Wir konstruieren eine Wasserrutsche!

Teilnehmer:

Leo Graumann	Heinrich-Hertz-Oberschule, Berlin
Anh Vu Ho	Andreas-Oberschule, Berlin
Yiyang Huang	Herder-Oberschule, Berlin
Felix Jäger	Lise-Meitner-Gymnasium, Leverkusen
Charlotte Kappler	Herder-Oberschule, Berlin
Wilhelm Mebus	Andreas-Oberschule, Berlin
Alice Wamser	Immanuel-Kant-Oberschule, Berlin

Gruppenleiter:

René Lamour	Humboldt-Universität zu Berlin Mitglied im DFG-Forschungszentrum MATHEON *Mathematik für Schlüsseltechnologien*
Caren Tischendorf	Humboldt-Universität zu Berlin Mitglied im DFG-Forschungszentrum MATHEON *Mathematik für Schlüsseltechnologien*

Wie konstruiert man eine Rutsche über zwei Findlinge hinweg, so dass die Kinder Spaß haben und sich dabei nicht verletzen?

Wir haben eine geeignete Funktion gesucht, die durch gegebene Punkte im Raum verläuft. Diese Aufgabe nennt man Interpolation. Wir untersuchten verschiedene Arten von Interpolation, lernten Wege zur Berechnung solcher Funktionen kennen und programmierten die entwickelten Algorithmen.

Am Ende testeten wir, welche Lösung am besten für unsere Rutsche geeignet ist.

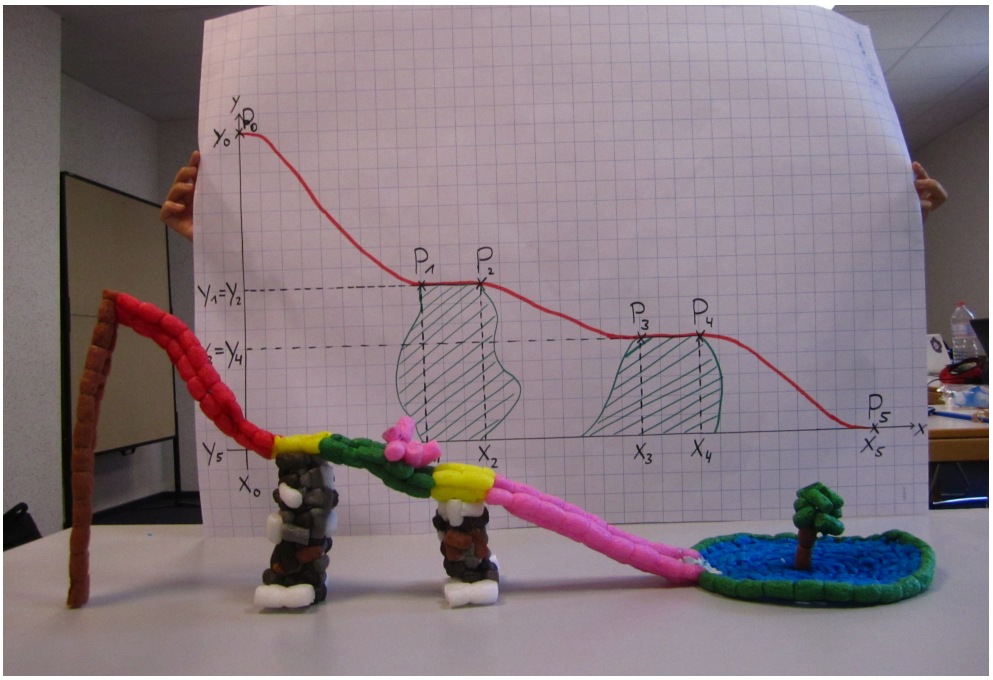
1 Einleitung

1.1 Aufgabe

Es soll eine Funktion gefunden werden, deren Graph eine ideale Wasserrutsche beschreibt. Dabei soll die Rutsche über zwei Felsen verlaufen. Zusätzlich zum Aufstellen der Funktionsgleichung soll der Graph mit PYTHON erzeugt werden.

1.2 Überlegungen

Der Graph unserer gesuchten Funktion soll durch sechs festgelegte Punkte gehen. Darunter sind die Punkte am Start und Ende der Rutsche $P_0 = (x_0, y_0)$ und $P_5 = (x_5, y_5)$. Dazwischen liegen die Schnittpunkte mit den fiktiven Felsen. Sowohl $P_1 = (x_1, y_2)$ und $P_2 = (x_2, y_2)$, wobei gilt: $y_1 = y_2$, als auch $P_3 = (x_3, y_3)$ und $P_4 = (x_4, y_4)$, wobei auch hier gilt: $y_3 = y_4$.

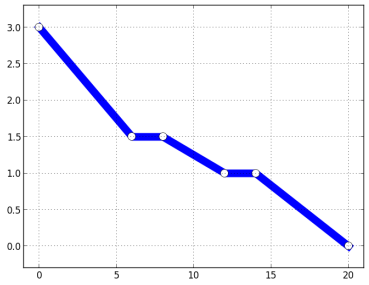


Gesucht ist nun eine Funktion $f(x)$ mit $f(x_i) = y_i \ \forall i = 0, \ldots, 5$.

2 Polynominterpolation

2.1 Idee 1: Lineare Interpolation

Am einfachsten erscheint es, die Punkte linear zu verbinden:



Dies ist jedoch unangenehm zu rutschen und es besteht eine hohe Verletzungsgefahr an den Knicken. ⇒ Funktion ohne Ecken gesucht!

2.2 Idee 2: Polynom 5.Grades

Wir können ein Polynom dafür verwenden. Es muss 5. Grades sein, da wir 6 feste Punkte verbinden wollen. Um die Polynomfunktion zu finden, gibt es folgende Möglichkeiten:

a) lineares Gleichungssystem:

$$y = a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$$

Die Koeffizienten des Polynoms können über ein lineares Gleichungssystem ausgerechnet werden, weil wir für jeden Punkt P_i die entsprechenden Koordinaten (x_i, y_i) einsetzen können, damit sechs Gleichungen aufstellen können, und es auch genau sechs Unbekannte gibt:

$$\begin{aligned} y_0 &= a_5x_0^5 + a_4x_0^4 + a_3x_0^3 + a_2x_0^2 + a_1x_0 + a_0 \\ \vdots &= \vdots \\ y_5 &= a_5x_5^5 + a_4x_5^4 + a_3x_5^3 + a_2x_5^2 + a_1x_5 + a_0 \end{aligned}$$

Es gibt jedoch eine einfachere Methode, das Polynom auszurechnen:

b) Lagrange-Polynom:
Die Lagrange-Basis-Polynome sehen folgendermaßen aus:

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{5} \frac{x - x_j}{x_i - x_j} \qquad \forall i = 0, \ldots, 5.$$

Wir haben bei $x = x_i$:

$$L_i(x_i) = \prod_{\substack{j=0 \\ j \neq i}}^{5} \frac{x_i - x_j}{x_i - x_j} = \prod_{\substack{j=0 \\ j \neq i}}^{5} 1 = 1$$

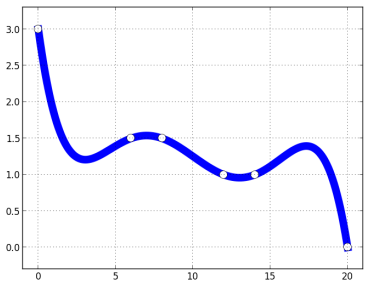
und bei $x = x_k$ mit $k \neq i$ und $0 \leq k \leq 5$:

$$L_i(x_k) = \prod_{\substack{j=0 \\ j \neq i}}^{5} \frac{x_k - x_j}{x_i - x_j} = 0.$$

Das eigentliche Lagrange-Polynom hat diese Funktionsvorschrift:

$$f(x) = \sum_{i=0}^{5} y_i L_i(x).$$

Da dadurch $f(x_k) = \sum_{i=0}^{5} y_i L_i(x_k) = y_k$, so geht das Polynom auch durch die gewünschten Punkte. So sieht die Funktion dann aus:



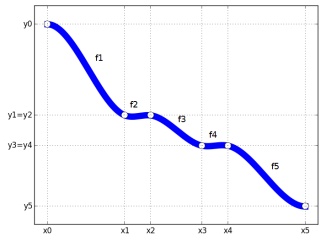
Sie ist aber extrem gewellt. Das Kind könnte steckenbleiben oder geschleudert werden. Ein normales Polynom ist demnach auch ungünstig.

3 Splinefunktion

Die Polynomfunktion 5. Grades aus dem vorherigen Kapitel erfüllt offensichtlich immer noch nicht die Standards einer geeigneten Wasserrutsche.
Deshalb ist ein neuer Ansatz erforderlich: eine Splinefunktion. Eine Splinefunktion ist eine zusammengesetzte Funktion aus Polynomen vom Grad n.

3.1 Bedingungen

Im Fall der Wasserrutsche mit den 6 vorgegebenen Punkten $P_0, \ldots, P_5$ suchen wir die 5 Polynomfunktionen $f_1, \ldots, f_5$:



Die Splinefunktion soll nun auch verschiedene Bedingungen erfüllen. Zuerst einmal muss sie stetig durch $P_0, \ldots, P_5$ verlaufen:

$$\begin{aligned} f_1(x_0) &= y_0 \\ f_i(x_i) &= y_i && \forall i = 1, \ldots, 5 \\ f_i(x_i) &= f_{i+1}(x_i) && \forall i = 1, \ldots, 4. \end{aligned}$$

Außerdem soll die Wasserrutsche keinen Knick aufweisen, d.h. die 1. Ableitung muss stetig sein:

$$f_i'(x_i) = f_{i+1}'(x_i) \qquad \forall i = 1, \ldots, 4.$$

Schließlich garantiert die Stetigkeit der Krümmung besonderen Rutschspaß:

$$f_i''(x_i) = f_{i+1}''(x_i) \qquad \forall i = 1, \ldots, 4.$$

Insgesamt haben wir also $6 + 4 + 4 + 4 = 18$ Bedingungen, die die Splinefunktion erfüllen muss.

3.2 Grad der Polynome

Die einzelnen Polynome sind vom Grad n und haben somit den Freiheitsgrad $n+1$. Das bedeutet, dass das Polynom durch $n + 1$ Bedingungen eindeutig festgelegt wird. Insgesamt erhalten durch die 5 Polynome den Freiheitsgrad $5n + 5$ für die Splinefunktion.
Um die 18 Bedingungen erfüllen zu können, muss nun $5n + 5 \geq 18$ gelten. Das kleinste n, das diese Ungleichung erfüllt ist $n = 3$. Damit haben die einzelnen Polynome den Grad 3 und es handelt sich um eine kubische Splinefunktion mit dem Freiheitsgrad $5 \cdot 3 + 5 = 20$. Es können also sogar noch 2 weitere Bedingungen willkürlich gewählt werden. Wir setzen:

$$f_1'(x_0) = v, \qquad f_5'(x_5) = w.$$

Nun haben wir ein Gleichungssystem mit 20 Gleichungen und 20 Variablen, den Koeffizienten der 5 Polynomfunktionen.

4 Das Gleichungssystem

Wir nehmen die folgende Form für die fünf kubischen Funktionen zwischen den Stützstellen:

$$\begin{aligned} f_i(x) &= a_i(x-x_i)^3 + b_i(x-x_i)^2 + c_i(x-x_i) + d_i && (4.1)\\ f_i'(x) &= 3a_i(x-x_i)^2 + 2b_i(x-x_i) + c_i && (4.2)\\ f_i''(x) &= 6a_i(x-x_i) + 2b_i. && (4.3) \end{aligned}$$

Wir haben also jetzt 20 zu bestimmende Koeffizienten. Diese leiten wir mit den Bedingungen her.
Setzen wir in die Bedingungen, die die Stetigkeit der Funktionen, deren 1. und 2. Ableitung garantieren, $x = x_i$ ein, so erhält man $\forall\, i = 1, \ldots, 4$

$$\begin{aligned} f_i(x_i) &= f_{i+1}(x_i) && \Rightarrow d_i = a_{i+1}(x_i - x_{i+1})^3 + \ldots + d_{i+1} && (4.4)\\ f_i'(x_i) &= f_{i+1}'(x_i) && \Rightarrow c_i = 3a_{i+1}(x_i - x_{i+1})^2 + \ldots + c_{i+1} && (4.5)\\ f_i''(x_i) &= f_{i+1}''(x_i) && \Rightarrow 2b_i = 6a_{i+1}(x_i - x_{i+1}) + 2b_{i+1}. && (4.6) \end{aligned}$$

In die Gleichung (4.1) setzt man für x x_i ein und man erhält für alle $i = 1, \ldots, 5$ $d_i = f_i(x_i) = y_i$.
Für die weitere Behandlung definiert man sich neue Variablen σ_i für $i = 1, \ldots, 5$ wie folgt: $\sigma_i := 2b_i \Rightarrow b_i = \frac{1}{2}\sigma_i$. Setzen wir nun in Gleichung (4.6) die neuen Variablen ein, so erhalten wir

$$2\sigma_i = 6a_{i+1}(x_i - x_{i+1}) + 2\sigma_{i+1} \quad \Rightarrow a_{i+1} = \frac{\sigma_i - \sigma_{i+1}}{6(x_i - x_{i+1})}.$$

Nun stellen wir die Gleichung (4.4) nach c_i um:

$$\begin{aligned} c_{i+1} &= \frac{d_i - d_{i+1} - a_{i+1}(x_i - x_{i+1})^3 - b_{i+1}(x_i - x_{i+1})^2}{x_i - x_{i+1}}\\ &= \frac{(y_i - y_{i+1}) - \frac{1}{6}(\sigma_i - \sigma_{i+1})(x_i - x_{i+1})^2 - \frac{1}{2}\sigma_{i+1}(x_i - x_{i+1})^2}{x_i - x_{i+1}}\\ &= \frac{y_{i+1} - y_i}{x_{i+1} - x_i} + (x_{i+1} - x_i)\left(\frac{1}{6}\sigma_i + \frac{1}{3}\sigma_{i+1}\right). \end{aligned}$$

Nun haben wir alle Koeffizienten in Abhängigkeit von σ_i für $i = 1, \ldots, 5$ bis auf a_1 und c_1. Um die Konsistenz der c_i zu bewahren, führen wir noch ein σ_0 ein, so dass $c_1 = \frac{y_1 - y_0}{x_1 - x_0} + (x_1 - x_0)\left(\frac{1}{6}\sigma_0 + \frac{1}{3}\sigma_1\right)$. Nun braucht man noch a_1. Dies bekommt man durch die Betrachtung der restlichen Bedingungen $f_1(x_0) = y_0, f_1'(x_0) = v$ und $f_5'(x_5) = w$.

$$\begin{aligned} a_1(x_0 - x_1)^3 + b_1(x_0 - x_1)^2 + c_1(x_0 - x_1) + d_1 &= y_0 && (4.7)\\ 3a_1(x_0 - x_1)^2 + 2b_1(x_0 - x_1) + c_1 &= v && (4.8)\\ c_5 &= w && (4.9) \end{aligned}$$

In (4.8) werden b_1 und c_1 eingesetzt, so dass

$$a_1 = \frac{v - \frac{y_1 - y_0}{x_1 - x_0} - (x_1 - x_0)\left(\frac{1}{6}\sigma_0 - \frac{2}{3}\sigma_1\right)}{3(x_1 - x_0)^2}.$$

Nun müssen wir noch Gleichungen für die 6 Sigmas finden. Setzen wir die Ausdrücke für a_i, b_i, c_i in (4.5) ein, so erhält man

$$\sigma_{i-1}\frac{1}{6}(x_i - x_{i-1}) + \sigma_i\frac{1}{3}(x_{i+1} - x_{x-1}) + \sigma_{i+1}\frac{1}{6}(x_{i+1} - x_i) = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} - \frac{y_i - y_{i-1}}{x_i - x_{i-1}} =: r_i.$$

Schließlich benötigen wir 2 weitere Gleichungen für die σ_i. Durch Einsetzen von a_1, b_1, c_1 und d_1 in (4.7) ergibt sich:

$$\frac{y_1 - y_0}{x_1 - x_0} + (x_1 - x_0)\left(\frac{1}{6}\sigma_0 + \frac{1}{3}\sigma_1\right) = 0.$$

Und aus (4.9) bekommt man durch Einsetzen von c_5 eine weitere Gleichung. Wir erhalten schließlich die geforderten 6 Gleichungen für die Sigmas.

$$\begin{aligned}
\sigma_0\frac{1}{3}(x_1 - x_0) + \sigma_1\frac{1}{6}(x_1 - x_0) &= \frac{y_1 - y_0}{x_1 - x_0} - v \\
\sigma_{i-1}\frac{1}{6}(x_i - x_{i-1}) + \sigma_i\frac{1}{3}(x_{i+1} - x_{x-1}) + \sigma_{i+1}\frac{1}{6}(x_{i+1} - x_i) &= z_i \quad \forall\, 1, \ldots, 4 \\
\sigma_4\frac{1}{6}(x_1 - x_0) + \sigma_5\frac{1}{3}(x_1 - x_0) &= w - \frac{y_5 - y_4}{x_5 - x_4}.
\end{aligned}$$

Diese können in die folgende Matrixschreibweise überführt werden.

$$\begin{pmatrix}
\frac{1}{3}(x_1-x_0) & \frac{1}{6}(x_1-x_0) & 0 & 0 & 0 & 0 \\
\frac{1}{6}(x_1-x_0) & \frac{1}{3}(x_2-x_0) & \frac{1}{6}(x_2-x_1) & 0 & 0 & 0 \\
0 & \frac{1}{6}(x_2-x_1) & \frac{1}{3}(x_3-x_1) & \frac{1}{6}(x_3-x_2) & 0 & 0 \\
0 & 0 & \frac{1}{6}(x_3-x_2) & \frac{1}{3}(x_4-x_2) & \frac{1}{6}(x_4-x_3) & 0 \\
0 & 0 & 0 & \frac{1}{6}(x_4-x_3) & \frac{1}{3}(x_5-x_3) & \frac{1}{6}(x_5-x_4) \\
0 & 0 & 0 & 0 & \frac{1}{6}(x_5-x_4) & \frac{1}{3}(x_5-x_4)
\end{pmatrix}
\begin{pmatrix} \sigma_0 \\ \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ \sigma_4 \\ \sigma_5 \end{pmatrix}
=
\begin{pmatrix} \frac{y_1-y_0}{x_1-x_0} - v \\ z_1 \\ z_2 \\ z_3 \\ z_4 \\ w - \frac{y_5-y_4}{x_5-x_4} \end{pmatrix}$$

5 Matrizenmultiplikation

Wir erklären die Multiplikation von zwei Matrizen $A \cdot B = C$ am Beispiel von Matrizen mit 3 Zeilen und 3 Spalten:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix}$$

Der erste Index bezeichnet die Zeile und der zweite Index die Spalte des Eintrags. Wenn man nun den Eintrag c_{ij} berechnen möchte, bildet man das Skalarprodukt zwischen der i-ten Zeile der Matrix A und der j-ten Spalte der Matrix B.

$$c_{ij} = \begin{pmatrix} a_{i1} & a_{i2} & a_{i3} \end{pmatrix} \cdot \begin{pmatrix} b_{1j} \\ b_{2j} \\ b_{3j} \end{pmatrix} = a_{i1}b_{1j} + a_{i2}b_{2j} + a_{i3}b_{3j}$$

Beispiel: $c_{23} = a_{21}b_{13} + a_{22}b_{23} + a_{23}b_{33}$.

6 Der Gauß-Algorithmus

Beim Lösen von linearen Gleichungssystemen mit n Gleichungen und n Variablen bietet sich der Gauß-Algorithmus an. Hierfür muss ein Gleichungssystem der Form

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= r_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= r_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= r_n \end{aligned}$$

vorhanden sein. Um es nach dem folgenden System lösen zu können, ist nun wichtig, dass die entscheidenden, im Folgenden behandelten Stellen nicht 0 sind. Falls diese Stellen doch Null sind, müssen die Gleichungen so getauscht werden, dass dieses Problem nicht mehr vorhanden ist. Nun beginnen wir mit der ersten Gleichung und stellen diese nach x_1 um (möglich falls $a_{11} \neq 0$). Diese Lösung für x_1 setzen wir in die folgenden Gleichungen ein und fassen diese zusammen. Nun gehen wir zur zweiten Gleichung über und stellen diese nach x_2 um (möglich, falls Faktor vor x_2 ungleich Null). Diese Lösung wird wieder in die folgenden Gleichungen eingesetzt und zusammengefasst. Dieser Vorgang wird wiederholt, bis in der letzten Gleichung nur noch eine Unbekannte vorhanden ist. Das Gleichungssystem hat nun die Form

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= r_1 \\ 0 + \tilde{a}_{22}x_2 + \cdots + \tilde{a}_{2n}x_n &= \tilde{r}_2 \\ &\vdots \\ 0 + 0 + \cdots + \tilde{a}_{nn}x_n &= \tilde{r}_n \end{aligned}$$

Es lässt sich in der letzten Zeile sehr einfach nach x_n umformen. Dieses x_n wird nun in die vorletzte Gleichung eingesetzt und so x_{n-1} berechnet. Dieser Vorgang wird wiederholt, bis alle Variablen bekannt sind.

7 LU-Zerlegung

In unserer Rechnung haben wir uns einer anderen Art des Gauß-Algorithmus bedient. Unsere Gleichung hatte nun die Form

$$A\sigma = r.$$

Mit dem Gauß-Algorithmus wird die Matrix A in eine Matrix L und eine Matrix U aufgeteilt. Es gilt $A = L \cdot U$ mit

$$L = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ l_{21} & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & l_{n3} & \dots & 1 \end{pmatrix}, \quad U = \begin{pmatrix} u_{11} & u_{12} & u_{13} & \dots & u_{1n} \\ 0 & u_{22} & u_{23} & \dots & u_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & u_{nn} \end{pmatrix}$$

Da gelten soll $A \cdot \sigma = r$ und bereits festgelegt ist, dass $A = L \cdot U$, gilt nun:

$$\begin{aligned} A \cdot \sigma &= L \cdot U \cdot \sigma \\ L \cdot z &= r \quad \text{mit} \quad z = U \cdot \sigma \qquad (7.1) \\ U \cdot \sigma &= z. \qquad (7.2) \end{aligned}$$

Somit ergibt sich für (7.1) die Gleichung

$$\begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ l_{21} & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & l_{n3} & \dots & 1 \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{pmatrix} = \begin{pmatrix} r_1 \\ r_2 \\ \vdots \\ r_n \end{pmatrix}$$

Da r und L bekannt sind, lässt sich diese Gleichung sehr einfach lösen. Die erste Zeile ergibt sofort $1 \cdot z_1 = r_1$ Daraus lassen sich in den folgenden Zeilen alle z berechnen. Nun sind U und z bekannt. Es ergibt sich aus Gleichung (7.2), dass

$$\begin{pmatrix} u_{11} & u_{12} & u_{13} & \dots & u_{1n} \\ 0 & u_{22} & u_{23} & \dots & u_{2n} \\ \vdots & & & & \\ 0 & 0 & 0 & \dots & u_{nn} \end{pmatrix} \begin{pmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_n \end{pmatrix} = \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{pmatrix}$$

Da die U-Matrix die Form des Gauß-Algorithmus aufweist, lässt sich die Gleichung nun von unten nach oben lösen, da sich in der untersten Zeile $u_{nn} \cdot \sigma_n = z_n$ ergibt. Durch diese Gleichung erhält man sofort σ_n und kann die weiteren Sigmas berechnen.

8 LU-Zerlegung bei Tridiagonalmatrizen

Bei unserem Gleichungssystem entsteht eine Tridiagonalmatrix, welche die Form hat:

$$\begin{pmatrix} a_{11} & a_{12} & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 0 & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 & 0 \\ 0 & 0 & a_{43} & a_{44} & a_{45} & 0 \\ 0 & 0 & 0 & a_{54} & a_{55} & a_{56} \\ 0 & 0 & 0 & 0 & a_{65} & a_{66} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ l_{21} & 1 & 0 & 0 & 0 & 0 \\ 0 & l_{32} & 1 & 0 & 0 & 0 \\ 0 & 0 & l_{43} & 1 & 0 & 0 \\ 0 & 0 & 0 & l_{54} & 1 & 0 \\ 0 & 0 & 0 & 0 & l_{65} & 1 \end{pmatrix} \cdot \begin{pmatrix} u_{11} & u_{12} & 0 & 0 & 0 & 0 \\ 0 & u_{22} & u_{23} & 0 & 0 & 0 \\ 0 & 0 & u_{33} & u_{34} & 0 & 0 \\ 0 & 0 & 0 & u_{44} & u_{45} & 0 \\ 0 & 0 & 0 & 0 & u_{55} & u_{56} \\ 0 & 0 & 0 & 0 & 0 & u_{66} \end{pmatrix}$$

Wie man sieht, sind bei einer Tridiagonalmatrix nur die Hauptdiagonale und die beiden Nebendiagonalen ungleich 0. In diesem Fall ist die LU-Zerlegung sehr einfach, da außer der Hauptdiagonalen nur ein Eintrag pro Zeile ungleich 0 ist. Wenn man die L und die U Matrix multipliziert, entstehen für die erste A-Zeile folgende Gleichungen:

$$\begin{aligned} a_{11} &= u_{11} \\ a_{12} &= u_{12}. \end{aligned}$$

Für die zweite A-Zeile ergeben sich folgende Gleichungen, welche sich rekursiv, für die weiteren Zeilen, fortsetzen lassen.

$$\begin{aligned} a_{21} &= u_{11}l_{21} & a_{i,i-1} &= u_{i-1,i-1}l_{i,i-1} & \forall i = 2,\dots,6 \\ a_{22} &= u_{12}l_{21} + u_{22} & a_{i,i} &= u_{i-1,i}l_{i,i-1} + u_{i,i} & \forall i = 2,\dots,6 \\ a_{23} &= u_{23} & a_{i,i+1} &= u_{i,i+1} & \forall i = 2,\dots,5. \end{aligned}$$

Für $i = 6$ entfällt die dritte Gleichung, weil die Elemente außerhalb der Matrizen liegen.

9 Implementierung in Python

Nun, da alle Gleichungen aufgestellt wurden, können wir mit Hilfe der Programmiersprache PYTHON die Funktion aufstellen und als Graph visualisieren. Bevor dies möglich ist, müssen zuerst verschiedene Bibliotheken importiert werden, die Methoden zur Verfügung stellen, mit denen z.B. Graphen gezeichnet werden können.
Damit der Graph der Wasserrutsche gezeichnet werden kann, muss zunächst das lineare Gleichungssystem $A\sigma = r$ gelöst werden. Dazu erzeugt man zuerst die Matrix A und die dazugehörige rechte Seite r erzeugt werden. Die Matrix wird mit 6x6 Zellen, die mit Nullen gefüllt werden, erzeugt. Dann wird die Matrix mit Werten gefüllt, nach demselben Prinzip wird die rechte Seite gefüllt. Durch diese kann nun σ durch eine Methode aus der Bibliothek berechnet werden.

```
s   = solve(A,r) # A = Matrix; r = rechte Seite
```

Anschließend werden die Koeffizienten in Abhängigkeit von σ berechnet, die in einer 5x4 Matrix dargestellt werden. Mit den Koeffizienten können wir die Funktion aufstellen.

```
def f(x,K,xp):
    for i in xrange(1,6):
        if (xp[i-1]<=x<=xp[i]):
            a = K[i-1][0]*(x-xp[i])**3
            b = K[i-1][1]*(x-xp[i])**2
            c = K[i-1][2]*(x-xp[i])
            d = K[i-1][3]
            f_x = a+b+c+d
    return f_x
```

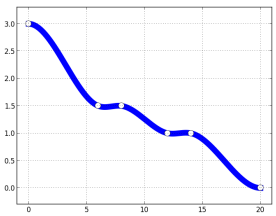
Zu beachten ist dabei, dass die Splinefunktion der Wasserrutsche aus mehreren Teilfunktionen besteht und somit in PHYTON eine Laufvariable i nötig ist, um zu testen, in welchem Teilintervall der Punkt x liegt. Zum Schluss müssen wir die Funktion nur noch plotten, d.h. grafisch darstellen.

```
x=array(arange(0,20,0.01))
y=array(arange(0,20,0.01))
for i in xrange(0,2000):
    y[i]=f(x[i],K,xp)
figure()
plot(x,y,color='blue',linewidth=10 )
plot(xp,yp,'o',markersize = 10,color ='white')
```

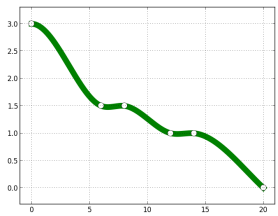
Da beim Plotten die Punkte nur linear verbunden werden, muss die Splinefunktion an vielen Punkten (hier 2000 Punkte) berechnet werden, um die gewünschte Kurve zu sehen.

10 Ergebnisse

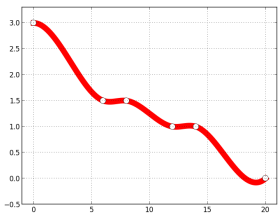
Die zuvor beschriebene Berechnung der Splinefunktion ergibt nun für den Anstieg $v = 0$, $v < 0$ und $v > 0$ an der rechten Seite folgende Form:



$v = 0$



$v < 0$



$v > 0$