Métodos de regularização para programação não linear

Sérgio de A. Cipriano Júnior
Prof. Dr. John Lenon C. Gardenghi
Faculdade do Gama · Universidade de Brasília

Resumo

A estratégia de regularização de modelos, para a resolução de problemas de programação não linear, tem sido alvo de constante pesquisa atualmente. Neste trabalho, o objetivo é estudar o estado-da-arte do método de regularização adaptativa de ordem p+1 e sua implementação computacional, a fim de verificar sua eficácia frente à outros métodos clássicos, por meio de experimentação numérica. O presente relatório parte de uma apresentação à um problema de otimização e ao modelo de regularização, seguido de uma introdução ao método de regularização adaptativa e, por fim, experimentos numéricos para validar o algoritmo proposto.

Palavras-chave regularização, gradientes conjugados, minimização irrestrita, programação não linear.

1 Introdução

Neste trabalho, iremos abordar o método de regularização adaptativa de ordem p+1, AR_{p+1}, e sua implementação computacional, a fim de verificar sua eficácia frente aos outros métodos clássicos, por meio de experimentação numérica.

O presente trabalho foi organizado do seguinte modo: na Seção 2 é apresentado o modelo matemático do problema em questão, na Seção 3 introduzimos o algoritmo proposto e características da sua implementação computacional, em Julia; na Seção 4 apresentamos e analisamos os experimentos numéricos, comparando o método proposto com o Ipopt [8]; na última seção finalizamos com nossas conclusões.

2 Modelo matemático

Dado um problema de programação não linear na forma

$$\begin{aligned} &\text{Minimizar } f(x) \\ &\quad \text{s. a } c_i(x) = 0,\ i \in E, \\ &\qquad c_i(x) \leq 0,\ i \in I, \end{aligned} \tag{1}$$

onde:

- $f : \mathbb{R}^n \to \mathbb{R}$ é a função objetivo;
- $x \in \mathbb{R}^n$ é o vetor de variáveis;
- c_i são funções que definem condições as quais x deve satisfazer;

- E e I são, respectivamente, os conjuntos de índices i das restrições de igualdade e desigualdade.

Dizemos que o problema (1) é um problema de minimização sem restrições quando $E = I = \emptyset$. Métodos de regularização para minimização irrestrita consiste em sucessivas minimizações de um modelo de regularização definido por:

$$m_p(x, s, \sigma) = T_p(x, s) + \frac{\sigma}{p+1}||s||^{p+1}, \tag{2}$$

sendo:

- σ um escalar positivo utilizado como parâmetro de regularização;
- $s \in \mathbb{R}^n$ a direção que tomamos a partir de um ponto x_i;
- $p \geq 1$ um inteiro que define o grau de regularização;
- $T_p(x, s)$ o p-ésimo polinômio de Taylor da função objetivo, avaliada em $x+s$, descrito como

$$T_p(x, s) = f(x) + \sum_{j=1}^{p} \frac{1}{j!} \nabla_x^j f(x)[s]^j. \tag{3}$$

Nosso objetivo é solucionar o problema (1), no qual a função objetivo é suave e p vezes continuamente diferenciável em $\mathbb{R}^n$.

3 Método de Regularização Adaptativa

O método de regularização adaptativa de ordem p+1, tal qual demais métodos de otimização descritos em [2], resolve um subproblema a cada iteração, nesse caso, calcular um minimizador aproximado do modelo de regularização (2).

O algoritmo proposto 1, na fase de implementação, possui uma variação significativa em relação ao apresentado em [1]. O subproblema é solucionado utilizando o método de Newton com gradientes conjugados e busca linear retroativa, como apresentado em [2].

Algoritmo 1: $\mathbf{AR}_{p+1}$ - Método de Regularização Adaptativa de Ordem $p+1$

Entrada. Dado $x_0 \in \mathbb{R}^n$, $p \in \{1, 2, ...\}$, $\alpha > 0$, $\eta_1, \eta_2 > 0$, $\sigma_{low} > 0$, $\theta > 0$, $J \in \mathbb{N}$, $\gamma_1, \gamma_2 > 0$, uma tolerância $e > 0$ e $k_{max} > 1$. Inicializar $k \leftarrow 1$.

Passo 1. Inicializar $j \leftarrow 0$ e definir $\sigma_k = 0$.

Passo 2. Caso $||\nabla f(x)|| <= e$, significa que o algoritmo convergiu e, portanto, podemos parar a execução. Da mesma forma, se $k \leq k_{max}$, isso significa que o algoritmo atingiu o limite de iterações definido, logo, também interrompemos a execução.

Passo 3. Calcular s_k de forma que

$$m_p(x_k, s_k, \sigma_k) \leq m_p(x_k, 0, \sigma_k) \tag{4}$$

e

$$||\nabla_s m_p(x_k, s_k, \sigma_k)|| \leq \theta ||s_k||^p. \tag{5}$$

Caso $j = 0$ e atender aos critérios (4) e (5) não foi possível, definir $\sigma_k = \sigma_{low}$, atribuir $j \leftarrow 1$ e repetir o Passo 3.

Passo 4. Caso

$$j \geq J \text{ ou } \left(\frac{T_p(x_k, 0) - T_p(x_k, s_k)}{max\{1, |T_p(x_k, 0)|\}} \leq \eta_1 \text{ e } \frac{||s_k||_\infty}{max\{1, ||x_k||_\infty\}} \leq \eta_2 \right)$$

e

$$f(x_k + s_k) \leq f(x_k) - \alpha ||s_k||^{p+1},$$

Defina $x_{k+1} = x_k + s_k$ e

$$\sigma_{low} = \gamma_1 * \begin{cases} \sigma_{low} & \text{se } \sigma_k \text{ for nulo,} \\ \sigma_k & \text{caso contrário.} \end{cases}$$

Faça $k \leftarrow k + 1$ e volte para o Passo 1.

Passo 5. Defina $\sigma_{k+1} = \max\{\sigma_{low}, \gamma_2 \sigma_k\}$, faça $j \leftarrow j + 1$ e volte para o Passo 3.

Algumas observações em relação ao Passo 4, do Algoritmo 1, que define como a escolha do passo é feita. A condição

$$\frac{||s_k||_\infty}{\max\{1, ||x_k||_\infty\}} \leq \eta_2 \tag{6}$$

impõe que a função não seja avaliada em pontos $x_k + s_k$ distantes de x_k. Em contrapartida, a condição

$$\frac{T_p(x_k, 0) - T_p(x_k, s_k)}{\max\{1, |T_p(x_k, 0)|\}} \leq \eta_1 \tag{7}$$

descarta, sem avaliar $f(x_k + s_k)$, decréscimos muito maiores do que uma quantidade proporcional ao valor da função objetivo, avaliada em x_k. Isso é feito pois tais decréscimos sugerem que o modelo não é uma boa aproximação da função objetivo.

Ambas as condições (6) e (7) almejam descartar passos desnecessários, sem que seja necessário avaliar a função objetivo. São condições empíricas, sem resultados teóricos, tal qual exposto em [1]. Ademais, o número de vezes que esse controle sobre o passo pode ser feito a cada iteração k é definido pela variável J, um dos parâmetros do algoritmo proposto.

4 Experimentos numéricos

Tendo apresentado o algoritmo estudado, iremos mostrar alguns testes numéricos feitos utilizando uma implementação do AR_{p+1}, em Julia utilizando o padrão NLPModels [7], e problemas propostos por Moré, Garbow e Hillstrom [6], fornecidos pela banco de testes do CUTEst [5].

Todos os experimentos foram realizados numa arquitetura amd64, 64 bits, modelo Intel(R) Core(TM) i5-4210U 8GB RAM, frequência base de 1.70GHz, no sistema operacional Debian GNU/linux 11 (bullseye). As bibliotecas utilizadas do Julia 1.5.3 foram: CUTEst v0.7.0, NLPModels v0.10.1 e TimerOutputs v0.5.6.

As funções que aparecem em [6] estão na forma de problemas de quadrados mínimos, ou seja, definidas por $f_1, f_1, \ldots, f_m$. Podemos obter um problema de minimização irrestrita ao resolvermos uma função objetivo descrita como:

$$\min_{x \in \mathbb{R}^n} f(x) = \sum_{i=1}^{m} f_i^2(x) \tag{8}$$

Os testes foram divididos em 2 partes:

1. Na **Parte 1**, são apresentados resultados dos 35 problemas de minimização irrestrita propostos em [6] para o algoritmo AR_{p+1};

2. Na **Parte 2**, é feita uma comparação entre o método de AR_{p+1} e o Ipopt.

4.1 Parte 1

Nesta parte, serão apresentados os resultados dos 35 problemas propostos por Moré, Garbow e Hillstrom [6], obtidos por meio da implementação, em Julia [1], do algoritmo proposto (1). Para avaliar o desempenho dessa implementação, seguiremos os critérios apresentados na Tabela 1.

Sigla	Descrição
AF	Quantidade de avaliações da função objetivo.
IT	Quantidade de iterações do algoritmo principal (AR_{p+1}).
VF	Valor de $f(x^*)$ definido em (8), sendo x^* o valor do vetor solução.
VG	Valor de $\|f(x^*)\|$.
CP	Critério de parada.

Tabela 1: Descrição dos critérios de avaliação adotados.

Os critérios de parada utilizados na implementação foram:

1. Norma do gradiente menor que a tolerância $e = 10^{-6}$.

2. Algoritmo excedeu o limite de k_{max} iterações, sendo $k_{max} = 500$.

3. Tempo limite de 10 minutos, interrompe execução após a primeira iteração.

Nos testes, serão utilizadas as mesmas numerações, dimensões n e m (sempre que descritas), e pontos iniciais do artigo [6]. Ademais, serão criadas siglas para cada problema para facilitar sua referência, tal como apresentados na Tabela 2.

Função	Sigla	n	m	Ponto Inicial
1	ROS	2	2	$(-1.2, 1)$
2	FRF	2	2	$(0.5, -2)$
3	PBS	2	2	$(0, 1)$
4	BBS	2	3	$(1, 1)$
5	BEF	2	3	$(1, 1)$
6	JSF	2	10	$(0.3, 0.4)$
7	HVF	3	3	$(-1, 0, 0)$
8	BAF	3	15	$(1, 1, 1)$
9	GAUS	3	15	$(0.4, 1, 0)$
10	MEYE	3	16	$(0.02, 4000, 250)$
11	GULF	3	99	$(5, 2.5, 0.15)$
12	BOX3	3	10	$(0, 10, 20)$
13	PSF	4	4	$(3, -1, 0, 1)$
14	WOOD	4	11	$(-3, -1, -3, -1)$
15	KOF	4	11	$(0.25, 0.39, 0.415, 0.39)$
16	BDF	4	20	$(25, 5, -5, -1)$

[1] Acesse `https://github.com/sergiosacj/PIBIC-2020-2021/blob/main/algorithms/AR.jl`

17	OB1	5	33	$(0.5, 1.5, -1, 0.01, 0.02)$
18	BIG	6	13	$(1, 2, 1, 1, 1, 1)$
19	OB2	11	65	$(1.3, 0.65, 0.65, 0.7, 0.6, 3, 5, 7, 2, 4.5, 5.5)$
20	WATF	12	31	$(0, ..., 0)$
21	EROS	10	10	$(\xi_j$, onde $\xi_{2j-1} = -1.2$ e $\xi_{2j} = 1)$
22	EPSF	4	4	$(\xi_j$, onde $\xi_{4j-3} = 3$, $\xi_{4j-2} = -1$, $\xi_{4j-1} = 0$ e $\xi_{4j} = 1)$
23	PF1	4	5	$(\xi_j$, onde $\xi_j = j)$
24	PF2	4	8	$(\frac{1}{2}, ..., \frac{1}{2})$
25	VDIM	10	12	$(\xi_j$, onde $\xi_j = 1 - \frac{j}{n})$
26	TRIG	200	200	$(\frac{1}{n}, ..., \frac{1}{n})$
27	BALF	10	10	$(\frac{1}{2}, ..., \frac{1}{2})$
28	DBVF	12	12	$(\xi_j$, onde $\xi_j = t_j(t_j - 1))$
29	DIEF	50	50	$(\xi_j$, onde $\xi_j = t_j(t_j - 1))$
30	BTF	10	10	$(-1, ..., -1)$
31	BBF	10	10	$(-1, ..., -1)$
32	LFFR	200	400	$(1, ..., 1)$
33	LFR1	200	400	$(1, ..., 1)$
34	LFRZ	200	400	$(1, ..., 1)$
35	CHEB	10	10	$(\xi_j$, onde $\xi_j = \frac{j}{n+1})$

Tabela 2: Problemas do Moré, Garbow e Hillstrom.

Os parâmetros fornecidos ao algoritmo foram retirados do artigo [1]. Os dados estão descritos na Tabela 3.

Variável	Valor atribuído
p	2
e	1 E-6
α	1 E-8
η_1	1000
η_2	3
σ_{low}	1 E-8
θ	100
J	20
γ_1	0.5
γ_2	10
k_{max}	500

Tabela 3: Valores utilizados como parâmetro para a implementação do AR_{p+1}.

Segue os resultados dos testes do algoritmo AR_{p+1} na Tabela 4.

Função	VF	VG	AF	IT	CP
ROS	1.343529 E-30	4.905578 E-14	20	10	1
FRF	−1.909305 E+08	1.386202 E+05	8	2	3
PBS	0.000000 E+00	0.000000 E+00	0	1	1
BBS	1.681558 E-44	2.548848 E-32	90	41	1
BEF	2.418904 E-19	3.580280 E-09	17	8	1
JSF	1.243622 E+02	2.737942 E-12	11	12	1
HVF	2.255008 E-23	5.575461 E-11	25	11	1

BAF	8.214877 E-03	5.247444 E-11	22	12	1
GAUS	0.000000 E+00	0.000000 E+00	0	1	1
MEYE	−1.069394 E+11	2.443458 E+07	34	2	3
GULF	3.283500 E+01	3.683667 E-10	11	3	1
BOX3	8.361206 E-19	3.761003 E-10	14	9	1
PSF	5.486734 E-10	1.955982 E-17	21	22	1
WOOD	2.177970 E-24	2.481384 E-02	38	29	1
KOF	1.035413 E-03	4.076510 E-09	152	68	1
BDF	8.582220 E+04	3.335801 E-10	11	10	1
OB1	1.309631 E+119	8.381631 E+121	24	7	3
BIG	−1.565695 E+35	2.737745 E+29	185	67	3
OB2	8.224600 E+108	1.085316 E+110	11	3	3
WATF	−2.586449 E-09	7.756991 E-09	49	22	1
EROS	1.244921 E-26	4.984938 E-12	6	7	1
EPSF	0.000000 E+00	0.000000 E+00	0	0	1
PF1	5.655597 E+13	1.047473 E-05	10	6	3
PF2	4.711628 E+13	1.475497 E-11	13	12	1
VDIM	5.368568 E-29	3.295856 E-17	47	31	1
TRIG	0.000000 E+00	0.000000 E+00	0	1	1
BALF	−6.013884 E+06	2.194623 E-01	21	4	3
DBVF	0.000000 E+00	0.000000 E+00	0	1	1
DIEF	0.000000 E+00	0.000000 E+00	0	1	1
BTF	0.000000 E+00	0.000000 E+00	0	1	1
BBF	7.012296 E+04	5.957036 E-05	11	2	3
LFFR	0.000000 E+00	0.000000 E+00	0	1	1
LFR1	2.000000 E+02	5.792169 E-13	2	3	1
LFR2	1.020086 E+02	4.746693 E-01	8	2	3
CHEB	−1.909058 E+01	2.895000 E-07	14	2	3

Tabela 4: Resultados dos 35 problemas para o algoritmo AR_{p+1} com Newton-MGC.

Diante desses resultados, podemos fazer algumas observações:

1. Ao todo 10 testes foram interrompidos pelo limite de tempo.
2. É possível ver que alguns testes já são finalizados na primeira iteração. Esses testes já satisfazem o segundo critério de parada antes mesmo de qualquer avaliação ser realizada e, portanto, não ofereceram muitas informações[2].

4.2 Parte 2

Nesta parte, serão comparados os resultados de todos os 35 problemas entre o algoritmo ARp_{p+1}, em Julia, e a interface do Ipopt [8] para Julia, Ipopt.jl, fornecida pela linguagem de modelagem JuMP [4].

É importante enfatizar que os algoritmos serão executados nas mesmas condições, isto é, com os mesmos critérios de parada e utilizando o mesmo ambiente de execução. Os critérios serão os mesmos apresentados na Tabela 1.

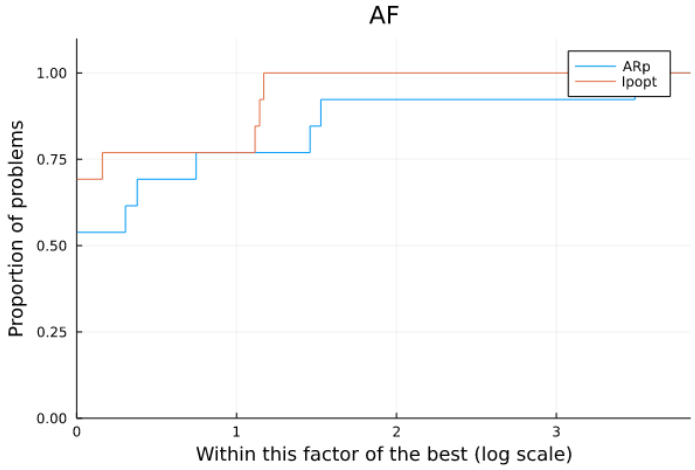
[2]Informações detalhadas de cada teste estão disponíveis em `https://github.com/sergiosacj/PIBIC-2020-2021/tree/main/Tests/ARp`

A primeira etapa é de comparar a quantidade de problemas resolvidos por cada algoritmo. O ARp$_{p+1}$ e o Ipopt resolveram, respectivamente, 24 e 27 problemas [3]. Essa comparação envolve verificar o seguinte critério:

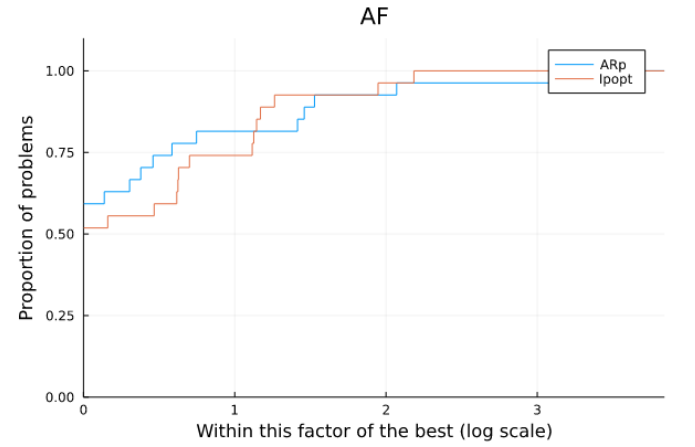
$$\frac{fm - fmin}{max(1, abs(fmin))} \leq e, \tag{9}$$

onde fm é o valor final da função objetivo, isto é, quando avaliada em $f(x^*)$, sendo x^* o k-ésimo ponto calculado pelo algoritmo em questão; e $fmin$ é o $min(fm_{ARp_{p+1}}, fm_{Ipopt})$.

A segunda etapa é uma comparação gráfica entre os algoritmos com o uso de perfis de desempenho [3]. Foram criados dois perfis de desempenho, ilustrados na Figura 1.



(a) Avaliações de Função para todos os problemas que ambos algoritmos resolvem.



(b) Avaliações de Função para todos os problemas em que AF é não nulo.

Figura 1: Perfis de desempenho resultantes de testes com os 35 problemas de Moré, Garbow e Hillstrom.

Esses resultados nos permitem fazer as seguintes avaliações:

1. Na Figura 1(a) é possível notar que o Ipopt apresenta melhores resultados quando levamos em conta somente os problemas em que ambos os algoritmos satisfizeram

[3]Mais detalhes disponíveis em `https://github.com/sergiosacj/PIBIC-2020-2021/blob/main/Tests/outputComparation.txt`

o critério (9). No entanto, conforme apresenta a Figura 1(b), o ARp$_{p+1}$ também demonstra-se robusto quando avaliamos todos os resultados em que os testes não são finalizados na primeira iteração.

5 Conclusão

Neste trabalho foi apresentado, implementado e testado o método de regularização adaptativa de ordem $p + 1$. Tal método possui forte desenvolvimento teórico, entretanto não possui implementação prática na literatura [1]. A escolha da linguagem Julia foi fator decisivo, por se tratar de uma linguagem em amplo desenvolvimento, principalmente para comunidade de otimização [4, 7].

Com os experimentos numéricos é possível observar que há espaço para melhorias e ajustes no algoritmo proposto. Além disso, tanto o algoritmo AR$_{p+1}$, quanto o Newton-MGC, utilizado para solucionar o subproblema, mostraram-se um campo fértil para avanços.

Referências

[1] E. G. Birgin, J. L. Gardenghi, J. M. Martínez, e S. A. Santos. On the use of third-order models with fourth-order regularization for unconstrained optimization. *Optimization Letters*, 2019.

[2] S. A. Cipriano Jr. e J. L. Gardenghi. Tópicos em programação não linear e álgebra linear computacional. Relatório técnico, Universidade de Brasília, 2020.

[3] E. D. Dolan e J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.

[4] Iain Dunning, Joey Huchette, e Miles Lubin. Jump: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320, 2017.

[5] N. I. M. Gould, D. Orban, e Ph. L. Toint. CUTEst: a constrained and unconstrained testing environment with safe threads for mathematical optimization. *Computational Optimization and Applications*, 60(3):545–557, 2014.

[6] J. J. Moré, B. S. Garbow, e K. E. Hillstrom. Testing unconstrained optimization software. *ACM Transactions on Mathematical Software (TOMS)*, 7(1):17–41, 1981.

[7] D. Orban e A. S. Siqueira. NLPModels.jl: Data structures for optimization models, 2020. Disponível em: `https://github.com/JuliaSmoothOptimizers/NLPModels.jl`. Acesso em: 30 nov. 2020.

[8] A. Wächter e L.T. Biegler. On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 2004.