Num. Ana. - Übung 2

Felix Dreßler (k12105003)
Elisabeth Köberle (k12110408)
Ricardo Holzapfel (k11942080)

22. Dezember 2022

1 Testfunktion 1

$$\int_{-1.25}^{1} \sqrt{|x|}\, dx$$

Zuerst berechnen wir mithilfe der Matlab-Funktion „integral“ das numerische Ergebnis des Integrals.

```
>> fun = @(x) sqrt(abs(x))

fun =

function_handle with value:

@(x)sqrt(abs(x))

>> integral(fun, -1.25,1)

ans =

1.598362906722469
```

Nun testen wir unsere Funktion mit Standardwerten.

```
>> [I, exitflag] = E_Trapez(@testfun1, -1.25, 1)

I =

1.598399277022473


exitflag =

1
```

Mit $m_{max} = 6$ konvergiert das Verfahren noch nicht zufriedenstellen, dazu benötigen wir $m_{max} = 24$:

```
>> [I, exitflag] = E_Trapez(@testfun1, -1.25, 1, 24)

I =

1.598361657434845


exitflag =

0
```

Die Lösung unterscheidet sich jedoch noch in der sechsten Nachkommastelle von der Matlab Lösung.

2 Testfunktion 2

$$\int_{0}^{\pi} sin(x)\, dx$$

Zuerst berechnen wir mithilfe der Matlab-Funktion „integral“ das numerische Ergebnis des Integrals.

```
>> fun = @(x) sin(x)

fun =

function_handle with value:

@(x)sin(x)

>> integral(fun, 0,pi)

ans =

2.000000000000000
```

Nun testen wir unsere Funktion mit Standardwerten.

```
>> [I, exitflag] = E_Trapez(@testfun2, 0, pi)

I =

2.000000000000008


exitflag =

0
```

Hier erhalten wir eine Abweichung an der 15-ten Nachkommastelle zum Matlab Ergebnis.

3 Testfunktion 3

$$\int_{1.1}^{3.7} e^x - \frac{x^2}{2}\,dx$$

Zuerst berechnen wir mithilfe der Matlab-Funktion „integral“ das numerische Ergebnis des Integrals.

```
>> fun = @(x) exp(x)-(x.^2)/2

fun =

function_handle with value:

@(x)exp(x)-(x.^2)/2

>> integral(fun, 1.1,3.7)

ans =

29.222805002787631
```

Nun testen wir unsere Funktion mit Standardwerten.

```
>> [I, exitflag] = E_Trapez(@testfun3, 1.1, 3.7)

I =

```

```
29.212897596611022


exitflag =

1
```

Um das Abbruchskriterium des Verfahrens zu erreichen musste hier entweder m_{max} sehr groß gewählt oder die Genauigkeit verringert werden. Mit verringerter Genauigkeit ist das Ergebnis jedoch wie hier ersichtlich schon ungenauer, als bei obiger Ausführung.

```
>> [I, exitflag] = E_Trapez(@testfun3, 1.1, 3.7, 6, ceil((3.7-1.1)/0.1), 1.e-3)

I =

29.182340731811546


exitflag =

0
```

4 Testfunktion 4

Da wir vor allem bei „einfachen“ Funktionen genaue Ergebnisse gewünscht sind, testen wir auch solche:

$$\int_0^1 x\,dx$$

Zuerst berechnen wir mithilfe der Matlab-Funktion „integral“ das numerische Ergebnis des Integrals.

```
>> fun = @(x) x

fun =

function_handle with value:

@(x)x

>> integral(fun, 0, 1)

ans =

0.500000000000000
```

Auch mit $m_{max} = 20$ bricht das Programm noch mit $exitflag = 1$ ab.

```
>> [I, exitflag] = E_Trapez(@testfun4, 0, 1, 20)

I =

0.499999980635652


exitflag =

1
```

Deshalb verringern wir hier auch die Genauigkeit.

```
>> [I, exitflag] = E_Trapez(@testfun4, 0, 1, 6, ceil((1)/0.1), 1.e-3)

I =

0.499351851851852


exitflag =

0
```

5 Testfunktion 5

Schnell wachsende Integrale sind ebenfalls Interessant zu betrachten:

$$\int_0^1 2^x \, dx$$

Zuerst berechnen wir mithilfe der Matlab-Funktion „integral“ das numerische Ergebnis des Integrals.

```
>> fun = @(x) 2.^x

fun =

function_handle with value:

@(x)2.^x

>> integral(fun, 0, 1)

ans =

1.442695040888963
```

Nun testen wir unsere Funktion mit Standardwerten.

```
>> [I, exitflag] = E_Trapez(@testfun5, 0, 1)

I =

1.442377527965380


exitflag =

1
```

Das Programm bricht mit $exitflag = 1$ ab, deshalb verringern wir die Genauigkeit.

```
>> [I, exitflag] = E_Trapez(@testfun5, 0, 1, 6, ceil((1)/0.1), 1.e-3)

I =

1.441398244528734

```

```

exitflag =

0
```

6 Testfunktion 6

Als letzte Testfunktion wurde folgende Funktion gewählt:

$$\int_{-1}^{1} x^2 \, dx$$

Zuerst berechnen wir mithilfe der Matlab-Funktion „integral“ das numerische Ergebnis des Integrals.

```
>> fun = @(x) x.^2

fun =

function_handle with value:

@(x)x.^2

>> integral(fun, -1, 1)

ans =

0.666666666666667
```

Nun testen wir unsere Funktion mit Standardwerten.

```
>> [I, exitflag] = E_Trapez(@testfun6, -1, 1)

I =

0.666666541732919


exitflag =

1
```

Da das Programm wieder mit $exitflag = 1$ abbricht, deshalb erhöhen wir m_{max}

```
>> [I, exitflag] = E_Trapez(@testfun6, -1, 1, 11)

I =

0.666666666662302


exitflag =

0
```

Mit $m_{max} = 11$ bricht das Programm erfolgreich ab.

7 Resultate

Wenn das Programm mit den Standardwerten erfolgreich abbricht, erhalten wir duchaus genau Ergebnise in relativ kurzer Zeit. Erhöhen wir jedoch m_{max} kann das Verfahren schnell einiges an Systemressourcen benötigen.

Beispielsweise könnten wir bei der dritten Testfunktion anstatt der Verringerung der Genauiggkeit auch m_{max} erhöhen. Das führt jedoch schnell zu einem erheblichen RAM-Verbrauch.

Versuchen wir nämlich die Funktion mit $m_{max} = 50$ auszuführen, werden schnell bis zu 30 GB RAM verwendet, der Vorgang dauert ebenso sehr lange. (Vermutung - Matlab greift auf viel langsamere „swap-memory“ zu)

Vermutlich resultiert dieser Verbrauch daraus, dass wir zuerst für jedes m das ganze Tableau speichern müssen. Das könnte vermutlich durch eine etwas abgeänderte Abbruchsvorschrift gelöst werden.

Das ganze Tableau zu speichern hat natürlich den Vorteil, dass dadurch pro Iterationsschritt über m die bereits berechneten Werte nicht neu berechnet werden müssen.

Ebenfalls ergibt sich ein Speicher-Overhead durch das Speichern der linken unteren Dreiecksmatrix T, welche noch viele unnötig gespeicherte Nullen enthält.