

probal.cpp

```

// probal.cpp - BALANCE A PROPELLER FROM TEST WEIGHT VIBRATIONS
//
// MAINTENANCE HISTORY
// DATE          PROGRAMMER AND DETAILS
// 10-09-10      JS          Original
//
//-----

#include <cmath>           // C-style mathematical functions
#include <cstdlib>         // C-style standard library
#include <iostream>       // C++ I/O stream declarations
using namespace std;     // Expand the standard namespace

//-----

// INPUT DATA STRUCTURE

struct Samp_t {
    double          Vi;           // Accelerometer output voltage
    double          Ri;           // Moment of test weight (gram-inches)
    double          Pi;           // Phase of test weight (degrees)
};

//-----

// INPUT DATA

struct Samp_t SAMP_ARR[] = {
    // V_OUT      Moment          Phase
    {0.510,      0.00,           0.00},
    {1.546,      309.53,        39.53},
    {1.141,      309.53,        159.54},
    {0.892,      309.53,        279.54},
    {0.391,      126.03,        235.65},
};

static const size_t N = sizeof(SAMP_ARR) / sizeof(SAMP_ARR[0]);

//-----

// CALCULATE THE SUM OF ERRORS

double
CalcSumE2 (
    double          Vn,           // Noise voltage
    double          Pp,           // Phase of unbalanced propeller
    double          Rp,           // Moment of unbalanced propeller
    double          Vpgi)         // Volts per gram-inch
{
    const Samp_t   *s;           // Sample pointer
    double          x, y;         // Moments in x and y directions
    double          e;           // This error
    double          e2;          // Sum of errors
    size_t          i;           // General purpose index

    e2 = 0;
    for (i = 0; i < N; i++) {
        s = SAMP_ARR + i;
        x = Rp*sin(Pp*M_PI/180.0) + s->Ri*sin(s->Pi*M_PI/180.0);
        y = Rp*cos(Pp*M_PI/180.0) + s->Ri*cos(s->Pi*M_PI/180.0);
        e = s->Vi - Vn - Vpgi*sqrt (x*x + y*y);
        e2 += e * e;
    }
    return e2;
}

```

```
}  
  
//-----  
  
int  
main ()  
{  
    size_t          i;                // General purpose index  
    double          Vn;               // Noise voltage  
    double          Pp;               // Theta of unbalanced propeller  
    double          Rp;               // Radius of vibration of unbal prop  
    double          Vpgi;             // Volts per gram-inch  
    double          e2;               // Sum of errors squared  
    double          bestE2;           // Best sum of errors squared  
    double          bestVn;           // Best Vn  
    double          bestPp;           // Best Pp  
    double          bestRp;           // Best Rp  
    double          bestVpgi;         // Best Vpgi  
    double          bestX;            // Vibration moment in X direction  
    double          bestY;            // Vibration moment in Y direction  
    double          x, y;             // Moments in x and y direction  
  
    // Search for a basic solution  
  
    bestE2 = 1e+128;  
    bestVn = 0;  
    bestPp = 0;  
    bestRp = 0;  
    bestVpgi = 0;  
    for (Pp = 0; Pp < 360.0; Pp += 1.0)  
    for (Vn = 0.0; Vn < 0.6; Vn += 0.05)  
    for (Rp = 0; Rp < 250.0; Rp += 5.0)  
    for (Vpgi = 0; Vpgi < 5e-3; Vpgi += 50e-6) {  
        e2 = CalcSumE2 (Vn, Pp, Rp, Vpgi);  
        if (e2 < bestE2) {  
            bestE2 = e2;  
            bestVn = Vn;  
            bestPp = Pp;  
            bestRp = Rp;  
            bestVpgi = Vpgi;  
        }  
    }  
  
    // Tweek the values to find the best solution  
  
    for (;;) {  
        e2 = CalcSumE2 (bestVn+0.001, bestPp, bestRp, bestVpgi);  
        if (e2 < bestE2) {  
            bestE2 = e2;  
            bestVn += 0.001;  
            continue;  
        }  
  
        e2 = CalcSumE2 (bestVn-0.001, bestPp, bestRp, bestVpgi);  
        if (e2 < bestE2) {  
            bestE2 = e2;  
            bestVn -= 0.001;  
            continue;  
        }  
  
        e2 = CalcSumE2 (bestVn, bestPp+0.1, bestRp, bestVpgi);  
        if (e2 < bestE2) {
```

```
        bestE2 = e2;
        bestPp += 0.1;
        continue;
    }

    e2 = CalcSumE2 (bestVn, bestPp-0.1, bestRp, bestVpgi);
    if (e2 < bestE2) {
        bestE2 = e2;
        bestPp -= 0.1;
        continue;
    }

    e2 = CalcSumE2 (bestVn, bestPp, bestRp+0.1, bestVpgi);
    if (e2 < bestE2) {
        bestE2 = e2;
        bestRp += 0.1;
        continue;
    }

    e2 = CalcSumE2 (bestVn, bestPp, bestRp-0.1, bestVpgi);
    if (e2 < bestE2) {
        bestE2 = e2;
        bestRp -= 0.1;
        continue;
    }

    e2 = CalcSumE2 (bestVn, bestPp, bestRp, bestVpgi+5e-6);
    if (e2 < bestE2) {
        bestE2 = e2;
        bestVpgi += 5e-6;
        continue;
    }

    e2 = CalcSumE2 (bestVn, bestPp, bestRp, bestVpgi-5e-6);
    if (e2 < bestE2) {
        bestE2 = e2;
        bestVpgi -= 5e-6;
        continue;
    }

    break;
}

cout << "bestE2      = " << bestE2 << '\n';
cout << "bestVn      = " << bestVn << '\n';
cout << "bestPp      = " << bestPp << '\n';
cout << "bestRp      = " << bestRp << '\n';
cout << "bestVpgi     = " << bestVpgi << '\n';

return 0;
}
```