DISTRIBUTED PARAMETER SYSTEM FOR OPTIMIZATION AND FILTERING IN ASTRODYNAMIC SOFTWARE

Jonathon Smith

Mission Design & Navigation Software Group
Jet Propulsion Laboratory, California Institute of Technology

©2016 California Institute of Technology. Government sponsorship acknowledged.

1. Software skills are worth investing in
2. Object-oriented > procedural for astrodynamic parameters
3. JPL's MONTE demonstrates the power of this approach

©2016 California Institute of Technology. Government sponsorship acknowledged.

“Software isn’t our career, but it is our job”

- Juan Arrieta, JPL

©2016 California Institute of Technology. Government sponsorship acknowledged.

Save yourself time

Demonstrate excellence to your peers

We enjoy things we're good at

©2016 California Institute of Technology. Government sponsorship acknowledged.

$$\boxed{b} = \boxed{w \cdot p}$$

$$\boxed{a} = \log b + \boxed{c}$$

$$\boxed{s} = a^2 + 1$$

©2016 California Institute of Technology. Government sponsorship acknowledged.

PROCEDURAL

shorter

explicit

brittle

```
# SOURCE CODE
def get_values(w, p, c):
    """Calculate parameter values """
    b = w * p
    a = log(b) + c
    s = a**2 + 1
    return b, a, s

def get_partials(w, p, c):
    """Calculate parameter partials"""
    dparams = np.zeros([3,3])
    # db/dw
    dparams[0,0] = p
    # db/dp
    dparams[0,1] = w
    # da/dw
    dparams[1,0] = 1./w
    # da/dp
    dparams[1,1] = 1./p
    # da/dc
    dparams[1,2] = 1.
    # ds/dw
    dparams[2,0] = 2*(log(w*p)+c)/w
    # ds/dp
    dparams[2,1] = 2.*(log(w*p)+c)/p
    # ds/dc
    dparams[2,2] = 2.*(log(w*p)+c)

    return dparams
```

Vs.

OBJECT-ORIENTED

longer

abstracted

flexible

```
# SOURCE CODE
class A(object):
    """Compute value and partials for b """
    def __init__(self, w, p):
        self._w, self._p = w, p

    def get_value( self ):
        return self._w * self._p

    def get_partials(self, params):
        dparams = np.zeros([len(params)])
        if "w" in params: dparams[params.index("w")] = self._p
        if "p" in params: dparams[params.index("p")] = self._w
        return dparams

class A(object):
    """Compute value and partials for a """
    def __init__(self, b, c):
        self._b, self._c = b, c

    def get_value( self ):
        return log(self._b.get_value()) + self._c

    def get_partials(self, params):
        dc_dparam = np.zeros([len(params)])
        if "c" in params: dc_dparam[params.index("c")] = 1.
        b = self._b.get_value()
        db_dparam = self._b.get_partials(params)
        return (1./b)*db_dparam + dc_dparam

class S(object):
    """Compute value and partials for s """
    def __init__(self, a):
        self._a = a

    def get_value( self ):
        return self._a.get_value()**2 + 1

    def get_partials(self, params):
        a = self._a.get_value()
        da_dparam = self._a.get_partials(params)
        return 2*a*da_dparam
```


©2016 California Institute of Technology. Government sponsorship acknowledged.

$$\boxed{b} = \boxed{w \cdot p}$$

$$\boxed{a} = \log b + \boxed{c}$$

$$\boxed{s} = a^2 + 1$$

$$\frac{\partial s}{\partial w} = ?$$

©2016 California Institute of Technology. Government sponsorship acknowledged.

$$\boxed{b} = \boxed{w \cdot p}$$

$$\boxed{a} = \log b + \boxed{c}$$

$$\boxed{s} = a^2 + 1$$

$$\frac{\partial s}{\partial w} = ?$$

$$\frac{\partial b}{\partial w} = p$$

$$\frac{\partial a}{\partial w} = \frac{1}{b} \cdot \frac{\partial b}{\partial w}$$

$$\boxed{\frac{\partial s}{\partial w} = 2 \cdot a \cdot \frac{\partial a}{\partial w}}$$

©2016 California Institute of Technology. Government sponsorship acknowledged.

$$\frac{\partial b}{\partial w} = p$$

$$\frac{\partial a}{\partial w} = \frac{1}{b} \cdot \frac{\partial b}{\partial w}$$

$$\frac{\partial s}{\partial w} = 2 \cdot a \cdot \frac{\partial a}{\partial w}$$

```python
# SOURCE CODE
class B(object):
    """Compute value and partials for b """
    def __init__(self, w, p):
        self._w, self._p = w, p

    def get_value( self ):
        return self._w * self._p

    def get_partials(self, params):
        dparams = np.zeros([len(params)])
        if "w" in params: dparams[params.index("w")] = self._p
        if "p" in params: dparams[params.index("p")] = self._w
        return dparams

class A(object):
    """Compute value and partials for a """
    def __init__(self, b, c):
        self._b, self._c = b, c

    def get_value( self ):
        return log(self._b.get_value()) + self._c

    def get_partials(self, params):
        dc_dparam = np.zeros([len(params)])
        if "c" in params: dc_dparam[params.index("c")] = 1.
        b = self._b.get_value()
        db_dparam = self._b.get_partials(params)
        return (1./b)*db_dparam + dc_dparam

class S(object):
    """Compute value and partials for s """
    def __init__(self, a):
        self._a = a

    def get_value( self ):
        return self._a.get_value()**2 + 1

    def get_partials(self, params):
        a = self._a.get_value()
        da_dparam = self._a.get_partials(params)
        return 2*a*da_dparam
```


©2016 California Institute of Technology. Government sponsorship acknowledged.

MISSION ANALYSIS OPERATIONS & NAVIGATION TOOLKIT ENVIRONMENT

Astrodynamic Platform

Python Library

Mission Operations

Mission Design & Analysis

©2016 California Institute of Technology. Government sponsorship acknowledged.

```python
import Monte as M

# ==========================================================================
# CALCULATE LEONARD NIMOY'S AGE AT THE FIRST AIRING OF    STAR TREK
# ==========================================================================

# Leonard Nimoy's Birthday
nimoyBday = M.Epoch( "26-MAR-1931 ET" )

# Date the first episode of Star Trek aired
startrekAirdate = M.Epoch( "8-SEP-1966 ET" )

# Calculate Age
nimoyAge = startrekAirdate - nimoyBday

# Get integer years and days
years = int( nimoyAge.years() )
days = int( ( nimoyAge.years() - years ) * 365 )

# Format and print output
msg  = "\nLeonard Nimoy was {0:d} years, {1:d} days old ".format( years, days )
msg += "when the first episode of Star Trek aired!"
print msg
```


©2016 California Institute of Technology. Government sponsorship acknowledged.

Q: What is the state transition matrix from time t1 to t2?

```
In [1]> import Monte as M
In [2]> boa = M.BoaLoad( "outputs/traj.boa" )
In [3]> tset = M.TrajSetBoa.read( boa )
In [4]> tset.getAll()
Out[4]: ['Cassini', 'Saturn Barycenter']

In [5]> tset.totalInterval( "Cassini" )
Out[5]:
TimeInterval(
   [ '25-JAN-2014 19:00:00.0000 ET',
     '10-MAR-2014 17:00:00.0000 ET' ],
)

In [6]> q = M.ParamList( boa, ["State / Cassini / Saturn Barycenter / EME2000 / STATE"] )
In [7]> q.setRefEpoch( '01-FEB-2014 ET' )
In [8]> q.transitionMatrix( "02-FEB-2014 ET", q )
Out[8]:
```

	0	1	2	3	4	5
0:	1.052813253513873e+00	5.243929170264378e-02	3.593690539447821e-02	8.794948297397887e+04	1.721551968413470e+03	9.538794498126867e+02
1:	5.272872199474303e-02	9.821566272265407e-01	1.884829275364848e-02	1.726105136967522e+03	8.599139804204683e+04	5.476860100105275e+02
2:	3.569778421177078e-02	1.861636425447536e-02	9.672403134187039e-01	9.501324002142400e+02	5.440468001739333e+02	8.530060992254554e+04
3:	1.273533914463510e-06	1.398191460483833e-06	7.695555164601427e-07	1.054813604027267e+00	6.788530717881826e-02	2.966788700185453e-02
4:	1.416321580913636e-06	-2.928061166738721e-07	4.427086457913854e-07	6.822833255551797e-02	9.912460141655987e-01	1.867241562156544e-02
5:	7.546715380462610e-07	4.282303725991347e-07	-8.720670557237546e-07	2.938759646195864e-02	1.839939539877240e-02	9.565053779689481e-01

©2016 California Institute of Technology. Government sponsorship acknowledged.

Q: What are the partials of my B-Plane encounter with respect to a maneuver?

```
In [1]> import Monte as M
In [2]> import mpy.io.data as defaultData
In [3]> boa = defaultData.load( [
   ...    "body",
   ...    "ephem/planet/de405",
   ...    "./inputs/sat365.boa",
   ...    "./outputs/traj.boa"
   ...    ])
In [4]> otm372_params = [
   ...     "Finite Burn / Cassini / OTM372 / DELTA V",
   ...     "Finite Burn / Cassini / OTM372 / RA[0]",
   ...     "Finite Burn / Cassini / OTM372 / DEC[0]",
   ...     ]
In [5]> controls = M.ParamList( boa, otm372_params )
In [6]> t99_time = M.Epoch("06-MAR-2014 16:27:54 ET")
In [7]> t99_encounter = [
   ...    "State / Cassini / Titan / EME2000 / Conic.bDotR",
   ...    "State / Cassini / Titan / EME2000 / Conic.bDotT",
   ...    "State / Cassini / Titan / EME2000 / Conic.linearizedTOF",
   ...    ]
In [8]> targets = M.ParamList(boa, t99_encounter)
In [9]> targets.transitionMatrix( t99_time, controls )
Out[9]:
            0                      1                      2
0: -1.134578515421030e+05  1.045272917593320e+03 -1.051150455489717e+03
1: -3.164093127031080e+06  2.755189381590511e+03 -2.318611375964326e+03
2:  4.593808333929498e+05 -2.987261692942101e+02  4.234231081899590e+02
```


©2016 California Institute of Technology. Government sponsorship acknowledged.

Q: What are the computed partials of my measurements with respect to estimated parameters?

```
In [1]> import Monte as M
In [2]> cmpFile = M.BoaLoad("outputs/computed.boa")
In [3]> cmpMgr = M.ComputedMsrRead(cmpFile)
In [4]> point = cmpMgr[0]
In [5]> resid = point.residVecDq()[0]
In [6]> resid.dq()
Out[6]:
           0                    1                    2                    3                    4                    5
        6                    7                    8                    9                   10                   11
     12                   13                   14                   15                   16                   17
  18                   19                   20                   21                   22                   23
24                   25                   26                   27                   28                   29
                  31                   32                   33                   34                   35
               37                   38                   39                   40                   41                   42
            43                   44                   45                   46                   47                   48
         49                   50                   51
0:  4.231231334983931e-03  8.456681539996680e-03 -9.730789765853842e-04 -3.835289566069096e+04 -4.615652194847912e+04 -1.7385558874124
 8.183059402396779e-06  0.000000000000000e+00  0.000000000000000e+00  0.000000000000000e+00  0.000000000000000e+00  0.000000000000000e
000000000000000e+00  0.000000000000000e+00  0.000000000000000e+00  0.000000000000000e+00  0.000000000000000e+00  0.000000000000000e+00
000000000000e+00  0.000000000000000e+00  0.000000000000000e+00  0.000000000000000e+00  0.000000000000000e+00  0.000000000000000e+00  0
000000000e+00  0.000000000000000e+00  0.000000000000000e+00  0.000000000000000e+00  0.000000000000000e+00  0.000000000000000e+00  0.00
000000e+00  0.000000000000000e+00  0.000000000000000e+00  0.000000000000000e+00  0.000000000000000e+00  0.000000000000000e+00  0.00000
000e+00  0.000000000000000e+00  0.000000000000000e+00  0.000000000000000e+00  0.000000000000000e+00  0.000000000000000e+00  0.00000000
e+00  0.000000000000000e+00  0.000000000000000e+00  0.000000000000000e+00  0.000000000000000e+00  0.000000000000000e+00  0.00000000000
0 -1.923039953537703e+06  1.066885983483285e+06  3.472674430663818e+09
```


©2016 California Institute of Technology. Government sponsorship acknowledged.

Shameless Plug:

MONTE is available for licensing to researchers and universities.

Email **mdn_software@jpl.nasa.gov** for information.

©2016 California Institute of Technology. Government sponsorship acknowledged.

Thank you, any questions?

©2016 California Institute of Technology. Government sponsorship acknowledged.