

Appendix

Listing –1 Across wind moments (ACI method)

```
#include <iostream.h>
#include <conio.h>
#include <fstream.h>
#include <math.h>

#define SQ(a) ((a)*(a))

void main()
{
float Vel(float, float);
float Dia(float);
// This program calculates the Moments due to Across Wind effects
// Using the ACI method

// Data assumed for the chimney
const float ht = 250; // meters
const float bdia = 20.8333; // meters
const float tdia = 12.5; // meters
const float bthk = 0.595238; // meters
const float tthk = 0.278274; // meters

// Data required as input for each mode shape - File input
float freq; // time period in that mode shape - to calculate freq later on
float mass56; // mass value at 5/6 the height
int modeno;

// Variables
char fname[20]; // The input file name
// Get the file name
clrscr();
cout << "Enter the file name : ";
cin >> fname;

ifstream fin(fname,ios::in);
fin >> modeno;
fin >> freq;
fin >> mass56;
mass56 = mass56 * 32 / (250 * 100); // weight per unit meter kN/m
mass56 = mass56 * 68.53; // lb/ft conversion

fin.close();

float Vzcr, Vcr; // The control wind velocities to be calculated
float strouhal;
strouhal = 0.25 * (0.333 + 0.206*log(820.25/45.569) );
Vzcr = 40*pow((5*250)/(6*10),.14);
Vcr = freq * 13.89 / strouhal;
// Convert to ft/sec
Vzcr = 3.281 * Vzcr;
Vcr = 3.281 * Vcr;
```

```

// Some variables
float term1, term2, term3; // The three terms the eqn is divided into
float i, cl0, fb, cl;
float V;
float modeshp = modeno==1?0.57:0.18;

// Calculations
i = 1/(log((820.25*5/6)/0.018) );
cl0 = -0.243 + 5.648*i - 18.182*i*i;
fb = -0.089 + 0.337*log(820.25/45.569);
if (fb > 1.0) fb = 1.0;
if (fb < 0.2) fb = 0.2;
cl = cl0 * fb;
term1 = (4.0/32.2)*modeshp*cl*(0.075/2)*SQ(Vcr)*45.569*SQ(820.25);

//cout << term1 << endl;
//cout << cl;

// Variables to be calculated inside with V
float betas, betaa, k, ka0, ka, sp;
float Ma, Maold=0;
// loop for the maximum Moment
for(V = 0.5*Vzcr; V < 1.3*Vzcr; V+=1) // do for i m/s intervals
{
    betas = 0.01 + (0.1*(V-Vzcr) )/(Vzcr);
    if (betas < 0.01) betas = 0.01;
    if (betas > 0.04) betas = 0.04;
    k = V/Vcr;
    ka0 = (-1.0) / ((1+5*i) *(1+(abs(k-1)/(i+0.1) ) ) );
    ka = ka0*fb;
//    cout << ka << endl;
    betaa = ka * 0.075 * SQ(45.569)/(mass56);
    sp = (pow(k,1.5)/(sqrt((0.1+2*i)*sqrt(M_PI) ) ) )*(exp((-1/2)*SQ((1-1/k)/(0.1+2*i) ) ) );
//    cout << betaa << endl;
    term2 = sqrt(M_PI)/(4*(betas + betaa) ) );
//    cout << term2 << " | ";
    term3 = sp*((2*1.2) / (820.25/45.569 + 3) );
    Ma = term1 * term2 * term3; // MNm
    if(abs(Ma) < abs(Maold) ) break;
    Maold = Ma;

//    cout << Ma << endl;
}
Ma = Maold;
// Convert to SI
Ma = Ma * 1.356 / 1000000;
cout << endl << "Output using data in file : " << fname << " according to the code ACI307-95";
cout << endl << "Base Moment for across winds : " << Ma << " MNm";

////////////////////////////////////

// Calculation of Base Along wind moments
float Vr = 89.5; //mph can be converted into req ft/s
float deltaz=10; //10 ft
float het, htmean;
float mom=0;

```

```

for(het=0; het<820.25; het+=deltaz)
{
    htmean = het + deltax/2;
    mom += 0.65*Dia(htmean) * 0.0013 * SQ(Vel(Vr,htmean) ) * htmean * deltax;
}
mom = mom * 1.356 / 1000000;
cout << endl << "Base Moment for along winds : " << mom << " MNm";

float gust;
gust = 1.3 + (11*pow(Vel(Vr,33)/freq,0.47) )/(pow((820.25+16),0.86) );
cout << endl << "Gust factor : " << gust;

cout << endl << endl << "Net Moment : " << sqrt(SQ(gust*mom)+SQ(Ma));

getch();

} // End Main

float Vel(float var, float z)
{
    float me = 1.47 * 0.78 * pow((80/var),0.09) * var * pow((z/33),0.14);
    return me;
}

float Dia(float z)
{
    z = z / 3.281; //m
    float me = 20.83333 - (20.83333-12.5)*z/250;
    me = me * 3.281; // ft
    return me;
}

```

Listing – 2 Across-wind moments (IS simplified method)

```

#include <iostream.h>
#include <conio.h>
#include <fstream.h>
#include <math.h>

void main()
{
// This program calculates the Moments due to Across Wind effects
// Using the approximate method given in IS:4998

// Data assumed for the chimney
const float ht = 250; // meters
const float bdia = 20.8333; // meters
const float tdia = 12.5; // meters
const float bthk = 0.595238; // meters
const float tthk = 0.278274; // meters

```

```

// Data required as input for each mode shape - File input
float freq; // time period in that mode shape - to calculate freq later on
float mass[32]; // mass values at nodes
float phi[32]; // The normalised mode shape function values wrt top

// Variables
char fname[20]; // The input file name
int i;
// Get the file name
clrscr();
cout << "Read Weights and Phi values from file? : ";
cin >> fname;

ifstream fin(fname,ios::in);
fin >> freq;
for(i=0; i<32; i++)
{
    fin >> mass[i];
    fin >> phi[i];
}

fin.close();

// The various requirements
float etaNum[32];
float denom[32];
float MeqNum[32];
// Calculate the various values required for eta and Meq
for(i=0; i<32; i++)
{
    etaNum[i] = (tdia + (bdia - tdia)/31*(31-i))*phi[i];
    MeqNum[i] = mass[i]*phi[i]*phi[i];
    denom[i] = phi[i]*phi[i];
}

float numm=0, numfac=0, den=0; // the sum of all the factors - basically integration
for(i=0; i<32; i++)
{
    numm += MeqNum[i];
    numfac += etaNum[i];
    den += denom[i];
}

// Variables for calculation
float Meq, Ksi, eta;
Meq = numm / (den * (ht/32));
Ksi = (2*Meq*(2*M_PI*0.016)) / (1.2*pow((tdia + (bdia - tdia)/6),2) );
eta = (numfac / den)*(0.16/(4*M_PI*0.04*Ksi));
cout << "eta : " << eta << endl;

// Calculate the moments
int j;
float integr;
float mom;

ofstream fout("out.txt", ios::out);

```

```

cout << "Writing to file..." << endl;
fout << "Moments for data from " << fname << " (MNm)" << endl;
for(j=0; j<32; j++)
{
    integr=0;
    for(i=j; i<32; i++)
    {
        integr += mass[i]*phi[i]*ht*(i-j)/31;
    }
    mom = 4*pow(M_PI,2)*pow(freq,2)*eta*integr/1000000; // MNm
    fout << mom << endl;
} // outer loop for the position
cout << "Done. Hit any Key";
fout.close();

getch();

} // End Main

```

Listing – 3 Across wind moments (IS Random Response method)

```

#include <iostream.h>
#include <conio.h>
#include <fstream.h>
#include <math.h>

#define SQ(a) ((a)*(a))

void main()
{
// This program calculates the Moments due to Across Wind effects
// Using the Random Response method given in IS:4998

// Data assumed for the chimney
const float ht = 250; // meters
const float bdia = 20.8333; // meters
const float tdia = 12.5; // meters
const float bthk = 0.595238; // meters
const float tthk = 0.278274; // meters

// Data required as input for each mode shape - File input
float freq; // time period in that mode shape - to calculate freq later on
float mass[32]; // mass values at nodes
float phi[32]; // The normalised mode shape function values wrt top

// Variables
char fname[20]; // The input file name
int i;
// Get the file name
clrscr();
cout << "Read Weights and Phi values from file? : ";
cin >> fname;

```

```

ifstream fin(fname,ios::in);
fin >> freq;
for(i=0; i<32; i++)
{
    fin >> mass[i];
    fin >> phi[i];
}

fin.close();

// The various requirements
float phisqarr[32];
float dia[32];
float denom[32];
float MeqNum[32];
// Calculate the various values required for eta and Meq
for(i=0; i<32; i++)
{
    phisqarr[i] = phi[i]*phi[i]*ht/32;
    dia[i] = bdia - (bdia-tdia)/31*i;
    MeqNum[i] = mass[i]*phi[i]*phi[i];
    denom[i] = phi[i]*phi[i];
}

float numm=0, den=0, phisqr=0; // the sum of all the factors - basically integration
for(i=0; i<32; i++)
{
    numm += MeqNum[i];
    den += denom[i];
    phisqr += phisqarr[i];
}
// Some required values
float Meq, diabar;
Meq = numm / (den * (ht/32));
diabar = tdia + (bdia-tdia)/6;

// Variables for calculation
float Num, Den, eta, etaold=-1000; // This is unreachable
float taper;
// Starting from H calculate the eta for all values and take the maximum val
// The equaion has been taken not from code but dm's interpretation
float fun, funold = -1000000;

int j;
for(j=31; j>=0; j--)
{
    taper = (1-tdia/bdia)/(ht/bdia) + (0.1*dia[j])/(j*ht/31);
    Num = 4.0*0.12*phi[j]*1.23*pow(dia[j],4)*sqrt(M_PI*1.0/(2*taper));
    Den = 8*SQ(M_PI)*SQ(0.2)*Meq*phisqr*sqrt(0.016 - 0.5*1.2*SQ(diabar)/Meq);
    eta = Num / Den;

    fun = (dia[j]*phi[j])/(sqrt(taper));
    //if (funold > fun) break;
    //funold = fun;
    //cout << "fun " << fun << endl;
}

```

```

        //cout << eta << endl;
        if(eta < etaold) break;
        etaold = eta;
    }
    eta = etaold; // The maximum value
    cout << "eta : " << eta << endl;

// Calculate the moments
float integr;
float mom;

ofstream fout("out.txt", ios::out);
cout << "Writing to file..." << endl;
fout << "Moments for data from " << fname << " (MNm)" << endl;
for(j=0; j<32; j++)
{
    integr=0;
    for(i=j; i<32; i++)
    {
        integr += mass[i]*phi[i]*ht*(i-j)/31;
    }
    mom = 4*pow(M_PI,2)*pow(freq,2)*eta*integr/1000000; // MNm
    fout << mom << endl;
} // outer loop for the position
cout << "Done. Hit any Key";
fout.close();

getch();

} // End Main

```

Listing – 4 Interaction curves

```

#include <iostream.h>
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
// Interaction curve of N vs M (or n vs m)
// Definitions that will be used in the whole of the program
// Know that this is very baad prog but will have to do
#define fck1 20 // N/mm2 actually fck'
#define fsyk 415 // N/mm2
#define rho 1.5 // percent %
#define epsC1 0.002 // The ultimate comp strain in conc
#define gammaS 1.5 // The safety factor for steel
#define gammaC 2.0 // The safety factor for concretes
#define Es 200000 // N/mm2
#define epsSml 0.07 // The ultimate strain for steel
#define epsSy (fsyk)/((gammaS)*(Es)) // Calculate the yield for steel
#define alphastep 1 // The step for the alpha calculations
// The angles are maintained as degrees but are converted into radians just

```

```

// before calling the functions or using in the calculations

// Globals
float alpha0; // for the simple reason that i do not want to pass it

void main()
{
// This is for <testing>
//float rho;
//rho = 0.2;
// </testing>
float fc(float eps);
float fs(float eps);
float eps(float alpha);
float correct(float, float);

//cout << fc(0.0015);
//cout << epsSy;
//cout << fs(0.0015) << endl;
//alpha0=0;
//cout << eps(150) << endl;

float maxn;

// Beginning
float alpha;
for(alpha0=0; alpha0<=180; alpha0+=1)
{
// Stepping through alpha0 by steps of one
float nc, mc, ns, ms, n,m;
nc = mc = ns = ms = m = n = 0;
// Calc nc, mc, ns, ms;
// Calc nc
for(alpha=alpha0; alpha<=180; alpha+=alphastep)
{
nc += fc(eps(alpha))*(M_PI*alphastep/180);
}
nc = nc*2*(1-(rho/100))/fck1;
// Calc mc
// Clockwise moment is negative (because of cos())
// Will change it later to positive (sign change)
for(alpha=alpha0; alpha<=180; alpha+=alphastep)
{
mc += fc(eps(alpha))*cos(alpha/180*M_PI)*(M_PI*alphastep/180);
}
mc = mc*2*(1-(rho/100))/fck1;
// Calc ns
for(alpha=alpha0; alpha<=180; alpha+=alphastep)
{
ns += fs(eps(alpha))*(M_PI*alphastep/180);
}
for(alpha=0; alpha<alpha0; alpha+=alphastep)
{
ns += fs(eps(alpha))*(M_PI*alphastep/180);
}
ns = ns*2*(rho/100)/fck1;

```

```

// Calc ms
for(alpha=alpha0; alpha<=180; alpha+=alphastep)
{
    ms += fs(eps(alpha))*cos(alpha/180*M_PI)*(M_PI*alphastep/180);
}
for(alpha=0; alpha<alpha0; alpha+=alphastep)
{
    ms += fs(eps(alpha))*cos(alpha/180*M_PI)*(M_PI*alphastep/180);
}
ms = ms*2*(rho/100)/fck1;

// Correcting for the new suggested fc curve
maxn = (fc(epsC1)*2*M_PI)/(fck1);
nc = correct(nc,maxn);
mc = correct(mc,maxn);
// Calc n,m
n = nc + ns;
m = mc + ms;
// Correct the sign of the moment
m = (-1)*m;
if(n<0)
{
    break;
}
// Debugging o/p format
// cout << alpha0 << " a0 " << nc << " nc " << mc << " mc " << ns << " ns" << ms << " ms" <<
endl;
// cout << alpha0 << " a0 " << n << " n " << m << " m " << endl;
cout << m << " , " << n << endl;
}
//cout << maxn;
}

// Function definitions
float fc(float eps)
{
    float ans;
    ans = (2*(eps/epsC1 ))-((eps/epsC1 )*(eps/epsC1 ));
    ans = ans * (0.85 * fck1 * gammaC );
    return ans;
}

float fs(float eps)
{
    float ans;
    if(fabs(eps) <= epsSy)
    {
        ans = Es*eps;
    }
    else
    {
        ans = fsyk*(eps/fabs(eps) )/gammaS;
    }
    return ans;
}

```

```

float eps(float alpha)
{
// Tension -ve compression positive
    float ans;
    float alp = (alpha/180)*M_PI;
    float alp0 = (alpha0/180)*M_PI;
    ans = (-epsC1)*((cos(alp)-cos(alp0)) / (1+cos(alp0)));
    return ans;
}

float correct(float n, float nmax)
{
n = n *(1 + (n/nmax*((0.95/0.85)-1)));
return n;
}

```