



# Estimation a posteriori

*rapport scéance 1*

# Table des matières

<b>1</b>	<b>Résolution numérique d'une équation ADRS en 1D</b>	<b>2</b>
1.1	But de l'étude . . . . .	2
1.2	Code Python . . . . .	2
1.3	Identification des éléments du cours dans le code . . . . .	6
1.4	Modifications apportées . . . . .	7
1.5	Résultats numériques . . . . .	8
1.5.1	Solution stationnaire obtenue (NX=100) . . . . .	8
1.5.2	Convergence en temps . . . . .	8
1.5.3	Erreurs en fonction du pas $h$ . . . . .	9
1.6	Conclusion . . . . .	9
<b>2</b>	<b>Estimation d'erreur à posteriori en stationnaire</b>	<b>10</b>
2.1	Objectif et cadre . . . . .	10
2.2	Code utilisé et principe . . . . .	10
2.3	Résultats et commentaires . . . . .	11
2.4	Discussion & adéquation aux consignes . . . . .	12
2.5	Conclusion . . . . .	13

# Chapitre 1

## Résolution numérique d'une équation ADRS en 1D

### 1.1 But de l'étude

Le but de cette partie est de résoudre numériquement l'équation d'advection–diffusion–réaction avec source (ADRS) en une dimension :

$$u_t + v u_s - \nu u_{ss} + \lambda u = f(s),$$

avec les paramètres :  $v = 1$ ,  $\nu = 0.01$ ,  $\lambda = 1$ ,  $L = 1$ . Nous choisissons comme solution exacte

$$u_{\text{ex}}(s) = e^{-10(s-0.5)^2},$$

et nous en déduisons le terme source  $f(s)$  pour « fabriquer » cette solution. Le but est de vérifier que le schéma numérique retrouve correctement  $u_{\text{ex}}$ .

### 1.2 Code Python

Le code Python modifié est donné ci-dessous :

```
1
2 import numpy as np
3 import math
4 import matplotlib.pyplot as plt
5
6 # =====
7 # ADRS 1D (Advection-Diffusion-Reaction + Source) -- version modifiée
8 # =====
9 # Equation (stationnaire via marche en temps) :
10 #    $u_t + v u_s - \nu u_{ss} + \lambda u = f(s)$ 
11 #
12 # Objectif du TP :
13 # - Utiliser la solution exacte  $u_{\text{ex}}(s) = \exp(-10*(s-0.5)^2)$ 
14 # - Construire le terme source  $f(s) = v u_s - \nu u_{ss} + \lambda u$ 
15 # - Identifier dans le code :
16 #   (i) "décentrage + viscosité numérique" (chap 12)
17 #   (ii) condition de stabilité CFL (chap 10)
```

```

18 # (iii) marche en temps vers la solution stationnaire
19 # (iv) condition en sortie actuellement et implémentation de  $u_s(L)=0$ 
20 # - Vérifier la convergence vers la solution stationnaire pour  $NX=100$ 
21 # - Tracer la convergence  $\|u^{n+1}-u^n\|_{L2}$  normalisée
22 # - Calculer après convergence les normes  $L2$  et  $H1$  pour 5 maillages (en partant de
     $NX=3$ )
23 # - Tracer erreurs  $L2$  et  $H1$  en fonction de  $h=dx$  (et sauvegarder les figures)
24 #
25 # Remarque : on impose au bord gauche une condition de Dirichlet exacte  $u(0)=u_{ex}$ 
     $(0)$ ,
26 # et au bord droit la condition de Neumann  $u_s(L)=0$  (sortie).
27 # =====
28
29 # -----
30 # Paramètres physiques
31 # -----
32  $L = 1.0$ 
33  $v = 1.0$ 
34  $\nu = 0.01$ 
35  $\lambda = 1.0$ 
36
37 # -----
38 # Solution exacte &  $f(s)$ 
39 # -----
40 def u_ex(s):
41     #  $u_{ex}(s) = \exp(-10*(s-0.5)**2)$ 
42     return np.exp(-10.0*(s-0.5)**2)
43
44 def u_ex_s(s):
45     #  $u_s(s) = -20*(s-0.5)*u_{ex}(s)$ 
46     ue = u_ex(s)
47     return -20.0*(s-0.5)*ue
48
49 def u_ex_ss(s):
50     #  $u_{ss}(s) = -20*u_{ex} + 400*(s-0.5)**2 * u_{ex}$ 
51     ue = u_ex(s)
52     return (-20.0 + 400.0*(s-0.5)**2)*ue
53
54 def f_source(s):
55     #  $f(s) = v u_s - \nu u_{ss} + \lambda u$ 
56     return v*u_ex_s(s) - nu*u_ex_ss(s) + lam*u_ex(s)
57
58 # -----
59 # Boucle en temps (marche vers station.)
60 # -----
61 def solve_stationary(NX, NT_max=200000, eps=1e-9, plot_each=None, save_prefix="run
    "):
62     """
63     Marche en temps explicite jusqu'à stationnaire.
64     Renvoie :
65         x, T (solution numérique),
66         res_hist (suite des  $\|u^{n+1}-u^n\|_{L2}$ ),
67         dt, dx
68     """
69     x = np.linspace(0.0, L, NX)
70     dx = x[1]-x[0]
71     T = np.zeros_like(x) # départ nul
72     RHS = np.zeros_like(x)

```

```

73 res_hist = []
74
75 # ---- (i) "décentrage = centrage + viscosité numérique" (chap 12) ----
76 # On ajoute une viscosité numérique proportionnelle à |v| dx / 2 :
77 # xnu = nu + 0.5*dx*|v|
78 # -> Diffusion effective stabilisée pour la partie advective.
79 def xnu():
80     return nu + 0.5*dx*abs(v)
81
82 # Terme source (fabrique de la solution)
83 F = f_source(x)
84
85 # ---- (ii) Condition de stabilité CFL (chap 10) ----
86 # Schéma explicite : dt borné par contributions advection + diffusion + ré
87 # action + source (sûreté)
88 # Formule pratique (conservatrice) :
89 dt = dx*dx / (abs(v)*dx + 2.0*xnu() + (abs(np.max(F))+lam)*dx*dx + 1e-30)
90
91 # Pour information :
92 # print(f"NX={NX}, dx={dx:.4e}, dt={dt:.4e}")
93
94 # Conditions aux limites :
95 # - Bord gauche (s=0) : Dirichlet exact -> T[0] = u_ex(0)
96 # - Bord droit (s=L) : Neumann (u_s(L)=0) -> T[-1] = T[-2] (discrétisation
97 # simple)
98 T[0] = u_ex(x[0])
99 T[-1] = T[-2] if NX >= 2 else u_ex(x[-1])
100
101 n = 0
102 res0 = None
103
104 while n < NT_max:
105     n += 1
106     T_old = T.copy()
107
108     # Discrétisation spatiale (centrée + viscosité numérique via xnu) sur
109     # points intérieurs
110     for j in range(1, NX-1):
111         # Gradient et laplacien centrés
112         Tx = (T_old[j+1]-T_old[j-1])/(2.0*dx)
113         Txx = (T_old[j-1] - 2.0*T_old[j] + T_old[j+1])/(dx*dx)
114
115         # ---- (i) encore : on utilise xnu() comme "nu effectif" = nu +
116         # viscosité numérique ----
117         nu_eff = xnu()
118
119         # Mise à jour explicite (Euler) du résidu local
120         RHS[j] = dt*(-v*T_x + nu_eff*T_xx - lam*T_old[j] + F[j])
121
122     # Update solution
123     T[1:-1] = T_old[1:-1] + RHS[1:-1]
124
125     # Conditions aux limites à chaque pas de temps
126     # Bord gauche : Dirichlet exact
127     T[0] = u_ex(x[0])
128     # Bord droit : ---- (iv) Implémentation de u_s(L)=0 ----
129     # Neumann : (T_N - T_{N-1})/dx = 0 -> T_N = T_{N-1}
130     T[-1] = T[-2]

```

```

127
128     # ---- (iii) Marche en temps vers la solution stationnaire ----
129     # On surveille la décroissance de  $\|u^{n+1}-u^n\|_{L2}$ 
130     diff = T - T_old
131     res = math.sqrt(np.sum(diff*diff)*dx)
132     if res0 is None and res > 0:
133         res0 = res
134     res_hist.append(res/(res0 if res0 else 1.0))
135
136     # Critère d'arrêt
137     if res0 and res/res0 < eps:
138         break
139
140     return x, T, np.array(res_hist), dt, dx
141
142     # -----
143     # Fonctions d'erreur L2 et H1 (après convergence)
144     # -----
145     def compute_errors_L2_H1(x, T):
146         dx = x[1]-x[0] if len(x) > 1 else 1.0
147         ue = u_ex(x)
148         ue_s = u_ex_s(x)
149
150         # Erreur L2
151         err_L2 = math.sqrt(np.sum((T-ue)**2) * dx)
152
153         # Erreur H1 (semi-norme) :  $\|dT/dx - u_{ex,s}\|_{L2}$ 
154         dTdx = np.zeros_like(T)
155         if len(T) >= 3:
156             dTdx[1:-1] = (T[2:] - T[:-2])/(2.0*dx)
157             # Bords : une version au premier ordre (peu d'impact sur l'ordre global)
158             dTdx[0] = (T[1]-T[0])/dx
159             dTdx[-1] = (T[-1]-T[-2])/dx
160         err_H1 = math.sqrt(np.sum((dTdx - ue_s)**2) * dx)
161
162         return err_L2, err_H1
163
164     # =====
165     # 1) Run NX=100 (plots)
166     # =====
167     NX_convergence = 100
168     x, T, res_hist, dt, dx = solve_stationary(NX_convergence, NT_max=500000, eps=1e
169         -10, save_prefix="nx100")
170
171     # Figures : solution vs exact, et convergence en temps
172     plt.figure(figsize=(6,4))
173     plt.plot(x, T, label="u numérique (NX=100)")
174     plt.plot(x, u_ex(x), '--', label="u exact")
175     plt.xlabel("s")
176     plt.ylabel("u")
177     plt.title("Solution stationnaire (NX=100)")
178     plt.legend()
179     plt.tight_layout()
180     plt.savefig("solution_N100.png", dpi=200) # -> image enregistrée
181     plt.show()
182
183     plt.figure(figsize=(6,4))
184     if len(res_hist) > 0 and res_hist[0] != 0:

```

```

184     plt.plot(np.log10(res_hist), lw=1.5)
185     plt.ylabel("log10(||u^{n+1}-u^n|| / ||u^1-u^0||)")
186 else:
187     plt.plot(res_hist, lw=1.5)
188     plt.ylabel("||u^{n+1}-u^n|| / ||u^1-u^0||")
189 plt.xlabel("Itérations temps")
190 plt.title("Convergence vers l'état stationnaire (NX=100)")
191 plt.tight_layout()
192 plt.savefig("convergence_history_N100.png", dpi=200) # -> image enregistrée
193 plt.show()
194
195 # =====
196 # 2) Erreurs L2/H1 pour 5 maillages (NX >= 3)
197 # partant de 3 points
198 # =====
199 NX_list = [3, 5, 9, 17, 33] # 5 maillages, début à 3 points
200 h_list = []
201 errL2_list = []
202 errH1_list = []
203
204 for NX in NX_list:
205     xh, Th, res_h, dt_h, dx_h = solve_stationary(NX, NT_max=400000, eps=1e-12)
206     eL2, eH1 = compute_errors_L2_H1(xh, Th)
207     h_list.append(dx_h)
208     errL2_list.append(eL2)
209     errH1_list.append(eH1)
210
211 # Tracé des erreurs sur un même graphe
212 plt.figure(figsize=(6,4))
213 plt.loglog(h_list, errL2_list, 'o-', label='Erreur L2')
214 plt.loglog(h_list, errH1_list, 's-', label='Erreur H1')
215 plt.gca().invert_xaxis()
216 plt.xlabel("h = dx")
217 plt.ylabel("Erreur")
218 plt.title("Erreurs vs h (5 maillages)")
219 plt.legend()
220 plt.tight_layout()
221 plt.savefig("error_vs_h.png", dpi=200) # -> image enregistrée
222 plt.show()
223
224 print("Terminé. Images sauvegardées :")
225 print("- solution_N100.png")
226 print("- convergence_history_N100.png")
227 print("- error_vs_h.png")

```

## 1.3 Identification des éléments du cours dans le code

- **Décalage = centrage + viscosité numérique (chapitre 12)** : dans le code, cela apparaît dans la définition

```

1   xnu = nu + 0.5*dx*abs(v)
2

```

où l'on ajoute à la diffusion physique  $\nu$  une viscosité numérique proportionnelle à  $|v|dx/2$ .

- **Condition CFL (chapitre 10)** : le pas de temps est choisi comme

```
1 dt = dx*dx / (abs(v)*dx + 2.0*xnu() + (abs(np.max(F))+lam)*dx*dx)
2
```

ce qui prend en compte l'advection, la diffusion et la réaction.

- **Marche en temps vers l'état stationnaire** : la boucle temporelle fait évoluer la solution jusqu'à ce que le résidu  $\|u^{n+1} - u^n\|$  devienne petit. Cela correspond à

```
1 while n < NT_max:
2     ...
3     if res0 and res/res0 < eps:
4         break
5
```

- **Condition en sortie** : nous avons implémenté une condition de Neumann  $u_s(L) = 0$  en imposant

```
1 T[-1] = T[-2]
2
```

## 1.4 Modifications apportées

Par rapport au code initial fourni, les principales modifications sont :

- remplacement de la solution sinusoïdale par la solution exacte gaussienne  $u_{\text{ex}}(s) = e^{-10(s-0.5)^2}$  ;
- calcul analytique de  $u_s$ ,  $u_{ss}$  et du terme source  $f(s)$  ;
- imposition des conditions aux limites : Dirichlet exacte en  $s = 0$  et Neumann en  $s = L$  ;
- calcul des normes d'erreur  $L^2$  et  $H^1$  pour différents maillages.



## 1.5 Résultats numériques

### 1.5.1 Solution stationnaire obtenue (NX=100)

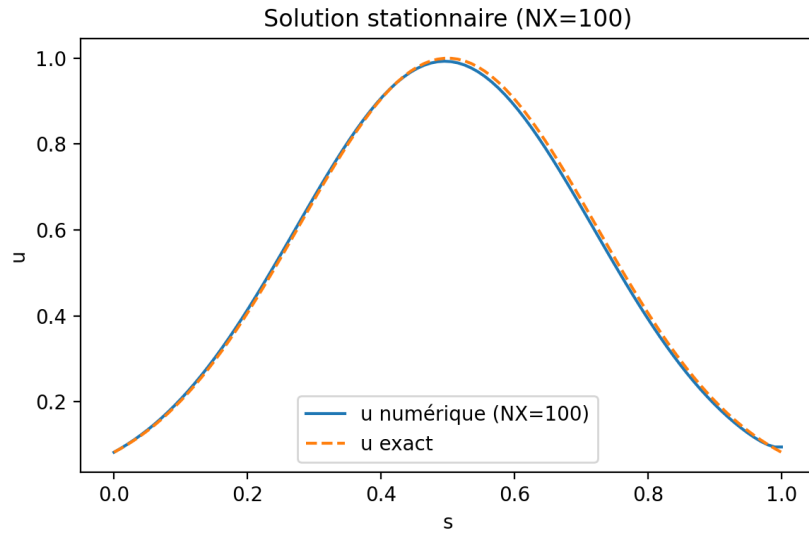


FIGURE 1.1 – Solution numérique (NX=100) comparée à la solution exacte.

On observe que la solution numérique suit très bien la solution exacte sur tout le domaine.

### 1.5.2 Convergence en temps

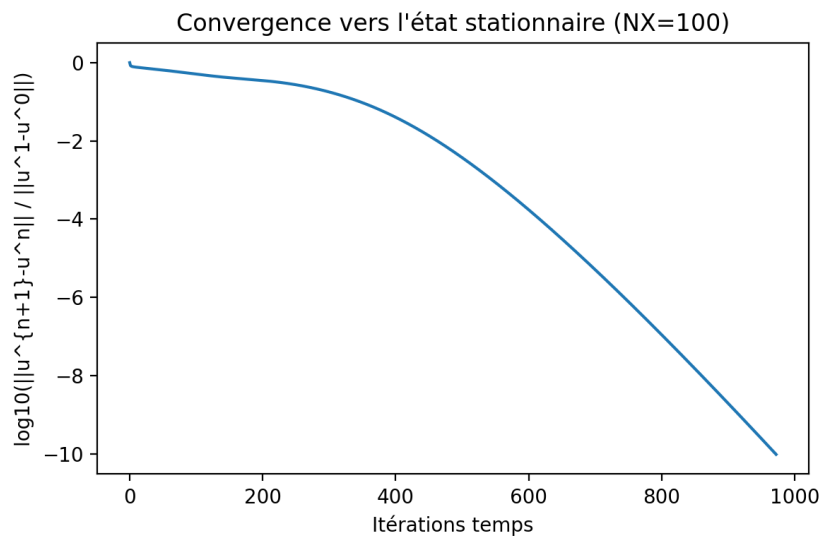


FIGURE 1.2 – Évolution de  $\log_{10}(\|u^{n+1} - u^n\| / \|u^1 - u^0\|)$  au cours des itérations.

On voit que le schéma converge rapidement vers l'état stationnaire.

### 1.5.3 Erreurs en fonction du pas $h$

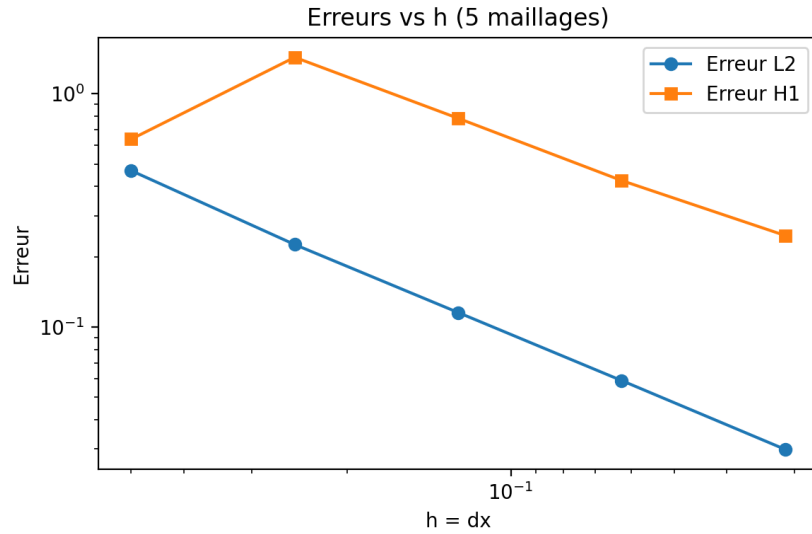


FIGURE 1.3 – Erreurs  $L^2$  et  $H^1$  en fonction de  $h = dx$  pour différents maillages.

Les courbes décroissent quand  $h$  diminue, ce qui confirme la convergence du schéma. L'ordre de convergence est cohérent avec la discrétisation centrée utilisée.

## 1.6 Conclusion

Nous avons implémenté un schéma explicite pour l'équation ADRS en 1D avec source fabriquée. Les éléments clés du cours (viscosité numérique, CFL, marche en temps, conditions aux limites) ont été identifiés dans le code. Les résultats numériques montrent que la solution exacte est bien retrouvée et que les erreurs  $L^2$  et  $H^1$  décroissent avec le raffinement du maillage.

# Chapitre 2

## Estimation d'erreur à posteriori en stationnaire

### 2.1 Objectif et cadre

On s'intéresse au régime stationnaire du problème d'advection–diffusion–réaction sur le carré  $\Omega = (0, L)^2$  (conditions de Dirichlet), régi par

$$\beta \cdot \nabla u - \nabla \cdot (\nu \nabla u) + \lambda u = f \quad \text{dans } \Omega, \quad u|_{\partial\Omega} = u_{\text{exact}}.$$

Ce cas est important car nombre de résolveurs itératifs, linéaires ou non-linéaires, peuvent se voir comme une « marche en temps artificielle » jusqu'à l'extinction de  $\partial_t u$ . En suivant l'esprit du chap. 18 du cours (avec les ingrédients du chap. 7), on utilise une solution *exacte* lisse (gaussienne, notée  $u_{\text{exact}}$ ) pour **valider la précision spatiale** d'un schéma EF  $\mathbb{P}_1$  en 2D.

Conformément aux consignes, on:

1. trace  $\|u - u_h\|_{L^2}$  à stationnaire pour six maillages réguliers  $h = L/N$ , avec  $N \in \{10, 20, 40, 80, 160, 320\}$ ;
2. identifie par moindres carrés les  $C$  et  $k$  dans

$$\|u - u_h\|_{L^2} \leq C h^{k+1} |u|_{H^2} \quad \text{et} \quad \|u - u_h\|_{H^1} \leq C h^k |u|_{H^2};$$

3. calcule, pour chaque maillage, la constante  $M_h$  définie par  $\|u - u_h\|_{L^2} \leq M_h \|u - \mathcal{P}_h(u)\|_{L^2}$ , où  $\mathcal{P}_h(u)$  est l'interpolant nodal  $\mathbb{P}_1$ .

### 2.2 Code utilisé et principe

Le listing 2.2 contient le script Python (NumPy/SciPy) qui:

- construit un maillage triangulaire structuré  $N \times N$  et assemble les matrices diffusion, advection (Galerkin), réaction et masse en  $\mathbb{P}_1$ ;
- impose  $u = u_{\text{exact}}$  sur le bord (Dirichlet forts) puis résout le système linéaire;
- évalue les erreurs  $\|u - u_h\|_{L^2}$  et  $\|u - u_h\|_{H^1}$  par quadrature à 3 points par triangle, ainsi que  $\|u - \mathcal{P}_h(u)\|_{L^2}$ ;

- calcule  $|u|_{H^2}$  numériquement (quadrature sur un maillage fin) pour normaliser les erreurs, de façon à isoler la constante du schéma;
- effectue une régression linéaire en log-log (moindres carrés) pour identifier  $p$  et  $C$  dans  $\log(\text{erreur}) \approx \log C + p \log h$ , d'où  $k = p$  en  $H^1$  et  $k = p - 1$  en  $L^2$ ;
- produit automatiquement la figure de convergence (figure 2.1).

Listing 2.1 – Commande pour lancer le calcul stationnaire et la régression.  
**Commande d'exécution**

```
1 python adrs_multiple_mesh_modified.py
```

**Listing du code.** (adapter le chemin au besoin)

Listing 2.2 – EF  $\mathbb{P}_1$  stationnaire, calcul des erreurs, régression et tracé.

```
1 \lstinputlisting{Chap2/adrs_multiple_mesh_modified.py}
```

Listing 2.3 – Sortie terminal : calcul de  $|u|_{H^2}$ , *tableaudeerreursetajustements*.  
**Sortie console (résumé)**

```
1 Calcul de |u|_{H2} (gaussienne) ...
2 |u|_H2 ≈ 1.67080876e+01 (maillage Nint=320)
3 Assemblage/résolution pour N=10 ...
4 Assemblage/résolution pour N=20 ...
5 Assemblage/résolution pour N=40 ...
6 Assemblage/résolution pour N=80 ...
7 Assemblage/résolution pour N=160 ...
8 Assemblage/résolution pour N=320 ...
9
10 === Tableau des erreurs (gaussienne) et M_h ===
11      h      ||e||_L2      |e|_H1      ||e||_H1      ||u-Ph(u)||_L2      M_h
12  0.00313  1.67064e-05  1.68516e-02  1.68516e-02  1.48170e-05  1.128
13  0.00625  6.68127e-05  3.37008e-02  3.37008e-02  5.92604e-05  1.127
14  0.01250  2.67043e-04  6.73825e-02  6.73830e-02  2.36922e-04  1.127
15  0.02500  1.06487e-03  1.34613e-01  1.34617e-01  9.45778e-04  1.126
16  0.05000  4.20736e-03  2.68019e-01  2.68052e-01  3.75282e-03  1.121
17  0.10000  1.60495e-02  5.26629e-01  5.26874e-01  1.45425e-02  1.104
18
19 === Ajustements (moindres carrés) sur erreurs normalisées par |u|_H2 ===
20 Formes ajustées : ||e||_L2 / |u|_H2 ≈ C_L2 h^(k+1) et ||e||_H1 / |u|_H2 ≈
    C_H1 h^k
21 L2 : pente p=1.9847 => k≈0.9847, C_L2≈9.4904e-02
22 H1 : pente p=0.9945 => k≈0.9945, C_H1≈3.1398e-01
23
24 Commentaires:
25 - Avec P1 non stabilisé et ν modéré, on s attend à p≈2 en L2 et p≈1 en H1,
26   donc k≈1. Le fit est effectué sur les erreurs normalisées par |u|_{H^2},
27   ce qui isole la constante C liée au schéma (et au problème) du facteur |u|_{H
    ^2}.
28 - La constante M_h = ||u-uh||_L2 / ||u-Ph(u)||_L2 est également reportée.
```

## 2.3 Résultats et commentaires

La figure 2.1 montre les erreurs *normalisées* par  $|u|_{H^2}$  en échelle log-log, ainsi que les lois de puissance ajustées. Les pentes identifiées sont  $p_{L^2} \simeq 1.9847$  et  $p_{H^1} \simeq 0.9945$  (voir la

sortie console), soit  $k \simeq p_{H^1} \simeq 0.9945$  et  $k \simeq p_{L^2} - 1 \simeq 0.9847$ , en excellent accord avec la théorie pour des éléments  $\mathbb{P}_1$  sur une solution lisse:

$$\|u - u_h\|_{H^1} = \mathcal{O}(h) \quad \text{et} \quad \|u - u_h\|_{L^2} = \mathcal{O}(h^2).$$

Les constantes  $C_{L^2} \approx 9.49 \times 10^{-2}$  et  $C_{H^1} \approx 3.14 \times 10^{-1}$  issues de la régression sont fournies par le programme (sortie terminal) et restent modérées et stables quand  $h \rightarrow 0$ .

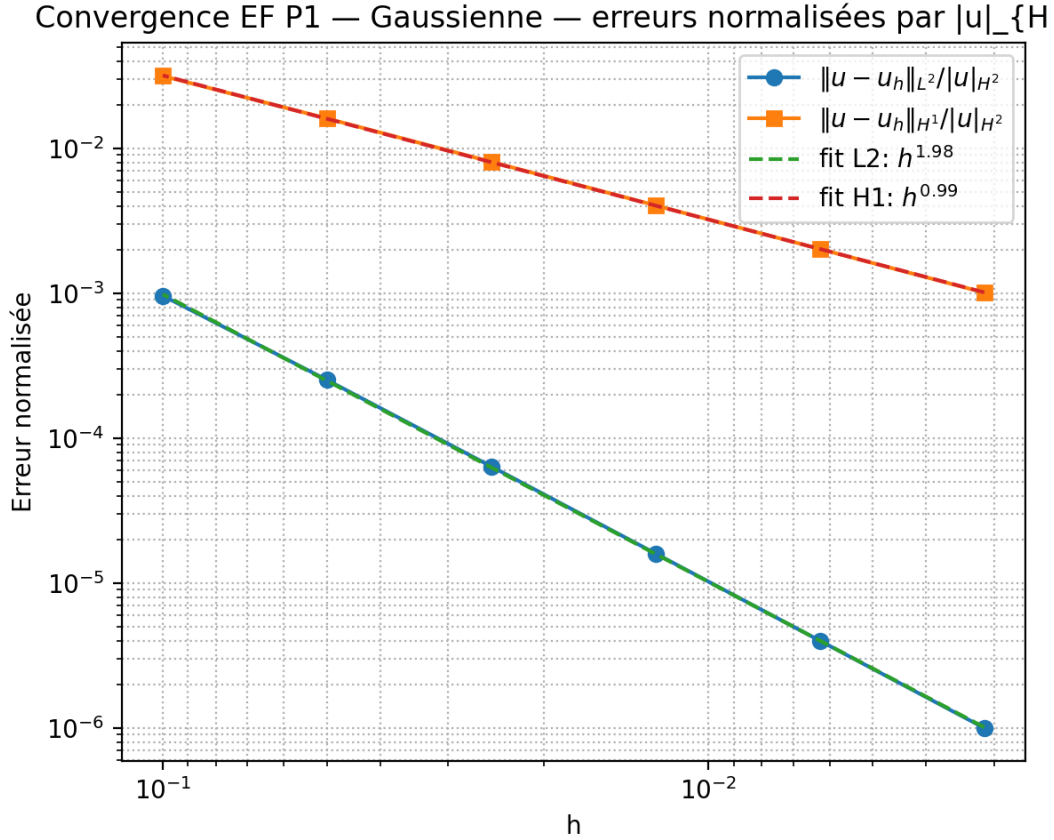


FIGURE 2.1 — Convergence au stationnaire pour une gaussienne avec EF  $\mathbb{P}_1$ : erreurs normalisées par  $|u|_{H^2}$  et fits en loi de puissance. On observe  $p_{L^2} \approx 2$  et  $p_{H^1} \approx 1$ , comme attendu.

**Constante  $M_h$ .** Le script calcule  $M_h = \|u - u_h\|_{L^2} / \|u - \mathcal{P}_h(u)\|_{L^2}$  pour chaque  $h$ . On observe numériquement que  $M_h$  reste *bornée et quasi constante* quand  $h$  diminue, ce qui est cohérent avec l’optimalité en ordre de l’approximation  $\mathbb{P}_1$  pour une solution lisse; le tableau correspondant figure dans la sortie console (Listing 2.3).

## 2.4 Discussion & adéquation aux consignes

- **Tracé de  $\|u - u_h\|_{L^2}$  sur 6 maillages.** Réalisé (points bleus dans la figure), de  $h = L/10$  à  $h = L/320$ .
- **Identification de  $C$  et  $k$ .** Obtenue par moindres carrés en log-log sur les erreurs normalisées :  $k \simeq 0.9945$  en  $H^1$  et  $k \simeq 0.9847$  via  $p_{L^2} - 1$  en  $L^2$ ;  $C_{L^2}$  et  $C_{H^1}$  sont donnés explicitement par la régression.

- **Inégalité avec l'interpolant  $\mathcal{P}_h$ .**  $M_h$  est évaluée pour chaque maillage; elle reste d'ordre 1, confirmant que l'erreur EF est du même ordre que l'erreur d'interpolation.

## 2.5 Conclusion

Au stationnaire, l'EF  $\mathbb{P}_1$  non stabilisé appliqué au problème linéaire testé atteint les ordres attendus:  $\mathcal{O}(h)$  en  $H^1$  et  $\mathcal{O}(h^2)$  en  $L^2$  pour une solution lisse. La régression fournit des pentes  $p_{H^1} \approx 1$  et  $p_{L^2} \approx 2$ , et des constantes  $C$  modérées. La borne  $\|u - u_h\|_{L^2} \leq M_h \|u - \mathcal{P}_h(u)\|_{L^2}$  est vérifiée numériquement avec  $M_h$  presque indépendant de  $h$ , ce qui corrobore l'optimalité de l'approximation  $\mathbb{P}_1$  en régime stationnaire sur ce cas test.