Nuorium Optimizer

C++SIMPLE外部接続マニュアル V27

株式会社NTTデータ数理システム

2024年12月

MATHEMATICAL SYSTEMS INCARNATION

目　次

第 1 章 はじめに

このドキュメントでは，簡単なデモンストレーションを通じて，Nuorium Optimizer[1]と外部プログラムの連結方法を VisualC++の IDE[2]を利用した例を用い解説します．なお，Windows 版についてのみ扱うため，Linux/Unix/macOS 版については Nuorium Optimizer サポート（nuopt-support@ml.msi.co.jp）までお問い合わせください．

外部プログラムとの連結方法には大きく分けて，

（ア）実行形式を作成して別プロセスで起動する（第 2 章）

（イ）ライブラリ `solveLP`，`solveQP` をコールする（第 3 章）

（ウ）C++SIMPLE のモデル記述を手続きの中に記述して利用する（第 4 章）

（エ）C++SIMPLE のモデル記述からクラスを生成して利用する（第 5 章）

という四つの方法があります．このドキュメントではこれらについて順に解説します．各方法の簡単な説明は，次の通りです．

（ア）の方法は Nuorium Optimizer を外部プログラムとして別プロセスで起動する方法です．プロセス起動やデータのやり取りにオーバーヘッドがありますが，実装・デバッグは最も容易です．

（ア）以外の（イ）（ウ）（エ）は，いずれも Nuorium Optimizer をライブラリとしてリンクし，その機能を利用する方法です．

（イ）は LP，MILP，QP に属する問題を C++SIMPLE を介さずに専用の関数を使って解く方法です．高速ではありますが，ユーザ側でデータを標準形に変形する必要がございます．

（ウ）（エ）はいずれも C++SIMPLE を介する方法です．（ウ）は（エ）よりは簡便ですが，Nuorium Optimizer のライブラリがリンクされているプログラム全体で同時に一つの問題しか扱うことができません．（エ）は手順がやや複雑になりますが，複数の問題を一つの外部プログラムで操作することが可能です．

Nuorium Optimizer の対応コンパイラに関するご案内

対応コンパイラに関する情報につきましては，https://www.msi.co.jp/solution/nuopt/spec_table.html をご覧ください．

- **サンプル**

  フォルダ：

  （Nuorium Optimizer のインストール場所）\samples\app

  の下に以下の zip ファイルがあります．

[1]「Nuorium Optimizer」につきまして，V15 までは「NUOPT」・V16 から V23 までは「Numerical Optimizer」と適宜読み替えをお願いいたします．

[2]Microsoft Visual Studio 2017 に基づいて説明します．

- app_VS2015.zip（VS2015 用）
- app_VS2017.zip（VS2017 用）
- app_VS2019.zip（VS2019 用）
- app_VS2022.zip（VS2022 用）

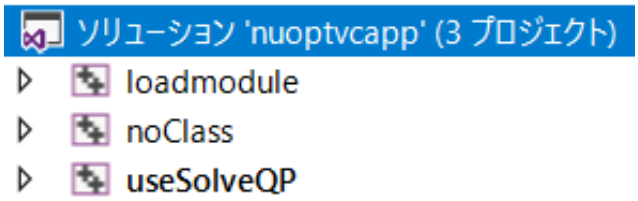
なお，デフォルトの Nuorium Optimizer のインストール場所は

```
C:\Program Files (x86)\Mathematical Systems Inc\NUOPT
```

となっています．
まず，ご利用になるコンパイラに対応した zip ファイルを解凍します．なお，OS の設定によってはサンプルの場所に書き込み権限が無いため，適切なフォルダにコピーしてからサンプルの解凍・実行をする必要があります．
解凍を行なうと，VC++のソリューションである

```
nuoptvcapp.sln
```

があります．
このソリューション内には以下の 3 つのプロジェクトがあります．



これらのプロジェクトが 3 章以降で説明する実行例に対応しています．本マニュアルで例として用いるのは簡単な整数計画問題である次のナップサック問題です．

変数	$x_i \in \{0,1\} \quad (i \in S)$
目的関数（最大化）	$\sum_{i \in S} c_i x_i,$
制約条件	$\sum_{i \in S} a_i x_i \leq b$

S の要素数だけ 0 − 1 変数があり，線形制約が一本，線形の目的関数を最大化するという問題です．この問題を設定するのに必要なデータは，

目的関数の係数 c_i，制約式の係数 a_i，制約式の右辺値 b

となります．この問題を解くというアプリケーションを上記の様々な方法で行います．

1.1 サポートプラットフォーム

Windows 版では，Win32プロジェクト設定ではなく64ビットプロジェクト構成にする必要があります．変更の方法については，「7.1 Microsoft Visual Studio プロジェクトの設定」を参照してください．ま

た，Nuorium Optimizer のインストール時に選択したコンパイラと外部接続で利用するコンパイラが異なっていますと，正常に動作しません．必ず同じコンパイラを用いてください．

UNIX/Linux/macOS 版については Nuorium Optimizer サポート（nuopt-support@ml.msi.co.jp）までお問い合わせください．

1.2 実行環境に含まれるもの

外部プログラムの連結を行うための実行環境には次が含まれます．一部環境によって必要なファイルが異なります．

- インクルードファイル（共通して必要）：

`userapp/include/*.h`	Nuorium Optimizer のインクルードファイル

- VC++版ライブラリとサンプル（VC++でのリンクにのみ必要）：

`userapp/lib/libnuopt_M*.lib`	Nuorium Optimizer ライブラリ（VC++版）
`SAMPLES/app/`	VC++版用サンプルディレクトリ

このライブラリから生成したロードモジュールは Nuorium Optimizer がインストールされている環境でのみ動作し，動作条件は Nuorium Optimizer に準じます．

1.3 サンプルディレクトリの内容

外部ライブラリの利用サンプルディレクトリです．以下がその一覧です．

`nuoptvcapp.sln`	VC++のソリューション
`useSolveQP.cpp`	`solveLP`，`solveQP` を用いるサンプルメイン
`useSolveQP/qp312.txt`	`useSolveQP.cpp` 用データ 1
`useSolveQP/qp312I.txt`	`useSolveQP.cpp` 用データ 2
`useSolveQP/knapsack.txt`	`useSolveQP.cpp` 用データ 3
`useSolveQP/hs23.txt`	`useSolveQP.cpp` 用データ 4
`useSolveQP/useSolveQP.vcxproj`	プロジェクトファイル
`loadmodule/loadmodule.vcxproj`	プロジェクトファイル
`noClass/noClass.vcxproj`	プロジェクトファイル
`noClass.cpp`	モデルの記述と実行を分離しないサンプル
`knapsack.smp`	クラスインターフェーステストモデル
`knapsack.cc`	`genClass` により自動生成されたクラスの実装
`knapsack.h`	`genClass` により自動生成されたヘッダー
`knapsackControl.cc`	`knapsack` のコントロール雛形
`main.cpp`	`knapsackSolve` のメイン
`driver.cpp`	`knapsackSolve` のドライバ

1.4 SIMPLE の行列クラスオブジェクトをご利用になる場合の注意点

外部接続機能をご利用になる際，C++SIMPLE のモデル記述において対称行列 SymmetricMatrix などの SIMPLE のクラスオブジェクトを使用する場合には原則として

```
#include "simpleMatrix.h"
```

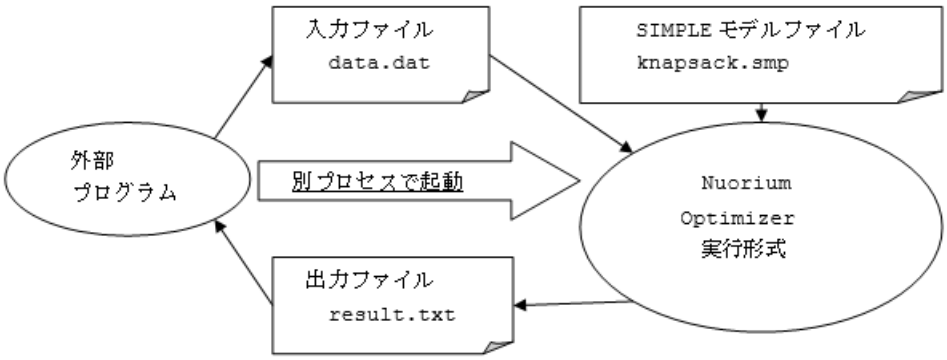
もしくは

```
using namespace SimpleMatrix;
```

の記述が必要となります．ただし，「5.2 genClass のコール」を実行された場合には記述する必要はありません．

第2章 実行形式を作成して別プロセスで起動する例

Nuorium Optimizer を外部プログラムと連結する方法として，まずは Nuorium Optimizer 部分を実行形式として作成し，外部プログラムから別プロセスで起動する例を紹介します．データの入出力はファイルを介して行います．この方法は入出力ファイル処理やプロセス起動のオーバーヘッドがありますが，Nuorium Optimizer と外部プログラムが独立しているので，問題が起こったときに原因の切り分けが容易で，開発しやすいというメリットがあります．特に上記のオーバーヘッドが問題になる場合を除いては，本方法を使用することをお勧めします．



2.1 実行形式の作成

まずは C++SIMPLE のモデルファイル `knapsack.smp` から実行形式を作成します．ここでは，

目的関数の係数： $c = (6 \quad 8 \quad 4 \quad 3 \quad 4)$
制約式の係数： $a = (4 \quad 2 \quad 3 \quad 6 \quad 7)$

という 5 変数のナップサック問題を解きます．以下にコマンドプロンプト上で実行形式を作成する例を示しますが，詳細については Nuorium Optimizer マニュアルをご参照ください．

knapsack.smp

```
//
//   ナップサック問題
//
Set S;
Element i(set = S);
IntegerVariable x(index = i, type = binary); // 整数変数
```

```
Parameter c(index = i);
Parameter a(index = i);
Parameter b;
Objective obj(type = maximize);

obj = sum(c[i] * x[i], i);    // 目的関数
sum(a[i] * x[i], i) <= b;     // 制約条件

solve(); // 求解

// 結果のファイル出力
FILE* fout = fopen("result.txt", "w");
if(fout){
  simple_fprintf(fout, "%f\n", obj);
  simple_fprintf(fout, "%d\n", x[i]);
  fclose(fout);
}else{
  fprintf(stderr, "file open error!\n");
}
```

- **実行形式の作成**

```
prompt> mknuopt.bat knapsack.smp
```

2.2 外部プログラムの例

Nuorium Optimizer 部分は別プロセスとして起動するので，外部プログラムを実装するプログラミング言語としては別プロセスを起動することができるものであれば何でも構いません．ここでは C 言語と Java を用いた外部プログラムの例を示します．

ここでは，モデルの b（制約式の上限）に相当するデータを 1 から 30 まで動かして繰り返し解くという手続きを作成します．

外部プログラム例（C 言語）

exeLoopC.c

```
//
// Numerical Optimizer 実行形式を別プロセスとして
// 実行する例（C 言語）
//
#include <stdio.h>
```

```
#include <stdlib.h>

void write(int n, double* a, double b, double* c, const char* input);
void read(int n, int* x, double* obj, const char* output);

int main(void)
{
  int i, n = 5;
  double a[] = {4, 2, 3, 6, 7};
  double b;
  double c[] = {6, 8, 4, 3, 4};
  int x[5];
  double obj[1];
  int ret;
  for(b = 1; b <= 30; b++){
    write(n, a, b, c, "data.dat"); // 入力データの作成
    ret = system("knapsack.exe data.dat"); // 別プロセスとして起動
    read(n, x, obj, "result.txt"); // 出力データの読み込み
    printf("ret = %d, b = %f, obj = %f, x =", ret, b, *obj);
    for(i = 0; i < n; i++)
      printf(" %d", x[i]);
    printf("\n");
  }
  return 0;
}

// 入力データの作成関数
void write(int n, double* a, double b, double* c, const char* input)
{
  int i;
  FILE* fp = fopen(input, "w");
  fprintf(fp, "a=\n");
  for(i = 0; i < n; i++)
    fprintf(fp, "[%d] %f\n", i + 1, a[i]);
  fprintf(fp, ";\nb = %f;\nc=\n", b);
  for(i = 0; i < n; i++)
    fprintf(fp, "[%d] %f\n", i + 1, c[i]);
  fprintf(fp, ";\n");
```

```
  fclose(fp);
}

// 出力データの読込関数
void read(int n, int* x, double* obj, const char* output)
{
  int i;
  char buf[256];
  FILE* fp = fopen(output, "r");
  fgets(buf, 256, fp);
  sscanf(buf, "%lf", obj);
  i = 0;
  while(fgets(buf, 256, fp) != NULL){
    sscanf(buf, "%d", x + i);
    i++;
  }
  fclose(fp);
}
```

外部プログラム例（Java）

exeLoopJ.java

```
//
// Numerical Optimizer 実行形式を別プロセスとして実行する例（Java）
//
import java.io.*;

class exeLoopJ{
  public static void main(String[] args){
    int n = 5;
    double[] a = {4, 2, 3, 6, 7};
    double[] c = {6, 8, 4, 3, 4};
    int[] x = new int[n];
    double[] obj = new double[1];
    for(double b = 1; b <= 30; b++){
      int ret = 1;
      try{
        write(n, a, b, c, "data.dat"); // 入力作成
        ret = execute("knapsack.exe","data.dat"); // 別プロセス起動
```

```
      read(n, x, obj, "result.txt"); // 出力読み込み
    }catch(IOException e){
      e.printStackTrace();
    }
    System.out.print("ret = "+ret+", b = "+b+", obj = "+obj[0]+", x =");
    for(int i = 0; i < n; i++)
      System.out.print(" "+x[i]);
    System.out.print("\n");
  }
}
// 入力作成メソッド
static void write(int n, double[] a, double b, double[] c, String input)
    throws IOException{
  FileWriter fw = new FileWriter(input);
  fw.write("a =\n");
  for(int i = 0; i < n; i++)
    fw.write("["+(i+1)+"] "+a[i]+"\n");
  fw.write(";\nb = "+b+";\nc =\n");
  for(int i = 0; i < n; i++)
    fw.write("["+(i+1)+"] "+c[i]+"\n");
  fw.write(";\n");
  fw.close();
}
// 別プロセス起動メソッド
static int execute(String exe, String input) throws IOException{
  ProcessBuilder pb = new ProcessBuilder(exe, input);
  Process p = pb.start();
  InputStream is = p.getInputStream();
  while(is.read() > 0);
  try{
    p.waitFor();
  }catch(java.lang.InterruptedException e){
    e.printStackTrace();
  }
  return(p.exitValue());
}
// 出力読込メソッド
static void read(int n, int[] x, double[] obj, String output) throws IOException{
```

```
    FileReader fr = new FileReader(output);
    BufferedReader br = new BufferedReader(fr);
    obj[0] = Float.valueOf(br.readLine());
    for(int i = 0; i < n; i++)
      x[i] = Integer.valueOf(br.readLine());
    br.close();
    fr.close();
  }
}
```

C 言語，Java どちらの例においても write 関数（メソッド）を定義して，Nuorium Optimizer への入力データをファイルとして出力しています．Nuorium Optimizer への入力ファイルのフォーマットについては，Nuorium Optimizer/C++SIMPLE マニュアルをご参照ください．

C 言語の例では system 関数を使って，Java の例では ProcessBuilder を使った execute メソッドを定義して Nuorium Optimizer 実行形式を別プロセスとして起動しています．この際，write 関数（メソッド）で作った入力データファイル data.dat を引数として与えています．

Nuorium Optimizer 実行形式は求解結果（x と obj）を result.txt に出力します．C 言語，Java どちらの例においても read 関数（メソッド）を定義して，Nuorium Optimizer の求解結果を読み込んでいます．

第3章 ライブラリ solveLP，solveQP

solveLP，solveQP はそれぞれ C++のライブラリです．solveLP は（混合整数）線形計画問題を対象とするのに対し，solveQP は（混合整数）二次計画問題や（混合整数ではない）二次制約付き二次計画問題を対象とします（次の表を参照してください）．

問題の種類	solveLP	solveQP
線形計画問題	○	○
混合整数線形計画問題	○	○
二次計画問題	×	○
混合整数二次計画問題	×	○
二次制約付き二次計画問題	×	○
二次制約付き混合整数二次計画問題	×	×

問題は次の形式に定式化されているものとし，変数や制約式の上下限や係数行列を C++の配列として直接引数に取ります．入力の際にモデリング言語は用いません．

- （混合整数）線形計画問題（solveLP が対応）

$$
\begin{aligned}
&\text{最小化・最大化} && \sum_j c_j \cdot x_j && j = 1, \cdots, m \\
&\text{条件} && cu_i \geq \sum_j A_{i,j} \cdot x_j \geq cl_i && \begin{matrix} i = 1, \cdots, n \\ j = 1, \cdots, m \end{matrix} \\
& && bu_j \geq x_j \geq bl_j && j = 1, \cdots, m \\
& && (x_j \in Z) && (j \in I)
\end{aligned}
$$

- （混合整数）二次計画問題・二次制約付き問題（solveQP が対応）

$$
\begin{aligned}
&\text{最小化・最大化} && \sum_j c_j \cdot x_j + \frac{1}{2} \sum_{j,k} Q_{j,k} \cdot x_j \cdot x_k && \begin{matrix} j = 1, \cdots, m \\ k = 1, \cdots, m \end{matrix} \\
&\text{条件} && cu_i \geq \sum_j A_{i,j} \cdot x_j + \frac{1}{2} \sum_{j,k} Q^i_{j,k} \cdot x_j \cdot x_k \geq cl_i && \begin{matrix} i = 1, \cdots, n \\ j = 1, \cdots, m \\ k = 1, \cdots, m \end{matrix} \\
& && bu_j \geq x_j \geq bl_j && j = 1, \cdots, m \\
& && (x_j \in Z) && (j \in I)
\end{aligned}
$$

3.1 呼び出し形式

以下がライブラリ関数 solveLP，solveQP の呼び出し形式です．"=0"は C++の記法のデフォルト引数（省略が可能な引数）を示しています．solveLP は整数変数が存在しないとき，ivtype 以降を，solveQP は制約式の二次の部分が存在しないとき，nQCelem 以降をそれぞれ一括して省略することができます．呼び出し形式はヘッダーファイル

nuoIf.h

に含まれています．このヘッダファイルは Nuorium Optimizer ライブラリの include ファイルの保管場所

（Nuorium Optimizer のインストール場所）\userapp\include

にあります．なお，nuoIf.h には NAMESPACE_NUOPT integer という型が書かれていますが，int 型だと考えていただいて問題ありません．

- solveLP

solveLP の呼び出し形式

```
nuoptResult*
solveLP
(
  nuoptParam* options
  ,int n, int m
  ,int minimize
  ,double* x0
  ,double* bL, double* bU, int* ibL, int* ibU
  ,double* cL, double* cU, int* icL, int* icU
  ,double* objL
  ,int nAelem, int* irowA, int* jcolA, double* a
// 以下は整数変数が存在しない場合一括して省略可能
  ,int* ivtype = 0
  ,int* pri  = 0
  ,int* dir = 0
  ,int* until = 0
  ,double* upc = 0
  ,double* dpc = 0
);
```

- solveQP

solveQP の呼び出し形式

```
nuoptResult*
solveQP
(
```

```
  nuoptParam* options
  ,int n, int m
  ,int minimize
  ,double* x0
  ,double* bL, double* bU, int* ibL, int* ibU
  ,double* cL, double* cU, int* icL, int* icU
  ,double* objL
  ,int nAelem, int* irowA, int* jcolA, double* a
  ,int nQelem, int* irowQ, int* jcolQ, double* q
// 以下は制約式の二次の部分が存在しない場合一括して省略可能
  ,int nQCelem = 0, int* ifunQC = 0, int* irowQC = 0, int* jcolQC = 0
  ,double* qc = 0
// 以下は整数変数が存在しない場合一括して省略可能
  ,int* ivtype = 0
  ,int* pri = 0
  ,int* dir = 0
  ,int* until = 0
  ,double* upc = 0
  ,double* dpc = 0
);
```

solveQP は solveLP の機能を含んでおり，solveQP で二次の項の指定をすべて消去すると，solveLP と同一の機能となります．

3.2 ルーチン仕様

solveLP，solveQP の引数の意味は共通しています．以降で各引数の解説を行います．

3.2.1 問題全体にかかわるもの（solveLP，solveQP 共通）

nuoptParam*	options	Nuorium Optimizer の求解オプション（0：デフォルト）
int	n	変数の総数
int	m	制約式の総数（等式，不等式含む）
int	minimize	最小化かどうかのフラグ（非零：最小化，0：最大化）
double*	x0	変数の初期値（長さ：n）
double*	bL	変数の下限ベクトル（長さ：n）
double*	bU	変数の上限ベクトル（長さ：n）
int*	ibL	変数の下限の有無（長さ：n，非零：あり，0：なし）
int*	ibU	変数の上限の有無（長さ：n，非零：あり，0：なし）

double*	cL	制約式の下限ベクトル（長さ：m）
double*	cU	制約式の上限ベクトル（長さ：m）
int*	icL	制約式の下限の有無（長さ：m，非零：あり，0：なし）
int*	icU	制約式の上限の有無（長さ：m，非零：あり，0：なし）

ここで紹介する引数は solveLP，solveQP に共通で，同じ意味を持ちます．最初の引数は Nuorium Optimizer の求解オプションを与えるものですが，0 を渡すことができます．その場合には求解オプションとしてデフォルトの設定を用いるという意味になります．通常はデフォルトで問題ありませんが，求解オプションの指定を行う場合には，

```
nuoptParam myparam;
myparam.method = "asqp"; // 解法の指定
solveQP(&myparam,...);
```

のようにして，myparam のメンバ（個別の求解オプションに相当する）に値を設定して，solveQP にアドレスを渡します．詳細は「6.8Nuorium Optimizer 求解オプション」を参照ください．

変数の上下限が存在しない場合には，対応する ibL または ibU のコンポーネントを 0 にします．例えば n = 3 の場合で

$$0 \le x_1$$

$$x_2 \le 1$$

$$2 \le x_3 \le 3$$

という上下限を表現する場合には次のように設定します．

配列の添字	bL	bU	ibL	ibU
0	0	任意	0 以外	0
1	任意	1	0	0 以外
2	2	3	0 以外	0 以外

C++では配列の添字は 0 はじまりですので，1 番目の変数は添字 0 の場所に対応します．配列の「任意」と書かれた場所は無視されます．「0 以外」という場所は 0 以外の任意の値です．制約式についても全く同様です．

変数の固定，および等式制約は上限と下限を一致させることによって表現します．ibL，ibU，icL，icU の場所に NULL ポインタを渡すことが許されています．その場合，すべて 0 が格納された配列を渡すのと同じ意味になります．

初期値に対応する x0 に NULL ポインタを渡すことができます．その場合には初期値はデフォルトのものを用いるという意味になります．

3.2.2 線形部分にかかわるもの（solveLP，solveQP 共通）

double*	objL	目的関数の線形部分（長さ：n）
int	nAelem	制約式の係数行列の非零要素数
int*	irowA	非零要素の行番号（長さ：nAelem）
int*	jcolA	非零要素の列番号（長さ：nAelem）
double*	a	非零要素の値（長さ：nAelem）

ここで扱う引数も solveLP，solveQP で同じ意味を持ちます．objL は目的関数の線形部分

$$\sum_j c_j \cdot x_j$$

の c_j に対応します．

制約式の係数行列は非零要素のみ与えます．同じ非零要素を二つ以上与えた場合には，それらの和が取られます．非零要素の場所は 1 始まりの番号で与えます．例えば nAelem=4 のとき，

配列の添字	irowA	jcolA	a
0	1	2	2.3
1	**1**	**3**	1.8
2	2	1	0.5
3	**1**	**3**	0.2

のように与えると

$$A = \begin{bmatrix} 0 & 2.3 & 2.0 \\ 0.5 & 0 & 0 \end{bmatrix}$$

を与えたことになります（A_{13} については二つ与えられているので和（1.8+0.2=2.0）が取られます）．非零要素の格納順番は任意です．

3.2.3 混合整数計画問題にかかわるもの（solveLP，solveQP 共通：一括省略可）

int*	ivtype	変数の種別（長さ：n，0:連続，1：整数，2：バイナリ）
int*	pri	NULL ポインタを設定します
int*	dir	NULL ポインタを設定します
int*	until	NULL ポインタを設定します
double*	upc	NULL ポインタを設定します
double*	dpc	NULL ポインタを設定します

引数 ivtype は変数の種別を与えるベクトルで，次の意味を持ちます．

0：連続変数

1：一般の整数変数
2：バイナリ変数（0 または 1）

ivtype 自体を NULL ポインタにするとすべての変数が連続変数であるという指定と同じ意味となります．until は V15 より廃止されましたので，NULL ポインタにします．また，pri，dir，upc，dpc は V20 より廃止されましたので，NULL ポインタにします．

整数変数が存在しない場合には，ivtype 以降の引数をすべて省略することができます（一括省略）．

3.2.4　目的関数の二次の部分にかかわるもの（solveQP のみ）

int	nQelem	目的関数のヘッセ行列の非零要素数
int*	irowQ	目的関数のヘッセ行列の行番号（長さ：nQelem）
int*	jcolQ	目的関数のヘッセ行列の列番号（長さ：nQelem）
double*	q	目的関数のヘッセ行列の非零要素の値（長さ：nQelem）

ここに挙げた引数で目的関数の二次の部分の係数行列（ヘッセ行列）

$$\frac{1}{2}\sum_{j,k} Q_{j,k}\cdot x_j \cdot x_k$$

の $Q_{i,j}$ を与えます．制約式の係数行列と同様に非零要素のみを与えます．指定の仕方は制約式の係数行列と同様です．非零要素の場所は 1 始まりの番号で与えます．しかし，目的関数のヘッセ行列は対称行列であることを前提としていますので，下三角部分の非零要素を与えると同時に上三角部分も与えたことになる（その逆も同じ）という点に注意してください．

例えば

$$Q=\begin{bmatrix}1 & 5\\ 5 & 7\end{bmatrix}$$

というヘッセ行列を定義するためには nQelem = 3 として

配列の添字	irowQ	jcolQ	q
0	1	1	1
1	2	1	5
2	2	2	7

→ 同時に上三角側も与えたことになる．

のように与えます．

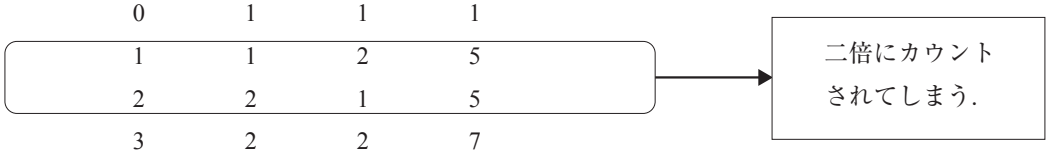
あるいは以下のようにしても同じ意味です．

配列の添字	irowQ	jcolQ	q
0	1	1	1
1	1	2	5
2	2	2	7

→ 同時に下三角側も与えたことになる．

同じ非零要素を二つ与えるとその和が取られます．従って nQelem=4 として

配列の添字	irowQ	jcolQ	q
0	1	1	1
1	1	2	5
2	2	1	5
3	2	2	7



とすると，上の下三角部分の非零要素を与えると同時に上三角部分も与えたことになるという原則（その逆も同じ）によって

$$Q = \begin{bmatrix} 1 & 10 \\ 10 & 7 \end{bmatrix}$$

を定義したことになりますのでご注意ください．また，非零要素の格納順番は任意です．

nQelem=0 とすると，目的関数に二次の部分が存在しないものと解釈されます．その場合には irowQ, jcolQ, q はすべて NULL ポインタとすることができます．

3.2.5 制約式の二次の部分にかかわるもの（solveQP のみ，一括省略可）

int	nQCelem	制約式のヘッセ行列の非零要素数
int*	ifunQC	制約式のヘッセ行列の非零要素が属する制約式番号（長さ：nQCelem）
int*	irowQC	制約式のヘッセ行列の行番号（長さ：nQCelem）
int*	jcolQC	制約式のヘッセ行列の列番号（長さ：nQCelem）
double*	qc	制約式のヘッセ行列の非零要素の値（長さ：nQCelem）

これらの引数で制約の二次部分

$$\frac{1}{2}\sum_{j,k} Q^i_{j,k} \cdot x_j \cdot x_k \quad (i \text{ は制約式の番号})$$

の係数行列（ヘッセ行列）の群

$$Q^i_{j,k}$$

を与えます．非零要素と，その非零要素が属する制約式の番号のみを与えます．

例えば，制約式 1，2 の二次の部分がそれぞれ

$$Q^1 = \begin{bmatrix} 4 & 3 \\ 3 & 6 \end{bmatrix},\ Q^2 = \begin{bmatrix} 8 & 2 \\ 2 & 3 \end{bmatrix}$$

である場合には nQCelem = 6 として

配列の添字	ifunQC	irowQC	jcolQC	qc
0	1	1	1	4
1	1	1	2	3
2	1	2	2	6

3	2	1	1	8
4	2	1	2	2
5	2	2	2	3

と設定します．制約式は 1 始まりの番号で指定します．また，行列の非零要素の場所も 1 始まりの番号で指定します．

制約式のヘッセ行列も対称であることを前提としているので，目的関数のヘッセ行列と同じく下三角部分の非零要素を与えると同時に上三角部分も与えたことになる（その逆も同じ）という原則が適用されます．また，目的関数の二次部分のヘッセ行列と同じく，同一の非零要素が二つ以上与えられると和が取られます．

非零要素の格納順番は任意です．

nQCelem=0 とすると，制約式に二次の部分が存在しないものと解釈されます．その場合には ifunQC，irowQC，jcolQC，qc はすべて NULL ポインタとすることができます．

制約式の二次の部分が存在しない場合には，nQCelem から以降の引数をすべて省略することができます．

3.2.6　出力とエラーメッセージ

solveLP，solveQP の戻り値として，nuoptResult というオブジェクトのポインタが返ります．このオブジェクトのメンバに実行結果についての情報が格納されています．利用可能なメンバは以下の通りです．このメンバの利用の詳細は次項で解説するサンプルコード useSolveQP.cpp をご覧ください．このオブジェクトは利用するコード内で開放（delete）する必要があります．

`int errorCode();`	エラーコード
`char* errorMessage();`	エラーメッセージ
`double optValue();`	最適値
`double VarVal(int i);`	i 番目の変数の値
`double VarDual(int i);`	i 番目の変数の dual 値
`double FuncVal(int i);`	i 番目の制約の値
`double FuncDual(int i);`	i 番目の制約の dual 値

上記で i は変数，制約式のインデクスですが，0 始まりであることにご注意ください（最初の変数/制約式が 0 番目）．

エラーコードとエラーメッセージは Nuorium Optimizer 本体のものと同じです．「Nuorium Optimizer マニュアル」の付録 A をご参照ください．また，solveLP，solveQP 特有のエラーとして，引数の矛盾や範囲オーバーがありますが，それは 151 から 165 のコード番号に対応します[3]．意味は以下のとおりです．

コード	意味

[3] V18 からコード番号が変更となりました．

151 n が 0 以下
152 m が負
153 変数の下限（bL）が上限（bU）よりも大きい
154 制約式の下限（cL）が上限（cU）よりも大きい
155 nAelem が負
156 irowA のコンポーネントの範囲が違反
157 jcolA のコンポーネントの範囲が違反
158 nQelem が負
159 irowQ のコンポーネントの範囲が違反
160 jcolQ のコンポーネントの範囲が違反
161 nQCelem が負
162 ifunQC のコンポーネントの範囲が違反
163 irowQC のコンポーネントの範囲が違反
164 jcolQC のコンポーネントの範囲が違反
165 ivtype の範囲が違反

エラーメッセージは

```
(NUOPT 156) irowA[0] = 9 should be in [1,3]
```

のように，実際のデータに即したより詳しい情報を含んでいます．

3.3 実行サンプル

開発環境に同梱されている useSolveQP.cpp は solveLP，solveQP を利用するサンプルです．線形計画問題なら，solveLP を，二次計画問題なら，solveQP を呼びます．

- **注意：**

以降では，サンプルを用いた実行について書かれています．読み進む前に，まず，お使いのコンパイラを確認してください．サンプルは

%NUOPT%\samples\app

（%NUOPT%は Nuorium Optimizer のインストール場所，たとえば C:\Program Files\Mathematical Systems Inc\nuopt）
にある

- app_VS2015.zip（VS2015 用）
- app_VS2017.zip（VS2017 用）
- app_VS2019.zip（VS2019 用）
- app_VS2022.zip（VS2022 用）

のうち，お使いのコンパイラに整合するものを展開してご利用ください．なお，マシンの設定

によってはzipファイルのある場所に書き込み権限が無いため展開できないことがございます．この場合は書き込み権限があるフォルダに展開してください．また，コンパイラとプロジェクトが整合していない場合，リンクエラーやコンパイルエラーが生じてしまいます．

ソリューション nuoptvcapp のプロジェクト

useSolveQP

がこのインタフェースを用いて LP，QP を解くプログラム例です．

useSolveQP.cpp は solveLP，solveQP を利用するサンプルで，1 つの main 関数のみから成ります．ロードモジュールの引数名で与えられたファイルから問題のデータを読み込み，それを solveLP, solveQP に渡します．二次の項がなく，線形計画問題ならば solveLP を，あれば solveQP を呼びます．以下はそのプログラムの抜粋です．

useSolveQP.cpp(.cc)

```
//
//  solveQP の利用例
//
#include "nuoIf.h" // 必須.
int main(int argc, char** argv)
{
  int n;
  int m;
  ifstream inputFile(argv[1]); // ロードモジュールの引数をファイル名とする
  inputFile >> n >> m ;

     （中略）

  nuoptResult* qpres = 0;
  nuoptParam myParam;
  if (nQelem || nQCelem) { // 2 次の係数があるなら solveQP をコール
    qpres = solveQP(&myParam
    ,n, m
    ,minimize
    ,x0
    ,bL, bU, ibL, ibU
```

```
    ,cL, cU, icL, icU
    ,objL
    ,nAelem, irowA, jcolA, a
    ,nQelem, irowQ, jcolQ, q
    ,nQCelem, ifunQC, irowQC, jcolQC, qc
    ,ivtype, pri, dir, until, upc, dpc
  );
  } else {// 2 次の係数がないのなら solveLP をコール
    qpres = solveLP(&myParam
    ,n, m
    ,minimize
    ,x0
    ,bL, bU, ibL, ibU
    ,cL, cU, icL, icU
    ,objL
    ,nAelem, irowA, jcolA, a
    ,ivtype, pri, dir, until, upc, dpc
  );
}

  if (qpres->errorCode()) {
    printf("error in solveQP code = %d, message = %s\n"
    ,qpres->errorCode(), qpres->errorMessage());
  fflush(stdout);
  exit(1);
  } else {
    printf("optimalValue = %17.10e\n", qpres->optValue());
    printf("X:\n");
    for (i = 0 ; i < n ; ++i) {
    printf("[%3d] %10.3e ", i + 1, qpres->VarVal(i));
    if ((i + 1) % 4 == 0) {
      printf("\n");
    }
  }
  printf("\n");
  printf("F:\n");
  for (i = 0; i < m; ++i) {
    printf("[%3d] %10.3e ", i + 1, qpres->FuncVal(i));
```

```
    if ((i + 1) % 4 == 0) {
      printf("\n");
    }
  }
    printf("\n");
  }

  delete qpres;
  delete [] bL;
  delete [] bU;
  delete [] ibL;
  delete [] ibU;

  （後略）
}
```

次のようなナップサック問題を考えます．

変数	$x_i \in \{0,1\} \quad (i \in S)$
目的関数（最大化）	$\sum_{i \in S} c_i x_i,$
制約条件	$\sum_{i \in S} a_i x_i \leq b$
目的関数の係数：	$c = (42 \quad 12 \quad 45 \quad 5 \quad 2 \quad 61 \quad 89 \quad 32 \quad 47 \quad 18)$
制約式の係数：	$a = (39 \quad 13 \quad 68 \quad 15 \quad 10 \quad 20 \quad 31 \quad 15 \quad 41 \quad 16)$
制約式の右辺：	$b = 121$

これを solveLP で解くには，一般の線形計画問題：

最小化・最大化	$\sum_j c_j \cdot x_j$	$j = 1, \cdots, m$
条件	$cu_i \geq \sum_j A_{i,j} \cdot x_j \geq cl_i$	$i = 1, \cdots, n$ $j = 1, \cdots, m$
	$bu_j \geq x_j \geq bl_j$	$j = 1, \cdots, m$
	$(x_j \in Z$	$j \in I)$

の形にこの問題を表現する，すなわち

$$c = (42 \quad 12 \quad 45 \quad 5 \quad 2 \quad 61 \quad 89 \quad 32 \quad 47 \quad 18)$$
$$Q = 0$$
$$cu = (121)$$
$$cl = (-\infty)$$
$$A = (39 \quad 13 \quad 68 \quad 15 \quad 10 \quad 20 \quad 31 \quad 15 \quad 41 \quad 16)$$
$$x_j \in \{0, 1\} \text{（バイナリ変数）}$$

とすればよいことがわかります．useSolveQP.cpp の入力ファイルとしてこのデータを表現したのが(Nuorium Optimizer のインストール場所)\samples\app にある zip ファイルを解凍した中にあるファイル

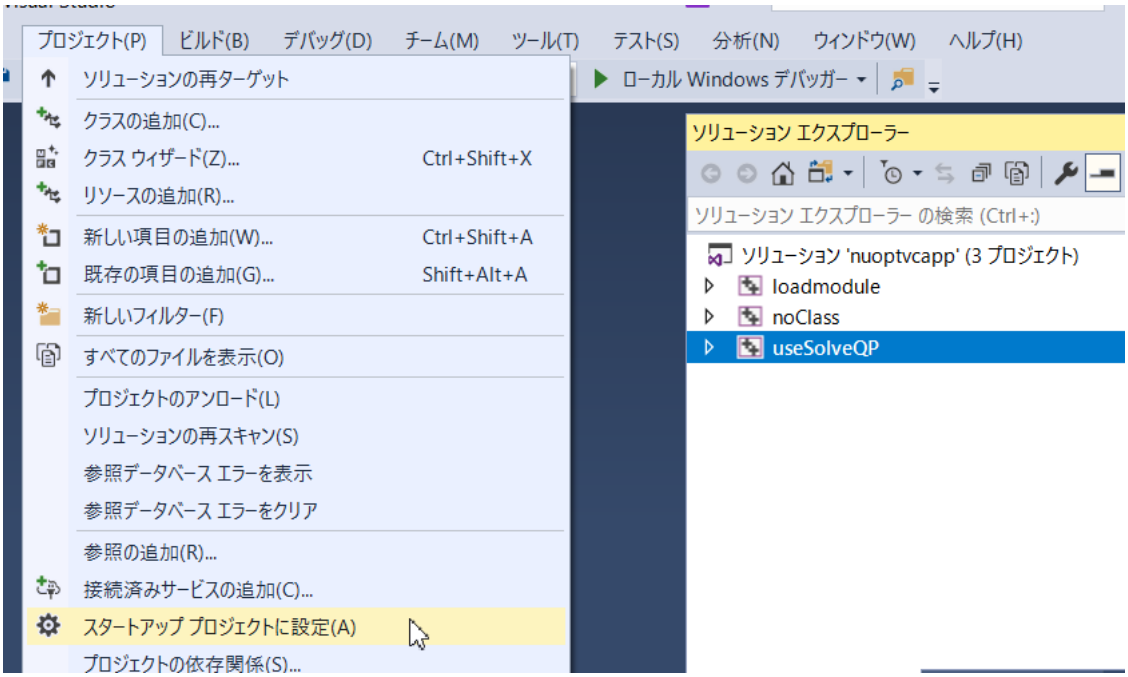
useSolveQP\knapsack.txt

です．

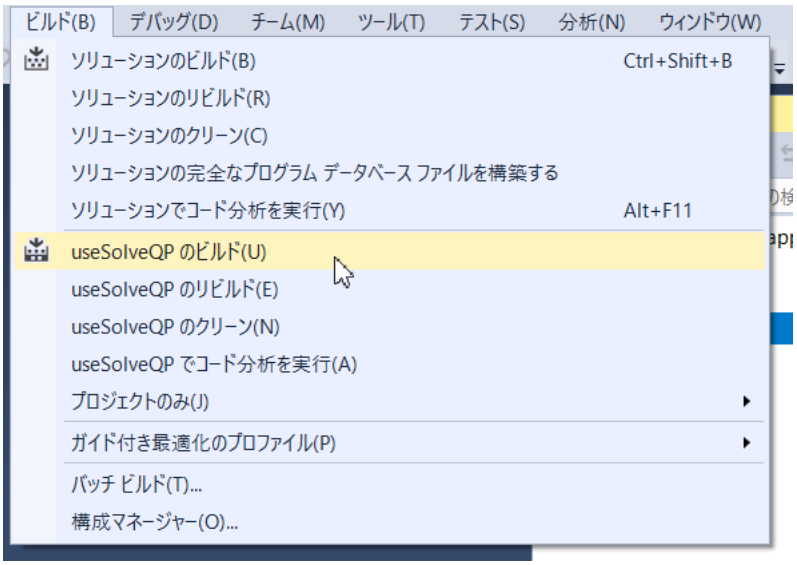
3.4 実行例

さて，サンプルコード useSolveQP の実行モジュールを作成して実行してみましょう．

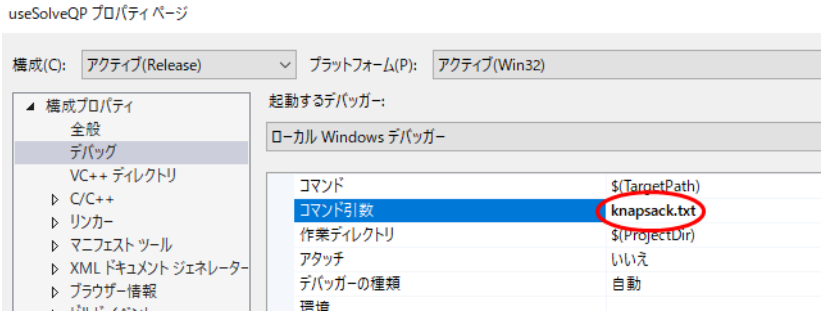
プロジェクト「useSolveQP」を選択後，VC++のメニューの「プロジェクト」から「スタートアッププロジェクトに設定」を選び，プロジェクトを選択します．なお useSolveQP の構成は「Release」に設定してあります．



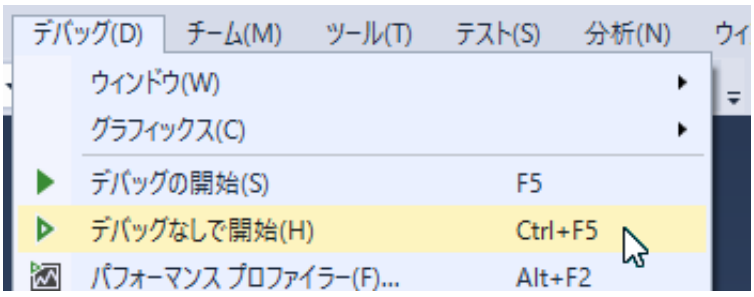
続いて「ビルド」メニューから「useSolveQP のビルド」:



を選択すると useSolveQP.cpp がコンパイル，Nuorium Optimizer のライブラリとリンクされて，実行モジュールの useSolveQP.exe が作成されます．引数として knapsack.txt を与えるように「プロジェクト」→「プロパティ」→「構成プロパティ」→「デバッグ」の項目で，「コマンド引数」に「knapsack.txt」と入力します．



つづいて「デバッグ」メニューから



と実行すると，実行経過表示ウィンドウが現れ，その中に以下のような出力が得られます．solveLP がコールされて出力が成されます．

```
[About Nuorium Optimizer]
Nuorium Optimizer x.x.x (NLP/LP/IP/SDP module)
         <with META-HEURISTICS engine "wcsp"/"rcpsp">
```

```
         <with Netlib BLAS>
, Copyright (C) 1991 NTT DATA Mathematical Systems Inc.

[Problem and Algorithm]
PROBLEM_NAME                                          anon.LP
NUMBER_OF_VARIABLES                                        10
(#INTEGER/DISCRETE)                                        10
NUMBER_OF_FUNCTIONS                                         2
PROBLEM_TYPE                                      MAXIMIZATION
METHOD                                                SIMPLEX

[Progress]
<preprocess begin>.........<preprocess end>
<iteration begin>
Coefficient Statistics (after scaling)
  Coefficient range         [min,max] : [4.47e-01,3.04e+00]
  RHS and bounds            [min,max] : [1.00e+00,5.41e+00]
  Objective                 [min,max] : [5.67e-02,2.52e+00]

#sol         upper         lower   gap(%)  time(s)  list  mem(MiB)
#1          -1e+50            -0  100.000      0.0     0       10  sol: init

<<wcsp tabu search begin>>
  number of column singleton : 0
  number of column selection : 0
  Modify coefficients

<preprocess begin>..<preprocess end>
preprocessing time: 0.002(s)
<iteration begin>
--- TryCount = 1 ---
# random seed = 1
(hard/soft) penalty= 19/194, time= 0.00(s)
<greedyupdate begin>.........<greedyupdate end>
greedyupdate time= 0(s)
--- End Phase-I iteration ---
(hard/soft) penalty= 0/273, time= 0.00(s), iteration= 2
(hard/soft) penalty= 0/217, time= 0.00(s), iteration= 53
```

```
(hard/soft) penalty= 0/173, time= 0.00(s), iteration= 60
(hard/soft) penalty= 0/126, time= 0.00(s), iteration= 61
(hard/soft) penalty= 0/124, time= 0.01(s), iteration= 70
(hard/soft) penalty= 0/112, time= 0.01(s), iteration= 71
(hard/soft) penalty= 0/111, time= 0.01(s), iteration= 590
# (hard/soft) penalty= 0/111
# cpu time = 0.01/0.01(s)
# iteration = 590/1000
<iteration end>
# (hard/soft) = 0/111
# iteration = 1000
# time =  0.01 (s), succ = 1
<<wcsp tabu search end>>

#2          -1e+50           242  100.000      0.0     0       10  sol: wcsp

<iteration end>

[Result]
STATUS                                              OPTIMAL
VALUE_OF_OBJECTIVE                                      242
SIMPLEX_PIVOT_COUNT                                       0
PARTIAL_PROBLEM_COUNT                                     1
ELAPSED_TIME(sec.)                                     0.03
SOLUTION_FILE                                     solver.sol
optimalValue =  2.4200000000e+02
X:
[  1]  1.000e+00 [  2]  0.000e+00 [  3]  0.000e+00 [  4]  0.000e+00
[  5]  0.000e+00 [  6]  1.000e+00 [  7]  1.000e+00 [  8]  1.000e+00
[  9]  0.000e+00 [ 10]  1.000e+00
F:
[  1]  1.210e+02

C:\app\x64\Release\useSolveQP.exe (プロセス 17024) は、コード 0 を伴って終了しました。
このウィンドウを閉じるには、任意のキーを押してください . . .
```

上に挙げた内容は Nuorium Optimizer からの求解に関する出力です．問題の変数の数（NUMBER_OF_VARIABLES）や目的関数の値（VALUE_OF_OBJECTIVE）を知ることができます．Nuorium Optimizer の標準出力の内容については「Nuorium Optimizer マニュアル」（別冊）をご覧ください．

Nuorium Optimizer の出力を抑制するには，

```
nuoptParam myParam; // 宣言
```

として，求解オプションを宣言したのち，

```
myParam.outputMode = "silent";
```

とします．また，デフォルトで出力される Nuorium Optimizer の求解レポートである解ファイル solver.sol の出力を抑制するには

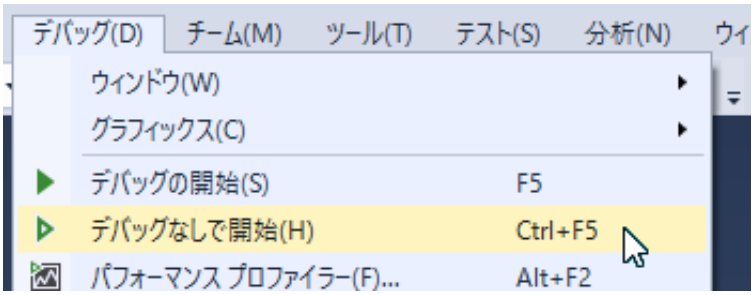
```
myParam.outfilename = "_NULL_";
```

と設定し，solveQP() の最初の引数として渡します．useSolveQP.cpp の以下のコメントを取ると，その設定になり，Nuorium Optimizer からの出力が抑制されて，useSolveQP が出力する表示のみになります．

出力を抑制する例

```
        ..
        nuoptParam myParam;
    // Numerical Optimizer の出力と解ファイルの出力を抑制する．
    myParam.outputMode = "silent";
    myParam.outfilename = "_NULL_";
if ( nQelem || nQCelem ) { // 2 次の係数があるのなら solveQP
     qpres = solveQP(&myParam
             ,n, m
     ..
```

再びビルドメニューから



とすると，コンパイルが自動的に実行されます．

knapsack.txt と同じ場所に別の useSolveQP.cpp の入力ファイル

useSolveQP\qp312.txt

があります．このデータは二次計画問題で

$$
\begin{array}{lll}
\text{最小化・最大化} & \displaystyle\sum_j c_j \cdot x_j + \frac{1}{2}\sum_{(j,k)} Q_{j,k} \cdot x_j \cdot x_k & j = 1, \cdots, m \\
 & & k = 1, \cdots, m \\
 & & i = 1, \cdots, n \\
\text{条件} & cu_i \geq \displaystyle\sum_j A_{i,j} \cdot x_j + \frac{1}{2}\sum_{j,k} Q^i_{j,k} \cdot x_j \cdot x_k \geq cl_i & j = 1, \cdots, m \\
 & & k = 1, \cdots, m \\
 & bu_j \geq x_j \geq bl_j & j = 1, \cdots, m \\
 & (x_j \in Z & j \in I)
\end{array}
$$

という定式で，次のように設定したものに対応します．2 変数，3 制約の問題で，制約式には二次の項はありません．

$$
\begin{aligned}
c &= (-3 \quad 1) \\
Q &= \begin{pmatrix} 11 & 0 \\ 0 & 22 \end{pmatrix},\ Q^i = 0 \\
cu &= (1000 \quad 1000 \quad 1000) \\
cl &= (-1 \quad -2 \quad 2) \\
A &= \begin{pmatrix} -1 & 0.1 \\ -0.2 & -1 \\ 2 & 1 \end{pmatrix} \\
bu &= (1 \quad 2) \\
bl &= (0 \quad 0)
\end{aligned}
$$

VC++の GUI で「プロジェクト」→「設定」→「デバッグ」タブで，「プログラムの引数」を

```
qp312.txt
```

として再び実行してみてください．

今度は `useSolveQP` が `solveQP` をコールして，次のような解が出力されます．

```
optimalValue = 2.3181818248e+000
X:
[  1] 9.394e-001 [  2] 1.212e-001
F:
[  1] -9.273e-001 [  2] -3.091e-001 [  3] 2.000e+000
```

次のデータ：

(`Nuorium Optimizer` のインストール場所)`\samples\app` にある zip ファイルを解凍した中にある `useSolveQP\qp312I.txt`

はこの二次計画問題の変数を連続変数ではなく，整数変数という指定をつけたもので，二次の整数計画問題となります．これを入力として実行すると

```
optimalValue = 2.5000000000e+000
X:
[  1] 1.000e+000 [  2] 0.000e+000
F:
[  1] -1.000e+000 [  2] -2.000e-001 [  3] 2.000e+000
```

のような解が出力されます.

さらに，次のデータ：

(Nuorium Optimizer のインストール場所)\samples\app にある zip ファイルを解凍した中にある useSolveQP\hs23.txt

は制約条件の中に二次式が含まれる問題となります．なお，このデータは [1] の No.23 を記述したものであり，次のような設定に対応します.

$$c = (0 \quad 0)$$

$$Q = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix},$$

$$Q^1 = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix},\ Q^2 = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix},\ Q^3 = \begin{pmatrix} 18 & 0 \\ 0 & 2 \end{pmatrix},\ Q^4 = \begin{pmatrix} 2 & 0 \\ 0 & 0 \end{pmatrix},\ Q^5 = \begin{pmatrix} 0 & 0 \\ 0 & 2 \end{pmatrix}$$

$$cu = (\infty \quad \infty \quad \infty \quad \infty \quad \infty)$$

$$cl = (1 \quad 1 \quad 9 \quad 0 \quad 0)$$

$$A = \begin{pmatrix} 1 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & -1 \\ -1 & 0 \end{pmatrix}$$

$$bu = (50 \quad 50)$$

$$bl = (-50 \quad -50)$$

これを入力として実行すると

```
optimalValue = 2.0000000009e+000
X:
[  1] 1.000e+000 [  2] 1.000e+000
F:
[  1] 2.000e+000 [  2] 2.000e+000 [  3] 1.000e+001 [  4] 2.133e-010
[  5] 2.133e-010
```

のような解が出力されます.

第4章 C++SIMPLEモデルとドライバを分離しない例

本章で紹介するのはC++SIMPLEによって記述したモデルをプログラムの一部として呼び出す方法です．この方法は次章のクラスを生成する方法と比べて簡便ですがNuorium Optimizerのライブラリが，リンクされているプログラム全体で同時に一つの問題のみしか定義できないという制限があります．

- **注意：**

以降では，サンプルを用いた実行について書かれています．読み進む前に，まず，お使いのコンパイラを確認してください．サンプルは

`%NUOPT%\samples\app`

（`%NUOPT%`はNuorium Optimizerのインストール場所，たとえば`C:\Program Files\Mathematical Systems Inc\nuopt`）
にある

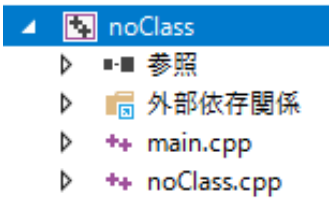
- `app_VS2015.zip`（VS2015用）
- `app_VS2017.zip`（VS2017用）
- `app_VS2019.zip`（VS2019用）
- `app_VS2022.zip`（VS2022用）

のうち，お使いのコンパイラに整合するものを展開してご利用ください．なお，マシンの設定によってはzipファイルのある場所に書き込み権限が無いため展開できないことがございます．この場合は書き込み権限があるフォルダに展開してください．また，コンパイラとプロジェクトが整合していない場合，リンクエラーやコンパイルエラーが生じてしまいます．

ソリューション`nuoptvcapp`のプロジェクト

`noClass`

が本方法に対応するプログラム例です．



4.1 モデル兼ドライバの記述

このプロジェクトでは`noClass.cpp`というソースファイルにモデルとモデルを利用する手続きが両

方書き込まれています．noClass.cpp では knapsackSolve という名前の手続きを定義しています．この手続きは入力として，ナップサック問題のモデルの

```
Parameter a(index = i);
Parameter b;
Parameter c(index = i);
```

に相当するデータ（aarg，barg，carg）を与えると

```
Variable x(index = i);
```

に相当する変数値（xarg）と目的関数（farg）を返します．

noClass.cpp

```
#include "simple.h"
//
//   ナップサック問題（モデルの記述と実行を分離しない）
//
int knapsackSolve(int narg, double* aarg, double barg, double* carg
       ,double* xarg, double* farg)
{
  //
  // システムの初期化.
  //
  SimpleInitialize();
  {
    Set S;
    Element i(set = S);

    IntegerVariable x(name = "決定ベクトル", index = i, type = binary); // 整数変数
    Parameter c(name = "価値", index = i);
    Parameter a(name = "重量", index = i);
    Parameter b(name = "許容重量");
    Objective obj(name = "総価値", type = maximize);

    // 引数からデータを設定
    a.readD(narg, aarg);
    b = barg;
    c.readD(narg, carg);

    sum(a[i] * x[i], i) <= b;     // 制約条件
```

```
  obj = sum(c[i] * x[i], i);    // 目的関数

  // Numerical Optimizer からの最適化結果ファイル.sol の
  // 名前の設定
  //    options.outfilename = "nuoptout";
  // .sol のファイル出力を抑制する場合には次のように書く
  options.outfilename = "_NULL_";
  // 出力を抑制する
  options.outputMode = "silent";

  // 最適化の実行
  solve();

  // x の内容を C++の配列にダンプ

  int lenx;
  int* indx;
  double* valx;
  x[i].val.dump(lenx, indx, valx);

  if (lenx != narg) {
    return 99; // x[i] の実際の長さが narg と異なる（データに矛盾あり）
  }

  // 引数に設定
  int it;
  for (it = 0; it < lenx; ++it) {
    xarg[indx[it] - 1] = valx[it];
    // indx[it] は 1,2, ... narg なので，1 を引く
  }

  // 目的関数値の設定
  *farg = result.optValue;

  // indx,valx は dump 内部で独自に allocate されるので free しておく.
  delete [] indx;
  delete [] valx;
}
```

```
  // Numerical Optimizer のエラーコードを返す.
  return result.errorCode;
}
```

このインタフェースを用いる場合, モデルはプログラム全体で 1 つだけ定義することが可能です．変数, 制約式は定義した場所によらずにそのモデルに追加されてゆくことになります．SimpleInitialize() のコールより後と, SimpleClearBuffer() のコールより前は中括弧{ }でくくる必要があります．SIMPLE のオブジェクト（Set や Element）の宣言は，この中括弧{ }の中で行います．SimpleInitialize() をコールせずに SIMPLE のオブジェクトを宣言して利用する，或いはこの中括弧{}の外で SIMPLE のオブジェクトを宣言すると実行時エラーとなります.

4.2 モデル兼ドライバのコール

次は knapsackSolve を C++で呼び出しているメインルーチンを記述します．ここでは，第 2 章と同様にナップサック問題

$$\text{目的関数の係数：}\quad c = (6 \quad 8 \quad 4 \quad 3 \quad 4)$$
$$\text{制約式の係数：}\quad a = (4 \quad 2 \quad 3 \quad 6 \quad 7)$$

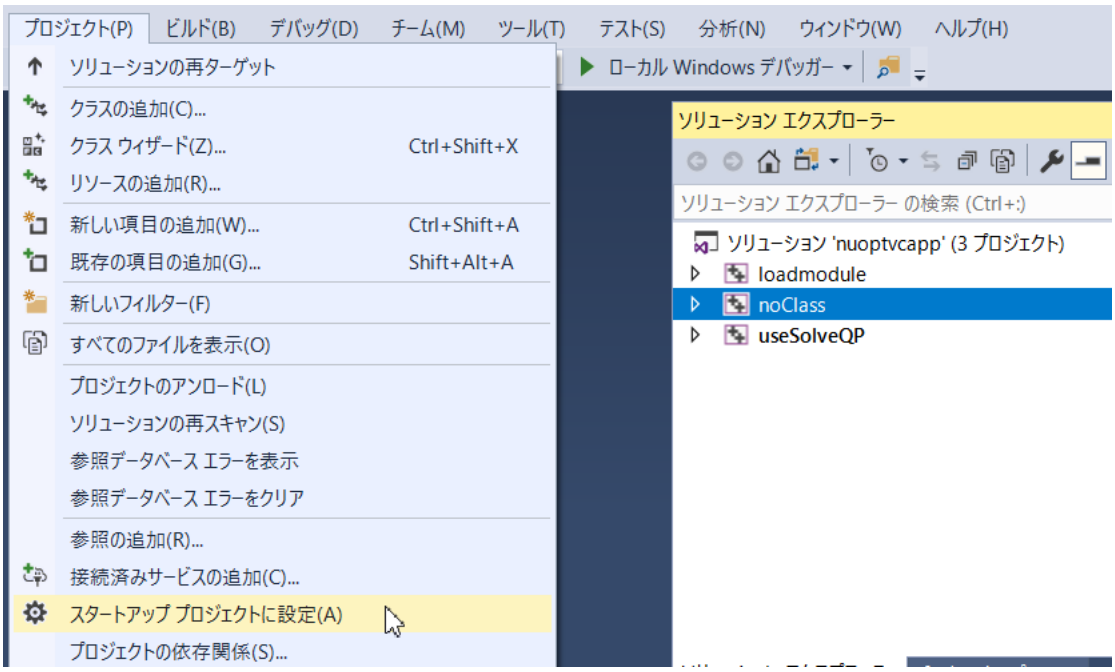
の容量制約の上限値を 1 から 30 の間で動かすような問題を考えます．このルーチンもプロジェクト noClass に追加されています.

main.cpp

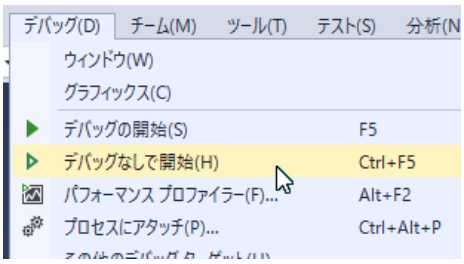
```
#include <stdio.h>

// main() の中でコールする最適化手続き（最適化ライブラリ）
int knapsackSolve(int narg, double* aarg, double barg, double* carg
        ,double* xarg, double* farg);
void main()
{
// 問題のデータ
  int narg = 5;
  double aarg[5] = {4, 2, 3, 6, 7};
  double carg[5] = {6, 8, 4, 3, 4};
  double xarg[5];
  double barg = 20;
  double farg;

  int errCode;
```

```
  // barg を変化させながら knapsackSolve を繰り返しコールする
  int ib;
  for (ib = 1 ; ib <= 30; ++ib) {
    barg = (double) ib;
    errCode = knapsackSolve(narg, aarg, barg, carg, xarg, &farg);
    printf("errCode=%d b=%g obj=%g ", errCode, barg, farg);
    printf(" x= ");
    int i;
    for (i = 0 ; i < narg ; ++i) {
      printf("%1.0f ", xarg[i]);
    }
    printf("\n");
  }
}
```

これを前項の knapsackSolve() と Nuorium Optimizer のライブラリとリンクするとロードモジュールが作成できます．VC++のメニューの「プロジェクト」から「スタートアップ プロジェクトに設定」を選び，次のようにこのプロジェクトを選択します．



続いて実行します：



すると，次のような出力が得られます．

```
errCode=0 b=1 obj=0  x= 0 0 0 0 0
errCode=0 b=2 obj=8  x= 0 1 0 0 0
errCode=0 b=3 obj=8  x= 0 1 0 0 0
errCode=0 b=4 obj=8  x= 0 1 0 0 0
errCode=0 b=5 obj=12  x= 0 1 1 0 0
errCode=0 b=6 obj=14  x= 1 1 0 0 0
     （中略）
errCode=0 b=26 obj=25  x= 1 1 1 1 1
errCode=0 b=27 obj=25  x= 1 1 1 1 1
errCode=0 b=28 obj=25  x= 1 1 1 1 1
errCode=0 b=29 obj=25  x= 1 1 1 1 1
errCode=0 b=30 obj=25  x= 1 1 1 1 1

C:\app\x64\Release\noClass.exe（プロセス 10156）は、コード 0 を伴って終了しました。
このウィンドウを閉じるには、任意のキーを押してください．．．
```

4.3 SIMPLE の例外処理

V21 より C++SIMPLE でエラーが発生した場合，C++の例外が投げられるようになりました．例外は std::exception で捕捉できます．以下 SIMPLE の例外処理の記述例です．なお，関数 knapsackSolve は C++SIMPLE モデルが記述された関数です．

SIMPLE の例外処理の記述例

```
#include "simple_exception.h"
       ..
try{
    knapsackSolve(n, a, b, c, x, f);
}catch(const std::exception &e){
    // catch SIMPLE error
}
```

第5章 C++SIMPLEモデル記述からクラスを生成して利用する例

本章で紹介するのもC++SIMPLEによって記述したモデルをプログラムの一部として呼び出す方法です．まず数理最適化モデルをモデリング言語C++SIMPLEで記述します．次にその記述から数理最適化モデルに対応するC++のクラスを生成します．ユーザプログラムではそのC++のクラスのオブジェクトを宣言することによって

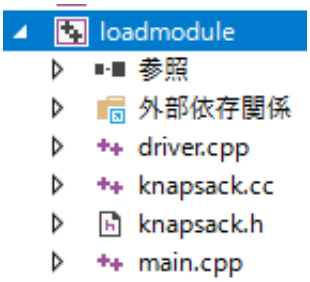
- データの読み込み
- 最適化の実行
- 変数の初期値の設定
- 最適解や関連する量の表示

などの操作を実現します．クラスオブジェクト同士は独立しているので，複数の問題をひとつの外部プログラムで操作するなどの操作が可能になります．

ソリューション `nuoptvcapp` のプロジェクト

`loadmodule`

がこのインタフェースを用いてナップサック問題を解くプログラム例です．



5.1 モデル

次に示すプログラムはこのアプリケーションの核心となるモデル記述です．通常のモデルと異なるのは，パラメータの宣言部分に `required` というキーワードがあることですが，これはこの指定のあったデータをC++の配列から入力することを示しています．その他は通常のモデル記述と同じです．この例ではこのモデルが次のような `knapsack.smp` というファイルに記述されたものとします．

knapsack.smp

```
//
//    ナップサック問題
//
```

```
Set S;
Element i(set = S);
IntegerVariable x(index = i, type = binary); // 整数変数
Parameter c(index = i, required);
Parameter a(index = i, required);
Parameter b(required);
Objective obj(type = maximize);

obj = sum(c[i] * x[i], i);    // 目的関数
sum(a[i] * x[i], i) <= b;     // 制約条件
```

このファイルは

（Nuorium Optimizer のインストール場所）\samples\app

にある zip ファイルに含まれています.

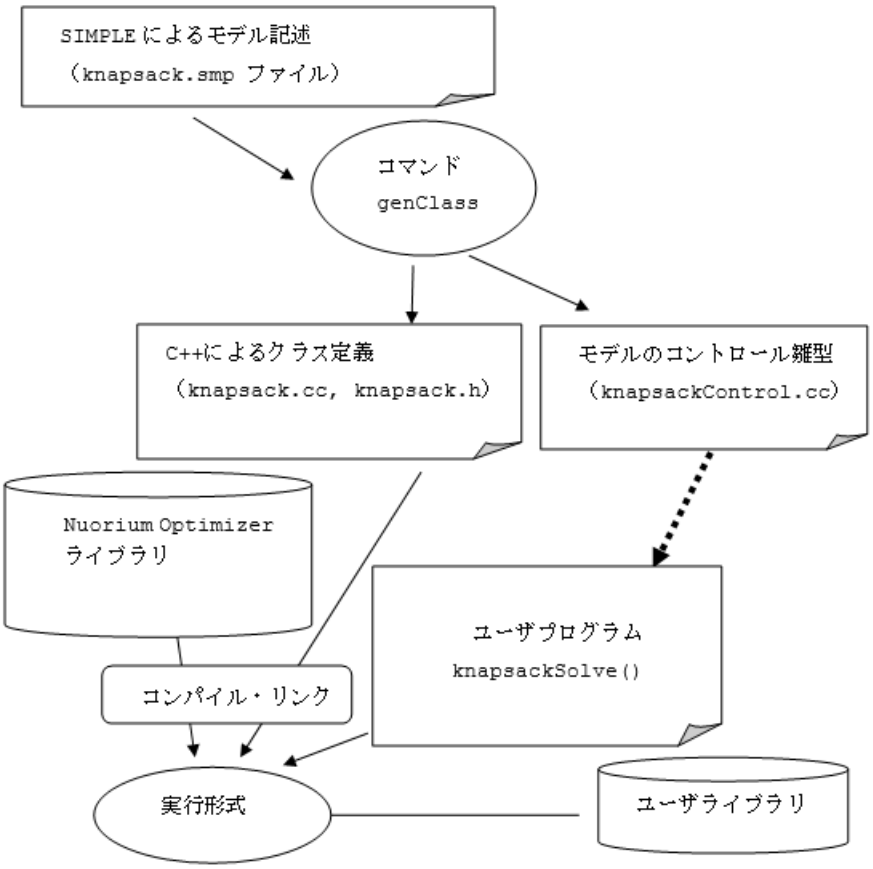
5.2 genClass のコール

前節のモデルはこのままでは C++で利用できる形ではありませんが．genClass というコマンドを用いて変換すると，対応するクラス宣言（knapsack.h）とクラスの実装（knapsack.cc）が自動生成されます.

次のページに関連するファイルの関係を図示します．ユーザはまず，モデリング言語 C++SIMPLE を用いてモデル記述を行い，genClass によってクラスの実装[4]を自動生成し，ユーザプログラムからそれを使います．モデルを利用する場合には，こうして生成されたクラス宣言（同 knapsack.h）をインクルードして，クラスオブジェクトを宣言します．このクラスオブジェクトのメンバを通じて，ユーザは自分の C++プログラム（同 knapsackSolve()）からこのモデルにデータや解を入出力（print(), dump()）する操作が可能になります.

実行形式の生成の際には，先ほど自動生成されたクラス実装をコンパイル，リンクします．こうして生成された C++SIMPLE のクラスは何個でも，また何回でも利用することができます．ただし C++SIMPLE のクラスを利用するに先立って，ユーザのプログラムから処理前に SimpleInitialize(), 処理後に SimpleClearBuffer() なる関数を呼び出す必要があります（これらは C++SIMPLE が独自に利用する記憶領域の確保と解放に必要となります）.

[4] このクラスの実装の内容をユーザは直接意識する必要はありません.



genClass の起動は DOS ウインドウのプロンプトから

```
> genClass knapsack.smp
```

とします．genClass は Nuorium Optimizer が提供するユーティリティーで，この実行のためには別途設定が必要となります．NUOPT 設定ツールの「環境設定」の「NUOPT へのパスを環境変数 PATH へ追加する。」の右側の「追加」ボタンをクリックし，Windows を再起動してください．詳細については Nuorium Optimizer インストールガイドを参照してください．以下が knapsack.smp に対する genClass の実行例です．

```
C:\app>genClass knapsack.smp
genClass.bat Ver *.*.* for Nuorium Optimizer
Copyright (C) 1991 NTT DATA Mathematical Systems Inc.
Build with Microsoft Visual Studio 2017(64bit) on 64bit Windows.
Output: knapsack.cc knapsack.h knapsackControl.cc knapsackCheck.cc knapsack_json.cc
genClass: Success.
```

この操作でこの数理最適化モデルに対応するクラス定義と実装が genClass を起動したフォルダに作成されます．

knapsack.h	knapsack.smp に対応するクラスの定義
knapsack.cc	同クラスの実装

knapsackControl.cc　利用方法サンプル

クラスの定義（knapsack.h）は，この数理最適化問題に対して操作を行うコントロール手続き（C++のコード）の中で操作を行う前に必ずインクルードする必要があります．クラスの実装（knapsack.cc）の内容は通常ユーザが意識する必要はありませんが，実行形式を作成する際にユーザのプログラムにコンパイル・リンクする必要があります．

ソリューション nuoptvcapp のプロジェクト

loadmodule

がこのインタフェースを用いてナップサック問題を解くプログラム例ですが，genClass で生成された実装（knapsack.cc）とクラスの定義（knapsack.h）を追加しています．

knapsackControl.cc は knapsack.smp で定義したクラスの利用方法のサンプルが記載されています．このコードは一度最適化実行を行うというものです．入力オブジェクトがあるクラスの場合には受け渡し用オブジェクトの宣言が書かれていますので，クラスの利用のコード作成時には参考になります．

knapsackControl.cc

```
#include "knapsack.h"
void simpleControl()
{
    Set S;
    Element i(set = S);
    Parameter c(name = "c", index = i);
    Parameter a(name = "a", index = i);
    Parameter b(name = "b");
    System_knapsack s1(c, a, b);
}
```

5.3 ドライバ

ではこのモデルを解く汎用の C++プログラムを準備しましょう．それはプロジェクトに追加されている

driver.cpp

というファイルで，knapsackSolve という名前の手続きです．このルーチンのインタフェースは入力として，ナップサック問題のモデルの

```
Parameter a(index = i);
Parameter b;
Parameter c(index = i);
```

に相当するデータ（aarg, barg, carg）を与えると

```
Variable x(index = i);
```

に相当する変数値（xarg）と目的関数（farg）を返すというものです．ドライバのこのインタフェースは問題に特化して調整できるので，より柔軟なコード作成が可能になります．

driver.cpp

```
#include "knapsack.h"   // インクルード宣言
//
//  ナップサック問題の求解
//
//
int knapsackSolve(int narg // 問題のサイズ
       ,double* aarg  // 係数 a
       ,double barg   // 係数 b
       ,double* carg  // 係数 c
       ,double* xarg  // 解ベクトル ( x )
       ,double* farg  // 目的関数
       )
{
  // 初期化
  SimpleInitialize();
  { // SimpleInitialize() のコールの後は{を付ける
    // 値転送用のオブジェクト
    Set S;
    Element i(set = S);
    Parameter b;
    Parameter c(index = i), a(index = i);

    // C 配列を初期化用のデータに与える.
    b = barg; // スカラはそのまま代入
    c.readD(narg, carg); // 配列から SIMPLE のオブジェクトへの設定
    a.readD(narg, aarg); // 配列から SIMPLE のオブジェクトへの設定

    //
    // knapsack 問題の求解
    //
    System_knapsack knap(c, a, b);
```

```
    int len;
    int* ind;
    double* knapx;
    knap.x.val.dump(len, ind, knapx);
    // knapsack 問題の x を C の配列である knapx に設定

    // 解を戻り配列に設定
    int it;
    for (it = 0; it < len; ++it) {
      xarg[it] = knapx[it];
    }
    *farg = result.optValue;
    // 目的関数値を取る簡単な方法 (asDouble() でも可能)

    // 不要な領域の破壊
    delete [] ind;
    delete [] knapx;
  } // SimpleClearBuffer() のコールの前を}で閉じる.
    // 終了処理
    SimpleClearBuffer();
    return result.errorCode; // エラーコードを返す.
}
```

SimpleInitialize() と SimpleClearBuffer() は C++SIMPLE を利用した処理の最初と最後に必ず必要なコールです．実装上の理由により，SimpleInitialize のコールより後と，SimpleClearBuffer() のコールより前は{ }でくくる必要があります．SIMPLE のオブジェクト（Set や Element）の宣言は，この{ }の中で行います．SimpleInitialize をコールせずに SIMPLE のオブジェクトを宣言して利用した場合，あるいはこの{}の外で SIMPLE のオブジェクトを宣言すると実行時エラーとなります．

手続きの中ほどで System_knapsack というクラスのオブジェクト knap を宣言していますが，これが knapsack.smp というモデル記述に対応するクラスのオブジェクトの宣言です．このプログラムの先頭の

```
#include "knapsack.h"
```

がこのモデル定義を含むファイルで，モデル knapsack に対応するクラスを利用する場合には必ずインクルードする必要があります．

一般に NAME.smp なるモデルには System_NAME というクラスが対応します．クラスの定義ファイル NAME.h には System_NAME の定義が書かれています．そのため，System_NAME を使用する際には必ず NAME.h のインクルードが必要になります．

クラス宣言の中で SIMPLE オブジェクトを required というキーワード付きで

```
// knapsack.smp の中
Parameter c(index = i, required);
Parameter a(index = i, required);
Parameter b(required);
```

のように宣言しているので，呼び出し側の手続きで同様に宣言した受け渡し用のオブジェクト

```
// driver.cpp の中
Set S;
Element i(set = S);
Parameter b;
Parameter c(index = i), a(index = i);
```

に

```
b = barg; // スカラはそのまま代入
c.readD(narg, cary); // 配列は readD を用いる.
a.readD(narg, aary); // 配列は readD を用いる.
```

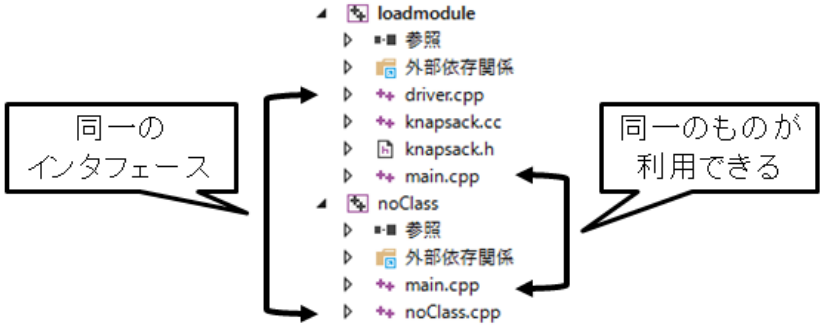
として値を設定，システムオブジェクト knap の宣言の際に

```
System_knapsack knap(c, a, b); // knapsack 問題の求解
```

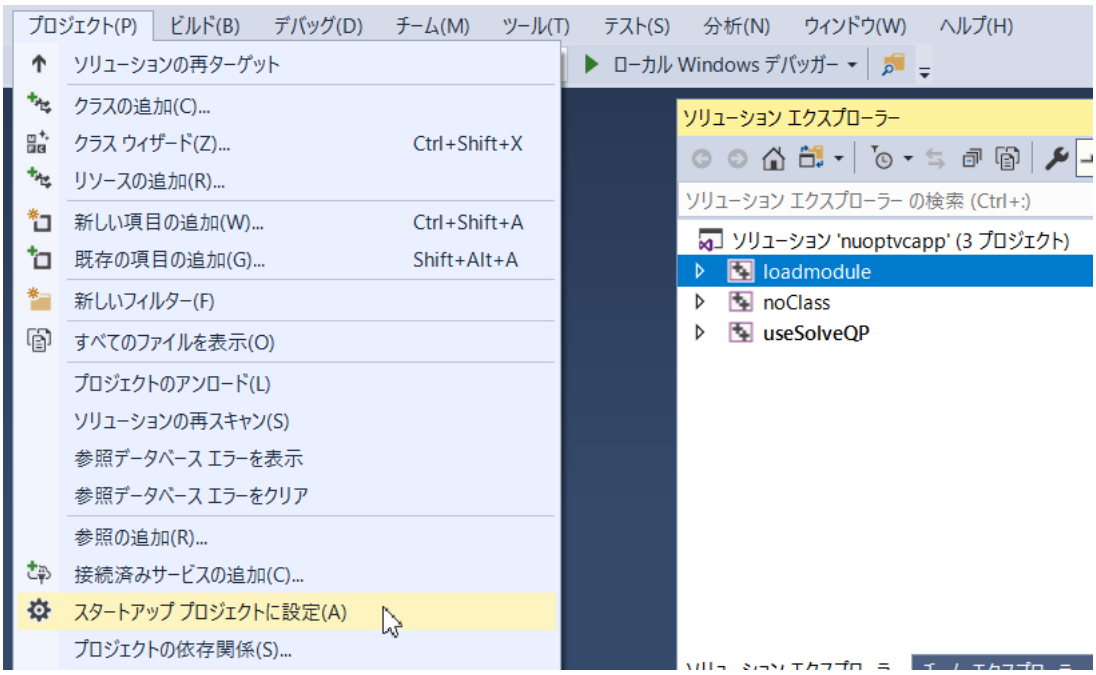
のように渡しています（この引数にはモデル中，required というキーワード付きで宣言したオブジェクトが出現順に並びます）.

5.4 C++関数からの呼び出し

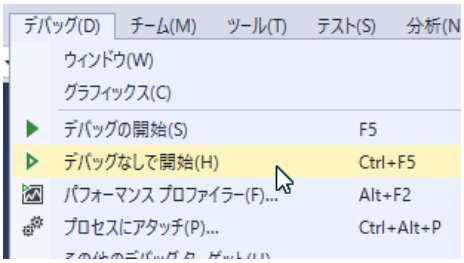
driver.cc で定義された knapsackSolve はプロジェクト noClass に含まれている noClass.cpp と同一の仕様ですので，プロジェクト noClass のメインルーチンをそのまま使うことができます.



VC++のメニューの「プロジェクト」から「スタートアップ プロジェクトに設定」を選び，次のようにこのプロジェクトを選択します.



続いて実行します：



すると，プロジェクト `noClass` の出力と同一の出力が得られます．

第6章 外部接続時に利用されるC++SIMPLEのツール

プロジェクト loadmodule の driver.cpp やプロジェクト noClass の noClass.cc では C++SIMPLE のモデルを定義するだけではなく，値を設定したり解を取得したりしていますが，このような場合に必要なツールについて解説します.

6.1 データファイルの読み込み

データファイルの読み込みには，readData 関数や readTable 関数を使用します.

```
char* datafilename = "c:\\temp\\data.dat";
FILE* fp = fopen(datafilename, "r"); // データファイルを開く
readData(fp, datafilename); // データファイルを読み込む
fclose(fp);

char* csvfilename = "c:\\temp\\data.csv";
fp = fopen(csvfilename, "r"); // csv ファイルを開く
readTable(fp, csvfilename); // csv ファイルを読み込む
fclose(fp);
```

readData 関数および readTable 関数のプロトタイプ宣言は次の通りです.

```
int readData(FILE* fp, char* filename);
int readTable(FILE* fp, char* filename);
```

いずれも第一引数に入力ファイルへのファイルポインタ，第二引数に入力ファイルのファイル名を与えます．readData 関数は dat 形式のデータ読み込みに，readTable 関数は csv 形式のデータ読み込みに用います．それぞれの入力ファイルの形式については，Nuorium Optimizer/C++SIMPLE マニュアルをご覧ください.

6.2 モデルから作成されたオブジェクト（システムオブジェクト）の操作

driver.cpp で行っているように，genClass で生成したクラス定義から

```
System_knapsack knap(c, a, b);
```

として生成されたオブジェクト（ここでは knap）がデータを与えられて作成したその数理最適化モデル（この場合には knapsack 問題）そのものに対応します（以下ではこれを説明のため「システムオブ

ジェクト」と呼びます）．ここで

```
knap.show();
```

とすると，問題の中身が表示されます（showSystem() と同じ）．

システムオブジェクトと数理最適化モデルの構成要素については，一般的な原則として以下があります．

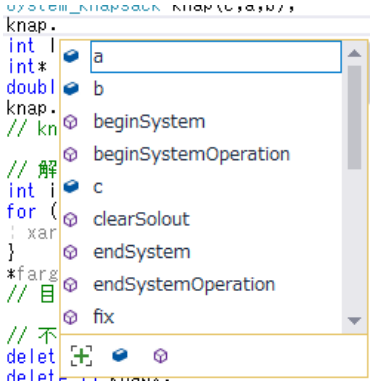
> System_NAME のオブジェクト s について s.x は NAME.smp 中のオブジェクト x に対応する．

すなわち，

knapsack.smp

```
//
//   ナップサック問題
//
    Set S;
    Element i(set = S);
    IntegerVariable x(index = i, type = binary); // 整数変数
    Parameter c(index = i, required);
    Parameter a(index = i, required);
    Parameter b(required);
    Objective obj(type = maximize);

    obj = sum(c[i] * x[i], i);     // 目的関数
    sum(a[i] * x[i], i) <= b;      // 制約条件
```

と定義されたモデルから生成された knap について，VC++の GUI で“.”を打った後に以下のように，参照可能なメンバーが現れますが，それぞれはこのモデル中のオブジェクトに対応しています．



例えば

```
knap.x
```

はモデル中の変数（Variable x）に対応し，このオブジェクトに対する表示手続き：

```
print(), cout, simple_printf()
```

は通常の SIMPLE オブジェクトに対するのと全く同様に可能です．すなわち

```
knap.x.print();
simple_printf("objective = %f\n", knap.obj);
```

とすると，それぞれ knapsack.smp のモデル中の x，obj の値が表示されます．

```
simple_printf("x[%s] = %d\n", knap.i, knap.x[knap.i]);
```

は少々複雑ですが，“knap.i”はモデル中の Element i の意味となりますので，モデル中で

```
simple_printf("x[%s] = %d\n", i, x[i]);
```

としたのと同じ，変数と添字の書式付表示となります．

6.3 C/C++の配列の内容の設定

モデルを操作するルーチンが行う最初の操作は C/C++の配列として渡されたデータを SIMPLE のオブジェクトに設定するということです．スカラー値の場合には，double または int の値をそのまま

```
Parameter b;
b = barg; // barg は double の入力引数
```

と代入できますが，添字付けられた大量のオブジェクトを配列から一気に読み込むには readD という手続きを使います．readD は一般に C の配列から SIMPLE のオブジェクトを設定するためのツールで，本章のようなアプリケーション接続の際によく利用されます．呼び出しの一般形は次の通りです．

```
void readD(int len1, int len2, .., int lenM, double* data) const
```

引数の型:

len1, len2, ..., lenM： 次元のサイズ

data： 実際のデータ

与えられる配列データ（data）を一般の多次元配列（添字は 1 から始まる整数値）に読みかえて解釈するので，行列など多次元の配列を定義することができます．次は実際の利用例です．

readD の使用例

```
// 配列内容
  double cont[27] = {1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7,
                     2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7,
                     3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7,
```

```
                    4.1, 4.2, 4.3, 4.4, 4.5, 4.6};
  Set a1(name = "a1");   Element i1(set = a1);
  Parameter p1(index = i1);    // 1 次元配列データ
  p1.readD(5, cont);
  // 結果: p1: [1]=1.1,[2]=1.2,[3]=1.3,[4]=1.4,[5]=1.5

  Set a2(name = "a2", dim = 2); Element i2(set = a2);
  Parameter p2(index = i2);   // 2 次元配列データ
  p2.readD(2, 3, cont);
  // 結果: p2: [1,1]=1.1,[1,2]=1.2,[1,3]=1.3
  //           [2,1]=1.4,[2,2]=1.5,[2,3]=1.6

  Set a3(name = "a3", dim = 3); Element i3(set = a3);
  Parameter p3(index = i3);  // 3 次元配列データ
  p3.readD(2, 3, 2, cont);
  // 結果: p3: [1,1,1]=1.1,[1,1,2]=1.2,[1,2,1]=1.3,[1,2,2]=1.4
  //           [1,3,1]=1.5,[1,3,2]=1.6,[2,1,1]=1.7,[2,1,2]=2.1
  //           [2,2,1]=2.2,[2,2,2]=2.3,[2,3,1]=2.4,[2,3,2]=2.5
```

6.4 求解

データの設定が済んだら数理最適化問題を解きますが，loadmodule プロジェクトの driver.cpp のように，genClass で生成したクラス定義から作ったモデルを解いている場合には宣言を

```
System_knapsack knap(c, a, b); // knapsack 問題の求解
```

と宣言を行えば自動的に求解が行われます．変数の初期値の設定を行ってから求解したい場合などは

```
options.noDefaultSolve = 1;
```

と設定してからオブジェクトの宣言を行い，

```
knap.x[i] = 1; // 初期値の設定
knap.solve(); // ここで求解
```

と solve() という手続きを呼びます．

noClass プロジェクトの noClass.cpp の場合のようにモデル定義と操作を同時に行っている場合には

```
solve(); // 最適化の実行
```

とすれば，それ以前に定義したモデルに対する求解が行われます．

6.5 C の配列への書き出し

操作する手続きが次に行うのは，求解した結果を C++の配列の形で取得することです．システムオブジェクトに対して dump() という手続きを起動します．loadmodule プロジェクトの driver.cpp では

```
int len;
char** ind;
double* knapx;
knap.x.val.dump(len, ind, knapx);
```

と行うことによって，C++の配列 knapx に解を書き出しています．一般に dump の引数は

```
dump(int len, char*& ind, double* data);
```

で，

len: ind，data の総長さ
ind: インデックス文字列の配列（長さ len）
data: データ本体（長さ len）

です．data[0]，data[1]，... には添字 ind[0]，ind[1]，... がそれぞれ対応しています．データの並び順はインデックスをソートした際の自然な順番（数字の場合には昇順，文字列の場合には辞書順）となり，この場合，入力データは readD によって設定されているので，データ並びに対応する knapx には番号順に値が設定されることが保証されます．そのため，driver.cpp ではインデックスを見ないで戻りの配列に値を設定しています．なお，ind，data に対応する領域は dump 内部で確保されるためコールした側で解放する必要があります．スカラー値（添字のないオブジェクト）は長さ 1 のオブジェクト（添字の値としてはヌル文字列が設定されます）として同様に dump で取得することができますが，便利な手続きとして double を返す asDouble() という手続きがあり，

```
double objval = obj.val.asDouble();
```

のように値を取ることができます．

6.6 計算結果に関する情報の取得

大域変数である result に直前の最適化の結果が設定されます．よく利用されるのは

```
int result.status  ... 最適化の終了時の状態
```

で，最適化のステータスコード（正常終了時には 0）を返します．

```
double result.optValue;
```

は最適化後の目的関数の値で，dump() による取得よりも簡便なので，ここでは目的関数値を戻すのに利用しています．

6.7 求解操作と代入

VariableParameter の値を変化させながら複数回最適化を行う場合，モデルに対応するシステムオブジェクト model について行う場合には

```
for (int p = 1; p <= 5; ++p) {
  model.p = p; // VariableParameter の設定
  model.solve();// 求解
}
```

のように記述を行います．(p がモデル中の VariableParameter の名前とします)．システムオブジェクトのメンバ（上記では p）に対しての代入は通常のモデル内での代入と同様に機能します．

noClass プロジェクトの noClass.cpp の場合のようにモデル定義と操作を同時に行っている場合には

```
for (int pi = 1; pi <= 5; ++pi) {
  p = pi; // VariableParameter の設定
  solve();// 求解
}
```

とします（VariableParameter p と同じ名前の int 変数 p を使うことはできませんので，pi としています)．

6.8 Nuorium Optimizer 求解オプション

プログラム内で Nuorium Optimizer の動作を制御する求解オプションを設定することができます．

C++SIMPLE モデルに対して求解オプションを設定するには，モデルやドライバ中に

```
options. 求解オプション名 = 値;
```

と書きます．“options”は大域的に有効な nuoptParam のクラスオブジェクトです．設定した内容は次の Nuorium Optimizer の起動時に反映されます．“options”は大域的に有効なので，以前の指定が残ることにご注意ください．

solveLP，solveQP を用いて問題を解く際に Nuorium Optimizer の求解オプションを設定するには

```
nuoptParam param;
```

のように Nuorium Optimizer の求解オプション群を示す nuoptParam なるクラスのオブジェクトを定義して（名前は任意)，

```
param. 求解オプション = 値
```

として，求解オプションを設定，その nuoptParam のオブジェクトを solveLP，solveQP の最初の引数として与えます．こうすると，その問題を解く際（solveLP，solveQP のコール）に対してここで設定

した求解オプションが与えられます．例えばモデルのアルゴリズムを単体法に指定する場合は，

```
nuoptParam param;
param.method = "simplex";
```

と書きます．

nuoptParam クラスで設定できる求解オプションは，モデリング言語 C++SIMPLE の options で定義できる求解オプションと同じです．設定できる求解オプションの詳細については「Nuorium Optimizer マニュアル」の「求解オプション設定」をご参考ください．

第 7 章　VC++プロジェクトの設定

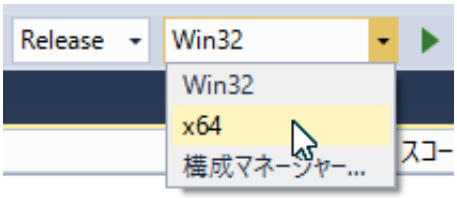
7.1　Microsoft Visual Studio プロジェクトの設定

ソリューション

```
nuopvcapp
```

に追加されているプロジェクトでは設定済みですが，一般に Nuorium Optimizer のライブラリとの連結を行ってライブラリ（.LIB），あるいは DLL を作成する場合には，プロジェクトの設定を行う必要があります．以下では，Visual Studio から Nuorium Optimizer を利用する際に行う必要のある設定を紹介します．なお，ここでは Visual Studio 2017 の画面をもとに説明していますが，他のバージョンでも同様の設定を実施してください．

まず，Win32 プロジェクト設定ではなく 64 ビットプロジェクト構成にする必要があります．変更の際は，画面上部にあるコンボボックスをクリックし，「x64」を選択します．

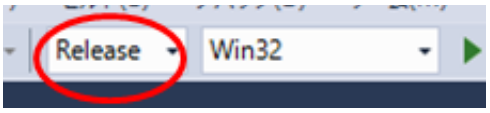


なお，「x64」が表示されていない場合は，以下のように設定します．

1. 画面上部にあるコンボボックスをクリックし，「構成マネージャー」を選択します．
2. アクティブソリューションプラットフォーム欄をクリックし，「新規作成」を選択します．
3. ［新しいプラットフォームを入力または選択してください］欄をクリックし，「x64」と入力後，[OK] ボタンをクリックします．

以上の設定で，ソリューション内のすべてのプロジェクトが 64 ビットプロジェクト構成になります．

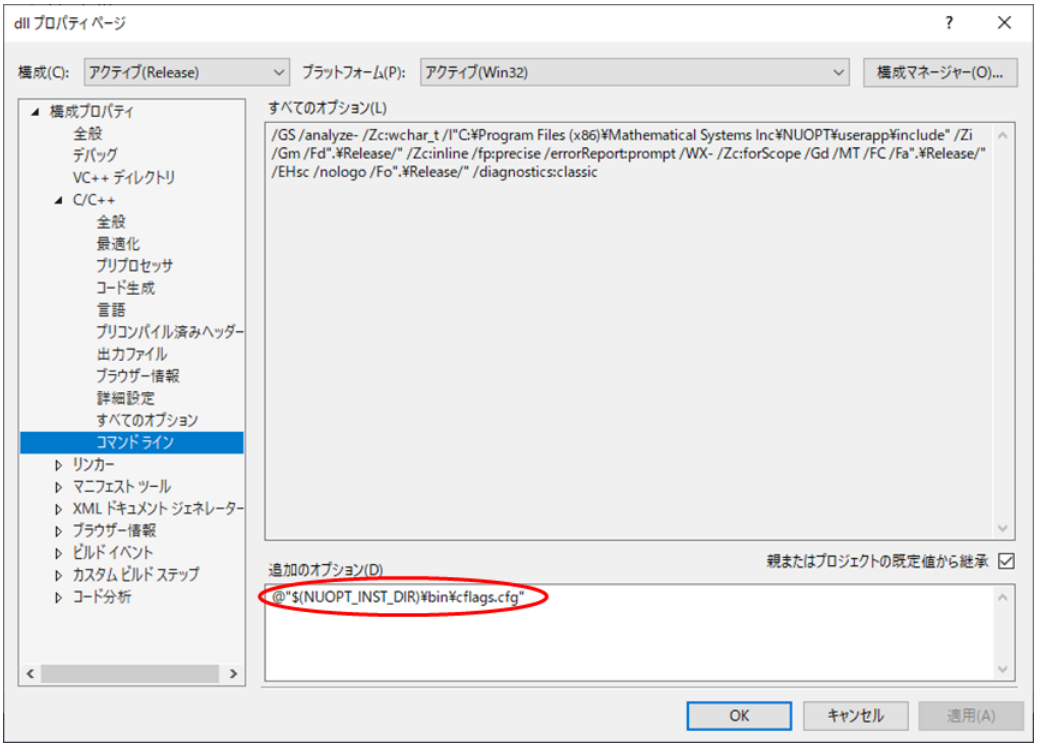
次にプロジェクトの構成を「Release」にする必要があります．下図のように，メニューバー下部にあるコンボボックスに「Release」と設定します．



次にビルドおよびリンクの設定を行います．これより以下の設定は全て［プロジェクト］→［プロパティ］→［構成プロパティ］から行います．まず「構成プロパティ」→「C/C++」→「コマンドライン」と選択し，ウインドウの下部ある「追加のオプション」に

```
@"$(NUOPT_INST_DIR)\bin\cflags.cfg"
```

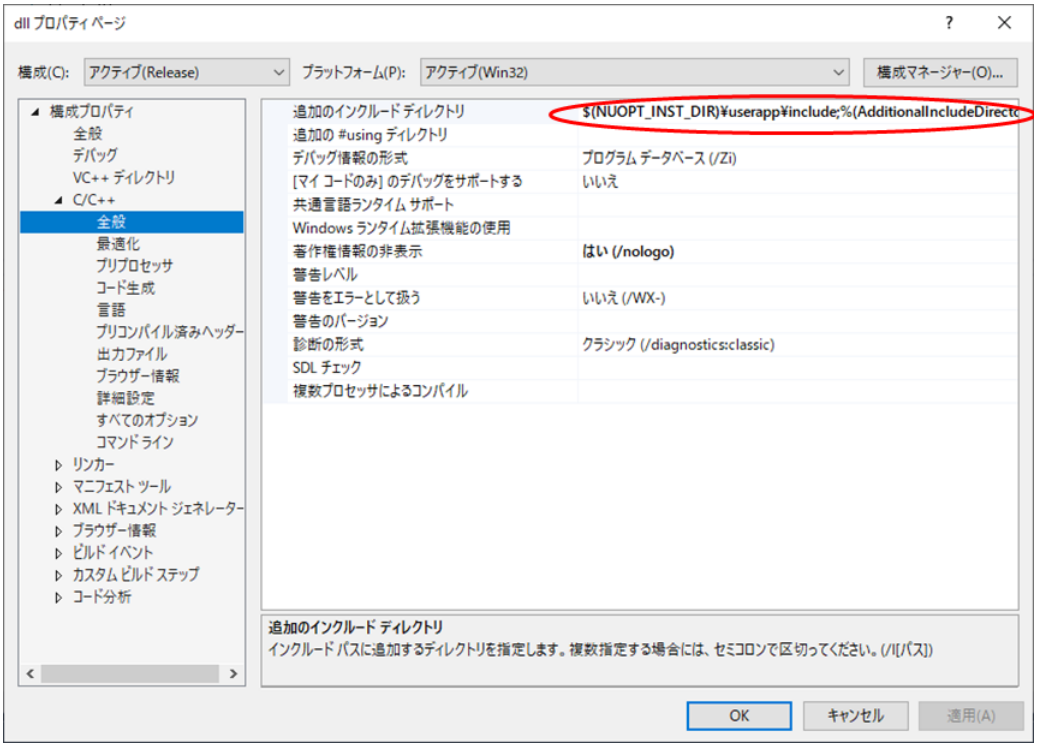
を追加します．NUOPT_INST_DIR は Nuorium Optimizer のインストール場所を表す環境変数です．



「構成プロパティ」→「C/C++」→「全般」と選択し，ウインドウ内にある「追加のインクルードディレクトリ」に

```
$(NUOPT_INST_DIR)\userapp\include
```

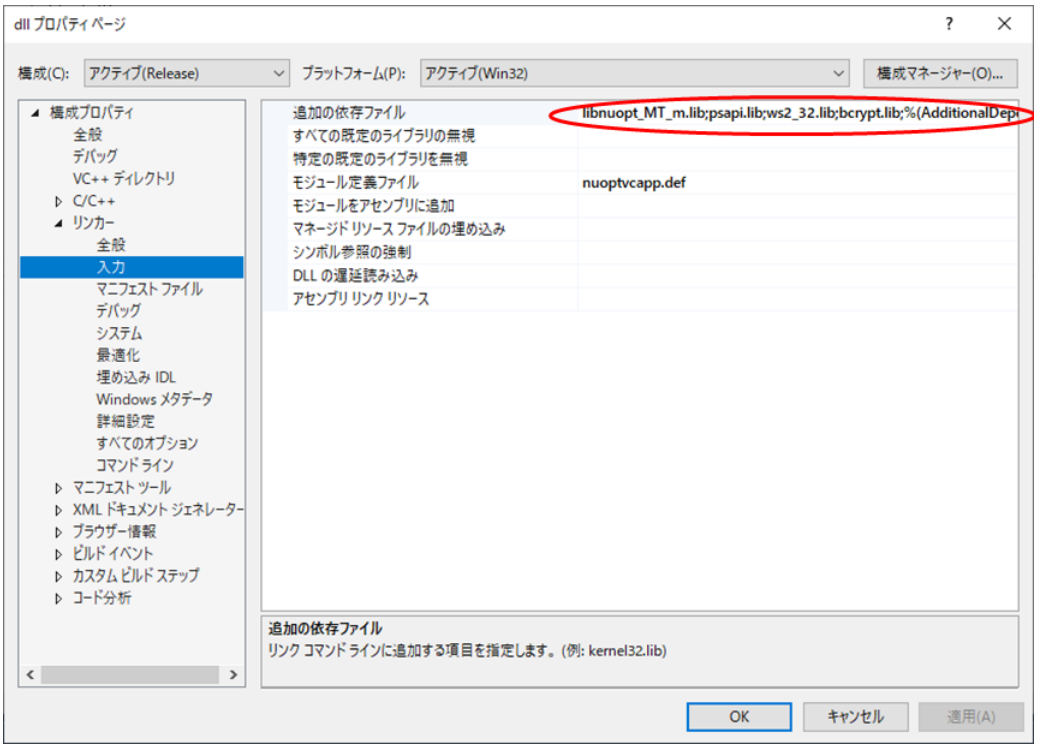
を追加します．なお，下線部は Nuorium Optimizer のインストール場所に対応します．



「構成プロパティ」→「リンカー」→「入力」と選択し，ウインドウ内にある「追加の依存ファイル」に

```
libnuopt_MT_m.lib;psapi.lib;ws2_32.lib;bcrypt.lib;%(AdditionalDependencies)
```

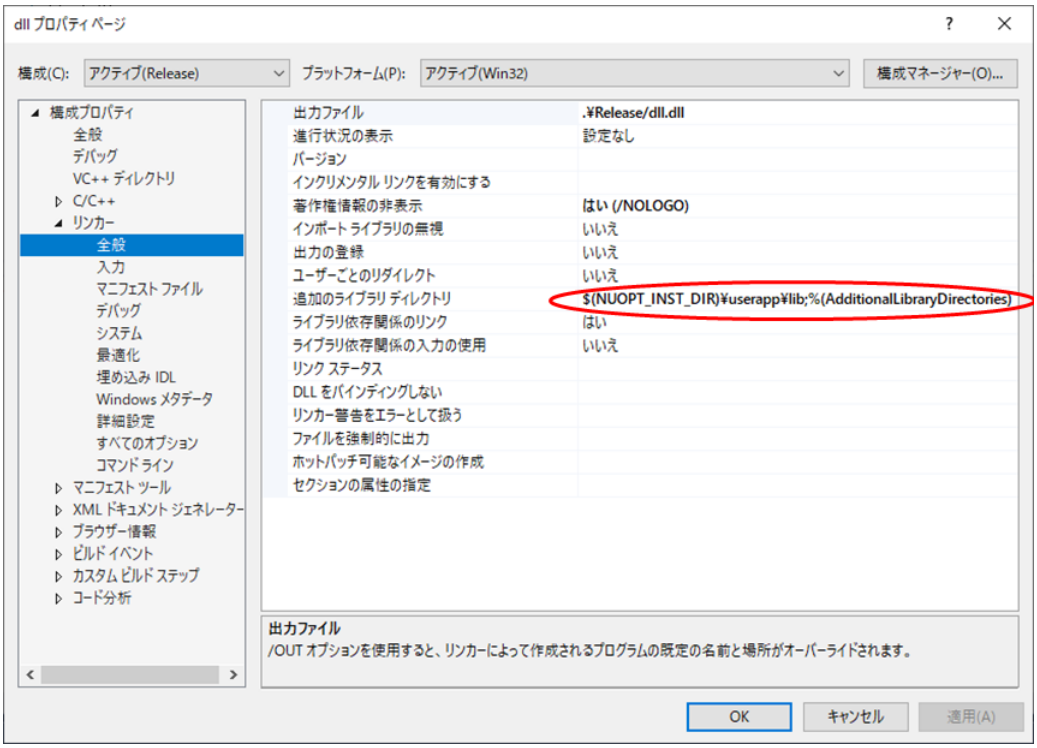
を追加します．なお本設定は Nuorium Optimizer のライブラリに接続しているアプリケーションで使用するライブラリがデフォルトの「マルチスレッド」でコンパイルする場合です．詳細は以降にある「※注意 1」をご覧ください．



「構成プロパティ」→「リンカー」→「全般」と選択し，ウインドウ内にある「追加のライブラリディレクトリ」に

```
$(NUOPT_INST_DIR)\userapp\lib;
```

を追加します．上記の下線部は Nuorium Optimizer のインストール場所に対応します．



※注意 1（アプリケーションで利用しているランタイムライブラリについて）：

上記は Nuorium Optimizer のライブラリに接続しているアプリケーションで使用するランタイムライブラリが「マルチスレッド」でコンパイルされている場合です（この設定はプロジェクト→プロパティ→ C/C++→コード生成で現れる「ランタイムライブラリ」の選択に対応します）．もし，Nuorium Optimizer のライブラリと接続するコードがデフォルトと違って「マルチスレッド DLL」になっている場合には，3. で指定するライブラリの名前を

```
libnuopt_MD_m.lib
```

としてください．

7.2 外部 CLAPACK(CBLAS) の使用方法

本節では，外部 CLAPACK(CBLAS) を Nuorium Optimizer ライブラリへリンクする方法について説明をします．基本的な設定方法はコンパイラのバージョンが異なっても同じであるため，ここでは Microsoft Visual Studio 2017 を例にします．

まずは，「7.1 Microsoft Visual Studio プロジェクトの設定」を行い，正しくソリューションがビルド

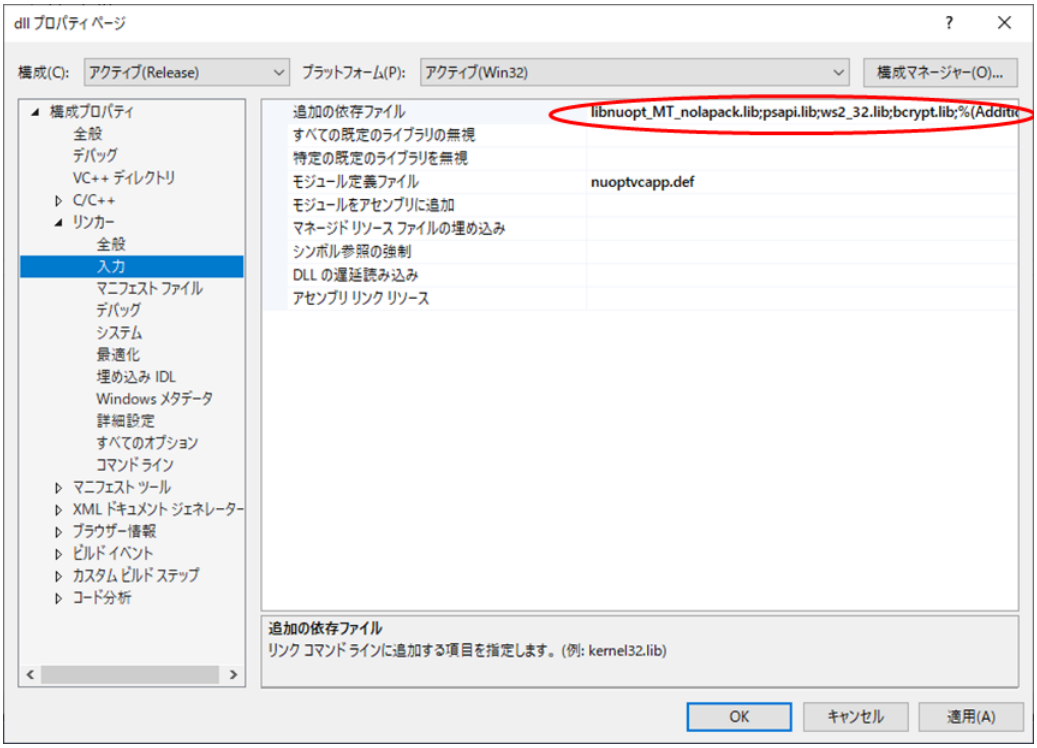
できることを確認してください.

次に，Nuorium Optimizer のライブラリを外部 CLAPACK とリンクするためのものに変更します．具体的には，プロジェクト→プロパティ→構成プロパティから，「構成プロパティ」→「リンカー」→「入力」と選択し，ウインドウ内にある「追加の依存ファイル」の

```
libnuopt_MT_m.lib;psapi.lib;ws2_32.lib;bcrypt.lib;%(AdditionalDependencies)
```

を次のように変更します.

```
libnuopt_MT_nolapack.lib;psapi.lib;ws2_32.lib;bcrypt.lib;%(AdditionalDependencies)
```

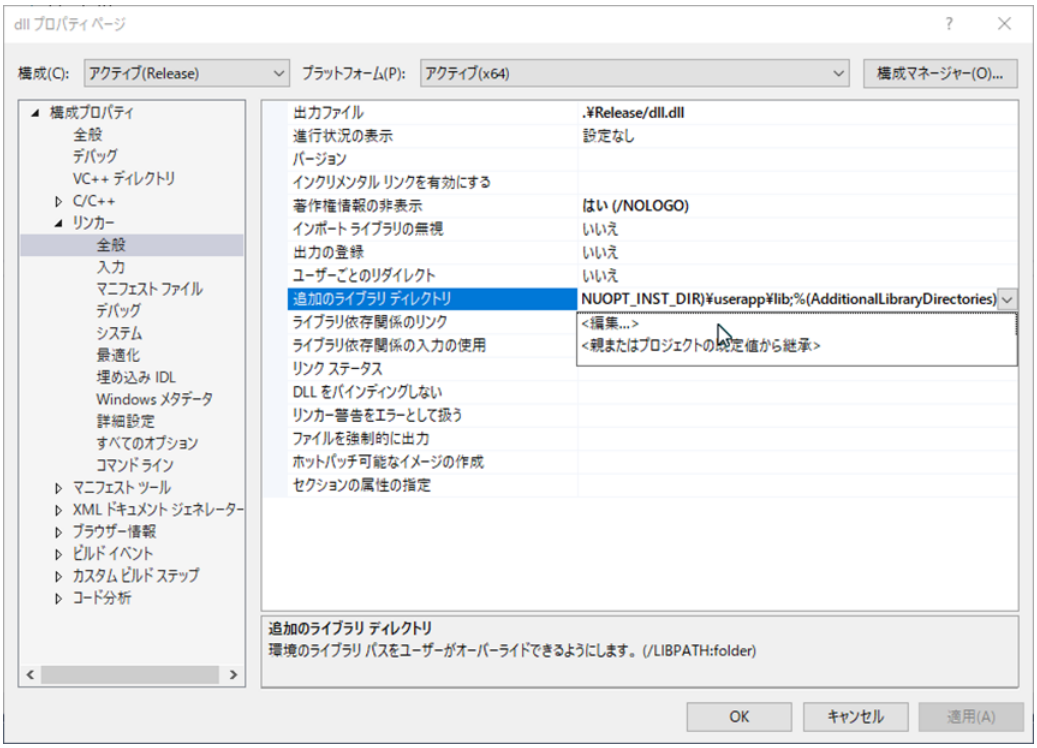


この後，ソリューションのビルドを行うと，以下のようなエラーメッセージが出力され正しくビルドが完了しません．これは外部 CLAPACK(CBLAS) の関数のリンクに失敗しているためです．そのため，以下で外部 CLAPACK(CBLAS) ライブラリの設定を行います.

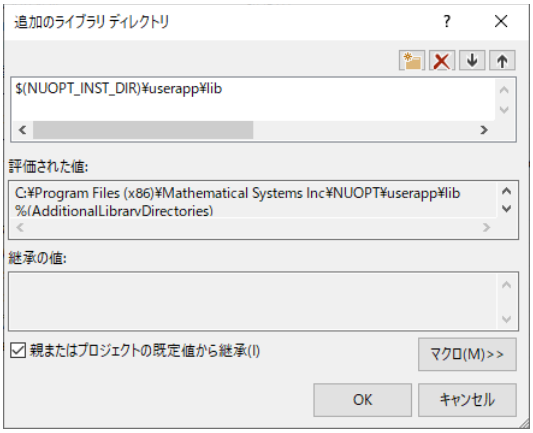
```
1>libnuopt_MT_nolapack.lib(nptOut.obj) : error LNK2019: 未解決の外部シンボル mkl_get_version_string が関数 "class std::basic_string<char,struct std::char_traits<char>,class std::allocator<char> > __cdecl getMKLVersion(void)" (?getMKLVersion@@YA?AV?$basic_string@DU?$char_traits@D@std@@V?$allocator@D@2@@std@@XZ) で参照されました。
1>C:\app\x64\Release\loadmodule.exe : fatal error LNK1120: 1 件の未解決の外部参照
1>プロジェクト "loadmodule.vcxproj" のビルドが終了しました -- 失敗。
```

先の「追加の依存ファイル」に外部 CLAPACK(CBLAS) のライブラリを入力してください．ライブラリ名は使用する外部 CLAPACK(CBLAS) により決まりますので，必要なら CLAPACK(CBLAS) のマ

ニュアル等をご覧ください．次に，この (これらの) ライブラリがあるフォルダを指定します．「構成プロパティ」→「リンカー」→「全般」と選択し，ウインドウ内にある「追加のライブラリディレクトリ」の右側にあるボタンをクリックし．「編集」を選択してください．



すると次のようなダイアログが表示されます．



このダイアログから，先に指定した外部 CLAPACK(CBLAS) のあるフォルダを追加してください．ライブラリが複数ある場合は複数のフォルダを指定する必要があるかもしれません．適切なフォルダに関しては外部 CLAPACK(CBLAS) のマニュアル等をご覧ください．

以上で外部 CLAPACK(CBLAS) の設定になります．ご不明な点がございましたら

nuopt-support@ml.msi.co.jp

までお問い合わせください．

参考文献

[1] W. Hock and K. Shittkowski, Test examples for nonlinear programming codes, Springer Verlag, 1981.

索　引

U

え

か

き

く

こ

し

せ

と

な

に

は

ら

れ