NUMERIK

Projekt 1

Name des Tutors: Johann Faschingleitner

Name der Autoren:

Christoph Mayer

Matrikelnummer: e01425430

Amanda Schöfl

Matrikelnummer: e0271473

Aufgabe 3

3.1 Aufgabenstellung

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

$$\begin{array}{rcccl} \int\limits_{\hat{Q}} f(x,y)d(x,y) & = & \int_0^1 \Big(\int_0^1 f(x,y)dx \Big) dy & = & \\ & = & \int_0^1 \Big(\sum_{i=0}^n \alpha_i f(x_i,y) \Big) dy & = & \\ & = & \sum_{i=0}^n \alpha_i \int_0^1 f(x_i,y)dy & = & \\ & = & \sum_{i=0}^n \alpha_i \Big(\sum_{j=0}^n \alpha_j f(x_i,y_i) \Big) & = & \sum_{i,j=0}^n \alpha_i \alpha_j f(x_i,y_j), \end{array} \tag{0.1}$$

wobei $\{\alpha_i : i = 0, ...n\}$ und $\{x_i, y_j : i, j = 0, ...n\}$ 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 $\int_{\hat{Q}} f(x,y)d(x,y)$ berechnet. Für welche n wird ein Polynom $p \in \prod_2^k := span\{x^i y^j : 0 \leq i, j \leq k\}$ exakt integriert?

3.1 Durchführung

Die folgende Implementierung der Funktion **quadInt**, hat die Inputparameter

- `f` ein Function Handle einer Funktion mit oben beschriebenen Eigenschaften,
- `n` der Grad der Gaußquadratur.

Sie gibt den approximierten Wert des Integrals der übergebenen Funktion auf $\hat{Q}$ zurück.

```
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
```

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

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:

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

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.

3.2 Durchführung

Transformationssatz für Integrale:
Es sei $\Omega \in \mathbb{R}$ eine offene Menge und $\Phi : \Omega \to \Phi(\Omega) \in \mathbb{R}^d$ 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 Ω integrierbar ist. In diesem Fall gilt:

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

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

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

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

ergibt sich

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

Dadurch ergibt sich insgesamt:

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

3.2 Code

Die folgende Implementierung der Funktion **trigInt**, hat die Inputparameter

- `f` ein Function Handle einer Funktion mit oben beschriebenen Eigenschaften,
- `n` der Grad der Gaußquadratur.

Sie gibt den approximierten Wert des Integrals nach der Transformation auf $\hat{T}$ auf das Referenzdreieck mit den Eckpunkten $(0,0),(0,1),(1,0)$ zurück.

```
--
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 $\Phi_T : \hat{T} \to T$ Ihre Implementierung von **duffyInt**, um auch Integralen auf beliebigen Dreiecken T berechnen zu können. Für welche n wird ein Polynom $p \in \prod_2^k := span\{x^i y^j : 0 \leq i + j \leq k\}$ 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.

3.3 Durchführung

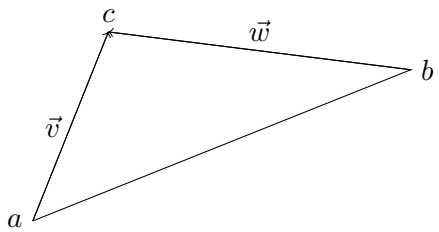
Anstatt einer affinen Transformation von $\Phi_{\hat{T}} : \hat{T} \to T$ verwenden wir eine affine Transformation $\Phi : \hat{Q} \mapsto T$.

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=$\begin{pmatrix} (1-s) * a_1 + s * c_1 \\ (1-s) * a_2 + s * c_2 \end{pmatrix}$ 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:



Die Transformation erfolgt mit

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

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

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

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

.

und wir sehen, dass unsere Transformation korrekt funktioniert.

Die Funktionaldeterminate berechnen wir mit Maple:

$v1 := c1 \cdot s + (1-s) \cdot a1$

$c1\,s + (1-s)\,a1$

$v2 := c2 \cdot s + (1-s) \cdot a2$

$c2\,s + (1-s)\,a2$

$w1 := c1 \cdot s + (1-s) \cdot b1$

$c1\,s + (1-s)\,b1$

$w2 := c2 \cdot s + (1-s) \cdot b2$

$c2\,s + (1-s)\,b2$

$d1 := simplify(t \cdot w1 + (1-t) \cdot v1)$

$a1\,s\,t - b1\,s\,t - a1\,s - a1\,t + b1\,t + c1\,s + a1$

$d2 := simplify(t \cdot w2 + (1-t) \cdot v2)$

$a2\,s\,t - b2\,s\,t - a2\,s - a2\,t + b2\,t + c2\,s + a2$

Variablendeklaration

$$\begin{bmatrix} a1\,s - b1\,s - a1 + b1 & a1\,t - b1\,t - a1 + c1 \\ a2\,s - b2\,s - a2 + b2 & a2\,t - b2\,t - a2 + c2 \end{bmatrix}$$

$$-a1\,b2\,s + a1\,c2\,s + a2\,b1\,s - a2\,c1\,s - b1\,c2\,s + b2\,c1\,s + a1\,b2 - a1\,c2 - a2\,b1 + a2\,c1 + b1\,c2 - b2\,c1$$

Jacobi-Matrix und Determinantenberechnung

3.3 Code

Die folgende Implementierung der Funktion **duffyInt**, hat die Inputparameter

- `f` ein Function Handle einer Funktion mit oben beschriebenen Eigenschaften,
- `n` der Grad der Gaußquadratur.
- a_1, a_2 : Koordinaten des Punktes a
- b_1, b_2 : Koordinaten des Punktes b
- c_1, c_2 : Koordinaten des Punktes c

Sie gibt den approximierten Wert des Integrals nach der Transformation auf ein beliebiges Dreieck mit den Eckpunkten (a,b,c) zurück:

```
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
$p \in \prod_2^k := span\{x^i y^j : 0 \leq i+j \leq k\}$ exakt integriert werden kann:

Da wir nur die Hülle $span\{x^i y^j : 0 \leq i+j \leq k\}$ 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 $\delta_0 s^i t^i s^j t^j = s^{(i+j)}t^{(i+j)}\delta_0$ mit einer Konstanten δ_0 ist, die in diesem Fall vernachlässigt werden kann.

Hingegen ist der höchste Term der Funktionaldeterminante $det|D\Phi|$ $\delta_1 st$.
Somit ist der größte Term von $\int_{\hat{Q}} p(\Phi(x),\Phi(y)) * det|d\Phi|d(s,t)$ vom Grad $\delta^{i+j+1}t^{i+j+1}$.
Da $i+j+1 \leq 2n+1$ gefordert wird, muss $i+j \leq 2n$ sein.

3.4 Aufgabenstellung

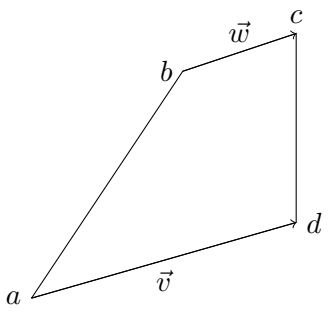
Erweitern Sie Ihre Implementierung von **quadInt(f,n)** zur Berechnung von Integralen auf beliebigen konvexen Vierecken $Q = conv(a1, a2), (b1, b2), (c1, c2), (d1, d2)$. Verwenden Sie dazu eine eine Transformation der Form:

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

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

3.4 Durchführung

Wir sollen nun nun unsere Implementierung auf beliebige konvexe Vierecke



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

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

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

Wenn man dies ausmultipliziert ergibt das genau die Transformation die in der Aufgabenstellung angegeben ist.

Die Berechnung der Funktionaldeterminante erfolgt wieder durch Maple:

$$alpha1 := a1$$

$$1$$

$$alpha2 := c1 - a1$$

$$2$$

$$alpha3 := b1 - a1$$

$$9$$

$$alpha4 := d1 - b1 + a1$$

$$-10$$

$$beta1 := a2$$

$$0$$

$$beta2 := c2 - a2$$

$$4$$

$$beta3 := b2 - a2$$

$$2$$

$$beta4 := d2 - b2 + a2$$

$$-1$$

Variablendeklaration

$$q1 := alpha1 + alpha2 \cdot u + alpha3 \cdot v + alpha4 \cdot u \cdot v$$

$$-10\,u\,v + 2\,u + 9\,v + 1$$

$$q2 := beta1 + beta2 \cdot u + beta3 \cdot v + beta4 \cdot u \cdot v$$

$$-u\,v + 4\,u + 2\,v$$

$$mq := Matrix([[diff(q1, u), diff(q1, v)], [diff(q2, u), diff(q2, v)]])$$

$$\begin{bmatrix} -10\,v + 2 & -10\,u + 9 \\ -v + 4 & -u + 2 \end{bmatrix}$$

$$det := simplify(Determinant(mq))$$

$$38\,u - 11\,v - 32$$

Jacobi-Matrix und Determinantenberechnung

• 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.

3.4 Code

Die folgende Implementierung der Funktion **quadInt2**, hat die Inputparameter

- `f` ein Function Handle einer Funktion mit oben beschriebenen Eigenschaften,
- `n` der Grad der Gaußquadratur.
- a_1, a_2 : Koordinaten des Punktes a
- b_1, b_2 : Koordinaten des Punktes b
- c_1, c_2 : Koordinaten des Punktes c
- d_1, d_2 : Koordinaten des Punktes d

```
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*u...
            -a2*d1*v+b1*c2*u-b1*d2*v-b2*c1*u+b2*d1*v+c1*d2*u-c2*d1*u-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
```

3.5 Aufgabenstellung

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

- $f(x,y) = x^7 + 3x^4y^4 + 3x^2y + 7y^6, G = conv\{(0,0),(0,5,-0,5),(1,1)\}$

Stimmt das benötigte n um das Prolynom exakt zu integrieren mit den theoretischen Überlegen zusammen? Wenn nicht, wieso?

Berechnung mittels Maple:

```
a1 := 0;
a2 := 0;
b1 := 0.5;
b2 := -0.5;
c1 := 1;
c2 := 1;
```

0
0
0.5
−0.5
1
1

$$simplify\left(int\left(int\left(\left(d1^7 + 3\cdot d1^4\cdot d2^4 + 3\cdot d1^2\cdot d2 + 7\cdot d2^6\right)\cdot det, s = 0..1\right), t = 0..1\right)\right)$$

0.2877808780

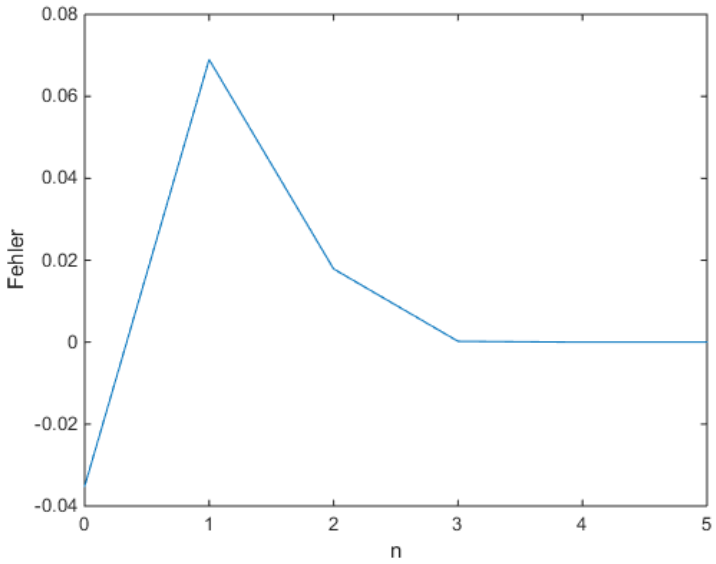
Endresultat

Testen der Quadratur für die Funktion
$f(x,y) = x^7 + 3x^4y^4 + 3x^2y + 7y^6, G = conv\{(0,0),(0,5,-0,5),(1,1)\}$:

```
f=@(x,y) x^7+3*x^4*y^4+3*x^2*y+7*y^6;
n_max = 10;
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
integral

% plot error
error = integral -0.2877808780;
loglog(0:n_max,abs(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
%}
```

Plot:

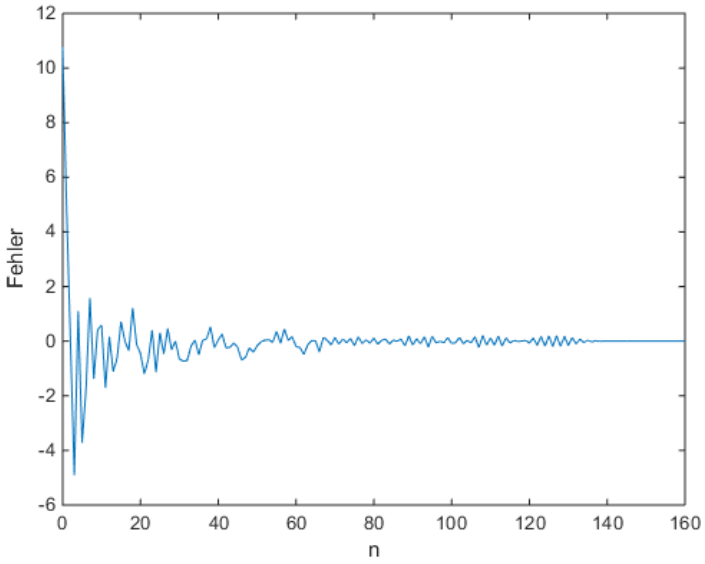


Plot 3.5.1

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

```
f=@(x,y) sin(50*x)*sin(50*y);
n_max = 147;
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 = abs(integral-(-8/625+(37/2500)*cos(50)+(27/125000)*sin(50)-...
    (27/125000)*cos(50)*sin(50)-(1/500)*cos(50)^2));
loglog(0:n_max,error)
xlabel('n')
ylabel('Fehler')
%loglog
```

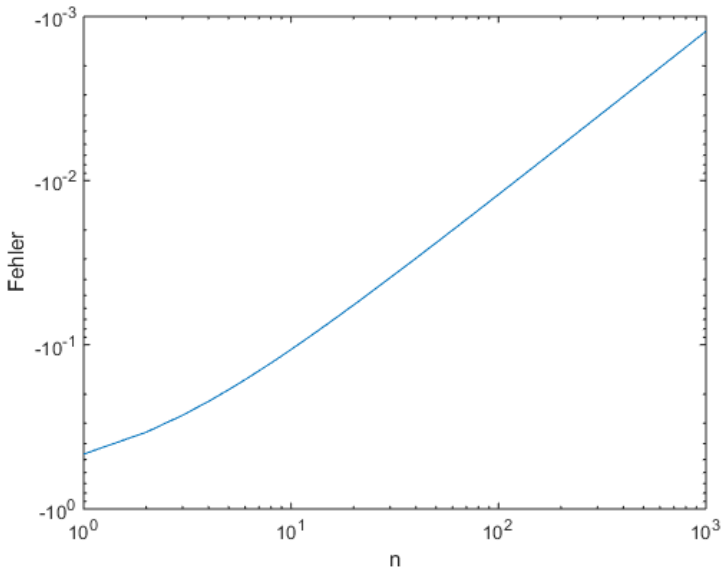
```
%{
    Ergebnis: konvergiert relativ langsam (ab n=147 konstant bis zur 4.
    Nachkommastelle) gegen 0.0003
%}
```



Plot 3.5.2

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

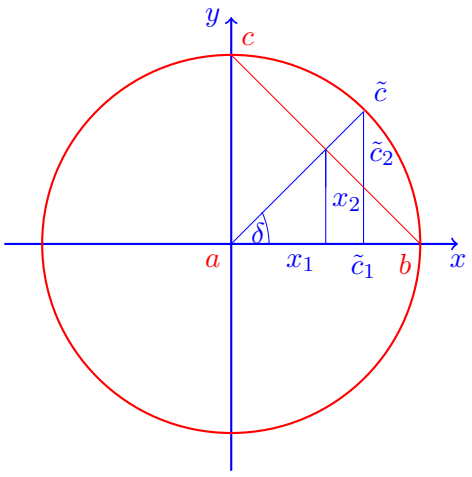
```
f=@(x,y) sqrt(9*y/(1-x^2));
n_max = 1000;
integral = zeros(1,n_max);
for n = 1:n_max
    integral(n) = quadInt(f,n);
end

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

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



Plot 3.5.3

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

Hinweis: Approximieren Sie den Kreis durch ein n-Eck. Verwenden Sie die letzten beiden Beispiele um π zu approximieren. Nehmen Sie als Referenzwert bei allen Integralen den analytisch bestimmten Integralwert, sofern dies mölgich ist und stellen Sie die Fehler grafisch dar.
Betrachten wir folgende Skizze:



Durch den Satz von Pythagoras ergibt sich:

$$x = (\tfrac{a_1+b_1}{2}, \tfrac{a_2+b_2}{2})).$$

Weiters folgt

$$tan\gamma = \tfrac{x_2}{x_1} = \tfrac{\sin\gamma}{\cos\gamma} = \tfrac{\tilde{(}c_2)}{\tilde{(}c_1)}.$$

Wir wissen aus

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

, dass

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

Daraus ergibt sich wiederum

$$\tilde{c}_1 \tfrac{x_2}{x_1} = \tilde{c}_2 \text{ und } \tilde{c}_1^2(1 + \tfrac{(x_2)^2}{(x_1)^2}) = 1$$

Somit erhalten wir

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

und schließlich als Endresultat

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

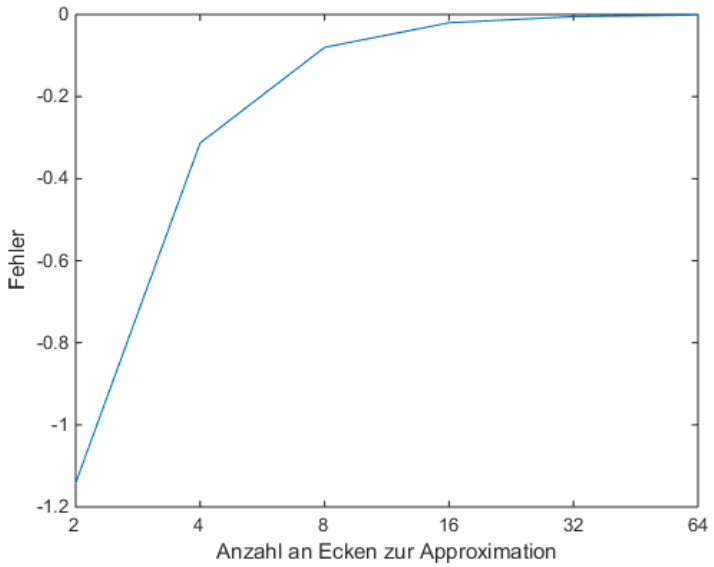
.

Dieses Verfahren wird nun iterativ angewandt um das gewünschte Resultat zu erreichen. Der folgende Code repräsentiert dies:

```
f=@(x,y) 1;
n_max = 7; % 2^n_max ... 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;
semilogy(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)



Plot 3.5.4