Problem 1. (a) According to a standard trigonometric identity, the following two expressions are equal in exact arithmetic:

\[
\begin{align*}
    x_1 &= \cos(2 + \delta) - \cos(2 - \delta) \\
    x_2 &= -2 \sin(2) \sin(\delta).
\end{align*}
\]

Use Matlab or other numerical software to evaluate approximations \( \hat{x}_1 \) and \( \hat{x}_2 \) in double-precision arithmetic for \( \delta = 10^{-n}, \ n = 4, 5, 6, 7, 8 \). For each choice of the integer \( n \) how many significant figures do \( \hat{x}_1 \) and \( \hat{x}_2 \) have in common?

(b) Which of the two approximations \( \hat{x}_1 \) or \( \hat{x}_2 \) do you believe is closer to the exact result \( x_1 = x_2 \), and why? Is either one accurate to double precision?

(c) How many significant figures are there in IEEE standard floating point double precision arithmetic? Using this result, explain the magnitude of the relative errors observed in part (a).

(d) Now consider a third approximation

\[
x_3 = -2 \sin(2) \delta
\]

which satisfies \( \lim_{\delta \to 0} (x_3/x_2) = 1 \) in exact arithmetic. Again use Matlab or other numerical software to evaluate approximations \( \hat{x}_2 \) and \( \hat{x}_3 \) in double-precision arithmetic for \( \delta = 10^{-n}, \ n = 4, 5, 6, 7, 8 \). For each choice of the integer \( n \) how many significant figures do \( \hat{x}_2 \) and \( \hat{x}_3 \) have in common?

(e) Explain the magnitude of the relative errors observed in part (d) for all values of the integer \( n \) and in particular for the case \( n = 8 \).

Solution: (a) See the Matlab script `prob1.m` on the following page. The relative differences \( \hat{x}_1/\hat{x}_2 - 1 \) for \( n = 4, 5, 6, 7, 8 \) are, respectively,

\[
9.9986 \times 10^{-13}, \ 5.4117 \times 10^{-12}, \ 3.1645 \times 10^{-11}, \ -3.3481 \times 10^{-10}, \ -6.7449 \times 10^{-9}
\]

so that the number of significant figures in common are

\[
12, \ 11, \ 11, \ 10, \ 8.
\]

(b) The expression \( x_1 \) for small \( \delta \) involves a subtraction of two nearly equal quantities, which will suffer from loss of significance in finite precision arithmetic. On the other hand, \( x_2 \) avoids such a subtraction, so that it can be expected to have full precision.
(c) There are 16 significant digits possible in standard double precision arithmetic. For $\delta = 10^{-n}$ the two subtracted expressions in $x_1$ should have about $n$ digits in common, so one expects that about $16 - n$ digits remain after subtraction, or about

$$12, 11, 10, 9, 8$$

for $n = 4, 5, 6, 7, 8$, respectively. These expectations agree closely with what was observed numerically in part (a).

(d) From `prob1.m`, relative differences $\hat{x}_2/\hat{x}_3 - 1$ for $n = 4, 5, 6, 7, 8$ are, respectively,

$$-1.6667 \times 10^{-9}, \quad -1.6667 \times 10^{-11}, \quad -1.6653 \times 10^{-13}, \quad -1.7764 \times 10^{-15}, \quad 0,$$

and the number of significant digits in common are

$$9, 11, 13, 15, 16.$$

In fact, for $n = 8$, the two approximations $\hat{x}_3 = \hat{x}_2$ identically!

(e) Since

$$\sin(\delta) = \delta - \frac{\delta^3}{6} + O(\delta^5)$$

the relative difference between $\hat{x}_2$ and $\hat{x}_3$ in exact arithmetic is

$$\frac{\hat{x}_2}{\hat{x}_3} - 1 = \frac{\sin \delta - \delta}{\delta} = -\frac{1}{6} \delta^2 + O(\delta^4)$$

This analytical formula agrees well with the numerical results in part (d), although effects of round-off error become noticeable for the larger $n$. When $n = 8$ then

$$\frac{1}{6} \delta^2 = 1.667 \times 10^{-17} < \text{eps}/2 = 1.1102 \times 10^{-16}$$

the unit round in double precision arithmetic, so that $\hat{x}_3 = \hat{x}_2$ to machine precision.
a=2;

disp('x versus y')
disp(' ')

for ii=4:8

del=10^(-ii)
x=cos(a+del)-cos(a-del)
y=-2*sin(a)*sin(del)
relerr=x/y-1

disp(' ')

end

pause

disp('z versus y')
disp(' ')

for ii=4:8

del=10^(-ii)
y=-2*sin(a)*sin(del)
z=-2*sin(a)*del
relerr2=y/z-1

disp(' ')

end
Problem 2. a) Use Taylor series with remainder to prove that
\[ \frac{f(x + h) - f(x - h)}{2h} = f'(x) + \frac{1}{6} f'''(\xi) h^2 \]
for \( f \in C^3 \) and some \( \xi \) between \( x - h, x + h \). You will need to use also the Intermediate Value Theorem to get the form of the remainder stated above.

(b) Use part (a) to estimate the total error in approximating the derivative \( f'(x) \) with the difference quotient
\[ \frac{\hat{f}(x + h) - \hat{f}(x - h)}{2h} \]
where \( \hat{f}(x) \) is the function evaluated in a computer arithmetic to relative precision \( \epsilon \). Derive an upper bound involving \( h, \epsilon, M_0 = \max_x |f(x)| \) and \( M_3 = \max_x |f'''(x)| \).

(c) Find the value \( h = h^* \) to give the smallest error bound in (b). Using this optimal value in double precision arithmetic, what is the approximate number of digits of accuracy that can be expected in the estimate of \( f'(x) \)? Assume that the quantities \( M_0, M_3 \) are of order unity.

(d) For the function \( f(x) = e^x \) evaluate the approximation in part (b) for \( x = 0 \) using double-precision arithmetic and \( h = 10^{-n} \), for integers \( 1 \leq n \leq 10 \) and calculate the error in this approximation of \( f'(0) \). Plot relative error versus \( n \) using semilogy in Matlab. Does the plot agree with the error bounds and optimal value \( h^* \) in part (c)?

Solution: (a) \( f(x \pm h) = f(x) \pm f'(x)h + \frac{1}{2} f''(x)h^2 \pm \frac{1}{6} f'''(\xi_{\pm})h^3 \), for some \( \xi_{\pm} \) between \( x \) and \( x \pm h \). Thus,
\[ \frac{f(x + h) - f(x - h)}{2h} = f'(x) + \frac{1}{6} \left( \frac{f'''(\xi_+) + f'''(\xi_-)}{2} \right) h^2. \]
Note that the average \( \frac{f'''(\xi_+) + f'''(\xi_-)}{2} \) lies between the values \( f'''(\xi_+), f'''(\xi_-) \). Thus the intermediate value theorem for the continuous function \( f'''(x) \) implies that there is some \( \xi \) between \( \xi_-, \xi_+ \) and, thus, between \( x - h, x + h \), such that
\[ f'''(\xi) = \frac{1}{2} (f'''(\xi_-) + f'''(\xi_+)). \]
Substituting gives the desired result.

(b) The rounding error is \( |\hat{f}(x) - f(x)| = |f(x)| \epsilon \leq M_0 \epsilon \). Thus,
\[ \left| \frac{\hat{f}(x + h) - \hat{f}(x - h)}{2h} - \left( \frac{f(x + h) - f(x - h)}{2h} \right) \right| \leq \frac{M_0 \epsilon}{h}. \]
From part (a), the truncation error is bounded by $\frac{1}{6}M_3h^2$. Thus the total error is bounded by

$$\text{err} = \frac{M_0\epsilon}{h} + \frac{1}{6}M_3h^2.$$  

(c) Taking the derivative to minimize the error gives $-\frac{M_0\epsilon}{h^2} + \frac{1}{3}M_3h$, or $h_\ast = (3M_0\epsilon/M_3)^{1/3}$, i.e. $h_\ast \approx 10^{-5}$ for $\epsilon = 10^{-16}$. Substituting into the error gives $1.5 \left( \frac{M_3M_0^2\epsilon^2}{3} \right)^{1/3} \approx 10^{-11}$. Thus, one expects optimally 11 digits of accuracy in double precision.

(d) See the Matlab script `prob2.m` for the calculation in double precision of the error

$$\text{err} = \frac{e^h - e^{-h}}{h} - 1$$

in $f'(0) = 1$ for $h = 10^{-n}$ and $n = 1, 2, \ldots, 10$, with results plotted below:

![Graph of Relative Error in Calculation of Df(0)=exp(0)=1](image)

The linear decrease for $n < 5$ corresponds to the logarithm of the truncation error

$$\log_{10}(M_3h^2/6) = -2n + \log_{10}(M_3/6)$$

with $h = 10^{-n}$. The minimum error is about $10^{-11}$ at $n = 5$, as predicted in part (c), and for $n > 5$ there is roughly linear growth consistent with the round-off error

$$\log_{10}(M_0\epsilon/h) = n + \log_{10}(M_0\epsilon).$$
for n=1:10
    h=10^(-n);
    relerr(n)=abs((exp(h)-exp(-h))/2/h-1);
end

semilogy(relerr,'LineWidth',2);
xlabel('n','FontSize',15);
ylabel('relerr','FontSize',15);
title('Relative Error in Calculation of Df(0)=exp(0)=1','FontSize',18);
print('prob2-fig.pdf','-dpdf')
Problem 3. (a) Suppose that \( f \in C[0,1] \), i.e. that \( f \) is a continuous function of \( x \) over the closed interval \( 0 \leq x \leq 1 \). Consider the problem of finding the anti-derivative function \( F \in C^1[0,1] \), i.e. the element \( F(x) \) in the space of continuously differentiable functions such that \( F'(x) = f(x) \). Does \( F \) exist? Is \( F \) unique? If not, what additional condition is required to make the anti-derivative \( F \) unique?

(b) Show for any function \( f_N \in C[0,1] \) approximating \( f \) that
\[
\max_{x \in [0,1]} |F_N(x) - F(x)| \leq \max_{x \in [0,1]} |f_N(x) - f(x)|.
\]
Is the problem of finding the anti-derivative well-posed? Explain your answer.

(c) Suppose now that \( f \in C^1[0,1] \). Show that the function defined by
\[
f_N(x) = f(x) + \frac{1}{N} \sin(2\pi N^2 x)
\]
is also in \( C^1[0,1] \) for all integers \( N \) and
\[
\max_{x \in [0,1]} |f(x) - f_N(x)| = \frac{1}{N}
\]
but
\[
\max_{x \in [0,1]} |f'(x) - f'_N(x)| = 2\pi N.
\]

(d) Is finding the derivative \( f' \in C[0,1] \) of a function \( f \in C^1[0,1] \) a well-posed problem? Explain in what sense it is or is not.

Solution: (a) By the Fundamental Theorem of Calculus, an anti-derivative exists for any \( f \in C[0,1] \) and is given by the definite integral
\[
F(x) = \int_0^x f(y) \, dy + C.
\]
The result is not unique, however, and depends upon the integration constant \( C \). The latter can be fixed uniquely by the condition \( f(0) = C \).

(b) From part (a),
\[
F_N(x) - F(x) = \int_0^x [f_N(y) - f(y)] \, dy
\]
for any \( x \in [0,1] \), so that
\[
|F_N(x) - F(x)| \leq \int_0^x |f_N(y) - f(y)| \, dy \leq \max_{y \in [0,1]} |f_N(y) - f(y)|.
\]
and taking the maximum over \( x \) gives
\[
\max_{x \in [0,1]} |F_N(x) - F(x)| \leq \max_{y \in [0,1]} |f_N(y) - f(y)|.
\]
It follows that
\[
\lim_{N \to \infty} \max_{x \in [0,1]} |f_N(x) - f(x)| = 0 \implies \lim_{N \to \infty} \max_{x \in [0,1]} |F_N(x) - F(x)| = 0.
\]

Hence, the output of the problem, the anti-derivative $F$, is continuous in the input, the function $f$, so that $F_N \to F$ when $f_N \to f$. Since the solution exists, is unique under condition $F(0) = C$, and is continuous in the data, the problem is well-posed.

(c) Since the maximum of $|\sin x|$ for $x \in [0, 2\pi]$ is 1, it follows easily that
\[
\max_{x \in [0,1]} |f(x) - f_N(x)| = \frac{1}{N} \max_{x \in [0,2\pi N^2]} |\sin(x)| = \frac{1}{N}.
\]
Furthermore,
\[
f_N'(x) - f'(x) = 2\pi N \cos(2\pi N^2 x)
\]
and the maximum of $|\cos x|$ for $x \in [0, 2\pi]$ is also 1 so that
\[
\max_{x \in [0,1]} |f_N'(x) - f'(x)| = 2\pi N \max_{x \in [0,2\pi N^2]} |\cos(x)| = 2\pi N.
\]

(d) For any $f \in C^1[0,1]$, by definition, the derivative $f'$ exists, is unique, and belongs to $C[0,1]$. The only issue is whether the output, the derivative $f'$, is continuous in the input, the function $f$. As we see from the results in part (c), it is possible that
\[
\lim_{N \to \infty} \max_{x \in [0,1]} |f_N(x) - f(x)| = 0 \quad \text{but} \quad \lim_{N \to \infty} \max_{x \in [0,1]} |f_N'(x) - f'(x)| = \infty.
\]
Thus, evaluating the derivative is not continuous in the data, unless one imposes a stronger meaning to $f_N \to f$.

Remark: As an example of the latter, if one requires that
\[
\max_{x \in [0,1]} |f_N(x) - f(x)| + \max_{x \in [0,1]} |f_N'(x) - f'(x)| \to 0
\]
as $N \to 0$, then taking the derivative is well-posed for this meaning of $f_N \to f$. The difficulty here is that one must know the output $f'$ itself in order to specify the sense of convergence of the input $f$!
Problem 4. In this problem we consider two relatively complicated functions, the sine integral function
\[ f(x) = \text{Si}(x) := \int_0^x \frac{\sin t}{t} \, dt \]
which is given in Matlab by \texttt{sinint} and the function
\[ f(x) = \sin(x) \sin(2x) \sin(3x) \sin(5x) \sin(7x). \]
We shall invert these functions by solving the equation \( f(x) = y \) for particular values of \( y \) using both the Newton method and secant method, and compare their efficiency.

(a) Rewrite the Matlab script \texttt{newton.m} from the course website with error tolerance \( \texttt{tol} = 10^{-15} \) and maximum iterations \( \texttt{maxit} = 1000 \) as a function file in the format
\[ [xx,ee] = \text{newton}(\@x F(x), \@x DF(x), x0) \]
with function \( F \), its derivative \( F' \), and an initial guess \( x_0 \) as inputs, and the vector of successive iterates \( x = (x_0, x_1, ..., x_N) \) and the corresponding vector of error estimates \( e = (|x_0 - x_1|, |x_1 - x_0|, ..., |x_N - x_{N-1}|) \) as outputs (for \( x_1 = 0 \)). Likewise, rewrite the script \texttt{secant.m} as a function file in the format
\[ [xx,ee] = \text{secant}(\@x F(x), x0) \]
with \( x_0 = a \) and \( x_1 = b = a - F(a)/F'(a) \) the two initial guesses.

(b) For function (i), use Newton’s method and secant method with \( F(x) = f(x) - y \) for \( y = 1 \) to find the value \( x_* \) such that \( f(x_*) = \text{Si}(x_*) = 1 \). Set \( x_0 = 1 \) for both methods. For secant method, generate the second guess \( x_1 = b \) by one Newton iteration. For each method, record the iterates, the estimated errors, and the wall clock time. To estimate the latter accurately, average over 100 trials using the loop
\[
\text{num} = 100 \\
\text{for} \ ii = 1: \text{num} \\
\quad \text{tic} \\
\quad [xxn, een] = \text{newton}(\@x F(x), \@x DF(x), x0); \\
\quad \text{timen} = \text{timen} + \text{toc}; \\
\quad \text{tic} \\
\quad b = a - F(a)/DF(a); \\
\quad [xxs, ees] = \text{secant}(\@x F(x), a, b); \\
\quad \text{times} = \text{times} + \text{toc}; \\
\quad \text{end} \\
\quad \text{timen} = \text{timen} / \text{num}; \\
\quad \text{times} = \text{times} / \text{num};
\]
Which method requires fewer iterations? Which method requires a smaller wall clock time? Explain your results in terms of the clock time required to evaluate each of the functions \( F(x) \) and \( F'(x) \) for one value of \( x \).

(c) Repeat part (b) for function (ii), \( y = 0.1 \), and \( x_0 = a = 0.25 \).
Solution: (a) For examples of Matlab function files newton.m and secant.m, see the following page.

(b) For the first function in (i), \( f(x) = \text{Si}(x) \) and

\[
 f'(x) = \frac{\sin x}{x}.
\]

The implementation of the root-finding is given in the Matlab script prob4.m. The Newton method obtains the root in 5 iterations

\[
\begin{array}{ll}
 x_n & e_n \\
 1.00000000000000 & 0.064074615294223 \\
 1.064074615294223 & 0.000764997615007 \\
 1.064839612909230 & 0.000000112627329 \\
 1.064839725536558 & 0.000000000000002 \\
 1.064839725536561 & 0 \\
\end{array}
\]

while secant requires 7 iterations (including the first Newton step)

\[
\begin{array}{ll}
 x_n & e_n \\
 1.00000000000000 & 0.064074615294223 \\
 1.064074615294223 & 0.000755826780440 \\
 1.064830442074663 & 0.000009282094888 \\
 1.064839724169551 & 0.000000001367007 \\
 1.064839725536558 & 0.000000000000002 \\
 1.064839725536561 & 0 \\
\end{array}
\]

The Newton method also requires less wall clock time than secant, only about 71% of the time, as calculated by prob4.m. In the present example (i), it is much more time-consuming to calculate \( f(x) \) than it is to calculate \( f'(x) \). In fact, averaging over many evaluations in prob4.m, we find evaluating \( f(x) = \text{Si}(x) \) is about 3500 times more expensive than \( f'(x) = \sin x/x \). Thus the additional computation by Newton method of \( f'(x) \) once per iteration takes very little extra time compared with secant method, so that the two methods spend about the same amount of clock time per iteration. In that case, the higher order of convergence of Newton method, 2 compared with 1.618 for secant, wins and Newton obtains the root faster by a ratio of \( 5/7 = 71\% \).

(c) For the second case (ii), the derivative of \( f \) is obtained from the product rule:

\[
 f'(x) = \cos(x) \sin(2x) \sin(3x) \sin(5x) \sin(7x) + \sin(x) \cdot 2 \cos(2x) \sin(3x) \sin(5x) \sin(7x) \\
 + \sin(x) \sin(2x) \cdot 3 \cos(3x) \sin(5x) \sin(7x) + \sin(x) \sin(2x) \sin(3x) \cdot 5 \cos(5x) \sin(7x) \\
 + \sin(x) \sin(2x) \sin(3x) \sin(5x) \cdot 7 \cos(7x)
\]

The implementation of the root-finding is again given in the Matlab script prob4.m.
The Newton method obtains the root in 5 iterations:

\[
\begin{array}{cc}
x_n & e_n \\
0.250000000000000 & 0.029001621830488 \\
0.279001621830488 & 0.001447853632960 \\
0.280449475463448 & 0.000010208541790 \\
0.280459684005237 & 0.00000000524166 \\
0.280459684529403 & 0 \\
\end{array}
\]

Newton:

while secant requires 8 iterations:

\[
\begin{array}{cc}
x_n & e_n \\
0.250000000000000 & 0.029001621830488 \\
0.279001621830488 & 0.001325071703099 \\
0.280326693533587 & 0.000132039889260 \\
0.280458733422846 & 0.000000950471574 \\
0.280459683894420 & 0.000000000000003 \\
0.280459684529403 & 0 \\
\end{array}
\]

secant:

However, the Newton method requires more wall clock time than secant, about 175% of the time, as calculated by `prob4.m`. In the present example (ii), it is more time-consuming to calculate \( f'(x) \) than it is to calculate \( f(x) \). In fact, averaging over many evaluations in `prob4.m`, we find evaluating \( f'(x) \) takes 150-160% of the time to evaluate \( f(x) \). Thus the additional computation by Newton method of \( f'(x) \) makes each Newton iteration about 2.5-2.6 more time-consuming than each secant iteration. In this case, the higher order of convergence of Newton method does not offset the longer time per iteration step, and secant method obtains the root faster.
function [xx,ee]=newton(f,Df,x0)

itmax=1000; tol=1e-15;

x=x0;
k=0;
if x ~= 0
    xold=0;
else
    xold=1;
end

while abs(x-xold)>tol*max(abs(x),1.0)
    if k+1>itmax
        break
    end
    xold=x;
    x=x-f(x)/Df(x);
    xx(k+1)=xold;
    ee(k+1)=abs(x-xold);
    k=k+1;
end

xx=xx.'; ee=ee.';
end
function [xx,ee]=secant(f,a,b)

itmax=1000; tol=1e-15;

k=0;
x(1)=a;
ee(1)=abs(a-b);
fa=f(a);
k=1;
while abs(b-a)>tol*max(abs(b),1.0)
    if k+1>itmax
        break
    end
    fb=f(b);
x = b + (b-a)/(fa/fb-1);
x(1)=b;
ee(1)=abs(b-x);
k=k+1;
a=b;
fa=fb;
b=x;
end

xx=xx.'; ee=ee.';
end
G=@(x) sinint(x)-1;
DG=@(x) sin(x)./x;

tol=1e-15;
timen=0;
times=0;
num=100;

for ii=1:num

x0=1;
tic
[xxn,een]=newton(@(x) G(x),@(x) DG(x),x0);
timen=timen+toc;

a=1;
tic
b=a-G(a)/DG(a);
[xxs,ees]=secant(@(x) G(x),a,b);
times=times+toc;
end

itn=length(xxn)
newtn=[xxn,een]
timen=timen/num

disp(' ')
its=length(xxs)+1
secnt=[xxs,ees]
times=times/num

ratio=timen/times

timeG=0;
timeDG=0;
for ii=1:num

x0=1;
tic
y=G(x0);
timeG=timeG+toc;

tic
y=DG(x0);
timeDG=timeDG+toc;
end
timeG=timeG/num
timeDG=timeDG/num

ratioG=timeG/timeDG

pause

tol=1e-15;
timen=0;
times=0;
num=1000;
for ii=1:num
x0=.25;
tic
[xxn,een]=newton(@(x) F(x),@(x) DF(x),x0);
timen=timen+toc;

a=.25;
tic
b=a-F(a)/DF(a);
[xxs,ees]=secant(@(x) F(x),a,b);
times=times+toc;
end

itn=length(xxn)
newtn=[xxn,een]
timen=timen/num

disp(' ')
its=length(xxs)+1
secnt=[xxs,ees]
times=times/num

ratio=timen/times

timeF=0;
timeDF=0;
for ii=1:num
x0=.25;
tic
y=F(x0);
timeF=timeF+toc;

tic
y=DF(x0);
timeDF=timeDF+toc;
end
timeF=timeF/num
timeDF=timeDF/num
ratioDF=timeDF/timeF
function y=f(x)

y=sin(x).*sin(2*x).*sin(3*x).*sin(5*x).*sin(7*x)-0.1;

end

function y=Df(x)

y=cos(x).*sin(2*x).*sin(3*x).*sin(5*x).*sin(7*x);
y=y+sin(x).*2.*cos(2*x).*sin(3*x).*sin(5*x).*sin(7*x);
y=y+sin(x).*sin(2*x).*3.*cos(3*x).*sin(5*x).*sin(7*x);
y=y+sin(x).*sin(2*x).*sin(3*x).*5.*cos(5*x).*sin(7*x);
y=y+sin(x).*sin(2*x).*sin(3*x).*sin(5*x).*7.*cos(7*x);

end