3D számítógépes geometria 2

Numerikus analízis alapok

Várady Tamás, Salvi Péter / BME

October 19, 2020

Gyökkereső algoritmusok

- Feladat: $f(x) = 0$
  - x skalár vagy n-dimenziós vektor
- Nemlineáris eset az érdekes
- Iteratív megoldás
  - Konvergencia sebessége általában ismert
  - Jó kezdőérték kell (nagyon fontos!)
- Egy dimenzióban keretezés
- Problémát okozhat
  - Ha nincs gyök
  - Ha többszörös gyök van
  - Ha nagyon közeli gyökök vannak

Keretező és nyitott módszerek

- Keretező módszerek
  - Megadunk a gyöknél kisebb és nagyobb értékeket
  - Mindig működnek
  - Általában lassan konvergálnak
- Nyitott módszerek
  - Csak kezdőérték kell
  - Nem mindig működnek
  - De ha igen, gyorsabban konvergálnak
- Kezdőérték gyakran a feladatból adódik
  - De jöhet inkrementális keresésből:
    - Egyenletesen felosztjuk az intervallumot
    - Előjelváltást keresünk: $f(x_k)f(x_{k+1}) < 0$, $x_i \in [x_{\min}, x_{\max}]$
    - Ha kevés részre osztunk, kihagyhatunk gyököt, ha sokra, lassú

Kettéosztás (bisection) ★

- Keret: x_{low} és x_{high}, $f(x_{\text{low}}) \cdot f(x_{\text{high}}) < 0$
- x_{mid} az intervallum közepe: $(x_{\text{low}} + x_{\text{high}})/2$
- Ha $f(x_{\text{low}}) \cdot f(x_{\text{mid}}) < 0$:
  - $x_{\text{high}} := x_{\text{mid}}$
- Ha $f(x_{\text{low}}) \cdot f(x_{\text{mid}}) > 0$:
  - $x_{\text{low}} := x_{\text{mid}}$
- Leállási feltételek:
  - Maximum iterációszám túllépése ($i_{\text{max}} \approx \log_2 \frac{x_{\text{high}} - x_{\text{low}}}{\epsilon}$)
  - Relatív hiba kicsi:

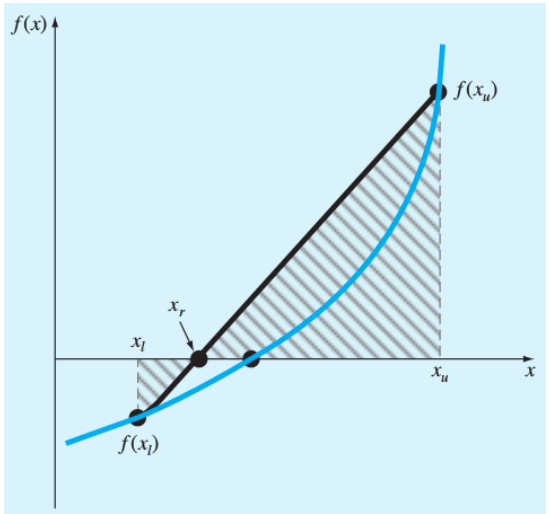
$$\left| \frac{x'_{\text{mid}} - x_{\text{mid}}}{x'_{\text{mid}}} \right| < \varepsilon$$

(x'_{mid} az újonnan számolt érték)

Hamis pozíció (lineáris interpoláció)

- Az előző algoritmus módosítása
- Új értéket lineáris interpolációval
  - Keretpontokat összekötő egyenes és az x tengely metszete
  - Hasonló háromszögekkel:

$$x = x_{\text{high}} - \frac{x_{\text{low}} - x_{\text{high}}}{f(x_{\text{low}}) - f(x_{\text{high}})} f(x_{\text{high}})$$



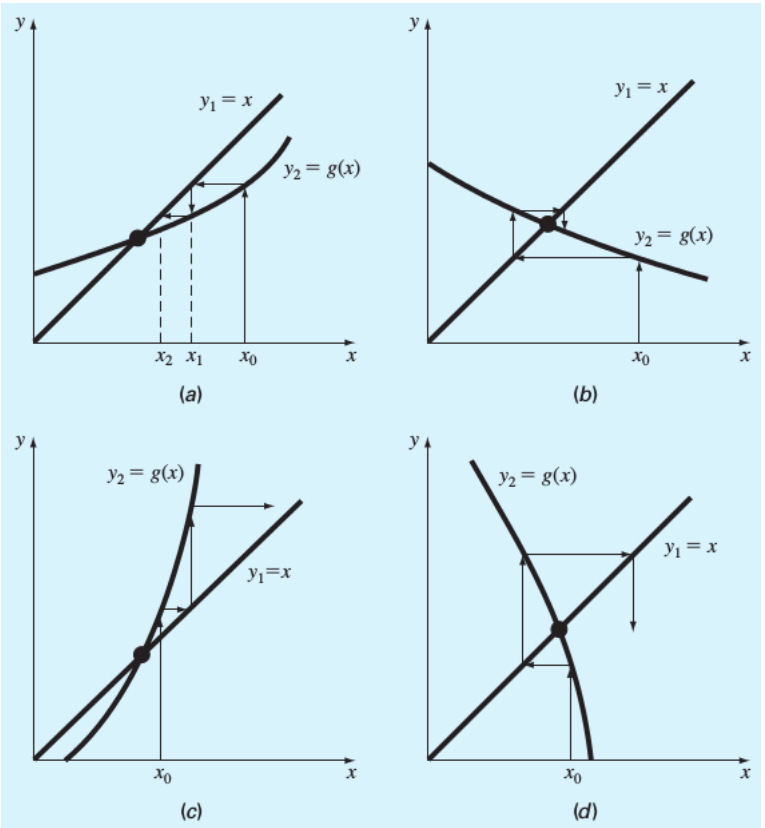
- Gyakran jobb, mint a kettéosztás
  - Nem mindig, pl. $x^{10} - 1$
  - Erős görbületű függvénynél nem hatékony
- Ridders-módszer:
  - Nem sokkal bonyolultabb (másodfokú egyenlet van benne)
  - Exponenciális függvénnyel linearizál – nagyon stabil
  - $\sqrt{2}$-rendű (2 fv.kiértékeléssel 2x annyi számjegyig pontos)

Fixpont iteráció

- $f(x) = 0$ helyett $x = g(x)$
  - Pl. mindkét oldalhoz $+x$
- Iteráció: $x_{k+1} = g(x_k)$
- Leállás, ha a változás elég kicsi:

$$\left| \frac{x_{k+1} - x_k}{x_{k+1}} \right| < \varepsilon$$

- Lineáris konvergencia



- A hiba arányos a g függvény deriváltjával
  - Ha $|g'(x)| > 1$, akkor nő

Newton–Raphson iteráció ★

- Taylor sor:

$$f(x+\delta) \approx f(x) + f'(x)\delta + \frac{f''(x)}{2}\delta^2 + \dots$$

- Kis δ esetén a lineáris utáni tagok elhagyhatóak
- Ebből $\delta = -f(x_k)/f'(x_k)$, és $x_{k+1} = x_k + \delta$
- Szokásos leállási feltételek (iteráció, relatív hiba)
- Használható többdimenziós esetben is
- Négyzetes (másodrendű) konvergencia
  - De van ahol lassú: ha $f'(x_k) \approx 0$
  - Ilyenkor instabil is (0-ra kiértékelhetetlen)
- Kell deriváltfüggvény
- Kombinálható a kettéosztással
  - Csak olyankor osztunk, ha a Newton–Raphson kimenne a keretből

Metszővonal (secant) módszer

- A derivált közelítése:

$$f'(x_k) = \frac{f(x_{k-1}) - f(x_k)}{x_{k-1} - x_k}$$

- Behelyettesítve a Newton–Raphson módszerbe:

$$x_{k+1} = x_k - \frac{x_{k-1} - x_k}{f(x_{k-1}) - f(x_k)} f(x_k)$$

- Ez gyakorlatilag a hamis pozíció képlete
  - De mindig a legújabb két pontot tartjuk meg (Nem pedig amelyek ellentétes előjelűek.)
- Alternatív módszer: csak az utolsó pont alapján

$$f'(x_k) = \frac{f(x_k + \delta x_k) - f(x_k)}{\delta x_k}$$

  - Kevésbé jó, mert több kiértékelés kell, és instabil

Brent-módszer ★

- Kombinálja a keretező és nyitott módszereket
  - Keretező: kettéosztás (ha a nyitott kivinne a keretből)
  - Nyitott: metszővonal & inverz négyzetes interpoláció
- Inverz négyzetes interpoláció
  - Mint a metszővonal módszer, csak egyenes helyett parabolával
  - „Elforgatott" $x = g(y)$ parabolát illeszt
    - Így biztosan lesz metszés az x tengellyel
    - Innen az „inverz" elnevezés
  - Illesztés Lagrange polinommal ($y_i = f(x_i)$ jelöléssel):

$$g(y) = \frac{(y - y_{k-1})(y - y_k)}{(y_{k-2} - y_{k-1})(y_{k-2} - y_k)} x_{k-2} + \frac{(y - y_{k-2})(y - y_k)}{(y_{k-1} - y_{k-2})(y_{k-1} - y_k)} x_{k-1} + \frac{(y - y_{k-2})(y - y_{k-1})}{(y_k - y_{k-2})(y_k - y_{k-1})} x_k$$

$$\Rightarrow x_{k+1} = \frac{y_{k-1} y_k}{(y_{k-2} - y_{k-1})(y_{k-2} - y_k)} x_{k-2} + \frac{y_{k-2} y_k}{(y_{k-1} - y_{k-2})(y_{k-1} - y_k)} x_{k-1} + \frac{y_{k-2} y_{k-1}}{(y_k - y_{k-2})(y_k - y_{k-1})} x_k$$

Stratégiák polinomokra

- Osztogatás
  - Az összes gyök megtalálásához
  - Ha találunk egy x_0 gyököt, leosztunk $(x - x_0)$-val
  - Vigyázat: egyre több hiba csúszik be
- Muller-módszer
  - Hasonló a metszővonal módszerhez
  - Három ponttal kvadratikus interpoláció
  - Komplex gyökpárokat is megtalál

- Sajátérték módszer
  - A sajátértékek a karakt. polinom gyökei
  - Hatékony algoritmusok vannak a kiszámításukra

$$\begin{bmatrix} -\frac{a_{n-1}}{a_n} & -\frac{a_{n-2}}{a_n} & \cdots & -\frac{a_1}{a_n} & -\frac{a_0}{a_n} \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & & 0 & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix}$$

Karakt. polinom: $a_n x^n + \cdots + a_1 x + a_0$.

Laguerre-módszer

- Komplex aritmetikát használ
- Jól működik többszörös gyökökre is
- Legyen

$$\begin{aligned}
P_n(x) &= (x-x_1)(x-x_2)\cdots(x-x_n) \\
\ln|P_n(x)| &= \ln|x-x_1| + \ln|x-x_2| + \cdots + \ln|x-x_n| \\
\frac{\partial}{\partial x}\ln|P_n(x)| &= \frac{1}{x-x_1} + \frac{1}{x-x_2} + \cdots + \frac{1}{x-x_n} = \frac{P_n'}{P_n} = G \\
-\frac{\partial^2}{\partial x^2}\ln|P_n(x)| &= \frac{1}{(x-x_1)^2} + \frac{1}{(x-x_2)^2} + \cdots + \frac{1}{(x-x_n)^2} \\
&= \left(\frac{P_n'}{P_n}\right)^2 - \frac{P_n''}{P_n} = H
\end{aligned}$$

- Tegyük fel, hogy $x_0 - x_1 = a$ és $x_0 - x_i = b$ $(i = 2, \ldots, n)$ (ahol x_0 az aktuális tippünk) [furcsa ötlet!]

Laguerre-módszer ★

- Ekkor

$$\begin{aligned} \frac{1}{a} + \frac{n-1}{b} &= G \\ \frac{1}{a^2} + \frac{n-1}{b^2} &= H \end{aligned}$$

- . . . amiből

$$a = \frac{n}{\sqrt{G \pm (n-1)(nH - G^2)}}$$

- Az előjelet úgy kell megválasztani, hogy a nevező normája nagyobb legyen
- A következő tipp $x_0 - a$
- Szokásos leállási feltételek:
  - Ha elég kicsi a hiba
  - Ha elég pici a változás
  - Ha túlléptük a maximális iterációszámot

Általános struktúra

- Solver objektum allokáció:
  - `gsl_root_f[df]solver_alloc`(típus)
- Solver paraméterek megadása:
  - `gsl_root_fsolver_set`(solver, f, lower, upper)
  - `gsl_root_fdfsolver_set`(solver, f, df, guess)
  - Függvények: `double (* f) (double x, void *params)`, `void (* fdf) (double x, void *params, double *f, double *df)`
- Iteráció (egy iterációs lépés!):
  - `gsl_root_f[df]solver_iterate`(solver)
  - Eredmény: `gsl_root_f[df]solver_root`(solver)
  - Keret: `gsl_root_fsolver_[lower|upper]`(solver)
- Solver objektum felszabadítása:
  - `gsl_root_f[df]solver_free`(solver)

Leállási tesztek

- Siker: `GSL_SUCCESS` visszatérési érték, kül.: `GSL_CONTINUE`
- `gsl_root_test_interval(x_low, x_high, eps_abs, eps_rel)`
  - Ellenőrzi, hogy az intervallum elég szűk-e
  - Ha nincsen benne 0 az intervallumban:

$$|x_{\text{low}} - x_{\text{high}}| < \varepsilon_{\text{abs}} + \varepsilon_{\text{rel}} \min(|x_{\text{low}}|, |x_{\text{high}}|)$$

  - Ha benne van:

$|x_{\text{low}} - x_{\text{high}}| < \varepsilon_{\text{abs}}$ (mivel 0 a minimum absz. az intervallumon)

- `gsl_root_test_delta(x1, x0, eps_abs, eps_rel)`
  - Ellenőrzi, hogy a változás elég kicsi-e:

$$|x_1 - x_0| < \varepsilon_{\text{abs}} + \varepsilon_{\text{rel}}|x_1|$$

Algoritmusok

- Keretező módszerek:
  - Kettéosztás: `gsl_root_fsolver_bisection`
  - Hamis pozíció: `gsl_root_fsolver_falsepos`
  - Brent-módszer: `gsl_root_fsolver_brent`
- Deriváltas módszerek:
  - Newton–Raphson: `gsl_root_fdfsolver_newton`
  - Metszővonal: `gsl_root_fdfsolver_secant`
  - Steffensen: `gsl_root_fdfsolver_steffenson`
    - A Newton–Raphson módszer változata
- Többdimenziós módszerek (nem teljes):
  - Mindenhol `gsl_root` helyett `gsl_multiroot`
  - Broyden: `gsl_multiroot_fsolver_broyden`
  - Discrete Newton: `gsl_multiroot_fsolver_dnewton`
  - Newton–Raphson: `gsl_multiroot_fdfsolver_newton`
  - Powell: `gsl_multiroot_fdfsolver_hybridsj`

Polinomok GSL-ben

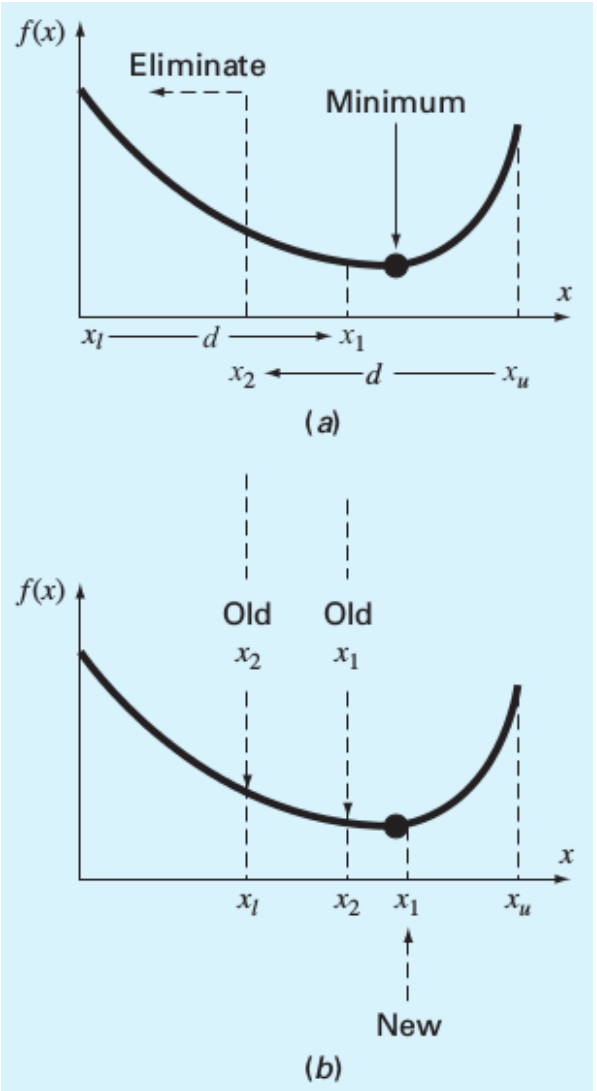
- Polinom kiértékelés: `gsl_poly_eval`(coeff[], len, x)
  - coeff[0] a konstans tag, a fokszám len-1
  - Komplex együtthatókra `gsl_complex_poly_complex_eval`
- Deriváltakkal: `gsl_poly_eval_derivs`(...)
- Gyökkeresés (sajátértékes megoldó, QR-felbontás):
  - `gsl_poly_complex_workspace_alloc`(n)
  - `gsl_poly_complex_solve`(coeff[], len, w, z)
    - z egy $2n$ elemű tömb
    - Felváltva valós és képzetes tagok
  - `gsl_poly_complex_workspace_free`(w)
- Speciális esetek:
  - `gsl_poly_[complex_]solve_quadratic`(a, b, c, x0, x1)
  - `gsl_poly_[complex_]solve_cubic`(b, c, d, x0, x1, x2)
    - A főegyüttható 1

Áttekintés

- Feladat: $f(x) \to \min$
  - Maximalizálásnál $-f(x) \to \min$
  - Egy megoldás: derivált/gradiens gyökei
- Lehetnek hozzá kényszerek
  - Egyenlőségi és egyenlőtlenségi
  - Speciális eset: lineáris programozás
- Lokális és globális minimumok
  - Különböző kezdőértékekkel próbálkozás
  - Heurisztikák
- Kényszer nélküli feladat:
  - Egy- és többdimenziós
  - Derivált használatával vagy anélkül

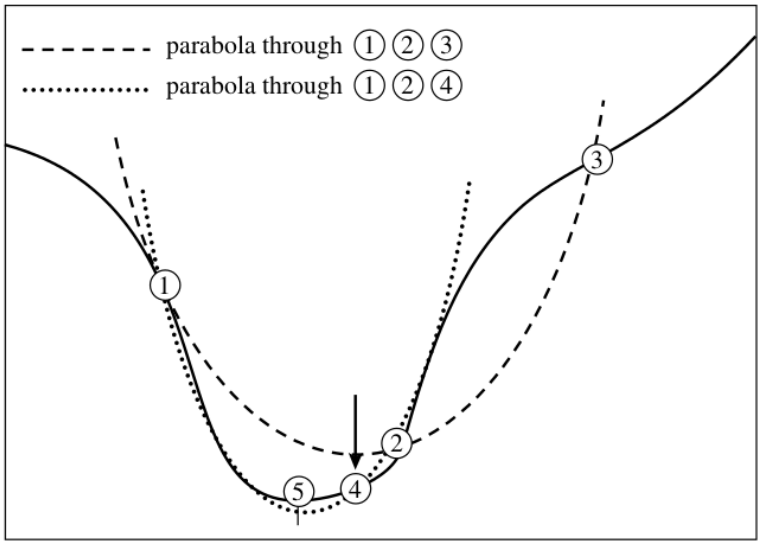
[1D] Aranymetszéses keresés ★

- Kettéosztásos gyökkeresés mintájára
  - Kell egy keret (x_{low} és x_{high})
- Aranymetszés aránya: $a : b = a + b : a$
  - $a = \phi b \Rightarrow \phi = (\sqrt{5} + 1)/2$
  - $\phi - 1 = \frac{1}{\phi}$; $\phi = [1; 1, 1, \ldots]$ lánctört
  - Fibonacci sorozat, spirál, ötszög stb.
  - Építészet, festészet, zene stb.
- $x_1 = x_{\text{low}} + d$ és $x_2 = x_{\text{high}} - d$, ahol $d = (\phi - 1)(x_{\text{high}} - x_{\text{low}})$
- Ha $f(x_1) < f(x_2)$, akkor $\Rightarrow x_{\text{low}} = x_2$
- Ha $f(x_2) < f(x_1)$, akkor $\Rightarrow x_{\text{high}} = x_1$
- Az aranymetszés miatt az egyik belső érték már ki van értékelve!



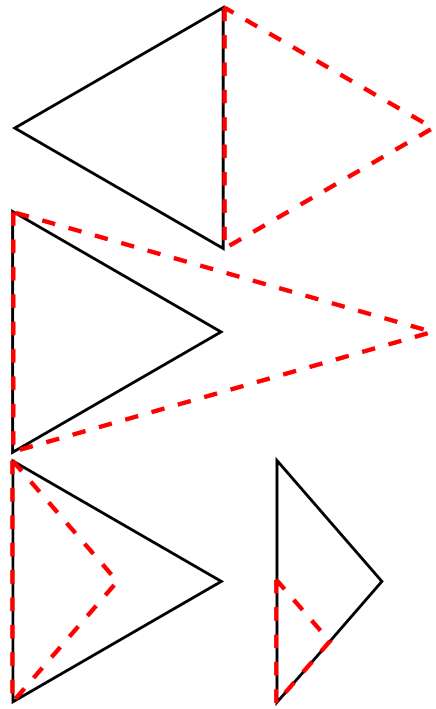
[1D] Brent-módszer

- Hasonló a gyökkereső módszerhez
- Parabolát illeszt és az aljára lép
- Elfogadjuk, ha:
  - A kereten belülre esik
  - Max. feleannyit lép, mint az előző lépésben
- Ha nem jó, akkor helyette aranymetszéses lépés
- Változat: parabola helyett deriváltra metszővonal módszer



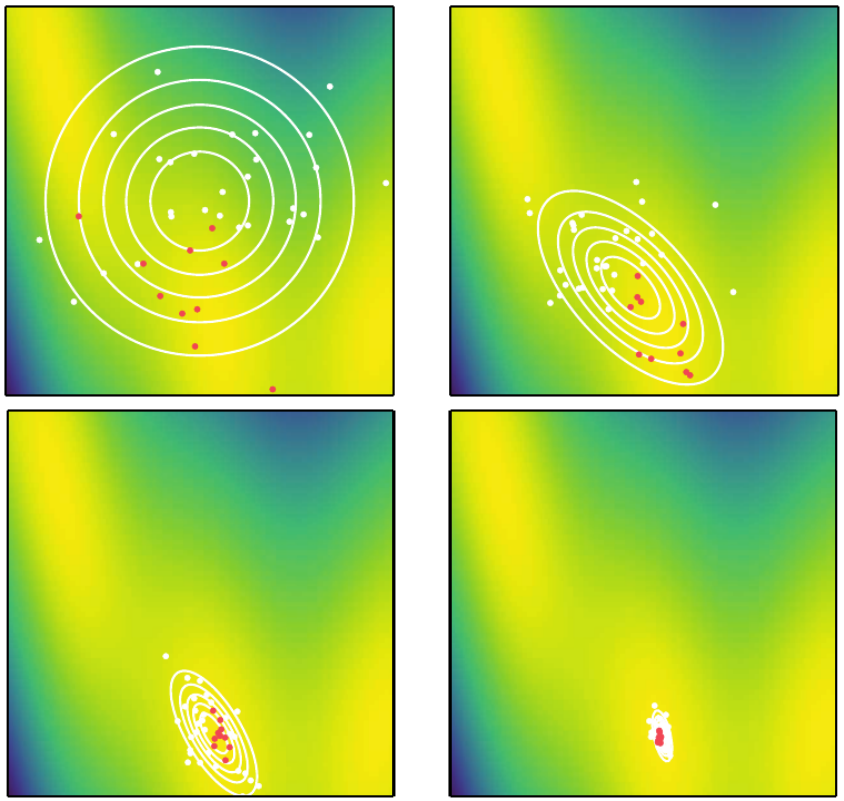
[nD] Lejtő szimplex (Nelder–Mead)

- Szimplex: $n+1$ pontú alakzat (háromszög, tetraéder stb.)
  - Legnagyobb fv.értékű pont: x_{high}, szemben levő lap: F_{high}
  - Hasonlóan x_{low} és második legnagyobb x_{nhigh}
  - Induló szimplexhez $n+1$ kezdőérték, pl. x_0 és $x_0 + \lambda e_i$
- Műveletek:
  - x_{high} pont F_{high} feletti magasságára szorzó
  - Tükrözés: -1
    - Az alapművelet, ezzel kezd az iteráció
  - Kihúzás: 2
    - Ha $f(x'_{\mathrm{high}}) < f(x_{\mathrm{low}})$, akkor nyújtjuk
  - Összehúzás: 0.5
    - Ha $f(x'_{\mathrm{high}}) > f(x_{\mathrm{nhigh}})$, akkor beszűkítjük
    - Ha nem segít: F_{low}-t az x_{low} felé (felezés)



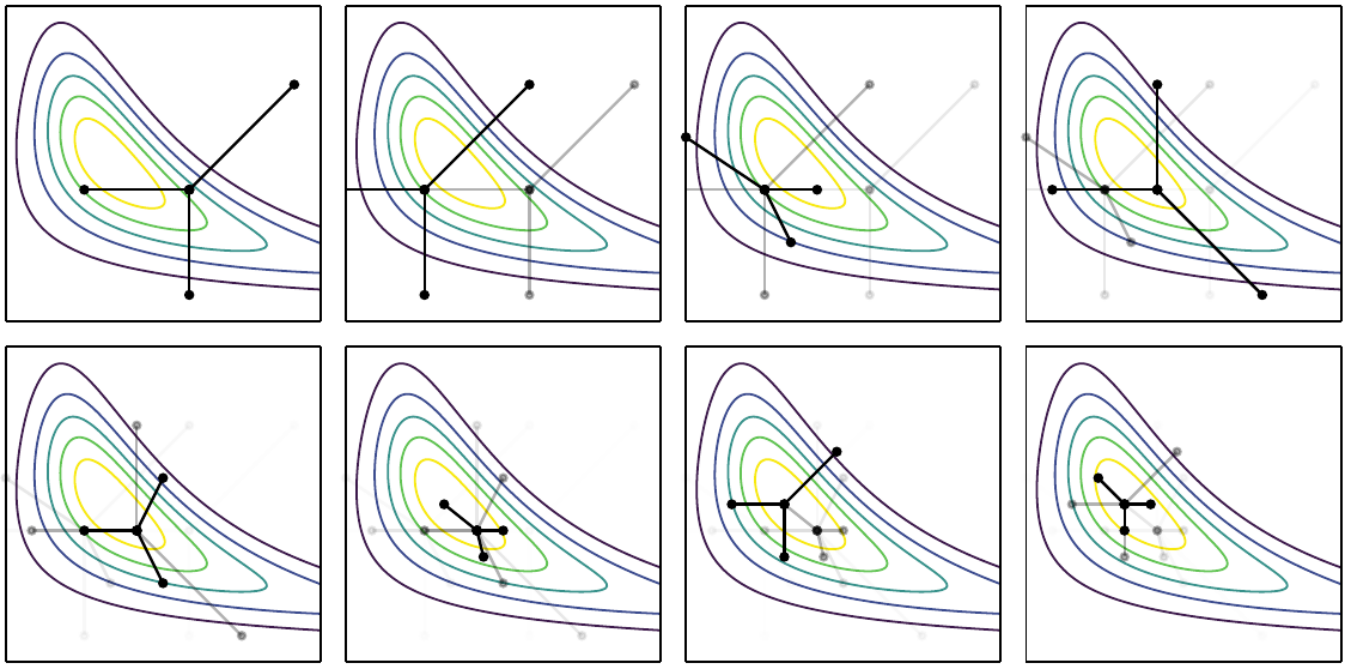
[nD] Kereszt-entrópia ★

- Feltételezünk egy valószínűségi eloszlást a minimumra
  - Minden dimenzióban külön!
- Pl. normális eloszlás
  m: várható érték, σ: szórás
- Ez alapján k mintapont
- Legjobb (elit) k_e minta
- Erre illesztünk új eloszlást
- Relaxáció m-re és σ-ra:
  $\lambda(1 - 1/i)^5$ ahol i az iteráció
- Nem engedi, hogy
  rögtön bezsugorodjon



[nD] MADS (Mesh-Adaptive Direct Search) ★

- Véletlenszerű pozitív feszítőirányok ($d_0 = -\sum_{i=1}^{n} d_i$)
- Sorban ellép, ha jobb (és megpróbál 4x akkorát lépni)
- Utána új irányok/lépésköz (javult: nagyobb, kül: kisebb)



[nD] Powell-módszer

- Irányonkénti 1D minimalizálás (pl. Brent-módszerrel)
  - $f(x_0 + \lambda u_i)$ minimalizálása λ szerint
- Probléma: kiolthatják egymás hatását
  - u irányú minimumban a gradiens merőleges u-ra
  - v irányú minimalizálásnál ezt meg akarjuk tartani
  - Feltétel: $u^T A v = 0$, ahol A az f Hesse-mátrixa
  - Cél: n lineárisan független, kölcsönösen konjugált irány
- Módszer:
  1. Induljunk ki a bázisvektorokból: $u_i = e_i$
  2. Minimalizáljunk sorban minden irányban ($\Rightarrow x_0'$)
  3. Legyen $u_i = u_{i+1}$ és $u_n = x_0' - x_0$
  4. Minimalizáljunk x_0'-ból kiindulva u_n mentén
  5. Vissza a 2-es ponthoz; n iteráció után az 1-eshez
- Alternatíva:
  - u_1 helyett azt az irányt dobjuk ki, amiben a legtöbbet változott

[nD] Konjugált gradiens

- Jobb, mint a(z ismertebb) *steepest descent* algoritmus
  - Nem „zavarják" egymást a lépések (mint a Powell-módszernél)
- Láttuk már lineáris egyenletrendszer megoldására
  - Most négyzetesen közelített függvény minimalizálására
  - $f(x) \approx \frac{1}{2} x^T A x - b^T x + c$ a Taylor-sor alapján
  - Nem ismerjük az A mátrixot $\Rightarrow$ kiváltja az 1D minimalizálás
- Módszer:
  1. Legmeredekebb irány: $\Delta x_k = -\nabla f(x_k)$
  2. Új irány: $u_k = \Delta x_k + \gamma_k u_{k-1}$, ahol γ_k-ra több képlet is van, pl.

$$\gamma_k^{\text{Fletcher–Reeves}} = \frac{\Delta x_k^T \Delta x_k}{\Delta x_{k-1}^T \Delta x_{k-1}}, \qquad \gamma_k^{\text{Polak–Ribière}} = \frac{\Delta x_k^T (\Delta x_k - \Delta x_{k-1})}{\Delta x_{k-1}^T \Delta x_{k-1}}$$

  3. 1D keresés x_k-ból u_k mentén ($\Rightarrow x_{k+1}$)

[nD] Kvázi-Newton módszerek

- Newton–Raphson módszer a gradiensre (A: Hesse-mátrix)

$$x_{k+1} = x_k - A^{-1} \cdot \nabla f(x_k)$$

- David–Fletcher–Powell (DFP) / Broyden–Fletcher–Goldfarb–Shanno (BGFS)
- Közelítő inverz Hesse-mátrix számítása: $\lim_{k\to\infty} H_k = A^{-1}$
  - Gyakran jobb, mint az igazival számolni
  - Pozitív definit A kell, hogy csökkenő irányba vigyen
  - A közelítés mindig szimmetrikus pozitív definit ($H_0 = I$)
- Nem feltétlenül tesszük meg a teljes Newton-lépést
  - Ha nem javít eleget, részlépéssel próbálkozunk (backtracking)

$$H_{k+1}^{\mathrm{DFP}} = H_k + \frac{(x_{k+1} - x_k)(x_{k+1} - x_k)^T}{(x_{k+1} - x_k)^T(\nabla f_{k+1} - \nabla f_k)} - \frac{[H_k(\nabla f_{k+1} - \nabla f_k)]\,[H_k(\nabla f_{k+1} - \nabla f_k)]^T}{(\nabla f_{k+1} - \nabla f_k)^T H_k(\nabla f_{k+1} - \nabla f_k)}$$

Lagrange szorzó

- Feladat: $f(x) \to \min$ és $g(x) = 0$
- A $g(x) = 0$ kontúrt bejárva keressük, hol nem változik az $f(x)$
  - Ha f és g kontúrja (és így gradiense) párhuzamos
  - Ha f lapos (semmilyen irányban nem változik)
  - Képlettel: $\nabla f = \lambda \nabla g \Rightarrow \mathcal{L}(x, \lambda) = f(x) - \lambda g(x)$
- Megoldandó: $\nabla \mathcal{L}(x, \lambda) = 0$
- 2D Példa: $f(x) = x_1 + x_2$, $g(x) = x_1^2 + x_2^2 - 1$

$$\nabla \mathcal{L}(x, \lambda) = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} - \begin{bmatrix} 2\lambda x_1 \\ 2\lambda x_2 \\ x_1^2 + x_2^2 - 1 \end{bmatrix} = 0$$

  - Ebből $x_1 = x_2 = 1/(2\lambda)$, az utolsóba beírva $\lambda = \pm 1/\sqrt{2}$
- Lehet több kényszer (mindegyikhez külön λ_i)
- Egyenlőtlenségi kényszerek $\Rightarrow$ Karush–Kuhn–Tucker (KKT)

Lineáris programozás

- $z = a_{01}x_1 + \cdots + a_{0n}x_n = A_0x \to \max;\ (x_l \geq 0)$
  - Kényszerek: $A_ix \leq b_i$, $A_jx \geq b_j$, $A_kx = b_k$; $(b_l \geq 0)$
  - Normálforma: csak $A_ix = b_i$ kényszerek
  - Korlátozott normálformához ezen felül
    $\forall i\ \exists j : a_{ij} > 0 \wedge (\forall k > 0 : a_{kj} \neq 0 \to k = j)$
  - Minden lin.prog. feladat átírható korlátozott normálformára
- A korlátozott normálformájú feladat táblázatba írható:

$$A = \begin{bmatrix} 0 & 2 & -4 & 0 \\ 1 & 6 & -1 & 0 \\ 0 & -3 & 4 & 1 \end{bmatrix},\ b = \begin{bmatrix} - \\ 2 \\ 8 \end{bmatrix} \equiv \begin{cases} z &= 2x_2 - 4x_3 \\ x_1 &= 2 - 6x_2 + x_3 \\ x_4 &= 8 + 3x_2 - 4x_3 \end{cases}$$

	b	x_2	x_3
z	0	2	-4
x_1	2	-6	1
x_4	8	3	-4

Szimplex módszer

- Kiválasztunk egy oszlopot, ami a z sorban pozitív (nő a célfv.)
- A pivot oszlop negatív elemei kényszerek
  - Max. növelés: a b oszlopban levő érték / |a negatív érték|
    - Az a pivot sor, ahol ez az érték a legkisebb ($\Rightarrow$pivot elem)
    - Ha nincs negatív érték, akkor bármeddig növelhető
- A pivot sorral/oszloppal csere:
  1. Pivot sor konstansszorosát hozzáadjuk a többi sorhoz úgy, hogy a pivot oszlop nullázódjon (de ahhoz nem nyúlunk)
  2. Pivot sort és oszlopot a pivot elemmel leosztjuk
  3. A pivot sort negáljuk (kivéve a pivot elemet)

	b	x_2	x_3
z	0	2	-4
x_1	2	-6	1
x_4	8	3	-4

$\Rightarrow$

	b	x_1	x_3
z	2/3	-1/3	-11/3
x_2	1/3	-1/6	1/6
x_4	9	-1/2	-7/2

$\Rightarrow$

$$\begin{cases} z_{\max} = \frac{2}{3} \\ x_1 = 0 \\ x_2 = \frac{1}{3} \\ x_3 = 0 \\ x_4 = 9 \end{cases}$$

Korlátozott normálformára váltás

- Egyenlőtlenségek kiválthatóak plusz változókkal
  - Pl. $x_1 + 3x_2 \leq 5 \equiv x_1 + 3x_2 + y_1 = 5$ (hiszen $y_1 \geq 0$)
  - Hasonlóan $x_1 + 3x_2 \geq 5 \equiv x_1 + 3x_2 - y_2 = 5$
- A „baloldali változó” feltétel is plusz változókkal
  - Minden egyenletbe beteszünk egy új z_i változót
  - Ez megváltoztatja a célfüggvényt, kivéve ha $\forall z_i = 0$
  - Segéd-célfüggvény: $z' = -\sum_i z_i$
- Első körben a segéd-célfüggvényt maximalizáljuk
  - z_i-k nullázódnak, kidobjuk $\Rightarrow$ eredeti célfüggvény

	b	x_1	x_2	y_1	y_2
z	0	2	3	0	0
z_1	10	-5	2	-1	0
z_2	3	1	-2	0	-1
z'	-13	2	-3	1	1

$$\Leftarrow \begin{cases} 2x_1 + 3x_2 & \rightarrow \max \\ 5x_1 - 2x_2 & \leq 10 \\ -x_1 + 2x_2 & \leq 3 \end{cases}$$

Szimulált hűtés

- Amikor nem ismerjük igazán a problémateret
- Cél: kiszabadulás a lokális minimumokból
- A hőmérséklet (T) szabályozza, mennyire vagyunk bátrak
  - Kezdetben nagy valószínűséggel lépünk kedvezőtlen irányba
  - Későbbi iterációkban egyre „konzervatívabbak” leszünk
  - Elfogadunk egy lépést, ha $e^{-\Delta/(kT)} > R(0,1)$
    - Δ a függvényérték változása (negatív a jó)
    - k a Boltzmann-állandónak felel meg
    - $R(0,1)$ egy véletlenszám 0 és 1 között
- Számon tartjuk a mindenkori minimumot
- Problémák:
  - Honnan jönnek a „rossz” lépések?
    - Pl. lejtős szimplexnél amikor nem csökken a célfüggvény
  - Hogyan csökkentsük a hőmérsékletet?
    - Nagyon problémafüggő...

Optimalizáló algoritmusok

- Kezelés nagyon hasonló a gyökkereső alrendszerhez
  - `gsl_[multi]min_f[df]minimizer_*`
- Szükséges lépések:
  1. Memóriafoglalás (`alloc`)
  2. Függvények és paraméterek beállítása (`set`)
     1D: keret, nD: lépéshossz, tolerancia
  3. Ciklusban iterációs lépés (`iterate`) és teszt hívása (`test_*`)
  4. Eredmény lekérdezése (`minimum`), felszabadítás (`free`)
- 1D:
  - Aranymetszés: `gsl_min_fminimizer_goldensection`
  - Brent: `gsl_min_fminimizer_brent`
  - Brent variáns: `gsl_min_fminimizer_quad_golden`

Optimalizáló algoritmusok

- nD (derivált nélkül):
  - Lejtő szimplex:
    `gsl_multimin_fminimizer_nmsimplex[2[rand]]`
    - A 2-es verzió az újabb $O(n)$ implementáció (a régi $O(n^2)$)
    - A `rand` variáns a tengelyek helyett random bázissal indul (adott lépéshosszal)
- nD (deriválttal):
  - Konjugált gradiens:
    `gsl_multimin_fdfminimizer_conjugate_[fr|pr]`
    - γ a Fletches–Reeves (`fr`) illetve Polak–Ribière (`pr`) szerint
  - Kvázi-Newton:
    `gsl_multimin_fdfminimizer_vector_bfgs[2]`
    - BFGS algoritmus, a 2-es verzió lényegesen hatékonyabb
  - Steepest descent:
    `gsl_multimin_fdfminimizer_steepest_descent`

Lineáris programozás

- Külön könyvtár: GNU Linear Programming Kit, `https://www.gnu.org/software/glpk/`
  - Kézikönyv >150 oldal – itt csak a minimum
- (De)init: `glp_create_prob()` / `glp_delete_prob(lp)`
- Optimalizációs irány: `glp_set_obj_dir(lp, GLP_MAX)`
- Sorok/oszlopok hozzáadása: `glp_add_[rows|cols](lp, n)`
- Típus: `glp_set_[row|col]_bnds(lp, i, típus, min, max)`
  - Lehet `GLP_FR`, `GLP_LO`, `GLP_UP`, `GLP_DB`, `GLP_FX`
- Célfüggvény együtthatói: `glp_set_obj_coef(lp, i, x)`
- Kényszermátrix: `glp_load_matrix(lp, n, i, j, v)`
  - Ritka mátrixos tárolás: a[i[k],j[k]]=v[k]
- Megoldás: `glp_simplex(lp, NULL)`
- Eredmény: `glp_get_obj_val(lp)` / `glp_get_col_prim(lp, i)`

Szimulált hűtés

- Egy függvényhívás: `gsl_siman_solve` – sok paraméter:
- `const gsl_rng *r` – [random generátor]
- `void *x0_p` – [kezdő konfiguráció]
- `double (* Ef) (void *xp)` – [energiafüggvény]
- `void (* take_step) (gsl_rng *r, void *xp, double step_size)` – [random lépés az xp állapotból, maximum step_size távolságra]
- `double (* distance) (void *xp, void *yp)` – [két állapot távolsága]
- `void (* print_position) (void *xp)` – [kiírja az adott állapotot]
- `void (* copyfunc) (void *source, void *dest)` – [állapotmásolás]
- `void (* copy_constructor) (void *xp)` – [másolással új állapot]
- `void (* destructor) (void *xp)` – [állapot törlése]
- `size_t element_size` – [állapot mérete]
- `gsl_siman_params_t params` – [beállítások]
- Eredmény az `x0_p` paraméterben

Szimulált hűtés

- Random generátor interface: `gsl_rng_env_setup()`, utána `gsl_rng_alloc(gsl_rng_default)` / `gsl_rng_free()`
- Ha a `print_position` nem `NULL`, akkor debug információt ír
  - #iter, #eval, T, position, energy, best energy
- Az állapotok tárolása lehet fix méretű vagy dinamikus
- Fixnél a `copyfunc`, `copy_constructor`, `destructor` = `NULL`
- Dinamikusnál az `element_size` értéke 0
- A `gsl_siman_params_t` struktúra:
  - `int n_tries` – [hány pontot nézzen lépésenként]
  - `int iters_fixed_T` – [egy-egy hőmérséklet hány iteráció]
  - `double step_size` – [maximum lépéshossz]
  - `double k, t_initial, mu_t, t_min` – [hűtési paraméterek]
    - $T \to T/\mu_T$, ahol μ_T kicsit nagyobb 1-nél

Áttekintés

- Feladat:

$$I = \int_a^b f(x)\,dx$$

- Differenciálegyenletként:

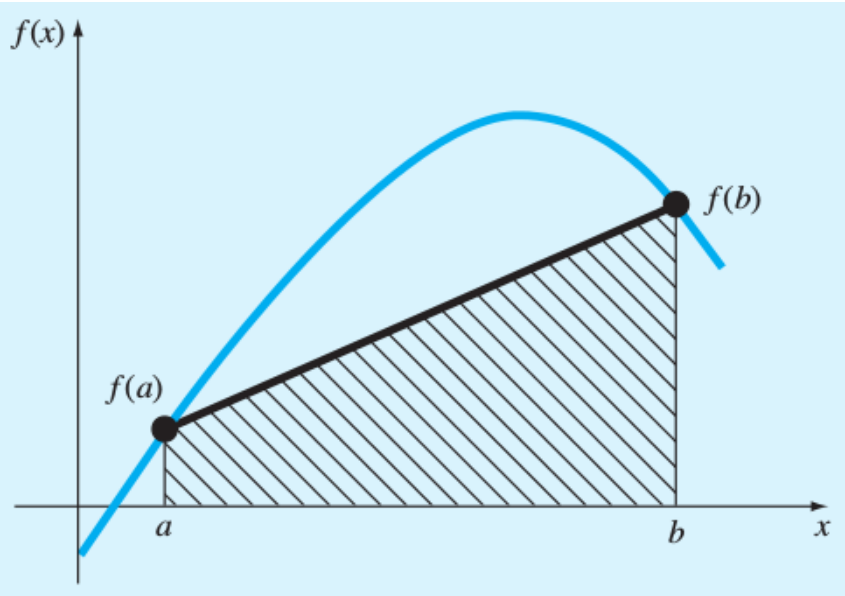
$$I = y(b), \quad \frac{dy}{dx} = f(x), \quad y(a) = 0$$

  - Ha a függvény nagyon egyenetlen, jó ilyen módszert használni
  - $\Rightarrow$ Majd a differenciálegyenleteknél
- Cél: minél pontosabban, minél kevesebb kiértékeléssel
- Választható rendű megoldások, pl. Romberg-integrálás
  - Magasabb rend $\rightarrow$ (általában) pontosabb
- Többdimenziós integrálok, pl. Monte-Carlo integrálás
- Függvényapprox. módszerek, pl. Clenshaw–Curtis kvadratúra

Newton–Cotes formulák

- Adott $x_0, x_1, \ldots, x_n, x_{n+1}$ egyenletes osztás ($x_i = x_0 + ih$)
- Ezekben ismert a függvényérték $f_i = f(x_i)$
- Zárt formula: használja a szélső értékeket
- Trapéz-szabály:

$$\int_{x_1}^{x_2} f(x)\,dx = h\left[\frac{1}{2}f_1 + \frac{1}{2}f_2\right] + O(h^3 f''(\hat{x}))$$



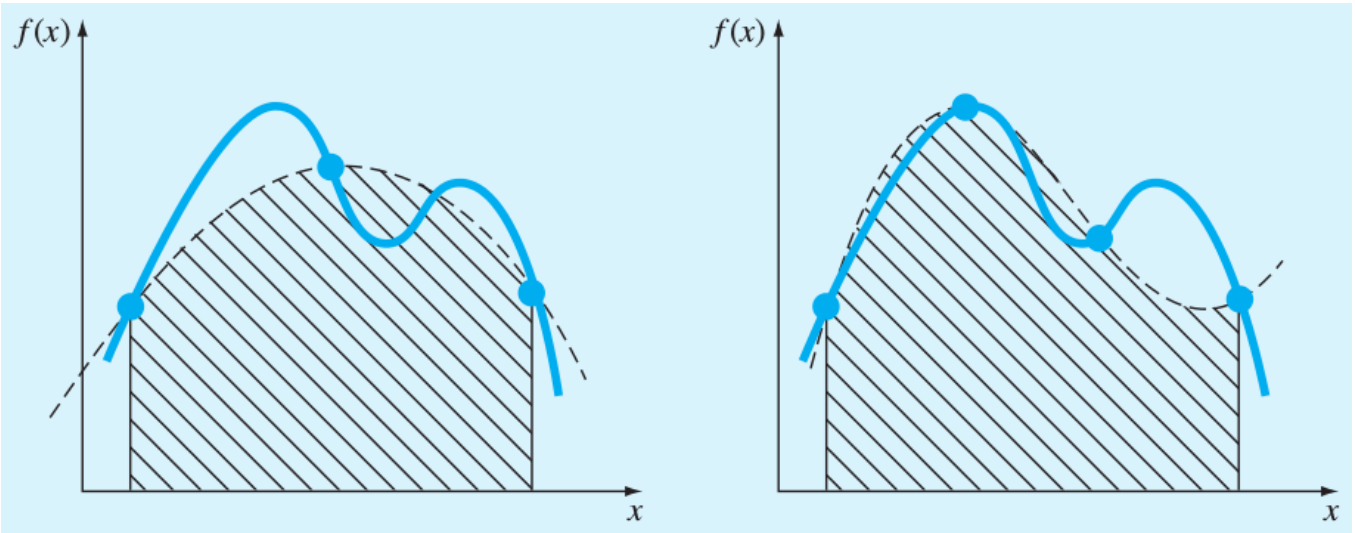
Newton–Cotes formulák

- Simpson-szabály:

$$\int_{x_1}^{x_3} f(x)\,dx = h\left[\frac{1}{3}f_1 + \frac{4}{3}f_2 + \frac{1}{3}f_3\right] + O(h^5 f^{(4)}(\hat{x}))$$

- Simpson $\frac{3}{8}$-szabály:

$$\int_{x_1}^{x_4} f(x)\,dx = h\left[\frac{3}{8}f_1 + \frac{9}{8}f_2 + \frac{9}{8}f_3 + \frac{3}{8}f_4\right] + O(h^5 f^{(4)}(\hat{x}))$$



Trapéz-módszer ★

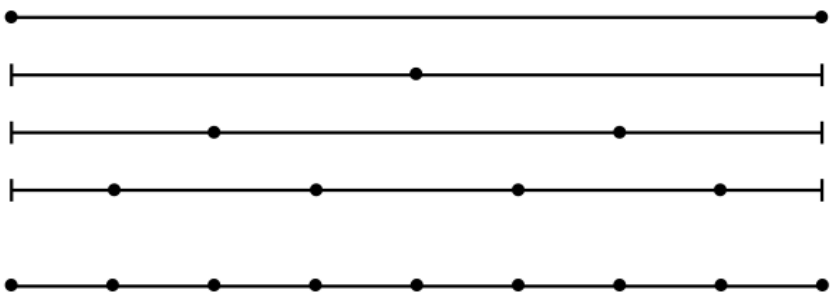
- Kiterjesztett trapéz-szabály:

$$\int_{x_1}^{x_n} f(x)\,dx = h\left[\frac{1}{2}f_1 + f_2 + f_3 + \cdots + f_{n-1} + \frac{1}{2}f_n\right] + O\left(\frac{(b-a)^3 f''(\hat{x})}{n^2}\right)$$

- Kiterjesztett Simpson-szabály:

$$\int_{x_1}^{x_n} f(x)\,dx = h\left[\frac{1}{3}f_1 + \frac{4}{3}f_2 + \frac{2}{3}f_3 + \frac{4}{3}f_4 + \cdots + \frac{2}{3}f_{n-2} + \frac{4}{3}f_{n-1} + \frac{1}{3}f_n\right] + O\left(\frac{f^{(4)}(\hat{x})}{n^4}\right)$$

- Ötlet: dinamikus sűrítés
- Ha S_k a kiterjesztett trapéz-szabály $2^k + 1$ pontra
  $\Rightarrow \frac{4}{3}S_{k+1} - \frac{1}{3}S_k$ a kiterjesztett Simpson-szabály!



Romberg-integrálás

- A trapéz-módszer kiterjesztése tetszőleges rendre
- k trapéz-módszert kombinálva eltünteti a hibatagokat
  - Csak $O(1/n^{2k})$ marad
  - Az előző algoritmus a $k = 2$ eset
- Ugyanúgy tovább sűríthető
  - Tárolni kell az előző $k - 1$ trapéz-módszeres eredményt
- Richardson-elv:
  - Polinomiális interpoláció (h_i, S_i) párokra
  - Mivel $\lim_{i\to\infty} h_i = 0$, ezért 0-ban értékeljük ki
    - Itt (h_i^2, S_i) párok, mert a trapéz-módszer hibatagjában csak páros hatványokon szerepel a h
  - Pl. $k = 2$ esetén $(1, S_i)$ és $(\frac{1}{4}, S_{i+1})$
    - Lineáris interpoláció $\Rightarrow \frac{4}{3}S_{i+1} - \frac{1}{3}S_i$

Nyílt módszerek

- Hasznos, amikor nem tudjuk kiértékelni a végpontokat
  - Szingularitás miatt, vagy mert végtelen
  - Végtelennél a változót a reciprokára cseréljük:

$$\int_a^b f(x)\,dx = \int_{1/b}^{1/a} f\left(\frac{1}{t}\right)\frac{1}{t^2}\,dt$$

- Kiterjesztett középpont-módszer:

$$\int_{x_1}^{x_n} f(x)\,dx = h\left[f_{3/2} + f_{5/2} + f_{7/2} + \cdots + f_{n-3/2} + f_{n-1/2}\right] + O\left(\frac{f'(\hat{x})}{n^2}\right)$$

- Ez is inkrementálisan számítható $\Rightarrow$ triplázni kell
  - Minden iterációban 3^{k-1} új pontot adunk hozzá
- A Romberg-interpoláció is ugyanúgy használható

Gauss-kvadratúra

- A kiértékelés helyét a módszerre bízzuk
  - Több szabadságfok $\rightarrow$ magasabb rend
- Általános képlet:

$$\int_a^b W(x) f(x)\, dx \approx \sum_{i=1}^{n} w_i f(x_i)$$

  - Adott W-re és n-re található w_i és x_i
  - Gauss–Legendre kvadratúra: $W(x) = 1$, $a = -1$, $b = 1$
    - Különböző n-ekre táblázatok w_i-re és x_i-re
  - Sok más változat is létezik:
    - Pl. Gauss–Csebisev: $W(x) = 1/\sqrt{1 - x^2}$, $a = -1$, $b = 1$
    - Pl. Gauss–Hermite: $W(x) = e^{-x^2}$, $a = -\infty$, $b = \infty$

Clenshaw–Curtis kvadratúra ★

- Csebisev-approximáció ($x \in [-1, 1]$):

$$f(x) \approx \sum_{k=0}^{n-1} c_k T_k(x) - \frac{1}{2} c_0, \quad c_k = \frac{2}{n} \sum_{i=1}^{n} f(x_i) T_k(x_i)$$

  - $T_n(x) = \cos(n \arccos x)$ az n-edfokú Csebisev-polinom
  - $T_0(x) = 1$, $T_1(x) = x$, és $T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$
  - $x_i = \cos((2i-1)\pi/2n)$ a $T_n(x)$ polinom gyökei
  - A c_i együtthatók gyorsan csökkennek, ha sima a függvény
- Az integrál képlete:

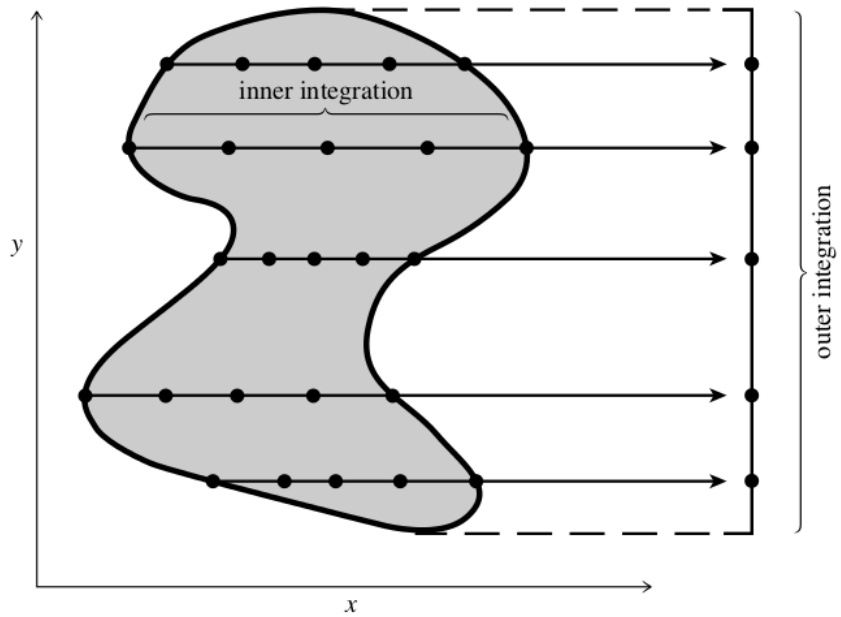
$$\int_a^b f(x)\,dx \approx (b-a)\left[\frac{1}{2}c_0 - \frac{1}{3}c_2 - \frac{1}{15}c_4 - \cdots - \frac{1}{(2k+1)(2k-1)}c_{2k} - \ldots\right]$$

- Elég addig kiértékelni, amíg c_{2k} nem lesz elhanyagolható

Stratégiák n dimenzióban

- A dimenzióval exponenciálisan nő a kiértékelések száma
- Az $n-1$ dimenziós határ nagyon bonyolult lehet
  - Nem feltétlenül konvex, vagy akár összefüggő

- Ha sima a függvény, és egyszerű a határ:
  - Egymásba ágyazott 1D integrálások
  - Többdimenziós Gauss-kvadratúra
- Egyébként:
  - Monte-Carlo integrálás
    - Statisztikai alapú
    - Lassan konvergál



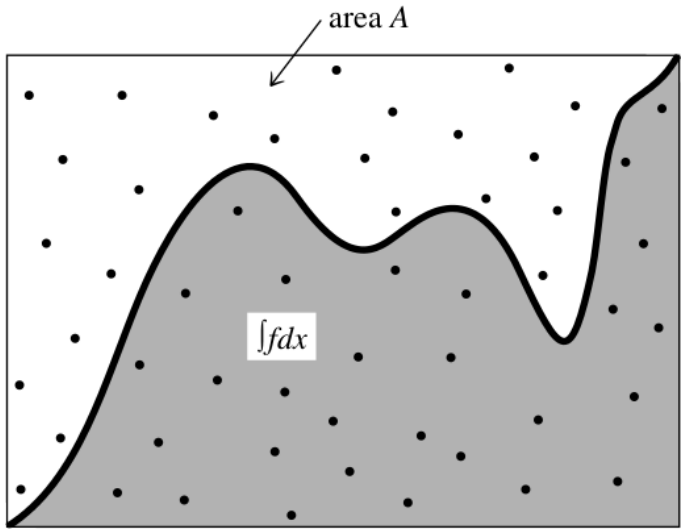
Monte-Carlo integrálás

- Kiértékeljük a függvényt N random pontban (x_i)
- Tétel:

$$\int f\,dV \approx V\langle f\rangle \pm V\sqrt{\frac{\langle f^2\rangle - \langle f\rangle^2}{N}}, \quad \langle f\rangle = \frac{1}{N}\sum_{i=1}^{N} f(x_i)$$

  ahol V a térfogat, amin integrálunk

- A $\pm$ tag csak szórás (de a hiba nem feltétlenül normáleloszlású)
- Bonyolult határfelületnél berakjuk egy nagyobb dobozba
  - A kiegészített részeken a függvényérték legyen 0



Mintavételezés

- Kvázi-random mintavételezés:
  - Alacsony diszkrepanciájú sorozattal (pl. Halton-/Sobol-sorozat)
- Fontosság szerinti mintavételezés:
  - Több mintapont ott, ahol a függvényérték nagy
  - Ha p a sűrűség (random esetben $p = 1/V$):

$$\int f \, dV \approx \left\langle \frac{f}{p} \right\rangle \pm \sqrt{\frac{\langle f^2/p^2 \rangle - \langle f/p \rangle^2}{N}}$$

- Réteges mintavételezés:
  - Részekre osztjuk a teret, mindegyikben külön integrálunk
  - Az összeg szórása csak kisebb lehet
  - Csak 2-3 dimenzióban hasznos
  - Keverhető a fontosság szerinti mintavételezéssel

QuadPack

- A kvadratúra alrendszer a QuadPack könyvtárat használja
- Elnevezés: Q(N|A)(G|W)(S|P|I|O|F|C)
  - N: nem adaptív, A: adaptív
  - G: általános függvény, W: súlyozott függvény
  - P: szinguláris pontok megadhatóak; I: végtelen határok stb
- Megadható $\varepsilon_{\mathrm{abs}}$ és $\varepsilon_{\mathrm{rel}}$
- Nem adaptív:
  - `gsl_integration_qng`: Gauss–Kronrod
    - 10 / 21 / 43 / 87-ponttal (automatikusan)
- Adaptívakhoz workspace:
  - `gsl_integration_workspace_alloc()` / `gsl_integration_workspace_free(w)`

QuadPack

- Adaptív módszerek (nem teljes lista):
  - `gsl_integration_qag`: Gauss–Kronrod
    - 15 / 21 / 31 / 41 / 51 / 61 ponttal, választható
  - `gsl_integration_qagp`: Gauss–Kronrod 21 módosítása
  - `gsl_integration_qaws`: Clenshaw–Curtis 25 módosítása
    - Csebisev-táblázatot kell hozzá foglalni: `gsl_integration_qaws_table_alloc` stb.
- Gauss–Legendre kvadratúra:
  - `gsl_integration_glfixed_table_alloc`(n)
  - `gsl_integration_glfixed`(f, a, b, table)
  - `gsl_integration_glfixed_point`(f, a, b, i, xi, wi, table) [lekérdezi az i-edik abszcisszát és súlyt]
  - `gsl_integration_glfixed_free`(table)

Monte-Carlo alrendszer

- Három típus:
  - PLAIN, MISER (réteges) és VEGAS (fontosság szerinti)
- `state = gsl_monte_TÍPUS_alloc(dim)`
- `gsl_monte_TÍPUS_init(state)`
- `gsl_monte_TÍPUS_integrate(f, xl[], xu[], dim, calls, rnd, state, result, abserr)`
  - xl és xu tömbök: a hiperkocka határai
  - calls: a függvényhívások száma
- `gsl_monte_TÍPUS_free(state)`
- A MISER és VEGAS esetében különféle paraméterek is vannak:
  - `gsl_monte_TÍPUS_params_get(state, params)`
  - `gsl_monte_TÍPUS_params_set(state, params)`

Áttekintés

- Feladat:

$$\frac{dy_i}{dx} = f_i(x, y_1, \ldots, y_n)$$

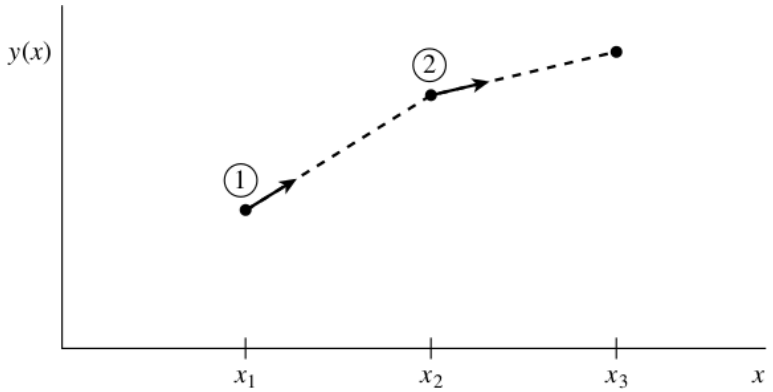
  - f_i ismert, y_i-t akarjuk meghatározni valahol
- Magasabb rendű feladatok visszavezethetők elsőrendűre
  - Új változókat kell bevezetni (ált. az eredetiek deriváltjai)
- Kezdeti érték probléma:
  - Minden y_i ismert egy x_s pontban
  - Keressük az értéküket egy x_f pontban
  - Pl. Euler, Runge–Kutta stb.
- Peremérték-probléma (2 pontra):
  - A feltételek egy része az x_s-re, másik az x_f-re van megadva
  - Pl. lövő módszer stb.

Euler- & Runge–Kutta módszer

- Nézzük a $\frac{dy}{dx} = f(x, y)$ feladatot
- Euler-módszer:

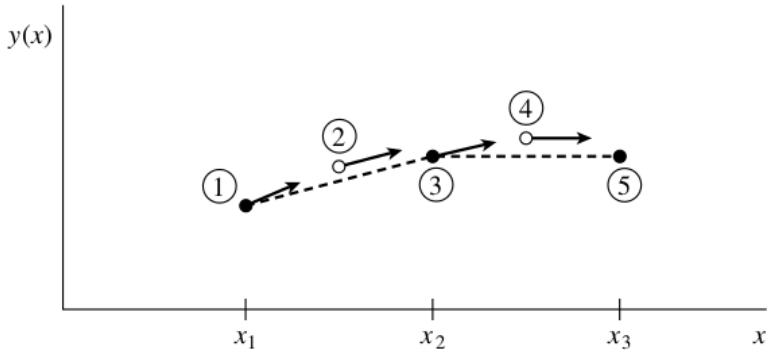
$$y_{n+1} = y_n + hf(x_n, y_n) + O(h^2)$$

ahol $x_{n+1} = x_n + h$, és y_n ismert



- Runge–Kutta midpoint módszer:

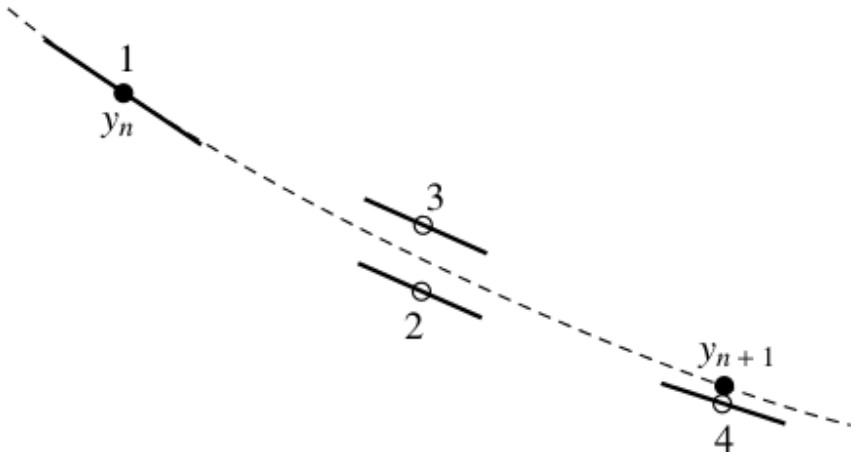
$$\begin{aligned} k_1 &= hf(x_n, y_n) \\ k_2 &= hf(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1) \\ y_{n+1} &= y_n + k_2 + O(h^3) \end{aligned}$$



Negyedrendű Runge–Kutta módszer ★

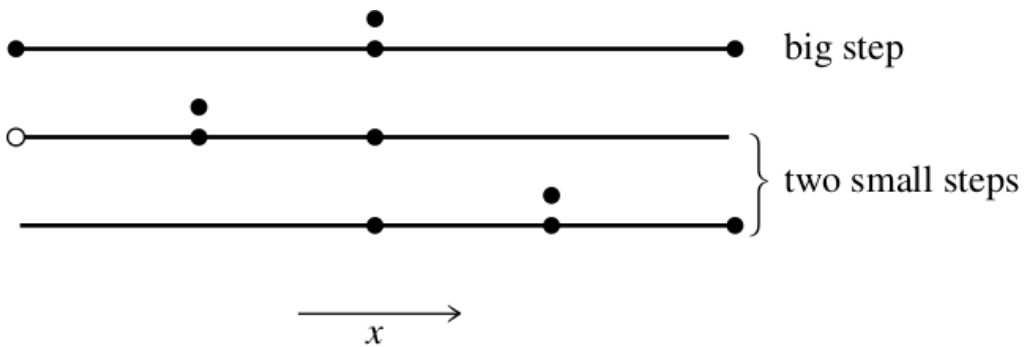
$$
\begin{aligned}
k_1 &= hf(x_n, y_n) \\
k_2 &= hf(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1) \\
k_3 &= hf(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_2) \\
k_4 &= hf(x_n + h, y_n + k_3) \\
y_{n+1} &= y_n + \frac{1}{6}k_1 + \frac{2}{6}k_2 + \frac{2}{6}k_3 + \frac{1}{6}k_4 + O(h^5)
\end{aligned}
$$

- A leggyakrabban használt KDE megoldó
- Ha f nem használja y-t, akkor ez a Simpson-szabály



Adaptív lépésközű Runge–Kutta módszer

- A sima (polinommal jól közelíthető) részen nagyobb lépések
  - Ehhez ismerni kéne a tényleges hibát
- Negyedrendű RK-val lépésduplázás:
  - Minden lépést 2x: egy darabban ill. két feleakkora lépésben
  - Egy lépés 4 kiértékelés, de a kezdő ugyanaz $\Rightarrow$ összesen 11
  - Ez 11/8=1.375 plusz munka
  - Az egyik hibája $(2h)^5$, a másiké $2h^5$
    - A két eredmény különbsége (Δ) jó hibabecslés
  - Δ-t akarjuk szinten tartani (pl. $\lambda = 0.9$, $\mu = 0.2$, $\sigma = 4$): $h \to h \cdot \lambda \cdot \min(\max((\varepsilon/\Delta)^\mu, 1/\sigma), \sigma)$



Gragg–Bulirsch–Stoer módszer

- A módosított midpoint módszer Richardson-extrapolációja
  - Kiértékelés egyre kisebb h értékekkel
  - Polinomillesztés h^2 felett, és kiértékelés $h^2 = 0$-ban
  - Adaptív lépésköz itt is alkalmazható
- Módosított midpoint módszer:

$$\begin{aligned} z_0 &= y(x) \\ z_1 &= z_0 + hf(x, z_0) \\ z_{i+1} &= z_{i-1} + 2hf(x + ih, z_i) \\ y(x + nh) &= \frac{1}{2}\left[z_n + z_{n-1} + hf(x + nh, z_n)\right] + O(nh^3) \end{aligned}$$

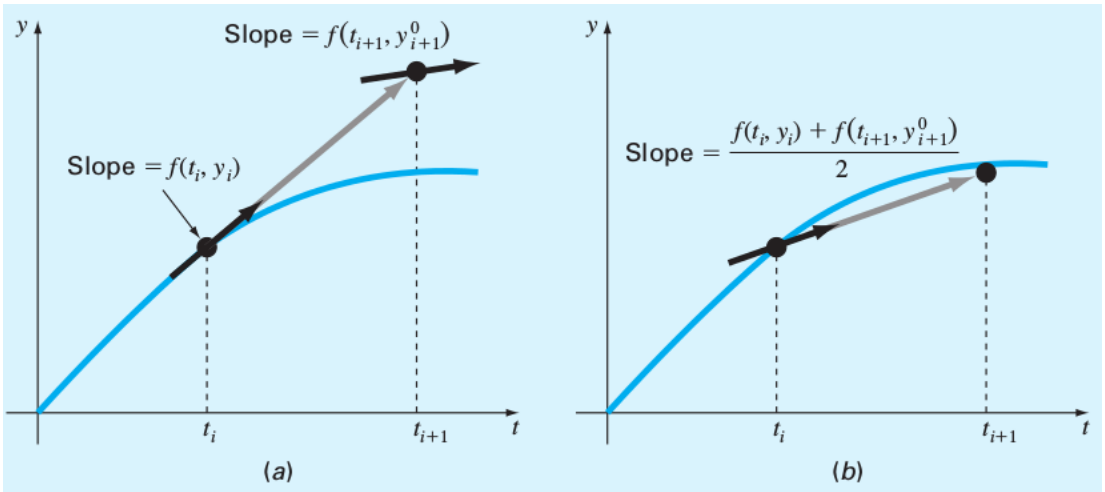
  - Csak ≈ 1 kiértékelés lépésenként (másodrendű RK-val 2)

Predikciós–korrekciós módszerek

- Heun-módszer: Euler-módszer alapján
- Sima Euler-módszer: $y^0_{n+1} = y_n + hy'_n = y_n + hf(x_n, y_n)$
- Ez alapján végderivált: $y'_{n+1} = f(x_{n+1}, y^0_{n+1})$
- A két derivált átlagával extrapolálunk:

$$y_{n+1} = y_n + h \cdot \frac{y'_n + y'_{n+1}}{2}$$

  - Az utolsó két lépés iterálható (új végderivált, új átlag, ...)



Lövő módszer

- Probléma: nincs elég információnk a kezdőpontról
- Megoldásra iterációs módszerek
- Célbalövés: a nem ismert paramétereket randomizáljuk, és így extrapolálunk a túloldalra
  - Sokszor tudunk jobbat, mint a random
  - Pl. magasabb deriváltak törlésére bevezetett deriváltat helyettesítő változó értékére differenciahányados
- A végponti határfeltétel hibáját akarjuk nullázni
  - n dimenziós gyökkeresési probléma: Newton–Raphson
  - az ismeretlen kezdeti paraméterek a változók
- Alternatíva:
  - Kétoldalról indítjuk, és középen sima találkozást akarunk

Differenciálegyenlet alrendszer

- Négy fajta függvényt szolgáltat:
  - Lépő függvény (`gsl_odeiv2_step_*`)
    - Egy lépést hajt végre
  - Adaptív lépésköz kontrolláló (`gsl_odeiv2_control_*`)
    - Több típus (az adaptív algoritmustól függően)
    - Különféle beállítások (pl. $\varepsilon_{\mathrm{abs}}$, $\varepsilon_{\mathrm{rel}}$ stb.)
  - Evolúciós függvény (`gsl_odeiv2_evolve_*`)
    - A lépő és kontrolláló függvényt hívja
    - Megfelelően változtatja a lépésközt
  - Meghajtó függvény (`gsl_odeiv2_driver_*`)
    - Ez kezel mindent, ez a fő felhasználói interfész
    - `..._alloc_KONTROLLER_new` / `..._free`
    - `..._apply`(driver, x0, xn, y0[])

Algoritmusok

- Mindegyik típus neve `gsl_odeiv2_step_*` alakú
- `_rk2`: Runge–Kutta (másodrendű)
- `_rk4`: Runge–Kutta (negyedrendű, adaptív)
- `_rk45`: Runge–Kutta–Fehlberg (adaptív RK verzió)
- `_rkck`: Runge–Kutta Cash–Karp (adaptív RK verzió)
- `_rk8pd`: Runge–Kutta Prince–Dormand (adaptív RK verzió)
- `_rk[124]imp`: Implicit Gauss Runge–Kutta (1,2 és 4-edrendű)
- `_bsimp`: [Gragg–]Bulirsch–Stoer
- `_msadams`: multistep Adams (predikciós-korrekciós módszer)
- `_msbdf`: multistep BDF (predikciós-korrekciós módszer)
  - BDF = Backward Differentiation Formula