi çeşitli matematiksel programlama dilleri de geliştirilmiştir (Ferris & Coddington,1995). Belirli bir problem için en uygun programlama dilinin seçimi, genellikle ilgili problemin türüne, boyutuna ve istenen çözümün hassasiyetine bağlı olarak yapılmaktadır. Üstün hesaplama ortamlarına sahip Matlab, Mathematica ve Maple gibi araçlar matematiksel ve cebirsel problemlerin yanı sıra sembolik problemlerin çözümünü gerçekleştirmede de kullanılmaktadır (Mohammed, 2018). Ayrıca belirli bir alana yönelik, örneğin cebirsel çalışmalar için CoCoA ve Macaulay, yüksek enerji fiziği hesaplamaları için Schoonship, diferansiyel denklemlerin hesaplanması için DELIA gibi özel amaçlı simgesel hesaplama araçları bulunmaktadır (Gökgöz & Pehlivan, 2018). 21. yüzyılın başlarında bilgisayar erişilebilirliğinin artmasıyla bilgisayar destekli aritmetik hızla gelişmiştir. Bu sayede simgesel hesaplamalardaki ilerlemeler algoritmaların verimli oluşturulmasını, test edilmesini, analiz edilmesini ve analitik olarak türetilen sonuçların doğrulanmasını mümkün kılmıştır.

Simgesel hesaplama alanlarında matematiksel işlemleri gerçekleştirmek için otomatik kod oluşturma araçları kullanılmaktadır. Otomatik kod oluşturma araçları, biçimsel dillerin analizini, yorumlanmasını ve derlenmesini kolaylaştırmak için tasarlanmıştır (Bergmann, 2017). Bu araçlar, programlama dillerinin kurallarını ve yapılarını tanımlamak için bağlamdan bağımsız gramerler (CFG) kullanmaktadır. Genel olarak CFG tanımlamaları üzerinde çalışan otomatik kod üretim araçları, ayrıştırıcı üretmek için kullanılmaktadırlar. Bu araçlar, CFG'leri Backus Naur Form (BNF) ve Extented Backus Naur Form (EBNF) notasyonları gibi çeşitli formatlarda okuyup, ayrıştırıcılar için farklı programlama dillerinde kaynak kod üretebilmektedir.

Matematik ve bilgisayar bilimleri alanlarında geniş kullanıma sahip EBNF notasyonu, belirli biçimleri bulunan ifadeler için formal bir gramer tanımlamanın üst dili olarak işlev

görmektedir (Quinlan, Wells, & Kamareddine 2019). Kaynak verilerin sözdizimi EBNF benzeri notasyonlarda temsil edilerek, dil ayrıştırıcıları kodu otomatik olarak kolayca üretilebilmektedir (Succi & Raymond, 1999). ANTLR ve JavaCC gibi kaynak kod üreticileri (Viswanadha, & Sankar, 2007), bir girdi dosyası içerisinden belirtilen formal grameri temel alarak ayrıştırıcı (parser) oluşturabilmektedir.

Bilim insanları, bilgisayar ve teknolojinin insan hayatına girmesi ile birlikte matematiksel ifadelerin çözümlenmesi için çeşitli algoritmalar geliştirip bu algoritmaların bilgisayar ortamında çözümünü sağlamışlardır. Karmaşık matematiksel problemlerin çözülmesi için güçlü bir araç olan simgesel hesaplamanın mühendislik, matematik, bilgisayar bilimleri gibi birçok alanda çok sayıda çalışması yapılmıştır. 1953 yılında John F. Nolan tarafından "Dijital Bilgisayarlarda Analitik Farklılaşma" adında sembolik hesaplamalar üzerine ilk yüksek lisans tezi başarıyla tamamlanmıştır (Nolan, 1953). 1961 yılında Lisp programla dili ile yazılan "Matematikte Sembolik İntegral Problemlerini Çözen Buluşsal Program" adındaki doktora çalışması sembolik integrali hesaplayan ilk çalışmadır (Slagle, 1963). 2002 yılında çok değişkenli polinomları ve fonksiyonları çözebilmek için GiNac adında sembolik bir hesaplama aracı geliştirilmiştir (Bauer, Frink & Kreckel, 2000). 2009 yılında web-tabanlı bir yorumlayıcı simgesel hesaplamalar kullanılarak gerçekleştirilmiştir (Üstübioğlu, 2009). 2013 yılında matematiksel ifadelerin türevleri üzerine bir yüksek lisans çalışması geliştirilmiştir (Tekbaş, 2013). 2015 yılında matematiksel ifadelerin adım adım değerlendirilmesinin yapıldığı genel bir çalışma doktora tezi olarak gerçekleştirilmiştir (Milani, 2014).

Matematiksel ifadelerden biri olan Pade yaklaşımı, 1892'de Fransız matematikçi Henri Eugene Pade'nin Charles Hermite ile yaptığı çalışma sonucu ortaya çıkmış (Brenzinski, 1996) ancak fikri ortaya atan ve kuvvet serilerinin rasyonel yaklaşımlarının özelliklerini araştıran George Freobenius'a kadar uzanmaktadır (Bojdi, Ahmadi-Asl, & Aminataei, 2013). Serilerin doğrusal olmayan toplamı olarak bilinen bu yaklaşım, 1960'lı yıllardan itibaren matematik, teorik fizik ve çeşitli uygulamalarda yaygın olarak kullanılmaya başlanmıştır (Baker& Gammel, 1961).

Pade yaklaşımı üzerine ilk sistematik çalışmaların 1870'li yıllarda Frobenius tarafından yapılan çalışmalar olduğu varsayılmaktadır (Brezinski, 1991). 1890'lı yılların başında Henri Eugene Pade yaklaşık bir değer üzerine çalışmalar yaparak bu çalışma sonucunda, Paris'teki Ecole Normale Superieure'un Bilimsel İşlemleri dergisinde bir makale

yayımlamış ve makalede bir fonksiyonun rasyonel kesirlerle yaklaşık temsilini ele almıştır (Fowe, 2007). 1965 yılından itibaren matematik, sayısal analiz, teorik fizik, kimya, mekanik ve elektronik alanlarında rasyonel kesirler ile birlikte Pade yaklaşımları üzerine çok miktarda araştırma mevcuttur (Brezinski, 1991). 1992 yılında Bustamante, Hermit-Pade yaklaşımlarına dayalı Nikishin sistemi üzerinde analitik fonksiyonları ele alarak yöntemin yakınsamasını çoklu nokta Pade yaklaşımı kullanarak kanıtlamıştır (Bustamante, 1992).

Pade yaklaşımı, rasyonel kesir yaklaşımının özel ve klasik bir türü olup (Baker & Graves-Morris, 1981) genel olarak, matematiksel fonksiyonların belirli bir sayı aralığında iki polinomun oranı biçimindeki ifadelerini hesaplamak için kullanılmaktadır. Bir matematiksel fonksiyonun polinom dereceleri oranında Taylor serisine açılımı yapılarak hem pay hem de payda ifadelerinin katsayıları belirlenmektedir (Andrianov & Shatroy, 2021). Bu yol içerisinde ilgili fonksiyonun en iyi rasyonel fonksiyon temsili elde edilmektedir. Pade yaklaşımı, bir fonksiyonun belirli noktalardaki değerleri için daha doğru sonuçlar üretebilmektedir. Tek başına kullanılan kuvvet serileri, sınır değeri problemlerini çözmek için pek kullanışlı değildir. Ancak Pade yaklaşımı ile birlikte kullanıldıklarında, adi diferansiyel denklemlerin sınır değeri problemlerini çözmek için oldukça etkili bir araç haline gelmektedir (Assche, 2006).

Matemetiksel gelişmelerle birlikte 2006 yılında Assche ve diğerleri Pade yaklaşımının bazı yakınsama özelliklerini göstermişlerdir (Gonchar, Rakhmanov & Sorokin, 2007). 2007 yılında Gonchar ve diğerleri, Hermit-Pade yaklaşımlarıyla Markov fonksiyon sistemi üzerinde vektör potansiyeline göre belirli denge problemlerini elde etmişlerdir (Liu, 2009). 2009 yılında Liu ve diğerleri, doğrusal olmayan diferansiyel fark denklemlerini Pade yaklaşımı ve adomian ayrıştırma yöntemi kullanarak araştırmışlar ve sayısal veriler yaklaşık çözüme adomian ayrıştırma yöntemini kullanmaktan daha hızlı yakınsama oranı ve daha yüksek hassasiyetle ulaşabileceğini göstermiştir (Lorentzen, 2010). 2010 yılında Lisa Lorentzen, Pade yaklaşımını ve sürekli kesirleri araştırmış, sürekli kesirlerin yakınsamasını sağlayarak hızlı ve istikrarlı bir yaklaşım yakınsaması elde (Natori, 2012) 2012 yılında, Natori, Pade yaklaşımından yararlanarak zaman gecikmeli tasarım yöntemini araştırmış ve bu sayede detaylı kutup konumu elde etmiştir (Turut & Güzel, 2013). 2013 yılında Turut ve Güzel, doğrusal ve doğrusal olmayan kesirli kısmi diferansiyel denklemleri çözerek çok değişkenli Pade yaklaşımını ortaya koymuşlardır (Dogonchi, Hatami, Hosseinzadeh & Domairry, 2015). 2015 yılında Dogonchi ve diğerleri, sıkıştırılamaz Newton ortamında dik

düşmenin durağan olmayan hareketinin hız ve ivmesini Pade yaklaşımı kullanılarak elde etmişlerdir (Kakde, 2008).

1.4. Pade Yaklaşımı

Pade yaklaşımı, bir fonksiyonun iki kuvvet serisi oranı olarak genişletilip bir $f(x)$ fonksiyonunun Taylor serisi açılımının katsayılarını kullanarak hem pay hem de payda katsayılarını belirleyen ifadedir (Quinlan, Wells, & Kamareddine 2019).

$f(x)$ bir fonksiyon ve $l, m > 0$ olmak üzere

$$f(x) = \sum_{i=0}^{\infty} c_i x^i \tag{1}$$

genelleştirilmiş bir Pade yaklaşım denklemi (1)'de gösterilmiştir.

$$\left[\frac{L}{M}\right] = \frac{\sum_{j=0}^{l} a_j x^j}{\sum_{k=0}^{m} b_k x^k} = \frac{a_0 + a_1 x + a_2 x^2 + \cdots + a_l x^l}{b_0 + b_1 x + b_2 x^2 + \cdots + b_m x^m} \tag{2}$$

Denklem (2), genelleştirilmiş hali denklem (1)'de verilmiş olan iki rasyonel polinomdan oluşan bir fonksiyonun Maclaurin serisine göre açılmış hali olup Pade yaklaşımını ifade etmektedir. Pay ve paydadaki polinom denklemlerinin katsayıları, Taylor Serisi açılımını mümkün olduğunca eşleştirmek amacıyla bu serinin katsayıları kullanılarak elde edilmektedir. Paydanın tanımsız çıkmaması için b_0 değeri 1 alınmaktadır. Denklem (2)'deki fonksiyonun payının $l + 1$ tane bağımsız katsayısı, paydasının m tane bağımsız katsayısı olup tüm yaklaşımda $l + m + 1$ tane bilinmeyen katsayı bulunmaktadır.

$l + m$ dereceli kuvvet serisine denk gelen Pade yaklaşımı denklem (3)'de gösterilmektedir.

$$\frac{a_0 + a_1 x + a_2 x^2 + \cdots + a_l x^l}{1 + b_1 x + b_2 x^2 + \cdots + b_m x^m} = c_0 + c_1 x + c_2 x^2 + \cdots + c_{l+m} x^{l+m} \tag{3}$$

Pade yaklaşım denklemleri (1), (2) ve (3)'te gösterilen denklemler biçimde tanımlanmaktadır. Bu denklemler sayesinde $l+m'$ ninci dereceden Pade yaklaşımı oluşturulmuştur. $f(x)=\sum_{i=0}^{\infty} c_i x^i$ kuvvet serisi denklemi için hesaplamaları kolaylaştırmak amacıyla ilgili açılım, koordinat düzleminin orijin noktasına göre yani Maclaurin serisine göre yapılarak, seri katsayılarının hesaplaması gerçekleştirilmiştir.

1.4.1. Pade Yaklaşımı için Örnek Çözüm

Bir [2,2] $f(x)=sin(x)$ fonksiyonu için;

$$f(x)Q_m(x)-P_n(x)=0 \tag{4}$$

eşitliğindeki Pade yaklaşımı polinomlarını belirleyelim. Fonksiyon derecesi [2,2] şeklinde belirtildiği için polinom denklemleri x^2 ifadesine kadar açılır.

$$P_2(x)=p_0+p_1x+p_2x^2 \tag{5}$$

$$Q_2(x)=1+q_1x+q_2x^2 \tag{6}$$

$f(x)=sin(x)$ fonksiyonunun Maclaurin serisine göre açılımı denklem (5) ve (6)'da gösterilen $P_n(x)$ ve $Q_m(x)$ biçiminde olup denklemlerde toplam 5 adet bilinmeyen bulunduğu için 5. terime kadar açılmaktadır. $sin(x)$ fonksiyonu 5.terime kadar açılıp $x=0$ değerlerinin yerine koyulduğu denklemler (7) ve (8)'de verilmiştir. Bu denklemlerin sonucuna göre ifadenin Maclaurin serisine göre açılımı sonucunda $x-\frac{x^3}{6}$ denklemi elde edilmiştir.

$$f(x)=sin(0)+sin'(0)x+\frac{sin''(0)x^2}{2!}+\frac{sin'''(0)x^3}{3!}+\frac{sin^{(4)}(0)x^4}{4!}+O(x)^5 \tag{7}$$

$$=0\ +\ 1*x\ +\ \frac{(0*x^2)}{2!}-\frac{(1*x^3)}{3!}+\frac{(0*x^4)}{4!}=x-\frac{x^3}{6}+O(x)^5 \tag{8}$$

$f(x)Q_m(x) - P_n(x) = 0$ denklemine göre $f(x)$, $Q_m(x)$ ve $P_n(x)$ ifadeleri denklem (8)'de gösterildiği gibi yerine konulur.

$$\left(x - \frac{x^3}{6}\right)(1 + q_1x + q_2x^2) - (p_0 + p_1x + p_2x^2) = 0 \quad (9)$$

$$-p_0 + x(1 - p_1) + x^2(p_2 + q_1) + x^3\left(q_2 - \frac{1}{6}\right) - x^4\left(\frac{q_1}{6}\right) - x^5\left(\frac{q_2}{6}\right) + O(x)^5 = 0 \quad (10)$$

$$-p_0 = 0 \qquad p_0 \to 0 \quad (11)$$

$$1 - p_1 = 0 \qquad p_1 \to 1 \quad (12)$$

$$q_1 - p_2 = 0 \qquad q_1 = p_2 \quad (13)$$

$$q_2 - \frac{1}{6} = 0 \qquad q_2 \to \frac{1}{6} \quad (14)$$

$$\frac{q_1}{6} = 0 \qquad q_1 \to 0, \qquad p_2 \to 0 \quad (15)$$

Yukarıda verilen denklem (11), (12), (13), (14) ve (15)'ten hareketle $f(x) = \sin(x)$ fonksiyonunu temsil edecek $R_{2,2}(x) = \frac{p_0+p_1x+p_2x^2}{1+q_1x+q_2x^2}$ biçimindeki rasyonel polinomlar, *x=0* noktasına göre bilinmeyen ifadelerin yerine koyularak seriye açılması sonucunda,

$$R_{2,2}(x) = \frac{0+1x+0x^2}{1+0x+\frac{x^2}{6}} = \frac{x}{1+\frac{x^2}{6}} = \frac{6x}{6+x^2} \quad (16)$$

denklem (16)'da gösterilen Pade yaklaşımı denklemi elde edilmiştir.

1.5. Yorumlayıcılar (Interpreter)

Yorumlayıcılar kaynak kodu yüksek seviyeli programlama dili ifadeleri üzerinden değerlendirebilen yazılım araçlarıdır. Bazı türleri ara kod dönüşümü yaparak yorumlama sürecinin performasını yükseltmeye çalışır. Bütün program için çalıştırılabilir kod üretilmeden, ara kodun değerlendirmesi ile ilgilenilir.

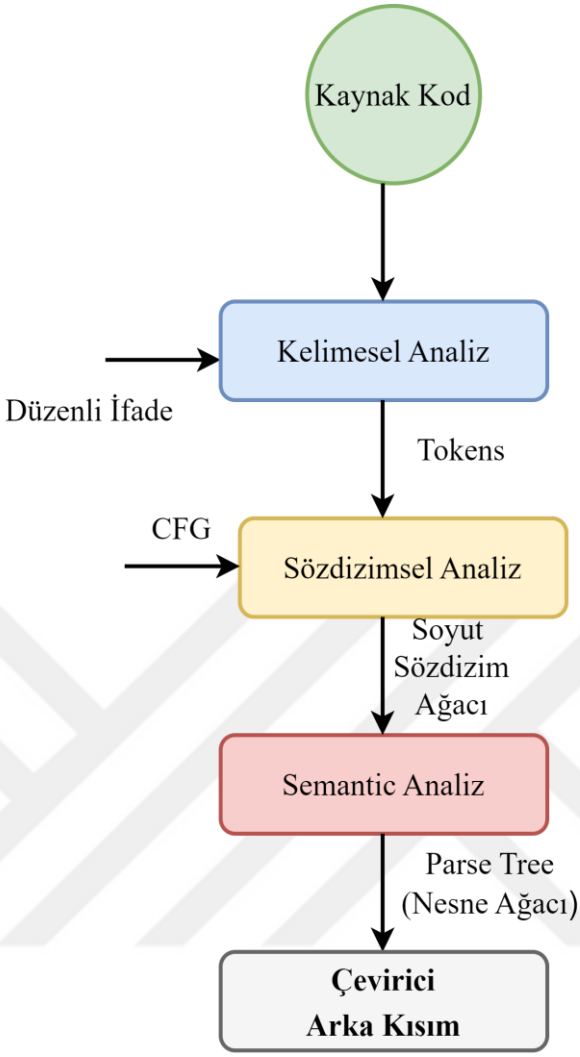
Bir yorumlayıcı, bir programlama dilinin yazım kurallarını ve sözdizimini anlayarak kaynak kodun doğru bir şekilde çalıştırılmasını sağlar. Bir program tekrar çalıştırılmak istendiğinde, yorumlayıcı kaynak kodu yeniden okuyup işler. Her satırı sırayla işledikleri için hataların bulunması ve düzeltilmesi kolaydır. Ancak, programın tamamlanması için fazla zaman harcanması bir dezavantajdır.

1.6. Derleyiciler (Compiler)

Derleyiciler, kaynak kodu makine diline çevirerek bilgisayar tarafından işlenebilir hale getirir. Bu süreçte, yüksek seviyeli bir programlama dilinde yazılmış olan kod, düşük seviyeli bir makine diline dönüştürülür(Farhanaaz, 2008).

1.7. Çevirici Ön Kısım

Bir çevirici ön kısmı kelimesel analiz, simgesel analiz, anlamsal analiz aşamalarından oluşmaktadır. Hem derleyiciler hem de yorumlayıcılarda çevirici ön kısımları Şekil 1'de gösterilmiş olup her ikisinde de soyut sözdizim ağacı oluşturmaktadır.



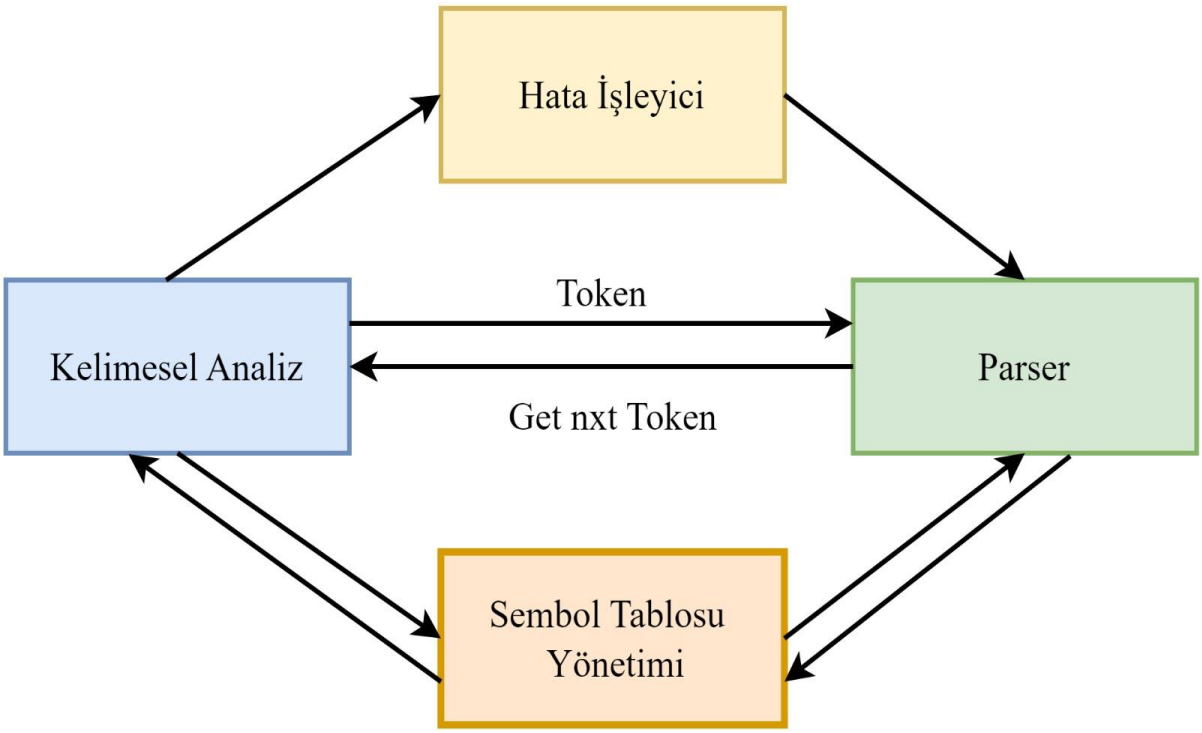
Şekil 1. Çevirici ön kısım aşamaları

1.7.1. Kelimesel Analiz

Bir derleyicinin derleme aşamalarından ilki kelimesel analiz aşamasıdır. Kelimesel analiz, aynı zamanda "Doğrusal Analiz" veya "Tarama" olarak da bilinir. Amacı kaynak kodu, yorumlayıcı veya derleyici tarafından işlenebilecek daha küçük ve yönetilebilir birimlere yani birim sözcük ayırmaktır. Bu aşama programlama dillerinin derlenmesi ve yorumlanması kadar doğal dilin işlenmesinde de önemli bir adımdır.

Kelimesel analiz aşamasında kaynak program tek tek okunarak birim sözcük (token) denilen anlamlı sözcük dizilerine dönüştürülür. Düzenli ifadeler yardımı ile temsil edilen birim sözcükler bir ya da daha çok karakterden oluşabilmektir.

Kelimesel analiz aşaması; sembol tablosunda birim sözcük tanımlamaya yardımcı olmak, kaynak programdaki boşlukları ve açıklamaları kaldırmak, hata mesajlarını kaynak programla ilişkilendirmek ve kaynak programdan giriş karakterlerini okumak görevlerini yerine getirmektedir.



Şekil 2. Kelimesel analiz aşaması

Şekil 2'de (Flex Ayrıştırıcı, 2024), kelimesel analiz aşaması gösterilmiştir. Bu aşamada kaynak program okunup birim sözcükler üretilir. Birim sözcükler sembol tablosunda saklandığı gibi ayrıştırıcı adı verilen bir sonraki aşamaya da gönderilebilir. Bu işlem, kaynak program tamamen taranana kadar devam eder. Kelimesel analiz aşamasında ayrıştırıcıda ortaya çıkan hatalar, hata işleyiciye rapor edilir. Bu aşama aynı zamanda boşlukların, sekmelerin, yeni satırların ve açıklamaların çıkarılması gibi kaynak programın temizlenmesinden de sorumludur.

$f(x) = x^2 sin(x)$ ifadesi için oluşturulan birim sözcüler dizisi Tablo 1'de gösterilmiştir. Bu dizi programdaki her kelime, sembol veya ifadenin karşılık geldiği birim sözcükleri ve bunların türlerini içerir.

Tablo 1. $f(x) = x^2 * sin(x)$ ifadesinin birim sözcük dizisi

Token	**Tür**
(	Açma parantezi
X	Değişken
)	Kapama parantezi
=	Atama operatörü
X	Değişken
^	Üs alma operatörü
2	Sayı
*	Çarpma Operatörü
Sin	Sinus fonksiyonu
(	Açma parantezi
X	Değişken
)	Kapama parantezi

Kelimesel analiz aşamasının gerçeklenmesi sürecinde düzenli ifadelerden yararlanılmaktadır. Düzenli ifadeler, biçimsel dil teorisinde derleyicilerdeki tarayıcıların bir parçası olarak ortaya çıkmış olup bir dizi dizeyi belirtmek için kullanılan gösterimdir. Ayrıca metinsel verilerde arama ve değiştirme işlemleri gerçekleştirmek için bilgisayar bilimlerindeki en güçlü araçlardan biridir. Düzenli ifadelerin önem arz etmesinin temel

nedenlerinden biri, metni çok esnek ve verimli bir şekilde aramaya ve değiştirmeye izin vermesidir.

Tablo 2. Düzenli ifade örnekleri

Sözdizimi	Tanım	Örnek Desen	Örnek Eşleşme
^	Satır başı işareti.	^k	**k**alem, **k**arkaburun
$	Satır bitişi işareti.	t$	sü**t,** ahtapo**t**
{}	Satır başındaki karakterin ya da ifadenin n kez tekrarlanması ile eşleştirilir.	“a{1,2}”	“**a**”, “**aa**”
\b	Bir kelimenin sonundaki ya da başındaki karakterleri eşleştirir.	\bçiçek\b	Bahar gelince **çiçek** açar.
x+	Bir veya daha fazla sayıda eşleştir.	va+	v**a**r, v**aa**t
x?	0 ya da daha fazla eşleştirme.	kar?n	ka**a**n, ka**ra**n

Tablo 2’de bazı düzenli ifadelerin sözdizimleri, açıklamaları ve bu sözdizimlerinden ortaya çıkan örnek eşleştirmeler gösterilmektedir.

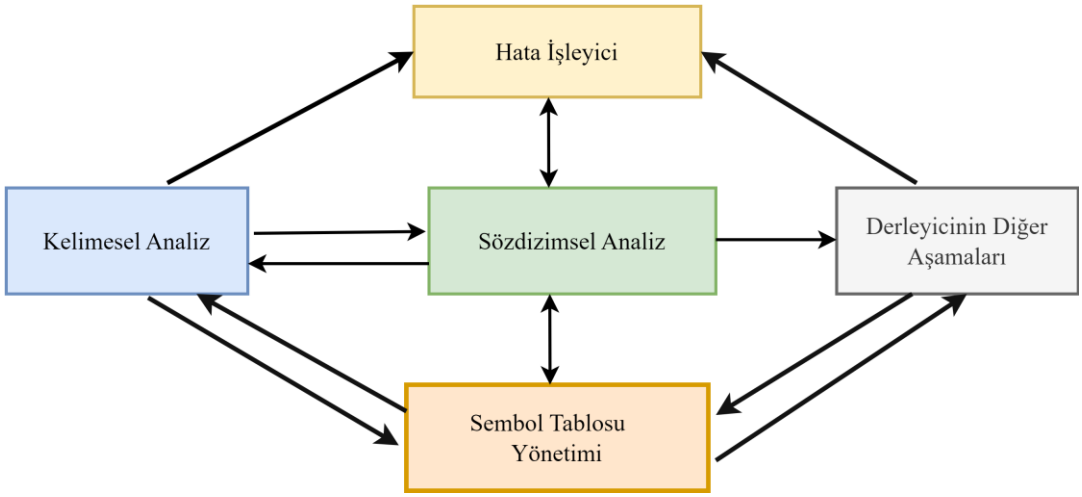
Düzenli ifadelerin aşağıda belirtildiği gibi çeşitli avantajları bulunmaktadır.

- Metin işleme, metin eşleştirme yapmak için son derece güçlü ve esnek bir araç olup karmaşık desenleri tanımlamak ve bulmak için kullanılabilir.

- Çoğu programlama dilinde düzenli ifadeler desteklendiği için farklı platformlarda aynı ifadeler kullanılabilir.
- Metinlerde belirli desenleri bulmak veya değiştirmek için kullanıldığından, metin işlemede büyük ölçüde kolaylık sağlar ve zaman kazandırır.
- Basit düzenli ifadelerini öğrenmek ve kullanmak, diğer metin işleme yöntemlerine göre daha kolaydır.

1.7.2. Sözdizimsel Analiz

Sözdizimi, bir dilin biçimsel yapısına uygun ifadeler oluşturan sözcük ve sözcük öbeklerinin düzeni ile ilgilidir. Kaynak kod kelimesel analiz aşamasında birim sözcüklere bölündükten sonra, sözdizimsel analiz aşamasında dil bilgisi kurallarına uygunluğu kontrol edilmektedir. Ayrıştırıcının (parser) kaynak verinin biçimsel denetimi ve sözdizim ağacının üretimi gibi iki temel görevi vardır. BNF, CFG gibi gramer türleri biçimsel denetim için kullanır ve içerisine sözdizimsel ağaç üreten ifadeler eklenir. Böylece kaynak kod, sözdizim ağacı adı verilen bir veri yapısına dönüştürülür. Veri yapısında, ağacın yaprakları sözdizimsel analiz sonucunda ortaya çıkarılan birim sözcükler olup, değerlendirilmeleri kaynak koddaki sıra ile soldan sağa ya da sağdan sola doğru yapılabilmektedir.



Şekil 3. Sözdizimsel analiz şeması

Şekil 3'te gösterilen sözdizimsel analiz aşamasında girdi olarak birim sözcük alınıp, çıktı olarak sözdizim ağacı üretilir. Ayrıca burada eksik parantez, fazla virgül vb. gibi sözdizimi hatalarının da kontrolü yapılır.

1.7.3. Semantik (Anlamsal) Analiz

Semantik diğer bir adıyla anlamsal analiz derleme sürecinin üçüncü aşamasıdır. Anlamsal analiz aşaması, kaynak kodun anlamsal olarak doğru olup olmadığının yani dilin tür sistemine ve diğer anlamsal kurallarına uyup uymadığının kontrol edilip anlamının incelenmesi sürecini ifade eder.

Anlamsal analiz aşaması hatasız ve verimli kod üretmede önemli bir rol oynar. Bu aşamada anlamsal hataların kontrol edilmesi, tür kontrolü, kodun belirli kurallar ve kısıtlamalara göre doğrulanması yapılır. Ayrıca bu aşama tür ve kapsam ihlalleri gibi hataları yakalayarak daha sağlam ve hatasız kod yazılmasına yardımcı olur.

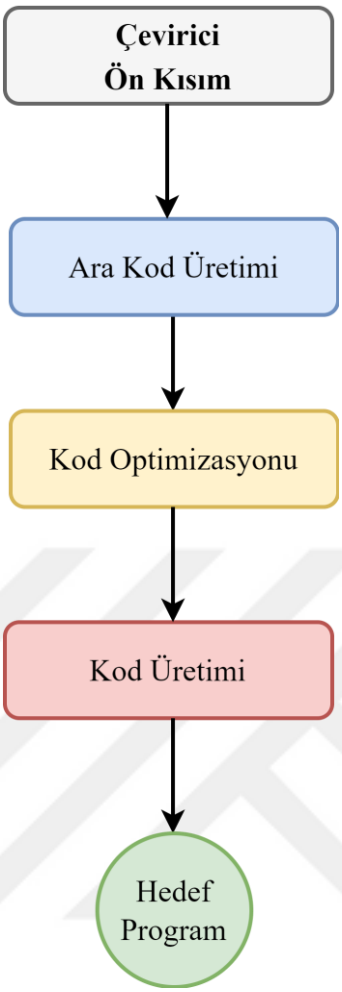
Anlamsal analiz aşamasında;

- Ayrıştırma ağacının doğrulanması ve modifikasyonu,
- Tür denetimi,
- Etiket denetimi,
- Akış denetimi

gerçekleştirilir.

1.8. Çevirici Arka Kısım

Çevirici arka kısmı, ara kod üretimi, kod optimizasyonu ve kod üretimi aşamalarından oluşup kaynak kodun makine koduna dönüşümünü sağlamaktadır. Kod optimizasyonu aşaması, daha iyi hedef kod elde edilecek şekilde ara kodu iyileştirmeye çalışır. Daha hızlı, daha kısa veya daha az güç tüketen hedef kod şekline dönüştürülür. Kod üretimi aşaması kaynak programın ara temsilini girdi olarak alır ve makine dilinde kod üretir. Bu aşamada, kod optimizasyonu, bellek yönetimi gibi çeşitli optimizasyon teknikleri kullanılır. Bu aşamanın, derleyicinin performansı ve ürettiği kodun kalitesi üzerinde önemli bir etkisi bulunmaktadır. Şekil 4'te çevirici arka kısım aşamaları gösterilmiştir.



Şekil 4. Çevirici arka kısım aşamaları

1.9. Bağlamdan Bağımsız Gramer (Context Free Gramer - CFG)

Gramer, bir dildeki kelimelerin ve kelime öbeklerinin bir araya getirilerek anlamlı cümleler oluşturmasını tanımlayan kurallar bütünüdür. Bu kurallar, kelimelerin sıralamasına ve işlevlerine ilişkin temel prensipleri içeren dilin sözdizimsel yapısını belirlemektedir.

Bağlamdan bağımsız gramer, bilgisayar bilimlerinde birçok programlama dilinin, doğal dilin ve diğer biçimsel dillerin sözdizimini tanımlayan temel bir kavramdır. Dilin sözdizimsel yapısını tanımlayan kuramsal bir modeldir. Bağlamdan bağımsız dilbilgisinde cümlelerin oluşturulması için kullanılan kurallar, kelimelerin veya kelime öbeklerinin bağlamına bakılmaksızın uygulanmaktadır. Şöyle ki bir kelimenin veya kelime öbeğinin anlamı, cümledeki konumundan veya diğer kelimelerle olan ilişkisinden etkilenmemektedir.

Bağlamdan bağımsız bir gramer, sonlu olmayan semboller dizisi (V), başlangıç sembolü (S), sonlu semboller dizisi (T), üretim kuralları kümesinden (P) oluşmaktadır. Sonlu semboller dizisi, kelimesel çözümleyiciler aracılığıyla oluşturulan sözcükler tarafından ifade edilmekte olup alfabedeki simgeler kullanılarak oluşturulan kelimeleri belirtmektedir. Başlangıç simgesi, gramerin başlatımıyla ilgili kuralı tanımlamaktadır. Sonlu olmayan semboller dizisi kuralın sol kısmında kalan yapıyı ifade etmektedir. Sonlu sembollere veya sonlu olmayan sembollere geçiş yapabilen ifadelerdir. Üretim kuralları dizisi, sonlu ve sonlu olmayan sembollerden oluşan ve dilin gramer kurallarını belirleyen yapıdır.

Tablo 3. Gramer örneği

$S \longrightarrow aSb \mid S\mathrm{A} \mid \varepsilon$
$\mathrm{A} \longrightarrow ab \mid \mathrm{bC}$
$\mathrm{C} \longrightarrow \mathrm{cc} \mid \mathrm{ac}$

Tablo 3'te verilen gramer örneğinde S başlangıç sembolünü, P sonlu ve sonlu olmayan sembollerden oluşan üretim kurallarını, a, b, c sonlu sembollerini, S, A, C sonlu olmayan sembolleri göstermektedir.

Söz dizimi analizinde ilk önce kaynak veri soldan sağa doğru taranmakta ve bir dizi terminale bölünmektedir. Sonrasında gramer yardımıyla terminallerin dizideki sırası incelenmektedir.

1.9.1. Ayrıştırıcı (Parser)

Ayrıştırıcı diğer bir adıyla parser, yorumlayıcı veya derleyicideki doğru sözdizimini kontrol eden ve giriş belirteçlerinde sembolik bir soyut sözdizim ağacı gibi veri yapısı oluşturan bileşendir. Oluşturulan sembolik yapı daha sonra yorumlanabilip derlenebilmektedir.

Bilgisayar bilimlerinde ayrıştırma, belirli bir dilbilgisine göre bir dizi belirteçten oluşan bir metni analiz etme sürecidir. Genellikle bir derleyici veya yorumlayıcı içinde kullanılmaktadır. Ayrıştırıcılar elle programlanabilir veya bir ayrıştırıcı oluşturma aracı tarafından oluşturulabilir.

1.9.2. BNF (Backus Naur Form)

BNF, açılımı Backus Naur Form olan bilgisayar bilimlerinde programlama dillerinin ve diğer biçimsel dillerin söz dizimini tanımlamak için kullanılan bir gösterimdir. Özellikle formel dil tanımlamalarında, gramerlerin, sentaks kurallarının ve dil yapılarının açıklanmasında yaygın olarak kullanılır.

BNF, basitçe bir dilin yapısını kurallar halinde ifade eder. Bu kurallar terminal ve non-terminal sembollerden oluşur. Terminal semboller, programlama dilindeki anahtar kelimeleri, değişken isimleri oluştururken, sayılar noktalama işaretleri gibi aslında programın kendisini oluşturan temel öğeleri oluşturmaktadır. Non-terminal semboller, "< >" işareti arasında yazılır ve programın daha karmaşık yapı taşlarını temsil eder. Bir deyim, terim veya bloğu ifade edebilir.

BNF, bir programlama dili ya da biçimsel dilin kurallarını net bir şekilde belirtilerek, anlaşılırlığa ve yapısal tutarlılığa olduğu kadar yorumlayıcı veya derleyici geliştirme süreçlerinin yönetimine de olumlu katkılar sağlar.

1.9.3. EBNF (Extended Backus-Naur Form)

EBNF, açılımı Genişletilmiş Backus-Naur Formu (Extended Backus-Naur Form) olan BNF'in daha gelişmiş halidir. Bilgisayar bilimlerinde programlama dillerinin ve biçimsel dillerin söz dizimini tanımlamak için kullanılır. BNF'nin daha açıklayıcı, daha esnek ve daha güçlü bir versiyonudur. EBNF, bilgisayar bilimlerinde ve özellikle de programlama dillerinin tanımlanması ve diğer formal dil sözdizimlerinin betimlenmesi için kullanılır.

EBNF sayesinde programlama dillerinin ve biçimsel dillerin kurallarını daha net ve anlaşılır bir şekilde ifade etmek mümkündür. Bu da dillerin tasarımı, anlaşılması ve derleyicilerin yazılması aşamalarında kolaylık sağlar.

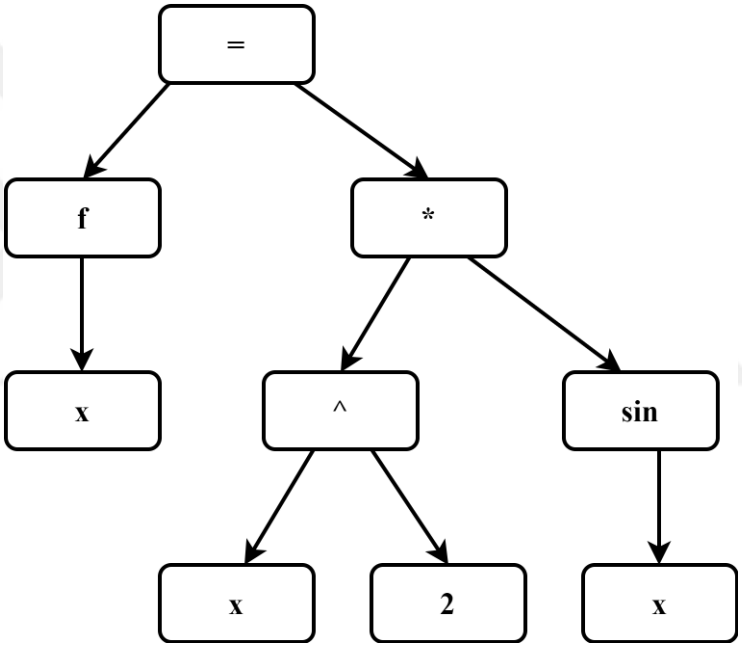
1.9.4. Soyut Sözdizim Ağacı (AST)

Soyut sözdizimi ağacı (AST), sözdizim sınıflarından türetilen nesneleri birbirine bağlayarak kaynak verilerini bir ağaç veri yapısı biçiminde oluşturmaktadır. Kaynak kodun operatörleri ve çağrım ifadeleri gibi işlevsel bileşenleri ağacın ara düğümlerini oluştururken, sabitler ve değişkenler gibi bileşenleri yaprakları yapar. Bu sayede ağacın her düğümü, ilgili kod bileşeninin türüne bağlı olarak farklı sözdizimi sınıfındaki nesneleri içerebilir.

Sözdizimi ağaçları, kaynak veriler üzerinden gerçekleştirilmesi zor olan tür kontrolü ve kod yorumlama gibi işlemleri gerçekleştirmek için kullanılır. Kaynak verinin sözdizim analizinde olduğu gibi sözdizim ağaçlarının üretiminde de JavaCC kullanılabilir.

Sözdizim ağaçları, dilbilgisi kurallarına göre kaynak veriler arasında ilişki kurmanın güçlü bir yöntemidir. Nesne tabanlı ve hiyerarşik yapıları sayesinde kod bileşenlerini daha kolay erişilebilir ve manipüle edilebilir olmasını sağlamaktadır.

$f(x) = x^2 sin(x)$ şeklindeki bir matematiksel ifade için Tablo 1'de gösterilen birim sözcük tablosu alınır ve Şekil 5'teki sözdizim ağacı üretilir.



Şekil 5. $f(x) = x^2 sin(x)$ ifadesi için soyut sözdizim ağacı

1.10. Derleyiciler ile Yorumlayıcıların Karşılaştırılması

Tablo 4'te derleyiciler ile yorumlayıcıların bir karşılaştırması yapılmıştır.

Tablo 4. Karşılaştırma tablosu

Derleyici	Yorumlayıcı
Programın tamamı girdi olarak alınır ve tamamı üzerinde aynı anda çalışır.	Programı satır satır çalıştırır. Girdi olarak her seferinde bir ifade alır.
Derleme sonrasında yürütülebilir bir dosya veya ara ürün üretirler.	Kaynak kodu doğrudan okuyup yürütürler, yeni bir dosya veya ara ürün üretmezler.
Derlenmiş programın her zaman derlenmesine gerek yoktur. Bir kez derlenip istenilen zamanda çalıştırılabilir.	Yorumlanan programlar her çalıştırıldıklarında satır satır yorumlanır.
Hata, programın tamamı söz dizimi ve diğer hatalar açısından kontrol edildikten sonra raporlanır.	Hata, ilk hatayla karşılaşıldığı anda raporlanır. Programın geri kalanı, mevcut hata giderilene kadar kontrol edilmez.
Derlenmiş bir dilde hata ayıklamak daha zordur.	Hata ayıklama kolaydır çünkü yorumlayıcı durur ve hatalarla karşılaştıkça rapor eder.
Derleyici tarafından oluşturulan yürütülebilir kod, optimize edildiğinden ve doğrudan yürütülebildiğinden, derlenmiş bir program genellikle hızlı çalışır.	Yorumlanan bir programın çalışma zamanında satır satır çevrilmesi ve çalıştırılması gerektiğinden daha yavaş çalışır.
C, C++, COBOL	Python, Ruby, PHP, Perl, MATLAB, Lisp

1.11. Ayrıştırma Algoritmaları

Ayrıştırma algoritmaları yukarıdan aşağıya ayrıştırma ve aşağıdan yukarıya ayrıştırma olmak üzere iki sınıfa ayrılmaktadır.

1.11.1. Yukarıdan Aşağıya Ayrıştırma Algoritması (Top-Down Parsing)

Yukarıdan aşağıya ayrıştırma algoritmasında, ayrıştırıcı, gramerin başlangıç sembolünden başlayarak ayrıştırma ağacını oluşturmaya başlar. Bu süreçte, ayrıştırma ağacı yukarıdan aşağıya yani kökten yaprağa doğru oluşturulur. Özyinelemeli iniş ayrıştırma ve LL ayrıştırma, yukarıdan aşağıya ayrıştırma algoritması örneğidir. LL(k) ayrıştırma, belirli bir gramerden tahmine dayalı bir ayrıştırma tablosu oluşturur ve bunu girişi ayrıştırmak için kullanırken, özyinelemeli ayrıştırma bir dilbilgisinin sözdizimini tanımak için bir dizi özyinelemeli algoritma oluşturmayı gerektirir.

Bir LL ayrıştırıcısı bağlamdan bağımsız bir dil için yukarıdan aşağıya bir ayrıştırıcıdır. Girdiyi soldan sağa ayrıştırarak cümlenin en soldan türetilmesini gerçekleştirir. LL ayrıştırıcısı bir cümleyi ayrıştırmak için k adet ileriye dönük birim sözcük kullanıyorsa buna LL(k) ayrıştırıcı adı verilir. LL(k) ayrıştırıcıda bulunan ilk L harfi soldan sağa doğru tarama yaklaşımını, ikinci L harfi soldan eksiltme yaklaşımını, k harfi ise kural seçiminin en az kaç birim sözcük ile yapılacağını belirtmektedir. LL gramerlerinin, özellikle de LL(1) gramerlerinin ayrıştırıcı olarak oluşturulması basit olduğundan, birçok bilgisayar dili LL(1) olacak şekilde oluşturulmuştur; 1 sayısı, ayrıştırma sürecinin her aşamasında ileriye dönük kararlar için tek bir giriş simgesinin kullanıldığını belirtir. LL(k) ayrıştırıcıları terminal olmayan bir şeyle karşılaştıklarında, bunun hangi üretimle değiştirileceğini tahmin etmeleri gerekir. Temel LL algoritması, [S, $] içeren bir yığınla başlar ve tamamlanana kadar ilgili eylemi gerçekleştirir.

1.11.2. Aşağıdan Yukarıya Ayrıştırma Algoritması (Bottom-Up Parsing)

Aşağıdan yukarıya ayrıştırma ile bir problem en küçük parçalara bölünerek bu parçalar bir araya getirilip çözüme ulaşılır. Yani bir ağacın yaprak düğümlerinden başlayıp kök düğüme ulaşana kadar yukarıya doğru çalışır. Bu yaklaşım, daha büyük ve karmaşık problemleri ele alırken genellikle daha iyi performans sağlar.

Yukarıdan aşağıya ayrıştırma algoritmasının tersine, bu algoritmada herhangi bir başlangıç noktası bulunmayıp problemleri en küçük bileşenlerden başlayarak çözer. Ardından, bu bileşenleri bir araya getirerek daha büyük yapıları oluşturur.

Daha küçük parçalara ayrılmış kod blokları, genellikle yeniden kullanılabilir ve daha iyi anlaşılabilir olduğundan aşağıdan yukarıya ayrıştırma algoritması genellikle daha verimli ve modüler bir kod oluşturmayı sağlar.

LR ayrıştırıcısı, aşağıdan yukarıya bir ayrıştırma örneğidir. LR ayrıştırıcısındaki L girdinin soldan sağa doğru okunacağını, R ise türetimin en sağdan yapılacağını belirtir. Yani LR(k) ayrıştırıcılar, soldan sağa doğru çalışan ve k adet ileriye dönük sembol kullanan ayrıştırıcı sınıfıdır. CFG'leri ayrıştırmak için güçlü bir yaklaşım sunmasının yanı sıra LL(1) gibi ayrıştırıcılara göre daha geniş bir gramer yelpazesini işleyebildikleri için derleyici tasarımında yaygın olarak kullanılır.

LR(k) ayrıştırma algoritmaları genellikle derleyicilerin ve yorumlayıcıların yapılandırılması ve dil analizi gibi alanlarda kullanılır. LR(k) algoritmalarının karmaşıklığı, belirli bir dil için daha karmaşık bir ayrıştırıcı oluşturabilme yetenekleri nedeniyle tercih edilir. Bununla birlikte, daha yüksek k değerleri genellikle daha fazla hesaplama gerektirir ve daha karmaşık hale gelebilir.

1.11.3. LL-LR Ayrıştırma Algoritmalarının Karşılaştırılması

LL(k) ve LR(k) algoritmalarının bir karşılaştırması Tablo 5'te verilmiştir.

Tablo 5. LL-LR ayrıştırma algoritmalarının karşılaştırılması

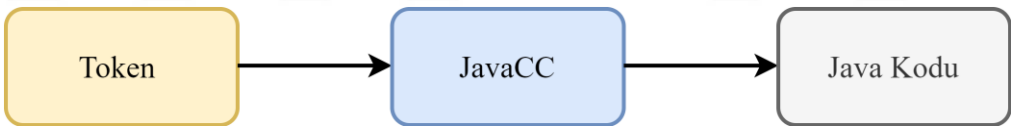
LL Ayrıştırıcı	LR Ayrıştırıcı
LL'nin ilk L'si soldan sağa, ikinci L ise en soldan türetme içindir.	LR'nin L'si soldan sağa, R ise en sağdan türetme içindir.
En soldaki türetimi takip eder.	En sağdaki türetmenin tersini takip eder.
LL ayrıştırıcı kullanılarak ayrıştırıcı ağacı yukarıdan aşağıya olacak şekilde oluşturulur.	Ayrıştırıcı ağacı aşağıdan yukarıya doğru inşa edilmiştir.
LL ayrıştırıcısında terminal olmayanlar genişletilir.	LR ayrıştırıcısında terminaller sıkıştırılır.
Başlangıç sembolü (S) ile başlar.	Başlangıç sembolü (S) ile biter.
Kullanılan yığın boşaldığında sona erer.	Boş bir yığınla başlar.

1.12. Kelimesel Çözümleyici Üretme Araçları

Kelimesel çözümleyici (ayrıştırıcı) üretme araçları karakter kümelerini, birim sözcük olarak adlandırılan kelimeler topluluğuna dönüştürür. Bu araçlar, biçimsel tanımlamalara göre kaynak veri içerisinde birim sözcük çözümlemesi yapacak kodu otomatik olarak üretir. Flex (URL-1), Lex (URL-2), Quex (URL-3), YaCC (URL-4) C/C++ programlama dili için, JavaCC (URL-5), SableCC (URL-6), ANTLR (URL-7) Java dili için bilinen farklı otomatik kod üretme araçları örneklerindendir. Bu tez çalışması Java programlama dili ile kodlandığından JavaCC ayrıştırıcı kullanılmıştır.

1.12.1. JavaCC

JavaCC (Java Compiler Compiler), Java programlama dilinde yazılmış açık kaynaklı bir sözcüksel ve sözdizimsel ayrıştırıcı üretecidir (Enseling, 2000). JavaCC, EBNF benzeri notasyonuyla yazılmış bir dilbilgisi kullanır ve girdi verisini yukarıdan aşağıya doğru analiz edecek bir ayrıştırıcının kaynak kodunu üretir. Şekil 6'da JavaCC ile ayrıştırıcı üretimi gösterilmiştir.



Şekil 6. Ayrıştırıcı üretimi

1.12.2. Diğer Çözümleyici Üretme Araçları

Flex (Hızlı Sözcük Analizcisi Oluşturucu), 1987 yıllarında Vern Paxson tarafından C programlama dilinde yazılan sözcüksel analizörler oluşturan bir programdır (URL-1).

Lex, sözcük analizörleri üreten bir program olup aynı zamanda bir derleyici işlevi görür. Lex derleyicisi bir kaynak kod alır ve bu girişi giriş modellerine dönüştürür. Yaygın olarak YACC (Yet Another Compiler Compiler) ile birlikte kullanılır. Mike Lesk ve Eric Schmidt tarafından yazılmıştır (URL-2).

Quex sözcük analizörlerini oluşturmak için bir araçtır. Bir karakter akışını, birim sözcük adı verilen parça akışına dönüştüren bir programdır (URL-3).

YaCC (Yet Another Compiler-Compiler), Stephen C. Johnson tarafından Unix işletim sistemlerinde kullanılması amacıyla geliştirilen ve kaynak kodun sözdizimsel anlamını oluşturması yönüyle bir derleyicinin parçası olan programıdır (URL-4).

SableCC, derleyiciler, yorumlayıcılar ve diğer metin ayrıştırıcıları oluşturmak için geliştirilen bir programdır (URL-5). JavaCC'den farklı olarak LR(k) gramerleri üzerinde çalışır.

1.13. Değerlendirme Metodolojileri

1.13.1. Ziyaretçi Tasarım Deseni

Ziyaretçi tasarım deseni, nesne yapısını değiştirmeden hesaplamaların gerektirdiği işlemleri tanımlamaya ve sözdizim ağacının içerebileceği farklı işlemleri bir sınıf altında toplamaya olanak sağlayan bir mimarisel tekniktir (Jeanmart, Gueheneuc, Sahraoui & Habra, 2021). Kısacası hiyerarşik yapıya sahip soyut sözdizim ağacı verilerinin değerlendirilmesinde kullanılan bir yöntemdir. Nesne yapısını değiştirmeden nesneler üzerinde birbirinden bağımsız yeni işlemler tanımlanıp icra edilebilmektedir. Bu desenin, karmaşık işlemleri nesne hiyerarşisinden ayırarak, yeni işlemler eklemeyi kolaylaştırmada önemli bir görevi bulunmaktadır.

Belirli bir ziyaretçinin gerçekleştirebileceği ziyaret dizilerinin tümünü kapsayan ziyaretçi tasarım deseninde, CFG'den yararlanılmaktadır. CFG tarafından tanımlanan kural dizileri Ziyaretçi tasarım deseninin yapılandırılmasında önemli role sahiptir.

Ziyaretçi tasarım deseni, AST düğümleri arasında geçişler yaparak ilgili düğümlerde bulunan nesne verilerini değerlendirir. Her bir değerlendirme işlemi, bir nesne referansının parametre olarak verildiği visit() metodunun çağrımı ile başlatılır. Değerlendirme esnasında bu nesne referansının içerdiği diğer düğümlerin her birine ait accept() metodunun çağrımı yapılır. AST sınıflarında tanımlanan accept() metotlarının temel görevi, bir visit() metodundan gelen değerlendirme isteğini ilgili düğüm üzerinde çalışacak doğru visit() metoduna aktarmaktır. Başka bir ifadeyle, accept() metotları, icra kontrolünün bir visit() metodundan diğerine geçmesine aracılık ederler. Bu şekilde AST düğümlerinin tamamı değerlendirilinceye kadar visit() ile accept() metotları birbirini çağırmaya devam edecektir.

Tablo 6. Visitor arayüzü

```
interface Visitor {
public Object visit(Exp e);
public Object visit(Poly e);
public Object visit(Plus e);
public Object visit(Minus e);
public Object visit(Times e);
public Object visit(Power e);
public Object visit(Id e);
public Object visit(Num e);}
```

Tablo 6'da gösterilen Visitor tasarım deseni örneğinde visit() metodu ile soyut sözdizim ağacı üzerinde bulunan nesneler değerlendirilmektedir. Visitor tasarım desenini kullanan sınıflar, AST sınıfları üzerinde yapılacak değişikliklerden sonra yeniden derlenmeye ihtiyaç duymazlar.

1.13.2. Sözdizim Sınıflarına Metot Ekleme

Bu yöntemde her sözdizim sınıfına, sınıfın özelliklerine göre yeni metotlar eklenir. Metotların farklı isimlerde tanımlanabilmesi mümkün olsa da, Tablo 7'deki örnekte gösterildiği gibi genellikle eval ile isimlendirilirler. Sözdizim ağacının değerlendirilme işlemi düğümde var olan nesneye ait eval() metodunun çağırılması ile yapılır. Tablo7'de tanımlanan eval() metodu bir Table nesne referansını parametre alarak bir Ratio nesne referansını geri döndürür.

Tablo 7. AST sınıflarına eval() metodu eklenmesi

```
class Sin extends Exp {
  Exp e;
  public Sin(Exp a) {    e = a;  }
   public Ratio eval(Table t) {
        return new Ratio(Math.sin(e.eval(t).getDecimal().doubleValue()));  }
```

1.13.3. instanceof Operatörü

Java dilinde, bir nesnenin bir sınıfa, sınıfın alt sınıflarına ya da ara yüze ait bir örnek olup olmadığını kontrol etmek için instanceof operatörü kullanılır. Instanceof operatörü bir karşılaştırma operatörü (boolean tipinde değer döndüren) olup, bir nesnenin türetildiği sınıf hakkında bilgi edinilmesini sağlar. Tablo 8'de Plus türünden bir AST düğümü için instanceof operatörü ile değerlendirme yapan bir eval() metodu gösterilmiştir.

Tablo 8. Instanceof operatörü örneği

```
public Object eval(Plus e) {
   if (e.e1 instanceof Num)
      return eval(e.e2);
   else if (e.e2 instanceof Num)
      return eval(e.e1);
   return new Plus(eval(e.e1), eval(e.e2));
}
```

1.14. Gauss Eliminasyon Yöntemi

Gauss Eliminasyon yöntemi, 1800'lü yıllarda Gauss ve arkadaşları tarafından geliştirilmiştir. Lineer denklem sistemi ve değişkenlerin art arda eliminasyonunu gerçekleştiren bu yöntem, Gauss'un adıyla anılmasına rağmen birkaç yüzyıl önce Çin el yazmalarında Gauss Eliminasyon yöntemi ile üç bilinmeyenli üç denklem sisteminin nasıl çözülebileceğinin açıklandığı bulunmuştur (Kaptan, 2011).

Lineer denklem sistemlerini çözmek için kullanılan yöntemlerden biri olan Gauss Eliminasyon yöntemi, katsayı matrisi üzerinde gerçekleştirilen bir dizi işlemden oluşmaktadır. Yöntemin amacı, matrisi basitleştirilmiş bir forma dönüştürerek çözümü kolaylaştırmaktır.

$$2x + y = 10 \tag{17}$$

$$x + 3y = 8 \tag{18}$$

Denklem (17) ve (18)'de verilen 2 bilinmeyenli bir denklem sisteminin Gauss Eliminasyon yöntemini kullanarak çözüm aşamaları aşağıda gösterilmektedir.

- İlk aşamada verilen denklemler, bilinmeyen ifadelerin ve sonuçların katsayıları bir matriste genişletilmiş katsayılar matrisi şeklinde yazılır.

$$\left[\begin{array}{cc|c} 2 & 1 & 10 \\ 1 & 3 & 8 \end{array}\right]$$

- Elementer (temel) satır işlemleri ile (satırları istediğimiz bir sayı ile çarpma, istenilen iki satırın yerini değiştirme, bir satırı diğer bir satıra ekleme ya da çıkarma, bir satırı diğer bir satır ile çarpıp, satıra ekleme ya da çıkartma) genişletilmiş katsayı matrisi üst üçgen matris şekline dönüştürülür.

$$\left[\begin{array}{cc|c} 2 & 1 & 10 \\ 1 & 3 & 8 \end{array}\right] \longrightarrow \left[\begin{array}{cc|c} 2 & 1 & 10 \\ 0 & \frac{5}{2} & 3 \end{array}\right]$$

$$\frac{-1}{2}S_1 + S_2 \rightarrow S_2 \qquad (19)$$

Genişletilmiş katsayı matrisi kullanılarak elde edilen Denklem (19) sonucunda;

$$\frac{5}{2}y = 3 \rightarrow y = \frac{6}{5} \qquad (20)$$

$$2x + y = 10 \rightarrow 2x + \frac{6}{5} = 10 \; x = \frac{22}{5} \qquad (21)$$

Denklem (20) ve (21)'de verilen ifadeler elde edilir ve buradan (x, y) = ($\frac{22}{5}$, $\frac{6}{5}$) çözümü bulunur.

1.15. Doküman Formatlama

LaTeX, bilimsel ve teknik belgeler için yüksek kalitede sonuç üreten bir dokümantasyon sistemidir. LaTeX, her türlü belge için kullanılabilir ancak özellikle karmaşık yapıya, tekrarlayan biçimlendirmeye veya matematiksel ifadeler gibi gösterimlere sahip belgeler

için daha avantajlıdır. Matematiksel ifadelerin kolay ve basit bir şekilde düzenlenebilmesini sağlayan araçlarından biridir.

Bir kelime işlemcisi olmayan LaTeX;

- Dergilerin, makalelerin teknik raporların, kitapların vb. yapılanmasını,
- Tablolar, grafikler, şekiller içeren büyük belgelerin üzerinde kontrol sağlanmasını,
- Karmaşık matematiksel ifadelerin dizinlenmesini,
- Gelişmiş matematiksel dizilerin bulunmasını,
- Kaynakçaların ve dizinlerin otomatik oluşturulmasını

sağlamakla birlikte son derece yüksek standartlara sahip olan belgelerin üzerinde büyük bir kontrol sağlamaktadır.

1.16. Java Programlama Dili

Java programlama dili, 1900'lü yıllarda geliştirilmiş olup günümüzde en yaygın kullanılan programlama dillerinden biridir. Nesne yönelimli programlanması, platformdan bağımsız yani kaynak kodu platforma özgü koda değil, bytecode'a dönüştürüp JVM (Java Virtual Machine) yorumlayıcısı tarafından her çalıştırmada yorumlanma özelliğine sahip olması ve açık kaynak olması kullanım alanlarını artırmaktadır.

Java programlama dili, fonksiyonlar ve kütüphaneler aracılığı ile farklı platformlar ve programlama ortamlarına entegrasyon sağlamaktadır. Programlama ortamlarına entegrasyon için Java kütüphaneleri, Java API'leri, Java JNI, Java web servisleri gibi çeşitli yöntemler bulunmaktadır. Java kütüphaneleri, Java programlama dili dosya işlemleri, veri tabanı vb. işlemleri için çok çeşitli kütüphaneler sunmaktadır. Java API'leri, Java programlama dilinin sağladığı fonksiyonlar ve sınıflara erişim sağlayan arayüzlerdir. Java JNI (Java Native Interface), Java programlama dilinin sağladığı fonksiyonlar ve kütüphanelerin yerel kod ile kullanılmasını sağlamaktadır. Java web servisleri, XML ya da JSON gibi formatlarda veri alışverişi yapabilen web tabanlı uygulamalar olup, Java programlama dilinde yazılmış web uygulamaları diğer programlama ortamlarında kullanılabilir hale gelmektedir.

Bu tez çalışmasında sistemi tasarlamak için Java programlama dili, Java konsol uygulaması ile birlikte kullanılmış, komut satırı ve terminal üzerinde çalışılmıştır. Konsol uygulamaları genellikle eğitim, veri işleme ve sistem komutları amaçlı çalışmalar olup platformdan bağımsız, geliştirmesi nispeten daha kolay ve güçlüdür.

Bir konsol uygulaması için; Java kodu yazmak ve derlemek, Java Development Kit (JDK)'in yüklü olması, Eclipse, Visual Studio Code gibi IDE'ler ya da basit bit metin editörünün olması gerekmektedir.

Java programlama dili konsol ile etkileşime girmek için girdi verisini alma, metin ya da sonuç verisini ekrana yazdırma, konsolun işlevlerini kontrol etme gibi çeşitli sınıf ve paket imkânları sunmaktadır. Kullanıcıdan veri girdisi almak için Scanner, karakter tabanlı çıktı işlemleri gerçekleştirmek için PrintWriter, konsolun işlevlerini kontrol etmek için Console, Java uygulamasından başka bir işletim sistemi komutunu veya programını çalıştırmaya olanak sağlayan ProcessBuilder, büyük sayılarla işlemler yapmak için BigDecimal ve BigInteger sınıf örnekleri vardır. Java.math paketi büyük sayılar, tam sayılar, kesirler ve ondalık sayılar gibi matematiksel işlemler için kullanılmakta olan birçok sınıf, java.io paketi dosyadan veya konsoldan veri okuma ve yazma, ağ bağlantıları kurma gibi giriş çıkış (I/O) işlemleri için kullanılmakta olan birçok sınıf, java.util paketi ise Java'nın en temel ve en çok kullanılan paketlerinden olup tarih, saat, rastgele sayılar, koleksiyonlar, diziler ve diğer birçok temel veri yapısı ve yardımcı sınıf gibi birçok önemli sınıf içermektedir. java.util paketinde bulunan Hashtable sınıfı sözlüğe benzer şekilde çalışan bir anahtar değer veri yapısını temsil etmektedir. Bir nesne anahtar olarak kullanılır ve bu anahtar ona karşılık gelen bir değer ile eşleştirilir.

2. YAPILAN ÇALIŞMALAR

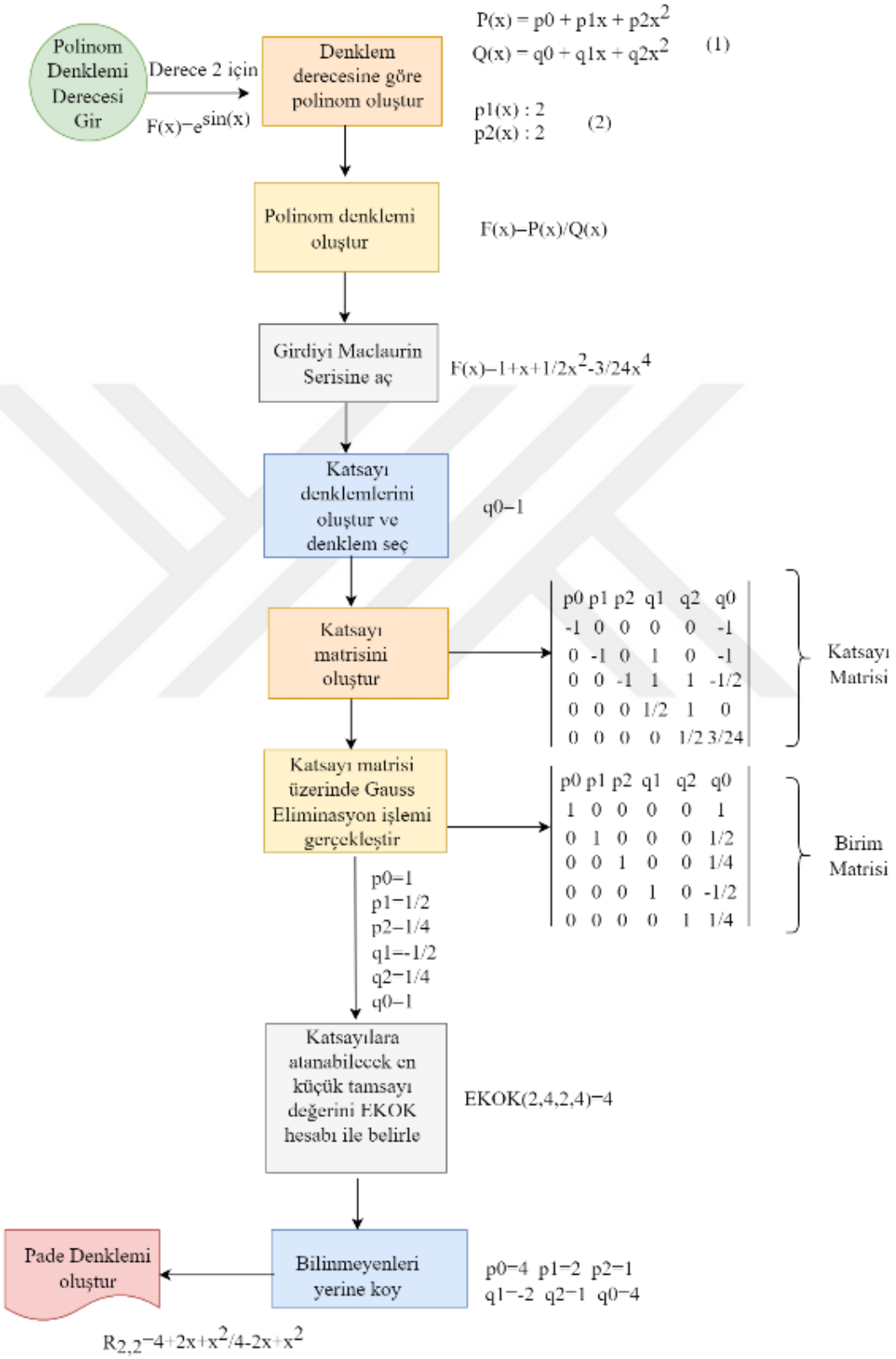
2.1. Giriş

Bu tez çalışmasında, Pade yaklaşımını simgesel matematik yöntemleriyle birlikte kullanarak rasyonel polinom ifadelerinin hesaplamasını ve işlem adımlarının PDF dokümana aktarımını yapabilen bir yazılım uygulaması geliştirilmiştir. Polinom ifadelerinin sözdizimini temsil etmek için EBNF benzeri notasyonuna sahip bir LL(k) grameri tasarlanmıştır. JavaCC formatı ile yapılan gramer tanımlamaları yardımıyla girdi verisi için ihtiyaç duyulan bir ayrıştırıcının kaynak kodu Java programlama dilinde üretilmektedir. Ayrıştırıcı tanımlamalarına polinom terimlerinin taşıdığı anlama uygun Java ifadeleri eklenerek bir soyut sözdizim ağacı oluşturulmuştur. Soyut sözdizim ağacı değerlendirmesi Ziyaretçi (Visitor) tasarım deseni, instanceof metodu ve sözdizim sınıflarına metot ekleme yöntemleri ile yapılabilmektedir.

Pade yaklaşım uygulaması sürecinin ilk adımı, ilgili fonksiyonların Taylor serisine açılımının yapılması ile başlamaktadır. Taylor serisi açılımında türev hesaplamaları ve fonksiyon değerlendirmeleri önemli bir yer tutmaktadır. Hem türev hesaplamaları hem de fonksiyon değerlendirmeleri için iki ayrı Ziyaretçi deseninin gerçeklemesi yapılmıştır. Ziyaretçi tasarım deseninde her bir ağaç düğümünün erişiminde accept() metodu, düğümün değerlendirilmesinde ise visit() metodu çağrılmaktadır. Fonksiyonların seriye açılımlarında genellikle sayı aralığının orta noktası kullanılmaktadır. Ancak hesaplamaları kolaylaştırmak için ilgili açılım koordinat düzleminin orijin noktasına göre yapılmıştır; yani Maclaurin serisine açılımdan yararlanılmıştır. Serinin sonlu sayıda terimine bağlı olarak bir doğrusal denklem sistemi elde edilmiştir. Bu denklem sistemindeki bilinmeyenler iki polinom ifadesindeki katsayılara karşılık gelmekte ve Gauss eliminasyon yöntemi gibi matris tabanlı yöntemlerle kolayca hesaplanabilmektedir.

Uygulamanın ihtiyaç duyduğu girdi verileri, standart girişten ya da bir dosya içerisinden okunmaktadır. Seriye açılım, denklem sisteminin düzenlenmesi ve çözüm yöntemlerinin uygulanması gibi çeşitli analiz ve sadeleştirme işlemleri, nesne yapılarıyla oluşturulan soyut sözdizim ağacını kullanmaktadır. Belirli bir matematiksel fonksiyonu temsil edecek rasyonel polinomları belirlemek için gerçekleştirilen hesaplamalar, işlem adımlarını içerecek biçimde hem bir terminal üzerinde gösterilirken hem de bir PDF dokümanına aktarılabilmektedir. PDF dosyalarının üretimi, polinom ifadelerine ait sözdizim

ağaçları üzerinden LaTex doküman formatlama diline yapılacak metinsel dönüşümlerle gerçekleştirilmiştir.



Şekil 7. Çalışmanın akış şeması

Çalışmanın akış şeması Şekil 7'de gösterilmiştir. Çalışma; iki farklı biçimde girdi olarak alınan matematiksel ifadeyi okuyup birim sözcük dizisine dönüştüren birim sözcük üretecinden, ifadenin sözdizim ağacını oluşturan bir ayrıştırıcıdan, değerlendiriciden, türev alıcıdan, Gauss Eliminasyon yöntemi çözücüsünden, ifade sadeleştiricisinden, ifade göstericisinden ve PDF olarak dışa aktarıcıdan oluşmaktadır.

Şekil 7'de gösterilen örnek bir $f(x) = e^{\sin(x)}$ fonksiyonu için belirtilen denklem derecesine yani derece 2'ye göre iki farklı biçimde girdi olacak şekilde polinom denklemleri oluşturulmuştur. Fonksiyonun grameri oluşturularak javaCC notasyonu yardımı ile bir ayrıştırıcı tanımlanmıştır. Ardından soyut sözdizim ağacı ile fonksiyonun nesne ağacı elde edilmiştir. Fonksiyonun rasyonel polinom değerlerinin belirlenebilmesi için türevinin alınması gerekmektedir. Türev alma işlemi, fonksiyonunun hesaplanmasını kolaylaştırmak amacıyla x=0 olacak şekilde Maclaurin serisine açılarak $f(x) = 1 + x + \frac{1}{2x^2} - \frac{3}{24x^3}$ fonksiyonu elde edilmiştir. Polinom denklemleri ve Maclaurin serisine göre açılan ifade $f(x) = \frac{P(x)}{Q(x)}$ eşitliğinde yerine konularak katsayı denklemleri elde edilmiştir. Buradan katsayı matrisi ve birim matrisi elde edilerek, denklemlerdeki bilinmeyen değerler bulunmuştur. Katsayılara atanabilecek en küçük katsayı değerleri EKOK hesabı ile belirlenerek Pade denklemi edilmiştir. Yapılan işlemler sonucunda elde edilen ifadeler, çözüm aşamaları ve Pade yaklaşım denklemi hem terminal ekranına hem de bir PDF dokümanına yazdırılmıştır.

2.2. Gramer Tasarımı

Bir $f(x) = sin(x)$ denkleminin Pade yaklaşımıyla [2,2] biçimindeki temsiline yönelik Tablo 9'da biçim 1 ve biçim 2 ile belirtilen ifadeler matematiksel girdi olarak kullanılmaktadır. Burada matematiksel ifadenin ne olduğu, yaklaşımın kaçıncı dereceden çözüleceği ve oluşturulan iki denklemin birbirine oranı şeklinde gösterimi verilmiştir.

Tablo 9. Girdi dosyası örneği

Biçim 1	**Biçim 2**
f(x) = sin(x) p1(x) : 2 p2(x) : 2 R(x) = p1(x)/p2(x)	f(x)=sin(x) p1(x)=p0+p1*x+p2*x^2 p2(x)=1+q1*x+q2*x^2 R(x) = p1(x)/p2(x)

Tablo 9'da verilen örnek girdi verisinin biçimine uygun bir LL(1) grameri Tablo 10'da gösterilmiştir. LL(1) gramerleri bir sözcük ile seçilebilen ve soldan özyinelemeli olmayan kurallardan oluşur ve kural genişletmeleri daima ileriye doğrudur. Tablo 10'da Pd ile başlangıç simgesi, Fx ile bir matematiksel fonksiyon, Px ve Qx ile sırasıyla bir rasyonel polinomun payı ve paydası, Rx ile rasyonel polinom, Fn, T ve P ile aritmetik işlemler ve F ile temel matematiksel fonksiyonlar temsil edilmiştir. Polinomların terim bazlı temsili ise Pn ve Qn kuralları ile yapılmaktadır Pade yaklaşımının pay (Px) ve paydasının (Qx) oranını, "Fn" aritmetik işlemleri, "T" çarpma ve bölme işlemlerini vb. her fonksiyon, ifade, değer, sayının tanımlaması yapılmıştır. Bir ifadeyi oluşturabilecek olan fonksiyon, işleç, parantez, sayı gibi belirteçler tırnak işareti içerisinde belirtilmiştir.

Tablo 10. Pade yaklaşımı girdi verisi için LL(1) grameri

Pd → Fx Px Qx Rx	F → "cos" "(" Fn ")"
Fx → "f(x)" "=" Fn	F → "tan" "(" Fn ")"
Px → "p(x)" (":" num | "=" Pn)	F → "ln" "(" Fn ")"
Qx → "q(x)" (":" num | "=" Qn)	F → "log" "(" Fn ")"
Rx →"p(x)" "/" "q(x)"	F → "exp" "(" Fn ")"
Fn →["-"|"+"] T (("+"|"-") T)*	Pn → Pt ("+" Pt)*
T →P (("*"|"/") P)*	Pt →"p" num (Xt)?
P → F ["^" P]	Xt → "x" ("^" num)?
F → id | num | "(" Fn ")"	Qn → Qt ("+" Qt)*
F → "sin" "(" Fn ")"	Qt → "q" num (Xt)?

2.3. Ayrıştırıcı Gerçekleme

2.3.1. Ayrıştırıcı (Parser)

JavaCC, bir parser yani ayrıştırıcı oluşturmak için kullanılan bir araçtır ve Java programlama dilinde ayrıştırma kodu üretmek için kullanılmaktadır. JavaCC bildirimleri, simgelerin listeleri ve dilbilgisi kuralları gibi bölümlerden oluşmaktadır. Tablo 6'da birim sözcükler için belirlenen simge listesi verilmiştir. Bu tabloya göre Pade yaklaşımında kullanılabilecek olan matematiksel ifadelerin tanımlanması yapılmıştır. Örneğin, bir toplama işlemi için PLUS, üs alma işlemi için POWER, eşittir ifadesi için EQUAL sözcükleri kullanılmaktadır.

Tablo 11. JavaCC ile birim sözcük (token) tanımlamaları

```
TOKEN : {
    < PLUS : "+" >
    | < MINUS : "-" >
    | < TIMES : "*" >
    | < DIVIDE : "/" >
    | < POWER : "^" >
    | < LPR : "(" >
    | < RPR : ")" >
    | < X : "x" >
    | < SIN: "sin" >
    | < COS: "cos" >
    | < TAN: "tan" >
    | < LN: "ln" >
    | < LOG: "log" >
    | < EULER: "exp" >
    | < ABS: "abs" >
    | < EQUAL: "=" >
    | < COLON: ":" >
    | < P: "p" >
    | < Q: "q" >
    | < FX: ("f(x)" | "F(x)") >
    | < PX: ("p(x)" | "P(x)") >
    | < QX: ("q(x)" | "Q(x)") >
    | < RX: ("r(x)" | "R(x)") >
    | < NUM : (["0"-"9"])+ >
}
SKIP: { " " | "\t" | "\n" | "\r" | "\r\n" }
```

$f(x)$ fonksiyonu için JavaCC notasyonuna göre Tablo 10'da gösterilen birim sözcük tanımlamalarına karşılık gelen sözcükler verilmiş olup, her bir birim sözcük bir işleç ile ifade edilmiştir. Bu bağlamda Tablo 11'e göre bazı matematiksel ifade örnekleri için JavaCC'nin oluşturduğu simge dizileri aşağıda gösterildiği gibidir.

- $f(x) = sin(x)$ fonksiyonu "FX, EQUAL, SIN, LPR, X, RPR"
- $f(x) = x^2 * sin(x)$ fonksiyonu "FX, EQUAL, X, POWER, NUM, TIMES, SIN, LPR, X, RPR"
- $f(x) = xln(x + 1)$ fonksiyonu "FX, EQUAL, X, TIMES, LN, LPR, X, PLUS, NUM, RPR"

Dilbilgisi tanımlamaları yapılırken operatör öncelikleri, terimlerin işaretleri gibi özellikler dikkate alınmaktadır. Toplama, çıkarma gibi operatörler düşük önceliğe, çarpma, bölme ve üs alma gibi operatörler ise yüksek önceliğe sahiptir. Tablo 12'de matematiksel ifadeler için kullanılan dilbilgisinin EBNF benzeri notasyonundaki tanımlamasının bir bölümü gösterilmektedir. Burada "Parser()" ayrıştırıcının başlangıç metodunu ifade ederken Tablo 11'de LL(1) gramerinde de gösterilen "Fx()", "Rx()", "Px()", Qx() ve ayrıca matematiksel işlemleri gerçekleyen tüm ifadelerin metotları bulunmaktadır.

Tablo 12. JavaCC ile dilbilgisi tanımlamaları

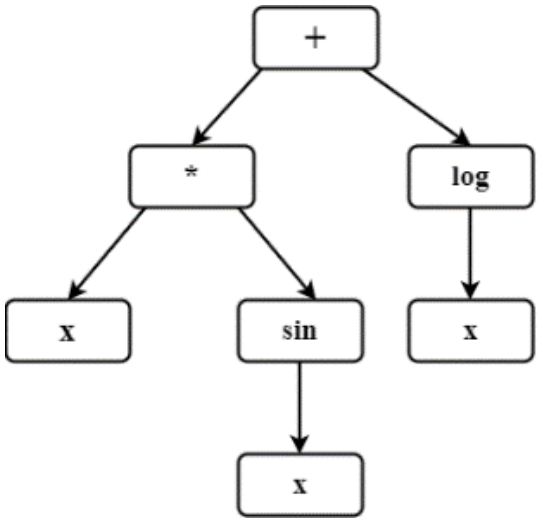
```
Pade Parser():
{ Pade pd; }
{
  pd=Pd() <EOF>   { return pd; }
}
Pade Pd():
{ Exp e1, e2, e3; }
{
   e1=Fx() e2=Px() e3=Qx() Rx()
   { return new Pade(e1, e2, e3); }
}
Exp Fx():
{ Exp e; }
   { <FX> <EQUAL> e=Fn()
      { return e; }      } }
void Rx():{ }
{<RX>  <EQUAL>  <PX>  <DIVIDE>
<QX>
Exp Px():
{ Token t; Exp e; }
{<PX> ( <COLON> t=<NUM>
{ e = PQ (Integer.parseInt(t.image),  "p");}
| <EQUAL> e=Pn() ) { return e; }}Exp T():
{ Exp e1, e2; int k = 0; }
{    e1=P() (
      <TIMES> e2=P() { e1 = new Times(e1,
e2); }
      | <DIVIDE> e2=P() { e1 = new Divide(e1,
e2); }
   )*   { return e1; }
}
……
```

2.3.2. Soyut Sözdizim Sınıfları (AST)

Soyut sözdizim ağacı süper sınıf tür yapısına sahip olup birbirine hiyerarşik bir şekilde bağlanan düğümlerden oluşur. Ağacı oluşturan düğümlerin her biri sözdizim sınıfından türetilir. Alt sınıflar, üst sınıflarından miras alarak ağacın düğümlerini oluşturur. Dilbilgisi kuralında bulunan işleç, operatör gibi her bir belirtecin AST sınıfı bulunur. AST sınıflarının isimleri kurallarının ifade ettiği işleçlerden üretilir. Tablo 13'te toplama operatörü ve sinüs fonksiyonu için oluşturulan AST sınıflarının örnekleri gösterilmiştir.

Tablo 13. Sözdizim sınıfları

```
class Plus extends Exp {
  Exp e1, e2;
  public Plus(Exp a, Exp b) {    e1 = a;    e2 = b;  }
  public Exp getE1() {    return e1;  }
  public Exp getE2() {    return e2;  }
  public Object accept(Visitor v) {    return v.visit(this);  }…. }
class Sin extends Exp {
  Exp e;
  public Sin(Exp a) {    e = a;  }
  public Exp getE1() {    return e;  }
  public Exp getE2() {    return e;  }
  public Object accept(Visitor v) {    return v.visit(this);  }….}
```



Şekil 8. $x * sin(x) + \log(x)$ fonksiyonu için sözdizim ağacı

Şekilde 8'de gösterilen $x * sin(x) + \log(x)$ matematiksel ifadesinin nesne ağacı Tablo 13'teki örnek AST sınıfları kullanılarak aşağıdaki gibi oluşturulabilir.

Exp exp = new Plus(new Times(new Var(),new Sin(new Var())),new Log(new Var()))

2.3.3. BigInteger ve BigDecimal Sınıfları

Basit rasyonel polinom ifadelerinde en geniş kapsamlı tamsayılar için long veri tipi, kayan noktalı (reel) sayılar için double veri tipi kullanılmaktadır. Long ve Double türünden nesneler ile sarmalanabilen bu veri tipleri Java programlama dilinde en geniş değer kümesine sahip olsalar da polinom katsayılarının belirlenmesi sürecinde daha büyük değerler kümesine ihtiyaç vardır.

Java çevrelerinde büyük sayılar üzerinde yüksek hassasiyet ile hesaplamalar yapılabilmek için BigInteger ve BigDecimal gibi iki sınıftan yararlanılır. BigInteger sınıfı, mevcut tüm ilkel veri türlerinin dışında kalan çok büyük tamsayı hesaplamalarını içeren matematiksel işlemler için kullanılırken BigDecimal sınıfı, aritmetik, ölçek işleme, yuvarlama, karşılaştırma, format dönüştürme gibi karmaşık işlemler için double sayılar üzerinde işlemler sağlar. Tablo 14'te Ratio nesnesinin kesirli olup olmadığını kontrol etmek için BigInteger sınıfının toBigIntegerExact() metodundan yararlanılmıştır.

Tablo 14. Ratio sınıfı ile seriye açılım örneği

```
public Exp maclaurinPoly(Exp  e, int pqn) {
   ……
   Poly p = new Poly(pqn+1);
     for (int i=0; i<=pqn; i++) {
        if (i>0) {
          fact *= i;
          e = (Exp)dv.visit(e);          }
           rt = ((Ratio)ev.visit(e));
     try {
           rt = new Ratio(rt.getR1().toBigIntegerExact());
        }
     catch (Exception exc) { }……
```

2.4. Rasyonel Polinom Hesaplama

Bir $f(x)$ fonksiyonunun rasyonel polinom eşdeğerinin belirlenmesinde seriye açılım, dolayısıyla simgesel türevinin hesaplanması ve *x=0* değeri ile değerlendirme yapılması gerekmektedir.

Örneğin;

- $f(x) = sin(x)$ fonksiyonunun Maclaurin serisine açılımı,

$$sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \ldots \quad (22)$$

Denklem (22) şeklinde olup, serinin [0, 1, 0, $-\frac{1}{3!}$, 0, $\frac{1}{5!}$, 0,$-\frac{1}{7!}$,0, $\frac{1}{9!}$, 0] listesinin nesne karşılığı olarak,

[new Ratio(0, new RFact(0)), new Ratio(1, new RFact(1)), new Ratio(0, new RFact(2)), new Ratio(-1,new RFact(3)), …]

- $f(x) = e^x$ fonksiyonunun Maclaurin serisine açılımı,

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^n}{n!} + \cdots \quad (23)$$

Denklem (23) şeklinde olup, serininin [1, 1, $\frac{1}{2!}$, , $\frac{1}{3!}$, $\frac{1}{4!}$, $\frac{1}{5!}$, ...] listesinin nesne karşılığı olarak,

[new Ratio(1, new RFact(1)), new Ratio(1, new RFact(1)), new Ratio(1, new RFact(2)), new Ratio(1,new RFact(3)), new Ratio(1,new RFact(4)), …]

- $f(x) = \ln(1 + x)$ fonksiyonunun Maclaurin serisine açılımı,

$$\ln(1 + x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \frac{x^5}{5} + (-1)^{n-1}\frac{x^n}{n} + \cdots \quad (24)$$

Denklem (24) şeklinde olup, serininin [1, -$\frac{1}{2}$, , $\frac{1}{3}$,- $\frac{1}{4}$, $\frac{1}{5}$, ...] listesinin nesne karşılığı olarak,

[new Ratio(0, new RFact(0)), new Ratio(1, new RFact(1)), new Ratio(-1, new RFact(2)), new Ratio(1,new RFact(3)), new Ratio(-1,new RFact(4)), …]

şeklinde üretilmektedir.

Maclaurin serisine göre açılım, türev alma ve değerlendirme aşamalarında, iki ayrı ziyaretçi tasarım deseni tanımlanmıştır. Tasarım deseni serinin ihtiyaç duyduğu terimleri bir nesne listesi içerisinden üretmektedir.

2.4.1. Türev Alıcı

Türev hesaplama işlemleri aşamasında, $\sum_{i=0}^{\infty} c_i x^i$ kuvvet serisi Taylor serisine açılır. Seriye açılım, işlemleri kolaylaştırmak amacıyla (25)'te gösterilen şekilde koordinat düzleminin orijin noktasına göre yani Maclaurin serisine göre açılır.

$$f(x) = c_0 + c_1 x + c_2 x^2 + \cdots = \frac{a_0 + a_1 x + a_2 x^2 + \cdots + a_l x^l}{b_0 + b_1 x + b_2 x^2 + \cdots + b_m x^m} \quad (25)$$

Bir fonksiyonun türevini hesaplarken, önce parametreler fonksiyona aktarılır. Ardından, fonksiyon ifadesi üzerinde türev işlemi uygulanır ve elde edilen sonuç değerlendirilerek hesaplama tamamlanır.

Visitor tasarım deseninin bir türü olan DeriveVisitor sınıfı AST ağacı üzerinde fonksiyon türevlerini gerçekleştirmektedir. Polinom ifadesi ve toplama işlemine ait DeriveVisitor sınıfı örneklemesi Tablo 15'te gösterilmiştir.

Tablo 15. Derive visitor örnekleri

```
class DeriveVisitor implements Visitor
{
public Object visit(Exp e) {
      return e.accept(this); }
  public Object visit(Poly e) {
      Poly ps = new Poly(e.ps.length);
      for (int i=0; i<e.ps.length; i++)
      {ps.add((Exp)e.ps[i].accept(this),i);}
        return ps; } . . .
public Object visit(Plus e) {
  if (e.e1 instanceof Num)
    return visit(e.e2);
  else if (e.e2 instanceof Num)
    return visit(e.e1);
     return new Plus((Exp)visit(e.e1),
(Exp)visit(e.e2)); }…
```

2.4.2. Değerlendirici

EvalVisitor sınıfı, Visitor sınıfından miras alarak matematiksel ifadeleri değerlendiren visit() metotlarını tanımlar. Bu tasarım deseninde her bir işlemin değerlendirilmesi ayrı bir visit() metodu ile yapılmıştır. Tablo 16'da polinom değerlendirmesi ve toplama işlemleri için oluşturulan metotlar gösterilmiştir.

Tablo 16. EvalVisitor sınıfındaki bazı visit() metotları

```
class EvalVisitor implements Visitor {
  Table t;
  public EvalVisitor(Table t){
   this.t = t;  }
public Object visit(Exp e) {
```

Tablo 16'nın devamı

```
  return e.accept(this);  }
 public Object visit(Poly e) {
  BigDecimal big = BigDecimal.ZERO;
  for (int i=0; i<e.ps.length; i++) {
   big = big.add(((Ratio)e.ps[i].accept(this)).getDecimal()) ;    }
  return new Ratio(big);  }
 public Object visit(Plus e) {
        Ratio a = (Ratio)e.e1.accept(this);
        Ratio b = (Ratio)e.e2.accept(this);
  return new Ratio(a.addR(b));
 }…..
```

2.4.3. Sadeleştirici

Matematiksel fonksiyonlar üzerinde türev benzeri hesaplamalar yapılırken, fonksiyonun yapısına bağlı olarak hesaplama sürecini karmaşıklaştırabilen birçok terim ortaya çıkabilmekte ve fonksiyonların okunabilirliğini zayıflatabilmektedir. Sadeleştirici ile bu gibi durumlar çözümlenerek değerlendirme performansı yükseltilir.

Sadeleştirme işlemleri yapılırken aşağıda verilen bazı kurallar takip edilerek işlemler gerçekleştirilir.

- Sıfırla çarpma işlemlerinde sonuç sıfır olur.
- Sıfırla toplama ya da çıkarma işlemlerinde sıfır olan ifadeler dikkate alınmaz.
- Çarpma işleminde bir sayı 1 ise, sonuç diğer sayıya eşittir.
- Bölme işlemlerinde bölen sayı 1 ise, sonuç diğer sayıya eşittir.
- Üs alma işlemlerinde, taban sayısı sıfır ise sonuç sıfır; üs 0 ise sonuç 1, üs 1 ise sonuç tabandır.

Tablo 17. İfadelerin sadeleştirilmesi

İfade	İfadenin Nesne Ağacı	Sadeleştirilmiş İfade	Sadeleştirilmiş İfadenin Nesne Ağacı
*x*0*	*new Times(new Var(),new Num(0))*	*0*	*new Num (0)*
(x+0)	*new Plus(new Var(),new Num(0))*	*x*	*new Var()*
(x- 0)	*new Minus(New Var(), new Num(0))*	*x*	*new Var()*
*(x*1)*	*new Times(new Var(),new Num(1))*	*x*	*new Var()*
(x/1)	*new Divide(new Var(),new Num(1))*		
0^1	*new Power(new Num(0),new Num(1))*	*0*	*new Num (0)*
x^0	*new Power(new Var(),new Num(0))*	*1*	*new Num (1)*
x^1	*new Power(new Var(),new Num(1))*	*x*	*new Var()*

Tablo 17'de sadeleştirilmesi gereken ifadeler ve sadeleştirme kurallarının bu ifadelere uygulanmış şekli gösterilmiştir. Bu çeşit işlemler ile SimplifyVisitor sınıfı içerisinden ilgilenilmiştir. Exp, Poly ve Plus gibi farklı türden ifadeler üzerinde tanımlanan visit() metotları ile sadeleştirme yapılmıştır.

Tablo 18. Sadeleştirme işlemleri

```
class SimplifyVisitor implements  Visitor {
  public Object visit(Exp e) {   return e.accept(this);  }
  public Object visit(Poly e) {
```

Tablo 18'in devamı

```
    Poly ps = new Poly(e.ps.length);
        for (int i=0; i<e.ps.length; i++)
       { ps.add((Exp)e.ps[i].accept(this), i);    }
        return ps;  }
  public Object visit(Plus e) {
          Exp a = (Exp)e.e1.accept(this);
          Exp b = (Exp)e.e2.accept(this);
          if (a instanceof Num)  {
           if (((Num)a).n.compareTo(BigInteger.ZERO)==0)
                  a = null;  }
          if (b instanceof Num)  {
           if (((Num)b).n.compareTo(BigInteger.ZERO)==0)
                  b = null;         }
          if (a==null && b==null)
           return new Num(0);
 ……..
```

Tablo 18'de gösterilen SimplifyVisitor ziyaretçi sınıfı örneğinde, Poly ve Plus ifadelerinin nasıl sadeleştirileceği gösterilmiştir. Sıfır olan katsayılar silinerek ve gereksiz toplama işlemlerini kaldırarak ifadeler daha basit hale getirilmektedir.

2.4.4. Gauss Eliminasyon Yöntemi ile Çözüm Aşamaları

Bir $f(x) = cos(x)$ fonksiyonunun [2,2] Pade yaklaşımını bulabilmek için Maclaurin serisine göre $x^{2+2} = x^4$ terimine kadar açılımının yapılması gerekmektedir

(5) ve (6)'da verilen denklemlere göre,

$$P_2(x) = p_0 + p_1 x + p_2 x^2 \tag{5}$$

$$Q_2(x) = 1 + q_1 x + q_2 x^2 \tag{6}$$

$p_0,\ p_1, p_2, q_1, q_2$ olmak üzere 5 adet bilinmeyen ifade bulunduğu için denklem (25)'te de gösterildiği gibi $f(x) = cos(x)$ ifadesinin Maclaurin serisine göre açılımı 5. terime kadar olmalıdır.

$$f(x) = cos(0) + cos'^{(0)} x + \frac{cos''(0)x^2}{2!} + \frac{cos'''(0)x^3}{3!} + \frac{cos^{(4)}(0)x^4}{4!} + \cdots \tag{26}$$

$f(x)Q_m(x) - P_n(x) = 0$ ifadesinde verilen denklemler yerine yazılırsa;

$$\left(1 - \frac{1}{2}x^2 + \frac{1}{24}x^4\right)(q_0 + q_1 x + q_2 x^2) - (p_0 + p_1 x + p_2 x^2) = 0 \tag{27}$$

Denklem (27) ifadesi elde edilir. Buradan x katsayılarına göre ifadeler düzenlenirse,

$$(-p_0 + q_0) + x(-p_1 + q_1) + x^2\left(-p_2 - \frac{q_0}{2} + q_2\right) + x^3\left(-\frac{q_1}{2}\right) + x^4\left(\frac{q_0}{24} - \frac{q_2}{2}\right) - x^5\left(\frac{q_1}{24}\right) + x^6\left(\frac{q_2}{24}\right) = 0 \tag{28}$$

(28)'deki polinom denklemi ortaya çıkar. Bu denklem yardımıyla aşağıda gösterilen (29) doğrusal denklem sistemi elde edilir.

$$\begin{aligned} -p_0 + q_0 &= 0 \\ -p_1 + q_1 &= 0 \\ -p_2 - \frac{q_0}{2} + q_2 &= 0 \end{aligned} \tag{29}$$

$$-\frac{q_1}{2} = 0$$

$$\frac{q_0}{24} - \frac{q_2}{2} = 0$$

$$\frac{q_1}{24} = 0$$

$$\frac{q_2}{24} = 0$$

Polinom ifadelerinde 5 bilinmeyen olduğundan, yukarıdaki denklem sistemindeki ilk 5 denklemin kullanılması yeterlidir. Gauss Eliminasyon yöntemi ile çözüm için aşağıdaki genişletilmiş (artırılmış) katsayı matrisi düzenlenir.

$$\begin{bmatrix} p_0 & p_1 & p_2 & q_1 & q_2 & q_0 \\ -1 & 0 & 0 & 0 & 0 & -1 \\ 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & \frac{1}{2} \\ 0 & 0 & 0 & \frac{-1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{-1}{2} & \frac{-1}{24} \end{bmatrix}$$

Burada çeşitli satır işlemleri uygulanarak aşağıdaki birim matrise dönüşüm yapılır.

$$\left[\begin{array}{ccccc|c} 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & \frac{-5}{12} \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \frac{1}{12} \end{array}\right]$$

Katsayı değerleri,

$$p_0 = 1 \qquad (30)$$

$$p_1 = 0$$

$$p_2 = -\frac{5}{12}$$
$$q_0 = 1$$
$$q_1 = 0$$
$$q_2 = \frac{1}{12}$$

bulunur. $p_0,\ p_1, p_2, q_0, q_1, q_2$ bilinmeyen değerlerini tamsayılar yapmak için EKOK hesabı ile

$$p_0 = 12 \tag{31}$$
$$p_1 = 0$$
$$p_2 = -5$$
$$q_0 = 12$$
$$q_1 = 0$$
$$q_2 = 1$$

olacaktır. Bu değerleri polinom ifadelerinde yerine koyarak

$$R_{2,2} = \frac{p_0+p_1x+p_2x^2}{q_0+q_1x+q_2x^2} = \frac{12+0x-5x^2}{12+0x+x^2} \tag{32}$$

rasyonel polinomu belirlenir. Bazı rasyonel polinom hesaplamalarında, katsayı matrisinin ana köşegeni üzerindeki elemanlarından biri ya da daha fazlası 0 olabildiğinden, Gauss eliminasyonu ile birim matrise dönüşüm yapılamayabilmektedir. Genel olarak, katsayı matrisinin çözüm üretmeyen veya sonsuz sayıda çözüm üreten iki özel durumu ortaya çıkabilmektedir. Sonsuz sayıda çözümün bulunduğu durumlarda, satırların yer değişimi ve bağımlı değişkenlerin kullanımı ile polinom katsayıları için çoğunluğu 0'dan farklı olabilecek biçimde değerler belirlenir. Diğer durumda rasyonel polinom hesaplanamamaktadır.

Tablo 19. Gauss eliminasyon yöntemi

```
public Exp[][] gaussMatrix(Exp[][] ns, int r, int r2) {
    int s = ns.length;
    //r. satiri kullanarak r2. satiri indirge
     Ratio factor = ((Ratio)ns[r2][r]).divideR((Ratio)ns[r][r]);
     for (int j=r; j<=s; j++) {
       ns[r2][j]= (Ratio)ns[r2][j]).subtractR(factor.multiplyR((Ratio)ns[r][j])).Simplify();
       }    return ns;  }
```

$f(x) = cos(x)$ fonksiyonunda [2,2] Pade yaklaşımı $P_n(x) = p_0 + p_1x + p_2x^2$, $Q_m(x) = 1 + q_1x + q_2x^2$ denklemlerinden elde edilen lineer denklem sisteminin çözümü için Tablo 19'da verilen kod Gauss Eliminasyon algoritmasının bir parçasını uygulamaktadır. Bir matrisi üçgensel formuna dönüştürmek için kullanılan gaussMatrix() metodu, Exp[][] türünde bir matris (ns), bir başlangıç satırı (r) ve bir hedef satır (r2) almaktadır. Algoritma, r satırı ile r2 satırı üzerinde indirgeme işlemi yaparak matrisin belirli bir satırını değiştirmektedir. İlk olarak, r2 satırını r satırına bölerek bir oran faktörü hesaplanıp sonrasında r2 satırının her bir öğesinden, bu faktörle çarpılmış ve r satırındaki aynı sütundaki öğeler çıkarılarak indirgenmiş bir matris oluşturulmaktadır.

Tablo 20'de Gauss Eliminasyon işlemleri aşamalarından katsayı matrisini oluşturan kod parçası gösterilmiştir. (dizi boyutu) x (dizi boyutu-1) boyutunda bir değişken oluşturulup her sütun indeksini sıfırlayarak başlatır. Bu şekilde Exp türünden bir listenin içindeki katsayıları alınarak katsayı matrisi oluşturulur. Özellikle, bu yöntemle birlikte bir sözdizim ağacı şeklindeki liste elemanlarından belirli türdeki katsayılar ayırt edilerek matris şekline getirilir.

Tablo 20. Katsayı matrisi

```
public Exp[][] coeffMatrix(Exp[] es) {
 int s = es.length;
 Exp[][] ns = new Exp[s][s-1];
 for (int i=0; i<s; i++) {
  int j = 0;      Exp e = es[i];
```

Tablo 20'nin devamı

```
while (true) {
        e = e.getE2();
        if (e instanceof Times) {
            ns[i][j++] = e.getE2();   break;          }
         ns[i][j++] = e.getE1().getE2();       }
   }
      return ns;  }
```

Parametre olarak alınan bir kare matrisin birim matrise dönüştürülmesi Tablo 21'de gösterilmiştir. Matrisin köşegen elemanları 1 yapılarak birim matrise dönüştürülür.

Tablo 21. Birim matris

```
public Exp[][] identityMatrix(Exp[][] ns) {
 int s = ns.length;
 //Kosegen elemanlarini 1 yap
 for (int i=0; i<s; i++) {
   ns[i][s]= ((Ratio)ns[i][s]).divideR((Ratio)ns[i][i]).Simplify();
   ns[i][i] = new Ratio(1);
 }
 return ns;
}
```

Bir matrisin satır indirgeme işlemlerinin gerçeklemesi Tablo 22'de gösterilmiştir. Burada ilk döngü matrisin alt üçgenini sıfırlamak için yukarıya doğru indirgeme, ikinci döngü üst üçgeni sıfırlamak için aşağıya doğru indirgeme işlemlerini yapmaktadır. Matrisin üst üçgensel forma dönüştürülmesi amacıyla Gauss Eliminasyon işlemleri gerçekleştirilmektedir.

Tablo 22. Gauss Eliminasyon yöntemi ile aşağı ve yukarı indirgeme metotları

```
Exp[][] e6 = ns;
    //Asagiya dogru indirge
    for (int i=0; i<e6.length;  i++) {
      for (int j=i+1; j<e6.length;  j++) {
        e6 = fd.gaussMatrix(e6,  i, j);
        System.out.println(fd.showMatrix(e6));       }    }
        //Yukariya dogru indirge
    for (int i=e6.length-1;  i>0; i--) {
      for (int j=i-1; j>=0; j--) {
        e6 = fd.gaussMatrix(e6,  i, j);
        System.out.println(fd.showMatrix(e6));     }   }
```

3. BULGULAR VE TARTIŞMA

3.1. Yorumlama Aşaması

Pade yaklaşımında hesaplanan bir polinom fonksiyonun yorumlanması ayrıştırıcı gerçekleme, seriye açma, türev alma, sadeleştirme ve ekrana yazdırma aşamalarının uyumlu bir şekilde birleşmesiyle oluşmaktadır. Çalışmada ayrıştırıcı gerçekleme için pade_exp.jj, türev alma için DeriveVisitor.java, sadeleştirme işlemleri için SimplifyVisitor.java, ekrana yazdırma için PrintVisitor.java ve seriye açma işlemleri için PadeApp,java, PadeText.java sınıfları kullanılmaktadır. Tablo 23'te verilen PadeTex sınıfında standart girişten veya bir txt dosyasından polinom biçimleri okunmakta ve genTex() metoduyla çözüm işlemleri yapılmaktadır.

Tablo 23. PadeTex sınıfı main() metodu

```
Public static void main(String[] args) throws ParseException, FileNotFoundException {
    //PadeExpParser pr = new PadeExpParser(System.in);  //Standart giristen oku
    PadeExpParser pr =
        new PadeExpParser(new FileInputStream(new  File("pade1a.txt")));
    Pade pd = pr.Parser();
    new PadeTex().genTex(pd);
}
```

AST.java'yı oluşturan yorumlayıcı sözdizim sınıfları, pade_exp.jj parserını oluşturan JavaCC aynı dizinde bulunmaktadır. Bir konsol üzerinden yorumlayıcı üretimi;

$> javacc pade_exp.jj

$> javac *.java

$> java PadeTex şeklinde yapılmaktadır.

3.2. PDF'e Aktarma

LaTeX matematiksel ifadeleri kolay ve basit bir şekilde yazan araçlarından biridir. Çalışmada kullanılan polinom denklemlerin daha anlaşılır bir şekilde ifade edilmesi açısından LaTeX kullanılmıştır. Bu bağlamda PDF dosyalarının üretimi, rasyonel polinom

ifadelerine ait sözdizim ağaçları üzerinden LaTex doküman formatlama diline yapılacak dönüşümlerle gerçekleştirilir. Geliştirilen uygulamalarda, üretilen LaTeX girdilerinin yorumlanabilmesi amacıyla bir LaTeX kütüphanesi olan MiKTeX kütüphanesi kullanılmaktadır. Polinom denklemlerinin işlem adımlarını içeren şekilde hem bir terminal üzerinde gösterilebilir hem de bir PDF dokümanına aktarım yapılabilmektedir. Tablo 24'te uygulamada kullanılan LaTeX formatı dönüşümünün bir kısmı gösterilmiştir. toLatex() metodu soyut (abstract) sınıftan türetilmektedir ve her söz dizim sınıfı için o sınıfın özelliklerine göre türetilmektedir.

Tablo 24. Latex formatı

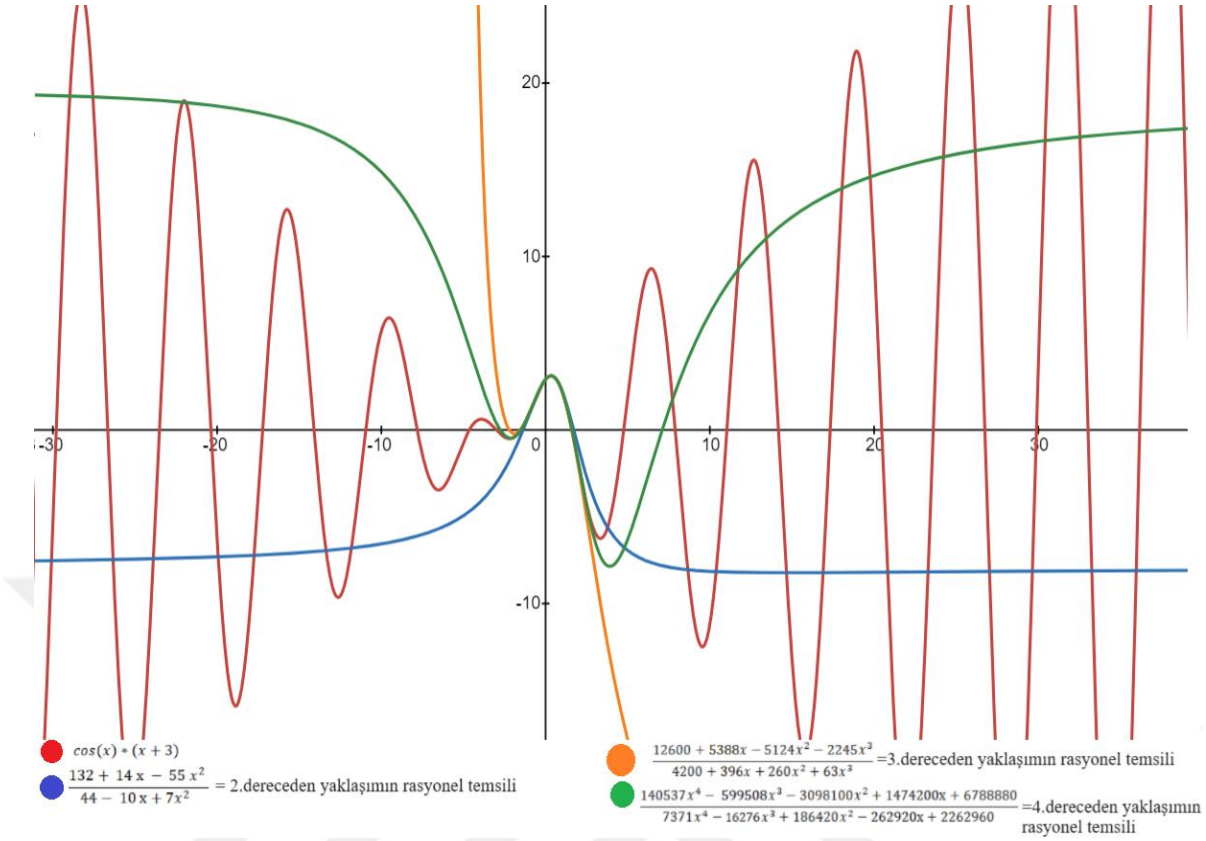
```
public void genTex(Pade pd) {
    Tex_Create();
    PutTextLn("\\section{Pade Polinomunun Hesaplanmasi}");
    PutTextLn("\\subsection{Polinomlar}");
    System.out.println("fx:  " + pd.fx.toClass());
    int px_s = polySize(pd.px);
    int qx_s = polySize(pd.qx);
    System.out.println("polinom dereceleri: " + px_s + " + " + qx_s);
    PutTextLn("$\\displaystyle P_{" + px_s + "}(x)=" + pd.px.toLatex(false)+"$\\\\");
    PutTextLn("$\\displaystyle Q_{" + qx_s + "}(x)=" + pd.qx.toLatex(false)+"$\\");

……
```

3.3. Rasyonel Polinom Örnekleri

$cos(x) * (x + 3)$, $\frac{sin(x)+5x}{3}$, $e^{\sin(x)}$, $x + \sin(x)$ gibi bazı karmaşık fonksiyonlar için polinom dereceleri 2,3,4,5,… olan örnekler ve rasyonel ifade temsilleri Tablo 25'te gösterilmektedir.

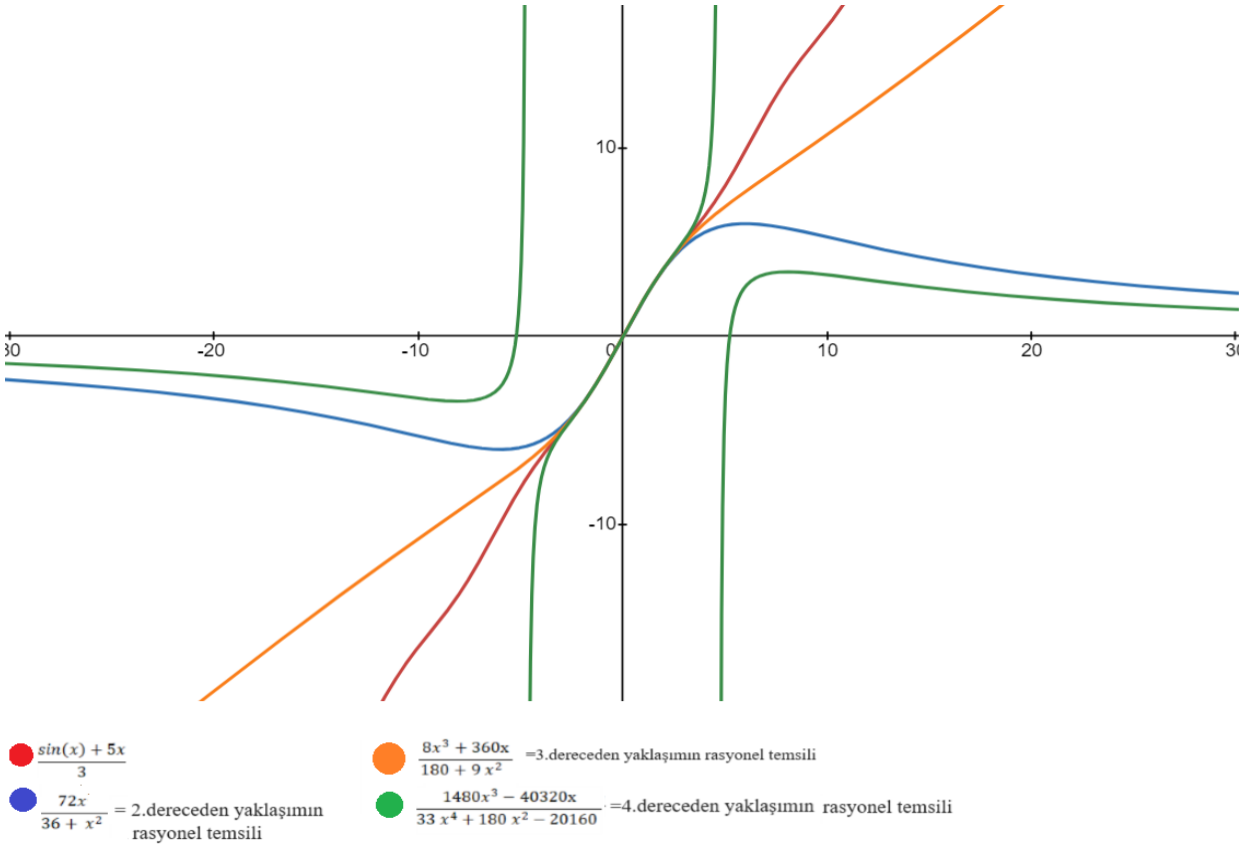
Tablo 25. Karmaşık fonksiyonların rasyonel polinom temsilleri

Fonksiyon	**Derecesi**	**Rasyonel Temsili $(P_n(x)/Q_n(x))$**
$cos(x) * (x+3)$	[2,2]	$\frac{132 + 14\,\mathrm{x} - 55\,x^2}{44 - 10\,\mathrm{x} + 7x^2}$
$cos(x) * (x+3)$	[3,3]	$\frac{12600 + 5388x - 5124x^2 - 2245x^3}{4200 + 396x + 260x^2 + 63x^3}$
$cos(x) * (x+3)$	[4,4]	$\frac{140537x^4 - 599508x^3 - 3098100x^2 + 1474200\mathrm{x} + 6788880}{7371x^4 - 16276x^3 + 186420x^2 - 262920\mathrm{x} + 2262960}$
$\frac{sin(x) + 5x}{3}$	[2,2]	$\frac{72\mathrm{x}}{36 + x^2}$
$\frac{sin(x) + 5x}{3}$	[3,3]	$\frac{8x^3 + 360\mathrm{x}}{180 + 9\,x^2}$
$\frac{sin(x) + 5x}{3}$	[4,4]	$\frac{1480x^3 - 40320\mathrm{x}}{33\,x^4 + 180\,x^2 - 20160}$
$e^{\sin(x)}$	[2,2]	$\frac{4 + 2\,\mathrm{x} + x^2}{4 - 2\,\mathrm{x} + x^2}$
$e^{\sin(x)}$	[3,3]	$\frac{120 + 60\,\mathrm{x} + 28\,x^2 - x^3}{120 - 60\,\mathrm{x} + 28\,x^2 + x^3}$
$e^{\sin(x)}$	[4,4]	$\frac{240 + 120\,\mathrm{x} - 140\,x^2 - 100\,x^3 - 49\,x^4}{240 - 120\,\mathrm{x} - 140\,x^2 + 100\,x^3 - 49\,x^4}$
$sin(x) * (x)$	[2,2],[3,3]	$\frac{6x^2}{6 + x^2}$
$sin(x) * (x)$	[4,4]	$\frac{5880x^2 - 620x^4}{5880 + 360x^2 + 11x^4}$



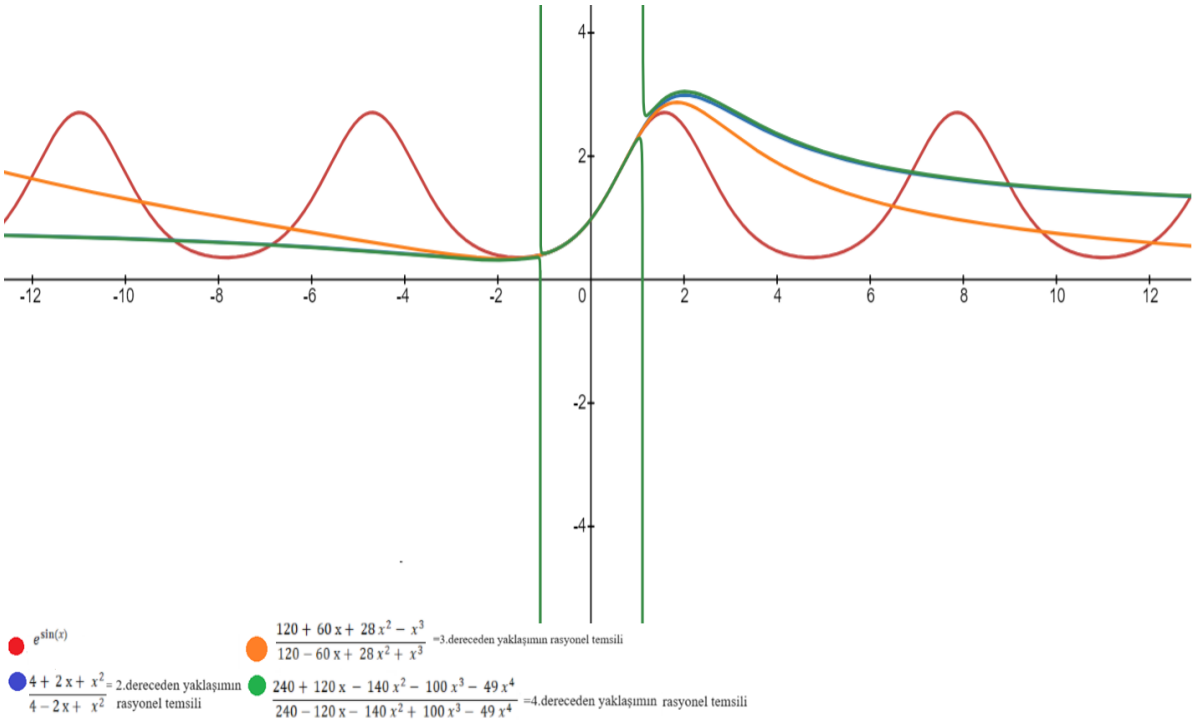
Şekil 9. $f(x) = cos(x) * (x + 3)$ polinom ifadesi ve rasyonel temsilleri

$cos(x) * (x + 3)$ polinom ifadesinin 2. , 3. ve 4.dereceden Pade yaklaşımının rasyonel temsili Şekil 9'da gösterilmiştir. Burada yaklaşım derecesi ne kadar artarsa polinom ifadesine yaklaşım o kadar yakın olmaktadır.



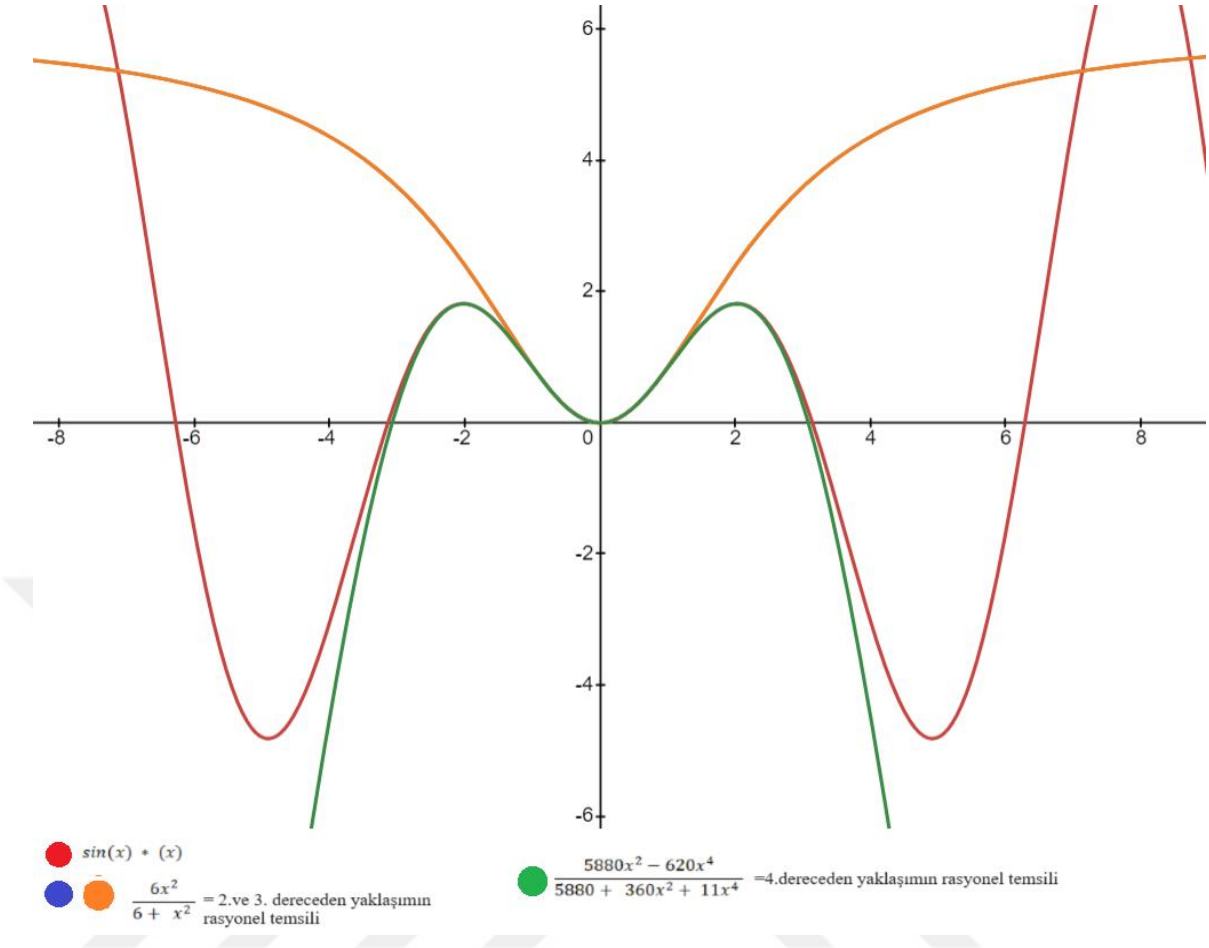
Şekil 10. $f(x) = \frac{sin(x)+5x}{3}$ polinom ifadesi ve rasyonel temsilleri

Şekil 10'da $\frac{sin(x)+5x}{3}$ polinom ifadesinin 2. , 3. ve 4.dereceden Pade yaklaşımının rasyonel temsili gösterilmiştir.



Şekil 11. $f(x) = e^{\sin(x)}$ polinom ifadesi ve rasyonel temsilleri

$e^{\sin(x)}$ polinom ifadesinin 2. , 3. ve 4.dereceden Pade yaklaşımının rasyonel temsili Şekil 11'de gösterilmiştir.



Şekil 12 . $f(x) = sin(x) * (x)$ polinom ifadesi ve rasyonel temsilleri

$sin(x) * (x)$ polinom ifadesinin 2. , 3. ve 4.dereceden Pade yaklaşımının rasyonel temsili Şekil 12'de gösterilmiştir.

Durum çalışması 1: $f(x)=cos(x) * (x + 3)$ polinom ifadesinin 2. dereceden rasyonel polinomlar ile temsili;

$$R_{2,2}(x) = \frac{p_2(x)}{q_2(x)} = \frac{132 + 14\,x - 55\,x^2}{44 - 10\,x + 7x^2} \quad (33)$$

Denklem (33)'te gösterilen şekildedir. $f(x)=cos(x) * (x + 3)$ fonksiyonu için elde edilen terminal çıktısı Şekil 13'te, Latex sonucu PDF çıktısı Şekil 14'te gösterilmiştir.

```
fx: Times(Cos(Id(x)),Plus(Id(x),Num(3)))
polinom dereceleri: 2 + 2

fx fonksiyonu Maclaurin serisine açılıyor.
fx_series: Poly((Ratio(RDec(3),RDec(1)),0)+(Ratio(RDec(1),RDec(1)),1)+(Ratio(RDec(-3),RDec(2)),2)+(Ratio(RDec(-3),RDec(6)),3)+(Ratio(RDec(3),RDec(24)),4))

Polinom terimlerinin katsayıları ve dereceleri
fx: [(3,0),(1,1),((-3/2),2),((-3/6),3),(3/24,4)]
px: [(p0,0),(p1,1),(p2,2)]
qx: [(q0,0),(q1,1),(q2,2)]


Bunların en küçük ortak katı hesaplanır.
p0=3
p1=7/22
p2=(-5/4)
q1=(-5/22)
q2=7/44
EKOK(22,4,22,44) = 44

Tüm katsayılar ekok ile çarpılır ve tabloya yazılır.

Bu değerler tablodan okunarak px ve qx polinomlarında yerlerine yazılır.
px: [(132,0),(14,1),(((-55)),2)]
qx: [(44,0),(((-10)),1),(7,2)]
```

Şekil 13. $f(x)= cos(x) * (x + 3)$ polinom ifadesinin derece 2 için terminal çıktısı

1 Pade Polinomunun Hesaplanması

1.1 Polinomlar

$P_2(x) = p_0 + p_1x + p_2x^2$
$Q_2(x) = q_0 + q_1x + q_2x^2$

1.2 Maclaurin serisi

$f(x) = cos(x) * x + 3$
$f(x) = 3 + x - \frac{3}{2}x^2 - \frac{3}{6}x^3 + \frac{3}{24}x^4$

1.3 Polinon denklemi

$P(x) - Q(x) * f(x) = 0$
$[p_0 + q_0 * (-3)] + [p_1 + q_0 * (-1) + q_1 * (-3)]x + [p_2 + q_0 * \frac{3}{2} + q_1 * (-1) + q_2 * (-3)]x^2 + [q_0 * \frac{3}{6} + q_1 * \frac{3}{2} + q_2 * (-1)]x^3 +$
$[q_0 * (\frac{-3}{24}) + q_1 * \frac{3}{6} + q_2 * \frac{3}{2}]x^4 + [q_1 * (\frac{-3}{24}) + q_2 * \frac{3}{6}]x^5 + q_2 * (\frac{-3}{24})x^6 = 0$

2.3 Katsayılar icin Tamsayı Değerleri

Katsayılara atanabilecek en küçük tamsayı değeri EKOK hesabi ile belirlenir.
EKOK(22,4,22,44)=44
$p_0 = 132$
$p_1 = 14$
$p_2 = -55$
$q_1 = -10$
$q_2 = 7$
$q_0 = 44$

2.4 Pade Polinomu

$R_{2,2} = \frac{132 + 14x - 55x^2}{44 - 10x + 7x^2}$

Şekil 14. $f(x)= cos(x) * (x + 3)$ polinom ifadesinin derece 2 için PDF çıktısı

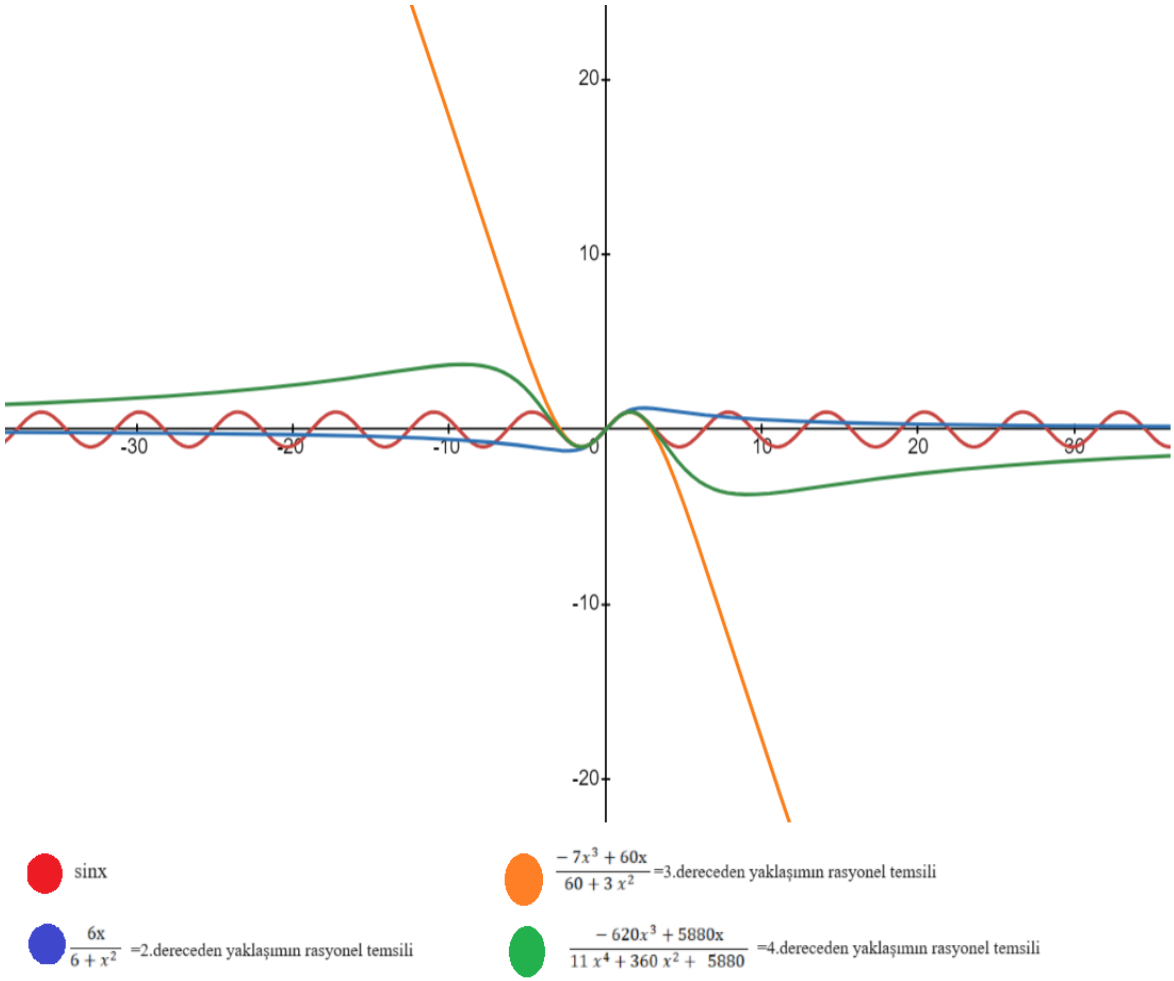
$sin(x)$, $\cos(x)$, $\tan(x)$, e^{x}, $\log(x)$ gibi bazı temel fonksiyonlar için polinom dereceleri 2,3,4,5 olan örnekler ve rasyonel ifade temsilleri Tablo 26'da gösterilmektedir.

Tablo 26. Temel fonksiyonların rasyonel polinom temsilleri

Fonksiyon	**Derecesi**	**Rasyonel Temsili** $(\frac{P_n(x)}{Q_n(x)})$
$\sin(x)$	[2,2]	$\frac{6x}{6+x^2}$
$\sin(x)$	[3,3]	$\frac{-7x^3+60x}{60+3x^2}$
$\sin(x)$	[4,4]	$\frac{-620x^3+5880x}{11x^4+360x^2+5880}$
$cos(x)$	[2,2]	$\frac{-5x^2+12}{x^2+12}$
$cos(x)$	[3,3]	$\frac{-5x^2+12}{x^2+12}$
$cos(x)$	[4,4]	$\frac{313x^4-6900x^2+15120}{13x^4+660x^2+15120}$
$\tan(x)$	[2,2]	$-\frac{3x}{x^2-3}$
$\tan(x)$	[3,3]	$\frac{x^3+3x}{3}$
$\tan(x)$	[4,4]	$\frac{105x-10x^3}{x^4-45x^2+105}$
e^{x}	[3,3]	$\frac{-x^3-12x^2-60x-120}{-x^3-12x^2+60x-120}$

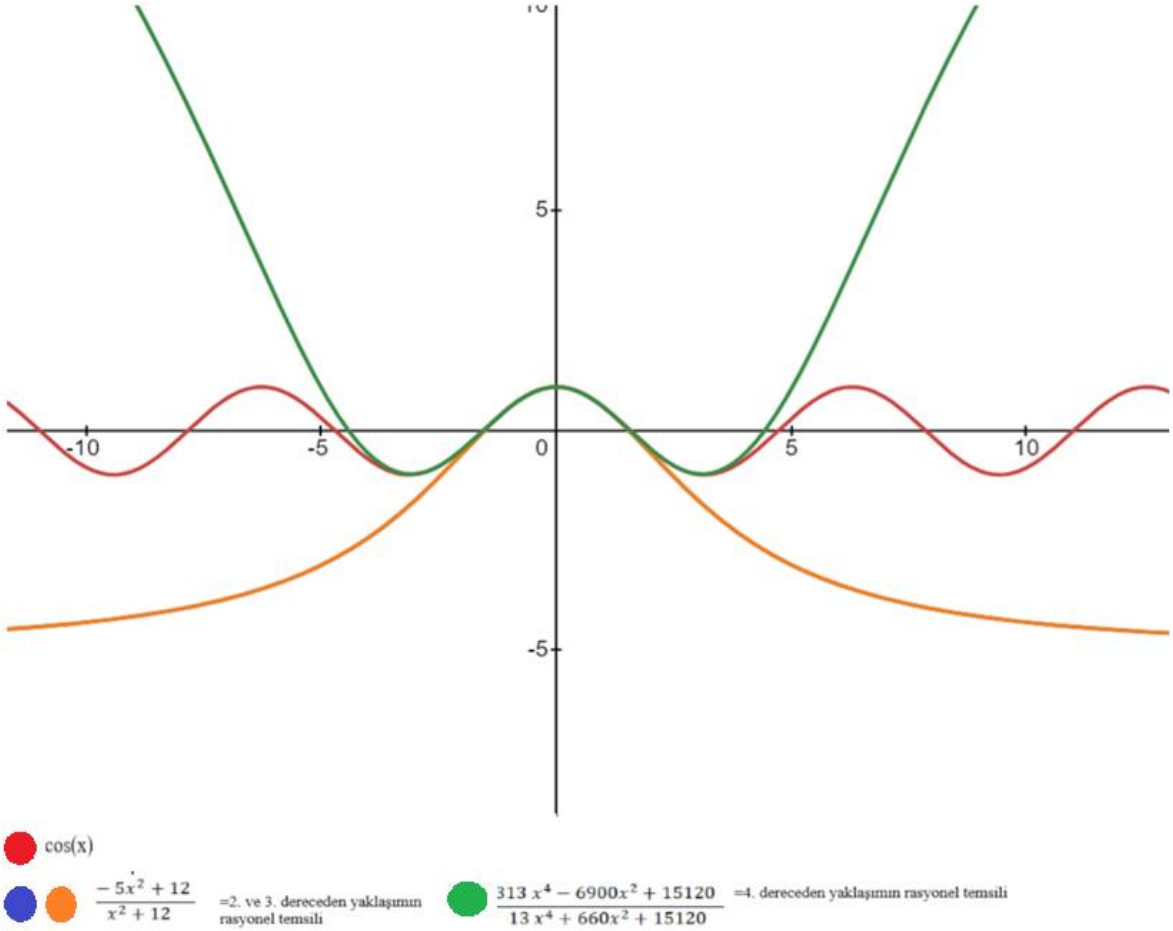
Tablo 26'nın devamı

e^{x}	[4,4]	$\frac{x^4 + 20x^3 + 180x^2 + 840x + 1680}{x^4 - 20x^3 + 180x^2 - 840x + 1680}$
e^{x}	[5,5]	$\frac{30240 + 15120x + 3360x^2 + 420x^3 + 30x^4 + x^5}{30240 - 15120x + 3360x^2 - 420x^3 + 30x^4 - x^5}$
e^{-x}	[2, 2]	$\frac{x^2 - 6x + 12}{x^2 + 6x + 12}$
e^{-x}	[3, 3]	$\frac{-x^3 + 12x^2 - 60x + 120}{x^3 + 12x^2 + 60x - 120}$
e^{-x}	[4, 4]	$\frac{x^4 - 20x^3 + 180x^2 - 840x + 1680}{x^4 + 20x^3 + 180x^2 + 840x + 1680}$



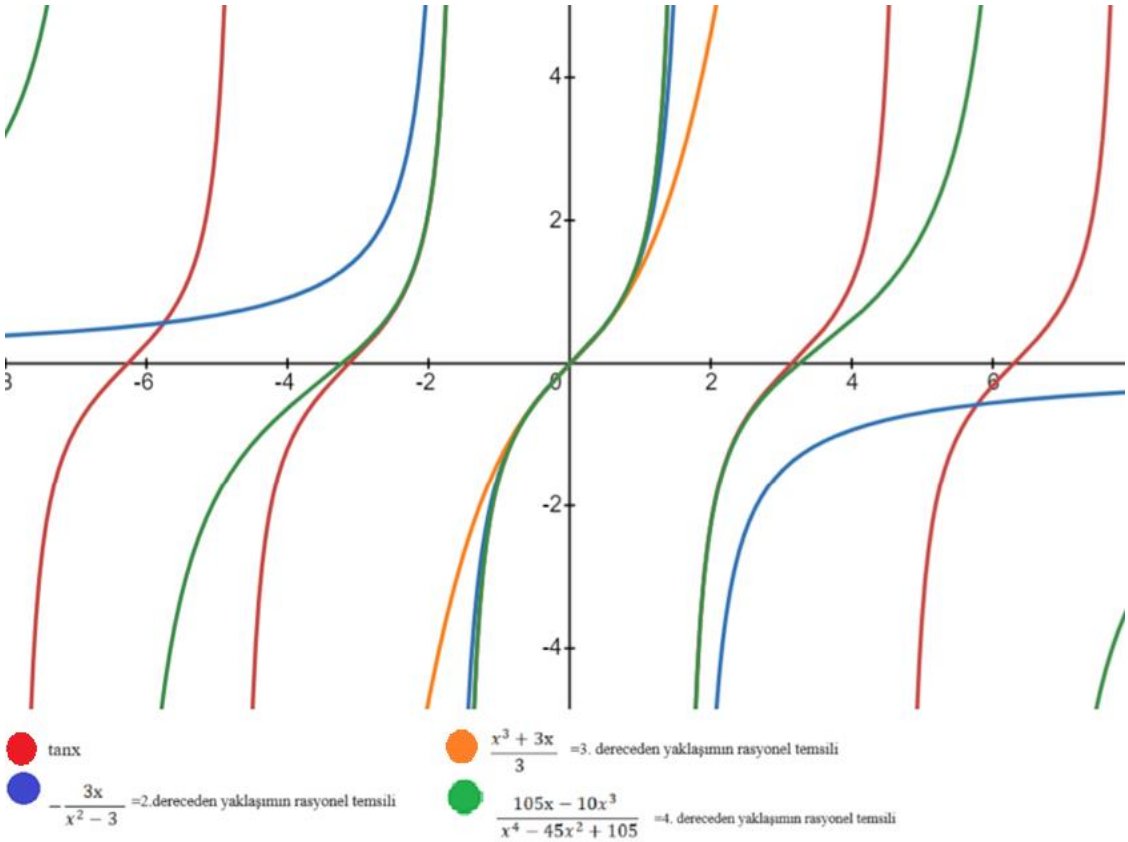
Şekil 15. $f(x)=\sin(x)$ polinom ifadesi ve rasyonel temsilleri

Şekil 15'te $\sin(x)$ polinom ifadesinin 2. , 3. ve 4.dereceden Pade yaklaşımının rasyonel temsili gösterilmiştir.



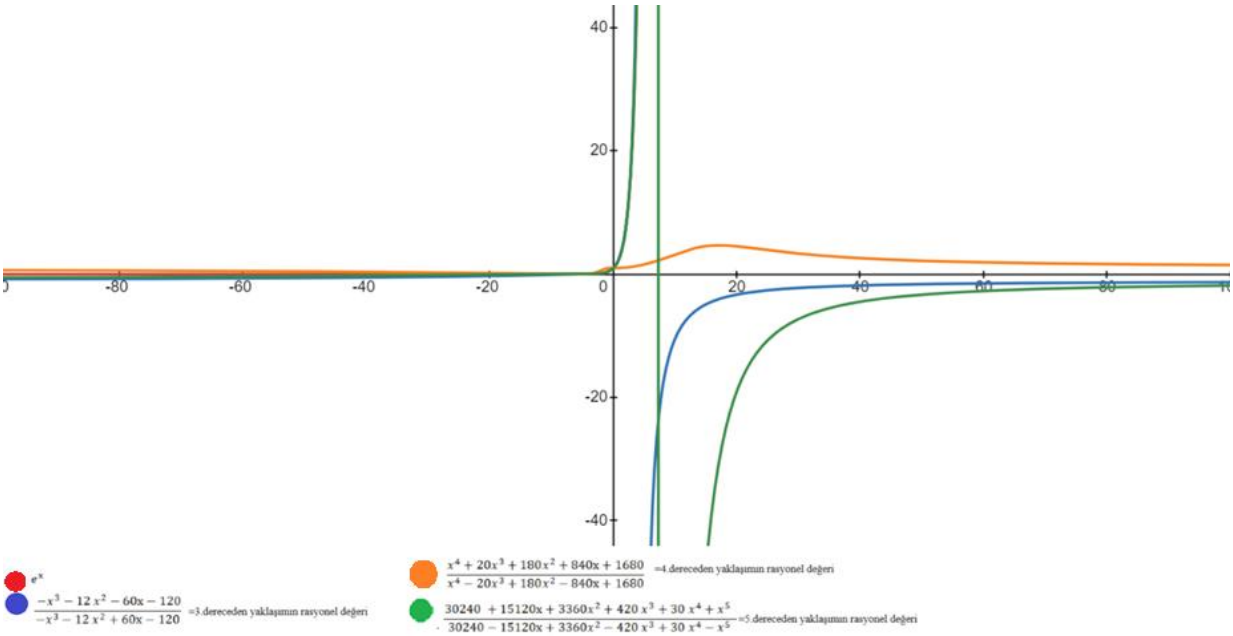
Şekil 16. $f(x) = cos(x)$ polinom ifadesi ve rasyonel temsilleri

Şekil 16'da $\cos(x)$ polinom ifadesinin 2. , 3. ve 4.dereceden Pade yaklaşımının rasyonel temsili gösterilmiştir.



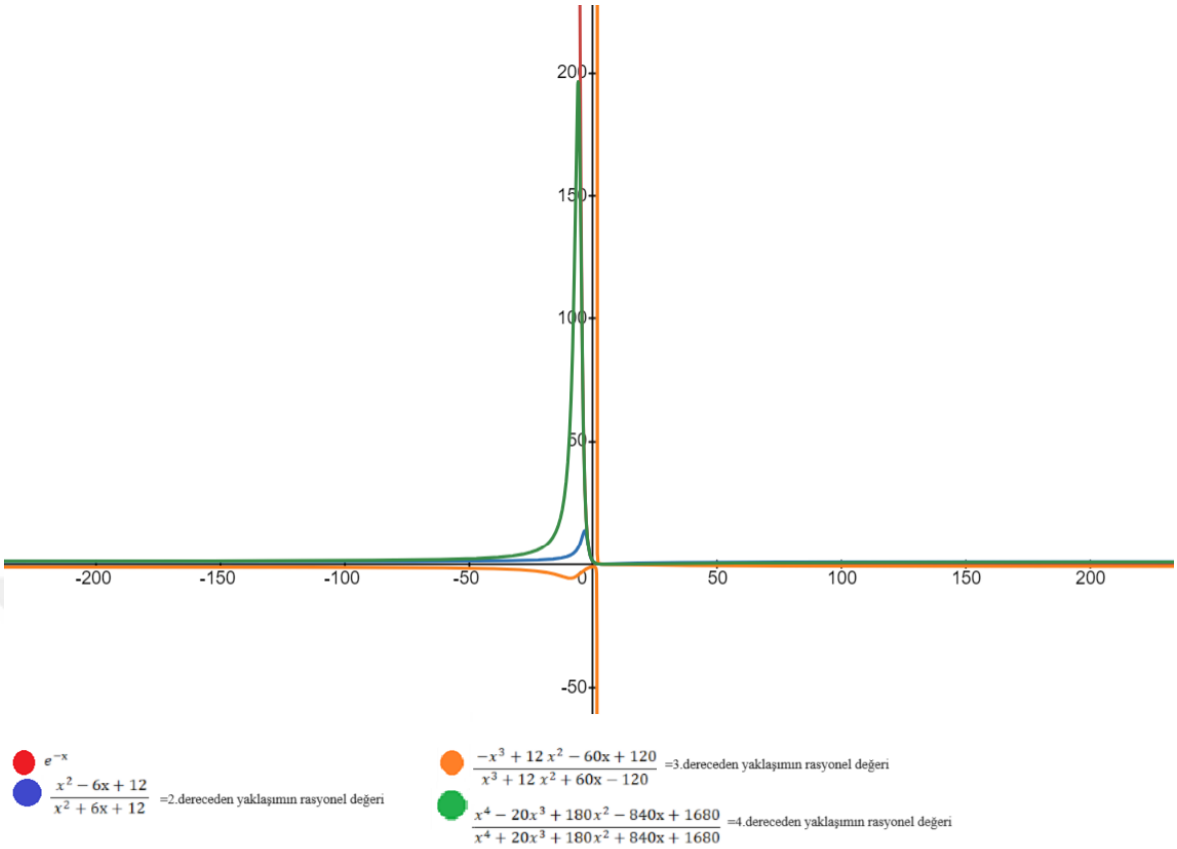
Şekil 17. $f(x) = tan(x)$ polinom ifadesi ve rasyonel temsilleri

Şekil 17'de $\tan(x)$ polinom ifadesinin 2. , 3. ve 4.dereceden Pade yaklaşımının rasyonel temsili gösterilmiştir.



Şekil 18. $f(x) = e^x$ polinom ifadesi ve rasyonel temsilleri

Şekil 18'de e^x polinom ifadesinin 2. , 3. ve 4.dereceden Pade yaklaşımının rasyonel temsili gösterilmiştir.



Şekil 19. $f(x)= e^{-x}$ polinom ifadesi ve rasyonel temsilleri

Şekil 19'da e^{-x} polinom ifadesinin 2. , 3. ve 4.dereceden Pade yaklaşımının rasyonel temsili gösterilmiştir

Durum çalışması 2: $f(x)= \sin(x)$ polinom ifadesinin 3. dereceden rasyonel polinomlar ile temsili;

$$R_{3,3}(x) = \frac{p_3(x)}{q_3(x)} = \frac{-7x^3+60x}{60+3x^2} \quad (34)$$

Denklem (34)'te gösterilen şekildedir. $f(x)= \sin(x)$ fonksiyonu için elde edilen terminal çıktısı Şekil 20'de gösterilmiştir.

```
fx: Sin(Id(x))
polinom dereceleri: 3 + 3

fx fonksiyonu Maclaurin serisine açılıyor.
fx_series: Poly((Ratio(RDec(0),RDec(1)),0)+(Ratio(RDec(1),RDec(1)),1)+(Ratio(RDec(0),RDec(2)),2)+(Ratio(RDec(-1),RDec(6)),3)+(Ratio(RDec(0),RDec(24)),4)+(Ratio(
RDec(1),RDec(120)),5)+(Ratio(RDec(0),RDec(720)),6))

Polinom terimlerinin katsayıları ve dereceleri
fx: [(0,0),(1,1),(0,2),((-1/6),3),(0,4),(1/120,5),(0,6)]
px: [(p0,0),(p1,1),(p2,2),(p3,3)]
qx: [(q0,0),(q1,1),(q2,2),(q3,3)]

Bunların en küçük ortak katı hesaplanır.
p0=0
p1=1
p2=0
p3=(-7/60)
q1=0
q2=1/20
q3=0
EKOK(60,20) = 60

Tüm katsayılar ekok ile çarpılır ve tabloya yazılır.

Bu değerler tablodan okunarak px ve qx polinomlarında yerlerine yazılır.
px: [(0,0),(60,1),(0,2),(((-7)),3)]
qx: [(60,0),(0,1),(3,2),(0,3)]
```

Şekil 20. $f(x) = \sin(x)$ polinom ifadesi için terminal çıktısı

4. SONUÇLAR

Bu çalışmada bir simgesel hesaplama yöntemi olan Pade yaklaşımını kullanarak matematiksel fonksiyonların çeşitli derecelerden rasyonel polinom temsillerini üreten bir yazılım uygulaması geliştirilmiştir. Yaklaşım, simgesel yöntemlerin programlanması için kullanılan EBNF benzeri notasyonunda bir LL(1) grameri oluşturularak simgesel hesaplama yöntemleriyle desteklenmiştir. Uygulamanın ayrıştırıcı bileşeni JavaCC formatında tanımlanarak oluşturulmuştur. Ayrıştırıcı ile bir yandan girdi olarak alınan fonksiyon ve polinom ifadelerinin gramer kurallarına uygunluğu denetlenirken, diğer yandan gramer kurallarıyla sözdizimi temsil edilen matematiksel ifadeler için soyut sözdizim ağacı üretilmektedir. İfadelere uygulanacak simgesel işlemler sözdizim ağacı düğümleri gezilerek yapılmaktadır. Girdi verisi olarak kullanılan rasyonel polinom fonksiyonları için türev alma ve değerlendirme aşamalarında sözdizim ağacını kullanan Ziyaretçi tasarım deseninden yararlanılmıştır.

Simgesel yaklaşımlar yardımıyla Pade yaklaşımının matematiksel ifadelere uygulanması farklı düzeylerde kodlama becerileri gerektirmektedir. Bu kapsamda bir f(x) fonksiyonunun seriye açılma süreci, türevinin alınması ve x=0 noktasında değerlendirilmesi gibi simgesel işlemler icra edilmiştir. Ayrıca PDF dosyalarının üretimi için rasyonel polinom ifadelerine ait sözdizim ağaçları üzerinden LaTeX diline yapılan dönüşümlerde MikTeX kütüphanesi kullanılmıştır. İşlem adımları ile üretilen ara ifadeler hem terminal üzerinde hem de PDF dokümanında gösterilmiştir.

5. ÖNERİLER

Pade yaklaşımı, matematiksel ifadeleri rasyonel polinomlar ile temsil etmenin güçlü bir yöntemi olsa da yüksek mertebeden polinomlar için yapılan değerlendirme ve türev alma gibi hesaplamalar performans kayıplarına neden olabilmektedir. Ziyaretçi tasarım deseninin kullanılması, kodun okunabilirliğini artırırken, büyük ölçekli hesaplamalarda daha fazla bellek kullanımı gerektirmektedir. Bu süreçlerde, özellikle türev hesaplamaları için optimizasyon yöntemlerinin kullanılması performans iyileştirmesine yardımcı olacaktır.

Polinom ifade üretimi için iki farklı girdi formatı kullanılmıştır. Özellikle belirli bir kalıba göre üretilen polinom ifadeleri için geliştirilen gramer yapısı, daha geniş bir matematiksel ifade yelpazesini kapsayacak şekilde zenginleştirilerek karmaşıklık seviyesi artırılabilir. Bu sayede, karmaşık matematiksel ifadelerin yaklaşık çözümlerine daha hızlı ve kolay bir şekilde ulaşmak mümkün olacaktır.

Pade yaklaşımı için polinom ifadelerin üretim aşamalarında elde edilen doğrusal denklem sistemi Gauss Eliminasyon yöntemi ile çözülmektedir. Bu yöntem, denklem sistemlerinin çözümlerinde etkili bir yöntem olsa bile büyük boyutlu sistemlerin çözümünde hesaplama karmaşıklığı ve maliyeti arttırmaktadır. Büyük matrislerin küçük matrislere bölündüğü bloklu Gauss Eliminasyon yöntemi, pivot seçimi yöntemi, LU ayrıştırma yöntemi, paralel hesaplama gibi çeşitli yöntemler kullanılarak hesaplama hataları ile birlikte karmaşıklık düzeyi de azaltılmış olacaktır.

Çalışma tarafından desteklenen PDF dokümanı üretme özelliği, görsel bir ara yüzle zenginleştirilerek kullanıcı için daha kullanışlı biçime getirilebilir. Pade yaklaşımı sonuçlarının ve grafiklerinin ekranda görsel olarak görüntülenmesi, kullanıcıların analiz süreçlerine daha etkin bir şekilde dahil olmalarını sağlayacaktır. Bu sayede çalışma, masaüstü veya web tabanlı bir uygulama olarak yeniden yapılandırılarak daha geniş bir kullanıcı kitlesine ulaşabilecek, kullanıcı deneyimini önemli ölçüde iyileştirecektir.

6. KAYNAKLAR

Andrianov, I., & Shatroy, A. (2021). *Pade approximants, their properties, and applications to hydrodynamic problems.*

Assche, W. V. (2006). *Pade and Hermite–Pade approximation and orthogonality. Surveys in Approximation Theory*, 61–91

Baker, G. A., & Graves-Morris, P. (1981). *Pade approximants. Addison-Wesley*.

Baker, G. A., & Gammel, J. L. (1961). *The Pade approximants. Journal of Mathematical Analysis and Applications*, 2(1), 21.

Bauer, C., Frink, A., & Kreckel, R. (2000*). Introduction to the GiNaC framework for symbolic computation within the C++ programming language*. Journal of Symbolic Computation.

Bergmann, S. D. (2017). *Compiler design: Theory, tools, and examples*.

Bojdi, Z. K., Ahmadi-Asl, S., & Aminataei, A. (2013). *A new extended Padé approximation and its application.*

Borwein, J., & Bailey, D. (2008). *Mathematics by experiment plausible reasoning in the 21st century*. AK Peters, Ltd.

Boyle, A., & Caviness, B. (1988). *Future directions for research in symbolic computation. Washington*, DC.

Brezinski, C. (1991). *History of continued fractions and Pade approximants*. Berlin.

Brenzinski, C. (1996). *Extrapolation algorithms and Pade approximations: A historical survey*.

Buchberger, B. (2021). *Symbolic computation in software science: My personal view. Research Institute for Symbolic Computation (RISC),* Johannes Kepler University.

Bustamante, Z. H. (1992). H*ermite–Padé approximations for Nikishin systems of analytic functions*. Mathematics Sbornik, 117–138.

Dogonchi, A. S., Hatami, M., Hosseinzadeh, Kh., & Domairry, G. (2015). *Non-spherical particles sedimentation in an incompressible Newtonian medium by Pade approximation.*

D'Inverno, R. A. (1975). *Algebraic computing in general relativity*. General Relativity and Gravitation, 6, 567–593.

Enseling, O. (2000, December 29). *Build your own languages with JavaCC*. JavaWorld.

Farhanaaz, S. V. (2016). *An exploration on lexical analysis.*

Ferris, M., & Coddington, D. P. (1995). *The state of the art in mathematical programming.*

Fowe, T. (2007*). Pade approximants and one of its applications.*

Gathen, J. V. Z., & Gerhard, J. (2013). *Modern computer algebra* (3rd ed.). Cambridge University Press.

Gonchar, A. A., Rakhmanov, E. A., & Sorokin, V. N. (2007). *Hermite–Padé approximations for systems of Markov-type functions.* Russian Academy of Sciences Sbornik Mathematics, 33–58.

Gökgöz, B., & Pehlivan, H. (2018). *Simgesel yaklaşımlarla sayısal kök bulma yöntemlerinin programlanması.*

Jeanmart, S., Gueheneuc, Y., Sahraoui, H., & Habra, N. (2021). *Impact of the visitor pattern on program comprehension and maintenance.*

Kakde, O. G. (2008). *Compiler design* (4th ed.). University Science Press.

Kaplan, T. (2011). *Lineer denklem sistemleri ve uygulama alanları*.

Liu, Y.-M., et al. (2009). *Adomian decomposition method and Padé approximation for nonlinear differential-difference equations*. Communications in Theoretical Physics, 581–587.

Lorentzen, L. (2010). *Pade approximation and continued fractions*. Applied Numerical Mathematics, 1364–1370.

Milani, M. M. R. (2014). D*esign and applications of grammar-based methodologies for automatic generation and step-by-step solving of mathematical expressions* (Doctoral dissertation, Karadeniz Teknik Üniversitesi, Fen Bilimleri Enstitüsü).

Mohammed, N. A. (2018). *Design and implementation of an interpreter for the least squares method using symbolic approach.*

Natori, K. (2012). *A design method of time-delay systems with communication disturbance observer by using Pade approximation.* IEEE International Workshop on Advanced Motion Control, 1–6.

Nolan, J. F. (1953). *Analytical differentiation on a digital computer* (Master's thesis).

Petkovic, I., & Herceg, D. (2017). *Symbolic computation and computer graphics as tools for developing and studying new root-finding methods*.

Slagle, J. R. (1963*). A heuristic program that solves symbolic integration problems in freshman calculus.* Journal of the ACM, 10(4), 509–520.

Succi, G., & Raymond, W. (1999). *The application of JavaCC to develop a C/C++ preprocessor*.

Quinlan, D. Q., Wells, K., & Kamareddine, F. (2019). *BNF-style notation as it is actually used.*

Tekbaş, Y. (2013). *Code production tools using automatic calculation of derivatives and simplification mathematical expressions* (Master's thesis, Karadeniz Teknik Üniversitesi, Fen Bilimleri Enstitüsü).

Turut, V., & Güzel, N. (2013). *On solving partial differential equations of fractional order by using the variational iteration method and multivariate Pade approximations.* European Journal of Pure & Applied Mathematics.

URL-1. (2024, May 11). Flex ayrıştırıcı: https://www.geeksforgeeks.org/flex-fast-lexical-analyzer-generator adresinden alınmıştır.

URL-2. (2024, May 11). Lex ayrıştırıcı: https://www.geeksforgeeks.org/what-is-lex-in-compiler-design adresinden alınmıştır.

URL-3. (2024, May 11). Quex ayrıştırıcı: adresi https://quex.sourceforge.net/ adresinden alınmıştır.

URL-4. (2024, May 11). YaCC ayrıştırıcı: https://en.wikipedia.org/wiki/Yacc adresinden alınmıştır.

URL-5. (2024, May 11). JavaCC: https://en.wikipedia.org/wiki/JavaCC adresinden alınmıştır.

URL-6. (2024, May 11). SableCC: https://sablecc.org/ adresinden alınmıştır.

URL-7. (2024, May 11). ANTLR: https://www.antlr.org/api/Java/org/antlr/v4/runtime/Lexer.html

Üstübioğlu, A. (2009). *Bir web-tabanlı Minihaskell yorumlayıcısı* (Master's thesis, Karadeniz Teknik Üniversitesi, Fen Bilimleri Enstitüsü).

Viswanadha, S., & Sankar, S. (2007). *The JavaCC FAQ*.

ÖZGEÇMİŞ

Esra ERDEN, Ondokuz Mayıs Üniversitesi Bilgisayar Mühendisliği Bölümü'nden 2015 yılında mezun olmuştur. 2018 yılından itibaren Polis Akademisi Başkanlığı'nda çalışmaktadır. İyi derecede İngilizce bilmektedir.

Yayınlar

- Erden E. & Pehlivan H.(2024), Determination of Rational Polynomial Functions Using Pade Approximation with Symbolic Mathematical Methods. VIII. Internatıonal Icontech Conference on Innovatıve Surveys ın Posıtıve Scıences, 275-286.