1. Lagrange's estimate using a Taylor approximation centered at c = 0 gives us the equation:

$$sin(x) \ = \ 0 + x + 0 \cdot \frac{x^2}{2!} - cos(\varsigma) \cdot \frac{x^3}{3!}$$

Given that we want a relative accuracy of $\frac{1}{2}\ 10^{-14}$, we can rearrange the equation as such:

$$|sin(x) - x| \ = \ \frac{1}{6}|cos(\varsigma) \cdot x^3| \ < \ \frac{1}{2}\ 10^{-14}$$

For any ς, $|cos(\varsigma)| \leqslant 1$, so we can simplify this to:

$$|x^3| < 3 \times \ 10^{-14}$$

Which will give us a range of x of $x < \pm\sqrt[3]{3 \times \ 10^{-14}}$.

2. a)

$$f(x+h) = f(h) + f'(h)(x-h) + \frac{f''(h)}{2!}(x-h)^2 + \frac{f'''(h)}{3!}(x-h)^3 + \frac{f''''}{4!}(h)(x-h)^4...$$

$$e^{x+2h} \ = e^x + e^x(2h) + e^x\frac{(2h)^2}{2!} + e^x\frac{(2h)^3}{3!} + e^x\frac{(2h)^4}{4!} + e^x\frac{(2h)^5}{5!}...$$

$$e^{x+h} \ = e^x \cdot \sum_{n=0}^{\infty} \frac{(2h)^2}{n!}$$

b)

$$f(x+h) = \sum_{n=0}^{\infty} \frac{f^n(x)}{n!}h^n$$

$$sin(x-3h) = sin(x) + cos(x) \cdot (-3h) - sin(x) \cdot \frac{(-3h)^2}{2!} - cos(x) \cdot \frac{(-3h)^3}{3!} + sin(x) \cdot \frac{(-3h)^4}{4!}...$$

$$sin(x-3h) \ = sin(x) - 3h \cdot cos(x) - \frac{9h^2}{2}sin(x) + \frac{27h^3}{6}cos(x) + \frac{81h^4}{24}sin(x)...$$

3.

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} \implies e^{0.5} = \sum_{n=0}^{\infty} \frac{0.5^n}{n!}$$

The following Matlab script:

```
actual = exp(0.5);
taylor = 0;
diff = abs(actual-taylor);
n = 0;
while diff > 10^-10
    taylor = taylor + 0.5^n/factorial(n);
    n = n + 1;
    diff = abs(actual-taylor);
end
fprintf('Actual:\t%.16f\nTaylor:\t%.16f\nDiff:\t%.16e\nn:\t%d', actual, taylor, diff, n)
```

Gives the following output:

```
Actual:  1.6487212707001282

Taylor:  1.6487212706873655

Diff:    1.2762679801880950e-11

n:       11
```

Thus, we need 11 terms of the series to achieve accuracy of 10^{-10}.

4. Using this Matlab script:

```
a = double(0);
b = double(10^-28);
c = (1-a)*(1+a);
while c == 1
    a = a + b;
    c = (1-a)*(1+a);
end
fprintf('%.12e', a)
```

I determined that the values of a for which the expression $(1-a)(1+a)$ will evaluate to 1 is $|a| < 5.551115123132 \cdot 10^{-17}$.

5.

a) An example where $(a+b)+c \neq a+(b+c)$ is when a and b are several magnitudes greater than c (and within the same order of magnitude of each other), and one is negative and the other positive. If this is the case, $(a+b)$ will cancel, and adding c after will give us the value of c. However, $(b+c)$ will round to b, as it is so much larger than c, and adding a afterwards will be the same as adding $a+b$, and cancel out to 0. For example, if we have

a = 10^300

b = -10^300

c = -10^50
In Matlab, $(a+b)+c = -1.0000 \cdot 10^{50}$ and $a+(b+c) = 0$ because $(a+b)$ equals zero, and $(b+c)$ rounds to c.

b) An example where $(a \cdot b) \cdot c \neq a \cdot (b \cdot c)$ is when $a \cdot b$ is large enough to compute to ∞ in computer arithmetic, but $b \cdot c$ does not, and $a \cdot (b \cdot c)$ does not. For example, if we have
a = 10^10
b = 10^305
c = 4.5*10^-200
In Matlab, $a \cdot b$ computes to infinity, and multiplying c afterwards does not change that. However, $b \cdot c$ computes to $4.5000e+105$, and multiplying a afterwards gives us $4.5000e+115$. Thus, $(a \cdot b) \cdot c = \infty$ and $a \cdot (b \cdot c) = 4.5000 \cdot 10^{115}$.

6. a) Between $|x_i - \widetilde{x}_i|$ (1) and $|x_i - \widehat{x}_i|$ (2), $|x_i - \widehat{x}_i|$ is more accurate. This is because in (1), rounding errors will accumulate with each iteration of i, where in (2), each value will be calculated afresh, and not have any accumulated rounding errors, only those from that iteration.

b) See spacing.m

7.

a)

$$f(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(\xi_1) \Longrightarrow$$

$$hf'(x) = f(x+h) - f(x) - \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(\xi_1)$$

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(\xi_2) \Longrightarrow$$

$$hf'(x) = -f(x-h) + f(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(\xi_2)$$

$$2hf'(x) = f(x+h) - f(x-h) - \frac{h^3}{6}f'''(\xi_1) - \frac{h^3}{6}f'''(\xi_2) \Longrightarrow$$

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{h^2}{12}[f'''(\xi_1) + f'''(\xi_2)]$$

Therefore, the truncation error for this approximation is $-\frac{h^2}{12}[f'''(\xi_1)+f'''(\xi_2)]$.

b) Given that the truncation error is $-\frac{h^2}{12}[f'''(\xi_1)+f'''(\xi_2)]$, and includes $\frac{h^2}{12}$, the error should be directly proportional to h. Minimizing h will minimize the error.

It may be possible to reduce the error to 0 with a non-zero h, given that the computer will have roundoff errors. I used this Matlab script

```
h = double(0);
a = double(10^-173);
h_sq = h^2;
while h_sq == 0
    h = h + a;
    h_sq = h^2;
end
fprintf('%.12e', h)
```

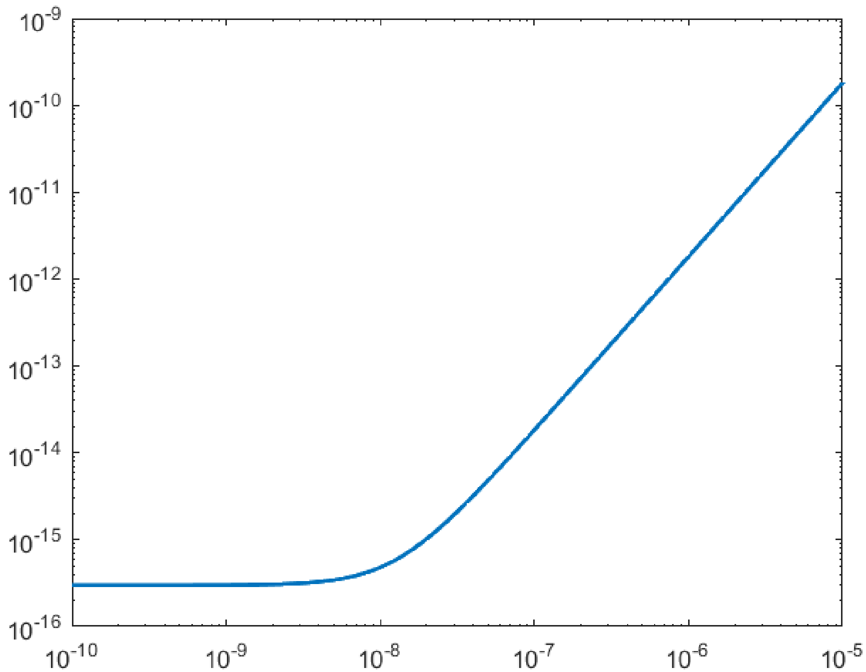
to determine that for all $|h| < 5.666945118888 \cdot 10^{-162}$, $\frac{h^2}{12}$ will compute to 0, thus the error should compute to zero when $|h| < 5.666945118888 \cdot 10^{-162}$.

c)

```
N = -10:0.01:-5;
h = 10.^N;

f_prime = @(x) (cos(x).*exp(cos(x))-(sin(x).^2).*exp(cos(x)));
f_approx = @(x, h) ((f(x+h)-f(x-h))./(2.*h));
error_f = @(h) (abs(f_prime(0) - f_approx(0, h)));

error = error_f(vpa(h));
loglog(h, error_f(vpa(h)), 'LineWidth', 1.5);
```



As we can see, I was correct in stating that h is directly proportional to the error. The graph follows a straight line until h reaches $\sim 10^{-8}$, at which point it levels off between 10^{-15} and 10^{-16}. This does show that my hypothesis about choosing an h small enough to zero out the error, because the machine error levels out between 10^{-15} and 10^{-16}, not at 0.

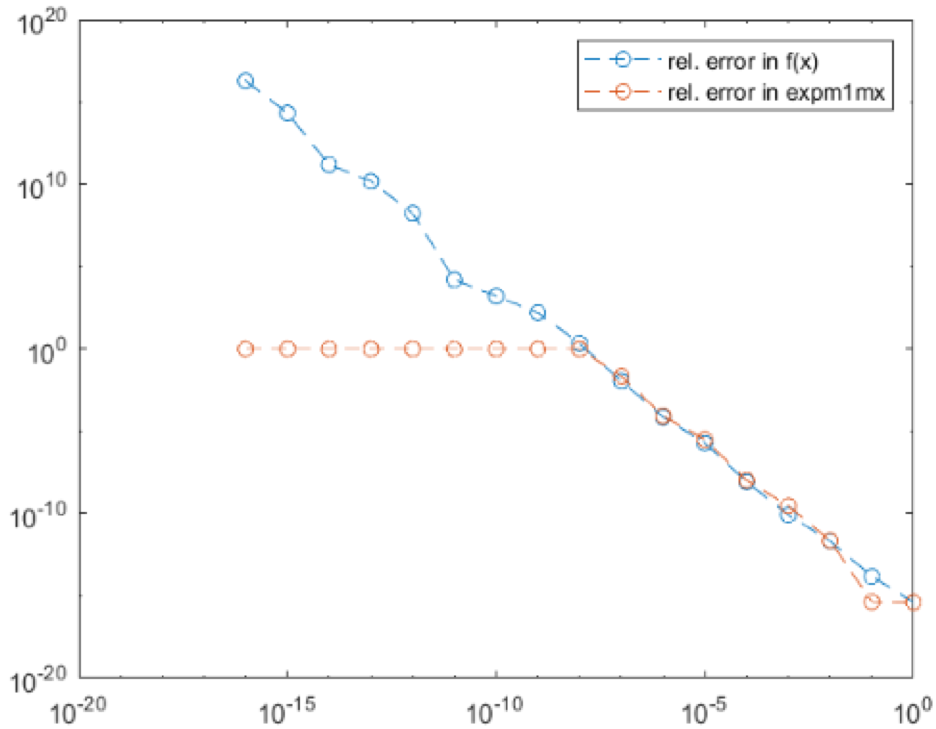
8.

a) The relative error when f(x) is evaluated with |x| can be quite large mainly due to the fact that the result will be small (~0.5 to 0), so any error will be relatively greater than with a large number. Eg. if the error is 0.05, the relative error for 0.5 is 10%, which is significant, whereas the relative error for 10 is 0.5%, which is not.

b) With my function

```
function y = expm1mx(x)
% Evaluates (exp(x) - 1 - x)/x2 accurately for |x| < 1.
    y = (exp(x) - (double(1) + x))./x.^2;
end
```

I get this plot



9.

The different results to the "same" equation - namely, g(x) having a different result to h(x) with x = 1e-10 - is due to rounding errors.

As we can see in this screenshot below, exp(x) is 1.0000, exp(x) - 1 is 1.0000e-10, and exp(x) - x is 1. When Matlab displays a number with no decimals - as it does with the 1 at the bottom of the screenshot - it is showing that it is exact, whereas the top 1 - displayed as 1.0000 - is not. Perhaps it is 1.0000000000000000002, but Matlab is only displaying the first four decimals of precision. The same it true for the 1.0000e-10 displayed on screen.

```
g = @(x) (exp(x)-1-x)./x.^2;
h = @(x) (exp(x)-x-1)./x.^2;
x = 1e-10;
disp(exp(x))
```

```
    1.0000
```

```
disp(exp(x)-1)
```

```
   1.0000e-10
```

```
disp(exp(x)-x)
```

```
     1
```

This means that when x is subtracted from $exp(x) - 1 = 1.0000e - 10$ and 1 is subtracted from $exp(x) - x = 1$, we get different results, because the first equation has a rounding error, and the second does not. The rounding error ends up being very significant, because although it is tiny, it is then divided by x^2 - and x equals 1e-10.

So why doesn't this happen with $x = 2^{-33}$? Well, in this case, x is initialized as a binary number, so there aren't any rounding errors - it's a binary number stored in binary, as opposed to a decimal number stored in binary. No rounding errors means it doesn't matter the order of operations, the result is the same both times.