Example of the Subtleties of Floating Point Arithmetic

Lucas Bouck

I was interested in plotting the solution of the following boundary value problem:

$$\begin{aligned} -u'' + bu + u &= 2x \text{ in } (0,1), \\ u(0) = u(1) &= 0 \end{aligned}$$

The solution is

$$\begin{aligned} u(x) &= c_1 e^{r_+ x} + c_2 e^{r_- x} + 2x - 2b, \text{ with} \quad (1) \\ r_\pm &= \frac{b \pm \sqrt{b^2+4}}{2}, \; c_2 = \frac{2b - 2 - 2be^{r_+}}{e^{r_-} - e^{r_+}}, \quad \text{and } c_1 = 2b - c_2 \end{aligned}$$

I was interested in the case where b is large, specifically $b = 100$.

The first attempt at implementing this function in MATLAB was to naively implement the expression in (1). Plotting the solution for $b = 0$ and $b = 100$ gives

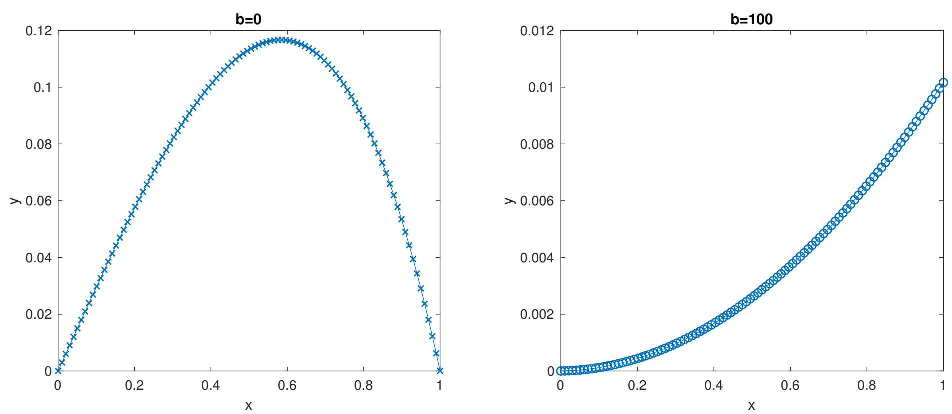

Figure 1: Plots of the our exact solution with the first and faulty implementation.

We can see that the boundary conditions are satisfied for $b = 0$, but our implementation totally misses the right boundary condition for $b = 100$. If we do a very simple error analysis with $\texttt{fl}(x)$ being the floating point answer of x, we get

$$\begin{aligned} \texttt{fl}\,(u(x)) &= c_1 e^{r_+ x}(1 + \varepsilon_+) + c_2 e^{r_- x}(1 + \varepsilon_-) + (2x - 2b)(1 + \varepsilon) \\ &= c_1 e^{r_+ x} + c_2 e^{r_- x} + 2x - 2b + \varepsilon_+ c_1 e^{r_+ x} + \varepsilon_- c_2 e^{r_- x} + \varepsilon(2x - 2b) \\ &= u(x) + \varepsilon_+ c_1 e^{r_+ x} + \varepsilon_- c_2 e^{r_- x} + \varepsilon(2x - 2b). \end{aligned}$$

The values $\varepsilon_+, \varepsilon_-$, and ε are the relative errors of computing $c_1 e^{r_+ x}, c_2 e^{r_- x}$ and $2x - 2b$ respectively. Since our boundary condition is $u(1) = 0$, it makes more sense to now consider the absolute error. If $b = 0$ and each relative error is around 10^{-15}, then the absolute error of computing $u(1)$ is around 10^{-14} or 10^{-13}. If $b = 100$, then the term $c_1 e^{r_+} \approx 10^{43}$ and will dominate the absolute error compared to the other terms. Assuming the relative error ε_+ is around machine precision, then the absolute error of computing $u(1)$ will be quite large.

How can we implement the exact solution and avoid the large absolute errors? My idea was to avoid computations like $a + b$ where a is much larger than b. If there are relative errors in computing a and b, the absolute error of computing $a + b$ is $a\varepsilon_a + b\varepsilon_b$. If $a >> b$ then $a\varepsilon_a$ will be the dominant term in the absolute error. One rearrangement of the terms of our exact solution is

$$u(x) = 2b\frac{e^{r_+x+r_-} - e^{r_-x+r_+}}{e^{r_-} - e^{r_+}} - 2b\left(1 - \frac{e^{r_-x} - e^{r_+x}}{e^{r_-} - e^{r_+}}\right) + 2\left(x - \frac{e^{r_-x} - e^{r_+x}}{e^{r_-} - e^{r_+}}\right) \tag{2}$$

This implementation that comes from (2) precisely tries to avoid the issues listed previously. For $b = 100$, we can see the improvement.

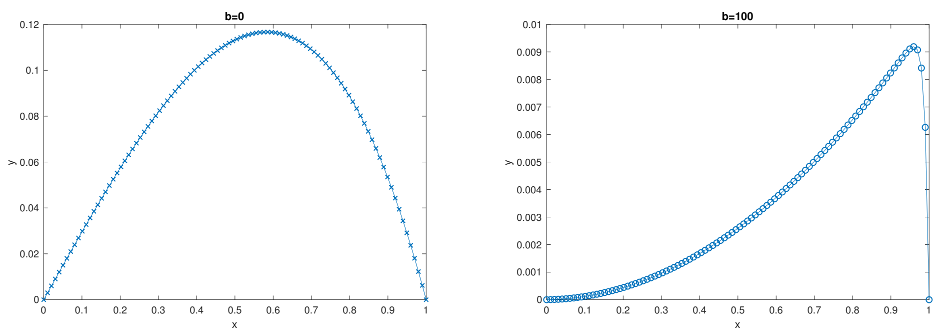

Figure 2: Plots of the our exact solution with the second implementation.

The final figure gives a comparison of the two implementations.

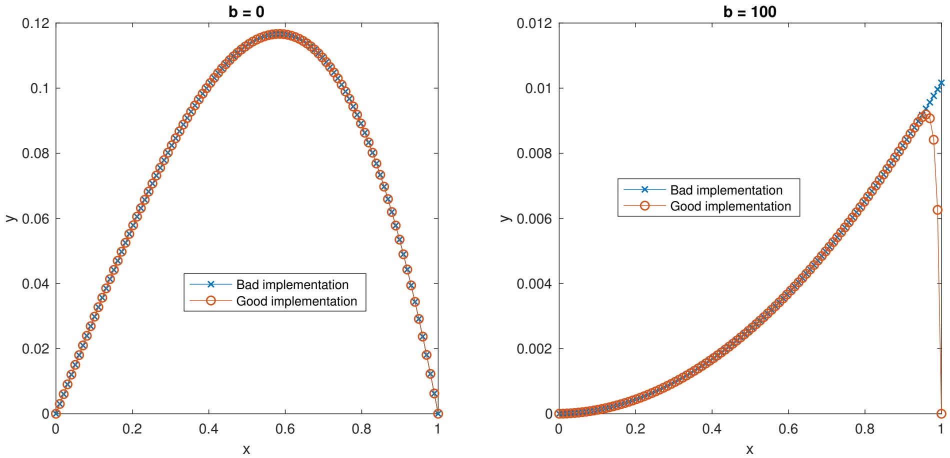

Figure 3: Comparisons of the first and second implementations for $b = 0$ and $b = 100$.