Universidade de São Paulo
-Instituto de Matemática e Estatística-

Otimização Não Linear

- Avaliação Final -
Método de Newton para problemas com restrições de igualdade

Erik Davino Vincent
Guilherme Kang
6 de Junho de 2020

Conteúdo

1 Introdução **1**

2 Método de Newton **1**

2.1 Condições de otimalidade 2

2.2 Algorítimo do Método de Newton 2

2.3 Método de Newton adaptado 2

3 Otimização com restrições **3**

3.1 Método de Newton para otimização com restrições 4

4 Problema da Catenária/Corrente **6**

4.1 Discretização 7

4.2 Outra discretização 9

5 Algencan **11**

5.1 Primeiro passo 11

5.2 Segundo passo 12

5.3 Terceiro passo 12

5.3.1 myevalf, myevalg e myevalh 12

5.3.2 myevalc, myevaljac, myevalhc 13

5.3.3 main 13

5.4 Execução 14

5.5 ALGENCAN vs. Nosso método 15

6 Considerações finais **16**

Referências **17**

1 Introdução

O seguinte relatório tem como objetivo apresentar a implementação do Método de Newton para resolver um problema de otimização com restrições de igualdade. Será apresentado um meio para a globalização do método proposto, assim como discretização para o problema de minimização não linear da catenária, além das motivações por trás das técnicas implementados para resolvê-lo numericamente. Ao final, serão apresentados os resultados obtidos.

2 Método de Newton

O Método de Newton n-dimensional é um algorítimo para solucionar sistemas de equações não lineares. Mas ele pode ser adaptado para resolver problemas de otimização. Utilizando uma aproximação pela expansão de Taylor de segunda ordem, podemos obter um passo do método. Dada $f : \mathbb{R}^n \to \mathbb{R}$ contínua duas vezes diferenciável e um ponto $x^k \in \mathbb{R}^n$, temos:

$$f(x^k + \delta x) \approx f(x^k) + \nabla f(x^k)^T \delta x + \frac{1}{2} \delta x^T \nabla^2 f(x^k) \delta x$$

de onde obtemos a direção de descida:

$$\delta x^k = -(\nabla^2 f(x^k))^{-1} \nabla f(x^k)$$

De forma iterativa podemos obter um novo ponto x^{k+1} por

$$x^{k+1} = x^k + \delta x^k$$

gerando uma sequência $\{x^k\}$ que, dadas as condições favoráveis, converge para $\bar{x}$, solução do problema, de forma quadrática.

Alternativamente podemos obter δx resolvendo o sistema dado pela equação

$$(\nabla^2 f(x^k)) \delta x^k = -\nabla f(x^k)$$

Para que o método acima funcione corretamente em um problema de minimização algumas condições são necessárias. De imediato é possível notar que o método falha se para algum ponto x^k a matriz hessiana não possui uma inversa. Além disso, se δx^k não for direção de descida, não é possível garantir a convergência do algorítimo para um ponto de mínimo.

2.1 Condições de otimalidade

Sabe-se de cálculo que se $\bar{x}$ é um ponto estacionário de $f(x)$, então $\nabla f(\bar{x}) = 0$. Essa é a primeira condição necessária de otimalidade, que pode ser usada como critério de parada do método. A segunda condição necessária de otimalidade é que se $\bar{x}$ é um mínimo local de $f(x)$, então a matriz hessiana da função é semidefinida positiva.

De fato, ao montar o algorítimo para o método apenas se utiliza a primeira condição de otimalidade, além de sempre se tentar obter uma direção de descida.

2.2 Algorítimo do Método de Newton

Utilizando o que foi discutido anteriormente podemos montar o algorítimo padrão para o Método de Newton.

Dado: $x^0 \in \mathbb{R}^n, f : \mathbb{R}^n \to \mathbb{R}$
$k = 0$
ENQUANTO $\nabla f(x^k) \neq 0$:
    $\delta x^k \leftarrow -(\nabla^2 f(x^k))^{-1} \nabla f(x^k)$
    $x^{k+1} \leftarrow x^k + \delta x^k$
    $k \leftarrow k + 1$
RETORNA x^k

Esse algorítimo não possui ainda nenhuma forma de contorno para os problemas que podem aparecer, de forma que será necessário adaptá-lo de forma que a direção seja sempre (ou quase sempre) de descida. Quanto ao problema de singularidade da matriz hessiana, não há muito o que fazer[1], mas ao menos esse ocorre raramente durante a execução do algorítimo. Em geral é um problema de valor inicial, ou para casos de divergência/não existência da solução.

2.3 Método de Newton adaptado

Uma forma que encontramos de garantir direção de descida é apresentada por Jarre, Florian (2003) [2]. Considere o seguinte teorema: Se $\nabla^2 f(x)$ é positiva definida, então a direção δx^k é de descida.

Porém, não há como garantir que a hessiana seja sempre uma matriz positiva definida. Por isso o autor em [2] sugere utilizar uma aproximação da hessiana que seja positiva definida, ou ao menos semipositiva definida. Uma forma de se obter essa aproximação é através da decomposição em autovalores.

[1] Se ocorrer, uma solução possível seria tentar atualizar x^k em uma direção aleatória com um passo bem pequeno.

Seja H a hessiana do lagrangiano. Podemos decompor H pela forma $H = UDU^T$, onde D é uma matriz diagonal de autovalores de H e U uma matriz unitária. Seja D^+ a matriz D onde seus valores negativos são substituídos por seus opostos (e.g. $-2.3 \to 2.3$). Assim construímos $H^+ = UD^+U^T$, que de acordo com [2] é uma das melhores aproximações (semi)positiva definida de H. Assim, substituindo H por H^+ permite que o algorítimo sempre caminhe em uma direção de descida. O algorítimo fica:

Dado: $x^0 \in \mathbb{R}^n, f : \mathbb{R}^n \to \mathbb{R}$
$k = 0$
ENQUANTO $\nabla f(x^k) \neq 0$:
    $H \leftarrow \nabla^2 f(x^k)$
    SE $\sim (H > 0)$ ENTÃO:
        $H \leftarrow H^+$
    $\delta x^k \leftarrow -(H)^{-1}\nabla f(x^k)$
    $x^{k+1} \leftarrow x^k + \delta x^k$
    $k \leftarrow k + 1$
RETORNA x^k

3 Otimização com restrições

Os problemas de otimização com restrições possuem a forma geral

$$\begin{aligned} \underset{x}{\text{minimizar}} \quad & f(x) \\ \text{sujeito a} \quad & g(x) \leq 0 \\ & h(x) = 0 \end{aligned}$$

onde $f : \mathbb{R}^n \to \mathbb{R}$, $g(x) = (g_1(x), \ldots, g_p(x))$ e $h(x) = (h_1(x), \ldots, h_m(x))$ são funções continuamente diferenciáveis. Além disso definimos o conjunto viável

$$\Omega = \{x \in \mathbb{R}^n | g(x) \leq 0, h(x) = 0\}$$

.

Seja $L(x, \mu, \lambda) = f(x) + \sum_{i=1}^{p} \mu_i g_i(x) + \sum_{j=1}^{m} \lambda_j h_j(x)$ a função Lagrangiana. O teorema de Karush-Kuhn-Tucker (KKT) afirma:
Seja $\bar{x} \in \Omega$ um minimizador local do problema com restrições e suponha $\bar{x}$ regular. Então existe $\mu \in \mathbb{R}^p$,$\lambda \in \mathbb{R}^m$ tais que

$$\nabla_x L(\bar{x}, \mu, \lambda) = \nabla f(\bar{x}) + \sum_{i=1}^{p} \mu_i \nabla g_i(\bar{x}) + \sum_{j=1}^{m} \lambda_j \nabla h_j(\bar{x}) = 0,$$

$$\mu \geq 0,$$

$$\mu_i g_i(\bar{x}) = 0, \ i = 1, \ldots, p$$

Essas condições necessárias de otimalidade mostram que é possível alterar o problema geral de minimizar $f(x)$ com restrições para o problema de minimizar o Lagrangiano (considerando as restrições impostas originalmente também). Novamente o objetivo se torna zerar o gradiente, e para tal será usado o Método de Newton alterado apresentado anteriormente.

Para o problema específico que será implementado, não há restrições de desigualdade, simplificando o Lagrangiano para

$$L(x, \lambda) = f(x) + \sum_{i=1}^{m} \lambda_i h_i(x)$$

e as condições necessárias de otimalidade para

$$\nabla_x L(\bar{x}, \lambda) = \nabla f(\bar{x}) + \sum_{j=1}^{m} \lambda_j \nabla h_j(\bar{x}) = 0,$$
$$h(x) = 0$$

3.1 Método de Newton para otimização com restrições

Para resolver o problema de minimização com restrições, adaptamos o Método de Newton para minimizar o lagrangiano do problema. Para isso, temos de adaptar a matriz utilizada no passo de Newton. Ratliff, Nathan (2014) [1] aponta como fazer isso:

Queremos ao mesmo tempo encontrar o minimizador do Lagrangiano e obedecer as restrições $h(x) = 0$. Assim, o método será utilizado para resolver o sistema

$$\begin{bmatrix} \nabla f(x) + J_h^T(x)\lambda \\ h(x) \end{bmatrix} = 0$$

Disso obtemos uma nova forma para o passo de Newton:

$$\underbrace{\begin{bmatrix} \nabla^2_{xx} L(x^k) & J_H^T(x^k) \\ J_h(x^k) & 0 \end{bmatrix}}_{A} \begin{bmatrix} \delta x^k \\ \delta \lambda^k \end{bmatrix} = \underbrace{\begin{bmatrix} -\nabla f(x^k) - J_h^T(x^k)\lambda \\ -h(x^k) \end{bmatrix}}_{-b}$$

Utilizando o que vimos nas seções anteriores, ainda temos de garantir a direção de descida para o sistema. Para tal, podemos substituir $\nabla^2_{xx} L(x^k)$ pela matriz H^+ como apontado anteriormente.

Nesse momento, o método de encontra muito bom. Mas ainda resta um problema. Queremos garantir a convergência global do método. Para isso, buscamos referência em Nocedal, J. & Wright,

S. [4], no capítulo 18, que descreve os algorítimos SQP (Sequential Quadratic Programming). O capítulo desenvolve a ideia de funções de mérito, que dado alguns hiper-parâmetros podem garantir a convergência global do algorítimo utilizando o passo do Método de Newton. Não entraremos em detalhes, apenas apontarei alguns conceitos chave para a construção do algorítimo final.

Primeiramente criamos a função de mérito

$$\phi(x, \mu) = f(x) + \mu\|h(x)\|$$

utilizada para executar uma busca linear e encontrar um tamanho de passo adequado t. Isso permite que não ocorra problemas de "overshoot"ao executar o passo de Newton. Para isso podemos usar a condição de Armijo:

$$\phi(x^k + t\delta x^k, \mu_k) \leq \phi(x^k, \mu_k) + t\eta D(\phi(x^k, \mu_k), \delta x^k)$$

onde $\eta \in (0, 1)$, constante. $D(\phi(x, \mu), \delta x)$ a derivada direcional de $\phi(x)$ na direção δx. Queremos que D seja direção de descida. Para isso, temos que garantir que δx seja de descida, o que fazemos pela substituição de H por H^+ quando necessário, e temos que garantir que μ seja suficientemente grande.

Está provado em [4] que se δx e $\delta\lambda$ são gerados pelo passo de Newton, então D tem a forma $D(\phi(x, \mu), \delta x) = \nabla f^T(x^k)\delta^k - \mu_k\|h(x^k)\|$ e que dado um hiper-parâmetro $\rho \in (0, 1)$, podemos garantir o tamanho de μ, se a cada passo garantirmos:

$$\mu_k \geq \frac{\nabla f^T(x^k)\delta x^k + 0.5(\delta x^k)^T H \delta x^k}{(1-\rho)\|h(x^k)\|}$$

Podemos garantir isso simplesmente fazendo:

Dado: $\rho \in (0, 1)$
$M \leftarrow \frac{\nabla f^T(x^k)\delta x^k + 0.5(\delta x^k)^T H \delta x^k}{(1-\rho)\|h(x^k)\|}$
SE $M > \mu_{k-1}$ ENTÃO:
    $\mu_k \leftarrow M$
SENÃO:
    $\mu_k \leftarrow \mu_{k-1}$

Com tudo isso estabelecido, podemos montar o algorítimo global:

Dado: $x^0 \in \mathbb{R}^n,\ f : \mathbb{R}^n \to \mathbb{R},\ \phi : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R},\ \rho \in (0,1)$
$k = 0$
ENQUANTO $\nabla L(x^k) \neq 0$:
$H \leftarrow \nabla^2 f(x^k)$
SE $\sim (H > 0)$ ENTÃO:
$H \leftarrow H^+$
$A \leftarrow \begin{bmatrix} H & J_h^T(x^k) \\ J_h(x^k) & 0 \end{bmatrix}$
$b \leftarrow \begin{bmatrix} \nabla_x L(x^k, \lambda^k) \\ h(x^k) \end{bmatrix}$
$(\delta x^k, \delta \lambda^k) \leftarrow -(A)^{-1} b$
DEFINA t (busca de Armijo definida acima);
$x^{k+1} \leftarrow x^k + t\delta x^k$
$\lambda^{k+1} \leftarrow \lambda^k + t\delta \lambda^k$
$k \leftarrow k + 1$
RETORNA x^k

4 Problema da Catenária/Corrente

"Considere um cabo de energia de 3 metros fixado entre dois postes a uma distância de 2 metros. Encontre a função $y(x)$ que mede a altura do cabo em cada posição x entre 0 e 2. Para isso, considere o problema de otimização:"

$$\underset{y(x)}{\text{minimizar}} \int_0^2 y(x)\sqrt{1 + y\prime(x)^2}dx$$

$$\text{sujeito a} \int_0^2 \sqrt{1 + y\prime(x)^2}dx,\ y(0) = 0,\ y(2) = 0.$$

"A função objetivo representa a energia potencial a ser minimizada e a restrição força que o comprimento da corda está fixado."

O problema possui solução conhecida $y(x) = a\cosh(\frac{x+b}{a}) - c$, para dados parâmetros a, b e c.

O problema acima, sobre minimizar a energia potencial da corda é em geral apresentado como o problema da catenária ou da corrente. Esse problema possui uma interpretação intuitiva; quanto mais alto um objeto está, maior a energia potencial esse objeto possui. A tendência da matéria é equilibrar a energia total sempre que possível, logo, os objetos caem. O que o problema descreve então é o todo da corda (os infinitos pontos que ela possui), tentando minimizar essa energia. Como a corda está preza, ela fará com que o máximo de pontos em seu comprimento desçam o máximo que podem.

Não é um exagero supor que uma corrente se comporta de forma quase igual a uma corda. Pensando na corrente, que possui pontos claros em que cada pedaço se liga ao outro, podemos imagina-la

como uma corda discretizada.

Para fazer a discretização, primeiramente veja os seguintes resultados de cálculo numérico para discretizar integrais

$$\int_a^b f(x)dx \approx S = \sum_{i=1}^{n} f(x_i + i\Delta x_i)\Delta x_i,$$
$$x_0 = a,\ x_n = b,\ \Delta x_i = (x_i - x_{i-1})$$
$$\lim_{n\to\infty} S = \int_a^b f(x)dx$$

e para derivadas temos diferenças finitas

$$f\prime(x) \approx \frac{f(x) - f(x - \Delta x)}{\Delta x},\ \Delta x_i = (x_i - x_{i-1})$$
$$\lim_{\Delta x\to 0} \frac{f(x) - f(x - \Delta x)}{\Delta x} = f\prime(x).$$

Além disso, considere uma função $y(x)$ conhecida em n pontos, de forma que $y(x_i) = y_i$, gerando o vetor $y = (y_0, \ldots, y_n)$.

Com esses recursos, considere o problema discretizado[2]:

4.1 Discretização

Seja $y \in \mathbb{R}^n$ e $dx = \frac{2}{n-1}$:

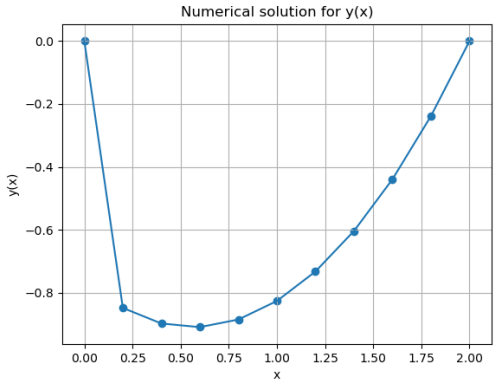
$$\underset{y}{\text{minimizar}}\ f(y) = \sum_{i=1}^{n} y_i \sqrt{1 + (\frac{y_i - y_{i-1}}{dx})^2 dx}$$
$$\text{sujeito a: } h(y) = \sum_{i=1}^{n} \sqrt{1 + (\frac{y_i - y_{i-1}}{dx})^2 dx} - 3,\ y_0 = 0,\ y_n = 0.$$

Essa forma de discretização do problema considera uma aproximação da derivada via ponto anterior. Seguem os resultados obtidos para diferentes valores de n[3]:

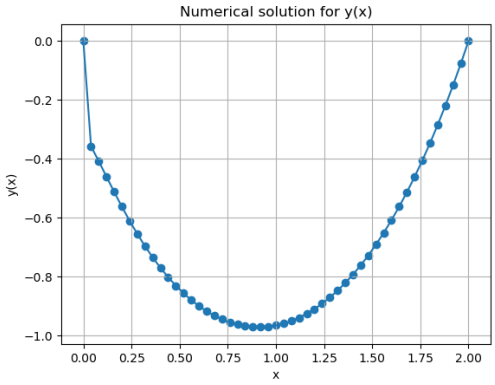
[2]Todos os gradientes e hessianos utilizados na implementação foram calculados com ajuda da linguagem Wolfram.
[3]Tolerância para declarar convergência: $1e - 10$. $\rho = 0.7$.

n	λ_1	λ_2	λ_3	$f(x)$	Tempo	Iterações
11	-1.14	-1.85	2.02	-1.94	4.20	47
51	-1.35	-1.55	1.66	-1.83	123.73	1000 (limite)
101	-1.47	-1.52	1.62	-1.81	15.57	101
201	-1.48	-1.51	1.62	-1.81	120.37	408

Tabela 1: Resultados obtidos para diferentes valores de n.

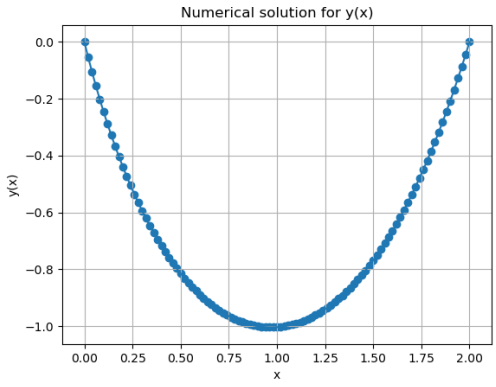


(a) $n = 11$

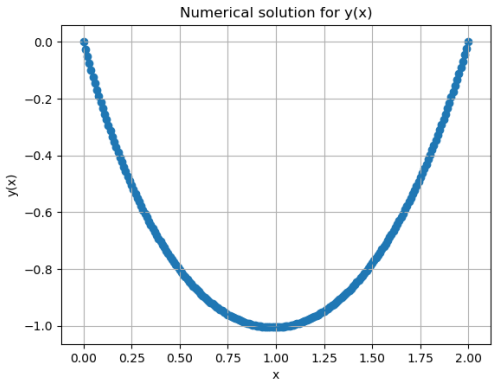


(b) $n = 51$

Figura 1: Gráficos das aproximações para $y(x)$.



(a) $n = 101$



(b) $n = 201$

Figura 2: Gráficos das aproximações para $y(x)$.

Utilizando a plataforma Desmos (`https://www.desmos.com/calculator`) podemos facilmente encontrar os parâmetros a, b e c que melhor aproximam a função $y(x) = a\cosh(\frac{x+b}{a}) + c$ dos pontos y obtidos através da minimização. Fizemos isso utilizando $n = 101$:

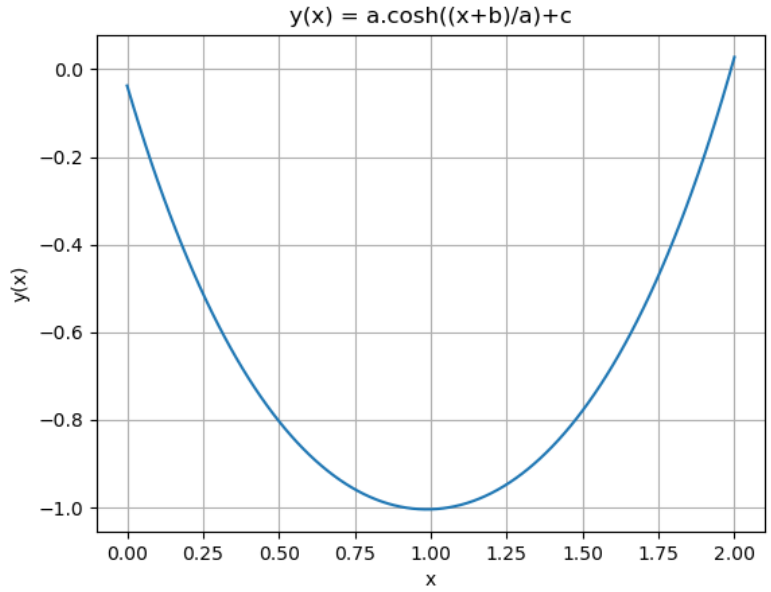


Figura 3: $a = 0.62$, $b = -0.98$ e $c = -1.62$.
$\int_0^2 \sqrt{1 + (y\prime(x))^2} dx = 2.99$

Note como o resultado final está "inclinado". Isso provavelmente se deve à forma como a derivada de $y(x)$ foi aproximada. Como as derivadas foram aproximadas utilizando o ponto anterior, há uma leve distorção na inclinação da derivada em cada ponto. Isso ocorre pois quanto mais à direita o ponto estiver, pior será a aproximação da derivada nele. Uma forma de driblar isso, seria utilizar uma aproximação de ponto médio para a derivada ou aumentar o número de pontos. Vejamos agora uma segunda forma de discretização.

4.2 Outra discretização

Griva, I. & Vanderbei, R. (2003) [3] apresentam em seu trabalho uma forma de discretizar o problema muito mais simples e numericamente mais estável, mas não produz exatamente o mesmo resultado que a discretização. De fato, a solução para a seguinte discretização é dada por $y(x) = a\cosh(\frac{x-b}{a}) + c$:

Seja $y \in \mathbb{R}^n$, $dx = \frac{2}{n-1}$:

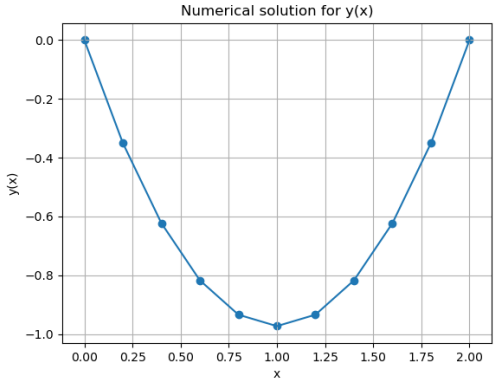
$$\underset{y}{\text{minimizar}} \; f(y) = \sum_{i=1}^{n} y_i dx$$

$$\text{sujeito a: } h(y) = \sum_{i=1}^{n} dx^2 + (y_i - y_{i-1})^2 - (\frac{3}{n-1})^2, \; y_0 = 0, \; y_n = 0.$$

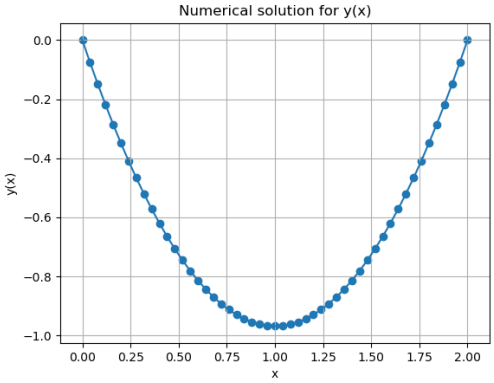
Vejamos os resultados obtidos para essa discretização:

n	λ_1	λ_2	λ_3	$f(x)$	Tempo	Iterações
11	-1.10	-1.10	1.28	-1.28	1.44	14
51	-1.02	-1.02	6.45	-1.29	4.22	44
101	-1.01	-1.01	12.90	-1.29	2.33	20
201	-1.005	-1.005	25.81	-1.29	9.22	49

Tabela 2: Resultados obtidos para diferentes valores de n.

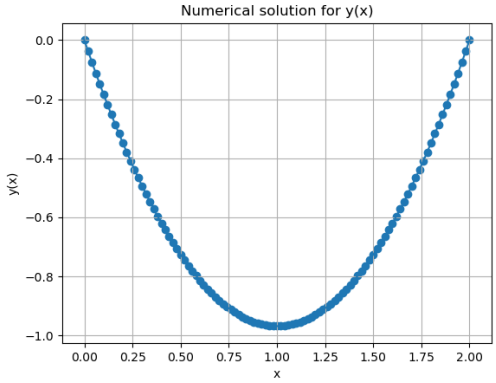


(a) $n = 11$

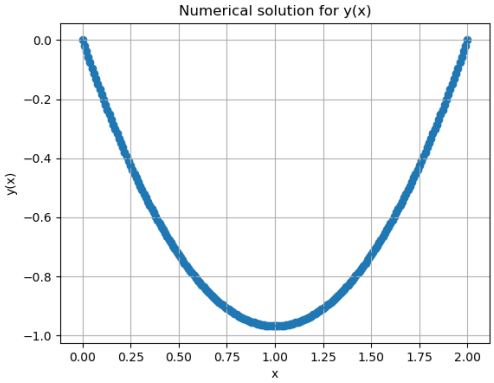


(b) $n = 51$

Figura 4: Gráficos das aproximações para $y(x)$.



(a) $n = 101$



(b) $n = 201$

Figura 5: Gráficos das aproximações para $y(x)$.

Utilizando a plataforma Desmos podemos facilmente encontrar os parâmetros a, b e c que melhor aproximam a função $y(x) = a\cosh(\frac{x-b}{a}) + c$ dos pontos y obtidos através da minimização. Fizemos isso utilizando $n = 101$:

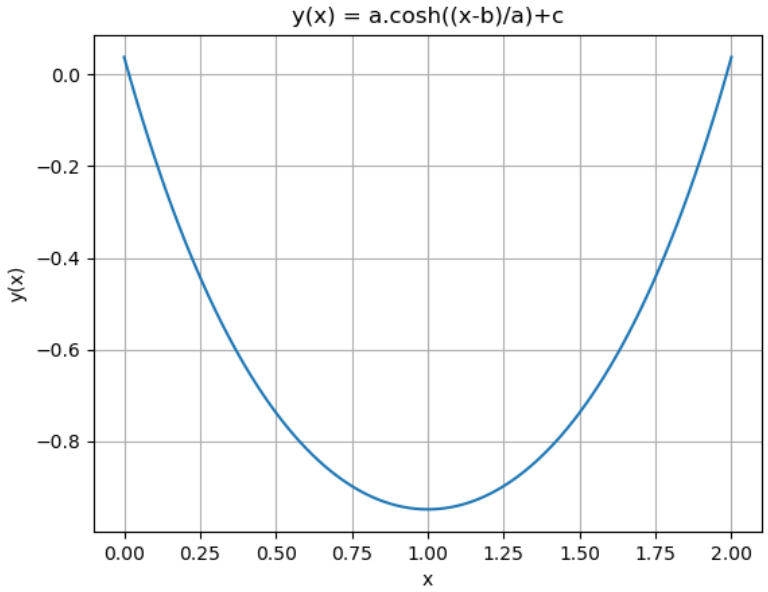


Figura 6: $a = 0.62$, $b = 1$ e $c = -1.57$.
$\int_0^2 \sqrt{1 + (y\prime(x))^2} dx = 2.97$

Podemos ver que essa curva se fixa levemente acima da curva encontrada anteriormente, mas ao menos ela foi encontrada com muito menos tempo e esforço computacional, além de ser totalmente simétrica. Essa discretização também não sofre devido a problemas com a derivada, logo não é necessário um grande valor de n para que a aproximação seja boa.

5 Algencan

A seguir comparamos os resultados obtidos pelo nosso otimizador com a aplicação Algencan. Mas antes, vejamos brevemente o como utilizar essa aplicação:

5.1 Primeiro passo

Recomendamos utilizar o sistema operacional Linux para executar o Algencan. Caso você não possua esse sistema operacional, é fácil encontrar programas de máquinas virtuais, como o "VirtualBox Oracle VM"(`https://www.virtualbox.org/`), para simular o Linux em seu computador. Essa parte fica para o leitor.

A versão mais recente do Algencan pode ser encontrada em `https://www.ime.usp.br/~egbirgin/tango/codes.php` na seção downloads. Será necessário submeter seu nome completo, instituição e e-mail. Uma vez baixado o arquivo zipado, você deverá descompactá-lo. Pode ser no desktop, por

exemplo.

Além disso, instale o GFortran. Ele é necessário para que o programa rode. Para isso, digite no terminal do Linux: sudo apt-get install gfortran. Faça o que for pedido em seguida. Se não for possível a instalação, atualize o apt-get, digitando o comando: sudo apt-get update && sudo apt-get upgrade.

5.2 Segundo passo

Agora, abra a pasta do Algencan através do terminal. No nosso caso seria: cd /home/erik/Desktop/algencan-3.1.1. Uma vez dentro da pasta, digite: make. Caso você não possua esse comando, o terminal irá te apontar como instalá-lo.

Para melhor organização do que deve ser feito em seguida, crie uma pasta dentro da pasta do Algencan. Se você estiver com a pasta do Algencan aberta no terminal, digite: mkdir myfolder. Abra a pasta digitando em seguida: cd myfolder.

Em seguida, copie o problema "toyprob.c"para myfolder: cp ../sources/examples/c/toyprob.c . (O ponto faz parte do comando). Estamos usando uma interface em C para montar o problema no Algencan, logo, pegamos esse problema exemplo em C como uma base.

5.3 Terceiro passo

Edite o programa em C, para resolver o seu problema. Para tal, é necessário entender as variáveis do problema e as sub-rotinas.

5.3.1 myevalf, myevalg e myevalh

"myevalf"será a primeira sub-rotina encontrada. Basta atribuir para *f a formula da sua função objetivo. Não mexa em *flag. Isso vale para todas as vezes em que a variável for instanciada.

Em seguida temos "myevalg", o gradiente de myevalf. Basta atribuir as respectivas derivadas parciais de myevalf a g[i]. Se o gradiente for muito grande, talvez seja útil saber que a variável n é o tamanho de x.

Por fim, temos "myevalh". O Algencan, para preservar memória, se aproveita da esparsidade das matrizes. No caso, estamos tratando do hessiano de myevalf. Primeiramente, deve-se definir o valor de *hnnz para o máximo de elementos não nulos na matriz hessiana. Para entender melhor como se define cada elemento, utilizemos um exemplo:

Digamos que temos $H \in \mathbb{R}^{2\times 2}$, $H = \begin{bmatrix} 2 & x_1 \\ x_1 & -1 \end{bmatrix}$.

Definimos $H_{1,1} = 2$, fazendo: hval[0] = 2; hrow[0] = 0; hcol[0] = 0.
Definimos $H_{1,2} = x_1 = H_{2,1}$, fazendo: hval[1] = x_1; hrow[2] = 1; hcol[1] = 1.
Definimos $H_{2,2} = -1$, fazendo: hval[3] = -1; hrow[2] = 3; hcol[2] = 3.

De forma genérica, definimos um valor $H_{n,m} = z$, fazendo: hval[i] = z; hrow[n] = i; hcol[m] = i. Além disso, note que não é preciso definir o triangulo superior da matriz, apenas o inferior.

5.3.2 myevalc, myevaljac, myevalhc

Essas sub-rotinas dizem respeito as restrições do problema de minimização. Em "myevalc", para cada uma das restrições, deve-se definir uma linha de comando da forma

```
if (ind == i) *c = /* alguma funcao de restricao */;
```

onde i é o índice da restrição. Além disso, não remova o comando que utiliza *flag.

Em "myevaljac"definimos o jacobiano das restrições. Dentro de cada linha de comando *if (ind == i)*, será necessário criar o gradiente de cada uma das funções de restrição. Isso é feito de forma semelhante a myevalg, porém é possível preservar memória, definindo *jcnnz como o total de elementos não nulos do gradiente da restrição. Então, com jcvar[i] = n, definimos que na posição n, o gradiente da restrição recebe o valor de jcval[i].

Em "myevalhc", definimos os hessianos das restrições. Para tal, dentro de cada *if (ind == i)*, defina os hessianos, como foi feito em myevalh.

5.3.3 main

Dentro de "main", defina as variáveis:

- n = tamanho do vetor x.
- m = número de restrições.
- l[i] = limite inferior do valor de x_i.
- u[i] = limite superior do valor de x_i.

- equatn[i] = se 1, a i-ésima restrição é de igualdade; se 0, a restrição é de desigualdade.

- linear[i] = se 1, a i-ésima restrição é linear; se 0, a restrição é não linear.

- jcnnzmax; hnnzmax = define o máximo de memória alocada para as matrizes esparsas. Sugerimos manter um valor alto.

- checkder = se 1, verifica se as derivadas estão corretas por diferenças finitas; se 0, não verifica. Recomendamos manter em 1.

- outputfnm = nome do texto de output. Se não quiser gravar o texto, deixe apenas “”.

- nvparam = se 1, imprime mensagens sobre a execução do programa; mantenha em 0 para que não imprima.

Definidas todas as variáveis e sub-rotinas, o programa pode ser executado.

5.4 Execução

Para executar o programa que você acabou de escrever/editar, abra a pasta myfolder com o terminal do Linux. Feito isso digite os seguintes comandos:

export ALGENCAN=/home/erik/Desktop/algencan-3.1.1
Uma vez que esse comando for executado, ele não precisa ser utilizado de novo até fechar o terminal. Em sequência digite:

gcc -O3 toyprob.c -L$ALGENCAN/lib -lalgencan -lgfortran -lm -o -algencan
Esse comando monta o programa. Toda vez que o programa for editado (não se esqueça de salvar antes), o usuário deve executá-lo. Em seguida, para rodar o programa, digite:

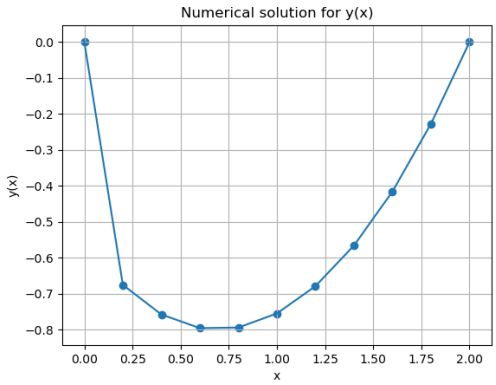
./-algencan
Os resultados aparecerão no próprio terminal do Linux. A interpretação dos resultados fica para o leitor (para o bom entendedor de inglês, não será difícil).

5.5 ALGENCAN vs. Nosso método

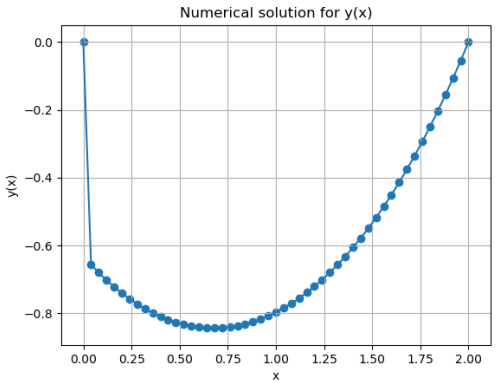
Vejamos por fim os resultados obtidos fazendo a implementação do problema em Algencan, para comparação dos resultados:

n	λ_1	λ_2	λ_3	$f(x)$	Tempo	Iterações
11	-1.11	-1.68	1.83	-1.55	0.01	50 (limite)
51	-1.30	-1.65	1.96	-1.80	0.01	14
101	-1.46	-1.51	1.61	-1.78	0.03	12
201	-1.48	-1.50	1.61	-1.80	0.56	15

Tabela 3: Resultados obtidos para diferentes valores de n com o ALGENCAN.

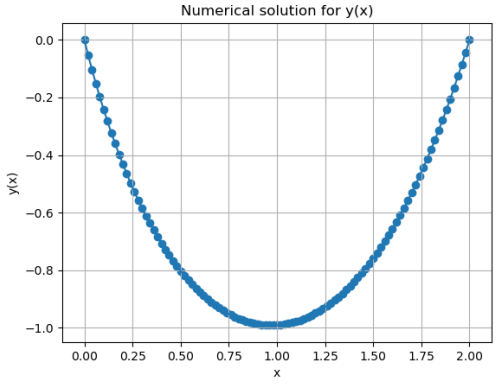


(a) $n = 11$

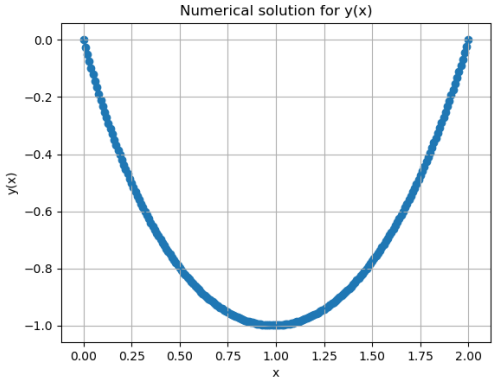


(b) $n = 51$

Figura 7: Gráficos das aproximações para $y(x)$ via ALGENCAN.



(a) $n = 101$



(b) $n = 201$

Figura 8: Gráficos das aproximações para $y(x)$ via ALGENCAN.

Podemos ver que os resultados obtidos pelo Algencan são muito semelhantes aos resultados que obtivemos, inclusive, geram a mesma "anomalia"numérica quando não há pontos suficientes para calcular $y\prime(x)$. Porém a diferença se faz clara na eficiência do método; o Algencan é capaz de obter os mesmos resultados (talvez mais precisos) com um número muito menor de iterações. Quanto ao tempo, é difícil afirmar algo, uma vez que o Algencan é programado em linguagens de baixo nível, e nossa implementação foi feita em Python.

Utilizando a plataforma Desmos podemos facilmente encontrar os parâmetros a, b e c que melhor aproximam a função $y(x) = a\cosh(\frac{x+b}{a}) + c$ dos pontos y obtidos através da minimização. Fizemos isso utilizando $n = 101$:

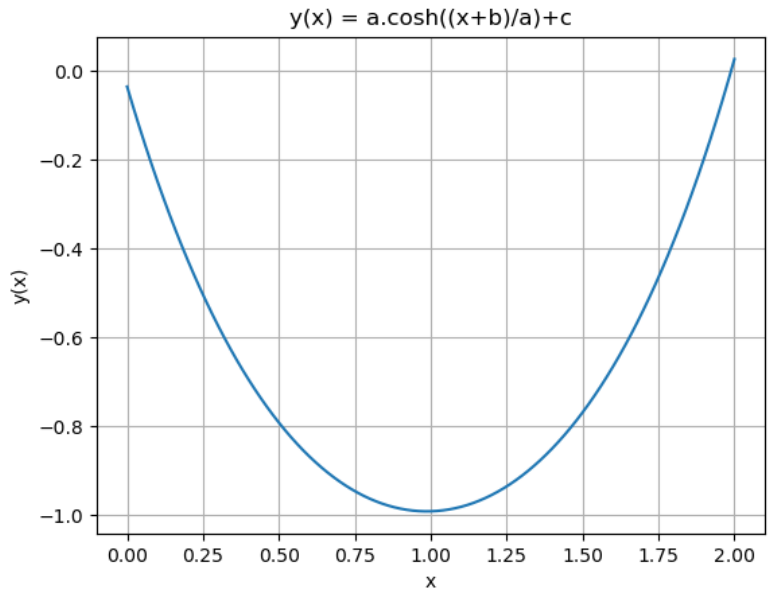


Figura 9: $a = 0.62$, $b = -0.98$ e $c = -1.61$.

$$\int_0^2 \sqrt{1 + (y\prime(x))^2}dx = 2.97$$

6 Considerações finais

O método que implementamos faz um bom trabalho em otimizar o problema posto, mas perde em eficiência se comparado ao Algencan. Além disso, ao observar os resultados, vemos que a forma como o problema é modelado tem grande importância, e entende-se que é necessário encontrar um modelo que faça bom balanco entre eficiência, estabilidade numérica e eficiência, velocidade.

Referências

[1] Ratliff, Nathan (2014).
Optimization II: Numerical Algorithms and Newton's Method.
Acessado em `https://ipvs.informatik.uni-stuttgart.de/mlr/wp-content/uploads/2014/12/mathematics_for_intelligent_systems_lecture8_notes.pdf`

[2] Jarre, Florian (2003).
Institute of Mathematics, University of Düsseldorf.
On an Approximation of the Hessian of the Lagrangian.
Acessado em `http://www.optimization-online.org/DB_FILE/2003/12/800.pdf`

[3] Griva, I. & Vanderbei, R. (2003).
Operations Research and Financial Engineering.
Princeton University.
Case Studies In Optimization: Catenary Problem
Acessado em `https://vanderbei.princeton.edu/tex/chain/chain.pdf`

[4] Nocedal, J. & Wright, S.
EECS Department Northwestern University & Computer Sciences Department University of Winsconsin.
Numerical Optimization, Second Edition.
Mathematics Subject Classification (2000): 90B30, 90C11, 90-01, 90-02