A Gentle Introduction to Computational Economics

Wouter J. Denhaan, Pietro Garibaldi, and Pontus Rendahl

February 28, 2018

Chapter 3

Nonlinear equations

3.1 Introduction

You can probably solve the equation $e^x = 5$, as it amounts to calculate $\ln(5)$ which even the most elementary calculator will do for you. But you will struggle with the seemingly simple extension $e^x + x = 5$.[1] The reason is that while the former has a closed-form solution, the latter does not.[2] Perhaps you will be surprised – and a bit disappointed – to learn that the set of equations that does *not* admit a closed-form solutions widely exceeds the set that does. And why wouldn't you? After all, almost all mathematics in both primary, secondary, and (parts of) higher education focus on equations that allows for a (often challenging) closed-form solution. But this was intentional; indeed, the courses have been designed to focus on such equations for the purpose of learning. But after learning comes doing, and the purpose of this chapter is teach you how to use the computer to solve functions that do not allow for a closed-form solution.

To get an idea of the underlying principles, let us go back to the previous equation, $e^x + x = 5$. Why not guess for an answer? The guess 1.61 is probably on the high end, as we know that $e^{1.61} \approx 5$. So pick something lower, say 1.2. A simple calculation shows that $e^{1.2} + 1.2 \approx 4.52$. That is not too bad, but we are still quite off the mark. How, about 1.3? Checking we find that $e^{1.3} + 1.3 \approx 4.97$, so we are making some progress here. But guessing and adjusting is a time-consuming, boring, and inefficient approach to finding the solution to our equation. It would be much nicer to outsource such a task to an unemotional and reliable friend who works for free; your computer.

In a nutshell, the science of numerically solving nonlinear equation actually relies on the simple procedure above: make a guess for x, call this guess x_0, updates the guess in some *systematic and efficient* way to find a new "guess" x_1, and keep on repeating (iterating) until the equation is solved. If this sounds simple to you, it is because it is! Well, at least in theory, as we will see it is not always simple in practice. And already at this point one caveat ought to be brought; an equation can never really be numerically "solved", but only solved up to a tolerable level of error. And what constitutes as tolerable involves judgement of the problem at hand. For instance, if a function is such that $f(1) = 10,000$ and $f(-1) = -10,000$, and you wish to find an x such that $f(x) = 0$, then a tolerance level of $1e(-3)$ may well do; but if the function instead delivers $f(1) = 1e(-8)$ and

[1]The solution to the first equation is $x = \ln(5) \approx 1.61$

[2]A closed-form solution is vaguely defined as a solution to an equation that can be expressed as relatively elementary mathematical operations on the functions parameters.

$f(-1) = -1e(-8)$, you may want to go for a tolerance level of $1e(-14)$ or lower.[3] Furthermore, the tolerance level ought also depend on the consequences of the error. If, for instance, you wish to launch a rocket from Pyongyang to the United States, and an error of $1e(-6)$ runs risk of the rocket ending up in your backyard, then perhaps you should consider tighten the tolerance criterion. In any case, tighter tolerance is related with precision, but also with the time involved in the computation, so there is always a tradeoff – a tradeoff that boils down to judgement and knowledge of the problem at hand.

Lastly, why do you *need* to know how to solve nonlinear equations? The answer is that they appear everywhere in economics, ranging from first order conditions to market clearing conditions; from maximum likelihood estimation to nonlinear least squares, etc. Linear (or log-linear) approaches were only the beginning – nonlinearities are where the action happens. Let's get on with the action.

3.2 Fixed-point iteration

The common notation for solving a nonlinear equation is to find x such that $f(x) = 0$. In the example used in the introduction, $e^x + x = 5$, would therefore be rewritten as $e^x + x - 5 = 0$. One of the foundations of solving nonlinear equations is known as *fixed-point iteration*. The method – which is very broad, and encompasses many of the approaches outlined below – is based on the idea that we can rearrange our equation as $x = h(x)$. Fixed point iteration then proceeds as $x_{n+1} = h(x_n)$, until $\|x_n - h(x_n)\| < \varepsilon$.[4]

In our example, this could amount to the iterative procedure $x_{n+1} = 5 - e^{x_n}$. Testing this out using an initial guess of 1.2 we quickly get stuck in an oscillation of $x_{n+1} = 5$ and $x_n = -143.41$. This oscillation does not depend on the initial guess, but emerges for almost any guess, and appears to be an inherent property of the equation. Alternatively, however, we can instead try the iteration $x_{n+1} = \ln(5 - x_n)$. This, on the other hand, robustly converges to the solution $x = 1.3066$ for any initial x_0 between the oscillation-points 5 and -143.41.

What underlies these diverging results? Let x^* denote the fixed point $x^* = h(x^*)$. Using a first-order Taylor approximation of $h(x_n)$ around x^* we find that

$$h(x_n) \approx h(x^*) + h'(x^*)(x_n - x^*),$$

where $h'(\cdot)$ denotes the derivative of $h(\cdot)$. Using the fact that $x_{n+1} = h(x_n)$ leads to

$$(x_{n+1} - x^*) \approx h'(x^*)(x_n - x^*).$$

[3] $1e(-3)$ is scientific notation for $1 \times 10^{-3} = 0.001$, and more generally $xe(-y) = x \times 10^{-y}$. Double precision numbers, the most common forms used on most computers, has a machine accuracy of about $1e(-16)$; anything smaller will be interpreted as a zero.

In 2007, the CFO of Goldman Sachs, David Viniar, famously said: "We were seeing things that were 25-standard deviation moves, several days in a row". Dowd, Cotter, Humphrey, and Woods (2008) attempted to compute the probability of such an event, but quickly ran into machine accuracy problems. Using a clever workaround, they found that a seven standard deviation event is so unlikely, that its expected occurrence is approximately once in a period that is five times greater than the time elapsed since multicellular life evolved on this planet (about 600 million years ago). A 20 standard deviation even is so rare that its expected occurrence would be once in every $1.453e(86)$ years; a number that is ten times larger than the higher estimate of the number of particles in the universe. A 25 standard deviation event is that number but with the decimal point moved 52 places to the left.

[4] The notation $\|\cdot\|$ refers to a *norm*, which provides a measure of distance between two element. For this section, it is sufficient to think of a norm as simply the absolute value of a difference, or possibly the square of the difference. Chapter XXX will provide a more detailed discussion about norms in more general terms.

Thus, for the above mapping to be (locally) stable, it must be the case that $|h'(x^*)| < 1$. If we calculate $|h'(x^*)|$ for the two iterative schemes above we find that $|h'(x^*)| \approx 3.69$ in the first case, and $|h'(x^*)| \approx 0.27$ in the second. These results indicate that the first iteration scheme is locally unstable, and for *any guess*, apart from the solution itself, divergence will appear. The latter scheme, however, was locally stable, and all initial guess within a broad interval converge to the true solution.

While flimsy, this type of unsystematic approach to solving an equation should not be completely discarded. Indeed, when it works, it is usually very fast as it relies on simple evaluations of (parts of) the function describing the equation. In addition, issues of non-convergence can sometimes be "fixed" by the iteration

$$x_{n+1} = (1-\rho)x_n + \rho h(x_n)$$

for some $\rho \in (0,1)$. That is, x_n is updated "slowly". It is therefore not a bad idea to try out some simple/unsystematic fixed-point iteration, in case your problem has nice convergent properties.[5]

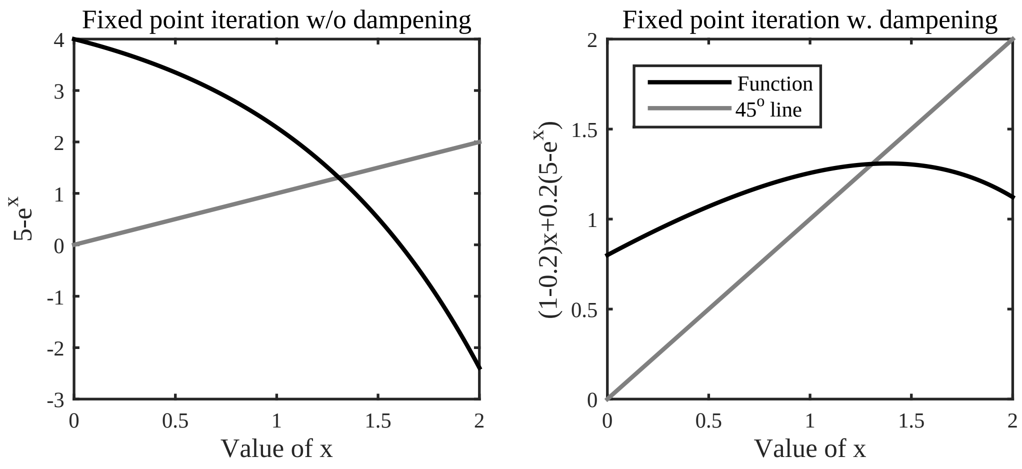

Figure 3.1: Fixed point iteration with and without dampening.

3.3 Newton's method (and friends)

Newton's method is quite straightforward, and relies on a first-order Taylor approximations to successively update guesses. In particular, given an initial guess for the solution x, call this x_0, Newton's method uses the approximation

$$f(x) \approx \hat{f}(x) = f(x_0) + f'(x_0)(x - x_0).$$

The main idea is then to proceed and update the guess by finding an x such that $\hat{f}(x) = 0$. That is,

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)},$$

[5]Indeed, for $\rho < 0.4264$, local stability is ensured and the iteration $x_{n+1} = (1-\rho)x_n + \rho(5 - e^{x_n})$ converges quickly to the true solution.

and more generally

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \tag{3.1}$$

until $\|f(x_n)\| < \varepsilon$. Newton's method does not require that the first order Taylor approximation is particularly "good", as it simply uses the information obtained from it to decide in which direction, and with which step-length, the guess ought to be updated.

If we apply this to the equation $e^x + x - 5 = 0$, with the initial guess $x_0 = 1.2$, we find that

$$e^x + x - 5 \approx e^{1.2} + 1.2 - 5 + (e^{1.2} + 1)(x_1 - 1.2)$$

or simply

$$\begin{aligned} x_1 &= 1.2 - \frac{-0.48}{4.32}, \\ &= 1.31, \end{aligned}$$

with an error of 0.021. Repeating again yields $x_2 = 1.3066$, with an error of $3.8e(-5)$. A third repetition delivers an error of $1.2e(-10)$, and we can practically consider the equation solved.

Figure 3.2 visualizes Newton's method for the quadratic equation $f(x) = 50 + x - x^2$ on the interval $[0, 19]$ and with an initial guess $x_0 = 2$. The upper left quadrant shows the function itself (grey line), the initial guess (black dot), the first order Taylor approximation of $f(x)$ from x_0 to x_1 (dashed line), as well as the new guess x_1 (grey dot). The upper right quadrant shows the analogues to the previous quadrant, but including the Taylor approximation at x_1 alongside with the new guess, x_2. The lower left quadrant shows the third iteration, and the lower right quadrant the function values for the first ten consecutive iterations. The function error at the tenth iteration equals $7e(-15)$.

The performance of Newton's method are in both the above cases impressive; it converges fast and reliably.[6] However, do not let these examples lull you into a false sense of security. Newton's method is not guaranteed to converge in all settings, and can wander off into oscillating patterns, or into imaginary territory. In addition, for non-differentiable problems, it is not even a feasible approach; and for values of x_n such that $f'(x_n)$=0, you can easily see that you are doomed from the very beginning. A "good" initial guess is a *prerequisite* for Newton's method to behave well, and only knowledge of the underlying problem can provide this crucial information. Additionally, an observant reader would have perhaps have noticed that the equation $50 + x - x^2 = 0$ indeed has two solutions (as all quadratic equations do). Newton's method found one of these, but with an initial guess of, for instance, $x_0 = -2$, would have found the other. Thus, it is often useful, and good practice, to solve your problem for several initial guesses in order to gain some confidence in ruling out multiplicity of solutions. Nevertheless, having said that Newton's method is many computational economists' go-to method, and has a very wide range of applications.

[6]Newton's method is guaranteed to converge for quadratic functions, and is indeed used by many calculators to compute the square root of a number, since the solution to, for instance, $x^2 - 262 = 0$, equals the square root of 262.

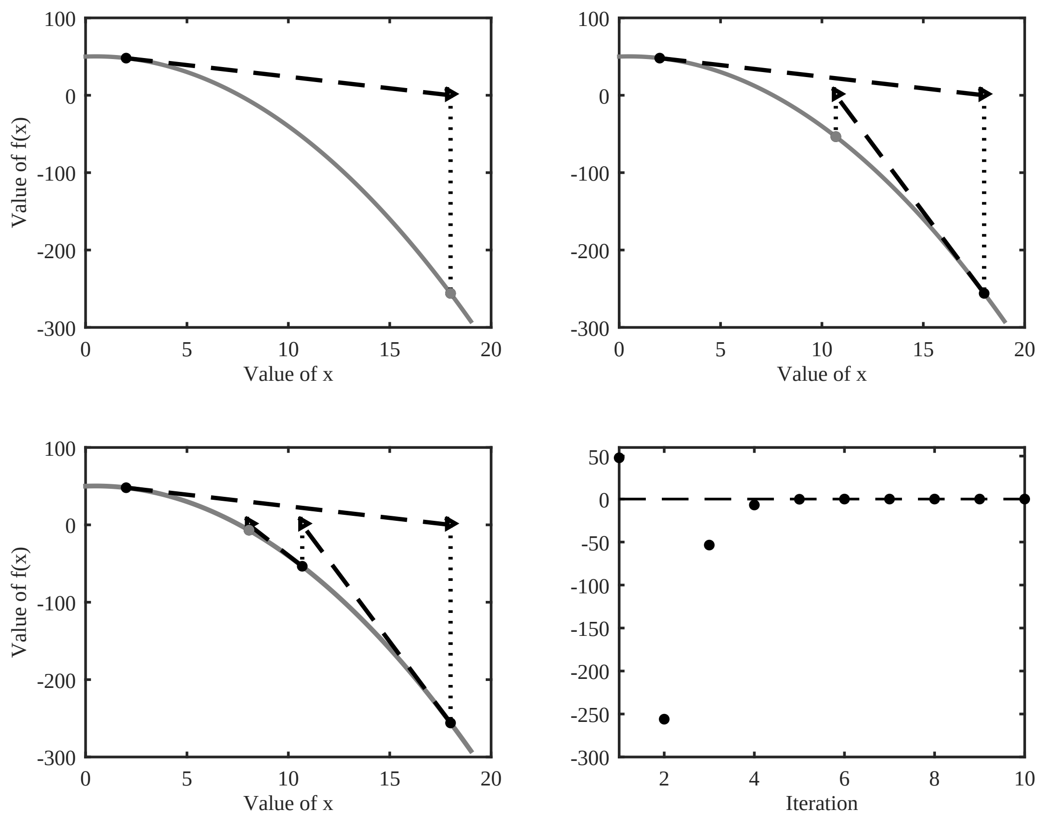

Figure 3.2: Newton's method at work, $f(x) = 50 + x - x^2$.

Box 3.1: Pseudo-code to solve a univariate problem using Newton's method.

```
1. Set initial guess x0, Tol=1e-8, and metric=1.

2. Evaluate the function f0=f(x0), and its derivative df0=f'(x0).

3. Find xn as

   xn=x0-f0/df0.

4. while metric>Tol

     • fn=f(xn),
     • dfn=f'(xn).
     • xn1=xn-fn/dfn.
     • xn=xn1, metric=abs(fn).

   end
```

3.3.1 The secant method

As mentioned above Newton's method requires the calculation of a derivative. This can sometimes be costly, and tends to slow the algorithm down. A closely related alternative to Newton's method is called the "secant method", and belongs to a broad class of nonlinear equation solvers known as "quasi-Newton methods". As the name suggests, the secant method relies on line secants as approximations of derivatives. As a consequence, the secant method is derivative-free, but relies on two initial guess, instead of one; the first to evaluate the function, and the second to form a line secant.

Let us revisit the equation that started this chapter, $e^x + x - 5 = 0$, and suppose we have two guesses $x_1 = 1$ and $x_2 = 1.2$. The function values at these points are approximately -1.28 and -0.48 for x_1 and x_2 respectively. The line secant

$$\frac{f(x_2) - f(x_1)}{x_2 - x_1} \approx 4.01.$$

then forms a useful approximation for the derivative of the function. Thus, as with Newton's method we will approximate $f(x)$ using a first order Taylor expansion, but with the derivative replaced by the line secant. That is,

$$f(x) \approx -0.48 + 4.01 \times (x - 1.2).$$

Solving for x gives us an updated guess x_2 as

$$\begin{aligned} x_2 &= x_1 - \frac{-0.48}{4.01} \\ &= 1.32. \end{aligned}$$

Repeating this procedure gives $x_3 = 1.3060$, with an error of 2.6e(-2). A third repetition delivers and error of $1.3e(-5)$, and a fourth of $2.9e(-9)$, and the equation is again solved. Notice, however, that compared to Newton's method we needed one more iteration to reach the same level of accuracy. On the other hand, we did not need compute any derivatives. The question then arises, what was more costly; evaluating the function *and* calculating its derivative three times, *or* evaluating the function only but five times?

Figure 3.3 provides a visualization of the secant method analogous to that in Figure 3.2. As can be seen from lower right quadrant, convergence is slightly slower than for Newton's method, and behaves somewhat more erratically. Thus, the cost of calculating the derivative will determine which method to use; if it is relatively costless, Newton's method is to be preferred.

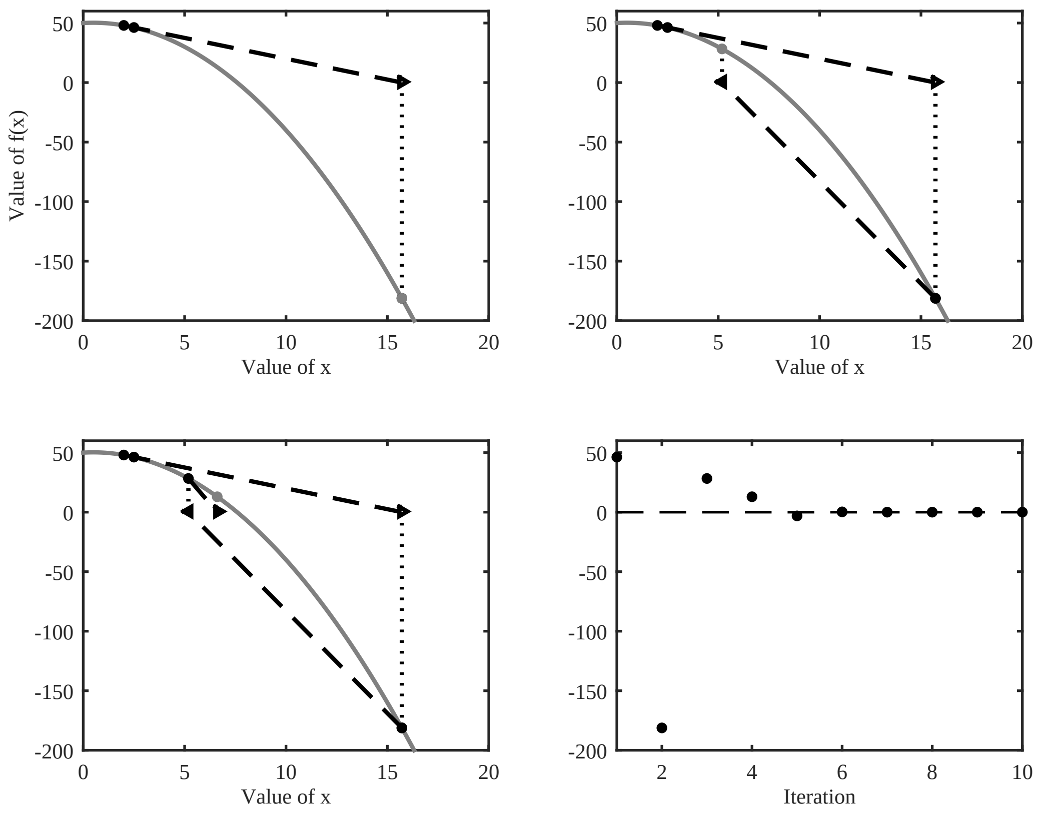

Figure 3.3: The secant method at work, $f(x) = 50 + x - x^2$.

Box 3.2: Pseudo-code to solve a univariate problem using the Secant method.

```
1. Set x0 and x1, Tol=1e-8, and metric=1.

2. Evaluate the function f0=f(x0) and f1=f(x1).

3. Calculate the secant

   df1=(f1-f0)/(x1-x0).

4. Find xn2 as

   xn2=x1-f1/df1.

5. Set xn=xn1 and xn1=xn2, fn=f1.

6. while metric>Tol

     • fn1=f(xn2).
     • dfn=(fn1-fn)/(xn1-xn).
     • xn2=xn1-fn1/dfn.
     • xn=xn1, xn1=xn2, fn=fn1, and metric=abs(fn).

   end
```

3.3.2 Multivariate problems

In many economic problems we are not only interested in solving one equation in one variable, but rather a system of n (simultaneous) equations in equally many variables. Fortunately, the ideas underpinning Newton's method easily generalizes to systems of equations; and there are similar generalizations for the secant method (although they are not as transparent).

Consider the system of equations

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0},$$

where $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)'$ is an $n \times 1$ vector of variables; $\boldsymbol{f}(\boldsymbol{x}) = (f_1(x_1, x_2, \ldots, x_n), \ldots, f_n(x_1, x_2, \ldots, x_n))'$ is an $n \times 1$ vector of functions each using n arguments (or less); and $\boldsymbol{0} = (0, 0, \ldots)$ is an $n \times 1$ vector of zeros.

As in the univariate case, Newton's method relies on the Taylor expansion

$$\boldsymbol{f}(\boldsymbol{x}) \approx \hat{\boldsymbol{f}}(\boldsymbol{x}) = \boldsymbol{f}(\boldsymbol{x_0}) + \boldsymbol{J}_{x_0}(\boldsymbol{x} - \boldsymbol{x_0}),$$

where $\boldsymbol{J}_{x_0}$ denotes the *Jacobian* of $\boldsymbol{f}(\boldsymbol{x})$ at $\boldsymbol{x_0}$. That is the $n \times n$ matrix

$$\boldsymbol{J}_x = \begin{pmatrix} \frac{\partial f_1(x)}{\partial x_1} & \frac{\partial f_1(x)}{\partial x_2} & \cdots & \frac{\partial f_1(x)}{\partial x_n} \\ \frac{\partial f_2(x)}{\partial x_1} & \ddots & & \frac{\partial f_2(x)}{\partial x_n} \\ \vdots & & \ddots & \vdots \\ \frac{\partial f_n(x)}{\partial x_1} & \frac{\partial f_n(x)}{\partial x_2} & \cdots & \frac{\partial f_n(x)}{\partial x_n} \end{pmatrix}.$$

As in the univariate case, Newton's method updates an initial guess, $\boldsymbol{x_0}$, by solving the Taylor approximation of $\boldsymbol{f}(\boldsymbol{x})$ around the guess, such that

$$\boldsymbol{x_1} = \boldsymbol{x_0} - \boldsymbol{J}_{x_0}^{-1}\boldsymbol{f}(\boldsymbol{x_0}),$$

and proceeds as

$$\boldsymbol{x}_{n+1} = \boldsymbol{x_n} - \boldsymbol{J}_{x_n}^{-1}\boldsymbol{f}(\boldsymbol{x_n}),$$

until $\|\boldsymbol{f}(\boldsymbol{x_n})\| < \varepsilon$.

To give an example, consider the system of equations

$$\begin{aligned} 0 &= 50 + y - x^2 \\ 0 &= e^y + x - 5. \end{aligned}$$

With an initial guess of $x_0 = 2$ and $y_0 = 1.2$, the Taylor approximation is given by

$$\hat{\boldsymbol{f}}(\boldsymbol{x}) = \begin{pmatrix} 47.2 \\ 0.32 \end{pmatrix} + \begin{pmatrix} -4 & 1 \\ 1 & 3.32 \end{pmatrix} \begin{pmatrix} x - 2 \\ y - 1.2 \end{pmatrix}.$$

Solving for x and y gives

$$\begin{aligned} \begin{pmatrix} x \\ y \end{pmatrix} &= \begin{pmatrix} 2 \\ 1.2 \end{pmatrix} - \begin{pmatrix} -0.23 & 0.07 \\ 0.07 & 0.28 \end{pmatrix} \begin{pmatrix} 47.2 \\ 0.32 \end{pmatrix} \\ &= \begin{pmatrix} 12.95 \\ -2.19 \end{pmatrix}. \end{aligned}$$

After four iterations we find $x = 5$ and $y = -25$, with a maximum absolute error of $1.4e(-10)$.

If, however, we used the initial guess of $x_0 = -2$ and $y_0 = 1.2$, the solution is instead given by $x = -7.25$ and $y = 2.51$. Having randomized for the initial guesses several times, we conclude that these are the only two solutions to the system of equations (although we can never be sure of this).

The main drawback with Newton's method for multivariate problems is the potentially very high cost associated with computing the Jacobian. Generalizations of the secant method to higher dimensions is therefore often preferred. However, since the $n \times 1$ vector $\boldsymbol{f}(\boldsymbol{x}_{n+1}) - \boldsymbol{f}(\boldsymbol{x_n})$ only provides partial information of the Jacobian, these methods are quite intricate, and the mathematics involved goes beyond the scope and level of this book. Nevertheless, there are quite useful "tricks" one can adopt to obtain the Jacobian relatively efficiently. For instance, if a closed-form expression of the Jacobian can be obtained, we can code a function that takes a vector $\boldsymbol{x}$ as inputs and delivers the $n \times n$ Jacobian as outputs; this is unlikely to be a costly procedure as we only need to "calculate" the Jacobian once, and subsequently merely evaluate the resulting function. In addition, many softwares allows for symbolic calculations, which can then be used to find the closed-form expression for the Jacobian. Lastly, with recent advances in computational methods, such as autodifferentiation, the cost of calculating a Jacobian can be kept at a minimum.

The pseudocode for the multivariate Newton's method is the same as for the univariate case, with *function* replaced by *system of functions*, and *derivative* with *Jacobian*. In addition, the metric must now be updated as `metric=max(abs(f(xn)))`.

3.4 The bisection method

As previously mentioned, Newton's method and its variations are not guaranteed to converge to a solution, *even* if such a solution is known to exist. Fortunately, there are methods for univariate problems that are guaranteed to converge under some quite weak assumptions. These methods can be incredibly useful when it comes to, for instance, finding an equilibrium price in a nonlinear economic model. The most commonly used approach goes under the name *the bisection method*.

The idea behind the bisection method is straightforward. Suppose that $f(x)$ is continuous and that $f(x_a)$ and $f(x_b)$ are of opposite sign. By the intermediate value theorem, these assumptions guarantee that there exist some x inbetween x_a and x_b such that $f(x) = 0$.[7] The bisection method then proposes to try out the midpoint value

$$x_c = \frac{x_a + x_b}{2}.$$

Now if $f(x_c)$ has the opposite sign to $f(x_a)$, then we set $x_b = x_c$, and otherwise $x_a = x_c$. Repeating this procedure ensures convergence towards a point such that $f(x) \approx 0$, as the function at the bracketing points are always of opposite sign. Take, for instance, the very simple equation $f(x) = e^x - e^{2.2} = 0$. Obviously, we know the answer to this problem ($x = 2.2$), but that just serves to highlight the underlying idea. We also know that that for $x_a = 0$ and $x_b = 4$, the values of $f(x_a)$ and $f(x_b)$ are of opposite sign. Trying out $x_c = 2$ gives $f(x_c) = -1.64$. Since $f(x_b)$ and $f(x_c)$ have opposite sign, we set $x_a = 2$. Proceeding suggests a new value for $x_c = 3$, which results in $f(x_c) = 11.06$. As a consequence, we set $x_b = 3$, etc.

Figure 3.4 illustrates the properties of this procedure. The black lines in the left panel show the interval $[x_a, x_b]$ for the first ten iterations. The grey dots illustrates the midpoints. The right panel shows the function values at the midpoints as dots when positive and as a cross when negative. Each time a cross is shown the lower bound of the interval in the left panel changes to the previous midpoint, and each time a dot is shown the upper bound of the interval instead changes to the previous midpoint. After ten iterations the value of the function is $-7e(2)$, and it takes roughly 25 iterations to reach a tolerable error of about $1e-(7)$.[8]

[7] Furthermore, if $f(\cdot)$ is monotone there exist only one such x, and the solution is unique.

[8] With an initial guess of four, it requires only six iterations for Newton's method to reach an error less than $1e-(11)$; and 11 iteration with an initial guess of zero.

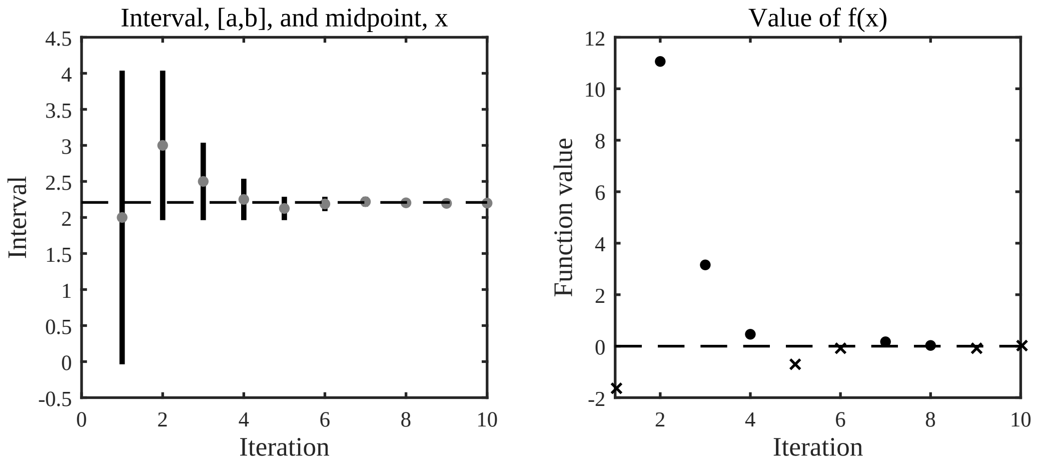

Figure 3.4: The bisection method at work, $f(x) = e^x - e^{2.2}$.

Box 3.3: Pseudo-code to solve a univariate problem using the Bisection method.

```
1. Set xa and xb, Tol=1e-8, and metric=1.

2. Verify that f(xa)f(xb)<0.

3. While metric>Tol

     • xc=(xa+xb)/2.

     • If f(xc)f(xb)<0, xa=xc.  Else, xb=xc

     • metric=abs(fxc)

   end
```

3.4.1 Brent-Dekker method

The slow convergence of the bisection method boils down to its somewhat mindless choice of midpoint. There are ways to improve this. Consider the inverse linear interpolation[9]

$$x = x_b\left(1 - \frac{f - f(x_b)}{f(x_b) - f(x_a)}\right) + x_a\left(\frac{f - f(x_b)}{f(x_b) - f(x_a)}\right).$$

Setting f to zero and simplifying yields

$$x_c = x_b - \frac{x_b - x_a}{f(x_b) - f(x_a)} f(x_b).$$

[9]This is known as inverse interpolation, as normally when we interpolate we find an approximation $\hat{f}(x)$, while for inverse interpolation we find $\hat{x}(f)$.

If x_c belongs to the interval formed by x_a and x_c, we proceed with the regular bisection update. If x_c falls outside of the interval, we proceed using the midpoint instead. This procedure is known as Dekker's method, and provides a faster way of solving the equation than the regular bisection method. Notice that this is indeed the secant method used in conjunction with the bisection method.

Brent's method extends Dekker's method in a couple of ways. First, Brent's method suggests to use inverse quadratic interpolation instead of inverse linear (and revert to linear if the suggested value of x_c falls outside of the interval). Second, he proposes criteria under which new values of x_c actually improves the function values fast enough, and suggests to use the midpoint if these are not met. While the precise details of Brent's method are somewhat involved, it is the most common solver used by pre-canned routines, such as `fzero` in Matlab.

3.5 Pathologies, anomalies, and some remedies

Newton's method and the related quasi-Newton methods are not fool-proof, and convergence is only guaranteed if the initial guess is "sufficiently close to" the solution x^*.[10] The purpose of this section is to explore some cases in which Newton's method fails, and what can be done to avoid this from happening.

The solid black line in the left graph of Figure 3.5 illustrates the equation $x^3 - 2x + 2$. The dashed lines show the first order Taylor approximations at $x = 0$ and $x = 1$. As you can see from the graph, and initial guess of zero, yields an "updated" guess of one, which then yields a new update of zero, etc. Hence, Newton's method falls into an oscillating pattern between zero and one.

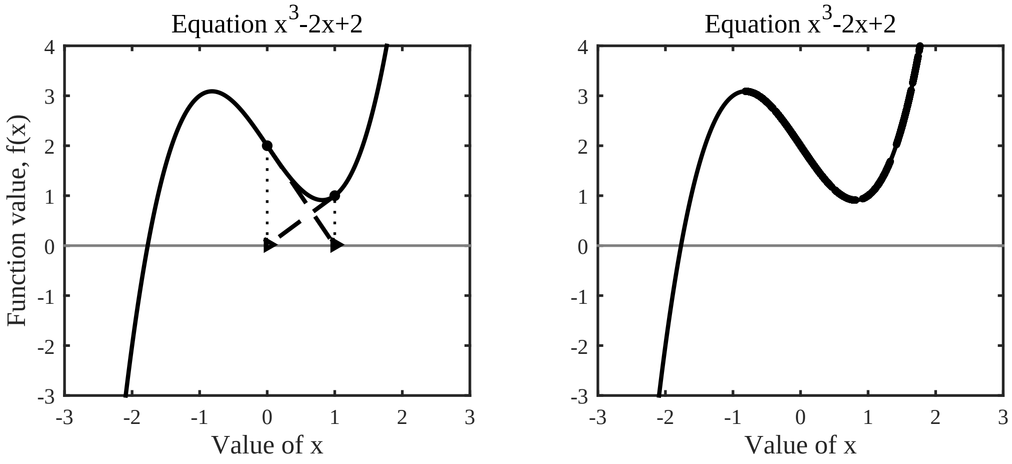

Figure 3.5: Newton's method for a pathological example.

The thick line of the right graph of Figure 3.5 shows all initial values of x such that Newton's method eventually falls into the oscillating pattern above. As is clear from the graph, there is quite a wide range of initial values such that Newton's method fails. Our own experience is that oscillations do not frequently occur for economic problems, but they occur frequently enough that you need some weapons in your arsenal to combat them if they appear.

[10]"Sufficiently close to" has a precise mathematical meaning. But the conditions under which this is satisfied is not possible to evaluate without knowing the solution x^*. Hence, for practical purposes these conditions are not particularly useful, but knowing that for some x_0 convergence is guaranteed is indeed informative.

3.5.1 Dampening

The main issue with Newton's method is that the "improvement steps" can be to large. Thus, as in Section 3.2, we can dampen the step size by simply updating the successive guesses slowly. In particular, let $\hat{x}_{n+1}$ be the suggested Newton step from the past guess of x_n. A simple procedure would then be to update according to

$$x_{n+1} = \rho \hat{x}_{n+1} + (1-\rho)x_n.,$$

for some $\rho \in (0,1)$. However, inspecting the univariate Newton step in equation (3.1) reveals that

$$\begin{aligned} x_{n+1} &= \rho \hat{x}_{n+1} + (1-\rho)x_n, \\ &= \rho(x_n - \frac{f(x_n)}{f'(x_n)}) + (1-\rho)x_n, \\ &= x_n - \rho\frac{f(x_n)}{f'(x_n)}. \end{aligned}$$

Thus, dampening amounts to a trivial modification of Newton's method that can very well remedy any problematic behavior encountered; indeed, setting $\rho = 0.9$, entirely removes every single pathological initial guess in Figure 3.5.

The procedure of dampening extends easily to multivariate cases, and is straightforward to implement. Nevertheless, sometimes ρ might need to take a value close to zero, which can increase the number of iterations and thereby slow down the procedure considerably. In addition, once x_n has reached a value "close to" the solution x^*, dampening may be unnecessary altogether, and a more efficient procedure would allow ρ to change as we are getting closer to the solution. Backtracking, which is the topic of the next subsection, resolves these issues.

3.5.2 Backtracking line search

Recall from equation (3.1) that a full "Newton step" is given by

$$\begin{aligned} x_{n+1} &= x_n - \frac{f(x_n)}{f'(x_n)}, \\ \text{or} \quad \Delta x_n &= -\frac{f(x_n)}{f'(x_n)}. \end{aligned}$$

A backtracking line search attempts to find an $\alpha \in (0,1)$ such that the update $x_{n+1} = x_n + \alpha\Delta x_n$ is guaranteed to yield an *improvement* relative to x_n.

To get an idea what an improvement is, notice that solving $f(x) = 0$ is akin to finding an x^* that satisfies the first order condition to the minimization problem

$$\min_x \frac{1}{2} f(x)^2.$$

Thus, we would like to set α such that, at the very least,

$$\frac{1}{2}f(x_n+\alpha\Delta x_n)^2 < \frac{1}{2}f(x_n)^2.$$

In principle, we could search for $\alpha \in (0,1)$ until the above condition holds, but, as usual, there are ways to be more systematic and efficient about this.

What would constitute a reasonable improvement? Consider the Taylor approximation

$$\frac{1}{2}f(x_{n+1})^2 \approx \frac{1}{2}f(x_n)^2 + f(x_n)f'(x_n)(x_{n+1}-x_n).$$

Using the fact that $x_{n+1} = x_n + \alpha\Delta x_n$ and that $\Delta x_n = -f(x_n)/f'(x_n)$ yields

$$\frac{1}{2}f(x_n+\alpha\Delta x_n)^2 \approx \frac{1}{2}f(x_n)^2 - f(x_n)^2\alpha.$$

or

$$\frac{1}{2}[f(x_n)^2 - f(x_n+\alpha\Delta x_n)^2] \approx f(x_n)^2\alpha.$$

Thus, a useful value for α would be such that

$$\frac{1}{2}[f(x_n)^2 - f(x_n+\alpha\Delta x_n)^2] \geq c\alpha f(x_n)^2. \tag{3.2}$$

for some parameter $c \in (0,1)$.

The backtracking line search usually starts with $\alpha_1 = 1$, and evaluates if the condition in equation (3.2) is satisfied. If it is, x_{n+1} is updated using a full Newton step. If it is not, we set $\alpha_2 = \tau\alpha_1$, for $\tau \in (0,1)$, and again evaluate equation (3.2). This process continues until α_j is such that equation (3.2) holds, and then sets $x_{n+1} = x_n + \alpha_j\Delta x_n$.

Ensuring that the solution method is robust to "bad" initial guesses is not the only benefit of backtracking line search, it can also increases efficiency quite a bit. For instance, for the problem illustrated in Figure 3.2, the first step is far to large, and the equation at the updated guess is even further away from zero than at the initial guess. The fact that we overshot causes excessive iterations, and slows down the algorithm. Using a value of $c = 0.75$ solves the problem in three, instead of seven, iterations, and at accuracy of 1e(-15) (which is smaller than the tolerance criterion of $1e(-8)$). Furthermore, the real benefits of backtracking line search really kicks in when solving large(r) scale systems, as α is still just one parameter, and the backtrack only involves relatively costless equation evaluations.

Box 3.4: Pseudo-code to solve an equation with backtrack line search.

```
1. Set x0, Tol=1e-8, c=0.5, tau=0.9, and metric=1.

2. Evaluate the function f0=f(x0) and the derivative df0=f'(x0).

3. dx0=x0-f0/df0, check=0, a=1.

4. while check == 0

     • F=f(x0+a*dx0)

          if 1/2*(f0^2-F^2)-c*a*f0^2>0

          check=1, fn=F, xn=x0+a*dx0

          else a=tau*a

          end

   end

5. while metric>Tol

     • dfn=f'(xn).

     • dxn=xn-fn/dfn, check=0, a=1.

     • while check == 0

        – F=f(xn+a*dxn)

             if 1/2*(fn^2-F^2)-c*a*fn^2>0

             check=1, fn=F, xn=xn+a*dxn

             else a=tau*a

             end

       end

     • metric=abs(fn)

   end
```

3.5.3 Homotopy

In contrast to both dampening and backtrack line search, homotopy approaches the issue of oscillations or divergence from a very different point of view. In particular, rather than improving, or dampening, the Newton step in order to avoid pathological behavior, homotopy focuses directly on finding a “good” initial guess, such that pathologies do not appear. As we will see, there is no conflict between using homotopy together with, say, back track line search, but it is instructive to illustrate how the method works standing on its own two feet.

The idea underlying homotopy is that if $f(x) = 0$ is a problematic equation to solve, we can always find an alternative equation, $h(x) = 0$, which is either very easy to solve, or for which we already know the solution. Homotopy then amounts to solve the problem(s)

$$\alpha_i f(x) + (1 - \alpha_i)h(x) = 0,$$

for α_i ranging from $\alpha_1 = 0$ to $\alpha_N = 1$. Then for each α_i denote the solution to the above equation as $x^*(\alpha_i)$. We then use $x^*(\alpha_i)$ as the initial guess for solving for $x^*(\alpha_{i+1})$ etc. If the grid for α is sufficiently fine, $x^*(\alpha_i)$ will indeed be a good initial guess for finding $x^*(\alpha_{i+1})$.

What is the equation $h(x)$? In principle, it could be anything that you either know the solution to, or know how to find the solution. But it is a good idea to pick some $h(x)$ that somewhat resembles $f(x)$. For instance, in the pathological example illustrated in Figure 3.5, the equation $f(x)$ was equal to $x^3 - 2x + 2$. Thus, a reasonable choice of $h(x)$ might be $x^3 + 2$, as the solution is immediately given by $x = 2^{1/3}$. Indeed, using homotopy with this choice of $h(x)$ and with $\alpha_i \in \{0, 0.1, 0.2, \ldots, 1\}$ remedies the issues of oscillations.[11]

3.6 Examples

3.6.1 Edgeworth's box

Consider the optimization problem for an individual i,

$$U_i = \max_{x_{1,i}, x_{2,i}} \left\{ \left(a_i^{\frac{1}{s_i}} x_{1,i}^{\frac{s_i - 1}{s_i}} + x_{2,i}^{\frac{s_i - 1}{s_i}} \right)^{\frac{s_i}{s_i - 1}} \right\},$$
$$\text{subject to} \quad e_{1,i} + pe_{2,i} = x_{1,i} + px_{2,i},$$

where $e_{i,j}$ denotes individual i's endowment of good $j \in \{1, 2\}$.

We assume that there are two individuals, A and B, and set $a_A = 1.5$, $a_B = 1$, $s_A = 0.3$, $s_B = 0.8$, $e_{1,A} = 2$, $e_{1,B} = 3$, $e_{2,A} = 4$ and $e_{2,B} = 1$. Thus, individual A has a slight preference for good 1 relative to good 2, but is quite well endowed with good 2. In addition, the elasticity of substitution, s_i, is lower for individual A than for individual B. This parameterization ensures that there is ample scope for Pareto-improving trade amongst the two agents. Indeed, the upper left quadrant of Figure 3.7 illustrates the endowment point in an Edgeworth box together with the agents' indifference curves. The grey shaded area illustrates the *core*. We know according to economic theory that a Pareto optimal consumption allocation should be within the core such that the agents' indifferences curves are tangent to each other. We also know from the first welfare theorem that a competitive equilibrium would attain this outcome. So let's have a look.

The demand functions by the agents are given by

$$x_{1,i}(p) = \frac{e_{1,i} + pe_{2,i}}{1 + \frac{p^{1-s_i}}{a_i}}, \quad \text{and} \quad x_{2,i}(p) = \frac{e_{1,i} + pe_{2,i}}{p + p^{s_i} a_i}.$$

[11] Unfortunately, it turns out that with an initial guess of $x = 2^{1/3}$ Newton's method does not display any oscillations, and converges quickly to the correct solution. Thus, to make things more challenging, setting $h(x) = x^3 + 1.9592$, yields a pathological initial guess, but yet the homotopy approach leads to convergence.

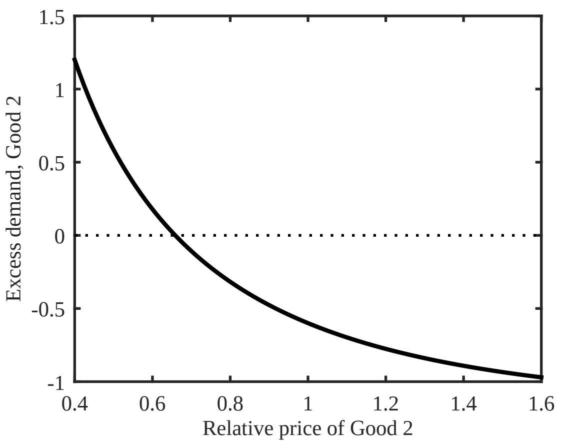

Figure 3.6: Excess Demand Function, $Z(p)$.

The *excess demand* function for good 2 is given by

$$Z(p) = x_{2,A}(p) + x_{2,B}(p) - (e_{2,A} + e_{2,B}).$$

Given the simplicity of the model, we can calculate the excess demand function, $Z(p)$, using a fine grip for p, and plot the result. Figure 3.6 illustrates the outcome, in which the price, p, is shown on the x-axis, and excess demand on the y-axis. The figure has quite a predictable pattern – as prices increase, excess demand monotonically decreases. Thus, to solve for the equilibrium price, p^*, the bisection method appears suitable.

While Figure 3.6 provide ample information regarding an initial guess, it is a rare luxury to be able to graphically illustrates excess demand on a fine grid for prices; indeed, evaluating the excess demand function in more complicated settings can be very costly, and is not an advisable procedure. Thus, to approach the problem from a more realistic perspective, we used the bisection method under the veil of ignorance regarding what the true excess demand function looked like. To this end, we started with the values $p_l = 0.5$ and $p_h = 3$, which delivered opposite signs of $Z(p)$.[12] The resulting equilibrium price is $p^* = 0.659$, and the equilibrium consumption allocation is illustrated as the black dot in the upper right quadrant of Figure 3.7.

The lower left quadrant shows the indifference curves at the equilibrium allocation, and the lower right quadrant includes the price line/budget constraint for individual A. Indeed, economic theory was correct; at the competitive equilibrium the indifference curves are tangent, and the outcome is Pareto optimal.

What are the gains from trade in this setting? A commonly used measure is to compute the proportional percentage increase in the endowments under autarchy needed to deliver the same utility as that under trade. In our setting that amounts to finding λ such that

$$\left(a_i^{\frac{1}{s_i}} x_{1,i}(p*)^{\frac{s_i-1}{s_i}} + x_{2,i}(p^*)^{\frac{s_i-1}{s_i}}\right)^{\frac{s_i}{s_i-1}} = \left(a_i^{\frac{1}{s_i}} (\lambda_i e_{1,i})^{\frac{s_i-1}{s_i}} + (\lambda_i e_{2,i})^{\frac{s_i-1}{s_i}}\right)^{\frac{s_i}{s_i-1}}$$

[12]As we know that if $a_1 = a_2$ implies $p^* = 1$, these initial values appear conservative.

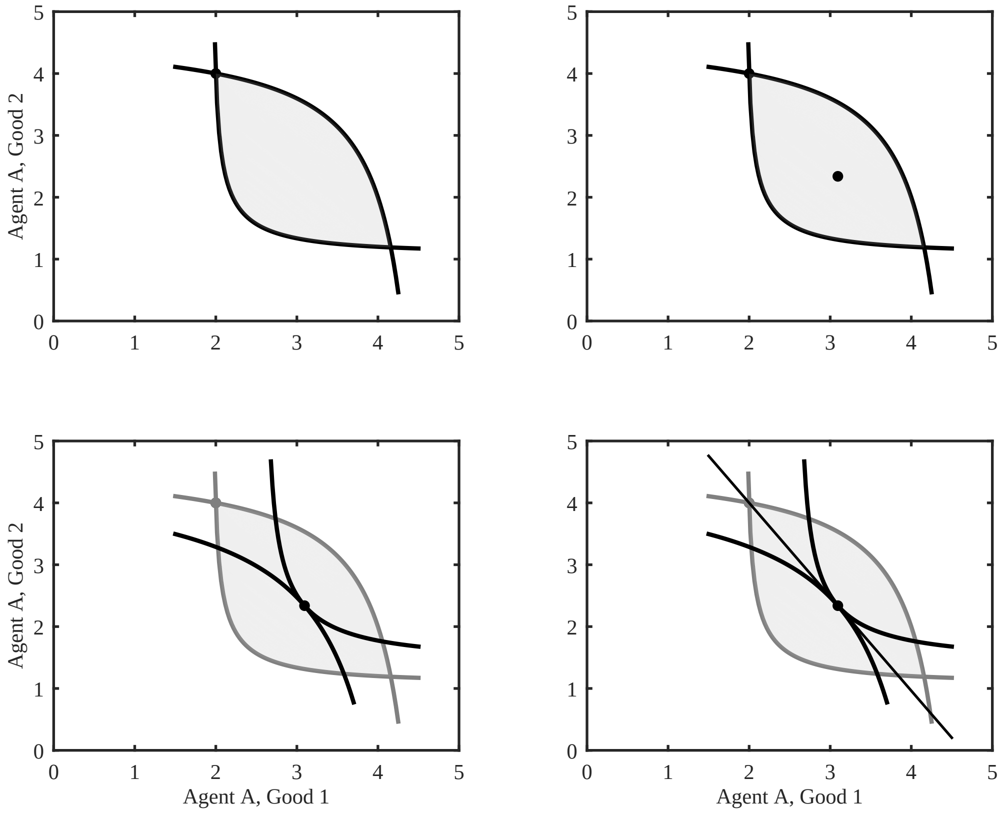

Figure 3.7: The Edgeworth's Box.

or simply

$$\lambda_i = \frac{\left(a_i^{\frac{1}{s_i}} x_{1,i}(p*)^{\frac{s_i-1}{s_i}} + x_{2,i}(p^*)^{\frac{s_i-1}{s_i}}\right)^{\frac{s_i}{s_i-1}}}{\left(a_i^{\frac{1}{s_i}} (\lambda e_{1,i})^{\frac{s_i-1}{s_i}} + (\lambda e_{2,i})^{\frac{s_i-1}{s_i}}\right)^{\frac{s_i}{s_i-1}}}.$$

For our framework, it turns out that $\lambda_A = 1.33$ and $\lambda_B = 1.35$. That is, both individuals experience a utility gain from trade that is similar to increasing their endowments by more than 30%. Not bad at all.

Lastly, an interesting proposition arising from general equilibrium theory is that irrespective of how the endowments are distributed, the equilibrium allocation will always belong to a single line, called the contract curve. This can be a somewhat surprising results, as it means that for any specific value of, say, $x^*_{1,A}$, the exist only *one* value of $x^*_{2,A}$ which constitutes an equilibrium – a result that is true for any distribution of endowments (and there are normally infinitely many) such that $x^*_{1,A}$ is the equilibrium allocation. We verify this proposition by solving for the equilibrium allocations at a large number of possible endowments. The left panel in Figure 3.8 illustrates the result of this exercise; and the right panel illustrates three stark examples of endowment allocations (grey dots) and the ensuing equilibrium outcome (black dots). Indeed, a unique contract curve emerges, and

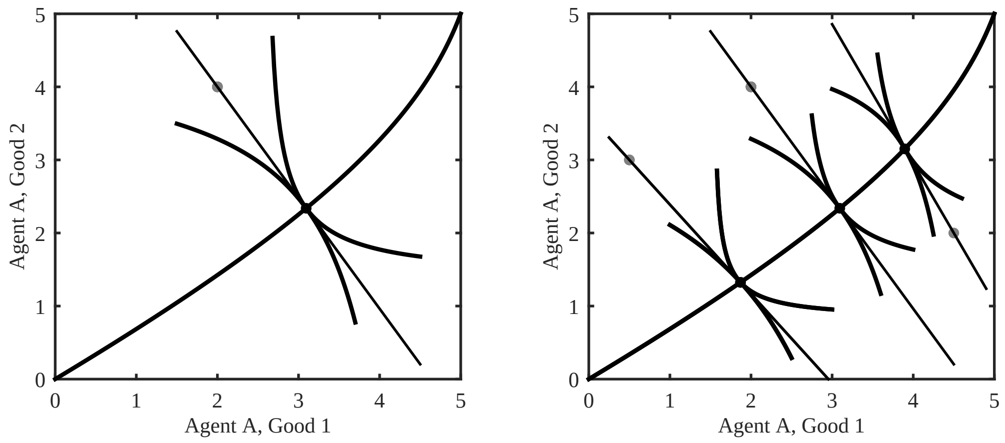

Figure 3.8: The Edgeworth's Box – contract curve.

illustrates all possible Pareto optimal allocations.

Bibliography

Dowd, Kevin, John Cotter, Chris Humphrey, and Margaret Woods. 2008. "How Unlucky is 25-Sigma?" *CRIS Discussion Paper Series* 2008.III.