Constrained optimization with Lagrange multipliers and autograd

Posted November 03, 2018 at 09:39 AM | categories: optimization, autograd | tags:

Constrained optimization is common in engineering problems solving. A prototypical example (from Greenberg, Advanced Engineering Mathematics, Ch 13.7) is to find the point on a plane that is closest to the origin. The plane is defined by the equation $2x - y + z = 3$, and we seek to minimize $x^2 + y^2 + z^2$ subject to the equality constraint defined by the plane. `scipy.optimize.minimize` provides a pretty convenient interface to solve a problem like this, ans shown here.

```
import numpy as np
from scipy.optimize import minimize

def objective(X):
    x, y, z = X
    return x**2 + y**2 + z**2

def eq(X):
    x, y, z = X
    return 2 * x - y + z - 3

sol = minimize(objective, [1, -0.5, 0.5], constraints={'type': 'eq', 'fun': eq})
sol
```

```
     fun: 1.5
     jac: array([ 2.00000001, -0.99999999,  1.00000001])
 message: 'Optimization terminated successfully.'
    nfev: 5
     nit: 1
    njev: 1
  status: 0
 success: True
       x: array([ 1. , -0.5,  0.5])
```

I like the minimize function a lot, although I am not crazy for how the constraints are provided. The alternative used to be that there was an argument for equality constraints and another for inequality constraints. Analogous to `scipy.integrate.solve_ivp` event functions, they could have also used function attributes.

Sometimes, it might be desirable to go back to basics though, especially if you are unaware of the `minimize` function or perhaps suspect it is not working right and want an independent answer. Next we look at how to construct this constrained optimization problem using Lagrange multipliers. This converts the problem into an augmented unconstrained optimization problem we can use `fsolve` on. The gist of this method is we formulate a new problem:

$$F(X) = f(X) - \lambda g(X)$$

and then solve the simultaneous resulting equations:

$F_x(X) = F_y(X) = F_z(X) = g(X) = 0$ where F_x is the derivative of $f*$ with respect to x, and $g(X)$ is the equality constraint written so it is equal to zero. Since we end up with four equations that equal zero, we can simply use fsolve to get the solution. Many years ago I used a finite difference approximation to the derivatives. Today we use autograd to get the desired derivatives. Here it is.

```
import autograd.numpy as np
from autograd import grad

def F(L):
    'Augmented Lagrange function'
    x, y, z, _lambda = L
    return objective([x, y, z]) - _lambda * eq([x, y, z])

# Gradients of the Lagrange function
dfdL = grad(F, 0)

# Find L that returns all zeros in this function.
def obj(L):
    x, y, z, _lambda = L
    dFdx, dFdy, dFdz, dFdlam = dfdL(L)
    return [dFdx, dFdy, dFdz, eq([x, y, z])]

from scipy.optimize import fsolve
x, y, z, _lam = fsolve(obj, [0.0, 0.0, 0.0, 1.0])
print(f'The answer is at {x, y, z}')
```

```
The answer is at (1.0, -0.5, 0.5)
```

That is the same answer as before. Note we have still relied on some black box solver inside of fsolve (instead of inside minimize), but it might be more clear what problem we are solving (e.g. finding zeros). It takes a bit more work to set this up, since we have to construct the augmented function, but autograd makes it pretty convenient to set up the final objective function we want to solve.

How do we know we are at a minimum? We can check that the Hessian is positive definite in the original function we wanted to minimize. You can see here the array is positive definite, e.g. all the eigenvalues are

positive. autograd makes this easy too.

```
from autograd import hessian
h = hessian(objective, 0)
h(np.array([x, y, z]))
```

```
array([[ 2.,  0.,  0.],
       [ 0.,  2.,  0.],
       [ 0.,  0.,  2.]])
```

In case it isn't evident from that structure that the eigenvalues are all positive, here we compute them:

```
np.linalg.eig(h(np.array([x, y, z])))[0]
```

```
array([ 2.,  2.,  2.])
```

In summary, autograd continues to enable advanced engineering problems to be solved.

org-mode source

Org-mode version = 9.1.14

Tweet Discuss on Twitter

A set of emacs functions and bindings to google under point.

collaborate
"real-time" collaboration in emacs via git

dft-book *(Python)*
A book on modeling materials using VASP, ase and vasp

csearch *(Jupyter Notebook)*
search utilities for Google colab

espyranto *(Python)*
Python library to read multi-well plates

f19-06623 *(Jupyter Notebook)*
Course repository for 06-623

dft-book-espresso *(Jupyter Notebook)*
New version of dft-book for Quantum Espresso

emacs-modules *(C)*
Dynamic modules for emacs

f18-06623 *(Jupyter Notebook)*
Fall 2018 - Mathematical Modeling of Chemical Engineering Processes

aenet *(Fortran)*
Atomic interaction potentials based on artificial neural networks

ghub *(Emacs Lisp)*
Minuscule client library for the Github API

example-project
An example project

emacs-win *(Emacs Lisp)*
An emacs distribution for use with jmax

autograd *(Python)*
Efficiently computes derivatives of numpy code.

ase-espresso
ase interface for Quantum Espresso

amp-tutorial *(TeX)*
null

emacs-ffi *(C)*
FFI for Emacs

f16-06625 *(TeX)*
Chemical and Reactive Systems for Fall 2016

aiche-2016 *(HTML)*
Expanding Molecular Simulation Use by Data/Code Sharing in Scientific Publishing

gitolite *(Perl)*
Hosting git repositories -- Gitolite allows you to setup git hosting on a central server, with very fine-grained access control and many (many!) more powerful features.

criticmarkup-emacs *(Emacs Lisp)*
Emacs minor mode for handling CriticMarkup.

ACS-2016-data-sharing *(HTML)*
My talk at the Spring ACS 2016 meeting

f15-06625 *(Emacs Lisp)*
06-625 Chemical and Reactive Systems

elpa
ELPA repository for my emacs packages

box-course *(Python)*
Python wrappers for the box.com v2 API

blogofile_blog *(Python)*
A Blogofile blog plugin

blogofile *(Python)*
A static website compiler and blog engine written in Python.

dft-course *(Emacs Lisp)*
Course repository for 06-640 - Molecular simulation