

N U M E R I K

# Projekt 1

**Name des Tutors: Johann Faschingleitner**

**Name der Autoren:**

Christoph Mayer

Matrikelnummer: e01425430

Amanda Schöfl

Matrikelnummer: e0271473

## Aufgabe 3:

### • 3.1.:

∴

Basierend auf einer 1D Quadratur kann durch Bildung des Tensorproduktes eine 2D Quadratur erzeugt werden. Sei dazu  $f : \mathbb{R}^2 \rightarrow \mathbb{R}$  eine auf dem Einheitsquadrat  $Q := [0, 1] \times [0, 1]$  integrierbare Funktion, dann gilt (mittels Fubini):

$$\begin{aligned} \int_Q f(x, y) d(x, y) &= \int_0^1 \left( \int_0^1 f(x, y) dx \right) dy = \\ &= \int_0^1 \left( \sum_{i=0}^n \alpha_i f(x_i, y) \right) dy = \sum_{i=0}^n \alpha_i \int_0^1 f(x_i, y) dy = \\ &= \sum_{i=0}^n \alpha_i \left( \sum_{j=0}^n \alpha_j f(x_i, y_j) \right) = \sum_{i,j=0}^n \alpha_i \alpha_j f(x_i, y_j) \end{aligned} \tag{0.1}$$

wobei  $\{\alpha_i = 0, \dots, n\}$  und  $\{x_i, y_i : i = 0, \dots, n\}$  die 1D Quadraturgewichte und -knoten sind. Somit kann man mit den bereits bekannte 1D Quadraturen auch Integrale am Einheitsquadrat numerisch berechnen. Implementieren Sie mit Hilfe des zur Verfügung gestellten Programms **gauss(n)** eine Funktion **quadInt(f,n)** die  $\int_Q f(x, y) d(x, y)$  berechnet.

Für welche n wird ein Polynom  $p \in \prod_2^k := \text{span}\{x^i y^j : 0 \leq i, j \leq k\}$  exakt integriert?

```
function ret_val = quadInt(f,n)
    [nodes, weights] = gauss(n);
    tmp = 0;
    for i = 1:n+1
        for j = 1:n+1
            tmp = tmp+weights(i)*weights(j)*f(nodes(i),nodes(j));
        end
    end
    ret_val = tmp;
end
```

Die Aufgabennummerierung in Absätzen, als 3.1 unter 3.2 etc. und Text daneben

Widmen wir uns jetzt noch der Frage für welche n ein Polynom  $p \in \prod_2^k := \text{span}\{x^i y^j : 0 \leq i, j \leq k\}$  exakt integriert wird:

Wir wissen dass für 1D Quadraturen Polynome vom Grad  $2n+1$  exakt numerisch berechnet werden können. Da in (0.1) lediglich die Gauß-Quadratur 2-mal hintereinander eindimensional durchgeführt wird, gilt, dass  $i=2n+1$  und  $j=2n+1$  die höchsten Grade sind, für welche die 2D Quadratur exakt numerisch berechnet werden kann.

### • 3.2.:

Da zum Beispiel bei der Finite-Element-Methode eine Quadratur auf Dreiecken benötigt wird, verwendet man gerne die Duffy-Transformation um die Quadratur auf dem Einheitsquadrat  $\hat{Q}$  auf das Referenzdreieck  $\hat{T}$  mit den Eckpunkten  $(0, 0)$ ,  $(1, 0)$  und  $(0, 1)$  zu transformieren. Die Duffy-Transformation ist definiert als:

$$= \begin{cases} \hat{Q} \rightarrow \hat{T} \\ (s, t) \mapsto (s, (1-s)t) \end{cases} \quad (0.2)$$

Implementieren Sie mithilfe der *Duffy-Transformation* die Funktion **trigInt(f,n)**, sodass Sie Integrale auf  $\hat{T}$  berechnen können. *Hinweis:* Verwenden Sie dazu die Substitutionsregel Satz 7.34 (Transformationssatz für Integrale) aus dem Analysis-Skript von Professor Engl.

Transformationssatz für Integrale:

Es sei  $\Omega \in \mathbb{R}$  eine offene Menge und  $\Phi : \Omega \rightarrow \Phi(\Omega) \in \mathbb{R}^d$  ein Diffeomorphismus. Dann ist die Funktion  $f$  auf  $\Phi(\Omega)$  genau dann integrierbar, wenn die Funktion  $x \mapsto f(\Phi(x))|det(D\Phi(x))|$  auf  $\Omega$  integrierbar ist. In diesem Fall gilt:

$$\int_{\Phi(\Omega)} f(y)dy = \int_{\Omega} f(\Phi(x))|det(D\Phi(x))|dx. \quad (0.3)$$

Dabei ist  $D\Phi(x)$  die Jacobi-Matrix und  $det(D\Phi(x))$  die Funktionaldeterminante von  $\Phi$ , also  $Df(\Phi) = \left( \frac{\partial f_i}{\partial x_j}(x) \right)_{i,j=1,\dots,n}$

Auf unseren Fall angewandt mit  $\Omega = \hat{Q}$ ,  $\Psi = \hat{T}$ , sowie mit der Funktionaldeterminante

$$det D\hat{T} = |det \begin{pmatrix} 1 & -t \\ 0 & (1-s) \end{pmatrix}| = |(1-s)|$$

ergibt sich:

$$\int_{\hat{T}} f(x, y) d(x, y) = \int_{\hat{Q}} f(s, (1-s)t) |(1-s)| d(t, s) = \sum_{i,j=0}^n \alpha_i \alpha_j f(s_i, (1-s_i)t_j) |(1-s_i)| \quad (0.4)$$

---

Dadurch ergibt sich insgesamt:

$$\text{trigInt}(f,n) = \text{quadInt}(f \circ (s,t) \mapsto (s, (1-s)t)) * (1-s) \quad (0.5)$$

Implementierung von quadInt(f,n):

```
.--  
function ret_val = quadInt(f,n)  
    [nodes, weights] = gauss(n);  
    function ret = f_n(s,t)  
        x = s;  
        y = t*(1-s);  
        ret = abs(1-s)*f(x,y);  
    end  
    tmp = 0;  
    for i = 1:n+1  
        for j = 1:n+1  
            tmp = tmp+weights(i)*weights(j)*f_n(nodes(i),nodes(j));  
        end  
    end  
    ret_val = tmp;  
end
```

### • 3.3.:

#### Aufgabenstellung:

Erweitern Sie unter Verwendung einer affinen Transformation  $\Phi_T : \hat{T} \rightarrow T$  Ihre Implementierung von **duffyInt**, um auch Integralen auf beliebigen Dreiecken  $T$  berechnen zu können. Für welche  $n$  wird ein Polynom  $p \in \prod_2^k := \text{span}\{x^i y^j : 0 \leq i+j \leq k\}$  exakt integriert?

*Hinweis:* Sie können die Implementation Ihrer Transformation testen, indem Sie das Integral über  $f \equiv 1$  numerisch berechnen und mit der analytisch bestimmten Fläche des Dreiecks  $T$  vergleichen.

#### Durchführung:

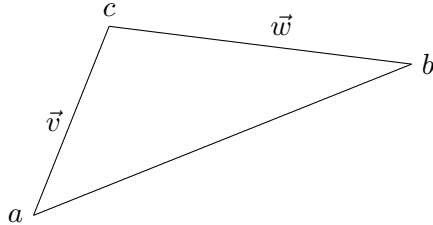
Anstatt einer affinen Transformation von  $\Phi_{\hat{T}} : \hat{T} \rightarrow T$  verwenden wir eine (direkte?) affine Transformation  $\Phi : \hat{Q} \mapsto T$ .

Wir nehmen nun ein beliebiges Dreieck  $T$  mit  $a = \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}$ ,  $b = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$ ,  $c = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}$ ,

---


$$v = \begin{pmatrix} (1-s) * a_1 + s * c_1 \\ (1-s) * a_2 + s * c_2 \end{pmatrix} \text{ und } w = \begin{pmatrix} (1-s) * b_1 + s * c_1 \\ (1-s) * b_2 + s * c_2 \end{pmatrix}$$

zur Hand, wobei die Vektoren  $\vec{v}$  und  $\vec{w}$  Konvex-Kombinationen der Punkte  $a$  und  $b$  sind:



Das (aus 3.2 oder bereits transformiert) Dreieck kann nun wie folgt dargestellt werden:

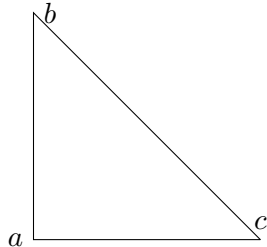
$$(1-t)v + s * w$$

und die angepasste *Duffy-Transformation* lautet wie folgt:

$$(s, t) \mapsto \begin{pmatrix} a_1 + s * (c_1 - a_1) + t * (b_1 - a_1) + st(a_1 - b_1) \\ a_2 + s(c_2 - a_2) + t(b_2 - a_2) + st(a_2 - b_2) \end{pmatrix}.$$

Durch Proberechnung anhand des Beispiel unseres Referenzdreiecks, mit

$$a = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, c = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, b = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \text{ folgt:}$$



$$(s, t) \mapsto c = \begin{pmatrix} t - st \\ s \end{pmatrix} = \begin{pmatrix} t * (1-s) \\ s \end{pmatrix}$$

Damit ist unsere Transformation korrekt und wir können uns an das Implementieren des Codes wagen:

```
function ret_val = duffyInt(f,n,a1,a2,b1,b2,c1,c2)
    [nodes, weights] = gauss(n);
    function ret = f_n(s,t)
        x = a1*s*t-b1*s*t-a1*s-a1*t+b1*t+c1*s+a1;
        y = a2*s*t-b2*s*t-a2*s-a2*t+b2*t+c2*s+a2;
```

---

```

        ret = abs(-a1*b2*s+a1*c2*s+a2*b1*s-a2*c1*s-b1*c2*s+b2*c1*s+a1*b2
        -a1*c2-a2*b1+a2*c1+b1*c2-b2*c1)*f(x,y);
    end
    tmp = 0;
    for i = 1:n+1
        for j = 1:n+1
            tmp = tmp+weights(i)*weights(j)*f_n(nodes(i),nodes(j));
        end
    end
    ret_val = tmp;
end

```

Widmen wir uns nun noch der Frage wann ein Polynom  $p \in \prod_2^k := \text{span}\{x^i y^j : 0 \leq i + j \leq k\}$  exakt integriert werden kann:

Da wir nur die Hülle  $\text{span}\{x^i y^j : 0 \leq i + j \leq k\}$  betrachten können wir das Polynom  $p = x^i y^j$  mit den höchstmöglichen Graden  $i$  und  $j$  heranziehen.

Sei nun  $p \in \prod_2^k \Rightarrow \int_{\hat{T}} p(x, y) d(x, y) = \int_{\hat{Q}} p(\Phi(x), \Phi(y)) * \det|d\Phi| d(s, t)$ .

Für  $p = x^i y^j$  lautet die Transformation wie folgt:

$$p(\Phi(x), \Phi(y)) = (\alpha_1 + s\alpha_2 + t\alpha_3 + st\alpha_4)^i (\beta_1 + s\beta_2 + t\beta_3 + st\beta_4)^j.$$

Aus diesem Ausdruck lesen wir heraus dass der höchste Term

$$\delta_0 s^i t^j s^j t^j = s^{(i+j)} t^{(i+j)} \delta_0$$

mit einer Konstanten  $\delta_0$  ist, die in diesem Fall vernachlässigt werden kann.

Hingegen ist der höchste Term der Funktionaldeterminante  $\det|D\Phi|$ :

$\delta_1 st$  KORRIGIEREN, schöner SCHREIBEN WO FEHLT OBEN DIE TRANSFORMATION BEI DER DIE FUNKTIONALDETERMINANTE NOCH ANGEHÄNGT WERDEN MUSS

Somit ist der größte Term von  $\int_{\hat{Q}} p(\Phi(x), \Phi(y)) * \det|d\Phi| d(s, t)$  vom Grad  $i+j+1 t^{i+j+1}$ . Da  $i + j + 1 \leq 2n + 1$  gefordert wird, ergibt sich  $i + j \leq 2n$ .

### • 3.4.:

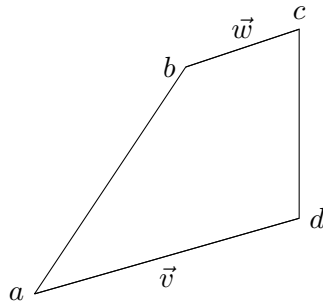
Erweitern Sie Ihre Implementierung von **[quadInt(f,n)]** zur Berechnung von Integralen auf beliebigen konvexen Vierecken  $Q = \text{conv}(a1, a2), (b1, b2), (c1, c2), (d1, d2)$ . Verwenden Sie dazu eine Transformation der Form:

$$\Psi_{\hat{Q}} = \begin{cases} \hat{Q} \rightarrow Q \\ (u, v) \mapsto \begin{pmatrix} \alpha_1 + \alpha_2 u + \alpha_3 v + \alpha_4 uv \\ \beta_1 + \beta_2 u + \beta_3 v + \beta_4 uv \end{pmatrix} \end{cases} \quad (0.6)$$

---

mit den Konstanten  $\alpha_i, \beta_i \in R$ . Überlegen Sie sich, warum nicht-konvexe Vierecke unzulässig sind.

Wir sollen nun unsere Implementierung auf beliebige konvexe Vierecke



erweitern.

Die Konvexkombination funktioniert analog zu der in Aufgabe (3.3):

$$\vec{v} * (1 - t) + \vec{w} * t$$

mit

$$\vec{v} = a * t + d * (1 - t) \text{ und } \vec{w} = b * t + c * (1 - t) .$$

Wenn man dies ausmultipliziert ergibt das genau die Transformation die in der Aufgabenstellung angegeben ist. (ausrechnen, wie bau ich das d ein?)

Letztlich widmen wir uns der Frage wir nur konvexe Vierecke verwenden dürfen: Da die Vektoren konvexkombiniert werden ist es augenscheinlich dass nur konvexe Vierecke für unsere Transformation geeignet sind..

Hier nun die Implementation von QuadInt:

WO SETZE ICH IN DIE KONVEXKOMBINATION EIN ALSO DIE PARAMETER ALPHA UND CO ALS T UND S; ERGÄNZUNG!!

Widmen wir uns nun wieder zur Implementierung des Codes:

```
function ret_val = quadInt2(f,n,a1,a2,b1,b2,c1,c2,d1,d2)
    [nodes, weights] = gauss(n);
    function ret = f_n(u,v)
        x = (d1-b1+a1)*u*v+(c1-a1)*u+(b1-a1)*v+a1;
        y = (d2-b2+a2)*u*v+(c2-a2)*u+(b2-a2)*v+a2;
        ret = abs(a1*b2*u-a1*c2*u-a1*d2*u+a1*d2*v-a2*b1*u+a2*c1*u+a2*d1*v-
            a2*d1*v+b1*c2*u-b1*d2*v-b2*c1*u+b2*d1*v+c1*d2*u-c2*d1*u-a1*c1*
            +a1*c2+a2*b1-a2*c1-b1*c2+b2*c1)*f(x,y);
    end
    tmp = 0;
    for i = 1:n+1
        for j = 1:n+1
            tmp = tmp+weights(i)*weights(j)*f_n(nodes(i),nodes(j));
        end
    end
```

---

```

    end
    ret_val = tmp;
end

```

• **3.5.:**

Testen Sie ihre Quadraturen für  $\int_G f(x, y) d(x, y)$  :

- $f(x, y) = x^7 + 3x^4y^4 + 3x^2y + 7y^6$ ,  $G = \text{conv}\{(0, 0), (0, 5, -0, 5), (1, 1)\}$   
 Stimmt das benötigte  $n$  um das Polynom exakt zu integrieren mit den theoretischen Überlegen zusammen? Wenn nicht, wieso?

```

f=@(x,y) x^7+3*x^4*y^4+3*x^2*y+7*y^6;
n_max = 5;
integral = zeros(1,n_max+1);
for n = 1:n_max+1
    integral(n) = duffyInt(f,n-1,0,0,0.5,-0.5,1,1);
end

plot error
error = integral-0.2877808780;
plot(0:n_max,error)
set(gca,'xtick',[min(xlim):max(xlim)])
set(gca,'xticklabel',[0:5])
xlabel('n')
ylabel('Fehler')

%{
    integral =    0.3568    0.3057    0.2880    0.2878    0.2878
}%

```

Transformation und Integration mit Maple von  $\int_0^1 \int_0^1 f(d_1(s, t), d_2(s, t)) |det(md)| ds dt$ .

Die Jacobi-Matrix lautet wie folgt und die daraus resultierende Determinante lautet wie folgt:

Wodurch sich ein Gesamtergebnis von



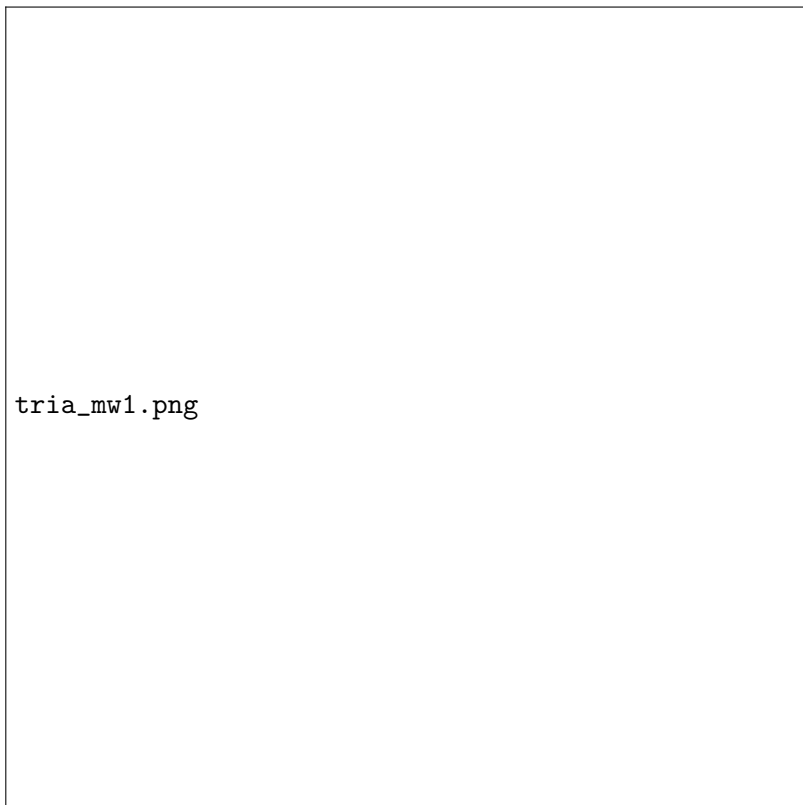


Abbildung 0.1: Erstes Bild



Abbildung 0.2: Erstes Bild

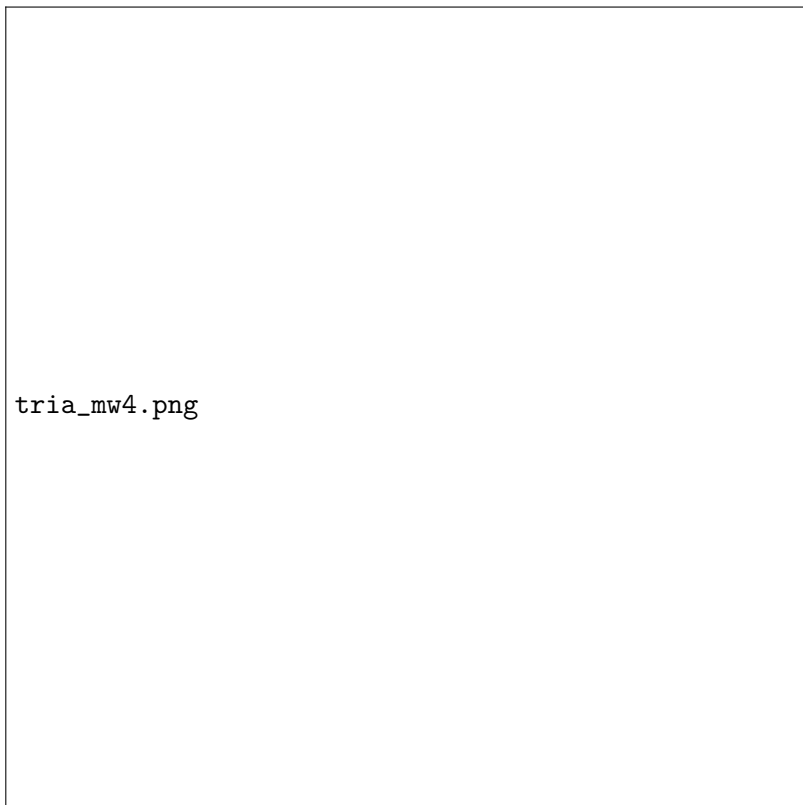



Abbildung 0.3: Erstes Bild

---



tria\_mw4.png

ergibt.

Plot:

---

plot1.png

–  $f(x, y) = \sin(50x) \sin(50y)$ ,  $G = \text{conv}\{(1, 0), (10.2), (3, 4)(-1, 1)\}$

```
f=@(x,y) sin(50*x)*sin(50*y);
n_max = 160;
integral = zeros(1,n_max+1);
for n = 1:n_max+1
    integral(n) = quadInt2(f,n-1,1,0,10,2,3,4,-1,1);
end

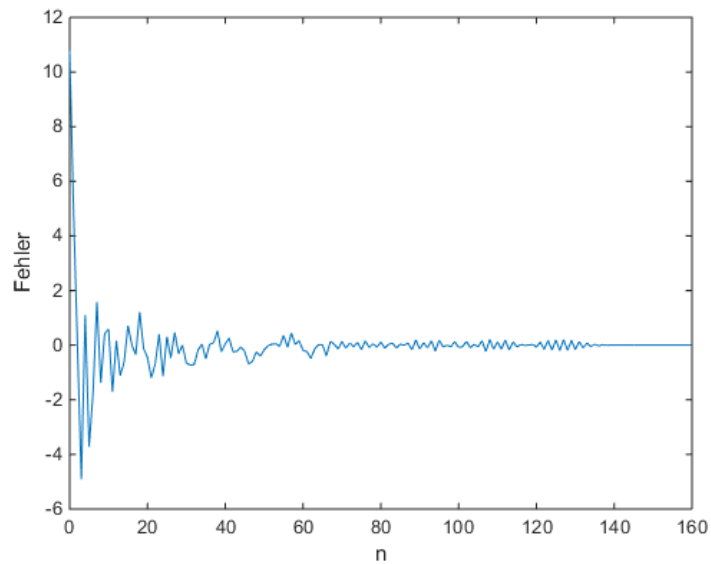
error = integral - (-8/625 + (37/2500)*cos(50) + (27/125000)*sin(50) - ...
    (27/125000)*cos(50)*sin(50) - (1/500)*cos(50)^2);
plot(0:n_max,error)
xlabel('n')
ylabel('Fehler')

%{
    Ergebnis: konvergiert relativ langsam (ab n=147 konstant bis zur
    Nachkommastelle) gegen 0.0003
%}
```

---

"plot2.png

Abbildung 0.4: Erstes Bild



$$- f(x, y) = \sqrt{\frac{9y}{(1-x^2)}}, G = \hat{Q}$$

```
f=@(x,y) sqrt(9*y/(1-x^2));  
n_max = 1000;  
integral = zeros(1,n_max);  
for n = 1:n_max  
    integral(n) = quadInt(f,n);  
end
```

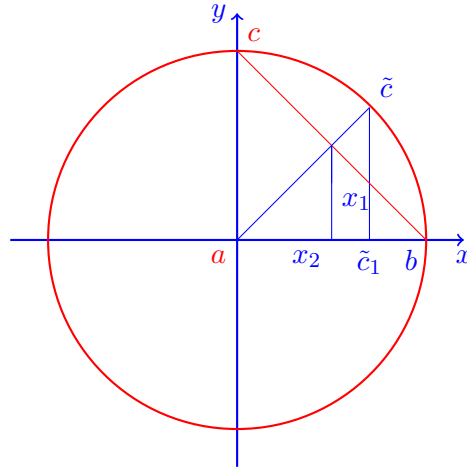
```
error = integral-pi;  
semilogx(1:n_max,error)  
plot(1:n_max, error)  
xlabel('n')  
ylabel('Fehler')
```

```
%{  
    Ergebnis: konvergiert sehr langsam gegen pi  
}%
```

$$- f(x, y) = \begin{cases} 1, & x^2 + y^2 \leq 1 \\ 0, & \text{sonst} \end{cases}, G = \mathbb{R}^2$$

*Hinweis:* Approximieren Sie den Kreis durch ein n-Eck. Verwenden Sie die letzten beiden Beispiele um  $\pi$  zu approximieren. Nehmen Sie als Referenzwert bei allen Integralen den analytisch bestimmten Integralwert, sofern dies möglich ist und stellen Sie die Fehler grafisch dar.

Betrachten wir folgende Skizze:



Es ergibt sich dass  $x = (\frac{a_1+b_1}{2}, \frac{a_2+b_2}{2})$  durch den Satz von Pythagoras.(?) Weiters folgt mit  $\tan \gamma = \frac{x_2}{x_1} = \frac{\sin \gamma}{\cos \gamma} = \frac{\tilde{c}_2}{(\tilde{c}_1)}$ .

Wir wissen aus

$$\sin(x)^2 + \cos(x)^2 = 1 \quad (0.7)$$

, dass

$$(\tilde{c}_1)^2 + (\tilde{c}_2)^2 = 1 \Rightarrow (\tilde{c}_1)^2 + (\tilde{c}_1)^2 * \frac{(x_2)^2}{(x_1)^2} = 1. \quad (0.8)$$

Daraus ergibt sich wiederum

$$\tilde{c}_1 \frac{x_2}{x_1} = \tilde{c}_2 \text{ und } \tilde{c}_1^2 (1 + \frac{(x_2)^2}{(x_1)^2}) = 1. \quad (0.9)$$

Somit erhalten wir

$$\tilde{c}_2 = \frac{x_2}{x_1} * \sqrt{\frac{1}{(1 + \frac{(x_2)^2}{(x_1)^2})}} \text{ und } \tilde{c}_1 = \sqrt{\frac{1}{(1 + \frac{(x_2)^2}{(x_1)^2})}} \quad (0.10)$$

---

und schließlich als Endresultat

$$\frac{x_2}{x_1} = \frac{\frac{c_2+b_2}{2}}{\frac{c_1+b_1}{2}} = \frac{c_2}{c_1+1}.$$

Dieses Verfahren wird nun iterativ angewandt um das gewünschte Resultat zu erreichen.

Der folgende Code repräsentiert dies:

```
f=@(x,y) 1;
n_max = 7; % maximale Anzahl der Ecken
integral = zeros(1,n_max-1);
c1=0;
c2=1;
for n = 2:n_max
    integral(n-1) = 2^n*duffyInt(f,0,0,0,1,0,c1,c2);
    c1_tmp=sqrt(1/(1+(c2^2/(c1+1)^2)));
    c2=c2/(c1+1)*sqrt(1/(1+(c2^2/(c1+1)^2)));
    c1 = c1_tmp;
end

error = integral-pi;
plot(1:n_max-1,error)
set(gca,'xtick',([min(xlim):max(xlim)]))
set(gca,'xticklabel',arrayfun(@(x) 2^x, [1:n_max-1]))
xlabel('Anzahl an Ecken zur Approximation')
ylabel('Fehler')

%{
    Output:   2.0000      2.8284      3.0615      3.1214      3.1365
3.1403
              3.1413      3.1415      3.1416
%}
```

Der folgende Plot veranschaulicht das Resultat graphisch: (loglog Plot missing)



---

"plot4.png

Abbildung 0.5: Erstes Bild

