HigherOrder

May 31, 2017

1 Higher-Order Equation of Motion

Our goal is to solve the equation of motion for the high-order acceleration f, a given higher-order Lagrangian L.

Related worksheet: https://cloud.sagemath.com/projects/b04b5777-e269-4c8f-a4b8-b21dbe1c93c6/files/Two%20Equations%20of%20Lagrange.sagews

Calculations are done using SageManifolds.

```
In [1]: %display typeset
        from sage.manifolds.utilities import exterior_derivative as d
        def ev(N): return (lambda x: N.contract(x))
```

Variables

```
In [2]: M = Manifold(5,'M')
        coord.<t, x, v, a, s> = M.chart()
```

Vectors (partial derivatives)

```
In [3]: [Dt,Dx,Dv,Da,Ds] = coord.frame()
```

Forms

```
In [4]: [dt,dx,dv,da,ds] = coord.coframe()
```

1.1 General Lagrangian

```
In [5]: L = M.scalar_field(function('L')(*list(coord))); L.display()
```

1.2 Kinematics

f is an unknown function that is the analogue of acceleration for a higher-order equation of motion.

```
In [6]: f = M.scalar_field(function('f')(*list(coord))); f.display()
```

```
In [7]: N = Dt + v*Dx + a*Dv + s*Da + f*Ds; N.display()
```

The Equation of Lagrange can be defined with the aid of the following auxillary fields

```
In [8]: r=Ds(L); r.display()
```

```
In [9]: q=Da(L)-N(r); q.display()
```

```
In [10]: p=Dv(L)-N(q); p.display()
```

```
In [11]: eq1 = (N(p)-Dx(L)); eq1.display()
```

Number of terms in this expression:

```
In [12]: len(eq1.expr().expand())
```

We want to solve this equation (eq1 = 0) for f in terms of L.
The equation can also be written in the following form:

```
In [13]: eq2 = N(Dv(L)) - N(N(Da(L))) + N(N(N(Ds(L)))) - Dx(L)
```

```
In [14]: eq1 == eq2
```

But we would prefer to write it as a polynomial in $N(N(f))$, $N(f)$, and powers of f. If we write f instead of $N(Ds(L))$, $N(Da(L))$ and $N(Dv(L))$ above then we get new coefficients and terms in powers of f.

```
In [15]: N(N(f)).display()
```

Terms 2^{nd} order in f arise from the Liebniz rule applied to the differential operator N^2 on the term $Ds\ f$ in $N(Ds(L))$. For example

```
In [16]: t2 = eq1.expr().coefficient(diff(f.expr(),t,t))*diff(f.expr(),t,t); M.scalar_field(t2).
```

The necessary coefficient can be read from the above.

```
In [17]: c2 = (Ds(Ds(L))*N(N(f))); c2.display()
```

After removing the term $c2$ above, the remaining terms are at most first-order in f.

```
In [18]: eq1a = eq1-c2; eq1a.display()
```

Similarly the first-order terms arise from N applied to the term $Ds\ f$ in $N(Da(L))$.

```
In [19]: N(f).display()
```

For example:

```
In [20]: t1 = eq1a.expr().coefficient(diff(f.expr(),t))*diff(f.expr(),t); M.scalar_field(t1).dis
```

```
In [21]: x1 = eq1a.expr().coefficient(diff(f.expr(),x))*diff(f.expr(),x); M.scalar_field(x1).dis
```

```
In [22]: v1 = eq1a.expr().coefficient(diff(f.expr(),v))*diff(f.expr(),v); M.scalar_field(v1).dis
```

```
In [23]: a1 = eq1a.expr().coefficient(diff(f.expr(),a))*diff(f.expr(),a); M.scalar_field(a1).dis
```

```
In [24]: s1 = eq1a.expr().coefficient(diff(f.expr(),s))*diff(f.expr(),s); M.scalar_field(s1).dis
```

Liebniz rule applies N to these coefficients.

```
In [25]: c1 = 3*N(Ds(Ds(L)))*N(f); c1.display()
```

```
In [26]: bool(c1.expr()==t1+x1+v1+a1+s1)
```

Removing the 1^{st} order terms:

```
In [27]: eq1b = eq1a-c1; eq1b.display()
```

Only terms algebraic in f remain. The number of terms is:

```
In [28]: len(eq1b.expr().expand())
```

The term of highest degree comes from N^3 applied to $Ds(L)$:

```
In [29]: t0 = eq1b.expr().coefficient(f.expr()^3)*f.expr()^3; M.scalar_field(t0.expand()).displa
```

```
In [30]: cf3 = Ds(Ds(Ds(Ds(L))))*f^3; cf3.display()
```

Removing the 3^{rd} degree term:

```
In [31]: eq1c = eq1b-cf3; eq1c.display()
```

leaves terms of at most 2^{nd} degree. These 2^{nd} degree terms come for N^2 applied to $Da(L)$,

```
In [32]: t0 = eq1c.expr().coefficient(f.expr()^2)*f.expr()^2; M.scalar_field(t0.expand()).displa
```

```
In [33]: cf2 = N(Ds(Da(L)))*f; cf2.display()
```

```
In [34]: cf2 = Ds(Ds(Da(L)))*f^2; cf2.display()
```

We cannot solve this for `f` algebraically since `f` appears as derivatives.

1.3 Lagrangian linear in s.

```
In [35]: ll=list(coord);ll.remove(s);ll
```

```
In [36]: Ll = M.scalar_field(function('L0')(*ll)) + s * M.scalar_field(function('L1')(*ll)); Ll.
```

```
In [37]: solve(eq1.expr().substitute_function(L.expr().operator(),Ll.expr().function(*list(coord
```

1.4 Second Equation of Lagrange

```
In [38]: (Dt(L)-(N(L)-(p*a+q*s+r*f)-(v*N(p)+a*N(q)+s*N(r)))).display()
```

Multiplying by v

```
In [39]: v*(N(p)-Dx(L)) == (Dt(L)-(N(L)-(p*a+q*s+r*f)-(v*N(p)+a*N(q)+s*N(r))))
```

1.5 Checking the calculations from the paper

```
In [40]: L = M.scalar_field(function('L')(*list(coord)))
         p = M.scalar_field(function('p')(*list(coord)))
         q = M.scalar_field(function('q')(*list(coord)))
         r = M.scalar_field(function('r')(*list(coord)))
         t=M.scalar_field(t)
         x=M.scalar_field(x)
         v=M.scalar_field(v)
         a=M.scalar_field(a)
         s=M.scalar_field(s)
```

Action differential Form

```
In [41]: alpha = L*dt + p*(dx-v*dt) + q*(dv-a*dt) + r*(da-s*dt)
         alpha.display()

In [42]: alpha == L*dt+p*dx+q*dv+r*da-(p*v+q*a+r*s)*dt

In [43]: d(alpha).display()

In [44]: d(alpha) == d(L).wedge(dt) + d(p).wedge(dx) + d(q).wedge(dv) + d(r).wedge(da) - d(p*v +

In [45]: ev(alpha)(N)==L

In [46]: Omega = -(p*dx+q*dv+r*da).wedge(d(t)); Omega.display()

In [47]: alpha == L*dt + ev(N)(Omega)
```

Equation of Motion ($E = 0$)

```
In [48]: E = ev(N)(d(alpha))
         E.display()
```

Rewriting it in various ways.

```
In [49]: E == N(L)*dt - N(t) * d(L) + N(p)*dx - N(x)*d(p) + N(q)*dv - N(v)*d(q) + N(r)*da - N(a)

In [50]: E == N(L)*dt - d(L) + N(p)*dx - v*d(p) + N(q)*dv - a*d(q) + N(r)*da - s*d(r) - N(p*v+q*

In [51]: E == N(L-p*v-q*a-r*s)*dt - d(L - p*v - q*a- r*s) - v*d(p) - a*d(q) - s*d(r) + N(p)*dx +

In [52]: d(p*v) == v*d(p) + p*d(v)

In [53]: d(q*a) == a*d(q) + q*d(a)

In [54]: d(r*s) == s*d(r) + r*d(s)

In [55]: E == N(L - p*v - q*a - r*s)*dt - d(L) + p*dv + q*da + r*ds + N(p)*dx + N(q)*dv + N(r)*d

In [56]: N(p*v) == p*N(v) + v*N(p)
```

```
In [57]: E == N(L)*dt - (p*a+q*s+r*f)*dt -(v*N(p) + a*N(q) + s*N(r) )*dt - d(L) + p*dv + q*da +

In [58]: E == ( N(L) - (p*a + q*s + r*f) - (v*N(p) + a*N(q) + s*N(r)))*dt + N(p)*dx + (N(q)+p)*d

In [59]: d(L) == Dt(L)*dt + Dx(L)*dx + Dv(L)*dv + Da(L)*da + Ds(L)*ds

In [60]: r=Ds(L); r.display()

In [61]: q=Da(L)-N(r); q.display()

In [62]: p=Dv(L)-N(q); p.display()
```

1.6 For example: The Schiff and Poirier Lagrangian

```
In [63]: hbar = var('hbar',latex_name='\hbar')
         m = var('m')
         V = M.scalar_field(function('V')(var('x')))
         Lp = 1/2*m*v^2  - V - hbar^2/4/m*(s/v^3-5/2*a^2/v^4); Lp.display()

In [64]: Lp1 = Ds(Lp); Lp1.display()

In [65]: Lp0 = Lp - s * Lp1; Lp0.display()

In [66]: Lp == Lp0 + s*Lp1

In [67]: eq1p = eq1.expr().substitute_function(L.expr().operator(),Lp.expr().function(*list(coor

In [68]: solve(eq1p,f.expr())

In [0]:
```