計画情報数理レポート課題

森田研究室 M1 本田雄一郎

2016 年 7 月 19 日

1 序論

最小包囲円問題，円詰め込み問題を勾配法で解くにあたって，今回は最急降下法を利用した．目的関数を $f(\boldsymbol{x})$ とすると，最急降下法のアルゴリズムは次で与えられる:

Algorithm 1 最急降下法

while $\|\nabla f(\boldsymbol{x})\| \geq \epsilon$ **do**
  $\boldsymbol{d} = -\nabla f(\boldsymbol{x})$
  $\boldsymbol{x} := \boldsymbol{x} + \alpha \boldsymbol{d}$ // α はステップ幅
end while

ただし，ϵ は十分小さい正数である．最急降下法の各反復での移動方向 $\boldsymbol{d}$ は次の式で与えられる．

$$\boldsymbol{d} = -\nabla f(\boldsymbol{x})$$

この式で与えられる移動方向 $\boldsymbol{d}$ は，勾配ベクトルとの内積が常に負になることから，関数の減少する移動方向であることが保証される．また，移動のステップ幅 α を与えるのには次の Armijo の方法を用いた:

Algorithm 2 Armijo の方法

$\alpha > 0$, $\beta \in (0, 1)$, $\tau \in (0, 1)$
while $f(\boldsymbol{x} + \alpha \boldsymbol{d}) > \tau \alpha \nabla f(\boldsymbol{x})^\top \boldsymbol{d}$ **do**
  $\alpha := \beta \alpha$
end while

なお，言うまでもないことだが，勾配法を利用する際，問題に制約条件が付いている場合には，問題の目的関数にそのまま勾配法を適用できないことに注意すべきである．そのまま適用した場合，制約条件に違反してしまうおそれがあるからである．このような場合，制約条件の違反度をペナルティとして目的関数に加えた緩和問題を新たな目的関数にするなどの一手間が必要である．
また，実行速度を上げるために工夫した点として，Armijo の方法で α を求める際に，1 つ前の反復で求めた α を渡し，それを各ステップでの α の初期値とした．これにより実行速度を 15 倍程度改善することができた．
入力のサイズは $n = 20, 100$ の場合でそれぞれテストした．
今回使用した初期値，パラメータなどは次の通りである:

パラメータ	文字	最小包囲円問題	円詰め込み問題
大円半径初期値	r	0.3	※ 1
重み係数	ρ	1.3	0.7
勾配ベクトルノルムの誤差	ϵ	10^{-4}	10^{-5}
α 初期値 (Armijo の方法)	α	1.0	1.0
α 更新時の乗数 (Armijo の方法)	β	0.9	0.9
一次近似係数 (Armijo の方法)	τ	10^{-4}	10^{-5}

※ 1 入力された円の面積の総和を S とすると，$\sqrt{\frac{3S}{\pi}}$

2　最小包囲円問題

最小包囲円問題の定式化は次の式による．

$$\begin{aligned} min \quad & r^2 \\ s.t. \quad & (x-x_i)^2+(y-y_i)^2 \leq r^2 \qquad (i=1,...,n) \\ & r \geq 0 \end{aligned}$$

この問題は制約付き問題なので，このままでは直線探索法が適用できない．そのため，次のようにして制約条件をペナルティに変換したものを新たな目的関数として，無制約問題を作る（ρ はペナルティの重みである）:

$$min \quad r^2+\rho\sum_{i=1}^{n} max\{0,(x-x_i)^2+(y-y_i)^2-r^2\}$$

実行結果:

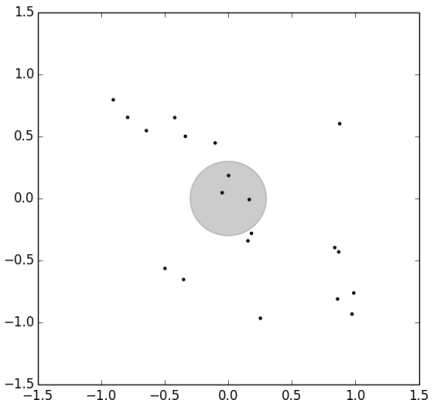

図 1: 最小包囲円 (n = 20, 初期状態)

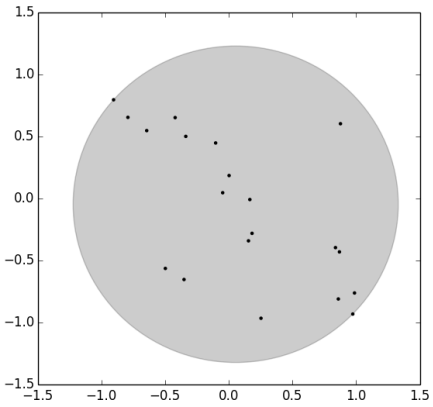

図 2: 最小包囲円 (n = 20, 出力)

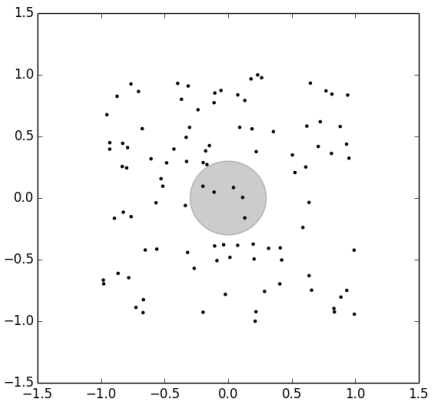

図 3: 最小包囲円 (n = 100, 初期状態)

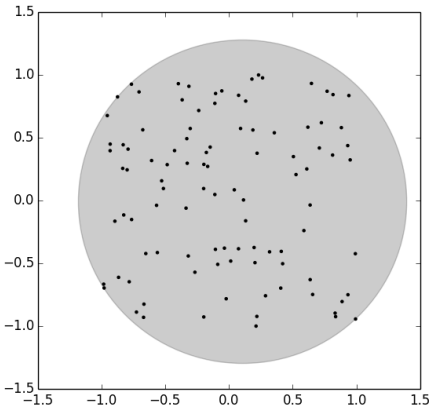

図 4: 最小包囲円 (n = 100, 出力)

いずれも目的関数の値は徐々に減少していく様子が確認でき，勾配法が成功していることがわかった．

3　円詰め込み問題

円詰め込み問題の定式化は次の式による．

$$
\begin{aligned}
min \quad & r \\
s.t. \quad & (x_i - x_j)^2 + (y_i - y_j)^2 \leq (r_i + r_j)^2 \qquad (i \neq j, \quad i, j = 1, ..., n) \\
& x_i^2 + y_i^2 \leq (r - r_i)^2 \\
& r \geq max\{r_i \,|\, i = 1, ..., n\}
\end{aligned}
$$

この問題もやはり制約付き問題なので，直線探索法が適用できない．そのため，次のようにして制約条件をペナルティに変換したものを新たな目的関数として，無制約問題を作る（ρ はペナルティの重みである）

$$
\begin{aligned}
min \quad & r + \rho \sum_{i=1}^{n} \sum_{j=i+1}^{n} max\{0, (r_i + r_j)^2 - (x_i - x_j)^2 - (y_i - y_j)^2\} \\
& + \rho \sum_{i=1}^{n} max\{0, x_i^2 + y_i^2 - (r - r_i)^2\} + \rho \, max\{0, max\{r_i \mid i = 1, ..., n\} - r\}
\end{aligned}
$$

実行結果:

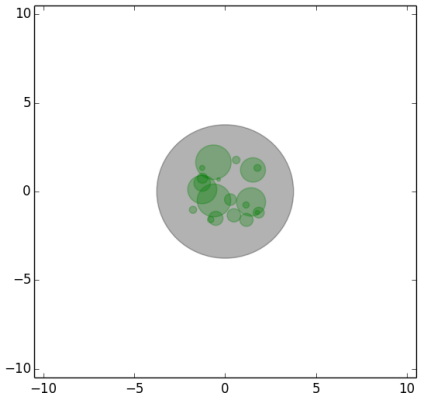

図 5: 円詰め込み (n = 20, 初期状態)

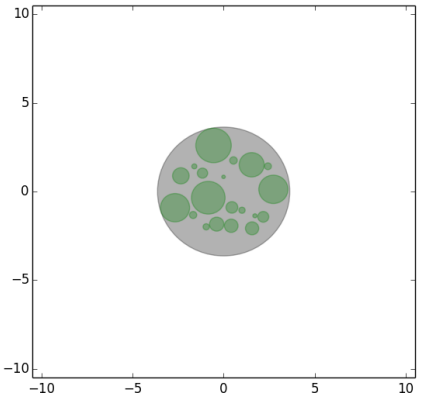

図 6: 円詰め込み (n = 20, 出力)

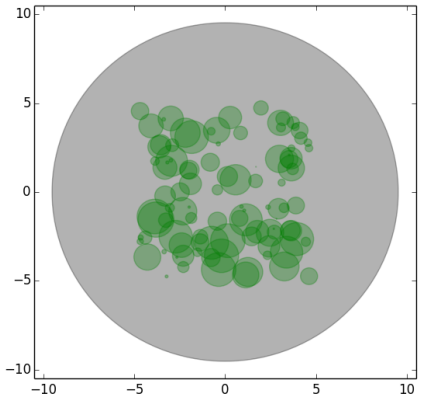

図 7: 円詰め込み (n = 100, 初期状態)

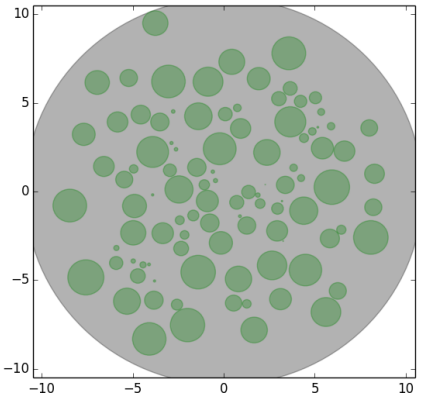

図 8: 円詰め込み (n = 100, 出力)

円詰め込み問題では最適解を出力することはできていないが，いずれも目的関数の値が徐々に減少していく様子が確認でき，勾配法自体は成功していることがわかった．

4　考察

今回は Armijo の方法による直線探索法を用いて実装したものを示した．序論に示したように，アルゴリズム自体は非常に単純であるが，反復の 1 回あたりは非常に高速であり，1 反復ごとに必ず関数の値を減少させることが確認された．円詰め込み問題に関しては，Wolfe の方法についても試してみたが，実行速度は Armijo の方法とほとんど変わらず，出力された解の精度が下がってしまったため，Armijo の方法によるものを採用した．τ_1, τ_2 の取り方をいろいろ変えて試してみたものの，いずれも実行速度に影響はほとんどなく，解の精度が下がっただけであった．
出力を見ると，最小包囲円問題では十分に信頼性の高い結果が得られたといえる．一方，円詰め込み問題では，目的関数の値は初期値に比べてかなり減少してはいるものの，最適値とは言い難い値に収束していることが伺える．これは最小包囲円問題が凸性を持つのに対して，円詰め込み問題は凸でなく，局所最適解が複数存在することによるものと思われる．円詰め込み問題に対してより高い精度で解を得ようとするならば，単純に直線探索をかけるだけでなく，入力された円の初期配置や，微妙にパラメーターを変更しながら直線探索を複数回かけるなどの工夫が必要と思われる．
最後に，大まかな実行時間であるが，MacBook Air (1.7GHz Intel Core i5, 4GB 1600MHz DDR3) で実行した結果，最小包囲円問題の $n = 20$ の場合 0.6±0.05 秒，$n = 100$ の場合 2.2±0.1 秒，円詰め込み問題の $n = 20$ の場合 2.2±0.2 秒，$n = 100$ の場合 120±10 秒であった．これらはそれぞれ 5 回実行した結果のおおよその平均と分散である．

5　作成したコード，入力データ

http://github.com/yuichiro12/python