Indeks pojęć (jest ich tu więcej, ale się nie zmieściły w spisie). Kolejność rzeczy na stronach jak w wykładach

Macierz diagonalizowalna / normalna - Macierz $A \in C^{N \times N}$, która ma N niezaleznych liniowo wektorów własnych.

Dla X – macierz z wektorów własnych: $X^{-1}AX = diag\{\lambda_1, \lambda_2, \ldots, \lambda_N\}$ //macierz diagonalna z λ na przekątnej

Macierz symetryczna – $A = A^T$, jej unormowane wektory własne tworzą bazę ortonormalną.
Macierz dodatnio określona – $x^TAx > 0$; wszystkie główne wyznaczniki (1×1, 2×2, ...) i wartości własne są > 0
Macierz osobliwa – wyznacznik równy 0.

Macierz hermitowska: $H^\dagger = H$
Macierz unitarna: $U^\dagger U = UU^\dagger = I$ oraz $U^\dagger = U^{-1}$

Macierze A i B są podobne, jeśli istnieje taka nieosbliwa macierz S, że: $B = S^{-1}AS$. Mają one takie same widma.
Ortagonalna transformacja podobieństwa: $B = O^TAO$, albo $A_k = O_k^T...O_1^TA_1O_1...O_k = P_k^TAP_k$. O jest macierzą ortogonalną.

Macierz rzadka – liczba elementów niezerowych rośnie wolnij niż N^2.
Macierz diagonalna – ma zera poza diagonalą (przekątną), oznaczana: $diag(a_i)$ lub $[diag(a_i)]$
Macierz trójdiagonalna – tak jak diagonalna, ale nad i pod przekątną ma jeszcze jedną przekątną

Macierz trójkątna – ma same zera poniżej (górna) lub powyżej (dolna) diagonali

Macierz ortogonalna – $AA^T = I$; macierz przekształcenia ortogonalnego; wyznacznik = 1 lub -1; $(AB)^T = A^T + B^T$ dla AB, BA ortogonalnych //I to macierz jednostkowa

Macierz symetryczna rzeczywista – ma rzeczywiste wartości własne, a jej unormowane wektory własne tworzą bazę w R^n. Jej norma równa się największemu modułowi wartości własnych. //n to wymiar macierzy

Ortogonalna macierz rzutowa – $P^T = P$, $P^2 = I$, $P^T = P^{-1}$

Norma macierzy – $||A|| = \max\{||Ax|| \div ||x||, x \neq 0\}$

Promień spetralny macierzy – $\rho = \sqrt{||AA^T||}$
$\rho(G) = \max\{|\lambda| : \exists y \neq 0 : Gy = \lambda y\}$ dla G diagonalizowalnej

Norma wektora x – $||x|| = \sqrt{(x_1^2 + x_2^2 + ...)}$

Wektor unormowany – o długści jeden (normalizacja = dzielenie przez długość)

Dla $A \in R^{M \times N}$
Jądro operatora: $Ker\ A = \{ x \in R^N : Ax = 0 \}$
Zasięg operatora: $Range\ A = \{ y \in R^M : \exists x \in R^N : Ax = y \}$
Dla M = N: $dim(Ker\ A) + dim(Range\ A) = N$

Biegun – punkt, w otoczeniu którego funkcja nie jest ograniczona

Szereg Taylora: $$f(x)=\sum_{k=0}^{n}\left(\frac{(x-a)^k}{k!}f^{(k)}(a)\right)+R_n(x,a)$$

Gradient funkcji: $$\nabla f = \left[\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\right]$$

Hessjan – macierz drugich pochodnych.
Funkcja Rossenbrocka – $f(x, y) = (1 - x)^2 + 100(y - x^2)^2$

W komputerze można reprezentować ściśle tylko liczby całkowite i wymierne ze skończonym rozwinięciem, ale tylko z pewnego zakresu. Inne liczby obarczone są błędem zaokrąglenia.

Liczba x jest poprawnie zaokrąglona na d-ej pozycji do liczby, którą oznaczamy $x^{(d)}$
jeśli $|\varepsilon| = |x - x^{(d)}| \leq \frac{1}{2} \cdot 10^d$, gdzie ε to błąd zaokrąglenia.

Jesli $\bar{x}$ jest przybliżeniem dokładnej wartości x, to k-tą cyfrę dziesiętną liczby $\bar{x}$ nazwiemy znaczącą,
jesli $|x - \bar{x}| \leq \frac{1}{2} \cdot 10^k$

$x = \bar{x} + \varepsilon$, dla ε będącego dowolną liczbą w przedziale $[-\frac{1}{2}10^d , \frac{1}{2}10^d]$.

Błędy, a działania:

$x + y = \bar{x} + \bar{y} + \varepsilon_x + \varepsilon_y$ *//błędy się sumują*

$x \cdot y \approx \bar{x} \cdot \bar{y} + \bar{x}\varepsilon_y + \bar{y}\varepsilon_x$ *//błędy się mnożą, mały błąd może znacznie urosnąć*

$x / y \approx \bar{x} / \bar{y} + (1 / \bar{y})\, \varepsilon_x + (\bar{x} / \bar{y}^2)\, \varepsilon_y$ *//dzielenie przez względnie małe liczby powoduje znaczne zwiększenie błędu*

Na komputerach pracujemy w arytmetyce ze skończoną dokładnością i dodawania nie zawsze jest przemienne.

Zagadnienie obliczenia ϕ(x) jest numerycznie dobrze uwarunkowane, jezeli niewielkie względne zmiany danych dają niewielkie względne zmiany rozwiązania. Inaczej jest źle uwarunkowane.

$\forall x, \bar{x} : \frac{||\phi(x) - \phi(\bar{x})||R^m}{||\phi(x)||R^m} \leq \kappa \cdot \frac{||x - \bar{x}||R^n}{||x||R^n}$ – κ to współczynnik uwarunkowania dla ϕ(x) będącego funkcją $R^n \rightarrow R^m$. Mówi on jak bardzo błąd przybliżenia wpływa na błąd wyniku. Mniejszy = lepiej.

Układ równań liniowych to układ typu Ax = b, gdzie A jest macierzą N×N, x i b są wektorami. A i b są znane.
Det A ≠ 0

Nie rozwązuje się ich wzorami Cramera, bo są bardzo złożone numerycznie.

Współczynnik uwarunkowania rozwiązywania układu równań liniowych – $\kappa = ||A|| \cdot ||A^{-1}||$
$\kappa = \max\{\lambda_i\} \div \min\{\lambda_i\}$ dla macierzy symetrycznych i rzeczywistych. Bo $||A^{-1}|| = \max_i \{ 1 / |\lambda_i| \}$

Kolumny i wiersze macierzy układu równań można zamieniać miejscami i dodawać do siebie.

Eliminacja Gaussa prowadzi do utworzenia równań z trójkątną macierzą górną. Polega ona na dzieleniu/mnożeniu przez następne współczynniki i odejmowaniu równań stronami. Ma złożoność $O(N^3)$. Algorytm nie działa, gdy trzeba dzielić przez 0, ale można wtedy próbować zamienić miejscami równania. Wymaga znajomości kolumny wyrazów wolnych.

Układ z trójkątną macierzą górną można rozwiązać metodą back substitution, która ma złożoność $O(N^2)$. Polega na podstawianiu rozwiązań od dołu do góry.

Zamiast brać kolejne współczynniki, można jako następne wybierać największe co do modułu i permutować wiersze. Nazywa się to pivoting (wybrany współczynnik to pivot, czyli element podstawowy) i zwiększa stabilność algorymu. Ma złożoność $O(N^2)$.
Pełny pivoting polega na wybieraniu pivota ze wszystkich kolumn i ma złożoność $O(N^3)$. Rzadko stosowany.

Układ równań można rozwiązać metodą macierzy odwrotnej, ALE TAK SIĘ NIE ROBI BO TO JEST $O(2N^3)$.
Zapis $A^{-1}b$ traktujemy jako znalezienie wektora z, takiego, że Az = b.

Faktoryzacja to przekształcenie macierzy równań (bez wyrazów wolnych) w iloczyn dwóch macierzy: A = Y · Z

Faktoryzacja LU – macierz ma postać $A = L \cdot U$, gdzie L jest macierzą trójkątną dolną z samymi jedynkami na diagonali, a U jest trójkątną górną. Rozwiązanie równania wygląda: $Ly = b$, $Ux = y$
Pierwsze równanie rozwiązujemy metodą forward substitution (podstawianie od góry do dołu), drugie back substitution. Koszt obu tych rozwiązań to $O(2N^2)$.

Algorytm Dollitle'a – służy do wyznaczania faktoryzacji LU. Jeśli mamy macierz A z elementami a_{ij}, L z l_{ij} oraz U z u_{ij} to:
W pierwszej kolumnie: $u_{11} = a_{11}$, $l_{21}u_{11} = a_{21}$, $l_{31}u_{11} = a_{31}$ itd. //$A_{ij} \equiv A_{\text{rząd kolumna}}$
W drugiej kolumnie: $u_{12} = a_{12}$, $l_{21}u_{12} + u_{22} = a_{22}$, $l_{31}u_{12} + l_{32}u_{22} = a_{32}$, $l_{N1}u_{12} + l_{N2}u_{22} = a_{N2}$ itd.
itd., ma złożoność $O(N^3)$, bo jeden element obliczamy O(N), a jest ich $O(N^2)$.

Algorytm Crouta – podobny do algorytmu Dollitle'a, jednak zawiera operacje Pivotingu. Złożoność się nie zmienia, przestawiamy tylko wyrazy już obliczone. Na koniec trzeba dokonać takich samych permutacji na kolumnie wyrazów wolnych jak na wierszach. Pełny Pivoting dla LU jest niemożliwy.

Przewaga LU nad eliminacją Gaussa jest taka, że wykonujemy to raz dla wielu równań z takimi samymi lewymi stronami. Faktoryzacja ta nie wymaga dodatkowej pamięci na miecierze L i U.

Faktoryzacja Cholesky'ego – macierz ma postać $A = CC^T$, gdzie C (czynnik Cholesky'ego) jest macierzą trójkątną dolną, z dodatnimi elementami diagonali. Jest mniej więcej o połowę szybsze od LU, ale macierz A musi być symetryczna i dodatnio określona oraz nie można wykorzystać pivotów. Elementy macierzy C można przechowywać w tym samym miejscu co elementy A, dzięki kolejności obliczeń.

$$l_{ss} = \sqrt{a_{ss} - \sum_{k=1}^{s-1}(l_{sk})^2} \qquad dla \quad s = 1,2,\dots,n$$

$$l_{is} = \frac{(a_{is} - \sum_{k=1}^{s-1} l_{ik}l_{sk})}{u_{ii}} \qquad dla \quad i = s+1,\dots,n$$

//l_{ij} to elementy macierzy C
//u_{ij} to elementy macierzy C^T

Rozwiązania wyznaczamy: $Cy = b$, $C^T x = y$

Faktoryzacja LDL – macierz ma postać $A = LDL^T$, gdzie L jest macierzą trójkątną dolną z samymi jedynkami na diagonali, a D jest macierzą diagonalną o dodatnich elementach. Faktoryzacja ta wymaga takich samych założeń jak Cholesky'ego, ale do jej wyznaczenia nie trzeba pierwiastkować.

Jeśli macierz jest rzadka, TRZEBA TO UWZGLĘDNIĆ w naszym algorytmie.

Faktoryzacja LU dla macierzy trójdiagonalnej jest rzędu O(N). //Ale nie można pivotować

Rozkład Cholesky'ego macierzy M-diagonalnej ma czynnik Cholesky'ego M-diagonalny. Jednak może się pojawić wypełnienie "wewnątrz" pasma. Jeśli jest ryzyko wypełnienia, rozwiązujemy układ $(PAP)^T (Px) = Pb$, gdzie P to macierz permutacji dobrana tak, by wypełnienie było możliwie małe. //wykorzystuje się do tego minimum degree algorithms, które są oparte o heurystykę

Transformacja Householdera – tworzymy macierz $P = I - 2 (uu^T / ||u^2||)$, dla u niezerowego oraz I - macierzy jednostkowej. Wtedy $P^T = P$, $P^2 = I$, $P^T = P^{-1}$, $PP^T = I$, czyli jest symetryczna, rzeczywista oraz ortogonalna = ortogonalna macierz rzutowa. Gdy $u = x \mp ||x|| \hat{e}_1$ ($\hat{e}_1$ to pierwszy wektor jednostkowy), macierz P jest macierzą Housholdera.
$Px = \pm ||x|| \hat{e}_1$, czyli działając macierzą P na wektor x, zerujemy wszystkie jego składowe oprócz pierwszej, która przyjmuje wartość jego długości. Operacja ta jest O(N).

Faktoryzacja QR – jeśli działamy transformacją Housholdera na kolejne kolumny macierzy (zaczynając od elementu diagonalnego w każdej kolumnie), otrzymujemy macierz trójkątną górną. Iloczyn przekształceń $P_1...P_N$, oznaczany Q^T, jest ortogonalny i nie musimy pamiętać kolejnych czynników. Faktoryzacja powstała w ten sposób wygląda tak: A = QR (R to wynikowa macierz trójkątna).

UWAGA: Transformacją Householdera działamy na całą macierz, jednak do każdego kroku można zastąpić policzone elementy macierzą jednostkową. Mimo iż transformacja działa na całą macierz (przez co jest $O(N^2)$) to zmienia tylko jedną kolumnę.
Złożoność całej faktoryzacji to $O(N^3)$ (jest gorsza od LU). Układ rozwiązujemy: $Rx = Q^Tb$, przez back substitution.

Obroty Givensa – macierz obrotów Givensa G(i,j) jest macierzą jednostkową, ale z elementami (i,i) i (j,j) zastąpionymi przez $c = \cos\theta$, oraz elementami (i, j) i (j, i) przez odpowiednio $s = \sin\theta$ i -s. //θ to kąt obrotu
Macierz ta jest ortogonalna. Jeśli y = G(i,j)x, to składowe y wynoszą:

$$y_k = \begin{cases} cx_i + sx_j & k = i \\ -sx_i + cx_j & k = j \\ x_k & \text{poza tym} \end{cases}$$

Jeśli chcemy, by y_j było równe 0, mamy warunek: $c = \frac{x_i}{\sqrt{x_i^2 + x_j^2}}, \quad s = \frac{x_j}{\sqrt{x_i^2 + x_j^2}}$
Obrót Givensa z tym warunkiem zeruje j-tą składową wybranego wektora, a i-ta ma wartość: $\sqrt{x_i^2 + x_j^2}$

Jeśli mamy macierz trójdiagonalną symetrzyczną A, możemy tak dobierać odpowiednie obroty, by zerować kolejne elementy poddiagonalne. Otrzymana macierz $G_{N-1}...G_2G_1A = R$ jest trójdiagonalna i otrzymujemy faktoryzację QR ($Q = G_{N-1}^T...G_1^T$). Wtedy $Rx = G_{N-1}...G_2G_1b$, a równanie $Rx = \tilde{b}$ możemy otrzymać w czasie liniowym, również w liniowym je rozwiązując (przez back substitution). A więc całkowita złożoność rozwiązania to O(N).

Wzór Shermana-Morrisona – jeśli $A_1 = A + uv^T$ to $\mathbf{A}_1^{-1} = \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}\mathbf{u}\mathbf{v}^T\mathbf{A}^{-1}}{1 + \mathbf{v}^T\mathbf{A}^{-1}\mathbf{u}}$

Algorytm Shermana-Morrisona – gdy chcemy policzyć $A_1^{-1}b$, gdzie b jest znane, a A^{-1} łatwo policzyć, wtedy szukamy: $\mathbf{w} = \mathbf{A}_1^{-1}\mathbf{b} = \left(\mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}\mathbf{u}\mathbf{v}^T\mathbf{A}^{-1}}{1 + \mathbf{v}^T\mathbf{A}^{-1}\mathbf{u}}\right)\mathbf{b}$
Żeby to zrobić, liczymy:
Az = b ^ Aq = u
Wtedy: $\mathbf{w} = \mathbf{z} - \frac{\mathbf{v}^T\mathbf{z}}{1 + \mathbf{v}^T\mathbf{q}}\mathbf{q}$ Problem sprowadza się do rozwiązania dwóch równań z macierzą A i dzięki temu możemy łatwo policzyć równanie z macierzą A_1.

Singular Value Decomposition (SVD) – $A = U[\mathrm{diag}(\omega_i)]V^T$, dla $A,U \in R^{M\times N}$, $V \in R^{N\times N}$ ortogonalna, $M \geqslant N$, $\omega_1 = 0$.
Macierz U jest kolumnowo ortogonalna. Kolumny macierzy V odpowiadające zerowym współczynnikom ω_i stanowią bazę operatora A. $|\det A| = \Pi_i\, \omega_i$. //Π to symbol iloczynu (odpowiednik Σ dla sumy)
Zastosowanie: dla Ax = b –> $x = A^{-1}b = V[\,\mathrm{diag}(\omega_i^{-1})\,]U^Tb$ //$\omega_i^{-1} = 0$, gdy $\omega_i = 0$ oraz $b \in \mathrm{Range}(A)$

Gdy macierz jest źle uwarunkowana, można użyć przybliżenia: $|\omega_i| \leqslant \tau \Rightarrow \omega_i^{-1} = 0$, gdzie τ to zadana tolerancja.
Jest to tzw. pseudoodwrotność ($\tilde{A}^{-1}$). Jeśli A jest nadokreślona: $||A(\tilde{A}^{-1}b) - b||_2 = \text{minimum}$

Metody iteracyjne – możemy uzyskać rozwiązanie w granicach błędu zaokrąglenia w skończonej liczbie kroków.

Metoda Jacobiego

$$x_i^{(k+1)} = \left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{N} a_{ij}x_j^{(k)}\right) \Big/ a_{ii}$$

//$x_i^{(k)}$ oznacza przybliżenie w k-tym kroku.
Zbieżna, gdy macierz jest silnie diagonalnie dominująca.

Metoda Gaussa-Seidela

$$x_i^{(k+1)} = \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{N} a_{ij} x_j^{(k)} \right) \Big/ a_{ii}$$

Różni się od metody Jacobiego tylko tym, że w jednym miejscu używamy wyrazów z aktualnego przybliżenia. Dzięki temu jest szybciej zbieżna, ale trudno ją zrównoleglić. Rozbieżna, gdy macierz nie jest symetryczna i dodatnio określona.

Ogólny zapis powyższych metod: $Mx^{(k+1)} = Nx^{(k)} + b$. $A = M - N$ jest podziałem macierzy. Det(M) musi być różny od 0, a promień spektralny $M^{-1}N < 0$.
Dla Jacobiego: $M = D$, $N = -(L + U)$ //D to macierz diagonalna, L trójkątna dolna, U górna
Dla Gaussa-Seidla: $M = D + L$, $N = -U$

Metoda succesive over-relaxation

$$x_i^{(k+1)} = w \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{N} a_{ij} x_j^{(k)} \right) \Big/ a_{ii} + (1 - w) x_i^{(k)},$$

ω to współczynnik relaksacji i należy go tak dobrać, by zminimalizować $\rho(M_\omega^{-1} N\omega)$ //ρ to promień spektralny

Metoda gradientów sprzężonych – forma kwadratowa $\frac{1}{2}x^TAx - b^Tx + c$ ma jedno minimum dla $Ax = b$, które jest w punkcie $\nabla f = 0$ i jest równoważne rozwiązaniu układu. Można to wykorzystać. A jest symetryczna i dodatnio określona (i dobrze, gdy jest rzadka).
Algorytm: //ten wariant metody nazywamy algebraicznym

$\mathbf{r}_1 = \mathbf{b} - \mathbf{A}\mathbf{x}_1$, $\mathbf{p}_1 = \mathbf{r}_1$
while $\|\mathbf{r}_k\| > \varepsilon$

$$\begin{aligned} \alpha_k &= \frac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{p}_k^T \mathbf{A} \mathbf{p}_k} \\ \mathbf{r}_{k+1} &= \mathbf{r}_k - \alpha_k \mathbf{A} \mathbf{p}_k \\ \beta_k &= \frac{\mathbf{r}_{k+1}^T \mathbf{r}_{k+1}}{\mathbf{r}_k^T \mathbf{r}_k} \\ \mathbf{p}_{k+1} &= \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k \\ \mathbf{x}_{k+1} &= \mathbf{x}_k + \alpha_k \mathbf{p}_k \end{aligned}$$

end

Koszt algorytmu to $O(N^3)$ (bo N kroków, każdy N^2, przez Ap_k). Dla macierzy rzadkiej, odpowiednio zaprogramowany algorytm ma złożoność $O(M \cdot N^2)$, gdzie M to szerokość pasma macierzy.
Metoda bierze nazwę od warunku sprzężenia, które spełniają wektory r_k (są wzajemnie prostopadłe w metryce A).

Metoda ma problem, jeśli macierz ma duży współczynnik uwarunkowania, bo jest wtedy powolna.
Wtedy macierz prewarunkujemy. Bierzemy macierz C, która jest odwracalna, symetryczna, rzeczywista i dodatnio określona. Wtedy $C^{-1}AC^{-1}Cx = C^{-1}b$, czyli $\tilde{A}\tilde{x} = \tilde{b}$.
Ostatecznie:

$\mathbf{r}_1 = \mathbf{b} - \mathbf{A}\mathbf{x}_1$
rozwiąż $\mathbf{M}\mathbf{z}_1 = \mathbf{r}_1$
$\mathbf{p}_1 = \mathbf{z}_1$
while $\|\mathbf{r}_k\| > \varepsilon$

$$\begin{aligned} \alpha_k &= \frac{\mathbf{r}_k^T \mathbf{z}_k}{\mathbf{p}_k^T \mathbf{A} \mathbf{p}_k} \\ \mathbf{r}_{k+1} &= \mathbf{r}_k - \alpha_k \mathbf{A} \mathbf{p}_k \end{aligned}$$

rozwiąż $\mathbf{M}\mathbf{z}_{k+1} = \mathbf{r}_{k+1}$

$$\begin{aligned} \beta_k &= \frac{\mathbf{r}_{k+1}^T \mathbf{z}_{k+1}}{\mathbf{r}_k^T \mathbf{z}_k} \\ \mathbf{p}_{k+1} &= \mathbf{z}_{k+1} + \beta_k \mathbf{p}_k \\ \mathbf{x}_{k+1} &= \mathbf{x}_k + \alpha_k \mathbf{p}_k \end{aligned}$$

end

Zagadnienie własne to problem szukania takich λ, że istnieje niezerowy wektor x, taki że $Ax = \lambda x$.
Liczbę λ nazywamy wartością własną, a x – wektorem własnym.

Równanie $Ax = \lambda x$ można zapisać również: $(A - \lambda I)\, x = 0$. Niezerowe rozwiązanie istnieje tylko, gdy $\det(A - \lambda I) = 0$. Jest to tak zwane równanie charakterystyczne macierzy, a zbiór jego rozwiązań to widmo macierzy.
Rozwiązywanie równania charakterystycznego jest na ogół numerycznie nieefektywne.

Algorytm PageRank – algorytm opracowany przez twórców Google, służący do liczenia rankingu stron WWW, który sprowadza się do zagadnienia własnego. Zakładamy, że strony można ponumerować. Popularność strony zależy od ilości linków, które do niej prowadzą oraz popularności stron linkujących. Niech r_i oznacza ranking i-tej strony. Wzór wygląda tak:
$r_i = C \cdot \Sigma_k\, r_k/l_k$, czyli suma po stronach k linkujących do i, gdzie C>0 jest stałą proporcjonalności, a l_k oznacza liczbę linków na stronie k.
Jeśli przyjmiemy notację $p_{ik} = \{l_k^{-1}$ gdy strona k linkuje do i; 0 gdy nie linkuje$\}$, to $r_i = C \cdot \Sigma_k\, p_{ik} r_k$ idzie po wszystkich k. Wtedy możemy użyć macierzy P dla wartości p_{ij} oraz wektora r dla rankingów. Otrzymujemy:
$Pr = \lambda r$, dla $\lambda = {}^1/_C$. Wektor rankingów to wektor własny macierzy P.

Metoda potęgowa – mamy macierz A, która jest symetryczna. e_i oznacza wektor własny, zakładamy, że wartości własne są uporządkowane malejąco.
Wektor y posiada rozkład $y = \Sigma_i\, \beta_i e_i$ (e_i to składowa wektora). Wtedy $A^k y = \Sigma_i\, \beta_i \lambda_i^k e_i$. Ponieważ założyliśmy, że λ_1 jest największa, to będzie dominować dla dużych k, czyli prawa strona dąży do e_1.

$$\mathbf{A}\mathbf{y}_k = \mathbf{z}_k$$
$$\mathbf{y}_{k+1} = \frac{\mathbf{z}_k}{\|\mathbf{z}_k\|}$$

Tak wyglądają kolejne iteracje. $\|y_1\| = 1$. Gdy y_k przestanie się zauważalnie zmieniać, liczymy $\lambda_1 = \|z_k\|$.

Tą metodą możemy znaleźć pierwszą wartość własną (największą). Jeśli chcemy znaleźć następne, y_k musi być prostopadły do każdego ze znalezionych z_k. Do algorytmu dodajemy krok pośredni (ortogonalizację): $z_k = z_k - e_1(e_1^T z_k)$. Reortogonalizacja jest potrzeba przynajmniej co kilka kroków (przez błąd obcięcia), co zwiększa koszt numeryczny.
Jeżeli w algorytmie zamienimy A na A^{-1}, znajdziemy wartość najmniejszą. Ujemne wartości własne powodują, że po stabilizacji, współrzędne wektora własnego zmieniają znak. Metoda jest bardzo wolna dla wartości własnych bliskich co do modułu.

Ortogonalna transformacja podobieństwa – liczymy kolejno: $P_0 = I$, $A_1 = Q_1^T A Q_1$, $P_1 = P_0 Q_1$, $A_2 = Q_2^T A_1 Q_2$ itd. W końcu: $P_S = P_{S-1} Q_S$ – macierz, której kolumnami są wektory własne macierzy A. //Q to jakaś macierz ortogonalna, taka, że A_{n+1} jest ortogonalnie podobne do A_n
Jeśli interesują nas wartości własne, a nie wektory, nie musimy pamiętać zakumulowanych macierzy transformacji.

Jeśli zrobimy $P_S^T A P_S$, otrzymamy macierz z wartościami własnymi na diagonali.

Algorytm QR – jeśli macierz ma faktoryzację QR, to $RQ = Q^T A Q$, czyli ortogonalna transformacja podobieństwa. Procedurę można iterować: $A_n = R_{n-1} Q_{n-1} = Q_n R_n$. Algorytm ten zachowuje symetrię, trójdiagonalność i postać Hessenberga. Jeśli A jest diagonalizowalna oraz ma różne i rzeczywiste wartości własne, to iteracja dąży do macierzy trójkątnej górnej z wartościami własnymi na przekątnej.

Ponieważ algorytm jest drogi numerycznie (N^3 na każdą iterację), używa się go dla macierzy rzadkich. Używając obrotów Givensa: $A' = RQ = G_{N-1} \ldots G_1 A G_1^T \ldots G_{N-1}^T$, faktoryzację macierzy można znaleźć w czasie liniowym, otrzymując macierz trójdiagonalną symetryczną.
Jeżeli macierz nie jest symetryczna, faktoryzacja prowadzi do górnej macierzy Hessenberga, która jest macierzą trójkątną górną z jedną dodatkową diagonalą pod główną. Kosz jej faktoryzacji to $O(N^2)$.

Macierze w prostej postaci (trójdiagonalnej/Hessenberga) diagonalizujemy za pomocą faktoryzacji QR.

Wartości własne macierzy Hermitowskiej – H jest macierzą hermitowską i Hu = λu. H można rozłożyć na część rzeczywistą i urojoną: H = A + iB, wtedy $A^T = A$, $B^T = -B$. Dla u = x + iy, mamy równanie: $\begin{bmatrix} A & -B \\ B & A \end{bmatrix} = \lambda \begin{bmatrix} x \\ y \end{bmatrix}$

Pierwsza macierz, $\tilde{H}$, ma 2 razy większe wymiary od H, czyli ma 2N wartości własnych, a każda jest co najmniej dwukrotnie zdegenerowana.

Rezolwenta – jeśli liczba zespolona ξ nie należy do widma A, to macierz $Z = (A - \xi I)^{-1}$, będąca funkcją argumentu ξ, nazywa się rezolwentą. $Au = \lambda u \Rightarrow (\lambda - \xi)^{-1}u = Zu$, czyli u jest wektorem własnym rezolwenty do wartości własnej $(\lambda - \xi)^{-1}$.

Transformacja Möbiusa – jeśli u_1, u_2 są wektorami własnymi, a λ_1, λ_2 odpowiadającymi im wartościami własnymi oraz $\exists\tau: \lambda_1 = \tau + \varepsilon_1, \lambda_2 = \tau + \varepsilon_2$ i $|\varepsilon_{1,2}| \ll 1$, rozważmy rezolwentę $(A - \tau I)^{-1}$. Wektory $u_{1,2}$ są jej wektorami własnymi do wartości własnych $\varepsilon_{1,2}^{-1}$, a różnica między nimi jest duża. Czyli: bliskie wartości własne są dalekie w rezolwencie, przy takich samych wektorach własnych. Koszt policzenia rezolwenty rekompensuje to, że łatwo znaleźć rozseparowane wartości.

Rezolwenty można użyć również do poszukiwania wektora znanej wartości własnej. Rezolwentę $(A - \tau I)^{-1}$ podstawiamy zamiast A do metody potęgowej (τ to liczba $\simeq \lambda$). Wynikiem jest wektor do wartości własnej λ. Macierz $A - \tau I$ jest źle uwarunkowana, dlatego trzeba zwiększyć precyzję.

Uogólnione zagadnienie własne – jeśli w równaniu charakterystycznym zamienimy I na jakieś B, otrzymamy $Ax = \lambda Bx$, czyli uogólnione zagadnienie własne, z uogólnionymi wartościami i wektorami własnymi. Zakładamy, że B posiada faktoryzację $B = CC^T$.
Mamy równanie: $\tilde{A}y = \lambda y$. $\tilde{A} = C^{-1}A(C^T)^{-1}$, $y = C^T x$. Macierz C^{-1} trzeba niestety znaleźć jawnie.
Uogólnione wektory własne są sprzężone względem macierzy B oraz ortogonalne względem iloczynu skalarnego generowanego przez macierz B.

Interpolacja jest gdy znamy kilka punktów funkcji i chcemy znaleźć co jest pomiędzy nimi. Można je połączyć prostymi liniami, ale taka funkcja może stwarzać problemy, m.in. nie jest gładka.

Macierz Vandermonde'a – n różnych punktów wyznacza wielomia n-1 stopnia. Jeśli za x w wielomanie podstawimy kolejno znane punkty, to otrzymujemy układ równań, w którym współrzędne y punktów robią za wektor b. Wyznacznik macierzy to iloczyn różnic każdej pary punktów i jest niezerowy, gdy żadne z punktów się nie pokrywają. Koszt rozwiązania tego to $O(n^3)$.

Wzór interpolacyjny Lagrange'a – zamiast poprzedniego równania, możemy użyć:
$f(x) = \Sigma_j l_j(x) f_j + E(x)$, gdzie $l_j(x) = \frac{(x - x_1)\dots(x - x_{j-1})(x - x_{j+1})\dots(x - x_n)}{(x_j - x_1)\dots(x_j - x_{j-1})(x_j - x_{j+1})\dots(x_j - x_n)}$

l_j jest wielomianem n-1 stopnia i $l_j(x_k) = \delta_{jk}$. Jeżeli f(x) jest stopnia ⩽n-1, to E(x), czyli błąd interpolacji (reszta), znika tożsamościowo. Mówimy, że intepolacja ma dokładność n-1. Takie równanie rozwiązujemy w czasie $O(n^2)$.
//$f_j \equiv f(x_j)$, n to liczba węzłów interpolacji

Alternatywnie: $l_j(x) = \frac{p_n(x)}{(x - x_j)p_n'(x_j)}$, p_n jest iloczynem Π $(x-x_i)$, a p_n' to jego pochodna.
Oznaczmy: $y(x) = \Sigma_j l_j(x) f_j$ (wielomianowa część wzoru, bez reszty).

Funkcja $F(z) = f(z) - y(z) - [f(x) - y(x)]\,(\,p_n(z) \div p_n(x)\,)$ ma n+1 miejsc zerowych równych $x_1,\dots x_n$, zakładając, że jest odpowiednio gładka, stosujemy n-krotnie twierdzenie Rolle'a i okazuje się, że n-ta pochodna ma co najmniej jedno miejsce zerowe, które oznaczamy ξ.
Otrzymujemy: $E(x) = (\,p_n(x) \div n!\,)f^{(n)}(\xi)$. Trudność w oszacowaniu błędu interpolacji polega na trudności w oszacowaniu wysokich pochodnych funkcji.

Oscylacje Rungego – zbyt duża liczba punktów do utworzenia wielomianu prowadzi w granicach przedziału do tzw. oscylacji Rungego, czyli wysokich amplitud między punktami. Dzieje się tak, bo narzucamy punkty wielomianom, które są sztywną strukturą.

Interpolacja Hermite'a – jeśli oprócz wartości funkcji w punktach znamy wartości pochodnej, można skonstruować interpolację wielomianową rzędu 2n – 1 (bo mamy 2n warunków).
Mamy funkcję: $y(x) = \Sigma_i h_i(x)f_i + \Sigma_i \bar{h}_i(x)f_i' + E(x)$, gdzie $h_i(x) = (1 - 2(x - x_i)l_i'(x_i))l_i^2(x)$ i $h_i(x) = (x - x_i)l_i^2(x)$, oraz $E(x) = \frac{p_n^2(x)}{(2n)!}f^{(2n)}(\xi)$

Wielomian ten zgadza się z interpolowaną funkcją oraz jej pochodną w zadanych węzłach.

Funkcja sklejana – "splajn" rzędu k to funkcja, która lokalnie jest wielomianem rzędu k i jest k-1 krotnie różniczkowalna (czyli pochodne rzędu < k-1 są ciągłe).

Splajn kubiczny to splajn trzeciego rzędu (cubic spline). Jeśli oprócz wartości funkcji znamy jakieś wartości ξ_i w węzłach, konstruujemy takie wyrażenie interpolacyjne, że będzie się zgadzać z wartościami f_i w węzłach, a druga pochodna wyrażenia będzie się zgadzać z ξ_i.
W przedziałach $[x_j, x_{j+1}]$ konstruujemy wielomany 3-stopnia: $y_j(x) = Af_j + Bf_{j+1} + C\xi_j + D\xi_{j+1}$, dla

$$A = \frac{x_{j+1} - x}{x_{j+1} - x_j}, B = \frac{x - x_j}{x_{j+1} - x_j}, C = \frac{1}{6}(A^3 - A)(x_{j+1} - x_j)^2, D = \frac{1}{6}(B^3 - B)(x_{j+1} - x_j)^2$$

Jednak tak na prawdę nie znamy wartości ξ. Ale, ξ to wartości drugich pochodnych, czyli żeby istniały, pierwsze pochodne muszą być ciągłe, czyli możemy zarządać, by pochodna $y_j(x)$ z prawej strony równała się $y_{j+1}(x)$ z lewej. Otrzymamy równanie:

$$\frac{x_j - x_{j-1}}{6}\xi_{j-1} + \frac{x_{j+1} - x_{j-1}}{3}\xi_j + \frac{x_{j+1} - x_j}{6}\xi_{j+1} = \frac{f_{j+1} - f_j}{x_{j+1} - x_j} - \frac{f_j - f_{j-1}}{x_j - x_{j-1}}$$

Jest to trójdiagonalny układ równań na nieznane wartości ξ.

Ponieważ dla n węzłów mamy n-2 wewnętrznych punktów zszycia, mamy układ n-2 równań z n niewiadomymi, czyli trzeba podać więcej warunków. Najczęściej jest to $\xi_1 = \xi_n = 0$, przez co uzyskujemy naturalny splajn kubiczny. Możemy oczywiście narzucić inne warunki jeśli trzeba.

Jeśli węzły są równoodległe, równanie jest szczególnie proste:

$$\begin{bmatrix} 4 & 1 & & & & & & \\ 1 & 4 & 1 & & & & & \\ & 1 & 4 & 1 & & & & \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \\ & & & & 1 & 4 & 1 & \\ & & & & & 1 & 4 & \end{bmatrix} \begin{bmatrix} \xi_2 \\ \xi_3 \\ \xi_4 \\ \vdots \\ \xi_{n-2} \\ \xi_{n-1} \end{bmatrix} = \frac{6}{h^2} \begin{bmatrix} f_1 - 2f_2 + f_3 \\ f_2 - 2f_3 + f_4 \\ f_3 - 2f_4 + f_5 \\ \vdots \\ f_{n-3} - 2f_{n-2} + f_{n-1} \\ f_{n-2} - 2f_{n-1} + f_n \end{bmatrix}$$

Macierz po lewej posiada łatwą do znaleziena faktoryzację Cholesky'ego. Z prawej strony są ilorazy różnicowe z dokładnością do czynników "6".

Aby użyć splajnów w praktyce, robimy:
1) Rozwiązujemy układ na wartości ξ, co jest O(n) i robimy to tylko raz.
2) Wykorzystujemy równanie $y_j(x)$ tyle razy, ile chcemy znaleźć wartości między węzłami. W każdym przedziale $[x_j, x_{j+1}]$ używamy odpowiedniego wielomianu $y_j(x)$.

Splajny bikubiczne – jeśli mamy funkcję dwóch zmiennych i znamy siatkę punktów, wiersze tej siatki odpowiadają wartościom y, a kolumny x. Chcemy znaleźć wartość funkcji punktów wewnętrznych siatki: f(x*, y*).

1) Przeprowadzamy splajn przez każdy z wierszy. //$O(n^2)$, bo n wierszy razy O(n)
2) Liczymy wartości każdego z tych splajnów w punkcie x*. Otrzymujemy n wartości w $f(x^*, y_n)$
3) Przez punkty x* przeprowadzamy splajn po y i liczymy wartość splajnu w (x*, y*)

Zamiast wielomianami, możemy interpolować za pomocą funkcji wymiernych:
$r(x) = (P_\mu(x) \div Q_\nu(x)$. Funkcje wymierne mogą modelować więcej różnych zachowań.

Algorytm Floatera-Hormanna – mamy węzły $x_1,...x_n$ i wybieramy parametr interpolacji $0 \leqslant d \leqslant n$. Jeśli $p_i(x)$ jest wielomianem interpolującym na $x_i,...x_{i+d}$, to

$$r(x) = \frac{\sum_{i=0}^{n-d} \lambda_i(x)\, p_i(x)}{\sum_{i=0}^{n-d} \lambda_i(x)}, \lambda_i(x) = \frac{(-1)^i}{(x - x_i) \cdots (x - x_{i+d})}$$

r(x) jest gładką mieszanką wielomianów interploracyjnych. Nie ma biegunów na osi rzeczywistej.

Alternatywnie (barycentrycznie):

$$r(x) = \frac{\sum_{k=0}^{n} \frac{w_k}{x - x_k} f_k}{\sum_{k=0}^{n} \frac{w_k}{x - x_k}}, w_k = \sum_{i \in J_k} (-1)^i \prod_{j=i, j \neq k}^{i+d} \frac{1}{x_k - x_j}$$

//Wagi ω_k obliczamy tylko raz.

//$J_k = \{i \in I : k - d \leqslant i \leqslant k\}$, $I = \{0,1,...,n - d\}$

Parametr d wystarcza brać 3, ale nieraz potrzebne jest 8. Jeśli interpolowana funkcja jest dostatecznie gładka to błąd iterpolacji wynosi $O(h_{max}^{d+1})$ //h_{max} to największa odległość między punktami
Algorytm ten w pewnych sytuacjach jest tak dokładny jak splajny i mniej złożony.

Różniczkowanie numeryczne – $f'_j = (f_{j+1} - f_{j-1}) \div 2h$ – symetryczne przekształcenie wzoru na pochodną. Stosowanie go jest niezalecane, bo jest mało dokładny i zależy od rozmiaru kroku imterpolacji (większy = gorzej).
Niesymetryczne wzory mają postać: $(f_{j+1} - f_j) \div h$ oraz $(f_j - f_{j-1}) \div h$ i są jeszcze gorsze.

Różniczkowanie splajnu: $y'_{j+1} = {}^1/_h (f_{j+2} - f_{j+1}) - {}^1/_6\, h (2\xi_{j+1} + \xi_{j+2})$ (w lewym krańcu), po przekształceniach:

$$y'(x_j) = \frac{f_{j+1} - f_{j-1}}{2h} - \frac{1}{12} h \left(\xi_{j+1} - \xi_{j-1}\right)$$

, j>1. Wykorzystujemy znajomość drugiej pochodnej.

Numerycznie wolno obliczać całki, o których wiemy, że istnieją. Kwadratura to całka przybliżona, którą otrzymuje się przez całkowanie odpowiednich wielomianów interpolacyjnych. Jeśli $f(x) = y(x) + E(x)$ (gdzie $y(x) = \Sigma_i h_i(x) f_i$ + ewentualne pochodne), a E(x) to błąd interpolacji, otrzymujemy: $\int_a^b f(x)\,dx = \Sigma_i H_i f_i + E$, dla $H_i = \int_a^b h_i(x)\,dx$, $E = \int_a^b E(x)\,dx$.

Należy tak wykonać interpolację, by całki z ewentualnych pochodnych znikały tożsamościowo. Z całkowania wielomianów Lagrange'a otrzymuje się kwadratury Newtona-Cotesa, a z interpolacji Hermite'a – kwadratury Gaussa. Jeśli granice całkowania wielomianów Lagrange'a są węzłami interpolacji, to otrzymujemy zamknięte kwadratury Newtona-Cotesa. Błąd całkowania E powinien być proporcjonalny do pochodnej rzędu (n+1), jednak dla parzystych wielomianów jest on o rząd wyższy (ze względu na symetrie i takie tam).

Kwadratury Newtona-Cotesa – opierają się na interpolacji wielomianami niskiego rzędu, bo są prostsze i jest mniej oscylacji Rungego, oraz szacowanie pochodnych wysokiego rzędu jest trudne.

4 najczęściej stosowane kwadratury: //$\zeta \in [a,b]$

Metoda trapezów

$$\int_a^b f(x)\,dx \simeq \frac{b-a}{2}(f_0 + f_1)$$
$$E = -\frac{1}{12}(b-a)^3 f''(\zeta)$$

Metoda 3/8

$$\int_a^b f(x)\,dx \simeq \frac{b-a}{8}(f_0 + 3f_1 + 3f_2 + f_3)$$
$$E = -\frac{3}{80}\left(\frac{b-a}{3}\right)^5 f^{(4)}(\zeta)$$

Metoda Simpsona

$$\int_a^b f(x)\,dx \simeq \frac{b-a}{6}(f_0 + 4f_1 + f_2)$$
$$E = -\frac{1}{90}\left(\frac{b-a}{2}\right)^5 f^{(4)}(\zeta)$$

Metoda Milne'a

$$\int_a^b f(x)\,dx \simeq \frac{b-a}{90}(7f_0 + 32f_1 + 12f_2 + 32f_3 + 7f_4)$$
$$E = -\frac{8}{945}\left(\frac{b-a}{4}\right)^7 f^{(6)}(\zeta)$$

Wzór na kwadraturę przybliżoną podaje pole pod wykresem odpowiedniej funkcji.

Większą dokładność, niż kwadraturami wyższego poziomu, można uzyskać dzieląc przedział na mniejsze i używając niższych kwadratur – są to tak zwane kwadratury złożone. Procedurę tą można iterować. Podziały najlepiej zagęszczać tak, by węzły grubszego podziału były też węzłami gęstszego, w ten sposób można korzystać z obliczonych już wartości.

Stosowanie kwadratur złożonych dodatkowo zmniejsza błąd. Np. dla metody trapezów, błąd zmniejsza się n^2 razy dla n podziałów (same błędy się sumują).
Złożony wzór trapezów (dla wielokrotnych podziałów) ma postać: $I_N \simeq h\,(\frac{1}{2}f_0 + f_1 + \ldots + f_N + \frac{1}{2}f_N)$, gdzie h to długość najmniejszego podprzedziału (średnica podziału), f_N jest wartością w prawym krańcu przedziału, a $N = 2^k$.
Widać, że przy kolejnych iteracjach trzeba pamiętać tylko sumę wartości z węzłów wyższego rzędu. Kończymy iterować, gdy uzyskamy zadaną dokładność. Jeżeli wartość całki jest bardzo mała/duża, trzeb stosować dokładność względną:

$$\frac{|I_{k+1} - I_k|}{|I_k| + \varepsilon'} < \varepsilon$$ //$0 < \varepsilon' << 1$, zabezpiecza przed zerem w mianowniku

Ekstrapolacja Richardsona – jak iterujemy zagęszczanie przedziału, otrzymujemy ciąg różnych przybliżeń całki, które możemy wykorzystać. Jeśli mamy 2 całki z metody trapezów:

$$I = I_n - \frac{(b-a)^3}{12n^2} f''(\zeta_n)\,,$$
$$I = I_{2n} - \frac{(b-a)^3}{12(2n)^2} f''(\zeta_{2n})$$

n i 2n to liczba podprzedziałów. Jeśli pochodne są odpowiednio podobne (zakładamy, że są równe), to po wyeliminowaniu ich mamy:

$$I \simeq \frac{4I_{2n} - I_n}{3}$$

Jeśli ciąg otrzymanych przybliżeń jest monotoniczny, to znaczy, że funkcja nie zmienia bardzo gładkości, czyli założenie o stałości drugiej pochodnej jest mniej więcej dobre. Jeśli jednak ten ciąg nie jest monotoniczny, ekstrapolacja jest wątpliwa.

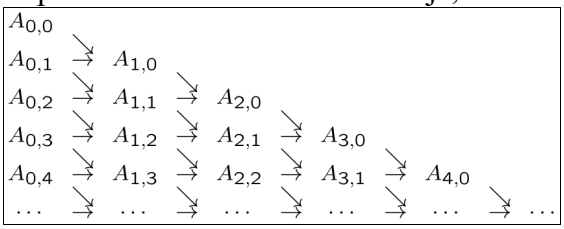
Metoda Romberga – możemy wyjść dalej poza ekstrapolację Richardsona. Korzystamy z faktu, że błąd metody trapezów zawiera jedynie parzyste potęgi średnicy przedziałów:

$$I = \int_a^b f(x)\,dx = h\left(\frac{1}{2}f_0 + f_1 + f_2 + \cdots + f_{N-1} + \frac{1}{2}f_N\right) + \sum_{j=1}^{\infty} \alpha_j h^{2j}$$

Niech $A_{o,k}$ oznacza przybliżenie całki uzyskane wzorem trapezów z 2^k podprzedziałami. Jeśli całka istnieje, to $\lim_{k\to\infty} A_{0,k} = I$ //może również zachodzić, gdy całki nie ma

Definiujemy:

$$A_{n,k} = \frac{1}{4^n - 1}\left(4^n A_{n-1,k+1} - A_{n-1,k}\right)$$



Ułożone po kolei elementy przypominają macierz trójkątną dolną (rysunek ↑), przy czym musimy pamiętać tylko ostatnio policzony wiersz. Na diagonali macierzy leżą wyrazy $A_{k,0}$, reszta wiersza to $A_{i,j}$, gdzie i to kolumna, a j to wiersz.
Ciąg $A_{0,k}$ można przekształcić w ciąg $A_{k,0}$, który jest szybciej zbieżny do tej samej wartości. Robimy to za pomocą specjalnej macierzy (trójkątnej dolnej).

Ogólne postępowanie:
Obliczamy $A_{0,k}$ za pomocą złożonej metody trapezów dla 2^k podprzedziałów. Aktualnym przybliżeniem całki jest $A_{k,0}$. Obliczamy $A_{0,k+1}$ korzystając ze złożonej metody trapezów dla 2^{k+1} podprzedziałów. Zapełniamy cały wiersz korzystając z powyższego wzoru. Kończymy, gdy A_k i A_{k+1} różnią się mniej niż zadana tolerancja (albo jest za dużo iteracji, czyli metoda jest rozbieżna).

Metoda Romberga, w zależności od przypadku, może być dużo szybsza lub nieco wolniejsza od trapezów złożonych.

Całki nieskończone liczymy w ten sposób, że wybieramy jakąś dużą stałą A i dzielimy całkę na 2 przedziały: [0, A] oraz [A, ∞). [0, A] liczymy numerycznie, a [A, ∞] analitycznie. Całka musi być oczywiście zbieżna oraz dobieramy A tak, by dla x > A zachodziło $|f(x)| \leqslant B \cdot g(x)$, gdzie g(x) jest funkcją, którą łatwo policzyć analitycznie.

Kwadratury adaptacyjne – jeśli funkcja, którą całkujemy posiada lokalne oscylacje, może być problem z dobraniem optymalnego kroku całkowania. Kwadratury adaptacyjne same lokalnie dobierają krok. W tym celu algorytm musi mieć dwa niezależne oszacowania całki, których różnica jest miarą popełnianego błędu. Szukamy przybliżenia:

$$\int_a^b f(x)\,dx \simeq I = \sum_{j=1}^{N} I_{[x_{j-1},x_j]}$$

gdzie sumujemy numeryczne przybliżenia całki w przedziałach. $\varepsilon_{[xj-1,\,xj]}$ oznacza błąd całkowania na danym przedziale, a τ to zadany błąd maksymalny.

$$\left|\mathcal{E}_{[x_{j-1},x_j]}\right| < \frac{x_j - x_{j-1}}{b-a}\tau$$

Chcemy oszacować całkę $\int_{\alpha}^{\beta} f(x)\,dx$ dla $a \leqslant \alpha < \beta \leqslant b$. Mamy aktualne przybliżenie na przedziale $I_{[\alpha,\,\beta]}$, mamy też I, czyli aktualną zakumulowaną sumę na tych przedziałach, które spełniły też warunek z τ. Liczymy:

$$I_{[\alpha,(\alpha+\beta)/2]} \simeq \int_{\alpha}^{(\alpha+\beta)/2} f(x)\,dx,\; I_{[(\alpha+\beta)/2,\beta]} \simeq \int_{(\alpha+\beta)/2}^{\beta} f(x)\,dx$$

Mając dwa oszacowania, czyli $I_{[\alpha,\,\beta]}$ oraz $I_{[\alpha,(\alpha+\beta)/2]} + I_{[(\alpha+\beta)/2,\beta]}$, szacujemy błąd $\varepsilon_{[xj-1,\,xj]}$.
Jeżeli błąd nie spełnia warunku z τ, prawy podprzedział [(α+β)/2, β] odkładamy na stos i powtarzamy procedurę dla lewego podprzedziału. *Jeśli spełnia oszacowanie, dodajemy obliczoną wartość do całki po przedziale [α,β] i zdejmujemy ze stosu kolejny przedział, dla którego powtarzamy procedurę.
Jeśli stos się przepełni, to znaczy, że algorytm zawiódł, albo zadaliśmy za dużą dokładność (wtedy zwiększamy τ).

Do algorytmu zwykle stosujemy kwadratury niskiego rzędu (trapezów, Simpsona). Tego przybliżenia całki używamy w kwadraturach adaptacyjnych (dla trapezów) w *:

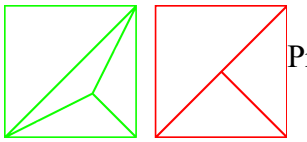
$$\int_{\alpha}^{\beta} f(x)\,dx \simeq I_{[\alpha,(\alpha+\beta)/2]} + I_{[(\alpha+\beta)/2,\beta]} + \frac{1}{3}\left(I_{[\alpha,(\alpha+\beta)/2]} + I_{[(\alpha+\beta)/2,\beta]} - I_{[\alpha,\beta]}\right.$$

Wynika on z pewnych różnych długich przekształceń wzoru metody trapezów.

Dla całek wielowymiarowych o wymiarze > 2, używamy metod Monte Carlo. Dla 2-wymiarowych, tradycyjna metoda jest zwykle lepsza.

Całkowanie po siatce prostokątnej – licząc całkę $I = \int_a^b dx \int_c^d dy\, f(x, y)$ po obszarze prostokątnym, korzystamy z metody iteracyjnej: $I = \int_a^b g(x)\,dx$, dla $g(x) = \int_c^d f(x, y)\,dy$. Stosujemy kwadratury jednowymiarowe, co wymaga N^2 obliczeń. Można też liczyć w odwrotnej kolejnośći.

Kwadratura adaptacyjna w dwóch wymiarach – stosujemy, gdy obszar całkowania nie jest prostokątem. Wtedy triangulujemy obszar, czyli pokrywamy go trójkątami w taki sposób, że trójkąty albo się nie stykają, albo mają wspólny wierzchołek, albo wspólną krawędź.



Przykład poprawnej i niepoprawnej triangulacji.

Wstępnej triangulacji możemy dokonać ręcznie. Natomiast dalej można tworzyć trójkąty ze środków ciężkości (średniej współrzędnych wierzchołków) już istniejących trójkątów, co tworzy 3 nowe trójkąty. W wyższych wymiarach, pokrywamy sympleksami (np. czworościanami w R^3).

Można zauważyć, że dla trójkątów $f(x_1, y_1)$ $f(x_2, y_2)$, $f(x_2, y_2)$ wyznaczają płaszczyznę. Objętość graniastosłupa ściętego utworzonego z tych płaszczyzn może służyć jako przybliżenie całki, co jest dwuwymiarowym odpowiednikiem metody trapezów. Zagęszczanie triangulacji wymaga tylko jednego dodatkowego policzenia wartości funkcji.

Postępowanie:
Mając trójkąt **Ω**, liczę przybliżenie całki ze "wzoru graniastosłupów": I_Ω. Dzielę trójkąt na 3 potomne i liczę przybliżenia całek: $I_{\Omega1}$, $I_{\Omega2}$, $I_{\Omega3}$. Miarą błędu jest: $\varepsilon_\Omega = I_\Omega - (I_{\Omega1} + I_{\Omega2} + I_{\Omega3})$.

$$|\mathcal{E}_\Omega| < \frac{S_\Omega}{S_D}\tau$$

Jeśli zachodzi taki warunek ←, gdzie S_Ω to powierzchnia trójkąta, a S_D to cała powierzchnia, za przybliżenie całki uznaję $I_{\Omega1} + I_{\Omega2} + I_{\Omega3}$. Zdejmuję kolejny trójkąt ze stosu i powtarzam. Jeśli warunek nie został spełniony, odkładam 2 trójkąty na stos i powtarzam procedurę dla trzeciego. Jeśli stos jest pusty i nie ma więcej trójkątów, to sukces. Porażka, gdy stos się przepełni.

Procedurę tą można stosować do obszarów niewielokątowych, zagęszczając i uzyskując coraz lepsze przybliżenie.

Równanie algebraiczne ma postać $f(x) = 0$. Zakładamy, że f jest ciągła i różniczkowalna ileś razy. Miejsce zerowe jest k-krotne, jeśli funkcja zeruje się tam z k-1 pochodnymi.

Metoda bisekcji – wybieramy sobie dwa punkty takie, że funkcja zeruje się między nimi. Punkt pośrodku tych dwóch traktujemy jako przybliżenie miejsca zerowego. Sprawdzamy w którym z dwóch przedziałów funkcja się zeruje i powtarzamy metodę na tym przedziale. Jeśli przybliżona wartość funkcji będzie mniejsza modułem od ε (jakiejś zadanej dokładności) to przerywamy. Zbieżność tej metody jest liniowa. Nie działa dla miejsc zerowych o parzystej krotnośći.

Metoda regula falsi – wybieramy dwa punky jak poprzednio i prowadzimy sieczną. Jako punkt trzeci wybieramy punkt przecięcia siecznej z OX: $x_3 = (f(x_1)x_2 - f(x_2)x_1) / (f(x_1 - x_2)$. Postępujemy jak w bisekcji.

Metoda siecznych – wybieramy dwa dowolne, różne punkty. Trzeci punkt to punkt przecięcia siecznej z OX, ale jako następne dwa punkty wybieramy zawsze dwa ostatnie. Metoda ta może być nieco szybciej zbieżna niż regula falsi, ale niekoniecznie musi być zbieżna do miejsca zerowego.

Możemy dokonać interpolacji funkcji odwrotnej. Jej wartość w zerze jest miejscem zerowym funkcji interpolowanej.

Metoda Newtona – kolejne iteracje wyglądają tak: $x_{n+1} = x_n - f(x_n) \div f'(x_n)$. Wynika to z zastosowania rozwinięcia Taylora: $f(x_0 + \delta) \simeq f(x\ 0\ ) + \delta \cdot f'(x_0)$ i żądania, żeby lewa strona rozwinięcia zniknęła. Metoda może być rozbieżna, gdy zastosowana na pewien przedział [a, b] wychodzi poza niego. Może również prowadzić do cykli. Pochodną w miarę możliwości powinniśmy znać analitycznie, inaczej metoda na ogół nie ma sensu.
Metoda ta jest tym bardziej efektywna, im bliżej miejsca zerowego zacznie. Dla miejsc o nieparzystej krotności ma zbieżność kwadratową, liniową dla parzystych. Można ją uogólnić na liczby zespolone, a baseny atrakcji tworzą często wtedy fraktale.

Jeśli iloraz w iteracji pomnożymy przez pewną stałą $\alpha \in (0, 1]$, to otrzymamy tłumioną metodę Newtona, i użycie jej na kilka kroków pozwala wyjść z wielocyklu.

Dla drugiej pochodnej, iteracje wyglądają tak: $$x_{n+1} = x_n - \frac{2f(x_n)}{f'(x_n) \pm \sqrt{[f'(x_n)]^2 - 2f(x_n)f''(x_n)}}$$

Znak dobieramy, by moduł mianownika był większy. Może prowadzić w iteraty zespolone dla rzeczywistych startów, w przeciwieństwie do normalnych iteracji.

Metoda Halleya – wygląda jak metoda Newtona, ale w mianowniku iteracji jest $\sqrt{|f(x)|}$. Po przekształceniu:

$$x_{n+1} = x_n - \frac{2f(x_n)f'(x_n)}{2[f'(x_n)]^2 - f(x_n)f''(x_n)}$$

Metody Newtona można użyć dla układów równań nieliniowych. Równanie takie ma postać:
$g_k(x_1,...x_n) = 0$, dla k = 1,...n. g jest funkcją $R^N \rightarrow R^N$.

Iteracja metody Newtona ma postać: $x_{k+1} = x_k - J^{-1}(x_k)g(x_k)$ //J to jakobian funkcji g
Jakobian trzeba liczyć w każdym kroku, ale można to robić co kilka, dla dużych macierzy.

Minimalizacja funkcji w celu znalezenia rozwiązań układu równań to zły pomysł. Chyba, że połączy się ją z metodą Newtona:

1. $w = 1$. Oblicz δx.
2. $x_{test} = x_i + w\,\delta x$.
3. Jeśli $G(x_{test}) < G(x_i)$, to
   (a) $x_{i+1} = x_{test}$
   (b) *goto* 1
4. Jeśli $G(x_{test}) > G(x_i)$, to
   (a) $w \rightarrow w/2$
   (b) *goto* 2

$\delta x = -J^{-1}g$, przerywamy, gdy ω jest za małe. Jest to forma tłumionej metody Newtona. Jeśli znalezione minimum nie jest globalnym (tym co nas interesuje), to zaczynamy gdzieś indziej.

Metoda Broydena (a.k.a. wielowymiarowa metoda siecznych) – dobra, gdy nie znamy pochodnych, albo liczenie jakobianu jest zbyt kosztowne. $x_{i+1} = x_i - B_i^{-1}g(x_i)$, gdzie B jest przybliżeniem jakobianu, liczonym:

$$\mathbf{B}_{i+1} = \mathbf{B}_i + \frac{(\Delta \mathbf{g}_i - \mathbf{B}_i \Delta \mathbf{x}_i)(\Delta \mathbf{x}_i)^T}{(\Delta \mathbf{x}_i)^T \Delta \mathbf{x}_i}$$

natomiast $\Delta x = x_{i+1}-x_i$, a $\Delta g_i = g(x_{i+1}) - g(x_i)$.
Do obliczenia $B_{i+1}^{-1}g(x_{i+1})$ można skorzystać ze wzoru Shermana-Morrisona. Metoda wymaga podania B_1 (np. $J(x_1)$) oraz x_1.

Gdy szukamy miejsc zerowych wielomianu, interesują nas wszystkie.
Wartości wielomianu najlepiej liczyć algorytmem Hornera: $P = a_n$, k = n
{ k--; $P = P \cdot z + a_k$ dopóki k>0 } //z to liczba dla której liczymy wielomian W(z)

$$|\varepsilon| \simeq \frac{\left|\sum_{k=0}^{n} \delta_k \tilde{z}_0^k\right|}{\left|\tilde{P}_n'(\tilde{z}_0)\right|}$$

Oszacowanie wpływu zaburzeń współczynników na zaburzenie miejsca zerowego (zwykle nie znamy dokładnych współczynników). $\tilde{P}_n$ oznacza zaburzony wielomian.
$//\tilde{P}_n(\tilde{z}_0) = 0 \Rightarrow \tilde{z}_o = z_0 + \varepsilon$ oraz $\tilde{a}_k = a_k + \delta_k$, i $\varepsilon, \delta_k << 0$

Zagadnienie to jest źle uwarunkowane. Przykład Wilkinsona: W(z) = (z+1)...(z+20). Jeśli zaburzymy jeden współczynnik (przy wysokiej potędze), miejsce zerowe zaburzy się 7 rzędów wielkości bardziej.

Powyższe oszacowanie załamuje się dla wielokrotnych miejsc zerowych, które w szczególności mogą się rozszczepiać. Żeby zapobiec takim różnym niedokładnościom, trzeba użyć odpowiedniej metody do szukania miejsc zerowych wielomianów oraz zmniejszać stopień wielomianu po każdym znalezionym miejscu.

Metoda Laguerre'a – $P_n(z)$ to wielomian stopnia n. Mamy taką iterację:

$$z_{i+1} = z_i - \frac{n\,P_n(z_i)}{P_n'(z_i) \pm \sqrt{(n-1)\left((n-1)\left[P_n'(z_i)\right]^2 - n\,P_n(z_i)\,P_n''(z_i)\right)}}$$

Znak w mianowniku wybieramy tak, by moduł mianownika był największy. Metoda jest zbieżna sześciennie do wszystkich pojedynczych miejsc zerowych, liniowo do wielokrotnych.

Może się zdarzyć, że trafimy drugi raz na to samo miejsce zerowe. Żeby tego uniknąć, znajdujemy faktoryzację: $P_n(z) = (z - z_1)P_{n-1}(z)$, czyli obniżamy stopień wielomianu. Jednak ze względu na to, że zaburzenia współczynników powodują duże rozbieżnośći, używamy miejsca zerowego $\tilde{z}_2$ znalezionego w wielomianie P_{n-1} do metody Laguerr'a w pierwotnym wielomianie (wygładzamy). Następnie znajdujemy faktoryzację P_{n-2} itd.

Uwagi: jeśli wielomian ma rzeczywiste współczynniki i znajdziemy zespolone miejsce zerowe, to automatycznie istnieje drugie sprzężone z nim miejsce. Jeżeli wielomian ma niewielkie, całkowite współczynniki, można wykorzystać znane twierdzenia do znalezienia wymiernych miejsc zerowych. Jeśli faktoryzując dojdziemy do wielomianu stopnia 2, liczymy jego pierwiastki ze znanego wzoru i je również wygładzamy.

Macierz stowarzyszona – może być użyta do szukania miejsc zerowych wielomianu. Na diagonali same jedynki, współczynniki wielomianu zajmują ostatni rząd, z zerowym po lewej i a_{n-1} po prawej (wszystkie mają też odwrotny znak). Ma ona równanie charakterystyczne:
$x^n + a_{n-1}x_{n-1}^{n-1} + a_{n-2}x_{n-2}^{n-2} + \ldots + a_1x + a_0 = 0$. W tym przypadku, a_n równy jest 1, co nie ma wpływu na miejsca zerowe.

Minimalizacja to szukanie minimum (jednego lub więcej) jakiejś funkcji. Jeśli szukamy maksimum funkcji f(x), to tak jak szukanie minimum g(x) = -f(x). Jesteśmy w stanie zlokalizować minimum z dokładnością równą co najwyżej pierwiastkowi z precyzji obliczeń. Szukanie przerywamy, gdy zmiana kolejnych wartości funkcji jest mniejsza od zadanej tolerancji.

Mówimy, że punkty a, b, c ograniczają minimum, jeśli $f(a) > f(b)$ oraz $f(c) > f(b)$.
Wybieramy dwa punkty. Najczęściej wyznaczają one kierunek spadku funkcji. W takim wypadku, bierzemy trzeci punkt, który jest oddalony w tym kierunku o jakiś krok. Jeśli powyższy warunek nie jest spełniony, podwajamy krok. Trzeba założyć maksymalną liczbę kroku (albo rozmiar), żeby nie iść sobie w nieskończoność.
Jak mamy już 3 punkty, które spełniają warunek, wybieramy jakiś czwarty pomiędzy nimi i posuwamy się w dół, zmieniając odpowiedne punkty (i obliczone wartości funcji, żeby nie liczyć znowu). Warunkiem skończenia iteracji jest: $|c - a| < \tau(|b| + |d|)$.

Metoda złotego podziału – minimum szukamy w większym z dwóch przedziałów: [a, b], [b, c]. Żeby zminimalizować szansę na minimum w mniejszym przedziale, przedziały powinny spełniać złotą proporcję. Jeśli przedział [a,b] jest dłuższy to: $|b - a| \div |c - a| = |c - b| \div |b - a|$. Żeby to zapewnić, robimy: {jeśli $|b - a| > |c - b|$ to $d = a + \omega \cdot |b - a|$, inaczej $d = b + \omega \cdot |c - b|$}. $\omega \simeq 0.381966$. Metoda ta jest zbieżna liniowo.

Interpolacja paraboliczna i metoda Brenta – stosujemy wzór:

$$d = \frac{1}{2} \cdot \frac{a^2(f_c - f_b) + b^2(f_a - f_c) + c^2(f_b - f_a)}{a(f_c - f_b) + b(f_a - f_c) + c(f_b - f_a)}$$

//$f_a \equiv f(a)$

Czyli tworzymy parabolę między trzema punktami i bierzemy jej minimum jako d. Mogą pojawić się kłopoty, gdy początkowo funkcja nie przypomina paraboli.
W takim razie odrzucamy d, gdy nie leży w przedziale [a, c] lub rozmiar przedziału nie jest mniejszy niż połowa przedziału w przedostatniej iteracji. Jeśli odrzuciliśmy punkt, dokonujemy bisekcji na [a, c] (bisekcja, bo jest prostsza od złotego podziału, a daje podobny rezultat).

Zamiast metody Brenta, można skorzystać z pochodnych: $d = \frac{af_c' - cf_a'}{f_c' - f_a'}$
Jest to jednak na ogół mniej efektywne.

Jeszcze gorszą metodą jest wykorzystanie interpolacji Hermite'a: robimy wielomian 3-ego stopnia, który zgadza się z badaną funkcją i jako d bierzemy jego minimum.

Żeby znaleźć minima funkcji wielu zmiennych, obieramy jakiś punkt startowy x_k, który jest aktualnym przybliżeniem minimum oraz kierunek p_k. Konstuujemy funkcję: $g(\alpha) = f(x_k + \alpha p_k)$. Znajdujemy minimum funkcji g, która jest funkcją jednej zmiennej. Wtedy $x_{k+1} = x_k + \alpha_{min} p_k$.
Powtarzamy, dopóki nie otrzymamy porządanej dokładności. Problem sprowadza się do odpowiedniego wyboru kierunku poszukiwań.

Kierunek powinniśmy wybierać inaczej, gdy jesteśmy blisko minimum, a inaczej gdy daleko.

Minimalizacja po współrzędnych – kolejnymi kierunkami są kierunki równoległe do kolejnych osi układu współrzędnych. Zły pomysł: za dużo kroków i są one niezwiązane z kształtem funkcji. To nie może być optymalne.

Metoda najszybszego spadku – kierujemy się minus gradientem funkcji. Jednak szukanie minimów kierunkowych jest dość kosztowne, więc kierujemy się tylko najszybszym spadkiem funkcji i tylko do okolic minimum. Używanie tej metody w okolicach minimum prowadzi do małych i licznych kroków.

Waunek na to, by f osiągała minimum kierunkowe (czyli minimum g):

$$\frac{dg_k}{d\alpha} = \sum_i \frac{\partial f}{\partial x_i}(\mathbf{p}_k)_i = \left(\nabla f|_{\mathbf{x}=\mathbf{x}_{\min}}\right)^T \mathbf{p}_k = 0$$

Czyli w minimum, gradient jest prostopadły do kierunku poszukiwań (styczny do poziomic).

Rozwinięcie minimalizowanej funkcji w szereg Taylora: $f(x) \simeq {}^1/_2 x^T A x - b^T x + f_o$. A to hessjan obliczany w minimum. Macierz ta jest dodatnio określona (bo minimum) i może być symetryczna, gdy funkcja jest gładka. Czyli w pobliżu minimum, funkcja zachowuje się podobnie do formy kwadratowej.

Gradienty sprzężone – przy rozwinięciu w szereg Taylora, gradient funkcji w x_k wynosi: $\nabla f|_{x = xk} = Ax_k - b$. Zmiana gradientu w kierunku poszukiwań p_k wynosi: $\delta(\nabla f) = \alpha A p_{k+1}$. Żeby nie naruszyć żadnego z poprzednio wyliczonych minimów kierunkowych, zmiana gradientu musi być prostopadła do wszystkich poprzednich kierunków poszukiwań: $p_i^T A p_j = 0$, $i > j$.

Algorytm: Zaczynamy w x_1. Bierzemy $r_1 = p_1 = -\nabla f|_{x1}$. Z punktu x_k minimalizujemy kierunkowo w kierunku p_k, otrzymując x_{k+1}. Obliczamy $r_{k+1} = -\nabla f|_{xk+1}$. Liczymy:

$$\beta = \frac{\mathbf{r}_{k+1}^T \mathbf{r}_{k+1}}{\mathbf{r}_k^T \mathbf{r}_k}$$

W końcu: $p_{k+1} = r_{k+1} + \beta p_k$. Powtarzamy.

Wektor r_{k+1} jest tutaj tym samym wektorem, który otrzymalibyśmy w algebraicznej metodzie gradientów. Dzięki temu, nie potrzebujemy hessjanu do obliczeń, bo zastępujemy go minimalizacją kierunkową, a resztę kroków metodą algebraiczną. Alternatywna postać równania, która może przyśpieszyć obliczenia, gdy grozi stagnacja:

$$\beta = \frac{\mathbf{r}_{k+1}^T (\mathbf{r}_{k+1} - \mathbf{r}_k)}{\mathbf{r}_k^T \mathbf{r}_k}$$

Metoda zmiennej metryki – ponieważ szukając minimum, mamy rozwiązać równanie $\nabla f = 0$, to możemy użyć takiego rozwiązania (opartego na metodzie Newtona):
$x_{i+1} = x_i - H_i^{-1} \nabla f|_{x_i}$, dla H będącego macierzą drugich pochodnych cząstkowych w x_i. Żeby dochodziło do zmniejszania się wartości funkcji, musi dodatkowo zachodzić: $-(x_{i+1} - x_i)^T H_i (x_{i+1} - x_i) < 0$ (jeśli H jest dodatnio określone, mamy to spełnione automatycznie). Zamiast hessjanu, w tej metodzie używamy przybliżenia hessjanu, które jest zbieżne do tego prawdziwego, w minimum.

Algorytm: H_0 to jakaś macierz symetryczna, dodatnio określona. Rozwiązujemy $H_k p_k = -\nabla f|_{x_k}$. Minimalizujemy kierunkowo $f(x_k + \alpha p_k)$, a α_k to znalezione minimum kierunkowe. $x_{k+1} = x_k + \alpha_k x_k$. Liczymy $y_k = \nabla f|_{x_{k+1}} - \nabla f|_{x_k}$.
Na koniec liczymy H_{k+1}, np. ze wzoru BFGS:

$$\mathbf{H}_{k+1} = \mathbf{H}_k + \frac{\mathbf{y}_k \mathbf{y}_k^T}{\alpha_k \mathbf{p}_k^T \mathbf{y}_k} - \frac{\nabla f|_{\mathbf{x}_k} \left(\nabla f|_{\mathbf{x}_k}\right)^T}{\mathbf{p}_k^T \mathbf{H}_k \mathbf{p}_k}$$

Kończymy, gdy $|p_k| < \varepsilon$ (zadanej tolerancji).
W tej metodzie, jeśli H_0 jest stymetrzyczne i dodatnio określone, to takie pozostaje. Metoda sprawdza się dla dużej liczby wymiarów.

Dwie ostatnie metody są dobre, gdy jesteśmy blisko minimum. Gdy zaczynamy daleko, to są wolne. Dzieje się tak dlatego, że daleko do minimum, funkcja nie jest podobna do formy kwadratowej.

Metoda Levenberga-Marquardta – modyfikujemy hessjan (żeby był dodatnio określony również z dala od minimum):

$$\widetilde{\mathbf{H}}_{ii} = (1+\lambda)\frac{\partial^2 f}{\partial x_i^2}, \qquad \widetilde{\mathbf{H}}_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}, \quad i \neq j$$

$\lambda \geqslant 0$. Zaczynamy z jakimś niewielkim λ, np. 2^{-10}. x_i to aktualne przybliżenie. Liczymy po kolei: $\nabla f(x_i)$, $\tilde{H}(x_i)$. $x_{test} = x_i - \tilde{H}^{-1}(x_i)\nabla f(x_i)$.
Jeżeli $f(x_{test}) > f(x_i)$, to zwiększamy λ 8 (lub nie 8) razy i liczymy znowu hessjan.
Inaczej, zmniejszamy λ 8 razy, $x_{i+1} = x_{test}$ i liczymy znowu gradient.

Daleko od minimum znowu nie minializujemy, a podążamy za spadkiem. Jeśli λ jest zbyt duże (ustalamy jakieś maksymalne), to znaczy, że jesteśmy poza basenem atrakcji i algorytm zawodzi. Jeśli λ jest bardzo małe, możemy przerzucić się na gradienty sprzężone albo zmienną metrykę. Gdy wybierzemy to drugie, aktualne przybliżenie hessjanu może służyć za początkowe w tamtej metodzie.
Jeśli chcemy przyśpieszyć algorytm, możemy zrezygnować na kilka kroków ze zmniejszana λ i przeliczania H na spadku. Daleko od minimum, ten algorytm zachowuje się jak metoda najszybszego spadku.

Metoda Powella – dobra, gdy nie możemy policzyć pochodnych (a więc i gradientów). Zaczynamy od minimalizacji po współrzędnych: $\{p_i\}_{i=1}^N = \{e_i\}_{i=1}^N$. Znajdujemy się w jakimś x_0, minimalizujemy wzdłuż kierunków p_i, osiągając kolejne x_i. Od i do N-1, $p_i = p_{i+1}$. $p_N = x_N - x_0$. Minimalizujemy wzdłuż p_N i nowy punkt oznaczamy jako x_0, i robimy następną iterację. Jeśli funkcja jest rozwijalna w szereg Taylora, p_i stają się sprzężone.

Wszystkie powyższe algorytmy są nieprzydatne dla minimalizacji globalnej. Nie sprawdzają się dla niej również metody zachłanne (sprawdzanie dużej ilośći punktów) oraz losujące punkty do sprawdzania, szczególnie w wielu wymiarach są zbyt złożone.
Algorytmy do szukania minimów globalnych idą zawsze w dół, więc jak wejdą do basenu atrakcji minimum, to mogą nie wyjść (a nie musi to być minimum globalne). Więc robimy tak, że losowo możemy zrobić "zły" krok pod górkę (wyższe kroki są rzadsze).

Algorytm Monte Carlo – mamy funkcję $f: R^N \to R$. Startujemy z jakiegoś x_o, liczymy $f_0 = f(x_0)$. $x_{min} = x_0$, $f_{min} = f_0$. $\sigma, T > 0$ są parametrami. Zaczynamy iterację: $\tilde{x} = x_k + \sigma\xi_k$ (ξ to N wymiarowa zmienna losowa gaussowska). $\tilde{f} = f(\tilde{x})$. Jeżeli $\tilde{f} < f_k$, to akceptujemy położenie. Jeżeli $\tilde{f} < f_{min}$, to $x_{min} = \tilde{x}$ oraz $f_{min} = \tilde{f}$. Jeśli $\tilde{f} > f_k$, to losujemy z o rozkładzie jednorodnym i jeśli $z < \exp(-(\tilde{f} - f_k)/T)$, to akceptujemy nowe położenie. //$\exp(-(\tilde{f} - f_k)/T)$ to kryterium boltzmannowskie
Nie ma kryterium stopu, zadajemy M iteracji i tyle robimy. Zamiast brać duże M, kilkakrotnie losujemy punkt startowy i robimy niewielką liczbę iteracji. Nie ma gwarancji, że wylosowane punkty same znajdą się w basenach atrakcji.

Kombinacja Gauss-Boltzmann powoduje, że im większ paramter T, tym łatwiej wydostać się funkcji z basenu atrakcji (czas na to potrzebny to ~exp(wysokość bariery / T). Parametry dobieramy np. metodą prób i błędów, T nie powinno być za wysokie. Metoda działa również dla problemów dyskretnych.
Algorytm ten ma charakter lokalny, bo nie wprowadza znacznych zmian w sposób sensowny i może być wolny.

Algorytmy genetyczne – opierają się na zasadzie survival of the fittest. Mamy jakąś populację osobników (dużo mniej niż możliwych stanów) i każdy reprezentowany jest przez chromosom. Dla każdego chromosomu można wyliczyć wartość funkcji dopasowania. Każdemu z nich przyporządkowujemy wycinek koła, proporcjonalny do jego wartości funkcji: $p_i = f_j / \Sigma_j f_j$ (tzw. metoda ruletki).

Wybieramy następnie dwoje "rodziców", którzy stworzą chromosom do następnej iteracji. Dzielimy chromosomy rodziców (najczęściej po połowie) i tworzymy nowy chromosom. Następnie z niewielkim prawdopodobieństwem, możemy zmienić losową z wartośći w chromosomie (zmutować). Tworzymy tyle potomków ile ma wynosić cała populacja, a operację tą można zrównoleglić.

Populację startową wybieramy losowo z puli dostępnych stanów i zadajemy jakąś maksymalną ilość epok (pokoleń). Jeśli wyczerpią się epoki, zmiany są zbyt małe, albo populacja jest w pobliżu jakiegoś stanu, przerywamy. Zapamiętujemy chromosom, który miał najlepszą wartość funkcji dopasowania. //np. możemy mieć dodatnią funkcję i jej maksimum będzie najlepsze

Particle Swarm Optimization – mamy pewne niezbyt liczne stado. Każdemu z osobników przyporządkowujemy położenie x_i oraz prędkość v_i. Początkowe położenia i prędkości są losowe (prędkości nie za duże). Dla każdego członka stada zapamiętujemy najmniejszą wartość funkcji dopasowania $f_{min,\,i}$ oraz odpowiadające jej położenie $x_{min,\,i}$. Zapamiętujemy też globalną najmniejszą wartość funkcji dopasowania f_{glob} oraz jej położenie. $0 < \omega < 1$; $a,b > 0$ są parametrami.

Stado eksploruje przestrzeń. Każdy członek ma tendencję do lecenia jak wcześniej, orientowania się w stronę swojej najlepszej wartości oraz orientowania w stronę najlepszej globalnej. Sekwencyjnie dla wszystkich wykonujemy: losujemy liczby p, q na przedziale [0, 1]; $v_{i+1} = \omega v_i + a \cdot p \cdot (x_{min,i} - x_i) + b \cdot q \cdot (x_{glob} - x_i)$; $x_{i+1} = x_i + v_{i+1}$; $f_{i+1} = f(x_{i+1})$; aktualizujemy wartości $f,x_{min,i}$ oraz f,x_{glob} na podstawie f,x_{i+1}.

Tendencja do podążania w stronę minimum znalezionego przez danego członka odpowiada za podążanie w dół, a w stronę globalnego, za inteligencję roju. Wadą algorytmu jest, że nie zwraca uwagi na kształt i zmienność funkcji, ale zaletą jest, że nie trzeba liczyć gradientów. Parametry można dobierać na podstawie wyników minimalizacji podobnych, ale prostszych problemów (metaoptymalizacja). Jeśli $\omega > 1$, to prędkości wybuchają. Żeby uniknąć sytuacji, gdy stado skupia się wokół jednego minimum (zdarza się dla wielu wymiarów), robimy kilka niezależnych stad.

Aproksymacja może być: punktowa – szukamy znanej funkcji oraz jej parametrów, która będzie przebiegać możliwie najbliżej danych punktów, oraz może być ciągła – mamy jakąś funkcję, której obliczanie jest trudne i szukamy podobnej, ale łatwiejszej.

Interpolacja to skrajny przypadek aproksymacji punktowej. W aproksymacji chcemy mieć prostą funkcję, która nie musi przechodzić przez wszystkie punkty. Zależność między x i y powinna być y = y(x), ale problem jest z błędami pomiaru.

Liniowe zagadnienie najmniejszych kwadratów – każdej zmierzonej wartości y_i odpowiada wartość teoretyczna $\tilde{y}_i$, którą zmienna y "powinna" przybrać dla x. Przypuśćmy, że $\tilde{y}$ jest kombinacją liniową znanych funkcji: $\tilde{y}_i = a_1 \cdot f_1(x_1) + ... + a_s \cdot f_s(x_s)$. Zespół wszystkich teoretycznych wartości możemy przedstawić jako $\tilde{y} = Ap$, gdzie A jest macierzą funkcji $f_i x_j$, z arumentami w rzędach i funkcjami w kolumnach, a p to wektor współczynników.
Problem, który chcemy rozwiązać to znalezenie najlepszego wektora p.
Przykładowa macierz dla n puntków pomiarowych i wielomianu 2 stopnia, który chcemy dopasować:

$$\tilde{y} = \begin{bmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ \dots & \dots & \dots \\ x_n^2 & x_n & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

Dopasowanie danych do wielomianu ustalonego stopnia jest zagadnieniem liniowym. Różnica między wartością teoretyczną, a pomiarową jest spowodowana błędem pomiaru: $y_i - \tilde{y}_i = \xi_i$. Przypuścy, że liczby ξ są zmiennymi losowymi z rozkładu Gaussa. Wektor wszystkich błędów pomiarowych oznaczamy $\xi = [\xi_1, \dots, \xi_n]^T$. Przyjmijmy też, że łącznie wszystkie błędy tworzą rozkład Gaussa o macierzy kowariancji $G = \langle \xi\xi^T \rangle$, gdzie $\langle ... \rangle$ oznacza średniowanie po realizacjach zmienych losowych. G jest symetryczna i dodatnio określona. Dla takiej macierzy, estymator największej wiarygodności wynosi: $Q = {}^1/_2\, \xi^T G^{-1} \xi$ i z pewnością posiada minimum. Obecność odwrotności G oznacza, że pomiary z większym błędem dają mniejszy wkład do Q.

W tym zagadnieniu, minimalizowana funkcja jest formą kwadratową w parametrach (zależy liniowo od parametrów, nie argumentów) i wiemy, że posiada jednoznaczne minimum. Aby znaleźć estymator, trzeba znaleźć taki wektor p, że forma kwadratowa Q osiąga minimum. Można to zrobić metodą zmiennej metryki albo gradientów sprzężonych, lub formalnie rozwiązując $\nabla Q = 0$. Otrzymujemy: $A^T G^{-1} A p = A^T G^{-1} y$. Członu $A^T G^{-1}$ nie da się uprościć, bo to nie jest macierz kwadratowa. Samo równanie jest dobrze określone, bo $A^T G^{-1} A$ jest symetrzyczna i dodatnio określona.

$y = Ap^* + \xi$, gdzie p^* jest zbudowany z "prawdziwych" współczynników a_i. Wtedy: $A^T G^{-1} A(p - p^*) = A^T G^{-1} \xi$, czyli $\langle p \rangle = p^*$, $\langle \xi \rangle = 0$. Obliczone estymatory są przybliżeniem "prawdziwych" wartośći parametrów. Macierz kowariancji estymatorów wynosi: $Cp = \langle (p - p*)(p - p*)^T \rangle$, co po przekształceniach daje: $\langle A^T G^{-1} A \rangle^{-1}$.

Możemy rozwiązać, aby $y_i = \tilde{y}_i$ było ściśle spełnione, czyli musimy rozwiązać układ równań $Ap = y$. Jest on nadokreślony i nie ma ścisłego rozwiązania, ale możemy znaleźć optymalne, w sensie najmniejszych kwadratów, rozwiązadnie za pomocą SVD. Jeśli pomiary są nieskorelowane, rozwiązanie uzyskane w ten sposób jest równoważne minimalizacji formy kwadratowej Q lub rozwiązaniu $\nabla Q = 0$. Nadoreślony układ $G^{-1}Ap = G^{-1}y$ również daje równoważne rozwiązanie z SVD (przy uwzględnieniu wag pomiarów).

Jeśli pomiary są nieskorelowane, Q upraszcza się do:

$$Q = \sum_{i=1}^{n} \frac{(a_1 f_1(x_i) + a_2 f_2(x_i) + \cdots + a_s f_s(x_i) - y_i)^2}{\sigma_i^2}$$

A macierz kowariancji estymatorów upraszcza się do: $Cp = \sigma^2 (A^T A)^{-1}$.

Kryterium Akaike – pozwala zbalansować najlepsze dopasowanie z najmniejszą liczbą parametrów.
$AIC = \ln Q + 2s / N$, należy to zminimalizować, dla s będącego liczbą parametrów, a N liczbą punktów. //AIC to skrót od Akaike Information Criterion
Kryterium to nagradza za lepsze dopasowanie i karze za dużą liczbę parametrów.

Nieliniowe zagadnienie najmniejszych kwadratów – kiedy dopasowana zależność teoretyczna zależy od danych nieliniowo: $\tilde{y}_i = f(x_i; p)$, dla p będącego wektorem parametów. f jest znaną funkcją o nieznanych parametrach, np.

$$y(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x - \bar{x})^2}{2\sigma^2}\right)$$

//parametrami są $\bar{x}$ oraz σ^2, widać, że zależność jest nieliniowa

Q do minimalizacji jest podobne, ale $\xi_i = y_i - (x_i; p)$. Dla najczęstszego przypadku nieskorelowanych danych, Q redukuje się do: $Q = \text{const}\ {}^1/_2 \Sigma_i (y_i - f(x_i; p))^2$. Jednak w nieliniowym zagadnieniu, Q nie jest formą kwadratową w parametrach. Q jest dodatnio określona, więc posiada minimum, ale niekoniecznie jedno. Należy je znaleźć numerycznie, metodą Levenberga-Marquardta, chyba, że $\nabla_p Q = 0$ można wyjątkowo łatwo rozwiązać.

Pseudolinearyzacja – stosuje się ją czasami do minimalizacji Q. Jeśli p_n jest aktualnym przybliżeniem p, zakładamy, że prawdziwe wartości są małą poprawką δp w stosunku do p_n. $y_i = f(x_i; p_n + \delta p)$ rozwijamy w szereg Taylora, otrzymując $f(x_i ; p_n) + [\nabla_p f \mid p_n]^T \delta p$. Funkcja: $Q = {}^1/_2 \Sigma_i (y_i - f(x_i; p_n) - [\nabla_p f \mid p_n]^T \delta p)^2$ jest formą kwadratową. Po znalezieniu minimum δp, podstawiamy: $p_{n+1} = p_n + \delta p_{min}$ i powtarzamy.

Działa ona dość dobrze dla nieliniowej metody najmniejszych kwadratów, ale nie należy jej polecać jako ogólnej metody minimalizacji. Niewątpliwym jej zastosowaniem jest, że po znalezieniu ostatecznych wartośći minimalizujących Q, za pomocą pseudolinearyzacji wokół tego punktu, znajdujemy macierz kowariancji estymatorów będącą charaktestystyką liniową.

Przybliżenie Padé – jedno z zagadnień aproksymacji ciągłej. Przypuśćmy, że znamy przybliżenie g(z) oraz jej pochodnych do rzędu N w z = 0 (jeśli w innym punkcie, to przesuwamy). Chcemy skonstruować jej przybliżenie w pewnym przedziale, by zgadzało się z nią oraz z jej pochodnymi w 0. Najprościej jest skonstruować rozwinięcie Maclaurina, ale ono jest wielomianem i załamuje się w niewielkiej nawet odległości od zera.

Lepiej użyć przybliżeń wymiernych:

$$R_{mk}(x) = \frac{P_m(x)}{Q_k(x)} = \frac{\sum_{j=0}^{m} a_j x^j}{\sum_{j=0}^{k} b_j x^j}$$

Dla $b_0 = 0$, a szeregiem Maclaurina aproksymowanej funkcji będzie:

$$g(x) = \sum_{j=0}^{\infty} c_j x^j$$

Przyjmijmy, że m + k + 1 = N + 1, czyli tyle, ile wyrazów zawiera szereg Maclaurina funkcji g(x) do rzędu N. Liczymy:

$$g(x) - R_{mk}(x) = \frac{\left(\sum_{j=0}^{\infty} c_j x^j\right)\left(\sum_{j=0}^{k} b_j x^j\right) - \sum_{j=0}^{m} a_j x^j}{\sum_{j=0}^{k} b_j x^j}$$

g(x) będzie się zgadzać z przybliżeniem wyżej, jeśli najniższy nieznikający wyraz sumy w liczniku będzie się zgadzać z X^{N+1}.

Otrzymujemy warunki, które są układem N+1 równań liniowych na N+1 współczynników a_r, b_j:

$$\sum_{j=0}^{k} c_{N-s-j} b_j = 0 \quad s = 0, 1, \ldots, N-m-1;\ c_j = 0 \text{ dla } j < 0$$

$$\sum_{j=0}^{r} c_{r-j} b_j = a_r \quad r = 0, 1, \ldots, m;\ b_j = 0 \text{ dla } j > k$$

Przybliżenie Czebyszewa – zamiast R_{mk}, lepszym przybliżeniem są wielomiany Czebyszewa:

$$C_{mk}(x) = \frac{\sum_{j=0}^{m} a_j T_j(x)}{\sum_{j=0}^{k} b_j T_j(x)}$$

//$T_j(x) = \cos(j \arccos x)$ dla $x \in (-1, 1)$

Po analitycznym przedłużeniu, $T_j(x)$ jest poza przedziałem (-1, 1) wielomianem stopnia j. Jeśli g(x) posiada rozwinięcie Czebyszewa:

$$g(x) = \frac{1}{2} c_0 + \sum_{j=1}^{\infty} c_j T_j(x)$$

To jego współczynniki obliczamy tak:

$$c_j = \frac{2}{\pi} \int_{-1}^{1} \frac{g(x) T_j(x)}{\sqrt{1 - x^2}} dx$$

Liczy się to kwadraturami Gaussa-Czebyszewa, których nie trzeba znać, bo nie były omówione.