uint(n) unsigned

$-2^{n-1}, \dots, 0, \dots, 2^{n-1}-1$

$0, \dots, 2^n-1$

Zweierkomplement

invertieren $+1$ bei negativ

Array slicing

array[start:stop:step]

step: Kann negativ, von hinten

stop: nicht mit drin

start, stop kann man weglassen

Bisektion

1. Def $[x_1, x_2]$
2. ∃ Nullst. falls $y(x_1)y(x_2) < 0$
3. Def. $x_3 = \frac{x_1+x_2}{2}$
4. Test ob $x_0 \in [x_1, x_3]$ oder $[x_3, x_2]$
5. Höre auf bei $|y(x_3)| < \varepsilon$ oder $|x_2 - x_1| < \varepsilon \cdot \frac{x_1+x_2}{2}$

Fehler $\delta_n = \frac{|x_2-x_1|}{2} 2^{-n}$

Verlet-Methode

$$\vec{r}_j(t+\tau) = \vec{r}_j(t) + \frac{1}{2}\left(\dot{\vec{r}}_j(t) + \dot{\vec{r}}_j(t+\tau)\right)\tau$$

$$\dot{\vec{r}}_j(t+\tau) = \dot{\vec{r}}_j(t) + \frac{1}{2}\left(\vec{a}_j(t) + \vec{a}_j(t+\tau)\right)\tau$$

Leapfrog

$$\dot{\vec{r}}_j(t+\tfrac{1}{2}\tau) = \dot{\vec{r}}_j(t-\tfrac{1}{2}\tau) + \vec{a}_j(t)\tau$$

$$\vec{r}_j(t+\tau) = \vec{r}_j(t) + \dot{\vec{r}}_j(t+\tfrac{1}{2}\tau)\tau$$

$$\dot{\vec{r}}_j(-\tfrac{1}{2}\tau) = \dot{\vec{r}}_j(0) - \vec{a}_j(0)\tfrac{1}{2}\tau$$ am Ende auch

Gleitkommazahlen

$x = (-1)^S \cdot m \cdot 2^e \qquad B = 2^{r-1} - 1$

$m = 1 + \frac{M}{2^p} \qquad e = E - B$

Typ	S	M (p bits)	E (r bits)	B
Half (16 bit)	1 bit	p=10	r=5	15
Single (32 bit)	1 bit	p=23	r=8	127
Double (64 bit)	1 bit	p=52	r=11	1023
Quadr. (128 bit)	1 bit	p=112	r=15	16383

Newton-Raphson

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Quadratische Konvergenz $\delta_{n+1} = \frac{f''(x^*)}{2f'(x^*)}\delta_n^2 + O(\delta_n^3)$

Sekantenverfahren

$$x_{n+1} = x_n - \frac{f(x_n)(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})}$$

Anonyme Funktionen

f = lambda x: np.sin(n*x)*n ⇒ f(x)

Einheiten in MD-Simulationen $k_B = 1$

Energien in K

Masse in Da

Ort in Å

Geschwindigkeit in $\sqrt{\frac{K}{Da}}$

Kraft in $\frac{K}{Å}$

Zeit in $Å\sqrt{\frac{Da}{K}}$

np.linalg.solve(A,b) np.linalg.pinv(A)@b

Inf ($E = max = 2^r - 1$, $M = 0$) zu groß

NaN ($E = max = 2^r - 1$, $M > 0$) $0/0$!

Multiplikation Mantisse der kleineren Zahl durch 2 dividieren, bis Exponent gleich, Mantisse addieren

Generalisierter y-Vektor

$$y = \begin{pmatrix} x_1 \\ \vdots \\ x_N \\ \dot{x}_1 \\ \vdots \\ \dot{x}_N \end{pmatrix} \rightarrow \frac{dy}{dt} = \begin{pmatrix} \dot{x}_1 \\ \vdots \\ \dot{x}_N \\ \ddot{x}_1 \\ \vdots \\ \ddot{x}_N \end{pmatrix}$$

B=np.hstack((A,b.reshape((n,1)))) Gauß: Zeile 1 von allen andren …

Pivotisierung: Verwende Zeile mit größten $|A_{k,1}| / |\max_{j>1} |A_{k,j}|$

LU-Zerlegung: $A = LU$, $u_{ij} = l_{ji} = 0$, für $i<j$ (upper, lower) $\quad b = Ly$, $y = Ux$

P,L,U = scipy.linalg.lu(A) p=(P.T@range(n)).astype(int) Reihenfolge der Zeilen

1. Rücksubstitution (L, b[p]) = y 2. Rücksubstitution (U, y) = x

QR-Zerlegung $A = QR^{\text{upper}}$ $Q^{-1} = Q^T$ 1. Multiplikation $y = Q^T b$ 2. Rücksubstitution $(R, y) = x$ Q,R=la.qr(A)

Trapezregel $\int_a^b f(x)\,dx \approx \frac{a-b}{2}(f(a) + f(b))$

wiederholt: $H = \frac{b-a}{N}$

$$\int_a^b f(x)\,dx = \sum_{k=1}^{N} H \frac{f(a_k) + f(b_k)}{2} = H\left[\tfrac{1}{2}f_0 + f_1 + \dots + f_{N-1} + \tfrac{1}{2}f_N\right]$$

Simpson-Regel $h = \frac{b-a}{2}$

$$\int_a^b f(x)\,dx \approx h\left[\tfrac{1}{3}f(a) + \tfrac{4}{3}f(a+h) + \tfrac{1}{3}f(b)\right] + O(h^5)$$

wiederholte Simpson-Regel h·(fi[0]+fi[-1]+2*np.sum(fi[2:-2:2])+4*np.sum(fi[1:-1:2]))/3 ; xi=np.linspace(a,b,2*n+1) ; fi=f(xi)

Bode-Methode $h = \frac{b-a}{4}$, $I_4 = \frac{4h}{90}\left[7f(a) + 32f(a+h) + 12f(a+2h) + 32f(a+3h) + 7f(b)\right]$; np.linspace(a,b,4*n+1) ; fi=f(xi) $H = \frac{a-b}{n}$ $h = \frac{H}{2}$

wiederholt: H/90*(7*fi[0]+7*fi[-1]+14*np.sum(fi[4:-4:4])+32*np.sum(fi[1:-1:2])+12*np.sum(fi[2:-2:4]))

Erste Ableitung einseitig $f'(x) = \frac{f(x+h) - f(x)}{h} + O(h)$

zweiseitig $f'(x) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2)$ bzw. $f'(x) = \frac{f(x-2h) - 8f(x-h) + 8f(x+h) - f(x+2h)}{12h} + O(h^4)$

Zweite Ableitung beidseitig $f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} + O(h^2)$

2D-Laplace

$$(\Delta_{2D}\Phi)_{ix,iy} = \frac{-4\Phi_{ix,iy} + \Phi_{ix+1,iy} + \Phi_{ix-1,iy}}{h^2} + \frac{\Phi_{ix,iy+1} + \Phi_{ix,iy-1}}{h^2}$$

2D-Laplace als Faltung

$(\Delta_{2D}\Phi)_{ix,iy}$ = scipy.signal.convolve2d(Φ, L_{2D})

$$L_{2D}^{(1)} = \frac{1}{h^2}\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix} \quad L_{2D}^{(2)} = \frac{1}{h^2}\begin{pmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{pmatrix} \quad L_{2D}^{(3)} = \frac{1}{h^2}\begin{pmatrix} 1/6 & 2/3 & 1/6 \\ 2/3 & -10/3 & 2/3 \\ 1/6 & 2/3 & 1/6 \end{pmatrix} \quad L_{2D}^{(4)} = \frac{1}{h^2}\begin{pmatrix} 1/4 & 1/2 & 1/4 \\ 1/2 & -3 & 1/2 \\ 1/4 & 1/2 & 1/4 \end{pmatrix}$$

Standardabweichung

$$\sigma_i^2 = \frac{1}{n_i - 1}\sum_{a=1}^{n_i}(y_{i,a} - \bar{y}_i)^2, \quad \bar{y}_i = \frac{1}{n_i}\sum_{a=1}^{n_i} y_{i,a}$$

χ^2/dof

$$\frac{\chi^2}{dof} = \frac{1}{N-R}\sum_{i=1}^{N}\frac{(y_i - S f(x_i; c_1, \dots, c_R))^2}{\sigma_i^2} \qquad S = \frac{\sum_{i=1}^{N} y_i \cdot f_i}{\sum_{i=1}^{N} f_i^2}$$

Numpy-/Python-Befehle zufällig: np.random.rand(n,m)

Einheitsmatrix np.eye(n) bzw. (n,m) oder np.empty((n,m))

Nullen np.zeros((n,m)) np.zeros_like(array)

Einzelnes Element M[i][j] oder M[i,j] Letztes Element v[-1] Aneinanderhängen np.vstack((x,y)) np.hstack((x,y)) np.concatenate((x,y)) beliebige Achse …((x,y),n)

Transponieren A.T Matrixmultiplikation A@B Skalarprodukt np.dot(x,y) Elementweise Multiplikation x*y

Reshape arr.reshape((n,m)) Diagonalmatrix np.diag(array, 0 / -1 / +1) Spalte/Zeile zu array (oder n×m-Matrix) vector.flatten()

Formatierte Strings "sin{0:2d}".format(i) 0tes Argument, 2 spacings d: Integer f: Fließkommazahl "{0:7.4f}".format(0.23478) 7 spacings, 4 Nachkommastellen