Selected Assignment # 3 Solutions.

1.2 #29. [This is not something you do in real life, but might help your understanding of the error formula!] With the given values,

\[ E_5 = E_{n+1} = \frac{f^{n+1}(\xi)}{(n+1)!}(x-c)^{n+1} = \frac{\cos(\xi)}{5!} \left(\frac{\pi}{4}\right)^5. \]

Now, \( f(x) = \) [series \( k = 0 \) to \( k = n = 4 \)] + \( E_{n+1} \), so

\[ \frac{1}{\sqrt{2}} = \sin \frac{\pi}{4} = \frac{\pi}{4} - \left(\frac{\pi}{4}\right)^3 / 6 + E_5 \]

since we know the first four terms of the Taylor series: \( \sin x \approx x - x^3/3! \). These facts allow us to solve for \( \cos(\xi) \):

\[ \cos(\xi) = 5! 4^5 \left(\frac{1}{\sqrt{2}} - \frac{\pi}{4} + \left(\frac{\pi}{4}\right)^3 / 6\right) \approx 0.985438. \]

Since \( \xi \) should be in the interval \([0, x] = [0, \pi/4]\), this means \( \xi \approx \arccos(0.985438) = 0.170864. \)

2.1 #2. a. \((27.1)_{10} = (33.06314)_{8}\)

b. \((12.34)_{10} = (14.25605075\ldots)_{8}\)
c. \((3.14)_{10} = (3.10753412\ldots)_{8}\)

2.1 #10. The conversion procedure referred to in the text is, of course, just the procedure that groups in fours and gives hex digits:

\[ N = (111100101001111110)_{2} = (111100101001111110)_{2} = (3CA7E)_{16}. \]

How to justify? Start by writing out the meaning of binary digits:

\[ N = (d_n d_{n-1} \ldots d_2 d_1 d_0)_{2} = d_n 2^n + d_{n-1} 2^{n-1} + \ldots d_2 2^2 + d_1 2^1 + d_0 2^0 \]

where \( d_i \in \{0, 1\} \). If needed, pads by zero until there are a multiple of four digits: \( n+1 = 4k \) or \( n = 4k - 1 \). Then you group into fours:

\[ N = (d_{4k-1} 2^{4k-1} + \ldots + d_{4k-4} 2^{4k-4}) + \ldots + (d_3 2^3 + d_2 2^2 + d_1 2^1 + d_0 2^0). \]

Then recognize that

\[ 0 \leq a 2^3 + b 2^2 + c 2^1 + d 2^0 < 16 \]

if each of \( a, b, c, d \) is a binary digit. Thus we can replace each group of four with a hexadecimal digit.

2.1 CP #3. Here’s my code. I played around refining it more than you need to. Note that strings are arrays (row vectors):

```matlab
function [ostr,bstr]=convOctBin(n);
% CONVOCTBIN [ostr,bstr]=convOctBin(n)
% Converts integers to octal and binary strings.
% The binary string is grouped in threes.
% For example, " [oct bin]=convOctBin(100) " gives
% oct = 144 and bin = 1 100 100
% while " [oct bin]=convOctBin(-57382) " gives
% oct = -160046 and bin = -1 110 000 000 100 110.
%(Ed Bueler, 10/5/02)

octdig='01234567'; binoct='00000101001111011101111';
if n<0, sstr='-'; else, sstr=''; end
n=abs(n); k=floor(log(n)/log(8)); pow=round(8^k);
ostr=''; bstr='';
```
for j=k:-1:0
    dig=floor(n/pow); n=rem(n,pow); pow=round(pow/8);
    ostr=[ostr octdig(dig+1)];
    bstr=[bstr binoct(3*dig+1:3*dig+3) ' '];
end
for i=1:2  % strip zeros
    if bstr(1)=='0', bstr=bstr(2:length(bstr)); end
end
ostr=[sstr ostr]; bstr=[sstr bstr];  % attach sign

For example,
>>[oct bin]=convOctBin(100)
oct =
144
bin =
1 100 100

And: (10)\textsubscript{10} = (12)\textsubscript{8} = (1010)\textsubscript{2},
(−57382)\textsubscript{10} = (−160046)\textsubscript{8} = (−1 110 000 000 100 110)\textsubscript{2},
(138251)\textsubscript{10} = (416013)\textsubscript{8} = (100 001 110 000 001 011)\textsubscript{2}.

2.2 #2.  
\textbf{a.} The format is $\overline{s|c|f}$. For $0.5 \equiv 2^{-1}$, $c - 127 = -1$, that is $c = 126 = (1111\ 1110)\textsubscript{2}$.

Then $0.5 = \overline{0|0111\ 1110|000\ 000\ 000\ 000\ 000\ 000\ 000\ 000\ 000\ 000\ 000}$.
And $-0.5 = \overline{1|0111\ 1110|000\ 000\ 000\ 000\ 000\ 000\ 000\ 000\ 000\ 000\ 000}$.

\textbf{b.} Then $0.125 = \overline{0|0111\ 1100|000\ 000\ 000\ 000\ 000\ 000\ 000\ 000\ 000\ 000\ 000}$.
And $-0.125 = \overline{1|0111\ 1100|000\ 000\ 000\ 000\ 000\ 000\ 000\ 000\ 000\ 000\ 000}$.

\textbf{c.} Then $0.0625 = \overline{0|0111\ 1011|000\ 000\ 000\ 000\ 000\ 000\ 000\ 000\ 000\ 000\ 000}$.
And $-0.0625 = \overline{1|0111\ 1011|000\ 000\ 000\ 000\ 000\ 000\ 000\ 000\ 000\ 000\ 000}$.

\textit{SORRY to assign such boring examples.}

2.2 #36.  I think this is a common phenomenon. For instance, I ran

\texttt{count=0;}
\texttt{for j=1:100000}
\texttt{    a=rand(1); b=rand(1); c=rand(1);}
\texttt{    if ((a+b)+c)-(a+(b+c))\neq 0, count=count+1; end}
\texttt{end}
\texttt{count}

which compares $(a+b)+c$ to $a+(b+c)$ for a hundred thousand examples where $a, b, c$ are random (uniform on $[0, 1]$). The count is 25537, that is,

$$(a + b) + c \neq a + (b + c)$$

about one quarter of the time in this context.

2.2 CP #7.  [This is an interesting problem but one which is more important to \textit{explore} than to \textit{get the right answer}. I have graded it with this in mind.]

The first question (“\ldots \textit{what is the largest value of} \ s \ldots \?”) requires recognizing that in the given pseudocode, $s$ will only change if $1.0/x$ is big enough to make a difference. That is, the largest the sum
could be is the first value of \( s \) for which

\[
s = s + \frac{1}{x}
\]

exactly, in machine floating point.

This will happen when \( \frac{1}{x} < \epsilon s \). Now \( s \approx \gamma + \ln x \) for large \( x \)—that’s the point of Euler’s constant, really.

So I would like to solve the inequality

\[
\frac{1}{x} < \epsilon (\gamma + \ln x) \quad \text{or} \quad x(\gamma + \ln x) > \frac{1}{\epsilon},
\]

for \( x \), but I don’t know how to do that exactly.

[Exactness isn’t needed, really. I could do trial and error on powers of two, for instance, or I can proceed as follows:]

Of course, \( s \) is bigger than one. Thus if \( \frac{1}{x} < \epsilon \) then the sum will not increase. Thus in the case of single precision (MARCS-32 or IEEE single), the sum \( s \) will not increase once

\[
\frac{1}{x} < \epsilon = 2^{-23}
\]

which is to say \( x > 2^{23} \). Thus the sum will certainly stop increasing after \( 2^{23} \approx 8 \times 10^6 \) steps in single precision. (One can actually test this in C or FORTRAN, but not in Matlab which is always in double precision.)

In double precision the sum will definitely not increase once \( x > \mathcal{O}(52) \approx 5 \times 10^{15} \), and that is a lot of steps.

I wrote the following program to compute estimates of \( \gamma \)—note \( x \) is just the index \( j \):

\[
\%
\text{badEuler} \quad \text{Approximates Euler’s constant in a dubious manner.}
\%
\]

\[
\text{format long, format compact} \quad \%
\text{cleans up appearance}
\]

\[
n = 5000
\]

\[
s = 1.0;
\]

\[
\text{for } j = 2:n
\]

\[
s = s + 1/j;
\]

\[
\quad \text{if } \text{rem}(j,100)==0, \text{disp}(s-log(j)), \text{end}
\]

\[
\text{end}
\]

\[
\%
\text{to compare last estimate to sum in reverse order:}
\%
\text{s}=1/n; \text{for } j=n-1:-1:1, \text{s}=s+1/j; \text{end, s-log(n)}
\]

It produced 50 numbers which slowly converged to \( \gamma \), but the last had only 3 digits correct:

\[
\text{badEuler}
\]

\[
n = 5000
\]

\[
0.58220733165153
\]

\[
0.57971358157341
\]

\[
\text{:}
\]

\[
0.57731982795127
\]

\[
0.57731770224705
\]

\[
0.57731566156817
\]

By the way, in double precision, even with \( n = 5,000,000 \), there was only a difference in the last two places if I reversed the order of the sum.
You may find the following useful: I have used the following commands in the solutions I have written already. I thought it might be a useful list. Note that “help disp” etc. will show how to use disp:

**Special character commands:**

```
;  
:   
.  
.*  
=  
==  
~=   
```

**General Matlab commands:**

clear
disp
error
inline
length
size

**Math commands:**

abs
fix
floor
log
min
rand
rem
round

**Plotting commands:**

grid (on/off)
hold (on/off)
legend
plot
semilogy
title

**Major algorithms:**

polyfit
polyval