Test Problem Database for Experimental Optimization (ver. 1.3)*

content by Gene A. Bunin†, typeset by Maurício Melo Câmara

LaTeXversion: February 22, 2017

*Extracted from `http://ccapprox.info/expopt/`.

†If needed, you can contact the author at `gene.a.bunin@ccapprox.info`.

Contents

1 Overview

The experimental optimization problems tested here have the standard form

$$\begin{array}{rll} \underset{\mathbf{u}}{\text{minimize}} & \phi_p(\mathbf{u}) & \\ \text{subject to} & g_{p,j}(\mathbf{u}) \leq 0, & j = 1, ..., n_{g_p} \\ & g_j(\mathbf{u}) \leq 0, & j = 1, ..., n_g \\ & u_i^L \leq u_i \leq u_i^U, & i = 1, ..., n_u, \end{array} \tag{1}$$

where $\mathbf{u} = (u_1, u_2, \ldots, u_{n_u})$ denotes the n_u decision variables, subject to the lower and upper limits u_i^L and u_i^U, and $\phi, g : \mathbb{R}^{n_u} \to \mathbb{R}$ denote the cost and constraint functions, respectively. The subscript p is used to denote those functions that are experimental in nature and whose values may only be evaluated for a given $\mathbf{u}$ by carrying out a (presumably) expensive experiment. By contrast, the functions without this subscript – namely, the numerical constraints g_j – may be evaluated for any $\mathbf{u}$ without running any experiments. For certain problems, the cost function may also be numerical in nature, in which case $\phi_p(\mathbf{u})$ is simply replaced by $\phi(\mathbf{u})$ in the formulation above.

To solve an experimental optimization problem, one starts with an initial decision-variable set $\mathbf{u}_0$ and generates a chain of additional experiments $\mathbf{u}_1, \mathbf{u}_2, \ldots$ by **using either an iterative algorithm or an algorithmic design procedure**, with the goal that the final experiments be close to the problem solution.

The goal of this project is to create a platform where different experimental optimization algorithms may be tested *en masse* for a large collection of experimental optimization problems and classes of such problems. While it is clear that **no algorithm can be the best for all problems**, it is very possible that certain algorithms would be the best for certain classes of problems on average, and it is hoped that this platform will succeed in discovering these links. Additionally, since the performance of an algorithm is itself subjective and depends on how each user judges performance, a number of performance metrics are used, such as convergence speed, constraint violations, and amount of suboptimality.

Because experimental optimization problems typically arise in engineering and real-life scenarios, the problems considered here are derived from case studies where the real experiments may be simulated by a reasonably accurate model of reality. Put otherwise, the focus is on well-thought-out case studies and not on mass-generated mathematical problems, although the latter also provide a valid manner of testing algorithms in certain contexts.

2 Solution Procedure

Letting k denote an experiment index, each of the algorithms tested must iteratively generate the next set of decision variables, $\mathbf{u}_{k+1}$, when given

- the past decision-variable sets $\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_k$,
- the corresponding measurements/estimates of the experimental cost/constraint function values,

$$\hat{\phi}_p(\mathbf{u}_0), \hat{\phi}_p(\mathbf{u}_1), \ldots, \hat{\phi}_p(\mathbf{u}_k) \text{ and } \hat{g}_{p,j}(\mathbf{u}_0), \hat{g}_{p,j}(\mathbf{u}_1), \ldots, \hat{g}_{p,j}(\mathbf{u}_k),$$

that are obtained for the experiments at $\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_k$,

- the statistical properties of the measurement/estimation noise,
- the definitions of the limits u_i^L, u_i^U and the numerical constraint functions g_j,
- (optional) a model of the experimental functions ϕ_p and $g_{p,j}$.

For the sake of simplicity, all of the noise corruption (w) in the measurements/estimates will be additive and white Gaussian, with

$$\hat{\phi}_p(\mathbf{u}_k) = \phi_p(\mathbf{u}_k) + w_{\phi,k}$$

$$\hat{g}_{p,j}(\mathbf{u}_k) = g_{p,j}(\mathbf{u}_k) + w_{j,k}$$

holding for any k, with $w_{\phi,k} \sim \mathcal{N}(0, \sigma_\phi^2)$ and $w_{j,k} \sim \mathcal{N}(0, \sigma_j^2)$. These probability distributions will be assumed to be known by the user – i.e., a priority in these test problems, at least for the time being, is how each algorithm rejects noise whose statistics are known. Problems for which the probability distributions are unknown may be included in the future, however.

The average performance of an algorithm for a given test problem is obtained by solving the problem 100 times with different pre-generated noise elements. The following MATLAB code (`algotest.m`) may be used to carry out this procedure:

```
function [perfave,convper] = algotest(u0,kfinal,sigmaphi,sigmag,uL,
    uU,ustar,Deltaphi,gpmax,gmax,algonum)
for i = 1:100
    noise = dlmread(strcat(['noise' num2str(i) '.txt']));
    wphi = noise(1,:);
    wg = noise(2:1+length(sigmag),:);
    u = u0;
    phiphat = [];
    gp = [];
    gphat = [];
    input = [];
    for k = 0:kfinal
        if exist('phieval.m') == 2
            phip(k+1,1) = phieval(u(k+1,:));
        else
            phip(k+1,1) = phipeval(u(k+1,:));
        end
        phiphat(k+1,1) = phip(k+1,1) + sigmaphi*wphi(k+1);
        if length(sigmag) > 0
            gp(k+1,:) = gpeval(u(k+1,:));
            gphat(k+1,:) = gp(k+1,:) + sigmag.*wg(:,k+1)';
        end
        tic;
        [u(k+2,:),output] = algo(u,phiphat,gphat,sigmaphi,sigmag,uL,
    uU,algonum,input);
        input = output;
        t(k+1) = toc;
    end
    perf(:,i) = perfeval(u(1:end-1,:),phip,gp,uL,uU,ustar,Deltaphi,
    gpmax,gmax,t);
end
```

```
for i = 1:11
    if i < 8 || i == 11
        perfave(i,:) = [mean(perf(i,:)) std(perf(i,:))];
    else
        perf0 = perf(i,:);
        perf0(perf0 > kfinal) = [];
        perfave(i,:) = [mean(perf0) std(perf0)];
        convper(i-7) = length(perf0);
    end
end
end
```

Here, the inputted `u0` is the initial decision-variable set $\mathbf{u}_0$ (in row vector form). `kfinal` is the number of additional experiments that are run to solve the problem. `sigmaphi` is σ_ϕ, i.e., the standard deviation of the noise element of the cost, while `sigmag` is a row vector specifying the standard deviations of the noise elements of the experimental constraints, with `sigmag(j)` corresponding to σ_j. In the case that the problem has no experimental constraints, the setting `sigmag = []` is used. `uL` and `uU` are both row vectors corresponding to the lower and upper limits, $\mathbf{u}^L = (u_1^L, u_2^L, \ldots, u_{n_u}^L)$ and $\mathbf{u}^U = (u_1^U, u_2^U, \ldots, u_{n_u}^U)$. `ustar` is the best known solution to the problem, while `Deltaphi`, `gpmax`, and `gmax` are scaling parameters for the performance metrics. Finally, `algonum` specifies which algorithm should be tested. Apart from `algonum`, which is varied to test different algorithms, the other settings are all fixed for each problem *a priori* and are provided together with the problem.

In order for this file to be executed, one needs to download:

- the pre-generated noise elements (`noise.rar`), extracting them where MATLAB will find them,
- the cost evaluation file `phipeval.m/phieval.m` and, if needed, the constraint evaluation files, `gpeval.m` and `geval.m` (available separately for each problem),
- models of the experimental functions, `phimod.m` and `gmod.m`, if the algorithm tested requires a model (available separately for each problem),
- any problem-dependent auxiliary files (available separately for each problem),
- the main algorithm file, `algo.m`,
- any algorithm-dependent auxiliary files (available separately for each algorithm),
- the performance evaluation file, `perfeval.m`.

All testing is carried out in MATLAB.

3 Performance Metrics

A total of 11 performance metrics is used to evaluate an algorithm's performance for a particular problem, with each computed via the performance evaluation file (`perfeval.m`).

3.1 Metrics 1-3: Average Suboptimality with Penalty for Constraint Violations

The first three metrics attempt to gauge "average suboptimality" as

$$\frac{1}{k_{\text{final}}+1}\sum_{k=0}^{k_{\text{final}}}\left(\phi_p(\mathbf{u}_k)-\phi_p(\mathbf{u}^*)\right),$$

where $\mathbf{u}^*$ is the best known optimal point of the problem. This quantity is particularly crucial in applications where the degree of optimality of each individual experiment is important, and where it is of interest to minimize the number of strongly suboptimal experiments. A penalty for constraint violations is included so as not to reward those algorithms that obtain low (or even negative) average suboptimality by violating the constraints:

$$\begin{aligned}
&\frac{1}{k_{\text{final}}+1}\sum_{k=0}^{k_{\text{final}}}\left(\phi_p(\mathbf{u}_k)-\phi_p(\mathbf{u}^*)\right)\\
&\quad+\frac{1}{k_{\text{final}}+1}\sum_{k=0}^{k_{\text{final}}}\sum_{j=1}^{n_{g_p}}\max\left[0,\lambda g_{p,j}(\mathbf{u}_k)\right]\\
&\quad+\frac{1}{k_{\text{final}}+1}\sum_{k=0}^{k_{\text{final}}}\sum_{j=1}^{n_{g}}\max\left[0,\lambda g_{j}(\mathbf{u}_k)\right]\\
&\quad+\frac{1}{k_{\text{final}}+1}\sum_{k=0}^{k_{\text{final}}}\sum_{i=1}^{n_{u}}\max\left[\lambda(u_i^L-u_{k,i}),0,\lambda(u_{k,i}-u_i^U)\right],
\end{aligned}$$

with $\lambda>0$ a penalty coefficient. So as not to introduce discrepancies due to scaling, a scaled version of the above is employed in defining the metrics:

$$\begin{aligned}
&\frac{1}{k_{\text{final}}+1}\sum_{k=0}^{k_{\text{final}}}\frac{\phi_p(\mathbf{u}_k)-\phi_p(\mathbf{u}^*)}{\Delta\phi_{max}}\\
&\quad+\frac{1}{k_{\text{final}}+1}\sum_{k=0}^{k_{\text{final}}}\sum_{j=1}^{n_{g_p}}\max\left[0,\lambda\frac{g_{p,j}(\mathbf{u}_k)}{g_{p,j}^{max}}\right]\\
&\quad+\frac{1}{k_{\text{final}}+1}\sum_{k=0}^{k_{\text{final}}}\sum_{j=1}^{n_{g}}\max\left[0,\lambda\frac{g_{j}(\mathbf{u}_k)}{g_{j}^{max}}\right]\\
&\quad+\frac{1}{k_{\text{final}}+1}\sum_{k=0}^{k_{\text{final}}}\sum_{i=1}^{n_{u}}\max\left[\lambda\frac{u_i^L-u_{k,i}}{u_i^U-u_i^L},0,\lambda\frac{u_{k,i}-u_i^U}{u_i^U-u_i^L}\right],
\end{aligned}$$

with $\Delta\phi_{max}$, $g_{p,j}^{max}$, and g_j^{max} all defined separately for each problem in such a manner so as to give each function a range that is approximately equal to unity. The value of λ is set to 1 for Metric 1, 10 for Metric 2, and 100 for Metric 3 to reflect light, medium, and heavy penalties for violating the constraints, respectively. In the case that a tested algorithm obtains a value of 9999 or above for any of these metrics, which would correspond to dramatically poor performance, the reported value is simply given as 9999 to indicate that this is the case.

3.2 Metric 4: Number of Constraint Violations

Metric 4 simply counts the total number of constraint violations without providing any information on the degree of violation, and can take values ranging from 0 to $(k_{\text{final}}+1)(n_{g_p}+n_g+n_u)$. This metric may be useful for those problems where any violations are extremely undesirable.

3.3 Metrics 5-7: Suboptimality at the Final Experiment with Penalty for Constraint Violations

Metrics 5-7 are similar to Metrics 1-3 but only consider the final experiment and ignore the suboptimality or constraint violations that may have taken place during the convergence process:

$$\frac{\phi_p(\mathbf{u}_{k_{\text{final}}})-\phi_p(\mathbf{u}^*)}{\Delta\phi_{max}}+\sum_{j=1}^{n_{g_p}}\max\left[0,\lambda\frac{g_{p,j}(\mathbf{u}_{k_{\text{final}}})}{g_{p,j}^{max}}\right]+\sum_{j=1}^{n_g}\max\left[0,\lambda\frac{g_j(\mathbf{u}_{k_{\text{final}}})}{g_j^{max}}\right]+\sum_{i=1}^{n_u}\max\left[\lambda\frac{u_i^L-u_{k_{\text{final}},i}}{u_i^U-u_i^L},0,\lambda\frac{u_{k_{\text{final}},i}-u_i^U}{u_i^U-u_i^L}\right].$$

Such a metric may be important for those applications where only the end result is important, with the potential costs of actually getting there considered to be negligible. As in Metrics 1-3, the penalty coefficient λ is set as 1 for Metric 5, 10 for Metric 6, and 100 for Metric 7. The upper limit of 9999 is enforced here as well when reporting the results of very poor performances.

3.4 Metrics 8-10: Number of Experiments Needed to Converge within a Tolerance

Metrics 8-10 are defined as the number of experiments needed for the algorithm to reduce the suboptimality of the initial experiment by a certain fraction and to maintain all further experiments below that threshold. They are defined as the lowest number of experiments, $\bar{k}$, satisfying

$$\bar{k}:\phi_p(\mathbf{u}_k)-\phi_p(\mathbf{u}^*)\leq\gamma\left[\phi_p(\mathbf{u}_0)-\phi_p(\mathbf{u}^*)\right],\ \forall k\in[\bar{k},k_{\text{final}}],$$

where γ is set as 0.5 for Metric 8, 0.3 for Metric 9, and 0.1 for Metric 10 to represent 50%, 70%, and 90% convergence, respectively. In the case that no $\bar{k}$ satisfy the above condition, that particular realization is not included in the averaged performance data, but the failure to converge within a tolerance in the allotted k_{final} experiments is nevertheless made known when the percentage of realizations that converged (out of the 100 tested) is reported. These metrics essentially reflect the convergence speed of the algorithm.

3.5 Metric 11: Average Computation Time of Algorithm per Iteration

Metric 11 reports the average time, in seconds, needed for the algorithm to carry out its computations, i.e., the time needed to execute the line

```
[u(k+2,:),output] = algo(u,phiphat,gphat,sigmaphi,sigmag,uL,uU,
  algonum,input);
```

in `algotest.m`. This may be relevant for those applications where the experiments are to be carried out at a high frequency and where fast computation is desired.

4 Algorithms

The following algorithms are currently being tested, with the code for each stored in the main algorithm file (`algo.m`). The priority of each algorithm, given as a score between 1 (lowest) and 10 (highest), determines the odds of that algorithm being tested in a given trial - e.g., an algorithm with Priority 3 is three times less likely to be tested than an algorithm with Priority 9.

No Algorithm (`nothing; algonum = 0`)

This "algorithm" does not adapt the decision variables and simply keeps them at $\mathbf{u}_0$ for all experiments. This is intended to provide a reference to the other algorithms, so that one may have a general idea of how much is gained (or lost) by attempting optimization.

SCFO Solver, Standard

The SCFO ("sufficient conditions for feasibility and optimality") solver is based on as-of-yet unpublished theoretical work [1], with the initial version of the solver released in May of 2013. The algorithms tested here correspond to all version numbers starting with 0.91.1, and use the SCFO's standard implementation (as opposed to the fast implementation - see below).

The `SCFO` file, as well as all of the other files it calls, should be downloaded separately. A users' guide to explain how to use the solver and the theory behind it is also available [2]. While the solver is designed to work in a model-free setting, it may employ the user-provided models `phimod.m` and `gmod.m` when they are available to assist in certain subroutines.

[1] G. A. Bunin, G. François, and D. Bonvin (2014). Feasible-side global convergence in experimental optimization. Unpublished, arXiv [math.OC] 1406.4063.

[2] G. A. Bunin (2017). The SCFO Experimental Optimization Solver: Users' Guide (version 0.91.3).

Version 0.91.1 (`SCFOv7; algonum = 1`)

Coded by: G. A. Bunin
Priority: see website

```
Wg = [];
for j = 1:length(sigmag)
    Wg{j} = [0 sigmag(j)^2];
end
if exist('phieval.m') == 2
```

```
    uopt = SCFO(u,[],gphat,[],Wg,uL,uU);
else
    uopt = SCFO(u,phiphat,gphat,[0 sigmaphi^2],Wg,uL,uU);
end
output = [];
```

Version 0.91.2 (`SCFOv8; algonum = 1.01`)

Coded by: G. A. Bunin
Priority: see website

```
Wg = [];
for j = 1:length(sigmag)
    Wg{j} = [0 sigmag(j)^2];
end
if exist('phieval.m') == 2
    [uopt,exitcond,output] = SCFO0_91_2(u,[],gphat,[],Wg,uL,uU,input
    );
else
    [uopt,exitcond,output] = SCFO0_91_2(u,phiphat,gphat,[0 sigmaphi
    ^2],Wg,uL,uU,input);
end
```

Version 0.91.3 (`SCFOv9; algonum = 1.02`)

Coded by: G. A. Bunin
Priority: see website

```
Wg = [];
for j = 1:length(sigmag)
    Wg{j} = [0 sigmag(j)^2];
end
if exist('phieval.m') == 2
    [uopt,exitcond,output] = SCFO0_91_3(u,[],gphat,[],Wg,uL,uU,input
    );
else
    [uopt,exitcond,output] = SCFO0_91_3(u,phiphat,gphat,[0 sigmaphi
    ^2],Wg,uL,uU,input);
end
```

SCFO Solver, Fast

This is the fast version of the SCFO solver (see above), and is essentially a modification of the standard version in that certain numerical routines are approximated to drastically reduce the solver's computational time. In terms of implementation, only the solver settings need modification.

Version 0.91.1 (`SCFOv7f; algonum = 2`)

Coded by: G. A. Bunin
Priority: see website

```
Wg = [];
for j = 1:length(sigmag)
   Wg{j} = [0 sigmag(j)^2];
end
if exist('phieval.m') == 2
   uopt = SCFO(u,[],gphat,[],Wg,uL,uU
    ,[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],1);
else
   uopt = SCFO(u,phiphat,gphat,[0 sigmaphi^2],Wg,uL,uU
    ,[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],1);
end
output = [];
```

Version 0.91.2 (`SCFOv8f; algonum = 2.01`)

Coded by: G. A. Bunin
Priority: see website

```
Wg = [];
for j = 1:length(sigmag)
   Wg{j} = [0 sigmag(j)^2];
end
if exist('phieval.m') == 2
   [uopt,exitcond,output] = SCFO0_91_2(u,[],gphat,[],Wg,uL,uU,input
    ,[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],1);
else
   [uopt,exitcond,output] = SCFO0_91_2(u,phiphat,gphat,[0 sigmaphi
    ^2],Wg,uL,uU,input
    ,[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],1);
end
```

Version 0.91.3 (`SCFOv9f; algonum = 2.02`)

Coded by: G. A. Bunin
Priority: see website

```
Wg = [];
for j = 1:length(sigmag)
   Wg{j} = [0 sigmag(j)^2];
end
if exist('phieval.m') == 2
   [uopt,exitcond,output] = SCFO0_91_3(u,[],gphat,[],Wg,uL,uU,input
    ,[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],1);
else
   [uopt,exitcond,output] = SCFO0_91_3(u,phiphat,gphat,[0 sigmaphi
    ^2],Wg,uL,uU,input
    ,[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],1);
end
```

Response Surface Optimization Following a Central Composite Design (`RSOCCD; algonum = 3`)

Coded by: G. A. Bunin
Priority: see website
This algorithm is not iterative but is an algorithmic design procedure, as a certain number of prescribed experiments are conducted regardless of what results they yield (i.e., in an open-loop matter), after which the resulting data is used to build a quadratic-model approximation of the experimental optimization problem, which is then solved numerically to yield an approximation of the optimal point. This point is then retained as the optimum for all following experiments. A central composite design to generate the initial set is used as this is a fairly popular and accepted procedure [1].

```
udoe = ccdesign(length(uL),'type','faced','center',1);
for i = 1:length(uL)
   for j = 1:length(udoe(:,1))
      set = 0;
      if udoe(j,i) == -1
         udoe(j,i) = uL(i);
         set = 1;
      end
      if udoe(j,i) == 0 && set == 0
         udoe(j,i) = 0.5*uL(i) + 0.5*uU(i);
         set = 1;
      end
      if udoe(j,i) == 1 && set == 0
         udoe(j,i) = uU(i);
      end
   end
end

if length(phiphat) <= length(udoe(:,1))
   uopt = udoe(length(phiphat),:);
elseif length(phiphat) == length(udoe(:,1))+1
   X = u.^2;
   comb = nchoosek(1:length(uL),2);
   for i = 1:length(comb(:,1))
      X = [X u(:,comb(i,1)).*u(:,comb(i,2))];
   end
   X = [X u ones(length(phiphat),1)];
   aphi = pinv(X)*phiphat;
   ag = [];
   for j = 1:length(sigmag)
      ag(:,j) = pinv(X)*gphat(:,j);
   end
   lowcost = 1e6;
   ubest = [];
   for i = 1:100
      urand = uL+rand(1,length(uL)).*(uU-uL);
      if length(sigmag) > 0 || exist('geval.m') == 2
         if exist('phieval.m') == 2
            [ucand,fcand,exitflag] = fmincon(@phieval,urand
    ,[],[],[],[],uL,uU,@(u)gdoe(u,ag),optimset('disp','none'));
         else
            [ucand,fcand,exitflag] = fmincon(@(u)phidoe(u,aphi),
    urand,[],[],[],[],uL,uU,@(u)gdoe(u,ag),optimset('disp','none'))
    ;
         end
      else
```

```
            if exist('phieval.m') == 2
                [ucand,fcand,exitflag] = fmincon(@phieval,urand
    ,[],[],[],[],uL,uU,[],optimset('disp','none'));
            else
                [ucand,fcand,exitflag] = fmincon(@(u)phidoe(u,aphi),
    urand,[],[],[],[],uL,uU,[],optimset('disp','none'));
            end
        end
        if exitflag > 0 && fcand < lowcost
            lowcost = fcand;
            ubest = ucand;
        end
    end
    if isempty(ubest) == 1
        uopt = u(1,:);
    else
        uopt = ubest;
    end
else
    uopt = u(length(udoe(:,1))+2,:);
end
output = [];
```

The files `phidoe.m` and `gdoe.m` complement this algorithm and act as the functions needed for the optimization of the constructed data-driven model. Note that this algorithm does not make use of a user-provided model even if the latter is available.

[1] R. H. Myers, D. C. Montgomery, and C. M. Anderson-Cook (2009). Response Surface Methodology. John Wiley & Sons.

Brute Simplex Algorithm (`SimplexB; algonum = 4`)

Coded by: G. A. Bunin
Priority: see website
The simplex algorithm is a classic tool for nonlinear optimization, with the version coded here essentially following the steps outlined in Section 2.4.1 of [1].

```
for i = 1:length(uL)
    scale(i,1) = 1/(uU(i)-uL(i));
    scale(i,2) = -uL(i)/(uU(i)-uL(i));
    u(:,i) = u(:,i)*scale(i,1) + scale(i,2);
    uL(i) = 0;
    uU(i) = 1;
end
output = input;

if length(phiphat) < length(uL)+1
    refind0 = find(abs(phiphat-min(phiphat)) < 1e-6);
    refind = refind0(1);
    uref = u(refind,:);
    objval = 1e32;
    uopt = [];
    s1 = 1;
    while isempty(uopt) == 1
        i = 1;
        while i < 1000
            du = randn(1,length(uL));
            utest = uref + s1*.1*du/norm(du,2);
```

```
22          Uaug = [u; utest];
23          if min(utest-uL) >= 0 && min(uU-utest) >= 0 && cond(diff(
    Uaug)) < objval
24              objval = cond(diff(Uaug));
25              uopt = utest;
26          end
27          i = i + 1;
28       end
29       s1 = .5*s1;
30    end
31    output.simptest = 0;
32    output.simpexpand = 0;
33    output.simpcont = 0;
34    output.simpshrink = 0;
35 else
36    delta = 0.5;
37    if length(phiphat) == length(uL)+1
38       usub = u;
39       phisub = phiphat;
40       output.simpusub = usub;
41       output.simpphisub = phisub;
42    else
43       usub = input.simpusub;
44       phisub = input.simpphisub;
45    end
46    simptest = input.simptest;
47    simpexpand = input.simpexpand;
48    simpcont = input.simpcont;
49    simpshrink = input.simpshrink;
50    [phipsort,sortord] = sort(phisub);
51    if simptest == 0
52       [uopt,output] = simpsub(usub,phisub,uL,uU,output);
53       output.simptest = 1;
54    elseif simptest == 1 && simpshrink == 0
55       phipmin = input.simpphipmin;
56       phipmax = input.simpphipmax;
57       phipmax2 = input.simpphipmax2;
58       minind = input.simpminind;
59       maxind = input.simpmaxind;
60       ucen = input.simpucen;
61       umax = input.simpumax;
62       umin = input.simpumin;
63       if simpexpand == 0 && simpcont == 0
64          if phiphat(end) >= phipmin && phiphat(end) < phipmax2
65             usub(maxind,:) = u(end,:);
66             phisub(maxind) = phiphat(end);
67             output.simpusub = usub;
68             output.simpphisub = phisub;
69             [uopt,output] = simpsub(usub,phisub,uL,uU,output);
70          elseif phiphat(end) < phipmin
71             feas = 0;
72             gamma0 = 2;
73             while feas == 0
74                uexp = gamma0*u(end,:)+(1-gamma0)*ucen;
75                if min(uexp-uL) >= 0 && min(uU-uexp) >= 0
76                   feas = 1;
77                   uopt = uexp;
78                else
79                   gamma0 = .5*gamma0+.5;
80                end
81             end
82             output.simpexpand = 1;
```

```
83          elseif phiphat(end) >= phipmax2
84              beta0 = 0.5;
85              if phiphat(end) < phipmax
86                  uopt = beta0*u(end,:)+(1-beta0)*ucen;
87              else
88                  uopt = beta0*umax+(1-beta0)*ucen;
89              end
90              output.simpcont = 1;
91          end
92      elseif simpexpand == 1
93          if phiphat(end) < phiphat(end-1)
94              usub(maxind,:) = u(end,:);
95              phisub(maxind) = phiphat(end);
96              output.simpusub = usub;
97              output.simpphisub = phisub;
98              [uopt,output] = simpsub(usub,phisub,uL,uU,output);
99          else
100             usub(maxind,:) = u(end-1,:);
101             phisub(maxind) = phiphat(end-1);
102             output.simpusub = usub;
103             output.simpphisub = phisub;
104             [uopt,output] = simpsub(usub,phisub,uL,uU,output);
105         end
106         output.simpexpand = 0;
107     elseif simpcont == 1
108         if phiphat(end-1) < phipmax
109             if phiphat(end) <= phiphat(end-1)
110                 usub(maxind,:) = u(end,:);
111                 phisub(maxind) = phiphat(end);
112                 output.simpusub = usub;
113                 output.simpphisub = phisub;
114                 [uopt,output] = simpsub(usub,phisub,uL,uU,output);
115             else
116                 usub = delta*usub + (1-delta)*ones(length(uL)+1,1)*
    umin;
117                 output.simpusub = usub;
118                 ushrink = usub;
119                 ushrink(minind,:) = [];
120                 uopt = ushrink(1,:);
121                 ushrink(1,:) = [];
122                 if length(ushrink(:,1)) > length(uL)
123                     output.simpshrink = 1;
124                     output.simpushrink = ushrink;
125                     output.simpshrinkind = 1;
126                 end
127             end
128         else
129             if phiphat(end) < phipmax
130                 usub(maxind,:) = u(end,:);
131                 phisub(maxind) = phiphat(end);
132                 output.simpusub = usub;
133                 output.simpphisub = phisub;
134                 [uopt,output] = simpsub(usub,phisub,uL,uU,output);
135             else
136                 usub = delta*usub + (1-delta)*ones(length(uL)+1,1)*
    umin;
137                 output.simpusub = usub;
138                 ushrink = usub;
139                 ushrink(minind,:) = [];
140                 uopt = ushrink(1,:);
141                 ushrink(1,:) = [];
142                 if length(ushrink(:,1)) > length(uL)
```

```
                    output.simpshrink = 1;
                    output.simpushrink = ushrink;
                    output.simpshrinkind = 1;
                end
            end
        end
        output.simpcont = 0;
    end
elseif simpshrink == 1
    ushrink = input.simpushrink;
    shrinkind = input.simpshrinkind;
    minind = input.simpminind;
    if shrinkind < minind
        phisub(shrinkind) = phiphat(end);
        output.simpphisub = phisub;
    else
        phisub(shrinkind+1) = phiphat(end);
        output.simpphisub = phisub;
    end
    shrinkind = shrinkind + 1;
    output.simpshrinkind = shrinkind;
    if isempty(ushrink) == 0
        uopt = ushrink(1,:);
        ushrink(1,:) = [];
        if length(ushrink(:,1)) > length(uL)
            output.simpushrink = ushrink;
            output.simpshrink = 1;
        end
    else
        output.simpshrink = 0;
        [uopt,output] = simpsub(usub,phisub,uL,uU,output);
    end
end
end

for i = 1:length(uL)
    uopt(i) = (uopt(i)-scale(i,2))/scale(i,1);
end
```

The file `simpsub.m` is a required subroutine and should be downloaded. Note that the algorithm uses scaled decision variables and generates the experiments $\mathbf{u}_1, \ldots, \mathbf{u}_{n_u}$ in a pseudo-random manner that ensures that the starting simplex has relatively balanced geometry. More importantly, this version of the algorithm is only applicable to problems with bound constraints only (Class 1 problems) and thus is only tested for this problem class. The adjective "brute" is employed since this version of the algorithm does not attempt to account for noise in the function values in any manner, and simply proceeds by using the noisy measurements in the iterative simplex constructions. The algorithm does not make use of user-provided models.

[1] J. C. Spall (2005). Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control. John Wiley & Sons.

Constraint Adaptation (`ConAdapt; algonum = 5`)

Coded by: G. A. Bunin
Priority: see website

The constraint-adaptation [1], or bias-update [2], algorithm proceeds by adding a zero-order correction term to the models of the experimental constraints and then carrying out a model optimization to determine the next experiment. In the algorithm as coded here, a filter gain of 0.7 is used to dampen the corrections slightly. This algorithm is applicable to Class 3 problems only and requires a model.

```
if length(u(:,1)) == 1
   eprev = zeros(1,length(sigmag));
else
   eprev = input.eprev;
end
ecur = 0.3*eprev + 0.7*(gphat(end,:)-gmod(u(end,:)));
output.eprev = ecur;
ubest = [];
u0 = u(end,:);
i = 1;
while max(gCA(u0,ecur)) > 0 && i < 100
   u0 = uL + rand(1,length(u(1,:))).*(uU-uL);
   i = i + 1;
end
if exist('phieval.m') == 2
   [ucand,fcand,exitflag] = patternsearch(@phieval,u0,[],[],[],[],
    uL,uU,@(u)gCA(u,ecur),optimset('disp','none'));
else
   [ucand,fcand,exitflag] = patternsearch(@(u)phimod(u),u0
    ,[],[],[],[],uL,uU,@(u)gCA(u,ecur),optimset('disp','none'));
end
if exitflag > 0
   ubest = ucand;
end
if isempty(ubest) == 1
   uopt = u(end,:);
else
   uopt = ubest;
end
```

The auxiliary file `gCA.m` is required as it defines the constraints in the model optimization. The version of the algorithm coded here uses the pattern search algorithm of MATLAB to minimize the model.

[1] B. Chachuat, A. Marchetti, and D. Bonvin (2008). Process optimization via constraints adaptation. J. Process Control, 18, 244-257.

[2] J. F. Forbes and T. E. Marlin (1994). Model accuracy for economic optimizing controllers: The bias update case. Ind. Eng. Chem. Res., 33(8), 1919-1929.

5 Test Problems and Results

The following are the test problems currently making up the database, together with the performances obtained by the different tested algorithms. To make it easier to pair up problems with algorithms and to analyze the results, the problems are broken up into three classes, with Class 1 problems denoting those with only bound constraints, Class 2 problems as those with general numerical

constraints but no experimental constraints, and Class 3 problems being those that possess experimental constraints.

- Class 1: P5.1, P5.7, P5.9;
- Class 2: P5.8, P5.10;
- Class 3: P5.3, P5.4, P5.5, P5.6, P5.11.

In reporting the metric values, both the average (left) and the standard error (right) are given, with the plus-minus (±) sign used to separate the two. For Metrics 8-10, which attempt to gauge convergence speed, the percentage of the trials that achieved the specified convergence is also given. In the case that convergence was not achieved for any of the trials, the letters "NA" (not available) are reported. Currently, testing is done automatically by dedicated computers, with a maximum of 100 trials done for each testing combination (generated by choosing the problem, the algorithm, and whether or not the model is used).

You may view the plotted results for the individual trials for each algorithm by clicking on the algorithm name in the "Name" column. For problems with only two variables, plots of the experimental iterates in the decision-variable space are given for particularly insightful illustrations – only their values are plotted for problems with three or more variables. For plots of the decision-variable space, green regions denote those that are feasible (where all constraints are satisfied), while red denotes the infeasible regions. Red dots denote the individual experimental iterates, while the green dot denotes the true optimum that the algorithm aims to find. Constant dotted lines are used to plot the optimal values in the three-or-more variable case. For the cost function value plots, the constant black line denotes the value at the true optimum.

Because the results are constantly being updated, they are not reported directly in this .pdf version. Readers are requested to check the website for the most up-to-date results.

5.1 Test Problem #1: Maximizing Profit in the Williams-Otto Reactor

Original code provided by: S. Costello

Test File Specs: `algotest([4.8 77],40,0.5,[],[3 70],[6 100],[4.79 89.7],100,[],[],algonum)`

Main Files: `phipeval.m` (required), `phimod.m` (model)

Auxiliary Files: `plantbalancesT.m` (required), `modelbalances2.m` (model)

Problem Description: The Williams-Otto reactor is a continuously-stirred tank reactor (CSTR) whose operation was originally formulated in the 1960 paper of Williams and Otto [1]. Since then, it has become a standard test problem in the real-time optimization (RTO) research community, where the objective is usually to maximize a steady-state profit function by varying both the flow rate of one of the feed components to the reactor and the reactor

temperature. The steady-state values that result are governed by the underlying chemical dynamics of the system, described by the three reactions

$$\begin{aligned} A + B &\xrightarrow{k_1} C, & k_1 &= 1.660 \times 10^6 e^{-6666.7/(T_R+273.15)} \\ C + B &\xrightarrow{k_2} P + E, & k_2 &= 7.212 \times 10^8 e^{-8333.3/(T_R+273.15)} \\ C + P &\xrightarrow{k_3} G, & k_3 &= 2.675 \times 10^{12} e^{-11111/(T_R+273.15)} \end{aligned}$$

and the corresponding rate laws, with T_R denoting the temperature of the reactor. A model of this system that approximates the three reactions by two [2] is assumed to be available:

$$\begin{aligned} A + 2B &\xrightarrow{k_1^*} P + E, & k_1^* &= 2.189 \times 10^8 e^{-8077.6/(T_R+273.15)} \\ A + B + P &\xrightarrow{k_2^*} G, & k_2^* &= 4.310 \times 10^{13} e^{-12438/(T_R+273.15)}. \end{aligned}$$

With regard to specific problem parameters and the definition of the optimization problem, the version of the problem recently reported in [3] is used here, where the profit function is defined as

$$1143.38X_P(F_B, T_R)(F_A+F_B)+25.92X_E(F_B, T_R)(F_A+F_B)-76.23F_A-114.34F_B,$$

with X denoting the steady-state concentrations of the different chemical reactants (all being innate functions of the decision variables) and F denoting the feed rates of the A and B reactants. Noting that the decision-variable vector is defined as $\mathbf{u} := (F_B, T_R)$ and that one should minimize the negative of the profit when solving the problem in standard minimization form, the cost function is stated as:

$$\phi_p(\mathbf{u}) := (-1143.38X_P(\mathbf{u}) - 25.92X_E(\mathbf{u}))(F_A + u_1) + 114.34u_1.$$

The feed rate of B is constrained to be between 3 and 6 kg/s, while the temperature is constrained to the 70-100 °C range. This leads to the experimental optimization problem in standard form:

$$\begin{aligned} \underset{\mathbf{u}}{\text{minimize}} \quad & \phi_p(\mathbf{u}) := (-1143.38X_P(\mathbf{u}) - 25.92X_E(\mathbf{u}))(F_A + u_1) + 114.34u_1 \\ \text{subject to} \quad & 3 \leq u_1 \leq 6 \\ & 70 \leq u_2 \leq 100. \end{aligned}$$

[1] T. J. Williams and R. E. Otto (1960). A generalized chemical processing model for the investigation of computer control. AIEE Trans., 79(5): 458-473.

[2] J. F. Forbes, T. E. Marlin, and J. F. MacGregor (1994). Model adequacy requirements for optimizing plant operations. Comput. Chem. Eng., 18(6): 497-510.

[3] A. G. Marchetti (2013). A new dual modifier-adaptation approach to iterative process optimization with inaccurate models. Comput. Chem. Eng., 59(5): 89-100.

5.2 Test Problem #2: Maximizing Profit in the Williams-Otto Reactor (Constrained)

Original code provided by: S. Costello

Test File Specs: `algotest([3.5 72],40,0.5,5e-4,[3 70],[6 100],[4.97 84.3],100,.1,[],algonum)`

Main Files: `phipeval.m` (required), `gpeval.m` (required), `phimod.m` (model), `gmod.m` (model)

Auxiliary Files: `plantbalancesT.m` (required), `modelbalances2.m` (model)

Problem Description: This problem is taken from [1] and is an extension of Test Problem #1 in that it adds an experimental constraint on the maximum concentration of the product G,

$$X_G(\mathbf{u}) \leq 0.08,$$

with the problem in standard form then becoming

$$\begin{array}{ll} \underset{\mathbf{u}}{\text{minimize}} & \phi_p(\mathbf{u}) := (-1143.38X_P(\mathbf{u}) - 25.92X_E(\mathbf{u}))(F_A + u_1) + 114.34u_1 \\ \text{subject to} & g_{p,1}(\mathbf{u}) := X_G(\mathbf{u}) - 0.08 \leq 0 \\ & 3 \leq u_1 \leq 6 \\ & 70 \leq u_2 \leq 100. \end{array}$$

[1] A. G. Marchetti (2013). A new dual modifier-adaptation approach to iterative process optimization with inaccurate models. Comput. Chem. Eng., 59(5): 89-100.

5.3 Test Problem #3: Minimizing the Batch Time of Polystyrene Production

Original code provided by: G. François

Test File Specs: `algotest([242.39 945.30],40,60,1e4,[50 600],[450 1000],[363.06 875.60],10000,1e6,[],algonum)`

Main Files: `phipeval.m` (required), `gpeval.m` (required), `phimod.m` (model), `gmod.m` (model)

Auxiliary Files: `integstyrbis.m` (required), `linint.m` (required)

Problem Description: This problem is a specific case of the case-study example presented in [1], where the problem of finding the optimal temperature profile for a polymerization batch reactor is addressed. Instead of solving the dynamic optimization problem, which is infinite-dimensional and intractable, the authors propose to break the profile into four pre-defined operating regimes and to optimize the "switching times" between these, which results in a finite-dimensional experimental optimization problem. An ODE model is used to simulate the actual dynamic behavior of the batch. The process has two major requirements, with (a) 60% conversion of monostyrene to polysterene required at the end of the batch, and (b) the molecular weight of the product at the end of the batch needing to be at least 2 million grams per mole.

In the particular case presented here, the batch is set to terminate as soon as 60% conversion is reached, with the goal being to find the temperature profile that minimizes the batch time, t_b, while satisfying the molecular weight constraint, $M(\mathbf{u}) \geq 2 \cdot 10^6$. Only the first two switching times are used as optimization variables, with the first switching time constrained to vary between 50 and 450 seconds and the second between 600 and 1000 seconds. The resulting experimental optimization problem is then stated as

$$\begin{aligned} \underset{\mathbf{u}}{\text{minimize}} \quad & \phi_p(\mathbf{u}) := t_b(\mathbf{u}) \\ \text{subject to} \quad & g_{p,1}(\mathbf{u}) := -M(\mathbf{u}) + 2 \cdot 10^6 \leq 0 \\ & 50 \leq u_1 \leq 450 \\ & 600 \leq u_2 \leq 1000. \end{aligned}$$

A model with errors in two of the parameters ($k_{\mathrm{p_0}}$ and $k_{\mathrm{tr,M_0}}$, as suggested in [1]) is assumed to be available.

[1] G. François, B. Srinivasan, and D. Bonvin (2005). Use of measurements for enforcing the necessary conditions of optimality in the presence of constraints and uncertainty. J. Process Control, 15(6): 701-712.

5.4 Test Problem #4: Maximizing Electrical Efficiency in a Solid Oxide Fuel Cell Stack

Original code provided by: A. Gopalakrishnan

Test File Specs: `algotest([2e-3 7e-3 26],60,6e-4,[0.1 0.1 1e-3 0.03],[1e-3 1.5e-3 1],[1e-2 3.5e-2 30],[1e-3 1.898e-3 23.6687],.1,[50 50 .1 50],[.01 .01 .01],algonum)`

Main Files: `phipeval.m` (required), `gpeval.m` (required), `geval.m` (required), `phimod.m` (model), `gmod.m` (model)

Auxiliary Files: `model_real.m` (required), `stack.m` (required), `model.m` (model)

Problem Description: This problem is derived from the simulated case study reported in [1], where the goal is to maximize the electrical efficiency of a 5-cell solid oxide fuel cell stack, denoted by η, by manipulating the molar inflow of the fuel, the molar inflow of the air, and the current, assigned respectively as u_1, u_2, and u_3.

A number of constraints are imposed to ensure healthy operating conditions for the stack. The first two experimental constraints define the desired operating temperature range of the stack, $1003\text{K} \leq T(\mathbf{u}) \leq 1073\text{K}$. Additionally, the cell voltages - assumed here to be identical - must be kept above 0.7V as operating below this limit has been shown to lead to significantly faster cell degradation. This constraint is written as $U_{\mathrm{cell}}(\mathbf{u}) \geq 0.7$. Finally, the last experimental constraint is on the power produced, which here is required to be at least 80W. Contrary to the study in [1], where it was formulated as an equality constraint, here it is formulated as the inequality $P_{\mathrm{el}}(\mathbf{u}) \geq 80$. Little is lost with this reformulation, since there are no significant drawbacks from producing more power than required (apart from inefficiency, which the optimization should

eliminate), and it is advantageous as it allows for the problem to be cast in the standard (inequality-only) form.

To avoid oxidation of the anode, an upper limit of 0.7 is set on the fuel utilization. This limitation is equivalent to the numerical constraint $\frac{5u_3}{2Fu_1} \leq 0.7$, with F denoting the Faraday constant. To make this constraint easier for optimization algorithms to handle, it is reformulated into the equivalent linear form $\frac{5u_3}{2F} \leq 0.7u_1$, which is permissible since u_1, a molar inflow, must always be positive. Limitations are also placed on the air-to-fuel ratio of inflows, with $3 \leq \frac{2u_2}{u_1} \leq 7$. These may also be reformulated into linear constraints to yield $3u_1 \leq 2u_2$ and $2u_2 \leq 7u_1$.

A lower limit on the molar inflow of the fuel is set as $0.001\frac{\text{mol}}{\text{s}}$ to avoid fuel starvation. Note that this is twice the lower bound used in [1] - the reason being that the simulated reality model provided has been noted to return imaginary values for certain conditions with lower molar inflows. An upper limit of $0.01\frac{\text{mol}}{\text{s}}$ is used as a sufficiently high value that one does not expect to be attained (no upper limit is specified in [1]). For the air molar inflow, implicit lower and upper limits resulting from the air-to-fuel ratio constraints are used, with the values of $0.0015\frac{\text{mol}}{\text{s}}$ and $0.035\frac{\text{mol}}{\text{s}}$, respectively. The current is constrained to vary between 1A and 30A, the latter being a limit imposed in [1].

Combining all of these specifications finally leads to the experimental optimization problem in standard form:

$$
\begin{aligned}
\underset{\mathbf{u}}{\text{minimize}} \quad & \phi_p(\mathbf{u}) := -\eta(\mathbf{u}) \\
\text{subject to} \quad & g_{p,1}(\mathbf{u}) := -T(\mathbf{u}) + 1003 \leq 0 \\
& g_{p,2}(\mathbf{u}) := T(\mathbf{u}) - 1073 \leq 0 \\
& g_{p,3}(\mathbf{u}) := -U_{\text{cell}}(\mathbf{u}) + 0.7 \leq 0 \\
& g_{p,4}(\mathbf{u}) := -P_{\text{el}}(\mathbf{u}) + 80 \leq 0 \\
& g_1(\mathbf{u}) := \tfrac{5u_3}{2F} - 0.7u_1 \leq 0 \\
& g_2(\mathbf{u}) := 3u_1 - 2u_2 \leq 0 \\
& g_3(\mathbf{u}) := 2u_2 - 7u_1 \leq 0 \\
& 0.001 \leq u_1 \leq 0.01 \\
& 0.0015 \leq u_2 \leq 0.035 \\
& 1 \leq u_3 \leq 30.
\end{aligned}
$$

A model, as given in [1], is assumed to be available.

[1] A. Marchetti, A. Gopalakrishnan, B. Chachuat, D. Bonvin, L. Tsikonis, A. Nakajo, Z. Wuillemin, and J. Van herle (2011). Robust real-time optimization of a solid oxide fuel cell stack. J. Fuel Cell Sci. Technol., 8(5): 1-11.

5.5 Test Problem #5: Minimizing the Overall Pumping Effort in the "Trois Bacs"

Original code provided by: G. A. Bunin

Test File Specs: `algotest([6 6.2],40,0,[0.2 0.2 0.2 0.2 0.2 0.2],[0 0],[8 8],[4.9481 4.4160],40,[20 20 20 20 20 20],[],algonum)`

Main Files: `phieval.m` (required), `gpeval.m` (required), `gmod.m` (model)

Problem Description: This problem is derived from the experimental testing set-up in the Laboratoire d'Automatique of the École Polytechnique Fédérale de Lausanne (EPFL) in Lausanne, Switzerland. While the problem does not present an industrial application, it is nevertheless considered to be a good problem for testing since it involves the optimal operation of an experimental system. A more detailed account is available in [1].

This simple set-up consists of three water tanks joined together, with water being pumped into the leftmost and rightmost ones by controlling the voltages of the two pumps. The test problem adopted here, much like the one outlined in [1], aims at maintaining the water levels in the three tanks within the acceptable ranges of 5 to 30 cm, while minimizing the overall pumping effort, represented by the sum of squares of the two voltages.

The heights, denoted by h_1, h_2, and h_3, correspond to the steady-state solution for a system of nonlinear equations for a given set of voltages and are thus experimental functions of the voltages. Letting the voltages be the decision variables u_1 and u_2, and noting that they are constrained to lie between 0 and 8 V, the resulting experimental optimization problem is stated as:

$$\begin{aligned} \underset{\mathbf{u}}{\text{minimize}} \quad & \phi(\mathbf{u}) := u_1^2 + u_2^2 \\ \text{subject to} \quad & g_{p,1}(\mathbf{u}) := 5 - h_1(\mathbf{u}) \leq 0 \\ & g_{p,2}(\mathbf{u}) := 5 - h_2(\mathbf{u}) \leq 0 \\ & g_{p,3}(\mathbf{u}) := 5 - h_3(\mathbf{u}) \leq 0 \\ & g_{p,4}(\mathbf{u}) := h_1(\mathbf{u}) - 30 \leq 0 \\ & g_{p,5}(\mathbf{u}) := h_2(\mathbf{u}) - 30 \leq 0 \\ & g_{p,6}(\mathbf{u}) := h_3(\mathbf{u}) - 30 \leq 0 \\ & 0 \leq u_1 \leq 8 \\ & 0 \leq u_2 \leq 8. \end{aligned}$$

The real experimental system is simulated by using the model provided in [1]. A model of the experimental system is assumed to be available and is obtained by perturbing the parameters of the model of [1] by 10%.

[1] A. Marchetti (2009). Modifier-adaptation methodology for real-time optimization. Ph. D. Thesis, EPFL (p. 120-123).

5.6 Test Problem #6: Maximizing Production in a Continuous Stirred-Tank Reactor

Original code provided by: G. A. Bunin

Test File Specs: `algotest([14.52 14.9],40,0.1,[0.03 0.03],[1 1],[50 50],[17.2 30.3],10,[1 1],[],algonum)`

Main Files: `phipeval.m` (required), `gpeval.m` (required), `phimod.m` (model), `gmod.m` (model)

Problem Description: This problem is derived from the case study example of [1], and seeks to maximize the steady-state production of the product C in a continuous stirred-tank reactor with the reactions

$$\begin{aligned} A + B &\rightarrow C \\ 2B &\rightarrow D. \end{aligned}$$

The decision variables of the problem are the feed rates of reactants A and B, denoted by u_1 and u_2, respectively, both of which are maintained between 1 and 50 liters per minute. The steady-state concentrations c_A, c_B, c_C, and c_D are obtained from the steady-state equations

$$\begin{aligned}
0 &= -k_1 c_A c_B + \tfrac{u_1}{V} c_{A,in} - \tfrac{u_1+u_2}{V} c_A \\
0 &= -k_1 c_A c_B - 2k_2 c_B^2 + \tfrac{u_2}{V} c_{B,in} - \tfrac{u_1+u_2}{V} c_B \\
0 &= k_1 c_A c_B - \tfrac{u_1+u_2}{V} c_C \\
0 &= k_2 c_B^2 - \tfrac{u_1+u_2}{V} c_D,
\end{aligned}$$

with the parameters k_1, k_2, V, $c_{A,in}$, and $c_{B,in}$ defined as in [1]. The steady-state cost to be minimized is the quantity

$$0.004(u_1^2 + u_2^2) - \frac{c_C^2 (u_1+u_2)^2}{u_1 c_{A,in}},$$

which seeks to maximize the production of C while penalizing the control action.

A constraint on maximal heat generation, $Q \leq 110$kcal, is also introduced, with the steady-state value of Q defined as

$$Q = -V(k_1 c_A c_B \Delta H_{r,1} + k_2 c_B^2 \Delta H_{r,2}),$$

with the parameters $\Delta H_{r,1}$ and $\Delta H_{r,2}$ fixed as in [1]. A second constraint on the molar fraction of D is also included as

$$\frac{c_D}{c_A + c_B + c_C + c_D} \leq 0.1.$$

Scaling the constraint functions, noting that the steady-state concentrations are implicit functions of $\mathbf{u}$, and placing the problem in standard form then leads to the following experimental optimization problem:

$$\begin{aligned}
\underset{\mathbf{u}}{\text{minimize}} \quad & \phi_p(\mathbf{u}) := 0.004(u_1^2 + u_2^2) - \tfrac{c_C(\mathbf{u})^2 (u_1+u_2)^2}{u_1 c_{A,in}} \\
\text{subject to} \quad & g_{p,1}(\mathbf{u}) := \tfrac{Q(\mathbf{u})}{110} - 1 \leq 0 \\
& g_{p,2}(\mathbf{u}) := 10 \tfrac{c_D(\mathbf{u})}{c_A(\mathbf{u}) + c_B(\mathbf{u}) + c_C(\mathbf{u}) + c_D(\mathbf{u})} - 1 \leq 0 \\
& 1 \leq u_1 \leq 50 \\
& 1 \leq u_2 \leq 50.
\end{aligned}$$

A model of the experimental system is assumed to be available and is obtained by using different values for the parameters k_1, k_2, and $c_{A,in}$, as done in [1].

[1] G. François and D. Bonvin (2013). Use of convex model approximations for real-time optimization via modifier adaptation. Ind. Eng. Chem. Res., 52(33): 11614-11625.

5.7 Test Problem #7: Maximizing Production in a Batch Reactor with a Reversible Reaction

Original code provided by: G. A. Bunin

Test File Specs: `algotest([0 0],40,0.01,[],[-1 -1],[1 1],[-0.2884 -1],0.1,[],[],algonum)`

Main Files: `phipeval.m` (required), `phimod.m` (model)

Problem Description: This problem is taken from the case study example of [1], where the reversible reaction

$$A \underset{k_2}{\overset{k_1}{\rightleftharpoons}} B$$

is carried out in a batch reactor, with the goal of maximizing the end-time production of B, or the concentration $c_{B,end}$. The task of the user is to find the optimal temperature profile, constrained to lie between 293K and 323K, that achieves this goal. As finding an entire profile is an infinite-dimensional problem, the author proposes a polynomial parameterization of the profile:

$$T(\tau) = 308 + 15[a_1 + a_2(1 - 2\tau)],$$

where the coefficients a_1 and a_2 essentially become the decision variables, with τ being the normalized time, defined as the time divided by the total batch time, the latter fixed here as 2.5 hours. From this definition, it may be shown that the constraints on the temperature will be met if and only if the constraints $-1 \leq a_1 \pm a_2 \leq 1$ hold. So as to simplify this problem to a minimization with only bound constraints, the following choice of decision variables is used

$$u_1 = a_1 + a_2, \; u_2 = a_1 - a_2,$$

with the bound constraints $-1 \leq u_1 \leq 1, \; -1 \leq u_2 \leq 1$ equivalent to $-1 \leq a_1 \pm a_2 \leq 1$.

The experimental optimization problem in standard form then reads as:

$$\begin{array}{ll} \underset{\mathbf{u}}{\text{minimize}} & \phi_p(\mathbf{u}) := -c_{B,end}(\mathbf{u}) \\ \text{subject to} & -1 \leq u_1 \leq 1 \\ & -1 \leq u_2 \leq 1. \end{array}$$

A model of the experimental system is assumed to be available and is obtained by using different values for the parameters that may be subject to error [1].

[1] C. Georgakis (2009). A model-free methodology for the optimization of batch processes: Design of dynamic experiments. 7th IFAC International Symposium on Advanced Control of Chemical Processes (ADCHEM), Istanbul: 644-649.

5.8 Test Problem #8: Maximizing Production in a Fed-Batch Reactor with Three Reactions

Original code provided by: G. A. Bunin

Test File Specs: `algotest([0 0 0],60,0.01,[],[-1 -0.5 -1.5],[1 0.5 1.5],[0.1953 0.3770 -1.0027],0.1,[],[1 1 1 1],algonum)`

Main Files: `phipeval.m` (required), `geval.m` (required), `phimod.m` (model)

Problem Description: This problem is a slightly modified version of the problem presented in [1], where the reactions

$$\begin{aligned} A + B &\xrightarrow{k_1} C \\ 2B &\xrightarrow{k_2} D \\ C &\xrightarrow{k_3} E \end{aligned}$$

take place in a fed-batch reactor, the reactant B being fed continuously over the course of the batch. The goal of the process is to find the feeding profile that maximizes the amount of C produced at the end of the batch ($c_{C,end}$). So as to avoid solving an infinite-dimensional optimization problem, the author proposes a parameterization that uses the first decision variable, u_1, to define (in hours) the total batch time:

$$t_b = 1 + 0.5u_1,$$

with u_1 constrained to lie between -1 and 1. Another two decision variables, u_2 and u_3, are then used to define the feeding profile by the polynomial

$$\begin{aligned} v(\tau) &= 30(1-\tau)\left(1 + \tfrac{2}{3}\omega(\tau)\right) \\ \omega(\tau) &= 6u_3\tau^2 + (2u_2 - 6u_3)\tau - 2u_2, \end{aligned}$$

with τ being the normalized time, defined as the time divided by the total batch time. So as to ensure that the profile meet certain required restrictions, the constraints $-1 \leq \omega(\tau) \leq 1$, $\forall \tau \in [0,1]$ are imposed. The additional constraints $-1.5 \leq 4u_2 + 3u_3 \leq 1.5$ are also introduced so as to ensure that the amount of B fed is between 10 and 20 gram-moles.

Because $\omega(\tau)$ achieves the value of $-2u_2$ for $\tau = 0$, it follows that any $u_2 \notin [-0.5, 0.5]$ will lead to a violation of the profile constraints, and so these limits may be imposed as the bounds on u_2. For u_3, the limits $-1.5 \leq u_3 \leq 1.5$ are chosen heuristically, as all tested profiles with values outside this range violated the constraints, and so it seems unlikely that the optimum would lie outside this range.

The experimental optimization problem that results is then written, in standard form, as

$$\begin{aligned} \underset{\mathbf{u}}{\text{minimize}} \quad & \phi_p(\mathbf{u}) := -c_{C,end}(\mathbf{u}) \\ \text{subject to} \quad & g_1(\mathbf{u}) := \max_{\tau\in[0,1]} \omega(\tau, \mathbf{u}) - 1 \leq 0 \\ & g_2(\mathbf{u}) := -1 - \min_{\tau\in[0,1]} \omega(\tau, \mathbf{u}) \leq 0 \\ & g_3(\mathbf{u}) := 4u_2 + 3u_3 - 1.5 \leq 0 \\ & g_4(\mathbf{u}) := -4u_2 - 3u_3 - 1.5 \leq 0 \\ & -1 \leq u_1 \leq 1 \\ & -0.5 \leq u_2 \leq 0.5 \\ & -1.5 \leq u_3 \leq 1.5. \end{aligned}$$

A model of the experimental system is assumed to be available and is obtained by using different values for k_1, k_2, and k_3.

[1] G. M. Troup and C. Georgakis (2013). Process systems engineering tools in the pharmaceutical industry. Comput. Chem. Eng., 51(5): 157-171.

5.9 Test Problem #9: Batch-to-Batch Tuning of a Temperature-Tracking Model-Predictive Controller

Original code provided by: G. A. Bunin

Test File Specs: `algotest([0 0.7],40,0.1,[],[-2 0],[2 1],[0.9492 0.6416],1,[],[],algonum)`

Main Files: `phipeval.m` (required), `phimod.m` (model)

Problem Description: The tuning of a model-predictive controller (MPC) in the experimental water-tank system described in [1] is addressed. The run-to-run iterative controller tuning strategy as outlined in [2] is pursued, with the problem here being a modified, simulated version of the first experimental case study of [2]. The MPC controller is tuned by modifying two parameters - the logarithm of the output tracking weight and the bias-update filter gain - with the desired objective being to minimize the sum of the norms of the tracking error and the aggressiveness of the control action during the batch profile where the controller follows a preset temperature profile. The performance metric is denoted by the implicit function $J(\mathbf{u})$ and is measured at the end of each batch.

The programmed MPC uses a dynamic matrix control strategy to compute an optimal control action to track the preset tank temperature profile, with the temperature of the water fed to the jacket surrounding the tank used as the control variable. In calculating the control action, the MPC is provided the (inaccurate) linear model

$$G(s) = \frac{0.2}{2s^2 + s + 0.2},$$

while the true dynamic experimental system, also assumed linear, behaves according to the law

$$G_p(s) = \frac{0.84s + 0.315}{s^3 + 3.993s^2 + 4.157s + 0.326}.$$

So as to add some realism to the problem in the form of output measurement noise but without adding an intrinsic stochastic element into the definition of J, the dynamic measurements used by the controller in computing the bias update are corrupted by an alternating sequence ± 0.1, which acts like zero-mean noise but is deterministic.

The output weight of the MPC objective function is allowed to vary between 0.01 and 100, which leads to $-2 \leq u_1 \leq 2$, while the bias-update filter gain is naturally fixed between zero and unity, leading to $0 \leq u_2 \leq 1$. The resulting experimental optimization problem is stated as

$$\begin{array}{rl} \underset{\mathbf{u}}{\text{minimize}} & \phi_p(\mathbf{u}) := J(\mathbf{u}) \\ \text{subject to} & -2 \leq u_1 \leq 2 \\ & 0 \leq u_2 \leq 1. \end{array}$$

The available model of the experimental system is obtained by setting $G_p(s) = G(s)$ when simulating the batch behavior (i.e., by assuming a perfect dynamic model).

[1] G. A. Bunin, F. V. Lima, C. Georgakis, and C. M. Hunt (2010). Model predictive control and dynamic operability studies in a stirred tank: rapid temperature cycling for crystallization. Chem. Eng. Commun., 197(5): 733-752.

[2] G. A. Bunin, G. François, and D. Bonvin (2013). A real-time optimization framework for the iterative controller tuning problem. Processes, 1(2): 203-237.

5.10 Test Problem #10: Iterative Tuning of a Fixed-Order Controller in a Torsional System

Original code provided by: G. A. Bunin

Test File Specs: `algotest([1 2.77 -2.6 1 0.5],100,0.001,[],[0 0 -5 -2 -1],[5 5 5 2 1],[4.812 0.3826 -2.9531 0.1292 -0.5116],0.05,[],[2 2],algonum)`

Main Files: `phipeval.m` (required), `geval.m` (required), `phimod.m` (model)

Problem Description: This problem is concerned with the tuning of a fixed-order controller for the torsional system [1] located in the Laboratoire d'Automatique of the École Polytechnique Fédérale de Lausanne (EPFL) in Lausanne, Switzerland, and is derived from the second case study reported in [2]. The control system considered is single-input-single-output (SISO), with the voltage of the motor acting as the input variable to control the angular position of the top of the torsional system's three disks. The dynamic law relating the change in disk position and the change in motor voltage was obtained by experimental identification in the laboratory, performed by Z. Emeđi and A. Nicoletti, with the linear transfer function describing the system chosen as

$$G_1(z) = \frac{0.0011z^4 - 0.0012z^3 + 0.0038z^2 - 0.0009z}{z^6 - 4.814z^5 + 10.47z^4 - 13.22z^3 + 10.21z^2 - 4.579z + 0.928},$$

with a sampling time of 0.04 seconds. A model for a slightly different configuration, with a greater load placed on the top disk, was also obtained as

$$G_2(z) = \frac{0.0004z^4 + 0.0015z^3 + 0.0036z^2 - 0.0008z}{z^6 - 4.651z^5 + 9.935z^4 - 12.47z^3 + 9.678z^2 - 4.413z + 0.924}.$$

The controller is tasked with following a periodic sinusoidal profile that repeats itself every 20 seconds, allowing for the controller parameters to be retuned between periods so as to achieve better performance. The controller employed is a fixed-order controller with 5 degrees of freedom that act as the decision variables:

$$K(z) = \frac{u_1 z^2 + u_2 z + u_3}{z^2 + u_4 z + u_5},$$

with the full closed-loop transfer function then being given by $\frac{K(z)G_1(z)}{1+K(z)G_1(z)}$ for the real system and by $\frac{K(z)G_2(z)}{1+K(z)G_2(z)}$ for the model. The metric by which controller performance is judged and which is to be minimized, denoted by J, is defined as

the average absolute tracking error during a single period. In the case that the magnitude of the output value goes over twice the amplitude of the setpoint, the output is saturated at this boundary for the rest of the time period. Physically, this may be seen as locking the system once the controller strays too far from the setpoint, which generally is a sign of instability and would lead to very large cost function values if allowed.

The search for the optimal controller is restricted to those controllers that are stable, which, following a Jury analysis [2], leads to the following constraints on the parameters that define the controller poles:

$$-1 \leq u_5 \leq 1,\ -1 + u_5^2 \leq u_4 - u_4 u_5 \leq 1 - u_5^2.$$

The bound constraints are chosen as $0 \leq u_1 \leq 5,\ 0 \leq u_2 \leq 5,\ -5 \leq u_3 \leq 5,\ -2 \leq u_4 \leq 2,\ -1 \leq u_5 \leq 1$, the latter two following implicitly from the constraints on controller stability and the others being chosen heuristically for this problem in particular - typically, the domains of tuning parameters in controller tuning are not clearly fixed, but they are fixed here to make the problem compatible with the database.

The resulting experimental optimization problem, in standard form, becomes

$$\begin{aligned} \underset{\mathbf{u}}{\text{minimize}} \quad & \phi_p(\mathbf{u}) := J(\mathbf{u}) \\ \text{subject to} \quad & g_1(\mathbf{u}) := u_4 - u_4 u_5 - 1 + u_5^2 \leq 0 \\ & g_2(\mathbf{u}) := -u_4 + u_4 u_5 - 1 + u_5^2 \leq 0 \\ & 0 \leq u_1 \leq 5 \\ & 0 \leq u_2 \leq 5 \\ & -5 \leq u_3 \leq 5 \\ & -2 \leq u_4 \leq 2 \\ & -1 \leq u_5 \leq 1. \end{aligned}$$

[1] Educational Control Products (2008). Manual for Model 205/205a: Torsional Control System.

[2] G. A. Bunin, G. François, and D. Bonvin (2013). A real-time optimization framework for the iterative controller tuning problem. Processes, 1(2): 203-237.

5.11 Test Problem #11: Minimizing the Steady-State Production Cost of a Gold Cyanidation Leaching Process

Original code provided by: Zhang Jun

Test File Specs: `algotest([52 18],40,2,.001,[10 5],[80 20],[31.0563 11.6218],200,.04,[],algonum)`

Main Files: `phipeval.m` (required), `gpeval.m` (required), `phimod.m` (model), `gmod.m` (model)

Auxiliary Files: `model_single.m` (required)

Problem Description: A slightly adapted version of the gold cyanidation leaching optimization problem presented in [1] is used here. The process itself is described as a continuous stirred tank reactor with the reactions

$$\begin{aligned} 2\mathrm{Au} + O_2 + 4\mathrm{CN}^- + 2H_2O &\rightarrow 2[\mathrm{Au(CN)_2}]^- + 2OH^- + H_2O_2, \\ 2\mathrm{Au} + 4\mathrm{CN}^- + H_2O_2 &\rightarrow 2[\mathrm{Au(CN)_2}]^- + 2OH^-, \end{aligned}$$

with gold ore and sodium cyanide being the two feed components. The steady state of the process is then characterized by the equations

$$\begin{aligned} &\frac{Q_s}{M_s}(C_{s,0} - C_s) - r_{Au} = 0, \\ &\frac{Q_l}{M_l}(C_{l,0} - C_l) + \frac{M_s}{M_l} r_{Au} = 0, \\ &\frac{Q_l}{M_l}(C_{cn,0} - C_{cn}) + \frac{Q_{cn}}{M_l} - r_{cn} = 0, \\ &r_{Au} = k_1 \left[C_s - C_{s,\infty}(\overline{d})\right]^{k_2} C_{cn}^{k_3} C_o^{k_4}, \\ &r_{cn} = k_5 C_{cn}^{k_6}, \end{aligned}$$

where $k_1, k_2, k_3, k_4, k_5, k_6$ denote the kinetic model parameters. The quantities Q_s, Q_l, Q_{cn} denote (respectively) the flow rates of the gold ore, the liquid in the gold feed, and the sodium cyanide. The quantities M_s, M_l denote the holdups in the gold ore and liquid in the leaching tank, respectively. The letter C denotes the corresponding concentrations, with $C_{s,\infty}(\overline{d})$ in particular denoting the residual gold concentration in the ore, a function of the average particle diameter $\overline{d}$:

$$C_{s,\infty}(\overline{d}) = 0.357(1 - 1.49e^{-0.0176\overline{d}}).$$

The concentration of the dissolved oxygen in the liquid, C_o, can be controlled and is chosen as one of the decision variables, together with the flow rate of the sodium cyanide – i.e., $\mathbf{u} = (Q_{cn}, C_o)$. The decision-variable domain is restricted by the limits $10\mathrm{kg/h} \leq Q_{cn} \leq 80\mathrm{kg/h}$ and $5\mathrm{mg/kg} \leq C_o \leq 20\mathrm{mg/kg}$, where it is noted that the limits on Q_{cn} have been slightly compressed from those given in [1] to avoid those parts of the decision-variable space where the solution clearly does not lie or where operating the process would be extremely expensive due to the cost function rising towards extremely large values.

The steady-state production cost, given in Chinese RMB per hour, is defined as

$$(u_1 + C_{cn,0}Q_l)P_{cn} + Q_l C_{cn}(\mathbf{u})P_{cn,d} + Q_l u_2 P_o + Q_s C_s(\mathbf{u})P_{Au},$$

with P denoting the different price factors.

Additionally, it is desired that the gold recovery be at least 75%, which is expressed by the experimental constraint

$$\frac{C_{s,0} - C_s(\mathbf{u})}{C_{s,0}} \geq 0.75.$$

Combining all of these specifications then yields the experimental optimization problem:

$$
\begin{aligned}
\underset{\mathbf{u}}{\text{minimize}} \quad & \phi_p(\mathbf{u}) := (u_1 + C_{cn,0}Q_l)P_{cn} + Q_l C_{cn}(\mathbf{u})P_{cn,d} + Q_l u_2 P_o + Q_s C_s(\mathbf{u})P_{Au} \\
\text{subject to} \quad & g_{p,1}(\mathbf{u}) := \frac{C_s(\mathbf{u}) - C_{s,0}}{C_{s,0}} + 0.75 \leq 0 \\
& 10 \leq u_1 \leq 80 \\
& 5 \leq u_2 \leq 20.
\end{aligned}
$$

The model of the simulated problem is obtained by applying a 10% error to the kinetic parameters $k_1, k_2, k_3, k_4, k_5, k_6$.

[1] Z. Jun, M. Zhizhong, J. Runda (2015). Real-time optimization based on SCFO for gold cyanidation leaching process. Chem. Eng. Sci., 134: 467-476.

6 Submit Problem/Algorithm

Researchers who have previously simulated experimental optimization problems are highly encouraged to submit their examples to this database, as every new problem benefits the community by enlarging the test domain of the different tested algorithms. The only rules for submission are that your problem should be placable in the standard form

$$
\begin{aligned}
\underset{\mathbf{u}}{\text{minimize}} \quad & \phi_p(\mathbf{u}) & \\
\text{subject to} \quad & g_{p,j}(\mathbf{u}) \leq 0, & j = 1, \ldots, n_{g_p} \\
& g_j(\mathbf{u}) \leq 0, & j = 1, \ldots, n_g \\
& u_i^L \leq u_i \leq u_i^U, & i = 1, \ldots, n_u
\end{aligned}
$$

and that it be based on a real-world case study (and is not just a constructed mathematical example). Please write to `gene.a.bunin@ccapprox.info` if you have a submission request, as this process is not automated.

You are also welcome to submit new algorithms, or to submit refined versions of already stored ones – in the latter case, you may choose to keep the old version(s) or to only provide test results for the latest version if you feel this to be the best or most representative. The only rule is that the algorithm be compatable with the testing code used here – i.e., that it can be incorporated into the main algorithm file in a manner that yields a new set of decision variables when the command

```
1 [u(k+2,:),output] = algo(u,phiphat,gphat,sigmaphi,sigmag,uL,uU,
    algonum,input);
```

is executed. Not all algorithms are applicable to all classes of problems, and so your algorithm will be tested for those problem classes that it is applicable for. There is no need for you to test it yourself, although you are more than welcome to do so if you want to verify that the results reported here are correct (all of the code necessary for testing being provided). As with the test problems, researchers are highly encouraged to submit their algorithms, whether they be performant or not, as this will help in deciphering the links between algorithmic characteristics and performances for different metrics and problem types. It is important as well to keep in mind that this is not a competition, though some

competitive spirit as we try to collectively write better and better algorithms is certainly encouraged. To submit your algorithm, please mail its code, together with a basic algorithm description, to `gene.a.bunin@ccapprox.info`. This is not a single-step submission process, however, and additional interaction to finalize the algorithm details will follow.

Finally, all of the algorithms considered here are open-source and available for use/modification by any user of the database. Commercial algorithms are not accepted for testing as they are inaccessible to the general user.

7 Update Log

February 22, 2017

Version 0.91.3 of the *SCFO Solver* added to the list of algorithms.

January 10, 2016

Version 0.91.2 of the *SCFO Solver* added to the list of algorithms. Testing files for the database modified to remove dlmread and dlmwrite routines. The ExpOpt database upgrades to Version 1.3.

June 8, 2015

Test Problem #11: Minimizing the Steady-State Production Cost of a Gold Cyanidation Leaching Process added to the database.

June 5, 2015

Some changes have been made to the code of *Test Problem #10: Iterative Tuning of a Fixed-Order Controller in a Torsional System* so as to avoid overly large cost function values for certain decision variable choices.

March 11, 2015

The ExpOpt database upgrades to Version 1.2. The testing of algorithms is now automated and is continuously being carried out on dedicated computers.

December 31, 2014

As 2014 ends, the ExpOpt database upgrades to Version 1.1. It is now possible to view the plots corresponding to the results of the individual trials for any algorithm/problem by clicking on the algorithm name in the results table.

December 24, 2014

Test Problem #10: Iterative Tuning of a Fixed-Order Controller in a Torsional System added to the database.

December 12, 2014

Test Problem #9: Batch-to-Batch Tuning of a Temperature-Tracking Model-Predictive Controller added to the database.

December 9, 2014

Test Problem #8: Maximizing Production in a Fed-Batch Reactor with Three Reactions added to the database.

December 1, 2014

Test Problem #7: Maximizing Production in a Batch Reactor with a Reversible Reaction added to the database.

November 17, 2014

Test Problem #6: Maximizing Production in a Continuous Stirred-Tank Reactor added to the database.

November 12, 2014

Test Problem #5: Minimizing the Overall Pumping Effort in the "Trois Bacs" added to the database.

October 15, 2014

Constraint Adaptation added to the list of algorithms.

October 12, 2014

Test Problem #4: Maximizing Electrical Efficiency in a Solid Oxide Fuel Cell Stack added to the database.

October 2, 2014

Initial version of the database officially launched, with three test problems and four algorithms.