b) *Automated dependency analysis:*
utilizing tools for dependency analysis, such as Dependency Structure Matrix (DSM) or visualization tools, allows developers to better understand the relationships between project components. This enhances the perception of the project structure and facilitates decision-making regarding modifications [1].
c) *Standardization and documentation:*
implementing strict coding and dependency documentation standards helps to establish a unified approach within the development team. Documenting dependencies in clear and understandable formats makes the project more accessible to the new team members and streamlines the maintenance process [2].
d) *Application of design patterns and architectural solutions:*
utilizing design patterns, such as Dependency Injection, contributes to creating the loosely coupled components, making the project more flexible and ensuring an easiness of making changes. Applying the architectural principles like SOLID also aids in dependency management and facilitates an easiness of maintenance [4].

REFERENCES

1. Eric Evans. (2003). Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley.
2. Martin Fowler. (2014). Dependency Injection. [Electronic resource].– Access mode: https://martinfowler.com/articles/injection.html
3. Robert C. Martin. (2002). Principles of Object-Oriented Design. [Electronic resource].– Access mode: https://web.archive.org/web/20140213162847/http://butunclebob.com/ArticleSUncleBob.PrinciplesOfOod
4. Steve McConnell. (2004). Code Complete: A Practical Handbook of Software Construction. MicrosoftPress.

N. Nomerchuk, T. Prokofiev, O. Hurko

OPTIMIZATION OF EXPONENTIAL FUNCTION COMPUTATION IN JAVA USING LINEAR INTERPOLATION METHOD

Exponential functions used by standard mathematical libraries have high precision but concurrently demonstrate considerable computational overhead. To optimize performance across various scenarios, such as neural network implementations or time series forecasting, leveraging exponential function approximations can be warranted, thus mitigating computational burdens.

For instance, to address a specific task of decomposing spectra into individual bands [1], a mathematical model for approximating the experimental spectrum into its constituent components was utilized and implemented as a Java application. However, the execution time of one computation was approximately 0.067 milliseconds, while the overall program execution time amounted to 275.4 seconds, which did not meet the requirements. Upon analyzing the application, it was found that the primary cause of slowdown was the utilization of the standard java.Math.exp() function, which is implemented as a native method and consequently invoked via the Java Native Interface (JNI). The process of invoking a native method involves suspending the execution thread of the Java Virtual Machine (JVM), transferring control to the native code of the operating system, and returning control to the suspended thread, which is quite time- and resource-intensive.

Therefore, as an alternative, the possibility of creating a custom *exp()* function utilizing linear interpolation was considered. To achieve this, it was necessary to take into account the representation of floating-point numbers. They are represented by the formula [2]:

$$k = (-1)^s \times (1 + f) \times 2^{x - x_0} \tag{1}$$

where,s – a binary variable that determines the sign of the number and can take on the values 0 or 1 (sign), f–a mantissa (fraction) – a binary fraction in the range [0,1), and x – the exponent shifted by a constant x_0. In the JVM, according to the IEEE-754 standard, floating-point numbers are defined with a 52-bit mantissa and an 11-bit exponent with bias x_0=1023, occupying 8 bytes (Fig. 1). The components of this representation can be manipulated by accessing memory bits as a pair of 4-byte *integers*. Any integer written to the x component (via the higher-order 4 bytes) will be exponentiated when the bytes are read back in floating-point format. This is a key concept for fast exponentiation.

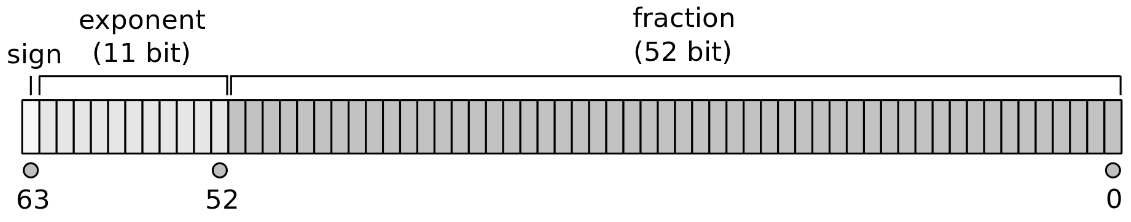

Figure 1. Bit representation of double-precision numbers according to the IEEE-754 standard

Since the component x resides in the 4 most significant bits of the number, the integer k to be exponentiated must be left-shifted by 20 bits after the addition of the x_0 offset. Thus, $i = 2^{20}(k+1023)$ calculates 2^k for integer k. For non-integer arguments, the fractional part of k will transition to the higher-order bits of the mantissa f, according to the IEEE-754 specification, representing linear interpolation between adjacent integer exponents. Therefore, this method effectively exponentiates real numbers by searching in a table of 2^{11} values and performing linear interpolation between them.

To compute e^y, one needs to divide i by $ln(2)$. The complete transformation of y, required for rapid approximation in IEEE-754 format, is defined by the expression:

$$i = ay + b \tag{2}$$

where, $a = 2^{20}/ln(2)$, $b = 1023 \cdot 2^{20}$. In the Java environment, expression (2) will take the following form:

```
public static double exp(double val) {
   final long tmp = (long) (1512775 * val + 1072693248);
   return Double.longBitsToDouble(tmp << 32);
}
```

The *long* data type is used to avoid overflow. Furthermore, by performing a bitwise shift by 32, the approximated value of the function e^y is returned.

A comparative performance analysis of the standard and custom functions for $e^{(y_i)}$, $i=10^8$is provided in Table 1.

Table 1. Comparison of functions

Function	Calculation time, ms	Maximum deviation	Minimal deviation
java.Math.exp	9244	0%	0%
Approximated exp	1864	3.02%	0.07%

The conducted study demonstrated that the application of linear interpolation for functions of the form e^y increases speed and provides a significant improvement in work efficiency by reducing the computation time of the exponential function compared to the standard java.math library by approximately 5 times. The developed Java method has a small deviation (approximately 3%) from the exact values of the exponential function, which does not significantly affect the overall quality of the final result.

REFERENCES

1. Спосіб аналізу експериментальних спектрів люмінесценції монокристалічних матеріалів: пат. 126608 Україна: G01N 21/62, G06F17/17. №а 2020 07016; заявл. 02.11.2020; опубл. 04.05.2022, Бюл. №18. 6с.
2. IEEE. (1985). Standard for binary foating-point arithmetic. ANSI/IEEE Std. 754–1985. New York: American National Standards Institute/Institute of Electrical and Electronic Engineers.

S. Orlov, T. Nakonechna, Yu. Honcharova

MODERN METHODS OF ADDITIVE MANUFACTURING OF THREE-DIMENSIONAL OBJECTS BASED ON FUGO TYPE PRINTER

Modern methods of additive manufacturing of three-dimensional objects (for example, stereolithography, 3D printing, etc.) allow the production of high-quality products with high precision, but such methods have significant limitations and drawbacks.

At the moment, there is a need for a system and method to ensure 3D printing with improved part quality and increased printing speed. A team of developers from the USA has patented [1] the Fugo 3D printer based on a centrifuge, which can significantly accelerate the printing of complex parts, their quality, and application conditions. Production based on a centrifuge does not require the influence of gravity, which is an invaluable breakthrough and a bright ray of light for the future of humanity, as it will allow printing any complex parts and tools in space, which until now has not been possible for liquid polymer printing technologies. In addition to eliminating the drawbacks of existing 3D printers and approaches to manufacturing super complex objects, this enables people to simplify printing from metal and other super hard materials greatly. Like other printing systems, this printer requires high-quality and fast software to convert user models into printer instructions. Developing such software for an innovative cylindrical printer is an extremely important task as it directly affects the quality and speed of printing models, which is necessary for many users and entire enterprises.

The basis for future research is the transformation of data about a 3D model into a sequential set of layers for printing on a cylindrical printer, which is interpreted as a set of instructions for the Fugo type printer that are executed using internal