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材料探索 ～

14:00 開始予定

マイク、カメラをOFFにしてしばらくお待ちください

事前準備がお済みでない方は、この時間にお済ませください
(事前準備の詳細はセミナーのご案内メールをご参照ください)

Zoomの表示名は、セミナー申し込み時の
お名前としていただけますようご協力お願いいたします

FIXSTARS Amplify

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材料探索～

本日の予定

第一部 14:00-14:30

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

質問は随時ZoomのQ&Aへお願いします

第二部 14:35-16:00

- Fixstars Amplify を用いたワークショップ
  - 組合せ最適化の基本【ハンズオン】
  - ブラックボックス最適化
    - FMQAの概要とフロー
    - 機械学習モデル FM【ハンズオン】
    - 材料探索 FMQA【ハンズオン】
  - まとめ

本セミナーのゴール

組合せ最適化に関する次の項目を実感していただく

- 身の回りに組合せ最適化問題は多い。一方で、複雑で非線形な物理・社会現象の場合、組合せ最適化に必要な定式化が困難なものもある。
- 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により、複雑現象に対する最適化を実施可能。
- ブラックボックス最適化を Fixstars Amplify により実施できる。

質問は随時ZoomのQ&Aへお願いします

FIXSTARS Amplify

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

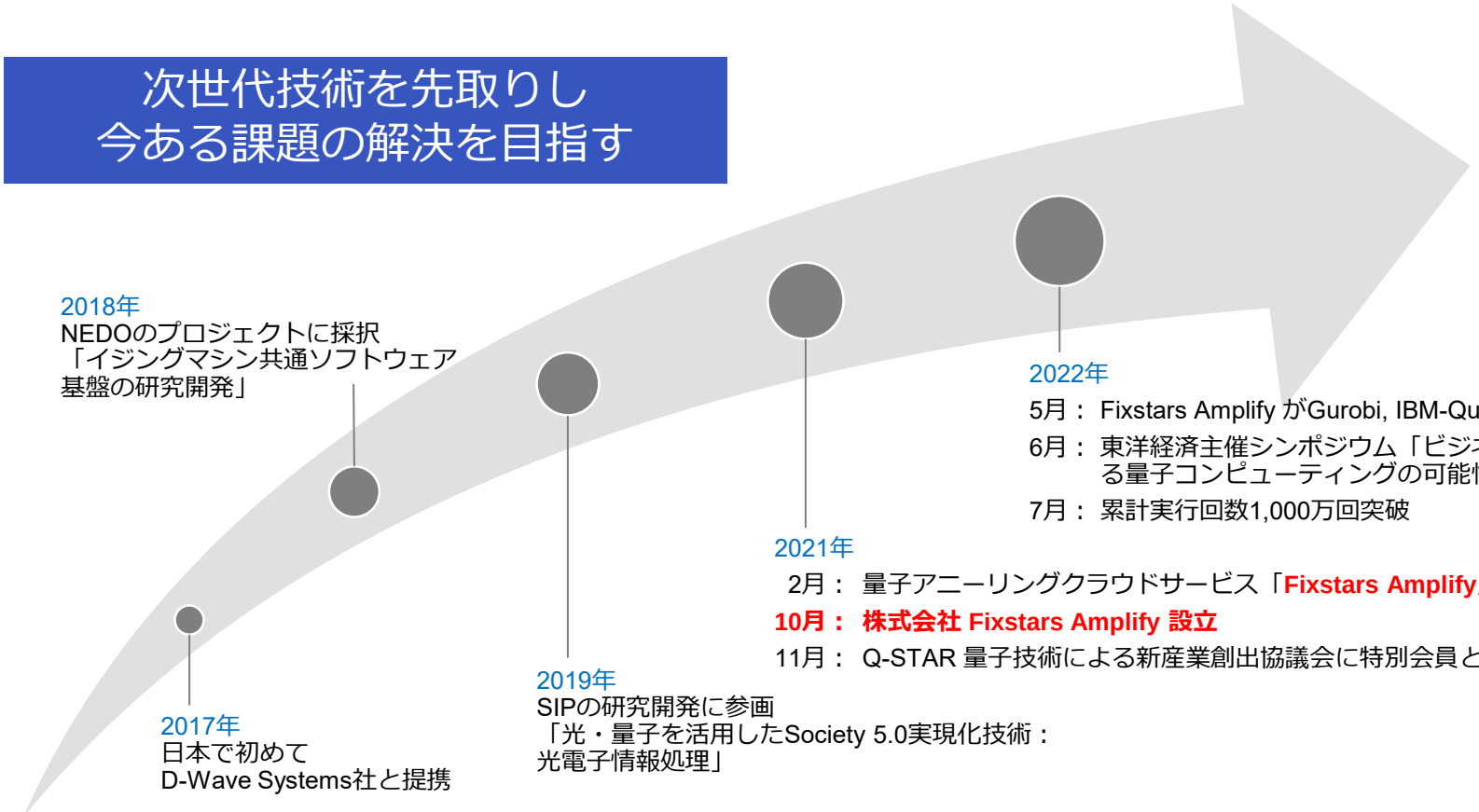
(株) Sider
完全子会社
開発支援SaaS「Sider」を運営

(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

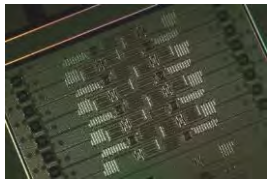
量子技術とFixstars Amplifyの対応領域

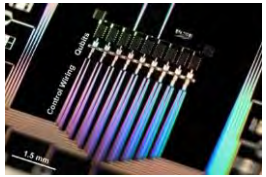
1. 量子コンピュータ

量子ゲート方式

古典汎用コンピュータの上位互換。量子力学の重ね合わせ状態を制御する量子ゲートを操作し、特定の問題を汎用的かつ高速に処理する。

QAOAにより組合せ最適化問題(**QUBO**)を取り扱うことが可能。





2. 量子アニーリング方式

量子焼きなまし法

イジングマシンの一種であり、量子焼きなまし法の原理に基づいて動作する。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する。自然計算により低エネルギー状態が出力される。組合せ最適化問題(**QUBO**)を扱う専用マシン。

3. イジングマシン

二値二次多項式模型

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**)を扱う専用マシン。

変数は0,1または±1。統計物理学におけるイジング模型(磁性体の性質を表す模型)に由来。様々な実装により実現されている。

Amplify AE

FIXSTARS Amplify

組合せ最適化問題 (QUBO)

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)

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

$f(\boldsymbol{q})$を最小化するような $\boldsymbol{q}$ を求める

クラウドサービス：**Fixstars Amplify**

f: 目的関数　q: 決定変数　Q: 係数

Fixstars Amplify とは

- **いつでも**　開発環境 と 実行環境 がセット
  すぐにプログラミングと実行が出来る
- **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」

Fixstars Amplify の内容と特徴

- 開発環境：Amplify SDK
- 実行環境：Amplify Annealing Engine (AE)

開発環境：Fixstars Amplify SDK

Fixstars Amplify SDK ならアニーリングのプログラミングが圧倒的に短縮されます

通常のプログラミング

1. **課題を定式化**
   マシンのSDKやAPI仕様に合わせて物理モデルをデータ化
2. **論理モデルへ変換**
   目的関数をマシンの動作モデルで再定義
3. **物理モデルへ変換**
   マシン仕様や制約を考慮した物理モデルに再変換
4. **マシンにデータを入力**
   マシンのSDKやAPI仕様に合わせて物理モデルをデータ化
5. **マシンを実行**
   特定マシンのみで実行可能

Fixstars Amplifyを用いたプログラミング

1. **課題を定式化**
   定式化された数式をプログラムコードで表現
2. **マシンを実行**
   複数マシンの中から選択可能

SDKが提供するAPIが、自動で各マシンに合った形式へ多段変換をして入力。実行結果は逆変換をして、ユーザーにとって結果の解釈が容易な形式で返却されます。

開発環境インストール

```
$ pip install amplify
```

最適化コード例

```python
from amplify import VariableGenerator, FixstarsClient, solve

# 入力モデルの構築
q = VariableGenerator().array("Binary", 2)
f = 1 - q[0] * q[1]

# 実行マシンの設定
client = FixstarsClient()
client.parameters.timeout = 1000

# アニーリングの実行
result = solve(f, client)

# 結果の解釈
solution = q.evaluate(result.best.values)

print(f"result: {q} = {solution}")
# result: [q_0, q_1] = [1. 1.]
```

FIXSTARS Amplify

Fixstars Amplify SDKによるシンプルプログラミング

数独を解くサンプルアプリ

SDKなし
最適化しても
200行以上

SDKあり
30行程度

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

出典:
Wikipdia

富士通・デジタルアニーラの設定用コード

SDKなし
59行

SDKあり
1行

日立CMOSアニーリングマシンの設定用コード

SDKなし
183行

SDKあり
1行

SDKが、各マシンに対して最適な形式に実装式を多段変換！

FIXSTARS Amplify

実行環境：Fixstars Amplify Annealing Engine (AE)

NVIDIA GPU V100/A100 で動作

- 独自の並列化シミュレーテッド
  アニーリングアルゴリズム

WEB経由で計算機能を提供

- 社会実装・PoC・検証が加速
- Amplify SDK の実装を直ぐに実行可能

商用マシンでは最大規模・最高速レベル

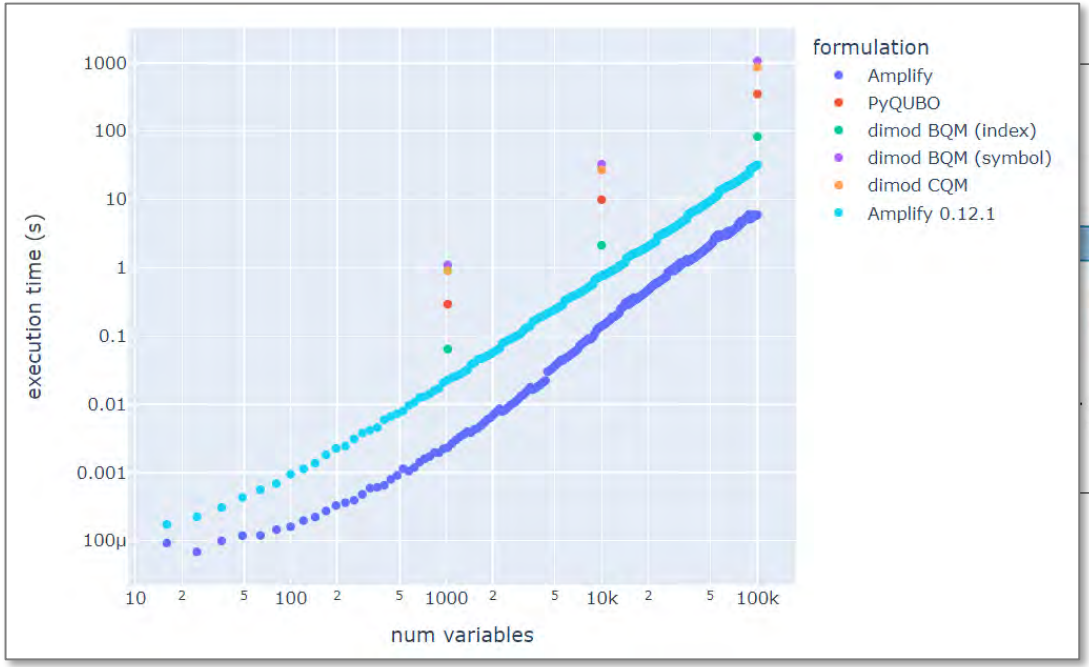
- 120,000 ビット（全結合）
- 260,000 ビット超（疎結合）

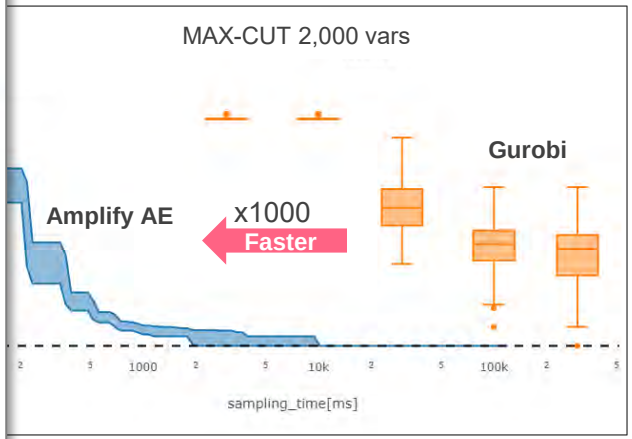
	標準マシン Fixstars Amplify AE	標準マシン D-Wave 2000Q/Advantage	標準マシン 東芝 SQBM+	日立 CMOS Annealing	富士通 Digital Annealer
装置型式	GPU	量子回路	GPU	デジタル回路	デジタル回路
最大ビット数	262,144以上	2,048 (16x16x8)/ 5,760 (16x15x24)	100,000 (SQBM+)/ 10,000 (SBM PoC 版)	61,952 (352x176)	8,192 (DA2)/ 100,000 (DA3)
係数パラメータ	デジタル (32/64bit)	アナログ (5bit程度)	デジタル (32bit)	デジタル (3bit)	デジタル (16/64 bit)
結合グラフ	全結合	キメラグラフ/ ペガサスグラフ	全結合	キンググラフ	全結合
全結合換算ビット数	131,072	64/124	31,000程度 (SQBM+)(*1)/ 1,000 (SBM PoC 版)	176	8,192 (DA2)/ 100,000 (DA3)
APIエンドポイント	Fixstars Amplify	D-Wave Leap	Fixstars Amplify / AWS	Annealing Cloud Web	DA Cloud

※評価・検証用途では無償提供

Fixstars Amplify SDK/AE パフォーマンス

Fixstars Amplify は最速レベルの定式化・求解速度を達成しています　←「高速に」





AE 求解性能・速度

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

様々な分野で利用が拡大しています

豊田通商株式会社 TOYOTA TSUSHO CORPORATION

NTT DATA

NRI 野村総合研究所 Nomura Research Institute

住友商事株式会社

IPA Better Life with IT

登録社・組織数: 約600

累計実行回数: 3,000万回超

Fusic Co.,Ltd.

慶應義塾 Keio University

早稲田大学

株式会社ネクスティ エレクトロニクス

豊田中央研究所

Fixstars Amplify ユーザー様事例のご紹介
（組合せ最適化事例）

最適配置自動化サービス（物流梱包業務のDX）

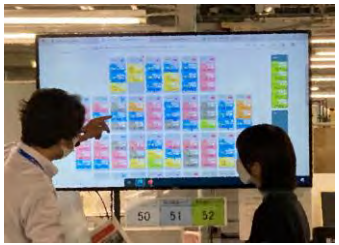
https://www.fixstars.com/ja/services/cases/amplify-bellemaison

業務内容：
梱包業務担当者（１チーム３名）をコンベア前のブースに割り当て

従来の方法：
前日夕方に、翌日の予測出荷目標数と出勤予定に基づいて、３人程度のリーダーが相談し**数時間をかけて**決定

課題：
公平になるよう、様々な配慮を行う必要があり、割り当て担当者に心理的負担がかかっていた

成果：　→ 2022年10月より実稼働開始！
アニーリング技術を活用して自動化・デジタル化
- 作業時間 → 15分程度に
- 心理負担 → ほぼゼロに



手動配置
一部事前配置
自動配置結果の微調整

自動配置（アニーリング）
各種条件を満たす形で未配置のメンバーを一括割り当て

- 割当業務の時間削減
- 担当者の心理的負担低減
- 配置情報のデジタル化
- Smileboard Connectと連携

※Smileboard Connect: 住友商事様開発の物流業務可視化サービス

国家プロジェクトSIP「光・量子を活用したSociety 5.0実現化技術」の一環として、住友商事、SCSK、ベルメゾンロジスコと、2019年より共同研究

FIXSTARS Amplify

生産計画最適化 (電気機器製造メーカー A社様)

複数の製品事業部から様々なプリント基板の注文を受け、生産を行う部門

課題

生産する基板に応じて製造装置の部品や材料を交換する「段取り時間」が必要。段取り時間を考慮した効率的な生産スケジュールを作成したい
従来は、専任者が、一日数回・毎回数十分かけて経験に基づいてスケジュールを作成。更なる生産性向上やノウハウ継承のため、生産スケジュール作成の自動化に着手

FIXSTARS Amplify

効果

生産スケジュール作成の時間・コストの大幅な削減！
(一日あたり数時間 → 数分)

段取りのための製造装置の停止回数の削減！
(10%以上削減)

最適化未経験のご担当者様1人がプログラム試作開始から約1～2ヵ月間取り組んでこの効果を実現
現在は試作段階で、実運用に向けてモデルを改良中！

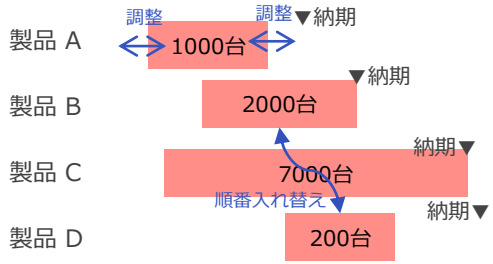
次期フェーズでは、Amplifyの活用領域の拡大を検討中！

生産計画最適化 (機器製造メーカー B社様)

各製品の生産数に応じて製造人員数の計画をする部門

課題

少量多品種の機器を製造している。それぞれの製品は生産が開始できる時期や納期が決まっていて、各製品をどの生産ラインでいつ製造するかによって日々の必要人員数が変化する。人員数と日毎の人員数の増減を最小化したい。
従来は、専任者が2名で日々計画を立案・修正していたが、効率的な生産計画の立案、業務負荷削減、属人化解消のため自動化に着手。

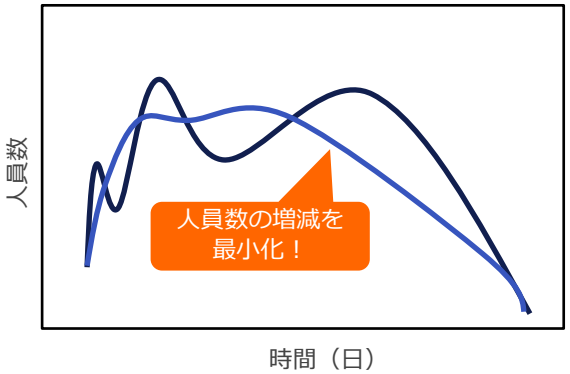


FIXSTARS Amplify

効果

必要人員数と日毎の人員数の増減の最小化！

計画立案の業務負荷削減・属人化解消！



Fixstars Amplify を活用した研究事例

	大学	研究室	論文タイトル
量子アニーリング/イジングマシンに関する研究	早稲田大学	戸川研究室	イジングマシンによる制約付きグラフ彩色問題の彩色数最小化手法 (リンク)
	東京大学	Prof. Codognet	Modeling the Costas Array Problem in QUBO for Quantum Annealing (リンク)
	名古屋大学	片桐研究室	Amplifyを用いたCMOSアニーリングマシンの特性の分析 (リンク)
	東北大学	小松研究室	組み合わせクラスタリングによるアニーリングマシンの評価 (リンク)
応用研究	慶應大学	村松研究室 (材料工学)	Phase-fieldモデルの量子アニーリングシミュレータ (リンク)
	東京大学	長谷川研究室 (量子ゲート)	ISAAQ:イジングマシンを活用した量子コンパイラ (リンク)
	山梨大学	鈴木研究室 (情報工学)	量子アニーリングによる疎行列直接解法向けフィルイン削減オーダリング (リンク)
	東京大学	津田研究室 (MI)	Designing metamaterials with quantum annealing and factorization machines (リンク)
	京都大学	野田研究室 (電子工学)	量子アニーリングを活用したフォトニック結晶レーザーの構造最適化 (リンク)
	東京大学	津田研究室 (MI)	Chemical Design with GPU-based Ising Machines (リンク)

Fixstars Amplify ご利用プラン

料金のご紹介

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

Fixstars Amplify クラウド利用料

	ベーシックプラン 評価・検証用の無料プラン 使い始める	スタンダードプラン 実運用レベルで使いたい人は 見積もりを依頼する	プレミアムプラン 最高性能で計算したい人は 見積もりを依頼する	
利用料金	無料	月額10万円（税込11万円）	月額20万円（税込22万円）	月額60万円（税込66万円）
ユーザー数	1	1	1	5
計算環境	スモール	ミディアム	ラージ	
D-Waveマシンの無料実行	3分/月	3分/月	3分/月	
サポート	ベーシック	スタンダード	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～

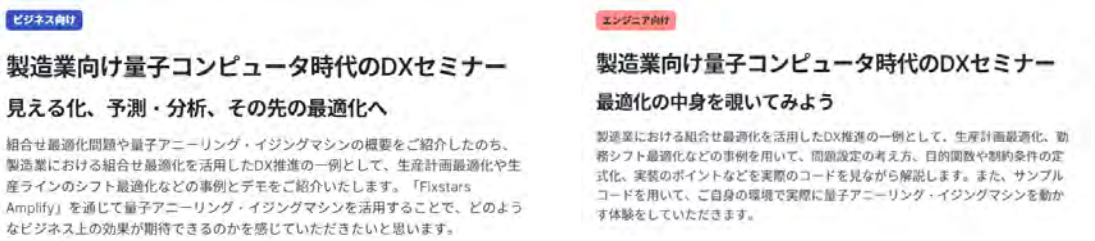
セミナー・トレーニングのご紹介

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。

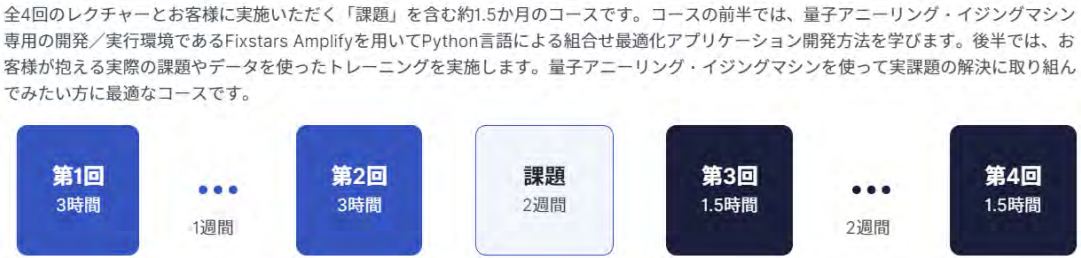
無料セミナー・ワークショップ

ビジネス向け、エンジニア向けに分けて
開催しています！



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！



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）

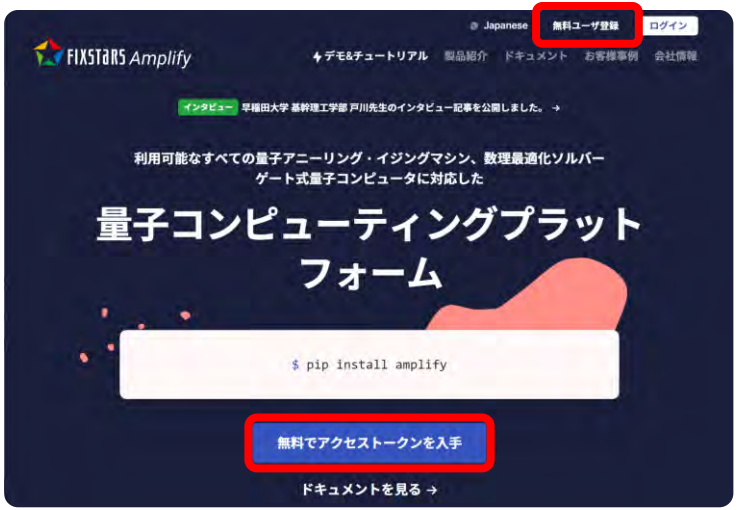
Google Colab 検索

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 = "**********" # 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)
```

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）

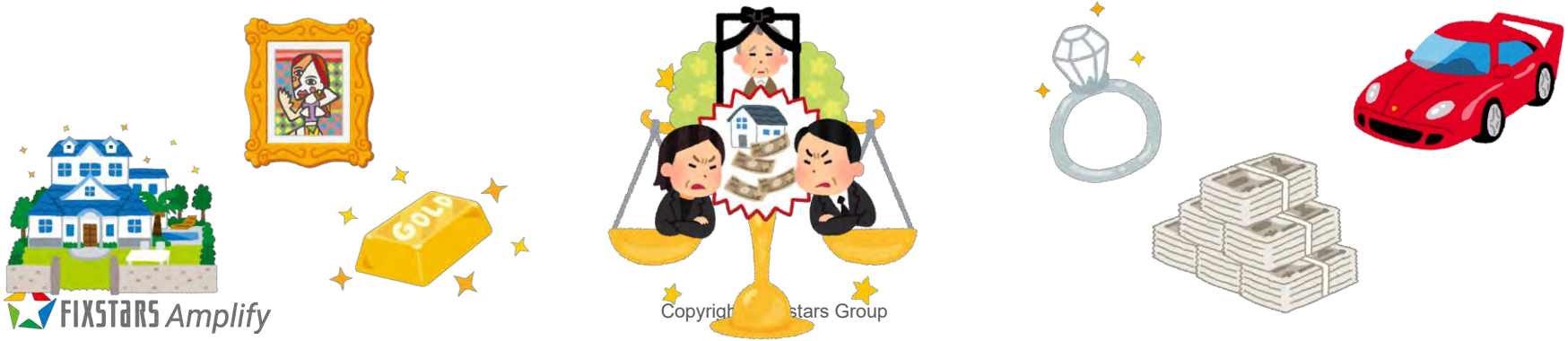
サンプルコード

- ワークショップで使うサンプルコードを取得して下さい。
- それぞれのサンプルコードにご自身のトークンを入力いただく必要があります。
- 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

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

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

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

絶対値を二次式で表す

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

±1をバイナリで表す

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

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

数の分割問題（プログラムコード）

1. 定式化

- 問題の定義と決定変数生成器による決定変数の生成

```
a = [2, 10, 3, 8, 5, 7, 9, 5, 3, 2]
q = amplify.VariableGenerator().array("Binary", len(a))
```

- 目的関数、$f = \left(\sum_{i=1}^{N}(2q_i - 1)a_i\right)^2$、の定式化（①②③は同等）

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

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

③
```
f = amplify.sum((2 * q - 1) * a) ** 2
```

色々な書き方が出来る

2. 実行

```
result = amplify.solve(f, client)
```

3. 結果

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

得られた目的関数の値0

各集合の合計値27

各数字に対して、集合0か、集合1か

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

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

- 課題に対し、定式化により**QUBO**式を取得可能
  - 数の分割

    $f = [\Sigma(2q_i - 1)a_i]^2$
  - 生産計画最適化

    $f = \Sigma\Sigma\left(\Sigma a_i p_i q_{m,i,t} + \Sigma\Sigma b_i s_i q_{m,i,t} q_{m,i-1,f}\right) \cdots$
  - TSP問題（お遍路巡り最短経路探索）

    $f = \Sigma\Sigma\Sigma d_{i,j} q_{n,i} q_{n+1,j} \cdots$
- **QUBO**式に対し、直接アニーリングを行う（量子アニーリング・イジングマシン）

ブラックボックス最適化

- 課題が複雑で直接の定式化が不可能。

  例1) 合成素材の性能向上に必要な、合成対象の材料の組合せ最適化

  例2) 最も計測精度が高くなる計測センサー群の配置は？

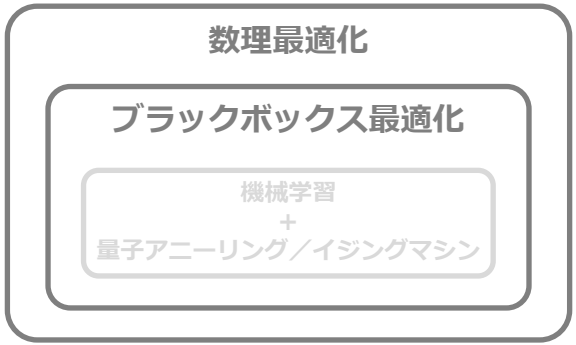
  ➔ 実験かシミュレーション（コスト大）
- シミュレーションや実験の試行回数の削減

  ➔ シミュレーションや実験が楽になる
- 逆問題に対するアプローチとしても適用化

  例：ある計測値を実現する実験条件を見つける

FMQAの紹介

ブラックボックス最適化



目的関数値 $f(x)$ のみ観測可能

- 目的関数の形状や勾配などは何も分からない。

目的関数の評価回数に限りがある

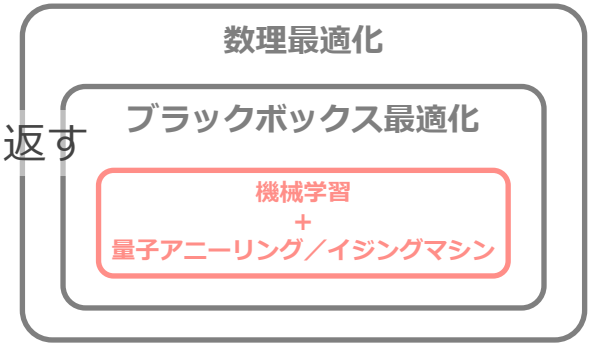
- 入力集合 Ω が有限集合であっても全検索できない。

イメージ

- 5種類の材料から、いくつかの材料を選択・合成し、最も電気抵抗の小さな物質を作る。
  - 決定変数（材料の選択肢）x ・・・ 5桁の01ビットでどの材料を選択するかを記述

    $x = [1,0,0,0,0],\ [1,1,0,0,0],\ [1,0,1,0,0],\ \cdots,\ [1,1,1,1,1] \rightarrow (2^5 - 1)$ 通り
  - 目的関数 $f(x)$ = 合成物質の電気抵抗（実験又は数値シミュレーション）

ブラックボックス最適化アプローチ



ブラックボックス関数 $f(\boldsymbol{x})$ に対して、以下サイクルを繰り返す
（実験・シミュレーション）

（フローはベイズ最適化に似ている、本質は異なる）

① 初期教師データを構築（N_0 回の目的関数の評価）

② 教師データから獲得関数 $g(\boldsymbol{x})$ を構築

③ 獲得関数 $g(\boldsymbol{x})$ が最小となる点 $\hat{\boldsymbol{x}}$ を推定

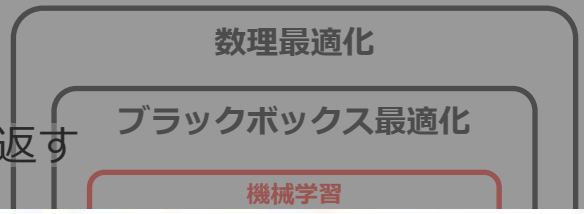
④ 目的関数（実験／ミュレーション）の評価結果 $(\hat{\boldsymbol{x}}, f(\hat{\boldsymbol{x}}))$ を教師データに追加

②～④を N 回繰り返す

課題：
1. $g(\boldsymbol{x})$ の構築？ → Factorization Machine（機械学習モデル）
2. $g(\boldsymbol{x})$ 最小化を実現する $\hat{\boldsymbol{x}}$ の推定？ → 学習モデルをQUBOとしてAmplifyで求解

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アプローチ



ブラックボックス関数 $f(\boldsymbol{x})$ に対して、以下サイクルを繰り返す
（実験・シミュレーション）

（フローはベイズ最適化に似ている、

① 初期教師データを構築（N_0 回の目

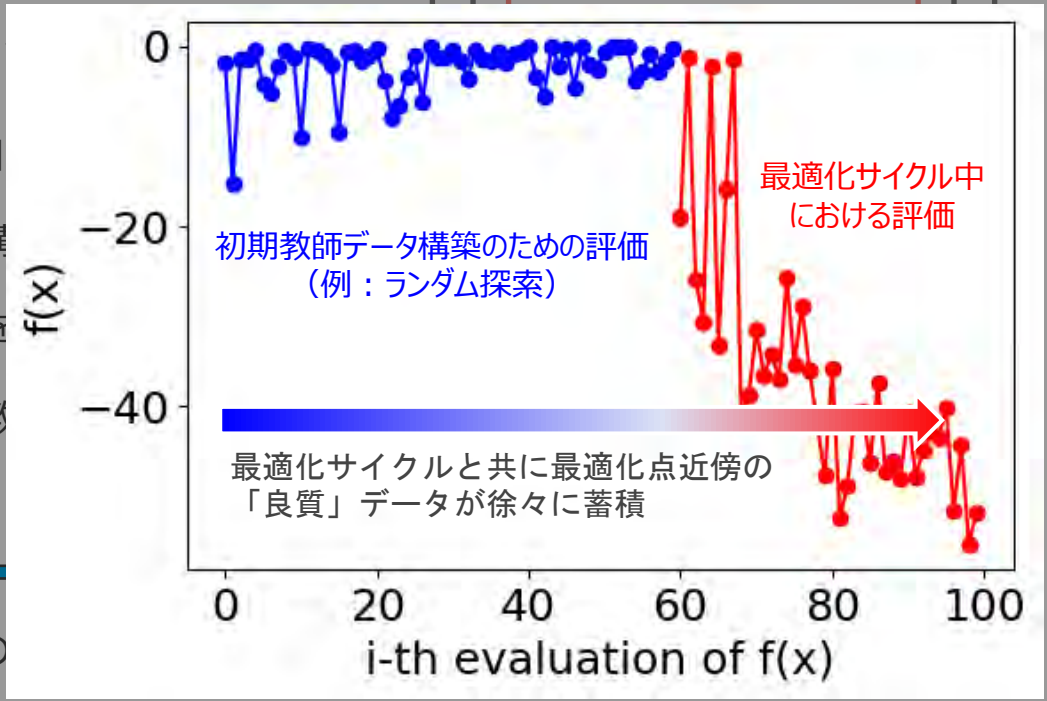
② 教師データから獲得関数 $g(\boldsymbol{x})$ を構

③ 獲得関数 $g(\boldsymbol{x})$ が最小となる点 $\hat{\boldsymbol{x}}$ を

④ 目的関数（実験 ミュレーション）の評価結果 $\left(\hat{\boldsymbol{x}}, f(\hat{\boldsymbol{x}})\right)$ を教

課題： 1. $g(\boldsymbol{x})$ の構築？

2. $g(\boldsymbol{x})$ 最小化を実現する $\hat{\boldsymbol{x}}$ の



獲得関数としての Factorization Machine (FM)

- 獲得関数 $g(\boldsymbol{x})$ 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** と呼ぶ場合がある。

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

ブラックボックス最適化 活用例

- デモ・チュートリアルページで公開の活用例及び非公開の活用例

Amplify デモ　検索

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

FMQA入門

物理・材料

プラント工学

流体工学

交通制御

機械学習

ブラックボックス最適化

1. 機械学習 (FM) ハンズオン
2. FMQA ハンズオン

ブラックボックス最適化のデモプログラム

ある代数式を未知のブラックボックス関数と見なした場合の FM 機械学習
- https://colab.research.google.com/drive/1SilVRpoZUqVp_jeCrn9LYPucPfzpB_ag

問題設定

- モデル化対象の真の関数 $f(\boldsymbol{x})$ として、

$$f(\boldsymbol{x}) = \boldsymbol{x}^T Q \boldsymbol{x}$$

を考慮。プログラム内で Q は乱数で定義

最適化サイクル

1. 教師データから獲得関数 $g(\boldsymbol{x})$ を構築 (FM)
2. 獲得関数 $g(\boldsymbol{x})$ が最小となる点 $\hat{\boldsymbol{x}}$ を推定 (QA)
3. 評価結果 $(\hat{\boldsymbol{x}}, f(\hat{\boldsymbol{x}}))$ を教師データに追加

↑のサイクルにより、最適化点近傍におけるFMの予測精度が向上し、組合せ最適化により、より良い $\hat{\boldsymbol{x}}$ の推定が期待される

FM デモプログラム 1/5（ブラックボックス関数の定義）

```python
import numpy as np
from typing import Callable, Any

# 乱数シードの固定
seed = 1234
rng = np.random.default_rng(seed)


def make_blackbox_func(d: int) -> Callable[[np.ndarray], float]:
    """入力が長さ d のバリナリ値のベクトルで出力が float であるような関数を返却する
    rng = np.random.default_rng(seed)
    Q = rng.random((d, d))
    Q = (Q + Q.T) / 2
    Q = Q - np.mean(Q)

    def blackbox(x: np.ndarray) -> float:
        assert x.shape == (d,)  # x は要素数 d の一次元配列
        return x @ Q @ x  # type: ignore

    return blackbox
```

- 乱数シードの固定
  - 実行毎の再現性のため
- ブラックボックス関数を生成する関数
  - 生成後のブラックボックス関数の使い方

```python
d = 100
blackbox = make_blackbox_func(d)

x = [0, 1, 0 , 1, ...]
answer = blackbox(x)
```

FM デモプログラム 2/5（FMの定義）

- FM モデルを PyTorch で定義する

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

```python
import torch
import torch.nn as nn

# 乱数シードの固定
torch.manual_seed(seed)


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 = torch.randn((d, k), requires_grad=True)
        self.w = torch.randn((d,), requires_grad=True)
        self.w0 = torch.randn((), requires_grad=True)
```

```python
    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

        Args:
            x (torch.Tensor): (データ数 × d) の 2 次元 tensor

        Returns:
            torch.Tensor: y の推定値 の 1次元 tensor (サイズはデータ数)
        """
        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

    def get_parameters(self) -> tuple[np.ndarray, np.ndarray, float]:
        """パラメータ v, w, w0 を出力する"""
        np_v = self.v.detach().numpy().copy()
        np_w = self.w.detach().numpy().copy()
        np_w0 = self.w0.detach().numpy().copy()
        return np_v, np_w, float(np_w0)
```

FM デモプログラム 3/5（FMの学習）

```python
def train(x: np.ndarray, y: np.ndarray, model: TorchFM, epochs: int = 2000, lr: flo
    # モデルの最適化関数
    optimizer = torch.optim.AdamW([model.v, model.w, model.w0], lr=lr)
    # 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])
    train_loader = DataLoader(train_set, batch_size=8, shuffle=True)
    valid_loader = DataLoader(valid_set, batch_size=8, shuffle=True)

    # 学習の実行
    min_loss = 1e18  # 損失関数の最小値を保存
    best_state = model.state_dict()  # モデルの最も良いパラメータを保存

    # `range` の代わりに `tqdm` モジュールを用いて進捗を表示
    for _ in trange(epochs, leave=False):
        # 学習フェイズ
        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**: 学習率

学習後、最も良いモデルに対し、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
y_pred = model(x_tensor)
print(f"corrcoef: {torch.corrcoef(torch.stack([y_tensor, y_pred]))[0
print(f"RMS error: {min_loss:.2f}")
```

FM デモプログラム 4/5（教師データ作成）

```python
def init_training_data(d: int, n0: int):
    """n0 組の初期教師データを作成する"""
    assert n0 < 2**d

    # n0 個の 長さ d の入力値を乱数を用いて作成
    x = rng.choice(np.array([0, 1]), size=(n0, d))

    # 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] != n0:
        x = np.vstack((x, np.random.randint(0, 2, size=(n0 - x.shape[0], d))))
        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(n0)
    for i in range(n0):
        y[i] = blackbox(x[i])

    return x, y


N0 = 50  # 初期教師データの数
x_init, y_init = init_training_data(d, N0)
```

教師データを乱数により生成

- **d**: 入力サイズ
- **n0**: 教師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 デモプログラム 5/5（メイン部分）

- 機械学習の実施

```
x, y = x_init, y_init

# 機械学習モデルの作成
model = TorchFM(d, k=10)
# モデルの学習の実行
train(x, y, model, epochs=2000, lr=0.1)
```

```
corrcoef: 0.92
RMS error: 128.43
```

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

1. 機械学習 (FM) ハンズオン
2. FMQA ハンズオン

ブラックボックス最適化のデモプログラム

ブラックボックス最適化による模擬超電導材料の探索
https://colab.research.google.com/drive/1E3Cy6Fe4EG-M1lE_33-Djp93aSZAUiQz

ブラックボックス関数

- ブラックボックス関数 $f(\boldsymbol{x})$ として、与えられた材料組合せによる模擬超電導材料の臨界温度を返却するモデル関数を考慮

※本関数はあくまでも実験やシミュレーションの代用であり、その中身やパラメータについては未知であるとして扱い、関数評価の回数にも制限があるものとして取り扱います。

最適化サイクル

1. 教師データから獲得関数 $g(\boldsymbol{x})$ を構築 (FM)
2. 獲得関数 $g(\boldsymbol{x})$ が最小となる点 $\hat{\boldsymbol{x}}$ を推定 (QA)
3. 評価結果 $(\hat{\boldsymbol{x}}, f(\hat{\boldsymbol{x}}))$ を教師データに追加

↑のサイクルにより、最適化点近傍におけるFMの予測精度が向上し、組合せ最適化により、より良い $\hat{\boldsymbol{x}}$ の推定が期待される

FMQA デモプログラム 1/3 （ブラックボックス関数の定義）

```python
def make_blackbox_func(d: int) -> Callable[[np.ndarray], float]:
    """入力が長さ d のバリナリ値のベクトルで出力が float であるような関数を返却する"""

    def set_properties(size: int) -> tuple[np.ndarray, np.ndarray, np.ndarray]:
        """（ランダムに決定した）材料物性係数を返却する"""
        mu, sigma, ratio = 0.0, 1.0, 0.2
        table1 = rng.random(size) * 1e5 * (0.1 * math.log(size) - 0.23)
        table2 = rng.lognormal(mu, sigma, size) * ratio
        table3 = rng.lognormal(mu, sigma, size) * ratio
        return table1, table2, table3

    def supercon_temperature(x, debye_table, state_table, interaction_table):
        """与えられた材料の組合せ（長さ d のバリナリ値のベクトル）に対し、臨界温度を計算し返却する"""
        debye_temperature = np.sum(x * debye_table) / np.sum(x)
        state_density = np.sum(x * state_table) / np.sum(x)
        interaction = np.sum(x * interaction_table) / np.sum(x)
        crit_temp = debye_temperature * math.exp(-1.0 / state_density / interaction)
        return crit_temp

    # 係数テーブルの準備
    debye_temperature_table, state_density_table, interaction_table = set_properties(d)

    # ブラックボックス関数の定義
    def blackbox(x: np.ndarray) -> float:
        """与えられた材料の組合せ（長さ d のバリナリ値のベクトル）に対し、超電導臨界温度の負値を返却する"""
        assert x.shape == (d,)  # x は要素数 d の一次元配列
        t_c = supercon_temperature(
            x, debye_temperature_table, state_density_table, interaction_table
        )
        return -t_c

    return blackbox
```

関数の利用例：

```python
num_materials = 100  # 決定変数のサイズ（材料選択肢の数）
blackbox_func = make_blackbox_func(num_materials)
critical_temp = blackbox_func(x)
```

FMQA デモプログラム 2/3 （アニーリング部分）

```python
def anneal(torch_model: TorchFM) -> np.ndarray:
    """FM モデルを受け取り、それらのパラメータにより記述されるモデルの最小値を与える x を求める"

    # 長さ d のバイナリ変数の配列を作成
    gen = VariableGenerator()
    x = gen.array("Binary", torch_model.d)

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

    # 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_model = Model(objective)  # 組合せ最適化モデルを構築

    # ソルバーの設定
    client = FixstarsClient()
    client.token = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
    client.parameters.timeout = timedelta(milliseconds=2000)

    # 最小化を実行
    result = solve(amplify_model, client)

    # モデルを最小化する入力ベクトルを返却
    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

- 決定変数配列の作成
- 学習済みモデルからパラメータを取得
- パラメータに基づき目的関数 $g(x)$ を作成
- ソルバーの設定
  - ※トークン入力を忘れずに
- solve の実行
- このサイクルにおける $\hat{x}$ を返却

FMQA デモプログラム 3/3（メイン部分）

```
N = 40  # FMQA サイクルの実行回数

x, y = x_init, y_init  # 教師データの初期化

# N - N0 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):
    model = TorchFM(d=num_materials, k=10)  # 機械学習モデルの作成

    train(x, y, model)  # モデルの学習の実行

    x_hat = anneal(model)  # 学習済みの最小値を与える入力ベクトルの値を取得

    # x_hat が重複しないようにする
    while (x_hat == x).all(axis=1).any():
        flip_idx = rng.choice(np.arange(num_materials))
        x_hat[flip_idx] = 1 - x_hat[flip_idx]

    y_hat = blackbox_func(x_hat)  # x_hatを用いてブラックボックス関数評価

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

    tqdm.write(f"FMQA cycle {i}: found y = {y_hat:.1f}; current best = {np.min(y):
```

実際にサンプルプログラムを実行してみましょう！

デモプログラムをベースに、

- サイクル数 (N)
- 初期教師データ数 (N0)
- FMハイパーパラメータ (k)

などを変化させた場合、最適化結果や最適化過程はどのように変化するでしょうか。

また、目的関数を変更し、最適化を実施してみましょう。

FMQA デモプログラムの実務での活用方法

```python
def make_blackbox_func(d: int) -> Callable[[np.ndarray], float]:
    rng = np.random.default_rng(seed)
    Q = rng.random((d, d))
    Q = (Q + Q.T) / 2
    Q = Q - np.mean(Q)

    def blackbox(x: np.ndarray) -> float:
        assert x.shape == (d,)  # x は要素数 d の一次元配列
        return x @ Q @ x  # type: ignore

    return blackbox
```

```python
def make_blackbox_func(d: int) -> Callable[[np.ndarray], float]:
    # 必要に応じて最適化対象以外の条件設定や初期設定などをここで実施

    def blackbox(x: np.ndarray) -> float:
        # x の組合せを元に実験を実施
        # または、シミュレーションを実施
        # 実験・シミュレーション結果を後処理し、最小化したい目的関数値を取得
        objective = hogehoge
        return objective  # type: ignore

    return blackbox
```

基本的に、blackbox()を変更する。必要に応じて、現在の教師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、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
  ➔ 実施例のデモ
- $h(x)$ を最大にするような入力 x を推定する場合は、$1/h(x)$ や $-h(x)$ などを目的関数 $f(x)$ とする。

今後について

- デモ・チュートリアルにチャレンジ！

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

制約のみで定式化

チュートリアル応用編

会議室割当問題

デモアプリケーション

グラフ彩色問題

目的関数＋制約で定式化

チュートリアル応用編

タクシーマッチング問題

デモアプリケーション

巡回セールスマン問題

困った時は
ドキュメンテーションを！

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

FIXSTARS Amplify

Q&A