**Raízes de Equação Polinomial com IA’s**

**Mário Leite**

**...**

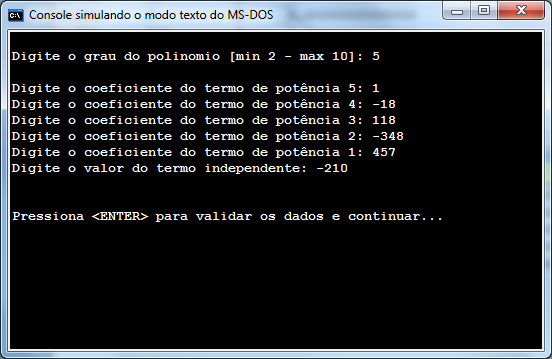
Como eu já havia apresentado uma postagem com um código para calcular as raízes de uma equação polinomial, não vou entrar em detalhes sobre esse tipo de equação aqui. Nesta postagem quis tirar as minhas dúvidas quanto às capacidades das chamadas Inteligências Artificiais (IA’S) resolverem problemas de computação numérica. Para isto solicitei à duas dessas IA‘s (***ChatGPT*** e ***Google Bard***) para criarem um código em Python que calculasse as raízes da seguinte equação polinomial: **x5 -18x4 + 118x3 - 348x2 + 457x – 210 = 0**.

Para criar a minha solução, primeiramente estudei o “Método de Birge-Vieta” e desenvolvi um código em **Visualg** (tirado do meu livro “*Programação Básica com Visualg*” - Amazon - em parceria com o ***prof. Antonio Nicolodi*** : gestor e um dos criadores dessa linguagem).

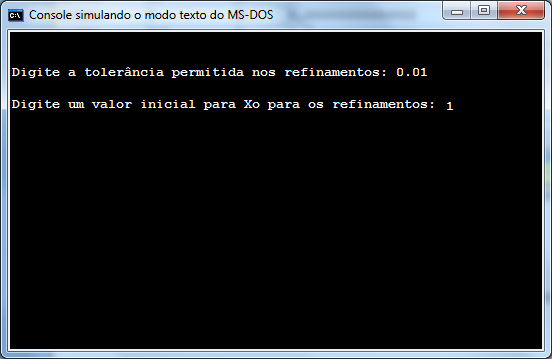
Depois, para conferir o resultado do meu código usei o **SciLab** (tirado no meu livro: “*SciLab: Uma Abordagem Prática e Didática*” - Amazon) para checar a correctude dele: bateu certinho (vide **figuras 1.c** e **2)**. Em seguida testei a solução com as duas IA’s citadas; os resultados estão apresentados nas figuras **3** e **4**, mostrando que tanto o ChatGPT quanto o Google Bard erraram a resposta com os seus códigos sugeridos! E para a minha decepção (estava empolgado com ela) Bard afirmou que a equação não tem raízes!Assim, se for preciso corrigir erros de programas criados por uma IA, então, para que serve uma IA no desenvolvimento de uma aplicação crítica? Talvez a explicação de muitos programadores cegamente confiantes nelas é a de que ***“elas ainda estão aprendendo***”; e como vou saber se elas já aprenderam!?

Conclusão: Os iniciantes em Programação não devem ficar preocupados em perder seus empregos para essas tais IA’s, como muitos “analistas” estão alardeando nas Redes Sociais. Eu, partucularmente, não acredito que as IA’s vão nos superar; talvez quardar detalhes de uma receita de bolo que elas podem acessar de seus bancos de dados, pode ser; mas, em termos de criar programas computacionais... desconfio muito!. Aliás, o que os iniciantes em programação têm que entender é que elas dependem, justamente, dos programadores...

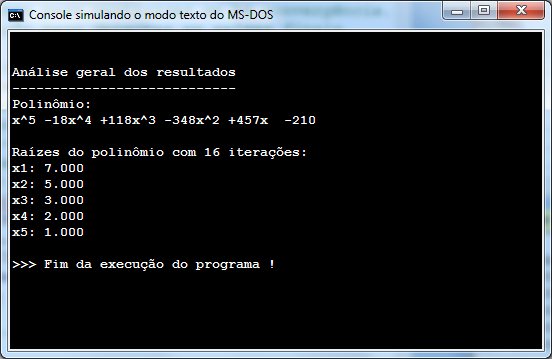
-------------------------------------------------------------------------------------------------------------------------



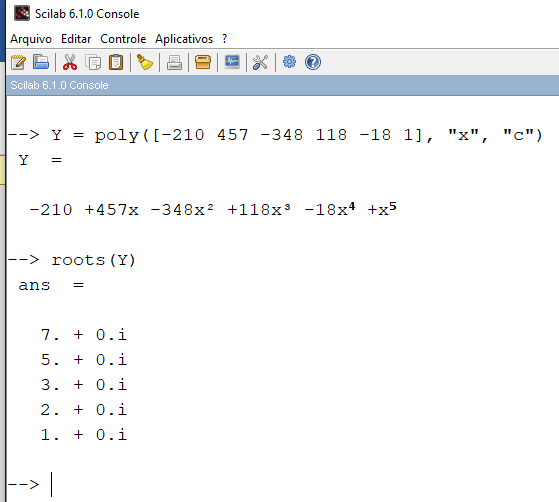
**Figura 1a - Entradas dos coeficientes da equação (código do programador)**



**Figura 1b - Entradas da tolerancia to e Xinicial (código do programador)**



**Figura 1c - Obtenção das raízes (código do programador)**

****

**Figura 2 - Cálculo das raízes da equação no ambiente do *Scilab***

**Algoritmo** **"RaizesDePolinomio"**

**{**

---------------------------------------------------------------------------

{Programa modular para calcular as raízes reais de um polinômio completo de grau **n** pelo *"Método de Birge-Vieta"*

**Autor: Mário Leite**

**Data: 30/07/2023**

---------------------------------------------------------------------------

**Xo**: valor inicial de raiz dada pelo usuário.

**Xi**: um valor de raiz dentro do loop de refinamento.

**Erro**: erro cometido no refinamento das raízes.

**tol**: tolerância permitida no erro cometido ao calcular a raiz.

**Ite**: número de iteraçõs para verificar se há convergência.

**Iter**: número total de iteraçõs para detectar as raízes finais.

**n**: grau do polinômio.

**NR**: número de raízes do polinômio

**Coef0[i]**, i=0,n: coeficientes originais do polinômio (lidos pelo teclado).

**Coef1[i]**, i=1,3: coeficiente derivativos auxiliares para os cálculos.

**VetRaizes**: vetor que armazena as raízes do polinômio.

---------------------------------------------------------------------------

**}**

*//Elementos globais*

**Const** MaxIte=1000 *//define o máximo de iterações para detectar possíveis raízes*

**Var** j, n, Iter: **inteiro**

Coef0, Coef1, Coef2, Coef3: **vetor**[0..10] **de** **real**

VetRaizes: **vetor**[1..10] **de** **real** //limita o grau do polinômio

a, b, c, Xo, Xi, Y, tol, Erro: **real**

x1R, x1C, x2R, x2C, Delta, Delta2: **real**

nada, x1S, x2S: **caractere**

Convergiu: **logico**

//---------------------------------------------------------------------------------

**Procedimento** **LePolinomio**(n:**inteiro**)

***//Lê os dados básicos do polinômio.***

**var** k:**inteiro**

**Inicio**

**Escreval**("")

*{Entradas dos coeficientes do polinômio}*

**Para** k **De** n **Ate** 0 **Passo** -1 **Faca**

**Se**(k>0) **Entao**

**Escreva**("Digite o coeficiente do termo de potência", k, ": ")

**Senao**

**Escreva**("Digite o valor do termo independente: ")

**FimSe**

**Leia**(Coef1[k])

Coef0[k] <- Coef1[k]

**FimPara**

**FimProcedimento** *//fim do procedimento "****LePolinomio****"*

//---------------------------------------------------------------------------------

**Procedimento** **MontaPolinomio**(n:**inteiro**)

***//Monta e exibe o polinômio.***

**var** k: **inteiro**

**Inicio**

**Para** k **De** n **Ate** 0 **Passo** -1 **Faca**

**Se**(k<>0) **Entao**

**Se**((Coef0[k]>0) **e** (k=n)) **Entao**

**Se**(Coef0[k]=1) **Entao**

**Escreva**("x^", NumpCarac(k), " ")

**Senao**

**Escreva**(NumpCarac(Coef0[k]), "x^", NumpCarac(k), " ")

**FimSe**

**FimSe**

**Se**((Coef0[k]<0) **e** (k=n)) **Entao**

**Se**(Coef0[k]=-1) **Entao**

**Escreva**("-x^", NumpCarac(k), " ")

**Senao**

**Escreva**(NumpCarac(Coef0[k]), "x^", NumpCarac(k), " ")

**FimSe**

**FimSe**

**Se**((Coef0[k]>0) **e** (k<>n) **e** (k>1)) **Entao**

**Escreva**("+", NumpCarac(Coef0[k]), "x^", NumpCarac(k), " ")

**FimSe**

**Se**((Coef0[k]>0) **e** (k<>n) **e** (k=1)) **Entao**

**Escreva**("+", NumpCarac(Coef0[k]), "x ")

**FimSe**

**Se**((Coef0[k]<0) **e** (k<>n) **e** (k=1)) **Entao**

**Escreva**(NumpCarac(Coef0[k]), "x ")

**FimSe**

**Se**((Coef0[k]<0) **e** (k<>n) **e** (k<>1)) **Entao**

**Escreva**("", NumpCarac(Coef0[k]), "x^", NumpCarac(k), " ")

**FimSe**

**Senao**

**Se**(Coef0[0]>0) **Entao**

**Escreva**("+",Coef0[0])

**Fimse**

**Se**(Coef0[0]<0) **Entao**

**Escreva**(Coef0[0])

**FimSe**

**FimSe**

**FimPara**

**FimProcedimento** *//fim do procedimento "****MontaPolinomio****"*

//------------------------------------------------------------------------------

**Procedimento** **ProRaiz2G**(a,b,c:**real**)

***//Calcula as raízes do polinômio de grau 2.***

**Inicio**

Delta <- b^2 - (4\*a\*c)

**Se**(Delta>=0) **Entao**

VetRaizes[1] <- (-b +RaizQ(Delta))/(2\*a)

VetRaizes[2] <- (-b -RaizQ(Delta))/(2\*a)

**Senao** *//raízes complexas*

Delta2 <- Abs(Delta)

x1R <- (-b)/(2\*a)

*x1R <-* Int*(x1R\*10^5+0.50)/10^5 //com cinco decimais*

x1C <- RaizQ(Delta2)/(2\*a)

x1C <- Int(x1C\*10^5+0.5)/10^5

x2R <- (-b)/(2\*a)

x2R <- Int(x2R\*10^5+0.50)/10^5

x2C <- RaizQ(Delta2)/(2\*a)

x2C <- Int(x2C\*10^5+0.5)/10^5

x1S <- NumpCarac(x1R) + " + " + NumpCarac(x1C) + "i"

x2S <- NumpCarac(x2R) + " - " + NumpCarac(x2C) + "i"

**FimSe**

**FimProcedimento** *//fim do procedimento "****ProRaiz2G****"*

//---------------------------------------------------------------------------------

**Procedimento** **RaizNG**(n:inteiro;Xo,tol:**real**)

***//Calcula as raízes do polinômio de grau n.***

**var** i, k, NR: **inteiro**

**Inicio**

**LimpaTela**

**Escreval**(**""**)

**Escreval**("Resultados parciais nas iterações")

NR <- n *//salva o número de raízes em NR*

*{Cálculos dos valores refinados de* ***Xi****}*

Iter <- 0

**Repita** *//loop para calcular uma nova raiz Xi*

Coef2[n] <- Coef1[n]

Coef3[n] <- Coef1[n]

**Repita**

Iter <- Iter + 1

**Para** i **De** (n-1) **Ate** 0 **Passo** -1 **Faca**

Coef2[i] <- Coef1[i] + Coef2[i+1]\*XO

Coef3[i] <- Coef2[i] + Coef3[i+1]\*XO

**FimPara**

Xi <- Xo - Coef2[0]/Coef3[1]

Erro <- Abs(Xi - Xo)

**Escreval**("Iteração:", Iter)

**Escreval**("Raiz=",Xi:8:5)

Xo <- Xi

**Ate**((Erro<tol) **ou** (Iter>MaxIte)) *//valida erro cometido com iterações*

**Se**(Iter>MaxIte) **Entao**

Convergiu <- **Falso**

**Senao**

**Escreval**("Raiz detectada = ",Xi:8:6)

**Escreval**("")

VetRaizes[n] <- Xi

**FimSe**

n <- n - 1 *//decrementa o grau do polinômio*

**Para** i **De** n **Ate** 0 **Passo** -1 **Faca**

Coef1[i] <- Coef2[i+1] *//ajeita os coeficientes auxiliares*

**FimPara**

**Ate**(n=0) *//fim do loop para calcular uma nova raiz*

*{Início dos cálculos de refinamento das raízes}*

**Se**(Convergiu) **Entao**

**Se**(Iter<=MaxIte) **Entao** *//verifica o número de refinamentos com o limite*

**Para** i **De** 1 **Ate** NR **Faca**

Xo <- VetRaizes[i]

Coef2[NR] <- Coef0[NR]

Coef3[NR] <- Coef0[NR]

**Repita**

**Para** k **De** (NR-1) **Ate** 0 **Passo** -1 **Faca**

Coef2[k] <- Coef0[k] + Coef2[k+1]\*Xo

Coef3[k] <- Coef2[k] + Coef3[k+1]\*Xo

**FimPara** *//fimPara-k*

Xi <- Xo - (Coef2[0]/Coef3[1])

Erro <- **Abs**(Xi-Xo)

Xo <- Xi

**Ate**(Erro<tol)

VetRaizes[i] <- Xi

**FimPara** *//fimPara-NR*

*{Fim dos cálculos de refinamento das raizes}*

**FimSe**

**FimSe** *//FimSe-convergiu*

**FimProcedimento** *//fim do procedimento "****RaizNG****"*

//=================================================================================

***//Programa principal***

**Inicio**

**LimpaTela**

**Escreval**("")

Convergiu <- **Verdadeiro**

n <- 1

**Enquanto** ((n<2) **ou** (n>10)) **Faca** //valida o grau do polinômio

**Escreva**("Digite o grau do polinômio [min 2 - max 10]: ")

**Leia**(n)

**FimEnquanto** *//fim da validação o grau do polinômio*

**LePolinomio**(n) *//chama rotina para ler os coeficientes do polinômio*

*{Analisa o grau do polinômio para calcular as raízes}*

**Escolha** n

**Caso** 2

a <- Coef1[2]

b <- Coef1[1]

c <- Coef1[0]

**ProRaiz2G**(a,b,c) *//chama rotina para o polinômio de grau 2*

**OutroCaso**

**Repita**

**Escreval**("")

**Escreva**("Digite a tolerância permitida nos refinamentos: ")

**Leia**(tol)

*//****tol=0.005*** *é um bom valor para maior rigor nos cálculos*

**Ate**(tol>0)

**Escreval**("")

**Escreva**("Digite um valor inicial para Xo para os refinamentos: ")

**Leia**(Xo)

*//****xo=0.5*** *é um bom valor para maior rigor nos cálculos*

**Escreval**("")

**Escreval**("")

**Escreval**("")

**Escreval**("")

**Escreva**("Pressiona <ENTER> para validar os dados e continuar... ")

**Leia**(nada) //apenas para provocar uma parada temporária

**RaizNG**(n,Xo,tol) *//chama rotina para o polinômio de grau superior a* ***2***

**FimEscolha**

**Escreval**("")

**LimpaTela**

**Se**(Convergiu) **Entao**

**Escreval**("")

**Escreval**("Análise geral dos resultados")

**Escreval**("----------------------------")

**Escreva**("Polinômio: ")

**Escreval**("")

**MontaPolinomio**(n) *//chama rotina para exibir textualmente o polinômio*

**Escreval**("")

**Escreval**("")

**Se**(Delta<0) **Entao**

**Escreval**("Polinômio de grau 2 com raízes complexas e conjugadas:")

**Escreval**("x1 = ",x1S)

**Escreval**("x2 = ",x2S)

**Senao**

**Se**(Iter>0) **Entao**

**Escreval**("Raízes do polinômio com", Iter," iterações:")

**Senao**

**Escreval**("Raízes do polinômio:")

**FimSe**

**Para** j **De** 1 **Ate** n **Faca**

**Escreval**(“x”,NumpCarac(j), “: “,VetRaizes[j]:5:2)

**FimPara**

**FimSe**

**Senao**

**Escreval**(“”)

**Escreval**(“Não convergiu com”,MaxIte,” iterações. Pode existir raiz complexa.”)

**Escreval**(“”)

**Escreval**(“”)

**Escreva**(“”)

**Escreva**(“Pressiona <ENTER> para encerrar...”)

**Leia**(nada) *//apenas para provocar uma parada temporária*

**FimSe**

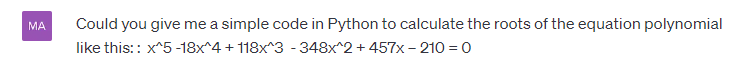
**Escreval**(“”)

**FimAlgoritmo** ***//fim do programa “RaizesDePolinomio”***

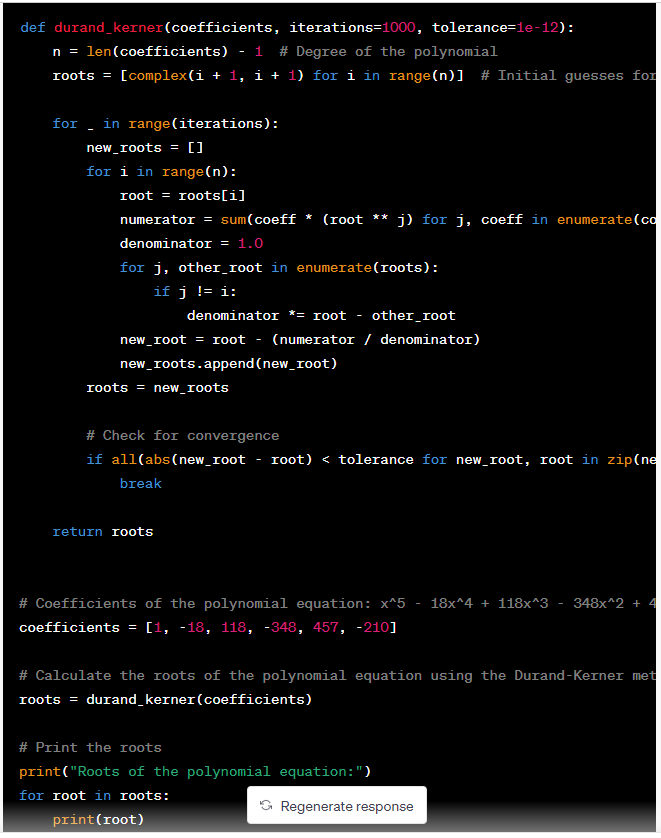
**------------------------------------------------------------------------------------------------------------------------------------**

**Conversas com as IA’s**

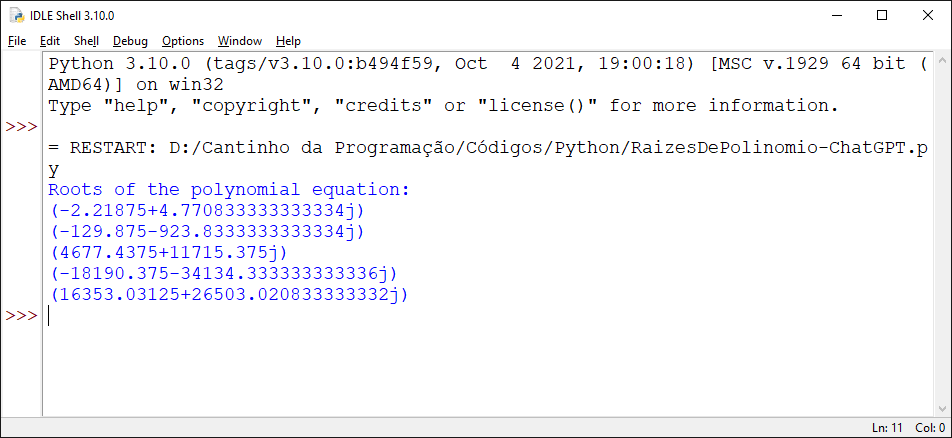
**1 - ChatGPT**

****

****

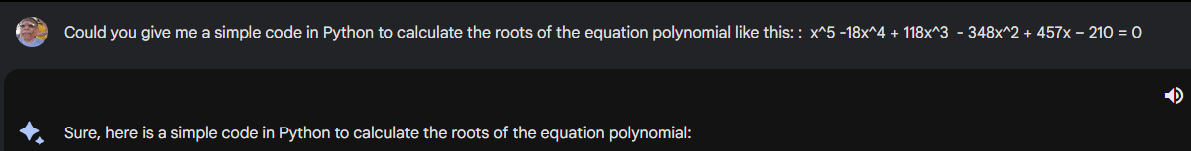
****

**Código em Pyton sugerido pelo *ChatGPT***



**Figura 3 - Saída do programa sugerido pelo *ChatGPT***

**2 - Google Bard**



**import** **math**

**def** **calculate\_roots(coefficients)**:

"""

Calculates the roots of the polynomial given by the coefficients.

Args:

coefficients: A list of the polynomial coefficients.

Returns:

A list of the polynomial roots.

"""

**# Find the discriminant of the polynomial.**

**discriminant = (coefficients[1] \*\* 2) - (4 \* coefficients[0] \* coefficients[2])**

**# If the discriminant is negative, the polynomial has no real roots.**

**if** **discriminant <** **0:**

return **[]**

**# If the discriminant is zero, the polynomial has one real root.**

**if** **discriminant ==** **0::**

**return [-coefficients[1] / (2 \* coefficients[0])]**

**# If the discriminant is positive, the polynomial has two or three real roots.**

**roots = [**

**(-coefficients[1] + math.sqrt(discriminant)) / (2 \* coefficients[0]),**

**(-coefficients[1] - math.sqrt(discriminant)) / (2 \* coefficients[0])**

**]**

**# If the polynomial has three real roots, the middle root is repeated.**

**if** **len(roots) == 3:**

**roots.remove(roots[1])**

**# Use the Newton-Raphson method to find the complex roots of the polynomial.**

**complex\_roots = []**

**for** **i in** range**(5):**

**start\_guess =** **1.0**

**for** **j** **in** range**(100)**:

**next\_guess = start\_guess - coefficients[i] / (coefficients[i + 1] + start\_guess \* coefficients[i + 2])**

**if** abs**(next\_guess - start\_guess)** **< 1e-10:**

**complex\_roots.append(next\_guess)**

**break**

**start\_guess = next\_guess**

**return** **roots + complex\_roots**

**if** **\_\_name\_\_ ==** "\_\_main\_\_":

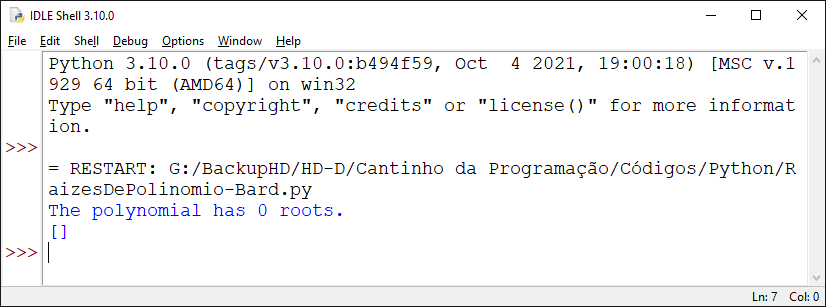
**coefficients =** **[1, -18, 118, -348, 457, -210]**

**roots = calculate\_roots(coefficients)**

**print("**The polynomial has", len(roots), "roots.")

print(roots)

**Código em Pyton sugerido pelo *Google Bard***



**Figura 4 - Saída do programa sugerido pelo *Google Bard***