|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| METODO | IF (metodo non applicabile) | CONDIZIONI DEL WHILE | k-esimo ITERATO | AGGIORNAMENTI |
| **Bisezione** | sign(fa) \* sign (fb) >= 0 | abs (b – a) > tolx | xk = a + (b+a) / 2 | Intervallo:  a = xk, fa = fxk  b = xk, fb = fxk |
| **Regula Falsi** | sign(fa) \* sign (fb) >= 0 | it < maxit and abs (b – a) > tolx and abs (fb -f | xk = a – fa \* (b -a) / (b – a) | Intervallo:  a = xk, fa = fxk  b = xk, fb = fxk |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| METODO | CONDIZIONI IF | fx0 | d | CONDIZIONI DEL WHILE | x1 |
| **Corde** | non ci sono | fname(x0) | fx0 / m | it < nmax  and abs(fx1) >= tolf  and abs(d) >= tolx\*abs(x1) | x0 - d |
| **Secanti** | non ci sono | fxm1=fname(xm1)  fx0=fname(x0) | fx0\*(x0-xm1) / (fx0-fxm1) | Già scritte dalla prof | x0 - d |
| **Newton** | abs (fpname(x0)) <= np.spacing(1) | fname(x0) | d = fx0 / fpname(x0) | it < nmax and abs(d) >= tolx \* abs(x1) and abs(fx1) >= tolf | x0 - d |
| **Newton modificato** | abs(fpname(x0) ) <= np.spacing(1) | fname(x0) | fx0 / fpname(x0) | it < nmax and abs(d) >= tolx \* abs(x1) and abs(fx1) >= tolf | x0 – m\*d |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| METODO | if | s | x1 | while |
| **Newton-Raphson** | npl.det(matjac) == 0 | -npl.solve (matjac, fun(x0)) | x0 + s | it <= nmax  and npl.norm (fx1, 1) >= tolf  and npl.norm (s, 1) >= tolx \* npl.norm (x1, 1) |
| **Newton Shamanski**  (jacobiano rivalutato ogni m iterazioni => matjac = jac(x0) ) | npl.det(matjac) == 0 | -npl.solve (matjac, fun(x0)) | x0 + s | it <= nmax  and npl.norm (fx1, 1) >= tolf  and np.linalg.norm (s, 1) >= tolx \* np.linalg.norm (x1, 1) |
| **Newton corde**  (jacobiano fisso) | npl.det(matjac) == 0 | -npl.solve (matjac, fun(x0)) | x0 + s | it <= nmax  and npl.norm (fx1, 1) >= tolf  and npl.norm (s, 1) >= tolx \* npl.norm (x1, 1) |
| **Newton minimo** (derivate parziali ed hessiano SENZA SIMPY) | 1. Calcolo hessiano con Hess(x0)  2. if npl.det(matHess) == 0: | - npl.solve (matHess, gradiente(x0)) | x0 + s | it <= nmax  and np.linalg.norm (grad\_fx1, 1) >= tolf  and npl.norm (s, 1) >= tolx \* np.linalg.norm (x1, 1) |
| **Newton minimo** (derivate parziali ed hessiano CON SIMPY) |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| METODI ITERATIVI | A = D + E + F  D = np.diag(d)  E = np.tril(A, -1)  F = np.triu(A, 1) | RELAZIONI DA RICORDARE | T = invM @ b | x = (T @ x0) + (invM @ b) |
| **Jacobi** | M - N | M = D,  N = - (E + F) |  |  |
| **Gauss-Seidel** | M - N | M = E + D  N = - F  temp = b-F@x0  x = SolveTr.Lsolve (M, temp) |  |  |
| **Gauss-Seidel SOR** | M - N | M = E + D  N = - F  temp = b - F@x0  xtilde, flag =Lsolve (Momega, temp)  xnew = (1 - omega) \* xold + omega\*xtilde #to do |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| METODI DI DISCESA | Vettore direzione p | Alpha | x | gamma |
| Steepest descent | P = -r | Alpha = - (r.T @ p) / (p.T @ Ap) | x = x + alpha \* p |  |
| Gradiente congiunto | P = -r | Alpha = (r.T @ p) / (p.T @ Ap) | x = x + alpha \* p | gamma **=** ([r**.**T@r) **/**](mailto:r.T@r)%20/)rtr\_old  p = -r + gamma \* p |