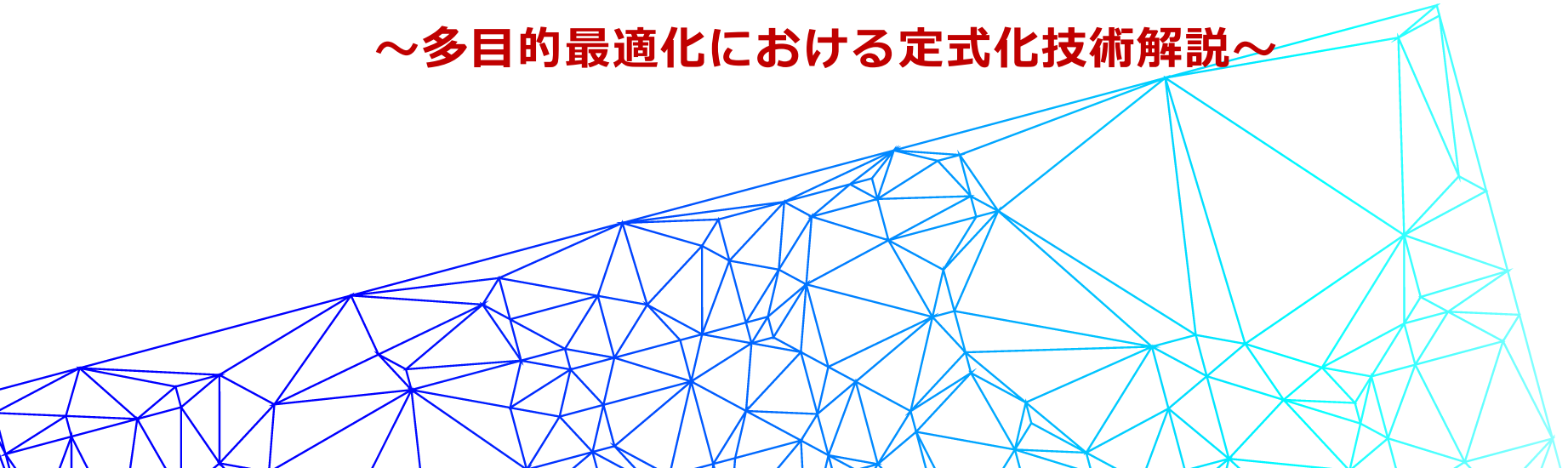
FIXSTARS Amplify

量子コンピュータ時代のプログラミングセミナー

～多目的最適化における定式化技術解説～



本日の予定

- 会社紹介
- Fixstars Amplify の紹介
- 組合せ最適化事例
- 多目的最適化実装のコツ
- 求解性能向上の工夫
- 自走開発の進め方

質問は随時 Zoom の Q&A へお願いします

(株) Fixstars Amplify の紹介

- **組合せ最適化**のための量子コンピューティングクラウド
  プラットフォーム「Fixstars Amplify」の提供

1,100 を超える企業、研究所、大学

1.2億 を超える実行回数 (*Amplify AE*)

- 2021年に設立（株式会社フィックスターズからスピンアウト）
  - 代表取締役社長CEO：松田 佳希（博士）
- 親会社 (株) フィックスターズ（東証P: 3687）
  - ソフトウェア高速化プロフェッショナル集団
  - 2017年 日本で初めてD-Wave Systems社と提携

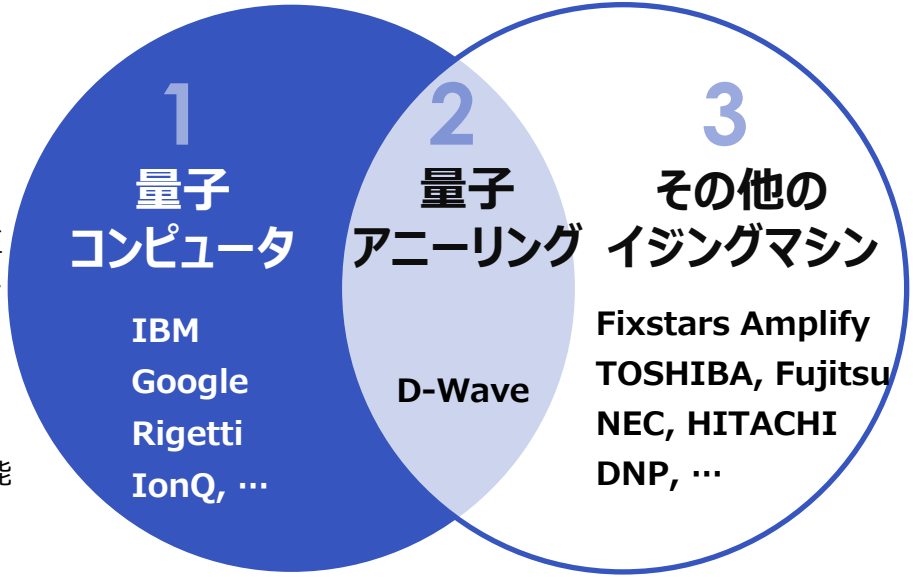
FIXSTARS Amplify

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
（量子インスパイアード技術）

- 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*）

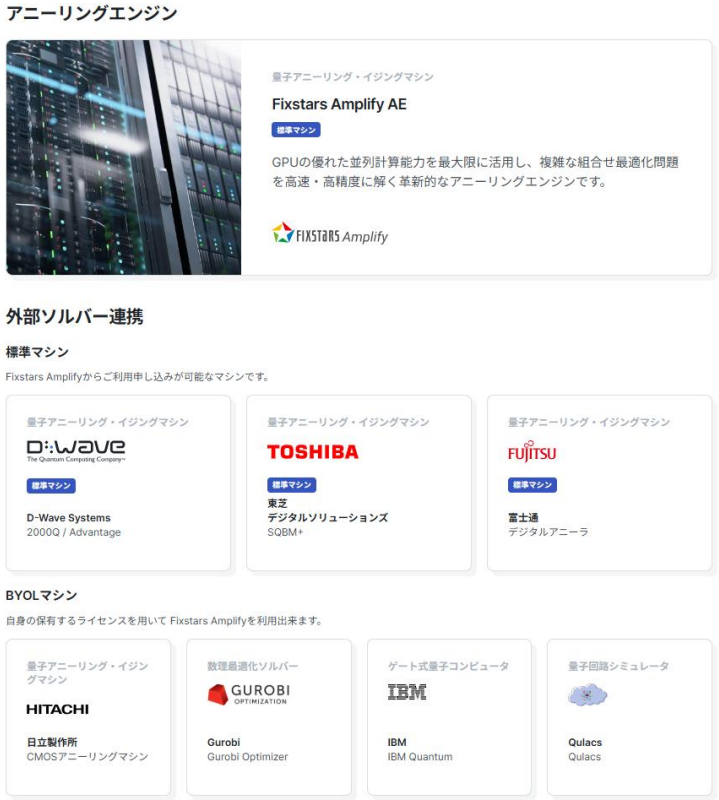
2 量子アニーリング（量子焼きなまし法式のイジングマシン）

- 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**

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 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**

（多目的）最適化実装のコツと求解性能向上のヒント

Amplify SDK による基本実装

- **制約条件**を満たしながら**目的関数**が最小となる**決定変数**の値を探索
  - **決定変数**　バイナリ変数 `q[0]`, `q[1]`
  - **目的関数**　`q[0] - 2 * q[0] * q[1]`
  - **制約条件**　one_hot ➔ q の要素1つが1
    - その他の制約ヘルパー関数
      - `equal_to`
      - `less_equal`
      - `greater_equal`
      - `clamp`
      - `domain_wall`

```python
# 決定変数の発行
q = VariableGenerator().array("Binary", 2)

# 目的関数と制約条件の定義
objective = q[0] - 2 * q[0] * q[1]
constraint = one_hot(q)

# モデルの構築
model = Model(objective, constraint)

# ソルバーの指定
client = AmplifyAEClient()
client.token = "Amplify AE のアクセストークン"
client.parameters.time_limit_ms = 1000  # ms

# 最適化の実行
result = solve(model, client)

# 結果の表示
print(q.evaluate(result.best.values))  # [1. 0.]
print(objective.evaluate(result.best.values))  # 0.0
```

制約条件のペナルティ化

- 制約条件のペナルティ化
  - 制約違反の場合には目的関数にペナルティが加算されるように Amplify SDK が論理変換
  - ペナルティ込み目的関数を最小化するよう探索
  - 結果として、全ペナルティゼロ（制約充足）な解が得られることが期待される
- 制約の重み
  - 一般的なイジングマシンでは、目的関数に対して、どの程度のペナルティを加算するかを調整する。不適切な重みを与えると、**実行可能解を得られない可能性大**。
  - **Amplify AEの場合は、ソルバー側で重みを自動調整するので不要**

```
# 決定変数の発行
q = VariableGenerator().array("Binary", 2)

# 目的関数と制約条件の定義
objective = q[0] - 2 * q[0] * q[1]
constraint = one_hot(q)

# モデルの構築
model = Model(objective, constraint)

# ソルバーの指定
client = AmplifyAEClient()
client.token = "Amplify AE のアクセストークン"
client.parameters.time_limit_ms = 1000  # ms

# 最適化の実行
result = solve(model, client)

# 結果の表示
print(q.evaluate(result.best.values))  # [1. 0.]
print(objective.evaluate(result.best.values))  # 0.0
```

Amplify AE 求解フロー

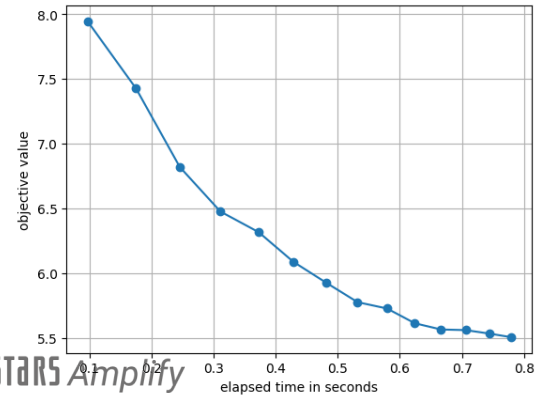
- 探索の実行
  - **求解時間**の中で、アルゴリズム実行単位の「探索」を繰り返し、より小さい目的関数値を実現する解を取得
- 解の返却
  - 探索回数分の解を SDK に返却
  - 最初の「探索」で真の最適解が得られた場合、SDK に返却される解は1つ
  - 最低1つの「解」が得られるまで求解を継続
- 制約充足の判定
  - ソルバーは制約ペナルティ値のみを考慮。
  - 最終的な制約充足は SDK によって判定。

```python
# 決定変数の発行
q = VariableGenerator().array("Binary", 2)

# 目的関数と制約条件の定義
objective = q[0] - 2 * q[0] * q[1]
constraint = one_hot(q)

# モデルの構築
model = Model(objective, constraint)

# ソルバーの指定
client = AmplifyAEClient()
client.token = "Amplify AE のアクセストークン"
client.parameters.time_limit_ms = 1000  # ms

# 最適化の実行
result = solve(model, client)

# 結果の表示
print(q.evaluate(result.best.values))  # [1. 0.]
print(objective.evaluate(result.best.values))  # 0.0
```

適切な求解時間の決め方は？

- 解のタイムスタンプ情報の活用
  - AE はタイムアウト内に得られた「探索」回数分の「解」全てを SDK に返却。
  - SDK は返却された「解」から、制約を満たす解（実行可能解）のみを `result` に格納
  - 実行可能解及びタイムスタンプをプロット

関連ドキュメントページ　：
https://amplify.fixstars.com/ja/docs/amplify/v1/timing.html#id4




```python
import matplotlib.pyplot as plt

# 定式化
・・・

# ソルバーの指定
client = AmplifyAEClient()
client.token = "Amplify AE のアクセストークン"
client.parameters.time_limit_ms = 1000  # ms

# ソルバーの実行
result = solve(model, client)

# それぞれの解の時刻と目的関数の値を取得
times = [solution.time.total_seconds() for solution in result]
objective_values = [solution.objective for solution in result]

# プロット
plt.plot(times, objective_values, "-o")
plt.xlabel("elapsed time in seconds")
plt.ylabel("objective value")
plt.grid(True)
```

単目的最適化：「重み」に関するヒント（Amplify AEでは考慮不要）

- 制約重み weight の例
  - 目的関数値が取りそうな値とする
    - max(d)
    - sum(d)
    - …
- ただし、
  - **Amplify AE の場合**、制約重みはソルバー内部で自動調整するため、**ユーザーによる重み指定は不要。**
- 目的関数が複数の場合は？

```
# 巡回セールスマン問題の定式化例

gen = VariableGenerator()
q = gen.array("Binary", N + 1, N)
q[-1, :] = q[0, :]

objective = 0
for k in range(N):
    for i in range(N):
        for j in range(N):
            objective += d[i, j] * q[k, i] * q[k + 1, j]

constraint1 = one_hot(q[:-1], axis=1)
constraint2 = less_equal(q[:-1], axis=0)

model = objective + weight * (constraint1 + constraint2)
```

$$\sum_{i,j,k}^{N} d_{i,j}\, q_{k,i} q_{k+1,j}$$

多目的最適化：スケーリングに関する基本的な方針

- 多目的最適化
  - 目的関数が複数存在
  - 「ソフト制約」も目的関数の1つ
  - 複数の目的関数が異なるレンジを取り得る
  - それぞれの目的関数をスケーリング（右例）。スケーリング係数（s_1, s_2）は、（obj_1, obj_2）の定式化に基づいて見積もり
- 多目的最適化における制約重み
  - Amplify AE では不要
  - スケーリング後の目的関数に対して必要であれば重みを設定

```
# 多目的最適化問題の例
obj_1 = ...
obj_2 = ...

const_1 = one_hot(...)
const_2 = less_equal(...)


# obj_1 と obj_2 のレンジが大きく異なる場合、不適
model = obj_1 + obj_2 + const_1 + const_2

# obj_1 と obj_2 のレンジが大きく異なる場合でも、対応可
# ここで、s_1 と s_2 はそれぞれ obj_1 と obj_2 の代表値
model = obj_1 / s_1 + obj_2 / s_2 + const_1 + const_2


# スケーリング後に各目的関数の重みを微調整しても良い
model = 10 * obj_1 / s_1 + obj_2 / s_2 + const_1 +...
```

多目的最適化：発展的なスケーリング係数決定

- スケーリング係数の自動決定 (AE)

1. 制約問題として求解

   ➔ 短いタイムアウトで多くの解を取得

   ➔ 制約を満たすランダム解とみなせる

2. 解を個々の目的関数に代入、代入結果を目的関数のスケーリング係数とする

3. 目的関数を除算しスケーリングを実施

   ➔ 目的関数の取りうる値が 1 程度になることが期待される

4. スケーリング後の目的関数と制約条件を考慮し、最適化問題として求解

```
# 多目的最適化問題の例
obj_1 = ...
obj_2 = ...
const_1 = one_hot(...)
const_2 = less_equal(...)

# 制約問題として求解
model = const_1 + const_2

result = solve(model, client)

# s_1 と s_2 はそれぞれ obj_1 と obj_2 の代表値
s_1 = max([obj_1.evaluate(sol.values) for sol in result])
s_2 = max([obj_2.evaluate(sol.values) for sol in result])

# obj_1 と obj_2 のレンジが大きく異なる場合でも、対応可
model = obj_1 / s_1 + obj_2 / s_2 + const_1 + const_2

# 実際の最適化問題として求解
result = solve(model, client)
```

多目的最適化：発展的なスケーリング係数決定

- スケーリング係数の自動決定 (AE)

1. 制約問題として求解（目的関数無し）
   ➔ 短いタイムアウトで多くの解を取得
   ➔ 制約を満たすランダム解とみなせる
2. 解を個々の目的関数に代入、代入結果を目的関数のスケーリング係数とする
3. 目的関数をスケーリング係数で除算しスケーリングを実施
   ➔ 目的関数の取りうる値が 1 程度になることが期待される
4. スケーリング後の目的関数と制約条件を考慮し、最適化問題として求解

amplify.fixstars.com/ja/demo/hems

求解性能を向上させる工夫

- ソルバーの直列実行 (num_solves)
  - https://amplify.fixstars.com/ja/docs/amplify/v1/serial.html
  - ソルバーによっては、長時間のタイムアウトを指定して 1 回実行するよりも短時間のタイムアウトで何回か繰り返し実行するほうがより良い解を見つける可能性

```
result = solve(model, client, num_solves=3)
```

求解性能を向上させる様々な工夫

- 上位プランのハードウェア利用
  - Basic/Standard: **V100**　　Premium: **A100**　　S Premium: **H100**
- GPU並列実行 (num_gpus)
  (Amplify AEでマルチGPUオプション設定の場合のみ)
  - アニーリングにて使用するGPU数を増やすことで、より高速・高精度な求解が期待

```
client = AmplifyAEClient()
client.token = "Amplify AE のアクセストークン"
client.parameters.timeout = timedelta(seconds=1)
client.parameters.num_gpus = 4
```

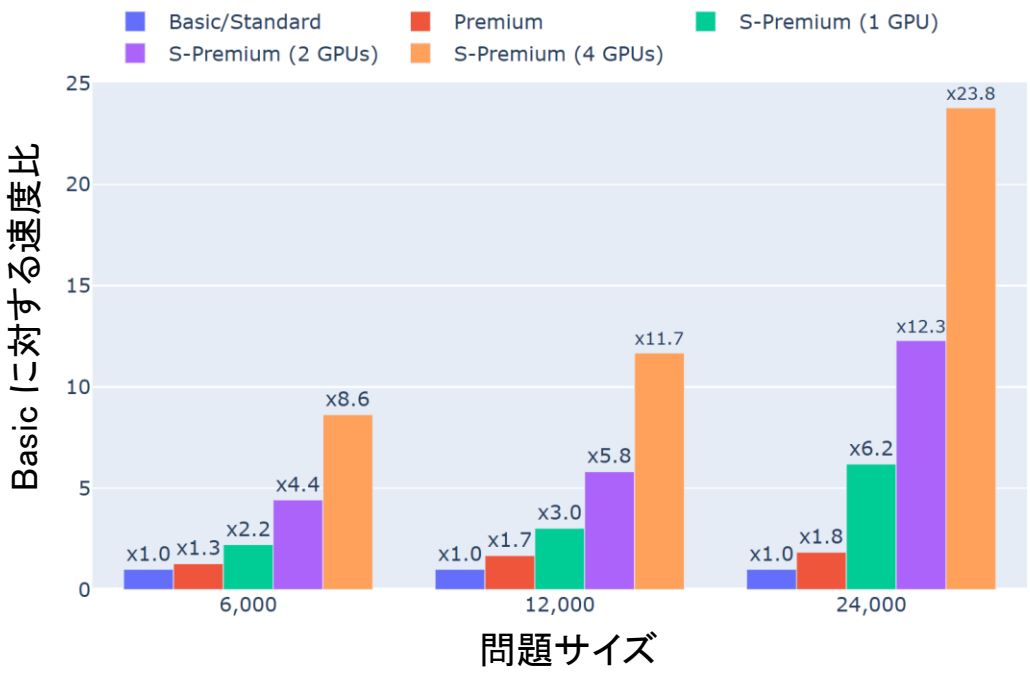
ハード別・並列数別の性能

amplify.fixstars.com/ja/engine#plan

異なる3つの問題サイズで、各プランにおけるGPUとアニーリング速度*の関係

- Basic/Standard: **V100**
- Premium: **A100**
- S Premium: **H100**

* Amplify AE内部で1回のモンテカルロサンプリングに要する時間の逆数を測定。



Fixstars Amplify
自走開発・進め方

プランのご紹介

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		✓	✓	✓

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

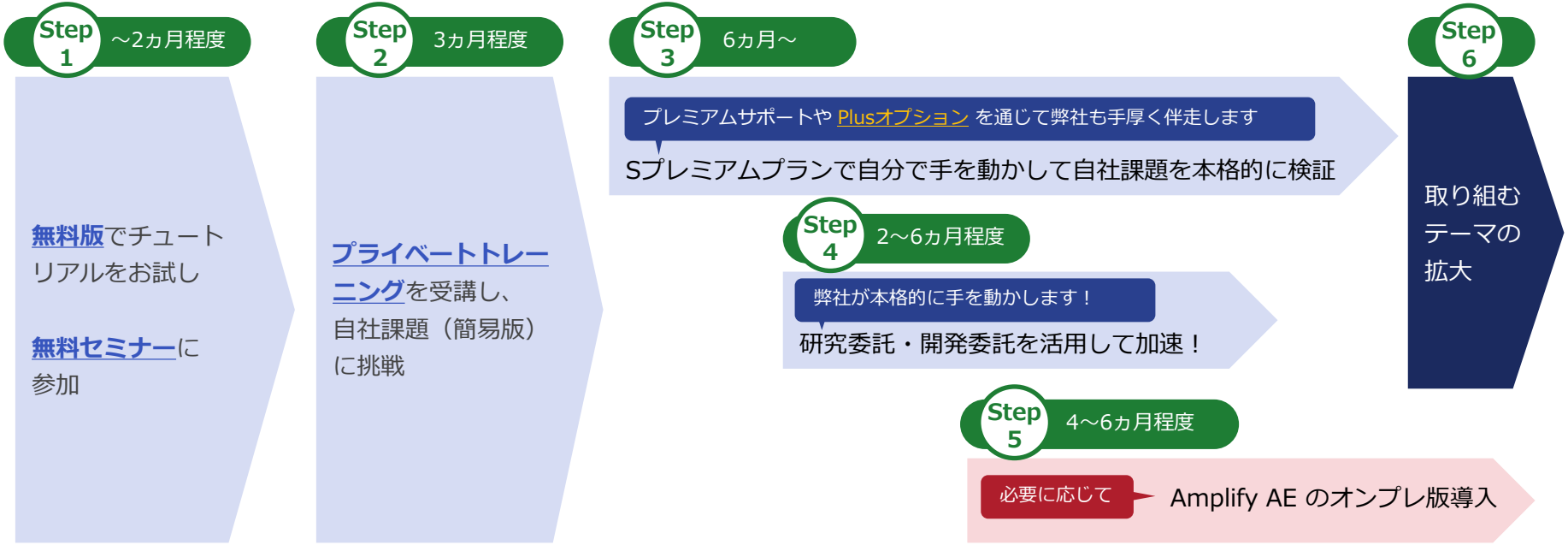
＋

月額利用料
百万円～

定式化や実装を**手厚く**支援します！

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。

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

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

制約条件のみで定式化

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

目的関数 + 制約条件

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

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

ブラックボックス最適化問題

概要

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

材料探索

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

翼型最適化

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

信号機制御

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

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

今後のセミナー予定・情報発信

2026/2/19（受付中）
「エネルギーマネジメント最適化 ハンズオン」

エネルギーマネジメント最適化をハンズオンで実施。

2026/3/5 （予定）
「Amplify-BBOpt 技術解説」

イジングマシン活用のブラックボックス最適化を簡単実装可能なライブラリAmplify-BBOptの使い方を紹介。

2026/3/18（予定）
「ブラックボックス最適化（機械学習の特徴量抽出）」

ブラックボックス最適化による機械学習の特徴量抽出をハンズオンで実施。

2026/4/9 （予定）
「Amplify AE技術解説」

メジャーアップデートされたAmplify独自開発イジングマシンであるAnnealing Engineについて解説。

@FixstarsAmplify

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