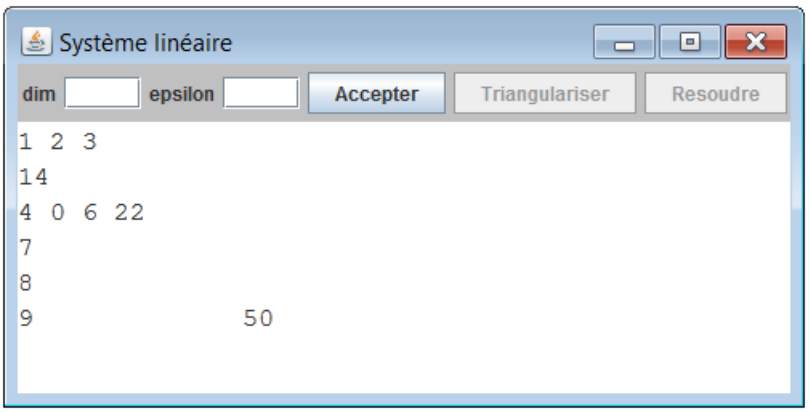
# 5. Gauss, encore...

Le but de cet exercice est de donner un habillage graphique au programme de résolution de systèmes linéaires de l'exercice précédent. Pour comprendre ce qu'il faut faire, vous pouvez faire tourner une démonstration de ce programme à l'adresse `http://henri.garreta.perso.luminy.univmed.fr/JavaMINT/gauss`), ou bien lire les explications suivantes.

L'utilisateur saisit la matrice :
- soit en tapant sa dimension $n$ dans le champ *dim*, ce qui crée un système $n \times (n+1)$ entièrement fait de zéros, qu'il lui faudra donc modifier par la suite,
- soit en tapant (ou en copiant-collant) les coefficients de la matrice dans la zone de texte ; le programme déduit alors $n$ du nombre de coefficients donnés.

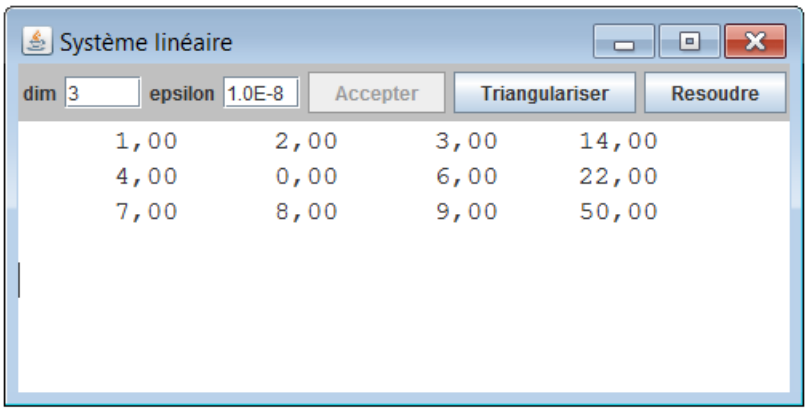
L'utilisateur *peut* donner la valeur de $\epsilon$, qui par défaut est de $10^{-8}$.

Ainsi, après la saisie, l'application peut ressembler à ceci (comme vous le voyez, les nombres constituant le système ont été écrits dans le bon ordre mais disposés n'importe comment) :

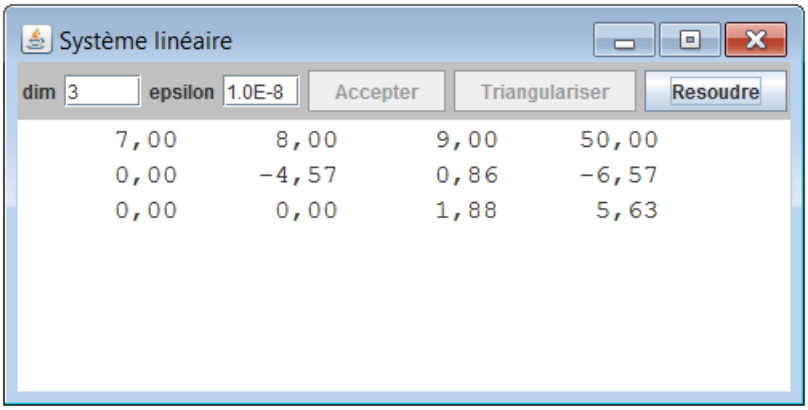



Le bouton *Accepter* commande au programme d'acquérir le système tapé dans la zone de texte. Les boutons *Triangulariser* et *Résoudre* déclenchent les opérations correspondantes. Ces boutons peuvent être actifs ou inactifs (estompés), rendant les opérations correspondantes possibles ou impossibles :
- si la matrice visible dans la zone de texte a été modifiée par l'utilisateur – c'est-à-dire qu'elle n'est pas identique à la matrice mémorisée dans le programme – seule l'opération *Accepter* doit être possible (c'est le cas représenté dans l'image ci-dessus) ;
- sinon, si l'état du système est `INITIAL` les boutons *Triangulariser* et *Résoudre* doivent être actifs ; si l'état est `TRIANGULAIRE`, seul *Résoudre* est possible. Dans les autres cas (pas de système, ou celui-ci est `RESOLU` ou `SINGULIER`) aucun bouton n'est actif : l'utilisateur ne peut que taper un autre système.

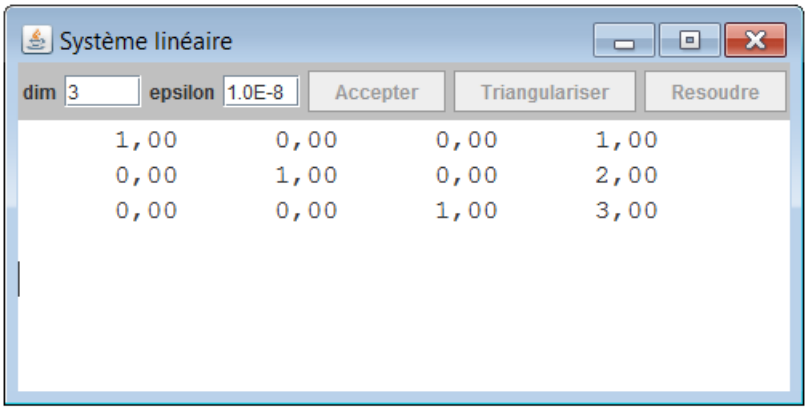
Dans le cas de l'exemple montré ci-dessus, après avoir pressé le bouton `Accepter` on aura ceci :




puis, si on presse le bouton `Triangulariser` :




enfin, si on presse le bouton *Résoudre* :



# Indications

On écrira une classe `SystemeVisible`, sous-classe de `JFrame`, comportant les variables d'instance :
- `systeme` (de type `SystemeLineaire`, cf. série d'exos précédente) le système linéaire sous-jacent ;
- `champDimension`, `champEpsilon` (de type `JTextField`) les champs de saisie ;
- `boutonAccepter`, `boutonTriangulariser`, `boutonResoudre` (de type `JButton`) ;
- `vueMatrice` (de type `JTextArea`) la zone de texte où apparaissent les coefficients ;
- `vueModifiee` (de type `boolean`) qui indique si la matrice visible a été modifiée à l'aide du clavier.

Ainsi que les méthodes :

`SystemeVisible()`

Constructeur sans argument, qui met en place l'interface graphique.

Dans une première étape, les composants sont sans effet, on ne s'occupe que de leur apparence. Il faut :
- créer les composants mentionnés (les trois boutons et les deux champs de texte) ;
- créer un panneau gris clair et lui ajouter successivement : une étiquette (`javax.swing.JLabel`) « dim », le champ de la dimension, une étiquette « epsilon », le champ de la précision et les trois boutons ; ensuite, ajouter ce panneau *au nord* du cadre en cours d'initialisation ;
- créer la zone de texte `vueMatrice` et l'ajouter *au centre* du cadre ; il est préférable de donner à cette zone une police de taille fixe (ou `"Monospaced"`)
- la construction se termine par le couple « `setSize(largeur, hauteur); setVisible(true);` »

Deuxième étape, installer le traitement des événements dont ces composants sont la source. Pour cela
- ajouter `this` comme `ActionListener` à chaque champ de texte et chaque bouton ; ajouter `this` comme `KeyListener` à la zone de texte,
- déclarer que notre classe `SystemeVisible` implémente les interfaces `ActionListener` et `KeyListener`,
- créer les méthodes (pour commencer réduites à l'affichage d'un message de contrôle) imposées par les interfaces précédentes : `actionPerformed` et `keyTyped` (ou une des deux autres méthodes de l'interface `KeyListener`).

`void keyTyped(KeyEvent e)`

Cette méthode se limite à changer éventuellement la valeur de `vueModifiee`.

`void actionPerformed(ActionEvent e)`

C'est dans cette méthode que se passent les choses les plus intéressantes. On doit d'abord déterminer quel composant est la source de l'événement `Action` en question :

```
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == champDimension) {
        traitement de l'événement « on a saisi une valeur dans le champ dim »
    }
    else if (e.getSource() == champEpsilon) {
        traitement de l'événement « on a saisi une valeur dans le champ epsilon »
    }

    else if (e.getSource() == boutonAccepter) {
        traitement de l'événement « on a pressé le bouton Accepter »
    }
    etc.
}
```

Pour la récupération des valeurs saisies dans les champs il faut savoir que le contenu de ces derniers est une chaîne de caractères, qu'on peut obtenir par la méthode `getText`, et qu'il faut ensuite convertir à l'aide de `Integer.parseInt` ou `Double.parseDouble`. Or ces méthodes peuvent lancer des exceptions (qui ne nous intéressent pas vraiment ici). Il faudra donc les appeler selon le modèle :

```
try {
    int n = Integer.parseInt(champDimension.getText());
    traitements utilisant la valeur de n
}
catch (Exception e) {
}
```

Se pose également dans cette méthode le problème de la lecture des coefficients saisis dans la zone de texte (c'est le traitement du bouton Accepter). Il est convenu que seuls des nombres, séparés par des blancs

et/ou des fins de ligne, peuvent être tapés. Un objet `StringTokenizer` est donc un bon moyen de les acquérir. Par exemple, voici une manière de connaître *combien* de nombres ont été donnés :

```
int p = 0;
try {
    StringTokenizer tokens = new StringTokenizer(vueMatrice.getText());
    while (tokens.hasMoreTokens()) {
        p++;
        tokens.nextToken();
    }
}
catch (Exception e) {
}
```

(la dimension du système est alors $n = \frac{\sqrt{1+4p}-1}{2}$).

NOTE. Le code ci-dessus est donné pour montrer comment se parcourt un objet `StringTokenizer` dans le cas général. Sachez néanmoins que, s'il s'agit uniquement de connaître le nombre de ses éléments, il est bien plus simple d'appeler sa méthode `countTokens()`.

`misaJourBoutons()`

Cette méthode privée « allume » ou « éteint » les boutons. Quatre cas sont à distinguer :
- la matrice a été modifié à la main (cela est indiqué par la valeur de `vueModifiee`),
- le système n'existe pas, ou bien il est dans l'un des états `RESOLU` ou `SINGULIER`,
- le système est dans l'état `INITIAL`,
- le système est dans l'état `TRIANGULAIRE`.

`void afficherSysteme()`

Cette méthode privée écrit les coefficients dans la zone de texte. Elle utilise la méthode `impression` de la classe `SystemeLineaire`. Le reste est une plomberie de flux, peu intéressante, que voici :

```
private void afficherSysteme() {
    ByteArrayOutputStream fluxOctets = new ByteArrayOutputStream();
    systeme.impression(new PrintStream(fluxOctets));
    vueMatrice.setText(fluxOctets.toString());
}
```

• • •