FME HighEFF

Centre for an Energy Efficient and Competitive Industry for the Future

Deliverable D2.1_2018.07
Heat exchanger modelling tool opportunities

Delivery date: 2018-12-17

Organisation name of lead beneficiary for this deliverable:

SINTEF ER

HighEFF- Centre for an Energy Efficient and Competitive Industry for the Future is one of Norway's Centre for Environment-friendly Energy Research (FME). Project co-funded by the Research Council of Norway and Industry partners. Host institution is SINTEF Energi AS.		
Dissemination Level		
PU	Public	X
RE	Restricted to a group specified by the consortium	

Deliverable number:	D2.1_2012.07
ISBN number:	
Deliverable title:	Heat exchanger modelling tool opportunities
Work package:	WP2.1 Heat Exchanges
Deliverable type:	MEMO
Lead participant:	SINTEF ER

Quality Assurance, status of deliverable		
Action	Performed by	Date
Verified (WP leader)	Geir Skaugen	20.12.2018
Reviewed (RA leader)	Armin Hafner	20.12.2018
Approved (dependent on nature of deliverable)*)		

*) *The quality assurance and approval of HighEFF deliverables and publications have to follow the established procedure. The procedure can be found in the HighEFF eRoom in the folder "Administrative > Procedures".*

Authors		
Author(s) Name	Organisation	E-mail address
Geir Skaugen	SINTEF Energy Research	Geir.skaugen@sintef.no
Brage Rugstad Knudsen	SINTEF Energy Research	brage.knudsen@sintef.no

Abstract

The first part of this memo contains the information presented at the COPRO Consortium meeting 2018 with some additional information and results.
The models that was used in this analysis were developed and prepared as a task in HighEFF RA 2.1
The results from a screening of a tubes-in-shell heat exchanger compared to a novel plate-type design for a given duty and desired maximum pressure loss indicated that for a plate type design a 90% core weight- and 85% core volume reduction could be possible.
The investigated case was a heat recovery heat exchanger used in an indirect bottoming cycle for power production for 150°C
The plate-type design will be followed up in HighEFF RA2.1 in cooperation with RA4 in 2019.

In the second part, experience on how derivative free optimisation framework NOMAD can be implemented for process and heat exchanger optimisation are discussed. Comparison with gradient based method in terms om time consumption and robustness are performed for a simple Rankine cycle.

Table of Contents

1 Introduction 3
2 The "mBundle" model 3
2.1 Description and operating conditions 3
2.2 Geometry calculations 6
2.3 Performance data 10
3 Alternative concepts 17
3.1 Previous work in COPRO and HighEFF 17
3.2 Spiral-in-shell concept 19
3.3 Plate-(no)-fin concept 20
4 Summary and conclusions 28
5 Part 2 – Memo on the implementation and experience of derivative-free optimisation in the process and heat exchanger modelling framework 29

1 Introduction

In COPRO WP2, an indirect cycle generating maximum of electric power from 150°C exhaust gas has been analysed and optimized with respect utilizing a reasonable fraction of available fan power and reasonable heat exchanger sizes. The indirect cycle is designed as pressurized water circuit between the warm exhaust and the hydro-carbon (HC) circuit as the power producing Rankine cycle. All heat exchangers in this process has been modelled quite detailed with a generic heat exchanger model that is integrated in the process optimisation. These are the Heat recovery heat exchanger (HRHE) that transfer heat from the exhaust gas to an indirect water circuit, the evaporator, condenser and recuperator in the HC circuit. This memo focuses on the heat recovery heat exchanger.

2 The "mBundle" model

2.1 Description and operating conditions

The mBundle-model was developed in cooperation with HighEFF RA2 as a simple and fast-solving heat exchanger suitable for studying various optimisation techniques, for studying the

sensitivity on design from choices and the accuracy of the underlying physical and thermo-hydraulic models and to investigate the possibilities of using variable tube geometry.

In COPRO WP1, the model has been used to do a screening of size/weight related to the suggested operating conditions from the indirect cycle process optimisation.

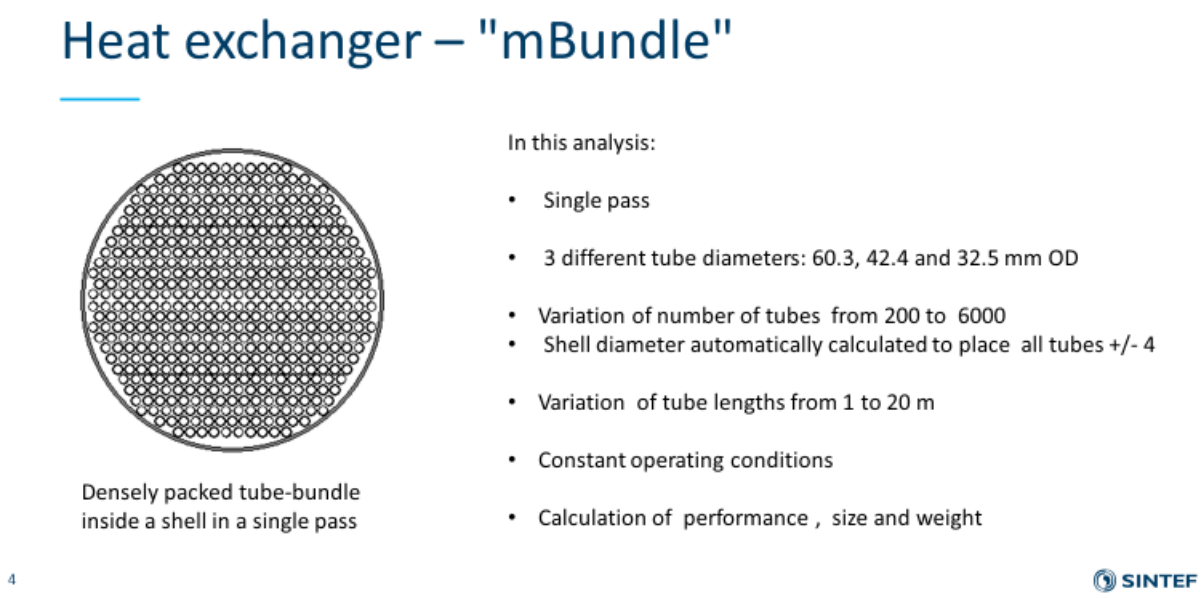


Figure 1 The mBundle model

Figure 1 describes the simple mBundle model. In the analysis the exhaust gas is flowing inside several parallel tubes that are placed inside a shell in a single pass design. Three tube diameters a used, 60.3, 42.4 and 32.5 mm outside diameter. These are plain tubes of carbon steel with wall thicknesses 3.95, 3.53, 2.88 mm, with no internal fins. The walls are assumed to be quite solid to withstand possible erosion from the flue gas.

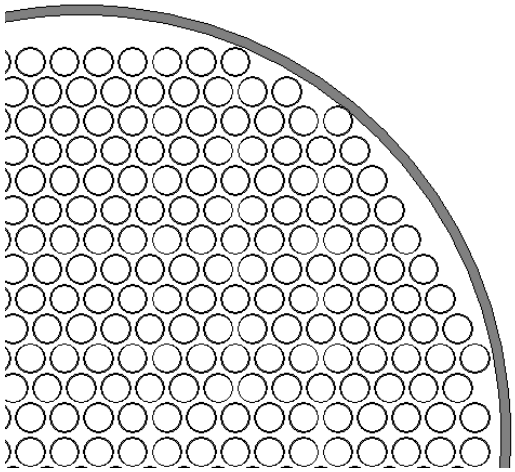


Figure 2 Details of tube stacking

In the screening, the number of tubes are varied from 200 to 600 an the shell ID calculated automatically by solving a tube-stacking algorithm within +/- 4 tubes. A fill ratio, defined here as a fraction of the shell "height" that is filled with tube rows is set to be 90%. The tubes are assumed to

be in a triangular pattern with equal tube distances of 1.15 times the outer tube diameter. Detail of the tube stacking is shown in Figure 2. The tube lengths are varied between 1 and 20 m. The process operating as inlet water and exhaust temperature and flow rate is held constant. The water operating conditions comes directly from the WP2 process optimisation. The performance of an "mBundle" is recorded for every combination of tube diameter, tube length and number of tubes.

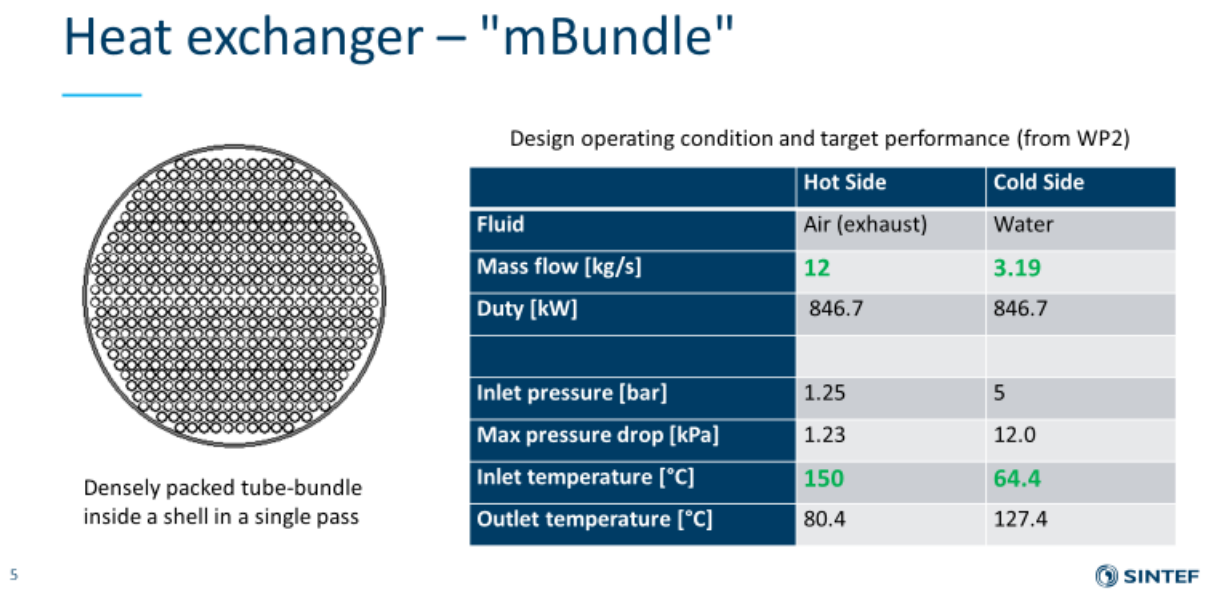


Figure 3 The operating conditions

Figure 3 show the operating conditions. This is for a modular case there the mass flow rate of 150°C exhaust gas is 12.0 kg/s. The flow rate in the indirect water circuit is 3.19 kg/s and the inlet temperature is 64.4°C. Since the exhaust temperature is 150°C, the water is pressurized to an inlet pressure of 5 bar to avoid boiling. At 5.0 bar, the boiling point of water is about 151°C. At design point the desired duty is 846.7kW which corresponds to an outlet water temperature of 127.4°C.

In the screening, the mBundle-performance will vary depending of the heat transfer surface and in cases with few, long tubes and small diameters, the pressure loss can be too high and boiling may occur. (However, this did not happen during this screening). The exhaust was treated as dry air in these simulations.

2.2 Geometry calculations

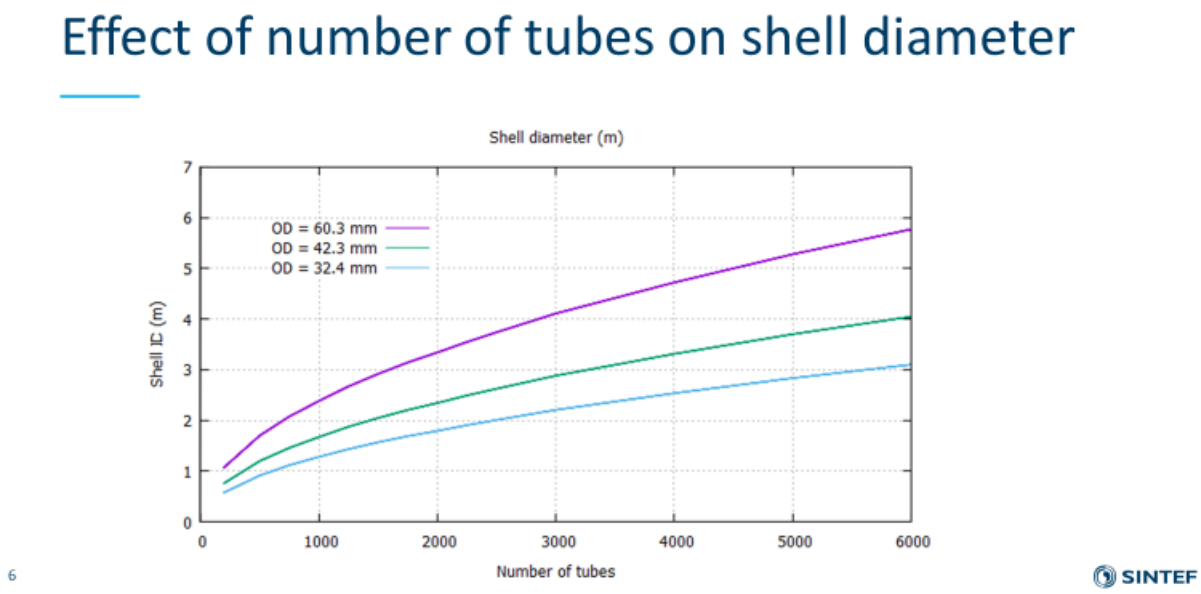


Figure 4 Calculated shell diameter as a function of number of tubes for outside tube diameter of 60.3, 42.3 and 32.4 mm

Figure 4 shows the relation between the number of tubes and the shell diameter. As seen, this is not linear since the number of tubes that are stacked inside the shell increase both horizontally and vertically. If a maximum accepted shell diameter is, say 4.0 m, the maximum number of tubes will be about 2800 for the largest diameters but 6000 and more for the smaller diameters.

Figure 5 show the relation between the specific heat transfer surface per. unit length for varying number of tubes and tube diameters. The heat transfer surface is in this heat exchangers defined as the sum of all the outside tube surfaces – or circumference as per unit length. This relation is linear.

In Figure 6 the total heat transfer surface is shown as a function of the number of tubes and the tube length. Here, only the 42.3 mm outside tube diameter is shown. As seen within the screening-range heat transfer surfaces between 200 and 10000 m² is "found". If there is a restriction of, say 4 m tube length inside the shell, the maximum heat transfer surface for this diameter that can be obtained is about 3000 m². This corresponds to a shell diameter of about 4.0 m as seen from Figure 7. Finally, in Figure 8 the corresponding weight of the heat exchanger core

is shown, ranging from 1 to 100 ton depending of the size. Much of weight is here related to the use of relative large tube wall thicknesses of about 2.5 to 3.5 mm as listed earlier

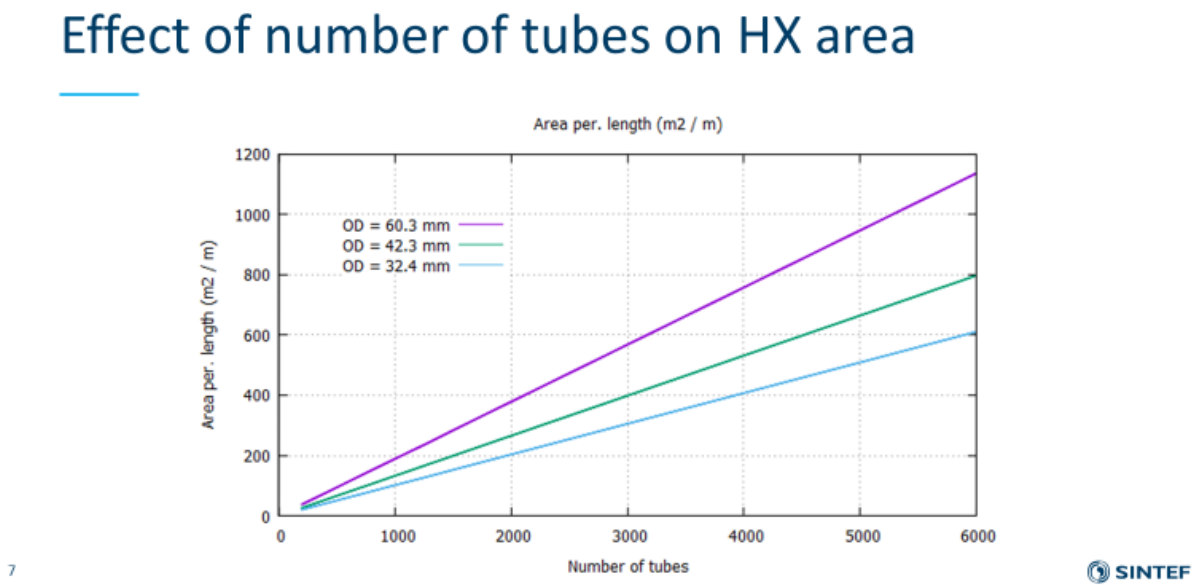


Figure 5 Heat exchanger area per unit length for different outside tube diameters and number of tubes

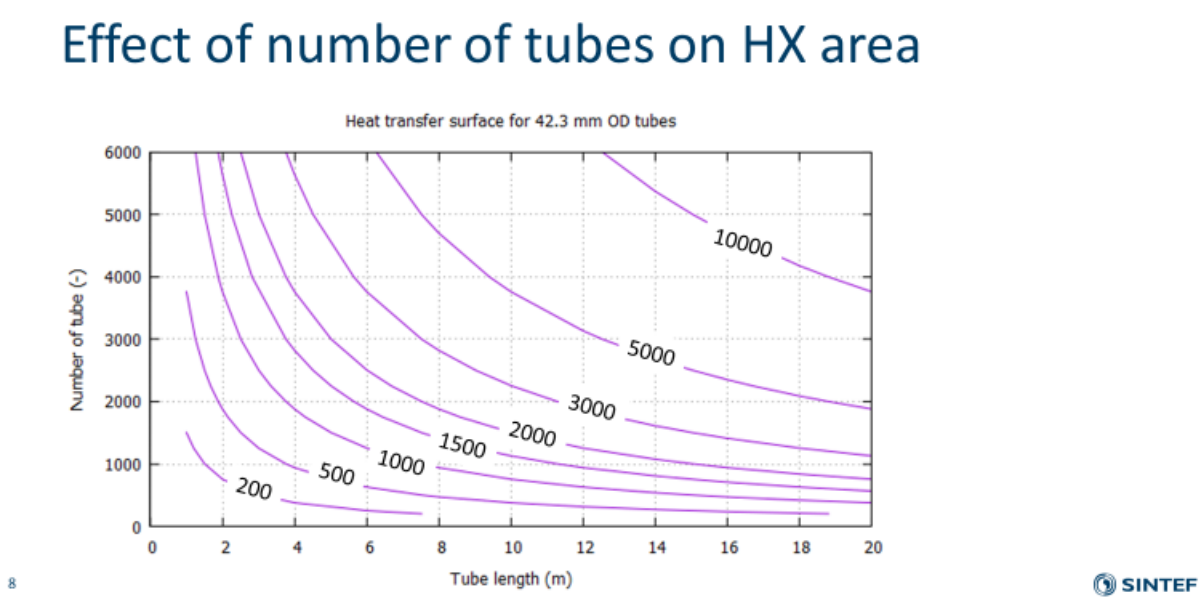


Figure 6 Iso lines showing heat transfer surface a function of tube length and number of tubes for tube OD of 42.3 mm

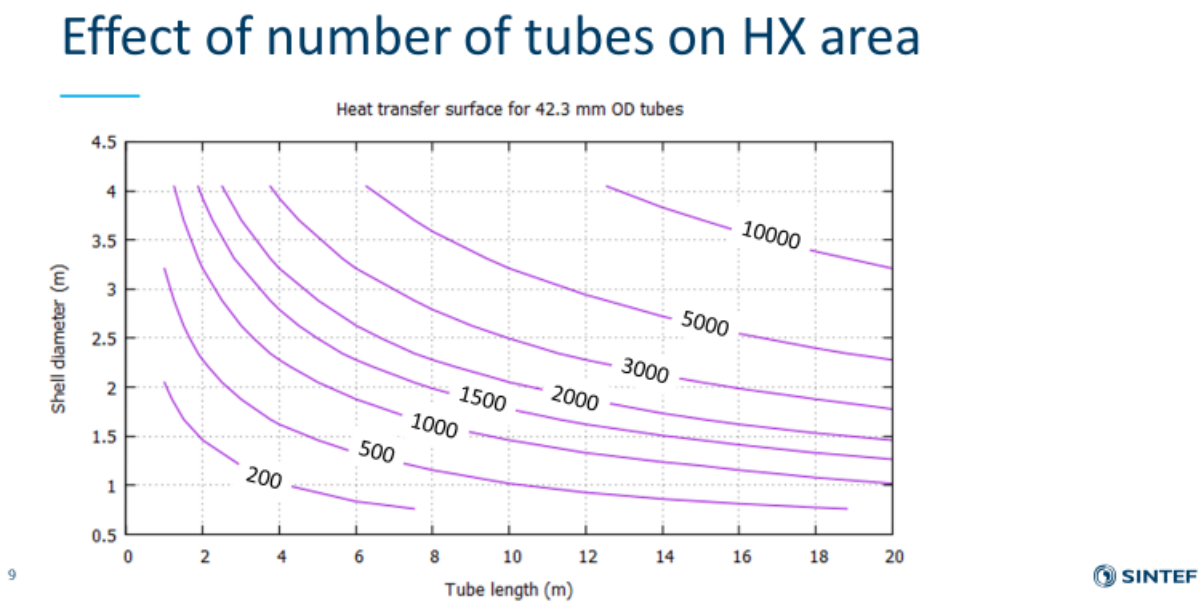


Figure 7 Heat exchanger surface as function of tube length and shell diameter for tubes with outside diameter of 42.3 mm

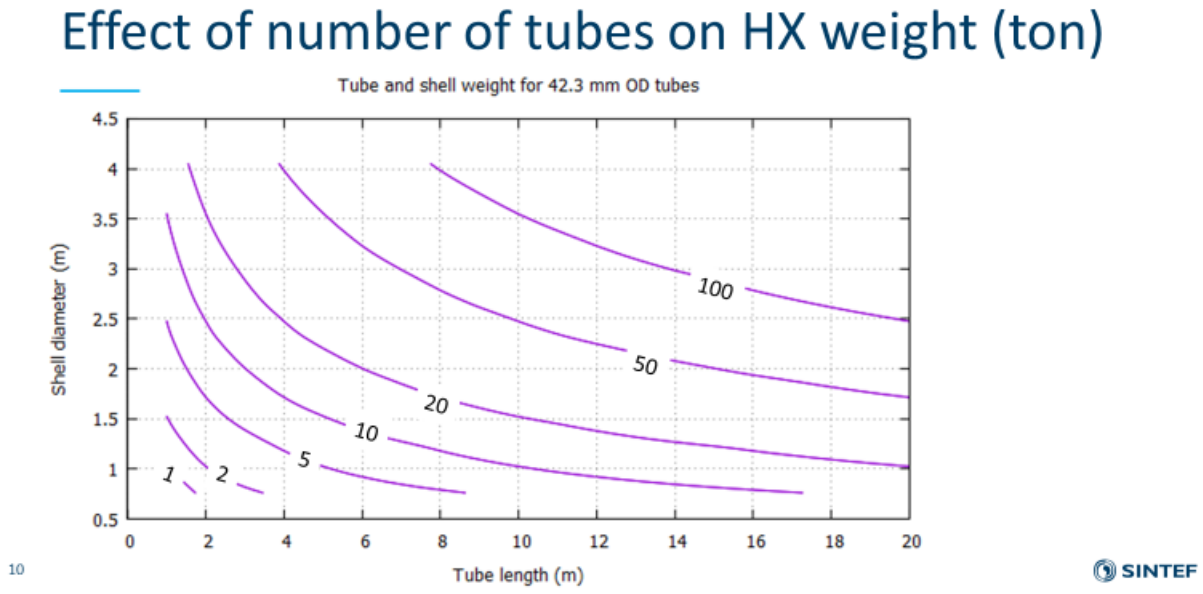


Figure 8 Calculated heat exchanger weight as a function of tube length and number of tubes (shell diameter) for tubes with outer diameter of 42.3 mm

2.3 Performance data

With tube length, number of tubes and tube diameter as input and with the inlet operating condition listed Figure 3, various performance maps can be generated.

All the performance maps are shown as a function of tube length and shell diameter. The shell diameter is a calculated value from the number of tubes and tube diameter given as input.

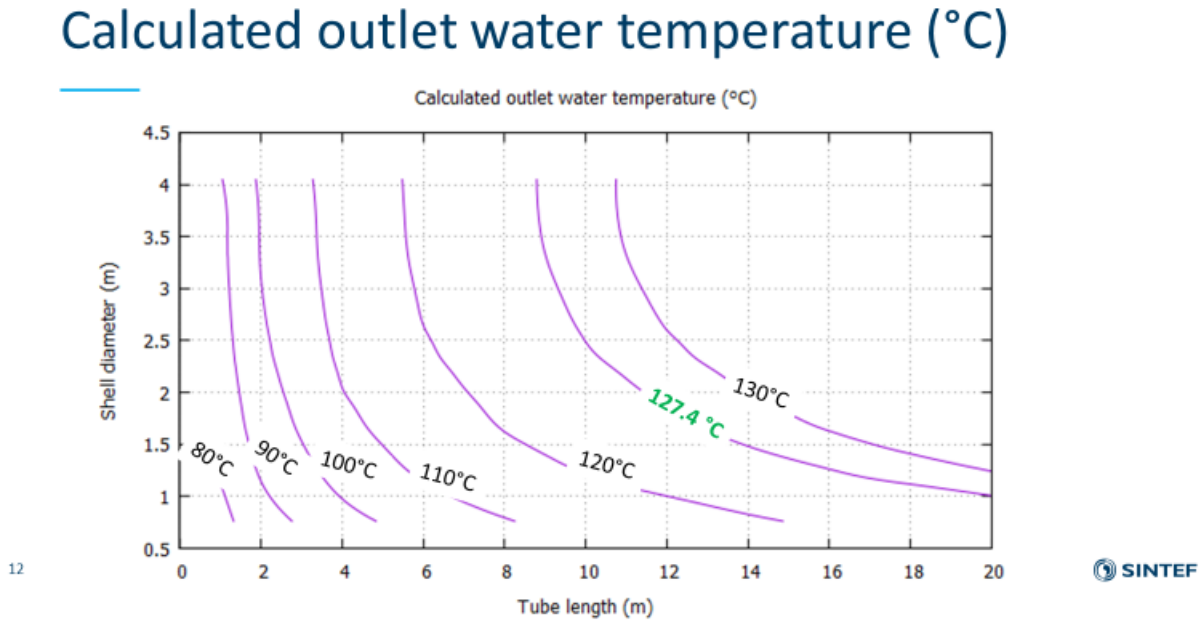


Figure 9 Outlet water temperature as a function of tube lengths and number of tubes (shell diameter)

In Figure 9 the iso-lines for the calculated outlet water temperature are shown. The inlet water temperature are 64.4°C for all cases. The design outlet water temperature of 127.4°C is labelled in green and as seen, a minimum of about 8.5 m tube length is will be required. We can also see that as the number of tubes increases, with increasing number of tubes, there is not more to gain as the iso-lines is steeper. From this graph only, a tube length of 10-12 m (possibly as to bundles in series) would be a best compromise giving a shell diameter of about 2.0 m.

The outlet temperature of 127.4°C comes from the process optimisation of COPRO WP2 where this water temperature is required as inlet to the hydrocarbon evaporator that is used in the power-producing cycle.

The companion graph, showing the capacity is Figure 10 where 127.4°C corresponds to ~846 kW.

In Figure 11 the corresponding heat transfer rate (UA-values) are shown. UA-values are often used to simplify the size of a heat exchanger when doing a process simulation of optimisation. The UA-value is defined as the ratio between heat exchanger "duty" and "temperature difference":

$$UA = \frac{\text{Duty}}{\text{Temperature difference}} = \frac{Q}{\Delta T}$$

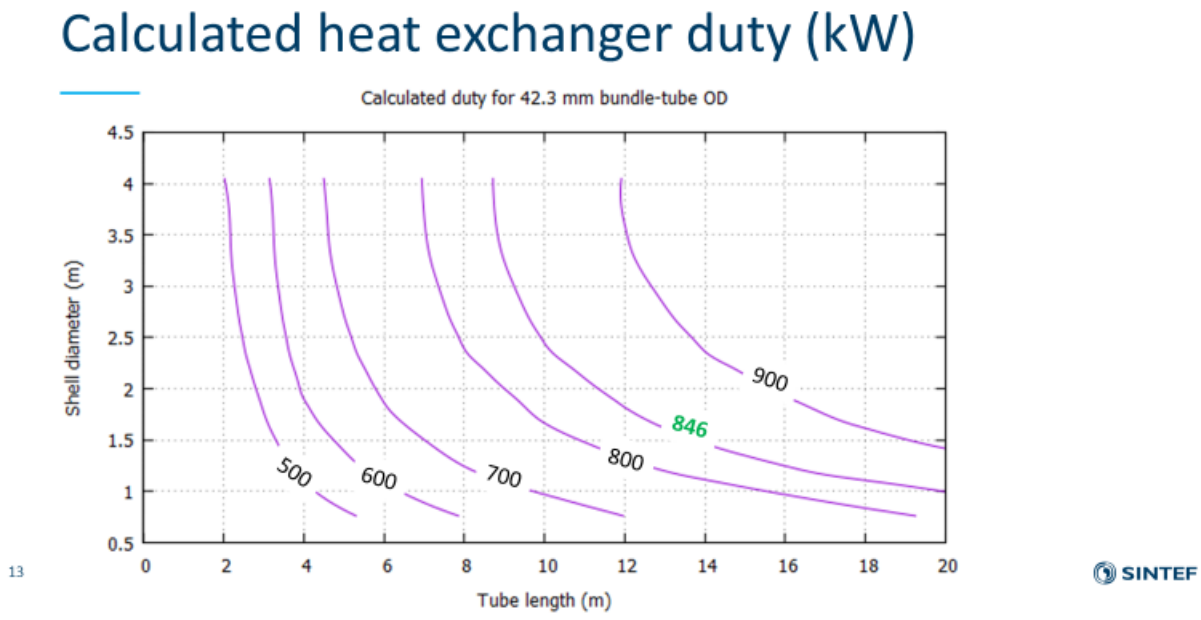


Figure 10 Heat exchanger capacity as a function of tube length and shell diameter

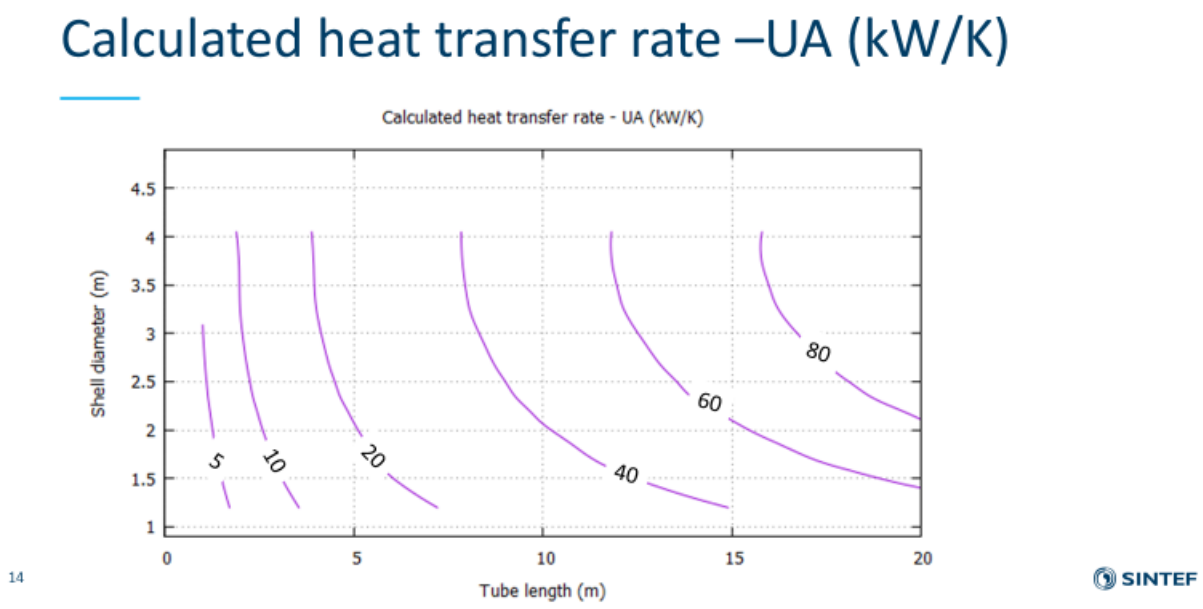


Figure 11 The heat transfer rate (UA) as a function of tube length and shell diameter

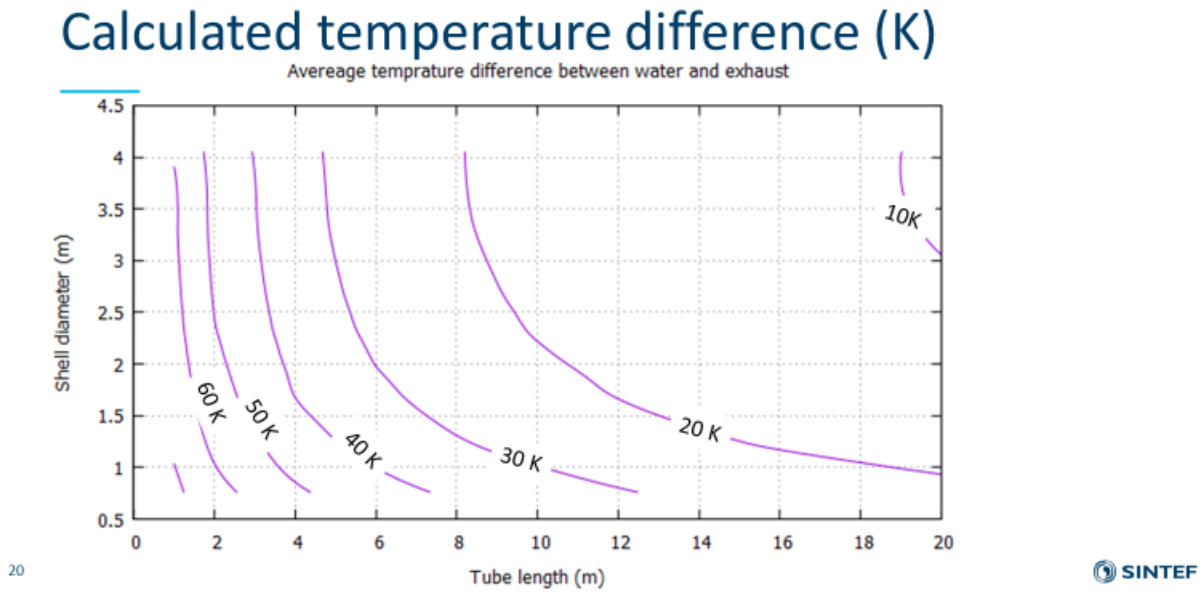


Figure 12 Average temperature difference

In Figure 12, the corresponding average temperature is shown. Here we see how a smaller sized heat exchanger really increased the temperature difference – or visa versa how large a heat exchanger need to be if a mean temperature difference of 10-15 K or less is desired based on some exergy loss minimisation optimisation.

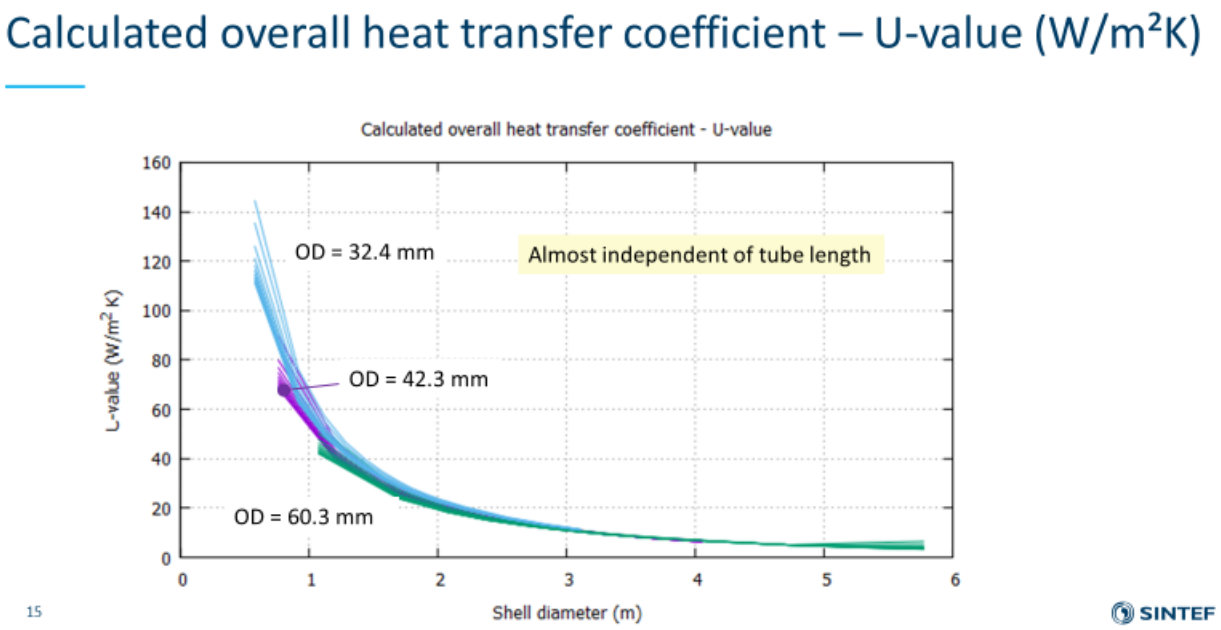


Figure 13 Obtained overall heat transfer coefficient related to outside heat transfer area and average stream temperature difference between exhaust and water.

The UA-values in Figure 11 is calculated from the calculated capacity divided by the arithmetic mean temperature difference between the cooled exhaust and heated water. From the UA-values, the U-values can directly obtained. This is shown in Figure 13

The U-values shown in Figure 13 we see how they are mosly dependent of the shell diameter. At small shell diameters , there are fewer tubes and thus a higher gas (and water) speed resulting in higher U-values. As the shell diameter (number of tubes) increases the heat transfer on both the water and exhaust side becomes very poor resulting in an overall heat transfer coefficient (U-value) of less than 20 W/m²K. We also see that it show little dependence of the tube length.

The HRHE that is investigated with the "mBundle" model would be part of a modular heat to power system where the size of the total system will matter.

From the same mapping, if a size restriction on both tube length and shell diameter is defined, the presented maps could indicate what the possibilities for heat recovery are. This is indicated in Figure 14. Here we can see that if we are limited to design with the coloured box, the maximum water temperature would only be slightly above 110°C. This would mean that there would be less heat available for the HC evaporator and less power generated. It also tell us that by integrating size, weight and cost as part of the KPI for the whole system, another optimum would possibly be found.

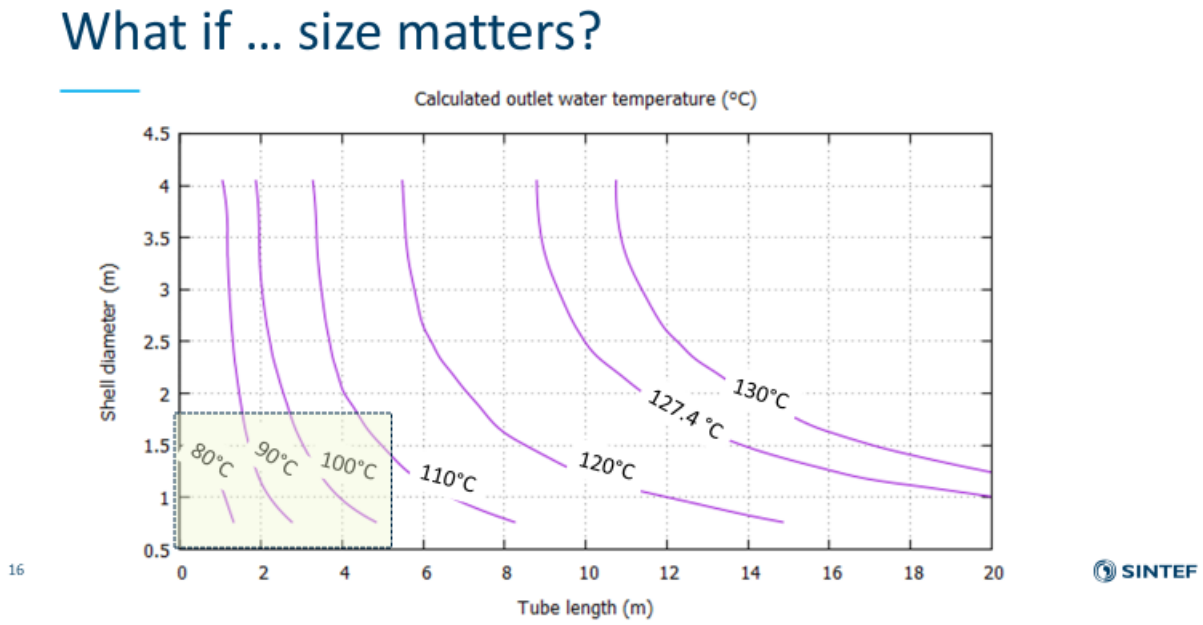


Figure 14 Size vs. performance

In Figure 15 and Figure 16 the exhaust side pressure loss is shown . In Figure 15 we can see how the pressure loss approaches a constant value for shell diameter above ~2.0 m. This is (probably) the inlet/outlet pressure loss to the bundle that seem to be more or less constant for this case. In Figure 16 some iso-pressure loss contours for the frictional pressure drop inside the tubes are shown for 42.3 mm tubes (Total subtracted 3.74 kPa). From the table in Figure 3, 1.23 kPa was listed as the desired maximum pressure loss (as calculated without inlet/outlet contributions) and is "easily" found to possible in the investigated range.

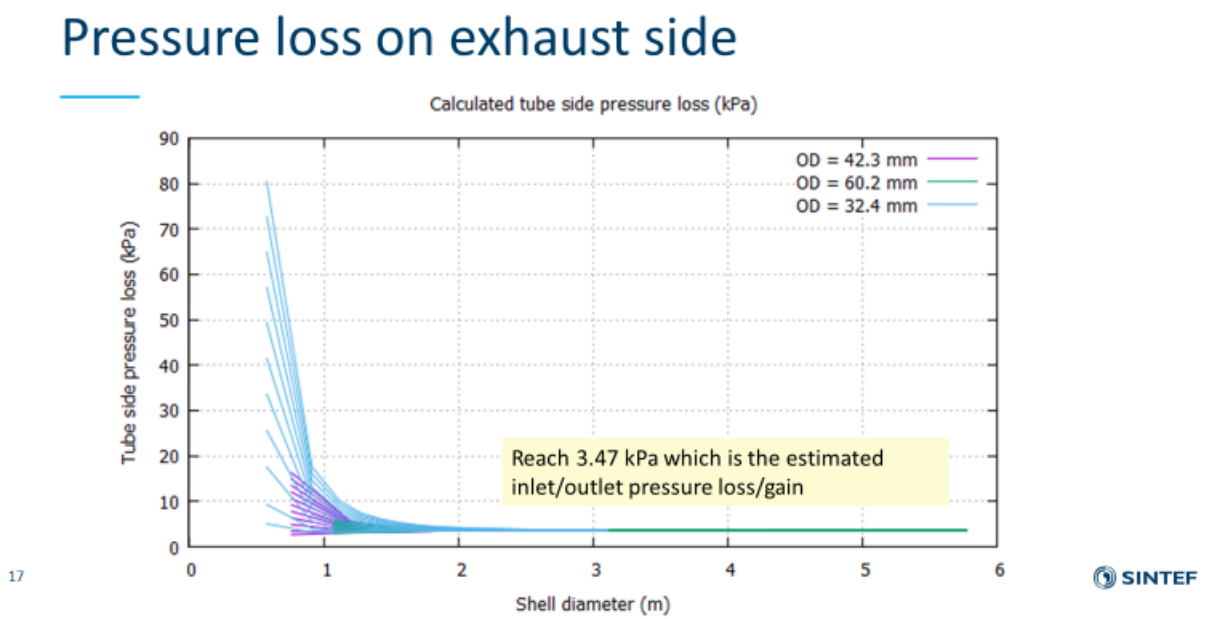


Figure 15 Pressure loss on the exhaust side

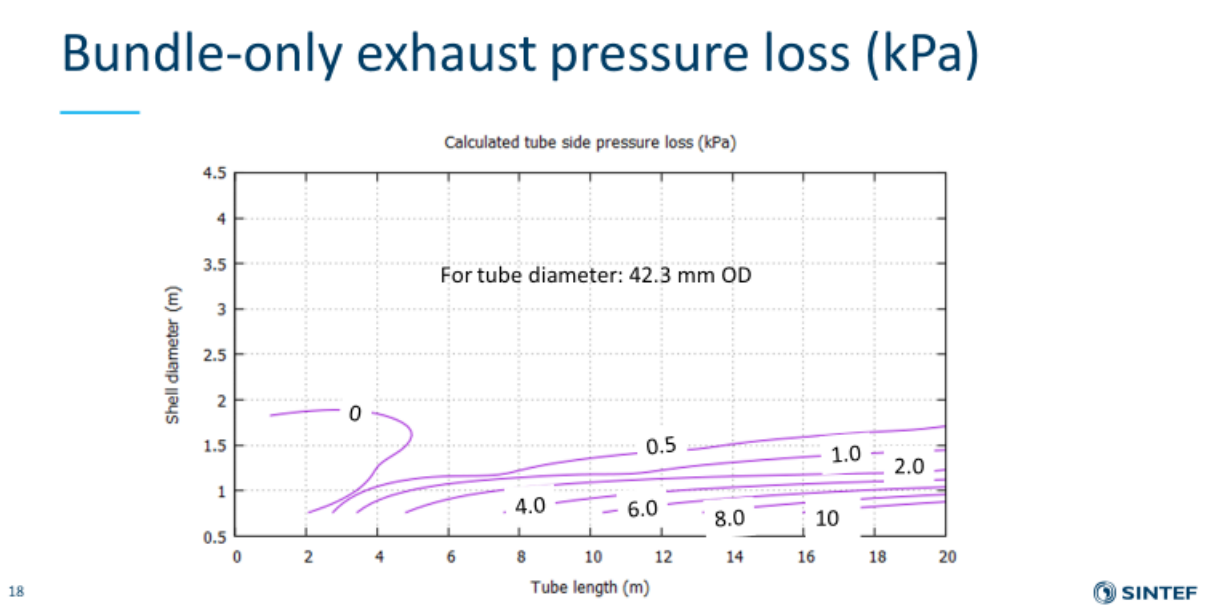


Figure 16 Exhaust side pressure loss for bundle- only

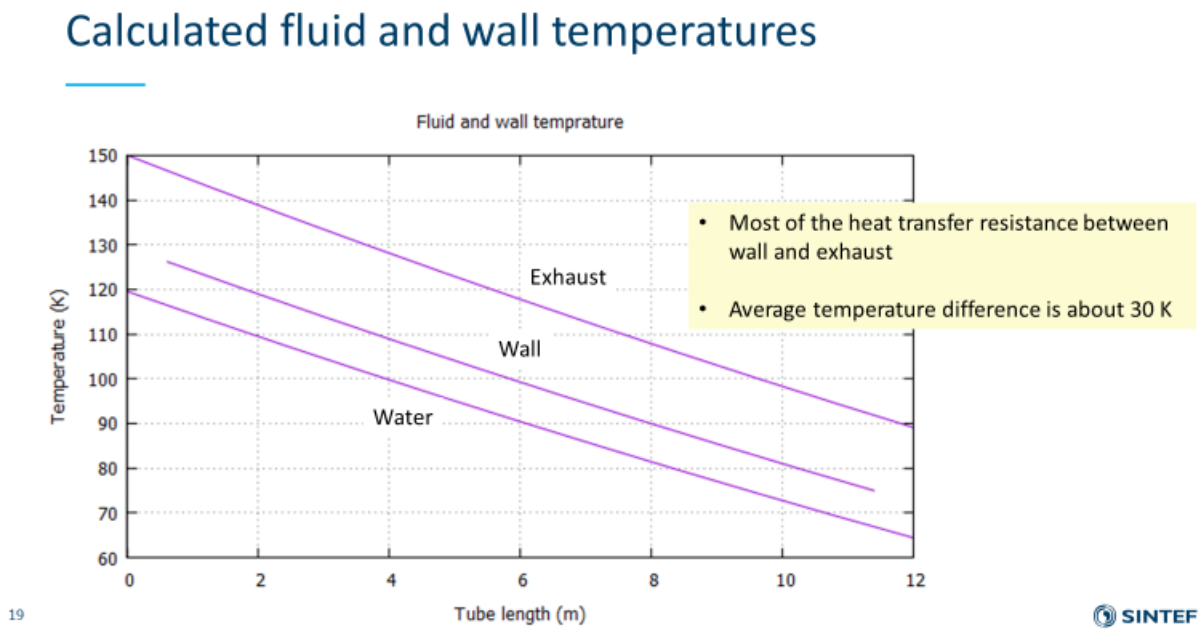


Figure 17 Heat transfer resistance in terms of temperature differences

The last Figure 17 of the performance graphs who the heat transfer resistance for a specific case with 12 m tube length and about 2.0 shell diameter (from 1500 43.3 mm tubes). Here we see the wall and fluid temperature profiles when the exhaust is cooled from 150°C to 90°C, while the water is heated from 64 to 120°C. The temperature difference between the exhaust and the wall are much higher than the difference between the water and wall which is another way of showing that most of the total heat transfer resistance between exhaust and water are on the exhaust side.

That was the reason why alternative concepts with varying heat transfer surface was investigated in COPRO.

3 Alternative concepts

3.1 Previous work in COPRO and HighEFF

In the first part of this work, a heat exchanger tube with variable perimeter along the flow length was investigated as a possibility to increase the heat transfer surface on the exhaust side in gas to water heat exchanger as the gas velocity decreased. A tube perimeter with a sine-wave pattern were investigated.

$$R(\theta, z) = (R_0 - A(z)) + A(z)\sin(N \cdot (\theta + B(z) \cdot \pi))$$

By letting the amplitude, *A* and the parameter *B* of the sine-wave be a function of the axial position in a tube circuit, a helix sin-wave pattern with varying perimeter can be described. The concept has previously been presented in COPRO and is summarized in Figure 18

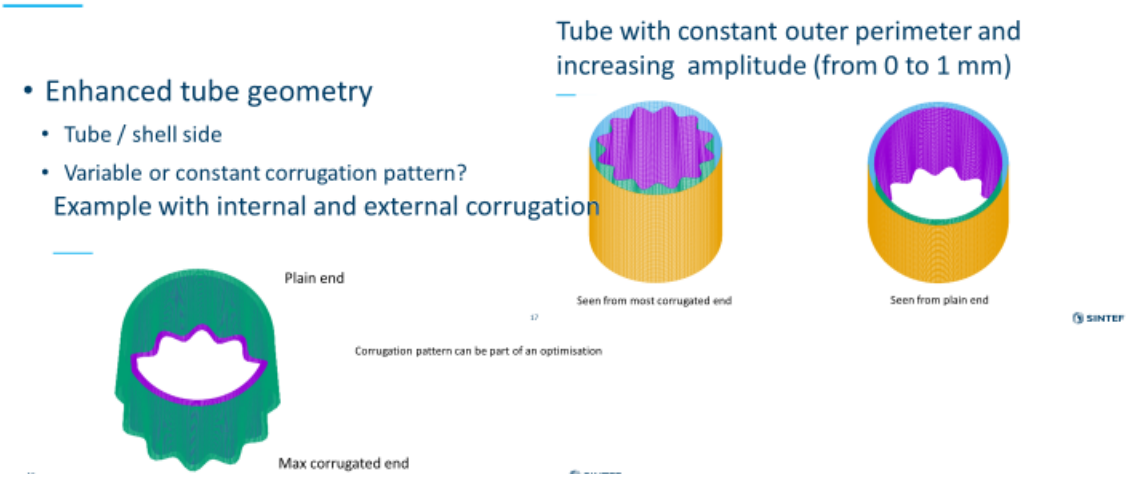


Figure 18 Simple concept for varying geometry HX

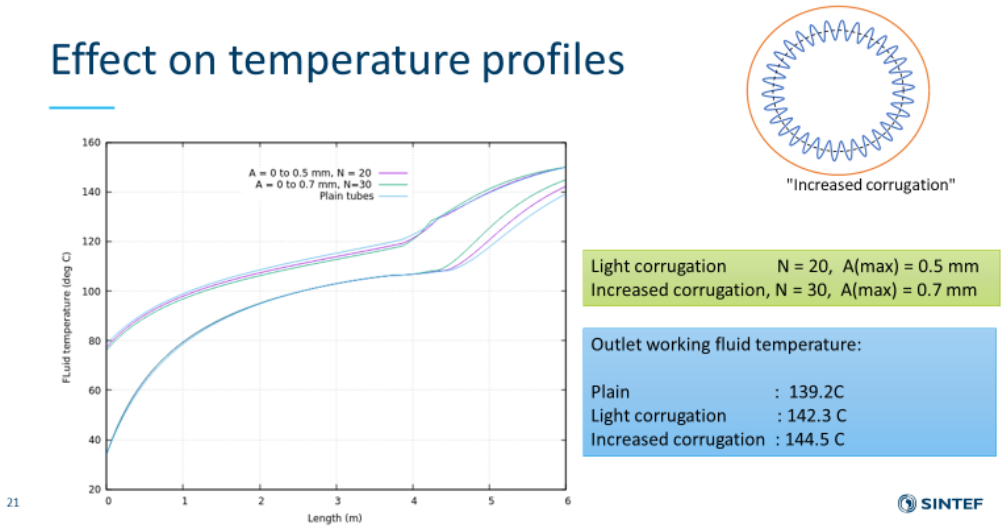


Figure 19 Previous results from use of variable geometry

The results previously shown in Figure 19 showed that there could be a potential of increasing the gas side tube perimeter along the tube. In the previous case, a mixture of propane and butane that was heated in the super-critical region was used as the working fluid and it could be heated additional 3-5 °C inside the same heat exchanger core size. The cost was increased pressure loss and an increase in tube weight. In that analysis standard heat transfer correlations for single phase flow were applied, but it was suggested as further work to investigate this more closely by computational fluid dynamic (CFD) analysis. This has been done in HighEFF RA.4 and the conclusions from that study showed that for single phase gas the "pockets" in the sine-wave circumference profile and an increased boundary layer thickness that had a negative effect on the effective heat transfer coefficient almost directly opposite that of the increased circumference and smaller hydraulic diameter. The increased boundary layer and thus increased heat transfer resistance is shown in Figure 20 below.

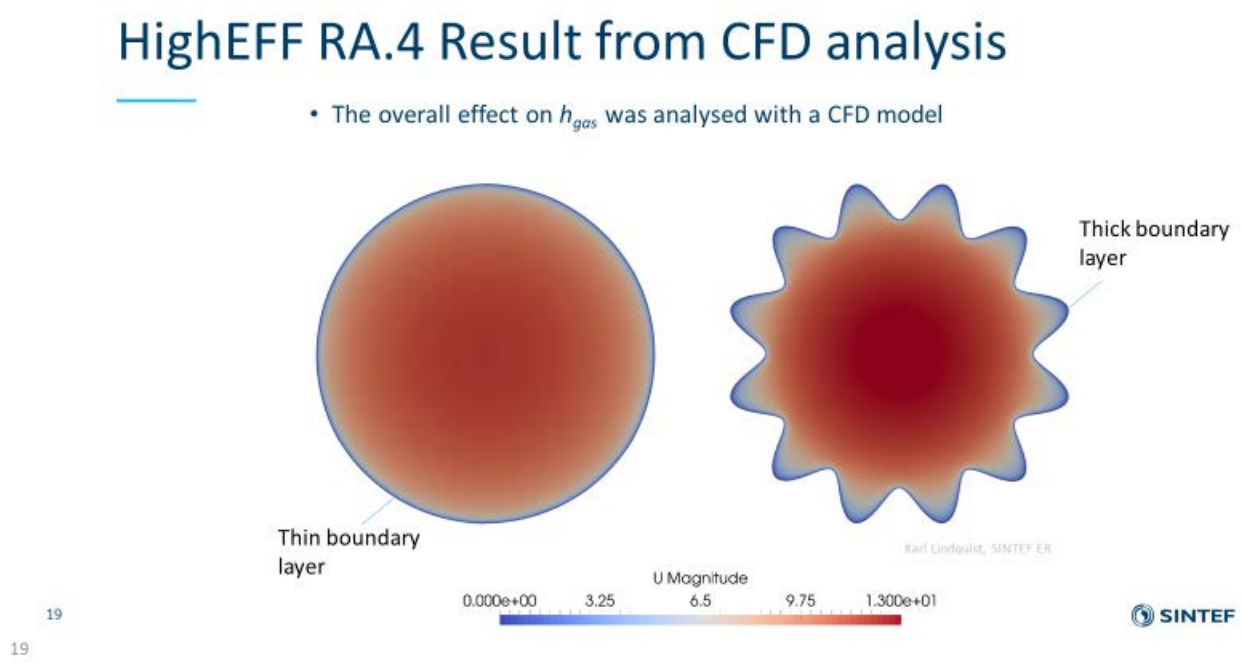


Figure 20 CDF analysis of a sine-wave tube perimeter

3.2 Spiral-in-shell concept

Another possibility of increasing the gas-side surface could be to use a spiral-in-shell concept as shown in Figure 21. The positive sides of such a case would be increased tube side heat transfer and area. In addition, the shell side heat transfer area would increase and due to a more cross flow design on the water side, the heat transfer would also increase.

The negative aspects would be increased pressure loss and probably increased weight.

This concept is not screened in the same as shown earlier, but for the same capacity the bundle length could be reduced significantly.

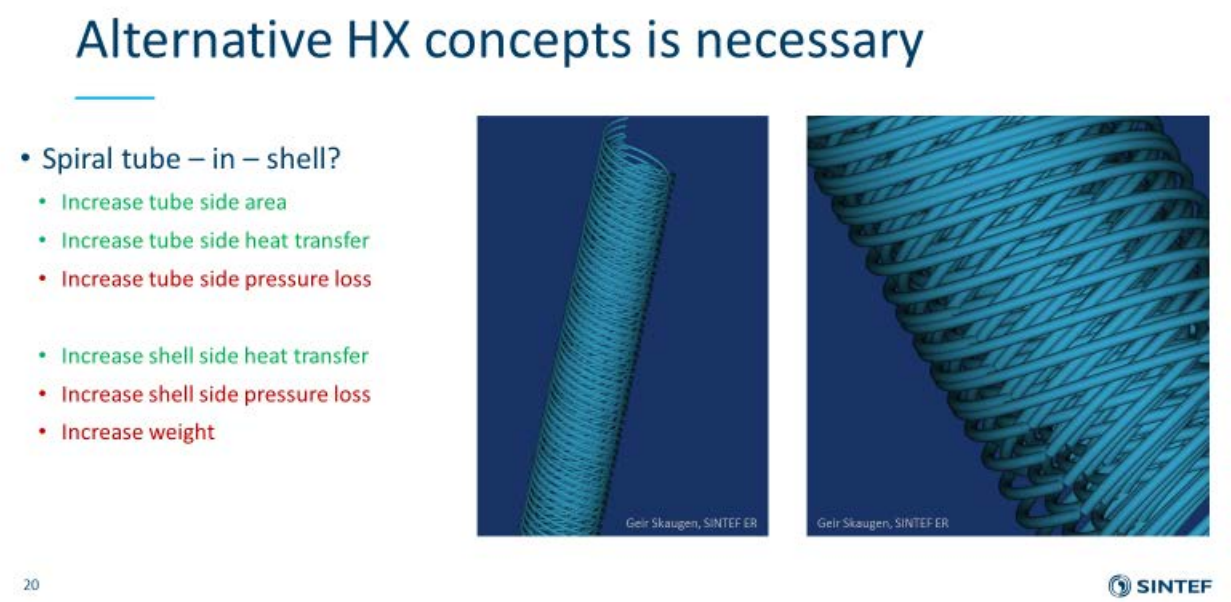


Figure 21 Spiral - in - shell concept

3.3 Plate-(no)-fin concept

As a second alternative a concept using stacked plates were briefly investigated. It is modelled as a plate-fin heat exchanger, but with only a few solid fins to keep the plates apart. In the simulations, 4 fins per. meter is used – 1.0 mm thick.

The setup used for the preliminary screening of this concept is shown in Figure 22.

Here, the plate distance (or the fin height) is used as a parameter when the plate length and number of layers are varied. The plate width is held constant as 1.0 m. The plate length is varied between 1.0 and 6.0 m while the number of stacked layers for each stream is varied between 50 and 300. The design is considered "single-banked" meaning that the number of layers for each stream is equal.

The "fin height" or here, the plate distance, is set to be 5.0 or 10.0 mm in all four combinations for the cold and warm stream.

The partition plates in this concept is set to be 2.0 mm.

A P(no)FHE is geometrically defined as "Depth" x "Width" x "Height" where the Depth means the plate length, the "Width" the plate width and the "Height" is the stacking height. The stacking height is a function of the total number of layers and is shown in Figure 23. The "compact" and "porous" designs refer to plate distance of 5.0 or 10 mm for warm and cold layers.

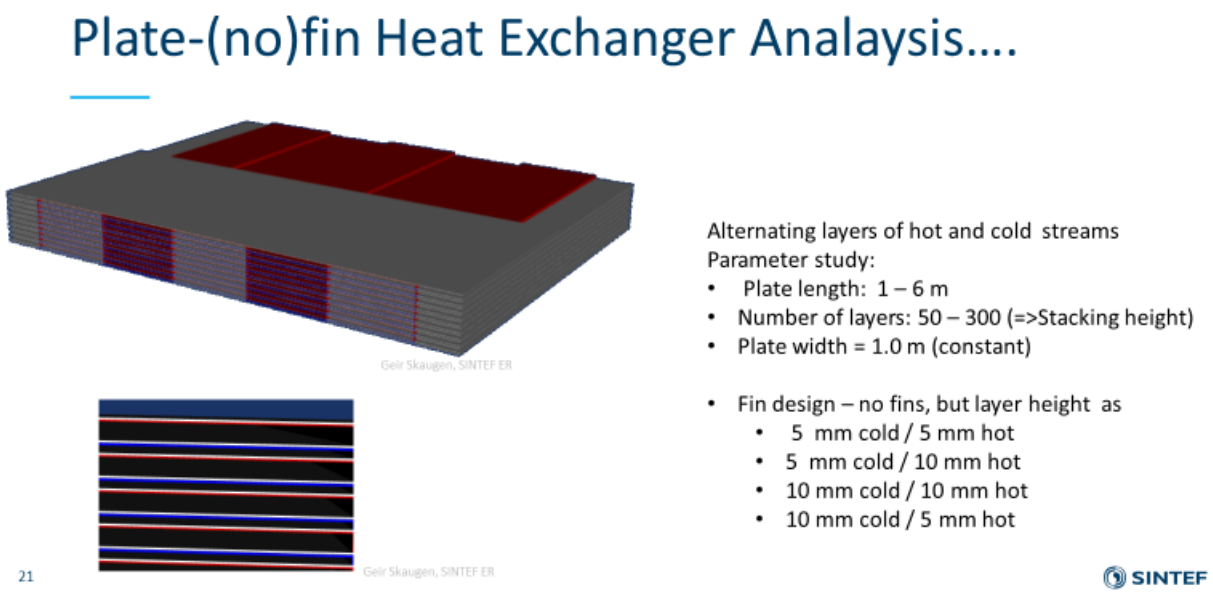


Figure 22 Plate - (no)Fin concept as HRHE

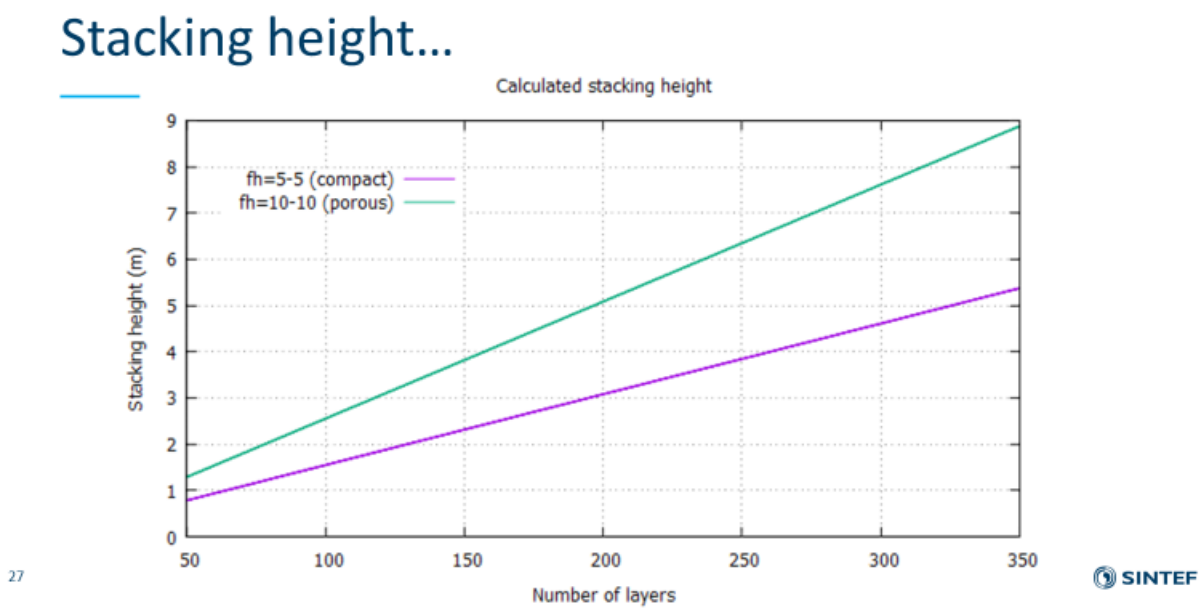


Figure 23 Stacking height as a function of number of layers

The heat transfer area range for the number of layers and plate length is shown below in Figure 24 and the corresponding core weights are shown in Figure 25.

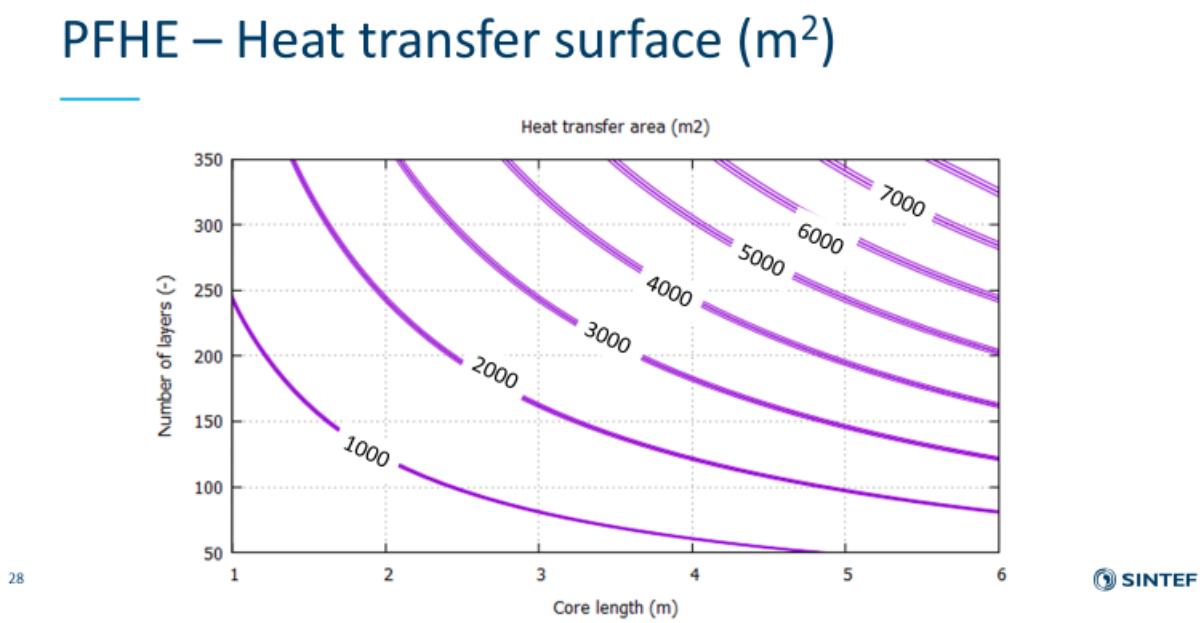


Figure 24 Heat transfer surface as a function of core length and stacking height - various plate distances

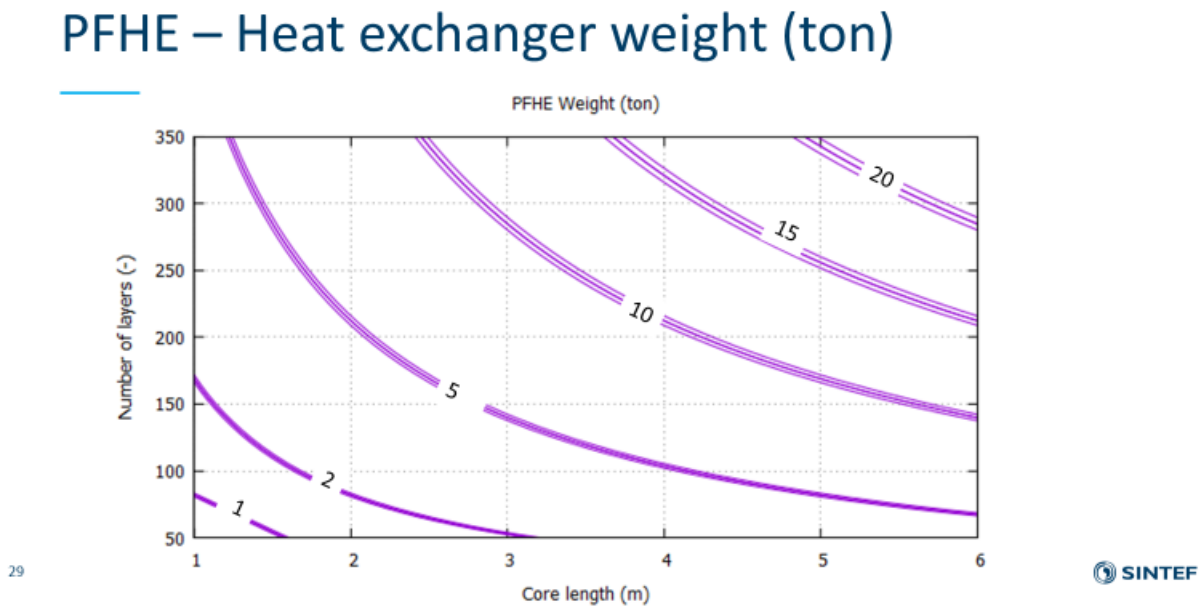
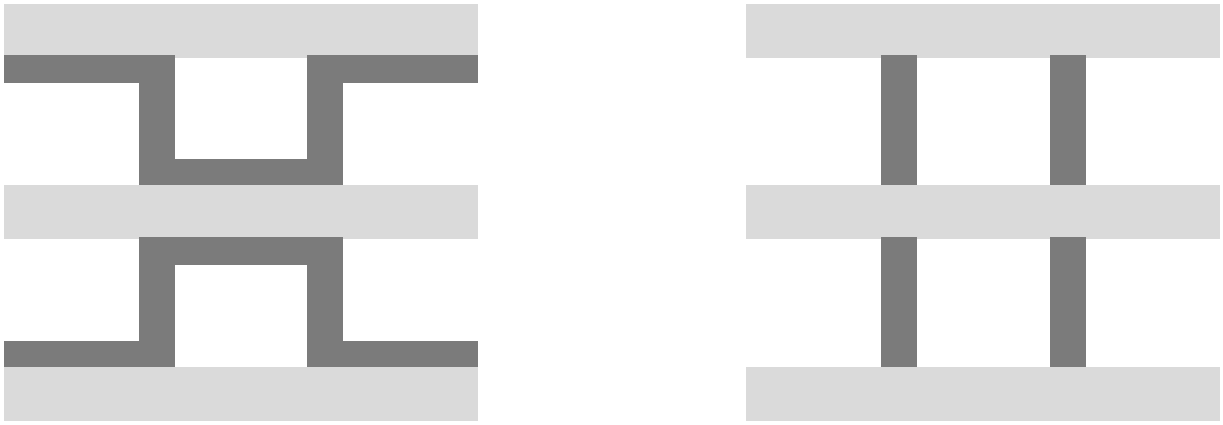


Figure 25 PFHE weight iso-lines

The weights in Figure 25 comes from a re-calculation after the first presentation of this concept. The material has been changed to steel. Moving from aluminium to steel would normally result in

a heavier core weight. However, the formulation of the fin attachment was changed to using thick single fins instead of the "normal" rectangular fin patterns where the fin thickness was also added to the partition plate thickness.

The change is illustrated in Figure 26 below, the original formulation as plain, rectangular profile is shown to the left while the new single fin formulation is shown to the right. The fin material is shown as dark grey so it is evident that with the previous formulation, even only 4 fins per/meter would include a full layer of unnecessary fin material as part of the heat exchanger weight. This is now corrected, an even using steel in both partition plate and in the support fins.



Rectangular fin profile

Single-fin profile

Figure 26 Fin profile formulation in evaluated concept

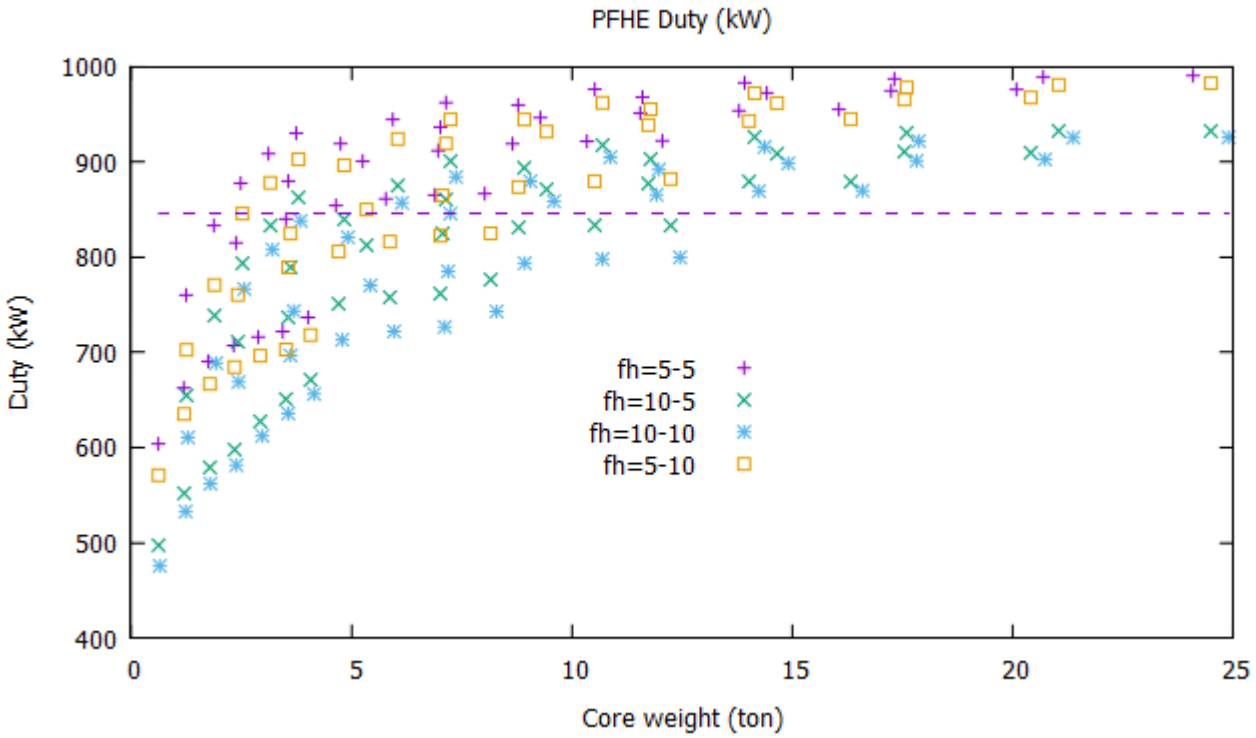


Figure 27 A map of duty vs weight for the plate(no)fin concept

A map of possible heat exchanger duty within this range of stacking heights and plate lengths are shown in a "duty vs weight" map in Figure 27. The different colours represent the variation in fin heights for the hot and cold layers. The dashed line represents the desired duty of 846 kW, and it can be seen that there are several combinations of plate length, number of layers and layer distances that can achieve that, already as light as 2.5 ton. The combination showing the lowest weight will be the ones with small layer distance, few layers and short plate lengths. These will also be subjected to the highest pressure loss. Pressure loss and pressure loss vs. duty is shown in Figure 28 and Figure 29. As seen from the latter of the two there are several design candidates possible.

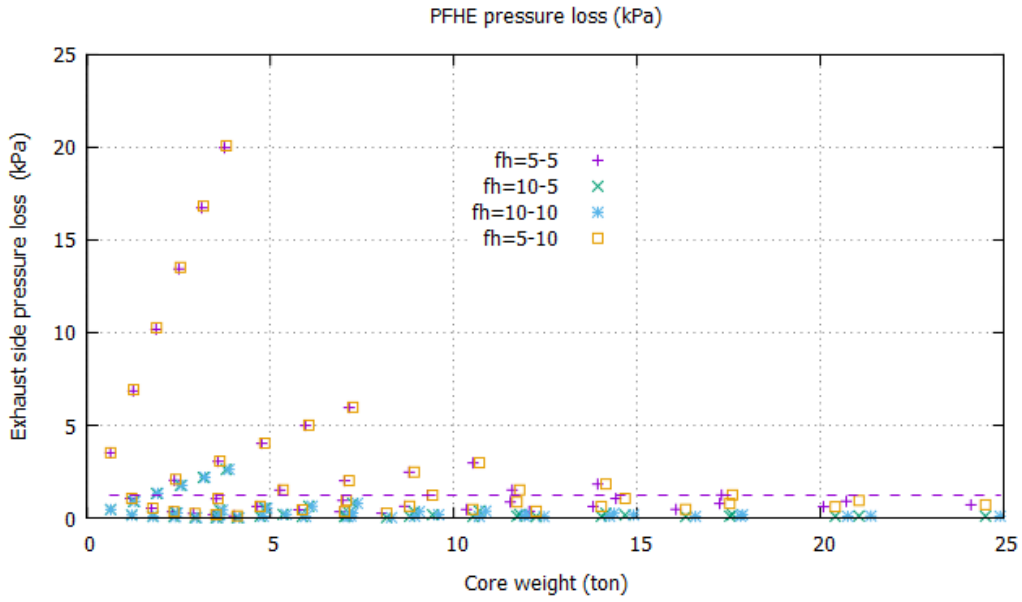


Figure 28 Exhaust pressure loss for plate concept

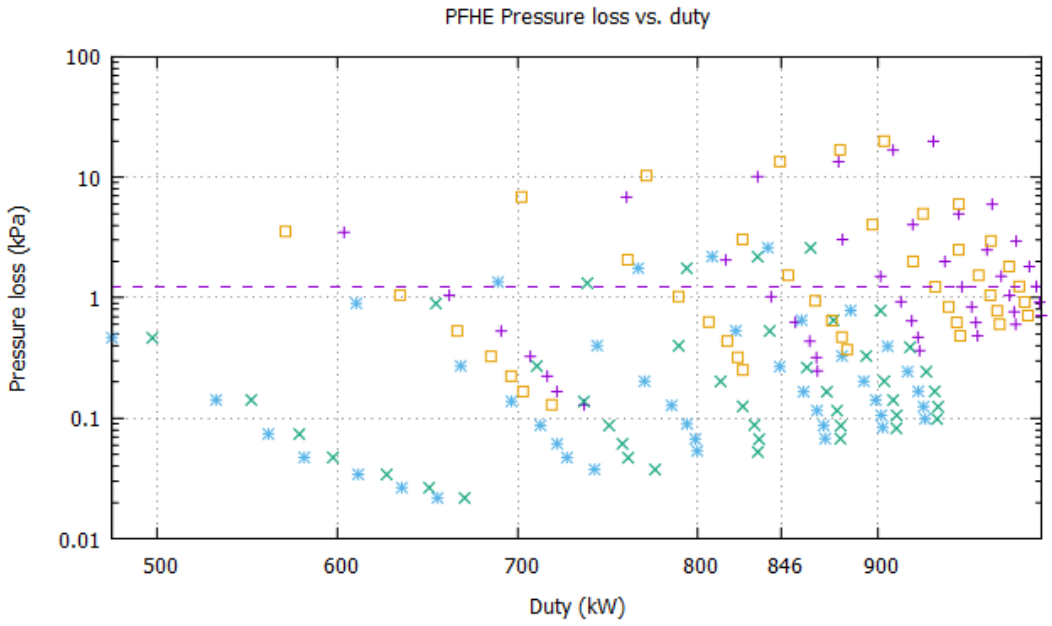


Figure 29 Pressure loss vs duty for plate concept

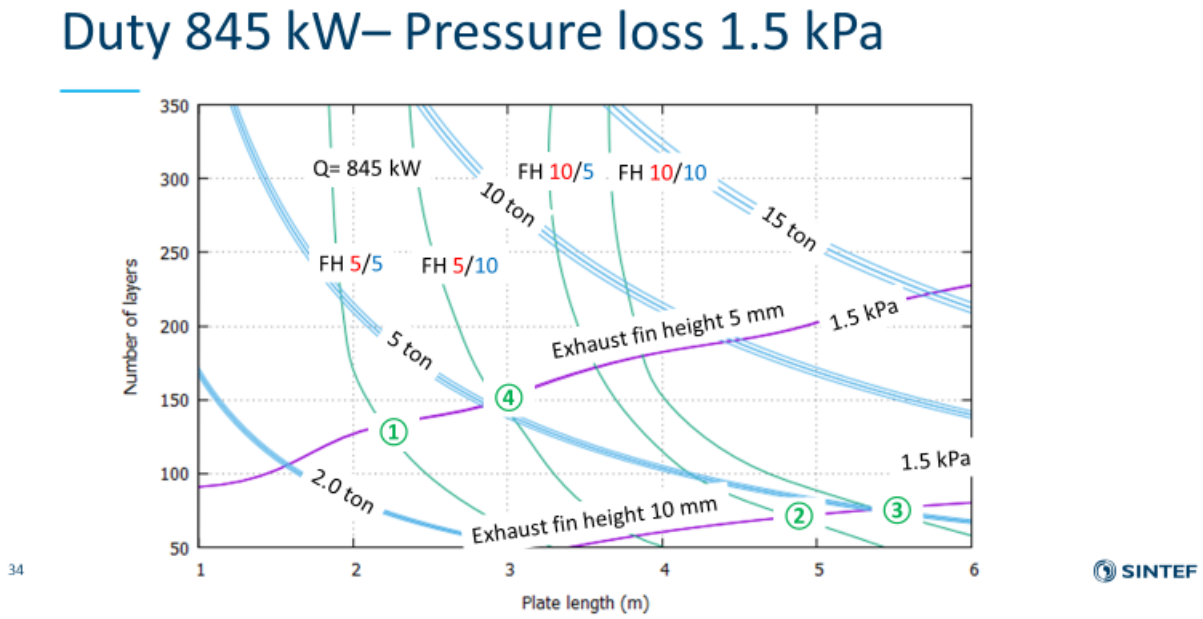


Figure 30 Duty, pressure loss and core weight for the plate concept

In Figure 30 all the important results are compiled into one graph. On the axis we see the plate length and the number of layers. The green curves show the heat exchanger duty of 845 kW for the four different combinations of the fin height. The two (actually 4) magenta curves show the exhaust side pressure low of 1.5 kPa for either 10 mm fin height (lower) and 5 mm din height (upper). The blue curves show the constant weight for 2, 5,10- and 15-ton core weight from Figure 25.

To fulfil the constraint of maximum pressure loss of for instance 1.5 kPa the solution must lie above this curve. The intersection between the performance curves (green) and the pressure loss curves (magenta) is labelled from 1 to 4 according to the weight. All the core weights in this analysis lie between 3 - ~5 ton. The design with 5 mm fin height (or layer spacing) for both hot and cold side have the lowest weight for a plate length of about 2.2 m requiring about 130 layers.

4 Summary and conclusions

This summary show how various robust heat exchanger models can be used to map out a range of possible design alternatives for a required duty and pressure loss constraint. The results in this memo as all done as simulations, and the more the free variables that is desired to investigate the more complex, time consuming and difficult to present in a clear manner. Here, more or less manually, some sort of optimum is found as shown in Figure 30. To do this more automatically, every heat exchanger model used in COPRO and HighEFF are prepared to be solved by optimisation techniques where a simulation is merely an optimisation with no degree of freedom.

As for the results itself. A tube -in -shell design is compared to a plate-(no)fin concept in terns of size and weight in terms of a specified duty and desired maximum exhaust side pressure drop. The analyse so far indicate that a plate concept could possibly be considerable more compact, with a core volume of 2 .2 (D) x 1 (W) x 2.0 (H) = 4.4 m^3 and with a core weight of about 3 ton, The plates are of 2.0 mm steel with 4 support fins per. meter if 1.0 mm thickness and 5 mm height.

The actual concept will be studied in more detail in HighEFF RA2.1 together with HighEFF RA4 in 2019.

The reference tubes-in-shell model on the other hand, would require a core volume of 12 (L) x 1.76 m (D) = about 29 m^3 and with a core weight of between 25 and 30 ton. (with 42.3 mm OD tubes, with 2.88 mm wall thickness).

So, if flat narrow channels can withstand erosion and fouling from the relevant flue gases, it could be a great potential for modular systems if space and weight are constrained.

5 Part 2 – Memo on the implementation and experience of derivative-free optimisation in the process and heat exchanger modelling framework

SINTEF Energi AS
SINTEF Energy Research
Address:
P.O. Box 4761 Torgarden
NO-7465 Trondheim
NORWAY
Telephone: +47 73597200
Telefax: +47 73597250
energy.research@sintef.no
www.sintef.no/energi
Enterprise/VAT No.:
NO 939 350 675 MVA

Memo

Derivative-free optimization in `FlexHX/CS`

PERSON RESPONSIBLE / AUTHOR
Brage Rugstad Knudsen

FOR YOUR ATTENTION
COMMENTS ARE INVITED
FOR YOUR INFORMATION
AS AGREED

DISTRIBUTION

PROJECT NO/ FILE CODE	DATE	CLASSIFICATION
HighEff, WP2.1/	2018-12-20	Unrestricted

Abstract

This memo presents results from implementation and testing of the numerical optimization solver `NOMAD` in the in-house software `FlexCS` for power cycles and heat exchanger design optimization. Details of the implementation are included.

SINTEF

Contents

1 Background **3**
1.1 Integrating NOMAD in FlexHX/CS 4

2 Numerical testing with NOMAD in FlexHX/CS **5**
2.1 Optimizing the design of heat-to-power cycles with NOMAD 5

3 Concluding remarks **6**

4 Future work **7**

References **7**

A Implementation of NOMAD in FlexCS **9**

B Additional files for integration of NOMAD in FlexHX/CS **14**

This memo builds upon HighEFF deliverable D2_1.2017.01 on numerical optimization routines in FlexHX/CS. The main conclusion from that memo was that a derivative-free optimization (DFO) solver with support for integer variables is likely a preferred supplement to the currently available optimization routines in FlexHX/CS, namely NLPQL . To this end, we have chosen to implement the NOMAD solver (Nonlinear Mesh-Adaptive Direct search) by Le Digabel (2011). Expanding the set of optimization routines in FlexHX and FlexCS with a DFO solver such as NOMAD provides the users with alternatives in selection of an adequate optimization approach for the black-box type optimization problems comprised by the FlexHX/CS models. This is important for the ability to optimizing a wide range of power cycle and heat exchanger design optimization problems. NOMAD can solve integer constrained optimization problems without access to derivatives. Additionally, the code includes metaheuristic search methods and functionality for surrogate-model generation which altogether may yield a more robust optimization approach than the available routines in FlexHX/CS when applied to complex heat exchanger and design problems. The positive outcome is an improved ability to optimize heat-exchanger and power-cycle design enabling innovative design, novel solutions, new applications and reduced effort required for successful numerical optimization of these models. This could benefit both HighEFF industry partners and beyond the center through solutions and engineering designs that improves industrial energy efficiency and utilization of surplus heat.

This memo presents implementation details of NOMAD in FlexCS , the solver's main features and comparative tests against NLPQL on a FlexCS model.

1 Background

FlexCS is an in-house SINTEF Energy software package for simulation and optimization of thermodynamic power cycles. FlexCS consists of many building blocks and components implemented in separate C or FORTRAN software modules and libraries. The different software modules provide access to fluid properties, models and correlations for heat transfer and pressure drop, and numerical routines and solvers. A central software module is the flexible modeling environment FlexHX for multi-stream heat exchangers (Skaugen et al., 2013). This framework can be used to model heat exchangers with a variety of surface geometries, flow areas and working fluids. FlexHX models heat exchangers by defining fluid nodes, solid wall elements and heat transfer resistance between them. Cycle models in FlexCS can choose heat-exchanger models of different detail levels and geometries, including shell-tube and plate-fine heat geometries.

Current optimization approach in the FlexHX/CS software FlexHX/CS applies a black-box type approach for solving design optimization problems. The software structure and integration ensure physical rigorous heat-exchanger and power-cycle models. Thermodynamic libraries and cycle components are efficiently integrated in the software, ensuring state-of-the-art and highly detailed heat-exchanger models to serve as a basis for the design optimization. The main optimization routine currently available in FlexHX/CS is the nonlinear programming (NLP) solver NLPQL (Schittkowski, 1986). NLPQL is a general-purpose sequential quadratic programming (SQP) type solver. It is a gradient-based solver, and features both augmented Lagrangian and ℓ_1 merit functions (Powell, 1978) with the BFGS Hessian approximation for implementation as a quasi-Newton method.

To supply gradient information to NLPQL, FlexHX/CS uses finite-difference calculations. This is the main hurdle for its numerical efficiency, and a prominent challenge in cases where outputs from the underlying FlexHX/CS are noisy. Moreover, handling of integer decision variables arising for instance in the geometrical design of heat exchangers is currently done through heuristic rounding procedures (Skaugen et al., 2015).

1.1 Integrating NOMAD in FlexHX/CS

NOMAD (Nonlinear Mesh Adaptive Direct Search) implements the MADS algorithm (Audet and Dennis, 2008) for constrained blackbox optimization. NOMAD[1] solves both single-objective and bi-objective constrained optimization problems without derivatives, and supports problems with binary variables, general integer variables and so-called categorical variables which allows problem structure (nature and number of variables) to change. The solver accepts optimization problems defined on the form

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & c_j(x) \leq 0,\ j = 1, \ldots, m, \\ & x^{\text{l}} \leq x_i \leq x^{\text{u}},\ i = 1, \ldots, n. \end{aligned} \tag{1}$$

where both the objective function $f : \mathbb{R}^n \mapsto \mathbb{R}$ and the constraints $c_j : \mathbb{R}^n \mapsto \mathbb{R}$ may be non-differentiable. x^{l} and x^{u} are lower and upper bounds on the optimization variables $x \in \mathbb{R}^n$, respectively, formulated separately from the constraints $c_j(x)$ but algorithmically handled in the same way. Equality constraints must be reformulated as inequality constraints. MADS algorithms as the one in NOMAD solves optimization problems (1) by generating iteration points on a series of meshes with varying size, where the mesh is a discretization of the variable space. In each iteration, the MADS algorithm generates a trial point on the mesh that seeks to improve the current best solution (incumbent). This principally defines the SEARCH step of the algorithm. When an iteration fails to improve the incumbent, the iteration is initiated on a finer mesh, adaptively searching and refining the mesh in the vicinity of the incumbent. This latter is the core of the algorithm's POLL step. See Audet and Dennis (2008) for further details.

NOMAD handles constraints through barrier functions, either by using an extreme barrier (EB) (Audet and Dennis, 2008) or a progressive barrier (Audet and Dennis Jr., 2009). The extreme barrier function simply rejects infeasible points, i.e. $f(x) = \infty$ outside the feasible set. The progressive barrier (PB) function approach allows iterates to be infeasible, and introduces a nonnegative barrier threshold $h^k_{\max}$ which is updated at the end of each iteration. Defining the constraint violation $h(x)$ as

$$h(x) := \begin{cases} \sum_{j=1}^{m} (\max\{c_j(x), 0\})^2 & \text{if } x \in X, \\ 0, & \text{otherwise}, \end{cases} \tag{2}$$

where $X = \{x \in \mathbb{R}^{\kappa} : x^{\text{l}} \leq x \leq x^{\text{u}}\}$, then every iteration point x^k such that $h(x^k) > h^k_{\max}$ is rejected. Upon this evaluation, PB performs a comparison of trial points and makes a ranking with respect to progression in feasbility and objective value, to some extent resembling filtering methods in nonlinear programming (Nocedal and Wright, 2006, Ch. 15.4). The threshold $h^k_{\max}$ is nonincreasing and updated at the end of each iteration. See (Audet and Hare, 2017, Ch 12.4) for further details.

NOMAD uses mesh refining to integer lattices for solving DFO problems with integer or binary variables. The solver incorporates the metaheuristic Variable Neighborhood Search (VNS) (Mladenović and Hansen, 1997). The latter improves the solvers ability to escape from regions with no feasible solution as well as local minima. Furthermore, NOMAD includes a library for surrogate-model generation and evaluation. Through this functionality, NOMAD uses the sequence of performed blackbox calls to generate surrogate models of certain types, e.g. quadratic or spline surrogates, which can then replace or complement the blackbox evaluations in the SEARCH and POLL steps. For FlexHX/CS models where simulations for a given configuration is highly time consuming, e.g. for detailed and complex heat-exchanger models, these features may prove to valuable. Finally, NOMAD has support for parallelization of blackbox evaluations.

[1] https://www.gerad.ca/nomad/Project/Home.html

2 Numerical testing with NOMAD in FlexHX/CS

2.1 Optimizing the design of heat-to-power cycles with NOMAD

Problem description This section compares the performance of NOMAD with NLPQL applied for optimizing the design of a simple organic Rankine cycle (ORC) implemented in FlexCS. We name the optimization problem *SimpleCycle*, which characteristics are shown in Table 1. The objective is to maximum net power produced by the ORC.

Extensive tuning of the NLP staring point and step-lengths in the FD approximations has been performed to optimize the performance of NLPQL for solving the *SimpleCycle* model. To evaluate the solvers' performance, we use as a reference this tuning of the derivative approximations for NLPQL and the solution obtained, while we use different initial starting points to compare the performance of the two solvers. In particular, with all variables normalized, we generate 10 different starting points by multiplying the reference stating point with a pseudo-random number in the range $[0, 1)$.

We use the following settings for the solvers:

- **NOMAD**:
  - Termination criteria: maximum iterations (equal to black-box/FlexCS calls) MAX_BB_EVAL = 1000 **OR** mesh size parameter $\Delta^m \leq 10^{-6}$ (MIN_MESH_SIZE).
  - Initial poll and mesh sizes (INITIAL_POLL_SIZE/INITIAL_MESH_SIZE are both set to 0.005.
  - Variable neighborhood search (VNS) is used by its default settings (VNS_SEARCH(true)), in which NOMAD will try to perform a maximum of 75% blackbox evaluations within the VNS search.
  - The extreme-barrier function (EB) is used for constraint handling.
- **NLPQL**:
  - Forward Euler FD approximations are applied as derivatives.
  - Maximum function calls in line search (MAXFUN) is set to 100000, while maximum number of iterations (MAXITER) is set to 2500000.

Model name	# Cont. opt var.	# Equality const.	# Inequality const.	Simple var. bounds.
SimpleCycle	5	0	5	10

Table 1: Properties of test problem *SimpleCycle*

Results from solving *SimpleCycle* with NLPQL and NOMAD, respectively, are shown in Table 2. Test #0 is the results from the reference starting point. NLPQL can be seen to require an order of magnitude less solution time compard with NOMAD for converging to a local solution to *SimpleCycle*. As expected, the superior local properties of NLPQL makes it outperform NOMAD whenever it is initialized sufficiently close to a stationary point and the FD approximations provide sufficiently accurate derivative information for approximate gradient and Hessian computations. However, NLPQL fails on converging to a solution on four of the starting points. In contrast, NOMAD finds a solution to *SimpleCycle* for all instances, indicating better robustness properties than NLPQL for solving this type of power cycle problems when information about a good starting is lacking, that is, with higher degree of infeasibility measured by some norm. NOMAD requires a high number of black-box (FlexCS) evaluations on the given model, and on most instances terminates either close to or on its set maximum number of iterations. At the optimal solution found by NLPQL, three out of five of the inequality constraints are active while none of the simple variable bounds.

	NLPQL				NOMAD			
Test #	Sol. time [s]	Obj. val.	BB calls	# SQP iter	Sol. time [s]	Obj. val.	BB calls	$\|\|\Delta_m\|\|_\infty$
0	1.4	4.40	32	5	15.9	4.39	1000	10^{-5}
1	1.2	4.40	25	4	14.0	4.39	994	10^{-6}
2	-	-	-	-	14.0	4.37	1000	10^{-5}
3	2.9	4.40	37	6	15.5	4.36	1000	10^{-4}
4	2.7	4.40	31	5	14.1	4.38	1000	10^{-4}
5	-	-	-	-	8.8	4.40	618	10^{-6}
6	1.8	4.40	74	10	13.4	4.39	943	10^{-6}
7	3.2	4.39	34	7	12.6	4.37	814	10^{-6}
8	-	-	-	-	15.3	4.37	816	10^{-6}
9	-	-	-	-	16.8	4.38	879	10^{-6}
10	1.67	4.40	38	6	14.6	4.39	773	10^{-6}

Table 2: Comparison of results from solving *SimpleCycle* with `NLPQL` and `NOMAD` . A - sign means that algorithm did not converge to a feasible solution. Note that objective is scaled.

`NOMAD` appears to not push solution close to constraint bounds, i.e. solutions are in the interior of the feasible set. Consequently, it can be seen in Table 2 that the solutions obtained by `NOMAD` often are close but slightly poorer than the solution found by `NLPQL` in the cases where the latter convergences to a stationary point (local solution).

The tuning of NOMAD may significantly impact its performance. To options (algorithmic settings) appear to be particularly important:

1. The type of barrier function for constraint handling (BB_OUTPUT_TYPE for each constraint).

2. Initial mesh and poll size (INITIAL_MESH_SIZE/INITIAL_POLL_SIZE).

From initial testing, EB seems to perform better than PB when starting from "difficult" initial points in the sense of being largely infeasible or far from a local solution. On the other hand, with a good initial starting point, `NOMAD` may perform better with PB for constraint handling, both in terms of reduced computation time and improved solution.

Remark 1. The step lengths ϵ_i for the FD approximations have been extensively tuned prior to the comparative test shown in Table 2. Comparing the effort and required preparation time for optimizing a power cycle in `FlexCS` with `NLPQL` and `NOMAD`, respectively, one should also consider the user's time spent on tuning the FD step lengths required for `NLPQL` as well as generation of a suitable NLP starting point. The latter is also required for `NOMAD`, that is, a suitable starting point, however, the Latin Hypercub (LH) and VNS metaheuristics implemented in the code can be exploited to generate an initial point.

Remark 2. `NOMAD` allows the user to fix variables. As such, `NOMAD` allows user of `FlexHX/CS` to easily switch or reduce the degrees-of-freedom during the course of designing and optimizing heat exchangers and power cycles.

3 Concluding remarks

If physical insight about the optimal design of an `FlexHX/CS` model can utilized to define a good starting point, and moderate effort is required to tune FD derivative approximations, then `NLPQL` should first be tested for optimization. When the design and specifications of either the heat

exchanger design, thermodynamical/fluid properties and/or the cycle components are complex and prohibits generation of good starting points, then NOMAD should be explored due to its ease of use, better robustness and integration of global-search heuristics.

4 Future work

It was observed during testing that NOMAD often quickly reduces the objective value in its first iteration after which the improvement significantly drops off. To speed up solution time, the user may therefore abort NOMAD when a satisfactory solution has been obtained. While the solver appears robust, it cannot compete with NLPQL's local support. As a result, it would be interesting to study the potential of combining NOMAD and NLPQL as an integrated DFO and FD gradient-type algorithmic approach to exploit the advantages of both. For this purpose, we propose to asses and compare the following standalone and hybrid optimization approaches on several instances of FlexHX/CS:

- NOMAD with VNS.
- NOMAD with VNS possibly with surrogate models.
- NLPQL standalone.
- NOMAD with VNS, switching to NLPQL for fast local convergence when improvement of NOMAD iterates is observed to abate.
- On FlexHX/CS models with integer variables: Use NOMAD to generate initial good solutions, fixing the integer variables and subsequently optimize continuous variables (degrees-of-freedom) using NLPQL. A challenge with this approach is that it is difficult to anticipate when NOMAD has found optimal values for the integer variables. A heuristic may be to switch to NLPQL if the values of integer/binary variables are observed to remain unchanged over many iterations of NOMAD.
- Finally, distinction between hard constraints for FlexHX/CS models compared to soft constraints that are desirable for the ultimate design may be exploited in specifying constraints for extreme (EB) and progressive barrier (PB), respectively.

The successful outcome of such a comparison could provide new insight and solutions of heat-exchanger and power-cycle design, and should lead to a journal publication.

References

Audet, C. and Dennis, J. E. (2008). Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on Optimization*, 18(4):1501–1503.

Audet, C. and Dennis Jr., J. (2009). A progressive barrier for derivative-free nonlinear programming. *SIAM Journal on Optimization*, 20(1):445–472.

Audet, C. and Hare, W. (2017). *Derivative-Free and Blackbox Optimization*. Springer.

Le Digabel, S. (2011). Algorithm 909: NOMAD: Nonlinear Optimization with the MADS Algorith. *ACM Transactions on Mathematical Software*, 37(4):1–15.

Le Digabel, S., Tribes, C., and Audet, C. (2017). *NOMAD User Guide*. https://www.gerad.ca/nomad/Downloads/user_guide.pdf.

Mladenović, N. and Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, 24(1):1097–1100.

Nocedal, J. and Wright, S. (2006). *Numerical Optimization*. Springer.

Powell, M. J. (1978). Algorithms for nonlinear constraints that use Lagrangian functions. *Mathematical Programming*, 14(1):224–248.

Schittkowski (1986). NLPQL: A FORTRAN subroutine for solving contrained non-linear programming problems. *Annals of Operations Research*, 5(11):485–500.

Skaugen, G., Hammer, M., Wahl, P. E., and Wilhelmsen, Ø. (2015). Constrained non-linear optimisation of a process for liquefaction of natural gas including a geometrical and thermo-hydraulic model of a compact heat exchanger. *Computers and Chemical Engineering*, 73:102–115.

Skaugen, G., Kolsaker, K., Walnum, H. T., and Wilhelmsen, Ø. (2013). A flexible and robust modelling framework for multi-stream heat exchangers. *Computers and Chemical Engineering*, 49:95–104.

A Implementation of NOMAD in FlexCS

NOMAD is implemented C++, and comes with a range of interfaces to different software. The structure of FlexCS requires integrating the NOMAD solver as a callable library (Le Digabel et al., 2017, p. 47). As FlexHX/CS is implemented in C and Fortran, a parser is required for linking with NOMAD. All FlexCS models is defined through three files: MODELNAME.c, MODELNAMEsim.c, MODELNAME.h. For using NOMAD, we will use the same set of files, subject to certain modifications, and define the file:

MODELNAME_optNOMAD.cpp: Defines an Evaluator class, specific for each model, and a function `system_optimizeNOMAD()` which replaces function `solvesystem()` in MODELNAME.c. The optimization call by `solvesystem()` is done in the file MODELNAMEsim.c.

Using NOMAD as a callable library requires the user to define the function `system_optimizeNOMAD()` for executing the underlying MADS solver, and an evaluator class that calls `CalcDeviation()` as a blackbox for evaluating the power cycle, thereby acting as an interface between the optimization solver and the model. In the evaluator class, the user defines how the black-box is called with the current iteration point x^k of NOMAD, and sets the output objective function and constraint residuals from the blackbox as iterate/evaluation-point attributes (through x.set_bb_output). `system_optimizeNOMAD()` defines output types from calling the black-box model (FlexCS), sets parameters for the solver and lower/upper bounds on the optimization variables. An example of MODELNAME_optNOMAD.cpp is shown in Listing 1. `system_optimizeNOMAD()` needs to be callable both from MODELNAMEsim.c and is thus declared in MODELNAME.h. Observe that instead of calling the optimization solver *inside* MODELNAME.c as is currently done with `solvesystem()`, we move the call to `system_optimizeNOMAD()` to the file MODELNAMEsim.c.

Below we describe changes that must be made to the existing FlexCS model files for calling NOMAD:

1. **Changes to file MODELNAMEsim.c**: MODELNAMEsim.c calls MODELNAME() to define cycle model and `system_optimizeNOMAD()` to optimize system using NOMAD. To this end, MODELNAMEsim.c need to access FCSSystem *system. An example of MODELNAMEsim.c in terms of the simple_cyclesim.c is shown in Listings 2. Observe that we call the "main" function `MODELNAME()` (e.g. simple_cycle (argv[1])) to *construct* but not optimize the power-cycle model.

2. **Changes to file MODELNAME.c**: Calling NOMAD to optimize a model in FlexCS requires commenting out `Solvesystem()` in function `MODELNAME(char *cfgFileN)` and the subsequent plotting and output functions. `CalcDeviation()` remains unchanged.

3. **Changes to file MODELNAME.h**: Additions to existing file MODELNAME.h is shown in Listings 3.

4. **The makefile**: A nomad.o target to must be created. An example makefile for the model simple_cycle is shown in appendix B, Listings 4.

Listing 1: Example of function `system_optimizeNOMAD()`

```
#include <stdio.h>
#include "nomad.hpp"

using namespace std;
using namespace NOMAD;
```

```
extern "C"{
        #include "simple_cycle.h"
}

/*-----------------------------------------*/
/*  The problem: the evaluation is made    */
/*  by calling CalcDeviation                        */
/*-----------------------------------------*/
class My_Evaluator : public Evaluator {
public:
  My_Evaluator  ( const Parameters & p ) :
    Evaluator ( p ) {}

  ~My_Evaluator ( void ) {}

  bool eval_x ( Eval_Point   &x          ,
                const Double &h_max      ,
                bool         &count_eval   ) const {

    int n                                = x.size();
    double *xx                   = new double[n];
        FCSSystem *system        = getFCSSystem();


        // Update scaled versions of variable vector x (as system->xscale):
    int i;
    for ( i = 0 ; i < n ; ++i ){
        xx[i] = x[i].value();
        system->xscale[i] = xx[i]; // Set new variable values to (FCSSystem) system.
        }

        // Execute black-box call:
    double g[system->m];
    CalcDeviation(&(*system).f, g);

    x.set_bb_output  ( 0 , system->f); // set obtained objective value:

    bool DisplayConstraintResidual = false;

    for (i=0 ; i<system->m ; ++i){
                x.set_bb_output  ( i+1 , -g[i]); // set constraint residuals
                if(DisplayConstraintResidual){
                        if(i==(system->m - 1)){
                                printf(" g_%i: %.6f \n", i,-g[i]);
                        }
                        else{
                                printf(" g_%i: %.6f ", i,-g[i]);
                        }
                }
        }

    count_eval = true; // count a black-box evaluation

    return true;        // the evaluation succeeded
  }
};



void system_optimizeNOMAD (
```

```
                        int  *n       ,            // # of variables
                        int     *m       ,         // # of outputs (obj + m-1 constraints)
                        double  *x       ,         // starting point (IN) / solution (OUT)
                        double  *lb      ,         // lower bounds for each variable
                        double  *ub      ,         // upper bounds
                        int     *max_bbe ,         // max # of evaluations (-1: not considered)
                        int     *display_degree, // display_degree (0-4; 0: no display)
                        char    outputlist[],   // list of strings (without quotes) for defining
                             output stats of NOMAD
                        char    WriteToFile[]   // output filename
                        ) {

Display out ( std::cout );
out.precision ( DISPLAY_PRECISION_STD );

try {

  int i;

  // parameters creation:
  Parameters p ( out );

  p.set_DIMENSION (*n);             // number of variables

  vector<bb_output_type> bbot (*m); // definition of output types:
  bbot[0] = OBJ;                    // first output : objective value to minimize
  for ( i = 1 ; i <*m ; ++i )      // other outputs: constraints cj <= 0
    bbot[i] = EB;
  p.set_BB_OUTPUT_TYPE ( bbot );

  // NOMAD starting point and simple variable bounds
  Point px0 ( *n );
  Point plb ( *n );
  Point pub ( *n );
  for ( i = 0 ; i < *n ; ++i ) {
    px0[i] = x [i];
    if ( lb[i] > -1e20 )
              plb[i] = lb[i];
    if ( ub[i] < 1e20 )
              pub[i] = ub[i];
  }
  p.set_X0          ( px0 );
  p.set_LOWER_BOUND ( plb );
  p.set_UPPER_BOUND ( pub );

      // SET OPTIONS FOR NOMAD

  // maximum number of black-box evaluations:
  if ( *max_bbe > 0 )
    p.set_MAX_BB_EVAL ( *max_bbe );

  // display degree:
  p.set_DISPLAY_DEGREE ( *display_degree );
  p.set_INITIAL_POLL_SIZE(0.005);
  p.set_INITIAL_MESH_SIZE(0.005);
  p.set_VNS_SEARCH(true);
  p.set_MIN_MESH_SIZE(1e-6);
  p.set_DISPLAY_STATS(outputlist);
  p.set_STATS_FILE(WriteToFile,outputlist);
  p.set_MIN_POLL_SIZE(1e-6);
  // p.set_DISPLAY_ALL_EVAL(true);
  // parameters validation:
```

```
    p.check();

    // custom evaluator creation:
    My_Evaluator ev ( p );

    // algorithm creation and execution:
    Mads mads ( p , &ev );
    mads.run();

    // get the solution:
    const Eval_Point * bf = mads.get_best_feasible();
    if ( bf )
      for ( i = 0 ; i < *n ; ++i ){
                x[i] = (*bf)[i].value();
          }
    // Set best feasible point to system:
    printf("\nPerforming final CalcDeviation with best found feasible solution\n");
    FCSSystem *system = getFCSSystem();

    for( i = 0 ; i < system->nvar ; ++i ){
      system->xscale[i] = x[i];
    }
    // Call simulator for simulation with final optimization result:
    double g[system->m];
    CalcDeviation(&system->f, g);
  }
  catch ( exception & e ) {
    cerr << "\nNOMAD has been interrupted (" << e.what() << ")\n\n";
  }
}
```

Listing 2: Example of file MODELNAMEsim.c used to call `NOMAD` for optimizing design of power-cycle models in `FlexCS` .

```
#include <stdio.h>
#include <stdlib.h>
#include "simple_cycle.h"
#include "timing.h"
#include "fhxcmdopt.h"


// Additional includefiles:
#include "fcssystem.h" // To access FCSSystem *system
// #include "simple_cycle_solvesystemNOMAD.h"

int main (int argc, char **argv) {

  int i;
  double CpuStart = CpuTime();
  options_init();
  cmdlineFile2(OptionPtr, COptionPtr, IOPT,IOPTC, argc, argv);

  // Use simple_cycle() to create but NOT solve system:
  int iret = simple_cycle (argv[1]);

  FCSSystem *system = getFCSSystem(); // Get the actual FCS system.

  int NOMADMaxIter      = 1000;
  int NOMADDisplayLevel = 2;
  int numconstraints    = system->m + 1; // m +1 (objective function)
```

```
  char listOfOutputStats[] = "BBE OBJ"; // Define outputs to show during execution of NOMAD and
      in output file
  char WriteToFile[]  = "SimpleCycleResNomad"; // Name of output result file.

  FCSSystem_scaleInVar(system); // Scale optimization variables

  // Call NOMAD:
  system_optimizeNOMAD(&(system->nvar), &numconstraints, system->xscale, system->xlowscale,
      system->xupscale, &NOMADMaxIter, &NOMADDisplayLevel, listOfOutputStats, WriteToFile);

  FCSSystem_scaleOutVar(system); // Update unscaled final variables.


  printf("\n\n Solution at termination of NOMAD: \n");
  for ( i = 0 ; i < system->nvar ; ++i ){
    if (i== system->nvar - 1){ printf("x_%i = %.3f. \n ",i ,system->xscale[i]);}
    else{printf("x_%i = %.3f. \t ",i ,system->xscale[i]);}
  }

  printf("Unscaled: \n ");
  for ( i = 0 ; i < system->nvar ; ++i ){
    if (i== system->nvar - 1){ printf("x_%i = %.3f. \n ",i ,system->x[i]);}
    else{printf("x_%i = %.3f. \t ",i ,system->x[i]);}
  }

  printf(" \n Obj. value at termination: f(x) = %.4f \n \n", *(&system->f));
  printf("Number of equality constraints: %i \n", system->me);
  printf("Number of inequalities: %i\n", (system->m-system->me));
  printf("Simple bounds on variables: \n");

  printf ("CPU time used is %g seconds\n", CpuTime() - CpuStart);

  return iret;
}
```

Listing 3: Additional definitions/declarations in MODELNAME.h

```
// Additional external declariations:
extern void CalcDeviation(double *f, double g[]);
extern void FreeAAAMemory(void);
extern FCSSystem* getFCSSystem(void);

#ifdef __cplusplus
extern "C"
#endif
void system_optimizeNOMAD (
                int     *n  ,
                int     *m              ,
                double  *x              ,
                double  *lb             ,
                double  *ub             ,
                int     *max_bbe        ,
                int     *display_degree,
                char    outputlist[],
                char    WriteToFile[]           );
```

B Additional files for integration of NOMAD in FlexHX/CS

Listing 4: Makefile for the model simple_cycle

```
#Common part for all models
#
ifeq ($(shell uname -o), Msys)
  CODES = /c/progs
  EXE = .exe
else
  CODES = $(HOME)/codes
  EXE =
endif

include $(CODES)/flexcs/models/Makefile.models.common

#LDFLAGS += -Wl,-V,-t

CO2DIR = $(CO2ROOT)/iupac
CO2VERSION = $(IUPACVERSION)
include $(CO2DIR)/version.mk

#########################################################
#
#                       NOMAD defintions:
#
LIB_DIRNOMAD                = $(NOMAD_HOME)/lib
LIB_NOMAD                       = libnomad.dll


LDLIBSNOMAD                 = -lm -lnomad -lstdc++

COMPILATOR_OPTIONSNOMAD     = -ansi -Wall -O3
INCLUDENOMAD                = -I$(NOMAD_HOME)/src -I$(NOMAD_HOME)/ext/sgtelib/src -I.
COMPILENOMAD                = $(CXX) $(COMPILATOR_OPTIONSNOMAD) $(INCLUDENOMAD)

OBJS =  \
        $(OUTDIR)/simple_cyclesim.o \
        $(OUTDIR)/simple_cycle.o
        # $(OUTDIR)/nomad.o
OBJSNOMAD = $(OUTDIR)/nomad.o
#########################################################


# $(info VAR="$(OUTDIR)")

FLEXHX_OPTIM = libflexhx_optim.a
FLEXHX_DEBUG = libflexhx_debug.a
FLEXHX_PROFILE = libflexhx_profile.a

FLEXCS_OPTIM = libflexcs_optim.a
FLEXCS_DEBUG = libflexcs_debug.a
FLEXCS_PROFILE = libflexcs_profile.a

# Additional targets
FHXCS_OPTIM = flexhx_optim flexcs_optim
FHXCS_DEBUG = flexhx_debug flexcs_debug
FHXCS_PROFILE = flexhx_profile flexcs_profile

# Define nomad.o target:
nomad.o:  $(SRC)/simple_cycle_solvesystemNOMAD.cpp
```

```
	$(COMPILENOMAD) $(CFLAGS) -c $(SRC)/simple_cycle_solvesystemNOMAD.cpp -o $(OUTDIR)/nomad
	    .o

$(info VAR="$(OUTDIR)")

cdebug: clean_objects debug
coptim: clean_objects optim
cprofile: clean_objects profile

debug: simple_cycle_debug
optim: simple_cycle_optim
profile: simple_cycle_profile

simple_cycle_optim: CFLAGS = $(CFLAGS_OPT)
simple_cycle_optim: FFLAGS = $(FFLAGS_OPT)
simple_cycle_optim: CLIBS = $(OPTIMLIBS)
simple_cycle_optim: objectdir  $(TPBLOCK) $(OBJS)
simple_cycle_optim: nomad.o
	$(CXX) -o simple_cycle_optim $(LDFLAGS) $(OBJS) $(OBJSNOMAD) $(CLIBS) $(LDLIBSNOMAD) -L$
	    (LIB_DIRNOMAD)
	strip $@$(EXE)



simple_cycle_profile: CFLAGS = $(CFLAGS_DEBUG) -fno-omit-frame-pointer -pg
simple_cycle_profile: FFLAGS = $(FFLAGS_DEBUG) -fno-omit-frame-pointer -pg
simple_cycle_profile: CLIBS = $(PROFILELIBS)
simple_cycle_profile:  objectdir  $(TPBLOCK) $(OBJS)
	$(CXX) -o simple_cycle_profile $(LDFLAGS) -pg $(OBJS) $(CLIBS)

simple_cycle_debug: CFLAGS = $(CFLAGS_DEBUG)
simple_cycle_debug: FFLAGS = $(FFLAGS_DEBUG)
simple_cycle_debug: CLIBS = $(DEBUGLIBS)
simple_cycle_debug: objectdir  $(TPBLOCK) $(OBJS)
simple_cycle_debug: nomad.o
	$(CXX) -o simple_cycle_debug $(LDFLAGS) $(OBJS) $(OBJSNOMAD) $(CLIBS) $(LDLIBSNOMAD) -L$
	    (LIB_DIRNOMAD)


simple_cyclegibb_optim: CO2DIR = $(CO2ROOT)/gibb
simple_cyclegibb_optim: CFLAGS = $(CFLAGS_OPT)
simple_cyclegibb_optim: FFLAGS = $(FFLAGS_OPT)
simple_cyclegibb_optim: CLIBS = $(OPTIMLIBS)
simple_cyclegibb_optim: CO2VERSION = $(GIBBVERSION)
simple_cyclegibb_optim: objectdir  $(TPBLOCK) $(OBJS)
simple_cyclegibb_optim: nomad.o
	$(CXX) -o simple_cyclegibb_optim $(LDFLAGS) $(OBJS) $(CLIBS)
	strip $@$(EXE)


simple_cyclegibb_profile: CO2DIR = $(CO2ROOT)/gibb
simple_cyclegibb_profile: CFLAGS = $(CFLAGS_PROFILE)
simple_cyclegibb_profile: FFLAGS = $(FFLAGS_PROFILE)
simple_cyclegibb_profile: CLIBS = $(OPTIMLIBS)
simple_cyclegibb_profile: CO2VERSION = $(GIBBVERSION)
simple_cyclegibb_profile: objectdir  $(TPBLOCK) $(OBJS)
	$(CXX) -o simple_cyclegibb_profile $(LDFLAGS) -pg $(OBJS) $(CLIBS)

simple_cyclegibb_debug: CFLAGS = $(CFLAGS_DEBUG)
simple_cyclegibb_debug: FFLAGS = $(FFLAGS_DEBUG)
simple_cyclegibb_debug: CLIBS = $(DEBUGLIBS)
simple_cyclegibb_debug: CO2DIR = $(CO2ROOT)/gibb
```

```
simple_cyclegibb_debug: CO2VERSION = $(GIBBVERSION)
simple_cyclegibb_debug: objectdir  $(TPBLOCK) $(OBJS)
	$(CXX) -o simple_cyclegibb_debug $(LDFLAGS) $(OBJS) $(CLIBS)

#	$(LIB_DIR)


# Testing new approach for defining components and mixtures
# Involves new commandline-parser that allows text values in addition to float.


flexhx_optim:
	cd $(FLEXHXDIR); make $(FLEXHX_OPTIM); cd -

flexhx_debug:
	cd $(FLEXHXDIR); make $(FLEXHX_DEBUG); cd -

flexhx_profile:
	cd $(FLEXHXDIR); make $(FLEXHX_PROFILE); cd -

flexcs_optim:
	cd $(FLEXCSDIR); make $(FLEXCS_OPTIM); cd -

flexcs_debug:
	cd $(FLEXCSDIR); make $(FLEXCS_DEBUG); cd -

flexcs_profile:
	cd $(FLEXCSDIR); make $(FLEXCS_PROFILE); cd -

clean_objects:
	rm -f $(OUTDIR)/*.o


objectdir:
	@mkdir -p $(OUTDIR)

# ===== GENERAL TARGETS =====


$(OUTDIR)/%.o: $(SRC)/%.c
	$(CC) -c $(CFLAGS) -o $@ $<


symlinks:
  include $(CODES)/flexcs/models/Makefile.models.symlinks
```