Efficient root finding schemes can be done in two steps: eyeballing and iterating. Eyeballing is qualitatively locating root by plotting. Iterating is looping reccurence relations unique to each algorithm. Nonlinear systems can be linearized and analyzed by linear algebra.

---

fixed-point-iteration:~# Also called *relaxation,* it has a reccurence relation expressed as $x_{n+1}=g(x_n)$. To assure convergence, $|g'(x)|<1$ must hold true.

```
for i in range(1, N):
    x1 = f1(x1,x2)
    x2 = f2(x1,x2)
```

Fixed point iteration can be extended to multiple dimensions yet easy to code.

```
jacobian_f = jacobian(f)
eig.jacobian_f
```

[autograd library] For multidimensional function f, check that all |λ|, eigenvalues of Jacobian matrix, are less than 1.

+ Advantage: Easy to code
! Pitfall: For functions whose derivatives can't be derived analytically, convergence can't be assured and one must be wary of unstable solutions.

newton-raphson:~# A special case of fixed-point iteration is when $g(x_n)=x_n-f(x_n)/f'(x_n)$. This is a relaxation method by means of linear approximations.

```
x_init = 10
x = x_init
for i in range (1,N):
    error = abs((f(x))/df(x))
    x = x - ((f(x))/df(x))
```

It is assumed here that df(x) is derived analytically although numerical one can be used. Observe that the error is merely $(x_{n+1}-x_n)$, an absolute flex of convergence.

```
gradf = gradient_f(x)
```

[autograd library] Extension of this leads to method of gradient descent (optimization) using a multidimensional Newton Raphson: $x_{n+1}=x_n-\gamma\ \text{grad}(f(x_n))$, where x is a vector.

+ Advantage: Superior quadratic convergence rate
! Pitfall: Chaotic nature of recurrence relation is prone to instability. Initial root must be properly picked within the vicinity of eyeballed root

bisection:~# Bisection iteration scheme relies on bracketing an eyeballed solution with two test roots of opposite sign and iteratively replacing leg of sign ± with their midpoint of sign ±.

```
if (f(right)*f(mid)) > 0:
    right = mid
else:
    left = mid
```

Snippet shows the update part of bisection method. If the sign of the function at midpoint is similar at right leg, change right leg to midpoint. Otherwise, change left leg to midpoint.

+ Advantage: Guarranteed convergence
! Pitfall: Here, eyeballing is not a recommendation but a requirement - this is to assure bracketing and avoiding even numbers of roots.