12th INTERNATIONAL SCIENTIFIC CONFERENCE ON DEFENSIVE TECHNOLOGIES
OTEH 2026
Serbia, Tara, 21 - 24 October 2026

AUTOMATING DIGITAL MAP PRODUCTION PROCESSES USING PYTHON GIS ALGORITHMS

MARKO SIMIĆ
Military Geographical Institute – "General Stevan Bošković", Belgrade, marko.simic@vs.rs

SAŠA BAKRAČ
Military Geographical Institute – "General Stevan Bošković", Belgrade, sasa.bakrac@vs.rs

ĐORĐE ĐERMANOVIĆ
Military Geographical Institute – "General Stevan Bošković", Belgrade, djordjedjermanovic99@gmail.com

MATIJA MILIĆ
University of Niš, Faculty of Science and Mathematics, Niš, matija.milic@pmf.edu.rs

SLAVIŠA TATOMIROVIĆ
Faculty of Diplomacy and Security, Union University – "Nikola Tesla", Belgrade, Slavisat04@gmail.com

Abstract: *The development of Python-based algorithms has enabled the automation of selected processes in digital topographic map production at the Military Geographical Institute – "General Stevan Bošković" (MGI). In this study, automation is presented as a broader strategy for improving efficiency, consistency, and sustainability in GIS-supported map production workflows. Algorithm development was moved from ArcMap to ArcGIS Pro in order to support the transition from Python 2 to Python 3 and to provide a more modern, maintainable, and extensible software environment. Only algorithms that depend on specific ArcMap functionality, such as legacy representation features, were retained in the ArcMap environment. Implemented as native GIS tools, the algorithms incorporate institutional cartographic rules into modular geoprocessing procedures that can be used by both developers and GIS operators. As an illustrative example, the paper highlights the automation of content alignment along adjacent map sheet boundaries in ArcGIS Pro, a traditionally labor-intensive task requiring detailed visual inspection and manual correction. Particular attention is given to the Python-based processing pipeline and to the formulation of the functions used for geometry refinement, consistency control, and operator-guided validation of remaining discrepancies. The migration to ArcGIS Pro also improved compatibility with current GIS technologies and established a stronger foundation for future upgrades, including adaptation to different cartographic products and scales. The results show that Python GIS algorithms can significantly reduce manual intervention, improve workflow consistency, and provide a transferable framework for the modernization of digital map production processes.*

Keywords: *GIS, Python, Geoprocessing Automation, Digital Topographic Maps.*

1. INTRODUCTION

Digital topographic map production is no longer limited to the visual design and reproduction of cartographic content. In modern production systems, topographic maps are created from complex geospatial databases in which geometric accuracy, attribute consistency, topological correctness, and repeatability of procedures are equally important. This has increased the importance of automated and semi-automated GIS workflows, especially in institutions responsible for systematic map production [1].

At the same time, the growing volume of geospatial data has made purely manual cartographic procedures increasingly difficult to maintain. Tasks that were traditionally performed through visual inspection and manual editing often require repeated execution on large numbers of features. Such procedures are time-consuming and may lead to inconsistent results, since repetitive manual editing increases operator fatigue and the possibility of overlooked error. For this reason, automation has become an important element of digital cartography, contributing to reduced processing time, standardized production rules, and improved quality of the final geospatial database.

Python has become one of the most widely used programming languages for GIS automation because it provides a flexible environment for processing spatial data and building customized geoprocessing workflows. In GIS applications, Python is commonly used for data conversion, spatial analysis, topology control, batch processing, and the integration of multiple geoprocessing operations into repeatable procedures. Through libraries such as ArcPy, PyQGIS, GDAL/OGR, GeoPandas, Shapely, and PyProj, Python enables the development of tools for practical production procedures in which institutional production guidelines are directly

Corresponding author: M. Simić;
E-mail: marko.simic@vs.rs

DOI: 10.5937/OTEH26099S

implemented [2, 3].

The role of Python in geospatial applications is not limited to conventional geoprocessing automation. Python-based workflows are also increasingly employed for the integration of GIS and remote-sensing data, automated spatial analysis, and machine- and deep-learning applications [4–6].

Within the ArcGIS environment, Python has an important role because it allows users to transform individual geoprocessing operations into structured tools that can be used directly from the program interface. Depending on the complexity of the task, Python code can be implemented as standalone scripts, script tools, or Python toolboxes. These approaches make it possible to automate repetitive procedures, define user parameters, control input data, and integrate developed tools into existing production workflows [7, 8]. Therefore, Python-based automation represents a practical bridge between software development and everyday cartographic production.

In the production of digital topographic maps at the Military Geographical Institute – "General Stevan Bošković" (MGI), Python-based algorithms have been developed to support selected cartographic and geoprocessing procedures. These algorithms are designed to incorporate institutional cartographic rules into repeatable GIS tools, while still allowing operator supervision where full automation is not appropriate. This is particularly important in topographic map production, where the final result must satisfy both geometric and cartographic requirements. Earlier automation procedures were primarily developed in the ArcMap environment and were based on Python 2. Although these tools provided practical benefits in production, the long-term sustainability of such workflows depends on their modernization and adaptation to current GIS technologies [9]. The transition from ArcMap to ArcGIS Pro and from Python 2 to Python 3 requires the revision of existing algorithms, adaptation of geoprocessing procedures, restructuring of code, and evaluation of which functions can be transferred to the new environment and which must remain connected to legacy ArcMap functionality.

An analysis of Python GIS algorithms is conducted within the broader framework of modernizing digital topographic map production. The focus is placed on the organization of automated workflows, the migration of algorithm development toward ArcGIS Pro, and the use of modular geoprocessing procedures. Content alignment along adjacent map sheet boundaries is used as an example of this approach because it represents a complex and traditionally labor-intensive procedure that combines geometry refinement, consistency control, and operator-guided validation. More broadly, the migration to Python 3 and ArcGIS Pro establishes a sustainable framework for the automation of additional production tasks, such as cartographic generalization in the derivation of smaller-scale maps from the 1:25,000 digital topographic map (DTM25).

2. MIGRATION FROM PYTHON 2 TO PYTHON 3

The migration from Python 2 to Python 3 represents an important step in the modernization of GIS-supported cartographic production. Earlier automation procedures were developed in the ArcMap environment which relied on Python 2. Although these tools were functional and useful in production, their long-term use is limited by software obsolescence, reduced compatibility with modern libraries, and the gradual transition of GIS platforms toward Python 3 [10]. ArcGIS Pro uses Python 3 and represents the current development environment for Python-based GIS automation in the Esri ecosystem.

This migration is not only a technical update of programming syntax, but also a transition toward a more sustainable production framework. Maintaining algorithms in an outdated software environment can create several practical problems, including limited support for new geoprocessing tools, difficulties in integrating updated Python libraries, reduced compatibility with new data formats, and increased dependence on legacy workflows. Since ArcGIS Desktop, including ArcMap, has reached the end of its product life cycle, migration to ArcGIS Pro is also necessary for ensuring the continuity of institutional GIS workflows.

One of the main advantages of migration to ArcGIS Pro is the use of a modern Python 3 environment managed through Conda. This enables better control of Python packages, easier management of project-specific environments, and access to updated scientific and geospatial libraries. This environment contains ArcPy and additional Python libraries used for geoprocessing, data analysis, and automation [11].

However, migration from Python 2 to Python 3 also requires careful revision of existing algorithms. Certain changes in syntax, string handling, dictionary behavior, module organization, and library compatibility can require modification of older code. Automated conversion tools can assist in this process, but they do not fully replace manual inspection and testing. For this reason, migration represents a controlled redesign of the workflow rather than a simple conversion of scripts from one Python version to another [12].

In the context of digital topographic map production at the MGI, this process involved analysing the existing ArcMap-based algorithms and determining how their logic could be transferred to the ArcGIS Pro environment. For the workflow presented in this study, complete migration was possible because the procedure did not depend on legacy cartographic representation mechanisms specific to ArcMap. The algorithms were therefore redesigned for Python 3 and ArcGIS Pro while preserving the original cartographic logic. In this way, an established production task was transferred into a modern software environment, improving maintainability, compatibility with current GIS technologies, and the potential for future expansion.

By adapting the tools to a modern GIS environment, the code becomes easier to update, document, test, and extend. This is especially important in institutional cartographic

production, where tools are not created for one-time use, but must remain operational across multiple production cycles. A modernized Python 3 workflow also makes it easier to introduce additional automated procedures, connect different processing stages, and gradually reduce repetitive manual work.

3. MATERIALS AND METHODS

DTM25 production at the MGI includes several interconnected phases, such as editorial preparation, aerial photogrammetric survey, photogrammetric mapping, field verification and completion, cartographic processing, and reproduction [9]. This paper focuses on the cartographic processing phase, which includes numerous repetitive geoprocessing operations that require consistency, precision, and repeatability across different production tasks.

The selected production procedure used for methodological demonstration is the alignment of content along adjacent DTM25 map sheet boundaries. This procedure was selected because the alignment of adjacent map sheets requires a combination of interrelated operations, including the detection of features near sheet boundaries, geometric refinement of line and polygon features, control of topological continuity between neighbouring sheets, and identification of remaining discrepancies for operator review. In manual workflows, this task requires detailed visual inspection of all feature vectors located near the map frame. Small geometric offsets, gaps, or overlaps may remain unnoticed at the standard display scale, especially in areas with dense content such as contour lines, hydrography, or communication networks.

The automated workflow is organized around three production scenarios. The first scenario refers to the case in which the adjacent sheet has already been completed, so the active sheet is adjusted according to the finalized geometry of the neighbouring sheet. The second scenario refers to the simultaneous processing of two adjacent sheets, where both sheets are adjusted in a coordinated manner. The third scenario refers to the preparation of an active sheet when the neighbouring sheet has not yet entered the cartographic processing phase. These scenarios correspond to unidirectional, bidirectional, and preparatory alignment procedures.

The input data consist of DTM25 feature classes stored in production geodatabases, together with the sheet index polygon layer used to define the boundaries of adjacent map sheets. The procedure processes line and polygon feature classes separately because these geometry types require different correction logic. Line features include contour lines, roads, railways, flowing water, communication objects, and other linear cartographic elements. Polygon features include vegetation areas, water bodies, marshland, and other surface elements (Figure 1.).

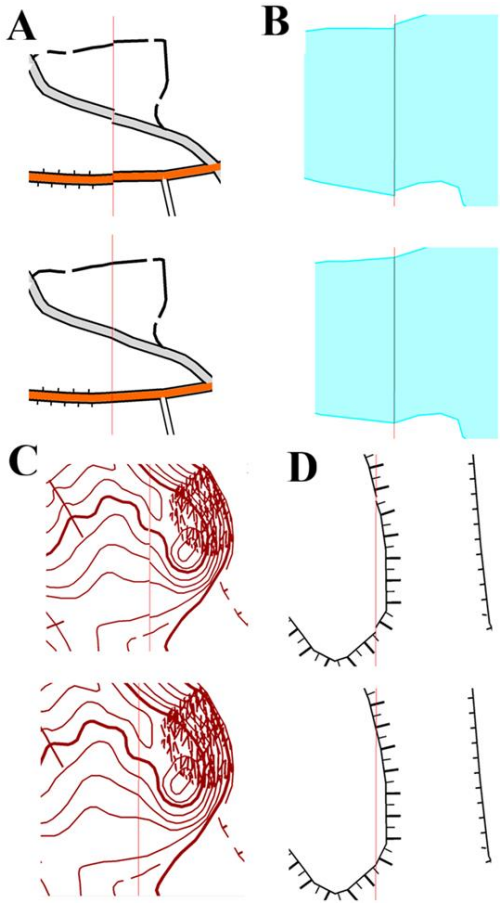


Figure 1. Examples of communication networks (A), hydrology (B), contour lines (C), and relief (D) at TK25 sheet boundaries (red line) before (top image) and after (bottom image) automatic vector processing

Point features are used as control outputs, marking locations where automatic processing detected unresolved or potentially problematic geometry (Figure 2.).

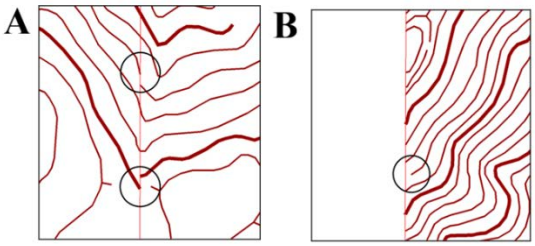


Figure 2. Point indicators (displayed as circles) marking unresolved contour-line geometry in bidirectional alignment (A) and preparatory alignment (B)

The workflow was implemented in ArcGIS Pro using Python 3 and ArcPy geoprocessing functions. The tools were designed as modular GIS procedures that can be executed from the ArcGIS Pro interface, allowing operators to use them without direct interaction with the source code. Before running the procedure, the operator defines the relevant input geodatabases, the map sheet pair, the feature classes to be processed, and the tolerance values for geometry correction. Repetitive and rule-based operations are performed automatically, while the operator remains responsible for parameter selection and validation of marked discrepancies making the workflow semi-

automated.

The general processing sequence consists of four main steps. First, user-defined parameters are read and assigned to internal variables. Second, the polygons of the selected DTM25 sheet pair are extracted from the sheet index layer and used as spatial references for selecting features along the boundary zone. Third, separate processing objects are initialized for line and polygon feature classes. Fourth, the tool performs hierarchical processing. It first iterates through each selected sheet pair and then through the selected feature classes within that pair. This structure allows the same logic to be applied to multiple sheets and multiple cartographic layers without rewriting the procedure for each individual case.

To make the processing logic more explicit, the workflow can be represented as a Python pipeline composed of reusable geoprocessing functions. This functional representation separates the algorithm into elementary operations. Formulating the process in this manner is useful because it shows the order of execution, and how the same functions can be reused in different production scenarios.

If F_L is defined as the function for processing line features, F_G as the function for processing polygon features, and P_L as the function for creating point features that indicate changes or unresolved discrepancies, the procedures can be represented as follows.

Active and Passive Replic

For line features, the active sheet is adjusted according to the passive, already finalized neighbouring sheet:

$$F_L(e) = (S_2 \circ L \circ S \circ I_2)(e) \quad (1)$$

$$P_L(e) = \big(P \circ M(I, I_2)\big)(e) \quad (2)$$

For polygon features, the algorithm first selects the relevant polygon elements on both sheets and then applies snapping and densification along the boundary zone:

$$G(e) = (L_A \circ L_1)(e) \quad (3)$$

$$G_2(e) = (L_A \circ L_2)(e) \quad (4)$$

Two Active Replicas

In the case of two active replicas, the alignment is performed in two phases because both neighbouring sheets are still in production.

For line features, the first phase prepares and adjusts selected features from both sheets:

$$B(e) = (I \circ L_1)(e) \quad (6)$$

$$B_2(e) = (I_2 \circ L_2)(e) \quad (7)$$

$$F_{L1}(e) = (S_2 \circ L \circ S_3(B_2, L_1))(e) \quad (8)$$

In the second phase, the corresponding features are adjusted in the opposite direction and point indicators are created for operator verification:

$$F_{L2}(e) = S_3(B, L_2)(e) \quad (9)$$

$$P_L(e) = \big(P \circ M_2(B, B_2)\big)(e) \quad (10)$$

For polygon features, the first phase selects and adjusts the relevant polygon elements:

$$G(e) = (L_A \circ L_1)(e) \quad (11)$$

$$G_2(e) = (L_A \circ L_2)(e) \quad (12)$$

$$F_{G1}(e) = (S_b \circ G_2)(e) \quad (13)$$

The second phase performs boundary refinement using snapping, densification, and intersection-based correction:

$$F_{G2}(e) = Sp(D \circ Sp(L_F \circ G, PP), IP(G_2))(e) \quad (14)$$

Active Replica Only

When only the active sheet is available, the procedure prepares its boundary content for later integration with the neighbouring sheet.

For line features:

$$F_L(e) = S_4(e) \quad (15)$$

$$P_L(e) = (H \circ C)(e) \quad (16)$$

For polygon features:

$$F_G(e) = S_b(e) \quad (17)$$

Definition of Pipeline Functions

The pipeline formulas use the following variables:

– e and f represent the features that are being processed;
– ω represents an auxiliary or intermediate feature used as an additional input to an operation;
– $TK25_1$ represents the polygon of the first sheet;
– $TK25_2$ represents the polygon of the second sheet;
– b represents the distance from the sheet frame in meters;
– c represents the feature type code used for selecting specific polygon features.

The line-feature functions used in the pipeline are defined as follows:

– $I(f)$ = intersect $(f, TK25_2)$
– $I_2(f)$ = intersect $(f, TK25_1)$
– $S(f)$ = snap $(f, I_2(f))$
– $L(f)$ = select_fc_that_isnt_snaped $(f, TK25_1)$
– $S_2(f)$ = snap $(f, TK25_2)$
– $M(f, \omega)$ = merge (f, ω)
– $P(f)$ = create_changes (f)
– C = create_parallel_line_to_frame $(TK25_1, TK25_2, a)$ – where $a = b - 3$.

– $H(f)$ = create_correction_lines (f, $TK25_2$, C)
– $L_1(f)$ = select_fc_on_replica (f, $TK25_1$)
– $L_2(f)$ = select_fc_on_replica (f, $TK25_2$)
– $S_3(f, \omega)$ = snap (f, ω)
– $M_2(f, \omega)$ = merge (f, ω)
– $S_4(f)$ = snap $(f, TK25_2)$

The polygon-feature functions used in the pipeline are defined as follows:

– $L_1(f)$ = select_fc_on_replica $(f, TK25_1)$
– $L_2(f)$ = select_fc_on_replica $(f, TK25_2)$
– $L_A(f)$ = select_features (f, c)
– PP = create_parallel_line_to_frame $(TK25_1, TK25_2, b)$
– $IP(f)$ = create_intersection_fc_with_polygon (f, $TK25_1$)
– $L_F(f)$ = select_fc_on_frame_border (f, PP)
– $S_p(f,\omega)$ = snap (f, ω)
– $D(f)$ = densify_fc (f)
– $S_b(f)$ = snap $(f, TK25_1)$

The python class methods used in these functions represent the elementary geoprocessing operations from which the complete workflow is composed. In the following method definitions, a, b, and c are used as generic input arguments:

– intersect (a, b) – performs the intersection of line object a with polygon b and creates intersection points.
– snap (a, b) – snaps points of object a to standalone points, points within a line object, or boundary lines of polygon object b.
– select_fc_that_isnt_snapped (a, b) – selects line objects a that touch the boundary of the adjacent sheet polygon b.
– merge (a, b) – merges objects a and b into a new object.
– create_changes (a) – creates a point object indicating lines of object a that require operator verification.
– create_parallel_line_to_frame (a, b, c) – creates a line at distance c, parallel to the frame of polygon b, and located on sheet polygon a.
– create_correction_lines (a, b, c) – creates a point object indicating lines of object a that cross the auxiliary line c, but do not touch the boundary of the adjacent sheet polygon b.
– select_fc_on_replica (a, b) – selects line or polygon objects a located on sheet polygon b.
– select_features (a, b) – selects polygon objects a with attribute code b.
– create_intersection_fc_with_polygon (a, b) – creates intersection lines between polygon object a and sheet polygon b.
– select_fc_on_frame_border (a, b) – selects polygon objects a that intersect line b.
– densify_fc (a) – increases the number of vertices in polygon object a, improving the precision of subsequent snapping and boundary refinement.

This formulation presents the workflow as a structured sequence of reusable geoprocessing functions rather than as a simple execution of individual GIS tools. Such an approach makes the logic of the algorithm more transparent and provides a basis for applying the same principle to similar cartographic production tasks. This structure can also be presented through workflow diagrams, as shown in Figures 3 and 4.

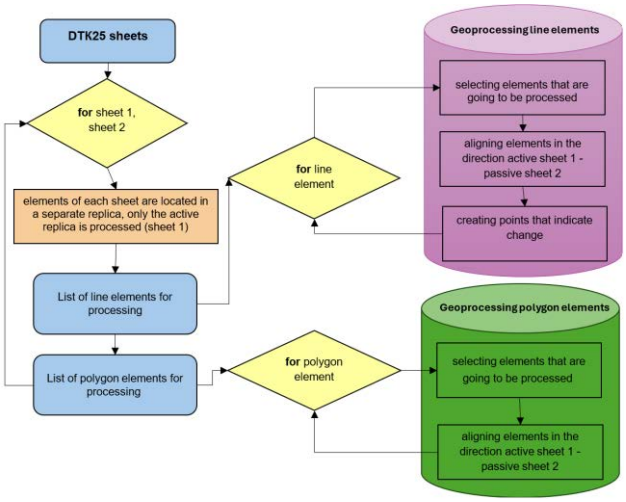


Figure 3. Diagram illustrating the content alignment workflow in the Active & Passive Replica tool

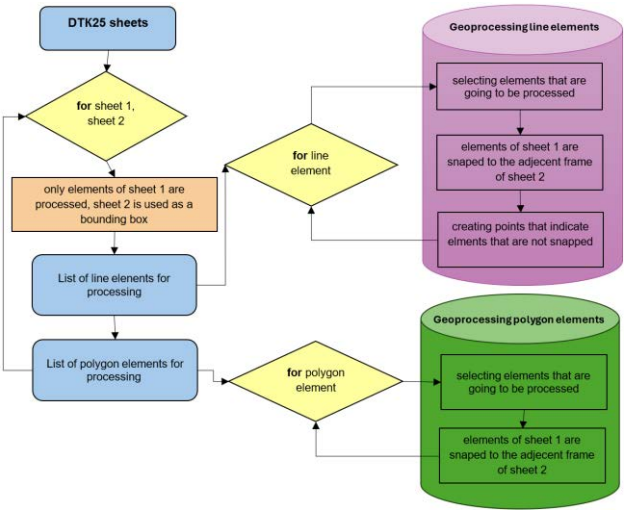


Figure 4. Diagram illustrating the content alignment workflow in the Active & Passive Replica tool

4. RESULTS AND DISCUSSION

The developed Python GIS workflow was tested on a series of adjacent DTM25 sheet pairs produced at the MGI. The main purpose of testing was to evaluate whether the developed workflow could correctly identify, refine, and control boundary geometries in different production scenarios.

The results showed that the automated processing successfully resolved most geometric inconsistencies that occurred within the defined tolerance values. In the case of line features, offsets and interruptions along sheet boundaries were corrected. The most visible improvement was observed for contour lines and road features, where manual correction is usually time-consuming because of

the number and density of elements along the map frame.

For polygon features, the workflow reduced unwanted gaps and overlaps between neighbouring boundary elements. This was especially important for large surface features that extend across sheet borders, such as vegetation areas, water surfaces, marshland, and similar polygonal elements.

The generated point control layers also proved to be useful. Instead of visually inspecting the entire sheet boundary, the operator was directed to specific locations where unresolved discrepancies or potentially problematic geometries remained. As a result, the operator's work was concentrated on complex cases that required expert assessment, while repetitive geometric corrections were performed automatically.

The effect of automation was also evaluated through the time required for sheet-boundary alignment. According to production records, the conventional manual procedure required approximately 5 working hours to align the content between two adjacent DTM25 sheets. Since one DTM25 sheet may need to be aligned with four neighbouring sheets, the total time for a complete sheet could reach approximately 20 working hours. After applying the Python GIS workflow, the average processing time was reduced to approximately 40 minutes per adjacent-sheet operation, or about 2.7 working hours for a complete sheet, including script execution, preliminary sheet analysis, and subsequent correction of marked discrepancies. This represents a time saving of more than 85% compared with the conventional procedure and an efficiency increase of more than seven times.

Overall, the results indicate that the developed Python GIS workflow provides a practical basis for semi-automated cartographic processing. The tested functions showed repeatable applicability across different feature classes and production scenarios. This confirms that the selected procedures can be performed in ArcGIS Pro while preserving the original production logic, making the workflow suitable for use in a real cartographic production environment. These results also show that the same principle can be extended to other DTM25 production tasks in which repetitive geoprocessing operations are present.

Expressing the workflow through clearly defined reusable geoprocessing functions makes the algorithm easier to analyse, test, modify, and extend. It also allows individual parts of the workflow to be adapted for other production tasks without redesigning the entire procedure.

5. CONCLUSION

Python GIS algorithms provide an effective basis for modernizing procedures in digital topographic map production. By transferring the workflow to ArcGIS Pro and Python 3, the developed solution was adapted to a current software environment while preserving the cartographic logic, production rules, and operator-guided validation procedures.

The workflow enables automatic correction of most inconsistencies in line and polygon features located along sheet borders. Generated control points allow the redirection of manual work from complete detail visual inspection of the sheet boundary to targeted verification of remaining problematic cases. In this way, automation did not remove cartographic judgement from the process. Instead, it reduced repetitive editing and focused operator attention on situations where expert control is necessary. This resulted in a substantial reduction in processing time and an efficiency increase of more than seven times.

The developed workflow establishes a methodological and practical knowledge base for applying Python programming to the automation of other processes within the production of geotopographic materials. The same logic can be adapted to other cartographic tasks, such as generalization. The results also indicate that semi-automation is an appropriate model for complex cartographic workflows. Full automation is limited by the variability of mapped content and the need for cartographic interpretation, but a carefully designed semi-automated workflow can significantly reduce manual work, improve consistency, and support a more sustainable digital topographic map production system.

References

[1] Eker, R. and Aydin, A. (2016). Customization and Automation of Geoprocessing with Python in ArcGIS for Natural Hazards and Forestry-related Applications. In Proceedings of International Forestry Symposium (IFS 2016). 07-10 December Kastamonu.

[2] Diener, M. (2015). Python Geospatial Analysis Cookbook. Birmingham: Packt Publishing Ltd. ISBN: 9781783555079

[3] Garrard, C. (2016). Geoprocessing with Python. New York: Manning Publications Co. ISBN: 9781617292149

[4] Drobnjak, S., Milonjić, Z., Đorđević, D., & Simić, M. (2025). Flash flood susceptibility mapping using LiDAR-Derived DEM and machine learning algorithms: Ljubovića case study, Serbia. Open Geosciences, 17(1). https://doi.org/10.1515/geo-2025-0859

[5] Simić, M., Vakanjac, B., & Drobnjak, S. (2025). NDGRI: A Novel Sentinel-2 Normalized Difference Gamma-Radiation Index for Pixel-Level Detection of Elevated Gamma Radiation. Remote Sensing, 17(19), 3331. https://doi.org/10.3390/rs17193331

[6] Simić, M., Vakanjac, B., Drobnjak, S., Novković, I. & Jovanović, J. (2026). Deep learning classification of elevated gamma-radiation zones using Sentinel-2 spectral data. Open Geosciences, 18(1). https://doi.org/10.1515/geo-2025-1000

[7] Zandbergen, A., P. (2020). Advanced Python Scripting for ArcGIS Pro. San Bernardino: Esri Press. ISBN: 9781589486188.

[8] Pimpler, E. (2021). Programming ArcGIS Pro with Python - 2ndEdition. Boerne: Geospatial Training Services. ISBN: 9798740349961

[9] Simić, M., Bobar, M., Bakrač, S., & Jovanović, J. M. (2025, November 20). PYTHON-BASED

AUTOMATION OF GEOPROCESSING IN DIGITAL TOPOGRAPHIC MAP PRODUCTION: CONTENT ALIGNMENT BETWEEN MAP SHEET BORDERS. 52nd Symposium on Operational Research (SYMOPIS 2025), Palić, Serbia. https://doi.org/10.5281/zenodo.17533755

[10] Dobesova, Z. (2011): Programming Language Python for Data Processing. Proceedings of International Conference on Electrical and Control Engineering (ICECE), Yichang, China, Volume 6 Institute of Electrical and Electronic Engineers (IEEE), CFP 1173J-PRT, 4866-4869 p. ISBN 978-1-4244-8163-7

[11] Toms, S. (2015). ArcPy and ArcGIS-Geospatial Analysis with Python (Vol. 1, No. 1, pp. 1-343). Birmingham, UK: Packt Publishing.

[12] Gowrishankar, S., and Veena, A. (2018). Introduction to Python Programming. Miami: CRC Press. ISBN: 9780815394372