**English specifications of Problem5 - Angle Computations**

1. **Input Data Variables**
   1. **typedef construct1**

typedef struct

{

double latitude;

double longitude;

}COORDINATES;

COORDINATES input;

* 1. **typedef construct2**

typedef struct

{

double l\_angle;

double g\_local;

}DATA;

DATA data;

* 1. **typedef construct3**

typedef struct

{

float gx;

float gy;

float gz;

float ax;

float ay;

float az;

}Chk\_Val;

Chk\_Val check\_val;

* 1. **double ppcf =0;**
  2. **double ppcf\_z = 0;**
  3. **int K\_correct = 0;**

**Input Data Table and Constraints**

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Data Type** | **Constraint** |
| latitude | double | -90 to 90 |
| longitude | double | 0 to 360 |
| I\_angle | double | 0 |
| g\_local | double | -9.5 to 9.5 |
| gx | float | -100 to 100 |
| gy | float | -100 to 100 |
| gz | float | -100 to 100 |
| ax | float | -1.5 to 1.5 |
| ay | float | -1.5 to 1.5 |
| az | float | -1.5 to 1.5 |
| ppcf | double | 0 |
| ppcf\_z | double | 0 |
| K\_correct | int | 0 |

1. **Output Data Variables**

|  |  |
| --- | --- |
| **Variable Name** | **Data Type** |
| output1 | double |
| output2 | double |
| output3 | double |

1. **Computations on input data to derive output parameters**

Step1: Derive Direction Matrix (DM)

Step2: Compute Angle from DM

Step2: Calculate output3 from angle\_si (which is computed in step2)

**Step1: Computations and logics for deriving DM**

**Related terms calculations**

*pi = 4.0\*atan(1.0);*

*cdr = (pi)/180.0;*

*earth\_rate = 15.0409 \* cdr/3600.0*

DM is matrix multiplication of X3x3 & Y3x3, where X and Y are defined as follows

**Derivation of X3x3**

1. Derive gravity[3]

*gravity[0] = -0.0*

*gravity [1] = -0.0*

*gravity[2] = data.g\_local*

1. Derive omega[3]

*omega[0] = earth\_rate \* cos(input.latitude \*cdr)*

*omega[1] = 0.0*

*omega[2] = earth\_rate \* sin (input.latitude \*cdr)*

1. Derive det

*det = (gravity[0] \* omega[2] ) –*

*(gravity[2] \* omega[0])*

1. Derive X[3][3]

*X[0][0] = omega[2]/det*

*X[0][1] = - gravity[2]/det*

*X[0][2] = 0.0*

*X[1][0] = 0.0*

*X[1][1] = 0.0*

*X[1][2] = -1/det*

*X[2][0] = - omega [0]/det*

*X[2][1] = gravity [0]/det*

*X[2][2] = 0.0*

**Derivation of Y3x3 as**

Y[0][0] = check\_val.ax\* data.g\_local

Y[0][1] = check\_val.ay\* data.g\_local

Y[0][2] = check\_val.az\* data.g\_local

Y[1][0] = check\_val.gx \* cdr/3600.0

Y[1][1] = check\_val.gy \* cdr/3600.0

Y[1][2] = check\_val.gz \* cdr/3600.0

Y[2][0] = (check\_val.ay\* check\_val.gz) –

(check\_val.az \*check\_val.gy)

Y[2][1] = (check\_val.az\* check\_val.gx) –

(check\_val.ax \* check\_val.gz)

Y[2][2] = (check\_val.ax\* check\_val.gy) –

(check\_val.ay \* check\_val.gx)

**DM[3][3] = Matrix multiplication of X3x3 and Y3x3**

**Step2: Angles Computation**

if (fabs(DM[2][1]) > 1.0)

DM[2][1]=DM[2][1]/fabs(DM[2][1]);

*output1 = sin -1(DM[2][1])*

*output2 = tan -1(-DM[2][0]/ DM[2][2])*

*angle\_si = tan -1 (-DM[0][1]/ DM[1][1])*

**Step2: Output3 Computation**

angle\_si = (180 - angle\_si/cdr) \*cdr

A = ppcf;

B = (check\_val.az *+ ppcf\_z)\** check\_val.ay *\*K\_correct*

C = *( (*check\_val.az *+ppcf\_z)\*tan (data.l\_angle \*cdr))*

*output3 = angle\_si – ( A +B +C)*