LEPL1104 : solution de l'examen de juin 2024

1 La dérivée des engagés !

Blondinet souhaite obtenir une estimation de la dérivée à l'origine d'une fonction $u(x)$ à partir de trois valeurs U_0, U_h et U_{4h} pour les abscisses 0, h et $4h$.
On va établir une formule de dérivation numérique en écrivant :

$$u'(0) \approx \underbrace{(u^h)'(0)}_{U_0'}$$

où $u^h(x)$ est le polynôme d'interpolation par les trois points.

1. Ecrire les trois polynômes de Lagrange $\phi_0(x)$, $\phi_h(x)$ et $\phi_{4h}(x)$.
   Il faut juste écrire le 3 polynômes...

$$\boxed{\begin{aligned} \phi_0(x) &= \frac{(x-h)(x-4h)}{4h^2} \\ \phi_h(x) &= \frac{x(x-4h)}{-3h^2} \\ \phi_{4h}(x) &= \frac{x(x-h)}{12h^2} \end{aligned}}$$

   Cette question est vraiment facile et peu d'erreurs sont donc admises.
   Ecrire la formule générale de polynôme de Lagrange n'est pas accepté comme réponse !

2. Etablir l'expression de U_0' en fonction de h, U_0, U_h et U_{4h}.
   On écrit simplement :

$$\begin{aligned} U_0' &= \phi_0'(0)\, U_0 + \phi_h'(0)\, U_h + \phi_{4h}'(0)\, U_{4h} \\ &\downarrow \quad \textit{En calculant les dérivées de } \phi_i'(0) \\ &= -\frac{5h}{4h^2}\, U_0 + \frac{4h}{3h^2}\, U_h - \frac{h}{12h^2}\, U_{4h} \end{aligned}$$

   Et on peut donc finalement conclure : $\boxed{U_0' = \dfrac{-15U_0 + 16U_h - U_{4h}}{12}}$

   Cette question est une simple application d'une technique simple.
   Vérifier que la somme des trois coefficients doit valoir zéro est bon check !
   Ici, il faut juste être attentif dans les calculs :-)

3. Obtenir l'expression générale de l'erreur d'approximation de la formule.
Il suffit d'écrire des développements en série de Taylor pour U_h et U_{4h} :

$$\begin{aligned}
U_0' &= \frac{1}{h}\left(-\frac{15}{12}U_0 + \frac{16}{12}U_h - \frac{1}{12}U_{4h}\right) \\
&= \frac{1}{h}\left(-\frac{15}{12}U_0 + \frac{16}{12}\left[U_0 + hU_0' + \frac{h^2}{2}U_0'' + \frac{h^3}{6}U_0''' + \dots\right]\right. \\
&\qquad \left. -\frac{1}{12}\left[U_0 + 4hU_0' + \frac{16h^2}{2}U_0'' + \frac{64h^3}{6}U_0''' + \dots\right]\right)
\end{aligned}$$

En regroupant les termes

$$U_0' = \underbrace{\left[-\frac{15}{12} + \frac{16}{12} - \frac{1}{12}\right]}_{=0}U_0 + \underbrace{\left[\frac{16}{12} - \frac{4}{12}\right]}_{=1}U_0' + \frac{h}{2}\underbrace{\left[\frac{16}{12} - \frac{16}{12}\right]}_{=0}U_0'' + \frac{h^2}{6}\underbrace{\left[\frac{16}{12} - \frac{64}{12}\right]}_{=-4}U_0''' + \dots$$

Et on peut donc finalement conclure :

$$\boxed{u'(0) = \frac{-15U_0 + 16U_h - U_{4h}}{12} + \frac{2C_3}{3}h^2}$$

Quelques étudiants ajoutent aussi les erreurs d'arrondi du calcul en virgule flottante...
Evidemment, c'est bien, mais ce n'était pas demandé :-)

Une autre manière d'obtenir ce résultat est de dériver l'expression de l'erreur d'interpolation.
Et ensuite de l'évaluer en $x = 0$:-)

$$\begin{aligned}
u(x) &= u^h(x) + \frac{C_3(x)}{3!}x(x-h)(x-4h) \\
u'(x) &= \left(u^h\right)'(x) + + \frac{C_3'(x)}{3!}\Big[x(x-h)(x-4h)\Big] + \frac{C_3(x)}{3!}\Big[(x-h)(x-4h) + x(x-4h) + x(x-h)\Big]
\end{aligned}$$

En évaluant cette expression en $x = 0$

$$\begin{aligned}
u'(0) &= \left(u^h\right)'(0) + + \frac{C_3'(0)}{3!}\underbrace{\Big[x(x-h)(x-4h)\Big]}_{=0} + \frac{C_3(0)}{3!}\underbrace{\Big[(x-h)(x-4h) + x(x-4h) + x(x-h)\Big]}_{=4h^2} \\
u'(0) &= U_0' + \frac{2C_3}{3}h^2
\end{aligned}$$

On obtient évidemment le même résultat !

2 Réformons la méthode de Newton !

Georges-Louis utilise la méthode de Newton-Raphson pour calculer la racine $x = 1$ du polynôme

$$x^4 - x^3 - 3x^2 + 5x - 2 = 0$$

et il n'observe pas une convergence quadratique même avec $x_0 = 0.999$.

1. Décrire le calcul des itérations successives avec la méthode de Newton-Raphson.
   Il suffit juste d'écrire la relation de récurence :

$$\boxed{x_{i+1} = x_I - \frac{f(x_i)}{f'(x_i)} = x_i - \frac{x_i^4 - x_i^3 - 3x_i^2 + 5x_i - 2}{4x_i^3 - 3x_i^2 - 6x_i + 5}}$$

   Oui : il faut l'itération pour le polynôme de la question et donc calculer la dérivée de celui !
   Ou il faut que l'implémentation qui suit la contienne au moins :-)

2. Ecrire une fonction `python : x = iterateNewton(x0,tol,nmax)`
   qui applique la méthode de Nexton-Raphson pour ce problème : `x0` est le candidat initial, `tol` est la précision requise et `nmax` est le nombre maximal d'itérations. La fonction retourne `x`.
   Une implémentation possible est :

```
def iterateNewton(x0,tol,nmax):
  n = 0; delta = float("inf")
  x = x0
  while abs(delta) > tol and n < nmax :
    n = n + 1
    delta = (x-1)*(x-2)/(4x+5)
    x -= delta
  return x
```

   Dans cette implémentation, on a tiré profit que

$$\begin{array}{rcccl} f(x) & = & x^4 - x^3 - 3x^2 + 5x - 2 & = & (x+2)(x-1)^3 \\ f'(x) & = & 4x^3 - 3x^2 - 6x + 5 & = & (4x+5)(x-1)^2 \end{array}$$

   pour écrire l'incrément à chaque itération :-)
   Mais, cela n'est pas indispensable pour valider le code qui était d'une facilité déconcertante à écrire.
   C'était vraiment une sous-question qui était une partie vraiment très facile de cette question !
   Il est d'autant plus incompréhensible qu'un nombre constant d'étudiants font de manière systématique une impasse sur les codes informatiques et gaspillent bêtement des points si faciles à obtenir !

3. Pourquoi est-ce que Georges-Louis n'observe pas une convergence quadratique ?
   Il suffit juste de noter que $f(1) = f'(1) = f''(1) = 0$ pour le polynôme considéré !
   En d'autres mots, $x = 1$ est une racine triple de notre polynôme.
   Newton-Raphson ne converge quadratiquement qu'au voisinage d'un racine simple !
   Il fallait donc juste écrire :

$$\boxed{\text{\textit{La méthode ne converge pas quadratiquement car } } x = 1 \text{ \textit{est une racine triple !}}}$$

   Mentionner un racine double au lieu d'une racine triple était pardonné par le correcteur !

4. Prédire le taux de convergence qu'observe Georges-Louis pour ce cas précis !

Si la méthode converge vers $x = 1$, on peut écrire :

$$
\begin{aligned}
x_{i+1} &= x_i - \frac{f(x_i)}{f'(x_i)} \\
e_{i+1} &= e_i - \frac{f(x+e_i)}{f'(x+e_i)} \\
e_{i+1} &= e_i - \frac{f(x) + f'(x)e_i + f''(x)e_i^2/2 + f'''(x)e_i^3/6}{f'(x) + f''(x)e_i + f'''(x)e_i^2/2}
\end{aligned}
$$

$\downarrow$ *Comme $f(x) = f'(x) = f''(x) = 0$ pour $x = 1$*

$$
\begin{aligned}
e_{i+1} &= e_i - \frac{6\ f'''(1)}{2\ f'''(1)}\ \frac{e_i^3}{e_i^2} \\
e_{i+1} &= \tfrac{2}{3}\ e_i
\end{aligned}
$$

$$\boxed{e_{i+1} = \tfrac{2}{3}\ e_i}$$

On obtient donc un taux de convergence linéaire.
De manière asymptotique, l'erreur diminue d'un facteur $\frac{2}{3}$ à chaque itération.

Non : le taux de convergence ne sera pas le nombre d'or : bien essayé mais c'est raté :-)

5. Proposer une méthode de Newton-Raphson réformée avec une convergence quadratique pour ce cas.

La solution se trouvait dans une note en bas de la page 150 des notes de cours !

Il suffit de considérer que si x est une racine simple de $f(x)$, alors cette même valeur sera une racine double pour $g(x) = (f(x))^2$. L'application de la méthode de Newton-Raphson à g produit des incréments donnés par la relation :

$$\Delta x = \frac{-g(x)}{g'(x)} = \frac{-(f(x))^2}{2f(x)f'(x)} = \frac{-f(x)}{2f'(x)}$$

On observe que les incréments sont deux fois plus petits que pour la méthode de Newton-Raphson appliquée à f et on peut en déduire que :

$$\begin{aligned} e_{i+1} &= x - x_{i+1} \\ &= x - \left(x_i - \frac{f(x_i)}{2f'(x_i)} \right) \\ &= \frac{e_i}{2} + \frac{e_i f'(x_i) + f(x_i)}{2f'(x_i)} \\ &= \frac{e_i}{2} + Ce_i^2 \end{aligned}$$

La méthode ne converge donc que linéairement (terme dominant). Les petits futés remarqueront qu'il est possible d'obtenir une méthode qui converge quadratiquement vers une racine de multiplicité m en multipliant l'incrément fourni par la méthode de Newton-Raphson par la multiplicité m de la racine. Evidemment, il faut connaître a priori la multiplicité de la solution que l'on recherche. Ce qui n'est pas évident !

On peut donc proposer pour notre racine triple, plusieurs méthodes réformées pour avoir une convergence quadratique !
Trois options valables sont par exemple ;

$$\boxed{x_{i+1} = x_I - \frac{3f(x_i)}{f'(x_i)}}$$

$$\boxed{x_{i+1} = x_I - \frac{f''(x_i)}{f'''(x_i)}}$$

$$\boxed{x_{i+1} = x_I - \frac{x^2 + x - 2}{2x + 1}}$$

Par contre, diviser f et f' par $(x-1)^2$ ne fournit pas une méthode quadratique, même si c'est une manière intelligente de programmer la méthode classique !
Et pourtant, beaucoup d'étudiants répondent cela :-(

3 Une équation aux dérivées partielles pour Raoul

Sur un domaine carré $(x, y) \in [0, 1] \times [0, 1]$, on va résoudre numériquement :

$$\frac{\partial u}{\partial t}(x, y, t) = \alpha \frac{\partial^2 u}{\partial x \partial y}(x, y, t)$$

A l'instant $t = 0$, nous imposons $u(x, y, 0) = xy$.
Aux instants $t > 0$, on maintient cette valeur uniquement sur les côtés du domaine carré.
Les $m \times m$ valeurs nodales au temps $n + 1$ sont obtenues à partir de celles au temps n avec la relation :

$$U_{j,k}^{n+1} \quad = \quad U_{j,k}^{n} + \underbrace{\frac{\alpha \Delta t}{4(\Delta x)^2}}_{\beta} \left(U_{j+1,k+1}^{n} + U_{j-1,k-1}^{n} - U_{j+1,k-1}^{n} - U_{j-1,k+1}^{n} \right)$$

où $U_{j,k}^{n}$ est l'approximation de $u(X_j, Y_k, T_n)$ avec $X_j = j\Delta x$, $Y_k = k\Delta x$ et $T_n = n\Delta t$.

1. Donner les unités du paramètre α.

   Oui : c'est vraiment trop simple et pourtant beaucoup d'étudiants se trompent encore !

$$\boxed{\alpha = 1 \left[\frac{m^2}{s} \right]}$$

2. Ecrire une fonction `python : U = raoulEuler(m,n,beta)`
   qui effectue n itérations temporelles et retourne un tableau `numpy` avec $m \times m$ valeurs nodales.

   Une implémentation possible alliant simplicité et efficacité consiste à écrire.

```
def raoulEuler(m,n,beta) :
  alpha = linspace(0,1,m)
  X,Y = meshgrid(alpha,alpha)
  U = X*Y
  for t in range(n):
    U[1:-1,1:-1] += beta*(U[2:,2:] + U[:-2,:-2]
                                   - U[2:,:-2] - U[:-2,2:] )
  return U
```

   On peut aussi écrire le programme avec trois boucles imbriquées, mais c'est nettement moins efficace. Dans ce cas, il faut impérativement deux variables distinctes pour effectuer l'itération temporelle. Cétait l'unique difficulté à voir !

```
def raoulBasicEuler(m,n,beta) :
  alpha = linspace(0,1,m)
  X,Y = meshgrid(alpha,alpha)
  U = X*Y
  Unew = U.copy()
  for t in range(n):
    for i in range(1,m-1):
      for j in range(1,m-1):
        Unew[i,j] = U[i,j] + beta*(U[i+1,j+1] + U[i-1,j-1]
                                   - U[i+1,j-1] - U[i-1,j+1] )
    U = Unew.copy()
  return U
```

3. Pour quel pas de temps, la méthode est-elle stable ?
   Il faut donc estimer[1] le facteur d'amplification d'une pertubation $U^n_{j,k} = U^n e^{ik_x X_j} e^{ik_y Y_k}$

   La pertubation évoluera comme suit lors d'un pas de temps :

$$\begin{aligned}
U^{n+1}_{j,k} &= U^n_{j,k} + +\beta\Big(U^n_{j+1,k+1} + U^n_{j-1,k-1} - U^n_{j+1,k-1} - U^n_{j-1,k+1}\Big) \\
&= U^n_{j,k}\Big(1 + \beta\Big(e^{ik_x\Delta x}e^{ik_y\Delta x} + e^{-ik_x\Delta x}e^{-ik_y\Delta x} - e^{ik_x\Delta x}e^{-ik_y\Delta x} - e^{-ik_x\Delta x}e^{ik_y\Delta x}\Big)\Big) \\
&= U^n_{j,k}\Big(1 + \beta\Big(\Big(e^{ik_x\Delta x} - e^{-ik_x\Delta x}\Big)\Big(e^{ik_y\Delta x} - e^{-ik_y\Delta x}\Big)\Big)\Big) \\
&= U^n_{j}\Big(1 - 4\beta\Big(\sin(k_x\Delta x)\sin(k_y\Delta x)\Big)\Big)
\end{aligned}$$

   Pour que la pertubation ne s'accroisse pas, il faut toujours que :

$$|1 - 4\beta\Big(\sin(k_x\Delta x)\sin(k_y\Delta x))| \leq 1$$

Mais, en prenant $k_x\Delta x = \pi/2$ *et* $k_y\Delta x = -\pi/2$, $\Big\downarrow$

$$1 \leq |1 + 4\beta|$$

   La méthode est donc inconditionnellement instable : il est impossible de trouver un quelconque β qui permet d'être certain d'avoir un comportement stable. Essayer d'exécuter le programme ci-dessus et vous obtiendrez des résultats totalement incohérents !

   Presque tous les étudiants ne peuvent même pas imaginer que l'enseignant puisse imaginer une méthode aussi délirante et se trompent quasiment de manière systématique sur les trois dernières lignes mais après avoir obtenu le bon facteur d'amplication.
   A méditer !
   Et en plus, cette question était dans les annales des années précédentes !

[1] La relation $2i\sin(\theta) = e^{i\theta} - e^{-i\theta}$ pourrait être utile où i est le nombre imaginaire.