ISSN 2076-5886

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Черкаський національний університет
імені Богдана Хмельницького

# ВІСНИК ЧЕРКАСЬКОГО УНІВЕРСИТЕТУ

## Серія ПРИКЛАДНА МАТЕМАТИКА. ІНФОРМАТИКА

Виходить 2 рази на рік

Заснований у березні 1997 року

№ 1-2 (2018)

**ЧЕРКАСИ – 2018**

**Засновник, редакція, видавець і виготовлювач –**
**Черкаський національний університет імені Богдана Хмельницького**
Свідоцтво про державну перереєстрацію КВ № 21396-11196Р від 26.06.2015

Матеріали «Вісника» присвячені прикладним і фундаментальним науковим дослідженням у галузі інформаційних технологій, моделювання систем, прикладних інформаційних систем у різних сферах діяльності, математичному моделюванню фізичних процесів, математичної фізики, інформаційних і обчислювальних технологій. У публікаціях відображаються різні аспекти досліджень учених і фахівців у сфері обчислювальних мереж і систем, елементів і пристроїв обчислювальної техніки та систем управління, програмного забезпечення,

Наукові статті збірника рекомендовані аспірантам, студентам, викладачам вищої школи, а також усім, хто цікавиться проблематикою прикладної математики та інформатики.

Журнал входить до переліку наукових фахових видань України з фізико-математичних, технічних наук (Наказ МОН України від 12 травня 2015 р. № 528).

Випуск №1-2 наукового журналу Вісник Черкаського університету, серія: «Прикладна математика. Інформатика» рекомендовано до друку та до поширення через мережу Інтернет Вченою радою Черкаського національного університету імені Богдана Хмельницького (протокол №5 від 16 грудня 2018 року).

*Журнал реферується Українським реферативним журналом "Джерело" (засновники: Інститут проблем реєстрації інформації НАН України та Національна бібліотека України імені В. І. Вернадського).*

**Головна редакційна колегія:**

*Черевко О.В.*, д.е.н., проф. (головний редактор); *Боєчко Ф.Ф.*, член-кор. НАПН України, д.б.н., проф. (заступник головного редактора); *Корновенко С.В.*, д.і.н., проф. (заступник головного редактора); *Кирилюк Є.М.*, д.е.н., проф. (відповідальний секретар); *Архипова С.П.*, д.пед.н., проф.; Біда О.А., д.пед.н., проф.; *Гнезділова К.М.*, д.пед.н., проф.; *Головня Б.П.*, д.т.н., доц.; *Гусак А.М.*, д.ф.-м.н., проф.; *Десятов Т.М.*, д.пед.н., проф.; *Земзюліна Н.І.*, д.і.н., проф.; *Жаботинська С.А.*, д.філол.н., проф.; *Кузьмінський А.І.*, член-кор. НАПН України, д.пед.н., проф.; *Кукурудза І.І.*, д.е.н., проф.; *Лизогуб В.С.*, д.б.н., проф.; *Ляшенко Ю.О.*, д.ф.-м.н., доц.; *Марченко О.В.*, д.філос.н., проф.; *Масненко В.В.*, д.і.н., проф.; *Мінаєв Б.П.*, д.х.н., проф.; *Морозов А.Г.,* д.і.н., проф.; *Перехрест О.Г.*, д.і.н., проф.; *Поліщук В.Т.*, д.філол.н., проф.; *Селіванова О.О.*, д.філол.н., проф.; *Чабан А.Ю.*, д.і.н., проф.; *Шпак В.П.*, д.пед.н., проф.

**Редакційна колегія серії:**

*Головня Б.П.,* д.т.н., проф. (відповідальний редактор); *Гришко Л.В.***,** к.пед.н., доц. (відповідальний секретар); *Авраменко В.С.***,** к. фіз.-мат. н., доц.; *Богатирьов О.О.,* к. фіз.-мат. н., доц.; *Гаєв Є.О.*, д.т.н., проф.; *Гусак А.М.*, д.ф.-м.н., проф.; *Єршов С.В.*, д.т.н., проф.; *Заурбеков Н.С.*, д.т.н., проф. (Казахстан); *Златкін А.А.*, д.т.н., проф.; *Круковський П.Г.*, д.т.н., проф.; *Кузьмук В.В.*, д.т.н., проф.; *Лагно В.І.*, д.ф.-м.н., *Лещенко Ю.Ю.,* к. фіз.-мат. н., доц.; *Марек Данилевский,* д-р габілітований, проф. (Польща); *Мовчан В.Т.*, д.ф.-м.н., проф.; *Онищенко Б.О.,* к. фіз.-мат. н., доц.; *Приходько О.А.*, д.ф.-м.н., проф.; *Соловйов В.М.,* д.ф.-м.н., проф.; *Стеблянко П.О.*, д.ф.-м.н., проф.; *Супруненко О.О.,* к.т.н., доц.; *Тесля Ю.М.,* д.т.н., проф.; Хенрик Лещинский, д-р габілітований, проф. (Польща); *Шрайбер О.А.*, д.т.н., проф.

*За дотримання права інтелектуальної власності, достовірність матеріалів та обгрунтування висновків відповідають автори*

**Адреса редакційної колегії:**
18000, Черкаси, бульвар Шевченка, 81,
Черкаський національний університет ім. Б. Хмельницького,
кафедра прикладної математики та інформатики. Тел. (0472) 36-13-55
web-сайт: http://ami-ejournal.cdu.edu.ua/index.
e-mail: (herald_pm@ukr.net)

© Черкаський національний
університет, 2018

# СЕКЦІЯ «ПРИКЛАДНА МАТЕМАТИКА»

UDK 532.517.4

PACS 02.60.-x, 02.60.Pn, 02.70.Wz

**GOLOVNYA Boris P.**
Department of Informatics and Applied Mathematics, The Bohdan Khmelnytsky National University of Cherkasy, Ukraine
e-mail: BPGolovnya@gmail.com

## ON CASCADE ENERGY TRANSFER IN TURBULENCE MODELING

*Cascade energy transfer plays a very important role in all turbulent flows. Unfortunately, this process is very difficult to model. This paper analyzes one of the possible causes of difficulties. It is shown that the turbulence model proposed earlier by the author successfully copes with these difficulties.*

***Key words:*** *cascade transfer modeling, k-ε turbulence model.*

### 1. The importance of cascade transfer

Almost all modern models of turbulence are based on the equation of turbulent energy transfer. If such model is reckoned as universal then it must answers three questions. First – where does the turbulent energy originate? Second – what happens to this energy? Third – how does the energy dissipate? Otherwise there are certainly exists such flow that this model can not reproduce.

For the vast majority of turbulent flows Kolmogorov's theory of cascade energy transfer answers these questions. By this reason results of simulations by high-quality models must be in agreement with this theory.

In this work, we will try to figure out which properties allow some models to simulate cascade energy transfer.

### 2. The Jones-Launder model

The Jones-Launder model [1] has the following form

$$\begin{cases} \dfrac{Dk}{Dt} = \dfrac{\partial}{\partial x_k}\left(\nu + \dfrac{\nu_t}{C_k}\right)\dfrac{\partial k}{\partial x_k} + P - \varepsilon, \\ \dfrac{D\varepsilon}{Dt} = \dfrac{\partial}{\partial x_k}\left(\nu + \dfrac{\nu_t}{C_\varepsilon}\right)\dfrac{\partial \varepsilon}{\partial x_k} + \dfrac{\varepsilon}{k}\left(C_1 P - C_2 \varepsilon\right). \end{cases} \tag{1}$$

It is stated in [2] that the equations system

$$\begin{cases} \dfrac{dk}{dt} = -\varepsilon, \\ \dfrac{d\varepsilon_e}{dt} = -C_2 \dfrac{\varepsilon^2}{k}. \end{cases} \tag{2}$$

reproduces well the grid turbulence decay if $C_2 = 1.4$. Note that it was shown in [3] that the requirement $C_2 < 1.5$ in (2) follows from the condition $\lim_{t\to\infty} L_e = 0$. Based on the coincidence of the dissipative terms of systems (1) and (2), we assume that system (2) is a

simplified Jones-Launder model (1). Because the grid turbulence decay is very similar to a cascade process, it was decided to test this model for the possibility of the cascade process simulating.

The model for the cascade process simulation is based on the following hypothesis. It is supposed that the dissipative term describes not the rate of the transition of the kinetic energy to the thermal energy, but the rate of energy transfer into the cascade process. On the base of this supposition we calculated several steps of a discrete cascade process. Each step of the process was calculated on a separate model. Therefore, the complete model of process consists of several similar systems of equations. The turbulence model as such describes the first step of the cascade process. Its dissipative term describes the transfer of energy to the second step of the cascade. Therefore, it is used as the production term in the second step. The dissipation of the second step is used as the production in the third step, and so on.

One must say that a similar model, although for other purposes, was proposed in the [4]. But any real using of this model was not found in the literature.

### 3. Governing equations

At the first step of the calculation, system (1) is solved. The equations for modeling of the next step of a discrete cascade process looks as follows. Here $i$ is the number of the cascade process step.

$$\begin{cases} P^i = \varepsilon^{i-1} \\ \dfrac{Dk^i}{Dt} = \dfrac{\partial}{\partial x_k}\left(\nu + \dfrac{\nu_t^i}{C_k}\right)\dfrac{\partial k^i}{\partial x_k} + P^i - \varepsilon^i, \\ \dfrac{D\varepsilon^i}{Dt} = \dfrac{\partial}{\partial x_k}\left(\nu + \dfrac{\nu_t^i}{C_\varepsilon}\right)\dfrac{\partial \varepsilon^i}{\partial x_k} + \dfrac{\varepsilon^i}{k^i}\left(C_1 P^i - C_2 \varepsilon^i\right). \end{cases} \quad (3)$$

The initial conditions of the whole simulation and are the values of energy and dissipation $k^0$ and $\varepsilon^0$ obtained from solution (1).

The initial conditions for calculating the next step of the cascade process are set as

$$k_0^j = \alpha k_0^{j-1},\ \varepsilon_0^j = \varepsilon_0^{j-1} = \varepsilon_0^0. \quad (4)$$

### 4. Solution requirements

1. The turbulent energy spectrum must satisfy the relation $E(\kappa) \sim \kappa^{-5/3}$. Here $E$ is the energy function, $\kappa$ is the wave number. In this paper, the values of the discrete energy function were found as $E_i(\kappa) = k_i/\kappa_i$, the values of the discrete wave number - $\kappa_i = 2\pi/L_i$. Here $k_i$ is the turbulent energy of the $i$-th step of the cascade process, $L_i$ is the value of the dissipative scale at this step.
2. The transition of the energy of turbulence to heat occurs at small scales of the vortices. It follows that the energy transmitted through the cascade per time unit, i.e. the rate of dissipation, is unchanged.
3. The values of turbulent energy in any cross section of a turbulent vortex decrease during the cascade process. Reason. In the cascade process vortices stretch. So their radiuses decrease. Decrease of vortices radiuses leads to decrease of linear velocities of vortices. As a result the levels of fluctuations decrease also.

4. The diameters of the vortices decrease in the process of cascade transfer. Hence, if we assume that the dissipative scale is a characteristic of the diameter of the vortices, then it must also decrease.

The following must be said. These are fairly general requirements for modeling a cascade process. But, if the model poorly reproduces the turbulence energy profile, turbulent viscosity, etc., then it is impossible to require that the solutions of the equations of the model satisfy all these requirements. Therefore, here we check that the results match only the first item on the list. In this regard, we mention that (see, for example [5]) the flow in round jet cannot be predicted with the standard k-e and therefore with Jones-Launder model.

**5. Results of simulations**

It is known that the Jones-Launder model poorly calculates near-wall turbulence. Therefore, we will check the capabilities of the model by calculating a cascade process in free flows - a round jet and a flat shear layer. In these cases, both the turbulence itself and the cascade process were simulated using the Jones-Launder model.

Figures 1 and 2 show the results of modeling the cascade transfer in a round jet and a plane shear layer.

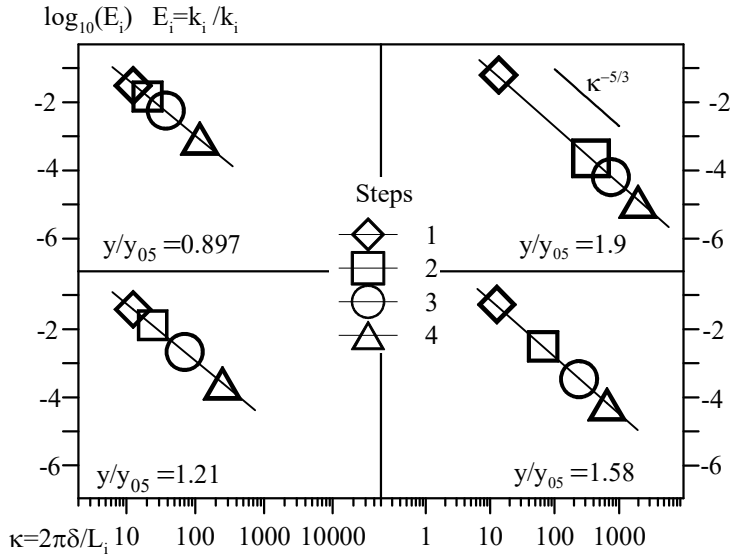



Fig.1. Calculation of 4 steps of cascade transfer in 4 cross serction of round jet .

The possibilities of calculating the near-wall flows were verified by simulations of the cascade process in the boundary layer. In this case, the flow in the boundary layer was calculated by the author's model [3], and the cascade transfere by the Jones-Launder model. The results are shown in Fig. 3.

The calculations of the correspondence of the spectrum to the "-5/3" law for several well-known models are shown in Figure 4.

Chen's model [6] shows that in the third step of the cascade process the wave numbers of the fluctuations decrease, i.e. the diameters of vortices increase. This is impossible. The Menter's SST model [7] does not demonstrate any orderliness of the results by the wave numbers in the steps of the process. Launder-Sharma [8] model in the third step shows scales of turbulence on the order of $\left(10^{-8} \div 10^{-9}\right)\delta$. This is why the third and fourth steps are not shown on the plot. But such scales are much smaller than Kolmogorov's scale.

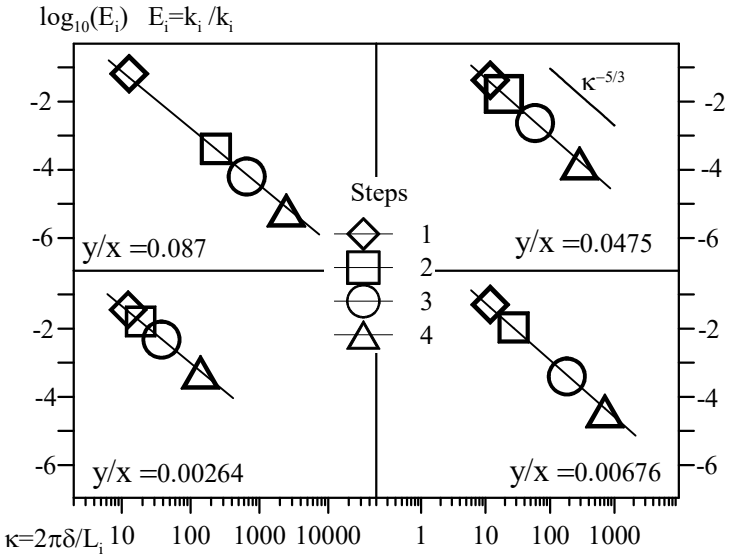



Fig.2. Calculation of 4 steps of cascade transfer in 4 cross serction of plane mixing layer .

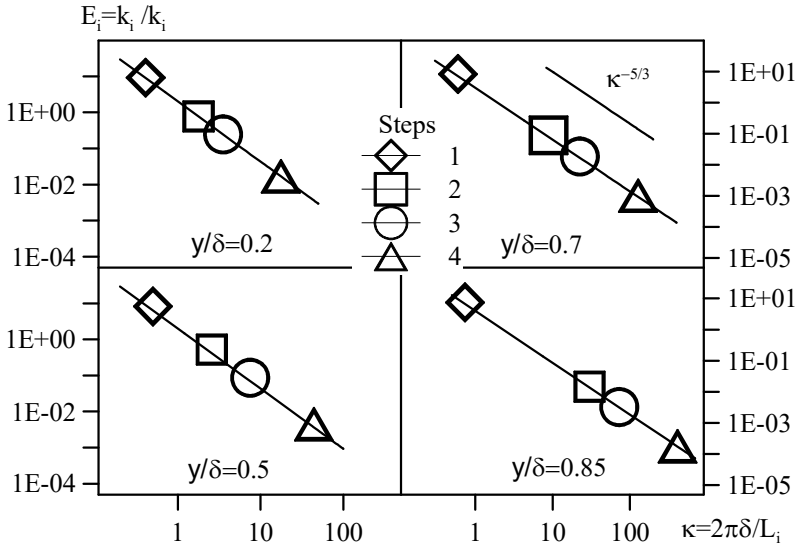



Fig.3. Calculation of 4 steps of cascade transfer in 4 cross serction of boundary layer.

Note that the Wilcox's model [8] correctly shows changes of the energy function in the cascade process steps.

Consider what this model has in common with the Jones-Launder model.

We can say that almost all k-ε models are the Jones-Launder model with corrections. A typical k-ε model can be written as

$$\begin{cases} \dfrac{Dk}{D\tau} = \dfrac{\partial}{\partial x_i}\left(\nu + \dfrac{\nu_t}{\sigma_k}\right)\dfrac{\partial k}{\partial x_i} + (P - \varepsilon) - E_k, \\ \dfrac{D\varepsilon}{D\tau} = \dfrac{\partial}{\partial x_i}\left(\nu + \dfrac{\nu_t}{\sigma_\varepsilon}\right)\dfrac{\partial \varepsilon}{\partial x_i} + \dfrac{\varepsilon}{k}\left(C_{\varepsilon 1} f_1 P - C_{\varepsilon 2} f_2 \varepsilon\right) + E_\varepsilon, \\ P = \tau_{ij}\left(\dfrac{\partial U_i}{\partial x_j}\right). \end{cases}$$

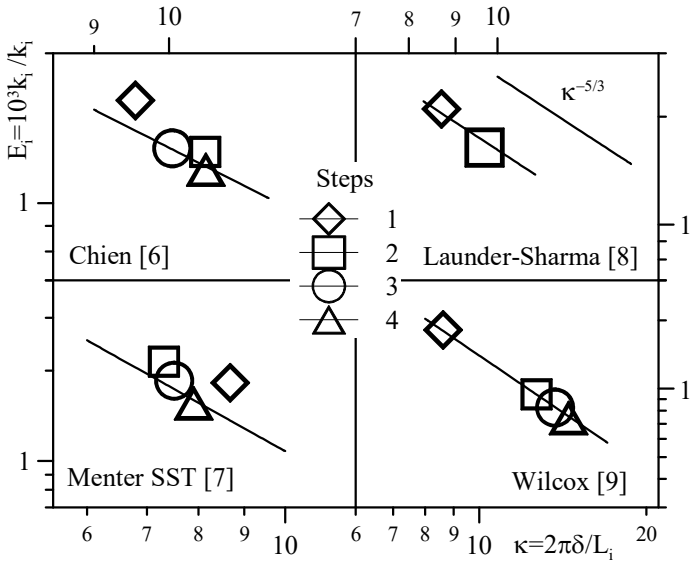



Fig.4. Calculation of 4 steps of cascade transfer in boundary layer by 4 models of turbulence .

Here $P = \tau_{ij}\left(\partial U_i / \partial x_j\right)$ is the rate of energy transfer from the mean flow to turbulence or the production of turbulence energy.

The corrections $E_k$ and $E_\varepsilon$ in fact take into account the near-wall effect of viscosity on diffusion and are not considered further. In most models, the correction $f_1 = 1$. The model is mainly determined by the $f_2$ correction. Note that at a distance from the wall $f_1 = f_2 = 1$, $E_k = E_\varepsilon = 0$, that is, the model coincides with the Jones-Launder model.

Wilcox's model can also be classified as a Jones-Launder model, but with a different dissipative variable.

Only in the Jones-Launder and Wilcox models, the correction $f_2 = 1$ in the whole calculation area. In the author's opinion, it is this fact that allows these models to reproduce the "-5/3" law. Unfortunately, the author failed to prove this mathematically.

Why are these corrections needed?

The meaning of the corrections introducing is obvious. It is noted that in the near-wall region the Jones-Launder model gives strongly overestimated values of the turbulence energy. Therefore, in order to decrease the energy near the wall, it is necessary to decrease the term in the energy transfer equation in this region. But all the terms of this equation, except for the dissipation rate, are exact and their profiles are verified by experiments. Therefore, one cannot change them. Therefore, in all models, corrections $f_1$ and $f_2$ are introduced into the dissipation rate transfer equation, so that the dissipation rate at the wall increases. For this, $f_2$ at the wall should decrease (as a result, the dissipative term of the dissipation rate transfer equation in the wall region will decrease), and $f_1$ at the wall should increase, which will increase ε in this region. It is clear that these corrections in no way follow from the Navier-Stokes equations.

As a result, we have a contradiction that cannot be resolved within the framework of the traditional approach. To reproduce the "-5/3" pattern the corresponding terms of the model equations must have the form $\left(P - \varepsilon\right)$ and $\left(C_1 P - C_2 \varepsilon\right)$, but in order to reproduce the near-wall turbulence, it is necessary to introduce corrections into the dissipation transfer equation.

In [3], the author succeeded in resolving this contradiction. The proposed model has the form

$$\begin{cases} \dfrac{Dk_0}{Dt} = f_0 \dfrac{\partial}{\partial x_k}\left(\nu + \dfrac{\nu_{t0}}{C_k}\right)\dfrac{\partial k_0}{\partial x_k} + \left(f_0 P - \varepsilon_0\right), \\ \dfrac{D\varepsilon_0}{Dt} = f_0 \dfrac{\partial}{\partial x_k}\left(\nu + \dfrac{\nu_{t0}}{C_\varepsilon}\right)\dfrac{\partial \varepsilon_0}{\partial x_k} + \dfrac{\varepsilon_0}{k_0}\left(C_1 f_0 P - C_2 \varepsilon_0\right). \end{cases}$$

It has already been said that in order to decrease the energy at the wall, the term $(P-\varepsilon)$ must be reduced. In the traditional approach, this is done by increasing ε in the near-wall region. But at the same time, the capabilities of the model are noticeably reduced. At the same time, it is also possible to decrease $(P-\varepsilon)$ while not changing $(C_1P - C_2\varepsilon)$ by decreasing $P$, which was done in []. Here $f_0$ is a correction function equal to zero on the wall and 1 in free flow. This approach is in good agreement with a variety of experimental data. Many calculations carried out using this model are not available to modern models in principle.

## 6. Conclusions

The paper analyzes the reasons of difficulties in modeling the cascade process of turbulent energy transfer. According to the simulation results, no one modern model can correctly reproduce the cascade transfer. It is shown that the reasons that do not allow modeling the cascade transfer lie in the base of models designing method. It is argued that proposed by the author the new methodology for models designing effectively copes with these difficulties.

**References**


1. Jones W. P., Launder B. E.. The prediction of laminarization with a two-equation model of turbulence/ W. P. Jones and B. E. Launder// Int. J. Heat Mass Transfer. - 1972 - V. 15. - 301-314.
2. Speziale C., Bernard P.. The energy decay in self-preserving isotropic turbulence revisited/ C. Speziale, P. Bernard // J. Fluid Mech. - 1992 - V. 241. - 645-667.
3. Golovnya, B.P. Important properties of turbulent near-wall flows which are not accounted by modern RANS models / B.P. Golovnya // Int. J. Heat Mass Transfer. – 2020. – V. 146. – 118813.
4. Schiestel R. Multiple-time-scale modeling of turbulent flows in one-point closures / R. Schiestel R. // Phus. of Fluids. - 1987 - V. 30 - 722
5. McGuirk J. J..W. Rodi W. The Calculation of Three-Dimensional Turbulent Free Jets. / J. J. McGuirk, W. Rodi // Turbulent-Shear-Flows-1- Pennsylvania, USA-1977
6. Chien K.-Y. Prediction of channel and boundary-layer flows with a low Reynolds number turbulence model / K.-Y. Chien // AIAA J.-1982 - V. 20 - 34–38.
7. Menter F.R. Zonal two-equation k-ω turbulence model for aerodynamic flows./ F.R. Menter // AIAA Paper. - 1993 - V. 93 - 2906
8. Launder B.E., Sharma B.I. Application of the energy-dissipation model of turbulence to the calculation of flow near a spinning disc. / B.E. Launder, B.I. Sharma, // Lett. Heat Mass Transfer - 1974 - V.1 - 131–138.
9. D.C. Wilcox, Turbulence Modelling for CFD. / D.C. Wilcox // DCW Industries, Clfrn. USA - 1994.


**ГОЛОВНЯ Борис Петрович,**

**ДО ПИТАННЯ ПРО КАСКАДНЕ ПЕРЕНЕСЕННЯ ЕНЕРГІЇ В ТУРБУЛЕНТНИХ ТЕЧІЯХ**

Каскадне перенесення енергії відіграє найважливішу роль у всіх турбулентних течіях. На жаль цей процес дуже погано піддається моделюванню. У даній роботі аналізується одна з можливих причин виникнення труднощів. Показується, що модель турбулентності, запропонована раніше автором, з цими труднощами успішно справляється

***Ключові слова:*** моделювання каскадного перенесення, $k-\varepsilon$ модель турбулентності.

*Одержано редакцією 11.07.2018*
*Прийнято до публікації 20.09.2018*

УДК 519.6:004.8

PACS 02.60.-x, 02.60.Pn, 02.70.Wz

**ГОЛОВНЯ Борис Петрович**
доктор технических наук, доцент кафедры прикладной математики и информатики Черкасского национального университета им. Б. Хмельницкого
e-mail: bpgolovnya@gmail.com

# МОДЕЛЬ ДЛЯ РАСЧЕТА СВОБОДНЫХ СДВИГОВЫХ ТУРБУЛЕНТНЫХ ТЕЧЕНИЙ

*Ранее автором была предложена модель для расчета сдвиговых турбулентных течений около твердых поверхностей. Важнейшая идея, лежащая в основе предложенных для этих течений моделей, базируется на том, что если вихрь соприкасается с твердой поверхностью и/или подвергается воздействию сдвига, то в течении возникают дополнительные вихри. В свободных течения отсутствует поверхность, но контакт со сдвигом остается. Поэтому предложенный подход применим и к построению моделей для расчета свободных сдвиговых течений. В результате была разработана модель для расчета сдвиговых свободных турбулентных течений. Показано, что предложенная модель удовлетворительно воспроизводит базовые свойства основных свободных турбулентных течений а именно - струи, сдвигового слоя и турбулентного следа. Когерентные структуры, получаемые в расетах соответствуют экспериментальным данным разных авторов.*

***Ключевые слова:*** *моделирование турбулентности, энергия турбулентности*

**Вступление**

Все рассмотренные выше модели относились к случаю течений около твердых поверхностей. Важнейшая идея, лежащая в основе предложенных для этих течений моделей, базируется на том, что если ламинарный вихрь соприкасается с твердой поверхностью и/или подвергается воздействию сдвига, то в течении возникают дополнительные вихри. В свободных течения отсутствует поверхность, но контакт со сдвигом остается. Поэтому предложенный подход применим и к построению моделей для расчета свободных сдвиговых течений.

Как известно, существуют три основных типа свободных сдвиговых течений – слой смешения, след и струя. В плоском случае уравнения, описывающие рассматриваемые течения, записываются следующим образом (см. Себиси, Брэдшоу [2])

$$U\frac{\partial U}{\partial x}+V\frac{\partial U}{\partial y}=\frac{\partial}{\partial y}\left(\nu+\nu_t\right)\frac{\partial U}{\partial y},$$
$$\frac{\partial U}{\partial x}+\frac{\partial V}{\partial y}=0$$

Краевые условия:

1 – плоская турбулентная струя -

$$y=0,\ \ V=0,\ \ \frac{\partial U}{\partial y}=0;\quad y=\infty,\ \ V=0,\quad U=0;$$

2 – плоский турбулентный след -

$$y=0,\ \ V=0,\ \ \frac{\partial U}{\partial y}=0;\quad y=\infty,\ \ V=0,\quad U=0;$$

3 – плоский турбулентный слой смешения -

$$y = -\infty,\ \ V = 0,\ \ U = U_1; \quad y = \infty,\ \ V = 0,\ \ U = U_2.$$

Из представленных уравнений и краевых условий следует что, уравнения для расчета турбулентных струй и следов полностью идентичны. По этой причине в данной работе рассматриваются только два типа течений, а именно – турбулентный слой смешения и турбулентная струя.

**1. Построение модели турбулентности для расчета свободных сдвиговых течений**

Очевидно, что в связи с изменением физики процесса все предваряющие построение модели выкладки должны быть проделаны с самого начала.

Построение модели фактически распадается на три этапа. Во-первых, необходимо определить вид уравнений модели. Во-вторых, используя какую-либо апробированную модель в качестве генератора данных, рассчитать предположительный вид функции $f_0$. В третьих, на основе предположительного вида функции найти для нее аналитическое представление.

Исходный вид уравнений модели будем строить по аналогии с модельными уравнениями для течений в пограничном слое. При построении вида модели отмечаем, что в уравнениях должны отсутствовать члены, описывающие пристенные вязкие взаимодействия. В уравнениях модели для течения в пограничном слое их наличие учитывается умножением вязких диффузионных слагаемых на функцию подавления. Таким образом, уравнения модифицированной $k$-$\varepsilon$ системы уравнений имеют вид (см. [1])

$$\frac{Dk_0}{Dt} = \nu \frac{\partial^2 k_0}{\partial y^2} + f_0 \frac{\partial}{\partial y} \nu_{t0} \frac{\partial k_0}{\partial y} + f_0 \nu_{t0} \left( \frac{\partial U}{\partial y} \right)^2 - \varepsilon_0, \tag{1}$$

$$\frac{D\varepsilon_0}{Dt} = \nu \frac{\partial^2 \varepsilon_0}{\partial y^2} + f_0 \frac{\partial}{\partial y} \frac{\nu_{t0}}{C_\varepsilon} \frac{\partial \varepsilon_0}{\partial y} + \frac{\varepsilon_0}{k_0} \left( C_1 f_0 \nu_{t0} \left( \frac{\partial U}{\partial y} \right)^2 - C_2 \varepsilon_0 \right). \tag{2}$$

Следующий шаг – определение вида функции $f_0$. При проведении расчетов предполагалось, что, так как и энергия турбулентности и турбулентная вязкость являются измеряемыми параметрами, то все хорошо апробированные модели обязаны рассчитывать их с достаточно высокой точностью. Целью расчета было достижение совпадения распределения энергии турбулентности и турбулентной вязкости, получаемых по модели (1)-(2), с распределениями, получаемыми по хорошо апробированным моделям. Для этого сначала решаются уравнения апробированной модели. Предполагаем, что из решения находятся достаточно точные значения турбулентной энергии и турбулентной вязкости. Подставляем эти рассчитанные значения турбулентной энергии и турбулентной вязкости в систему (1)-(2). После подстановки система (1)-(2) содержит только две неизвестных функции, а именно функцию $f_0$ и скорость диссипации. Эти неизвестные находим, решая систему (1)-(2). В данной работе в качестве генератора данных использовалась модель Джонса-Лаундера [3].

Результаты (см. [4]) приведены на рис. 1 и 2.

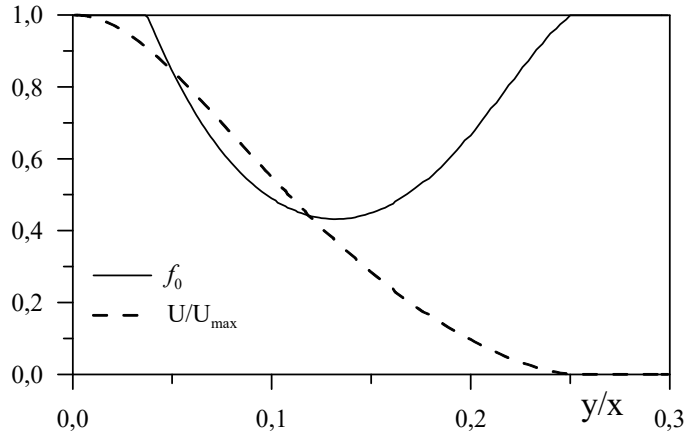



Рис. 1. Расчет функции $f_0$ в круглой струе.

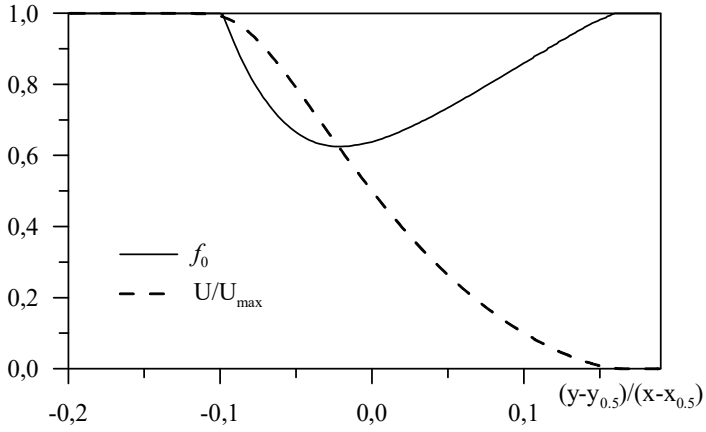



Рис. 2. Расчет функции $f_0$ в слое смешения.

Из графиков 1 и 2 можно предположить, что в свободных сдвиговых течениях основная причина возникновения подавления состоит в растяжении вихря вдоль течения, которое вызывается сдвигом среднего потока. После проведения вариантных расчетов для функции подавления была выбрана следующая форма записи

$$f_0(y) = \left(1 - \exp\left(-0.5A(y)\right)\right), \tag{3}$$

$$A(y) = \frac{\left|U\left(y - L_{\varepsilon 0}\right) - U\left(y + L_{\varepsilon 0}\right)\right|}{U_0},\ L_{\varepsilon 0} = \frac{k_0^{3/2}}{\varepsilon_0}. \tag{4}$$

Здесь $U_0$ – максимальная осредненная скорость в области сдвига.

Модель замыкалась соотношением Колмогорова-Буссинеска.

$$\nu_{t0} = C_\nu \frac{k_0^2}{\varepsilon_0}. \tag{5}$$

Использованные константы: $C_2$=1.45, $C_1$=0.685$C_2$, $C_\nu$=0.09, $C_\varepsilon$=1.3.

**2. Расчет течения в турбулентной круглой струе**

Результаты расчетов течения в круглой струе приведены на рис. 3-5. Расчеты средней скорости сопоставляются с результатами измерений Александера, Бэрона и Камингса, взятыми в работе Хинце [5] и

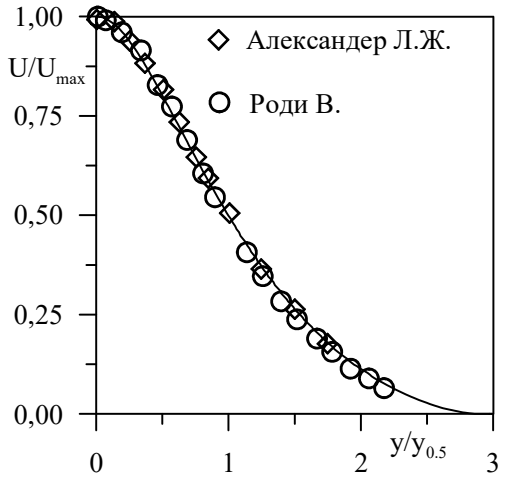



Рис. 3. Турбулентная круглая струя. Расчет средней скорости.

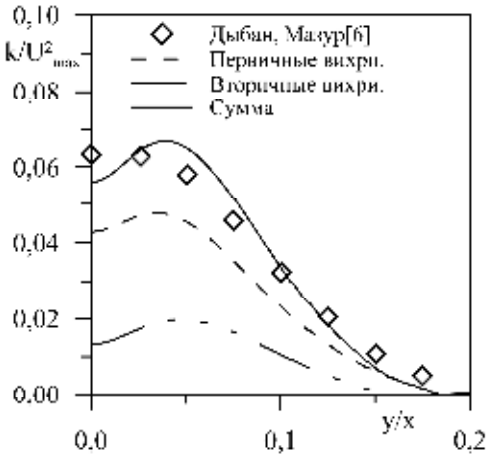



Рис. 4. Турбулентная круглая струя. Расчет энергии турбулентности.

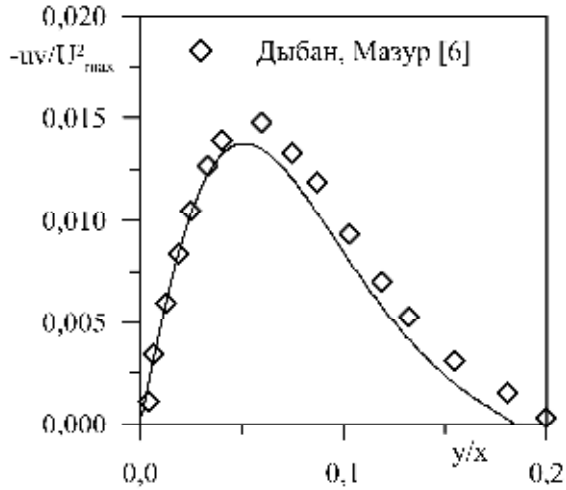



Рис. 5. Турбулентная круглая струя. Расчет турбулентного трения.

измерениями Роди, взятыми в работе [7]. Распределения пульсационных параметров сопоставляются с данными работы Дыбана, Мазура [6].

Недостатки расчета, как то снижение энергии турбулентности на оси струи, скорее всего объясняются недостатками исходной модели Джонса-Лаундера [3]. Эти недостатки неоднократно отмечались в литературе (см. например [7]).

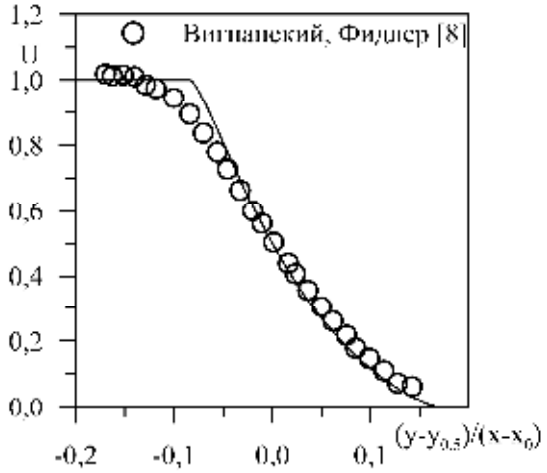



Рис. 6. Турбулентный слой смешения. Расчет средней скорости.

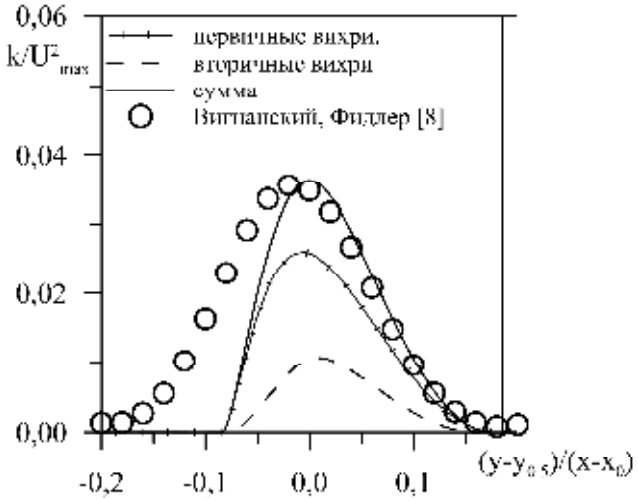



Рис. 7. Турбулентный слой смешения. Расчет энергии турбулентности.

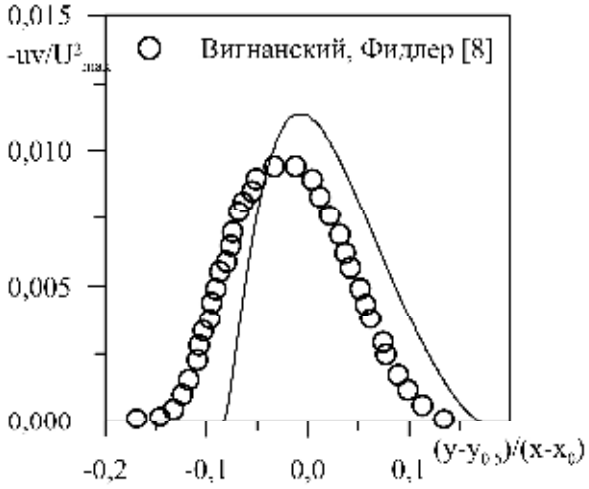



Рис. 8. Турбулентный слой смешения. Расчет турбулентного трения.

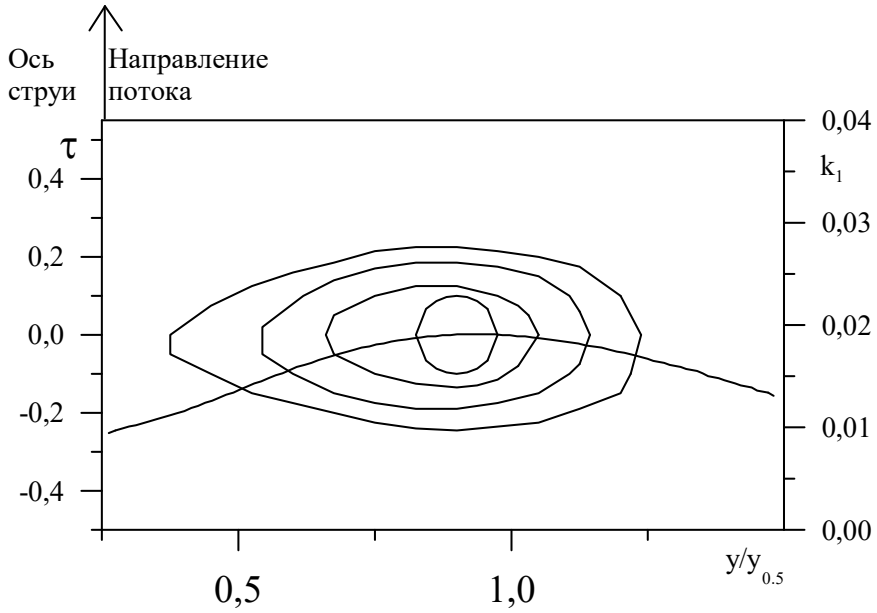



Рис. 9. Энергия вторичной турбулентности и распределение завихренности в спиралевидной когерентной структуре в круглой струе.

**3. Расчет течения в турбулентном плоском слое смешения**

Эта же модель была использована для расчета течения в плоском турбулентном слое смешения. Результаты расчетов представлены на рис. 6-8. Результаты расчетов сопоставляются с экспериментальными данными Вигнанского-Фидлера [8]. Несоответствие со стороны большей скорости полностью объясняются недостатками модели Джонса-Лаундера [3]. Расчеты, проведенные автором (здесь не приводятся), показывают, что все эти недостатки присущи также и k-ε модели Джонса-Лаундера.

**4. Когерентные структуры в турбулентных струе, следе и слое смешения**

В работе Тсо и Хуссейна [9] экспериментально обнаружено, что в круглой струе присутствуют когерентные образования трех видов - спиралевидные, в виде двойных спиралей и кольцевидные. Важнейшую роль из них играют спиралевидные образования. Отметим, что в экспериментах на поток накладывались периодические возмущения, позволяющие гораздо отчетливей выявлять когерентные структуры. На рис. 9 приведены контуры распределения завихренности в срезе спиралевидной структуры, полученные авторами [10] обработкой экспериментальных данных. Здесь же приведено распределение энергии вторичных вихрей. Видно, что и здесь соответствие максимума энергии и центра структуры практически точное.

На рис. 10 приведены значения функции тока когерентного движения в дальнем плоском следе за цилиндром, взятые из работы Антониа и др. [9]. Графики получены авторами [9] обработкой экспериментальных данных. На этом же рисунке приведены результаты расчетов энергии вторичных вихрей. Как видно из рис. 10, расположение пика этой энергии практически точно совпадает с центром структуры. Совпадение здесь следует рассматривать как совпадение расстояния от оси следа. Масштаб по оси Y не играет никакой роли.

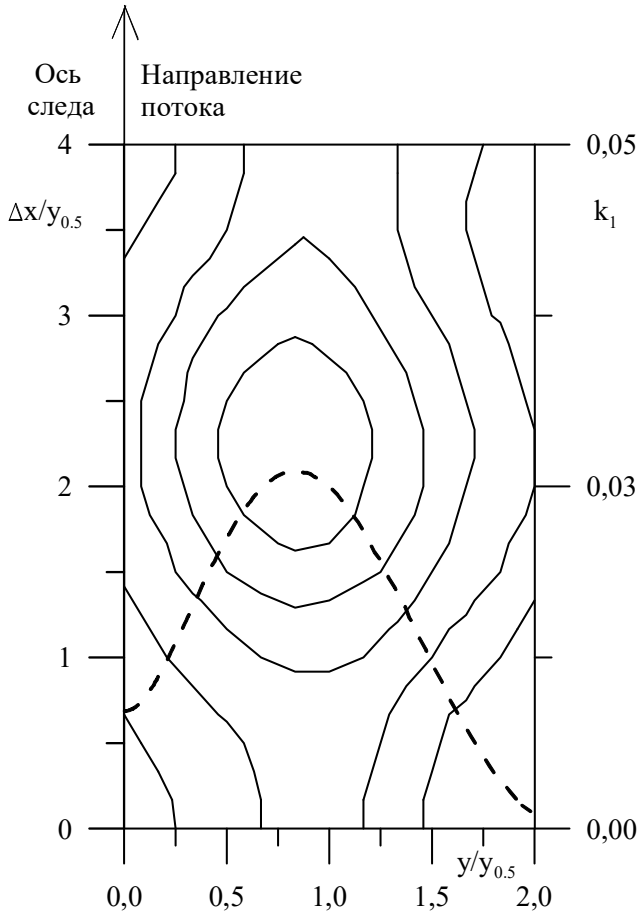



Рис. 10. Энергия вторичных вихрей и график функции тока когерентной структуры в плоском следе за цилиндром.

В работе Вигнанского и др. [11] изучался плоский турбулентный слой смешения. Так же как и в экспериментах Тсо и Хуссейна [10], в поток вносились синусоидальные осцилляции, помогающие гораздо отчетливей выявлять когерентные образования. Поток мог нагреваться до 25°С по сравнению с температурой в помещении. Температура использовалась авторами в качестве пассивного маркера. В слое обнаружено наличие когерентного образования. Авторами [11] отмечается, что пульсации температуры в центре образования, в связи с интенсивным перемешиванием, были заметно ниже, чем на его краях. На рис. 11 приведено распределение пульсаций температуры, зафиксированное авторами [11] в экспериментах. Здесь же показано распределение энергии вторичных вихрей. Как видно и здесь максимум графика этой энергии практически точно совпадает с минимумом пульсаций температуры, т.е. с центром когерентного образования. По данным авторов [11] аналогичное, но менее ярко выраженное распределение температурных пульсаций наблюдается и в отсутствии возбуждения потока.

Таким образом, расчеты по модели вида (4.1)-(4.4) четырех важнейших сдвиговых турбулентных течений, а именно - пограничного слоя, струи, следа и слоя смешения показывают, что распределение энергии вторичных вихрей хорошо соответствует расположению когерентных структур в течении.

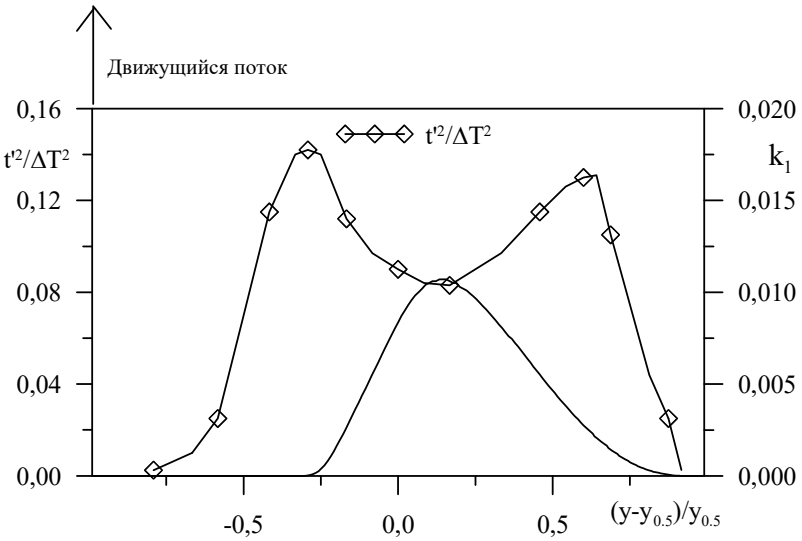



Рис. 11. Энергия вторичной турбулентности и распределение пульсаций тепературы в слое смешения.

**Выводы**

На основе предложенных [1] автором идей получена модель для расчета сдвиговых свободных турбулентных течений. В качестве генератора данных при разработке использовалась модель Джонса-Лаундера [3]. Надо сказать, что недостатки этой модели при расчете свободных турбулентных течений неоднократно отмечались в литературе (см. например Ханъялич, Лаундер, Шистель [7]). Так как данный подход не предполагает введения дополнительных слагаемых в модели, то все недостатки модели Джонса-Лаундера [3] автоматически перебрались и в новую модель.

**Список использованной литературы**

1. *B.P. Golovnya*, Important properties of turbulent near-wall flows which are not accounted by modern RANS models // Int. J. Heat Mass Transfer . - 2020 -V.146 - 118813.
2. *Себиси Т., Брэдшоу П.* Конвективный теплообмен. М.: - Мир – 1987.
3. *Jones W.P., Launder B.E.* The prediction of laminarisation with a two-equation model of turbulence // Int. J. Heat Mass Transf. – 1972 – V.15 – P.301-314.
4. *Головня Б.П.* Модель для расчета свободных турбулентных течений // Вісник СНУ ім. В.Даля – 2004 - № 4 (74) – с. 7-13.
5. *Хинце И.О.* Турбулентность – М.: ГИФМЛ. – 1963 – С.680
6. *Дыбан Е.П., Мазур А.И.* Конвективный теплообмен при струйном обтекании тел. Киев: - Наукова думка - 1982.
7. *Ханъялич К., Лаундер Б.Е., Шистель Р.* Концепция многих временных масштабов в моделировании турбулентного переноса //В кн. Турбулентные сдвиговые течения 2, М:.- Машиностроение - 1983 - С.42-57.
8. *Wygnansky I., Fiedler H.E.* The two-dimensional mixing region. // J.Fluid Mech. – 1970 - V.41 - P.327-361.
9. *Antonia R.A., Browne L.W.B., Bisset D.K., Fulachier L.* A Description of the Organized Motion in the Turbulent Far Wake of a Cylinder at Low Reynolds Number. // J. of Fluid. Mech. – 1987 - V.184 - P.423-444.
10. *Tso J., Hussain F.* Organized Motions in a Fully Developed Turbulent Axisymmetric Jet. // J. of Fluid. Mech. – 1989 - V.203 - P.425-448.
11. *Вигнанский И., Остер Д., Фидлер Г.* Плоский турбулентный слой смешения с искусственным возбуждением - задача для расчета. // В кн.: Турбулентные сдвиговые течения 2. М.: Машиностроение – 1983 - с.335-348.

**References**

1. *B.P. Golovnya*, Important properties of turbulent near-wall flows which are not accounted by modern RANS models // Int. J. Heat Mass Transfer . - 2020 -V.146 - 118813.
2. *T. Cebeci,., P. Bradshaw.* Physical and Computational Aspects of Convective Heat Transfer, Springer-Verlag, New York, 1984.

3. *Jones W.P., Launder B.E.* The prediction of laminarisation with a two-equation model of turbulence // Int. J. Heat Mass Transf. – 1972 – V.15 – P.301-314.
4. *B.P. Golovnya.* Model for simulation of free turbulent flows // Proc. of V. Dal's SNU – 2004 - № 4 (74) – c. 7-13.
5. J.O. Hinze, Turbulence, an Introduction to Its Mechanism and Theory, McGraw-Hill, New York, 1959.
6. *E.P. Dyban, A.I. Mazur.* Convective heat transfer at turbulent jet flows around bodies. Kyiv: - Naukova dumka - 1982.
7. *Hanjalic K, Launder B.E, Schieste R.,l* Multiple-Time-Scale Concepts in Turbulent Transport Modeling // In Turbulent Shear Flows 2: Selected Papers from the Second International Symposium on Turbulent Shear Flows, Imperial College, London, July 2-4, 1979
8. *Wygnansky I., Fiedler H.E.* The two-dimensional mixing region. // J.Fluid Mech. – 1970 - V.41 - P.327-361.
9. *Antonia R.A., Browne L.W.B., Bisset D.K., Fulachier L.* A Description of the Organized Motion in the Turbulent Far Wake of a Cylinder at Low Reynolds Number. // J. of Fluid. Mech. – 1987 - V.184 - P.423-444.
10. *Tso J., Hussain F.* Organized Motions in a Fully Developed Turbulent Axisymmetric Jet. // J. of Fluid. Mech. – 1989 - V.203 - P.425-448.
11. *Wignansky I., Oster D., Fiedler G.* Plane turbulent mixing layer with artificial exciting - problem for simulating // In Turbulent Shear Flows 2: Selected Papers from the Second International Symposium on Turbulent Shear Flows, Imperial College, London, July 2-4, 1979.

**GOLOVNYA Boris P.,**
Cherkasy National University named after Bogdan Khmelnitsky, Doctor of Science, Department of Applied Mathematics and Informatics

**MODEL FOR TURBULENT FREE FLOWS SIMULATION**

***Abstract. Introduction.*** Earlier, the author proposed a model for calculating shear turbulent flows near solid surfaces. The most important idea underlying the models proposed is based on the fact that if a vortex comes into contact with a solid surface and / or is subjected to shear action, then additional vortices arise in the flow. In free flows, there is no surface, but contact with shear remains. Therefore, the proposed approach is applicable to the construction of models for calculating free shear flows. As a result, a model was developed for calculating shear free turbulent flows. It is shown that the proposed model satisfactorily reproduces the basic properties of the main free turbulent flows, namely, a jet, a shear layer and a turbulent wake. The coherent structures obtained in the races correspond to the experimental data of different authors.

**Key words:** turbulence modeling, turbulence energy

**ГОЛОВНЯ Борис Петрович,**
Черкаський національний університет ім. Б. Хмельницького, доктор технічних наук, кафедра прикладної математики та інформатики

***Анотація. Вступ.*** Раніше автором була запропонована модель для розрахунку зсувних турбулентних течій поблизу твердих поверхонь. Найважливіша ідея, що лежить в основі пропонованих моделей, заснована на тому, що якщо вихор входить в контакт з твердою поверхнею і / або піддається сдвиговому впливу, то в потоці виникають додаткові вихори. У вільному потоці поверхню відсутній, але залишається контакт із зсувом. Таким чином, запропонований підхід можна застосувати до побудови моделей для розрахунку вільно зсувних течій. В результаті була розроблена модель для розрахунку турбулентних течій без зсуву. Показано, що запропонована модель задовільно відтворює основні властивості основних вільних турбулентних течій: струменя, сдвигового шару і турбулентного сліду. Отримані в гонках когерентні структури відповідають експериментальним даним різних авторів.

**Ключові слова**: моделювання турбулентності, енергія турбулентності.

*Одержано редакцією 23.03.2018*
*Прийнято до публікації 20.09.2018*

УДК 519.6:004.8

PACS 02.60.-x, 02.60.Pn, 02.70.Wz

**ЮХИМЕЦЬ Максим Вячеславович**
студент Черкаського національного університету ім. Б. Хмельницького

**КРАСНОШЛИК Наталія Олександрівна**
кандидат технічних наук, доцент, доцент кафедри прикладної математики та інформатики Черкаського національного університету ім. Б. Хмельницького
e-mail: wlik007@ukr.net

# ВИКОРИСТАННЯ МЕТОДІВ З ТЕОРІЇ ІГОР ДЛЯ ЗАСТОСУВАННЯ В ШАХОВОМУ ПРОГРАМУВАННІ

*У роботі розглянуто метод minimax, який бере свій початок в теорії ігор для гри двох осіб з нульовою сумою у випадках послідовних і одночасних ходів, а також метод його оптимізації альфа-бета обрізка яка прагне скоротити кількість вузлів, які оцінюються в дереві пошуку алгоритмом minimax. Метою даної роботи є застосування методі теорії ігор та методів їх оптимізація для практичної задачі створення шахового двигуна здатного на змістовні кроки. У роботі наведено формальне визначення даних методів, їх опис та приклади застосування. Проведено порівняння швидкодії роботи minimax з іншими методами цього ж класу, такими як negamax і negascout, та визначено доцільність використання останніх шляхом їх тестування у власному середовищі. Визначимо ефективність застосування альфа-бета обрізки та інших методів оптимізації для minimax та подібних йому алгоритмів пошуку.*

***Ключові слова:*** *minimax, negamax, negascout mtd(f), альфа-бета обрізка, шаховий двигун, теорія ігор.*

## Вступ

Перші згадування автономних шахових пристроїв датуються вісімнадцятим століттям. У той час ідея створення автоматичної машини домінувала у свідомості інженерів, хоча обмежений прогрес дозволив зробити лише декілька імітацій у 18-19 століттях. Турок, Аджиб і Мефісто - працювали завдяки людині-оператору прихованому в середині. Перша наукова робота в цій галузі була зроблена Клодом Шеноном у 1950 році. У ній обговорювалися два можливі підходи, перший покладався на вичерпну оцінку ходу, а другий на вибірковий підхід, коли двигун оцінював лише „хороші ходи”. На той час останній був більш привабливий, оскільки навіть найдосконаліші комп'ютери не могли обчислити більше трьох слоїв глибини (ходи, зроблені кожним з гравців) за розумний проміжок часу. Чемпіон світу з шахів Михайло Ботвіннік був одним з небагатьох шахових гросмейстерів, який присвятив час дослідженням комп'ютерних шахів. Його робота базувалася на вибірковому підході і втіленні досвіду гравців у шаховій програмі. Він намагався створити шахову програму здатну думати як гравець і робити тактичні жертви. Проте він так і не закінчив свою роботи, залишивши після себе масу теоретичних матеріалів. Першим вдалим двигуном, який відмовився від вибіркового пошуку і зосередився на пошуку на всю ширину був Chess 4.0. Він використовував класичний нині підхід Alpha-Beta та negamax, вдосконалена версія алгоритму minimax, обговореного Клодом Шенноном. Після того як пошук в ширину зарекомендував себе як вдалий підхід для вирішення шахової задачі, методи які здійснювали цей пошук почали активно розвиватися. Лише один minimax, що згадував в своїй роботі Клод Шенон, породив декілька вдосконалених алгоритмів пошуку: Negamax, SSS*, NegaScout, MTD(f). Для

комп'ютера далеко не очевидно, який із багатьох дозволених кроків є вдалим, а який ні. Найкращий спосіб розрізнити ці два способи – оцінити їх наслідки (тобто знайти ряд кроків, скажімо 4 для кожної сторони та переглянути результати.) І щоб переконатися, що ми робимо якомога менше помилок, ми припустимо, що супротивник також ходить свої найкращі кроки. Це головний принцип, покладений в основу мінімаксного алгоритму пошуку, який лежить в основі всіх шахових програм. На жаль, складність мінімаксу становить O (b *n), де b ("коефіцієнт розгалуження") - це кількість легальних кроків, доступних у середньому в будь-який момент часу, а n (глибина) - кількість "слоїв" на яку відбувається пошук, де один шар - це один хід сторони. Це число зростає дуже швидко, тому було проведено значний обсяг роботи над розробкою алгоритмів, які мінімізують зусилля, витрачені на пошук заданої глибини. Поглиблення ітерацій Alphabeta, NegaScout та MTD (f) є одними з найбільш успішних з цих алгоритмів, разом із структурами даних та евристиками, які роблять можливим сильну гру, наприклад, таблиці транспонування та історична евристика та евристика вбивці.

**1. Опис Minimax алгоритму та похідних від нього**

Minimax - це правило прийняття рішень, яке використовується у штучному інтелекті, теорії рішень, теорії ігор, статистиці та філософії для мінімізації можливих збитків у гіршому випадку (максимальна втрата). Коли йдеться про прибутки, це називається "максимін" - щоб максимізувати мінімальний виграш. Спочатку був сформульований для теорії ігор з нульовою сумою для n-гравців, що охоплює як випадки, коли гравці виконують почергові кроки, так і випадки, коли вони роблять одночасні кроки, пізніше також був поширена на більш складні ігри та загальне прийняття рішень за наявності невизначеності.

Для кращого розуміння цієї ідеї уявимо наступне. Наша програма отримує дошку з щойно зробленим кроком з боку гравця, цей стан або надалі вузол буде вершиною нашого рекурсивного дерева пошуку, рекурсія – це альтернатива ітераціям при якій повторення відбувається через виклик функції із самої себе до досягнення умови зупинки, після чого рух починається в зворотному боці ніби підіймаючись з глибини до місця першого виклику. Отримавши всі свої можливі ходи-відповіді, скориставшись спеціальним методом генерації кроків, програма по черзі робить кожен з них. Тут в нашій реалізації використовується методи push  зробить крок і pop – відмінити крок. Зробивши крок за свою сторону програма опиняється в стані дошки де черга кроку користувача, зробивши за нього найкращій крок, програма буде продовжувати спускатися в глиб дерева почергово роблячи найкращі кроки за кожну із сторін доки не досягне умови зупинки. Умови зупинки це перемога однієї зі сторін, нічия, досягнення заданої глибини, закінчення часу на крок. На останньому вузлі дерева ми виконуємо оцінку якості дошки і отримуємо число яке являється наближеним значенням того наскільки такий стан дошки хороший для однієї сторони і відповідно поганий для іншої. Це число буде в інтервалі [-inf, inf ] де нуль буде значити, що шанси на перемого обох гравців рівні. Тут використовується той факт, що шахи симетрична гра, тому функція оцінки повинна давати симетричну оцінку. Це доволі просто зрозуміти, якщо білі мають на одного пішака більше, то це значить, що чорні програють на одного пішака. На цій властивості базується подальше вдосконалення negamax, яке використовується в нашій програмі. Отже, після того як ми оцінили останній вузол потік виконання програми повертається разом з отриманим значенням на рівень рекурсії вище в точку виклику функції. Тут ми зберігаємо це значення і перевіряємо інші кроки. Отримавши всі значення оцінки останніх вузлів дерева ми обираємо найбільше або найменше значення, залежно від того чия черга

кроку на цьому вузлі. Якщо черга кроку біли, то ми обираємо найбільше значення, якщо чорних, то найменше. То якого кольору вузол визначається булевим параметром який передається як аргумент у функцію minimax. Класичний вигляд цього алгоритму припускає, що гравець білий, а комп'ютер чорний. В Nagamax цей момент опущений і ми максимізуємо для кожної сторони лише змінюючи знак оцінки на кожному кроці повернення. Повертаючи значення таким чином, опинившись в початковому вузлі, в нас буде оцінка кроку який буде приводити до кращого результату для нашої сторони при тому, що суперник буде ходити свої найкращі ходи.

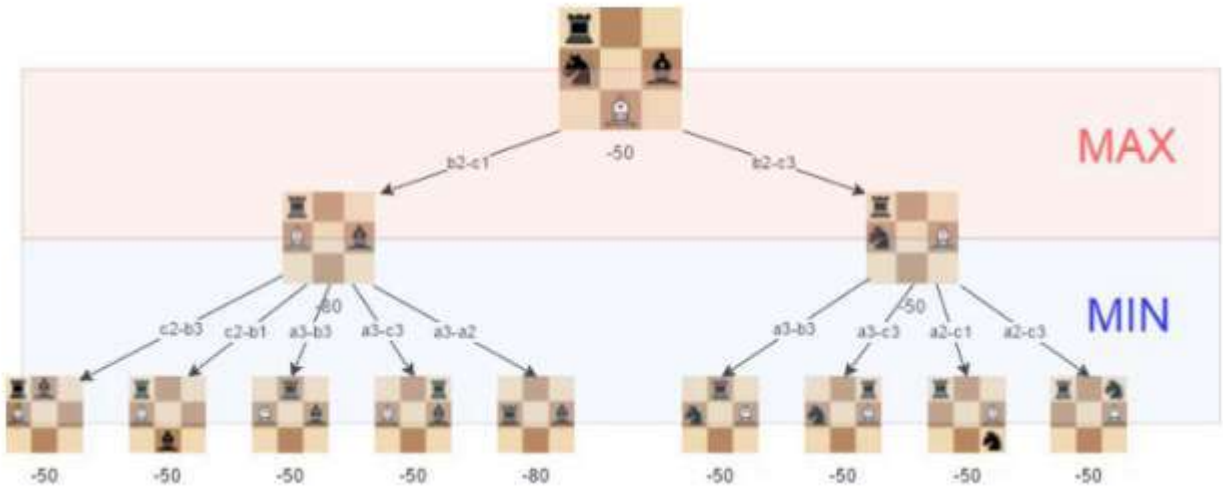



Рис. 1.1 Minimax дерево пошуку

Тут важливо відмітити, що minimax як і negamax виходить з припущення, що суперник завжди буде обирати найкращій хід в свою чергу. З цього слідує, що якісна функція оцінки життєво необхідна. Наведемо псевдокод minimax алгоритму.

```
int Minimax(state, level) {
  if(state is a solution)  //base case
    return state's value

  if(level is a max) {
    while(state has more children)
      Minimax(child, level+1)
      if value returned is > best so far, store it
    }
    return maximum returned value
  } else {  //level is a min level
    while(state has more children)
      Minimax(child, level+1)
      if value returned is < best so far, store it
    }
    return minimum returned value
  }
}
```

Рис. 1.2 Minimax псевдокод

Маючи в своїй програмі minimax і хорошу функцію оцінки можна вже грати в досить непогані шахи. Проте тут з'являється основна проблема, яка буде переслідувати

розробника впродовж всієї подальшої роботи - це компроміс між швидкодією і якістю кроків. Під час пошуку по всій ширині дерева, кожна додаткова глибина буде суттєво збільшувати кількість вузлів і це стане помітно вже на третій глибині, як показано на рис. 4.3.

| Depth (ply) | Number of Positions | Time to Search |
|---|---|---|
| 2 | 900 | 0.018 s |
| 3 | 27,000 | 0.54 s |
| 4 | 810,000 | 16.2 s |
| 5 | 24,300,000 | 8 minutes |
| 6 | 729,000,000 | 4 hours |
| 7 | 21,870,000,000 | 5 days |

Табл. 1.3 Зростання кількості позицій залежно від глибини

Всі методи модифікації функції пошуку які будуть описані далі призначені для зменшення цієї кількості. Ключовим серед них є AlphaBeta обрізка. Альфа-бета обрізка – це метод оптимізації minimax алгоритму і його похідних. Цей метод дозволяє нам ігнорувати перевірку деякої частини вузлів якщо виявляється, що знайдений в них результат точно не буде кращій ніж вже отриманий з іншої гіки дерева. Цей метод не впливає на якість роботи алгоритму, як може здатися, він лише пришвидшує його. Порядок перевірки вузлів впливає на якість обрізки, з цього слідує подальші вдосконалення.

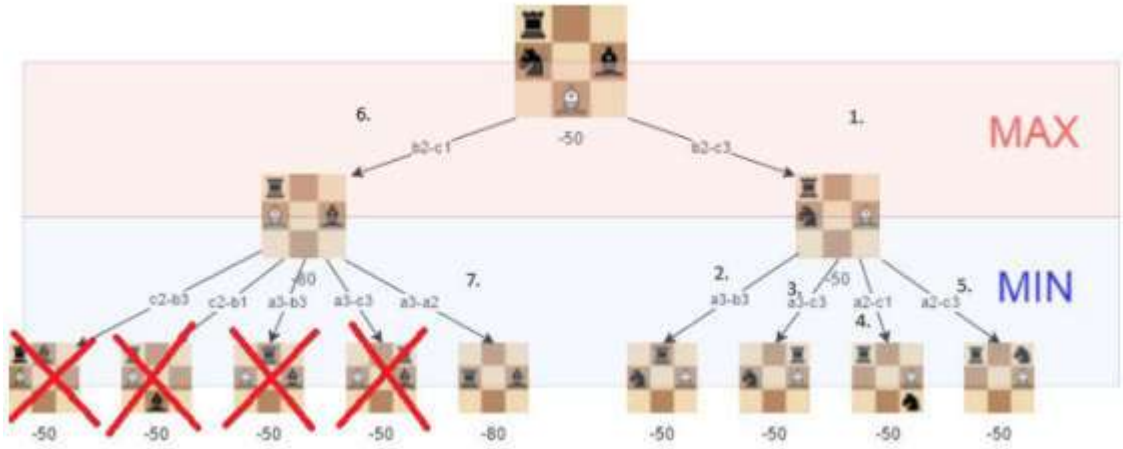



Рис. 1.4 Minimax дерево плюс альфа-бета обрізка

З рис.4.4 видно, що на 7 кроці ми отримали значення вузла -80, проте пройшовши одну гілку ми бачимо, що на рівні максимізації у нас вже є значення -50, а значення досліджуваного зараз вузла мінімізації по вже оціненому одному вузлу буде -80 або менше, тобто у нас вже є верхня мінімальна границя beta, більше якої значення точно не буде, також у нас є нижня максимальна оцінка alpha -50. Після повернення потоку управління в місце виклику у вузол під номером 6 виконається умова альфа-бета обрізки і наступні вузли не будуть перевірятися. Альфа-бета-пошук ефективно зменшує коефіцієнт розгалуження на кожному вузлів з 30-40 до приблизно 7 або 8 за умови, що у вас є достатньо хороша процедура початкового впорядкування ходу. Очевидно, вигідно мати найкращі ходи вгорі списку, так як вони набагато частіше

спричиняють обрізку. В коді логіка альфа-бета реалізується шляхом додавання двох аргументів у функцію пошуку і умови виходу з циклу перебору кроків.

```
initially alpha  -INFINITY, beta INFINITY

search(position,side,depth,alpha,beta) {

  best_score  -INFINITY

  for each move {

    do_move(position, move)

      if ( depth is 0 )  move_score  static_score(position, side)
      else  move score =  search(position, opponent side, depth 1,  beta,  alpha)

    undo move(position,move)

    if ( move score > best score )  best score = move score
    if ( best_score > alpha )  alpha  best_score
    if ( alpha >= beta )  return alpha
  }

  return best_score

}
```

Рис. 1.5 Псевдокод negamax плюс альфа-бета обрізка

Negascout - це алгоритм negamax, який може бути швидшим, ніж альфа-бета-обрізка. Як і альфа-бета-обрізка, NegaScout - це алгоритм спрямованого пошуку для обчислення мінімального значення вузла в дереві. Він домінує над альфа-бета-обрізанням в тому сенсі, що він ніколи не опрацює вузол, який можна обрізати альфа-бета; однак він покладається на точний порядок вузлів, щоб скористатися цією перевагою. NegaScout найкраще працює, коли є хороший порядок кроків. На практиці порядок переміщення часто визначається попередніми більш дрібними пошуками. Він дає більше обмежень, ніж альфа-бета, припускаючи, що перший досліджений вузол є найкращим. Іншими словами, передбачається, що перший вузол знаходиться в основному варіанті. Потім він може перевірити, чи це відповідає дійсності, шукаючи інші вузли за допомогою нульового вікна (також відомого як вікно розвідника; коли альфа та бета рівні), що швидше, ніж пошук за допомогою звичайного альфа-бета-вікна. Якщо перевірка не вдається, то перший вузол не був у головному варіанті, і пошук триває як звичайна альфа-бета. Отже, NegaScout найкраще працює, коли порядок кроків хороший. При випадковому впорядкуванні ходу NegaScout займе більше часу, ніж звичайна альфа-бета; хоча він не буде досліджувати ті вузли які б не досліджував альфа-бета, йому доведеться здійснити повторний пошук у багатьох інших вузлах. Олександр Рейнефельд винайшов NegaScout через кілька десятиліть після винаходу альфа-бета-обрізки. У своїй книзі він наводить докази правильності роботи NegaScout.

| Алгоритм | Глибина пошуку | | |
|---|---|---|---|
| | 3 | 4 | 5 |
| Minimax | 3.9 | 31.1 | 336.3 |
| Negamax | 0.74 | 5.8 | 249.4 |
| Negascout | 1.2 | 7.2 | 27.9 |

Табл. 1 Порівняння швидкодії роботи алгоритмів пошуку найкращого кроку

2. **Методи покращення**

У розробленій програмі реалізовано декілька технік покращення роботи шахового агенту. Наведемо перелік останніх, де квадратами позначимо ті які

залишилися у фінальній реалізації додатку, а кругами які ні, і пройдемося по кожному з них.

- Iterative deepening
- Transposition table
- Zobrist key
- Move ordering:
- Best move from previous iterations
  - Killer move
  - MVV/LVA
  - Null move
- Opening book
- Quiscience search
- Principal Variation Search

Цей список удосконалень не являється навіть чвертю усіх запропонованих на сьогоднішній день підходів, деякі з них доволі складні,інші займають декілька строк коду, але критерій для всіх однаковий, збільшення швидкодії або якості знайдених кроків. Евристики які стосуються функції оцінки будуть наведені у відповідному розділі, а зараз розглянемо кожен пункт в порядку представлення.

***Iterative deepening*** – контрінтуїтивний метод який замість пошуку на задану глибину проводить ітеративний пошук починаючи з глибини 1, 2, … , n-1, n де n – задана глибина пошуку. Пошук буде продовжуватися 1) доки не буде досягнуто мінімальної глибини пошуку; 2) поки не вичерпається час пошуку; 3) або поки не буде досягнуто заданої глибини. Проміжні результати пошуку зберігаються у transposition table. У цьому власне і полягає причина працездатності цього підходу, адже зберігаючи знайдені на попередніх ітераціях кращі кроки, наступна ітерація програма не буде шукати їх повторно, а візьме з таблиці. Про те як кроки додаються і повертаються з таблиці буде пізніше, зараз же відмітимо, що такий підхід пришвидшує пошук порівняно з пошуком на відразу фіксовану глибину. Окрім цього ітеративний пошук дає ще декілька переваг. Перша очевидна перевага цього методу полягає в тому, що у вас завжди буде якийсь результат, який ви зможете показати після пошуку, і ви знаєте, що це не буде повною помилкою, принаймні до деякої глибини. Також з'являється можливість додавати обмеження по часу для пошуку, а інформація про попередні кращі кроки дасть можливість сортувати їх у такому порядку щоб альфа-бета обрізка наступала швидше.

***Transposition table*** – невід'ємна складова попереднього методу, по своїй суті являється хеш-таблицею, в якій окрім оцінки кроку зберігаються ще деякі параметри. Перш ніж здійснювати пошук, двигун перевіряє хеш-таблицю, чи не було раніше здійснено пошук поточної позиції дошки. Якщо її раніше шукали, принаймні до тієї глибини, до якої зараз, то немає необхідності шукати його знову. Якщо позиції немає в таблиці, пошук здійснюється як зазвичай. Таблиця транспонування в шахових двигунах працює шляхом створення хеш-ключів по яким зберігається значення оцінки кращого кроку в цьому положенні дошки, сам хід, глибину на якій його знайшли і верхні та нижні границі альфа-бета. Функція яка перетворює стан дошки в хеш-ключ називається Zobrist key, вона реалізована в нашій програмі.

***Zobrist key*** – це спосіб хешування який використовується як частина комп'ютерних програм, які грають в настільні ігри з ідеальною інформацією, наприклад, в шахи та Go, для виконання таблиць транспонування. Ідея полягає в тому, що ми створюємо масив 8*8 де кожний ij елемент це список із 12 випадкових 64 – бітних чисел. Після цього ми користуючись цим масивом відображаємо дошку в одне число наступним чином.

```
def zobristHash(board):
    ''' returns hash according to current board config'''
    hash = 0
    for each cell on board :
        if cell is not empty:
            piece = board[cell]
            hash ^= table[cell][piece]
    return hash
```

Відповідно тепер, коли ми робимо якусь зміну на дошці, нам потрібно взяти по індексу рядка і колонки випадкове число асоційоване з цією фігурою і виконати операцію XOR по тому індексу де “стояла”фігура, а потім по тому куди вона перемістилася.

```
# move white Rook to at a new postion in right
    piece = board[1][1]

    board[1][1] = '-'
    hashValue ^= zobTable[1][1][indexing(piece)]

    board[3][1] = piece
    hashValue ^= zobTable[3][1][indexing(piece)]
```

Таким чином ми отримаємо унікальне 64-бітне число для кожної ситуації на дошці. В цю систему можна також легко включити відслідковування рокіровки, можливість взяття на проході і інші характеристики дошки.

***Move ordering*** – це доволі просте в технічному плані доповнення, але неймовірно ефективне на практиці. Вище вже згадувалося, що порядок в якому алгоритм пошуку обходить кроки суттєвий і що необхідно ставити потенційно кращі кроки ближче до початку. В нашій програмі ми використовуємо три прийоми щоб розмістити кращі ходи першими. Best move from previous iteration – назва говорить сама за себе, ми просто беремо кроки які потрапили у таблицю транспонування на відповідній глибині і розміщуємо їх спереду черги. Killer move – це хеш таблиця в якій зберігаються кроки з попередніх ітерацій на яких частіше за все виконувалась умова альфа-бета обрізки. Якщо спочатку перевіряти ці ходи, тоді обрізка буде набагато кращою, оскільки комп’ютеру не доведеться витрачати час на пошук багатьох інших поганих ходів. MVV/LVA – проста техніка яка застосовується тільки для кроків атак. Якщо фігура атакує іншу фігуру більшої вартості, то така атака більш пріорітетна ніж коли фігура більшої вартості атакує меншу. Тобто, якщо пішак б'є ворожу королеву то така атака повинна перевірятися першою

***Null move*** – це розумний і відносно недавній метод, вперше запропонований Доннінгером (1993). Алгоритм простий, і насправді просто кодує поняття, яке люди використовують вже багато років, не знаючи цього. В основному теорія наступна; якщо ви вирішили НЕ грати хід ("пас"), дозволяючи супернику зіграти два послідовних ходи, і його рахунок все ще не є хорошим, то, очевидно, ваша позиція дуже сильна. У термінології програмування, якщо сторона, яка грає, вирішує не рухатися, і рахунок, повернутий пошуком на зменшеній глибині для опонента, нижчий за альфа або оцінка для сторони, що грає, більша за бета, тоді просто поверніть це значення замість того, щоб заважати собі правильно шукати гілку. Зрозуміло, що цей вузол настільки сильний, що йому ніколи не буде дозволено виникнути. Фактично все,

що ми зробили, – це лише зпрогнозували відсікання бета-версії без фактичного пошуку всього дерева. Однак цей метод надзвичайно небезпечний, і при необережному використанні він може спричинити велику кількість тактичних помилок. Найголовніше - проблема позицій цугцвангу. Це німецький термін, що означає, що людина, яка першою зіграла на певній позиції, програє. Null move обрізка не повинна застосовуватися, коли є ймовірність цугцвангу. Є кілька способів уникнути цього. По-перше, null move ніколи не використовується, коли у сторони залишається лише кілька фігур. По-друге, вона ніколи не використовується, коли стороні, яка грає, можуть поставити мат найближчим часом.

***Opening book*** – також проста в реалізації, але ефективна ідея, яка полягає у використанні дебютних книг. Це об'ємні бінарні файли у яких зберігаються початкові ходи відомих шахових партій та типові стратегії початкового розіграшу. Доволі корисно, так як можна повністю прибрати пошук перших 5 – 8 кроків, але працює лишу у випадку, коли суперник відповідає правильними ходами і продовжує дебют. По аналогу з дебютними книгами існують такі ж книги для ендшпилів, проте вони займають куди більше пам'яті.

***Quiscience search*** – метод який створений для боротьби з ефектом горизонту. Це ситуація різкої зупинки пошуку на фіксованій глибині яка може призвести до хибного рішення зробити бій не бачучи що на глибину нижче суперник зможе забрати вашу більш цінну фігури. Щоб обійти цю проблему всі хороші шахові програми застосовують метод, який називається "спокійний пошук". Як випливає з назви, він передбачає пошук за кінцевими пошуковими вузлами та тестування всіх неспокійних рухів атак, поки ситуація не стане спокійною. Це дозволяє програмам виявляти довгі послідовності захоплення та обчислювати, чи варто їх ініціювати. У цього методу є недолік, якщо не контролювати глибину на яку він продовжує додатковий пошук по всіх крокам боям, то час пошуку може зростати в декілька разів. На власній практиці цей метод не дав суттєвої переваги в якості, тому потребує детальнішого вивчення і повторних випробуваннях.

***Principal Variation Search*** – цей метод належить до класу тих які змінюють початкові значення альфи і бети, тим самим роблячи рамки обрізки більш жорсткими, що при не правильній реалізації може проявитися в помилках які майже неможливо відслідкувати. Конкретно цей метод полягає в тому, що при якісній функції сортування кроків найкращій хід буде першим, тому тільки його і є сенс шукати належним чином з альфа = -inf, бета = inf інші кроки можна продивитися з вузьким вікном альфа-бета тільки щоб переконатися, що вони не кращі. Якщо ми були правими у своєму припущенні і найкращий хід дійсно був у передній частині списку, тоді цей пошук поверне приблизне значення набагато швидше, ніж повний пошук. Однак, якщо ми помилялися, він вийде з високим відхиленням бети, що вказує на те, що цей крок насправді покращить альфу, і тому його потрібно правильно шукати. Даний метод також потребує додаткового дослідження, для його коректного застосування у нашій програмі.

### 3. Функція оцінки

Функція оцінки – це елемент шахової програми який визначає пріоритети і тактику штучного агента. Оцінка дошки складається з набору евристик кожна з яких втілює певний патерн успішної гри. Найпростіший аналіз дошки - це просто підсумовувати бали, які кожна сторона має на дошці. Це так звана матеріальна евристика або матеріал. В кожній шаховій програмі задається цінність фігур, зазвичай це 100 для пішака 300 для коня і слона, 500 для тури, 900 для королеви і дуже велике значення для короля. Ці бали відкриті для налаштування. Дійсно, багато програм

трохи підвищують бали за тури або роблять різні бали коню і слону. Однак очевидно, що просто додавання матеріальних балів не дасть сильної програми. Потрібно враховувати ще багато факторів. Часто два гравці можуть мати однаковий матеріал, але один гравець матиме величезну перевагу в розвитку чи структурі пішаків, що призводить до простого виграшу. Безумовно, потрібно розрізняти позиції, які є дещо кращими для однієї сторони, і позиції, які дають явну перевагу іншій стороні. Це евристика розміщення фігур на дошці.

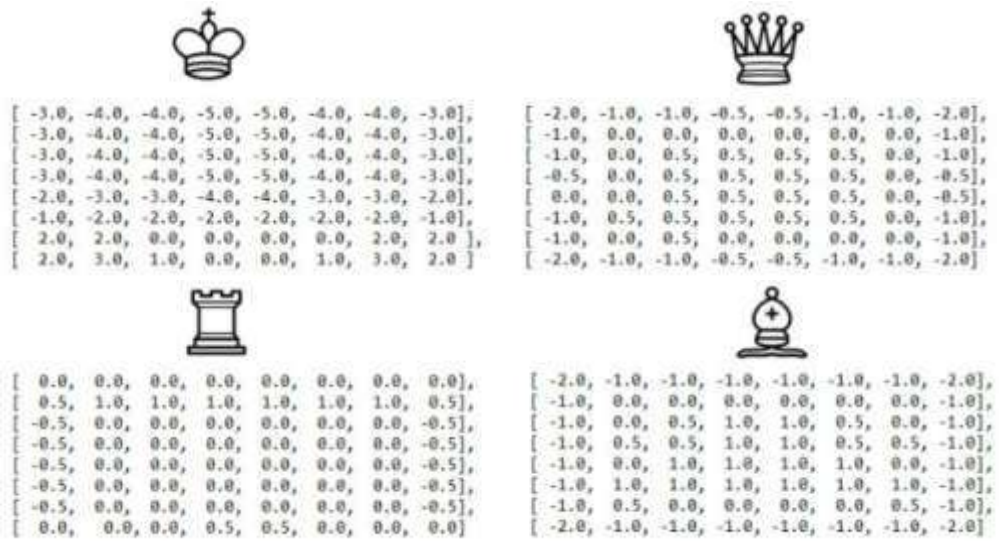



Рис. 3.1 Приклад позиційних таблиць

Третя обов'язково евристика це перевірка на шах і мат, функція оцінки буде повертати –inf, +inf в залежності від сторони якій був поставлений шах і мат. Це необхідно щоб штучний агент міг закінчувати гру не тільки у нічию, так як без цієї евристики програма буде намагатися побити всі фігури уникаючи можливості поставити шах і мат. Однак ця евристика може спричинити поведінку, при якій штучний агент буде намагатися попередити шах і мат зі сторони суперника і заради цього буде віддавати фігури. Це цікава ситуація яка не являється помилкою, але і не є допустимою для хорошої гри. На даний момент єдиний спосіб коректування цього це збільшення глибини пошуку. Окрім цих трьох евристик може бути ще з десяток інших, але треба розуміти, що функція оцінки дошки виконується найбільшу кількість разів в процесі виконання пошуку, тому треба шукати компроміс між швидкодією і якістю. Також треба відмітити, що в процесі розробки програми виявиться, що далеко не всі евристики будуть однозначно покращувати якість програми. Проте приведемо деякий набір останніх:

Структура пішаків – штрафуємо коли два і більше пішаки знаходяться на одному файлі, так як це перешкоджає просуванню; штрафуємо коли пішак заблокований і не може рухатися; заохочуємо просування пішака під захистом; заохочуємо структури із пішаків, коли позаду прикриває того що спереду; заохочуємо просування до протилежного рангу для promotion.

Розміщення фігу – заохочуємо коней займати центр дошки; заохочуємо слонів займати основні діагоналі; заохочуємо королеву та тур захищати один одного; заохочуємо рокіровку на початку гри; штрафуємо за нереалізовані важливі фігури у середині гри; заохочуємо контроль клітинок та захист союзних фігур; заохочує короля до знаходження в центрі дошки в середині гри.

Як видно з цих порад також важливо розділяти етапи гри і для кожного мати свій набор евристик, в тому числі і різні таблиці позиціонування та оцінку матеріалів.

**Висновки**

У роботі розглянуто метод minimax та похідні від нього методи пошуку кроку, метод negamax і negascout, які відносяться до методів теорії ігор, а також способи їх вдосконалення. Ефективність пошукових методів була перевірена у власному шаховому двигуні та представлена в таблиці результатів. Серед способів вдосконалення алгоритму пошуку було перебрано деяку частину. Кожен таких спосіб реалізовувався у програмі та перевірявся на практиці під час гри, деякі з них показували суттєвий приріст швидкодії і якості гри, такі методи залишалися в кінцевій версії проекту. Інші давали, неочікувано, погіршення результату.

Метод перевірки полягав в багаторазовій прогонці поточної комплектації пошукового модуля в ігровому середовищі, та подальшого порівняння отриманого співвідношення виграшів / програшів з результатом пошукового модуля стандартної комплектації. Таким чином визначалося який з способів вдосконалення покращує гру, а який залишає без змін, але з втратою у швидкодії.

Оціночна функція, як одна з ключових складових шахової програми, неодноразово піддавалася переробці. Характеристики гри, по яким відбувається оцінка дошки, можуть бути самими різноманітними, але обов'язково повинні включати три евристики: матеріал, позиція, перевірка на шах і мат. Інші евристики будуть додатково акцентувати увагу на певних стратегічних моментах і можуть покращувати або погіршувати гру, також можлива ситуація перекосу в певний патерн поведінки під час гри, наприклад атакуючий стиль або прагнення виконати спеціальний хід. Підбір оптимального набору евристик кропітка задача і може затребувати багато часу на тестування. В кінцевому варіанті мого пошукового модулю було вирішено використовувати евристику матеріалу, позиції, перевірку на шах і мат, пішакові структури і рокіровку.

Отже, була проведена робота по дослідженню і порівнянню деяких методів з теорії ігор по доцільності застосування у задачах шахового програмування з застосуванням способів оптимізації останніх.

**Список використаної літератури:**


1. Клод Шеннон; «Програмування комп'ютера для гри в шахи» (англ. «Programming a Computer for Playing Chess »), опублікованій в Philosophical Magazine, 1950 р.
2. "Chess For Dummies" – by James Eade (Author) August 29, 2016
3. Словарь терминов шахматной композиции / Авт.-сост. Басистый М. Б. - К.: Книга, 2004. - С. 252.
4. "FIDE Laws of Chess taking effect from 1 January 2018". FIDE.
5. Сайт, присвячений шаховому програмуванню [Електронний ресурс]. – Режим доступу: https://www.chessprogramming.org
6. Edward W. Kozdrowicki, Dennis W. Cooper (1973). COKO III: The Cooper-Koz Chess Program. Communications of the ACM, Vol. 16, 7, Fig. 2
7. “A Chess Engine” - Paul Dailly, Dominik Gotojuch, Neil Henning, Keir Lawson, Alec Macdonald, Tamerlan Tajaddinov. University of Glasgow Department of Computing Science Sir Alwyn Williams Building Lilybank Gardens Glasgow G12 8QQMarch 18, 2008 https://www.researchgate.net/publication/268426213_A_Chess_Engine
8. Авторський блог присвячений шаховому програмуванню [Електронний ресурс]. – Режим доступу: https://www.youtube.com/channel/UCB9-prLkPwgvlKKqDgXhsMQ
9. Серія статей на інтернет ресурсі gamedev.net [Електронний ресурс]. – Режим доступу: https://www.gamedev.net/tutorials/_/technical/artificial-intelligence/chess-programming-part-i-getting-started-r1014/
10. Авторський блог Computer Chess Programming Theory by Colin Frayn [Електронний ресурс]. – Режим доступу: http://www.frayn.net/beowulf/theory.html

11. Документація шахового двигуна DarkThought [Електронний ресурс]. – Режим доступу: http://people.csail.mit.edu/heinz/dt/

**References:**

1. Claude Shannon; "Programming a Computer for Playing Chess", published in Philosophical Magazine, 1950.Talimonchyk A., Krasnoshlyk N. (2016)
2. "Chess For Dummies" – by James Eade (Author) August 29, 2016
3. Dictionary of terms of chess composition / Author. Bassist MB - K .: Book, 2004. - P. 252.
4. "FIDE Laws of Chess taking effect from 1 January 2018". FIDE.
5. Site dedicated to chess programming [Electronic resource]. - Access mode: https://www.chessprogramming.org
6. Edward W. Kozdrowicki, Dennis W. Cooper (1973). COKO III: The Cooper-Koz Chess Program. Communications of the ACM, Vol. 16, 7, Fig. 2
7. “A Chess Engine” - Paul Dailly, Dominik Gotojuch, Neil Henning, Keir Lawson, Alec Macdonald, Tamerlan Tajaddinov. University of Glasgow Department of Computing Science Sir Alwyn Williams Building Lilybank Gardens Glasgow G12 8QQMarch 18, 2008 https://www.researchgate.net/publication/268426213_A_Chess_Engine
8. Site dedicated to chess programming [Electronic resource]. - Access mode: https://www.youtube.com/channel/UCB9-prLkPwgvlKKqDgXhsMQ
9. A series of articles on the Internet resource gamedev.net [Electronic resource]. - Access mode: https://www.gamedev.net/tutorials/_/technical/artificial-intelligence/chess-programming-part-i-getting-started-r1014/
10. Author's blog Computer Chess Programming Theory by Colin Frayn [Electronic resource]. - Access mode: http://www.frayn.net/beowulf/theory.html
11. Documentation of the chess engine DarkThought [Electronic resource]. - Access mode: http://people.csail.mit.edu/heinz/dt/

**YUKHIMETS Maxim,**
The Bohdan Khmelnytsky National University of Cherkasy, student
**KRASNOSHLYK Nataliya,**
The Bohdan Khmelnytsky National University of Cherkasy, PhD, Senior Lecturer

**USE OF GAME THEORY METHODS FOR APPLICATION IN CHESS PROGRAMMING**

***Abstract. Introduction.*** *To a computer, it is far from obvious which of many legal moves are "good" and which are "bad". The best way to discriminate between the two is to look at their consequences (i.e., search series of moves, say 4 for each side and look at the results.) And to make sure that we make as few mistakes as possible, we will assume that the opponent is just as good as we are. This is the basic principle underlying the minimax search algorithm, which is at the root of all chess programs. Unfortunately, minimax' complexity is O(b[sup]n[/sup]), where b ("branching factor") is the number of legal moves available on average at any given time and n (the depth) is the number of "plies" you look ahead, where one ply is one move by one side. This number grows impossibly fast, so a considerable amount of work has been done to develop algorithms that minimize the effort expended on search for a given depth. Iterative-deepening Alphabeta, NegaScout and MTD(f) are among the most successful of these algorithms, and they will be described in Part IV, along with the data structures and heuristics which make strong play possible, such as transposition tables and the history/killer heuristic. Another major source of headaches for chess programmers is the "horizon effect", first described by Hans Berliner. Suppose that your program searches to a depth of 8-ply, and that it discovers to its horror that the opponent will capture its queen at ply 6. Left to its own devices, the program will then proceed to throw its bishops to the wolves so that it will delay the queen capture to ply 10, which it can't see because its search ends at ply 8. From the program's point of view, the queen is "saved", because the capture is no longer visible... But it has lost a bishop, and the queen capture reappears during the next move's search. It turns out that finding a position where a program can reason correctly about the relative strength of the forces in presence is not a trivial task at all, and that searching every line of play to the same depth is tantamount to suicide. Numerous techniques have been developed to defeat the horizon effect.*

***Purpose.*** *The purpose of the article is to investigate the given methods, implement them and test their performance in chess programming tasks*

***Results.*** *The paper considers the minimax method and its derived step search methods, the negamax and negascout method, which belong to the methods of game theory, as well as ways to improve them. The effectiveness of search methods was tested in our own chess engine and presented in the table of results. Among the ways to improve the search algorithm, some were listed. Each of these methods was implemented in the program and tested in practice during the game, some of them showed a significant increase in speed and quality of the game, such methods remained in the final version of the project. Others gave, unexpectedly, a worsening of the result.*

*The method of verification was to repeatedly run the current configuration of the search module in the game environment, and then compare the obtained ratio of winnings / losses with the result of the search engine standard configuration. Thus, it was determined which of the ways to improve improves the game, and which leaves unchanged, but with a loss of speed.*

*The evaluation function, as one of the key components of the chess program, has been repeatedly processed. The characteristics of the game on which the board is evaluated can be very diverse, but must include three heuristics: material, position, check for chess and checkmate. Other heuristics will further focus on certain strategic points and may improve or worsen the game, as well as a situation of distortion in a certain pattern of behavior during the game, such as attacking style or the desire to make a special move. Finding the optimal set of heuristics is a painstaking task and can take a long time to test. In the final version of my search module, it was decided to use material heuristics, positions, chess and checkmate, pawn structures and castling.*

***Conclusion.*** *Thus, work was carried out to study and compare some methods of game theory on the feasibility of application in chess programming problems using methods to optimize the latter.*

***Key words****: minimax, negamax, negascout mtd (f), alpha beta pruning, chess engine, game theory.*

*Стаття надійшла 11.06.2018*
*Прийнято до друку 20.09.2018*

УДК 519.6:004.8

PACS 02.60.-x, 02.60.Pn, 02.70.Wz

**СЕРДЮК Олександр Анатолійович**
старший викладач кафедри прикладної математики та інформатики Черкаського національного університету ім. Б. Хмельницького

**КОНОГРАЙ Анастасія Миколаївна**
учениця 10-А класу Чорнобаївської гімназії Чорнобаївської районної ради Черкаської області

# ПОШУК ГЕОМЕТРИЧНИХ ФІГУР НА ЗОБРАЖЕННЯХ З УРАХУВАННЯМ ПЕРЕТВОРЕННЯ ХАФА

*Одним з найбільш поширених форматів даних є цифрові зображення. Одна з найбільш складних операцій при обробці зображень – пошук і виділення на них образу або фігури, яку треба знайти. Хаф запропонував перетворення, що дозволяє переводити координати точок двомірного простору в дані акумуляторного масиву з подальшим застосуванням процедури голосування. Дослідження в області пошуку геометричних об'єктів на зображеннях є метою вивчення перетворення Хафа, його застосування і пошуку методів його оптимізації є дуже актуальними.*

*Метою статті є розробка алгоритму пошуку полігонів на зображенні з використанням перетворення Хафа та визначення особливостей роботи алгоритму. В процесі роботи необхідно розв'язати наступні задачі:*

*1) проаналізувати умови, за яких можлива автоматизація виділення певних геометричних фігур на зображенні;*

*2) розглянути метод пошуку прямих ліній на зображенні із використанням перетворення Хафа;*

*3) розв'язати задачу визначення точки перетину двох прямих ліній;*

*4) розробити власну програмну реалізацію пошуку многокутників (полігонів) на зображенні.*

*Опрацювавши дану тему, зробивши практичні дослідження, приходимо до висновку, що на підставі отриманих теоретичних результатів розроблено математичну модель знаходження геометричних фігур на зображенні.*

***Ключові слова:*** *перетворення Хафа, геометричні перетворення на площині.*

## Вступ

Кожен день прогрес і розвиток сучасних наук і технологій вносить новий вклад у життя і розвиток суспільства. Мабуть, на землі залишилося мало людей, які не використовують сучасні електронно-обчислювальні засоби в своєму житті. Практично неможливо уявити життя більшості людей і галузей господарства без таких звичних для нас речей як персональний комп'ютер, мобільний телефон та інших.

У зв'язку з цією тенденцією кількість інформації на землі збільшується дуже швидко і цей процес неможливо зупинити. Сучасна людина, як би вона не хотіла і не прагнула, не здатна засвоїти і обробити всю інформацію, збережену в сучасному світі. І практично вся ця інформація зберігається в цифровому вигляді на різних пристроях зберігання в різних частинах світу. Це дає можливість легко отримувати до неї доступ за допомогою будь-якого електронного пристрою, здатного працювати з даним типом файлу і носія, також полегшується копіювання інформації та її поширення, наприклад за допомогою глобальної мережі інтернет.

У зв’язку із проблемою зростання кількості інформації і практичною неможливістю обробити її всю вручну, виникла ідея її автоматичної обробки за допомогою електронно–обчислювальних пристроїв. Одним з найбільш поширених форматів даних є цифрові зображення. Одна з найбільш складних операцій при обробці зображень – пошук і виділення на них образу або фігури, яку треба знайти. Дані методи особливо актуальні в наш час, коли їх активно застосовують не тільки у картографії і фотографіях, а й у багатьох автоматизованих системах реального часу, де важлива точність та надійність.

***Актуальність теми***. Завдання автоматизованої обробки даних стояло перед людством ще з часів появи самих даних. Але тільки з появою цифрових форматів збереження цей процес значно просунувся вперед. Ще в 60-х роках XX століття Хаф запропонував перетворення, що дозволяє переводити координати точок двомірного простору в дані акумуляторного масиву з подальшим застосуванням процедури голосування [11]. І хоча в той час цифровий формат зберігання був не особливо поширений, проте, дане перетворення дуже зацікавило наукове співтовариство. У подальшому різними вченими були запропоновані варіанти даного алгоритму для знаходження не тільки лінії, але й кіл, еліпсів і навіть об'єктів довільної форми [1].

З часом розповсюдження цифрових форматів зображень дало ще більший поштовх до освоєння області автоматичної обробки зображень. І до перетворення Хафа стали звертатися все частіше. Виникло безліч його модифікацій, які прискорювали алгоритм і робили його досить швидким. Це дозволяло застосовувати його в реальних системах виявлення об'єктів [8] і навіть в системах реального часу. Так само позитивним моментом у розвитку цього напряму став грати той факт, що з кожним роком обчислювальна потужність сучасних ЕОМ зростає, що так само допомагає прискорити процес пошуку об'єктів на зображенні.

Вже зараз перетворення Хафа може застосовуватися в таких областях як розпізнавання контурів будівель на зображеннях [12], визначення лінії горизонту [10], знаходження ліній дорожньої розмітки, визначення кількості осей у рухомого транспорту та інших сферах.

Таким чином, дослідження в даній області з метою вивчення даного перетворення, його застосування і пошуку методів його оптимізації є дуже актуальними.

**Аналіз існуючих методів знаходження графічних примітивів**

**Графічні примітиви і методика їх відшукання**

Під графічним примітивом розуміється найпростіший геометричний об'єкт, який відображається на екрані дисплея. Основне призначення примітивів – забезпечити програмістів і користувачів зручним набором засобів для формування геометричних об'єктів. Опис графічного примітиву зазвичай містить метричну і атрибутивну частини. Метрична частина дозволяє зіставити ті величини, в яких заданий графічний примітив для відображення його на дисплеї і ті величини, які характеризують його фізичне або логічне представлення. Атрибутивна частина передає геометричні параметри, що характеризують форму і розташування графічного примітиву. В основі побудови будь-яких графічних елементів в сучасних графічних бібліотеках лежать три основних примітиву: точка, відрізок і трикутник. Нерідко в якості примітиву виступають прямокутники, кола, еліпси, полігони довільних форм та інші. Класифікація графічних примітивів приведена в таблиці 1 (додаток А1). У більшості завдань, що вирішуються в рамках комп'ютерного зору, особливе значення мають точки, відрізки, трикутники, кола та прямокутники. Для виявлення графічних примітивів на зображенні використовується ряд методів та алгоритмів.

У даному дослідженні використаємо метод, який базується на перетворенні Хафа.

**Аналіз і попередня обробка вхідних даних**

Вхідними даними для перетворення Хафа служать монохромні зображення. Процес отримання геометричного опису простору зображення полягає в проходженні наступних етапів:

а) введення кольорового зображення;
б) перетворення в півтоноване зображення;
в) згладжування та порогова фільтрація (для усунення шумів);
г) бінаризація.
д) перетворення Хафа (для знаходження прямих ліній).

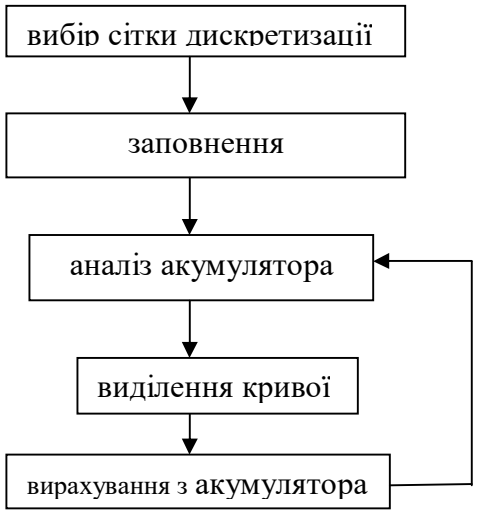
**Базове перетворення Хафа для пошуку об'єктів на площині**

Перетворення Хафа – метод з безпечного вилучення елементів із зображення, що використовується в обробці зображень і комп'ютерному зорі. Метод призначений для пошуку параметричних і непараметричних об'єктів з використанням процедури голосування. Застосовуючи процедуру голосування і заповнюючи акумуляторний масив, в кінці знайдемо в ньому максимум і за координатами цього максимуму зможемо відновити параметри досліджуваного об'єкта, а, отже, і сам об'єкт.

Ідея перетворення Хафа полягає в пошуку кривих, які проходять через достатню кількість точок інтересу [15]. Розглянемо сімейство кривих на площині, заданих параметричним рівнянням: $F(a_1, a_2, \ldots, a_n, x, y) = 0$, де $F$ – деяка функція, $a_1, a_2, \ldots, a_n$ – параметри сімейства кривих, $(x, y)$ – координати на площині.

Параметри сімейства кривих утворюють фазовий простір, кожна точка якого (конкретні значення параметрів $a_1, a_2, \ldots, a_n$) є певною кривою. З огляду на дискретність машинного уявлення і вхідних даних (зображення), потрібно перевести безперервний фазовий простір в дискретний простір. Для цього в фазовому просторі вводиться сітка, що розбиває його на осередки, кожен з яких відповідає набору кривих з близькими значеннями параметрів. Кожному осередку фазового простору можна поставити у відповідність число (лічильник), яке вказує кількість точок інтересу на зображенні, що належать хоча б одній з кривих, відповідних даному осередку. Аналіз лічильників осередків дозволяє знайти на зображенні криві, на яких лежить найбільша кількість точок інтересу [6].

Базовий алгоритм виділення кривих складається з наступних кроків:



Класичне перетворення Хафа є лінійним і застосовується для виявлення прямих. Пряма задається рівнянням $y = mx + b$ і може обчислюватися для будь-якої пари активних точок на зображенні *(x,y)*. Головна ідея перетворення Хафа – враховувати характеристики прямої в термінах її параметрів, тобто параметрів *m* і *b*. Таким чином, простір параметрів для ліній буде двомірним і складається з двох параметрів – кута нахилу *m* і точки перетину прямої з віссю *Oy, b*. Але для даного типу рівняння існує проблема завдання вертикальних прямих, тому що в цьому випадку отримуємо нескінченні значення параметрів *m* і *b*. Якщо ж представити пряму через параметри вектора, перпендикулярного цій прямій, який проходить через початок координат, то ця проблема зникне. Задаймо пряму через два параметра *R* і *θ*. *R* являє собою довжину зазначеного вектора, а θ – кут вектора до осі координат.

У цьому випадку рівняння прямої буде представлено в такому вигляді:

$$y = -\left(\frac{\cos\theta}{\sin\theta}\right) \cdot x + \left(\frac{R}{\sin\theta}\right).$$

Або ж воно може бути перетворено до наступного вигляду: $R = x\cos\theta + y\sin\theta$, де *R* – довжина перпендикуляра опущеного на пряму з початку координат; (*x,y*) – координати точки на прямій; θ – кут між перпендикуляром до прямої та віссю О*x* (рис. 1).

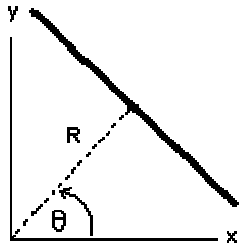



Рис. 1. Параметричне представлення прямої

Таким чином, кожна пряма на зображенні може бути представлена двома параметрами свого вектора нормалі – $R$ і $\theta$. Ці параметри будуть унікальними за умови $\theta\epsilon[0,\pi]$ і $R \in R$ або якщо $\theta\epsilon[0,2\pi]$ і $R \geq 0$. Створивши акумуляторний масив з певним кроком, зможемо заносити в нього значення заданих параметрів. Цей масив також називають простором Хафа для прямих на площині або просто накопичувальним простором. Через одну точку може проходити нескінченне число прямих ліній (рис. 2). Якщо ця точка має координати $(x_0, y_0)$, то всі прямі лінії, що проходять через неї, будуть мати наступне рівняння: $R(\theta) = x_0 \cos\theta + y_0 \sin\theta$, де *R* і $\theta$ можуть мати довільні значення з вищевказаного діапазону.

Це відповідає синусоїдальній кривій (рис. 3) в накопичувальному просторі *(R,θ)*, яка унікальна для кожної окремої точки. Синусоїди декількох точок накладаються одна на одну.

Точки їх перетину в просторі параметрів визначають параметри прямих *(R, θ)*, які проходять через точки, що задають синусоїди. Таким чином точки, які формують пряму лінію, визначають синусоїди, які перетинаються в одній точці накопичувального простору, яка задає параметри шуканої лінії. Задача пошуку прямій лінії, у кінці кінців, зводиться до задачі пошуку максимуму в накопичувальному просторі параметрів.

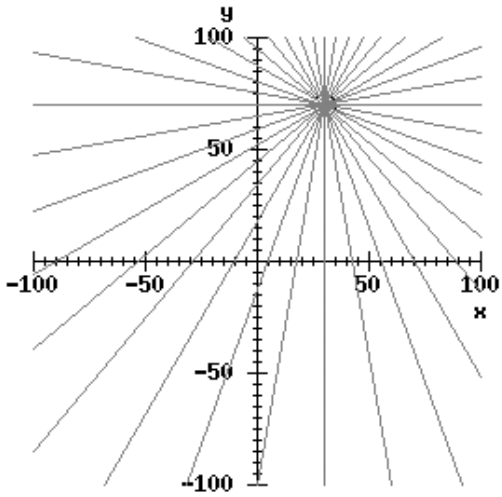



Рис. 2. Побудова сімейства кривих на площині

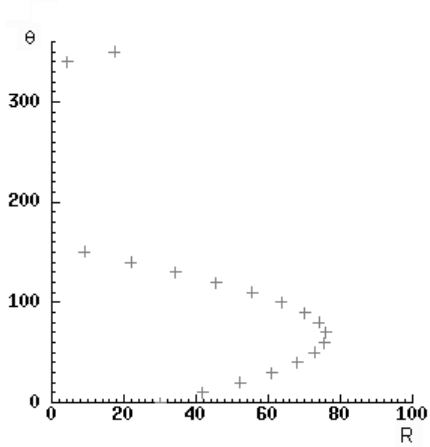



Рис. 3. Фазовий простір сімейства прямих

**Перетворення Хафа для пошуку ліній**

Нехай задано *N* активних точок на зображенні (точок по яких потрібно знайти пряму лінію).

Нехай є безліч плоских ліній *L(R,* $\theta$*),* які задаються в параметричному вигляді параметрами *R,* $\theta$. Перший параметр це довжина перпендикуляра від прямої до точки початку координат, другий – кут між прямою і віссю координат (нехай це буде вісь О$x$). Таким чином, підставивши ці параметри в параметричне рівняння, ми зможемо відновити вихідне рівняння прямої.

Параметри прямих утворюють двовимірний простір Хафа, по одній осі якого буде змінюватися параметр *R,* по іншій осі параметр $\theta$. Це простір нам необхідно зробити дискретним, для цього нам потрібно знати мінімальне і максимальне можливе значення радіуса *(R)* і кута *(*$\theta$*)*. Мінімальний радіус дорівнює нулю (пряма проходить через початок координат), максимальний – відстані від найдальшої активної точки зображення до початку координат. Мінімальний кут дорівнює 0 градусів, максимальний – $2\pi$ (або 360 градусів). Також визначимо розмірність нашої таблиці –

нехай по осі *R* вона дорівнює *k,* а по осі $\theta$ дорівнює *l.* Тоді масив прямих можна зобразити як $L_{kl}$. Значення радіусу в клітинці $L_{ij}$ дорівнює:

$R\ (L_{ij}) = R_{min} + dRi$, де $R_{min}$ – мінімальний радіус (дорівнює нулю), $dR = \frac{R_{max}-R_{min}}{k}$.

Значення кута в клітинці $L_{ij}$ дорівнює: $\theta(L_{ij}) = jd\theta$, де $d\theta = \frac{2\pi}{l}$.

Таким чином, якщо на прямій лінії із заданими параметрами *R* і $\theta$ лежить активна точка, то значення її клітинки збільшується на 1. Клітинка, значення якої найбільше, як раз і буде вказувати на параметри знайденої прямої.

Отже, у цьому розділі була проведена класифікація графічних примітивів, виявлені параметри, що характеризують той чи інший геометричний об'єкт. Розглянуто підходи до знаходження геометричних об’єктів, оцінені їх переваги та недоліки. Завершальним кроком алгоритму знаходження примітивів є застосування перетворення Хафа. Розглянуто базове перетворення Хафа для знаходження прямих ліній на зображенні. Проведено аналіз класичного методу на основі перетворення Хафа.

**Розробка алгоритму пошуку геометричних фігур на зображеннях**

**Алгоритм пошуку прямих на зображеннях**

Алгоритм Хафа для пошуку прямої лінії на зображенні буде виглядати так.

1. Дізнаємося максимальне значення віддаленості активних точок від початку координат – $R_{max}$ (довжина діагоналі зображення).
2. Задаємо розмірність простору Хафа для кута ($\theta$) і для радіуса (R).
3. Дізнаємося параметри $dR$ і $d\theta$.
4. Створюємо матрицю, в яку будемо заносити дані про кількість активних точок, що лежать на прямій з заданими параметрами – $L(R, \theta)$, розмірність дорівнює – $L_{kl}$.
5. Обнулити всі значення матриці $L(R, \theta)$.
6. Цикл по всім активним точкам зображення – $T_i(x_i, y_i)$:

Цикл по *j* від 0 до $2\pi$ з кроком $d\theta$:

$\theta = j \cdot d\theta;$

$R = x_i \cdot cos\theta + y_i \cdot sin\theta;$

$R_{пот} = R_{min};$

$k = 0;$

Поки $|R_{пот} - R| > \frac{dR}{2}$:

$R_{пот} = R_{пот} + dR;$

$k = k + 1;$

кінець Поки;

$L_{kj} = L_{kj}+1;$

Кінець циклу по *j*;

Кінець циклу по всіх точках.

7. Знайти точку $L_{ij}$ з максимальним значенням лічильника.
8. Відновити параметри прямої по клітинці $L_{ij}$ відповідно до формул:

$R(L_{ij}) = i \cdot dR;\ \theta(L_{ij}) = j \cdot d\theta;$

9. Вивести знайдені параметри $R$ і $\theta$.

**Пошук многокутників (полігонів) на зображенні**

Полігоном називається замкнута крива на площині, яка утворена відрізками прямих ліній. Полігон буде простим, якщо він не перетинає самого себе.

**Визначення точки перетину двох прямих ліній**

Нескінчена пряма лінія AB, яка проходить через дві точки A і B, може бути записана в параметричній формі

$$P(t) = A + t(B - A), \qquad (1)$$

де параметр $t$ може набувати будь – яке значення в діапазоні дійсних чисел (якщо значення $t$ обмежене діапазоном $0 \leq t \leq 1$, то рівняння (1) описує відрізок прямої лінії AB). Параметрична форма опису прямої лінії установлює відповідність між дійсними числами і точками на прямій лінії. На рис. 4 показані точки прямої лінії, які відповідають різним значенням параметра t.

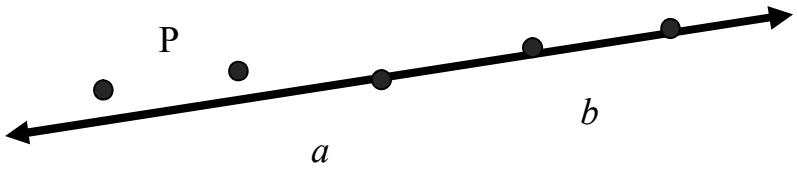



Рис. 4. Різні точки прямої лінії, що проходить через точки A і B

Для знаходження параметричного значення t використаємо скалярний добуток $ab$ для двох векторів $a(x_a, y_a)$ і $b(x_b, y_b)$, яке визначається як $ab = x_a x_b + y_a y_b$. Скалярний добуток має декілька важливих властивостей, що включають основні:

1. Якщо *a*, *b* – вектори, то *ab* = *ba*.
2. Якщо *a*, *b* і *c* – вектори, *a*(*b* + *c*) = *ab* + *ac* = (*b* + *c*)*a*.
3. Якщо *s* – скаляр, то (*sa*)*b* = *s*(*ab*) і *a*(*sb*) = *s*(*ab*).
4. Якщо *a* – нульовий вектор, то *aa* = 0, то інакше *aa* > 0.
5. $|a|^2 = aa.$

На основі цих основних властивостей отримаємо важливу властивість, на якій базується ряд операцій з прямими лініями: два вектори *a* і *b* перпендикулярні тоді і тільки тоді, коли їх скалярний добуток дорівнює нулю *ab* = 0. Щоб довести це твердження, зауважимо, що два вектори *a* і *b* перпендикулярні, якщо $|a - b| = |a + b|$.

Піднесемо до квадрату праву і ліву частини $(a - b)(a - b) = (a + b)(a + b)$. Враховуючи властивості 1–3, отримаємо $aa - 2ab + bb = aa + 2ab + bb$, скоротивши подібні доданки $4ab = 0, ab = 0$. Отже, умова $ab = 0$ буде виконуватися тоді, коли вектори $a$ і $b$ перпендикулярні. Якщо кут між векторами $a$ і $b$ менший за $90°$, то $|a - b| < |a + b|$, що еквівалентно умові $ab > 0$. Якщо кут між векторами більший $90°$, то $|a - b| > |a + b|$, $ab < 0$. Має місце наступна теорема (про скалярний добуток): Нехай $a$ і $b$ – вектори і $\theta$ – кут між ними. Тоді:

$$ab \begin{Bmatrix} > \\ = \\ < \end{Bmatrix} 0, \text{тоді і тільки тоді } \theta \begin{Bmatrix} > \\ = \\ < \end{Bmatrix} 90^{\circ}.$$

Теорему про скалярний добуток використаємо при знаходженні точок перетину двох прямих ліній AB і CD. Якщо пряма лінія AB описується рівнянням 2.1, то будемо шукати значення t таке, що прямі лінії AB і CD перетинаються у точці P(t). Оскільки вектор P(t) – C повинен співпадати з прямою лінією CD, то вектор P(t) – C і пряма лінія CD повинні бути перпендикулярними одному й тому ж вектору n. Отже, на основі теореми про скалярний добуток нам необхідно розв'язати відносно t рівняння

$$n( P(t) – C) = 0 \quad (2)$$

Оскільки P(t) = A + t(B – A), рівняння 2 можна переписати у вигляді

$$n((A + t (B – A)) – C) = 0.$$

Враховуючи основні властивості скалярного добутку, отримаємо n(A – C) + n(t(B – A)) = 0. Виведемо параметр t за дужки, отримаємо

$$n(A – C) + t[(n(B – A)] = 0. \text{ Звідси } t = -\frac{n(A-C)}{n(B-C)}, n(B - A) \neq 0. \quad (3)$$

Рівняння 3 виконується тоді і тільки тоді, якщо нескінчені прямі лінії AB і CD не паралельні і вони перетинаються у єдиній точці. Якщо дві прямі лінії паралельні або співпадають, то цей факт відповідає умові $n(B - A) = 0$, оскільки обидва вектори (B – A) і (D – C) перпендикулярні одному й тому ж вектору n.

**Виділення прямих за допомогою перетворення Хафа**

Монохромним зображенням вважається зображення, що складається з точок двох типів: фонових точок і точок інтересу. Завдання перетворення Хафа полягає у виділенні прямих ліній, утворених точками інтересу.

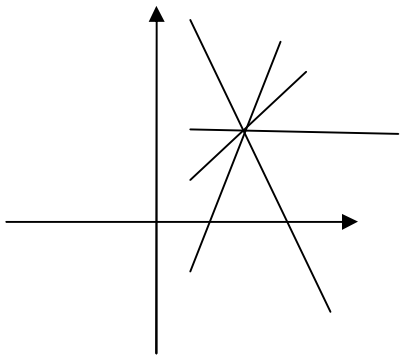


Рис.5. Прямі, які можна провести через точку зображення

Через кожну точку зображення (*x, y*) можливо провести декілька прямих з різними значеннями параметрів (*R,θ*) (рис. 2). Їх кількість обумовлена дискретністю представлення зображення та обраною в фазовому просторі сіткою. Отже, кожній точці зображення відповідає множина комірок в фазовому просторі (ця множина утворює синусоїду). Одночасно кожній точці H(i,j) в фазовому просторі відповідає пряма в просторі зображення.

Кожна комірка відповідає множині кривих з близькими значеннями параметрів. Кожній комірці фазового простору ставиться у відповідність лічильник, який показує кількість точок інтересу на зображенні, які лежать принаймні на одній прямій лінії, що відповідає комірці. Після проходу по всіх точках інтересу на зображенні аналіз

лічильників дозволяє знайти на зображенні прямі лінії, на які припадає найбільша кількість точок інтересу.

Кожній точці в фазовому просторі ставится у відповідність лічильник. На бінарному зображенні 0 – точки фону, 1 – точки інтересу. Комірці в фазовому просторі ($R,\theta$) відповідає набір прямих з близькими значеннями параметрів. Значення лічильника в комірці (комірка є квадратом в просторі параметрів прямих) $[\theta_i, \theta_{i+1}] \times [R_i, R_{i+1}]$ рівне кількості точок ($x$, $y$) на зображенні (рис. 6), що задовольняють умові: $x\cos\theta + y\sin\theta = R, \theta \epsilon [\theta_i, \theta_{i+1}], R \epsilon [R_i, R_{i+1}]$.

Розмір комірок варто вибирати, з огляду на наступні міркування. Якщо комірки будуть дуже великими, то за «пряму» може прийматися розрізнений набір точок. Якщо ж навпаки, комірки будуть занадто малі, є ймовірність, що ні одна пряма не знайдеться – всі лічильники будуть мати невелике значення.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | | |
| | | | | | | | |
| 1 | | | | | | | |
| 1 | 2 | 1 | | | | | |
| | | 1 | 1 | 1 | | | |
| | | | | 1 | 1 | | |
| | | | | | | | |

Рис. 6. Значення лічильника в комірці

Кількісний аналіз лічильників дозволяє знайти на зображенні прямі, на яких знаходиться найбільше точок інтересу (рис. 7). Значення параметрів (R,θ), в яких функція H(L(R,θ)) досягає локальних максимумів, є параметрами шуканих прямих.

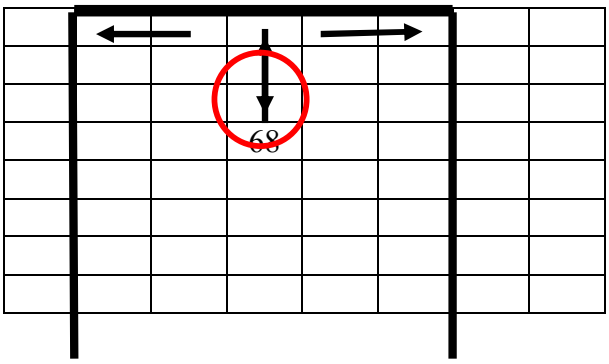



Рис. 7. Перевірка точки на локальний максимум у області радіусом 2

Чим більше значення H, тим яскравішою є відповідна точка на зображенні. Оскільки локальні максимуми можуть накладатися при малому радіусі околу точки інтересу, то маємо можливість збільшити радіус. На рис. 8 виділяються яскравіші точки, що відповідають найбільшому значенню лічильника, в області радіуса 2.

**Програмна реалізація знаходження многокутників на зображенні**

У ході роботи над даним дослідженням була написана програма, що виділяє геометричні об’єкти (полігони) на монохромному зображенні. Програма написана на мові Delphi у програмному середовищі Delphi.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | | |
| | | 35 | 33 | 34 | | | |
| | | 37 | 39 | 36 | 35 | 34 | |
| | | 37 | 38 | 33 | 32 | | |
| | | | 37 | 31 | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |

Рис. 8. Лічильник локального максимуму

1. Відкриваємо кольорове зображення. Для завантаження графічного зображення необхідно задати параметр, що вказує на місце знаходження та ім'я необхідного файлу. Так, зчитується графічне зображення з файлу pict_8.bmp і виводиться в область виведення ілюстрації зображення (рис. 9).

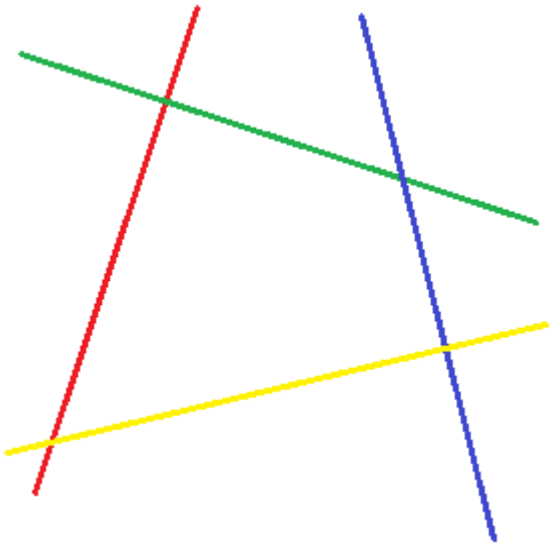


Рис. 9. Вхідне зображення

2. Перетворюємо у відтінки сірого. Спочатку зображення з кольорового переводиться у відтінки сірого (код G=0,21R+0,72G+0,07B). Це дозволяє скоротити обсяг використовуваної пам'яті і зберігати в три рази менше інформації про колір точки. Результат цього кроку продемонстровано на рисунку 10.

3. Перетворюємо у монохромне зображення. Підбираємо порогове значення так, щоб розділити зображення на об'єкт і фон. Об'єкт – сукупність, яскравість яких перевищує поріг. Для даного зображення візьмемо поріг – 20 (рис. 11).

4. Виконуємо перетворення Хафа (Hough Transform). Побудуємо синусоїдальні криві в накопичувальному просторі *(R,θ)* ($\theta \in [0;179^{\circ}], R \in \left[0;\sqrt{l^2+h^2}\right]$, де $l$ і $h$ – розміри зображення), які унікальні для кожної окремої точки. Синусоїди декількох точок накладаються одна на одну (рис. 12а). Точки перетину на рисунку зображені біль яскравішими точками.

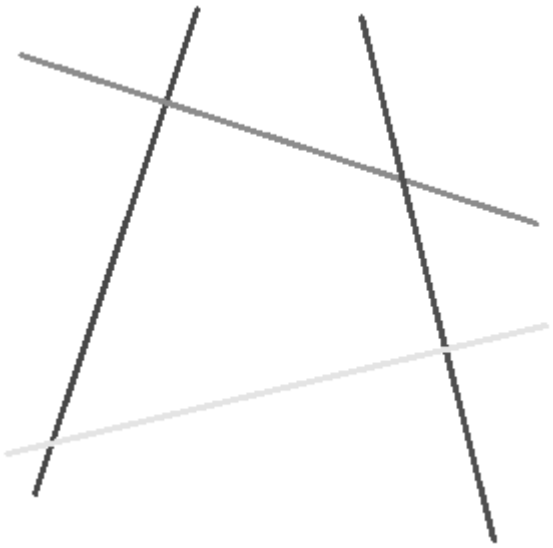


Рис. 10. Перехід до формату з відтінками сірого

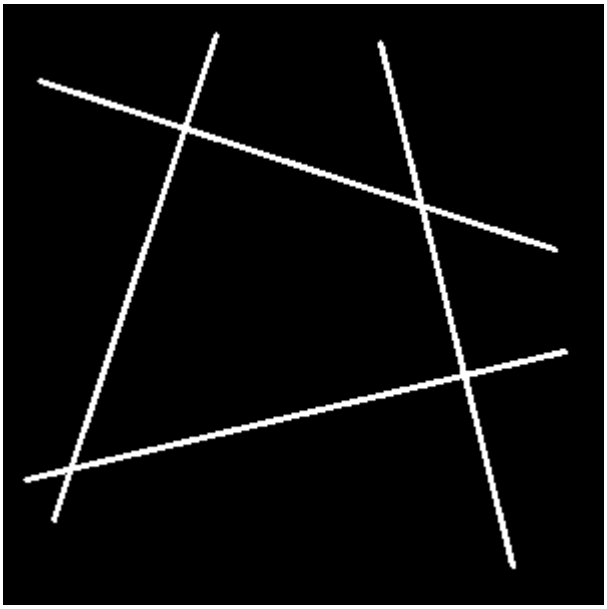
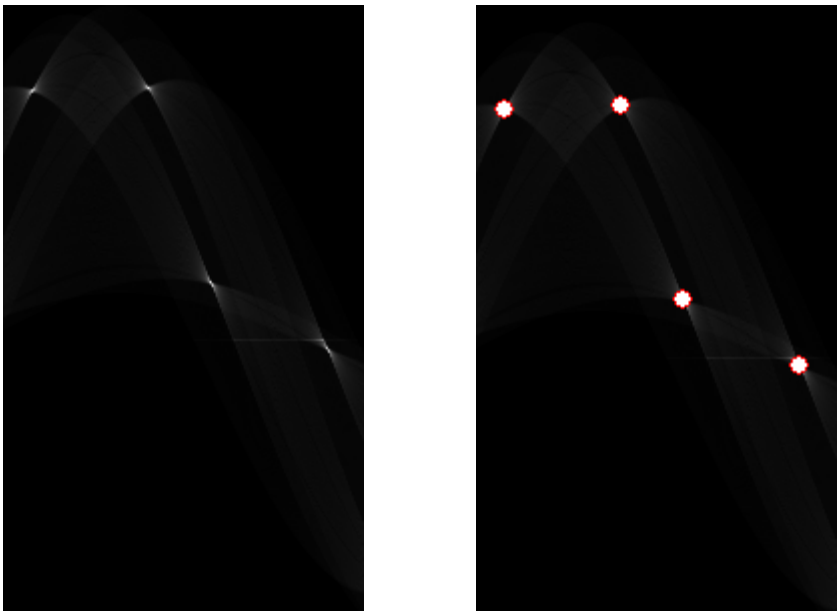


Рис. 11. Монохромне зображення

5. Знаходимо локальні максимуми (findLocalMaxs). Для виявлення прямих ліній введемо радіус пошуку локального максимуму та порогове значення. На зображені маємо чотири прямі, то підбираємо так радіус пошуку локальних максимумів, щоб їх було чотири. Радіус – 5, порогове значення – 70 (рис. 12б).



а б

Рис. 12. а – фазовий простір, б – можливі локальні максимуми

6. Для знайдених максимумів виконаємо зворотнє перетворення Хафа (inverseHoughTransform) (рис. 13).

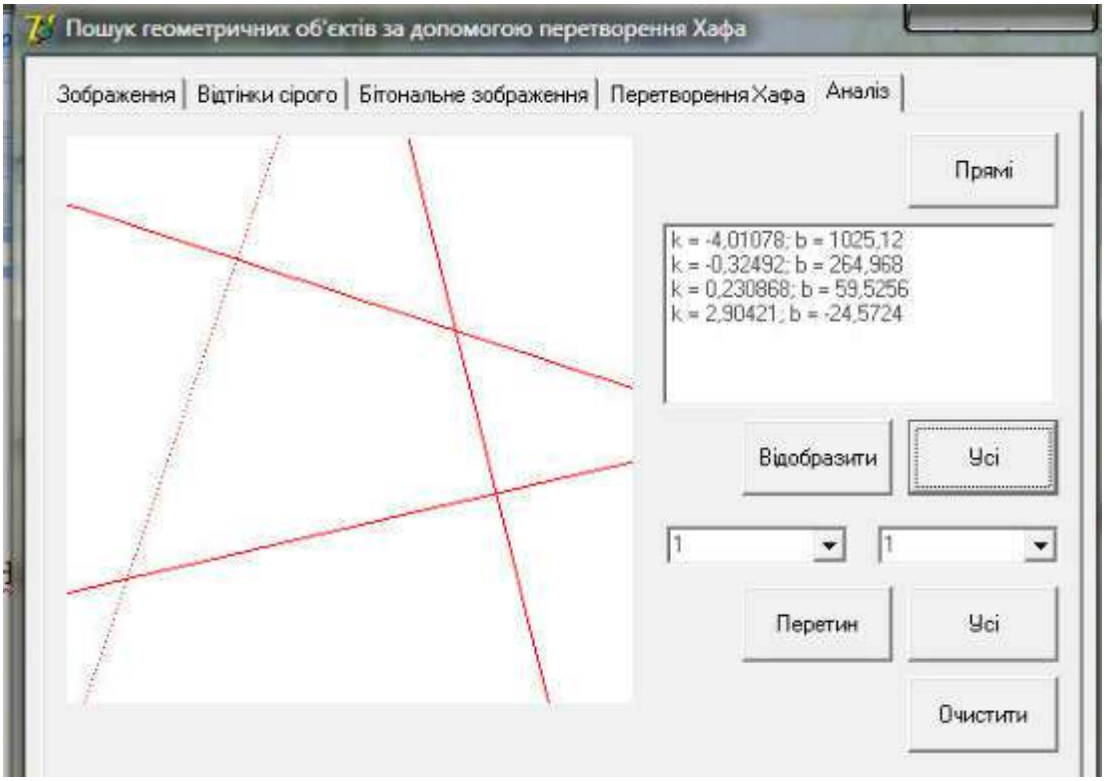



Рис. 13. Зворотнє перетворення Хафа

7. Шукаємо можливі точки перетину прямих (рис. 2.3.6). Розглянемо точки перетину пар прямих: 1 і 2, 1 і 3, 1 і 4, 2 і 3, 2 і 4, 3 і 4. У програмі здійснено аналіз перетину прямих на зображенні. Пряма 1 визначається точками A(0; 1025,12) , B(31,196; 0), пряма 2 – C(0; 264,968), D(815,487; 0), пряма 3 – E(0; 59,5256), F(– 257,834;

0), пряма 4 – M(0; – 24,5724), N(8,46; 0). Всі пари прямих ліній можливо мають точки перетину. Достатньо визначити параметричне значення точки перетину t(1,2) = 0,7; t(1,3) = 0,9; t(1,4) = 1,02; t(2,3) = 1,09; t(2,4) = 0,8; t(3,4) = 0,6. Для кожної точки перевіряємо, чи дійсно вона є на зображенні.

Враховуючи те, що

$$t \in \begin{cases} t < 0, & \text{точка на зображенні, але не існує;} \\ 0 \le t \le 1, & \text{точка на зображенні й існує;} \\ t > 1, & \text{точка за межами зображення;} \end{cases}$$

робимо висновок: точки перетину прямих ліній 1 і 2, 1 і 3, 2 і 4, 3 і 4 знаходяться на зображенні й існують, тобто являються точками перетину відрізків, що лежать на даних прямих. Знайдені відрізки об'єднуємо у фігури.

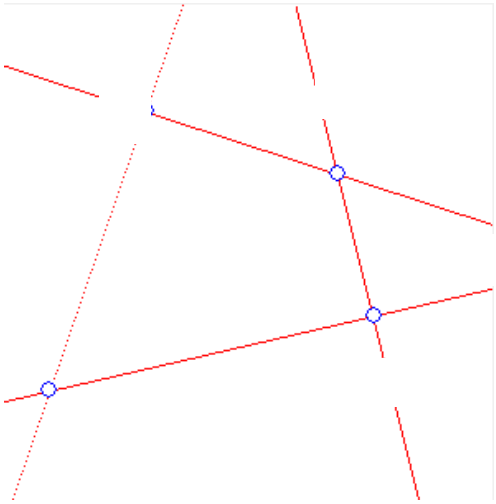


Рис. 14. Зображення точок, в яких перетинаються прямі

Отже, створений алгоритм перетворення Хафа для пошуку прямої лінії, описаний спосіб визначення точок перетину двох прямих ліній, обгрунтований метод виявлення прямих ліній на зображенні з допомогою перетворення Хафа.

**Висновки**

У даній роботі розглядалося перетворення Хафа як метод, за допомогою якого можна знаходити різні геометричні фігури, а також об'єкти довільної форми на зображенні. Цей метод здатний з досить великою точністю знаходити параметри, якими задаються шукані об'єкти на бінарному зображенні. Таким чином, після знаходження цих параметрів, ми можемо відновити об'єкт, який шукаємо. Для цього детально розглянуто базове перетворення Хафа і алгоритми реалізації пошуку прямих ліній на зображенні.

В ході дослідження даної проблеми обгрутовано визначення точок перетину прямих ліній та виявлення прямих на зображення з допомогою перетворення Хафа.

У даному дослідженні реалізована програма пошуку многокутників на зображенні.

**Список використаної літератури:**

1. Алгоритм Хафа для обнаружения произвольных фигур на изображениях [Електронний ресурс]. – Режим доступу: http://habrahabr.ru/blogs/algorithm/102948/.

2. Arefi H. Levels 43п detail 43п 3D building reconstruction from lidar data / [H. Arefi, J. Engels, M. Hahn, H. Mayer.]. – Stuttgart : Stuttgart University 43п Applied Sciences; Munich : Bundeswehr University Munich, 2008. – 490c.
3. Ballard D. H. Generalizing the Hough transform to detect arbitrary shapes / Ballard D. H. – New York : Computer Science Department, University 43п Rochester, 1980 – 122c.
4. Gonsales R., Woods R., Eddins S. Digital Image Processing using MATLAB. – New Jersey, USA: Prentice Hall, 2004. – 616 p.
5. Дегтярева А. Преобразование Хафа [Електронний ресурс]. – Режим доступу: http://www.cgm.computergraphics.ru/content/view/36.
6. Ecabert O., Thiran J. Adaptive Hough transform for the detection of natural shapes under weak affine transformation // Pattern Recognition Letters (25), No. 12, September 2004. – pp. 1411–1419.
7. James Matthews. Hough Transforms [Електронний ресурс]. – Режим доступу: http://www.generation5.org/content/2008.
8. Запрягаев С. А. Программная оболочка для поиска примитивов на изображении / С. А. Запрягаев, А. И. Сорокин // Вестник ВГУ, серия: системный анализ и информационные 43ппарат43и. – Воронеж, 2008. – № 2. – С. 37 – 47.
9. Использование преобразований Хафа (Hough transform) для выделения 43ппара на изображении [Електронний ресурс]. – Режим доступу: http://eddy-em.livejournal.com/932.html.
10. OpenCV шаг за шагом. Преобразование Хафа [Електронний ресурс]. – Режим доступу : http://robocraft.ru/blog/computervision....
11. Преобразование Хафа [Електронний ресурс]. – Режим доступу: http://ru.wikipedia.org/wiki....
12. Преобразования Хафа [Електронний ресурс]. – Режим доступу: http://locv.ru/wiki....
13. Сай И. С. Эффективность алгоритмов поиска оттиска печати в изображении документа / Сай И. С. // Вестник ТОГУ. – 2009. – № 4. – С. 53 – 60.
14. Samuel A. Inverso. Computer Vision: Ellipse Detection Using Randomized Hough Transform [Електронний ресурс]. – Режим доступу: http://www.saminverso.com/res/vision/.
15. Семенов А.Б. Обработка и анализ 43ппарат43ий с использованием 43ппар JAVA, курс 43ппара / Семенов А.Б. – Тверь: Тверской государственный 43ппарат43ий43, 2007. – 10 с.
16. Сойфер В.А. Методы компьютерной обработки 43ппарат43ий. – 2–е издание, исправленное. – М.: ФИЗМАТЛИТ, 2003. – 784 с.
17. The Hough Transform [Електронний ресурс]. – Режим доступу: http://homepages.inf.ed.ac.uk/rbf/CVonline....

**SERDIUK Oleksandr,**
The Bohdan Khmelnytsky National University of Cherkasy, PhD, Senior Lecturer
**KONOHRAI Anastasiya,**
student of 10-A class of Chornobay gymnasium of Chornobay district council of Cherkasy region

## SEARCH FOR GEOMETRIC FIGURES ON IMAGES TAKING INTO ACCOUNT THE HOUGH TRANSFORMATION

***Summary.*** *One of the most common data formats is digital images. One of the most difficult operations in image processing is finding and selecting the image or shape you want to find. Hough proposed a transformation that allows the one to translate the coordinates of the points of two-dimensional space in the data of the battery array with the subsequent application of the voting procedure. Research in the field of finding geometric objects in images is aimed at studying the Huff transformation, its application and finding methods for its optimization are very relevant.*

*The aim of the article is to develop an algorithm for finding polygons in an image using the Hough transform and to determine the features of the algorithm. In the process of work it is necessary to solve the following tasks:*

*1) to analyze the conditions under which it is possible to automate the selection of certain geometric shapes in the image;*

*2) consider the method of finding straight lines in the image using the Hough transform;*

*3) solve the problem of determining the point of intersection of two straight lines;*

*4) to develop own software implementation of search of polygons (polygons) on the image.*

*Having studied this topic, making practical research, we conclude that on the basis of the obtained theoretical results developed a mathematical model of finding geometric figures in the image.*

*Стаття надійшла 24.08.2018*
*Прийнято до друку 20.09.2018*

УДК 519.6:004.8

PACS 02.60.-x, 02.60.Pn, 02.70.Wz

**ІЛЬЯХОВА Наталія Олександрівна**
студентка Черкаського національного університету ім. Б. Хмельницького

**СЕРДЮК Олександр Анатолійович**
старший викладач кафедри прикладної математики та інформатики Черкаського національного університету ім. Б. Хмельницького

# МОДЕЛЮВАННЯ ДЕФОРМАЦІЇ ЕЛАСТИЧНОЇ ПОВЕРХНІ

*У роботі досліджено спосіб моделювання мімічних проявів емоцій на обличчі людини. Для цього розглянуто анатомічну будову обличчя людини, на основі якої реалізовано модель вираження емоцій людини. Наведено приклади вираження емоцій у вигляді малюнків, описано рухливі точки на обличчі. Розглянуто геометричну інтерпретацію поставленої задачі. Програмно реалізовано модель, у межах чого розроблено та реалізовано функції для побудови трикутної та прямокутної сітки, функцію виконання ітерацій. Для графічного відображення реалізовано функцію побудови точок і функцію побудови ліній з'єднання між точками. Також реалізовано графічний інтерфейс, за допомогою якого можна перевірити результати виконання функцій і побудувати довільну модель.*

***Ключові слова:*** *математичне моделювання, побудова алгоритму.*

## Вступ

Дослідження мімічного вираження емоцій почалося більш 100 років тому. Одним з перших виникло запитання: чому в людини в емоційному стані специфічно змінюється напруга різних лицьових м'язів? Міміка обумовлена уродженими механізмами. Звідси випливає, що мімічні реакції повинні бути тісно зв'язані з визначеними емоціями. Установлення таких зв'язків зробило б можливим розпізнавання емоцій по мімічному вираженню.

Метою роботи є моделювання деформації еластичної поверхні. Отримані результати можуть бути використані для подальшого моделювання інтелектуальної діяльності людини в системах штучного інтелекту як складові при розробці алгоритмів і програмно-апаратних засобів для систем комп'ютерного розпізнавання та відтворення зорових образів. Застосування даної реалізації дозволить створювати мультимедійні технології для використання в різних галузях наукової діяльності.

Для досягнення поставленої мети маємо розглянути як пов'язані емоції людини з програмною реалізацією необхідних алгоритмів. Для цього потрібно реалізувати такі алгоритми, як визначення рухомих та нерухомих точок на обличчі при вираженні емоцій, побудова сітки, її деформація для зміни положення точок. Дослідити, наскільки є ефективним даний алгоритм для його подальшого використання.

## Вираження емоцій людини

Міміка - це вираз обличчя, при якому людина видає свої внутрішні відчуття, переживання, почуття, настрій, емоції та інші душевні якості. Міміка обличчя людини проявляється по-різному, і найчастіше її можна побачити в людських емоціях.

Для визначення рухомих і нерухомих точок на обличчі розглянемо деякі прояви емоцій людини. Нерухомі точки позначимо червоним кольором, а нерухомі – зеленим.

*Радість, щастя*. Брови і губи розслаблені, куточки рота підняті з обох сторін, в куточках очей невеликі зморшки.

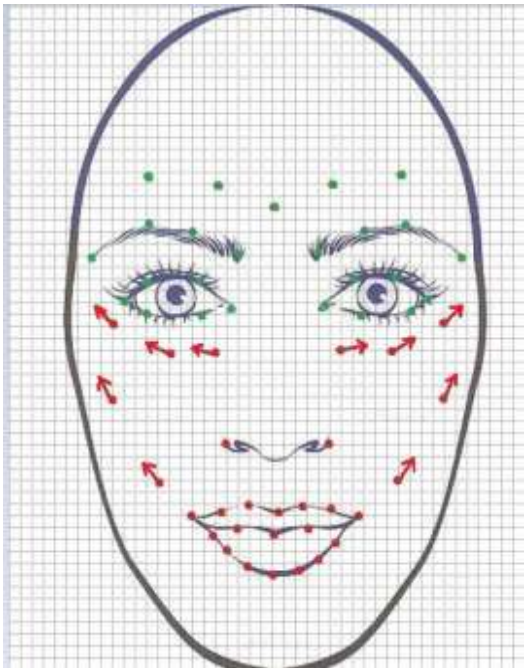


Рис. 1 Вираження радості, щастя

*Злість, роздратування*. Брови напружені, зведені разом і опущені, рот щільно закритий. Губи, куточки яких при гніві або сильному невдоволенні дивляться вниз.

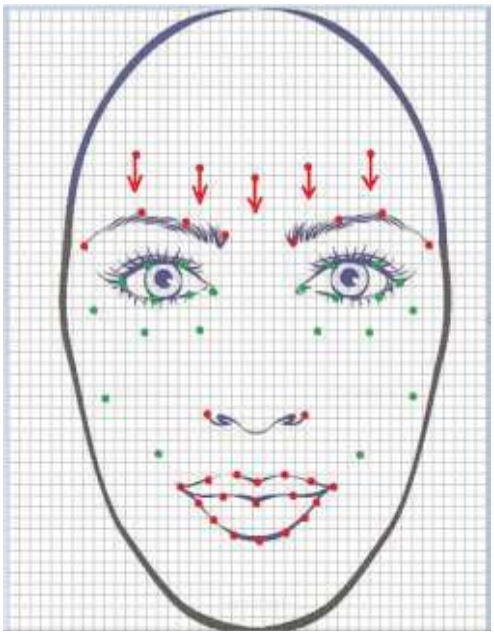


Рис. 2 Вираження злості, роздратування

*Подив.* Губи і обличчя в цілому розслаблені, очі округлі звичайного стану, брови вгнуті вгору, а рот відкритий.

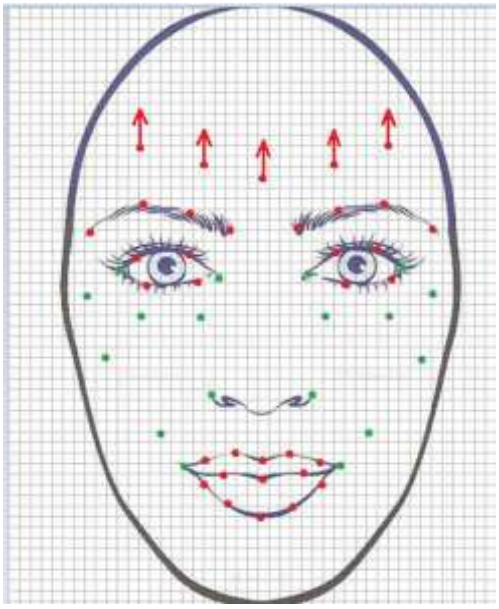


Рис. 3 Вираження подиву

*Печаль, прикрість.* Трохи опущені верхні повіки і підняті брови, розслаблені губи з куточками, що дивляться вниз.

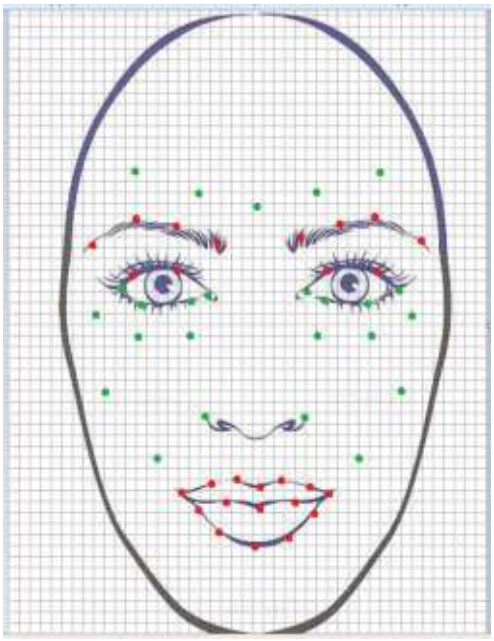


Рис. 4 Вираження печалі, прикрості

**Анатомічна будова обличчя**

Мімічні м'язи відрізняються від інших м'язів тіла тим, що, починаючись від кісток лицьового черепа, вплітаються в шкіру і мають, таким чином, рухливу точку прикріплення не на кістках, а в м'яких тканинах. Скорочення мімічних м'язів викликає зміщення шкіри, утворення складок і швидку зміну зморшок в різних поєднаннях.

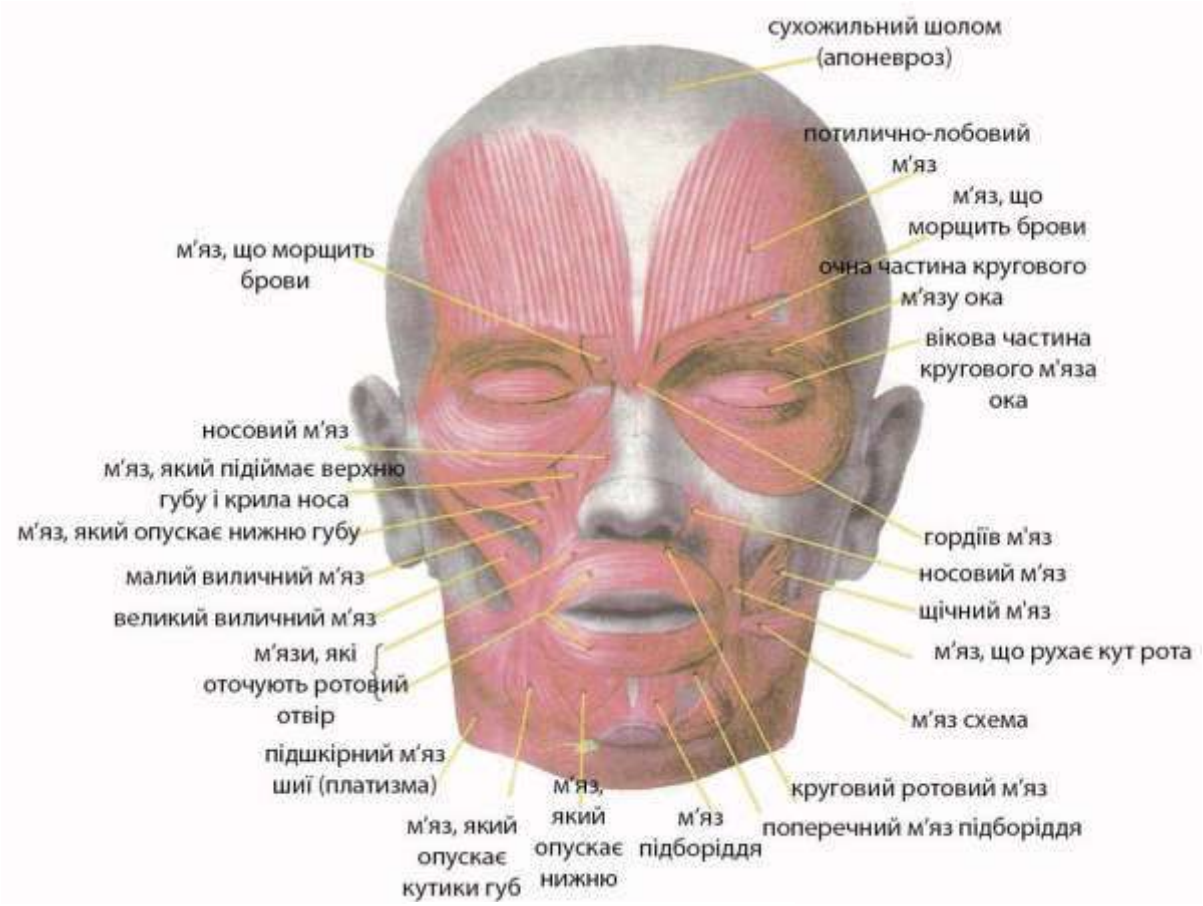



Рис. 5 М'язи обличчя та шиї

Мімічні м'язи розподілені на обличчі нерівномірно. Вони розташовуються групами навколо природних отворів - рота, очей, носа, - і обумовлюють форму і рух губ, ніздрів, повік.

Мімічні м'язи підпорядковані нервовим імпульсам, що йдуть з головного мозку по особовим нерву, і відображають найрізноманітніші емоційні стани людини.

Скорочення кожної мімічної м'язи викликає певну, відповідну їй складку на обличчі. Однак одиничне скорочення мімічних м'язів спостерігається рідко, зазвичай скорочується разом ціла група мімічних м'язів. Від численних комбінацій цих скорочень і залежить величезне розмаїття нашої міміки.

Наприклад, щокова велика і мала виличні м'язи піднімають куточки рота і формують посмішку, а м'яз, що опускає кут рота, створює гримасу невдоволення. Ці м'язи є антагоністами і не можуть скорочуватися одночасно. Більш того, під час напруги м'язи її антагоністи отримують гальмують імпульси, і їх тонус стає нижче середнього значення.

Важливо розуміти, що вираз обличчя формується не тільки зонами напруження, але і зонами розслаблення. Зазвичай в зоні затиску утворюються мімічні зморшки, а в зоні розслаблення - провисання тканин (деформація).

***Потилично-лобовий м'яз*** складається з потиличного і лобного черевця, з'єднаних апоневрозом, який отримав назву сухожильного шолома. Обидва черевця завжди скорочуються одночасно, при цьому лобне черевце розвинене сильніше, і його скорочення викликає на лобі поперечні хвилі. З віком, коли м'яз, які змикають повіки, слабшають, лобне черевце працює одночасно з ними, підтягуючи повіку і брови догори. Даний м'яз надає обличчю вираз уваги, зосередженості.

**Виведення алгоритму розрахунків**

Розглянемо систему з чотирьох точок, з'єднаних пружинами. Нехай точка $X$ з'єднана з точками $A_1$, $A_2$, $A_3$ так, як показано на рис. 6.

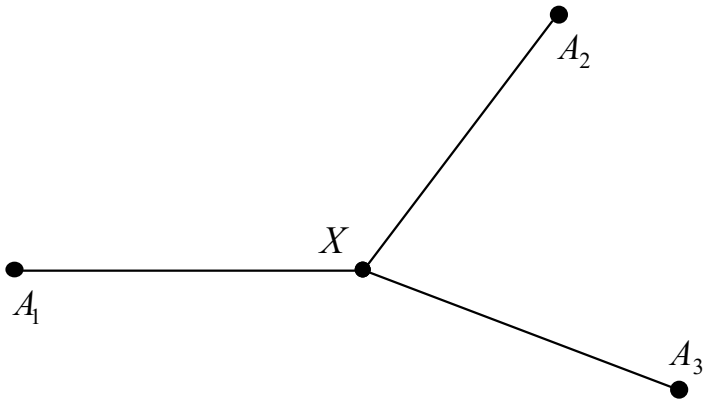



Рис. 6 Система з чотирьох точок

Загальну силу пружності, що діє на точку $X$, позначимо вектором $\vec{F}$, що є сумою векторів сил, з якими діють кожна з точок $A_1$, $A_2$, $A_3$ на $X$, тобто,

$$\vec{F} = \vec{F}_{A_1X} + \vec{F}_{A_2X} + \vec{F}_{A_3X}. \qquad (1)$$

Кожна з сил визначається за формулою сили пружності (закон Гука), тобто, $F = k\Delta x$, де $k$ - коефіцієнт жорсткості пружини, а $\Delta x$ - абсолютний розтяг (чи стиснення) пружини. $\Delta x$ визначається як різниця між довжиною пружини у спокійному (рівноважному) стані $L$ і поточною довжиною пружини $r$ (розтягнутої чи стиснутої), тобто, $\Delta x = L - r$. Таким чином, наприклад, у випадку, коли пружина розтягнута, тобто, $r > L$, $\Delta x = L - r < 0$ і вектор $F$ направлений у протилежний до розтягу бік, оскільки завжди $k > 0$ і напрям вектора сили, таким чином, залежить лише від знаку різниці $\Delta x$.

Визначимо нове положення точки в залежності від сил, що діють на неї. Для цього візьмемо Декартову систему координат і розглянемо взаємодію точок в ній. Оскільки нові координати чи зміщення точки залежать від зміщення вздовж осі абсцис $OX$ та ординат $OY$, необхідно розглянути довжини проекцій вектора сили на відповідні осі.

Відомо, що якщо маємо вектор $\vec{a}(a_x, a_y)$, то довжинами його проекцій будуть $|a_x|$ на вісь абсцис та $|a_y|$ на вісь ординат. Якщо ж брати до уваги ще й напрямок вектора, то самими проекціями на осі будуть його координати, $a_x$ та $a_y$.

Взявши позначення $\overrightarrow{A_1X}$ для $\vec{F}_{A_1X}$, отримаємо значення проекцій результуючого вектора сили в залежності від проекцій векторів сил кожної з пружинок (з рис. 6):

$$\vec{F}_x = \overrightarrow{A_1X}_x + \overrightarrow{A_2X}_x + \overrightarrow{A_3X}_x;\ \vec{F}_y = \overrightarrow{A_1X}_y + \overrightarrow{A_2X}_y + \overrightarrow{A_3X}_y. \qquad (2)$$

Очевидно, що формули (1) та (2) є еквівалентними, і векторна сума $\vec{F}_x + \vec{F}_y = \vec{F}$.

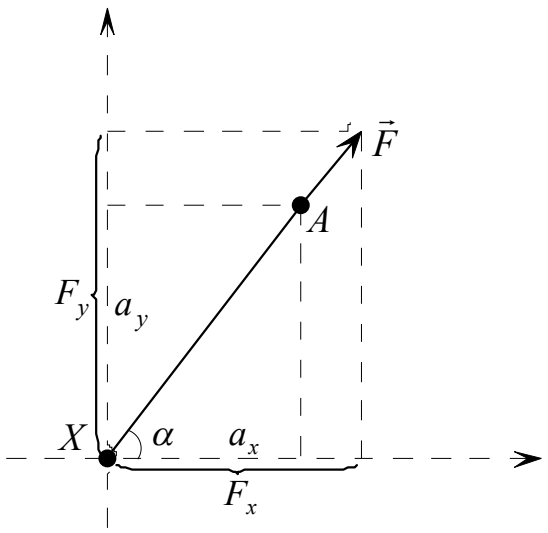



Рис. 7 Взаємодія між точками $X(x,y)$ та $A(a_x,a_y)$ силою $\vec{F}(F_x,F_y)$

Нехай маємо взаємодію між точками $X(x,y)$ та $A(a_x,a_y)$ силою $\vec{F}(F_x,F_y)$ (рис. 7). Проекція $F_x$ вектора $\vec{F}$ на вісь абсцис рівна добутку довжини вектора $\vec{F}$ на косинус кута між вектором та додатнім напрямом осі абсцис, тобто, $F_x = \left\|\vec{F}\right\|\cos\alpha$. Однак, зважаючи на те, що вектор лежить на прямій $XA$, кут нахилу прямої співпадає з кутом нахилу вектора, а тому $\cos\alpha = \dfrac{XA_x}{\|XA\|}$, де $XA_x$ - проекція вектора $XA$ на вісь абсцис, а $\|XA\|$ - довжина вектора. Тому, підставивши отримане значення у попередній вираз, матимемо:

$$F_x = \left\|\vec{F}\right\|\cos\alpha = \left\|\vec{F}\right\|\frac{XA_x}{\|XA\|} = \left\|\vec{F}\right\|\frac{a_x - x}{\sqrt{(a_x - x)^2 + (a_y - y)^2}}. \qquad (3)$$

Аналогічно отримаємо:

$$F_y = \left\|\vec{F}\right\|\sin\alpha = \left\|\vec{F}\right\|\frac{XA_y}{\|XA\|} = \left\|\vec{F}\right\|\frac{a_y - y}{\sqrt{(a_x - x)^2 + (a_y - y)^2}}. \qquad (4)$$

Об’єднавши формули (3), (4) та (2), можемо отримати загальну формулу для знаходження проекцій (враховуючи знаки, тобто, напрямки) результуючого вектора на осі Декартової системи координат.

За другим законом Ньютона, $\vec{F} = m\vec{a}$. Тоді, знаючи силу та масу матеріальної точки, можна визначити її прискорення, $\vec{a} = \dfrac{\vec{F}}{m}$, а знаючи проекції сили (вектори вздовж координатних осей), можна знайти й проекції прискорення, тобто

$$a_x = \frac{F_x}{m},\ a_y = \frac{F_y}{m}. \qquad (5)$$

Візьмемо систему без початкової швидкості та без інерції. Тоді для визначення переміщення можна використати формулу $\Delta S = at^2$. Підставивши у останню формулу почергово значення прискорень з (5), матимемо:

$$\Delta x = \frac{F_x}{m}t^2,\ \Delta y = \frac{F_y}{m}t^2. \qquad (6)$$

Отже, загалом можна скласти наступну задачу та її розв'язок. Маємо точку $X(x,y)$ та точки $A_i(a_x,a_y)$, з'єднані з точкою $X$ пружинками з коефіцієнтом жорсткості $k$ та довжиною $L$ у стані рівноваги без ніяких взаємодій.

Тоді, у випадку, коли на точку $X$ діють, зміщуючись, точки $A_i$, зміщення точки $X$ через час $t$ можна обрахувати наступним чином:

1) для кожної з точок $A_i$:

– знаходимо відстань до точки $X$: $r = \sqrt{(x-a_x)^2 + (y-a_y)^2}$;

– знаходимо силу взаємодії між $A_i$ та $X$: $F = k(L-r)$;

– знаходимо проекції вектора сили на координатні осі:

$$F_x = F\frac{a_x - x}{r},\ F_y = F\frac{a_y - y}{r}.$$

2) знаходимо суми: $F_X = \sum F_x$, $F_Y = \sum F_y$

3) знаходимо зміщення: $\Delta x = \frac{F_X}{m}t^2,\ \Delta y = \frac{F_Y}{m}t^2$.

В подальшому реалізуємо описану модель.

**Опис реалізованих функцій**

Для реалізації даного алгоритму використаємо математичний пакет Octave.

1. Опишемо функцію makeTriMesh, яка будує трикутну сітку. Передаємо вектори (набір наявних точок) X, Y, типи точок T та крок (відстань між точками) h.

Визначимо розмір вектора X та запишемо результат у змінну N.

```
N = length( X );
```

Горизонтальний крок сітки між точками у нас дорівнює h, тоді вертикальний крок за формулою $\sqrt{\mathbf{h^2 - \frac{h^2}{4}}} = \mathbf{\frac{\sqrt{3}}{2}h}$ :

```
vh = sqrt( 3 ) / 2 * h;
```

Для заданих точок, за допомогою вбудованих функцій DelaunayTri та convexHull визначаємо випуклу оболонку, знаходимо індекси точок, які утворюють випуклу оболонку та записуємо у відповідні змінні.

```
dt = DelaunayTri( X,Y );
k = convexHull( dt );
```

```
chOnX = X( k );
chOnY = Y( k );
chOnT = T( k );
chOnN = length( chOnX );
```

Далі нам необхідно вибрати усі точки, які знаходяться всередині випуклої оболонки та записати їх індекси у відповідні змінні.

```
k = setdiff( 1:N,k );
chInX = X( k );
chInY = Y( k );
chInT = T( k );
chInN = length( chInX );
```

Переходимо безпосередньо до побудови сітки. Необхідно визначити прямокутник для генерації сітки. Для цього знаходимо мінімальні і максимальні точки в X та Y.

```
x0 = min( X );
x1 = max( X );
y0 = min( Y );
y1 = max( Y );
```

Так як необхідно побудувати трикутну сітку, запишемо для першого та другого рядка у вигляді векторів абсциси і ординати точок з кроком h та vh відповідно.

```
% абсциси першого рядка
xx1 = x0-h:h:x1+h;
% абсциси другого рядка
xx2 = x0-h/2:h:x1+h/2;
% ординати першого стовпчика
yy1 = y0-vh:2*vh:y1+vh;
% ординати другого стовпчика
yy2 = y0:2*vh:y1;
```

Так як сітка у нас з більшою кількістю точок, то потрібно продублювати рядки, щоб отримати матрицю необхідної нам розмірності.

```
% матриці координат точок непарних рядків
X1 = repmat( xx1,length(yy1),1 );
Y1 = repmat( yy1',1,length(xx1) );
% матриці координат точок парних рядків
X2 = repmat( xx2,length(yy2),1 );
Y2 = repmat( yy2',1,length(xx2) );
```

Утворимо лінійні масиви координат.

```
X3 = [ reshape( X1,numel(X1),1 ); reshape( X2,numel(X2),1 ) ];
Y3 = [ reshape( Y1,numel(Y1),1 ); reshape( Y2,numel(Y2),1 ) ];
```

Визначаємо та вибираємо точки сітки, що належать випуклій оболонці за допомогою вбудованих функцій find та inpolygon.

```
in = find( inpolygon( X3,Y3,chOnX,chOnY ) );
```

```
meshX = X3( in );
meshY = Y3( in );
meshN = length( meshX );
```

Для того, щоб точки побудованої сітки та задані вручну точки не знаходилися дуже близько, необхідно знайти та вилучити ті точки, які знаходяться дуже близько до наявних точок моделі.

```
modelX = zeros( meshN,1 );
modelY = modelX;
idx = 0;
% проходимо по усіх точках сітки
for i = 1:meshN
    % знаходимо точки, що лежать ближче за половину вістані сітки до
    % поточної точки
    iii = find( ( abs( chInX - meshX( i ) ) < h/2 ) & ...
                ( abs( chInY - meshY( i ) ) < h/2 ),1 );
    % якщо таких точок немає - додаємо поточну точку до допустимих
    if( isempty( iii ) )
        idx = idx + 1;
        modelX( idx ) = meshX( i );
        modelY( idx ) = meshY( i );
    end
end
modelX = modelX( 1:idx,1 );
modelY = modelY( 1:idx,1 );
modelN = length( modelX );
```

Далі об’єднуємо знайдені точки з точками що лежать всередині випуклої оболонки.

```
modelX = [ modelX; chInX ];
modelY = [ modelY; chInY ];
modelT = [ ones( modelN,1 ); chInT ];
modelN = modelN + chInN;
```

Триангулюємо отриманий набір точок та створюємо розріджену матрицю суміжності, для визначення з’єднаних точок.

```
dt = DelaunayTri( modelX,modelY );
M = sparse( chOnN+modelN-1,chOnN+modelN-1 );
```

Потім проходимо по всім тріангульованим точкам та визначаємо їх сусідів.

```
for i = 1:size( dt.Triangulation,1 )
    idx = dt.Triangulation( i,: );
    M( idx( 1 ),idx( 2 ) ) = 1;
    M( idx( 2 ),idx( 1 ) ) = 1;
    M( idx( 2 ),idx( 3 ) ) = 1;
    M( idx( 3 ),idx( 2 ) ) = 1;
    M( idx( 1 ),idx( 3 ) ) = 1;
    M( idx( 3 ),idx( 1 ) ) = 1;
end
```

Необхідно побудувати випуклу оболонку для триангульованих точок, щоб з'єднати з точками існуючої випуклої оболонки.

```
k2 = convexHull( dt );
```

Вилучаємо останню точку, яка дублює першу, з основної випуклої оболонки.

```
chOnX = chOnX ( 1:end-1,1 );
chOnY = chOnY ( 1:end-1,1 );
chOnT = chOnT ( 1:end-1,1 );
chOnN = length( chOnX );
```

Проходимо по точках випуклої оболонки та визначаємо їх сусідів.

```
for i = 1:chOnN
    % запис сусідів у випуклій оболонці
    if( i == chOnN )
        M( i+modelN,modelN+1 ) = 1;
        M( 1+modelN,modelN+i ) = 1;
    else
        M( i+modelN,i+1+modelN ) = 1;
        M( i+1+modelN,i+modelN ) = 1;
    end
    % пошук сусідніх точок з поточною
    idx = find( ( abs( chOnX( i ) - modelX( k2 ) ) <= h ) & ...
                ( abs( chOnY( i ) - modelY( k2 ) ) <= h ) );
    M( i+modelN,k2( idx ) ) = 1;
    M( k2( idx ),i+modelN ) = 1;
End
```

Об'єднуємо знайдені точки з точками моделі.

```
Xres = [ modelX; chOnX ];
Yres = [ modelY; chOnY ];
Tres = [ modelT; chOnT ];
```

Створюємо списки сусідів за допомогою вбудованої функції cell.

```
nbrs = cell( length( Xres ),1 );
for i = 1:size( M,1 )
    nbrs{ i } = find( M( i,: ) );
end
```

2. Опишемо функцію doOneIteration, яка виконує одну ітерацію деформації сітки, тобто зміщення точок. На вхід функція приймає такі параметри:k – коефіцієнт жорсткості, L – довжина рівноваги, m - масу частинки, t – крок по часу.

Оголошуємо глобальні змінні, що містять дані розробленої моделі.

```
global gblPOINTS;
global gblNBRS;
```

Вибираємо окремо координати наявних точок.

```
X = gblPOINTS( :,1 );
Y = gblPOINTS( :,2 );
pType = gblPOINTS( :,3 );
```

Визначимо розмір вектора X та запишемо результат у змінну N.

```
N = length( X );
```

Створимо вектор-стовпчики нулів dX та dY розмірності $N \times 1$ .

```
dX = zeros( N,1 );
dY = zeros( N,1 );
```

Виконання одного кроку моделі полягає у почерговій обробці усіх точок, що реалізовано у циклі for:

```
for i = 1:N
```

У тілі циклу виконуються наступні кроки. Для поточної точки:

1) Виписуємо її координати у змінні srcX та srcY, а також індекси її сусідів у змінну nbrs.

```
    if( pType( i ) ~= 1 ), continue; end
    srcX = X( i );
    srcY = Y( i );
    nbrs = gblNBRS{ i };
```

2) Знаходимо відстані до сусідів.

```
    r = sqrt( ( srcX - X( nbrs ) ).^2 + ( srcY - Y( nbrs ) ).^2 );
```

3) Знаходимо силу взаємодії між поточною точкою та її сусідами.

```
    F = -k * ( L - r );
```

4) Знаходимо проекції вектора сили на координатні осі.

```
    Fx = F .* ( X( nbrs ) - srcX ) ./ r;
    Fy = F .* ( Y( nbrs ) - srcY ) ./ r;
```

5) Розраховуємо суми проекцій і зберігаємо їх у відповідних елементах векторів dX та dY.

```
    dX( i ) = sum( Fx );
    dY( i ) = sum( Fy );
```

6) Перераховуємо суми проекцій сил у зміщення точок на площині.

```
    dX = dX / m*t*t;
    dY = dY / m*t*t;
```

7) Знаходимо нові координати точок і зберігаємо їх у змінній gblPOINTS.

```
    gblPOINTS( :,1 ) = gblPOINTS( :,1 ) + dX;
    gblPOINTS( :,2 ) = gblPOINTS( :,2 ) + dY;
```

8) Обраховуємо загальне зміщення усіх точок моделі.

```
    err = sum( abs( dX ) ) + sum( abs( dY ) );
```

3. Опишемо функцію drawPoints, яка малює точки на площині.

Оголошуємо глобальні змінні, що містять дані розробленої моделі.

```
global gblPOINTS;
global PPOINTS;
global CONST;
```

Задаємо параметри для наявних точок. Маємо вказівники на графіки, що відображують три типи точок: PPOINTS(1) – звичайна точка ,має чорний колір; PPOINTS(2) - нерухома точка, має червоний колір; PPOINTS(3) – рухома точка, має синій колір.

```
if( isempty( gblPOINTS ) )
    set( PPOINTS( 1 ),'XData',[], 'YData',[] );
    set( PPOINTS( 2 ),'XData',[], 'YData',[] );
    set( PPOINTS( 3 ),'XData',[], 'YData',[] );
    return;
end
```

Спочатку відображуємо рухомі (чорні) точки.

Вибираємо константу, що відповідає типу рухомих точок і знаходимо індекси потрібних точок, що розміщуються у масиві gblPOINTS. Знайдені індекси поміщаємо у змінну idx. Якщо точок шуканого типу немає (змінна idx пуста), то встановлюємо в якості значень властивостей XData та YData для відповідного графіка пусті множини (тобто, поточний графік точок на даний момент не має). Якщо ж такі точки існують, то вибираємо окремо їх абсциси та ординати і записуємо у відповідні властивості графіка.

```
   i = CONST.NORMAL;
idx = find( gblPOINTS( :,3 ) == i );
if( isempty( idx ) )
    set( PPOINTS( i ), 'XData',[], 'YData',[] );
else
    set( PPOINTS( i ),'XData',gblPOINTS( idx,1 ), ...
                      'YData',gblPOINTS( idx,2 ) );
end
```

Аналогічні дії виконуємо для точок інших типів.

```
i = CONST.STABLE;
idx = find( gblPOINTS( :,3 ) == i );
if( isempty( idx ) )
    set( PPOINTS( i ), 'XData',[], 'YData',[] );
else
    set( PPOINTS( i ),'XData',gblPOINTS( idx,1 ), ...
                      'YData',gblPOINTS( idx,2 ) );
```

```
end
 i = CONST.MOVING;
idx = find( gblPOINTS( :,3 ) == i );
if( isempty( idx ) )
    set( PPOINTS( i ), 'XData',[], 'YData',[] );
else
    set( PPOINTS( i ),'XData',gblPOINTS( idx,1 ), ...
                      'YData',gblPOINTS( idx,2 ) );
end
```

4. Опишемо функцію drawMeshLines, яка малює лінії, що з'єднують точки.

Оголошуємо глобальні змінні, що містять дані розробленої моделі.

```
global gblPOINTS;
global gblNBRS;
global PMESH;
```

Створюємо пустий масив для подальшого заповнення значень точок.

```
X = [];
Y = [];
```

Заповнюємо масив значеннями точок, які необхідно з'єднати між собою.

```
for i = 1:length( gblNBRS )
    srcX = gblPOINTS( i,1 );
    srcY = gblPOINTS( i,2 );
    idx = gblNBRS{ i };
    for j = 1:length( idx )
        X = [ X srcX gblPOINTS( idx( j ),1 ) NaN ];
        Y = [ Y srcY gblPOINTS( idx( j ),2 ) NaN ];
    end
end
```

Відкидаємо останній елемент, який має значення NaN.

```
X( end ) = [];
Y( end ) = [];
```

З'єднуємо точки лінією.

```
set( PMESH, 'XData',X, 'YData',Y );
```

**Результати виконання алгоритму**

У командному вікні інструментального середовища викликаємо функцію meshModel, яку будемо використовувати для графічної побудови моделі.

У даному вікні можна обирати «Режим» (тип точки), «Перегляд», задавати параметри зміщення точок та перевіряти показники роботи. Для виконання деформації натискаємо кнопку «1 ітерація» для виконання однієї ітерації алгоритму відповідно, або кнопку «Виконати» для виконання 100 ітерацій алгоритму.

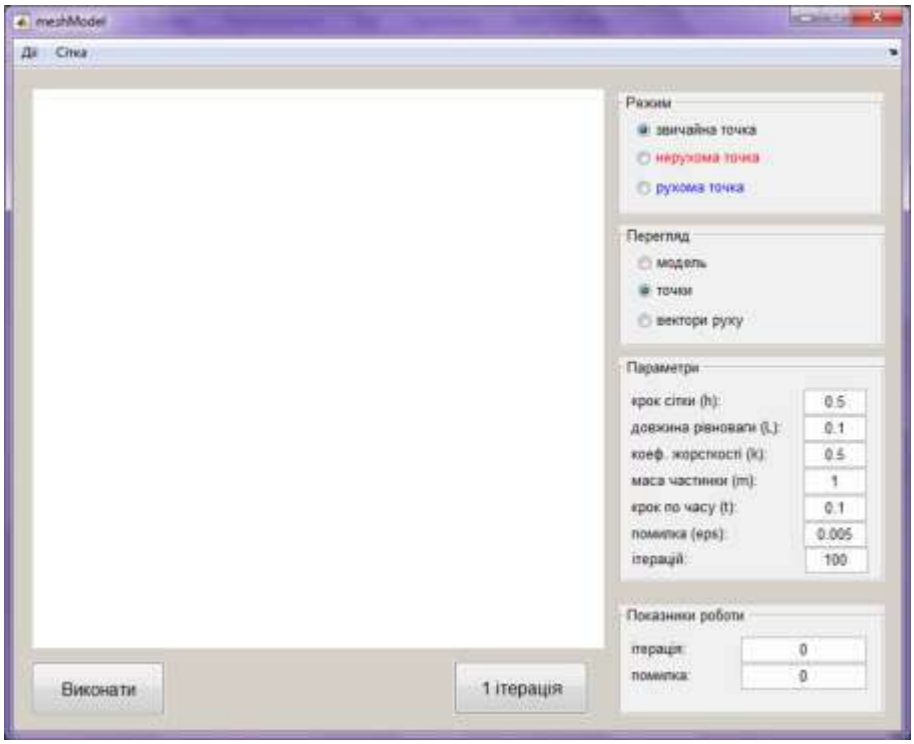



Рис. 8 Вікно для побудови

Для побудови довільної моделі нанесемо на область три види точок – звичайні, нерухомі, рухому(в довільному порядку). Для того, щоб видалити точку, необхідно натиснути на неї правою кнопкою миші. Щоб змінити тип точки, достатньо обрати інший тип і клацнути по точці, яку потрібно змінити.

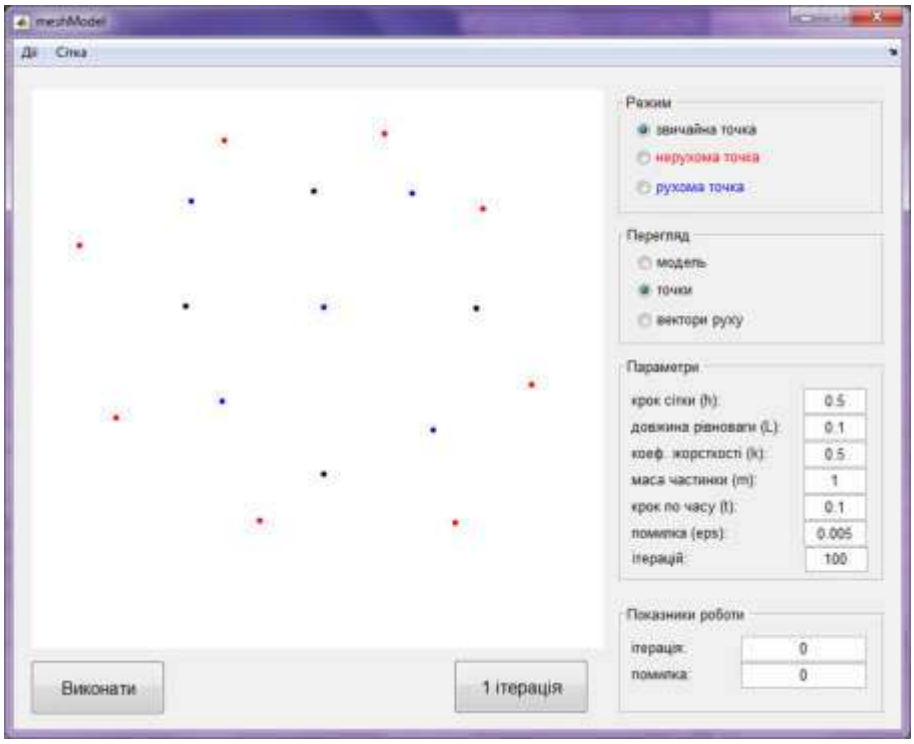



Рис. 9 Нанесення точок на площину

Далі побудуємо трикутну сітку. В меню «Сітка» обираємо пункт «Трикутна» і будуємо.

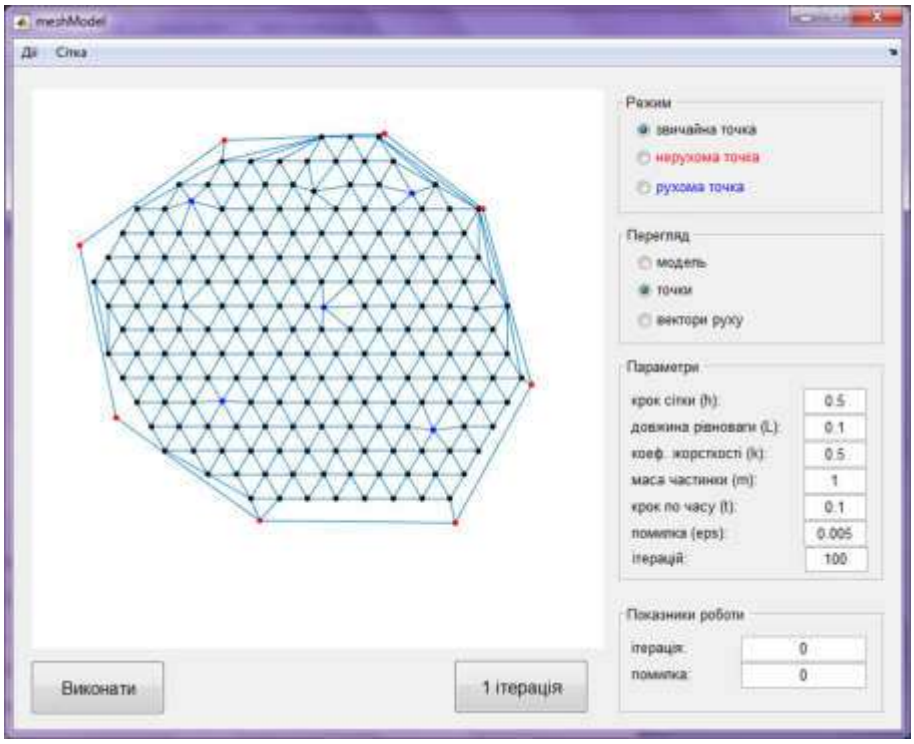



Рис. 10 Побудова трикутної сітки

Тепер спробуємо виконати деформацію сітки, виконавши одну ітерацію.

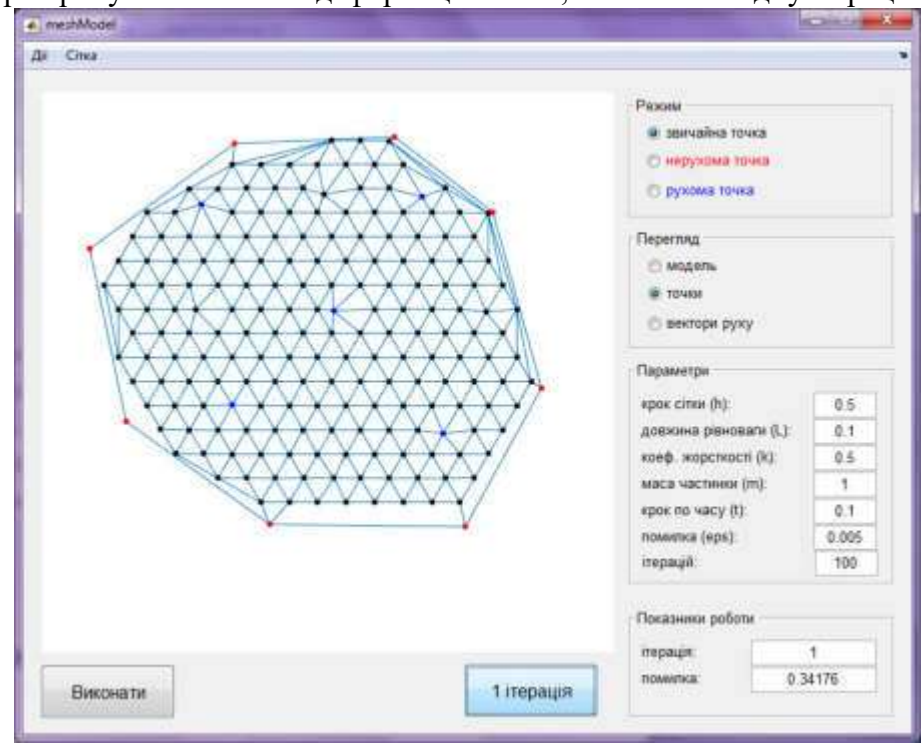



Рис. 11 Виконання однієї ітерації алгоритму

Як бачимо, помітних змін не відбулося, але точки все одно змістилися, сітка деформувалася. Для того, щоб побачити як будуть зміщуватися точки, виконаємо більше ітерацій, натиснувши декілька разів на кнопку «Виконати» і отримаємо результат виконання алгоритму.

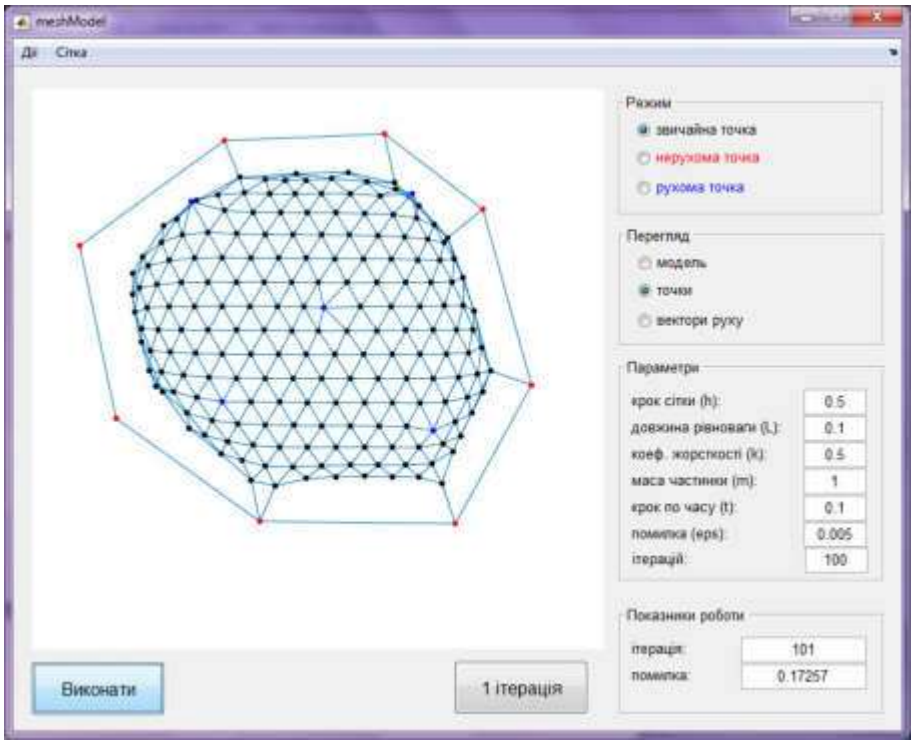



Рис. 12 Деформація сітки після 101 ітерації

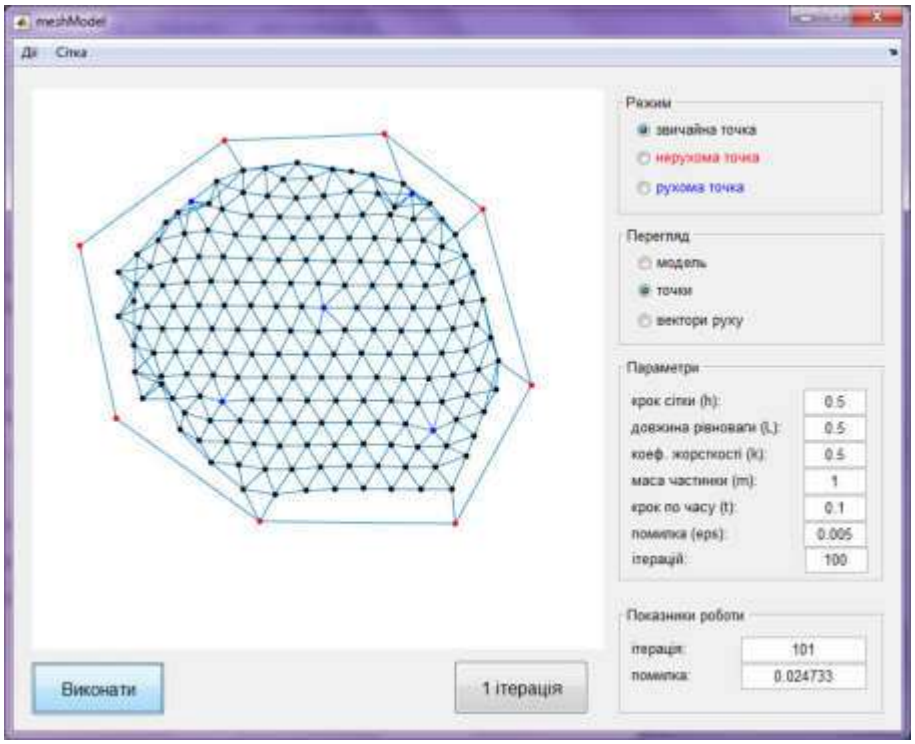



Рис. 13 Деформації сітки після зміни параметру

Отже, можна бачити, що сітка стискається всередину, нерухомі точки залишаються на місці, а рухомі «відтягують» пружинки, які з'єднують між собою інші точки сітки.

Тепер спробуємо повернути сітку на вихідне положення. Для цього змінимо значення довжини рівноваги на значення 0.5 і спробуємо виконати декілька ітерацій (рис. 13).

Отже, бачимо, що сітка поступово розтягується і вертає точки у вихідне положення.

Таким чином, можна стверджувати, що реалізований алгоритм працює правильно, виконує поставлену задачу – моделює деформацію еластичної поверхні. В подальшому цей алгоритм можна використовувати для моделювання різних проявів емоцій людини, застосовуючи його до графічних зображень.

**Висновки**

У роботі поставленою задачею було дослідити мімічні прояви емоцій на обличчі людини. Для цього було розглянуто анатомічну будову обличчя людини, на основі якої було реалізовано в подальшому модель вираження емоцій людини в математичному пакеті Matlab. Наведено приклади вираження емоцій у вигляді малюнків, описано рухливі точки на обличчі.

Також було розглянуто геометричну інтерпретацію поставленої задачі, в якій розглянуто основні геометричні властивості векторів та фізичні закони.

Програмно було реалізовано модель у вигляді декількох функцій. Розроблено та реалізовано функції для побудови трикутної та прямокутної сітки, в якій було враховано такі параметри як, сила пружності, сила взаємодії між точками, проекції вектора сили та прискорення. Також було розроблено функцію виконання ітерацій, в якій демонструється деформація сітки – зміщення точок, стиснення та розтяг.

Для графічної побудови реалізовано функцію побудови точок, задано параметри для них і функцію побудови ліній з'єднання між точками. Також було реалізовано функцію інтерфейсу, в якій можна перевірити результати виконання функцій і побудувати довільну модель.

**Список використаної літератури:**

1. Кетков Ю.Л., Кетков А.Ю., Шульц М.М. MATLAB 7: программирование, численные методы. – СПб.: БХВ-Петербург, 2005 – 752 с.: ил.
2. Майкл Ласло. Вычислительная геометрия и компьютерная графика на C++: Пер. С англ.. – М.: «Издательство БИНОМ», 1997. – 304 с.: ил.
3. Варій, М. Й. Загальна психологія [Текст] : підруч. для студ. вищ. навч. закл. - 3-тє вид. І Варій М. Й. - К. і Центр учбової л-ри, 2009. - 1007 с.
4. М'язи обличчя та шиї. [http://sainua.com/myazy-oblychchya-ta-shyyi]

**ILYAKHOVA Nataliya,**
student, The Bohdan Khmelnytsky National University of Cherkasy
**SERDIUK Oleksandr,**
The Bohdan Khmelnytsky National University of Cherkasy, PhD, Senior Lecturer

## MODELING OF ELASTIC SURFACE DEFORMATION

*The method of modeling facial expressions of emotions on a person's face is investigated in the work. For this purpose the anatomical structure of the human face is considered, on the basis of which the model of expression of human emotions is developed. Examples of expression of emotions in the form of drawings are given, moving points on the face are described. The geometric interpretation of the problem is considered. The model is implemented programmatically, within which functions for construction of triangular and rectangular grid, function of iterations execution are developed and*

*realized. For graphical visualisation, the function of constructing points and the function of constructing connecting lines between points are implemented. There is also a graphical interface with which the one can check the results of functions and build an arbitrary model.*

***Keywords:*** *mathematical modeling, algorithm construction.*

*Стаття надійшла 21.02.2018*
*Прийнято до друку 20.09.2018*

# СЕКЦІЯ «ІНФОРМАТИКА»

УДК 004.4, 004.5

PACS 07.05.Bx, 07.05.Wr

**ІГНАТЕНКО Ігор Олександрович**
студент Черкаського національного університету ім. Б. Хмельницького
e-mail: igignt@gmail.com

**СЕРДЮК Олександр Анатолійович**
старший викладач кафедри прикладної математики та інформатики Черкаського національного університету ім. Б. Хмельницького

## РОЗРОБКА ОСВІТНЬОГО ПОРТАЛУ ДЛЯ ПЕРЕВІРКИ КОМП'ЮТЕРНИХ ПРОГРАМ НАПИСАНИХ СТУДЕНТАМИ

*У роботі розглянуто методи та засоби створення повноцінного вебсайту з базою даних, клієнтською та серверною частиною. Наразі онлайн навчання широко використовується в навчальних закладах та у сфері IT. Coursera, EDX, Udemy, Udacity є провідними ресурсами онлайн навчання у світі. Варто також згадати український проект масових відкритих онлайн курсів Prometheus та освітню платформу Stepik, які є дуже популярними в Україні. На даних платформах використовують різні засоби перевірки домашнього завдання. Для контролю теоретичних знань застосовують традиційні методи, переважно тестові задачі. Якщо це стосується коду, то перевірка, а саме реалізація цього процесу є складним, ресурсомістким процесом та водночас цікавим.*

*Метою роботи є створення вебсервісу для розміщення та перевірки практичних завдань, а саме завдань, де потрібно реалізувати певну програму. Також метою мого проекту є розробка алгоритму перевірки коду, який буде реалізовано шляхом прогону деяких вхідних даних та порівняння їх з попередньо заданими вихідними даними.*

*Отримане програмне забезпечення може бути використане будь-де, тому що створене воно у вигляді програмного пакета, який можна з легкістю розгорнути на сервері чи персональній машині. Безпосередньо програма може бути використана для перевірки робіт студентів в моєму навчальному закладі.*

***Ключові слова:*** *веб додаток, мова розмітки сайтів html, стилі css, фреймворк flask, мова програмування python, база даних.*

### Вступ

Що, взагалі таке тестування? Тестування - це дослідження, призначений для виявлення інформації про якість роботи продукту відносно контексту, в якому він має використовуватись. Техніка тестування також включає як процес пошуку помилок або інших дефектів, так і випробування програмних складових з метою оцінки. [1]

Наша задача полягає в тому, щоб написати додаток, що буде перевіряти невеликі програми на правильність їх роботи. Що полягає під словом “правильність”? У нашому випадку, це позитивне проходження тестів та коректне завершення роботи програми. Сам тест реалізує прогін деяких вхідних даних через програму, та порівняння вихідних даних з попередньо підготовленими правильними вихідними даними.

Реалізувати у вигляді графічного інтерфейсу чи вебсайту? На нашу думку, краще використати другий варіант. Ось декілька причин чому:

– легкий доступ до вебдодатку. В цьому випадку користувачу потрібен лише комп'ютер з браузером і з'єднання з інтернетом;

- оновлення додатку. Для того, щоб оновити вебдодаток, його необхідно оновити його тільки на сервері та всі користувачі відразу можуть працювати з новою версією;
- вебдодатки більш універсальні та практичні для кінцевого користувача. Вам достатньо буде встановити вебдодаток на сервер, що працює під будь-якою сучасної ОС, і ви зможете користуватися ним через інтернет на будь-якому комп'ютері або мобільному пристрої.

Дивлячись на ринок, можна сказати, що вебдодатки давно обігнали звичайні графічні програми. Вони легші в розробці, розгортанні на сервері, вони гнучкіші, та більш доступні. Тому я не бачу сенсу в GUI, принаймні при реалізації мого програмного забезпечення.

**Стек технологій**

Перше що потрібно обрати для створення будь-якого додатку, це мова програмування. Ми зупинились на Python 3.x, тому що вона має досить цікаві фреймворки для створення веб сервісів, які варто дослідити. Основні характеристики:

- інтерпретована мова програмування високого рівня;
- динамічна типізація;
- лаконічний синтаксис;
- мобільність програм;
- популярність, і як наслідок - велика спільнота розробників.

Недоліки:

- низька швидкодія, що характерно для всіх інтерпретованих мов;
- відсутність статичної типізації, що також уповільнює роботу програм.

Наступним завданням є обрання фреймворку для створення серверної частини. Існує два найпопулярніших кандидати: Flask та Django. Перший - легкий і гнучкий фреймворк, в якому не встановлено нічого зайвого. Це дозволяє вибирати модулі для вирішення конкретних завдань і встановлювати їх за потреби. Django - надає пакет «все включено»: у вас є панель адміністратора, інтерфейси баз даних, ORM, і структура каталогів для ваших додатків і проектів.

Flask частіше використовують коли потрібно більше контролю над компонентами додатку, або створення чогось нетипового. Django, якщо вас цікавить кінцевий продукт. Особливо якщо потрібно працювати з прямолінійним додатком, таким як сайт з новинами, онлайн магазин чи блог. [2] Тому ми зупинились на першому варіанті - Flask.

Надалі при безпосередній розробці програмного продукту знадобляться ще декілька модулів та бібліотек для реалізації необхідних можливостей додатку. Серед яких:

- wtforms - для реалізації функціонування вебформ;
- subprocess - для запуску зовнішніх програм;
- flask_sqlalchemy - для роботи з базою даних;
- flask_admin - для управління базою даних;
- flask_bootstrap - для інтеграції з Bootstrap4;
- os, shutil - модулі для системних команд та налаштувань.

Для розробки клієнтської частини програми використаємо мову розмітки HTML 5 та мову таблицю стилів CSS, та набір інструментів Bootstrap4, що містить HTML і CSS-шаблони оформлення для типографій, вебформ, кнопок, міток, блоків навігації та інших компонентів вебінтерфейсу, включаючи JavaScript-розширення.

При створенні розмітки будемо використовувати шаблонизатор Jinja2. Це рушій шаблонів для мови програмування Python створений Арміном Ронакером з ліцензією BSD. На відміну від схожого рушія шаблонів у Django, Jinja використовує вирази у стилі мови Python, тому процес розмітки документу стає подібним до написання звичайного коду.

**Засоби реалізації**

Для розгортання та створення проекту використаємо нову розробку Microsoft - WSL. Windows Subsystem for Linux - це прошарок між ядром Windows і додатками для Linux, який дозволяє перетворювати системні виклики до ядра Linux в виклики до ядра Windows. [3] Завдяки тому, що віртуалізація практично відсутня, таке рішення працює швидше традиційної віртуалізації, де повністю емулюється комп'ютер, як це відбувається в Oracle VirtualBox і VMWare Player.

Для розробника WSL - це зручне середовище розробки. Все ж встановлення багатьох мов, компіляторів та інтерпретаторів, утиліт в Linux відбувається куди простіше.

Також при створенні будемо використовувати VS Code - зручний редактор коду з безліч розширеннями, Git система контролю версій, та Github для збереження мого додатку.

Програму будемо писати у вигляді пакету. Це дозволяє надалі спрости запуск програми, її повторне використання, та розгортання на сервері. Файл з розширенням '.py' - це модуль. В ньому можна створити в ньому класи, функції та змінні. Якщо в директорію покласти кілька модулів і створити файл '__init__.py', створиться пакет з ім'ям даної директорії.

Далі буде повністю описано процес реалізації цього додатку та тестуванні його на певних даних.

**Макет проекту**

Файлова ієрархія проекту буде виглядати наступним чином (рис. 1):

- CodeChecker/ - основна директорія додатку
- .venv/ - директорія в якій знаходяться файли віртуального середовища Python.
- app/ - безпосередньо пакет нашого додатку
- static/ - директорія для статичних файлів, зокрема файлів стилю css
- templates/ - директорія для файлів розмітки html
- temp/ - директорія тимчасових файлів
- app.db - база даних
- config.py – код налаштувань
- run.py – програма запуску сервісу

**Створення та стилізація вебсторінки**

Наступним кроком є створення HTML сторінок та їх CSS оформлення за допомогою BootStrap4. Створюємо в теці templates/ 4 файли:

- base.html - базовий шаблон, що містить заголовок, навігацію, та так званий "підвал" сайту;
- main.html – головна сторінка, що містить список задач. Наслідується від сторінки base за допомогою можливостей Jinja;
- result.html - сторінка, на яку будуть передаватися результати тестування;
- task.html – сторінка з формою для вводу розв'язання певної задачі.


Рис. 1

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1, shrink-to-fit=no">
    <link rel="shortcut icon" href="../static/code.ico">
    {{ bootstrap.load_css() }}
    <title>CodeChecker</title>
</head>
```

В лістингу вище наведено “голову” сторінки base.html, де вказано основну інформацію: версія HTML, кодування, розташування файлу стилів, титулка.

```
<body>
    <div class="container">
        <header>
            <nav class="navbar navbar-default navbar-dark bg-
primary rounded mt-3 mb-3">
                <div>
                    <a class="navbar-
brand" href="{{ url_for('mainPage') }}">
                        <img src="static/code.svg" width="30" height
="30" hspace="10" class="d-inline-block align-middle" alt="">
                    <strong class="text-light align-
middle">Cody</strong>
                    </a>
                </div>
                <div>
                    <a class="text-
light" href="https://github.com/Ihoriam">Github</a>
```

```
                </div>

            </nav>
        </header>
        {% block content %}
        {% endblock %}
```

В цій частині HTML розмітки файлу base.html наведена одна з цікавих можливостей Jinja - наслідування. {% block content %}{% endblock %}

Ми використали оператор керування блоком, щоб визначити місце, де можна ставити похідні шаблони. Блокам присвоюється унікальне ім'я, на яке похідні шаблони можуть посилатися.

Тепер можна написати main.html, наслідуючи його від base.html, що є хорошою практикою в програмуванні.

```
{% extends "base.html" %}
{% block content %}
<main role="main">
  <section class="jumbotron text-center mb-3">
    <div class="container">
      <h1 class="jumbotron-heading">Вітаю!</h1>
      <p class="lead text-
muted">Ви піймались! Даний сервіс створено для перевірки Вашого дома
шнього завдання з програмування!</p>
      <p class="text-black-
50">Тепер не вийде аби як скинути пусту папку з домашкою на пошту!</
p>
      <p>
        <a href="#" class="btn btn-primary my-2">До роботи</a>
      </p>
    </div>
  </section>
...
{% endblock %}
```

Цікавість цього коду полягає в використанні готових html класів, реалізованих в наборі інструментів Bootstrap4. [4] Це дозволяє економити час на створення CSS стилів власноруч.

Далі створено ще два html документа task.html та result.html. Шаблони також підтримують керуючі оператори, їх задають в {% ...%}. [2] В даному сценарії використано оператор умовного блока {% if flag %}.

```
<div class="row bg mt-3 mb-3 rounded">
        <div class="col text-center">
          {% if flag %}
            <h3 class="text-success">Успішно виконано</h3>
            {% else %}
            <h3 class="text-danger">Ой, десь помилка</h3>
            {% endif %}
        </div>
    </div>
```

Ось так виглядає основна сторінка сайту (рис. 2).

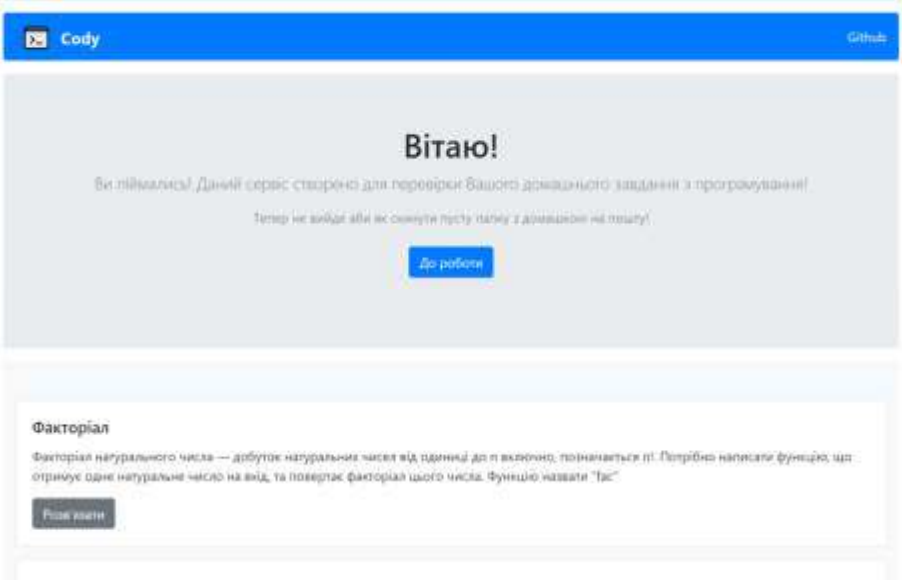



Рис. 2

**Ініціалізація пакету**

В директорії app розміщений файл з іменем__init__.py, в якому записано наступний код:

```
from flask import Flask
from config import Config
from flask_sqlalchemy import SQLAlchemy
from flask_admin import Admin
from flask_bootstrap import Bootstrap
app = Flask(__name__)
app.config.from_object(Config)
bootstrap = Bootstrap(app)
db = SQLAlchemy(app)
admin = Admin(app, name='cody', template_mode='bootstrap3')
from app import routes, models
app.run(debug=True)
```

Сценарій вище імпортує основні модулі для функціювання додатку. Серед яких модулі для роботи з базою даних, з формами та безпосередньо модуль Flask. А також створює екземпляр класу Flask.

**Реалізація вебформ**

Щоб отримувати певні дані від користувача, на сторінках використовують форми, а саме тег <form>. Щоб зв'язати HTML форму та Python код, використано розширення flask-wtf, яке встановлене раніше.

Створюємо в директорії checky/ файл з назвою forms.py:

```
from flask_wtf import FlaskForm
from wtforms import TextAreaField, SubmitField, SelectField
from flask_codemirror.fields import CodeMirrorField
from wtforms.validators import DataRequired
class CheckerForm(FlaskForm):
    codeField = TextAreaField("Ваш код", render_kw={"rows": 15, "col
s": 10}, validators=[DataRequired()])
```

```
    selectLang = SelectField('', choices=[('gcc', 'C'), ('g++', 'C++
'), ('py', 'Python'), ('java', 'Java'), ('js', 'JavaScript')])
    submit = SubmitField('Перевірити')
```

В перших рядках імпортується клас FlaskForm, та декілька функцій. Далі створюється екземпляр класу, в полях якого ініціалізую змінні, як один з типів форм. В нашому випадку це TextAreaField, тобто текстове поле, в яке користувач буде вводити дані (рис. 3).

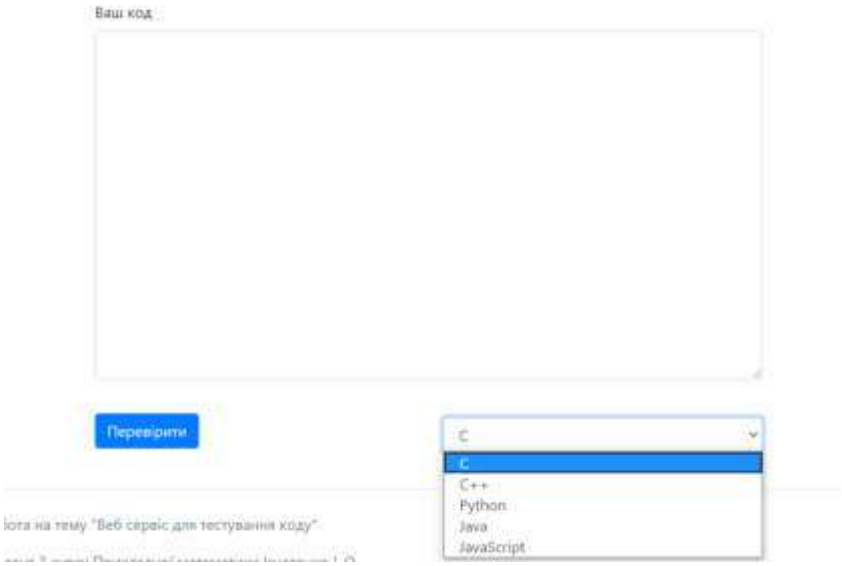



Рис. 3

Для роботи розширення flask-wtf необхідно задати секретний ключ. Для цього створюється файл з налаштуваннями в теці CodeChecker/ :

```
import os
class Config(object):
SECRET_KEY = os.environ.get('SECRET_KEY') or 'you-will-never-guess
```

Flask і деякі його розширення використовують значення SECRET_KEY як криптографічний ключ для генерації підписів або токенів. Розширення Flask-WTF використовує його для захисту форм від вебатаки під назвою Cross-Site Request Forgery або CSRF (вимовляється як «seasurf»). [2]

**База даних**

Для повноцінного функціювання сайту необхідна база даних, де будуть зберігатися задачі, частини коду, тести, та ще певна інформація. Для цього створюємо скрипт models.py.

```
from app import db
from app import admin
from flask_admin.contrib.sqla import ModelView

class Code (db.Model):
    id = db.Column(db.Integer(), primary_key=True)
    id_task = db.Column(db.Integer(), db.ForeignKey('task.id'))
```

```
    id_lang = db.Column(db.Integer(), db.ForeignKey('language.id'))
    pre_code = db.Column(db.Text())
    main_code = db.Column(db.Text(), nullable=False)
    post_code = db.Column(db.Text())
    tests = db.relationship('Test', backref='code', lazy=True)
```

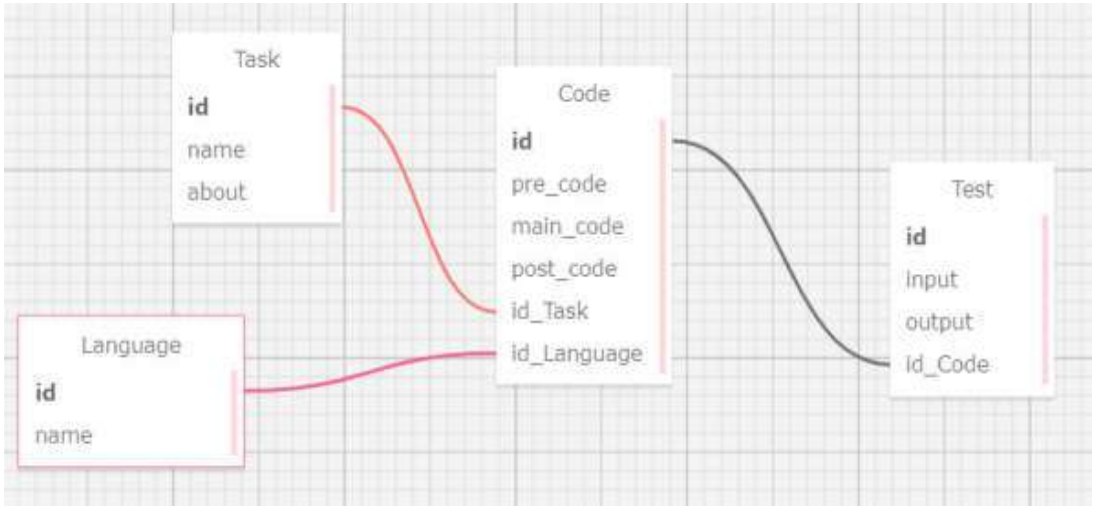



Рис. 4

В даній частині коду наведеній вище реалізована одна з таблиць бази даних. На малюнку вище продемонстрована діаграма всієї бази даних (рис. 4).

**Алгоритм перевірки**

В цьому розділі будемо працювати з файлом routes.py. Це основний скрипт вебдодатку, що містить в собі логіку URL сторінок, та алгоритм тестування коду, введеного користувачем. Алгоритм працює на таких мовах як Python, C, C++, Java та JavaScript. Нижче наведено алгоритм перевірки для мови Python, для інших мов він схожий.

```
@app.route('/task/<int:id_task>', methods=['GET', 'POST'])
def taskPage(id_task):
    form = CheckerForm()
    task = Task.query.filter_by(id = id_task).first()
    if request.method == 'POST' and form.validate_on_submit():
        # print(form.selectLang.data)
        if form.selectLang.data == 'py':
            code = Code.query.filter_by(id_task=id_task, id_lang=3).
first()
            tests = Test.query.filter_by(id_code=code.id).all()
            num_tests = len(tests)
            count = 0
            with open("temp/py/sub.py", "w") as sub:
                sub.writelines([code.pre_code, '\n', form.codeField.
data, '\n', code.post_code])
            for test in tests:
                # print(test.input_data)
                process = subprocess.run(['python3', 'temp/py/sub.py
', *test.input_data.split()],
```

```
                                        check=False,
                                        capture_output=True,
                                        universal_newlines=True)

                if (process.stdout) and process.stdout.strip() == te
st.output_data.strip():
                    count += 1
                    # print(count)
                else:
                    break
```

1) програма отримує з бази даних інформацію та тести на конкретно вибрану задачу;
2) оброблює форму з кодом, що заповнив користувач;
3) в циклі, програма проганяє тести через subprocess.run() [5], що запускає на фоні програму користувача з різними вхідними даними;
4) потім додаток отримує вихідні дані та звіряє їх з правильними.

```
if 'post_process' in locals():
            error = post_process.stderr
        else:
            error = process.stderr

        if count == num_tests:
            flag_valid=True
        else:
            flag_valid=False

        return render_template('result.html',
                               code=form.codeField.data,
                               propose_code = code.main_code,
                               error=error,
                               flag=flag_valid,
                               count=count,
                               num_tests=num_tests
                               )
```

Далі програма перевіряє кількість коректно пройдених тестів, та використовується функцію render_template, що генерує сторінку result.html та передає дані до неї.

**Адміністрування даних**

В проекті також створена сторінка адміністратора за допомогою додатку flask-admin. В ній реалізовані 4 базові функції управління даними «створення, зчитування, зміна і видалення» CRUD (рис. 5).

**Висновок**

В результаті завершення цього комплексного проекту було використано такі технології, як: мова Python 3.x, фреймворк Flask, форми, база даних, wsl, шаблонізатор Jinja2, мова розмітки HTML, таблиця стилів CSS, Bootstrap4 та ще декілька модулів Python.

Поєднавши цей стек технологій, було створено корисний додаток з серверною та клієнтською частиною.

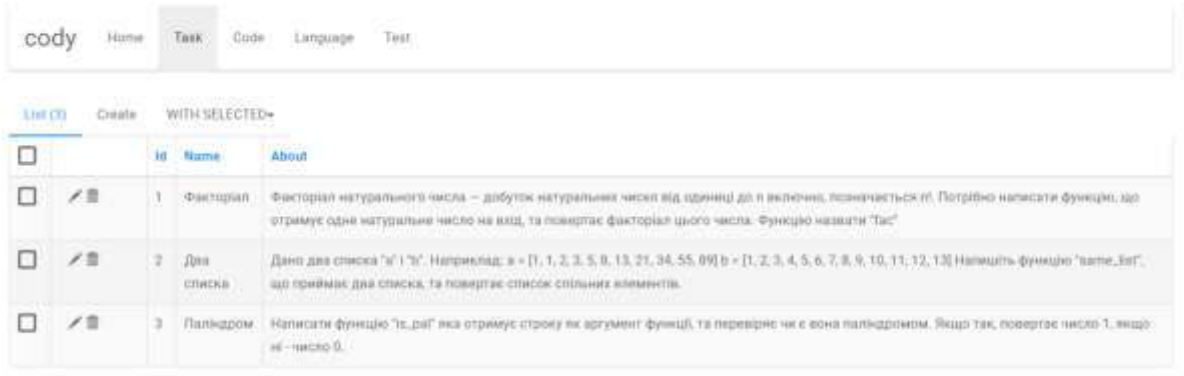



Рис. 5

На нашу думку, проект цікавий як з теоретичної, так і з практичної точки зору. В майбутньому його можна використати, як основу чи як мікросервіс для більш масштабного проекту. Наприклад – для кафедрального сайту, де студенти можуть дистанційно опрацьовувати матеріал. Шляхами удосконалення можуть бути наступні:

- особистий кабінет;
- підсвітка синтаксису;
- більше мов для проходження завдань;
- підвищення швидкодії за допомогою паралельного програмування;
- більш детальна система сповіщення про помилки.

**Список використаної літератури:**

1. Криспин Л., Грегори Д. / Гибкое тестирование. Практическое руководство для тестировщиков ПО и гибких команд - М.: Вильямс, 2010. - 464 с.
2. Miguel Grinberg / Flask Web Development: Developing Web Applications with Python - Publisher(s): O'Reilly Media, Inc.
3. Microsoft blog [Електронний ресурс] // Learn About Windows Console & Windows Subsystem For Linux (WSL) - Режим доступу: https://devblogs.microsoft.com/commandline/learn-about-windows-console-and-windows-subsystem-for-linux-wsl/
4. w3schools.com // Bootstrap Tutorial - Режим доступу: https://www.w3schools.com/bootstrap/
5. Python 3 documentation // subprocess - Subprocess management - Режим доступу: https://docs.python.org/3/library/subprocess.html

**References:**

1. Crispin L., Gregory D. / Flexible testing. A practical guide for software testers and flexible teams - M .: Williams, 2010. - 464 p.
2. Miguel Grinberg / Flask Web Development: Developing Web Applications with Python - Publisher(s): O'Reilly Media, Inc.
3. Microsoft blog [Електронний ресурс] // Learn About Windows Console & Windows Subsystem For Linux (WSL) - Access mode: https://devblogs.microsoft.com/commandline/learn-about-windows-console-and-windows-subsystem-for-linux-wsl/
4. w3schools.com // Bootstrap Tutorial - Access mode: https://www.w3schools.com/bootstrap/
5. Python 3 documentation // subprocess - Subprocess management - Access mode: https://docs.python.org/3/library/subprocess.html

**IHNATENKO Ihor,**
The Bohdan Khmelnytsky National University of Cherkasy, student
**SERDIUK Oleksandr,**
The Bohdan Khmelnytsky National University of Cherkasy, PhD, Senior Lecturer

**DEVELOPMENT OF AN EDUCATIONAL PORTAL FOR VERIFICATION OF COMPUTER PROGRAMS WRITTEN BY STUDENTS**

***Summary. Introduction.*** *Currently, online learning is widely used in schools and in the field of IT. Coursera, EDX, Udemy, Udacity are the world's leading online learning resources. It is also worth mentioning the Ukrainian project of mass open online courses Prometheus and the educational platform Stepik, which are very popular in Ukraine. These platforms use various way to check homework. Traditional methods are used to control theoretical knowledge. But when we need to verify code, this process becomes complex, resource-intensive and at the same time interesting to solve .*

***Purpose.*** *The purpose of the course work is to create a web service for posting and testing practical tasks, namely tasks where you want to implement a program. Also, the goal of my project is to develop a method of verifying the code, which will be implemented by running some input data and comparing them with predefined output data.*

***Results.*** *The resulting software can be used anywhere, because it is created in the form of a software package that can be easily deployed on a server or personal machine. The program can be used directly to check the work of students in my university.*

***Conclusion.*** *As a result of completing this complex project, I used technologies such as Python, Flask framework, forms, database, wsl, Jinja2 template, HTML markup language, CSS stylesheet, Bootstrap4 and several other Python modules.*

*Combining this technology stack, I created an interesting application, with a server and client part. I also improved my skills in working with the database and using the Linux command terminal.*

*In my opinion, the project is interesting both from a theoretical and a practical point of view. In the future, it can be used as a basis or as a micro-service for a larger project. For example - for our cathedral site, where students can remotely process the material.*

***Keywords:*** *web application, html site markup language, css styles, flask framework, python programming language, database.*

*Стаття надійшла 28.08.2018*
*Прийнято до друку 20.09.2018*

УДК 004.4, 004.5

PACS 07.05.Bx, 07.05.Wr

**СЕРДЮК Олександр Анатолійович**
старший викладач кафедри прикладної математики та інформатики Черкаського національного університету ім. Б. Хмельницького

**ОСАДЧА Богдана Анатоліївна**
учениця Драбівського НВК «загальноосвітня школа I-III ступенів імені С.В. Васильченка - гімназія» Драбівської районної ради

# РЕАЛІЗАЦІЯ ТА ПОРІВНЯЛЬНИЙ АНАЛІЗ МЕТОДУ РОЮ ЧАСТИНОК ТА АЛГОРИТМУ СВІТЛЯЧКІВ

*У роботі розглянуто два ройові методи оптимізації: метод рою частинок та алгоритм світлячків. Для перевірки роботи розглянутих методів розроблено програму з графічним інтерфейсом користувача для чисельного розв'язування задач безумовної оптимізації.*

*Розроблена програма написані у середовищі Lazarus, що дозволяє досить просто розробляти програми, що мають графічний інтерфейс, а мова програмування Object Pascal, якою реалізована програма, є нескладною для розуміння, однак, потужною для створення складних програм з великою кількістю обрахунків.*

*Отримані результати аналізу роботи реалізованих методів співпали з очікуваними і показали дієвість обраних алгоритмів при розв'язанні задач безумовної оптимізації навіть для складних функцій, однак, застосування розглянутих методів вимагає більш детального розгляду способів вибору їх параметрів для кожної конкретної задачі.*

*Одним із можливих подальших напрямків роботи може бути використання методу рою частинок та алгоритму світлячків саме для підбору параметрів цих методів при розв'язанні задач оптимізації.*

***Ключові слова:*** *методи ройової оптимізації, розробка програмного продукту.*

## Вступ

У сучасних умовах у науці і техніці існує стійка тенденція, пов'язана з необхідністю розв'язання широкого спектру задач на пошук глобальних оптимальних розв'язків. Для цієї мети застосовуються різноманітні широковідомі методи, але далеко не всі з цих задач можуть бути розв'язані з використанням традиційних підходів за умови наявності незначної кількості допоміжної інформації, відомої апріорі.

Умовно методи розв'язання задач глобальної оптимізації ділять на детерміновані та стохастичні. У основі такої класифікації лежить спосіб організації пошуку відповідно без та з застосуванням псевдовипадкових елементів. Особливе місце серед стохастичних методів посідають методи, що опираються на імітацію природних процесів і реалізують адаптивний випадковий пошук. У теорії оптимізації такі методи фігурують під назвою метаевристичних. Найбільш популярними серед таких методів є група еволюційних алгоритмів, зокрема – генетичні алгоритми. Однак, значне місце посідають методи, що опираються на дослідження сукупної поведінки великої кількості частинок, так звані ройові алгоритми.

**Метою роботи** є реалізація та дослідження двох алгоритмів оптимізації: рою частинок та рою світлячків. Ці алгоритми можуть бути застосовані для ефективного розв'язання широкого кола задач оптимізації, в тому числі неперервної, дискретної, комбінаторної і багатокритеріальної.

Для досягнення поставленої мети необхідно розглянути канонічний алгоритм оптимізації роєм частинок та канонічний алгоритм рою світлячків. Для цього реалізуємо ці алгоритми, протестуємо їх роботу на деяких тестових функціях, виконаємо порівняльний аналіз отриманих результатів та зробимо деякі рекомендації щодо підвищення швидкості розв'язання оптимізаційних задач за допомогою розглянутих алгоритмів.

**Канонічний алгоритм оптимізації роєм частинок**

Алгоритм оптимізації роєм частинок (PSO, Particle Swarm Optimization) відноситься до біонічних мультиагентних методів глобальної оптимізації і моделює соціальну поведінку взаємодіючих агентів. Ідея методу PSO належить Дж. Кеннеді та Р. Еберхарту, які вперше сформулювали і успішно застосували його для розв'язку оптимізаційних задач і навчання штучних нейронних мереж. Основи ройової оптимізації були закладені у більш ранніх роботах з комп'ютерного моделювання переміщення живих істот у пташиній зграї чи зграї риб. У роботі Кеннеді та Еберхарта висловлена ідея, що стала основою методу: «Принаймні в теорії окремі члени зграї можуть отримати вигоду від досліджень та попереднього досвіду усіх інших членів зграї при пошуку їжі. Ця перевага стає вирішальною і навіть переважає конкуренцію всякий раз, коли харчові ресурси розташовані на шляху випадковим або невідомим чином». Тобто, соціальний поділ інформації серед представників одного виду дає еволюційні вигоди усім членам популяції. Ця гіпотеза домінування колективного інтелекту стала фундаментальною при розробці оптимізації роєм частинок.

Задача безумовної глобальної оптимізації формулюється як задача мінімізації деякої цільової функції $F(\mathbf{x})$ у просторі пошуку $D$:

$$F(\mathbf{x}) \to \min,\ \mathbf{x} \in R^d, \tag{1}$$

де область $D$ представляє собою дійсний гіперкуб розмірністю $d$, а $\mathbf{x}$ – вектор аргументів цільової функції $F$, що має значення глобального оптимуму у деякій точці $\mathbf{x}^*$.

У методі PSO рій частинок являє собою сукупність точок, що є потенційними розв'язками, які переміщаються у просторі у пошуках глобального оптимуму. При своєму русі частинки намагаються покращити знайдений ними раніше розв'язок і обмінюються інформацією зі своїми сусідами.

Позначимо сукупність позицій частинок рою як

$$X = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}, \tag{2}$$

де $n$ – кількість частинок у рої.

Під позицією $i$-ї частинки розуміється сукупність її координат $(x_{i1}, x_{i2}, ..., x_{id})$ у просторі пошуку розмірністю $d$, $i = \overline{1,n}$. При проведенні оптимізації зазвичай досить 10-30 частинок. У багатьох випадках рій дає можливість знайти глобальний оптимум навіть коли кількість частинок у ньому менша розмірності $d$ простору пошуку.

На початковому етапі роботи алгоритму PSO проводиться випадкова ініціалізація рою частинок. Якщо відсутня яка-небудь апріорна інформація про оптимізовану функцію, то найпростіше початкові положення частинок генерувати за формулою

$$x_{ij} = rand\left(x_{j\min}, x_{j\max}\right), \tag{3}$$

де $x_{ij}$ – $j$-а координата $i$-ї частинки; $rand\left(x_{j\min}, x_{j\max}\right)$ – випадкове число з рівномірним законом розподілу на інтервалі $\left(x_{\min}, x_{\max}\right)$, який визначає межі простору пошуку для $j$-го виміру. Зауважимо, що загалом межі можуть бути різними для різних координат частинки: наприклад, у двовимірному просторі пошуку може бути, що перша координата частинок, $x_{i1} \in (0,100)$, а друга $x_{i2} \in (-10,10)$.

З роєм частинок також асоціюється множина векторів їх швидкостей

$$V = \left\{\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_n\right\}. \tag{4}$$

Кожний вектор швидкості має стільки ж координат, скільки й частинки. На початковому етапі всі швидкості можна вважати рівними нулю, або ж задавати швидкостям випадкові значення з того ж діапазону, з якого задаються значення координат.

На наступних кроках алгоритму компоненти швидкостей і позицій частинок оновлюються за наступними формулами:

$$\mathbf{v}_i^{t+1} = w \cdot \mathbf{v}_i^t + c_1 \cdot r_1 \cdot \left(\mathbf{p} - \mathbf{x}_i^t\right) + c_2 \cdot r_2 \cdot \left(\mathbf{g} - \mathbf{x}_i^t\right), \tag{5}$$

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \mathbf{v}_i^{t+1}, \tag{6}$$

де $\mathbf{v}_i^{t+1}$ і $\mathbf{x}_i^{t+1}$ – нові вектор швидкості і положення $i$-ї частинки (у момент часу $t+1$); $\mathbf{p}$ –найкраще положення частинки, знайдене нею на поточний момент (personal best); $\mathbf{g}$ – найкращий розв'язок, знайдений усім роєм на поточний момент (global best); $w$ – коефіцієнт, чи вагова частка, інерції частинки; $c_1$ – когнітивний коефіцієнт; $c_2$ – соціальний коефіцієнт; $r_1$, $r_2$ – випадкові числа з рівномірним розподілом з інтервалу $[0,1]$, що беруться окремо для кожної частинки.

Говорячи простими словами, коефіцієнт $w$ показує, наскільки буде впливати на частинку інерція руху, коефіцієнт $c_1$ відповідає за те, наскільки на наступне положення частинки впливатиме найкращий розв'язок, знайдений нею на поточний момент, а коефіцієнт $c_2$ показує, наскільки на наступне положення частинки впливає найкращий розв'язок, знайдений усім роєм на даний момент. Наприклад, значення $c_1 = 0$ свідчить, що частинка «не враховує» свого власного досвіду, а значення $c_2 = 0$ – що частинку абсолютно «не цікавить» попередній досвід усього рою, і вона опирається лише на власний досвід.

Значення коефіцієнта $w$ відіграє велику роль і визначає компроміс між дослідженням простору і локалізацією положення частинки у просторі пошуку. При $w \geq 1$ швидкість частинки збільшується (з урахуванням раніше розглянутого

обмеження) і рій «розходиться». Частинкам важко змінити напрямок руху для того, щоб повернутися до перспективної області пошуку рішення, внаслідок великої інерції руху. При $w<1$ частинки сповільнюються до тих пір, поки швидкість не стане рівною нулю. Таким чином, великі значення $w$ сприяють дослідженню простору пошуку, а малі – локалізації розв'язку. Однак, варто зазначити, що занадто малі значення $w$ позбавляють рій здатності досліджувати простір пошуку: чим менше значення $w$, тим більший вплив когнітивної та соціальної компонент на рух частинок. Як і інші параметри, оптимальне значення $w$ є проблемно-орієнтованим, тобто, залежить від задачі, що розв'язується.

Якщо у процесі руху частинка виходить за межі простору пошуку, то для відповідних компонент встановлюються граничні значення, повертаючи, таким чином, частинку на кордон простору пошуку.

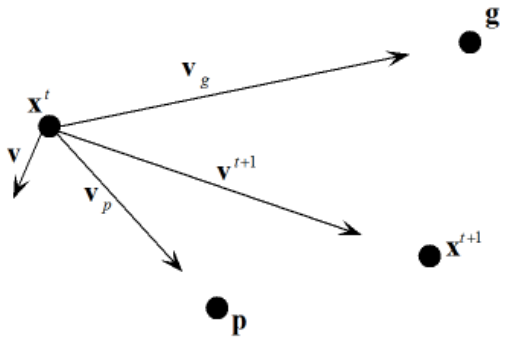



Рис.1. Геометрична ілюстрація правила оновлення швидкостей та положення частинки

На рис. 1 через $\mathbf{v}_p = c_1 \cdot r_1 \cdot (\mathbf{p} - \mathbf{x})$ і $\mathbf{v}_g = c_2 \cdot r_2 \cdot (\mathbf{g} - \mathbf{x})$ позначені відповідно когнітивна та соціальна компоненти нової швидкості частинки. Випадкові числа $r_1$ і $r_2$ вносять у пошук елемент випадковості.

Величини вагових коефіцієнтів рівняння (5) часто вибираються у діапазоні $1 < c_1, c_2 \le 2$ (хоча це й не безумовна вимога), що у багатьох випадках забезпечує збіжність описуваного методу оптимізації.

Для підтримання балансу між локальним і глобальним пошуком чисельні значення коефіцієнтів $c_1$ і $c_2$ зазвичай обираються рівними, що забезпечує у більшості випадків найкращі результати і стабільність пошуку. Можливі також різні способи динамічного налаштування параметрів рою, однак адаптація вимагає додаткових початкових ітерацій алгоритму, що може призвести до збільшення кількості обчислень цільової функції, необхідних для знаходження оптимуму.

Рій володіє пам'яттю про найкращі рішення, знайдені його окремими частинками і усім роєм в цілому. Під час ініціалізації початкові позиції частинок вважаються й найкращими (за відсутності попереднього пошуку). На кожній же наступній ітерації алгоритму PSO після застосування формул (5, 6) індивідуальні кращі положення кожної частинки $\mathbf{p}_i$ і найкращий знайдений роєм розв'язок $\mathbf{g}$ оновлюються за правилами

$$\begin{aligned} \mathbf{p}_i &= \mathbf{x}_i, \text{ якщо } F(\mathbf{x}_i) < F(\mathbf{p}_i), \\ \mathbf{g} &= \mathbf{p}_i, \text{ якщо } F(\mathbf{p}_i) < F(\mathbf{g}). \end{aligned} \tag{7}$$

Так як розв'язок рою $\mathbf{g}$ є найкращим розв'язком, знайденим однією з його частинок, то досить пам'ятати її номер. Часто у випадку, якщо глобальний найкращий розв'язок рою збігається з найкращим розв'язком, знайденим частинкою, то додавання соціального компонента швидкості $c_2 \cdot r_2 \cdot (\mathbf{g} - \mathbf{x})$ не проводиться. Як вже зазначалось раніше, слід також звернути увагу на те, що для різних координат частинки генеруються різні випадкові числа $r_1$, $r_2$.

**Метод світлячків**

Оптимізація рою світлячків (glowworm swarm optimization або firefly swarm optimization), запропонована Крішнанандом (Krishnanand) і Гозе (Ghose), опирається на поведінку жуків-світлячків. У основі алгоритму лежить спостереження про те, що кожен світлячок може світитися з різною інтенсивністю, і це світло використовується для приваблювання жучком інших особин рою.

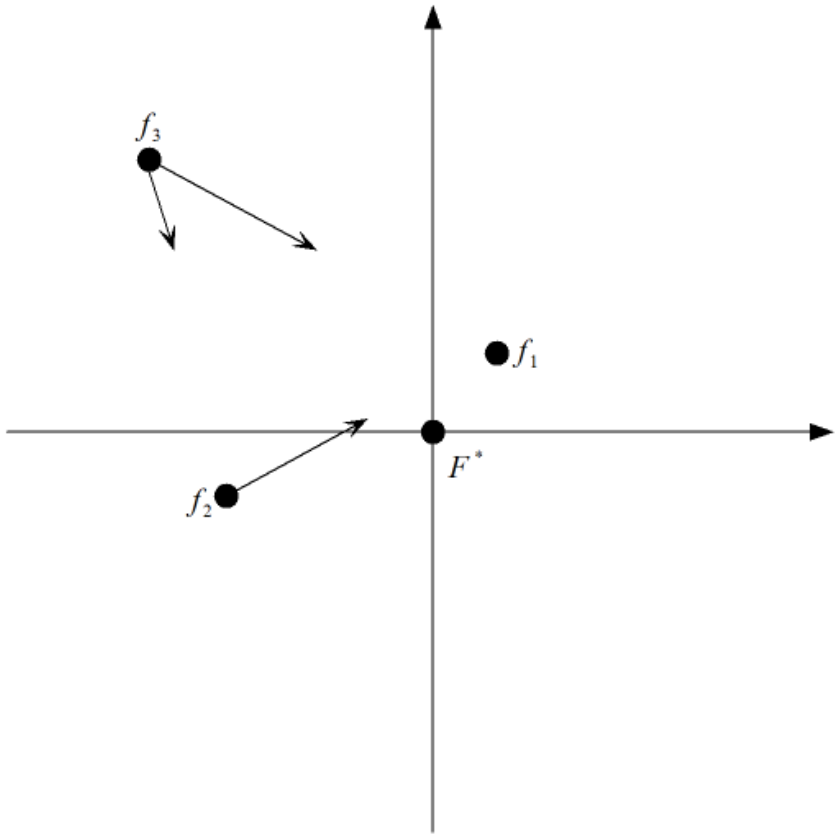



Рис.2. Геометрична ілюстрація основної ідеї алгоритму світлячків

Розглянемо метод на прикладі трьох світлячків (рис. 2). У даному методі світлячки, що знаходяться у більш хорошій позиції (ближче до мінімуму цільової функції $F^*$) світяться з більшою інтенсивністю. У такому випадку на рис. 2 світлячок $f_1$ світиться найяскравіше, світлячок $f_2$ – з середньою яскравістю, а світлячок $f_3$ – зі слабкою. Базова ідея методу полягає в тому, що світлячка притягує будь-який інший світлячок, який світиться більш яскраво, і це притягування (відстань, що долається світлячком до більш яскравого) тим сильніше, чим менша відстань між світлячками. Отже, світлячок $f_2$ притягуватиметься лише світлячком $f_1$, оскільки лише цей світлячок для даного має більшу яскравість, в той час, як світлячок $f_3$

притягуватиметься обома (сильніше притягуватиметься світлячком $f_1$ і слабше – світлячком $f_2$).

Світлячки породжують світло з інтенсивністю, обернено пропорційною квадрату відстані між ними. Це явище є свідченням того, що світлячка видно на обмеженій відстані. Вводяться наступні припущення:

- притягування світлячка пропорційне яскравості, тому у довільній парі світлячків менш яскравий світлячок буде прагнути підлетіти до більш яскравого; притягування зменшується зі збільшенням відстані. Якщо немає більш яскравого світлячка, то поточний рухатиметься випадковим чином [10];
- оскільки метод світлячків є метаевристичним алгоритмом, ми можемо задати інтенсивність як завгодно за умови, що більш висока інтенсивність пов'язана з кращим положенням світлячка (більш оптимальним розв'язком);
- інтенсивність яскравості світлячка визначається обернено пропорційно до значення цільової функції, таким чином мале значення функції відповідатиме високій інтенсивності, а більше значення функції – нижчій;
- слід задати притягування так, щоб до більш яскравого світлячка сильніше зміщувалися світлячки, що знаходяться ближче, а не більш віддалені [3].

Розглянемо рис. 3a. Світлячок $i$ знаходиться у діапазоні датчика (і на однаковій відстані) світлячків $j$ та $k$. Але вони мають різну область розв'язків. Отже, лише світлячок $j$ використовує інформацію з $i$.

Перейдемо до рис. 3b. Поява орієнтованого графа базується на відносному рівні кількості речовини, що світиться (люциферину), у кожного світлячка і наявності лише локальної інформації. Світлячки класифікуються у порядку зростання їх кількості речовини. Наприклад, світлячок, значення люциферину якого є найбільшим, на рис. 3 приймає значення «1».

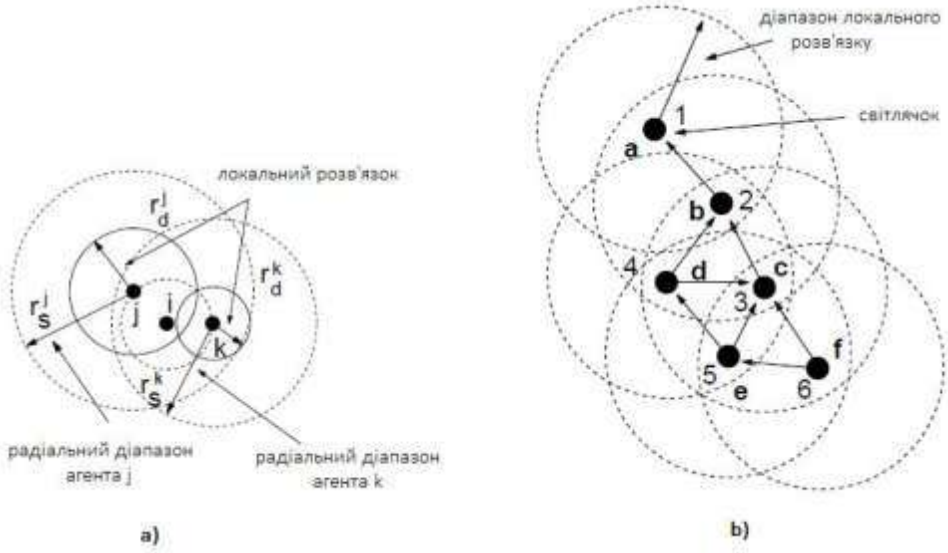



Рис. 3. Рух світлячків

Алгоритм світлячків починається з їх розміщення випадковим чином у просторі так, щоб вони були добре розсіяні. Спочатку всі світлячки містять однакову кількість люциферину. Кожна ітерація складається з фази оновлення люциферину з подальшим переходом у фазу, що базується на правилі переходу [10]. Нижче подано повний канонічний алгоритм розв'язання задачі за допомогою рою світлячків.

***Алгоритм методу світлячків*:**

1. Ініціалізація.
   1.1. Задаємо параметр притягування $\beta_{\max}$, коефіцієнт поглинання світла $\gamma$, параметр $\alpha$ для обчислення вектору позиції, причому $0<\beta_{\max}<1$, $0<\gamma<1$.
   1.2. Задаємо максимальну кількість ітерацій $N$, розмір популяції $K$, довжину вектору позиції світлячка $M$, мінімальні і максимальні значення для вектора позицій $\left(x_{j\min}, x_{j\max}\right)$, $j\in\overline{1,M}$.
   1.3. Задаємо цільову функцію.
   1.4. Випадковим чином створюємо кращий розв'язок $\mathbf{x}^*=\left(x_1^*, x_2^*, ..., x_M^*\right)$, $x_j^*=x_{j\min}+\left(x_{j\max}-x_{j\min}\right)\cdot r$, де $r$ –рівномірно розподілене випадкове число у діапазоні $[0,1]$.
   1.5. Створюємо початкову популяцію $P$.
      1.5.1. Перебираємо усі номери світлячків $k=\overline{1,K}$.
      1.5.2. Випадковим чином створюємо вектор позиції світлячка $\mathbf{x}_k$, $\mathbf{x}_k=\left(x_{k1}, x_{k2}, ..., x_{kM}\right)$.
      1.5.3. Якщо $x_k\notin P$, то $P=P\cup\mathbf{x}_k$, $k=k+1$.
      1.5.4. Якщо $k\le K$, то перейдемо на крок 1.5.2.
2. Номер ітерації $n=1$.
3. Номер світлячка $k=1$.
4. Номер світлячка $l=1$.
5. Якщо $F\left(\mathbf{x}_k\right)>F\left(\mathbf{x}_l\right)$, то перемістити $k$-го світлячка у напрямку $l$-го світлячка, тобто змінити позицію $k$-го світлячка.
   5.1. $\beta=\beta_{\max}\cdot e^{-\gamma d_{kl}^2}$, де $d_{kl}$ – відстань між світлячками.
   5.2. $\mathbf{x}_k=\mathbf{x}_k+\beta\left(\mathbf{x}_l-\mathbf{x}_k\right)+\alpha\left(r-0.5\right)$.
   5.3. $x_{kj}=\max\left\{x_{kj}, x_{j\min}\right\}$, $x_{kj}=\min\left\{x_{kj}, x_{j\max}\right\}$, $j\in\overline{1,M}$.
6. Якщо $l<K$, то $l=l+1$, перейти на крок 5.
7. Якщо $k<K$, то $k=k+1$, перейти на крок 4.
8. Визначити кращого світлячка за цільовою функцією $k^*=\arg\min F\left(\mathbf{x}_k\right)$.
9. Якщо $F\left(\mathbf{x}_k\right)<F\left(\mathbf{x}^*\right)$, то $\mathbf{x}^*=\mathbf{x}_k$.
10. Якщо $n<N$, то $n=n+1$, перейти на крок 3.
11. Відповіддю є $\mathbf{x}^*$.

**Критерії зупинки алгоритмів**

Як видно із поданого вище, основним критерієм зупинки роботи алгоритму є проходження певної кількості ітерацій. Тобто, перед запуском роботи задається потрібна кількість ітерацій, яку необхідно виконати, і по її закінченні отриманий найкращий розв'язок вважається розв'язком усієї задачі. Однак, часто наперед заданої кількості ітерацій буває недостатньо. Тому ми обрали в якості додаткових критеріїв наступні:

1. Зупинка обрахунків у випадку, якщо значення цільової функції не міняється

протягом певної кількості ітерацій.

2. Зупинка обрахунків у випадку, якщо наступне значення цільової функції відрізняється від попереднього не більше, ніж на деяку точність $\varepsilon$.

Перший з додаткових критеріїв слідкує за найкращим розв'язком, отриманим для усього рою. При розв'язанні задач стохастичної оптимізації на перших ітераціях часто буває швидке наближення до оптимального (чи умовно оптимального) розв'язку, після чого наступні більш точні розв'язки знаходяться все рідше й рідше і не дозволяють суттєво покращити значення цільової функції. У такому випадку нами використовується підрахунок ітерацій, протягом яких не відбувалось ніякого покращення розв'язку. Якщо кількість таких ітерацій (обов'язково підряд!) стає рівною певному наперед заданому числу, то пошук розв'язку припиняється і отриманий на поточний момент розв'язок для усього рою вважається розв'язком задачі. У дослідженнях нами використовувались значення від 50 до 250 ітерацій, що дозволило отримати розв'язки, близькі до оптимальних.

Другий критерій передбачає порівняння попереднього й наступного значення цільової функції, отриманого для розв'язку, кращого для усього рою. При отриманні нового значення $F^{t+1}$ проводиться його порівняння з попереднім значенням $F^t$: $\delta = \left| F^{t+1} - F^t \right|$, і у випадку, якщо отримується $\delta < \varepsilon$, де $\varepsilon$ – деяка наперед задана точність, то вважається, що подальше покращення цільової функції є недоцільним, і обчислення припиняються. Зауважимо, що такий же критерій можна застосовувати й окремо до координат попереднього та наступного кандидатів на розв'язок задачі, однак, досягнення даного критерію для усіх координат є набагато складнішим (за кількістю ітерацій, що потрібно виконати), тому дана модифікація нами у роботі не використовувалась.

**Розробка програмного продукту**

**Основні типи даних, використовувані у програмі**

Для реалізації методу рою частинок та алгоритму світлячків було створено кілька типів даних.

Опис частинки для методу рою частинок проводиться за допомогою типу `TParticle`:

```
TParticle = record
   x: TRealArray;       // значення координат (поточний розв'язок)
   fitness: real;       // значення цільової функції
   v: TRealArray;       // значення координат швидкості
   bestX: TRealArray;   // найкращий знайдений частинкою розв'язок
   bestF:  real;              // найкраще знайдене значення цільової
функції
end;
```

Кожна частинка зберігає інформацію про поточне положення (`x`), значення цільової функції (значення фітнесу `fitness`), поточну швидкість частинки (`v`), знайдене на даний момент найкраще положення частинки (`bestX`) та значення цільової функції для даного положення (`bestF`).

Рій частинок задається змінною типу `TParticleSwarmOptimisation`:

```
TParticleSwarmOptimisation = record
   numberParticles: integer;    // кількість частинок
   dim: integer;                // розмірність частинок
```

```
   ps: Array of TParticle;      // частинки
   f: TFitnessFunction;         // цільова функція

   // параметри частинок
   w,                           // вагова частка інерції
   c1,                          // когнітивна вагова частка
   c2: real;                    // соціальна вагова частка

   minX: TRealArray;            // максимальні значення координат
   maxX: TRealArray;            // мінімальні значення координат

   maxIterations: Integer;         // максимальна кількість ітерацій
   maxIterationsFlag: boolean;       // чи використовувати критерій
зупинки
   maxStableIterations:  Integer;     //  кількість  ітерацій  при
незмінному f
   maxStableIterationsFlag: boolean; // чи використовувати критерій
зупинки
   maxAccuracy: real;              // точність, яку треба досягти
   maxAccuracyFlag: boolean;         // чи використовувати критерій
зупинки

   iterations: Integer;        // виконано ітерацій
   stableIterations: Integer; // скільки ітерацій не мінялось f
   bestOldF: real;             // попереднє найкраще значення фітнесу
   stopCriterium: byte;        // який критерій зупинки виконався

   bestX: TRealArray;   // найкращий розв'язок
   bestF: real;         // найкраще значення фітнесу
end;
```

У такій змінній зберігаються: кількість частинок (`numberParticles`), розмірність простору (`dim`), а, відповідно, й розмірність векторів, що відповідають положенню частинки та швидкості частинки, масив частинок (`ps`), цільова функція, для якої шукається оптимальне значення (`f`), параметри методу (`w`, `c1`, `c2`), описані у розділі 1.1, мінімальні та максимальні граничні значення для кожної з координат (`minX`, `maxX`), максимальна кількість ітерацій (`maxIterations`), кількість ітерацій, протягом яких може бути незмінним значення цільової функції (`maxStableIterations`), точність, яку необхідно досягти (`maxAccuracy`), найкращий розв'язок, отриманий усім роєм (`bestX`), та відповідне значення цільової функції (`bestF`).

Додатково у змінній рою зберігаються допоміжні значення поточної кількості виконаних ітерацій (`iterations`), кількості ітерацій, протягом яких не мінялось значення цільової функції (`stableIterations`), попереднє значення цільової функції (`bestOldF`) та код критерію зупинки, який виконався (`stopCriterium`). Останнє значення потрібне для виводу користувачу інформації про те, чому саме алгоритм припинив роботу.

Вказані типи даних описані у модулі `PSOUnit`, який підключається у головному модулі програми.

Типи даних для роботи алгоритму світлячків майже ідентичні за наступними відмінностями.

Для світлячка не потрібно зберігати швидкість руху, а тому відповідних полів у

типі даних (TFireFly) немає, натомість є значення інтенсивності світла поточного світлячка:

```
TFireFly = record
   x: TRealArray;       // значення координат (поточний розв'язок)
   fitness: real;       // значення фітнесу
   intensity: real;     // яскравість світлячка
end;
```

Замість параметрів методу рою частинок у типі даних рою світлячків зберігаються параметри $\beta$, $\gamma$, $\alpha$:

```
   // параметри світлячків
   B0,                  // базова бета
   g,                   // гама
   a: real;             // альфа
```

Реалізація типів даних для алгоритму світлячків та відповідних підпрограм знаходиться у модулі FFAUnit.

Загальна структура згаданих модулів подібна: у кожному з них наявні описи типів даних та процедур для виконання необхідних операцій. Кожний з модулів містить 4 основні процедури:

1. Ініціалізація рою частинок (initPSO) чи світлячків (initFFA).

2. Виконання одного кроку методу рою частинок (oneStepPSO) чи алгоритму світлячків (oneStepFFA).

3. Виконання кроків методу частинок (allStepsPSO) чи алгоритму світлячків (allStepsFFA) до досягнення одного з критеріїв зупинки.

4. Перевірка, чи виконується критерій зупинки у методі рою частинок (checkForStopPSO) чи алгоритмі світлячків (checkForStopFFA).

5. Додатково для алгоритму світлячків реалізовано функцію розрахунку відстані між двома світлячками distance.

Підпрограми з п.1-2 відповідають наведеним у розділах 1.1 та 1.2 алгоритмах, а тому зупинятись на них детально не будемо. Зауважимо лише, що для можливості використовувати динамічно змінні масиви нами часто використовується вбудований тип Array, що дозволяє без перекомпіляції програми використовувати масиви різної довжини у змінних.

**Загальна структура програми**

Розроблений програмний продукт складається з чотирьох модулів, структурна схема яких наведена на рис. 4.

Головним модулем програми є MainUnit, у якому реалізовано графічний інтерфейс та виклик необхідних підпрограм і звертання до змінних, описаних у додаткових модулях.

Для дослідження роботи методу рою частинок використовується модуль PSOUnit, що підключається до головного модуля, а для дослідження роботи алгоритму світлячків – модуль FFAUnit, що теж підключається аналогічно.

Усі три зазначені модулі використовують додатковий модуль FunctionsUnit, у якому описано тип динамічного масиву дійсних чисел TRealArray, тип цільової функції TFitnessFunction та реалізовано тестові функції, про які буде згадано далі.

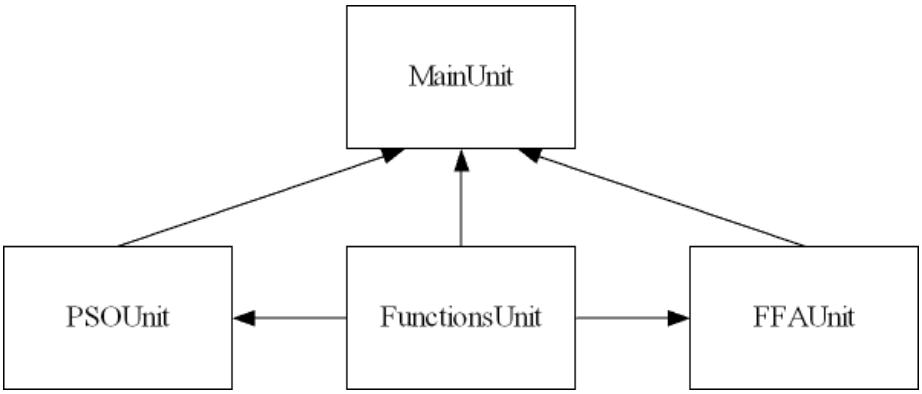



Рис. 4. Структура програмного продукту

**Функції для тестування**

У розробленому продукті реалізовано підпрограми обчислення значень кількох функцій, що часто використовуються для оцінки результатів оптимізаційних алгоритмів. Нижче наведено математичне подання функцій для довільної кількості змінних, їх оптимальні значення та графіки функцій зі знайденими (у середовищі Octave) мінімальними значеннями для випадку двох змінних.

Для подання функцій використано наступні позначення:

- $D$ – розмірність простору;
- $\mathbf{x}$ – вектор аргументів функції;
- $\mathbf{x}^*$ – вектор, у якому функція набуває мінімального значення;
- $\varepsilon$ – допустима точність обчислень.

1. Параболоїд (рис. 5).

$$f(\mathbf{x}) = \sum_{i=0}^{D-1} x_i^2 \,,\ -100 \le x_i \le 100 \,,\ i = 0,1,...,D-1;$$

$$f(\mathbf{x}^*) = 0 \,,\ x_i^* = 0 \,,\ \varepsilon = 10^{-6} \,.$$

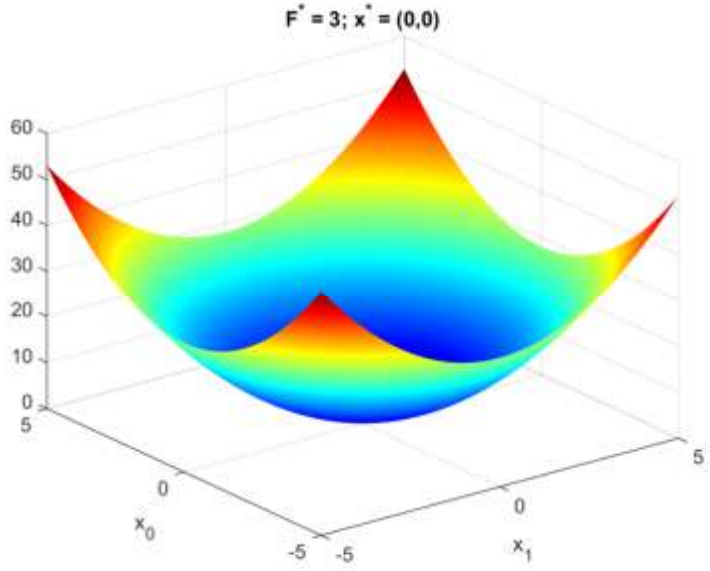



Рис. 5

2. Функція Розенброка (рис. 6).

$$f(\mathbf{x}) = \sum_{i=0}^{D-2} \left(100 \cdot \left(x_{i+1} - x_i^2\right)^2 + \left(x_i - 1\right)^2\right), \; -30 \le x_i \le 30, \; i = 0,1,\ldots,D-2;$$

$$f(\mathbf{x}^*) = 0, \; x_i^* = 1, \; \varepsilon = 10^{-6}.$$

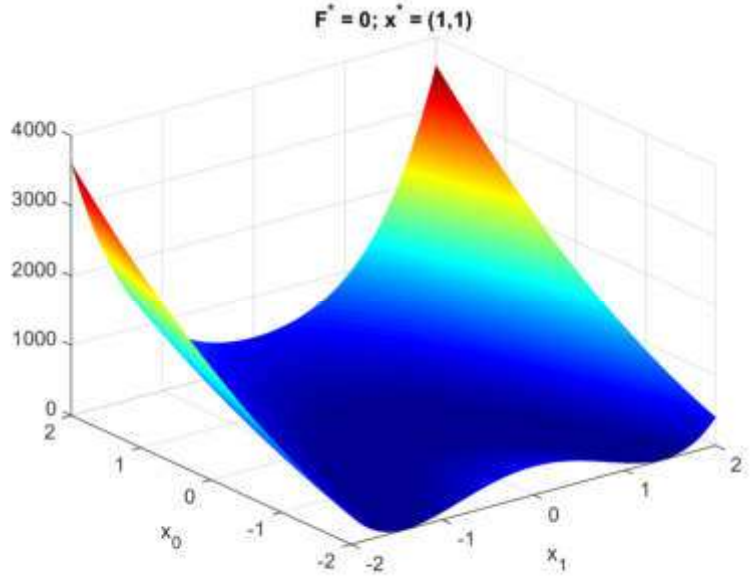



Рис. 6

3. Функція Швефеля (рис. 7).

$$f(\mathbf{x}) = \sum_{k=0}^{D-1} \left( \sum_{i=0}^{k} x_i \right)^2, \; -100 \le x_i \le 100, \; i = 0,1,\ldots,D-1;$$

$$f(\mathbf{x}^*) = 0, \; x_i^* = 0, \; \varepsilon = 10^{-6}.$$

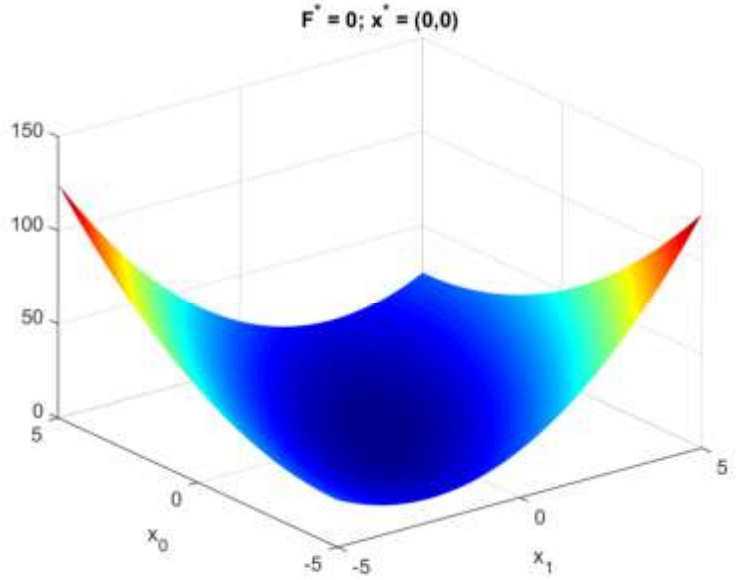



Рис. 7

4. Функція Еклі (рис. 8).

$$f(\mathbf{x}) = -20 \cdot \exp\left(-0.2 \cdot \sqrt{\frac{1}{D} \cdot \sum_{i=0}^{D-1} x_i^2}\right) - \exp\left(\frac{1}{D} \cdot \sum_{i=0}^{D-1} \cos(2\pi \cdot x_i)\right) + 20 + e,$$

$$-30 \le x_i \le 30,\ i = 0,1,...,D-1;$$

$$f(\mathbf{x}^*) = 0,\ x_i^* = 0,\ \varepsilon = 10^{-6}.$$

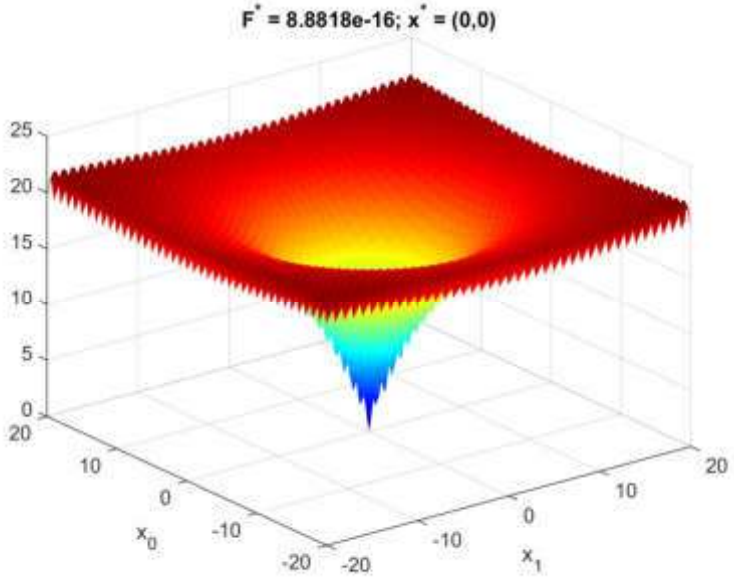



Рис. 8

5. Функція Грінвонка (рис. 9).

$$f(\mathbf{x}) = \frac{1}{4000} \cdot \sum_{i=0}^{D-1} x_i^2 - \prod_{i=0}^{D-1} \cos\left(\frac{x_i}{\sqrt{i+1}}\right) + 1,\ -600 \le x_i \le 600,\ i = 0,1,...,D-1;$$

$$f(\mathbf{x}^*) = 0,\ x_i^* = 0,\ \varepsilon = 10^{-6}.$$

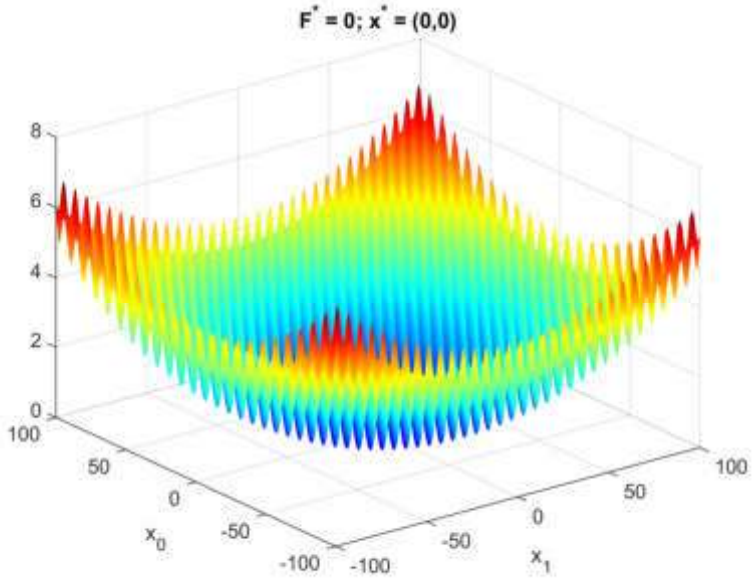



Рис. 9

6. Функція Растрігіна (рис. 10).

$$f(\mathbf{x}) = \sum_{i=0}^{D-1} \left( x_i^2 - 10 \cdot \cos(2\pi \cdot x_i) + 10 \right), \; -5.12 \le x_i \le 5.12, \; i = 0,1,...,D-1;$$

$$f(\mathbf{x}^*) = 0, \; x_i^* = 0, \; \varepsilon = 10^{-6}.$$

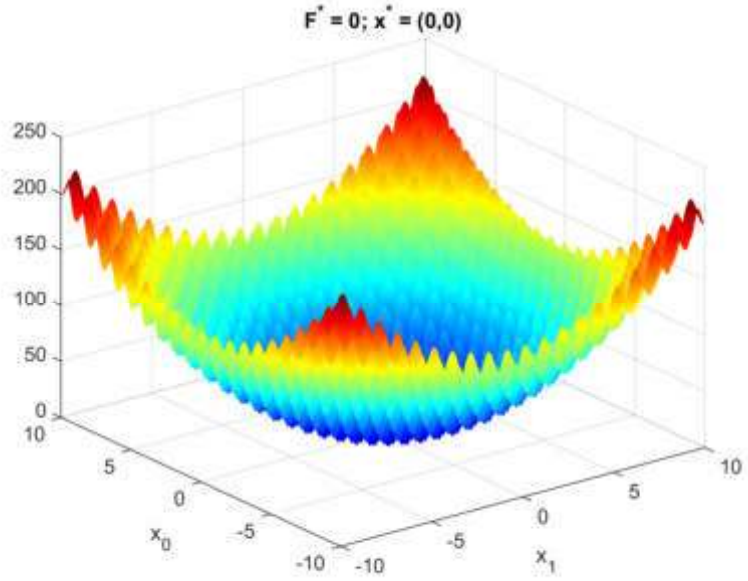



Рис. 10

7. Функція Міхалевіча (рис. 11).

$$f(\mathbf{x}) = \sum_{i=0}^{D-1} \sin(x_i) \cdot \left( \sin\left( \frac{i+1}{\pi} \cdot x^2 \right) \right)^{2m}, \; 0 \le x_i \le \pi, \; i = 0,1,...,D-1, \; m \in N.$$

У математичних поданнях функцій вказано рекомендовані проміжки для аргументів функцій, однак, використання інших проміжків теж цілком допустиме (як видно на рисунках та використано у розробленій до роботі програмі).

Зауважимо, що у функції Міхалевіча у роботі нами було використано значення $m = 10$.

З графіків функцій видно, що оптимальні значення деяких з них можуть відшукуватись і давновідомими оптимізаційними методами (наприклад, методом градієнтного спуску тощо). Це, наприклад, такі функції як параболоїд, функція Розенброка та функція Швефеля. Хоча функція Розенброка має 2 точки, у яких набуває мінімального значення (має два глобальні мінімуми), а функція Швефеля має надто повільне спадання вздовж певного вектора, що, без сумніву, затрудняє пошук глобального мінімуму функцій відомими методами.

Для інших функцій – внаслідок їх особливостей – практично неможливо знайти глобальний мінімум стандартними оптимізаційними методами, натомість, методи стохастичної оптимізації (зокрема, й ройові методи) дозволяють досить швидко наблизитись до глобального мінімуму.

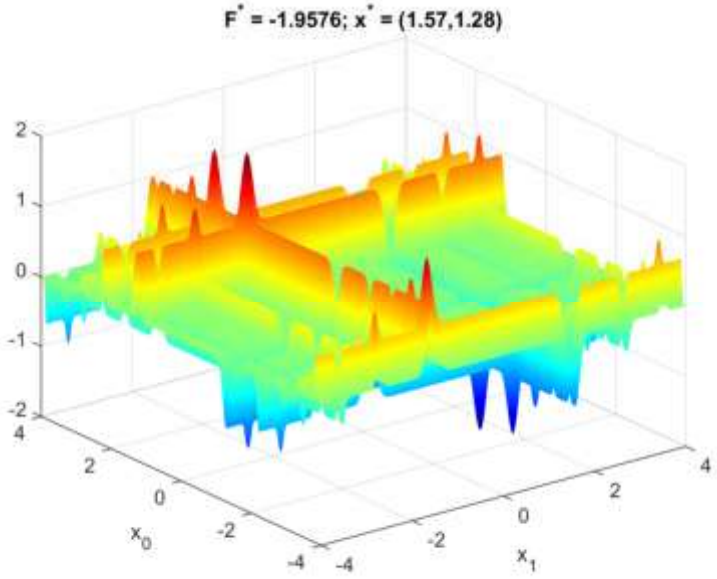



Рис. 11

**Робота з програмним продуктом**

Головне вікно програмного продукту, розробленого в межах роботи, подано на рис. 12.

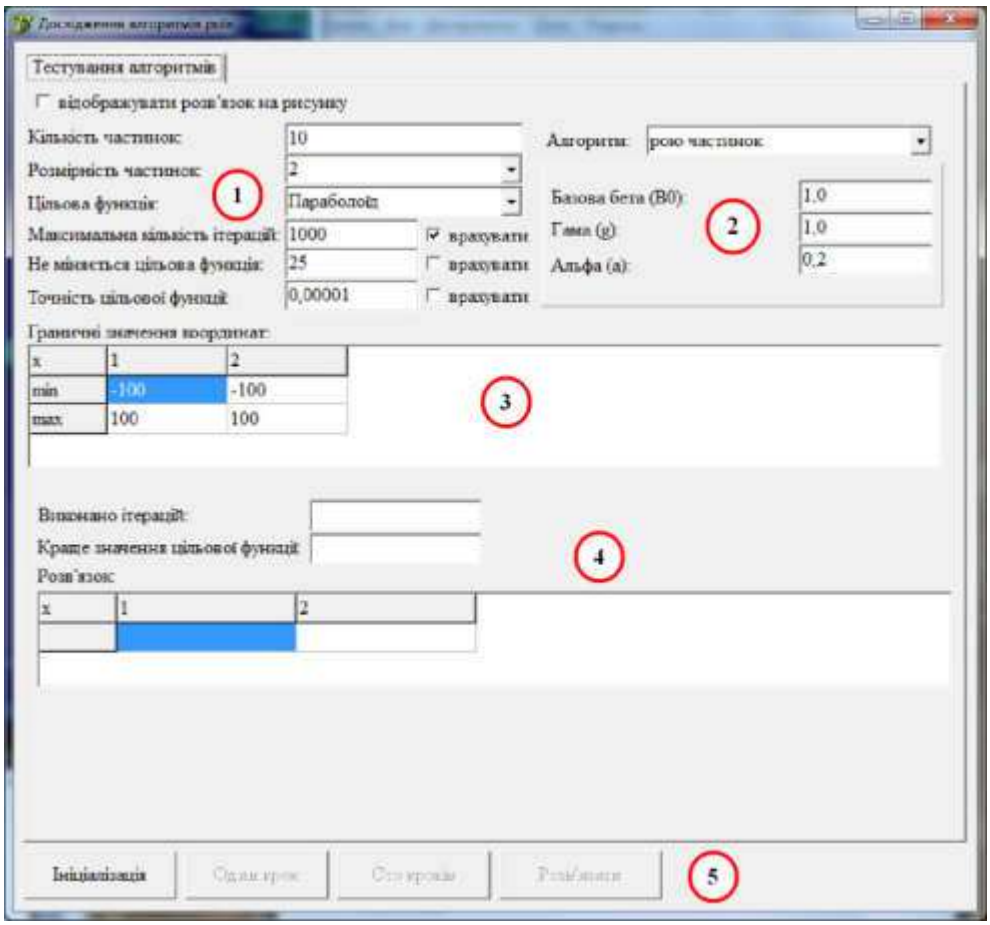



Рис. 12. Головне вікно програми

Робоча область вікна складається з кількох частин (на рис. 12 позначено номерами у кружечках).

1. Загальні параметри ройового алгоритму, що є спільними для обох досліджуваних алгоритмів: 1) кількість частинок – значення в полі вводиться з клавіатури; 2) розмірність частинок – реалізовано у вигляді випадного списку зі значеннями від 2 до 10 (таким чином, можна використовувати розмірності задач 2-10); 3) цільова функція – одна з поданих вище функцій 1-7, вибір функції реалізовано у вигляді випадного списку з назвами функцій; 4) максимальна кількість ітерацій – значення вводиться з клавіатури; 5) кількість ітерацій, протягом яких не повинне мінятись значення функції, щоб алгоритм припинив свою роботу – значення вводиться з клавіатури; 6) точність пошуку значення цільової функції – значення вводиться з клавіатури.

Очевидно, що поля 4, 5, 6 є критеріями зупинки виконання алгоритму, описаними у розділі 1. Для того, щоб якийсь критерій враховувався, необхідно включити бокс «врахувати», що знаходиться справа від критерія. Так, наприклад, на рис. 12 буде використовуватись лише перший критерій, а два інші – ігноруватимуться.

При запуску програми зазначені параметри мають початкові значення для того, щоб можна було відразу ж запускати алгоритм на обрахунок.

2. Спеціальні параметри обраного алгоритму. Поле «Алгоритм» є випадним списком з двома значеннями: «рою частинок» та «світлячків». При виборі зі списку конкретного значення нижче з'являються параметри, що відповідають обраному алгоритму. Наприклад, на рис. 12 відображено спеціальні параметри, що відповідають обраному у полі «Алгоритм» методу рою частинок.

3. Область «Граничні значення координат» служить для введення мінімального та максимального значення кожної координати простору, у якому проводиться пошук оптимального значення функції. У таблиці, що зображена нижче, міняється кількість координат при зміні значення поля «Розмірність частинок» (див. групу параметрів 1). Усі значення по замовчуванню встановлюються в «-100» для мінімальних значень та «100» для максимальних.

4. У даній області відображують проміжні результати під час виконання обраного алгоритму, а після його роботи – остаточний результат. У полях відображуються: 1) реальна кількість ітерацій, що було виконано алгоритмом для розв'язання задачі; 2) краще знайдене на поточний момент значення цільової функції; 3) значення кожної з координат для знайденого значення цільової функції. Усі поля у цій області неможливо редагувати, оскільки вони призначені лише для відображення результатів.

5. Дана область містить 4 кнопки. При запуску програми чи зміні будь-якого з параметрів доступною є лише кнопка «Ініціалізація», натиснення на яку дозволяє провести початковий налаштунок параметрів обраного алгоритму оптимізації в залежності від введених у полях областей 1-3 значень. Після проведення ініціалізації відкриваються інші кнопки.

Кнопка «Один крок» призначена для виконання однієї ітерації обраного алгоритму, після чого проміжні (чи кінцеві, якщо алгоритм закінчив свою роботу) результати відображуються у області 4.

Оскільки кількість ітерацій при розв'язування задач стохастичної оптимізації часто буває досить великою (сотні чи тисячі ітерацій), то додатково була створена

кнопка «Сто кроків», натиснення на яку приводить до виконання 100 ітерацій обраного алгоритму. У випадку, якщо критерій оптимізації виконується до повного проходження 100 кроків, обрахунок задачі припиняється і результат виводиться у область 4.

Нарешті, розв'язати задачу у повністю автоматичному режимі можна, натиснувши на кнопку «Розв'язати». Тоді робота процесу розв'язку припиниться лише при виконанні одного з обраних критеріїв. Зазначимо, що у програмі не реалізовувалась перевірка того, чи включений хоча б один з критеріїв зупинки обрахунку, тому виключення усіх трьох критеріїв призведе до зациклення процесу обрахунків.

Покажемо послідовність роботи з програмою.

Після запуску програми буде відображене вікно у вигляді, поданому на рис. 12. Використаємо обраний по замовчуванню алгоритм рою частинок для пошуку мінімуму параболоїда при використанні двох аргументів цільової функції. Натиснемо кнопку «Ініціалізація» (рис. 13).

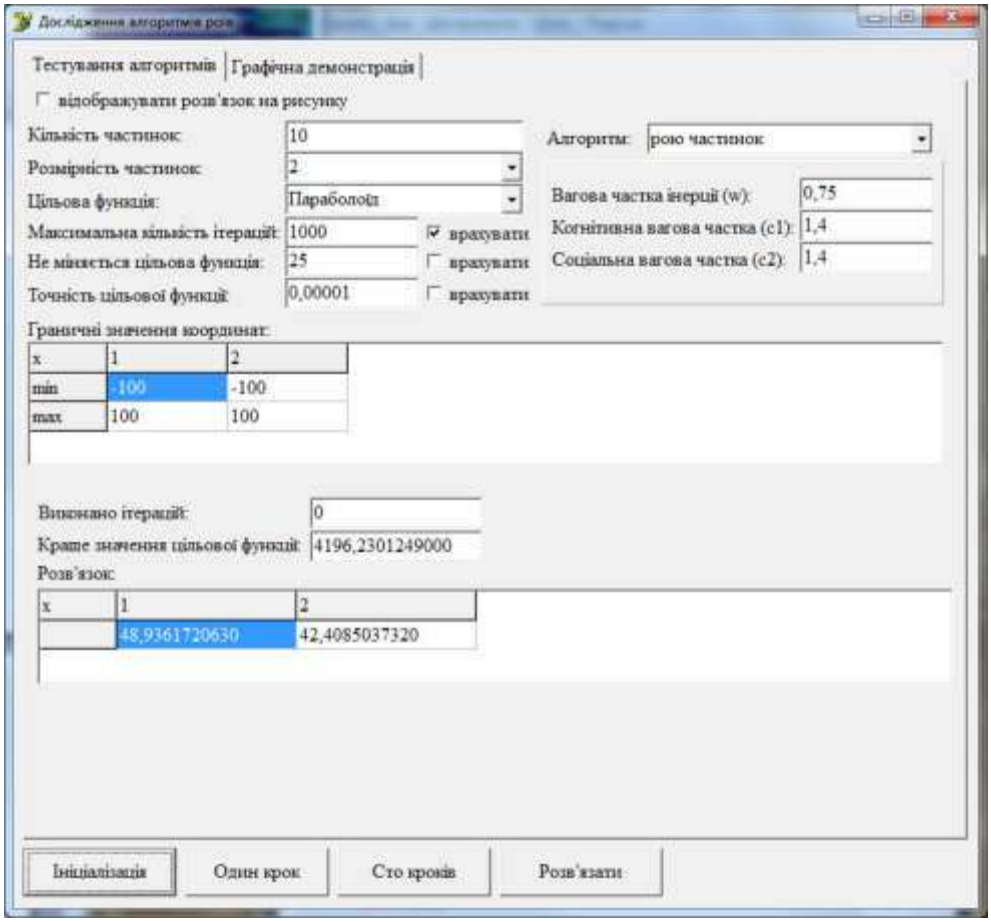



Рис. 13

Як видно з рисунка, у області 4 з'явились значення поточного найкращого розв'язку. Цей розв'язок, як пам'ятаємо з розділу 1.1 було обрано серед випадковим чином створених частинок.

Виконаємо 5 кроків алгоритму, послідовно натиснувши кнопку «Один крок» 5 разів (рис. 14).

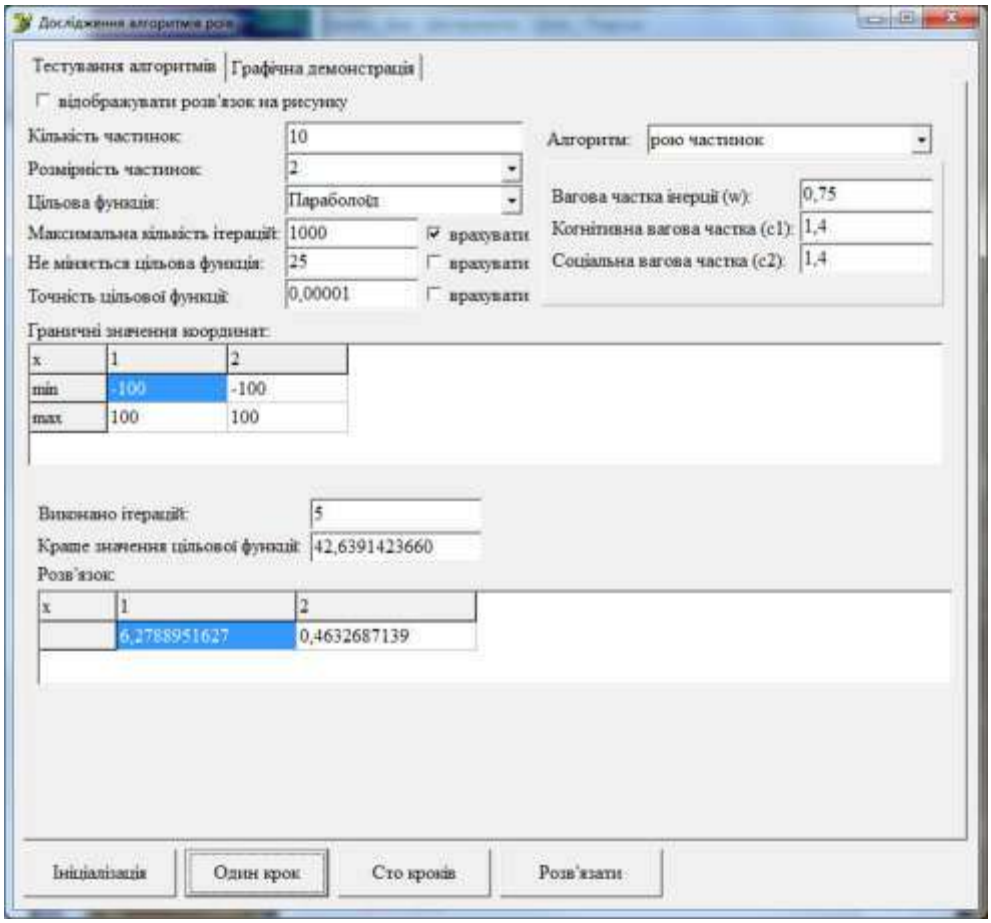



Рис. 14

Новий найкращий розв'язок, отриманий після 5 кроків, суттєво кращий, оскільки значення цільової функції для даного розв'язку, 42.6391, менше за початкове 4196.2301. У полі «Виконано ітерацій» відображена очікувана кількість виконаних ітерацій – 5.

Натиснемо кнопку «Сто кроків» для виконання наступних 100 ітерацій методу рою частинок (рис. 15).

З рисунка видно, що оптимальне значення цільової функції майже досягнуте (з точністю $10^{-10}$), при цьому значення $x_1 \approx 0$, $x_2 \approx 0$. Це свідчить про те, що метод уже знайшов потрібне значення, однак програма свою роботу не закінчила. Пояснюється це тим, що в якості критерія зупинки обрано лише критерій виконання певної кількості ітерацій – 1000, а тому допоки програма не виконає потрібну кількість ітерацій, вона не зупиниться.

Закінчимо роботу методу, натиснувши кнопку «Розв'язати» (рис. 16).

По закінченні роботи на екрані з'явиться вікно з повідомленням про закінчення роботи та буде вказано критерій, який виконався для зупинки методу. Для досліджуваного прикладу метод закінчив роботу при значенні цільової функції $F^* = 0$ та розв'язку $(0,0)$ (враховуючи похибку подання значень у ПК).

Додатково для графічного відображення послідовності пошуку розв'язку реалізовано відображення профіля цільової функції та частинок. Це відображення знаходиться на вкладці «Графічна демонстрація», яка з'являється після проведення ініціалізації алгоритму. У поданому вище прикладі вкладка з'явилась вже на рис. 13 (після натиснення кнопки «Ініціалізація»).

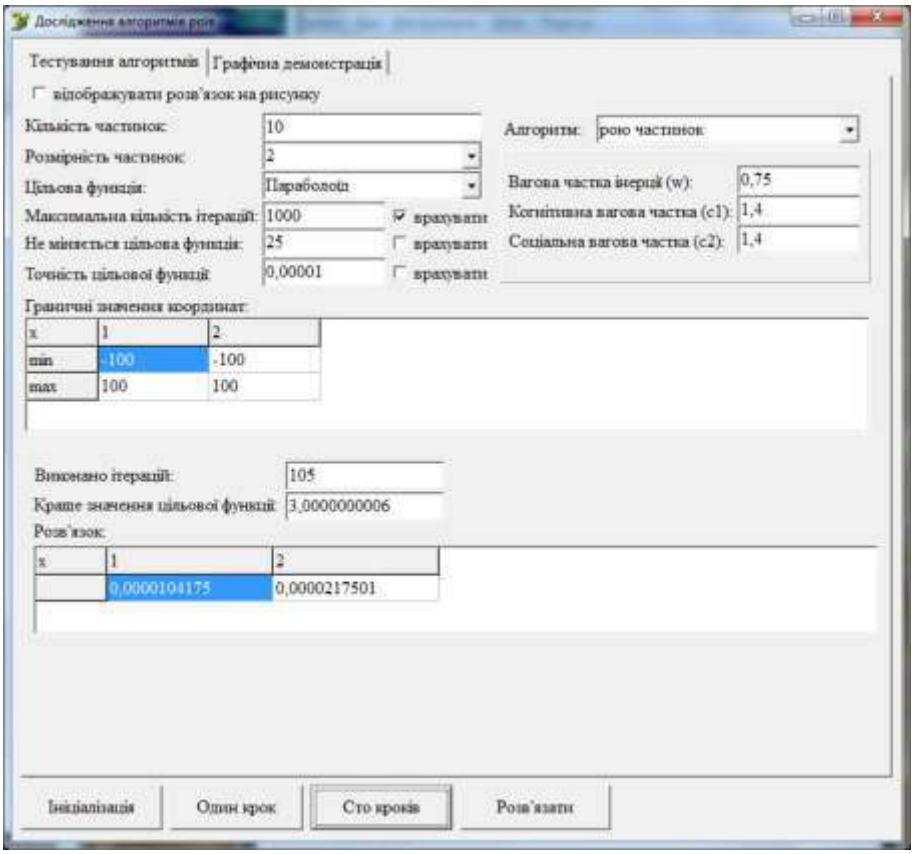



Рис. 15

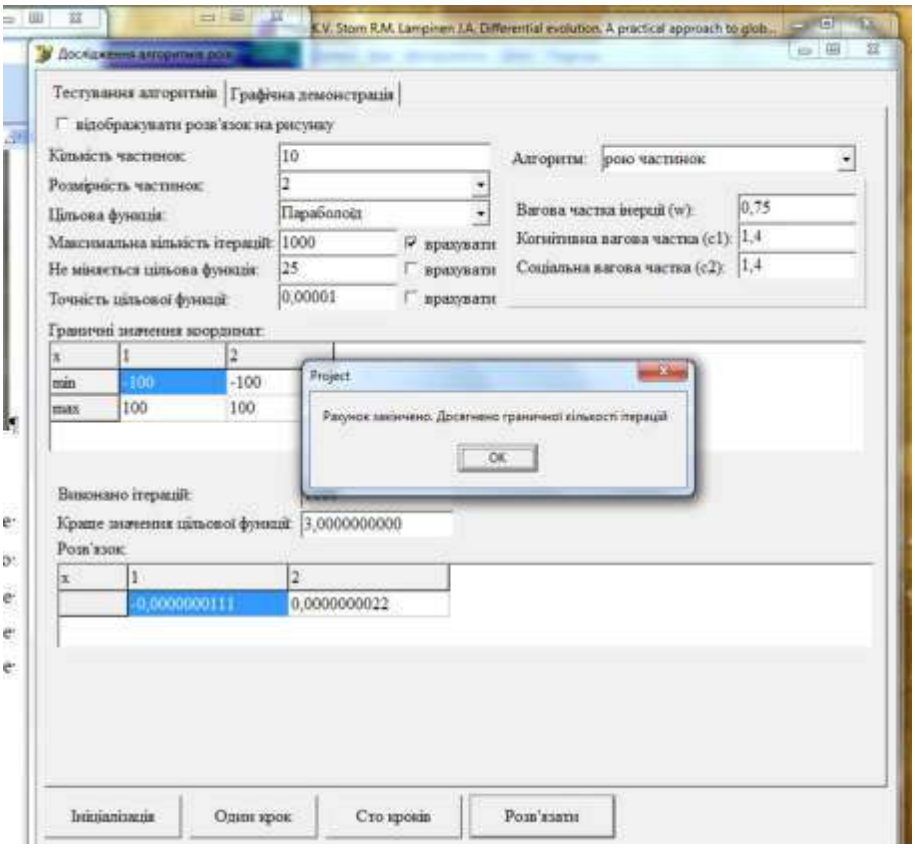



Рис. 16

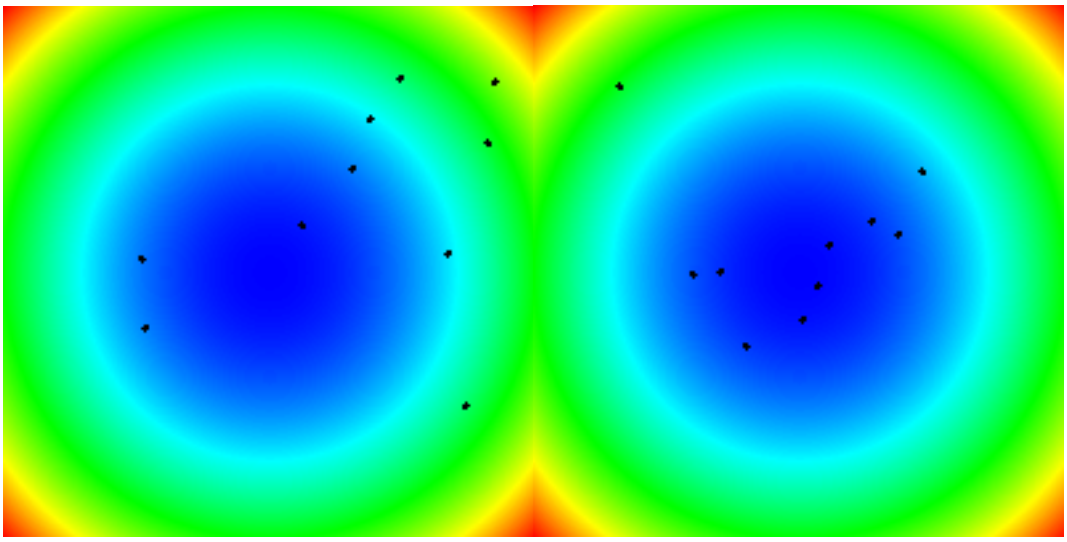
На вкладці відображується кольорова карта функції, створена для перших двох

координат. Тобто, якщо використовується більше 2-х координат, карта буде відображуватись, однак, братимуться до уваги завжди лише перша та друга координата. На карті кольором позначається значення цільової функції у кожній точці досліджуваної області за наступним правилом: найменші значення цільової функції відображують темно-синім кольором, найбільші – яскраво-червоним. Усі проміжні значення відображуються іншими кольорами та відтінками спектру.

Як і кнопки виконання кроків задачі, вкладка приховується при зміні будь-якого з параметрів алгоритмів і з'являється після проведення ініціалізації.

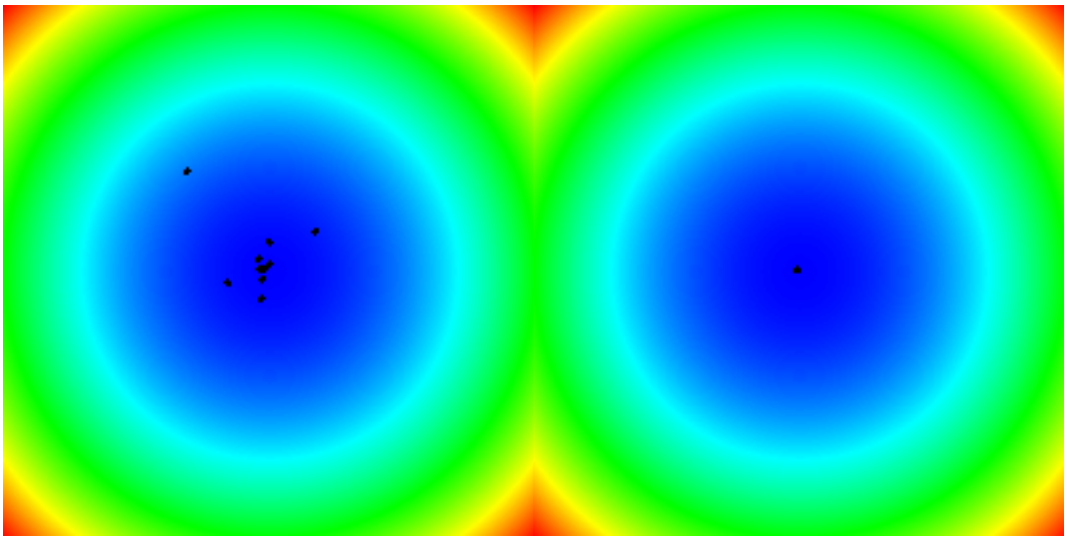
На вкладці знаходиться кнопка «Зберегти зображення», при натисненні на яку відкривається діалог вибору директорія та імені файлу, куди й зберігається отриманий графік. Формат файлів – bmp.

На рис. 17-18 зображено приклад розташування частинок для тієї ж задачі, що розглядалась вище, після ініціалізації (рис. 17а), 5 кроків (рис. 17б), 15 кроків (рис. 18а) та повного розв'язку (рис. 18б).



а б

Рис. 17



а б

Рис. 18

Як видно з рис. 17-18, частинки поступово наближаються до точки глобального

мінімуму у процесі розв'язання задачі.

Сам процес можна покроково відстежувати під час роботи програми (використовуючи кнопку «Один крок»).

**Результати застосування алгоритмів**

Дослідимо здатність алгоритмів знаходити глобальні мінімуми описаних вище функцій. Для методу рою частинок використовуються рої з 15 частинками, а для алгоритму світлячків – використовується 30 особин у рої. Параметри алгоритмів використовуються встановлені за замовчуванням. Розглядаються цільові функції від 2-х аргументів.

Для кожної функції та кожного методу проводиться 5 спроб і середній результат записується у таблицю. В якості критеріїв зупинки використана точність $\varepsilon = 0.0001$ та кількість ітерацій, протягом яких не міняється значення функції, рівна 100.

Для аналізу бралась кількість ітерацій, використана алгоритмом до виконання критерія зупинки, та знайдене оптимальне значення функції.

Отримані результати подано у таблиці 1.

Таблиця 1

Усереднені результати виконання алгоритмів пошуку глобального мінімуму функції

| Функція | Метод рою частинок | | Алгоритм світлячків | |
|---|---|---|---|---|
| | ітерацій | $F^*$ | ітерацій | $F^*$ |
| Параболоїд | 80 | 3.0001 | 9 | 3.0000 |
| Міхалевіча | 202.8 | -1.8029 | 170.4 | -1.9631 |
| Розенброка | 106.6 | 0.0990 | 546.4 | 0.0000 |
| Швефела | 59.4 | 0.0001 | 8.4 | 0.0000 |
| Еклі | 78 | 0.0052 | 20.2 | 12.9237 |
| Грінвангка | 18.6 | 1.0005 | 4.4 | 1.0000 |
| Растрігіна | 96.4 | 0.0011 | 55.2 | 20.6950 |

На основі таблиці 1 можна зробити наступні висновки.

1. Обидва методи знайшли точний розв'язок для функції параболоїда, при цьому у алгоритмі світлячків було використано менше ітерацій.
2. Для функції Міхалевіча точніше значення цільової функції (й за меншу кількість ітерацій) було знайдене алгоритмом світлячків.
3. Для функції Розенброка кращі результати при достатній точності показав метод рою частинок, оскільки алгоритм світлячків затратив уп'ятеро більше ітерацій.
4. Мінімум для функції Швефела точніший і швидше знайшов алгоритм світлячків.
5. Метод рою частинок знайшов майже точний розв'язок для функції Еклі, в той час як алгоритм світлячків абсолютно не справився з цією задачею.
6. Обидва алгоритми не змогли знайти мінімум для функції Грінвагка.
7. Алгоритм світлячків не зміг знайти й мінімум функції Растрігіна, в той час як за допомогою методу рою частинок знайдено задовільний розв'язок.

Отже, загалом обидва реалізовані методи показали спроможність розв'язувати оптимізаційні задачі, однак, результати перевірки їх роботи для параметрів по замовчуванню свідчать про необхідність підбору параметрів для кожної задачі, оскільки різні цільові функції вимагають різної поведінки частинок рою при пошуку глобального мінімуму (наприклад, менша інерційність чи більша швидкість руху до кращого світлячка). Тому наступним кроком у дослідженні описаних у роботі та інших ройових алгоритмів може бути аналіз параметрів та підбір рекомендацій до способу

розв'язку та значень параметрів ройових методів в залежності від виду цільової функції.

**Висновки**

У роботі розглянуто два ройові методи оптимізації: метод рою частинок та алгоритм світлячків. Для перевірки роботи розглянутих методів розроблено програму з графічним інтерфейсом користувача для чисельного розв'язування задач безумовної оптимізації.

Розроблена програма написані у середовищі Lazarus, що дозволяє досить просто розробляти програми, що мають графічний інтерфейс, а мова програмування Object Pascal, якою реалізована програма, є нескладною для розуміння, однак, потужною для створення складних програм з великою кількістю обрахунків.

Отримані результати аналізу роботи реалізованих методів співпали з очікуваними і показали дієвість обраних алгоритмів при розв'язанні задач безумовної оптимізації навіть для складних функцій, однак, застосування розглянутих методів вимагає більш детального розгляду способів вибору їх параметрів для кожної конкретної задачі.

Одним із можливих подальших напрямків роботи може бути використання методу рою частинок та алгоритму світлячків саме для підбору параметрів цих методів при розв'язанні задач оптимізації.

**Список використаної літератури:**


1. Письменная В.А [Электронный журнал] // Алгоритмическое и программное обеспечение меметического алгоритма поиска условного глобального экстремума.
2. Ахмедова Шахназ Агасувар кызы КОЛЛЕКТИВНЫЙ САМОНАСТРАИВАЮЩИЙСЯ МЕТОД ОПТИМИЗАЦИИ НА ОСНОВЕ БИОНИЧЕСКИХ АЛГОРИТМОВ – ДИССЕРТАЦИЯ на соискание ученой степени кандидата технических наук / Ахмедова – Красноярск , 2016 - с.150
3. SWARM INTELLIGENCE Principles, Advances, and Applications Aboul Ella Hassanien Eid Emary – 2016.- p 220 140 4 75 101 139 99
4. Karaboga, D. An Idea Based On Honey Bee Swarm for Numerical Optimization. Technical Report-TR06, Erciyes University, Engineering Faculty, Computer Engineering Department 2005.
5. Ахмедова, Ш.А. Исследование эффективности «стайного» алгоритма для задач многокритериальной оптимизации. Молодежь и наука: сборник материалов VIII Всероссийской научно-технической конференции студентов, аспирантов и молодых ученых, посвященной 155-летию со дня рождения К.Э. Циолковского, 2012.
6. Xin-She Yang Nature-Inspired Optimization Algorithms / 32 Jamestown Road, London NW1 7BY/225 Wyman Street, Waltham, MA 02451, USA. – 2014. –265 p
7. Kennedy J, Eberhart R. Particle swarm optimization. In: Proceedings of the IEEE International Conference on Neural Networks, Piscataway, NJ, USA; 1995. p. 1942–48.
8. Скобцов Ю.А., Федоров Е.Е. С 44 Метаэвристики: монография / Ю.А. Скобцов, Е.Е. Федоров. – Донецк: Изд-во «Ноулидж» (Донецкое отделение), 2013. – 426 с.
9. K.N. Krishnanand Glowworm swarm based optimization algorithm for multimodal functions with collective robotics applications/ Debasish Ghose// Multiagent and Grid Systems – An International Journal..- 2006.-p.209-222
10. Орловская Н.М. Анализ эффективности биоинспирированных методов глобальной оптимизации // Электронный журнал «Труды МАИ». - №73.-с.1-22
11. Microsoft [Електронний ресурс] // Тесты - Оптимизация по алгоритму светлячков .- Режим доступу: https://msdn.microsoft.com/ru-ru/magazine/mt147244.aspx



**SERDIUK Oleksandr,**
The Bohdan Khmelnytsky National University of Cherkasy, PhD, Senior Lecturer
**OSADCHA Bohdana,**
student of Drabivskyi NVK "Secondary school of I-III degrees named after S.V. Vasylchenko - gymnasium "of Drabiv district council

## IMPLEMENTATION AND COMPARATIVE ANALYSIS OF THE PARTICLES SWARM METHOD AND FIREFLIES ALGORITHM

*The paper considers two swarm optimization methods: the particle swarm method and the firefly algorithm. To test the operation of the considered methods, a program with a graphical user interface for numerical solution of unconditional optimization problems has been developed.*

*The developed programs are written in the Lazarus environment, which allows the one to easily develop programs with a graphical interface, and the programming language Object Pascal, which implements the program, is easy to understand, but powerful for creating complex programs with many calculations.*

*The results of the analysis of the implemented methods coincided with the expected ones and showed the effectiveness of the selected algorithms in solving unconditional optimization problems even for complex functions, however, the application of these methods requires more detailed consideration of how to select their parameters for each problem.*

*One of the possible further directions of work may be the use of the particle swarm method and the firefly algorithm to select the parameters of these methods when solving optimization problems.*

***Keywords:*** *swarm optimization methods, software development.*

*Стаття надійшла 07.08.2018*
*Прийнято до друку 20.09.2018*

## ЗМІСТ

### СЕКЦІЯ «ПРИКЛАДНА МАТЕМАТИКА»



### СЕКЦІЯ «ІНФОРМАТИКА»

## CONTENTS

### APPLIED MATHEMATICS SECTION



### INFORMATICS SECTION