Selected Assignment # 2 Solutions.

3.3 CP #16. Here's my version. There is more than needed because I also looked at how the error decreased at each step:

```
% False Position  (ELB 9/27/02)

%f=inline('x.^3-5*x+3','x');a=0;b=1;  % for #5
f=inline('x-cos(x)','x'); a=0; b=1;  % to solve x=cos(x)
N=20;
clear err; fa=f(a); fb=f(b);
if fa*fb>0, error('f(a), f(b) not opposite in sign'), end
for j=1:N
   [a b fa fb] % show bracket  and values at each stage
   err(j)=min(abs([fa fb]));
   c=(b*fa-a*fb)/(fa-fb);
   fc=f(c);
   if fc*fa>0, a=c; fa=fc; else, b=c; fb=fc; end
end;
[a b fa fb]
% plot error logarithmically
j=1:N;
semilogy(j,err,'o')
title('Value of min(|f(a)|,|f(b)|) in False Position.')
```

I ran the program on #5 (that is, on $x^3 - 5x + 3 = 0$) using $[-3, 3]$ as the initial bracket. I got $[a, b] = [-3.000000, -2.490863]$ after 20 steps, with $f(b) \approx 5 \times 10^{-11}$ which means $b = -2.490863$ is close to a root. Of course it is not the largest positive root because it isn't positive.

Then I used $[1, 3]$ as the starting bracket and got $[1.83410, 3.00000]$ with $f(a) \approx -7 \times 10^{-4}$ after 20 steps. With $[0, 1]$ as the starting bracket, I got $[0, .65662]$ with $f(b) \approx -6 \times 10^{-15}$ after 20 steps.

I think I have found all the roots, and 1.83410 is the approximate location of the largest. I compare to:

```
>>roots([1 0 -5 3])
ans =
 -2.49086361536104
 1.83424318431392
 0.65662043104711
```

Finally, starting with $[0, 1]$ I found $[.7390851332151606, .7390851332151607]$ as a bracket for the solution to $x = \cos(x)$ after 20 steps. So it seemed to work great on this example, but closer inspection showed that the interval shrank only "accidently" when one of the endpoints of the interval was essentially correct already.

The problem with the method is that the interval isn't actually shrinking, even though one end is close to a root. Drawing pictures suggests why, and the next problem (#17, not assigned!) suggests how it might be solved. Don't give up bisection, Newton, and secant for this method!

1.1 #2. We have

$$\left|\frac{0.6032 - x}{x}\right| = \frac{1}{1000},$$

or $|0.6032 - x| = \frac{x}{1000}$ since clearly x is positive. Either

$$0.6032 - x = \frac{x}{1000} \quad \text{so } x = \frac{1000}{1001}\,0.6032 = \frac{6032}{10010}$$

or

$$x - 0.6032 = \frac{x}{1000} \quad \text{so } x = \frac{1000}{999}\,0.6032 = \frac{6032}{9990}.$$

1.1 #8ab. **a.** $p(x) = x^{32} = x^{(2\cdot2\cdot2\cdot2\cdot2)} = (((((x^2)^2)^2)^2)^2)^2$, which requires only 5 multiplications.

b. Compute $x - 1$ first, which requires one subtraction. Let $q(x) = ((x-1)^2)^2$ (two multiplications). Then

$$p(x) = (x-1)q(x)\,(3 + 7q(x))\,,$$

which requires 3 more multiplications and one addition for a total of 5 multiplications and 2 additions/subtractions.

1.1 #9. Here is running *Matlab*, which may always be substituted for a pseudocode:

```
% EXPPOLY for sect 1.1 #9

x=-1.3456 %sample value
v=exp(x);
y=11 + v*(9 + v*(7 + v*5))
```

It requires one evaluation of the exponential, 3 multiplications and 3 additions.

1.1 #10. First make sure you know what you are calculating, by writing it out:

$$z = \frac{a_1}{b_1} + \frac{a_1a_2}{b_2} + \cdots + \frac{a_1a_2\ldots a_n}{b_n}.$$

Here's code that runs:

```
% SUMPROD10 for sect 1.1 #10
n=5 % sample values for n,a,b
a=[2 3 4 1 2]
b=[1 2 3 3 2]
% result should be: z=2+3+8+8+24=45 (by hand, to check)
p(1)=a(1);
for j=2:n, p(j)=p(j-1)*a(j); end
z=p(1)/b(1);
for j=2:n, z=z+p(j)/b(j); end
z % display
```

It does $n-1$ multiplications and n divisions, but is wasteful by storing a new array `p`. Can you do better? In fact, the program uses subscripted variables when it does not need to.

1.1 #13. I wrote out what z was after each of the first few iterations of the **for** loop. I saw the (temporary) values for z:

$$z = b_n + 1, \quad b_nb_{n-1} + b_{n-1} + 1, \quad b_nb_{n-1}b_{n-2} + b_{n-1}b_{n-2} + b_{n-2} + 1, \ldots$$

This is a sum of products, and eventually gives $z = 1 + b_2 + b_2b_3 + \cdots + b_2b_3\ldots b_n$ or

$$z = 1 + \sum_{j=2}^{n} \prod_{k=2}^{j} b_k.$$

1.1 CP #4. I entered at the command line:

```
>>n=8.^(1:20)
>>y=(1+1./n).^n
```

(Do you see how it works, and how it takes advantage of the use of the colon notation and the vectorized nature of *Matlab*?) It produced:

```
y =
 Columns 1 through 3
```

```
   2.56578451395035 2.6973449525651 2.71563200016899
   Columns 4 through 6
   2.71795008118967 2.71824035193029 2.71827664376605
⋮
   Columns 16 through 18
   2.71828182845904 2.71828182845904 1
   Columns 19 through 20
   1                1
```

Compare to `exp(1.0)=2.71828182845905`. Thus the $n = 8^{17}$ result, and previous, is great, but the $n = 8^{18}, 8^{19}, 8^{20}$ estimates are very poor! Why?

Note

$$\frac{1}{8^{18}} \approx 5.55 \times 10^{-17}$$

while `eps = 2.2204e-016` is *Matlab*'s `EPSILON`. This explain the problem. In fact, the machine thinks $1 + \frac{1}{8^{18}} = 1$ and raises it to a large power to get 1 as an estimate for e.

This problem shows you what is meant by the definition of `EPSILON(X)` in section 2.2.

1.1 CP #7. I offer the following two programs. The first uses traditional (non*Matlab*) programming style:

```
% MEANVARSTD for sect 1.1 CP #7

a=[1 2 3 4]; n=4;  % some test data for which I can check the results by hand
%a=100*rand(1,20); n=20;  % some more test data

m=a(1);  for k=2:n, m=m+a(k); end
m=m/n
v=(a(1)-m)^2;  for k=2:n, v=v+(a(k)-m)^2; end
v=v/(n-1)
sigma=sqrt(v)
```

The second uses *Matlab* relatively well, and is a function:

```
function [m, v, sigma]=mvs(a);
% MVS Function for sect 1.1 #7.

n=size(a,2);
m=sum(a)/n;
v=sum((a-m).^2)/(n-1);
sigma=sqrt(v);
```

You run this second program at the command line by

```
>>[m v sig]=mvs([1 2 3 4])
```

Now, you can test the above with builtin *Matlab* functions which do the same thing:

```
>>a=[1 2 3 4], mean(a), var(a), std(a)
```

1.2 #5. I find the series by noting $\cosh x$ is the even part of the exponential function:

$$\cosh x = \frac{e^x + e^{-x}}{2} = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \frac{x^6}{6!} + \dots$$

Thus

$$\cosh(0.7) \approx 1 + \frac{(0.7)^2}{2} + \frac{(0.7)^4}{24} + \frac{(0.7)^6}{720} = 1.25516757.$$

Compare $\cosh(0.7) = 1.25516901$ from *Matlab*, so the above has absolute error of 1.4×10^{-6}, and a relative error of about the same (since $\cosh(0.7)$ is actually pretty close to one).

1.2 #8. I used Taylor's theorem as stated on page 27. We want e^1 so $x = 1$, and we are using the series on page 22 for e^x, so $c = 1$. To answer the question, all we care about is to determine n so that the error term E_{n+1} is less than 5×10^{-16}. (This gives correct rounding to 15 digits past the decimal point in all cases.) Now, $f^{(n+1)}(\xi) = e^\xi$. Thus

$$E_{n+1} = \frac{e^\xi}{(n+1)!}1^{n+1} = \frac{e^\xi}{(n+1)!}.$$

This needs to be less than 5×10^{-16} in absolute value, as stated. The largest ξ could be, given x and c here, is $\xi = 1$. Note we know $e < 3$. (*Exercise*: Prove this without summing infinite series or knowing e exactly.) Thus we need n so that

$$|E_{n+1}| = \left|\frac{e^\xi}{(n+1)!}\right| \leq \frac{3}{(n+1)!} < 5 \times 10^{-16}.$$

I don't care to invert a factorial, so I do trial and error to get n large enough:

```
>>n=1:20; 3./cumprod(n+1)
```

⋮

```
 Columns 16 through 18
 8.43437176303656e-015 4.68576209057587e-016 2.4661905739873e-017
```

⋮

This tells me that if $n = 17$ then the error is at most 4.69×10^{-16}, which is small enough. Thus $n = 17$, or eighteen terms (why?) is enough.

1.2 #28. This involves the $n = 3$ case of Taylor's theorem for $\cos x$, because the question is how accurate is $1 + 0 - \frac{x^2}{2} + 0$ as an approximation to $\cos x$. But $E_4 = E_{n+1} = \frac{\cos\xi}{4!}x^4$ since the fourth derivative of cos is cos. And

$$|E_4| \leq \frac{1}{24}|x|^4 \leq \frac{1}{384} \approx 0.0026$$

for $|x| < \frac{1}{2}$.

1.2 CP #12. The program I get is

```
% PISINEST for sect 1.2 CP #12

st=1 % sin(theta_2)=1
p=2*st % p_2=2 sin(theta_2)
for j=3:20
   temp=sqrt(1-st^2);
   st=st/(sqrt(2*(1+temp)))
   p=(2^(j-1)) * st
end
```

which gives $p_{20} = 3.14159265357099$. That compares with `4.0*atan(1.0) = pi = 3.14159265358979`, with absolute error about 2×10^{-11}.

How to get there? By looking at each sector I note

$$A_{\text{sector}} = 2 \cdot (1/2)\sin(\theta_n/2)\cos(\theta_n/2) = (1/2)\sin\theta_n.$$

Next I note $\sin(2\theta_n) = \sin(\theta_{n-1})$ (why?). But $\sin(2\theta_n) = 2\sin(\theta_n)\cos(\theta_n)$, so I want to solve

$$2\sin\theta_n\sqrt{1-\sin^2\theta_n} = \sin\theta_{n-1}$$

for $\sin\theta_n$. That is, solve $2x\sqrt{1-x^2} = y$ for x. This is quadratic in x^2, actually, and we want the positive x satisfying $x^2 = (1/2)\left(1 - \sqrt{1-y^2}\right)$ because the other is close to one. Rationalizing the numerator and using $x = \sin\theta_n$, etc. gives

$$\sin\theta_n = \frac{\sin\theta_{n-1}}{\sqrt{2[1+\sqrt{1-\sin^2\theta_{n-1}}]}}.$$