A

付録A

AMPL モデリングの流儀

現在推奨される「現代的な」AMPL コードの書き方をまとめる

この章の内容

1. 現代的な AMPL のワークフロー
2. ファイルの分け方
3. amplpy の書き方
4. AMPL MP 式
5. ソルバーの選び方

A.1 現代的な AMPL のワークフロー

.dat も .run も使わない。Python がオーケストレーション層になる

ワークフローは 2 つの要素からなる

.dat ファイルと .run ファイルは使わない（AMPL 単独のスクリプト環境だった時代の遺物）

1 .mod ファイル

モデルの定義（集合、パラメータ、変数、目的関数、制約）を書く。データは埋め込まない

2 Python ファイル

amplpy でそれ以外のすべてを担う。モデルとデータの読み込み、ソルバーの設定、求解、結果の確認

3 データの出どころ

多くの場合 pandas の DataFrame や Python の辞書から amplpy を通して渡す

避けるべき書き方と推奨される書き方

避けるべき書き方

- .dat ファイル・.run ファイルを書かない
- AMPL IDE ではなく VS Code 拡張を使う
- big-M など古いやり方の線形化をしない
- データの読み込みに ampl.eval() を使わない

推奨される書き方

- モデル定義だけのすっきりした .mod
- amplpy で読み込み・求解・結果取得
- 論理・条件・カウントなどの MP 式を使う
- ソルバーは既定で HiGHS を使う
- 求解後は solve_result を必ず確認

AMPL 構文のよくある落とし穴

- ord() には ordered または circular な集合が必要
  - ord(j1) < ord(j2) を使うなら set JOBS ordered; と宣言
- a <= expr <= b の二重不等式は正しい構文
  - 制約を 2 本に分けるより好ましい
- セミコロンは宣言の終わりを表す（1 行の終わりではない）
- subject to、subj to、s.t. はいずれも制約を表す

! ポイント

ただの set に ord() を使うとエラーになる。複数行にわたる制約は 1 つの文なので、セミコロンは 1 つだけ付く。

A.2

ファイルの分け方

モデルとオーケストレーションを 1 つのファイルに混ぜない

2 つの別々のファイルに分ける

どちらがどちらか分かるようにファイル名を付けておく

1 .mod ファイル

モデルの定義のみを書く

2 Python ファイル

amplpy によるオーケストレーションのコードを書く

A.3

amplpy の書き方

詳しい API の例は付録 B。ここでは要点をまとめる

インストール

```bash
# AMPL の Python API をインストールする
python -m pip install amplpy --upgrade

# ソルバーモジュールをインストールする
# （必要なものだけ入れればよい）
python -m amplpy.modules install highs gurobi xpress cplex

# ライセンスを有効化する
# （無償ライセンスは ampl.com/ce または ampl.com/courses）
python -m amplpy.modules activate <your-license-uuid>
```

amplpy の基本の骨格（1）読み込み

solve.py

```
from amplpy import AMPL
import pandas as pd

ampl = AMPL()

# モデルを読み込む
ampl.read("model.mod")

# pandas の DataFrame からデータを読み込む
ampl.set_data(df, "SET_NAME")
ampl.get_parameter("param_name").set_values(df)

# 別法: 辞書やリストから読み込む
ampl.set["SET_NAME"] = ["a", "b", "c"]
ampl.param["param_name"] = {"a": 1, "b": 2, "c": 3}
```

- モデルは ampl.read() で読み込む
- set_data は集合を初期化し、列をパラメータに入れる
- 辞書やリストからも渡せる

データの読み込み方

書き方	用途
ampl.set_data(df, "SET_NAME")	インデックスが集合の要素、各列がパラメータの値（構造化データに推奨）
ampl.get_parameter("p").set_values(df)	行列など複数の集合で添字付けされたパラメータ
ampl.set["S"] = [...]	集合を初期化する（辞書・リストから）
ampl.param["p"] = {...}	添字から値への辞書でパラメータを設定する
ampl.param["p"] = {("i1","j1"): val, ...}	多次元のパラメータ

1000 万行を超えるデータは、テーブルハンドラ（*amplcsv*、*amplxl*、*eodbc*）で直接読み込むことも検討する

amplpy の基本の骨格（2）求解と結果

solve.py

```
# ソルバーを設定する
ampl.option["solver"] = "highs"
# 必要ならソルバー固有のオプションを渡す
ampl.option["highs_options"] = "outlev=1"

# 解く
ampl.solve()
assert ampl.solve_result == "solved"

# 結果を取り出す
obj_value = ampl.get_objective("ObjName").value()
results_df = ampl.get_variable("VarName").get_values().to_pandas()

# 任意の式を取り出す
expr_df = ampl.get_data("{i in SET} expr[i]").to_pandas()
```

結果の取り出しとソルバーの設定

solve.py

```
# 目的関数値
ampl.get_objective("Cost").value()
# 変数の値を pandas の DataFrame として
ampl.get_variable("X").get_values().to_pandas()
# 制約の双対変数
ampl.get_constraint("Balance").get_values().to_pandas()
# 任意の AMPL の式
ampl.get_data("{i in NODES} Capacity[i] - Flow[i]").to_pandas()

ampl.option["solver"] = "highs"
# ソルバー固有のオプションは {solver}_options と書く
ampl.option["highs_options"] = "outlev=1 timelimit=300"
ampl.option["gurobi_options"] = "outlev=1 timelimit=300 mipgap=0.01"
```

A.4 AMPL MP 式

ここが肝心である。手作業で線形化する必要はない

現行の AMPL のソルバーはすべて MP ベース

- Gurobi、CPLEX、COPT、Xpress、Mosek
- BARON、HiGHS、cuOpt、CBC、SCIP、GCG
- MP2NL メタソルバー
  - Knitro、Ipopt などの非線形ソルバーは MP2NL 経由で MP の機能を利用
- 式をネイティブに、または自動的な書き換えで処理する

! ポイント

対応する式の全一覧と例は付録 C にある。big-M などによる手作業の書き換えは不要である。

MP 式の大きな分類

分類	演算子
条件演算子	if-then-else、==>、<==、<==>
論理演算子	or、and、not、exists、forall
計数（カウント）演算子	count、atleast、atmost、exactly、numberof、alldiff
区分線形関数	abs、min、max、<< 折れ点; 傾き >>
比較・関係演算子	!=、alldiff
非線形関数	exp、log、sin、cos、tan、sinh、cosh、tanh と逆関数

条件演算子と論理演算子

model.mod

```
# 目的関数の中で変数に依存する条件式を使う
minimize TotalCost:
   sum {j in JOBS, k in MACHINES}
      if MachineForJob[j] = k then cost[j,k];

# 含意の制約：MP はこれをそのまま扱える
subject to Least_Use {j in SCHEDS}:
   Use[j] = 1 ==> Work[j] >= least_assign else Work[j] = 0;

# 選言的な制約：big-M は不要
subject to No_Conflict
      {i1 in JOBS, i2 in JOBS: ord(i1) < ord(i2)}:
   Start[i2] >= Start[i1] + t_offset[i1,i2] or
   Start[i1] >= Start[i2] + t_offset[i2,i1];

# 添字付きの論理演算子
subject to Multi {i in ORIG, j in DEST}:
   forall {p in PROD} Trans[i,j,p] = 0 or
   minload <= sum {p in PROD} Trans[i,j,p] <= limit[i,j];
```

カウント・区分線形・比較の演算子

model.mod

```
subject to Min_Serve {i in ORIG}:
   atleast minserve {j in DEST} (Ship[i,j] >= minload);

subject to OnePersonPerPosition:
   alldiff {i in 1..nPeople} Pos[i];

maximize Total_Profit:
   sum {p in PROD, t in 1..T} revenue[p,t]*Sell[p,t] -
   sum {t in 1..T} time_penalty[t] * abs(Use[t] - avail_min[t]);

minimize Max_Cost:
   max {i in PEOPLE} sum {j in PROJECTS} cost[i,j] * Assign[i,j];

subject to Different_Colors {(c1,c2) in Neighbors}:
   Color[c1] != Color[c2];
```

MP モデリングの要点

- 手作業の線形化より MP の論理的な書き方を選ぶ
  - x > 0 ==> y >= 5 は big-M より明快で間違いが少ない
- 区分線形式には option pl_linearize 0; を設定
- 区分線形近似では変数の範囲をきつく取る
  - [−1e4, 1e4] の内側に収めると数値的に安定
- 可能なら指示制約を使う

! ポイント

binary_var = 1 ==> 制約 の形の指示制約は Gurobi、CPLEX、COPT、Xpress がネイティブに処理する。

A.5 ソルバーの選び方

既定は HiGHS（オープンソース、LP/MIP/QP に対応）

線形・二次のソルバー（MP ベース）(1)

Y：その問題クラスをネイティブに扱える

ソルバー	LP / MILP	QP / MIQP	QCP 凸 / 非凸	錐 / MINLP
Gurobi	Y / Y	Y / Y	Y / Y	Y / Y
Xpress	Y / Y	Y / Y	Y / Y	Y / Y
CPLEX	Y / Y	Y / Y	Y / –	Y / –
COPT	Y / Y	Y / Y	Y / –	Y / –
Mosek	Y / Y	Y / Y	Y / –	Y* / –

* *Mosek* は錐最適化に特化した対応（*SOCP*、*SDP*、指数錐）。*QCP* は *(MI)QCP* を表す

線形・二次のソルバー（MP ベース）(2)

オープンソース系と GPU ソルバー

ソルバー	LP / MILP	QP / MIQP	QCP 凸 / 非凸	錐 / MINLP
SCIP	Y / Y	Y / Y	Y / Y	Y / Y
GCG	Y / Y	– / –	– / –	– / –
HiGHS	Y / Y	Y / –	– / –	– / –
CBC	Y / Y	– / –	– / –	– / –
cuOpt	Y / Y	– / –	– / –	– / –

「–」のクラスも *MP* の区分線形近似で解けることがあるが、得られるのは近似解（警告に *PLApprox* と表示）

非線形ソルバー

MP2NL：MP から NL への変換層を通して MP の機能を利用する

1 NLP・MINLP・大域

BARON（ネイティブ MP）、Lindo、Raposa、Couenne（MP2NL 経由）

2 NLP・MINLP

Knitro、Bonmin（MP2NL 経由）

3 NLP・大域

LGO（MP2NL 経由）

4 NLP

CONOPT、LOQO、MINOS、SNOPT、Ipopt（MP2NL 経由）

ソルバー選択の早見表

線形・二次

- LP/MIP：HiGHS、商用は Gurobi 等
- QP/MIQP：HiGHS（QP のみ）、商用
- 凸 QCP・錐：Mosek、Gurobi など
- 非凸 QCP：Gurobi、Xpress、SCIP

非線形・その他

- MINLP：Gurobi、Xpress、SCIP、BARON 等
- NLP：Knitro、Ipopt、CONOPT 等
- 大域最適化：BARON、Lindo、Couenne 等
- GPU による LP/MIP：cuOpt

まとめ

- ✓ .mod でモデル、Python（amplpy）でそれ以外を担う
- ✓ .dat・.run・ampl.eval() によるデータ読み込みは使わない
- ✓ データは DataFrame や辞書から渡し、solve_result を確認
- ✓ big-M で手作業の線形化をせず MP 式を使う
- ✓ ソルバーの既定は HiGHS、問題に応じて選ぶ