



This document is part of LIBPF

Paolo Greppi - 3iP © Copyright 2004 - 2008

All rights reserved; do not distribute without permission.

LIBPF

Case Study salt & pepper

Summary

Introduction.....	2
Problem description.....	2
Flowsheet.....	2
Main data.....	3
Problem translation in LIBPF.....	3
Input.....	3
Flowsheet output.....	5
Resolution.....	5
Results.....	6



Introduction

LIBPF (**LIB**rary for **P**rocess **F**lowsheeting) is a modelling tool to rapidly prototype and deploy small-footprint custom applications implementing computations for training, process engineer support, on-line process diagnostic, and data reconciliation. For more information regarding **LIBPF**, please see the website <http://www.libpf.com>.

The purpose of the **Salt and Pepper LIBPF demo** is to illustrate the process flowsheet solving capabilities of LIBPF, plus the front-end (User Interface), database repository and calculation server, based on a simple, fictitious process with reaction, separation and recycle. The prototype shows basic functionality, but it could be easily extended to become a tool for off-line process diagnostic and reconciliation.

Problem description

The process is the hypothetical production of pepper from salt; the alchemic recipe is based on a proprietary spell which transform the commodity inorganic “salt” in the high-valued spice “pepper”.

The incomplete reaction takes place in the watery phase, where the reactant is soluble. The reaction product is insoluble in water and precipitates, which makes possible to recover most of the product on a belt filter.

The mother liquor is recycled to recover the unreacted salt and increase the overall yield.

Flowsheet

The process flowsheet is portrayed in figure 1.

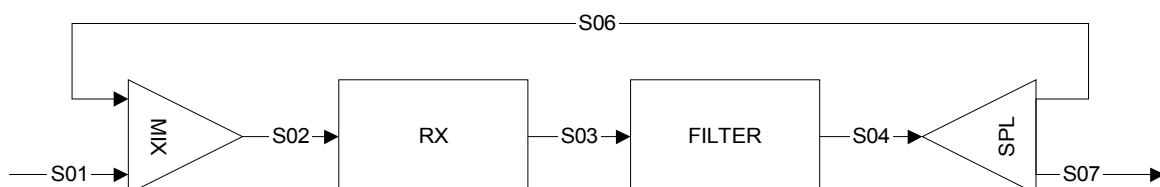


Figure 1 - Process Flowsheet



Main data

The feed to the process is 1000 kg/h of watery solution with 5% of salt.

The reaction converts 1 kg salt into 1 kg pepper, but the one-pass yield is 90 %.

The solid pepper is filtered with a 99 % recovery to obtain a cake with 10 % imbibition solution.

The mother liquor is assumed to have the same salt content as the cake imbibition solution.

The mother liquor purge is 5 %.

Problem translation in LIBPF

Input

The problem is specified in LIBPF in four steps:

- 1) subclass a new class from `flowsheet` to represent the case

```
class sale_pepe : public modelBase, public flowsheet<zero_zero> {
private:
    static std::string type_;
    void maketables(long);
public:
    sale_pepe(const std::string &t, const std::string &d);
    sale_pepe(long cid);

    void makeuserassembly(std::list<assignment *>::iterator &p);
    void setup(void);

    const std::string &type(void) const { return type_; }
}; // sale_pepe
```

- 2) define case connectivity in `maketables` virtual method of new type

```
void sale_pepe::maketables(long CID) {
    if (CID == -1) {
        makeVertex<mixer> ("MIX", "Mixer", CID);
        makeVertex<genflashN1> ("RX", "Reactor", CID,
            boost::assign::map_list_of<std::string, long>("nReactions", 1),
            boost::assign::map_list_of<std::string, std::string>("embeddedTypeReactions[00]",
"reactionYield"));
        makeVertex<gensep<2> > ("FILTER", "Belt filter", CID);
        makeVertex<splitter> ("SPL", "Three-way valve", CID,
            boost::assign::map_list_of<std::string, long>("nStreams", 2));
```



```

makeEdge<stream_L> ("S01", "Feed", "source", "out", "MIX", "in", CID);
makeEdge<stream_L> ("S02", "Reactor feed", "MIX", "out", "RX", "in", CID);
makeEdge<stream_L> ("S03", "Reactor product", "RX", "out", "FILTER", "in", CID);
makeEdge<stream_L> ("S04", "Product", "FILTER", "out1", "sink", "in", CID);
makeEdge<stream_L> ("S05", "Mother liquor", "FILTER", "out2", "SPL", "in", CID);
makeEdge<stream_L> ("S06", "Recycled mother liquor", "SPL", "out1", "MIX", "in",
CID);
makeEdge<stream_L> ("S07", "Purged mother liquor", "SPL", "out2", "sink", "in",
CID);
}
} // sale_pepe::maketables

```

3) set up model inputs in setup virtual method of new type

```

void sale_pepe::setup(void) {
    flowsheet<zero_zero>::setup();

    // Setting input variables
    O("S01")->S("flowoption")->set("Mx");
    O("S01")->O("Tphase")->Q("mdot")->set(1000.0, "kg/h");
    O("S01")->O("Tphase")->Q("x[00]")->set(0.95);
    O("S01")->O("Tphase")->Q("x[01]")->set(0.05);
    O("S01")->O("Tphase")->Q("x[02]")->set(0.0);

    O("RX")->O("reactions", 0)->I("keycomp")->set(1);
    O("RX")->O("reactions", 0)->Q("coeff", "Water")->set(0.0);
    O("RX")->O("reactions", 0)->Q("coeff", "Table Salt")->set(-1.0);
    O("RX")->O("reactions", 0)->Q("coeff", "Pepper")->set(1.0);
    O("RX")->O("reactions", 0)->Q("z")->set(0.9);

    // filter efficiencies
    O("FILTER")->Q("outSplit[00]", "Water")->set(0.002);
    O("FILTER")->Q("outSplit[00]", "Table Salt")->set(0.002);

    // 1% pepper loss in filter
    O("FILTER")->Q("outSplit[00][02]")->set(0.99);

    O("SPL")->Q("outSplit[00]")->set(0.95);

    // define cut stream
    addcut("S06");

    // make selected outputs visible in the UI
    O("FILTER")->Q("outSplit[00]", "Water")->setOutput();
    O("FILTER")->Q("outSplit[00]", "Table Salt")->setOutput();
    O("RX")->O("reactions", 0)->Q("conv")->setOutput();
    O("S06")->O("Tphase")->Q("mdot")->setOutput();
} // sale_pepe::setup

```



- 4) enter feedback specifications to be converged iteratively in makeuserassembly virtual method of new type

```
void sale_pepe::makeuserassembly(std::list<assignment*>::iterator &p) {
    long int i(0);

    MakeScaledAssignment(
        *O("FILTER")->Q("outSplit[00]", "Water"),
        (*O("S04")->O("Tphase")->Q("mdot")) * 0.1 / (*O("S03")->O("Tphase")->Q("mdot")),
        "Cake humidity", scalingmode::rational, 0.0, 1.0, Qdouble(100.0));
    MakeScaledAssignment(
        *O("FILTER")->Q("outSplit[00]", "Table Salt"),
        (*O("S04")->O("Tphase")->Q("mdot")) * 0.1 / (*O("S03")->O("Tphase")->Q("mdot")),
        "Equating salt split to water split", scalingmode::rational, 0.0, 1.0,
        Qdouble(100.0));
} // sale_pepe::makeuserassembly
```

Flowsheet output

The connectivity is represented within LIBPF as the Directed Cyclic Graph shown in figure 2.

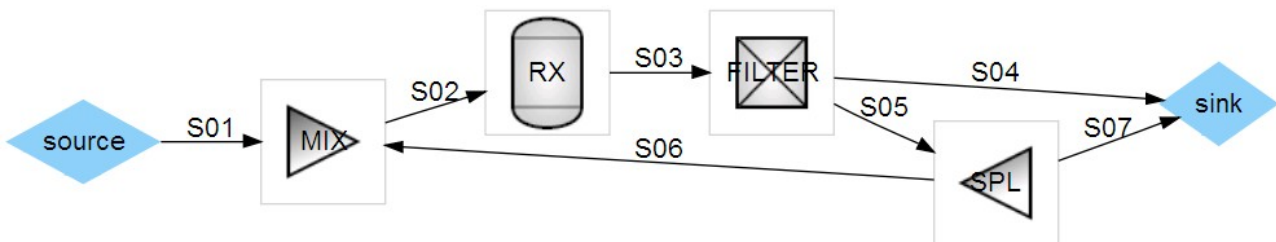


Figure 2 - Internal representation of flowsheet

Resolution

The command `addcut("S06")` in the setup method cuts stream S06 and makes the graph a Directed Acyclic Graph shown in figure 3.

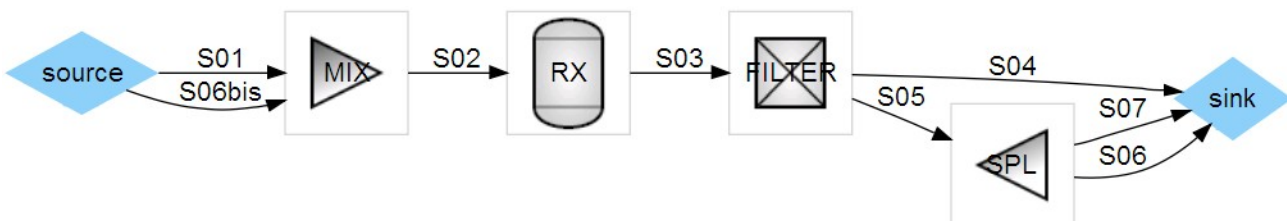


Figure 3 - Internal representation of flowsheet after cutting stream S06



Stream S06bis is a new feedstream, whose values are initialized to default values (10000 kg/h equimolar); the DAG can be then solved sequentially.

For maximum speed only two sequential iterations are performed, then the solution proceeds simultaneously.

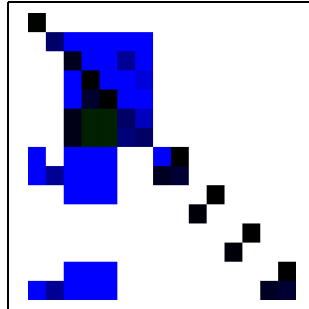


Figure 4 - Jacobian for simultaneous resolution

For the simultaneous solver this problem exposes 15 unknowns, and gets solved in 9 iterations; the jacobian is shown in figure 4, where the blue cells represent tiny elements.

The timings in typical modern hardware are 2 s for the initial resolution and 1 s for the subsequent resolutions.

Results

The mass balance results are:

		S01	S02	S03	S04	S05	S06	S06bis	S07
		Feed	Reactor feed	Reactor product	Product	Mother liquor	Recycled mother liquor	Recycled mother liquor cut	Purged mother liquor
Pressure	bar	1.01325	1.01325	1.01325	1.01325	1.01325	1.01325	1.01325	1.01325
Temperature	K	298.15	298.15	298.15	298.15	298.15	298.15	298.15	298.15
	°C	25	25	25	25	25	25	25	25
AMW	kg/kmol	18.0	18.0	18.0	18.0	18.0	18.0	18.0	18.0
Mass flow	kg/h	1000	18951	18951	55	18896	17951	17951	945
Water	kg/h	950	18895	18895	6	18890	17945	17945	944
Table Salt	kg/h	50	55	6	0	6	5	5	0
Pepper	kg/h	0	0	50	50	1	0	0	0
Water	kg/kg	95.00%	99.71%	99.71%	9.97%	99.97%	99.97%	99.97%	99.97%
Table Salt	kg/kg	5.00%	0.29%	0.03%	0.00%	0.03%	0.03%	0.03%	0.03%
Pepper	kg/kg	0.00%	0.00%	0.26%	90.03%	0.00%	0.00%	0.00%	0.00%