Q1.

// Header files and namespaces

#include<bits/stdc++.h>

using namespace std;

int fact(int n){

int ans = 1;

for(int i = 1; i < n+1; i++){

ans = ans \* i;

}

return ans;

}

int Que1( double x){

double ans = 1;

double target = cos(x);

bool req = true ;

int n = 2;

int terms = 1;

while(abs(target-ans) > 1e-7 ){

// cout << abs(target-ans) << " " ;

double val = pow(x,n)/fact(n);

// cout << val << " ";

// cout << req << " ";

if (req == true){

ans = ans - val;

req = false;

}

else{

ans = ans + val;

req = true;

}

// cout << ans << endl ;

n = n+2;

terms ++ ;

cout << ans << endl ;

}

return terms ;

}

int main() {

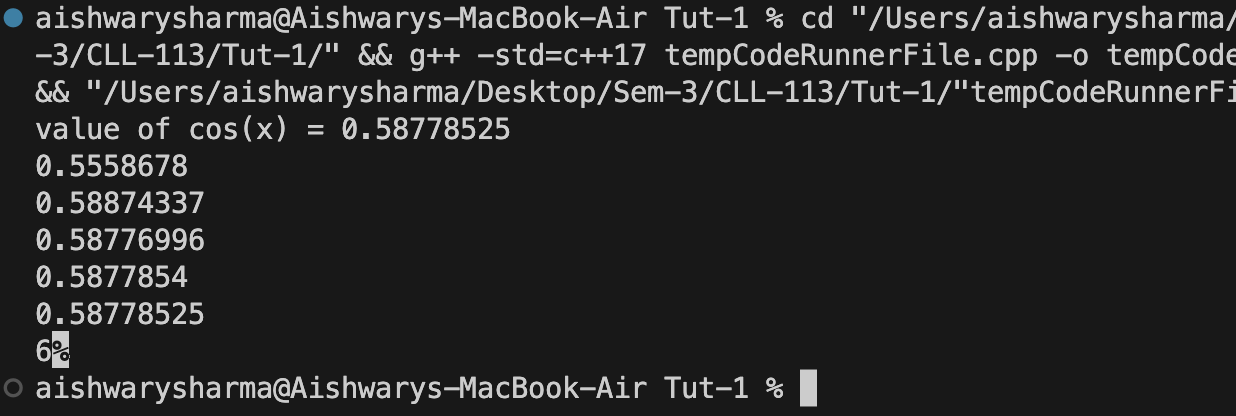
cout.precision(8);

cout << "value of cos(x) = " << cos(0.3\*M\_PI) << endl;

cout << Que1(0.3\*M\_PI);

return 0;

}

Output; 

Q2.

// Header files and namespaces

#include<bits/stdc++.h>

using namespace std;

float power(int x){

float ans = 1 ;

int i = 0;

while(i<4){

ans = ans / x ;

i++;

}

return ans ;

}

float func(){

float ans = 0 ;

for (int i = 1; i< 10000+1;i++ ){

ans = ans + power(i);

}

setprecision(7);

return ans ;

}

float func2(){

float ans = 0 ;

for (int i = 10000; i> 0; i-- ){

ans = ans + power(i);

}

return ans ;

}

int main() {

cout.precision(8);

cout << func() << endl ;

cout << func2()<< endl;

float actual = pow(M\_PI,4)/90 ;

cout<< actual<<endl ;

cout << "Percent error from 1 to 10k = " << ((actual-func())/actual)\*100 << endl;

cout << "Percent error from 10k to 1 = " << ((actual-func2())/actual)\*100;

return 0;

}

A computer screen with white text

Description automatically generated

Explanation-

Due to the limited precision of single-precision floating-point arithmetic, adding a large number of small values can lead to significant rounding errors. When summing from small to large values, the larger terms contribute more to the sum, potentially reducing the impact of rounding errors. However, when summing from large to small values, the smaller terms can get lost in the noise of the rounding errors, leading to a larger discrepancy between the computed sum and the true value.

This is why the forward and backward iterations yield different results in terms of the true per cent relative error.

Q3.

#include<bits/stdc++.h>

using namespace std;

double func(int a){

double x = 0.005 ;

double tol = x;

while ( tol >= 1e-6){

double x0 = x ;

x = (x+ a/x)/2 ;

tol = abs(x0-x);

}

return x ;

}

void truePercentRelativeError (int n, int a){

double x = 0.005;

double xt = sqrt(a);

while(n--){

double x0 = x;

x = (x+ a/x)/2 ;

cout << (abs(xt-x)/xt)\*100<< endl ;

}

}

void approximatePercentRelativeError(int n, int a){

double x = 0.005;

while(n--){

double x0 = x;

x = (x+ a/x)/2 ;

cout << (abs(x-x0)/x0)\*100 << endl ;

}

}

int main() {

cout << func(5) << endl ;

cout << endl;

truePercentRelativeError (4, 5);

cout << endl ;

approximatePercentRelativeError(4,5);

}

Output;

#include<bits/stdc++.h>

using namespace std;

double func(int a){

double x = 0.005 ;

double tol = x;

while ( tol >= 1e-6){

double x0 = x ;

x = (x+ a/x)/2 ;

tol = abs(x0-x);

}

return x ;

}

void truePercentRelativeError (int n, int a){

double x = 0.005;

double xt = sqrt(a);

while(n--){

double x0 = x;

x = (x+ a/x)/2 ;

cout << (abs(xt-x)/xt)\*100<< endl ;

}

}

void approximatePercentRelativeError(int n, int a){

double x = 0.005;

while(n--){

double x0 = x;

x = (x+ a/x)/2 ;

cout << (abs(x-x0)/x0)\*100 << endl ;

}

}

int main() {

cout << func(5) << endl ;

cout << endl;

truePercentRelativeError (4, 5);

cout << endl ;

approximatePercentRelativeError(4,5);

}

