无约束最优化方法的基本结构：编程实践

龚梓阳

日期：2021 年 10 月 8 日

对于无约束最优化问题

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}) \tag{1}$$

一种常用的解决该问题的迭代算法如下：

> 选定初始点 $\mathbf{x}_0 \in \mathbb{R}^n$，在 $\mathbf{x}_0$ 处，确定一个使函数值下降的方向 $\mathbf{d}$ 与步长 α，继而求得下一个迭代点，以此类推，产生一个迭代点列 $\{\mathbf{x}_k\}$，$\{\mathbf{x}_k\}$ 或者其子列应收敛于最优解。当给定的某种终止准则满足时，或者 $\mathbf{x}_k$ 已满足要求的近似最优解的精度，或者算法已经无力进一步改善迭代点，则迭代结束。

评论. 对于上述迭代算法，下降方向与步长的选取顺序不同，导致产生不同类型的方法。

- 线搜索方法：在 $\mathbf{x}_k$ 点求得下降方向 $\mathbf{d}_k$，再沿 $\mathbf{d}_k$ 确定步长 α_k；
- 信赖域方法：在 $\mathbf{x}_k$ 点，先限定步长的范围，再同时确定下降方法 $\mathbf{d}_k$ 和步长 α_k。

1 线搜索方法

无约束最优化算法中线搜索方法的基本结构如下：

Algorithm 1: 线搜索方法的基本结构（P15）

Input: 目标函数 $f(\mathbf{x})$，初始点 $\mathbf{x}_0 \in \mathbb{R}^n$ 以及终止准则

Output: 最优解 $\mathbf{x}^*$ 以及 $f(\mathbf{x}^*)$

for $k = 0, \ldots,$ **do**

    确定下降方向 $\mathbf{d}_k$，使得 $\nabla f(\mathbf{x}_k)' \mathbf{d}_k < 0$;

    确定步长 α_k 使得 $f(\mathbf{x}_k + \alpha_k \mathbf{d}_k) < f(\mathbf{x}_k)$;

    $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \alpha_k \mathbf{d}_k$;

    if $\mathbf{x}_k$ 满足给定的终止准则 **then**

        break;

    end

end

$\mathbf{x}^* \leftarrow \mathbf{x}_k$，$f(\mathbf{x}^*) \leftarrow f(\mathbf{x}_k)$;

评论. `while` 与 `for` 有什么区别？

```python
def unconstrained_optimize(f, x0, epsilon=1e-8, max_iter=1000):
    x = np.empty((max_iter+1, x0.shape[0]))  # 定义 x 初始存储空间

    x[0] = x0
    for k in range(max_iter):
        d = search_desc_direction(f, x[k], ...)

        phi = lambda alpha: f(x[k] + alpha * d)
        alpha = search_step_length(phi, ...)

        x[k+1] = x[k] + alpha * d

        # if np.linalg.norm(g(x[k+1])) <= epsilon:
        # if f(x[k]) - f(x[k+1]) <= epsilon:
        if np.linalg.norm(x[k] - x[k+1]) <= epsilon:
            break

    return x[k+1], f(x[k+1])
```

Listing 1: 线搜索方法的基本结构：Python 实现

1.1 精确线搜索方法

1.1.1 确定步长搜索区间

从一点出发，按一定步长，试图确定函数值呈现“高-低-高”的三点，从而得到一个近似的单峰区间。

Algorithm 2: 进退法求初始搜索区间（P26）

Input: 一元函数 $\phi(\alpha)$，初始点 α_0，初始步长 γ_0，步长因子 t

Output: 初始区间 $[a, b]$

for $i = 0, \ldots,$ **do**
  $\alpha_{i+1} \leftarrow \alpha_i + \gamma_i$;
  if $\phi(\alpha_{i+1}) \geq \phi(\alpha_i)$ 或 $\alpha_{i+1} \leq 0$ **then**
    if $i = 0$ **then** $\gamma_{i+1} \leftarrow -\gamma_i$，$\alpha \leftarrow \alpha_{i+1}$ **else** break;
  else
    $\gamma_{i+1} \leftarrow t\gamma_i$，$\alpha \leftarrow \alpha_i$，$\alpha_i \leftarrow \alpha_{i+1}$;
  end
end
$a \leftarrow \min\{\alpha, \alpha_{i+1}\}$，$b \leftarrow \max\{\alpha, \alpha_{i+1}\}$;

评论. 在该算法中 α_i，α_{i+1}, α 分别起到了什么作用？

```python
def search_unimodal_interval(phi, alpha0, gamma=0.1, t=2, max_iter=100):
    alphas = np.empty(max_iter + 1)

    alphas[0] = alpha0
    for i in range(max_iter):
        alphas[i+1] = alphas[i] + gamma
        if phi(alphas[i+1]) >= phi(alphas[i]) or alphas[i+1] <= 0:
            if i == 0:
                gamma = -gamma
                alpha = alphas[i+1]
            else:
                break
        else:
            gamma = t * gamma
            alpha = alphas[i]
            alphas[i] = alphas[i+1]

    return min(alpha, alphas[i+1]), max(alpha, alphas[i+1])
```

Listing 2: 进退法求初始搜索区间：Python 实现

评论. 在该代码中，是否有必要存储每一个 α_i 么？

1.1.2 缩小步长搜索区间

0.618 方法 通过试探点函数值的比较，使包含极值点的搜索区间不断缩小。

Algorithm 3: 0.618 方法求一元函数 $\phi(\alpha)$ 的近似极小点（P27）

Input: 一元函数 $\phi(\alpha)$，初始搜索区间 $[a_0, b_0]$ 满足 $a_0 > b_0 > 0$，容许误差 $\varepsilon > 0$

Output: 近似极小点 α^*

$\tau \leftarrow \frac{\sqrt{5}-1}{2} \approx 0.618$;

for $i = 0, \ldots,$ **do**

  $a_i^l \leftarrow a_i + (1-\tau)(b_i - a_i)$，$a_i^r \leftarrow a_i + \tau(b_i - a_i)$;

  if $\phi\left(a_i^l\right) < \phi\left(a_i^r\right)$ **then**

    $a_{i+1} \leftarrow a_i$，$b_{i+1} \leftarrow a_i^r$;

  else

    $a_{i+1} \leftarrow a_i^l$，$b_{i+1} \leftarrow b_i$;

  end

  if $b_{i+1} - a_{i+1} < \varepsilon$ **then** break;

end

$a^* \leftarrow \frac{b_i + a_i}{2}$;

```python
def search_step_length_gold(phi, a0, b0, epsilon=1e-8, max_iter=1000):
    a, b = np.empty(max_iter + 1), np.empty(max_iter + 1)

    a[0], b[0] = a0, b0
    tau = (np.sqrt(5) - 1) / 2
    for i in range(max_iter):
        a_l = a[i] + (1 - tau) * (b[i] - a[i])
        a_r = a[i] + tau * (b[i] - a[i])
        if phi(a_l) < phi(a_r):
            a[i+1], b[i+1] = a[i], a_r
        else:
            a[i+1], b[i+1] = a_l, b[i]
        if b[i+1] - a[i+1] < epsilon:
            break
    return (a[i+1] + b[i+1]) / 2
```

Listing 3: 0.618 方法求一元函数 $\phi(\alpha)$ 的近似极小点：Python 实现

评论. 在该代码中，是否利用了0.618 法简化计算的目的（每次少算一次 $\phi(\cdot)$）？如果没有，我们应该怎么解决呢？

多项式插值法 通过在搜索区间中不断地使用二次多项式去近似目标函数，并逐步用插值多项式的极小点去逼近线搜索问题的极小点。

设 $\alpha_1 < \alpha_2 < \alpha_3$，$\phi(\alpha_1) > \phi(\alpha_2)$，$\phi(\alpha_2) < \phi(\alpha_3)$，拟合如下的二次插值多项式:

$$p(\alpha) = a\alpha^2 + b\alpha + c \tag{2}$$

满足插值条件：

$$\begin{cases} p(\alpha_1) = a\alpha_1^2 + b\alpha_1 + c = \phi(\alpha_1) \\ p(\alpha_2) = a\alpha_2^2 + b\alpha_2 + c = \phi(\alpha_2) \\ p(\alpha_3) = a\alpha_3^2 + b\alpha_3 + c = \phi(\alpha_3) \end{cases} \tag{3}$$

从极值的必要条件知 $p'(\alpha_p) = 2a\alpha_p + b = 0$，求得

$$\alpha_p = -\frac{b}{2a} \tag{4}$$

从而可以算出

$$\alpha_p = \frac{1}{2}\left(\alpha_1 + \alpha_2 - \frac{c_1}{c_2}\right) \tag{5}$$

其中

$$c_1 = \frac{\phi(\alpha_3) - \phi(\alpha_1)}{\alpha_3 - \alpha_1}, \quad c_2 = \frac{\frac{\phi(\alpha_2)-\phi(\alpha_1)}{\alpha_2-\alpha_1} - c_1}{\alpha_2 - \alpha_3} \tag{6}$$

Algorithm 4: 多项式插值法（三点二次插值法）求一维函数 $\phi(\alpha)$ 的近似极小点

Input: 一元函数 $\phi(\alpha)$，初始搜索区间 $[a_0, b_0]$ 满足 $a_0 > b_0 > 0$，容许误差 $\varepsilon > 0$

Output: 近似极小点 α^*

任取 $c_0 \in [a_0, b_0]$;

for $i = 0, \ldots,$ **do**

    $\alpha_p \leftarrow \frac{1}{2}\left(a_i + b_i - \frac{c_1}{c_2}\right)$，其中 $c_1 = \frac{\phi(c_i)-\phi(a_i)}{c_i-a_i}, c_2 = \frac{\frac{\phi(b_i)-\phi(a_i)}{b_i-a_i}-c_1}{b_i-c_i}$;

    if $\phi(c_i) \leq \phi(\alpha_p)$ **then**

        if $c_i \leq \alpha_p$ **then**

            $a_{i+1} \leftarrow a_i$，$c_{i+1} \leftarrow c_i$，$b_{i+1} \leftarrow \alpha_p$;

        else

            $a_{i+1} \leftarrow \alpha_p$，$c_{i+1} \leftarrow c_i$，$b_{i+1} \leftarrow b_i$;

        end

    else

        if $c_i \leq \alpha_p$ **then**

            $a_{i+1} \leftarrow c_i$，$c_{i+1} \leftarrow \alpha_p$，$b_{i+1} \leftarrow b_i$;

        else

            $a_{i+1} \leftarrow a_i$，$c_{i+1} \leftarrow \alpha_p$，$b_{i+1} \leftarrow c_i$;

        end

    end

    if $b_{i+1} - a_{i+1} < \varepsilon$ **then** break;

end

$\alpha^* \leftarrow \alpha_p$;

```python
def search_step_length_poly32(phi, a0, b0, epsilon=1e-8, max_iter=1000):
    a, b, c = np.empty(max_iter + 1), np.empty(max_iter + 1), np.empty(max_iter + 1)

    a[0], c[0], b[0] = a0, (a0 + b0) / 2, b0
    for i in range(max_iter):
        c1 = (phi(c[i]) - phi(a[i])) / (c[i] - a[i])
        c2 = ((phi(b[i]) - phi(a[i])) / (b[i] - a[i]) - c1) / (b[i] - c[i])
        alpha_p = 0.5 * (a[i] + b[i] - c1 / c2)
        if phi(c[i]) <= phi(alpha_p):
            if c[i] <= alpha_p:
                a[i+1], c[i+1], b[i+1] = a[i], c[i], alpha_p
            else:
                a[i+1], c[i+1], b[i+1] = alpha_p, c[i], b[i]
        else:
            if c[i] <= alpha_p:
                a[i+1], c[i+1], b[i+1] = c[i], alpha_p, b[i]
            else:
                a[i+1], c[i+1], b[i+1] = a[i], alpha_p, c[i]
        if b[i+1] - a[i+1] < epsilon:
            break

    return alpha_p
```

Listing 4: 多项式插值法（三点二次插值法）求一维函数 $\phi(\alpha)$ 的近似极小点：Python 实现

1.1.3 牛顿切线法

Algorithm 5: 牛顿切线法

Input: 一元函数 $\phi(\alpha)$ 及其一阶导函数 $\phi'(\alpha)$ 与二阶导函数 $\phi''(\alpha)$，初始点 α_0，容许误差 $\varepsilon > 0$

Output: 近似极小点 α^*

1 **for** $i = 0, \ldots,$ **do**

2 $\quad \alpha_{i+1} = \alpha_i - \frac{\phi'(\alpha_i)}{\phi''(\alpha_i)}$;

3 $\quad$ **if** $|\alpha_{i+1} - \alpha_i| < \varepsilon$ **then** break;

4 **end**

5 $\alpha^* \leftarrow \alpha_{i+1}$;

```python
def search_step_length_newton(phi, phi_grad, phi_hess, alpha0, epsilon=1e-8, max_iter=1000):
    alpha = np.empty(max_iter + 1)

    alpha[0] = alpha0
    for i in range(max_iter):
        alpha[i+1] = alpha[i] - phi_grad(alpha[i]) / phi_hess(alpha[i])
        if abs(alpha[i+1] - alpha[i]) < epsilon:
            break

    return alpha[i+1]
```

Listing 5: 牛顿切线法：Python 实现