Un integratore robusto per l'equazione di Kramers

Gianluca Becuzzi

18 novembre 2023

1 Introduzione

Il seguente sistema di SDE per il moto di una particella sottoposto ad una forza stocastica:

$$\begin{aligned} dx(t) &= v(t)dt \\ dv(t) &= a(x,v,t)dt + \sigma(x,t)dW(t) \end{aligned}$$

è equivalente alla equazione di Fokker-Planck per la probabilità di transizione $p = p(x,v,t|x_0,v_0,t_0)$

$$\partial_t p + \partial_x(vp) + \partial_v(ap) = \frac{1}{2}\frac{\partial^2}{\partial v^2}(\sigma^2 p) \tag{1}$$

I primi due termini descrivono la advezione deterministica di p nello spazio delle fasi che si muove sotto un campo di velocità

$$\mathbf{u} = \big(v, a(x,v,t)\big) \tag{2}$$

che nei casi di interesse non è solenoidale, mentre il termine a secondo membro è il termine diffusivo dovuto al rumore. Dal momento che (1) si presenta come somma di due operatori differenziali rispetto ad una sola variabile, è naturale approcciare il problema con il metodo degli step frazionari [1]. Questo è equivalente a risolvere

$$\partial_t p + \partial_v(ap) = \sigma^2 \partial_v^2 p \qquad x = x_i \tag{3}$$

$$\partial_t p + \partial_x(vp) = 0 \qquad v = v_j \tag{4}$$

per ogni punto di una griglia regolare $x_i = i\Delta x$, $v_j = j\Delta v$.

Sia (1) che (3-4) rappresentano leggi di conservazione rispettivamente per l'intero dominio e per il singolo dominio 1D, quindi i metodi di integrazione impiegati sono della forma [2]

$$P_i^{n+1} = P_i^n - \frac{\Delta t}{\Delta}(F_{i\to i+1} - F_{i-1\to i}) \tag{5}$$

in cui P_i è il valore medio di p nella i-esima cella del dominio e F una funzione di flusso. Eq. (5) garantisce che

$$\sum_i P_i^{n+1} = \sum_i P_i^n - \frac{\Delta t}{\Delta}(F_{1,0} - F_{M-1,M}) \approx \sum_i P_i^n$$

se la funzione di flusso è circa nulla agli estremi del dominio, per cui la normalizzazione è conservata.

2 Schemi di integrazione

Dato che (4) e (3) sono formalmente la stessa equazione di advezione-diffusione, è sufficiente studiare il comportamento di (3) considerando poi (4) come il caso specifico $a = v$, $\sigma = 0$. Nel seguito si indica quindi con z la variabile di integrazione e Δz la spaziatura della griglia. Per motivi dimensionali le costanti che compaiono nel seguito sono il numero di Fourier per la diffusione e il numero di Courant per la advezione:

$$\eta = \frac{\sigma^2}{2}\frac{\Delta t}{\Delta z^2} \qquad \theta = \frac{\Delta t}{\Delta z} \tag{6}$$

2.1 Completamente implicito

Uno schema completamente implicito, proposto in [3], consiste nella seguente coppia di equazioni

$$\frac{p_i^{n+1} - p_i^n}{\Delta t} = -\frac{F_{i+1/2}^{n+1} - F_{i-1/2}^{n+1}}{\Delta z} \tag{7}$$

$$F_{i+1/2} = a_{i+1/2}\frac{p_i + p_{i+1}}{2} - \frac{\sigma^2}{2}\frac{p_{i+1} - p_i}{\Delta z} \tag{8}$$

che corrisponde ad imporre che nella singola cella la funzione sia costante

$$p(z) = p_i \quad z \in [z_i, z_{i+1}]$$
$$P_i = p_i$$

con un flusso dato da

$$F_{i\to i+1} = a_{i+\frac{1}{2}}P_i - \frac{\sigma^2}{2}\frac{P_{i+1}^{n+1} - P_i^{n+1}}{\Delta z} \tag{9}$$

In questo caso non c'è differenza tra uno schema di integrazione ai volumi finiti e alle differenze finite.

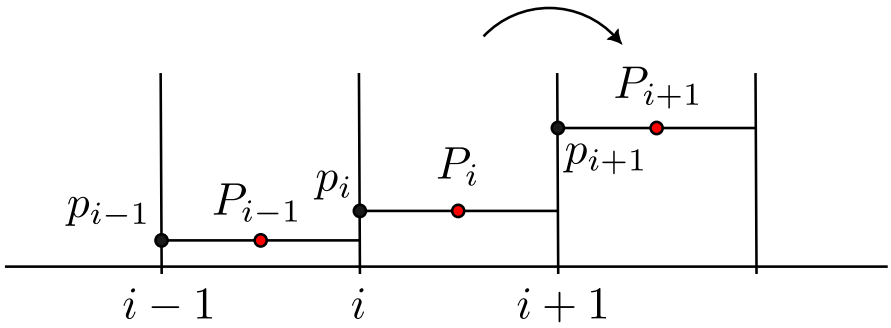

Figura 1: Caption

Il coefficiente di amplificazione lineare dello schema è dato da

$$g_{IMP} = \left(1 + 4\eta\sin^2\left(\frac{k\Delta z}{2}\right) + \frac{\partial a}{\partial z}\Delta t\cos^2\left(\frac{k\Delta z}{2}\right) - 4i\theta a\sin(k\Delta z)\right)^{-1} \tag{10}$$
$$\approx \left(1 + 4\eta\sin^2\left(\frac{k\Delta z}{2}\right) - 4i\theta a\sin(k\Delta z)\right)^{-1}$$

per cui stabile per ogni valore di Δz e Δt finchè vale

$$\gamma\Delta t \ll 4\eta \qquad \gamma = \max_{\Omega}(|\partial_z a|) \tag{11}$$

e la condizione di Courant per l'advezione

$$\frac{A\Delta t}{\Delta z} < 1 \qquad A = \max_{\Omega}(|a|) \tag{12}$$

2.2 Crank-Nicholson

Un'altra possibilità è quella di considerare anche i valori medi $\overline{P}_i$ delle celle [4]. Imponendo

$$p(z) = a + bz + cz^2 \qquad z \in [z_i, z_{i+1}] \tag{13}$$

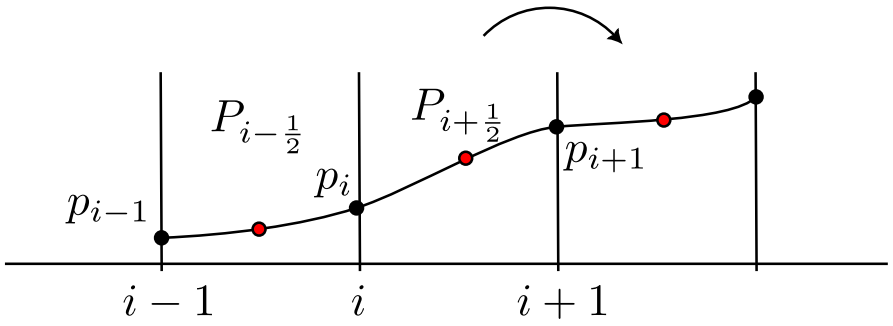

Figura 2: Caption

si ottiene per la derivata all'estremo destro della cella

$$\left.\frac{\partial p}{\partial z}\right|_{right,i} = \frac{4p_{i+1} + 2p_i - 6\overline{P}_i}{\Delta x} \tag{14}$$

Usando un flusso

$$F_{i\to i+1} = a_{i+1}p_{i+1} - \frac{\sigma^2}{2}\left.\frac{\partial p}{\partial z}\right|_{right,i} \tag{15}$$

3 AAA

I tre termini di (1) sono operatori differenziali in una sola variabile. Questi possono essere considerati come tre termini indipendenti oppure raggruppati in base alla variabile di interesse

$$\partial_t p = [\mathcal{L}_1 + \mathcal{L}_2 + \mathcal{L}_3]\, p = [\mathcal{L}_x + \mathcal{L}_v]\, p \tag{16}$$

per cui la soluzione può essere approssimata applicando un operator split a due o tre step. Come suggerito in [3] (vedi app. C), si approssima il termine di drift in v con

$$\partial_v(ap) \approx \frac{1}{2\Delta v}\left(a_{i,j+\frac{1}{2}}p_{i,j+1} - a_{i,j-\frac{1}{2}}p_{i,j-1}\right) + \frac{1}{2\Delta v}\left(a_{i,j+\frac{1}{2}} - a_{i,j-\frac{1}{2}}\right)p_{ij} \tag{17}$$

In seguito il problema algebrico complessivo:

$$\begin{aligned}\frac{p_{ij}^{n+1} - p_{ij}^{n}}{\Delta t} = &-v_j\frac{p_{i+1,j}^{n+1} - p_{i-1,j}^{n+1}}{\Delta x} + \\ &+ \frac{\sigma^2}{2}\left(\frac{p_{i,j+1}^{n+1} - 2p_{i,j}^{n+1} + p_{i,j-1}^{n+1}}{\Delta v^2}\right) - \partial_v(ap)\end{aligned}$$

viene approssimato dai due sotto-problemi tridiagonali

$$\frac{p_{ij}^{*} - p_{ij}^{n}}{\Delta t} = -\left[\partial_v(ap)\right]^{*} + \frac{\sigma^2}{2}\left(\frac{p_{i,i+1}^{*} - 2p_{i,j}^{*} + p_{i,j-1}^{*}}{\Delta v^2}\right) \tag{18}$$

$$\frac{p_{ij}^{n+1} - p_{ij}^{*}}{\Delta t} = -v_j\frac{p_{i+1,j}^{n+1} - p_{i-1,j}^{n+1}}{\Delta x} \tag{19}$$

ovvero, per ogni vettore riga $\mathbf{r}$ e per ogni vettore colonna $\mathbf{c}$ della matrice p_{ij}^{n} si hanno i tre sistemi:

$$\mathbf{V}\mathbf{r}^{n+\frac{1}{3}} = \mathbf{r}^{n} \tag{20}$$

$$\mathbf{X}\mathbf{c}^{n+\frac{2}{3}} = \mathbf{c}^{n+\frac{1}{3}} \tag{21}$$

$$\mathbf{D}\mathbf{r}^{n+1} = \mathbf{r}^{n+\frac{2}{3}} \tag{22}$$

$$\begin{aligned}\mathbf{D}_{jm} = &-\eta\delta_{j+1,m} \\ &+ \left(1+2\eta\right)\delta_{jm} \\ &-\eta\delta_{j-1,m}\end{aligned} \qquad (23)$$

$$\begin{aligned}\mathbf{X}_{im} = &\left(\alpha v_j\right)\delta_{i+1,m} \\ &+ \delta_{im} \\ &+ \left(-\alpha v_j\right)\delta_{i-1,m}\end{aligned} \qquad (24)$$

$$\begin{aligned}\mathbf{V}_{jm} = &\left(\theta a_{j+\frac{1}{2}}\right)\delta_{j+1,m} \\ &+ \left(1+\theta\left[a_{j+\frac{1}{2}} - a_{j-\frac{1}{2}}\right]\right)\delta_{jm} \\ &+ \left(-\theta a_{j-\frac{1}{2}}\right)\delta_{j-1,m}\end{aligned} \qquad (25)$$

in cui $\mathbf{V}$, $\mathbf{X}$ e $\mathbf{D}$ sono matrici tridiagonali date da

$$\eta = \frac{\sigma^2}{2}\frac{\Delta t}{\Delta v^2} \qquad \theta = \frac{1}{2}\frac{\Delta t}{\Delta v} \qquad \alpha = \frac{1}{2}\frac{\Delta t}{\Delta x} \qquad (26)$$

che sono i coefficienti legati rispettivamente a diffusione, drift in v e drift in x.

In (20-22) è implicito il fatto che il sistema vada risolto prima per ogni riga, poi per ogni colonna, e infine di nuovo per ogni riga. Inoltre in eq. 25 si è posto per brevità $a_j \equiv a^n_{ij}$, dando per scontato che per ogni riga i la matrice $\mathbf{V}^{(i)}$ vada ricalcolata poichè cambiano i valori di a.

Se a è lineare in x e v la diagonale di (25) non dipende dagli indici i e j. Evitando la costruzione di questa si riduce il tempo di esecuzione dell'algoritmo di circa il 7% (vedi appendice A).

Nel caso il rumore dipenda dal tempo e dalla posizione lo schema di integrazione è identico al precedente con $\eta = \eta(x,t)$.

3.1 Stabilità

Il coefficiente di amplificazione di (**??**) è

$$g_B = (1 + 2i\cdot\alpha v\sin(k\Delta x))^{-1} \qquad (27)$$

per cui incondizionatamente stabile, mentre quello di (**??**) è

$$g_A = \left(1 + 4\eta\sin^2\left(\frac{k\Delta v}{2}\right) + \frac{\partial a}{\partial v}\Delta t\cos^2\left(\frac{k\Delta v}{2}\right) - 2i\theta a\sin(k\Delta v)\right)^{-1} \qquad (28)$$

per cui

$$\begin{aligned} g_A &\approx 1 - \frac{\partial a}{\partial v}\Delta t & k &\to 0 \\ g_A &\approx 1 - 4\eta & k &\to \frac{\pi}{\Delta v}\end{aligned}$$

quindi le regioni in cui $p \approx const.$ vengono giustamente amplificate dal termine $\partial_v(ap) \approx p\partial_v a$ mentre i modi ad alta frequenza spaziale vengono attenuati. Si noti che se fosse stata fatta l'approssimazione:

$$\partial_v(ap) = p\partial_v a + a\partial_v p \approx \frac{\partial a}{\partial v}p_{ij} + a_{ij}\frac{p_{i,j+1} - p_{i,j-1}}{2\Delta v}$$

il termine di drift in v avrebbe influenzato anche l'attenuazione dei modi ad alta frequenza:

$$g_A \approx 1 - 4\eta - \frac{\partial a}{\partial v}\Delta t \qquad k \to \frac{\pi}{\Delta v}$$

3.2 Normalizzazione e condizioni al bordo

La b.c. naturale per il problema è che la funzione decada in maniera sufficientemente rapida ai bordi e che la normalizzazione sia conservata. Se la griglia è sufficientemente fine:

$$N(t) = \int_\Omega p(x, v, t) \approx \sum_{ij} p^n_{ij} \Delta x \Delta v \tag{29}$$

Una condizione sufficiente alla conservazione della norma è richiedere che sia la matrice inversa di (**??**) che quella di (**??**) conservino la somma degli elementi rispettivamente dei vettori riga e dei vettori colonna, ovvero richiedere che $\mathbf{A}^{-1}$ e $\mathbf{B}^{-1}$ siano matrici stocastiche sinistre, e quindi che lo siano anche $\mathbf{A}$ e $\mathbf{B}$. La matrice $\mathbf{B}$ verifica questa condizione, mentre per $\mathbf{A}$:

$$\mathbf{A} = \mathbf{I} + \theta \begin{pmatrix} \ddots & a_{m-\frac{1}{2}} & \\ -a_{m-\frac{1}{2}} & \left[a_{m+\frac{1}{2}} - a_{m-\frac{1}{2}}\right] & a_{m+\frac{1}{2}} \\ & -a_{m+\frac{1}{2}} & \ddots \end{pmatrix} \tag{30}$$

quindi la somma della m-esima colonna è uguale a

$$\sum_k A_{km} = 1 + \theta(a_{m-1} - a_{m+1}) - \frac{1}{4}\Delta t \left((\partial_v a)_{m-1} + 2(\partial_v a)_{m+1} + (\partial_v a)_m\right) \approx 1 + \frac{1}{4}\Delta t \Delta v^2 \left.\frac{\partial^2 a}{\partial v^2}\right|_{v_m}$$

dunque la normalizzazione non è esattamente conservata se la funzione $a(x, v, t)$ è più che di primo grado in v.

Per quanto riguarda l'errore di arrotondamento, in Fig. **??** viene mostrato che (almeno per tempi piccoli)

$$\begin{aligned} N(t) &\approx \exp(mt) \\ |m| &\approx 10^{-14} \end{aligned} \tag{31}$$

Sono state testate anche condizioni al bordo assorbenti e riflettenti, senza rimarcabili differenze in termini di accuratezza e stabilità.

3.3 Add-ons

È possibile migliorare l'accuratezza del metodo aggiungendo una correzione di tipo Crank-Nicholson a (**??**) con $\eta \leftarrow \frac{\eta}{2}$

$$\begin{aligned} \mathbf{A}\mathbf{r}^* &= \mathbf{D}\mathbf{r}^n \\ \mathbf{D}_{jm} &= \eta\left(\delta_{j+1,m} - 2\delta_i + \delta_{j-1,m}\right) \end{aligned}$$

In questo caso si ottiene la correzione a (28):

$$|g'_A| = |g_A| \cdot \left(1 - 4\eta \sin^2\left(\frac{k\Delta v}{2}\right)\right) < |g_A|$$

quindi la stabilità non viene intaccata.

4 Accuratezza

Le soluzioni di (1) sono note analiticamente solo per l'oscillatore armonico smorzato (appendice B) in cui

$$a(x, v, t) = -\omega^2 x - \gamma v \tag{32}$$

Come estimatori della precisione della soluzione numerica vengono calcolati:

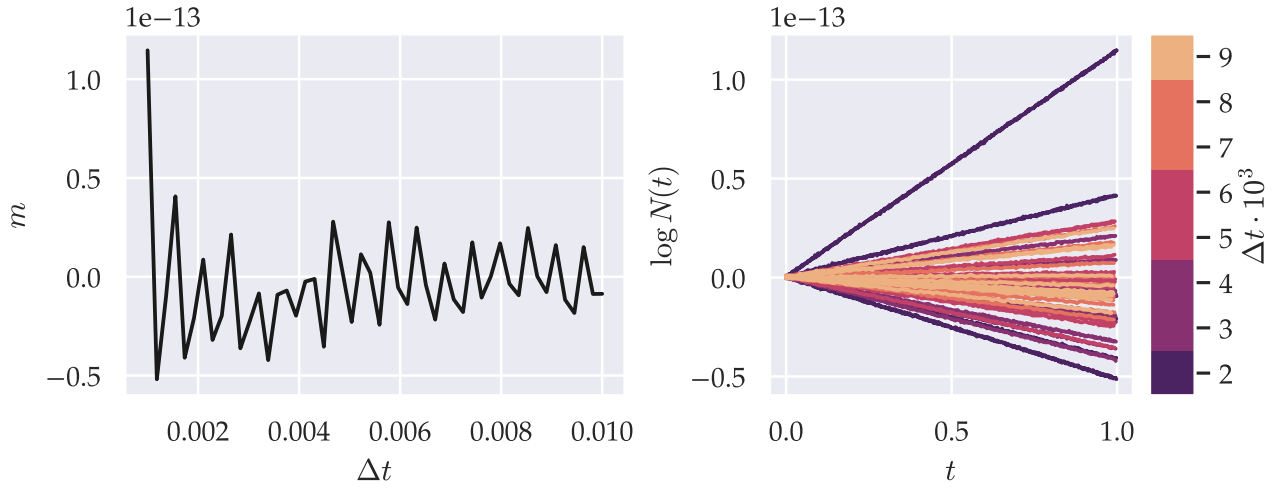

Figura 3: Norma in funzione del tempo per l'integratore a singolo split di eqs. (**??**) e (**??**). (Destra) La norma si discosta dal valore 1 esponenzialmente (sia crescente che descrescente) con un tempo caratteristico molto lungo. (Sinistra) L'esponente della norma (eq. 31) cambia segno in maniera irregolare ed è maggiore per timesteps piccoli, quindi è presumibilmente dovuto all'errore di arrotondamento.

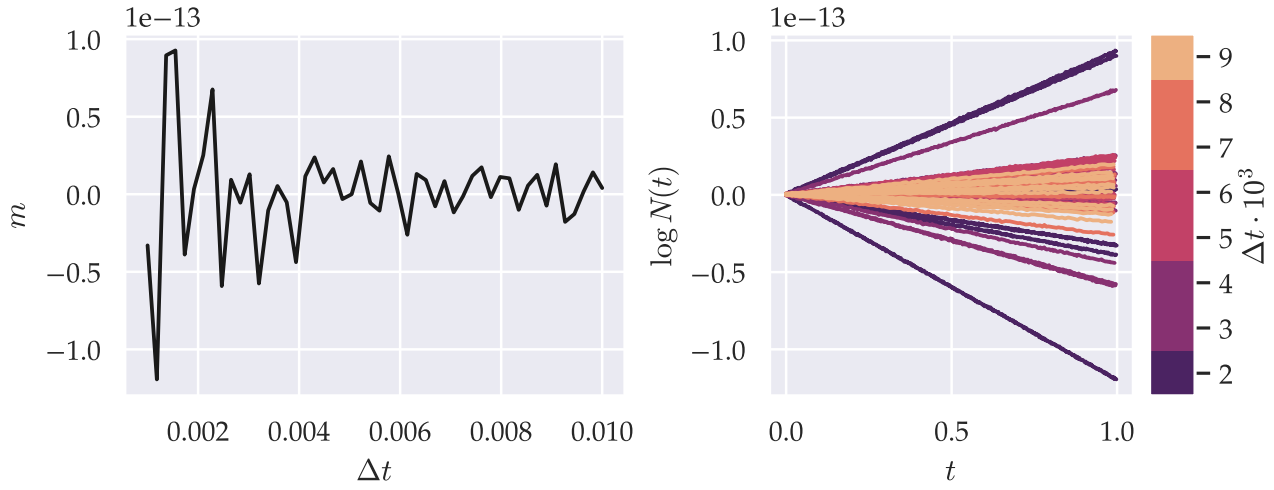

Figura 4: Norma in funzione del tempo per l'integratore a doppio split con correzione Crank-Nicholson per il secondo (x-drift) e terzo (diffusione) operatore. La figura è simile a Fig. 3.

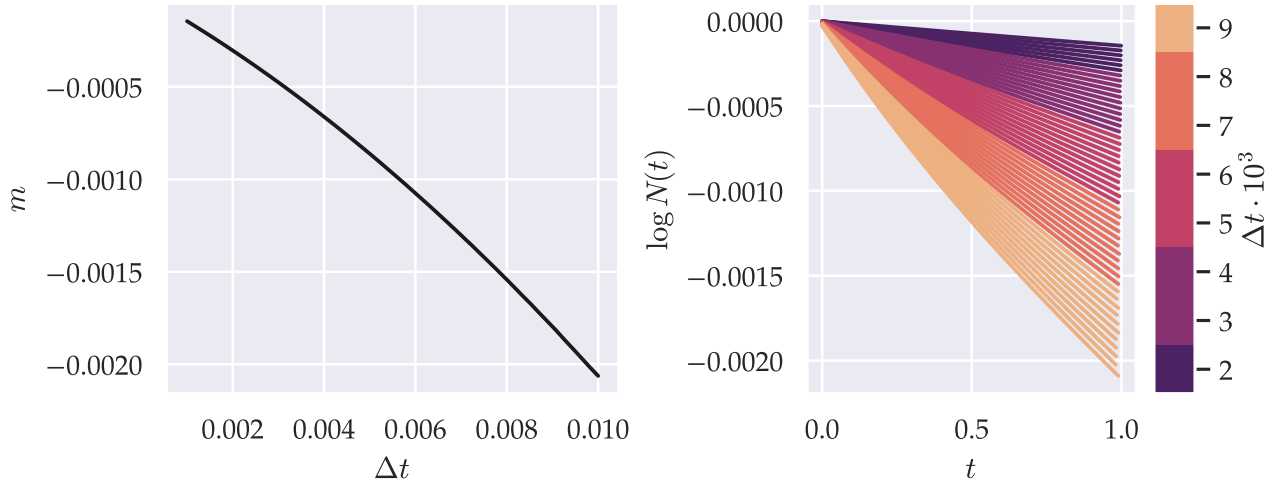

Figura 5: Norma in funzione del tempo per l'integratore a 2 split con correzione Crank-Nicholson. La figura è simile a 3

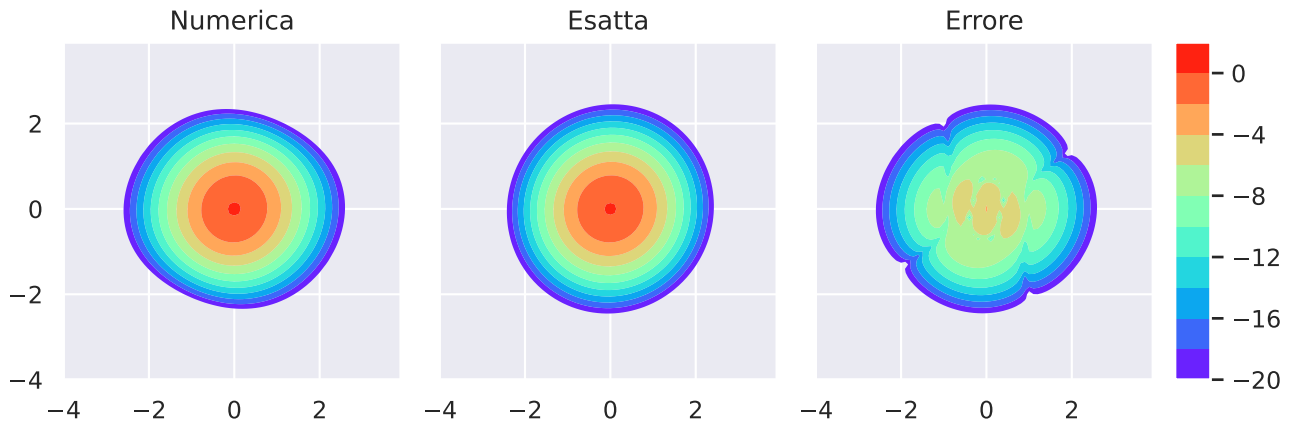

Figura 6: Soluzione numerica, esatta e valore assoluto dell'errore dopo 800 timesteps (scala logaritmica).

$$\|e\|_{RMS} = \sqrt{\frac{1}{NM}\sum_{ij}(p_{ij}^{num} - p_{ij}^{exact})^2}$$
$$\|e\|_{SUP} = \max_{ij}|p_{ij}^{num} - p_{ij}^{exact}|$$

Il confronto presentato nasconde il fatto che i due problemi non sono esattamente equivalenti. Il RMSE è infatti indicativo della precisione del metodo soltanto nel limite in cui i due problemi sono equivalenti, ovvero quando il lato del quadrato $L \to \infty$. Inoltre se il sup dell'errore commesso nel valutare p_{ij} è circa costante, il RMSE può essere reso piccolo a piacere aumentando L (Fig. 8).

Infine, più è piccolo il supporto della condizione iniziale $p_0(x, v)$ più l'errore dovuto alla discretizzazione del reticolo è grande (Fig. 7).

Come condizione iniziale è stata scelta la soluzione esatta per $t = 0.95$. Nel seguito vale $\Delta t = \pi/1000$, $\Delta x = \Delta v = 0.1$.

4.1 Stato stazionario

Per $\gamma > 2\omega^2$ il sistema raggiunge rapidamente l'equilibrio senza oscillare. La distribuzione viene fatta evolvere numericamente per 100 passi di integrazione fino a stabilizzare la quinta cifra significativa dei momenti secondi[1] ($t \approx 10$). A stazionarietà questi sono:

$$\langle x \rangle = \langle v \rangle = \langle xv \rangle = 0$$
$$\omega^2 \langle x^2 \rangle = \langle v^2 \rangle = \frac{\sigma^2}{2\gamma} \tag{33}$$

I valori di momenti di x e v calcolati a stazionarietà sono mostrati in Tab.1. Per la distanza tra le due distribuzioni si ottiene invece $\|e\| = 1.6 \cdot 10^{-4}$.

4.2 Dipendenza dal tempo

I risultati per la media e la varianza di x sono mostrati in Fig. 9 e Fig. 10

A Codice e performance

Il codice viene implementato in Cython e profilato tramite il pacchetto python `line_profiler`. A sinistra di ogni riga, dove necessario, e' indicato il costo temporale percentuale della singola linea (dove non riportato e' minore del 1%).

[1] I momenti sono stati calcolati con un integrazione trapezoidale [5]

	Numerico	Esatto	Stazionarietà
$E[xx]$	0.15109	0.15238	0.15238
$E[vv]$	0.15010	0.15238	0.15238
$E[xv]$	$3.2612 \cdot 10^{-8}$	$3.2984 \cdot 10^{-8}$	0

Tabella 1: Confronto tra i momenti secondi di $p(x, v)$ per $\omega = 1$, $\gamma = 2.1$ e $\sigma = 0.8$. Il valore esatto ed il valore teorico (eq. 33) a stazionarietà corrispondono entro la precisione scelta. L'errore della stima numerica è circa lo 0.5%.

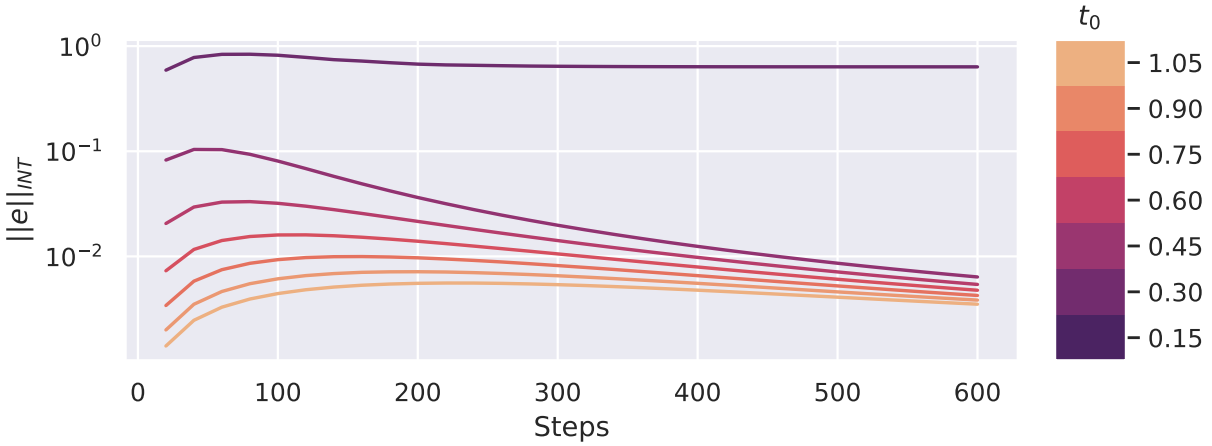

Figura 7: Dipendenza dell'errore dalle condizioni iniziali. Più la condizione iniziale ha supporto piccolo ($t_0 \to 0$) più la discretizzazione del reticolo impatta l'errore. Se questo non è troppo grande la soluzione converge comunque alla soluzione stazionaria.

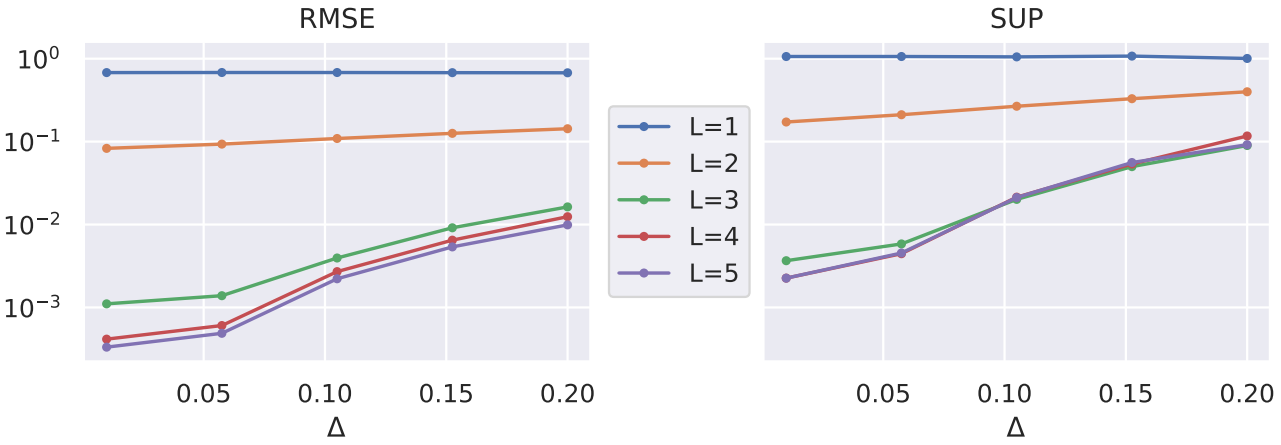

Figura 8: Dipendenza dell'errore dalla dimensione del dominio.

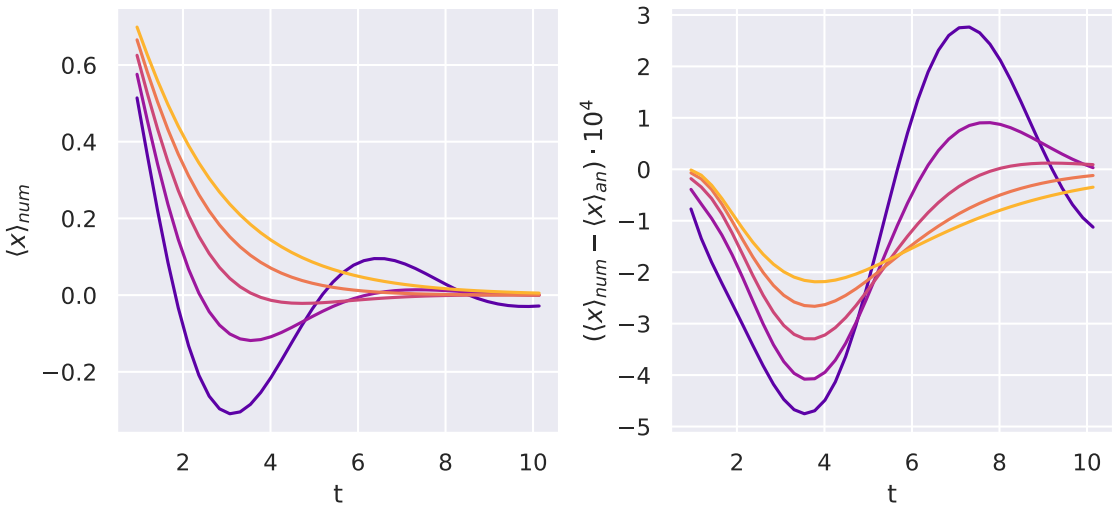

Figura 9: Valore di aspettazione di x nel tempo ed errore rispetto alla soluzione esatta per vari valori di γ con $\omega = 1$. Da notare a tempi piccoli il valore di $\langle x \rangle$ viene sottostimato, come se la soluzione numerica evolvesse più rapidamente di quella analitica.

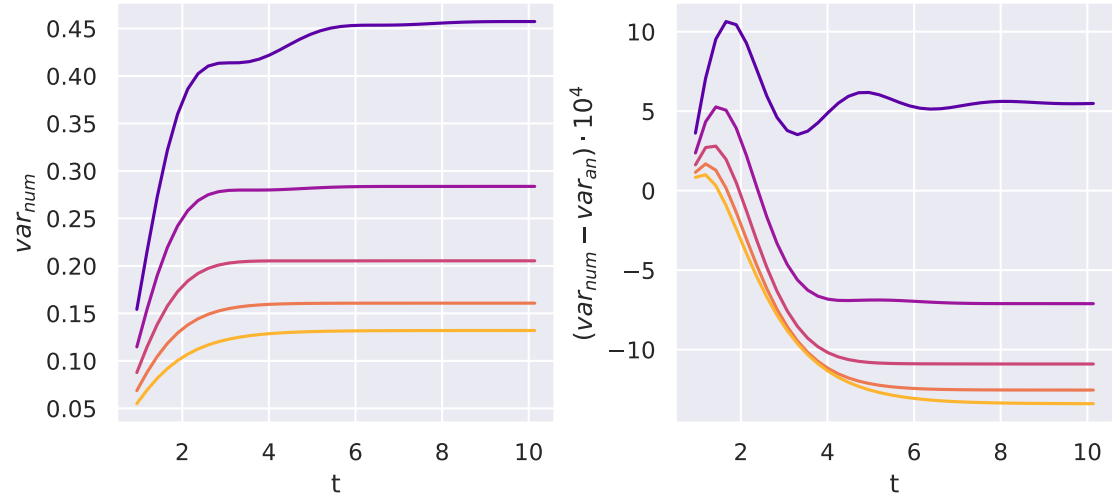

Figura 10: Caption

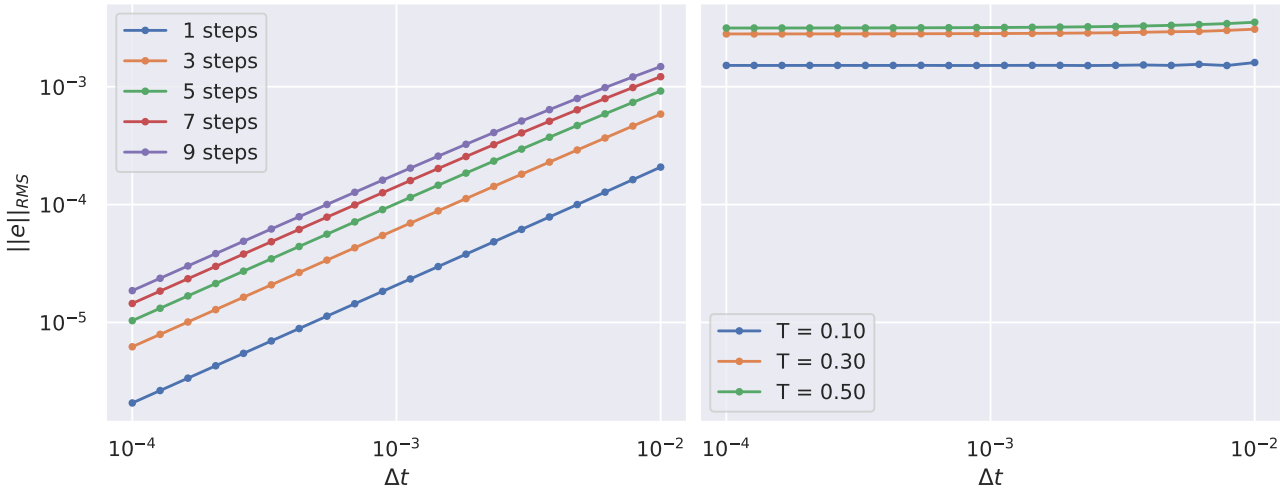

Figura 11: Errore di troncamento locale (sinistra) e globale (destra).

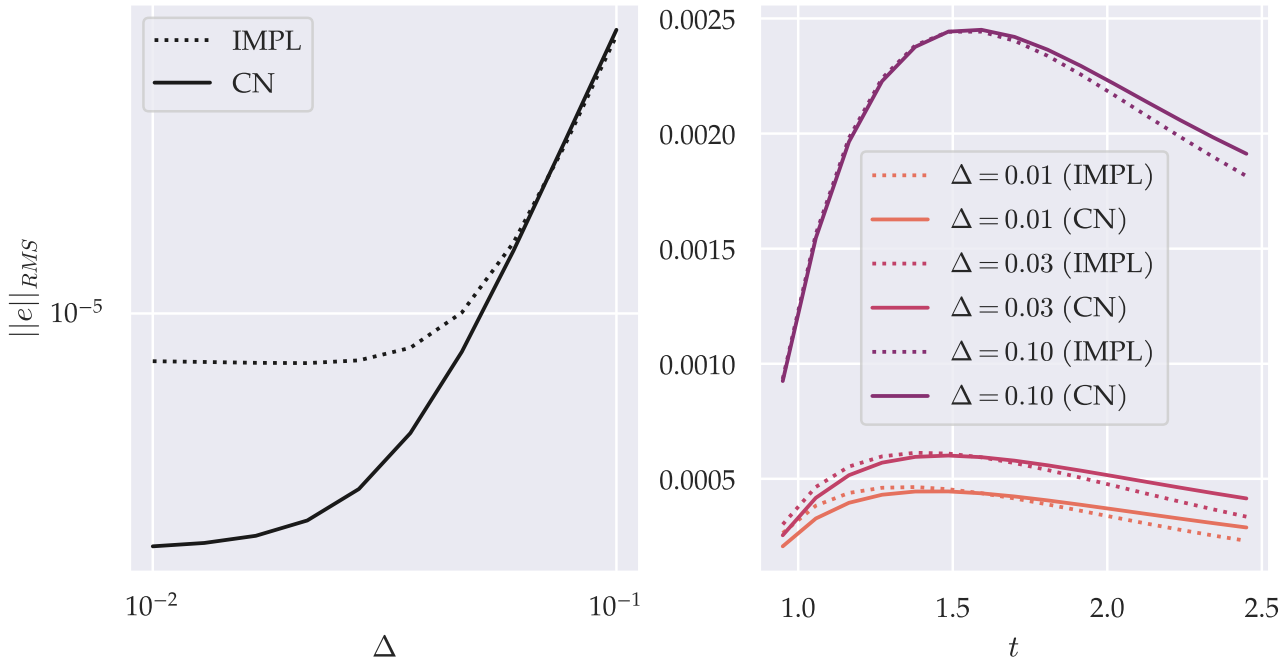

Figura 12: Caption

Inaspettatamente il tempo di preparazione delle matrici e' superiore al tempo necessario per risolverle: la risoluzione di entrambi i sistemi tridiagonali occupa il 14% del tempo mentre la preparazione della prima matrice il 55%, la seconda il 9% e il riassestamento dei dati l' 8%.

Trascurando gli overhead di inizializzazione, il singolo step di update ha un costo temporale dato da:

$$\tau_{step} \approx \beta N^2 + \gamma N$$

dove β e' una costante dipendente dalla macchina mentre N è il numero di punti su ciascun asse del dominio, supposto quadrato.

Sulla macchina testata (Intel i5-1135G7) nel caso σ costante e $\partial_v a$ costante:

$$\beta \approx 0.13\mu s$$
$$\gamma \approx 4\mu s$$

per cui 1000 step di update su un reticolo 80x80 impiegano circa 1.5 secondi, mentre per il metodo usato in [3]

$$\beta \approx 0.2\mu s$$
$$\gamma \approx 4.4\mu s$$

Per quanto riguarda il caso $\sigma = \sigma(x,t)$ e $\partial_v a \neq const.$

$$\beta \approx 0.21\mu s$$
$$\gamma \approx 4.6\mu s$$

.

```
    def generic_3_step( double [:,:] p0,
                        physical_params,
                        integration_params,
                        save_norm = False,
                        save_current=False
                        ):
      ## Time
      cdef double dt   = integration_params['dt']
      cdef unsigned int n_steps = integration_params['n_steps']
      cdef double t0    = physical_params.get('t0', 0.0)

      ## Space
      cdef double Lx, Lv,dx,dv
      Lx, Lv,dx,dv = map(integration_params.get, ["Lx", "Lv", "dx", "dv"])
      cdef unsigned int N = int(Lx/dx), M = int(Lv/dv)
      cdef double [:] x = np.arange(-int(N)//2, int(N)//2)*dx

      cdef double [:] v = np.arange(-int(M)//2, int(M)//2)*dv

      ## Add-ons
      cdef bool [:] CN_ized_steps = integration_params.get('CN', np.array([False, False,
    False]))
      cdef bool ADI = integration_params.get("ADI", False)

      cdef unsigned int time_index = 0, i = 0, j = 0

      cdef double [:,:] p = p0.copy(), p_star = p0.copy(), p_dagger = p0.copy()
      cdef double [:] norm = np.zeros(n_steps)
      cdef double [:] amplification_average = np.zeros(3)

      if ADI:
        print("Set to ADI mode")
        dt = dt/3.0

```

```
34        cdef double theta = 0.5 * dt/dv
35        cdef double alpha = 0.5 * dt/dx
36        cdef double eta   = 0.5 * dt/dv**2
37        cdef double time  = t0
38
39        # Halves the bros in case Crank-Nicholson is chosen
40        if CN_ized_steps[0]:
41          print("V-drift Crank-Nicholson-ized")
42          theta = 0.5 * theta
43
44        if CN_ized_steps[1]:
45          print("X-drift Crank-Nicholson-ized")
46          alpha = 0.5 * alpha
47
48        if CN_ized_steps[2]:
49          print("Diffusion Crank-Nicholson-ized")
50          eta = 0.5 * eta
51
52        # Declarations of the diagonals
53        cdef double [:] lower_1, diagonal_1, upper_1, b_1
54        cdef double [:] lower_2, diagonal_2, upper_2, b_2
55        cdef double [:] lower_3, diagonal_3, upper_3, b_3
56
57        diagonal_1, lower_1, upper_1, b_1 = np.ones(M), np.ones(M), np.ones(M), np.ones(M)
58        diagonal_2, lower_2, upper_2, b_2 = np.ones(N), np.ones(N), np.ones(N), np.ones(N)
59        diagonal_3, lower_3, upper_3, b_3 = np.ones(M), np.ones(M), np.ones(M), np.ones(M)
60
61        # Working variables
62        cdef double a_plus, a_minus, s
63
64        cdef dict currents = dict(top=np.zeros(n_steps),
65                                  bottom=np.zeros(n_steps),
66                                  left=np.zeros(n_steps),
67                                  right=np.zeros(n_steps))
68
69        for time_index in range(n_steps):
70          time = t0 + time_index*dt
71          ##################################### First evolution: v-drift
    ########################################
72          for i in range(N):
73
74            # Constant part of coefficient vector does not depend on j
75 [ 2%]      b_1 =  p[:, i].copy()
76
77            # Diffusion coefficient does not depend on j
78            s = eta * sigma_squared(x[i], time, physical_params)
79
80            for j in range(M):
81              # Note tha theta is absorbed in working variable
82 [ 5%]        a_plus  =  theta * a(x[i],v[j] + 0.5*dv, time, physical_params)
83 [ 5%]        a_minus =  theta * a(x[i],v[j] - 0.5*dv, time, physical_params)
84
85 [ 2%]        diagonal_1[j] = 1 +  a_plus - a_minus
86 [ 2%]        upper_1[j]    =   + a_plus
87              # Since lower has an offset of one
88              # (write the matrix down and you'll see)
89 [ 2%]        lower_1[j]    =   -  a_plus
90
91 [ 2%]        if ADI:
92                # ADI-sytle
93                if i != 0 and i != N-1:
94                  # X-drift
95                  b_1[j] += - alpha * v[j] * (p[j, i+1] - p[j, i-1])
96
97                if j != 0 and j != M-1:
98                  # Diffusion
```

```
                b_1[j] +=   (   s)* p[j+1,i]
                b_1[j] +=   (-2*s)* p[j,i]
                b_1[j] +=   (   s)* p[j-1, i]

[ 3%]         if CN_ized_steps[0]:
                if j > 0 and j < M-1:

                  # Drift
                  b_1[j] += - a_plus*p[j+1, i]
                  b_1[j] += (a_minus - a_plus)*p[j, i]
                  b_1[j] += a_minus*p[j-1, i]

            # Solves the tridiagonal system for the column
[ 8%]       p_star[:, i] = tridiag(lower_1, diagonal_1, upper_1, b_1)
[ 4%]       amplification_average[0] += np.sum(p_star[:, i])/np.sum(p[:,i])/N/n_steps
            # print(f"Sum of V-rows\n{np.sum(get_tridiag(lower_1, diagonal_1, upper_1),
    axis=1)}")
            # print(f"Sum of V-cols\n{np.sum(get_tridiag(lower_1, diagonal_1, upper_1),
    axis=0)}")

          ################################### Second evolution: x-drift
    ########################################
          for j in range(M):

            # Does not depend on i
[ 1%]       b_2 = p_star[j, :].copy()

            for i in range(N):
[ 2%]         lower_2[i] = - alpha * v[j]
[ 2%]         upper_2[i] =   alpha * v[j]

[ 2%]         b_2[i] = p_star[j, i]

[ 2%]         if ADI:
                # ADI-style
                a_plus  = theta * a(x[i], v[j] + 0.5*dv, time, physical_params)
                a_minus = theta * a(x[i], v[j] - 0.5*dv, time, physical_params)
                s = eta * sigma_squared(x[i], time, physical_params)
                if j != 0 and j != M-1:

                  # V-Drift
                  b_2[j] += (- a_plus)*p_star[j+1, i]
                  b_2[j] += ( a_minus - a_plus)*p_star[j, i]
                  b_2[j] += (+ a_minus)*p_star[j-1, i]

                  # Diffusion
                  b_2[i] +=   (   s)* p_star[j+1,i]
                  b_2[i] +=   (-2*s)* p_star[j,i]
                  b_2[i] +=   (   s)* p_star[j-1, i]

[ 2%]         if CN_ized_steps[1]:
                if i != 0 and i != N-1:
                  b_2[i] += alpha * v[j] * (p_star[j, i-1] - p_star[j, i+1])

            # Solves the tridiagonal system for the row
[ 6%]       p_dagger[j, :] =  tridiag(lower_2, diagonal_2, upper_2, b_2)
[ 4%]       amplification_average[1] += np.sum(p_dagger[j, :])/np.sum(p_star[j,:])/M/
    n_steps

          ################################### Third evolution: diffusion
    ########################################
          for i in range(N):
            s = eta * sigma_squared(x[i], time, physical_params)
[ 2%]       b_3 = p_dagger[:, i].copy()
            for j in range(M):

```

```
[ 2%]         diagonal_3[j] = 1 + 2 * s
[ 2%]         upper_3[j]    = - s
[ 2%]         lower_3[j]    = - s

[ 2%]         if ADI:
                a_plus  = theta * a(x[i], v[j] + 0.5*dv, time, physical_params)
                a_minus = theta * a(x[i], v[j] - 0.5*dv, time, physical_params)
                # V-drift
                if j!=0 and j!= M-1:
                  b_3[j] += (- theta * a(x[i], v[j] + 0.5*dv, time, physical_params))*
    p_dagger[j+1, i]
                  b_3[j] += (- theta *( a(x[i],v[j] + 0.5*dv, time, physical_params) - a(x
    [i],v[j] - 0.5*dv, time, physical_params)) )*p_dagger[j, i]
                  b_3[j] += (+ theta * a(x[i], v[j] - 0.5*dv, time, physical_params))*
    p_dagger[j-1, i]
                if i != 0 and i != N-1:
                  # X-drift
                  b_3[j] += alpha * v[j] * (p_dagger[j, i-1] - p_dagger[j, i+1])

[ 2%]         if CN_ized_steps[2]:
                if j > 0 and j < M-1:
                  b_3[j] +=   (   s)* p_dagger[j+1,i]
                  b_3[j] +=   (-2*s)* p_dagger[j,i]
                  b_3[j] +=   (   s)* p_dagger[j-1, i]

            # ## Raw conservation of norm
            # diagonal_v[0] = 1 - lower_v[0]
            # diagonal_v[M-1] = 1- upper_v[M-2]

            # Solves the tridiagonal system for the column
[ 8%]       p[:, i] = tridiag(lower_3, diagonal_3, upper_3, b_3)
[ 4%]       amplification_average[2] += np.sum(p[:, i])/np.sum(p_dagger[:,i])/N/n_steps

          ###################################### UTILS
    ######################################
          # Takes trace of normalization
          if save_norm:
            norm[time_index] = quad_int(p, integration_params)

          if save_current:
            # Integral in v
            for j in range(M):
              currents['right'][time_index] += v[j]*p[j, N-2]*dv
              currents['left'][time_index]  -= v[j]*p[j, 1]*dv

            # Integral in x
            for i in range(N):
              currents['top'][time_index] += a(x[i], v[M-2],  t0 + time_index*dt,
    physical_params)*p[M-2, i]*dx
              currents['top'][time_index] -= 0.5*physical_params['sigma_squared']*( (p[M
    -1,i] - p[M-2,i])/dv )*dx

              currents['bottom'][time_index] -= a(x[i], v[2],  t0 + time_index*dt,
    physical_params)*p[2, i]*dx
              currents['bottom'][time_index] += 0.5*physical_params['sigma_squared']**2*(
    (p[1,i] - p[0,i])/dv )*dx
        print(f"Amplification averages = {np.array(amplification_average)}")
        return p, norm, currents
```

B Soluzione analitica dell'eq. di Kramers a coefficienti lineari

Il seguito è tratto da [6], [7] e [8], in cui il problema viene risolto in uno spazio infinito e non in un quadrato.

L'equazione

$$\partial_t P = -v\partial_x P + \partial_v((\omega^2 x + \gamma v)P) + \frac{\sigma^2}{2}\partial_v^2 P$$

diventa per la funzione caratteristica $\phi(\mathbf{k}, t) = \langle \exp i\mathbf{k}\cdot\mathbf{x}\rangle$

$$\partial_t \phi(\mathbf{k}, t) = k_x \partial_{k_v}\phi(\mathbf{k}, t) - \omega^2 k_v \partial_{k_x}\phi(\mathbf{k}, t) - \gamma k_v \partial_{k_v}\phi(\mathbf{k}, t) + \frac{\sigma^2}{2}k_v^2\phi(\mathbf{k}, t)$$

per cui si cerca una soluzione della forma

$$\phi(\mathbf{k}, t) = \exp(-i\mathbf{k}\boldsymbol{\mu}(t))\exp(-\frac{1}{2}\mathbf{k}\boldsymbol{\sigma}(t)\mathbf{k})$$

per cui il vettore e la matrice ausiliari $\boldsymbol{\mu}(t)$ e $\boldsymbol{\sigma}(t)$ rispettano

$$\frac{d\boldsymbol{\mu}}{dt} + \boldsymbol{\Gamma}\boldsymbol{\mu} = 0 \tag{34}$$

$$\frac{d\boldsymbol{\sigma}}{dt} + \boldsymbol{\Gamma}\boldsymbol{\sigma} + (\boldsymbol{\Gamma}\boldsymbol{\sigma})^\dagger = 2\mathbf{D} \tag{35}$$

dove la matrice $\boldsymbol{\gamma}$ e la matrice $\mathbf{D}$ sono

$$\boldsymbol{\Gamma} = \begin{pmatrix} 0 & -1 \\ \omega^2 & \gamma \end{pmatrix} \tag{36}$$

$$\mathbf{D} = \begin{pmatrix} 0 & 0 \\ 0 & \sigma^2 \end{pmatrix} \tag{37}$$

Cioe' $\boldsymbol{\mu}$ e $\boldsymbol{\sigma}$ sono e soluzioni dell' equazione di Langevin omogenea associata al sistema e rappresentano i momenti primi e i momenti secondi. In particolare, le eqs. (34) e (35) hanno soluzioni

$$\boldsymbol{\mu}(t) = \mathbf{G}(t)\mathbf{x}_0 \tag{38}$$

$$\boldsymbol{\sigma}(t) = \int_0^t \mathbf{G}(\tau)\mathbf{D}\mathbf{G}^\dagger(\tau)d\tau \tag{39}$$

$$\mathbf{G}(t) = \exp(-\boldsymbol{\Gamma}t) \tag{40}$$

Infine, applicando la trasformata inversa si ottiene per la probabilità di transizione:

$$P(\mathbf{x}, t|\mathbf{x}_0, t_0 = 0) = \frac{1}{2\pi}|\boldsymbol{\sigma}(t)|^{-\frac{1}{2}}\exp\left[-\frac{1}{2}(\mathbf{x} - \mathbf{G}(t)\mathbf{x}_0)\boldsymbol{\sigma}^{-1}(t)(\mathbf{x} - \mathbf{G}(t)\mathbf{x}_0)\right] \tag{41}$$

C Refuso di eq. 7

Nell'equazione 7 di [3] è presente un refuso che porta alla risoluzione di un' altra equazione. In particolare lo schema di integrazione per l'operatore in v viene definito come:

$$\frac{p_{ij}^* - p_{ij}^n}{\Delta t} = F_{i,j+1} - F_{i,j-1}$$
$$F_{i,j\pm1} = \pm\frac{\sigma^2}{2\Delta v}\left(\frac{p_{i,j\pm1} - p_{ij}}{\Delta v}\right) + a(x, v \pm \Delta v)\left(\frac{p_{i,j\pm1} + p_{ij}}{2\Delta v}\right)$$

in cui la densità di probabilita e implicitamente calcolata al tempo intermedio. La differenza, se sviluppata, porta a:

$$F_{i,j+1} - F_{i,j-1} = \frac{\sigma^2}{2\Delta v^2}\left[p_{i,j+1} - 2p_{ij} + p_{i,j-1}\right] + a(x, v + \Delta v)\left(\frac{p_{i,j+1} + p_{ij}}{2\Delta v}\right) - a(x, v - \Delta v)\left(\frac{p_{i,j-1} + p_{ij}}{2\Delta v}\right)$$

in cui la prima parte è l'usuale termine diffusivo, mentre la parte riguardante la funzione $a(x,v)$ è circa uguale a

$$\begin{aligned} & a(x,v)\left[\frac{p_{i,j+1}-p_{i,j-1}}{2\Delta v}\right]+\frac{\partial a}{\partial v}\left[\frac{p_{i,j+1}+p_{i,j-1}}{2}+p_{ij}\right] \\ & \approx a(x,v)\frac{\partial p}{\partial v}+2\frac{\partial a}{\partial v}p(x,v)\neq \partial_v(ap) \end{aligned}$$

L'errore viene banalmente risolto definendo

$$F_{i,j\pm1}=\pm\frac{\sigma^2}{2\Delta v}\left(\frac{p_{i,j\pm1}-p_{ij}}{\Delta v}\right)+a\left(x,v\pm\frac{\Delta v}{2}\right)\left(\frac{p_{i,j\pm1}+p_{ij}}{2\Delta v}\right)$$

per cui la matrice da risolvere al posto della (**??**) è

$$\begin{aligned} \mathbf{A}^{(i)}_{jm} = & \left(-\eta-\theta a^{(i)}_{j+1/2}\right)\delta_{j+1,m} \\ & +\left(1+2\eta-\theta\left[a^{(i)}_{j+1/2}-a^{(i)}_{j-1/2}\right]\right)\delta_{jm} \\ & +\left(-\eta+\theta a^{(i)}_{j-1/2}\right)\delta_{j-1,m} \end{aligned} \tag{42}$$

con gli stessi parametri di eq. (26).

Riferimenti bibliografici

[1] N. N. Yanenko. *The Method of Fractional Steps: The Solution of Problems of Mathematical Physics in Several Variables*. A cura di Maurice Holt. Springer, 1971. ISBN: 9783642651106, 3642651100, 9783642651083, 3642651089.

[2] Randall J. LeVeque. *Finite Volume Methods for Hyperbolic Problems*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2002. ISBN: 978-0521009249.

[3] M.P. Zorzano, H. Mais e L. Vazquez. "Numerical solution of two dimensional Fokker—Planck equations". In: *Applied Mathematics and Computation* 98.2 (1999), pp. 109–117. ISSN: 0096-3003. DOI: https://doi.org/10.1016/S0096-3003(97)10161-8. URL: https://www.sciencedirect.com/science/article/pii/S0096300397101618.

[4] Tung-Lin Tsai, Jinn-Chuang Yang e Liang-Hsiung Huang. "An Accurate Integral-Based Scheme for Advection–Diffusion Equation". In: *Communications in Numerical Methods in Engineering* 17 (2001), pp. 701–713. DOI: 10.1002/cnm.442.

[5] Richard L. Burden e J. Douglas Faires. *Numerical Analysis*. 9th. Brooks Cole, 2010, p. 236.

[6] N. G. Van Kampen. *Stochastic Processes in Physics and Chemistry*. North-Holland Personal Library. North-Holland/Elsevier Science, 2007.

[7] Hannes Risken. *The Fokker-Planck Equation: Methods of Solution and Applications*. Springer, 1996.

[8] Sau Fa Kwok. *Langevin And Fokker-Planck Equations And Their Generalizations: Descriptions And Solutions*. World Scientific, 2017.