Department of Physics, University of Dhaka
Computational Physics
Solved Problems (Functions & Arrays)

1. A block of mass m slides along a floor while a force F is applied to it at an upward angle θ.The coefficient of kinetic friction between the block and the floor is $\mu_k = 0.40$. Applying Newton's second law of motion along both the horizontal and vertical axes we get the following expression for the acceleration of the block.

$$a = \frac{F}{m}cos\theta - \mu_k(g - \frac{F}{m}sin\theta)$$

(a) Write a function double accl(double theta, double mu) that will return the value of acceleration following the above definition.

(b) Call the function **accl()** in the **main()** to write the values of acceleration for the range $\theta\epsilon\left[0,\frac{\pi}{2}\right]$ in a file. Take the increment of θ to be $\frac{\pi}{30}$. Your program should write the values of θ in the first column and the values of acceleration in the second column. Consider m=3.0 kg and F=12.0 N.

(c) Plot the acceleration Vs. θ graph and from the figure find the approximate values of θ for which the acceleration is maximum.

```
#include<iostream>
#include<cmath>
#include<fstream>
#define g 9.8
using namespace std;
double acc(double theta, double mu){
double F=12.0, m=3.0;
double c=(F/m)*cos(theta)-mu*(g-(F/m)*sin(theta));
return c;
}
int main(){
ofstream fout("acc.dat");
for(double theta=0.0;theta<=M_PI/2;theta+=M_PI/40)
{
fout<<theta<<"    "<<acc(theta,0.40)<<endl;
}
return 0;}
```

2. The value of π can be calculated by using the Leibniz series

$$1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots = \frac{\pi}{4}$$

(a) Write a C++ function double pi(int n) that will calculate the value of π using the above series.

(b) Call the function pi() in the main to calculate the value of π for the first 100 terms in the series. Also calculate the differences abs(pi(n)-M_PI) for $n\epsilon[10,500]$ and write the values of n and the difference in a data file. Plot the differences Vs. n by using gnuplot and save the plot as **pierror.png**.

```cpp
#include<iostream>
#include<cmath>
#include<fstream>
using namespace std;
double pi(int n){
double sum=0.0;
for(int i=0;i<=n;i++)
sum+=(pow(-1,i)/double(2*i+1));
return 4*sum;
}
int main(){
ofstream fout("pierror.dat");
int n;
cin>>n;
cout<<"The value of pi for the first "<<n<<" terms is "<<pi(n)<<endl;
for(n=10;n<=500;n++)
fout<<n<<"    "<<abs(pi(n)-M_PI)<<endl;
return 0;}
```

3. A sky diver of mass m falls vertically downward. At time $t = 0$, she/he attains a velocity v_0, and opens the parachute. The force of air resistance b is proportional to the velocity. The velocity of the sky diver at any time t is given by,

$$v(t) = \frac{mg}{b} + \left(v_0 - \frac{mg}{b}\right) e^{-bt/m}$$

(a) Write a C++ function double vel(double t) that returns the velocity of the diver at time t.
(b) Call the function double vel() into a main() function to find the velocity of the diver of mass $m = 60\ kg$, initial velocity $v_0 = 15\ ms^{-1}, b = 1\ kgs^{-1}$ after 10 sec of falling.
(c) Write values of t and $v(t)$ in afile for $t\epsilon[0, 100]$ with an increment of 0.01 sec and plot velocity vs. time graph for the diver.

```cpp
#include<iostream>
#include<cmath>
#include<fstream>
using namespace std;
double vel(double t){
double m=60,g=9.8,b=1,v0=15;
double a=(m*g)/b;
return (a+((v0-a)*exp(-(b*t)/m)));
}
int main(){
ofstream fout("velocity.dat");
cout<<"velocity after 10 sec is "<<vel(10)<<" m/s"<<endl;
for(double t=0.0;t<=100.0;t+=0.01)
fout<<t<<"    "<<vel(t)<<endl;
return 0;}
```

4. The Hermite Polynomials may be defined recursively by

$$H_0(x) = 1,\ \ H_1(x) = 2x$$

$$H_n(x) = 2xH_{n-1}(x) - 2(n-1)H_{n-2}(x) \quad \text{with n=2,3,...}$$

(a) Write a function double hermite (double x, int n), that recursively calculates the Hermite Polynomial of order n given an integer n from the standard input.

(b) Use the above function **hermite** (x, n) to calculate the values of $H_n(-1), H_n(1), H_{2n}(0)$ and $H_{2n+1}(0)$ for n=3,4,5.

(c) Write the values of the first five Hermite Polynomials for $x\epsilon[-3,3]$ in a file. Your program should write the values of x in the first column, the values of zeroth order Hermite Polynomial in the second column, the values of the first order Hermite Polynomial in the third column and so on. Plot the first five Hermite Polynomials in a single plot and save the plot as **hermite.png**.

```
#include<iostream>
#include<cmath>
#include<fstream>
using namespace std;
double hermite(double x, int n){
if(n==0) return 1;
else if (n==1) return 2*x;
else return (2*x*hermite(x,n-1)-2*(n-1)*hermite(x,n-2));
}
int main(){
for(int n=3;n<=5;n++){
cout<<"H_"<<n<<"(-1)= "<<hermite(-1,n)<<endl;
cout<<"H_"<<n<<"(1)= "<<hermite(1,n)<<endl;
cout<<"H_"<<2*n<<"(0)= "<<hermite(0,2*n)<<endl;
cout<<"H_"<<2*n+1<<"(0)= "<<hermite(0,2*n+1)<<endl;}
cout<<hermite(-1,3)<<endl;
cout<<hermite(0,8)<<endl;
ofstream file("hermite.dat");
for(double x=-3.0; x<=3.0;x+=0.01){
file <<x<<"  "<<hermite(x,0)<<"  "<<hermite(x,1)<<"  "<<
hermite(x,2)<<"  "<<hermite(x,3)<<"  "<<hermite(x,4)<<endl;
}
return 0;}
```

5. Write a C++ function void **mult()** that will write the product of two 3×3 matrices in a file. Call the function **mult()** in **main()** to get $\boldsymbol{A} \times \boldsymbol{B}$.

$$A = \begin{bmatrix} 2 & 6 & 8 \\ 7 & 9 & 11 \\ 3 & 6 & 9 \end{bmatrix} \text{ and } B = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 5 & 8 \\ 4 & 8 & 10 \end{bmatrix}$$

```cpp
#include<iostream>
#include<fstream>
using namespace std;
void matrix_mult(){
int a[10][10], b[10][10], c[10][10];
int r1,c1,r2,c2;
cout<<"no. of rows and columns in mat1 and mat2: ";
cin>>r1>>c1>>r2>>c2;
while(c1!=r2){
cout<<"Error\n";
cout<<"no. of rows and columns in mat1 and mat2: ";
cin>>r1>>c1>>r2>>c2;}
//matrix1 input
for(int i=0; i<r1; i++){
for(int j=0; j<c1; j++){
cout<<"a["<<i<<","<<j<<"]: ";
cin>> a[i][j];}}
//write the matrix1 in a file
ofstream fout("mult.dat");
fout<<"Matrix1"<<endl;
for(int i=0; i<r1; i++){
for(int j=0; j<c2; j++){
fout<<a[i][j]<<"\t";
}fout<<endl;}
fout<<endl;
//matrix2 input
for(int i=0; i<r2; i++){
for(int j=0; j<c2; j++){
cout<<"b["<<i<<","<<j<<"]: ";
cin>> b[i][j];}}
//write the matrix2 in a file
fout<<"Matrix2"<<endl;
for(int i=0; i<r1; i++){
for(int j=0; j<c2; j++){
fout<<b[i][j]<<"\t";}
fout<<endl;}
fout<<endl;
```

```cpp
//initialize the mult matrix
for(int i=0; i<r1; i++){
for(int j=0; j<c2; j++){
c[i][j]=0;}}
for(int i=0; i<r1; i++){
for(int j=0; j<c2; j++){
//k<r2 or k<c1
for(int k=0; k<r2; k++){
c[i][j]+=a[i][k]*b[k][j];}}}
//Show the ouput
cout<<endl;
for(int i=0; i<r1; i++){
for(int j=0; j<c2; j++){
cout<<c[i][j]<<"\t";}
cout<<endl;}
//write the ouput in a file
fout<<"The product"<<endl;
for(int i=0; i<r1; i++){
for(int j=0; j<c2; j++){
fout<<c[i][j]<<"\t";}
fout<<endl;}}
int main(){
matrix_mult();
return 0;}
```

6. The Bernoulli numbers for even integers $m = 2n$ are defined by the following series representation:

$$B_m = B_{2n} = \frac{(-1)^{n-1}\, 2\, (2n)!}{(2\pi)^{2n}} \sum_{p=1}^{\infty} p^{-2n}$$

Where $m = 2,4,6, \ldots ; or\ m = 2n\ with\ n = 1,2,3, \ldots.$

(a) Define a function **double factorial (int m)**, that returns the factorial of an integer variable m.

(b) Now write a function **double bernoulli(int n)** that calculates the Bernoulli numbers for $m = 2n$ using up to the first 20 terms in the series i.e. using $\sum_{p=1}^{20} p^{-2n}$.

(c) A series expansion for tan(x) is given by

$$\tan(x) = \sum_{n=0}^{\infty} (-1)^{n-1}\, 2^{2n}(2^{2n} - 1)B_{2n}x^{2n-1}/(2n)! \ ; \qquad for\ -\frac{\pi}{2} < x < \frac{\pi}{2}$$

Define a function **double mytan (double x, int N)** that will call the function **bernoulli ()** to evaluate tan(x) by using the first N terms in the above series. Call the function **mytan()** in the **main()** to write the values of tan(x) for the range $-\frac{\pi}{3} < x < \frac{\pi}{3}$ in a file. Take N = 20 and the increment of x as $\frac{\pi}{50}$. Also write the values of tan(x) for the same range by using the built-in function **tan()** in <cmath> in the same file and plot both of them with respect to x in a single plot. Save the plot as **tan.png.**

```
#include<iostream>
#include<cmath>
#include<fstream>
using namespace std;
//factorial
double factorial (int m){
double fact=1;
for(int i=1;i<=m;i++)
fact*=i;
return fact;
}
//Bernoulli number
double bernoulli(int n){
double sum=0.0;
double val=(pow(-1,(n+1))*2*factorial(2*n))/(pow((2*M_PI),(2*n)));
for(int p=1;p<=20;p++){
sum+=pow(p,(-(2*n)));}
double result=val*sum;
return result;
}
//tan(x)
double mytan(double x){
double sum=0.0;
for (int n=0;n<=20;n++){
sum+=((pow(-1,n-1)*pow(2,2*n)*(pow(2,2*n)-1)*bernoulli(n)*pow(x,2*n-1))/factorial(2*n));
}
return sum;
}
int main(){
ofstream file("tan.dat");
for (double x=-M_PI/3;x<=M_PI/3;x+=M_PI/50){
file<<x<<"    "<<(mytan(x))<<"    "<<tan(x)<<endl;
}

return 0;}
```

7. A numerical method to find a square root of a positive real number b is given by the recursive algorithm:

$$x_0 = 1, x_{n+1} = \frac{1}{2}\left[x_n + \frac{b}{x_n}\right], n = 0,1,2,3, \ldots.$$

(a) Write a function **double mysqroot (double b, int n)** that finds the square root of the positive real number b for a given number of iterations n and using the above algorithm.
(b) Plot the differences between the square root of b = 100.0 found from your function **mysqroot()** and from the built-in function **sqrt ()** in <cmath> for n = 1, 2, 3, . . ., 20 and save the plot as **sqrt.png**. In doing so, you have to write the values of n in the first column and the differences in the second column of a file.

```
#include<iostream>
#include<cmath>
#include<fstream>
#include<iomanip>
using namespace std;
double mysquareroot(double b, int n){
if (n==0)
return 1;
else return 0.5*(mysquareroot(b,n-1)+(b/mysquareroot(b,n-1)));
}

int main(){
ofstream fout ("sqrt.dat");
double b=100;
for(int n=1;n<=20;n++)
{
fout<<n<<"    "<<abs(mysquareroot(b,n)-sqrt(b))<<endl;
}

return 0;}
```

8. The Sackur-Tetrode equation for the entropy of a monoatomic gas is given by

$$S = NK_B[\ln\left(\frac{V}{N}\left(\frac{4\pi mU}{3Nh^2}\right)\right)^{3/2} + \frac{5}{2}]$$

Where $U = \frac{3}{2}NK_BT$the internal energy, m is is the mass of the gas molecule, K_B is the Boltzmann constant, h is the Planck's constant and N is the number of molecules or atoms at temperature T.
Define function Sackur_Tetrode(double T, double V, double N, double m) that calculates the entropy of a monoatomic gas at temperature T. Find the value of the entropy at T =300 K, V=0.025 m^3, $N = 6.023\text{X}10^{23}$ and $m = 6.64217\text{X}10^{-27}\ Kg$.

```cpp
#include<iostream>
#include<cmath>
#include<fstream>
using namespace std;
#define KB 1.38e-23
#define h 6.63e-34
double SackurTetrode(double T, double V, double N, double m){
double U=1.5*N*KB*T;
double result = N*KB*(log(V)-log(N)+1.5*log(4*M_PI*m*U)-1.5*log(3*N*h*h)+2.5);
return result;
}
int main(){
double V=0.025, N=6.023e23, T=300.0,m = 6.64216e-27;
cout<<"Entropy= "<<SackurTetrode(T,V,N,m)<<" J/K"<<endl;
return 0;}
```

9. Write a **3 × 3** matrix by taking all the elements using cin command. Print the matrix in a file in matrix form. Add the elements in each column and write in the same file below the respective column.

```cpp
#include<iostream>
#include<fstream>
using namespace std;
int main(){
double a[10][10];
for(int i=0; i<3; i++){
for(int j=0; j<3; j++){
cout<<"a["<<i<<","<<j<<"]: ";
cin>> a[i][j];
}
}
//write the matrix in a file
ofstream fout("matrix.dat");
fout<<"Matrix"<<endl;
for(int i=0; i<3; i++){
for(int j=0; j<3; j++){
fout<<a[i][j]<<"\t";
}
fout<<endl;
}
double c[10]={};
for(int j=0;j<3;j++){
double sum=0.0;
for(int i=0;i<3;i++)
{sum+=a[i][j];}
c[j]=sum;
}
for(int j=0;j<3;j++)
fout<<c[j]<<'\t';
return 0;}
```

10. The sine function has the following infinite product representation

$$sinx = x\left(1-\frac{x^2}{\pi^2}\right)\left(1-\frac{x^2}{4\pi^2}\right)\left(1-\frac{x^2}{9\pi^2}\right)\ldots\ldots = x\prod_{k=1}^{\infty}(1-\frac{x^2}{k^2\pi^2})$$

(a) Write a C++ function **double mysin(double x, int n)** that evaluates the sine function from the above definition using only the first n terms in the product.

(b) Plot the difference **abs (sin(x)-mysin(x))** Vs. **n** for $n\epsilon[10,100]$ at $x=\frac{\pi}{4}$ and $x=\frac{\pi}{2}$ in the same plot and save the plot as **dsin.png**. In doing so, you have to write the values of n in the 1st column and differences for $x=\frac{\pi}{4}$ and $x=\frac{\pi}{2}$ in the 2nd and 3rd column respectively of a data file.

```
#include<iostream>
#include<fstream>
#include<cmath>
using namespace std;
double mysin(double x, int n){
double prod=1.0;
for (int k=1;k<=n;k+=1)
prod*=1-((x*x)/(k*k*M_PI*M_PI));
return x*prod;
}
int main(){
ofstream fout("sin.dat");
for (int n=10;n<=100;n+=1){
fout<<n<<"  "<<abs(mysin(M_PI/4,n)-sin(M_PI/4))<<"  "<<abs(mysin(M_PI/2,n)-sin(M_PI/2))<<endl;
}
return 0;}
```

11. An object of mass m is dropped from a height h_0 experiences to air-resistance and is acted upon by gravity. After a time t seconds, its height is given by

$$h(t)=h_0-\frac{mg}{k}t+\frac{m^2g}{k^2}\left(1-e^{-\frac{kt}{m}}\right)$$

Where $g=9.8ms^{-2};\ k=0.1\ Nsm^{-1};\ s_0=300\ m;\ and\ m=0.25\ kg.$

(a) Write a C++ function **double height(double t)** that returns the height of the object at time t.

(b) Call the function **height()** in the **main()** to find the height of the object after 10 sec.

(c) Write the values of $h(t)$ for $t\epsilon[0,20]$ in a data file with a time interval of 0.01 second. Plot $h(t)$ vs. t by using gnuplot and save the plot as **height.png**. From the plot, find the time taken for this object to hit the ground (i.e. when $h=0$) and write it down in your exam copy.

```
#include<iostream>
#include<cmath>
#include<fstream>
using namespace std;
double height(double t){
double g=9.8, k=0.1,m=0.25, s0=300.0;
double a=(m*g*t)/k;
double b=(m*m*g)/(k*k);
double c=1-exp((-k*t)/m);
return s0-a+(b*c);
}
int main(){
cout<<"height after 10 sec is "<<height(10)<<" m"<<endl;
ofstream fout("height.dat");
for (double t=0.0;t<=20;t+=0.01){
fout<<t<<"    "<<height(t)<<endl;
}
return 0;
}
```

12. (a) Define a C++ function **eps(int i, int j, int k)** that returns the value of Levi-civita symbol. Write a function **double det(int A[][])** that returns the determinant of the matrix A by using the function **eps()**.
(b) A system of n linear equations for n unknowns can be represented in matrix multiplication form as $Ax = b$ where the $n \times n$ matrix A has a non-zero determinant and the vector $x = (x_1, x_2, .., x_n)^T$ is the column vector of the variables. Then according to Cremer's rule the solutions of the equations are

$$x_i = \frac{\det(A_i)}{\det(A)}; where\ i = 1,2, \dots, n.$$

The matrix A_i is formed by replacing the the i^{th} column of A by the column vector b
Now write a complete C++ program to find the solution of the following set of equations by using Cremer's rule (Call the function **det()** to find the various determinants).

$$2x_1 + 3x_2 - x_3 = 15$$
$$4x_1 - 3x_2 - x_3 = 19$$
$$x_1 - 3x_2 + 3x_3 = -4$$

```
#include<iostream>
using namespace std;
int epsilon(int i,int j,int k)
{return (i-j)*(j-k)*(k-i)/2;}
double det(double a[10][10]){
double determinant =0.0;
for(int i=0; i<3; i++){
for(int j=0; j<3; j++){
for(int k=0; k<3; k++){
determinant+=epsilon(i,j,k)*a[0][i]*a[1][j]*a[2][k];}}}
return determinant;}
int main(){double D[10][10],DX[10][10],DY[10][10],DZ[10][10];
int m,n;
for(m=0; m<3; m++){
for(n=0; n<3; n++){
cout<<"D["<<m<<","<<n<<"]: ";
cin>>D[m][n];}}
for(m=0; m<3; m++){
for(n=0; n<3; n++){
cout<<"DX["<<m<<","<<n<<"]: ";
cin>>DX[m][n];}}
for(m=0; m<3; m++){
for(n=0; n<3; n++){
cout<<"DY["<<m<<","<<n<<"]: ";
cin>>DY[m][n];}}
for(m=0; m<3; m++){
for(n=0; n<3; n++){
cout<<"DZ["<<m<<","<<n<<"]: ";
cin>>DZ[m][n];}}
cout<<det(D)<<endl;
cout<<det(DX)<<endl;
cout<<det(DY)<<endl;
cout<<det(DZ)<<endl;
cout<<"x= "<<det(DX)/det(D)<<endl;
cout<<"y= "<<det(DY)/det(D)<<endl;
cout<<"z= "<<det(DZ)/det(D)<<endl;
return 0;}
```

13. The differential equation for damped oscillation is given by $\frac{d^2x}{dt^2} + \frac{b}{m}\frac{dx}{dt} + \omega_0^2 x = 0$. The general solution for this equation can be written as

$$x = Ae^{-bt/2m}\cos(\omega t + \phi)$$

Where x is the displacement of an object executing damped oscillation and $\omega = \sqrt{(\frac{k}{m} - \frac{b^2}{4m^2})}$.

(a) Write a C++ function **double displ(double t, double b)** that returns the values of x using the above form of the solution. Take $m = 1\ kg, k = 4N/m$ and consider that the oscillation of the object executing damped oscillation starts from the maximum displacement i.e. from A and take $A = 1$.

(b) Call the function **displ()** in a complete C++ program to calculate the values of x with **b= 0.3** (underdamping) and **b=4** (critical damping) for $t\epsilon[0, 40]$ and write the values in a data file. Take the increment of t to be 0.01 sec.

(c) Plot the values of x with respect to t for $b = 0.3$ and 4 in a single plot using gnuplot and save the plot as **damp.png**. (You need to write t in column 1 and values of x for $b = 0.3$ in column 2, and for $b = 4$ in column 3).

```
#include<iostream>
#include<cmath>
#include<fstream>
#define k 4.0
#define m 1.0
using namespace std;
double omega(double b){
return sqrt((k/m)-((b*b)/(4*m*m)));
}
double damp(double t, double b){
double A=1.0, phi=0.0;
double p=exp((-b*t)/(2*m));
double q=cos(omega(b)*t+phi);
return A*p*q;
}
int main(){
ofstream fout("damp.dat");
for (double t=0.0;t<=40;t+=0.01){
fout<<t<<"   "<<damp(t,.3)<<"   "<<damp(t,4)<<endl;
}
return 0;}
```

14. The Legendre Polynomials may be defined recursively by

$P_0(x) = 1,\ P_1(x) = x$ with $nP_n(x) = (2n-1)xP_{n-1}(x) - (n-1)P_{n-2}(x)$ for n=2,3,4….

(a) Write a function **double legend(double x, int n)**, that recursively calculates the Legendre Polynomial of order n, at some point x, given an integer n from the standard input.

(b) Use the above function **legend()** in the **main()** to calculate the values of $P_n(1)$ and $P_{2n+1}(0)$ for n = 3, 4, 5.

(c) Plot the first five Legendre Polynomials for $x\epsilon[-1,1]$ with an increment of 0.01 and save the plot as **legendre.png**. In doing so, you should write the values of x in the 1st column, the values of zeroth order Legendre Polynomial in the 2nd column, the values of the first order Legendre Polynomial in the 3rd column and so on in a data file.

```cpp
#include<iostream>
#include<cmath>
#include<fstream>
using namespace std;
double legendre(double x, int n){
if(n==0) return 1;
else if (n==1) return x;
else return (((2*n-1)*x*legendre(x,n-1))-((n-1)*legendre(x,n-2)))/n;
}
int main(){
for (int n=3;n<=5;n++){
cout<<"P_"<<n<<"(1)= "<<legendre(1,n)<<endl;
cout<<"P_"<<2*n+1<<"(0)= "<<legendre(0,2*n+1)<<endl;
}
ofstream file("legendre.dat");
for(double x=-1.0; x<=1.0;x+=0.01){
file <<x<<"  "<<legendre(x,0)<<"  "<<legendre(x,1)<<"  "<<legendre(x,2)
<<"  "<<legendre(x,3)<<"  "<<legendre(x,4)<<"  "<<legendre(x,5)<<endl;
}
return 0;}
```

15. The **standard deviation** of a data set of n numbers $x_0, x_1, \ldots, x_{n-1}$ is defined by the formula $s = \sqrt{\frac{\sum_{i=0}^{n-1}(x_i - \bar{x})^2}{n-1}}$; Where $\bar{x}$ is the mean of the data.

(a) Write a C++ function **double mean(double A[], int n)** that will return the mean of the data.

(b) Write a function **double stdev(double A[], int n)** that will call the function mean() and use the above defined formula to return the standard deviation.

(c) Call the function **stdev()** in the function main() in order to print the standard deviation of the following data set: 12, 45, 23, 67, 48, 16, 54, 29, 49, 60, 39, 44, 27, and 56. Take the input by using cin command.

```cpp
#include<iostream>
#include<cmath>
#include<fstream>
using namespace std;
double mean(double A[], int n){
double sum=0.0;
for(int i=0;i<n;i++){
sum+=A[i];}
return sum/n;
}
double stdev(double A[], int n){
double sum=0.0;
double deviation;for(int i=0;i<n;i++){
deviation=A[i]-mean(A,n);
sum+=(deviation*deviation);
}
return sqrt(sum/(n-1));
}
int main(){int n;
cout<<"Enter the number of data ";
cin>>n;
double A[100];
for (int i=0;i<n;i++)
cin>>A[i];
cout<<stdev(A,n)<<endl;
return 0;}
```

16. The cosine function has the following infinite product representation

$$cosx = \prod_{k=1}^{\infty} 1 - \frac{4x^2}{(2k-1)^2\pi^2}$$

(a) Write a C++ function **double mycos(double x, int n)** that evaluates the cos function from the above definition using only the first n terms in the product.

(b) Plot the difference **abs (cos(x)-mycos(x))** Vs. **n** for $n\epsilon[10,100]$ at $x = \frac{\pi}{4}$ and $x = \frac{\pi}{3}$ in the same plot and save the plot as **dcos.png**. In doing so, you have to write the values of n in the 1st column and differences for $x = \frac{\pi}{4}$ and $x = \frac{\pi}{3}$ in the 2nd and 3rd column respectively of a data file.

```
#include<iostream>
#include<fstream>
#include<cmath>
using namespace std;
double mycos(double x, int n){
double prod=1.0;
for (int k=1;k<=n;k+=1){
prod*=1-((4*x*x)/(((2*k-1)*(2*k-1))*(M_PI*M_PI)));}
return prod;
}
int main(){
ofstream fout("cos.dat");
for (int n=10;n<=100;n+=1){
fout<<n<<"  "<<abs(mycos(M_PI/4,n)-cos(M_PI/4))<<"  "<<abs(mycos(M_PI/3,n)-cos(M_PI/3))<<endl;
}
return 0;}
```

17. A series expansion for the Gauss' error function is given by:

$$erf(x) = \frac{2e^{-x^2}}{\sqrt{\pi}} \sum_{k=0}^{\infty} \frac{2^k x^{2k+1}}{(2k+1)!!}$$

where $(2k + 1)!! = 1.3.5.7 \ldots (2k + 1)$ is the double factorial.

(a) Define a C++ function **double dfact(int n)** that will return the double factorial of odd numbers using the above definition.

(b) Define a C++ function **double gerf (double x, int n)** that evaluates the series expansion for the error function using the above expansion up to the n^{th} term in the series for a given value of the arguments x and n.

(c) Plot your function **gerf(x, n)** with $n = 20$, for $x \epsilon [-2.0, 2.0]$ with an increment of 0.01 using gnuplot and save the plot as **error.png**. In doing so, you have to write the values of x in the 1st column and the values of error function in the 2nd column of a data file.

```cpp
#include<iostream>
#include<cmath>
#include<fstream>
using namespace std;
double dfact(int n){
double prod=1.0;
for(int i=1;i<=n;i+=2)
prod*=i;
return prod;
}
double gerf(double x, int n){
double sum=0.0;
for(int k=0;k<=n;k++){
sum+=((pow(2,k)*pow(x,2*k+1))/dfact(2*k+1));
}
double a=(2*exp(-x*x))/sqrt(M_PI);
return a*sum;
}
int main(){
ofstream fout("error.dat");
int n=20;
for(double x=-2.0;x<=2.0;x+=0.01){
fout<<x<<"    "<<gerf(x,n)<<"   "<<erf(x)<<endl;
}
return 0;}
```

18. The Fourier series representation of the sawtooth wave f (x) = x for $x\epsilon[-\pi,\pi]$ with $f(x \pm 2\pi) = f(x)$, is given by

$$f(x) = 2[sinx - \frac{sin2x}{2} + \frac{sin3x}{3} - \cdots + \frac{(-1)^{n+1}sinnx}{n} + \cdots]$$

(a) Write a C++ program that evaluates the n-th partial sum (i.e. sum of the first n terms), where n = 2, 3, 4, . . . 8 of the series, for x ∈ [−π, π] and writes the values in a file. Take the increment of x to be π/50. Your program should write the values of x in the first column, the partial sum for n = 2 in the second column of the file, that for n = 3 in the third column, etc.

(b)Plot the partial sums (for n = 2, 3, 4,…, 8) in a single plot, using gnuplot or any other available plotting program, for x ∈ [−π, π].

```cpp
#include<iostream>
#include<cmath>
#include<fstream>

using namespace std;
double fourier(double x, int n){
double sum;
for(int i=1;i<=n;i++){
sum+=(pow(-1,i+1)*sin(i*x))/i;
}
return 2*sum;
}
int main(){
ofstream fout("sum.dat");
int n; double x;
for( x=-M_PI;x<=M_PI; x+=M_PI/50){
fout<<x<<"    "<<fourier(x,2)<<"    "<<fourier(x,3)<<"    "<<fourier(x,4)<<"    "
<<fourier(x,5)<<"    "<<fourier(x,6)<<"    "<<fourier(x,7)<<"    "<<fourier(x,8)<<endl;
}
return 0;
}
```

19. A periodic function $f(x)$, with the fundamental period [-π ,π], has the Fourier expansion

$$f(x) = \frac{1}{\pi} + \frac{1}{2}sinx - \frac{2}{\pi}\left(\frac{cos2x}{2^2-1} + \frac{cos4x}{4^2-1} + \frac{cos6x}{6^2-1} + \cdots\right)$$

(a) Write a C++ function **double fourier(double x, int n)**, that evaluates the n-th partial sum in the Fourier series expansion (i.e. upto the term $\frac{\cos(2nx)}{(2n)^2-1}$ in the bracket).

(b) Call the function **fourier()** in a complete C++ program and evaluate $f(x)$ with $n = 2,3,4,6$ for $x\epsilon[-3\pi, 3\pi]$ and write the values in a file. Take the increment of x to be $\pi/20$. Plot the partial sums in a single plot using gnuplot and save the plot as **periodic.png**.

```cpp
#include<iostream>
#include<cmath>
#include<fstream>
using namespace std;
double fourier(double x, int n){
double a=(1/M_PI)+(0.5*sin(x));
double sum=0.0;
for(int i=1;i<=n;i++)
sum+=(cos(2*i*x)/(pow((2*i),2)-1));
return a-((2/M_PI)*sum);
}
int main(){
ofstream fout("fourier1.dat");
for(double x=-3*M_PI;x<=3*M_PI;x+=M_PI/20)
fout<<x<<"      "<<fourier(x,2)<<"      "<<fourier(x,3)<<"      "<<fourier(x,4)<<
"      "<<fourier(x,5)<<"    "<<fourier(x,6)<<endl;
return 0;}
```

20. The ramanujan's approximation to the factorial of a large number n is given by

$$n! = \sqrt{\pi}\, n^n e^{-n} \left(8n^3 + 4n^2 + n + \frac{1}{30}\right)^{1/6}$$

(a) Write a C++ function **double myfact(int n)** that evaluates the factorial of a number directly from the definition $n! = n(n-1)!$ With $0! = 1$

(b) Write a C++ function **Ramanujan(int n)** that evaluates the factorial of a number n from ramanujan's approximation.

(c) Plot the diffferences of the two values of $n!$ (calculated as above) divided by the exact value found from the function myfact() for n=2,4,6,….,12.

```
#include<iostream>
#include<cmath>
#include<fstream>
using namespace std;
double myfact(int n){
if (n==0)
return 1;
else if (n==1)
return 1;
else return n*myfact(n-1);
}
double myramanujan(int n){
double a=sqrt(M_PI)*pow(n,n)*exp(-n);
double b=8*pow(n,3)+4*pow(n,2)+n+0.033333333;
return a*pow(b,.1666666666666);
}
int main(){
ofstream fout("fact.dat");
int n;
cin>>n;
cout<<myfact(n)<<endl;
cout<<myramanujan(n)<<endl;
for(n=2;n<=12;n++)
fout<<n<<"    "<<(abs(myramanujan(n)-myfact(n)))/myfact(n)<<endl;
return 0;}
```

21. $\tan^{-1}(x)$ has the following series representation

$$\tan^{-1}(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \cdots; for\ |x| \leq 1$$

(a) Write a C++ function **double invtan(double x, int n)** that will calculate the values of $\tan^{-1}(x)$ at a given value of x by using the first n terms in the above series.

(b) Call the function **invtan()** into the **main()** to find the values of $\tan^{-1}(x)$ at $x = 1$. Use n=1000.

(c) Write the values of the difference **abs(invtan(x,n)-atan(x))** in a data file at $x = 1$ for $n\epsilon[10,1000]$and plot the differences with respect to n and save the plot as **invtan.png**.

```cpp
#include<iostream>
#include<cmath>
#include<iomanip>
#include<fstream>
using namespace std;
double invtan(double x, int n){
double sum=0;
for(int i=0; i<=n; i++){
sum+=pow(-1,i)*((pow(x,(2*i)+1))/((2*i)+1));
}
return sum;
}
int main()
{ofstream fout("invtan.dat");
cout<<setprecision(7)<<fixed;
cout<<invtan(1,1000)<<endl;
for(int n=10;n<=1000;n++)
fout<<n<<"    "<<abs(invtan(1,n)-atan(1))<<endl;
return 0;
}
```

22. The Riemann Zeta function $\zeta(s)$ is defined by the following series representation

$$\zeta(s) = \sum_{n=1}^{\infty} n^{-s}, \quad s > 1$$

Write a program that approximately calculates the Zeta function $\zeta(s)$ for s = 2, 4, 6, , …, 20. In doing so, Use the summation $\sum n^{-s}$ up to the first 30 terms i.e. $\zeta(s) = \sum_{n=1}^{30} n^{-s}$ to calculate the value of the given function defined by zeta (int s). Plot the function with respect to s for the values given above.

(c) The Euler-Mascheroni constant is defined in terms of the ζ -function

$$\gamma = \sum_{s=2}^{\infty} \frac{(-1^s)\zeta(s)}{s}$$

Evaluate the approximate value of γ by including 19 the first terms i.e. $\sum_{s=2}^{20} \frac{(-1)^s\zeta(s)}{s}$.

```cpp
#include<iostream>
#include<cmath>
#include<fstream>
using namespace std;
double zeta(int s){
double sum=0.0,n;
for(n=1;n<=30;n++)
sum+=pow(n,-s);
return sum;
}
//Euler-Mascheroni
double Euler_Mascheroni(int s){
double result=0.0;
for(int i=2;i<=s;i++)
result +=pow(-1,i)*(zeta(i)/i);
return result;
}
int main(){
ofstream fout ("zeta.dat");
for(int s=2;s<=20;s+=1)
fout<<s<<"    "<<zeta(s)<<endl;
cout<<Euler_Mascheroni(20)<<endl;

return 0;}
```

23. Write a C++ program to find the solutions of the following set of equations by using inverse matrix method.

$$2x_1 + 3x_2 - x_3 = 15$$
$$4x_1 - 3x_2 - x_3 = 19$$
$$x_1 - 3x_2 + 3x_3 = -4$$

#include<iostream>

```
using namespace std;
int epsilon(int i,int j,int k)
{return (i-j)*(j-k)*(k-i)/2;}
double det(double a[3][3]){
double determinant =0.0;
for(int i=0; i<3; i++){
for(int j=0; j<3; j++){
for(int k=0; k<3; k++){
determinant+=epsilon(i,j,k)*a[0][i]*a[1][j]*a[2][k];}}}
return determinant;}
int main(){
double mat[3][3],co_mat[3][3],trans_mat[3][3],inv_mat[3][3],mult_mat[3][3],B_mat[3][3];
//Store the matrix mat[3][3]
int i,j;
for(i=0;i<3;i++){for(j=0;j<3;j++)
{cout<<"mat["<<i<<", "<<j<<"]: ";
cin>>mat[i][j];}}
if(det(mat)==0)         cout<<"The matrix is not invertible!\n";
//To display  the co-factor matrix
for(i=0;i<3;i++){for(j=0;j<3;j++)
{cout<<(mat[(i+1)%3][(j+1)%3]*mat[(i+2)%3][(j+2)%3])
                -mat[(i+1)%3][(j+2)%3]*mat[(i+2)%3][(j+1)%3]<<"\t";}
cout<<endl;}
//To store the co-factor matrix
for(i=0;i<3;i++){for(j=0;j<3;j++)
{cout<<"co_mat["<<i<<", "<<j<<"]: ";    cin>>co_mat[i][j];}}
//Transpose of the co-factor matrix
for(i=0;i<3;i++){for(j=0;j<3;j++)
{trans_mat[j][i]=co_mat[i][j];}}
//display the transpose matrix (the adjoint matrix) of the co-factor matrix
cout<<endl<<"The adjoint matrix is:\n";
for(i=0;i<3;i++){for(j=0;j<3;j++)
{cout<<trans_mat[i][j]<<"\t";}  cout<<endl;}
//The inverse matrix
for(i=0;i<3;i++){for(j=0;j<3;j++)
{inv_mat[i][j]=trans_mat[i][j]/det(mat); }}
//To display the inverse matrix
```

```
cout<<endl<<"The inverse matrix is:\n";
for(i=0;i<3;i++){for(j=0;j<3;j++)
{cout<<inv_mat[i][j]<<"\t";}    cout<<endl;}
//Store the no. of rows and columns of matrix inv_mat and B
double r1,c1,r2,c2;
cout<<"The no. of rows and columns of matrix inv_mat and B\n";
cin>>r1>>c1>>r2>>c2;
//Store the matrix B
for(i=0;i<r2;i++){for(j=0;j<c2;j++)
{cout<<"B_mat["<<i<<", "<<j<<"]: ";
cin>>B_mat[i][j];}}
//To display the B_mat matrix
cout<<endl<<"The matrix B_mat is :\n";
for(i=0;i<r2;i++){for(j=0;j<c2;j++)
{cout<<B_mat[i][j]<<"\t";}      cout<<endl;}
//mult_matrix_initialization to 0
for(int i=0; i<r1; i++){for(int j=0; j<c2; j++){
mult_mat[i][j]=0;}}
//multiplication of inv_mat and B_mat
for(int i=0; i<r1; i++){
for(int j=0; j<c2; j++){
for(int k=0; k<r2; k++){                    //k<r2 or k<c1
mult_mat[i][j]+=inv_mat[i][k]*B_mat[k][j];}}}
//To display multiplication result matrix
cout<<endl<<"The output is :\n";
for(i=0;i<r1;i++){for(j=0;j<c2;j++)
{cout<<mult_mat[i][j]<<"\t";}   cout<<endl;}
//To display the solutions
cout<<"x= "<<mult_mat[0][0]<<"; "<<"y= "<<mult_mat[1][0]<<"; "<<"z= "<<mult_mat[2][0]<<endl;
return 0;}
```

24. Write a C++ function void sort(int a[], int n) that sorts the n elements of an array a[]. Call the function sort() in the main() to sort the elements of the array a[]={55, 22, 99, 66, 44, 88, 33, 100, 77} in an ascending order.

```
#include<iostream>
using namespace std;
void sort(int a[], int n)
{
for (int i=1; i<n; i++)
 {
   for (int j=0; j<n-i; j++)
   {
     if (a[j]>a[j+1]) swap(a[j],a[j+1]);
   }

  }
for (int i=0; i<n; i++)
      {cout<<a[i]<<"\t";
      }
      cout<<endl;
}

int main()
{
int a[]={55, 22, 99, 66, 44, 88, 33, 100, 77};
sort(a, 9);
return 0;
}
```