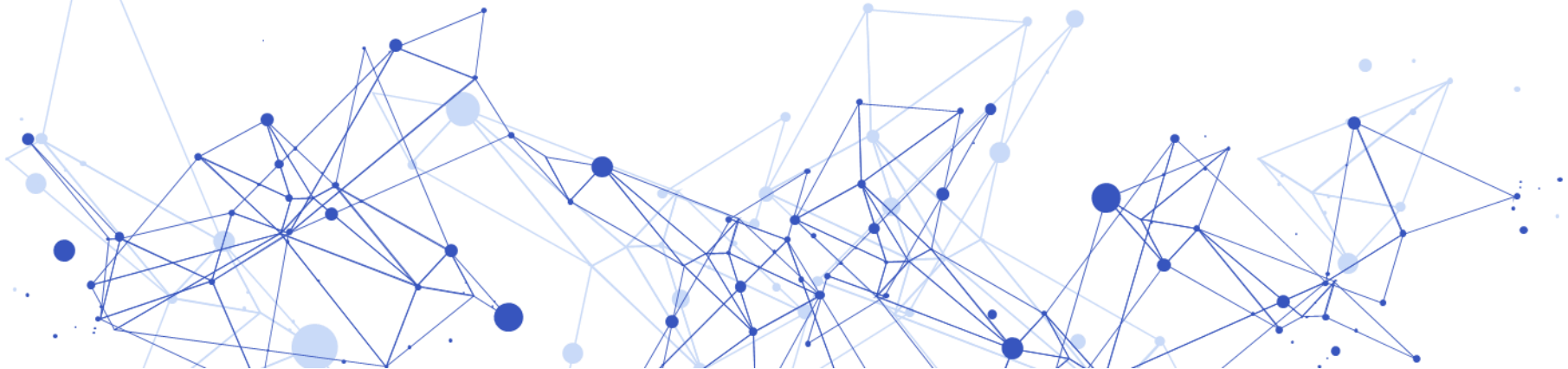
FIXSTARS Amplify

量子コンピュータ時代のプログラミングセミナー

～ブラックボックス最適化を活用した撹拌機器の設計・運転条件最適化～



本日の予定

第一部

- 本セミナーのゴール
- 会社紹介
- Fixstars Amplify の紹介
- 組合せ最適化事例
- ワークショップ事前準備

第二部

- 組合せ最適化の基本
  - 数の分割ハンズオン

第三部

- ブラックボックス最適化とは
- FMQAの概要とフロー
- FMQAによる設計最適化ハンズオン
  - 問題の説明
  - FM
  - FMQA
- Amplify-BBOpt
- まとめ

質問は随時 Zoom の Q&A へお願いします

会社紹介

フィックスターズグループの基本情報

会社名	株式会社フィックスターズ
本社所在地	東京都港区芝浦3-1-1 msb Tamachi 田町ステーションタワーN 28階
設立	2002年8月
上場区分	東証プライム（証券コード：3687）
代表取締役社長	三木　聡

資本金	5億5,446万円
社員数（連結）	292名（2023年9月現在）
主なお客様	キオクシア株式会社 ルネサスエレクトロニクス株式会社 トヨタグループ（トヨタ自動車株式会社・ 豊田通商株式会社・株式会社デンソー） みずほ証券株式会社 キヤノン株式会社

グループ会社

Fixstars Solutions, Inc.
完全子会社
米国での営業及び開発を担当

(株) Fixstars Autonomous Technologies
株式会社ネクスティ エレクトロニクスとのJV
自動運転向けソフトウェアを開発

2021/10/1 設立

(株) Fixstars Amplify
完全子会社
量子コンピューティングのクラウド事業を運営

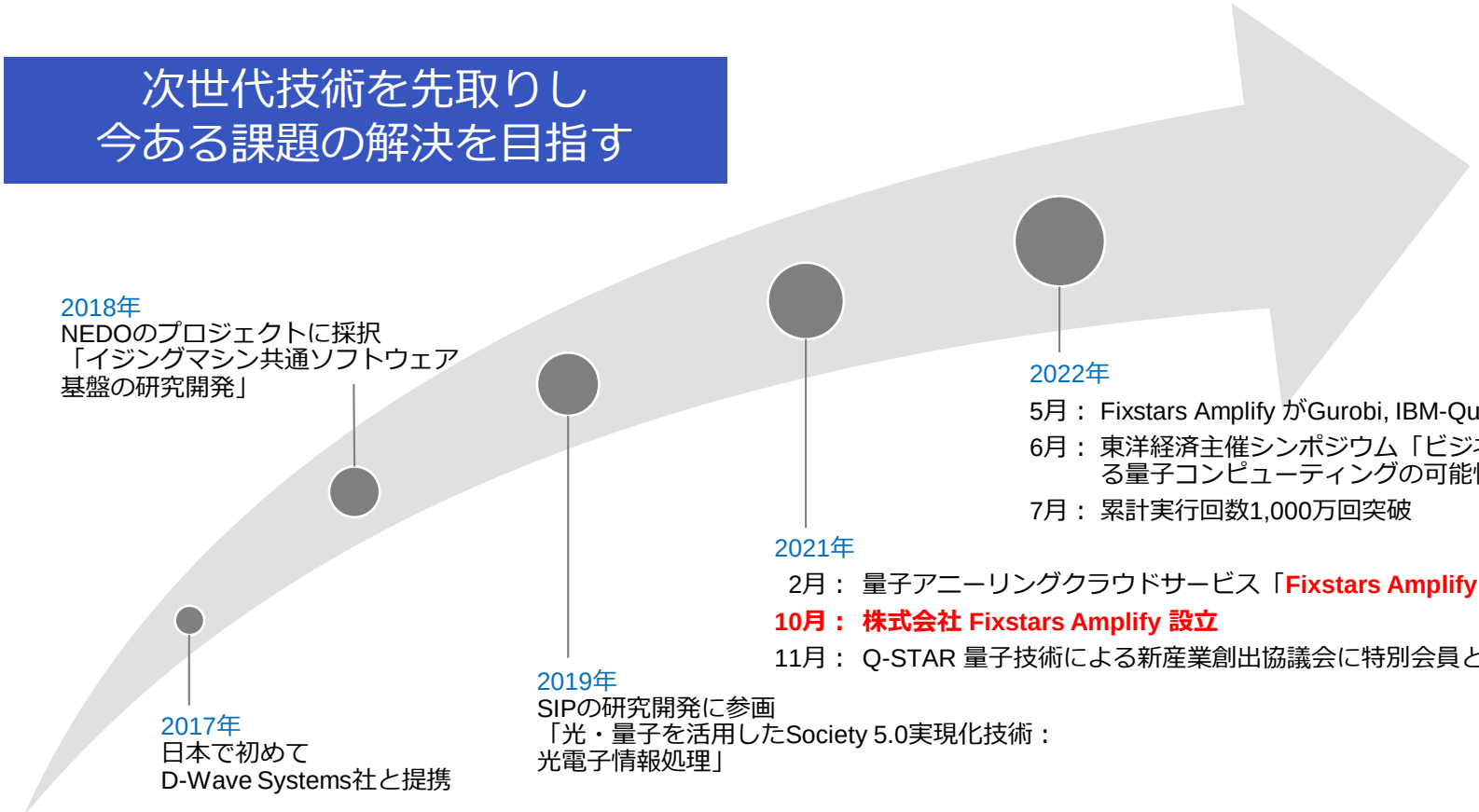
(株) Sider
完全子会社
開発支援SaaS「Sider」を運営

(株) Smart Opinion
連結子会社
乳がんAI画像診断支援事業を運営

オスカーテクノロジー (株)
連結子会社
ソフトウェア自動並列化サービスを提供

フィックスターズの量子技術への取り組み

次世代技術を先取りし
今ある課題の解決を目指す



2017年
日本で初めて
D-Wave Systems社と提携

2018年
NEDOのプロジェクトに採択
「イジングマシン共通ソフトウェア
基盤の研究開発」

2019年
SIPの研究開発に参画
「光・量子を活用したSociety 5.0実現化技術：
光電子情報処理」

2021年
2月： 量子アニーリングクラウドサービス「**Fixstars Amplify**」提供開始
10月： 株式会社 Fixstars Amplify 設立
11月： Q-STAR 量子技術による新産業創出協議会に特別会員として加入

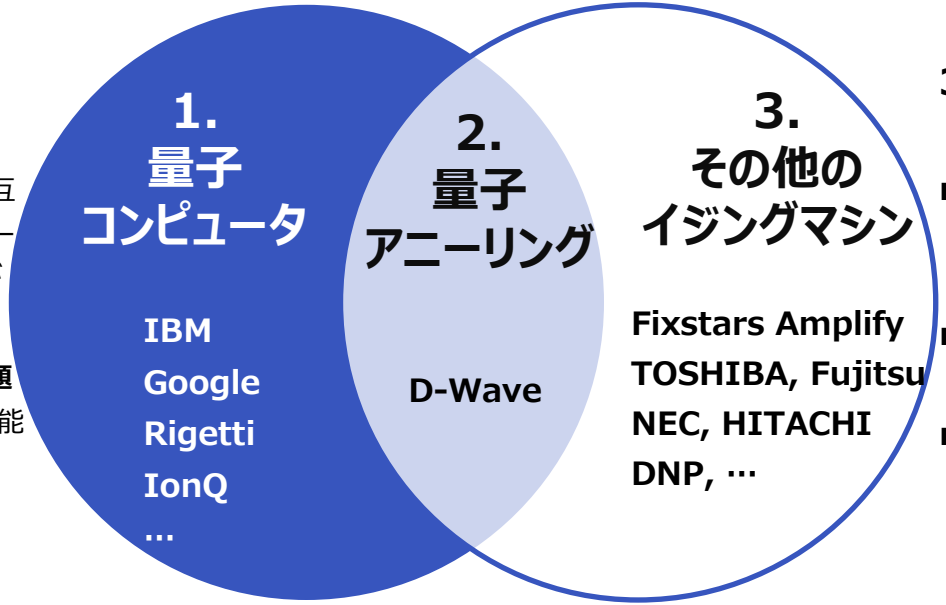
2022年
5月： Fixstars Amplify がGurobi, IBM-Quantumをサポート
6月： 東洋経済主催シンポジウム「ビジネスを劇的に変える量子コンピューティングの可能性」に登壇
7月： 累計実行回数1,000万回突破

量子技術とFixstars Amplify

量子・量子インスパイアード技術

1. 量子コンピュータ
（量子ゲート方式）

- 古典汎用コンピュータの上位互換。量子ゲートを操作。エラー訂正機能の無いNISQ型実機がクラウド利用可能
- QAOAにより**組合せ最適化問題**（**QUBO**）を取り扱うことが可能
- 演算規模：**～数100ビット**



3. その他のイジングマシン
（半導体技術に基づくイジングマシン）

- 二次の多変数多項式で表される目的関数の**組合せ最適化問題**（**QUBO**）専用マシン
- 統計物理学におけるイジング模型に由来。様々な実装により実現。
- 演算規模：
  260,000+ビット（*Amplify AE*）

2. 量子アニーリング（量子焼きなまし法式のイジングマシン）

- イジングマシンの一種。量子イジング模型を物理的に搭載したプロセッサで実現。
  量子効果を物理的に調整し、自然計算により低エネルギー状態が出力
- **組合せ最適化問題**（**QUBO**）を扱う専用マシン
- 演算規模：**～数1,000ビット**

最適化問題の分類

数理最適化問題

- 連続最適化問題
  - 決定変数が連続値（実数など）
- 決定変数が離散値 (整数など)
  - 整数計画問題 (決定変数が整数)
  - **0-1整数計画問題 (決定変数が二値)**

量子アニーリング・イジングマシン

Quadratic　二次形

Unconstrained　制約条件なし

Binary　0-1整数 (二値)

Optimization　計画（最適化）

QUBO目的関数 (0-1整数二次計画問題)

$$f(\boldsymbol{q}) = \sum_{i<j} Q_{ij} q_i q_j + \sum_i Q_{ii} q_i$$

f: 目的関数　q: 決定変数　Q: 係数

$f(q)$を最小化するような q を求める

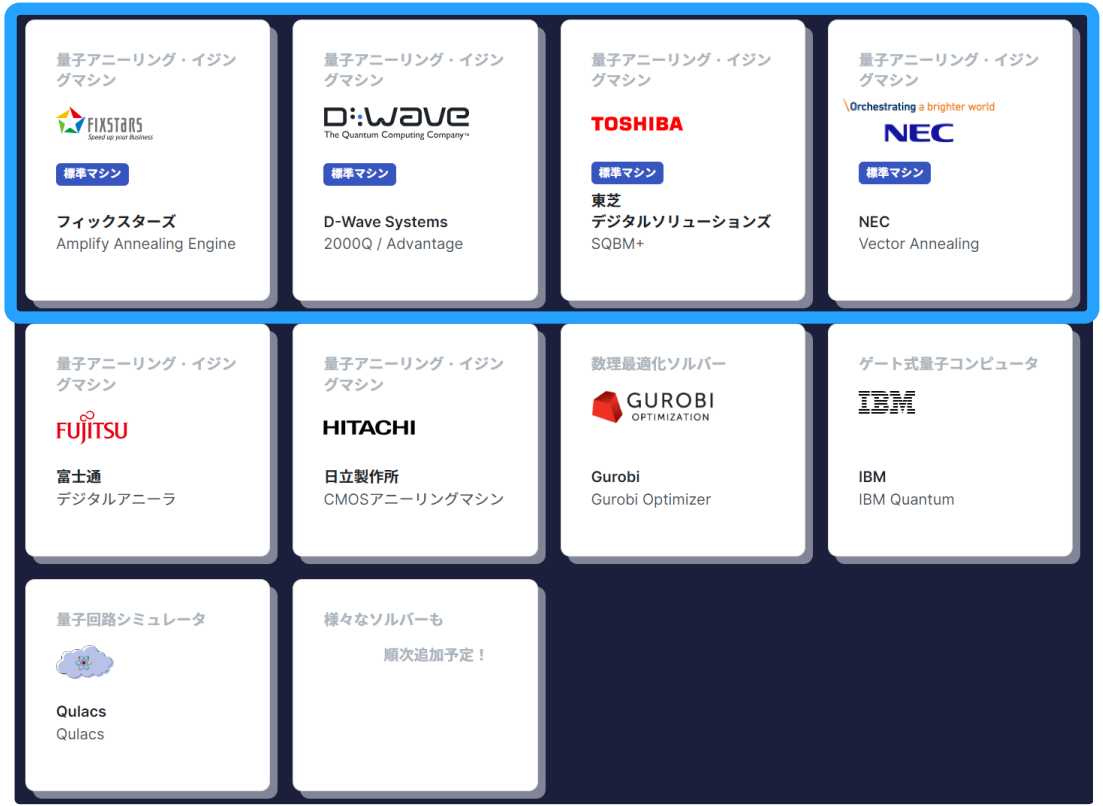
クラウドサービス：**Fixstars Amplify**

Fixstars Amplify とは

- **いつでも**　開発環境 と 実行環境 がセット
  すぐにアプリ開発と実行が出来る
- **誰でも**　ハードウェアや専門的な知識が不要
  無料で開発がスタート可能
  多くの解説、サンプルコード
- **高速に**　26万ビットクラスの大規模問題の
  高速処理と高速実行が可能
- **あらゆる**　一般に公開されている全てのイジング
  マシンを利用可能

FIXSTARS Amplify

Fixstars Amplify の対応マシンの一例



標準マシン は、

- ベンダ各社と個別マシン利用契約なし、
- 評価・検証用ベーシックプランなら無料、

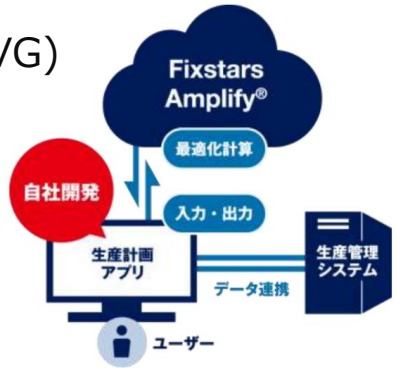
で利用可能！←「いつでも」、「誰でも」

今後も幅広い対応マシンの追加が続々と行われる予定です！←「あらゆる」

活用領域とユースケース（PoC・実稼働）

Amplify インタビュー　検索

- **生産計画**
  - 多品種少量生産、保全計画、設備投資、在庫
- **従業員割り当て**
  - 食品、輸送、製造
- **エネマネ**
  - エネルギーミックス、装置の運転制御
- **経路**
  - 配送、船舶、無人搬送車 (AVG)
- **メディア**
  - 最適広告配信
- **研究開発、設計**
  - 材料設計
  - 物理シミュレーション
- **ブラックボックス最適化**



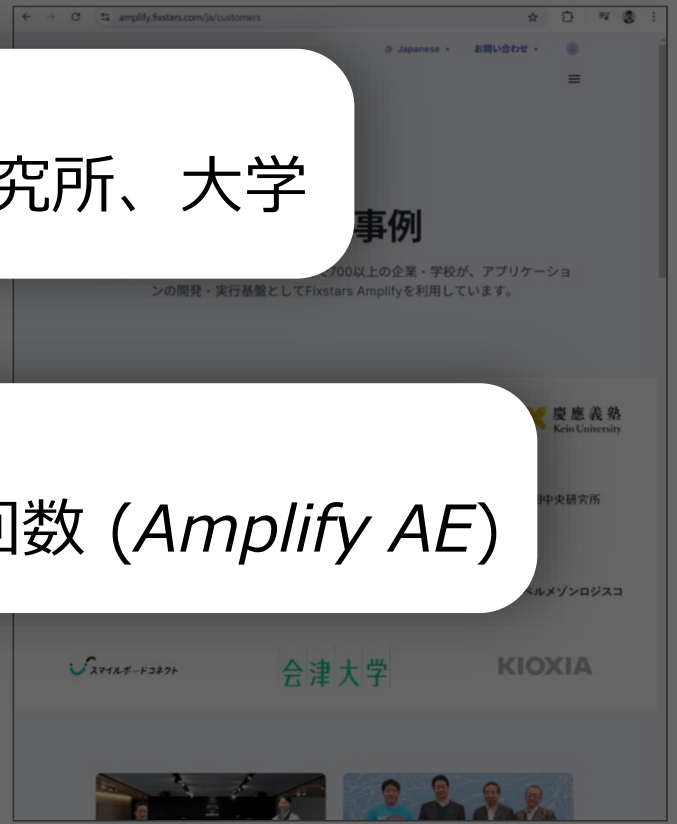
活用領域とユースケース（PoC・実稼働）

- **生産計画**
  - 多品種少量生産、保全計画、設備投資、在庫
- **従業員割り当**
  - 食品、輸送
- **エネルギーマ**
  - エネルギーミックス、機器の運転制御
- **経路**
  -
- **メ**
  -
- **研究開発、設計**
  - 材料設計
  - 物理シミュレーション
  - ブラックボックス最適化

900を超える企業、研究所、大学

8,000万を超える実行回数 (*Amplify AE*)

Amplify インタビュー　検索



アニーリングマシンのプログラミング体験

イジングマシンの実行手順

1. 数理モデル検討 — **解きたい課題**の「目的関数」「決定変数」とその「制約条件」を検討する

2. QUBO定式化 — 「 2値決定変数+二次形式」で「目的関数」と「決定変数」を記述 (変換) する
「決定変数」 に対する「制約条件」を Amplify で表現する

3. QUBO定式化 (物理) — 各マシンの仕様や制限に準拠した形式にQUBO模型を変換する
(例: 二次項に制約がある場合は「グラフマイナー埋め込み」問題を解く)

4. 入力データの準備 — 各マシンのSDKやAPI仕様に合わせてQUBO模型 (物理) をデータ化する

5. マシンの実行 — マシンを実行して出力の変数値やエネルギー値(コスト値)を解析する
上記の逆の手順を**辿り解きたい課題**の「決定変数」を解釈する

Amplify SDK によるサポート

Amplifyの基本的な使用方法 (1)

- まずはインポート

```
# Install Amplify SDK to Google Colab
! pip install -q amplify

#Import all functions and classes
from amplify import *
```

- 使用するマシンを選択

```
# Fixstars Amplify AE
client = FixstarsClient()

# Timeout 1s
client.parameters.timeout = 1000 #ms

# API token
client.token = "AE/XXXXXXXXXXXXXXXXXXXx"
```

その他のクライアントを使用する場合はドキュメントを参照

https://amplify.fixstars.com/ja/docs/amplify/v1/clients.html

Amplifyの基本的な使用方法 (2)

- 目的関数の定式化 (多項式)
  - バイナリ多項式の構築

```
# Create variable generator
g = VariableGenerator()

# Create variable array with length of 2
q = g.array("Binary", 2)
```

バイナリ変数 (**“Binary”**) だけでなく、イジング変数 (**“Ising”**) や、整数変数 (**“Integer”**)、実数変数 (**“Real”**) も指定可能

  - 式の構築

```
f = 2 * q[0] * q[1] + q[0] - q[1] + 1
print(f)
# 2 q_0 q_1 + q_0 - q_1 + 1
```

3次以上の高次多項式も可能。マシンが対応していない場合は Amplify SDK が内部的に次数下げを行う

Amplifyの基本的な使用方法 (3)

- モデルの作成とマシンの実行

```
model = Model(f)
result = solve(model, client)
```

目的関数と制約条件からモデルを作成し
使用するクライアントと共に求解

- 結果の取得

```
print(f"objective = {result.best.objective}")
# objective = 0.0

print(f"q = {q.evaluate(result.best.values)}")
# q = [0. 1.]
```

最良解を `best` で指定
目的関数の値を `objective` にて
変数の値を `values` で得る

Amplify SDK によるプログラミング例

```python
from amplify import *

# Generate variable array (numpy like)
g = VariableGenerator()
q = g.array("Binary", 2)  # [q[0], q[1]]

# Objective function
f = 1 - q[0] * q[1]

# Constraint
g = one_hot(q)  # q[0] + q[1] = 1

# Formulate the model
model = Model(f, g)

# Machine client
client = FixstarsClient()
client.token = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
client.parameters.timeout = 1000

# Run solver
result = solve(model, client)

# Analyze result
print(f"objective = {result.best.objective}")
print(f"q = {q.evaluate(result.best.values)}")
```

```
objective = 1.0
q = [0. 1.]
```

1. 定式化
   - 決定変数：スカラーあるいは配列型
   - 目的関数：決定変数による数式処理
   - 制約条件：制約条件の構築及び管理
2. ソルバークライアントの選択
   - ソルバークライアントオブジェクトの構築
   - ほぼ全てのパラメータの設定が可能
3. ソルバーを実行
   - 論理モデルをハードウェアのスペック等に合わせたモデルに変換
   - 適切なモデル変換・定式化手法を選択
4. 解の取得
   - マシンの出力解を逆変換し決定変数の形式で出力

Fixstars Amplify ご利用プラン

料金のご紹介

https://amplify.fixstars.com/ja/pricing

	ベーシック 評価・検証用の無料プラン 使い始める	スタンダード 実用レベルの計算環境 見積りを依頼	プレミアム 高性能な計算環境 見積りを依頼	Sプレミアム 最高性能の計算環境 見積りを依頼
利用料金	無料	月額10万円（1名） （税込11万円） 月額20万円（最大5名） （税込22万円）	月額20万円（1名） （税込22万円） 月額60万円（最大5名） （税込66万円）	月額30万円（1名） （税込33万円） 月額90万円（最大5名） （税込99万円）
計算環境	スモール	ミディアム	ラージ	スーパーラージ
D-Waveマシンの無料実行	3分/月	3分/月	3分/月	3分/月
SQBM+オプション	無料	月額30万円（1名） （税込33万円） 月額90万円（最大5名） （税込99万円）	月額30万円（1名） （税込33万円） 月額90万円（最大5名） （税込99万円）	月額30万円（1名） （税込33万円） 月額90万円（最大5名） （税込99万円）
サポート	ベーシック	スタンダード	プレミアム	プレミアム
評価・検証フェーズでの利用	✓	✓	✓	✓
実運用フェーズでの利用		✓	✓	✓

定式化や実装を
手厚く
支援します！

開発支援サービス(個別見積り)

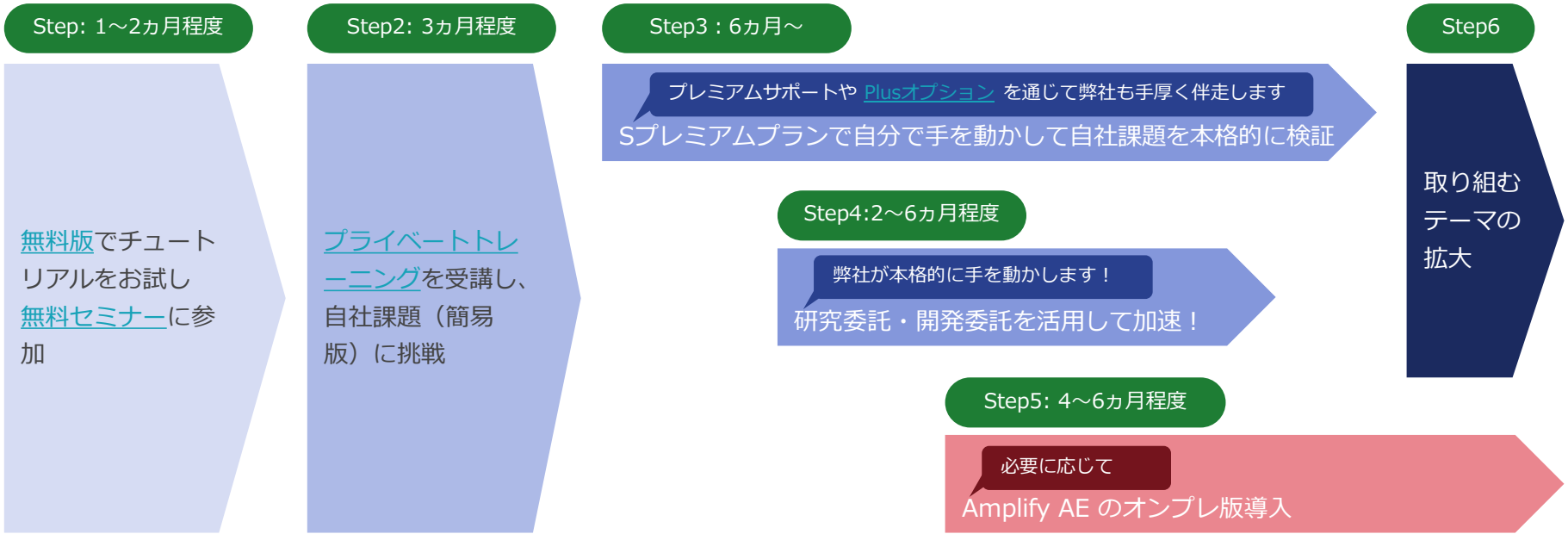
コンサル・システム開発等
数百万円～数千万円

＋

月額利用料
百万円～

研究・開発者向けおすすめの進め方

二次・非線形を上手に使いこなせるように、**弊社と一緒に**取り組みを進めていきましょう！



セミナー・トレーニングのご紹介

https://amplify.fixstars.com/ja/news/seminar

お客様の実際の課題解決をご支援するために、**無料セミナー**や**有償トレーニング**を提供しています。

無料セミナー・ワークショップ

ビジネス向け、エンジニア向けに分けて
開催しています！

ビジネス向け

製造業向け量子コンピュータ時代のDXセミナー

見える化、予測・分析、その先の最適化へ

組合せ最適化問題や量子アニーリング・イジングマシンの概要をご紹介したのち、製造業における組合せ最適化を活用したDX推進の一例として、生産計画最適化や生産ラインのシフト最適化などの事例とデモをご紹介いたします。「Fixstars Amplify」を通じて量子アニーリング・イジングマシンを活用することで、どのようなビジネス上の効果が期待できるのかを感じていただきたいと思います。

エンジニア向け

製造業向け量子コンピュータ時代のDXセミナー

最適化の中身を覗いてみよう

製造業における組合せ最適化を活用したDX推進の一例として、生産計画最適化、勤務シフト最適化などの事例を用いて、問題設定の考え方、目的関数や制約条件の定式化、実装のポイントなどを実際のコードを見ながら解説します。また、サンプルコードを用いて、ご自身の環境で実際に量子アニーリング・イジングマシンを動かす体験をしていただきます。

企業向けプライベートトレーニング

お客様が抱える実際の課題やデータを使った
カスタムメイド のトレーニングです！

全4回のレクチャーとお客様に実施いただく「課題」を含む約1.5か月のコースです。コースの前半では、量子アニーリング・イジングマシン専用の開発／実行環境であるFixstars Amplifyを用いてPython言語による組合せ最適化アプリケーション開発方法を学びます。後半では、お客様が抱える実際の課題やデータを使ったトレーニングを実施します。量子アニーリング・イジングマシンを使って実課題の解決に取り組んでみたい方に最適なコースです。

ワークショップ
事前準備（事前メールの内容）

ワークショップの事前準備 (1)

- 【事前メールに記載】ご自身のPC (ブラウザ上) で Python プログラミングを行います。Google Colaboratory を使うので、事前にログイン出来ることを確認をお願いします（要 Google アカウント）

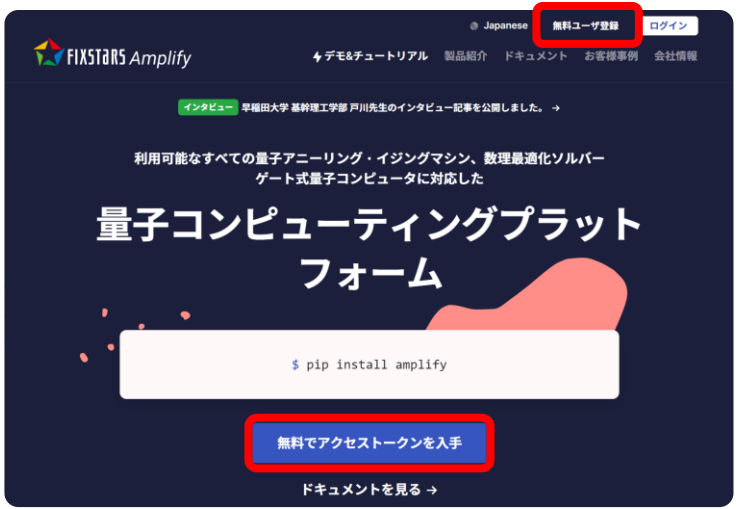
Google Colab 検索

https://colab.research.google.com/

- 【事前メールに記載】 Fixstars Amplify ホームページよりユーザ登録の上、無料トークンの取得をお願いします（1分で終わります）

Fixstars Amplify 検索

https://amplify.fixstars.com/



質問は随時ZoomのQ&Aへお願いします

ワークショップの事前準備 (2)

【事前メールに記載】

- 取得されたトークンを用いて、トークンチェック用サンプルコードが動くか確認をお願いします。

  https://colab.research.google.com/drive/1bg2Ql3McJck_Sto8uvxtmPUMWtRFhf7a （※URLはZoomのチャット欄を参照）

- サンプルコードは閲覧のみ可能な状態です。「ファイル」→「ドライブにコピーを保存」の上、ご自身のトークンを入力してください。その後、Shift + Enterで実行下さい。

```
! pip install amplify

token = "AE/**************************" # ご自身のトークンを入力
```

- ご自身のトークン番号は、Amplifyウェブページ → よりご確認いただけます。

- 実行後、以下の結果が出力されればOKです。

```
result: [q_0, q_1] = [1. 1.] (f = 0.0)
```

FIXSTARS Amplify

ワークショップ

通常の組合せ最適化

（ブラックボックス最適化への導入）

サンプルコード

サンプルコードを開き、「ファイル」→「ドライブにコピー」の上、トークンを入力し実行して下さい。

- 数の分割サンプルコード
  https://colab.research.google.com/drive/1bl6BGPwnpd3N0AveoDMGb8egprEdQMqU
  （※URLはZoomのチャット欄を参照）

質問は随時 Zoom の Q&A へお願いします

数の分割問題（概要）

- 与えられた n 個の整数$a_0, \cdots, a_{n-1}$を二つの集合に分ける。
  集合内の数の和が、もう一方の集合内の数の和と等しくなるようできるか?
  - NP完全問題: とても難しい問題として知られている → 全通り試すしか方法は無い
  - 問題のバリエーション
    - 判定問題: 完全に等しく出来るか？または等しい組合せは何か？
    - 最適化問題: 完全に等しいか、または最も惜しい組合せは何か？

数の分割問題（具体例と解法の方針）

具体例

`{2,10,3,8,5,7,9,5,3,2}` の10個の数の完璧な分割は見つけられるか？

答え

- 存在する
  - `{2,3,5,7,10}` と `{2,3,5,8,9}`
  - どちらも和は 27
- 分割方法は 23 通り存在する (対称を除く)

どうやって解くか？

- ひとつの『数』がどちらの集合に分割されるか全通り試す → $2^{10} = 1024$通り
- 効率のよい厳密な方法は知られていない・・・ (もし発見されたら大騒ぎ)

数の分割問題（定式化）

最適化問題：数の分割において最も惜しい組合せは何か？

- **目的関数**

  {集合1の和} – {集合2の和} の絶対値を最小化

- **決定変数**

  数 a_i がどちらの集合に属するかを s_i で表す

  - a_i = { 2,10, 3, 8, 5, 7, 9, 5, 3, 2}
  - s_i = {-1, 1,-1, 1,-1,-1, 1, 1, 1, 1}

数理モデル

- 目的関数

$$f = \left| \sum_{i=0}^{N-1} s_i a_i \right| \quad (s_i \in \{-1, +1\})$$

$\sum s_i a_i$ は、自然と
{〖1〗の集合の和} – {〖-1〗の集合の和}
となる！

数の分割問題（バイナリへの式変形）

- 0-1整数二次計画問題への変換
  - Quadratic Unconstrained Binary Optimization (QUBO) 式

$$f = \left| \sum_{i=0}^{N-1} s_i a_i \right| \quad (s_i \in \{-1, +1\})$$

$$\rightarrow \left(\sum_{i=0}^{N-1} s_i a_i \right)^2 \quad (s_i \in \{-1, +1\})$$

$$\rightarrow \left(\sum_{i=0}^{N-1} (2q_i - 1) a_i \right)^2 \quad (q_i \in \{0, +1\})$$

絶対値を二次式で表す

±1をバイナリで表す
（イジングで定式化するなら不要）

数の分割問題（定式化の具体例）

問題

- a_i ={2,10,3,8,5,7,9,5,3,2} の10個の数の完璧な分割は見つけられるか？

決定変数

- $q_i = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_9\} (q_i \in \{0,1\})$ で集合0又は集合1、どちらに所属するかを表す

目的関数

$$f = \left(\sum_{i=1}^{N} (2q_i - 1)a_i \right)^2$$

目的関数を展開

$$f = \begin{pmatrix} 2(2q_0 - 1) + 10(2q_1 - 1) + 3(2q_2 - 1) + 8(2q_3 - 1) + 5(2q_4 - 1) \\ +7(2q_5 - 1) + 9(2q_6 - 1) + 5(2q_7 - 1) + 3(2q_8 - 1) + 2(2q_9 - 1) \end{pmatrix}^2$$

プログラムコード

https://colab.research.google.com/drive/1bl6BGPwnpd3N0AveoDMGb8egprEdQMqU

- 定式化

```python
# Numbers
a = [2, 10, 3, 8, 5, 7, 9, 5, 3, 2]

# Variables
g = VariableGenerator()
q = g.array("Binary", len(a))

# Objective
f = 0
for i in range(len(a)):
    f += (2 * q[i] - 1) * a[i]
f = f**2
```

NumPyのような配列演算で簡潔に書ける！

```python
f = ((2 * q - 1) * a).sum() ** 2
```

- 結果

$\{2, 10, 3, 5, 7\}$ と $\{8, 9, 5, 3, 2\}$

目的関数の値0

集合の合計値27

```
q = [1, 1, 1, 0, 1, 1, 0, 0, 0, 0], f = 0.0, w = 27
```

オンラインデモ & チュートリアル

Amplify デモ 検索

https://amplify.fixstars.com/ja/demo

デモアプリケーション

ピクロスパズルの求解

プログラミング難易度 ★★★★☆

複雑な定式化の例として、数字で与えられるヒントを元にマスを塗り、絵を完成させるパズルゲーム、ピクロスを解くアプリを開発します。

デモアプリ　サンプルコード

チュートリアル応用編

ブラックボックス最適化 (1)

プログラミング難易度 ★★★★☆

複雑で未知な目的関数にも適用可能な、機械学習と組み合わせ最適化を組み合わせたブラックボックス最適化手法を紹介し、Amplifyを用いて実装します。

サンプルコード

チュートリアル応用編

ブラックボックス最適化 (2)

プログラミング難易度 ★★★★☆

機械学習と量子アニーリング・イジングマシンを活用するブラックボックス最適化の適用例として、疑似的な高温超電導を実現する材料探索を取り扱います。

サンプルコード

チュートリアル応用編

ブラックボックス最適化 (3)

プログラミング難易度 ★★★★☆

化学プラントにおける生産量を最大化するための運転条件最適化を行います。最適化には、機械学習モデルに基づくブラックボックス最適化と化学反応に関する物理シミュレーションを用います。

サンプルコード

チュートリアル応用編

ブラックボックス最適化 (4)

プログラミング難易度 ★★★★☆

流体機器設計に不可欠な翼型の最適化問題を取り上げます。最適化には、組み合わせ最適化及び機械学習に基づくブラックボックス最適化と流体シミュレーションを用い、翼の揚抗比を最大化するように翼型の探索を行います。

サンプルコード

デモアプリケーション

容量制約つき運搬経路問題 (CVRP)

プログラミング難易度 ★★★★☆

運送業における効率的な配送計画の策定やごみ収集や道路清掃における訪問順序の最適化等での応用が期待される容量制約つき運搬経路問題（CVRP）を取り扱います。

デモアプリ　サンプルコード

チュートリアル応用編

ブラックボックス最適化 (5)

プログラミング難易度 ★★★★☆

ブラックボックス最適化により、商業施設による交通集中が発生し得る都市における、交通渋滞を低減するような信号機群の最適制御を実施します。最適化の実施及び実証には、マルチ・エージェント・シミュレーションによる交通シミュレーションを用います。

サンプルコード

チュートリアル応用編

定式化による交通信号機の最適化

プログラミング難易度 ★★★★☆

都市における渋滞を最小化するために、刻一刻と変化する交通状況に応じ、組合せ最適化を用いてリアルタイムに信号機の最適制御を実施します。また、その様な信号機制御を実施した際の都市の交通量をシミュレーションします。

サンプルコード

チュートリアル応用編

10. 整数長ジョブスケジューリング問題

プログラミング難易度 ★★★☆☆

あらかじめ決まった数のジョブとマシンがあり、それぞれのジョブにかかる時間が分かっているとします。それぞれのジョブをいずれかのマシンに割り当てます。ジョブスケジューリング問題では、最も早く全ジョブが完了するような割り当て方を求めます。

サンプルコード

チュートリアル基礎編

画像のノイズ除去

プログラミング難易度 ★★★☆☆

画像のノイズ除去を行うアプリケーションを開発します。

サンプルコード

チュートリアル応用編

会議室割当問題

プログラミング難易度 ★★★☆☆

制約条件を用いて定式化するアプリケーションの例として会議室割当問題のアプリケーションを開発します。

サンプルコード

チュートリアル応用編

タクシーマッチング問題

プログラミング難易度 ★★★☆☆

目的関数と制約条件を用いて定式化するアプリケーションの例としてタクシーマッチング問題のアプリケーションを開発します。

サンプルコード

デモアプリケーション

グラフ彩色問題

プログラミング難易度 ★★★☆☆

Fixstars Amplifyによる、グラフ彩色問題の定式化を体験します。

デモアプリ　サンプルコード

デモアプリケーション

巡回セールスマン問題

プログラミング難易度 ★★★★☆

Fixstars Amplifyによる、巡回セールスマン問題の定式化を体験します。

デモアプリ　サンプルコード

デモアプリケーション

数独

プログラミング難易度 ★★★☆☆

Fixstars Amplifyによる、数独の定式化を体験します。

デモアプリ　サンプルコード

デモアプリケーション

ライドシェア

プログラミング難易度 ★★★★★

集合型ライドシェアの最適化アプリケーションを体験します。

デモアプリ　サンプルコード

デモアプリケーション

タスク割当問題

プログラミング難易度 ★★★★★

店舗とタスクに従業員を割り当てる組合せ最適化問題のアプリケーションを体験します。

デモアプリ　サンプルコード

デモアプリケーション

ポートフォリオ最適化

プログラミング難易度 ★★★★☆

リスクとリターンを考慮した株式ポートフォリオの最適化アプリケーションを体験します。

デモアプリ　サンプルコード

組合せ最適化と
ブラックボックス最適化

通常の組合せ最適化とブラックボックス最適化

通常の数理最適化

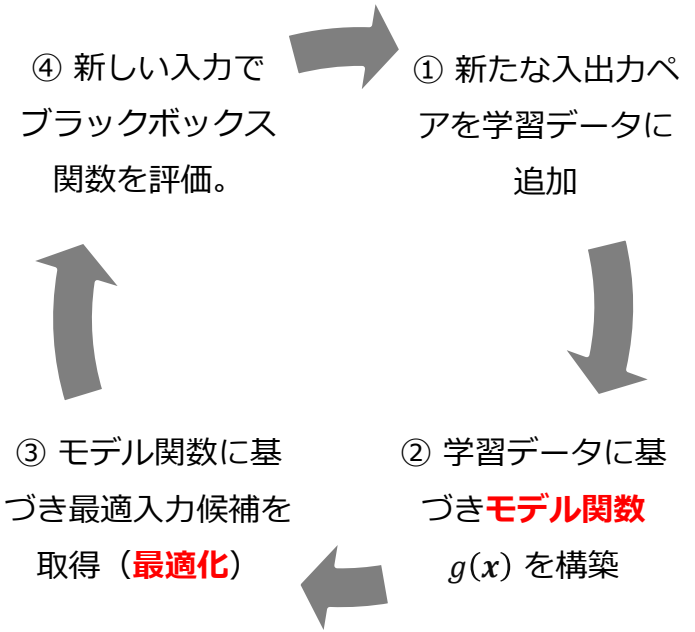
- **目的関数を定式化（例：QUBO）**
  - 数の分割（差の最小化）
    $f = [\Sigma(2q_i - 1)a_i]^2$
  - 経路最適化（経路距離最小化）
    $f = \Sigma\Sigma\Sigma d_{i,j} q_{n,i} q_{n+1,j} \cdots$
- **最適化の実施**
  - イジングマシンにより、定式化された目的関数を最小化

ブラックボックス最適化 (BBO)

- **直接の定式化が困難な目的関数**
  - 低損失な流体デバイス形状？
  - 高性能な材料/構造トポロジー？
  - ターゲットに近い結果を実現する実験条件？
- **最適化の実施**
  - 実験やシミュレーションによる**試行錯誤**により定式化できない目的関数を最小化

ブラックボックス最適化アプローチ

- 次の最適化サイクルを ブラックボックス関数 $f(\boldsymbol{x})$ を対象に実施
  （実験計測や数値シミュレーション）



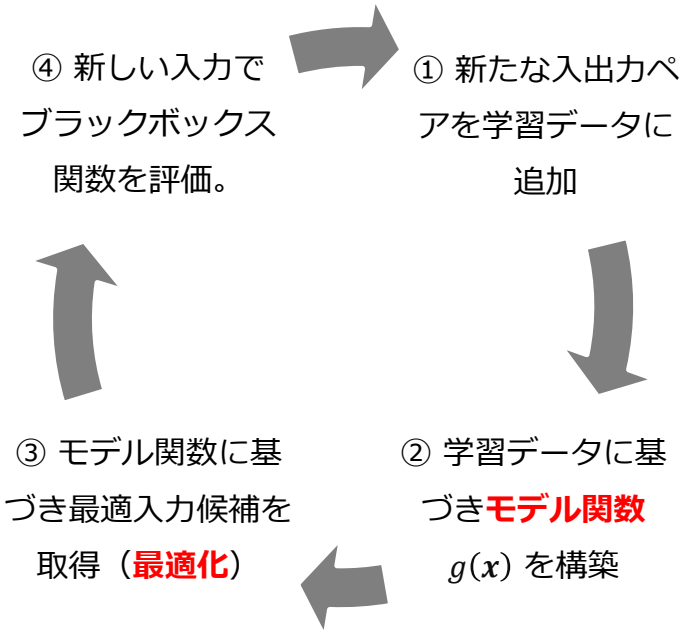
課題：

- **モデル関数** $g(\boldsymbol{x})$ の構築手法
- モデル関数の**最適化**手法
  ※ $g(\boldsymbol{x})$ 最小を実現する$\boldsymbol{x}$ の取得

より少ないサイクル（評価回数）で
より良い解が得られるよう工夫

ブラックボックス最適化アプローチ

- 次の最適化サイクルを ブラックボックス関数 $f(\boldsymbol{x})$ を対象に実施
  （実験計測や数値シミュレーション）



FMQA :
K. Kitai, et al., Phys. Rev. Res. (2020).
T. Inoue, et al., Opt. Express (2022).

- **モデル関数** $g(\boldsymbol{x})$
  Factorization Machine
  (FM、機械学習モデルの一種)
- **最適化**
  学習モデルをQUBOとして
  Amplify で求解

➔ 次元の呪いに強い
➔ 制約条件に強い

モデル関数としての Factorization Machine (FM)

- モデル関数 $g(\boldsymbol{x})$ に機械学習モデルの一種である Factorization Machine (FM) を用いると、次のように変数 $\boldsymbol{x}$ に対する2次式での記述ができる。

$$g(\boldsymbol{x}|\boldsymbol{w},\boldsymbol{v}) = w_0 + \langle \boldsymbol{w},\boldsymbol{x}\rangle + \sum_{i=1}^{n}\sum_{j=i+1}^{n}\langle \boldsymbol{v}_i,\boldsymbol{v}_j\rangle\, x_i\, x_j$$

$$= w_0 + \sum_{i=1}^{n} w_i x_i + \frac{1}{2}\sum_{f=1}^{k}\left(\left(\sum_{i=1}^{n} v_{if}\, x_i\right)^2 - \sum_{i=1}^{n} v_{if}^2\, x_i^2\right)$$

➔ **QUBO式**

- k はハイパーパラメータ、 $\boldsymbol{w}$ 及び $\boldsymbol{v}$ は FM 学習後に取得される FM パラメータ。
- FM パラメータ数は k に依存。$k = n$ のときは QUBO の相互作用項と同じ自由度がある一方、 k を小さくすることでパラメータ数を減らし過学習を抑制する効果
- このようなブラックボックス最適化手法を **FMQA** と呼ぶ。

K. Kitai, et al., Phys. Rev. Res. (2020).
T. Inoue, et al., Opt. Express (2022).

ブラックボックス最適化 活用例

材料分野に限らず、幅広い分野へ適用可能

QA-BBO: 活用例 (Amplify サンプルプログラム)

Amplify デモ　検索

ハンズオン

チュートリアル応用編

ブラックボックス最適化 (2)

プログラミング難易度 ★★★★☆

機械学習と量子アニーリング・イジングマシンを活用するブラックボックス最適化の適用例として、疑似的な高温超電導を実現する材料探索を取り扱います。

サンプルコード

材料最適化

FMQA
×
物理モデル

チュートリアル応用編

ブラックボックス最適化 (3)

プログラミング難易度 ★★★★☆

化学プラントにおける生産量を最大化するための運転条件最適化を行います。最適化には、機械学習モデルに基づくブラックボックス最適化と化学反応に関する物理シミュレーションを用います。

サンプルコード

化学プラント運転条件最適化

FMQA
×
化学シミュレーション

チュートリアル応用編

ブラックボックス最適化 (4)

プログラミング難易度 ★★★★☆

流体機器設計に不可欠な翼型の最適化問題を取り上げます。最適化には、組み合わせ最適化及び機械学習に基づくブラックボックス最適化と流体シミュレーションを用い、翼の揚抗比を最大化するように翼型の探索を行います。

サンプルコード

翼形状最適化

FMQA
×
流体シミュレーション

チュートリアル応用編

ブラックボックス最適化 (5)

プログラミング難易度 ★★★★☆

ブラックボックス最適化により、商業施設による交通集中が発生し得る都市における、交通渋滞を低減するような信号機群の最適制御を実施します。最適化の実施及び実証には、マルチ・エージェント・シミュレーションによる交通シミュレーションを用います。

サンプルコード

信号制御最適化

FMQA
×
マルチ・エージェント・シミュレーション

チュートリアル応用編

ブラックボックス最適化 (6)

プログラミング難易度 ★★★★☆

ブラックボックス最適化により、撹拌性能に影響を与える設計パラメータに対して、混合効率が最大化されるような撹拌機の最適設計を実施します。最適化の実施および評価には、濃度分布に基づく簡易的な撹拌シミュレーションを用います。

サンプルコード

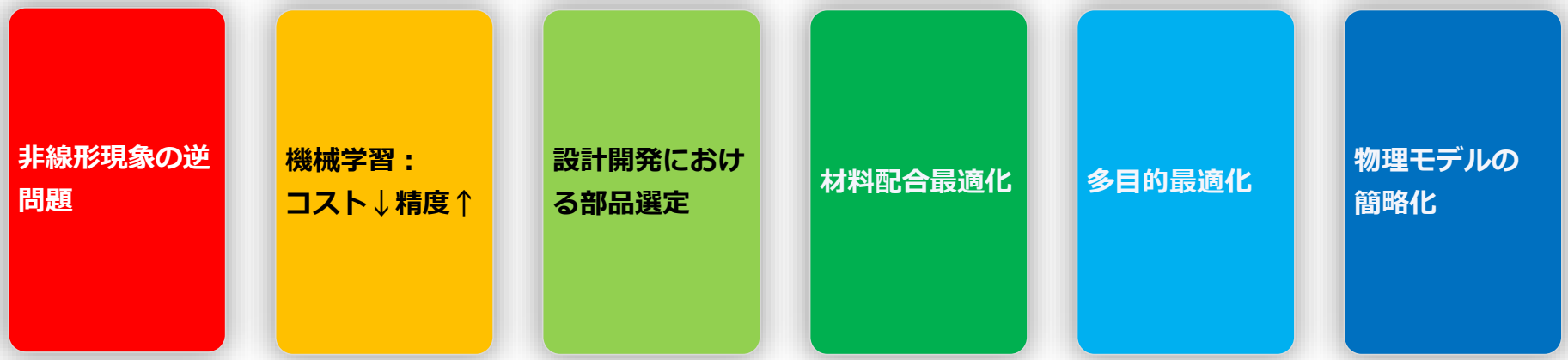
機器設計最適化

FMQA
×
撹拌シミュレーション

FIXSTARS Amplify

QA-BBO: 活用例 (Amplify ユーザー)

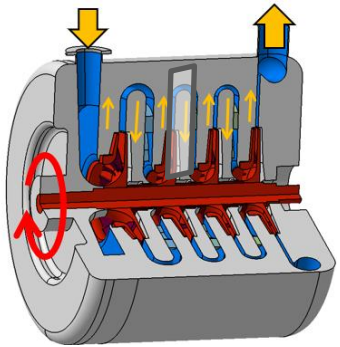
- **活用領域**
  - 化学、 創薬、 食品、 自動車、 電機、 通信、 重工、 エネルギー、ヘルスケア・・・



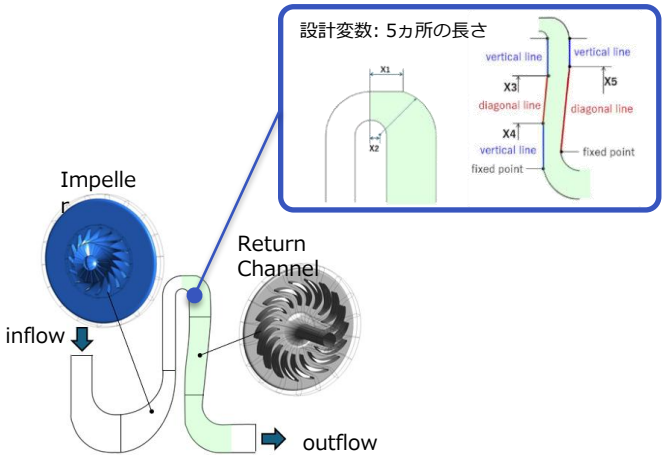
事例: ターボ機械の形状最適化（川崎重工業様）

- ターボ機械の開発では、従来より商用最適化ソフトによる遺伝的アルゴリズム (GA) を使用し形状最適化を行うことが多かったが、最適化規模が大きくなると最適解の求解までに時間がかかり、開発期間が長期化するといった課題があった
- 量子アニーリング・イジングマシンを活用した BBO により、従来手法と比べ、同じ計算回数でもより優れた解が得られることを確認。今後はさらに設計変数を増やしていく予定

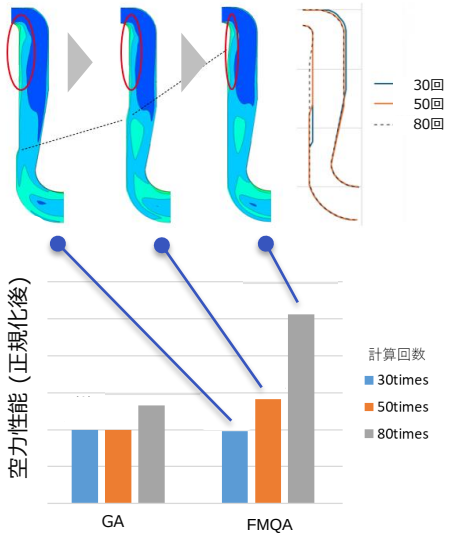
遠心圧縮機



流路形状を最適化したい
(圧縮機全体の空力性能（ポリトロープ効率）の最大化)



最適化が進むごとに損失発生領域
が低減する形状へ



事例: 車両設計最適化（マツダ様）

amplify マツダ Search

- 複数車種の車体構造同時設計最適化問題。衝突性能を含めた品質特性の条件を満たした上で、部品の軽量化と共通部品数の最大化の実現する多目的最適化問題
- 2017年にマツダ・JAXA がベンチマーク問題として公開し[*1]、国内外の研究グループ[*2,3] により様々な手法が試されていた。1～3万回程度の試行により、ある程度よい解が得られることは確認されていた

- FMQA により、1,000回程度の試行で、従来手法と同等以上の解を見つけることに成功！
- 今後は、QUBO 式近似の計算コストを削減しつつ、最適化性能も向上させるような手法の検討を予定

*1 応答曲面法を用いた複数車種の同時最適化ベンチマーク問題の提案
*2 進化計算コンペティション2017開催報告
*3 Multi-objective Bayesian optimization over high-dimensional search spaces

FIXSTARS Amplify

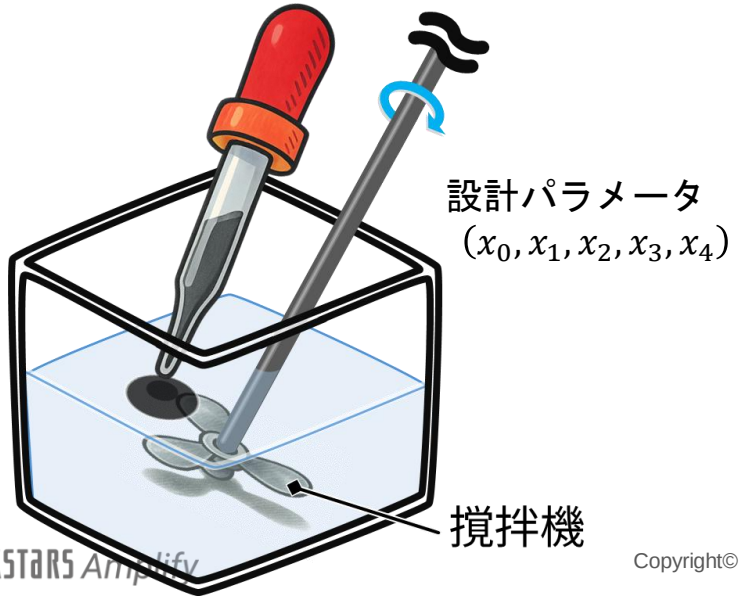
FIXSTARS Amplify

ブラックボックス最適化ハンズオン

問題設定及び目的関数

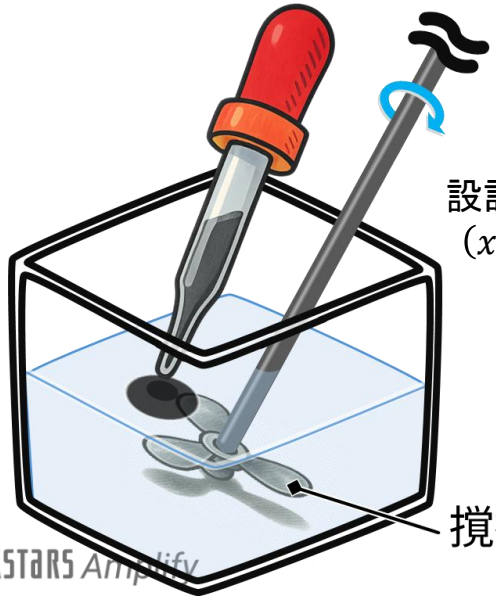
撹拌機器の設計･運転条件最適化

- 撹拌機器の設計・運転最適化
  - 一定時間でできるだけ均一に撹拌したい



撹拌機器の設計･運転条件最適化

- 撹拌機器の設計・運転最適化
  - 一定時間でできるだけ均一に撹拌したい
    - ➔ 目的関数：撹拌後の物質濃度の空間変動値



- ブラックボックスな目的関数

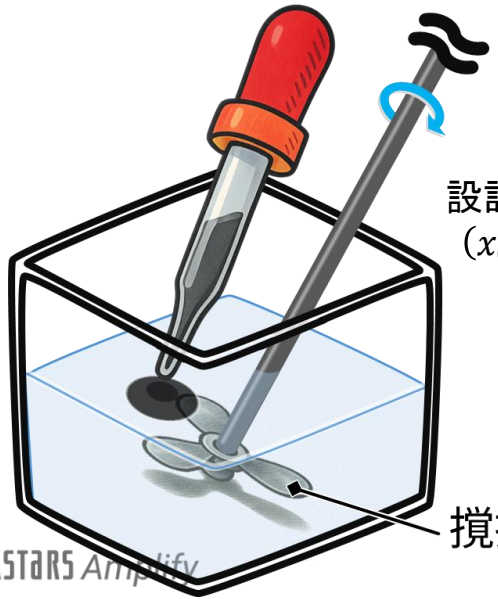
【入力】
設計パラメータ $(x_0, x_1, x_2, x_3, x_4)$

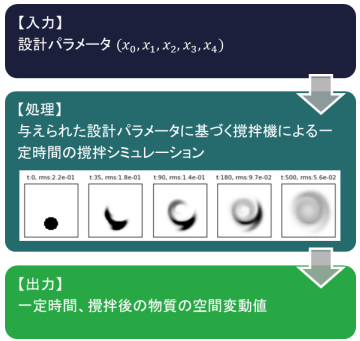
【処理】
与えられた設計パラメータに基づく撹拌機による一定時間の撹拌シミュレーション

【出力】
一定時間、攪拌後の物質の空間変動値

撹拌機器の設計･運転条件最適化

- 撹拌機器の設計・運転最適化
  - 一定時間でできるだけ均一に撹拌したい
    - ➔ 目的関数：撹拌後の物質濃度の空間変動値





① 新たな入出力ペアを学習データに追加

② 学習データに基づき**モデル関数** $g(\boldsymbol{x})$ を構築

③ モデル関数に基づき最適入力候補を取得（**最適化**）

ブラックボックス関数の実装

- 撹拌シミュレータ MixingSimulator

```
# 与えられたパラメータ値でシミュレーターを初期化
simulator = MixingSimulator(x0, x1, x2, x3, x4)

# 時刻 500 までの撹拌シミュレーションを実施し、濃度の標準偏差 c_std を取得
c_std = simulator.simulate(duration=500)

# 結果表示
print(f"{c_std=:.3f}")  # 撹拌後の濃度の標準偏差
simulator.plot_evolution(num_snaps=5)  # 撹拌過程の濃度分布の時系列変化をプロ
```

- ブラックボックス関数

```
def blackbox(x0: int, x1: int, x2: int, x3: int, x4: int) -> float:
    s = MixingSimulator(x0, x1, x2, x3, x4)
    c_std = s.simulate(duration=500)
    s.plot_evolution()
    print(f"{c_std=:.3f}")
    return c_std
```

【入力】
設計パラメータ $(x_0, x_1, x_2, x_3, x_4)$

【処理】
与えられた設計パラメータに基づく撹拌機による一定時間の撹拌シミュレーション

【出力】
一定時間、撹拌後の物質の空間変動値

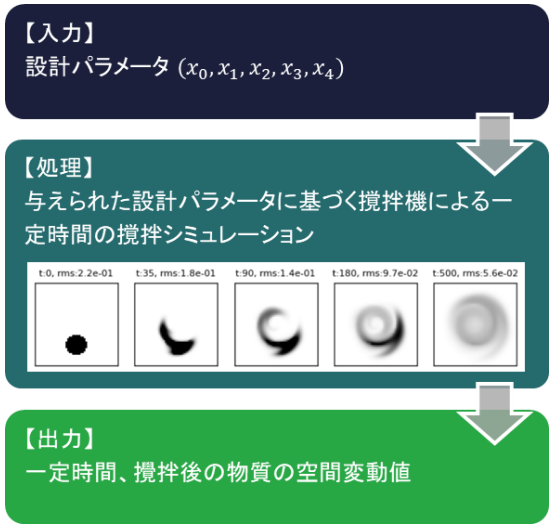
ブラックボックス最適化ハンズオン

1. FM機械学習
2. FMQA

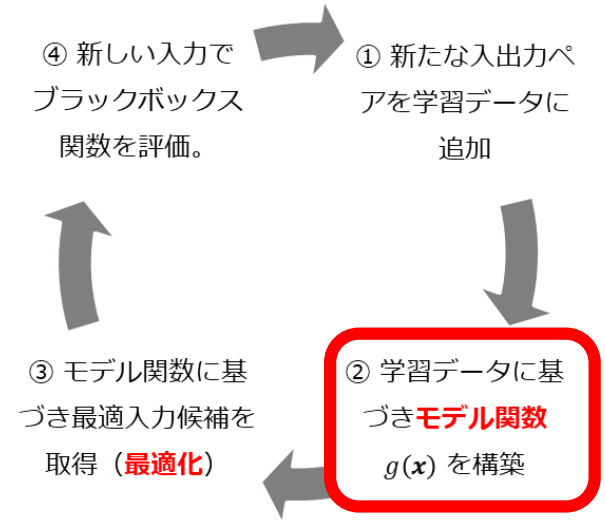
ブラックボックス最適化のデモプログラム

撹拌機の設計最適化、サンプルプログラム（FM機械学習）
https://colab.research.google.com/drive/195_mSDUKv3poUuXqzOTzbBFQ7Bh5p2SW

ブラックボックス関数



最適化サイクル



FM プログラム 1/6 （ブラックボックス関数の定義）

- 整数決定変数（設計パラメータ）の値域の指定

```python
# 撹各設計パラメータの上限・下限値
bounds: dict[str, tuple[int, int]] = {
    "x0": (2, 10),  # 2 <= x0 <= 10
    "x1": (5, 20),  # 5 <= x1 <= 20
    "x2": (0, 45),  # 0 <= x2 <= 45
    "x3": (1, 5),  # 1 <= x3 <= 5
    "x4": (1, 4),  # 1 <= x4 <= 4
}
```

- ブラックボックス関数の実装

```python
def blackbox(x0: int, x1: int, x2: int, x3: int, x4: int) -> float:
    s = MixingSimulator(x0, x1, x2, x3, x4)
    c_std = s.simulate(duration=500)
    s.plot_evolution()
    print(f"{c_std=:.3f}")
    return c_std
```

FM プログラム 2/6（整数エンコーダーの補助クラス）

- 決定変数は整数、FM (QUBO)はバイナリ変数を考慮
  - 整数⇔バイナリの変換を行うクラス

```python
class IntegerVariable:
    """ブラックボックス最適化のための整数決定変数クラス。"""

    def __init__(
        self, bounds: tuple[int, int]
    ):
        self._bounds = bounds

    def encode(self, x: int) -> np.ndarray:
        """決定変数値をエンコードし、バイナリ化する関数"""
        if x < self._bounds[0] or x > self._bounds[1]:
            raise ValueError(f"x must be in {self._bounds}")
        ret = np.zeros(self._bounds[1] - self._bounds[0])
        ret[0 : x - self._bounds[0]] = 1
        return ret

    def decode(self, x: np.ndarray) -> int:
        """バイナリ値を整数決定変数値にデコードする関数"""
        if x.shape != self._q.shape:
            raise ValueError(f"x must be of shape {self._q.shape}")
        return x.sum() + self._bounds[0]
```

```python
class Variables:
    """整数決定変数のリストを管理するクラス。"""

    def __init__(self, variable_list: list[IntegerVariable]):
        self._variable_list = variable_list

    def encode(self, x: list[int]) -> np.ndarray:
        """決定変数値をエンコードし、バイナリ化する関数"""
        ret: list[int] = []
        for i, var in enumerate(self._variable_list):
            ret += var.encode(x[i]).tolist()  # type: ignore
        return np.array(ret)

    def decode(self, x: np.ndarray) -> np.ndarray:
        """バイナリ値を整数決定変数値にデコードする関数"""
        ret: list[int] = []
        ista = 0
        for _, var in enumerate(self._variable_list):
            iend = ista + len(var.binary_variables)
            ret.append(var.decode(x[ista:iend]))
            ista = iend
        return np.array(ret, dtype=int)


var_list = [IntegerVariable(bounds=b) for b in bounds.values()]
variables = Variables(var_list)
```

FM プログラム 3/6（FMモデルの定義）

```python
class TorchFM(nn.Module):
    def __init__(self, d: int, k: int):
        """モデルを構築する

        Args:
            d (int): 入力ベクトルのサイズ
            k (int): パラメータ k
        """
        super().__init__()
        self.d = d
        self.v = nn.Parameter(torch.randn((d, k)))
        self.w = nn.Parameter(torch.randn((d,)))
        self.w0 = nn.Parameter(torch.randn(()))

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        """入力 x を受け取って y の推定値を出力する"""
        out_linear = torch.matmul(x, self.w) + self.w0
        out_1 = torch.matmul(x, self.v).pow(2).sum(1)
        out_2 = torch.matmul(x.pow(2), self.v.pow(2)).sum(1)
        out_quadratic = 0.5 * (out_1 - out_2)

        out = out_linear + out_quadratic
        return out
```

FM モデルを PyTorch で定義する

$$f(\boldsymbol{x}|\boldsymbol{w},\boldsymbol{v}) = \underbrace{w_0 + \sum_{i=1}^{d} w_i x_i}_{\text{out\_linear}} + \underbrace{\frac{1}{2}\left[\underbrace{\sum_{f=1}^{k}\left(\sum_{i=1}^{d} v_{if}x_i\right)^2}_{\text{out\_1}} - \underbrace{\sum_{f=1}^{k}\sum_{i=1}^{d} v_{if}^2 x_i^2}_{\text{out\_2}}\right]}_{\text{out\_quadratic}}$$

FM プログラム 4/6（FMの学習）

```python
def train(
    x: np.ndarray,
    y: np.ndarray,
    model: TorchFM,
    plot_learning_curve=False,
) -> None:

    # イテレーション数
    epochs = 2000
    # モデルの最適化関数
    optimizer = torch.optim.AdamW([model.v, model.w, model.w0], lr=0.1)
    # 損失関数
    loss_func = nn.MSELoss()

    # データセットの用意
    x_tensor, y_tensor = (torch.from_numpy(x).float(), torch.from_numpy(y).float())

    dataset = TensorDataset(x_tensor, y_tensor)

    train_set, valid_set = random_split(dataset, [0.8, 0.2])
    if len(valid_set) == 0:
        valid_set = train_set
    train_loader = DataLoader(train_set, batch_size=8, shuffle=True)
    valid_loader = DataLoader(valid_set, batch_size=8, shuffle=True)

    # `range` の代わりに `tqdm` モジュールを用いて進捗を表示
    for i in trange(epochs, leave=False):
        # 学習過程
        for x_train, y_train in train_loader:
            optimizer.zero_grad()
            pred_y = model(x_train)
            loss = loss_func(pred_y, y_train)
            loss.backward()
            optimizer.step()
```

FM 学習は、通常の機械学習と同様に進める。教師データを学習・検証データに分割し、ミニバッチ学習。

- **x, y**: 教師データ
- **model**: FM モデル（TorchFM）
- **epochs**: エポック（繰り返し）の数
- **lr**:（初期）学習率

学習済みモデルに対し、次のような評価を実施

```python
# 教師データの入力値に基づきモデルを評価
print(f"corrcoef: {compute_corrcoef(model(x_tensor), y_tensor):.3f}")
print(f"RMS error: {min_loss:.3f}")
```

FM プログラム 5/6（初期学習データ作成）

```python
def generate_random_input() -> np.ndarray:
    x: list[int] = []
    for v_min, v_max in bounds.values():
        x.append(rng.integers(v_min, v_max + 1))
    return np.array(x)


def init_training_data(num_samples: int):
    # n0 個の 長さ d の入力値を乱数を用いて作成
    data: list[np.ndarray] = []
    for i in range(num_samples):
        data.append(generate_random_input())
    x = np.array(data)

    # 入力値の重複が発生していたらランダムに値を変更して回避する
    x = np.unique(x, axis=0)
    while x.shape[0] != num_samples:
        x = np.vstack((x, generate_random_input()))
        x = np.unique(x, axis=0)

    # blackbox 関数を評価して入力値に対応する n0 個の出力を得る
    y = np.zeros(num_samples)
    for i in range(num_samples):
        y[i] = blackbox(*x[i])
    return x, y
```

学習データを乱数により生成

- `num_samples`：学習データのサンプル数
- `blackbox`：ブラックボックス関数（実験又はシミュレーション） $f(\boldsymbol{x})$

FM プログラム 6/6（メイン部分）

https://colab.research.google.com/drive/195_mSDUKv3poUuXqzOTzbBFQ7Bh5p2SW

```
# 初期教師データの作成
n_0 = 30
x, y = init_training_data(num_samples=n_0)
x_encoded = np.array([variables.encode(x[i]) for i in range(x.shape[0])])

# 機械学習モデルの作成
model = TorchFM(len(x_encoded[0]), k=10)

# モデル学習の実行
train(x_encoded, y, model, plot_learning_curve=True)
```

実際にサンプルプログラムを実行してみましょう。

デフォルトの条件から、

- FMのハイパーパラメータ (k)
- エポック数 (epochs)
- 学習率 (lr)

などを変更した場合、真値と予測値の相関係数及びRMS誤差はどのように変化するでしょうか？

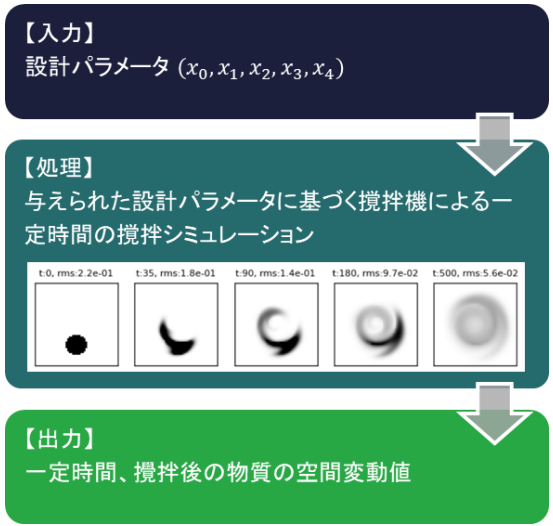
ブラックボックス最適化ハンズオン

1. FM機械学習
2. FMQA

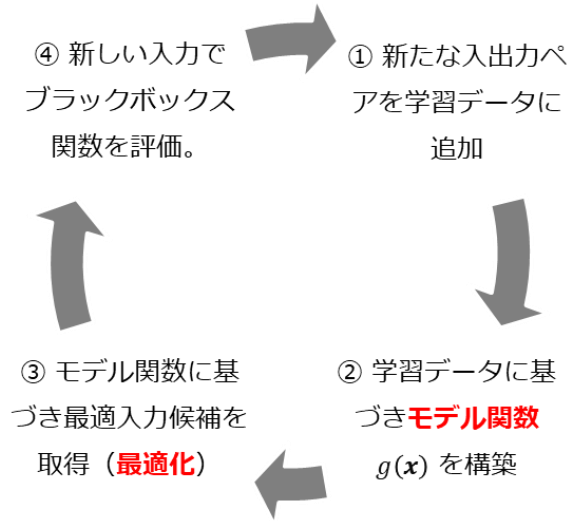
ブラックボックス最適化のデモプログラム

撹拌機の設計最適化、サンプルプログラム（FMQA）
https://colab.research.google.com/drive/1E_Re1W2hhGtRmMWDhi1_VjTg-0o2T9Mk

ブラックボックス関数



最適化サイクル



FMQA プログラム 1/2 （アニーリング部分）

```python
def anneal(torch_model: TorchFM) -> np.ndarray:
    """FM モデルのパラメータを受け取り、それらのパラメータにより記述される FM

    # TorchFM からパラメータ v, w, w0 を取得
    v, w, w0 = torch_model.get_parameters()

    # Amplify のバイナリ決定変数を取得
    x = variables.binary_variables

    # FM モデルと等価な QUBO モデル（目的関数）を作成
    out_linear = w0 + (x * w).sum()
    out_1 = ((x[:, np.newaxis] * v).sum(axis=0) ** 2).sum()  # type: ignore
    out_2 = ((x[:, np.newaxis] * v) ** 2).sum()
    objective: Poly = out_linear + (out_1 - out_2) / 2

    # Amplify モデルを定義
    amplify_model = Model(objective, constraints)

    # 最小化を実行（構築したモデルと、始めに作ったソルバークライアントを引数
    result = solve(amplify_model, client)
    if len(result.solutions) == 0:
        raise RuntimeError("No solution was found.")

    # モデルを最小化する入力ベクトル（最適設計パラメータ候補）を返却
    return x.evaluate(result.best.values).astype(int)
```

学習済みFMに基づき $\hat{x}$ を推定する関数

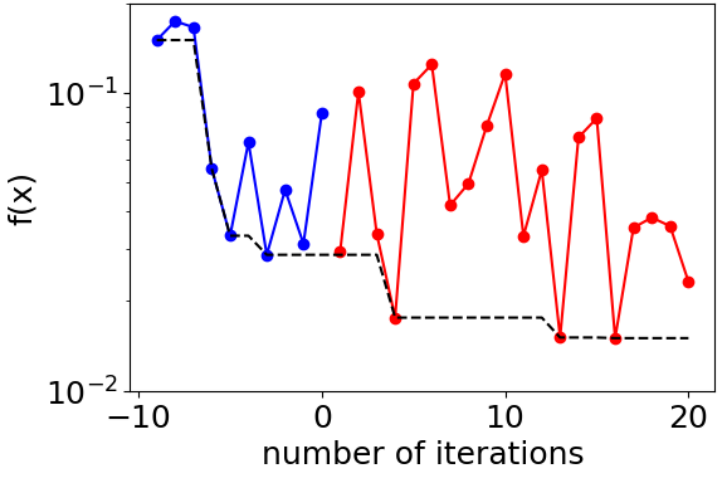
- バイナリ決定変数配列の取得
- 学習済みモデルからモデル係数を取得
- モデル係数に基づき目的関数 $g(x)$ を構築
- solve の実行（トークン有力を忘れずに）
- 本サイクルにおける $\hat{x}$ を返却

FMQA プログラム 2/2（メイン部分）

https://colab.research.google.com/drive/1E_Re1W2hhGtRmMWDhi1_VjTg-0o2T9Mk

```
# 初期学習データ（x）をバイナリ値にエンコーディング
x_encoded = np.array([variables.encode(x[i]) for i in range(x.shape[0])])

# N 回のイテレーションを実行
# `range` の代わりに `tqdm` モジュールを用いて進捗を表示
for i in trange(n):
    # 機械学習モデルの作成
    model = TorchFM(len(x_encoded[0]), k=10)

    # モデル学習の実行
    train(x_encoded, y, model)

    # 学習済みモデルの最小値を与える入力ベクトルの値（バイナリ値にエンコード済み）を取得
    x_hat = anneal(model)

    # x_hat が学習データ内サンプルと同一の場合はランダムに再生成
    while (x_hat == x_encoded).all(axis=1).any():
        x_hat_random = generate_random_input()
        x_hat = variables.encode(x_hat_random.tolist())  # type: ignore
        print("deduplication")

    # バイナリ決定変数値を整数決定変数値にデコード
    x_hat_decoded = variables.decode(x_hat)

    # 推定された入力ベクトルを用いてブラックボックス関数を評価
    y_hat = blackbox(*x_hat_decoded)

    # 評価した値をデータセットに追加
    x_encoded = np.vstack((x_encoded, x_hat))
    y = np.append(y, y_hat)
```

実際にサンプルプログラムを実行してみましょう！



FMQA プログラムの実務での活用方法

```python
def blackbox(x0: int, x1: int, x2: int, x3: int, x4: int) -> float:
    s = MixingSimulator(x0, x1, x2, x3, x4)
    c_std = s.simulate(duration=500)
    s.plot_evolution()
    print(f"{c_std=:.3f}")
    return c_std
```

```python
def blackbox(x: np.ndarray) -> float:
  # xの組合せを元に実験を実施
  objective = input(f"{x=} で実験を実施し、結果を入力してください")

  # または、シミュレーションを実施
  objective = run_simulation(x)  # type: ignore

  # 最小化したい目的関数値を取得・返却
  return objective
```

基本的に、blackbox()を変更する。必要に応じて、現在の学習データの出力などを追加。

- 例①：blackbox() 内でシミュレーションを呼び出し、後処理、その戻り値を最小化するように最適化。
- 例②：blackbox() 内で実験を行う。つまり、1回のFMQA で推定された探索候補 $\hat{x}$ を対象に実験し、結果を教師データに追加、次のFMQA 試行を行う。
- $h(x)$ を最大にするような入力 x を推定する場合は、$-h(x)$ などを目的関数 $f(x)$ とする。

今後について

ぜひ、デモ・チュートリアルにあるサンプルコードにも挑戦してみてください！

一般的な組合せ最適化問題

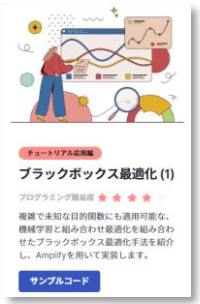
目的関数のみで定式化

制約条件のみで定式化

目的関数 + 制約条件

ブラックボックス最適化問題

概要



材料探索

翼型最適化

信号機制御

困った時はドキュメンテーションを！

https://amplify.fixstars.com/docs/amplify/v1/index.html

今後のセミナーの予定

今後も定期的に無料セミナーを開催します！

2025/8/21
ブラックボックス最適化
（撹拌機器の最適設計）

- はじめに
- 会社紹介
- Fixstars Amplifyの紹介
- ブラックボックス最適化のワークショップ
- Wrap Up
  - 事例のご紹介
  - 今後の進め方
  - Q&A

2025/9/25（2枠実施！）
最新事例と技術解説
(Annealing Engine)

- はじめに
- Fixstars Amplifyのご紹介
- 最新事例紹介
- 技術解説
  - 制約条件の重み
  - 求解時間の設定
  - 多目的最適化
  - その他、躓きポイント

2025/10/23
シフト最適化
(Annealing Engine)

- はじめに
- 会社紹介
- Fixstars Amplifyのご紹介
- シフト最適化のワークショップ
- Wrap Up
  - 事例のご紹介
  - 今後の進め方
  - Q&A

ご質問・ご不明点がありましたら、お問い合わせフォームでご連絡下さい
https://amplify.fixstars.com/ja/contact

FIXSTARS Amplify

Q&A