Universität Potsdam
Mathematisch-Naturwisssenschaftliche Fakultät
Institut für Physik und Astronomie

Computational Physics

Korteweg-de-Vries

Autor: Christian Gößl
Matrikel-Nr.:762627

Korrektor: Prof. Dr. Arkady Pikovsky

8. Juli 2019

Inhaltsverzeichnis

1 Einleitung

Das Soliton ist ein Phänomen aus der Wellenmechanik, welches durch die nichtlineare Physik beschrieben werden kann. Es sind Wellen, die über einen langen Zeitraum ihre Form und Geschwindigkeit beibehalten können. Die bekannteste Eigenschaft ist, dass bei einem Zusammenstoß von zwei Solitonen beide nach dem Aufeinandertreffen ihre Form und Geschwindigkeit beibehalten. Sinngemäß, als würden sie sich durchdringen und danach unbeschadet weiter fließen. Wir haben es hier mit einer Wellen zu tun. Für die mathematische Beschreibung benutzen wir die Korteweg de Vries Gleichung, damit können wir die das Phänomen simulieren. Hierbei beschränken wir uns auf den eindimensionalen Fall. In dieser Arbeit wird es hauptsächlich, um die Anwendung der Korteweg-de-Vries Gleichung auf den Fall von Solitonwellen gehen.

2 Korteweg-de-Vries Gleichung

$$\eta(x,t)_t + \sqrt{gd}\left(\eta(x,t)_x + \frac{d^2}{6}\eta(x,t)_{xxx} + \right. \tag{1}$$

$$\left. + \frac{3}{2d}\eta(x,t)\eta(x,t)_x\right) = 0 \tag{2}$$

Hierbei beschreibt $\eta(x,t)$ die Anhebung des Wassers. Wie gewohnt beschreiben x,t die Raumkoordinate und Zeit, g ist die Fallbeschleunigung und d ist die Wassertiefe. Die tiefergestellten Variablen zeigen eine partielle Ableitung an $\partial\eta/\partial t = \eta_t$. Diese Gleichung werden wir nun etwas reskalieren. Zuerst fassen wir den zweiten und vierten Term zusammen und setzen den Faktor vor der Klammer zum ersten Term um.

$$\frac{1}{\sqrt{gd}}\eta_t + \eta_x\left(1 + \frac{3}{2d}\eta\right) + \frac{d^2}{6}\eta_{xxx} = 0$$

Die Gleichung multiplizieren wir mit $6d$ und reskalieren wir die Zeit, Raumkoordinate wie folgt:

$$t = 6\sqrt{\frac{d}{g}}\tau, \quad x = dy \tag{3}$$

Daraus ergibt sich folgende Gleichung

$$\eta_\tau + \eta_y\left(1 + \frac{3}{2d}\eta\right) + \eta_{yyy} = 0$$

Wir machen eine Transformation von η nach u wie folgt:

$$1 + \frac{3}{2d}\eta = u, \quad u_y = \frac{3}{2d}\eta_y, \quad u_\tau = \frac{3}{2d}\eta_\tau \tag{4}$$

Diese setzen wir wieder in die Ausgangsgleichung ein:

$$u_\tau + 6uu_y + u_{yyy} = 0$$

Wir schreiben die reskalierten Variablen in der alten Notation:

$$u(t,x)_t + 6u(t,x)u(t,x)_x + u(t,x)_{xxx} = 0 \tag{5}$$

Eine mögliche Lösung für diese Differentialgleichung für Soliton Wellen ist:

$$u(x,t) = \frac{A}{\cosh^2(kx - wt - \eta_0)} \tag{6}$$

mit $A = 2k^2, w = 4k^3$ und η_0 als Anfangsanhebung. Die Integrale der Momente der Korteweg-de-Vries Gleichung sollen in der Zeit konstant bleiben:

$$\frac{\mathrm{d}P_k}{\mathrm{d}t} = 0 \tag{7}$$

$$P_0 = \int_{-\infty}^{\infty} u \,\mathrm{d}x = \text{constant}$$

$$P_1 = \int_{-\infty}^{\infty} u^2 \,\mathrm{d}x = \text{constant}$$

$$P_2 = \int_{-\infty}^{\infty} \left(2u^3 - (u_x)^2\right) \,\mathrm{d}x = \text{constant} \tag{8}$$

2.1 Numerisches Lösungsverfahren: Zabuski-Kruskal Schema

Zur Lösung der KdV Gleichung benutzen wir das Zabuski-Kruskal Schema:

$$\begin{aligned}
u &= \frac{1}{3}\left(u_{i-1}^m + u_i^m + u_{i+1}^m\right) \\
u_t &= \frac{1}{2\Delta t}\left(u_i^{m+1} - u_i^{m-1}\right) \\
u_x &= \frac{1}{2\Delta x}\left(u_{i+1}^m - u_{i-1}^m\right) \\
u_{xxx} &= \frac{1}{2\Delta x^3}\left(u_{i+2}^m - 2u_{i+1}^m + 2u_{i-1}^m - u_{i-2}^m\right) \\
u_t + 6uu_x + u_{xxx} &= \frac{1}{2\Delta t}\left(u_i^{m+1} - u_i^{m-1}\right) + \frac{1}{\Delta x}\left(u_{i-1}^m + u_i^m + u_{i+1}^m\right)\left(u_{i+1}^m - u_{i-1}^m\right) + \\
&\quad + \frac{1}{2\Delta x^3}\left(u_{i+2}^m - 2u_{i+1}^m + 2u_{i-1}^m - u_{i-2}^m\right) = 0
\end{aligned} \tag{9}$$

m steht für den Zeitindex und i steht für den Ortsindex. Für den ersten Zeitschritt wird ein 1. Ordnung Schemata verwendet. Die Gleichung 5 wird umgeschrieben:

$$\frac{\mathrm{d}u}{\mathrm{d}t} = -6uu_x - u_{xxx}$$
$$u_i^1 = u_i^0 + \Delta t\left(-6uu_x - u_{xxx}\right)$$

Damit die Simulation stabil bleibt müssen wir eine geeignete Wahl von Anfangsbedingungen und Schrittgröße ermitteln. Die Schrittgröße lässt sich aus dem Stabilitätskriterium ableiten:

$$\frac{\Delta t}{\Delta x}\left|-2u_0 + \frac{1}{\Delta x^2}\right| < \frac{2}{3\sqrt{3}} \tag{10}$$

Hierbei ist u_0 das Maximum der Lösung $u(t,x)$:

$$u_0 = N(N+1)$$

Die Anfangsbedingungen sind für die gesamte Arbeit für $t = 0$ auf

$$u(0,x) = \frac{N(N+1)}{\cosh^2(x)}$$

gesetzt. Weitere Simulationsparameter sind, die Größe G des Gitter, die Schrittzahl n und die Startposition $x_0 = G \cdot e$ des Solitons, wobei die Werte von $-1 \leqslant e \leqslant 1$ gehen. Aus dem Stabilitätskriterium lässt sich Δt für ein beliebiges Δx abschätzen. Wir wählen periodische Randbedingungen aus. Daraus erhalten wir eine Lösung die uns N Solitonen in der Simulation erzeugt, damit ist N ein weiterer Paramter zur Steuerung der Simulation für die gesamte Arbeit. Dieses Verfahren bildet die Grundlagen für das Problem 1 im nächsten Kapitel.

2.2 Korteweg-de-Vries Gleichung bei variabler Tiefe

Für die Simulation eines Bodenprofils des Wasser führen wir die Wassertiefenfunktion $h(x)$ ein. Dazu wird die Gleichung 5 mit $h(x)$ folgendermaßen modifiziert:

$$u(t,x)_t + \frac{6}{h^{7/4}(x)}u(t,x)u(t,x)_x + h^{1/2}(x)u(t,x)_{xxx} = 0 \tag{11}$$

Das Zabuski-Kruskal Schemta wird mit den gleichen ausgeführten Anfangsbedingungen und Lösungen verwendet. Gegeben falls muss die Schrittweite oder Startpunkt der Lösung angepasst werden, damit die Simulation stabil bleibt.

3 Erzeugung von Solitonwellen mit konstanter Tiefe (Problem 1)

Wir können mittels der Variable N die Anzahl an Wellenbergen und deren Art bestimmen. Ist N eine natürliche Zahl so erhalten wir eine N-Soliton Lösung. Damit die Stabilität der Lösung erhalten bleibt, muss eine kleiner Schritt Δx getan werden, womit der Zeitschritt Δt aus Gleichung 10 abgeschätzt werden kann. Durch mehrere Test hat sich der Wert $\Delta x = 0{,}1$ als stabil ergeben. In den Diagrammen ist auf der vertikalen Achse u die Anhebung und an der horizontalen Achse x der Ort verzeichnet. Im rechten oberen Bereich sind die Simulationszeit, Schrittzahl n und die Integrale $P_{0,1,2}$ eingeblendet. Die Intervallgröße des Diagramms wird dynamisch durch die Formel:

$$u_{\max} = N(N + 4{,}5)$$

bestimmt, welche durch den maximalen Wert der Soliton-Lösung, nur durch bloßem Ausprobieren erlangt worden ist. Wir setzen nun für die Parameter

$$G = 5{,}5\,, N = 3\,, n = 600\,, x_0 = G \cdot 0{,}3$$

fest, damit erzeugen wir eine 3-Soliton Lösung:

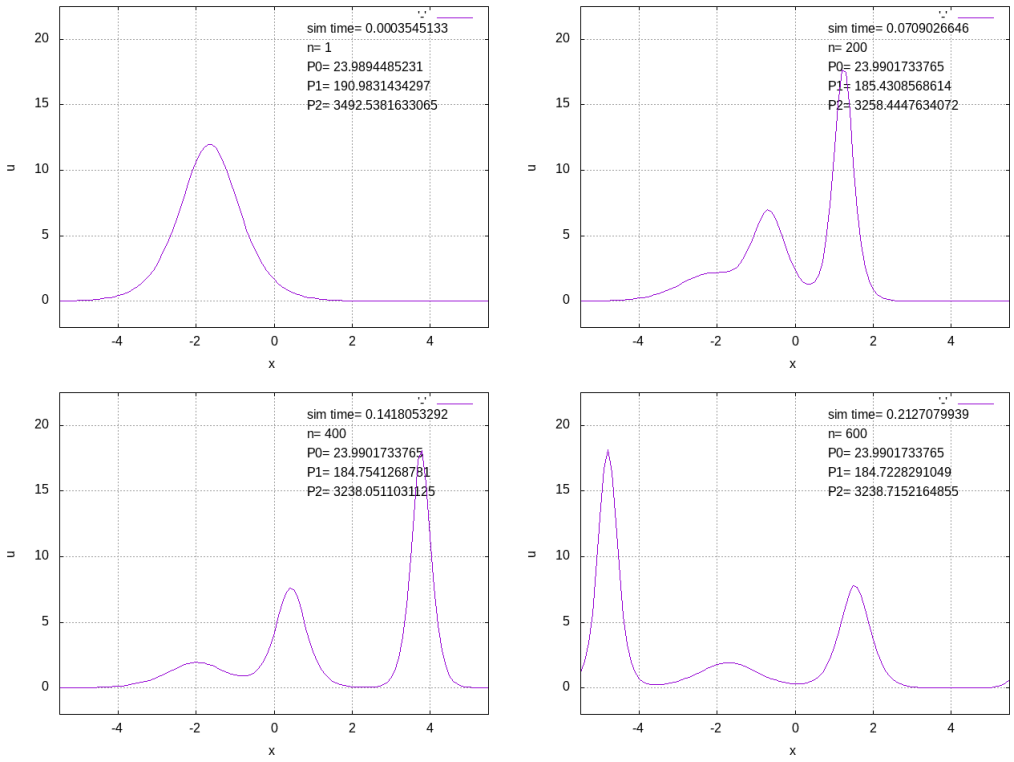

Abbildung 1: Eine 3-Soliton Lösung. Die Welle teilt sich in drei einzelne Solitonwellen auf. Die Integrale $P_{0,1,2}$ verbleiben wie geplant konstant.

Die Simulation bleibt stabil, die Integrale $P_{0,1,2}$ bleiben konstant. Die Welle teilt sich nach einer bestimmten Zeit in zwei weitere Wellen auf. Die Anfangslösung, eine Solitonwelle, wird in drei Solitonwellen transformiert. Die Anzahl der Solitonwellen richtet sich nach dem Parameter N. Wenn sich zwei Solitonwellen begegnen, vermischen sie sich kurzzeitig zu einer Welle. Danach trennen sie sich wieder und gehen in ihrem Ursprungszustand zurück. Dabei behalten sie ihre Bewegungsrichtung bei. Die Wellen bleiben erhalten. Diese fundamentale wichtige Eigenschaft der Soliton wird durch dieses Programm deutlich wiedergegeben.

Was passiert nun wenn wir für N eine rationale Zahl einsetzen? Demnach setzen wir für unsere weitere Untersuchung für $N = 2{,}5$ an. Wir benutzen zudem die selben Paramter wie in der vorherigen Simulation.

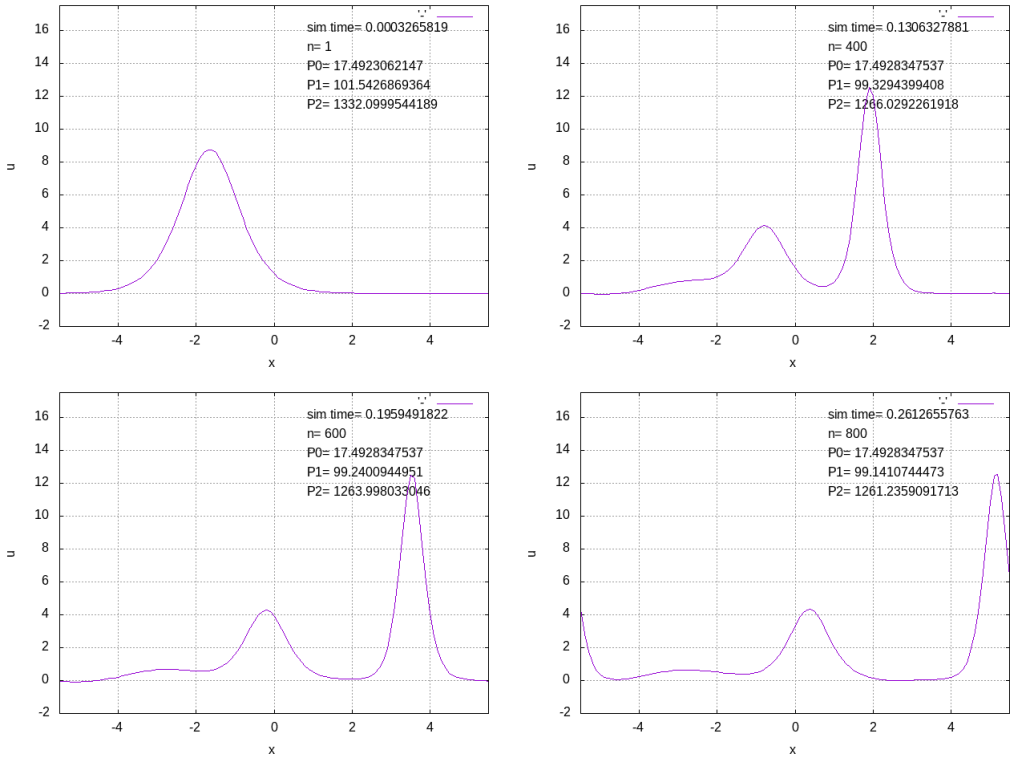

Abbildung 2: Lösung für $N = 2{,}5$. Die Solitonwelle teilt sich wie erwartet in weitere Wellenberge auf. Hierbei verbleibt eine kleine Welle übrig, die sich nicht bewegt.

Die Lösung teilt sich in zwei Solitonwellen und in einem Wellenberg, der sich nicht bewegt, auf. Der unbewegte Wellenberg verhält sich wie ein Soliton, beim Aufeinandertreffen mit einer Welle vermischen sie sich und gehen danach wieder unverändert auseinander. Die Integrale $P_{0,1,2}$ verbleiben einigermaßen konstant.

4 Solitonwellen mit variabler Tiefe (Problem 2)

Wir variieren nun die Wassertiefe mittels eine Wassertiefenfunktion $h(x)$. Die Tiefe wird durch den Parameter h_0 charakterisiert. Die mathematische Beschreibung geht aus dem Abschnitt 2.2 hervor.

4.1 $h_1(x)$ Unterwassererhöhung

Wir platzieren im Wasser ein kleine Erhöhung. In unserer ersten Untersuchung bezeichnen wir die Wassertiefenfunktion mit $h_1(x)$. Sie ist folgendermaßen definiert:

$$h_1(x) = \begin{cases} 1 & x < 0 \\ \frac{1+h_0+(1-h_0)\cos(\pi x/L))}{2} & 0 < x < L \\ h_0 & x > L \end{cases} \tag{12}$$

Dabei haben wir einen neuen Parameter eingefügt, der sich dynamisch aus der Gittergröße G berechnet: $L = G/6$. Wir benutzen als Parameter:

$$\Delta x = 0{,}1\,, G = 16{,}4\,, N = 1{,}0\,, n = 4400\,, h_0 = 0{,}5\,, x_0 = G \cdot (-0{,}2)$$

Im Diagramm sind zum einen die Höhenfunktion $h_1(x)$ in blau und die Welle u in violett dargestellt.

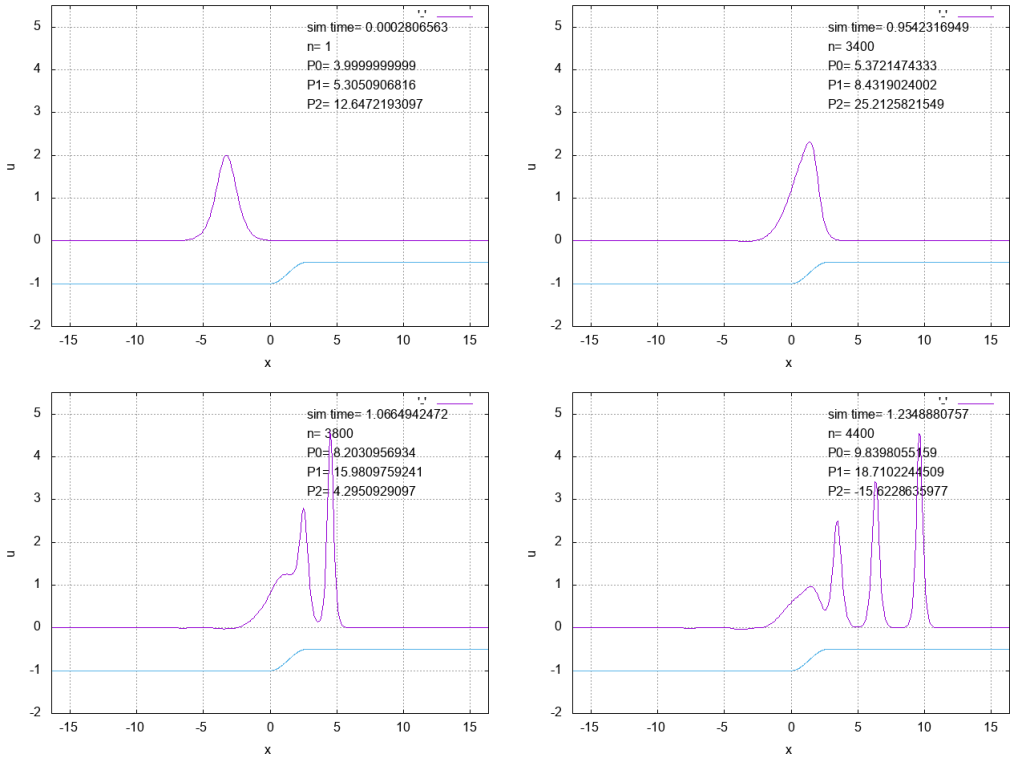

Abbildung 3: Durch passieren der Erhebung wird die Welle transformiert. Es werden weitere Wellen mit höherer Amplitude erzeugt.

Durch das Passieren des Höhenunterschieds werden nacheinander neue Wellen erzeugt, die Solitonwelle wird in Weitere transformiert, obwohl wir den Parameter auf $N = 1$ gesetzt haben. Die Amplitude der Welle wird vergrößert. Im Gegenzug verdünnt sich deren Breite. Bevor sie den Rand erreichen brechen wir die Simulation ab, sonst wird durch den Sprung in der Höhenfunktion, die Simulation ungewollt gestört.

4.2 Abhängigkeit der Amplitude $A(h_0)$ von der Tiefe h_0

Weiterhin untersuchen wir die Abhängigkeit der Amplitude von der gewählten Höhe h_0 der Anhebung. Dazu verwenden wir wieder die gleichen Parameter. Die Schrittzahl wurde anhand der gewählten h_0 gesetzt. Je größer der Wert von h_0 ansteigt desto schneller werden Wellen erzeugt und sie bewegen sich mit größere Geschwindigkeit auf den Rand der Simulation zu.

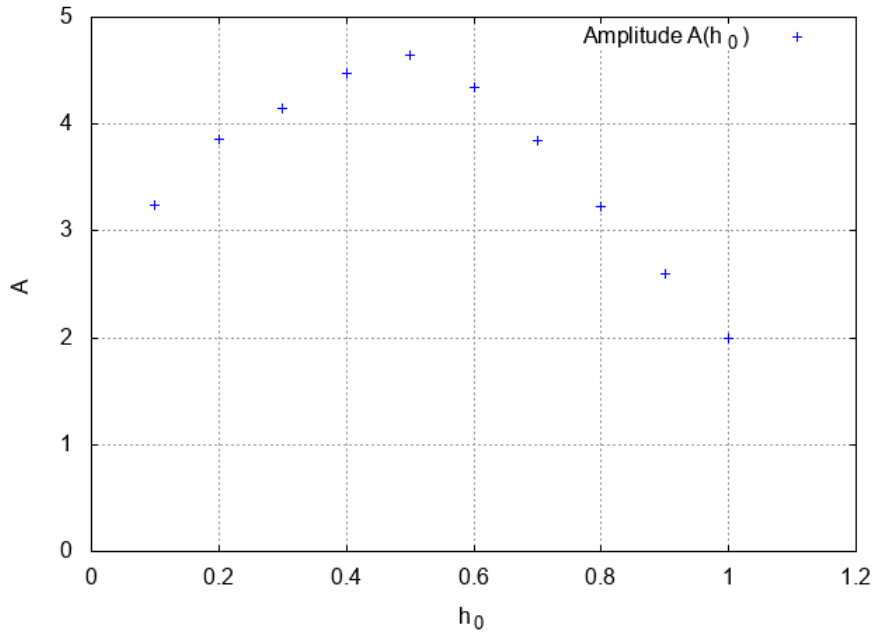

Abbildung 4: Die Amplitude steigt linear von $A(0{,}1) = 2{,}59$ bis auf $A(0{,}5) = 4{,}64$ an und danach fällt sie wieder ab auf den ursprünglichen Wert der Amplitude $A(1) = 2$ ab.

Es lässt sich aus dem Diagramm ableiten, dass die Amplitude linear ansteigt bis $h_0 = 0{,}5$ und danach wieder linear abfällt bis zur ursprünglichen Amplitude $A = 2$. Wir setzen für den Fit die vermutete lineare Abhängigkeit

$$A(h_0) = a \cdot h_0 + b$$

auf beide Stiegungen separat an.

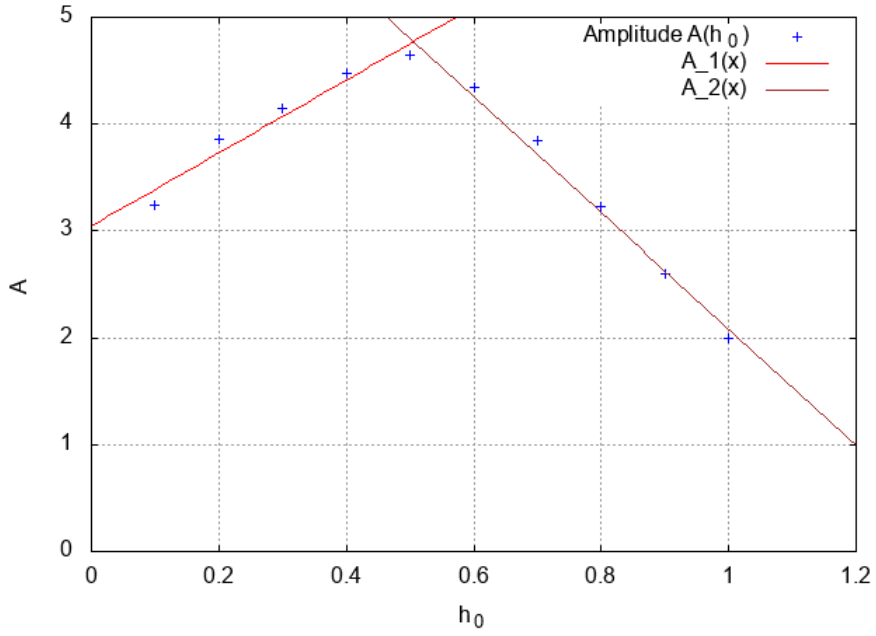

Abbildung 5: Die Amplitudenfunktion $A(h_0)$ hängt linear von h_0 und ähnelt der Betragsfunktion.

Wir bezeichnen $A_1(h_0)$ als die Funktion im Bereich von $0 < h_0 < 0{,}5$ und $A_2(h_0)$ als Funktion für das Intervall $0{,}5 < h_0 < 1$.

$$A_1(h_0) = 3{,}41h_0 + 3{,}05$$
$$A_2(h_0) = -5{,}45h_0 + 7{,}53 \tag{13}$$

4.3 $h_2(x)$ Beispiel eines Unterwasserhügels, exponentielle Wassertiefenfunktion

Zur weiteren Untersuchung benutzen wir nun eine weitere Wassertiefenfunktion, den Unterwasserhügel.

$$h_2(x) = 1 - h_0 \exp(-x^2/L^2) \tag{14}$$

Wir benutzen die Werte:

$$\Delta x = 0{,}05\,, G = 10{,}4\,, N = 1{,}0\,, n = 60000\,, h_0 = 0{,}5\,, x_0 = 0$$

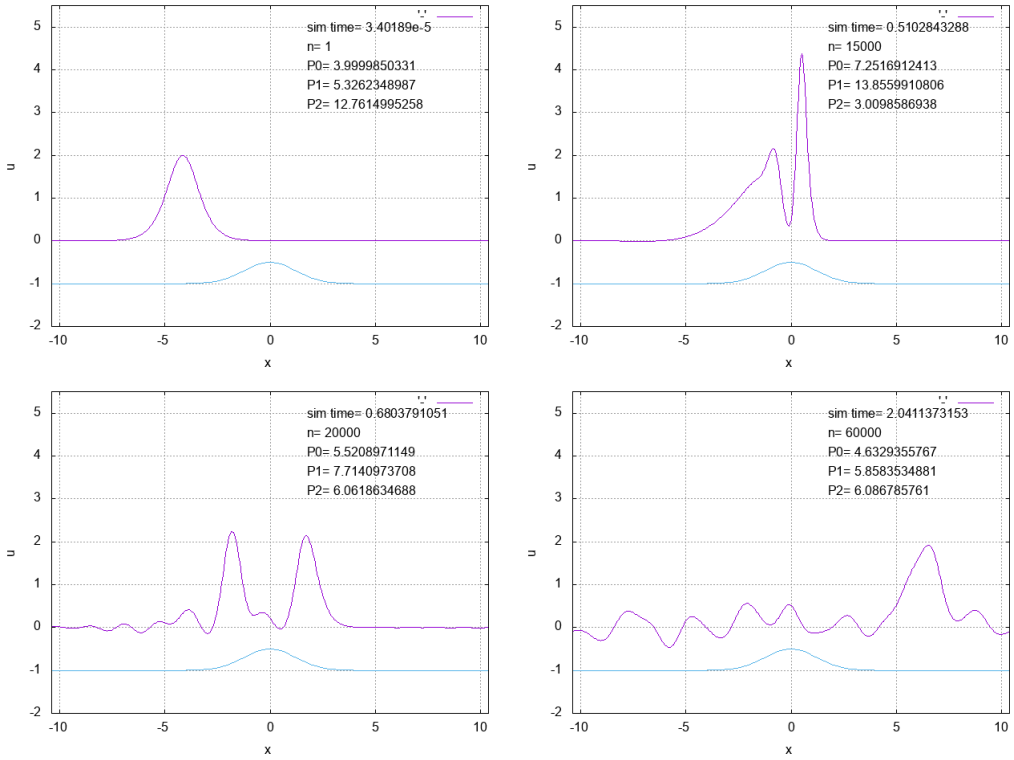

Abbildung 6: Die Solitonwelle bewegt sich nach rechts und teilt sich beim Unterwasserhügel in weitere Wellen auf. Dabei werden einige Solitone erzeugt, die sich in die entgegengesetzte Richtung bewegen.

Die Solitonwelle teilt sich in weitere Wellen auf, davon bewegen sich einige von rechts nach links. Ein Soliton bewegt sich langsam weiter nach rechts, dabei trifft sie auf die übrigen kleinen Wellen.

5 Zusammenfassung

Wir haben mit diesem Programm das Phänomen des Solitons untersucht. Die fundamentale Eigenschaften solch einer Welle konnten wir mit dieser Simulation aufzeigen. Wir konnten zeigen, dass die Wellen nach einem Zusammenstoß wieder in ihren vorherigen Zustand zurückkehrten. Es ist zu sehen, dass die Wellen bei ihrer Bewegung ihre Form und Geschwindigkeit beibehalten haben. Wir haben die Tiefe des Wasser variiert, um das Verhalten eines Solitons in dieser physikalischen Situation zu studieren und kamen auf das Ergebnis, dass die Welle sich transformierte und weitere Wellen erzeugte. Das Soliton konnte durch keine der gemachten Störungen zerstört werden. Die Eigenschaften blieben erhalten. Die Amplitude der Wellen wurde verändert. Wir haben zwei lineare Zusammenhänge zwischen der Amplitude und Tiefe finden können. Die Form der Wassertiefenfunktion haben verändert und konnten das selbe Verhalten der Wellen beobachten. Hierbei haben wir nur den eindimensionalem Fall betrachtet, der bestimmt mit weiteren Überlegungen und Differentialgleichungen auf Fälle höherer Dimension erweitert werden kann.

6 Korrekturen

6.1 Konstanz der Momente

Als erstes zeigen wir, dass die Integrale der Momente der Korteweg-de-Vries Gleichung für alle Zeiten konstant ist. Es gilt

$$\frac{\mathrm{d}P_k}{\mathrm{d}t} = 0$$

Dazu nehmen wir als Anfangsbedingungen:

$$\begin{aligned} u(\infty) &= u(-\infty) = 0 \\ u_x(\infty) &= u_x(-\infty) = 0 \\ u_{xx}(\infty) &= u_{xx}(-\infty) = 0 \\ u_{xxx}(\infty) &= u_{xxx}(-\infty) = 0 \\ u_{xxxx}(\infty) &= u_{xxxx}(-\infty) = 0 \end{aligned} \tag{15}$$

Beim 1. Integral erhalten wir

$$\frac{\mathrm{d}P_0}{\mathrm{d}t} = \frac{\mathrm{d}}{\mathrm{d}t}\int u \,\mathrm{d}x = 0$$

Wir benutzen die Korteweg Gleichung und setzen sie für die Zeitableitung ein

$$\begin{aligned} \int u_t \,\mathrm{d}x &= \int (-6uu_x - u_{xxx})\,\mathrm{d}x \\ &= -3u^2 - u_{xx}\,\Big|_{-\infty}^{\infty} \end{aligned}$$

Wir benutzen die Anfangsbedingungen aus 15

$$-3u^2 - u_{xx}\,\Big|_{-\infty}^{\infty} = 0\,.$$

Beim 2. Integral gehen wir genauso vor

$$\begin{aligned} \frac{\mathrm{d}P_1}{\mathrm{d}t} &= \frac{\mathrm{d}}{\mathrm{d}t}\int_{-\infty}^{\infty} u^2 \,\mathrm{d}x = 0 \\ &= \int_{-\infty}^{\infty} uu_t \,\mathrm{d}x = \\ &= -2uu_{xx} + (u_x)^2\,\Big|_{-\infty}^{\infty} \end{aligned}$$

Wir benutzen die Anfangsbedingungen aus 15

$$-2uu_{xx} + (u_x)^2\,\Big|_{-\infty}^{\infty} = 0\,.$$

Beim 3. Integral ebenso

$$\begin{aligned} \frac{\mathrm{d}P_2}{\mathrm{d}t} &= \frac{\mathrm{d}}{\mathrm{d}t}\int_{-\infty}^{\infty} \left(2u^3 - (u_x)^2\right)\,\mathrm{d}x = 0 \\ &= \int_{-\infty}^{\infty} (-36u^3u_x - 6u^2u_{xxx} + 12u_x\frac{\mathrm{d}}{\mathrm{d}x}(uu_x) + 2u_xu_{xxxx})\,\mathrm{d}x \\ &= -9u^4 - 6u^2u_{xx} + 12u_x^2u + 2u_xu_{xxx} - (u_{xx})^2\,\Big|_{-\infty}^{\infty} + 12\int_{-\infty}^{\infty} uu_{xx}u_x\mathrm{d}x - 12\int_{-\infty}^{\infty} uu_{xx}u_x\mathrm{d}x \end{aligned}$$

Wir benutzen die Anfangsbedingungen aus 15

$$-9u^4 - 6u^2u_{xx} + 12u_x^2u + 2u_xu_{xxx} - (u_{xx})^2\,\Big|_{-\infty}^{\infty} = 0\,.$$

Wir untersuchen nun die Konstanz der Momente in den Simulationen. Numerisch berechnen wir die Integrale wie folgt:

$$P_{i+1,0} = P_{i,0} + \frac{1}{3}\left(u_{i-1}^m + u_i^m + u_{i+1}^m\right)\Delta x \tag{16}$$

$$P_{i+1,1} = P_{i,1} + \left(\frac{1}{3}\left(u_{i-1}^m + u_i^m + u_{i+1}^m\right)\right)^2 \Delta x \tag{17}$$

$$P_{i+1,2} = P_{i,2} + \left(2\left(\frac{1}{3}\left(u_{i-1}^m + u_i^m + u_{i+1}^m\right)\right)^3 + \left(\frac{1}{2\Delta x}\left(u_{i+1}^m - u_{i-1}^m\right)\right)^2\right)\Delta x \tag{18}$$

Wir erhöhen die Größe des Gitters und lassen die Simulation länger laufen. Als Parameter wählen wir:

$$\Delta x = 0{,}1\,, G = 20\,, N = 2\,, n = 50000\,, x_0 = G \cdot 0{,}3\,, \Delta t = 0{,}00028\,.$$

Hierbei verändern wir ausschließlich den Parameter $N = [1\,, 2\,, 2{,}5\,, 3]$.

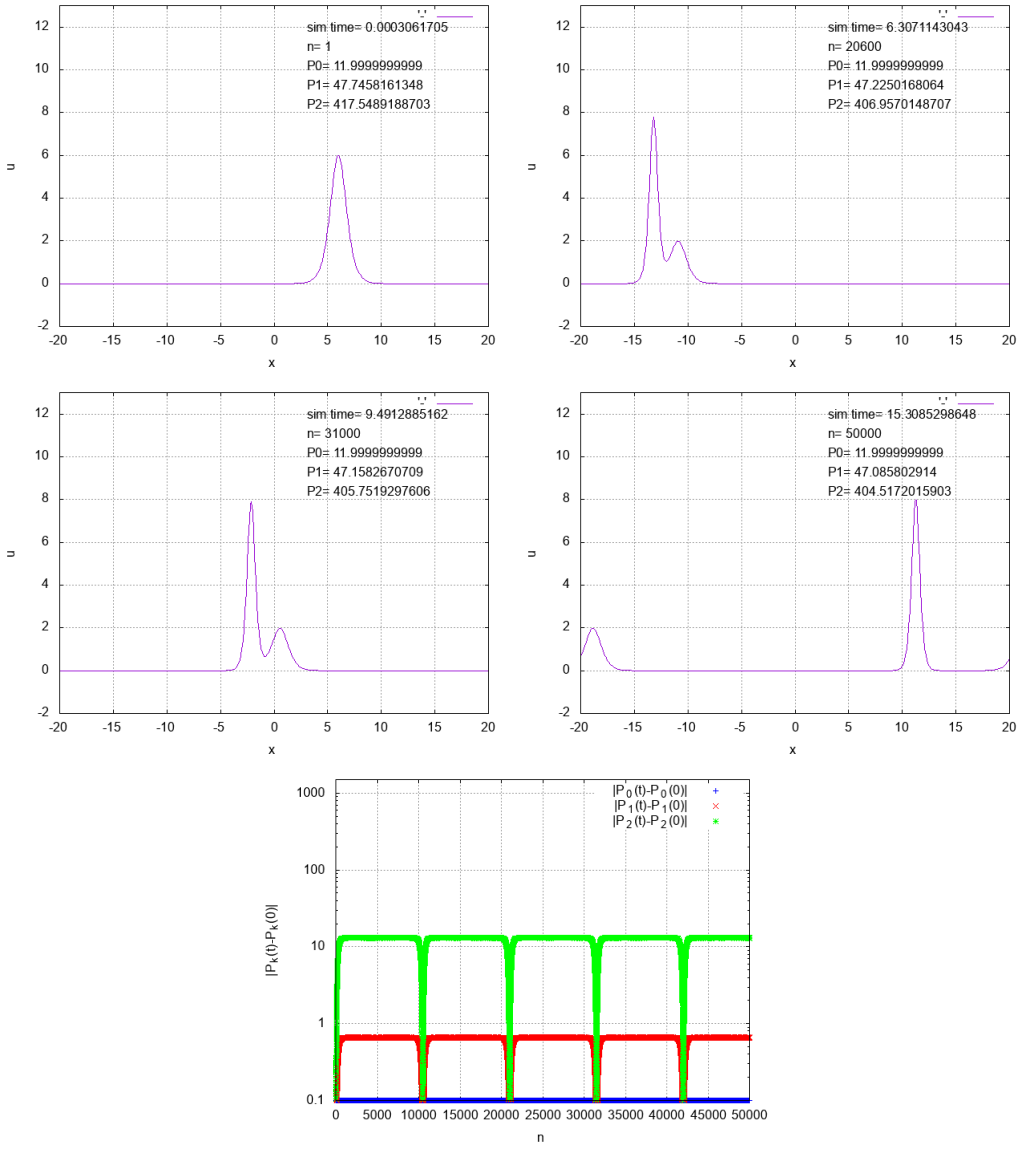

Abbildung 7: Die Solitonwelle bewegt sich nach rechts und teilt sich entsprechend $N = 2$ in zwei weitere Wellen auf. Im letzten Diagramm sieht man deutlich bei welchen Zeitpunkten sich die Wellen wieder vereinigen. Die Werte der Integrale sinken wieder auf Null runter.

Das selbe Phänomen können wir bei größeren Werten vom Parameter N feststellen. Hier für

$N = 3$

Für den Fall $N = 2{,}5$ beobachten wir fasst das selbe, im Unterschied zu dem davor, haben wir hier eine

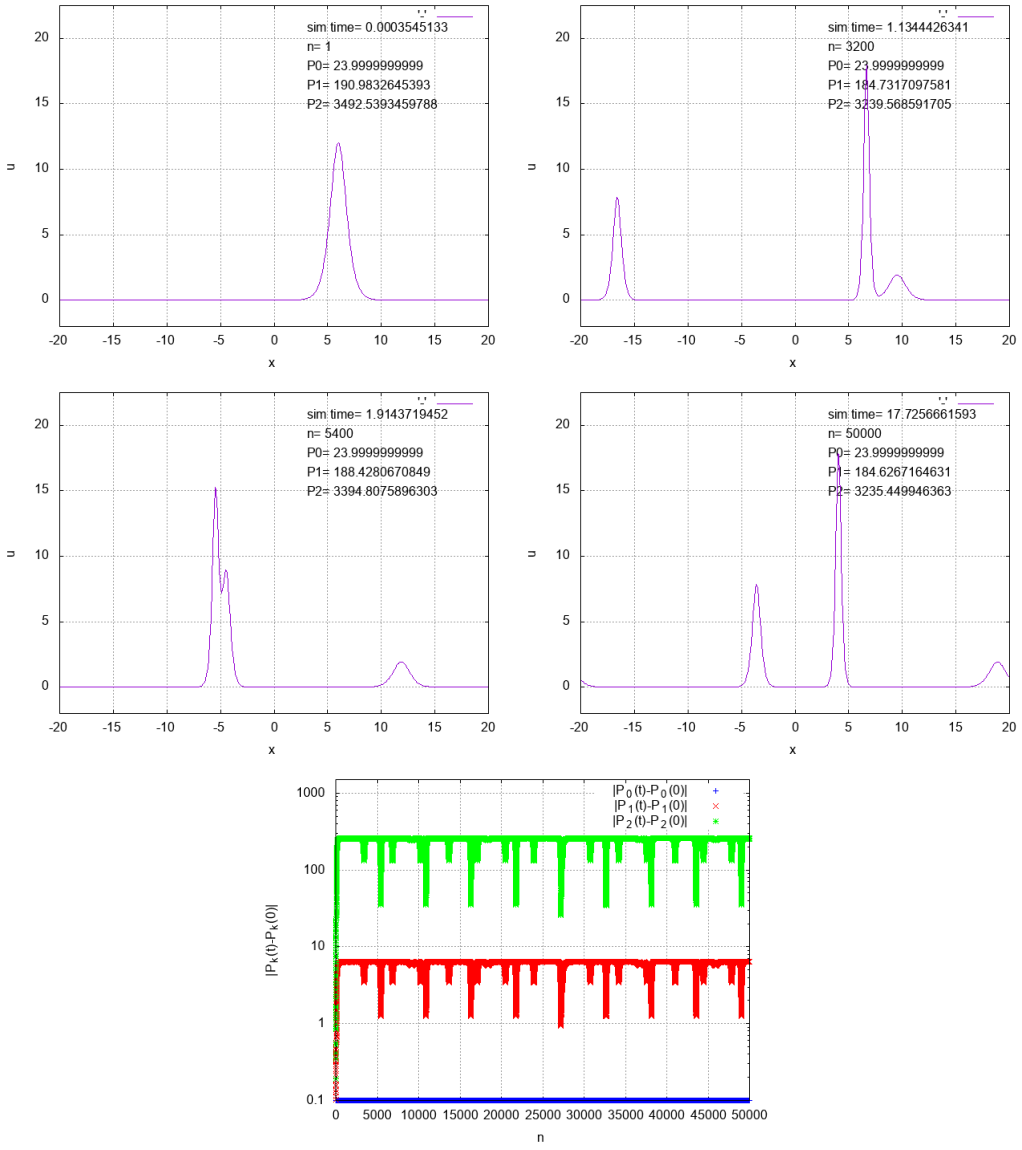

Abbildung 8: Die Solitonwelle bewegt sich nach rechts und teilt sich entsprechend $N = 3$ in drei weitere Wellen auf. Im letzten Diagramm sieht man deutlich bei welchen Zeitpunkten sich die Wellen wieder vereinigen. Die Werte der Integrale sinken wieder auf Null runter.

kleinere Welle die sich sehr langsam bewegt und viele winzige Wellen, die sich nach links bewegen.

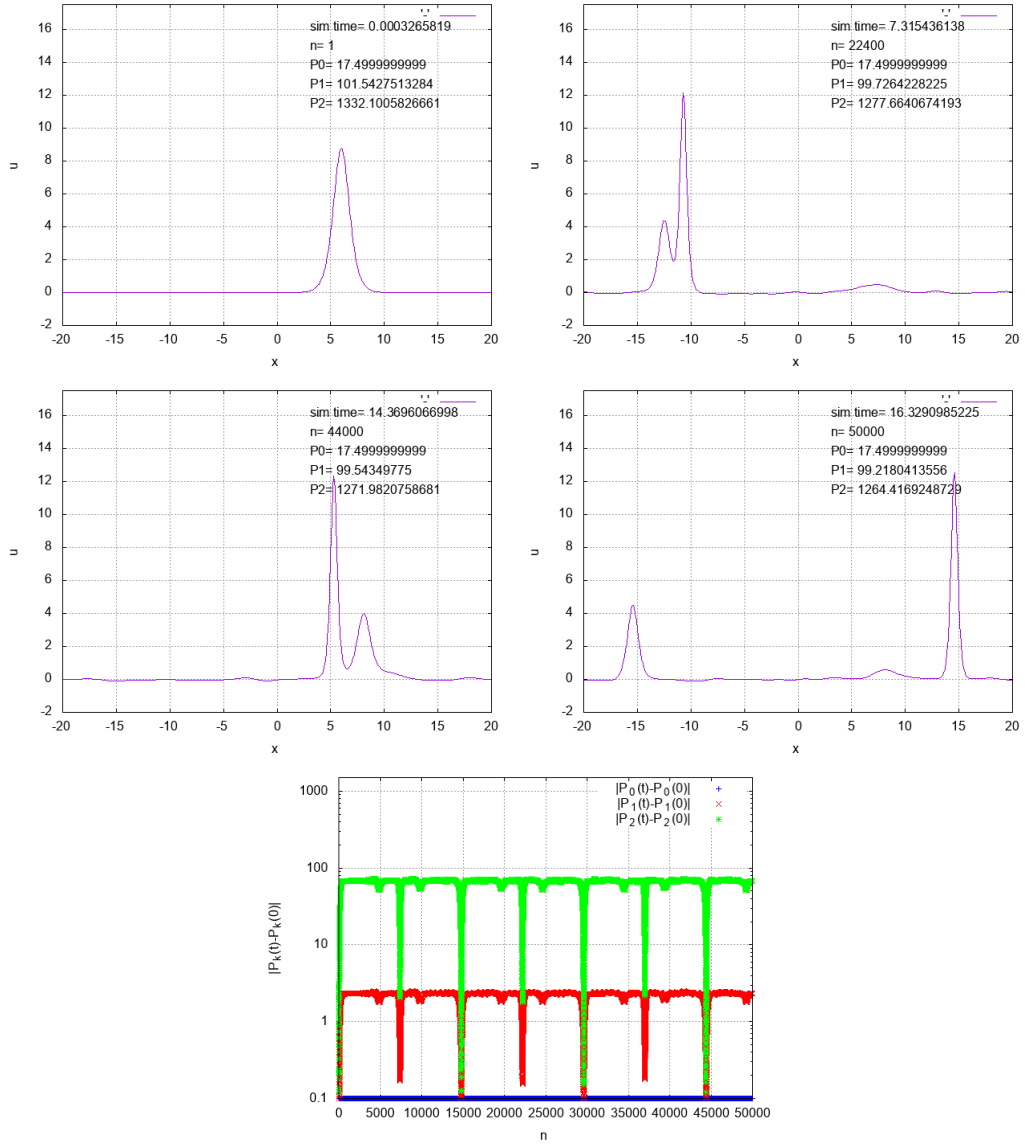

Abbildung 9: Die Solitonwelle bewegt sich nach rechts und teilt sich entsprechend $N = 2{,}5$ in drei weitere Wellen auf. Im letzten Diagramm sieht man deutlich bei welchen Zeitpunkten sich die Wellen wieder vereinigen. Die Werte der Integrale sinken wieder auf Null runter.

6.2 Wassertiefenfunktion h_1

In dieser neuen Betrachtung der Wassertiefe haben die folgenden Dinge verändert: Die Größe des Gitters G erhöht, die Länge L der Stufe im Wasser und dementsprechend die Simulationszeit erhöht. Für die erste Wassertiefenfunktion $h_1(x)$ von Gleichung 12 benutzen wir für die Höhen $h_0 = 0{,}2\,, 0{,}3$

$$\Delta x = 0{,}05\,, -8 < G < 40\,, N = 1\,, x_0 = G \cdot 0{,}1\ .$$

Für größere Höhen benutzen wir

$$\Delta x = 0{,}1\,, -20 < G < 100\,, N = 1\,, x_0 = G \cdot 0{,}1\ .$$

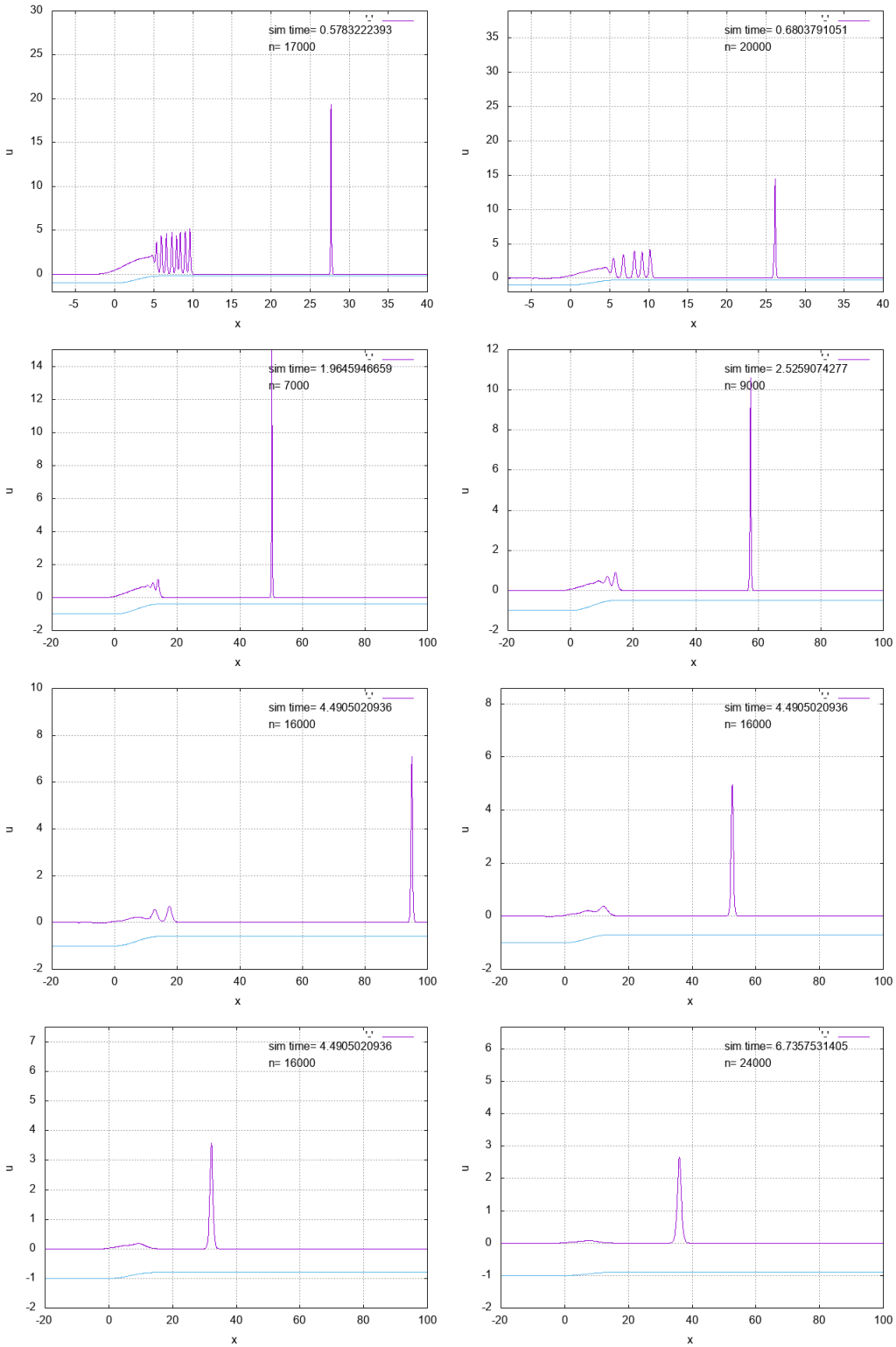

Abbildung 10: Von links nach rechts von oben nach unten, sind die Abbildungen mit aufsteigender Höhe h_0 angeordnet. Es ist immer das Endprodukt der Simulation zu sehen. Die Solitonwelle bewegt sich nach rechts und transformiert sich an der Anhöhe in ein viel größere Welle. Die Amplitude steigt sehr schnell an. Die übrig gebliebene Wellenberg teilt sich weiter in kleinere Welle auf.

6.2.1 Abhängigkeit der Amplitude $A(h_0)$ von der Tiefe h_0

Dazu verwenden wir wieder die gleichen Parameter. Die Schrittzahl wurde anhand der gewählten h_0 gesetzt. Für den linearen Fit erhalten wir

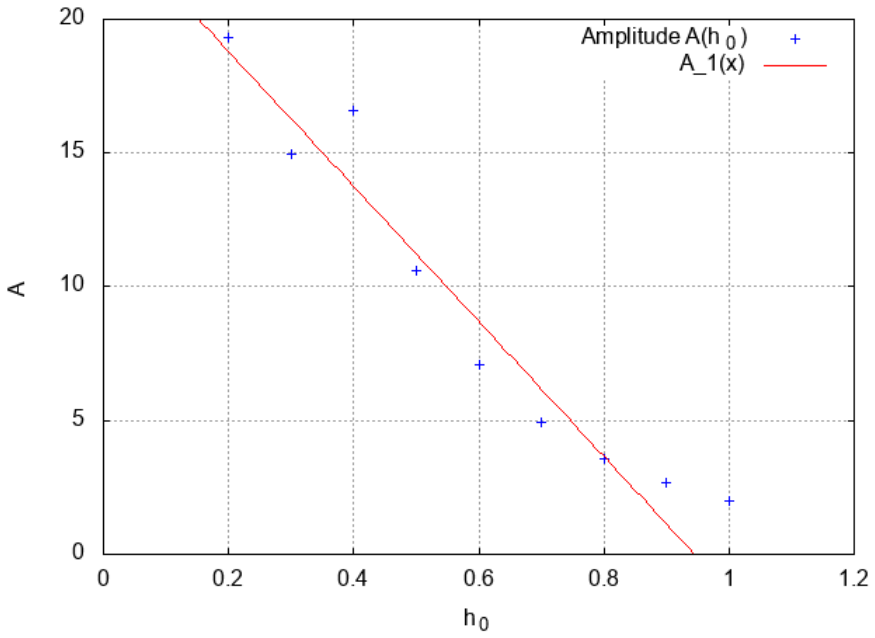

Abbildung 11: Die Amplitude sinkt auf den Wert von $A = 1$ ab. Ein linearer Zusammenhang ist nur schwer zu vermuten.

$$A_1(h_0) = -25{,}23h_0 + 23{,}84 \tag{19}$$

Wenn wir die Werte für h_0 aus den Bereich $0{,}1 < h_0 < 0{,}4$ heraus nehmen und als Fit Funktion die Umkehrfunktion benutzen

$$A_2(h_0) = \frac{c}{h_0 + e} + d$$

bekommen wir

$$A_2(h_0) = \frac{4{,}69}{h_0 - 0{,}17} - 3{,}77\ . \tag{20}$$

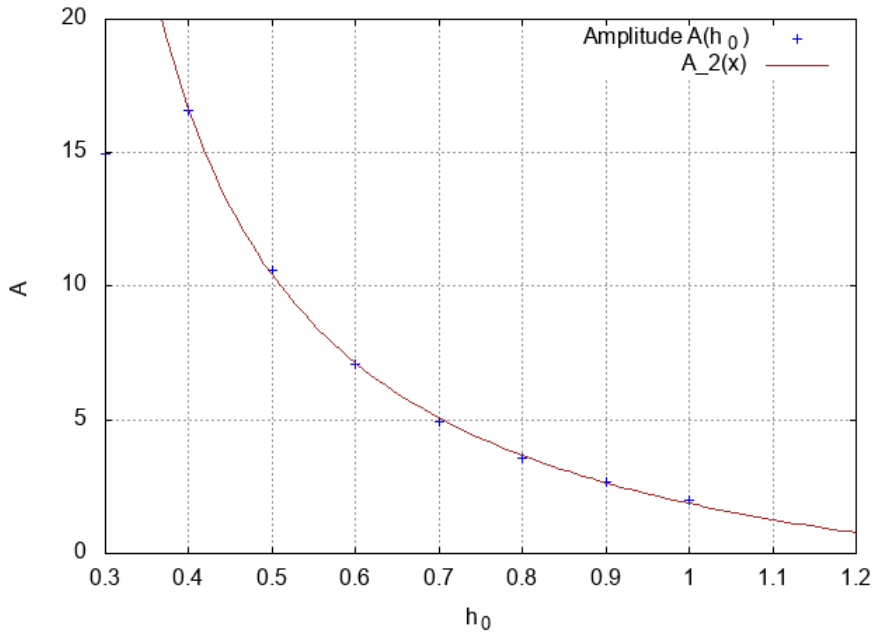

Abbildung 12: Die Amplitudenfunktion $A(h_0)$ ist eine Umkehrfunktion ohne die Werte von$0{,}1 < h_0 < 0{,}4$. Vermutlich spielt hier die Wahl der Schrittweite Δx eine große Rolle. Für die Werte $0{,}1 < h_0 < 0{,}4$ haben wir eine kleinere Schrittweite $\Delta x = 0{,}05$ benutzt.

6.3 Wassertiefenfunktion h_2

Für die zweite Wassertiefenfunktion h_2 von Gleichung 14 nehmen wir die gleichen Veränderungen vor wie im vorherigen Abschnitt. Als Parameter wählen wir

$$\Delta x = 0{,}1\,, -30 < G < 90\,, N = 1\,, x_0 = G \cdot -0{,}6\,, h_0 = [0{,}3, 0{,}4, 0{,}5, 0{,}6]\,.$$

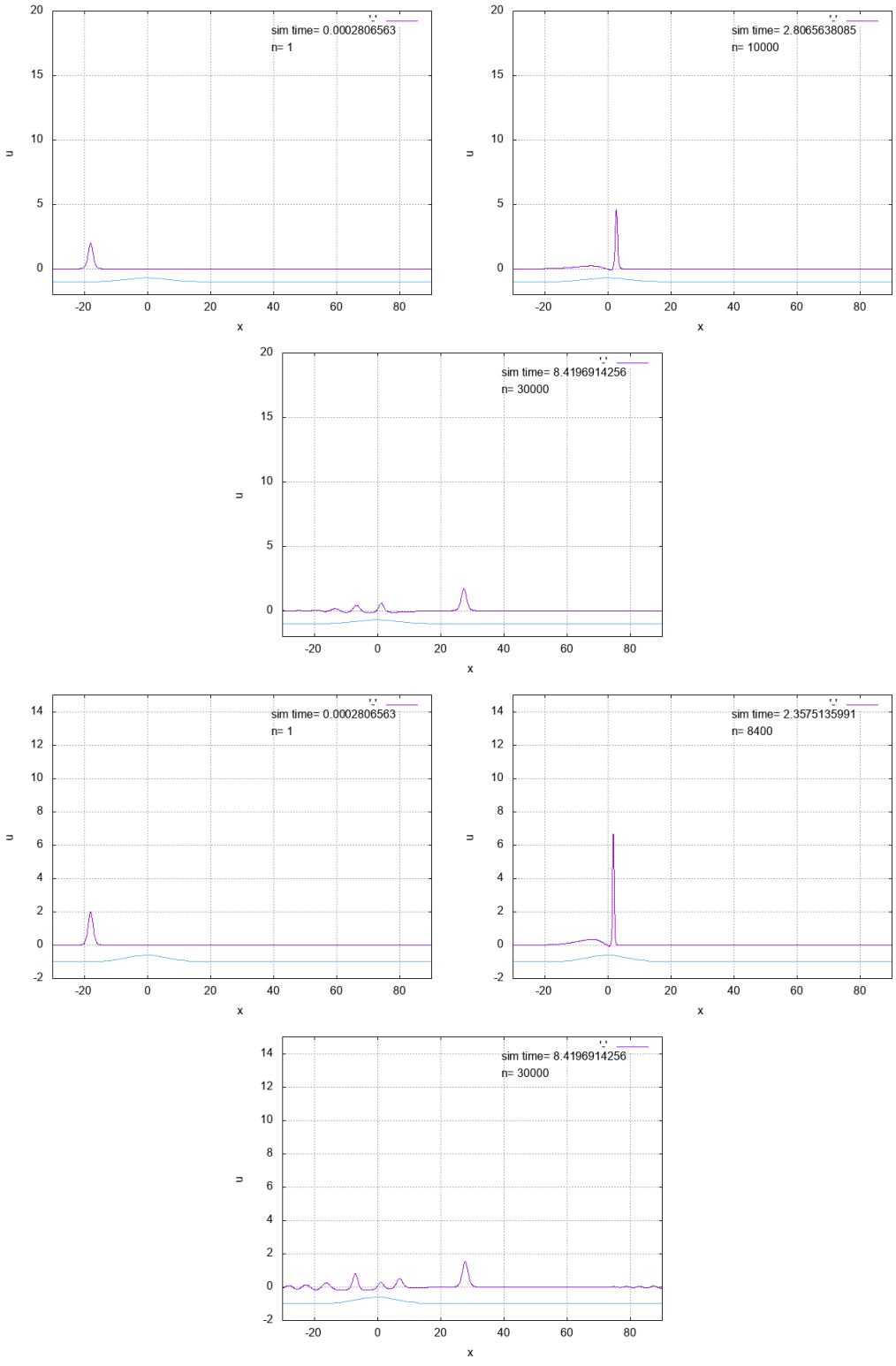

Abbildung 13: Von links nach rechts von oben nach unten, sind die Abbildungen mit aufsteigender Höhe $h_0 = [0{,}3, 0{,}4]$ angeordnet Die Solitonwelle bewegt sich nach rechts und transformiert sich an der Anhöhe in ein viel größere Welle. Die Amplitude steigt sehr schnell an. Die übrig gebliebene Wellenberg teilt sich weiter in kleinere Welle auf. Die Amplitude nimmt mit steigender Höhe zu. Nach der Überquerung der Anhöhe geht die Solitonwelle fasst zurück zu ihrer Ursprungsgröße.

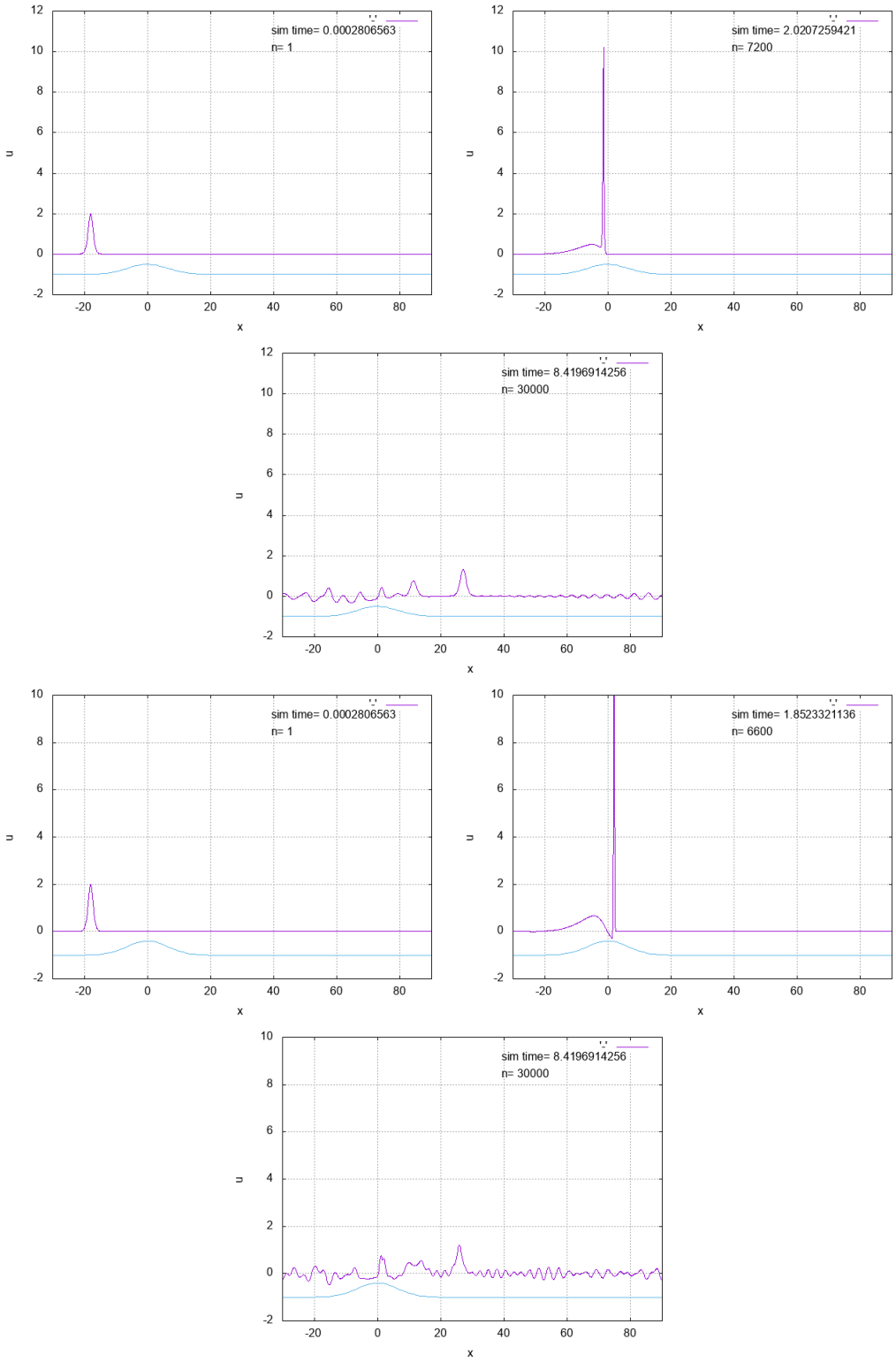

Abbildung 14: Von links nach rechts von oben nach unten, sind die Abbildungen mit aufsteigender Höhe $h_0 = [0{,}5, 0{,}6]$ angeordnet Die Solitonwelle bewegt sich nach rechts und transformiert sich an der Anhöhe in ein viel größere Welle.

7 Programm

Das Programm wurde in der Programmiersprache Julia(`https://julialang.org/`) geschrieben. Auf den nachfolgenden Seiten können sie den gesamten Code einsehen. Zusätzlich ist die gesamte Arbeit mit der Software noweb geschrieben worden. In noweb kann alles in einer Datei gespeichert werden und in Latex die einzelnen Codeabschnitte übersichtlich dargestellt werden. Das Programm kann mit Hilfe des Befehls

```
julia korteweg.jl 0 5000 0.1 5.5 1 0.5 1 0.3 > filled.log
```

gestartet werden. Dabei wird die gesamte Ausgabe in die Datei filled.log geschrieben. Die Plots wurden mit gnuplot erstellt. Die Parameter können direkt über eingeben werden über die Kommandozeile:

```
julia korteweg.jl p1 p2 p3 p4 p5 p6 p7 p8
```

mit

- p1 - Variable `schalter`
- p2 - Variable `n`, Ende der Schleife über n
- p3 - Variable `delta_x`
- p4 - Variable `grenze`, Größe des Gitters G
- p5 - Variable `N`, Variable N
- p6 - Variable `h0`, Tiefe des Wassers h_0
- p7 - Schalter für Dateioutput ja oder nein
- p8 - Startposition x_0

21a ⟨*korteweg.jl* 21a⟩≡

```
#=
Korteweg de Vries Simulationsprogramm
=#
⟨PeriodicCond 21b⟩
⟨Maximum 21c⟩
⟨Init 22a⟩
⟨h Funktion 22b⟩
⟨NumCalc 23a⟩
⟨IntCon 23b⟩
⟨Output 24a⟩
⟨Main 24b⟩
```

21b ⟨*PeriodicCond* 21b⟩≡ (21a)

```
function PeriodicCond(pos)# Periodische Randbedingungen
  temp1 = u[pos,scal-3]
  temp2 = u[pos,scal-2]
  u[pos,1], u[pos,2],u[pos,scal-1], u[pos,scal] =
  temp1, temp2, u[pos,3], u[pos,4]
end
```

21c ⟨*Maximum* 21c⟩≡ (21a)

```
function Maximum(n)
  max = 0
  if n == 1
    for i in 3:(scal-2)
      if max < u[1,i]
        max = u[1,i]
      end
    end
  else
    for i in 3:(scal-2)
      if max < u[3,i]
        max = u[3,i]
      end
    end
  end
  return max
end
```

22a ⟨*Init* 22a⟩≡ (21a)

```
function Init()
  global file = open("P_K-data.txt","w")
  global P0alt =0.
  global P1alt =0.
  global P2alt =0.
  global printer =Int64[0:200:parse(Int64,ARGS[2]);]# Variable zur Steuerung der
  #Ausgabe von Dateien, alle 200 Schritt erfolgt die Ausgabe
  printer[1]=1
  global schalter=parse(Int,ARGS[1])
  global N = parse(Float64,ARGS[5]) # N- Soliton Lösung
  global delta_x = parse(Float64,ARGS[3])
  global delta_t = 2*delta_x/(3*sqrt(3)*(-2*(N*(N+1))+1/(delta_x)^2))*0.7
  global grenze = parse(Float64,ARGS[4]) # Größe des Gitters
  global L = grenze*0.3
  global h0 = parse(Float64,ARGS[6]) # Tiefe des Wassers
  global l = Int64(2)
  global pil = Int(1)
  global x = Float64[-grenze:delta_x:3*grenze;] # x Koordinate
  global scal = length(x)+4# Indexgröße von u festlegen
  global u = (zeros(Float64,3,scal)) # dimensionslose Anhebung
  for i in 3:(scal-2) #Anfangsbedinung anlegen
    u[1,i] = ((N*(N+1))/(cosh(x[i-2]+grenze*(-parse(Float64,ARGS[8]))))^2)
  end
  println("reset")
  println("#$(parse(Int,ARGS[1]))-$(parse(Float64,ARGS[2]))-\\
  $(parse(Float64,ARGS[3]))-$(parse(Float64,ARGS[4]))-\\
  $(parse(Float64,ARGS[5]))-$(parse(Float64,ARGS[6]))-\\
  $(parse(Float64,ARGS[7]))-$(parse(Float64,ARGS[8]))")
  println("# Maxmimum $(Maximum(1))")
  println("set grid")
  println("set xrange [",-grenze,":",3*grenze,"]")
  println("set yrange [",-2,":",N*(N+5)/h0,"]")#4.5 , 10.5 , 18
  println("set xlabel 'x'")
  println("set ylabel 'u'")
end
```

22b ⟨*h Funktion* 22b⟩≡ (21a)

```
function h(pos)
  if (schalter == 1) # h_1(x) Funktion
    if (x[pos] < 0)
      return 1
    elseif (0<=x[pos])&&(x[pos]<=L)
      return 1/2*(1+h0+(1-h0)*cos(pi*(x[pos])/L))
    else
      return h0
    end
  elseif (schalter == 2) # eine Testfunktion
    if (L<=x[pos])&&(x[pos]<=2*L)
      return 1/2*(1+h0+(1-h0)*cos(pi*(x[pos]+L)/L))
    elseif x[pos] > L
      return h0
    elseif (x[pos]<=-grenze+L)&&(x[pos]>=-grenze)
      return 1/2*(1+h0+(1-h0)*cos(pi*(x[pos]-grenze+L)/L))
    else
      return 1
    end
  elseif (schalter == 3) # h_2(x) Funktion
    return  (1-h0*exp(-x[pos]*x[pos]/(L*L)))
  else
    return 1
  end
end
```

23a ⟨*NumCalc* 23a⟩≡ (21a)

```
function NumCalc(n)
   for i in 3:(scal-2) # Zabuski-Kruskal Schema
      if n == 2
         u[2,i]=u[1,i] - delta_t*(((h(i-2))^(-7/4))*
         (u[1,i-1]+u[1,i]+u[1,i+1])*
         (u[1,i+1]-u[1,i-1])/delta_x+(h(i-2)^(1/2))*
         (u[1,i+2]-2*u[1,i+1]+2*u[1,i-1]-u[1,i-2])/
         (2*(delta_x)^3))
         global l=2
      else
         u[3,i]=u[1,i] - 2*delta_t*(((h(i-2))^(-7/4))*
         (u[2,i-1]+u[2,i]+u[2,i+1])*
         (u[2,i+1]-u[2,i-1])/delta_x+(h(i-2)^(1/2))*
         (u[2,i+2]-2*u[2,i+1]+2*u[2,i-1]-u[2,i-2])/
         (2*(delta_x)^3))
         global l=3
      end
   end
end
```

23b ⟨*IntCon* 23b⟩≡ (21a)

```
function IntCon(pos,n)
  P0=Float64(0)
  for i in 3:(scal-2)
    P0=P0+(1/3*(u[pos,i-1]+u[pos,i]+u[pos,i+1]))*delta_x
  end
  P1=Float64(0)
  for i in 3:(scal-2)
    P1=P1+((1/3*(u[pos,i-1]+u[pos,i]+u[pos,i+1]))^2)*delta_x
  end
  P2=Float64(0)
  for i in 3:(scal-2)
    P2=P2+(2*(1/3*(u[pos,i-1]+u[pos,i]+u[pos,i+1]))^3-
      ((u[pos,i+1]-u[pos,i-1])/delta_x/2)^2)*delta_x
  end
  #println("set label 3 \"P0= ",
  # trunc(P0,digits=10)," \" at screen 0.6,0.8")# Anzeige des Integrals im Diagramm
  #println("set label 2 \"P1= ",
  # trunc(P1,digits=10)," \" at screen 0.6,0.75")# Anzeige des Integrals im Diagramm
  #println("set label 1 \"P2= ",
  # trunc(P2,digits=10)," \" at screen 0.6,0.7")# Anzeige des Integrals im Diagramm
  if n>1
    #write(file,"$n $(trunc(abs(Pk[1]-P0),digits=10)) $(trunc(abs(Pk[2]-P1),digits=10)) $(trunc(abs(Pk[3]-P2)
    write(file,"$n $(abs(Pk[1]-P0)) $(abs(Pk[2]-P1)) $(abs(Pk[3]-P2)) \n")
    #write(file,"$n $(abs(Pk[1]-P0)) $(Pk[1]) \n")
  else
    global Pk=zeros(3)
    Pk[1]=P0
    Pk[2]=P1
    Pk[3]=P2
    write(file,"$n 0 0 0 \n")
    #write(file,"$n $(Pk[1]) $(Pk[2]) $(Pk[3]) \n")
  end
end
```

24a ⟨*Output* 24a⟩≡ (21a)

```
function Output(pos, tsim, treal, n) # Ausgabe ins Terminal
  if (pil <= length(printer)) & ( parse(Int,ARGS[7])> 0 )
    if (n == printer[pil])
      #println("set term qt;")
      println("set term png; set output sprintf('korteweg_frame$n-$(parse(Int,ARGS[1]))-$(parse(Float64,ARGS[3
      global pil = pil+1
      println("set label 5 \"sim time= ",
      trunc(tsim,digits=10)," \" at screen 0.6,0.9")# Anzeige der Zeit im Diagramm
      println("set label 6 \"n= ",n," \" at screen 0.6,0.85")
      if (schalter == 1) # Anzeige der h_1,2 Funktionen im Diagramm
      println("plot '-' w l, [-$grenze:0] -1 linecolor 3 notitle, [0:$L] \\
      -(1+$(h0)+(1-$(h0))*cos(pi*x/$(L)))/2 linecolor 3 notitle, \\
      [$L:$(5*grenze)] -$(h0) linecolor 3 notitle")
      elseif (schalter == 2)
        println("plot '-' w l, [$L:$(2*L)] (1+$(h0)+(1-$(h0))*cos((pi*(x+$L))/$L))/2-1\\
        linecolor 3 notitle, [-$(grenze):$(-grenze+L)] (1+$(h0)+\\
        (1-$(h0))*cos(pi*(x-$(grenze+L))/$L))/2-1 linecolor 3 notitle, \\
        [$(-grenze+L):$L] 1-1 linecolor 3 notitle, [$(2*L):$grenze] $(h0)-1 linecolor 3 notitle")
      elseif (schalter == 3)
        println("plot '-' w l, [-$grenze:$(3*grenze)] -(1-$(h0)*exp(-x*x/$(L*L))) linecolor 3 notitle")
      else
        println("plot '-' w l")
      end
      global y=zeros(scal-4)
      for i in 3:(scal-2) # Werteausgabe
        println(x[i-2]," ",u[pos,i])
      end
      println("elapsed time: 0.672879991 seconds")
    end
  end
  if n == (parse(Int,ARGS[2])) # Maximum Anzeige und Berechnung
    println("# Maxmimum $(Maximum(n))")
  end
  #close("P_K.txt")
end
```

24b ⟨*Main* 24b⟩≡ (21a)

```
for n in 1:(parse(Int,ARGS[2]))
   if n == 1
      global treal=0.
      global tsim=0.
      Init()
      IntCon(n,n)
      PeriodicCond(n)
      tsim=delta_t
      Output(n, tsim, treal, n)
   else
      NumCalc(n)
      PeriodicCond(l)
      if n>2 #Vertauschung der Reihen in dem Array der Anhebung
         for k in 1:scal
            temp3=u[2,k]
            u[2,k]=u[3,k]
            u[1,k]=temp3
         end
      end
      tsim=tsim+delta_t
      #println("elapsed time: 0.672879991 seconds")
      IntCon(l,n)
      Output(l, tsim, treal, n)
   end
end
```