```
1   using System;
2   using System.Collections.Generic;
3   using static Program.Constants;
4   using System.Threading;
5   using System.Threading.Tasks;
6   using System.Linq;
7   namespace Structures
8   {
9           internal class FundamentalVectors {
10                  // The fundamental vectors of an orbit. Used by
    OrbitalElements
11                  public Vector3 angularMomentum {get; set;}
12                  public Vector3 eccentricity {get; set;}
13                  public Vector3 node {get; set;}
14                  public FundamentalVectors(Vector3 position, Vector3 velocity,
    double stdGrav) {
15                          this.angularMomentum = Vector3.cross
    (position,velocity);
16                          this.node = Vector3.cross
    (Vector3.k,this.angularMomentum);
17                          var mag_r = Vector3.Magnitude(position);
18                          var mag_v = Vector3.Magnitude(velocity);
19                          this.eccentricity = (1/stdGrav)*((Math.Pow(mag_v,2) -
    stdGrav/mag_r)*position - Vector3.dot(position,velocity)*velocity);
20                  }
21                  public override String ToString() {
22                          return $"Angular Momentum: {angularMomentum.ToString
    ()}\nEccentricity: {eccentricity.ToString()}\nNode: {node.ToString()}";
23                  }
24
25          }
26          public class OrbitalElements {
27                  // The six classical orbital elements
28                  public double semilatusrectum {get; set;}
29                  public double eccentricity {get; set;}
30                  protected double _inclination;
31                  public double inclination {
32                          get {
33                                  return _inclination;
34                          } set {
35                                  _inclination = value%Math.PI;
36                          }
37                  }
38                  protected double _ascendingNodeLongitude;
39                  public double ascendingNodeLongitude {
40                          get {
41                                  return _ascendingNodeLongitude;
42                          } set {
43                                  _ascendingNodeLongitude = value%(2*Math.PI);
44                          }
45                  }
46                  protected double _periapsisArgument;
47                  public double periapsisArgument {
48                          get {
49                                  return _periapsisArgument;
50                          } set {
51                                  _periapsisArgument = value%(2*Math.PI);
52                          }
53                  }
54                  protected double _trueAnomaly;
55                  public double trueAnomaly {
56                          get {
57                                  return _trueAnomaly;
58                          } set {
59                                  _trueAnomaly = value%(2*Math.PI);
60                          }
```

```
61                  }
62                  public OrbitalElements() {} // Parameterless constructor for
    serialisation
63                  public OrbitalElements(Vector3 position, Vector3 velocity,
    double stdGrav) {
64                          // stdGrav is the gravitational parameter of the
    parent body
65                          var fVectors = new FundamentalVectors
    (position,velocity,stdGrav);
66                          this.eccentricity = Vector3.Magnitude
    (fVectors.eccentricity);
67                          this.semilatusrectum = Math.Pow(Vector3.Magnitude
    (fVectors.angularMomentum),2)/stdGrav;
68                          this.inclination = Math.Acos
    (fVectors.angularMomentum.z/Vector3.Magnitude(fVectors.angularMomentum)); //
    0 <= i <= 180deg
69                          double cosAscNodeLong = fVectors.node.x/
    Vector3.Magnitude(fVectors.node);
70                          if (fVectors.node.y >= 0) this.ascendingNodeLongitude
    = Math.Acos(cosAscNodeLong);
71                          else this.ascendingNodeLongitude = 2*Math.PI -
    Math.Acos(cosAscNodeLong);
72                          double cosAnomaly = 0;
73                          try {
74                                  double cosPeriArg = Vector3.UnitDot
    (fVectors.node,fVectors.eccentricity);
75                                  if (fVectors.eccentricity.z >= 0)
    this.periapsisArgument = Math.Acos(cosPeriArg);
76                                  else this.periapsisArgument = 2*Math.PI -
    Math.Acos(cosPeriArg);
77                                  cosAnomaly = Vector3.UnitDot
    (fVectors.eccentricity,position);
78                          } catch (DivideByZeroException) {
79                                  // This will be dealt with along with
    extremely small values below
80                          }
81                          if (this.eccentricity < 1e-10 ) {
82                                  // acceptable error, the orbit has no
    periapsis
83                                  this.eccentricity = 0;
84                                  this.periapsisArgument = 0;
85                                  // we assume the periapsis is at the node
    vector
86                                  if (Vector3.Magnitude(fVectors.node) < 1e-10)
    {
87                                          // but if the node vector also does
    not exist we assume the i vector
88                                          cosAnomaly = Vector3.UnitDot
    (Vector3.i,position);
89                                  } else {
90                                          cosAnomaly = Vector3.UnitDot
    (fVectors.node, position);
91                                  }
92                          }
93                          if (Vector3.UnitDot(position,velocity) >= 0)
    this.trueAnomaly = Math.Acos(cosAnomaly);
94                          else this.trueAnomaly = 2*Math.PI - Math.Acos
    (cosAnomaly);
95                          if (Math.Abs(fVectors.angularMomentum.x/
    fVectors.angularMomentum.z) < 1e-10
96                           && Math.Abs(fVectors.angularMomentum.y/
    fVectors.angularMomentum.z) < 1e-10) {
97                                  // acceptable error, the orbit is not inclined
98                                  this.ascendingNodeLongitude = 0;
99                          }
100                 }
```

```
101             }
102     }
```