MA2507 Computing Mathematics Laboratory: Week 4

1. The "while" loop. When we use a "for" loop to evaluate some formula repeatedly, we need to know the number of iterations. Often, the number of iteration is unknown, then the "while" loop becomes useful. Let us consider Newton's method for solving a nonlinear equation $f(x) = 0$. Starting from an initial guess x_1, Newton's method gives us a sequence x_2, x_3, ..., by

$$x_{j+1} = x_j - \frac{f(x_j)}{f'(x_j)}.$$

If the equation $f(x) = 0$ has a solution x_* and if x_1 is sufficiently close to x_*, then under some suitable conditions, $\lim_{j\to\infty} x_j = x_*$. In reality, we only calculate a finite number of iterations, and use the last iteration as the approximate solution. Typically, we use a small number `tiny` to control the "while" loop, and to finish the loop when some kind of "relative error" is smaller than `tiny`. For Newton's method, typically, we can use the ratio $(x_{j+1} - x_j)/x_{j+1}$ as the "relative error". Now, for $f(x) = x^2 - a$, we use Newton's method to find the square root of a. The following script file calculates $\sqrt{3}$.

```
% solve f(x) = x^2 -a by Newton's method.
a= 3;
x = 1.5;          % initial guess
tiny = 1.0e-10;   % small number
rat = 1;          % any number larger than tiny
while rat > tiny
    dx = (x^2-a)/(2*x);   % f(x_j)/f'(x_j)
    x = x - dx
    rat = abs(dx/x);
end
```

The program gives the following

```
x = 1.750000000000000
x = 1.732142857142857
x = 1.732050810014728
x = 1.732050807568877
x = 1.732050807568877
```

It agrees with the "exact" value of $\sqrt{3}$ very well.

2. Catastrophic cancellation. Let us write a program for

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + ...$$

If we write the program based on the "for" loop, we need to specify an integer for the maximum number of terms. Meanwhile, we can define a variable $y = x^n/n!$, and update y in each iteration. This leads to

```
nmax=50;    % truncate the Taylor series to power nmax.
s=1;        % initialize s, the final s approximates exp(x)
y=1;
```

```
for n=1:nmax
   y=y*x/n;
   s=s+y;
end
```

This is not ideal, since when x is large (e.g. $x = 50$), the approximation is not at all accurate. For small x, it is possible that we have already kept too many terms. To improve this, we can use a "while" loop that compares $|y/s|$ with a given small number. But in the "while" loop, we no longer have an integer index. Therefore, we need to initialize the index and manually increase the index in the loop.

```
s=1;
y=1;
n=0;      % integer index
rat=1;
while rat > 1.0e-14
    n=n+1;    % increase the index by 1
    y=y*x/n;
    s=s+y;
    rat=abs(y/s);
end
s
```

Notice that we have used a small number 10^{-14} to control the "while" loop. Using `format long` and for $x = 2$, the above program gives $e^x \approx 7.389056098930645$, but the MATLAB internal program gives the slightly more accurate value 7.389056098930650.

However, the program has trouble if x is negative and $|x|$ is large. For $x = -21$, the above program gives $s = -3.164859560770680 \times 10^{-9}$. For $x = -50$, the program gives $s = 1.107293338289196 \times 10^4$. This is very bad, since $e^{-21} \approx 7.582560427911907 \times 10^{-10}$ and $e^{-50} \approx 1.928749847963918 \times 10^{-22}$. To understand that, we need to think about the magnitude of y for those negative x with large absolute values. For n near $|x|$, $y = x^n/n!$ has very large absolute value, but we can only store y for about 16 digits. The truncation error (caused by only keeping 16 digits of y) could be much larger than the exact value of e^x.

3. The "if ... else ... end" statement. If we divide the unit square $[0, 1] \times [0, 1]$ to $m \times m$ small squares (of equal size) and drop m^2 random points on the unit square, how many small squares (call them boxes) will be hit by one or more of these points on average? This is a problem in probability theory. We will do a simulation here. We will use an $m \times m$ matrix B to keep track of the boxes. The matrix B is initialized as a zero matrix. If the (i, j) box is hit by a point, then we turn the (i, j) entry of B to 1. For a point (x, y), we need to figure out which box it belongs to.

```
m = 100;
B = zeros(m,m);
numbox = 0;
for k = 1:m^2
    x = rand;             % get a random point (x,y)
    y = rand;
    ix = ceil(m*x);       % (x,y) goes to box (ix,iy)
```

```
    iy = ceil(m*y);      % ceil(z) is the integer >or= z.
    if B(ix,iy) == 0
        B(ix,iy) = 1;
        numbox = numbox+1;
    end
end
numbox
```

The typical answer is around 6300.

4. Hénon map: Staring from a point (x_1, y_1), we can generate an sequence in the plane by

$$x_{j+1} = a - x_j^2 + by_j, \quad y_{j+1} = x_j, \quad j = 1, 2, ...$$

where a and b are constants. For some a and b, the sequence $\{(x_j, y_j)\}$ can have very complicated behavior. The following program shows 10^7 points for $a = 1.28$ and $b = 0.3$.

```
N = 10^7;
x(1) = rand;
y(1) = rand;
for j=1:N-1
    x(j+1) = 1.28 - (x(j))^2 + 0.3*y(j);
    y(j+1) = x(j);
end
plot(x,y,'b.','MarkerSize',1)
axis equal
axis([-2 2 -2 2])
```

The above gives the first plot below. The others are zoom-in plots.