유체 운동의 실시간 시뮬레이터 개발을 위한 적합직교분해 및 인공신경망 기반의 차수축소모델 구성 프레임워크 개발

2022 9th OpenFOAM Korea Users' Community Conference

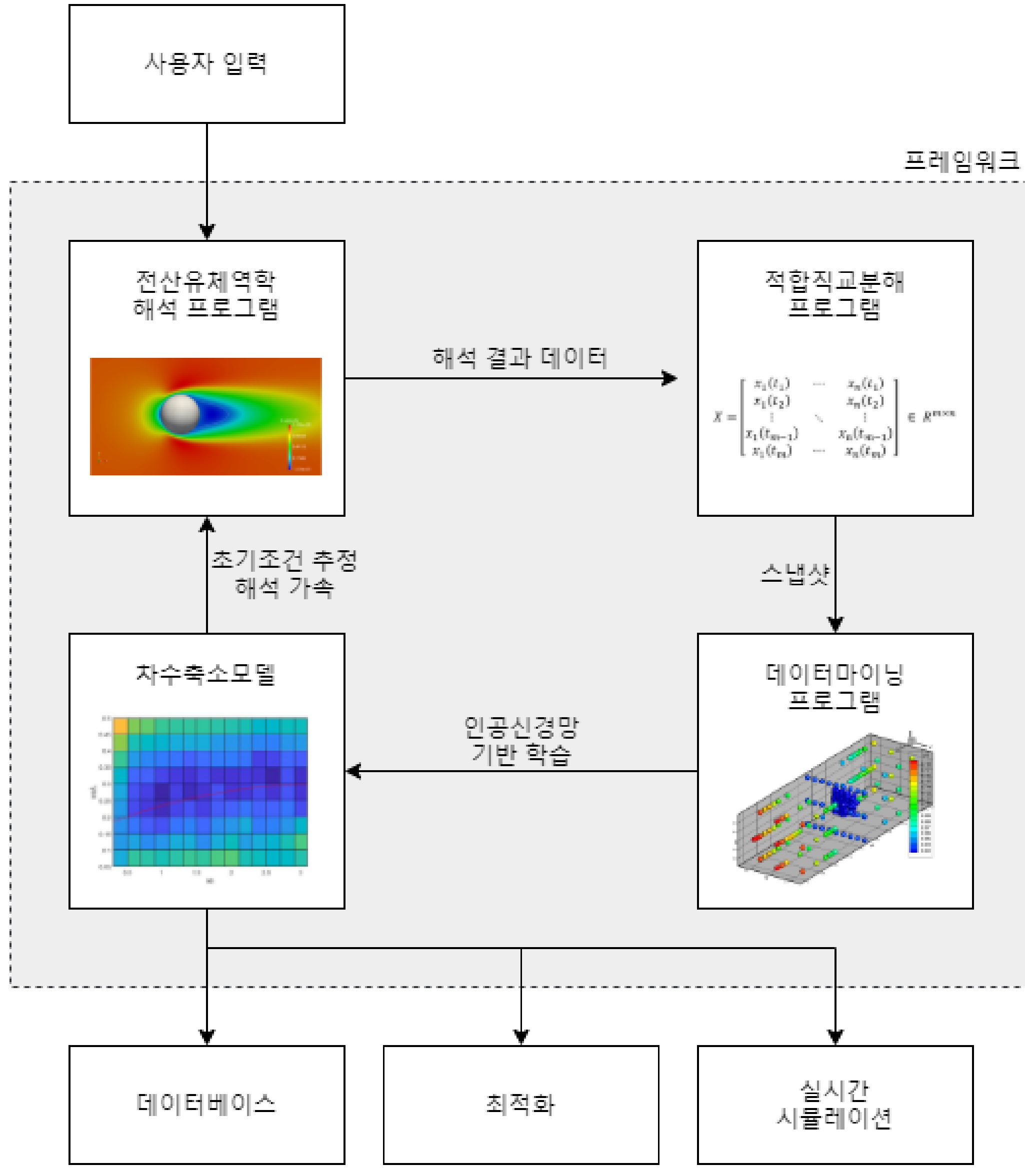
2022.9.22.

(주) 넥스트폼 이용현

NEXTfoam open source CFD consulting

개요

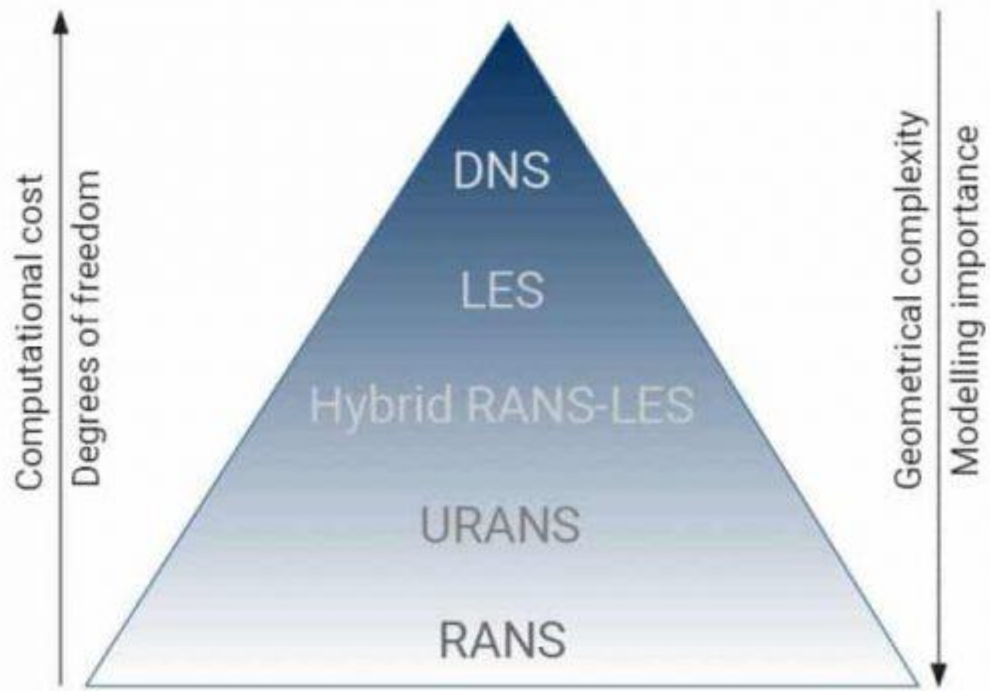
- 연구 목표
  - 반복해석 소요 시간 저감
  - 기 존재하는 구성 요소 프로그램을 결합, 자동화된 프레임워크 개발
- 키워드
  - 전산유체역학 (CFD)
  - 적합직교분해 (POD)
  - 데이터마이닝
  - 차수축소모델 (ROM)
  - 인공신경망 (ANN)
  - 데이터베이스
  - 최적화
  - 실시간 시뮬레이션
- 목차
  - (1) 적합직교분해
  - (2) DAKOTA toolkit
  - (3) 프레임워크 개발



NEXTfoam open source CFD consulting

적합직교분해 (POD)

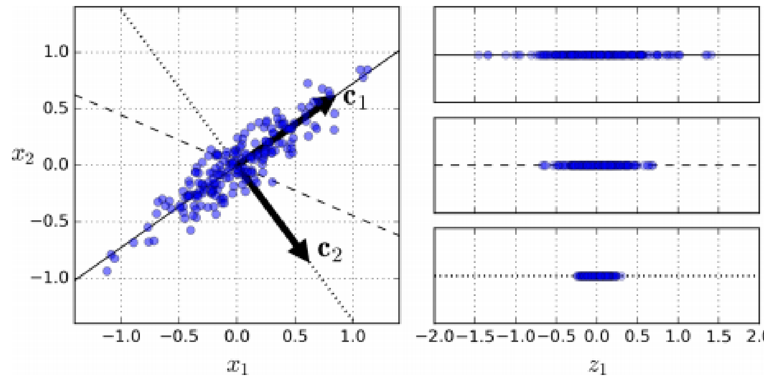
- 기 존재하는 전산유체역학 해석 결과 사용, 모드 추출
- 신규 조건에 대한 해석 결과를 예측

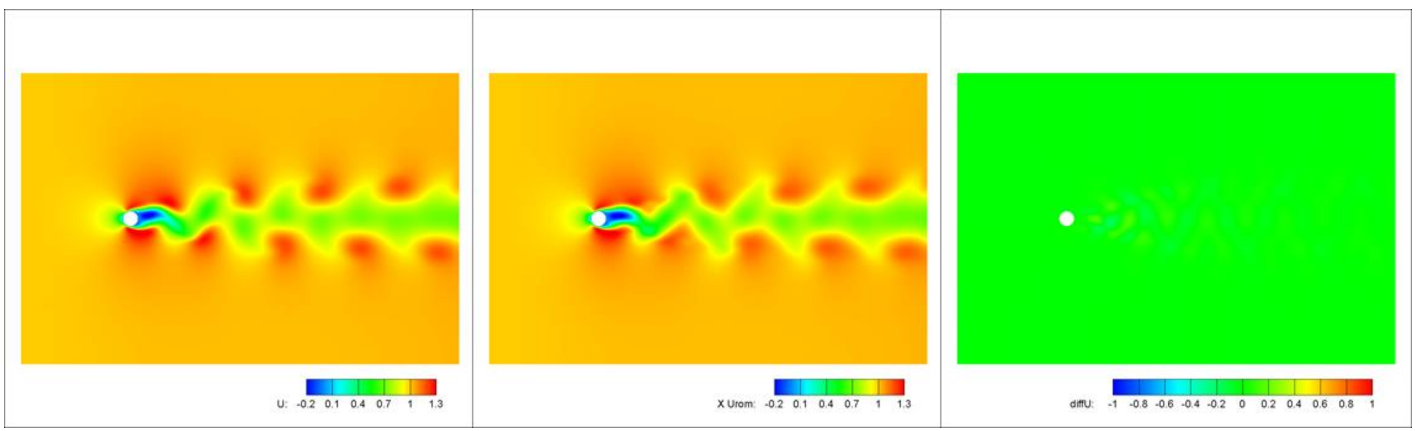


POD, Flow angle, Panel method, etc.

적합직교분해 (POD)

- 주성분 분석 (PCA) 기법 기반
  - 특이치 분해 (SVD)
  - 이미지 압축 등 광범위한 분야에 사용
  - CFD 해석 결과에 적용 가능





NEXTfoam open source CFD consulting

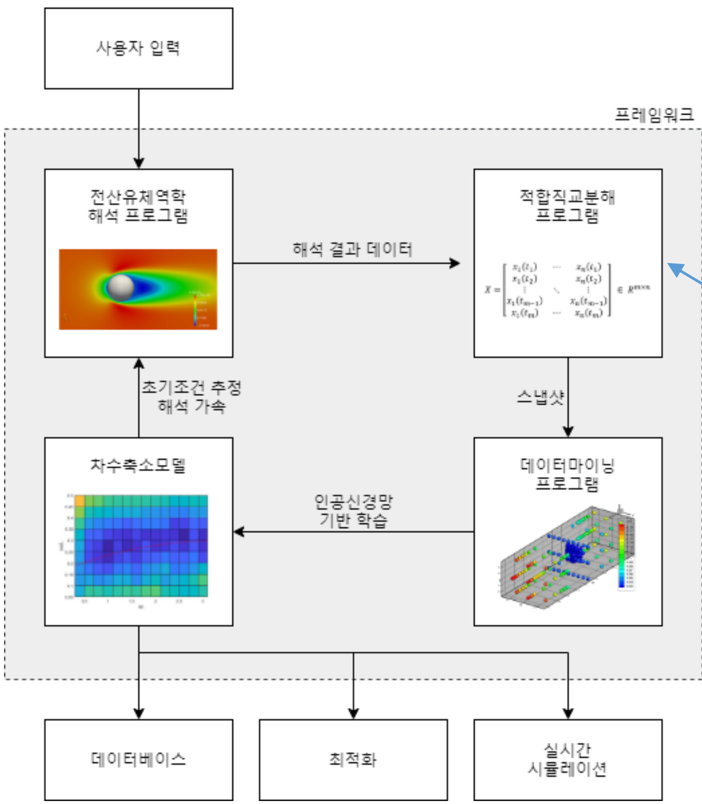
적합직교분해 (POD)

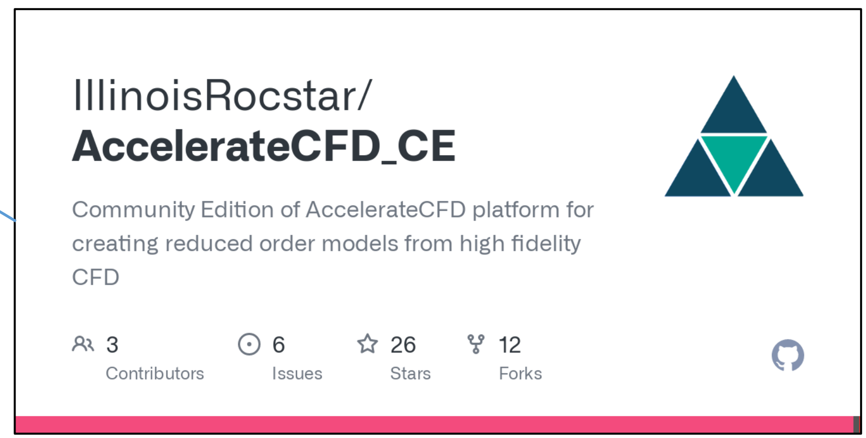
- 고전 적합직교분해
  - 연속적인 시간에 대해 적합직교분해 수행
  - 1967, Lumley, J. L., "The structure of inhomogeneous turbulent flows", Atmosphere and its Influence on Radio Wave Propagation. pp. 166-178, Moskow: Nauka.

- 스냅샷 적합직교분해 *
  - 특정 시점들에 대한 해석 결과만으로 적합직교분해 수행
  - 1987, Sirovich, L., "Turbulence and the dynamics of coherent structures. Part I: coherent structures", Applied mathematics, vol.45, no.3, pp.561-571.

적합직교분해 (POD)

- 프레임워크 구성 요소 프로그램 선정 (AccelerateCFD)
  - 공개 소스 (GNU GPL v3), 절차 지향적 코드
  - OpenFOAM utility 형태
  - 계산 결과 입출력 및 추가 기능 개발 용이





적합직교분해 (POD)

▪ 절차

r/AccelerateCFD_CE

README.md

There are five modules to this software.

podBasisCalc

- This application calculates POD basis for velocities in CFD case directory and gives information about flow energy contained in each POD basis. It solves eigen value problem to identify most important flow characteristics from the flow.

podPrecompute

- This application calculates gradient and tensor terms as well as some tensor innerproducts for velocity and POD basis vectors. All the pre-computed data which is essential for computing the reduced order model (ROM) is written in the case directory. The file "prevVals.csv" contains time varying coefficients obtained using proper orthogonal decomposition.

podROM

- This application uses all the data written out from **podPrecompute** application and calculates the time varying coefficients for spatial POD basis to construct reduced order model. It writes values of time varying coefficients in the case directory which are used for reconstructing the velocity field.

podReconstruct

- This application reads in the values of time varying coefficients and reconstructs velocities. These reconstructed velocities are automatically written into their respective time directories of CFD case for ease of visualization.

podPostProcess

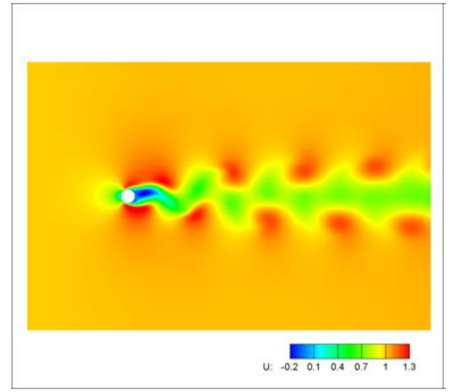
- This application allows users to obtain additional information from reduced order as well as full order models for comparison and reference purposes. Right now this utility supports calculation of time varying coefficients from full order model that can serve as a reference to reduced order time coefficients calculated using podROM utility. This utility operates based on command line arguments. All available arguments are explained later in this guide.

02_SVD	• 공분산행렬(Covariance Martix) 생성 • Eigenvalue & Eigenvectors 추출 ✓ Left Eigenvector = POD mode (Basis function)	$R = X^T X$ $X = \begin{bmatrix} x_1(t_1) & \cdots & x_n(t_1) \\ \vdots & \ddots & \vdots \\ x_1(t_m) & \cdots & x_n(t_m) \end{bmatrix} \in R^{m \times n}$ $R\beta_m = \lambda_m \beta_m$
03_POD	• Calculate the basis function(ϕ_m) • Normalize basis function($\bar{\phi}_m$) • Calculate POD's expansion coefficient(a_m)	$\phi_m = X^T \bullet \beta_m$ $\bar{\phi}_m = \phi_m / \sqrt{\phi^2}$ $a_m = X^T \bullet \bar{\phi}_m$
04_Interpolation Coefficient	• POD's expansion coefficient(a_m)**을 Design space의 변수에 따라 보간 또는 회귀모델 생성**	• Linear interpolation using SciPy module (1D) • 또는 RBF 모델 사용(2D, Isight 사용)
05_Reconstruction	• 보간/회귀모델로 생성된 POD's expansion coefficient과 basis function을 이용하여 Flow filed 를 재생성	$X^{new} = a_m \bullet \bar{\phi}_m$

NEXTfoam open source CFD consulting

적합직교분해 (POD)

- 절차
  - (1) 해석 결과 (snapshot) 확보

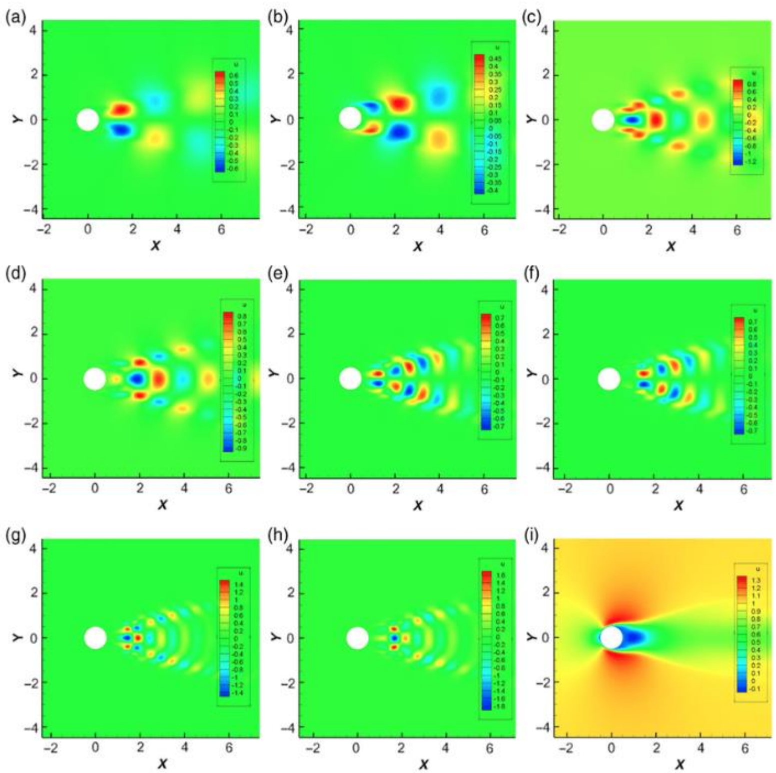


  - 입력 parameter 조건 별 해석 결과
    - AOA, BETA, Mach #, geometric parameter 등 종류 무관
    - 많을수록 정확도↑
  - AccelerateCFD 구동을 위해 단일 OpenFOAM 케이스 폴더로 취합
  - N개 격자 * M개 시점 해석 결과가 취합된 N×M 행렬 Y

$$\overrightarrow{y_j} = \begin{bmatrix} y_{1j} \\ y_{2j} \\ \cdots \\ y_{Nj} \end{bmatrix} \qquad Y = \begin{bmatrix} \overrightarrow{y_1} & \overrightarrow{y_2} & \cdots & \overrightarrow{y_M} \end{bmatrix}$$

적합직교분해 (POD)

- 절차
  - (2) 모드 (basis) 추출
    - 해석 결과의 기저 벡터
    - 선형결합으로 유동장 재현



POD modes of the oscillating cylinder: (a) POD mode #1, (b) POD mode #2, (c) POD mode #3, (d) POD mode #4, (e) POD mode #5, (f) POD mode #6, (g) POD mode #7, (h) POD mode #8 and (i) average field.

02_SVD	• 공분산행렬(Covariance Martix) 생성 • Eigenvalue & Eigenvectors 추출 ✓ Left Eigenvector = POD mode (Basis function)	$R = X^T X$ $X = \begin{bmatrix} x_1(t_1) & \cdots & x_n(t_1) \\ \vdots & \ddots & \vdots \\ x_1(t_m) & \cdots & x_n(t_m) \end{bmatrix} \in R^{m\times n}$ $R\beta_m = \lambda_m \beta_m$

$$Y = U\Sigma V^T$$

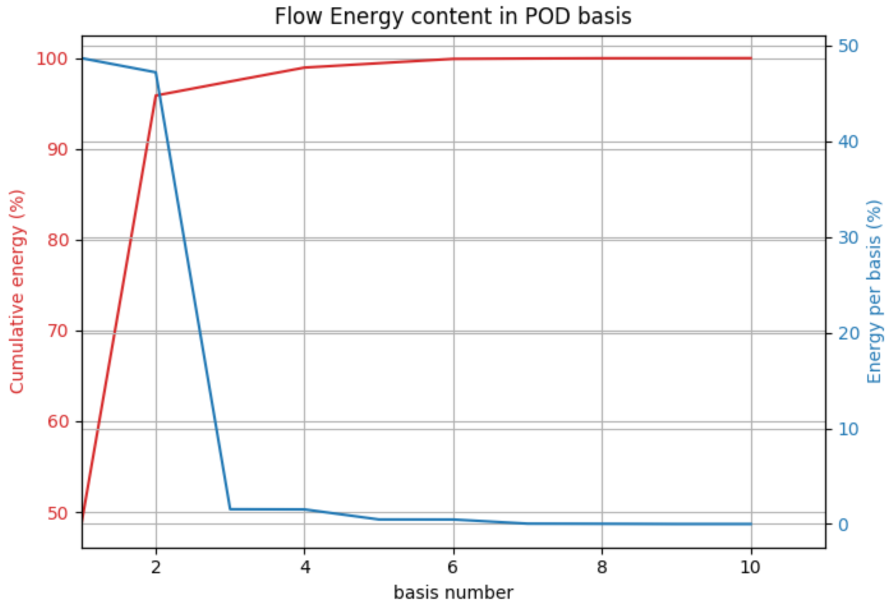
$$R = Y^T Y \qquad R\vec{v_i} = \sigma_i^2 \vec{v_i}, \qquad \vec{u_i} = \frac{\vec{Yv_i}}{\sigma_i}$$

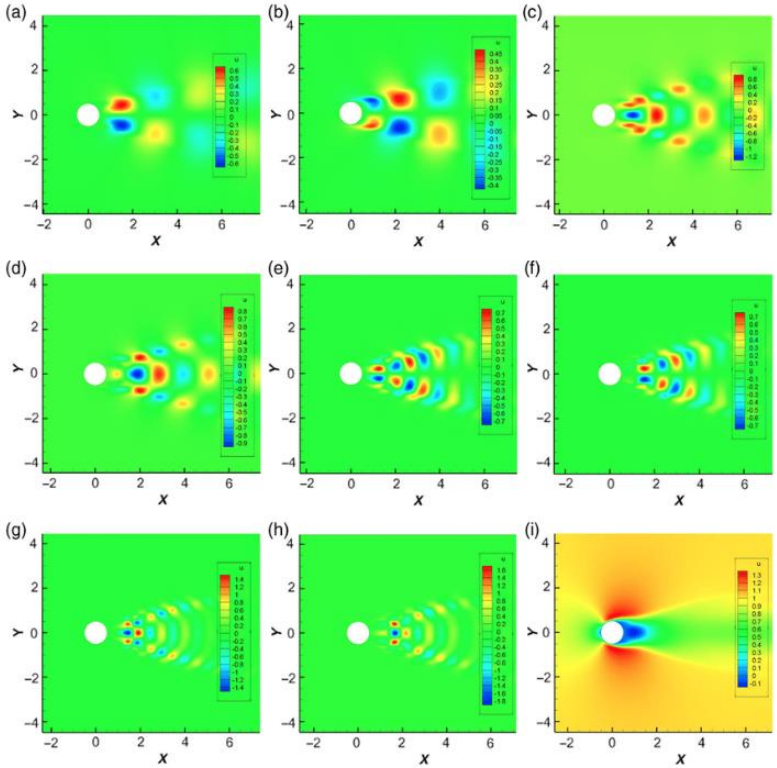
```
forAll(timeDirs, timei)
{
  n = 0;
  forAll(timeDirs, timej)
  {
     Cmn(m, n) = 0.0;
    // applying symmetry
    if (n < m)
    {
      Cmn(m, n) = Cmn(n, m);
      n++;
      continue;
    }

    volScalarField U1dotU2(generateCustomField(runTime,mesh,"U1dotU2"),
                                      (vels[timei]&vels[timej])*cellVolume);

    Cmn(m,n) = Cmn(m,n) + gSum(U1dotU2);
    n++;
  }
  m++;
}

// Normalization of correlation matrix by dividing with total number of
Cmn = Cmn/nDim;

// Self Adjoint Eigen Solver is used here to solve for eigenvalue proble
Info<< "Solving eigenvalue problem" << nl;

Eigen::SelfAdjointEigenSolver<Eigen::MatrixXd> es(Cmn);
```

적합직교분해 (POD)

- 절차
  - (2) 모드 (basis) 추출
    - 해석 결과의 기저 벡터
    - 선형결합으로 유동장 재현
    - 5~10개 모드에 99.9% 에너지 분포
    - 전체 해석 결과 요약/예측 가능

02_SVD	• 공분산행렬(Covariance Martix) 생성 • Eigenvalue & Eigenvectors 추출 ✓ Left Eigenvector = POD mode (Basis function)	$R = X^T X$ $X = \begin{bmatrix} x_1(t_1) & \cdots & x_n(t_1) \\ \vdots & \ddots & \vdots \\ x_1(t_m) & \cdots & x_n(t_m) \end{bmatrix} \in R^{m \times n}$ $R\beta_m = \lambda_m \beta_m$





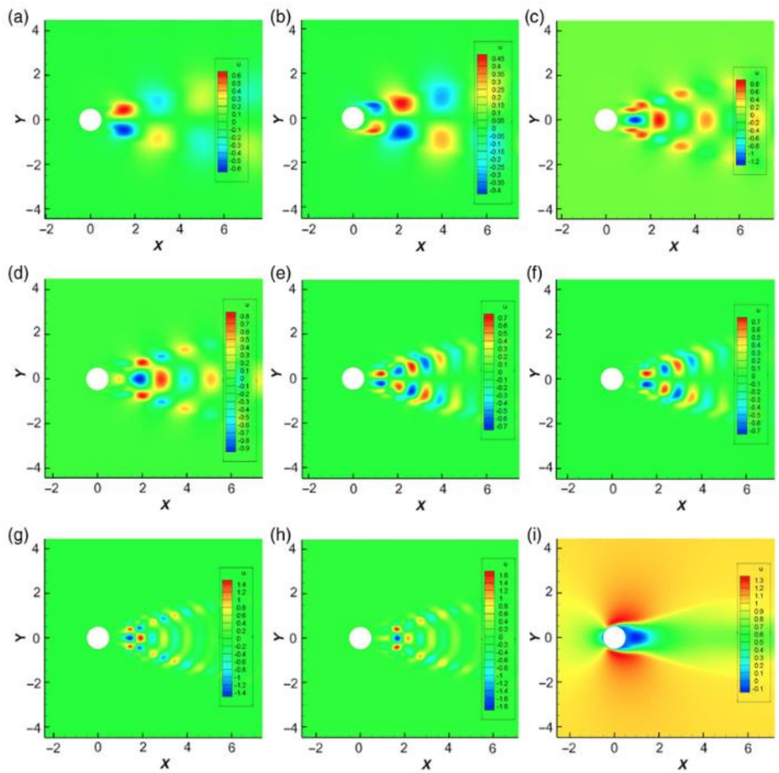
POD modes of the oscillating cylinder: (a) POD mode #1, (b) POD mode #2, (c) POD mode #3, (d) POD mode #4, (e) POD mode #5, (f) POD mode #6, (g) POD mode #7, (h) POD mode #8 and (i) average field.

NEXTfoam open source CFD consulting

적합직교분해 (POD)

- 절차
  - (3) expansion coefficient 계산
    - 각 snapshot에 포함된 basis의 비중

03_POD	• Calculate the basis function(ϕ_m) • Normalize basis function($\bar{\phi}_m$) • Calculate POD's expansion coefficient(a_m)	$\phi_m = X^T \bullet \beta_m$ $\bar{\phi}_m = \phi_m / \sqrt{\phi^2}$ $a_m = X^T \bullet \bar{\phi}_m$



POD modes of the oscillating cylinder: (a) POD mode #1, (b) POD mode #2, (c) POD mode #3, (d) POD mode #4, (e) POD mode #5, (f) POD mode #6, (g) POD mode #7, (h) POD mode #8 and (i) average field.

podBasisCalc.C | podPrecompute.C | podROM.C | podFlowReconstruct.C | podPostProcess.C

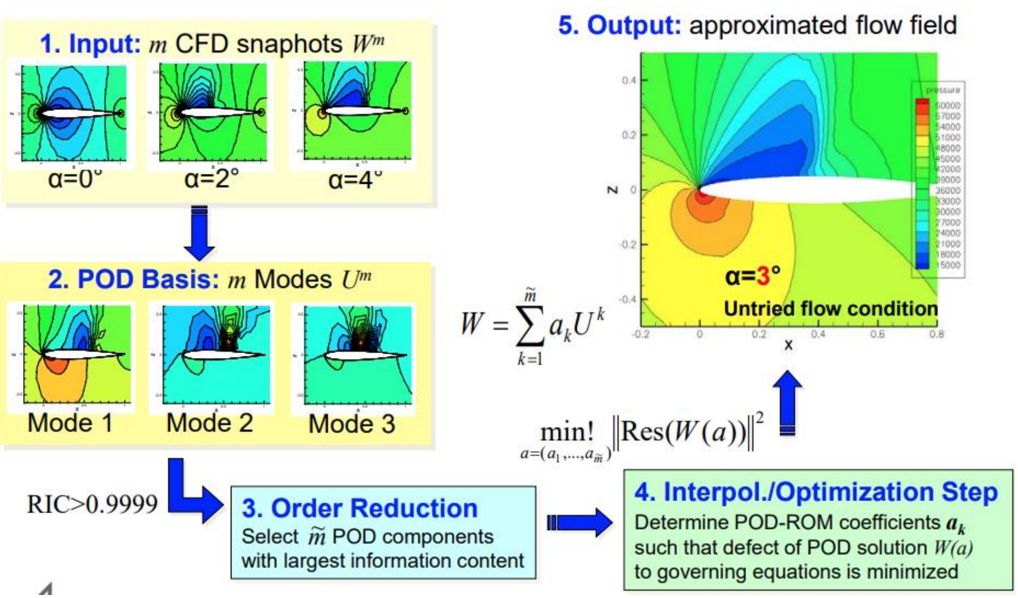
```
for (int i=0; i<nDim; i++) {
  std::string uGradSigName;
  uGradSigName = "uGradSigName_" + std::to_string(i);
  volVectorField uGradSig(generateCustomField(runTime,mesh,uGradSigName),
                          UMean&gradSigs[i]);

  uGradSigs.push_back(uGradSig);

  std::string sigGradUName;
  sigGradUName = "sigGradUName_" + std::to_string(i);
  volVectorField sigGradU(generateCustomField(runTime,mesh,sigGradUName),
                          sigs[i]&gradU);
  sigGradUs.push_back(sigGradU);
}

for (int i=0; i<nDim; i++) {
  for (int j=0; j<nDim; j++) {
    std::string sigGradSigName;
    sigGradSigName = "sigGradSigName_" + std::to_string(i+nDim*j);
    volVectorField sigGradSig(generateCustomField(runTime,mesh,sigGradSigName),
                          sigs[i]&gradSigs[j]);
    sigGradSigs[i+nDim*j] = sigGradSig;
  }
}

std::vector<double> constant(nDim,0.0);
std::vector<std::vector<double>> linear(nDim, std::vector<double>(nDim,0.0));
std::vector<std::vector<std::vector<double>>> quadratic(nDim, std::vector<std::vector<double>>(nDim, std::vec

// this loop calculates the Galerkin System matrices Q L C for the ROM equation.
// constant term, linear term, and quadratic term
for (int k=0; k<nDim; k++) {
  constant[k] = -1*innerProductPOD(sigs[k],UgradU,cellVolume)+ (nu+nu_tilda)*innerProductPOD(sigs[k],laplUMea
  for (int m=0; m<nDim; m++) {
    linear[k][m] = -1*innerProductPOD(sigs[k],uGradSigs[m],cellVolume)
    - innerProductPOD(sigs[k],sigGradUs[m],cellVolume) + (nu+nu_tilda)*innerProductPOD(sigs[k],laplSigs[m],ce
    for (int n=0; n<nDim; n++) {
      quadratic[k][m][n] = -1*innerProductPOD(sigs[k],sigGradSigs[m+nDim*n],cellVolume);
    }
  }
}
```

적합직교분해 (POD)

- 절차
  - (4) expansion coefficient 보간
    - 입력 매개변수 (AOA, etc.)
    - POD‘s expansion coefficient
    - Snapshot으로부터 보간 / 대체모델 생성

$$\vec{y_j} \approx \sum_{i=1}^{d} a_{ij}\vec{w_i}$$

04_Interpolation Coefficient	• POD's expansion coefficient(a_m)을 Design space의 변수에 따라 보간 또는 회귀모델 생성	• Linear interpolation using SciPy module (1D) • 또는 RBF 모델 사용(2D, Isight 사용)



podBasisCalc.C | podPrecompute.C | podROM.C | podFlowReconstruct.C | podPostProcess.C

```
  int writeSteps = 0;

  if(writeFreq == 0){
    writeSteps = nSteps/numDirs;}
  else{
    writeSteps = writeFreq;}

  std::ofstream afiles;
  afiles.open("avals.csv");

  for (int t=0; t<nSteps+1; t++){
    cout << "t = " << timeElapsed << endl; // Case progress info in terminal
    for (int i=0; i<nDim; i++) {
      double da = constant[i];
      for (int j=0; j<nDim; j++) {
        da += linear[i+j*nDim]*prevAvals[j];
        for (int k=0; k<nDim; k++) {
          da += quadratic[i+j*nDim+k*nDim*nDim]*prevAvals[k]*prevAvals[j];
        }
      }
      avals[i] = prevAvals[i] + da*dt;
    }

    for (int i=0; i<nDim; i++){
      prevAvals[i] = avals[i]; // updating previous avals
      avalues[t][i] = avals[i]; // storing a values for in 2D vector for lat
    }

    if(t%writeSteps == 0){
      double tcol = startTime + dt*t;
      afiles << tcol << ",";
      for (int j=0; j<nDim; j++){
        afiles << std::fixed << std::setprecision(16) << avalues[t][j] << ",
      }
      afiles << endl << std::flush;
    }

    timeElapsed += dt;
  }
  afiles.close();
```

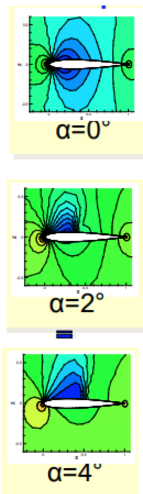
적합직교분해 (POD)

- 절차
  - (4) expansion coefficient 보간
    - 입력 매개변수 (AOA, etc.)
    - POD's expansion coefficient
    - Snapshot으로부터 보간 / 대체모델 생성

$$\vec{y}_j \approx \sum_{i=1}^{d} a_{ij}\vec{w}_i$$

04_Interpolation Coefficient	• POD's expansion coefficient(a_m)을 **Design space의 변수에 따라 보간 또는 회귀모델 생성**	• Linear interpolation using SciPy module (1D) • 또는 RBF 모델 사용(2D, Isight 사용)

Snapshot (Known)



const.

U(0) = a1(0) σ1 + a2(0) σ2 + a3(0) σ3

U(2) = a1(2) σ1 + a2(2) σ2 + a3(2) σ3

U(4) = a1(4) σ1 + a2(4) σ2 + a3(4) σ3

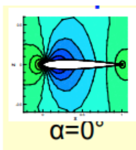
U(3) = a1(3) σ1 + a2(3) σ2 + a3(3) σ3

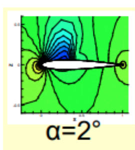
적합직교분해 (POD)

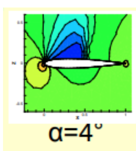
- 절차
  - (5) 유동장 재생성
    - Basis × Expansion coefficient

05_Reconstruction	• 보간/회귀모델로 생성된 POD's expansion coefficient과 basis function을 이용하여 Flow filed 를 재생성	$X^{new} = a_m \bullet \bar{\phi}_m$

Snapshot (Known)





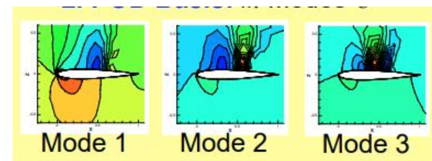


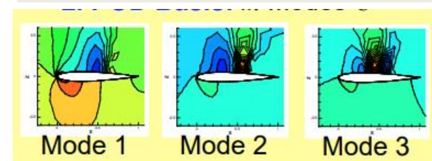
const.

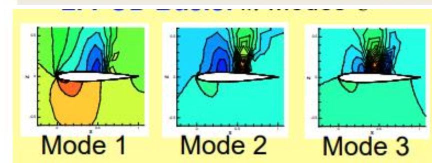
$$U(0) = a1(0)\,\sigma1 + a2(0)\,\sigma2 + a3(0)\,\sigma3$$

$$U(2) = a1(2)\,\sigma1 + a2(2)\,\sigma2 + a3(2)\,\sigma3$$

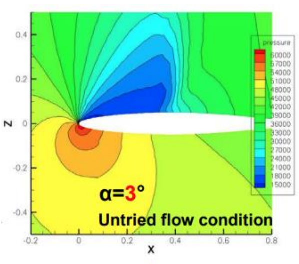
$$U(4) = a1(4)\,\sigma1 + a2(4)\,\sigma2 + a3(4)\,\sigma3$$



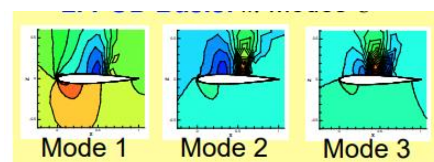




Prediction



$$U(3) = a1(3)\,\sigma1 + a2(3)\,\sigma2 + a3(3)\,\sigma3$$



적합직교분해 (POD)

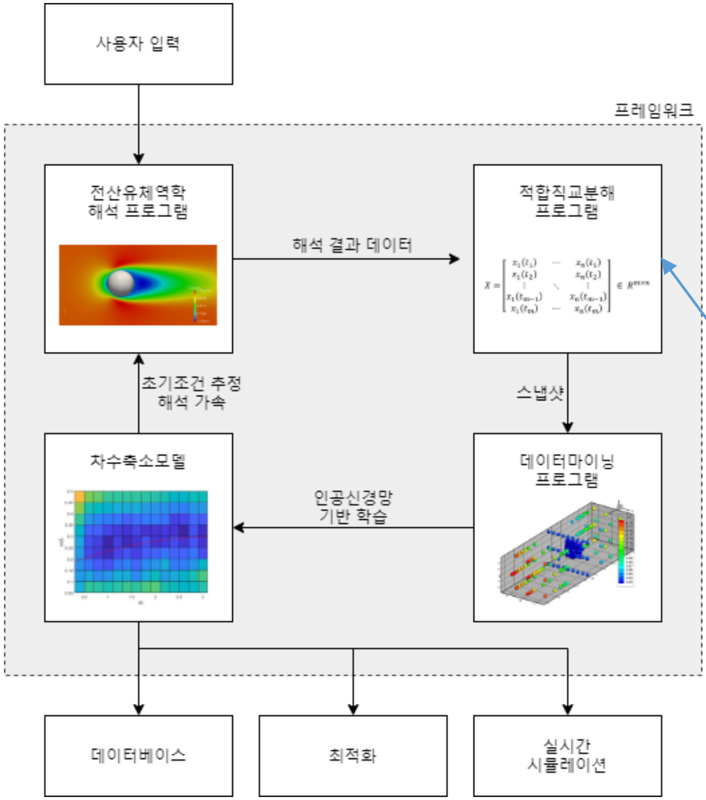
- 절차
  - (5) 유동장 재생성
    - Basis × Expansion coefficient

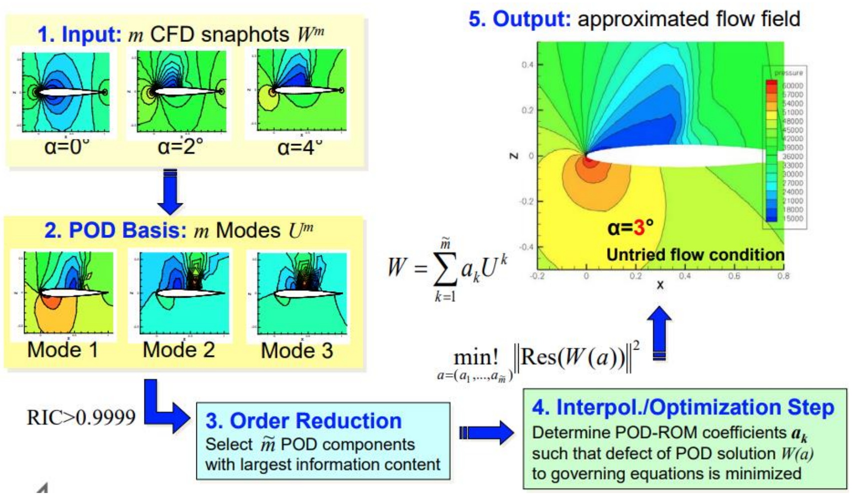
05_Reconstruction	• 보간/회귀모델로 생성된 POD's expansion coefficient과 basis function을 이용하여 Flow filed 를 재생성	$X^{new} = a_m \bullet \bar{\phi}_m$

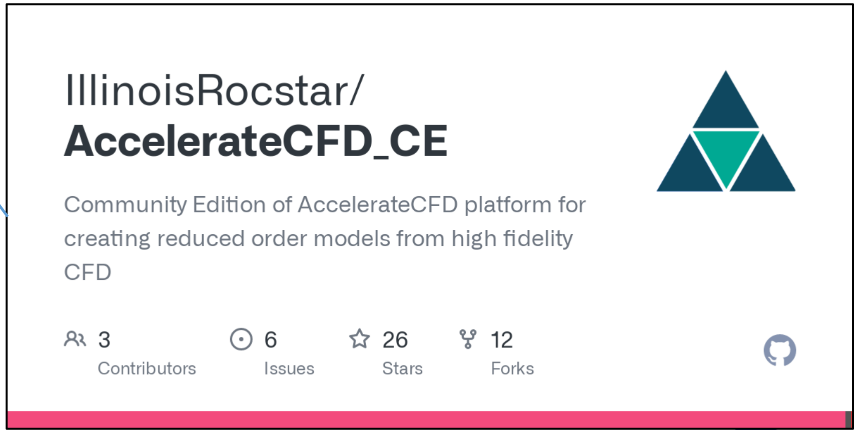
```
// this loops through all time directories in case and writes reconstructed..
// velocity (Urom)
forAll(timeDirs,timei)
{
  std::vector<double> aVect = aVals[i];
  Info << "t = " << timeDirs[timei].value() << nl; // Case progres info in terminal
  runTime.setTime(timeDirs[timei],0);

  std::string UName;
  UName = "Urom";
  volVectorField Urom(generateCustomField(runTime,mesh,UName),UMean);

  for (int i=0; i<nDim; i++)
    Urom += aVect[i]*sigs[i];

  Urom.write(); // writes Urom in every time directory
  i++;
}
```

적합직교분해 (POD)

- 요약
  - 고차원 CFD 해석 결과를 저차원 basis들의 선형결합으로 압축 가능
  - 프레임워크 구성 프로그램 선정
    - OpenFOAM
    - AccelerateCFD



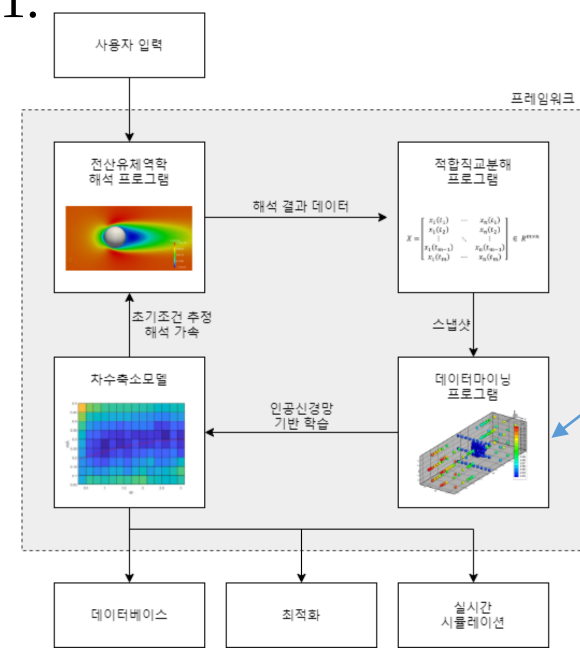




NEXTfoam open source CFD consulting

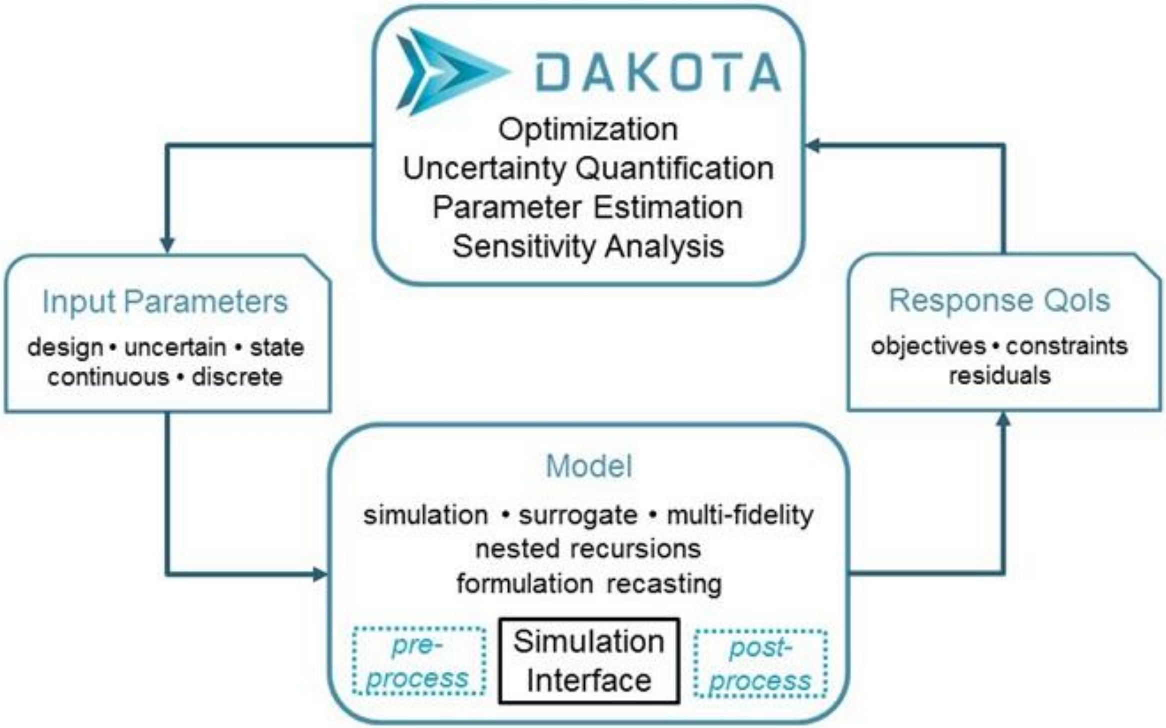
데이터마이닝 (DAKOTA toolkit)

- 설계 및 데이터마이닝 툴박스
  - 각종 기법 알고리즘 수백 가지 내장
  - 키워드 input (ex. Surrogate, kriging, ann) 으로 간단한 실행
- GNU Lesser General Public License (versions 5.0+)
- M. S. Eldred et al., "DAKOTA : a multilevel parallel object- oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis," E. United States. Department Of, Ed., ed: Sandia National Laboratories, 2011.



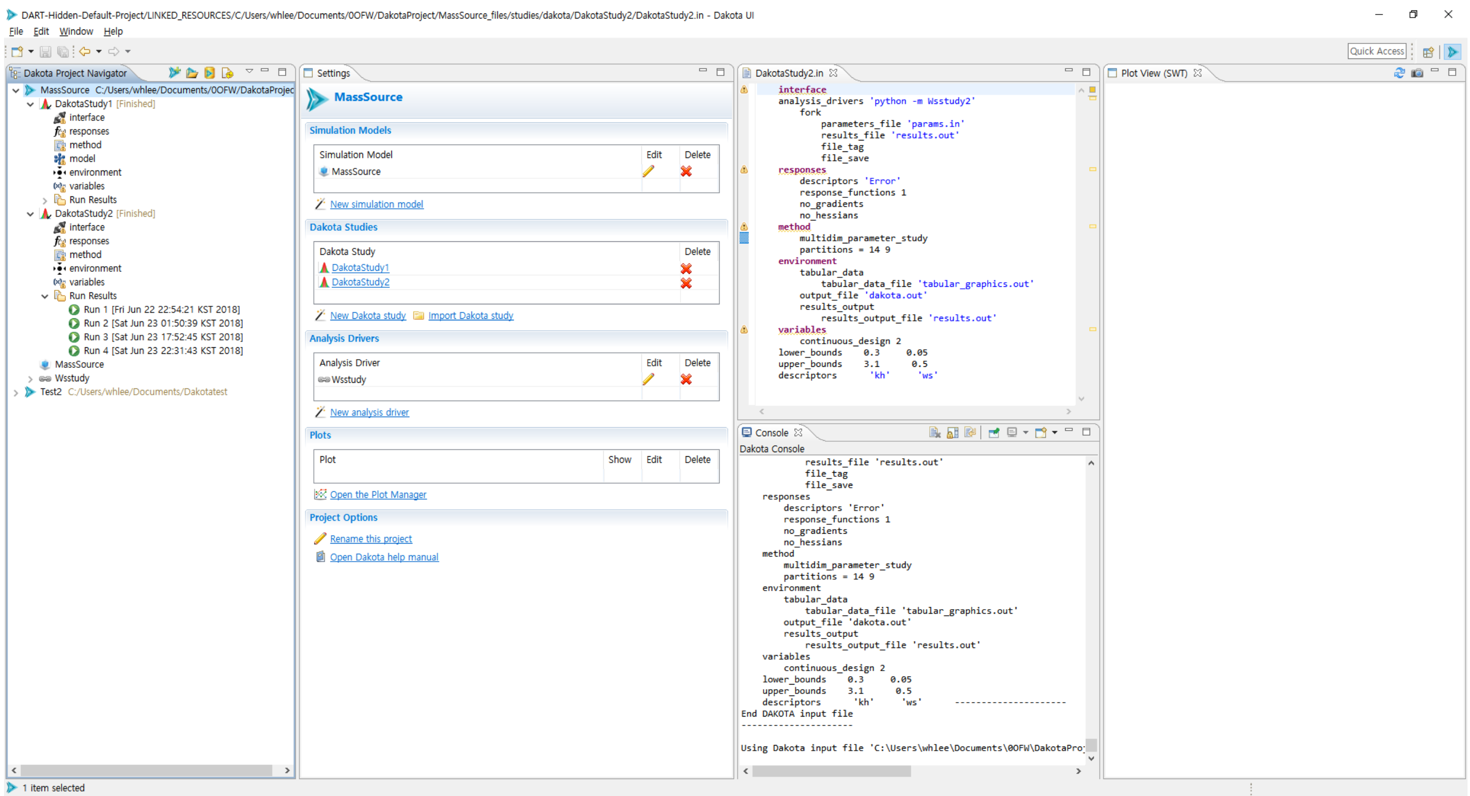
데이터마이닝 (DAKOTA toolkit)

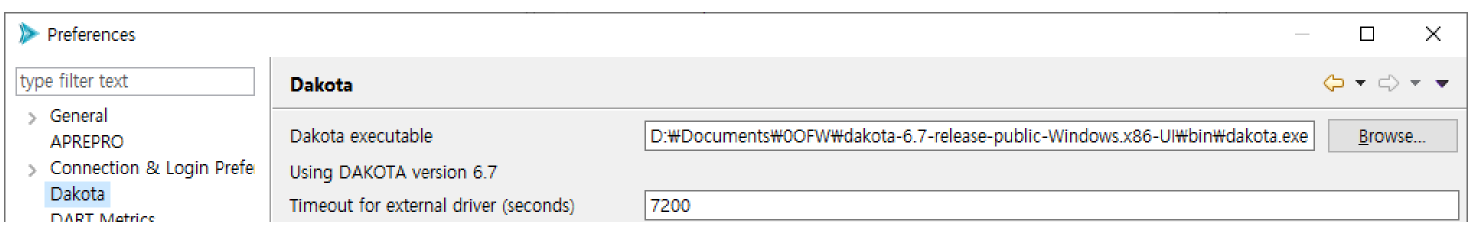
- 작동 방식
  - 실행 목적
    - 매개 변수 sweep
    - 최적화
    - 민감도 분석
    - …
  - 입력 변수 조작
    - 임의 형식 파일 대응 가능
      - Line / Column 지정
  - 솔버 실행
    - Python 인터페이스 모듈 지원
  - 출력 변수 인식
    - 임의 형식 파일 혹은 stdout 인식 가능
      - Line / Column 지정



데이터마이닝 (DAKOTA toolkit)

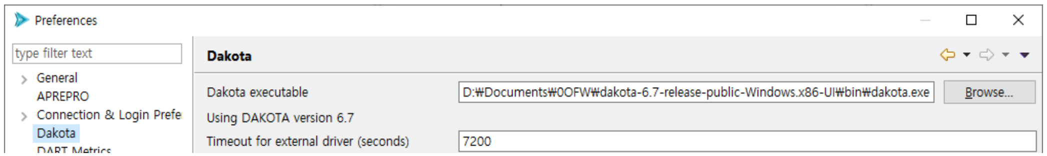
- CLI 프로그램 + GUI 지원





데이터마이닝 (DAKOTA toolkit)

- CLI 프로그램 + GUI 지원
  - GUI만으로는 구동 불가
  - Dakota Executable 연동 필요



  - Linux binary는 RHEL만 제공
  - Ubuntu 등에서 사용 시 직접 빌드 필요
    - mkdir build && cd build
    - cmake ..
      - 3.15 or higher
  - 빌드 시 Python module 제공
    - Binary 다운로드 시 제공 X


download.html

Public

Release 6.16.0 (*Release Notes*) — Released Mon, 05/

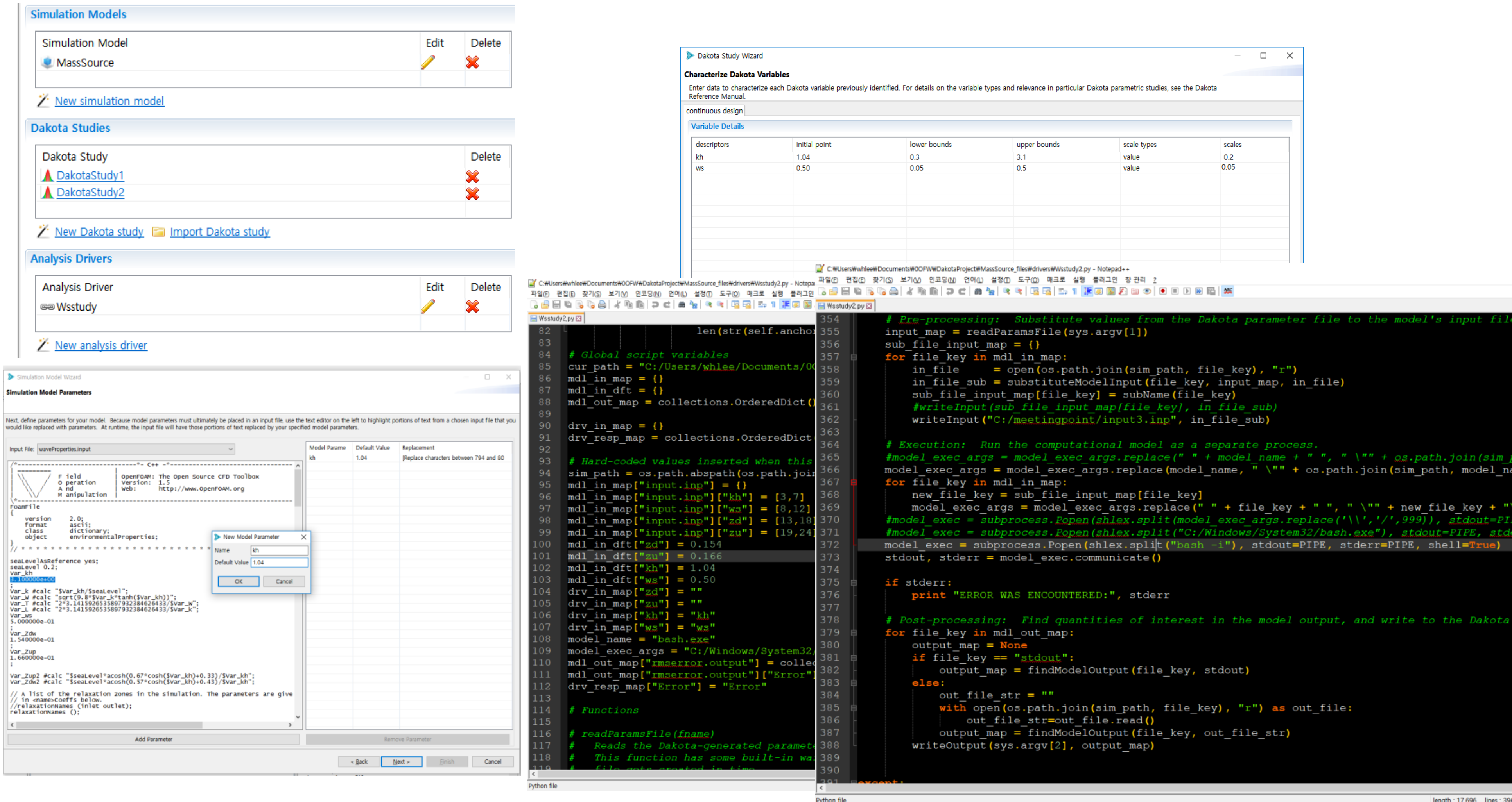
Platform	Link
Linux	dakota-6.16.0-release-public-Linux.x86_64-gui.tar.gz
Mac OS X	dakota-6.16.0-release-public-Darwin.x86_64-gui.tar.gz
Mac OS X	dakota-6.16.0-public-darwin.Darwin.x86_64-gui_cli.tar.gz
Mac OS X	dakota-6.16.0-public-darwin.Darwin.x86_64-cli.tar.gz
Windows	dakota-6.16.0-release-public-Windows.x64-gui.zip
Windows	dakota-6.16.0-public-windows.Windows.x64-gui_cli.zip
Windows	dakota-6.16.0-public-windows.Windows.x64-cli.zip
Source	dakota-6.16.0-release-public-src-gui.zip
Linux (RHEL7)	dakota-6.16.0-public-rhel7.Linux.x86_64-gui_cli.tar.gz
Linux (RHEL7)	dakota-6.16.0-public-rhel7.Linux.x86_64-cli.tar.gz
Source (Windows)	dakota-6.16.0-public-src-cli.zip
Source (Unix/OS X)	dakota-6.16.0-public-src-cli.tar.gz

Description

Dakota supports a library-linked interface to Python, but it must be explicitly enabled when compiling Dakota from source.

데이터마이닝 (DAKOTA toolkit)

- 초기 사용자 GUI 지원
  - 솔버 실행 스크립트 자동 작성 지원



데이터마이닝 (DAKOTA toolkit)

- 숙련 사용자 CLI 실행
  - 간소한 양식의 input file로 용이한 실행

rosen_opt_sbo.in

```
# Dakota Input File: rosen_opt_sbo.in

environment
  tabular_data
    tabular_data_file = 'rosen_opt_sbo.dat'
  top_method_pointer = 'SBLO'

method
  id_method = 'SBLO'
  surrogate_based_local
    model_pointer = 'SURROGATE'
    method_pointer = 'NLP'
    max_iterations = 500
  trust_region
    initial_size = 0.10
    minimum_size = 1.0e-6
    contract_threshold = 0.25
    expand_threshold   = 0.75
    contraction_factor = 0.50
    expansion_factor   = 1.50

method
  id_method = 'NLP'
  conmin_frcg
    max_iterations = 50
    convergence_tolerance = 1e-8

model
  id_model = 'SURROGATE'
  surrogate global
    correction additive zeroth_order
    polynomial quadratic
    dace_method_pointer = 'SAMPLING'
  responses_pointer = 'SURROGATE_RESP'

variables
  continuous_design = 2
    initial_point    -1.2  1.0
    lower_bounds     -2.0 -2.0
    upper_bounds      2.0  2.0
    descriptors      'x1' 'x2'
```

whlee@DESKTOP-ECSUKD1: /mnt/c/Users/whlee/Documents/code/220704_dakota_suropt_test/src

```
<<<<< global_neural_network approximation builds completed.
Beginning Approximate Fn Evaluations...
<<<<< Function evaluation summary (APPROX_INTERFACE_1): 100 total (100 new, 0 duplicate)
<<<<< Function evaluation summary (SimulationInterface): 10 total (10 new, 0 duplicate)
<<<<< Best parameters          =
                      5.1892411019e+01 x1
                      7.7085409127e-01 x2
<<<<< Best objective function  =
                     -7.3934829807e+01
<<<<< Best evaluation ID: 37
-----------------------------------------------------------------------------
Statistics based on 100 samples:

Sample moment statistics for each response function:
                            Mean           Std Dev          Skewness          Kurtosis
        obj_fn  2.4296102525e+03  1.9546250171e+03  5.0011856105e-01 -8.9182874568e-01

95% confidence intervals for each response function:
                     LowerCI_Mean      UpperCI_Mean    LowerCI_StdDev    UpperCI_StdDev
        obj_fn  2.0417702432e+03  2.8174502618e+03  1.7161741452e+03  2.2706395157e+03

Simple Correlation Matrix among all inputs and outputs:
                     x1           x2       obj_fn
          x1  1.00000e+00
          x2  2.08607e-02  1.00000e+00
      obj_fn  2.23349e-02  8.15550e-01  1.00000e+00

Partial Correlation Matrix between input and output:
                 obj_fn
          x1  9.19860e-03
          x2  8.15465e-01

Simple Rank Correlation Matrix among all inputs and outputs:
                     x1           x2       obj_fn
          x1  1.00000e+00
          x2  2.26463e-02  1.00000e+00
      obj_fn  3.69757e-02  8.19694e-01  1.00000e+00

Partial Rank Correlation Matrix between input and output:
```


NEXTfoam open source CFD consulting

데이터마이닝 (DAKOTA toolkit)

- DAKOTA input 예시 (parametric study)
  - $ dakota DakotaStudy.in

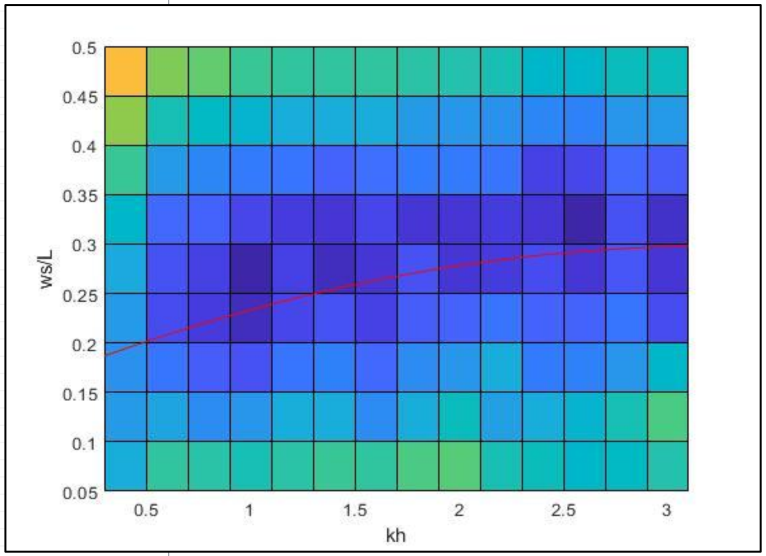
DakotaStudy2.in

```
interface
analysis_drivers 'python -m Wsstudy2'
    fork
        parameters_file 'params.in'
        results_file 'results.out'
        file_tag
        file_save
responses
    descriptors 'Error'
    response_functions 1
    no_gradients
    no_hessians
method
    multidim_parameter_study
    partitions = 14 9
environment
    tabular_data
        tabular_data_file 'tabular_graphics.out'
    output_file 'dakota.out'
    results_output
        results_output_file 'results.out'
variables
    continuous_design 2
lower_bounds    0.3     0.05
upper_bounds    3.1      0.5
descriptors      'kh'     'ws'
```

Tabular Data

Plot a single trace from this data

#	kh	ws	Error
1	0.3	0.05	0.1177772
2	0.5	0.05	0.1539656
3	0.7	0.05	0.1518281
4	0.9	0.05	0.1438151
5	1.1	0.05	0.1490063
6	1.3	0.05	0.1595631
7	1.5	0.05	0.1532637
8	1.7	0.05	0.1649633
9	1.9	0.05	0.1696581
10	2.1	0.05	0.1417427
11	2.3	0.05	0.1379622
12	2.5	0.05	0.1318971
13	2.7	0.05	0.1356013
14	2.9	0.05	0.144816
15	3.1	0.05	0.164606
16	0.3	0.1	0.1027305
17	0.5	0.1	0.1106198
18	0.7	0.1	0.09194064
19	0.9	0.1	0.09729371
20	1.1	0.1	0.1176898
21	1.3	0.1	0.1174136
22	1.5	0.1	0.09094501
23	1.7	0.1	0.1162946
24	1.9	0.1	0.1371688
25	2.1	0.1	0.1063633
26	2.3	0.1	0.1166866
27	2.5	0.1	0.1270368
28	2.7	0.1	0.1425002
29	2.9	0.1	0.1641292
30	3.1	0.1	0.2036203
31	0.3	0.15	0.09396669
32	0.5	0.15	0.07343904
33	0.7	0.15	0.06004792
34	0.9	0.15	0.05047691
35	1.1	0.15	0.07353609
36	1.3	0.15	0.08284247
37	1.5	0.15	0.06530631
38	1.7	0.15	0.09023968
39	1.9	0.15	0.09920157
40	2.1	0.15	0.1187019
41	2.3	0.15	0.0786704
42	2.5	0.15	0.08205784
43	2.7	0.15	0.09739424



NEXTfoam open source CFD consulting

데이터마이닝 (DAKOTA toolkit)

- DAKOTA input 예시 (최적화)
  - $ dakota DakotaStudy.in

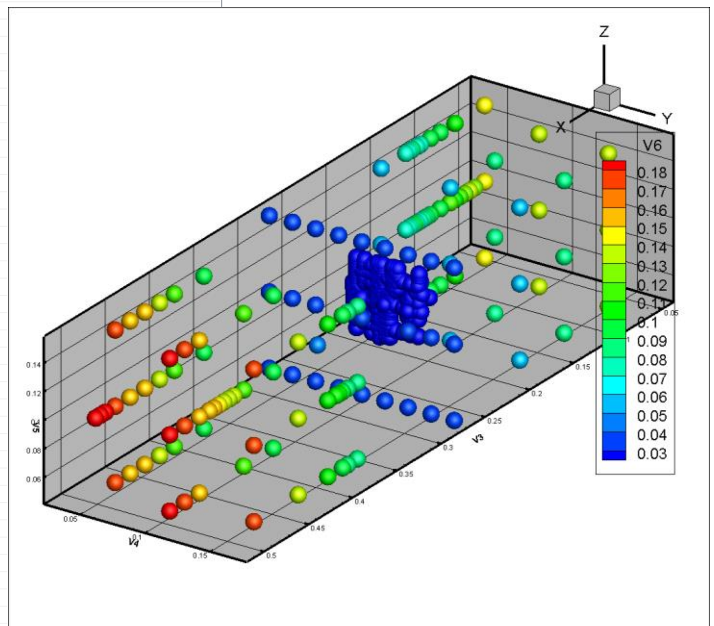
DakotaStudy3.in

```
interface
    analysis_drivers 'python -m Wsstudy3'
        fork
            parameters_file 'params.in'
            results_file 'results.out'
            file_tag
            file_save
responses
    descriptors 'Error'
    objective_functions 1
        sense 'min'
    numerical_gradients
    numerical_hessians
method
coliny_direct
    max_function_evaluations 999999
model
    single
environment
    tabular_data
        tabular_data_file 'tabular_graphics.out'
    output_file 'dakota.out'
    results_output
        results_output_file 'results.out'
variables
    continuous_design 3
        initial_point 0.30 0.154 0.166
    lower_bounds    0.05  0.02 0.02
    upper_bounds    0.5   0.18 0.18
        descriptors 'ws'  'zd'  'zu'
```

Tabular Data

Plot a single trace from this data

#	kh	ws	Error
1	0.3	0.05	0.1177772
2	0.5	0.05	0.1539656
3	0.7	0.05	0.1518281
4	0.9	0.05	0.1438151
5	1.1	0.05	0.1490063
6	1.3	0.05	0.1595631
7	1.5	0.05	0.1532637
8	1.7	0.05	0.1649633
9	1.9	0.05	0.1696581
10	2.1	0.05	0.1417427
11	2.3	0.05	0.1379622
12	2.5	0.05	0.1318971
13	2.7	0.05	0.1356013
14	2.9	0.05	0.144816
15	3.1	0.05	0.164606
16	0.3	0.1	0.1027305
17	0.5	0.1	0.1106198
18	0.7	0.1	0.09194064
19	0.9	0.1	0.09729371
20	1.1	0.1	0.1176898
21	1.3	0.1	0.1174136
22	1.5	0.1	0.09094501
23	1.7	0.1	0.1162946
24	1.9	0.1	0.1371688
25	2.1	0.1	0.1063633
26	2.3	0.1	0.1166866
27	2.5	0.1	0.1270368
28	2.7	0.1	0.1425002
29	2.9	0.1	0.1641292
30	3.1	0.1	0.2036203
31	0.3	0.15	0.09396669
32	0.5	0.15	0.07343904
33	0.7	0.15	0.06004792
34	0.9	0.15	0.05047691
35	1.1	0.15	0.07353609
36	1.3	0.15	0.08284247
37	1.5	0.15	0.06530631
38	1.7	0.15	0.09023968
39	1.9	0.15	0.09920157
40	2.1	0.15	0.1187019
41	2.3	0.15	0.0786704
42	2.5	0.15	0.08205784
43	2.7	0.15	0.09739424



데이터마이닝 (DAKOTA toolkit)

- DAKOTA input 예시 (대체모델 기반 최적화)

D:₩Documents₩OOFW₩dakota-6.7-release-public-Windows.x86-UI₩examples₩users₩rosen_opt_sbo.in - Notepad++

파일(F) 편집(E) 찾기(S) 보기(V) 인코딩(N) 언어(L) 설정(T) 도구(O) 매크로 실행 플러그인 창 관리 ?

rosen_opt_sbo.in

```
# Dakota Input File: rosen_opt_sbo.in

environment
  tabular_data
    tabular_data_file = 'rosen_opt_sbo.dat'
  top_method_pointer = 'SBLO'

method
  id_method = 'SBLO'
  surrogate_based_local
    model_pointer = 'SURROGATE'
    method_pointer = 'NLP'
    max_iterations = 500
  trust_region
    initial_size = 0.10
    minimum_size = 1.0e-6
    contract_threshold = 0.25
    expand_threshold   = 0.75
    contraction_factor = 0.50
    expansion_factor   = 1.50

method
  id_method = 'NLP'
  conmin_frcg
    max_iterations = 50
    convergence_tolerance = 1e-8

model
  id_model = 'SURROGATE'
  surrogate global
    correction additive zeroth_order
    polynomial quadratic
    dace_method_pointer = 'SAMPLING'
  responses_pointer = 'SURROGATE_RESP'

variables
  continuous_design = 2
    initial_point   -1.2  1.0
    lower_bounds    -2.0 -2.0
    upper_bounds     2.0  2.0
    descriptors      'x1' 'x2'

responses
  id_responses = 'SURROGATE_RESP'
  objective_functions = 1
  numerical_gradients
    method_source dakota
    interval_type central
    fd_step_size = 1.e-6
  no_hessians

method
  id_method = 'SAMPLING'
  sampling
    samples = 10
    seed = 531
    sample_type lhs
    model_pointer = 'TRUTH'

model
  id_model = 'TRUTH'
  single
    interface_pointer = 'TRUE_FN'
    responses_pointer = 'TRUE_RESP'

interface
  id_interface = 'TRUE_FN'
  analysis_drivers = 'rosenbrock'
    direct
  deactivate evaluation_cache restart_file

responses
  id_responses = 'TRUE_RESP'
  objective_functions = 1
  no_gradients
  no_hessians
```

```
experimental_gaussian_process
experimental_polynomial
function_train
gaussian_process
mars
moving_least_squares
neural_network
polynomial
radial_basis
```

대체모델
생성 기법

Normal text file   length : 1,617   lines : 77   Ln : 77   Col : 1   Pos : 1,618   Windows (CR LF)   UTF-8   INS

데이터마이닝 (DAKOTA toolkit)

- DAKOTA input 예시 (Multi-fidelity surrogate model)

C:₩Users₩whlee₩Documents₩code₩220215_dakota610₩dakota-6.10.0-release-public-Windows.x86-UI₩share₩dakota₩examples₩advanced₩rosenbrock_sbo_hierarchical.in - Notepad++

rosenbrock_sbo_hierarchical.in

```

environment,
## Activate Dakota's legacy X Windows-based graphics on Unix systems
## Consider newer capabilities in the Dakota GUI
    graphics
    tabular_data
    top_method_pointer = 'SBLO'

method,
    id_method = 'SBLO'
    surrogate_based_local
    model_pointer = 'SURROGATE'
    method_pointer = 'NLP'
    trust_region
      initial_size = 0.10
      contract_threshold = 0.25
      expand_threshold = 0.75
      contraction_factor = 0.50
      expansion_factor   = 1.50

method,
    id_method = 'NLP'
## (NPSOL requires a software license; if not available, try
## conmin_frcg or optpp_newton instead)
    npsol_sqp
      max_iterations = 50
      convergence_tolerance = 1e-10

model,
    id_model = 'SURROGATE'
    surrogate hierarchical
      ordered_model_fidelities = 'LOFI' 'HIFI'
      correction additive       second_order

variables,
    continuous_design = 2
      initial_point   -1.2      1.0
      lower_bounds    -2.0     -2.0
      upper_bounds     2.0  2.0
      descriptors      'x1'     'x2'

responses,
    objective_functions = 1
    analytic_gradients
    analytic_hessians

model,
    id_model = 'LOFI'
    single
      interface_pointer = 'LOFI_FN'

interface,
    id_interface = 'LOFI_FN'
  analysis_drivers = 'lf_rosenbrock'
    direct
  deactivate restart_file

model,
    id_model = 'HIFI'
    single
      interface_pointer = 'HIFI_FN'

interface,
    id_interface = 'HIFI_FN'
  analysis_drivers = 'rosenbrock'
    direct
  deactivate restart_file

```

Normal text file   length : 1,443   lines : 68   Ln : 68   Col : 1   Pos : 1,444   Windows (CR LF)   UTF-8   INS

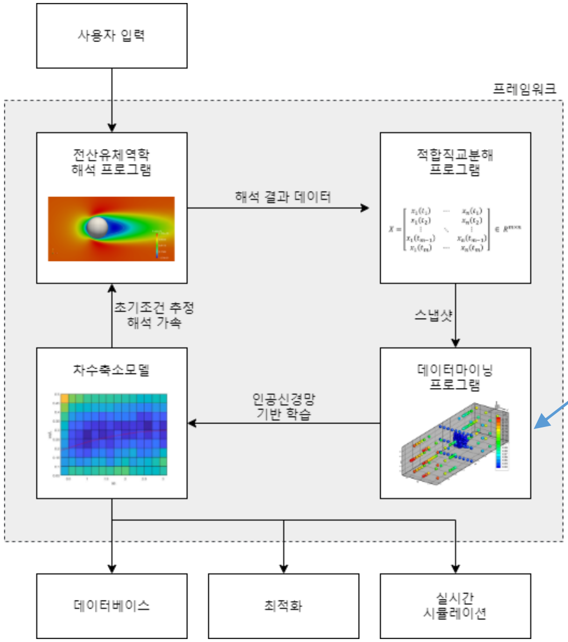
데이터마이닝 (DAKOTA toolkit)

- DAKOTA 스크립트 예시
  - dakota.interfacing 모듈을 통해 연동
    - PYTHONPATH 환경변수에 Dakota 설치 경로 추가
  - 솔버 실행 / 입출력 순서 임의 구성 가능

```python
#!/usr/bin/env python3
import numpy as np
import dakota.interfacing as di

params, results = di.read_parameters_file()

write_input(params)

run_solver()

results = read_output()

results.write()
```

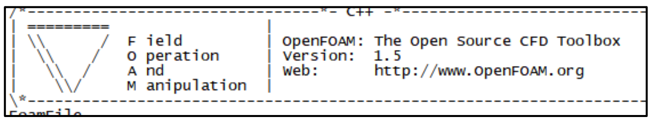
데이터마이닝 (DAKOTA toolkit)

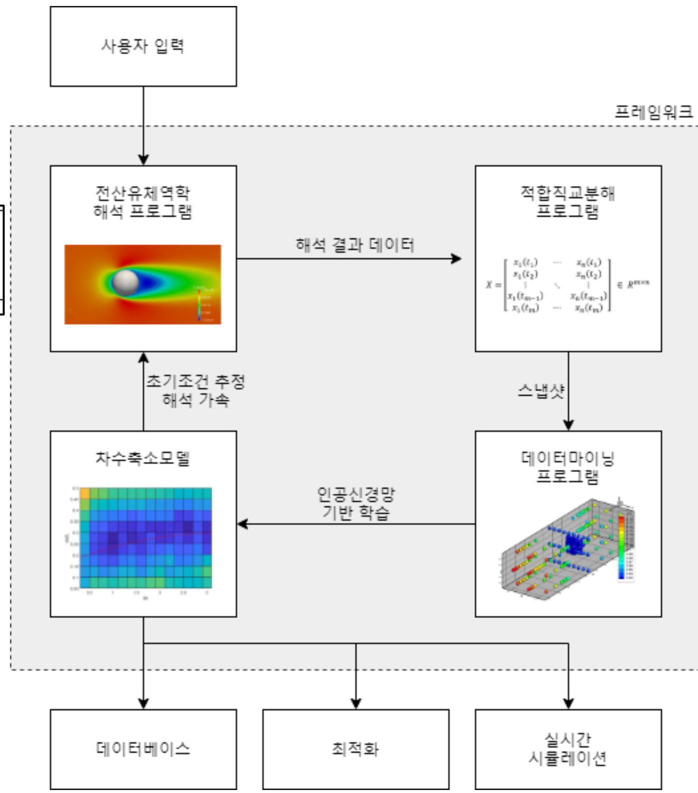
- 요약
  - 수백개의 기법을 간단한 input 파일 작성으로 사용 가능한 데이터마이닝 툴박스 DAKOTA
  - 프레임워크 구성 요소 프로그램으로 선정

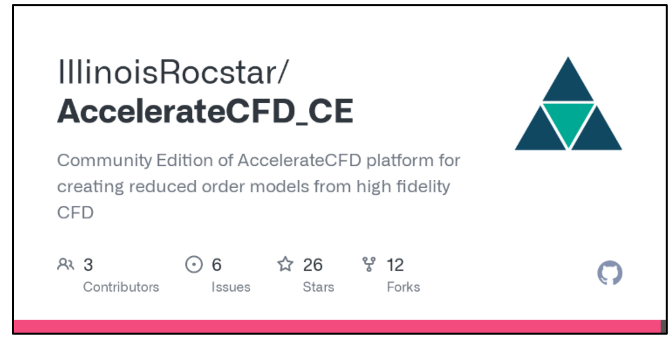


프레임워크 개발

- 구성 요소 프로그램
  - OpenFOAM
  - AccelerateCFD
  - DAKOTA



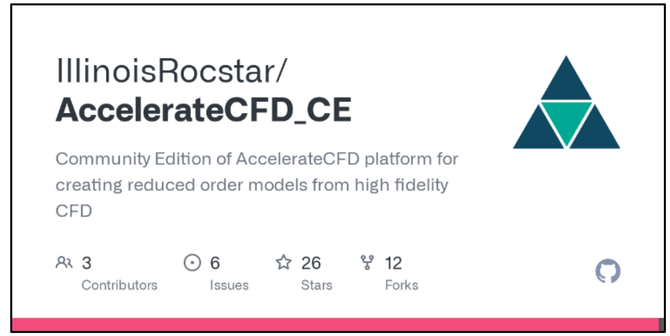


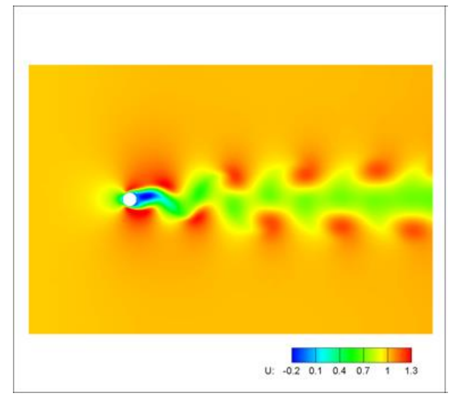


NEXTfoam open source CFD consulting

프레임워크 개발

- AcceleratedCFD 보완
  - 기존
    - Umean + U(t)
    - Vector U(t) 의 시간에 따른 POD 추정 용도
  - 개량
    - Umean 등 시간 평균 물리량 없이 POD를 수행하도록 수정
    - Scalar P에 대한 POD 추가
    - podROM 이후
    신규 입력조건에 대한 expansion coefficient를
    추가 행으로 직접 기록하는 절차 추가
    - => podFlowReconstruct에서 신규 유동장 생성
    - Expansion coefficient 보간에 DAKOTA 사용

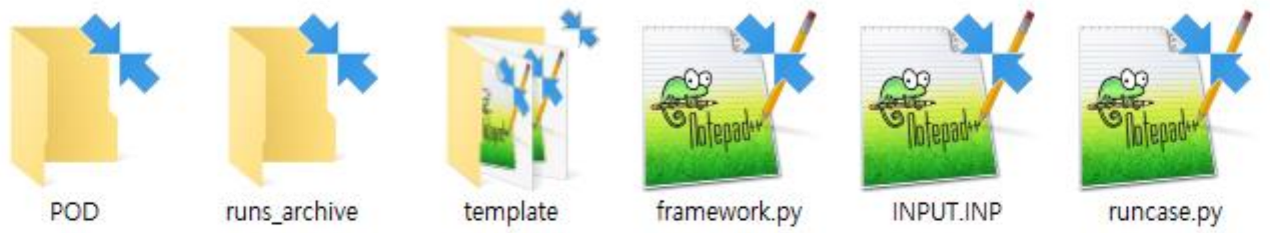





```
12.5991600000000012,-0.0693057512687179,-0.1339006029651310,-0.0207532644376687,-0.0130499781112034,0.0011800165798592,
12.7491500000000002,0.0459280961045100,-0.1440396562984638,-0.0128330907700062,0.0223013963593660,-0.0008483570761670,
12.8991400000000009,0.1332179382399867,-0.0683375036788400,0.0221005212119751,0.0128224325816151,-0.0002058570163162,
13.0491300000000017,0.1390199634468887,0.0469109259700145,0.0115742263361113,-0.0222412342683103,0.0010644760384889,
13.1991200000000006,0.0599394305805869,0.1311692048711383,-0.0224389413060522,-0.0109656204400389,-0.0013391637849161,
13.3491100000000014,-0.0554414918077339,0.1330694382196505,-0.0095945744277698,0.0225072508508146,0.0009611004502626,
13.4991000000000003,-0.1366694682455949,0.0512435328522021,0.0232928128399615,0.0084756140662863,-0.0002845926286241,
13.6490900000000011,-0.1338251272003634,-0.0641111796107720,0.0075689723628032,-0.0232688799541422,-0.0010006680928169,
13.7990800000000018,-0.0489346767104756,-0.1419509074480909,-0.0237595245725301,-0.0058962798581446,0.0011893254758908,
13.9490700000000007,0.0658592268507926,-0.1350081499591214,-0.0056720592052062,0.0250283322828671,-0.0012461203150463,
14.0990600000000015,0.1405168875164147,-0.0477514676189012,0.0246149142836608,0.0055390426245586,0.0004149955424102,
14.2490500000000004,0.1292046263280497,0.0664293679184010,0.0043421317940559,-0.0246470606172846,0.0005651070976550,
14.3990400000000012,0.0391072649817660,0.1377133430606408,-0.0244819198335083,-0.0037042434384433,-0.0012108524841335,
14.5490300000000019,-0.0745809908107926,0.1226416020205486,-0.0022799283069106,0.0242403232407572,0.0012326165070807,
14.6990200000000009,-0.1423793349210730,0.0301806476744027,0.0246748151990811,0.0012395202013030,-0.0008729662446150,
14.8490100000000016,-0.1226078935902394,-0.0828002942144563,0.0002771795817250,-0.0243053187103875,-0.0004372786646340,
14.9990000000000006,-0.0277564215228415,-0.1468467960749114,-0.0245856912556688,0.0014876802015931,0.0009411264980396,
```

프레임워크 개발

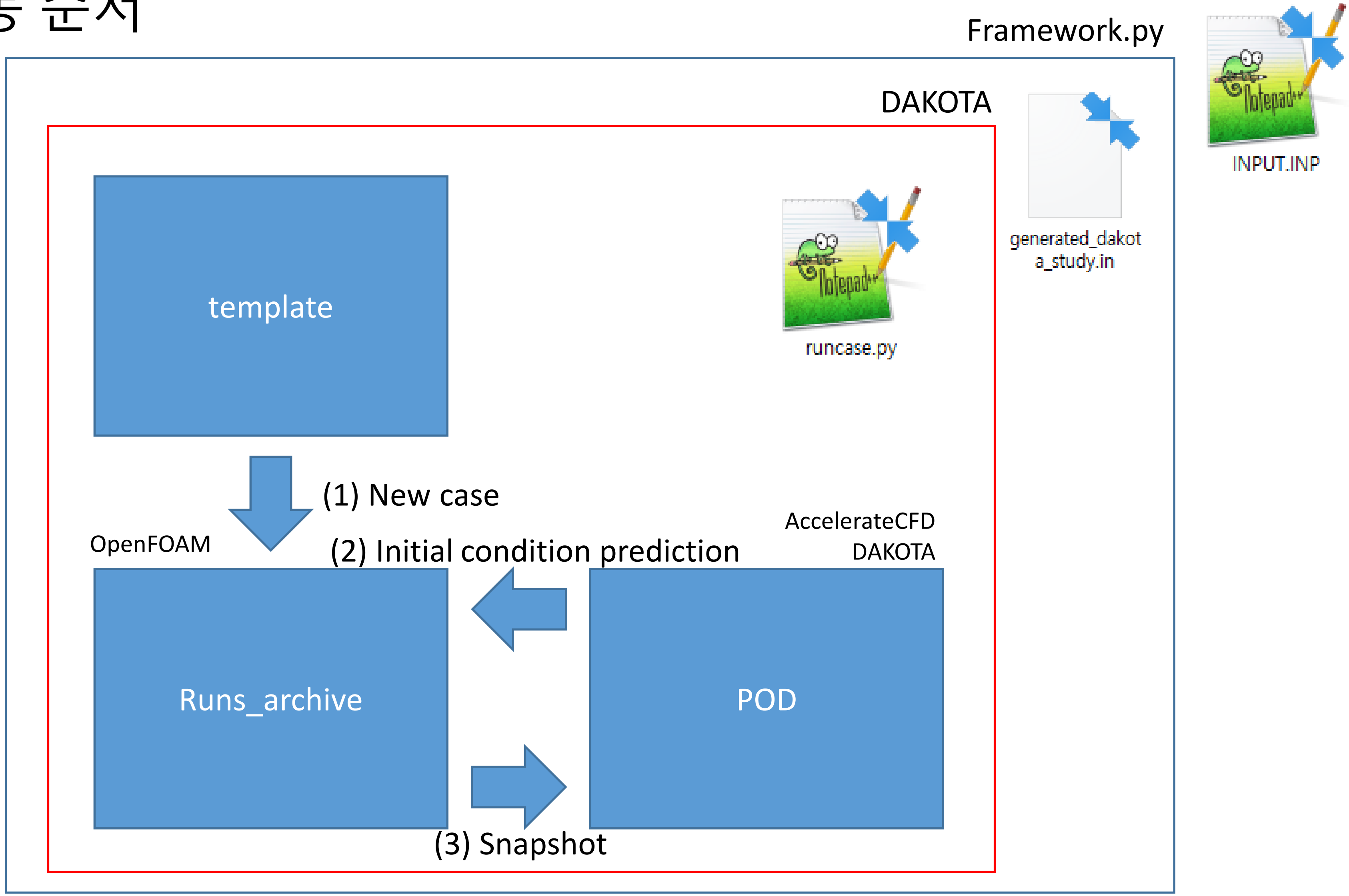
- 구성



  - Template : 해석 케이스 템플릿
  - Runs_archive : 입력 조건 별 해석이 진행/완료된 케이스 폴더 모음
  - POD : 해석 결과를 취합하여 AccelerateCFD로 POD를 수행하는 폴더
  - Runcase.py : POD로 신규 케이스 초기조건 생성 및 해석 수행
  - Framework.py : runcase.py를 driver script로 하여 DAKOTA 실행
  - INPUT.INP : framework.py의 input으로 프레임워크 실행 목적/조건 지정

프레임워크 개발

- 작동 순서



프레임워크 개발

- $ python3 framework.py
  - Template 폴더 내 input 파일 경로, input 개수, 위치, 범위
  - 실행 명령어
  - 출력 파일 경로 및 출력값 위치
  - 실행 목적 (DB 생성 / 최적화)

framework.py

INPUT.INP

INPUT.INP

```
INPUT_FILE_LOCATION
INPUT.INP

#_OF_INPUT_VARIABLE
2

LOCATION_OF_INPUT_VARIABLE (Line / column)
7 1
8 1

BOUND
-20 20
-20 20

RUN_COMMAND
run.sh

OUTPUT_FILE_LOCATION
postProcessing/forces/0/forceCoeffs.dat

LOCATION_OF_OUTPUT_VARIABLE (Line / column)
1009 1

OBJECTIVE (1: DB, 2: OPTIMIZATION)
1
```

rosen_opt_sbo.in

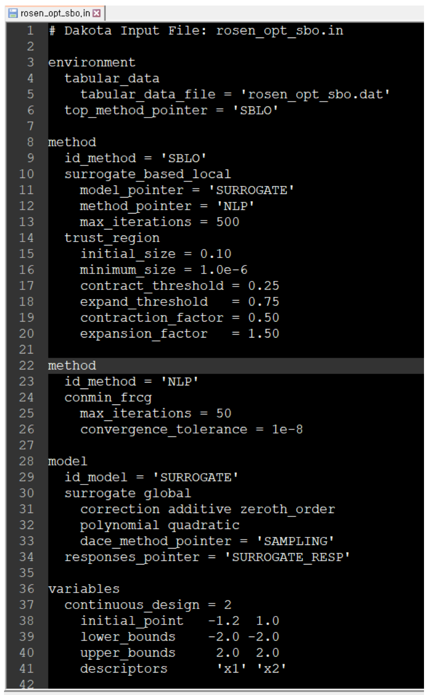
```
# Dakota Input File: rosen_opt_sbo.in

environment
  tabular_data
    tabular_data_file = 'rosen_opt_sbo.dat'
  top_method_pointer = 'SBLO'

method
  id_method = 'SBLO'
  surrogate_based_local
    model_pointer = 'SURROGATE'
    method_pointer = 'NLP'
    max_iterations = 500
  trust_region
    initial_size = 0.10
    minimum_size = 1.0e-6
    contract_threshold = 0.25
    expand_threshold   = 0.75
    contraction_factor = 0.50
    expansion_factor   = 1.50

method
  id_method = 'NLP'
  conmin_frcg
    max_iterations = 50
    convergence_tolerance = 1e-8

model
  id_model = 'SURROGATE'
  surrogate global
    correction additive zeroth_order
    polynomial quadratic
    dace_method_pointer = 'SAMPLING'
  responses_pointer = 'SURROGATE_RESP'

variables
  continuous_design = 2
    initial_point    -1.2  1.0
    lower_bounds     -2.0 -2.0
    upper_bounds      2.0  2.0
    descriptors      'x1' 'x2'
```

Normal text file

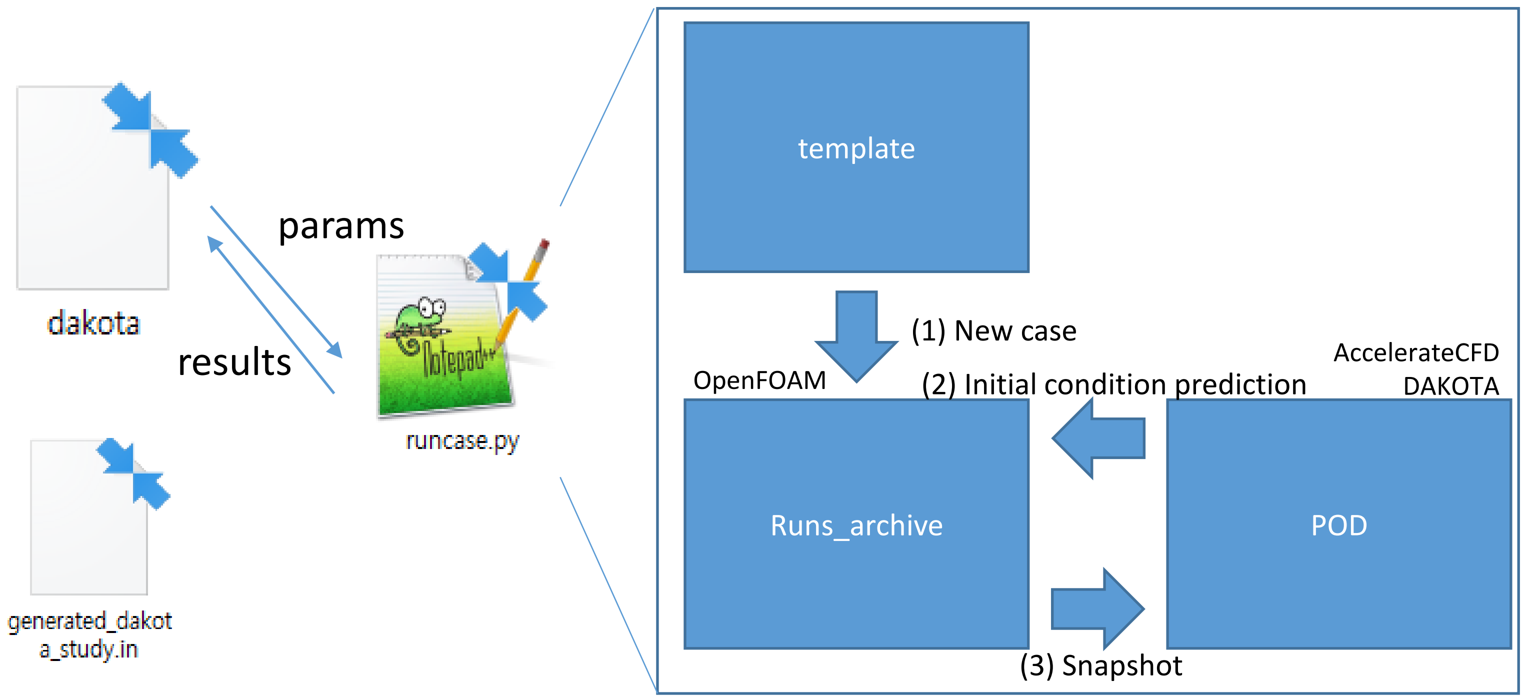
generated_dakota_study.in

프레임워크 개발

- (하위 프로세스 실행) $ dakota generated_dakota_study.in
  - dakota.interfacing 모듈을 통해 runcase.py와 입출력 교환
  - DAKOTA가 시도할 입력 매개 변수를 전달하면 runcase.py가 솔버 실행
  - 해석이 완료되면 runcase.py가 결과값을 읽어 DAKOTA에 전달
  - 입출력을 반복하며 DAKOTA의 실행 목적 (DB/최적화) 달성


rosen_opt_sbo.in

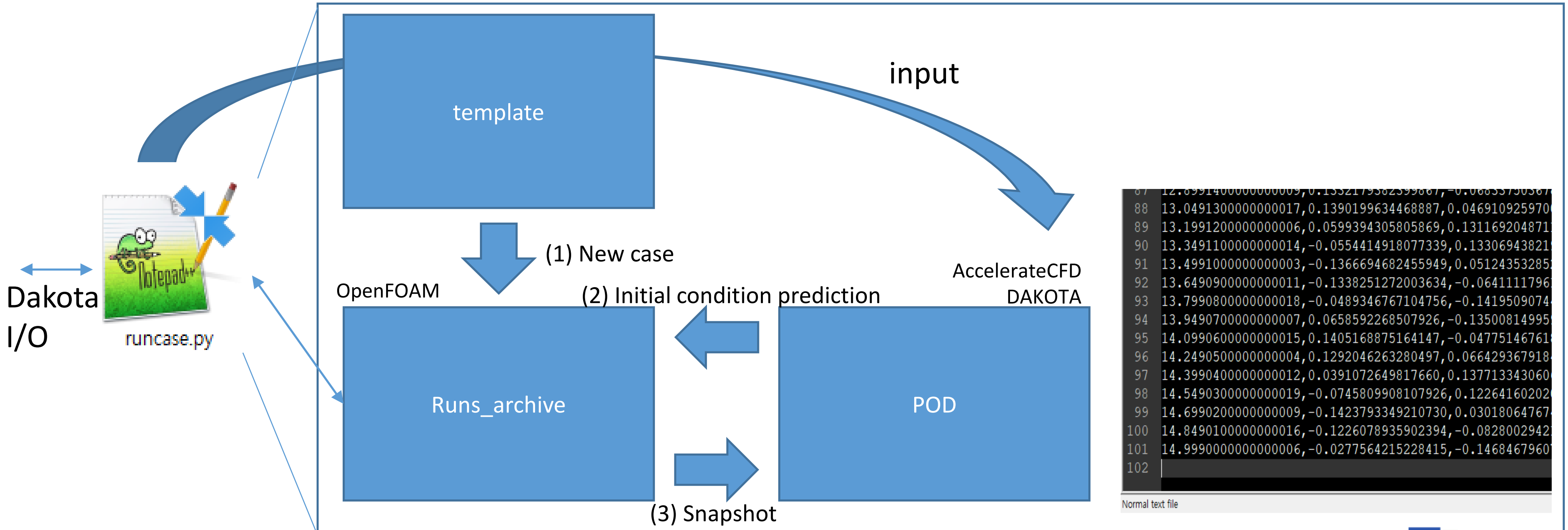
```
# Dakota Input File: rosen_opt_sbo.in

environment
  tabular_data
    tabular_data_file = 'rosen_opt_sbo.dat'
  top_method_pointer = 'SBLO'

method
  id_method = 'SBLO'
  surrogate_based_local
    model_pointer = 'SURROGATE'
    method_pointer = 'NLP'
    max_iterations = 500
  trust_region
    initial_size = 0.10
    minimum_size = 1.0e-6
    contract_threshold = 0.25
    expand_threshold   = 0.75
    contraction_factor = 0.50
    expansion_factor   = 1.50

method
  id_method = 'NLP'
  conmin_frcg
    max_iterations = 50
    convergence_tolerance = 1e-8

model
  id_model = 'SURROGATE'
  surrogate global
    correction additive zeroth_order
    polynomial quadratic
    dace_method_pointer = 'SAMPLING'
  responses_pointer = 'SURROGATE_RESP'

variables
  continuous_design = 2
    initial_point   -1.2  1.0
    lower_bounds    -2.0 -2.0
    upper_bounds     2.0  2.0
    descriptors      'x1' 'x2'

```
Normal text file



NEXTfoam open source CFD consulting

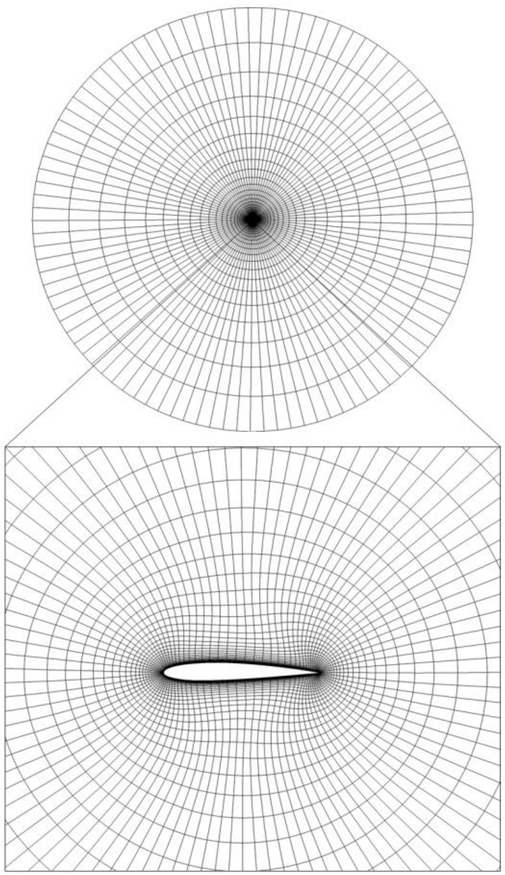
프레임워크 개발

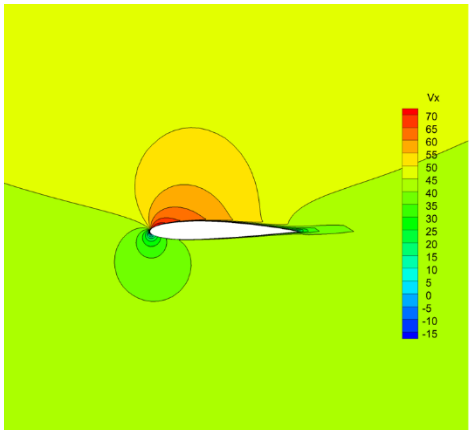
- (하위 프로세스 반복 실행) $ python3 runcase.py
  - Template 폴더를 복제하여 runs_archive에 신규 케이스 생성
  - POD 폴더에 input 매개변수를 전달 및 DAKOTA를 실행
  - Expansion coefficient를 DAKOTA로 보간하여 해석 초기조건 예측
  - 예측된 초기조건을 runs_archive로 복사하고 그대로 해석하여 빠른 수렴
  - Runcase.py에 결과값 전달 및 POD snapshot 추가

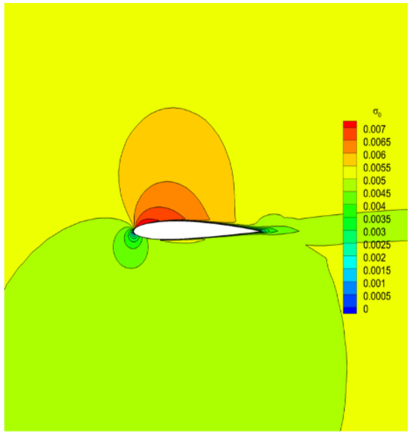


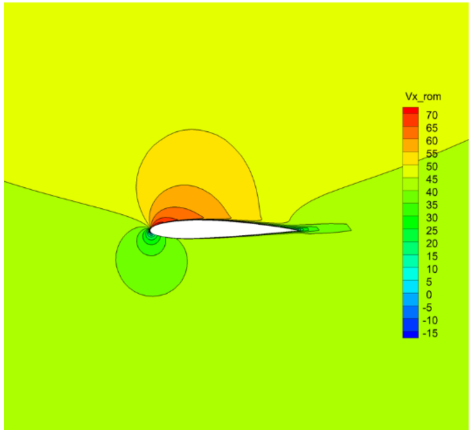
프레임워크 개발

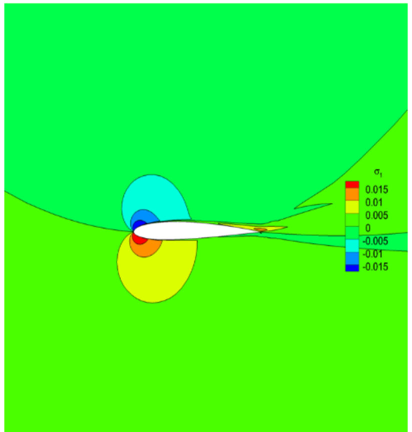
- 정확도 검증
  - NACA2412 익형 아음속 정상 유동 해석
    - 입력 변수 : 자유류 받음각, 속도
    - 5개 스냅샷 데이터로부터 신규 조건에 대한 해석 결과 추정
    - 해석 격자, CFD / POD+ANN 비교, POD 모드 (2개 모드에 에너지 99.999% 분포)











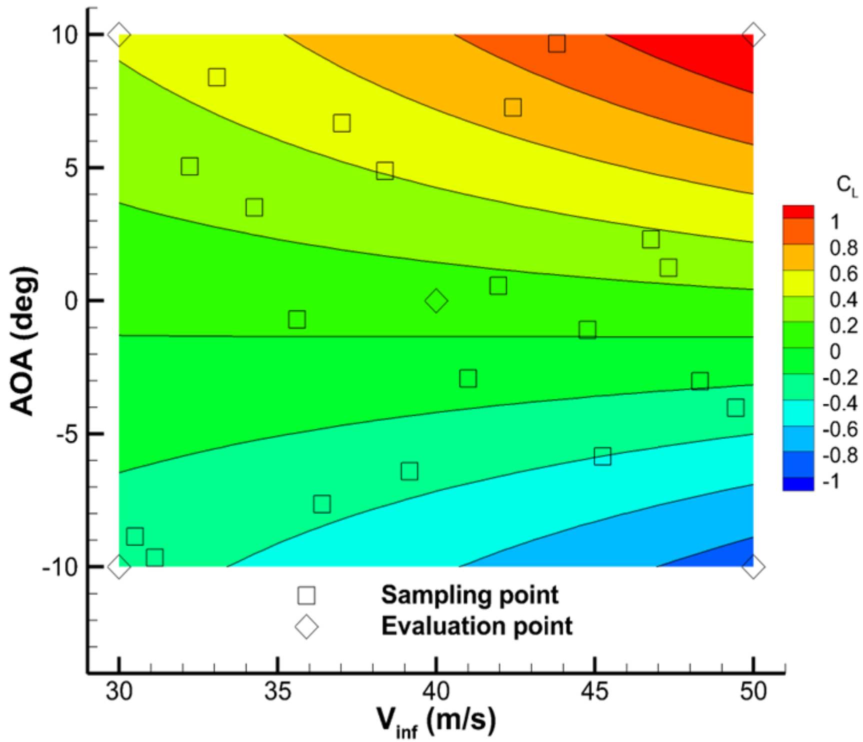
프레임워크 개발

- 정확도 검증
  - NACA2412 익형 아음속 정상 유동 해석
    - POD 수행 과정 포함 시 소요 시간 5% 이내
    - 기 존재하는 POD 모드로부터 보간만 수행할 경우 추가 단축 가능

Case No.	Method	V_∞ (m/s)	Angle of attack (deg)	Expansion coefficient		Calculation Time (sec)	Error (%)
1	CFD	30	5	5864.55	-20.0794		
2	CFD	40	0	7792.01	654.545		
3	CFD	40	4	7818.70	109.646	5.60	-
4	CFD	40	10	7787.36	-707.902		
5	CFD	50	5	9774.26	-33.4612		
6	POD + ANN	45	7	8791.20	-302.488	0.27	4.98

프레임워크 개발

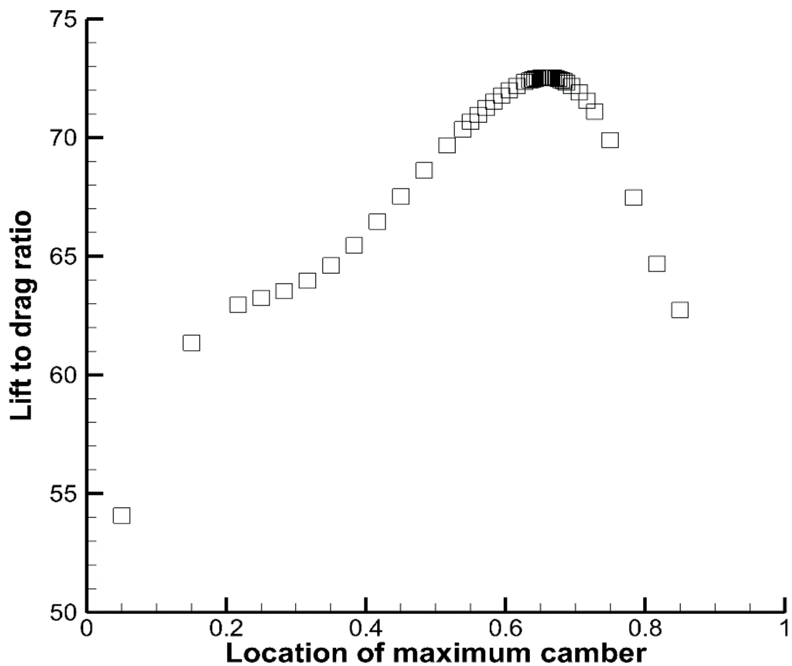
- 데이터베이스 생성 성능 검증
  - NACA2412 익형 아음속 정상 유동 해석
    - 자유류 속도 30m/s ~ 50m/s
    - 받음각 -10º ~ 10º
    - 20개의 입력 매개 변수 표본 사용
    - 5개 해석 결과 누적 시 POD 수행, 이후 해석 결과 초기 조건 추정
    - 소요 시간 : 1개 케이스 해석 소요 시간의 6.55배



Case No.	V_∞ (m/s)	AOA (deg)	Lift coefficient		Error (%)
			CFD	ANN	
1	30	-10	-0.321	-0.321	0.005
2	30	10	0.433	0.433	0.008
3	40	0	0.0965	0.0965	0.022
4	50	-10	-0.900	-0.908	0.878
5	50	10	1.220	1.211	0.790

프레임워크 개발

- 최적화 성능 검증
  - NACA2412 익형 아음속 정상 유동 해석
    - 자유류 속도 30m/s, 받음각 8°
    - 양항비가 최대가 되는 최대 캠버 위치 탐색 (모핑된 격자 사용)
    - 소요 시간 : 1개 케이스 해석 소요 시간의 11배

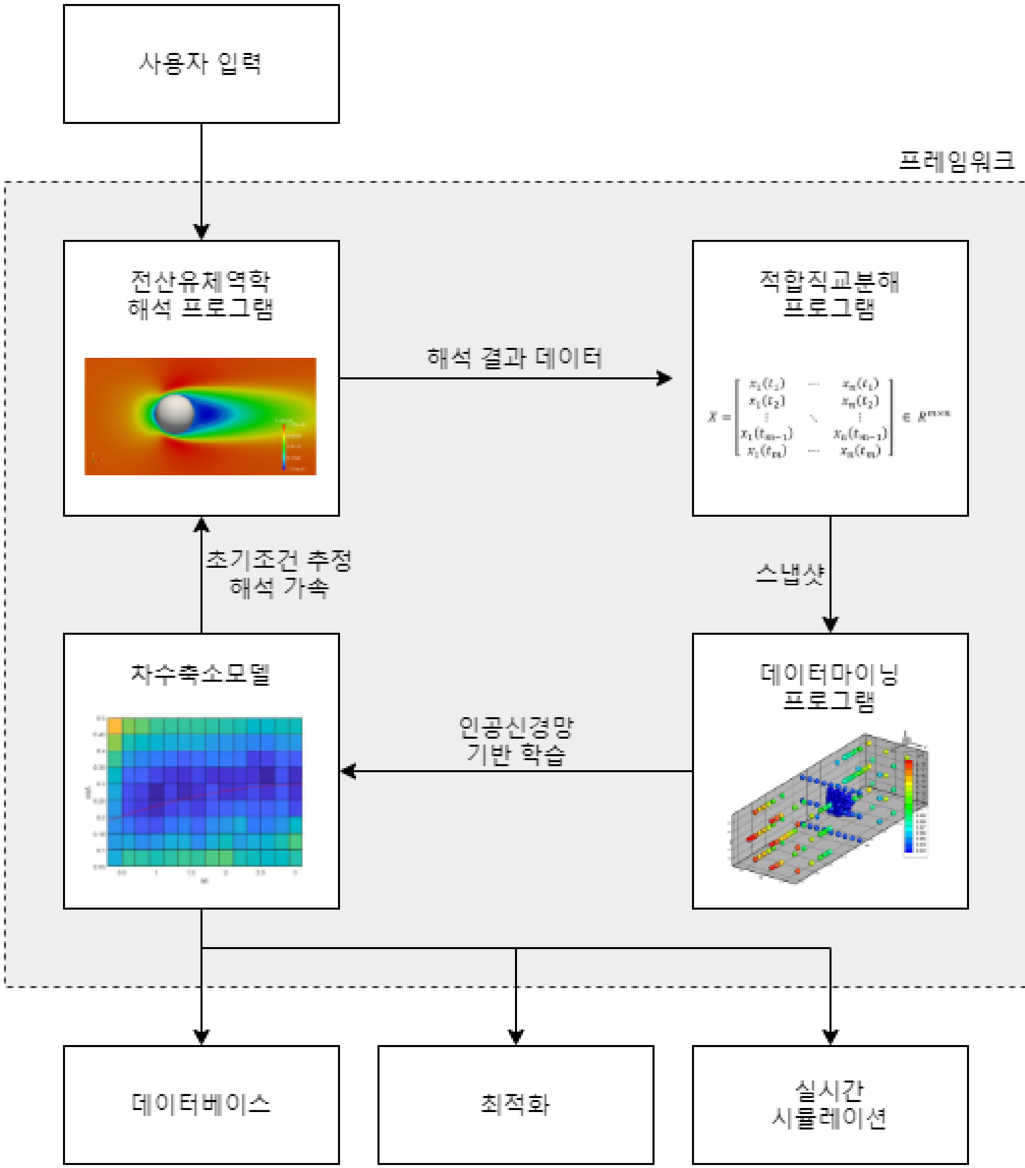


NEXTfoam open source CFD consulting

요약

- 연구 성과
  - 3개 프로그램을 연동/실행하는 프레임워크 개발 완료
  - 프레임워크 정확도 및 성능 검증 완료
  - 반복해석에 필요한 계산량을 용이하게 저감할 수 있는 기반 확보

- 향후 진행 예정 사항
  - 비정상 유동해석 결과에 대한 POD 수행
    - ITHACA-FV 로 교체 검토
  - 실시간 시뮬레이션 예제 구현



NEXTfoam
open source CFD consulting

감사합니다