A Appendix

A.1 Computing integer powers by the binary method

The algorithm in this session and much more is discussed in [**?**, §4.6.3]. We derive an algorithm to compute x^n when n is an integer. First define the recursion

$$r_k = \lfloor r_{k-1}/2 \rfloor, \qquad b_k = (r_{k-1} \mod 2).$$

with $r_0 = n$ and $b_0 = 0$ and

$$Y_k = Y_{k-1} Z_{k-1}^{b_k}, \qquad Z_k = Z_{k-1}^2$$

with $Y_0 = 1$ and $Z_0 = x$.

Now verify that

$$Y_k Z_k^{r_k} = (Y_{k-1} Z_{k-1}^{b_k})(Z_{k-1}^2)^{r_k} = Y_{k-1} Z_{k-1}^{r_{k-1}} = Y_0 Z_0^{r_0} = x^n$$

In particular the algorithm terminates at k^* when $r_{k^*} = 0$ which implies that $Y_{k^*} = x^n$ is the answer to the problem.

The following program implements this algorithm:

```
/* Program w4-22.c */

/* Compute the nth integer power using the binary algorithm */

#include <stdio.h>
main() {

  /* declare variables */
  double x, yk, zk;
  int n, rk, bk;

  /* input base and exponent */
  printf("Enter x: ");
  scanf("%lf", &x);
  printf("Enter an integer exponent: ");
  scanf("%d", &n);

  /* Compute the power */

  /* initialize algorithm */
  rk = n;  bk = 0;
```

```
  yk = 1;  zk = x;

  /* infinite loop */
  while (1) {

    /* update bk and rk */
    bk = rk % 2;
    rk /= 2;

    /* update yk */
    if (bk)
      yk *= zk; /* yk = yk * zk; */

    if (!rk)
      break;

    /* update zk */
    zk *= zk;  /* zk = zk * zk; */

  }

  /* print power */
  printf("%f ^ %d = %g\n", x, n, yk);

}
```

A.2 Basic numerical integration

This section is concerned with computing a numerical approximation to the definite integral

$$\int_a^b f(x)\,dx$$

If $f(x)$ is sufficiently smooth then one can think of approximating $f(x)$ by its value at the middle of the interval $[a, b]$, i.e. at

$$\xi := \frac{a+b}{2}$$

to produce the approximation

$$\int_a^b f(x)\,dx \approx \int_a^b f(\xi)\,dx = f(\xi)\int_a^b dx = (b-a)f(\xi)$$

This is the *rectangle rule*, as the approximate integral corresponds to the area of the rectangle of width $b-a$ and height $f(\xi)$.

Alternatively one can think of using the trapezoid defined by the points $(a,0)$, $(a,f(a))$, $(b,f(b))$, $(b,0)$. This produces the approximation

$$\int_a^b f(x)\,dx \approx \frac{1}{2}(b-a)[f(a)+f(b)]$$

which is known as the *trapezoidal rule*.

Better approximations can be obtained by dividing the interval $[a,b]$ in sub-intervals. For instance, subdividing $[a,b]$ in n equally spaced sub-interval, repeated application of the trapezoidal rule produces the approximation

$$\begin{aligned}\int_a^b f(x)\,dx = \sum_{i=0}^{n-1}\int_{a+i\delta}^{a+(i+1)\delta} f(x)\,dx &\approx \sum_{i=0}^{n-1}\frac{1}{2}\delta[f(a+i\delta)+f(a+(i+1)\delta)] \\ &= \frac{1}{2}\delta\,[f(a)+f(b)] + \sum_{i=1}^{n-1}\delta\,f(a+i\delta)\end{aligned}$$

where

$$\delta = \frac{1}{n}(b-a).$$

A third possibility, this time based on interpolation by polynomials, is the Simpson's rule which produces the approximation:

$$\int_a^b f(x)\,dx \approx \frac{1}{6}(b-a)\left[f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right].$$

Repeated application of the Simpson's rule produces the approximation

$$\begin{aligned}\int_a^b f(x)\,dx \approx{}& \frac{1}{3}\delta\,[f(a)+f(b)]+ \\ & \frac{1}{3}\delta\,[4f(a+\delta)+2f(a+2\delta)+4f(a+3\delta)+2f(a+4\delta)+\cdots \\ & \qquad \cdots+2f(a+(n-2)\delta)+4f(a+(n-1)\delta)]\end{aligned}$$

where δ is the same as used for the trapezoidal rule. Note that the above formula can only be used for n even! Do you see why?

A.3 Finding zeros of functions

In this section we describe a method known as the Newton-Raphson Method, or Newton's Method for short, for solving algebraic equations numerically. Variants of Newton's Method are widely used in scientific and engineering software.

Instead of computing the zeros of a given function directly, Newton's Method iteratively find the zeros of the function's first order Taylor series expansion. Given a function f which is continuous with a continuous first derivative, than one should expect that the value of f near a given point x can be aproximated by its first order Taylor expansion

$$f(y) \approx f(x) + f'(x)(y - x) \tag{1}$$

where $f'(x)$ denotes the first derivative of f with respect to x.

If x is close to a zero of f and $f'(x) \neq 0$, we have that

$$y = x - f(x)/f'(x)$$

after equating the right-hand side of (1) to zero.

The above idea is then iterated

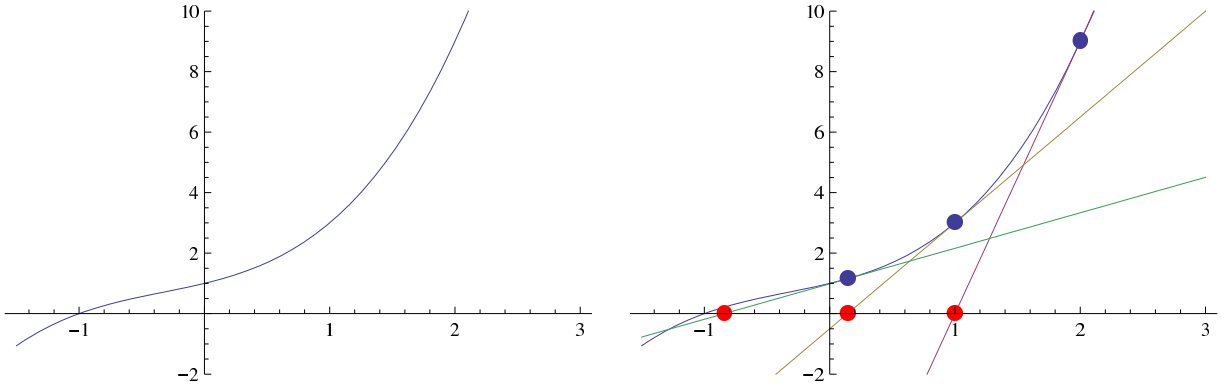
$$x_{k+1} = x_k - f(x_k)/f'(x_k).$$

Starting from and arbitrary x_0, we continue until either the successive x-values are sufficiently close or the absolute value of the function is sufficiently small. The process is illustrated in Figure 2.

Zeros of functions provide solutions to many problems is science and engineering. In order to illustrate one very simple application, consider the problem of find extreme points (maxima and minima) of a function in a single variable. If the function is continuous with continuous first and second derivatives then, as you should have seen in Calculus, at an extreme point we shall have a zero first derivative, that is

$$f'(x^*) = 0.$$

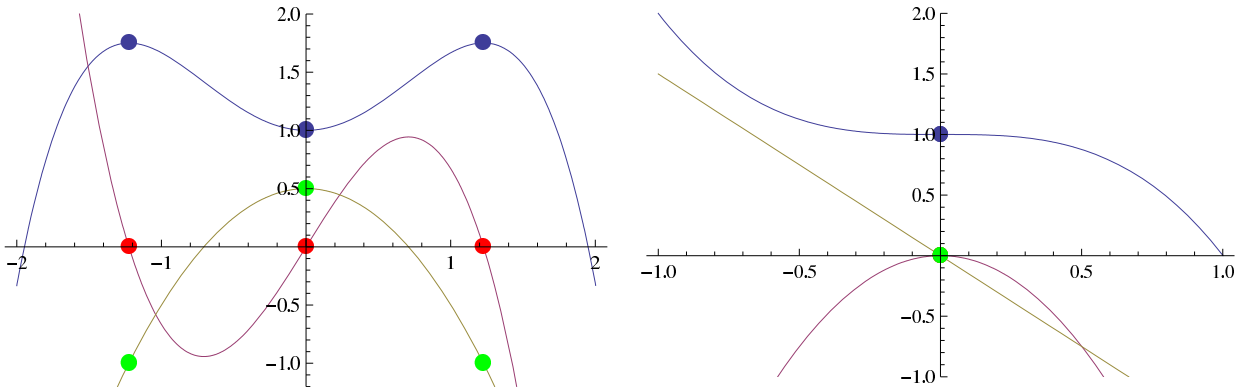
If the point x^* is a maximum then we will have $f''(x^*) < 0$. Likewise, if x^* is a minimum, $f''(x^*) > 0$. If $f''(x^*) = 0$ then x^* is not a maximum nor a minimum, and in this simple case with a single variable it will be an inflection



(a) $f(x)$ (b) Newton iterations

Figure 2: Illustration of three iterations of Newton's method. Blue dots are $f(x_k)$'s and red dots are the roots of the associated first order Taylor expansion, i.e. the x_{k+1}'s.

point. Equipped with Newton's method one can then search for extreme points by numerically computing the roots of $f(x)$. Some examples of functions, their first and second order derivatives and the corresponding extreme points are shown in Figure 3.



(a) Two maxima and one minimum

(b) No maxima or minima, one inflexion point

Figure 3: Some functions and their first and second derivatives with extreme points.