Approximation Theory and Proof Assistants : Certified Computations

par Nicolas Brisebarre, Damien Pous, Mioara Joldes et Florent Bréhard

Devoir maison à rendre jeudi 9 novembre 2023 au début du cours

Validation a posteriori de la fonction exponentielle

On souhaite mettre en place un algorithme de validation pour une approximation polynomiale φ de la fonction exponentielle $\exp : x \mapsto e^x$ sur un intervalle $[0, M]$ avec $M > 0$. Cette fonction est l'unique solution de l'équation différentielle avec conditions initiales :

$$\begin{cases} y' - y = 0, \\ y(0) = 1. \end{cases} \tag{1}$$

La norme considérée est la norme infinie $\|y\| := \max_{0 \le x \le M} |y(x)|$ sur l'espace $\mathscr{C}^0$ des fonctions continues sur $[0, M]$. On considérera aussi les sous-espaces $\mathscr{C}^1$ des fonctions dérivables sur $[0, M]$ dont la dérivée est dans $\mathscr{C}^0$, et $\mathscr{C}^\infty$ des fonctions infiniment dérivables sur $[0, M]$. Le problème (1) se réécrit alors comme $\mathscr{F}(y) = (0, 1) \in \mathscr{C}^0 \times \mathbb{R}$ où l'opérateur linéaire $\mathscr{F}$ est défini par :

$$\begin{aligned} \mathscr{F} : \mathscr{C}^1 &\to \mathscr{C}^0 \times \mathbb{R}, \\ y &\mapsto \begin{pmatrix} y' - y \\ y(0) \end{pmatrix}. \end{aligned}$$

Comme dans le cours, la construction d'un opérateur de Newton pour la validation demande d'abord de comprendre le comportement de l'opérateur inverse de $\mathscr{F}$. On définit l'opérateur linéaire $\mathscr{G} : \mathscr{C}^0 \times \mathbb{R} \to \mathscr{C}^1$:

$$\mathscr{G}(z, c) : x \mapsto ce^x + \int_0^x e^{x-t} z(t) \mathrm{d}t = e^x \left(c + \int_0^x e^{-t} z(t) \mathrm{d}t \right).$$

Question 1. Vérifier que $\mathscr{G}$ est l'inverse de $\mathscr{F}$ en montrant que :
- $\mathscr{G}(z, c)$ est une fonction de $\mathscr{C}^1$ pour tout $(z, c) \in \mathscr{C}^0 \times \mathbb{R}$;
- $\mathscr{F} \circ \mathscr{G}$ est l'identité sur $\mathscr{C}^0 \times \mathbb{R}$;
- $\mathscr{G} \circ \mathscr{F}$ est l'identité sur $\mathscr{C}^1$.

Il est clair que nous ne pouvons pas calculer directement avec $\mathscr{G}$ puisque nous ne disposons pas (encore !) d'algorithme calculant un modèle pour la fonction exp. On souhaite donc construire un opérateur linéaire $\widetilde{\mathscr{G}} : \mathscr{C}^0 \times \mathbb{R} \to \mathscr{C}^1$ qui l'approche. Pour ce faire, on suppose disposer d'approximations polynomiales φ et ψ de exp et $1/\exp$ sur $[0, M]$, satisfaisant l'objectif :

$$\varphi(x) \approx \varphi'(x) \approx e^x, \qquad \psi(x) \approx -\psi'(x) \approx e^{-x}, \qquad x \in [0, M]. \tag{2}$$

Question 2. Proposer une expression pour $\widetilde{\mathscr{G}}$ comportant φ et ψ, qui puisse s'implémenter en n'utilisant que la donnée de φ et de ψ ainsi que les opérations sur les modèles rappelées dans l'annexe.

On définit maintenant l'opérateur de Newton $\mathscr{N}(y) = y - \widetilde{\mathscr{G}}(\mathscr{F}(y) - (0, 1))$. À première vue, il s'agit d'un opérateur $\mathscr{C}^1 \to \mathscr{C}^1$ puisque :

$$\mathscr{N}(y) : x \mapsto y(x) + (1 - y(0))\varphi(x) - \varphi(x) \int_0^x \psi(t) \big(y'(t) - y(t) \big) \mathrm{d}t, \tag{3}$$

exige de pouvoir dériver la fonction y.

Question 3. Montrer que la fonction exp est un point fixe de $\mathcal{N}$, c'est-à-dire $\mathcal{N}(\exp) = \exp$.

Nous souhaitons considérer l'équation de point fixe $\mathcal{N}(y) = y$ dans $\mathscr{C}^0$ muni de la norme $\|\cdot\|$ et utiliser le théorème de point fixe de Banach pour l'algorithme de validation. Pour cela, nous allons devoir étendre $\mathcal{N} : \mathscr{C}^1 \to \mathscr{C}^1$ en un opérateur $\mathscr{C}^0 \to \mathscr{C}^0$.

Question 4. Montrer que pour tout $y \in \mathscr{C}^1$, on a :

$$\mathcal{N}(y)(x) = y(x) + \varphi(x)\left(1 - y(0) + \psi(0)y(0) - \psi(x)y(x) + \int_0^x \left(\psi'(t) + \psi(t)\right) y(t)\mathrm{d}t\right), \tag{4}$$

et que cette expression permet d'étendre $\mathcal{N}$ en un opérateur $\mathscr{C}^0 \to \mathscr{C}^0$.

On fait de plus l'hypothèse suivante, qui sera vérifiée ultérieurement par calcul dans l'algorithme de validation :

$$\varphi(x) \neq 0, \qquad \psi(x) \neq 0, \qquad \text{pour tout } x \in [0, M]. \tag{5}$$

Question 5. Montrer, en utilisant l'hypothèse (5) et l'expression (4) de $\mathcal{N}(y)$, que si $y \in \mathscr{C}^0$ est un point fixe de $\mathcal{N}$, alors $y \in \mathscr{C}^\infty$.

Question 6. Montrer, toujours sous l'hypothèse (5), que l'opérateur $\widetilde{\mathcal{G}} : \mathscr{C}^0 \times \mathbb{R} \to \mathscr{C}^1$ que vous avez défini dans la question 2 est injectif.

Question 7. En déduire que si l'hypothèse (5) est vérifiée, alors exp est l'unique point fixe de $\mathcal{N}$ dans $\mathscr{C}^0$.

Soit $\mathcal{L} : \mathscr{C}^0 \to \mathscr{C}^0$ la partie linéaire de l'opérateur affine $\mathcal{N}$, de sorte que $\mathcal{N}(y_1) - \mathcal{N}(y_2) = \mathcal{L}(y_1 - y_2)$:

$$\mathcal{L}(y)(x) = \varphi(x)(\psi(0) - 1)\, y(0) + (1 - \varphi(x)\psi(x))\, y(x) + \varphi(x) \int_0^x \left(\psi'(t) + \psi(t)\right) y(t)\mathrm{d}t.$$

Question 8. Montrer que la quantité μ définie comme suit :

$$\mu = \|\varphi\|\,|\psi(0) - 1| + \|1 - \varphi\psi\| + M\|\varphi\|\,\|\psi' + \psi\|,$$

est une borne supérieure pour $\|\mathcal{L}\|$, c'est-à-dire que $\|\mathcal{L}(y)\| \leq \mu\|y\|$ pour tout $y \in \mathscr{C}^0$.

Question 9. Montrer que μ peut-être rendu aussi petit que souhaité dès lors que l'on choisit des approximations polynomiales φ et ψ de plus en plus précises vis-à-vis de l'objectif (2).

Question 10. Donner une expression pour $\mathcal{N}(\varphi) - \varphi$ et montrer, en faisant apparaître des termes censés être petits, que le défaut $b = \|\mathcal{N}(\varphi) - \varphi\|$ tend vers 0 quand l'erreur d'approximation de φ et ψ dans l'objectif (2) tend vers 0 (on pourra utiliser l'expression (3) pour $\mathcal{N}$ puisque φ est $\mathscr{C}^\infty$).

Question 11. Rappeler l'énoncé du théorème du point fixe de Banach appliqué au cas présent. Quelle borne donne-t-il sur $\|\varphi - \exp\|$? Quelle condition sur μ faut-il vérifier pour que cela ait un sens ?

Question 12. Montrer que cette même condition sur μ implique que l'hypothèse (5) est vérifiée (et donc que tout notre raisonnement est correct !)

On se penche maintenant sur l'implémentation de cette méthode de validation. Pour ce faire, on consultera l'annexe qui décrit le langage de programmation fictif que l'on utilisera avec les types de données et fonctions à disposition.

Question 13. Compléter ci-dessous l'algorithme de validation `mexp_aux` qui, étant donné des approximations polynomiales φ et ψ de exp et $1/\exp$ satisfaisant l'objectif (2), calcule une borne d'erreur entièrement rigoureuse pour $\|\varphi - \exp\|$ et renvoie le modèle correspondant. On utilisera pour cela uniquement les fonctions listées en annexe et on pourra prendre modèle sur la fonction `mdiv_aux` donnée en exemple.

```
function mexp_aux(phi : poly, psi : poly) : model
    ...
end
```

Annexe

Dans le cadre de ce devoir, on se donne un langage fictif avec quatre types de données numériques :

Les nombres en virgule flottante («flottants», `float`), comme `0.0`, `1.0` ou `-7.5`, disposent des opérations arithmétiques usuelles : `+`, `-`, `*` et `/`, des opérateurs de comparaison `<`, `<=`, `==`, `>=`, `>` et `!=`, ainsi que d'une fonction `abs(x)` calculant la valeur absolue de `x`.

Les intervalles de flottants (`interval`), qui permettent de calculer rigoureusement sur des nombres réels, disposent des mêmes opérations que les flottants, qui seront notées de la même façon. Si une opération binaire fait intervenir un intervalle et un flottant, on supposera que le flottant est automatiquement converti en intervalle de sorte que le résultat est calculé rigoureusement.

Les approximations polynomiales (`poly`) sont des polynômes (en base quelconque) à coefficients flottants, servant de brique de base pour les *modèles*. Les opérations disponibles sont les suivantes :

```
function pzer() : poly                          // 0 (polynôme constant)
function pone() : poly                          // 1 (polynôme constant)
function pcst(c : float) : poly                 // c (polynôme constant)
function padd(p : poly, q : poly) : poly        // p + q
function psub(p : poly, q : poly) : poly        // p - q
function pmul(p : poly, q : poly) : poly        // p * q
function peval(p : poly, x : float) : float     // p(x)
function pnorm(p : poly) : float                // ||p||
function pdiff(p : poly) : poly                 // derivative of p
function pprim(p : poly) : poly                 // primitive of p
```

Quelques remarques :

- `pnorm(p)` calcule une borne supérieure pour $\|p\| = \max_{0 \le x \le M} |p(x)|$.
- `pdiff(p)` calcule la dérivée p' de p.
- `pprim(p)` calcule la primitive de p s'annulant en 0, c'est-à-dire $x \mapsto \int_0^x p(t)\mathrm{d}t$.

Les modèles (`model`) sont des approximations polynomiales rigoureuses sur un intervalle donné, disons `[0,M]` (et on supposera que l'on dispose de `M` comme une constante globale). Un modèle `F` est constitué d'une approximation polynomiale `p` et d'une borne d'erreur `r` de sorte que pour $f \in \mathscr{C}^0$, on ait :

$$f \in \mathsf{F} \qquad \Leftrightarrow \qquad \|f - \mathsf{p}\| := \max_{0 \le x \le M} |f(x) - \mathsf{p}(x)| \le \mathsf{r}.$$

Les opérations disponibles sont les suivantes :

```
function model(p : poly, r : float)                 // constructeur
function mzer() : model                             // 0 (modèle constant)
function mone() : model                             // 1 (modèle constant)
function mcst(c : interval) : model                 // c (modèle constant)
function madd(F : model, G : model) : model         // F + G
function msub(F : model, G : model) : model         // F - G
function mmul(F : model, G : model) : model         // F * G
function meval(F : model, x : interval) : interval  // F(x)
function mnorm(F : model) : interval                // ||F||
function mprim(F : model) : model                   // primitive of F
```

Quelques remarques :

- Si `r` est un `interval`, alors `model(p,r)` est construit en prenant comme borne d'erreur la borne supérieure de `r`.
- Pour additionner rigoureusement un polynôme `p` et un modèle `F` (ou deux polynômes `p` et `q`), on peut écrire `madd(p,F)` (ou `madd(p,q)`)et supposer que `p` et `q` sont automatiquement convertis en modèles. Il en va de même pour les autres opérations prenant des arguments de type `model`.
- `meval(F,x)` est un intervalle contenant $f(x)$ pour tout $x \in$ `x` et $f \in$ `F`.
- `mnorm(F)` est un intervalle contenant une borne m vérifiant $m \ge \|f\|$ pour tout $f \in$ `F`.
- `mprim(F)` calcule un modèle contenant la primitive s'annulant en 0 de tout $f \in$ `F`, mais il n'est en revanche *pas possible de dériver un modèle* car une fonction $f \in \mathsf{F} \subset \mathscr{C}^0$ n'est pas forcément dérivable !

Un exemple. Pour illustrer l'utilisation de ce langage fictif, des types de données et des fonctions, voici un algorithme de validation pour la division de modèles vue en cours.

L'algorithme `mdiv_aux` procède ainsi. Partant de modèles `F` et `G`, on souhaite calculer la division `F/G`. Pour ce faire, l'utilisateur doit fournir à l'algorithme, en plus de `F` et de `G`, des approximations polynomiales `h` $\approx$ `F/G` et `w` ≈ 1/`G`. `mdiv_aux` calcule alors des bornes `mu` et `b` pour $\|1 - \mathsf{wG}\|$ et $\|\mathsf{w}(\mathsf{Gh} - \mathsf{F})\|$ respectivement. Si `mu` est inférieur à 1, il calcule la borne `r` donnée par le théorème de point fixe de Banach et renvoie finalement le modèle de partie polynomiale `h` et de reste `r`. Si la condition n'est pas vérifiée, l'algorithme échoue avec un message d'erreur.

```
function mdiv_aux(h : poly, w : poly, F : model, G : model) : model
    K1 = msub(mone, mmul(w, G))
    K2 = mmul(w, msub(mmul(G, h), F))
    mu = mnorm(K1)
    b = mnorm(K2)
    if mu < 1.0 then
        r = b / (1.0 - mu)
        return model(h, r)
    else
        error "condition mu < 1 non vérifiée"
    end
end
```