SRA の技術情報誌
GS LetterNeo
Vol.191　2024 年 6 月

Pymanopt を使った多様体最適化

熊澤 努

技術本部 先端技術研究室

はじめに

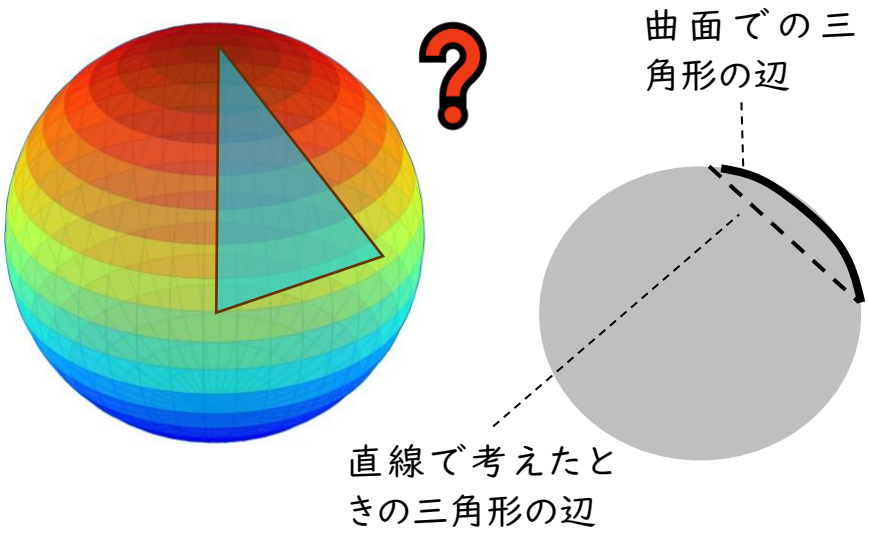
今回は多様体最適化に挑戦します．過去に最適化法の回と同様に Python を使います．Python パッケージ Pymanopt[1]を使い，リーマン多様体上の最適化問題を自動的に解いてみましょう．多様体最適化の詳細については [1]を参考にしてもらうことにして，ここでは実際に問題を解く流れを見ていきます．

曲がった立体を考える

最初に，高校までの数学で学ぶ図形を考えてみましょう．平面なら三角形や円，立体なら立方体や球があったと思います．教科書に書かれていた三角形の絵を思い出す読者もいるかもしれません．では，現実の三角形はどうでしょうか．例えば，地球の表面に大きな三角形を描いてみましょう． 次のページの図のように描いては少しマズいです．下の三角形の辺は地球の表面上ではなく，内側を通るからです．辺は直線ではなく，表面の曲がり具合に従った曲線でなくてはいけません．さらに，三角形の頂点にあたる 2 つ地点の一方から他方に歩いて移動する場合も，最短ルートは直線ではなく曲線です（3 辺のどれかになります）．直線で移動したければ地面を掘り進めていく必要があります．移動距離も曲線の長さを考え

[1] https://pymanopt.org/

る必要があるので，扱い方が複雑になります．このように，現実の世界では曲がった立体のうえで「最短性」を考える場合があります．もちろん，高校数学が扱うように背後に「真っすぐな」平面や空間の存在を仮定したうえで円や球を考えてもよいのですが，初めから曲がった空間（地球の表面）を考えておくのも１つの方法です．最初から球面を考えるので，その上での移動がすっきり扱えそうです．数学ではこのような曲がった空間は多様体という概念を使って議論することができます．最適化の話題はこれまでも何回か取り上げましたが，今回は土台自体が曲がっている場合です．



Pymanopt を準備する

Pymanopt は pip でインストールできます。

```
> pip install pymanopt
```

Pymanopt の公式サイトの例を使いながら，最適化問題を実際に解いてみましょう．公式サイトのプログラムは，合わせて autograd[2]という数学計算用のパッケージも使っています．必要に応じてインストールしてください．この記事で紹介するプログラムは Python 3.10.12，Pymanopt 2.0.1 で動作を確認しています。

[2] https://github.com/HIPS/autograd

固有値分解を行う

今回は，公式サイトに掲載されている行列の最大固有値に対する固有ベクトルを求める例[3]で Pymanopt の使い方を勉強します．例のプログラムを少し簡略化，変更して最小固有値に対する固有ベクトルを求めるプログラムを掲載します．

```
import autograd.numpy as anp
import pymanopt
import pymanopt.manifolds
import pymanopt.optimizers

# 固有ベクトルを求める行列(正規乱数から生成した値)
A = anp.array(
    [[ 1.62434536, -0.84236252,  0.60832001],
     [-0.84236252,  0.86540763, -1.5313728 ],
     [ 0.60832001, -1.5313728,   0.3190391 ]])

# 3次元球面の定義
sphere = pymanopt.manifolds.Sphere(3)

# 目的関数(関数については注釈3または[1]参照)
@pymanopt.function.autograd(sphere)
def objective(x):
    return x @ A @ x

# 最小化問題の定義
problem = pymanopt.Problem(sphere, objective)

# 最適化の実行
optimizer = pymanopt.optimizers.SteepestDescent()
ret = optimizer.run(problem)

# 結果の表示
print("最小固有値に対する固有ベクトル:", ret.point)
```

上のプログラムは行列 A を 2 次元配列で定義しています．2 次元行列は autograd の anp.numpy を使って作っていて，numpy の ndarray 配列と同じものと考えて構いません．次に，3 次元球面を表す sphere を用意します．関数 objective は目的関数といい，球面 sphere の表面を飛び出したり，球の中に入り込んだりすることがないようにしながら，この関数の値が最小になるような球面上の点を計算します．目的関数の意味は省略します．興味ある読者は注３の公式サイトのチュートリアルや文献[1]を見てください．

次に，球面と目的関数から最小化問題オブジェクト problem を生成して，最適化を実行します．実行するアルゴリズムは（球面上の）最急降下法と呼ばれるもので，optimizer オブジ

[3] https://pymanopt.org/docs/stable/quickstart.html

エクトとしています．optimizer オブジェクトのメソッド run に問題 problem を渡すと，最適化の結果が得られます．

実行結果は以下のようになります．最終行以外は，Pymanopt が出力する途中経過です．目的関数（ここでは Cost となっています）の値が小さくなっていく過程が分かります．実行結果の最終行で A の固有ベクトルが正しく求められていることは，例えば numpy の固有ベクトル計算関数 np.linalg.eig を使えばすぐに分かります．

```
Optimizing...
Iteration    Cost                       Gradient norm
---------    -----------------------    --------------
   1         +3.3473868321161482e-01    3.68037736e+00
   2         -8.4691063924393362e-01    1.33572452e+00
   3         -9.5782136649498939e-01    3.46665111e-01
   4         -9.6097939359316764e-01    2.74323464e-01
   5         -9.6531526680663782e-01    8.63725410e-02
   6         -9.6578260655985559e-01    1.22143191e-02
   7         -9.6578977059817950e-01    6.47023156e-03
   8         -9.6579215356663517e-01    2.21659485e-03
   9         -9.6579222003186316e-01    1.98633837e-03
  10         -9.6579241565610396e-01    9.36864454e-04
  11         -9.6579244037975331e-01    7.00484707e-04
  12         -9.6579246111791306e-01    4.06727164e-04
  13         -9.6579246181667333e-01    3.93025912e-04
  14         -9.6579246441339550e-01    3.37252495e-04
  15         -9.6579247108072996e-01    9.57116914e-05
  16         -9.6579247155828241e-01    4.08684359e-05
  17         -9.6579247166320326e-01    4.95246099e-06
  18         -9.6579247166476334e-01    2.39893144e-07
Terminated - min grad norm reached after 18 iterations, 0.02 seconds.

最小固有値に対する固有ベクトル: [0.03312778 0.6500833  0.75914047]
```

多様体を切り替える

次に多様体を球面から別のものに切り替えてみます．ここでは，グラスマン多様体にして，先ほどと同じ，最小固有値に対する固有ベクトルを求めます．不正確ですが感覚的に述べると，使う多様体を変えると「曲面の形状」も変わり，目的関数も変更する必要があります．ここでは，文献[1]に従って変更します．プログラムは以下の通りです．

```
import autograd.numpy as anp
import pymanopt
import pymanopt.manifolds
import pymanopt.optimizers

# 乱数の種
anp.random.seed(2)

# 固有ベクトルを求める行列(正規乱数から生成した値)
A = anp.array(
    [[ 1.62434536, -0.84236252,  0.60832001],
     [-0.84236252,  0.86540763, -1.5313728 ],
     [ 0.60832001, -1.5313728,   0.3190391 ]])

# グラスマン多様体の定義
grassmann = pymanopt.manifolds.Grassmann(n=3, p=1)

# 最小化問題の定義
# 目的関数(関数については [1]参照)
@pymanopt.function.autograd(grassmann)
def objective(x):
    return anp.trace(x.T @ A @ x)

problem = pymanopt.Problem(grassmann, objective)

# 最適化の実行
optimizer = pymanopt.optimizers.GradientDescent()
ret = optimizer.run(problem)

# 結果の表示
print("最小固有値に対する固有ベクトル:", ret.point)
```

最初のプログラムからの変更点は，球面をグラスマン多様体に変えたところ（コメント「グラスマン多様体」の次の行），目的関数を変えたところ（コメント「目的関数」の次の３行），最適化法を最急降下法から共役勾配降下法に変えたところです（コメント「最適化の実行」の次の行）．この問題の場合は最急降下法でも答えを求めることができるので，本質的な変更ではありません．問題を解くための流れは先ほどと同じです．

結果は下のようになります．ベクトルの各成分の符号が負になっていますが重要ではなく，固有ベクトルが正しく求められていることが分かります．

.

```
Optimizing...
Iteration    Cost                       Gradient norm
---------    -----------------------    -------------
   1         +5.0973869030228347e-01    3.67231078e+00
   2         -7.6949303431480010e-01    1.51258905e+00
   3         -9.4274781031233035e-01    4.03246782e-01
   4         -9.6572522371276559e-01    2.71332674e-02
   5         -9.6578248606515349e-01    8.51282875e-03
   6         -9.6578986957579871e-01    4.31079104e-03
   7         -9.6579178445664293e-01    2.21502258e-03
   8         -9.6579247112483890e-01    6.23447548e-05
   9         -9.6579247165950899e-01    8.88208896e-06
  10         -9.6579247166350668e-01    4.26204810e-06
  11         -9.6579247166442639e-01    2.10290033e-06
  12         -9.6579247166465443e-01    1.11058301e-06
  13         -9.6579247166474991e-01    3.45930973e-07
Terminated - min grad norm reached after 13 iterations, 0.01 seconds.

最小固有値に対する固有ベクトル:
[[-0.0331277 ]
 [-0.65008327]
 [-0.7591405 ]]
```

おわりに

今回は Pymanopt を使って多様体上の最適化問題の求解を体験しました。行列の固有値問題を題材に，球面とグラスマン多様体の場合について求解プログラムを解説しました.

引用文献

[1] 佐藤寛之 , 笠井裕之, “リーマン多様体上の最適化の基本と最新動向,” システム／制御／情報, 第 巻 62, 第 1, pp. 21-27, 2018.

GSLetterNeo Vol.191
2024 年 6 月 20 日発行
発行者　株式会社 SRA 技術本部 先端技術研究室

編集者　熊澤努　方 学芬
バックナンバー https://www.sra.co.jp/public/sra/gsletter/
お問い合わせ　gsneo@sra.co.jp

株式会社SRA
〒171-8513 東京都豊島区南池袋 2-32-8