

# ENIAC's Problem

## 1 Discussion

### A Bit of History

Initiated during the height of World War II, the Electronic Numerical Integrator and Calculator (ENIAC) was designed to make ballistics tables, which required solving  $\mathbf{F} = m\mathbf{a}$  for the projectile as it moves down the barrel of the gun, and then flies, one hopes, to its intended target. John Mauchly (Ph.D., physics, Johns Hopkins) and J. Presper Eckert (MS EE, UPenn) completed the machine at University of Pennsylvania's Moore Engineering School in 1946—too late to contribute to the war effort.

Patent courts have determined that the design of the ENIAC was based on the much less sophisticated “ABC” computer built at Iowa State just before the war by John Vincent “JV” Atanasoff (Ph.D., physics, Madison) and Clifford Berry (then a student at Iowa State, but he completed his Ph.D., in physics at Iowa State in 1948). Atanasoff had been motivated to build his computer because of the lengthy quantum mechanical calculations required to complete his thesis.

A third “first” computer was designed by Howard Aiken (B.S. Madison, Ph.D., physics, Harvard) and built by IBM. Called the Automatic Sequence Controlled Calculator (ASCC) or Mark I this machine relied on IBM's electromechanical technology—relays rather than vacuum tubes. It was therefore slow (e.g., multiplication of two numbers took 6 seconds) and contributed little to the technology of electronic computers. Nevertheless this computer was in operation in 1944, and ran essentially continuously until the end of the war under Navy command.

Immediately following the war, computers were quietly built for code breaking and publicly built for nuclear weapon calculations. The business world (and hence IBM) was initially not much interested in duplicating the expensive, fragile, one-of-a-kind computers that were the immediate descendants of the ENIAC. The first commercial computer was the UNIVAC (Universal Automatic Computer), completed in 1951 by Eckert and Mauchly for the U.S. Bureau of the Census. A total of 46 were sold. Perhaps the first scientific computer, ERA 1103, was designed by Seymour Cray (MS, applied math, UMn) for NSA in 1951 and first publicly delivered in October 1953. IBM's famous “failure” at scientific computing, the Stretch (IBM 7030), was delivered to Los Alamos ten years after the first UNIVAC. A total of 8 were produced. Gene Amdahl (Ph.D., physics, Madison) was an initial planner for Stretch, and went on to become the Manager of Architecture for the IBM System/360, which was to dominate business computing for two decades. Meanwhile Cray's machines, like the CDC 6600, dominated scientific computing for an equal period of time.

### A Bit of Physics

The problem of solving  $F = ma$  in real-world situations is two fold.

First, while the fundamental force laws are analytic (like Newton's  $1/r^2$ ) the messy forces experienced in real life are usually not expressible with a simple formula. A body moving

“slowly”<sup>1</sup> through a fluid (like air) is subject to a simple drag force, linearly proportional to velocity, known as Stokes’ law (discovered by Sir George Stokes in 1845).

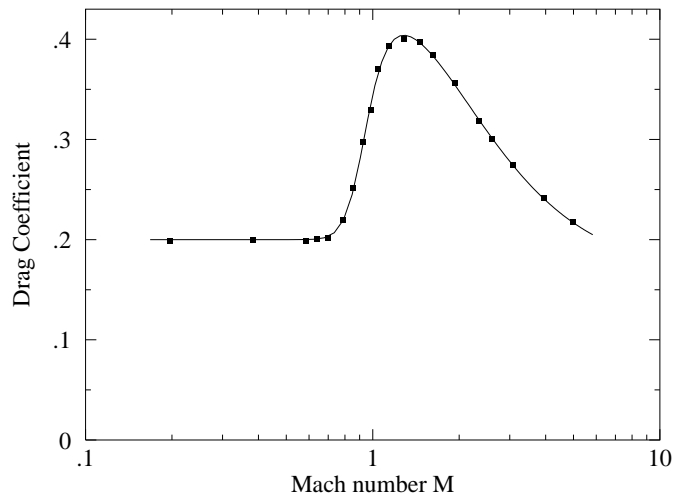
In the above context, “slow” means so slow that viscous forces dominate. In 1883 Osborne Reynolds defined a dimensionless number that now bears his name:

$$\text{Re} = \frac{2Rv}{\nu}$$

where  $\nu$  is the kinematic viscosity of the fluid through which a sphere of radius  $R$  is moving at a velocity  $v$ . For a sphere moving through a fluid,  $\text{Re} < 1$  is required for Stokes’ law to be closely followed, whereas for  $10^3 < \text{Re} < 10^5$ , the drag force is approximately:

$$\frac{1}{2}C_D\rho v^2\pi R^2$$

where  $C_D \approx .45$ . (For a 75 mm diameter shell, the upper limit corresponds to about 20 m/s.) For  $10^5 < \text{Re} < 10^6$ , the “drag crisis” occurs, and  $C_D$  is reduced. That takes us to nearly the speed of sound, at which point effects having to do with the compressibility of air become important. Above “Mach 1”<sup>2</sup>, the drag force depends approximately linearly on velocity (but unlike Stokes’ law: with a negative  $y$ -intercept). All the above formulae hide the fact that at “high” speeds, the fluid flow around the shell becomes turbulent and the actual force on the shell varies in time. Thus for real objects, the time-averaged drag force is experimentally determined at a variety of velocities and then an interpolating formula is used to approximate the drag force in between the known values. Rather than do the experiment we take our data from Marion & Thornton’s *Classical Dynamics*, which in turn takes its data from Rheinmetall’s *Handbook of Weaponry*. Below find the data with approximating curve



<sup>1</sup>What does moving “slowly” mean? In some contexts it means moving at near the speed of light, in other contexts it can mean moving less than 1 cm/year. Nothing that has units can be said to be big or small!! In this case you need a comparison velocity, say,  $v_c$ , so you can define “slow” as less than  $v_c$ , or better still, to say that the unitless number  $v/v_c$  is less than some pure number (often 1).

<sup>2</sup>Mach number:  $M = v/c$ , where  $c$  is the speed of sound, is named for Ernst Mach (1836–1916). “Mach 1” is traveling at the speed of sound. Mach is also well known for his provocative works on the philosophy of physics.

You can get this approximating function for  $C_D$  by including a package:

```
In[1] := <<Cd.m ... Learn the function cd[m]
```

Second, in textbooks we find differential equations with formula solutions. Thus:

$$\frac{d^2x}{dt^2} = -\omega^2x$$

has the nice solution:

$$x(t) = A \sin(\omega t + \phi)$$

Cases like this are the exception. If the coefficients of your differential equation are slightly tricky, the differential equation must be solved “numerically.” This means that (in our case) the time interval of interest is partitioned into lots of individual instants:  $t_i = i\Delta t$  separated by the fixed time  $\Delta t$ . We seek the position and velocity at those instants:  $x_i = x(t_i)$  and  $v_i = v(t_i)$ . (If we want to know the position or velocity at times in between those instants we must interpolate.) The differential equation can then be used to determine  $x_{i+1}$  and  $v_{i+1}$  based on the earlier position and velocity. The best methods of taking one “step” forward in time are quite sophisticated, but let’s look at a very simple example. Say you want to solve the differential equation:

$$\frac{d^2x}{dt^2} = a(x, v, t)$$

where  $a(x, v, t)$  is some known function. Then:

$$\begin{aligned}x_{i+1} &= x_i + v_i\Delta t \\v_{i+1} &= v_i + a(x_i, v_i, i\Delta t)\Delta t\end{aligned}$$

that is the position is changed according to the current velocity; the velocity is changed according to the current acceleration. In order to start such a process you need to know the initial position and velocity, but of course, you’d also need those initial conditions to solve the differential equation with textbook methods. If  $\Delta t$  is made sufficiently small, the method will reproduce the motion (at least for a while).

*Mathematica* is able to solve a long list of textbook differential equations with the command `DSolve`; it is able to “numerically” approximate the solution to any differential equation using the command: `NDSolve`.

We consider here a  $R = 5$  cm,  $m = 15$  kg shell, with a muzzle velocity,  $v_0$ , of 1000 m/s (approximately Mach 3). The density of air,  $\rho$ , is about: 1.1 kg/m<sup>3</sup>; the speed of sound in air,  $c$  is about: 330 m/s. We assume the effect of the air is only drag: no lift. We seek to solve the differential equation:

$$m\frac{d^2\mathbf{r}}{dt^2} = -\frac{1}{2}C_D(v/c)\rho v\pi R^2\mathbf{v} - mg\mathbf{k}$$

with the initial condition:

$$\begin{aligned}x(0) &= 0 \\z(0) &= 0\end{aligned}$$

$$\begin{aligned}v_x(0) &= v_0 \cos \theta \\v_z(0) &= v_0 \sin \theta\end{aligned}$$

where  $\theta$  is the elevation angle of the gun barrel.

To help you solve the above problem (without doing it for you), consider a slightly different problem:

```
v=10
theta=.7
solution= NDSolve[{
x''[t]== -Sqrt[x'[t]^2+z'[t]^2] x'[t],
z''[t]== -Sqrt[x'[t]^2+z'[t]^2] z'[t] - 1,
  x[0]== 0,
  z[0]== 0,
  x'[0]== v Cos[theta],
  z'[0]== v Sin[theta]},{x,z},{t,0,4}]
```

Notice: the vector equation becomes two coupled equations (N.B.: ==). Those two differential equations, plus the four initial conditions (e.g.,  $x[0]==0$ ), are the six elements of a set (N.B.:  $\{ x''[t]== \dots, z''[t]== \dots$  etc. ) and that set is the first argument of `NDSolve`. The second argument of `NDSolve` is the set of functions you want to solve for. The last argument reports the name of the independent variable and the interval (between 0 and 4) through which you seek a solution.

The slightly different differential equation we have solved with *Mathematica* is:

$$\frac{d^2 \mathbf{r}}{dt^2} = -v \mathbf{v} - \mathbf{k}$$

I hope you can see how the vector differential equation became two coupled differential equations as we wrote  $\mathbf{r}$ ,  $\mathbf{v}$ , and the unit vector  $\mathbf{k}$  in components, and expressed the speed  $v$  in terms of the components of  $\mathbf{v}$ .

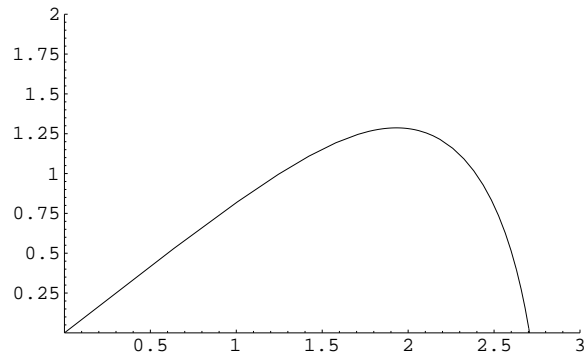
The result of the *Mathematica* commands is:

```
Out[3]= {{x -> InterpolatingFunction[{{0., 4.}}, <>],
  z -> InterpolatingFunction[{{0., 4.}}, <>]}}
```

That is to say *Mathematica* is hiding the complexity of the interpolation that is necessary to make a function out of the series of points it has calculated in solving the differential equation. The trajectory of the shell is the trace of the moving point  $\mathbf{r}(t)$ ;  $t$  is the parameter describing the location of the shell. To plot such a moving point we need to do a `ParametricPlot`, but first we must extract the `InterpolatingFunction` from the solution:

```
ParametricPlot[Evaluate[{x[t],z[t]} /. First[solution]],{t,0,3.2},PlotRange->All]
```

The “/.” says make the substitutions (“->”) that constitute our `First[solution]`. (Our solution actually only has a `First` part, but other equations might have several solutions, for example, the quadratic equation.) The results should look like this:



## 2 Homework

Follow the above method, but use the proper data including the `cd[m]` function. Consider the last two digits of your social security number:  $XZ$ . By varying the elevation of the gun hit the point  $(1X000, Z00)$ . Turn in a plot of the result.

Also turn in a printout showing each step as *Mathematica* solves the problem.

*Extra Credit:* The density of air decreases as you go up:

$$\rho = \rho_0 e^{-z/H}$$

where the scale height  $H \approx 8000$  m. Solve the motion of the shell in the thinning air and produce a plot that shows how the changing density of air affects the trajectory.