Bi7740: Scientific computing

Root finding

Vlad Popovici
popovici@iba.muni.cz

Institute of Biostatistics and Analyses
Masaryk University, Brno

Outline

Nonlinear equations

- *scalar problem*: $f : \mathbb{R} \to \mathbb{R}$, find $x \in \mathbb{R}$ such that $f(x) = 0$
- *vectorial problem*: $f : \mathbb{R}^n \to \mathbb{R}^n$, fing $\mathbf{x} \in \mathbb{R}^n$ such that $f(\mathbf{x}) = \mathbf{0}$
- in any case, here we consider f to be continuously differentiable everywhere in the neighborhood of the solution
- an interval $[a, b]$ is a bracket for the function f if $f(a)f(b) < 0$
- f continuous $\to f([a, b])$ is an interval
- Bolzano's thm.: if $[a, b]$ is a bracket for f than there exists at least one $x^* \in [a, b]$ s.t. $f(x^*) = 0$
- if $f(x^*) = f'(x^*) = \cdots = f^{(m-1)}(x^*) = 0$ but $f^{(m)} \neq 0$ then x^* has multiplicity m
- note: in $\mathbb{R}^n$ things are much more complicated

Conditioning

- absolute condition number is for a scalar problem $1/|f'(x^*)|$
- a multiple is ill-conditioned
- absolute condition number for a vectorial problem is $||\mathbf{J}_f^{-1}(\mathbf{x}^*)||$, where $\mathbf{J}_f$ is the Jacobian matrix of f,

$$[\mathbf{J}_f(\mathbf{x})]_{ij} = \frac{\partial f_i(\mathbf{x})}{\partial x_j}$$

- if the Jacobian is nearly singular, the problem is ill-conditioned

Sensitivity and conditioning

- possible interpretations of the *approximate solution*:
  - $||f(\hat{\mathbf{x}}) - f(\mathbf{x}^*)|| \leq \epsilon$: small residual
  - $||\hat{\mathbf{x}} - \mathbf{x}^*|| \leq \epsilon$ closeness to the true solution
- the two criteria might not be satistfied simultaneously
- if the problem is well-conditioned: small residual implies accurate solution

Convergence rate

- usually, the solution is find iteratively
- let $\mathbf{e}_k = \mathbf{x}_k - \mathbf{x}^*$ be the error at the $k-$th iteration, where $\mathbf{x}_k$ is the approximation and $\mathbf{x}^*$ is the true solution
- the method converges with rate r if

$$\lim_{k\to\infty} \frac{||\mathbf{e}_{k+1}||}{||\mathbf{e}_k||^r} = C, \quad \text{for } C > 0 \text{ finite}$$

- if the method is based on improving the bracketing, then $\mathbf{e}_k = b_k - a_k$
- if
  - $r = 1$ and $C < 1$, the convergence is linear and a constant number of digits are "gained" per iteration
  - $r = 2$ the convergence is quadratic, the number of exact digits doubles at each iteration
  - $r > 1$ the converges is superlinear, increasing number of digits are gained (depends on r)

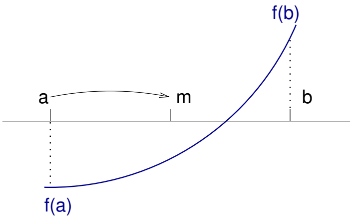
Outline

Bisection method

Idea: refine the bracketing of the solution until the length of the interval is small enough. Assumption: there is only one solution in the interval.

Algorithm 1: Bisection

while $(b - a) > \epsilon$ **do**
- $m \leftarrow a + \frac{b-a}{2}$
- **if** $\text{sign}(f(a)) = \text{sign}(f(m))$ **then**
  - $a \leftarrow m$
- **else**
  - $b \leftarrow m$



HOWEWORK Implement the above procedure in MATLAB.

Bisection, cont'd

- convergence is certain, but slow
- convergence rate is linear ($r = 1$ and $C = 1/2$)
- after k iterations, the length of the interval is $(b-a)/2^k$, so achieving a tolerance ϵ requires

$$\left\lceil \log_2 \frac{b-a}{\epsilon} \right\rceil$$

  iterations, *idependently* of f.
- the value of the function is not used, just the sign

Fixed-point methods

- a fixed point for a function $g : \mathbb{R} \to \mathbb{R}$ is a value $x \in \mathbb{R}$ such that $f(x) = x$
- the fixed-point iteration

$$x_{k+1} = g(x_k)$$

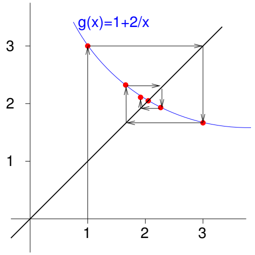
  is used to build a series of successive approximations to the solution
- for a given equation $f(x)$ there might be several equivalent fixed-point problems $x = g(x)$

The solutions of the equation

$$x^2 - x - 2 = 0$$

are the fixed points of each of the following functions:

- $g(x) = x^2 - 2$
- $g(x) = \sqrt{x+2}$
- $g(x) = 1 + 2/x$
- $g(x) = \frac{x^2+2}{2x-1}$



Conditions for convergence

- a function $g : S \subset \mathbb{R} \to \mathbb{R}$ is called Lipschitz-bounded if $\exists \alpha \in [0, 1]$ so that $|f(x_1) - f(x_0)| \leq \alpha |x_1 - x_0|, \forall x_0, x_1 \in S$
- in other words, if $|g'(x^*)| < 1$, then g is Lipschitz-bounded
- for such functions, there exists an interval containing x^* s.t. iteration

$$x_{k+1} = g(x_k)$$

  converges to x^* if started within that interval
- if $|g'(x^*)| > 1$ the iterations diverge
- in general, convergence is linear
- *smoothed iterations*:

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

  with $\lambda_k \in [0, 1]$ and $\lim_{k \to \infty} \lambda_k = 0$

Stopping criteria

If either

1. $|x_{k+1} - x_k| \leq \epsilon_1 |x_{k+1}|$ (relative error)
2. $|x_{k+1} - x_k| \leq \epsilon_2$ (absolute iteration error)
3. $|f(x_{k+1}) - f(x_k)| \leq \epsilon_3$ (absolute functional error)

stop the iterations.

Newton-Raphson method

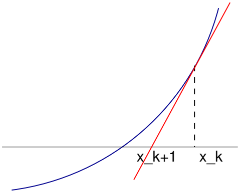
- from Taylor series:

$$f(x+h) \approx f(x) + f'(x)h$$

so in a small neighborhood around x $f(x)$ can be approximated by a linear function of h with the root $-f(x)/f'(x)$

- Newton iteration:

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



Implement the Newton-Raphson procedure in MATLAB.

```
function [r, xk] = NRiter(f, fprime, x, tol, max_iter)
  xk(1) = x;
  for k=1:max_iter
    xk(k+1) = ................
    if <approx is good enough> break
  end

  r = xk(...);
return
```

Apply it like:

```
>> f  = @(x) (x^2 -4*sin(x));
>> fp = @(x) (2*x-4*cos(x));
>> NRiter(f, fp, 3)
ans =
    1.9338
```

Newton-Raphson method, cont'd

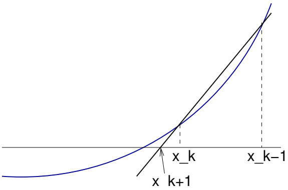
- convergence for a simple root is quadratic
- to converge, the procedure needs to start close enough to the solution, where the function f is monotonic

Secant method (lat.: Regula falsi)

- secant method approximates the derivative by finite differences:

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

- convergence is normally superlinear, with $r \approx 1.618$
- it must be started in a properly chosen neighborhood
- implement in MATLAB `[r,xk] = secant(f, x0, x1,...)`



Interpolation methods and other approaches

- secant method uses linear interpolation
- one can use higher-degree polynomial interpolation (e.g. quadratic) and find the roots of the interpolating polynomial
- inverse interpolation: $x_{k+1} = p^{-1}(y_k)$ where p is an interpolating polynomial
- fractional interpolation
- special methods for finding roots of the polynomials

Fractional interpolation

- previous methods have difficulties with functions having horizontal or vertical asymptotes
- *linear fractional interpolation* uses

$$\phi(x) = \frac{x-u}{vx-w}$$

function, which has a vertical asymptote at $x = w/v$, a horizontal asymptote at $y = 1/v$ and a zero at $x = u$

Fractional interpolation, cont'd

- let x_0, x_1, x_2 be three points where the function is estimates, yielding f_0, f_1, f_2
- find u, v, w for ϕ by solving

$$\begin{bmatrix} 1 & x_0 f_0 & -f_0 \\ 1 & x_1 f_1 & -f_1 \\ 1 & x_2 f_2 & -f_2 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$$

- the iteration step swaps the values: $x_0 \leftarrow x_1$ and $x_1 \leftarrow x_2$
- the new approximate solution is the zero of the linear fraction, $x_2 = u$. This can be implemented as

$$x_2 \leftarrow x_2 + \frac{(x_0 - x_2)(x_1 - x_2)(f_0 - f_1)f_2}{(x_0 - x_2)(f_2 - f_1)f_0 - (x_1 - x_2)(f_2 - f_0)f_1}$$

Outline

Systems of nonlinear equations

- much more difficult than the scalar case
- no simple way to ensure convergence
- computational overhead increases rapidly with the dimension
- difficult to determine the number of solutions
- difficult to find a good starting approximation

Fixed-point methods in $\mathbb{R}^n$

- $\mathbf{g} : \mathbb{R}^n \to \mathbb{R}^n$, $\mathbf{x} = \mathbf{g}(\mathbf{x}) = [g_1(\mathbf{x}), \ldots, g_n(\mathbf{x})]$
- fixed-point iteration: $\mathbf{x}_{k+1} = \mathbf{g}(\mathbf{x}_k)$
- denote $\rho(\mathbf{J}_g(\mathbf{x}))$ the *spectral radius* (maximum absolute eigenvalue) of the Jacobian matrix of $\mathbf{g}$ evaluated at $\mathbf{x}$
- if $\rho(\mathbf{J}_g(\mathbf{x}^*)) < 1$, the fixed point iteration converges if started close enough to the solution
- the convergence is linear with $C = \rho(\mathbf{J}_g(\mathbf{x}^*))$

Newton-Raphson method in $\mathbb{R}^n$

- $\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{J}_f^{-1}(\mathbf{x}_k)\mathbf{f}(x_k)$
- no need for inversion; solve the system

$$\mathbf{J}_f(\mathbf{x}_k)\mathbf{s}_k = -\mathbf{f}(\mathbf{x}_k)$$

for Newton step $\mathbf{s}_k$ and iterate

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_k$$

HOMEWORK: Implement in MATLAB the above procedure

Broyden's method

- uses approximations of the Jacobian
- the initial approximation of $\mathbf{J}$ can be the actual Jacobian (if available) or even $\mathbf{I}$

Algorithm 2: Broyden's method

for $k = 0, 1, 2, \dots$ **do**
- solve $\mathbf{B}_k \mathbf{s}_k = -\mathbf{f}(\mathbf{x}_k)$ for $\mathbf{s}_k$
- $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_k$
- $\mathbf{y}_k = \mathbf{f}(\mathbf{x}_{k+1}) - \mathbf{f}(\mathbf{x}_k)$
- $\mathbf{B}_{k+1} = \mathbf{B}_k + ((\mathbf{y}_k - \mathbf{B}_k \mathbf{s}_k)\mathbf{s}_k^T)/(\mathbf{s}_k^T \mathbf{s}_k)$
- **if** $\|\mathbf{x}_{k+1} - \mathbf{x}_k\| \geq \epsilon_1(1 + \|\mathbf{x}_{k+1}\|)$ **then**
  - continue
- **if** $\|\mathbf{f}(\mathbf{x}_{k+1})\| < \epsilon_2$ **then**
  - $\mathbf{x}^* = \mathbf{x}_{k+1}$
  - break
- **else**
  - algorithm failed

Further topics

- secant method is also extended to $\mathbb{R}^n$ (see Boryden's method)
- robust Newton-like methods: enlarge the region of convergence, introduce a scalar parameter to ensure progression toward solution
- in MATLAB: `roots()` for roots of a polynomial; `fzero()` for scalar case; `fsolve()` for the $\mathbb{R}^n$ case.

MATLAB exercise

- `help fsolve`
- try the examples from the documentation
- solve the system:

$$\begin{cases} x^2 + 2y^2 - 5x + 7y & = 40 \\ 3x^2 - y^2 + 4x + 2y & = 28 \end{cases}$$

with initial approximation $[2, 3]$