札幌C++勉強会 #11 (2016.3.19)

関数の最小値を求めることから機械学習へ

H.Hiro @h_hiro_

自己紹介

H.Hiro

http://hhiro.net/ http://twitter.com/h_hiro_

- もともと札幌にいましたが、今年度当初から名古屋在住
- 情報系の研究の仕事をしています
- 今の仕事についてしばらくは
  RやMatlabを使ったりもしたけど
  最近はC++ばかり
- 趣味では以前からだけど
  Rubyメイン

はじめに

はじめに

- 今回は何かと数学の話が多いので
  なるべく用語を増やしすぎないようにしています
- なので「用語だけ出す」場合も多いです
  （この発表では説明しないけど、気になったら
  ぐぐれるように、という意味で）
- スライドはこちらにあるので、よろしければ
  http://www.slideshare.net/maraigue
- サンプルコードはこちら
  https://github.com/maraigue/sapporocpp-20160319

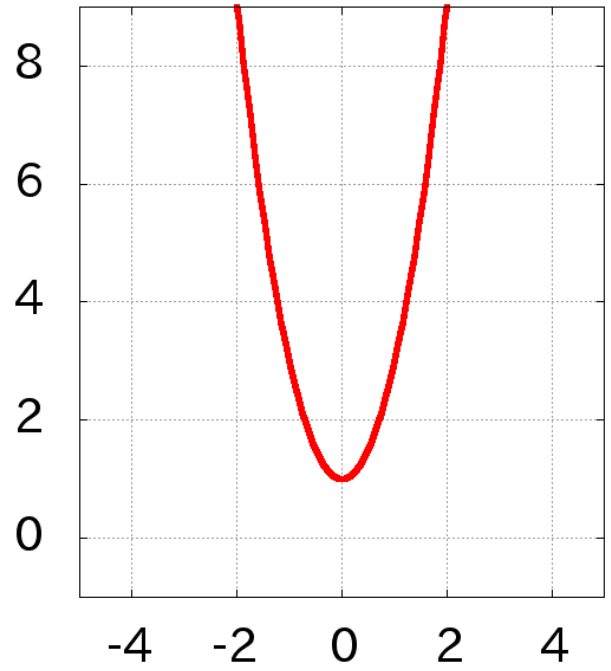
1.
今回のテーマ

関数の最小値を求める(1/4)

例えば、中学・高校数学でおなじみの二次関数

$$f(x) = 2x^2 + 1$$

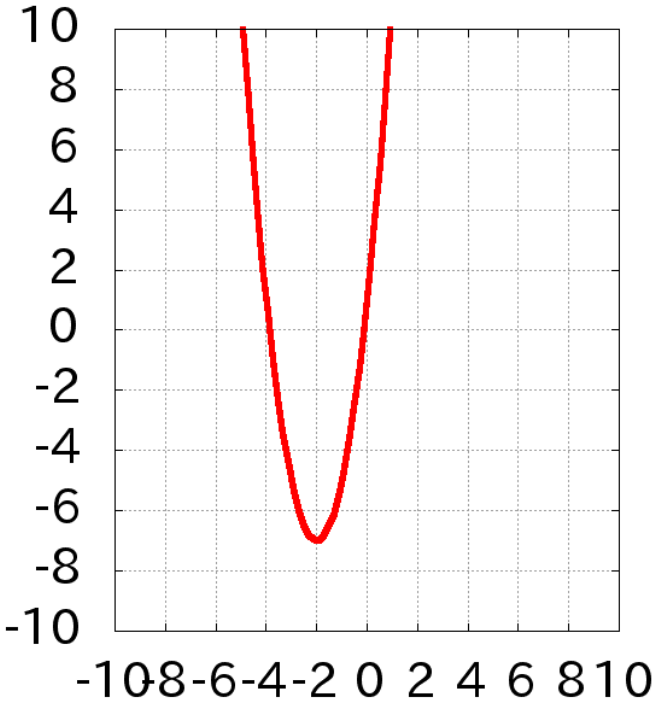
これの最小値は？また、これを最小にするxは？



関数の最小値を求める(2/4)

$$f(x) = 2x^2 + 8x + 1$$

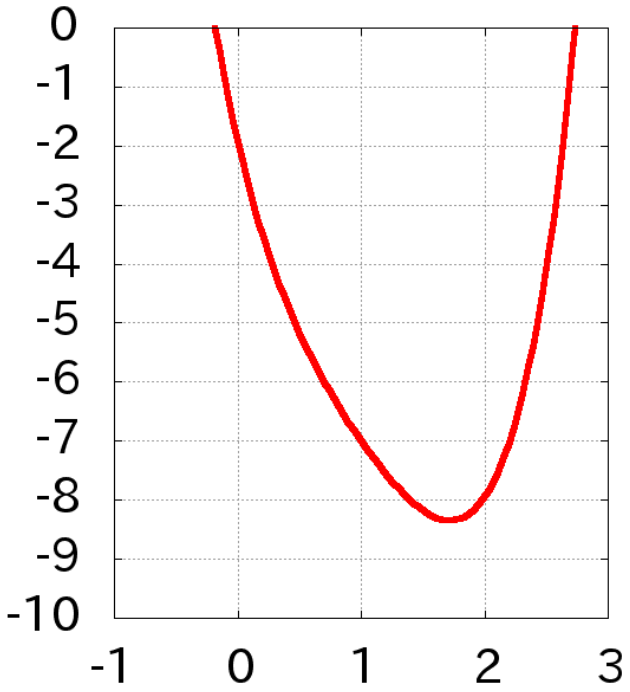
これの最小値は？また、これを最小にするxは？



関数の最小値を求める(3/4)

$$f(x) = (x - 1)^4 + (x - 2)^2 - 4\sqrt{x + 3}$$

これの最小値は？また、これを最小にするxは？



関数の最小値を求める(4/4)

- 与えられた式を最小化するような変数の値を求める
  - 手計算でできることもあれば、そうでないこともある
  - 今回は特に、手計算でできるとは限らない場合を扱う
    （そうでないとプログラムを書く必要はないので、当たり前と言えば当たり前ですが）
- 最大化問題も全く同様に考えることができるので今回の発表では最小化だけ扱う
  （一般に「最適化」という）

で、これが
どう機械学習と
関わるのか？

の前に、
「機械学習」について
概要を

「機械学習」とは

- いろんな解説があるけど
- 基本的には、コンピュータによる判断・推測を
  - 人手でルールを記述するのではなく
  - データをもとにルールを作ること
- 人の「学習」プロセスを、コンピュータ等に模倣させるという意味合い

「機械学習」とは

- 例えば、最近Googleが見せて話題になったもの
  - 画像分類
  - 囲碁AI
- 事例を多数用意しておく
  答えが分かっている画像（この画像は猫である）とか
  囲碁で勝てた打ち方とか
- 行いたい出力（処理）を、その事例に合う形に調整する
  画像の分類条件とか
  実際にAIが打つ手とか

「関数の最小化」から「機械学習」

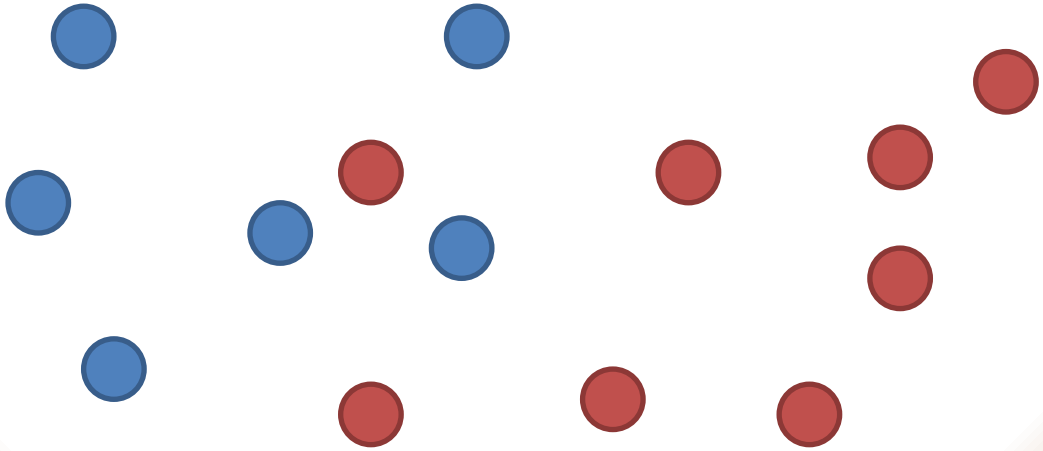
機械学習では多くの場合、こんな流れになる

- 学習結果をうまく表現できそうな数式やデータ構造を決める。そのとき、「学習するデータからの外れ具合」を計算できるようにしておく。
  例：“学習がうまくいけば、データを (p, q) としたとき、$f(a, b) = ap + bq - 1$ が最小になるはずだ。”
- その中にある変数（パラメータ）を、データに合うように調整する。これが「関数の最小化」となる。
  例：実際のデータが用意できた。実際に $f(a, b)$ を作って、最小になるような a, b を計算しよう。

2.
機械学習の
簡単な問題を
関数として見てみる

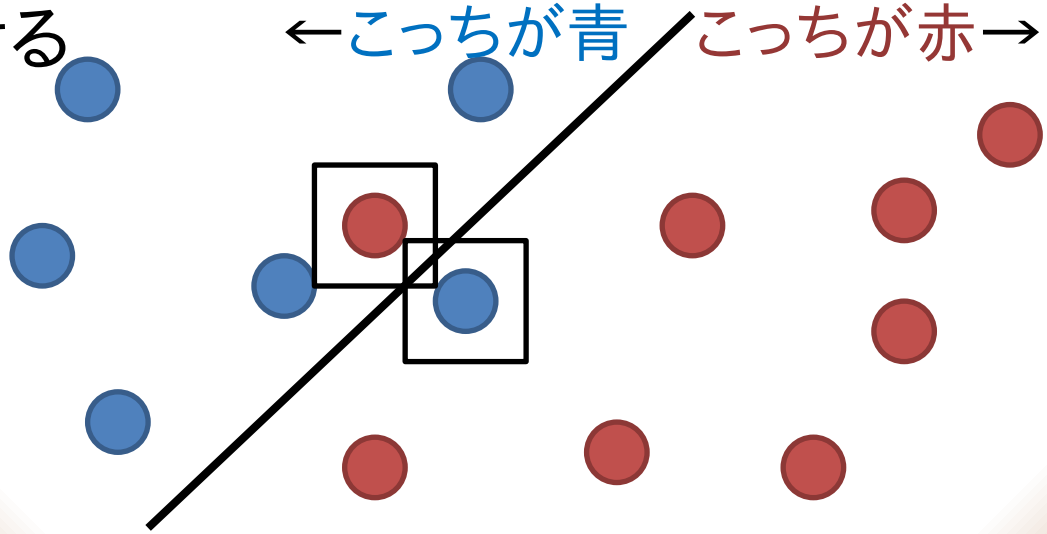
機械学習での例(1/3)

- 座標空間上に2種類のデータが置かれているとする
- それを分類する条件を求めたい（2値分類）
  例えば、青が「猫の画像」赤が「犬の画像」とか
  どうやって座標空間に置くか、という話も大事な話ではあるのだが
  （特徴抽出/特徴選択）、今回はそこは扱いません。



機械学習での例(2/3)

- 例えば、直線 $y = ax + b$ でこの二つを分けたいとする。（「線形分類」という）
- 最小化したい関数 $f(a, b)$: **分離できていない点の数**
  直線でなくても基本的なアイデアは同じです
- ↑と書いておくと、「パラメータ a, b を調整する問題」とみなせる



機械学習での例(3/3)

機械学習としては簡単なほうな例
（2種類しか分類できない、直線でしか分類できない）
を扱ったものの、これだけでも考えることは結構多い

- そもそも解ける？
- 解けるなら、どうやったら効率よく計算できる？

「どんな関数を最小化しようとしているのか」
意識することで、それが分かってくる（必ずではないけど）

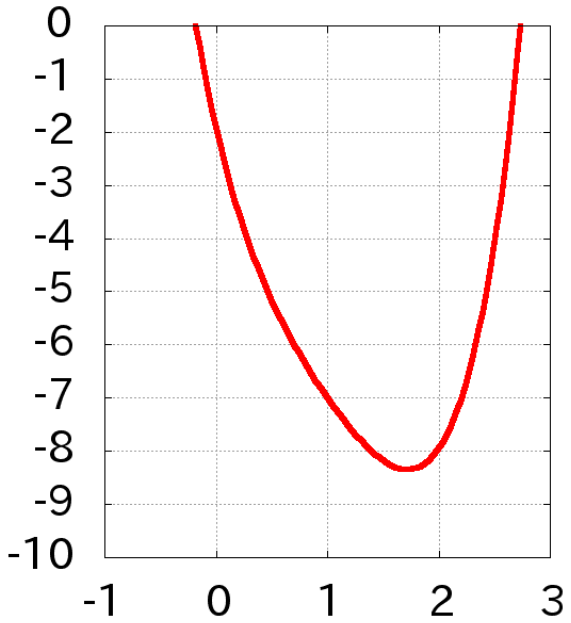
ここからお話しすること

もし以下のキーワードをご存知であれば、
こんな話をするのだと思っていていただければと
思います。

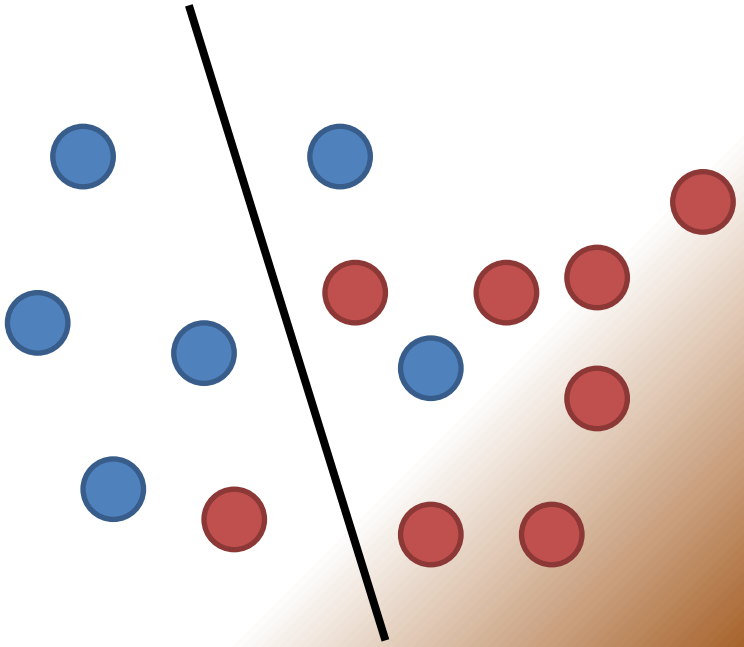
- 極小値、局所解
- 凸関数
- （多変数関数の）勾配

今回やること

1. 手計算で最小値を
   求めにくい関数
   $(x-1)^4+(x-2)^2-4\sqrt{x+3}$
   の最小値を求める

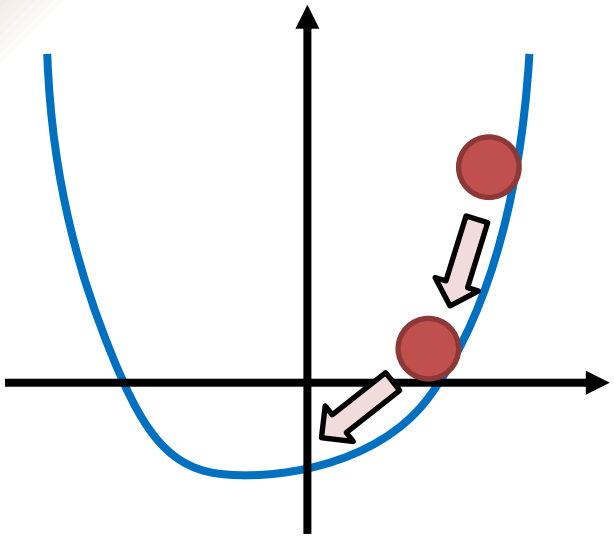


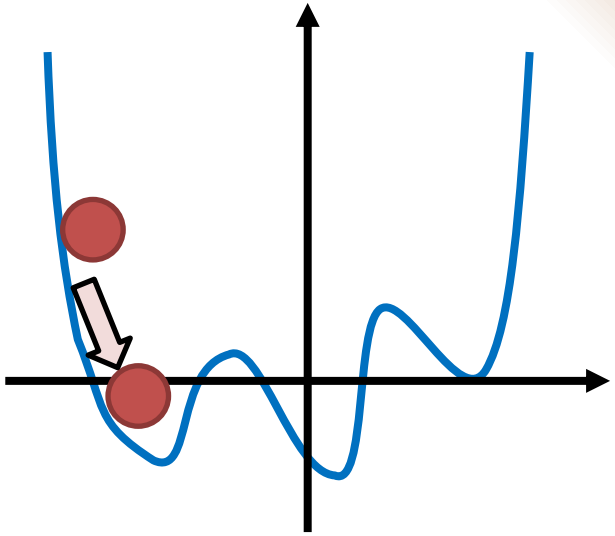
2. 線形分類問題を解く



3.
「関数の最小化」と
いっても
どんな最小値？

「最小値」の種類

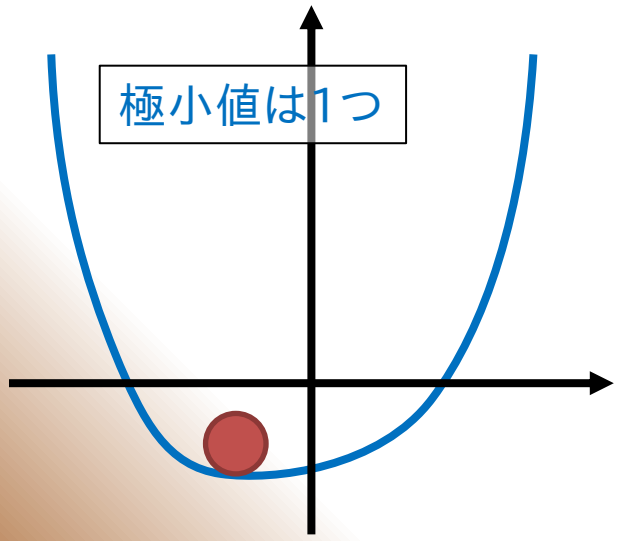


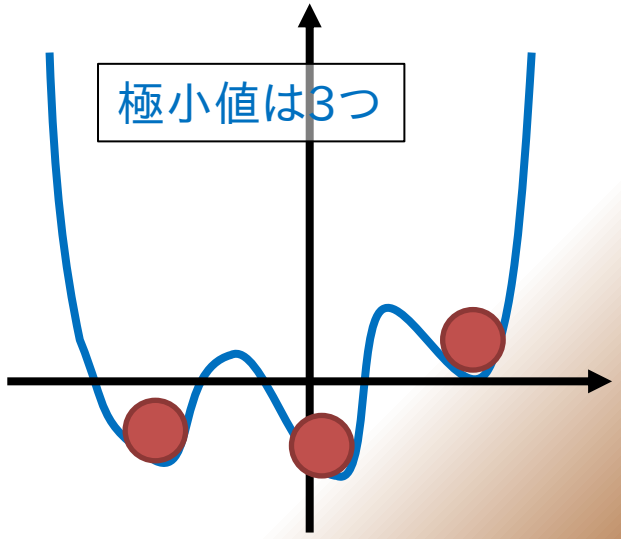


- 左の図のように、「いま見ている地点の付近だけ見て動けば、自然と最小値に辿り着ける」関数は扱いが簡単
- 逆に右の図のような場合は難しい

極小値

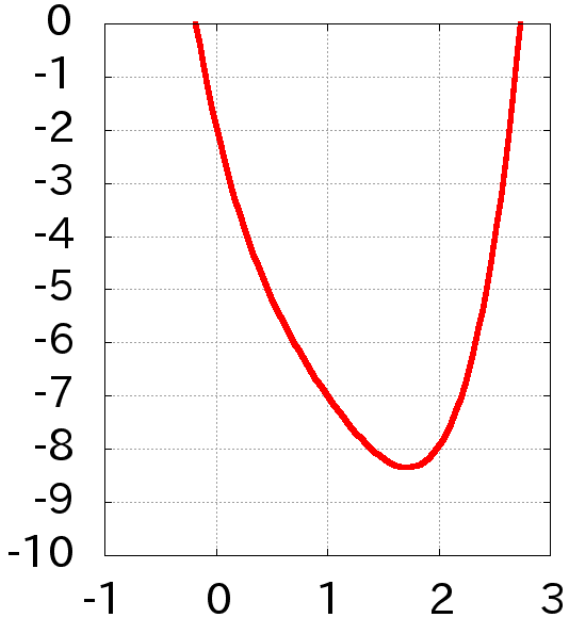
- 周辺のどこに行っても値が大きくなる場所
- 極小値が複数あることを想定しないとならない場合、最小値を求めるのが非常に面倒になる
  実際の最小点を「大域解」、最小とは限らない極小点を「局所解」という
- まずは極小値が一つしか存在しない場合を考える



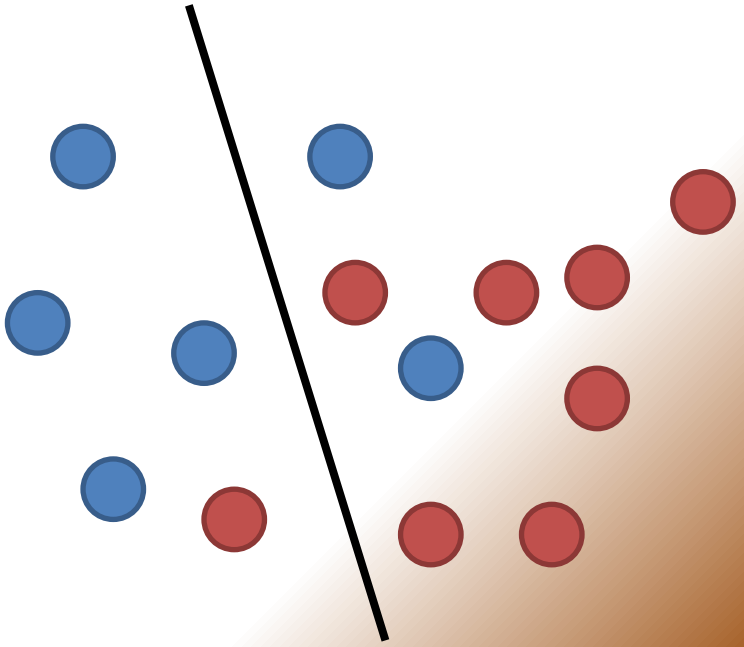


今回やること（再掲）

1. 最初に出した例
   $(x-1)^4 + (x-2)^2 - 4\sqrt{x+3}$
   の最小値を求める



2. 線形分類問題を解く



今回やること（再掲）

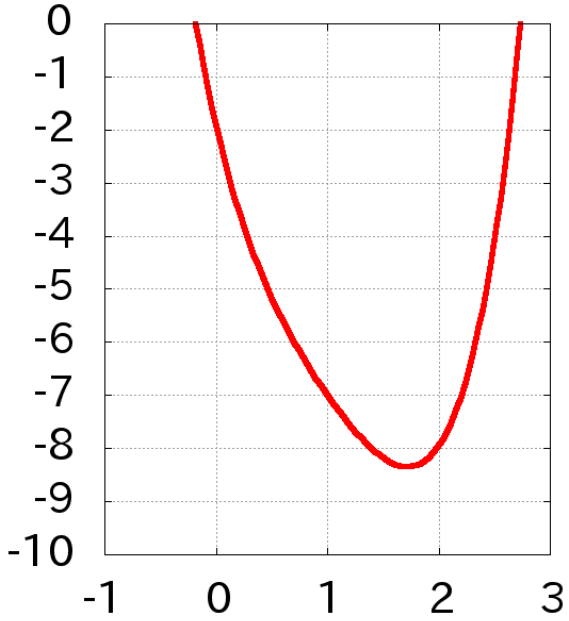
1.

最初に出した例

$(x-1)^4+(x-2)^2-4\sqrt{x+3}$

の最小値を求める

→見るからに極小値は一つ！

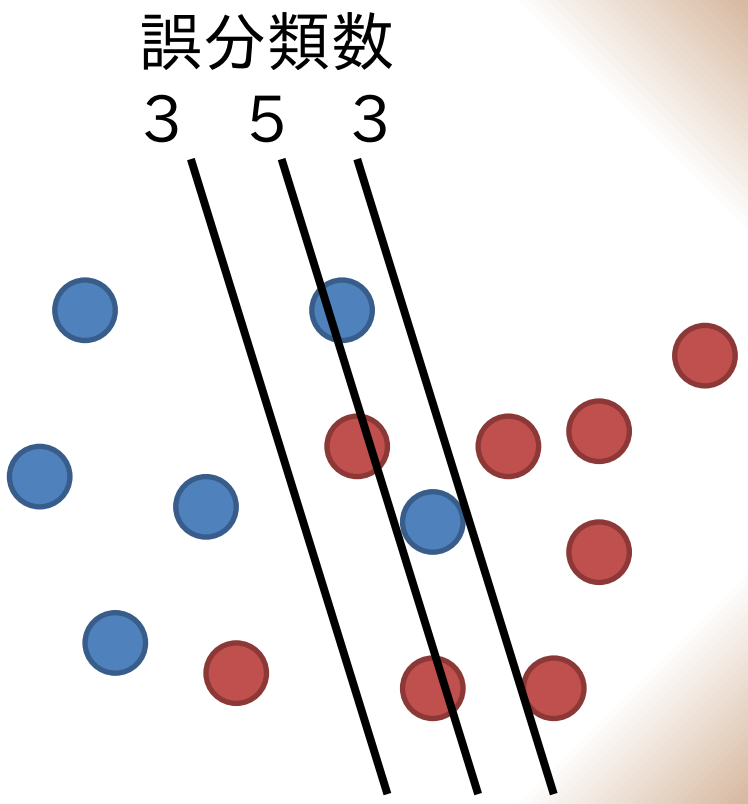


今回やること（再掲）

2.

線形分類問題を解く

→残念ながら
　極小値は一つにならない。



なので、後で説明するが、

極小値が一つになるように

問題を変形するのが常套手段。

解けないとどうしようもないし。

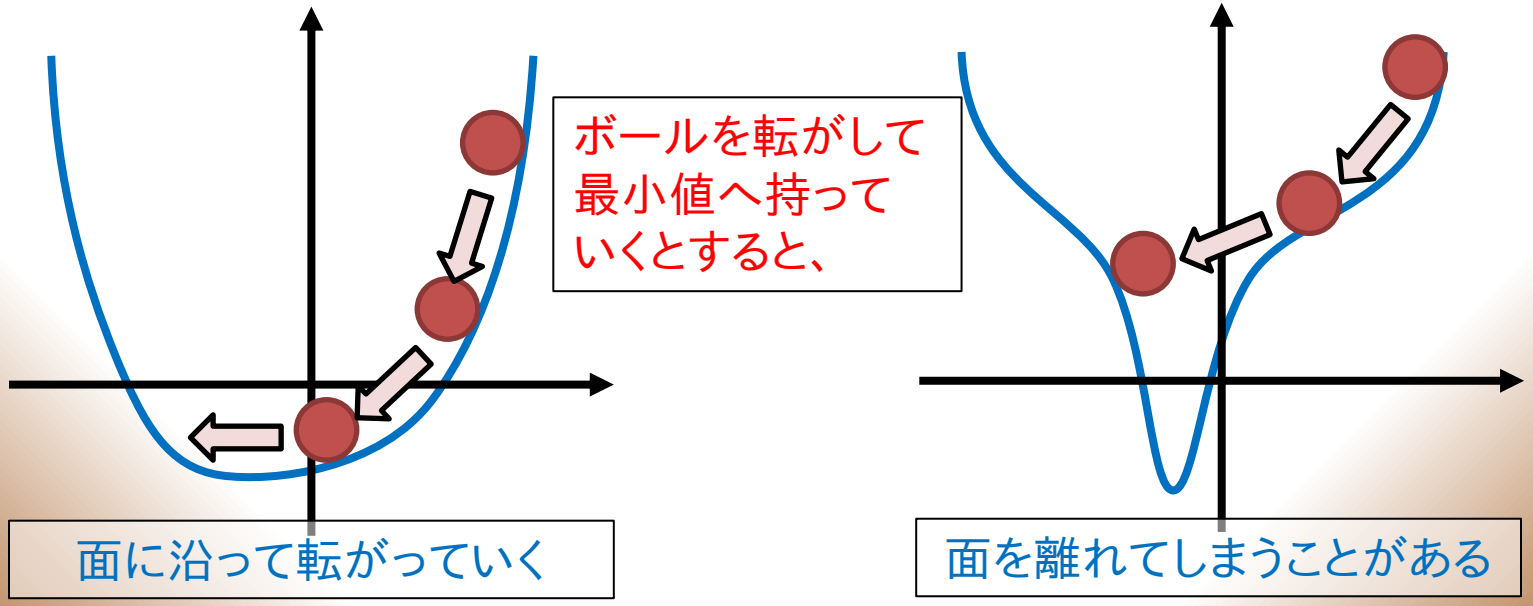
4.
「関数の最小化」が特に楽なもの：凸関数

極小値が一つでも…

最小化が簡単なものとそうでないものがある。

左の図のようになっていると、コンピュータにとっても計算が楽になる。

具体的には、最小値に収束する（＝いくらでも近づける）アルゴリズムが構築できる。

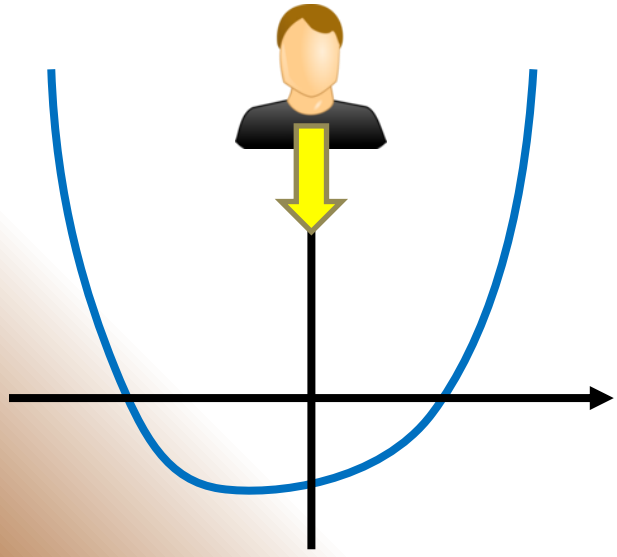


凸関数(1/2)

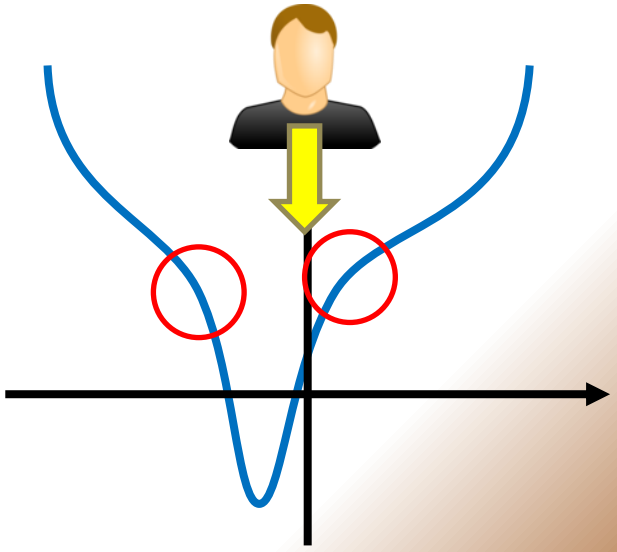
前ページで示したことに対する数学的な概念として「**凸関数**」がある。

直感的には、下図のように「上から見たときに、へこんでいる場所がない」関数のこと

凸な極小値が一つの関数

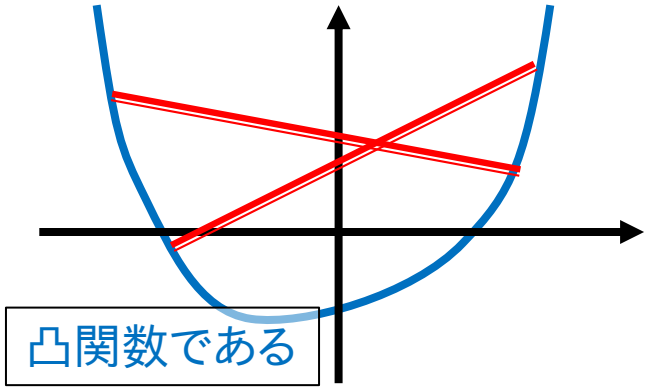


凸ではない極小値が一つの関数

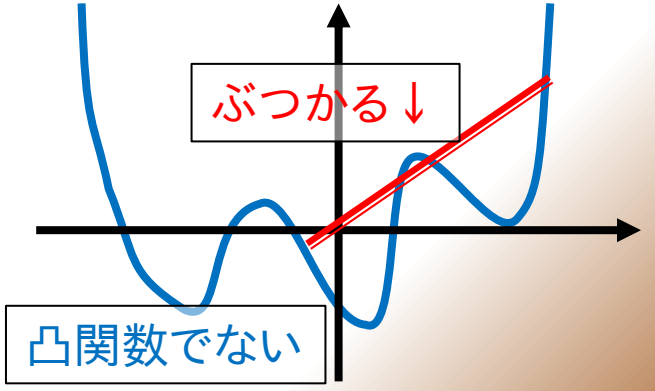


凸関数(2/2)

- 凸関数のより数学的な定義：
  右図のように「どの2点間に
  直線の橋（赤線）をかけても
  地面にぶつからない」
  直線は凸関数とみなす（広義凸関数）



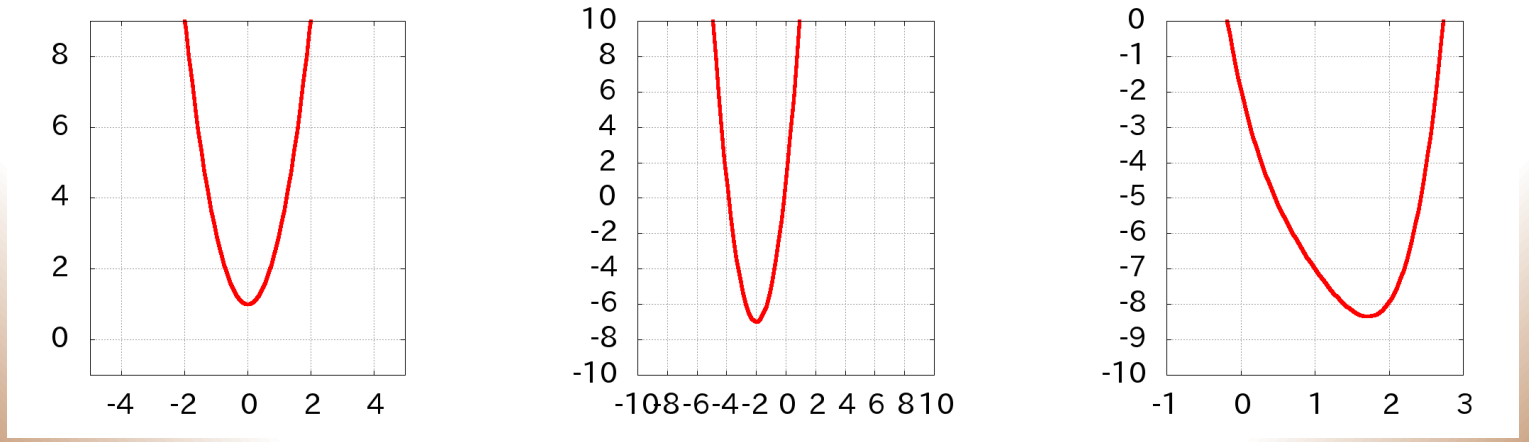
**計算の都合上「凸関数で
ないものを凸関数で近似する」
ことが行われるくらい便利**



例

- 実は、最初に出した三つの例はどれも凸関数
  - $f(x) = 2x^2 + 1$
  - $f(x) = 2x^2 + 8x + 1$
  - $f(x) = (x - 1)^4 + (x - 2)^2 - 4\sqrt{x + 3}$

これに数学的な操作を施してみる

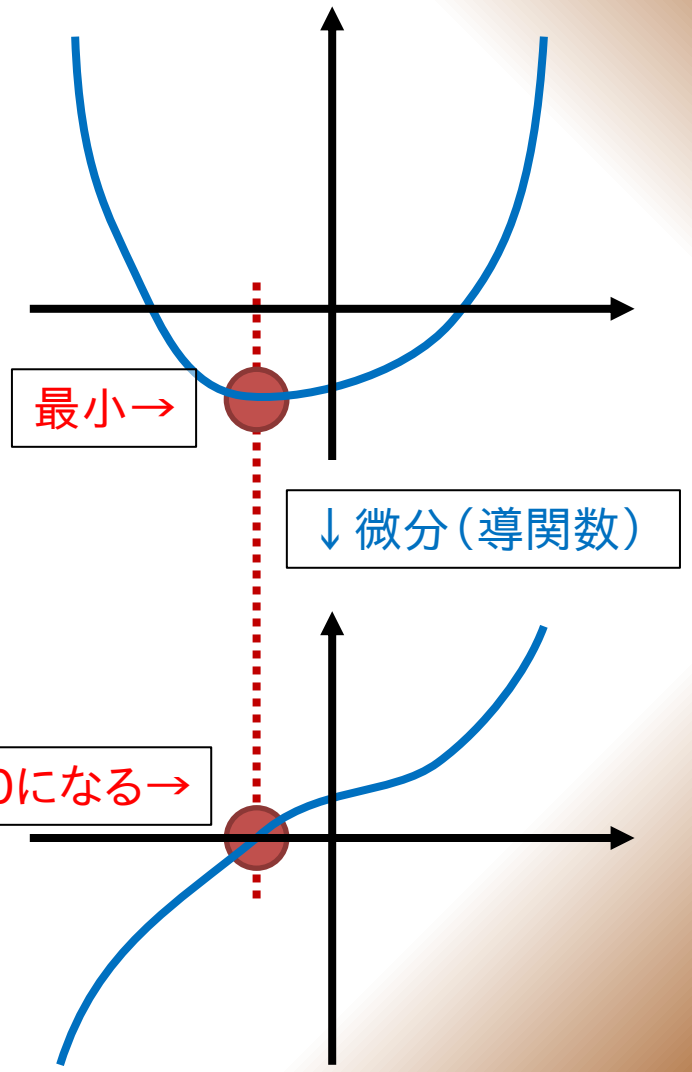


凸関数の数学的な特徴

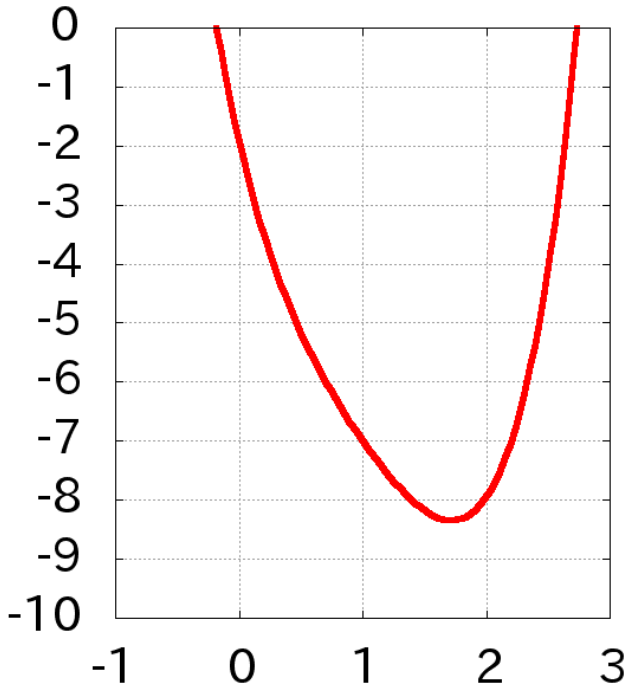
微分可能な1変数凸関数は：
2変数以上の場合はあとで述べる。

- **微分結果（導関数）が単調増加**
- **最小化される点においては微分係数が0になる**
  （定義域に制限がない場合）

微分：簡単に言えば「傾き」
右下に向いていればマイナス、
右上に向いていればプラス



例



- 実は、最初に出した三つの例は
  - $f(x) = 2x^2 + 1$
  - $f(x) = 2x^2 + 8x + 1$
  - $f(x) = (x - 1)^4 + (x - 2)^2 -$
- さっきの「最小化される点における微分が0になる」を計算してみる
  - $f'(x) = 4x = 0 \quad \Rightarrow x = 0$
  - $f'(x) = 4x + 8x = 0 \quad \Rightarrow x = -2$
  - $f'(x) = 4(x - 1)^3 + 2(x - 2) - 2 / \sqrt{x + 3} = 0$
    $\Rightarrow$???

使う道具：ニュートン法(1/2)

- 手計算で値が求められないなら、コンピュータにがんばってもらえばいいじゃない
- 今回は「ニュートン法」を使う。
  $g(x) = 0$ の形の方程式を解く方法
  ただし、$g(x)$ が微分可能である必要がある
- $g(x) = f'(x) = 4(x - 1)^3 + 2(x - 2) - 2 / \sqrt{x + 3} = 0$ を解いてみよう

ライブコーディングします。

- $g(x) = f'(x) = 4(x - 1)^3 + 2(x - 2) - 2 / \sqrt{x + 3}$
  $\Rightarrow g'(x) = 12(x - 1)^2 + 2 - 1 / (\sqrt{x + 3})^3$
- ニュートン法の式はここを参考にします
  http://akita-nct.jp/yamamoto/lecture/2005/5E/nonlinear_equation/text/html/node4.html

使う道具：ニュートン法(2/2)

ニュートン法自体は目新しいわけではないのだが
こういった「基本的な手段の組み合わせで解ける」
ようにすると、数値計算上も都合がよい。

今回の問題だと

- 凸関数
- 2階微分可能

ということを確認できたおかげで、ニュートン法に
到達できた。

5.
凸関数の最小化を2変数以上に増やす

2変数以上の場合(1/3)

- さっきの分類問題のように、機械学習では
  2変数以上の場合も扱わないとならない
  時には、万単位の変数を使うことも
- 凸関数の定義は同じ
- 最小化の際は、1階微分の代わりに「勾配」、2階微分の代わりに「ヘッシアン」を使えばよい
  - 勾配：どれか1変数についてだけ微分したもの（偏微分）を全変数について集めてベクトルにしたもの
  - ヘッシアン：あらゆる2変数の組み合わせについて1回ずつ微分したものを、全組み合わせについて並べて行列にしたもの（右式）

$$\begin{pmatrix} \frac{\partial^2}{\partial x^2} f(x,y) & \frac{\partial^2}{\partial x \partial y} f(x,y) \\ \frac{\partial^2}{\partial y \partial x} f(x,y) & \frac{\partial^2}{\partial y^2} f(x,y) \end{pmatrix}$$

2変数以上の場合(2/3)

- 勾配やヘッシアンを考えるのは、「どう動けば最小化できるかを、全変数の組み合わせについて考えないとならない」ため

例（右図）
横がa軸、奥行がb軸、
高さが関数 $f(a, b)$ とする。
関数を最小化する
向き（矢印）は無数に
考えられる。

2変数以上の場合(3/3)

- 勾配やヘッシアンを考えるのは、「どう動けば最小化できるかを、全変数の組み合わせについて考えないとならない」ため
- ただし問題によっては、これらを全部考えなくてもよい。あるいは、全部考えなくてよいように問題を単純化することがある。
  - 特にヘッシアンは、変数の数の2乗でサイズが大きくなることや、ヘッシアンの「逆行列」という重い演算を使わないとならないため回避しても答えが出るのであれば便利。
  - 1変数ずつ順に最小化してもよいのであればヘッシアンはいらない。また「準ニュートン法」という、明示的にヘッシアンを求めず逐次的に更新することで得る方法もある。

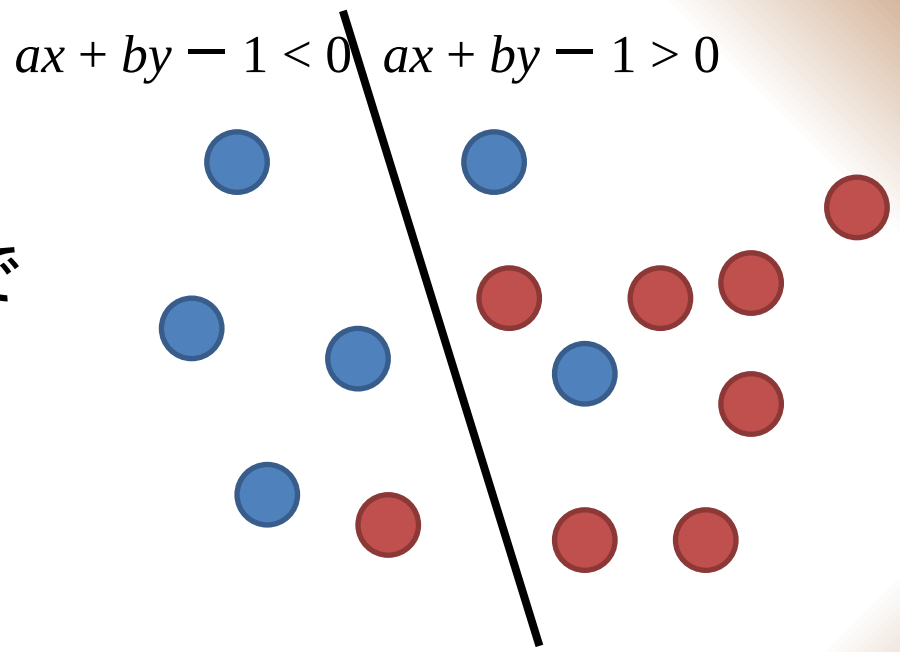
例題

- $f(x, y) = x^2 + 2xy + 3y^2 + 4x + 5y + 6$ の最小値を求める
- 実装は準ニュートン法
  （2階微分＝ヘッシアンを直接求めず、それに類する行列を逐次更新しながら求める）
- グラフを表示してみる
  gnuplotだと splot [-3:3][-3:3]x*x + 2*x*y + 3*y*y + 4*x + 5*y + 6
- 手計算の結果と比較する
  凸関数であることの確認は省略するが、凸関数であることがわかっているならば、$f(x, y)$ の x, y それぞれについての偏微分から
  $2x + 2y + 4 = 0, 2x + 6y + 5 = 0$ の両方が満たされれば最小値を取る

いよいよ
さっきの分類問題を解きます

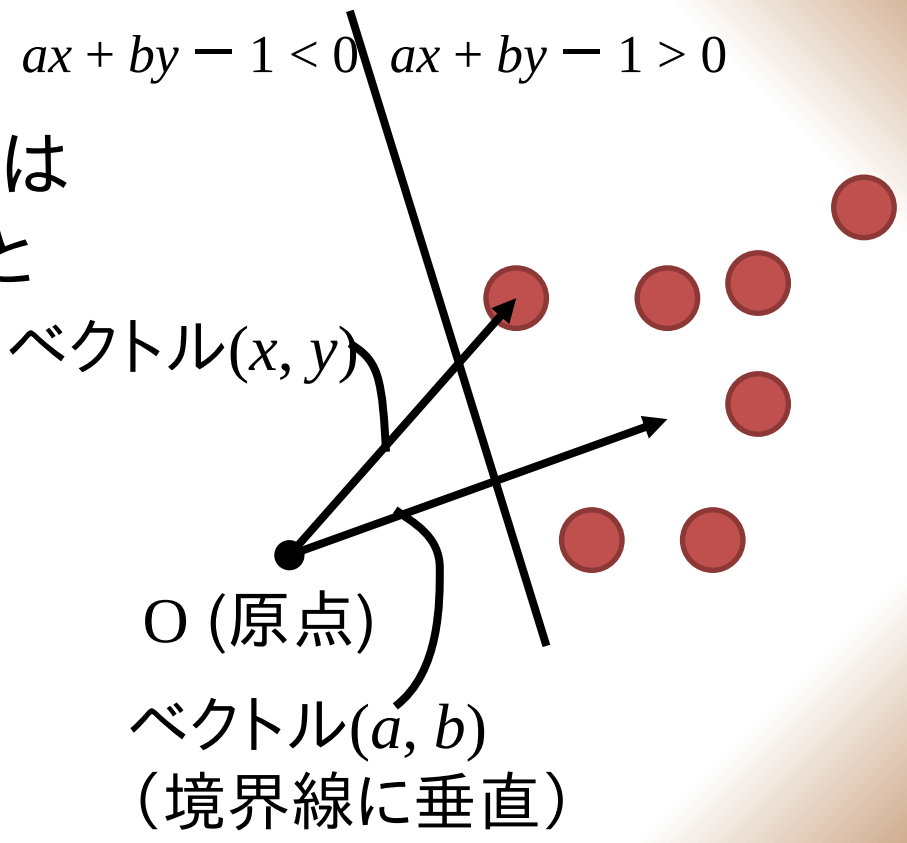
2変数以上の凸関数を最小化する例(1/4)

- さっきの線形分類問題
- 境界線が $y = ax + b$ では
  ちょっと扱いが面倒なので
  $ax + by - 1 = 0$ とします
- その上で、「点(x, y)が
  $ax + by - 1 > 0$ なら赤、
  $ax + by - 1 < 0$ なら青」
  として判定します



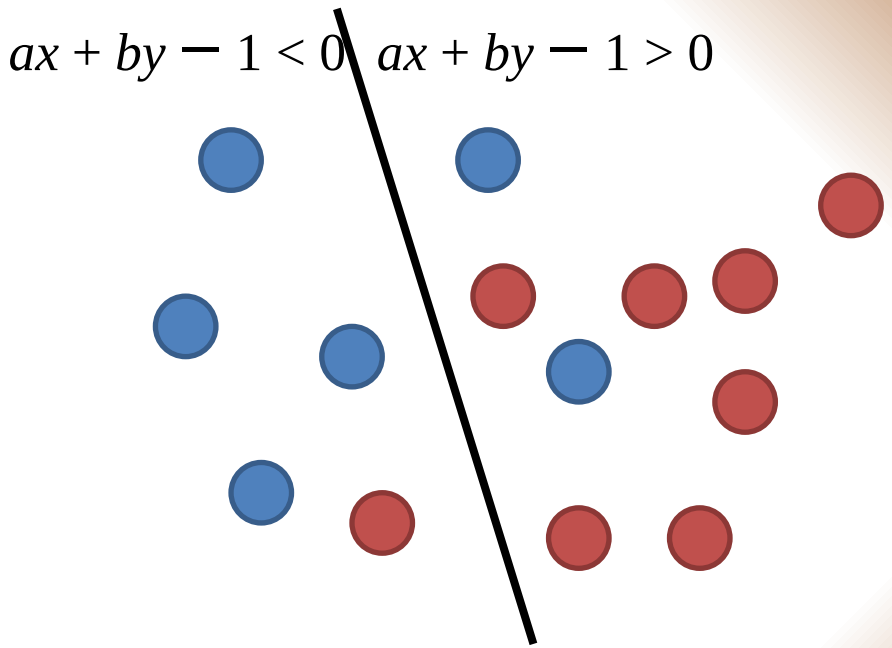
2変数以上の凸関数を最小化する例(1/4)

- 少し補足:
  「$ax + by - 1 > 0$ なら赤」は
  「$(a, b) \cdot (x, y) > 1$なら赤」と
  同じ
  ただし、「$\cdot$」は内積

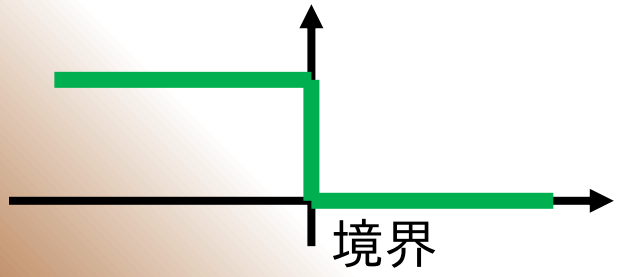


2変数以上の凸関数を最小化する例(2/4)

- 最小化すべき値として
  さっきは
  「分類を間違えた数」を
  使うと言いましたが
  残念ながら
  凸関数になりません。



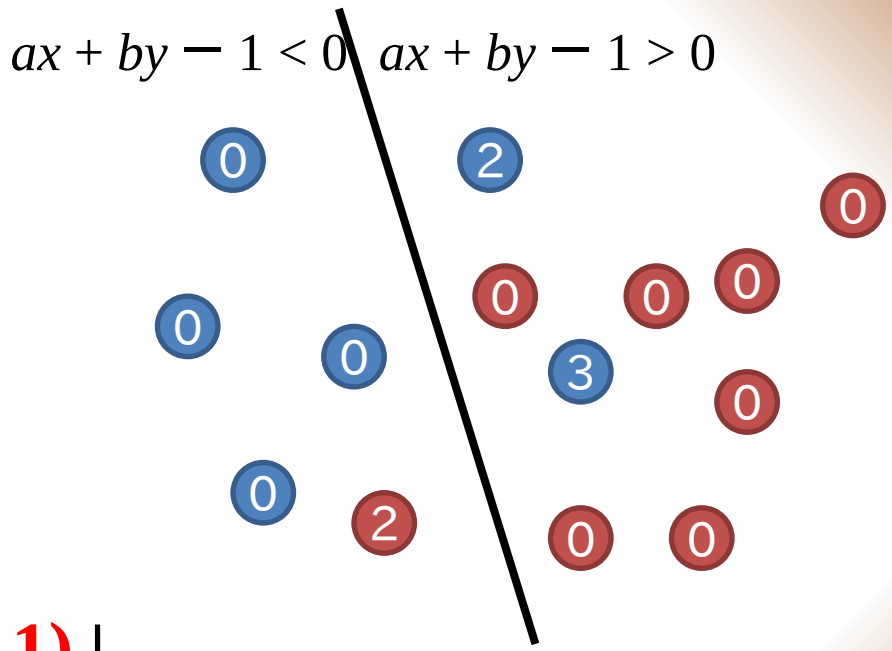
「分類を間違えた数」だと…



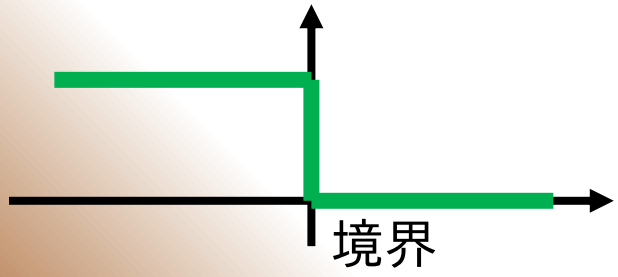
2変数以上の凸関数を最小化する例(2/4)

そこで代わりに
「$ax + by - 1 > 0$ なら赤」の
条件に着目します

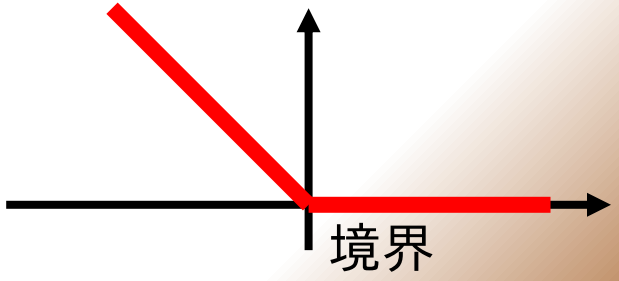
- 赤い点が条件を満たす
  → 関数値は0
- 赤い点が条件を満たさず
  → 関数値は「$-(ax + by - 1)$」



「分類を間違えた数」だと…



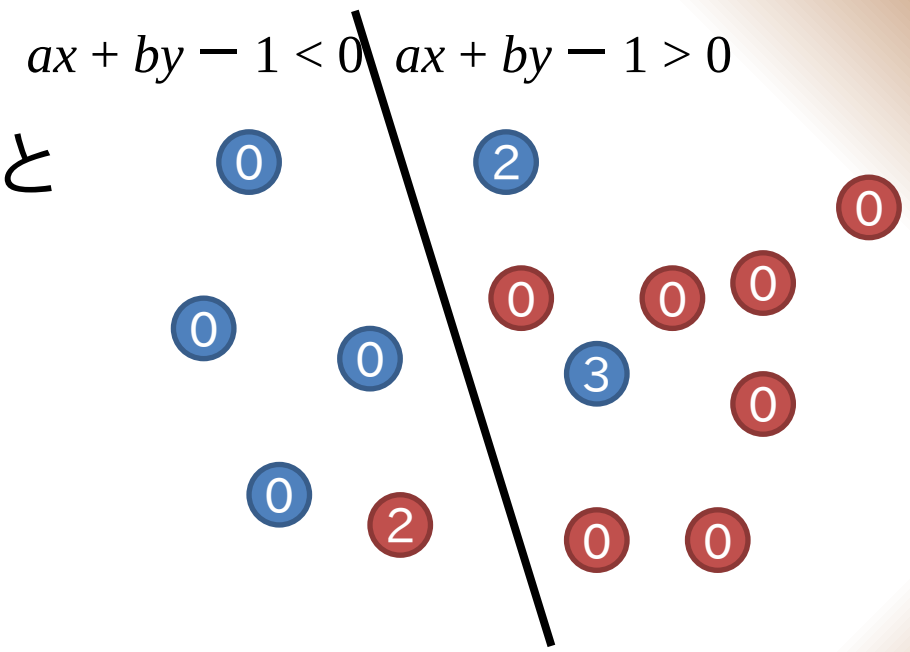
ここで書いた式だと…**凸関数**!



2変数以上の凸関数を最小化する例(3/4)

与えられた点を
$\{(p_1, q_1), \cdots, (p_n, q_n)\}$ と書くと
最小化したい式 $f(a, b)$ は
こう書ける

$$f(a, b)$$
$$= \Sigma^{n}_{i=1}\{[c_i(ap_i + bq_i - 1)]_+\}$$



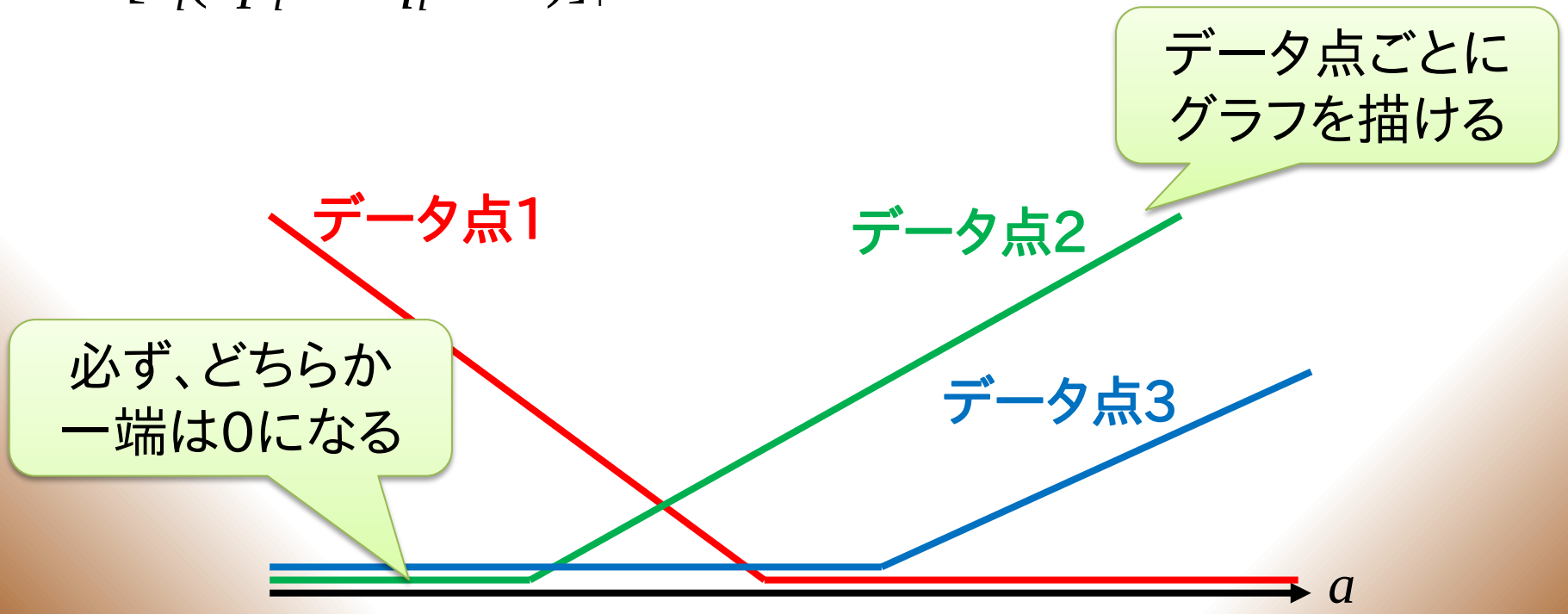
- c_iは、点iが赤なら−1、点iが青なら1を返す
- $(T)+$ は、Tが正のときはその値そのもの、Tが負のときは0を返す

計算してみます

- アイデアとしては
  「aを変化させて関数値を最小化」
  「bを変化させて関数値を最小化」
  「aを変化させて関数値を最小化」
  …と繰り返しています
  実際は、「双対問題」という形に変形して解くことが多く、その方が効率的なのですが、かなり追加の説明が必要なので扱いません
- 式が単純なので、「変数を一つ変更したときの最小値」は、ニュートン法のような方法を使わずとも求められてしまいます（どちらかといえば力技）

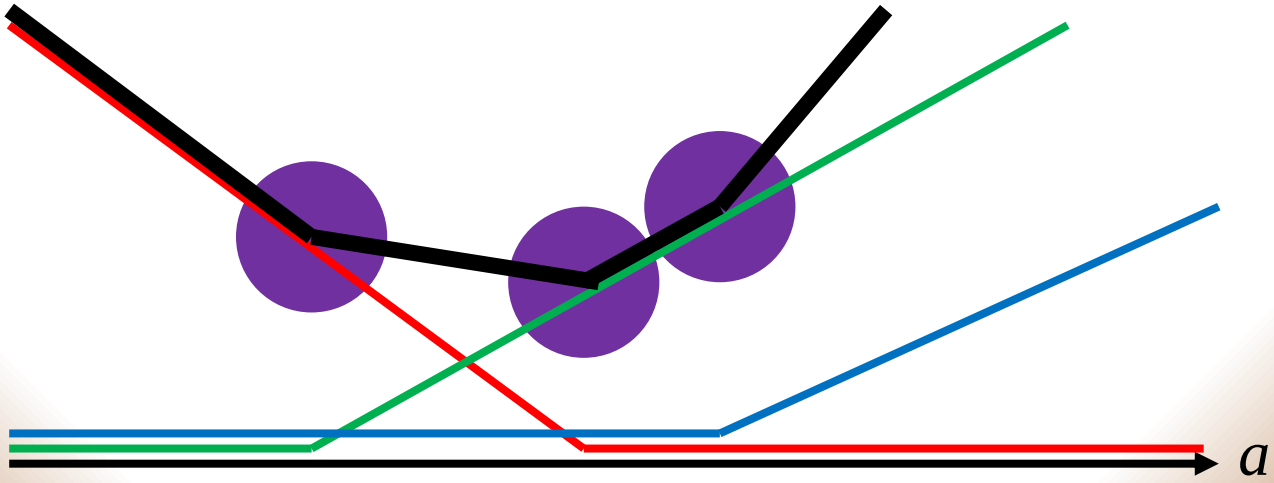
計算してみます

- 1変数ずつ最小化する、という前提であったので aについて最小化する場合を考える
- $[c_i(ap_i + bq_i - 1)]_+$ は下のような形をしている



計算してみます

- これを全データについて足すとこんな形になる
- それぞれのデータ点の「曲がり角」を覚えておけば
  その中で値が最小のものを選べばよい
  曲がり角になる条件は $c_i(ap_i + bq_i - 1) = 0$
  ただしこのほかに「曲がり角一覧をソートする」という処理も入るので、
  データ数が増えると重いです。今回は例なので単純な方法を使います。

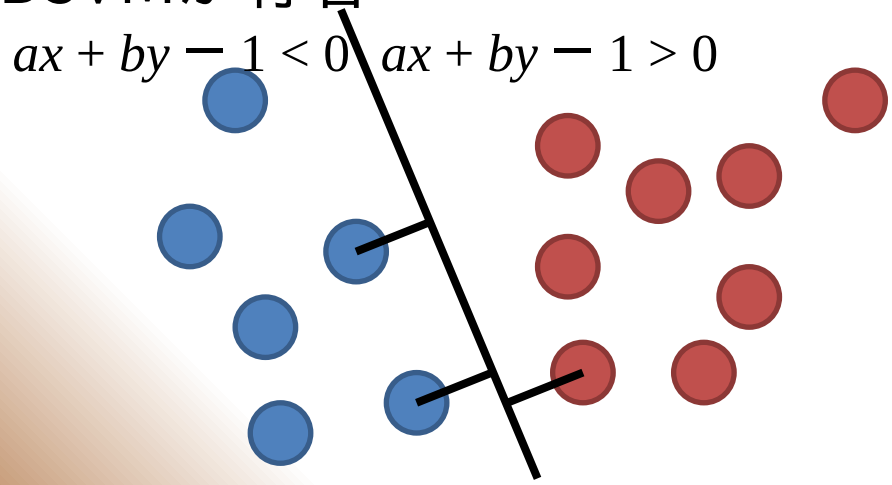


サンプルコードをお見せします
＆実際に動かしてみます

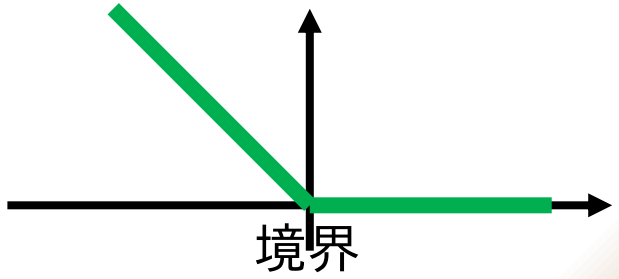
2変数以上の凸関数を最小化する例(補足)

「境界線からの距離」を基準にすると、実はサポートベクトルマシン（SVM）に近い計算をしていることになる

- 十数年前に話題になった機械学習手法。アイデアとしては「境界線がいかに二つのクラスから離れているか」
- 高速&わかりやすい実装としてはLIBSVMが有名



今回の問題設定



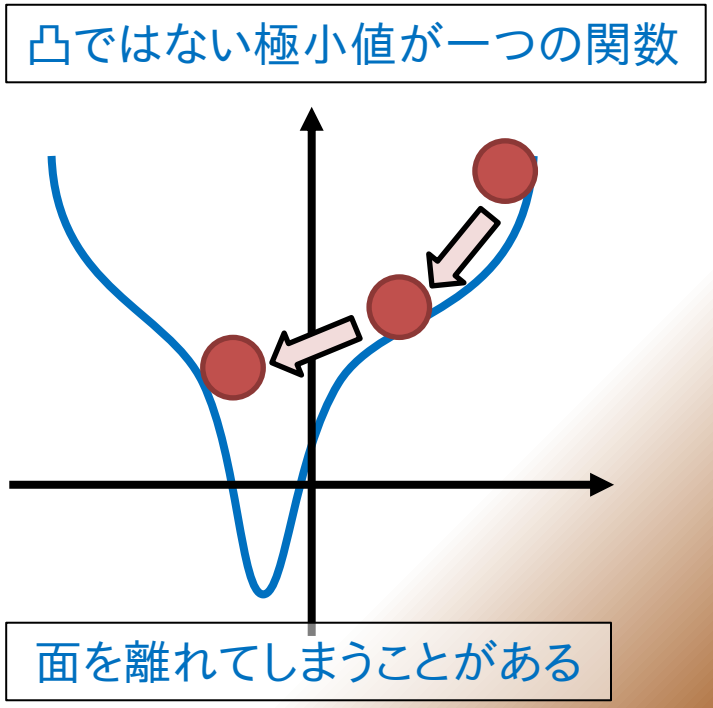
SVM

6.
他の最小化が難しい関数と、その対策

凸関数ではないけど
極小値が一つだとわかっていたら？

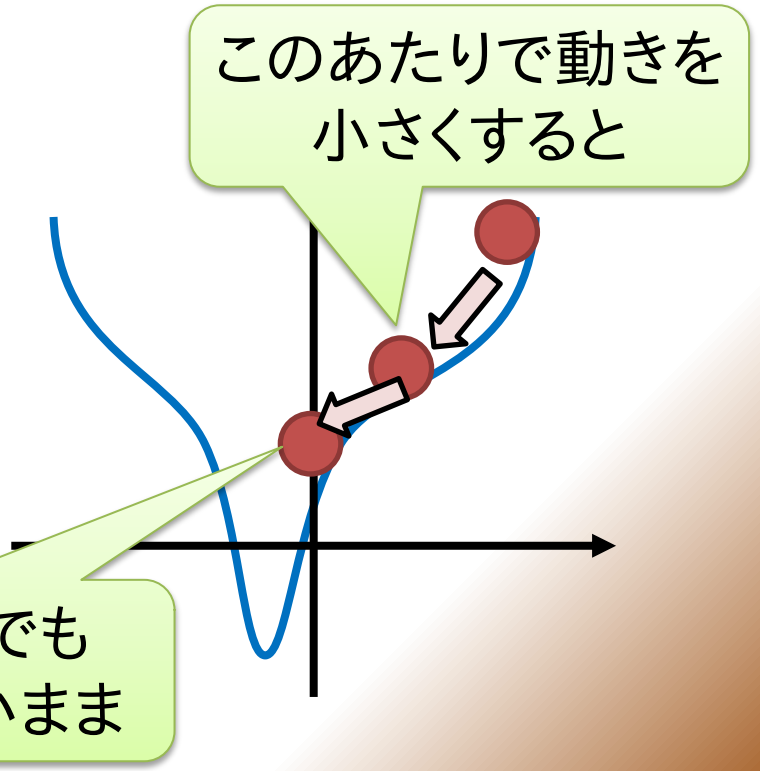
凸関数ではないけど極小値が一つの場合の最小化

- 基本的には、傾きを見て下がるほうに移動すればよいのだが、「効率が悪い」とか「収束が保証されない」とかの問題がある
- 何が問題かというと、移動幅（矢印の幅）を事前に決められない
- 徐々に小さくしていくという作戦はよく用いられる（ただし万能ではない）



凸関数ではないけど極小値が一つの場合の最小化

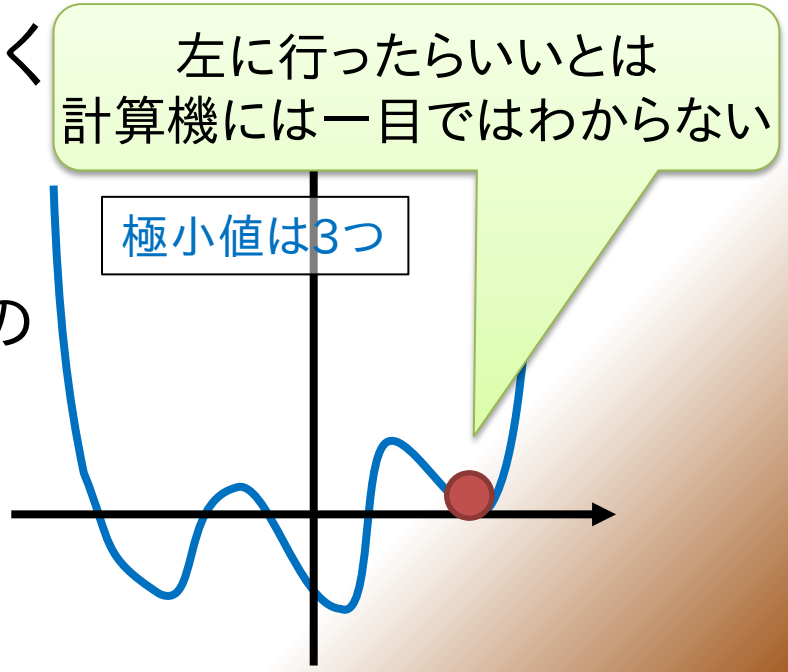
- 基本的には、傾きを見て下がるほうに移動すればよいのだが、「効率が悪い」とか「収束が保証されない」とかの問題がある
- 何が問題かというと、移動幅（矢印の幅）を事前に決められない
- 徐々に小さくしていくという作戦はよく用いられる（ただし万能ではない）



そもそも極小値が一つとは
限らなければ？

極小値が一つとは限らない関数の最小化

- どうしようもない
  （厳密に言えば、「無条件に正しい答えを見つける方法は存在しない」）ので、
  - さっきの分類問題のように、凸関数などで近似する
  - 最小値という保証なしに解く
- 基本的には、トライ&エラー
  いろんな場所に動いてみる
- いろんな場所にトライすると、未知の最小値を見つけられる可能性は高まる一方、計算時間も増える
- 「焼きなまし法」などが有名



7.
まとめ

まとめ

- 「関数の最小化」の特徴を知っておくと
  機械学習などの問題を解くのに役立つ。
- 関数を立てたら、それが最小化しやすい関数なのか
  判断することで、どんな実装にするかの方針が立つ。
  特に以下が大事。
  - 「極小値が1つであるか否か」
  - 「1つであるならば、それは凸関数か」
- 時には解けることを最優先に、厳密さを失ってでも
  凸関数に置き換えたりする。機械学習でもよくある。