Math 701 Programming Project 1

Newton's Method: Solution Key

1. Consider Newton's method for solving $f(x) = 0$ where $f(x) = x^2 - 2$ using the starting point $x_0 = 1$.

(i) Let $e_n = x_n - \sqrt{2}$ and create a table with three columns showing n, x_n and e_n for $n = 0, 1, \ldots, 8$.

The program is

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

double f(double x){
    return x*x*x-3;
}
double df(double x){
    return 3*x*x;
}
double g(double x){
    return x-f(x)/df(x);
}
int main(){
    printf("#p1i.c\n");
    double x=1,xinf=sqrt(2.0);
    printf("#%2s %22s %22s\n","n","xn","en");
    for(int i=0;;i++){
        printf("%3d %22.14e %22.14e\n",i,x,x-xinf);
        if(i>=8) break;
        x=g(x);
    }
    return 0;
}
```

The output from running the program was

```
#p1i.c
# n                     xn                     en
  0   1.00000000000000e+00  -4.14213562373095e-01
  1   1.66666666666667e+00   2.52453104293571e-01
  2   1.47111111111111e+00   5.68975487380159e-02
  3   1.44281209824934e+00   2.85985358762482e-02
  4   1.44224978959900e+00   2.80362272259045e-02
  5   1.44224957030744e+00   2.80360079343465e-02
  6   1.44224957030741e+00   2.80360079343132e-02
  7   1.44224957030741e+00   2.80360079343132e-02
  8   1.44224957030741e+00   2.80360079343132e-02
```

(ii) A sign of quadratic convergence is that the number of significant digits double at each iteration. Does that happen in this case?

The number of significant digits in the approximation x_n is defined as the largest nonnegative integer k such that

$$\frac{|x_n - \sqrt{2}|}{|\sqrt{2}|} \le 5 \times 10^{-k}.$$

For reference this is Definition 1.16 on page 18 in Burden, Faires and Burden. We modify the program to report number of significant digits in each iteration. The program is

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

double f(double x){
    return x*x-2;
}
double df(double x){
    return 2*x;
}
double g(double x){
    return x-f(x)/df(x);
}
unsigned sigdig(double ps,double p){
    double relerror=fabs((p-ps)/p);
    return (int)ceil(-log(relerror/5)/log(10));
}
int main(){
    printf("#p1ii.c\n");
    double x=1,xinf=sqrt(2.0);
    printf("#%2s %22s %22s %12s\n","n","xn","en","sigdig");
    for(int i=0;;i++){
        printf("%3d %22.14e %22.14e %12u\n",i,x,x-xinf,sigdig(x,xinf));
        if(i>=8) break;
        x=g(x);
    }
    return 0;
}
```

The output from running the program was

```
#p1ii.c
# n                   xn                     en      sigdig
  0   1.00000000000000e+00  -4.14213562373095e-01           2
  1   1.50000000000000e+00   8.57864376269049e-02           2
  2   1.41666666666667e+00   2.45310429357160e-03           4
  3   1.41421568627451e+00   2.12390141474117e-06           7
  4   1.41421356237469e+00   1.59472435257157e-12          13
  5   1.41421356237310e+00   0.00000000000000e+00  2147483648
  6   1.41421356237309e+00  -2.22044604925031e-16          17
  7   1.41421356237310e+00   0.00000000000000e+00  2147483648
  8   1.41421356237309e+00  -2.22044604925031e-16          17
```

For the range $n = 1, 2, 3, 4$ the number of significant digits approximately doubles from iteration to iteration. This can also be seen from the fact that the exponents

`e-02`, `e-03`, `e-06`, `e-12`

appearing in the exponential notation representation of e_n approximately double from iteration to iteration.

Note that number of significant digits don't double between the $n = 0$ and $n = 1$ iterations, because x_0 isn't close enough to $\sqrt{2}$ that the asymptotic regime is in effect. Moreover, for iterations $n = 5$ and greater, further quadratic convergence is prevented by rounding error present in the double precision arithmetic used for the computation.

(iii) Comment on how rounding error effects the numerical convergence of Newton's method.

In general Newton's method is resistant to rounding errors, because any errors made in earlier iterations are corrected by subsequent iterations. Thus, there is no accumulation of rounding errors. On the other hand, rounding errors can also prevent the convergence of x_n to a fixed point. In this example, rather than converging to a fixed point, the algorithm converges to a two cycle as $n \to \infty$. In particular, if n is large, then

$$x_n = \begin{cases} \texttt{1.41421356237309e+00} & \text{for } n \text{ even} \\ \texttt{1.41421356237310e+00} & \text{for } n \text{ odd} \end{cases}$$

Essentially the last bit in our floating point approximations of $\sqrt{2}$ flips back and forth between iterations. This behavior is very different than the analytical result of converging to a fixed point in the absence of rounding error and needs to be taken into account when coding stopping conditions for practical problems.

(iv) Write $|e_{n+1}| = M_n|e_n|^2$ and compute M_n for $n = 1, 2, 3$, and 4. In this case is M_n bigger or less than 1?

The program to compute M_n is

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

double f(double x){
    return x*x-2;
}
double df(double x){
    return 2*x;
}
double ddf(double x){
    return 2;
}
double g(double x){
    return x-f(x)/df(x);
}
int main(){
    printf("#p1iii.c\n");
    double x=1,xinf=sqrt(2.0);
    printf("#%2s %22s %22s %22s\n","n","xn","en","Mn");
    for(int i=0;i<=4;i++){
        double eold=x-xinf;
        double y=g(x);
        double enew=y-xinf;
        double M=fabs(enew/eold/eold);
        printf("%3d %22.14e %22.14e %22.14e\n",i,x,eold,M);
        x=y;
    }
    return 0;
}
```

and it produces the output

```
#p1iii.c
# n                 xn                     en                     Mn
  0   1.00000000000000e+00  -4.14213562373095e-01   4.99999999999999e-01
  1   1.50000000000000e+00   8.57864376269049e-02   3.33333333333331e-01
  2   1.41666666666667e+00   2.45310429357160e-03   3.52941176468277e-01
  3   1.41421568627451e+00   2.12390141474117e-06   3.53522384487246e-01
  4   1.41421356237469e+00   1.59472435257157e-12   0.00000000000000e+00
```

Therefore, the values of M_n for $n = 1, 2, 3, 4$ are less than 1.

The fact that M_n is not extremely large helps explains why the number of significant digits nearly doubles between each iteration. Suppose the approximation x_n was good to k significant digits. Then

$$|e_n/\sqrt{2}| \le 5 \times 10^{-k}.$$

Since

$$\begin{aligned}|e_{n+1}/\sqrt{2}| = M_n|e_n|^2/\sqrt{2} &\le (\texttt{3.53522384487246e-01})(\sqrt{2})\big(5 \times 10^{-k}\big)^2 \\ &\le 5 \times 2.49978075372170 \times 10^{-2k} \le 5 \times 10^{0.4-2k},\end{aligned}$$

it follows that x_n is good to at least $2k - 0.4$ significant digits. As 0.4 is small compared to even a few significant digits, then in this case, quadratic convergence is observed from nearly the first iteration.

(v) [Extra Credit and for Math/CS 666] Use multi-precision arithmetic with at least 10 000 digits precision to determine the asymptotic value of M_n when n is large. Can you also find this value analytically?

It is easiest to code this in Maple as

```
restart;
kernelopts(printbytes=false):
printf("#p1v.mpl\n");
Digits:=10000:
f:=x->x^2-2:
g:=x->x-f(x)/D(f)(x):
xn:=1.0:
xinf:=sqrt(2.0):
printf("#%2s %22s %50s\n","n","en","Mn");
for i from 0 to 12
do
   eold:=xn-xinf;
   xn:=g(xn);
   enew:=xn-xinf;
   M:=abs(enew/eold^2);
   printf("%3d %22.13e %50.43e\n",i,eold,M);
end:
```

which produces the output

```
#p1v.mpl
# n                     en                                                 Mn
  0   -4.1421356237310e-01  5.0000000000000000000000000000000000000000000e-01
  1    8.5786437626905e-02  3.3333333333333333333333333333333333333333333e-01
  2    2.4531042935716e-03  3.5294117647058823529411764705882352941176471e-01
  3    2.1239014147551e-06  3.5355285961871750433275563258232235701906412e-01
  4    1.5948618246069e-12  3.5355339059287504674427091702873139427835106e-01
  5    8.9929283216505e-25  3.5355339059327376220042195622921647838109046e-01
  6    2.8592838433340e-49  3.5355339059327376220042218105242451964241797e-01
  7    2.8904771932154e-98  3.5355339059327376220042218105242451964241797e-01
  8   2.9538885168370e-196  3.5355339059327376220042218105242451964241797e-01
  9   3.0849150376058e-392  3.5355339059327376220042218105242451964241797e-01
 10   3.3646618312998e-784  3.5355339059327376220042218105242451964241797e-01
 11  4.0025599881848e-1568  3.5355339059327376220042218105242451964241797e-01
 12  5.6640973065394e-3136  3.5355339059327376220042218105242451964241797e-01
```

Note that, although the computation is performed using 10 000 digits of precision, for economy of space we only print the first 44 digits of each number. The fact that we are working with very-high precision arithmetic is reflected by the tabulated values of e_n whose exponents continue to double over 12 iterations. It is evident that M_n has converged to a number close to 0.35. Analytically we compute

$$\lim_{n\to\infty} \frac{e_{n+1}}{e_n^2} = \lim_{n\to\infty} \frac{g(x_n)-\sqrt{2}}{(x_n-\sqrt{2})^2} = \lim_{x\to\sqrt{2}} \frac{g(x)-\sqrt{2}}{(x-\sqrt{2})^2}$$

Since $g(x) \to \sqrt{2}$ and $x \to \sqrt{2}$ as $x \to \sqrt{2}$ then both the numerator and denominator vanish in the limit. Therefore, by L'Hôpital's rule

$$\lim_{n\to\infty} \frac{e_{n+1}}{e_n^2} = \lim_{x\to\sqrt{2}} \frac{g'(x)}{2(x-\sqrt{2})}.$$

Since $f'(\sqrt{2}) \neq 0$ we know that

$$g'(x) = 1 - \frac{f'(x)^2 - f(x)f''(x)}{f'(x)^2} = \frac{f(x)f''(x)}{f'(x)^2} \to 0 \qquad \text{as} \qquad x \to \sqrt{2}.$$

Consequently, we may apply L'Hôpital's rule again to obtain

$$\lim_{n\to\infty} \frac{e_{n+1}}{e_n^2} = \lim_{x\to\sqrt{2}} \frac{g''(x)}{2} = \frac{g''(\sqrt{2})}{2}.$$

For the function $f(x) = x^2 - 2$ considered here

$$g''(x) = \frac{d}{dx}\frac{x^2-2}{2x^2} = \frac{d}{dx}\Big(\frac{1}{2} - \frac{1}{x^2}\Big) = 2x^{-3}.$$

Therefore

$$\lim_{n\to\infty} M_n = \Big|\frac{g''(\sqrt{2})}{2}\Big| = \frac{1}{2\sqrt{2}}.$$

We finish by providing an alternative code written in C using the mpfr library that computes using 10 000 digits of precision. The alternative code is

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <gmp.h>
#include <mpfr.h>

// 33220 bits = 10000 decimal digits
#define N 33220

mpfr_t rf,rdf,rg;
mpfr_ptr f(mpfr_t x){
    mpfr_mul(rf,x,x,MPFR_RNDN);
    mpfr_sub_ui(rf,rf,2,MPFR_RNDN);
    return rf;
}
mpfr_ptr df(mpfr_t x){
    mpfr_mul_ui(rdf,x,2,MPFR_RNDN);
    return rdf;
}
mpfr_ptr g(mpfr_t x){
    mpfr_ptr y=f(x), dy=df(x);
    mpfr_div(rg,y,dy,MPFR_RNDN);
    mpfr_sub(rg,x,rg,MPFR_RNDN);
    return rg;
}

int main(){
    printf("#p1v2.c\n");
    mpfr_init2(rf,N); mpfr_init2(rdf,N);
    mpfr_init2(rg,N);

    mpfr_t xn,xinf,eold,enew,M;
    mpfr_init2(xn,N); mpfr_init2(xinf,N);
    mpfr_init2(eold,N); mpfr_init2(enew,N);
    mpfr_init2(M,N);
    mpfr_set_ui(xn,1,MPFR_RNDN);          // xn=1
    mpfr_sqrt_ui(xinf,2,MPFR_RNDN);       // xinf=sqrt(2)

    int digits=N*log(2.0)/log(10.0);
//  printf("Using %d digit arithmetic...\n",digits);
    printf("#%2s %22s %50s\n","n","en","Mn");
    for(int i=0;i<=12;i++){
        mpfr_sub(eold,xn,xinf,MPFR_RNDN);  // eold=xn-xinf
        mpfr_ptr yn=g(xn);                 // yn=g(xn)
        mpfr_sub(enew,yn,xinf,MPFR_RNDN);  // enew=yn-xinf
        mpfr_mul(M,eold,eold,MPFR_RNDN);   // M=abs(enew/enew^2)
        mpfr_div(M,enew,M,MPFR_RNDN);
        mpfr_abs(M,M,MPFR_RNDN);
        mpfr_set(xn,yn,MPFR_RNDN);         // xn=yn
        mpfr_printf("%3d %22.13Re %50.43Re\n",i,eold,M);
    }
    return 0;
}
```

and the output is

```
#p1v2.c
# n                    en                                               Mn
  0   -4.1421356237310e-01  5.0000000000000000000000000000000000000000000e-01
  1    8.5786437626905e-02  3.3333333333333333333333333333333333333333333e-01
  2    2.4531042935716e-03  3.5294117647058823529411764705882352941176471e-01
  3    2.1239014147551e-06  3.5355285961871750433275563258232235701906412e-01
  4    1.5948618246069e-12  3.5355339059287504674427091702873139427835106e-01
  5    8.9929283216505e-25  3.5355339059327376220042195622921647838109046e-01
  6    2.8592838433340e-49  3.5355339059327376220042218105242451964241797e-01
  7    2.8904771932154e-98  3.5355339059327376220042218105242451964241797e-01
  8   2.9538885168370e-196  3.5355339059327376220042218105242451964241797e-01
  9   3.0849150376058e-392  3.5355339059327376220042218105242451964241797e-01
 10   3.3646618312998e-784  3.5355339059327376220042218105242451964241797e-01
 11  4.0025599881848e-1568  3.5355339059327376220042218105242451964241797e-01
 12  5.6640973065394e-3136  3.5355339059327376220042218105242451964241797e-01
```

2. Consider the secant method for solving $f(x) = 0$ where $f(x) = x^2 - 2$ using the starting points $x_0 = 0$ and $x_1 = 1$.

(i) Let $e_n = x_n - \sqrt{2}$ and create a table with three columns showing n, x_n and e_n for $n = 0, 1, \ldots, 8$.

The C code is

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

double f(double x){
    return x*x-2;
}
double g(double x,double xold){
    return x-f(x)*(x-xold)/(f(x)-f(xold));
}
int main(){
    printf("#p2i.c\n");
    double x=1,xold=0,xinf=sqrt(2.0);
    printf("#%2s %22s %22s\n","n","xn","en");
    for(int i=0;;i++){
        printf("%3d %22.14e %22.14e\n",i,xold,xold-xinf);
        if(i>=8) break;
        double y=g(x,xold);
        xold=x;
        x=y;
    }
    return 0;
}
```

and the output is

```
#p2i.c
# n                     xn                     en
  0   0.00000000000000e+00  -1.41421356237310e+00
  1   1.00000000000000e+00  -4.14213562373095e-01
  2   2.00000000000000e+00   5.85786437626905e-01
  3   1.33333333333333e+00  -8.08802290397617e-02
  4   1.40000000000000e+00  -1.42135623730950e-02
  5   1.41463414634146e+00   4.20583968368193e-04
  6   1.41421143847487e+00  -2.12389822507042e-06
  7   1.41421356205732e+00  -3.15774739689800e-10
  8   1.41421356237310e+00   2.22044604925031e-16
```

(ii) A sign of quadratic convergence is that the number of significant digits double at each iteration. Does that happen in this case?

Making similar changed to the case as done for question 1(i) we obtain

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

double f(double x){
    return x*x-2;
}
double g(double x,double xold){
    return x-f(x)*(x-xold)/(f(x)-f(xold));
}
unsigned sigdig(double ps,double p){
    double relerror=fabs((p-ps)/p);
    return (int)ceil(-log(relerror/5)/log(10));
}
int main(){
    printf("#p2ii.c\n");
    double x=1,xold=0,xinf=sqrt(2.0);
    printf("#%2s %22s %22s %12s\n","n","xn","en","sigdig");
    for(int i=0;;i++){
        printf("%3d %22.14e %22.14e %12d\n",i,
            xold,xold-xinf,sigdig(xold,xinf));
        if(i>=8) break;
        double y=g(x,xold);
        xold=x;
        x=y;
    }
    return 0;
}
```

with output given by

```
#p2ii.c
# n                     xn                     en       sigdig
  0   0.00000000000000e+00  -1.41421356237310e+00            1
  1   1.00000000000000e+00  -4.14213562373095e-01            2
  2   2.00000000000000e+00   5.85786437626905e-01            2
  3   1.33333333333333e+00  -8.08802290397617e-02            2
  4   1.40000000000000e+00  -1.42135623730950e-02            3
  5   1.41463414634146e+00   4.20583968368193e-04            5
  6   1.41421143847487e+00  -2.12389822507042e-06            7
  7   1.41421356205732e+00  -3.15774739689800e-10           11
  8   1.41421356237310e+00   2.22044604925031e-16           17
```

With this data, it is difficult to tell whether the number of significant digits is essentially doubling or not at each iteration. The rate of convergence is notably slower than Newton's method and we know, from the next problem, that asymptotically the rate is not quadratic but $\alpha \approx 1.618$. Therefore, there are theoretical reasons to believe that the number of significant digits is not doubling at each iteration.

While not necessary for a complete answer to this question, a Maple code was written to compute with more precision. The code

```
restart;
kernelopts(printbytes=false):
printf("#p2ii2.mpl\n");
Digits:=10000:
sigdig:=proc(ps,p)
    local relerror;
    relerror:=abs((p-ps)/p);
    ceil(-log(relerror/5.0)/log(10.0));
end proc:
f:=x->x^2-2:
g:=(x,xold)->x-f(x)*(x-xold)/(f(x)-f(xold)):
xn:=1.0:
xold:=0.0:
xinf:=sqrt(2.0):
printf("#%2s %22s %24s %10s\n","n","xn","en","sigdig");
i:=0:
while(true)
do
    printf("%3d %22.13e %24.13e %10d\n",i,
        xold,xold-xinf,sigdig(xold,xinf));
    eold:=xn-xinf;
    if i>=19 then
        break;
    end;
    yn:=g(xn,xold);
    xold:=xn;
    xn:=yn;
    i:=i+1;
end:
```

produces the output

```
#p2ii2.mpl
# n                     xn                       en     sigdig
  0    0.0000000000000e+00     -1.4142135623731e+00          1
  1    1.0000000000000e+00     -4.1421356237310e-01          2
  2    2.0000000000000e+00      5.8578643762690e-01          2
  3    1.3333333333333e+00     -8.0880229039762e-02          2
  4    1.4000000000000e+00     -1.4213562373095e-02          3
  5    1.4146341463415e+00      4.2058396836837e-04          5
  6    1.4142114384749e+00     -2.1238982250315e-06          7
  7    1.4142135620573e+00     -3.1577458617355e-10         11
```

```
   8    1.4142135623731e+00      2.3711892058579e-16         17
   9    1.4142135623731e+00     -2.6472709293147e-26         27
  10    1.4142135623731e+00     -2.2193183616621e-42         43
  11    1.4142135623731e+00      2.0771746000879e-68         69
  12    1.4142135623731e+00    -1.6298499226021e-110        111
  13    1.4142135623731e+00    -1.1969489443672e-178        179
  14    1.4142135623731e+00     6.8972862240904e-289        290
  15    1.4142135623731e+00    -2.9188305375427e-467        468
  16    1.4142135623731e+00    -7.1177402737057e-756        756
  17    1.4142135623731e+00    7.3452405711368e-1223       1223
  18    1.4142135623731e+00   -1.8484306763933e-1978       1979
  19    1.4142135623731e+00   -4.8002537800567e-3201       3202
```

From this high-precision output, it is clear that the number of significant digits is not doubling, even at the later iterations.

(iii) According to Wikipedia `https://en.wikipedia.org/wiki/Secant_method` the order of convergence of the secant method is

$$\alpha = \frac{1+\sqrt{5}}{2} \approx 1.618,$$

which is less than quadratic. Write $|e_{n+1}| = M_n |e_n|^\alpha$ and compute M_n for $n = 1, 2, \ldots, 7$. In this case is M_n bigger or less than 1?

The code is

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

double f(double x){
    return x*x-2;
}
double g(double x,double xold){
    return x-f(x)*(x-xold)/(f(x)-f(xold));
}
unsigned sigdig(double ps,double p){
    double relerror=fabs((p-ps)/p);
    return (int)ceil(-log(relerror/5)/log(10));
}
int main(){
    printf("#p2iii.c\n");
    double alpha=(1+sqrt(5.0))/2;
    double x=1,xold=0,xinf=sqrt(2.0);
    printf("#%2s %22s %22s %22s\n","n","xn","en","Mn");
    for(int i=0;i<=7;i++){
        double eold=x-xinf;
        double y=g(x,xold);
        double enew=y-xinf;
        double M=fabs(enew)/pow(fabs(eold),alpha);
        printf("%3d %22.14e %22.14e %22.14e\n",i,x,eold,M);
        xold=x;
        x=y;
    }
    return 0;
}
```

and the output is

```
#p2iii.c
# n                     xn                     en                     Mn
  0   1.00000000000000e+00  -4.14213562373095e-01   2.43827890556285e+00
  1   2.00000000000000e+00   5.85786437626905e-01   1.92153393677974e-01
  2   1.33333333333333e+00  -8.08802290397617e-02   8.31478649741950e-01
  3   1.40000000000000e+00  -1.42135623730950e-02   4.10055031684882e-01
  4   1.41463414634146e+00   4.20583968368193e-04   6.16388831558557e-01
  5   1.41421143847487e+00  -2.12389822507042e-06   4.76724936560529e-01
  6   1.41421356205732e+00  -3.15774739689800e-10   5.23323852141357e-01
  7   1.41421356237310e+00   2.22044604925031e-16   0.00000000000000e+00
```

Note that M_n is less than 1.

From this data it is difficult to determine whether M_n is, in fact, converging to a limit. While not necessary for a complete answer to this question, again a Maple code was written to compute with more precision. The code

```
restart;
kernelopts(printbytes=false):
Digits:=10000:
printf("#p2iii2.mpl\n");
sigdig:=proc(ps,p)
    local relerror;
    relerror:=abs((p-ps)/p);
    ceil(-log(relerror/5.0)/log(10.0));
end proc:
f:=x->x^2-2:
g:=(x,xold)->x-f(x)*(x-xold)/(f(x)-f(xold)):
alpha:=(1.0+sqrt(5.0))/2.0:
xn:=1.0:
xold:=0.0:
xinf:=sqrt(2.0):
printf("#%2s %22s %22s %22s\n","n","xn","en","Mn");
for i from 0 to 19
do
    eold:=xn-xinf;
    yn:=g(xn,xold);
    enew:=yn-xinf;
    M:=abs(enew)/abs(eold)^alpha;
    printf("%3d %22.14e %22.12e %22.14e\n",i,xn,eold,M);
    xold:=xn;
    xn:=yn;
end:
```

produces the output

```
#p2iii2.mpl
# n                     xn                     en                     Mn
  0   1.00000000000000e+00    -4.142135623731e-01   2.43827890556285e+00
  1   2.00000000000000e+00     5.857864376269e-01   1.92153393677974e-01
  2   1.33333333333333e+00    -8.088022903976e-02   8.31478649741951e-01
  3   1.40000000000000e+00    -1.421356237310e-02   4.10055031685049e-01
  4   1.41463414634146e+00     4.205839683684e-04   6.16388831546844e-01
  5   1.41421143847487e+00    -2.123898225031e-06   4.76724704811289e-01
  6   1.41421356205732e+00    -3.157745861736e-10   5.58852058531633e-01
```

```
 7   1.41421356237310e+00     2.371189205858e-16   5.06563786196404e-01
 8   1.41421356237310e+00    -2.647270929315e-26   5.38271204837746e-01
 9   1.41421356237310e+00    -2.219318361662e-42   5.18448282665523e-01
10   1.41421356237310e+00     2.077174600088e-68   5.30611601387976e-01
11   1.41421356237310e+00   -1.629849922602e-110   5.23060994155132e-01
12   1.41421356237310e+00   -1.196948944367e-178   5.27714745625001e-01
13   1.41421356237310e+00    6.897286224090e-289   5.24833705146293e-01
14   1.41421356237310e+00   -2.918830537543e-467   5.26612424049138e-01
15   1.41421356237310e+00   -7.117740273706e-756   5.25512405067369e-01
16   1.41421356237310e+00   7.345240571137e-1223   5.26191982663751e-01
17   1.41421356237310e+00  -1.848430676393e-1978   5.25771876953996e-01
18   1.41421356237310e+00  -4.800253780057e-3201   5.26031476954935e-01
19   1.41421356237310e+00   3.137056728066e-5179   5.25871020205594e-01
```

The values of M_n are near 0.52 from this output; however, it is notable that even though M_n seems uniformly bounded, even with 10 000 digits of precision it is not clear as $n \to \infty$ that there really is a limit.

(iv) [Extra Credit and for Math/CS 666] Prove that the order of convergence of the secant method is α. If you look the proof up, please cite your references and rewrite the proof in your own words.

We assume f is twice continuously differentiable and that x_∞ is such that $f(x_\infty) = 0$ and $f'(x_\infty) \neq 0$. The secant method is

$$x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}.$$

First claim if x_0 and x_1 with $x_0 \neq x_1$ are chosen close enough to x_∞ that $x_n \to x_\infty$ as $n \to \infty$. That is, the secant method converges. Let

$$\kappa(\alpha, \beta) = 1 - \frac{f'(\alpha)}{f'(\beta)}.$$

Since $f'(x_\infty) \neq 0$ it follows that the limit supremum of $|\kappa(\alpha,\beta)|$ is zero as $\alpha \to x_\infty$ and $\beta \to x_\infty$. Therefore, there is $\delta > 0$ so small such that

$$|\kappa(\alpha,\beta)| \le \gamma < 1 \quad \text{whenever} \quad |\alpha - x_\infty| < \delta \quad \text{and} \quad |\beta - x_\infty| < \delta.$$

Choose x_0 and x_1 such that $|x_0 - x_\infty| < \delta$ and $|x_1 - x_\infty| < \delta$. By the mean-value theorem

$$\frac{f(x_n) - f(x_\infty)}{x_n - x_\infty} = f'(a_n) \qquad \text{and} \qquad \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}} = f'(b_n)$$

for some a_n between x_n and x_∞ and for some b_n between x_n and x_{n-1}. For induction assume $|x_n - x_\infty| < \delta$ and $|x_{n-1} - x_\infty| < \delta$, in which case $|a_n - x_\infty| < \delta$ and $|b_n - x_\infty| < \delta$. Denoting $e_n = x_n - x_\infty$ we obtain

$$\begin{aligned} e_{n+1} &= e_n - f(x_n)\frac{e_n - e_{n-1}}{f(x_n) - f(x_{n-1})} \\ &= e_n - \frac{(f(x_n) - f(x_\infty))(e_n - e_{n-1})}{f(x_n) - f(x_{n-1})} \\ &= e_n - \frac{f'(a_n)e_n(e_n - e_{n-1})}{f'(b_n)(e_n - e_{n-1})} = \Big(1 - \frac{f'(a_n)}{f'(b_n)}\Big)e_n. \end{aligned}$$

Therefore, $|e_{n+1}| \le \gamma |e_n|$ and by induction $|e_{n+1}| \le \gamma^n |e_1|$. Since $\gamma < 1$, it follows that $x_n \to x_\infty$ as $n \to \infty$ and the secant method converges.

Claim there exists C such that $|e_{n+1}|/|e_n e_{n-1}| \to C$ as $n \to \infty$. First note that

$$\begin{aligned} e_{n+1} &= e_n - f(x_n)\frac{e_n - e_{n-1}}{f(x_n) - f(x_{n-1})} \\ &= e_n - \frac{f(x_n)e_n - f(x_{n-1})e_n + f(x_{n-1})e_n - f(x_n)e_{n-1}}{f(x_n) - f(x_{n-1})} \\ &= \frac{f(x_n)e_{n-1} - f(x_{n-1})e_n}{f(x_n) - f(x_{n-1})} = \frac{f(x_n)/e_n - f(x_{n-1})/e_{n-1}}{f'(b_n)(x_n - x_{n-1})}e_n e_{n-1}. \end{aligned}$$

Define

$$A = \max\{\, |f''(x)| : |x - x_\infty| \le \delta \,\}.$$

Since f'' is continuous, it follows that $A < \infty$. By Taylor's theorem we have

$$f(x_n) = f(x_\infty) + f'(x_\infty)e_n + \int_{x_\infty}^{x_n} f''(t)(x_n - t)dt$$

Now

$$\begin{aligned}\frac{f(x_n)}{e_n} - \frac{f(x_{n-1})}{e_{n-1}} &= \frac{1}{e_n}\int_{x_\infty}^{x_n} f''(t)(x_n - t)dt - \frac{1}{e_{n-1}}\int_{x_\infty}^{x_{n-1}} f''(t)(x_{n-1} - t)dt \\ &= J_1 + J_2\end{aligned}$$

where

$$J_1 = \Big(\frac{1}{e_n} - \frac{1}{e_{n-1}}\Big)\int_{x_\infty}^{x_n} f''(t)(x_n - t)dt$$

and

$$\begin{aligned}J_2 &= \frac{1}{e_{n-1}}\Big(\int_{x_\infty}^{x_n} f''(t)(x_n - t)dt - \int_{x_\infty}^{x_{n-1}} f''(t)(x_{n-1} - t)dt\Big) \\ &= J_3 + J_4\end{aligned}$$

Here

$$J_3 = \frac{x_n - x_{n-1}}{e_{n-1}}\int_{x_\infty}^{x_n} f''(t)dt \qquad \text{and} \qquad J_4 = \frac{1}{e_{n-1}}\int_{x_{n-1}}^{x_n} f''(t)(x_{n-1} - t)dt.$$

Estimate

$$\begin{aligned}|J_1| &\le A\Big|\frac{1}{e_n} - \frac{1}{e_{n-1}}\Big|\Big|\int_{x_\infty}^{x_n}(x_n - t)dt\Big| \\ &\le A\frac{|x_n - x_{n-1}|}{|e_n e_{n-1}|}\frac{|e_n|^2}{2} \le A\Big|\frac{e_n}{e_{n-1}}\Big|\frac{|x_n - x_{n-1}|}{2}\end{aligned}$$

Therefore

$$\frac{|J_1|}{|x_n - x_{n-1}|} \le \frac{A}{2}\Big|\frac{e_n}{e_{n-1}}\Big| = \frac{A}{2}\Big|1 - \frac{f'(a_n)}{f'(b_n)}\Big| \to 0 \qquad \text{as} \qquad n \to \infty.$$

Also

$$\frac{|J_3|}{|x_n - x_{n-1}|} \le A\Big|\frac{e_n}{e_{n-1}}\Big| \to 0 \qquad \text{as} \qquad n \to \infty.$$

The estimate of J_4 will be done more carefully. Consider two cases: If $x_{n-1} < x_n$ then the mean-value theorem for integrals implies

$$\frac{2}{(x_n - x_{n-1})^2} \int_{x_{n-1}}^{x_n} f''(t)(t - x_{n-1})dt = f''(\xi_n) \qquad \text{where} \qquad \xi_n \in [x_{n-1}, x_n].$$

If $x_n < x_{n-1}$ then

$$\frac{2}{(x_{n-1} - x_n)^2} \int_{x_n}^{x_{n-1}} f''(t)(x_{n-1} - t)dt = f''(\xi_n) \qquad \text{where} \qquad \xi_n \in [x_n, x_{n-1}].$$

In either case it holds that

$$\frac{J_4}{x_n - x_{n-1}} = f''(\xi_n)\frac{x_n - x_{n-1}}{2e_{n-1}} = \frac{f''(\xi_n)}{2}\Big(\frac{e_n}{e_{n-1}} + 1\Big) \to \frac{f''(x_\infty)}{2}$$

as $n \to \infty$. It follows that

$$J_2 \to \frac{f''(x_\infty)}{2} \qquad \text{as} \qquad n \to \infty.$$

Consequently

$$\Big|\frac{e_{n+1}}{e_n e_{n-1}}\Big| \to C \qquad \text{where} \qquad C = \Big|\frac{f''(x_\infty)}{2f'(x_\infty)}\Big| \qquad \text{as} \qquad n \to \infty.$$

Claim that the secant method converges with order α. Note that

$$\frac{1}{\alpha - 1} = \alpha, \qquad \alpha^2 - 1 = \alpha, \qquad \text{and} \qquad \frac{1}{\alpha} + \frac{1}{\alpha^2} = 1.$$

Define $K_n = |e_{n+1}|/|e_n e_{n-1}|$ and $M_n = |e_{n+1}|/|e_n|^\alpha$. Then

$$M_n^\alpha M_{n-1} = \Big(\frac{|e_{n+1}|^\alpha}{|e_n|^{\alpha^2}}\Big)\Big(\frac{|e_n|}{|e_{n-1}|^\alpha}\Big) = \Big(\frac{|e_{n+1}|}{|e_n e_{n-1}|}\Big)^\alpha = K_n^\alpha.$$

It follows that

$$M_n = \frac{K_n}{M_{n-1}^{1/\alpha}} \qquad \text{and similarly} \qquad M_{n+1} = \frac{K_{n+1}}{M_n^{1/\alpha}}.$$

Combining the above two inequalities implies

$$M_{n+1} = \frac{K_{n+1}}{K_n^{1/\alpha}} M_{n-1}^{1/\alpha^2}.$$

Since $K_n \to C$ as $n \to \infty$ then

$$\frac{K_{n+1}}{K_n^{1/\alpha}} \to C^{1-1/\alpha} = C^{2-\alpha} \qquad \text{as} \qquad n \to \infty.$$

Since $2 - \alpha > 0$, the above limit makes sense even when $f''(x_\infty) = 0$. Let $L = 1 + C^{2-\alpha}$. By the definition of limit, there exists N large enough such that

$$M_{n+1} \le L M_{n-1}^{1/\alpha^2} \qquad \text{for all} \qquad n \ge N.$$

The above inequality and the fact that $1/\alpha^2 < 1$ implies that the sequence M_n is bounded. In particular, suppose $M_{2n-1} > L^\alpha$ where $2n \ge N$, then

$$M_{2n+1} \le L M_{2n-1}^{1/\alpha^2} < M_{2n-1}^{1/\alpha + 1/\alpha^2} = M_{2n-1}.$$

Therefore $M_{2(n+k)+1} \le M_{2n-1}$ for all $k \in \mathbf{N}$. Similarly if $M_{2n} > L^\alpha$ for some $2n \ge N$, then $M_{2(n+k)} \le M_{2n}$ for all $k \in \mathbf{N}$. Having consider both even and odd terms, we conclude in general that M_k is bounded. Consequently there exists M large enough such that $M_n \le M$ for every $n \in \mathbf{N}$. Thus

$$|e_{n+1}| \le M |e_n|^\alpha \qquad \text{for every} \qquad n \in \mathbf{N}$$

and so the secant method converges with order at least α.

The following references were consulted when preparing the above proof:

[1] Burden, Fairs and Burden, *Numerical Analysis, Tenth Edition*, hint given in for Problem 14 in Section 2.4.

[2] Dahlquist and Björck, *Numerical Methods in Scientific Computer, Volume I*, proof of Theorem 6.2.1.

3. Consider the fixed point iteration for solving $f(x)=0$ given by $x_{n+1}=h(x_n)$ where

$$h(x)=x-\frac{f(x)f'(x)}{[f'(x)]^2-f(x)f''(x)}.$$

(i) Show that $f(x)=0$ implies $h(x)=x$. Conversely show that if $h(x)=x$ then either $f(x)=0$ or $f'(x)=0$.

Suppose $f(x)=0$. Assuming $f'(x)\neq 0$ we have $h(x)=x-0/f'(x)^2=x$. More generally, if $f^{(k)}(x)=0$ for $k=1,\ldots,m$ and $f^{(m+1)}(x)\neq 0$ we claim $h(t)\to x$ as $t\to x$. In this case Taylor's theorem implies

$$f(t)=\int_x^t \frac{f^{(m+1)}(s)}{m!}(t-s)^m ds,$$

$$f'(t)=\int_x^t \frac{f^{(m+1)}(s)}{(m-1)!}(t-s)^{m-1}ds \qquad \text{and} \qquad f''(t)=\int_x^t \frac{f^{(m+1)}(s)}{(m-2)!}(t-s)^{m-2}ds.$$

Choose $\varepsilon>0$ but small enough such that $(1+m)^2(1-\varepsilon)^2-(1+\varepsilon)^2>0$. Since $f^{(m+1)}(x)\neq 0$, then there is $\delta>0$ such that

$$(1-\varepsilon)|f^{(m+1)}(x)|\le |f^{(m+1)}(t)|\le (1+\varepsilon)|f^{(m+1)}(x)| \qquad \text{for} \qquad |t-x|<\delta.$$

It follows that

$$\frac{(1-\varepsilon)|f^{(m+1)}(x)|}{(m+1)!}|t-x|^{m+1}\le |f(t)|\le \frac{(1+\varepsilon)|f^{(m+1)}(x)|}{(m+1)!}|t-x|^{m+1}$$

$$\frac{(1-\varepsilon)|f^{(m+1)}(x)|}{m!}|t-x|^{m}\le |f'(t)|\le \frac{(1+\varepsilon)|f^{(m+1)}(x)|}{m!}|t-x|^{m},$$

and

$$\frac{(1-\varepsilon)|f^{(m+1)}(x)|}{(m-1)!}|t-x|^{m-1}\le |f''(t)|\le \frac{(1+\varepsilon)|f^{(m+1)}(x)|}{(m-1)!}|t-x|^{m-1}.$$

Therefore, $h(t)=t-\omega(t)$ where

$$|\omega(t)|\le \frac{(m+1)(1+\varepsilon)(1+\varepsilon)}{(1+m)^2(1-\varepsilon)^2-(1+\varepsilon)(1+\varepsilon)}|t-x|\to 0 \qquad \text{as} \qquad t\to x.$$

Consequently $h(t)\to x$ as $t\to x$.

Conversely, suppose $h(x)=x$ or that $h(t)\to x$ as $t\to x$. Then

$$\frac{f(x)f'(x)}{[f'(x)]^2-f(x)f''(x)}=0 \qquad \text{or} \qquad \lim_{t\to x}\frac{f(t)f'(t)}{[f'(t)]^2-f(t)f''(t)}=0.$$

It follows that either $f(x)=0$ or $f'(x)=0$.

(ii) Compute $h'(x)$ and show that

$$h'(x) = \begin{cases} 0 & \text{when } f(x) = 0 \text{ and } f'(x) \neq 0 \\ 2 & \text{when } f(x) \neq 0,\ f'(x) = 0 \text{ and } f''(x) \neq 0. \end{cases}$$

Conclude that the fixed points of h for which $f(x) = 0$ are stable, but the fixed points for which $f'(x) = 0$ are not.

Computing with Maple and the commands

```
restart;
kernelopts(printbytes=false):
printf("#p3ii.mpl\n");
h:=x->x-f(x)*D(f)(x)/(D(f)(x)^2-f(x)*(D@@2)(f)(x));
dh:=simplify(D(h)(x));
```

yields the output

```
#p3ii.mpl
                                          f(x) D(f)(x)
                   h := x -> x - ----------------------------
                                             2           (2)
                                 D(f)(x)  - f(x) (D   )(f)(x)

dh := - f(x)

              2    (2)                         (2)         2                        (3)
    (D(f)(x)   (D     )(f)(x) - 2 f(x) (D     )(f)(x)   + f(x) D(f)(x) (D     )(f)(x))

         /          2           (2)         2
        /  (D(f)(x)   - f(x) (D     )(f)(x))
       /
```

or in other words that

$$h'(x) = \frac{f(x)\big(2f(x)f''(x)^2 - f'(x)^2 f''(x) - f(x)f'(x)f'''(x)\big)}{\big(f'(x)^2 - f(x)f''(x)\big)^2}.$$

If $f(x) = 0$ and $f'(x) \neq 0$ we have

$$h'(x) = \frac{0 \cdot \big(0 \cdot f''(x)^2 - f'(x)^2 f''(x) - 0 \cdot f'(x)f'''(x)\big)}{\big(f'(x)^2 - 0 \cdot f''(x)\big)^2} = 0.$$

On the other hand, if $f(x) \neq 0$, $f'(x) = 0$ and $f''(x) \neq 0$ then

$$h'(x) = \frac{f(x)\big(2f(x)f''(x)^2 - 0^2 f''(x) - f(x) \cdot 0 \cdot f'''(x)\big)}{\big(0^2 - f(x)f''(x)\big)^2} = \frac{2f(x)f(x)f''(x)^2}{\big(f(x)f''(x)\big)^2} = 2.$$

For an iteration of the form $x_{n+1} = h(x_n)$ to converge to a fixed point x_∞, it is a necessary condition that $h'(x_\infty) < 1$. Since $f(x) \neq 0$ implies $h'(x_\infty) = 2$, iterations will not converge to those fixed points where $f(x) \neq 0$.

(iii) Use this fixed point iteration with $x_0 = 1$ to solve $f(x) = 0$ where $f(x) = x^2 - 2$. Compare the performance of this method with your results for Newton's method.

The code

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

double f(double x){
    return x*x-2;
}
double df(double x){
    return 2*x;
}
double ddf(double x){
    return 2;
}
double h(double x){
    double y=f(x),dy=df(x),ddy=ddf(x);
    return x-y*dy/(dy*dy-y*ddy);
}
unsigned sigdig(double ps,double p){
    double relerror=fabs((p-ps)/p);
    return (int)ceil(-log(relerror/5)/log(10));
}
int main(){
    printf("#p3iii.c\n");
    double x=1,xinf=sqrt(2.0);
    printf("#%2s %22s %22s %12s\n","n","xn","en","sigdig");
    for(int i=0;;i++){
        printf("%3d %22.14e %22.14e %12u\n",i,x,x-xinf,sigdig(x,xinf));
        if(i>=8) break;
        x=h(x);
    }
    return 0;
}
```

produced the output

```
#p3iii.c
# n                     xn                     en      sigdig
  0   1.00000000000000e+00  -4.14213562373095e-01           2
  1   1.33333333333333e+00  -8.08802290397619e-02           2
  2   1.41176470588235e+00  -2.44885649074233e-03           4
  3   1.41421143847487e+00  -2.12389822507042e-06           7
  4   1.41421356237150e+00  -1.59494639717650e-12          13
  5   1.41421356237309e+00  -2.22044604925031e-16          17
  6   1.41421356237310e+00   0.00000000000000e+00  2147483648
  7   1.41421356237309e+00  -2.22044604925031e-16          17
  8   1.41421356237310e+00   0.00000000000000e+00  2147483648
```

The output is almost exactly the same as Newton's method.

(iv) Suppose $f(x) = (x-\xi)^m q(x)$ where $\lim_{x\to\xi} q(x) \neq 0$. Let $g(x) = x - f(x)/f'(x)$ as in Newton's method and show that

$$\lim_{x\to\xi} g'(x) = \frac{m-1}{m} \qquad \text{and} \qquad \lim_{x\to\xi} h'(x) = 0.$$

Conclude that even when $f'(x) = 0$ this method, unlike Newton's method, has an accelerating rate of convergence as x_n approaches the solution $x = \xi$ to $f(x) = 0$.

Differentiating yields

$$\begin{aligned}
f'(x) &= m(x-\xi)^{m-1}q(x) + (x-\xi)^m q'(x)\\
f''(x) &= m(m-1)(x-\xi)^{m-2}q(x) + 2m(x-\xi)^{m-1}q'(x) + (x-\xi)^m q''(x)
\end{aligned}$$

Therefore

$$\begin{aligned}
g'(x) &= 1 - \frac{f'(x)^2 - f(x)f''(x)}{f'(x)^2} = \frac{f(x)f''(x)}{f'(x)^2}\\
&= \frac{(x-\xi)^m q(x)\big(m(m-1)(x-\xi)^{m-2}q(x) + 2m(x-\xi)^{m-1}q'(x) + (x-\xi)^m q''(x)\big)}{\big(m(x-\xi)^{m-1}q(x) + (x-\xi)^m q'(x)\big)^2}\\
&= \frac{q(x)\big(m(m-1)q(x) + 2m(x-\xi)q'(x) + (x-\xi)^2 q''(x)\big)}{\big(mq(x) + (x-\xi)q'(x)\big)^2}\\
&\to \frac{q(\xi)m(m-1)q(\xi)}{\big(mq(\xi)\big)^2} = \frac{m-1}{m} \qquad \text{as} \qquad x \to \xi.
\end{aligned}$$

For the calculation of $h'(x)$ we resort to the Maple script

```
restart;
kernelopts(printbytes=false):
printf("#p3iv.mpl\n");
f:=x->(x-xi)^m*q(x):
h:=x->x-f(x)*D(f)(x)/(D(f)(x)^2-f(x)*(D@@2)(f)(x)):
dh:=D(h)(x);
printf("The limit as x->xi is\n");
limit(dh,x=xi);
```

with output

```
#p3iv.mpl
                      2                        m
                    %2                 (x - xi)  q(x) %1              m
dh := 1 - ----------------------- - ----------------------- + (x - xi)  q(x) %2
            2           m             2           m
          %2  - (x - xi)  q(x) %1   %2  - (x - xi)  q(x) %1

    /                       /        m  3                  m  2
    |                m      |(x - xi)  m  q(x)   3 (x - xi)  m  q(x)
```

```
      |%2 %1 - (x - xi)  q(x) |----------------- - -------------------
      |                       |            3                    3
      \                       \    (x - xi)            (x - xi)

                  m  2                        m                   m
         3 (x - xi)  m  D(q)(x)   2 (x - xi)  m q(x)   3 (x - xi)  m D(q)(x)
       + --------------------- + ----------------- - ---------------------
                        2                     3                      2
               (x - xi)              (x - xi)              (x - xi)

                  m     (2)                                 \\
         3 (x - xi)  m (D   )(q)(x)           m   (3)       ||   /
       + ------------------------- + (x - xi)  (D   )(q)(x)||  /
                  x - xi                                    || /
                                                            //

         2           m         2
      (%2  - (x - xi)  q(x) %1)

                m  2              m                   m
        (x - xi)  m  q(x)   (x - xi)  m q(x)   2 (x - xi)  m D(q)(x)
  %1 := ---------------- - --------------- + ---------------------
                   2                  2               x - xi
            (x - xi)          (x - xi)

                 m   (2)
       + (x - xi)  (D   )(q)(x)

               m
        (x - xi)  m q(x)           m
  %2 := ---------------- + (x - xi)  D(q)(x)
             x - xi

  The limit as x->xi is
                                     0
```

Therefore, as shown by the above Maple calculation $h'(x) \to 0$ as $x \to \xi$. Since $f(\xi) = 0$ then $h(\xi) = \xi$. It follows that

$$|x_{n+1} - \xi| = |h(x_n) - h(\xi)| = \Big| \int_{x_n}^{\xi} h'(t)dt \Big| \le \rho_n |x_n - \xi|$$

where

$$\rho_n = \max\{\, |h'(t)| : t \text{ is between } x_n \text{ and } \xi \,\}.$$

We interpret ρ_n as the ratio by which the error contracts between the n to the $n+1$ iteration. Since $h'(t) \to 0$ as $t \to \xi$ then $\rho_n \to 0$ as $n \to \infty$. It follows that this sequence has an accelerating rate of convergence as $n \to \infty$. On the other hand, for Newton's method, since $g'(x) \to (m-1)/m$ as $x \to \xi$, the rate of convergence levels off to a linear ratio of $(m-1)/m$ as $n \to \infty$.

4. The function

$$f(x) = 2\cos(5x) + 2\cos(4x) + 6\cos(3x) + 4\cos(2x) + 10\cos(x) + 3$$

has two roots on the interval $[0, 3]$; one root is near 1 and the other near 2.

(i) Use Newton's method $x_{n+1} = g(x_n)$ with $x_0 = 1$ and also with $x_0 = 2$ to approximate these two roots. Use the fact that the exact roots are $\pi/3$ and $2\pi/3$ to compute the error e_n at each iteration for $n = 0, 1, \ldots, 18$.

The code

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

double f(double x){
    return 2*cos(5*x)+2*cos(4*x)+6*cos(3*x)+4*cos(2*x)+10*cos(x)+3;
}
double df(double x){
    return -10*sin(5*x)-8*sin(4*x)-18*sin(3*x)-8*sin(2*x)-10*sin(x);
}
double g(double x){
    return x-f(x)/df(x);
}
int main(){
    printf("#p4i.c\n");
    double x=1,xinf=M_PI/3;
    double y=2,yinf=2*M_PI/3;
    printf("#%2s %18s %18s %18s %18s\n",
        "n","xn","en","yn","en");
    for(int i=0;;i++){
        printf("%3d %18.10e %18.10e %18.10e %18.10e\n",
            i,x,x-xinf,y,y-yinf);
        if(i>=18) break;
        x=g(x); y=g(y);
    }
    return 0;
}
```

produces the output

```
#p4i.c
# n               xn                en                yn                en
  0   1.0000000000e+00  -4.7197551197e-02   2.0000000000e+00  -9.4395102393e-02
  1   1.0226323341e+00  -2.4565217065e-02   2.0327367304e+00  -6.1658372020e-02
  2   1.0346334653e+00  -1.2564085860e-02   2.0537663411e+00  -4.0628761277e-02
  3   1.0408389641e+00  -6.3585870488e-03   2.0674991299e+00  -2.6895972473e-02
  4   1.0439982526e+00  -3.1992985566e-03   2.0765429375e+00  -1.7852164921e-02
  5   1.0455927850e+00  -1.6047661895e-03   2.0825268979e+00  -1.1868204458e-02
  6   1.0463938739e+00  -8.0367725365e-04   2.0864972711e+00  -7.8978312884e-03
  7   1.0467953871e+00  -4.0216406374e-04   2.0891361042e+00  -5.2589981827e-03
  8   1.0469963876e+00  -2.0116363032e-04   2.0908918299e+00  -3.5032725260e-03
  9   1.0470969490e+00  -1.0060224474e-04   2.0920607875e+00  -2.3343148837e-03
 10   1.0471472450e+00  -5.0306233608e-05   2.0928394225e+00  -1.5556799278e-03
 11   1.0471723968e+00  -2.5154395855e-05   2.0933582170e+00  -1.0368853819e-03
 12   1.0471849737e+00  -1.2577516050e-05   2.0937039494e+00  -6.9115294687e-04
 13   1.0471912624e+00  -6.2888362424e-06   2.0939343799e+00  -4.6072245719e-04
 14   1.0471944068e+00  -3.1444357924e-06   2.0940879744e+00  -3.0712796824e-04
 15   1.0471959790e+00  -1.5722235340e-06   2.0941903593e+00  -2.0474309283e-04
 16   1.0471967651e+00  -7.8610765075e-07   2.0942586108e+00  -1.3649161752e-04
 17   1.0471971581e+00  -3.9308790645e-07   2.0943041099e+00  -9.0992468730e-05
 18   1.0471973546e+00  -1.9660660611e-07   2.0943344418e+00  -6.0660575847e-05
```

Note that the second and third columns correspond to the values of x_n and e_n for Newton's iteration starting with $x_0 = 1$ while the fourth and fifth columns correspond Newton's iteration starting with $x_0 = 2$.

(ii) Use the method $x_{n+1} = h(x_n)$ with $x_0 = 1$ and again also with $x_0 = 2$ to approximate these two roots. Again use the fact that the exact roots are $\pi/3$ and $2\pi/3$ to compute the error e_n at each iteration for $n = 0, 1, \ldots, 18$.

The code

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

double f(double x){
    return 2*cos(5*x)+2*cos(4*x)+6*cos(3*x)+4*cos(2*x)+10*cos(x)+3;
}
double df(double x){
    return -10*sin(5*x)-8*sin(4*x)-18*sin(3*x)-8*sin(2*x)-10*sin(x);
}
double ddf(double x){
    return -50*cos(5*x)-32*cos(4*x)-54*cos(3*x)-16*cos(2*x)-10*cos(x);
}
double h(double x){
    double y=f(x),dy=df(x),ddy=ddf(x);
    return x-y*dy/(dy*dy-y*ddy);
}
int main(){
    printf("#p4ii.c\n");
    double x=1,xinf=M_PI/3;
    double y=2,yinf=2*M_PI/3;
    printf("#%2s %18s %18s %18s %18s\n",
        "n","xn","en","yn","en");
    for(int i=0;;i++){
        printf("%3d %18.10e %18.10e %18.10e %18.10e\n",
            i,x,x-xinf,y,y-yinf);
        if(i>=18) break;
        x=h(x); y=h(y);
    }
    return 0;
}
```

produces the output

```
#p4ii.c
# n                xn                 en                 yn                 en
  0  1.0000000000e+00  -4.7197551197e-02  2.0000000000e+00  -9.4395102393e-02
  1  1.0489744220e+00   1.7768708227e-03  2.0896423074e+00  -4.7527949859e-03
  2  1.0472007669e+00   3.2157141134e-06  2.0943883125e+00  -6.7898500431e-06
  3  1.0471975512e+00   9.8343555521e-13  2.0943999121e+00   4.8097554064e-06
  4  1.0471975512e+00   1.9668711104e-12  2.0943791702e+00  -1.5932145905e-05
  5  1.0471975512e+00   3.9335201762e-12  2.0943947851e+00  -3.1729864558e-07
  6  1.0471975512e+00   7.8670403525e-12  2.0943946265e+00  -4.7593073838e-07
  7  1.0471975512e+00   1.5733858660e-11  2.0943943892e+00  -7.1317747530e-07
  8  1.0471975512e+00   3.1467717321e-11  2.0943940281e+00  -1.0742473977e-06
  9  1.0471975512e+00   3.1467717321e-11  2.0943934667e+00  -1.6357170938e-06
 10  1.0471975512e+00   3.1467717321e-11  2.0943925953e+00  -2.5071035799e-06
 11  1.0471975512e+00   3.1467717321e-11  2.0943897902e+00  -5.3122035357e-06
 12  1.0471975512e+00   3.1467717321e-11  2.0943921943e+00  -2.9081381792e-06
 13  1.0471975512e+00   3.1467717321e-11  2.0943887686e+00  -6.3337454868e-06
 14  1.0471975512e+00   3.1467717321e-11  2.0943919936e+00  -3.1087908559e-06
 15  1.0471975512e+00   3.1467717321e-11  2.0943867588e+00  -8.3435748301e-06
 16  1.0471975512e+00   3.1467717321e-11  2.0943933693e+00  -1.7331209072e-06
 17  1.0471975512e+00   3.1467717321e-11  2.0943927346e+00  -2.3677631007e-06
 18  1.0471975512e+00   3.1467717321e-11  2.0943927346e+00  -2.3677631007e-06
```

Again the second and third columns correspond to the values of x_n and e_n for the h-iteration starting with $x_0 = 1$ while the fourth and fifth columns correspond the h-iteration starting with $x_0 = 2$.

(iii) Comment on the rate of convergence and the effects of rounding error in the above two computations.

We node that the rate of convergence of Newton's method is very slow and looks linear. This is further demonstrated by the convergence ratios $M_n = |e_{n+1}|/|e_n|$ for each of the runs. The code

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

double f(double x){
    return 2*cos(5*x)+2*cos(4*x)+6*cos(3*x)+4*cos(2*x)+10*cos(x)+3;
}
double df(double x){
    return -10*sin(5*x)-8*sin(4*x)-18*sin(3*x)-8*sin(2*x)-10*sin(x);
}
double g(double x){
    return x-f(x)/df(x);
}
int main(){
    printf("#p4iii.c\n");
    double x=1,xinf=M_PI/3;
    double y=2,yinf=2*M_PI/3;
    printf("#%2s %18s %18s %18s %18s\n",
        "n","xn","Mn","yn","Mn");
    for(int i=0;i<=18;i++){
        {
            double eold=x-xinf;
            double xnew=g(x);
            double enew=xnew-xinf;
            double M=fabs(enew/eold);
            printf("%3d %18.10e %18.10e",i,x,M);
            x=xnew;
        }
        {
            double eold=y-yinf;
            double ynew=g(y);
            double enew=ynew-yinf;
            double M=fabs(enew/eold);
            printf(" %18.10e %18.10e\n",y,M);
            y=ynew;
        }
    }
    return 0;
}
```

produces the output

```
#p4iii.c
# n               xn                 Mn                 yn                 Mn
   0   1.0000000000e+00   5.2047651716e-01   2.0000000000e+00   6.5319460922e-01
   1   1.0226323341e+00   5.1145836921e-01   2.0327367304e+00   6.5893340914e-01
   2   1.0346334653e+00   5.0609229513e-01   2.0537663411e+00   6.6199341617e-01
   3   1.0408389641e+00   5.0314614427e-01   2.0674991299e+00   6.6374863150e-01
   4   1.0439982526e+00   5.0159938535e-01   2.0765429375e+00   6.6480477357e-01
   5   1.0455927850e+00   5.0080644700e-01   2.0825268979e+00   6.6546134389e-01
   6   1.0463938739e+00   5.0040493484e-01   2.0864972711e+00   6.6587876984e-01
   7   1.0467953871e+00   5.0020289840e-01   2.0891361042e+00   6.6614826708e-01
   8   1.0469963876e+00   5.0010155705e-01   2.0908918299e+00   6.6632409164e-01
   9   1.0470969490e+00   5.0005080639e-01   2.0920607875e+00   6.6643962159e-01
  10   1.0471472450e+00   5.0002542530e-01   2.0928394225e+00   6.6651588373e-01
  11   1.0471723968e+00   5.0001264678e-01   2.0933582170e+00   6.6656639101e-01
  12   1.0471849737e+00   5.0000621885e-01   2.0937039494e+00   6.6659985936e-01
  13   1.0471912624e+00   5.0000280993e-01   2.0939343799e+00   6.6662252610e-01
  14   1.0471944068e+00   5.0000179295e-01   2.0940879744e+00   6.6663773410e-01
  15   1.0471959790e+00   4.9999738188e-01   2.0941903593e+00   6.6664821573e-01
  16   1.0471967651e+00   5.0004335420e-01   2.0942586108e+00   6.6665243174e-01
  17   1.0471971581e+00   5.0015938644e-01   2.0943041099e+00   6.6665490775e-01
  18   1.0471973546e+00   4.9975955952e-01   2.0943344418e+00   6.6661111543e-01
```

which shows that $M_n \to 1/2$ as $n \to \infty$ when $x_0 = 1$ and that $M_n \to 2/3$ when $x_0 = 2$. Therefore the convergence is linear and solving for m in the relationship

$$\frac{|e_{n+1}|}{|e_n|} \to \lim_{x\to\xi} g'(x) = \frac{m-1}{m} \qquad \text{as} \qquad n \to \infty$$

we conclude that $\pi/3$ is a root of multiplicity 2 and $2\pi/3$ is a root of multiplicity 3.

Comparing Newton's iteration to the h-iteration, we see that the h-iteration converges much faster. It looks quadratic between the first, second and third iterations with starting with $x_0 = 1$. When starting with $x_0 = 2$ the number of significant digits doubles between the first and second iterations, but after that rounding error plays a significant role. In fact, rounding error plays a significant role in the h-iterations for either starting value with the final accuracy being only 12 digits out of 15 in the case of $x_0 = 1$ and a limited 7 digits out of 15 when $x_0 = 2$. Loss of precision likely occurs in the denominator $[f'(x)]^2 - f(x)f''(x)$ of the method with cancelation of two nearly equal quantities; however, this needs to be verified to be certain. Such verification is beyond the scope of the present report.

5. [Extra Credit and Math/CS 666] Define

$$x_{n+1} = x_n - \frac{F_n F_n'}{[F_n']^2 - F_n F_n''}$$

where

$$\begin{aligned} F_n &= f\Big(\frac{x_n + 2x_{n-1} + x_{n-2}}{4}\Big), \\ F_n' &= \frac{2}{x_n - x_{n-2}}\Big\{f\Big(\frac{x_n + x_{n-1}}{2}\Big) - f\Big(\frac{x_{n-1} + x_{n-2}}{2}\Big)\Big\}, \\ F_n'' &= \frac{2}{x_n - x_{n-2}}\Big(\frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}} - \frac{f(x_{n-1}) - f(x_{n-2})}{x_{n-1} - x_{n-2}}\Big) \end{aligned}$$

to create a secant-method-like approximation of the method given by h that doesn't involve f' and f''. Study this method both numerically and analytically. Test this method for the functions $f(x) = x^2 - 2$ and

$$f(x) = 2\cos(5x) + 2\cos(4x) + 6\cos(3x) + 4\cos(2x) + 10\cos(x) + 3.$$

How does this method compare to the usual secant method? Can you think of an improvement that works better and still doesn't involve f' or f''?

The code

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

double f2(double x){
    return 2*cos(5*x)+2*cos(4*x)+6*cos(3*x)+4*cos(2*x)+10*cos(x)+3;
}
double f1(double x){
    return x*x-2;
}
double H(double x2,double x1,double x0,double (*f)(double)){
    double y=f((x2+2*x1+x0)/4);
    double dy=2/(x2-x0)*(f((x2+x1)/2)-f((x1+x0)/2));
    double ddy=2/(x2-x0)*((f(x2)-f(x1))/(x2-x1)-(f(x1)-f(x0))/(x1-x0));
    return x2-y*dy/(dy*dy-y*ddy);
}
double G(double x1,double x0,double (*f)(double)){
    double y=f(x1);
    double dy=(f(x1)-f(x0))/(x1-x0);
    return x1-y/dy;
}

void iterate(double x2,double x1,double x0,double xinf,double (*f)(double)){
    printf("#%40s %37s\n",
        "the extra-credit method","the usual secant method");
    printf("#%2s %18s %18s %18s %18s\n",
        "n","xn","en","yn","en");
    double y0=x0,y1=x1;
    printf("%3d %18.10e %18.10e %18.10e %18.10e\n",
        0,x0,x0-xinf,y0,y0-xinf);
    printf("%3d %18.10e %18.10e %18.10e %18.10e\n",
        1,x1,x1-xinf,y1,y1-xinf);
    double y=G(y1,y0,f);
    y0=y1; y1=y;
    for(int i=2;;i++){
        printf("%3d %18.10e %18.10e %18.10e %18.10e\n",
            i,x2,x2-xinf,y1,y1-xinf);
        if(i>=24) break;
        double y=G(y1,y0,f);
        y0=y1; y1=y;
        double x=H(x2,x1,x0,f);
        x0=x1; x1=x2; x2=x;
    }
}

int main(){
    printf("#p5i.c\n");
    printf("\n" "#f(x)=x^2-2,\n");
    iterate(1.0,0.9,1.1,sqrt(2),f1);
    printf("\n\n"
        "#f(x)=2*cos(5*x)+2*cos(4*x)+6*cos(3*x)+4*cos(2*x)+10*cos(x)+3,\n");
    iterate(1.0,0.9,1.1,M_PI/3,f2);
    printf("\n\n"
        "#f(x)=2*cos(5*x)+2*cos(4*x)+6*cos(3*x)+4*cos(2*x)+10*cos(x)+3,\n");
    iterate(2.0,1.9,2.1,2*M_PI/3,f2);
    return 0;
}
```

produces the output

```
#p5i.c

#f(x)=x^2-2,
#                  the extra-credit method               the usual secant method
# n                 xn                 en                 yn                 en
  0   1.1000000000e+00  -3.1421356237e-01   1.1000000000e+00  -3.1421356237e-01
  1   9.0000000000e-01  -5.1421356237e-01   9.0000000000e-01  -5.1421356237e-01
  2   1.0000000000e+00  -4.1421356237e-01   1.4950000000e+00   8.0786437627e-02
  3   1.3467538657e+00  -6.7459696667e-02   1.3968684760e+00  -1.7345086381e-02
  4   1.6430782437e+00   2.2886468133e-01   1.4137290148e+00  -4.8454753414e-04
  5   1.7207468265e+00   3.0653326408e-01   1.4142165527e+00   2.9902961425e-06
  6   1.5370579207e+00   1.2284435833e-01   1.4142135619e+00  -5.1236526133e-10
  7   1.2785086132e+00  -1.3570494915e-01   1.4142135624e+00  -6.6613381478e-16
  8   1.1708177108e+00  -2.4339585161e-01   1.4142135624e+00  -2.2204460493e-16
  9   1.2651698438e+00  -1.4904371862e-01   1.4142135624e+00   0.0000000000e+00
 10   1.4429860048e+00   2.8772442432e-02   1.4142135624e+00  -2.2204460493e-16
 11   1.5648047758e+00   1.5059121342e-01   1.4142135624e+00  -2.2204460493e-16
 12   1.5499553217e+00   1.3574175931e-01               -nan               -nan
 13   1.4291172269e+00   1.4903664546e-02               -nan               -nan
 14   1.3159665738e+00  -9.8246988605e-02               -nan               -nan
 15   1.2990421422e+00  -1.1517142018e-01               -nan               -nan
 16   1.3711817755e+00  -4.3031786838e-02               -nan               -nan
 17   1.4608284676e+00   4.6614905270e-02               -nan               -nan
 18   1.4989405696e+00   8.4727007190e-02               -nan               -nan
 19   1.4648165200e+00   5.0602957658e-02               -nan               -nan
 20   1.3966494751e+00  -1.7564087292e-02               -nan               -nan
 21   1.3539492235e+00  -6.0264338826e-02               -nan               -nan
 22   1.3651019310e+00  -4.9111631325e-02               -nan               -nan
 23   1.4111025755e+00  -3.1109868331e-03               -nan               -nan
 24   1.4509084573e+00   3.6694894885e-02               -nan               -nan


#f(x)=2*cos(5*x)+2*cos(4*x)+6*cos(3*x)+4*cos(2*x)+10*cos(x)+3,
#                 the extra-credit method              the usual secant method
# n                 xn                 en                 yn                 en
  0   1.1000000000e+00   5.2802448803e-02   1.1000000000e+00   5.2802448803e-02
  1   9.0000000000e-01  -1.4719755120e-01   9.0000000000e-01  -1.4719755120e-01
  2   1.0000000000e+00  -4.7197551197e-02   1.1196553035e+00   7.2457752263e-02
  3   1.0658955138e+00   1.8697962592e-02   1.1618045574e+00   1.1460700620e-01
  4   1.1199000538e+00   7.2702502611e-02   1.0860467711e+00   3.8849219907e-02
  5   1.1009463529e+00   5.3748801676e-02   1.0739484024e+00   2.6750851211e-02
  6   1.0478137039e+00   6.1615271508e-04   1.0624733436e+00   1.5275792428e-02
  7   1.0032660503e+00  -4.3931500893e-02   1.0567157632e+00   9.5182120399e-03
  8   9.9375220868e-01  -5.3445342520e-02   1.0529893596e+00   5.7918084065e-03
  9   1.0288784885e+00  -1.8319062674e-02   1.0507714277e+00   3.5738764925e-03
 10   1.0716527356e+00   2.4455184448e-02   1.0493976366e+00   2.2000853617e-03
 11   1.0871856735e+00   3.9988122289e-02   1.0485555420e+00   1.3579907807e-03
 12   1.0689428006e+00   2.1745249438e-02   1.0480358121e+00   8.3826085446e-04
 13   1.0379818679e+00  -9.2156833295e-03   1.0477153215e+00   5.1777029537e-04
 14   1.0191979248e+00  -2.7999626351e-02   1.0475174150e+00   3.1986375576e-04
 15   1.0249416116e+00  -2.2255939643e-02   1.0473951908e+00   1.9763963651e-04
 16   1.0469555816e+00  -2.4196962207e-04   1.0473196801e+00   1.2212893400e-04
 17   1.0651640555e+00   1.7966504340e-02   1.0472730240e+00   7.5472815601e-05
 18   1.0661212939e+00   1.8923742727e-02   1.0472441932e+00   4.6642022625e-05
 19   1.0524746937e+00   5.2771424647e-03   1.0472263765e+00   2.8825323730e-05
 20   1.0373074795e+00  -9.8900717319e-03   1.0472153658e+00   1.7814632219e-05
 21   1.0322684910e+00  -1.4929060162e-02   1.0472085611e+00   1.1009896627e-05
 22   1.0395665294e+00  -7.6310218432e-03   1.0472043556e+00   6.8044334973e-06
 23   1.0514725471e+00   4.2749959460e-03   1.0472017565e+00   4.2053526790e-06
 24   1.0579089715e+00   1.0711420291e-02   1.0472001502e+00   2.5990370549e-06


#f(x)=2*cos(5*x)+2*cos(4*x)+6*cos(3*x)+4*cos(2*x)+10*cos(x)+3,
#                 the extra-credit method              the usual secant method
# n               xn               en               yn               en
  0   2.1000000000e+00   5.6048976068e-03   2.1000000000e+00   5.6048976068e-03
  1   1.9000000000e+00  -1.9439510239e-01   1.9000000000e+00  -1.9439510239e-01
  2   2.0000000000e+00  -9.4395102393e-02   2.0999938553e+00   5.5987528838e-03
  3   2.0824700223e+00  -1.1925080106e-02   2.0999877310e+00   5.5926285676e-03
  4   2.1566853422e+00   6.2290239780e-02   2.0981284968e+00   3.7333944315e-03
  5   2.1580851602e+00   6.3690057761e-02   2.0973429599e+00   2.9478575233e-03
  6   2.1301393942e+00   3.5744291797e-02   2.0965823314e+00   2.1872289569e-03
  7   2.0793280717e+00  -1.5067030685e-02   2.0960576651e+00   1.6625627019e-03
  8   2.0627733620e+00  -3.1621740380e-02   2.0956471123e+00   1.2520099139e-03
  9   2.0634752538e+00  -3.0919848553e-02   2.0953412813e+00   9.4617891509e-04
 10   2.0874011682e+00  -6.9939341849e-03   2.0951091498e+00   7.1404745095e-04
 11   2.1075296815e+00   1.3134579149e-02   2.0949342409e+00   5.3913855676e-04
 12   2.1097199705e+00   1.5324868147e-02   2.0948020862e+00   4.0698380650e-04
 13   2.1053312362e+00   1.0936133831e-02   2.0947023452e+00   3.0724283397e-04
 14   2.0926817983e+00  -1.7133040620e-03   2.0946270388e+00   2.3193641692e-04
 15   2.0868510333e+00  -7.5440690518e-03   2.0945701907e+00   1.7508835391e-04
 16   2.0868510334e+00  -7.5440690228e-03   2.0945272742e+00   1.3217179595e-04
 17   2.0919434432e+00  -2.4516591678e-03   2.0944948784e+00   9.9775979922e-05
 18   2.0974474291e+00   3.0523267204e-03   2.0944704242e+00   7.5321841337e-05
 19   2.0982622094e+00   3.8671069739e-03   2.0944519678e+00   5.6865400062e-05
 20   2.0974873227e+00   3.0922203305e-03   2.0944380226e+00   4.2920173231e-05
 21   2.0942625557e+00  -1.3254668057e-04   2.0944274954e+00   3.2393005667e-05
 22   2.0925192587e+00  -1.8758437104e-03   2.0944195788e+00   2.4476456089e-05
 23   2.0925113284e+00  -1.8837740019e-03   2.0944135338e+00   1.8431410462e-05
 24   2.0935826223e+00  -8.1248004605e-04   2.0944089692e+00   1.3866784173e-05
```

It would appear this method is worse than the usual secant method. This is somewhat surprising, because it appears to be a straight-forward approximation of the h-iterations of the previous question. Perhaps there is so much rounding error that the method is unable to work.

This hypothesis will now be tested using the Maple script

```
restart;
kernelopts(printbytes=false):
printf("#p5i2.mpl\n");
Digits:=1000:
f2:=x->2*cos(5*x)+2*cos(4*x)+6*cos(3*x)+4*cos(2*x)+10*cos(x)+3:
f1:=x->x*x-2:
H:=proc(x2,x1,x0,f)
    local y,dy,ddy;
    y:=f((x2+2*x1+x0)/4);
    dy:=2/(x2-x0)*(f((x2+x1)/2)-f((x1+x0)/2));
    ddy:=2/(x2-x0)*((f(x2)-f(x1))/(x2-x1)-(f(x1)-f(x0))/(x1-x0));
    x2-y*dy/(dy*dy-y*ddy);
end proc:
G:=proc(x1,x0,f)
    local y,dy;
    y:=f(x1);
    dy:=(f(x1)-f(x0))/(x1-x0);
    x1-y/dy;
end proc:

iterate:=proc(p2,p1,p0,xinf,f)
    local i,y0,y1,y,x2,x1,x0,x;
    y0:=p0; y1:=p1; x0:=p0; x1:=p1; x2:=p2;
    printf("#%40s %37s\n",
        "the extra-credit method","the usual secant method");
    printf("#%2s %18s %18s %18s %18s\n",
        "n","xn","en","yn","en");
    printf("%3d %18.10e %18.10e %18.10e %18.10e\n",
        0,x0,x0-xinf,y0,y0-xinf);
    printf("%3d %18.10e %18.10e %18.10e %18.10e\n",
        1,x1,x1-xinf,y1,y1-xinf);
    y:=G(y1,y0,f);
    y0:=y1; y1:=y;
    i:=2;
    while(true)
    do
        printf("%3d %18.10e %18.10e %18.10e %18.10e\n",
            i,x2,x2-xinf,y1,y1-xinf);
        if i>=24 then
            break;
        end;
        y:=G(y1,y0,f);
        y0:=y1; y1:=y;
        x:=H(x2,x1,x0,f);
        x0:=x1; x1:=x2; x2:=x;
        i:=i+1;
    end;
end proc:

printf("\n" "#f(x)=x^2-2,\n");
iterate(1.0,0.9,1.1,sqrt(2),f1):
printf("\n\n"
    "#f(x)=2*cos(5*x)+2*cos(4*x)+6*cos(3*x)+4*cos(2*x)+10*cos(x)+3,\n");
iterate(1.0,0.9,1.1,evalf(Pi/3),f2):
printf("\n\n"
    "#f(x)=2*cos(5*x)+2*cos(4*x)+6*cos(3*x)+4*cos(2*x)+10*cos(x)+3,\n");
iterate(2.0,1.9,2.1,evalf(2*Pi/3),f2):
```

which produces the output

```
#p5i2.mpl

#f(x)=x^2-2,
#                the extra-credit method                 the usual secant method
# n                   xn                 en                   yn                 en
  0   1.1000000000e+00  -3.1421356237e-01   1.1000000000e+00  -3.1421356237e-01
  1   9.0000000000e-01  -5.1421356237e-01   9.0000000000e-01  -5.1421356237e-01
  2   1.0000000000e+00  -4.1421356237e-01   1.4950000000e+00   8.0786437627e-02
  3   1.3467538657e+00  -6.7459696667e-02   1.3968684760e+00  -1.7345086381e-02
  4   1.6430782437e+00   2.2886468133e-01   1.4137290148e+00  -4.8454753414e-04
  5   1.7207468265e+00   3.0653326408e-01   1.4142165527e+00   2.9902961427e-06
  6   1.5370579207e+00   1.2284435833e-01   1.4142135619e+00  -5.1236510311e-10
  7   1.2785086132e+00  -1.3570494915e-01   1.4142135624e+00  -5.4168684728e-16
  8   1.1708177108e+00  -2.4339585161e-01   1.4142135624e+00   9.8125716225e-26
  9   1.2651698438e+00  -1.4904371862e-01   1.4142135624e+00  -1.8792568277e-41
 10   1.4429860048e+00   2.8772442432e-02   1.4142135624e+00  -6.5196455153e-67
 11   1.5648047758e+00   1.5059121342e-01   1.4142135624e+00   4.3317673776e-108
 12   1.5499553217e+00   1.3574175931e-01   1.4142135624e+00 -9.9849091070e-175
 13   1.4291172269e+00   1.4903664546e-02   1.4142135624e+00 -1.5291998567e-282
 14   1.3159665738e+00  -9.8246988605e-02   1.4142135624e+00   5.3983789937e-457
 15   1.2990421422e+00  -1.1517142018e-01   1.4142135624e+00 -2.9186540856e-739
 16   1.3711817755e+00  -4.3031786838e-02   1.4142135624e+00   0.0000000000e+00
 17   1.4608284676e+00   4.6614905270e-02   1.4142135624e+00   0.0000000000e+00
 18   1.4989405696e+00   8.4727007190e-02                NaN                NaN
 19   1.4648165200e+00   5.0602957658e-02                NaN                NaN
 20   1.3966494751e+00  -1.7564087292e-02                NaN                NaN
 21   1.3539492235e+00  -6.0264338826e-02                NaN                NaN
 22   1.3651019310e+00  -4.9111631325e-02                NaN                NaN
 23   1.4111025755e+00  -3.1109868331e-03                NaN                NaN
 24   1.4509084573e+00   3.6694894885e-02                NaN                NaN


#f(x)=2*cos(5*x)+2*cos(4*x)+6*cos(3*x)+4*cos(2*x)+10*cos(x)+3,
#                the extra-credit method                 the usual secant method
# n                   xn                 en                   yn                 en
  0   1.1000000000e+00   5.2802448803e-02   1.1000000000e+00   5.2802448803e-02
  1   9.0000000000e-01  -1.4719755120e-01   9.0000000000e-01  -1.4719755120e-01
  2   1.0000000000e+00  -4.7197551197e-02   1.1196553035e+00   7.2457752263e-02
  3   1.0658955138e+00   1.8697962592e-02   1.1618045574e+00   1.1460700620e-01
  4   1.1199000538e+00   7.2702502611e-02   1.0860467711e+00   3.8849219907e-02
  5   1.1009463529e+00   5.3748801676e-02   1.0739484024e+00   2.6750851211e-02
  6   1.0478137039e+00   6.1615271507e-04   1.0624733436e+00   1.5275792428e-02
  7   1.0032660503e+00  -4.3931500893e-02   1.0567157632e+00   9.5182120399e-03
  8   9.9375220868e-01  -5.3445342520e-02   1.0529893596e+00   5.7918084065e-03
  9   1.0288784885e+00  -1.8319062674e-02   1.0507714277e+00   3.5738764925e-03
 10   1.0716527356e+00   2.4455184448e-02   1.0493976366e+00   2.2000853617e-03
 11   1.0871856735e+00   3.9988122289e-02   1.0485555420e+00   1.3579907807e-03
 12   1.0689428006e+00   2.1745249438e-02   1.0480358121e+00   8.3826085444e-04
 13   1.0379818679e+00  -9.2156833294e-03   1.0477153215e+00   5.1777029539e-04
 14   1.0191979248e+00  -2.7999626351e-02   1.0475174150e+00   3.1986375573e-04
 15   1.0249416116e+00  -2.2255939643e-02   1.0473951908e+00   1.9763963646e-04
 16   1.0469555816e+00  -2.4196962210e-04   1.0473196801e+00   1.2212893398e-04
 17   1.0651640555e+00   1.7966504340e-02   1.0472730240e+00   7.5472815680e-05
 18   1.0661212939e+00   1.8923742727e-02   1.0472441932e+00   4.6642023276e-05
 19   1.0524746937e+00   5.2771424647e-03   1.0472263765e+00   2.8825323416e-05
 20   1.0373074795e+00  -9.8900717318e-03   1.0472153658e+00   1.7814631862e-05
 21   1.0322684910e+00  -1.4929060162e-02   1.0472085611e+00   1.1009896904e-05
```

```
  22   1.0395665294e+00  -7.6310218432e-03   1.0472043556e+00   6.8044325976e-06
  23   1.0514725471e+00   4.2749959459e-03   1.0472017565e+00   4.2053485499e-06
  24   1.0579089715e+00   1.0711420291e-02   1.0472001502e+00   2.5990398979e-06
```

```
#f(x)=2*cos(5*x)+2*cos(4*x)+6*cos(3*x)+4*cos(2*x)+10*cos(x)+3,
#                   the extra-credit method              the usual secant method
# n                 xn                 en                 yn                 en
   0   2.1000000000e+00   5.6048976068e-03   2.1000000000e+00   5.6048976068e-03
   1   1.9000000000e+00  -1.9439510239e-01   1.9000000000e+00  -1.9439510239e-01
   2   2.0000000000e+00  -9.4395102393e-02   2.0999938553e+00   5.5987528838e-03
   3   2.0824700223e+00  -1.1925080106e-02   2.0999877310e+00   5.5926285676e-03
   4   2.1566853422e+00   6.2290239780e-02   2.0981284969e+00   3.7333944904e-03
   5   2.1580851602e+00   6.3690057761e-02   2.0973429599e+00   2.9478575555e-03
   6   2.1301393942e+00   3.5744291797e-02   2.0965823314e+00   2.1872289814e-03
   7   2.0793280717e+00  -1.5067030687e-02   2.0960576651e+00   1.6625627294e-03
   8   2.0627733620e+00  -3.1621740381e-02   2.0956471123e+00   1.2520099219e-03
   9   2.0634752538e+00  -3.0919848553e-02   2.0953412813e+00   9.4617891297e-04
  10   2.0874011682e+00  -6.9939341844e-03   2.0951091499e+00   7.1404747249e-04
  11   2.1075296815e+00   1.3134579150e-02   2.0949342410e+00   5.3913856184e-04
  12   2.1097199705e+00   1.5324868147e-02   2.0948020862e+00   4.0698382904e-04
  13   2.1053312362e+00   1.0936133832e-02   2.0947023453e+00   3.0724294633e-04
  14   2.0926817983e+00  -1.7133040546e-03   2.0946270388e+00   2.3193644459e-04
  15   2.0868510333e+00  -7.5440690486e-03   2.0945701909e+00   1.7508851260e-04
  16   2.0868510334e+00  -7.5440690185e-03   2.0945272751e+00   1.3217269774e-04
  17   2.0917170307e+00  -2.6780717319e-03   2.0944948781e+00   9.9775664135e-05
  18   2.0970446265e+00   2.6495241367e-03   2.0944704216e+00   7.5319203891e-05
  19   2.0980685568e+00   3.6734544040e-03   2.0944519596e+00   5.6857243254e-05
  20   2.0975199521e+00   3.1248497140e-03   2.0944380229e+00   4.2920520874e-05
  21   2.0944873314e+00   9.2229007826e-05   2.0944275023e+00   3.2399890542e-05
  22   2.0926582116e+00  -1.7368907602e-03   2.0944195604e+00   2.4458037757e-05
  23   2.0926253662e+00  -1.7697361592e-03   2.0944135653e+00   1.8462874415e-05
  24   2.0935219110e+00  -8.7319137575e-04   2.0944090396e+00   1.3937238852e-05
```

As can been seen, the results with higher precision are equally disappointing and consistent with the results obtained using standard floating point arithmetic. This indicates that the suggested method simply does not work well.

Upon examining the method, notice that F_n is actually the value of f evaluated at $x = (x_n + 2x_{n-1} + x_{n-2})/4$ and that both F'_n and F''_n are good approximations for the derivative and second derivatives at this same point. Therefore, a significantly better approximation of h is given by

$$h\Big(\frac{x_n + 2x_{n-1} + x_{n-2}}{4}\Big) \approx \frac{x_n + 2x_{n-1} + x_{n-2}}{4} - \frac{FnF'_n}{[F'_n]^2 - F_nF''_n}.$$

This suggests the modified extra-credit method

$$x_{n+1} = \frac{x_n + 2x_{n-1} + x_{n-2}}{4} - \frac{FnF'_n}{[F'_n]^2 - F_nF''_n}.$$

Modifying the code as

```
restart;
kernelopts(printbytes=false):
printf("#p5i4.mpl\n");
Digits:=1000:
f2:=x->2*cos(5*x)+2*cos(4*x)+6*cos(3*x)+4*cos(2*x)+10*cos(x)+3:
f1:=x->x*x-2:
H:=proc(x2,x1,x0,f)
    local x,y,dy,ddy;
    x:=(x2+2*x1+x0)/4;
    y:=f(x);
    dy:=2/(x2-x0)*(f((x2+x1)/2)-f((x1+x0)/2));
    ddy:=2/(x2-x0)*((f(x2)-f(x1))/(x2-x1)-(f(x1)-f(x0))/(x1-x0));
    x-y*dy/(dy*dy-y*ddy);
end proc:
G:=proc(x1,x0,f)
    local y,dy;
    y:=f(x1);
    dy:=(f(x1)-f(x0))/(x1-x0);
    x1-y/dy;
end proc:

iterate:=proc(p2,p1,p0,xinf,f)
    local i,y0,y1,y,x2,x1,x0,x;
    y0:=p0; y1:=p1; x0:=p0; x1:=p1; x2:=p2;
    printf("#%40s %37s\n",
        "the modified extra-credit method","the usual secant method");
    printf("#%2s %18s %18s %18s %18s\n",
        "n","xn","en","yn","en");
    printf("%3d %18.10e %18.10e %18.10e %18.10e\n",
        0,x0,x0-xinf,y0,y0-xinf);
    printf("%3d %18.10e %18.10e %18.10e %18.10e\n",
        1,x1,x1-xinf,y1,y1-xinf);
    y:=G(y1,y0,f);
    y0:=y1; y1:=y;
    i:=2;
    while(true)
    do
        printf("%3d %18.10e %18.10e %18.10e %18.10e\n",
            i,x2,x2-xinf,y1,y1-xinf);
        if i>=24 then
            break;
        end;
        y:=G(y1,y0,f);
        y0:=y1; y1:=y;
        x:=H(x2,x1,x0,f);
        x0:=x1; x1:=x2; x2:=x;
        i:=i+1;
    end;
end proc:

printf("\n" "#f(x)=x^2-2,\n");
iterate(1.0,0.9,1.1,sqrt(2),f1):
printf("\n\n"
    "#f(x)=2*cos(5*x)+2*cos(4*x)+6*cos(3*x)+4*cos(2*x)+10*cos(x)+3,\n");
iterate(1.0,0.9,1.1,evalf(Pi/3),f2):
printf("\n\n"
    "#f(x)=2*cos(5*x)+2*cos(4*x)+6*cos(3*x)+4*cos(2*x)+10*cos(x)+3,\n");
iterate(2.0,1.9,2.1,evalf(2*Pi/3),f2):
```

now leads to the output

```
#p5i4.mpl

#f(x)=x^2-2,
#       the modified extra-credit method              the usual secant method
# n                 xn                en                 yn                en
  0   1.1000000000e+00  -3.1421356237e-01   1.1000000000e+00  -3.1421356237e-01
  1   9.0000000000e-01  -5.1421356237e-01   9.0000000000e-01  -5.1421356237e-01
  2   1.0000000000e+00  -4.1421356237e-01   1.4950000000e+00   8.0786437627e-02
  3   1.3217538657e+00  -9.2459696667e-02   1.3968684760e+00  -1.7345086381e-02
  4   1.3557550322e+00  -5.8458530180e-02   1.4137290148e+00  -4.8454753414e-04
  5   1.4034833411e+00  -1.0730221235e-02   1.4142165527e+00   2.9902961427e-06
  6   1.4131005588e+00  -1.1130035803e-03   1.4142135619e+00  -5.1236510311e-10
  7   1.4140663756e+00  -1.4718676151e-04   1.4142135624e+00  -5.4168684728e-16
  8   1.4142097595e+00  -3.8028571806e-06   1.4142135624e+00   9.8125716225e-26
  9   1.4142135184e+00  -4.4015756028e-08   1.4142135624e+00  -1.8792568277e-41
 10   1.4142135618e+00  -5.2977755619e-10   1.4142135624e+00  -6.5196455153e-67
 11   1.4142135624e+00  -3.3461946029e-13   1.4142135624e+00   4.3317673776e-108
 12   1.4142135624e+00  -4.4897167384e-17   1.4142135624e+00 -9.9849091070e-175
 13   1.4142135624e+00  -6.2175424205e-21   1.4142135624e+00 -1.5291998567e-282
 14   1.4142135624e+00  -2.4755430393e-27   1.4142135624e+00   5.3983789937e-457
 15   1.4142135624e+00  -4.4567004584e-35   1.4142135624e+00 -2.9186540856e-739
 16   1.4142135624e+00  -8.5422687268e-43   1.4142135624e+00   0.0000000000e+00
 17   1.4142135624e+00  -1.3541788224e-55   1.4142135624e+00   0.0000000000e+00
 18   1.4142135624e+00  -4.3889632875e-71                NaN               NaN
 19   1.4142135624e+00  -1.6124322765e-86                NaN               NaN
 20   1.4142135624e+00 -4.0521644234e-112                NaN               NaN
 21   1.4142135624e+00 -4.2565615733e-143                NaN               NaN
 22   1.4142135624e+00 -5.7451052552e-174                NaN               NaN
 23   1.4142135624e+00 -3.6283497395e-225                NaN               NaN
 24   1.4142135624e+00 -4.0036201279e-287                NaN               NaN
```

```
#f(x)=2*cos(5*x)+2*cos(4*x)+6*cos(3*x)+4*cos(2*x)+10*cos(x)+3,
#       the modified extra-credit method             the usual secant method
# n                xn                en                yn                en
  0    1.1000000000e+00   5.2802448803e-02  1.1000000000e+00   5.2802448803e-02
  1    9.0000000000e-01  -1.4719755120e-01  9.0000000000e-01  -1.4719755120e-01
  2    1.0000000000e+00  -4.7197551197e-02  1.1196553035e+00   7.2457752263e-02
  3    1.0408955138e+00  -6.3020374078e-03  1.1618045574e+00   1.1460700620e-01
  4    1.0478435726e+00   6.4602138730e-04  1.0860467711e+00   3.8849219907e-02
  5    1.0472148484e+00   1.7297164971e-05  1.0739484024e+00   2.6750851211e-02
  6    1.0471963142e+00  -1.2370318910e-06  1.0624733436e+00   1.5275792428e-02
  7    1.0471975534e+00   2.1803682599e-09  1.0567157632e+00   9.5182120399e-03
  8    1.0471975512e+00  -5.7627693341e-12  1.0529893596e+00   5.7918084065e-03
  9    1.0471975512e+00  -6.7737008472e-16  1.0507714277e+00   3.5738764925e-03
 10    1.0471975512e+00  -3.1832974585e-21  1.0493976366e+00   2.2000853617e-03
 11    1.0471975512e+00   9.8589290019e-28  1.0485555420e+00   1.3579907807e-03
 12    1.0471975512e+00   5.4465089372e-37  1.0480358121e+00   8.3826085444e-04
 13    1.0471975512e+00  -7.9272859986e-49  1.0477153215e+00   5.1777029539e-04
 14    1.0471975512e+00   1.3563300676e-64  1.0475174150e+00   3.1986375573e-04
 15    1.0471975512e+00  -1.0905866508e-85  1.0473951908e+00   1.9763963646e-04
 16    1.0471975512e+00 -2.7158597973e-113  1.0473196801e+00   1.2212893398e-04
 17    1.0471975512e+00 -3.7363108141e-150  1.0472730240e+00   7.5472815680e-05
 18    1.0471975512e+00  7.4814358041e-199  1.0472441932e+00   4.6642023276e-05
 19    1.0471975512e+00  2.5631131171e-263  1.0472263765e+00   2.8825323416e-05
 20    1.0471975512e+00 -7.0606613286e-349  1.0472153658e+00   1.7814631862e-05
 21    1.0471975512e+00  4.8436210389e-462  1.0472085611e+00   1.1009896904e-05
 22    1.0471975512e+00 -4.5712037976e-612  1.0472043556e+00   6.8044325976e-06
 23    1.0471975512e+00 -2.3995118497e-541  1.0472017565e+00   4.2053485499e-06
 24    1.0471975512e+00  8.4224161806e-540  1.0472001502e+00   2.5990398979e-06
```

```
#f(x)=2*cos(5*x)+2*cos(4*x)+6*cos(3*x)+4*cos(2*x)+10*cos(x)+3,
#       the modified extra-credit method             the usual secant method
# n                xn                en                yn                en
  0    2.1000000000e+00   5.6048976068e-03  2.1000000000e+00   5.6048976068e-03
  1    1.9000000000e+00  -1.9439510239e-01  1.9000000000e+00  -1.9439510239e-01
  2    2.0000000000e+00  -9.4395102393e-02  2.0999938553e+00   5.5987528838e-03
  3    2.0574700223e+00  -3.6925080106e-02  2.0999877310e+00   5.5926285676e-03
  4    2.0755870908e+00  -1.8808011565e-02  2.0981284969e+00   3.7333944904e-03
  5    2.0880569331e+00  -6.3381692487e-03  2.0973429599e+00   2.9478575555e-03
  6    2.0911325294e+00  -3.2625729642e-03  2.0965823314e+00   2.1872289814e-03
  7    2.0932012130e+00  -1.1938893671e-03  2.0960576651e+00   1.6625627294e-03
  8    2.0938740404e+00  -5.2106196553e-04  2.0956471123e+00   1.2520099219e-03
  9    2.0941601316e+00  -2.3497076052e-04  2.0953412813e+00   9.4617891297e-04
 10    2.0943159425e+00  -7.9159898169e-05  2.0951091499e+00   7.1404747249e-04
 11    2.0943502700e+00  -4.4832377454e-05  2.0949342410e+00   5.3913856184e-04
 12    2.0943822799e+00  -1.2822523221e-05  2.0948020862e+00   4.0698382904e-04
 13    2.0943872100e+00  -7.8924277316e-06  2.0947023453e+00   3.0724294633e-04
 14    2.0943926121e+00  -2.4902694840e-06  2.0946270388e+00   2.3193644459e-04
 15    2.0943938549e+00  -1.2475161902e-06  2.0945701909e+00   1.7508851260e-04
 16    2.0943945739e+00  -5.2847376575e-07  2.0945272751e+00   1.3217269774e-04
 17    2.0943949259e+00  -1.7646665457e-07  2.0944948781e+00   9.9775664135e-05
 18    2.0943949945e+00  -1.0785524205e-07  2.0944704216e+00   7.5319203891e-05
 19    2.0943950770e+00  -2.5373722073e-08  2.0944519596e+00   5.6857243254e-05
 20    2.0943950825e+00  -1.9917986487e-08  2.0944380229e+00   4.2920520874e-05
 21   2.0943950974e+00  -5.0005072883e-09   2.0944275023e+00   3.2399890542e-05
 22   2.0943950992e+00  -3.2378697850e-09   2.0944195604e+00   2.4458037757e-05
 23   2.0943951012e+00  -1.1724680388e-09   2.0944135653e+00   1.8462874415e-05
 24   2.0943951019e+00  -4.5095734652e-10   2.0944090396e+00   1.3937238852e-05
```

The secant method still does better for finding the root of $x^2 - 2$, which is not unexpected as this is a root of multiplicity one. However, the modified extra-credit method now does noticeably better than the secant method when looking for roots of higher multiplicity. The speed of convergence is particularly satisfying when finding the root near 1 of multiplicity 2 which satisfies

$$2\cos(5x) + 2\cos(4x) + 6\cos(3x) + 4\cos(2x) + 10\cos(x) + 3 = 0.$$

Another way to do better than the secant method is to simply use Aitken's Δ^2 method to accelerate the convergence of the secant method. In particular we define

$$\Delta^2 x_n = x_{n-2} - (x_{n-1} - x_{n-2})^2/(x_n - 2x_{n-1} + x_{n-2})$$

and check how fast $\Delta^2 x_n$ converges. The code

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

double f(double x){
    return 2*cos(5*x)+2*cos(4*x)+6*cos(3*x)+4*cos(2*x)+10*cos(x)+3;
}
double G(double x1,double x0){
    double y=f(x1);
    double dy=(f(x1)-f(x0))/(x1-x0);
    return x1-y/dy;
}
void iterate(double x1,double x0,double xinf){
    printf("#%2s %18s %18s %18s %18s\n",
        "n","xn","en","delta^2 xn","en");
    printf("%3d %18.10e %18.10e\n",
        0,x0,x0-xinf);
    printf("%3d %18.10e %18.10e\n",
        1,x1,x1-xinf);
    double x2=G(x1,x0);
    for(int i=2;;i++){
        double dx=x1-x0;
        double p2=x0-dx*dx/(x2-2*x1+x0);
        printf("%3d %18.10e %18.10e %18.10e %18.10e\n",
            i,x2,x2-xinf,p2,p2-xinf);
        if(i>=24) break;
        x0=x1; x1=x2;
        x2=G(x1,x0);
    }
}

int main(){
    printf("#p5i3.c\n");
    printf("\n"
        "#f(x)=2*cos(5*x)+2*cos(4*x)+6*cos(3*x)+4*cos(2*x)+10*cos(x)+3,\n");
    iterate(0.9,1.1,M_PI/3);
    printf("\n\n"
        "#f(x)=2*cos(5*x)+2*cos(4*x)+6*cos(3*x)+4*cos(2*x)+10*cos(x)+3,\n");
    iterate(1.9,2.1,2*M_PI/3);
    return 0;
}
```

produces the output

```
#p5i3.c

#f(x)=2*cos(5*x)+2*cos(4*x)+6*cos(3*x)+4*cos(2*x)+10*cos(x)+3,
# n                xn                 en         delta^2 xn                en
   0   1.1000000000e+00   5.2802448803e-02
   1   9.0000000000e-01  -1.4719755120e-01
   2   1.1196553035e+00   7.2457752263e-02   1.0046836781e+00  -4.2513873117e-02
   3   1.1618045574e+00   1.1460700620e-01   1.1718130028e+00   1.2461545161e-01
   4   1.0860467711e+00   3.8849219907e-02   1.1347227634e+00   8.7525212220e-02
   5   1.0739484024e+00   2.6750851211e-02   1.0716491271e+00   2.4451575933e-02
   6   1.0624733436e+00   1.5275792428e-02   8.5121892386e-01  -1.9597862734e-01
   7   1.0567157632e+00   9.5182120399e-03   1.0509177996e+00   3.7202483860e-03
   8   1.0529893596e+00   5.7918084065e-03   1.0461528871e+00  -1.0446640998e-03
   9   1.0507714277e+00   3.5738764925e-03   1.0475103642e+00   3.1281305166e-04
  10   1.0493976366e+00   2.2000853617e-03   1.0471618696e+00  -3.5681595907e-05
  11   1.0485555420e+00   1.3579907807e-03   1.0472218427e+00   2.4291542510e-05
  12   1.0480358121e+00   8.3826085446e-04   1.0471978815e+00   3.3029266833e-07
  13   1.0477153215e+00   5.1777029537e-04   1.0471997898e+00   2.2386515470e-06
  14   1.0475174150e+00   3.1986375576e-04   1.0471979035e+00   3.5230084605e-07
  15   1.0473951908e+00   1.9763963651e-04   1.0471978037e+00   2.5251204128e-07
  16   1.0473196801e+00   1.2212893400e-04   1.0471976196e+00   6.8365705985e-08
  17   1.0472730240e+00   7.5472815601e-05   1.0471975839e+00   3.2692268892e-08
  18   1.0472441932e+00   4.6642022625e-05   1.0471975621e+00   1.0917508453e-08
  19   1.0472263765e+00   2.8825323730e-05   1.0471975557e+00   4.5458055098e-09
  20   1.0472153658e+00   1.7814632219e-05   1.0471975528e+00   1.6442622819e-09
  21   1.0472085611e+00   1.1009896627e-05   1.0471975518e+00   6.4514082965e-10
  22   1.0472043556e+00   6.8044334973e-06   1.0471975514e+00   2.5255775249e-10
  23   1.0472017565e+00   4.2053526790e-06   1.0471975513e+00   1.1395617783e-10
  24   1.0472001502e+00   2.5990370549e-06   1.0471975512e+00  -1.6476597864e-11
```

```
#f(x)=2*cos(5*x)+2*cos(4*x)+6*cos(3*x)+4*cos(2*x)+10*cos(x)+3,
# n               xn               en         delta^2 xn               en
  0   2.1000000000e+00  5.6048976068e-03
  1   1.9000000000e+00 -1.9439510239e-01
  2   2.0999938553e+00  5.5987528838e-03   1.9999984638e+00  -9.4396638598e-02
  3   2.0999877310e+00  5.5926285676e-03   2.0999877311e+00   5.5926287551e-03
  4   2.0981284968e+00  3.7333944315e-03   2.0999938755e+00   5.5987731239e-03
  5   2.0973429599e+00  2.9478575233e-03   2.0967682465e+00   2.3731440768e-03
  6   2.0965823314e+00  2.1872289569e-03   2.0733549396e+00  -2.1040162832e-02
  7   2.0960576651e+00  1.6625627019e-03   2.0948910607e+00   4.9595826620e-04
  8   2.0956471123e+00  1.2520099139e-03   2.0941700422e+00  -2.2506020014e-04
  9   2.0953412813e+00  9.4617891509e-04   2.0944481281e+00   5.3025726996e-05
 10   2.0951091498e+00  7.1404745095e-04   2.0943780052e+00  -1.7097146740e-05
 11   2.0949342409e+00  5.3913855676e-04   2.0943996071e+00   4.5047348691e-06
 12   2.0948020862e+00  4.0698380650e-04   2.0943935906e+00  -1.5117546379e-06
 13   2.0947023452e+00  3.0724283397e-04   2.0943954306e+00   3.2823474694e-07
 14   2.0946270388e+00  2.3193641692e-04   2.0943949472e+00  -1.5523914598e-07
 15   2.0945701907e+00  1.7508835391e-04   2.0943951100e+00   7.6147572692e-09
 16   2.0945272742e+00  1.3217179595e-04   2.0943950680e+00  -3.4375686297e-08
 17   2.0944948784e+00  9.9775979922e-05   2.0943951241e+00   2.1713285392e-08
 18   2.0944704242e+00  7.5321841337e-05   2.0943951247e+00   2.2271672506e-08
 19   2.0944519678e+00  5.6865400062e-05   2.0943951726e+00   7.0232361704e-08
 20   2.0944380226e+00  4.2920173231e-05   2.0943949146e+00  -1.8780884936e-07
 21   2.0944274954e+00  3.2393005667e-05   2.0943950731e+00  -2.9269143553e-08
 22   2.0944195788e+00  2.4476456089e-05   2.0943955724e+00   4.6997272962e-07
 23   2.0944135338e+00  1.8431410462e-05   2.0943940080e+00  -1.0943707163e-06
 24   2.0944089692e+00  1.3866784173e-05   2.0943948949e+00  -2.0748021701e-07
```

The results converge faster than just the secant method, however not as fast as the correctly modified extra credit method. No further ideas for improvements on the secant method in the case of multiple roots were explored.