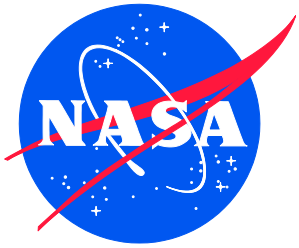
NASA/TM–20240013031/Volume II/Part 2
NESC-RP-23-01853





Expansion of Check-Cases for 6DOF Simulation

Appendix B

Heather M. Koehler/NESC, Matthew J. Hawkins
Marshall Space Flight Center, Huntsville, Alabama

Jason R. Neuhaus
Langley Research Center, Hampton, Virginia

Jesse Couch, Robert L. Bossinger,
Kyler Brazukas, and Trey Fiebelkorn
Adaptive Aerospace Group, Inc., Hampton, Virginia

Edwin Z. Crues
Johnson Space Center, Houston, Texas

Benjamin W. Margolis
Ames Research Center, Moffett Field, California

Tristan D. Hasseler
Jet Propulsion Laboratory, Pasadena, California

Justin T. Ganiban, Nicholas S. Olson
Amentum, Chantilly, Virginia

Zu Qun Li
Johnson Space Center, Houston, Texas

Nathan T. Perreau
Langley Research Center, Hampton, Virginia

Timothy Curry
Dynamic Concepts, Inc, Huntsville, Alabama

Dane W. Erickson
Marshall Space Flight Center, Huntsville, Alabama

Matthew J. Andreini
Analytical Mechanics Associates, Hampton, Virginia

Zhiqiang Zhou
Langley Research Center, Hampton, Virginia

Bruce Jackson
Adaptive Aerospace Group, Inc., Hampton, Virginia

Robert O. Shelton
Johnson Space Center, Houston, Texas

October 2024

NASA STI Program Report Series

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA scientific and technical information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI program provides access to the NTRS Registered and its public interface, the NASA Technical Reports Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA Programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include organizing and publishing research results, distributing specialized research announcements and feeds, providing information desk and personal search support, and enabling data exchange services.

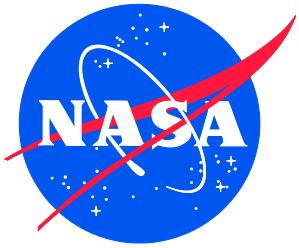
For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at http://www.sti.nasa.gov

- Help desk contact information:

https://www.sti.nasa.gov/sti-contact-form/
and select the "General" help request type.

NASA/TM–20240013031/Volume II/Part 2
NESC-RP-23-01853





Expansion of Check-Cases for 6DOF Simulation

Appendix B

Heather M. Koehler/NESC, Matthew J. Hawkins
Marshall Space Flight Center, Huntsville, Alabama

Jason R. Neuhaus
Langley Research Center, Hampton, Virginia

Jesse Couch, Robert L. Bossinger,
Kyler Brazukas, and Trey Fiebelkorn
Adaptive Aerospace Group, Inc., Hampton, Virginia

Edwin Z. Crues
Johnson Space Center, Houston, Texas

Benjamin W. Margolis
Ames Research Center, Moffett Field, California

Tristan D. Hasseler
Jet Propulsion Laboratory, Pasadena, California

Justin T. Ganiban, Nicholas S. Olson
Amentum, Chantilly, Virginia

Zu Qun Li
Johnson Space Center, Houston, Texas

Nathan T. Perreau
Langley Research Center, Hampton, Virginia

Timothy Curry
Dynamic Concepts, Inc, Huntsville, Alabama

Dane W. Erickson
Marshall Space Flight Center, Huntsville, Alabama

Matthew J. Andreini
Analytical Mechanics Associates, Hampton, Virginia

Zhiqiang Zhou
Langley Research Center, Hampton, Virginia

Bruce Jackson
Adaptive Aerospace Group, Inc., Hampton, Virginia

Robert O. Shelton
Johnson Space Center, Houston, Texas

October 2024

Assessment Team Significant Contributors

James M. Gentile
Johnson Space Center, Houston, Texas

Abhinandan Jain
Jet Propulsion Laboratory, Pasadena, California

Marcus A. Lobbia
Jet Propulsion Laboratory, Pasadena, California

William C. Krolick
CFD Research Corporation, Huntsville Alabama

Alejandro R. Pensado
Analytical Mechanics Associates, Hampton, Virginia

Wei A. Lin
Ames Research Center, Moffett Field, California

John M. Penn
CACI International Inc., Santa Monica, California

Curtis J. Zimmerman
DESE Research, Inc., Huntsville, Alabama

Gregory A. Dukeman
Marshall Space Flight Center, Huntsville, Alabama

Rafael A. Lugo
Langley Research Center, Hampton, Virginia

Juan I. Orphee
Marshall Space Flight Center, Huntsville, Alabama

Jeremy D. Shidner
Langley Research Center, Hampton, Virginia

Everett L. Bolduc
Johnson Space Center, Houston, Texas

The use of trademarks or names of manufacturers in the report is for accurate reporting and does not constitute an official endorsement, either expressed or implied, of such products or manufacturers by the National Aeronautics and Space Administration.

Available from:

NASA STI Program / Mail Stop 050
NASA Langley Research Center
Hampton, VA 23681-2199

NASA Engineering and Safety Center
Technical Assessment Report

Expansion of Check-Cases for 6DOF Simulation
TI-23-01853

Volume II, Part 2: Appendix B

NESC Lead Heather Koehler
Co-Technical Lead Matt Hawkins
Co-Technical Lead Jason Neuhaus

September 17, 2024

Appendix B. Keplerian Propagation

Keplerian Propagator Tool Introduction

For cases of spherical gravity, it is possible to provide a two-body solution without recourse to numerical integration and thus it is accurate to machine precision. Python code for a Keplerian Propagator (propagate.py) which produced a reference trajectory for Case 1 is provided in the appendix. There is also code for generating test cases which was used as an independent verification of the propagator. This is a high-level description of the algorithm employed. The documentation of each function includes implementation details, including equations for each task.

Orbit determination is performed by the function determine_orbit which reads initial position and velocity in MI coordinates and computes the following orbital parameters:

- Vector angular momentum, thus determining the orbital inclination, longitude of ascending node, and latus rectum of the ellipse.
- Vector eccentricity, thus determining the argument of periapsis, semi-major axis and orbital period.

From these values, an orbital coordinate system is derived for propagation and transformation of propagated state back to MI coordinates. An initial eccentric anomaly is computed directly from the initial conditions and the corresponding mean anomaly is given by Kepler's equation. The mean anomaly is proportional to elapsed time, and propagation through time accomplished by converting the advancing mean anomaly to eccentric anomaly which maps to the propagated position and velocity.

The entry point is the menu function which provides options to load initial conditions, view orbit parameters, and propagate the trajectory. The function loadICs determines the orbit and computes the initial mean anomaly.

In addition to predicting position and velocity as a function of time, acceleration, right ascension, and true anomaly are computed as derived quantities. Test code is included to check solutions for internal consistency.

One interesting feature of a Keplerian solver is that unlike a numerical integration, the initial position and velocity are computed based on the orbit determination process, and in general, are not exactly equal to the given initial values. This 'round trip' calculation of orbital elements from initial conditions, back to initial position and velocity serves as a consistency check, and these values are accurate to machine precision.

Keplerian Propagator Tool: README.txt

Contents:

* Code:

- kepler.py: Astronomical constants for radius and gravitation of Earth and Moon, utilities for command line I/O, and Kepler's equation.

- make_IC.py: Creates earth or lunar check cases based on user-provided orbit description.

- propagate.py: Determines Keplerian solution based on initial position and velocity. Displays orbital elements, propagates trajectory for specified duration and sampling interval, and writes a .csv file compatible with minimal standards for posting and comparison.

* Data:

A data directory with three subdirectories, one each for ICs and Trajectories, and a third for reference data. ICs contains initial conditions for several check cases, including lunar case 1, and a couple generated by makeIC. Trajectories contains results of propagating the input check cases.

Notes on Coding:

There is a significant duplication of code between makeIC and propagate. This is due to the similarity of making check cases from orbital elements and determining a Keplerian orbit from an initial state. Of course, all of this code could have been consolidated in a utility module, but the intent was for the orbital parameters to be derived independently from either user input or an initial state.

Usage:

From your terminal in the Kepler directory, enter python <FILE>.py, where <FILE is either makeIC or propagate.

Alternatively, you can fire up the python interpreter and import any of the modules. Use the python help command to find out what's available in each.

File Conventions:

Check case ICs should go in data/ICs. Since you can generate and run check cases for Earth and Moon, you might want to indicate which body is the primary for any new check cases you generate. All output trajectories are saved to data/Trajectories. Again, probably best to indicate the primary in the file name if you plan to work with both Earth and Moon.

Keplerian Propagator Tool: kepler.py

```
import numpy as np
import pathlib as pl

def vector3 (x=0, y=0, z=0) :
  """Creates a 3-vector whose components default to 0
  """
  return np.array ([x,y,z], dtype=float)

# constants
ICDir = 'data/ICs/'
trajDir = 'data/Trajectories/'
primery = -1
mu_earth = 3.986004418e14 # m3s-2
mu_moon = 4.9028001184575496e12 # m3s-2.
radius_earth = 6378.1e3 #m
radius_moon = 1737.400e3 #m
radius, mu = (0,0)

def readline (prompts, scaleFactor=1.0) :
  """ Reads from the command line and returns float
  scaled values for each string in the prompts list
  """
  scaleString = ''
  if scaleFactor == 1000.0 :
    scaleString = ' (in kilometers)'
  elif scaleFactor == 1.0 :
    scaleString = ''
  else :
    scaleString = ' (scale factor: %f)' % scaleFactor

  n =len(prompts)
  promptString = ''
  for (i,p) in enumerate(prompts) :
    promptString+=p
    if i < n-1 :promptString+= ', '
  promptString = 'Enter ' + promptString +scaleString+'>'
  while True :
    try :
      resp = input (promptString)
      splitz = resp.split(',')
      ret = []
      for (i, s) in enumerate (splitz) :
```

```
        ret.append (scaleFactor*float (s))
      break
    except ValueError as v :
      print (splitz[i] + 'is not a number.')
  return ret

def readMenu (choices, default=None) :
  """
  Displays a menu of @choices
  and prompts for an integer input reflecting
  the user selection. Returns the
  selection.
  """
  while True :
    print ("Choices:")
    for cp in enumerate (choices) :
      print ("%d -> %s" % (cp[0], cp[1]))
    defString = ''
    if default != None : defString = "default %d" % default
    inp = input ("Enter selection> " + defString)
    try :
      choice = int(inp or default)
      if choice >= 0 and choice < len(choices) : return choice
      else : print ("Invalid selection %d" % choice)
    except ValueError :
      print ("%s is not a valid choice" % inp)

def setPrimary (p) :
  """
  Sets the primary depending on @p.
  p=0 for Earth; 1 for Moon.
  """
  global mu
  global radius
  if p == 0 : mu, radius = (mu_earth, radius_earth)
  elif p == 1 : mu,radius = (mu_moon, radius_moon)

def choosePrimary () :
  """
  Prompts user to select a primary sets mu and radius accordingly.
  """
  global primary
  choices = ['Earth', 'Moon']
  print ('Select a primary body.')
```

```
  primary = readMenu (choices, 1)
  setPrimary (primary)

def switchPrimary () :
  """Displays the current primary and asks the user if
  they want to switch.
  """
  global primary
  if primary == 0 : print ('Current Primary: Earth')
  elif primary == 1 : print ('Current primary: Moon')
  else : choosePrimary ()
  r = input ('Switch? Default no') or 'n'
  if r.startswith('y') or r.startswith ('Y') : primary = 1-primary
  setPrimary(primary)

def getFilePath (dir, pat, makeNew=False, default = None) :
  """
  Presents a menu of files in @dir matching
  @pat and prompts for a selection. Returns
  the path of the selected file or @dir when
  a new file is to be created.
  """
  p = pl.Path(dir)
  choices = [pth for pth in p.glob (pat) if pth.is_file ()]
  choiceNames = [pth.name for pth in choices]
  if makeNew :
    choiceNames.insert (0, 'New file')
    choices.insert (0, p)
  choice = readMenu (choiceNames, default)
  return choices[choice]

choosePrimary()

# Kepler's Equation
def ecc2mean (e, epsilon) :
  """
  Computes mean anomaly from eccentric anomaly and eccentricity
  """
  return e-epsilon*np.sin(e)

# its derivative
def dMeandEcc(e, epsilon) :
  """
  Computes derivative of mean with respect to eccentric. Takes
```

```
  a parameter, @epsilon, for eccentricity
  """
  return 1-epsilon*np.cos(e)

def d2MeandEcc2(e, epsilon) :
  """
  Computed d2mean/decc2, the second derivative
  of ecc2mean, for eccentric
  anomaly @e and parameter @epsilon.
  """
  return epsilon*np.sin(e)

def mean2ecc (m, epsilon) :
  """
  Uses Newton-Raphson to invert ecc2mean.
  """
  e1 = m
  e2 = 0
  while True :
    e2 = e1 +(m-ecc2mean(e1, epsilon))/dMeandEcc (e1, epsilon)
    if abs (e1-e2) < 1.0e-13 : return e2
    e1 = e2
```

Keplerian Propagator Tool: makeIC.py

```
import numpy as np
import pandas as pd
import pathlib as pl
import kepler as kp
from kepler import dMeandEcc, ecc2mean, getFilePath, mean2ecc, readline,
readMenu, switchPrimary, vector3

cosdeg = lambda x: np.cos(np.deg2rad(x))
sindeg = lambda x: np.sin(np.deg2rad(x))

# orbital elements
a,b, epsilon, f = (0,0, 0, 0)
arg_peri, inc, omega = (0,0, 0)
xAxis = vector3()
yAxis = vector3()
zAxis = vector3()

def orbElem () :
  """
  Prompts user for periapsis, apoapsis, argument of periapsis,
  inclination, and longitude of ascending node, all angles
  entered in degrees. Computes and exports major and minor axes, eccentricity,
focal length and three
  coordinate axes, -- xAxis
  aligned with periapsis, zAxis perpendicular to
  orbital plane, and y = z\times x.
  """

  global a #semi-major axis
  global b # semi-minor axis
  global epsilon # eccentricity
  global f # focal length = epsilon a
  global arg_peri # Argument of periapsis
  global inc # orbital inclination
  global omega # Longitude of ascending node
  global period # Orbital period
# Axes of canonical coordinate system in which the
# orbit is confined to the X-Y plane,
# and the x-axis coincides with periapsis.
  global xAxis
  global yAxis
  global zAxis
```

```
  peri, apo = readline (['Periapsis', 'apoapsis'], 1000.0)
  a = kp.radius + 0.5*(peri+apo)
  f = a-kp.radius-peri
  epsilon = f/a
  b = a*np.sqrt(1-epsilon**2)
  period = 2*np.pi*np.sqrt (a**3/kp.mu)
  inc, omega, arg_peri = readline (['Inclination', 'Lon asc node', 'ArgPeri'])
  tempX = vector3 (cosdeg(omega), sindeg(omega), 0)
  theta = omega+90
  tempY = vector3 (cosdeg(inc)*cosdeg(theta), cosdeg(inc)*sindeg(theta),
sindeg(inc))
  xAxis = cosdeg(arg_peri)*tempX + sindeg(arg_peri)*tempY
  zAxis = np.cross (tempX, tempY)
  yAxis = np.cross (zAxis, xAxis)

def computeState (periodFraction) :
  """
  Returns the time since periapsis,
  position and velocity of an object in a Keplerian
  orbit at the given fractional
  period.
  """
# Mean anomaly
  m = 2*np.pi*periodFraction
# time
  t = period*periodFraction
  dMdT = 2*np.pi/period
# eccentric anomaly
  e = mean2ecc (m, epsilon)
  dEccDt = dMdT/dMeandEcc(e, epsilon)
  x = a*np.cos(e)-f
  y = b*np.sin(e)
  xDot = -a*np.sin(e)*dEccDt
  yDot = b*np.cos(e)*dEccDt
  c = x*yDot - y*xDot
  r = np.sqrt(x**2+y**2)
  h = 0.5*(xDot**2 + yDot**2) - kp.mu/r
  R = x*xAxis + y*yAxis
  V = xDot * xAxis + yDot*yAxis
  return t, R, V, c, h

def makeIC () :
  """
  Creates and saves a pandas DataFrame with
```

```
  initial conditions specified by orbital elements and
  the fraction of the period since periapsis.
  """
  switchPrimary ()
  orbElem ()
  prompt = ['period fraction to locate ICs']
  periodFraction = readline(prompt)[0]
  cols = ['dyn time', 'time', 'x', 'y', 'z', 'xDot', 'yDot', 'zDot' ,'c', 'h']
  t, R, V, c, h = computeState (periodFraction)
  l = [0, t] + list(R) + list (V) + [c, h]
  df = pd.DataFrame([l], columns=cols)
  p = getFilePath (kp.ICDir, '*.csv', True, 0)
  if p.is_dir () :
    fileName = input("Enter file name")
    outputPath = pl.Path(kp.ICDir+fileName)
  else :
    outputPath = p
  df.to_csv(outputPath, index = False)

def menu () :
  """
  Command line user interface.
  """
  choiceDict = {'Create ICs' : makeIC,
  'Quit' : 1}
  choices = list (choiceDict.keys())
  while True :
    c =readMenu (choices, 0)
    if choiceDict[choices[c]] == 1 : break
    choiceDict[choices[c]]()

if __name__ == '__main__' : menu()
```

Keplerian Propagator Tool: propagate.py

```
"""
This module performs a two-body propagation from an
initial position vector R and velocity vector V.
"""
from math import acos, asin, atan2
import numpy as np
import numpy.linalg as npl
import pandas as pd
import kepler as kp
from kepler import d2MeandEcc2, dMeandEcc, ecc2mean, getFilePath, mean2ecc,
readline, readMenu, switchPrimary, vector3

a = 0.0
ascNode = vector3()
b = 0.0
C = vector3()
c = 0.0
epsilon = 0.0
f = 0.0
inc = 0.0
mDot = 0.0
omega = 0.0
p = 0.0
period = 0.0
xAxis = vector3()
yAxis = vector3()
zAxis = vector3()

def determineOrbit (filename) :
  """
  This method determines the orbital path and a coordinate
  system where all orbital motion will
  occur in the X-Y plane. Since we know angular momentum is conserved and
  the angular momentum, C, will be perpendicular
  to the orbital motion, the Z-axis of our system
  is the unit vector C/|C|

  The X-axis should point to the periapsis of the orbit.
  There is a vector constant of motion, Epsilon, which
  is obtained by integrating the identity
  d/dt(R/|R|) = AXC/mu, therefore
  VXC/mu = R/|R| + Epsilon
```

```
  The norm of Epsilon, epsilon, turns
  out to be the eccentricity of the orbit, and
  the vector Epsilon points to the periapsis, thus
  our X-axis is Epsilon/|epsilon.
  The Y-axis will then be the cross product of Z and X.
  """
  global a
  global ascNode
  global b
  global C
  global c
  global epsilon
  global f
  global inc
  global mDot
  global omega
  global p
  global period
  global xAxis
  global yAxis
  global zAxis

# Read initial conditions
  df = pd.read_csv (filename)
  R0 = vector3 (df.at[(0,'x')], df.at[(0,'y')], df.at[(0,'z')])
  V0 = vector3 (df.at[(0,'xDot')], df.at[(0,'yDot')], df.at[(0,'zDot')])
  C = np.cross (R0, V0)
  c = npl.norm(C)
  r0 = npl.norm(R0)
  zAxis = C/c
  inc = acos (zAxis[2])
  if inc > 1.0e-9 : omega = atan2 (zAxis[1], zAxis[0])+np.pi/2
  else: omega = 0.0
  ascNode = vector3(np.cos(omega), np.sin(omega))
  Epsilon = np.cross (V0, C)/kp.mu - R0/r0
  epsilon = npl.norm(Epsilon)
  if epsilon < 1.0e-9 :
    xAxis = ascNode
  else : xAxis = Epsilon/epsilon
  if epsilon >= 1.0 :
    raise ValueError ('Open orbit. Case not processed')
  yAxis = np.cross(zAxis, xAxis)
  p = c**2/kp.mu
  a = p/(1-epsilon**2)
```

```
  f = a*epsilon
  b = a*np.sqrt(1-epsilon**2)
  period = 2.0*np.pi*np.sqrt (a**3/kp.mu)
  mDot = 2*np.pi/period
  return R0, V0

def getPeri () :
  """
  Returns the periapsis of the orbit in km.
  """
  return 0.001*(a-f-kp.radius)

def getApo () :
  """
  Returns apoapsis of orbit in km.
  """
  return 0.001*(a+f-kp.radius)

def eccDerivs (m) :
  """
  Calculates derivatives of the
  eccentric anomaly with respect
  to the mean anomaly @m.
  The suffix Prime indicates differentiation with respect to m
  """
  e = mean2ecc (m, epsilon)
  ePrime = 1/dMeandEcc(e, epsilon)
  ePrime2 = -d2MeandEcc2(e, epsilon)/dMeandEcc(e, epsilon)**3
  return e, ePrime, ePrime2

def meanAnomaly (x, y) :
  """
  Computes mean anomaly corresponding to position
  (x, y) on planar orbit.
  """
  if x**2+y**2 < 1.0e-24 :
    raise ValueError ("(%f, %f) is not a valid  orbital position" % (x, y))
  e = atan2 (y/b, (x+f)/a)
  return ecc2mean(e, epsilon)


def getState (t) :
  """
  Returns 3-dimensional position,
```

```
  velocity, and acceleration corresponding
  to time since periapsis @t.
  Notes:
  The suffix "Dot" indicates time
  derivative.
  Equations:
  x = a*np.cos(e)-f
  y = b*np.sin(e)
  eDot = de/dm*mDot
  eDot2 = d2m/de2*mDot**2 (mDot is constant)
  xDot = -a*np.sin(e)*eDot
  yDot = b*np.cos(e)*eDot
  xDot2 = -a*np.cos(t)*eDot**2 - a*np.sin(e)*eDot2
  yDot2 = -a*np.cos(e)*eDot**2 - b*np.sin(e)*eDot2
  R = x*xAxis + y*yAxis
  V = xDot*xAxis + yDot*yAxis
  A = xDot2*xAxis + yDot2*yAxis
  """
  m = t*mDot
  e, e1, e2 = eccDerivs (m)
  eDot = e1*mDot
  eDot2 = e2*mDot**2
# efficiency hack
  cosE, sinE = np.cos(e), np.sin(e)
  x = a*cosE-f
  y = b*sinE
  xDot = -a*sinE*eDot
  yDot = b*cosE*eDot
  xDot2 = -a*cosE*eDot**2 -a*sinE*eDot2
  yDot2 = -b*sinE*eDot**2 + b*cosE*eDot2
#  return x, y, xDot, yDot, xDot2, yDot2
  R = x*xAxis + y*yAxis
  V = xDot*xAxis + yDot*yAxis
  A = xDot2*xAxis + yDot2*yAxis
  return R, V, A

# Test code
def energy (R, V) :
  """
  Returns total energy of system which must
  be conserved -- constant in time
  """
  r = npl.norm (R)
  v = npl.norm (V)
```

```
  return v**2/2 - kp.mu/r

def deltaA (R, A) :
  """
  Computes acceleration -muR/r**3
  from the EOM and compares to computed acceleration A.
  """
  r = npl.norm (R)
  aEOM = -kp.mu*R/r**3
  return aEOM-A

# Baseline values
def loadICs () :
  """
  Loads initial conditions from the chosen .csv file.
"""
  global R0
  global V0
  global A0
  global C0
  global x0
  global y0
  global h0
  global timeSincePeri
  switchPrimary()
  path = getFilePath (kp.ICDir, '*.csv')
  R0, V0 = determineOrbit (path)
  x0 = np.dot(R0, xAxis)
  y0 = np.dot (R0, yAxis)
  timeSincePeri = meanAnomaly(x0, y0)/mDot
  h0 = energy(R0, V0)
  C0 = np.cross (R0, V0)
  A0 = deltaA(R0, vector3())

def testOutput (t) :
  """
  Computes a solution for time t and
  checks for conservation of
  energy and angular momentum. Also compares
  the acceleration from the computed solution
  with the acceleration derived
  from the equation of motion.
  """
  R, V, A = getState(t)
```

```
  return h0-energy(R,V), C0-np.cross(R,V), deltaA (R,A)

def simLoop (duration, step, tolerance=1.0e-10) :
  """
  Returns a list of sim outputs over @duration seconds
  at intervals of @step seconds.
  """
  output = []
  for t in np.arange (0.0, duration+0.5*step, step) :
    R,V, A = getState(t+timeSincePeri)
    relDeltaH = abs(h0-energy(R,V))/abs(h0)
    relDeltaC = npl.norm(C0-np.cross(R,V))/npl.norm(C0)
    relDeltaA = npl.norm(deltaA(R, A))/(kp.mu/npl.norm(R)**2)
    if max (relDeltaH, relDeltaC, relDeltaA) > tolerance :
      print (t, relDeltaH, relDeltaC, relDeltaA)
    output.append ([t] + list(R) + list (V) + list (A))
  return np.array(output)

def doOutput () :
  """
  Creates and writes a pd.DataFrame containing
  agreed headers and data.
  """
  dur, stp = readline(['Duration, output frequency'])
  a = simLoop(dur, stp)
  outputPath = getFilePath (kp.trajDir, '*.csv', True, 0)
  cols = ['elapsedTime_s', # 'j2000UtcTime_s', 'j2000TtTime_s',
  'miPosition_m_X', 'miPosition_m_Y', 'miPosition_m_Z',
  'miVelocity_m_s_X', 'miVelocity_m_s_Y', 'miVelocity_m_s_Z',
  'miAccel_m_s2_X', 'miAccel_m_s2_Y',
  'miAccel_m_s2_Z']
  if outputPath.is_dir() :
    newName = input ('Enter name for new file> ')
    outputPath = outputPath/newName
  pd.DataFrame(a, columns=cols).to_csv(outputPath, index=False)

def showOrbElems () :
  """
  Displays classical orbital
  elements plus apoapsis,
  periapsis, period and semi-minor axis.
  """
  print ("          Orbital Elements\n")
  print ("Semi-major: %f (m)" %a)
```

```
  print ("Semi-minor %f (m)" %b)
  print ("Eccentricity %f" % epsilon)
  print ("Inclination %f (deg.)" %np.rad2deg(inc))
  print ("Longitude of ascending node %f (deg.)" % np.rad2deg(omega))
  s = np.dot(zAxis, np.cross(ascNode, xAxis))
  c = np.dot(ascNode, xAxis)
  argPeri = atan2(s,c)
  print ("Argument of periapsis %f (deg.)" % np.rad2deg(argPeri))
  print ("Period %f sec" % period)
  print ("Minimum altitude %f (km)" % getPeri())
  print ("Maximum altitude %f (km)" % getApo ())

def menu () :
  """
  Command line user interface.
  """
  commandDict = {
  'Load initial conditions' : loadICs,
  'Show orbit' : showOrbElems,
  'Make trajectory' : doOutput,
  'Quit' : -1}
  choices = list(commandDict.keys())
  while True:
    choice = readMenu (choices, 0)
    c = commandDict [choices[choice]]
    if c == -1 : break
    c()

if __name__ == '__main__' : menu ()
```