Projeto 3 de Cálculo 2 Honors
Busca linear

Prof. Lucas Pedroso

2º semestre de 2024

Última atualização: 10/11 (confira no final do arquivo a lista com as modificações desde a publicação)

Este arquivo trata do achievement de busca linear do Projeto 3.

1 Motivação

No projeto principal, estamos usando o passo constante α. No entanto, em geral essa não é uma boa ideia: passos que obriguem o método a diminuir mais (ou seja, algo mais forte que $f(x^{k+1}) < f(x^k)$) são preferíveis e, às vezes, até necessários do ponto de vista teórico.

A principal regra de *decréscimo suficiente* é a condição de Armijo: em vez de exigirmos que $f(x^k + \alpha_k d^k) < f(x^k)$ (decréscimo simples), procuraremos α_k que satisfaça para $\tau > 0$

$$f(x^k + \alpha_k d^k) \leq f(x^k) + \alpha_k \tau \nabla f(x^k)^T d^k.$$

Note que isso é exigir uma diminuição maior na função objetivo do iterando atual x^k para o próximo $x^{k+1} = x^k + \alpha_k d^k$, uma vez que a derivada direcional $f(x^k)^T d^k$ é negativa, conforme veremos em aula.

Para encontrar o passo em cada iteração, dada a direção de busca d e o ponto atual x, o passo será o resultado do algoritmo a seguir.

Algorithm 1 Busca linear

Data: $f : \mathbb{R}^n \to \mathbb{R}$, $x, d \in \mathbb{R}^n$, $\tau, \gamma \in (0, 1)$.
Initialization: $\alpha \leftarrow 1$.
  while $f(x + \alpha d) > f(x) + \alpha\tau\nabla f(x)^T d$ **do**
    $\alpha \leftarrow \gamma\alpha$.
  end while
  return α

Os dados de entrada são os seguintes:

- f, a função que está sendo minimizada;
- x, o ponto atual a partir do qual se fará a busca;
- d, a direção de busca. No caso do método do gradiente, a direção de busca é $d = -\nabla f(x)$. Os métodos de Newton e BFGS têm direções próprias, que basicamente é o que define os métodos.
- τ, um parâmetro para a busca. Deve ser pequeno, vamos usar sempre $\tau = 10^{-3}$.
- γ, o parâmetro que reduz o passo α caso a condição de Armijo não tenha sido satisfeita. Vamos dividir o passo por 2 sempre, ou seja, $\gamma = 0.5$.

1.1 Método do gradiente com busca linear

Caso seja implementada a busca linear, o algoritmo do gradiente ficará da seguinte forma:

Algorithm 2 Método do gradiente com busca linear

Data: $f : \mathbb{R}^n \to \mathbb{R}$, $x^0 \in \mathbb{R}^n$, $\epsilon > 0$, $K \in \mathbb{N}$.
Initialization: $x \leftarrow x^0$ e $k \leftarrow 0$.
  while $\|\nabla f(x)\| > \epsilon$ e $k < K$ **do**
    $k \leftarrow k + 1$.
    Chame a função de busca linear para calcular α.
    $x \leftarrow x - \alpha\nabla f(x)$.
  end while
  return x, k

Modificação semelhante ocorre para os métodos de Newton e BFGS.

2 Implementação

A implementação basicamente será um código de poucas linhas implementando o pseudocódigo da busca linear descrito anteriormente.

2.1 Primeira linha da função

```
def linesearch(f,x,g,d):
```

sendo

- `f`: a função que está sendo minimizada;
- `x`: o ponto atual, a partir do qual a busca será feita;
- `g`: o gradiente da função `f` no ponto `x`;
- `d`: a direção de busca.

A saída será o passo `alpha` que satisfaz o critério de Armijo.

3 Exemplo

Para a função

```
def f(x):
return x[0]**4-2*x[0]**2+x[0]-x[0]*x[1]+x[1]**2
def grad(x):
return np.array([4*x[0]**3-4*x[0]+1-x[1],-x[0]+2*x[1]])
```

ao executar

```
x,k = gd(f,np.array([10,10]),grad,alpha = 1e-2,itmax = 100000,
         eps = 1e-8,plot = True)
print(f"x = x")
print(f"k = k")
```

o algoritmo não converge! Isso ocorre, neste caso, porque o passo é muito grande. Algo desse tipo pode aparecer como output

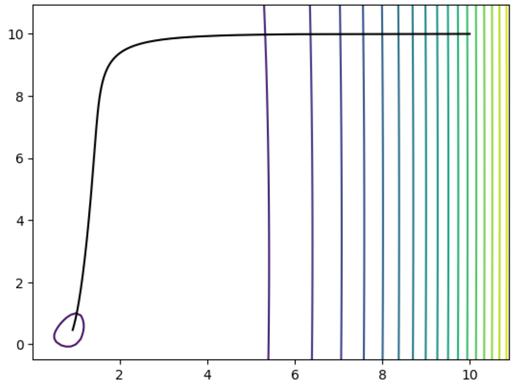
```
x = [ -inf 9.73495111e+184]
k = 7
```

Diminuindo o passo, executando

```
x,k = gd(f,np.array([10,10]),grad,alpha = 1e-3,itmax = 100000,
         eps = 1e-8,plot = True)
print(f"x = x")
print(f"k = k")
```

o algoritmo passa a convergir, porém demora muitas iterações (quase 12000).

```
x = [0.92442503 0.46221252]
k = 11859
```

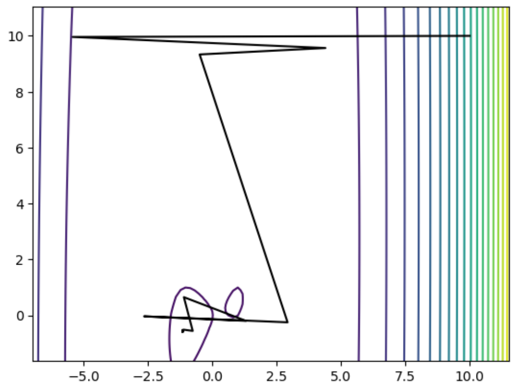


Por fim, usando busca linear em vez de passo constante

```
x,k = gd(f,np.array([10,10]),grad,search = True,itmax = 100000,
         eps = 1e-8,plot = True)
print(f"x = x")
print(f"k = k")
```

o algoritmo converge em bem menos iterações (menos de 70)

```
x = [-1.15797021 -0.57898511]
k = 65
```



Notem um detalhe interessante. Apesar de sempre estarmos começando do mesmo chute inicial (`x0 = [10,10]`), o método com passo constante e o método com busca linear convergiram pra pontos diferentes. Ambos são minimizadores do problema, sendo o ponto `x = [-1.15797021 -0.57898511]` minimizador global. Ou seja, em havendo mais de um ponto estacionário, não se sabe para qual se dará a convergência.

4 Detalhes e comentários

- No código do método do gradiente deverá haver um `if search:`. Se for `True`, chamar a função `linesearch(f,x,g,d)` pra calcular o `alpha`. Se for `False`, usar o `alpha` constante que é entrada da função `gd`. O mesmo vale para Newton e BFGS.

5 Updates neste arquivo

- 10/11 - foi acrescentado um exemplo de execução do método do gradiente com e sem busca linear.