Solução do problema de mínimos quadrados com uso de matrizes de reflexão

Bruno Sesso 8536002
Gustavo Estrela de Matos 8536051

1 de Dezembro de 2014

1 Introdução

1.1 Apresentação do problema de mínimos quadrados

O problema de mínimos quadrados é parte do problema de achar uma solução para o sistema linear $Ax = b$, onde A é uma matriz e x e b são vetores. No caso em que $A \in \mathbf{R}^{n \times n}$, $x \in \mathbf{R}^n$ e $b \in \mathbf{R}^n$ conhecemos alguns métodos, como a *Decomposição LU* ou a *Decomposição de Cholesky*, para calcular um x que satisfaça a equação. Nesses casos o problema é reduzido em resolver dois sistemas lineares nos quais a matriz em questão é triangular.
No entanto o principal problema, concentra-se no caso onde $A \in \mathbf{R}^{n \times m}$, $x \in \mathbf{R}^m$, $b \in \mathbf{R}^n$ e $n \neq m$. Chamamos o sistema de *sobredeterminado* se $n > m$ e *subdeterminado* caso $n < m$.
Em um sistema subdeterminado há infinitas soluções que satisfazem o sistema. Já no sistema sobredeterminado não há nenhuma solução exata para a qual $Ax = b$. Nesse caso queremos achar o x mais *próximo* da solução, isso é, minimizar $\|Ax - b\|_2$.

1.2 Objetivos do trabalho

Agora que conhecemos o problema de mínimos quadrados, queremos implementar um algoritmo em C que dado uma matriz $A \in \mathbf{R}^{n \times m}$ e um vetor $b \in \mathbf{R}^n$ nos apresente um $x \in \mathbf{R}^m$ que resolva ou que aproxime ao máximo o sistema $Ax = b$. Queremos que isso seja feita de forma eficiente e precisa (levando em conta o problema de representação dos números no computador).

2 Geração do problema

2.1 Como gerar o problema

Os problemas de mínimos quadrados solucionados e gerados nesse trabalho são aplicações do método para achar polinômios que, quando traçados, minimizam a distância euclidiana de pontos do $\mathbf{R}^2$ até a curva do polinômio. Para desenvolver o problema, seguimos os seguintes passos:

- Geramos um polinômio aleatório de grau k com coeficientes reais entre zero e um.
- Geramos n pontos aleatórios entre zero e um.
- Calculamos o valor do polinômio nesses pontos. Esse será nosso vetor b
- Construimos a matriz A, da forma descrita em 2.1.1

2.1.1 Construindo o sistema

Sejam $t_1, t_2, ..., t_n$ pontos de $\mathbf{R}[\mathbf{0}, \mathbf{1}]$ gerados aleatóriamente; $p(x)$ um polinômio com coeficientes reais em $\mathbf{R}[\mathbf{0}, \mathbf{1}]$ e $b = \begin{bmatrix} p(t_1) & p(t_2) & ... & p(t_n) \end{bmatrix}^T$. Se tivermos em mãos os pontos $t_1, t_2, ..., t_n$ e o vetor b, somos capazes de contruir o sistema sobredeterminado $Ax = b$ tal que

$$A = \begin{pmatrix} \phi_1(t_1) & \phi_2(t_1) & \cdots & \phi_m(t_1) \\ \phi_1(t_2) & \phi_2(t_2) & \cdots & \phi_m(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(t_n) & \phi_2(t_n) & \cdots & \phi_m(t_n) \end{pmatrix}$$

e $x = \begin{bmatrix} x_1 & x_2 & ... & x_m \end{bmatrix}^T$ onde $\phi_i(t) = t^i$. Portanto, temos que x, a solução para esse sistema sobredeterminado (se $n > m$), são os coeficientes do polinômio que minimiza a distância euclidiana entre os pontos $t_1, ..., t_n$ e a curva de tal polinômio no $\mathbf{R}^2$.

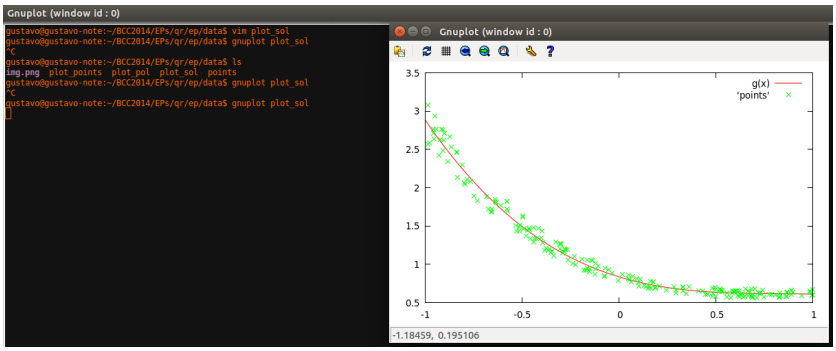
Para testar a eficácia de nosso solucionador de problemas de quadrados mínimos, após produzir nossa matriz A, perturbamos o vetor b. Para isso, definimos um "coeficiente de pertubação", c, que vale exatamente 1%. Percorremos, então nosso vetor b e somamos a cada b_i uma constante pertencente ao intervalo $[-c * b_i, c * b_i]$

2.2 Gerando gráficos do problema

Ao gerar um problema de mínimos quadrados, salvamos os dados gerados em arquivos, possibilitando a visualização desses dados em gráficos feitos pelo programa $gnuplot$. A cada vez que o gerador de problemas é rodado, são criados os seguintes arquivos no diretório $/data$:

- plot_points: script em gnuplot que mostra na tela os pontos gerados, ou seja, os elementos de b_i.
- plot_pol: script em gnuplot que mostra na tela o polinômio aleatório $f(x)$ gerador dos pontos presentes em b_i
- plot_sol: script em gnuplot que mostra na tela o polinômio resposta para o problema de mínimos quadrados.
- points: pontos do $\mathbf{R}^2$ com coordenadas $(t_i,\ f(t_i) \sim b_i)$.

Para ver qualquer um dos gráficos use: $gnuplot\ nome\_do\_arqvivo$



2.3 Uso do gerador de problemas

Para compilar o gerador de problemas use: *gcc gen_pmq.c -o gen_pmq*

Para gerar um problema use: *./gen_pmq [número de linhas LI na matriz A do sistema] [grau do polinômio gerador do problema] [quantidade de pontos gerados] [grau do polinomio utilizado para aproximar sua curva aos pontos] [semente do gerador aleatório]*

É importante observar que a utilização de uma mesma semente permite gerar várias vezes o mesmo problema.

3 Conceitos fundamentais

Temos um problema em mãos e agora precisamos soluciona-lo. Não podemos utilizar nenhum dos algoritmos até então conhecidos (LU, Cholesky, etc.), pois esses exigem que A seja quadrada. Precisamos, então, de outro algoritmo que possa resolver nosso problema.

3.1 Decomposição QR

A *Decomposição QR* consiste em decompor A em $Q \in \mathbf{R}^{n \times n}$ e $R \in \mathbf{R}^{n \times m}$ tais que Q é ortogonal simétrica e R é triangular superior. Nesse caso, nosso sistema fica:

$$Ax = b \Leftrightarrow QRx = b$$

Como Q é ortogonal, sabemos que $Q^{-1} = Q^T$. Logo:

$$Rx = Q^T b = c$$

O problema é reduzido em resolver um sistema triangular superior, que pode ser feito facilmente utilizando *substituição para trás*. Nos resta saber como encontrar Q e R.

3.2 Decomposição QR com reflexões

Para realizar a decomposição QR, queremos uma matriz Q_i tal que ao multiplica-la por um vetor v, ele seja transformado em um multiplo de um vetor canônico, tal que a constante que multiplica esse vetor seja $\pm\|v\|$. Isso é, se temos $v = \begin{bmatrix}1 & 1 & 1 & 1\end{bmatrix}^T$, queremos $Q_i v = \begin{bmatrix}2 & 0 & 0 & 0\end{bmatrix}^T$, por exemplo.
Tendo tal matriz, podemos multiplicar Q_1 por A para obtermos uma matriz da forma:

$$Q_1 A = A' = \begin{pmatrix} \sigma_1 & a_{1,2}^{(1)} & \cdots & a_{1,m}^{(1)} \\ 0 & a_{2,2}^{(1)} & \cdots & a_{2,m}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & a_{n,2}^{(1)} & \cdots & a_{n,m}^{(1)} \end{pmatrix}$$

Semelhantemente fazemos isso com as demais colunas, de forma que após multiplicar m matrizes tenhamos:

$$Q_m Q_{m-1} \cdots Q_2 Q_1 A = R = \begin{pmatrix} \sigma_1 & a_{1,2}^{(m)} & \cdots & a_{1,m-1}^{(m)} & a_{1,m}^{(m)} \\ 0 & \sigma_2 & \cdots & a_{2,m-1}^{(m)} & a_{2,m}^{(m)} \\ 0 & 0 & \cdots & a_{3,m-1}^{m} & a_{3,m}^{(m)} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & \sigma_m \\ 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 \end{pmatrix}$$

Onde:

$$\sigma_i = \sqrt{\sum_{k=i}^{n} (a_{k,i}^{(i-1)})^2}$$

$$Q_i = I - \frac{2uu^T}{\|u\|_2^2}, u = A_i - \sigma_i e_i$$

(A_i é a i-ésima coluna e e_i é a i-ésima base canonica)

4 Solução do problema

Agora com as ferramentas certas, podemos começar a implementar o algoritmo. Para implementar o algoritmo de *decomposição QR*, pensamos primeiramente no seguinte simples pseudo-código:

for *j = 0 .. m - 1* **do**
  $max = max\{A_{i,j}\}$, i = j + 1 .. n - 1
  $\sigma_j = \sqrt{\sum_{i=j}^{n-1} \frac{A_{i,j}^2}{max^2}}$
  if $A_{j,j} < 0$ **then**
    $\sigma_j = -\sigma_j$
  end
  $A_{i,j} += \sigma_j$
  $\gamma = \frac{1}{\sigma_j A_{j,j}}$
  $\sigma_j * = max$
  for *k = j + 1 .. m - 1* **do**
    $\alpha = \sum_{i=j}^{n-1} A_{i,j} A_{i,k}$
    $b_i -= \gamma \beta A_{i,j}$, i = j .. n - 1
  end
  $\beta = \sum_{i=j}^{n-1} A_{i,j} b_i$
  $A_{i,k} -= \gamma \alpha A_{i,j}$, i = j .. n - 1
  $\sigma_j = -\sigma_j$
end

Nessa primeira versão, a matriz Q e R são armazenadas em A e temos um vetor σ para armazenar a diagonal principal de R. Também levamos em conta a possibilidade do quadrado das normas não ser representavel, mas a norma for. Por isso dividimos os elementos de A_j por max e só então calculamos a sua norma.
Logo notamos que é possível que σ possa vir a ser 0 acarretando futuramente em uma divisão por 0. Para evitar isso, fizemos *pivoteamento de colunas*. Isso consiste em ordenar as colunas de A de forma a sempre ter a de maior norma mais a esquerda.
Implementamos então o seguinte código que prepara a matriz A para ser colocada no algoritmo acima:

for *k = 0 .. m - 1* **do**
  $maxnorm = max\{\sum_{i=k} n - 1 A_{i,j}\}$, j = k .. m - 1
  $swapcolumns(A, maxnorm_{index}, k)$
end

Aparentemente, todos os problemas resolvidos. De fato o código acima funciona, mas são feitos muitos cálculos de normas sem necessidade. Calculemos os flops realizados no total pelo algoritmo.

Para a permutação de colunas:

$$\sum_{k=0}^{m-1}\sum_{j=k}^{m-1}\sum_{i=k}^{n-1} 1 \approx \frac{m^2 n}{2} - \frac{m^3}{6}$$

Multiplicar Q_j por A_j:

$$\sum_{j=0}^{m-1}\sum_{k=j+1}^{m-1}\sum_{i=j}^{n-1} 2 \approx nm^2 - \frac{m^3}{3}$$

Total:

$$Total \approx \frac{3nm^2 - m^3}{2}$$

Para esse valor ser melhorado, passamos a permutar as colunas junto com o código do cálculo da QR. Primeiramente calcula-se as normas de todas as colunas e as armazenamos em σ. A cada iteração do código que calcula QR atualiza as normas em σ levando em conta que avançaremos para a próxima linha e as normas serão de tal linha até n.
A nova versão começa calculando as normas de cada coluna:

for *j = 0 .. m - 1* **do**
  $max = max\{A_{i,j}\}$, j = 1 .. n - 1
  if $|max| < \epsilon$ **then**
    $\sigma_j = 0$
    $continue$
  end
  $\sigma_j = sqrt\sum_{i=0}^{n-1} \frac{A_{i,j}^2}{max^2}$
  $\sigma_j* = max$
end

Isso leva em torno de:

$$\sum_{j=0}^{m-1}\sum_{i=0}^{n-1} 4 = 4nm$$

O novo cálculo de QR muda para:

$rank = 0$ **for** *j = 0 .. m - 1* **do**
  $max = max\{\sigma_k\}$, k = j .. m - 1
  if $|max| < \epsilon$ **then**
    $print$ (Matriz possui posto incompleto)
    return $rank$
  end
  if $|max_{index}| \neq j$ **then**
    $swapcolumns(A, max_{index}, j)$
    $swap(\sigma_{max_{index}}, \sigma_j)$
  end
  if $A_{j,j} < 0$ **then**
    $\sigma_j = -\sigma_j$
  end
  $A_{i,j} += \sigma_j$
  $\gamma = \frac{1}{\sigma_j A_{j,j}}$
  for *k = j + 1 .. m - 1* **do**
    $\alpha = \sum_{i=j}^{n-1} A_{i,j} A_{i,k}$
    $b_i -= \gamma\beta A_{i,j}$, i = j .. n - 1
  end
  $\beta = \sum_{i=j}^{n-1} A_{i,j} b_i$
  $A_{i,k} -= \gamma\alpha A_{i,j}$, i = j .. n - 1
  $\sigma_j = -\sigma_j$
  $\sigma_k = \sqrt{\sigma_k^2 - A_{j,k}^2, k = j+1..m}$
  $rank++$
end
return $rank$

Esse novo algoritmo apresenta uma eficiência bem melhor:

$$\sum_{j=0}^{m-1} \sum_{k=j+1}^{m-1} \sum_{i=j}^{n-1} 2 \approx nm^2 - \frac{m^3}{3}$$

Agora que temos um algoritmo melhorado de *Decomposição* QR temos que resolver o sistema, dado a decomposição que calculamos. Isso pode parecer simples, mas temos que ter cuidado, pois A pode ter posto incompleto e por consequência, R também. Nos próximos ítens explicamos a estratégia adotada para o caso de posto completo e incompleto.

4.1 Sistema com posto completo

Quando estamos resolvendo um sistema com A possuindo posto completo, temos que $A \in \mathbf{R}^{n \times m}$ com m colunas LI, ou seja, ao decompor $A = QR$ teremos que R é uma matriz triangular superior com posto completo igual a m. Portanto R uma matrix triangular superior e com m linhas LI.

Notemos que:

$$QA = R = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$$

Logo, R_1 triangular superior e não-singular.

Portanto, nosso problema de minimizar $\|Ax - b\|_2$ é o mesmo que minimizar $\|QAx - Qb\|_2$, isso porque Q é ortogonal e $\|Q|_2$ = 1. Logo, considerando que $Qb = c$:

$$\begin{pmatrix} R_1 \\ 0 \end{pmatrix}$$

* $\begin{bmatrix} x_1 & x_2 \end{bmatrix} = \begin{bmatrix} c_1 & c_2 \end{bmatrix}$

Assim, basta minimizar $\|R_1 x_1 - c_1\|_2$ + $\|c_2\|_2$. Mas como R_1 é triangular superior e não singular, basta achar x_1 tal que $R_{11} x_1 = c_1$.

Observe que não podemos fazer nada com o termo $\|b_2\|_2$. Este será o resíduo para nossa solução

4.2 Sistema com posto incompleto

Quando a matriz A do sistema possui posto $posto(A) = k < m$ continua existindo a decomposição QR, porém sua solução é um pouco diferente:

$QA = R =$

$$\begin{pmatrix} R_{11} & R_{12} \\ 0 & 0 \end{pmatrix}$$

Neste caso, a matriz R_{11} é triangular superior com dimensões kxk, portanto não singular; e R_{12} uma matriz qualquer. Considerando $Qb = c$ podemos rescrever nosso sistema $QAx = Qb$, e obtemos:

$$\begin{pmatrix} R_{11} & R_{12} \\ 0 & 0 \end{pmatrix}$$

* $\begin{bmatrix} x_1 & x_2 \end{bmatrix} = \begin{bmatrix} c_1 & c_2 \end{bmatrix}$ Assim, basta minimizar $\|R_{11} x_1 + R_{12} x_2 - c_1\|_2$ + $\|c_2\|_2$.

Observe que, R_{11} triangular superior e não-singular, logo basta escolher qualquer valor para x_2 e resolvemos o sistema $R_{11}x_1 = c_1 - R_{12}x_2$. Em nosso código, escolhemos x_2 mais trivial possível, vetor nulo. Portanto, bastou resolver o sistema $R_{11}x_1 = c_1$.

Novamente, não é possível interferir no termo c_2. Este será novamente o nosso resíduo.

É importante, para o entendimento da próxima sessão que o termo c_2 sendo o resíduo é a nossa principal métrica para definir se possuimos, ou não, uma resposta boa para o problema de mínimos quadrados que estamos tentando resolver.

5 Resultados

Para verificar o funcionamento, eficácia de nosso algoritmo, usamos $gnuplot$ e os scripts de ”plot”feitos pelo gerador de problemas. Além disso, verificamos sempre o resíduo da resposta obtida após a solução do problema.

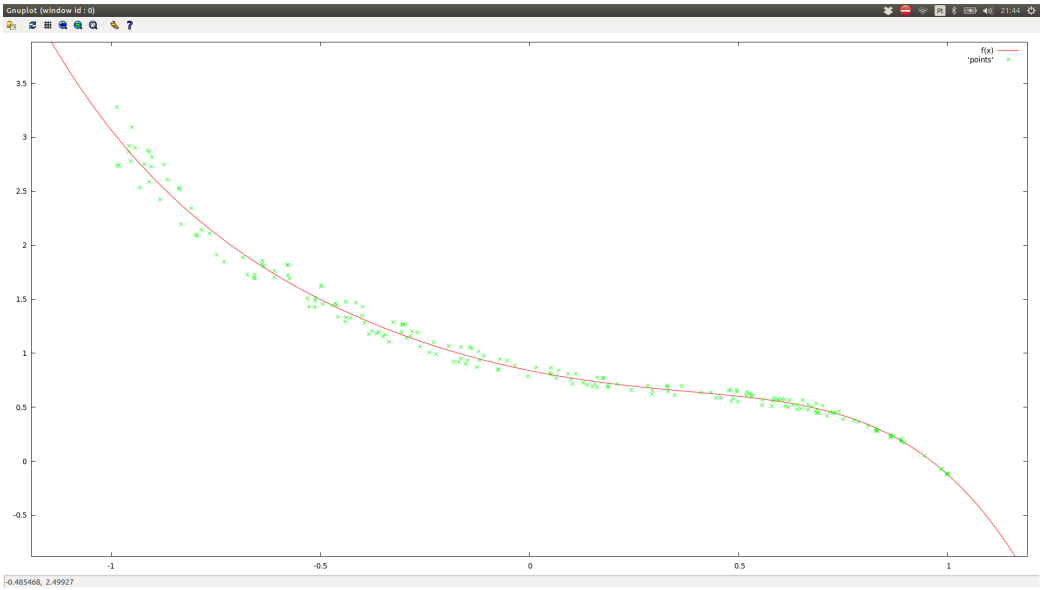
5.1 Gráficos do polinomio e respectivas respostas

Veja abaixo gráficos que com as respostas obtidas pelo nosso solucionador de problemas QR.

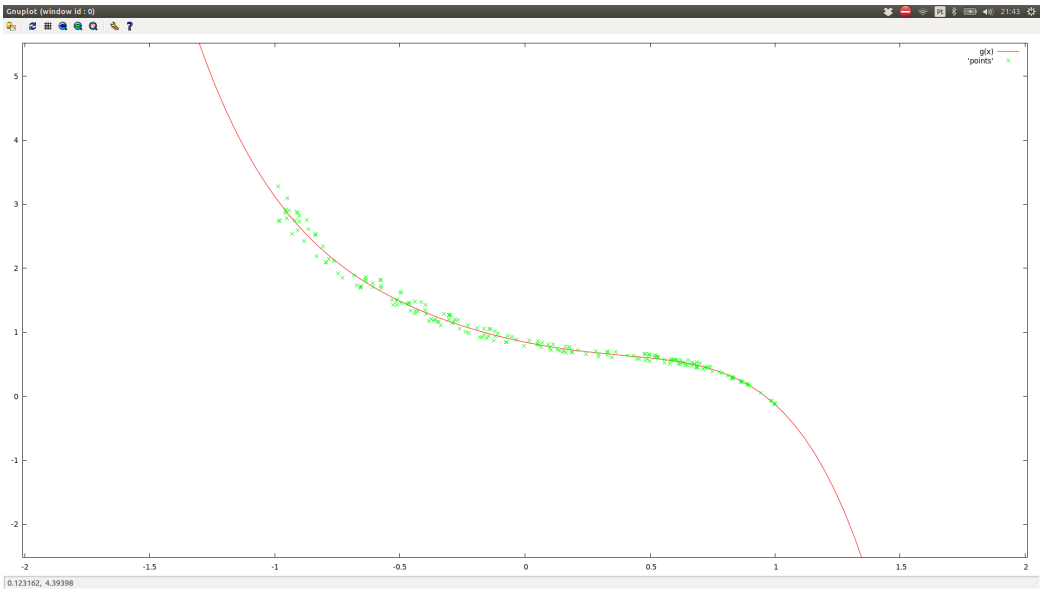
5.1.1 Matriz quadrada com posto completo

Gráficos do problema gerado por uso de $./gen\_pmq\ 200\ 5\ 200\ 1$

Problema gerado:



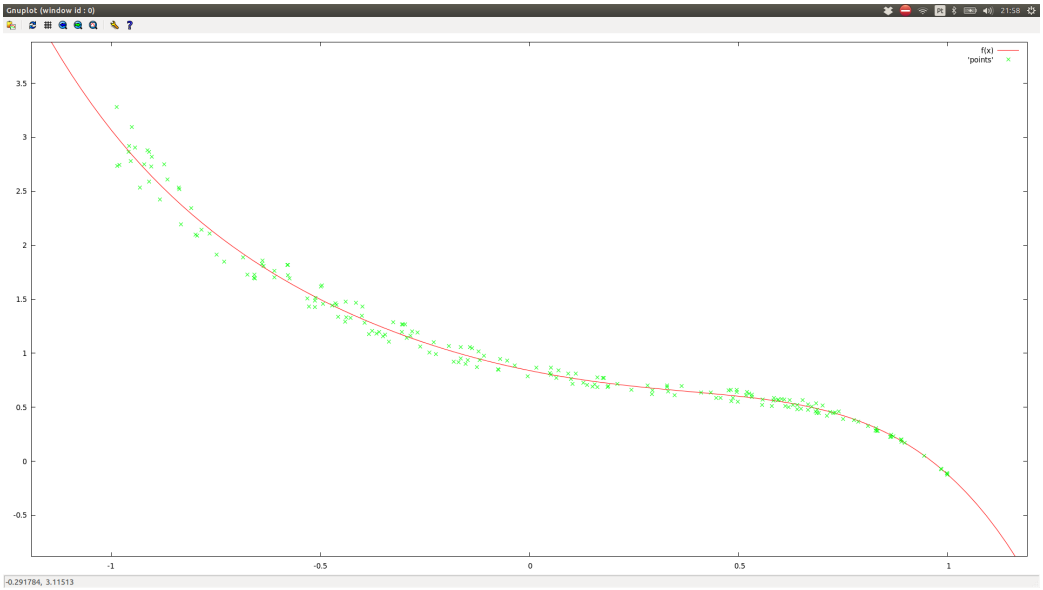
Resposta gerada:



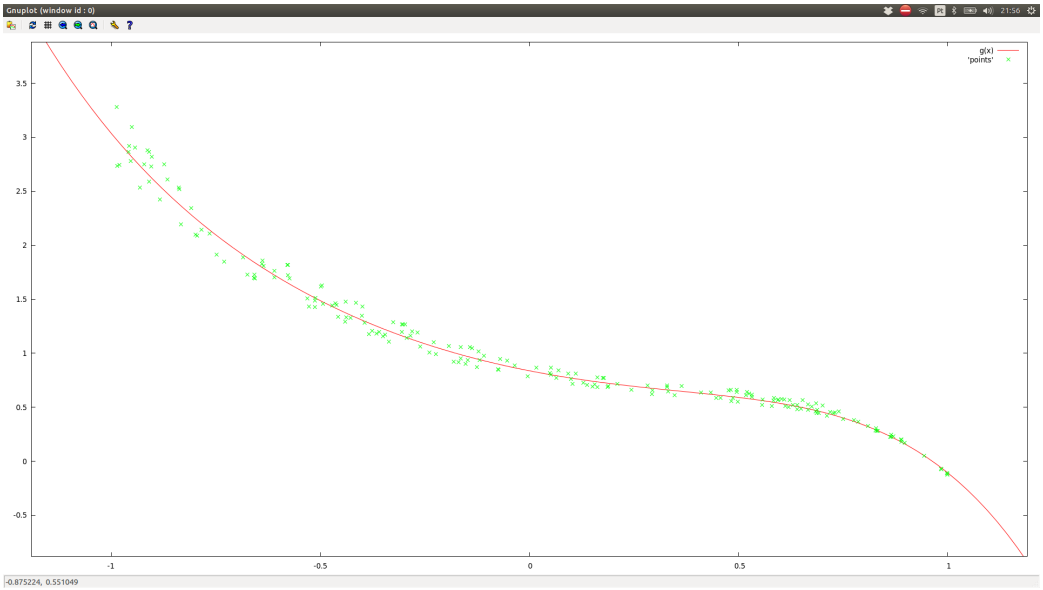
5.1.2 Problemas de posto incompleto

Gráficos do problema gerado por uso de $./gen\_pmq\ 100\ 5\ 200\ 1$

Problema gerado:



Resposta gerada:



5.2 Resíduo

Quando solucionamos um problema de mínimos quadrados $Ax = b$, é possível determinar o resíduo $r = \|Ax - b\|_2$. Observando como a norma

do resíduo é calculada na secção 4, é de se esperar, que ao aumentarmos o posto da matriz A de nosso problema, o resíduo r deve ter norma menor.

Para uso de *./gen_pmq 50 5 100 5 1* foi verificado resíduo de norma 9.63, enquanto que para o uso de *./gen_pmq 90 5 100 5 1* foi verificado resíduo de norma 5.93.

Confira a tabela abaixo com valores variados:

Argumentos	Resíduo
200 6 300 5 1	13.748073
290 6 300 5 1	4.215362
400 5 600 3 1	18.441638
550 5 600 3 1	9.693304
100 3 600 3 1	29.258007
600 3 600 3 1	1.786309

6 Conclusão

Ao longo do desenvolvimento desse trabalho enfrentamos diversas dificuldades que nos levaram ao código final e entendimento da solução do problema de mínimos quadrados com o uso de matrizes de reflexão.

Um dos primeiros desafios foi a construção do nosso gerador de problemas. Apesar de sua construção não ser tão complexa, procuramos, ao máximo, deixar nosso gerador flexível para que pudéssemos avançar no solucionador do problema. Além disso, nos preocupamos com o espaço de polinômios que poderiam ser gerados, bem como os possíveis pontos.

Nosso solucionador de problemas foi desnvolvido em várias etapas. Começamos seu desenvolvimento com um algoritmo bobo, e com paciência e acompanhamento de aulas e do capítulo 3 do livro ”Fundamentals of Matrix Computations”pudemos construir o código final. Fazer o código ser orientado por linha também foi um problema. Tentamos de diversas maneiras construí-lo sem precisar de um vetor auxiliar, mas acabamos concluindo que não era possível fazer tal código.

Ao perceber que os dados que precisáva-mos tratar eram muito grande, decidimos que tanto nosso gerador quanto nosso solucionador de problemas deveriam gerar scripts em $gnuplot$ para facilitar a visualização de

nossos problemas e resultados. A ferramenta $gnuplot$ foi de extrema importância e o desenvolvimento desse trabalho sem o seu auxílio seria muito mais difícil.

Por fim, com o desenvolvimento desse código, podemos perceber com os mínimos detalhes a solução de problemas de quadrados mínimos com matrizes de reflexão.

Referências

[1] Watkins, D.S.. ”Fundamentals of Matrix Computations.”, Second Edition, Wiley–Interscience, New York, 2002.