第一章作业

问题叙述：使用$\cos x$ 的迈克劳林级数展开式，从$\cos x = 1$开始，然后每次加入一项来分别计算$\cos(\pi/3)$。在每加入一个新项后，分别计算真百分比相对误差和近似百分比相对误差（真值为 0.5）。直到近似误差估计值的绝对值小于与两位有效数字一致的误差准则时停止计算。

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \cdots \tag{1}$$

问题分析：

虽然(1)的计算是能够收敛到真值的，但在计算前不能得到真值，故用近似百分比误差估计值来衡量计算值与真值的接近程度，在第一次计算时，可以初始化为$-\infty$。

$$\varepsilon_a = \frac{\text{当前近似值} - \text{前一近似值}}{\text{当前近似值}} \times 100\%$$

事实上，真实的误差应当由真百分比相对误差来描述。

$$\varepsilon_a = \frac{\text{真值} - \text{近似值}}{\text{真值}} \times 100\%$$

由于误差准则与两位有效数字一致，则可以计算误差容限如下：

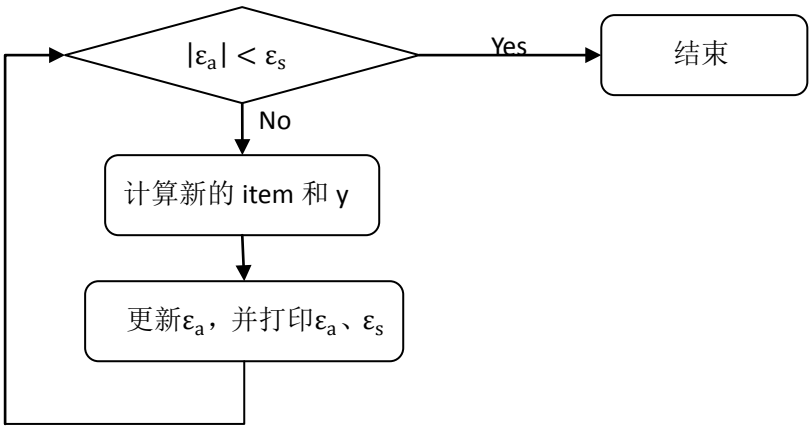
$$\varepsilon_s = (0.5 \times 10^{2-2})\% = 0.5\%$$

计算的终止条件是：

$$|\varepsilon_a| < \varepsilon_s$$

算法设计：

由于每次加入 y 的新项$item_i$ 即 $(-1)^i \frac{x^{2i}}{(2i)!}$具有递推关系，故按照以下步骤设计算法：



图表 1 基本算法流程

以上基础算法简单明了，时间复杂度显然为 o(n)

MATLAB 程序：

在 MATLAB2008a 中用脚本程序实现下面算法。

```
% This program is developed to calculate the value of y=cos(x) with the
% cos(x) series. Especially, an x=pi/3 is demonstrated as an example,
% with the precision in constraint of the tolerant error named e_s.
clear;
x = pi/3;
y = 1;      % Initialize the result of cos(x)
ys = y;     % Store the y series.
e_s = 5e-3; % The tolerant error can guarantee 2 significant digit.
e_a = inf;  % e_a stores estimating error for each step in the loop, in
            % percentage.
e   = 1;    % e   stores the actual error between the calculated one and
            % cos(pi/3) in percentage.
i = 1;      % i stands for a loop mark
item = 1;   % item indicates each new term added to the result,
            % (-1)^i*x^(2i)/(2i)!
true = 0.5; % take x=pi/3,cos(pi/3)=0.5 as an example

while  abs(e_a) >= e_s
   item = (-1) * item * x*x / (2*i-1)/2/i;
   y = y + item;
   e_a = item/y;    %update the error estimation
   e = (true-y)/true;
   fprintf('y=%e\t  eror_a=%e\terror=%e\n\n', y, e_a, e); %print
   i = i+1;
end
```

通过执行上述程序可以得到以下数据表格

表格 1 误差统计表

序号	当前计算值 y	近似百分比相对误差 ε_a	真百分比相对误差 ε_s
1	1	Inf	1
2	0.451689	-121.3914%	9.6623%
3	0.501796	9.9856%	-0.3592%
4	0.499965	-0.3664%	0.0071%

计算结果显示近似误差在刚达到 0.5%以下时，实际误差却已经远小于 0.5%。通过改变ε_s的大小理应可以调节结果的精度。

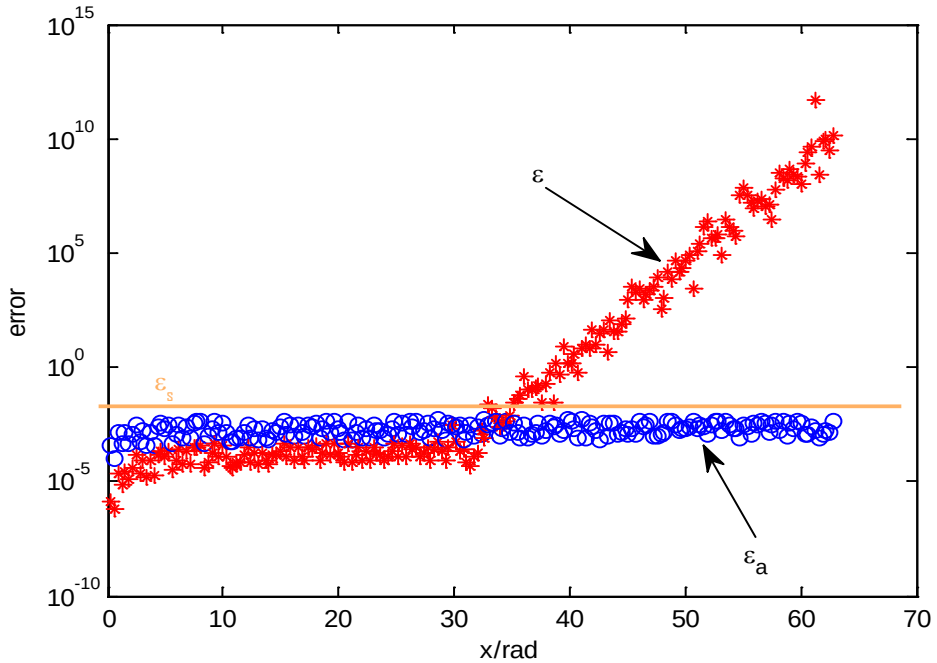
对计算结果的进一步分析：

$\cos x$ 的迈克劳林级数展开式显然是一个交错级数，理应会产生一定的拖尾效应，但是当 x=pi/3 时，$\frac{x^{2i}}{(2i)!}$ 一直保持递减，当 x 增大到使得$\frac{x^{2i}}{(2i)!}$ 在$i < i_0$的范围内保递增时，就会显现出明显的拖尾效益，为了图示这种拖尾效应，另 x 在[0,20*pi]上取值，用与前面相同的求

解程序运行求的计算结果。

```
% Analyse the error in variety of  x
clear;
y = 1;      % Initialize the result of cos(x)
e_s = 5e-3; % The tolerant error can guarantee 2 significant digit.
e   = 1;    % e stores the actual error between the caculated one and
            %cos(pi/3) in percentage.
for x=0:pi*99/100/10:pi*20
    true = cos(x);
    item = 1;
    i = 1;
    y = 1;
    e_a = inf;
while  abs(e_a) >= e_s %abs(item)
    item = (-1) * item * x*x / (2*i)/(2*i-1);
    y = y + item;
    e_a = item/y;
    e = (true-y)/true;
    i = i+1;
end
semilogy(x,abs(e),'r*');  % plot both the e_a and error in log-y-axis
hold on;
semilogy(x,abs(e_a),'bo');
end
```

得到以下数据图：



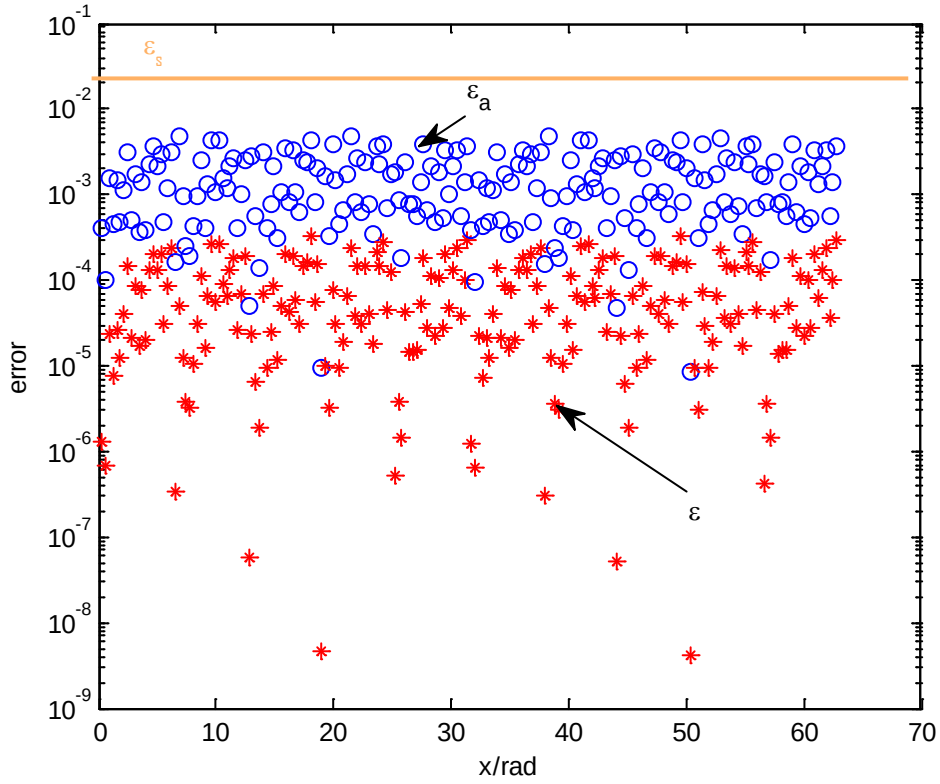
图表 2 不同 x 值时误差的比较

在 x=10*pi 左右时，开始显现出“拖尾效应”，开始迭代时每一加入项$\frac{x^{2i}}{(2i)!}$大大超过了前一次计算值 y 本身，使得“大数”吃掉了“小数”。虽然最后经过多步计算 $|\varepsilon_a| < \varepsilon_s$ 也已经满足，但是计算值已经积累了大量的累积误差。

为了解决计算精度的问题，虽然可以采用提高计算有效位数的办法，但却会耗费存储资源；如果用保留每一项级数，在每一次处理时采用从小到大的加法策略，亦可以解决拖尾效益，本人经过在 MATLAB 命令行中试验确实可行，但此方法却既耗费额外的时间亦耗费空间对于本问题并不实用。

如果利用 cos(x)的周期性就可以完美解决该问题，即$cos(x) = cos(x'), x' = x + 2k\pi, x' \in [0, \pi]$。

此时得到的误差分析图可与图 2 进行比较，可以发现效果显著变好。



图表 3 采用“周期性算法“不同 x 值时误差的比较

小结：

通过交错级数的方法可以较快计算 cos(x)，本文计算 cos(pi/3)只用了 4 次迭代事实上获得了 4 位有效数字（四舍五入后）的数值计算值。但是由于交错级数计算存在这拖尾效应，本文通过利用 cos(x)的周期性，完美地解决了该问题。从而，只要用户输入需要的精度限制 ε_s，本程序可以在$(-\infty, +\infty)$上得到满足要求的余弦值。

最后把最终的程序附在下面。

```
% This program is developed to caculate the value of y=cos(x) with the
% cos(x) series. Especially, an x=pi/3 is demonstrated as an example,
% with the precision in constraint of the tolerant error named e_s.
% This version can calculate cos(x) for ANY x in real number domain.
```

```
clear;
x = pi/3;
y = 1; % Initialize the result of cos(x)
e_s = 5e-3; % The tolerant error can guarantee 2 significant digit.
e_a = inf; % e_a stores estimating error for each step in the loop, in
percentage.
e   = 1; % e   stores the actual error between the caculated one and
cos(pi/3) in percentage.

i = 1; % i stands for a loop mark
item = 1; % item indicates each new term added to the result,
(-1)^i*x^(2i)/(2i)!
true = cos(x);
while  abs(e_a) >= e_s
    x1 = x - 2*pi*floor(x/pi/2);   %modify x
    item = (-1) * item * x1*x1 / (2*i-1)/2/i;
    y = y + item;
    e_a = item/y;    %update the error estimation
    e = (true-y)/true;
    fprintf('y=%e\t  eror_a=%e\terror=%e\n', y, e_a, e); %print
    i = i+1;
end
fprintf('*****\n');
```

对上述程序还可以封装在函数中利于实际使用，这里不再赘述。