Pseudospectrale methode bij een golf-simulatie

Arthur van Dam

Februari 2001

1 Inleiding

In dit artikel zullen watergolven en hun reflectie aan vaste wanden bestudeerd worden en de effecten van waterdiepten en hellende bodem hierop. Gekeken wordt naar de oppervlakte-uitwijking, voortplantingssnelheid en dieptegemiddelde snelheid.

De differentiaalvergelijkingen (1) en (2) die het systeem beschrijven, worden opgelost met de pseudospectrale methode.

2 Probleemstelling

De voortplanting en reflectie aan vaste wanden van een watergolf worden beschreven door de continuïteitsvergelijking en de impulsvergelijking:

$$\eta_t + \left((h+\eta)\,u\right)_x = 0, \tag{1}$$

$$u_t + uu_x = -g\eta_x. \tag{2}$$

Hierin is η de oppervlakte-uitwijking van het water en u is de dieptegemiddelde snelheid; beide worden bekeken op het domein $[0, x_{\text{tot}}]$. $g = 9.81$ is de zwaartekrachtsversnelling, en de waterdiepte h wordt beschreven door

$$h = h_0 - h_1 \cdot \frac{x}{x_{\text{tot}}}. \tag{3}$$

Begin- en randvoorwaarden

Op $t = 0$ wordt de oplossing beschreven door:

$$\begin{aligned} \eta(x,0) &= 0.1e^{-(x-x_{\text{mid}})^2/dx^2}\sin(-kx), \\ u(x,0) &= g \cdot k \cdot \eta(x,0)/\omega, \end{aligned}$$

waarin $x_{\text{mid}} = x_{\text{tot}}/2$, $dx^2 = 0.2 \cdot x_{\text{mid}}^2$, $k = 2\pi/x_{\text{mid}}$ en $\omega = k\sqrt{gh_0}$.

De beginvoorwaarden worden gezet in de routine `initT0` in appendix A.3.

Het domein krijgt breedte $x_{\text{tot}} = 2$km, met aan beide zijden een reflecterende rand. De snelheid loodrecht op de wand is natuurlijk nul:

$$u_0 = u_{x_{\text{tot}}} = 0$$

Aangezien u op de rand constant gelijk is aan 0, geldt $u_t = 0$, dit samen ingevuld in (2) levert een randvoorwaarde voor η:

$$\eta_x = 0 \quad \text{(op de rand)}.$$

Hiermee wordt (1) op de rand een stukje eenvoudiger.

3 Oplosmethode

Zoals gezegd, zal de pseudospectrale methode worden gebruikt om (1) en (2) op te lossen. Hierbij worden de plaats-afgeleiden in een spectrale ruimte bepaald, door de oplossing in termen van Chebychev-polynomen te schrijven. De tijdsintegratie gebeurt vervolgens met een tweede-orde eindige differentie methode. Het hoofdprogramma en IO wordt geregeld in de routine `main` in appendix A.1.

3.1 Chebychev-polynomen

Gegeven een functie u, kunnen uit $N+1$ samples van deze functie op de collocatiepunten $\tilde{x}_k$ de N Chebychev-coëfficiënten worden bepaald volgens:

$$c(j) = \frac{2-\delta_{0j}}{N} \sum_{k=0}^{N} \alpha_k \eta\left(\tilde{x}_k, t\right) \cos\left(\frac{\pi j k}{N}\right) \quad \text{voor} \quad j = 0, \ldots, N-1, \tag{4}$$

met $\alpha_0 = \alpha_N = 0.5$ en de andere $\alpha_k = 1$. De collocatiepunten $\tilde{x}_k$ ontstaan uit de genormaliseerde collocatiepunten x_k:

$$x_k = \cos\left(\frac{\pi k}{N}\right) \quad \text{voor} \quad k = 0, \ldots, N \quad \text{Hier}: \tilde{x}_k = \frac{x_{\text{tot}}}{2} \cdot (x_k + 1). \tag{5}$$

Vergelijking (4) staat in de C-routine `chebyFwd` in appendix A.2.

Gegeven een vector met Chebychev-coëfficiënten is het natuurlijk wenselijk om de waarde van $u(x)$ te bepalen. Eén van de voordelen van de methode is dat u op ieder punt x geëvalueerd kan worden, zelfs als de coëfficiënten op discrete samples zijn gebaseerd. Deze terugtranformatie van spectraal naar fysisch domein is vrij recht-toe-recht-aan en is te vinden in de routine `chebyBwd` in appendix A.2.

Tot slot kunnen uit de coëfficiënten c van een functie u ook de coëfficiënten $c1$ van de ruimtelijke afgeleide u_x bepaald worden. De eenvoudige rekenregels hiervoor staan in routine `chebyDer1` in appendix A.2. Waar goed op moet worden gelet is, zodra de waarde van de ruimtelijke afgeleiden op het fysische domein wordt bepaald uit de $c1$, deze waarde nog herschaald moet worden. De afgeleide wordt namelijk bepaald op het domein $[-1, 1]$. De werkelijke afgeleide ontstaat door dit met $2/x_{\text{tot}}$ te vermenigvuldigen.

Nauwkeurigheidstest Om een korte blik op de nauwkeurigheid van benadering met Chebychev-polynomen te werpen, wordt een eenvoudige testfunctie:

$$u(x) = 4x^3 + 4x^2 - 3x - 1$$

op het domein $[-1, 1]$ heen en teruggetransformeerd.

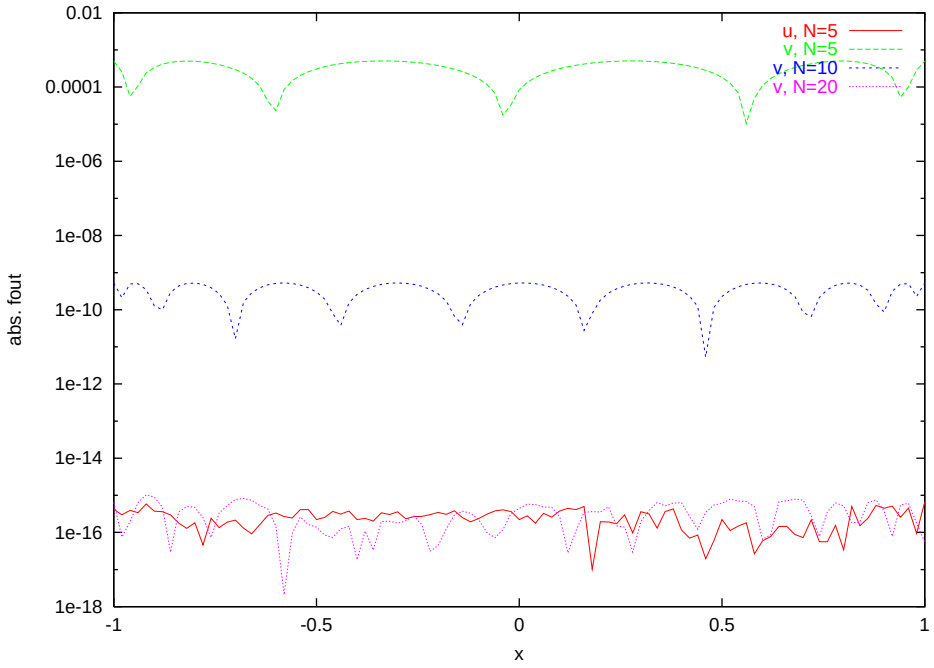
Allereerst is bekend dat, gebruikmakend van:

$$\begin{aligned} T_0(x) &= 1, \\ T_1(x) &= x, \\ T_2(x) &= 2x^2 - 1, \\ T_3(x) &= 4x^3 - 3x, \end{aligned}$$

de Chebychev-coëfficiënten gelijk zijn aan: $c_0 = 1, c_2 = 2, c_3 = 1$ en de rest is 0. Aangezien deze testfunctie 'mooi is gekozen', zal de nauwkeurigheid wel vrij hoog zijn. Ter vergelijking wordt ook nog een wat willekeurigere functie genomen:

$$v(x) = \sin(x) + \cos(x).$$

In figuur 1 staan de resultaten voor beide functies voor verschillende N. De foutcurve voor u



Figuur 1: De absolute fout van functies u en v na heen-en-terug Chebychev transformatie, voor verschillende N.

is alleen voor $N = 5$ getoond: zelfs daar is de fout al ter grootte van de machineprecisie. De resultaten bij v zijn een stuk interessanter: een verdubbeling van N lijkt de fout een factor 10^{-6} te verbeteren.

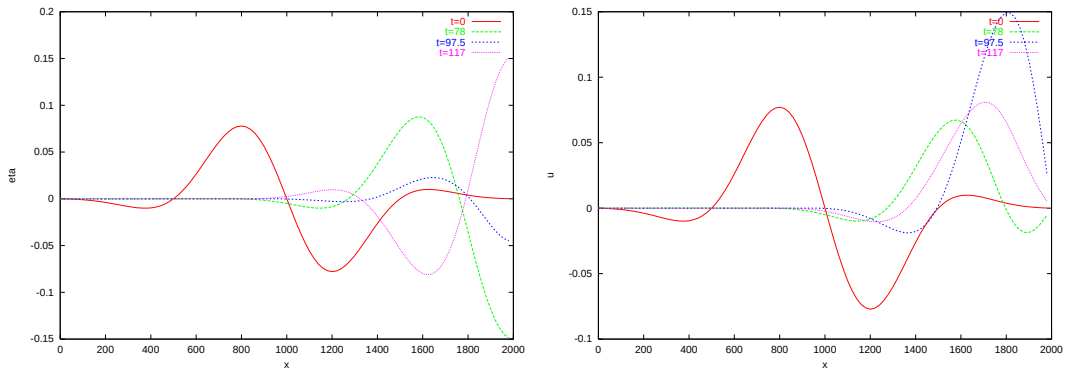
De nauwkeurigheid van Chebychev zal echter vooral onderzocht moeten worden in de problemen waar het wordt toegepast; de opgelegde randvoorwaarden en ruimtestapgrootte gaan dan ook sterk de nauwkeurigheid van de oplossing beïnvloeden. De testresultaten van zonet en de theoretische nauwkeurigheid van $\mathcal{O}(\Delta x^N)$ geven echter voldoende vertrouwen om deze methode op het waterreflectieprobleem te gaan toepassen.

3.2 Tijdsintegratie

Voor de tijdsintegratie wordt de tweede-orde methode Adams-Bashfort gebruikt, met een Euler-beginstap op $t = 0$.
De tijdsafgeleiden staan geformuleerd in niet-lineaire termen en ruimtelijke afgeleiden, die op het huidige en vorige tijdstip bekend zijn. Het pseudospectrale algoritme wordt hiermee:

1. Bepaal de Chebychev-coëfficiënten c van de oplossingen op het huidige tijdstip.
2. Bepaal uit c de coëfficiënten $c1$ van de ruimteafgeleiden. Bepaal hieruit de afgeleiden op het fysische domein (denk aan de domeinherschaling!)
3. Evalueer de niet-lineaire termen in de fysische ruimte.
4. Bepaal de oplossing op het nieuwe tijdstip met

$$u^{n+1} = u^n + \Delta t \left(\frac{3}{2} u_t^n - \frac{1}{2} u_t^{n-1} \right)$$



Figuur 2: De oplossingen op vier verschillende tijdstippen, gedurende de botsing tegen de rechterwand bij $h_0 = 10$ en $h_1 = 0$, $N = 25$ en $\Delta t = 0.1$. Links de oplossingen voor η, rechts de oplossingen voor u.

Het Adams-Bashfort algoritme staat in de routine `abLoop` in appendix A.3; hier staat ook de Eulerstap `eulerStep`. De tijdsafgeleiden van η en u worden bepaald in `f\_eta` respectievelijk `f\_u`.

4 Experimenten

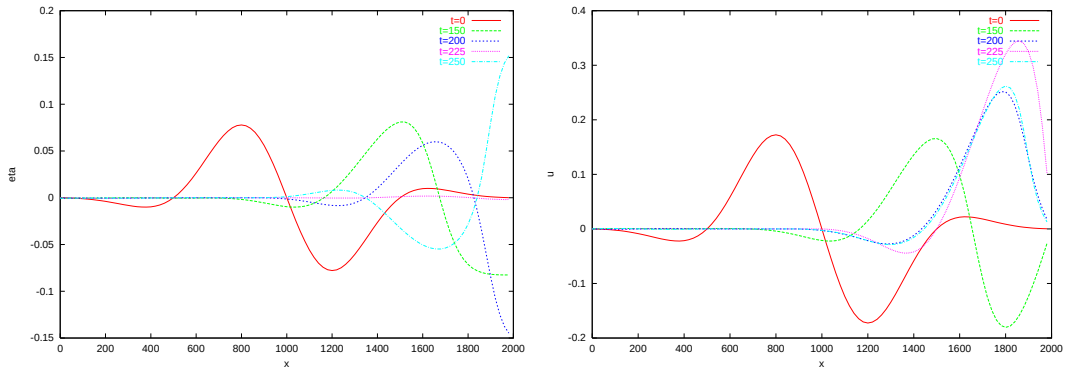
Een eerste simulatie zal worden gedaan met waterdiepte $h_0 = 10$ m, vlakke bodem ($h_1 = 0$) voor een periode van 400 s. In de discretisatie wordt $N = 25$ en $\Delta t = 0.1$ gekozen. Dit voldoet niet aan de theoretische stabiliteitseis

$$\Delta t \leq \frac{1}{cN^2},$$

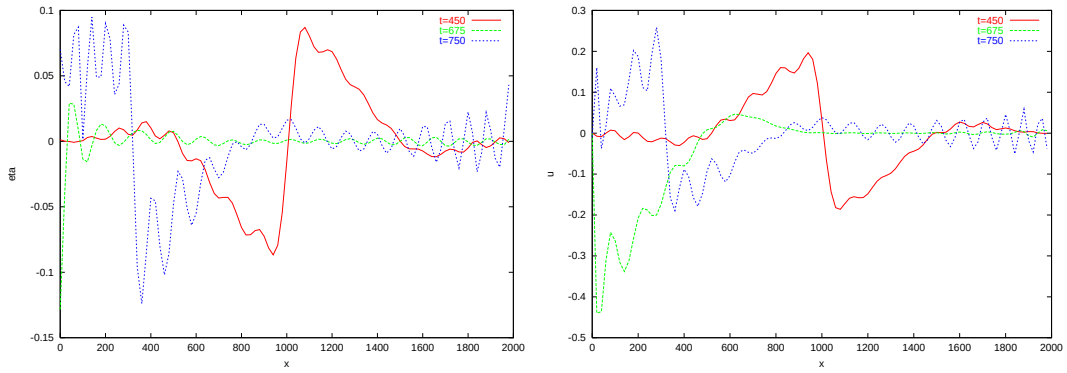
waarin c de advectiesnelheid is. De resultaten laten zien dat de bepaalde oplossing nog best aardig is. In figuur 2 staan voor u en η de oplossingen op vier tijdstippen, min of meer tijdens de botsing tegen de rechterrand. De golf beweegt eerst onveranderd naar rechts; tijdens de botsing gaan de terugkerende en nog aankomende golf interfereren, de som-amplitude is een stuk groter, bijvoorbeeld op $t = 78$ en $t = 117$.
De dieptegemiddeldesnelheid is tussen deze twee tijdstippen het grootst en positief. Dit komt doordat de hele grote (negatieve!) oppervlakteuitwijking omdraait in een positieve grote uitwijking. Hiervoor is veel water nodig en dat wordt dus snel naar rechts getransporteerd.

Ondiepe bak Vervolgens wordt een ondiepe bak van 2 m genomen, en de oplossing bekeken over een tijd van 900 s. In figuur 3 staat de oplossing tijdens de botsing met de rechterrand weergegeven. Wanneer de oppervlakte-uitwijking met de 10 m-bak wordt vergeleken (linker diagrammen in fig. 2 en 3), dan is er weinig veranderd: de amplitude is min of meer gelijk en de terugkaatsing wijkt ook weinig af. Het diagram met u laat echter zien, dat de dieptegemiddelde snelheid ruim twee keer zo groot is. Dit komt doordat de waterkolom die de golven moet opstuwen minder hoog is, het volume moet dus in de breedte gezocht worden. De voortplantingssnelheid van de golf ligt lager, op ongeveer de helft van het vorige experiment. Volgens de theorie moet de factor tussen de snelheden gelijk zijn aan $\frac{v'}{v} = \frac{\sqrt{h'}}{\sqrt{h}} = \frac{\sqrt{2}}{\sqrt{10}} = \sqrt{\frac{1}{5}} \approx 0.45$.

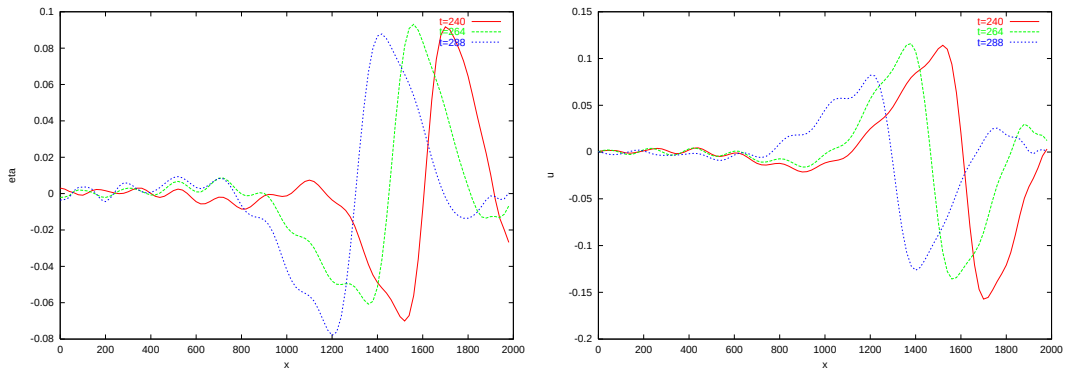
In de loop van de tijd, komt de golf bij de linkerwand ($x = 0$) aan, alwaar een nieuwe botsing plaatsvindt. Inmiddels is de oplossing ook al flink verstoord, zoals figuur 4 laat zien. Vermoedelijk worden de kleine golfjes veroorzaakt door de niet-lineariteit van het stelsel. Verkleining van stapgrootte en verhoging van het aantal collocatiepunten heeft te weinig effect. Een extra controle zou zijn om de eerste en laatste Chebychev-coëfficiënt met elkaar te vergelijken; wanneer c_N slechts een factor 10 tot 100 kleiner is dan c_0, dan is de oplossing niet erg betrouwbaar meer.



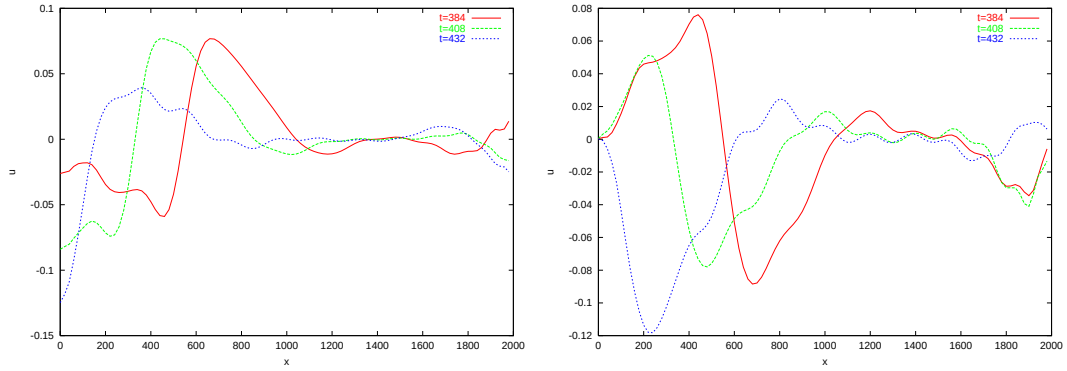
Figuur 3: De oplossingen op vijf verschillende tijdstippen, gedurende de botsing tegen de rechterwand bij $h_0 = 2$ en $h_1 = 0$, $N = 25$ en $\Delta t = 0.05$. Links de oplossingen voor η, rechts de oplossingen voor u.



Figuur 4: De oplossingen op drie verschillende tijdstippen, gedurende de botsing tegen de linkwand bij $h_0 = 2$ en $h_1 = 0$, $N = 25$ en $\Delta t = 0.05$. Links de oplossingen voor η, rechts de oplossingen voor u.



Figuur 5: De oplossingen op drie verschillende tijdstippen in het ondiepe gedeelte, $h_0 = 10$ en $h_1 = 8$, $N = 25$ en $\Delta t = 0.1$. Links de oplossingen voor η, rechts de oplossingen voor u.



Figuur 6: De oplossingen op drie verschillende tijdstippen in het diepe gedeelte, $h_0 = 10$ en $h_1 = 8$, $N = 25$ en $\Delta t = 0.1$. Links de oplossingen voor η, rechts de oplossingen voor u.

Hellende bodem Tot slot wordt de invloed van een hellende bodem onderzocht. De starthoogte wordt weer teruggezet naar 10 m en het hoogteverval op 9.9 m. Hier loopt de simulatie spaak: de waarden blazen op, en de bak blijkt te ondiep. Aangezien het vorige experiment bij diepte 2 m wel goed liep, wordt nu het hoogteverval op 8 m gezet. De oplossingen zijn in figuur 5 in het ondiepe gedeelte bekeken. In figuur 6 worden de oplossingen in het diepe gedeelte bekeken. De kleine golfjes komen weer in de oplossing terug, een gevolg van de niet-lineariteit. Door de variërende diepte verandert de voortplantingssnelheid en de dieptegemiddelde snelheid. Gekeken wordt rond $x = 1600$ en $x = 400$. De verhouding tussen de snelheden zou ongeveer $\sqrt{3.6/8.4} \approx 0.65$ moeten zijn; dit komt redelijk overeen met de voorplantingssnelheden in figuur 5 en 6.

5 Conclusie

De spectrale methode met Chebychev-polynomen blijkt hoge nauwkeurigheid op te leveren, die snel groeit bij ophoging van het aantal basisfuncties. Het voordeel is dat de oplossing in de x-richting overal bekend is. Nadeel is het ontstaan van kleine golfjes in de oplossing, die vermoedelijk te maken hebben met de niet-linearitet en ook met de computational mode van Adams-Bashfort.

Wat betreft het golfgedrag, blijkt dat diep of ondiep water (niet *te* ondiep) weinig invloed heeft op de oppervlakte-uitwijking. De voortplantingssnelheid ligt wel lager in ondiep water. De dieptegemiddelde-snelheid ligt juist hoger in ondiep water, omdat er minder water uit de diepte is

om de golf op te stuwen. Bij een hellende bodem veranderen deze snelheden dus.

Bij een te ondiepe bodem (< 1 m) blaast de oplossing op. Misschien kan dit verbeterd worden door een impliciete methode.

A Listings

A.1 Hoofdprogramma

```
int main(int argc, char *argv[])
{ int N, nt, nx, npt;
  double h0, h1, dt, T, xtot, xmid, g;
  double *c1_eta, *c1_u, *c1_eta_old, *c1_u_old, *eta_new, *eta, *eta_old, *u_new, *u, *u_old, *x;
  double **c_eta, **c_u;

  N = -1;    // number of Chebychev-coefficients.
  h0 = -1;   // initial height.
  h0 = -1;   // total height-difference
  dt = -1;   // time-stepsize
  T = -1;    // endtime
  xtot = 2000; // space-domain [0,xtot]
  g = 9.81;
  nx = -1;
  npt = -1;

  /* Handle commandline arguments */
  for(int a = 1; a < argc; a++)
  { char *pchar = argv[a];
    switch ( pchar[0] )
    {
      case '-':
      {
        switch( pchar[1] )
        { case 'v':
            VERBOSE = true;
            break;
          case 'N':
            N = atoi(argv[++a]);
            break;
          case 'h':
            h0 = atof(argv[++a]);
            h1 = atof(argv[++a]);
            break;
          case 't':
            dt = atof(argv[++a]);
            break;
          case 'T':
            T = atof(argv[++a]);
            break;
          case 'X':
            xtot = atof(argv[++a]);
            break;
          case 'x':
            nx = atoi(argv[++a]);
            break;
          case 'y':
            npt = atoi(argv[++a]);
            break;

          default:
            fprintf(stderr, "Unrecognised commandline option: -%c\n", pchar[1]);
            break;
        }
      }
      break;
    }
  }

  if(N <= 0)
  { fprintf(stderr, "ERROR: no valid number of Chebychev-coefficients was specified, use '-N [nr of coeff's]'\n");
    exit(-1);
  }
  if(h0 <= 0)
  { fprintf(stderr, "ERROR: no valid waterheight, use '-h [initial height] [height difference]'\n");
    exit(-1);
  }
  if(h1 > h0)
  { fprintf(stderr, "ERROR: total height-difference must be smaller than/equal to initial height, use '-h [initial height] [height differ
    exit(-1);
  }
  if(dt <= 0)
  { fprintf(stderr, "ERROR: no valid time-stepsize was specified, use '-t [time-stepsize]'\n");
    exit(-1);
  }
```

```
if(T <= 0)
{ fprintf(stderr, "ERROR: no endtime was specified, use '-T [endtime]'\n");
  exit(-1);
}
if(xtot <= 0)
{ fprintf(stderr, "ERROR: no domainsize was specified, use '-X [xtot]'\n");
  exit(-1);
}
if(nx <= 0)
{ fprintf(stderr, "ERROR: no valid number of xpoints for output was specified, use '-x [# xpoints]'\n");
  exit(-1);
}
if(npt <= 0)
{ fprintf(stderr, "ERROR: no valid number of timepoints for output was specified, use '-y [# timepoints]'\n");
  exit(-1);
}

/** okay, all relevant variables are set, now deduce some more **/
nt = (int)(T/dt);
xmid = xtot/2.0;

if(VERBOSE)
{ printf("+=================================================================+\n");
  printf("|  NUST problem 4: Pseudospectra solution to a wave-problem     |\n");
  printf("+-----------------------------------------------------------------+\n");
}

/** Allocate data-structures **/
c_eta      = matallocd(nt+1, N);
c_u        = matallocd(nt+1, N);
c1_eta     = vecallocd(N);
c1_u       = vecallocd(N);
c1_eta_old = vecallocd(N);
c1_u_old   = vecallocd(N);
eta_new    = vecallocd(N+1);
eta        = vecallocd(N+1);
eta_old    = vecallocd(N+1);
u_new      = vecallocd(N+1);
u          = vecallocd(N+1);
u_old      = vecallocd(N+1);
x          = vecallocd(N+1);


/** initialize **/
collocationPoints(N, xmid, x);

/** initialize at t=0 **/
initT0(N, x, xmid, g, h0, eta, u);

/** first step with Euler **/
eulerStep(N, h0, h1, xmid, g, dt, x, eta, u, c_eta, c_u, c1_eta, c1_u, eta_new, u_new);

/** next steps with Adams-Bashort **/
abLoop(N, nt, h0, h1, xmid, g, dt, x, eta_old, u_old, eta, u, eta_new, u_new, c_eta, c_u, c1_eta_old, c1_u_old, c1_eta, c1_u);
chebyFwd(N, eta_new, c_eta[nt]);
chebyFwd(N, u_new, c_u[nt]);

/** print results **/
FILE *file_eta, *file_u, *file_dyneta, *file_dynu;
file_eta = fopen("eta.dat", "w");
file_u = fopen("u.dat", "w");
file_dyneta = fopen("dyna_eta.gnu", "w");
file_dynu = fopen("dyna_u.gnu", "w");

double curx;
int ts;
ts = nt/npt;

fprintf(file_dyneta, "set data style lines\n");
fprintf(file_dynu, "set data style lines\n");
fprintf(file_dyneta, "set view 90,0\n");
fprintf(file_dynu, "set view 90,0\n");

for(int k = 0; k <= nt; k+=ts)
{ for(int i = 0; i < nx; i++)
  { curx = i*xtot/nx;
    fprintf(file_eta, "%G %G %G\n", curx, k*dt,chebyBwd(N, c_eta[k], phys2spec(xmid, curx)));
    fprintf(file_u, "%G %G %G\n", curx, k*dt,chebyBwd(N, c_u[k], phys2spec(xmid, curx)));
  }
  fprintf(file_eta, "\n\n");
```

```
    fprintf(file_u, "\n\n");
    fprintf(file_dyneta, "splot [:] [:] [-0.2:0.2] 'eta.dat' index %d\npause 0\n", k/ts);
    fprintf(file_dynu, "splot [:] [:] [-0.2:0.2] 'u.dat' index %d\npause 0\n", k/ts);
  }
  fprintf(file_dyneta, "reset\n");
  fprintf(file_dynu, "reset\n");

  fclose(file_eta);
  fclose(file_u);
  fclose(file_dyneta);

  /** free datastructures **/
  free(c_eta);
  free(c_u);
  free(c1_eta);
  free(c1_u);
  free(c1_eta_old);
  free(c1_u_old);
  free(eta_new);
  free(eta);
  free(eta_old);
  free(u_new);
  free(u);
  free(u_old);
  free(x);


  if(VERBOSE)
  { fprintf(stdout, "+============================================================+\n\n");
  }
  return 0;
}
```

A.2 Chebychev-routines

```
\subsection{Chebychev-routines}
\label{list:cheby}
\begin{scriptsize}
\begin{verbatim}
/**
 * Computes the Chebychev-coefficients of a function u.
 * @param N The number of Chebychev-coefficients.
 * @param u The vector containing the values of u at the collocation-points (ranging from 1 to -1, descending)
 *          the collocation-points should also have been tranformed to physical domain before u was applied.
 * @param c The vector into which the Chebychev-coefficients will be written.
 **/
void chebyFwd(unsigned int N, double *u, double *c)
{ int j, k;
  double p, pk;

  p = M_PI/N;

  for(j = 0; j < N; j++)
  { c[j] = 0.5 * u[0];
    for(k = 1; k < N; k++)
    { pk = p*k;
      c[j] += (u[k] * cos(pk*j));
    }
    c[j] += (0.5 * u[N] * cos(M_PI*j));

    c[j] *= (2.0/N);
  }
  c[0] /= 2.0;
}

/**
 * Computes the value of a function u at point x.
 * @param N The number of Chebychev-coefficients.
 * @param c The vector containing the N Chebychev-coefficients of function u.
 * @param x The point at which u should be evaluated; should be in [-1, 1].
 **/
double chebyBwd(unsigned int N, double *c, double x)
{ int j;
  double d, dd, sv;

  d = 0.0;
  dd = 0.0;

  for(j = N-1; j > 0; j--)
```

```
  { sv = d;
    d = 2.0*x*d - dd + c[j];
    dd = sv;
  }
  return x*d - dd + c[0];

}

/**
 * Computes the Chebychev-coefficients of the first derivative of a function u, given the coefficients of u itself.
 * @param N The number of chebychev-coefficients.
 * @param c The vector containing the N Chebychev-coefficients of u.
 * @param c1 The vector into which the Chebychev-coefficients of the derivative will be written.
 **/
void chebyDer1(unsigned int N, double *c, double *c1)
{ int j;
  c1[N-1] = 0.0;
  c1[N-2] = 2*(N-1)*c[N-1];
  for(j = N-3; j > 0; j--)
  { c1[j] = c1[j+2] + 2*(j+1)*c[j+1];
  }
  c1[0] = 0.5*c1[2] + c[1];
}


/**
 * Computes the collocation-points at the PHYSICAL domain.
 * @param N N+1 collocation-points are computed.
 * @param xmid Half the length of the physical domain.
 * x The vector into which the collocation-points will be written.
 **/
void collocationPoints(unsigned int N, double xmid, double *x)
{ int i;
  double p;

  p = M_PI/N;

  for (i = 0; i <= N; i++)
  { x[i] = cos(p*i);
    x[i] += 1.0;
    x[i] *= xmid;
  }
}

/**
 * Transforms a point x in the PHYSICAL domain [0,xtot]
 * to a point in the SPECTRAL domain [-1,1]
 **/
double phys2spec(double xmid, double x)
{ return (x/xmid)-1.0;
}
```

A.3 Oplosmethoden

```
/**
 * Initializes the two variables eta and u at t=0.
 * @param N N+1 collocation-points are used.
 * @param x The collocation-points at the PHYSICAL domain.
 * @param xmid Half the length of the physical domain.
 * @param g The gravitational constant.
 * @param h0 The initial height of the water.
 * @param eta The vector into which the values of eta at the collocation-points will be written.
 * @param u The vector into which the values of u at the collocation-points will be written.
 **/
void initT0(unsigned int N, double *x, double xmid, double g, double h0, double *eta, double *u)
{ int i;
  double dx2, k, omega;

  dx2 = 0.2*xmid*xmid;
  k = 2*M_PI/xmid;
  omega = k*sqrt(g*h0);

  for(i = 0; i <= N; i++)
  { eta[i] = 0.1*exp(-(x[i]-xmid)*(x[i]-xmid)/dx2) * sin(-k*x[i]);
    u[i] = g*k*eta[i]/omega;
  }
}

/**
```

```
 * Computes the solutions of eta and u at t=1*dt using Euler-Forward
 * @param N The number of Chebychev-coefficients.
 * @param h0 The initial height.
 * @param h1 The total height difference.
 * @param xmid Half the length of the physical domain.
 * @param g The gravitational constant.
 * @param dt The time-stepsize.
 * @param x The vector containing the collocation-points.
 * @param eta The vector containing the values of eta at the collocation-points at t=0
 * @param u The vector containing the values of u at the collocation-points at t=0
 * @param c_eta The matrix of the Chebychev-coefficients of eta. (empty)
 * @param c_u The matrix of the Chebychev-coefficients of u. (empty)
 * @param c1_eta The vector containing the Chebychev-coefficients of the derivative of eta. (empty)
 * @param c1_u The vector containing the Chebychev-coefficients of the derivative of u. (empty)
 * @param eta_new The vector into which the values of eta at the collocation-points at t=1*dt will be written
 * @param u_new The vector into which the values of u at the collocation-points at t=1*dt will be written
 */
void eulerStep(unsigned int N, double h0, double h1, double xmid, double g, double dt, double *x, double *eta, double *u,
               double **c_eta, double **c_u, double *c1_eta, double *c1_u, double *eta_new, double *u_new)
{ int j;
  chebyFwd(N, eta, c_eta[0]);
  chebyFwd(N, u, c_u[0]);
  chebyDer1(N, c_eta[0], c1_eta);
  chebyDer1(N, c_u[0], c1_u);

  eta_new[0] = eta[0] + dt*(f_eta(N, h0, h1, xmid, c1_eta, c1_u, eta[0], u[0], x[0]));
  eta_new[N] = eta[N] + dt*(f_eta(N, h0, h1, xmid, c1_eta, c1_u, eta[N], u[N], x[N]));
  u_new[0] = 0.0;
  u_new[N] = 0.0;
  for(j = 1; j < N; j++)
  { eta_new[j] = eta[j] + dt*f_eta(N, h0, h1, xmid, c1_eta, c1_u, eta[j], u[j], x[j]);
    u_new[j] = u[j] + dt*f_u(N, xmid, g, c1_eta, c1_u, u[j], x[j]);
  }

}

/**
 * Computes the solutions of eta and u at all t in [2*dt, nt*dt] using Adams-Bashfort
 * @param N The number of Chebychev-coefficients.
 * @param nt The number of timesteps.
 * @param h0 The initial height.
 * @param h1 The total height difference.
 * @param xmid Half the length of the physical domain.
 * @param g The gravitational constant.
 * @param dt The time-stepsize.
 * @param x The vector containing the collocation-points.
 * @param eta_old The vector into which the values of eta at the collocation-points at the previous t were written
 * @param u_old The vector into which the values of u at the collocation-points at the previous t were written
 * @param eta The vector containing the values of eta at the collocation-points at current t
 * @param u The vector containing the values of u at the collocation-points at current t
 * @param eta_new The vector into which the values of eta at the collocation-points at next t will be written
 * @param u_new The vector into which the values of u at the collocation-points at next t will be written
 * @param c_eta The matrix of the Chebychev-coefficients of eta.
 * @param c_u The matrix of the Chebychev-coefficients of u.
 * @param c1_eta_old The vector containing the Chebychev-coefficients of the derivative of eta at previous t.
 * @param c1_u_old The vector containing the Chebychev-coefficients of the derivative of u at previous t.
 * @param c1_eta The vector containing the Chebychev-coefficients of the derivative of eta.
 * @param c1_u The vector containing the Chebychev-coefficients of the derivative of u.
 */
void abLoop(unsigned int N, int nt, double h0, double h1, double xmid, double g, double dt,
            double *x, double *eta_old, double *u_old, double *eta, double *u, double *eta_new, double *u_new,
            double **c_eta, double **c_u, double *c1_eta_old, double *c1_u_old, double *c1_eta, double *c1_u)
{ int n, j;

  for(n = 1; n < nt; n++)
  { veccopyd(N+1, eta, eta_old);
    veccopyd(N+1, eta_new, eta);
    veccopyd(N+1, u, u_old);
    veccopyd(N+1, u_new, u);

    chebyFwd(N, eta, c_eta[n]);     /* det. Cheby coeff from current solution */
    chebyFwd(N, u, c_u[n]);

    veccopyd(N, c1_eta, c1_eta_old);
    veccopyd(N, c1_u, c1_u_old);

    chebyDer1(N, c_eta[n], c1_eta); /* det. Cheby coeff from current derivative */
    chebyDer1(N, c_u[n], c1_u);

    eta_new[0] = eta[0] + dt*(1.5*f_eta(N, h0, h1, xmid, c1_eta, c1_u, eta[0], u[0], x[0])
```

```
                              - 0.5*f_eta(N, h0, h1, xmid, c1_eta_old, c1_u_old, eta_old[0], u_old[0], x[0]));
    eta_new[N] = eta[N] + dt*(1.5*f_eta(N, h0, h1, xmid, c1_eta, c1_u, eta[N], u[N], x[N])
                              - 0.5*f_eta(N, h0, h1, xmid, c1_eta_old, c1_u_old, eta_old[N], u_old[N], x[N]));
    u_new[0] = 0.0;
    u_new[N] = 0.0;

    for(j = 1; j < N; j++)
    { eta_new[j] = eta[j] + dt*(1.5*f_eta(N, h0, h1, xmid, c1_eta, c1_u, eta[j], u[j], x[j])
                                - 0.5*f_eta(N, h0, h1, xmid, c1_eta_old, c1_u_old, eta_old[j], u_old[j], x[j]));
      u_new[j] = u[j] + dt*(1.5*f_u(N, xmid, g, c1_eta, c1_u, u[j], x[j])
                            - 0.5*f_u(N, xmid, g, c1_eta_old, c1_u_old, u_old[j], x[j]));
    }
  }
}

/**
 * Computes d eta/dt
 * @param N The number of Chebychev-coefficients
 * @param h0 The initial height
 * @param h1 The total height difference
 * @param xmid Half the length of the physical domain.
 * @param c1_eta The vector containing the Chebychev-coefficients of d eta/dx
 * @param c1_u The vector containing the Chebychev-coefficients of du/dx
 * @param eta The value of eta at the point x
 * @param u The value of u at the point x
 * @param x The point at which the derivative should be evaluated.
 */
double f_eta(unsigned int N, double h0, double h1, double xmid, double *c1_eta, double *c1_u, double eta, double u, double x)
{ double f;

  f = -(h0 - h1*x/(2.0*xmid) + eta)*chebyBwd(N,c1_u, phys2spec(xmid, x))/xmid;
  if(x==0.0 || x==2.0*xmid)
  { return f;
  }
  else
  { f += -(-h1/(2.0*xmid) + chebyBwd(N, c1_eta, phys2spec(xmid, x))/xmid)*u;
    return f;
  }
}

/**
 * Computes du/dt
 * @param N The number of Chebychev-coefficients
 * @param xmid Half the length of the physical domain.
 * @param g The gravitational constant
 * @param c1_eta The vector containing the Chebychev-coefficients of d eta/dx
 * @param c1_u The vector containing the Chebychev-coefficients of du/dx
 * @param u The value of u at the point x
 * @param x The point at which the derivative should be evaluated.
 */
double f_u(unsigned int N, double xmid, double g, double *c1_eta, double *c1_u, double u, double x)
{ double f;
  if(x==0.0 || x==2.0*xmid)
  { f = 0.0;
  }
  else
  { f = -u*chebyBwd(N, c1_u, phys2spec(xmid, x))/xmid - g*chebyBwd(N, c1_eta, phys2spec(xmid, x))/xmid;
  }

  return f;
}
```