Problem 1: Floating point representation and conversion. (20 pts)

According to the IEEE Standard for Binary Floating-Point Arithmetic, a 32-bit floating point number has the following format:

![Figure 1: The format of single precision floating point numbers](image)

In the regular cases that $0 < exponent < 255$, it represents the number

$$(-1)^S \times 2^{exponent-127} \times 1.F_1 F_2 \cdots F_{23}$$

1. What number does the following float represent?

![sign exponent (8 bits) fraction (23 bits) = ](image)

2. Fill the representation of $-0.3125$ in the following figure. Trailing zeros in the fraction part can be omitted.

![sign exponent (8 bits) fraction (23 bits) = -0.3125](image)

3. What’s the floating point representation of integer $2^{30}$? What’s the floating point representation of $2^{30} + 10$? If we do subtraction $(2^{30} + 10) - 2^{30}$ in floating point, what’s the result we get?
Problem 2: Propagation of roundoff error. (20 pts)

Recall that when a real number is represented in the single precision floating point format in Problem 1, there will be up to $2^{-24}$ round off error in the mantissa $1.F_1F_2\cdots F_{23}$.

1. Suppose you are computing the value $x = \sin(\pi - 0.01)$ using the code $x = \sin(pi - 0.01)$.
   
   (a) Give a bound for the error in the floating point representation of $\pi - 0.01$.
   
   (b) Give a bound for the absolute error in $x$ compared to $x$.
   
   (c) Give a bound for the relative error in $x$ compared to $x$.

2. Repeat the above 3 questions for $x = \sin(1000\pi - 0.01)$

Explain your reasoning.

The function $\sin()$ takes a single-precision argument, but the argument is computed in double precision and is only converted to single precision just before the function call.

It is OK to make approximations that change your answers by less than 5%. In particular, you may use the fact that $|\sin(n\pi - x)| \approx |x|$ when $|x|$ is small. Assume that the library function $\sin$ is correctly rounded, meaning that it always returns the floating point number nearest to the exact value for the given argument.
**Problem 3:** Comparison of root finding methods. (20 pts)

Fill in the following table.

<table>
<thead>
<tr>
<th>Method</th>
<th>#function evaluations per iteration&lt;sup&gt;a&lt;/sup&gt;</th>
<th>Convergence speed&lt;sup&gt;b&lt;/sup&gt;</th>
<th>Needs function derivative (Y/N)</th>
<th>Guarantee to converge&lt;sup&gt;c&lt;/sup&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bisection</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Newton</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Secant</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<sup>a</sup>Includes evaluations of the function and of the derivative.

<sup>b</sup>Write 1 for the fastest method among the three, 3 for the slowest, and 2 for the one in between.

<sup>c</sup>You may assume the function is smooth and a root has been bracketed if appropriate.

**Problem 4:** Vectorization. (20 pts)

Vectorize the following fragments of Matlab code, replacing them with equivalent compact and efficient code. Assume that \( x \) and \( y \) are column vectors of equal length.

1. \( z = \text{zeros} \left( \text{size}(x) \right); \)
   
   \[
   \text{for } i = 1: \text{length}(x) \\
   \quad z(i) = x(i)^2; \\
   \text{end}
   \]

2. \( z = 0; \)
   
   \[
   \text{for } i = 1: \text{length}(x) \\
   \quad z = z + x(i) \cdot y(i); \\
   \text{end}
   \]

3. \( z = 0; \)
   
   \[
   \text{for } i = 1: \text{length}(x) \\
   \quad \text{for } j = 1: \text{length}(x) \\
   \quad \quad z = z + A(i,j) \cdot x(i) \cdot x(j); \\
   \quad \text{end} \\
   \text{end}
   \]
   (For this one, \( A \) is square, with both dimensions equal to the length of \( x \).)

4. \( [m,n] = \text{size}(A); \)
   \[
   z = \text{zeros}(1,n) \\
   \text{for } j = 1:n \\
   \quad z(j) = \text{norm}(A(:,j)); \\
   \text{end}
   \]
**Problem 5:** Reverse engineering. (20 pts)

You are reading some code from another programmer and you find iterative numerical calculations with cryptic variable names and no comments in several places. Each of the following lines appears as the body of a while loop.

1. \( x = x - \tan(x); \)
2. \( x = x - \tan(x) + a / \cos(x); \)
3. \( t = x - (x - y) / (\exp(x) - \exp(y)) * (\exp(x) - a); y = x; x = t; \)
4. \( x = 0.5 * (x + a / x); \)
5. \( t = x - (x^2 - a) / (x + y); y = x; x = t; \)

The variable \( a \) is a constant value. From context you guess that these iterations are solving functions of one variable. For each case, answer these questions:

(a) What value does the iteration converge to? If there is more than one possible answer, any of them is fine.

(b) What algorithm is being used?

(c) What equation is being solved? Explain the code in terms of the standard equation for the algorithm.