Numerische Approximation von partiellen Differentialgleichungen

Carla Grolimund & Laura Ricklin

22. Dezember 2011

Inhaltsverzeichnis

1 Gewöhnliche Differentialgleichungen (ODE)

Eine ODE ist eine Gleichung, die nur von einer Variablen (t = Zeit) abhängig ist. Die Problemstellung lautet:

Sei $T > 0$ und m die Dimension vom Raum vom Bild der Funktion u.
Finde eine Funktion $u : [0,T] \longrightarrow \mathbb{R}^m$ so, dass

$$\begin{aligned} \dot{u}(t) &= f(t,u(t)), \\ u(0) &= u_0, \end{aligned}$$

wobei $f : [0,T] \times \mathbb{R}^m \longrightarrow \mathbb{R}^m$ eine gegebene genügend glatte Funktion ist.
Zuerst unterteilen wir das Intervall I = [0,T] in $0 = t_0 < t_1 < \ldots < t_N = T$, wobei $N \in \mathbb{N}$ und $t_0, t_1, \ldots, t_N$ die Zeitpunkte sind, wo die numerische Lösung berechnet

wird. Definiere $I_n = [t_n, t_{n+1}]$ als das Intervall zwischen den einzelnen Zeitpunkten. Am einfachsten ist die äquidistante Verteilung:

$$t_n = t_0 + n \cdot h, \quad h = \frac{T}{N}.$$

Wir präsentieren zwei verschiedene Wege um das Problem numerisch zu lösen.

1.1 Methoden basierend auf endliche Differenzen

Wir schreiben $\dot{u}(t) = f(t, u(t))$ als $\dot{u}(t_n) = f(t_n, u(t_n))$ und approximieren $\dot{u}(t_n)$ mit Differenzenquotienten. Wir präsentieren drei verschiedene Möglichkeiten $\dot{u}(t_n)$ zu approximieren, nämlich mit dem Vorwärts-, Rückwärts- und symmetrischen Differenzenquotienten.

z.B. benützen wir für das Explizit Euler-Verfahren den Vorwärtsdifferenzenquotienten $(D^+u(t_n))$:

$$\dot{u}(t_n) \approx D^+u(t_n) = \frac{u(t_{n+1}) - u(t_n)}{h}$$

und erhalten durch Umformen die erste der folgenden Gleichungen:

$$\begin{aligned} u_{n+1} &= u_n + hf(t_n, u_n) &&\longrightarrow \text{ Explizit Euler-Verfahren} \\ u_{n+1} &= u_n + hf(t_{n+1}, u_{n+1}) &&\longrightarrow \text{ Implizit Euler-Verfahren} \\ u_{n+1} &= u_{n-1} + 2hf(t_n, u_n) &&\longrightarrow \text{ Leapfrog-Verfahren.} \end{aligned}$$

Wobei u_{n+1} die Approximation von der exakten Lösung $u(t_{n+1})$ ist.

1.2 Methoden basierend auf Quadraturformeln

Wir integrieren $\dot{u}(t) = f(t, u(t))$ im Intervall $I_n = [t_n, t_{n+1}]$ und erhalten:

$$u(t_{n+1}) - u(t_n) = \int_{t_n}^{t_{n+1}} f(s, u(s))\, ds = \mathscr{I}_n.$$

Nun approximieren wir das Intergral $\mathscr{I}_n$ mit Hilfe von Quadraturformeln:

$$\begin{aligned} \mathscr{I}_n &\approx hf(t_n, u_n) &&\longrightarrow \text{ Explizit Euler-Verfahren} \\ \mathscr{I}_n &\approx hf(t_{n+1}, u_{n+1}) &&\longrightarrow \text{ Implizit Euler-Verfahren} \\ \mathscr{I}_n &\approx hf(t_n + h/2, u(t_n + h/2)) &&\longrightarrow \text{ Modifiziert Euler} \\ \mathscr{I}_n &\approx \frac{h}{2}\left[f(t_n, u_n) + f(t_{n+1}, u_{n+1})\right] &&\longrightarrow \text{ Crank-Nicolson.} \end{aligned}$$

1.3 Definition

Wir erhalten nun numerische Verfahren der Art:

$$u_{n+1} = F\left(h; t_{n+1}, u_{n+1}; t_n, u_n; \ldots\right)$$

z.B. für das Explizit Euler-Verfahren gilt:

$$u_{n+1} = F(h; t_n, u_n) \quad \text{mit} \quad F(h; t_n, u_n) = u_n + hf(t_n, u_n),$$

weil es nur vom vorherigen Wert abhängt.

1. Falls F von q vorherigen Werten $u_{n-j}, j = 0, \ldots, q-1$, abhängt, sagt man, dass es ein q-Schrittverfahren ist.

2. Falls F nicht von der Lösung zum Zeitpunkt t_{n+1} abhängt, sagt man, es sei explizit. Sonst sagt man, dass es implizit ist.

1.4 Approximationsfehler

Der Approximationsfehler lautet:

$$e_n = u\left(t_{n+1}\right) - F\left(h; t_{n+1}, u_{n+1}; t_n, u\left(t_n\right); \ldots\right),$$

wobei $u(t)$ die Lösung des ODE $\dot{u}(t) = f(t, u(t))$ und F die übliche Form eines numerischen Verfahrens für das ODE ist.
Die Fehlerordnung ist p, falls $e_n = \mathscr{O}\left(h^{p+1}\right)$, $h \longrightarrow 0$. Falls $p \geq 1$ sagt man, dass das Verfahren konsistent ist.

1.5 Stabilität

Wir betrachten hier die Absorptionsgleichung für $f = 0$ und $\alpha \in \mathbb{R}^+$:

$$\begin{aligned} \dot{u}(t) + \alpha u(t) &= 0 \\ u(0) &= u_0. \end{aligned}$$

Die exakte Lösung ist dann $u(t) = \exp(-\alpha t)u_0$ mit $\lim_{t\to\infty} u(t) = 0$. Falls man diese Lösung mit dem Explizit Euler-Verfahren berechnet kriegt man eine Folge von Werten $u_n = (1-\alpha h)^n u_0$:

- Falls $h > \dfrac{2}{\alpha}$, dann ist $1 - \alpha h \leq -1$ und die Folge (u_n) divergiert;

- Falls $0 < h < \dfrac{2}{\alpha}$, dann ist $\mid 1 - \alpha h \mid < 1$ und die Folge (u_n) konvergiert gegen 0 für $t \to \infty$.

Beispiel: Wir betrachten die Absorptionsgleichung für $f = 0$, $\alpha = 4$ und $u_0 = 1$:

$$\begin{aligned} \dot{u}(t) + \alpha u(t) &= 0 \\ u(0) &= 1. \end{aligned}$$

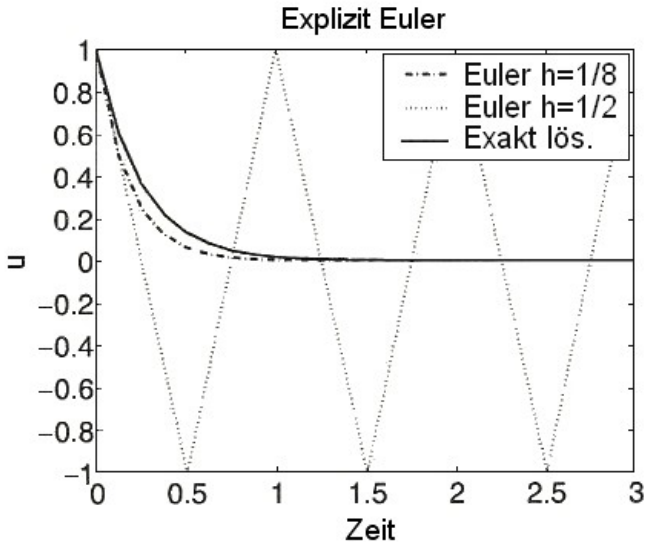


Abbildung 1: Explizit Euler-Verfahren

In der Abbildung 1 sieht man, dass das Explizit Euler-Verfahren stabil ist, nur falls die Schrittweite h klein genug ist.

In der Abbildung 2 haben wir das Problem mit dem Implizit Euler-Verfahren gelöst. Man sieht, dass die numerische Lösung genauer ist, je kleiner h gewählt wird. In der oberen Graphik haben wir h grösser gewählt als in der unteren Graphik, deshalb ist diese Lösung ungenauer. Es gibt keine Probleme mit der Stabilität (d.h.: man muss nicht auf die Wahl von h achten).

2 Partielle Differentialgleichungen (PDE)

2.1 Die Konvektionsgleichung

Die Problemstellung lautet:

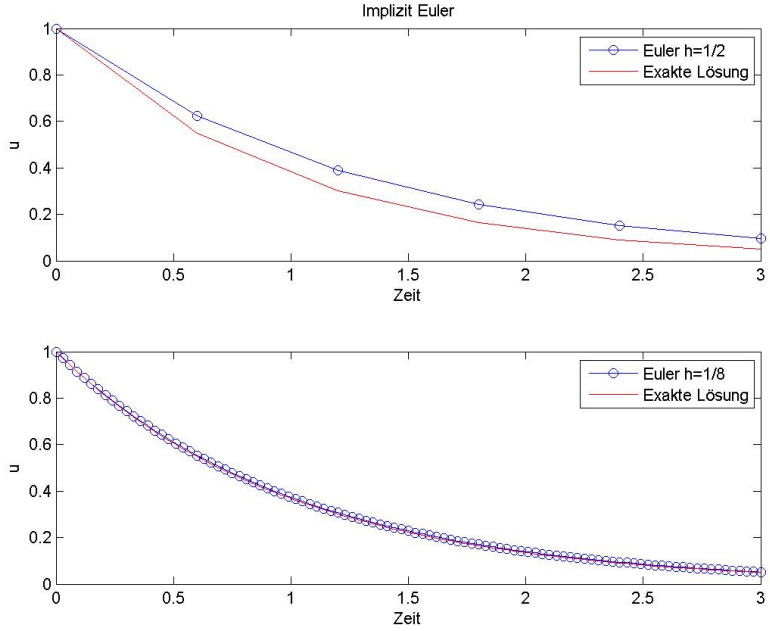


Abbildung 2: Implizit Euler-Verfahren

Wir suchen eine Funktion $u(x,t)$ so, dass

$$\begin{aligned} \partial_t u(x,t) + c\partial_x u(x,t) &= f(x,t), \quad x \in]a,b[\quad \text{und} \quad t > 0, \\ u(x,0) &= u_0(x), \\ u(a,t) &= \varphi(t). \end{aligned}$$

Die Konvektionsgleichung beschreibt die Konvektion oder den Transport von einer Menge $u(x,t)$ mit Geschwindigkeit c (konst.), d.h. sie hängt von der Zeit (t) und dem Ort (x) ab. Die Funktion $f(x,t)$ (gegeben) modelliert normalerweise die Produktion von u in der Zeit. Seien $a,b \in \mathbb{R}$ mit $a < b$, $u_0(x)$ gegebener Anfangswert und $\varphi(t)$ eine gegebene Funktion.

Wir lösen das Problem numerisch wie folgt :
Wir nehmen an, es gäbe keine Produktion von $u(x,t)$ zu den Zeiten $t > 0$, d.h. $f = 0$. Seien $c > 0$, $N,J \in \mathbb{N}$, $T > 0$ und $u_j^n \approx u(x_j,t_n)$. Wir definieren:

$$x_j = a + j\delta x, \quad \delta x = \frac{b-a}{J}, \quad j = 0,1,\ldots,J$$

und

$$t_n = n\delta t, \quad \delta t = \frac{T}{N}, \quad n = 0,1,\ldots,N.$$

Wir approximieren nun die partiellen Ableitungen ∂_t, ∂_x mit Hilfe vom Differenzenquotienten und erhalten:

$$u_j^{n+1} = u_j^n - \frac{c\delta t}{\delta x}(u_j^n - u_{j-1}^n), \quad j = 0,1,\ldots,J; \quad n = 0,1,\ldots,N.$$

2.1.1 Stabilität

Die hinreichende Bedingung für die Stabilität, die CFL-Bedingung (Courant-Friedrichs-Lewy) der Konvektionsgleichung und somit auch für die Richtigkeit der numerischen Lösung lautet:

$$\frac{c\delta t}{\delta x} \leq 1.$$

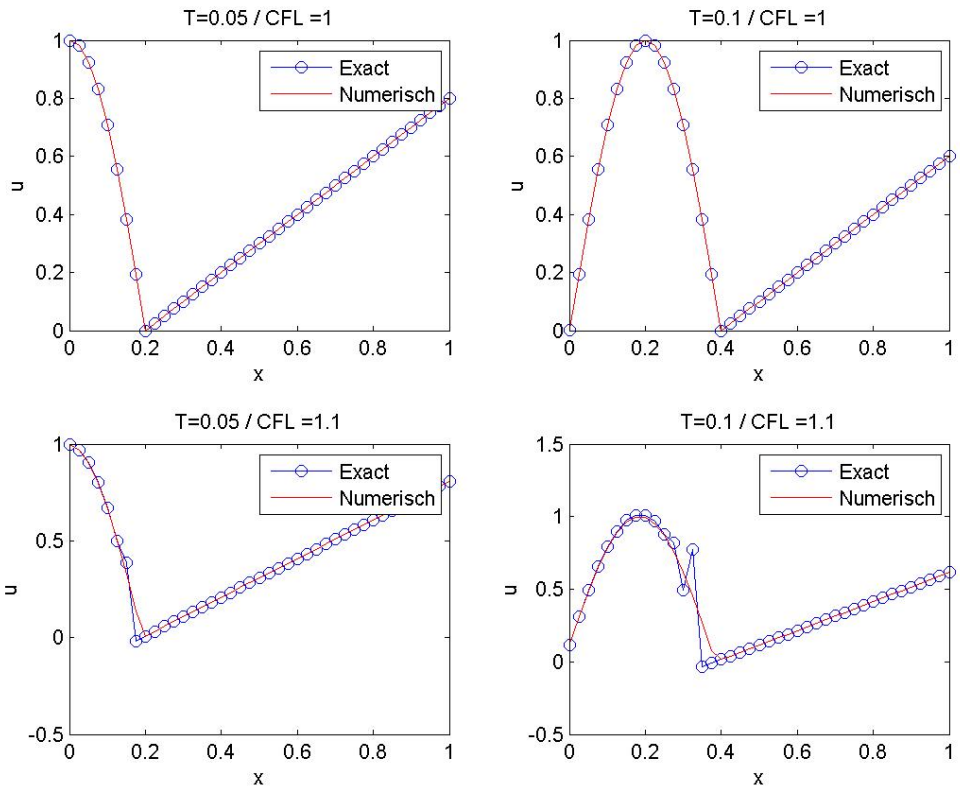


Abbildung 3: Konvektionsgleichung

Wir sehen in der Abblidung 3, dass wenn die CFL-Bedingung eingehalten wird, die numerische Lösung fast übereinstimmt. Wenn dies jedoch nicht der Fall

ist, wird die numerische Lösung ungenauer, je grösser T gewählt wird. Es kann sogar explodieren.

Seminar "12 Probleme aus der Numerik"
Projekt 3: Polynomial Approximation

Elisa Tekin und Zoe Zarri

Herbstsemester 2011

Problemstellung:[1] Wir wollen eine Funktion $f(x)$, die durch die $n+1$ verschiedene Punkte (x_i, y_i), $i = 0, ..., n$ geht, durch ein eindeutiges Polynom p in $\mathbb{P}_n$ approximieren. Es gibt drei Möglichkeiten, um $f(x)$ zu approximieren: *Interpolation*, *beste Approximation* und *stückweise polynomiale Approximation*.

Inhaltsverzeichnis

1 Interpolation

In diese erste Teil $f : [a, b] \to \mathbb{R}$ ist eine stetige Funktion, x_i $(i = 0, ..., n)$ sind $n+1$ verschiedene Punkte in $[a, b]$. Wir suchen ein Polynom $p(x)$ so, dass $p(x_i) = f(x_i)$ für $i = 0, ..., n$ gilt.

1.1 Polynominterpolation nach Lagrange

Gegeben seien $n+1$ verschiedene Punkte $(x_i, y_i), i = 0, ..., n$ mit $y_i = f(x_i)$. Die Lagrange-Polynome von Grad n sind so definiert:

$$L_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j} = \frac{(x - x_0) \cdot \ldots \cdot (x - x_{i-1}) \cdot (x - x_{i+1}) \cdot \ldots \cdot (x - x_n)}{(x_i - x_0) \cdot \ldots \cdot (x_i - x_{i-1}) \cdot (x_i - x_{i+1}) \cdot \ldots \cdot (x_i - x_n)}.$$

Das Interpolationspolynom $p(x)$ ist durch

$$p(x) = \sum_{i=0}^{n} y_i \cdot L_i(x)$$

[1] Kapitel 3 im Buch "An Introduction to Scientific Computing, Twelve Computational Projects Solved with MATLAB" von I. Danaila, P. Joly, S. M. Kaber, M. Postel.

definiert.

Bemerkung: Die Polynominterpolation nach Lagrange ist ungünstig, wenn man mehr Stützpunkte (x_i, y_i) nimmt.

1.2 Polynominterpolation von Newton

Zuerst definieren wir die dividierten Differenzen. Seien (x_i, y_i) (mit $i = 0, ..., k$) $k+1$ verschiedene Punkte. Die dividierten Differenzen sind rekursiv wie folgt definiert:

$$\begin{aligned} y[x_i] &= y_i\,, \\ \delta y[x_i, x_j] &= \frac{y[x_j] - y[x_i]}{x_j - x_i}\,, \\ \delta^k y[x_{i_0}, x_{i_1}, ..., x_{i_k}] &= \frac{\delta^{k-1} y[x_{i_1}, ..., x_{i_k}] - \delta^{k-1} y[x_{i_0}, ..., x_{i_{k-1}}]}{x_{i_k} - x_{i_0}}\,. \end{aligned}$$

Mit den dividierten Differenzen erhalten wir das Interpolationspolynom von Newton:

$$\begin{aligned} p(x) = y_0 &+ (x - x_0) \cdot \delta y[x_0, x_1] + (x - x_0) \cdot (x - x_1) \cdot \delta^2 y[x_0, x_1, x_2] + \dots \\ &+ (x - x_0) \cdot \ldots \cdot (x - x_{n-1}) \cdot \delta^n y[x_0, ..., x_n]\,. \end{aligned}$$

1.3 Lagrange Interpolationsfehler

Sei $f : [a, b] \to \mathbb{R}$ eine $(n+1)$-mal stetig differenzierbare Funktion und $p(x)$ das Interpolationspolynom von Grad n. Für jedes $x \in [a, b]$ existiert ein $\xi_x \in [a, b]$ so, dass

$$f(x) - p(x) = \frac{1}{(n+1)!}\, \omega(x)\, f^{n+1}(\xi_x)\,,$$

wobei $\omega(x) = (x - x_0) \cdot \ldots \cdot (x - x_n)$. Dann gilt

$$|f(x) - p(x)| \le \frac{1}{(n+1)!} \sup_{x \in [a,b]} |\omega(x)| \sup_{\xi_x \in [a,b]} |f^{(n+1)}(\xi_x)|$$

für alle $x \in [a, b]$.

Um $\sup_{x \in [a,b]} |\omega(x)|$ zu minimieren, ist es ein guter Wahl, die Chebyshev Punkte zu nehmen. Die Chebyshev Punkte sind durch

$$x_i = \frac{a+b}{2} + \frac{b-a}{2}\, \cos(\theta_i)$$

definiert, wobei $\theta_i = \dfrac{\pi}{2n} + i\,\dfrac{\pi}{n}$, $i = 0, ..., n-1$.
In Abbildung 1 sieht man, dass das Interpolationspolynom mit äquidistanten Punkten oszilliert und dieses mit den Chebyshev Punkten genauer ist.

2 Beste Polynom Approximation

Sei χ ein lineares Raum. Zunächst suchen wir ein Polynom $p(x)$, das möglichst nah zur Funktion $f(x)$ bezüglich der Norm $\|\cdot\|_\chi$ ist, d.h.

$$|f(x) - p(x)| = \inf_{q \in \mathbb{P}_n} \|f(x) - q(x)\|_\chi\,, \quad p(x) \in \mathbb{P}_n\,.$$

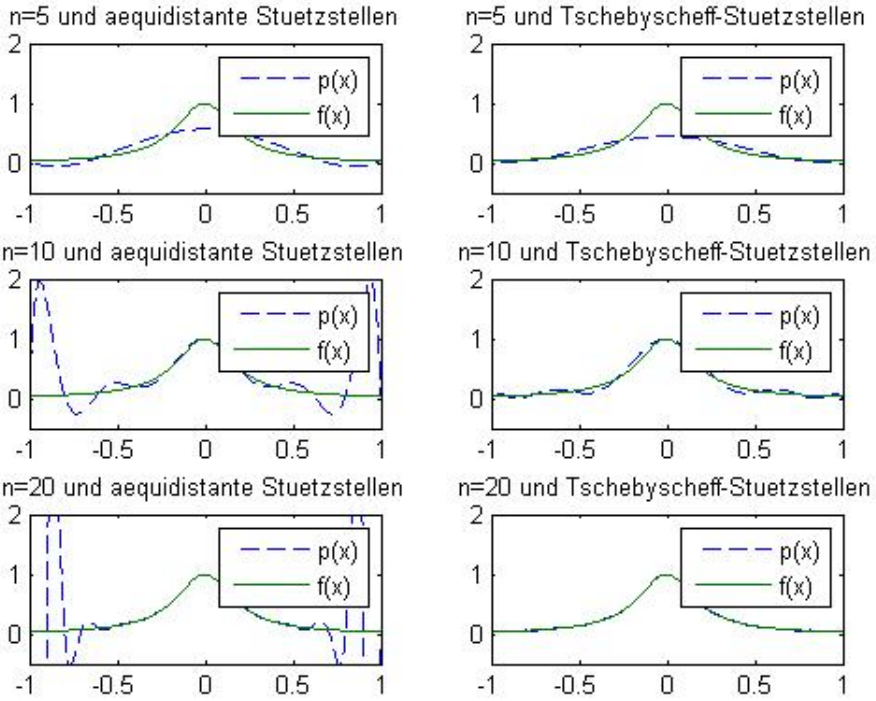


Abbildung 1: Newton vs Chebyshev für die Funktion $f(x) = \frac{1}{1+25x^2}$

Dann nennt man das Polynom $p(x)$ die beste Polynom Approximation von $f(x)$ in $\mathbb{P}_n$.

Wir betrachten zwei verschiedene Räume χ:

- *Beste Approximation bezüglich Supremumsnorm*: man nimmt $I = [a, b]$ und dann $\chi = C(I)$, der Raum aller stetigen Funktionen mit Supremumsnorm.
- *Beste Hilbert Approximation*: man nimmt $I =]a, b[$ und $\chi = L^2(I)$, definiert als den Raum messbarer Funktionen in $]a, b[= I$, so dass die Integrale $\int_a^b |f(x)|^2 \, dx$ endlich sind. Die L^2-Norm definiert man durch

$$\|f\|_{L^2(I)} = \sqrt{\langle f, f \rangle} := \sqrt{\int_a^b |f|^2 \, dt} \, .$$

2.1 Beste Approximation bezüglich Supremumsnorm

Sei φ eine stetige Funktion. Man sagt φ ist gleichmässig oszillierend auf $n + 1$ Punkte $x_0 < x_1 < ... < x_n$ im Intervall $[a, b]$, wenn φ wechselweise die Werte $\pm\|\varphi\|_\infty$ in den gegebenen Punkten nimmt.

Satz 1. *Sei $f(x)$ eine stetige Funktion im Intervall $I = [a, b]$. Die beste Polynomapproximation bezüglich Supremumsnorm $p(x) \in \mathbb{P}_n$ von $f(x)$ ist das einzige Polynom in $\mathbb{P}_n$, für welchen die Funktion $h(x) = f(x) - p(x)$ in (mindestens) $n + 2$ verschiedene Stellen in I gleichmässig oszillierend ist.*

Um die beste Approximation bezüglich Supremumsnorm einer Funktion $f(x)$ zu bestimmen, genügt es ein Polynom $p(x) \in \mathbb{P}_n$ und $n+2$ Punkte zu finden, so dass $f(x)-p(x)$ in diesen Punkten gleichmässig oszilliert, siehe das Algorithmus von Remez.

Remez Algorithmus

- Wähle $n + 2$ verschiedene Punkte $x_0^0 < x_1^0 < ... < x_{n+1}^0$.
- k-ter Schritt: seien die $n + 2$ Punkte $x_0^k < x_1^k < ... < x_{n+1}^k$ gegeben. Berechne ein Polynom $p_k(x) \in \mathbb{P}_n$ so, dass

$$f(x_i^k) - p_k(x_i^k) = (-1)^i (f(x_0^k) - p_k(x_0^k)) \quad i = 1, \ldots, n + 1\,.$$

  * Falls $\|f(x) - p_k(x)\|_\infty = |f(x_i^k) - p_k(x_i^k)|$ für $i = 0, \ldots, n + 1$ das Algorithmus endet, da die Funktion $h(x) = f(x)-p_k(x)$ gleichmässig in diesen Punkten oszilliert.
  * Sonst, es existiert $y \in [a, b]$ so, dass

$$\|f(x)-p_k(x)\|_\infty = |f(y)-p_k(y)| > |f(x_i^k)-p_k(x_i^k)| \quad i = 0, \ldots, n+1\,.$$

    Man definiert die neue Punkte $x_0^{k+1} < x_1^{k+1} < ... < x_{n+1}^{k+1}$ beim Ersetzen einem der Punkte x_i^k mit y so, dass

$$(f(x_j^{k+1}) - p_k(x_j^{k+1})) \cdot (f(x_{j-1}^{k+1}) - p_k(x_{j-1}^{k+1})) \leq 0 \quad j = 1, \ldots, n + 1\,.$$

In Abbildung 2 findet man ein Beispiel für die beste Approximation bezüglich Supremumsnorm von der Funktion $f(x) = \sin(2\pi \cos(\pi x))$.

2.2 Beste Hilbert Approximation

Hier betrachten wir das Intervall $I =]-1, 1[$, da jedes beliebiges Intervall $]a, b[$ auf I geschickt (mit einer einfachen Transformation) werden kann.

Basis von $L^2(I)$
Die Legendre Polynome $L_n(x)$ bilden eine orthogonale Basis von $L^2(I)$. Die Legendre Polynome sind durch folgende rekursive Beziehung definiert

$$(n + 1)\, L_{n+1}(x) = (2n + 1)\, L_n(x) - n\, L_{n-1}(x)$$

für $n \geq 1$ und $L_0(x) = 1$, $L_1(x) = x$. Die so definierten Polynome sind orthogonal, d.h.

$$\langle L_n(x), L_m(x)\rangle = \begin{cases} 0 & \text{für } n \neq m\,, \\ \dfrac{1}{n + \frac{1}{2}} & \text{für } n = m\,. \end{cases}$$

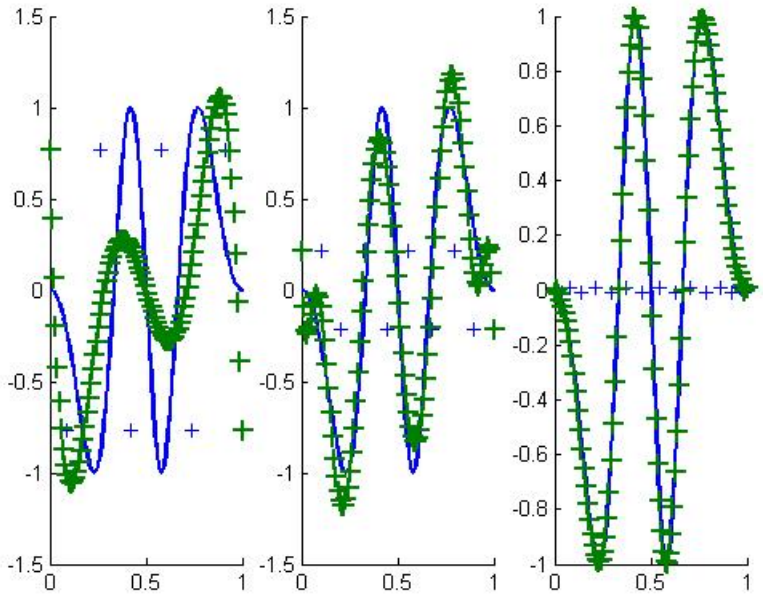


Abbildung 2: Die beste Approximation bezüglich Supremumsnorm in $\mathbb{P}_n$ für $f(x) = \sin(2\pi\cos(\pi x))$ berechnet mit Hilfe vom Remez Algorithmus: $n = 5$ (links), $n = 10$ (mitte) und $n = 15$ (rechts).

Satz 2. *Sei $f(x) \in L^2(I)$ und $n \in \mathbb{N}$.*

(i) Es existieren reelle Zahlen $\widehat{f}_k$ so, dass $f(x) = \sum_{k=0}^{\infty} \widehat{f}_k \cdot L_k(x)$.

(ii) Es existiert ein eindeutiges Polynom $\pi_n f \in \mathbb{P}_n$ so, dass

$$\|f(x) - \pi_n f\|_{L^2(I)} = \inf_{q \in \mathbb{P}_n} \|f(x) - q(x)\|_{L^2(I)}\,,$$

d.h. $\pi_n f$ ist die beste Hilbert Approximation von $f(x)$. Auch gilt

$$\langle f(x) - \pi_n f(x), p(x)\rangle = 0 \quad \textit{für alle } p(x) \in \mathbb{P}_n\,.$$

Bemerkung: Die reelle Zahlen $\widehat{f}_k$ heissen Fourier-Legendre Koeffizienten der Funktion $f(x)$. Mit Hilfe der Orthogonalität von Legendre Polynome bestimmen wir

$$\widehat{f}_k = \frac{\langle f(x), L_k(x)\rangle}{\|L_k(x)\|^2} = \left(k + \frac{1}{2}\right)\int_{-1}^{1} f(t)\cdot L_k(t)\,dt\,.$$

Dann ist $\pi_n f$ die Legendre Reihe von $f(x)$ von Ordnung n, d.h.

$$\pi_n f(x) = \sum_{k=0}^{n} \widehat{f}_k \cdot L_k(x)\,.$$

Die Berechnung der beste Hilbert Approximation einer Funktion besteht meistens aus der Berechnung von ihrer Fourier-Legendre Koeffizienten.

Solving an Advection-Diffusion Equation by a Finite Element Method

Sergio Mouzo

16. Dezember 2011

Zusammenfassung

In diesem Projekt betrachten wir das Advektion-Diffusionsproblem. Wir lösen das Problem mit Hilfe der Finiten Elemente Methode numerisch. (Quelle: siehe Fussnote[1])

Inhaltsverzeichnis

1 Einführung

In diesem Projekt suchen wir eine numerische Approximation der Lösung $u : [0,1] \to \mathbb{R}$ des *Advektion-Diffusionsproblems*

$$\begin{aligned} -\varepsilon \cdot u''(x) + \lambda \cdot u'(x) = f(x) \quad x \in]0,1[\,, \\ u(0) = 0\,,\ u(1) = 0\,. \end{aligned} \tag{1}$$

Die Funktion f und die Konstanten ε und λ sind so gegeben, dass eine einzige Lösung dieses Problems existiert. Unser Ziel ist es, die Lösung von (1) mit einer stetigen, stückweisen polynomialen Funktion zu approximieren.

Bemerkung: Advektion bedeutet grundsätzlich der Transport von Stoffen mit der Strömung eines Mediums. Diffusion ist ein physikalischer Prozess, der zu einer gleichmässigen Verteilung von Teilchen führt, d.h. Teile aus Bereiche hoher Konzentration bewegen sich in Bereiche geringer Konzentration.

Das Advektion-Diffusionsproblem (1) modelliert also verschiedene Phänomene, wie z.B. die Konzentration eines chemischen Stoffes in einem Fluid. In diesem Fall bezeichnet $\lambda \in \mathbb{R}$ die Geschwindigkeit des Fluids, $\varepsilon \in \mathbb{R}_{>0}$ den Diffusionskoeffizient des chemischen Stoffes, $f(x)$ die Produktion und das Verschwinden des chemischen Stoffes und $\theta = \dfrac{\lambda}{\varepsilon}$ misst die Wichtigkeit der Advektion im Vergleich zur Diffusion.

Die exakte Lösung vom Problem (1)

Für konstante Funktionen $f(x) \equiv f$ lautet die exakte Lösung von (1)

$$u(x) = \frac{f}{\lambda} \cdot \left(x - \frac{e^{\theta \cdot x} - 1}{e^{\theta} - 1} \right) .$$

[1] I. Danaila, P. Joly, S. M. Kaber, M. Postel, *An introduction to scientific computing: Twelve computational projects solved with MATLAB*, 2007

Hieraus wird ersichtlich (siehe Abbildung 1), dass falls $|\theta|$ gross wird, so springt die Funktion in einem immer kleineren Intervall von ihrem Maximum $\frac{f}{\lambda}$ auf 0. In diesem sogenannten "boundary layer" ist die Funktion schwer approximierbar, da die Steigung im Betrag sehr gross wird.

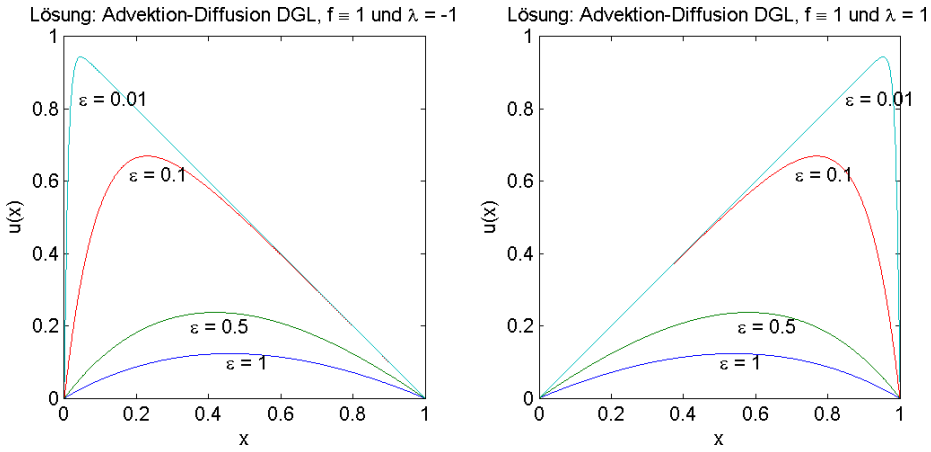


Abbildung 1: Lösung der Advektion-Diffusion Differentialgleichung zu verschiedenen ε

2 Schwache Formulierung

Die Finite Elemente (FE) Methode basiert auf der Berechnung der Lösung einer schwachen Formulierung von (1) und nicht auf der direkten Diskretisierung der Gleichung (1) z.B. durch eine Finite Differenzenmethode. Die schwache Formulierung von (1) lautet:

Finde $u \in V$ so, dass

$$a(u,v) = \int_0^1 f(x) \cdot v(x)\, dx \quad \text{für alle } v \in V\,, \tag{2}$$

mit

$$V := H_0^1(0,1) = \left\{ v : [0,1] \to \mathbb{R} \;:\; \int_0^1 |v|^2 \, dx\,, \int_0^1 |v'|^2 \, dx < \infty\,,\; v(0) = v(1) = 0 \right\}$$

und mit der Bilinearform

$$a(u,v) = \varepsilon \cdot \int_0^1 u'(x) \cdot v'(x)\, dx + \lambda \cdot \int_0^1 u'(x) \cdot v(x)\, dx\,.$$

Eine Lösung u des Randwertproblems (1) ist auch eine Lösung des schwachen Problems (2).

3 Finite Elemente Approximation

Für $n \in \mathbb{N}_{>0}$ dividiere das Intervall $[0,1]$ in $n+1$ Teilintervalle $I_i\,(i = 0, \ldots, n)$. Für $l \in \mathbb{N}_{>0}$ bezeichne $\mathbb{P}_l(I_i)$ die Schar von Polynomen von Grad $\leq l$ auf I_i und $\mathcal{V}_l^h$ die Schar der auf $[0,1]$ definierten, stetigen Funktionen, dessen Einschränkung auf jedes Intervall I_i zu $\mathbb{P}_l(I_i)$ gehört. Nun folgt ein anschauliches Beispiel mit Polynomen von Grad 1 und ein weiteres mit Polynomen von Grad 2 (siehe Abbildung 2).

Bemerkung: Die Approximation mit Polynomen $v_h \in \mathcal{V}_1^h$, d.h. eine lineare Approximation, nennt man auch P1-Finite Elemente Methode. Die Approximation mit Polynomen $v_h \in \mathcal{V}_2^h$, d.h. eine quadratische Approximation, nennt man auch P2-Finite Elemente Methode.

Bei der FE Methode geht man so vor, dass man eine Approximation $u_h \in \mathcal{V}_l^h$ der Lösung u von (2) sucht. Die diskrete Funktion $u_h \in \mathcal{V}_l^h$ findet man als Lösung vom folgenden Problem:

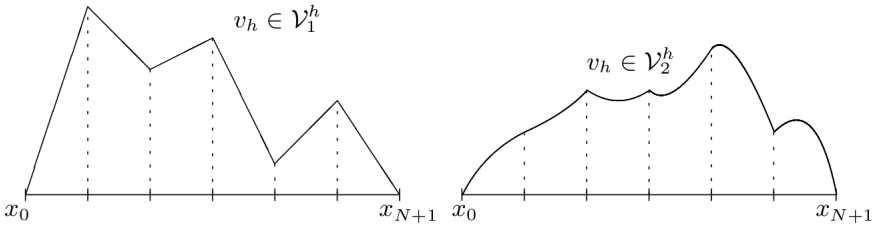


Abbildung 2: Eine diskrete Funktion $v_h \in \mathcal{V}_1^h$ (links) und eine diskrete Funktion $v_h \in \mathcal{V}_2^h$ (rechts).

Finde $u_h \in \mathcal{V}_l^h \subset H_0^1$ so, dass

$$a(u_h, v_h) = \int_0^1 f(x) \cdot v_h(x)\, dx \quad \text{für alle } v_h \in \mathcal{V}_l^h\,. \tag{3}$$

Die Probleme (1) und (2) sind kontinuierlich, wohingegen das Problem (3) diskret ist. Klar, denn $\mathcal{V}_l^h \subset V$ ist ein endlichdimensionaler Unterraum.

3.1 Lineare Finite Elemente

Für $n \in \mathbb{N}_{>0}$ definiere Punkte $x_k = k \cdot h\,(k = 0, \dots, n+1)$ und Intervalle $I_k =]x_k, x_{k+1}[$ mit Gittergrösse $h = \dfrac{1}{n+1}$. Wir definieren die "Hutfunktionen" $\varphi_{h,j} \in \mathcal{V}_1^h, j = 1, \dots, n$, so dass

$$\varphi_{h,j}(x_i) = \delta_{ji} = \begin{cases} 1 & j = i \\ 0 & j \neq i \end{cases}$$

für alle $j = 1, \dots, n$, (siehe Abbildung 3), folglich gilt

$$\begin{cases} \varphi_{h,k}(x) = \dfrac{x - x_{k-1}}{h}, & \varphi'_{h,k}(x) = \dfrac{1}{h} & , x \in I_{k-1} \\ \varphi_{h,k}(x) = \dfrac{x_{k+1} - x}{h}, & \varphi'_{h,k}(x) = -\dfrac{1}{h} & , x \in I_k \\ \varphi_{h,k}(x) = \varphi'_{h,k}(x) = 0 & & , \text{sonst.} \end{cases}$$

Lemma 1. *Die Hutfunktionen $(\varphi_{h,k})_{k=1}^n$ bilden eine Basis von $\mathcal{V}_1^h$.*

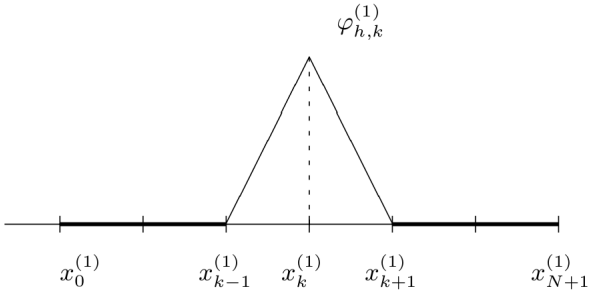


Abbildung 3: Hutfunktionen

Zunächst suchen wir eine lineare Approximation $u_h \in \mathcal{V}_1^h$ der Lösung von (2). Zuerst betrachten wir die Basisdarstellung von $u_h \in \mathcal{V}_1^h$

$$u_h = \sum_{k=1}^{n} \alpha_k \cdot \varphi_{h,k}(x)$$

mit $\alpha_k \in \mathbb{R}, k = 1, \dots, n$. Dann bekommen wir aus (3)

$$a\left(\sum_{k=1}^{n}\alpha_k\cdot\varphi_{h,k},v_h(x)\right)=\int_0^1 f(x)\cdot v_h(x)\,dx\quad\text{für alle } v_h\in\mathcal{V}_1^h\,. \tag{4}$$

Da es reicht nur mit den Basisfunktionen in (4) zu testen, haben wir

$$a\left(\sum_{k=1}^{n}\alpha_k\cdot\varphi_{h,k},\varphi_{h,j}(x)\right)=\sum_{k=1}^{n}a(\varphi_{h,k},\varphi_{h,j}(x))\cdot\alpha_k=\int_0^1 f(x)\cdot\varphi_{h,j}(x)\,dx\quad\text{für alle } j=1,\dots,n. \tag{5}$$

Das Problem (5) besitzt die folgende Matrixdarstellung

$$A_h\cdot\tilde{u}_h=b_h, \tag{6}$$

wobei $\tilde{u}_h=(u_h(x_1),\dots,u_h(x_n))=(\alpha_1,\dots,\alpha_n)\in\mathbb{R}^n$. Die Matrix $A_h\in\mathbb{R}^{n\times n}$ wird durch

$$A_h=\varepsilon\cdot B_h^{(1)}+\lambda\cdot C_h^{(1)}$$

definiert, wobei

$$(A_h)_{j,k}=a(\varphi_{h,k},\varphi_{h,j})=\varepsilon\cdot\int_0^1\varphi'_{h,k}\cdot\varphi'_{h,j}\,dx+\lambda\cdot\int_0^1\varphi'_{h,k}\cdot\varphi_{h,j}\,dx=\varepsilon\cdot(B_h^{(1)})_{j,k}+\lambda\cdot(C_h^{(1)})_{j,k}\quad j,k=1,\dots,n$$

und

$$B_h^{(1)}=\frac{1}{h}\begin{pmatrix}2&-1&0&\dots&\dots&0\\-1&2&-1&0&&\vdots\\0&\ddots&\ddots&\ddots&&\vdots\\\vdots&&\ddots&\ddots&\ddots&\vdots\\\vdots&&0&-1&2&-1\\0&\dots&\dots&0&-1&2\end{pmatrix},\quad C_h^{(1)}=\frac{1}{2}\begin{pmatrix}0&1&0&\dots&\dots&0\\-1&0&1&0&&\vdots\\0&\ddots&\ddots&\ddots&&\vdots\\\vdots&&\ddots&\ddots&\ddots&\vdots\\\vdots&&0&-1&0&1\\0&\dots&\dots&0&-1&0\end{pmatrix}$$

Wenn die rechte Seite $f(x)$ eine konstante Funktion ist, kann man den Vektor $(b_h)_j\in\mathbb{R}^n$ explizit berechnen. In diesem Fall bekommt man

$$(b_h)_j=\int_0^1 f(x)\cdot\varphi_{h,j}(x)\,dx=f\cdot\int_0^1\varphi_{h,j}(x)\,dx=f\cdot h\quad\text{für alle } j=1,\dots,n.$$

Wäre f von x abhängig, müssten wir eine Quadraturformel (z.B. Trapez- oder Simpsonregel) benützen um das Integral auszurechnen.

Die Matrix A_h ist symmetrisch und positiv definit. Somit ist A_h invertierbar und aus dem folgt wiederum, dass das System (4) eine einzige Lösung hat, welches berechnet werden kann, indem man das lineare System (6) löst.

Anschliessend zwei Plots (siehe Abbildung 4 und 5) um zu veranschaulichen, wie die P1 FE Methode die Lösung approximiert. Dabei halten wir das λ und das f konstant und variieren das ε und das n. Ein grösseres n bedeutet natürlich eine bessere Approximation. Falls aber $|\theta|$ zu gross wird (kleines ε), dann wird der sogenannte "boundary layer" so klein, dass die Approximation trotz vielen Stützstellen anfängt zu oszillieren.

3.2 Quadratische Finite Elemente

Für $n\in\mathbb{N}_{>0}$ setze $h=\dfrac{1}{n+1}$ und definiere die Punkte $x_k=\dfrac{k\cdot h}{2}$ für $k=0,\dots,2\cdot(n+1)$ und Intervalle $I_k=]x_{2k},x_{2k+2}[$. Jeder zweite Punkt ist gleich wie in P1 FE Methode und die Intervalle sind die von

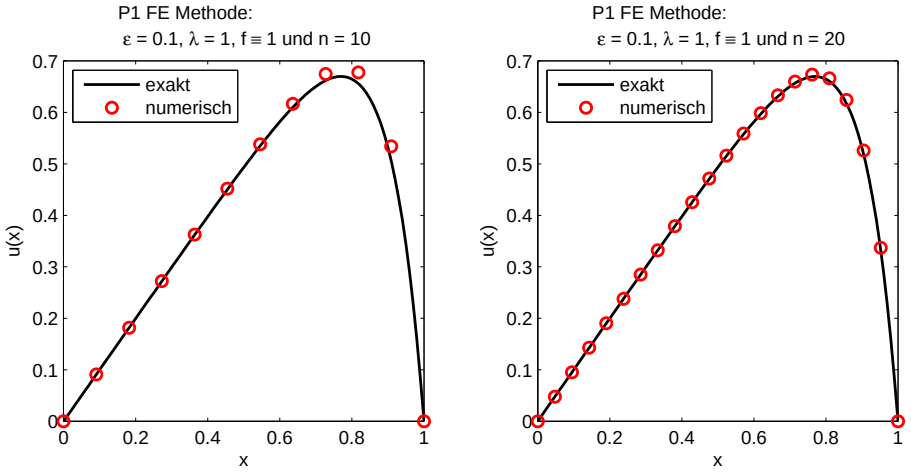


Abbildung 4: Approximation des Problems mit linearen FE und $\varepsilon = 0.1$. $n = 10$ (links), $n = 20$ (rechts)

vorhin. Wir behalten also die gleiche Anzahl von Intervallen bei und fügen einfach den mittleren Punkt hinzu.

Wir suchen also eine Approximation $u_h \in \mathcal{V}_l^h$ der Funktion u, eine Lösung des Problems (3) mit $l = 2$. Wie bei der linearen FE Methode, beginnen wir auch hier mit der Konstruktion einer Basis von $\mathcal{V}_2^h$ (siehe Abbildung 6). Auf jedem Intervall I_k definieren wir drei quadratische Lagrange-Polynome, die mit den Punkten x_{2k}, x_{2k+1} und x_{2k+2} verbunden sind:

$$\begin{cases} \psi_{h,k}^{(-)}(x) = 2 \cdot (x - x_{2k+1}) \cdot (x - x_{2k+2}) \cdot \dfrac{1}{h^2}\,, \\ \psi_{h,k}^{(0)}(x) = -4 \cdot (x - x_{2k}) \cdot (x - x_{2k+2}) \cdot \dfrac{1}{h^2}\,, \\ \psi_{h,k}^{(+)}(x) = 2 \cdot (x - x_{2k}) \cdot (x - x_{2k+1}) \cdot \dfrac{1}{h^2}\,. \end{cases}$$

Zu jedem inneren Knoten $x_l\,,\ l = 1, \ldots, 2n+1$ assoziieren wir die Funktion $\varphi_{h,l} \in \mathcal{V}_2^h$ definiert als

$$\varphi_{h,2k+1}(x) = \begin{cases} \varphi_{h,k}^0(x) & , x \in I_k \quad (k = 0, \ldots, n)\,, \\ 0 & , \text{sonst}\,, \end{cases}$$

$$\varphi_{h,2k}(x) = \begin{cases} \varphi_{h,k}^{(-)}(x) & , x \in I_k\,, \\ \varphi_{h,k-1}^{(+)}(x) & , x \in I_{k-1} \quad (k = 1, \ldots, n)\,, \\ 0 & , \text{sonst.} \end{cases}$$

Lemma 2. *Die Funktionen $(\varphi_{h,k})_{k=1}^{2n+1}$ bilden eine Basis von $\mathcal{V}_2^h$.*

Die Vorgehensweise zum Lösen des Problems (3) mit quadratischen Finiten Elementen ist analog zu der beschriebenen im Abschnitt 3.1. Man betrachtet wieder die Basisdarstellung von $u_h \in \mathcal{V}_2^h$ und testet das Problem (3) mit den Basisfunktionen $\varphi_{h,k}, k = 1, \ldots, 2n+1$. Dann ist das Problem (3) wieder zu einem linearen Gleichungssystem äquivalent.

An- und abschliessend drei Plots zur P2 FE Methode (siehe Abbildung 7, 8 und 9). Wir bemerken, dass die P2 FE Methode die Lösung wesentlich besser approximiert als die P1 FE Methode, da wir bei gleichem n auch mit doppelt sovielen Stützstellen arbeiten. Weiter stellen wir aber fest, dass für das "boundary layer"-Problem auch die P2 FE Methode (trotz $n = 20$) nicht optimal ist. In der zu Beginn zitierten Quelle, auf welche sich diese Arbeit stützt, wird in einem weiteren Unterkapitel eine Stabilisationsmethode erläutert, mit der man letzteres Problem vermeiden kann.

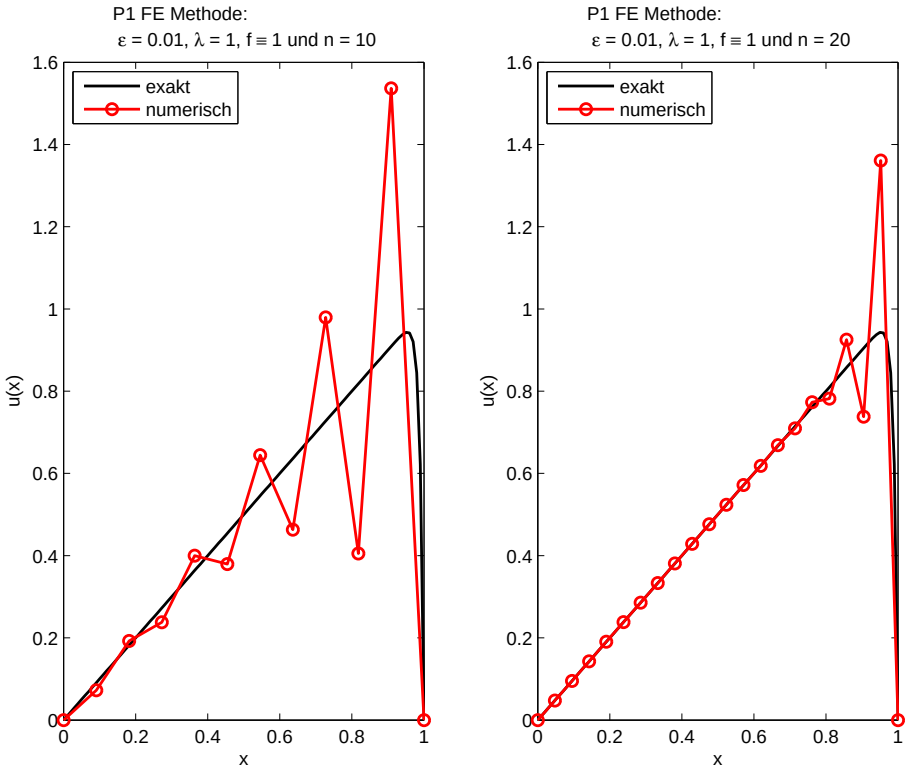


Abbildung 5: Approximation des Advektion-Diffusion Problems mit linearen FE und $\varepsilon = 0.01$

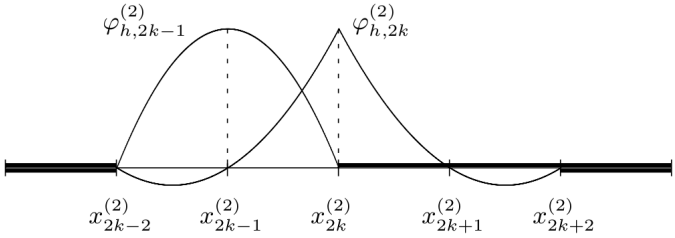


Abbildung 6: Basis von $\mathcal{V}_2^h$

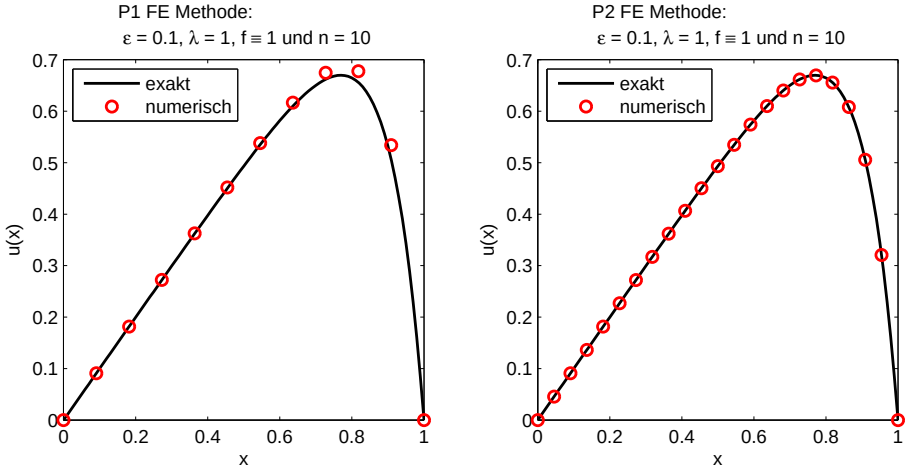


Abbildung 7: Für grosses ε und $n = 10$: P1 FE Methode und P2 FE Methode im Vergleich

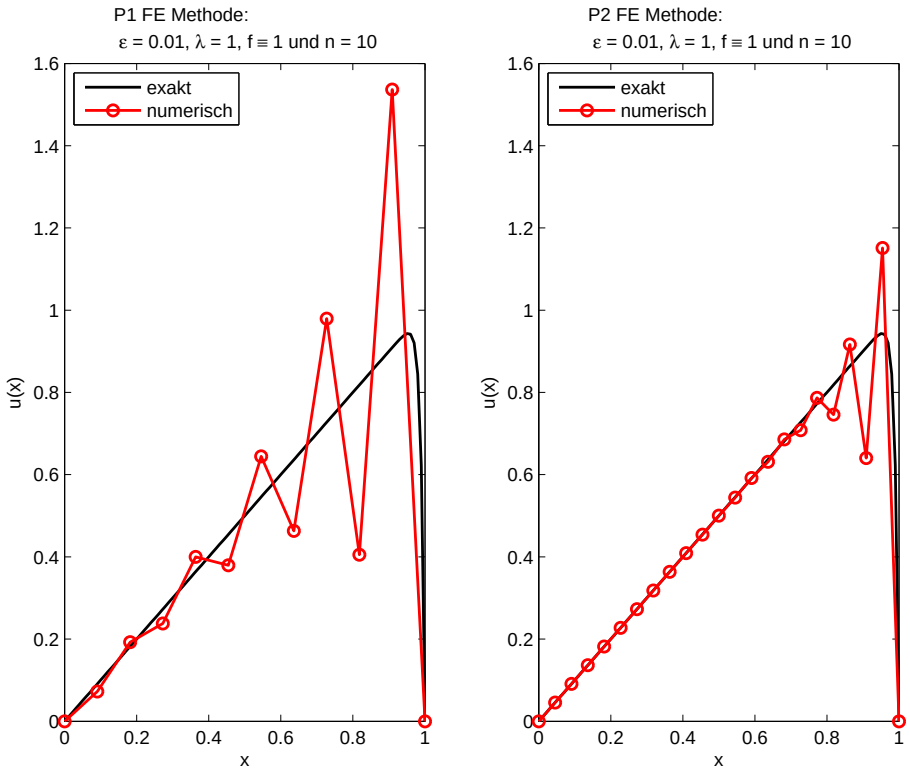


Abbildung 8: Für kleines ε und $n = 10$: P1 FE Methode und P2 FE Methode im Vergleich

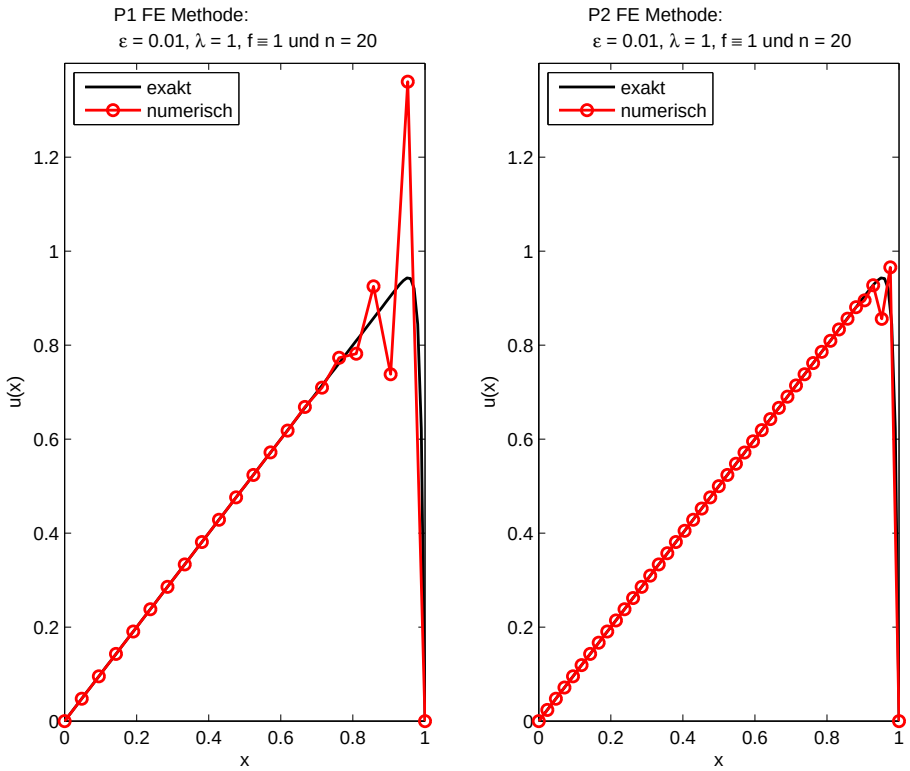


Abbildung 9: Für kleines ε und $n = 20$: P1 FE Methode und P2 FE Methode im Vergleich

Seminar: Geometric Design
Bézier Curve and Surface

Jet Tang

19. November 2011

Zusammenfassung

In der grafischen Modellierung von Objekten versuchen die Leute Objekte so einfach wie möglich mit mathematischen Elementen zu beschreiben. Anhand der gegebenen Elementen werden die Objekte mittels einfache mathematische Funktionen grafisch dargestellt. In diesem Seminar wird eine mathematische Beschreibung von Objekte und die grafische Umsetzung durch gegebenen Elementen vorgestellt.

1 Einleitung

Heutzutage sieht man in vielen Gebieten grafische Modellierungen von Objekten wie Computergrafiken in Spielen oder in Filmen, Geometrische Modellierungen, Chemische Moleküle, Biologische Substanzen, Bauteile und Baupläne von Geräten, etc. Die Objekte werden durch sogenannte Basiselemente wie Punkte, Kurve, Fläche und Volumen beschrieben. Für die Umsetzung benutzt man elementare mathematische Funktionen wie Polynome und rationale Funktionen.

Das Problem der Objektmodellierungen kam ursprünglich aus den Industrien in den 60er Jahren. Damals beschäftigten sich viele Leute wie Pierre Bézier (1910) und Paul de Casteljau (1930) um dieses Problem. Pierre Bézier war als Ingenieur in Renault tätig und bekam von seinem Chef die Aufgabe, Karosserieformen zu gestalten. Da beschrieb Bézier die Bézierkurve, die später für Computergrafiken als Basiselement verwendet wird. [1] Paul de Casteljau arbeitete als Physiker für Citröen und beschäftigte sich um die Modellierung von Kurven und Flächen. Und später entwickelte er den De Casteljau Algorithmus um die Bézierkurve numerisch zu berechnen. [2]

Als erstes wird hier die Bézierkurve vorgestellt. Dann wird erklärt wie man zwei Bézierkurven "glatt zusammen verbinden" kann und anschließend die Berechnung mit dem De Casteljau Algorithmus. Zum Schluss werden die Bézierfläche und der Boor-Coox Algorithmus betrachtet.

2 Bézierkurve

2.1 Definition und Bernsteinpolynom

Die *Bézierkurve* $\mathcal{B}_m$ beschreibt eine Kurve durch gegebenen Kontrollpunkte $P_0, \ldots, P_m \in \mathbb{R}^n$ mit $n \geq 2$ und $m \geq 0$. Die Kurve ist definiert durch das Polynom $P(t)$ mit

$$P(t) = \sum_{k=0}^{m} B_k^m(t) P_k,$$

wobei $t \in [0, 1]$ und

$$B_k^m(t) = \binom{m}{k} t^k (1-t)^{m-k}$$

die Gewichtung von P_k ist. $B_k^m(t)$ heisst auch *Bernsteinpolynom* vom Grad m und hat folgenden Eigenschaften:

[1] `http://de.wikipedia.org/wiki/Pierre_Bézier`
[2] `http://de.wikipedia.org/wiki/Paul_de_Casteljau`

(i) $B_k^m(t) \in [0,1]$ für alle $t \in [0,1]$, $0 \leq k \leq m$

(ii) $\sum_{k=0}^{m} B_k^m(t) = 1$ für alle $t \in [0,1]$

(iii) $B_k^m(0) = \delta_{0k} = \begin{cases} 0 & \text{für} \quad 0 < k \leq m \\ 1 & \text{für} \quad k = 0 \end{cases}$

(iv) $B_k^m(1) = \delta_{mk} = \begin{cases} 0 & \text{für} \quad 0 \leq k < m \\ 1 & \text{für} \quad k = m \end{cases}$.

Beweis:

(i)-(ii): Sei $t \in [0,1]$.

$$1 = 1^m = (t + 1 - t)^m = \sum_{k=0}^{m} \binom{m}{k} t^k (1-t)^{m-k} = \sum_{k=0}^{m} B_k^m(t).$$

Da $1 - t, t \in [0,1]$, gilt $(1-t)^{m-k}, t^k \in [0,1]$ für alle $< 0 \leq k \leq m$. Die Binomialkoeffizienten sind per Definition positiv. Daher gilt $B_k^m = \binom{m}{k} t^k (1-t)^{m-k} \geq 0$ für alle $< 0 \leq k \leq m$. Somit ergibt sich

$$B_k^m(t) \in [0,1], \text{ wegen } \sum_{k=0}^{m} B_k^m(t) = 1, \forall t \in [0,1].$$

(iii)-(iv): Da $0^k = 0$ für alle $k \geq 1$ und $0^k = 1$ für $k = 0$, gilt

$$\Rightarrow B_k^m(0) = \binom{m}{k} 0^k = \delta_{0k} = \begin{cases} 0 & \text{für} \quad 0 < k \leq m \\ 1 & \text{für} \quad k = 0 \end{cases}$$

und

$$\Rightarrow B_k^m(1) = \binom{m}{k} 0^{m-k} = \delta_{mk} = \begin{cases} 0 & \text{für} \quad 0 \leq k < m \\ 1 & \text{für} \quad k = m \end{cases} .$$

■

In der Abbildung 1 werden die Bernsteinpolynome B_k^m für $m = \{1,2,3,4\}$ und $0 \leq k \leq m$ dargestellt.

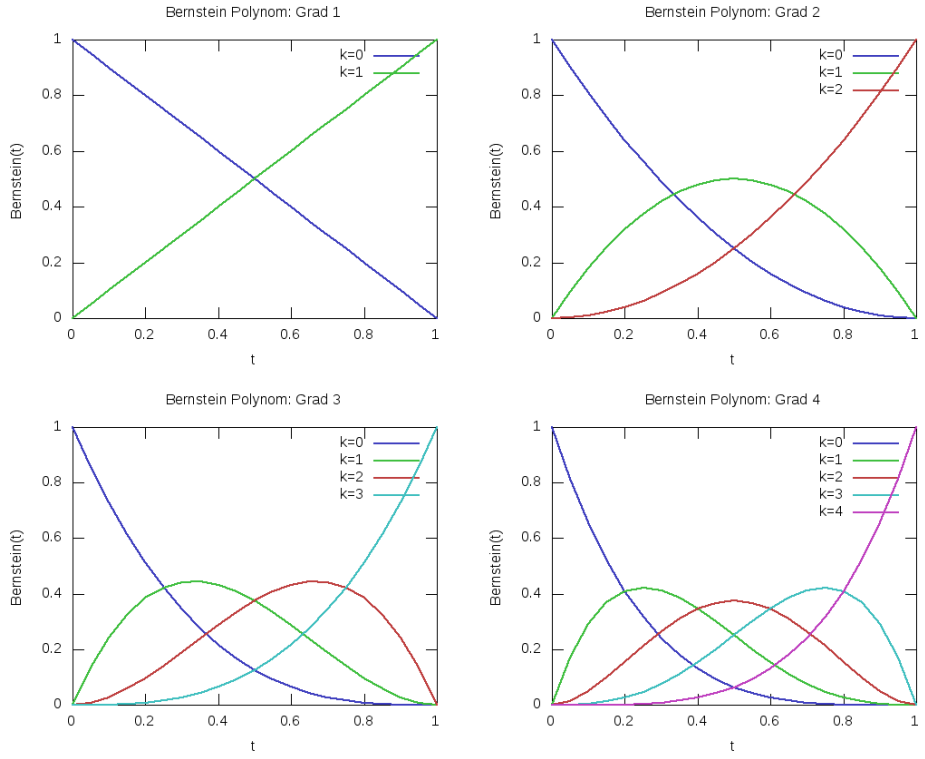
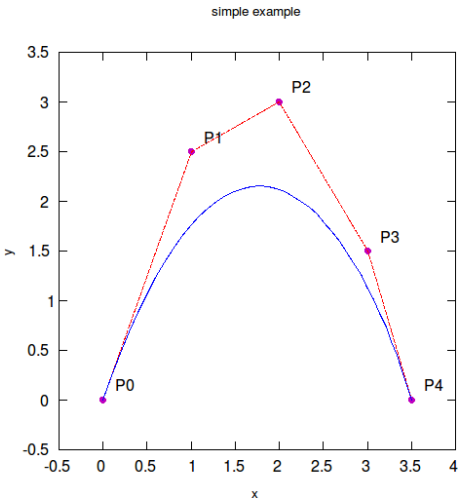


Abbildung 1: Bernsteinpolynome

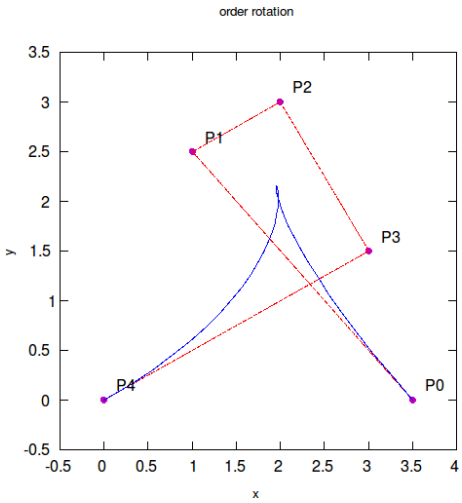
Speziell gelten:

$$P(0) = \sum_{k=0}^{m} B_k^m(0) P_k = \sum_{k=0}^{m} \delta_{0k} P_k = P_0 \quad \text{und} \quad P(1) = \sum_{k=0}^{m} B_k^m(1) P_k = \sum_{k=0}^{m} \delta_{mk} P_k = P_m.$$

Die Endpunkte P_0 und P_m liegen also auf $\mathcal{B}_m$. Man kann auch sagen, $P(t)$ ist ein Weg von P_0 nach P_m und $\mathcal{B}_m$ ist das Graph von diesem Weg. Im allgemeinen liegen nur P_0 und P_m auf $\mathcal{B}_m$.



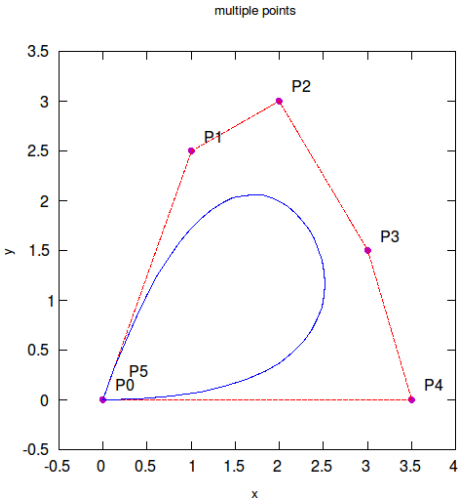
(a) Einfache Bézierkurve



(b) Einfache Bézierkurve mit vertauschten Endpunkten

Abbildung 2: Zwei Bézierkurven mit 5 Kontrollpunkte

Die Reihenfolge der Punkte spielt bei der Konstruktion von $\mathcal{B}_m$ eine Rolle; vertauscht man die Reihenfolge Punkte, dann verändert sich $\mathcal{B}_m$ entsprechend (vergleiche Abbildungen 2a) und 2b)). Zwei Bézierkurven mit verschiedener Anzahl von Kontrollpunkten können dasselbe Bild haben. Es ist erlaubt, Kontrollpunkte zu haben, die mehrfach vorkommen. Zum Beispiel für abgeschlossene Kurven ist es notwendig, dass der Anfangspunkt und der Endpunkt übereinstimmen (siehe Abbildung 3).



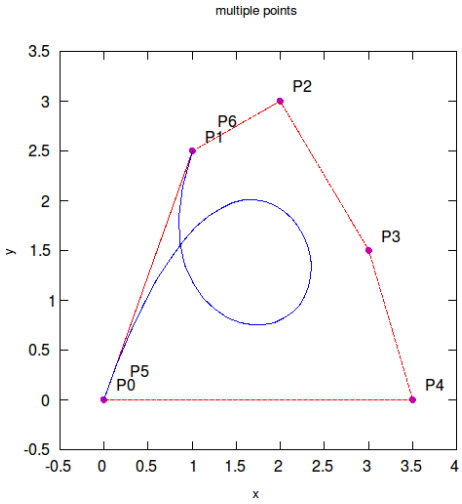


Abbildung 3: Abgeschlossene Kontrollpunkte

2.2 Verbindungsstelle

Die Bézierkurve wird durch ein Polynom beschrieben, wessen Grad die Anzahl der Kontrollpunkte entspricht, d.h. je mehr Punkte, desto größer ist der Grad. Um zu vermeiden Polynome von hohen Grad zu bekommen, versucht man das Polynom durch mehrere Polynome von niedrigen Grad zu ersetzen. Man berechnet die Kurve stückweise und verbindet sie dann. Sei $\mathcal{B}_m$ die Bézierkurve durch $P_0, \ldots, P_m$ und sei $\mathcal{B}_k$ die Fortsetzung von $\mathcal{B}_k$

durch $Q_0, \ldots, Q_k$. Sei $P(t)$ bzw. $Q(t)$ das Polynom von $\mathcal{B}_m$ bzw. von $\mathcal{B}_k$. Man verlangt für die Stetigkeit die $\mathcal{C}^0$-"Connection", d.h. der Endpunkt von der ersten Kurve muss mit dem Anfangspunkt von der zweiten Kurve übereinstimmen, also $P(1) = P_m = Q_0 = Q(0)$. Damit die Kurve "glatt" wird, setzt man zusätzlich entweder die $\mathcal{G}^1$ "Continuity Condition" oder die $\mathcal{C}^1$ "Continuity Condition" voraus. Die $\mathcal{G}^1$ "Continuity Condition" ist erfüllt, falls $\mathcal{B}_m$ tangential zum Liniensegment an $\overrightarrow{P_{m-1}P_m}$ liegt und $\mathcal{B}_k$ tangential zu dem Liniensegment an $\overrightarrow{Q_0Q_1}$ liegt. Dies ist nur dann möglich, falls die Punkte $P_{m-1}, P_m = Q_0$ und Q_1 auf einer Gerade liegen. Die $\mathcal{C}^1$ "Continuity Condition" ist erfüllt, falls die Steigung an den Endpunkt von $\mathcal{B}_m$ und die Steigung an den Anfangspunkt von $\mathcal{B}_k$ übereinstimmen, d.h. $P'(1) = Q'(0)$.

Sei $\tau(t)$ der Tangentialvektor von $P(t)$, d.h.

$$\tau(t) := \frac{\partial P}{\partial t}(t) = \sum_{k=0}^{m} \frac{\partial B_k^m}{\partial t}(t) P_k.$$

Betrachte $\frac{\partial B_k^m}{\partial t}(t) = \frac{\partial t^k (1-t)^{m-k}}{\partial t}$. Es gilt

$$\frac{\partial B_k^m}{\partial t}(t) = \begin{cases} -m(1-t)^{m-1} & \text{für} \quad k = 0 \\ m(1-mt)(1-t)^{m-2} & \text{für} \quad k = 1 \\ \binom{m}{k}(k-mt)t^{k-1}(1-t)^{m-k-1} & \text{für} \quad k \in \{2, \ldots, m-2\} \\ mt^{m-2}(m-1-mt) & \text{für} \quad k = m-1 \\ mt^{m-1} & \text{für} \quad k = m \end{cases}$$

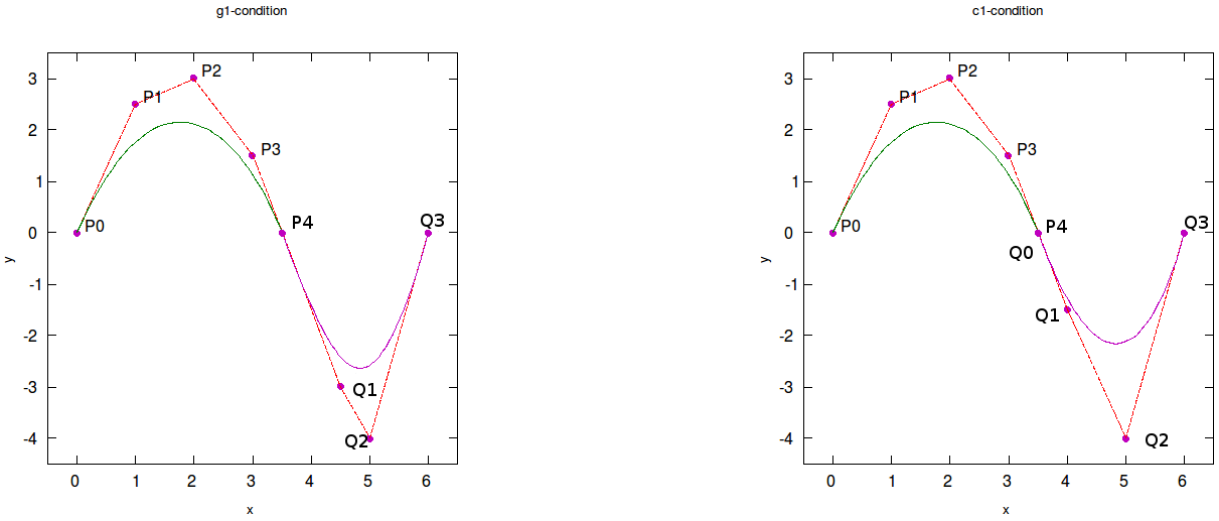
Es gilt $\tau(0) = P'(0) = -mP_0 + mP_1 = m\overrightarrow{P_0, P_1}$ und $\tau(1) = P'(1) = -mP_{m-1} + mP_m = m\overrightarrow{P_{m-1}, P_m}$. Somit ist die $\mathcal{C}^1$ "Continuity Condition" äquivalent zu

$$m\overrightarrow{P_{m-1}, P_m} = k\overrightarrow{Q_0, Q_1}.$$

Falls $k = m$ folgt

$$\overrightarrow{P_{m-1}, P_m} = \overrightarrow{Q_0, Q_1}.$$

In der Abbildung 4 wird eine aus zwei zusammengesetzten Bézierkurven dargestellt. In der Abbildung 4 a) wird die $\mathcal{G}^1$ "Continuity Condition" verlangt und in der Abbildung 4 b) wird die $\mathcal{C}^1$ "Continuity Condition" verlangt.



(a) $\mathcal{G}^1$ "Continuity Condition"

(b) $\mathcal{C}^1$ "Continuity Condition"

Abbildung 4: Verbindungen von zwei Bézierkurven

2.3 Casteljau Algorithmus

Das direkte Ausrechnen des Bernsteinpolynoms ist in der Praxis meistens ungenau und aufwendig. Die Idee ist, man benutzt die rekursive Darstellung von Bernsteinpolynome und berechnet dann die Beziérkurve rekursiv.
Lemma 1: Für alle $m \in \mathbb{N}$, $0 \leq k \leq m$ und $t \in [0, 1]$ gilt

$$B_{k+1}^{m+1}(t) = tB_k^m(t) + (1-t)B_{k+1}^m(t).$$

Beweis: Sei $m \in \mathbb{N}$ und $0 \leq k \leq m$. Für ein $t \in [0,1]$ definiere $B_{k+1}^{m+1} := B_{k+1}^{m+1}(t)$. Dann hat man

$$\begin{aligned}
B_{k+1}^{m+1} &= \binom{m+1}{k+1} t^{k+1}(1-t)^{(m+1)-(k+1)} \\
&= \left[\binom{m}{k} + \binom{m}{k+1}\right] t^{k+1}(1-t)^{m-k} \\
&= \binom{m}{k} t^{k+1}(1-t)^{m-k} + \binom{m}{k+1} t^{k+1}(1-t)^{m-k} \\
&= t\binom{m}{k} t^{k}(1-t)^{m-k} + (1-t)\binom{m}{k+1} t^{k+1}(1-t)^{m-(k+1)} \\
&= tB_k^m + (1-t)B_{k+1}^m.
\end{aligned}$$

■

Der *De Casteljau Algorithmus* berechnet $P(t)$ exakt und basiert auf dieser Idee. Der Algorithmus lautet:

```
%% De Casteljau Algorithmus
% Input: P0,...,Pm Punkte und t in [0,1]
% Output: P(t)
% Initialisierung
for j = 0 : m
  P(0,j) = Pj
% Konstruktion
for i = 1 : m
  for j = i : m
    P(i,j) = (1-t) * P(i-1,j-1) + t * P(i-1,j)
% Rückgabe
P(t) = P(m,m)
```

Für jede Berechung von $P(t)$ mit $t \in [0,1]$ mit dem Algorithmus wird $\mathcal{O}(m^2)$ Muliplikationen benötigt. Im Gegensatz dafür benötigt die Berechnung von den Binomialkoeffizienten $\binom{m}{k}$ $2 \cdot k$ Multiplikationen und die Berechnung von $t^k \cdot (1-t)^{m-k}$ $m+1$ Muliplikationen, d.h. für alle $k \in \{0, \ldots, m\}$ zusammen beträgt der Aufwand $\mathcal{O}(3m^2)$ (Multiplikationen).

Lemma 2: Für alle $0 \leq i \leq j \leq m$ gilt

$$P_j^i = \sum_{l=0}^{i} B_{i-l}^i P_{j-l}.$$

Beweis: (durch Induktion über i)

Induktionsanfang: Sei $i = 0$. Es gilt $\sum_{l=0}^{i} B_{i-l}^i P_{j-l} = B_0^0 P_j = P_j^0 \overset{\text{Def.}}{=} P_j$ für alle $0 \leq j \leq m$.

Induktionsschritt: Die Behauptung gilt für alle $\leq i-1$. Es gilt:

$$\begin{aligned}
P_j^i &\overset{\text{Def.}}{=} tP_j^{i-1} + (1-t)P_{j-1}^{i-1} \\
&\overset{\text{Ind.}}{=} t \cdot \left[\sum_{l=0}^{i-1} B_{i-1-l}^{i-1} P_{j-l}\right] + (1-t) \cdot \left[\sum_{l=0}^{i-1} B_{i-1-l}^{i-1} P_{j-1-l}\right] \\
&= t \cdot \left[\sum_{l=0}^{i-1} B_{i-1-l}^{i-1} P_{j-l}\right] + (1-t) \cdot \left[\sum_{l=1}^{i} B_{i-l}^{i-1} P_{j-l}\right] \\
&= t \cdot \underbrace{B_{i-1}^{i-1}}_{=t^{i-1}} P_j + \sum_{l=1}^{i-1} \left[t \cdot B_{i-1-l}^{i-1} + (1-t) \cdot B_{i-l}^{i-1}\right] P_{j-l} + (1-t) \cdot \underbrace{B_0^{i-1}}_{=(1-t)^{i-1}} P_{j-i} \\
&\overset{\text{Lemma 1}}{=} \underbrace{t^i}_{=B_i^i} P_j + \sum_{l=1}^{i-1} B_{i-l}^i P_{j-l} + \underbrace{(1-t)^i}_{=B_0^i} P_{j-i} = \sum_{l=0}^{i} B_{i-l}^i P_{j-l}.
\end{aligned}$$

■

Lemma 3: Es gilt: $P(t) = P(m, m)$

Beweis: $P(m,m) \overset{\text{Lemma 2}}{=} \sum_{l=0}^{m} B_{m-l}^{m} P_{m-l} = \sum_{k=0}^{m} B_k^m P_k = P(t)$

■

Der De Casteljau Algorithmus wird in der Abbildung 5 illustriert.

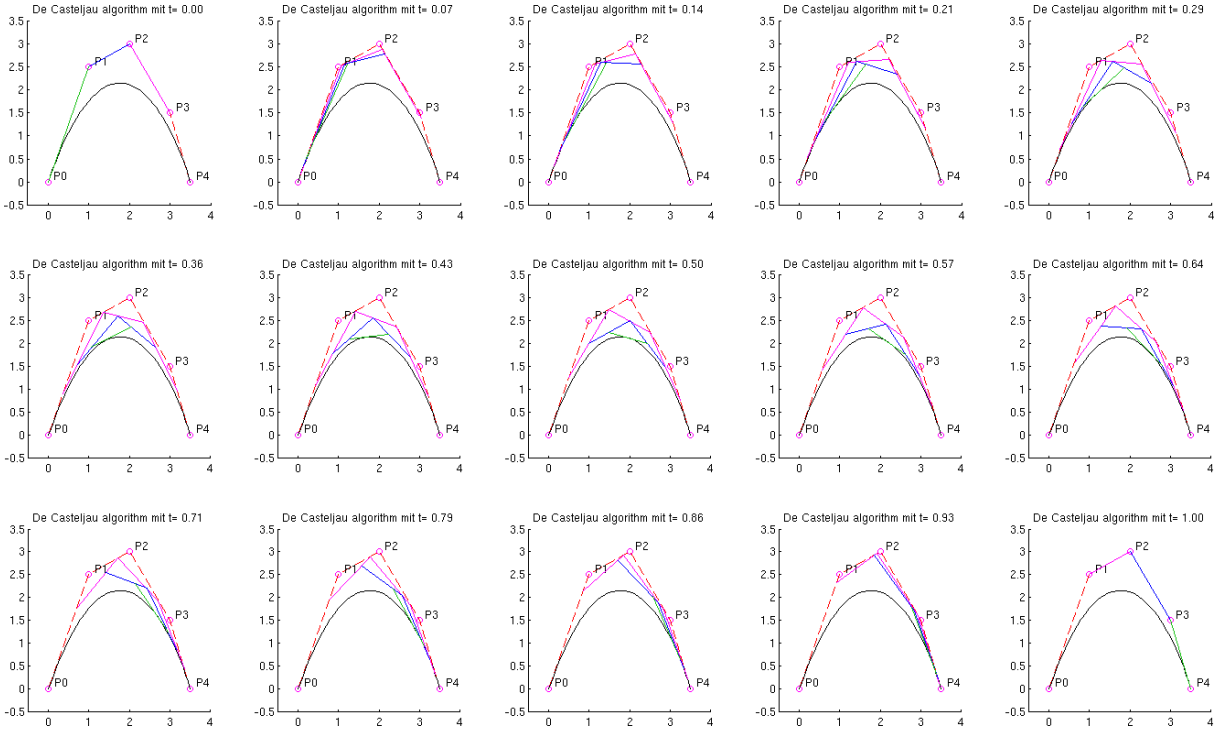


Abbildung 5: De Casteljau Algorithmus: Jede Abbildung stellt die Bézierkurve am Zeitpunkt $t \in [0, 1]$ dar. Die Geraden oberhalb der Kurve entsprechen die Verbindungslinien zwischen P_j^i und P_{j-1}^i. Die Berührungsstelle der Verbindungslinie mit der Kurve ist genau P_m^m.

2.4 Hülle und Schnittpunkt

Wie man in der Abbildung 2 sehen kann, bilden die Kontrollpunkte ein Polygon. Ist dieses Polygon konvex, dann liegt die Bézierkurve innerhalb des Polygons, welches auch als *konvexe Hülle* bezeichnet wird. Falls man wissen will, ob zwei Bézierkurven, die innerhalb ihren konvexen Hüllen liegen, ein Schnittpunkt besitzt, kann man zuerst überpüfen, ob die Hüllen ein Schnittpunkt hat. Hat die Hüllen keinen Schnittpunkt, somit haben die Kurven auch keinen. Falls sie einen haben, zerlege die Kurven jeweils in zwei Teilkurven und betrachte dann deren Hüllen. Haben die Hüllen keinen Schnittpunkt, dann haben die ursprüngliche Kurve auch keinen. Falls sie wieder einen Schnittpunkt haben, betrachte ihre Teilkurven rekursiv weiter bis die Teilkurven genügend klein sind. Dann haben die Kurven einen Schnittpunkt.

3 Bézieroberfläche

3.1 Definition

In Bézierkurve ist ein Kontrollpunkt ein Vektor und jeder Vektor zeigt eine Richtung. Er bestimmt die Richtung des Verlaufs der Kurve. Für die Flächenmodellierung werden zwei Punkte benötigt um eine Fläche darzustellen.

Seien $t_1, t_2 \in [0, 1]$ und $P_{i,j} \in \mathbb{R}^n$ mit $0 \leq i \leq m, 0 \leq j \leq k$. Die *Bézieroberfläche* $\mathcal{B}_{m,k}$ ist das Bild von folgendem Polynom:

$$P(t_1, t_2) = \sum_{i=0}^{m} \sum_{j=0}^{k} B_i^m(t_1) B_j^k(t_2) P_{ij}$$

Ein Beispiel wird in der Abbildung 6 dargestellt.

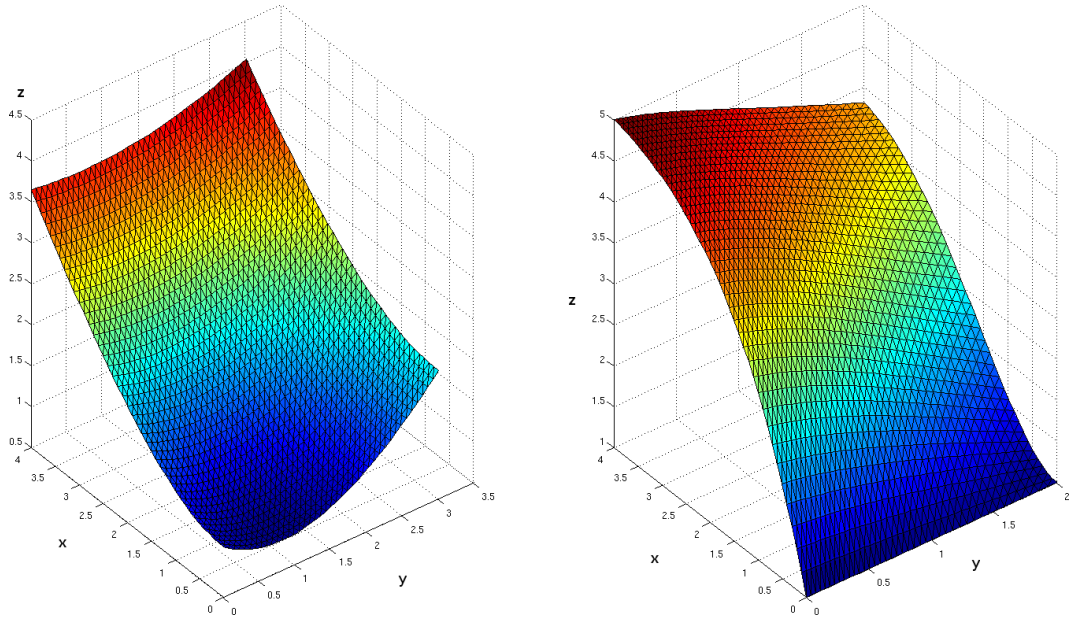


Abbildung 6: Zwei Bézieroberflächen

3.2 Boor-Coox Algorithmus

Die Punkte $P(t_1, t_2)$ werden hier ebenfalls nicht direkt ausgerechnet, sondern genau wie bei der Bézierkurve werden die Punkte rekursiv berechnet und zwar mit dem *Boor-Coox Algorithmus*. Man betrachte für $t_1, t_2 \in [0, 1]$

$$P(t_1, t_2) = \sum_{i=0}^{m}\sum_{j=0}^{k} B_i^m(t_1) B_j^k(t_2) P_{ij} = \sum_{i=0}^{m} B_i^m(t_1) \sum_{j=0}^{k} B_j^k(t_2) P_{ij} = \sum_{i=0}^{m} B_i^m(t_1) P_i \tag{1}$$

mit

$$P_i = \sum_{j=0}^{k} B_j^k(t_2) P_{ij} \tag{2}$$

für $0 \leq i \leq m, 0 \leq j \leq k$. Die Gleichung (1) kann man mit *De Casteljau Algorithmus* berechnen. Man bekommt $P_0, \ldots, P_m$. Mit diesen Punkten kann man wiederum mit *De Casteljau Algorithmus* die Gleichung (2) lösen. Der Boor-Coox Algorithmus lautet:

```
%% Boor-Coox Algorithmus:
% Input: P00,...,Pmk Punkte und t1, t2 in [0,1]
% Output: P(t1, t2)
% Pi berechnen
  for i = 0 : m
    Pi = Casteljau(Pi,1:k, t2)
% P(t) berechnen
    P(t1, t2) = Casteljau(P1:m, t1)
```

Literatur

[1] Ionut Danaila, Pascal Joly, Sidi Mahmoud Kaber, Marie Postel *An Introduction to Scientific Computing Twelve Computational Projects Solved with MATLAB*, Springer New York, 1. Auflage, 2006

[2] `http://www.mathi.uni-heidelberg.de/~theiders/PS-Analysis/Bernstein-Polynome.pdf`, *Bernstein-Polynome*

[3] `http://de.wikipedia.org/wiki/Bézierkurve`

[4] `http://de.wikibooks.org/wiki/Blender_Dokumentation:_Kurven`

[5] `http://en.wikipedia.org/wiki/Bézier_curve`

[6] `http://en.wikipedia.org/wiki/De_Casteljau's_algorithm`

[7] `http://idav.ucdavis.edu/education/CAGDNotes/Bernstein-Polynomials.pdf`

Inkompressible Navier-Stokes-Gleichungen in zwei Dimensionen*

Fabian Müller

Zusammenfassung

In diesem Projekt ging es darum, ein numerisches Verfahren für die Lösung der inkompressiblen Navier-Stokes-Gleichungen zu finden. Das Verfahren besteht darin, in jedem Zeitschritt eine Helmholtz- und eine Poissongleichung zu lösen. Auch für diese Gleichungen sollen Methoden hergeleitet werden. Schlussendlich finden sich noch zwei numerische Beispiele.

Inhaltsverzeichnis

1 Problemstellung

Unter einem *Fluid* versteht man in der Physik ein Gas oder eine Flüssigkeit. Die Bewegung eines Fluids in einem Gebiet $\Omega \subseteq \mathbb{R}^2$ ist zeit- und ortabhänging. Kennt man zu jedem Zeitpunkt $t \in [0, T]$ an jedem Ortspunkt $(x, y) \in \Omega$...

- ...den Druck $p(x, y; t) \in \mathbb{R}$,
- ...die Dichte $\varrho(x, y; t) \in \mathbb{R}$,
- und die Geschwindigkeit $v(x, y; t) = (\, v_1(x, y; t) \;\; v_2(x, y; t)\,) \in \mathbb{R}^2$,

so ist die Bewegung des Fluids bis zum Zeitpunkt T vollständig bestimmt.

Die *Navier-Stokes-Gleichungen* ist ein System partieller Differentialgleichungen, das eine Beziehung zwischen diesen Grössen vorgibt.

*Nach [1], Kapitel 12.

Im *inkompressiblen Fall*, der näherungsweise für Flüssigkeiten zutrifft, lauten die Gleichungen:

$$\frac{\partial v_1}{\partial x} + \frac{\partial v_2}{\partial y} = 0 \tag{1}$$

$$\frac{\partial v_1}{\partial t} + \frac{\partial v_1^2}{\partial x} + \frac{\partial v_1 v_2}{\partial y} = -\frac{\partial p}{\partial x} + \frac{1}{Re}\left(\frac{\partial^2 v_1}{\partial x^2} + \frac{\partial^2 v_1}{\partial y^2}\right) \tag{2}$$

$$\frac{\partial v_2}{\partial t} + \frac{\partial v_2^2}{\partial y} + \frac{\partial v_1 v_2}{\partial x} = -\frac{\partial p}{\partial y} + \frac{1}{Re}\left(\frac{\partial^2 v_2}{\partial x^2} + \frac{\partial^2 v_2}{\partial y^2}\right) \tag{3}$$

Hierbei ist $Re \in \mathbb{R}_{>0}$ (die *Reynolds*-Zahl) eine physikalische Konstante des Fluids, die bestimmt, wie turbulent die Fliessbewegung wird.

(1) - (3) ist ein gekoppeltes System nichtlinearer partieller Differentialgleichungen. Numerische Schwierigkeiten tauchen bei grossen Reynoldszahlen auf, da dann $\frac{1}{Re}$ so klein wird, dass maschinelle Ungenauigkeiten auftreten können.

Als Rechengebiet wird nun das Rechteck $\Omega := [0, L_x] \times [0, L_y]$ gewählt. Um Existenz und Eindeutigkeit der Lösung zu garantieren, werden noch Anfangsbedingungen und Randbedingungen vorgegeben. Unter *Anfangsbedingungen* versteht man die Vorgabe der exakten Lösung zum Zeitpunkt $t = 0$:

$$v(x, y; t = 0) \equiv v_0(x, y), \qquad p(x, y; t = 0) \equiv p_0(x, y).$$

Unter *Randbedingungen* versteht man Bedingungen an die Lösung am Rand $\partial\Omega$. In diesem Projekt werden *periodische Randbedingungen* mit Periode L_x, bzw. L_y gewählt, d.h.:

$$v(x+L_x, y; t) = v(x, y; t), \quad v(x, y+L_y; t) = v(x, y; t), \qquad \forall\, (x, y) \in \Omega, t \in [0, T],$$

$$p(x+L_x, y; t) = p(x, y; t), \quad p(x, y+L_y; t) = p(x, y; t), \qquad \forall\, (x, y) \in \Omega, t \in [0, T].$$

2 Zeitdiskretisierung

Die hier vorgestellte Methode besteht darin, das Problem zuerst in der Zeit zu diskretisieren und die Ortsableitungen nicht zu beachten.
Dazu diskretisiere das Zeitintervall äquidistant für fixes δt in $t_n := n\,\delta t$, $n \geq 0$.
Es seien nun zum Zeitpunkt t_n die Geschwindigkeitsfunktionen
$v^n(x, y) = (v_1^n(x, y), v_2^n(x, y))$ und $p^n(x, y)$ gegeben. Man suche nun die Funktionen v^{n+1} und p^{n+1}.

Umformung der Gleichungen: Definiere

$$\mathcal{H}_1 := -\left(\frac{\partial v_1^2}{\partial x} + \frac{\partial v_1 v_2}{\partial y}\right), \qquad \mathcal{H}_2 := -\left(\frac{\partial v_2^2}{\partial y} + \frac{\partial v_1 v_2}{\partial x}\right).$$

Die Gleichungen (2) und (3) lauten dann[1]

$$\frac{\partial v}{\partial t} = -\nabla p + \mathcal{H} + \frac{1}{Re}\Delta v, \qquad \mathcal{H} := (\mathcal{H}_1 \;\; \mathcal{H}_2)\,.$$

[1] Die Notation $\Delta v = \Delta \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$ steht kurz für den Vektor $\begin{bmatrix} \Delta v_1 \\ \Delta v_2 \end{bmatrix}$.

Nach Konstruktion ist $\mathcal{H}$ zeitabhänging. Setze $\mathcal{H}^n := \mathcal{H}(t^n)$.

Diskretisierung der Gleichungen (2) und (3): Um einen ersten Approximationswert $v^* \approx v^{n+1}$ zu erhalten, nehme $p^{n+1} = p^n$ an.
Die Gleichungen (2) und (3) lassen sich nun approximieren, indem man die restlichen Unbekannten $\mathcal{H}^{n+1}$ und Δv^{n+1} numerisch approximiert. In diesem Projekt wurde ein 2-Schritt Adams-Bashforth-Verfahren für $\mathcal{H}$ und das Crank-Nicolson-Verfahren für Δv gewählt. Man erhält nun eine Gleichung für v^*:

$$\left(I - \frac{\delta t}{2Re}\Delta\right)(v^* - v^n) = \delta t \left(-\nabla p^n + \frac{3}{2}\mathcal{H}^n - \frac{1}{2}\mathcal{H}^n\right). \tag{4}$$

Die Lösung v^* von Gleichung (4) erfüllt aber noch nicht (1), d.h. $\operatorname{div}(v^*) \neq 0$. Deswegen muss man v^* korrigieren, um v^{n+1} zu erhalten.

Korrektur von v^*: Um v^{n+1} zu erhalten suche man eine Funktion φ, so dass

$$v^{n+1} := v^* - \delta t\, \nabla\varphi \qquad \text{mit } \operatorname{div}\left(v^{n+1}\right) = 0. \tag{5}$$

Wendet man nun den Divergenzoperator auf die erste Gleichung an, so erhält man

$$\Delta\varphi = \frac{1}{\delta t}\operatorname{div}(v^*). \tag{6}$$

Hat man v^{n+1} ausgerechnet, braucht man noch den Druck p^{n+1} zum Zeitpunkt t_{n+1}, um den Algorithmus wieder von vorne starten zu lassen.

Berechnung des neuen Drucks: Aus v^{n+1} kann man den Druck p^{n+1} ausrechnen. Eine genauere Betrachtung liefert die Gleichung

$$p^{n+1} = p^n + \varphi - \frac{\delta t}{2Re}\Delta\varphi. \tag{7}$$

Das Zeitschrittverfahren lautet also folgendermassen:

Algorithmus (Fractional Step Method für 2D-inkompressible NSE)
Zum Zeitpunkt t_n seien v_n und p_n bekannt.

1. *Berechne $\mathcal{H}^n$.*
2. *Löse die Gleichung (4), um eine Approximation v^* zu erhalten.*
3. *Berechne φ als Lösung von (6) und erhalte v^{n+1} aus (5).*
4. *Berechne den Druck p^{n+1} mit (7).*

3 Ortsdiskretisierung

In den Schritten 2 und 3 des obigen Algorithmus sind jeweils lineare partielle Differentialgleichungen zu lösen, die nur vom Ort abhängen.
Für die Diskretisierung im Ort ist ein *Gitter* aufzustellen. Wähle dazu n_x und n_y, die Anzahl Punkte in x- bzw. y-Richtung; sowie Schrittweiten $\delta x := \dfrac{L_x}{n_x - 1}$

und $\delta y := \dfrac{L_y}{n_y - 1}$.
Man erhält nun ein Ortsgitter (x_i, y_j), $i = 1, \ldots, n_x - 1$, $j = 1, \ldots, n_y - 1$.
Die periodischen Randbedingungen lauten diskret

$$v(x_1, y_j) = v(x_{n_x}, y_j) \text{ und } v(x_i, y_1) = v(x_i, y_{n_y}) \quad \forall\, i, j.$$

ADI-Verfahren für die Helmholtzgleichung (4)

Der Differentialoperator in Gleichung (4) ist von der Form $(I - \delta t\, \lambda \Delta)$. Berechne nun:

$$\begin{aligned} \left(I - \delta t\, \lambda \frac{\partial}{\partial x^2}\right) \cdot \left(I - \delta t\, \lambda \frac{\partial}{\partial y^2}\right) &= I - \delta t\, \lambda \left(\frac{\partial}{\partial x^2} + \frac{\partial}{\partial y^2}\right) + \delta t^2\, \lambda^2 \frac{\partial}{\partial x^2} \frac{\partial}{\partial y^2} \\ &\approx I - \delta t\, \lambda \Delta + \mathcal{O}\left(\delta t^2\right). \end{aligned}$$

Wir erhalten also folgendes Verfahren der Ordnung 2:

Algorithmus (ADI-Verfahren)
Zu lösen sei $\left(I - \delta t\, \lambda \frac{\partial}{\partial x^2}\right) \left(I - \delta t\, \lambda \frac{\partial}{\partial y^2}\right) (v^* - v^n) = RHS$.

1. *Löse* $\left(I - \delta t\, \lambda \frac{\partial}{\partial x^2}\right) \widehat{V} = RHS$.
2. *Substituiere zurück:* $\left(I - \delta t\, \lambda \frac{\partial}{\partial y^2}\right) V = \widehat{V}$.
3. *Setze* $v^* := V + v^n$.

Um Gleichungen der Form $\left(I - \delta t\, \lambda \frac{\partial}{\partial x^2}\right) V = \text{RHS}$ zu lösen, ersetze $\frac{\partial}{\partial x^2}$ durch einen Differenzenquotienten der zweiten Ordnung:

$$\frac{\partial \widehat{V}}{\partial x^2}(x_i, y_j) \approx \frac{\widehat{V}(x_{i-1}, y_j) - 2\widehat{V}(x_i, y_j) + \widehat{V}(x_{i+1}, y_j)}{\delta x^2}.$$

Somit erhält man für Schritt 1 in jeder Zeile j ein Gleichungssystem der Form

$$-\beta_x \widehat{V}(i-1, j) + (1 + 2\beta_x)\, \widehat{V}(i, j) - \beta_x \widehat{V}(i+1, j) = \text{RHS}(i, j),$$

wobei $\beta_x := \dfrac{1}{2Re} \dfrac{\delta t}{\delta x^2}$. Die periodischen Randbedingungen lauten nun:

$$\begin{aligned} i = 1: &\qquad \widehat{V_k^*}(i-1, j) = \widehat{V_k^*}(0, j) = \widehat{V_k^*}(n_x - 1, j), \\ i = n_x - 1: &\qquad \widehat{V_k^*}(i+1, j) = \widehat{V_k^*}(n_x, j) = \widehat{V_k^*}(1, j). \end{aligned}$$

Dies entspricht $n_y - 1$ linearen Gleichungssystemen, die mit herkömmlichen Methoden zu lösen sind.

In Schritt 2 lautet der Differenzenquotient

$$\frac{\partial V}{\partial y^2}(x_i, y_j) \approx \frac{V(x_i, y_{j-1}) - 2V(x_i, y_j) + V(x_i, y_{j+1})}{\delta y^2}$$

und man kommt analog für jedes $i = 1, \ldots, n_x - 1$ auf $n_x - 1$ lineare Gleichungssysteme[2] der Form

$$-\beta_y V(i, j-1) + (1 + 2\beta_y)\, V(i, j) - \beta_y V(i, j+1) = \mathrm{RHS}(i, j),$$

wobei $\beta_y := \dfrac{1}{2Re}\dfrac{\delta t}{\delta y^2}$ und für $j = 0, n_y - 1$ analoge Randbedingungen gelten wie oben.

Ein Verfahren für die Poissongleichung (6)

Die Gleichung (6) involviert nur den Laplace-Operator $\Delta = \frac{\partial}{\partial x^2} + \frac{\partial}{\partial y^2}$.
Ersetzt man die zweiten Ableitungen durch die oben beschriebenen Differenzenquotienten, so erhält man:

$$\begin{aligned} \Delta\varphi &= \frac{\partial \varphi}{\partial x^2} + \frac{\partial \varphi}{\partial y^2} \\ &\approx \frac{\varphi(i-1, j) - 2\varphi(i, j) + \varphi(i+1, j)}{\delta x^2} + \frac{\varphi(i, j-1) - 2\varphi(i, j) + \varphi(i, j+1)}{\delta y^2}. \end{aligned}$$

Die Gleichung $\Delta\varphi = \frac{1}{\delta t}\,\mathrm{div}\,(v^*)$ diskretisiert sich also zu einem linearen Gleichungssystem, das nicht wie oben pro Zeile lösbar ist, denn jeder Punkt ist gleichzeitig an seine Nachbarn in x- sowie y-Richtung gekoppelt. Daher empfiehlt es sich, die Punkte nicht wie gehabt mit (i, j) zu indexieren, sondern einen Index einzuführen, der alle Punkte (z.b. von links nach rechts, von oben nach unten) durchnummeriert. Abbildung 1 zeigt, welche Werte den Wert am Punkt (i, j) beeinflussen. Weiter kann man ablesen, welche Indices sie bei der neuen Nummerierung tragen. Die periodischen Randbedingungen lauten gleich wie bei der Diskretisierung der Helmholtzgleichung.

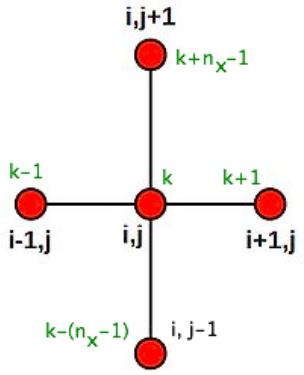


Abbildung 1: Fünf-Punkte-Stern zweiter Ordnung für den Laplaceoperator.

4 Numerische Beispiele

Zum Schluss des Projekts soll das hergeleitete Verfahren für zwei Beispiele ausgeführt werden. Es handelt sich dabei um Prozesse, die in den Anwendungen in

[2] es sei angemerkt, dass die beschriebenen Gleichungssysteme jeweils zwei Gleichungssystemen entsprechen, da V ein Vektor der Länge 2 ist.

drei Dimensionen oft simuliert werden, um fluidmechanische Prozesse im Detail zu verstehen.

Jetfluss

Die Grundsituation ist, dass zwei unvermischbare Flüssigkeiten aneinander vorbeiziehen, wobei die eine eine grössere Geschwindigkeit hat als die andere. Für eine Reynoldszahl gross genug entstehen dann Wirbel an den Rändern der schnelleren Flüssigkeit.
In Abbildung 2 wird der Zustand des Systems bei verschiedenen Zeitpunkten gezeigt. Man sieht, wie sich an den Rändern langsam Wirbel bilden, die gleich gerichtet sind. In diesem numerischen Experiment wurde Reynoldszahl 1000 gewählt, was eine gute Beobachtbarkeit des Phänomens erlaubt. Bei zu grossen Reynoldszahlen wird das System hochturbulent und es entstehen an vielen Orten Instabilitäten (siehe Abbildung 3).

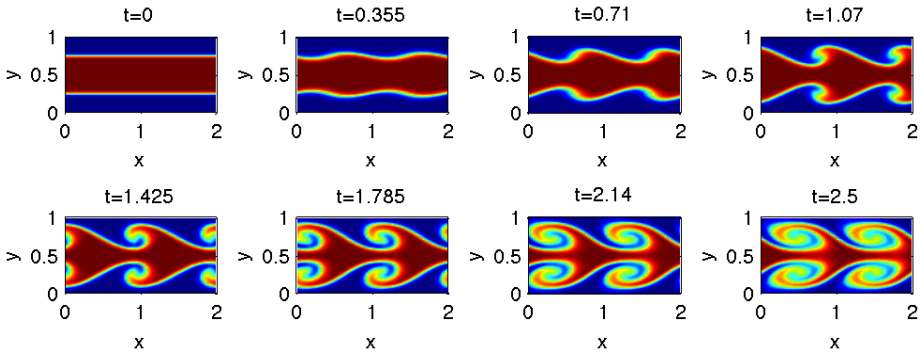


Abbildung 2: Entwicklung eines Jetflusses in der Zeit bei $Re = 1000$. Die rote Flüssigkeit bewegt sich schneller in x-Richtung als die Blaue.

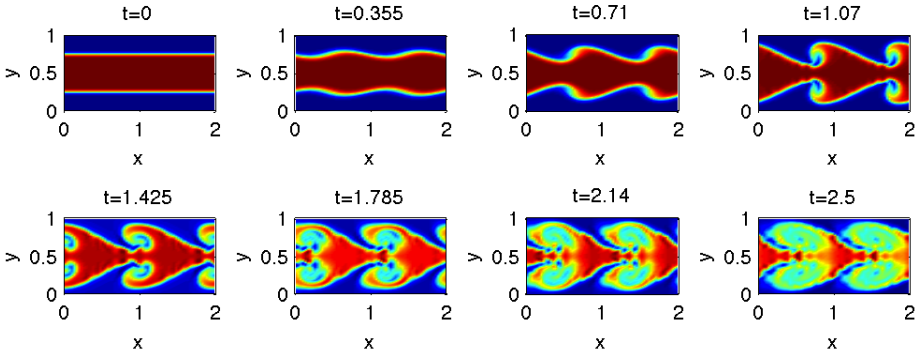


Abbildung 3: Entwicklung eines hochturbulenten Jetflusses in der Zeit bei $Re = 10^6$.

Wirbeldipol

Hier hat man zwei entgegengesetzte, gleich starke Wirbel, deren Zentren auf derselben y-Achsenparallelen liegen. Fangen sie sich nun an zu drehen, so regen sie sich zu einer Bewegung in x-Richtung an. In Abbildung 4 wird die Entwicklung des Systems in der Zeit beschrieben. Die Wirbel transportieren die Flüssigkeit in der Mitte langsam nach links und erhalten selbst eine Drift nach rechts.

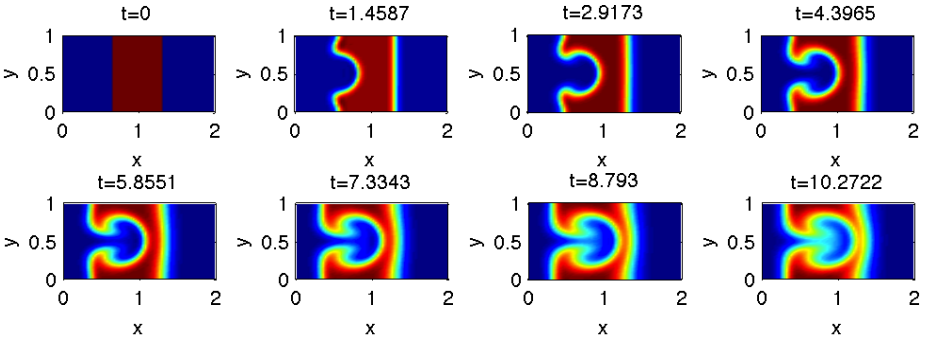


Abbildung 4: Entwicklung eines Wirbeldipols in der Zeit bei $Re = 1000$. Die Wirbel fangen links im Bild an zu rotieren, bewegen sich gemeinsam nach rechts und transportieren dabei immer mehr rote Flüssigkeit nach links.

Literatur

[1] I. Danaila, P. Joly, S. M. Kaber, M. Postel, *An introduction to scientific computing: Twelve computational projects solved with MATLAB*, Springer, 2007.