Chapter 1

Simulation Solution

1.1 Smoothed Particle Hydrodynamics

1.1.1 PySPH

The simulations were conducted with PySPH ***, open source SPH code written in Python and compiled in Cython. This allows a majority of the code to be written in pure Python, but is then converted to high performance Cython code to run at speeds closer to C and FORTRAN. For further optimizations, PySPH can run in parallel with OpenMP and MPI to take advantage of multiple cores / threads. Also, one of the main reasons for choosing PySPH is that it allows user defined classes, and equations which was needed to incorporate the primordial black hole.

1.2 Code

In order to simulate a primordial black hole collision with a neutron star we must first write a class to include the acceleration. The acceleration of course is just

$$\vec{a} = -\frac{1}{m_{part}} \vec{\nabla} \Phi.$$

In this simulation and coordinates

$$\vec{a} = \frac{-m}{(x^2 + s^2 + (y + v\tau)^2)^{3/2}} \begin{pmatrix} x \\ y + v\tau \end{pmatrix}$$

with $G = 1$, $\tau = t - t\_hit$, and s a softening length so the solution does not diverge at $\tau = 0$ at the origin. The mass of the particle is also omitted in the simulation because PySPH uses acceleration per unit mass. In the class function (Figure 1.1) the velocity is hard coded to be 1, and the parameter *t_hit* is used to offset the collision of the primordial black hole since the simulation starts at $t = 0$. Now that the class function has been written, it can be called within the equations block of the main file, *blackhole.py.* Figure 1.2 shows the important sections of *blackhole.py*; first the acceleration equation is imported, and the values for the simulation are specified in the preamble. Then, the black hole can be added to the equations group in the main acceleration block. The example *hydrostatic_tank.py* was used as the base for *blackhole.py*, it was the most useful starting point since it is a $2D$ tank filled with fluid, and included the force of gravity.

The parameters of the simulation were mostly the same as what were used to generate the images of the analytic solution, everything set equal to unity, with the exceptions of *t_hit* = 200, and $s = 0.01$. The reason *t_hit* is so large is due how the particles are initially placed. They are simply placed on a grid with spacing dx; once the simulation starts gravity pulls

```python
from pysph.sph.equation import Equation

class BlackHole2D(Equation):
    def __init__(self, dest, sources, soft=0.05, t_hit=5.0, M=1.0):
        self.soft = soft # softening length to not divide by zero
        self.t_hit = t_hit # time when the black hole crosses the origin
        self.M = M # mass of black hole
        super(BlackHole2D, self).__init__(dest, sources)

    def initialize(self, d_idx, d_au, d_av):
        d_au[d_idx] = 0.0
        d_av[d_idx] = 0.0

    # calculate the force due to the black hole
    def loop(self, d_x, d_y, d_idx, d_au, d_av, t):
        d_au[d_idx] += -self.M * d_x[d_idx] / pow((d_x[d_idx]**2 + self.soft**2 + (d_y[d_idx] + t - self.t_hit)**2),3.0/2.0)
        d_av[d_idx] += -self.M * (d_y[d_idx] + t - self.t_hit) / pow((d_x[d_idx]**2 + self.soft**2 + (d_y[d_idx] + t - self.t_hit)**2),3.0/2.0)
```

Figure 1.1: Class file for adding the acceleration due to the primordial black hole, *BlackHoleEquation.py*.

```python
# Import the equations
from pysph.sph.equation import Group
from pysph.sph.BlackHoleEquation import BlackHole2D

# Domain and reference values
Lx = 120.0; H = 15.0; Ly = 1.5*H
gy = -1.0
Vmax = np.sqrt(abs(gy) * H)
c0 = 10 * Vmax; rho0 = 1.0
p0 = c0*c0*rho0
gamma = 1.0

soft = 0.01
t_hit = 200.0
Mass = 1.0
tf = 300.0

# Reynolds number and kinematic viscosity
Re = 0; nu = 0.01 # Ideal fluid

# Numerical setup
nx = 1600; dx = Lx/nx
ghost_extent = 5.5 * dx
hdx = 1.2

class BlackHole(Application):

    def create_equations(self):
        # Formulation for REF1
        equations1 = [

            # Main acceleration block
            Group(equations=[

                # Add the black hole
                BlackHole2D(dest='fluid', sources=None, soft=soft, t_hit=t_hit, M=Mass)

            ]),
        ]
```

Figure 1.2: Modifications of the hydrostatic tank example, *blackhole.py*.

the surface down attempting to equilibrate to a linear pressure gradient. However, since the fluid is not at equilibrium, the surface undergoes harmonic motion and is slowly dampened to equilibrium, and so the long *t_hit* is to dampen out a majority of the oscillations.

Determining the dimensions of the tank required a considerable amount of care. If the tank was too narrow, the walls at the edges would reflect the initial waves which would then start to interfere with the secondary waves. This caused erroneous results in the energy calculations during testing. Similarly, if the tank was too shallow, the waves resembled shallow water waves instead of deep, as in our model. And of course, if the tank was needlessly large, it greatly effected the computation time of the simulation. In the end, $Lx = 120$, $Ly = 15$, and $dx = 0.075$ were decided on.

1.3 Simulation Results

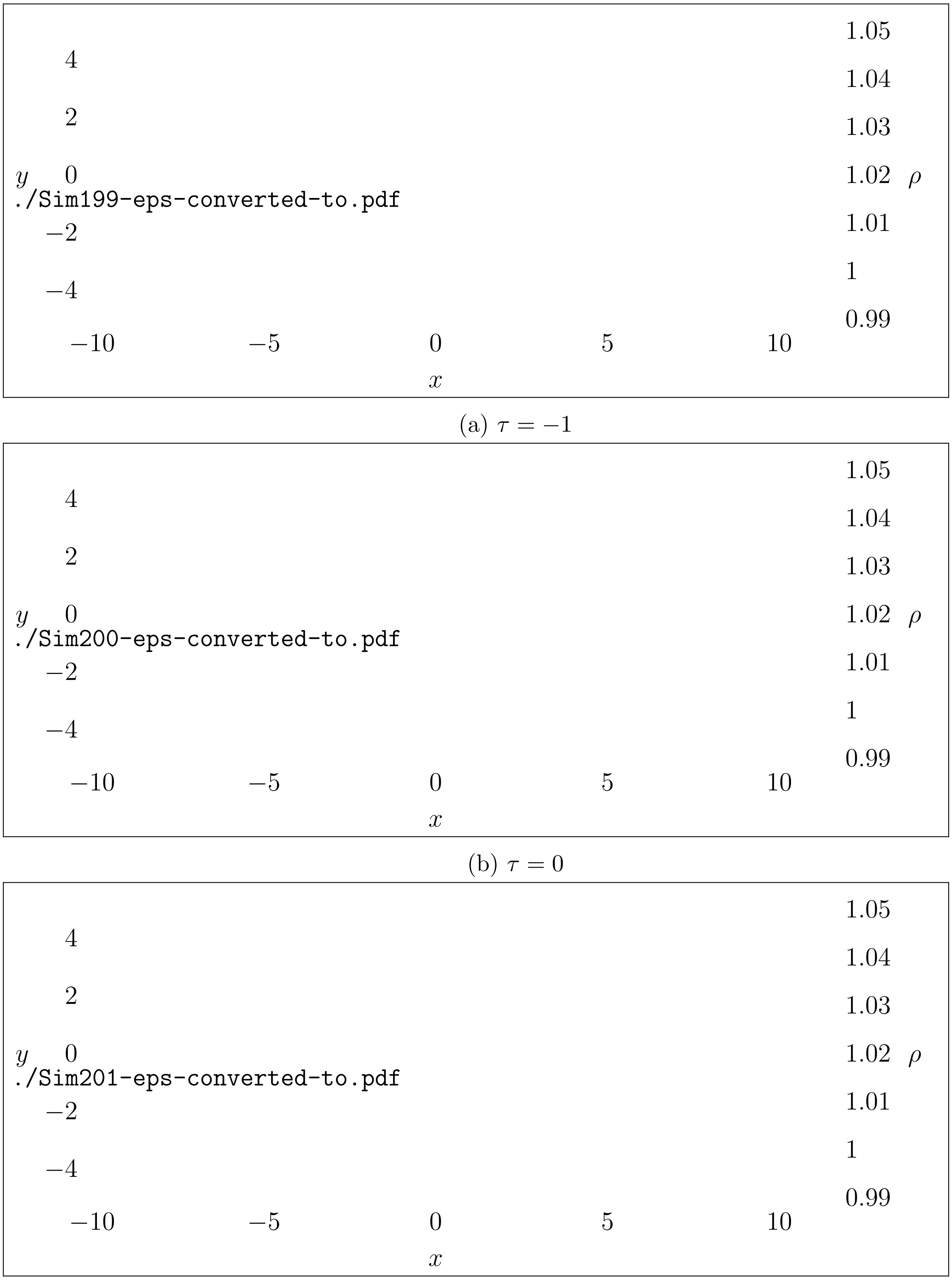
In total the simulation took about 330 CPU hours over the course of three days.

plots of waves

energy: calculation, 2d to 3d, noise, damping, weird increase

shock wave: didn't see in testing, PBH exiting tank, line in the middle is an artifact of the plot

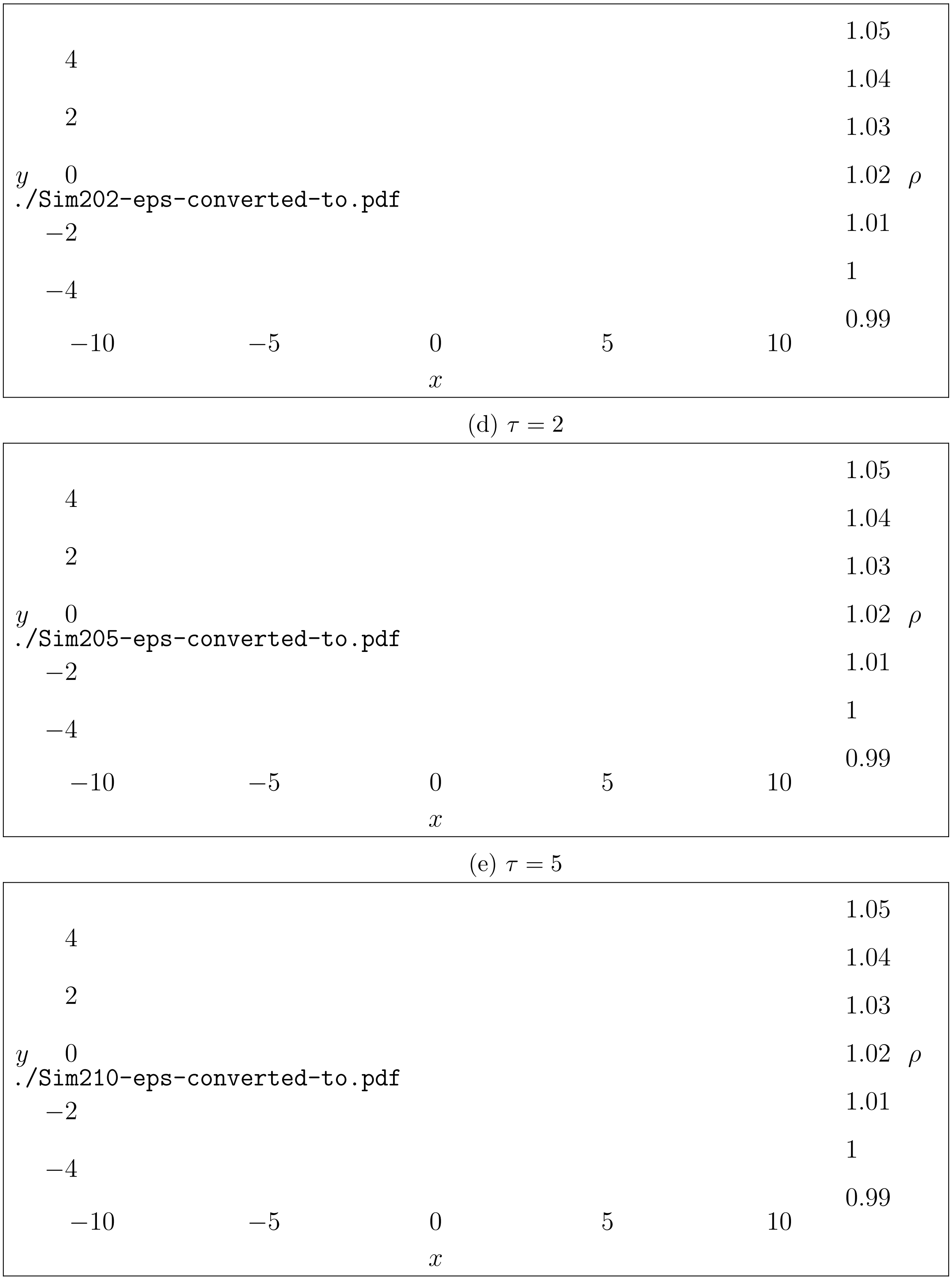
waves crashing into walls



(a) $\tau = -1$

(b) $\tau = 0$

(c) $\tau = 1$

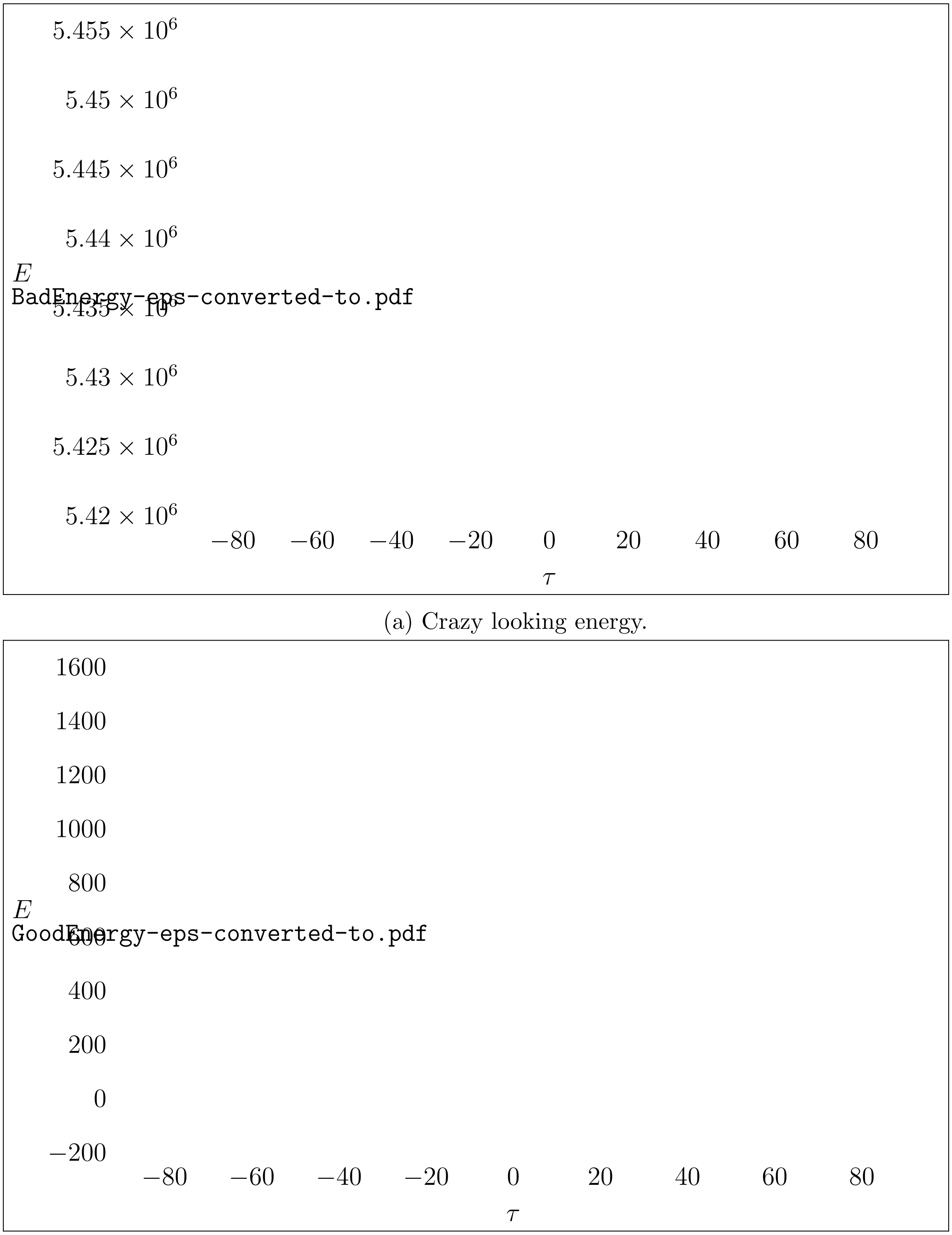


(d) $\tau = 2$

(e) $\tau = 5$

(f) $\tau = 10$

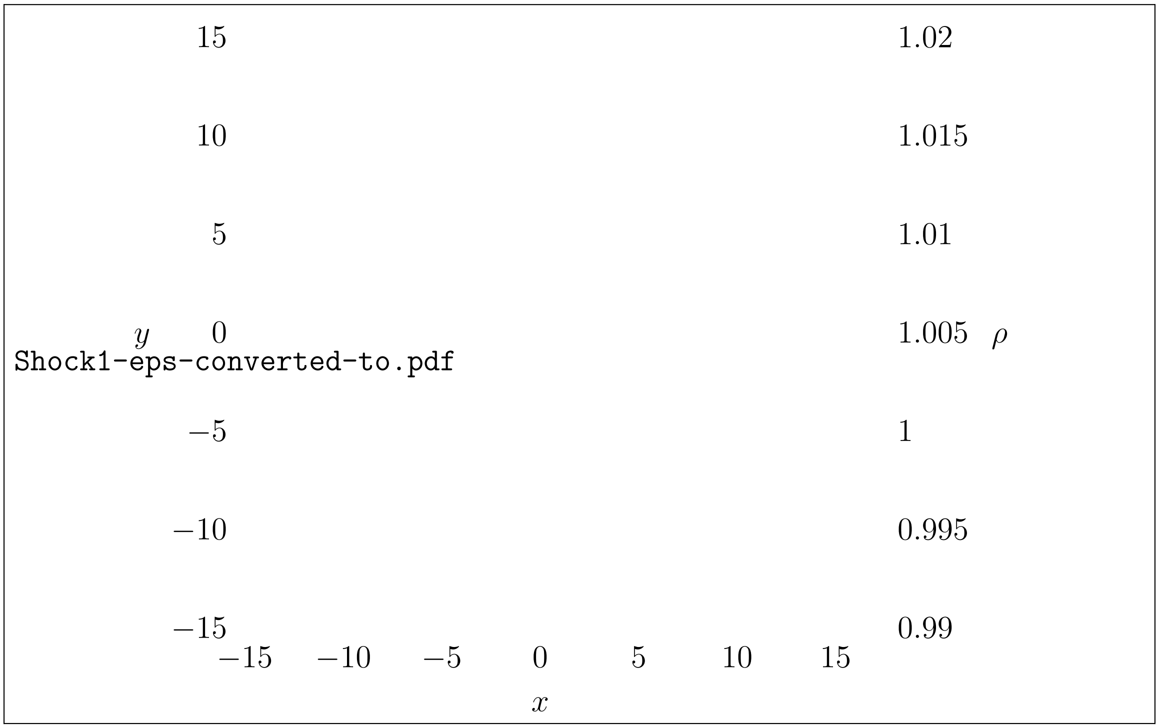
Figure 1.3: Simulated results. ***



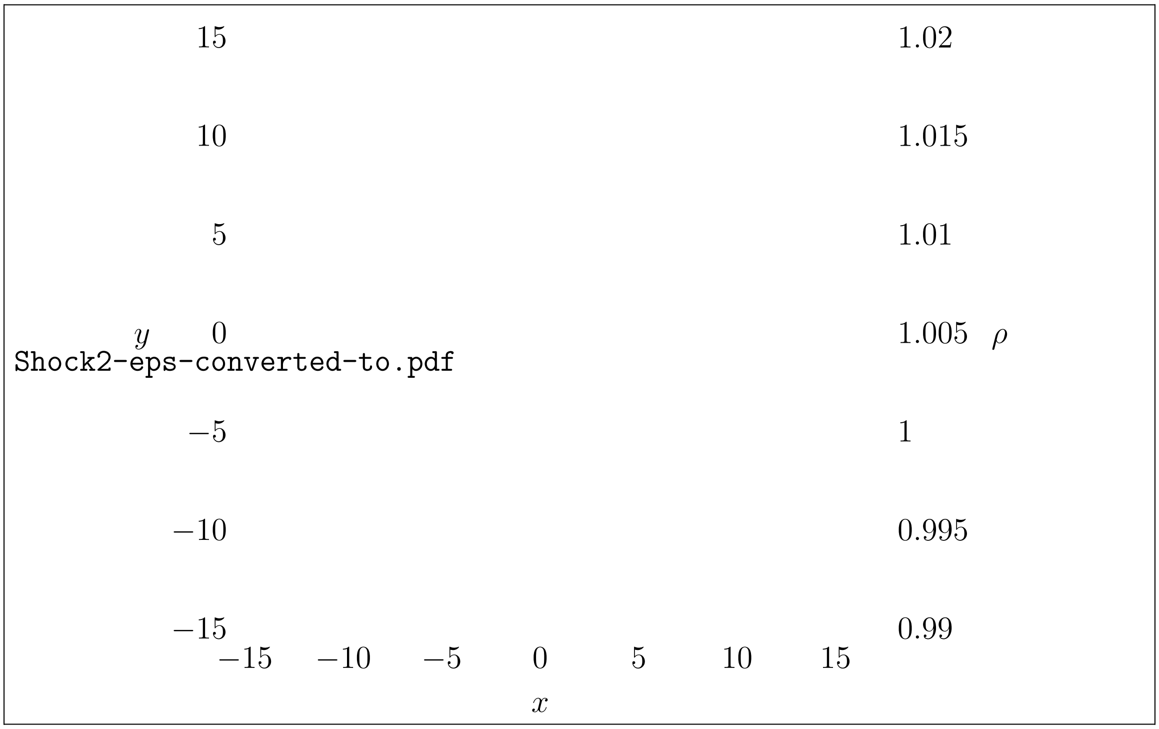
(a) Crazy looking energy.

(b) Smoothed energy without noise.

Figure 1.4: Energy transfer from the simulation.



(a) Initial shock-wave. $\tau = 15.24$.



(b) Reflected shock-wave. $\tau = 15.56$.

Figure 1.5: Shock-wave created by PBH leaving tank.