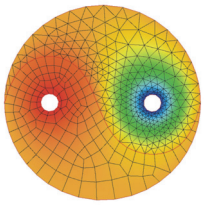
Publishing House
AKAPIT

COMPUTER METHODS IN MATERIALS SCIENCE

Informatyka w Technologii Materiałów

Vol. 14, 2014, No. 1



APPROACH FOR AN AUTOMATIC OPTIMISATION OF PRODUCTION CHAIN AS A TOOL FOR INTELLIGENT MANUFACTURING IN METAL FORMING

ADAM LEGWAND, KONRAD PERZYŃSKI*, ŁUKASZ MADEJ, MACIEJ PIETRZYK

AGH University of Science and Technology, Krakow, Poland
**Corresponding author: kperzyns@agh.edu.pl*

Abstract

The objective of the paper is development of an automatic system for optimisation of production chains in metal-forming. Product properties and their uniform distribution are usually the key parameters for formulation of the objective function in the optimization problem. Such parameters as strain/stress or final grain size distribution significantly influence material behaviour under exploitation conditions, therefore, theoretical prediction of those variables is inevitable for the optimization of the manufacturing chain. Thus, the main focus in the paper is on description of the developed complex tool capable of realization of automatic optimisation for subsequent manufacturing stages. Description of particular components responsible for automatic geometrical model generation, incorporation into the finite element (FE) software, optimisation operations and finally data transfer between subsequent modules are described in detail. A rod rolling operations of elliptical profiles were selected as case studies to demonstrate system capabilities.

Key words: optimization, modeling, finite element method

1. INTRODUCTION

Dynamically evolving global markets are forcing manufacturers into a state of continuous adaptation of production strategies/technologies to meet new customer expectations and maintain competitiveness. In metal forming industry the major challenge is obtaining required exploitation properties of products. Parameters i.e. strength, ductility, fatigue resistance, wear resistance or thermal resistance are crucial for increasing the safety by extending the life cycle of products (Senkov et al., 2004; Timokhina et al., 2007; Robertson et al., 2008; Beladi et al., 2009; Muszka et al., 2012). On the other hand, the life cycle of products becomes shorter due to rise and fall of demand and increasing demand for customized dedicated products. In order to meet these market pressures, modern manufacturing systems have to be intelligent and flexible, what can be achieved mainly by the feedback of information from the subsequent manufacturing stage.

The impact of the product life cycle can be twofold. First, it can influence planning and organization of the production system, which has to become flexible, reconfigurable and cost efficient (Hon & Xu, 2007; Feng et al., 2013; Pereira & Paulre, 2001; Bruccoleri et al., 2005). Second aspect, which is the topic of the present paper, is accounting for the product properties at the subsequent stages of the manufacturing technology design. Influence of subsequent manufacturing operations on material behaviour cannot be underestimated and have to be taken into account during new technology development (Rauch et al., 2008; Pietrzyk et al., 2010). Development of such intelligent system, which uses optimization techniques with the objective function composed of internal state of the material is the subject of the present paper.

ISSN 1641-8581

2. MODELLING OF THE PRODUCTION CHAIN

During last thirty years FE method was applied to simulate various forming operations i.e. cold forging, hot forging, rolling, extrusion, stamping etc. Usually FE models were applied to obtain information regarding shape, stress, strain or temperature distribution in one particular manufacturing stage i.e. rolling or forging (Pietrzyk et al., 1993; Pietrzyk et al., 1993; Pietrzyk, 2001; Pietrzyk & Kuziak, 2004). These information, as shown in figure 1a, where then an input parameters for the optimisation procedures, applied to obtain final numerical results, which are in close agreement with real material behaviour under industrial conditions. During further years, rheological models commonly used in the FE simulations were improved to take in to account microstructure evolution under loading conditions. Information regarding changes in grain shape and size during the static and dynamic recrystallization became available from the FE simulation e.g.:

$$\frac{dD}{dt} = -D\frac{dX}{dt}\ln N \qquad (1)$$

where: X – recrystallized volume fraction, N – number of new grains per one old grain.

As a results number of input parameters for the optimisation procedure increased, what resulted in higher accuracy and reliability of the numerical results (figure 1b).

However, in present days, a modern approach to numerical simulation is evolving rapidly. The manufacturing chain modelling (4D modelling) is used to simulate not only one particular manufacturing process i.e. forging or rolling but to simulate the entire production chain and relationships between subsequent operations. In this approach despite three dimensional space the manufacturing chain is considered as the fourth dimension. An example of modelling of such manufacturing chain as well as investigation of final part behaviour under exploitation conditions is presented in figure 1c. Due to this methodology physical and numerical simulations of phenomena and processes occurring in metallic materials during manufacturing stages can be performed. Optimisation procedures can be again applied to each particular manufacturing operation but also they can be applied to the entire chain. That allows to manufacture products with better in use properties, such as strength, crack resistance, fatigue resistance, shear resistance, wear resistance, creep resistance, corrosion resistance, hardness, ductility, porosity, adhesive properties, high temperature

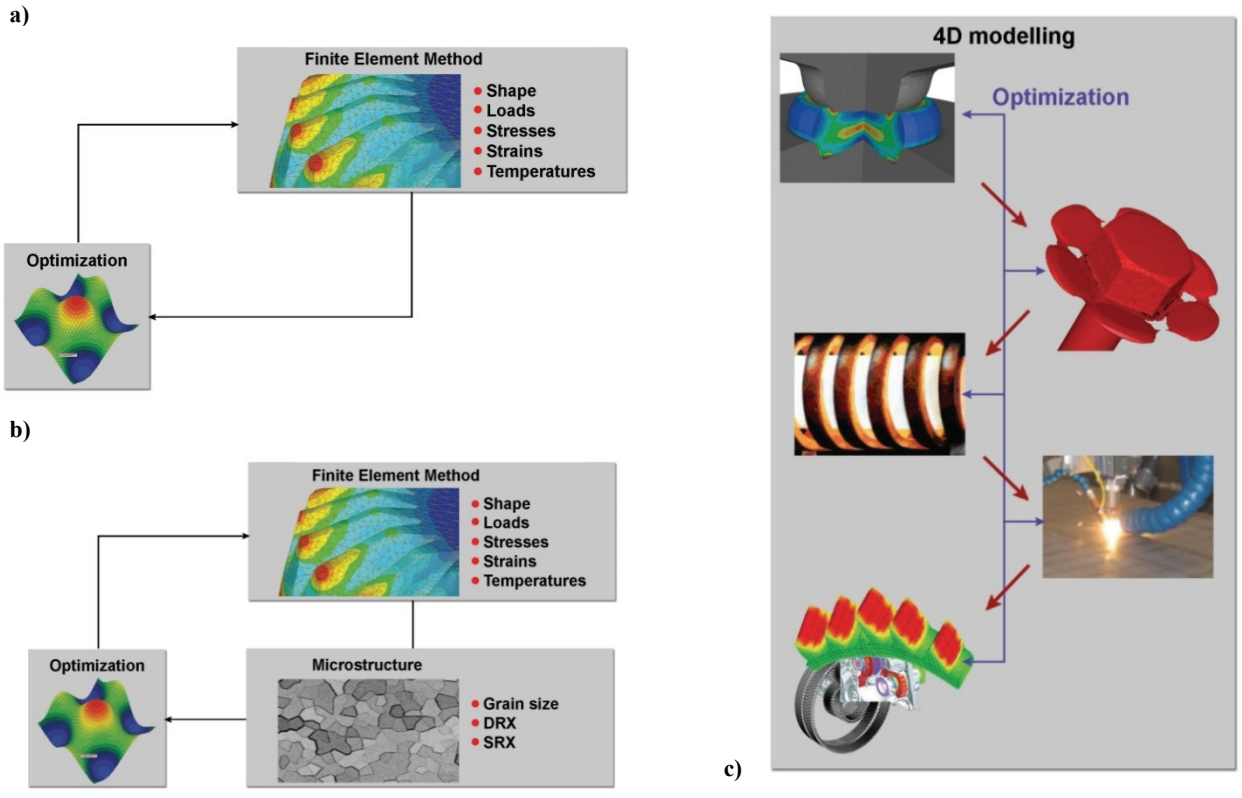


Fig. 1. *Development of the methodology of the computer aided technology design.*

resistance, toughness, conductivity, low temperature resistance, etc. According to this analysis proper product behaviour during exploitation can be designed before final product is manufactured and ready to use. The idea of the Life Cycle modelling is in detail described in other authors work (Madej et al., 2007), as well as in (Bariani et al., 2007).

However, commercial finite element software still lacks of capability supporting modelling of production chains. A lot of operations have to be realized manually what is time consuming and extends new technology development time. That is why, development of automatic system for optimisation of production chains is the main subject of the present research. Description of particular proposed components responsible for automatic geometrical model generation, incorporation into the finite element (FE) software and finally optimisation operations are described in detail in the following chapters.

software. Such an approach provides flexibility in selection of different optimization algorithms and different finite element software, but it requires development of an efficient communication protocols between subsequent modules. Thus, the second objective of this work is to design data transfer interface for communication between finite element software and optimization modules. Similar methodology was presented in previous authors work (Legwand & Perzyński, 2012).

To create such flexible tool, the developed system is divided into free modules, which can work separately. The first is responsible for the optimization process and is written using JAVA programing language. The second module is a finite element software responsible for modeling of subsequent metalforming operations. At this stage of the research authors decided to use commercial Abaqus software. Finally, the third module, based on XML

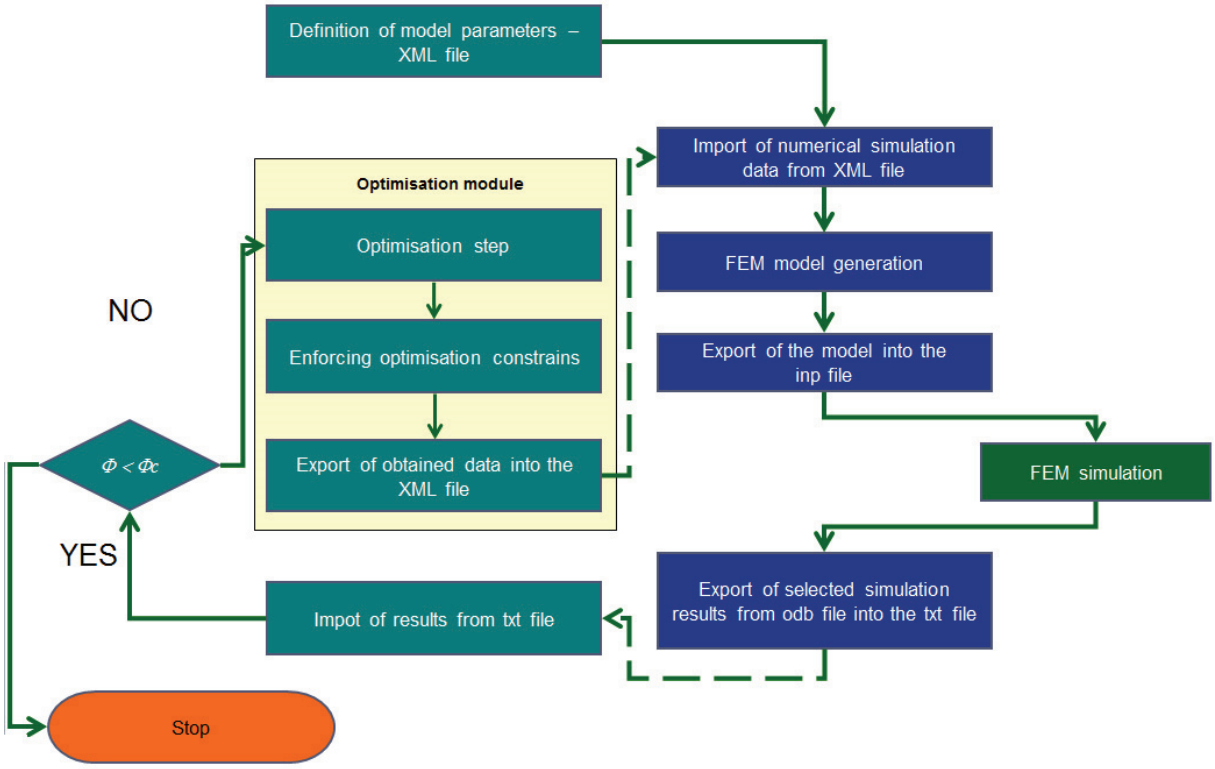


Fig. 2. *Sequence diagram of the optimization system.*

3. SYSTEM FOR AUTOMATIC OPTIMISATION OF PRODUCTION CHAINS

As mentioned the main objective of the work is to create a user friendly automatic optimization system based on the standalone optimization module, which interacts with commercial finite element

language, is responsible for input/output data transfer between above mentioned parts. Such architecture facilitates further development of the application as subsequent modules can be extended and modified independently. Schematic diagram presenting proposed optimization system of production chains is shown in figure 2 and detailed description of particular components is in the following chapters.

3.1. Optimization module

As seen in figure 2, optimization module is created using widely used JAVA software platform, which represent independent development environment with considerable set of libraries. JAVA code is also easily transferable between various operating systems what increases flexibility of the developed system. Implemented optimization module is responsible for:

- Importing finite element simulation output data.
- Evaluation of the defined objective function.
- Execution of the optimization process based on implemented algorithms.
- Handling optimization limitations with the penalty function.
- Generation of the finite element simulation setup data file.
- Launching python script and ultimately finite element software.

Presently, there are no available open source libraries, which could handle above tasks thus most of the functionality had to be implemented as an in house solutions. Schematic illustration representing sequence diagram of the developed optimization module is presented in figure 3.

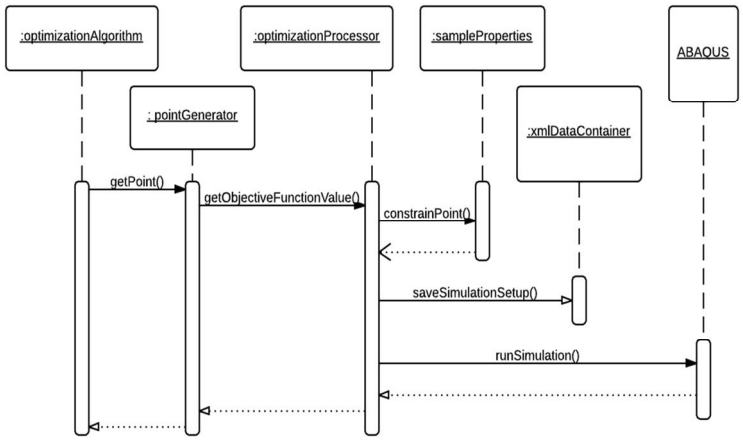


Fig. 3. *Sequence diagram of the optimization module.*

The key part in this module is optimization algorithm that is used during the further research. At this stage of work the Nelder-Mead method (Nedler & Meadf, 1965) also called downhill simplex method was implemented to manage the optimization task. This method is commonly used to deal with mutlidimensional problems, which are common in metalforming. It is based on a concept of a simplex, which is a polytope of $N + 1$ vertices in N dimensions (figure 4), which is being formed in subsequent iterations by a set of geometrical transformations. As a result simplex is searching in the neighborhood of a local minimum of a goal function.

In the Simplex method, three operations, which are dedicated to replace simplex vertex with "worst" objective function value in current iteration by new point are used. First operation, which is performed at the beginning of every iteration is to set points P_l and P_h:

$$P_l = \min_i f(P_i) \tag{2}$$

$$P_h = \max_i f(P_i) \tag{3}$$

First operation is *reflection* of P_h, which is defined as:

$$P^* = (1 + \alpha)\ \overline{P}\ - \alpha\, P_h \tag{4}$$

where: P^* – reflection point of P_h, α – reflection coefficient, $\overline{P}$ – centroid of the points with excluding P_h.

If reflection operation has produced new minimum of the current simplex, another operation is performed, called *expansion*:

$$P^{**} = \gamma P^* + (1 - \gamma)\,\overline{P} \tag{5}$$

where: P^{**} – expansion point, γ – expansion coefficient that should be greater than 1.

Expansion is successful if $f(P^{**}) < f(P_l)$ and in that case P_h is replaced by P^{**}, than optimization process moves to the next iteration. Otherwise P_h is replaced by P^*.

Next operation is performed if after reflection $f(P^*) > f(P_i)$ for all $i \neq h$ and it is called *contraction*:

$$P^{**} = \beta P_h + (1 - \beta)\,\overline{P} \tag{6}$$

where: β – contraction coefficient, which lies between 0 and 1.

Contraction fails when $f(P^{**}) > \min(f(P_h), f(P^*))$ and in that case all vertices of simplex are replaced by $(P_i + P_l)/2$ and process moves to the next iteration. Otherwise P^{**} replaces P_h.

These steps are realized until the required stop criterion is reached and a minimum of a goal function is evaluated.

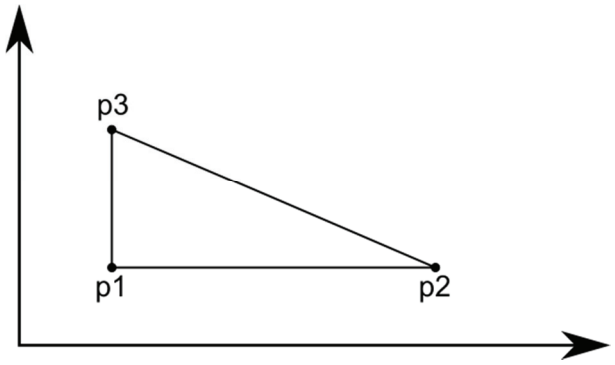


Fig. 4. Schematic illustration of a simplex algorithm for 2-dimensional optimization problem.

3.2. Finite element software and scripting module

Second module of the optimization system is commercial finite element Abaqus software. In this case FE model of the metalforming operation is also created automatically using developed and implemented set of scripts. Those scripts, use API (Application Programing Interface) of finite element software to create and run subsequent simulation tasks. Choice of programming language in this case depends on the FE software that is used. In the Abaqus application the most convenient solutions is to use feature called PDE (Python Development Environment). This is a full supported scripting Python language environment.

The developed scripting module is divided into following tasks:

– Importing the simulation setup generated by optimization module.
– Creation of finite element model which includes:
  - Automatic creation of geometry of subsequent parts: sample and tools (listing 1, figure 5),

```
defcreateDoubleRadiusSampleProfile(self):
   s = mdb. models[self. modelName].
ConstrainedSketch(name='__profile__',
      sheetSize=200. 0)
   g, v, d, c = s. geometry, s. vertices, s.
dimensions, s. constraints
      s. setPrimaryObject(option=STANDALONE)

   if self. dimensionsMap[self. R1_DIM] >= self.
dimensionsMap[self. R2_DIM]:
      self. dimensionsMap[self. TOTAL_WIDTH_DIM]
   = 2. 0 *
         self. dimensionsMap[self. R1_DIM]
      self. dimensionsMap[self.
TOTAL_HEIGHT_DIM] = 2. 0 *
         self. dimensionsMap[self. R2_DIM] +
self. dimensionsMap[self. H_DIM]

      a = math. pow(self. dimensionsMap[self.
H_DIM] / 2. 0, 2)
      b = math. pow(self. dimensionsMap[self.
R1_DIM] -
         self. dimensionsMap[self. R2_DIM], 2)
      x = math. sqrt(a - b)
      sketch_width = (self. dimensionsMap[self.
R1_DIM] *
         self. dimensionsMap[self. H_DIM]) / x
      sketch_height = (self. dimensionsMap[self.
R1_DIM] *
         self. dimensionsMap[self. H_DIM]) /
(self. dimensionsMap[self. R1_DIM] -
         self. dimensionsMap[self. R2_DIM])

      pointA = (-sketch_width / 2. 0, 0. 0)
      pointB = (0. 0, sketch_height / 2. 0)
      pointC = (sketch_width / 2. 0, 0. 0)
      pointD = (0. 0, -sketch_height / 2. 0)

      s. Line(pointA, pointB)
      s. Line(pointB, pointC)
      s. Line(pointC, pointD)
      s. Line(pointD, pointA)

      s. FilletByRadius(radius=self.
dimensionsMap[self. R1_DIM], curve1=g[5],
         nearPoint1=(pointD[0] + pointA[0] / 2,
pointD[1] + pointA[1] / 2),
         curve2=g[2], nearPoint2=(pointA[0] +
pointB[0] / 2, pointA[1] +
         pointB[1] / 2))
      s. FilletByRadius(radius=self.
dimensionsMap[self. R1_DIM], curve1=g[3],
         nearPoint1=(pointB[0] + pointC[0] / 2,
pointB[1] + pointC[1] / 2),
         curve2=g[4], nearPoint2=(pointC[0] +
pointD[0] / 2, pointC[1] +
         pointD[1] / 2))
      s. FilletByRadius(radius=self.
dimensionsMap[self. R2_DIM], curve1=g[2],
         nearPoint1=(pointA[0] + pointB[0] / 2,
pointA[1] + pointB[1] / 2),
         curve2=g[3], nearPoint2=(pointB[0] +
pointC[0] / 2, pointB[1] +
         pointC[1] / 2))
      s. FilletByRadius(radius=self.
dimensionsMap[self. R2_DIM], curve1=g[4],
         nearPoint1=(pointC[0] + pointD[0] / 2,
pointC[1] + pointD[1] / 2),
         curve2=g[5], nearPoint2=(pointD[0] +
pointA[0] / 2, pointD[1] +
         pointA[1] / 2))

   else:
      self. dimensionsMap[self. TOTAL_WIDTH_DIM]
= 2. 0 *
         self. dimensionsMap[self. R2_DIM]
      self. dimensionsMap[self.
TOTAL_HEIGHT_DIM] = 2. 0 *
         self. dimensionsMap[self. R2_DIM] +
self. dimensionsMap[self. H_DIM]

      s. Line(point1=(-self. dimensionsMap[self.
R2_DIM], -
         self. dimensionsMap[self. H_DIM] / 2.
0), point2=(-
         self. dimensionsMap[self. R2_DIM],
self. dimensionsMap[self. H_DIM] /
         2. 0))
      s. ArcByCenterEnds(center=(0. 0, self.
dimensionsMap[self. H_DIM] / 2. 0),
         point1=(-self. dimensionsMap[self.
R2_DIM], self. dimensionsMap[self. H_DIM]
         / 2. 0), point2=(self.
dimensionsMap[self. R2_DIM],
         self. dimensionsMap[self. H_DIM] / 2.
0), direction=CLOCKWISE)
      s. Line(point1=(self. dimensionsMap[self.
R2_DIM], self. dimensionsMap[self. H_DIM]
         / 2. 0), point2=(self.
dimensionsMap[self. R2_DIM], -
         self. dimensionsMap[self. H_DIM] / 2.
0))
      s. ArcByCenterEnds(center=(0. 0, -self.
dimensionsMap[self. H_DIM] / 2. 0),
         point1=(self. dimensionsMap[self.
R2_DIM], -self. dimensionsMap[self. H_DIM] /
         2. 0), point2=(-self.
dimensionsMap[self. R2_DIM], -
         self. dimensionsMap[self. H_DIM] / 2.
0), direction=CLOCKWISE)

      p = mdb. models[self. modelName].
Part(name=self. sampleName,
         dimensionality=TWO_D_PLANAR,
type=DEFORMABLE_BODY)
      p = mdb. models[self. modelName].
parts[self. sampleName]
      p. BaseShell(sketch=s)
```


```
      s. unsetPrimaryObject()

defloadToolGeometryFromFile(self, filePath):
      acis = mdb. openAcis(filePath,
scaleFromFile=OFF)
      mdb. models[self. modelName].
PartFromGeometryFile(
         name=self. toolName, geometryFile=acis,
combine=False,
         stitchAfterCombine=False,
dimensionality=TWO_D_PLANAR,
         type=DISCRETE_RIGID_SURFACE,
topology=WIRE)
      p = mdb. models[self. modelName].
parts[self. toolName]
      p. ReferencePoint(point=p. vertices[0])
```

Listing 1. *Developed script for automatic creation of geometry of subsequent parts.*

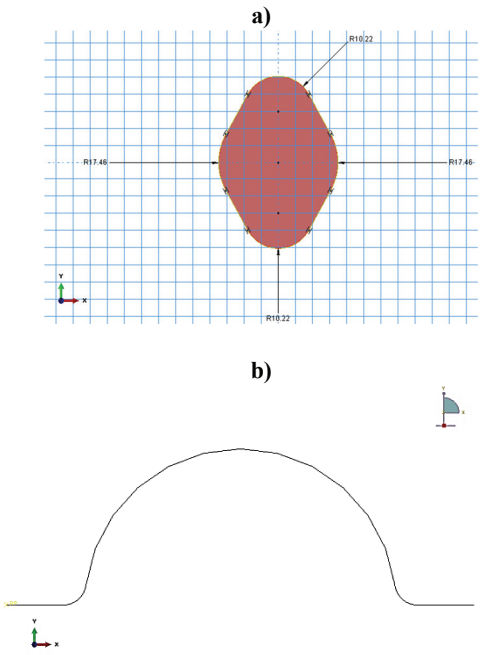


Fig. 5. *Result obtained from developed script a) geometry of the sample b) geometry of the tool.*

- Generation of the mesh for each part (listing 2, figure 6),

```
defsetSampleMesh(self, approxSeedSize):
      p = mdb. models[self. modelName].
parts[self. sampleName]
      f = p. faces
      pickedRegions = f.
getSequenceFromMask(mask=('[#1 ]', ), )
      p. setMeshControls(regions=pickedRegions,
elemShape=QUAD)
      p. seedPart(size=approxSeedSize,
deviationFactor=0. 1, minSizeFactor=0. 1)
      p = mdb. models[self. modelName].
parts[self. sampleName]
        p. generateMesh()

defcreateToolMesh(self, approxSeedSize):
      p = mdb. models[self. modelName].
parts[self. toolName]
      p. seedPart(size=approxSeedSize,
deviationFactor=0. 1, minSizeFactor=0. 1)
      p. generateMesh()
```

Listing 2. *Developed script for automatic finite element mesh generation.*

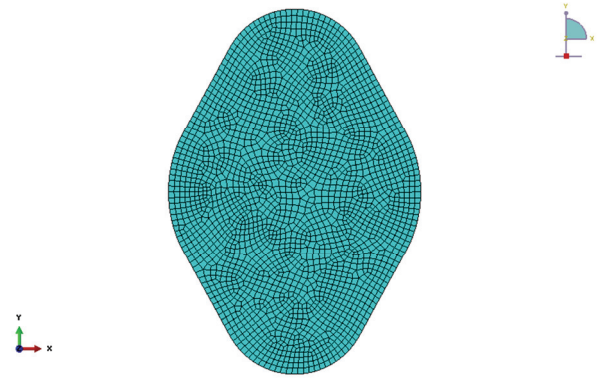


Fig. 6. *Result obtained from developed script for generation of the FE mesh. Quadratic elements are selected in this case.*

- Assembling the model (listing 3, figure 7)

```
defsetAssembly(self):
      a = mdb. models[self. modelName].
rootAssembly
      if self. topToolHeight> (self.
sampleHeight + self. minToolsGap) / 2:
         a. translate(instanceList=(self.
topToolInstanceName, ), vector=(-0. 5 *
            self. topToolWidth, 0. 5 * self.
minToolsGap, 0. 0))
      else:
         a. translate(instanceList=(self.
topToolInstanceName, ), vector=(-0. 5 *
         self. topToolWidth, (0. 5 + self.
initialToolGap) * self. sampleHeight -
         self. topToolHeight, 0. 0))
      if self. bottomToolHeight> (self.
sampleHeight + self. minToolsGap) / 2:
         a. rotate(instanceList=(self.
bottomToolInstanceName, ), axisPoint=(0. 0, 0.
0,
            0. 0), axisDirection=(0. 0, 0. 0, 1.
0), angle=180. 0)
         a. translate(instanceList=(self.
bottomToolInstanceName, ), vector=(0. 5 *
            self. bottomToolWidth, -(0. 5 *
self. minToolsGap), 0. 0))
      else:
         a. rotate(instanceList=(self.
bottomToolInstanceName, ), axisPoint=(0. 0, 0.
0,
            0. 0), axisDirection=(0. 0, 0. 0, 1.
0), angle=180. 0)
         a. translate(instanceList=(self.
bottomToolInstanceName, ), vector=(0. 5 *
            self. bottomToolWidth, -((0. 5 +
self. initialToolGap) * self. sampleHeight -
            self. bottomToolHeight), 0. 0))
```

Listing 3. *Developed script for automatic model assembly.*

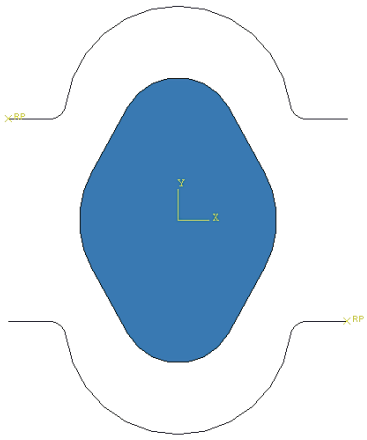


Fig. 7. *Result obtained from developed script for automatic model assembly.*

- Incorporation of the boundary conditions, interactions and constraints (listing 4, figure 8).

```
defsetGeneralContact(self):
      mdb. models[self. modelName].
ContactStd(name='General-contact',
         createStepName='Initial')
      mdb. models[self. modelName].
interactions[
         'General-ontact']. includedPairs.
setValuesInStep(
         stepName='Initial', useAllstar=ON)
      mdb. models[self. modelName].
interactions[
         'General-contact'].
contactPropertyAssignments. appendInStep(
         stepName='Initial',
assignments=((GLOBAL, SELF, 'Friction'), ))

defsetContactFriction(self, frictionCoeff):
   mdb. models[self. modelName].
ContactProperty('Friction')
   mdb. models[self. modelName].
interactionProperties['Friction'].
TangentialBehavior(
         formulation=PENALTY,
directionality=ISOTROPIC,
slipRateDependency=OFF,
         pressureDependency=OFF,
temperatureDependency=OFF, dependencies=0,
         table=((frictionCoeff, ), ),
shearStressLimit=None,
         maximumElasticSlip=FRACTION,
fraction=0. 005, elasticSlipStiffness=None)

defsetBoundaryConditions(self):
      topToolDisplacement = -((0. 5 + self.
initialToolGap) *
         self. sample. dimensionsMap[self.
TOTAL_HEIGHT_DIM] -
         self. topTool. getToolHeight() - 0. 5 *
self. minToolsGap)
      bottomToolDisplacement = (0. 5 + self.
initialToolGap) *
         self. sample. dimensionsMap[self.
TOTAL_HEIGHT_DIM] -
         self. bottomTool. getToolHeight() - 0.
5 * self. minToolsGap
      self. topTool. setToolDisplacement(self.
STEP_NAME, topToolDisplacement)
      self. bottomTool.
setToolDisplacement(self. STEP_NAME,
bottomToolDisplacement)
```

Listing 4. *Developed script for automatic assignment of boundary conditions, interactions and constraints.*

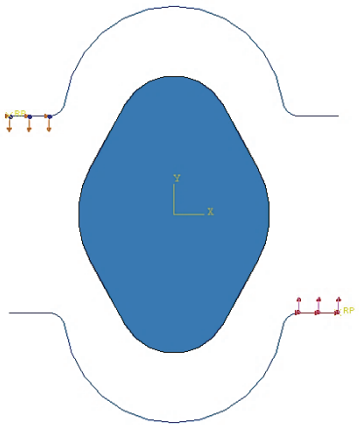


Fig. 8. *Result obtained from developed script for automatic assignment of boundary conditions, interactions and constraints.*

– Assigning number parameters for the solver e.g. number of increments.
– Initiating finite element calculations.
– Extracting selected results from output database and transferring them into the regular text file.

3.3. Communication interface

Both described above modules require proper communication interface, which allows data transfer between JAVA application and Python scripts (FE part). To provide interaction between optimization and finite element module XML standard was selected. The XML is a common standard of storing data that is both human- and machine-readable. Another advantage of this standard is that both JAVA and Python support libraries for parsing and handling XML files. Additionally in JAVA tool called JAXB (Java architecture for XML binding) is used, what facilitates implementation of data files processor, by mapping java classes to XML representations. The proposed structure of the XML file is presented in listing 5.

```
<simulationSetup>
      <simulationId>48beed13-ebc9-477a-a12c-
f60d88812caf</simulationId>
      <currentIteration>0</currentIteration>
      <simulationWorkspace>C:\AbaqusOptimization
App\</simulationWorkspace>
      <modelName>walcowanie</modelName>
      <minToolsGap>2. 0</minToolsGap>
      <sample name="sample" shape="DOUBLE_RADIUS
      " source="USER_DEFINED">
      <sampleDimmensions>
<dimmension>
<key>R1</key>
<value>17. 46040901912515</value>
</dimmension>
<dimmension>
<key>R2</key>
<value>16. 22028680689232</value>
</dimmension>
<dimmension>
<key>H</key>
<value>3. 0521387560752893</value>
</dimmension>
</sampleDimmensions>
      </sample>
      <tool name="bottom_tool" position="bottom"
source="FROM_FILE">
            <initialSampleToolGap>0.
25</initialSampleToolGap>
      <geometryFilePath>. . . \toolGeometry.
sat</geometryFilePath>
      </tool>
      <tool name="top_tool" position="top"
source="FROM_FILE">
      <initialSampleToolGap>0.
25</initialSampleToolGap>
            <geometryFilePath>. . .
\toolGeometry. sat</geometryFilePath>
</tool>
<sampleMaterial name="steel_alloy">
      <density>7. 95E-9</density>
      <youngModulus>210000. 0</youngModulus>
      <poissonsRatio>0. 3</poissonsRatio>
            <stressStrainValue>
      <strain>0. 0</strain>
      <stress>190. 0</stress>
      </stressStrainValue>

            . . .

            <stressStrainValue>
      <strain>0. 6</strain>
      <stress>767. 3321217</stress>
      </stressStrainValue>
```


```
</sampleMaterial>
</simulationSetup>
```

Listing5. *Example of XML file containing simulation setup.*

Setup file is divided into 5 sections: *general simulation setup, sample, top tool, bottom tool, sample material*. Those tags, which are used in the setup file have human-readable form to allow additional analysis of the performed simulation.

General simulation setup contains items that are necessary to create and save model and output database files. It also contains <*minToolsGap*> element, which defines distance between tools at the end of the process. Sample section defines single name, which is necessary to create model, sample shape and sample shape source. *USER_DEFINED* in code means that shape should be drawn using provided sample dimensions. Both top and bottom tool sections contain similar items to sample section but since tools shape source are set to *FROM_FILE*, file path needs to be provided. In addition tool section contains <*sampleToolGap*> element, which defines initial distance between tool and sample. That parameter is useful when there is large difference in shape of this two elements. Last section contains sample material properties that are necessary to define material that is further used in the simulation. It contains both elastic and plastic properties including flow stress model in tabular form.

Combination of described modules provides a possibility to automatically run and optimise the metalforming process parameters. An optimisation of the rod rolling operations was selected in the present work as a case study to show capabilities of the developed system. Defined optimisation task and obtained results are described in the following part of this work.

4. CASE STUDY OF THE DEVELOPED OPTIMISATION SYSTEM

The problem of manufacturing rods with uniform distribution of properties at the cross section is investigated within the present work. The rod rolling is a combination of several passes realized in properly designed rolls. Subsequent deformations play a role in development of appropriate shape as well as required properties of the final product. Particularly important are last two stages of rod rolling changing the shape from oval to required circular one, as seen in figure 9.

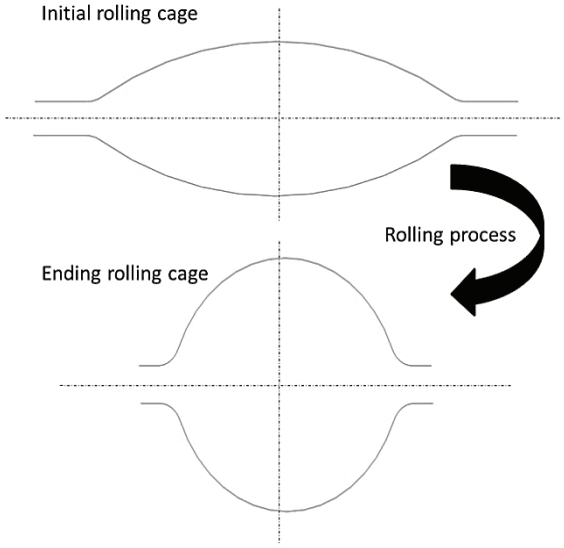


Fig. 9. *Schematic illustration representing initial and final cages used in the rod rolling process.*

Proper design of rolls with oval shape provides possibility to obtain required product properties after final rolling in the rolls with circular shape. Thus, the objective of this work is to identify proper shape of the sample prior final rolling to provide uniform strain distribution at the cross section in the final product. Two independent optimization variables were defined: r_1 and r_2, respectively as seen in figure 10. These two radiuses are required to describe elliptical shape of the sample/rolls prior final rolling. Additionally, during the calculation a constant volume condition was enforced by the following equation:

$$S_{sample} = \frac{\sqrt{H^2 - 4(r_1 - r_2)^2}\,H^2(r_1 + r_2) + \pi H R_1^2 (r_1 - r_2)}{H^2} \tag{7}$$

where: S_{sample} – sample area; H, r_1, r_2 – sample dimensions.

The goal function is defined as a square root error between average and actual equivalent strain calculated in subsequent finite element Gauss points at the cross section of the sample:

$$\Phi = \sqrt{\sum_{i=1}^{N} \frac{(\varepsilon_{avg} - \varepsilon_i)^2}{N}} \tag{8}$$

where: N – number of finite element Gauss points, ε_{avg} – average strain value calculate from hole model, ε_i – strain value in the i-th node.

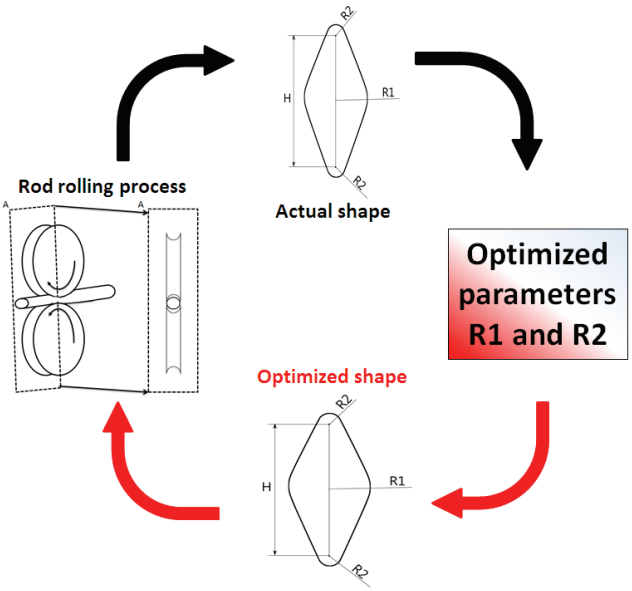


Fig. 10. Schematic illustration presenting optimization procedure during rod rolling process.

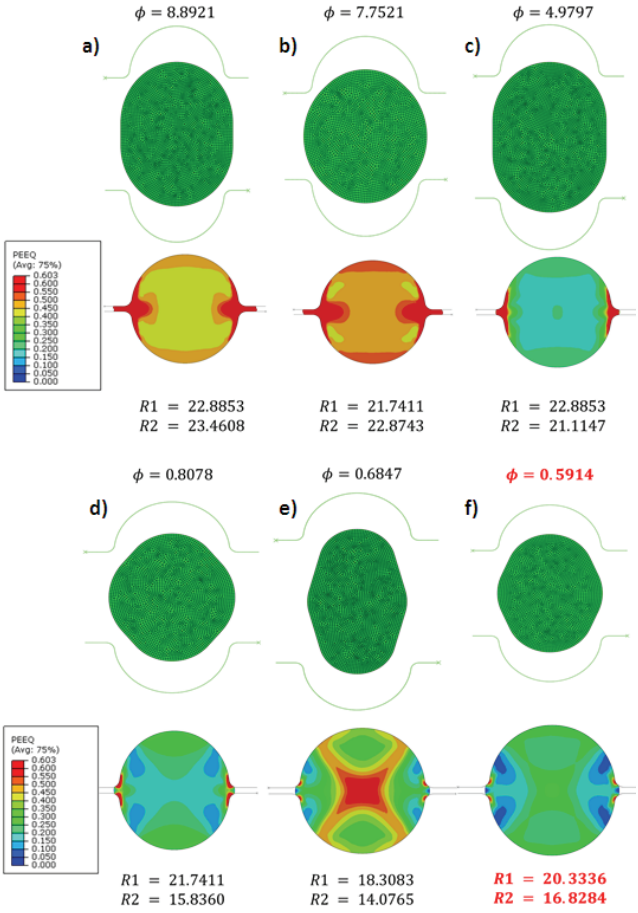


Fig. 11. Selected examples of obtained results of initial and final shape of the rod with corresponding equivalent strain distribution at the cross section.

The objective function is calculated in the JAVA optimization module on the basis of data extracted by Python scripts from finite element solution. The stop criterion in the investigated case was set as 30 Simplex optimization steps. Initial simplex was generated randomly in the possible solution space. Examples of obtained results during subsequent optimization steps are shown in figure 11.

Change in the objective function during 30 cycles of Simplex algorithm is present in figure 12.

As seen in figure 12, mean square root error has been reduced by 93.4%, however it should be mentioned that some of initial random points coordinates exceed their boundary values and as a result penalty has been applied to the objective function value at that point. Another interesting fact is that even initial simplex contained undesirable solutions, which were distorted by penalty function, downhill algorithm managed to direct simplex into acceptable range of independent variables. After 5 cycles all potential solutions were inside prescribed range. Presented optimization problem required 127 numerical simulations to perform 30 optimization cycles, however objective function values close to the given solution were reached after 19 cycles and additional stop condition could be applied to reduce duration of optimization process.

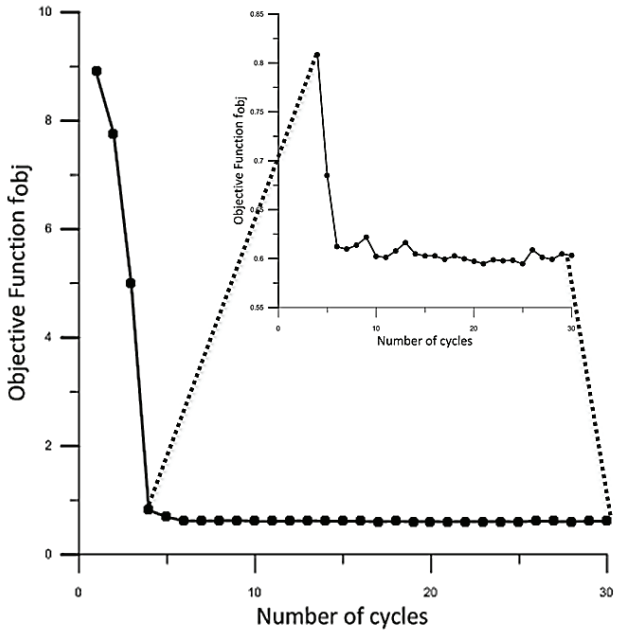


Fig. 12. Change in the objective function during 30 cycles of the Simplex algorithm.

5. CONCLUSIONS

The developed automatic system is based on numerical simulations of production process, supported by optimization module and data transfer protocols. To obtain such tool a combination of various programming technologies was required. Particularly important stage was efficient design of

data transfer solutions between finite element and optimization module. The XML technology seems to be a good solution to this problem. Selection of scripting language for automatic creation of finite element model is forced by the finite element software. In the present work finite element program supports Python scripting language. Examples of developed and implemented functions can be easily used as a user guide for scientists working with similar problems.

Presented optimization tests proved high efficiency of the system in practical applications. Results confirmed usefulness of such automatic system for metal forming engineers. Thus, optimization of the production chain with product properties included in the objective function is possible, but require adding new modelling components, which will able to simulate microstructure evolution during deformation. This will be the subject of further research.

ACKNOWLEDGMENT

Financial support provided by National Centre for Research and Development (NCBIR), project no. R07 0006 10, is acknowledged. FEM calculations were realized at the ACK CYFRONET AGH MNiSW/IBM_BC_HS21/AGH/075/2010.

REFERENCES

Bariani, P. F., Bruschi, S., Ghiotti, A., 2007, Material testing and physical simulation in modelling process chains based on forging operations, *Computer Methods in Materials Science*, 7, 378-382.

Beladi, H., Adachi, Y., Timokhina, I., Hodgson, P. D., 2009, Crystallographic analysis of nanobainitic steels, *Scripta Materialia*, 60, 455-458.

Bruccoleri, M., Lo Nigro, G., Perrone, G., Renna, P., Noto La Diega, S., 2005, Production planning in reconfigurable enterprises and reconfigurable manufacturing systems, *Annals of the CIRP*, 54, 433-436.

Feng, C., Gao, X. N., Tang, Y. T., Zhang, Y., 2013, Comparative life cycle environmental assessment of flue gas desulphurization technologies in China, *Journal of Cleaner Production*, in press.

Hon, K. K. B., Xu, S., 2007, Impact of product life cycle on manufacturing systems reconfiguration, *Annals of the CIRP*, 56, 455-458.

Legwand, A., Perzynski, K., 2012, New numerical software approach with shape optimization applied on the landing gear used in construction of ultralight airplanes, *Zeszyty Studenckiego Towarzystwa Naukowego*, 25, 181-187.

Madej, L., Szeliga, D., Kuziak, R., Pietrzyk, M., 2007, Physical and numerical modelling of forging accounting for exploitation properties of products, *Computer Methods in Material Science*, 7, 397-405.

Muszka, K., Sun, L., Wynne, B. P., Palmiere, E. J., Rainforth, W.M., 2012, On the effect of strain reversal on static recrystallisation and strain-induced precipitation process kinetics in microalloyed steels, *Materials Science Forum*, 715-716, 655-660.

Nedler, J. A., Meadf, R., 1965, A simplex method for function minimization, *The Computer Journal*, 7(4), 308-313.

Pereira, J., Paulre, B., 2001, Flexibility in manufacturing systems: a relational and a dynamic approach, *European J. Operational Research*, 130, 70-82.

Pietrzyk, M., 2001, Identification of Parameters in the History Dependent Constitutive Model for Steels, *Annals of the CIRP*, 50, 161-164.

Pietrzyk, M., Kedzierski, Z., Kusiak, H., Madej, W., Lenard J. G., 1993, Evolution of the Microstructure in the Hot Rolling Process, *Steel Research International*, 64, 549-556.

Pietrzyk, M., Kuziak, R., 2004, Development of the Constitutive Law for Microalloyed Steels Deformed in the Two-Phase Range of Temperatures, *Steel GRIPS*, 2, 465-470.

Pietrzyk, M., Lenard, J. G., Dalton, G. M., 1993, A Study of the Plane Strain Compression Test, *Annals CIRP*, 42, 331-334.

Pietrzyk, M., Madej, L., Kuziak, R., 2010, Optimal design of manufacturing chain based on forging for copper alloys, with product properties being the objective function, *The CIRP Annals*, 59, 319-322.

Rauch, L., Madej, L., Weglarczyk, S., Pietrzyk, M., 2008, System for design of the manufacturing process of connecting parts for automotive industry, *Archives of Civil and Mechanical Engineering*, 8, 157-165.

Robertson, L. T., Hilditch, T. B., Hodgson, P. D., 2008, The effect of prestrain and bake hardening on the low-cycle fatigue properties of TRIP steel, *International Journal of Fatigue*, 30, 587-594.

Senkov, O. N., Miracle, D. B., Firstov, S. A., 2004, Metalic Materials with high structural efficiency, *NATO Science Series - Mathematics, Physics and Chemistry*, 146.

Timokhina, I. B., Hodgson, P. D., Ringer, S. P., Zheng, R. K., Pereloma, E. V., 2007, Precipitate characterisation of an advanced high-strength low-alloy (HSLA) steel using atom probe tomography, *Scripta Materialia*, 56, 601-604.

WYKORZYSTANIE METOD OPTYMALIZACJI W ZAGADNIENIACH KOMPUTEROWEGO WSPOMAGANIA PROJEKTOWANIA PROCESÓW PRZERÓBKI PLASTYCZNEJ

Streszczenie

Praca porusza zagadnienia opracowania automatycznego systemu komputerowego do optymalizacji kolejnych etapów łańcucha produkcyjnego stosowanego w procesie przeróbki cieplno-plastycznej metali i ich stopów. Takie parametry jak rozkład naprężeń, odkształceń czy wielkości ziarna po odkształceniu istotnie wpływają na zachowanie materiału podczas dalszej eksploatacji. Dlatego też, własności produktu, które zazwyczaj powinny być równomiernie w całej objętości wyrobu są podstawą do zdefiniowania funkcji celu dla problemu optymalizacyjnego. Głównym celem pracy jest opracowanie złożonego systemu umożliwiającego automatyczną optymalizację kolejnych etapów cyklu produkcyjnego tak aby możliwe było uzyskiwanie wyrobów

o wymaganych parametrach. Opis poszczególnych komponentów odpowiedzialnych za automatyczne generowanie modeli numerycznych, dyskretyzację siatką elementów skończonych, optymalizację oraz transfer danych pomiędzy kolejnymi modułami zostanie szczegółowo przedstawiony w ramach niniejszej pracy. Jako przykład praktycznego zastosowania opracowanego systemu wybrano proces walcowania prętów.

Received: August 19, 2013
Received in a revised form: December 12, 2013
Accepted: December 28, 2013