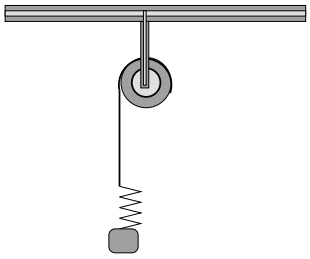
Espace d'Intégration - Dynamique d'un lève-charge
Informatique



Plan des activités

séance du 04/02	**1 -**	Compléter un programme de calcul matriciel en y ajoutant le calcul de l'exponentielle, opération intervenant dans l'expression de la solution ;
séance du 18/03	**2 -**	Calculer la solution pour des valeurs de t échantillonées ;
	-	Rendre ce calcul automatique à partir des paramètres physiques du dispositif
	3 -	Valider les calculs en comparant la courbe obtenue avec un jeu de paramètres pour lequel le résultat attendu est connu de façon analytique
séance du 25/03	**-**	Tracer la courbe théorique obtenue à partir de vos propres paramètres mesurés et la comparer à la courbe expérimentale ;
	4 -	Programmer un algorithme d'optimisation qui modifie automatiquement les valeurs des paramètres physiques pour maximiser l'adéquation entre la courbe théorique et expérimentale.
après le 25/03	**5 -**	Fusionner votre programme avec ceux des membres de votre groupe
séance du 04/04	**-**	Déboguer le programme final, lancer différents tests en interprétant les résultats.

Organisation

Jusqu'au 25 Mars

Le travail est **individuel**. Vous disposez de 3 séances de 2h pour réaliser ces travaux. A l'issue de chaque séance vous déposerez votre travail sur Campus. Ce travail pourra faire l'objet d'une évaluation.

Du 25 Mars au 14 Avril

Le travail se fait **par équipe**. Chaque équipe commence par confronter les différentes versions des travaux faits par (exclusivement) les membres de l'équipe. Elle met au point un unique programme, qu'elle améliore en vue du rapport final.

L'archive finale de l'équipe devra être déposée sur Campus au plus tard le **14 Avril à 14h**. C'est ce programme

qui devra être utilisé pour la production du rapport. Le jury devra pouvoir retrouver les résultats lui-même en exécutant cette archive.

Vos encadrants

– Gilles Chabert : gilles.chabert@mines-nantes.fr
– Romuald Debruyne : romuald.debruyne@mines-nantes.fr

1 Outil de calcul préliminaire

Objectifs

Un des objectifs principaux de cette troisième thématique est la simulation informatique de la trajectoire d'un système (le monte-charge). Cette simulation consiste à tracer la courbe à partir des formules théoriques.

Vous verrez que les formules théoriques font intervenir des calculs sur les matrices et en particulier l'exponentielle d'une matrice, dont on rappelle la définition :

$$exp(M) = I + M + \frac{1}{2}M^2 + \ldots + \frac{1}{k!}M^k + \ldots \quad (1)$$

L'objectif de ce TP est d'enrichir une classe `Matrice` (fournie) en y ajoutant le calcul approchée de l'exponentielle.

La difficulté est qu'on ne peut pas utiliser la formule (1) telle quelle. En effet, les calculs intermédiaires (telle que la puissance k^{eme} d'une matrice) peuvent conduire à des nombres tellement grand que votre ordinateur atteint ses limites.

Il faut donc utiliser un algorithme plus astucieux, et veiller à ce que les données aient le bon type.

Avant de commencer :

– Téléchargez l'archive `ei2010.zip` depuis Campus et décompressez-la dans un répertoire.
– Compilez `TestCalculs.java` puis exécutez `TestCalculs`. Cela devrait afficher :

```
16 erreurs
```

Chaque fois que vous implémentez une méthode, exécutez `TestCalculs` et modifiez votre implémentation jusqu'à ce que les tests relatifs à cette méthode passent.

1.1 Echauffement

Avant de faire des calculs sur les matrices, nous commencerons par nous « échauffer » en manipulant simplement des scalaires.

Question 1.

Implémentez dans la classe `CalculsScalaires` les méthodes suivantes (testez au fur et à mesure) :

1. `factorielle(int n)` qui retourne $n!$
2. `sommeFactorielleNaif(int n)` qui retourne $\sum_{k=0}^{n} k!$, en utilisant le premier algorithme qui vous vient à l'esprit
3. `sommeFactorielle(int n)` qui retourne la même chose avec un algorithme plus efficace
4. `sommePuissances(double t, int n)` qui retourne $\sum_{k=0}^{n} t^k$.

1.2 Complexité

Les questions suivantes sur la complexité apparaissent dans les commentaires des méthodes que vous venez d'implémenter. Répondez-y en complétant les commentaires directement dans le fichier source.

Question 2.

Q1.1 combien d'opérations environ sont effectuées par `factorielle(10)` ?

Q1.2 quelle est la complexité de `factorielle(n)` ?

Q2.1 combien d'opérations environ sont effectuées par `sommeFactorielleNaif(10)` ?

Q2.2 quelle est la complexité de `sommeFactorielleNaif(n)` ?

Q3.1 combien d'opérations environ sont effectuées par `sommeFactorielle(10)` ?

Q3.2 quelle est la complexité de `sommeFactorielle(n)` ?

Q4 comparez `sommeFactorielleNaif(100)` et `sommeFactorielle(100)` ?

Q5 que pouvez-vous dire de `sommePuissances(10,500)` ?

1.3 Calcul matriciel

Nous allons maintenant passer au calcul matriciel. Pour cela, une classe `Matrice` vous est fournie. Elle comporte déjà de nombreuses opérations sur les matrices : addition, multiplication, etc. Parcourez rapidement la documentation en ouvrant dans votre navigateur le fichier `Matrice.html` situé sous le répertoire `doc`.

Il est très important (pour la suite) que vous remarquiez que pour chaque opération, il existe deux variantes qui sont (en prenant l'exemple de la multiplication) :

- la variante qui applique l'opération (et donc modifie) la matrice courante :
  `public void multiplie(Matrice M)`
- la variante qui applique l'opération entre 2 matrices (sans les modifier) et qui retourne une nouvelle matrice contenant le résultat :
  `public static Matrice produit(Matrice a, Matrice b)`

Question 3.

En utilisant les opérations existantes, implémentez dans la classe `Matrice` les méthodes suivantes :

1. `sommePuissances(int n)` qui retourne $\sum_{k=0}^{n} A^k$, en notant A l'objet courant
2. `exponentielle(int n)` qui retourne $\sum_{k=0}^{n} \frac{1}{k!} A^k$, en notant A l'objet courant