Korteweg–de Vries solitoniyhtälön diskretointi- ja aikaintegrointimenetelmien vertailua

Pro Gradu
Turun yliopisto
Fysiikan laitos
Fysiikka
2008
LuK. Mikko Laine
Tarkastajat:
Prof. Jarmo Hietarinta
Dos. Tom Kuusela

TURUN YLIOPISTO
Fysiikan laitos

Laine, Mikko Korteweg–de Vries solitoniyhtälön diskretointi- ja aikaintegrointimenetelmien vertailua

Pro Gradu, 48 s., 9 liites.
Fysiikka
Joulukuu 2008

Solitoni on tunnettu ilmiönä jo 1800-luvun alkupuolelta lähtien. Se on eräänlainen muotonsa säilyttävä ja vakionopeudella etenevä aalto. 1800-luvun loppupuolella esitettiin osittaisdifferentiaaliyhtälö kuvaamaan tällaista matalassa ja kapeassa kanavassa esiintynyttä solitoniaaltoa. Tätä Kortewegin ja de Vries'n mukaan nimettyä osittaisdifferentiaaliyhtälöä tutkivat numeerisesti ensimmäistä kertaa Zabusky ja Kruskal vuonna 1965.

Osittaisdifferentiaaliyhtälöiden ratkaisuun tarvitsee usein käyttää numeerisia menetelmiä. Tämän työn alkupuoli käsittelee yleisesti tarvittavia matemaattisia menetelmiä sekä KdV-yhtälön analyyttistä tarkastelua. Loppupuolella tutkitaan KdV-yhtälön mallintamista tietokoneen avulla. Zabuskyn ja Kruskalin käyttämien menetelmien lisäksi kokeillaan montaa muutakin tapaa KdV-yhtälön mallintamiseen. Näistä menetelmistä vertaillaan laskentatehokkuutta sekä menetelmän tarkkuutta.

Zabuskyn ja Kruskalin käyttämä paikkadiskretointi todettiin mallinnuksissa tarkimmaksi, mutta ei kuitenkaan mallinnusaikaa tarkastellen tehokkaimmaksi. Aikaintegroinneista Runge-Kutta-menetelmät todettiin parhaiksi.

Menetelmien vertailun lisäksi niistä parhaiksi havaittuja sovellettiin muutaman erikoistapauksen, kuten kolmen aallon törmäyksen, mallintamiseen.

Avainsanat: Korteweg-de Vries, KdV-yhtälö, solitoni, mallintaminen, aikaintegrointi, diskretointi, Runge-Kutta, Low Storage Runge-Kutta

Sisältö

1 Johdanto

Solitoni tarkoittaa itseään ylläpitävää yksittäistä aaltoa tai aaltopakettia, joka etenee vakionopeudella ja säilyttää muotonsa edetessään ja jopa törmätessä toisen aallon kanssa.

Solitoni-ilmiötä tutki ensimmäisenä John Russel. Vuonna 1834 hän havaitsi matalassa ja tasalevyisessä kanavassa aallon, joka eteni muotoaan muuttamatta ja heikkenemättä hyvin pitkään. Russel kutsui ilmiötä nimellä ”Wave of Translation” tai ”Solitary Wave” johtuen aallon yksinäisestä luonteesta, ja siitä, että väliainetta siirtyi solitoniaallon mukana toisin kuin tavallisilla aalloilla. Ilmiötä tutkivat niin Russel kuin muutamat muutkin 1800-luvulla, mutta Diederik Kortewegin ja Gustav de Vries'n vuonna 1895 julkaiseman teoreettisen tarkastelun [1] jälkeen tutkiminen hiipui. Solitoniaaltoa mallintava osittaisdifferentiaaliyhtälö

$$u_t + 6uu_x + u_{xxx} = 0, \tag{1.1}$$

jota Korteweg ja de Vries tutkivat, on jälkeenpäin nimetty heidän mukaansa, vaikka sen olikin Joseph Boussinesq esitellyt jo vuonna 1877 artikkelissaan [2] ennen lähdettä [1]. Yhtälöön (1.1) viitataan jatkossa lyhenteillä KdV tai KdV-yhtälö.

Hyvin usein osittaisdifferentiaaliyhtälöille ei löydy analyyttistä ratkaisua, vaan niitä käsitellessä on käytettävä numeerisia menetelmiä. Tällöin pitää pohtia esimerkiksi differentiaaliyhtälön diskretointia ja sen numeerista integrointia ajan suhteen. KdV on valittu tähän työhön käsiteltäväksi yhtälöksi, koska sille voidaan johtaa analyyttiset ratkaisut yhden tai useamman aallon tapaukselle. Lisäksi KdV-yhtälölle saadaan laskettua suureita, jotka säilyvät mentäessä ajassa eteenpäin.

Tässä työssä tutkitaan erilaisten numeeristen menetelmien soveltamista KdV-yhtälön numeeriseen mallintamiseen. Tarkoituksena on tarkastella hieman erilaisia differentiaaliyhtälön diskretointimenetelmiä ja aikaintegrointeja, kartoittaa niiden helppokäyttöisyyttä ja suoritustehokkuutta sekä tarkastella kuinka hyvin ne soveltuvat KdV-yhtälön mallintamiseen. Soveltuvuutta testataan tarkastelemalla mallinnuksen aikana KdV-yhtälön säilyviä suureita ja mallinnuksen lopuksi vertaamalla aallon muotoa analyyttisen ratkaisun antamaan muotoon.

Työssä käytetään päälähteenä Norman Zabuskyn ja Martin Kruskalin artikkelia ”Interaction of 'Solitons' in a Collisionless Plasma and the Recurrence of Initial States” [3]. Artikkelissaan vuonna 1965 Zabusky ja Kruskal tarkastelivat KdV-yhtälöä numeerisesti mallintamalla ja huomasivat aaltojen törmäyksissä erikoista käytöstä. Törmätessään kaksi solitoniaaltoa säilyttävät muotonsa ja nopeutensa, mutta niillä havaitaan vaihesiirtymää, jossa hitaampi jää jälkeen ja nopeampi hyppää edelle. Sittemmin KdV-yhtälölle on löydetty analyyttisiä ratkaisuja, jotka kuvaavat useamman kuin yhden aallon etenemistä. Zabuskyn ja Kruskalin diskretointia käsitellään työssä hyväksi havaittuna tunnettuna menetelmänä. Sen lisäksi kokeillaan itse johdettujen diskretointien toimivuutta ja

verrataan niitä Zabuskyn ja Kruskalin diskretointiin.

Toisena päälähteenä käytettiin Burdenin ja Fairesin kirjaa ”Numerical Analysis” [4]. Kirjaa käytettiin aikaintegrointimenetelmien lähteenä ja sitä apuna käyttäen johdettiin Eulerin menetelmä sekä neljännen kertaluvun Runge-Kutta. Kirjasta otettiin myös muutama muu aikaintegrointimenetelmä mukaan testattavaksi.

Työtä varten tekijä on johtanut käytetyt KdV-yhtälön diskretoinnit ja osan aikaintegroinneista sekä kirjoittanut mallinnuksissa käytetyn ohjelman kokonaisuudessaan. Ohjelman keskeiset moduulit ovat nähtävillä liitteessä B.

2 Matemaattisia menetelmiä

Tietokone osaa laskea tehokkaasti yhteen- ja kertolaskuja, mutta symbolinen laskenta sillä on hankalampaa. Differentiaaliyhtälön ratkaisu saadaan integroimalla ja se onnistuu harvalle yhtälölle analyyttisesti. Näin on usein jopa välttämätöntä käyttää numeerisia menetelmiä. Toisin kuin integrointi, on derivaatan ratkaisu analyyttisesti helppoa ja vaikeimmillaankin mahdollista. Numeerisessa tarkastelussa tämä kuitenkin on hieman hankalampaa ja derivaatat pitääkin ensin muuttaa peruslaskutoimituksia käyttävään muotoon diskretoimalla yhtälö.

Käytettäessä numeerisia menetelmiä osittaisdifferentiaaliyhtälön (ODY) tarkasteluun täytyy ottaa huomioon edellä mainitun lisäksi toinenkin rajoitus. Jatkuvien arvojen käsittely on mahdotonta ja tästä syystä yhtälöt ja funktiot tulee korvata diskreeteillä vastineillaan. Myös jatkuvat paikka- ja aikaulottuvuudet korvataan diskreeteillä arvoilla. Paikkaulottuvuutta voidaan käsitellä hilana, josta tunnetaan arvot pisteissä x_0+nh, jossa h on hilavakio ja $n \in \mathbb{Z}$. Aikaulottuvuus diskretoidaan samoin, mutta hilavakion sijaan puhutaan aika-askeleesta ja arvoja käsitellään vain arvoa t_0 suuremmilla ajanhetkillä.

Esimerkiksi funktiosta $f(x, y, t)$ saadaan tällä tavoin $f(x_0 + nh, y_0 + mh, t_0 + kh)$ $\forall\, m,n,k \in \mathbb{Z}$. Tässä työssä käsittelemme tilaa $u(t, x)$, jossa $t = t_0+n\Delta t$ ja $x = x_0+mh$. Tilaa mallinnetaan ajassa vain eteenpäin, joten $n \in \mathbb{N}$, ja mallinnettava hila on äärellinen, joten m on kokonaisluku väliltä $[0, L]$, jossa L on hilapisteiden määrä. Hilavakiota h pidetään kiinteänä kun taas aika-askelta Δt muutetaan usein. Funktio $u(t, x)$ voi saada mitä vain arvoja tietokoneen suoman tarkkuuden rajoissa.

2.1 Yhtälön diskretointi

Differentiaaliyhtälön diskretoinnissa on kyse siitä, että yhtälössä esiintyvät derivaatat esitetään peruslaskutoimitusten ja diskreettien hilapisteiden avulla. Analyyttisesti käsitellen derivointi on yksikäsitteistä ja suoraviivaista, mutta numeerisesti sen voi tehdä monella tapaa, joista toiset ovat parempia ja toiset huonompia. Derivaatta tarkoittaa periaatteessa funktion kasvunopeutta tietyllä hetkellä ja sen ratkaisu numeerisesti on suhteellisen helppoa suoraan derivaatan määritelmää hyödyntäen.

Derivaatan määritelmä on

$$f'(x) = \lim_{h \to 0} \left(\frac{f(x+h) - f(x)}{h} \right)$$

ja keskimääräinen derivaatta alueella $[x_0, x_0 + h]$ saadaan laskettua kaavalla

$$f'_{avg}(x_0) = \frac{f(x_0+h) - f(x_0)}{h}. \tag{2.1}$$

Numeerista laskentaa varten ei anneta h:n lähestyä nollaa, vaan se pidetään pienenä.

Luonnollista on ajatella, että mitä pienempi h, sitä lähempänä todellista derivaattaa tulos on.

Yhtälö (2.1) ei välttämättä kuitenkaan anna aivan sitä, mitä haluttiin. Jos lasketaan $\frac{f(x_0+h)-f(x_0)}{h}$, niin saadaankin $f'_{avg}(x_0)$:n sijaan luku, joka kuvaa parhaiten derivaattaa puoli askelta sivussa x_0:sta eli $f'_{avg}(x_0 + \frac{h}{2})$. Paikka-askeleen pituudesta riippuu, kuinka kaukana tämä on todellisesta, mutta käytetyllä askeleellakin voidaan parantaa tarkkuutta.

Jos otetaan pisteet kohdista $x_0 - h$ ja $x_0 + h$, saadaan keskimääräiseksi derivaataksi

$$f'_{avg}(x_0) = \frac{f(x_0 + h) - f(x_0 - h)}{2h}. \tag{2.2}$$

Nyt approksimaatio ei puolla kumpaankaan suuntaan, mutta väli, jolta se otetaan on kasvanut. Tämän voi korjata vielä ottamalla pisteet puolen askeleen päästä toivotusta pisteestä, eli

$$f'_{avg}(x_0) = \frac{f(x_0 + \frac{h}{2}) - f(x_0 - \frac{h}{2})}{h}. \tag{2.3}$$

Yhtälöä ei kuitenkaan voida suoraan käyttää derivaatan laskemiseen, jos muuttujan x arvot tunnetaan vain hilapisteissä, mutta esimerkiksi toisen derivaatan laskemiseen sitä voidaan soveltaa.

2.2 Eulerin menetelmä

Fysikaalisissa ongelmissa ODY:n muuttujina ovat usein paikka- ja aikaulottuvuudet. Ajan suhteen ensimmäisen kertaluvun yhtälö on tyyppiä

$$u_t = f(u(t,x), u_x(t,x), u_{xx}(t,x), u_{xxx}(t,x), \ldots).$$

Merkitään oikeaa puolta $f(t,x)$:llä ja approksimoidaan vasenta puolta yhtälön (2.1) mukaisella erotusosamäärällä. Saadaan

$$\frac{u(t + \Delta t, x) - u(t,x)}{\Delta t} = f(t,x). \tag{2.4}$$

Tästä voidaan ratkaista $u(t + \Delta t, x)$.

$$u(t + \Delta t, x) = u(t,x) + \Delta t \cdot f(t,x). \tag{2.5}$$

Näin voidaan ratkaista u:n arvo seuraavalla ajanhetkellä. Tätä kutsutaan *Eulerin menetelmäksi*.

Eulerin menetelmällä ei kuitenkaan saada kovinkaan hyviä tuloksia. Menetelmän virhe on suoraan verrannollinen aika-askeleeseen Δt eli kyseessä on ns. ensimmäisen kertaluvun menetelmä. Tästä syystä pitäisi käytettävän aika-askeleen olla erittäin pieni. Pieni

askel tarkoittaa kuitenkin pitkää simulaatioaikaa.

Eulerin menetelmää voidaan käyttää myös yhtälön (2.2) kaltaisesti. Merkitsemällä

$$\frac{u(t+\Delta t,x)-u(t-\Delta t,x)}{2\Delta t}=f(t,x)$$

saadaan Eulerin menetelmän yhtälöksi

$$u(t+\Delta t,x)=u(t-\Delta t,x)+2\Delta t\cdot f(t,x). \tag{2.6}$$

Nyt funktion uusi arvo ratkaistaan käyttämällä yhtälön oikeaa puolta kohdassa (t,x), ja sen lisäksi vanhaa arvoa kohdasta $(t-\Delta t,x)$.

2.3 Neljännen kertaluvun Runge-Kutta

Aikaintegrointia voidaan parantaa approksimoimalla u:n aikaderivaattaa useammassa pisteessä yhden sijaan. Neljässä pisteessä laskettaessa menetelmä voidaan kirjoittaa muotoon

$$\begin{aligned}u(t+\Delta t,x)=u(t,x)+\Delta t\,(&\omega_1 f(u(t+\alpha_1,x+\beta_1))+\omega_2 f(u(t+\alpha_2,x+\beta_2 k_1))+\\&\omega_3 f(u(t+\alpha_3,x+\beta_3 k_2))+\omega_4 f(u(t+\alpha_4,x+\beta_4 k_3)))\,,\end{aligned} \tag{2.7}$$

jossa k_i lasketaan kaavalla

$$k_i=\omega_i f(u(t+\alpha_i,x+\beta_i k_{i-1}))\quad\text{ja}\quad k_0=1. \tag{2.8}$$

Nyt siis kertoimilla α_i määrätään, mistä päin väliä $[t,t+1]$ pisteet valitaan. Kertoimilla β_i otetaan huomioon funktion arvon muuttuminen aikaa muutettaessa. Kertoimet ω_i määräävät, millä painotuksella mikäkin piste huomioidaan lopullista arvoa laskettaessa.

Funktiot (2.8) voidaan korvata funktion $f(u)$ osittaisderivaatoilla pisteessä (t,x) soveltaen Taylorin sarjakehitelmää seuraavasti:

$$\begin{aligned}\omega_i f(u(t+\alpha_i,x+\beta_i k_{i-1}))=\omega_i f(u(t,x))&+\omega_i\alpha_i\frac{\partial}{\partial t}f(u(t,x))\\&+\omega_i\beta_i k_{i-1}\frac{\partial}{\partial x}f(u(t,x))+\omega_i R_i(u(t+\alpha_i,x+\beta_i k_{i-1})),\end{aligned} \tag{2.9}$$

jossa

$$R_i(u(t+\alpha_i,x+\beta_i k_{i-1}))=\frac{\alpha_i^2}{2}\frac{\partial^2}{\partial t^2}f(u(\xi,\eta))+\alpha_i\beta_i\frac{\partial^2}{\partial t\partial x}f(u(\xi,\eta))+\frac{\beta_1^2}{2}\frac{\partial^2}{\partial x^2}f(u(\xi,\eta))$$

joillain arvoilla ξ väliltä $[t,t+\alpha_i]$ ja η väliltä $[x,x+\beta_i k_{i-1}]$.

Sovelletaan nyt yhtälöä (2.9) kaavaan (2.7) niin, että lausekkeissa esiintyy $(\Delta t)^4$ saak-

ka vain funktio f ja sen osittaisderivaattoja laskettuna pisteessä (t, x). Tätä suurempien kertaluokkien termeillä ei ole merkitystä, sillä tavoitteenamme on neljännen kertaluvun menetelmä.

Toisaalta Eulerin menetelmän tarkkuutta voidaan parantaa Taylorin sarjakehitelmällä

$$\frac{\mathrm{d}}{\mathrm{d}t}u(t,x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(u(t,x))}{n!} \cdot (\Delta t)^n \tag{2.10}$$

Monesti mitä pidemmälle sarjaa lasketaan, sitä tarkempi tulos saadaan. Nyt halutaan neljännen kertaluvun menetelmä, joten lasketaan termejä sen mukaisesti. Kehitelmän avulla saadaan yhtälöstä (2.5)

$$u(t+\Delta t) = u(t) + \Delta t\, f(u(t,x)) + \frac{(\Delta t)^2}{2} f'(u(t,x)) + \frac{(\Delta t)^3}{6} f''(u(t,x)) + \frac{(\Delta t)^4}{24} f'''(u(t,x)) + \mathcal{O}((\Delta t)^5), \tag{2.11}$$

jossa $\mathcal{O}((\Delta t)^5)$ tarkoittaa jäljelle jäävää epätarkkuutta ja sen kertaluokkaa. Huomataan, että yhtälössä tarvitsisi laskea derivaatan $f'(u(t,x))$ lisäksi sen toinen ja kolmas derivaatta. Tämä voi olla analyyttisesti työlästä ja numeerisesti tekemällä sitä varten tarvitsisi pitää mukana tarvittava määrä vanhoja arvoja ja laskea niistä derivaattoja.

Sovelletaan nyt yhtälöön (2.11) kokonaisdifferentiaalin kaavaa

$$f'(u(t,x)) = \frac{\mathrm{d}}{\mathrm{d}t} f(u(t,x)) = \frac{\partial}{\partial t} f(u(t,x)) + \frac{\partial}{\partial x} f(u(t,x)) \cdot u'(t,x),$$

jossa $u'(t,x) = f(u(t,x))$. Merkitään vielä osittaisderivaattoja alaindeksein t ja x, niin voidaan esittää ylläoleva yhtälö lyhyemmin muodossa

$$f'(u(t,x)) = f_t(u(t,x)) + f_x(u(t,x)) \cdot f(u(t,x)). \tag{2.12}$$

Tätä yhtälöä käyttäen korvataan yhtälössä (2.11) esiintyvät f:n derivaatat sen osittaisderivaatoilla.

Nyt olemme saaneet kaksi yhtälöä, eli (2.7) ja (2.11) johdettuna samanlaisiksi. Yksinkertaisuuden vuoksi oletetaan, että $\alpha_1 = 0$ ja $\beta_1 = 0$. Verrataan yhtälöistä muita kertoimia ja saadaan seuraavanlaiset ehdot kertoimille α_1, β_i ja ω_1.

$$
\begin{aligned}
f: &\quad \omega_2 + \omega_3 + \omega_4 = 1,\\
f_t: &\quad \omega_2\alpha_2 + \omega_3\alpha_3 + \omega_4\alpha_4 = \frac{\Delta t}{2},\\
f_{tt}: &\quad \omega_2\alpha_2^2 + \omega_3\alpha_3^2 + \omega_4\alpha_4^2 = \frac{(\Delta t)^2}{3}\\
f_{ttt}: &\quad \omega_2\alpha_2^3 + \omega_3\alpha_3^3 + \omega_4\alpha_4^3 = \frac{(\Delta t)^2}{4}\\
f_t f_x: &\quad \omega_3\alpha_2\beta_3 + \omega_4\alpha_3\beta_4 = \frac{(\Delta t)^2}{6},\\
f_t f_{tx}: &\quad \omega_3\alpha_2\alpha_3\beta_3 + \omega_4\alpha_3\alpha_4\beta_4 = \frac{(\Delta t)^3}{8},\\
f_{tt} f_x: &\quad \omega_3\alpha_2^2\beta_3 + \omega_4\alpha_3^2\beta_4 = \frac{(\Delta t)^3}{12},\\
f_x^2 f_t: &\quad \omega_4\alpha_2\beta_3\beta_4 = \frac{(\Delta t)^3}{24}
\end{aligned}
$$

Yhtälöitä on vähemmän kuin muuttujia, joten yhtälöryhmälle ei saada yksikäsitteistä ratkaistua. Valitaan $\alpha_2 = \beta_2 = \frac{1}{2}\Delta t$. Näin saadaan ratkaistua muuttujille arvot

$$
\begin{aligned}
\omega_1 &= \frac{1}{6}, \quad \alpha_1 = 0, \qquad \beta_1 = 0,\\
\omega_2 &= \frac{2}{6}, \quad \alpha_2 = \frac{1}{2}\Delta t, \quad \beta_2 = \frac{1}{2}\Delta t,\\
\omega_3 &= \frac{2}{6}, \quad \alpha_3 = \frac{1}{2}\Delta t, \quad \beta_3 = \frac{1}{2}\Delta t,\\
\omega_4 &= \frac{1}{6}, \quad \alpha_4 = \Delta t, \qquad \beta_4 = \Delta t.
\end{aligned}
\tag{2.13}
$$

Kun nämä muuttujien arvot sijoitetaan yhtälöön (2.7), niin saadaan yleisimmin käytetty neljännen kertaluvun Runge-Kutta menetelmä (RK4). Menetelmän etuja muihin muuttujien (2.13) arvoihin nähden ovat sen laskentatehokkuus, yksinkertaisuus ja hyvä tarkkuus [4]. Usein menetelmä kirjoitetaan algoritmin muotoon seuraavasti:

$$
\begin{aligned}
k_1 &= f(t, u)\\
k_2 &= f(t + \frac{1}{2}\Delta t, u + \frac{1}{2}\Delta t k_1)\\
k_3 &= f(t + \frac{1}{2}\Delta t, u + \frac{1}{2}\Delta t k_2)\\
k_4 &= f(t + \Delta t, u + \Delta t k_3),
\end{aligned}
$$

Nyt voidaan laskea arvo hetkellä $t + 1$ yhtälöllä

$$u(t + \Delta t, x) = u(t, x) + \frac{\Delta t}{6} \cdot (k_1 + 2k_2 + 2k_3 + k_4)\,. \tag{2.14}$$

Eulerin menetelmän virheen ollessa suoraan verrannollinen aika-askeleen pituuteen Δt on neljännen kertaluvun Runge-Kuttan virhe verrannollinen $(\Delta t)^4$:n. Toisin sanoen Eulerin menetelmällä täytyy käyttää kolmea kertaluokkaa pienempää aika-askelta saavuttaakseen saman tarkkuuden kuin RK4:llä. Simulaation kesto on taas kääntäen verrannollinen aika-askeleen pituuteen, joten RK4 on mallintaessa kolmea kertaluokkaa Eulerin menetelmää nopeampi. Kuitenkin huomioon pitää ottaa vielä se, että RK4 vaatii nelinkertaisen määrän laskutoimituksia integrointia kohden Eulerin menetelmään verrattuna. Virheen kertaluokan vaikutus mallinnuksen kokonaiskestoon on kuitenkin huomattavasti tätä suurempi.

3 Solitoniyhtälön tutkiminen analyyttisesti

3.1 Yksittäinen aalto

Korteweg-de Vries yhtälön yksiaaltoisen ratkaisun voi löytää muun muassa seuraavasti. Yhtälön (1.1) ratkaisuksi halutaan tasaisella nopeudella liikkuva muuttumaton aalto, joten kokeillaan yritettä $u = f(x - vt)$. Halutaan myös huippu kohtaan $f(0)$, jolloin siis huipun paikka saadaan kaavasta $x = vt$.

Tehdään muuttujanvaihto $z = x - vt$ ja kirjoitetaan KdV-yhtälö uusiksi

$$f''' + 6ff' - vf' = 0.$$

Integroidaan tämä muuttujan z suhteen

$$f'' + 3f^2 - vf + C = 0.$$

Halutaan, että $f \to 0$, kun $|x| \to \infty \quad \forall t$, joten myös $|z| \to \infty$. Täten siis $C = 0$. Nyt kerrotaan yhtälö derivaatalla f' ja saadaan

$$f''f' + 3f^2f' - vff' = 0.$$

Integroidaan tämä muuttujan z suhteen

$$\frac{1}{2}\left(f'\right)^2 + f^3 - \frac{v}{2}f^2 + D = 0.$$

Samoin kuin C edellä, on integroimisvakio D myös nolla. Kerrotaan yhtälö kahdella ja otetaan siitä neliöjuuri

$$\frac{\partial f}{\partial z} = \pm\sqrt{vf^2 - 2f^3}.$$

Separoidaan tämä ja lasketaan integraali

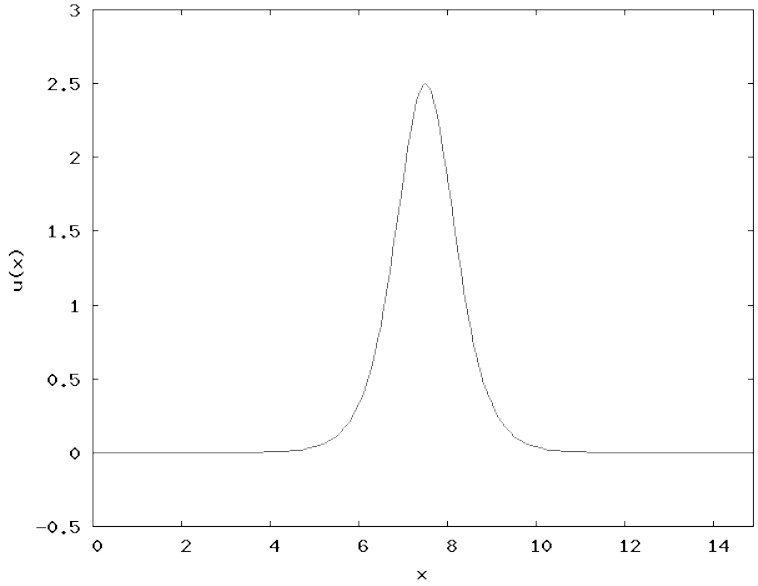
$$\int \frac{\mathrm{d}f}{f\sqrt{v - 2f}} = \int \mathrm{d}z.$$

Otetaan käyttöön uusi funktio $h(x)$ seuraavasti

$$f = \frac{v/2}{h^2}, \quad \mathrm{d}f = -v\frac{\mathrm{d}h}{h^3}.$$

Sijoitetaan tämä integraaliin ja sievennetään

$$\int \mathrm{d}z = \int \frac{-v\frac{\mathrm{d}h}{h^3}}{\frac{v/2}{h^2}\sqrt{v - \frac{v}{h^2}}} = \int -\frac{2\,\mathrm{d}h}{h\sqrt{v - \frac{v}{h^2}}} = -\frac{2}{\sqrt{v}}\int \frac{\mathrm{d}h}{\sqrt{h^2 - 1}}.$$



Kuva 3.1. Ratkaisun (3.1) kuvaaja. Nopeutena on $v = 5{,}0$ ja alkusijaintina $x_0 = 7{,}5$.

Merkitään nyt

$$h = \cosh\, p, \quad \mathrm{d}h = \sinh\, p\, \mathrm{d}p$$

ja muistetaan, että $\cosh^2 p - 1 = \sinh^2 p$. Sijoitetaan nämä integraaliin, sievennetään ja suoritetaan integrointi

$$\int \mathrm{d}z = -\frac{2}{\sqrt{v}} \int \frac{\sinh\, p\, \mathrm{d}p}{\sqrt{\cosh^2 -1}} = -\frac{2}{\sqrt{v}} \int \frac{\sinh\, p\, \mathrm{d}p}{\sinh\, p}$$

$$\Rightarrow p = -\frac{\sqrt{v}}{2}\,(z - z_0)\,,$$

$$\Rightarrow h = \cosh\left(\frac{\sqrt{v}}{2}\,(z - z_0)\right),$$

$$\Rightarrow f(z) = \frac{v/2}{h^2} = \frac{v/2}{\cosh^2\left(\frac{\sqrt{v}}{2}\,(z - z_0)\right)}.$$

Vaihdetaan vielä muuttuja z alkuperäisiin muuttujiin x ja t, niin saadaan ratkaisuksi yhtälölle (1.1)

$$u(x,t) = \frac{v/2}{\cosh^2\left(\frac{\sqrt{v}}{2}\,(x - vt - x_0)\right)}. \tag{3.1}$$

Tarkastellaan ratkaisua (3.1). Huomataan kaavasta ja sen kuvaajasta 3.1, että aallon huipun sijainnin määrittää funktion cosh argumenttina oleva $(x - vt - x_0)$. Tästä nähdään muuttujan v olevan nopeus. Muuttuja v löytyy yhtälöstä myös muista kohdista. Se vaikuttaa niin aallon korkeuteen kuin leveyteenkin.

Teoreettisesti tarkastellen siis yhtälön mallintamalla aallolla on seuraavat ominaisuu-

det: se etenee nopeudella v ja se on sitä korkeampi, mitä nopeampi. Lisäksi se säilyttää muotonsa edetessään.

3.2 Kahden aallon törmäys

Solitoniaalto säilyttää muotonsa edetessään, mutta myös törmätessä toisen aallon kanssa se palauttaa alkuperäisen muotonsa törmäyksen jälkeen. Törmäyksessä on havaittavissa vaihesiirtymä, jossa hitaampi aalto jää jälkeen ja nopeampi hyppää hiukan edelle. Näiden siirtymien summa on nolla niin kahden kuin useampienkin aaltojen törmätessä.

Kahden aallon tilanteelle voidaan löytää analyyttinen ratkaisu Bäcklundin muunnoksella:

$$u = -2\frac{\partial^2}{\partial x^2}\ln F(\eta_1, \eta_2), \tag{3.2}$$

jossa

$$F(\eta_1, \eta_2) = 1 + e^{\eta_1} + e^{\eta_2} + A_{12}e^{\eta_1+\eta_2} \quad \text{ja} \quad \eta_i = \sqrt{v_i}(x - v_i t - x_i^0).$$

Näistä saadaan laskettua funktiolle $u(t, x)$ lauseke

$$u = -2\frac{v_1 e^{\eta_1} + v_2 e^{\eta_2} + A_{12}\left(\sqrt{v_1} + \sqrt{v_2}\right)^2 e^{\eta_1+\eta_2}}{1 + e^{\eta_1} + e^{\eta_2} + A_{12}e^{\eta_1+\eta_2}} + 2\left(\frac{\sqrt{v_1}e^{\eta_1} + \sqrt{v_2}e^{\eta_2} + A_{12}\left(\sqrt{v_1} + \sqrt{v_2}\right)e^{\eta_1+\eta_2}}{1 + e^{\eta_1} + e^{\eta_2} + A_{12}e^{\eta_1+\eta_2}}\right)^2$$

Tähän voidaan vielä sijoittaa A_{12}:n arvo

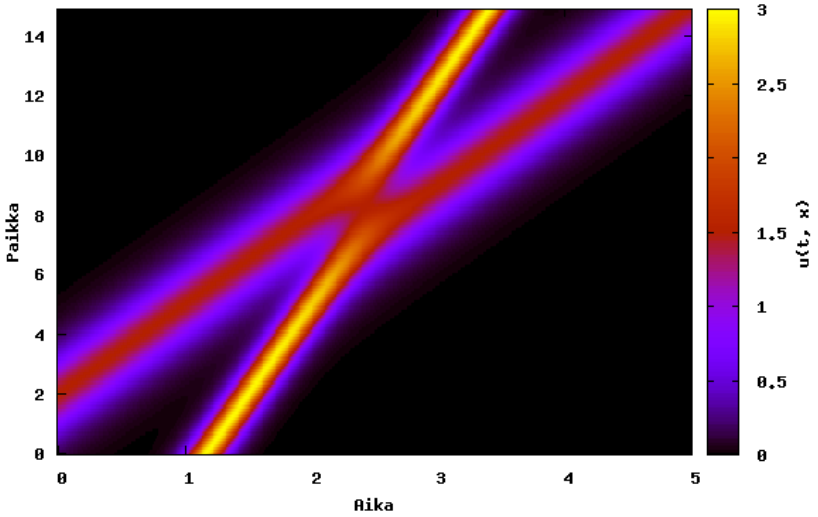
$$A_{12} = \left(\frac{\sqrt{v_1} - \sqrt{v_2}}{\sqrt{v_1} + \sqrt{v_2}}\right)^2,$$

niin saadaan lopulliseksi yhtälöksi

$$u = 2\left(\frac{\sqrt{v_1}e^{\eta_1} + \sqrt{v_2}e^{\eta_2} + \frac{(\sqrt{v_1} - \sqrt{v_2})^2}{\sqrt{v_1} + \sqrt{v_2}}e^{\eta_1+\eta_2}}{1 + e^{\eta_1} + e^{\eta_2} + \left(\frac{\sqrt{v_1} - \sqrt{v_2}}{\sqrt{v_1} + \sqrt{v_2}}\right)^2 e^{\eta_1+\eta_2}}\right)^2 - 2\frac{v_1 e^{\eta_1} + v_2 e^{\eta_2} + \left(\sqrt{v_1} - \sqrt{v_2}\right)^2 e^{\eta_1+\eta_2}}{1 + e^{\eta_1} + e^{\eta_2} + \left(\frac{\sqrt{v_1}-\sqrt{v_2}}{\sqrt{v_1}+\sqrt{v_2}}\right)^2 e^{\eta_1+\eta_2}} \tag{3.3}$$

Lasketaan nyt yhtälöstä (3.3) kahden aallon tilan kehitys. Valittiin ensimmäisen aallon nopeudeksi $v_1 = 3{,}0$ ja aallon huipun alkupaikaksi $x_1^0 = 0{,}0$ sekä toisen aallon nopeudek-

si $v_2 = 6{,}0$ ja aallon huipun alkupaikaksi $x_2^0 = -7{,}0$. Laskettiin yhtälön arvoja muuttujan x arvoilla $[0{,}0; 15{,}0]$ sekä muuttujan t arvoilla $[0{,}0; 5{,}0]$. Tulokset on nähtävissä kuvassa 3.2.



Kuva 3.2. Kahden aallon törmäys analyyttisellä ratkaisulla (3.3) laskettuna.

Kuvassa 3.2 nähdään nopeamman aallon lähestyvän hitaampaa ja törmäävän tämän kanssa aikavälillä $t \in [2{,}0; 3{,}0]$. Törmäyksessä havaitaan tapahtuvan vaihesiirtymä, jonka jälkeen aallot jatkavat alkuperäisillä nopeuksillaan.

3.3 Säilyvät suureet

KdV:lle voidaan laskea säilyviä suureita rekursiokaavalla [5]

$$I_n(t) = \int_{-\infty}^{\infty} g_{2n}(x,t)\mathrm{d}x, \tag{3.4}$$

jossa

$$g_{n+1} = -g_n' + \sum_{k=0}^{n-1} g_k g_{n-k-1}, \quad \text{ja} \quad g_0 = u. \tag{3.5}$$

Lasketaan seuraavaksi g_i, jossa $i \in [1, 2, 3, 4]$:

$$g_1 = -g_0' = -u', \tag{3.6}$$

$$g_2 = -g_1' + \sum_{k=0}^{0} g_k g_{1-k-1} = -g_1' + g_0 g_0$$
$$= u'' + u^2, \tag{3.7}$$

$$g_3 = -g_2' + \sum_{k=0}^{1} g_k g_{2-k-1} = -\frac{\mathrm{d}}{\mathrm{d}x}(u^2 + u'') + (g_0 g_1 + g_1 g_0)$$
$$= -2uu' - u''' + 2u(-u')$$
$$= -4uu' - u''' \tag{3.8}$$

ja

$$g_4 = -g_3' + \sum_{k=0}^{2} g_k g_{3-k-1} = -\frac{\mathrm{d}}{\mathrm{d}x}(-4uu' - u''') + (g_0 g_2 + g_1 g_1 + g_2 g_0)$$
$$= 4uu'' + 4(u')^2 + u^{(4)} + 2(u^3 + uu'') + (u')^2$$
$$= 2u^3 + 5(u')^2 + 6uu'' + u^{(4)}. \tag{3.9}$$

Nyt voidaan laskea säilyvät suureet yhtälöstä (3.4). Huomataan, että funktioiden g_i arvoista parittomat antavat vastaavalle I:lle arvon nolla, joten katsotaan vain indeksiltään i parilliset arvot. Lasketaan kolme ensimmäistä säilyvää suuretta I_0, I_1 ja I_2:

$$I_0(t) = \int_{-\infty}^{\infty} g_0 \mathrm{d}x = \int_{-\infty}^{\infty} u(x,t)\mathrm{d}x. \tag{3.10}$$

Suure I_0 on tulkittavissa systeemin kokonaisliikemääräksi.

$$I_1(t) = \int_{-\infty}^{\infty} g_2 \mathrm{d}x = \int_{-\infty}^{\infty} (u'' + u^2)\mathrm{d}x = \int_{-\infty}^{\infty} u^2 \mathrm{d}x + \int_{-\infty}^{\infty} (u'')\mathrm{d}x$$

Tässä $\int_{-\infty}^{\infty} u'' = 0$, joten

$$I_1(t) = \int_{-\infty}^{\infty} u(x,t)^2 \mathrm{d}x. \tag{3.11}$$

Suureen I_1 voi tulkita systeemin kokonaisenergiaksi.

$$
\begin{aligned}
I_2(t) &= \int_{-\infty}^{\infty} g_4 \mathrm{d}x = \int_{-\infty}^{\infty} (2u^3 + 5(u')^2 + 6uu'' + u^{(4)})\mathrm{d}x \\
&= \int_{-\infty}^{\infty} (2u^3 + 5(u')^2)\mathrm{d}x + \int_{-\infty}^{\infty} u^{(4)}\mathrm{d}x + \int_{-\infty}^{\infty} 6uu''\mathrm{d}x \\
&= \int_{-\infty}^{\infty} (2u^3 + 5(u')^2)\mathrm{d}x + \int_{-\infty}^{\infty} (u''') + 6\int_{-\infty}^{\infty} (uu') - 6\int_{-\infty}^{\infty} (u')^2\mathrm{d}x \\
&= \int_{-\infty}^{\infty} \left[2u^3 - (u')^2\right],
\end{aligned}
$$

eli

$$
I_2(t) = \int_{-\infty}^{\infty} \left[2u(x,t)^3 - (u(x,t)')^2\right] \mathrm{d}x. \tag{3.12}
$$

Koska ohjelmallisesti käsittelemme hilaa emmekä jatkuvia arvoja, täytyy säilyvät suureet vielä ilmaista diskreetissä muodossa. Näin säilyville suureille saadaan esitykset

$$
I_0(t) = \sum_{i=0}^{N} u_i, \tag{3.13}
$$

$$
I_1(t) = \sum_{i=0}^{N} u_i^2, \tag{3.14}
$$

ja (3.15)

$$
I_2(t) = \sum_{i=0}^{N} \left(2u_i^3 - \left(\frac{u_i^t - u_i^{t-1}}{\Delta t}\right)^2\right). \tag{3.16}
$$

Tässä työssä puhuttaessa säilyvistä suureista tarkoitetaan juurikin näitä yhtälöitä (3.13), (3.14) ja (3.16).

4 Paikkadiskretoinnit

Simuloitavaksi valittiin KdV-yhtälö (1.1), koska sille saatiin konstruoitua niin analyyttinen ratkaisu, kuin simulaation toimivuutta mittaamaan käytetyt säilyvät suureet.

Simulaation aluksi lasketaan käytetty alkutila KdV:n analyyttisestä ratkaisusta (3.1). Joillain menetelmillä kokeiltiin myös alkutiloja, joissa oli kaksi solitoniaaltoa. Nämä tilat laskettiin myös samasta ratkaisusta summaamalla kaksi tilaa yhteen. Solitonin luonteen huomioiden kahden aallon tilan laskeminen tällä tavoin on väärin, mutta aaltojen ollessa tarpeeksi kaukana toisistaan on menetelmästä koituva virhe hyvin pieni.

Yhtälöt (3.13), (3.14) ja (3.16) kuvaavat solitoniaallon liikemäärää, energiaa ja kolmatta säilyvää suuretta. Näitä verrataan simulaation alussa laskettuihin alkuarvoihin. Sallittuina virherajoina pidetään 20 prosentin muutosta alkuperäisestä. Tämän ylittämisen ohjelma tulkitsee virheeksi ja aloittaa simulaation uudestaan hieman pienemmällä aika-askeleella.

Seuraavissa simulaatioissa käytetään arvoja (ellei toisin sanota):

`l` = 15.0
`h` = 0.1
`v_1` = 5.0
`t_max` = 10.0
`m` = 100

Hilapisteita tulee siis olemaan yhteensä 150 ja mallinnettavana yksi solitoniaalto nopeudeltaan $5{,}0$. Simulaation tulkitaan toimivan, jos säilyvät suureet pysyvät annetuissa rajoissa yli $10{,}0$ aikayksikköä. Jos suureet eivät pysy rajoissaan, pienennetään aika-askelta neljänneksellä edellisestä arvosta ja yritetään uudelleen.

4.1 Suoraviivainen epäsymmetrinen diskretointi

Sovelletaan kappaleessa 2.1 mainittua diskretointia KdV-yhtälöön (1.1). KdV:ssä on mukana funktion $u(x,t)$ ensimmäinen ja kolmas derivaatta x:n suhteen, eli u_x ja u_{xxx}. Nyt siis

$$u_x = \frac{u(x+h) - u(x)}{h} \tag{4.1}$$

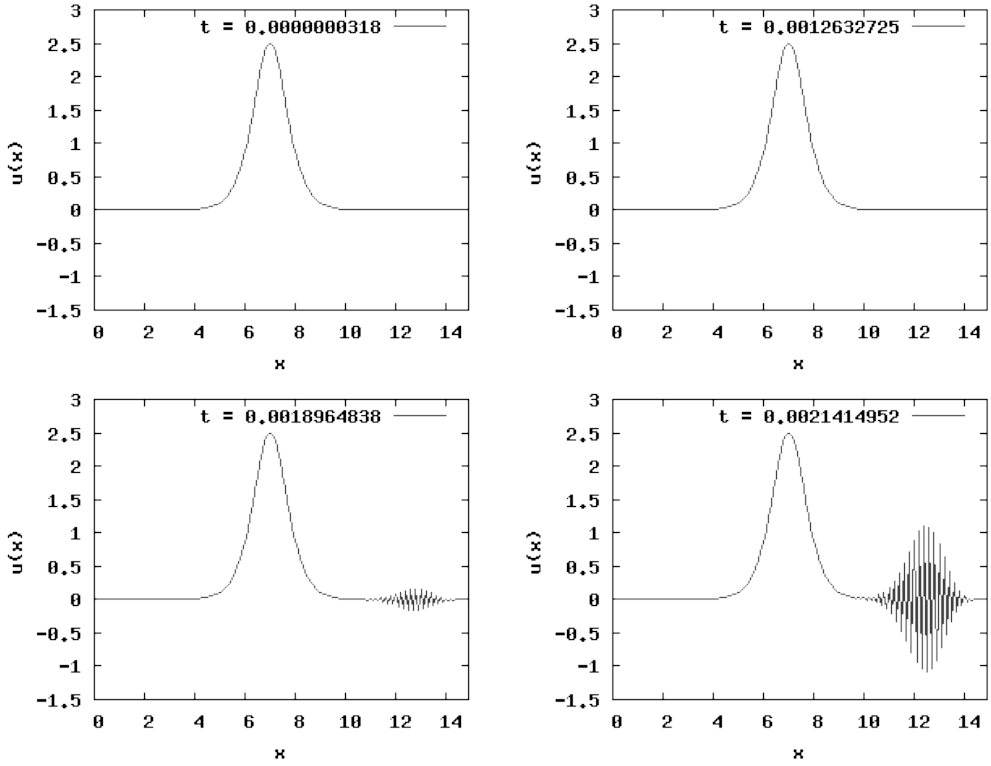
ja samoin

$$u_{xxx} = \frac{u_{xx}(x+h) - u_{xx}(x)}{h} \quad \text{ja} \quad u_{xx} = \frac{u_x(x+h) - u_x(x)}{h}.$$

Sijoitetaan u_x u_{xx}:n yhtälöön:

$$u_{xx} = \frac{u_x(x+h) - u_x(x)}{h} = \frac{1}{h}\left(\frac{u(x+h+h) - u(x+h)}{h} - \frac{u(x+h) - u(x)}{h}\right).$$

Otetaan käyttöön lyhennysmerkintä $u(x+ah) = u_{n+a}$. Huomioitavaa on, että merkintänä



Kuva 4.1. Epäsymmetrisen diskretoinnin ratkaisun mallintamista Runge-Kuttalla käyttäen aika-askelta $\Delta t = 3{,}1819 \cdot 10^{-8}$. Kuvassa nähdään tilan neljässä eri ajankohdassa.

u:n alaindeksinä n tarkoittaa hilapistettä ja x osittaisderivaattaa, kun taas u:n argumenttina oleva x viittaa paikkakoordinaattiin.

$$u_{xx} = \frac{1}{h^2}\left(u_{n+2} - 2u_{n+1} + u_n\right)$$

Sijoitetaan nyt tämä u_{xxx}:n yhtälöön.

$$\begin{aligned} u_{xxx} &= \frac{u_{xx}(x+h) - u_{xx}(x)}{h} \\ &= \frac{1}{h^2}\left(\frac{u(x+3h) - 2u(x+2h) + u(x+h)}{h} - \frac{u(x+2h) - 2u(x+h) + u(x)}{h}\right) \\ &= \frac{1}{h^3}\left(u_{n+3} - 3u_{n+2} + 3u_{n+1} - u_n\right), \end{aligned} \tag{4.2}$$

joten yhtälöstä (1.1) voidaan diskretoida kaava

$$u_t = -\frac{u_{n+3} - 3u_{n+2} + 3u_{n+1} - u_n}{h^3} - 6u_n\frac{u_{n+1} - u_n}{h}. \tag{4.3}$$

Yhtälön vasemman puolen mallintamiseen käytetään Eulerin menetelmää (2.5) tai

neljännen kertaluvun Runge-Kuttaa (2.14). Valitaan menetelmissä tarvittavan aika-askeleen alkuarvoksi $\Delta t_0 = 0{,}01$.

Epäsymmetrisellä diskretoinnilla ei saatu simulaatiota toimimaan ollenkaan. Aika-askelia kokeiltiin niin Eulerin menetelmällä kuin Runge-Kuttallakin arvoon $\Delta t = 1{,}0^{-10}$ asti ja simulaatio pysyi toiminnassa vain aikaan $t = 0{,}00214$ saakka. Käyttäen hilavakiota $h = 0{,}05$ päästiin ajanhetkeen $t = 0{,}000290$ ja hilavakiolla $h = 0{,}01$ ajanhetkeen $t = 0{,}00000242$. Parannusta hilapisteiden tiheyttä kasvattamalla ei siis saatu.

Kuvan 4.1 alaoikealla nähdään solitonin tila ajanhetkellä $t = 0{,}002141$. Tämän oli viimeinen iteraatio, millä säilyvät suureet olivat vielä sallituissa vaihtelurajoissaan. Huomataan, että alkuperäisen aallon rinnalle on syntynyt sekä ylös että alaspäin ulottuva solitoniaallon muotoinen muodostelma. Saatu lopputila oli hyvin samanlainen kaikilla kokeilluilla aika-askelilla väliltä $\Delta t \in [1 \cdot 10^{-10}, 2 \cdot 10^{-6}]$. Tätä suuremmilla säilyvien suureiden testaustiheys oli liian suuri suhteessa aikaan, jonka simulaatio pysyi toiminnassa. Pienempiä aika-askelia kuin $1 \cdot 10^{-10}$ ei kokeiltu, sillä niiden käyttö ei olisi enää kovin käytännöllistä.

4.2 Suoraviivainen symmetrinen diskretointi

Yritettiin parantaa tarkkuutta approksimoimalla derivaattaa epäsymmetrisen menetelmän sijaan symmetrisellä.

Approksimoidaan derivaattaa paikassa x yhtälöllä

$$u_x = \frac{u_{n+1} - u_{n-1}}{2h}. \tag{4.4}$$

Tätä jatkamalla saadaan tilan kolmanneksi derivaataksi

$$u_{xxx} = \frac{u_{n+3} - 3u_{n+1} + 3u_{n-1} - u_{n-3}}{8h^3}. \tag{4.5}$$

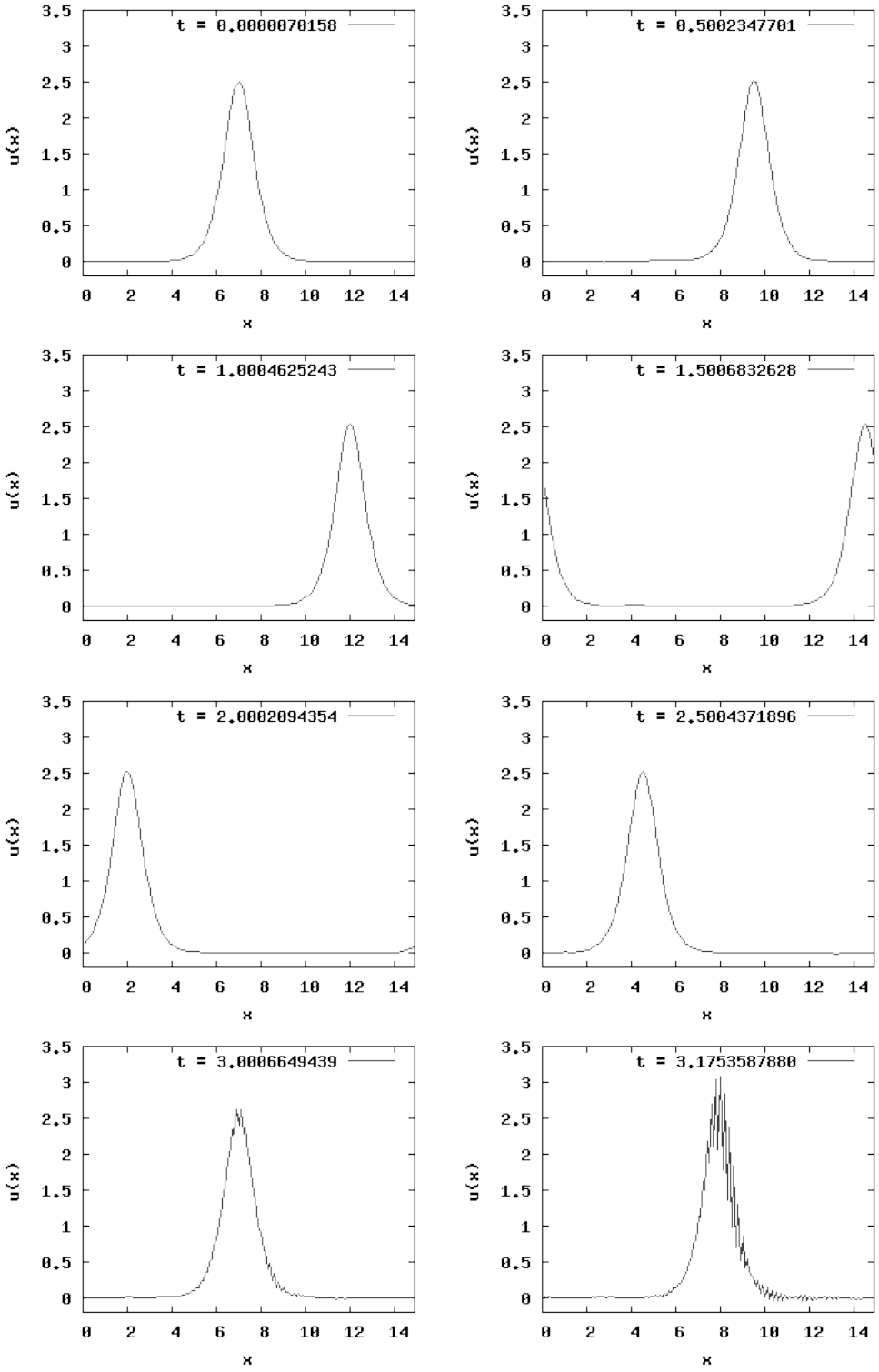
Näistä saadaan koko mallinnettavaksi yhtälöksi

$$u_t = -\frac{u_{n+3} - 3u_{n+1} + 3u_{n-1} - u_{n-3}}{8h^3} - 6u_n \frac{u_{n+1} - u_{n-1}}{2h}. \tag{4.6}$$

Ajettiin simulaatiot kuten edellä. Aika-askeleen alkuarvona käytettiin $\Delta t_0 = 0{,}01$.

Aika-askelella $\Delta t = 9{,}354 \cdot 10^{-6}$ ja sitä pienemmillä simulaatio toimi ajankohtaan $t = 3{,}17$ saakka. Tämä ei parantunut pienemmilläkään aika-askelilla kertaluokkaan 10^{-10} asti. Runge-Kuttalla päästiin ajanhetkeen $t = 3{,}17$ asti jo aika-askelella $\Delta t = 0{,}0028$, mutta sen pidemmälle simulaatio ei edennyt millään kokeilluista arvoista.

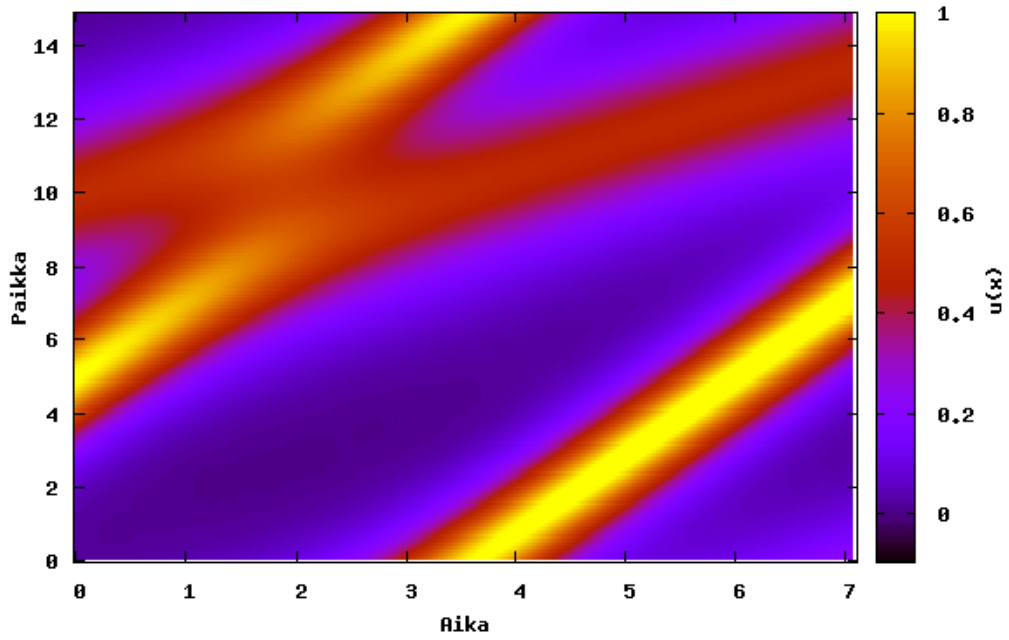
Kuvassa 4.2 huomataan kuinka Eulerin menetelmää käyttäessä aalto alkaa ajanhetkien $t = 2{,}5$ ja $t = 3{,}0$ välillä hajota. Tarkasteltaessa lähemmin näyttää joka toinen hilapiste kuuluvan alkuperäistä aaltoa korkeampaan ja joka toinen alkuperäistä matalampaan



Kuva 4.2. Yhtälön (4.6) mallintamista Eulerin menetelmällä. Seitsemässä ensimmäisessä kuvassa nähdään tilan aikakehitystä puolen aikayksikön välein. Kahdeksas kuva on ajanhetkeltä $t = 3.17$, jonka jälkeen säilyvät suureet eivät enää pysyneet toleranssirajoissaan.

aaltoon. Samanlaista hajoamista havaittiin epäsymmetrisen diskretoinnin aallon viereen syntyneissä uusissa aalloissa.

Hilavakiota pienentämällä arvoon $h = 0{,}05$ päästiin aika-askelta $\Delta t = 0{,}0002839$ käyttämällä ajanhetkeen $t = 3{,}45$ asti. Verrattuna arvoilla $h = 0{,}1$ ja $\Delta t = 0{,}0028$ suoritettuun mallinnukseen suoritusaika oli 20-kertainen ja simulaatio toimi vain 8% pidemmälle. Etua tästä ei siis oikeastaan ollut. Kokeiltiin vielä hilavakiota $h = 0{,}01$ ja päästiin aika-askelella $\Delta t = 0{,}0000028$ ajanhetkeen $t = 3{,}724$. Jälleen siis vähän pidemmälle, mutta suoritusaika alkuarvoihin $h = 0{,}1$ ja $\Delta t = 0{,}0028$ verrattuna on 10000-kertainen. Hyötyä tästä ei siis ole.



Kuva 4.3. Kahden solitoniaallon törmäys mallinnettaessa Runge-Kuttalla yhtälöllä (4.6). Aaltojen nopeudet ovat $v_1 = 2{,}0$ sekä $v_2 = 1{,}0$ ja hilavakio $h = 0{,}1$. Simulaatio hajosi ajankohdan $t = 7{,}06$ jälkeen.

Kokeiltiin myös simulaatiota kahden aallon alkutilasta hilavakiolla $h = 0{,}1$. Nopeuksilla $v_1 = 15{,}0$ ja $v_2 = 5{,}0$ etenevät aallot ehtisivät vielä törmäämään $3{,}17$ aikayksikössä, minkä simulaatio toimi näillä alkuarvoilla. Kahdella aallolla ei kuitenkaan päästy näin pitkälle, vaan tila alkoi hajota jo paljon aikaisemmin. Kokeiltiin vielä kahta aaltoa pienemmillä nopeuksilla $v_1 = 2{,}0$ ja $v_2 = 1{,}0$. Nyt päästiin aikaan $t = 7{,}06$ asti ja aallot ehtivät törmätä. Törmäys nähdään kuvassa 4.3. Aallot etenivät vielä jonkin matkaa tämän jälkeen, mutta ajanhetkellä $t = 7{,}06$ suureet karkaavat rajoistaan ja simulaatio päätetään.

4.3 Keskiarvon käyttäminen symmetrisessä diskretoinnissa

Otetaan yhtälö (4.6) ja korvataan sen epälineaarisessa osassa oleva u_n keskiarvolla edellisestä ja seuraavasta hilapisteestä $u_{ka,1} = \frac{1}{2}(u_{n-1} + u_{n+1})$. Yhtälö saadaan muotoon

$$u_t = -\frac{u_{n+3} - 3u_{n+1} + 3u_{n-1} - u_{n-3}}{8h^3} - 3 \cdot (u_{n-1} + u_{n+1}) \frac{u_{n+1} - u_{n-1}}{2h}. \tag{4.7}$$

Kokeillaan yhtälöä vielä kahdella muulla tavalla otetuilla keskiarvoilla. Kolmella hilapisteellä laskettuna saadaan $u_{ka,2} = \frac{1}{3}(u_{n-1} + u_n + u_{n+1})$ ja viidellä pisteellä laskettuna $u_{ka,3} = \frac{1}{5}(u_{n-2} + u_{n-1} + u_n + u_{n+1} + u_{n+2})$. Laskentatehokkuuden takia kokeillaan myös viiden pisteen keskiarvoa valitsemalla pisteet samoiksi, mitkä joka tapauksessa tarvitsee laskea. Tästä saadaan siis $u_{ka,4} = \frac{1}{5}(u_{n-3} + u_{n-1} + u_n + u_{n+1} + u_{n+3})$. Yhtälöt näille ovat

$$u_t = -\frac{u_{n+3} - 3u_{n+1} + 3u_{n-1} - u_{n-3}}{8h^3} - (u_{n-1} + u_n + u_{n+1}) \frac{u_{n+1} - u_{n-1}}{h}, \tag{4.8}$$

$$u_t = -\frac{u_{n+3} - 3u_{n+1} + 3u_{n-1} - u_{n-3}}{8h^3} - 3 \cdot (u_{n-2} + u_{n-1} + u_n + u_{n+1} + u_{n+2}) \frac{u_{n+1} - u_{n-1}}{5h}, \tag{4.9}$$

$$u_t = -\frac{u_{n+3} - 3u_{n+1} + 3u_{n-1} - u_{n-3}}{8h^3} - 3 \cdot (u_{n-3} + u_{n-1} + u_n + u_{n+1} + u_{n+3}) \frac{u_{n+1} - u_{n-1}}{5h}. \tag{4.10}$$

Simulaatioissa havaittiin yhtälöiden (4.7), (4.8), (4.9) ja (4.10) käyttäytyvän hyvin samankaltaisesti. Eulerin menetelmällä saatiin $1{,}5$ aikayksikköä kestävä simulaatio kaikilla yhtälöillä aika-askelella $\Delta t \approx 0{,}000024$. Pientä vaihtelua oli havaittavissa, mutta ei mitään merkittävää. Aina pienennettäessä aika-askelta $\frac{3}{4}$-osan verran piteni simulaatio $\frac{4}{3}$-osalla. Yli 10 aikayksikön kesto saataisiin täten aika-askelella $\Delta t = 0{,}0000032$.

Pienentämällä aika-askelta epäonnistuneen mallinnuksen jälkeen normaalin verran, eli kolmeen neljäsosaan, ei eri yhtälöiden välille saatu näkyviin eroja. Neljännen kertaluvun Runge-Kuttaa käyttäen saatiin kaikilla keskiarvoyhtälöillä suurimmaksi toimivan aika-askeleen arvoksi sama $\Delta t = 0{,}00277$. Jotta yhtälöiden välille saatiin jotain eroa, kokeiltiin aika-askelta pienentää vähemmän kerrallaan. Ensin sitä pienennettiin $0{,}99$ kertaiseksi, mutta vasta pienentämällä $0{,}9999$ edellisestä saatiin näkyviä eroja. Aika-askeleella $\Delta t = 0{,}0028331$ päästiin kaikilla näillä yhtälöillä yli 10 aikayksikön, mutta parhaiten näistä toimi (4.8). Tuloksia taulukossa 4.3

Kokeiltaessa yhtälöllä (4.7) aika-askeleen arvoja isommalla välimatkalla $(\Delta t)_i = 0{,}75 \cdot (\Delta t)_{i-1}$, eli pienennettäessä aika-askelta neljänneksellä epäonnistumisen jälkeen, saatiin arvolla $\Delta t = 0{,}002373$ toimiva simulaatio. Tätä edellisellä aika-askelella simulaatio toimi ajanhetkeen $t = 3{,}16$ saakka, mutta askelella $\Delta t = 0{,}002373$ päästiin jo yli ajanhetken $t = 300{,}0$. Tähän simulaation pyörittäminen lopetettiin. Havaitaan, että saman kertaluokan aika-askelella, jolla yhtälön (4.6) mallinnus hajosi ajanhetkellä $t = 3{,}17$, päästiin nyt niin pitkälle, kuin vain annettiin simulaation edetä. Tämän kappaleen yhtälöistä (4.7) on

Taulukko 4.1. Mallinnuksia keskiarvoyhtälöille. Tavallisella aika-askeleen pienentämisellä ei yhtälöiden välille saatu eroja, joten muutettiin ohjelmaa pienentämään askelta $0{,}9999$:n edellisestä. Näin saatiin pienet erot yhtälöiden toiminnan välille. Erot eivät kuitenkaan ole merkittäviä.

Δt	Yhtälö (4.7)	Yhtälö (4.8)	Yhtälö (4.9)	Yhtälö (4.10)
0,002836	5,07644	5,1048	4,540436	4,228476
0,0028357164	5,3906968764	5,4190540404	4,8519107604	4,4832676284
0,00283543282836	5,73040974612	5,7587640744	5,22003183701	4,79471691276
0,00283514928508	6,18346059075	6,24016357645	6,18346059075	5,13445535527
0,00283486577015	6,77532919066	6,86037516376	6,71863187525	5,52798825179
0,00283458228357	7,42944016524	7,57116927942	7,34440269673	5,98380320062
0,00283429882534	8,22230089232	8,30732985708	8,08058595105	6,52172159711
0,00283401539546	8,95548864966	9,09718941943	8,87046818779	7,14171879656
0,00283373199392	10,0002402065	10,0002402065	9,92089571072	7,93444958298
0,00283344862072	> 10	> 10	10,0020736311	8,92819660389
0,00283316527586	> 10	> 10	> 10	10,0010734238

laskentatehokkuudeltaan paras, sillä siinä tehdään vähiten laskutoimituksia iteraatiota kohden.

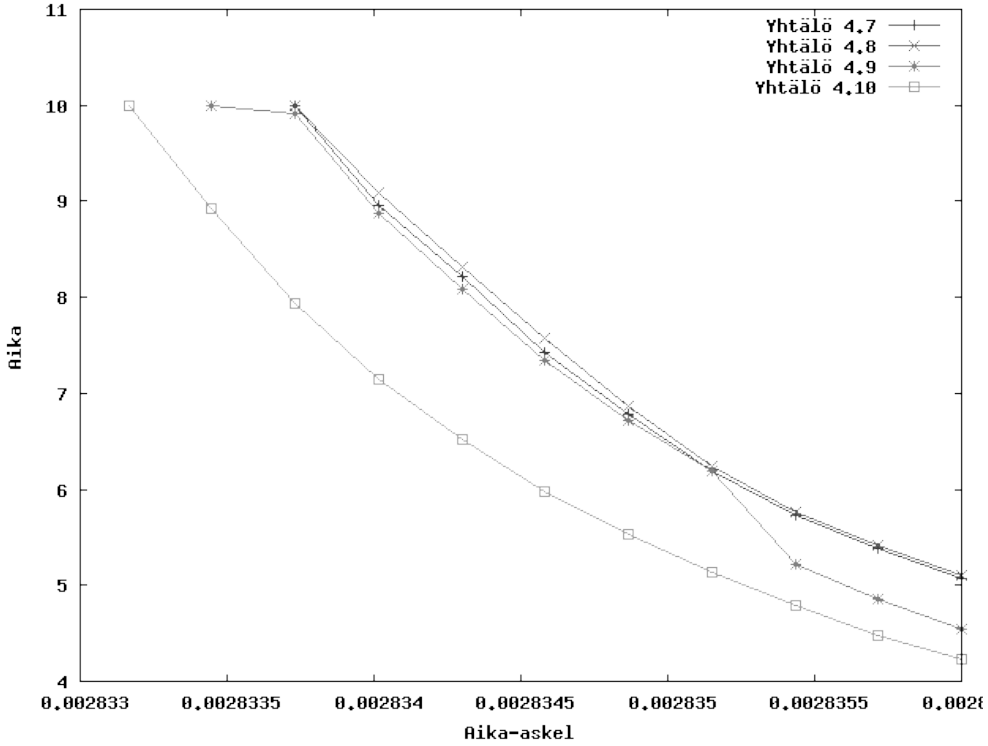
Ajettiin kullakin yhtälöllä simulaatio ja tarkasteltiin niiden tiloja hetkellä $t = 9{,}492$. Lasketaan tälle ajanhetkelle teoreettinen solitoniaallon huipun sijainti, kun nopeus $v_1 = 5{,}0$. Lähtien kohdasta $x_0 = 7{,}5$ ja edeten ajan t saadaan huipun sijainniksi $x_{teor} = 7{,}5 + 9{,}492 \cdot 5{,}0 = 54{,}960$. Yhtälön (4.7) simulaatiossa saatiin huipun sijainniksi $x_{ka1} = 10{,}111$, joka syklisyys huomioiden on $x_{ka1} = 55{,}111$. Verrataan tätä teoreettiseen ja saadaan erotukseksi $0{,}151$. Tämä on $0{,}27\%$ koko matkasta. Samat tulokset muille yhtälöille nähdään taulukossa 4.3.

Taulukko 4.2. Keskiarvoyhtälöiden lopputilojen vertailua. Verrataan huipun sijaintia sen teoreettiseen sijaintiin ja lasketaan paljonko mallinnetun aallon nopeus poikkeaa teoreettisesta.

	$x_{huippu}(9{,}492)$	$x_{simu} - x_{teor}$	$\frac{x_{simu} - x_{teor}}{x_{teor}}$
Yhtälö (4.7)	$55{,}111$	$0{,}151$	$0{,}275\%$
Yhtälö (4.8)	$55{,}061$	$0{,}101$	$0{,}184\%$
Yhtälö (4.9)	$55{,}244$	$0{,}284$	$0{,}517\%$
Yhtälö (4.10)	$55{,}464$	$0{,}504$	$0{,}917\%$

Huomataan, että yhtälön (4.8) simuloinnit sekä toimivat suurimmalla aika-askelella että ovat tuloksiltaan lähimpänä teoreettista arvoa.

Ajettiin vielä yhtälöllä (4.8) kahden aallon simulaatio. Aaltojen nopeudet ovat $v_1 = 5{,}0$ ja $v_2 = 10{,}0$, hilavakio $h = 0{,}1$ ja aika-askel $\Delta t = 0{,}002373$. Tulos törmäyksistä nähdään kuvassa 4.3. Simuloidulla ajanjaksolla ehtivät aallot törmätä neljä kertaa. Kuvassa näkyy törmäyksissä tapahtuva vaihesiirtymä selkeästi.



Kuva 4.4. Kuvaaja taulukon 4.3 arvoista. Kuvaajasta nähdään simulaatioajan pidentyminen aika-askelta lyhennettäessä.

4.4 Zabuskyn ja Kruskalin diskretointi

Artikkelissaan [3] Zabusky ja Kruskal käyttivät KdV:lle seuraavanlaista diskretointia:

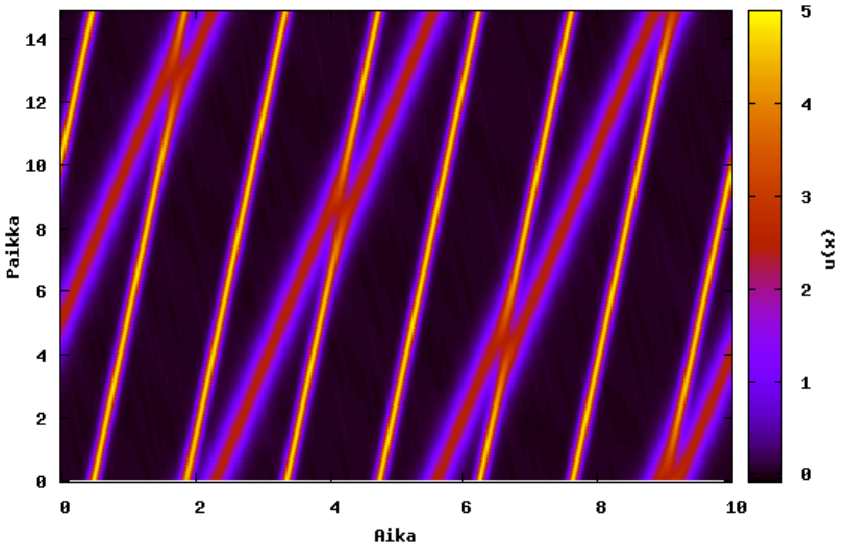
$$u_n^{t+1} = u_n^{t-1} - 2\Delta t \left(\frac{(u_{n+1}^t + u_n^t + u_{n-1}^t)(u_{n+1}^t - u_{n-1}^t)}{h} + \frac{u_{n+2}^t - 2u_{n+1}^t + 2u_{n-1}^t - u_{n-2}^t}{2h^3} \right). \tag{4.11}$$

Tässä ensimmäinen osa vastaa KdV:n (1.1) epälineaarista osaa $6uu_x$ ja jälkimmäinen lineaarista osaa u_{xxx}.

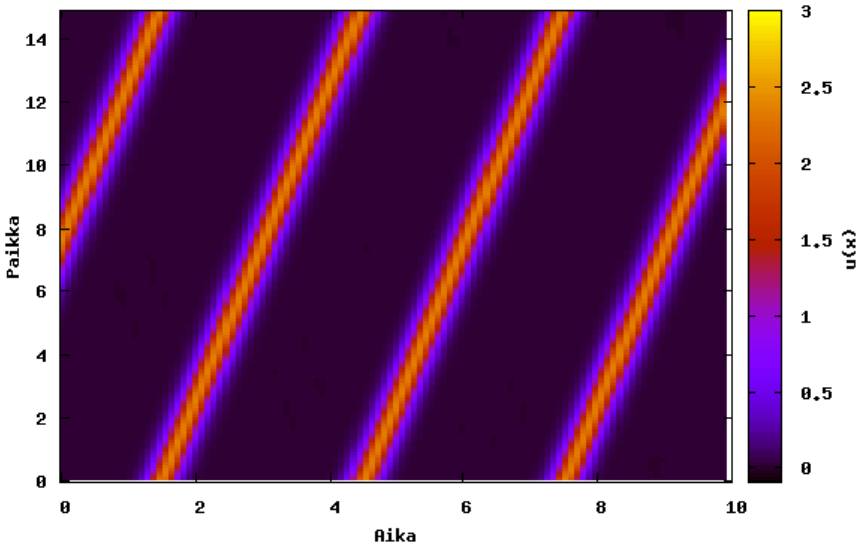
Epälineaarisessa osassa u on korvattu keskiarvolla hilapisteistä $u(x-1), u(x)$ ja $u(x+1)$. Kappaleessa 4.3 huomattiinkin tämän olevan kokeilluista keskiarvon laskutavoista paras vaihtoehto. Derivaatta u_x on otettu kuten yhtälössä (2.2):

$$\begin{aligned} 6uu_x &= 6 \cdot \frac{u(x-1) + u(x) + u(x+1)}{3} \cdot \frac{u(x+1) - u(x-1)}{2h} \\ &= \frac{(u(x-1) + u(x) + u(x+1))(u(x+1) - u(x-1))}{h} \\ &= \frac{(u_{x+1} + u_x + u_{x-1})(u_{x+1} - u_{x-1})}{h}. \end{aligned}$$

Lineaarinen osa u_{xxx} on ensin approksimoitu kaksi kertaa yhtälön (2.3) mukaan ja kerran



Kuva 4.5. Kuva törmäyksistä mallinnettuna Runge-Kuttalla ja yhtälöllä (4.8). Aaltojen nopeudet ovat $v_1 = 5{,}0$ ja $v_2 = 10{,}0$, hilavakio $h = 0{,}1$ ja aika-askel $\Delta t = 0{,}002373$.



Kuva 4.6. Solitoniaallon tilan edistys ajan funktiona mallinnettaessa yhtälöä (4.11) Runge-Kuttalla

yhtälön (2.1) mukaan

$$u_x = \frac{u(x+\frac{h}{2}) - u(x-\frac{h}{2})}{h}$$
$$u_{xx} = \frac{1}{h}\left(\frac{u(x+h)-u(x)}{h} - \frac{u(x)-u(x-h)}{h}\right) = \frac{u(x+h)-2u(x)+u(x-h)}{h^2}$$

$$\begin{aligned} u_{xxx} &= \frac{1}{h^2}\left(\frac{u(x+2h)-u(x)}{2h} - 2\frac{u(x+h)-u(x-h)}{2h} + \frac{u(x)-u(x-2h)}{2h}\right) \\ &= \frac{u(x+2h)-2u(x+h)+2u(x-h)-u(x-2h)}{2h^3} \\ &= \frac{u_{x+2h}-2u_{x+h}+2u_{x-h}-u_{x-2h}}{2h^3}. \end{aligned}$$

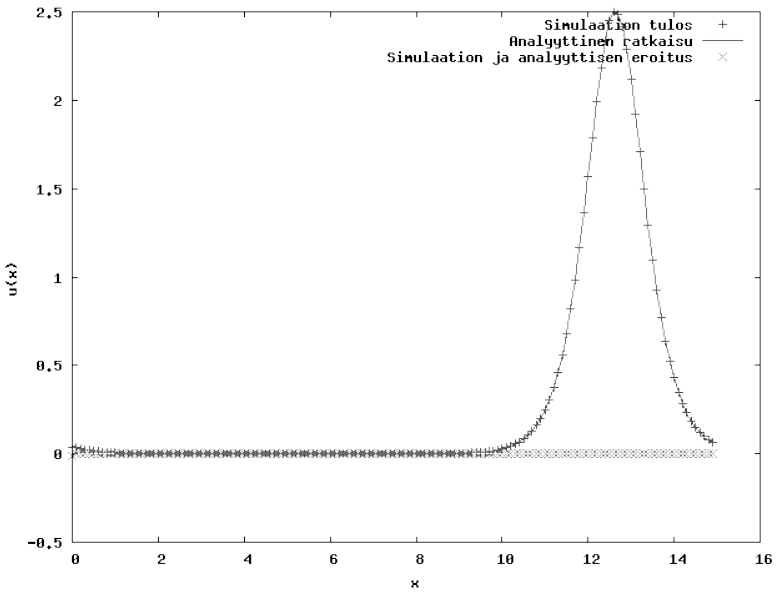
Mallinnetaan yhtälöä (4.11). Käytetään samaa alkuasetelmaa kuin edellisessäkin kohdassa. Hilapisteiden välinen etäisyys on siis $h = 0{,}1$, hilan pituus $l = 15$ ja hilapisteiden määrä täten $N = 150$. Mallinnetaan yhtä aaltoa, joka etenee nopeudella $v_1 = 5{,}0$. Kokeillaan jälleen samoja aikadiskretointeja kuin edellä.

Yhtälössä (4.11) on käytetty Eulerin menetelmää kahden aika-askeleen hypyllä. Vielä ei käytetä tätä vaan kokeillaan ensin tavallista Eulerin menetelmää. Aika-askelella $\Delta t = 0{,}000016$ simulaatio toimi $0{,}313$ aikayksikköä ja tästä pienentämällä aika-askelta $\frac{3}{4}$:n verran piteni simulaation toiminta-aika $\frac{4}{3}$:lla. Askelella $\Delta t = 5{,}34 \cdot 10^{-7}$ saatiin simulaation pituudeksi $9{,}27$ aikayksikköä. Tässä vaiheessa alkoi simulaation kesto olla vuorokauden luokkaa.

Käytettäessä Runge-Kuttaa aikaiterointina alettiin jo saada hyviä tuloksia järkevän kokoisella aika-askelella. Aika-askelella $\Delta t = 0{,}00108$ simulaatio kesti yli 300 aikayksikköä, kun sitä edellisellä ($\Delta t = 0{,}001092$) vain $1{,}3$. Solitonin huippu on ajanhetkellä $t = 10{,}05285$ pisteessä $x_{sim} = 12{,}6325$. Se on tässä vaiheessa kiertänyt hilan kolme kertaa, joten poistettaessa hilan syklisyys sijaitsee huippu paikassa $x_{sim} = 57{,}6325$. Laskettaessa nopeudella $v = 5{,}0$ etenevän solitonin huipun etenemistä saadaan sen teoreettiseksi sijainniksi $x_{teor} = 57{,}76422$. Nyt saadaan simulaation ja teoreettisen arvon erotukseksi ja suhteelliseksi eroksi

$$\begin{aligned} x_{simu} - x_{teor} &= 57{,}6325 - 57{,}764225 = -0{,}131725, \\ \frac{x_{simu} - x_{teor}}{x_{teor}} &= \frac{57{,}6325 - 57{,}764225}{57{,}764225} = -0{,}228\%. \end{aligned}$$

Lasketaan yhtälöllä (3.1) aalto, jonka huippu on pisteessä x_{sim} hetkellä $t = 0$ ja nopeus $v = 5{,}0$. Verrataan tätä simulaation tuloksena saatuun tilaan. Kuvaajassa 4.7 nähdään tämä sekä simulaation tuloksena saatu aalto ajanhetkellä $t = 10{,}00085$. Kuvaajaan on laskettu myös näiden aaltojen erotus. Huomataan, että vaikka simuloitu aalto ei ole edennyt



Kuva 4.7. Vertailu simulaation tuloksen ja analyyttisestä ratkaisusta lasketun aallon välillä. Aika-askeleena käytettiin $\Delta t = 0{,}00108$, hilavakiona $h = 0{,}1$ ja aallon nopeutena $v_1 = 5{,}0$. Kuvassa simulaation tila hetkeltä $t = 10{,}052845$

aivan oikealla nopeudella, on sen muoto kuitenkin pysynyt erittäin hyvin oikeana.

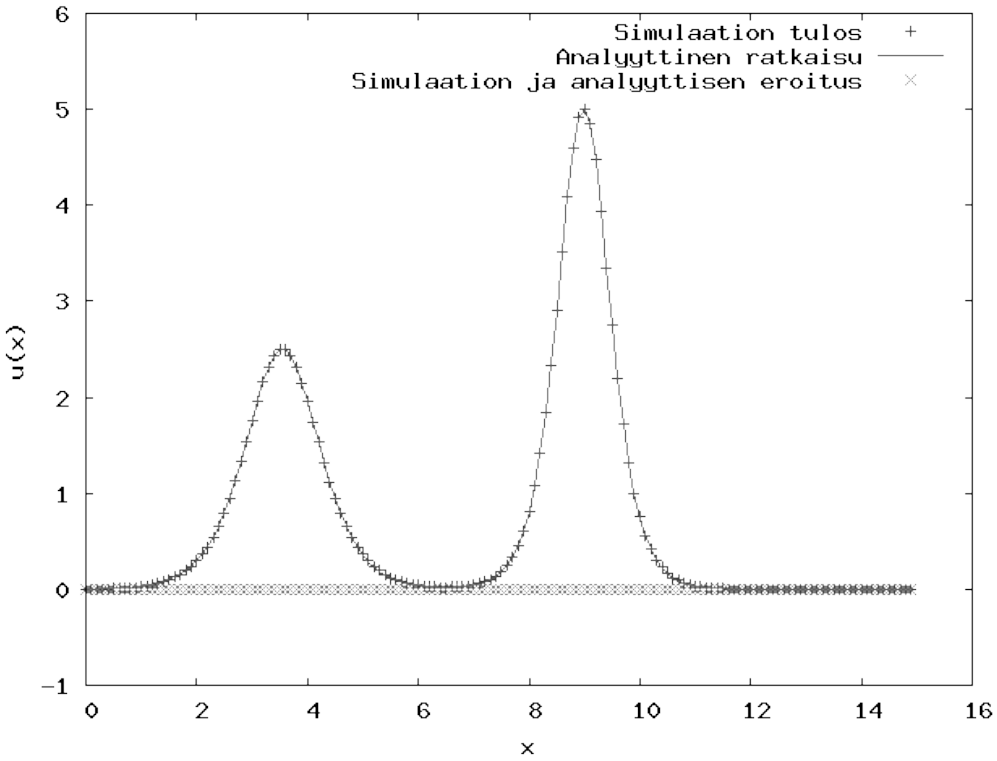
Ajetaan sama simulaatio käyttäen hilavakioita $h = 0{,}05$ ja $h = 0{,}01$. Näistä ensimmäisellä saatiin simulaatio toimimaan aika-askelella $\Delta t = 0{,}0001125$ ja jälkimmäisellä aika-askelella $\Delta t = 8{,}457 \cdot 10^{-7}$. Hilavakiolla $h = 0{,}05$ käyttäen aika-askelta $\Delta t = 0{,}0001125$ aalto eteni $49{,}9674$ pituusyksikköä ajassa $t = 10{,}0$. Teoreettisesti sen olisi pitänyt edetä $50{,}0$. Näiden erotukseksi saadaan $0{,}0327$, joka on $0{,}0568\%$ koko edetystä matkasta. Hilassa huipun teoreettisen sijainnin ja simulaation tuloksen ero on $0{,}261\%$ eli alle kolmasosa aika-askelella $h = 0{,}1$ saadusta virheestä.

Mallinnetaan seuraavaksi yhden aallon sijaan kahta käyttäen samoja hilan ja aika-askeleen arvoja kuin edellä. Laitetaan ne etenemään samaan suuntaan, mutta eri nopeuksilla. Nopeudet ovat $v_1 = 5{,}0$ ja $v_2 = 10{,}0$ sekä aaltojen sijainnit alkutilassa $x_1 = 5{,}0$ sekä $x_2 = 10{,}0$. Alkutilat on laskettu summaamalla kaksi yhden aallon ratkaisua (3.1).Tämä ei kuitenkaan haittaa, sillä aaltojen etäisyys toisistaan alkutilassa on tarpeeksi suuri.

Mallinnetaan tätä kahden aika-askeleen Runge-Kuttalla käyttäen aika-askelta $\Delta t = 0{,}000902$. Aallot kiertävät hilaa syklisesti, joten ne törmäävät useita kertoja pitkän simulaation aikana. Ajettiin simulaatiota ajankohtaan $t = 10{,}0$ asti ja todetaan, että aallot ovat useista törmäyksistä huolimatta säilyttäneet muotonsa hyvin.

Etsittiin huippujen sijainnit sovittamalla hilapisteet sopivaan funktioon. Huiput aalloille löytyvät pisteistä $x_1 = 3{,}54797$ ja $x_2 = 8{,}98662$. Lasketaan analyyttisestä ratkaisusta kahden solitonin tila kuten simulaation alussakin, nyt kuitenkin käyttämällä huippujen sijainteina pisteitä x_1 ja x_2. Lasketaan vielä simulaation lopputilan ja samoihin paikkoihin

sijoitettujen aaltojen erotus. Nähdään kuvaajassa 4.8, että aallot ovat säilyttäneet muotonsa erittäin hyvin.



Kuva 4.8. Kahden aallon mallinnuksen tila ajanhetkeltä $t = 10{,}00085$.

Tarkastellaan vielä solitonien törmäystä. Etsittiin datatiedostosta kohta, jossa näkyisi selkeä törmäys mahdollisimman keskellä hilaa. Tällainen löytyi muun muassa aikaväliltä $t \in [5{,}7;\ 7{,}8]$. Kuvassa 4.9 voidaan nähdä tämä. Nähdään nopeamman aallon lähestyvän hitaampaa ja ajanhetken $t = 6{,}0$ jälkeen alkavat aallot sulautua yhteen. Kuvassa nähdään aikavälillä $t \in [6{,}5;\ 6{,}8]$, kuinka nopeamman aallon huippu pienenee vähitellen hitaamman aallon huipun korkeuksiseksi ja samalla hitaamman aallon huippu kasvaa nopeamman aallon huipun korkeuksiseksi.

Kun tarkastellaan tilaa kahtena eri nopeuksisena aaltona ennen ja jälkeen törmäyksen, huomataan kummallakin aallolla selkeä vaihesiirtymä. Nopeampi aalto on ohittanut hitaamman ja kulkee noin leveytensä verran edempänä, kuin ilman törmäystä olisi kulkenut. Samoin hitaampi aalto on törmäyksen jälkeen jäänyt jälkeen noin leveytensä verran.

Tuloksia

Ehkä hieman ennakko-oletusten vastaisesti parhaat tulokset saatiin käyttämällä yhtälöä (4.8). Sen lisäksi, että sillä pystyi käyttämään suurinta aika-askelta, olivat sen tulokset

Taulukko 4.3. Tuloksia menetelmistä, joilla ei saatu simulaatiota toimimaan. Simulaatiot tehty yhdellä aallolla nopeudeltaan $v = 5{,}0$ ja hilalla, kooltaan $l = 15{,}0$.

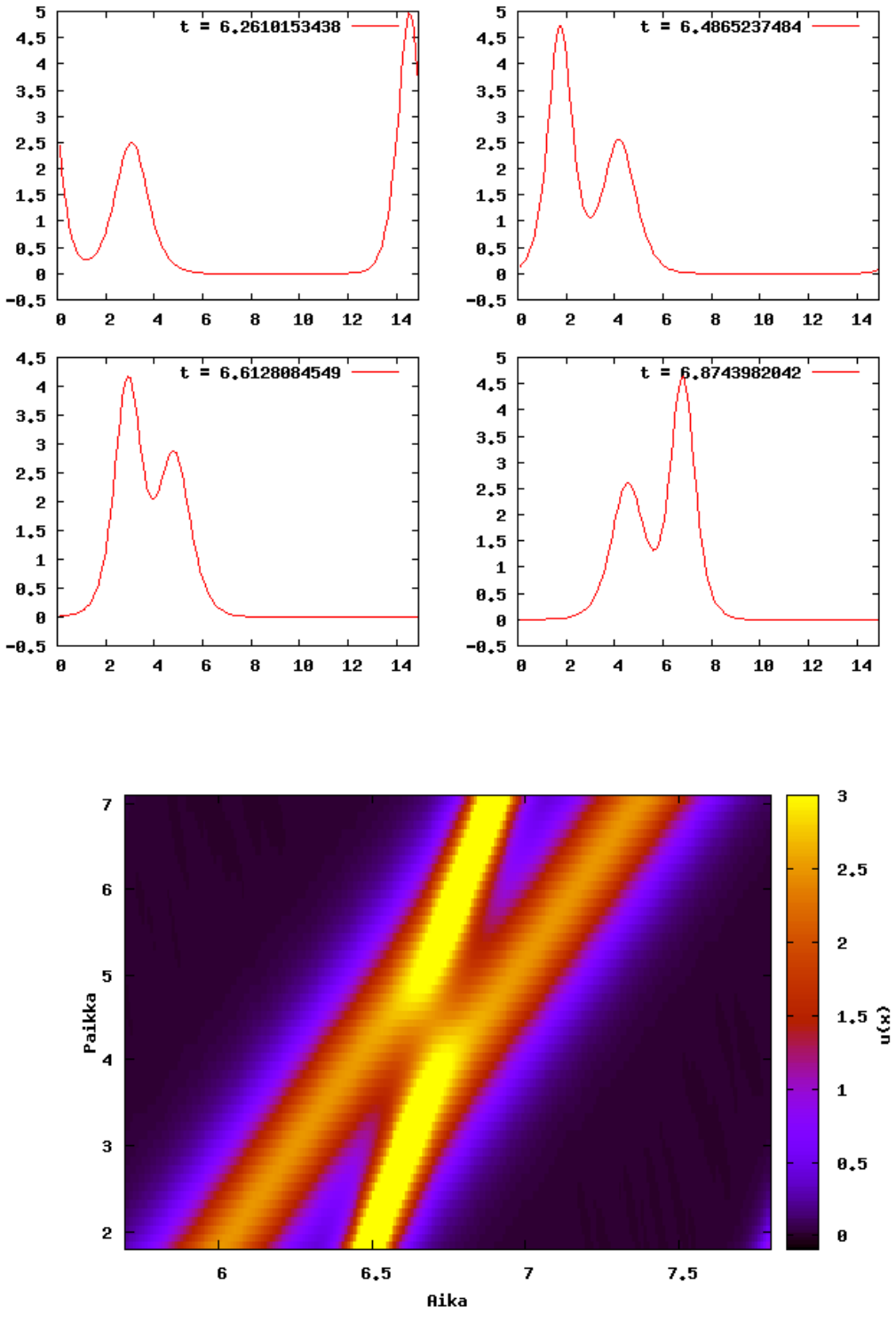
Yhtälö	h	$t_{max,Euler}$	$t_{max,RK}$
Suoraviivainen epäsymmetrinen	0,1	0,00231	0,00228
Suoraviivainen epäsymmetrinen	0,01	0,00000242	0,00000242
Suoraviivainen symmetrinen	0,1	3,17	3,17
Suoraviivainen symmetrinen	0,01	3,72	3,72

Taulukko 4.4. Tuloksia menetelmistä, joilla simulaatio saatiin toimimaan. Simulaatiot tehty yhdellä aallolla nopeudeltaan $v = 5{,}0$ ja hilalla, kooltaan $l = 15{,}0$.

Yhtälö	h	$\Delta t_{max,Euler}$	$\Delta t_{max,RK}$	$\frac{x_{simu}-x_{teor}}{x_{teor}}$
Keskiarvo (4.7)	0,1	$2{,}679 \cdot 10^{-6}$	0,00283373199392	$0{,}275\%$
Keskiarvo (4.8)	0,1	$2{,}679 \cdot 10^{-6}$	0,00283373199392	$0{,}184\%$
Keskiarvo (4.9)	0,1	$2{,}679 \cdot 10^{-6}$	0,00283344862072	$0{,}517\%$
Keskiarvo (4.10)	0,1	$2{,}679 \cdot 10^{-6}$	0,00283316527586	$0{,}917\%$
Zabuskyn ja Kruskalin	0,1	$4{,}238 \cdot 10^{-7}$	0,001001	$-0{,}228\%$
Zabuskyn ja Kruskalin	0,01	$(5{,}689 \cdot 10^{-13})$	$1{,}005 \cdot 10^{-6}$	

myös tarkimpia. Taulukossa 4.3 nähdään maksimiajat, mihin suoraviivaisilla diskretointimenetelmillä yllettiin. Aikadiskretointeina käytettiin Euleria ja Runge-Kuttaa sekä hilavakioina kokeiltiin $h = 0{,}1$ sekä $h = 0{,}01$. Havaitaan, että näitä diskretointeja ei kannata käyttää KdV:n mallintamiseen

Taulukossa 4.4 nähdään aika-askelet, joilla simulaatio alkoi toimimaan niin pitkälle, että aaltojen törmäyttämistä olisi mahdollista tutkia. Aikarajaksi valittiin $t = 10$. Keskiarvomenetelmillä ja neljännen kertaluvun Runge-Kuttalla lasketut arvot ovat tarkemmat kuin muilla, koska niiden väliltä haluttiin löytää eroja. Zabuskyn ja Kruskalin diskretoinnilla käyttäen hilavakiota $h = 0{,}01$ Eulerin menetelmällä olisi kestänyt niin kauan, ettei simulaatiota ajettu. Taulukossa oleva arvo on oletus, joka laskettiin kertomalla mittaustulosten maksimiaikaa $\frac{4}{3}$:lla, kunnes ylitettiin $t = 10$. Hilavakiolla $h = 0{,}1$ saatu tulos ennustettiin kahta kertaluokkaa suuremmasta aika-askeleen arvosta ja vahvistettiin simulaatiolla oikeaksi.



Kuva 4.9. Kahden solitoniaallon törmäys

5 Muita aikaintegrointimenetelmiä

5.1 Two-step Euler

Multistep-menetelmät tarkoittavat aikaiteraatioita, joissa hyödynnetään useampaa edellisistä ajanhetkistä laskemaan uuden ajanhetken tilaa. Yksinkertaisimpana esimerkkinä käytetään kahden askeleen Euleria (two-step Euler):

$$u_{t+1} = u_{t-1} + 2\Delta t f(u_t) \tag{5.1}$$

Kokeillaan mikä vaikutus simulaation toimintaan on vaihtamalla aikaiteraatioyhtälö (2.5) yhtälöön (5.1). Mallinnetaan yhtälöitä (4.3), (4.6), (4.7), (4.8), (4.9), (4.10) ja (4.11) two-step Eulerilla.

Samoin kuin kappaleessa 4.1 ei tälläkään aikaintegroinnilla saatu simulaatiota toimimaan kunnolla. Käytettäessä hilavakiota $h = 0{,}1$ päästiin aika-askelella $\Delta t = 0{,}000018$ ajanhetkeen $t = 0{,}00269$, ennen kuin säilyvät suureet kasvoivat yli sallittujen rajojensa. Yhden askeleen Eulerilla tai Runge-Kuttalla hilavakion pienentäminen vaikutti negatiivisesti simulaation toimivuuteen. Sama tapahtui myös kahden aika-askeleen Eulerilla. Hilavakiota $h = 0{,}05$ käyttäen samalla $\Delta t = 0{,}000018$ aika-askelella kuin edellä toimi simulaatio ajanhetkeen $t = 0{,}000357$ eikä mentäessä aika-askeleen kertaluokkaan 10^{-9} parannusta ollut näkyvissä.

Yhtälön 4.6 mallinnus pysähtyi aika-askelta $\Delta t = 0{,}0007$ pienemmillä askelilla ajanhetkeen $t = 2{,}25$. Tilan kehitys voidaan nähdä kuvaajasta 5.1.

Kun pienennettiin hilavakiota jälleen puoleen $h = 0{,}05$ päästiin aika-askelella $\Delta t = 0{,}00010$ ajanhetkeen $t = 2{,}36$. Edistys näyttää samalta kuin kappaleessa 4.2 Runge-Kuttalla tehdyt mallinnukset. Hilavakion pienentämisestä on siis vain minimaalista hyötyä, kun taas suoritusaika kasvoi 20- tai 10000-kertaiseksi.

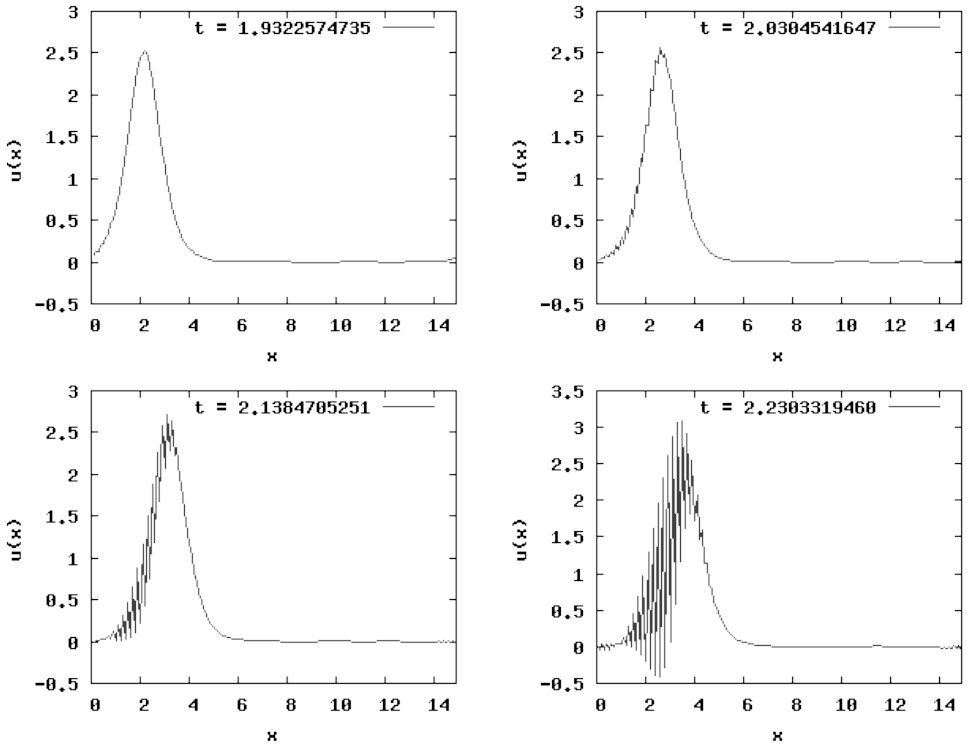
Keskiarvoja käyttävillä yhtälöillä (4.7), (4.8), (4.9) ja (4.10) saatiinkin jo hieman erilaisia tuloksia kuin single-step-menetelmillä. Yhtälöä (4.9) ei saatu etenemään yli ajanhetken $t = 6{,}20$. Muut yhtälöt toimivat yli 10 aikayksikön aika-askelella $\Delta t = 0{,}0010$. Kuvassa 5.2 nähdään epäonnistuneen mallinnuksen tiloja $0{,}1$ aika-askeleen välein. Onnistunut mallinnus yhtälöllä (4.8) nähdään kuvaajassa 5.3.

5.2 Backward Euler

Sen sijaan, että otetaan derivaatan määritelmä yhtälön (2.1) mukaan, kokeillaankin seuraavaa

$$u' = \frac{u(t) - u(t - \Delta t)}{\Delta t},$$

josta saadaan



Kuva 5.1. Yhtälöä (4.6) mallinnettu kahden askeleen Eulerilla. Ajanhetkeen $t = 1{,}9$ simulaatio näyttää toimivan hyvin, mutta sen jälkeen alkaa aallon takaosaan ilmaantua epätasaisuutta.

$$u_{t+1} = u_t + \Delta t \cdot f(u_{t+1}). \tag{5.2}$$

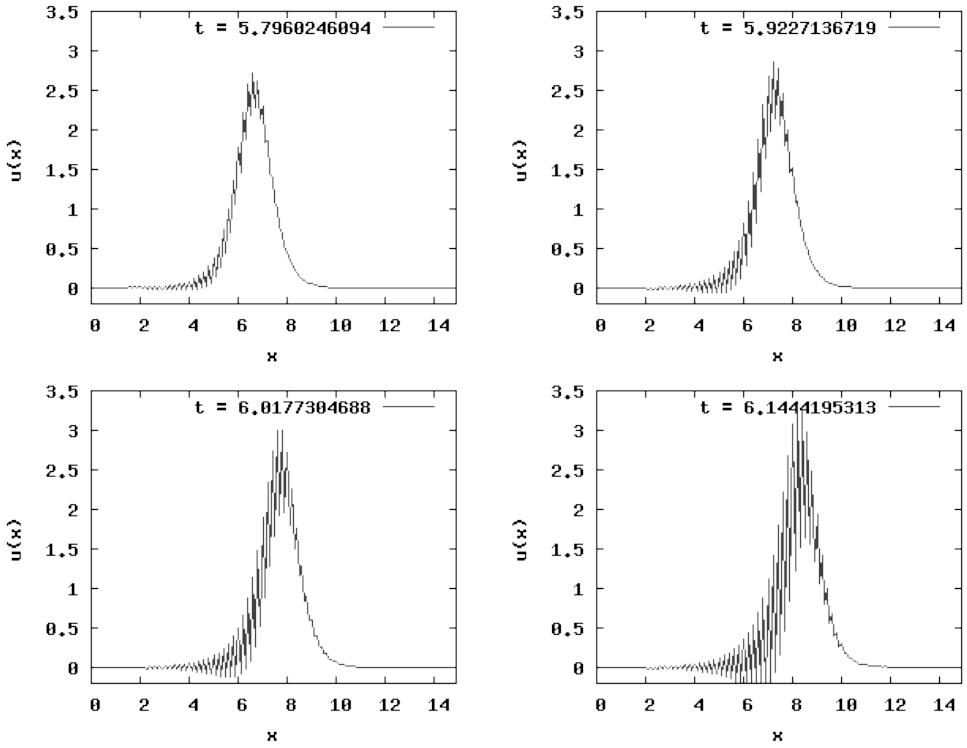
Tilanteesta riippuen tästä saattaa olla mahdollista ratkaista u_{t+1} eksplisiittisesti. Nyt ei kuitenkaan tätä edes yritetä, vaan käytetään yhtälöä muodossa

$$\begin{aligned} \tilde{u}_{t+1} &= u_t + \Delta t \cdot f(u_t), \\ u_{t+1} &= u_t + \Delta t \cdot f(\tilde{u}_{t+1}), \end{aligned} \tag{5.3}$$

jossa siis approksimaatio $\tilde{u}_{t+1}$ on laskettu tavallisella Eulerin menetelmällä.

Ajettiin simulaatio suoraviivaisesta epäsymmetrisestä diskretoinnista eli yhtälöstä (4.3) ja huomattiin tulosten olevan samanlaisia kuin kaikilla aikaisemmillakin aikamallinnuksilla. Jatkossa tämän yhtälön simulointi jätetään väliin kokonaan. Suoraviivaisella symmetrisellä saatiin myös samankaltaisia tuloksia kuin aikaisemminkin. Aika-askelella $\Delta t = 0{,}001001$ päästiin ajanhetkeen $t = 3{,}40$ asti ja tätä pienemmillä aika-askelilla suunnilleen samaan.

Keskiarvoja käyttävillä yhtälöillä päästiin aika-askelella $\Delta t = 0{,}0005631$ ajanhetkeen $t = 10{,}474$. Pienennettäessä aika-askelta $\frac{3}{4}$ edellisestä päästiin jälleen $\frac{4}{3}$ pidemmälle simulaatiossa. Taulukossa 5.1 nähdään simulaatio-ohjelman tulostetta simuloidessa yhtälöä



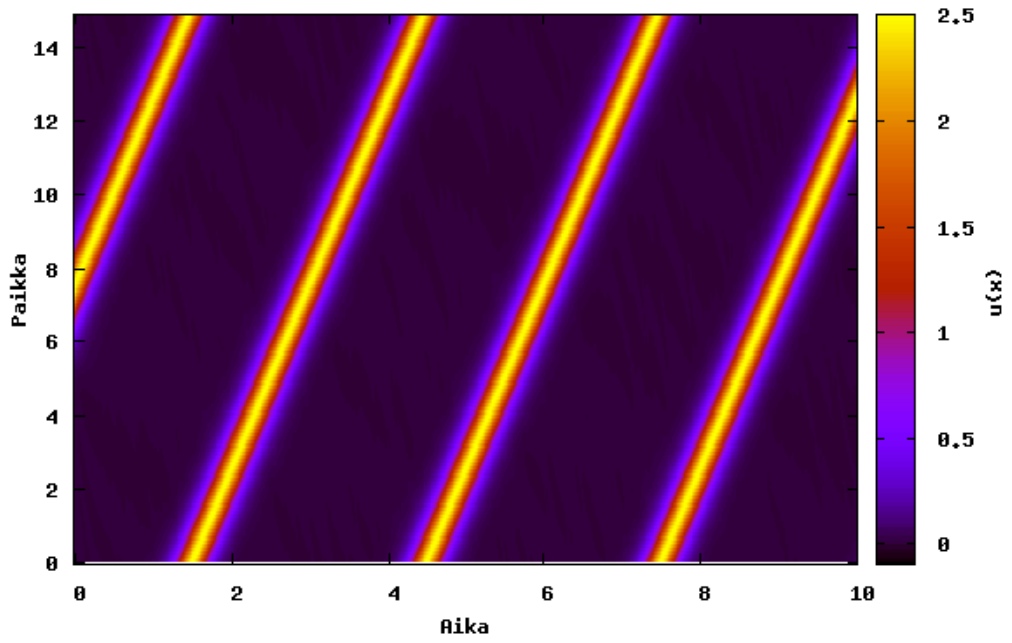
Kuva 5.2. Yhtälön (4.9) mallinnusta two-step Eulerilla. Nähdään, että kuten kuvassa 5.1 alkaa tässäkin ilmaantua epätasaisuutta aallon takaosaan, kunnes tila hajoaa.

(4.8). Aika-askeleen alkuarvona käytettiin $(\Delta t)_0 = 0{,}01$ ja sitä lähdettiin pienentämään aina säilyvien suureiden mentyä pois rajoistaan kaavalla $(\Delta t)_{i+1} = \frac{3}{4}(\Delta t)_i$. Säilyvät suureet testattiin joka sadas iteraatio. Ensimmäinen kerta, kun ne testataan, on siis ajanhetkellä $100 \cdot \Delta t$. Nähdään, että arvolla $\Delta t = 0{,}001001$ alettiin saada tuloksia, joissa mallinnus on edennyt pidemmälle kuin $100 \cdot \Delta t$.

Kaikki yhtälöt (4.7), (4.8), (4.9) ja (4.10) toimivat jälleen lähes samalla tavalla. Ajettiin kaikilla yhtälöillä simulaatio käyttäen aika-askelelta $\Delta t = 0{,}0002376$ ja otettiin tarkasteluun tila ajankohdasta $t \approx 10{,}00$. Esimerkiksi yhtälöllä (4.8) $t = 10{,}0018081516$ ja siten nopeudella $v_1 = 5{,}0$ liikkuva aalto olisi edennyt $50{,}009040758$ yksikköä. Verrataan simulaatioiden tiloista laskettuja aaltojen huippuja tähän. Tulokset ovat taulukossa 5.2.

Tuloksista huomataan, että Runge-Kuttalla mallintamisesta poiketen eniten teorian mukaisia tuloksia saadaan yhtälöllä (4.10). Muutenkin erilaista Runge-Kuttaan nähden on se, että nyt aallot liikkuvat hitaammin, kuin teorian mukaan pitäisi, ja Runge-Kuttalla nopeammin. Lisäksi erotukset ovat nyt suurempia. Kappaleen 4.3 taulukon 4.3 mukaan yhtälö (4.8) noudattaa teoreettista arvoa parhaiten ja ero siihen olisi $0{,}184\,\%$. Nyt paras eli yhtälö (4.10) eroaa teoreettisesta jopa $0{,}7995\,\%$.

Kokeillaan seuraavaksi Backward Euleria kahden aallon tapaukselle. Yhtälöllä (4.10) simulaatio pysähtyy aina $t = 4{,}2$ kohdalle. Voidaan olettaa, ettei suoraviivaisella diskretoinnilla (4.6) päästä tämän pidemmälle. Zabuskyn ja Kruskalin diskretoinnilla kahden



Kuva 5.3. Onnistunut mallinnus yhtälöllä (4.8) käyttäen kahden askeleen Euleria. Nähdään aallon kulkeneen hilan läpi useita kertoja. Simulaatio pysäytettiin ajanhetkeen $t = 10{,}0$, vaikka se olisi toiminut tästä eteenpäinkin.

Taulukko 5.1. Yhtälön (4.8) simuloimista Backward Eulerilla. Pienennettäessä aika-askelta Δt $\frac{3}{4}$:n edellisestä yrityksestä nähdään simulaation toimivan $\frac{4}{3}$ pidemmälle. Maksimiajaksi oli säädetty $t_{max} = 20$, joten tästä eteenpäin simulaation etenemistä ei testattu.

Δt	t
0.01	1.0
0.0075	0.7575
0.005625	0.568125
0.00421875	0.42609375
0.0031640625	0.31640625
0.002373046875	0.2373046875
0.00177978515625	0.179758300781
0.00133483886719	0.134818725586
0.00100112915039	5.90766311645
0.000750846862793	7.88389205933
0.000563135147095	10.474313736
0.000422351360321	13.980252378
0.000316763520241	18.5940186381
0.000237572640181	20.0000527136

Taulukko 5.2. Keskiarvoyhtälöiden mallinnusten tuloksia, kun simulaatiota oli ajettu noin $10{,}00$ aikayksikköä. Ensimmäisen sarakkeen arvot x_{simu} eivät ole aivan tarkkaan keskenään vertailtavissa, sillä kaikista simulaatioista ei tallentunut tilaa saman iteraation kohdalta. Erot tosin näkyvät vasta viidennessä merkitsevässä numerossa. Teoreettinen arvo x_{teor} on laskettu jokaiselle erikseen, joten muut kohdat ovat vertailtavissa.

Yhtälö	x_{simu}	$x_{simu} - x_{teor}$	$\frac{x_{simu}-x_{teor}}{x_{teor}}$
(4.7)	$49{,}2623$	$-0{,}74792$	$-1{,}4956\,\%$
(4.8)	$49{,}2126$	$-0{,}79644$	$-1{,}5924\,\%$
(4.9)	$49{,}3916$	$-0{,}61744$	$-1{,}2347\,\%$
(4.10)	$49{,}6104$	$-0{,}39983$	$-0{,}7995\,\%$

aallon tapaus saatiin aika-askelella $\Delta t = 0{,}0003$ toimimaan noin ajankohtaan $t = 7{,}0$ asti ja tästä eteenpäin jälleen toiminta-aika kääntäen verrannollisesti aika-askeleeseen. Aika-askelella $\Delta t = 0{,}0001002$ päästiin yli maksimiajaksi asetetun $t = 20{,}0$. Simulaatio nähtävissä kuvasta 5.4

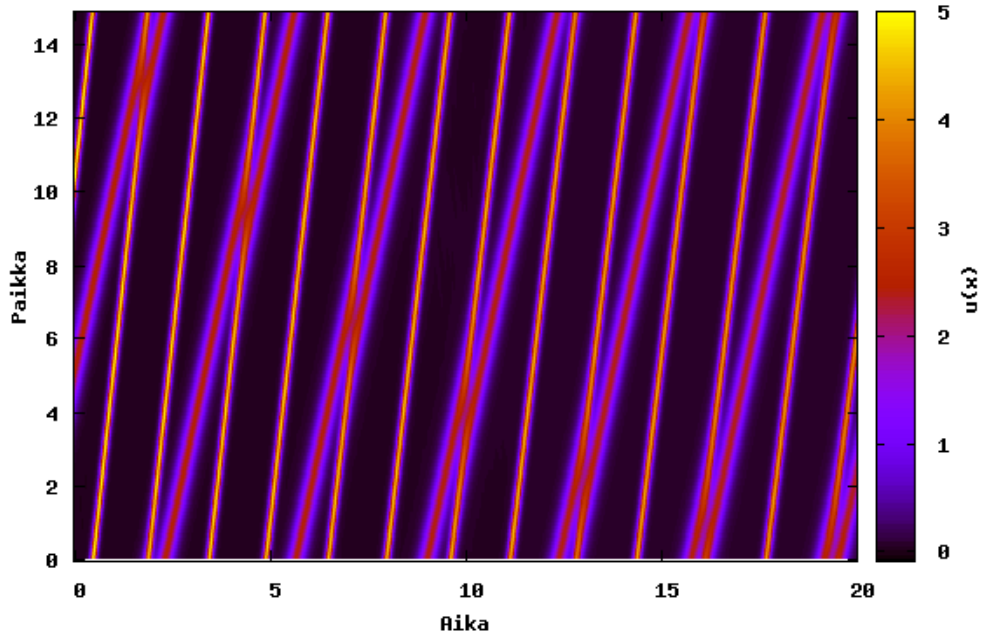
5.3 Adams-Bashforth-Moulton

Kokeilaan seuraavaksi neljännen kertaluvun Adams-Bashforth-Moulton-menetelmää (A-B-M). Tämä koostuu implisiittisestä Adams-Moultonista ja eksplisiittisestä Adams-Bashforthista. Samalla tavalla kuin kappaleessa 5.2 ensin lasketaan eksplisiittisellä menetelmällä approksimaatio $\tilde{u}_{t+1}$, jota käyttäen sitten implisiittisellä todellinen u_{t+1}. Nyt yhtälöinä ovat

$$\begin{aligned} \tilde{u}_{t+1} &= u_t + \frac{\Delta t}{24}\left(55f(u_t) - 59f(u_{t-1}) + 37f(u_{t-2}) - 9f(u_{t-3})\right), \\ u_{t+1} &= u_t + \frac{\Delta t}{24}\left(9f(\tilde{u}_{t+1}) + 19f(u_t) - 5f(u_{t-1}) + f(u_{t-2})\right). \end{aligned} \tag{5.4}$$

Poiketen aikaisemmin käytetyistä menetelmistä, tarvitaan nyt tiedot tilasta neljällä peräkkäisellä ajankohdalla. Ensin approksimaation $\tilde{u}_{t+1}$ laskemiseen tarvitaan tilat ajankohdilta $t, t-1, t-2, t-3$. Itse yhtälössä kuitenkin esiintyy vain funktion $f(u_i)$ arvoja. Laskentatehokkuuden vuoksi näitä ei lasketa joka iteraatiolla uudestaan vaan u:n aikaisempien arvojen sijaan talletetaan muuttujiin $f(u)$:n arvoja. Tilasta u pidetään muistissa vain nykyinen ja säilyvien suureiden laskemisen takia edellinen tila. Iteraation aluksi lasketaan $f(t)$ ja lopuksi kierrätetään muuttujia yhdellä indeksillä taaksepäin. Näin säästetään iteraatiota kohden kolmen suurehkon operaation suoritus. Kokonaissuoritusaikaan tämä vaikuttanee hyvinkin merkittävästi.

Ajettiin simulaatiot kaikilla yhtälöillä. Suoraviivaisilla diskretoinneilla (4.3) ja (4.6) ei saatu taaskaan merkittäviä aikoja toimivia mallinnuksia. Keskiarvoyhtälöillä päästiin yli ajanhetken $t = 10{,}0$ käyttämällä aika-askelta $\Delta t = 0{,}000394$. Zabuskyn ja Kruskalin yhtälöllä (4.11) taas käyttämällä arvoa $\Delta t = 0{,}000121$. Otettaessa huomioon algoritmin



Kuva 5.4. Kahden solitoniaallon törmäys simuloidessa Zabuskyn ja Kruskalin diskretointia Backward Eulerilla. Aika-askeleena käytettiin $\Delta t = 0{,}00010$ ja aaltojen nopeuksina $v_1 = 5{,}0$ ja $v_2 = 10{,}0$.

suoritusaika ja muistissa pidettävien tilojen määrä ei A-B-M-menetelmä vaikuta kovinkaan hyvältä vaihtoehdolta.

Yhtälöillä (4.7) ja (4.8) solitonin nopeuden suhteellinen virhe oli samaa luokkaa kuin paljon A-B-M-menetelmää nopeammalla Two Step Eulerilla. Edes tarkkuutta ei saada tällä menetelmällä siis parannettua.

5.4 Low Storage Runge-Kutta

Tämän työn sovelluksissa ei ohjelman muistinkäyttöön ole puututtu, sillä ohjelmassa on tallessa ollut parhaimmillaan 7 kertaa tila, joka sisältää 150 kertaa 32 bittisen luvun. Tästä saadaan laskettua $32\,\text{bit} \cdot 150 \cdot 7 = 33{,}6\,\text{kbit}$, joka ei nykykoneiden muistiavaruuden huomioiden ole merkittävän paljon. Mallinnettavana kuitenkin saattaa olla paljon suurempia kokonaisuuksia ja muistinkäyttö saattaa nousta merkittäväksikin tekijäksi. Tällaista tilannetta varten kokeiltiin vielä pienen muistikapasiteetin vaativaa neljännen kertaluvun *Low Storage Runge-Kuttaa*.

Neljännen kertaluvun LSRK valittiin, koska sitä haluttiin vertailla tavalliseen neljännen kertaluvun Runge-Kuttaan. Tavallinen Runge-Kutta tarvitsee muistia 5 kertaa

mallinnettavan hilan verran, mutta Low Storage -menetelmillä tämä voidaan vähentää kahteen. Menetelmä voidaan esittää algoritmina

$$\begin{aligned} du^0 &= 0, \\ u^0 &= u_t, \\ \forall i \in [0,n] &\begin{cases} du^{i+1} = A^i \cdot du + \Delta t \cdot f(u^i), \\ u^{i+1} = u^i + B^i \cdot du^{i+1}, \end{cases} \\ u_{t+1} &= u^n \end{aligned} \tag{5.5}$$

jossa kertoimina A^i ja B^i on käytetty "Solution 1"-sarjaa julkaisusta [6]. Kertoimet nähdään taulukossa 5.3.

Taulukko 5.3. Algoritmissa (5.5) käytettävät kertoimet.

i	A^i	B^i
1	0,0	0,0976183546921
2	-0,481231743137	0,412253292916
3	-1,04956260671	0,440216963931
4	-1,60252957428	1,42631146322
5	-1,77826719392	0,197876053732

Samaa algoritmi soveltuu moneen muuhunkin kahden muistipaikan Low Storage Runge-Kutta -menetelmään. Iterointien määrää ja kertoimia A^i ja B^i vaihtamalla saadaan aikaan nopeampi tai tarkempi menetelmä. Yleisesti pyritään käyttämään menetelmiä, joissa vaiheita on yhtä monta kuin menetelmän kertaluku. On kuitenkin osoitettu, ettei neljännen kertaluvun kahden muistipaikan Runge-Kutta-menetelmää voida kirjoittaa nelivaiheisena [7]. Tästä syystä käytetään viisivaiheista menetelmää. Viiden vaiheen käyttö pitäisi myös lisätä menetelmän tarkkuutta hieman tavallista neljännen kertaluvun Runge-Kuttaa paremmaksi.

Mallinnettiin taas kaikkia käsiteltävistä yhtälöistä. Suoraviivaisella symmetrisellä yhtälöllä (4.6) päästiin korkeintaan ajanhetkeen $t = 3{,}19$, mutta jo aika-askeleella $\Delta t = 0{,}00323$. Samalla aika-askeleella $\Delta t = 0{,}00323$ saavutettiin keskiarvoyhtälöillä ajanhetki $t = 10{,}0$. Zabuskyn ja Kruskalin yhtälöllä päästiin yli ajanhetken $t = 10{,}0$ aika-askeleella $\Delta t = 0{,}00128$.

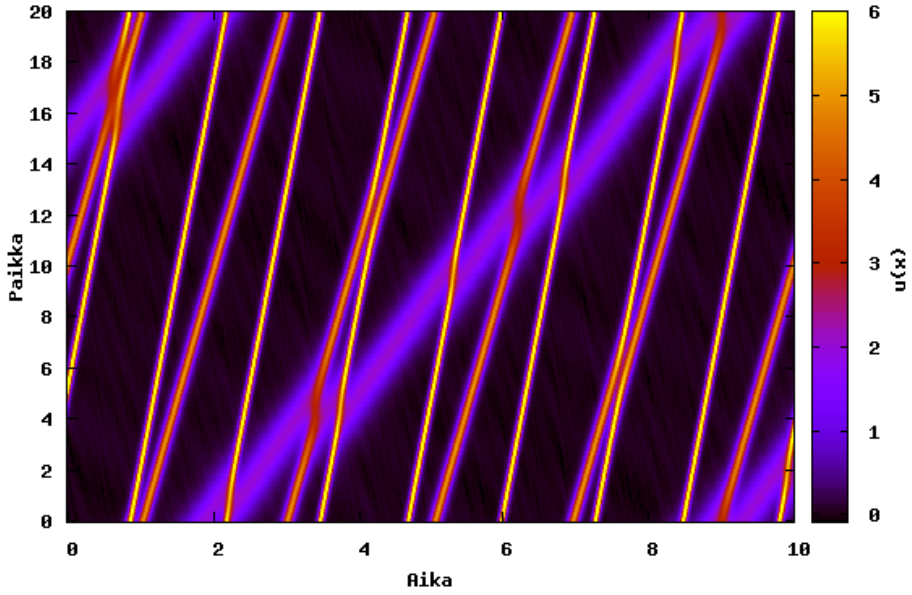
Samoin kuin edellä laskettiin tässäkin suhteelliset virheet solitoniaallon huipun sijainnille. Keskiarvoyhtälöllä (4.8) saatiin virheeksi $0{,}212\,\%$ ja Zabuskyn ja Kruskalin yhtälöllä (4.11) $-0{,}262\,\%$.

Huomataan, että tällä menetelmällä suoritetut mallinnukset toimivat suuremmalla aika-askeleella kuin tavallisella neljännen kertaluvun Runge-Kuttalla aallon etenemisnopeudessa havaittavan virheen ollessa suunnilleen samaa luokkaa.

6 Sovelluksia

6.1 Kolmen aallon tapaus

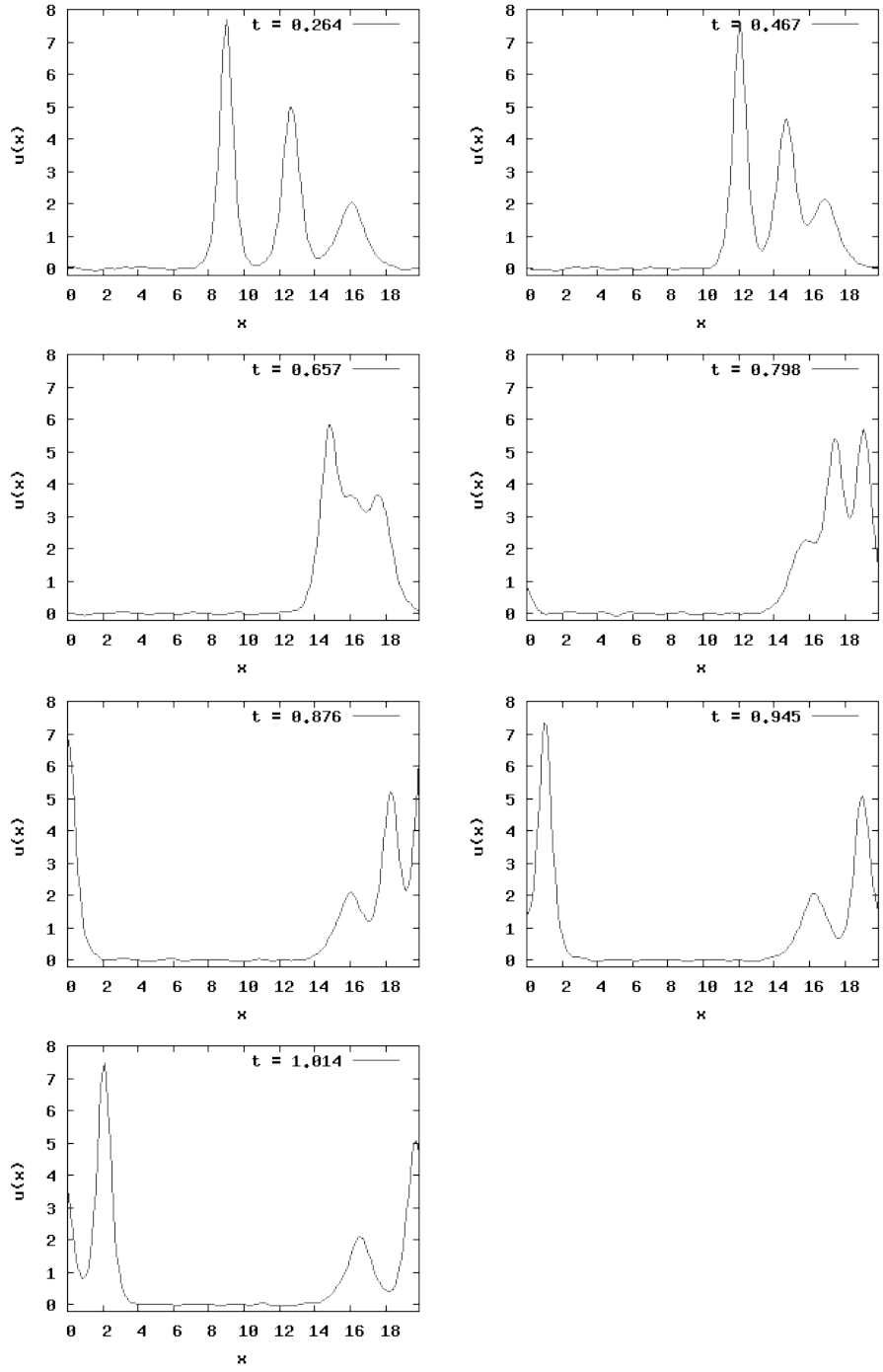
Kahden aallon törmäyksiä kokeiltiin jo monilla onnistuneilla menetelmillä. Kokeiltiin myös kolmen aallon alkutilan kehitystä. Yhtälönä käytettiin (4.8) ja aika-iteraationa Runge-Kuttaa. Hilavakio oli $h = 0{,}1$ ja aaltojen nopeudet $v_1 = 15{,}0$, $v_2 = 10{,}0$ ja $v_3 = 4{,}0$. Hilan pituudeksi valittiin tällä kertaa $l = 20{,}0$, jotta aallot olisivat tarpeeksi etäällä toisistaan alkutilassa. Alkutila laskettiin edelleen yksittäisistä aalloista, jotka summattiin keskenään. Maksimiajaksi asetettiin $t = 10$. Mallinnuksen kehitys nähdään kuvassa 6.1.



Kuva 6.1. Kolmen aallon mallinnuksen kehitys. Nähdään useita törmäyksiä, mutta vain yksi kolmen aallon törmäys. Tämä havaitaan aikavälillä $t \in [0{,}5; 1{,}0]$. Aaltojen nopeudet ovat $v_1 = 15{,}0$, $v_2 = 10{,}0$ ja $v_3 = 4{,}0$.

Kuvassa 6.2 nähdään tilan kehitys välillä $t \in [0{,}264; 1{,}014]$. Tässä tapahtui ensimmäinen ja simulaatioajan ainut kunnollinen kolmen aallon törmäys. Loput törmäykset olivat ennemminkin kaksi kahden aallon törmäystä peräkkäin. Kuvasta 6.1 nähdään tilanteessa syntynyt vaihesiirtymä.

Kolmen aallon tapausta ei verrata analyyttiseen ratkaisuun, sillä kolmella aallolla sellainen on aivan liian monimutkainen.



Kuva 6.2. Seitsemän pysäytyskuvaa kolmen aallon simulaatiosta. Havaitaan aaltojen lähenevän toisiaan ja kolmannessa kuvassa sulautuvan yhteen. Neljännessä kuvassa ohitusta on jo tapahtunut. Ohituksen jälkeinen aallot jatkavat eriytymistään. Mielenkiintoista näissä seitsemässä kuvassa on, kuinka matalin aalto on viimeisessä kuvassa samassa kohtaa kuin ensimmäisessäkin vaikka aikaa on kulunut lähes $0{,}8$ yksikköä. Nopeudeltaan 4 olevan aallon olisi pitänyt edetä jo yli 3 matkayksikköä. Tässä havaitaan solitonien törmäyksessä tapahtuva vaihesiirtymä

6.2 Väärän muotoinen aalto

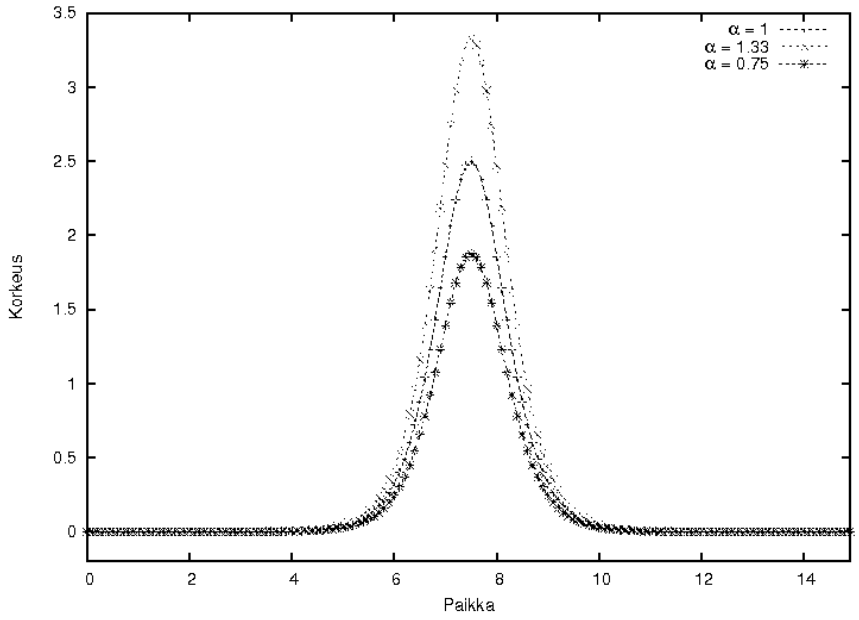
Alkutilan aallot ovat kaikki ratkaistu yhtälöstä (3.1). Kokeillaan, miten mallinnuksissa käyttäytyisi aalto, joka on muodoltaan lähes KdV:n ratkaisun mukainen, mutta esimerkiksi leveämpi tai kapeampi kuin pitäisi. Yhtälössä

$$u(x,t) = \frac{v/2}{\cosh^2\left(\frac{\sqrt{v}}{2}\left(x - vt - x_0\right)\right)}$$

jakoviivan yläpuolella oleva $\frac{v}{2}$ määrää aallon korkeuden. Lisätään yhtälöön kerroin α seuraavasti:

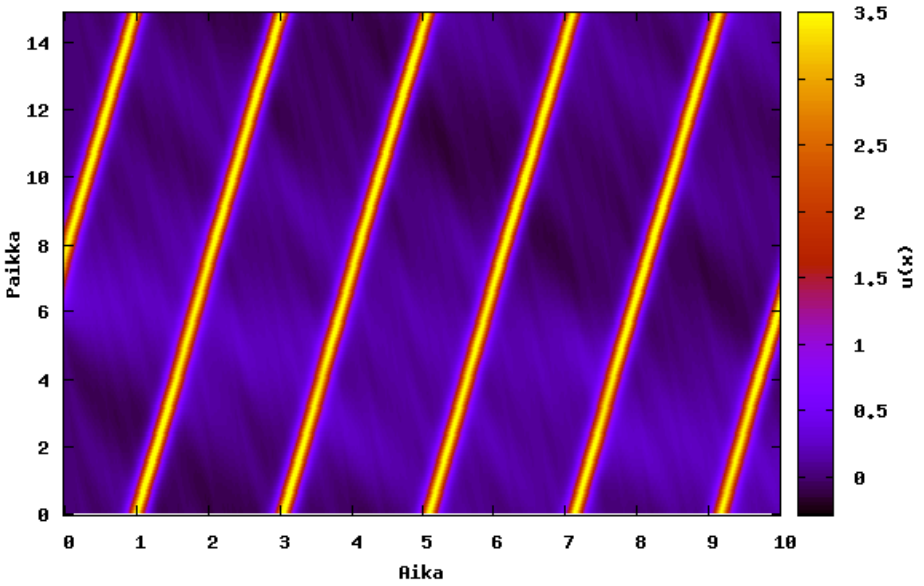
$$u(x,t) = \frac{\alpha(v/2)}{\cosh^2\left(\frac{\sqrt{v}}{2}\left(x - vt - x_0\right)\right)}. \tag{6.1}$$

Lasketaan nyt alkutilat kertoimen α arvoilla $\alpha_1 = 0{,}7$ ja $\alpha_2 = 1{,}33$. Nämä sekä yhtälön (3.1) mukaiset aallot nähdään kuvaajassa 6.3.

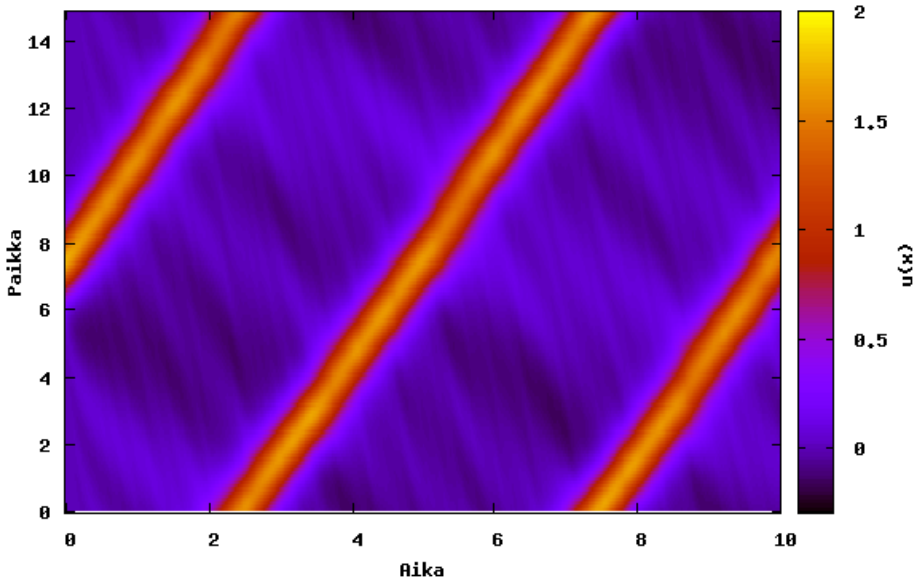


Kuva 6.3. Yhtälöstä (6.1) eri muuttujan α arvoilla laskettuja alkutiloja. Keskellä nähdään oikea KdV:n ratkaisun mukainen aalto ja sen lisäksi liian korkea ja liian matala aalto.

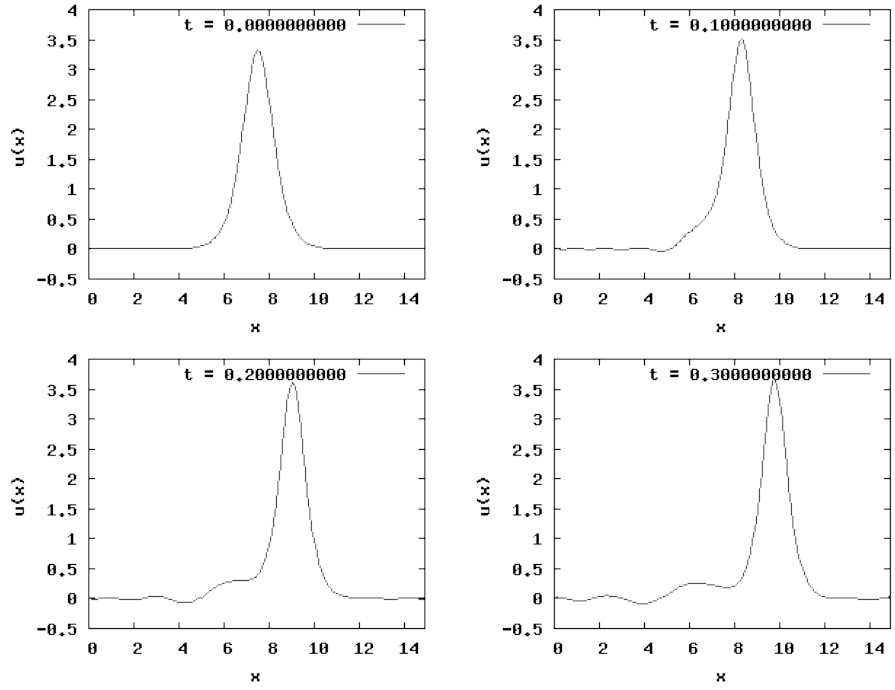
Kuvassa 6.6 nähdään liian korkean aallon ($\alpha = 1{,}33$) kehitystä alkutilasta. Huomataan, että aalto muotoutuu KdV:n ratkaisun muotoiseksi, mutta hilaan jää paljon epätasaisuuksia. Kuvasta 6.4 nähdään epätasaisuuksien säilyvän hilassa simulaation loppuun, mutta ne eivät häiritse aallon etenemistä. Samasta kuvasta voidaan nähdä aallon kiertävän hilan hieman yli kahdessa aikayksikössä. Kun hilan pituus on $l = 15{,}0$, voidaan laskea aallon nopeuden olevan hieman alle $v = 7{,}5$. Alkutilaa laskiessa nopeudeksi annettiin $v = 5{,}0$, mutta α ilmeisesti muutti sitä hieman.



Kuva 6.4. Liian korkean aallon ($\alpha = 1{,}33$) koko simulaatio. Huomataan sen etenevän hyvin solitonimaisesti. Aallon ulkopuolella näkyy hyvin paljon epätasaisuutta. Esimerkiksi verrattuna kuvaan 5.3 sivulla 32 nähdään poikkeavuus hyvin. Tästä kuvasta voidaan laskea aallon nopeudeksi noin $7{,}5$, joka on huomattavan paljon enemmän kuin yhtälölle 3.1 annettu $5{,}0$.



Kuva 6.5. Liian matalan aallon ($\alpha = 0{,}70$) koko simulaatio.Nähdään epätasaista etenemistä ja hyvin paljon epätasaisuutta aallon ulkopuolella. Taustakohina jopa häiritsee aallon etenemistä ja esimerkiksi ajanhetkellä $t = 5{,}8$ nähdään hieman vaihesiirtymän kaltainen ilmiö.



Kuva 6.6. Liian korkean aallon ($\alpha = 1{,}33$) alkutilan kehitystä ensimmäisten 3000 iteraation aikana ($\Delta t = 0{,}0001$). Huomataan kuinka aalto jätti taakseen epätasaisuutta.

Muuttujan α arvolla $0{,}7$ saatiin kuvan 6.5 mukainen tulos. Kuvasta voidaan nähdä aallon ulkopuoleisen epätasaisuuden vaikuttavan aallon etenemiseen. Esimerkiksi aikavälillä $5{,}0 < t < 6{,}0$ voidaan nähdä hieman aaltojen törmäyksessä tapahtuvan vaihesiirtymän tapainen ilmiö. Nopeutena aallolla on $v \approx 3{,}0$, eli hitaampi kuin yhtälölle annettu $v = 5{,}0$.

6.3 Kanttiaalto

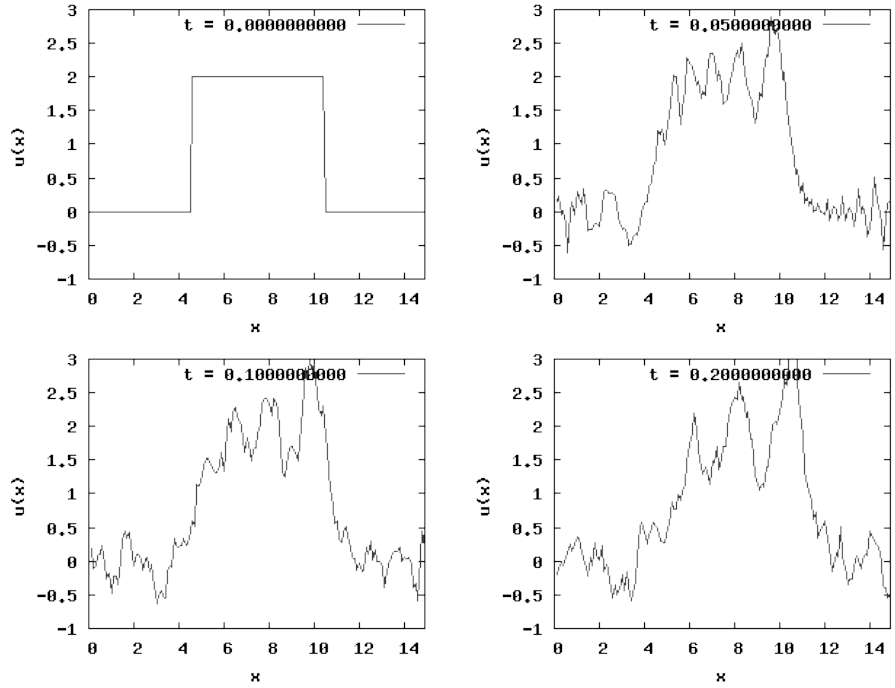
Sen sijaan, että asetetaan alkutilaksi KdV:n analyyttisen ratkaisun (3.1) mukainen aalto, kokeillaan, mitä yksinkertaiselle kanttiaallolle tapahtuu. Alkutilasta siis tehdään

$$u(x) = 0, \text{ kun } x < 4{,}5,$$
$$u(x) = 2, \text{ kun } 4{,}5 \leq x \leq 10{,}5,$$
$$u(x) = 0, \text{ kun } 10{,}5 < x.$$

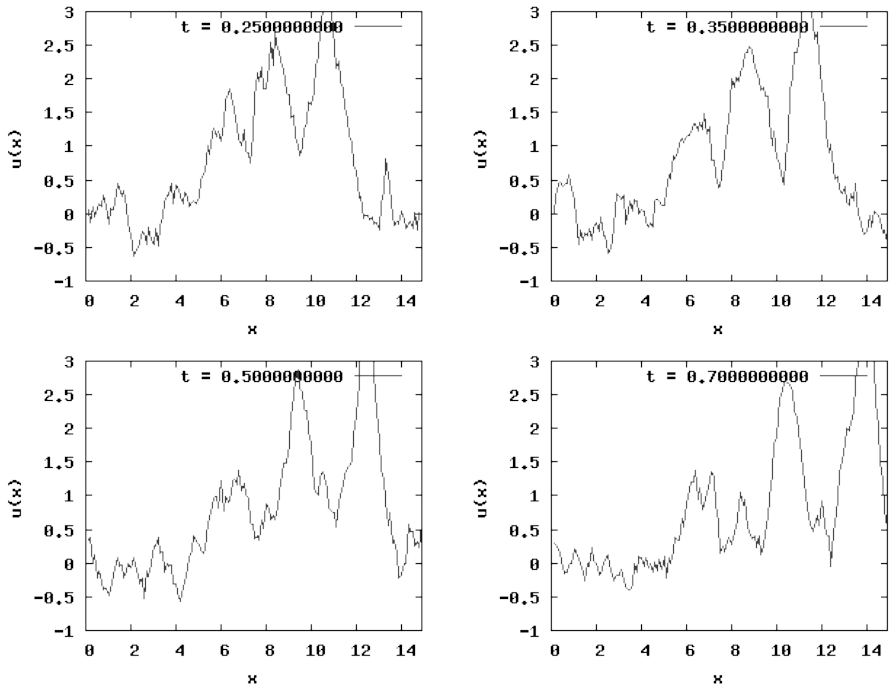
Mallinnuksessa käytettiin hyväksi havaittua yhtälöä (4.8) ja Runge-Kuttaa. Ajettiin simulaatiota aika-askeleen alkuarvoilla $\Delta t = 0{,}001$ sekä $\Delta t = 0{,}0001$. Mallinnettaessa yhtälön (3.1) mukaista aaltoa toimi simulaatio erittäin hyvin kummallakin näistä.

Mallinnukset päättyivät hyvin nopeasti. Kolmas säilyvä suure (3.16) ei pysynyt rajoissaan kovinkaan pitkään. Kaksi ensimmäistä kuitenkin pysyivät, joten kokeiltiin kyt-

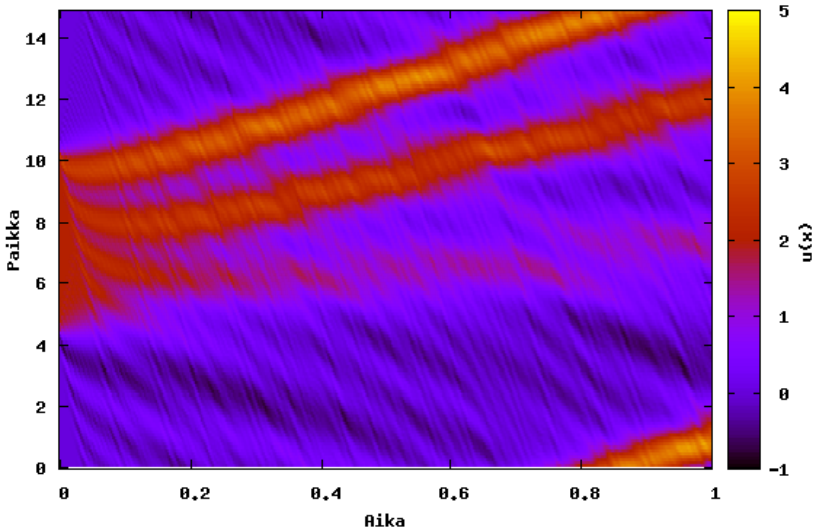
keä kolmannen säilyvän suureen tarkastaminen pois päältä ohjelmakoodin tasolla. Nyt päästiinkin pitkälle. Kuvissa 6.7 ja 6.8 nähdään 8 tilaa väliltä $t \in [0; 0{,}7]$. Huomataan, kuinka kanttiaalto nopeasti hajoaa kolmeksi selvästi erilliseksi aalloksi. Nämä kuitenkaan eivät saavuta KdV:n analyyttisen ratkaisun (3.1) muotoa, vaan pysyvät epämääräisenä. Vielä kuvassa 6.9 koko simulaatio.



Kuva 6.7. Leveydeltään 6 ja korkeudeltaan 2 olevan kanttiaallon tilan kehitys. Nähdään alkutilan hyvin nopeasti hajoavan kolmeksi selkeästi erilliseksi aalloksi.



Kuva 6.8. Jatkoa kuvalle 6.7. Kanttiaallon hajottua kolmeksi erilliseksi aalloksi, jatkavat nämä etenemistään omilla nopeuksillaan. Näiden aaltojen lisäksi tilassa havaitaan huomattavan paljon taustakohinaa.



Kuva 6.9. Leveydeltään 6 ja korkeudeltaan 2 olevan kanttiaallon koko mallinnus. Nähdään hajoaminen kolmeen aaltoon ja lisäksi kohina koko hilan pituudelta.

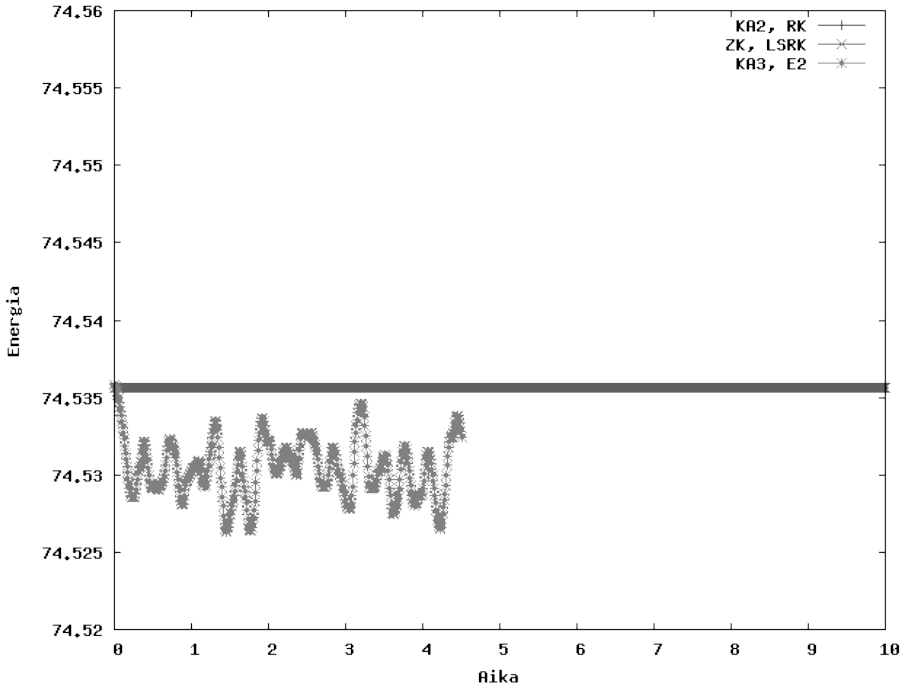
7 Tuloksia

7.1 Säilyvät suureet

KdV:n säilyviä suureita on käytetty kuvaamaan mallinnuksen toimivuutta itse ohjelman tasolla. Katsotaan seuraavaksi, miten ne itse asiassa käyttäytyvät. Valittiin vertailtavaksi kolme eri mallinnusta: keskiarvoyhtälö (4.8) neljännen kertaluvun Runge-Kuttalla, Zabuskyn ja Kruskalin yhtälö (4.11) Low Storage Runge-Kuttalla sekä keskiarvoyhtälö (4.9) Two Step Eulerilla. Näistä kaksi ensimmäistä valittiin, koska ne ovat erittäin hyvin toimivia, ja kolmas, koska se ei toimi.

Kaikki diskretoinnit on tehty sellaisiksi, että ne säilyttävät ensimmäisen säilyvän suureen erittäin hyvin. Tietokoneen suoman tarkkuuden rajoissa ei siinä huomattu millään toimivalla mallinnuksella yhtään muutosta.

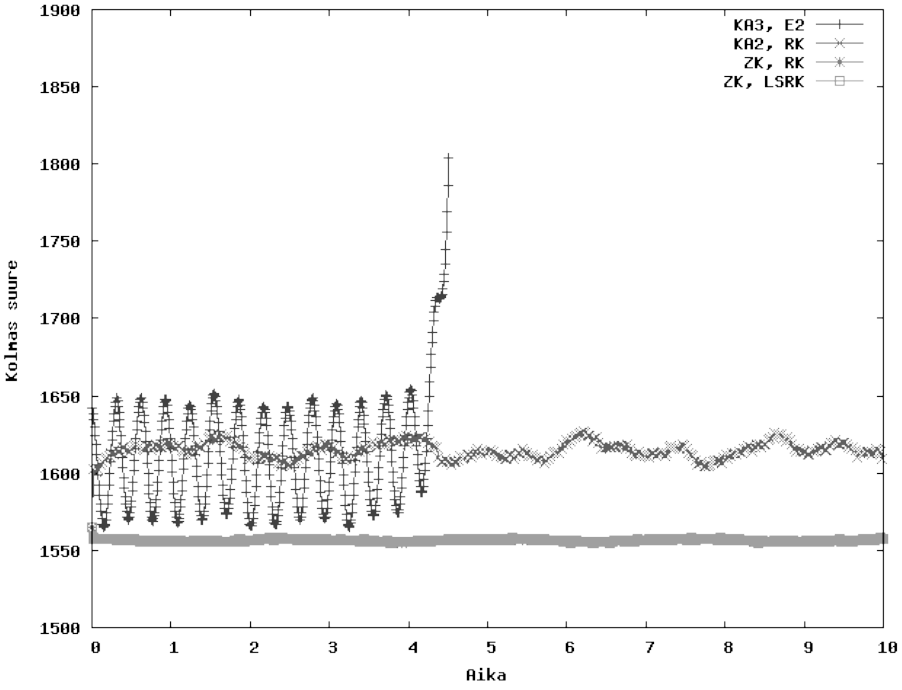
Toinen säilyvä suure (3.14) säilyy sekin toimivilla menetelmillä erittäin hyvin. Kuvasta 7.1 nähdään, että yhtälöllä (4.9) ja Two Step Eulerilla toinen säilyvä suure ei todellisuudessa säily, mutta kummallakin toimivalla menetelmällä säilyy jälleen erittäin hyvin. Huomataan kuitenkin, että yhtälöllä (4.8) ja neljännen kertaluvun Runge-Kuttalla toisen säilyvän suureen arvo pienenee hyvin pienen määrän tässä kymmenen aikayksikön pituisessa mallinnuksessa.



Kuva 7.1. Toisen säilyvän suureen (3.14) eli energian kehitys ajan funktiona kolmella eri mallinnusmenetelmällä.

Kolmannessa säilyvässä suureessa saatiinkin jo eroja aikaan. Huomataan yhtälön (4.9) mallinnuksessa kolmannen säilyvän suureen vaihtelevan nopeasti välillä [1560,

1650] ja lopulta nousevan lähes arvoon 1900, jolloin ohjelma keskeyttää simulaation. Myös yhtälön (4.8) arvo vaihtelee merkittävän paljon. Zabuskyn ja Kruskalin yhtälön kolmas säilyvä suure pysyy paremmin vakiona. Pientä vaihtelua siinäkin on havaittavissa, mutta määrältään se ei ole edes verrattavissa muihin. Kuvaan otettiin mukaan vielä Zabuskyn ja Kruskalin yhtälö mallinnettuna tavallisella Runge-Kuttalla, jotta huomataan suureen (3.16) säilyvyyden johtuvan yhtälöstä eikä aikaintegroinnista.



Kuva 7.2. Kolmannen säilyvän suureen (3.16) kehitys ajan funktiona neljällä eri mallinnuksella.

7.2 Tuloksen poikkeama teoreettisesta

Säilyvien suureiden lisäksi tutkittiin itse aallon etenemisnopeutta. Yksittäisen solitoniaallon kuuluisi edetä vakionopeudella ja koska tätä nopeutta on käytetty laskemaan mallinnuksen alkutila, on nyt helppo selvittää mallinnuksen lopputuloksen poikkeama teoreettisesta arvosta. Lopputila luettiin ohjelman kirjaamasta datatiedostosta ja sen hilapisteisiin sovitettiin Gnuplot-ohjelmalla yhtälön (3.1) muotoinen käyrä. Näin saatiin mallinnuksen määräämä huipun sijainti x_{sim}. Teoreettinen arvo x_{teor} laskettiin vain käyttämällä alkuarvona annettua aallon nopeutta, sekä mallinnusaikaa. Nyt laskettiin näiden suhteellinen ero kaavalla

$$\frac{x_{sim} - x_{teor}}{x_{teor}}. \tag{7.1}$$

Lasketut virheet nähdään taulukossa 7.1. Huomataan, että jälleen kerran Runge-Kutta

menetelmillä saatiin parhaat tulokset. Muilla testatuilla menetelmillä saatiin myös joillekin yhtälöille hyviä tuloksia paitsi Backward Eulerilla, jolla poikkeama oli aina yli prosentin luokkaa.

Taulukko 7.1. Laskettiin suhteelliset virheet $\frac{x_{sim}-x_{teor}}{x_{teor}}$ solitoniaallon huipun sijainnille eri menetelmillä mallinnettuna. Erikoisena piirteenä huomataan, että Zabuskyn ja Kruskalin yhtälön virhe on aina negatiivinen ja Backward Euleria lukuunottamatta lähes sama joka kerta.

	Euler	Runge-Kutta	2-step E.	Backward E	A-B-M	LSRK
KA1	0,341 %	0,275 %	0,317 %	−1,495 %	0,317 %	0,317 %
KA2	0,235 %	0,184 %	0,212 %	−1,592 %	0,212 %	0,212 %
KA3	-	0,517 %	0,481 %	−1,235 %	0,598 %	0,598 %
KA4	-	0,917 %	1,059 %	-	1,059 %	1,059 %
ZK	-	−0,228 %	−0,264 %	−2,518 %	−0,263 %	−0,285 %

7.3 Aika-askeleet ja suoritusaika

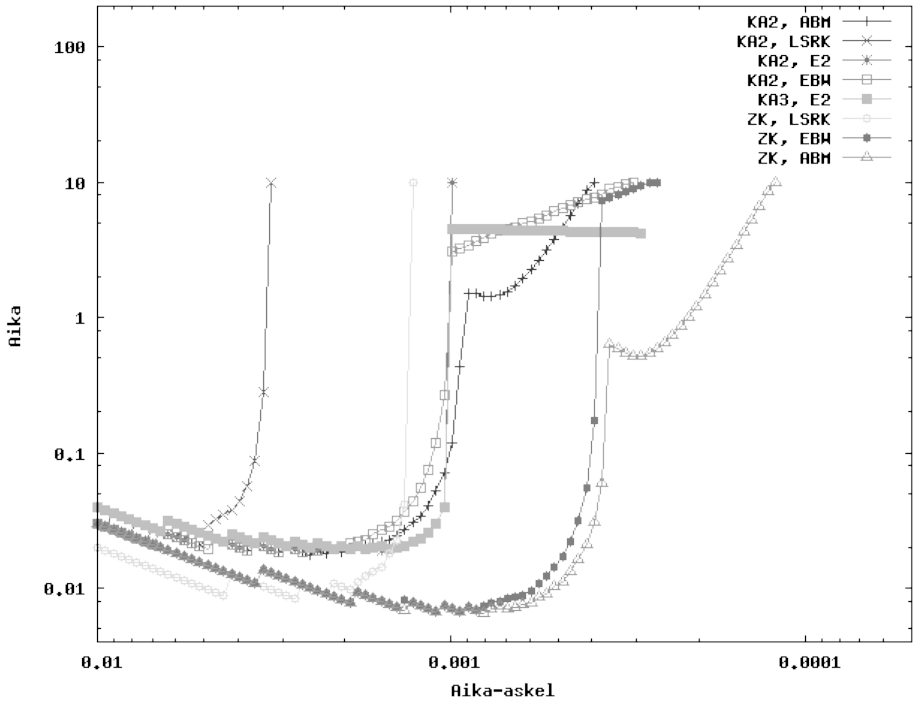
Testattujen menetelmien työmäärä iteraatiota kohden oli kaikissa se verran lähellä toisiaan, että käytetyn aika-askeleen pituus oli lähes aina ratkaiseva tekijä menetelmän tehokkuudessa. Mitä suuremmalla aika-askeleella simulaatioita voitiin ajaa, sitä nopeammin simulaatio saatiin ajettua.

Taulukko 7.2. Suurimmat aika-askeleen arvot, millä kukin menetelmä saatiin toimimaan yli 10 aikayksikköä. Naivi epäsymmetrinen (4.3) sekä naivi symmetrinen yhtälö (4.6) jätettiin pois vertailusta, sillä niitä ei saatu millään aika-askeleella toimimaan kovin pitkiä aikoja.

	Euler	Runge-Kutta	2-step E.	Backward E.	A-B-M	Low S. RK
KA1	$3{,}31 \cdot 10^{-6}$	0,00277	0,000994	0,000305	0,000394	0,003235
KA2	$3{,}31 \cdot 10^{-6}$	0,00277	0,000994	0,000305	0,000394	0,003235
KA3	$3{,}31 \cdot 10^{-6}$	0,00277	-	0,000305	0,000394	0,003235
KA4	$3{,}31 \cdot 10^{-6}$	0,00277	0,000994	0,000305	0,000394	0,003235
ZK	$< 1{,}0 \cdot 10^{-6}$	0,00105	0,000375	0,000262	0,000121	0,001285

Taulukossa 7.2 nähdään ne aika-askeleet, millä simulaatio saatiin toimimaan yli 10 aikayksikköä. Huomataan Runge-Kutta-menetelmien olevan huomattavasti muita parempia. Näistä vieläpä Low Storage -menetelmä toimi hiukan suuremmilla aika-askelilla, johtuen sen viisivaiheisuudesta. Havaitaan myös Backward Eulerin ja Adams-Bashforth-Moulton-menetelmän alkavan toimimia suunnilleen samansuuruisella aika-askeleella. Two Step Eulerin ja tavallisen Eulerin ero on myös mielenkiintoinen yksityiskohta. Hyvinkin samankaltaiset menetelmillä selvästi erilaiset tulokset. Paikkadiskretoinnissa samalla derivaatan ottamisella epäsymmetrisesti tai symmetrisesti ei ollut näin ratkaisevaa eroa.

Aika-askelien vertailua nähdään myös kuvassa 7.3. Mitä enemmän yläoikealla pisteet ovat, sitä suuremmilla aika-askeleen arvoilla päästään pidemmälle mallinnuksessa. Mukaan vertailuun otettiin vain keskiarvoyhtälöistä parhaiten suoriutunut, eli yhtälö (4.8), sekä Zabuskyn ja Kruskalin yhtälö (4.11). Aikaintegroinneista testattiin näillä menetelmillä kaikki. Mukaan kuvaajaan otettiin myös keskiarvoyhtälö KA3 (4.9) Two Step Eulerilla integroituna, sillä sen käytös poikkesi merkittävästi muista keskiarvoyhtälöistä.



Kuva 7.3. Eri menetelmien toiminta-aikoja aika-askeleen funktiona logaritmisessa koordinaatistossa. Nähdään siis, miten kauan aikaa mallinnus toimii milläkin aika-askeleella.

Pelkästä aika-askeleen suuruudesta ei voida suoraan päätellä menetelmän tehokkuutta mallinnuksessa. Eri paikkadiskretointeja laskettaessa saatetaan käyttää eri määrää hilapisteitä ja aikaintegroinneissa tehdään eri määrä työtä askelta kohti. Suoritusaika mitattiin kirjaamalla ylös tietokoneen kellon aika ennen ja jälkeen suorituksen ja lopussa laskemalla näiden erotus. Tämä vielä jaettiin suoritetun mallinnuksen viimeisellä kirjatulla ajanhetken t arvolla, niin saatiin suoritusaika mallinnuksen aikayksikköä kohden. Tulos riippuu käytettävän tietokoneen nopeudesta sekä siitä, mitä muita prosesseja taustalla on käynnissä. Taulukossa 7.3 olevia arvoja mitattaessa taustaprosessit pyrittiin pitämään koko ajan samoina. Ne ovat siis vertailtavia keskenään, mutta eivät minkään muun kanssa. Tästä syystä ajan yksikkö jätettiin pois taulukosta.

Huomataan, että jälleen Runge-Kutta-menetelmät suoriutuivat parhaiten, joskin Two Step Euler pärjäsi niille kohtuullisen hyvin suhteellisen suuren aika-askeleensa ja varsinkin pienen työmääränsä ansiosta.

Taulukko 7.3. Menetelmien suoritusajoista nähdään, että suurta aika-askelta hyödyntävät pärjäsivät paremmin kuin pienempää vaikka niillä tehtäisiinkin enemmän työtä iteraatiota kohden. Parhaiten tässä pärjäsivät Runge-Kutta-menetelmät.

	Euler	Runge-Kutta	2-step E.	Backward E.	A-B-M	LSRK
KA1	1600	3,7	4,2	20	16	3,5
KA2	1600	3,6	4,6	19	17	3,3
KA3	1700	4,1	-	24	16	4,2
KA4	1800	3,8	8,5	24	21	4,4
ZK	1600	8,7	11	22	48	9,4

8 Yhteenveto

Työssä tutkittiin erilaisten menetelmien soveltuvuutta KdV-yhtälön diskretointiin ja aikaintegrointiin. Kokeiltuja diskretointeja olivat suoraviivainen epäsymmetrinen (4.3), suoraviivainen symmetrinen (4.6), neljä erilaista tapaa käyttää keskiarvoa (4.7), (4.8), (4.9) ja (4.10) sekä Zabuskyn ja Kruskalin artikkelissaan [3] tutkimaa yhtälöä (4.11).

Testattuja aikaintegrointeja olivat Eulerin menetelmä (2.5), kahden aika-askeleen Euler (5.1), backward Euler (5.2), Adams-Bashforth-Moulton (5.4), neljännen kertaluvun Runge-Kutta (2.14) sekä pienen tallennuskapasiteetin neljännen kertaluvun Runge-Kutta (5.5).

Kokeiltujen menetelmien välille saatiin kaikilla mittapuilla selkeitä eroja. Diskretoinneista keskimäärin parhaiten suoriutui keskiarvoyhtälö KA2 (4.8). Suoritusaika oli lähes aina pienin ja lisäksi menetelmän virhe aallon etenemisnopeudelle oli kaikilla aikaintegroinneilla pienin. Tutkiessa säilyviä suureita, huomattiin kolmannen suureen (3.16) vaihtelevan melko laajalla välillä. Rajoiksi asetettu $\pm 20\,\%$ kuitenkin riitti.

Toinen merkittävä diskretointi oli Zabuskyn ja Kruskalin yhtälö (4.11). Tulokset poikkesivat teoreettisesta arvosta saman noin $-0{,}25\,\%$ kaikilla aikaintegroinneilla. Suoritusaika yhtälöä käyttäen oli yli kaksinkertainen verrattuna keskiarvoyhtälöihin, mutta toisaalta säilyvistä suureista kaikki kolme laskettua säilyivät erittäin hyvin.

Aikaintegroinneista Runge-Kutta menetelmät suoriutuivat parhaiten. Vaikka neljännen kertaluvun menetelmä onkin, ei A-B-M pärjännyt Runge-Kutta-menetelmille suoritusajassa eikä tarkkuudessa. Low-Storage Runge-Kutta 5.4 suoriutui vielä tavallista RK4:ää 2.14 paremmin. Syynä tähän on sen käyttämät viisi vaihetta, joiden ansiosta sen tarkkuus on hieman RK4:ää parempi. Nelivaiheista menetelmää ei kuitenkaan kahden muistipaikan menetelmille saada aikaan, joten tätä vertailua ei voitu suorittaa.

Laskentatehokkuudessa pärjäsi hyvin myös Two Step Euler 5.1 menetelmän yksinkertaisuuden vuoksi. Se toimi vielä suhteellisen isolla aika-askelella, mutta koska aikaintegrointia kohden tehtiin vain yksi laskutoimitus, menetelmä pärjäsi hyvin sitä työläämmille.

Lähteet

[1] Korteweg, D. and de Vries, G., Philosophical Magazine **39** (1895) 422.

[2] Boussinesq, J. M., Mémoires présentés par divers savants a l'académie des sciences de l'insitut de France **T23** (1877).

[3] Zabusky, N. J. and Kruskal, M. D., Phys. Rev. Lett. **15** (1965) 240.

[4] Burden, R. L. and Faires, J. D., *Numerical Analysis*, PWS-KENT Publishing Company, 4 edition, 1988.

[5] R.M. Miura, C. S. G. and Kruskal, M. D., Journal of Mathematical Physics **9** (1968) 1204.

[6] Carpenter, M. H. and Kennedy, C. A., NASA Technical Memorandum (1994).

[7] Williamson, J. H., Journal of Computational Physics (1980) 48.

[8] Python programming language, http://www.python.org, 2008.

[9] Numpy, http://numpy.scipy.org, 2008.

[10] Psyco, http://psyco.sourceforge.net/introduction.html, 2008.

[11] Rigo, A., Representation-based just-in-time specialization and the psyco prototype for python, in *PEPM*, pages 15–26, 2004.

A Mallinnusympäristö

Simulaatioissa käytettiin tarkoitukseen kirjoitettua Python-ohjelmaa [8]. Tulkattavana kielenä Pythonin nopeus ei ole paras mahdollinen tämän kaltaiseen mallintamiseen. Kieli on valittu sen yksinkertaisuuden ja valmiiden kirjastojen laajuuden vuoksi.

Pythonille on tehty numeerista mallinnusta varten kirjasto NumPy [9], josta tässä työssä käytettiin matemaattisia funktioita, numarray-objektia ja muutamassa simulaatiossa mielivaltaisen tarkkoja desimaalilukuja. NumPy:n numarray-objekti toimii monilta tavoin kuten Pythonin list-tietorakenne tai monien muiden ohjelmointikielien taulukko tai vektori. Etuna numarray-objektissa on sen helpompi käyttö laskutoimituksissa. Tavalliselle list-tietorakenteelle pitää esimerkiksi listoja summatessa yhteen tehdä toistorakenne, joka summaa listat alkioittain. Numarray-objektissa riittää kirjoittaa pelkkä yhteenlaskutoimitus samoin kuin esimerkiksi kokonaisluvuilla laskiessa. Tässä sovelluksessa tämä on toivottavampaa käytöstä ja näin säästytään useilta toistorakenteiden kirjoittamisilta.

Nopeutta parantamaan käytettiin Psyco-kirjastoa [10], joka parhaimmillaan nopeuttaa Pythonin suoritusta jopa 100-kertaiseksi [11]. Näin suurta hyötyä tämän ohjelman kohdalla ei saavuteta, mutta Psycon vaikutus on selvästi havaittavissa. Nopeutta voidaan parantaa enemmänkin muilla Pythonille tehdyillä kirjastoilla, mutta niistä monet vaativat muutoksia itse ohjelmakoodiin ja näin kielen valinnalla saavutettu ohjelmoimisen helppous menetetään.

Ohjelmassa tila u on tallennettu määrätyn pituiseen numarray-objektiin. Objektia voidaan ulkoa päin käsitellä kuten Pythonin list-tietorakennetta tai monista kielistä tuttua taulukkoa. Jatkossa objektiin viitataankin taulukkona. Taulukko tulkitaan hilaksi, jossa taulukon indeksi viittaa paikkakoordinaattiin x ja taulukon solun sisältö arvoon $u(x)$. Ohjelma pitää muistissa myös u_{t-1}:n kahden aika-askeleen menetelmiä varten.

Alkuarvoina ohjelmalle annetaan käytettävän hilan pituus `l`, hilapisteiden välimatka `h`, aika-askeleen alkuarvo `g_0`, solitonin nopeus `v_1` ja sallittu ajan maksimiarvo `t_max`. Halutessa voidaan antaa myös `v_2` ja `v_3` toisen ja kolmannen solitonin nopeudeksi. Jos näitä ei syötetä, mallintaa ohjelma vain yhtä aaltoa.

Simulaation aikana lasketaan hilapisteiden sen hetkisistä arvoista u KdV:n säilyvien suureiden arvot. Alkuarvoissa voidaan muuttujalla `m` määrittää, miten monen iteraation välein arvot kirjataan levylle ja samalla säilyvät suureet tarkastetaan.

Keskeisimmät osat simulaatio-ohjelman lähdekoodeista ovat nähtävissä liitteessä B. Liite ei sisällä käyttöliittymän lähdekoodeja eikä käytettyä pääohjelmaa vaan pelkät matematiikkaosuudet ja osan tulosten kirjaamisesta. Mukaan on kirjoitettu yksinkertainen pääohjelma, jolla simulaatioita voidaan ajaa. Huomattavaa on lisäksi, että lähdekoodi on kirjoitettu Pythonin versiolle 2.5, ja siinä käytetään monia rakenteita, mitä versio 3.0 ja uudemmat eivät tue.

B Lähdekoodit

KdV.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import solitoni, os, sys, time, data
import psyco     # Kommentoi nämä pois
psyco.full()     # jos et halua käyttää psycoa

l    = 20        # Hilan pituus
h    = 0.1       # Paikka-askel
g   = 0.0007703  # Alustetaan aika-askel
v1   = 5.0       # Solitonin 1 nopeus
v2   = None      # Solitonin 2 nopeus, jos 0, niin ei solitonia
v3   = None      # Solitonin 3 nopeus, jos 0, niin ei solitonia
tmax = 300       # Iterointiin käytettävä maksimiaika
m    = 300       # Kirjattavien mittapisteiden välimatka aika-askeleina

funk  = 7        # Määritetään käytettävä diskretointi (1-7)
itera = 3        # Määritetään käytettävä aikaintegrointi (1-6)

t    = 0         # Alustetaan kello

# Alustetaan solitoniolio
u  = solitoni.solitoni(funk, itera, h, g, l, v1 = v1, v2 = v2, v3 = v3)

# Määritetään hakemisto tulostettavalle datalle
hak = "/home/mhjlai/kdvdata/"

data   = data.Kirjaus()
data.alustus(hak, tmax, l, sol = u)

print 'Tulostetaan tulokset tiedostoon ' + data.tulokset.f.name

M0 = u.momentti()
E0 = u.energia()
K0 = u.kolmas()
if K0 < 0:
    K0 = -K0
M = M0
E = E0
K = K0

data.uusi("data")
data.kirjaa(u.getU(), (M0, E0, K0), u.t, h, False)
loppuaika = None

# I'm in ur loop uppin ur t to tmax
while u.t < tmax:
    if int(u.t/g)%m == 0:
        M      = u.momentti()
        E      = u.energia()
        K      = u.kolmas()
        if K < 0:
            K = -K
        if ( (str(M) == 'nan') | (str(E) == 'nan') | (str(K) == 'nan')  | (M < 0.8*M0) | \
             (M > 1.2*M0) | (E < 0.8*E0) | (E > 1.2*E0) | (K < 0.8*K0) | (K > 1.2*K0)):
            loppuaika = (time.time() - aika)/u.t
            data.kirjaa(u.getU(), (M0, E0, K0), u.t, h, tulokset = True, aika = loppuaika)
            break
        data.kirjaa(u.getU(), (M, E, K), u.t, h)
    u.iteroi()

if loppuaika == None:
    loppuaika = (time.time() - aika)/u.t

print str(g)+ ' ' + str(u.t) + ' ' + str(loppuaika)
if (u.t >= tmax):
    data.kirjaa(u.getU(), (M, E, K), u.t, h, tulokset = True, aika = loppuaika)

print "Valmis"
```

solitoni.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

from numpy import *
import math

class solitoni:
    """
    Luokka mallintaa Korteveg-de Vries-yhtälön mukaista solitoniaaltoa. Luokkaan on tallennettu
    hilapisteiden arvot kahdella eri ajan hetkellä muuttujiin listoihin self.u (nykyinen) ja
    self.ued (edellinen).
    """

    def __init__(self, funk, iter, h, g, l, v1 = None, v2 = None, v3 = None):
        # Alkuarvot
        self.t   = 0.0
        self.h   = h                # paikka-askel
        self.g   = g                # aika-askel
        self.l   = l                # kokonaispituus
        self.L   = int(l/h)         # hilapisteiden määrä

        try:
            self.fu = {
             1: naiviepasym(L = self.L, h = self.h),
             2: naivisym(   L = self.L, h = self.h),
             3: keskiarv1(  L = self.L, h = self.h),
             4: keskiarv2(  L = self.L, h = self.h),
             5: keskiarv3(  L = self.L, h = self.h),
             6: keskiarv4(  L = self.L, h = self.h),
             7: zk(         L = self.L, h = self.h)}[funk]
        except KeyError:
            print "Funktiota " + str(funk) + " ei ole toteutettu!"
            print "Valitse jokin toteutettu funktio."
            print "Oletusarvoisesti valitaan funktio 3."
            self.fu = zk
        self.fid = self.fu.nimi
        self.f   = self.fu.iter

        try:
            self.iter = {
             1: self.euler,
             2: self.rk,
             3: self.lsrunge_four,
             4: self.euler_back,
             5: self.euler2,
             6: self.adams_mb_four}[iter]
            self.iid = {
             1: 'Euler',
             2: 'Runge-Kutta',
             3: 'Low Storage Runge-Kutta',
             4: 'Backward Euler',
             5: 'Euler 2',
             6: 'Adams-Moulton-Bashforth Four-step'}[iter]
        except KeyError:
            print "Iterointia " + str(iter) + " ei ole toteutettu!"
            print "Valitse jokin toteutettu iterointimenetelmä."
            print "Oletusarvoisesti valitaan menetelmä 2 (Runge-Kutta)."
            self.iter = self.rk

        self.v1 = v1
        # Jos v3 on annettu, mutta v2:ta ei, vaihdetaan ne päikseen
        if (v2 == None) & (v3 != None):
            temp = v2
            v2 = v3
            v3 = temp
        self.v2 = v2
        self.v3 = v3

        # Yhden solitonin alkutilanne
        if (self.v2 == None) & (self.v3 == None):
            self.x1   = l/2.0

        # Kahden solitonin alkutilanne
        if (self.v2 != None) & (self.v3 == None):
```

```
            self.x1 = l/3
            self.x2 = 2*l/3

        # Kolmen solitonin alkutilanne
        if (self.v2 != None) & (self.v3 != None):
            self.x1 = l/5
            self.x2 = l/2
            self.x3 = 4*l/5

        if iter < 6:
            self.luo_normaali()

        if iter == 6:
            self.luo_4()

    def luo_normaali(self):
        """ Luodaan yhden vanhan arvon sisältävä tila 1-3:lle aallolle """

        self.u    = self.analyyttinen(self.h, self.v1, self.l, self.x1, 0.0)
        self.ued  = self.analyyttinen(self.h, self.v1, self.l, self.x1, -self.g)

        if self.v2 != None:
            self.u    += self.analyyttinen(self.h, self.v2, self.l, self.x2, 0.0)
            self.ued  += self.analyyttinen(self.h, self.v2, self.l, self.x2, -self.g)

        if self.v3 != None:
            self.u    += self.analyyttinen(self.h, self.v3, self.l, self.x3, 0.0)
            self.ued  += self.analyyttinen(self.h, self.v3, self.l, self.x3, -self.g)

    def luo_4(self):
        """ Luodaan 4 vanhaa arvoa sisältävä tila 1-3:lle aallolle """
        self.luo_normaali()

        self.ued2  = self.analyyttinen(self.h, self.v1, self.l, self.x1, -2.0*self.g)
        self.ued3  = self.analyyttinen(self.h, self.v1, self.l, self.x1, -3.0*self.g)
        self.ued4  = self.analyyttinen(self.h, self.v1, self.l, self.x1, -4.0*self.g)

        if self.v2 != None:
            self.ued2  += self.analyyttinen(self.h, self.v2, self.l, self.x2, -2.0*self.g)
            self.ued3  += self.analyyttinen(self.h, self.v2, self.l, self.x2, -3.0*self.g)
            self.ued4  += self.analyyttinen(self.h, self.v2, self.l, self.x2, -4.0*self.g)
        if self.v3 != None:
            self.ued2  += self.analyyttinen(self.h, self.v3, self.l, self.x3, -2.0*self.g)
            self.ued3  += self.analyyttinen(self.h, self.v3, self.l, self.x3, -3.0*self.g)
            self.ued4  += self.analyyttinen(self.h, self.v3, self.l, self.x3, -4.0*self.g)
        self.uder  = self.f(self.u)
        self.uder1 = self.f(self.ued)
        self.uder2 = self.f(self.ued2)
        self.uder3 = self.f(self.ued3)
        self.uder4 = self.f(self.ued4)

    def iteroi(self):
        self.iter()

    def euler(self):
        """ Eulerin menetelmä """
        self.ued = self.u.copy()
        self.u += self.g*self.f(self.u)
        self.t += self.g

    def euler2(self):
        """ Two Step Euler """
        temp     = self.u.copy()
        self.u   = self.ued + 2.0*self.g*self.f(self.u)
        self.ued = temp.copy()
        self.t  += self.g

    def euler_back(self):
        """ Backward Euler """
        self.ued = self.u.copy()
        self.u   = self.u + self.g*self.f(self.u + self.g*self.f(self.u))
        self.t  += self.g

    def rk(self):
        """ Runge-Kutta-menetelmä """
```

```
    temp = self.u.copy()
    k1 = self.g*self.f(self.u)
    k2 = self.g*self.f(self.u + 0.5*k1)
    k3 = self.g*self.f(self.u + 0.5*k2)
    k4 = self.g*self.f(self.u + k3)
    self.u += (k1/6. + k2/3. + k3/3. + k4/6.)

    self.ued = temp.copy()
    self.t += self.g

def adams_mb_four(self):
    """ Adams Bashforth Four-step """

    temp = self.u.copy()
    self.uder = self.f(self.u)

    temp2    = self.u + self.g*(55.*self.uder - 59.*self.uder1 + 37.*self.uder2 - 9.*self.uder3)/24
    self.u  += self.g*(9*self.f(temp2) + 19*self.uder - 5*self.uder1 + self.uder2)/24
    self.t += self.g
    self.uder3 = self.uder2.copy()
    self.uder2 = self.uder1.copy()
    self.uder1 = self.uder.copy()

    self.ued = temp

def lsrunge_four(self):
    """ Fourth order five stage Low Storage Runge-Kutta """

    a = (0.0, -0.481231743137, -1.04956260671, -1.60252957428, -1.77826719392)
    b = (0.0976183546921, 0.412253292916, 0.440216963931, 1.42631146322, 0.197876053732)

    self.ued = self.u.copy() # päivitetään edellinen, koska muutokset tehdään suoraan nykyiseen arvoon
    du = zeros(self.L)       # varataan muistipaikka listalle du
    for i in range(0, 5):    # itse algoritmi
        du = a[i]*du + self.g*self.f(self.u)
        self.u  = self.u + b[i]*du
    self.t += self.g

def analyyttinen(self, h, v, l, x, t):
    """ Solitoniyhtälön analyyttinen ratkaisu """

    if v < 0: absv = -v
    else: absv = v
    tulos = zeros(self.L)
    for i in range(0, self.L):
        tulos[i] = (absv/2)/(cosh(sqrt(absv)/2*((h*i)- (v*t)%l - x))**2) \
             + (absv/2)/(cosh(sqrt(absv)/2*((h*i)- (v*t)%l - x + l))**2)
    return tulos

def getU(self):
    """ Palauttaa u:n """
    return self.u.copy()

def momentti(self):
    """ Laskee solitonin ensimmäisen säilyvän suureen. """
    try:
        M = self.u.sum()
        return M
    except OverflowError:
        print "fail"
        return False

def energia(self):
    """ Laskee solitonin toisen säilyvän suureen. SUM(0.5*u(i)^2) """
    try:
        E = 0
        for i in range(self.L):
            E = E + pow(self.u[i], 2)
        return E
    except OverflowError:
        print "fail"
        return False
def kolmas(self):
    """ Laskee solitonin kolmannen säilyvän suureen. SUM(2*u(i)^3 - (Du(i))^2) """
```

```
        try:
            kolmas = 0
            for i in range(self.L):
                kolmas = kolmas + (2*math.pow(self.u[i],3) - math.pow(((self.u[i]-self.ued[i])/self.g),2))
            return kolmas
        except OverflowError:
            print "fail"
            return False

class naiviepasym:
    nimi   = "Naivi epäsymmetrinen diskretointi"
    lyh    = "NEsym"
    def __init__(self, L = 15, h = 0.1):
        self.nimi   = "Naivi epäsymmetrinen diskretointi"
        self.lyh    = "NEsym"
        self.L = L
        self.h = h
    def iter(self, u):
        temp = zeros(self.L)
        for i in range(0, self.L):
            u0 = u[ i     ]
            u1 = u[(i+1)%self.L]
            u2 = u[(i+2)%self.L]
            u3 = u[(i+3)%self.L]
            temp[i] = -(u3 - 3*u2 + 3*u1 - u0)/(self.h**3) - 6*u0*(u1 - u0)/(self.h)
        return temp

class naivisym:
    nimi   = "Naivi symmetrinen diskretointi"
    lyh    = "Nsym"
    def __init__(self, L = 15, h = 0.1):
        self.nimi   = "Naivi symmetrinen diskretointi"
        self.lyh    = "Nsym"
        self.L = L
        self.h = h
    def iter(self, u):
        temp = zeros(self.L)
        for i in range(0, self.L):
            u_3 = u[(i-3)%self.L]
            u_1 = u[(i-1)%self.L]
            u0  = u[ i     ]
            u1  = u[(i+1)%self.L]
            u3  = u[(i+3)%self.L]
            temp[i] = -(u3 - 3*u1 + 3*u_1 - u_3 )/(8*self.h**3) - 6*u0*(u1 - u_1)/(2*self.h)
        return temp

class keskiarv1:
    nimi   = "Naivi symmetrinen diskretointi keskiarvolla 1"
    lyh    = "KA1"
    def __init__(self, L = 15, h = 0.1):
        self.nimi   = "Naivi symmetrinen diskretointi keskiarvolla 1"
        self.lyh    = "KA1"
        self.L = L
        self.h = h
    def iter(self, u):
        temp = zeros(self.L)
        for i in range(0, self.L):
            u_3 = u[(i-3)%self.L]
            u_1 = u[(i-1)%self.L]
            u0  = u[ i     ]
            u1  = u[(i+1)%self.L]
            u3  = u[(i+3)%self.L]
            temp[i] = -(u3 - 3*u1 + 3*u_1 - u_3 )/(8*self.h**3) - 3*(u1 + u_1)*(u1 - u_1)/(2*self.h)
        return temp

class keskiarv2:
    nimi   = "Naivi symmetrinen diskretointi keskiarvolla 2"
    lyh    = "KA2"
    def __init__(self, L = 15, h = 0.1):
        self.nimi   = "Naivi symmetrinen diskretointi keskiarvolla 2"
        self.lyh    = "KA2"
        self.L = L
        self.h = h
    def iter(self, u):
        temp = zeros(self.L)
```

```
        for i in range(0, self.L):
            u_3 = u[(i-3)%self.L]
            u_1 = u[(i-1)%self.L]
            u0  = u[ i     ]
            u1  = u[(i+1)%self.L]
            u3  = u[(i+3)%self.L]
            temp[i] = -(u3 - 3*u1 + 3*u_1 - u_3 )/(8*self.h**3) - (u1 + u0 + u_1)*(u1 - u_1)/(self.h)
        return temp

class keskiarv3:
    nimi   = "Naivi symmetrinen diskretointi keskiarvolla 3"
    lyh    = "KA3"
    def __init__(self, L = 15, h = 0.1):
        self.nimi   = "Naivi symmetrinen diskretointi keskiarvolla 3"
        self.lyh    = "KA3"
        self.L = L
        self.h = h
    def iter(self, u):
        temp = zeros(self.L)
        for i in range(0, self.L):
            u_3 = u[(i-3)%self.L]
            u_2 = u[(i-2)%self.L]
            u_1 = u[(i-1)%self.L]
            u0  = u[ i     ]
            u1  = u[(i+1)%self.L]
            u2  = u[(i+2)%self.L]
            u3  = u[(i+3)%self.L]
            temp[i] = -(u3 - 3*u1 + 3*u_1 - u_3 )/(8*self.h**3) - 3*(u2 + u1 + u0 + u_1 + u_2)*(u1 - u_1)/(5*self.h)
        return temp

class keskiarv4:
    nimi   = "Naivi symmetrinen diskretointi keskiarvolla 4"
    lyh    = "KA4"
    def __init__(self, L = 15, h = 0.1):
        self.nimi   = "Naivi symmetrinen diskretointi keskiarvolla 4"
        self.lyh    = "KA4"
        self.L = L
        self.h = h
    def iter(self, u):
        temp = zeros(self.L)
        for i in range(0, self.L):
            u_3 = u[(i-3)%self.L]
            u_1 = u[(i-1)%self.L]
            u0  = u[ i     ]
            u1  = u[(i+1)%self.L]
            u3  = u[(i+3)%self.L]
            temp[i] = -(u3 - 3*u1 + 3*u_1 - u_3 )/(8*self.h**3) - 3*(u3 + u1 + u0 + u_1 + u_3)*(u1 - u_1)/(5*self.h)
        return temp

class zk:
    nimi   = "Zabuskyn ja Kruskalin diskretointi"
    lyh    = "ZK"
    def __init__(self, L = 15, h = 0.1):
        self.nimi   = "Zabuskyn ja Kruskalin diskretointi"
        self.lyh    = "ZK"
        self.L = L
        self.h = h
    def iter(self, u):
        temp = zeros(self.L)
        for i in range(0, self.L):
            u_2 = u[(i-2)%self.L]
            u_1 = u[(i-1)%self.L]
            u0  = u[ i     ]
            u1  = u[(i+1)%self.L]
            u2  = u[(i+2)%self.L]
            temp[i] = -(u2 - 2*u1 + 2*u_1 - u_2)/(2*self.h**3) - (u1 + u0 + u_1)*(u1 - u_1)/(self.h)
        return temp
```

data.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import datetime, os
```

```
class Kirjaus:
    """ Tulostusvirran wrapperiluokka. """

    def __init__(self):
        self.ts        = self.timestamp()
        self.alustettu = False
        self.h         = ''
        self.tulokset  = None
        self.data      = None

    def alustus(self, root, tmax, l, sol = None):
        """ Tämä hoitaa oikeasti alustuksen """

        if self.alustettu: pass
        else:
            self.h = root + self.ts + "/"

            # Testataan onko hakemisto olemassa ja jos ei niin tehdään.
            os.system("if [ ! -d " + self.h +" ]; then mkdir -p " + self.h + "; fi")

            self.tulokset  = Tulokset(hak = self.h, solitoni = sol)
            self.data      = Data(self.h, tmax, l, solitoni = sol)
            self.alustettu = True

    def timestamp(self):
        """ Palauttaa aikaleiman. """
        d         = datetime.datetime.now().timetuple()
        timestamp = "%d-%d-%d-%.2d%.2d%.2d" % (d[0],d[1],d[2],d[3],d[4],d[5])
        return timestamp

    def kirjaa(self, u, S, t, h, tulokset = False, aika = None):
        """ Hoidetaan kirjaus. """
        if tulokset:
            self.tulokset.kirjaa(aika = aika)
        self.data.kirjaa(u, t, h, S)

    def uusi(self, prefix, param = None):
        try:
            if self.data.f.closed == False:
                self.data.suljeTiedosto()
            self.data.avaaTiedosto(prefix, param)
        except TiedostoAukiVirhe:
            self.data.suljeTiedosto()
        except EiTiedostoaVirhe:
            self.data.avaaTiedosto(prefix, param)
        except AttributeError:
            self.data.f = open("temp" + self.ts, 'w')
            self.data.f.close()
            os.system("rm temp" + self.ts)
            self.data.avaaTiedosto("a", 0)
        else:
            if self.data.f.closed:
                print "Fail\nTiedoston sulkeminen tai uuden avaaminen ei onnistu."

    def setSolitoni(self, solitoni = None):
        self.data.setSolitoni(solitoni = solitoni)
        self.tulokset.setSolitoni(solitoni = solitoni)


class Tulokset:
    """ Tulosten kirjauksen hoitava luokka """

    def __init__(self, hak = 'temp', solitoni = None):
        self.tiedosto = hak + "tulokset.data"
        self.f        = open(self.tiedosto, 'w')
        self.solitoni = solitoni
        self.alkujutut()

    def setSolitoni(self, solitoni = None):
        self.solitoni = solitoni

    def alkujutut(self):
        self.f.write("\n# Tiedosto " + self.tiedosto)
        self.f.write("\n# Käytetään funktiota "         + str(self.solitoni.fu.nimi))
```

```
        self.f.write("\n# Käytetään aikaiteraatiota "   + str(self.solitoni.iid))
        self.f.write("\n# Solitonin 1 nopeus v1     = " + str(self.solitoni.v1))
        self.f.write("\n# Solitonin 2 nopeus v2     = " + str(self.solitoni.v2))
        self.f.write("\n# Solitonin 3 nopeus v3     = " + str(self.solitoni.v3))
        self.f.write("\n# Käytettävä paikka-askel h = " + str(self.solitoni.h))
        self.f.write("\n")
        self.f.write("\n# Aika-askel\t Aika\n")

    def kirjaa(self, aika = None):
        self.f.write("%.20f\t %f\t %f\n" % (self.solitoni.g, self.solitoni.t, aika))

    def sulje(self):
        self.f.close()

class Data:
    """ Raakadatan kirjauksen hoitava luokka """

    def __init__(self, h, tmax, l, solitoni = None):
        self.h         = h
        self.tmax      = tmax
        self.l         = l
        self.solitoni  = solitoni
        self.f = open("temp", 'w')
        self.f.close()
        os.system("rm temp")

    def setSolitoni(self, solitoni = None):
        self.solitoni = solitoni

    def avaaTiedosto(self, prefix, param = None):
        """ Avaa uuden tiedoston. """
        if param == None:
            tiedosto = self.h + prefix + "_" + str(self.solitoni.g)
        else:
            tiedosto = self.h + prefix + "-" + str(param) + "_" + str(self.solitoni.g)
        if self.f.closed:
            self.f = open(tiedosto, 'w')
        else:
            raise TiedostoAukiVirhe(tiedosto)

    def suljeTiedosto(self):
        """ Sulkee avoinna olevan tiedoston, jos on auki. """
        if self.f.closed == False:
            self.f.flush()
            self.f.close()
        else:
            raise EiTiedostoaVirhe()

    def kirjaa(self, u, t, h, S, muisti = None):
        """ Kirjaa tiedostoon annetut argumentit """
        self.f.write("%.10f %f\t %f\t %f\t %f\t %f\n" % (t, 0.0, u[0], S[0], S[1], S[2]) )
        for i in range(1, len(u)):
            self.f.write("%.10f %f\t %f\n" % (t, (h*i), u[i]) )
        self.f.write("\n")

    def getFile(self):
        """ Palauttaa kirjaustiedoston (polkuineen) """
        return self.f.name

class TiedostoAukiVirhe(Exception):
    def __init__(self, value):
        self.value = value
    def __str__(self):
        return repr(self.value)

class EiTiedostoaVirhe(Exception):
    def __init__(self, value):
        self.value = value
    def __str__(self):
        return repr(self.value)
```