汎用自動チューニング機構を実現するためのソフトウェア基盤の研究

須田礼仁

2012/12/25

科研費の概要

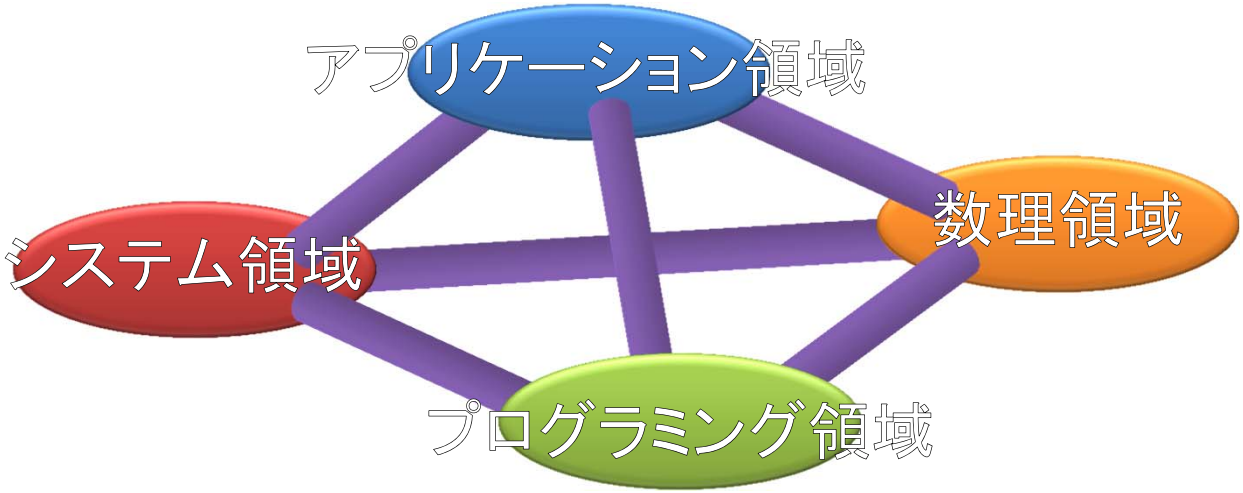
- **汎用自動チューニング機構を実現するためのソフトウェア基盤の研究**

- 基盤研究(A)
- 平成23年度～平成25年度

- http://olab.is.s.u-tokyo.ac.jp/~reiji/kiban_a/

全体の方針

- 汎用の自動チューニングのために研究推進が必要な４つの領域を設定



- 各領域に，自動チューニングおよび関連分野の重要な研究者を招聘

メンバー

数理領域

須田礼仁（東大・代表）
弓場敏嗣（電通大・協力）

プログラミング領域

佐藤周行（東大・分担）
片桐孝洋（東大・連携）
八杉昌宏（京大・協力）

アプリケーション領域

山本有作（神戸大・分担）
美添一樹（東大・分担）
藤井昭宏（工学院大・連携）
玉田嘉紀（東大・協力）

システム領域

今村俊幸（電通大・分担）
鴨志田良和（東大・分担）
大澤範高（千葉大・協力）

自動チューニング数理基盤ライブラリ
ATMathCoreLib 入門

ATMathCoreLib とは

- 自動チューニングのための数理技術として開発してきたアルゴリズムを実装

- 一般公開，利用自由
- http://olab.is.s.u-tokyo.ac.jp/~reiji/atmathcorelib/

- Scilab バージョン，C バージョンがある
- C の 2012/12/25 版の使い方を説明します

想定されている場面

- 変種とチューニングパラメタは実装済み
- チューニングパラメタの値は離散的で有限個
  - どのパラメタ値でも正しく動く
- 特徴量はない

- コストは測定可能，ばらつき（正規分布）がある
- 性能モデルがある or ない

- オンライン or オフライン自動チューニング
  - 反復的な計算

想定されていない場面

- チューニングパラメタが連続的
  - 離散化してよければ OK
- チューニングパラメタ値によっては停止しない
- コストにばらつきがまったくない
  - つまり，同じパラメタだと同一のコストになる
- 特徴量がある
  - 現状の ATMathCoreLib で可能な手段
    - 特徴量ごとにチューニングする
    - 特徴量の効果（トレンド）を消去する

ATMathCoreLib の機能

- 基本統計：試行回数・平均・分散
- 線形モデルフィッティング・分散推定
- ベイズ推定による性能推定
- 自動チューニングのための実験計画
  - どの選択肢を使って次の実行をするか？
    - オンライン自動チューニング（分散既知・未知）
    - オフライン自動チューニング
    - 選択肢変更コストを伴うオンライン AT
    - 確率的候補選択
- チューニングメーター
  - どのぐらい最適に近いところまで来ているか？

ATMathCoreLib 入門（1）

モデルなしの簡易実験計画

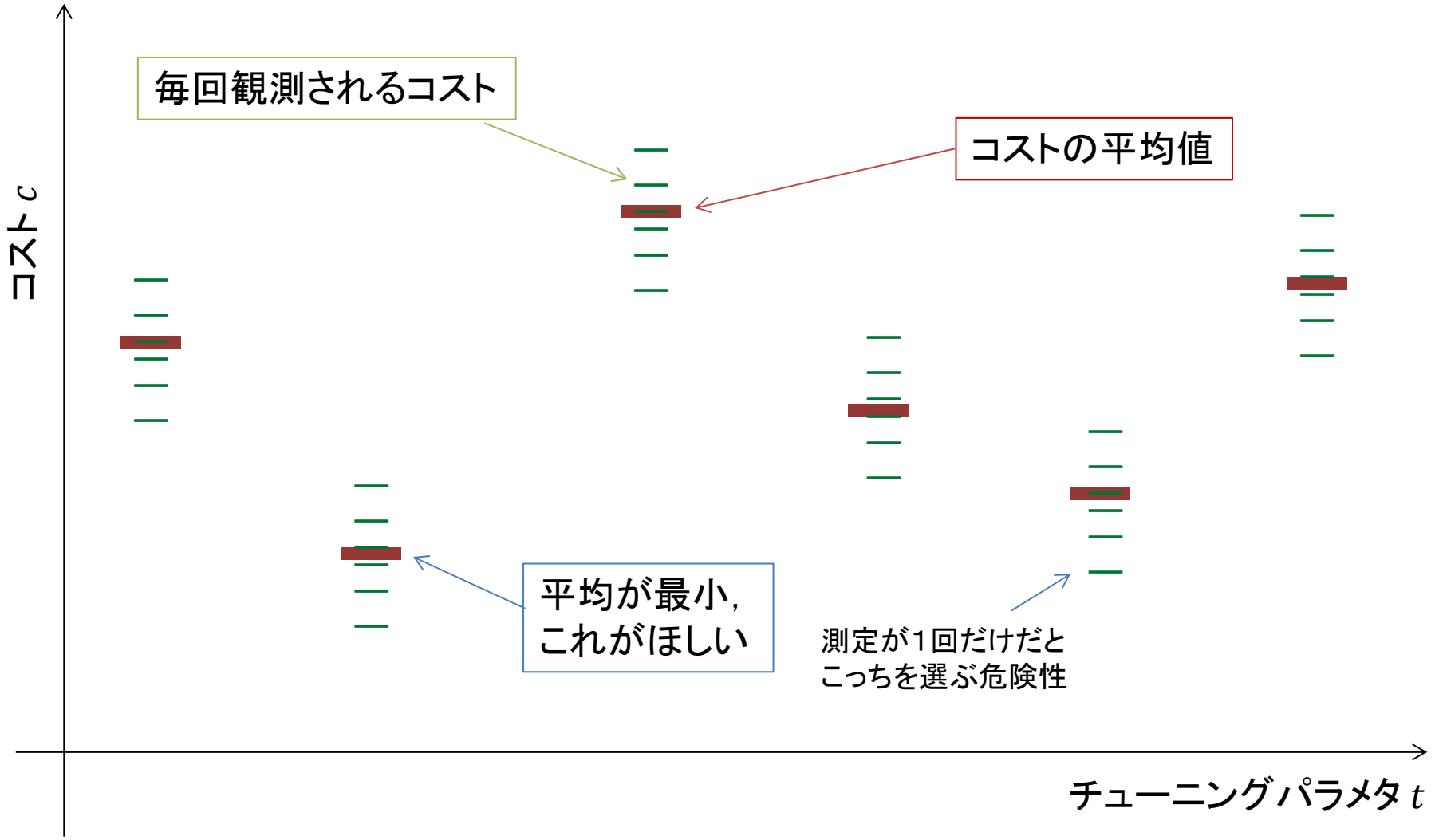
想定する問題

- ある計算を K 回繰り返す
- 反復ごとにコスト（有限）がかかる
- コストはばらつきがあり，1 回の測定ではコストが確定しない

- チューニングパラメタがあって，M 個の選択肢がある
  - どの選択肢のコストが少ないか，選択肢とコストの関係について，事前情報はない

- トータルのコストを最小にしたい

イメージ図


毎回観測されるコスト
コストの平均値
コスト c
平均が最小，
これがほしい
測定が1回だけだと
こっちを選ぶ危険性
チューニングパラメタ t

使い方

もとのユーザーコード

```
// func_param は動作を決める
// tune_param は性能を決める
// 0 <= tune_param < M とする
void function(func_param, tune_param) {
  主要な計算;
}

void main() {

  for (i = 0; i < K; i++) {
    function(fp_value, tp_value);
    // cost を測る手段 get_cost() がある
  }

}
```

使い方

もとのユーザーコード

```
// func_param は動作を決める
// tune_param は性能を決める
// 0 <= tune_param < M とする
void function(func_param, tune_param) {
  主要な計算;
}

void main() {

  for (i = 0; i < K; i++) {
    function(fp_value, tp_value);
    // cost を測る手段 get_cost() がある
  }

}
```

改変後のユーザーコード

```
void function(func_param, tune_param) {
  主要な計算;
}

void main() {
  x = new_exdesign(M, 0.0);
  for (i = 0; i < K; i++) {
    // 残り実行回数 K – i – 1
    // 次に選ぶ選択肢 tp
    tp = exdes_nomodel_online(x, K-i-1);

    function(fp_value, tp);
    // コストは get_cost() で得られる
    update_exdes(x, tp, get_cost());
  }
}
```

使い方

exdesign_t *new_exdesign(m, invk);

実験計画のための新しいデータ構造を返す（初期化ずみ）

引数
int m; 選択肢の個数
double invk;（オフライン時に使用）

返り値
実験計画のための新しいデータ構造

改変後のユーザーコード

```
void function(func_param, tune_param) {
  主要な計算;
}

void main() {
  x = new_exdesign(M, 0.0);
  for (i = 0; i < K; i++) {
    // 残り実行回数 K – i – 1
    // 次に選ぶ選択肢 tp
    tp = exdes_nomodel_online(x, K-i-1);

    function(fp_value, tp);
    // コストは get_cost() で得られる
    update_exdes(x, tp, get_cost());
  }
}
```

使い方

int exdes_nomodel_online(x, r)

オンライン自動チューニングの実験計画に従い，次に選ぶべき選択肢を返す

引数
exdesign_t *x; データ構造
int r; 残り実行回数．r < 0 だと，デフォルト値（これまでの実行回数）が使われる

返り値 int
次に選ぶべき選択肢のインデクス

注
残り実行回数がわからない場合，r = -1 とすることができる．ただし，残り実行回数がわかっている場合に比べ，効率は落ちる．

改変後のユーザーコード

```
void function(func_param, tune_param) {
  主要な計算;
}

void main() {
  x = new_exdesign(M, 0.0);
  for (i = 0; i < K; i++) {
    // 残り実行回数 K – i – 1
    // 次に選ぶ選択肢 tp
    tp = exdes_nomodel_online(x, K-i-1);

    function(fp_value, tp);
    // コストは get_cost() で得られる
    update_exdes(x, tp, get_cost());
  }
}
```

使い方

void update_exdes(x, t, c)

観測されたコストを記録する.

引数
exdesign_t *x; データ構造
int t; 選ばれた選択肢のインデクス
double c; 観測されたコスト

返り値
なし

改変後のユーザーコード

```
void function(func_param, tune_param) {
  主要な計算;
}

void main() {
  x = new_exdesign(M, 0.0);
  for (i = 0; i < K; i++) {
    // 残り実行回数 K – i – 1
    // 次に選ぶ選択肢 tp
    tp = exdes_nomodel_online(x, K-i-1);

    function(fp_value, tp);
    // コストは get_cost() で得られる
    update_exdes(x, tp, get_cost());
  }
}
```

サンプルプログラム

- sample_exdes_nomodel.c
  - 最初のほうにある定義を以下のように設定

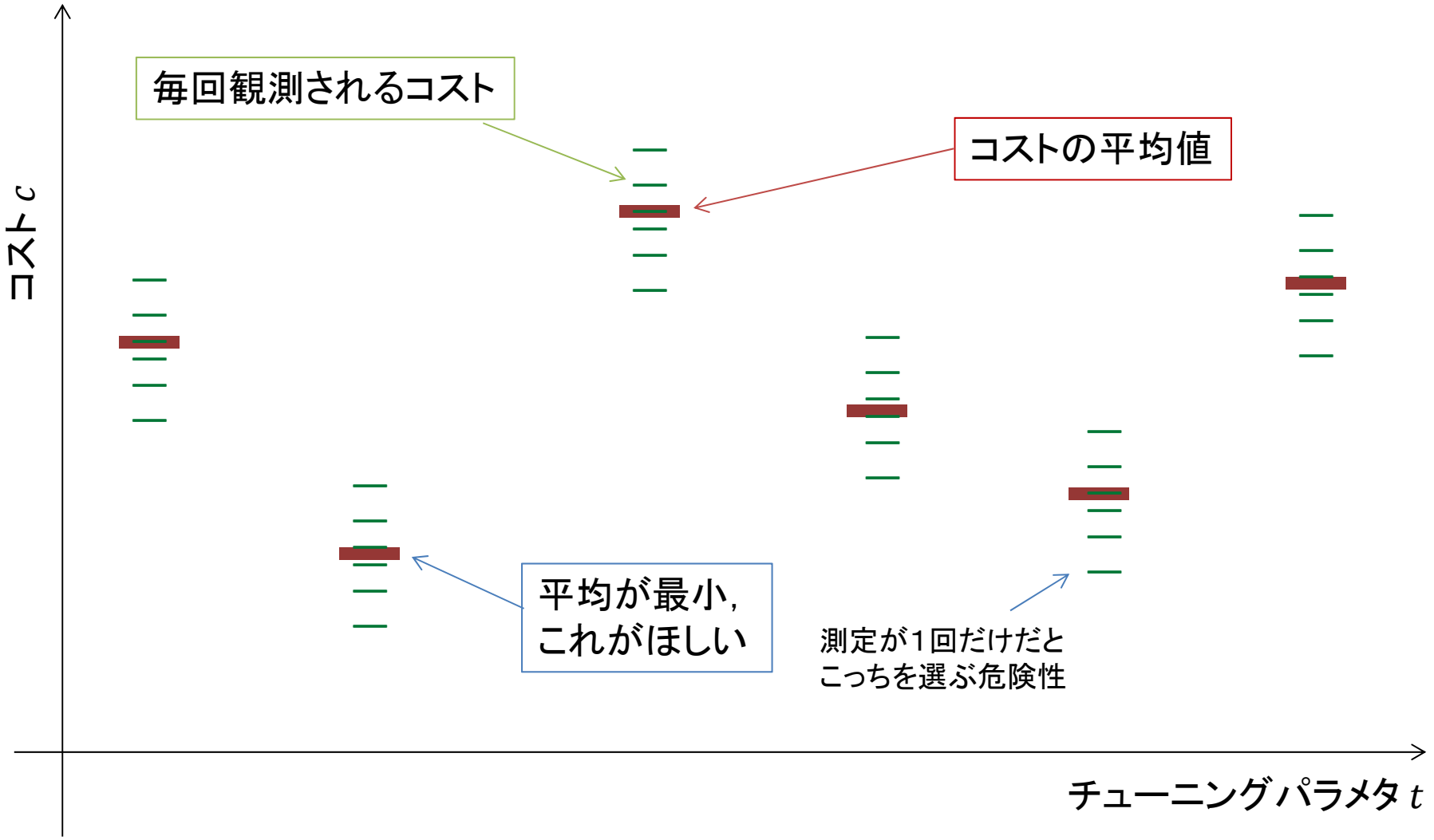
    #define OFFLINE 0

    #define TRUE_REM 1
- M = 100 個の選択肢
  - コストは正規分布で作成
- 全実行回数 K = 1000 回

選択肢 t の平均コスト c_t は，平均 5，分散 1 の正規分布で作成

選択肢 t の j 回目の実行コスト c_{tj} は，平均 c_t，分散 0.1 の正規分布で作成

イメージ図（再掲）


毎回観測されるコスト
コストの平均値
コスト c
平均が最小，
これがほしい
測定が1回だけだと
こっちを選ぶ危険性
チューニングパラメタ t

出力

生成された各候補についての
コストの平均値と分散

```
cand   0: mu = 6.197019, var = 0.100000
cand   1: mu = 5.152732, var = 0.100000
cand   2: mu = 5.838434, var = 0.100000
cand   3: mu = 3.668651, var = 0.100000
cand   4: mu = 3.778930, var = 0.100000
cand   5: mu = 4.561337, var = 0.100000
cand   6: mu = 4.924627, var = 0.100000
cand   7: mu = 7.280149, var = 0.100000
cand   8: mu = 5.535411, var = 0.100000
…
cand  97: mu = 4.741761, var = 0.100000
cand  98: mu = 7.333109, var = 0.100000
cand  99: mu = 5.945253, var = 0.100000
```

各実行につき，選択肢とコスト，
選択肢の平均コスト（本来未知）

```
n =    0,  99th cand, value = 5.498407, tru mean = 5.945253
n =    1,  99th cand, value = 6.166772, tru mean = 5.945253
n =    2,  99th cand, value = 5.844730, tru mean = 5.945253
n =    3,   5th cand, value = 4.466517, tru mean = 4.561337
n =    4,   5th cand, value = 4.286948, tru mean = 4.561337
n =    5,   5th cand, value = 4.438112, tru mean = 4.561337
n =    6,  31th cand, value = 5.412936, tru mean = 4.965538
n =    7,  31th cand, value = 4.881156, tru mean = 4.965538
n =    8,  31th cand, value = 5.116117, tru mean = 4.965538
…
n =  997,  53th cand, value = 3.001980, tru mean = 2.462269
n =  998,  53th cand, value = 2.550384, tru mean = 2.462269
n =  999,  53th cand, value = 2.474955, tru mean = 2.462269
```

観測で最小コストだった選択肢，真の最小コストの選択肢，全コスト，リグレット，ロス

```
obs best: 53th obs num = 884 mean = 2.473630 tru mean = 2.462269 var = 0.100000
tru best: 53th obs num = 884 mean = 2.473630 tru mean = 2.462269 var = 0.100000
total 2736.038471, average 2.736038, regret 273.769142 (0.273769), loss 0.000000
```

出力

生成された各候補についての
コストの平均値と分散

```
cand   0: mu = 6.197019, var = 0.100000
cand   1: mu = 5.152732, var = 0.100000
cand   2: mu = 5.838434, var = 0.100000
cand   3: mu = 3.668651, var = 0.100000
cand   4: mu = 3.778930, var = 0.100000
cand   5: mu = 4.561337, var = 0.100000
cand   6: mu = 4.924627, var = 0.100000
cand   7: mu = 7.280149, var = 0.100000
cand   8: mu = 5.535411, var = 0.100000
…
cand  97: mu = 4.741761, var = 0.100000
cand  98: mu = 7.333109, var = 0.100000
cand  99: mu = 5.945253, var = 0.100000
```

乱数を使って生成した，仮想の「選択肢」と
そのコストの平均・分散．（実用的な場面では
これらの平均・分散は未知．ここでは確認の
ため表示している．）

平均値 mu (c_t) は，平均 5 分散 1 の正規
分布に従って生成されている．

つまり，ランダムに選んで K = 1000 回実行
すると，コストの期待値は 5000 となるはず．

選択肢を固定して実行すると，コストは平均
mu, 分散 0.1 の正規分布に従う．

出力

各実行について，
・選ばれた選択肢
・その時の観測コスト
・選択肢の平均コスト

観測コストは，平均コストから
すこしずれている．

最初の 12 回は 4 つの選択肢を
3 回ずつ実行（分散を測定）．

最後のほうは，ほぼコスト最小
のものを実行する．

各実行につき，選択肢とコスト，
選択肢の平均コスト（本来未知）

```
n =   0,  99th cand, value = 5.498407, tru mean = 5.945253
n =   1,  99th cand, value = 6.166772, tru mean = 5.945253
n =   2,  99th cand, value = 5.844730, tru mean = 5.945253
n =   3,   5th cand, value = 4.466517, tru mean = 4.561337
n =   4,   5th cand, value = 4.286948, tru mean = 4.561337
n =   5,   5th cand, value = 4.438112, tru mean = 4.561337
n =   6,  31th cand, value = 5.412936, tru mean = 4.965538
n =   7,  31th cand, value = 4.881156, tru mean = 4.965538
n =   8,  31th cand, value = 5.116117, tru mean = 4.965538
…
n = 997,  53th cand, value = 3.001980, tru mean = 2.462269
n = 998,  53th cand, value = 2.550384, tru mean = 2.462269
n = 999,  53th cand, value = 2.474955, tru mean = 2.462269
```

出力

obs best: 観測された中で「コストが最小」と判定された選択肢.
  obs num: 観測（実行）された回数. 1000 回中 884 回これが使われた.
  mean: 観測された平均コスト.
  tru mean, var: コストの真の平均と分散.

tru best: コストの真の平均が最小の選択肢. 今回は obs best と同じ.

total, average: 1000 回の実行のコストの総和と平均.
  ランダムに選べば 5000 (5.000) のところを，2736 (2.736) でできた.

観測で最小コストだった選択肢，真の最小コストの選択肢，全コスト，リグレット，ロス

```
obs best: 53th obs num = 884 mean = 2.473630 tru mean = 2.462269 var = 0.100000
tru best: 53th obs num = 884 mean = 2.473630 tru mean = 2.462269 var = 0.100000
total 2736.038471, average 2.736038, regret 273.769142 (0.273769), loss 0.000000
```

出力

regret:「もし，コストの真の平均が最小の選択肢があらかじめ分かっていて，
　最初からそれを 1000 回実行した場合」に比べてどれだけ余計なコストが
　かかったか．
これが，「100 個の中からよいものを選ぶのにかかったコスト」に相当．
合計コストを最小化するのは，regret を最小化するのと等価．

loss: obs best と tru best の性能差．今回はゼロ．

loss はゼロである必要はない．loss を小さくしようとすると，実験のコストが
かかってしまう．実験のコストを抑えようとすると，loss が大きくなる．これの
最適なバランスを取ることが重要．

観測で最小コストだった選択肢，真の最小コストの選択肢，全コスト，リグレット，ロス

```
obs best: 53th obs num = 884 mean = 2.473630 tru mean = 2.462269 var = 0.100000
tru best: 53th obs num = 884 mean = 2.473630 tru mean = 2.462269 var = 0.100000
total 2736.038471, average 2.736038, regret 273.769142 (0.273769), loss 0.000000
```

Regret と loss のバランス

	Regret	Loss
0	0.261	0.000
1	0.415	0.339
2	0.163	0.000
3	0.091	0.000
4	0.432	0.323
5	0.285	0.000
6	0.242	0.000
7	0.241	0.000
8	0.481	0.459
9	0.057	0.000
	0.267	**0.112**

Regret と Loss の比が 2:1 ぐらいが理想的

この実験は，まあそれに近い．

残り実行回数を未知にすると

残り実行回数既知（rk = K – i – 1）

	Regret	Loss
0	0.261	0.000
1	0.415	0.339
2	0.163	0.000
3	0.091	0.000
4	0.432	0.323
5	0.285	0.000
6	0.242	0.000
7	0.241	0.000
8	0.481	0.459
9	0.057	0.000
	0.267	**0.112**

残り実行回数未知（rk = – 1）

	Regret	Loss
0	0.552	0.000
1	1.370	1.148
2	1.837	0.000
3	0.204	0.000
4	0.594	0.000
5	1.631	1.588
6	0.360	0.000
7	0.108	0.000
8	0.480	0.459
9	0.226	0.000
	0.736	**0.320**

残り回数未知だと，リグレットが倍ぐらい悪くなることが多い

残り回数がなぜ重要か

残り実行回数が既知

- 最初のうちは，「後々の実行のために，どれがコストが小さいか，今のうちに調べておこう」
- 最後のほうは，「残り回数が少ないので，これ以上調べるのはやめよう．今一番コストが小さいと思うものを使おう」

残り実行回数が未知

- 最初のうちは，「このソフトはあまり使われないかもしれない．あまり詳しく比較せず，今コストが小さいと思うものをとりあえず使おう」
- 最後のほうは，「これは結構使われるソフトだ．どれがコストが小さいか，詳しく調べよう」

オフライン自動チューニング

想定する問題

- ある計算をたくさん繰り返す
- 反復ごとにコスト（有限）がかかる
- コストはばらつきがあり，1 回の測定ではコストが確定しない

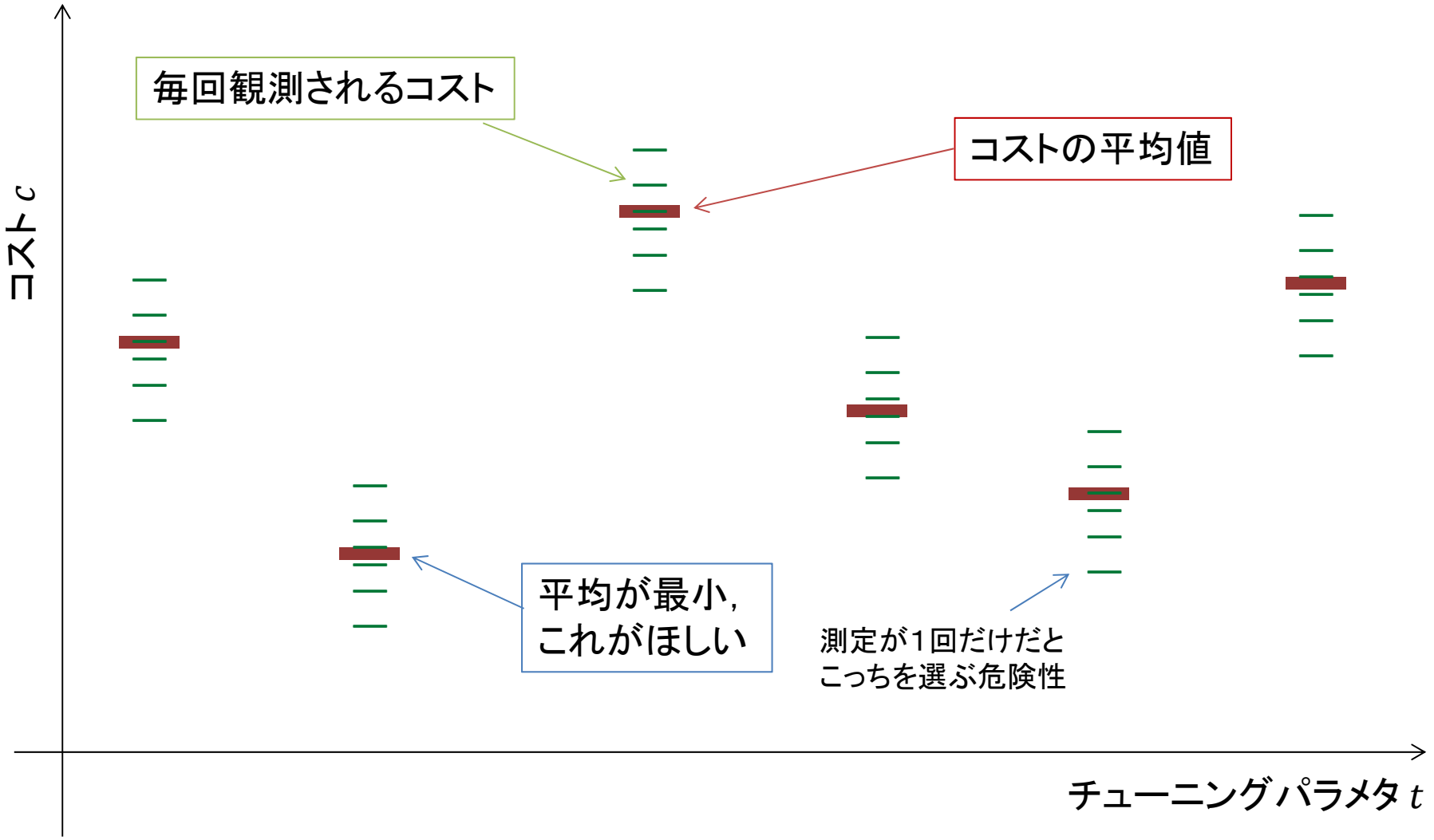
- チューニングパラメタがあって，M 個の選択肢がある
  - どの選択肢のコストが少ないか，選択肢とコストの関係について，事前情報はない

- 事前にどれが低コストか調べておきたい

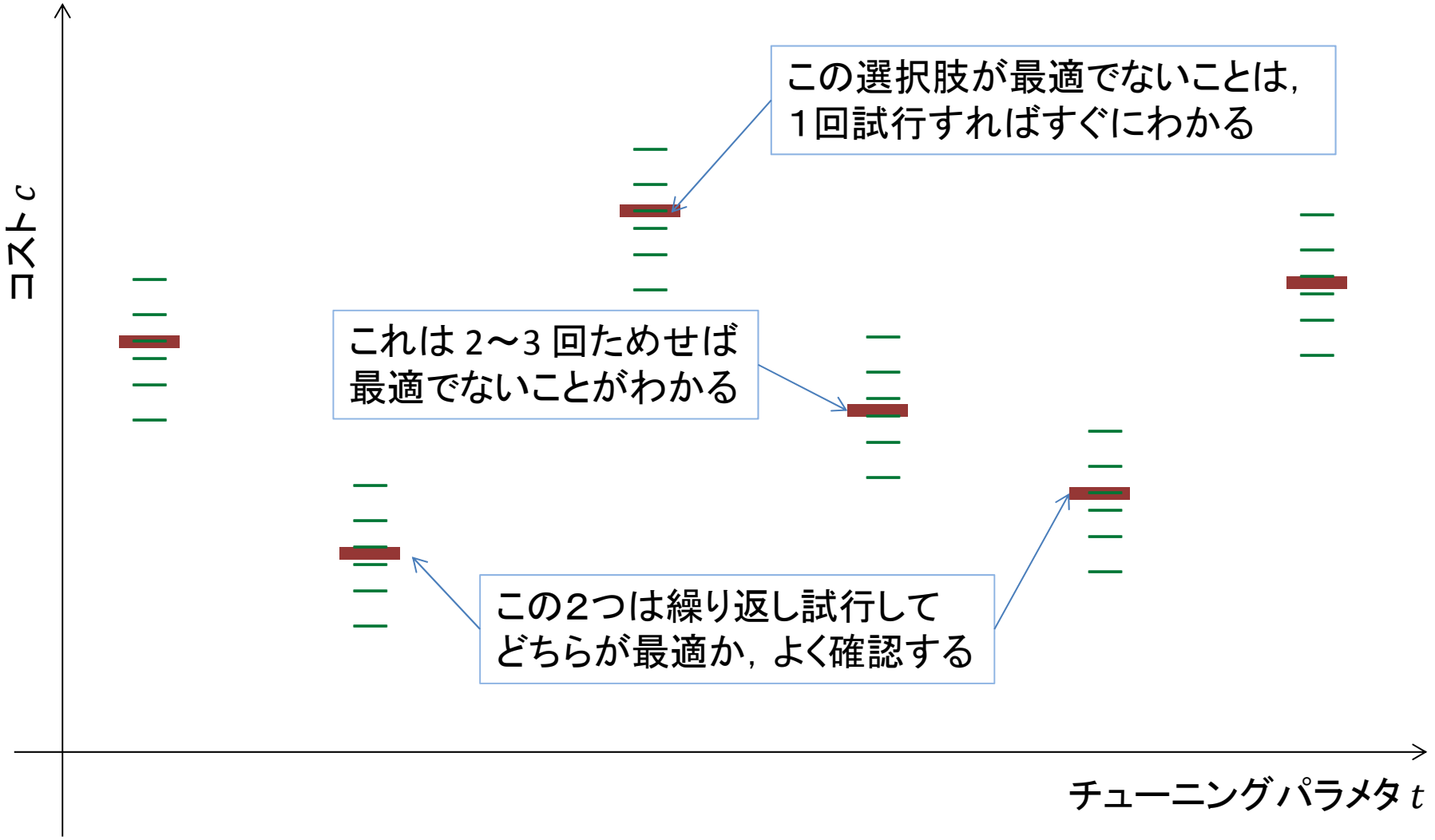
オフライン自動チューニング

- 事前にどれが低コストか調べておきたい
  - すべての選択肢について，十分な回数だけ試行を反復して，コストの平均値を十分な精度で求められれば，それがよい
- 試行のコストを考慮すると．．．
  - 選択肢が沢山ある場合，非現実的かもしれない
  - 明らかに最適じゃない選択肢は，平均値を十分な精度で求める必要はないはず

イメージ図（再々掲）


毎回観測されるコスト
コストの平均値
コスト c
平均が最小，
これがほしい
測定が1回だけだと
こっちを選ぶ危険性
チューニングパラメタ t

イメージ図（再々掲）


コスト c
この選択肢が最適でないことは，
１回試行すればすぐにわかる
これは 2〜3 回ためせば
最適でないことがわかる
この２つは繰り返し試行して
どちらが最適か，よく確認する
チューニングパラメタ t

使い方

試行回数は上限があったほうがよい

invk には，非負の（通常はかなり小さい）実数を入れる．たとえば 1e-8 とか．

意味としては，「最適化の後で，実際に計算する回数」を K として，invk = 1/K.

invk = 0 も可（つまり $K = \infty$）.

関数の名前が … offline になっていて，「残り計算回数」の引数がない．

「だいたい最適解が求まった」ところで，tp >= M を返す．

invk = 0 にすると tp >= M は返さない．

tp – M を選んで続行することも可能．

どれが最適だと思っているかを返す．

改変後のユーザーコード

```
void main() {
  x = new_exdesign(M, invk);
  for (i = 0; i < max_iter; i++) {
    // 次に選ぶ選択肢 tp
    tp = exdes_nomodel_offline(x);

    if (tp >= M) break; // 試行終了

    function(fp_value, tp);
    // コストは get_cost() で得られる
    update_exdes(x, tp, get_cost());
  }

  j = exdes_getbest(exdes); // 得られた解
}
```

出力

生成された各候補についての
コストの平均値と分散

```
cand   0: mu = 3.470406, var = 0.100000
cand   1: mu = 3.740407, var = 0.100000
cand   2: mu = 4.952641, var = 0.100000
cand   3: mu = 4.521744, var = 0.100000
cand   4: mu = 5.507065, var = 0.100000
cand   5: mu = 4.681375, var = 0.100000
cand   6: mu = 4.129311, var = 0.100000
cand   7: mu = 7.416433, var = 0.100000
cand   8: mu = 6.745340, var = 0.100000
…
cand  97: mu = 5.612537, var = 0.100000
cand  98: mu = 5.491783, var = 0.100000
cand  99: mu = 4.242336, var = 0.100000
```

各実行につき，選択肢とコスト，
選択肢の平均コスト（本来未知）

```
n =   0,  88th cand, value = 5.678771, tru mean = 5.733646
n =   1,  88th cand, value = 5.634916, tru mean = 5.733646
n =   2,  88th cand, value = 5.783071, tru mean = 5.733646
n =   3,  81th cand, value = 4.826334, tru mean = 5.084664
n =   4,  81th cand, value = 4.579020, tru mean = 5.084664
n =   5,  81th cand, value = 4.933074, tru mean = 5.084664
n =   6,   2th cand, value = 4.558923, tru mean = 4.952641
n =   7,   2th cand, value = 5.408287, tru mean = 4.952641
n =   8,   2th cand, value = 4.651616, tru mean = 4.952641
…
n = 110,  24th cand, value = 3.372226, tru mean = 2.984829
n = 111,  37th cand, value = 3.678079, tru mean = 3.607037
n = 112,  25th cand, value = 3.119146, tru mean = 3.071027
```

最大 1000 回まで行くが，今回は 113 回目で停止した

観測で最小コストだった選択肢，真の最小コストの選択肢，全コスト，ロス

```
obs best: 9th obs num = 2 mean = 2.511607 tru mean = 2.743887 var = 0.100000
tru best: 9th obs num = 2 mean = 2.511607 tru mean = 2.743887 var = 0.100000
total 559.274642, average 4.949333, found best 9 2.743887, loss 0.000000
```

得られた「最適解」

	得られた解	真の最適解	Loss	試行回数	試行コスト
0	2.462	2.462	0.000	112	528.5
1	2.930	2.708	0.222	108	532.4
2	1.364	1.364	0.000	114	563.1
3	2.057	2.057	0.000	111	512.5
4	2.427	2.427	0.000	123	573.1
5	2.397	2.397	0.000	119	580.2
6	2.973	2.626	0.347	111	529.4
7	3.026	3.026	0.000	113	599.7
8	2.002	2.002	0.000	107	505.5
9	2.305	2.305	0.000	105	514.9

ロス/invk が試行コストと同程度がよい.
invk = 1e-8 の場合. この設定ではロスはほぼゼロが理想.

ATMathCoreLib の機能（再掲）

- 基本統計：試行回数・平均・分散
- 線形モデルフィッティング・分散推定
- ベイズ推定による性能推定
- 自動チューニングのための実験計画
  - どの選択肢を使って次の実行をするか？
    - オンライン自動チューニング（分散既知・未知）
    - オフライン自動チューニング
    - 選択肢変更コストを伴うオンライン AT
    - 確率的候補選択
- チューニングメーター
  - どのぐらい最適に近いところまで来ているか？

ありがとうございました