Muhammet Kaan ALSANCAK

245113009|

Mühendisler İçin Uygulamalı Sayısal Yöntemler

Vize RAporu

Giriş

Ödev içeriği AraSınav\_Uygulamalı Sayısal Yöntemler 20241120.pdf dosyasında istenilen konulara göre hazırlanmıştır. İlk 3 soru daha önceki ödev kısmında verildiği için bu raporda yer almamaktadır.

Açıklama

Rayleigh Ritz metodunda yazılı bir çıktı istenilmediği çıkarımıyla raporda bu metodun elle çözümü ile ilgili bir ek bulunmamaktadır.

Adams Fourth Order Predictor-Corrector method

Bu kısımda Adams Fourth Order Predictor-Corrector metodunu kodları açıklanacaktır.

Aşağıdaki satırlarda temiz bir başlangıç yapabilmek için matlabta oluşturulmuş değişkenler ve konsol ekranı temizlenmektedir.

% Adams Dördüncü Derece Predictor-Corrector Yöntemi

clc;

clear;

Bu kısımda ödevde verilen  diferansiyel denklem tanımlanmaktaır.

% Diferansiyel denklem: y' = y - t^2 + 1

f = @(t, y) y - t^2 + 1;

Başlangıç zamanı ve ilk  anındaki değeri tanımlanmıştır.

% Başlangıç koşulları

t0 = 0;

y0 = 0.5;

Burada adım aralıkları ve iterasyon yapılacak maksimum nokta tanımlanır.

% Zaman aralığı ve adım boyutu

h = 0.2; % Adım boyutu

t\_end = 2; % Aralığın sonu

Toplam adım sayısı hesaplanır tam sayı sonuç eldeetmek için ceil komutu kullanılır. Bu komutla eğer sonuç virgüllü sayı ise yukarı yuvarlama işlemi yapılır.

n = ceil((t\_end - t0) / h); % Adım sayısı

Bu kısımda for döngüsünde kullanılacak zaman ve sonuç vektörleri oluşturulur. sonuç dizisinin ilk elemanı  değerini alır.

% Verimlilik için dizi önceden ayrılıyor

t = t0:h:t\_end;

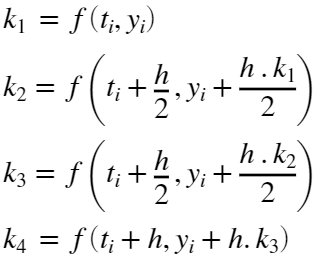
y = zeros(size(t));

% Başlangıç değeri

y(1) = y0;

İlk üç adım için runge kutta 4th order metodu uygulanır.

Runge kutta 4th order metodunun formulü aşağıdaki gibidir.



Yeni y değerinin hesaplanması



bu formuller ile üç y değeri aşağıda hesaplanmıştır.

% İlk üç noktayı Runge-Kutta 4. derece ile hesapla

for i = 1:3

k1 = f(t(i), y(i));

k2 = f(t(i) + h/2, y(i) + h\*k1/2);

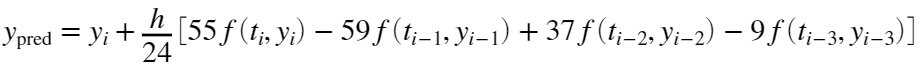
k3 = f(t(i) + h/2, y(i) + h\*k2/2);

k4 = f(t(i) + h, y(i) + h\*k3);

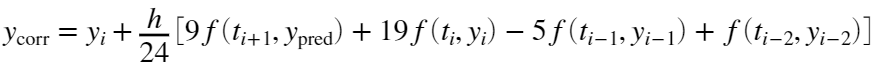
y(i+1) = y(i) + h\*(k1 + 2\*k2 + 2\*k3 + k4)/6;

end

Predictor adımı, y'yi tahmin etmek için geçmişteki f(t,y) türev değerlerini kullanır



Corrector adımı, tahmin edilen �'i kullanarak 'yi düzeltir:





% Adams Dördüncü Derece Predictor-Corrector uygulaması

for i = 4:n

% Predictor adımı (Adams-Bashforth 4. derece)

y\_pred = y(i) + h/24 \* (55\*f(t(i), y(i)) - 59\*f(t(i-1), y(i-1)) + ...

37\*f(t(i-2), y(i-2)) - 9\*f(t(i-3), y(i-3)));

% Corrector adımı (Adams-Moulton 4. derece)

y\_corr = y(i) + h/24 \* (9\*f(t(i+1), y\_pred) + 19\*f(t(i), y(i)) - ...

5\*f(t(i-1), y(i-1)) + f(t(i-2), y(i-2)));

% Düzeltilmiş değeri güncelle

y(i+1) = y\_corr;

end

Sonuçları ekrana bastır.

% Sonuçları göster

results = table(t', y', 'VariableNames', {'Zaman', 'Çözüm'})

|  | **Zaman** | **Çözüm** |
| --- | --- | --- |
| **1** | 0 | 0.5000 |
| **2** | 0.2000 | 0.8293 |
| **3** | 0.4000 | 1.2141 |
| **4** | 0.6000 | 1.6489 |
| **5** | 0.8000 | 2.1272 |
| **6** | 1 | 2.6408 |
| **7** | 1.2000 | 3.1799 |
| **8** | 1.4000 | 3.7324 |
| **9** | 1.6000 | 4.2834 |
| **10** | 1.8000 | 4.8151 |
| **11** | 2 | 5.3054 |

disp(results)

% Çözümün grafiği

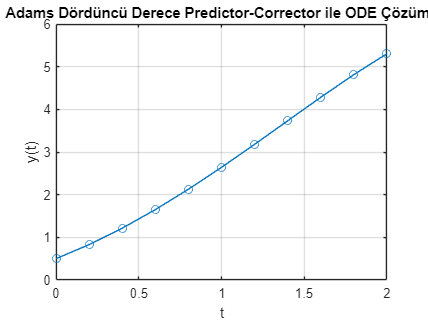
plot(t, y, '-o');

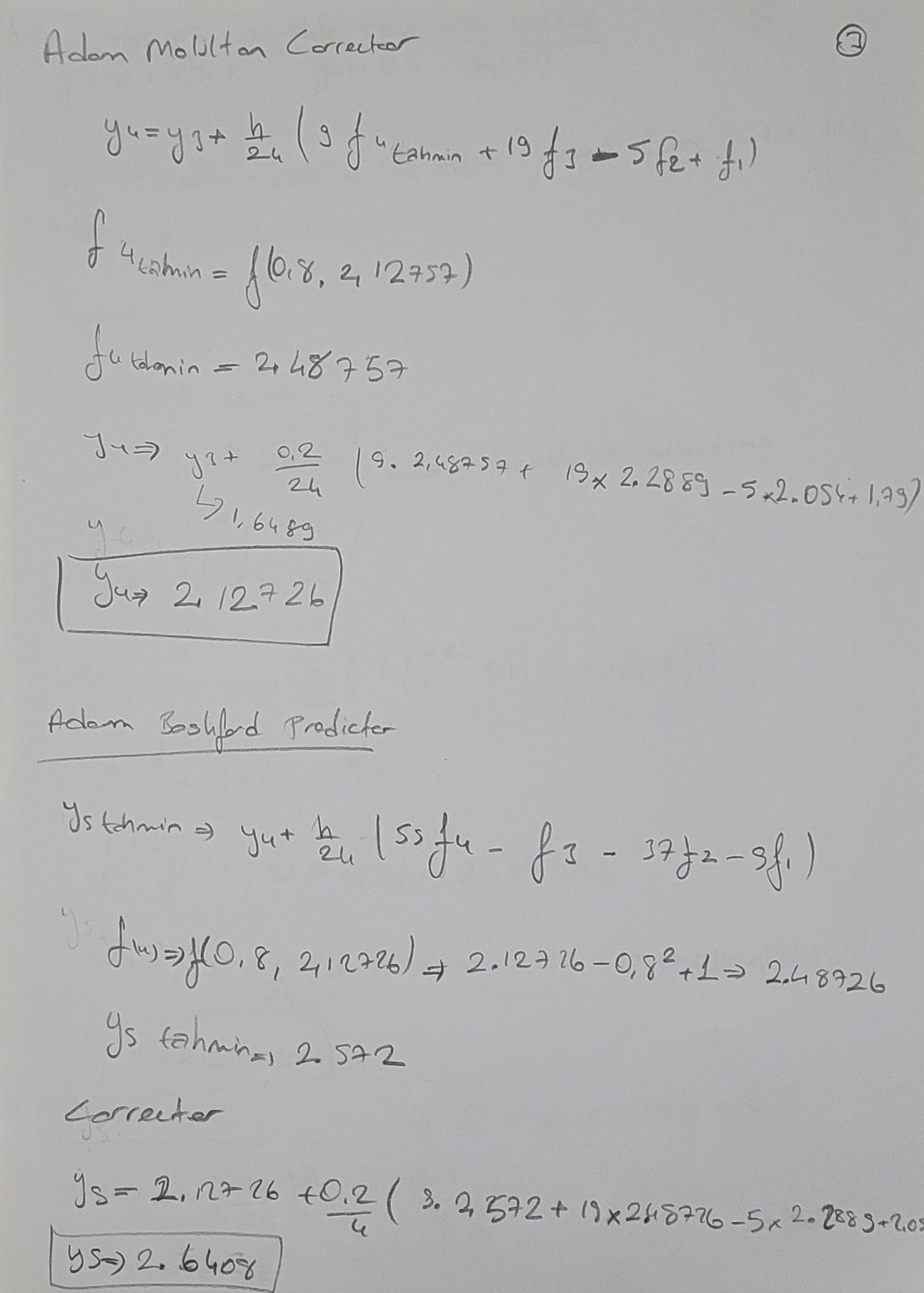
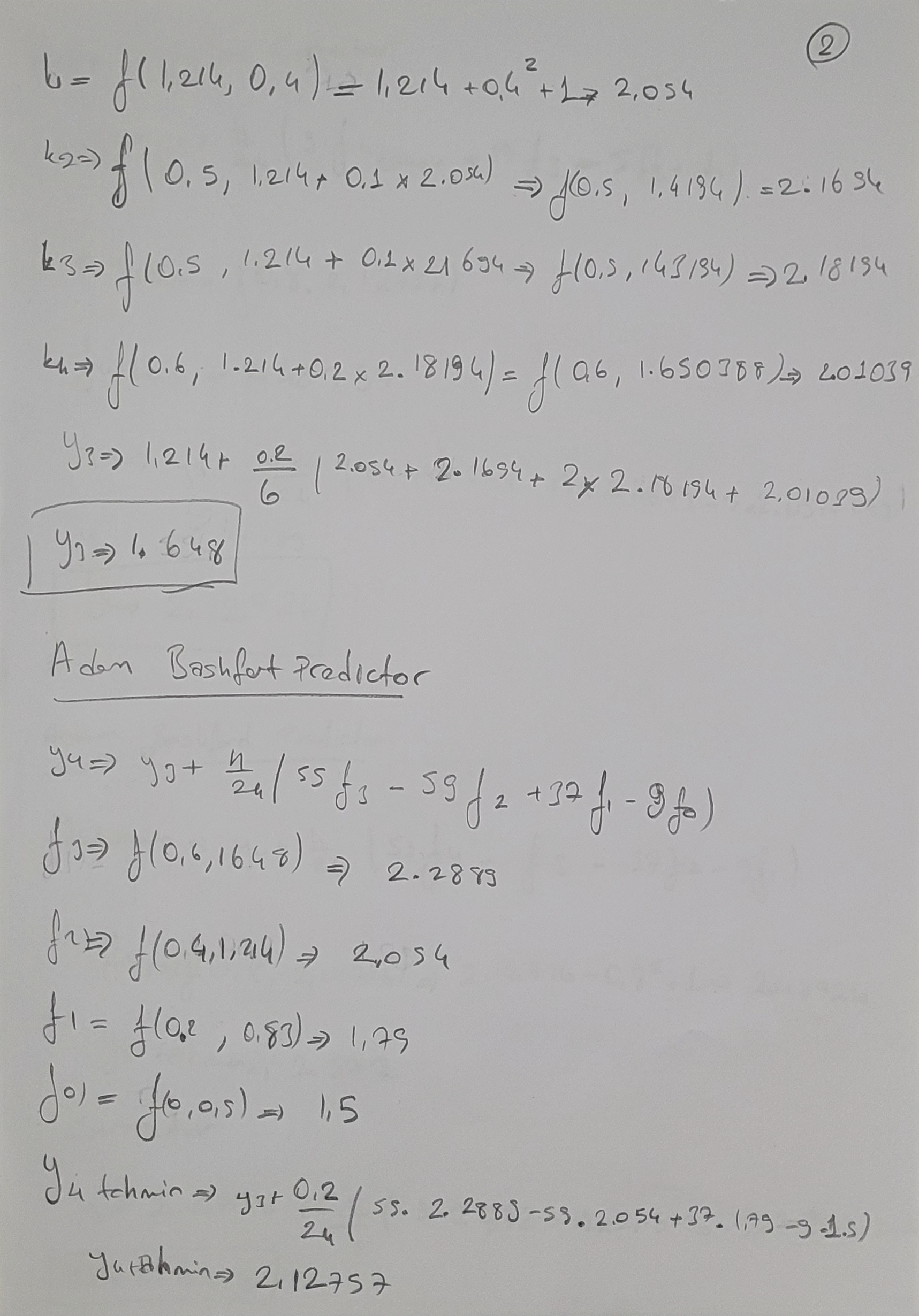
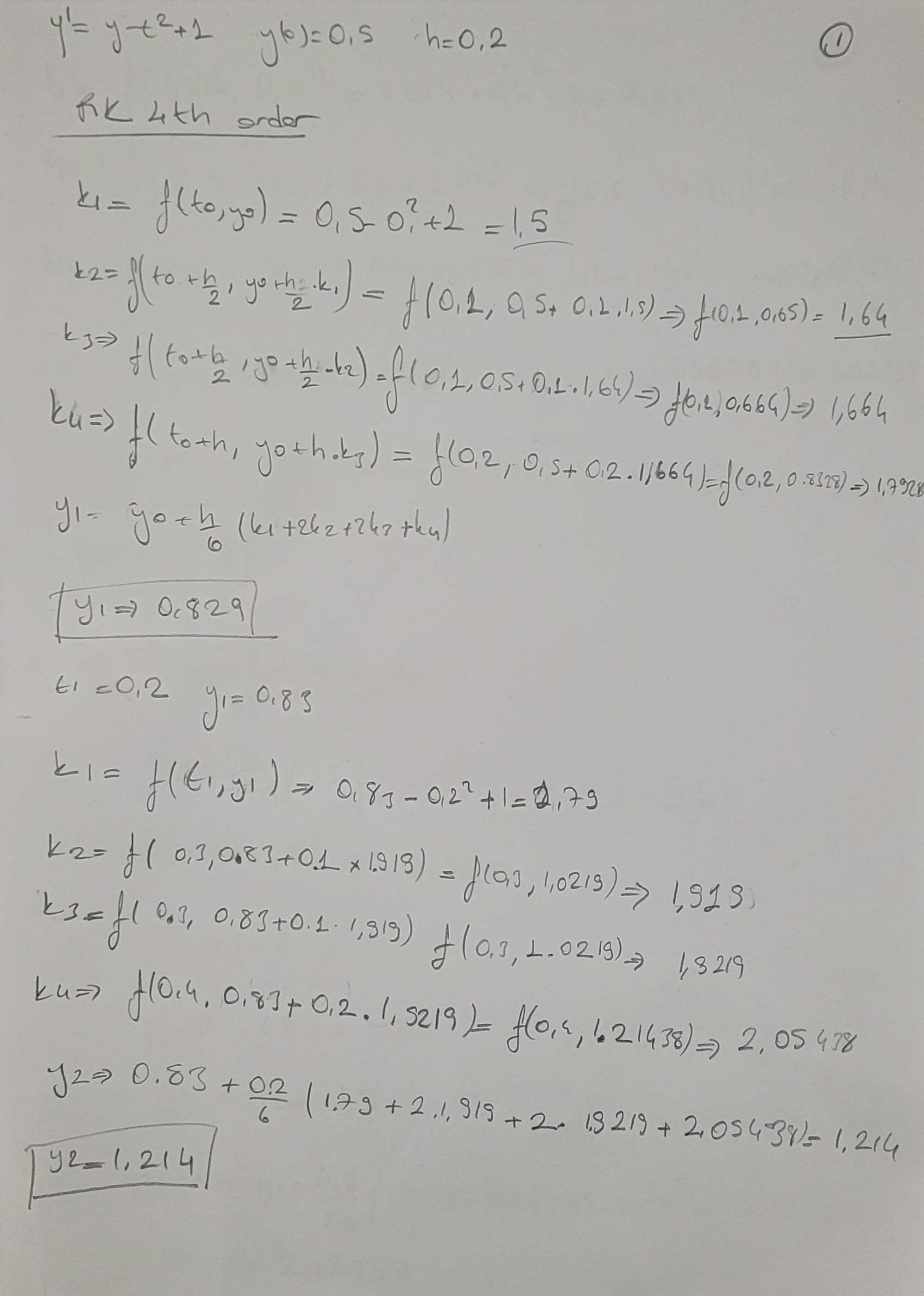
title('Adams Dördüncü Derece Predictor-Corrector ile ODE Çözümü');

xlabel('t');

ylabel('y(t)');

grid on;





Adams Variable Step Size Predictor Corrector

**Kodun Amacı:**Bu kod,  diferansiyel denklemini çözmek için Adams Variable Step Size Predictor Correcto metodunu kullanır.

Aşağıdaki satırlarda temiz bir başlangıç yapabilmek için matlab variablelar ve consol ekranı temizlenmektedir.

%

clc;

clear;

Bu kısımda ödevde verilen  diferansiyel denklem tanımlanmaktaır.

% Diferansiyel denklem: y' = y - t^2 + 1

f = @(t, y) y - t^2 + 1;

y\_exact = @(t) (t + 1).^2 - 0.5 \* exp(t); % Analitik çözüm

Başlangıç noktası bitiş noktası ve a,b, , TOL ve minimum maksimum adım aralıkları belirlenir.

% Giriş değerleri

a = 0; % Başlangıç noktası

b = 2; % Bitiş noktası

alpha = 0.5; % Başlangıç koşulu y(a) = alpha

TOL = 10e-5; % Tolerans

hmax = 0.2 ; % Maksimum adım boyutu

hmin = 0.001; % Minimum adım boyutu

Dongüde kullanılacak değişken atamaları yapılır.

% Değişkenleri başlat

t0 = a; % Başlangıç zamanı

w0 = alpha; % Başlangıç çözümü

h = hmax; % Başlangıçta maksimum adım boyutu

FLAG = 1; % Döngüyü kontrol etmek için

LAST = 0; % Son adımı belirlemek için

i = 1; % Adım sayacı

sigma =0.0; % Sigma

Grafik çıktısı için verilerin tutulacağı arrayler oluşturuldu.

% Çözüm değerlerini saklamak için diziler

T\_all = []; % Tüm zaman değerleri

W\_all = []; % Tüm çözüm değerleri

errors = []; % Gerçek hata |y\_exact - w|

Konsol çıktısının başlık yazılılarıyazdırılır.

% Başlangıç değerlerini çıktı olarak yazdır

fprintf('Adım\t t\t w\t h sigma\n');

Adım t w h sigma

fprintf('%4d\t%8.4f\t%8.6f\t%8.4f\t%8.4f\n', i, t0, w0, h,sigma);

1 0.0000 0.500000 0.2000 0.0000

Vektörler oluşturulur.

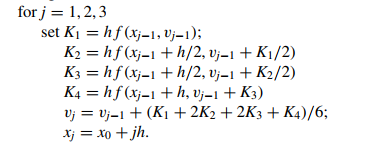
% İlk üç adım için Runge-Kutta 4. derece ile hesapla

T = zeros(1, 4); % Zaman değerlerini sakla

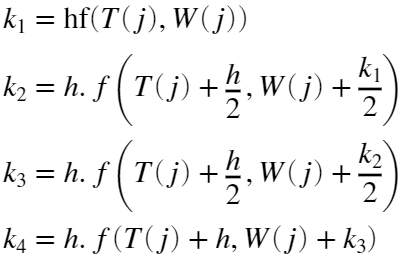
W = zeros(1, 4); % Çözüm değerlerini sakla

T(1) = t0;

W(1) = w0;



Bu adımlarda runge kutta 4. derece çözümü yapılır.



Yeni değer çözümü için aşağıdak formulü kullanır.



for j = 1:3

% Runge-Kutta katsayılarını hesapla

k1 = h \* f(T(j), W(j));

k2 = h \* f(T(j) + h / 2, W(j) + k1 / 2);

k3 = h \* f(T(j) + h / 2, W(j) + k2 / 2);

k4 = h \* f(T(j) + h, W(j) + k3);

% Bir sonraki çözüm değerini bul

W(j + 1) = W(j) + (k1 + 2 \* k2 + 2 \* k3 + k4) / 6;

T(j + 1) = T(j) + h;

% Adım sayacını artır ve sonucu yazdır

i = i + 1;

fprintf('%4d\t%8.4f\t%8.6f\t%8.4f\t%8.4f\n', i, T(j + 1), W(j + 1), h, sigma);

end

2 0.2000 0.829293 0.2000 0.0000

3 0.4000 1.214076 0.2000 0.0000

4 0.6000 1.648922 0.2000 0.0000

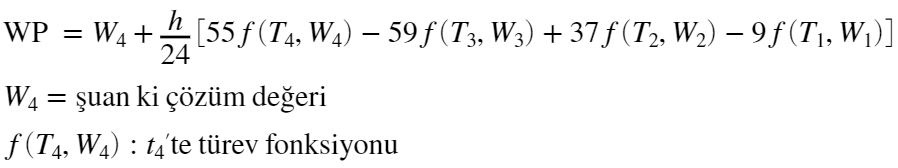
Sonuçların grafik çıktısı için sonuçları bir vektörde toplandı.

% Tüm sonuçları kaydet

T\_all = [T\_all, T];

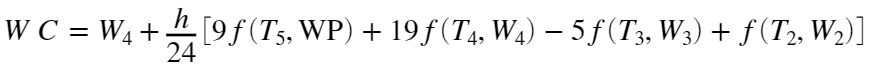
W\_all = [W\_all, W];

### Predictor Adımı



Bu adım bir tahminde bulunur.

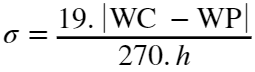
### Corrector adımı



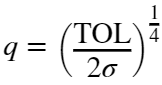
 Predictor adımında hesaplanan tahmin.

Corrector adımında doğrulanmış değer

### Hata Tahmini ()



### Faktör Hesabı

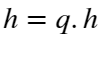


### Yeni Adım Boyutu

Eğer 



değilse

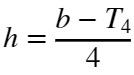


**Adım Boyutunun Sınırlandırılması**

**Maksimum Sınır**



eğer 



**Minimum Sınır**

Eğer h<hmin

işlemi durdur hata mesajı yazdır.

% Predictor-Corrector döngüsünü başlat

while FLAG

% Predictor adımı (Adams-Bashforth 4. derece)

WP = W(4) + h / 24 \* (55 \* f(T(4), W(4)) - 59 \* f(T(3), W(3)) + ...

37 \* f(T(2), W(2)) - 9 \* f(T(1), W(1)));

% Corrector adımı (Adams-Moulton 4. derece)

WC = W(4) + h / 24 \* (9 \* f(T(4) + h, WP) + 19 \* f(T(4), W(4)) - ...

5 \* f(T(3), W(3)) + f(T(2), W(2)));

% Hata tahminini hesapla

% sigma = 19 \* abs(WC - WP) / (270 \* h);

errors = [errors, abs(y\_exact(t\_next) - WC)];

if sigma <= TOL

% Sonucu kabul et

t\_next = T(4) + h;

w\_next = WC;

% Sonucu yazdır

fprintf('%4d\t%8.4f\t%8.6f\t%8.4f\t%8.4f\n', i + 1, t\_next, w\_next, h, sigma);

% Değerleri güncelle

T = [T(2:4), t\_next];

W = [W(2:4), w\_next];

i = i + 1;

% Sonuçları sakla

T\_all = [T\_all, t\_next];

W\_all = [W\_all, w\_next];

% Son adım kontrolü

if LAST

FLAG = 0;

else

% Adım boyutunu gerekirse ayarla

if sigma <= 0.1 \* TOL || T(4) + h > b

q = (TOL / (2 \* sigma))^(1 / 4);

h = min(4 \* h, min(q \* h, hmax))

if T(4) + 4 \* h > b

h = (b - T(4)) / 4

LAST = 1;

end

end

end

else

% Sonucu reddet ve adım boyutunu küçült

q = (TOL / (2 \* sigma))^(1 / 4);

h = max(0.1 \* h, q \* h)

if h < hmin

% Minimum adım boyutu aşıldı, hesaplama başarısız

FLAG = 0;

fprintf('hmin sınırı aşıldı. Çözüm başarısız.\n');

break;

else

% Daha küçük adım boyutuyla tekrar hesapla

T = T(1:3);

W = W(1:3);

i = i - 3;

end

end

end

İşlem başarılı tamamlanma mesajını yazdır. Grafik çıktısı al

% Hesaplama tamamlandı

disp('Hesaplama tamamlandı.');

% Sonuçların Grafiği

figure;

plot(T\_all, W\_all, '-o', 'LineWidth', 1.5);

title('Adams-Bashforth & Adams-Moulton Çözümü');

xlabel('t (Zaman)');

ylabel('w (Çözüm)');

grid on;

legend('Yaklaşık Çözüm', 'Location', 'Best');

Adım t w h sigma

1 0.0000 0.500000 0.2000 0.0000

2 0.2000 0.829293 0.2000 0.0000

3 0.4000 1.214076 0.2000 0.0000

4 0.6000 1.648922 0.2000 0.0000

5 0.8000 2.127206 0.2000 0.0000

6 1.0000 2.640829 0.2000 0.0000

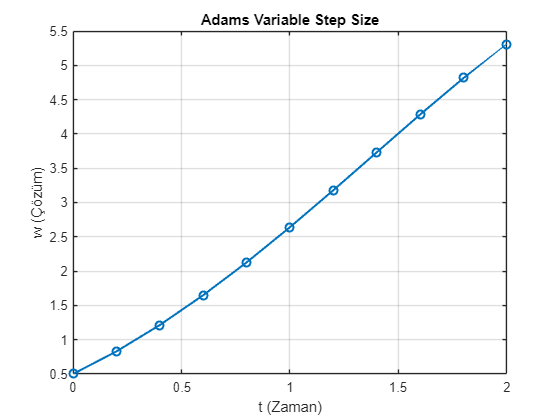
7 1.2000 3.179903 0.2000 0.0000

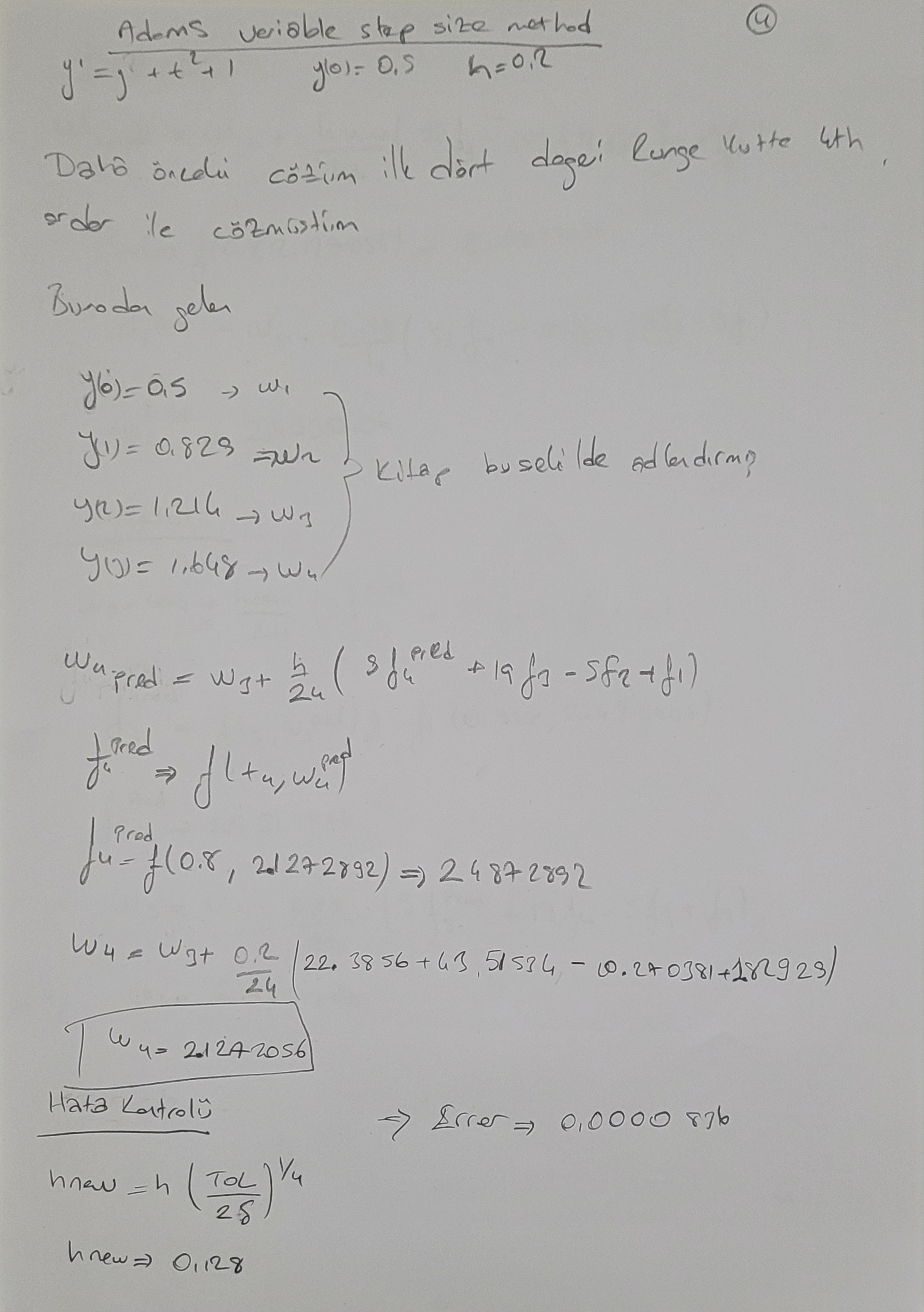
8 1.4000 3.732350 0.2000 0.0001

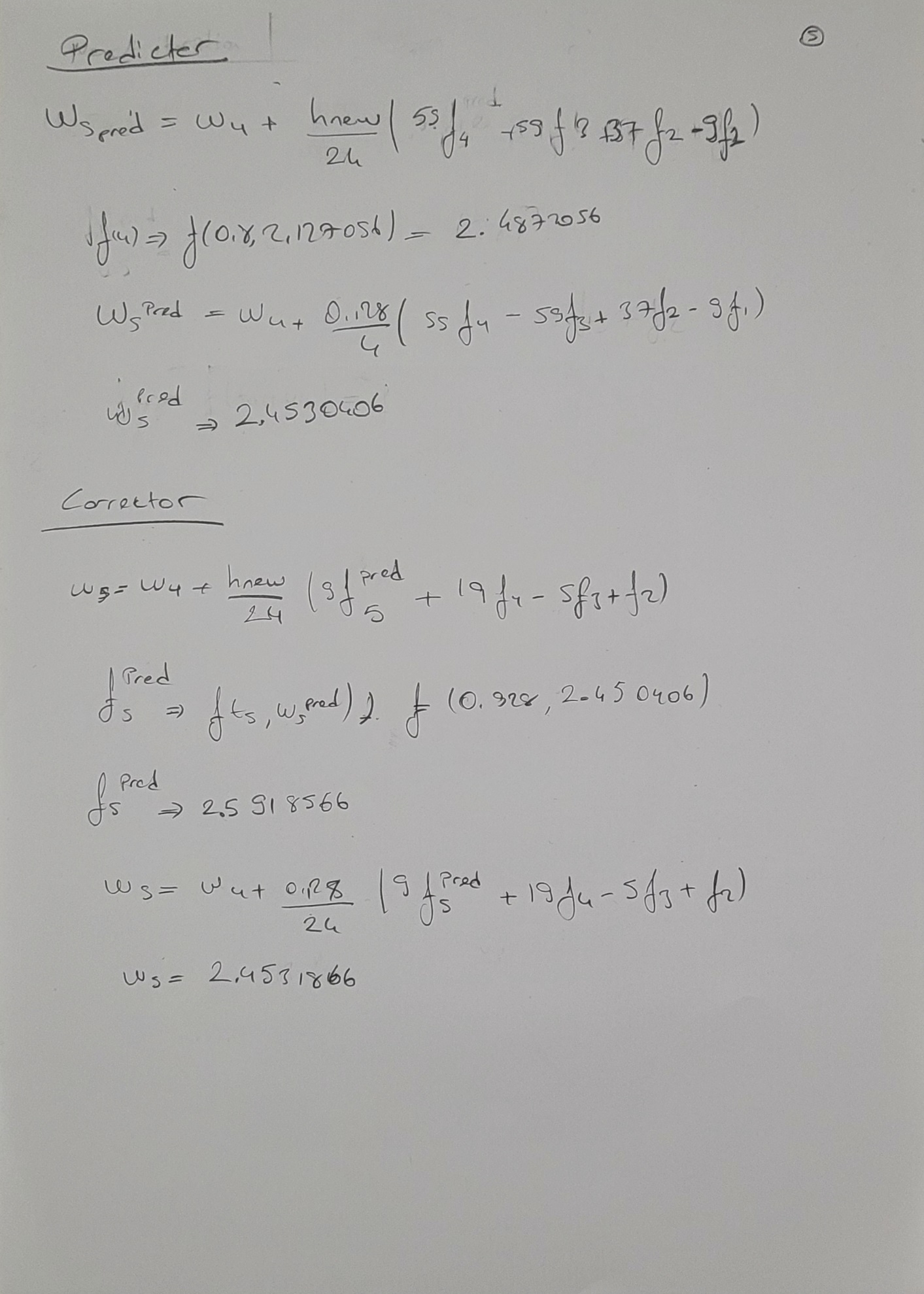
9 1.6000 4.283421 0.2000 0.0001

10 1.8000 4.815096 0.2000 0.0001

11 2.0000 5.305371 0.2000 0.0001







Raleigh Ritz Metodu

Bu kod, bir radyal simetrik malzemede elastik deformasyonu ve gerilmeleri modellemek için enerji yöntemini kullanır. Kodun her bir adımı, sistemdeki yer değiştirme ve gerilme fonksiyonlarının optimize edilmesini ve sonuçların hesaplanmasını içerir.

Matlab değişkenlerini ve konsol temizleme işlemi

clc;

clear;

% Malzeme özellikleri

ElasticModulus = 207e9; % Elastisite modülü (Pa)

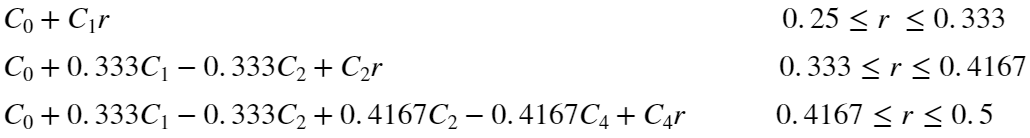
PoissonRatio = 0.3; % Poisson oranı

Parçalı fonksiyonlar için sembolik değişkenlerin oluşturulması.

% Parçalı fonksiyonun tanımı için sembolik değişkenler

syms Radius Coef0 Coef1 Coef2 Coef3 Coef4

Parçalı bir yer değiştirme fonksiyonu u(r)u(r)u(r), radyal bölgeler için şu şekilde tanımlanmıştır:

DisplacementFunction = piecewise(...

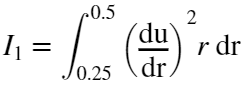
0.25 <= Radius & Radius <= 0.33, Coef0 + Coef1\*Radius, ...

0.33 < Radius & Radius <= 0.4167, Coef0 + 0.33\*Coef1 - 0.33\*Coef2 + Coef2\*Radius, ...

0.4167 < Radius & Radius <= 0.5, Coef0 + 0.33\*Coef1 - 0.33\*Coef2 + 0.4167\*Coef2 - 0.4167\*Coef4 + Coef4\*Radius);

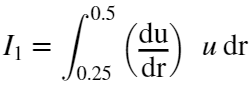
Malzemenin elastik enerjisi üç ana bileşene ayrılmıştır:

## Elastik deformasyon enerjisi:



Integral1 = int((DisplacementDerivative)^2 \* Radius, Radius, 0.25, 0.5);

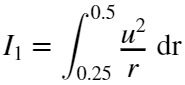
## Poisson etkisi enerjisi:



Integral2 = int(DisplacementDerivative \* DisplacementFunction, Radius, 0.25, 0.5);

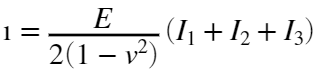
Integral2 = 2 \* PoissonRatio \* Integral2; % Poisson oranı ile çarpılır.

## Radyal gerilme enerjisi:



Integral3 = int(DisplacementFunction^2 / Radius, Radius, 0.25, 0.5);

Bu enerji bileşenleri, elastisite modülü  ve Poisson oranı  ile ağırlıklandırılarak toplam enerji fonksiyonunu oluşturur:



% Elastisite modülü ve Poisson oranı kullanılarak enerji bileşenleri hesaplanır.

Integral1 = ElasticModulus / (2 \* (1 - PoissonRatio^2)) \* Integral1;

Integral2 = ElasticModulus / (2 \* (1 - PoissonRatio^2)) \* Integral2;

Integral3 = ElasticModulus / (2 \* (1 - PoissonRatio^2)) \* Integral3;

TotalEnergy = Integral1 + Integral2 + Integral3;

## Kenar Koşulları ve Toplam Enerji

Kenar koşulları enerjiyi etkiler:



BoundaryCondition = 0.25 \* (200e6) \* (Coef0 + Coef1 \* 0.25);

TotalEnergy = TotalEnergy - BoundaryCondition;

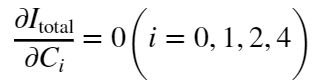
Toplam enerji fonksiyonu şu şekilde güncellenir:



TotalEnergy = 2 \* pi \* 0.25 \* TotalEnergy;

## Enerji Minimizasyonu

Enerji fonksiyonu sabitler açısından minimize edilir:



PartialDerivatives = [diff(TotalEnergy, Coef0); diff(TotalEnergy, Coef1); ...

diff(TotalEnergy, Coef2); diff(TotalEnergy, Coef4)];

% (6) Denklem sistemi oluşturuluyor

% Enerji minimizasyonundan elde edilen denklemler çözülür.

Equations = PartialDerivatives == 0;

% (7) Sabitlerin çözümü

% Sabitlerin analitik çözümü bulunur.

Solutions = solve(Equations, [Coef0, Coef1, Coef2, Coef4]);

% Çözümleri sayısal değerlere dönüştürme

Constants = structfun(@vpa, Solutions, 'UniformOutput', false);

## Optimize Edilmiş Fonksiyonlar

Yer değiştirme fonksiyonu ve türevleri sabitler ile güncellenir:

**Yer değiştirme fonksiyonu:**



OptimizedDisplacement = piecewise(...

0.25 <= Radius & Radius <= 0.33, Constants.Coef0 + Constants.Coef1\*Radius, ...

0.33 < Radius & Radius <= 0.4167, Constants.Coef0 + 0.33\*Constants.Coef1 - 0.33\*Constants.Coef2 + Constants.Coef2\*Radius, ...

0.4167 < Radius & Radius <= 0.5, Constants.Coef0 + 0.33\*Constants.Coef1 - 0.33\*Constants.Coef2 + 0.4167\*Constants.Coef2 - 0.4167\*Constants.Coef4 + Constants.Coef4\*Radius);

**Türev:**

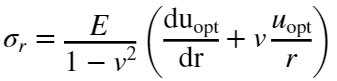


OptimizedDisplacementDerivative = diff(OptimizedDisplacement, Radius);

## Gerilme Hesaplamaları

Optimize edilmiş türevler kullanılarak radyal ve çevresel gerilmeler hesaplanır:

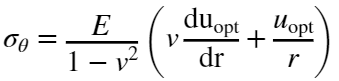
**Radyal gerilme**



RadiusValues = [0.250001, 0.375, 0.4999999];

RadialStressValues = double(subs(RadialStress, Radius, RadiusValues));

**Çevresel gerilme**



CircumferentialStress = (ElasticModulus / (1 - PoissonRatio^2)) \* (OptimizedDisplacementDerivative \* PoissonRatio + OptimizedDisplacement / Radius);

**Displacement Hesaplaması**

DisplacementValues = double(subs(OptimizedDisplacement, Radius, RadiusValues));

**Dataların Çıktı Alınması**

disp('Constants :');

disp(Constants);

disp('Radial Stress (Pa):');

disp(RadialStressValues);

disp('Circumferential Stress (Pa):');

disp(CircumferentialStressValues);

disp('Displacement Values (m):');

disp(DisplacementValues);

