# MAP2210 - P3Q1 - Relatório

Lucas Panfilo Donaire NºUSP: 12556552
Beatriz Campanha Silva N°USP: 12676657
**Professor:** Alexandre Roma

IME-USP 2022

## Sumário



# 1 Objetivo

Queremos comparar a performance numérica em termos do tempo de execução e da qualidade da solução fornecida por três algoritimos de fatoração $QR$, sendo eles por Gram-Schmidt Clássico, Gram-Schmidt Modificado e Householder.

# 2 Implementações

Implementamos Gram-Schmidt Clássico e Gram-Schmidt Modificado a partir do código em Matlab disponível no *blog* [1], reescrevendo-os em Python.

E a implementação da reflexão e fatoração de Householder foi baseada em um código disponível na referência [2].

## 2.1 Gram-Schmidt Clássico

Dada uma matriz $X$ de vetores linearmente independentes, cada vetor é ortogonalizado em relação ao anterior. Ou seja, para cada vetor, é retirada a contribuição (ou projeção) dos anteriores em relação a este.

Assim, dados $a_1, a_2, \ldots$, podemos construir os vetores $q_1, q_2, \ldots$ por sucessivas ortogonalizações. Essa é a ortogonalização por Gram-Schmidt Clássico.

Desse modo, no $j$-ésimo passo, queremos encontrar um vetor unitário $q_j \in (a_1, \ldots, a_j)$ que seja ortogonal a $q_1, \ldots, q_{j-1}$. Então vemos que

$$v_j = a_j - (q_1^\star a_j)q_1 - (q_2^\star a_j)q_2 - \cdots - (q_j^\star a_j)q_{j-1}$$

é o vetor que queríamos, exceto que ainda não foi normalizado. Se dividirmos pela sua norma $\|v_j\|$, o resultado é o vetor $q_j$ necessário.

Então podemos reescrever a expressão anterior na forma

$$q_n = \frac{a_n - \sum_{i=1}^{n-1} r_{in} q_i}{r_{nn}}$$

E disso é evidente que

$$\begin{aligned} r_{ij} &= q_i^\star a_j, \quad (i \neq j) \\ |r_{jj}| &= \|a_j - \sum_{i=1}^{j-1} r_{ij} q_i\|, \quad r_{jj} > 0 \end{aligned}$$

Por fim, chegamos ao algoritmo implementado (em Python).

## 2.2 Gram-Schmidt Modificado

Para cada valor de $j$, o algoritmo de Gram-Schmidt Clássico computa uma única projeção ortogonal de posto $m - (j-1)$,

$$v_j = P_j a_j$$

Já o algoritmo de Gram-Schmidt modificado computa o mesmo resultado por uma sequência de $j-1$ projeções de posto $m-1$. Seja $P_{\perp q}$ a projeção ortogonal, de posto $m-1$ no espaço ortogonal ao vetor $q \in \mathbb{C}$, $q \neq 0$. Por definição de $P_j$, temos

$$P_j = P_{\perp q_{j-1}} \ldots P_{\perp q_2} P_{\perp q_1},$$

em que $P_1 = I$. Então

$$v_j = P_{\perp q_{j-1}} \ldots P_{\perp q_2} P_{\perp q_1} a_j$$

e o algoritmo é baseado no uso de $v_j$ como definido nessa última passagem.

O algoritmo modificado calcula $v_j$ resolvendo as seguintes fórmulas em ordem:

$$\begin{aligned} v_j^{(1)} &= a_j \\ v_j^{(2)} &= P_{\perp q_1} v_j^{(1)} = v_j^{(1)} - q_1 q_1^\star v_j^{(1)} \\ v_j^{(3)} &= P_{\perp q_2} v_j^{(2)} = v_j^{(2)} - q_2 q_2^\star v_j^{(2)} \\ &\vdots \\ v_j = v_j^{(j)} &= P_{\perp q_{j-1}} v_j^{(j-1)} = v_j^{(j-1)} - q_{j-1} q_{j-1}^\star v_j^{(j-1)} \end{aligned}$$

### 2.3 Fatoração de Householder

Na transformação ou reflexão de Householder, pega-se um vetor e reflete-o em relação a algum plano ou hiperplano. Essa operação pode ser utilizada para calcular a fatoração $QR$ de uma matriz $A$ de ordem $m \times n$ com $m \geq n$.

$Q$ pode ser utilizada para refletir um vetor de tal forma que todas as coordenadas, exceto uma, desapareçam.

Seja $x$ um vetor coluna real arbitrário de $A$ de dimensão $m$, tal que $\|x\| = \alpha$ para algum escalar $\alpha$. Com o algoritmo implementado usando a aritmética de ponto flutuante, $\alpha$ recebe o sinal contrário ao da $k$-ésima coordenada de $x$, onde $x_k$ deve ser a coordenada pivô a partir da qual todas as entradas são 0 na forma triangular superior final de $A$, para evitar a perda de significância.

## 3 Testes e Resultados

Durante o desenvolvimento, geramos matrizes aleatórias de dimensão baixa, para conferir a ortogonalidade de $Q$, e se o produto matricial $QR$ realmente coincidia com $X$. Com os algoritmos prontos, executamos o programa para algumas matrizes quadradas de ordem $n$ do tipo quadrado mágico, e para matrizes de Hilbert de ordem $n$. Programamos uma função geradora das matrizes de Hilbert na mão, mas usamos um pacote nativo para os magic squares, que precisa do comando 'pip install magic_square' antes de executar o código de fato. Partimos da ordem de 50 até 1000, com passo 50, para observar a evolução dos erros e do tempo gasto. Os resultados foram coletados em uma tabela e fizemos alguns gráficos, que serão mostrados a seguir:

### 3.1 Figuras e Tabelas: Hilbert

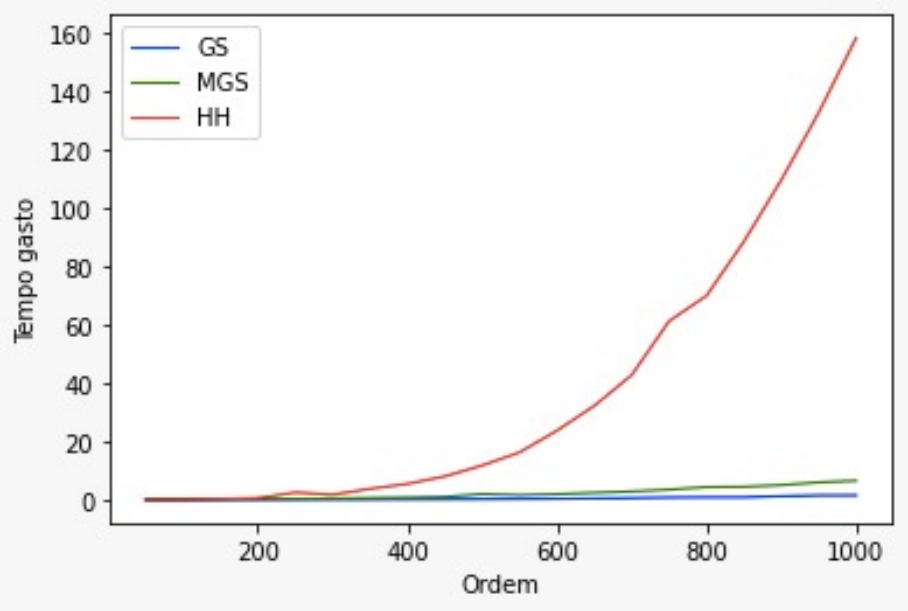


Figura 1: Tempo gasto para cada algoritmo

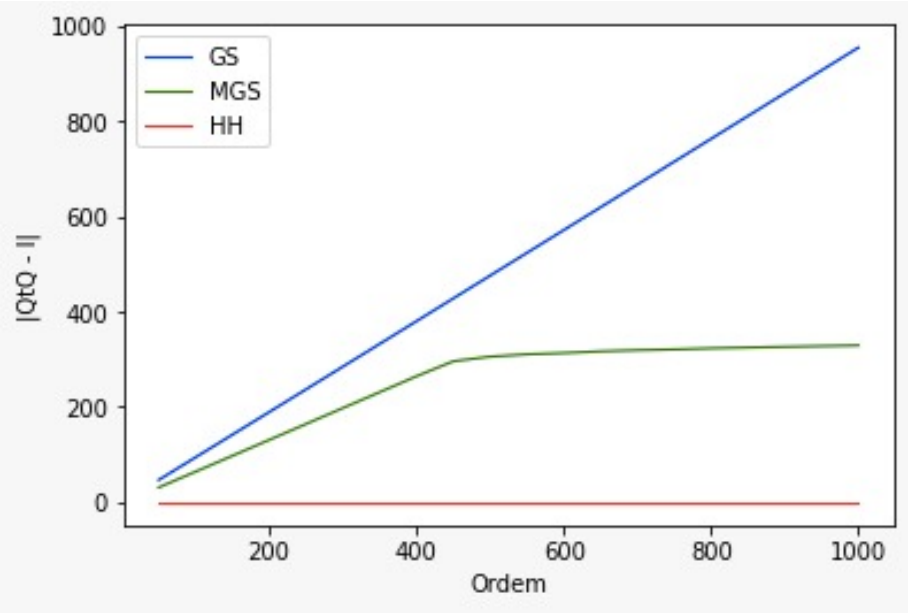


Figura 2: Gráfico de $|Q^TQ - I|$

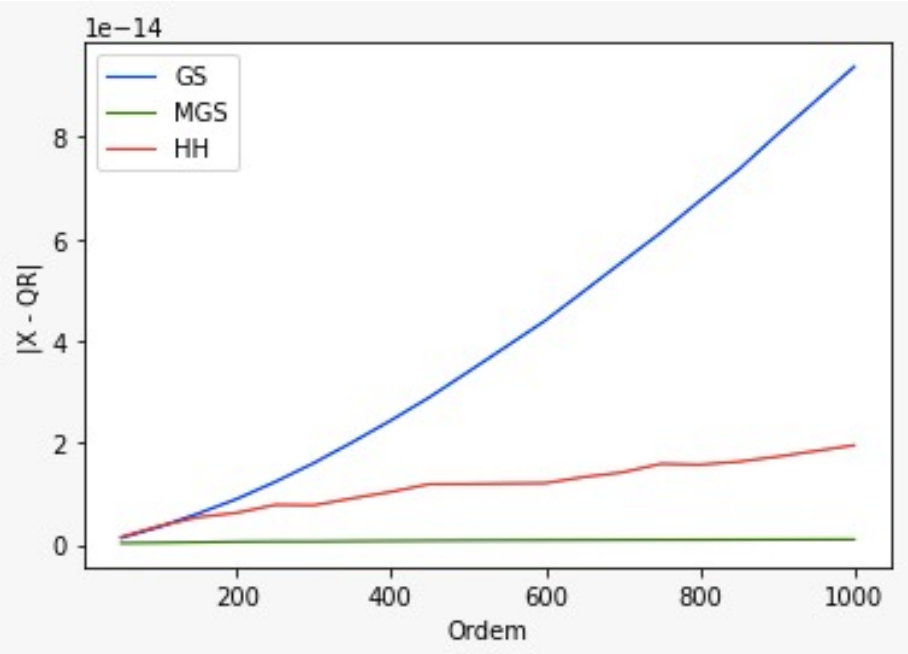


Figura 3: Gráfico de $|X - QR|$

| | ordem | condição | GS:\|X - QR\| | MGS:\|X - QR\| | HH:\|X - QR\| | GS:\|QtQ - I\| | MGS:\|QtQ - I\| | HH:\|QtQ - I\| | GS:tempo | MGS:tempo | HH:tempo |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 50 | 2.312546e+19 | 1.320388e-15 | 3.018120e-16 | 1.482755e-15 | 46.930634 | 31.276292 | 1.169048e-14 | 0.001604 | 0.013259 | 0.006589 |
| **1** | 100 | 2.525413e+19 | 3.481999e-15 | 3.749391e-16 | 3.669210e-15 | 94.411015 | 64.309751 | 2.915131e-14 | 0.003765 | 0.052642 | 0.037008 |
| **2** | 150 | 2.833016e+19 | 6.110348e-15 | 4.667450e-16 | 5.383528e-15 | 141.845317 | 97.595627 | 5.518396e-14 | 0.006400 | 0.120745 | 0.167271 |
| **3** | 200 | 9.284973e+19 | 8.968480e-15 | 5.621546e-16 | 6.248613e-15 | 189.351894 | 130.941286 | 7.934280e-14 | 0.018371 | 0.198984 | 0.400395 |
| **4** | 250 | 4.091520e+20 | 1.234006e-14 | 6.178041e-16 | 7.810947e-15 | 236.926678 | 164.311485 | 1.170455e-13 | 0.035234 | 0.296769 | 0.989998 |
| **5** | 300 | 1.472655e+20 | 1.602959e-14 | 6.281997e-16 | 7.739863e-15 | 284.556031 | 197.687588 | 1.231177e-13 | 0.039229 | 0.391963 | 2.237379 |
| **6** | 350 | 9.324977e+20 | 2.012270e-14 | 6.901626e-16 | 9.093215e-15 | 332.228317 | 231.066336 | 1.461151e-13 | 0.067242 | 0.549871 | 3.165423 |
| **7** | 400 | 3.254593e+20 | 2.441332e-14 | 7.332793e-16 | 1.037082e-14 | 379.934708 | 264.423272 | 1.849368e-13 | 0.097090 | 0.741130 | 4.398873 |
| **8** | 450 | 1.140206e+20 | 2.896402e-14 | 7.795250e-16 | 1.188385e-14 | 427.668602 | 295.923501 | 2.132685e-13 | 0.125746 | 0.959932 | 7.933888 |
| **9** | 500 | 1.255394e+21 | 3.393193e-14 | 8.208539e-16 | 1.190502e-14 | 475.425016 | 305.553392 | 2.544605e-13 | 0.170034 | 1.186949 | 10.767235 |
| **10** | 550 | 5.536030e+20 | 3.896048e-14 | 8.490260e-16 | 1.199886e-14 | 523.200134 | 310.451281 | 2.314003e-13 | 0.218131 | 1.525900 | 14.613735 |
| **11** | 600 | 2.577351e+20 | 4.400922e-14 | 8.714751e-16 | 1.207343e-14 | 570.990984 | 313.135878 | 2.662693e-13 | 0.268544 | 1.880719 | 20.661120 |
| **12** | 650 | 9.764051e+20 | 4.976937e-14 | 8.957529e-16 | 1.326411e-14 | 618.795218 | 316.725751 | 3.027700e-13 | 0.350751 | 2.307384 | 30.581484 |
| **13** | 700 | 1.773202e+20 | 5.555395e-14 | 9.212176e-16 | 1.418111e-14 | 666.610952 | 319.035009 | 3.476548e-13 | 0.413266 | 2.744730 | 40.623007 |
| **14** | 750 | 1.437526e+23 | 6.129182e-14 | 9.378353e-16 | 1.586743e-14 | 714.436656 | 320.695455 | 3.776978e-13 | 0.531662 | 3.410560 | 50.022053 |
| **15** | 800 | 3.847589e+20 | 6.749021e-14 | 9.520101e-16 | 1.569404e-14 | 762.271072 | 323.072588 | 3.652688e-13 | 0.661398 | 4.188265 | 58.409312 |
| **16** | 850 | 1.872126e+20 | 7.352921e-14 | 9.681724e-16 | 1.627406e-14 | 810.113157 | 324.669268 | 3.988376e-13 | 0.805155 | 5.122262 | 79.805849 |
| **17** | 900 | 7.039115e+20 | 8.045237e-14 | 9.860940e-16 | 1.725260e-14 | 857.962035 | 326.206794 | 4.348859e-13 | 0.940761 | 6.914297 | 93.258584 |
| **18** | 950 | 1.194675e+22 | 8.706654e-14 | 1.002695e-15 | 1.837259e-14 | 905.816964 | 327.718681 | 4.773327e-13 | 1.274294 | 9.048352 | 113.301649 |
| **19** | 1000 | 8.387308e+20 | 9.380937e-14 | 1.018159e-15 | 1.950347e-14 | 953.677315 | 329.097733 | 5.260761e-13 | 1.977710 | 15.687650 | 131.232430 |

Figura 4: Tabela de resultados: Matrizes de Hilbert

## 3.2 Figuras e Tabelas: Quadrados mágicos

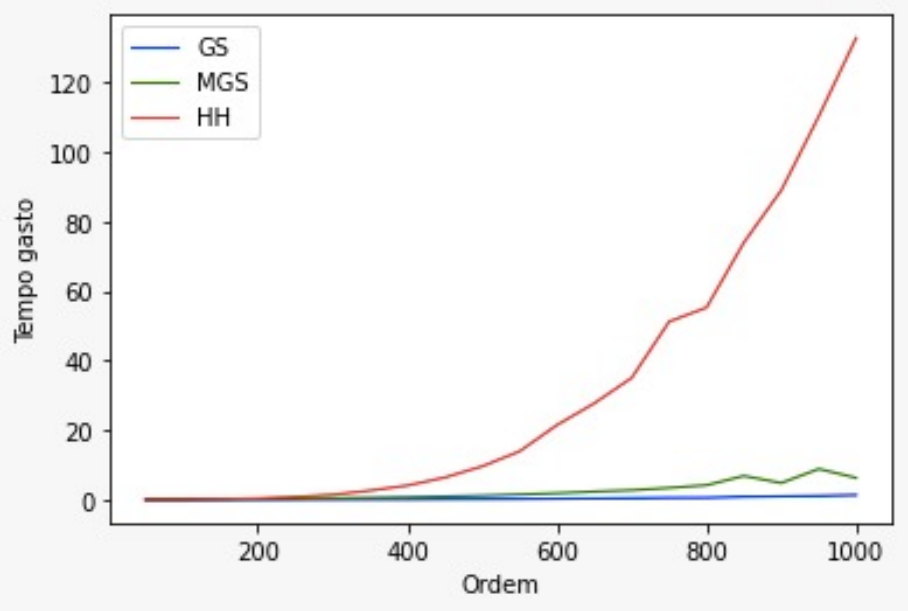


Figura 5: Tempo gasto para cada algoritmo

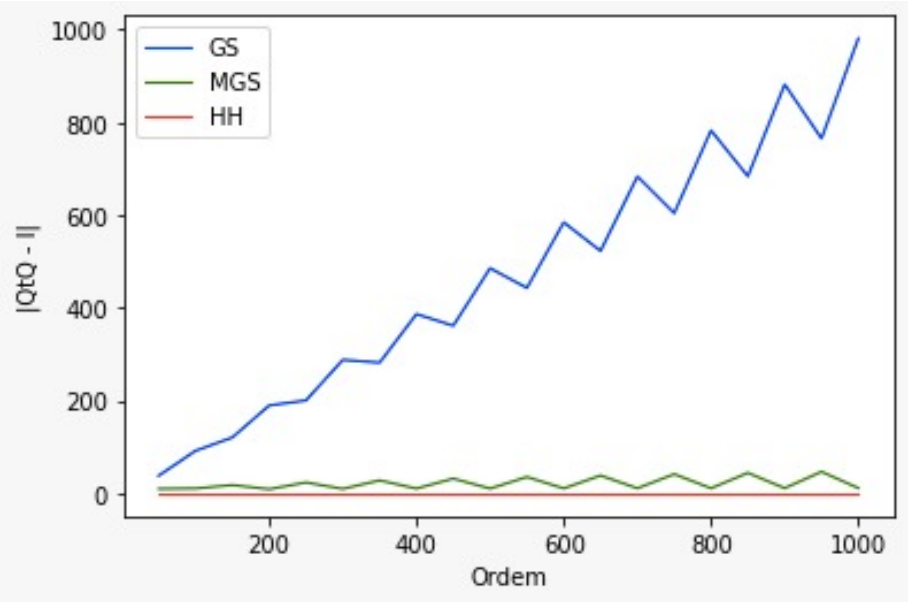


Figura 6: Gráfico de $|Q^TQ - I|$

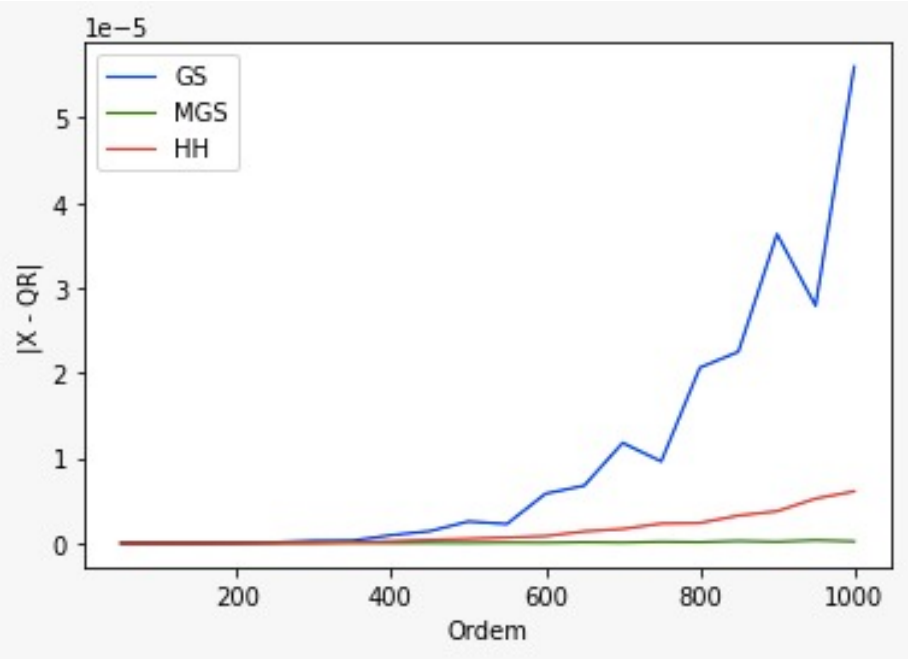


Figura 7: Gráfico de $|X - QR|$

| | ordem | condição | GS:\|X - QR\| | MGS:\|X - QR\| | HH:\|X - QR\| | GS:\|QtQ - I\| | MGS:\|QtQ - I\| | HH:\|QtQ - I\| | GS:tempo | MGS:tempo | HH:tempo |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 50 | 2.265667e+20 | 7.609407e-11 | 1.607414e-11 | 8.636254e-11 | 37.993951 | 9.690169 | 1.225312e-14 | 0.001263 | 0.007487 | 0.004388 |
| **1** | 100 | 5.488506e+20 | 2.050568e-09 | 1.308230e-10 | 9.503759e-10 | 91.682724 | 10.459634 | 2.792111e-14 | 0.004021 | 0.052962 | 0.042315 |
| **2** | 150 | 7.837984e+18 | 7.039510e-09 | 7.598183e-10 | 5.947752e-09 | 120.420309 | 17.788103 | 5.714364e-14 | 0.007299 | 0.123486 | 0.151307 |
| **3** | 200 | 8.012859e+18 | 4.331599e-08 | 1.159403e-09 | 1.497064e-08 | 189.400670 | 8.998895 | 8.073853e-14 | 0.013695 | 0.183382 | 0.353436 |
| **4** | 250 | 1.193212e+19 | 9.618118e-08 | 4.478799e-09 | 4.068602e-08 | 200.255504 | 23.372091 | 1.099915e-13 | 0.029612 | 0.266633 | 0.790732 |
| **5** | 300 | 1.790937e+20 | 2.683753e-07 | 4.377393e-09 | 6.729236e-08 | 287.756304 | 9.592357 | 1.177435e-13 | 0.045738 | 0.415374 | 1.413221 |
| **6** | 350 | 1.476302e+21 | 2.989489e-07 | 1.270585e-08 | 1.376409e-07 | 281.769090 | 27.891420 | 1.592395e-13 | 0.077748 | 0.545114 | 2.617676 |
| **7** | 400 | 3.063080e+19 | 9.319703e-07 | 1.179706e-08 | 2.047741e-07 | 386.413779 | 10.280284 | 1.835241e-13 | 0.085633 | 0.720580 | 4.068232 |
| **8** | 450 | 1.013007e+19 | 1.403214e-06 | 3.194456e-08 | 3.669642e-07 | 361.413509 | 31.787196 | 2.191621e-13 | 0.138385 | 0.929524 | 6.376227 |
| **9** | 500 | 9.369848e+19 | 2.542514e-06 | 2.209400e-08 | 5.013380e-07 | 485.255082 | 10.233586 | 2.529866e-13 | 0.161097 | 1.156132 | 9.584880 |
| **10** | 550 | 3.431870e+19 | 2.246442e-06 | 6.132852e-08 | 6.545146e-07 | 442.875272 | 35.261592 | 2.402373e-13 | 0.197574 | 1.500389 | 13.905363 |
| **11** | 600 | 1.881178e+21 | 5.811585e-06 | 4.308061e-08 | 8.215712e-07 | 584.222862 | 10.614764 | 2.657671e-13 | 0.265879 | 1.848476 | 21.455012 |
| **12** | 650 | 2.057460e+20 | 6.746223e-06 | 1.060924e-07 | 1.366062e-06 | 522.494772 | 38.426917 | 3.251045e-13 | 0.326135 | 2.335259 | 27.717258 |
| **13** | 700 | 2.724245e+20 | 1.178136e-05 | 6.521309e-08 | 1.664553e-06 | 683.284266 | 10.838936 | 3.436873e-13 | 0.403696 | 2.714108 | 35.046365 |
| **14** | 750 | 1.057384e+20 | 9.599766e-06 | 1.592446e-07 | 2.294613e-06 | 603.940221 | 41.353116 | 4.096733e-13 | 0.521741 | 3.385915 | 51.185952 |
| **15** | 800 | 1.525858e+20 | 2.061959e-05 | 1.128739e-07 | 2.371021e-06 | 782.418461 | 10.937289 | 3.725532e-13 | 0.545649 | 4.166515 | 55.237258 |
| **16** | 850 | 2.366275e+20 | 2.249382e-05 | 2.538622e-07 | 3.234148e-06 | 683.550854 | 44.087214 | 4.222101e-13 | 0.850554 | 6.827524 | 73.960533 |
| **17** | 900 | 4.902960e+20 | 3.632555e-05 | 1.605830e-07 | 3.742485e-06 | 881.611279 | 11.105435 | 4.425962e-13 | 0.967771 | 4.800655 | 88.961068 |
| **18** | 950 | 1.586890e+20 | 2.789797e-05 | 3.357468e-07 | 5.232433e-06 | 764.988797 | 46.662616 | 4.943066e-13 | 1.083744 | 8.835342 | 110.174394 |
| **19** | 1000 | 3.759670e+20 | 5.600374e-05 | 2.057007e-07 | 6.081060e-06 | 980.852579 | 11.237182 | 5.359940e-13 | 1.335488 | 6.276842 | 132.732360 |

Figura 8: Tabela de resultados: Quadrados mágicos

## 3.3 Comentários sobre os resultados

É nítido que a fatoração usando as matrizes de Householder tem ordem computacional maior, o que faz sentido, partindo do pressuposto que geramos as matrizes $Hv$ a cada rodada, e temos que pegar as matrizes reduzidas, para depois "aumentarmos" elas novamente. Isso gera iterações encadeadas, o que prejudica o tempo do algoritmo. O tempo da $MGS$ é maior que o da $GS$ clássica, mas ordem é a mesma.

Quando calculamos o erro de ortogonalização, é nítido o péssimo desempenho do algoritmo de Gram-Schmidt clássico. O modificado já um pouco melhor, apesar do erro ainda pairar entre 10 e 40 nos nossos testes. Isso ocorre devido ao cálculo usando ponto flutuante, que pode ser diminuído, mas sempre existe. O algoritmo de Gram-Schmidt modificado é pensado justamente para isso - e há de fato uma nítida melhora. Já o algorítmo de Householder é feito de forma a gerar colunas ortonormais, então esse erro sempre será pequeno, e de fato apresenta erros da ordem de $10^{-13}$, enquanto o erro da Gram-Schmidt pode saltar até para a ordem de quase $10^3$.

Tivemos baixos erros de $|X - QR|$, o que mostra que nossos algoritmos estão funcionando como esperado. O Gram-Schmidt modificado ficou com o melhor desempenho nesse quesito, seguido por pouco pelo Householder.

# Referências

[1] C. MOLER, *Compare Gram-Schmidt and Householder Orthogonalization Algorithms*. Disponível em https://blogs.mathworks.com/cleve/2016/07/25/compare-gram-schmidt-and-householder-orthogonalization-algorithms/

[2] *Decomposição QR*. Disponível em: https://pt.wikipedia.org/wiki/Decomposicao_QR

[3] TREFETHEN, BAU, *Numerical Linear Algebra*.