1. [Kincaid and Cheney Problem 3.2#4] Steffensen's method is given by the iteration

$$x_{n+1} = x_n - \frac{f(x_n)}{g(x_n)} \qquad \text{where} \qquad g(x) = \frac{f\big(x + f(x)\big) - f(x)}{f(x)}.$$

Show this method is quadratically convergent under suitable hypothesis.

Let r be a solution such that $f(r) = 0$ and $f'(r) \neq 0$. Moreover assume $|f''|$ is bounded in a neighborhood of r. We shall show that Steffensen's method is quadratically convergent provided that x_0 is sufficiently close to r.

Consider the function

$$\kappa(x, \xi, \eta) = \frac{f''(\xi)\big(f'(x) - \frac{1}{2}f''(\eta)(x - r)\big) + f''(\eta)}{2f'(x) + f''(\xi)f(x)}.$$

By assumption the limit supremum of $\big|\kappa(x, \xi, \eta)\big|$ exists as $x \to r$, $\xi \to r$ and $\eta \to r$. Let C denote this limit and choose $\delta_1 > 0$ so small that $\big|\kappa(x, \xi, \eta)\big| \leq 2C$ for every x, ξ and η such that $|x - r| < \delta_1$, $|\xi - r| < \delta_1$ and $|\eta - r| < \delta_1$. Choose $\delta_2 > 0$ so small that $|x - r| < \delta_2$ implies $2\big|f(x)\big| \leq \delta_1$. Now let

$$\delta = \min\Big\{\frac{\delta_1}{2}, \delta_2, \frac{1}{2C}\Big\}.$$

Claim that if $x_0 \in (r - \delta, r + \delta)$ then x_n defined according to Steffensen's method satisfies $x_n \in (r - \delta, r + \delta)$ for all $n \in \mathbf{N}$.

Let $x_0 \in (r - \delta, r + \delta)$ and define x_n according to Steffensen's method. By Taylor's theorem there exists ξ_n between x_n and $x_n + f(x_n)$ such that

$$f\big(x_n + f(x_n)\big) = f(x_n) + f'(x_n)f(x_n) + \tfrac{1}{2}f''(\xi_n)f(x_n)^2.$$

Therefore

$$g(x_n) = f'(x_n) + \tfrac{1}{2}f''(\xi_n)f(x_n).$$

It follows that

$$e_{n+1} = x_{n+1} - r = e_n - \frac{f(x_n)}{f'(x_n) + \frac{1}{2}f''(\xi_n)f(x_n)}$$

where $e_n = x_n - r$.

Using Taylor's series again yields

$$0 = f(r) = f(x_n - e_n) = f(x_n) - f'(x_n)e_n + \tfrac{1}{2}f''(\eta_n)e_n^2$$

for some choice of η_n between r and x_n. Therefore

$$\begin{aligned}
e_{n+1} &= e_n - e_n\frac{f'(x_n) + \frac{1}{2}f''(\eta_n)e_n}{f'(x_n) + \frac{1}{2}f''(\xi_n)f(x_n)} \\
&= e_n - e_n\frac{f'(x_n) - \frac{1}{2}f''(\xi_n)f(x_n) + \frac{1}{2}f''(\xi_n)f(x_n) + \frac{1}{2}f''(\eta_n)e_n}{f'(x_n) + \frac{1}{2}f''(\xi_n)f(x_n)} \\
&= -e_n\frac{f''(\xi_n)f(x_n) + f''(\eta_n)e_n}{2f'(x_n) + f''(\xi_n)f(x_n)} \\
&= -e_n\frac{f''(\xi_n)\big(f'(x_n)e_n - \frac{1}{2}f''(\eta_n)e_n^2\big) + f''(\eta_n)e_n}{2f'(x_n) + f''(\xi_n)f(x_n)} \\
&= -e_n^2\frac{f''(\xi_n)\big(f'(x_n) - \frac{1}{2}f''(\eta_n)e_n\big) + f''(\eta_n)}{2f'(x_n) + f''(\xi_n)f(x_n)} = -e_n^2\kappa(x_n, \xi_n, \eta_n).
\end{aligned}$$

Claim that $x_n \in (r-\delta, r+\delta)$ for every $n \in \mathbf{N}$. For induction suppose $x_n \in (r-\delta, r+\delta)$. Since ξ_n is between x_n and $x_n + f(x_n)$ then

$$|\xi_n - r| \leq |\xi_n - x_n| + |x_n - r| \leq |f(x_n)| + |x_n - r| < \delta_2 + \delta \leq \delta_1.$$

Since η_n is between x and x_n then

$$|\eta_n - r| \leq \delta < \delta_1.$$

Thus x_n, ξ_n and η_n satisfy $|x_n - r| < \delta_1$, $|\xi_n - r| < \delta_1$ and $|\eta_n - r| < \delta_1$ so consequently $\big|\kappa(x_n, \xi_n, \eta_n)\big| < 2C$. Therefore

$$|x_{n+1} - r| = |e_{n+1}| \leq 2Ce_n^2 = 2C|x_n - r|^2 \leq 2C\delta^2 < \delta.$$

This shows $x_{n+1} \in (r - \delta, r + \delta)$ and completes the induction. Moreover, we have shown that x_n is quadratically convergent.

2. [Kincaid and Cheney Problem 3.2#6] To compute reciprocals without division, we can solve $x = 1/R$ by finding a zero of the function $f(x) = x^{-1} - R$. Write a short algorithm to find $1/R$ by Newton's method applied to f. Do not use division or exponentiation in your algorithm. For positive R, what starting points are suitable?

Differentiating yields

$$f'(x) = -x^{-2}.$$

Therefore

$$\phi(x) = x - \frac{f(x)}{f'(x)} = x + \frac{x^{-1} - R}{x^{-2}} = x + x - Rx^2 = x(2 - Rx)$$

leads to the iteration formula $x_{n+1} = x_n(2 - Rx_n)$. To determine what starting points x_0 are suitable we look for an interval centered at $1/R$ over which $|\phi'|$ is strictly less than 1.

$$\left|\phi'(x)\right| = |2 - 2Rx| < 1 \qquad \text{implies} \qquad \left|\frac{1}{R} - x\right| < \frac{1}{2R}.$$

Therefore any starting point

$$x_0 \in \left(\frac{1}{2R}, \frac{3}{2R}\right)$$

is suitable. In fact, this interval can be extended to

$$x_0 \in \left(0, \frac{3}{2R}\right)$$

because if $x \in (0, 1/2R]$ then

$$\phi(x) - x = x(2 - Rx) - x = x(1 - Rx)$$

implies

$$\frac{x}{2} \le \phi(x) - x < \frac{1}{2R}.$$

Consequently, if $x_0 \in (0, 1/2R]$, then x_n form an increasing sequence of points that are eventually contained in the interval where $|\phi'| < 1$. Similarly

$$x_0 \in \left[\frac{3}{2R}, \frac{2}{R}\right) \qquad \text{implies} \qquad x_1 \in \left(0, \frac{3}{4R}\right].$$

It follows that the method iteration will converge for any

$$x_0 \in \left(0, \frac{2}{R}\right).$$

We now procede to find a choice for x_0 that does not involve performing any divisions. Recall that R is stored on the computer using a binary expansion of the form

$$R = \sum_{i=1}^{m} b_i 2^{n-i} = \sigma 2^n \qquad \text{where} \qquad \sigma \in [1/2, 1).$$

Therefore taking

$$x_0 = \sigma 2^{-n} < \frac{1}{\sigma 2^n} = \frac{1}{R}$$

provides a suitable initial condition by simply changing the sign of the exponent in the floating point representation while doing no division or exponentiation to the mantissa. The following C program implements this approach.

```
/*
    newtdiv.c -- Find 1/R without hardware divide instruction
    Written October 17, 2012 by Eric Olson for Math 701
*/
#include <stdio.h>
#include <math.h>

typedef union {
    struct {
        long long unsigned sigma:52;
        unsigned n:11;
        unsigned s:1;
    } bits;
    double value;
} ieeebits;

double invert(double R){
    double x,x0;
    ieeebits guess;
    guess.value=R;
    guess.bits.n=2044-guess.bits.n;
    x0=guess.value;
    int i;
    for(i=1;i<100;i++){
        x=x0*(2-R*x0);
        if(x==x0) return x;
        x0=x;
    }
    return NAN;
}

int main(){
    printf(
        "newtdiv.c -- Find 1/R without hardware divide instruction\n"
        "Written October 17, 2012 by Eric Olson for Math 701\n");

    double R[]={ 1.0 , 2.0, 0.999, 214.32, 0.000128 };
    int N=sizeof(R)/sizeof(double);
    printf("\n%2s %22s %22s %22s\n","n","R","1/R","|1-R(1/R)|");
    int n;
    for(n=0;n<N;n++){
        double x=R[n],y=invert(x);
        printf("%2d %22.15e %22.15e %22.15e\n",n+1,x,y,fabs(1.0-x*y));
    }
    return 0;
}
```

The output is

```
newtdiv.c -- Find 1/R without hardware divide instruction
Written October 17, 2012 by Eric Olson for Math 701

 n                      R                     1/R             |1-R(1/R)|
 1  1.000000000000000e+00  1.000000000000000e+00  0.000000000000000e+00
 2  2.000000000000000e+00  5.000000000000000e-01  0.000000000000000e+00
 3  9.990000000000000e-01  1.001001001001001e+00  8.749511531958021e-17
 4  2.143200000000000e+02  4.665920119447556e-03  9.063930161978817e-17
 5  1.280000000000000e-04  7.812500000000000e+03  4.526544070126981e-17
```

3. [Kincaid and Cheney Problem 3.2#17] Which of the following sequences converge quadratically.

$$\text{a. } 1/n^2, \quad \text{b. } 1/2^{2^n}, \quad \text{c. } 1/\sqrt{n}, \quad \text{d. } 1/e^n, \quad \text{e. } 1/n^n$$

A sequence x_n converging to a limit L, converges quadratically if there is a constant C and an integer N such that $|x_{n+1} - L| \leq C|x_n - L|^2$ for every $n \geq N$. Since $L = 0$ for each of the sequences in question and each sequence is moreover positive, then this inequality may be rewritten as

$$\frac{x_{n+1}}{x_n^2} \leq C \qquad \text{for every} \qquad n \geq N.$$

This inequality will hold for some C and N large enough if and only if

$$\limsup_{n\to\infty} \frac{x_{n+1}}{x_n^2}$$

is bounded. We proceed as follows:

$$\limsup_{n\to\infty} \frac{1/(n+1)^2}{(1/n^2)^2} = \lim_{n\to\infty} \frac{n^4}{n^2+2n+1} = \infty$$

$$\limsup_{n\to\infty} \frac{1/2^{2^{n+1}}}{(1/2^{2^n})^2} = \lim_{n\to\infty} \frac{4^{2^n}}{4^{2^n}} = 1$$

$$\limsup_{n\to\infty} \frac{1/\sqrt{n+1}}{(1/\sqrt{n})^2} = \lim_{n\to\infty} \frac{n}{\sqrt{n+1}} = \infty$$

$$\limsup_{n\to\infty} \frac{1/e^{n+1}}{(1/e^n)^2} = \lim_{n\to\infty} \frac{e^{2n}}{e^{n+1}} == \lim_{n\to\infty} e^{n-1} = \infty$$

$$\limsup_{n\to\infty} \frac{1/(n+1)^{n+1}}{(1/n^n)^2} = \lim_{n\to\infty} \frac{n^{2n}}{(n+1)^{n+1}} \geq \lim_{n\to\infty} \frac{n^{2n}}{(2n)^{n+1}} = \lim_{n\to\infty} \frac{1}{4}\Big(\frac{n}{2}\Big)^n = \infty.$$

Therefore only $1/2^{2^n}$ converges quadratically while the others don't.

4. [Kincaid and Cheney Computer Problem 3.2#1] Write a computer program to solve the equation $x = \tan x$ by Newton's method. Find the roots nearest 4.5 and 7.7.

The program is

```
/*
    newtontan.c -- Newton's method to solve tan(x) = x.
    Kincaid and Cheney Computer Exercise 3.2#1
    Written October 15, 2012 for Math 701 by Eric Olson
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define N 100

double phi(double x){
    double t=tan(x);
    double f=t-x;
    double df=t*t;
    return x-f/df;
}
double newton(double x0){
    double x[N];
    x[0]=x0;
    int n;
    for(n=1;n<N;n++) {
        x[n]=phi(x[n-1]);
        if(x[n]==x[n/2]) return x[n];
    }
    printf("Newton failed to converge!\n");
    exit(1);
}
double x0[]={ 4.5, 7.7 };
int K=sizeof(x0)/sizeof(double);
main(){
printf(
        "Kincaid and Cheney Computer Exercise 3.2#1\n"
        "Written October 15, 2012 for Math 701 by Eric Olson\n");

    printf("\n%4s %24s %24s\n","n","x0","root");
    int n;
    for(n=0;n<K;n++)
        printf("%4d %24.15e %24.15e\n",
            n+1,x0[n],newton(x0[n]));
    return 0;
}
```

and the output is

```
Kincaid and Cheney Computer Exercise 3.2#1
Written October 15, 2012 for Math 701 by Eric Olson

   n                       x0                       root
   1    4.500000000000000e+00    4.493409457909064e+00
   2    7.700000000000000e+00    7.725251836937707e+00
```

5. [Kincaid and Cheney Computer Problem 3.2#8] Program the Newton algorithm in complex arithmetic and test it on these functions with the given starting points.
   a. $f_1(z) = z^2 + 1, \qquad z_0 = 3 + i$
   b. $f_2(z) = z + \sin z - 3, \qquad z_0 = 2 - i$
   c. $f_3(z) = z^4 + z^2 + 2 + 3i, \qquad z_0 = 1$

We shall use the Maple code generation library to create the functions $\phi(z) = z - f(z)/f'(z)$ needed for Newton's iteration. The Maple script

```
restart;
#   makephi.mpl -- Make phi to solve F=x for Newton's Iteration
#   Written October 15, 2012 for Math 701 by Eric Olson
#
#   F is an expression in x
#   N is the number of the function
#
kernelopts(printbytes=false);
fname:=sprintf("phi%d.i",N);
fd:=open(fname,WRITE);
fprintf(fd,"/* %s -- Created by Maple */\n",fname);
close(fd);

with(codegen,C,optimize):
cmp:=[I=cI,cos=ccos,sin=csin,exp=cexp];
tmp:=[seq(t||i=T[i],i=1..64)];

f:=F;
df:=diff(f,z);
r1:=[phi=z-f/df];
r2:=[optimize(r1,'tryhard')];
r3:=subs(cmp,r2);
r4:=subs(tmp,r3);
C(r4,filename=fname);
```

takes an expression F and an integer N and creates the file `phiN.i` containing C code to compute ϕ in terms of z. The script can be used to generate code for the functions f_n given in the problem with the commands

```
$ maple -q -DN=1 -DF="z^2+1" makephi.mpl
$ maple -q -DN=2 -DF="z+sin(z)-3" makephi.mpl
$ maple -q -DN=3 -DF="z^4+z^2+2+3*I" makephi.mpl
```

The resulting files are

```
/* phi1.i -- Created by Maple */
      phi = z-(z*z+1.0)/z/2.0;
```

```
/* phi2.i -- Created by Maple */
      phi = z-(z+csin(z)-3.0)/(1.0+ccos(z));
```

```
/* phi3.i -- Created by Maple */
      T[0] = z*z;
      phi = z-(T[0]*T[0]+T[0]+2.0+3.0*cI)/(4.0*T[0]+2.0)/z;
```

Notice that `phi3.i` contains an optimization where z^2 is stored in a temporary variable. These files are included in our main program `newtcomp.c` which is given by

```
/*
    newtcomp.c -- Newton's method to solve f(z) = 0.
    Kincaid and Cheney Computer Exercise 3.2#8
    Written October 15, 2012 for Math 701 by Eric Olson
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <complex.h>

#define cI I
typedef double complex Complex;
typedef Complex (*Function)(Complex z);

Complex phi1(Complex z){
    Complex T[64],phi;
#include "phi1.i"
    return phi;
}
Complex phi2(Complex z){
    Complex T[64],phi;
#include "phi2.i"
    return phi;
}
Complex phi3(Complex z){
    Complex T[64],phi;
#include "phi3.i"
    return phi;
}

#define N 100
Complex iterate(Function phi,Complex z0){
    Complex z[N];
    z[0]=z0;
    int n;
    for(n=1;n<N;n++) {
        z[n]=phi(z[n-1]);
        if(z[n]==z[n/2]) return z[n];
    }
    printf("Newton failed to converge!\n");
    exit(0);
}
Complex z0[]={ 3+I, 2-I, 1 };
Function phi[]={ phi1, phi2, phi3 };
int K=sizeof(z0)/sizeof(Complex);

main(){
    printf(
        "Kincaid and Cheney Computer Exercise 3.2#8\n"
        "Written October 15, 2012 for Math 701 by Eric Olson\n");

```

```
53      printf("\n%4s %16s %16s %16s %16s\n",
54          "n","re(z0)","im(z0)","re(z)","im(z)");
55      int n;
56      for(n=0;n<K;n++)
57          printf("%4d %16.8e %16.8e %16.8e %16.8e\n",
58              n+1,z0[n],iterate(phi[n],z0[n]));
59      return 0;
60  }
```

The output is

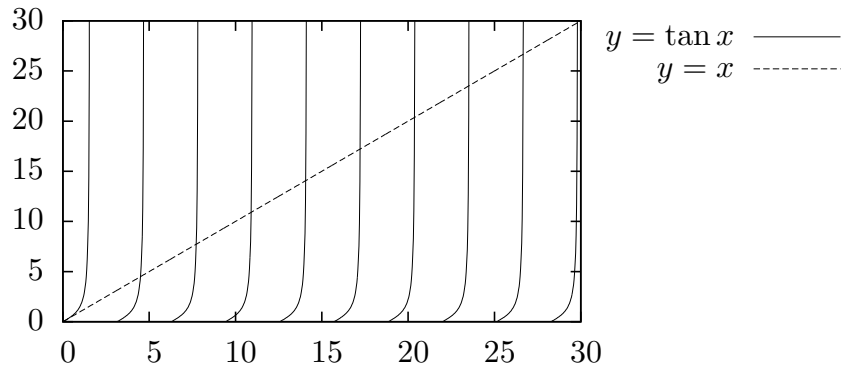
```
Kincaid and Cheney Computer Exercise 3.2#8
Written October 15, 2012 for Math 701 by Eric Olson

   n           re(z0)           im(z0)            re(z)            im(z)
   1   3.00000000e+00   1.00000000e+00   0.00000000e+00   1.00000000e+00
   2   2.00000000e+00  -1.00000000e+00   2.17975707e+00   0.00000000e+00
   3   1.00000000e+00   0.00000000e+00   1.02469479e+00  -7.88542500e-01
```

6. [Kincaid and Cheney Computer Problem 3.2#9] Use Steffensen's method to find the first 10 positive roots of the equation $x = \tan x$.

First graph the equation $y = \tan x$ and $y = x$ to identify on what intervals the roots will be. The gnuplot script

```
set terminal pstex
set output "xtanx.tex"
set key outside
set key width 1
set samples 1000
set size 0.8,0.5
plot [0:30] [0:30] tan(x) ti "$y=\\tan x$",x ti "$y=x$"
```

produces the output



There is root between each vertical asymptote. The first 10 positive roots satisfy

$$x_n \in \left(\frac{(2n-1)\pi}{2}, \frac{(2n+1)\pi}{2}\right) \qquad \text{where} \qquad n = 1, 2, \ldots, 10.$$

Steffensen's method requires the initial guess be very close to the correct root to converge because of the aymptotes. To overcome this problem we check whether each Steffensen's iterate remains in the interval where the n-th root is located. If it does not remain in the interval, then we take a bisection step and try again. The program is

```
/*
    stefbisect.c -- Steffensen's method to solve tan(x) = x.
    Kincaid and Cheney Computer Exercise 3.2#9
    Written October 15, 2012 for Math 701 by Eric Olson
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define N 3000
typedef long double Real;

Real f(Real x){
    return tan(x)-x;
}
```

```
17 Real df(Real x){
18     Real t=tan(x);
19     return t*t;
20 }
21 Real g(Real x){
22     Real t=f(x);
23     return (f(x+t)-t)/t;
24 }
25 Real steffensen(Real a, Real b, Real x0){
26     Real x[N];
27     x[0]=x0;
28     int n;
29     for(n=1;n<N;n++){
30         Real r=x[n-1];
31         Real t1=f(r);
32         if(t1==0.0) return r;
33         Real t2=g(r);
34         if(t2==0.0) return r;
35         r=r-t1/t2;
36         if(r<a||r>b){
37             r=(a+b)/2;
38             if(f(r)<0) a=r;
39             else b=r;
40         }
41         x[n]=r;
42         if(x[n]==x[n/2]) return x[n];
43     }
44     printf("Steffensen failed to converge!\n");
45     exit(0);
46 }
47 int main () {
48     printf(
49         "Kincaid and Cheney Computer Exercise 3.2#9\n"
50         "Written October 15, 2012 for Math 701 by Eric Olson\n");
51     int n;
52     printf("\n%4s %30s\n","n","x[n]");
53     for(n=1;n<=10;n++){
54         Real r=steffensen((2*n-1)*M_PI/2,(2*n+1)*M_PI/2,3*n*M_PI/2);
55         printf("%4d %30.20Le\n",n,r);
56     }
57     return 0;
58 }
```

and the output is

```
Kincaid and Cheney Computer Exercise 3.2#9
Written October 15, 2012 for Math 701 by Eric Olson

   n                           x[n]
   1     4.49340945790906443839e+00
   2     7.72525183693770697282e+00
   3     1.09041216594289006193e+01
   4     1.40661939128314738452e+01
   5     1.72207552719307686595e+01
   6     2.03713029592875630482e+01
```

```
 7     2.35194524986890063350e+01
 8     2.66660542588126723597e+01
 9     2.98115987908929581697e+01
10     3.29563890398224784671e+01
```

An alternative method is to invert the equations and work with $\arctan(y)$. This is implemented in the program

```
/*
    stefatan.c -- Steffensen's method to solve tan(x) = x.
    Kincaid and Cheney Computer Exercise 3.2#9
    An alternative approach with arctan.
    Written October 15, 2012 for Math 701 by Eric Olson
*/

#include <stdio.h>
#include <math.h>

int k;
double f(double x){
    return atan(x)-x+k*M_PI;
}
int main(){
    printf(
        "Kincaid and Cheney Computer Exercise 3.2#9\n"
        "An alternative approach with arctan.\n"
        "Written October 15, 2012 for Math 701 by Eric Olson\n");

    printf("\n%4s %24s\n","n","x");
    for(k=0;k<=10;k++){
        double x=1.0;
        int n;
        for(n=1;n<=10;n++){
            double y=f(x);
            double d=f(x+y)-y;
            if(d==0.0 || y==0.0) break;
            x=x-y*y/d;
        }
        printf("%4d %24.15e\n",k,x);
    }
    return 0;
}
```

which produces the output

```
Kincaid and Cheney Computer Exercise 3.2#9
An alternative approach with arctan.
Written October 15, 2012 for Math 701 by Eric Olson

   n                         x
   0    1.156162205308371e-02
   1    4.493409457909064e+00
   2    7.725251836937707e+00
   3    1.090412165942890e+01
   4    1.406619391283147e+01
   5    1.722075527193077e+01
```

```
 6    2.037130295928756e+01
 7    2.351945249868901e+01
 8    2.666605425881267e+01
 9    2.981159879089296e+01
10    3.295638903982248e+01
```

It is worth noting that this second method involves transforming the original problem into something easier to solve and is significantly more efficient than the first.

7. [Kincaid and Cheney Problem 3.5#1] Use Horner's algorithm to find $p(4)$ where

$$p(z) = 3z^5 - 7z^4 - 5z^3 + z^2 - 8z + 2.$$

We arrange the calculation as

	$\vert$	3	-7	-5	1	-8	2
4	$\vert$		12	20	60	244	944
	$\vert$	3	5	15	61	236	946

Therefore $p(4) = 946$.

8. [Kincaid and Cheney Problem 3.5#5] For the polynomial

$$p(z) = 3z^5 - 7z^4 - 5z^3 + z^2 - 8z + 2$$

find a disk centered at the origin that contains all the roots.

By the localization theorem all roots lie inside an open disk centered at the origin of radius

$$\begin{aligned}\rho &= 1 + |a_n|^{-1} \max\left\{\, |a_k| : k = 0, 1, \ldots, n-1 \,\right\} \\ &= 1 + 3^{-1} \max\{2, 8, 1, 5, 7\} = 1 + 8/3 = 11/3.\end{aligned}$$

9. [Kincaid and Cheney Problem 3.5#19] Prove that if p is a polynomial of degree n having real coefficients, then

$$(n-1)\big(p'(x)\big)^2 \geq np(x)p''(x).$$

Assume that the roots are real.

We prove this statement by induction. Let $\mathcal{P}_n$ be the set of all polynomials of degree n having real coefficient and real roots.

Base case: If $n = 1$ then the statement reduces to the fact that $0 \geq 0$.

Induction step: Suppose $(n-1)\big(p_n'(x)\big)^2 \geq np_n(x)p_n''(x)$ for every $p_n \in \mathcal{P}_n$. Let $p_{n+1} \in \mathcal{P}_{n+1}$. Claim that $n\big(p_{n+1}'(x)\big)^2 \geq (n+1)p_{n+1}(x)p_{n+1}''(x)$.

Let $r \in \mathbf{R}$ be a root of p_{n+1}. Then there is $p_n \in \mathcal{P}_n$ such that

$$p_{n+1}(x) = (x-r)p_n(x)$$

and therefore

$$\begin{aligned} p_{n+1}'(x) &= p_n(x) + (x-r)p_n'(x) \\ p_{n+1}''(x) &= 2p_n'(x) + (x-r)p_n''(x). \end{aligned}$$

For notational simplicity write p_n in place of $p_n(x)$ and so forth. Estimate using the induction hypothesis and completing the square to obtain

$$\begin{aligned} (n+1)p_{n+1}p_{n+1}'' &= (n+1)(x-r)p_n\big(2p_n' + (x-r)p_n''\big) \\ &= 2(n+1)(x-r)p_np_n' + (n+1)(x-r)^2p_np_n'' \\ &\leq 2(n+1)(x-r)p_np_n' + (n+1)(x-r)^2\frac{n-1}{n}(p_n')^2 \\ &= 2(n+1)(x-r)p_np_n' + \frac{n^2-1}{n}(x-r)^2(p_n')^2 \\ &= n\Big\{\big[(x-r)p_n'\big]^2 + 2p_n\big[(x-r)p_n'\big]\Big\} \\ &\qquad\qquad - \frac{1}{n}\Big\{\big[(x-r)p_n'\big]^2 - 2np_n\big[(x-r)p_n'\big]\Big\} \\ &= n\big((x-r)p_n' + p_n\big)^2 - np_n^2 - \frac{1}{n}\big((x-r)p_n' - np_n\big)^2 + np_n^2 \\ &= n\big((x-r)p_n' + p_n\big)^2 - \frac{1}{n}\big((x-r)p_n' - np_n\big)^2 \\ &\leq n\big((x-r)p_n' + p_n\big)^2 = n(p_{n+1}')^2. \end{aligned}$$

This completes the induction.

10. [Kincaid and Cheney Computer Problem 3.5#5] Write a computer routine that uses Newton's method and deflation for finding all roots of a polynomial. Test the routine using Wilkinson's polynomial

$$p_{20}(x) = \prod_{n=1}^{20} (x - n).$$

Using Newton's method to find a root of an arbitrary polynomial requires care in choosing the initial guess x_0. In the code that follows we use Theorem 3 in Kincaid and Cheney page 110 on the localization of polynomial roots to guide choices for the initial guess and to determine whether Newton's iteration has diverged.

Let $\rho_{\max}$ be ϕ times the bound obtained by Theorem 3 where $\phi = (1 + \sqrt{5})/2$ and $\rho_n = \mu\phi^n$ where μ is the unit roundoff error. Choose N to be the largest integer such that $\rho_N \le \rho_{\max}$. The following heuristics are used to guess x_0. Since it is desired to find roots of smallest magnitude first, we always try $x_0 = 0$ as the first guess. If this doesn't work we guess randomly an x_0 inside a circle centered ρ_n where $n = 0, 1, \ldots, N$ until a root is found.

It is worth noting our strategy for guessing x_0 is simple, but may not lead to a root in all cases. If a certain maximum number of initial guesses K has been tried without finding a root the program exits with an error.

The program `roots.c` is given by

```
/*
    roots.c -- Newton's method to find polynomial roots
    Kincaid and Cheney Computer Exercise 3.5#5
    Written September 25, 2012 for Math 701 by Eric Olson

    This code assumes all roots are simple roots.

    Input file format is

        n c0 c1 c2 c3 c4 ... cn

    where n is degree of polynomial and ck are real coefficients.
    End of data is indicated by n equal to -1.

Version 1 to 2 changes:

    The input file format now allows complex coefficients.

    If n > 1000 then n - 1000 is the degree of the polynomial and
    the format for the coefficients is

        n c0.x c0.y c1.x c1.y ... cn.x cn.y

*/

#include <stdio.h>
#include <stdlib.h>
#include <complex.h>
```

```
#include <math.h>
#include <fenv.h>
#define FE_BAD (FE_INVALID|FE_DIVBYZERO|FE_OVERFLOW|FE_UNDERFLOW)

/*
    Compile time constants are

        K maximum number of initial guesses
        N maximum number of iterations per guess
        L maximum degree of polynomial
        P number of pollishing iterations
*/

#define K 1000
#define N 1000
#define L 30
#define P 7

typedef long double complex Complex;
typedef long double Real;

typedef struct {
    int d;
    Complex c[L+1];

} polynomial;
typedef struct {
    Complex y,dy;
} jet;

Complex guess(Real rho){
    Real r=rho*random()/RAND_MAX;
    Real theta=2*M_PI*random()/RAND_MAX;
    return r*cexp(I*theta);
}

Real getrho(polynomial *p){
    Real cmax=0.0;
    int i;
    for(i=0;i<p->d;i++){
        Real r=cabsl(p->c[i]);
        if(r>cmax) cmax=r;
    }
    return cmax/cabsl(p->c[p->d])+1.0;
}
jet polyjet(polynomial *p,Complex z){
    int j;
     jet r;
    r.y=p->c[p->d];
    r.dy=0;
    for(j=1;j<=p->d;j++){
        r.dy=r.dy*z+r.y;
        r.y=r.y*z+p->c[p->d-j];
    }
    return r;
```

```
84 }
85 polynomial deflate(polynomial *p,Complex z){
86     polynomial q;
87     int i,j;
88     q.d=p->d-1;
89     q.c[q.d]=p->c[p->d];
90     for(j=1;j<=q.d;j++) q.c[q.d-j]=q.c[p->d-j]*z+p->c[p->d-j];
91     return q;
92 }
93 Complex findroot(polynomial *p){
94     Complex z[N];
95     Real phi=(1.0+sqrt(5.0))/2.0;
96     Real rhomin=pow(2.0,-54.0);
97     Real rhomax=phi*getrho(p);
98     int i,j,k;
99     z[0]=guess(0.0);
100    Real rho=rhomin;
101    for(k=0;k<K;k++){
102        for(i=1;i<N;i++){
103            jet r=polyjet(p,z[i-1]);
104            z[i]=z[i-1]-r.y/r.dy;
105            if(z[i]==z[i>>1]) return z[i];
106            if(fetestexcept(FE_BAD)){
107                feclearexcept(FE_BAD);
108                break;
109            }
110            if(cabsl(z[i])>rhomax) break;
111        }
112        z[0]=guess(rho);
113        rho*=phi;
114        if(rho>rhomax) rho=rhomin;
115    }
116    printf("Failed to find a root!\n");
117    exit(4);
118 }
119 Complex pollish(polynomial *p,Complex z0){
120    Complex z=z0;
121    int i;
122    for(i=0;i<P;i++){
123        jet r=polyjet(p,z);
124        z=z-r.y/r.dy;
125    }
126    if(fetestexcept(FE_BAD)){
127        feclearexcept(FE_BAD);
128        puts("Floating point exception while pollishing!");
129        exit(3);
130    }
131    return z;
132 }
133 int cmpmodulus(const void *z1,const void *z2){
134    Real r1=cabsl(*(Complex *)z1);
135    Real r2=cabsl(*(Complex *)z2);
136    if(r1>r2) return 1;
137    if(r1<r2) return -1;
138    return 0;
```

```
139 }
140
141 int main(){
142     printf(
143         "roots -- Newton's method to find polynomial roots Version 2\n"
144         "Written September 25, 2012 for Math 701 by Eric Olson\n\n");
145     int k;
146     for(k=1;;k++){
147         Complex z[L];
148         polynomial p;
149         int i,cflag;
150         if(scanf("%d",&i)!=1) exit(0);
151         if(i<0) exit(0);
152         if(i>1000) {
153             p.d=i-1000; cflag=1;
154         } else {
155             p.d=i; cflag=0;
156         }
157         if(p.d>L) {
158             puts("Memory exceeded!");
159             exit(1);
160         }
161         for(i=0;i<=p.d;i++){
162             Real x,y;
163             if(scanf("%Lg",&x)!=1){
164                 puts("Incomplete input data!");
165                 exit(2);
166             }
167             if(cflag) {
168                 if(scanf("%Lg",&y)!=1){
169                     puts("Incomplete complex number!");
170                     exit(5);
171                 }
172             } else y=0;
173             p.c[i]=x+I*y;
174         }
175         for(i=0;i<p.d;i++){
176             int j;
177             polynomial q=p;
178             for(j=0;j<i;j++) q=deflate(&q,z[j]);
179             z[i]=pollish(&p,findroot(&q));
180             qsort(z,i,sizeof(Complex),cmpmodulus);
181         }
182         printf("Coefficents of polynomial %d:\n",k);
183         printf("%4s %24s %24s\n","n","Re(Cn)","Im(Cn)");
184         for(i=0;i<=p.d;i++)
185             printf("%4d %24.15Le %24.15Le\n",
186                 i,creall(p.c[i]),cimagl(p.c[i]));
187         printf("\nRoots of polynomial %d:\n",k);
188         printf("%4s %24s %24s\n","n","Re(Zn)","Im(Zn)");
189         for(i=0;i<p.d;i++)
190             printf("%4d %24.15Le %24.15Le\n",
191                 i+1,creall(z[i]),cimagl(z[i]));
192         printf("\n");
193     }
```

```
194 }
```

and Wilkinson's polynomial `wilkpoly.dat` is given by

```
1  20
2  2432902008176640000
3  -8752948036761600000
4  13803759753640704000
5  -12870931245150988800
6  8037811822645051776
7  -3599979517947607200
8  1206647803780373360
9  -311333643161390640
10 63030812099294896
11 -10142299865511450
12 1307535010540395
13 -135585182899530
14 11310276995381
15 -756111184500
16 40171771630
17 -1672280820
18 53327946
19 -1256850
20 20615
21 -210
22 1
23
24 -1
```

Running the program with the command

```
$ ./roots <wilkpoly.dat
```

produces the output

```
roots -- Newton's method to find polynomial roots Version 2
Written September 25, 2012 for Math 701 by Eric Olson

Coefficents of polynomial 1:
   n                  Re(Cn)                  Im(Cn)
   0    2.432902008176640e+18    0.000000000000000e+00
   1   -8.752948036761600e+18    0.000000000000000e+00
   2    1.380375975364070e+19    0.000000000000000e+00
   3   -1.287093124515099e+19    0.000000000000000e+00
   4    8.037811822645052e+18    0.000000000000000e+00
   5   -3.599979517947607e+18    0.000000000000000e+00
   6    1.206647803780373e+18    0.000000000000000e+00
   7   -3.113336431613906e+17    0.000000000000000e+00
   8    6.303081209929490e+16    0.000000000000000e+00
   9   -1.014229986551145e+16    0.000000000000000e+00
  10    1.307535010540395e+15    0.000000000000000e+00
  11   -1.355851828995300e+14    0.000000000000000e+00
  12    1.131027699538100e+13    0.000000000000000e+00
  13   -7.561111845000000e+11    0.000000000000000e+00
  14    4.017177163000000e+10    0.000000000000000e+00
  15   -1.672280820000000e+09    0.000000000000000e+00
  16    5.332794600000000e+07    0.000000000000000e+00
```

```
  17   -1.256850000000000e+06    0.000000000000000e+00
  18    2.061500000000000e+04    0.000000000000000e+00
  19   -2.100000000000000e+02    0.000000000000000e+00
  20    1.000000000000000e+00    0.000000000000000e+00

Roots of polynomial 1:
   n                   Re(Zn)                   Im(Zn)
   1    1.000000000000000e+00    0.000000000000000e+00
   2    1.999999999999999e+00   -0.000000000000000e+00
   3    3.000000000000035e+00    0.000000000000000e+00
   4    3.999999999998008e+00   -0.000000000000000e+00
   5    5.000000000023786e+00    0.000000000000000e+00
   6    6.000000000154543e+00   -0.000000000000000e+00
   7    6.999999994901812e+00    0.000000000000000e+00
   8    7.999999968307939e+00   -0.000000000000000e+00
   9    8.999999979100598e+00    0.000000000000000e+00
  10    9.999999912241915e+00   -0.000000000000000e+00
  11    1.100000027036567e+01    0.000000000000000e+00
  12    1.199999918601061e+01   -0.000000000000000e+00
  13    1.300000450574287e+01    0.000000000000000e+00
  14    1.400000206872498e+01    0.000000000000000e+00
  15    1.500000473937224e+01    0.000000000000000e+00
  16    1.600000224002417e+01   -0.000000000000000e+00
  17    1.699999906543233e+01    0.000000000000000e+00
  18    1.799999979181205e+01   -0.000000000000000e+00
  19    1.900000006055330e+01    0.000000000000000e+00
  20    1.999999998278379e+01   -0.000000000000000e+00
```

The roots are sorted and the polynomial is deflated in order of roots in increasing magnitude to reduce rounding error. To further reduce rounding error, each root obtained using Newton's method on a reduced polynomial has been refined by applying Newton's method to the original polynomial. However, note that although the `long double complex` data type used in this program retains over 18 digits of precision the resulting roots are accurate to only 7 significant digits. This is because the conditioning number corresponding to the problem of finding the roots of a polynomial is very large.

11. [Kincaid and Cheney Problem 3.6#1] Solve the system of equations

$$\begin{cases} x - 2y + y^2 + y^3 - 4 = 0 \\ -x - y + 2y^2 - 1 = 0 \end{cases}$$

by the homotopy method starting with the point $(x_0, y_0) = (0, 0)$. Perform all calculations without recourse to numerical methods.

Following Kincaid and Cheney Example 2 on page 133 we write

$$\begin{aligned} h(t, x, y) &= tf(x, y) + (1 - t)\big(f(x, y) - f(x_0, y_0)\big) \\ &= f(x, y) + (1 - t)f(0, 0) = \begin{bmatrix} x - 2y + y^2 + y^3 - 8 - 4t \\ -x - y + 2y^2 - 2 - t \end{bmatrix}. \end{aligned}$$

Therefore

$$\frac{dh}{ds} = Dh\frac{d}{ds}\begin{bmatrix} t \\ x \\ y \end{bmatrix} = \begin{bmatrix} -4 & 1 & 3y^2 + 2y - 2 \\ -1 & -1 & 4y - 1 \end{bmatrix}\begin{bmatrix} t' \\ x' \\ y' \end{bmatrix} = 0.$$

We use Cramer's rule via Lemma 1 on page 135 to choose a solution $\big(t(s), x(s), y(s)\big)$ to this equation that is continuous in s. We obtain the system

$$\begin{aligned} t' &= \det\begin{bmatrix} 1 & 3y^2 + 2y - 2 \\ -1 & 4y - 1 \end{bmatrix} = 3y^2 + 6y - 3 \\ x' &= -\det\begin{bmatrix} -4 & 3y^2 + 2y - 2 \\ -1 & 4y - 1 \end{bmatrix} = -3y^2 + 14y - 2 \\ y' &= \det\begin{bmatrix} -4 & 1 \\ -1 & -1 \end{bmatrix} = 5 \end{aligned}$$

This system of ordinary differential equations is triangular and solved by integrating first for y, then for x and finally for t. The computation is

$$\begin{aligned} y(s) &= y_0 + \int_0^s 5\,du = 5s \\ x(s) &= x_0 + \int_0^s \big(-3(5u)^2 + 14(5u) - 2\big)du = -25s^3 + 35s^2 - 2s \\ t(s) &= \int_0^s \big(3(5u)^2 + 6(5u) - 3\big)du = 25s^3 + 15s^2 - 3s \end{aligned}$$

Now solve for s such that $t(s) = 1$. We obtain

$$25s^3 + 15s^2 - 3s - 1 = 0$$

which we first try to factor over $\mathbf{Q}$. By the rational root theorem the possible roots are

$$\Big\{\pm 1, \pm\frac{1}{5}, \pm\frac{1}{25}\Big\}.$$

Testing these yields that $s_1 = -1/5$ is a root so we obtain

$$(5s+1)(5s^2+2s-1) = 0.$$

The remaining roots can be found by the quadratic formula as

$$s_2 = \frac{-2+\sqrt{4+20}}{10} = \frac{-1+\sqrt{6}}{5} \qquad \text{and} \qquad s_3 = \frac{-1-\sqrt{6}}{5}.$$

The corresponding solutions are

$$\begin{aligned}
\big(x(s_1), y(s_1)\big) &= (2, -1) \\
\big(x(s_2), y(s_2)\big) &= \big(14 - 5\sqrt{6}, -1 + \sqrt{6}\,\big) \\
\big(x(s_3), y(s_3)\big) &= \big(14 + 5\sqrt{6}, -1 - \sqrt{6}\,\big).
\end{aligned}$$

We now check the solutions using Maple. The Maple script is

```
1  restart;
2  f1:=(x,y)->x-2*y+y^2+y^3-4;
3  f2:=(x,y)->-x-y+2*y^2-1;
4
5  x1:=2; y2:=-1;
6  'f1(x1,y1)'=simplify(f1(x1,y1));
7  'f2(x1,y1)'=simplify(f2(x1,y1));
8
9  x2:=14-5*sqrt(6); y2:=-1+sqrt(6);
10 'f1(x2,y2)'=simplify(f1(x2,y2));
11 'f2(x2,y2)'=simplify(f2(x2,y2));
12
13 x3:=14+5*sqrt(6); y3:=-1-sqrt(6);
14 'f1(x3,y3)'=simplify(f1(x3,y3));
15 'f2(x3,y3)'=simplify(f2(x3,y3));
```

and the output is

```
                                    2    3
f1 := (x, y) -> x - 2 y + y  + y  - 4

                                   2
  f2 := (x, y) -> -x - y + 2 y  - 1

                x1 := 2

                y2 := -1

                                  2     3
  f1(x1, y1) = -2 - 2 y1 + y1  + y1

                                   2
     f2(x1, y1) = -3 - y1 + 2 y1

                          1/2
          x2 := 14 - 5 6
```

```
                1/2
y2 := -1 + 6

 f1(x2, y2) = 0

 f2(x2, y2) = 0

                 1/2
x3 := 14 + 5 6

                1/2
y3 := -1 - 6

 f1(x3, y3) = 0

 f2(x3, y3) = 0
```

which shows we have indeed found three roots.

12. [Kincaid and Cheney Problem 3.6#2] Consider the homotopy

$$h(t,x) = tf(x) + (1-t)g(x)$$

in which

$$f(x) = x^2 - 5x + 6 \qquad \text{and} \qquad g(x) = x^2 - 1.$$

Show that there is no real path connecting a root of g to a root of f.

If there were such a path then it would have to pass through a point when $t = 1/2$ of the form $(1/2, x)$ where $h(1/2, x) = 0$. However

$$h(1/2,x) = \frac{1}{2}(x^2 - 5x + 6) + \frac{1}{2}(x^2 - 1) = \frac{1}{2}(2x^2 - 5x + 5) = 0$$

implies by the quadratic formula that

$$x = \frac{5 \pm \sqrt{25 - 40}}{4} \notin \mathbf{R}.$$

Therefore there is no real path connecting a roots of g to a root of f.