Gravity Waves and Solitons

A Scottish engineer in the mid nineteenth century John Scott Russell built a 30’ wave tank in his backyard. Dropping a weight at one end of the channel produced a solitary gravity wave with speed

\[ c = \sqrt{g(h + a)} \]

where \( h \) is the height of undisturbed water in the channel, \( a \) is the wave amplitude, and \( g \) is the acceleration of gravity. He observed a compound wave which split into two with the larger one racing ahead of the smaller. Wave pools in amusement parks operate on a similar principle, using a plunger at one end of pool, blowing air on surface at one end, or dumping a large volume of water at one end.

Boussinesq in (1871) and Rayleigh in (1876) showed that the waveform of Russell’s wave is

\[ u(x, t) = a \sech^2[\beta(x - ct)] , \quad \beta = \frac{1}{2h} \sqrt{\frac{3a}{h + a}}. \]

Korteweg-de Vries equation

Korteweg and de Vries (1895) discovered the KdV equation

\[ \frac{\partial u}{\partial t} + (1 + u) \frac{\partial u}{\partial x} + \frac{\partial^3 u}{\partial x^3} = 0 , \]

which has this solitary wave as a solution.
The KdV equation can be written in different forms by making changes of dependent and independent variables, e.g., let \( u \rightarrow u - 1 \)

\[
    u_t + uu_x + u_{xxx} = 0
\]

where a subscript \( t \) or \( x \) stands for \( \partial/\partial t \) or \( \partial/\partial x \).

Letting \( u \rightarrow -6u \) gives another commonly used form

\[
    u_t - 6uu_x + u_{xxx} = 0
\]

**Korteweg - de Vries Solitons**

Let \( z = x - ct \) where \( c \) is a positive constant. Let's look for a right-moving wave solution \( u(x, t) = f(z) \)

\[
    -cf' + ff' + \frac{d^3f}{dz^3} = \frac{d}{dz} \left(-cf + \frac{1}{2}f^2 + f''\right) = 0. \]

This can be integrated in \( z \)

\[
    -cf + \frac{1}{2}f^2 + f'' = A
\]

where \( A \) is an integration constant.

If \( f(z) \) represents a localized solution in \( z \) then \( f(\pm \infty) = f'(\pm \infty) = f''(\pm \infty) = 0 \) (unlike a plane wave), so the integration constant \( = 0 \).
Multiplying the above by the integrating factor \( f' \)

\[
f' \left( -cf + \frac{1}{2} f^2 + f'' \right) = \frac{d}{dz} \left( -\frac{c}{2} f^2 + \frac{1}{6} f^3 + \frac{1}{2} f'^2 \right) = 0 .
\]

This can be integrated again, with zero integration constant

\[
-\frac{c}{2} f^2 + \frac{1}{6} f^3 + \frac{1}{2} f'^2 = 0
\]

\[
\frac{df(z)}{dz} = \sqrt{cf(z)} \sqrt{1 - \frac{f(z)}{3c}}.
\]

This equation can be integrated analytically:

\[
u(x, t) = 3c \text{sech}^2 \left[ \frac{\sqrt{c}}{2} (x - ct - x_0) \right]
\]

where \( x_0 \) is an arbitrary integration constant which gives the location of the peak at \( t = 0 \). Note that the soliton amplitude is proportional to its speed \( c \).
Multi-soliton solutions


\[ u_t + uu_x + \delta^2 u_{xxx} \equiv \frac{\partial u(x, t)}{\partial t} + u(x, t) \frac{\partial u(x, t)}{\partial x} + \delta^2 \frac{\partial^3 u(x, t)}{\partial x^3} = 0 \]

Choosing the parameter \( \delta = 0.022 \) and the initial condition

\[ u(x, 0) = \cos(\pi x) \]

they solved the equation numerically using a finite difference scheme and discovered solitons! They showed that

- Solitons accelerate as they pass through one another
- The joint amplitude decreases instead of simply adding – nonlinear!
- Soliton shapes are preserved, but there is a phase shift

Exact multi-soliton solutions

Later, the inverse scattering transform was used to derive analytic multi-soliton solutions by Gardner, Greene, Kruskal and Miura Phys. Rev. Lett. 19, 1095 (1967).
An example of a 2-soliton solution is

\[ u(x, t) = \frac{4}{3} + 4 \cosh(2x - 8t) + \cosh(4x - 64t) \]
\[ \frac{1}{[3 \cosh(x - 28t) + \cosh(3x - 36t)]^2} \]
\[ \sim \frac{4}{3} \sech^2 \left[ 2(x - 16t) \mp \frac{\log 3}{2} \right] + \frac{1}{3} \sech^2 \left[ x - 4t \pm \frac{\log 3}{2} \right] \]
\[ \text{as } t \to \pm \infty \]

- One soliton is 4 times taller than the other and travels 4 times faster
- In the scattering the taller moves forward \( \frac{1}{2} \log 3 \) and the shorter moves back \( \log 3 \)

An \( N \)-soliton solution has the asymptotic behavior

\[ u(x, t) \sim \frac{1}{3} \sum_{n=1}^{N} n^2 \sech^2 \left[ n(x - 4n^2t) \mp x_n \right] , \quad \text{as } t \to \pm \infty \]

At the initial time

\[ u(x, t = 0) = \frac{N(N + 1)}{6} \sech^2 x \]

As \( t \to \infty \) the solitons separate with the tallest moving fastest and the shortest slowest. The phases are given by

\[ x_n = \frac{1}{2} \log \left[ \prod_{\substack{m=1 \atop m \neq n}}^{N} \frac{|n - m|}{|n + m|} \right]^{\text{sgn}(n-m)} \]
Discrete Fourier Transforms

The discrete Fourier sine transform

\[ a_k = \sum_{i=1}^{63} x_i \sin \frac{ik\pi}{64}, \quad k = 1, 2, \ldots, 63 \]

was used by Fermi, Pasta and Ulam.

In general, the DFT transforms the number sequence \( x_j, j = 0, \ldots, N - 1 \) into the sequence \( a_k, k = 0, \ldots, N - 1 \) with an \( N \times N \) matrix of complex \( N \)-th roots of unity

\[ a_k = \sum_{j=0}^{N-1} W^{-kj} x_j, \quad W = e^{2\pi i/N} \]

The inverse transformation is

\[ x_j = \frac{1}{N} \sum_{k=0}^{N-1} W^{jk} a_k. \]

For an even number \( 2N \) of real values \( x_j = 0, \ldots, 2N - 1 \) the DFT

\[ a_k = \sum_{j=0}^{2N-1} x_j \cos \left( \frac{\pi jk}{N} \right) - i \sum_{j=0}^{2N-1} x_j \sin \left( \frac{\pi jk}{N} \right) \]
naturally decomposes into a discrete cosine transform (DCT)

\[ R_{a_k} = \sum_{j=0}^{N-1} \left[ x_j + (-1)^k x_{j+N} \right] \cos \left( \frac{\pi j k}{N} \right) \]

and sine (DST) transforms

\[ S_{a_k} = \sum_{j=0}^{N-1} \left[ x_j + (-1)^k x_{j+N} \right] \sin \left( \frac{\pi j k}{N} \right) \]

**Complex Discrete Fourier Transform**

We will use the GNU Scientific Library functions, which use the following **Mathematical Definitions** for the FFT

\[ x_j = \sum_{k=0}^{N-1} z_k \exp \left( -\frac{2\pi i j k}{N} \right) \]

and inverse FFT

\[ z_j = \frac{1}{N} \sum_{k=0}^{N-1} z_k \exp \left( \frac{2\pi i j k}{N} \right) \]

These transforms are periodic in the indices with period \( N \). The routines are most efficient when \( N \) is a power of 2.

The GSL also has real versions of these functions which can be used to compute Fourier sine and cosine transforms of length \( N/2 - 1 \).
The Split Step Fourier Method

The Split Step Fourier method is one of the most efficient algorithms to solve the initial value problem for nonlinear partial differential equations.

Wikiwaves has a nice page on Numerical Solution of the KdV equation using the split step method.

Decomposition into linear and nonlinear operators

The KdV equation can be written

\[
\frac{\partial u(x, t)}{\partial t} = -\delta^2 u_{xxx} - uu_x = \mathcal{L}u(x, t) + \mathcal{N}u(x, t)
\]

where

\[
\mathcal{L} \equiv -\delta^2 \frac{\partial^3}{\partial x^3}, \quad \mathcal{N} \equiv -u(x, t) \frac{\partial}{\partial x}.
\]

Note that the nonlinear operator depends on the solution, which is not known explicitly.

The solution can be written as a superposition of Fourier modes in wavenumber space

\[
u(x, t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} dp \ e^{ipx} \tilde{u}(p, t).
\]

The linear part of the equation is then a simple ordinary differential equation

\[
\frac{d\tilde{u}(p, t)}{dt} = i\delta^2 p^3 \tilde{u}(k, t)
\]
for each mode amplitude

\[ \tilde{u}(p, t) = \tilde{u}(p, 0) e^{i \delta^2 p^3 t} . \]

**Split Step Algorithm**

The algorithm is based on the formal operator solution

\[ u(x, t) = \exp \left[ \int_0^t dt' (N + L) \right] u(x, 0) . \]

This is not an explicit solution because the nonlinear operator depends on the unknown solution for all times \( 0 < t' < t \).

The algorithm approximates the formal solution over a small time step \( dt \)

\[ u(x, t + dt) = \exp \left[ \int_t^{t+dt} dt' (N + L) \right] u(x, 0) \approx \exp \left[ \int_t^{t+dt} dt' N \right] \exp \left[ \int_t^{t+dt} dt' L \right] u(x, t) . \]

This factorization introduces errors of \( O(dt^2) \) because the operators \( N, L \) do not commute with one another

\[ [N, L] \neq 0 . \]

The errors can be estimated using the Baker-Campbell-Hausdorff Formula.

For the Korteweg-de Vries equation in the form used by Zabusky and Kruskal

\[ u_t + uu_x + \delta^2 u_{xxx} = 0 , \]
the nonlinear term can be written

\[ uu_x = F_x , \quad F[u(x)] = \frac{1}{2} u^2 . \]

A simple split-step Fourier method is described in Wikiwaves: Numerical solution of the KdV. This simple method uses the Euler forward difference scheme to evolve the Fourier modes in time. This works well for short times, but is unstable for long integration times.

Zabusky and Kruskal used finite differencing in space and leap-frog time stepping, which is stable for long times.


**Stable Fourier Leapfrog Method for the KdV Equation**

Consider the KdV equation on the spatial interval \(0 \leq x < L\) with periodic boundary conditions \(u(0,t) = u(L,t)\). Divide the interval into \(N\) segments with \(x_j = jL/N, \ j = 0,1,\ldots,N-1\), and define the discrete Fourier transform of the solution vector \(\{u_j = u(x_j)\}\)

\[
\tilde{u}_k = \sum_{j=0}^{N-1} W^{-kj} u_j , \quad W = e^{i2\pi/N} , \quad k = 0,1,\ldots,N-1 ,
\]

with inverse transform

\[
u_j = \frac{1}{N} \sum_{k=0}^{N-1} W^{jk} \tilde{u}_k .
\]
The wavenumbers of the Fourier modes are defined by

\[ e^{ix_j p_k} = e^{i2\pi j k/N} , \quad p_k = \begin{cases} 2\pi k/L & \text{for } k \leq N/2 \\ -2\pi(N - k)/L & \text{for } k > N/2 \end{cases} , \quad k = 0, 1, \ldots, N - 1 \]

Choose a finite time step \( \tau \) and consider successive times \( t_n = n\tau, n = 0, 1, 2, \ldots \). The Chan-Kerkhoven algorithm involves the solution and its Fourier transform at three successive times: the current time \( t \), the previous step time \( t^- = t-\tau \), and the next step time \( t^+ = t+\tau \). The KdV equation

\[ u_t + \frac{\partial}{\partial t} \left( \frac{u^2}{2} \right) + \delta^2 u_{xxx} = 0 \]

is discretized on the wavenumber-time grid using the algorithm

\[ \frac{\tilde{u}^+_k - \tilde{u}^-_k}{2\tau} + ip_k C_k \left( \frac{u^2}{2} \right) + \delta^2 (ip_k)^3 \left[ \frac{\tilde{u}^+_k + \tilde{u}^-_k}{2} \right] = 0 , \]

where the operator \( C \) implements a Fourier Collocation method. The simplest collocation method takes the Fourier transform of the inverse transform of the argument:

- compute the inverse Fourier transform of \( u^2(x, t)/2 \),
- and then compute the Fourier transform of the result.

To solve for \( \tilde{u}^+_k \) at the next time, the solution is needed at the two previous times. To start the algorithm given an initial waveform \( u(x, 0) \), an explicit time-step algorithm must be used to generate \( u(x, \tau) \), for example using the split-step Fourier method with a simple explicit Euler step

\[ \tilde{u}_1(p, \tau) = \tilde{u}(p, 0) e^{i\delta^2 p^3 \tau} , \]
\[ \frac{\ddot{u}_k - \ddot{u}_k}{\tau} + i p_k C_k \left( \frac{u_1^2}{2} \right) = 0. \]

If greater accuracy is desired the Euler step can be replaced by an adaptive Runge-Kutta step.
C++ Code for Fig. 1 in Zabusky-Kruskal article


#include <cmath>
#include <cstdlib>
#include <fstream>
#include <iostream>
#include <vector>
using namespace std;

int main()
{

    const double pi = 4 * atan(1.0); 

    double delta = 0.022;

    // spatial grid
    int N = 256;
    double L = 2.0;
    double h = L / double(N);    // spatial interval
vector<double> x(N);
for (int i = 0; i < N; ++i)
    x[i] = i * h;

// initialize solution vector on spatial grid
vector<double> u(N);
for (int i = 0; i < N; ++i)
    u[i] = cos(pi * x[i]);

// write initial waveform to file
ofstream ofs("kdv.out");
for (int i = 0; i < N; ++i)
    ofs << x[i] << 't' << u[i] << '\n';
ofs << '\n';

// temporal grid
double T_B = 1 / pi;
double k = h * h * h; // time step must satisfy CFL criterion

// take one time step to prime the algorithm
vector<double> u_minus(N);
for (int i = 0; i < N; ++i) {
    u_minus[i] = u[i];
    // u = cos(pi(x-ut)) if delta^2u_xxx is negligible
\[ u[i] = \cos(\pi \times (x[i] - u[i] \times k)) ; \]

\[
\text{double } t_{\text{max}} = 4 \times T_{\text{B}} ; \\
\text{double } t = 0 ; \\
\text{vector<double> } u_{\text{plus}}(N) ; \\
\text{for (t = 0; t } \leq t_{\text{max}} ; t += k) \{ \\
\text{// write waveform to file at selected times} \\
\text{if (abs(t - T_{\text{B}}) < k / 2 ||} \\
\text{abs(t - 3.6 \times T_{\text{B}}) < k / 2) \{ \\
\text{\text{for (int } i = 0 ; i < N ; ++i) \{ \\
\text{\quad ofs } \ll x[i] \ll 't' \ll u[i] \ll 'n';} \\
\text{\quad ofs } \ll 'n'; \\
\text{\}} \\
\text{\}} \\
\text{// use algorithm in Footnote 6 to evolve in time} \\
\text{for (int } i = 0 ; i < N ; ++i) \{ \\
\text{\quad int ip2 = (i + 2 + N) } \% \ N, \ ip1 = (i + 1 + N) } \% \ N, \\
\text{\quad im1 = (i - 1 + N) } \% \ N, \ im2 = (i - 2 + N) } \% \ N; \\
\text{\quad u_{\text{plus}}[i] } = \text{u_{\text{minus}}[i] } - (1 / 3.0) \times (k / h) *
(u[ip1] + u[i] + u[im1]) * (u[ip1] - u[im1]) -
(delta * delta * k / (h * h * h)) *
(u[ip2] - 2.0 * u[ip1] + 2.0 * u[im1] - u[im2]);

for (int i = 0; i < N; ++i) {
    u_minus[i] = u[i];
    u[i] = u_plus[i];
}

return 0;