Київський національний університет імені Тараса Шевченка
Факультет радіофізики, електроніки та комп'ютерних систем
Кафедра комп'ютерної інженерії

«Системи та методи візуалізації фізичних розрахунків на прикладі дисперсії зворотних об'ємних магнітостатичних хвиль в структурі метал-діелектрик-ферит-діелектрик-метал»

Кваліфікаційна робота бакалавра
студентки 4 року навчання
спеціальності 123 «Комп'ютерна інженерія»
Юлії БАЗЕНКО
_____(підпис)

Науковий керівник,
доц. канд.ф.-м.н.
Олексій НЕЧИПОРУК
_____(підпис)

Рецензент
доц. канд.ф.-м.н.

_____(підпис)

До захисту допускаю

Завідувач кафедри
к.ф.-м.н., доцент
Юрій БОЙКО

КИЇВ - 2022

РЕФЕРАТ

Кваліфікаційна робота бакалавра з комп’ютерної інженерії: – 48 с., 7 рис., 1 додаток, 7 джерел

Аналізується дисперсія зворотних об’ємних магнітостатичних хвиль (ЗОМСХ) в багатошарових структурах. Отримано та проаналізовано відповідне дисперсійне співвідношення. Для аналізу використовується метод комп’ютерного моделювання. Створено програму для моделювання дисперсії зворотних об’ємних магнітостатичних хвиль при довільному куті поширення в структурі метал-діелектрик-ферит-діелектрик-метал (МДФДМ). Наводено приклади моделювання дисперсії хвиль в тривимірному просторі хвильових чисел та частот.

КЛЮЧОВІ СЛОВА: зворотні об’ємні магнітостатичні хвилі, дисперсія, комп’ютерне моделювання.

ЗМІСТ

ВСТУП

Науково-технійчний прогрес на теперішньому етапі потребує складну радіоелектронну апаратуру з надійною економічністю. Тому великою популярністю користуються монокристалічні плівки ферит-гранатів. Їх застосовують в широкому спектрі електронних приладів, в мікроелектроніці, надвисокочастотній техніці.

Магнітностатичні хвилі (МСХ) в феритових кристалах порівняно з іншими типами елементарних збуджень твердих тіл мають велике різноманіття лінійних і нелінійних властивостей, дисперсійних характеристик, які керуються різними методами. Плівки залізо-ітрієвого гранату (ЗІГ) $Y_3Fe_5O_{12}$ мають однорідний розподіл внутрішнього магнітного поля та невеликі втрати на поширення та перетворення в НВЧ діапазоні.

Дослідження тонких плівок не зменшується на технологіях рідинно-фазової епітаксії ферит-гранатів та інших технологій формування гетероструктур, через їх фізичні властивості. Для можливості реалізувати планарні пристрої та обробку сигналів на несучій частоті та в реальному масштабі часу створюють складні планарні структури, наприклад, такі як метал-діелектрик-ферит-діелектрик-метал (МДФДМ). Їх коливальні та хвильові властивості зручно моделюють сучасними аналітичними та чисельними методами.

Через фізичні властивості МСХ на їх основі можна створювати різноманітні прилади НВЧ-електроніки (конвольвери, лінії затримки, фільтри, обмежувачі потужності) з великою кількістю варіантів зміни параметрів. Затримкою МСХ можна керувати використанням башатошарових ферит діелектричних структур та завдяки неоднорідності зовнішнього магнітного поля. Тому залишається перспективним є дослідження та використання МСХ у неоднорідних феритових плівках у яких буде неперервна зміна матеріальних параметрів по товщині плівки.

Використвуючи МСХ в феритових структурах потрібна точність їх характеристик з реальними параметрами. Щоб підібрати ту чи іншу планарну структуру треба можливість швидко побачити її залежність тих чи інших характеристик з різними параметрами. Тому метою данної роботи є дослідити дисперсію ЗОМСХ в структурі метал-діелектрик-ферит-діелектрик-метал, і провести її моделювання в тривимірному просторі частот та хвильових чисел.

1. Дисперсійні характеристики МСХ в феритових структурах

1.1 Магнітостатичні спінові хвилі

Власними хвилями в необмеженому феромагнітному середовищі завжди є Н-хвилі, які в деяких частинних випадках можуть переходити в хвилі Т-типу. Вектор електричного поля ℓ завжди є перпендикулярним до напрямку поширення хвилі. В спектрі власних електромагнітних хвиль в необмеженому феромагнетику існують хвилі, які обмежено зверху частотою

$$\omega = \sqrt{\omega_H(\omega_H + \omega_M)}.$$

Вони характеризуються низькими в порівнянні з іншими типами хвиль значеннями групової і фазової швидкостей. Ці хвилі називають магнітостатичними спіновими хвилями (МСХ). Існує можливість в достатньо широких межах керувати спектром МСХ, змінюючи величину зовнішнього постійного магнітного поля і намагніченостістю фериту $\vec{M}$. Спектр МСХ необмеженого феритового середовища знаходиться всередині смуги, яку займають хвилі, що поширюються під різними кутами $0 \leq \theta \leq \pi/2$ по відношенню до вектора магнітного моменту $\overline{M_0}$

В магнітогіротропному середовищі одним з різновидів електромагнітних хвиль є МСХ.Область існування магнітостатичних хвиль знаходиться між, з одного боку, областю звичайних електромагнітних хвиль, а з іншого - обмінних спінових хвиль. Тобто хвильове число МСХ змінюється в межах

$$k_{EMX} < k_{MCX} < k_{OCX}$$

Напруженість ефективного обмінного магнітного поля записується у вигляді:

$$\vec{H}_{обм} = \alpha_0 \Delta\vec{M}(\vec{r},t) = \alpha_0 \Delta\vec{M}_0(\vec{r}) + \alpha_0 \Delta\tilde{m}(\vec{r},t),$$

де α_0 - обмінна константа. Величину $l_0 = \sqrt{\alpha_0}$ - обмінна довжина. Наприклад, для кристала ЗІГ ця величина складає $l_0 \approx 6 \cdot 10^{-6}$см при кімнатній температурі.

Якщо феромагнетик є однорідним, то статичний доданок дорівнює нулю, і тоді вплив обмінних ефектів на поширення МСХ вважається малим за умови, що довжина хвилі $\lambda_{МСХ}$ і характерний розмір феріту (або ж відповідне хвильове число МСХ) задовольняють співвідношенням:

$$\lambda_{MCX},\ s >> l_0$$

або

$$k_{MCX} << 2\pi / l_0$$

Тобто, МСХ не мають відчувати впливу короткодіючих обмінних сил. Взагалі ж при виконанні цієї умови в реальних феромагнетиках немає необхідності враховувати вплив обміну на дисперсію МСХ. Отже, критерій безобмінного наближення буде наперед виконуватися для кристала ЗІГ за умови $\lambda_{МСХ} \geq 6 \cdot 10^{-4}$см та, відповідно, $k_{МСХ} \leq 10^4$ см$^{-1}$

1.2 МСХ в ізольованому феритовому шарі

Вперше в нескінченній феромагнітній пластинці в рамках магнітостатичного наближення дослідження поверхневих та об'ємних МСХ згадується в роботі [3].

При розв'язанні рівняння Уокера, використовуючи магнітостатичне наближення, розглянемо ті МСХ, що поширюються під довільним кутом θ_k до $\vec{M}_0$ в феритовому дотично-намагніченому шарі (рис.1.1).

Отримуємо багатошарову структуру діелектрик-ферит-діелектрик, для якої, завдяки різному вигляду тензора магнітної проникності рішення рівнянь Уокера відрізняються для кожного із шарів структури.

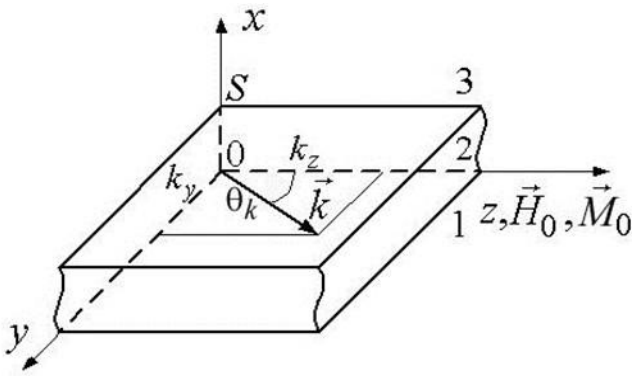

Рис.1.1 Необмежений Дотично-намагнічений феритовий шар. Будемо

вважати, що таким тензором описуються магнітні властивості діелектричних шарів:

$$\mu_{ij} = \delta_{ij}$$, (δ_{ij} - символ Кронеккера).

МСХ в цій структурі описується такою системою рівнянь :

$$\Psi = \begin{cases} \Psi_1, & x < 0 \quad (\textit{область } 1), \\ \Psi_2, & 0 < x < s \quad (\textit{область } 2), \\ \Psi_3, & x > s \quad (\textit{область } 3). \end{cases} \tag{1.1}$$

Тоді задача зводиться до розв'язку рівнянь:

$$\begin{cases} \Delta\Psi_1 = 0, \ x < 0; & (1.2) \\ \mu\left(\dfrac{\partial^2\Psi_2}{\partial x^2} + \dfrac{\partial^2\Psi_2}{\partial y^2}\right) + \dfrac{\partial^2\Psi_2}{\partial z^2} = 0, \quad 0 < x < s; & (1.3) \\ \Delta\Psi_3 = 0, \quad x > s; & (1.4) \end{cases}$$

Рівняння (1.3) в залежності від знака параметра μ μ μ може бути як еліптичного ($\mu > 0$ $\mu > 0$ $\mu > 0$), так і гіперболічного типу ($\mu < 0$ $\mu < 0$ $\mu < 0$) а рівняння (1.2) і (1.4) завжди є еліптичного типу. При розв'язанні рівняння (1.3) являють собою поверхневі хвилі, а при $\mu < 0$ $\mu < 0$ $\mu < 0$ – зворотні об'ємні хвилі. Це розглянуто докладніше в роботі [4].

Шукаючи розв'язки рівнянь (1.2) – (1.3), які знаходяться у вигляді неоднорідних плоских хвиль підставляємо хвильовий вектор $\vec{k} = (0, k_y, k_z)$ розташований у площині ZY:

$$\Psi_i = X_i e^{i(\omega t - k_y y - k_z z)}, i = 1,3. \tag{1.5}$$

Якщо підставимо вирази (1.4) у рівняння (1.2) – (1.3) , то побачимо, що це зводиться до диференційних рівнянь з постійними коефіцієнтами:

$$\frac{dX_i}{dx^2} - k_s^2 X_i = 0, \ \ i = 1,3, \tag{1.6}$$

$$\frac{d^2 X_2}{dx^2} - \kappa^2 X_2 = 0, \tag{1.7}$$

де

$$k_s = \sqrt{k_y^{\,2} + k_z^{\,2}}, \kappa = \sqrt{k_y^{\,2} + \frac{k_z^{\,2}}{\mu}}, k_y = k\sin(\theta_k), k_z = k\cos(\theta_k) \qquad (1.8)$$

1.3 Поверхневі МСХ

Ті розв'язки рівнянь (1.3), що відповідають дійсним та додатним величинам сталої поширення κ є поверхневими МСХ (ПМСХ). Це справджується в діапазоні частот $\omega_H < \omega < \omega_1$ $\omega_H < \omega < \omega_1$ $\omega_H > \omega > \omega_1$ за умови ($\mu < 0$ $\mu < 0$ $\mu > 0$). Розглянемо також випадок ізольованого феритового шару (рис.1.2).

Якщо враховувати обмеженість потенціалів ψ_1 ψ_1 ψ_1 ψ_1 і ψ_3 магнітостатичних хвиль на нескінченності, отримуємо вирази для магнітостатичних потенціалів:

$$\Psi_1 = Ae^{k_s x + i(\omega t - k_y y - k_z z)} \qquad (1.9)$$

$$\Psi_2 = (B\ \mathrm{ch}\ \kappa x + C\ \mathrm{sh}\ \kappa x)e^{i(\omega t - k_y y - k_z z)} \qquad (1.10)$$

$$\Psi_3 = De^{-k_s x + i(\omega t - k_y y - k_z z)} \qquad (1.11)$$

Щоб одержати дисперсійне співвідношення для магнітостатичних хвиль, використаємо умови компонент магнітної індукції на межах середовищ та граничні умови неперервності тангенційних компонент напруженості магнітного поля. Звідки отримаємо вирази:

$$\Psi_1\big|_{x=0} = \Psi_2\big|_{x=0},$$
$$\Psi_2\big|_{x=s} = \Psi_3\big|_{x=s},$$

та

$$\frac{\partial \Psi_1}{\partial x}\Big|_{x=0} = \mu \frac{\partial \Psi_2}{\partial x}\Big|_{x=0} - i\mu_a \frac{\partial \Psi_2}{\partial y}\Big|_{x=0},$$
$$\mu \frac{\partial \Psi_2}{\partial x}\Big|_{x=s} - i\mu_a \frac{\partial \Psi_2}{\partial y}\Big|_{x=s} = \frac{\partial \Psi_3}{\partial x}\Big|_{x=s},$$

Звідки можемо дістати систему рівнянь, однорідну відносно констант A, B, C, D, після підстановки сюди виразів для магнітостатичних потенціалів (1.9)-(1.11)

$$\begin{cases} A = B \\ De^{-k_s S} = B\ \mathrm{ch}\ \kappa S\ + C\ \mathrm{sh}\ \kappa S \\ Ak_s = C\mu\kappa - B\mu_a k_y \\ -Dk_s e^{-k_s S} = -B\mu\kappa\ \mathrm{sh}\ \kappa S + C\mu\kappa\ \mathrm{ch}\ \kappa S - \\ -\mathrm{B}\mu_a k_y\ \mathrm{ch}\ \kappa S\ - \mathrm{C}\mu_a k_y\ \mathrm{sh}\ \kappa S \end{cases} \qquad (1.12)$$

Власні хвилі є нетривіальнимим розв'язками системи (1.12), тобто, її визначник має дорівнювати нулю. Розкривши його, виведемо дисперсійне співвідношення в неявній формі:

$$2\mu\kappa k_s \mathrm{cth}\ \kappa S + k_s^2 - \mu_a^2 k_y^2 - \mu^2\kappa^2 = 0 \qquad (1.13)$$

Якщо підставити замість сталої поширення κ її вираз (1.8), то можемо записати дисперсійне співвідношення в такій формі:

$$2k_s\sqrt{\mu(k_z{}^2 + \mu k_y{}^2)}\mathrm{cth}\ \sqrt{k_y{}^2 + \frac{k_z{}^2}{\mu}s + \mu(k_z{}^2 + \mu k_y{}^2)} + k_s^2 - \mu_a^2 k_y^2 = 0, \qquad (1.14)$$

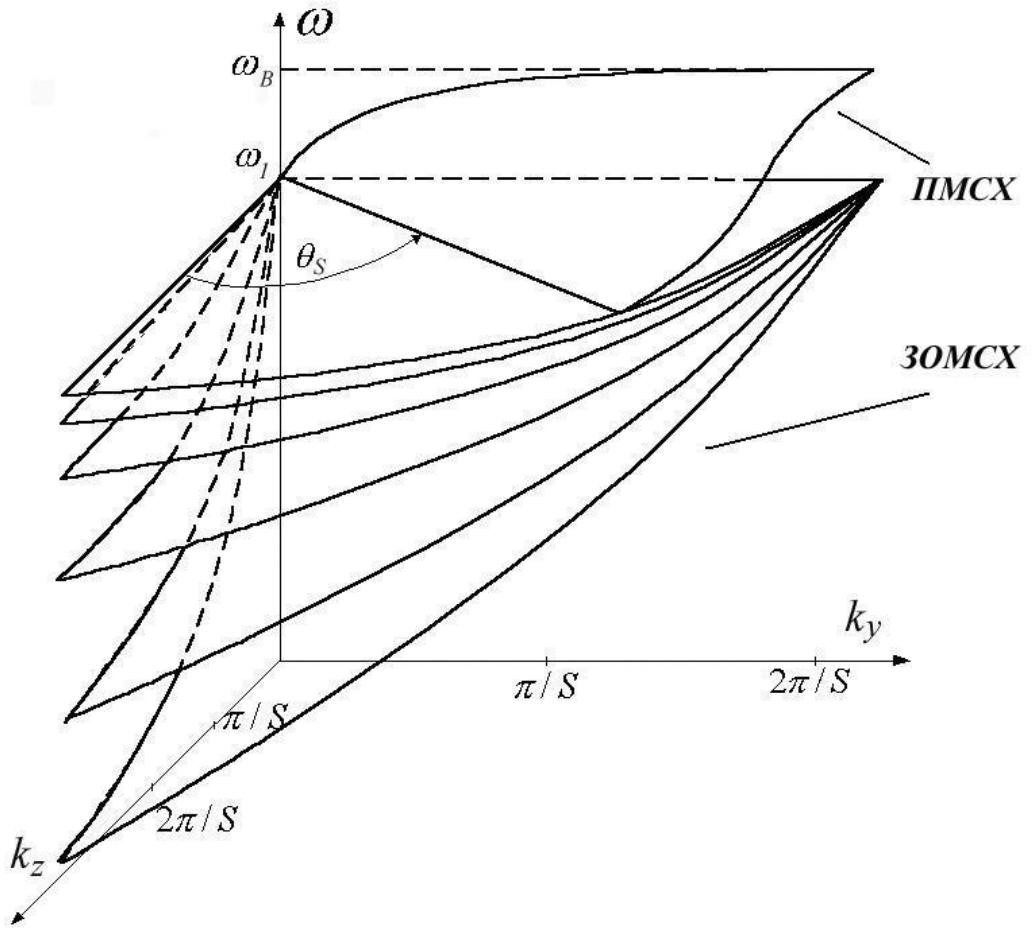

Рис.1.2 Спектр магнітостатичних хвиль у дотично-намагніченому феритовому шарі.

Аналізуючи співвідношення (1.14) можемо побачити (рис.1.3), що в в області частот $\omega > \omega_H$ у цього рівняння немає дійсних коренів , тобто спектр поверхневих МСХ (ПМСХ) знаходиться в області $\omega > \omega_1$. Та
верхня частотна межа цього спектру дорівнює:

$$\omega_B = \omega_H + \frac{\omega_M}{2}. \qquad (1.15)$$

Цей вираз (1.15) відповідає хвилям, що поширюються перпендикулярно до вектора M_0 (тобто, $\theta_k = \pi / 2$). Якщо вважаючи, що $k_z = 0$ (та,

відповідно, $\partial\Psi/\partial z = 0$), для тих хвиль з рівняння (1.14) ми можемо отримати дисперсійне співвідношення:

$$2\mu \operatorname{cth} \left|k_y\right| s = \mu_a^2 - \mu^2 - 1 \qquad (1.16)$$

Враховуючи явний вид компонента тензору магнітної проникності цей вираз (1.16) можна звести до:

$$\omega_k^2 = \left(\omega_H + \frac{\omega_M}{2}\right)^2 - \left(\frac{\omega_M}{2}\right)^2 e^{-2\left|k_y\right| s}. \qquad (1.17)$$

Поверхневі хвилі з кутом поширення $\theta_k = \pi / 2$ частіше всього досліджують експериментально і широко застосовують в приладах спін-хвильової НВЧ мікроелектроніки. тому останній вираз є досить важливим. Розглянемо ці хвилі більш детально. Відповідно до співвідношення (1.17) якщо змінювати величину хвильового числа k_y від 0 до ∞ то отримаємо такий частотний діапазон:

$$\omega_1 \le \omega \le \omega_H, \qquad (1.18)$$

який охоплює всю область частот $\Delta\omega_s$ існування поверхневих хвиль. Оскільки

$$\Delta\omega_s = \omega_B - \omega_1 = \frac{1}{4}\frac{\omega_M^2}{\omega_B + \omega_1},$$

то $\Delta\omega_s \to 0$ при $H_0 \to +\infty$. Тобто, при збільшенні величини зовнішнього магнітного поля H_0 , чи при збільшенні частоти, діапазон частоти існування ПМСХ звужується і прямує до нуля. Враховуючи, що випливає з рівняння (1.17), $d\omega/d\left|k_y\right| > 0$ ПМСХ є хвилями з прямою дисперсією. Групову швидкість цих хвиль можна визначити як

$$V_{гр} = \frac{d\omega}{dk_y} = \omega_k^2 = \frac{\omega_M^2}{\omega_k} s e^{-2|k_y|s} \operatorname{sgn} k_y \qquad (1.19)$$

та можна побачити що з ростом частоти вона зменшується. Якщо враховувати малі значення добутку $|k_y|s$ можемо отримати $e^{-2|k_y|s} \cong 1$ та при одному і тому ж значенні хвильового числа хвилі стають повільнішими в тонкіших шарах, тобто групова швидкість є прямо пропорційною до товщини шару. Якщо підставимо параметри ЗІГ у вираз для групової швидкості (1.19) то отримаємо на частоті 5 ГГц і при товщині феритового шару $s = 0,01$ см величину $V_{гр} \approx 8 \cdot 10^7$ см/сек, тобто $V_{гр}$ ця величина є більш ніж на два порядки менша за швидкість світла. Зменшуючи s, що є цілком реальним для феритових плівок, можемо ще зменшити $V_{гр}$. Це ілюструє рис.1.5 , на якому наведено дисперсію поверхневих МСХ в феритових плівках різної товщини при однаковій величині зовнішнього сталого магнітного поля $H_0 = 1,25$ кЕ.

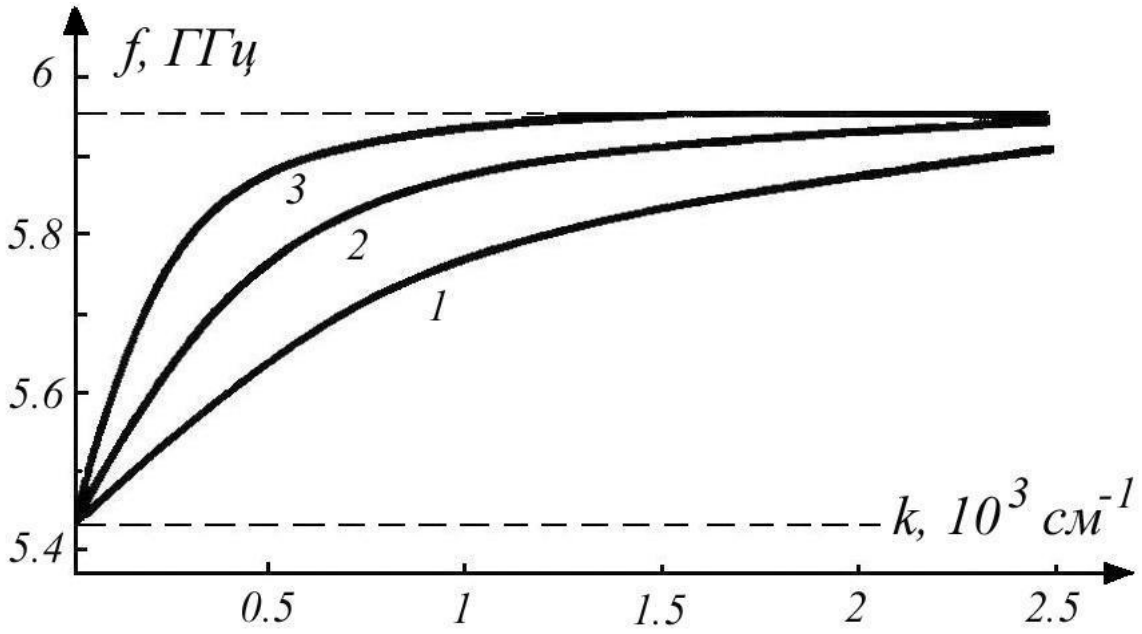

Рис.1.3 Дисперсія ПМСХ у феритових плівках різної товщини:

5, 10 та 20 мкм (відповідно, криві 1, 2, 3), $H_0 = 1,25$ кЕ.

1.4 Зворотні об'ємні МСХ

Ті розв'язки рівняння, які відповідають уявним величинам сталої поширення κ називають зворотними об'ємними МСХ (ЗОМСХ). Це працює за умови $\mu < 0$, тобто, в діапазоні частот

$$\omega_H < \omega < \omega_1 . \qquad (1.20)$$

Дисперсійне співвідношення для цих хвиль можемо дістати з виразу (1.13), покладаючи $\kappa = i\kappa_0$, де

$$\kappa_0 = \sqrt{-\left(\frac{k_z^2}{\mu} + k_y^2\right)} \qquad (1.21)$$

є дійсною величиною. Враховуючи, що $cth\ i\kappa_0 s = -i\ ctg\ \kappa_0 s$, дістаємо дисперсію ЗОМСХ:

$$2k_s \mu\kappa_0\ ctg\ \kappa_0 s - \mu^2\kappa_0^2 + k_s^2 - \mu_a^2 k_y^2 = 0 . \qquad (1.22)$$

Спектр об'ємних хвиль показано на (рис.1.4) Він лежить нижче за спектр поверхневих і займає область частот, яка визначається виразом (1.20); набір відповідних поверхонь на (рис.1.4) відповідає різним кореням (1.22). Із співвідношення (1.22) випливає, що дисперсія ЗОМСХ не змінюється при зміні напрямку поширення хвиль на протилежний.

Далі розглянемо ті ЗОМСХ, що здійснюють поширення вздовж вектора намагніченості $\acute{M_0}$ (тобто, $\theta_k = 0$). Покладаючи у виразі (1.21) $k_y = 0$ (та, відповідно, $\partial\Psi / \partial y = 0$), знаходимо:

$$2ctg\frac{|k_Z|s}{\sqrt{-\mu}} = \frac{1}{\sqrt{-\mu}} - \sqrt{-\mu} . \qquad (1.23)$$

Відповідна система рівнянь для опису електродинамічних граничних умов у випадку ЗОМСХ має вигляд:

$$\Psi_1\big|_{x=0}=\Psi_2\big|_{x=0}\ ,\ \Psi_2\big|_{x=s}=\Psi_3\big|_{x=s}\ ,$$

$$\frac{\partial\Psi_1}{\partial x}\Big|_{x=0}=\mu\frac{\partial\Psi_2}{\partial x}\Big|_{x=0}\ ,\ \mu\frac{\partial\Psi_2}{\partial x}\Big|_{x=s}=\frac{\partial\Psi_3}{\partial x}\Big|_{x=s}\ .$$

Використовуючи формулу котангенса подвійного аргументу, дисперсійне співвідношення (2.23) можемо переписати так:

$$\left(tg\frac{|k_z|s}{2\sqrt{-\mu}}-\sqrt{-\mu}\right)\left(tg\frac{|k_z|s}{2\sqrt{-\mu}}+\frac{1}{\sqrt{-\mu}}\right)=0\ ,$$

звідки

$$|k_{zn}|s=2\sqrt{-\mu}\left(n\pi+arctg\sqrt{-\mu}\right), \qquad (1.24)$$

де $n=0,1,2,..$ та

$$|k_{zm}|s=2\sqrt{-\mu}\left(m\pi-arctg\frac{1}{\sqrt{-\mu}}\right), \qquad (1.25)$$

де $m=1,2,3,..$

Тобто, бачимо що спектр ЗОМСХ утворюється з нескінченного числа гілок, що відповідні різним значенням n та m (симетричним та антисиметричним модам, відповідно). Кожна з цих гілок буде займати область частот (1.25), причому $k=0$ на тих частотах, що $\omega=\omega_1$, а $k\to\infty$ при $\omega\to\omega_H$ (рис.1.6).

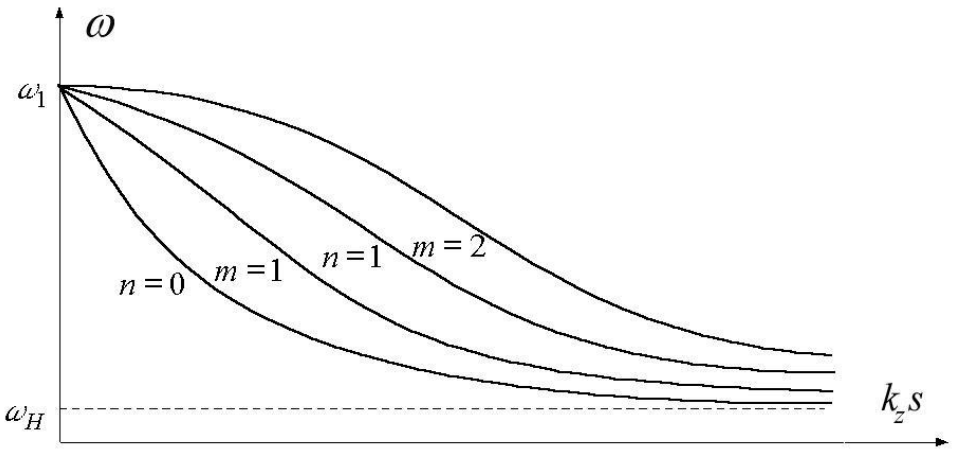

Рис.1.4. Дисперсія зворотних об’ємних МСХ, які поширюються вздовж напрямку вектора намагніченості $\acute{M_0}$ ($\theta_k = 0$)

Величина частотного діапазону, який займають ЗОМСХ, дорівнює:

$$\Delta\omega_V = \omega_1 - \omega_H = \frac{\omega_M}{\sqrt{1 + \frac{\omega_M}{\omega_H}} + 1}.$$

Звідси випливає, що при $\omega_H \to +\infty$ маємо $\Delta\omega_V \to \omega_M / 2$, тобто на відміну від поверхневих хвиль, діапазон існування ЗОМСХ при збільшенні ω_H (тобто, збільшенні величини зовнішнього магнітного поля) не звужується до нуля, а прямує до постійного значення, яке при $\omega_H = \infty$ складає $\Delta\omega = \omega_M / 2$.

2. ЗОМСХ у структурі МДФДМ

2.1 Дисперсійне співвідношення.

Структура, що є найбільш загальною для структур з одним шаром фериту є метал-діелектрик-ферит-діелектрик-метал (МДФДМ). З структури МДФДМ шляхом відповідних граничних переходів можна отримати будь-яку іншу структуру з одним шаром фериту.

При дослідженні екранованої структури з одним феритовим шаром можна найбільш наочно побачити зміну спектру різних типів МСХ через металеві підкладки.

Розглянемо дотичне намагнічування структури. Якщо покласти в дисперсійному співвідношенні для власних МСХ багатошарової структури

$$[\ell^{-\ell^{(2)}\ell} - \ell^{\ell^{(2)}\ell} - 2\ell\ell^{-2\ell\ell_0}\ell(\ell^{(2)}\ell)] \times$$

$$\times [\ell^{2\ell\ell-\ell^{(4)}\ell}_4 - \ell^{2\ell\ell+\ell^{(4)}\ell}_4 - 2\ell(\ell^{(2)}\ell)]\ell^{2\ell\ell}_3{}^2 -$$

$$- [\ell^{-2\ell\ell+\ell^{(2)}\ell}_2 - \ell^{-2\ell\ell-\ell^{(2)}\ell}_2 + 2\ell(\ell^{(2)}\ell)] \times$$

$$\times [\ell^{\ell^{(4)}\ell}_4 - \ell^{-\ell^{(4)}\ell}_4 + 2\ell\ell^{2\ell\ell}_4 2\ell(\ell^{(4)}\ell)] = 0$$

$s_3 = s_4 = 0$ і опустити індекс “2”, зможемо отримати дисперсійне співвідношення для МСХ, які поширюються в площині YZ:

$$\left(Ae^{-\kappa s_1} - Be^{\kappa s_1} - 2ABe^{-2ks_0} sh\kappa s_1\right)e^{2ks_2} - \left(Ae^{\kappa s_1} - Be^{-\kappa s_1} - 2e^{2ks_0} sh\kappa s_1\right)e^{-2ks_0} = 0 \quad .(2.1)$$

Звертаючись до виразу для сталої поширення κ

$$\kappa^{(2)} = k\sqrt{\frac{\mu_{22}^{(2)}\xi^2 + \mu_{33}^{(2)}}{\mu_{11}^{(2)}(1+\xi^2)}}$$

неважко помітити, що поверхневим і зворотним об'ємним типам МСХ відповідають нерівності (за відсутності анізотропії, тобто, при $\mu_{11} = \mu_{22} = \mu$)

$$\mu > 0 \ , \ \omega > \omega_1 \ , \tag{2.2}$$

$$-\xi^{-2} < \mu < 0 \ , \ \sqrt{\omega_H\left(\omega_H + \frac{\omega_M}{1+\xi^{-2}}\right)} < \omega < \omega_1 \ . \tag{2.3}$$

Розглянемо хвилі, що відносно напрямку постійного ефективного магнітного поля поширюються перпендикулярно (тобто, справедливо $\xi \to \infty, \kappa = k$). Для визначення граничної частоти спектра ПМСХ при $k \to 0$ в (2.1) необхідно розкласти в ряд експоненти до перших степенів нормованого хвильового числа ks_i, в результаті чого отримуємо:

$$\omega_{k\to 0} = \sqrt{(\omega_H + \omega_M)\left(\omega_H + \frac{\omega_M}{s_0/s_1 + s_2/s_1 + 1}\right)} \ . \tag{2.4}$$

Звідси випливає, що довгохвильова межа спектра залежить від розташування металевих екранів відносно феритового шару. У випадках ізольованого фериту (при $s_0 \, , s_2 \to \infty$), а також структури МДФ з одним шаром металу, вираз (2.4) збігається з вже отриманими раніше співвідношеннями. Наявність двох металевих шарів і їхнє наближення до фериту відповідно до виразу (2.4) призводить до збільшення нижньої межі спектра ПМСХ $\omega_{k\to 0}$ до величини $\omega_H + \omega_M$ в структурі МФМ без діелектричних прошарків. З іншого боку, такий вираз, при сталих поширення $\kappa = k \to \infty$ дисперсійного співвідношення (2.1), для короткохвильової межі спектра ПМСХ:

$$\omega_{k\to\infty}=\begin{cases}\omega_H+\omega_M/2\,,\ s_2\neq 0\,,s_0\geq 0\\ \omega_H+\omega_M,\quad s_2=0\,,\ s_0\geq 0\end{cases}. \qquad (2.5)$$

Величина $\omega_H+\omega_M,$ відповідає нижній межі спектра ПМСХ, а величина $\omega_H+\omega_M/2$ - верхній межі ізольованого феритового шару.

Дисперсійні співвідношення для хвиль що поширюються в протилежних напрямках у несиметричній структурі МДФДМ ($s_0\neq s_2$), отримані в результаті замін, є різними, що вказує на невзаємність ПМСХ. Тільки ті зворотні об'ємні МСХ, що поширюються вздовж вектора $H_i^{'}$ що є вектором внутрішнього постійного магнітного поля в дотично намагнічених структурах характеризуються взаємністю відносно напрямку поширення в несиметричних структурах, а такж ті прямі об'ємні МСХ дисперсійне співвідношення яких не залежить від компоненти тензора магнітної проникності μ_{12}, тобто ті, що знаходяться у нормально намагнічених структурах. Якщо здійснити в (2.1) заміну $A\to B^{-1}$, $B\to A^{-1}$ та перейти до границі при великих значеннях хвильового числа *k*, отримаємо

$$\omega_{k\to\infty}=\begin{cases}\omega_H+\omega_M/2\,,\ s_0\neq 0\,,\ s_2\geq 0\\ \omega_H+\omega_M\ ,\ s_0=0\,,\ s_2\geq 0\end{cases}. \qquad (2.6)$$

Великий вплив на характеристики хвилі має той металевий шар, що розташований ближче до поверхні фериту, там де спостерігається максимум амплітуди хвилі.

Перейдемо до розгляду зворотних об'ємних хвиль, які поширюються вздовж вектора напруженості постійного магнітного ефективного поля. Вважаючи $\xi=0$ і, враховуючи, що відповідно до (2.3) $\mu<0$, отримуємо

$$\text{[illegible]}=-\text{[illegible]}\sqrt{-\mu}\,\mu_{\pm}=\pm\text{[illegible]}\sqrt{-\mu}\,\text{[illegible]}=\text{[illegible]}^{*}=(-\mu-1+2\text{[illegible]}-\mu)/(1-\mu)$$

$$\left[2\sqrt{-\mu}-(\mu+1)\,tg\frac{ks_1}{\sqrt{-\mu}}\right]e^{k(s_0+s_2)}-\left[2\sqrt{-\mu}+(\mu+1)\,tg\frac{ks_1}{\sqrt{-\mu}}\right]e^{-k(s_0+s_2)}+$$

$$+\ 2(1-\mu)\frac{\square}{\sqrt{-\mu}}th[\square_0-\square_2)]=0 \qquad (2.8)$$

Оскільки вираз (2.8) не залежить від компоненти тензора магнітної проникності μ_{12}, а також є інваріантним щодо заміни $s_0 \to s_2$ і $s_2 \to s_0$, зворотні об'ємні МСХ, які поширюються уздовж внутрішнього магнітного поля в несиметричній структурі МДФДМ, є взаємними відносно напрямку їх поширення.

Дисперсійне співвідношення для ЗОМСХ (2.8) можна записати і в такому вигляді:

$$(thks_0+thks_2)\sqrt{-\mu}-(\mu+thks_0thks_2)tg\frac{ks_1}{\sqrt{-\mu}}=0 \qquad (2.9)$$

Таким чином, вираз (2.9) може бути використаним для комп'ютерного моделювання дисперсії ЗОМСХ в структурі з метал-діелектрик-ферит-діелектрик-метал в тривимірному просторі хвильових чисел та частот.

Розглянемо деякі частинні випадки. Поклавши в (2.9) $s_2 \to \infty$, переходимо до дисперсійного співвідношення для ЗОМСХ у структурі метал-діелектрик-ферит (МДФ):

$$tg\frac{ks_1}{\sqrt{-\mu}}=\frac{(1+\mu thks_0)\sqrt{-}}{\mu+thks_0} \qquad (2.10)$$

Цей вираз при $s_0 \to \infty$ перетворюється в дисперсійне співвідношення для ЗОМСХ в ізольованому феритовому шарі:

$$tg\frac{ks_1}{\sqrt{-\mu}}=\frac{2\sqrt{-\mu}}{1+\mu}, \qquad (2.11)$$

яке за допомогою формули котангенса подвійного аргументу розбивається на дисперсійні співвідношення для антисиметричних і симетричних мод. На рис.2.1 наведено дисперсійні залежності для основних мод ЗОМСХ при різних значеннях товщин діелектричних шарів структури МДФДМ. Порівнюючи дисперсійні залежності ЗОМСХ в структурі МДФДМ, можемо прийти до висновку, що хвилі у відсутності металевих екранів мають більшу групову і фазову швидкості, ніж за їх присутності.

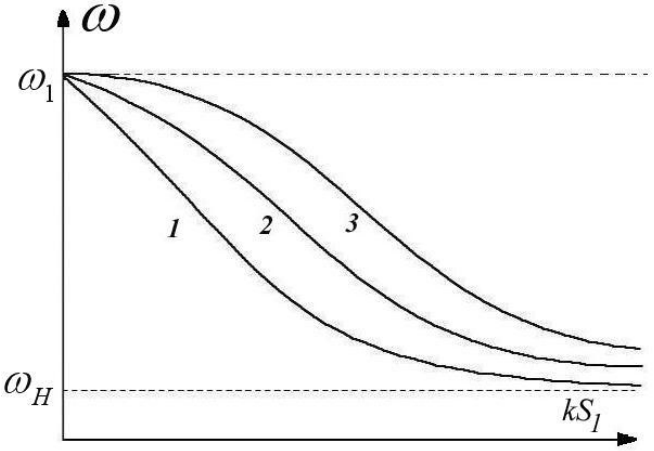

Рис.2.1. Дисперсія ЗОМСХ в структурі МДФДМ при різних співвідношеннях

товщин шарів s_0/s_1 та s_2/s_1: крива 1 - випадок $(\infty;\infty)$, крива 2 - $(0;\infty)$, крива 3 – $(0;0)$.

3.Комп’ютерне моделювання дисперсії ЗОМСХ у структурі МДФДМ

Метою роботи є розробка програми комп’ютерного моделювання дисперсії зворотних об’ємних МСХ в тривимірному просторі хвильових чисел та частот, що може забезпечити візуальний аналіз і допомогти в виборі багатошарової структури для конкретних застосувань.

Було створено програму в середовищі QtCreator в поєднанні з бібліотекою OpenGL для моделювання (текст програми у додатку 1). В інтерфейсі програми присутні такі поля взаємодії: область вводу та область відображення графічних результатів (Див. рис.3.1).

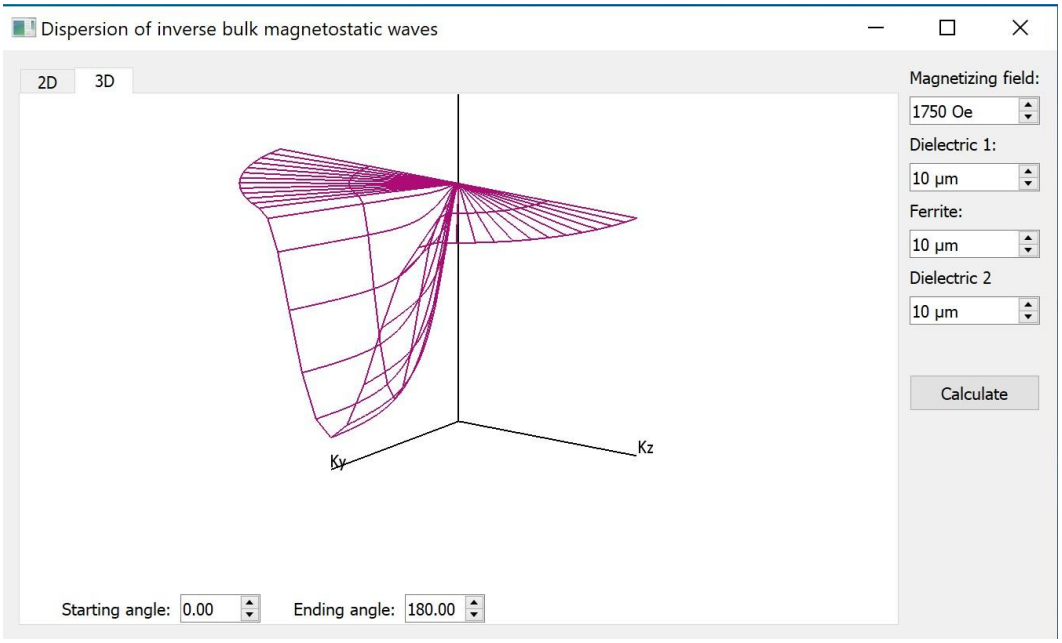

Рис.3.1 Інтерфейс програми для обрахунку дисперсії ЗОМСХ

Розглянемо область вводу. Область вводу розташована в правій частині інтерфейсу програми. В ній ми бачимо наступні поля вводу: «Magnetizing Field» – це величина зовнішнього магнітного поля, яку можна задавати в діапазоні від 0 до 4000 ерстед; «Dielectric 1», «Ferrite», «Dielectric 2» – задають товщини прошарків діелектрику та феритів у

мікрометрах; Щоб обрахувати значення рішень дисперсійного рівняння треба натиснути кнопку «Calculate», після цього результат обчислення відображується у вигляді тривимірного та двовимірного зображень у відповідних областях.

При підбиранні параметрів товщин слоїв діелектриків та фериту можна отриамти дисперсійні залежності для структури з одним шаром діалектрику, або для структури з двома шарами діалектрику

Область відображення графіків. В лівій верхній частині вікна програми є дві вкладки для відображення двовимірного та тривимірного представлення відповідно.

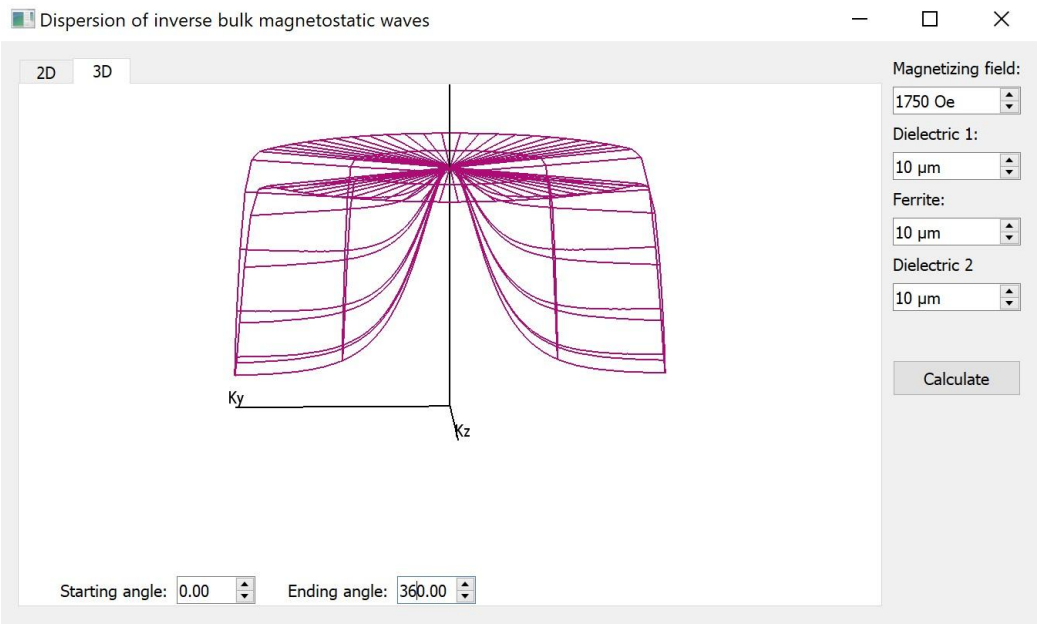

Рис.3.2 Приклад тривимірного зображення дисперсії ПМСХ, для кутів поширення 0º - 360º.

Область відображення тривимірного представлення. В цій області знаходиться площина (k_y, k_z); значення k_y та k_z, що обмежені площиною, підбираються в залежності від товщини феритового шару.

Також можемо побачити вісь, проведену перпендикулярно до центру площини, для відображення частот (ω).

В залежності від обраних параметрів виводиться тривимірне зображення дисперсії ЗОМСХ для обраного випадку. Параметри відображення можна задати в нижній частині вікна, під дисперсійною характеристикою. Поля для введення «Starting angle» та «Ending angle» дозволяють змінити відображення дисперсійної залежності, відповідно початковий та кінцевий кути поширення.

В вкладці «2D» зображена область відображення двовимірного представлення. В цій вкладці відображаються дисперсійні залежності, для заданих характеристик, в залежності від кута поширення θ_k. Кут поширення можна змінювати за допомогою слайдера у нижній частині області.

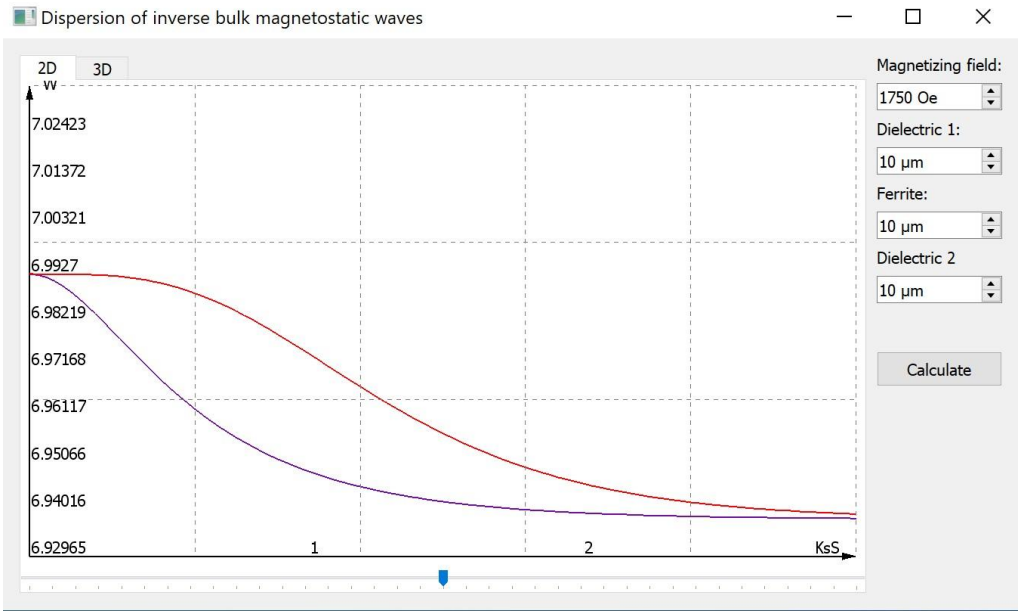

Рис.3.3 Приклад двовимірного зображення дисперсії ПМСХ, для кута поширення $\theta_k = \pi / 2$

ВИСНОВКИ

Досліджено дисперсійні залежності в ізольованих епітаксійних плівках ЗІГ. Було розглянуто поверхневі магнітостатичні хвилі в випадку ізольованого феритового шару та зворотні об’ємні магнітостатичні хвилі та одержано дисперсійне співвідношення використовуючи граничні умови неперервності тангенційних компонент напруженості магнітного поля. Розглянуто структуру метал-діелектрик-ферит-діелектрик-метал, з якої можна легко переходити до інших структур з одним феритовим шаром. Для цієї структури було отримано дисперсійне співвідношення для зворотних об’ємних магнітостатичних хвиль. Побудовано програму моделювання дисперсійних залежностей структури МДФДМ від параметрів, які можна змінювати в інтерфейсі програми. Програму можна використовувати для полегшення вибору структури при створенні приладу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

[1] Wolfram T. Magnetostatic surface waves in layered magnetic structures // J. Appl. Phys. – 1970. – V. 41. – P. 4748-4749.

[2] Гуляев Ю. В., Зильберман П.Е. Спинволновая электроника. – М.: Знание, 1988. – 54 с.

[3] Damon R.W., Eschbach J.R. Magnetostatic modes of ferromagnetic slab // J. Phys. and Chem. Solids. – 1961. – V. 19. – No. 3-4. – P. 308-320.

[4] Данилов В.В., Зависляк І.В., Нечипорук О.Ю. Спін-хвильова електродинаміка. - Киев: ВПЦ «Київський університет», 2008. - 351 с.

[5] Гуревич А. Г., Мелков Г. А. Магнитостатические колебания и волны. –М.: Физматлит, 1994. – 464 с.

[6] Pfeifer H. Characteristic of magnetostatic surface waves for a system of two magnetic films // Phys. Stat. Sol. (a). – 1973. – V. 18 – №1 – P. K53-K56.

[7] Pfeifer H. Magnetostatic modes in a combination of two magnetic films // Phys. Stat. Sol. (a). – 1973. – V. 19. – №1 – P. K85-K87.

ДОДАТКИ

Додаток 1.

Текст програми

main.cpp:

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char **argv)
{
   QApplication app(argc, argv);


   MainWindow mainwindow;
   mainwindow.show();
   return app.exec();
}
```

mainwindow.h:

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QWidget>
#include <QSpinBox>

#include "glwidget.h"
#include "glwidget3d.h"

class MainWindow : public QWidget
{
   Q_OBJECT
public:
   explicit MainWindow(QWidget *parent = 0);

private:
   QSpinBox * s1;
   QSpinBox * s2;
   QSpinBox * s3;
   QSpinBox * H0;
   QWidget* Create2dWidget();
   QWidget* Create3dWidget();

private slots:
   void onCalculateClicked();
   void checkParameters();
};

#endif // MAINWINDOW_H
```

mainwindow.cpp:

```
#include "mainwindow.h"
#include "QtWidgets/qboxlayout.h"
#include "QtWidgets/qdialog.h"
#include "QtWidgets/qlabel.h"
#include <QSlider>
#include <qpushbutton.h>
#include <qmessagebox.h>
#include <qcheckbox.h>
#include <qtabwidget.h>

MainWindow::MainWindow(QWidget *parent) :
    QWidget(parent)
{
    setWindowTitle("Dispersion of inverse bulk magnetostatic waves");
    QWidget * mainPanel = new QWidget;
    QVBoxLayout * mainPanelLayout = new QVBoxLayout;
    mainPanelLayout->setContentsMargins(0,0,0,0);
    QLabel * lblS1 = new QLabel("Dielectric 1:");
    QLabel * lblS2 = new QLabel("Ferrite:");
    QLabel * lblS3 = new QLabel("Dielectric 2");
    QLabel * lblH0 = new QLabel("Magnetizing field:");
    s1 = new QSpinBox;
    s1->setValue(10);
    s1->setSuffix(" µm");
    s1->setRange(0,100);
    s2 = new QSpinBox;
    s2->setValue(10);
    s2->setSuffix(" µm");
    s2->setRange(0,100);
    s3 = new QSpinBox;
    s3->setValue(10);
    s3->setSuffix(" µm");
    s3->setRange(0,100);
    H0 = new QSpinBox;
    H0->setRange(0,4000);
    H0->setSingleStep(50);
    H0->setValue(1750);
    H0->setSuffix(" Oe");
    lblS1->setBuddy(s1);
    lblS1->setBuddy(s2);
    lblS1->setBuddy(s3);
    lblH0->setBuddy(H0);

    mainPanelLayout->addWidget(lblH0);
    mainPanelLayout->addWidget(H0);
    mainPanelLayout->addWidget(lblS1);
    mainPanelLayout->addWidget(s1);
    mainPanelLayout->addWidget(lblS2);
    mainPanelLayout->addWidget(s2);
    mainPanelLayout->addWidget(lblS3);
    mainPanelLayout->addWidget(s3);

    mainPanelLayout->addSpacing(20);
    mainPanelLayout->addWidget(new QFrame);
    mainPanelLayout->addSpacing(20);

    QPushButton * btnCalculate = new QPushButton("Calculate");
    mainPanelLayout->addWidget(btnCalculate);
```

```
    mainPanelLayout->addStretch();

    mainPanel->setLayout(mainPanelLayout);

    QTabWidget * tabwidget = new QTabWidget;
    tabwidget->addTab(Create2dWidget(),"2D");
    tabwidget->addTab(Create3dWidget(),"3D");

    QHBoxLayout * mainLayout = new QHBoxLayout();
    mainLayout->addWidget(tabwidget);
    mainLayout->addWidget(mainPanel);

    setLayout(mainLayout);

                        connect(btnCalculate,       &QPushButton::clicked,       this,
&MainWindow::onCalculateClicked);
       connect(s1,  static_cast<void  (QSpinBox::*)(int)>(&QSpinBox::valueChanged),
this, &MainWindow::checkParameters);
       connect(s2,  static_cast<void  (QSpinBox::*)(int)>(&QSpinBox::valueChanged),
this, &MainWindow::checkParameters);
       connect(s3,  static_cast<void  (QSpinBox::*)(int)>(&QSpinBox::valueChanged),
this, &MainWindow::checkParameters);
}

void MainWindow::onCalculateClicked()
{
    double S1 = 1E-6 * s1->value();
    double S2 = 1E-6 * (s1->value() + s2->value());
    double S3 = 1E-6 * (s1->value()+ s2->value() + s3->value());
    Model::instance()->Calculate(S1, S2, S3, H0->value());
}

void MainWindow::checkParameters()
{
    if (s1->value() == 0 && s3->value() == 0)
    {
          QMessageBox::information(this, "Zero width", "Both Ferrites can't be zero
width");
        s1->setValue(10);
    }
}

QWidget* MainWindow::Create2dWidget()
{
    QWidget * widget2d = new QWidget;
    QVBoxLayout * widget2dLayout = new QVBoxLayout();
    widget2dLayout -> setContentsMargins(0,0,0,0);
    QSlider * slider = new QSlider(Qt::Horizontal);
    slider->setRange(0,SENSETIVITY_THETA);
    slider->setValue(SENSETIVITY_THETA / 2);
    slider->setSingleStep(1);
    slider->setPageStep(1);
    slider->setTickPosition(QSlider::TicksBelow);

    GLWidget * widget = new GLWidget(0);
    widget->resize(640, 480);

    widget2dLayout->addWidget(widget);
    widget2dLayout->addWidget(slider);
```

```
    widget2d->setLayout(widget2dLayout);

    connect(slider, &QSlider::valueChanged, widget, &GLWidget::onAngleSwitch);
    connect(Model::instance(), &Model::calculated, widget, &GLWidget::reloadData);
    widget->onAngleSwitch(SENSETIVITY_THETA / 2);

    return widget2d;
}

QWidget* MainWindow::Create3dWidget()
{
    QWidget * widget3d = new QWidget;

    QVBoxLayout * widget3dLayout = new QVBoxLayout;
    widget3dLayout->setContentsMargins(0,0,0,0);

    QWidget * widget3dPanel = new QWidget;
    widget3dPanel->setSizePolicy(QSizePolicy::Minimum, QSizePolicy::Minimum);
    QHBoxLayout * widget3dPanelLayout = new QHBoxLayout;
    widget3dPanelLayout->setContentsMargins(3,0,0,0);
    QCheckBox * cbUpperMode = new QCheckBox;
    cbUpperMode->setText("Second mode");
    cbUpperMode->setChecked(true);
    //widget3dPanelLayout->addWidget(cbUpperMode);
    QCheckBox * cbDownMode = new QCheckBox;
    cbDownMode->setText("First mode");
    cbDownMode->setChecked(false);
    //widget3dPanelLayout->addWidget(cbDownMode);

    widget3dPanelLayout->addSpacing(50);

    QLabel * lblStartAngle = new QLabel("Starting angle:");
    QLabel * lblEndAngle = new QLabel("Ending angle:");

    QDoubleSpinBox * sbStartAngle = new QDoubleSpinBox;
    sbStartAngle->setSingleStep(180 / SENSETIVITY_THETA);
    sbStartAngle->setMaximum(360);
    sbStartAngle->setValue(0);
    widget3dPanelLayout->addWidget(lblStartAngle);
    widget3dPanelLayout->addWidget(sbStartAngle);

    widget3dPanelLayout->addSpacing(30);

    QDoubleSpinBox * sbEndAngle = new QDoubleSpinBox;
    sbEndAngle->setSingleStep(180 / SENSETIVITY_THETA);
    sbEndAngle->setMaximum(360); sbEndAngle-
    >setValue(180); widget3dPanelLayout-
    >addWidget(lblEndAngle);
    widget3dPanelLayout->addWidget(sbEndAngle);

    widget3dPanelLayout->addStretch();

    widget3dPanel->setLayout(widget3dPanelLayout);

    GlWidget3d * widget3 = new GlWidget3d(0);
    widget3->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Expanding);


    widget3dLayout->addWidget(widget3);
```

```
    widget3dLayout->addWidget(widget3dPanel);
    widget3d->setLayout(widget3dLayout);


        connect   (cbUpperMode,     &QCheckBox::stateChanged,               widget3,
&GlWidget3d::onUpperModeStateChanged);
        connect   (cbDownMode,       &QCheckBox::stateChanged,              widget3,
&GlWidget3d::onDownModeStateChanged);
                              connect         (sbStartAngle,         static_cast<void
(QDoubleSpinBox::*)(double)>(&QDoubleSpinBox::valueChanged),                widget3,
&GlWidget3d::onStartAngleChanged);
                       connect       (sbEndAngle,                    static_cast<void
(QDoubleSpinBox::*)(double)>(&QDoubleSpinBox::valueChanged),                widget3,
&GlWidget3d::onEndAngleChanged);
                connect     (Model::instance(),     &Model::calculated,     widget3,
&GlWidget3d::reloadData);

    return widget3d;
}
```

model.h:

```
#ifndef MODEL_H
#define MODEL_H

#include <QObject>
#include <QVector>

#define S (5*10E-6)
#define PI 3.14159265358979323846
#define Vt (3.75*10E3)
#define Vl (7*10E3)
#define MAXPIXEL_X 6.0
#define MAXPIXEL_Y 10.0
#define SENSETIVITY_KS 200
#define SENSETIVITY_W 500
#define SENSETIVITY_THETA 36

class Model : public QObject
{
    Q_OBJECT
public:
    static Model *instance();
     void Calculate(double S1 = 10E-6, double S2 = 20E-6, double S3 = 30E-6, double
H0 = 1750);

    QVector <QVector <double> > getData();
    QVector<double> getKs();
    QVector<double> getOmega();

signals:
    void calculated();

public slots:

private:
    explicit Model(QObject *parent = 0);
    QVector <QVector <double> > data;
    static Model * m_model;
```

```
    QVector<double> Ks;
    QVector<double> omega;
};
#endif // MODEL_H
```

model.cpp:

```
#include "model.h"

Model* Model::m_model = nullptr;

Model::Model(QObject *parent) :
    QObject(parent)
{
    Calculate();
}

Model* Model::instance()
{
    if (m_model == nullptr)
        m_model = new Model();
    return m_model;
}

void Model::Calculate(double S1, double S2, double S3, double H0)
{
    //clearing last data
    data.clear();
    Ks.clear();
    omega.clear();

//    double S1 = 10E-6;                        double S2 = 20E-6;
    double M0_1 = 1750;                       double M0_2 = 1750;
    double Wm_1 = 2.8E6 * M0_1;
    double Wm_2 = 2.8E6 * M0_2;
    double Wh = 2.8E6 * H0;                 double Wh2 = Wh * Wh;
    double W1_12 = Wh * (Wh + Wm_1);          double W1_1 = sqrt(W1_12);
    double W1_22 = Wh * (Wh + Wm_2);          double W1_2 = sqrt(W1_22);
    double Wm = qMin(Wm_1, Wm_2);             double W1 = qMax(W1_1, W1_2);

    //setting grid values
    for (int i = 0; i <= MAXPIXEL_X; i++)
        Ks.append(i / S1 / 2);
    for (int i = 0; i <= MAXPIXEL_Y; i++)
        omega.append(i * (Wh + Wm / 2 - W1) / (MAXPIXEL_Y * 4) + W1);

    double dks = MAXPIXEL_X / S1 / SENSETIVITY_KS;
    double dw = (Wh + Wm_1 / 2 - W1_1) / SENSETIVITY_W / 10;
    double dtheta = PI / SENSETIVITY_THETA;
    for (double theta = 0; theta < PI; theta += dtheta)
    {
        for (double Ks = 0; Ks <= MAXPIXEL_X / S1 / 2; Ks += dks)
        {
            double Ks2 = Ks * Ks;
            double func1 = 1;
            int count = 0;
            double answer[2] = {0};

            for (double w = W1; w <= Wh + Wm_1 / 2; w += dw)
            {
```

```
                    double M_12 = (w * w - W1_12) / (w * w - Wh2);    double M_1 =
sqrt(M_12);

double func = exp(Ks*(S1+S3))*(2*-sqrt(M_1) - (M_2+1)*tan((Ks*S2)/-sqrt(M_1))) -
exp(-Ks*(S1+S3))*(2*-sqrt(M_1)+(M_1+1)*tan((Ks*S2)/-sqrt(M_1)))+
                +2*(1-M_1)*tan((Ks*S2)/-sqrt(M_1))*cosh(Ks*(S1+S3));

                                if (func * func1 < 1)
                                {
                                     answer[count] = (-(w - W1) / (Wh + Wm / 2 - W1)
* MAXPIXEL_Y * 4) + MAXPIXEL_X;
                                    if (++count == 2)
                                    {
                                        w = Wh + Wm / 2;
                                    }
                                }
                                func1 = func;
                }

                            if (count == 1)
                            {
                                answer[1] = answer[0];
                                answer[0] = 0;
                            }

                            QVector<double> res;
                            res.append(Ks * S1 * 2);
                            res.append(theta);
                            if (count > 0)
                            {
                                res.append(answer[0]);
                                res.append(answer[1]);
                            }
                            if (count == 0)
                            {
                                if (!data.isEmpty() && (Ks != 0))
                                {
                                    res.append(data.last().at(2));
                                    res.append(data.last().at(3));
                                }
                                else
                                {
                                    res.append(0);
                                    res.append(0);
                                }
                            }
                            data.append(res);
                        }
                    }
                    emit calculated();
                }

                QVector<QVector<double> > Model::getData()
                {
                    return data;
                }


                QVector<double> Model::getKs()
```

```
{
    return Ks;
}

QVector<double> Model::getOmega()
{
    return omega;
}
```

glwidget.h:

```
#pragma once

#include <QOpenGLWidget>
#include "model.h"

class GLWidget : public QOpenGLWidget
{
public:
   GLWidget(QWidget *parent);

public slots:
   void onAngleSwitch(int angle);
   void reloadData();

protected:
   virtual void initializeGL();
   virtual void resizeGL(int w, int h);
   virtual void paintGL();
   virtual void timerEvent(QTimerEvent *);

   GLuint makeObject();
    void arrow(GLenum primitive, GLdouble x0, GLdouble y0, GLdouble x1, GLdouble
y1);
   void dashLine(GLdouble x1, GLdouble y1, GLdouble x2, GLdouble y2);

private:
   QVector<GLuint> lists;
   QVector<QColor> colorList;
   Model * mod;

   GLdouble xRot;
   GLdouble yRot;
   GLdouble zRot;
   GLdouble xTra;
   GLdouble yTra;
   GLdouble zTra;
   GLdouble xSca;
   GLdouble ySca;
   GLdouble zSca;

   int _currentAngle;

   void qglColor(QColor color);
   void qglClearColor(QColor clearColor);
    void renderText(double x, double y, double z, const QString &str, const QFont &
font = QFont());
```

```
    void transformPoint(GLdouble out[4], const GLdouble m[16], const GLdouble
in[4]);

   GLint project(GLdouble objx, GLdouble objy, GLdouble objz,
       const GLdouble model[16], const GLdouble proj[16],
       const GLint viewport[4],
       GLdouble * winx, GLdouble * winy, GLdouble * winz);
};
```

glwidget.cpp:

```
#include "glwidget.h"
#include <math.h>
#include <qdebug.h>
#include <qpainter.h>

#pragma comment( lib, "OpenGL32.lib" )

#ifndef GL_MULTISAMPLE
#define GL_MULTISAMPLE  0x809D
#endif

#define V_COLOR "blue"
#define GRID_COLOR "gray"
#define BACKGROUND_COLOR "white"
#define AXIS_COLOR "black"

#define SCALE 0.98f

GLWidget::GLWidget(QWidget *parent)
   : QOpenGLWidget(parent)
{
   startTimer(40);
   setWindowTitle("Nauch");
   xSca = SCALE / MAXPIXEL_X;
   ySca = SCALE / MAXPIXEL_Y;
   zSca = 1.0f;
   xTra = -0.5 * SCALE;
   yTra = -0.5 * SCALE;
   zTra = 0.0f;
   xRot = 180.0f;
   yRot = 0.0f;
   zRot = 0.0f;
   setMinimumSize(1000,600);

   _currentAngle = 0;
}

void GLWidget::initializeGL()
{
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glOrtho(-.5, .5, .5, -.5, -1000, 1000);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
```

```
    colorList.append(QColor("red"));
    colorList.append(QColor("#760CAD"));
    colorList.append(QColor("orange"));
    colorList.append(QColor("#398DFF"));
    colorList.append(QColor("red"));
    colorList.append(QColor("#760CAD"));
    colorList.append(QColor("orange"));
    colorList.append(QColor("#398DFF"));

    GLuint list = makeObject();
    for (int i = 0; glIsList(list+i); i++)
    {
        lists.append(list+i);
    }
}

void GLWidget::resizeGL(int w, int h)
{
    glViewport(0, 0, (GLint)w, (GLint)h);
}

void GLWidget::paintGL()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    qglClearColor(QColor(BACKGROUND_COLOR));
    glLineWidth(2);

    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();
    glDisable(GL_MULTISAMPLE);
    glRotatef(xRot, 1.0f, 0.0f, 0.0f);
    glRotatef(yRot, 0.0f, 1.0f, 0.0f);
    glRotatef(zRot, 0.0f, 0.0f, 1.0f);
    glTranslatef(xTra, yTra, zTra);
    glScalef(xSca, ySca, zSca);

    //grid glLineWidth(1);
    qglColor(QColor(GRID_COLOR));
    for (double d=2*MAXPIXEL_X/10; d<=MAXPIXEL_X; d+=2*MAXPIXEL_X/10)
    {
        dashLine(d,0,d,MAXPIXEL_Y);
    }
    for (double d=MAXPIXEL_Y/3; d<=MAXPIXEL_Y; d+=MAXPIXEL_Y/3)
    {
        dashLine(0,d,MAXPIXEL_X,d);
    }
    //graphic

    glLineWidth(2);
    for (int i = 0; i < lists.size(); i++)
    {
        if (i % SENSETIVITY_THETA == _currentAngle)
        {
            qglColor(colorList.at(i/SENSETIVITY_THETA));
            glCallList(lists.at(i));
        }
```

```
    }

    //axis glLineWidth(2);
    qglColor(QColor(AXIS_COLOR));
    glBegin(GL_LINES);
        //x axis glVertex2d(0,0);
        glVertex2d(MAXPIXEL_X,0);
        //y axis glVertex2d(0,0);
        glVertex2d(0,MAXPIXEL_Y);
    glEnd();
    arrow(GL_TRIANGLES,0,0,0,MAXPIXEL_Y);
    arrow(GL_TRIANGLES,0,0,MAXPIXEL_X,0);

    //text
    glDisable(GL_DEPTH_TEST);
    renderText(0.1, MAXPIXEL_Y - 0.1, 0, QString("W"), QFont("arial", 14));
    renderText(MAXPIXEL_X - 0.3, 0.05, 0, QString("KsS"), QFont("arial", 14));
    QVector<double> ks = Model::instance()->getKs();
    for (int i = 2; i < ks.size() - 2; i += 2)
                renderText(i,  0.05,  0,  QString("%1").arg(ks.at(i)  /  100000,  2),
QFont("arial", 10));

    QVector<double> omega = Model::instance()->getOmega();
    for (int i = 0; i < omega.size() - 1; i++)
           renderText(0.02,  i  +  0.05,  0,  QString("%1").arg(omega.at(i)  /  1E9,  4),
QFont("arial", 10));
    glEnable(GL_DEPTH_TEST);

    glPopMatrix();

}

void GLWidget::timerEvent(QTimerEvent *)
{
    update();
}

GLuint GLWidget::makeObject()
{
    qDebug()<<"makeObject2d";
    makeCurrent();
    const int NUMBER_OF_MODES = 2;

    GLuint list;
    list = glGenLists(NUMBER_OF_MODES * SENSETIVITY_THETA);

    QVector <QVector<double> > data = Model::instance()->getData();
    for (int i = 0; i < NUMBER_OF_MODES; i++)
    {
        auto dataItem = data.begin();
        int listNumber = 0;
        while(dataItem!=data.end())
        {
            double lastTheta = dataItem->at(1);
            glNewList(list+listNumber+i * SENSETIVITY_THETA, GL_COMPILE);
            {
```

```
                glBegin(GL_LINE_STRIP);
                  for (; (dataItem!= data.end()) && (lastTheta == dataItem->at(1));
dataItem++)
                {
                    glVertex2d(dataItem->at(0),dataItem->at(2+i));
                }
                glEnd();
            }
            glEndList();
            listNumber++;
        }
    }
    return list;
}

void GLWidget::arrow(GLenum primitive, GLdouble x0, GLdouble y0, GLdouble x1,
GLdouble y1)
{
    GLdouble beta = atan2(y0 - y1, x1 - x0);
    GLdouble alfa = PI / 12;
    GLdouble r1 = MAXPIXEL_X / width() * 20;
    GLdouble r2 = MAXPIXEL_Y / height() * 20;
    GLdouble x2 = x1 - r1 * cos(beta + alfa);
    GLdouble y2 = y1 + r2 * sin(beta + alfa);
    GLdouble x3 = x1 - r1 * cos(beta - alfa);
    GLdouble y3 = y1 + r2 * sin(beta - alfa);
    glBegin(primitive);
    {
        glVertex2d(x1,y1);
        glVertex2d(x2,y2);
        glVertex2d(x3,y3);
    }
    glEnd();
}

void GLWidget::dashLine(GLdouble x1, GLdouble y1, GLdouble x2, GLdouble y2)
{
    glPushAttrib(GL_ENABLE_BIT);

    glLineStipple(6, 0xAAAA);
    glEnable(GL_LINE_STIPPLE);
    glBegin(GL_LINES);
        glVertex2d(x1,y1);
        glVertex2d(x2,y2);
    glEnd();

    glPopAttrib();
}

void GLWidget::onAngleSwitch(int angle)
{
    _currentAngle = angle;
}

void GLWidget::reloadData()
{
    glDeleteLists(lists.at(0), lists.size());
    lists.clear();
    GLuint list = makeObject();
```

```
    for (int i = 0; glIsList(list+i); i++)
    {
        lists.append(list+i);
    }
}

void GLWidget::qglColor(QColor color)
{
    glColor4f(color.redF(), color.greenF(), color.blueF(), color.alphaF());
}

void GLWidget::qglClearColor(QColor clearColor)
{
        glClearColor(clearColor.redF(),  clearColor.greenF(),  clearColor.blueF(),
clearColor.alphaF());
}

void GLWidget::renderText(double x, double y, double z, const QString &str, const
QFont & font)
{
    int width = this->width();
    int height = this->height();

    GLdouble model[4][4], proj[4][4];
    GLint view[4];
    glGetDoublev(GL_MODELVIEW_MATRIX, &model[0][0]);
    glGetDoublev(GL_PROJECTION_MATRIX, &proj[0][0]);
    glGetIntegerv(GL_VIEWPORT, &view[0]);
    GLdouble textPosX = 0, textPosY = 0, textPosZ = 0;

    project(x, y, z, &model[0][0], &proj[0][0], &view[0],
                &textPosX, &textPosY, &textPosZ);

    textPosY = height - textPosY;

    QPainter painter(this);
    painter.setPen(Qt::black);
    painter.setFont(QFont("Helvetica", 8));
    painter.setRenderHints(QPainter::Antialiasing | QPainter::TextAntialiasing);
    painter.drawText(textPosX, textPosY, str);
    painter.end();
}

inline GLint GLWidget::project(GLdouble objx, GLdouble objy, GLdouble objz,
    const GLdouble model[16], const GLdouble proj[16],
    const GLint viewport[4],
    GLdouble * winx, GLdouble * winy, GLdouble * winz)
{
    GLdouble in[4], out[4];

    in[0] = objx;
    in[1] = objy;
    in[2] = objz;
    in[3] = 1.0;
    transformPoint(out, model, in);
    transformPoint(in, proj, out);

    if (in[3] == 0.0)
        return GL_FALSE;
```

```
    in[0] /= in[3];
    in[1] /= in[3];
    in[2] /= in[3];

    *winx = viewport[0] + (1 + in[0]) * viewport[2] / 2;
    *winy = viewport[1] + (1 + in[1]) * viewport[3] / 2;

    *winz = (1 + in[2]) / 2;
    return GL_TRUE;
}

inline void GLWidget::transformPoint(GLdouble out[4], const GLdouble m[16], const
GLdouble in[4])
{
#define M(row,col)  m[col*4+row]
    out[0] =
        M(0, 0) * in[0] + M(0, 1) * in[1] + M(0, 2) * in[2] + M(0, 3) * in[3];
    out[1] =
        M(1, 0) * in[0] + M(1, 1) * in[1] + M(1, 2) * in[2] + M(1, 3) * in[3];
    out[2] =
        M(2, 0) * in[0] + M(2, 1) * in[1] + M(2, 2) * in[2] + M(2, 3) * in[3];
    out[3] =
        M(3, 0) * in[0] + M(3, 1) * in[1] + M(3, 2) * in[2] + M(3, 3) * in[3];
#undef M
}
```

glwidget3d.h:

```
#pragma once

#include <QOpenGLWidget>
#include "model.h"

class GlWidget3d : public QOpenGLWidget
{
public:
    GlWidget3d(QWidget *parent);

public slots:
    void onStartAngleChanged(double);
    void onEndAngleChanged(double);
    void onUpperModeStateChanged(int state);
    void onDownModeStateChanged(int state);
    void reloadData();

protected:
    virtual void initializeGL() Q_DECL_OVERRIDE;
    virtual void resizeGL(int w, int h) Q_DECL_OVERRIDE;
    virtual void paintGL() Q_DECL_OVERRIDE;
    virtual void timerEvent(QTimerEvent *) Q_DECL_OVERRIDE;
    virtual void mousePressEvent(QMouseEvent * event) Q_DECL_OVERRIDE;
    virtual void mouseReleaseEvent(QMouseEvent * event) Q_DECL_OVERRIDE;
    virtual void mouseMoveEvent(QMouseEvent * event) Q_DECL_OVERRIDE;

    GLuint makeObject(double startingTheta, double endTheta);
     void arrow(GLenum primitive, GLdouble x0, GLdouble y0, GLdouble x1, GLdouble
y1);
    void dashLine(GLdouble x1, GLdouble y1, GLdouble x2, GLdouble y2);

private:
```

```
    QVector<GLuint> lists;
    QVector<QColor> colorList;
    Model * mod;
    QPoint _mouseLastPos;

    GLdouble xRot;
    GLdouble yRot;
    GLdouble zRot;
    GLdouble xTra;
    GLdouble yTra;
    GLdouble zTra;
    GLdouble xSca;
    GLdouble ySca;
    GLdouble zSca;

    bool _drawUpperMode;
    bool _drawDownMode;
    double _startAngle;
    double _endAngle;

    void generateLists();
    void qglColor(QColor color);
    void qglClearColor(QColor clearColor);
    void renderText(double x, double y, double z, const QString &str, const QFont &
font = QFont());
     void transformPoint(GLdouble out[4], const GLdouble m[16], const GLdouble
in[4]);

    GLint project(GLdouble objx, GLdouble objy, GLdouble objz,
        const GLdouble model[16], const GLdouble proj[16],
        const GLint viewport[4],
        GLdouble * winx, GLdouble * winy, GLdouble * winz);
};
```

glwidget3d.cpp:

```
#include "GlWidget3d.h"
#include <math.h>
#include <QMouseEvent>
#include <qpainter.h>

#ifndef GL_MULTISAMPLE
#define GL_MULTISAMPLE  0x809D
#endif

#define V_COLOR "blue"
#define GRID_COLOR "gray"
#define BACKGROUND_COLOR "white"
#define AXIS_COLOR "black"

GlWidget3d::GlWidget3d(QWidget *parent)
    : QOpenGLWidget(parent)
{
    startTimer(40);
    setWindowTitle("Nauch3d");
    xSca = 1;
```

```
    ySca = 1;
    zSca = 1;
    xTra = 0;
    yTra = 0;
    zTra = 0;
    xRot = 70.0f;
    yRot = 0.0f;
    zRot = 32.0f;

    _drawUpperMode = true;
    _drawDownMode = false;
    _startAngle = 0;
    _endAngle = PI;
}

void GlWidget3d::initializeGL()
{
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-12, 12, 4, -8, -1000, 1000);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f);

    colorList.append(QColor("red"));
    colorList.append(QColor("#760CAD"));
    colorList.append(QColor("red"));
    colorList.append(QColor("#760CAD"));
    colorList.append(QColor("orange"));
    colorList.append(QColor("#398DFF"));
    colorList.append(QColor("red"));
    colorList.append(QColor("#760CAD"));
    colorList.append(QColor("orange"));
    colorList.append(QColor("#398DFF"));

    generateLists();
}

void GlWidget3d::resizeGL(int w, int h)
{
    glViewport(0, 0, (GLint)w, (GLint)h);
}

void GlWidget3d::paintGL()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnable(GL_DEPTH_TEST);

    qglClearColor(QColor(BACKGROUND_COLOR));
    glLineWidth(2);

    glMatrixMode(GL_PROJECTION);
    glPushMatrix();
    glRotatef(xRot, 1.0f, 0.0f, 0.0f);
    glRotatef(yRot, 0.0f, 1.0f, 0.0f);
    glRotatef(zRot, 0.0f, 0.0f, 1.0f);
    glTranslatef(xTra, yTra, zTra);
    glScalef(xSca, ySca, zSca);
```

```
    glLineWidth(2);
    for (int i = 0; i < lists.size(); i++)
    {
              if (((i % 2 == 0) && (!_drawDownMode)) || ((i % 2 == 1) &&
(!_drawUpperMode)))
              continue;
        qglColor(colorList.at(i));
        glCallList(lists.at(i));
    }


    glLineWidth(2);
    qglColor(QColor(AXIS_COLOR));
    glBegin(GL_LINES);
        //x axis glVertex3d(0,0,0);
        glVertex3d(MAXPIXEL_X,0,0);
        //y axis glVertex3d(0,0,0);
        glVertex3d(0,MAXPIXEL_X,0);

        glVertex3d(0,0,0);
        glVertex3d(0,0,MAXPIXEL_Y);
    glEnd();

    //text
    glDisable(GL_DEPTH_TEST);
    renderText(0.1, 0.1, MAXPIXEL_Y + 0.1, QString("W"), QFont("arial", 14));
    renderText(MAXPIXEL_X + 0.1, 0.1, 0.1, QString("Kz"), QFont("arial", 14));
    renderText(0.1, 0.2 + MAXPIXEL_X, 0.1, QString("Ky"), QFont("arial", 14));
    glEnable(GL_DEPTH_TEST);

    glPopMatrix();
}

void GlWidget3d::timerEvent(QTimerEvent *)
{
    update();
}

GLuint GlWidget3d::makeObject(double startingTheta, double endTheta)
{
    makeCurrent();
    const int NUMBER_OF_MODES = 2;
    const double tolerance = 0.01;
    GLuint list;
    list = glGenLists(NUMBER_OF_MODES*2);

    QVector<QVector<double> > data = Model::instance()->getData();

    QVector<QVector<QVector<double>>> ksSortedData;
    ksSortedData.resize(ceil(MAXPIXEL_X + 1));
    for (int i = 0; i < NUMBER_OF_MODES; i++)
    {
        glNewList(list+i, GL_COMPILE);
        {
            for (int half = 0; half < 2; half++)
            {
                auto dataItem = data.begin();
```

```
                while(dataItem!=data.end())
                {
                    double lastTheta = dataItem->at(1) + PI * half;
                    glBegin(GL_LINE_STRIP);
                    for (; (dataItem!= data.end()) && (lastTheta == dataItem->at(1)
+ PI * half); dataItem++)
                    {
                                if ((dataItem->at(1) + PI * half - startingTheta <
-tolerance) || (dataItem->at(1) + PI * half - endTheta > tolerance))
                            continue;

                        if (i==0)
                        {
                                 if (abs(dataItem->at(0) - ceil(dataItem->at(0))) <
tolerance)
                            {
                                QVector<double> newData = *dataItem;
                                newData[1] += PI * half;

ksSortedData[ceil(dataItem->at(0))].append(newData);
                            }
                             else if (abs(dataItem->at(0) - floor(dataItem->at(0)))
< tolerance)
                            {
                                QVector<double> newData = *dataItem;
                                newData[1] += PI * half;

ksSortedData[floor(dataItem->at(0))].append(newData);
                            }
                        }
                           double x = dataItem->at(0) * cos(dataItem->at(1) + PI *
half);
                           double y = dataItem->at(0) * sin(dataItem->at(1) + PI *
half);
                        glVertex3d(x, y, dataItem->at(2+i));
                    }
                    glEnd();
                }
            }
        }
        glEndList();
    }
    for (int i = 0; i < NUMBER_OF_MODES; i++)
    {
        glNewList(list + NUMBER_OF_MODES + i, GL_COMPILE);
        {
            for (auto dataItem : ksSortedData)
            {
                static int count = 0;
                glBegin(GL_LINE_STRIP);
                for (auto innerDataItem : dataItem)
                {
                    double x = innerDataItem.at(0) * cos(innerDataItem.at(1));
                    double y = innerDataItem.at(0) * sin(innerDataItem.at(1));
                    glVertex3d(x, y, innerDataItem.at(2 + i));
                }
                glEnd();
                count++;
            }
```

```
        }
        glEndList();
    }
    return list;
}

void GlWidget3d::arrow(GLenum primitive, GLdouble x0, GLdouble y0, GLdouble x1,
GLdouble y1)
{
    GLdouble beta = atan2(y0 - y1, x1 - x0);
    GLdouble alfa = PI / 12;
    GLdouble r1 = MAXPIXEL_X / width() * 20;
    GLdouble r2 = MAXPIXEL_Y / height() * 20;
    GLdouble x2 = x1 - r1 * cos(beta + alfa);
    GLdouble y2 = y1 + r2 * sin(beta + alfa);
    GLdouble x3 = x1 - r1 * cos(beta - alfa);
    GLdouble y3 = y1 + r2 * sin(beta - alfa);
    glBegin(primitive);
    {
        glVertex2d(x1,y1);
        glVertex2d(x2,y2);
        glVertex2d(x3,y3);
    }
    glEnd();
}

void GlWidget3d::dashLine(GLdouble x1, GLdouble y1, GLdouble x2, GLdouble y2)
{
    glPushAttrib(GL_ENABLE_BIT);

    glLineStipple(6, 0xAAAA);
    glEnable(GL_LINE_STIPPLE);
    glBegin(GL_LINES);
        glVertex2d(x1,y1);
        glVertex2d(x2,y2);
    glEnd();

    glPopAttrib();
}
void GlWidget3d::mousePressEvent(QMouseEvent * event)
{
    _mouseLastPos = event->globalPos();
}

void GlWidget3d::mouseMoveEvent(QMouseEvent *event)
{
    zRot -= (event->globalPos().x() - _mouseLastPos.x())/5.0;
    xRot -= (event->globalPos().y() - _mouseLastPos.y())/5.0;
    _mouseLastPos = event->globalPos();
}

void GlWidget3d::onStartAngleChanged(double angle)
{
    _startAngle = PI * angle / 180;
    reloadData();
}

void GlWidget3d::onEndAngleChanged(double angle)
{
```

```
    _endAngle = PI * angle / 180;
    reloadData();
}

void GlWidget3d::mouseReleaseEvent(QMouseEvent * event)
{
    Q_UNUSED(event);
}

void GlWidget3d::onUpperModeStateChanged(int state)
{
    _drawUpperMode = state;
}

void GlWidget3d::onDownModeStateChanged(int state)
{
    _drawDownMode = state;
}

void GlWidget3d::generateLists()
{
    GLuint list = makeObject(_startAngle, _endAngle);
    for (int i = 0; glIsList(list+i); i++)
    {
        lists.append(list+i);
    }
}

void GlWidget3d::reloadData()
{
    glDeleteLists(lists.at(0), lists.size());
    lists.clear();
    generateLists();
}

void GlWidget3d::qglColor(QColor color)
{
    glColor4f( color.blueF(), color.greenF(),color.redF() , color.alphaF());
}

void GlWidget3d::qglClearColor(QColor clearColor)
{
        glClearColor(clearColor.redF(),  clearColor.greenF(),  clearColor.blueF(),
clearColor.alphaF());
}

void  GlWidget3d::renderText(double  x,  double  y,  double  z,  const  QString  &str,
const QFont & font)
{
    int width = this->width();
    int height = this->height();

    GLdouble model[4][4], proj[4][4];
    GLint view[4];
    glGetDoublev(GL_MODELVIEW_MATRIX,  &model[0][0]);
    glGetDoublev(GL_PROJECTION_MATRIX,  &proj[0][0]);
    glGetIntegerv(GL_VIEWPORT,  &view[0]);
    GLdouble textPosX = 0, textPosY = 0, textPosZ = 0;
```

```
    project(x, y, z, &model[0][0], &proj[0][0], &view[0],
                &textPosX, &textPosY, &textPosZ);

    textPosY = height - textPosY;

    QPainter painter(this);
    painter.setPen(Qt::black);
    painter.setFont(QFont("Helvetica", 8));
    painter.setRenderHints(QPainter::Antialiasing | QPainter::TextAntialiasing);
    painter.drawText(textPosX, textPosY, str);
    painter.end();
}

inline GLint GlWidget3d::project(GLdouble objx, GLdouble objy, GLdouble objz,
    const GLdouble model[16], const GLdouble proj[16],
    const GLint viewport[4],
    GLdouble * winx, GLdouble * winy, GLdouble * winz)
{
    GLdouble in[4], out[4];

    in[0] = objx;
    in[1] = objy;
    in[2] = objz;
    in[3] = 1.0;
    transformPoint(out, model, in);
    transformPoint(in, proj, out);

    if (in[3] == 0.0)
        return GL_FALSE;

    in[0] /= in[3];
    in[1] /= in[3];
    in[2] /= in[3];

    *winx = viewport[0] + (1 + in[0]) * viewport[2] / 2;
    *winy = viewport[1] + (1 + in[1]) * viewport[3] / 2;

    *winz = (1 + in[2]) / 2;
    return GL_TRUE;
}

inline void GlWidget3d::transformPoint(GLdouble out[4], const GLdouble m[16],
const GLdouble in[4])
{
#define M(row,col)  m[col*4+row]
    out[0] =
        M(0, 0) * in[0] + M(0, 1) * in[1] + M(0, 2) * in[2] + M(0, 3) * in[3];
    out[1] =
        M(1, 0) * in[0] + M(1, 1) * in[1] + M(1, 2) * in[2] + M(1, 3) * in[3];
    out[2] =
        M(2, 0) * in[0] + M(2, 1) * in[1] + M(2, 2) * in[2] + M(2, 3) * in[3];
    out[3] =
        M(3, 0) * in[0] + M(3, 1) * in[1] + M(3, 2) * in[2] + M(3, 3) * in[3];
#undef M
}
```