KARADENİZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

POLİNOMLARA DAYALI YAKLAŞIMLARLA SAYISAL ENTEGRASYON İFADELERİNİN ÜRETİLMESİ, DEĞERLENDİRİLMESİ VE DOKÜMANTASYONU

YÜKSEK LİSANS TEZİ

Özgür AKINCI

EKİM 2024
TRABZON

KARADENİZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

POLİNOMLARA DAYALI YAKLAŞIMLARLA SAYISAL ENTEGRASYON İFADELERİNİN ÜRETİLMESİ, DEĞERLENDİRİLMESİ VE DOKÜMANTASYONU

Özgür AKINCI

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

Tezin Enstitüye Verildiği Tarih : 05 / 09 / 2024

Tezin Savunma Tarihi : 01 / 10 / 2024

Tez Danışmanı : Doç. Dr. Hüseyin PEHLİVAN

Trabzon 2024

ÖNSÖZ

"Polinomlara Dayalı Yaklaşımlarla Sayısal Entegrasyon İfadelerinin Üretilmesi, Değerlendirilmesi ve Dokümantasyonu" isimli bu tez Karadeniz Teknik Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı, Yüksek Lisans Programı'nda hazırlanmıştır.

Bu çalışmada danışmanlığımı üstlenen Sayın Doç. Dr. Hüseyin PEHLİVAN hocama yardımlarından dolayı teşekkür ederim. Ayrıca her zaman bana destek olan eşime ve aileme desteklerinden dolayı teşekkür ederim.

Özgür AKINCI

Trabzon 2024

TEZ ETİK BEYANNAMESİ

Yüksek Lisans Tezi olarak sunduğum Polinomlara Dayalı Yaklaşımlarla Sayısal Entegrasyon İfadelerinin Üretilmesi, Değerlendirilmesi ve Dokümantasyonu 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. 25/10/2024

Özgür AKINCI

İÇİNDEKİLER

Sayfa No

Yüksek Lisans Tezi

ÖZET

POLİNOMLARA DAYALI YAKLAŞIMLARLA SAYISAL ENTEGRASYON İFADELERİNİN ÜRETİLMESİ, DEĞERLENDİRİLMESİ VE DOKÜMANTASYONU

Özgür AKINCI

Karadeniz Teknik Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı
Danışman: Doç. Dr. Hüseyin PEHLİVAN
2024, 65 Sayfa

Sayısal entegrasyon, çeşitli bilimsel disiplinlerde karşılaşılan birçok problemin içerdiği matematiksel hesaplamaların önemli bir bileşenidir. Polinom yaklaşımı gibi değişik yöntemler yardımıyla farklı doğruluk düzeylerine sahip entegrasyon ifadeleri türetilebilmektedir. Bu çalışmada polinomlara dayalı yöntemler kullanılarak sayısal entegrasyon ifadelerinin üretilmesi, değerlendirilmesi ve dokümantasyonunun yapılması amaçlanmıştır. Çalışma kapsamında geliştirilen yazılım uygulaması, polinom derecesi ve noktalar kümesi gibi bilgilerden oluşan bir girdi dosyası kullanmaktadır. Bu bilgiler, JavaCC aracı ile dosya sözdizimine uygun olarak oluşturulan bir LL(1) ayrıştırıcısı tarafından soyut sözdizim ağacı denilen bir veri yapısına dönüştürülmektedir. Bir Visitor tasarım deseni kullanılarak ağaç yapısı üzerinden gerçekleştirilen matematiksel hesaplamalar yardımıyla sayısal entegrasyon ifadeleri üretilmektedir. Hesaplama süreci polinom ifadelerinin açılımı, lineer denklem sistemine dönüşüm, Gauss eliminasyon yönteminin uygulanması, ifadelerin düzenlenmesi ve değerlendirilmesi, işlem adımlarının PDF dokümanına aktarılması gibi aşamalardan oluşmaktadır. Uygulama bileşenlerinin Java çevrelerine entegre edilebilmesi ile entegrasyon hesaplamalarının programlama süreçlerine destek verilmektedir. Ayrıca, ifade üretim adımlarının dokümantasyonu ile entegrasyon yöntemlerinin öğrenimine önemli bir katkı sağlanacağı, değerlendirilmektedir.

Anahtar Kelimeler: Sayısal entegrasyon, biçimsel gramerler, simgesel hesaplama, doküman formatlama

Master Thesis

SUMMARY

GENERATION, EVALUATION AND DOCUMENTATION OF NUMERICAL INTEGRATION EXPRESSIONS WITH POLYNOMINAL-BASED APPROACHES

Özgür AKINCI

Karadeniz Technical University
The Graduate School of Natural and Applied Sciences
Computer Engineering Graduate Program
Supervisor: Assoc. Prof. Hüseyin PEHLİVAN
2024, 65 Pages

Numerical integration is an important component of mathematical calculations in many problems encountered in various scientific disciplines. Several techniques, including polynomial approximation, can be used to derive integration equations with varying degrees of accuracy. This study aims to produce, evaluate and document numerical integration expressions using methods based on polynomials. The software application developed within the scope of the study uses an input file consisting of information such as polynomial degree and set of points. An LL(1) parser made in compliance with the file syntax using the JavaCC tool transforms this information into a data structure known as an abstract syntax tree. Numerical integration expressions are produced with the help of mathematical calculations performed on the tree structure using a Visitor design pattern. The calculation process consists of stages such as expansion of polynomial expressions, transformation to linear equation system, application of Gaussian elimination method, editing and evaluation of expressions, and transfer of processing steps to PDF document. The integration of application components to Java environments supports the programming processes of integration calculations. In addition, it is evaluated that the documentation of expression generation steps will make a significant contribution to the learning of integration methods.

Key Words: Numerical integration, formal grammars, symbolic computation, document formatting

ŞEKİLLER DİZİNİ

Sayfa No

TABLOLAR DİZİNİ

Sayfa No

SEMBOLLER VE KISALTMALAR DİZİNİ

$LL(k)$: Bir dilin belirli bir türdeki ayrıştırma yöntemini ifade eder

AST : Soyut sözdizim ağacı

n : Örneklem büyüklüğü

$p(x)$: Polinom

JAVACC : Java Compiler Compiler

AST : Abstract Syntax Tree (Soyut Sözdizim Ağacı)

1. GENEL BİLGİLER

1.1. Giriş

Sayısal entegrasyon, özellikle mühendislik problemlerinin içerdiği matematiksel hesaplamalarda kullanılan önemli bir çözümleme aracıdır. Birçok sayısal çözümleme yönteminin temelini oluşturan polinomlar, entegrasyon ifadelerinin belirlenmesinde ve değerlendirilmesinde de etkili bir yaklaşım sunarlar. Polinomlar, karmaşık matematiksel problemleri sadeleştirmek ve sayısal entegrasyon işlemlerini optimize etmek için temel araç olarak kullanılmaktadır.

Sayısal yöntemler, matematiksel problemlerin çözümlerini belirli bir hata payı ile gerçekleştirirken, simgesel yöntemler hata içermeyecek şekilde çözmeye çalışmaktadır. Matematiğin analitik çözüm üretemediği veya yüksek hesaplama karmaşıklığına sahip olduğu durumlarda sayısal yöntemler kullanılmaktadır. Simgesel yöntemler hatasız sonuç hedeflemesine rağmen, endüstriyel uygulamaların gerektirdiği hızdan yoksundur. Sayısal yöntemler ise, çıktıların doğruluğunu kesin olarak garanti edemese de verimlilik açısından simgesel hesaplamalara göre daha avantajlıdır. Sayısal entegrasyon, belirli integralin değerine sonlu bir toplam ile yaklaşır ve genellikle interplasyon polinomu yardımı ile çözüme ulaşır. Fonksiyon değerlerinin belirli bir setindeki noktalardan geçen bir polinom elde edilirken en yakın sonuca ulaşılmaya çalışılır.

İntegral hesaplamaları, matematik ve bilim dünyasında temel bir role sahiptir. İntegral kavramı bir fonksiyonun alanını veya hacmini hesaplama sürecini ifade eder ve geniş bir uygulama yelpazesinde önemli sonuçlar doğurur. Hesaplamaların aşırı karmaşık hale geldiği durumlarda sayısal entegrasyon yöntemleri uygulanarak birçok matematiksel problem çözülebilmektedir. Gerçek hayat problemlerinde integral hesaplamalarının doğrudan analitik olarak çözülememesi sıkça karşılaşılan bir durumdur. Örneğin, karmaşık fiziksel sistemlerin modellenmesi veya mühendislik problemlerinin çözümü sırasında, integral ifadeleri genellikle karmaşık ve analitik olarak çözümü imkânsız olabilir. Bu noktada sayısal entegrasyon yöntemleri devreye girerek, bu tür problemlerin pratik çözümlerini mümkün kılar.

Fizik ve mühendislik gibi disiplinlerdeki birçok bilimsel problem karmaşık matematiksel modellerle açıklanırlar. Bu modeller genellikle integral ifadelerini içerir ve analitik olarak çözümlenmeleri mümkün olmayabilir. Sayısal entegrasyon olmazsa, bu tür

modellerin çözümü büyük ölçüde kısıtlanır ve problemlerin gerçekçi bir şekilde modellemesi zorlaşır. İstatistik ve veri bilimi alanlarında, çoğu temel işlem integral hesaplamalarına dayanır. Örneğin, olasılık dağılımlarının altında kalan alanların hesaplanması veya bir veri setinin özet istatistikleri, integral kullanılarak elde edilir. Sayısal entegrasyonun olmaması durumunda, bu hesaplamaların doğru ve etkili bir şekilde yapılamayacağı gibi, istatistiksel sonuçların güvenilirliği de önemli ölçüde azalabilir. Mühendislikte, elektrik, mekanik veya kimyasal sistemlerin analizinde de integral hesaplamaları sıkça kullanılır. Örneğin, enerji dağılımının veya akışkan dinamiğinin modellenmesi integral yöntemlerini gerektirir. Sayısal entegrasyon, bu tür sistemlerin davranışlarının öngörülmesini ve optimizasyonunu kolaylaştırır. Ayrıca, bilimsel ve mühendislik uygulamalarında, doğrudan integral hesaplamalarının mümkün olmadığı durumlar da vardır. Örneğin, bir maddenin sıcaklık dağılımını hesaplarken, sürekli bir sıcaklık profili altında kalan enerjinin toplamını bulmak için integral kullanabiliriz. Ancak, bu integralin analitik olarak çözülmesi zor veya imkânsız olabilir. Bu gibi durumlarda sayısal entegrasyon yöntemleri, doğru sonuçlar elde etmek için vazgeçilmezdir. Fiziksel sistemlerin simülasyonları ve hesaplamalı fizik çalışmaları, genellikle integral ifadelerini içeren denklemlerle modellenir. Özellikle karmaşık sistemlerin dinamiklerini anlamak ve gelecekteki durumları tahmin etmek için integral hesaplamaları önemlidir. Sayısal entegrasyonun olmaması durumunda, bu tür simülasyonların doğruluğu ve güvenilirliği de ciddi şekilde etkilenebilmektedir. Sonuç olarak, sayısal entegrasyon yöntemlerinin gerekliliği, analitik çözümün imkânsız veya pratik olmadığı durumlarda ortaya çıkar. Gerçek hayat uygulamalarında, matematiksel modellerin ve hesaplamaların karmaşıklığı nedeniyle, integral hesaplamalarının doğrudan analitik olarak yapılamaması sık rastlanan bir durumdur. Bu nedenle, sayısal entegrasyon teknikleri, matematiksel modelleme, bilimsel hesaplama ve mühendislik uygulamalarında önemli bir araç olarak kabul edilir ve bu alandaki gelişmeler, sayısal analizin temel taşlarından birini oluşturur.

Polinomlara dayalı sayısal entegrasyon yöntemleri, mühendislik disiplinlerinde uzun yıllardır kullanılan temel araçlardan biri olmuştur. Newton-Cotes formülleri gibi klasik yöntemlerden başlayarak, günümüzde kullanılan adaptif ve iteratif yöntemlere kadar uzanan bir gelişim süreci izlenmektedir. Bu yöntemlerin bilgisayar teknolojilerindeki hızlı ilerlemeyle birlikte daha da güçlenmesi, mühendislik analizlerinin doğruluğunu ve verimliliğini artırmıştır. Gelecekte, yapay zekâ ve makine öğrenimi gibi teknolojilerin bu alandaki etkileriyle birlikte, daha karmaşık problemlerin çözümü için yeni yöntemler

geliştirilmesi beklenmektedir. Özellikle derin öğrenme modelleri, yüksek boyutlu integrallerde ve karmaşık geometrilere sahip problemlerde yeni ufuklar açabilir.

Mühendisliğin hemen her alanında polinom bazlı sayısal entegrasyon yöntemlerine rastlamak mümkündür. Otomotiv endüstrisinde, araç simülasyonlarından çarpışma testlerine kadar geniş bir uygulama alanı bulan bu yöntemler, araçların güvenliği ve performansının artırılmasında önemli bir rol oynamaktadır. Havacılık sektöründe ise uçak kanat tasarımı, roket motorlarının performans analizi gibi kritik uygulamalarda kullanılmaktadır. Enerji sektöründe rüzgâr türbinlerinin tasarımı, nükleer reaktörlerin güvenlik analizleri gibi alanlarda da bu yöntemlerin önemi büyüktür. Biyomedikal mühendisliğinde ise MR görüntüleme, ilaç geliştirme gibi alanlarda karmaşık biyolojik sistemlerin modellenmesinde kullanılmaktadır.

Günümüzde, mühendisler polinom bazlı sayısal entegrasyon yöntemlerini uygulamak için çeşitli yazılım araçlarından yararlanmaktadır. MATLAB, Python (SciPy, NumPy) gibi popüler yazılımların yanı sıra, GNU Octave, Scilab gibi açık kaynaklı alternatifler de bulunmaktadır. Bu yazılımlar, kullanıcı dostu arayüzleri ve geniş bir fonksiyon kütüphanesi sayesinde mühendislerin karmaşık hesaplamaları kolayca yapmalarını sağlamaktadır. Açık kaynak projeler, hem akademik hem de endüstriyel kullanımlarda önemli bir rol oynamakta ve bu yöntemlerin daha geniş kitlelere ulaşmasına katkıda bulunmaktadır.

1.2. Tezin Amacı

Bu tezde, metinsel girdilerle belirtilen bir noktalar kümesini kullanarak sayısal entegrasyon ifadelerinin türetilmesi, Java programlama çevrelerine entegrasyonunun sağlanması ve hesaplama adımlarının matematiksel notasyonlara uygun olarak bir PDF dokümanına aktarılması amaçlanmıştır. Girdi verisinin sözdizimine ilişkin JavaCC formatında yapılan tanımlamalar ile bir dil ayrıştırıcısı için kaynak kod üretimi yapılmıştır. Ayrıştırıcı kaynak veriye hiyerarşik bir yapı kazandırarak sözdizim ağacı oluşturmaktadır. Entegrasyon ifadeleri, sözdizim ağacı üzerinde gerçekleştirilen yapısal dönüşümler ve matematiksel hesaplamalar yardımıyla üretilmektedir. Bu kapsamda bir dönüşüm polinom ifadelerinin genişletilmesi ve doğrusal denklem sisteminin düzenlemesine yönelik uygulanmaktadır. İfadelerin değerlendirilmesi, programlama çevrelerinden kullanımı ve

doküman içeriğinin oluşturulması süreçlerinde sözdizim sınıflarından ve Visitor tasarım deseninden yararlanılmıştır.

Sembolik yaklaşımlar, polinomlar üzerinden sayısal integrasyonun gerçekleştirilmesinde güçlü bir yöntem olarak öne çıkmaktadır. Bu bağlamda, sembolik hesaplama yöntemleriyle türetilen entegrasyon ifadelerinin doğruluğu ve verimliliği, sayısal analiz ve mühendislik uygulamaları açısından büyük bir önem arz etmektedir. Tez çalışması, bu süreçlerin etkin ve verimli bir şekilde gerçekleştirilmesine olanak tanıyan önemli bir yenilik sunmaktadır.

Bu tezdeki bulgular, lisans düzeyli sayısal analiz derslerindeki programlama uygulamalarına daha kapsamlı katkı sağlayacaktır. Öğrencilerin polinom interpolasyonu, nümerik quadrature ve diğer ilgili konulardaki kavrayışlarının derinleştirilmesine yardımcı olacaktır. Ayrıca, tezde geliştirilen algoritmalar ve kodlama metodolojileri, öğrencilerin bu konuları daha iyi anlamalarını sağlamanın yanı sıra pratik uygulamalarda başvuru kaynağı olabilir ve mühendislik fakültelerindeki lisans düzeyindeki matematik derslerine de entegre edilebilir. Örneğin, diferansiyel denklemler dersinde, sayısal çözüm yöntemleri anlatılırken polinomlara dayalı entegrasyon yöntemlerine yer verilebilir. Bu sayede öğrenciler, teorik bilgileri gerçek dünya problemlerine uygulama becerileri kazanabilirler.

1.3. Literatür Taraması

Fauziyah ve arkadaşları (2021), düzensiz şekilli bir alanın sayısal integrasyon yöntemi kullanılarak hesaplanması üzerine odaklanmışlardır. Çalışmada, Trapez yöntemi kullanılarak, bilinmeyen bir fonksiyon tarafından sınırlanan bir alanın hesaplanması ele alınmıştır. Bu yöntem, fonksiyonun belirli bir aralıkta doğrusal bir polinom ile yaklaşık olarak ifade edilmesi ve bu şekilde alanın tahmin edilmesi esasına dayanmaktadır. Araştırma, Batı Java Eyaleti'nin alanını hesaplamak amacıyla gerçekleştirilmiştir. Sonuçlar, tahmini alanın gerçek alana yakınlığının, kullanılan bölüm sayısının artmasıyla birlikte iyileştiğini göstermiştir. Bu çalışma, düzensiz şekilli alanların hesaplanmasında Trapez yöntemi'nin etkinliğini vurgulamakta ve bölümlerin sayısının sonuçlar üzerindeki etkisini ortaya koymaktadır.

Sanır (2021)'ın çalışmasında ise, sayısal entegrasyon tekniklerinden olan Gauss Kuadratörünü kullanarak sayısal entegrasyonun hesaplanması ile ilgili uygulamalı örneklere yer verilmiştir. Benzer şekilde, diğer yöntemler ile Gauss Kuadratörü'nün

kıyaslanmasına da değinilmiştir. Çalışmanın çıktılarında ise, Gauss kuadratörü'nün diğer yöntemlere göre en büyük avantajının N+1 noktanın kullanıldığı durumlarda derecesi 2N+1 olan polinomları tam olarak integre edebilmesi ve bu sebeple çıkan sonuçların diğer yöntemlere göre daha keskin olduğu belirtilmiştir.

Casas ve arkadaşları (2021), OpenAPI/Swagger spesifikasyonunun yazılım geliştirme süreçlerindeki kullanımını ve uygulamalarını derinlemesine incelemiştir. OpenAPI, ya da daha yaygın olarak bilinen adıyla Swagger, web API'lerinin (REST, RESTful vb.) tanımlanmasında referans bir standart haline gelmiştir ve bu nedenle birçok araştırma makalesinin merkezinde yer almaktadır. Bu çalışma, OpenAPI/Swagger spesifikasyonunun çeşitli yazılım geliştirme faaliyetlerindeki kullanımlarını, bu süreçlerden elde edilen yazılım ürünlerini ve kapsadığı alanları kapsamlı bir şekilde derlemeyi amaçlamaktadır. Analiz edilen literatür taramasında, 2011 ile 2020 yılları arasında yayımlanan toplam 47 makale ele alınmıştır. Bu makaleler, spesifikasyonun kullanımına ilişkin geliştirme faaliyetlerini, ürettiği yazılım ürünlerini ve bu ürünlerin doğrulamasını içermekte, ayrıca kapsadığı alanları da referans almaktadır. Sonuçlar, OpenAPI/Swagger spesifikasyonunun çeşitli yazılım geliştirme aktivitelerine katkıda bulunduğunu ve en belirgin şekilde dokümantasyon iyileştirmesi sağladığını (%43) göstermektedir. Benzer şekilde, makalelerin %77'si web API tüketicilerinin gereksinimlerine ve sorunlarına çözüm sunmaktadır. Görevlerin otomasyonu, çeşitli araçlara dayanarak, %68 oranında en yaygın sonuç olarak öne çıkmaktadır. Ayrıca, incelenen çalışmaların %40'ı belirli bir alanı (IoT, Bulut Bilişim vb.) kapsamaktadır. Bu çalışma, OpenAPI/Swagger spesifikasyonunun yazılım geliştirme süreçlerinde nasıl etkili bir araç olduğunu ve çeşitli alanlarda sağladığı katkıları ortaya koymaktadır.

Sayısal çözümleme ve entegrasyon yöntemleri hakkında literatürde yeteri kadar çalışma olmasa da benzer çalışmalar analiz edilmiştir. Pehlivan (2019) yaptığı çalışmada, matematiğin analitik çözüm üretemediği veya yüksek hesaplama karmaşıklığına sahip olduğu durumlarda sayısal yöntemlerin kullanıldığını belirtmiştir. Sayısal yöntemlerin programlanabilmesini sağlayan bir programlama dilinin tasarımını gerçekleştirmiş ve bu dilde yazılan kaynak kodun değerlendirmesini yapabilen bir yorumlayıcı geliştirmiştir. Çalışmanın önemli sonuçlarından biri olarak bu tezin içeriğine ve uygulamanın gerçekleştirilmesine oldukça yardımcı olmuş ve ışık tutmuştur.

Ding ve arkadaşları (2011) tarafından gerçekleştirilen çalışma, frezeleme işleminin kararlılığını öngörmek amacıyla geliştirilmiş bir sayısal şemayı ele almaktadır. Bu

çalışmada, frezeleme dinamikleri, rejeneratif etkinin dikkate alınmasıyla integral denklem formunda temsil edilmiştir. Frezeleme süreci, analitik çözümün mümkün olduğu serbest titreşim fazı ve yaklaşık çözüm gerektiren zorlanmış titreşim fazı olmak üzere ikiye ayrılmıştır. Zorlanmış titreşim fazı sırasında integral denkleminin sayısal çözümünü elde etmek için, ilgi alanındaki zaman aralığı eşit şekilde ayrılmış ve Newton-Cotes veya Gauss integrasyon formülleri kullanılarak integral terimi yaklaşık olarak hesaplanmıştır. Sistem için bir periyottaki durum geçiş matrisi oluşturulduktan sonra, Floquet teorisi kullanılarak frezeleme kararlılığı öngörülmüştür. Yazarların sunduğu örnekler, önerilen yaklaşımın yüksek verimlilik ve doğruluğa sahip olduğunu göstermektedir. Bu çalışma, frezeleme işlemlerinde kararlılık analizine yönelik sayısal yöntemlerin geliştirilmesine önemli bir katkı sağlamaktadır.

Emiris (1998) tarafından yayınlanan çalışmada ise, sembolik ve sayısal hesaplama yöntemleri bir araya getirilerek, bu yöntemlerin ortak kullanımıyla yeni bir yaklaşım geliştirilmiştir. Ayrıca, sembolik ve sayısal yöntemlerin avantajlarını birleştirmenin neden önemli olduğu konusuna vurgu yapılmış, ne tür problemlere çözüm sağladığı konusunda da bilgiler verilmiştir. Emiris, sembolik ve sayısal hesaplama arasındaki hassasiyet ve doğruluk farklarını da inceleyerek hangi durumda hangi yaklaşımın daha verimli olduğu konusuna da değinmiştir.

Smyth (1997), sayısal integrasyonun, özellikle kesin matematiksel entegrasyonun mümkün olmadığı durumlarda belirli bir integralin yaklaşık değerini bulma süreci üzerinde durmuştur. Bu süreç, özellikle marjinal yoğunluk fonksiyonlarının veya rastgele değişkenlerin beklenen değerlerinin gerektiği istatistiksel analizlerde önem kazanmaktadır. Çalışmada, Trapez yöntemi, Simpson kuralı, Newton-Cotes formülleri, Clenshaw-Curtis integrasyonu ve Gauss Quadrature gibi klasik tek değişkenli quadrature yöntemleri ayrıntılı olarak ele alınmıştır. Çok boyutlu integrasyon (cubature) için kullanılan küresel adaptif yöntemler, polinomsal dereceli kurallar, kafes (lattice) yöntemleri ve Monte Carlo integrasyonu gibi çeşitli yöntemlerin olanakları ve sınırlamaları üzerine de kapsamlı bir inceleme sunulmuştur. Smyth, bu yöntemlerin uygulanabilirliğine yönelik mevcut yazılımlara da detaylı referanslar vermiştir. Bu çalışma, sayısal analizde integral hesaplamalarının önemli bir bileşeni olan quadrature yöntemlerinin anlaşılması ve uygulanması açısından temel bir kaynak niteliğindedir.

Herceg (2010) yayınladığı makalesinde GeoGebra araçları kullanarak lise öğrencilerinin sayısal integrasyonu daha iyi anlamalarını sağlamayı amaçlamaktadır. Kesin

integral kavramı genellikle Riemann toplamlarıyla tanıtılır ve bu kavramın anlaşılması öğrenciler için zor olabilir. Makalede, GeoGebra kullanarak sayısal entegrasyonun adım adım görsel ve etkileşimli bir şekilde öğretilmesi önerilmektedir. Öğrencilerin her adımı görsel yardımlarla manuel olarak yapabilmeleri, entegrasyonun temel prensiplerini daha iyi anlamalarını sağlar.

1.4. Sayısal İntegral ve Uygulamaları

Sayısal analiz, matematiğin önemli bir dalı olup, matematiksel problemlerin sayısal çözümlerini bulmak için çeşitli yöntemler geliştirir. Bu yöntemlerden biri de sayısal integraldir. Sayısal integral, bir fonksiyonun belirli bir aralıktaki integralini, yani alanını yaklaşık olarak hesaplamak için kullanılan yöntemler bütünüdür. Genellikle fonksiyonun analitik olarak entegre edilemediği durumlarda ya da fonksiyonun sadece belirli noktalardaki değerlerinin bilindiği durumlarda kullanılır. Sayısal integral, mühendislikten ekonomiye, fizikten biyolojiye kadar geniş bir uygulama alanına sahiptir. Matematiksel olarak, integral, bir eğrinin altındaki alanı ifade eder ve bu alanı bulmak için bir fonksiyonun integrali alınır. Ancak bazı fonksiyonlar için bu integral analitik yöntemlerle bulunamaz veya hesaplaması son derece karmaşık olabilir. Bu gibi durumlarda sayısal integral yöntemleri devreye girer. Sayısal integrasyon, bir integralin tanımına dayalı olarak, eğri altında kalan alanı küçük parçalara bölüp her bir parçanın alanını hesaplayarak toplam alanı bulma prensibine dayanır. Bu küçük parçalar, fonksiyonun üzerinde tanımlandığı aralığın belirli sayıda alt aralığa bölünmesi ile elde edilir. Her bir alt aralıktaki fonksiyon değerleri kullanılarak bu alt aralıkların alanları hesaplanır ve tüm alanlar toplanarak fonksiyonun integrali yaklaşık olarak bulunur. Sayısal integrasyon yöntemleri, bir integralin belirli aralıklar içindeki değeri için bir tahmin sağlar. Bu yöntemler, fonksiyonun eğrisini küçük dilimlere böler ve bu dilimlerin alanlarını toplar. Dilimlerin sayısı arttıkça, sayısal sonuç analitik sonuca daha fazla yaklaşır.

Sayısal integral, çok çeşitli bilimsel ve mühendislik alanlarında yaygın olarak kullanılan bir tekniktir. Fizik ve mühendislik disiplinlerinin çeşitli alanlarındaki problemlerin çözümünde hayati bir rol oynar. Örneğin, elektrik alan ve manyetik alan hesaplamaları, yapıların stres ve gerilme analizleri, aerodinamik kuvvetlerin tahmini gibi birçok problemde sayısal integrasyon kullanılır. Ayrıca, ısı transferi ve akışkanlar dinamiği gibi alanlarda da yaygın olarak uygulanır. Ekonomi ve finans alanında, olasılık yoğunluk

fonksiyonları ve risk analizi gibi konularda sayısal integrasyon yöntemleri sıklıkla kullanılır. Örneğin, opsiyon fiyatlandırma modellerinde (Black-Scholes modeli gibi), olasılık dağılımlarının entegrasyonu için sayısal yöntemlerden yararlanılır. Bu yöntemler, yatırım kararlarının optimize edilmesinde ve risk yönetiminde de büyük önem taşır. Biyolojik ve tıbbi araştırmalarda sayısal integral, çeşitli biyolojik süreçlerin modellenmesi ve analizi için kullanılır. Örneğin, ilaç dağılımı ve metabolizması, hücre büyüme modelleri ve sinir sistemi simülasyonları gibi alanlarda sayısal integrasyon yöntemleri kullanılarak önemli sonuçlar elde edilir. Bilgisayar grafiklerinde, ışık yayılımı ve gölgeleme modellerinde sayısal integrasyon teknikleri uygulanır. Özellikle ışığın bir yüzeyden nasıl yansıdığını veya geçirildiğini modellemek için Monte Carlo yöntemleri kullanılarak ışık simülasyonları yapılır.

Sonuç olarak, sayısal integral, matematiksel integrallerin yaklaşık çözümlerini bulmak için kullanılan etkili ve çok yönlü bir tekniktir. Çeşitli yöntemler ve teknikler sayesinde, analitik çözümün mümkün olmadığı veya zor olduğu durumlarda, fonksiyonların altındaki alanların yaklaşık hesaplanması sağlanır. Bu yöntemlerin geniş uygulama alanı, sayısal integrasyonun önemini ve pratikteki değerini açıkça ortaya koymaktadır. Fizik, mühendislik, ekonomi, biyoloji ve daha pek çok alanda, sayısal integrasyon yöntemleri, karmaşık problemlerin çözümünde vazgeçilmez araçlar arasında yer almaktadır.

1.5. Ayrıştırıcılar

Bilgi teknolojilerinde, verilerin doğru bir şekilde işlenmesi ve anlamlandırılması, modern yazılım sistemlerinin temel yapı taşlarından biridir. Bu sürecin merkezinde yer alan ayrıştırıcılar (parsers), bir dilin veya veri formatının kurallarını izleyerek ham girdileri anlamlı ve işlenebilir yapıdaki veriye dönüştüren araçlardır. Ayrıştırıcılar, programlama dillerinin derleyicilerinden veri işleme yazılımlarına, web tarayıcılarından veri tabanı sistemlerine kadar geniş bir yelpazede kritik rol oynamaktadır.

1.5.1. Sözdizim Analizi

Sözdizim analizi, bilgisayar bilimlerinde ve dilbilimde, bir dilin gramer kurallarına göre ifadelerin yapısal analizini yapan bir süreçtir. Bu analiz, bir dilin belirli bir giriş

dizesini (örneğin, bir programlama dilindeki kodu) alır ve bu girişin gramer kurallarına uygun olup olmadığını belirler.

Sözdizim analizi genellikle bir derleyici veya yorumlayıcının bir parçası olarak kullanılır ve bu aşamada kodun yapısal doğruluğu kontrol edilir. Bu analiz, kodun hangi yapıya sahip olduğunu belirlemek için bir sözdizim ağacını (parse tree) oluşturur. Örneğin, bir programlama dilinde bir if-else ifadesi yazarken sözdizimi analizi, ifadenin doğru yapıda olup olmadığını ve tüm parçaların (koşullar, bloklar vb.) uygun şekilde yerleştirildiğini kontrol eder.

Ayrıştırma sürecinde belirli bir dilin veya formatın sözdizimsel kuralları kullanılarak ham girdi analiz edilir ve daha yüksek seviyeli bir yapıya dönüştürülür. Bu süreç, genellikle iki ana aşamadan (Tokenization, Parsing) oluşur. Tokenization (belirteçleme)'da ham girdi, anlamlı en küçük birimlere yani belirteçlere (token) bölünür. Örneğin, bir programlama dilinde, bir satırdaki anahtar kelimeler, operatörler, değişken isimleri ve semboller belirteç olarak tanımlanır. Parsing (parse etme)'de ise ayrıştırıcı, belirteçleri analiz ederek, dilin gramer kurallarına göre yapısal bir temsilini oluşturur. Bu temsil genellikle bir soyut sözdizim ağacı (AST) şeklinde olur. Bu ağaç, girdinin mantıksal yapısını temsil eder ve dilin grameri ile uyumlu olup olmadığını kontrol eder.

Ayrıştırıcılar, kullanılan yöntem ve algoritmalara göre çeşitli kategorilere (LL, LR, Recursive Descent) ayrılabilir. Kaynak veriyi soldan sağa doğru analiz eden ve en sol türetimi kullanan araçlar, LL (Left-to-right, Leftmost derivation) ayrıştırıcıları olarak bilinirler. LL(k) ayrıştırıcıları, k belirteç ileriye bakış kullanarak girdiyi analiz ederler; örneğin, LL(1) ayrıştırıcıları, sadece bir belirteç ileriye bakış yapar. Bu tür ayrıştırıcılar genellikle üstten aşağıya (top-down) bir yaklaşımla çalışır. LR (Left-to-right, Rightmost derivation) ayrıştırıcıları, girdiyi soldan sağa doğru analiz ederken en sağ türetimi kullanarak çalışan araçlardır. LR ayrıştırıcıları, ileri yönde belirli bir derinliğe kadar bakarak girdiyi analiz edebilir ve bu sayede daha karmaşık dil yapılarının işlenmesine olanak tanır. LR(1) ayrıştırıcıları, yaygın olarak kullanılan bir alt türdür ve genellikle alttan yukarıya (bottom-up) bir yaklaşım kullanır. Recursive Descent ayrıştırıcıları ise, genellikle elle yazılan ve dilin gramerine göre fonksiyon çağrıları kullanılarak oluşturulan üstten aşağıya ayrıştırıcılardır. Her bir gramer kuralı, ayrıştırıcıda bir fonksiyon olarak temsil edilir ve bu fonksiyonlar, belirteç akışını işlemek için kendilerini veya diğer fonksiyonları çağırır.

Ayrıştırıcılar, yazılım mühendisliğinde ve bilgi teknolojilerinde birçok kritik alanda hizmet vermektedir. Bunların başında programlama dillerinin derlenmesi, veri formatlarının işlenmesi ve doğal dil işleme uygulamaları gelir. Derleyiciler, bir programlama dilinde yazılmış kodu makine diline veya bir ara koda çevirmek için ayrıştırıcıları kullanır. Ayrıştırıcı, kaynak kodu alır, bunu belirteçlere böler ve gramer kurallarına göre analiz eder. Analiz sonucu üretilen soyut sözdizim ağacı, kodun anlamını temsil eder ve bu temsil, sonraki derleme aşamalarında kullanılır. XML, JSON ve CSV gibi yapılandırılmış veri formatları, ayrıştırıcılar kullanılarak işlenir. Bu formatlardaki veriler anlamlı bir veri yapısına dönüştürülerek uygulamaların daha kolay erişmesi ve manipüle etmesi sağlanır. Metinlerin sözdizimsel analizi, cümlelerin dil bilgisi kurallarına göre ayrıştırılması ve anlamlandırılması için ayrıştırıcılar kullanılır. Bu şekilde metinler daha ileri düzeyde işlenebilir ve anlamları çıkarılabilir.

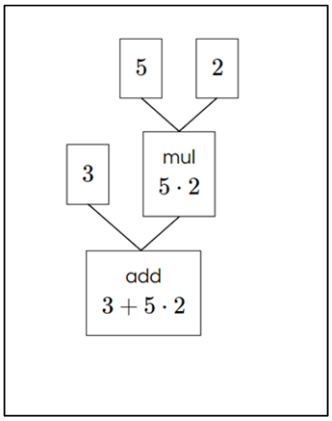
Ayrıştırıcıların doğru ve verimli bir şekilde tasarımı, yazılımın genel performansını ve güvenilirliğini doğrudan etkileyebilmektedir. Dolayısıyla, ayrıştırıcılar üzerine yapılan çalışmalar, yazılım mühendisliğinin temel taşlarından biri olarak kabul edilmektedir. Gelecekte, ayrıştırıcı teknolojilerinin daha da gelişmesi ve farklı alanlarda daha geniş uygulama bulması beklenmektedir.

1.5.2. Sözdizim Ağaçları

Sözdizim ağaçları ("parsing trees" ya da "syntax trees" olarak da bilinir), gramer kurallarına göre bir dilin ifade yapısını gösteren ağaç yapılarıdır. Bir dil (örneğin, bir programlama dili veya doğal bir dili) ifadesinin gramer kurallarına uygun olarak nasıl çözümlendiğini görsel olarak temsil eder. Bu ağaç, cümledeki her bir kelime veya sembolün gramatik olarak nasıl birbiriyle ilişkili olduğunu gösterir.

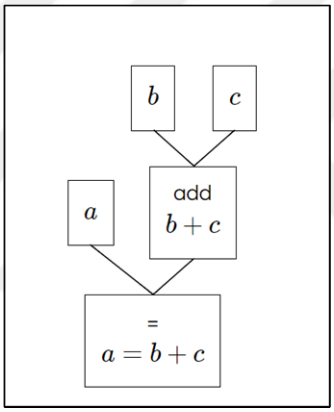
Sözdizim ağaçları, genellikle dilin bir ifadeyi nasıl anladığını veya yorumladığını anlamak için kullanılır. Programlama dillerinde, sözdizim ağaçları, bir derleyici veya yorumlayıcının kaynak kodunu anlamak ve çalıştırmak için kullandığı içsel bir yapıdır.

Basit bir aritmetik ifadeyi, örneğin 3 + 5 * 2 ifadesini çözümlemeye çalışalım. Bu ifade bir sözdizim ağacı olarak Şekil 1’deki gibi temsil edilecektir.



Şekil 1. “3 + 5 * 2” ifadesinin sözdizim ağacı

Ayrıca, C dilindeki basit ifadeyi (int a = b + c;) düşünelim. Benzer şekilde sözdizim ağacı Şekil 2’deki gibi oluşturulacaktır.



Şekil 2. “int a = b + c;” ifadesinin sözdizim ağacı

Sözdizim ağaç türleri “Somut Sözdizim Ağacı” (Concrete Syntax Tree - CST) ve “Soyut Sözdizim Ağacı” (Abstract Syntax Tree - AST) olmak üzere iki çeşittir. AST’de ifadenin mantıksal yapısını temsil eden düğümler bulunur. Gereksiz parantezler veya belirli dil yapıları çıkartılmıştır. AST’ler daha çok derleyicilerde kullanılır. CST’ler ise ifadenin tam anlamıyla her yönünü (parantezler, operatörler vb.) değerlendirir. Kaynak kodunun gramerine tam olarak uygundur.

Sözdizim ağaçları, bir dilin cümlelerini veya ifadelerini anlamlandırmak ve bu anlamlandırmayı bir yapı olarak temsil etmek için oldukça etkilidir. Bu ağaçlar, dilin gramerine uygun olarak, ifadeleri veya cümleleri analiz etmek için temel bir araç olarak kullanılır.

1.5.3. Ayrıştırıcı Üreteçleri

Ayrıştırıcı üreteçleri (parser generators), belirli bir dilin gramerini, bir dizi kurala dayalı olarak tanımlamanıza olanak tanır ve bu kurallardan bir ayrıştırıcıyı otomatik olarak oluşturur. Özellikle dil işleme, derleyici oluşturma veya metin analizi gibi alanlarda faydalıdır. Bir ayrıştırıcı üreteci, genellikle iki ana bileşenle (gramer tanımı, otomatik ayrıştırıcı oluşturma) çalışır. Gramer tanımı, ayrıştırıcının anlayacağı dilin kurallarını tanımlar. Bu kurallar genellikle bağlam bağımsız gramerler (Context-Free Grammars – CFG) olarak ifade edilirler. Otomatik ayrıştırıcı oluşturma bileşeni ise, gramer tanımından yola çıkarak, belirli bir dilde yazılmış metinleri analiz edebilecek bir ayrıştırıcı oluşturur. Günümüzde oldukça sık kullanılan popüler ayrıştırıcı üreteçleri vardır ve bunların başında, JavaCC (Java Compiler Compiler), YACC (Yet Another Compiler Compiler) ve ANTLR (Another Tool for Language Recognition) gelmektedir. Bu tez kapsamında, ayrıştırıcı olarak JavaCC tercih edilmiştir.

JavaCC (Java Compiler Compiler), Java programlama dili için bir ayrıştırıcı oluşturma aracıdır. Genellikle derleyici yazılımı geliştirmek, dillerin gramerlerini çözümlemek ve soyut sözdizim ağaçları (AST) oluşturmak için kullanılır. JavaCC, bir dilin gramer tanımını alır ve bu tanıma dayalı olarak bir ayrıştırıcı ve gerekli kelimesel analizleyici (lexer) üretir. JavaCC, programcıların karmaşık dil çözümleyicileri yazmalarını kolaylaştıran güçlü bir araçtır ve geniş bir yelpazede programlama dillerinin yorumlayıcılarını ve derleyicilerini geliştirmek için kullanılır. Örneğin, bir programlama dilinin gramerini tanımlayıp bu tanıma göre bir parser oluşturmak mümkündür. Veritabanı sorgu dilleri veya özel dosya formatları gibi belirli veri türlerinin işlenmesini kolaylaştırır. Örneğin, kod stilini kontrol eden araçlar veya kodu başka bir formata dönüştüren araçları geliştirmek için kullanılabilir.

JavaCC'nin temel işleyişini anlamak için basit bir örnek üzerinden gidilebilir. Örneğin, aritmetik ifadeleri çözümleyen bir parser yazmak isteyelim. Bu parser, toplama ve çıkarma işlemlerini tanımlayan basit bir gramer üzerinde çalışabilir. Öncelikle, çözümleyeceğimiz dilin gramerini tanımlarız. Bu tanım JavaCC'de .jj uzantılı bir dosya içinde yapılır. Bu gramer tanımını yaptıktan sonra, JavaCC komutuyla (javacc SimpleParser.jj) parser oluştururuz. Bu komut, gramer tanımına dayalı olarak Java sınıfları oluşturur. Bu sınıfları derleyip çalıştırdığımızda, belirtilen grameri çözümleyen bir parser elde ederiz. Böylece karmaşık gramer yapılarını ve dil tanımlarını çözmek ve bu tanımlara

uygun parser'lar üretmek mümkün olur. JavaCC, özellikle Java ekosisteminde yaygın olarak kullanılan bir araçtır ve sunduğu esneklik sayesinde çeşitli projelerde kullanılabilir. Dil çözümleme ve derleyici geliştirme alanlarında çalışanlar için vazgeçilmez bir araçtır. Geliştiriciler, JavaCC'yi kullanarak kendi dillerini tanımlayabilir ve bu diller için parser'lar oluşturabilir ve dil çözümleme süreçlerini otomatikleştirebilirler.

1.6. Doğrusal Denklem Sistemleri

Doğrusal denklem sistemleri, birden fazla bilinmeyeni içeren ve her bir denklemdeki bilinmeyenlerin katsayıları ile bilinmeyenlerin çarpımı toplamının sabit bir değere eşit olduğu denklemlerden oluşan sistemlerdir. Bu tür denklemler, genellikle matematik, mühendislik, ekonomi ve fizik gibi alanlarda karşılaşılan problemlerin çözümünde kullanılır. Örneğin, bir elektrik devresinde akımların dağılımı, bir ekonomik modelde talep ve arz dengesi gibi birçok durum, doğrusal denklem sistemleriyle modellenebilir. Bir doğrusal denklem sistemi, genel olarak aşağıdaki formda ifade edilir.

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1$$
$$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2$$
$$\dots$$
$$\dots$$
$$a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_m$$

Bu denklemler, matris formunda daha kompakt bir şekilde yazılabilir. Bir doğrusal denklem sistemi, matris ve vektörlerin yardımıyla aşağıdaki şekilde ifade edilir:

$$AX=B$$

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ . & . & . & . \\ . & . & . & . \\ a_{m1} & a_{m2} & . & a_{mn} \end{bmatrix}, \quad X = \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ x_n \end{bmatrix}, \quad B = \begin{bmatrix} b_1 \\ b_2 \\ . \\ . \\ b_m \end{bmatrix}$$

Burada:

- A, sistemin katsayılarını içeren m*n boyutunda bir matrisi temsil eder.
- X, bilinmeyenleri içeren n*1 boyutunda bir sütun vektördür.
- B, denklemlerin sağ tarafındaki sabit değerleri içeren m*1 boyutunda bir sütun vektördür.

Doğrusal denklem sistemlerini çözmek için çeşitli yöntemler bulunmaktadır. En yaygın kullanılan yöntemler arasında üçgen matris, Gauss eliminasyon ve ters matris yöntemleri yer alır.

1.6.1. Alt Üçgen ve Üst Üçgen Matris Yöntemleri

Alt üçgen matris yöntemi, bir doğrusal denklem sistemi AX=B şeklinde verildiğinde, A matrisinin alt üçgen matris (tüm elemanları sıfır olan bir üst üçgen matris) formunda olması durumunda kullanılır. Üst üçgen matris yöntemi ise, A matrisinin üst üçgen matris (tüm elemanları sıfır olan bir alt üçgen matris) formunda olduğu durumlarda kullanılır. Bu yöntemlerde, geriye doğru ya da ileriye doğru yerine koyma işlemi ile hızlı bir şekilde çözüme ulaşılır.

Bu yöntemler, özellikle matrisin belirli bir yapıya sahip olduğu durumlarda (örneğin, üçgen matrisler) oldukça etkilidir. Alt üçgen yöntemi (forward substitution) ve üst üçgen yöntemi (backward substitution), sırasıyla alt üçgen ve üst üçgen matrislerdeki denklemlerin çözümünde kullanılır. Bu yöntemlerin hesaplama karmaşıklığı, genel olarak O(n^2) olarak ifade edilir. Bu, her bir denklemin çözümünün bir önceki denkleme bağlı olduğu ve bu nedenle çözüm sürecinin ardışık olarak ilerlediği anlamına gelir.

1.6.2. Gauss Eliminasyon Yöntemi

Gauss eleminasyon yöntemi, doğrusal denklem sistemlerini çözmek için kullanılan bir algoritmadır. Bu yöntem, özellikle birden fazla denklem ve bilinmeyen içeren sistemlerde çözüm bulmak amacıyla kullanılır. Doğrusal denklem sistemlerinin çözümlerini bulmak için sistemin matris formunu kullanarak çözüm sürecini basitleştirir. Bu yöntem, her bir adımda matrisin şeklini değiştirerek sistemi daha basit bir forma (genellikle üçgen forma) getirir, ardından bu basit form üzerinden çözüme ulaşılır.

Bu yöntem ile çözümlemeler 5 temel adımdan (başlangıç matrisinin oluşturulması, öncelikli eleman seçimi, satır düzenlemeleri, geriye doğru ikili çözümleme, sonuçları kontrol etme) oluşmaktadır. Başlangıç matrisinin oluşturulması adımında, denklem sistemi genişletilmiş matris formunda yazılır. Öncelikli eleman seçimi(pivotlama) adımında, matristeki ana diyagonal üzerinde ve genellikle ilk satırdaki (veya mevcut satırdaki) en büyük mutlak değere sahip olan eleman seçilir. Bu eleman, pivot olarak adlandırılır. Bu

adım, hesaplama hata olasılığını azaltır ve sayısal stabiliteyi artırır. Satır düzenlemeleri adımında ise, pivot elemanı, aynı sütundaki diğer elemanların sıfır olması için satır işlemleri yaparak, matrisi üst üçgen formuna (sıfırların alt kısmında yer aldığı) getirilir. Bunu yapmak için, diğer satırları pivot elemanla oranlayarak gerekli işlemler yapılır. Geriye doğru ikili çözümleme adımında da, üçgen formdaki matrisi kullanarak, bilinmeyenleri bulmak için geri doğru çözümleme yapılır. İlk olarak en alt satırdaki bilinmeyen ve ardından üst satırlardaki bilinmeyenler sırasıyla çözülür. Sonuçları kontrol etme adımında ise, çözüm setine göre doğrulama yapılır ve gerekli ise çözümdeki herhangi bir hata için tekrar kontrol gerçekleştirilir.

1.6.3. Ters Matris Yöntemi

Eğer A matrisi kare bir matris ise (yani m=n), o zaman çözüm X=A^{-1}B formunda bulunabilir. Burada A^{-1}, A matrisinin tersini ifade eder. Bu yöntem, özellikle küçük boyutlu sistemler için kullanışlıdır. Ancak, büyük boyutlu sistemlerde ters matrisin hesaplanması, oldukça zaman alıcı ve hesaplama açısından pahalı olabilir. Ters matris yönteminde hesaplama adımları sırasıyla, denklemlerin matris formuna dönüştürülmesi, matrisin tersinin bulunması ve ters matris ile sonuç vektörünün çarpılmasıdır.

$$x + 2y = 5$$
$$3x - y = 2$$

Üst bölümdeki denklem sistemini ele alacak olursak, ilk adım olarak matris formuna dönüşüm işlenin gerçekleştirilmesi gerekecektir.

$$A = \begin{bmatrix} 1 & 2 \\ 3 & -1 \end{bmatrix}, \qquad b = \begin{bmatrix} 5 \\ 2 \end{bmatrix}$$

Daha sonra A matrisinin determinantı alınarak, matrisin tersinin bulunması sağlanmalıdır.

$$\det(A) = \big(1 * (-1)\big) - (2 * 3) = -1 - 6 = -7$$

$$A^{-1} = \frac{1}{\det(A)} * adj(a)$$

Adjoint (adjungate) matrisin elemanlarının bulunması sağlanır (A'nın kofaktör matrisini bulup, transpoze edilir). Böylece A matrisinin tersi kolayca bulunabilir (Adjoint matrisin det(A)'ya bölünmesiyle ters matris elde edilir).

$$\text{adj(A)} = \begin{bmatrix} 1 & 2 \\ 3 & -1 \end{bmatrix}$$

$$A^{-1} = \frac{1}{-7} * \begin{bmatrix} -1 & -2 \\ -3 & 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{7} & \frac{2}{7} \\ \frac{3}{7} & \frac{-1}{7} \end{bmatrix}$$

Ters matris yöntemi ile çözümü bulmak için, sonuç vektörü olan B ile A^{-1}çarpılır ve denklem çözümü bulunur.

$$x = A^{-1}B = \begin{bmatrix} \frac{1}{7} & \frac{2}{7} \\ \frac{3}{7} & \frac{-1}{7} \end{bmatrix} \begin{bmatrix} 5 \\ 2 \end{bmatrix} = \begin{bmatrix} \frac{1}{7} * 5 & \frac{2}{7} * 2 \\ \frac{3}{7} * 5 & \frac{-1}{7} * 2 \end{bmatrix} = \begin{bmatrix} \frac{5}{7} & \frac{4}{7} \\ \frac{15}{7} & \frac{-2}{7} \end{bmatrix} = \begin{bmatrix} \frac{9}{7} \\ \frac{13}{7} \end{bmatrix} = \begin{bmatrix} 1.29 \\ 1.86 \end{bmatrix}$$

Dolayısıyla, çözüm x≈1.29 ve y≈1.86 olur. Bu, denklemlerin çözümünü ters matris yöntemiyle nasıl bulabileceğinizi gösterir.

1.7. Sayısal İntegral Hesaplama Yöntemleri

Sayısal integral hesaplama yöntemleri, genellikle analitik çözümler bulmanın zor olduğu durumlarda kullanılır. En sık kullanılan sayısal integral hesaplama yöntemlerinden bazıları Riemann, Orta Nokta, Trapez, Simpson, Boole yöntemleridir.

1.7.1. Riemann Yöntemi

Riemann yöntemi, sayısal integrasyonun temel yöntemlerinden biridir ve bir fonksiyonun belirli bir aralıktaki integralini yaklaşık olarak hesaplamak için kullanılır. Bu yöntem, integral hesaplamanın geometrik anlamını kullanır ve bir eğrinin altındaki alanı bulmak için o eğrinin altındaki bölgeyi küçük dikdörtgenlere böler ve bu dikdörtgenlerin alanlarını toplar.

Riemann toplamı, genellikle integralin gerçek değerine yakındır, ancak bu yaklaşımın doğruluğu dikdörtgenlerin sayısına (yani n değerine) bağlıdır. n arttıkça, Riemann toplamı gerçek integral değerine daha yakın olur. Ayrıca, Riemann yöntemi, hem sürekli hem de parçalı sürekli fonksiyonlar için kullanılabilir, ancak fonksiyonun düzgün bir şekilde tanımlandığı aralıklarda daha doğrudur.

Riemann yöntemi iki temel adımda (aralık belirleme, dikdörtgenlerin yüksekliğini belirleme, toplam alan hesaplama) açıklanabilir. Aralık belirleme adımında, f(x) fonksiyonu ve [a,b] aralığı verilir. Daha sonra bu aralık n eşit uzunlukta aralığa(dikdörtgenlerin taban genişliği) bölünür. Her bir alt aralığın genişliği ise aşağıdaki formül kullanılarak hesaplanır.

$$\Delta x = \frac{b - a}{n}$$

Dikdörtgenlerin yüksekliğini belirleme adımında ise, dikdörtgenlerin yükseklikleri, fonksiyonunun belirli noktadaki değerine göre belirlenir. Bu nokta, her bir alt aralığın sol ucu, sağ ucu veya ortası olabilir. Bu seçime göre, Riemann toplamının üç farklı türü (Sol Riemann toplamı, Sağ Riemann toplamı ve Orta Nokta Riemann toplamı) vardır. Sol Riemann toplamında dikdörtgen yüksekliği alt aralığın sol ucundaki $f(x_i)$ değerine eşittir. Sağ Riemann toplamında dikdörtgenlerin yüksekliği alt aralığın sağ ucundaki $f(x_{i+1})$ değerine eşitken, Orta Nokta Riemann toplamında ise, alt aralığın orta noktasındaki $f(\frac{x_i + x_{i+1}}{2})$ değerine eşittir. Toplam alan hesaplama adımında da, tüm dikdörtgenlerin alanları toplanarak yaklaşık integral değeri elde edilir. Riemann toplamları için genel formül aşağıdaki gibidir.

$$R_n = \sum_{i=1}^{n} f(x_i) * \Delta x$$

1.7.2. Orta Nokta Yöntemi

Orta nokta yöntemi, sayısal integral hesaplamada kullanılan basit bir yöntemdir. Bu yöntem, bir fonksiyonun belirli bir aralıkta (a, b) integralini yaklaşık olarak hesaplamak için kullanılır. Bu yöntemde, İlk olarak, integral alınacak aralık [a,b] belirlenir. Bu aralık, n eşit parçaya bölünür. Her bir parçanın genişliği $\Delta x = \frac{b-a}{n}$ olur. Ardından, her bir aralık için, orta noktalar belirlenir ($x_i = a + (i - 0.5)\,\Delta x$). Son olarak, her bir parçacığın genişliği Δx ile orta noktadaki fonksiyon değeri çarpılır ve tüm bu çarpımların toplamı ise integralin yaklaşık değerini verecektir. Bu işlemler matematiksel olarak aşağıdaki şekilde ifade edilebilir.

$$\int_a^b f(x)\ dx \approx \Delta x \sum_{i=1}^{n} f(x_i)]$$

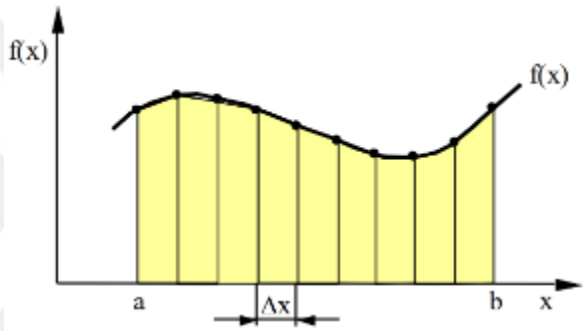
Orta nokta yöntemi, özellikle fonksiyonun belirli bir aralıkta düzgün ve sürekli olduğu durumlarda iyi sonuçlar verir. Ancak, daha karmaşık fonksiyonlar veya daha fazla

doğruluk gerektiğinde, bu yöntem daha hassas yöntemlerle (örneğin trapez yöntemi veya Simpson'un kuralı) karşılaştırılabilir.

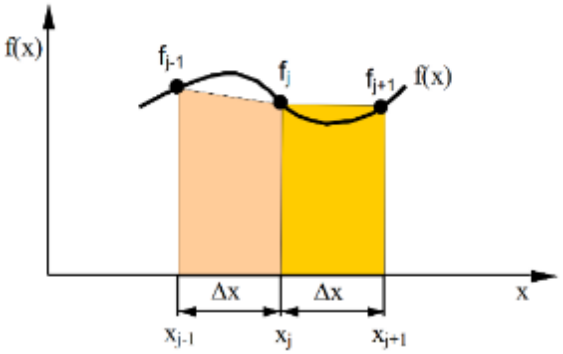
1.7.3. Trapez (Yamuk, İki Nokta Yaklaşımı) Yöntemi

Trapez yöntemi olarak bilinen 2 nokta yaklaşımında birbiri ardından gelen iki nokta bir doğru ile birleştirilir ve böylece yamuk şekline sahip dilimler elde edilir. Bu işlemin sonucunda ise integral hesabı, bu yamuk alanların toplamı olarak ifade edilir.

Trapez (yamuk) kuralı, aralığı eşit parçalara bölen düğümlerdeki fonksiyon değerlerini kullanır. Bu kural, periyodik aralıklardaki integraller için oldukça doğru sonuçlar verirken, periyodik olmayan durumlarda genellikle hatalı sonuçlar üretebilir.



Şekil 3. Yamuk yöntemi



Şekil 4. İki yamuk dilimi

Matematiksel olarak, bir [a,b] aralığında f(x) fonksiyonunun integralini yaklaşık olarak bulmak için trapez kuralı aşağıdaki formülleri kullanmaktadır.

$$\int_a^b f(x)dx \approx \frac{b-a}{2}[f(a)+f(b)]$$

Bu formülde, f(a) ve f(b), fonksiyonun ilgili aralıktaki değerlerini temsil eder. Trapez(yamuk) kuralı, integralin altında kalan alanı hesaplamak için kullanılırken, integralin değerini tam olarak doğru bir şekilde hesaplayamasa bile yeterince yakın bir sonuç döndürür. Bu yöntem, özellikle analitik olarak çözülemeyen veya çok karmaşık fonksiyonlar için sayısal çözüm gerektiren durumlarda kullanışlı olmaktadır.

Pratikte integral hesaplarken, aralığı daha küçük parçalara böler ve her parça için trapez kuralı uygulanır. Böylece daha hassas bir integral yaklaşımı elde edilebilir. Ayrıca, trapez (yamuk) kuralı, basit yapısı ve kolay uygulanabilirliği nedeniyle sıklıkla tercih edilen bir sayısal entegrasyon yöntemi olmuştur.

1.7.4. Simpson Kuralı

Simpson kuralı, bir aralık boyunca fonksiyon değerlerini kullanarak bu aralık içindeki alanı tahmin eder. Doğru sonuçlar elde edebilmek için aralığın yeterince küçük parçalara bölünmesini gerektirir. Ayrıca, fonksiyonun düzgün bir şekilde değişmesi bu yöntemin doğruluğunu artırır. İki temel versiyonu vardır (Simpson'un 1/3 kuralı ve Simpson'un 3/8 kuralı).

Simpson'un 1/3 kuralı, integralin hesaplanacağı aralığı eşit parçalara böler ve her parçayı bir parabol ile yaklaşık olarak temsil eder. Basit formülü aşağıdaki gibidir.

$$\int_{a}^{b} f(x)\ dx \approx \frac{h}{3}\left[f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \cdots + 4f(x_{n-1}) + f(x_n)\right]$$

Bu formülde, [a,b] integrali alınacak aralığı temsil ederken, $h(\frac{b-a}{n})$, her parçacığın genişliğini, $x_0, x_1, x_2, \ldots, x_n$ aralıktaki noktaları, $f(x_i)$ ise bu noktalardaki fonksiyonun değerini temsil etmektedir.

Simpson'un 3/8 kuralı ise, integral aralığını üç eşit parçaya böler ve her parçayı bir kübik polinom ile temsil eder. Basit formülü aşağıdaki gibidir.

$$\int_{a}^{b} f(x)\ dx \approx \frac{3h}{8}\left[f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)\right]$$

1.7.5. Boole Yöntemi

Boole yöntemi, belirli bir aralıkta bir fonksiyonun integralini hesaplamak için kullanılan bir yaklaşık yöntemdir. Bu yöntem, belirli bir aralıkta (örneğin [a,b])

fonksiyonun değerlerini, genellikle bir dizi düzeltilmiş polinomla temsil eder ve bu polinomları kullanarak integral hesaplar. Boole yöntemi için integral hesaplama formülü genel olarak aşağıdaki gibidir (h: aralık genişliği, $f(x_i)$: x_i noktalarındaki değerler).

$$I = \frac{7h}{90}\left[f(x_0) + 32f(x_1) + 12f(x_2) + 32f(x_3) + 7f(x_4)\right]$$

Boole yöntemi, genellikle daha hassas sonuçlar sağlar, çünkü daha yüksek dereceden bir polinom kullanır. Hata oranı genellikle daha düşüktür (özellikle fonksiyon düzgün ve sürekli olduğunda). Hesaplama maliyeti, özellikle daha karmaşık fonksiyonlar ve büyük aralıklar için artabilir. Bu yöntem, özellikle fonksiyonların integralini hesaplamak için hassasiyetin önemli olduğu durumlarda kullanışlıdır.

1.7.6. Yöntem Karşılaştırmaları

Sayısal integral hesaplama yöntemleri arasında Orta Nokta Yöntemi, Boole Yöntemi, Simpson Kuralı ve Trapez Yöntemi, çeşitli avantajlar ve dezavantajlar sunar, ve bu yöntemlerin seçiminde genellikle doğruluk, hesaplama maliyeti ve performans göz önünde bulundurulur.

Orta Nokta Yöntemi'nin en büyük avantajı, uygulamasının ve hesaplamasının oldukça basit olmasıdır. Ancak, bu basitlik, doğruluk açısından bazı sınırlamalara yol açabilir. Özellikle fonksiyonun değişimlerinin büyük olduğu veya doğrusal olmayan bölgelerde, bu yöntem doğru sonuçlar vermeyebilir. Bu nedenle, Orta Nokta Yöntemi genellikle daha az karmaşık fonksiyonlar için tercih edilir ve doğruluğun kritik olduğu durumlarda yetersiz kalabilir.

Boole Yöntemi, daha yüksek doğruluk sağlayan bir yöntemdir. Bu yöntem, integral hesaplaması için daha karmaşık polinomlar kullanır ve bu sayede fonksiyonun daha detaylı bir modellemesini sağlar. Bu yüksek doğruluk, Boole Yöntemi'nin genellikle daha karmaşık hesaplamalar gerektirmesine neden olur. Bu yöntemin avantajı, genellikle daha iyi sonuçlar sunmasıdır. Ancak, daha karmaşık hesaplamalar ve daha fazla fonksiyon değerlendirmesi gerektiğinden, hesaplama maliyeti ve süresi Orta Nokta Yöntemi'ne göre daha yüksektir. Boole Yöntemi, düzgün ve sürekli fonksiyonlar için etkili sonuçlar verirken, hesaplama maliyetinin yüksek olması bazı durumlarda sınırlayıcı olabilir.

Simpson Kuralı, Orta Nokta Yöntemi ve Trapez Yöntemi'nin avantajlarını birleştiren bir yöntemdir. Bu yöntem, fonksiyonun ikinci dereceden bir polinomla yaklaşık olarak modellenmesini sağlar ve genellikle yüksek doğruluk sunar. Simpson Kuralı'nın avantajı,

doğruluk ve performansı arasında iyi bir denge sağlamasıdır. Ancak, bu yüksek doğruluk da daha fazla hesaplama gerektirir, çünkü her adımda daha karmaşık hesaplamalar yapılır. Bu nedenle, Simpson Kuralı, doğruluğun önemli olduğu durumlarda tercih edilirken, hesaplama maliyeti orta seviyededir.

Trapez Yöntemi, integral hesaplamasında orta düzey bir doğruluk sunar. Bu yöntemde, her bir aralık trapez şekline dönüştürülerek alan hesaplamaları yapılır. Trapez Yöntemi, Orta Nokta Yöntemi'ne göre daha iyi sonuçlar sağlar çünkü fonksiyonun her iki uç noktası da dikkate alınır. Ancak, Simpson Kuralı kadar yüksek doğruluk sağlamaz. Bu yöntemin hesaplama maliyeti ve süresi, genellikle orta düzeydedir ve fonksiyonun düzgün olduğu durumlarda yeterli performans sağlar. Trapez Yöntemi, genellikle daha az karmaşık hesaplamalar gerektirdiğinden, Simpson Kuralı'na göre daha düşük maliyetli olabilir.

1.8. Uygulama Programlama Arayüzü (API)

API, "Application Programming Interface" yani "Uygulama Programlama Arayüzü" anlamına gelir. Bir API, farklı yazılım uygulamalarının birbirleriyle iletişim kurmasına ve veri alışverişinde bulunmasına olanak tanıyan bir arayüzdür. API'ler, bir uygulamanın diğer bir uygulamanın işlevselliğini kullanmasını sağlar, böylece geliştiriciler belirli görevleri yerine getirmek için yeniden kod yazmak zorunda kalmazlar. API'ler genellikle belirli kurallar ve protokoller çerçevesinde çalışır. Örneğin, bir hava durumu uygulaması, bir hava durumu API'si aracılığıyla mevcut hava koşulları hakkında bilgi almak için bir sunucuya istek gönderir. Sunucu, talep edilen bilgileri döndürür ve uygulama bu verileri kullanıcıya gösterir. API'ler, web servisleri, işletim sistemleri, veritabanları ve uygulama geliştirme platformları dahil birçok alanda kullanılır. Ayrıca, REST, SOAP ve GraphQL gibi çeşitli API türleri ve standartları vardır. Bu tez kapsamında, JSON Api tercih edilmiştir.

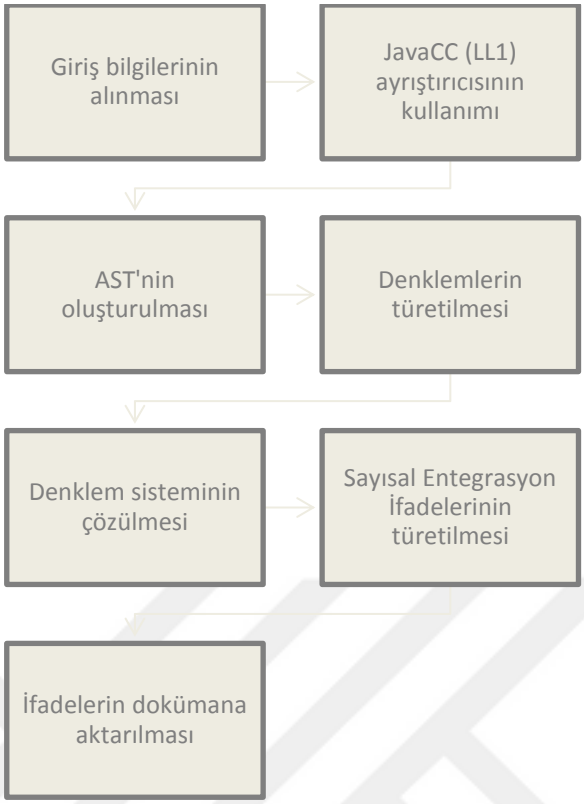
Swagger API Dokümantasyonu, RESTful API'lerin tasarımını, geliştirilmesini ve belgelendirilmesini kolaylaştıran bir araç ve framework'tür. Swagger, özellikle API'lerin ne tür veri alıp ne tür veri döndürdüğünü, hangi endpoint'lerin mevcut olduğunu, bu endpoint'lerin ne tür HTTP metotları (GET, POST, PUT, DELETE, vb.) kullandığını açıkça belirtmek için kullanılır. Bu tez çalışmasında da, api belgelendirme altyapısı için Swagger API kullanılmaktadır.

2. YAPILAN ÇALIŞMALAR

2.1. Giriş

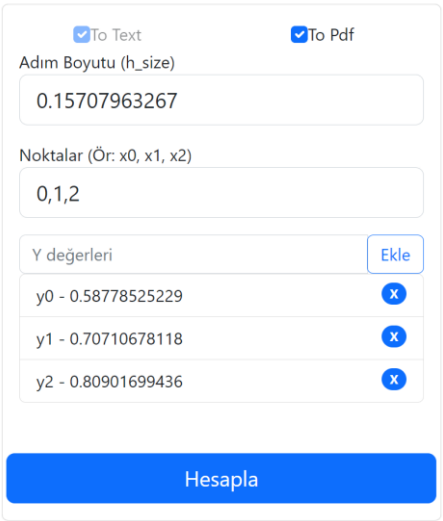
Sayısal entegrasyon hesaplamaları için genellikle iteratif yada fonksiyonel yaklaşımlar kullanılır. Özellikle polinomlar gibi yaklaşım fonksiyonları ile entegrasyonun matematiksel ifadeleri de belirlenebilmektedir. Bu tezde, polinomlara dayalı yaklaşımlar kullanılarak sayısal entegrasyon ifadeleri hesaplanmış ve Java programlama çevrelerine entegrasyonu sağlanmıştır. Polinom derecesine bağlı olarak gerçekleştirilen hesaplamaların işlem adımları için bir PDF dokümanı oluşturulmaktadır. Newton-Cotes formülleri olarak bilinen Yamuk, Simpson, Simpson 3/8 ve Boole kuralları sırasıyla birinci dereceden dördüncü dereceye kadar polinomlar kullanılarak türetilmektedir. Çalışmada, bu kuralların içerdiği ifadelerin yanı sıra daha yüksek dereceden polinomlara dayandırılan ve sayısal entegralleri daha yüksek doğruluk ile hesaplayabilen yeni ifadelerin üretimleri yapılabilmektedir.

Tez kapsamında, sayısal entegrasyon ifadelerinin türetilebildiği, web ortamından çalıştırılabilen bir yazılım uygulaması geliştirilmiş ve ifade hesaplama adımlarının dokümantasyonu LaTeX biçimlendirme dili kullanılarak hazırlanmıştır. Bu uygulama, sayısal entegrasyonun temel prensiplerini ve polinom tabanlı ifade türetim tekniğini derinlemesine kavramak isteyen öğrenciler, araştırmacılar ve programcılar için değerli bir kaynak niteliği taşımaktadır.



2.2. Girdi Bilgilerinin Alınması

Sayısal entegrasyon ifadelerinin hesaplanabilmesi için girdiye ihtiyaç vardır. İfade üretimi için gerekli olan temel parametreler geliştirilen bir web uygulaması arayüzünden ya da rest api isteği ile okunmaktadır.



Şekil 5. Web arayüzü ile parametre tanımı

Şekil 5'te gösterilen örnek girdideki gibi, *adim_boyutu* değeri, noktalar ve y değerleri parametre olarak programa girilmektedir. h ile ifade edilen *adim_boyutu* x değerlerinin genişliğini, y ise fonksiyon değerlerini temsil etmektedir. Bu çalışmada geliştirilen program, parametre olarak aldığı bu değerler ile sayısal entegrasyon ifadelerinin türetilmesini sağlamaktadır. İlgili girdi ifadeleri bir arayüz yerine doğrudan bir txt uzantılı dosyadan okunmak istenirse aşağıdaki formata uygun şekilde hazırlanmalıdır.

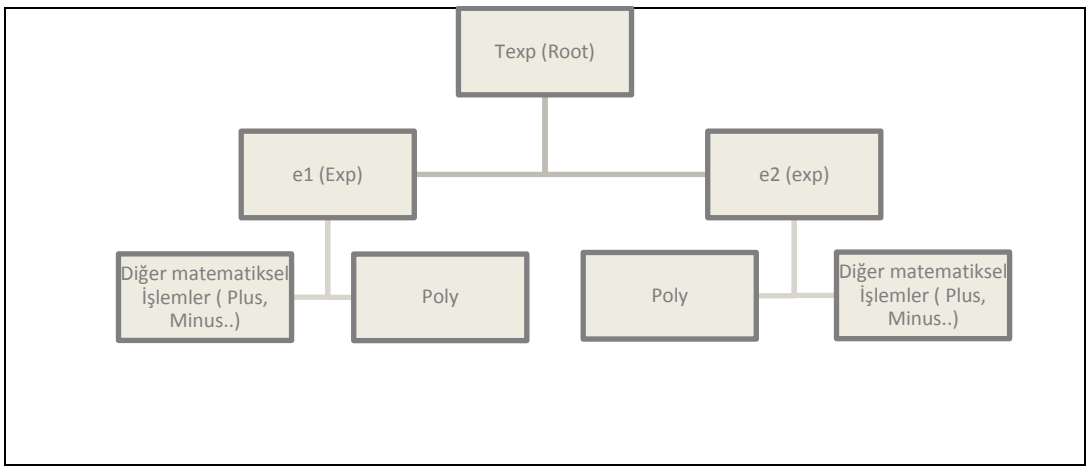
Tablo 1. Girdi dosyası örneği (txt)

```
h_size=0.15707963267;
m_list={0,1,2}
y_vals={0.58778525229,
0.70710678118, 0.80901699436 };
```

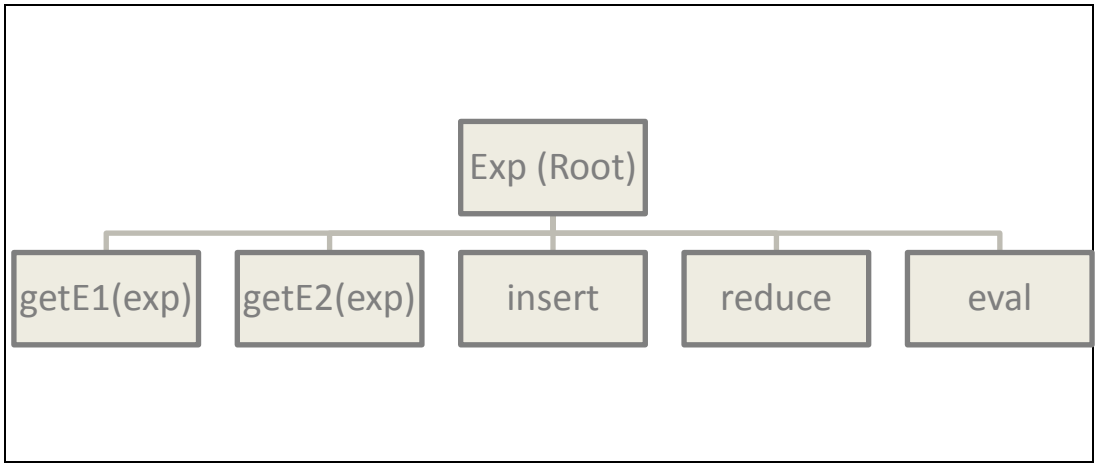
Tablo 1'deki gibi bir txt dosyası ile alınan parametrelerden *h_size* değişkeni double tipinde bir veri olmakla birlikte adım boyutunu (h yüksekliği) temsil etmektedir. *m_list* listesi int[] tipinde bir veridir ve polinomun standart interpolasyon formülünün oluşturulmasında kullanılmaktadır. *y_vals* listesi ise, double[] tipinde bir değişkendir ve noktaları temsil etmektedir.

2.3. Soyut Sözdizim Ağacı

JavaCC yardımı ile girdi dosyası içerisinde yer alan verilerin, belirli gramer tanımlarına göre parçalanması ve bu parçalanan verilerin (birim sözcük) ağaç yapısında gösterimi ile soyut sözdizim ağacı oluşmaktadır. Texp (root) ana düğüm olmakla birlikte alt düğümler aşağıdaki görsellerde ifade edilmiştir.



Şekil 6. Soyut sözdizim ağacı (Texp)



Şekil 7. Soyut sözdizim ağacı (Exp)

Tablo 2. Uygulamanın sözdizim sınıfı

```
class Texp {
  Exp e1, e2;
  public Texp(Exp a, Exp b) {
    e1 = a;
    e2 = b;
  }
  public Exp getE1() {return e1;}
  public Exp getE2() {return e2;}
  public String toString() {
    return e1 + "=" + e2;
  }
  public String toLatex(boolean x) {
    return e1.toLatex(x) + "=" +
e2.toLatex(x);
  }
  public String toClass() {
    return
"Texp("+e1.toClass()+","+e2.toClass()+")";
  }
}
abstract class Exp {
  public abstract Exp getE1();
  public abstract Exp getE2();
  public abstract String toClass();
  public abstract Exp insert(Table t);
  public abstract Exp reduce();
  public Ratio eval(Table t) {
     return new Ratio();
  }
  public BigDecimal getDecimal() {
     return BigDecimal.ZERO;
  }
  public String toLatex(boolean x) {
     return "";
  }
}
```

```
class Poly extends Exp {…}
class Plus extends Exp {…}
class Minus extends Exp {…}
class Times extends Exp {…}
class Divide extends Exp {…}
class Power extends Exp {…}
class Par extends Exp {…}
class RFact extends Exp {…}
class Id extends Exp {…}
class Num extends Exp {…}
class Diff extends Exp {…}
class RDec extends Exp {…}
class Ratio extends Exp {…}
```

Texp sınıfı, iki ifadeyi Exp türünden tutan bir sınıftır. e1 ve e2 ile referans verilen bu iki ifade bir denklemin sol ve sağ tarafını temsil eder. getE1() ve getE2() metotları sırasıyla, e1 ve e2 değişkenlerine erişimi sağlamaktadır. toString() ve toLateX() metotları ise bu ifadelerin sırasıyla string ve LaTeX formatında temsillerini döndürür. Exp sınıfı, tüm matematiksel ifadeleri temsil eden soyut bir sınıftır. Bu sınıftan türetilmiş sınıflar (Plus, Minus, Times, Divide, Power, vb.) matematiksel işlemleri ve ifadeleri tanımlar.

2.4. Bütünsel Hesaplama Adımları

Entegrasyon ifadelerinin üretilmesi için ihtiyaç duyulan hesaplamalar, bir Visitor tasarım deseni yardımı ile ağaç yapısı üzerinde gerçekleştirilmektedir. Bu hesaplamalar,

polinom ifadelerinin açılımı, lineer denklem sistemine dönüşüm, Gauss eleminasyon yönteminin uygulanması, entegrasyon ifadelerinin üretilmesi ve hesaplama adımlarının LaTeX dili ile formatlanarak pdf dokümanına aktarılması gibi adımları içermektedir.

Çalışma Java ve JavaCC teknolojileri kullanılarak gerçekleştirilmektedir. Programlama desteği bakımından, diğer uygulamalara kolayca entegre olabilmesi ve api desteği ile kolay kullanıma sahip olması önemli özelliklerindendir. Gelişmiş dokümantasyon desteği ile de tüm hesaplama adımlarının LaTeX formatında oluşturulup, MiKTeX kütüphanesi yardımıyla pdf formatına dönüştürülerek çıktı alınması amaçlanmaktadır.

2.4.1. Denklemlerin Türetilmesi

Bir $P_n(x)$ polinomu genel olarak aşağıdaki ifade ile temsil edilir ve girdi olarak tanımlanan n değeri için polinom ifadesi oluşturulur. Bu polinomun oluşturulmasında, sözdizim sınıflarında tanımlanan ve girdi değerlerinin parametre olarak geçildiği metotlar önemli rol oynar.

Tablo 3. $P_n(x)$ polinomunun genel biçimi

$P_n(x) = c_0 + c_1 x + c_2 x^2 + c_3 x^3 + \cdots + c_n x^n$

2.4.2. Örnek Noktaların Belirlenmesi

Sözdizim sınıflarına dayalı olarak oluşturulan polinom ifadesi, $x_m = x_0 + mh$ biçiminde seçilen her bir nokta için yeniden düzenlenir. Örneğin, *h=0.15707963267*, *m_list={0,1,2}* ve y_list={0.58778525229, 0.70710678118, 0.80901699436} parametreleri için, ilgili polinom düzenlemesi Tablo 4'te gösterildiği gibi yapılacaktır.

Tablo 4. $P_2(x)$ polinomunun *m_list={0,1,2}* için düzenlenmesi

$P_2(0 * h) = y_0 = c_0 + 0 * c_1 * h + 0^2 * c_2 * h^2$
$P_2(1 * h) = y_1 = c_0 + c_1 * h + c_2 * h^2$
$P_2(2 * h) = y_2 = c_0 + 2 * c_1 * h + 2^2 * c_2 * h^2$

2.4.3. Denklem Çözümü ve Alan Hesabı

Örnek noktalar üzerinden düzenlenen polinomlar aracılığı ile oluşturulan katsayı matrisi Tablo 5'te gösterilmiştir.

Tablo 5. Katsayı matrisi - 1

$$\left[\begin{array}{ccc|c} c_0 & c_1 & c_2 & * \\ 1 & 0 & 0^2 & y_0 \\ 1 & 1 & 1^2 & y_1 \\ 1 & 2 & 2^2 & y_2 \end{array}\right]$$

Tablo 5'teki matriste bulunan üslü ifadelerin değerlendirmesi yapılarak Tablo 6'da verilen matris elde edilir.

Tablo 6. Katsayı matrisi - 2

$$\left[\begin{array}{ccc|c} c_0 & c_1 & c_2 & * \\ 1 & 0 & 0 & y_0 \\ 1 & 1 & 1 & y_1 \\ 1 & 2 & 4 & y_2 \end{array}\right]$$

Tablo 6'da oluşturulan katsayı matrisi ve Gauss eliminasyon yöntemi kullanılarak denklem çözümü gerçekleştirilerek c_0, c_1 ve c_2 bilinmeyenleri belirlenmeye çalışılır. Denklem çözüm adımları olarak, "aşağı yönlü indirgeme", "yukarı yönlü indirgeme" ve "birim matrise dönüştürme" işlemleri uygulanmaktadır. Bu işlem adımları Tablo 7'de gösterilmiştir.

Tablo 7. Gauss eliminasyon yönteminin uygulanması

1. satır kullanılarak aşağısındaki (2,1) elemanı 0 yapılır.

$$\left[\begin{array}{ccc|c} c_0 & c_1 & c_2 & * \\ 1 & 0 & 0 & y_0 \\ 0 & 1 & 1 & -y_0 + y_1 \\ 1 & 2 & 4 & y_2 \end{array}\right]$$

1. satır kullanılarak aşağısındaki (3,1) elemanı 0 yapılır.

$$\left[\begin{array}{ccc|c} c_0 & c_1 & c_2 & * \\ 1 & 0 & 0 & y_0 \\ 0 & 1 & 1 & -y_0 + y_1 \\ 0 & 2 & 4 & -y_0 + y_2 \end{array}\right]$$

2. satır kullanılarak aşağısındaki (3,2) elemanı 0 yapılır.

$$\left[\begin{array}{ccc|c} c_0 & c_1 & c_2 & * \\ 1 & 0 & 0 & y_0 \\ 0 & 1 & 1 & -y_0 + y_1 \\ 0 & 0 & 2 & y_0 - 2y_1 + y_2 \end{array}\right]$$

$$\left[\begin{array}{ccc|c} c_0 & c_1 & c_2 & * \\ 1 & 0 & 0 & y_0 \\ 0 & 1 & 0 & -\frac{3}{2}y_0 + 2y_1 - \frac{1}{2}y_2 \\ 0 & 0 & 2 & y_0 - 2y_1 + y_2 \end{array}\right]$$

$$\left[\begin{array}{ccc|c} c_0 & c_1 & c_2 & * \\ 1 & 0 & 0 & y_0 \\ 0 & 1 & 0 & -\frac{3}{2}y_0 + 2y_1 - \frac{1}{2}y_2 \\ 0 & 0 & 1 & \frac{1}{2}y_0 - y_1 + \frac{1}{2}y_2 \end{array}\right]$$

Birim matrise dönüştürme işleminin ardından, katsayı çözümleri Tablo 8'deki gibi belirlenir.

Tablo 8. Katsayı çözümleri

$c_0 = y_0$
$c_1 = \frac{1}{h}(-\frac{3}{2}y_0 + 2y_1 - \frac{1}{2}y_2)$
$c_2 = \frac{1}{h^2}(\frac{1}{2}y_0 - y_1 + \frac{1}{2}y_2)$

Bu aşamada $P_2(x)$ polinomunun Tablo 9'daki gibi $[0,2h]$ aralığında integrali hesaplanır.

Tablo 9. $P_2(x)$ polinomunun integrali

$I = \int_0^{2h} P_2(x)\ dx$
$I = \int_0^{2h} (c_0 + c_1 x + c_2 x^2)\ dx$
$I = (c_0 x + c_1 \frac{x^2}{2} + c_2 \frac{x^3}{3})\Big\|_0^{2h}$

Tablo 9'da x yerine 2h değeri yerleştirildiğinde polinom ifadesinde bulunan $c_k * x^k$ terimlerinin h ortak parantezine alınabildiği görünmektedir. Bu ifade, katsayı çözümleri ile düzenlenerek Tablo 10'da belirlenen çözüme ulaşılır.

Tablo 10. $P_2(x)$ polinomunun integralinin düzenlenmesi

$I=c_0 * (2h) + c_1 \frac{(2h)^2}{2} + c_2 \frac{(2h)^3}{3}$
$I= h * ((y_0) * 2 + (-\frac{3}{2} y_0 + 2y_1 - \frac{1}{2}y_2) * \frac{2^2}{2} + (\frac{1}{2} y_0 - y_1 + \frac{1}{2}y_2) * \frac{2^3}{3})$
$I= h * (\frac{1}{3} y_0 + \frac{4}{3} y_1 + \frac{1}{3}y_2)$
$I= \frac{1}{3} * h * (y_0 + 4y_1 + y_2)$

Ortak paranteze alma işlemi tamamlandıktan sonra, h ve y değerleri ile integralin hesaplanması tamamlanır.

Tablo 11. İntegrali hesaplama

$I= \frac{1}{3} * 0.15707963267 * (1 * 0.58778525229 + 4 * 0.70710678118 + 0.80901699436)$
I= 0. 2212324925

2.5. Tek Parça Yönteminin Parçalı Uygulaması

Tek parça yönteminin parçalı uygulamasını göstermek için belirli sayıda noktayı iki aralığa bölerek her bir aralıkta ayrı ayrı hesaplamalar yapılır. Bu yaklaşım, her bir bölüm için daha doğru bir sonuç elde etmemizi sağlar. Bu şekilde, her parçada elde edilen polinomlar daha küçük bir aralıkta, fonksiyonun davranışını daha iyi yansıtabilir. Örneğin, toplamda 31 nokta (30 parça) kullanılıyorsa, iki parçalı hesaplamada bu parçaların yarısı birinci aralıkta, diğer yarısı ise ikinci aralıkta uygulanabilir. Her bir aralık için polinomları ayrı ayrı hesaplayarak toplam alan bulunur.

Örnek olarak $f(x) = \sin(x)$ fonksiyonunun $[0,30]$ aralığındaki entegrasyon hesabı için toplam 31 nokta seçimi yapıldığını varsayalım. Alan hesabı iki parça üzerinden gerçekleştirilecekse nokta konumları m=[0,1,2…15] ve m=[15,16,17…30] ve adım uzunluğu $h = 1$ olacaktır.

Birinci aralık (ilk 15 parça) için;

$y_i = \sin(x + n[i] * h)$

$y_0 = \sin(0 + 0 * 1) = \sin(0) = 0$

$y_1 = \sin(0 + 1 * 1) = \sin(1) = 0.84147098$

$y_2 = \sin(0 + 2 * 1) = \sin(2) = 0.90929742682$

$y_3 = \sin(0 + 3 * 1) = \sin(3) = 0.14112000806$

$y_4 = \sin(0 + 4 * 1) = \sin(4) = -0.7568024953$

$y_5 = \sin(0 + 5 * 1) = \sin(5) = -0.95892427466$

$y_6 = \sin(0 + 6 * 1) = \sin(6) = -0.27941549819$

$y_7 = \sin(0 + 7 * 1) = \sin(7) = 0.65698659871$

$y_8 = \sin(0 + 8 * 1) = \sin(8) = 0.98935824662$

$y_9 = \sin(0 + 9 * 1) = \sin(9) = 0.41211848524$

$y_{10} = \sin(0 + 10 * 1) = \sin(10) = -0.54402111088$

$y_{11} = \sin(0 + 11 * 1) = \sin(11) = -0.99999020655$

$y_{12} = \sin(0 + 12 * 1) = \sin(12) = -0.536572918$

$y_{13} = \sin(0 + 13 * 1) = \sin(13) = 0.42016703682$

$y_{14} = \sin(0 + 14 * 1) = \sin(14) = 0.99060735569$

$y_{15} = \sin(0 + 15 * 1) = \sin(15) = 0.65028784$

İkinci aralık (son 15 parça) için;

$y_i = \sin(x + n[i] * h)$

$y_{15} = \sin(0 + 15 * 1) = \sin(15) = 0.65028784$

$y_{16} = \sin(0 + 16 * 1) = \sin(16) = -0.28790332$

$y_{17} = \sin(0 + 17 * 1) = \sin(17) = -0.96139749$

$y_{18} = \sin(0 + 18 * 1) = \sin(18) = -0.75098725$

$y_{19} = \sin(0 + 19 * 1) = \sin(19) = 0.14987721$

$y_{20} = \sin(0 + 20 * 1) = \sin(20) = 0.91294525$

$y_{21} = \sin(0 + 21 * 1) = \sin(21) = 0.83665564$

$y_{22} = \sin(0 + 22 * 1) = \sin(22) = -0.00885131$

$y_{23} = \sin(0 + 23 * 1) = \sin(23) = -0.8462204$

$y_{24} = \sin(0 + 24 * 1) = \sin(24) = -0.90557836$

$y_{25} = \sin(0 + 25 * 1) = \sin(25) = -0.13235175$

$y_{26} = \sin(0 + 26 * 1) = \sin(26) = 0.76255845$

$y_{27} = \sin(0 + 27 * 1) = \sin(27) = 0.95637593$

$y_{28} = \sin(0 + 28 * 1) = \sin(28) = 0.27090579$

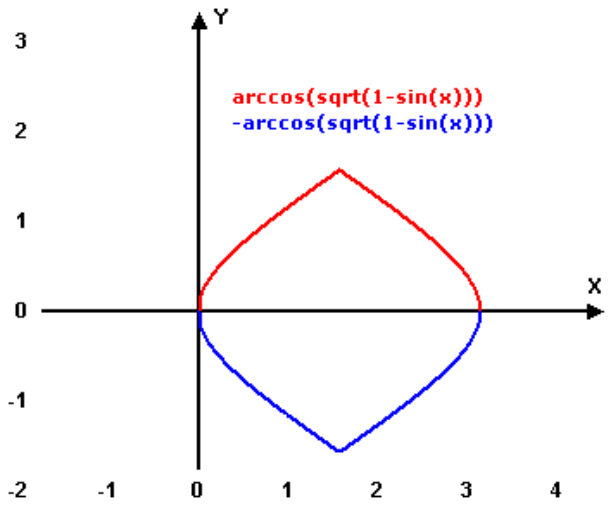
$$y_{29} = \sin(0 + 29 * 1) = \sin(29) = -0.66363388$$
$$y_{30} = \sin(0 + 30 * 1) = \sin(30) = -0.98803162$$

Hesaplanan y değerleri geliştirilen uygulamaya verildiğinde aşağıdaki sonuca ulaşılmaktadır.

$$I_{0-15} = 1.7625724281$$
$$I_{15-30} = -0.9154375652$$
$$I_{0-30} = I_{0-15} + I_{15-30} = 0.8520561389$$

2.6. Kapalı Çevre Alanlarının Hesaplanması

Kapalı çevreler genellikle iki boyutlu şekiller tarafından sınırlanan bölgelere sahiptir. Bu çeşit bölgelerin alanları da, bazı referans noktaları kullanılarak "Tek Parça Yöntemi" ile hesaplanabilmektedir. Örneğin, $\sin(x) + \cos^2(y) = 1$ denklemi ile temsil edilen geometrik şeklin alanını $[0, \pi]$ aralığı için hesaplamaya çalışalım. Şekil 8'de görüldüğü gibi kapalı bölge alanının üst kısmı $f_{üst}(x) = arccos\sqrt{1 - \sin(x)}$, alt kısmı ise $f_{alt}(x) = -arccos\sqrt{1 - \sin(x)}$ fonksiyonu ile temsil edilmektedir.



Şekil 8. $\sin(x) + \cos^2(y) = 1$ denklemi ile oluşturulan kapalı bölge

Alan hesabında Tablo 23'te verilen aşağıdaki 7 noktalı formül ve $h = \pi/6$ değeri ile belirlenen x_i noktaları kullanılacaktır.

$x_i = \left\{0, \frac{\pi}{6}, \frac{2\pi}{6}, \frac{3\pi}{6}, \frac{4\pi}{6}, \frac{5\pi}{6}, \frac{6\pi}{6}\right\}$

$$I = \frac{41h}{140}\left[y_0 + \frac{216}{41}y_1 + \frac{27}{41}y_2 + \frac{272}{41}y_3 + \frac{27}{41}y_4 + \frac{216}{41}y_5 + y_6\right]$$

$y_i = arccos\sqrt{1 - \sin(y_i)}$ ifadesini kullanarak, $[0, \pi]$ arasındaki y değerleri üst kısım için aşağıdaki gibi belirlenecektir.

$y_0 = \arccos\left(\sqrt{1 - \sin(0)}\right) = 0$

$y_1 = \arccos\left(\sqrt{1 - \sin(\frac{\pi}{6})}\right) = 0.7853981633974483$

$y_2 = \arccos\left(\sqrt{1 - \sin(\frac{2\pi}{6})}\right) = \arccos\left(\sqrt{1 - \sin(\frac{\pi}{3})}\right) = 1.1960618940861565$

$y_3 = \arccos\left(\sqrt{1 - \sin(\frac{3\pi}{6})}\right) = \arccos\left(\sqrt{1 - \sin(\frac{\pi}{2})}\right) = 1.5707963267948966$

$y_4 = \arccos\left(\sqrt{1 - \sin\left(\frac{4\pi}{6}\right)}\right) = 1.1960618940861567$

$y_5 = \arccos\left(\sqrt{1 - \sin(\frac{5\pi}{6})}\right) = 0.7853981633974487$

$y_6 = \arccos\left(\sqrt{1 - \sin(\pi)}\right) = 1.4901161193847656$

$I_{üst} = \frac{41h}{140}[0 + \frac{216}{41}(0.7853981633974483) + \frac{27}{41}(1.1960618940861565) + \frac{272}{41}(1.5707963267948966) + \frac{27}{41}(1.1960618940861567) + \frac{216}{41}(0.7853981633974487) + 1.4901161193847656] = 3.108441185486832$

$y_i = -arccos\sqrt{1 - \sin(y_i)}$ ifadesini kullanarak, alt kısım için y değerleri üst kısım için hesaplanan y değerlerinin negatif halleridir.

$y_0 = 0$

$y_1 = -0.7853981633974483$

$y_2 = -1.1960618940861565$

$y_3 = -1.5707963267948966$

$y_4 = -1.1960618940861567$

$y_5 = -0.7853981633974487$

$y_6 = -1.4901161193847656$

$I_{alt} = \frac{41h}{140}\,[0 + \frac{216}{41}(-0.7853981633974483) + \frac{27}{41}(-1.1960618940861565) + \frac{272}{41}(-1.5707963267948966) + \frac{27}{41}(-1.1960618940861567) + \frac{216}{41}(-0.7853981633974487) + -1.4901161193847656]$ = -3.108441185486832

Böylece, üst ve alt alanlar için hesaplanan değerler toplandığında, toplam alana ulaşılacaktır.

I = 3.108441185486832 + | -3.108441185486832 | = 6.216882370973664

2.7. Dokümantasyon

LaTeX, belgeleri biçimlendirmek için metin tabanlı bir işaretleme dilini kullanır. Bu dilde, kullanıcılar belgeleri düzenlemek için metin dosyalarında özel komutlar ve etiketler kullanır. Örneğin, bir metin paragrafı başlatmak için "\begin{document}" ve "\end{document}" komutları arasına yazılarak LaTeX belgesi oluşturulabilir. LaTeX, kullanıcıların içerik üzerinde çok ince ayarlar yapmasına ve karmaşık matematiksel ifadeleri kolayca yazmasına olanak tanır. LaTeX belgeleri genellikle tex uzantılı metin dosyalarında yazılır ve daha sonra LaTeX derleyicisi kullanılarak PDF veya diğer formatlara dönüştürülür. LaTeX'in esnekliği ve gücü, özellikle matematiksel içeriği olan belgeler için tercih edilmesini sağlar.

Soyut sözdizim ağacı (AST)'nın oluşturulması bölümünde de belirtildiği gibi entegrasyon ifadelerinin her bir farklı bileşeni (toplama, üst alma, parantezleme, vb.) için farklı bir Java sınıfı kullanılır. Örneğin, toplama işlemlerine yönelik olarak tanımlanan Plus sınıfı Tablo 12'de gösterilmektedir. Bu sınıfa eklenen toString() ve toLatex() metotları ile sözdizim ağacındaki Plus türünden düğümler için sırasıyla matematiksel notasyona ve LaTeX formatlama diline dönüşüm yapılır.

Tablo 12. Plus sözdizim sınıfı

```
class Plus extends Exp {
  Exp e1, e2;
  public Plus(Exp a, Exp b) {
    e1 = a;
    e2 = b;
  }

  public String toString() {
    return e1 + "+" + e2;
  }

  public String toLatex(boolean x) {
    return e1.toLatex(x) + "+" + e2.toLatex(x);
  }
}
```

Benzer şekilde, AST'nin kök (root) düğümü ile ilgili olarak Texp sınıfı ve dönüşüm metotları Tablo 13'de tanımlanmıştır.

Tablo 13. Texp sözdizim sınıfı

```
class Texp {
  Exp e1, e2;
  public Texp(Exp a, Exp b) {
    e1 = a;
    e2 = b;
  }

  public String toString() {
    return e1 + "=" + e2;
  }

  public String toLatex(boolean x) {
    return e1.toLatex(x) + "=" + e2.toLatex(x);
  }
}
```

Tüm AST düğümlerinde bulunan nesne verileri üzerinden yapılan dönüşümler ile birlikte matematiksel gösterimler ve Latex dili için gereken dokümanlar hazırlanır. Tablo 14'te verilen Java kod bloğunda, toText ve toPdf koşulları ile polinomların gösterimini sağlayacak doküman içerikleri düzenlenmektedir. MiKTeX kütüphanesinde bulunan pdflateX aracı PDF dokümanlarının üretimi gerçekleştirilir.

Tablo 14. Matematiksel gösterimler ve LaTeX dili için polinomların düzenlemesi

```
Texp[] e = fd.polynomial();
for (int i=0; i<e.length; i++) {
        if (toText)
                stringBuilder
        .append(e[i])
        .append("\n");
        if (toPdf)
                latexStringBuilder
                .append("$\\displaystyle P_{")
                .append(e.length - 1)
                .append("}(")
                .append(fd.mh[i])
                .append("*h)=")
                .append(e[i].toLatex(false))
                .append("$\\\\")
                .append("\n");
}
```

2.7.1. Entegrasyon

Programcı girdilerinin doğruluğunu kontrol ederek Java programlama çevrelerine entegrasyonu sağlamak amacıyla RequestDTO isimli bir sınıf oluşturulmuştur. Bu sınıf, String tipinde n, double tipinde h, double[] tipinde yValues, boolean tipinde toText ve toPdf değişkenlerine sahiptir. Tablo 15'te gösterildiği gibi, girdiler uygulama arayüzünden alınıp, ilgili hesaplamalar için gerekli metot çağrımlarının yapılacağı ApiService isimli sınıf içerisindeki process() metoduna gönderilmektedir.

Tablo 15. Girdi değerlerinin kullanılması

```
public ResponseDTO process (RequestDTO requestDTO) {
        String mTxt = requestDTO.getN();
        String[] strArr = mTxt.split(",");
        int[] m = new int[strArr.length];
        for(int i=0; i<strArr.length; i++) {
                m[i] = Integer.parseInt(strArr[i]);
        }
        Table table = new Table();
        double h = requestDTO.getH();
        table.put("h", new Ratio(h));
        var yDoubleVals = requestDTO.getYvalues();
        for (int i=0; i<yDoubleVals.length; i++) {
                table.put("y"+i, new Ratio(yDoubleVals[i]));
        }

        return run(table, m, requestDTO.isToText(),
requestDTO.isToPdf());
}
```

Uygulamanın ana çatısını oluşturan metotlar PInteg sınıfıyla tedarik edilmektedir. Sayısal integral ifadelerinin türetilmesi ve değerlendirilmesi sürecinin temel adımları farklı metotlar tarafından yönetilmektedir. Metot çağrım dizilerinin bazıları Tablo 16'da sunulmuştur.

Tablo 16. Metot çağrımları ile entegrasyon

```
public void run(int[] m) {
        PInteg fd = new PInteg(m);
        Texp[] e = fd.polynomial();
        Exp[][] e2 = fd.coeffMatrix(e);
        Exp[][] e3 = fd.integMatrix(e2);
        Exp[][] e3 = fd.gaussMatrix(e3);
        Exp[][] e3 = fd.identityMatrix(e3);
        Exp p = fd.polynomial_integ();
        ...
}
```

Tablo 16'da kullanılan PInteg sınıfı, entegrasyon süreçlerinde ihtiyaç duyulan tüm metotları barındırmaktadır. Bu sınıf temel olarak arayüzde tanımlanan örnek noktalar listesini kendisine parametre olarak almaktadır. Bu sınıf içerisinde tanımlı diğer metotlar yardımı ile Text ve Exp türünden nesneler üretilmektedir. toString() ve toLatex() gibi metotlar ile bu nesnelerin verileri dokümana aktarılmaktadır.

3. BULGULAR VE TARTIŞMA

Bu bölümde, Matlab kullanılarak farklı sayısal integral hesaplama yöntemleri ile ayrık olarak belirtilen ya da fonksiyon örneklemelerinden alınan noktalar kümesi üzerinde entegrasyon işlemi gerçekleştirilmiştir. Entegrasyon sonuçları tez kapsamında geliştirilen uygulamanın hesaplama sonuçları ile karşılaştırılmıştır.

3.1. Sayısal İntegral Hesaplama Yöntemlerinin Karşılaştırılması

Tablo 17'de 3 noktası bilinen (iki parçalı) bir bölgenin entegrasyon hesaplaması gösterilmiştir.

Tablo 17. Yöntemsel karşılaştırma - 1

```
% Örnek m ve y değerleri
m= [0, 1, 2];
y = [0.58778525229, 0.70710678118, 0.80901699436];

% Adım uzunluğu
h = 0.15707963267;

% Sol Rieman Yöntemi
I_riemann_left = h * sum(y(1:end-1));

% Orta Nokta Yöntemi
y_mid = (y(1:end-1) + y(2:end)) / 2;
I_midpoint = h * sum(y_mid);

% Yamuk Kuralı
I_trapezoidal = h * (sum(y) - 0.5 * (y(1) + y(end)));

% Simpson's 1/3 Kuralı
n = length(m) - 1;
if mod(n, 2) == 0  % n çift olmalı
   I_simpson13 = (h/3) * (y(1) + y(end) + 4 * sum(y(2:2:end-1)) + 2 *
sum(y(3:2:end-2)));
else
   I_simpson13 = NaN; % n çift değilse Simpson's 1/3 uygulanamaz
end

% Simpson's 3/8 Kuralı
if mod(n, 3) == 0  % n 3'ün katı olmalı
   I_simpson38 = (3*h/8) * (y(1) + y(end) + 3 * sum(y(2:3:end-1)) + 3 *
sum(y(3:3:end-1)) + 2 * sum(y(4:3:end-2)));
Else
```

Tablo 17’nin devamı

```
   I_simpson38 = NaN; % n 3'ün katı değilse Simpson's 3/8 uygulanamaz
end

% Boole Kuralı
if mod(n, 4) == 0  % n 4'ün katı olmalı
   I_boole = (2*h/45) * (7*y(1) + 7*y(end) + 32 * sum(y(2:4:end-1)) + 12 *
sum(y(3:4:end-2)) + 32 * sum(y(4:4:end-3)));
else
   I_boole = NaN; % n 4'ün katı değilse Boole kuralı uygulanamaz
end

% Tek parça kuralı (2. dereceden polinom)
I_App = (h/3) * (y(1) + y(end) + 4 * sum(y(2:2:end-1)) + 2 *
sum(y(3:2:end-2)));

% Sonuçları tablo halinde yazdır
T = table(I_riemann_left, I_midpoint, I_trapezoidal, I_simpson13,
I_simpson38, I_boole, I_App);
disp(T);
```

Riemann: 0.2034
Orta Nokta: 0.22078
Trapez: 0.22078
Simpson 1/3: 0.22123
Simpson 3/8: Hesaplanamaz (n 3’ün katı değil)
Boole 3/8: Hesaplanamaz (n 4’ün katı değil)
Tek Parça Kuralı: 0.22123

Kod içerisinde tanımlanmış n değeri entegrasyon aralığının kaç parçaya bölündüğünü ifade etmektedir. İlgili yöntemlerin uygulanabilirliği için, n değeri seçilen yöntemin koşullarını sağlaması gerekmektedir. Tablo 17’de verilen kısmi örnekleme noktaları [0, 1, 2] için aralık sayısı (n değeri) 2’dir. Simpson 3/8 kuralı n değerinin 3’ün katı olması ve Boole kuralının 4’ün katı olması gerekliliğini zorunlu kıldığı için ilgili hesaplamalar gerçekleştirilememiştir. Uygulamada kullanılan tek parça kuralı ile Simpson 1/3 kuralı, aynı ifadeye sahip olduğundan aynı I=0.2212324925 değerini hesaplamaktadır.

Tablo 18’de 7 noktası bilinen (altı parçalı) bir bölgenin integrasyon hesaplaması gösterilmiştir.

Tablo 18. Yöntemsel karşılaştırma - 2

```
% Örnek m ve y değerleri (7 nokta, 6 aralık)
m = [0, 1, 2, 3, 4, 5, 6];
y = [0.58778525229, 0.70710678118, 0.80901699436, 0.89100652419,
0.95105651629, 0.98768834059, 1.00000000000];

% Adım uzunluğu
h = 1; % Burada h = 1 olarak alınmıştır.

% Simpson's 1/3 Kuralı (2. dereceden polinom)
% [0,2], [2,4] ve [4,6] aralıkları için hesaplamalar
I_simpson13_0_2 = (h/3) * (y(1) + 4*y(2) + y(3));
I_simpson13_2_4 = (h/3) * (y(3) + 4*y(4) + y(5));
I_simpson13_4_6 = (h/3) * (y(5) + 4*y(6) + y(7));

% Toplam Simpson's 1/3 integrali
I_simpson13_total = I_simpson13_0_2 + I_simpson13_2_4 +
I_simpson13_4_6;

% Simpson's 3/8 Kuralı (3. dereceden polinom)
% [0,3] ve [3,6] aralıkları için hesaplamalar
I_simpson38_0_3 = (3*h/8) * (y(1) + 3*y(2) + 3*y(3) + y(4));
I_simpson38_3_6 = (3*h/8) * (y(4) + 3*y(5) + 3*y(6) + y(7));

% Toplam Simpson's 3/8 integrali
I_simpson38_total = I_simpson38_0_3 + I_simpson38_3_6;

% Boole Kuralı (4. dereceden polinom) uygulanamaz çünkü parça sayısı 4'ün
katı değil.
I_boole = NaN;

% Tek parça kuralı (6. dereceden polinom)
I_App= h/140 *
[41*y(1)+216*y(2)+27*y(3)+272*y(4)+27*y(5)+216*y(6) + 41*y(7)]

% Sonuçları tablo halinde yazdır
T = table(I_simpson13_0_2, I_simpson13_2_4, I_simpson13_4_6,
I_simpson13_total, I_simpson38_0_3, I_simpson38_3_6, I_simpson38_total,
I_boole, I_App);
disp(T);
```

Simpson 1/3 (2. derece)

[0-2]: 1.4084

[2-4]: 1.7747

[4-6]: 1.9673

Toplam:5.1504

Tablo18'in devamı

Simpson 3/8 (3. derece) [0-3]: 2.2602 [3-6]: 2.8902 **Toplam:5.1504** **Tek parça kuralı** (6. Derece) **toplam:5.1504**

Tablo 18'de görüldüğü gibi, 2. dereceden polinomla belirlenen Simpson kuralı kullandığında, [0,2], [2,4] ve [4,6] aralıkları için ayrı hesaplamalar yapılıp bu sonuçlar toplandığında 5.1504 sonucuna ulaşılmaktadır. 3. dereceden polinomla belirlenen Simpson 3/8 kuralı için [0,3] ve [3,6] aralıkları için yapılan ayrı hesaplamaların sonuçlarının toplanması ile 5.1504 değeri bulunmaktadır. Bu entegrasyona, parça sayısı 4'ün katı olmadığı için 4. dereceden polinomla belirlenen Boole kuralını uygulayamayız. Tek parça kuralının 6. dereceden bir polinomla hesapladığı ifade, [0,6] aralığında tek bir hesaplama yaparak 5.1503621500 sonucunu üretmektedir. Hesaplanan sonuçlar Tablo 19'da karşılaştırılmıştır.

Tablo 19. Sonuç karşılaştırma tablosu – 1

Simpson 1/3	Simpson 3/8	Boole (Matlab)	Tek parça kuralı
5.1504	5.1504	5.1398	5.1503621500

Diğer bir karşılaştırma için belirli bir fonksiyon ($y = \sin(x)$, $x_0 = 0$) üzerinde $m = [0, 1, 2, 3]$ ile örnekleme yapıldığında, adım boyutu $h = 0.01$ değeri kullanılarak aşağıda gösterilen y değerleri hesaplanır.

y(0) = sin(0) = 0

y(1) = sin(0 + 1*0.01) = sin(0.01) = 0.00999983333

y(2) = sin(0 + 2*0.01) = sin(0.02) = 0.01999866669

y(3) = sin(0 + 3*0.01) = sin(0.03) = 0.02999550020

Analitik çözüm için

$x = x_0 + m * h$

$m = 0\ için\ x = 0 + 0 * 0.01 = 0$

$m = 3\ için\ x = 0 + 3 * 0.01 = 0.03$

$$I = \int_0^{0.03} (\sin(x))\ dx = (-\cos(x))\ \Big|_0^{0.03} = -\cos(0.03) + \cos(0) = 0.000449$$

bulunur.

Tablo 20. Sonuç karşılaştırma tablosu – 2

Riemann	Orta Nokta Yaklaşımı	Trapez	Simpson 3/8	Tek Parça Kuralı	Gerçek değer
0.00029999	0.00044996	0.00044996	0.00022498	0. 00044996	0.00044996

Tablo 20'de gösterilen sonuçlara göre, Orta nokta yaklaşımı, Trapez ve tek parça ile hesaplanan değerlerin, gerçek değere daha yakın olduğu, Riemann yönteminde ise hata oranının yüksek olduğu belirlenmiştir. Simpson 1/3 kuralı ve Boole yöntemi ile hesaplama yapılamamıştır.

3.2. Polinom Derecesinin Entegrasyon Sonucu Üzerindeki Etkisinin Değerlendirilmesi

Polinomun derecesi, entegrasyon sonucuna doğrudan etki etmektedir. Genel olarak, yüksek dereceli polinomlar daha karmaşık entegrasyon hesaplamalarına yol açabilirken, düşük dereceli polinomlar daha basit ve hızlı çözümler sunabilmektedir. Polinomun derecesinin entegrasyon sonucuna etkisini göstermek için aşağıdaki hesaplama işlemleri gerçekleştirilmiştir.

Fonksiyon: x^5

$x_0 = 2$

m: [0, 1, 2, 3, 4, 5, 6]

h (aralık uzunluğu): 0.01

Yukarıdaki girdiler yardımı ile x^5 fonksiyonu için, y değerleri aşağıdaki gibi hesaplanmaktadır.

$$y_i = (x + n[i] * h)^5$$

$$y_0 = (2 + 0 * 0.01)^5 = 32$$

$$y_1 = (2 + 1 * 0.01)^5 = 32.8080401001$$

$$y_2 = (2 + 2 * 0.01)^5 = 33.6323216032$$

$$y_3 = (2 + 3 * 0.01)^5 = 34.4730881243$$
$$y_4 = (2 + 4 * 0.01)^5 = 35.3305857024$$
$$y_5 = (2 + 5 * 0.01)^5 = 36.2050628125$$
$$y_6 = (2 + 6 * 0.01)^5 = 37.0967703776$$

Aynı parametreler ile x^{10} fonksiyonu değerlendirilmiş ve y değerleri aşağıdaki gibi hesaplanmıştır.

$$y_0 = (2 + 0 * 0.01)^{10} = 1024$$
$$y_1 = (2 + 1 * 0.01)^{10} = 1076.36749521$$
$$y_2 = (2 + 2 * 0.01)^{10} = 1131.13305642$$
$$y_3 = (2 + 3 * 0.01)^{10} = 1188.39380483$$
$$y_4 = (2 + 4 * 0.01)^{10} = 1248.25028607$$
$$y_5 = (2 + 5 * 0.01)^{10} = 1310.80657326$$
$$y_6 = (2 + 6 * 0.01)^{10} = 1376.17037245$$

Bu y değerleri ile türetilen entegrasyon ifadeleri çözümlendiğinde Tablo 21'de yer alan değerlere ulaşılmaktadır.

Tablo 21. Sonuç karşılaştırma tablosu – 3

	Riemann	Orta Nokta Yaklaşımı	Trapez	Simpson 1/3	Simpson 3/8	Tek Parça Kuralı	Gerçek Değer
x^5	2.0445	2.07	2.07	2.0699	2.0699	2.06989116	2.06989116
x^{10}	69.79	71.55	71.55	71.537	71.537	71.53736065	71.5373606

Farklı derecelere sahip polinomlar için alınan sonuçlar karşılaştırıldığında, fonksiyonun derecesinin yükselmesi durumunda gerçek değere en yakın sonucu Simpson 1/3, Simpson 3/8 ve Tek parça kuralının verdiği görülmüştür. Özellikle Riemann kuralı için gerçek değerden oldukça uzaklaşıldığı tespit edilmiştir. Boole yöntemi ile hesaplama yapılamamıştır. Bu sonuç, yüksek karmaşıklığın olduğu problemler için polinomlara dayalı yaklaşımları önemli kılmaktadır.

Benzer şekilde, ilgili fonksiyon için 2. dereceden, 10. dereceye kadar polinomlarla entegrasyon ifadelerinin üretilmesi ve farklı sayısal entegrasyon hesaplama yöntemleri ile elde edilen sonuçların değerlendirmesi yapılmıştır. Seçilen polinomlar için gerçek değer hesabı yapılarak, gerçek değerler ile diğer yöntemlerin verdiği sonuçlar karşılaştırılmıştır.

Fonksiyonlar: x^2- x^{10}

$x_0 = 0$

m: [0-30]

h (aralık uzunluğu): 0.1

Gerçek değerlerin hesaplanma adımları x^2 polinomu referans alınarak 31 noktalı hesaplama işlemleri aşağıda gösterilmiştir.

$x = x_0 + m * h$

$m = 0\ için\ x = 0 + 0 * 0.1 = 0$

$m = 30\ için\ x = 0 + 30 * 0.1 = 3$

$$I = \int_0^3 (x^2)\ \mathrm{dx} = \left(\frac{x^3}{3}\right)\Big|_0^3 = 9$$

Tablo 22. Farklı dereceli polinomlar entegrasyonlarını karşılaştırılması

	Riemann	Orta Nokta Yaklaşımı	Trapez	Simpson 1/3	Simpson 3/8	Tek Parça Kuralı	Gerçek Değer
x^2	8.555	9.005	9.005	9	9	9	9
x^3	18.922	20.273	20.273	20.25	20.25	20.25	20.25
x^4	44.64	48.69	48.69	48.6	48.6	48.6	48.6
x^5	109.69	121.84	121.84	121.5	121.5	121.5	121.5
x^6	277.19	313.64	313.64	312.43	312.43	312.42857	312.42857
x^7	715.03	824.38	824.38	820.13	820.15	820.125	820.125
x^8	1873.5	2201.6	2201.6	2187	2187.1	2187	2187
x^9	4969.9	5954.1	5954.1	5905.1	5905.4	5904.9	5904.9
x^{10}	13316	16268	16268	16105	16106	16104.272	16104.272

Tablo 22'de yer alan sonuçlar değerlendirildiğinde;

Riemann yöntemi, genellikle basit ve hızlı bir yaklaşım sunar. Ancak, özellikle yüksek dereceli polinomlarda gerçek değer ile elde edilen sonuçlar arasında önemli farklar gözlemlenmektedir. Örneğin, x^8 için 1873.5 olarak elde edilen sonuç, gerçek değerden (2187) oldukça uzaktır. Bu durum, Riemann yönteminin yüksek dereceli polinomlar için yeterince hassas olmadığını göstermektedir. Orta nokta yaklaşımı genellikle Riemann yöntemine göre daha iyi bir sonuç verir, çünkü fonksiyonun ortasında bir nokta kullanarak hesaplama yapar. Ancak, yine de yüksek dereceli polinomlarda hata payı artmaktadır. Örneğin, x^6 için 313.64 değeri, gerçek değerden belirgin bir sapma göstermektedir. Orta nokta yaklaşımı, düşük dereceli polinomlarda daha iyi performans gösterse de, yüksek dereceli polinomlarda başarısı düşmektedir. Trapez yöntemi, entegrasyon işlemi için daha iyi bir tahmin sağlar çünkü iki uç nokta arasındaki alanı trapezle yaklaşıklar. Tabloya bakıldığında, sonuçları genellikle Riemann ve Orta Nokta Yaklaşımı'ndan daha doğru çıkmaktadır. Simpson 1/3 yöntemi ikinci dereceden polinomlar için oldukça etkili bir

sonuç vermektedir. Gerçek değerle karşılaştırıldığında, özellikle düşük ve orta dereceli polinomlar için yakın sonuçlar elde edilmektedir. Örneğin, x^5 ve x^6 için hesaplanan değerler, gerçek değere oldukça yakındır. Ancak, yüksek dereceli polinomlarda (özellikle x^8 ve sonrasında) hata payı yine artmaktadır, fakat bu artış diğer yöntemlere göre daha az belirgin olmaktadır. Simpson 3/8 yöntemi bu çalışma içerisinde gerçek sonuca oldukça uzak sonuçlar üretmiştir. Gerçek değere en yakın sonuçlar bir önceki bölümde yapılan tespit ile benzer şekilde, tek parça kuralı ile alındığı görülmüştür.

3.3. Polinom Derecesine Göre Türetilen İntegral İfadeleri

Tek parça kuralı 2. dereceden ve 10. dereceye kadar polinomlarla uygulanarak belirlenen ifadeler Tablo 23'te listelenmiştir.

Tablo 23. Polinom derecesine göre türetilen integral ifadeleri

Derece	İntegral ifadesi
0	$h[y_0]$
1	$\frac{h}{2}\,[y_0 + y_1]$
2	$\frac{h}{3}\,[y_0 + 4y_1 + y_2]$
3	$\frac{3h}{8}\,[y_0 + 3y_1 + 3y_2 + y_3]$
4	$\frac{14h}{45}\,[y_0 + \frac{32}{7}y_1 + \frac{12}{7}y_2 + \frac{32}{7}y_3 + y_4]$
5	$\frac{95h}{288}\,[y_0 + \frac{75}{19}y_1 + \frac{50}{19}y_2 + \frac{50}{19}y_3 + \frac{75}{19}y_4 + y_5]$
6	$\frac{41h}{140}\,[y_0 + \frac{216}{41}y_1 + \frac{27}{41}y_2 + \frac{272}{41}y_3 + \frac{27}{41}y_4 + \frac{216}{41}y_5 + y_6]$
7	$\frac{5257h}{17280}\,[y_0 + \frac{3577}{751}y_1 + \frac{1323}{751}y_2 + \frac{2989}{751}y_3 + \frac{2989}{751}y_4 + \frac{1323}{751}y_5 + \frac{3577}{751}y_6 + y_7]$
8	$\frac{3956h}{14175}\,[y_0 + \frac{256}{43}y_1 - \frac{928}{989}y_2 + \frac{10496}{989}y_3 - \frac{4540}{989}y_4 + \frac{10496}{989}y_5 - \frac{928}{989}y_6 + \frac{256}{43}y_7 + y_8]$
9	$\frac{25713h}{89600}\,[y_0 + \frac{15741}{2857}y_1 + \frac{1080}{2857}y_2 + \frac{19344}{2857}y_3 + \frac{5778}{2857}y_4 + \frac{5778}{2857}y_5 + \frac{19344}{2857}y_6 + \frac{1080}{2857}y_7 + \frac{15741}{2857}y_8 + y_9]$

Tablo 23'ün devamı

10	$\frac{80335h}{299376}\left[y_0+\frac{106300}{16067}y_1-\frac{48525}{16067}y_2+\frac{272400}{16067}y_3-\frac{260550}{16067}y_4+\frac{427368}{16067}y_5-\frac{260550}{16067}y_6+\frac{272400}{16067}y_7-\frac{48525}{16067}y_8+\frac{106300}{16067}y_9+y_{10}\right]$

3.4. Formül Hesaplama Karmaşıklığı

Sayısal entegrasyon yöntemlerinde, belirli bir aralık için yapılacak hesaplamalarda daha küçük adım büyüklüğü (h) seçilmesi daha fazla sayıda fonksiyon noktasının (y_i) kullanılmasını gerektirir. Bu durum, bir yandan daha hassas sonuçlar elde edilmesine olanak sağlarken diğer yandan hesaplamanın içereceği işlem sayısını da artırmaktadır.

Genel olarak sayısal entegrasyon formüllerinin hesaplama maliyeti, nokta sayısına (n) bağlı olarak artar. O(n) ile temsil edilen bu maliyet, hesaplamanın zaman karmaşıklığının bir ölçüsüdür. Tek Parça Yöntemi'nde, polinom denklemlerinin üretiminden entegrasyon formülünün elde edilmesine kadar geçen süreç, çeşitli sayıda işlemlerden oluşmaktadır. Bu işlemler, nokta sayısı (n) ile doğrudan ilişkilidir ve her bir adımın hesaplama maliyeti artan n ile değişiklik gösterir.

Polinomun katsayılarını bulmak için yapılan her bir hesaplama, n terimi içerir. Bu terimler katsayılarla çarpılarak sonuçları toplandığından, çarpma ve toplama işlemlerinin toplam sayısı O(n^2) olacaktır. Ayrıca, n sayıda bölme işlemi gerçekleştirilmektedir. Bu da toplamda O(n) bölme işlemi demektir.

Gauss eleminasyon yöntemi ile n bilinmeyenli bir sistemin çözümünde işlem maliyeti O(n^3) olarak ifade edilir. Bu karmaşıklık, pivot seçimi, satır değiştirme ve diğer satır işlemleri sebebi ile oluşmaktadır. Maliyet, n pivot için O(n)'dir. Her pivot işleminde, diğer satırlar pivot satırıyla karlaştırılarak güncellendiği için, satır işlemlerindeki maliyet O(n^2)'dir. Genel olarak, n denklem ve n bilinmeyen içeren bir sistemin çözümü için yapılan işlem sayısı O(n^3) olarak ifade edilir. Bu işlem şu şekilde özetlenebilir.

Bölme İşlemleri: $\frac{n(n+1)}{2}$ bölme işlemi yapılır.

Çarpma ve Çıkarma İşlemleri: $\frac{2n^3+3n^2-5n}{6}$ çarpma ve çıkarma işlemi gerçekleştirilir.

Polinom katsayılarının belirlenmesinde kullanılan bu Gauss eliminasyon adımı, tüm sistemin karmaşıklığını etkileyen önemli bir bileşendir. Polinom üretiminde yer alan

katsayı matrisinin çözümü, O(n^3) maliyetli olduğu için, toplam işlem karmaşıklığı üzerinde doğrudan bir etkiye sahiptir.

4. SONUÇLAR

Bu tezde, polinomlara dayalı yaklaşımlarla sayısal entegrasyon ifadelerinin üretilmesi, değerlendirilmesi ve dokümantasyonu üzerinde çalışmıştır. Çalışma sonuçlarına ve çıktılarına göre, polinom tabanlı yöntemlerin sayısal entegrasyon problemlerinde önemli bir rol oynayabileceğini göstermektedir. Sonuçlar, bu yöntemlerin doğruluk ve hesaplama performansı açısından diğer geleneksel yöntemlere kıyasla önemli avantajlar sunduğunu göstermektedir. Bu avantajlar, özellikle karmaşık entegrasyon problemleriyle karşılaşıldığında belirgin hale gelmektedir.

Tez kapsamında polinom tabanlı yaklaşımlar ile entegrasyon ifadelerinin üretim süreci üzerine de odaklanılmıştır. Bu yaklaşımların sayısal entegrasyon ifadelerinin üretilmesindeki esnekliği ve kullanım kolaylığı, araştırmacıların çeşitli entegrasyon problemlerini çözmelerine olanak tanımaktadır. Bu da, bu yöntemlerin geniş bir uygulama alanına sahip olabileceğini göstermektedir.

Son olarak, yapılan çalışmanın dokümantasyonu ve sonuçlarının paylaşılması önemlidir. Çalışma bulgularının bilimsel topluluğa sunulması, polinomlara dayalı yaklaşımların sayısal entegrasyon alanındaki potansiyelini daha geniş bir kitleye ulaştıracaktır. Bu şekilde, gelecekteki araştırmacılar ve programcılar, bu yöntemlerden en iyi şekilde yararlanabilecek ve sayısal entegrasyon problemlerinin çözümünde daha etkili stratejiler geliştirebilecektir.

5. ÖNERİLER

Bu çalışmada, polinomlara dayalı yaklaşımlar ile analitik çözümlerin, performans ve doğruluk açısından karşılaştırması incelenmiş ve edinilen sonuçlara göre aşağıdaki önerilerin yapılması uygun görülmüştür.

- Matematiksel gösterimlere uygun biçimde PDF dokümanlarının içeriğine benzer web içeriğinin oluşturulması (pdf vb.) sonrası bu içeriğin tarayıcıda gösterilmesi sağlanabilir.
- Uygulama seviyesinde alınan hatalar için exception mekanizmasının daha iyi düzeye getirilmesi amacıyla global bir exception handler oluşturularak, fırlatılabilecek tüm hatalar yakalanabilir. Bu iyileştirme sonucunda, kullanıcılar ve api hizmetinden yararlanan diğer uygulamalar, alınacak hatalar için daha anlamlı mesajlar ile karşılaşabilecektir.
- Üretilen yazılımın api dokümantasyonun geliştirilmesi amacıyla OpenAPI (swagger) tercih edilmiştir. OpenAPI, daha geniş ölçekli ve yüksek miktarda endpoint'e sahip uygulamalar için oldukça verimli olmasına rağmen, tez kapsamında geliştirilen yazılım için oldukça büyük ölçekli bir üründür. Bu bağlamda, özelleştirilmiş bir api dokümantasyonu oluşturularak, daha sade ve anlaşılır bir api arayüzü oluşturulabilir.
- Uygulamanın arayüz tasarımı oluşturulurken tercih edilen bootstrap kütüphanesi her ne kadar popüler bir kütüphane olsa da yalnızca 1 adet arayüze sahip bir yazılım için oldukça büyük boyutlu javascript ve css dosyalarına sahiptir. Bu nedenle, bu tez kapsamında geliştirilen uygulama özelinde daha küçük boyuta sahip javascript ve css dosyaları yaratılarak, uygulamanın, kullanıcıların tarayıcısında daha hızlı yüklenmesi sağlanabilir.

6. KAYNAKLAR

Pehlivan, H. (2019). Sayısal Çözümleme Yöntemlerinin Programlanması ve Yorumlanması. Düzce Üniversitesi Bilim Ve Teknoloji Dergisi, 7(1), 872-894. https://doi.org/10.29130/dubited.509310

Emiris, I.Z. (1997). Symbolic-numeric algebra for polynomials. Encyclopedia of Computer Science and Technology, A. Kent and JG Williams, editors, 39, 261.

Gauss Sayısal İntegrasyon Yöntemi, May. 20, 2024. [Online]. Erişim: http://acikerisim.harran.edu.tr:8080/jspui/bitstream/11513/3176/1/675935.pdf

MİLANİ, Muhammed, Özlem ORHAN, and Bahar MİLANİ. "Sembolik Hesaplama Metodolojisine Dayalı Bazı Matematiksel Problemlerin Otomatik Çözümü." Electronic Letters on Science and Engineering 16.2(2020):130-142.

Yeh, S. T. (2002). Using trapezoidal rule for the area under a curve calculation. Proceedings of the 27th SAS ® User Group International (SUGI'02), 1-5.

Fornberg, B. (2001). Improving the accuracy of the trapozoidal rule. SIAM Review, 63(1), 167-180.

Pehlivan, H.(2019). Designing and interpreting a mathematical programming language. Sakarya University Journal of Science, 23(6), 1027-1041

Wolfram,(5 Şubat 2024). [Online]. Erişim: http://www.wolfram.com.

EFENDIOGLU, Seda. Polinomlar için bir simgesel hesaplama çatısının tasarımı ve gerçeklenmesi. 2017. Master's Thesis. Fen Bilimleri Enstitüsü.

Davis, Philip J., and Philip Rabinowitz. Methods of numerical integration. Courier Corporation, 2007.

Moheuddin, M. M., Titu, M. A. S., & Hossain, S. (2020). A new analysis of approximate solutions for numerical integration problems with quadrature-based methods. *Pure and Applied Mathematics Journal*, *9*(3), 46-54.

Zheng, C., Hu, J., Du, Q., & Zhang, J. (2017). Numerical solution of the nonlocal diffusion equation on the real line. SIAM Journal on Scientific Computing, 39(5), A1951-A1968.

Yang WY, Cao W, Chung TS, Morris J. Applied numerical method using Matlab. John Wiley & Sons, Inc. Publication; 2005.

Sinha, R. K., & Kumar, R. (2010). Numerical method for evaluating the integrable function on a finite interval. *International Journal of Engineering Science and Technology*, *2*(6), 2200-2206.

Das, S. C. (1956, July). The numerical evaluation of a class of integrals. II. In *Mathematical Proceedings of the Cambridge Philosophical Society* (Vol. 52, No. 3, pp. 442-448). Cambridge University Press.

Atkinson, K. (1991). *An introduction to numerical analysis*. John wiley & sons.

Davis, P. J., & Polonsky, I. (1972). 25. Numerical Interpolation, Differentiation, and Integration. *Numerical Interpolation Differentiation and Integration*.

Matlab,(8 Haziran 2024). [Online]. Erişim: https://matlab.mathworks.com/.

Delves, L. M. (1968). The numerical evaluation of principal value integrals. *The Computer Journal*, *10*(4), 389-391.

Ding, Y., Zhu, L., Zhang, X., & Ding, H. (2011). Numerical integration method for prediction of milling stability.

Smyth, G. K. (1998). Numerical integration. Encyclopedia of biostatistics, 3088-3095.

Fauziyah, W. N., Irmansyah, A. Z., & Purwani, S. (2021). Numerical Integration Implementation Using Trapezoidal Rule Method to Calculate Aproximation Area of West Java Province. International Journal of Quantitative Research and Modeling, 2(2), 117-124.

Casas, S., Cruz, D., Vidal, G., & Constanzo, M. (2021, November). Uses and applications of the OpenAPI/Swagger specification: a systematic mapping of the literature. In *2021 40th International Conference of the Chilean Computer Science Society (SCCC)* (pp. 1-8). IEEE.

Herceg, D., & Herceg, D. (2010). Numerical integration with GeoGebra in high school. International Journal for Technology in Mathematics Education, 17(4), 205-210.

ÖZGEÇMİŞ

Özgür AKINCI, İlköğrenimini Ayfer Karakullukçu İlköğretim Okulu'nda bitirdikten sonra lise eğitimini 2011 yılında Trabzon (Anadolu) Lisesi'nde tamamladı. 2010-2011 öğretim yılında Karabük Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliği bölümünde lisans eğitimine başladı. 2016 yılında bu bölümden mezun oldu. 2021-2022 öğretim yılında ise Karadeniz Teknik Üniversitesi, Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği Anabilim dalında tezli yüksek lisans programına başladı. 2024 yılında "Polinomlara Dayalı Yaklaşımlarla Sayısal Entegrasyon İfadelerinin Üretilmesi, Değerlendirilmesi ve Dokümantasyonu" başlıklı yüksek lisans tezi üzerinde çalışmıştır.

Yayınlar

1. AKINCI, Ö., & Pehlivan, H. (2024). Polinomlara dayalı yaklaşımlarıyla sayısal entegrasyon ifadelerinin üretilmesi, değerlendirilmesi ve dokümantasyonu. In Proceedings of the 7th International Conference on Engineering Sciences (pp. 151-160). Ankara, Turkey.