GAZLARIN ÖZGÜL ISI KAPASİTESİ

MAT353 NÜMERİK ANALİZ DERSİ PROJE RAPORU

Taha Yücel – 23120205067

Mehmet Efe Kayacık – 23120205061

Selim Öztürk – 23120205012

https://github.com/tahayucel230/n-merik-analiz-3.1.2-proje--devi

Özet

Bu projenin amacı karbondioksit gazının özgül ısı kapasitesini, mol sayısına bağlı olarak bir polinom fonksiyon ile modellemektir. Bu projede ele alınan temel problem; sera gazı etkisinde, endüstriyel ısıtma-soğutma sistemlerinde, termodinamik çevrim analizlerinde kritik rolü olan karbondioksit gazının sıcaklığa bağlı özgül ısı kapasitesinin değişimini incelemektir. Bu çalışmanın temel amacı, deneysel verilerden hareketle karbondioksit gazının termodinamik davranışını modellemek ve belli bir ısı kapasitesi değeri için gereken sıcaklığı yüksek doğruluk oranıyla tahmin etmektir. Bu amaçla, eldeki deney verilerine En Küçük Kareler Yöntemi kullanılmıştır. Elde edilen polinomsal denklemin, denklemlerin köklerini bulmak ve hedef özgül ısı değerine karşılık gelen sıcaklığı hesaplamak için Newton-Raphson, Bisection, Regula-Falsi, Secant yöntemleri kullanılmıştır.

Giriş

Termodinamik süreçlerin, sistemlerin tasarlanması sürecinde akışkanların özgül ısı kapasitelerinin sıcaklıkla nasıl değiştiğinin bilinmesi büyük bir önem taşımaktadır. Özellikle karbondioksit gazı gibi geniş bir sıcaklık aralığında kullanılan gazlar için bu değişim lineer değildir ve karmaşık, zorlu fonksiyonlarla ifade edilmektedir. Karbondioksit gazının endüstriyel alanlarda ne kadar çok kullanıldığı ve sera gazı olarak etkisi de düşünüldüğü zaman, bu değişimin bilinmesi daha büyük bir anlam ifade etmektedir. Mühendislik uygulamalarında optimizasyon için bu verilerin matematiksel bir modele dönüştürülmesi gerekmektedir. Bu yüzden bu projedeki matematiksel yöntemleri kullanmak fayda sağlayabilir. Literatürde; Ely'nin çalışması olarak geçen deneyde, karbondioksit gazı kontrollü sıcaklık ve basınç koşulları altında kapalı bir deney düzeneğine alınmış, gazın sıcaklığı bilinen bir miktar ısı enerjisi verilerek artırılmıştır. Sisteme verilen ısı miktarı ve sıcaklık değişimi ölçülerek, özgül ısı kapasitesi hesaplanmıştır. Sabit basınç altındaki özgül ısı ise termodinamik bağıntılar ve deneysel veriler yardımıyla elde edilmiştir. Ancak bu yöntemin zayıf yönleri bulunmaktadır. Deneyler düşük-orta basınç aralıklarıyla sınırlıdır. Ölçüm hassasiyeti, dönemin deneysel ekipmanlarıyla sınırlı olduğundan belirsizlik payı daha yüksektir [1]. Literatürde, "Thermal physical applications of carbon dioxide" araştırmasında; karbondioksit gazının sıcaklık ve basınca bağlı özgül ısı verileri, güvenilir deneysel veri tabanlarından ve önceki çalışmalardan derlenmiştir. Bu veriler, durum denklemleri (Peng-Robinson, Span-Wagner) ve ampirik korelasyonlar yardımıyla modellenmiştir. Yüksek basınç noktalarında, özgül ısı kapasitesindeki ani değişimler analiz edilebilmiştir. Bu yöntemin en güçlü yönü, geniş çalışma aralığı olmuştur. Ancak, bazı zayıf yönleri de bulunmaktadır. Sonuçlar büyük ölçüde kullanılan termodinamik modele ve korelasyonların doğruluğuna, önceki çalışmalara bağlıdır. Kritik nokta civarında özgül ısı değerleri çok hızlı değiştiğinden, küçük model hataları büyük hesaplama sapmalarına yol açabilmektedir [2]. Literatürdeki üçüncü bir araştırmada ise, saf karbondioksit ve karbondioksit içeren başka gaz karışımları için özgül ısı değerleri hesaplanmıştır. Hesaplamalarda durum denklemleri ve faz dengesi modelleri kullanılmıştır. Özgül ısı kapasitesi, entalpi ve iç enerji türevleri üzerinden termodinamik olarak türetilmiştir. Bu çalışmanın zayıf yönleri ise, sonuçların doğruluğunun büyük ölçüde kullanılan durum denkleminin uygunluğuna bağlı olmasıdır. Endüstriyel karışımlar çok safsızlık içerdiğinden, model basitleştirmeleri bazı belirsizliklere yol açabilecektir [3]. Bu projenin literatürdeki çalışmalardan ayrılan yönü, çeşitli

verisetlerinden alacağı verileri kullanabilir ve bu verileri nümerik yöntemlerle işledikten sonra istenilen noktaya en yakın verileri hızlı bir şekilde verebilmesidir.

Yöntem

Projede kullanılan temel matematiksel yaklaşımlar; En Küçük Kareler Yöntemi, Newton-Raphson Yöntemi, Secant Yöntemi, Bisection Yöntemi, Regula-Falsi Yöntemidir. Newton-Raphson Yönteminde; hedeflenen y değerini sağlayan x değerini bulmak için $\mathrm{f(x)} = \mathrm{P(x)} - \mathrm{y} = 0$ formülü kullanılmıştır. İterasyonun formülü ise, $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$ olarak kullanılmıştır. Burada f'(x), polinomun analitik türevidir. Diğer formüllerde de yine $f(x) = P(x) - y = 0$ formülü hedef y değerini sağlayan x değerini bulmak için kullanılmıştır. Bu yöntemlerin arasında iterasyon formülleri değişmektedir.

Bisection yöntemi için; $x_r = \frac{a+b}{2}$

Regula-Falsi Yöntemi için; $x_r = \frac{a \cdot f(b) + b \cdot f(a)}{2}$

Secant Yöntemi için; $x_{i+1} = x_i - (f(x_i) \cdot (x_i - x_{i-1}))/(f(x_i) - f(x_{i-1}))$

En Küçük Kareler Yönteminde, verilen x ve y veri noktalarına en yakın P(x) polinomunu bulmak için hata kareleri toplamı minimize edilmiştir. Matris formatında çözüm; $A^T Ac = A^T$y. Bu formülde A katsayılar matrisi, c ise bulunacak polinom katsayılarıdır. Analizler Python programlama dili kullanılarak; matris işlemleri için NumPy, grafikler için Matplotlib kütüphaneleriyle gerçekleştirilmiştir.

Uygulama

Matematiksel işlemler için NumPy, görsel modelleme için Matplotlib kullanılan Notebook sisteminde önce ihtiyaç duyulacak matematiksel fonksiyonları, daha sonrasında fonksiyonlarda kök bulma gibi amaçlarla kullanmak üzere bu matematiksel fonksiyonların da kullanıldığı fonksiyonlar yazılmıştır. Kullanılan veri seti (https://webbook.nist.gov/cgi/cbook.cgi?ID=C124389&Type=JANAFG&Table=on, National Institute of Standarts and Technology) [4], karbondioksit gazının alakalı deneylerin sonucunda belirli miktarlarda (mol) özgül ısı kapasitesini barındırmaktadır. Bu deney sonuçları ve Python algoritma kullanılarak polinom tahminlerine, ve bu polinom tahminlerinin gerçek sonuçlara ne kadar yakın olduğuna hataları hesaplayarak ulaşılmıştır. Grafikler üzerindeki değerlerde T = 0.0001 hata oranı kabul edilebilir varsayılmıştır ve işlem sonuçları buna göre hesaplanmıştır.

```
#gerektiğinde fonksiyon üzerinde x değerini verip y değerine ulaşabilmemiz
için kullanacağımız fonksiyon
def solveY(x, coefficientsi):
    coefficients = coefficientsi.copy()
    y = 0
    factorAmount = len(coefficients)
    countedFactorAmount = 0
    for factor in coefficients:
        xa = pow(x, factorAmount - countedFactorAmount - 1)
        y += factor * xa
        countedFactorAmount += 1
    return y
```

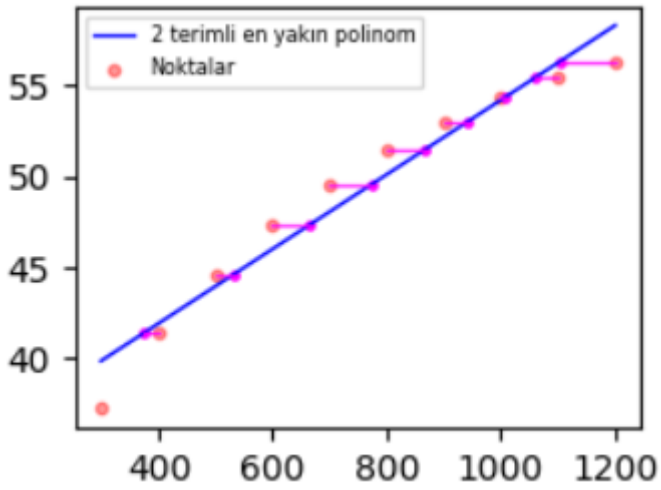
```
def derivePolynomial(coefficientsi):
    coefficients = coefficientsi.copy()
    derivedCoefficients = []
    polynomialLength = len(coefficients)
    countedCoefficients = 0
    while countedCoefficients < (polynomialLength - 1):
        coefficientPower = polynomialLength - countedCoefficients - 1
        derivedCoefficients.append(coefficients[countedCoefficients]*coefficie
ntPower)
        countedCoefficients += 1
    #print(derivedCoefficients)
    return derivedCoefficients

def smallestSquareClosestPolynomialMethod(polynomialLength, pointsi):
    points = pointsi.copy()

    numberOfPoints = len(points)
    #print("len(points)", len(points))
    coefficientsSideMatrix = np.zeros((numberOfPoints, polynomialLength))
    for pointIndex in range(numberOfPoints):
        arrayPiecePerPoint = np.zeros(polynomialLength)
        for factorIndex in range(polynomialLength):
            arrayPiecePerPoint[factorIndex] = pow(points[pointIndex][0],
(polynomialLength - factorIndex - 1))
        coefficientsSideMatrix[pointIndex] = arrayPiecePerPoint

    valuesSideMatrix = np.zeros((numberOfPoints, 1))
    for pointIndex in range(numberOfPoints):
        valuesSideMatrix[pointIndex][0] = points[pointIndex][1]

    fT = np.dot(np.transpose(coefficientsSideMatrix), coefficientsSideMatrix)
    vT = np.dot(np.transpose(coefficientsSideMatrix), valuesSideMatrix)
    coefficientsResult = np.linalg.solve(fT, vT)

    return coefficientsResult
```

```
#bize istenen hata aralığında, polinom çarpanlarını verdiğimiz takdirde
bisection yöntemi ile kabul edilebilir bir kök aralığı verecek fonksiyon
def bisection(coefficientsi, rangeIni, errorMargin):
    coefficients = coefficientsi.copy()
    rangeIn = rangeIni.copy()

    #print("rangeIn: ", rangeIn)

    #verilen aralığın sınırlarında y değerlerinin hesaplanması
    limitValues = [solveY(rangeIn[0], coefficients), solveY(rangeIn[1],
coefficients)]
    #print("limit values: ", limitValues[0], ", ", limitValues[1])

    #eğer sınır değerlerinin ortasında bir tek katlı kök yoksa kodun
durdurulması
    if ((limitValues[0] * limitValues[1] > 0) == True):
        #print("Secilen aralik uygun bir aralik degil.")
        return [None, None]

    #değerler arasındaki fark hata payından büyük olduğu sürece yeni bir
aralık hesaplanması
    while (abs(limitValues[1] - limitValues[0]) <= errorMargin) == 0:

        #bulunan aralığın sınırlarından herhangi biri kökün kendisi mi diye
kontrol edilmesi
        if limitValues[0] == 0:
            #print("\nBir cozum bulundu!: " + str(rangeIn[0]))
            rangeIn[1] = rangeIn[0]
            return rangeIn
        if limitValues[1] == 0:
            #print("\nBir cozum bulundu!: " + str(rangeIn[1]))
            rangeIn[0] = rangeIn[1]
            return rangeIn

        #bisection yöntemine göre yeni limitin seçilmesi
        newLimit = (rangeIn[0] + rangeIn[1])/2

        #yeni limitin aralıkta gerekli yerine yerleştirilmesi
        newLimitY = solveY(newLimit, coefficients)
        if (newLimitY * limitValues[0] > 0):
            rangeIn[0] = newLimit
            limitValues[0] = newLimitY
        else:
            rangeIn[1] = newLimit
            limitValues[1] = newLimitY

    return rangeIn
#bize istenen hata aralığında, polinom çarpanlarını verdiğimiz takdirde
regula-falsi yöntemi ile kabul edilebilir bir kök aralığı verecek fonksiyon
```

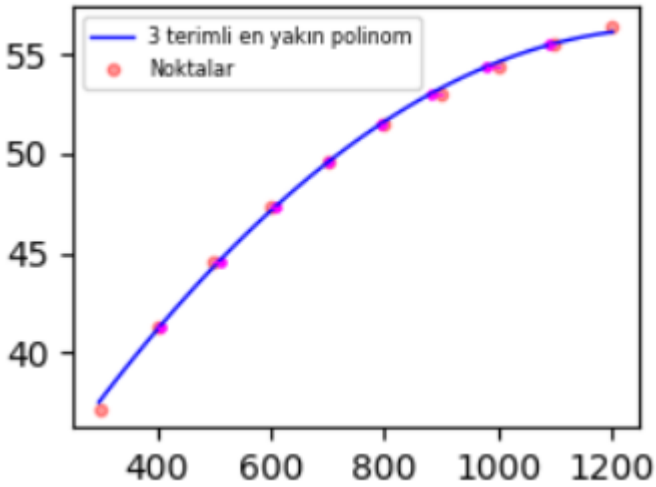
```
def regulaFalsi(coefficientsi, rangeIni, errorMargin):
    coefficients = coefficientsi.copy()
    rangeIn = rangeIni.copy()

    #verilen aralığın sınırlarında y değerlerinin hesaplanması
    limitValues = [solveY(rangeIn[0], coefficients), solveY(rangeIn[1],
coefficients)]

    #eğer sınır değerlerinin ortasında bir tek katlı kök yoksa kodun
durdurulması
    if ((limitValues[0] * limitValues[1] > 0) == True):
        #print("Secilen aralik uygun bir aralik degil.")
        return [None, None]

    #değerler arasındaki fark hata payından büyük olduğu sürece yeni bir
aralık hesaplanması
    while (abs(limitValues[1] - limitValues[0]) <= errorMargin) == 0:

        #bulunan aralığın sınırlarından herhangi biri kökün kendisi mi diye
kontrol edilmesi
        if limitValues[0] == 0:
            #print("\nBir cozum bulundu!: " + str(rangeIn[0]))
            rangeIn[1] = rangeIn[0]
            return rangeIn
        if limitValues[1] == 0:
            #print("\nBir cozum bulundu!: " + str(rangeIn[1]))
            rangeIn[0] = rangeIn[1]
            return rangeIn

        #regulaFalsi yöntemine göre yeni limitin seçilmesi
        newLimit = (rangeIn[0]*solveY(rangeIn[1], coefficients) -
rangeIn[1]*solveY(rangeIn[0], coefficients))/(solveY(rangeIn[1], coefficients)
- solveY(rangeIn[0], coefficients))
        #print(newLimit)

        #yeni limitin aralıkta gerekli yerine yerleştirilmesi
        newLimitY = solveY(newLimit, coefficients)
        if (newLimitY * limitValues[0] > 0):
            rangeIn[1] = newLimit
            limitValues[1] = newLimitY
        else:
            rangeIn[0] = newLimit
            limitValues[0] = newLimitY

    return rangeIn
```

```
def newtonRaphson(coefficientsi, initialGuess, errorMargin):
    coefficients = coefficientsi.copy()

    #polinomun türevinin alınması
    fd = derivePolynomial(coefficients)

    #hata payı kabul edilebilir olana kadar yeni iterasyon denenmesi
    newGuess = initialGuess - (solveY(initialGuess,
coefficients)/solveY(initialGuess, fd))
    #print(f"{initialGuess} -> {newGuess}")
    while (abs(newGuess - initialGuess) > errorMargin):
        initialGuess = newGuess
        newGuess = initialGuess - (solveY(initialGuess,
coefficients)/solveY(initialGuess, fd))
        #print(f"{initialGuess} -> {newGuess}")

    return newGuess
```

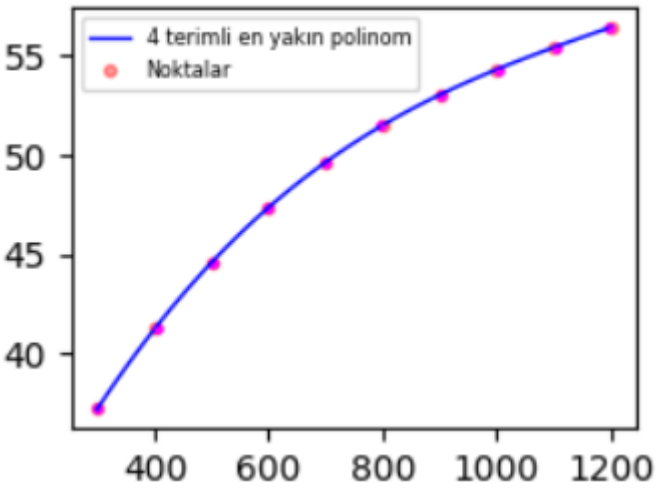
```
#secant yöntemi ile denklem için kök bulma
def secant(coefficientsi, oldGuess, initialGuess, errorMargin):
    coefficients = coefficientsi.copy()

    #hata payı kabul edilebilir olana kadar yeni iterasyon denenmesi
    newGuess = initialGuess - ( (solveY(initialGuess,
coefficients)*(initialGuess - oldGuess)) / (solveY(initialGuess, coefficients)
- solveY(oldGuess, coefficients)) )
    #print(f"{initialGuess} -> {newGuess}")
    while (abs(newGuess - initialGuess) > errorMargin):
        oldGuess = initialGuess
        initialGuess = newGuess
        newGuess = initialGuess - ( (solveY(initialGuess,
coefficients)*(initialGuess - oldGuess)) / (solveY(initialGuess, coefficients)
- solveY(oldGuess, coefficients)) )
        #print(f"{initialGuess} -> {newGuess}")

    return newGuess
```



1. dereceden en yakın polinom (doğrusal denklem) şekilde görüldüğü gibi bulundu ve “Mean Squared Error” (MSA) skoru 3154.9260 olarak hesaplandı.



2. dereceden en yakın polinom şekilde görüldüğü gibi bulundu ve MSA skoru 134.7317 olarak hesaplandı.



3. dereceden en yakın polinom şekilde görüldüğü gibi bulundu ve MSA skoru 4.3392 olarak hesaplandı.

Tartışma

Sonuçlar incelendiğinde, 3. dereceden polinomun karbondioksit verileri üzerinde 4.3392 MSA skoru ile en yüksek benzerliği sağladığı görülmüştür.

Test Süreçleri

Bazı bilinen sıcaklık değerleri için polinomların ürettiği değerler kontrol edilerek polinom test edilmiştir. Kök bulma algoritmaları bilinen denklemler ile doğrulukları kontrol edildikten sonra kullanılmıştır. Polinom bulurken kullanılan algoritmanın doğruluğu bazı n sayıda nokta için n terimli polinomlar üretmesini sağlayarak ve aynı polinomların sağlaması yapılarak test edilmiştir. Algoritmalarda 0'a bölme, istenen aralıkta kök bulamama gibi problemlerle karşılaşılmış, bulunan problemler ilgili fonksiyonlar içerisinde çözülmüştür.

Sonuç ve Öneriler

Kullanılan algoritma, doğruluğunu ve güvenilirliğini göstermekle beraber, hazır mühendislik tablolarından daha pratik oluşu ile çeşitli teorik hesaplamalarda hız ve verimlilik sağlayacağını belli etmiştir. Daha verimli ve isabetli sonuçlar için algoritma düzenlenebilir, farklı matematiksel yöntemler denenebilir, parçalı fonksiyonlar kullanmak gibi yöntemlere başvurulabilir. Çalışma bu yöntemlerin mümkün ve verimli olacağına işaret etmektedir.

Kaynakça

[1] J. F. Ely, "Heat Capacity of Gaseous Carbon Dioxide,"
Journal of Research of the National Bureau of Standards,
vol. 48, no. 3, pp. 179–188, 1952.

[2] Y. Zhang, H. Wang, and Q. Li, "Thermal physical applications of carbon dioxide,"
Case Studies in Thermal Engineering,
vol. 47, Art. no. 103032, 2023.

[3] H. Li, "Thermodynamic Properties of CO_2 Mixtures and Their Applications in Engineering,"
Ph.D. dissertation, KTH Royal Institute of Technology,
Stockholm, Sweden, 2008.

[4] NIST Chemistry WebBook, SRD 69, "JANAF Thermochemical Tables (C124389),"
National Institute of Standards and Technology. [Online].
Available: https://webbook.nist.gov/cgi/cbook.cgi?ID=C124389&Type=JANAFG&Table=on.