



PROGRAMMIER-PRAKTIKUM PHYSIK / CES

Numerische Lösung des Minimalflächenproblems

Dipl.-Ing J. Lotz, Stud. B.Sc. F. Lux, Prof. Dr. U. Naumann

Wintersemester 2012/13

Inhaltsverzeichnis

1 Einführung	2
2 Mathematischer Hintergrund	3
2.1 Partielle Differentialgleichungen	3
2.2 Diskretisierung des Gebietes	6
2.3 Diskretisierung der Problemstellung	7
2.4 Lösung eines Systems nicht-linearer Gleichungen	11
2.5 Zusammenfassung	13
3 Minimalflächenproblem	14
3.1 Einleitung	14
3.2 Einführung in die Variationsrechnung	15
3.3 Lagrangesche Minimalflächengleichung	18
3.4 Diskretisierung	20
3.5 Minimalflächen in Natur und Technik	21
4 Algorithmen und Implementierung	23
4.1 Berechnung des Residuums	24
4.2 Berechnung der Jacobimatrix	24
4.2.1 Ausnutzung der Dünnbesetztheit	26
4.2.2 Algorithmus: Dünnbesetztheitsstruktur	27
4.2.3 Algorithmus: Distanz-1-Färbung	29
4.2.4 Zusammenfassung: Berechnung der Jacobimatrix	31
4.3 Berechnung der Lösung des nicht-linearen Gleichungssystems	34
4.3.1 Verwendung der GSL Bibliothek	34
4.4 Visualisierung	36
4.4.1 Datenformat	36
4.4.2 gnuplot	36
4.4.3 Paraview	39
A Referenzproblem	42
B Datenstruktur des Residuums	44

Kapitel 1

Einführung

Ziel dieses Praktikums ist es, die bisher gelernten Inhalte zu vertiefen und einen Einblick in algorithmische Vorgehensweise zu erlangen. Dies wird im Rahmen der Lösung eines numerischen Problems aus der Physik, beziehungsweise der Mathematik, geschehen. Genauer gesagt: der Lösung des sogenannten Minimalflächenproblems. Um dieses Praktikum erfolgreich zu bestehen, ist es nicht notwendig jede mathematische Finesse zu durchdringen. Der Schwerpunkt wird für Sie darauf liegen, die in der Globalübung erläuterte Vorgehensweise in C++ Code zu übersetzen. Zusätzliche Informationen zur gegebenen Problemstellung können allerdings hilfreich bei der Lösung der Übungsaufgaben sein (auch wenn sie dazu nicht notwendig sind). Die zusätzlichen Informationen dienen auch dazu, Ihnen aufzuzeigen wo die dargestellten Lösungswege ihre Anwendung finden, beziehungsweise wo diese später für Sie nützlich sein können. Während die nötigen Hintergrundinformationen in der Vorlesung vermittelt werden, ist es ihre Aufgabe die dort angedeuteten Algorithmen in den Aufgaben der Teilprojekte zum Leben zu erwecken. Die Abgabe der Lösungen erfolgt in ihrem SVN Verzeichnis (näheres dazu auf den Übungsblättern). Bei der Korrektur Ihrer Lösungen werden diese stichprobenartig auf ihre Sinnhaftigkeit überprüft und nicht auf Korrektheit. Es wird aber von Ihnen erwartet, dass sie sich mit den Aufgabenstellung auseinandersetzen, was an ihrer Aktivität im SVN Repository zu erkennen ist.

Kapitel 2

Mathematischer Hintergrund

2.1 Partielle Differentialgleichungen

Zweidimensionale *partielle Differentialgleichungen* sind Gleichungen der Form

$$F(x, y, z(x, y), \frac{\partial z(x, y)}{\partial x}, \frac{\partial z(x, y)}{\partial y}, \dots, \frac{\partial^2 z(x, y)}{\partial x \partial y}, \dots) = 0. \quad (2.1)$$

In einer solchen Gleichung treten neben der Funktion $z : \mathbb{R}^2 \mapsto \mathbb{R}$ auch partielle Ableitungen dieser Funktion auf. Darunter versteht man die Ableitungen einer Funktion mehrerer Variablen nach einer dieser Variablen.

Beispiel 1 :

$$\frac{\partial (y \cdot x^2 + z \cdot e^y)}{\partial x} = 2y \cdot x.$$

Ist m die höchste auftretende Ableitungsordnung, so hat die Differentialgleichung die Ordnung m . Jede Funktion $z(x, y)$, welche auf einem Gebiet¹ $\Omega \subset \mathbb{R}^2$ (Teilgebiet des \mathbb{R}^2) definiert ist und für alle $(x, y) \in \Omega$ die Differentialgleichung 2.1 erfüllt, heißt eine Lösung der Differentialgleichung. Im zweidimensionalen Fall stellt jede Lösung $z(x, y)$ über Ω eine Fläche dar [?].

Beispiel 2 :

$$\frac{\partial z(x, y)}{\partial x} = 2, \text{ mit } \Omega = \mathbb{R}^2.$$

In Worten bedeutet diese Gleichung, dass die Steigung der Funktion in x -Richtung immer den konstanten Wert 2 besitzt. Trägt man die Funktion $z(x, y)$ für einen konstanten y -Wert gegen x auf, ergibt sich also als allgemeine Lösung eine Geradengleichung. Über Integration erhält man:

$$z(x, y) = \int \frac{\partial z(x, y)}{\partial x} dx = 2x + c(y). \quad (2.2)$$

¹Im \mathbb{R}^2 ist ein Gebiet eine einfach zusammenhängende Fläche, deren Rand nicht zum Gebiet selbst gehört.

Die Lösung ist nur bis auf eine y -abhängige Funktion $c(y)$ bekannt. Um $c(y)$ zu bestimmen fehlt eine Gleichung, welche Informationen über das Verhalten der Lösungsfunktion in y -Richtung Auskunft gibt. Beispielsweise:

$$\frac{\partial z(x,y)}{\partial y} = \cos y, \quad \text{mit } \Omega = \mathbb{R}^2.$$

Setzt man hier die zuvor erhaltene Gleichung 2.2 ein, lässt sich $c(y)$ und damit die allgemeine Lösung bestimmen:

$$\begin{aligned} \frac{\partial c(y)}{\partial y} &= \cos y \\ \Rightarrow c(y) &= \int \frac{\partial u(y)}{\partial y} dy = \sin y + C, \quad C \in \mathbb{R}. \end{aligned}$$

Einsetzen in 2.2:

$$z(x,y) = 2x + \sin y + C.$$

Übrig bleibt ein konstanter Term, der weder von x , noch von y abhängt. Als Lösung erhält man damit eine *einparametrische Schar*. Dies ist eine Gruppe von Funktionen, welche sich nur in einem Parameter (in diesem Fall C) unterscheiden. Abbildung 2.1 zeigt eine Visualisierung dieser Funktion im Programm Mathematica für verschiedene Werte von C .

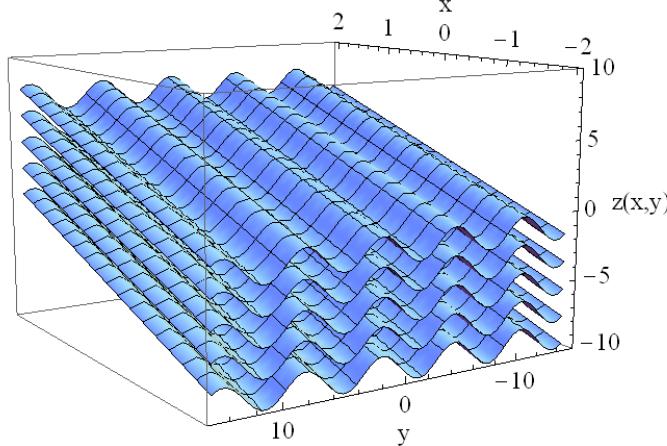


Abbildung 2.1: Lösung des Beispielproblems.

Wird ein Punkt festgelegt, welcher auf der Lösungsfläche liegen soll, so ist der Parameter C bestimmt. Es gibt jedoch auch Beispiele, in denen die Angabe eines Punktes nicht genügt, wie im Fall der eindimensionalen Schwingungsgleichung.

Beispiel 3 (eindimensionale Schwingungsgleichung):

$$\frac{d^2 f(x)}{dx^2} + k^2 \cdot f(x) = 0 \quad \text{mit } \Omega = (1, 2) \subset \mathbb{R}, k \in \mathbb{R}.$$

Die allgemeine Lösung dieser Gleichung ist:

$$y(x) = A \sin(kx) + B \cos(kx), \quad A, B \in \mathbb{R}.$$

Eine Möglichkeit die Konstanten A und B zu bestimmen besteht darin, Funktionswerte auf dem Rand $\delta\Omega$ des Gebietes Ω festzulegen. Im Beispiel besteht der Rand des Gebietes aus den zwei Punkten $x = 1$ und $x = 2$:

$$y(1) \stackrel{!}{=} y_1 = A \sin(k) + B \cos(k),$$

$$y(2) \stackrel{!}{=} y_2 = A \sin(2k) + B \cos(2k).$$

Durch die beiden Bedingungen an $y(x)$ gewinnt man ein lineares Gleichungssystem zur Bestimmung von A und B . Die festgelegten Randwerte bezeichnet man als *Dirichlet-Randbedingungen*.

In der Verallgemeinerung auf zwei Dimensionen bleibt es nicht bei der Angabe zweier Punkte. Prinzipiell muss der gesamte Rand durch die Dirichlet-Randbedingungen erfasst werden, welcher im zweidimensionalen Fall einer oder mehreren geschlossenen Kurven entsprechen kann. Mehrere geschlossene Kurven sind beispielsweise nötig, wenn das Gebiet Löcher enthält.

In den gezeigten Beispielen waren die Differentialgleichungen *linear* in der gesuchten Funktion. Das bedeutet, dass Vielfache einer Lösung wieder Lösungen der Differentialgleichung sind und die Addition zweier Lösungen ebenfalls auf eine Lösung der Differentialgleichung führt. Man kann diese beiden Eigenschaften auf die Linearität des Ableitungsoperators zurückführen. Komplizierter wird es, wenn eine *nicht-lineare* Differentialgleichung vorliegt, wie es unter anderem beim sogenannten Bratu-Problem der Fall ist:

$$\frac{\partial^2 z}{\partial x^2}(x, y) + \frac{\partial^2 z}{\partial y^2}(x, y) = \lambda \cdot \exp(z(x, y)).$$

Die Ableitungen sind zwar auch hier lineare Operationen, da die gesuchte Funktion z jedoch im Argument der Exponentialfunktion auftaucht, ist diese Differentialgleichung nicht-linear. Sei u_1 ein Lösung des Bratu-Problems. Wenn die Differentialgleichung linear wäre, so müsste auch $u_2 = \alpha \cdot u_1$ mit $\alpha \in \mathbb{R} \setminus \{0\}$ eine Lösung sein:

$$\begin{aligned} & \frac{\partial^2 u_2}{\partial x^2}(x, y) + \frac{\partial^2 u_2}{\partial y^2}(x, y) = \lambda \cdot \exp(u_2(x, y)) \\ \Leftrightarrow & \alpha \cdot \left(\frac{\partial^2 u_1}{\partial x^2}(x, y) + \frac{\partial^2 u_1}{\partial y^2}(x, y) \right) = \lambda \cdot \exp(\alpha \cdot u_1(x, y)) \\ \Leftrightarrow & \alpha \cdot \lambda \cdot \exp(u_1(x, y)) = \lambda \cdot \exp(\alpha \cdot u_1(x, y)). \end{aligned}$$

Die Annahme das Bratu-Problem sei linear führt auf einen Widerspruch, wodurch die gegenteilige Annahme bewiesen ist.

2.2 Diskretisierung des Gebietes

Selbst wenn partielle Differentialgleichungen linear sind, ist eine analytische Lösung oft nicht möglich. Als Alternative bietet sich eine *numerische* oder *diskrete* Lösung an, welche das Lösungsgebiet durch einzelne Punkte approximiert anstatt, wie bei der analytischen Lösung, unendlich viele Punkte zu betrachten. Die numerische Lösung einer partiellen Differentialgleichung ist eine Näherung und kann keine geschlossene Formel liefern. Ein erster Schritt auf dem Weg zu dieser Lösung ist die *Diskretisierung*, mit der man den Übergang vom kontinuierlichen Lösungsgebiet Ω zur diskreten Punktemenge Ω_H bezeichnet:

$$\underbrace{\Omega \rightarrow \Omega_H}_{\text{Diskretisierung}}$$

Im Folgenden werden wieder zweidimensionale Differentialgleichungen eine Rolle spielen, deren Lösungsfläche $z(x, y)$ bestimmt werden soll. Als Lösungsgebiet $\Omega \subset \mathbb{R}^2$ werden wir dabei ein Einheitsquadrat betrachten. Besonders einfach ist in diesem Fall eine sogenannte äquidistante² Diskretisierung, bei der sich insgesamt $s \times s$ Datenpunkte ergeben, wobei s die Anzahl der Datenpunkte pro Raumrichtung ist. Jeder Punkt dieses Gitters erhält eine eindeutige Indizierung. Der Index i soll entlang der x -Achse laufen, der Index j entlang der y -Achse. In Abbildung 2.2 ist dies für den Fall $s = 5$ veranschaulicht. Das verwendete Einheitsquadrat hat per Definition die Kantenlänge eins. Daher ist die Gitterbreite durch die Anzahl der Punkte bestimmt:

$$h = \frac{1}{s-1}.$$

Legt man das Quadrat in den ersten Quadranten ($x > 0, y > 0$) und setzt die Ecke $(\hat{x}_0, \hat{y}_0) = (0, 0)$ fest, lässt sich die Indizierung angeben als:

$$(\hat{x}_i, \hat{y}_j) := (i \cdot h, j \cdot h) \quad \text{mit } i, j = 1, \dots, s.$$

Mit $\hat{\mathbf{z}}$ soll die diskrete Lösung bezeichnet werden. Sie existiert nur an den eingezeichneten Datenpunkten (\hat{x}, \hat{y}) , welche Elemente der Menge Ω_H sind, und nicht wie die analytische Lösung $z(x, y)$ an allen Punkten des Gebietes Ω .

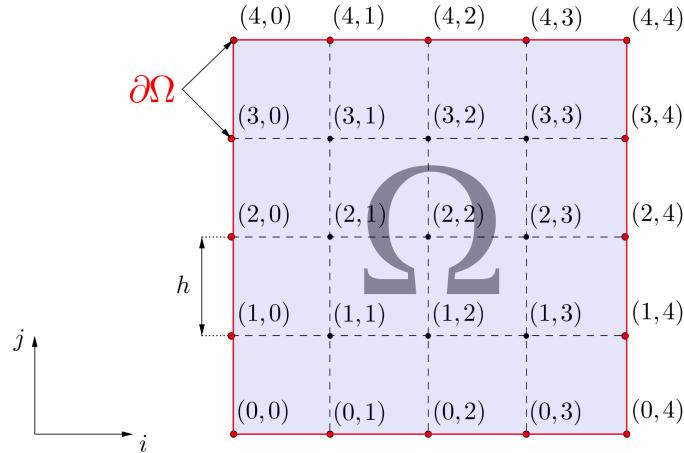
$\hat{\mathbf{z}}$ ist ein Vektor im \mathbb{R}^n , wobei $n = s \cdot s$. Im Fall der Diskretisierung mit äquidistantem Gitter lässt sich dieser Vektor $\hat{\mathbf{z}}$ auch bijektiv³ auf eine Matrix \mathcal{Z} abbilden:

$$\underbrace{\hat{\mathbf{z}} \in \mathbb{R}^n \leftrightarrow \mathcal{Z} \in \mathbb{R}^{s \times s}}_{\text{Bijektion}}$$

Die Elemente der Matrix können so angeordnet werden, dass \mathcal{Z}_{ij} den Funktionswert am Punkt (\hat{x}_i, \hat{y}_j) enthält.

²Der Abstand h zwischen den Datenpunkten ist über das gesamte Gebiet identisch.

³Die Abbildung ist umkehrbar eindeutig.

Abbildung 2.2: Diskretisierung des Gebietes Ω mit Rand $\partial\Omega$ in rot.

2.3 Diskretisierung der Problemstellung

Da die Differentialgleichung noch in der ursprünglichen kontinuierlichen Formulierung ist, kann sie nicht mehr auf das diskretisierte Gebiet angewendet werden. Stattdessen muss man sie zu einer Bedingung an $\hat{\mathbf{z}}$ ändern. Dies ist die Diskretisierung der Problemstellung. Wir verwenden hier das Verfahren der *finiten Differenzen* zur Approximation der auftretenden partiellen Ableitungen. Im eindimensionalen Fall ergibt sich sofort eine intuitive Näherung aus den beiden Steigungsdreiecken in Abbildung 2.3. Für jeden Punkt x kann die Steigung der Beispieldfunktion $f(x)$ durch das davor liegende Steigungsdreieck f'_- , oder das dahinter liegende Steigungsdreieck f'_+ approximiert werden:

$$f'_+(x) = \frac{f(x+h) - f(x)}{h} ,$$

$$f'_-(x) = \frac{f(x) - f(x-h)}{h} .$$

Der Mittelwert beider Steigungsdreiecke wird als *zentraler Differenzenquotient* bezeichnet und wird von uns als Näherung für $f'(x)$ verwendet:

$$f'(x) \approx \frac{f'_+(x) + f'_-(x)}{2} = \frac{f(x+h) - f(x-h)}{2h} .$$

Analog gilt dieser Zusammenhang auch für die Richtungsableitung mehrdimensionaler Funktionen:

$$\frac{\partial z}{\partial x}(\hat{x}_i, \hat{y}_j) \approx \frac{\mathcal{Z}(\hat{x}_i + h, \hat{y}_j) - \mathcal{Z}(\hat{x}_i - h, \hat{y}_j)}{2h} ,$$

$$\frac{\partial z}{\partial y}(\hat{x}_i, \hat{y}_j) \approx \frac{\mathcal{Z}(\hat{x}_i, \hat{y}_j + h) - \mathcal{Z}(\hat{x}_i, \hat{y}_j - h)}{2h} .$$

In unserem Beispiel ist h die Gitterbreite der äquidistanten Diskretisierung.

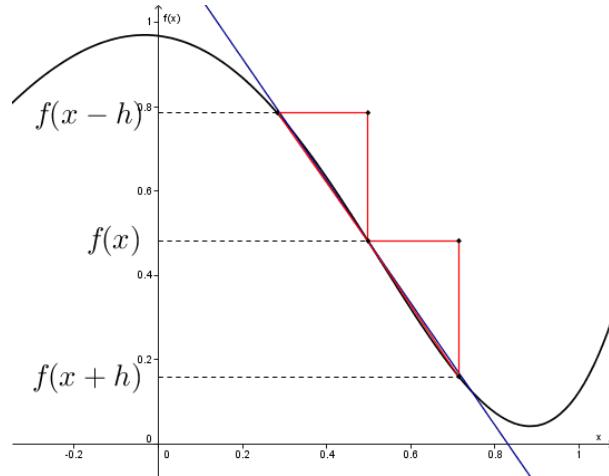


Abbildung 2.3: Ermittlung des zentralen Differenzenquotienten über Steigungs-dreiecke.

Dadurch ergibt sich mit vereinfachter Schreibweise $(\hat{x}_i, \hat{y}_j) \rightarrow (i, j)$:

$$\frac{\partial z}{\partial x}(i, j) \approx \frac{\mathcal{Z}_{i+1,j} - \mathcal{Z}_{i-1,j}}{2h},$$

$$\frac{\partial z}{\partial y}(i, j) \approx \frac{\mathcal{Z}_{i,j+1} - \mathcal{Z}_{i,j-1}}{2h}.$$

Alternative Erklärung (freiwillig) : Eine weitere Möglichkeit die partiellen Ableitungen zu approximieren ist es, die Taylorentwicklungen der Beispieldunktionen $f(x + h)$ und $f(x - h)$ zu betrachten:

$$f(x + h) = f(x) + f'(x)h + \frac{1}{2}f''(x)h^2 - \frac{1}{6}f'''(x)h^3 + \mathcal{O}(h^4) \quad (2.3)$$

$$f(x - h) = f(x) - f'(x)h + \frac{1}{2}f''(x)h^2 + \frac{1}{6}f'''(x)h^3 + \mathcal{O}(h^4) \quad (2.4)$$

Durch Addition der Gleichungen 2.3 und 2.4 ergibt sich die Beziehung

$$\frac{f(x - h) - 2f(x) + f(x + h)}{h^2} = f''(x) + \mathcal{O}(h^2).$$

Neben einer Näherung der zweiten Ableitung ergibt sich auf diese Weise auch die *Größenordnung* des Diskretisierungsfehlers zu $\mathcal{O}(h^2)$. Halbiert man h , so verkleinert sich der Diskretisierungsfehler um den Faktor vier. Ein analoger Zusammenhang gilt wieder für die multivariate Funktion $z(x, y)$:

$$\frac{\partial^2 z}{\partial x^2}(\hat{x}_i, \hat{y}_j) \approx \frac{\mathcal{Z}(\hat{x}_i - h, \hat{y}_j) - 2\mathcal{Z}(\hat{x}_i, \hat{y}_j) + \mathcal{Z}(\hat{x}_i + h, \hat{y}_j)}{h^2},$$

$$\frac{\partial^2 z}{\partial y^2}(\hat{x}_i, \hat{y}_j) \approx \frac{\mathcal{Z}(\hat{x}_i, \hat{y}_j - h) - 2\mathcal{Z}(\hat{x}_i, \hat{y}_j) + \mathcal{Z}(\hat{x}_i, \hat{y}_j + h)}{h^2}.$$

Und auch diesen Ausdruck kann man über die Indizierung formulieren:

$$\frac{\partial^2 z}{\partial x^2}(\hat{x}_i, \hat{y}_j) \approx \frac{\mathcal{Z}_{i-1,j} - 2\mathcal{Z}_{i,j} + \mathcal{Z}_{i+1,j}}{h^2},$$

$$\frac{\partial^2 z}{\partial y^2}(\hat{x}_i, \hat{y}_j) \approx \frac{\mathcal{Z}_{i,j-1} - 2\mathcal{Z}_{i,j} + \mathcal{Z}_{i,j+1}}{h^2}.$$

Am Beispiel der gemischten partiellen Ableitung kann man sehen, dass man auch durch geometrische Überlegungen und mit Hilfe der Differenzenformel auf das richtige Ergebnis kommt (allerdings für den Preis, dass jede Information über die Größenordnung des Fehlers verloren geht). Abbildung 2.4 legt nahe die Differenzenformel in gewisser Weise „rekursiv“ anzuwenden:

$$\frac{\partial^2 z}{\partial x \partial y}(\hat{x}_i, \hat{y}_j) = \frac{\partial}{\partial y} \left(\underbrace{\frac{\partial z}{\partial x}(\hat{x}_i, \hat{y}_j)}_{=:g(\hat{x}_i, \hat{y}_j)} \right) = \frac{g(\hat{x}_i, \hat{y}_j + h) - g(\hat{x}_i, \hat{y}_j - h)}{2h} \quad (2.5)$$

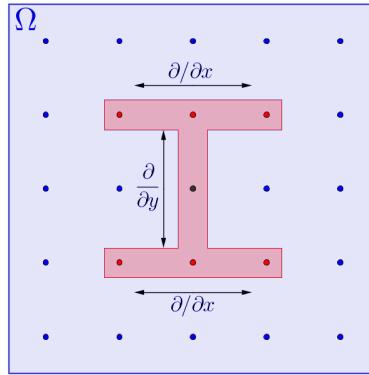


Abbildung 2.4: Geometrische Überlegung zur Berechnung der gemischten partiellen Ableitung.

Bei der Auflösung von innen nach außen betrachtet man zunächst den finiten Differenzenquotienten in x -Richtung an den Stellen $(\hat{x}, \hat{y} + h)$ und $(\hat{x}, \hat{y} - h)$:

$$\begin{aligned} g(\hat{x}, \hat{y} + h) &\approx \frac{\mathcal{Z}(\hat{x} + h, \hat{y} + h) - \mathcal{Z}(\hat{x} - h, \hat{y} + h)}{2h} = \frac{\mathcal{Z}_{i+1,j+1} - \mathcal{Z}_{i-1,j+1}}{2h} \\ g(\hat{x}, \hat{y} - h) &\approx \frac{\mathcal{Z}(\hat{x} + h, \hat{y} - h) - \mathcal{Z}(\hat{x} - h, \hat{y} - h)}{2h} = \frac{\mathcal{Z}_{i+1,j-1} - \mathcal{Z}_{i-1,j-1}}{2h} \end{aligned}$$

Einsetzen in 2.5:

$$\frac{\partial^2 z}{\partial x \partial y}(\hat{x}, \hat{y}) \approx \frac{\mathcal{Z}_{i+1,j+1} - \mathcal{Z}_{i-1,j+1} - \mathcal{Z}_{i+1,j-1} + \mathcal{Z}_{i-1,j-1}}{4h^2}$$

Auf diese Weise ließe sich auch für beliebige andere partielle Ableitungen eine diskrete Variante bestimmen. Für die Problemstellung des Praktikums werden

aber nur Ableitungen bis zur zweiten Ordnung benötigt. Ersetzt man alle partiellen Ableitungen der kontinuierlichen Differentialgleichung durch die oben hergeleiteten Differenzenquotienten, erhält man ein Gleichungssystem in $n = s \cdot s$ Variablen (Anzahl der Gitterpunkte). Die Ursache dafür ist, dass im Verfahren der finiten Differenzen alle Ableitungen nur punktweise betrachtet werden können. Hat man die Differentialgleichung F in impliziter Form gegeben und verwendet man finite Differenzen, so erhält man für jeden Punkt der Diskretisierung Ω_H eine Gleichung:

$$F(x, y, z(x, y)) = 0 \xrightarrow{\text{F.D.}} R(\hat{z}_{i,j}) = 0$$

Oder zusammenfassend:

$$R(\hat{\mathbf{z}}) = 0$$

Mit R bezeichnet man das *Residuum* der Differentialgleichung. Gesucht ist der Vektor $\hat{\mathbf{z}}$, beziehungsweise die Matrix \mathcal{Z} , deren Elemente alle Gleichungen lösen. Kann man das Gleichungssystem nur näherungsweise lösen, so gibt das Residuum ein Maß für die Güte der Lösung, als Abweichung vom gewünschten Ergebnis (in unserem Fall Null).

Hat man eine lineare Differentialgleichung gegeben, so führt die Anwendung finiter Differenzen auf ein lineares Gleichungssystem. Im nicht-linearen Fall ergibt sich entsprechend ein nicht-lineares Gleichungssystem.

Beispiel : Als lineares Beispiel sei die folgende eindimensionale Differentialgleichung gegeben:

$$\frac{d^2u}{dx^2}(x) = 2 \quad \text{mit } \Omega = (0, 1) \subset \mathbb{R} .$$

Für eine äquidistante Diskretisierung bestehend aus N Punkten einschließlich der beiden Randwerte ergibt sich mit $h = 1/(N - 1)$:

$$u_{i-1} - 2u_i + u_{i+1} - 2h^2 = 0 \quad \forall i \in \mathbb{N} : 1 < i < N - 1 .$$

Eine äquivalente Formulierung bietet die Matrixschreibweise:

$$\begin{pmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & 1 & -2 & 1 & \\ & & \ddots & \ddots & \ddots \\ & & & 1 & -2 & 1 \end{pmatrix} \cdot \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{N-2} \\ u_{N-1} \end{pmatrix} = \begin{pmatrix} 2h^2 - u_0 \\ 2h^2 \\ \vdots \\ 2h^2 \\ 2h^2 - u_{N-1} \end{pmatrix} .$$

Dies ist ein *unterbestimmtes* Gleichungssystem. Für die Punkte u_0 und u_{N-1} kann keine Gleichung formuliert werden, da man aufgrund des Differenzenquotienten Informationen über Punkte außerhalb des Gebietes benötigen würde. Die Randwerte des Gebietes sind damit Parameter in der numerischen Lösung. Betrachtet man den Spezialfall $N = 6$ mit $u_0 = 1$ und $u_5 = -1$:

$$\begin{pmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & 1 & -2 & 1 & \\ & & 1 & -2 & \end{pmatrix} \cdot \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix} = \begin{pmatrix} -0.92 \\ 0.08 \\ 0.08 \\ 1.08 \end{pmatrix}$$

$$\Rightarrow \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix} = \begin{pmatrix} 0.44 \\ -0.04 \\ -0.44 \\ -0.76 \end{pmatrix}.$$

Vergleicht man dieses Ergebnis mit der analytischen Lösung, welche durch $u(x) = x^2 - 3x + 1$ gegeben ist, stellt man fest, dass man auf diese Weise die exakte Lösung an jedem der diskreten Punkte ermitteln konnte. Dies ist überraschend, da man die Ableitungen nur näherungsweise betrachtet hat. Da die analytische Lösung ein Polynom zweiten Grades darstellt, verschwinden jedoch alle Ableitungsordnungen größer zwei und die Differenzenquotienten werden exakt. In Abbildung 2.5 ist eine Visualisierung der Lösung im Programm Maple 15 dargestellt.

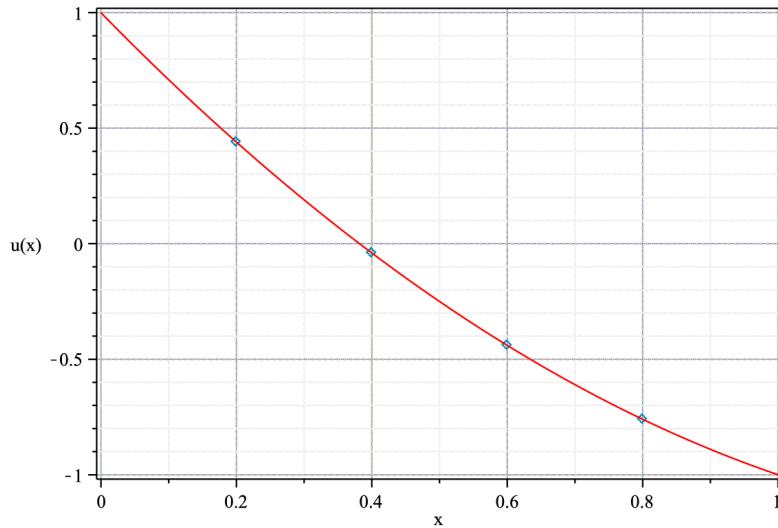


Abbildung 2.5: Lösung des Beispielproblems in Maple 15. Der analytischen Lösung entspricht die rote Kurve, der numerischen die blauen Punkte.

2.4 Lösung eines Systems nicht-linearer Gleichungen

Das Bratu-Problem führt für alle $(\hat{x}_i, \hat{y}_j) \in \Omega_H \setminus \partial\Omega$ auf die Gleichung:

$$\mathcal{Z}_{i-1,j} + \mathcal{Z}_{i+1,j} + \mathcal{Z}_{i,j-1} + \mathcal{Z}_{i,j+1} - 4\mathcal{Z}_{i,j} - h^2\lambda \cdot e^{\mathcal{Z}_{i,j}} = 0.$$

Diese Gleichung repräsentiert ein nicht-lineares System, in dem es im Allgemeinen nicht möglich ist nach $\mathcal{Z}_{i,j}$ aufzulösen. Man wendet daher numerische Verfahren an. Im eindimensionalen Fall ist das Newtonverfahren aus der Vorlesung bekannt. Für eine Residuumsfunktion $r(x)$ lautet die *Iterationsvorschrift*

$$x_{i+1} = x_i + \frac{r(x_i)}{r'(x_i)}.$$

Dies lässt sich auch auf den Fall der Nullstellen mehrdimensionaler Funktionen erweitern. Insbesondere lässt sich das Residuum als eine solche mehrdimensionale Funktion auffassen:

$$R : \mathbb{R}^n \rightarrow \mathbb{R}^n, \hat{\mathbf{z}} \rightarrow R(\hat{\mathbf{z}}) .$$

Die mehrdimensionale Verallgemeinerung des Newton-Verfahrens ist dann folgende:

$$\hat{\mathbf{z}}^{i+1} = \hat{\mathbf{z}}^i - \left(R'(\hat{\mathbf{z}}^i) \right)^{-1} R(\hat{\mathbf{z}}^i) . \quad (2.6)$$

Der Index i bezeichnet hier den Iterationsschritt und $R'(\hat{\mathbf{z}})$ ist die mehrdimensionale Ableitung der Residuumsfunktion, welche auch *Jacobimatrix* genannt wird. Sie ist wie folgt definiert:

$$R'(\hat{\mathbf{z}}) = \left(R'_{i,j} \right) \in \mathbb{R}^{n \times n}, \quad R'_{i,j} = \frac{\partial R_i}{\partial \hat{z}_j}(\hat{\mathbf{z}}) .$$

Im Eintrag (i, j) von $R'(\hat{\mathbf{z}})$ wird demnach die i -te Residuumsfunktion nach der j -ten Variablen abgeleitet. Dies setzt eine eindeutige Nummerierung der Diskretisierung voraus, was durch die Wahl eines äquidistanten Gitters einfach zu gewährleisten ist (siehe vorheriges Kapitel). Nach der vorherrschenden Konvention bezeichnet man mit dem ersten Index i die Zeile der Jacobimatrix und mit dem zweiten Index j die Spalte:

$$R'(\hat{\mathbf{z}}) = \begin{pmatrix} \frac{\partial R_1}{\partial \hat{z}_1}(\hat{\mathbf{z}}) & \cdots & \frac{\partial R_1}{\partial \hat{z}_n}(\hat{\mathbf{z}}) \\ \vdots & \ddots & \vdots \\ \frac{\partial R_n}{\partial \hat{z}_1}(\hat{\mathbf{z}}) & \cdots & \frac{\partial R_n}{\partial \hat{z}_n}(\hat{\mathbf{z}}) \end{pmatrix} .$$

Anschaulich gesprochen enthält die Jacobimatrix Informationen darüber, in welchem Maße die Ausgabewerte von den Eingabewerten abhängen. Als Beispiel betrachten wir die Funktion $F : \mathbb{R}^3 \rightarrow \mathbb{R}^3$:

$$F(x, y, z) = \begin{pmatrix} xyz \\ y \sin(x) \\ yz \cdot e^x \end{pmatrix} \Rightarrow F'(x, y, z) = \begin{pmatrix} yz & xz & xy \\ y \cos(x) & \sin(x) & 0 \\ yz \cdot e^x & z \cdot e^x & ye^x \end{pmatrix} .$$

Dadurch, dass die Jacobimatrix von F einen Nulleintrag an der Stelle $(2, 3)$ besitzt, kann man eindeutig schließen, dass die zweite Komponente von F nicht von z abhängt. Diese Tatsache wird in Kapitel 4.2.1 explizit verwendet werden. Das mehrdimensionale Newton-Verfahren werden wir jedoch nicht implementieren. Stattdessen ziehen wir bereits vorhandene Implementierungen heran (*Bibliotheken*). Die Bibliothek GSL stellt Verfahren zur Verfügung ⁴ zu deren Anwendung unter anderem die Jacobimatrix benötigt wird.

⁴Es handelt sich um Verfahren, welche ähnlich zum hier gezeigten Newton Verfahren funktionieren.

2.5 Zusammenfassung

Insgesamt spricht man bei dem vorgestellten Lösungsweg vom sogenannten *finiten Differenzenverfahren* zur Lösung einer partiellen Differentialgleichung. Die Lösungsschritte werden von [?] so zusammengefasst:

1. Diskretisierung des Rechengebiets
2. Diskretisierung der Gleichungen und Randbedingungen.
3. Aufstellen des zugehörigen Gleichungssystems.
4. Lösen des Gleichungssystems mit einer geeigneten Methode (z.B. mit einer numerischen Bibliothek).
5. Visualisierung der berechneten Lösung.

Kapitel 3

Minimalflächenproblem

3.1 Einleitung



Abbildung 3.1: Seifenbläser, Öl auf Leinwand von Jean Siméon Chardin (18. Jahrhundert).

„Seifenblasen galten üblicherweise als Allegorie des Illusionshaften, als Symbole der Unbeständigkeit und Vergänglichkeit, und so konnte ein (in mehreren Versionen existierendes) Bild des französischen Rokokomalers Chardin (1699-1779), welches einen in seine ephemer Tätigkeit vertieften jugendlichen Seifenbläser zeigt, auch nur allzuleicht als vollkommener Ausdruck der Geisteshaltung seiner Epoche verstanden werden.“

Als aber dann im letzten Jahrhundert der belgische Physiker Plateau (1801-1883) durch Experimente, bei denen er eine Drahtschlinge in Seifenlauge tauchte und dann die Form der sich in dem Draht einspannenden Seifenhaut beobachtete, aufgrund der Vielfalt und Schönheit der auftretenden Formen und Gestalten die Faszination seiner Zeitgenossen erregte, waren Seifenblasen nunmehr zum anschaulichen und ästhetisch reizvollen Ausdruck physikalischer Gesetzmäßigkeiten geworden.

Da Seifenhäute sehr dünn und daher leicht sind, lässt sich meist der Einfluß der Schwerkraft auf ihre Gestalt vernachlässigen, und sie streben aufgrund inhärenter Molekularkräfte dazu, sich soweit wie möglich zusammenzuziehen. Aufgrund der Experimente Plateaus lag es daher nahe, das mathematische Problem zu studieren, in eine vorgegebene Jordankurve im dreidimensionalen Raum eine Fläche kleinsten Inhaltes - eine Minimalfläche - einzuspannen.“

- Auszug aus J.Jost, *Das Existenzproblem für Minimalflächen* [?].

3.2 Einführung in die Variationsrechnung

Das mathematische Problem der Bestimmung einer Minimalfläche fällt in das Gebiet der *Variationsrechnung*. Um ein einfaches Beispiel als Einführung in die Methoden der Variationsrechnung zu geben betrachten wir eine verwandte Problemstellung im zweidimensionalen Fall. Wir suchen mit Hilfe der Variationsrechnung die kürzeste Verbindung zwischen zwei Punkten $A = (x_A, y_A)$ und $B = (x_B, y_B)$. Ist diesem Fall ist die Lösung trivial: offensichtlich ist eine Gerade \overline{AB} die einzige Lösung. In einer vereinfachenden Annahme gehen wir davon aus, dass sich die Verbindungsstrecke von A nach B durch eine Funktion $y = y(x)$ darstellen lässt (siehe Abbildung 3.2). Die Kurvenlänge können wir dann bestimmen, indem wir die infinitesimalen *Wegelemente* ds entlang der Kurve aufsummieren, das heißt integrieren. Aus dem Satz des Pythagoras erhält man:

$$ds = \sqrt{dx^2 + dy^2} = \sqrt{1 + \frac{dy^2}{dx^2}} dx = \sqrt{1 + y'(x)^2} dx .$$

Damit lautet das Integral der Kurvenlänge:

$$L = \int_A^B ds = \int_{x_A}^{x_B} \sqrt{1 + y'(x)^2} dx .$$

L ist eine Funktion, welche von der Beschaffenheit einer anderen Funktion abhängt. In diesem Zusammenhang spricht man bei L von einem *Funktional*. Um deutlich zu machen, dass L ein Funktional ist, schreibt man die funktionale Abhängigkeit von L in eckige Klammern:

$$L = L[x, y(x)] .$$

Die Bedingung, dass sich bei kleiner Änderung der Funktion $y(x)$ das Funktional L nicht ändert ist notwendig dafür, dass L minimal wird. Es ist gleichbedeutend

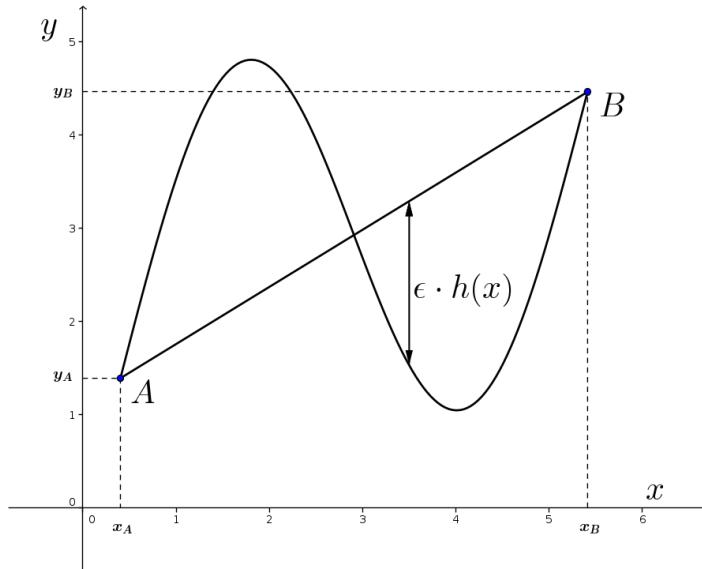


Abbildung 3.2: Kürzeste Verbindung zwischen zwei Punkten und Variation mit der Störfunktion $h(x)$.

mit der Aussage, dass L stationär ist. Ist L stationär, so könnte im Allgemeinen jedoch auch ein Sattelpunkt oder ein Maximum vorliegen, was von der Problemstellung abhängig ist. Zur Untersuchung des Funktionals L betrachten wir eine *Störung* der Funktion $y(x)$:

$$y(x) \rightarrow y(x) + \epsilon \cdot h(x), \quad \epsilon \in \mathbb{R}.$$

Dabei soll $h(x)$ eine beliebige Funktion sein, welche an den Punkten A und B den Wert null annimmt. Man betrachtet jetzt die *Gâteaux-Ableitung*¹ δL :

$$\delta L = \lim_{\epsilon \rightarrow 0} \frac{L[x, y + \epsilon h] - L[x, y]}{\epsilon} = \left. \frac{dL[x, y + \epsilon h]}{d\epsilon} \right|_{\epsilon=0}.$$

Sie beschreibt das Änderungsverhalten von L bei einer Variation mit der Funktion $h(x)$ in Abhängigkeit von ϵ . An der Stelle $\epsilon = 0$ stimmen die Funktionale $L[x, y + \epsilon h]$ und $L[x, y]$ überein. Die Gâteaux-Ableitung gibt genau den Wert an, um den L bei infinitesimaler Änderung von ϵ bei gegebener Störfunktion $h(x)$ variiert. Daher wird diese Ableitung auch *Variationsableitung* genannt. Ist L stationär muss die Variation verschwinden:

$$\delta L = 0.$$

Diese Gleichung können wir ausnutzen um eine Differentialgleichung zur Bestimmung von $y(x)$ zu gewinnen:

$$\delta L = \left[\frac{d}{d\epsilon} \int_{x_A}^{x_B} \sqrt{1 + (y'(x) + \epsilon h'(x))^2} dx \right]_{\epsilon=0}$$

¹Die mehrdimensionale Ableitung von L in Richtung der Störung h

$$\begin{aligned}
&= \int_{x_A}^{x_B} \left[\frac{d}{d\epsilon} \sqrt{1 + (y'(x) + \epsilon h'(x))^2} \right]_{\epsilon=0} dx \\
&= \int_{x_A}^{x_B} \left[\frac{h'(x)(y'(x) + \epsilon h'(x))}{\sqrt{1 + (y'(x) + \epsilon h'(x))^2}} \right]_{\epsilon=0} dx \\
&= \int_{x_A}^{x_B} \frac{h'(x)y'(x)}{\sqrt{1 + y'(x)^2}} dx .
\end{aligned}$$

Durch eine partielle Integration (P.I.) können wir die Ableitung von h auf y umwälzen:

$$\stackrel{\text{P.I.}}{\Rightarrow} \delta L = \left[\frac{h(x)y'(x)}{\sqrt{1 + y'(x)^2}} \right]_{x_A}^{x_B} - \int_{x_A}^{x_B} h \left(\frac{d}{dx} \frac{y'(x)}{\sqrt{1 + y'(x)^2}} \right) dx .$$

Da nach Voraussetzung $h(x_A) = h(x_B) = 0$ gilt, ist der erste Term null.

$$\Rightarrow \delta L = - \int_{x_A}^{x_B} h \left(\frac{d}{dx} \frac{y'(x)}{\sqrt{1 + y'(x)^2}} \right) dx = 0$$

$h(x)$ ist eine beliebe Störfunktion, welche die Randbedingungen erfüllt. Damit dieses Integral für alle $h(x)$ den Wert null ergibt muss also der Ausdruck in Klammern gleich null sein:

$$\begin{aligned}
&\frac{d}{dx} \frac{y'(x)}{\sqrt{1 + y'(x)^2}} = 0 \\
\Rightarrow &\frac{y'(x)}{\sqrt{1 + y'(x)^2}} = C_1, \quad C_1 \in \mathbb{R} \\
\Rightarrow &y'(x) = \sqrt{\frac{C_1^2}{1 - C_1^2}} = \text{const.}
\end{aligned}$$

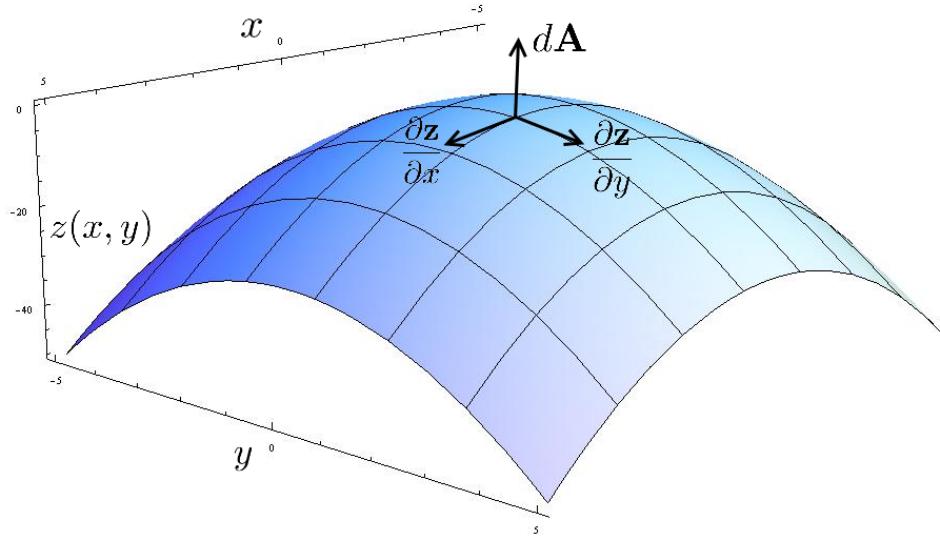
Bei konstanter Steigung folgt durch Integration:

$$y(x) = mx + b, \quad m, b \in \mathbb{R} .$$

Damit erhält man als allgemeine Lösung eine Geradengleichung mit den reellen Paramtern m und b . Diese müssen den Randbedingungen entsprechend gewählt werden. Man erhält:

$$y(x) = \frac{y_B - y_A}{x_B - x_A} \cdot x + \left(y_A - \frac{y_B - y_A}{x_B - x_A} \cdot x_B \right) .$$

Dies entspricht der erwarteten Lösung einer direkten Verbindung zwischen den Punkten A und B .

Abbildung 3.3: Tangentialvektoren an $z(x, y) := -(x^2 + y^2)$.

3.3 Lagrangesche Minimalflächengleichung

Die mathematische Herangehensweise bei Bestimmung von Minimalflächen ist dem Problem des vorherigen Kapitels analog und stellt die zweidimensionale Verallgemeinerung dar. Anstelle der einfach zu lösenden, eindimensionalen Differentialgleichung treten jedoch im zweidimensionalen Fall partielle Differentialgleichungen, welche im Allgemeinen nicht analytisch lösbar sind. Am Anfang steht wieder die Formulierung eines Funktionals (in diesem Fall die Formulierung des Flächenfunktionals A). Bei der Aufstellung des Functionals L haben wir über infinitesimale Streckenelemente integriert. Zur Bestimmung von A müssen wir daher analog über infinitesimale Flächenelemente integrieren. Wir nehmen an, dass sich die gesuchte Minimalfläche als eine Funktion $z : \mathbb{R}^2 \mapsto \mathbb{R}$, $(x, y) \mapsto z(x, y)$ auffassen lässt, dann ergibt sich folgende vektorielle Darstellung dieser Fläche:

$$\mathbf{z}(x, y) = \begin{pmatrix} x \\ y \\ z(x, y) \end{pmatrix}.$$

$z(x, y)$ wird durch x und y parametrisiert. Gedanklich hat man die Fläche selbst also mit einem Koordinatengitter überzogen. Bewegt man sich in x -Richtung um den Wert 1 entlang dieses Gitters, muss die zurückgelegte Strecke nicht notwendigerweise 1 betragen. Die Form der Fläche bestimmt den Wert dieses *Wegelements*. Betrachtet man differentielle Änderungen von x und y so geben die *Tangentialvektoren* an in welche Richtung und mit welcher Steigung die Achsen des krummlinigen Koordinatengitters verlaufen:

$$\frac{\partial \mathbf{z}}{\partial x} = \begin{pmatrix} 1 \\ 0 \\ z_x \end{pmatrix} \text{ und } \frac{\partial \mathbf{z}}{\partial y} = \begin{pmatrix} 0 \\ 1 \\ z_y \end{pmatrix} \text{ mit } z_x = \frac{\partial z(x, y)}{\partial x}, z_y = \frac{\partial z(x, y)}{\partial y}.$$

Das differentielle Flächenelement dA lässt sich dann aus dem Kreuzprodukt der Tangentialvektoren gewinnen:

$$dA = \left\| \frac{\partial \mathbf{z}}{\partial x} \times \frac{\partial \mathbf{z}}{\partial y} \right\| = \sqrt{1 + z_x^2 + z_y^2} .$$

Der gesamte Flächeninhalt folgt schließlich durch Integration über das gesamte Gebiet Ω auf dem $z(x, y)$ definiert ist:

$$A = \int_{\Omega} dA .$$

Um dieses Integral zu berechnen gehen wir davon aus, dass das Gebiet in x -Richtung durch zwei konstante Werte x_1 und x_2 begrenzt wird und die Begrenzungen in die y -Richtung durch die Werte y_1 und y_2 gegeben ist. Dieses Gebiet hat die Form eines Rechtecks. Die Berechnung des Funktionals A wird dadurch leichter zu handhaben:

$$A[x, y, z] = \int_{x_1}^{x_2} \int_{y_1}^{y_2} \sqrt{1 + z_x^2 + z_y^2} \ dydx .$$

Von diesem Funktional verlangen wir wieder, dass es stationär wird:

$$\delta A = 0 .$$

Als Störung verwenden wir ebenfalls analog

$$z(x, y) \rightarrow z(x, y) + \epsilon h(x, y) .$$

mit einer beliebigen Störfunktion $h(x, y)$ für die gilt: $h(x, y) = 0$ falls $(x, y) \in \partial\Omega$ und betrachten die Variationsableitung:

$$\begin{aligned} \delta A &= \frac{dA[x, y, z + \epsilon h]}{d\epsilon} \Big|_{\epsilon=0} \\ &= \int_{x_1}^{x_2} \int_{y_1}^{y_2} \frac{d}{d\epsilon} \left(\sqrt{1 + (z_x + \epsilon h_x)^2 + (z_y + \epsilon h_y)^2} \right) \Big|_{\epsilon=0} dydx \\ &= \int_{x_1}^{x_2} \int_{y_1}^{y_2} \left[\frac{(z_x + \epsilon h_x)h_x + (z_y + \epsilon h_y)h_y}{\sqrt{1 + (z_x + \epsilon h_x)^2 + (z_y + \epsilon h_y)^2}} \right]_{\epsilon=0} dydx \\ &= \int_{x_1}^{x_2} \int_{y_1}^{y_2} \frac{z_x h_x + z_y h_y}{\sqrt{1 + z_x^2 + z_y^2}} \ dydx . \end{aligned}$$

Dadurch, dass das Gebiet ein Rechteck ist, kann die Integrationsreihenfolge vertauscht werden:

$$\delta A = \int_{x_1}^{x_2} \int_{y_1}^{y_2} \frac{z_y h_y}{\sqrt{1 + z_x^2 + z_y^2}} \ dydx + \int_{y_1}^{y_2} \int_{x_1}^{x_2} \frac{z_x h_x}{\sqrt{1 + z_x^2 + z_y^2}} \ dxdy .$$

Ganz analog zum eindimensionalen Fall kann jetzt durch eine partielle Integration die Ableitung von h auf z umgewälzt werden

$$\begin{aligned} & \stackrel{\text{P.I.}}{=} \int_{x_1}^{x_2} \left[\frac{h z_x}{\sqrt{1+z_x^2+z_y^2}} \Big|_{y_1}^{y_2} - \int_{y_1}^{y_2} h \cdot \frac{d}{dx} \frac{z_x}{\sqrt{1+z_x^2+z_y^2}} dy \right] dx \\ & + \int_{y_1}^{y_2} \left[\frac{h z_y}{\sqrt{1+z_x^2+z_y^2}} \Big|_{x_1}^{x_2} - \int_{x_1}^{x_2} h \cdot \frac{d}{dy} \frac{z_y}{\sqrt{1+z_x^2+z_y^2}} dx \right] dy \end{aligned}$$

h verschwindet am Rand $\partial\Omega$ des Gebietes Ω :

$$\delta A = \int_{x_1}^{x_2} \int_{y_1}^{y_2} h \cdot \left(\frac{d}{dx} \frac{z_x}{\sqrt{1+z_x^2+z_y^2}} + \frac{d}{dy} \frac{z_y}{\sqrt{1+z_x^2+z_y^2}} \right) dy dx .$$

Da $h(x, y)$ beliebig gewählt werden kann muss gelten:

$$\begin{aligned} & \Leftrightarrow \frac{d}{dx} \left(\frac{z_x}{\sqrt{1+z_x^2+z_y^2}} \right) + \frac{d}{dy} \left(\frac{z_y}{\sqrt{1+z_x^2+z_y^2}} \right) = 0 \\ & \Rightarrow \frac{z_{xx}+z_{yy}}{\sqrt{1+z_x^2+z_y^2}} - \frac{z_x(z_x z_{xx}+z_y z_{xy})+z_y(z_y z_{yy}+z_x f_{xy})}{(1+f_x^2+f_y^2)^{3/2}} = 0 \\ & \Leftrightarrow (1+z_x^2)z_{yy} - 2z_x z_y z_{xy} + (1+z_y^2)z_{xx} = 0 . \end{aligned}$$

Diese partielle Differentialgleichung ist die Minimalflächengleichung nach Lagrange. Im Folgenden wollen wir uns damit beschäftigen wie man diese Gleichung in C++ löst.

3.4 Diskretisierung

Für das Problem der Minimalfläche wurde im vorherigen Kapitel die Gleichung

$$\underbrace{(1+z_x^2)z_{yy}}_{I(x,y)} - \underbrace{2z_x z_y z_{xy}}_{II(x,y)} + \underbrace{(1+z_y^2)z_{xx}}_{III(x,y)} = 0$$

hergeleitet. Die in Kapitel 2 erarbeitete Diskretisierung der partiellen Ableitung lässt sich in diesem Fall anwenden:

$$\begin{aligned} I(i,j) &= \left(1 + \left(\frac{\mathcal{Z}_{i+1,j} - \mathcal{Z}_{i-1,j}}{2h} \right)^2 \right) \frac{\mathcal{Z}_{i,j-1} - 2\mathcal{Z}_{i,j} + \mathcal{Z}_{i,j+1}}{h^2} , \\ II(i,j) &= -\frac{1}{8h^3} (\mathcal{Z}_{i+1,j} - \mathcal{Z}_{i-1,j})(\mathcal{Z}_{i,j+1} - \mathcal{Z}_{i,j-1}) \\ &\quad \cdot (\mathcal{Z}_{i+1,j+1} - \mathcal{Z}_{i-1,j+1} - \mathcal{Z}_{i+1,j-1} + \mathcal{Z}_{i-1,j-1}) , \\ III(i,j) &= \left(1 + \left(\frac{\mathcal{Z}_{i,j+1} - \mathcal{Z}_{i,j-1}}{2h} \right)^2 \right) \frac{\mathcal{Z}_{i-1,j} - 2\tilde{z}_{i,j} + \mathcal{Z}_{i+1,j}}{h^2} . \end{aligned}$$

Unter der Annahme, dass das äquidistant diskretisierte Gebiet (inklusive Rand) mit s Knoten pro Seite ein Einheitsquadrat darstellt, lässt sich die Gitterbreite h aus Abbildung 2.2 ablesen:

$$h = \frac{1}{s-1} .$$

Damit definieren wir die Residuumsmatrix $R = (R_{i,j}) \in \mathbb{R}^{s \times s}$:

$$R_{i,j} = I(i,j) + II(i,j) + III(i,j) .$$

Die Abhängigkeit des Residuums von den Funktionswerten des Gebietes lässt sich anschaulich in einem *Differenzenstern* (auch *Differenzenmolekül* genannt) darstellen (siehe Abbildung 3.4).

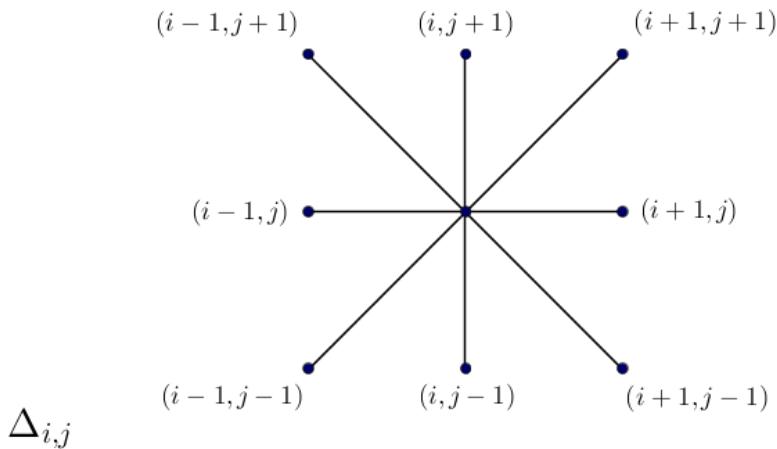


Abbildung 3.4: Differenzenstern des Minimalflächenproblems.

3.5 Minimalflächen in Natur und Technik

Abbildung 3.5 zeigt eine Seifenhaut, welche über einem Gestell aus Kupferdraht eine Minimalfläche aufspannt. Was passiert physikalisch? Betrachtet man ein Molekül Inneren der Seifenlauge, so ist die resultierende Kraft, welche von anderen Molekülen auf dieses Molekül ausgeübt wird, im zeitlichen Mittel gleich Null. Das gilt jedoch nicht mehr für ein Molekül an der Oberfläche der Seifenlauge. Denn außerhalb der Seifenlauge gibt es keine Moleküle, welche eine anziehende Kraft auf Moleküle der Oberfläche ausüben könnte. Es bleibt eine resultierende Kraft ins Innere der Seifenlauge. Um ein Molekül aus dem Inneren an die Oberfläche zu bringen, muss gegen diese Kraft Arbeit geleistet werden. Da zur Vergrößerung der Oberfläche um ΔA Moleküle aus dem Innern an die Oberfläche gebracht werden müssen, muss die Energie ΔW aufgewendet werden (vgl. [?]). Der Quotient dieser Größen ist als *spezifische Oberflächenenergie* definiert:

$$\epsilon = \frac{\Delta W}{\Delta A} .$$

Die Moleküle der Seifenlauge streben danach, diese Energie zu minimieren. Ist ϵ positiv, so ist die Minimierung der Energie äquivalent zur Minimierung der Oberfläche (vgl. [?]). Diese natürliche Minimierung funktioniert so gut, dass sie zu Zeiten geringer Rechenleistung verwendet wurde, um zu komplexen architektonischen Problemen effizient eine Lösung zu finden. So wurde beispielsweise die Dachkonstruktion des Olympiastadions in München (siehe Abbildung 3.6) mit Seifenhautmodellen simuliert um zu gegebenen Randbedingungen die stabilste Lösung mit geringstem Materialaufwand zu finden.



Abbildung 3.5: Erzeugung von Minimalflächen zu bestimmten Randbedingungen mittels Seifenlauge.



Abbildung 3.6: Minimalflächen in der Architektur: Das Münchener Olympiastadion.

Kapitel 4

Algorithmen und Implementierung

Im Allgemeinen hat man es auf dem Weg zur numerischen Lösung eines physikalischen Problems mit folgenden Punkten zu tun:

1. Beobachtung eines physikalischen Prozesses.
2. Modellierung mit Hilfe einer mathematischen Beschreibung.
3. Implementierung der aufgestellten Gleichungen.
4. Anwendung numerischer Verfahren zur Lösung der Gleichungen, beziehungsweise Nutzung vorhandener numerischen Bibliotheken.
5. Visualisierung der Lösung, beziehungsweise Nutzung vorhandener Visualisierungsprogramme.
6. Eventuelle Optimierung bezüglich der Punkte 2,3 oder 4.

Die Punkte 3 bis 5 sind dabei jene, welche unter den Oberbegriff *Softwareentwicklung* fallen, und mit denen wir uns maßgeblich befassen wollen. Das Praktikum ist für diesen Zweck in folgende Teile angelegt:

1. Eigene Implementierung der Problemstellung.
2. Eigene Implementierung numerische Verfahren.
3. Benutzung externer Bibliotheken (NAG, GSL) zur Lösung der aufgestellten Gleichungen.
4. Visualisierung der Lösung mit der OpenSource-Software Gnuplot und Paraview.

4.1 Berechnung des Residuums

Im Folgenden sollen alle Bezeichnungen und Voraussetzungen der vorherigen Kapitel weiterhin gelten:

- $\Omega \subset \mathbb{R}^2$ bezeichnet das Lösungsgebiet. Es handelt sich dabei um das Einheitsquadrat $\Omega = [0, 1] \times [0, 1]$.
- Die Diskretisierung erfolgt durch ein äquidistantes Gitter mit s Gitterpunkten pro Seite. Damit lautet die Gitterbreite $h = 1/(s-1)$. Die Anzahl der Punkte ergibt sich zu $n = s^2$.
- $\Omega_H \subset \mathbb{R}^2$ ist die Menge der diskreten Punkte
- Die Matrix $\mathcal{Z} \in \mathbb{R}^{s \times s}$ beinhaltet die Funktionswerte an den diskreten Punkten.
- $\hat{\mathbf{z}} \in \mathbb{R}^n$ geht durch die bijektive Abbildung von \mathcal{Z} auf einen Vektor hervor.
- Das Residuum kann als Funktion $R : \mathbb{R}^n \mapsto \mathbb{R}^n$ aufgefasst werden. Es gibt für einen gegebenen Vektor $\hat{\mathbf{z}}$ an, welche Differenz zum Nullvektor $\mathbf{0}$ besteht.

Alle Routinen, welche die Diskretisierung betreffen, sind in der Klasse *Discretization* enthalten. Zur Berechnung des Residuums stellt diese Klasse die Funktion *calc_residual* zur Verfügung. Für einen gegebenen Eingabevektor $\hat{\mathbf{z}}$ berechnet diese Routine das Residuum und gibt das Ergebnis in einem separaten Vektor gleicher Dimension zurück.

```
void calc_residual(const X* IN, X* OUT);
```

Das entspricht der mathematischen Formulierung mit der Residuumsfunktion R :

$$\mathbf{OUT} = R(\mathbf{IN}) ,$$

wobei $\mathbf{IN}, \mathbf{OUT} \in \mathbb{R}^n$.

4.2 Berechnung der Jacobimatrix

Um das System nicht-linearer Gleichungen zu lösen, welches sich durch die Diskretisierung ergeben hat, benötigt die Bibliothek GSL die Jacobimatrix des Residuums. Diese hatten wir in Kapitel 2.4 wie folgt eingeführt:

$$R'(\hat{\mathbf{z}}) = \left(R'_{i,j} \right) \in \mathbb{R}^{n \times n}, \quad R'_{i,j} = \frac{\partial R_i}{\partial \hat{z}_j}(\hat{\mathbf{z}}) .$$

Die partielle Ableitung lässt sich analog zum eindimensionalen Fall verstehen:

$$R'_{i,j} = \frac{\partial R_i}{\partial \hat{z}_j}(\hat{\mathbf{z}}) = \lim_{\epsilon \rightarrow 0} \frac{R_i(\hat{\mathbf{z}} + \epsilon \cdot \mathbf{e}_j) - R_i(\hat{\mathbf{z}})}{\epsilon} ,$$

wobei \mathbf{e}_j der j -te Einheitsvektor im \mathbb{R}^n ist. Die finite Differenzen Approximation für hinreichend kleines ϵ lautet damit:

$$R'_{i,j} \approx \frac{R_i(\hat{\mathbf{z}} + \epsilon \cdot \mathbf{e}_j) - R_i(\hat{\mathbf{z}})}{\epsilon}.$$

Beispiel zur Jacobimatrix (Teil 1): Sei $F : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ eine vektorwertige Funktion, gegeben durch die Identitätsabbildung

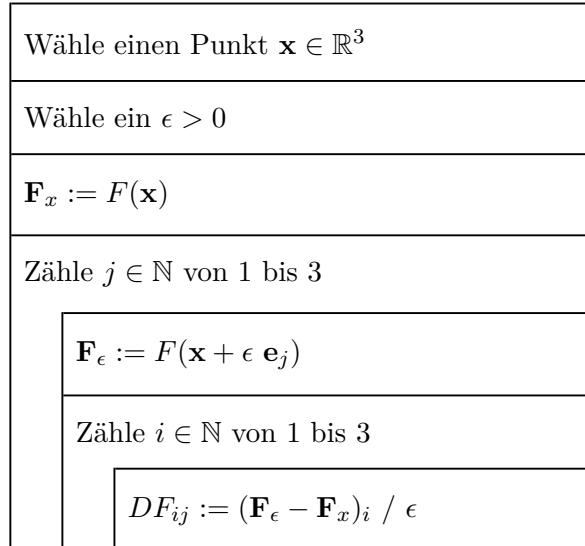
$$F(\mathbf{x}) := \mathbf{x}.$$

$DF(\mathbf{x}) = \left(\frac{\partial F_i}{\partial x_j} \right)$ sei die Jacobimatrix der Funktion $F(\mathbf{x})$ am Punkt \mathbf{x} . Analytisch ergibt sich

$$DF(\mathbf{x}) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Wir betrachten nun einen Algorithmus, basierend auf dem oben beschriebenen Verfahren mit finiten Differenzen.

Algorithmus — „Straight-forward“-Berechnung der Jacobimatrix.



Diesen Algorithmus werten wir nun für $F(\mathbf{x}) := \mathbf{x}$ aus:

$$\epsilon > 0 , \quad \mathbf{F}_x = \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

1) $j = 1$

$$\mathbf{F}_\epsilon = \begin{pmatrix} x_1 + \epsilon \\ x_2 \\ x_3 \end{pmatrix} \Rightarrow \begin{cases} DF_{11} = (x_1 + \epsilon - x_1)/\epsilon = 1 \\ DF_{21} = (x_2 - x_2)/\epsilon = 0 \\ DF_{31} = (x_3 - x_3)/\epsilon = 0 \end{cases}$$

2) $j = 2$

$$\mathbf{F}_\epsilon = \begin{pmatrix} x_1 \\ x_2 + \epsilon \\ x_3 \end{pmatrix} \Rightarrow \begin{cases} DF_{12} &= (x_1 - x_1)/\epsilon = 0 \\ DF_{22} &= (x_2 + \epsilon - x_2)/\epsilon = 1 \\ DF_{32} &= (x_3 - x_3)/\epsilon = 0 \end{cases}$$

3) $j = 3$

$$\mathbf{F}_\epsilon = \begin{pmatrix} x_1 \\ x_2 \\ x_3 + \epsilon \end{pmatrix} \Rightarrow \begin{cases} DF_{13} &= (x_1 - x_1)/\epsilon = 0 \\ DF_{23} &= (x_2 - x_2)/\epsilon = 0 \\ DF_{33} &= (x_3 + \epsilon - x_3)/\epsilon = 1 \end{cases}$$

4.2.1 Ausnutzung der Dünnbesetztheit

Am Beispiel des letzten Kapitels sieht man, dass das vorgeschlagene Verfahren zur Berechnung der Jacobimatrix noch optimiert werden kann. Wäre die Struktur der Jacobimatrix bekannt, könnten mehrere Berechnungen zusammengefasst werden. Das wollen wir wieder am Beispiel des letzten Kapitels verdeutlichen.

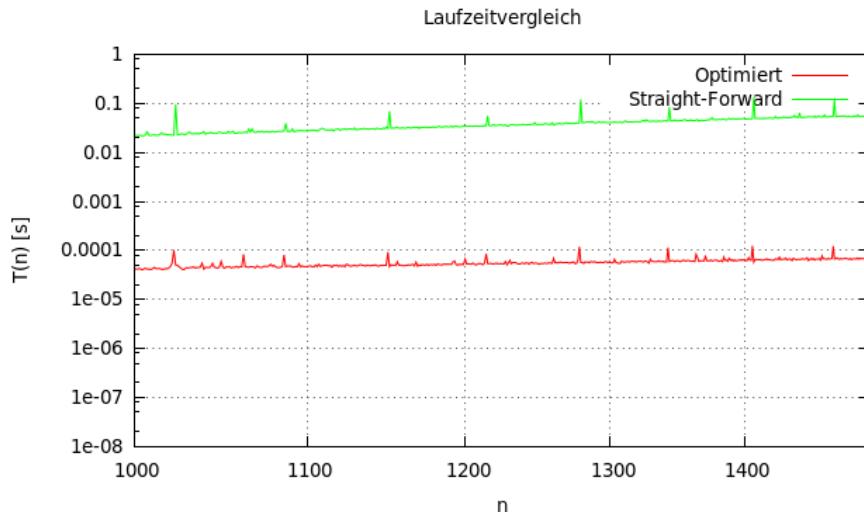


Abbildung 4.1: Vergleich der Laufzeiten

Beispiel zur Jacobimatrix (Teil 2): Die Jacobimatrix der Funktion $F(\mathbf{x}) := \mathbf{x}$ war gegeben durch

$$DF(\mathbf{x}) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

In der Jacobimatrix, wie auch in der Definition von F sieht man, dass die i -te Komponente von F ausschließlich von der i -ten Komponente von \mathbf{x} abhängt.

Das Ergebnis der Ableitung $\frac{\partial F_i}{\partial x_j}$ ist demnach unabhängig davon, welchen Wert die anderen Komponenten mit Index $j \neq i$ von F besitzen. Dadurch ist es möglich alle Komponenten gleichzeitig mit ϵ zu stören um die Diagonaleinträge der Jacobimatrix zu berechnen:

$$\mathbf{F}_\epsilon = \begin{pmatrix} x_1 + \epsilon \\ x_2 + \epsilon \\ x_3 + \epsilon \end{pmatrix} \Rightarrow \begin{cases} DF_{11} &= (x_1 + \epsilon - x_1)/\epsilon = 1 \\ DF_{22} &= (x_2 + \epsilon - x_2)/\epsilon = 1 \\ DF_{33} &= (x_3 + \epsilon - x_3)/\epsilon = 1 \end{cases}$$

Alle anderen Einträge sind gleich null und müssen nicht neu berechnet werden.

4.2.2 Algorithmus: Dünnbesetztheitsstruktur

Die Dünnbesetztheit lässt sich für allgemeine Residuen mittels Operatorüberladung in C++ ermitteln. Dabei wird ein neuer Datentyp *jsp* eingeführt, welcher später zur Berechnung des Residuums verwendet werden soll. Neben einer Double-Variable besitzt dieser Datentyp zusätzlich eine Indexmenge, in welcher sich vor der Berechnung des Residuums eine eindeutige Indexnummer befindet. Operationen auf diesem Datentyp (Addition, Multiplikation, ...) sollen dazu führen, dass sich die Indexmengen der beteiligten *jsp*-Typen vereinigen. So kann die Information über die Dünnbesetztheit der Jacobimatrix aus dem Vergleich von eingehenden und ausgehenden Indizes bestimmt werden.

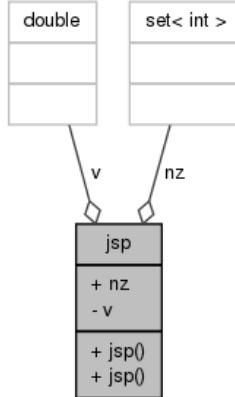


Abbildung 4.2: Klassendiagramm der Klasse *jsp*

Beispiel: Sei $F : \mathbb{R}^3 \mapsto \mathbb{R}^3$ mit

$$F(\mathbf{x}) := \begin{pmatrix} x_0 \sin(x_1) \\ \exp(x_2) \\ x_1 \cos(x_2) \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \end{pmatrix} = \mathbf{y} \quad \text{mit } \mathbf{x}, \mathbf{y} \in \mathbb{R}^3$$

und der Jacobimatrix

$$DF(\mathbf{x}) = \begin{pmatrix} \sin(x_1) & x_0 \cos(x_1) & 0 \\ 0 & 0 & \exp(x_2) \\ 0 & \cos(x_2) & -x_1 \cos(x_2) \end{pmatrix}.$$

Die Nulleinträge dieser Matrix wollen wir nun verifizieren ohne die Ableitung direkt zu berechnen. Zur Untersuchung der Abhängigkeitsstruktur bilden wir einen Eingangsvektor aus *jsp*-Datentypen im Sinne des Klassendiagramms in Abbildung 4.2 und wenden darauf die Funktion F an. Dazu definieren wir etwas abstrakter

$$[x, S] \in jsp \Leftrightarrow x \in \mathbb{R} \text{ und } S \subset (\mathbb{N} \cup \{0\}) .$$

Eine Verknüpfung von *jsp*-Datentypen soll zur Vereinigung ihrer Indexmengen führen. Als Beispiel kann man die Addition wie folgt definieren

$$[x_1, S_1] + [x_2, S_2] := [x_1 + x_2, S_1 \cup S_2] .$$

Die Anwendung von Funktionen, wie der Exponentialfunktion, soll hingegen die Indexmenge unverändert lassen:

$$\exp([x_1, S_1]) := [\exp(x_1), S_1] .$$

Analog kann die Anwendung anderer Grundrechenarten oder Funktionen definiert werden. Als Eingangsvektor wählen wir nun

$$\mathbf{x}_{jsp} = \begin{pmatrix} [x_0, \{0\}] \\ [x_1, \{1\}] \\ [x_2, \{2\}] \end{pmatrix}, \quad \mathbf{x}_{jsp} \in jsp^3$$

und setzen $\mathbf{y}_{jsp} = f(\mathbf{x}_{jsp})$

$$\Rightarrow \mathbf{y}_{jsp} = \begin{pmatrix} [x_0 \sin(x_1), \{0, 1\}] \\ [\exp(x_2), \{2\}] \\ [x_1 \cos(x_2), \{1, 2\}] \end{pmatrix} .$$

Wichtig ist, dass die Elemente des Eingangsvektors eindeutig über ihre Indexmenge indiziert werden bevor man die Funktion F anwendet. Das Ergebnis kann man in einem *Abhängigkeitsdiagramm* analysieren:

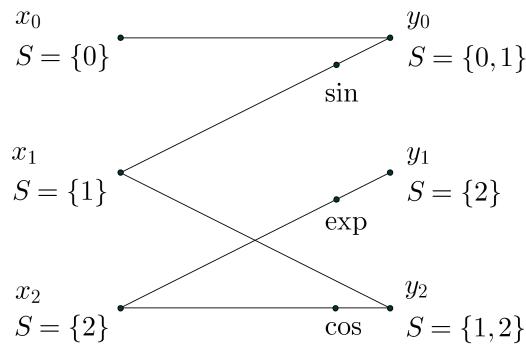


Abbildung 4.3: Abhängigkeitsdiagramm für die Beispiefunktion F.

Die Indexmengen S geben an, von welchen Eingangsparametern der jeweilige

Wert y_i abhängt. So kann man ablesen, welche Einträge der Jacobimatrix immer null sind:

$$\frac{\partial F_0}{\partial x_2} = \frac{\partial F_1}{\partial x_0} = \frac{\partial F_1}{\partial x_1} = \frac{\partial F_2}{\partial x_0} = 0$$

$$\Rightarrow DF = \begin{pmatrix} \bullet & \bullet & 0 \\ 0 & 0 & \bullet \\ 0 & \bullet & \bullet \end{pmatrix}, \text{ wobei } \bullet \neq 0 \text{ i.A.}$$

Das analytische Ergebnis konnte verifiziert werden. Durch die formale Überladung der Grundrechenoperationen, sowie der Funktionen cos, sin und exp hat sich die Indexsignatur jedes einzelnen Parameters ins Ergebnis fortgepflanzt. Die Aufschlüsselung der resultierenden Indexmengen in einem Abhängigkeitsdiagramm liefert die Nulleinträge der Jacobimatrix. Über andere Einträge kann keine allgemeine Aussage getroffen werden.

In der verwendeten Implementierung werden die Nicht-Null-Einträge als Paare aus Zeilen- und Spaltennummer in einem Vektor gespeichert:

```
std :: vector<std :: pair<int , int> > nonzeros ;
```

Der Vektor *nonzeros* ist dabei Teil der Klasse *SparsityPattern*, welche die Dünnbesetztheitsstruktur repräsentiert und später als Objekt der Jacobimatrix zugeordnet werden soll.

4.2.3 Algorithmus: Distanz-1-Färbung

Angenommen man hätte folgende dünnbesetzte Jacobimatrix nach dem oben beschriebenen Verfahren ermittelt:

$$R'_{Bsp} = \begin{pmatrix} \bullet & \bullet & 0 & 0 \\ 0 & \bullet & \bullet & 0 \\ 0 & \bullet & 0 & \bullet \\ 0 & 0 & \bullet & \bullet \end{pmatrix}$$

Die Tatsache, dass die erste und die dritte Spalte keine Nicht-Null-Elemente in der gleichen Zeile haben bedeutet nach der Definition der Jacobimatrix, dass die beiden Spalten nicht von den gleichen Eingangsvariablen abhängen und somit nach den Erkenntnissen aus Kapitel 4.2.1 simultan berechnet werden können. Spalten mit dieser Eigenschaft nennt man *strukturell orthogonal*. Aus dieser Eigenschaft lässt sich ein Konzept entwickeln um die dünnbesetzte Jacobimatrix effektiver zu speichern. Die einzige Information, welche mit einem Nulleintrag verbunden ist, ist seine Position in der Jacobimatrix. Ermittelt man einmal die Dünnbesetztheitsstruktur nach dem Verfahren des vorherigen Kapitels, so hat man diese Information von jedem Nulleintrag. Danach braucht man ihn in gewisser Weise nicht mehr. Für die Berechnung ist er nicht von Relevanz, da sein Wert für alle Eingabewerte bereits bekannt ist. Man kann daher die strukturell orthogonalen Spalten paarweise zu neuen Spalten zusammenfassen ohne Information zu verlieren. Auf dem Weg zu einem geeigneten Algorithmus legt man zunächst einen *Spalteninzidenzgraphen* an. Jede Spalte der Jacobimatrix erhält

in diesem Graphen einen *Knoten*. Zwei Spalten, die beide in gleicher Zeile ein Nicht-Null-Element enthalten, erhalten im Graphen eine Verbindung (*Kante*). Im Beispiel der Matrix R'_{Bsp} ergäbe sich der in Abbildung 4.4 gezeigte Graph.

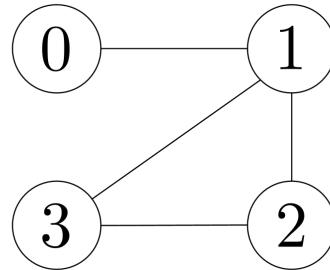


Abbildung 4.4: Beispiel für einen Spalteninzidenzgraphen.

Der Spalteninzidenzgraph zeigt hier, dass es möglich wäre die Spalte mit dem Index 0 und die Spalte mit dem Index 2 zu vereinen¹, da sie sich keine Kante teilen. Mathematisch kann man diese Vereinigung als lineare Abbildung auffassen, welche durch eine Matrix S repräsentiert wird:

$$R'_{Bsp} \cdot S = \begin{pmatrix} \bullet & \bullet & 0 \\ \bullet & \bullet & 0 \\ 0 & \bullet & \bullet \\ \bullet & 0 & \bullet \end{pmatrix}.$$

S bezeichnet man als *Saatmatrix* der Dünnbesetzungstruktur. In diesem einfachen Fall kann man die Saatmatrix direkt angeben:

$$S = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Die Matrix, welche aus der Multiplikation der erhaltenen Saatmatrix S mit der Jacobimatrix R' hervorgeht wird als komprimierte Jacobimatrix CR' bezeichnet:

$$C = R' \cdot S.$$

Ein möglicher Algorithmus zur Bestimmung der Saatmatrix ist die *Distanz-1-Färbung*. Bei dieser Methode erhält jeder Knoten des Spalteninzidenzgraphen eine Farbe (mathematisch dargestellt durch eine Zahl). Zwei durch eine Kante verbundene Knoten sollen dabei unterschiedliche Farben erhalten und es sollen so wenig Farben wie möglich verwendet werden. Haben zwei Knoten nach Abschluss der Färbung die gleiche Farbe, so bedeutet das, dass die entsprechenden Spalten keine Kante im Spalteninzidenzgraphen besitzen. In der Saatmatrix kann daher in den Zeilen einer Spalte eine 1 stehen, die später mit den Spalten der Ausgangsmatrix multipliziert werden, welche die gleiche Farbe besitzen.

¹Es wäre natürlich auch möglich die Spalte mit dem Index 0 und die Spalte mit dem Index 3 zu vereinen.

Die Anzahl der Spalten der Saatmatrix entspricht analog dazu der Anzahl der Farben. Für das Beispiel der Matrix R'_{Bsp} ergibt sich folgende Färbung:

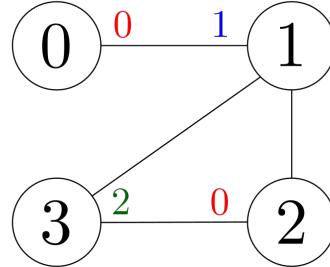


Abbildung 4.5: Distanz-1-Färbung.

Die Färbung lässt sich somit als Indexmenge einer neuen Matrix verstehen. Die ursprüngliche Spaltennummerierung der Matrix R'_{Bsp} wird zur Zeilennummerierung der Saatmatrix. Die Spaltennummierung der Saatmatrix repräsentiert die unterschiedlichen Farben:

$$\begin{array}{l} (0,0) \\ (1,1) \\ (2,0) \\ (3,2) \end{array} \rightarrow \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Ein Variante der Distanz-1-Färbung kann so aussehen:

1. Laufe die Knoten in beliebiger Reihenfolge einmal ab.
2. Gebe dem aktuellen Knoten die kleinstmögliche Farbe (keine Übereinstimmung mit einem benachbarten Knoten).

4.2.4 Zusammenfassung: Berechnung der Jacobimatrix

Gegeben ist die Residuumsfunktion $R : \mathbb{R}^n \mapsto \mathbb{R}^n$ deren Jacobimatrix R' mit dem Verfahren der finiten Differenzen bestimmt werden soll:

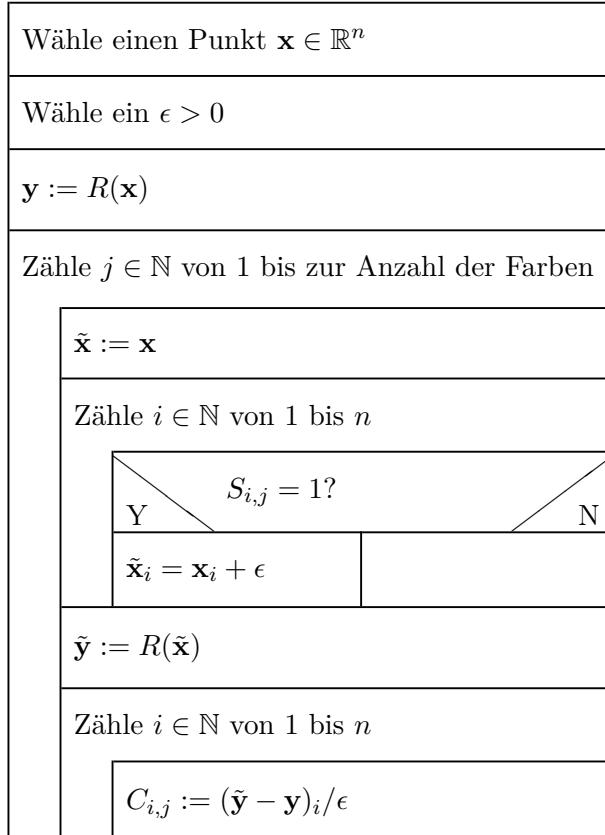
$$R'_{i,j} \approx \frac{R_i(\hat{\mathbf{z}} + \epsilon \cdot \mathbf{e}_j) - R_i(\hat{\mathbf{z}})}{\epsilon}.$$

Bei erstmaliger Berechnung der Jacobimatrix müssen zunächst folgende Schritte durchgeführt werden:

1. Ermittlung der Dünnbesetztheitsstruktur SP .
2. Erstellung des Spalteninzidenzgraphen CIG aus SP .
3. Distanz-1-Färbung des Spalteninzidenzgraphen CIG .
4. Erstellung der Saatmatrix S aus den Farbinformation von CIG .

Die Saatmatrix ist in einer Instanz der Klasse *Jacobian* gespeichert und kann zur erneuten Berechnung der Jacobimatrix an einem anderen Punkt wiederverwendet werden.

Algorithmus — Spaltenweise Berechnung der komprimierten Jacobimatrix



Um die Jacobimatrix aus ihrem komprimierten Abbild zu rekonstruieren, muss wieder auf die Farbinformation der Saatmatrix zurückgegriffen werden. Die komprimierte Jacobimatrix enthält alle Einträge, die auch die ursprüngliche Jacobimatrix beinhaltet. Sie besitzt nur weniger Nullen. Bezeichnet p die Anzahl der Farben und n die Anzahl der Gitterpunkte, so gilt allgemein:

$$R' \in \mathbb{R}^{n \times n} \wedge S \in \mathbb{R}^{n \times p} \Rightarrow C \in \mathbb{R}^{n \times p}.$$

Man kann an folgendem Beispiel einige Beobachtungen machen:

$$\underbrace{\begin{pmatrix} a_{11} & a_{12} & 0 \\ 0 & a_{22} & a_{23} \\ a_{31} & 0 & 0 \end{pmatrix}}_{R'} \cdot \underbrace{\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{pmatrix}}_S = \underbrace{\begin{pmatrix} a_{11} & a_{12} \\ a_{23} & a_{22} \\ a_{31} & 0 \end{pmatrix}}_C.$$

Zum einen stehen hier alle Elemente einer Zeile der Jacobimatrix nach der Komprimierung durch die Saatmatrix wieder in einer Zeile. Zum anderen enthält die Saatmatrix alle Informationen darüber, wo sich einzelne Elemente nach

der Komprimierung befinden. Diese Beobachtungen kann man für einen Algorithmus nutzen, der die ursprüngliche Jacobimatrix aus der komprimierten Jacobimatrix rekonstruiert: Für jeden Nicht-Null-Eintrag der Jacobimatrix (siehe Dünnbesetztheitsstruktur) suche den entsprechenden Wert in der komprimierten Jacobimatrix (mit Hilfe der Saatmatrix) und kopiere ihn.

Beispiel (Rekonstruktion der Jacobimatrix) :

Gegeben sei die Jacobimatrix R' mit der Dünnbesetztheitsstruktur

$$R' = \begin{pmatrix} \bullet & \bullet & 0 & 0 \\ 0 & \bullet & \bullet & 0 \\ 0 & \bullet & 0 & \bullet \\ 0 & 0 & \bullet & \bullet \end{pmatrix}$$

und der Saatmatrix S mit

$$S = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

sodass gilt

$$R' \cdot S = \underbrace{\begin{pmatrix} 7 & 3 & 0 \\ 2 & 9 & 0 \\ 0 & -1 & 5 \\ 4 & 0 & 6 \end{pmatrix}}_{=:C}.$$

Exemplarische Rekonstruktion der Jacobimatrix R' für einzelne Nicht-Null Einträge:

1. Nicht-Null Eintrag: $R_{0,1}$.
Zeilennummer 0 bleibt erhalten.
Spaltennummer = Farbe des Knotens = 1.

$$\Rightarrow R_{0,1} = C_{0,1} = 3$$

2. Nicht-Null Eintrag: $R_{1,1}$.
Zeilennummer 1 bleibt erhalten.
Spaltennummer = Farbe des Knotens = 1.

$$\Rightarrow R_{1,1} = C_{1,1} = 9$$

3. Nicht-Null Eintrag: $R_{1,2}$.
Zeilennummer 1 bleibt erhalten.
Spaltennummer = Farbe des Knotens = 0.

$$\Rightarrow R_{1,2} = C_{1,1} = 2$$

Verfährt man so für alle Nicht-Null Einträge, wird die Jacobimatrix vollständig aus der komprimierten Jacobimatrix rekonstruiert:

$$\Rightarrow R' = \begin{pmatrix} 7 & 3 & 0 & 0 \\ 0 & 9 & 2 & 0 \\ 0 & -1 & 0 & 5 \\ 0 & 0 & 4 & 6 \end{pmatrix}$$

4.3 Berechnung der Lösung des nicht-linearen Gleichungssystems

Wie in Kapitel 2.4 angedeutet wurde, soll das durch $R(\hat{\mathbf{z}}) = 0$ definierte, nicht-lineare Gleichungssystem durch die Verwendung externer Bibliotheken gelöst werden. Die Verwendung externer Bibliotheken hat zum einen den pragmatischen Vorteil, dass man die numerische Lösung nicht selbst implementieren muss. Zum anderen sind vorhandene Implementierungen gut ausgereift und dadurch oft weniger fehleranfällig als eigene Implementierungen. Unser Gleichungssystem $R(\hat{\mathbf{z}}) = 0$ wollen wir im Folgenden mit der *GNU GSL* Bibliothek lösen.

4.3.1 Verwendung der GSL Bibliothek

Die *GSL* Bibliothek ist eine numerische Bibliothek, welche unter GNU Lizenz angeboten wird. Sie ist damit kostenlos erhältlich und ihr Code ist frei einsehbar. Unter anderem verwendet das Framework *ROOT*, welches vom CERN in Genf angeboten wird, einige Algorithmen der *GSL* Bibliothek. Im Netz findet man *GSL* unter folgender Adresse: <http://www.gnu.org/software/gsl/>. Dort befinden sich die Dokumentation der *GSL* Bibliothek, sowie Installationsanleitungen für verschiedene Betriebssysteme. Zur Abkürzung des Installationsprozesses, steht Ihnen ein *Makefile* zur Verfügung. Um dieses zu verwenden, navigieren Sie innerhalb des SVN Repository der Vorlesung zunächst in das Verzeichnis:

```
./downloads/Programmierpraktikum/gsl/
```

Dort führen Sie im Terminal den Befehl *make* aus. *GSL* wird dann in Ihr *home*-Verzeichnis installiert. Auf dem RWTH-Cluster ist dies der Ordner, welcher mit Ihrer TIM-ID bezeichnet ist. Dort besitzen Sie genügend Rechte um *GSL* zu installieren. Um ein C++ Programm zusammen mit der *GSL*-Bibliothek zu kompilieren, müssen dem Compiler und dem Linker mitgeteilt werden, wo die entsprechenden Teile der *GSL*-Bibliothek zu finden sind, welche innerhalb des Programms verwendet werden. Dies geschieht mit den Befehlen:

- \$ g++ -c -DGSL -I/home/gsl/local/include *.cpp
- \$ g++ -L/home/gsl/local/lib *.o -lgsl -lgslcblas

Unter Umständen kann die Angabe dieser Verzeichnisse schief gehen, je nachdem welche Rechte Sie in Ihrem Betriebssystem besitzen. Dann bleibt nur herauszufinden wo sich der `gsl` Ordner und die jeweiligen Unterverzeichnisse `include` und `lib` befinden, um dann jeweils die korrigierte Verzeichnisangabe zum Komplizieren und Linken zu verwenden. Es ist bereits vorgekommen, dass je nach Version von `g++` oder des Betriebssystems der Error:

```
error while loading shared libraries
```

auftritt. Ist das der Fall, so muss man zusätzlich die `-static` flag zum Linken verwenden.

Wenden wir uns jetzt den Teilen der GSL Bibliothek zu, welche wir zur Lösung unseres nicht-linearen Gleichungssystems benötigen. Zunächst binden wir den Teil der GSL Bibliothek ein, welcher die gewünschten Algorithmen bereitstellt und definieren, nach welchem Lösungsverfahren vorgegangen werden soll:

```
#include <gsl/gsl_multiroots.h>;
...
const gsl_multiroot_fdfsolver_type *T;
T = gsl_multiroot_fdfsolver_newton;
```

Damit wurde festgelegt, dass GSL das Newtonverfahren verwendet werden soll. Dieser Löser erwartet unter anderem einen ersten Schätzwert zum Starten der Iteration. Der Schätzwert soll als `gsl`-Vektor Datentyp vorliegen, dessen Dokumentation sich ebenfalls auf der GSL Homepage befindet. Man kann einen `gsl`-Vektor der Länge `dof`² auf folgende Weise erzeugen und in jedem Eintrag den Wert 1 speichern:

```
gsl_vector *z_gsl = gsl_vector_alloc (dof);
for (int i=1;i<dof; i++){
    gsl_vector_set (z_gsl, i, 1.0);
}
```

Anschließend allozieren wir eine Instanz des Lösen:

```
gsl_multiroot_fdfsolver *s_gsl;
s_gsl = gsl_multiroot_fdfsolver_alloc (T, dof);
```

Die mehrdimensionale Funktion, deren Nullstelle bestimmt werden soll, muss auch in einem speziellen Datenformat vorliegen:

```
gsl_multiroot_function_fdf f={&f1,&df1,&fdf1,dof,NULL};
```

Dabei ist `&f1` die Adresse der Funktion, welche das Residuum bereitstellt, während die Adresse `&df1` auf eine Routine zur Berechnung der Jacobimatrix verweist. In `&fdf1` werden sowohl das Residuum, als auch die Jacobimatrix berechnet. Diese Funktionen sind in der Datei

`gsl_multiroot_function_fdf.h`

²Degrees of freedom; Die Dimension des Residuums.

implementiert und werden auf dem Übungsblatt C dieses Programmierpraktikums thematisiert. Neben diesen Funktionen wird wieder die Dimension des Residuums übergeben, sowie der void Pointer *NULL*, welcher dazu verwendet werden könnte um der Residuumsfunktion weitere Parameter hinzuzufügen. Mit dem Befehl *set* wird dem Löser *s_gsl* mitgeteilt, dass er die Nullstelle der *gsl*-Funktion *f* mit der Adresse *&f* für den Startwert *z_gsl* suchen soll:

```
gsl_multiroot_fdfsolver_set (s_gsl, &f, z_gsl);
```

Der *iterate* Befehl veranlasst GSL dazu, einen Iterationsschritt vorwärts zu gehen.

```
gsl_multiroot_fdfsolver_iterate (s_gsl);
```

Die Function *calc_residual* der Klasse *Discretization* ist so implementiert, dass sie in jedem Aufruf den von GSL vorgeschlagenen Lösungsvektor auf das zweidimensionale Array *omega_H* abbildet. Weitere Details zur Anwendung des GSL Lösen findet sich im Kapitel *Multidimensional Root Finding* der GSL Dokumentation.

4.4 Visualisierung

4.4.1 Datenformat

Der letzte Teil des Programmierpraktikums befasst sich mit der Visualisierung der Lösung. Diese liegt nach Abschluss der GSL Iteration als zweidimensionales Array in einem Objekt der Klasse *Discretization* vor:

```
double** omega_H;
```

Der Zugriff auf einzelne Elemente kann innerhalb des *Discretization* Objekts mit eckigen Klammern erfolgen. Beispielsweise zur Ausgabe auf dem Bildschirm:

```
std::cout << omega_H[ i ][ j ] << std::endl;
```

In diesem Fall wird die Lösung am Punkt (i, j) des Lösungsgebietes ausgegeben, welches wie in Kapitel 2.2 diskretisiert wurde. Die Umrechnung der Indizes in *x* und *y* Koordinaten erfolgt damit ebenfalls analog:

$$(\hat{x}_i, \hat{y}_j) := (i \cdot h, j \cdot h) \text{ mit } i, j = 1, \dots, s.$$

4.4.2 gnuplot

Gnuplot ist eine frei verfügbare Software zur Visualisierung wissenschaftlicher Daten und Funktion. Es kann unter Linux im Terminal mittels

```
$ sudo apt-get install gnuplot
```

installiert werden. Auf dem RWTH Cluster ist bereits eine Version installiert. Das Programm startet über den Kommandozeilenaufruf

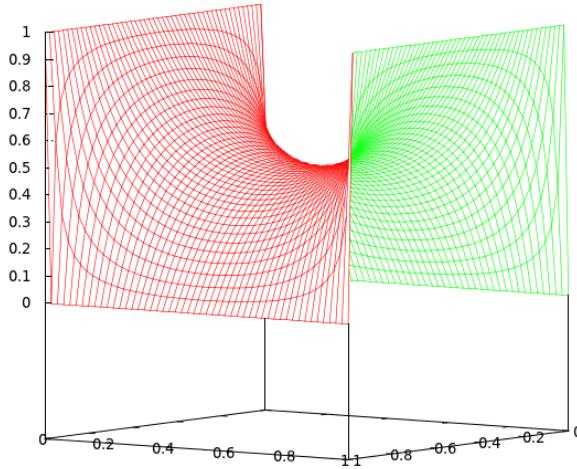


Abbildung 4.6: Lösung des Beispielproblems aus Kapitel 3.5 in Gnuplot.

```
$ gnuplot
```

Nach dem Start von Gnuplot können Befehle über die Kommandozeile gegeben werden. Von Gnuplot selbst werden dabei einige mathematische Funktionen bereitgestellt. Beispielsweise gibt der Befehl

```
gnuplot> plot sin(x)
```

die Sinus-Funktion auf dem Bildschirm aus.

Beispiel (Sinc-Funktion in Gnuplot) :

Die Sinc-Funktion ist definiert durch:

$$\text{sinc}(x) := \frac{\sin(x)}{x} \quad \forall x \in \mathbb{R} .$$

Unter anderem kann die Intensitätsverteilung bei Beugung von Licht an einem Einzelpunkt durch diese Funktion beschrieben werden. Auch für rechteckige Lochblenden hat man es mit dieser Funktion zu tun. Für diesen Fall erhält man einen Term der Form

$$I(x, y) = I_0 \cdot \text{sinc}(x)^2 \cdot \text{sinc}(y)^2 ,$$

wobei die Argumente bereits so skaliert wurden, dass sie dimensionslos sind. Um eine solche Funktion ansprechend darzustellen, benötigt man in gnuplot eine Reihe von Befehlen. Gnuplot bietet die Möglichkeit diese in einer Skriptdatei der Endung **.gp* zusammengefasst abzuspeichern und gemeinsam aufzurufen. So könnte man folgende Befehlskette in einer Datei *sinc.gp* ablegen:

```
#Definition der Sinc-Funktion
```

```

sinc(x)=sin(x)/x

#Wertebereich
set xrange [-10:10]
set yrange [-10:10]

#Auflösung des Gebietes
set isosamples 100

#Projektion der 3D Fläche auf x-y Ebene
set pm3d map
set samples 10**3

#Achsenbeschriftung
set xlabel 'x'
set ylabel 'y'
set cblabel 'Intensity I/I0'

#Plotten
splot sinc(x)**2 * sinc(y)**2
pause -1

```

Die Datei kann anschließend im Terminal via `$ gnuplot sinc.gp` geladen werden. Folgende Grafik wird ausgegeben:

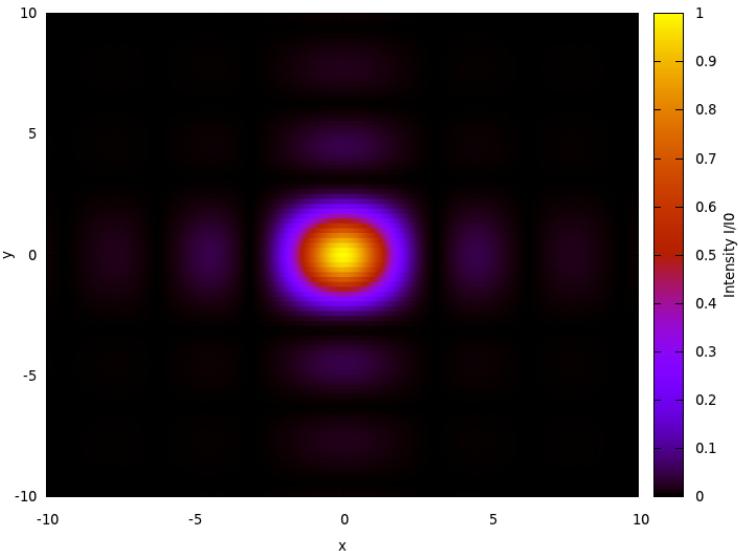


Abbildung 4.7: Beugung an einer quadratischer Lochblende.

Das einfachste Datenformat zur Verwendung in Gnuplot ist das `*.csv` Format. Zur Ausgabe unserer Lösung trennen wir dabei einzelne Punkte durch einen Zeilenumbruch und die einzelnen Koordinaten durch Tabulator:

```
output << x << "\t" << y << "\t" << z << std :: endl;
```

In diesem Format legen wir alle Datenpunkte in einer Datei `Discretization.csv` ab. Die diskreten Datenpunkte unserer Lösung des Minimalflächenproblems werden von Gnuplot so noch nicht als zusammenhängende Fläche dargestellt. Gnuplot benötigt dazu noch weitere Informationen. Etwa die Information, wie viele Randpunkte s pro Raumrichtung diskretisiert wurden. Der Befehl

```
set dgrid3d s,s
```

legt ein äquidistantes $s \times s$ -Gitter über die Datenpunkte. Die Kommandozeilenangabe

```
splot 'Discretization.csv' with lines
```

verbindet die Punkte dann entsprechend dieses Gitters und plottet das Ergebnis. Um verborgene Linien des Plots auszublenden kann die Option

```
set hidden3d
```

verwendet werden. Abbildung 4.6 zeigt die Ausgabe der Lösung für $s = 50$ in Gnuplot. Im Internet findet man die gesamte Gnuplot Dokumentation unter

<http://www.gnuplot.info/>.

Ein sehr empfehlenswerter Einführungskurs in Gnuplot wird von der FU Berlin angeboten unter

<http://userpage.fu-berlin.de/~voelker/gnuplotkurs/gnuplotkurs.html>.

4.4.3 Paraview

Paraview ist genau wie Gnuplot eine *OpenSource* -Software. Es wird jedoch nicht über die Kommandozeile gesteuert, sondern besitzt eine grafische Benutzeroberfläche. Unter Linux kann man Paraview im Terminal mittels

```
$ sudo apt-get install paraview
```

installieren. Grundgerüst dieses Programms ist das *Visualization Toolkit* (VTK). Das ist eine C++ Bibliothek für 3D Computergrafiken. Obwohl Daten auch im csv-Format in Paraview eingelesen werden können, ist es zweckmäßiger das VTK-eigene Datenformat `*.vtk` zu verwenden, da Paraview ähnlich wie Gnuplot

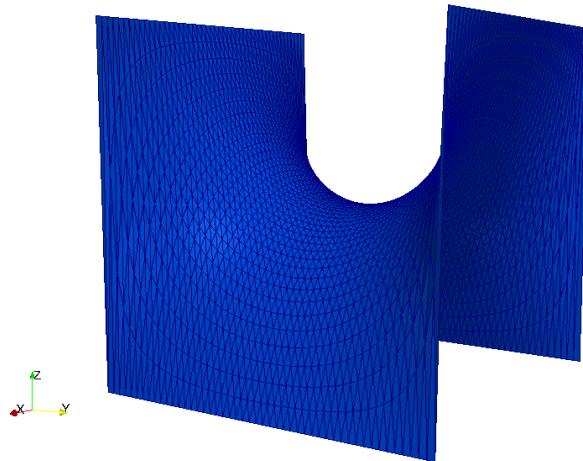


Abbildung 4.8: Lösung des Beispielproblems aus Kapitel 3.5 in Paraview.

nicht weiß, welche Struktur die Datenpunkte der csv-Datei besitzen. Wir verwenden die VTK Datenstruktur *Polydata*. Im Prinzip funktioniert es so, dass jeder Punkt einzeln in Paraview eingelesen wird und von Paraview automatisch eine Indizierung zugewiesen bekommt. Diese Indizierung kann man verwenden um Punkte zu *Polygonen* (Vielecken) zu verbinden. Eine vtk-Datei, welche das quadratische Lösungsgebiet mit $n = s \cdot s$ Punkten mit Dreiecken bedeckt kann so aussehen:

```
# vtk DataFile Version 3.0
vtk output
ASCII
DATASET POLYDATA
POINTS n double
x1 y1 z1
x2 y2 z2
...
xn yn zn
POLYGONS 2(s - 1)(s - 1) 4 · 2(s - 1)(s - 1)
3 0 s s+1
3 1 s+1 s+2
...
```

$2(s - 1)(s - 1)$ ist dabei die Anzahl der Polygone. Die Zeile „3 0 s s+1“ gibt an, dass die Punkte mit dem Index 0, s und s+1 durch ein Dreieck verbunden werden sollen. Dass es sich um ein Dreieck handelt, wird Paraview mit der vorangestellten 3 mitgeteilt. Dadurch benötigt man vier Zahlen zur Definition eines Dreieckes, wodurch sich die Anzahl der Zahlen, welche zur Auflistung aller Polygone benötigt werden zu $4 \cdot 2(s - 1)(s - 1)$ ergibt. Für komplizierte Geometrien kann es sehr aufwändig werden einen Algorithmus zu finden, welcher das Gebiet mit Polygonen bedeckt. In diesem Fall kann man auf Klassen der VTK

C++ Bibliothek zurückgreifen, in der solche Algorithmen in Form von *Filtern* zur Verfügung stehen. Man kann diese auch manuell in Paraview aufrufen. Im Internet findet man die gesamte Dokumentation für Paraview und VTK unter:

- <http://www.paraview.org/>
- <http://www.vtk.org/>

Anhang A

Referenzproblem

Zur Validierung der verwendeten Implementierung ist es nützlich ein *Referenzproblem* zu untersuchen, dessen analytische Lösung bekannt ist. Für das Problem der Minimalflächen hat der Mathematiker Heinrich Ferdinand Scherk im 19. Jahrhundert eine analytische Lösung bestimmt. Er ging dabei von folgendem Ansatz aus:

$$z(x, y) := g(x) + h(y) .$$

Setzt man diesen Ansatz in die Lagrangesche Minimalflächengleichung ein erhält man:

$$\begin{aligned} & \left(1 + g'(x)^2\right) h''(y) + \left(1 + h'(x)^2\right) g''(x) = 0 \\ \Leftrightarrow & -\frac{g''(x)}{1 + g'(x)^2} = \frac{h''(y)}{1 + h'(y)^2} . \end{aligned}$$

Es ergeben sich so entkoppelte Differentialgleichungen für $g(x)$ und $h(y)$ mit $c \in \mathbb{R}$:

$$\begin{aligned} -\frac{g''(x)}{1 + g'(x)^2} = c & \Rightarrow g''(x) + c \cdot (1 + g'(x)^2) = 0 , \\ \frac{h''(y)}{1 + h'(y)^2} = c & \Rightarrow h''(y) - c \cdot (1 + h'(y)^2) = 0 . \end{aligned}$$

Eine eindeutige Lösung ergibt sich mit folgenden Anfangsbedingungen:

$$g(0) = h(0) = g'(0) = h'(0) = 0 .$$

Wie man durch Einsetzen leicht nachprüfen kann lautet diese:

$$\begin{aligned} \Rightarrow g(x) &= \frac{1}{c} \log(\cos(c \cdot x)) , \\ \Rightarrow h(y) &= -\frac{1}{c} \log(\cos(c \cdot y)) . \end{aligned}$$

Um die Implementierung zu testen, kann man die Randwerte des Lösungsgebiets entsprechend dieser Funktionen setzen und dann die numerische Lösung mit der analytischen Vergleichen. Abbildung A.1 zeigt eine Visualisierung der Scherk-Minimalfläche mit Parameter $c = 1$ auf dem Einheitsquadrat um den Nullpunkt.

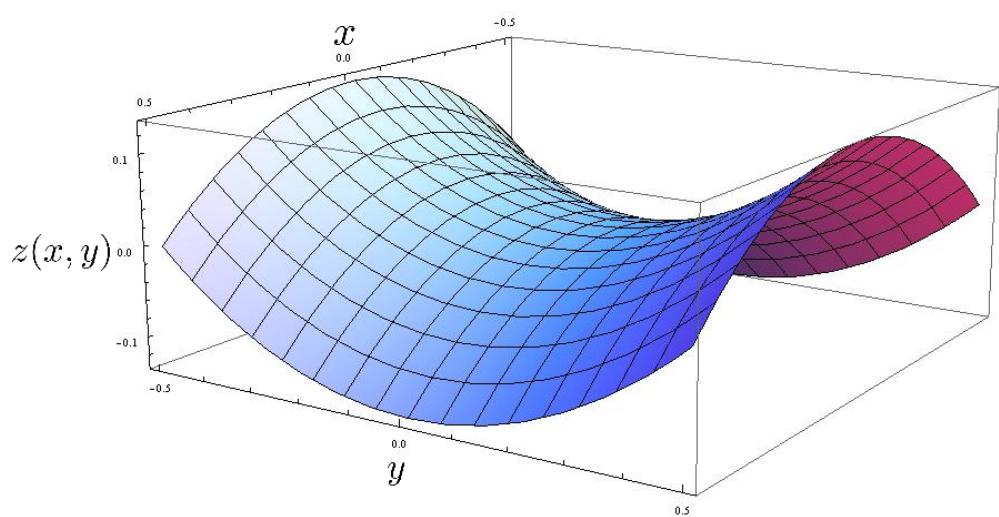


Abbildung A.1: Scherk Minimalfläche mit $c = 1$.

Anhang B

Datenstruktur des Residuum

Für eine spätere Darstellung der Lösung soll jeder Knoten des diskretisierten Gebietes abgespeichert werden. Also auch alle Randwerte, welche nicht als Variablen an die Löser (GSL) übergeben werden dürfen, da dieser nicht zwischen Randwerten und Werten innerhalb des Gebietes unterscheiden kann. Da die Lösung als Matrix darstellbar ist eignet sich die Implementierung von \mathcal{Z} als zweidimensionales dynamisches Array. Um es in der Bezeichnung vom Vektor $\hat{\mathbf{z}}$ abzugrenzen wird in der Implementierung die Bezeichnung *omega_H* für die Matrixdarstellung der Lösung verwendet:

```
double** omega_H=double*[ s ] ;
for( int i=0;i<s ; i++){
    omega_H[ i ]=new double[ s ] ;
}
for( int i=0;i<s ; i++)
    for( int k=0;k<s ; i++)
        omega_H[ i ][ k ]=value ;
```

Da wir uns die Möglichkeit nicht nehmen lassen wollen beliebige Punkte der Diskretisierung mit einem festen Wert zu belegen, müssen wir sicherstellen, dass diese Werte nicht an die externe Bibliothek übergeben werden. Folgende Vorgehensweise wäre denkbar:

- Speichere die Information über die Existenz eines Randwertes in einem dynamischen zweidimensionalen Array.
- Zähle wie viele Randwerte es gibt und speichere den Werte.
- Erzeuge einen Rückgabevektor.
- Setze einen Zeiger auf das erste Element des Rückgabevektors.
- Durchlaufe Ω_H . Falls der aktuelle Punkt ein Randpunkt ist, ignoriere ihn. Ansonsten speichere seinen Wert in der Speicherzelle, auf die der Zeiger aktuell verweist und inkrementiere den Zeiger.

Analog lässt sich die entgegengesetzte Richtung realisieren. *Set*-Routinen bilden den Eingabevektor $\hat{\mathbf{z}}$ auf die in *omega_H* gespeicherte Matrix \mathcal{Z} ab, *Get*-Routinen stellen die Umkehrabbildung dar. Es muss dabei jeweils unterschieden

werden, von welcher Lösungsbibliothek auf die Diskretisierung zugegriffen werden soll, da beispielsweise die GSL-Bibliothek einen speziellen eigenen Datentyp benötigt. Das Klassendiagramm ist in Abbildung B.1 dargestellt.

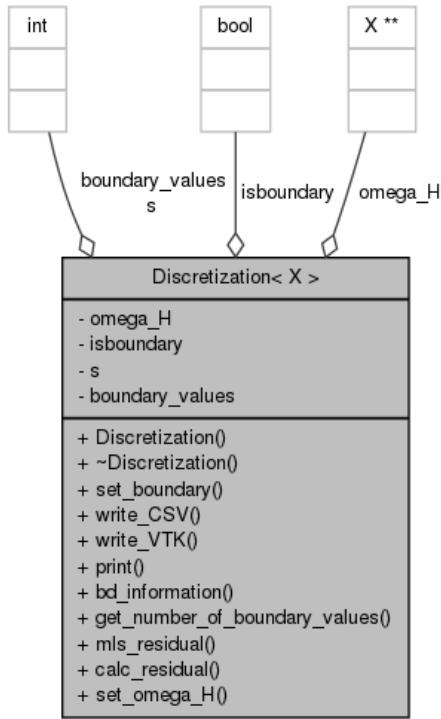


Abbildung B.1: Klasse *Discretization.h*