23 березня 2020 р.

Додаткові нотатки до лабораторної роботи № 10. Власні вектори і власні числа матриць з елементами-дійсними числами

1) Проблема неквадратного пікселя. Розглянемо одну типову помилку при виконанні ЛР №10, пов'язану з різними масштабами по осях, яка виникає при використанні на комп'ютері екранних режимів з неквадратним пікселем.

Нагадую, що ми малюємо насамперед одиничне коло, яке описує кінець одиничного вектора **x** з компонентами $x_1 = \cos\varphi$ та $x_2 = \sin\varphi$. Для цього кут φ має перебрати всі значення від 0 до 2π з малим кроком δ.

Наприклад, можна взяти $\delta = \pi/1800 = 0.00174533$ радіана, тобто 0.1º, тоді на коло вийде 3600 точок, і вони зіллються на екрані в суцільну лінію. Чи просто $\delta = 0.001$, тоді на коло вийде $2\pi*1000 = 6283$ точки. То вже не принципово.

Потім ми на малюнок нанесемо й інші лінії (про еліпс і хрест поверх нього – трохи пізніше), а зараз давайте відволікнемось від них і подивимось саме на коло.

Ось таким вийшов наш малюнок (рис.1). Масштабна сітка – клітинки нанесені через одну одиницю. Наче, красиво. Одиничне коло, яке і має бути, вписане в квадратик 2x2.

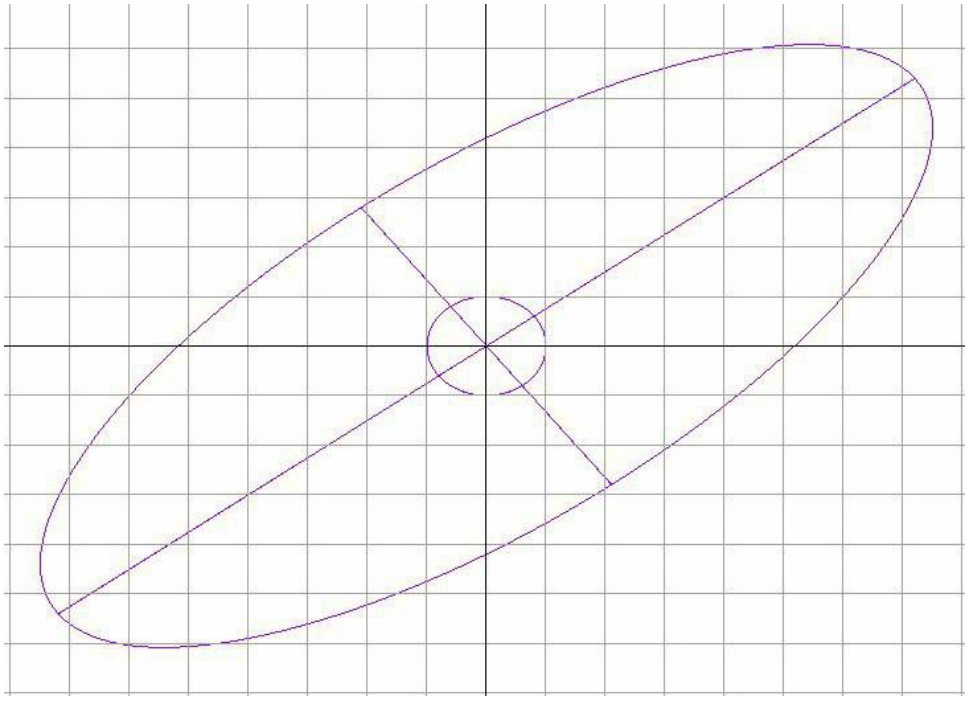

Рис. 1

Але ж це не квадратик! Це прямокутник!

І, відповідно, одиничне коло в середині малюнку – не коло, а еліпс.

Причина в тому, що на нашому рисунку по горизонтальній і вертикальній осях – різні масштаби. Як так, ви скажете, ми ж при малюванні призначали масштаби однаковими, по 40 пікселів (умовно кажучи) на одиницю довжини!? І якщо тюкнути на формат рисунку у Word-файлі, побачимо:

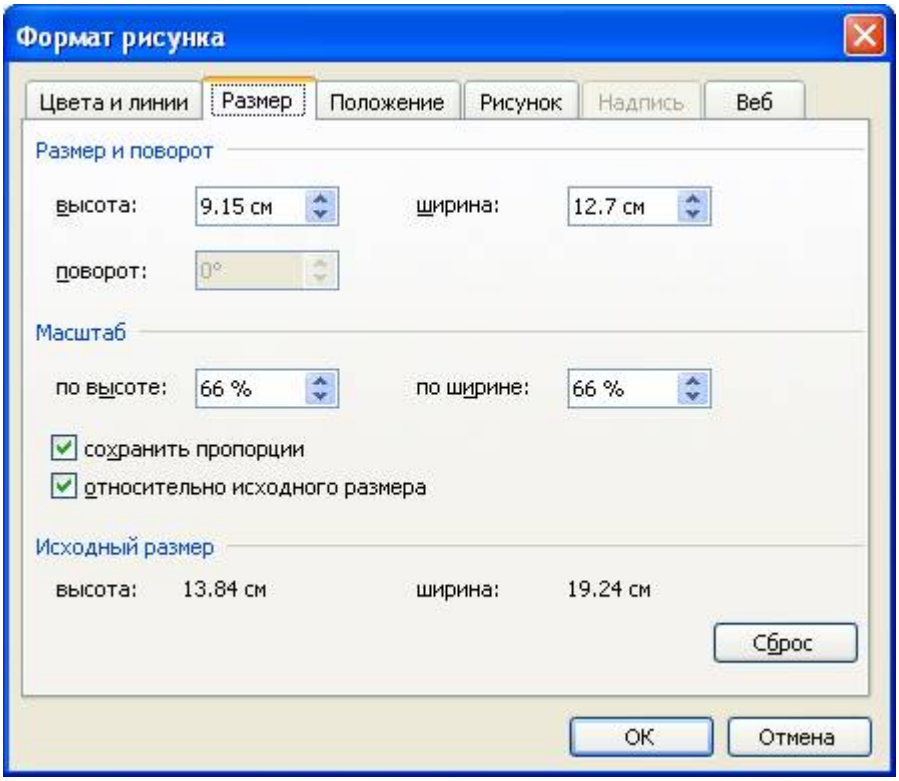

Рис. 2

тобто масштаби по висоті і ширині теж однакові, по 66%. В чому ж справа?

А в тому, що екранний піксель у данному випадку – не квадратний.

Що його робити, дорога редакціє?

Дивимось на малюнок, рахуємо клітинки.

По горизонталі шкала йде від –8.1 до +8.1, тобто всього ширина малюнка 16.2 одиниці. По вертикалі – від –7.1 до +6.9, тобто висота становить 14.0 одиниць.

Вибираємо масштаб, зважаючи на розміри друкованої сторінки: нехай одиниця сітки = 0.7 см, тоді малюнок повинен мати розмір по ширині 16.2 × 0.7 см = 11.34 см, а по висоті – 14.0 × 0.7 см = 9.8 см.

Після цього тюкаємо на формат рисунку і виправляємо його параметри:

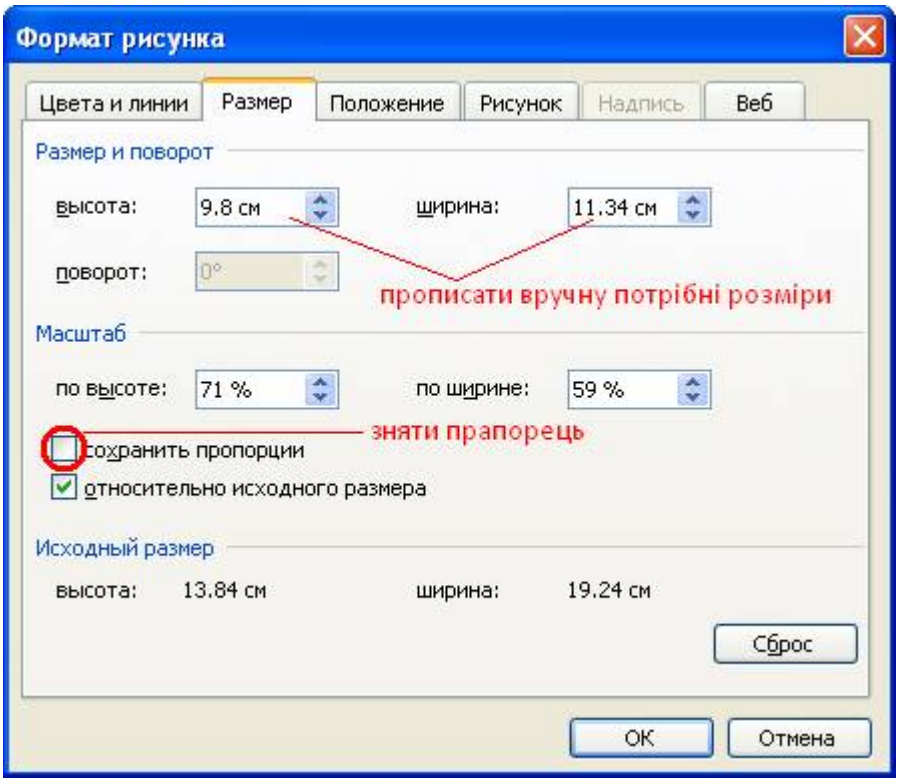

Рис. 3.

Зверніть увагу, раніше на малюнку масштаби по висоті і ширині були однакові – по 66%., а тепер, після того, як ми примусово прописали потрібні розміри в сантиметрах, вони стали різними – 71% та 59%.

Ось як тепер виглядає неспотворений малюнок:

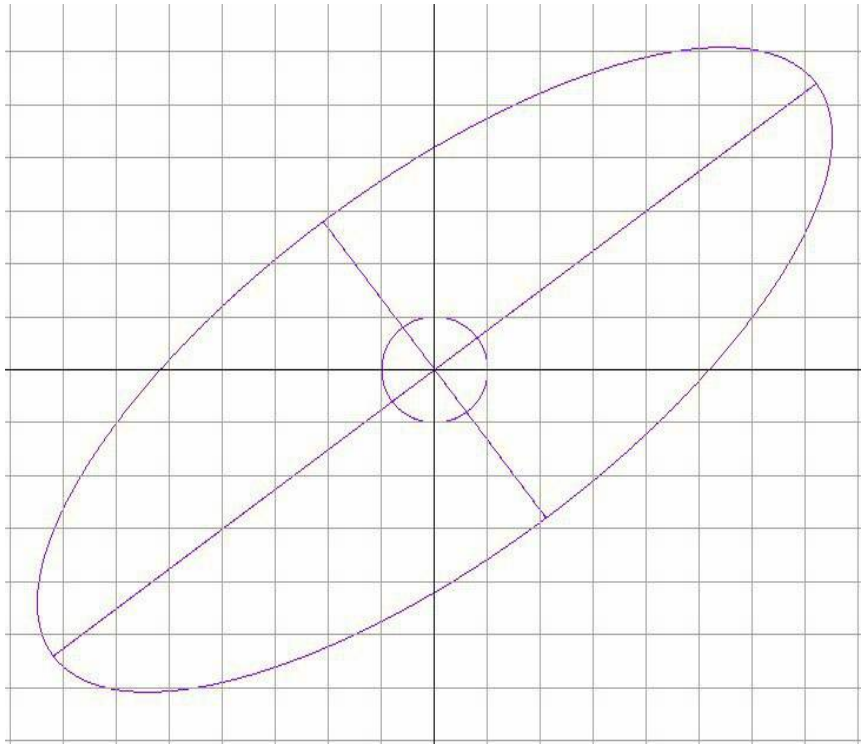

Рис. 4

2) Малюємо еліпси. Тепер, коли з масштабами все в порядку, подивимось, як будувати ці криві.

Отже, використовуємо квадратну матрицю

$$\mathbf{A} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

Вектор $\mathbf{y} = \mathbf{Ax}$ матиме компоненти $y_1 = ax_1 + bx_2$ та $y_2 = cx_1 + dx_2$.

Тож фрагмент нашої програми, що відповідає за малювання кривих, матиме приблизно такий вигляд (можливі різні відмінності в залежності від транслятора):

```
.....
pi=3.14159265;
delta = pi/1800;

for(phi = 0; phi <= 2*pi; phi += delta) {
   x1 = cos(phi);
   x2 = sin(phi);
   y1 = a*x1 + b*x2;
   y2 = c*x1 + d*x2;
   // нанести точку з координатами (x1, x2)
   .....;
   // нанести точку з координатами (y1, y2)
   .....;
}
```

Власне кажучи, це все. (Тільки не забудьте на початку задекларувати змінні і визначити компоненти матриці `a, b, c, d`)

Як бачите, ми викрутились, застосовуючи лише скалярні змінні, а масиви в програмі навіть не знадобилися.

Ледь не забув: наведений рисунок побудовано для матриці $\mathbf{A} = \begin{pmatrix} 4.5 & 6 \\ 6 & 1 \end{pmatrix}$.

Залишилось з’ясувати, як розрахувати і намалювати отой хрест поверх еліпса.

3) Визначаємо власні вектори і власні числа. Власний вектор, як ви пам'ятаєте, це такий, який після множення на матрицю **A** не змінює свого напрямку. Тобто, коли вектори **x** та **y** мають один напрямок, то **x** і буде таким власним вектором.

В нашому двовимірному випадку «напрямок» вектора – це кут, який він складає з горизонтальною віссю. Для вектора **x** цей кут дорівнює φ, а для вектора **y** його треба обчислити, перевевши його декартові координати у полярні:

$$(y_1, y_2) \rightarrow (y, \psi),$$

Якщо кути φ та ψ рівні, то вектор **x** власний, якщо ж ні – то й ні.

Точніше, **x** є власним вектором, якщо $\varphi = \psi$ або ж $\varphi = \psi \pm \pi$.

А власне число? Це коефіцієнт, який показує, у скільки разів змінюється довжина власного вектору, якщо його помножити на матрицю **A**. Оскільки довжина **x** дорівнює 1, то довжина **y** дорівнюватиме відповідному власному числу. (Якщо $\varphi = \psi \pm \pi$, то власному числу треба приписати знак мінус).

В нашу програму, отже, слід додати кілька операторів:

```
.....
pi=3.14159265;
delta = pi/1800;

for(phi = -pi; phi <= pi; phi += delta) {
   x1 = cos(phi);
   x2 = sin(phi);
   y1 = a*x1 + b*x2;
   y2 = c*x1 + d*x2;
   // нанести точку з координатами (x1, x2)
   .....;
   // нанести точку з координатами (y1, y2)
   .....;
   // перевести декартові координати (y1,y2) в полярні (y,psi)
   y = sqrt(x1*x1 + x2*x2);
   psi = atan2(y2,y1);
   // порівняти кути, тим самим перевірити, чи не є вектор власним
   if (phi ≈≈ psi || phi ≈≈ psi+pi || phi ≈≈ psi-pi){
      // і якщо так, відобразити його відрізком (0,0)-(y1,y2)
      .....;
      // та надрукувати кут власного вектора
      // і власне число, попередньо визначивши його знак:
      if (phi ≈≈ psi) {lambda = y;} else {lambda = -y;}
      cout << "кут власного вектора =" << phi*180/pi
           << " градусів; власне число =" << lambda << endl;
   }
}
```

Кілька пояснень.

По-перше, розглянемо функцію `atan2(y2,y1)`. Її призначення – переводити декартові координати в полярний кут і вона присутня в стандартних бібліотеках багатьох трансляторів. Коли кінець вектора (y_1, y_2) знаходиться в I квадранті, цей кут дорівнює просто $\psi=\text{arctg}(y_2/y_1)$ (див. рис.5).

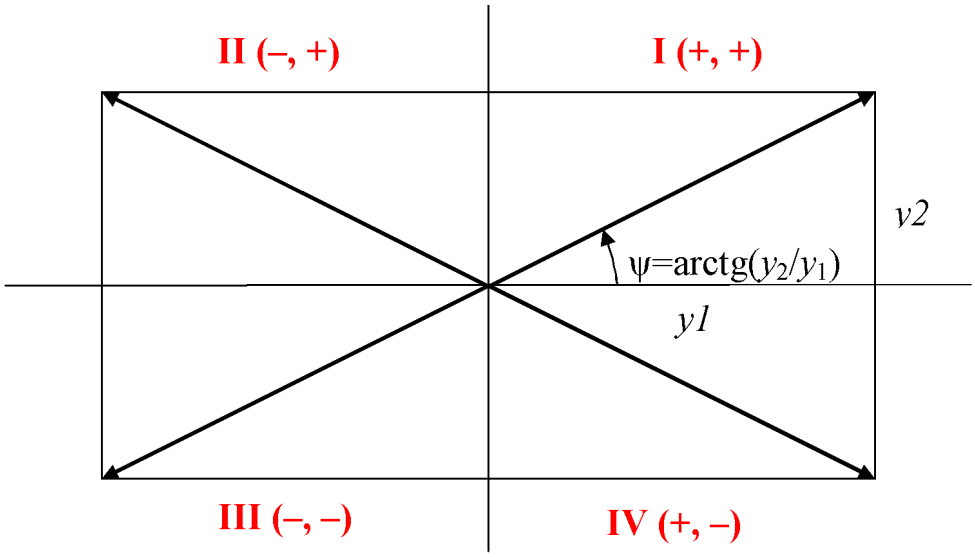

Рис. 5

Але якщо користуватись звичайною функцією арктангенаса `atan(y2/y1)`, то для полярного кута в інших квадрантах у формули потрібно додавати $\pm\pi$. Крім того, особливим чином повинна оброблятися ситуаця, коли $y_1 = 0$. Саме цей аналіз робить функція двох аргументів `atan2(y2,y1)`, спрощуючи життя програмістам. Будьте уважними, порядок її аргументів саме такий – **спочатку довжина протилеглого катету, потім – прилеглого!**

По друге, результат, який повертає функція `atan2`, лежить в межах від $-\pi$ до $+\pi$. Саме тому в головному циклі ми змінили параметри таким чином, щоб кут φ перебирав всі значення не від 0 до 2π, а від $-\pi$ до $+\pi$:

```
for(phi = -pi; phi <= pi; ...
```

По-третє, що це за дивна конструкція `if (phi ≈≈ psi || ...` ? В жодному трансляторі такого символа «приблизно дорівнює» немає.

Спочатку давайте розберемось, чому ми змушені вимагати саме приблизної, а не точної рівності кутів φ та ψ.

На рис. 6 показані кілька векторів **x** та **y**. Крок δ між послідовними положеннями **x** навмисно перебільшений, а кожне положення вектора **x** та відповідного йому вектора **y** відмічено своїм кольором (червоний – жовтий – зелений – синій – фіолетовий, як на веселці).

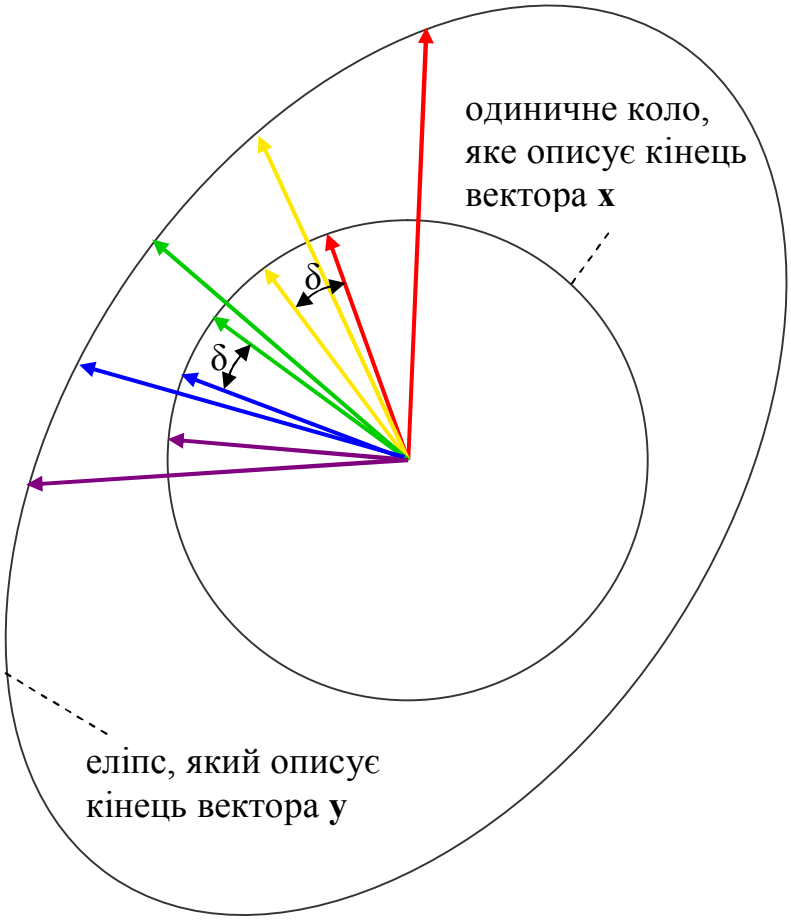

Рис. 6

Вектори **x** та **y** обертаються з різними швидкостями: на рисунку спочатку **y** відстає від **x**, а потім десь на проміжку між зеленим та синім починає його випереджати. Точка, де вони порівнюються, і є напрямком власного вектора. Але ж із-за дискретного кроку ми її промайнули! І яким би малим ми не брали наш крок δ по куту, повного збігу у всіх десяткових знаках φ та ψ нам не домогтися.

Тому й доводиться обмежуватись перевіркою наближеної рівності. Фактично, замість перевірки $\varphi = \psi$ або $\varphi = \psi \pm \pi$ треба робити перевірку:

$$|\varphi - \psi| < \varepsilon \quad \text{або} \quad ||\varphi - \psi| - \pi| < \varepsilon ,$$

де ε – мале число, за порядком величини близьке до кроку δ.

Тож той дивний оператор з символами ≈≈, насправді, треба записувати як

```
if (fabs(phi-psi)<eps || fabs(fabs(phi-psi)-pi)<eps) {
```

попередньо присвоївши величині ε значення, скажімо

```
eps = 3*delta
```

Поекспериментуйте з різними значеннями ε. Занадто велике ε призведе до того, що умова буде виконуватись при декількох суміжних значеннях φ, і намальовані вектори **y** заметуть цілий сектор. Замале ж значення ε може призвести до того, що ви проскочите напрямок власного вектора, не помітивши його.

Але замести невеличкий сектор – біда невелика. Ваша програма кілька разів надрукує

```
кут власного вектора = ..... градусів; власне число = .....
```

з близькими значеннями, а от промайтнути напрямок власного вектора – то гірше.

4) Випадок симетричних матриць. Для матриці $\mathbf{A} = \begin{pmatrix} 4.5 & 6 \\ 6 & 1 \end{pmatrix}$, що розглядалася як приклад, програма видає такі значення:

```
кут власного вектора = -53.1 градусів; власне число = -3.5
кут власного вектора = 36.9 градусів; власне число = 9
```

Зазначимо, що кут між двома власними векторами виявився рівним 36.9 – (–53.1) = 90 градусів.

Це є загальною властивістю симетричних матриць: їхні власні числа завжди дійсні, а власні вектори – взаємно-ортогональні. (У несиметричних матриць загального вигляду власні вектори можуть перетинатися під довільними кутами.)

Зверніть увагу також на таке. Той факт, що власні вектори досліджуваної матриці, дійсно, перетинаються під прямим кутом, добре видно з рис. 4, де виправлені масштаби по координатних осях. Цей рисунок міг би нас наштовхнути на думку дослідити, чи є ця властивість загальною.

Але ця обставина, напевно, залишилася б непоміченою з початкового рис.1.