1 问题1

正弦函数的无穷级数展开为$sinx = \Sigma_{i=0}^{\infty}\frac{(-1)^i x^{2i+1}}{(2i+1)!}$
（1）分别以单精度和双精度数据类型，计算x=0.5时近似值，要求计算结果具有4 位有效数字；
（2）如果采用单精度数据类型要求计算结果达到机器精度，此时结果如何?(测试机器精度:满足$1+\varepsilon>1$的最小浮点数)

1.1 解题思路

题目中给出了sinx的泰勒展开，由此可以用一个循环来将级数前若干项加起来来逼近准确值，同时判断新加上的一项是否达到精度要求，如果达到直接跳出循环，并且将最终值保留四位有效数字。
同时此级数虽然为符号交替类型的级数，但是随着i的增大，每一项的绝对值都是远远小于上一项的，因此我们可以猜测此级数中并不存在某一项的值大于和值本身的拖尾效应，至于猜测是否成立我们可以在程序中解答。

1.2 代码实现

```
x=0.5;
x_single=single(x);
x_double=double(x);
eps_four=1e-5;
eps_single=eps("single");
eps_double=eps("double");
```

首先定义好初始值，x_single与x_double分别表示单、双精度下数据类型的x。而对于精度，由于题目中要求四位有效数字，而sinx函数本身值不会大于一，因此10^{-5}作为误差限已经足够，为了更高的精度要求，我们也可以以其各自的机器精度作为误差限。两种数据类型的机器精度的求法在上机课作业中已经做过，我们也可以用其自带的eps值分别得到。

```
function sum=sin_tylor(x,eps)
    i_value=[];
    term_value=[];
    sum_value=[];
    sum=0;
    term=0;
    i=0;
    while 1
        term=((-1)^i)*(x^(2*i+1))/factorial(2*i+1);
        sum=sum+term;
        i_value=[i_value;i];
        term_value=[term_value;term];
        sum_value=[sum_value;sum];
        i=i+1;
```

```
        if abs(term)<eps
            break;
        end
    end
    T = table(i_value,term_value,sum_value,'VariableNames',
{'Index','Term','Sum'});
    disp(T);
end
```

然后定义求此级数的函数。为了方便展示函数循环的具体过程，定义了三个列表分别记录每一次的i，term(当前项)，sum(当前总和)并在最后汇总成表格输出。函数的输入为x值，eps(跳出循环的临界误差限，精度要求)，具体实现方法与数学公式同理。

```
y_double_four=sin_tylor(x_double,eps_four);
y_single_four=sin_tylor(x_single,eps_four);
y_double=sin_tylor(x_double,eps_double);
y_single=sin_tylor(x_single,eps_single);
fprintf("%.16f\n",y_double_four);
fprintf("%.16f\n",y_single_four);
fprintf("%.16f\n",y_double);
fprintf("%.16f\n",y_single);
```

自上而下依次为以10^{-5}为精度的双、单精度数据类型的值，以双精度的机器精度为精度的双精度数据类型值，以单精度的机器精度为精度的单精度数据类型值。此时获得的值还未保留有效数字，但是可以根据这些值来探究单双精度数据类型以及误差限选择对值准确性的影响。

```
y_double_four=vpa(y_double_four,4);
y_single_four=vpa(y_single_four,4);
y_double=vpa(y_double,4);
y_single=vpa(y_single,4);

disp(y_double_four);
disp(y_single_four);
disp(y_double);
disp(y_single);
```

保留4位有效数字并输出最终结果。

```
y_single_double=sin_tylor(x_single,eps_double);
fprintf("%.16f\n",y_single_double);
y_double_single=sin_tylor(x_double,eps_single);
fprintf("%.16f\n",y_double_single);
```

第二问中，问如果以单精度数据类型来达到双精度的机器精度，结果是什么样的，这段代码可以实现。也可以实现以双精度数据类型来达到单精度的机器精度。

1.3 结果分析

由计算器计算得到的 $sin0.5 = 0.479425538604203$

第一问中得到的四个值分别为

```
0.4794255332341270
0.4794255197048187
0.4794255386042030
0.4794255197048187
```

自上而下依次为以10^{-5}为精度的双、单精度数据类型的值，以双精度的机器精度为精度的双精度数据类型值，以单精度的机器精度为精度的单精度数据类型值。
由结果可得双精度数据的准确度大于单精度，而数据类型相同时，双精度受误差限的影响较大，而单精度受误差限影响不明显。

Index	Term	Sum
0	0.5	0.5
1	-0.020833	0.47917
2	0.00026042	0.47943
3	-1.5501e-06	0.47943

Index	Term	Sum
0	0.5	0.5
1	-0.020833	0.47917
2	0.00026042	0.47943
3	-1.5501e-06	0.47943
4	5.3823e-09	0.47943
5	-1.2232e-11	0.47943
6	1.9603e-14	0.47943
7	-2.3337e-17	0.47943

Index	Term	Sum
0	0.5	0.5
1	-0.020833	0.47917
2	0.00026042	0.47943
3	-1.5501e-06	0.47943
4	5.3823e-09	0.47943

四个值的循环过程如图所示。由图得和值远远大于新项值，不会出现拖尾效应造成的舍入误差。

四个值均保留四位有效数字后均为0.4794 。

```
0.4794

0.4794

0.4794

0.4794
```

虽然单双精度之间有精度差别，但是在四位有效数字的前提下，这些差别可以忽略，故sin0.5近似值为0.4794

第二问中的两个值为

```
0.4794255386164159
0.4794255386164159
```

与前面得到的值以及真值做对比。发现无论是单精度数据配双精度限制还是双精度数据配单精度限制，都不如纯双精度类型的准确度，但是高于纯单精度类型的准确度。

问题2

自行给定满足题目要求的具体值，编程计算下列公式，并寻找合理的等价公式再次进行计算以提高计算结果的精度(分别用单精度和双精度测试)

(1) $\frac{1-cosx}{sinx}$（其中|x|充分小）

(2) $\int_N^{N+1}\frac{1}{x^2+1}dx$（其中正整数 N 充分大）

2.1.1 解题思路

用matlab自带的cosx与sinx函数分别计算出值，再代入。

可以用泰勒公式分别求出cosx与sinx的值，再代入。

注意到原方程可以用二倍角公式化简:$1-cosx=2sin^2\frac{x}{2}\quad sinx=2sin\frac{x}{2}cos\frac{x}{2}$，当x不等于0的时候，可以得出原式子即$tan\frac{x}{2}$

2.1.2 代码实现

第一种思路

```
function result=answer1(x)
    result=(1-cos(x))/sin(x);
end
```

定义了一个用matlab自带的函数来求此值的函数

第二种思路

```
function result=answer2(x)
    sum_sin=0;
    sum_cos=0;
    term=0;
    i=0;
    while 1
        term=((-1)^i)*(x^(2*i+1))/factorial(2*i+1);
        sum_sin=sum_sin+term;
        i=i+1;
        if abs(term)<eps
            break;
        end
    end
```

```
    fprintf("%.16f\n",sum_sin);
    i=0;
    term=0;
    while 1
        term=((-1)^i)*(x^(2*i))/factorial(2*i);
        sum_cos=sum_cos+term;
        i=i+1;
        if abs(term)<eps
            break;
        end
    end
    fprintf("%.16f\n",sum_cos);
    result=(1-sum_cos)/sum_sin;
end
```

其中eps为双精度的机器精度。可以得到sinx与cosx的较精确值。整个函数返回整个式子的值。

第三种思路

```
function result=answer3(x)
    result=tan(x/2);
end
```

直接调用matlab自带的tanx函数

也可以尝试对tanx进行泰勒展开，查资料有$tanx=\sum_{n=1}^{\infty}\frac{B_{2n}(-4)^n(1-4^n)}{(2n)!}x^{2n-1}$，其中Bn是伯努利数。但是伯努利数太难表达，难以找到通式，故只能选其前八项。

```
function result=answer4(x)
    result=x+(x^3)/3+(x^5)*2/15+(x^7)*7/315+(x^9)*62/2835+
(x^11)*1382/155925+(x^13)*21844/6081075+(x^15)*929569/638512875
end
```

虽然很丑陋，但是也能算。

用于计算的初始值

```
x1=-0.01;
x1_single=single(x1);
x1_double=double(x1);
x2=0.0001;
x2_single=single(x2);
x2_double=double(x2);
```

由题x绝对值充分小，但是不能等于0，如果等于0，结果会是NaN。故可取x为0.0001和-0.01来验证结果，最终结果保留16位小数。选这两个数是因为在后续实验中发现，单精度在计算0.0001时已经无法区分，结果全为0了，如果选择再小的数将没有意义了。

2.1.3 结果分析

由计算器计算得到的答案

$$f(-0.01) = -0.00500004166708333754964588888075$$

$$f(0.0001) = 0.00005000000004166666708333333375496$$

第一种思路

```
y1_ans1_single=answer1(x1_single);
y1_ans1_double=answer1(x1_double);
y2_ans1_single=answer1(x2_single);
y2_ans1_double=answer1(x2_double);
fprintf("%.16f\n",y1_ans1_single);
fprintf("%.16f\n",y1_ans1_double);
fprintf("%.16f\n",y2_ans1_single);
fprintf("%.16f\n",y2_ans1_double);%后续所有代入值的代码都与之类似，均按照自上而下单精度-0.01、双-0.01、单0.0001、双0.0001的顺序
```

结果

```
-0.0050009130500257
-0.0050000416670848
0.0000000000000000
0.0000499999997795
```

与真值比较，在-0.01上单精度明显不如双精度，在0.0001上单精度直接无法表达，双精度两者也都存在明显的精度问题。

第二种思路

结果

```
-0.0050009130500257
-0.0050000416670848
0.0000000000000000
0.0000499999997795
```

发现和第一种一样，精度并未得到改善。

第三种思路

用自带的tanx

结果

```
-0.0050000413320959
-0.0050000416670833
0.0000499999987369
0.0000500000000417
```

发现不仅单精度的值准确度得到了改善，甚至0.0001的结果都可以表达了；双精度的值准确度也得到了提高，几乎与真值相同

用泰勒展开（注意输入值时要除以二）

结果

```
-0.0050000413320959
-0.0050000416670833
0.0000499999987369
0.0000500000000417
```

与用自带的tanx相同，matlab自带的sinx、cosx、tanx应该本质上都是用泰勒展开逼近的。

将保留位数拓展到20位

```
-0.00500004133209586143493 7
-0.00500004166708333756125 0
0.00004999999873689375817 8
0.00005000000004166671155
```

```
-0.00500004133209586143493 7
-0.00500004166708333495916 5
0.00004999999873689375817 8
0.00005000000004166671155
```

0.0001的几乎未受影响。-0.01的明显tanx更胜一筹，而tanx也出现了略微偏差，应该是超出了机器精度。如果能实现tanx的完全泰勒展开，应该精确度更高。

由此我们可以得出本题用二倍角公式转换为$tan\frac{x}{2}$再用$tanx$公式求解的方法结果精度最高。因为如果不化简，不仅有两个因素sinx与cosx产生了误差，而且涉及到了除法，会导致误差的传递与累积，因此式子越简单，能够产生误差的因素越少，最终精度越高。

2.2.1 解题思路

用matlab自带的积分函数来积分

将$\frac{1}{1+x^2}$积出来，为arctan(x)，再求

2.2.2 代码实现

第一种思路

```
syms x;
f=1/(1+x^2);
result=int(f,x,N,N+1);
```

f是被积函数，N是下限，N+1是上限。N是充分大的正整数。

第二种思路

```
function result=answer(N)
    result=atan(N+1)-atan(N);
end
```

泰勒展开$arctanx = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} x^{2n+1}$在此处并不能使用，因为此展开限定x范围为[-1,1]，超出这个范围都发散。

确定初始值

```
N1=100;
N2=10000;
N1_single=single(N1);
N1_double=double(N1);
N2_single=single(N2);
N2_double=double(N2);
```

保留24位小数，因为如果保留16位结果没什么区别。

2.2.3 结果分析

计算器得结果

$$f(100) = 0.000099000098676665031700049094179481$$

$$f(10000) = 0.000000009999000000009998667666666650003$$

第一种结果

```
N1_ans1_single=int(f,x,N1_single,N1_single+1);
N1_ans1_double=int(f,x,N1_double,N1_double+1);
N2_ans1_single=int(f,x,N2_single,N2_single+1);
N2_ans1_double=int(f,x,N2_double,N2_double+1);
fprintf("%.24f\n",N1_ans1_single);
fprintf("%.24f\n",N1_ans1_double);
fprintf("%.24f\n",N2_ans1_single);
fprintf("%.24f\n",N2_ans1_double);%按照单100、双100、单10000、双10000的顺序
```

结果

```
0.000099000098676665031867
0.000099000098676665031867
0.000000009999000000009998
0.000000009999000000009998
```

第二种结果

结果

```
0.00009906291961669921875 0
0.00009900009867647163730 5
0.00000000000000000000000 0
0.00000009999000072369313
```

我们发现两种方法结果相差极小，已经超出了机器精度。但是我们通过与真值的对比发现。第一种结果的准确度更高，也就表明了，积分所带来的误差传递小于反正切函数带来的误差传递，可能是由于反正切函数涉及三角函数，计算机比较难计算。

个人心得

通过本次实验更明显地看出了双精度之于单精度的优势。当数据极小时，单精度可能会超出范围失效。双精度一般足以应对大部分情况。在用泰勒展开或者其他迭代逼近真实值时，误差限可以由有效数字位数定，但是以防万一可以用eps。

计算机计算sinx、cosx、tanx、lnx等函数时，本质上是通过泰勒展开或者其他展开来逼近得到的较为精确的值，因此这些函数本身就会产生误差，在用计算机计算复杂的式子时，要尽量减少这种函数的个数以减少误差产生源，同时尽可能化简式子，避免加减乘除运算过多引起的误差传递与累积，这样才能尽可能减小误差，提高精度。同时代数运算的积分产生的误差是小于三角函数等产生的误差的。