Mesure :

$$\mu\left(\bigcup_{k=1}^{\infty} E_k\right) = \sum_{k=1}^{\infty} \mu(E_k)$$

Théorème de Vaschy-Buckingham (Pi) :

$$f(q_1, q_2, \ldots, q_n) = 0$$
$$F(\pi_1, \pi_2, \ldots, \pi_p) = 0$$
$$\pi_i = q_1^{a_1}\, q_2^{a_2} \cdots q_n^{a_n}$$

```
# solver method
pi = sympy.solve(Symbol('x')**2-1, Symbol('x'))
# numeric method
g = lambda x : (x^a)*(x^b)
pi = scipy.fsolve(g,[i,1])
```

Un choc élastique (3 loi de conservations) :

$$\begin{cases} \vec{\mathbf{r_1}} \wedge m_1\vec{\mathbf{v_1}} + \vec{\mathbf{r_2}} \wedge m_2\vec{\mathbf{v_2}} = \vec{\mathbf{r_1'}} \wedge m_1\vec{\mathbf{v_1'}} + \vec{\mathbf{r_2'}} \wedge m_2\vec{\mathbf{v_2'}} \\ m_1\vec{\mathbf{v_1}} + m_2\vec{\mathbf{v_2}} = m_1\vec{\mathbf{v_1'}} + m_2\vec{\mathbf{v_2'}} \\ m_1\mathbf{v}_1^2 + m_2\mathbf{v}_2^2 = m_1\mathbf{v}_1'^2 + m_2\mathbf{v}_2'^2 \end{cases}$$

Solution vectorielle :

$$\mathbf{v}'_1 = \mathbf{v}_1 - \frac{2m_2}{m_1+m_2}\,\frac{\langle \mathbf{v}_1-\mathbf{v}_2,\, \mathbf{x}_1-\mathbf{x}_2\rangle}{\|\mathbf{x}_1-\mathbf{x}_2\|^2}\,(\mathbf{x}_1-\mathbf{x}_2),$$
$$\mathbf{v}'_2 = \mathbf{v}_2 - \frac{2m_1}{m_1+m_2}\,\frac{\langle \mathbf{v}_2-\mathbf{v}_1,\, \mathbf{x}_2-\mathbf{x}_1\rangle}{\|\mathbf{x}_2-\mathbf{x}_1\|^2}\,(\mathbf{x}_2-\mathbf{x}_1)$$

```
# consevation law
d = np.linalg.norm(ri - rj)**2
ui = vi - 2*mj / M * np.dot(vi-vj, ri-rj) / d * (ri - rj)
```

Symétrique par rotation :

$$R(\theta).\vec{\mathbf{x}} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}.\vec{\mathbf{x}}$$

Champs de force global (PFD) :

$$m\frac{\mathrm{d}\vec{\mathbf{v}}}{\mathrm{d}t} = m\vec{\mathbf{\Gamma}}(\vec{\mathbf{x}}) \Leftrightarrow \vec{\mathbf{v}_{t+1}} = \vec{\mathbf{v}_t} + dt.\vec{\mathbf{\Gamma}}(\vec{\mathbf{x}})$$

Champes de force local (Vicsek) :

$$v(\mathbf{r},\Theta) = \mathbf{r}.e^{i\Theta} = \Re(v) + i\Im(v)$$
$$\Theta_i(t+\Delta t) = \langle\Theta_j\rangle_{|r_i-r_j|<R} + \eta_i(t)$$
$$\mathbf{r}_i(t+\Delta t) = \mathbf{r}_i(t) + v\Delta t\begin{pmatrix}\cos\Theta_i(t)\\ \sin\Theta_i(t)\end{pmatrix}$$

```
# field perturbation (euler method)
v[0] += dt*(lambda x,y : (x*y, 9.81 + x+y))
# local field (complex)
U = v[0]+1j*v[1]; d = np.absolute(u - U)
```

Potentiel thermodynamique :

$$\mu_i = \left(\frac{\partial U}{\partial n_i}\right)_{V,S,n_{j\neq i}} = \left(\frac{\partial G}{\partial n_i}\right)_{P,T,n_{j\neq i}} \mathrm{d}U = -P\,\mathrm{d}V + T\,\mathrm{d}S + \sum_{i=1}^{N}\mu_i\,\mathrm{d}n_i G = V\,\mathrm{d}P - S\,\mathrm{d}T + \sum_{i=1}^{N}\mu_i\,\mathrm{d}n_i$$

Equation locale de diffusion (loi de Fick) :

$$c(t+1,x_i) = h + D\sum_{r<R} c(t,x_j) = h + D(c(t,x_{i+1}) + c(t,x_{i-1}))$$

Diffusion locale avec réactif (Turing) :

$$c(t+1, x_i) = sign\left[h + \sum_k D_k \sum_{r<R_k} c(t, x_j)\right]$$

```
# local diffusion
d = np.linalg.norm(X - X[i], axis=1)
c_i += d*np.sum(C[d<r])
new_C[c] = np.sign(h+c_i)
```

Réseau de Bravais :

$$\mathbf{R} = n_1\mathbf{a}_1 + n_2\mathbf{a}_2$$

Méthodes d'approximation des milieux continues (serie de Taylor) :

$$f(x_0 + a) = f(x_0) + a\frac{\mathrm{d}f(x_0)}{\mathrm{d}x} + \frac{a^2}{2}\frac{\mathrm{d}^2 f(x_0)}{\mathrm{d}x^2} + O(n)$$

Equation de la diffusion :

$$\frac{\partial c}{\partial t} = D\frac{\partial^2 c}{\partial x}$$

Schéma d'Euler (Laplacien) :

$$u''(x,y) = \frac{1}{h^2}\begin{bmatrix} & 1 & \\ 1 & -4 & 1 \\ & 1 & \end{bmatrix} u(x,y)$$

Résolution dans l'espace propre du systeme :

$$\frac{U_{n+1} - U_n}{dt} - \frac{1}{h^2}\begin{bmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & & & & \\ & & & & -1 & 2 \end{bmatrix} U_n = 0 \Leftrightarrow U_{n+1} = M \cdot U_n$$

```
# local and global laplacian
LAP = signal.convolve2d(U, kernel, boundary='symm', mode='same')
M = spdiags(main,[0],N*M,N*M) + spdiags(upper,[1],N*M,N*M)+spdiags(lower,[-1],N*M,N*M)
LAP = (new_U-U_).reshape(LAP)
```

Cinétique chimique de $2A + B \xrightarrow{\mathbf{k}} 2C$:

$$\frac{\mathrm{d}[A]}{\mathrm{d}x} = \frac{\mathrm{d}[B]}{\mathrm{d}x} = -\frac{\mathrm{d}[C]}{\mathrm{d}x} = -k[A]^2[B]^1$$

Modèle de Gray-Scott :

$$\begin{cases} U + 2V \xrightarrow{1} 3V \\ P \xrightarrow{\mathrm{f}} U \\ V \xrightarrow{\mathrm{f+k}} P \\ U \xrightarrow{\mathrm{f}} P \end{cases} \Leftrightarrow \begin{matrix} \dfrac{\partial u}{\partial t} = -uv^2 + f(1-u) \\ \dfrac{\partial v}{\partial t} = +uv^2 - (f+k)v \end{matrix}$$

Marche aléatoire (Chaine de Markov) :

$$\mathbb{P}\Big(X_{n+1} = j \mid X_0 = i_0, X_1 = i_1, \ldots, X_{n-1} = i_{n-1}, X_n = i\Big) = \mathbb{P}\left(X_{n+1} = j \mid X_n = i\right).$$

```
# new position of walker with continum noise
WAY = 2*np.pi*(np.random.randint(0,4,N_WALKER)/4.)
VAR = delta*(np.random.random(WALKER_NUMBER) - 0.5)
```

Force electromagnétique :

$$\vec{F} = q\vec{E} + q\vec{v} \wedge \vec{B} \Leftrightarrow X < R \Rightarrow X_{t+1} = X_t$$

Dimension Fractale (autosimilarité) :

$$a^{D_h} \ ; \ D_h = \frac{\ln(N)}{\ln(\frac{1}{r})}$$

Évolution du rapport de dimension :

$$C(r) \equiv N^{-1} \sum_{r'} \rho(r')\rho(r'+r)$$

```
# calculate distance for each node
G = nx.from_pandas_dataframe(tree, 'min_dist_index', 'initial_index')
SHORT_PATH = nx.shortest_path(G)
```

Entropie (theorie de l'information) :

$$H_b(X) = -\mathbb{E}[\log_b P(X)] = \sum_{i=1}^{n} P_i \log_b \left(\frac{1}{P_i}\right) = -\sum_{i=1}^{n} P_i \log_b P_i$$

Lagrangien (symetrie de Noether) :

$$L = E_c - E_p = T - V$$

$$\tfrac{\mathrm{d}}{\mathrm{d}t}\tfrac{\partial \mathcal{L}}{\partial \dot{q}_k} = \tfrac{\partial \mathcal{L}}{\partial q_k}$$

Double pendule (Espace des phases $\dot{\theta}(\theta)$):

$$\dot{p}_{\theta_1} = \frac{\partial L}{\partial \theta_1} = -\tfrac{1}{2} m l^2 \left(\dot{\theta}_1 \dot{\theta}_2 \sin(\theta_1 - \theta_2) + 3\frac{g}{l}\sin\theta_1\right)$$

$$\dot{p}_{\theta_2} = \frac{\partial L}{\partial \theta_2} = -\tfrac{1}{2} m l^2 \left(-\dot{\theta}_1 \dot{\theta}_2 \sin(\theta_1 - \theta_2) + \frac{g}{l}\sin\theta_2\right)$$

```
# differential equation solver method
res = integrate.odeint(derivs, state, t)
```

Fonction de transfert :

$$Y(s) = H(s)\ X(s)$$

Réponse impulsionnelle :

$$y(t) = \int_{-\infty}^{\infty} h(t-\tau)u(\tau)d\tau = \int_{-\infty}^{\infty} h(\tau)u(t-\tau)d\tau.$$

Transformations en Z :

$$H[z] = \frac{Y[z]}{X[z]} = \frac{1 - z^{-1}}{T_e}$$

```
# transfert signal
H = scipy.signal.firwin()
freq = scipy.signal.freqz(b,a)
conv = scipy.signal.convolve()
```

Series de Fourier (Diffusion) :

$$\frac{Y'(t)}{\alpha Y(t)} = \frac{X''(x)}{X(x)} T(x,t) = \sum_{n=1}^{+\infty} D_n \sin\left(\frac{n\pi x}{L}\right) e^{-\frac{n^2\pi^2\alpha t}{L^2}}$$

Coefficient de Fourier (Parseval) :

$$\sum_{n=-\infty}^{+\infty} |c_n(f)|^2 = \frac{1}{T}\int_T |f(t)|^2\,\mathrm{d}t = \|f\|^2$$

Harmonique d'un signal :

$$c_n(f) = \frac{1}{T}\int_T f(t)\mathrm{e}^{-\mathrm{i}2\pi\frac{n}{T}t}\mathrm{d}t$$

```
# integration method
cf += [[simps(X[1]*np.exp(-1j*2*n*np.pi*X[0]/T)/T, X[0]), simps(X[2]*np.exp(-1j*2*n*np.pi*X[0]/T)/T, X[0])]]
# direct method
freq = np.fftfreq(x.size, d=1/rate)
```

Filtre passe-bande (second ordre) :

$$h(jw) = \frac{A_0}{1 + j \cdot Q \cdot (x - \frac{1}{x})}$$

Filtre de Kalman ($X_t = X_{t-1} + v_{(x,t-1)}.dt.$) :

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{F}_k \hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}_k \mathbf{u}_k + \mathbf{w}_k \mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^T + \mathbf{Q}_k$$

Régulateur PID :

$$u(t) = K_\text{p} \epsilon(t) + K_\text{i} \int_0^t \epsilon(\tau)\, d\tau + K_\text{d} \frac{d\epsilon(t)}{dt}$$

Commande linéaire quadratique :

$$\dot{\mathbf{x}}(t) = A(t)\mathbf{x}(t) + B(t)\mathbf{u}(t) + \mathbf{v}(t)\mathbf{y}(t) = C(t)\mathbf{x}(t) + \mathbf{w}(t)$$

```
# control
PID = x
```

Méthodes Monte-Carlo (Theorement de transfert) :

$$\mathbb{E}\left[\varphi(X)\right] \stackrel{\text{def.}}{=} \int_\Omega \varphi\big(X(\omega)\big) \mathbb{P}(\mathrm{d}\omega) = \int_\mathbb{R} \varphi(x) \mathbb{P}_X(\mathrm{d}x)$$

Fonction d'un transition d'etats optimum :

$$V^*(s) = \max_{a \in A} \sum_{s' \in S} [R(s, \pi(s), s') + \gamma V^*(s')] T(s, a, s')$$

Equation de Bellman :

$$Q^*(s, a) = \sum_{s' \in S} [R(s, a, s') + \gamma \max_{a' \in A} Q^*(s', a')] T(s, a, s')$$

```
# update Q(s,a)
max_future_q = np.max(q_table[new_discrete_state])
current_q = q_table[discrete_state][action]
new_q = (1-alpha)*current_q+alpha*(reward+gamma*max_future_q)
q_table[discrete_state][action]=new_q
```

Récompense et Regret (Exploration/Exploitation) :

$$r_n = n\mu^* - \sum_{k=1}^{n} \mathbb{E}(\mu_{I_k})$$