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: $x^5 - 18x^4 + 118x^3 - 348x^2 + 457x - 210 = 0$.

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 **2** e **3**, 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 <u>a equação não tem raízes!</u> 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!?

<u>Conclusão</u>: 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, <u>partucularmente</u>, 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...

```
Digite o grau do polinomio [min 2 - max 10]: 5

Digite o coeficiente do termo de potência 5: 1

Digite o coeficiente do termo de potência 4: -18

Digite o coeficiente do termo de potência 3: 118

Digite o coeficiente do termo de potência 2: -348

Digite o coeficiente do termo de potência 1: 457

Digite o valor do termo independente: -210

Pressiona <ENTER> para validar os dados e continuar...
```

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

```
Console simulando o modo texto do MS-DOS

Digite a tolerância permitida nos refinamentos: 0.01

Digite um valor inicial para Xo para os refinamentos: 1
```

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

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

```
Scilab 6.1.0 Console
Arquivo Editar Controle Aplicativos ?

Scilab 6.1.0 Console

--> Y = poly([-210 457 -348 118 -18 1], "x", "c")
Y =

-210 +457x -348x² +118x³ -18x⁴ +x⁵

--> roots(Y)
ans =

7. + 0.i
5. + 0.i
3. + 0.i
2. + 0.i
1. + 0.i

--> |
```

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: <a href="mailto:vetor">vetor</a>[0..10] de <a href="mailto:real">real</a>
       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] \leftarrow Coef1[k]
   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] \leftarrow (-b + RaizQ(Delta)) / (2*a)
      VetRaizes[2] <- (-b -RaizQ(Delta))/(2*a)</pre>
   Senao //raízes complexas
      Delta2 <- Abs (Delta)
      x1R <- (-b)/(2*a)
      x1R \leftarrow Int(x1R*10^5+0.50)/10^5 //com cinco decimais
      x1C \leftarrow RaizQ(Delta2)/(2*a)
      x1C \leftarrow Int(x1C*10^5+0.5)/10^5
      x2R <- (-b)/(2*a)
      x2R \leftarrow Int(x2R*10^5+0.50)/10^5
      x2C \leftarrow RaizQ(Delta2)/(2*a)
      x2C \leftarrow Int(x2C*10^5+0.5)/10^5
      x1S <- NumpCarac(x1R) + " + " + NumpCarac(x1C) + "i"
x2S <- NumpCarac(x2R) + " - " + NumpCarac(x2C) + "i"</pre>
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] \leftarrow Coef1[n]
         Coef3[n] <- Coef1[n]</pre>
      Repita
          Iter <- Iter + 1</pre>
          Para i De (n-1) Ate 0 Passo -1 Faca
             Coef2[i] \leftarrow Coef1[i] + Coef2[i+1]*XO
             Coef3[i] \leftarrow Coef2[i] + Coef3[i+1]*XO
          FimPara
          Xi <- Xo - Coef2[0]/Coef3[1]</pre>
         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</pre>
      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]</pre>
             Coef2[NR] <- Coef0[NR]
             Coef3[NR] <- Coef0[NR]</pre>
             Repita
                Para k De (NR-1) Ate 0 Passo -1 Faca
                    Coef2[k] \leftarrow Coef0[k] + Coef2[k+1]*Xo
                    Coef3[k] \leftarrow Coef2[k] + Coef3[k+1]*Xo
                FimPara //fimPara-k
                Xi \leftarrow Xo - (Coef2[0]/Coef3[1])
                Erro <- Abs(Xi-Xo)</pre>
                Xo <- Xi
             Ate (Erro<tol)
             VetRaizes[i] <- Xi</pre>
          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
  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: ")
           //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)
        //\mathbf{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:")
               Escreval ("Raízes do polinômio:")
            FimSe
```

.....

1 - ChatGPT

MA

Could you give me a simple code in Python to calculate the roots of the equation polynomial like this: $x^5 - 18x^4 + 118x^3 - 348x^2 + 457x - 210 = 0$

Here's the Python code to calculate the roots of the polynomial equation you provided using the Durand-Kerner method:

```
def durand_kerner(coefficients, iterations=1000, tolerance=1e-12):
   n = len(coefficients) - 1 # Degree of the polynomial
   roots = [complex(i + 1, i + 1) for i in range(n)] # Initial guesses for
   for _ in range(iterations):
       new_roots = []
       for i in range(n):
           root = roots[i]
           numerator = sum(coeff * (root ** j) for j, coeff in enumerate(co
           denominator = 1.0
           for j, other_root in enumerate(roots):
                if j != i:
                    denominator *= root - other_root
           new_root = root - (numerator / denominator)
           new_roots.append(new_root)
       roots = new_roots
       # Check for convergence
       if all(abs(new_root - root) < tolerance for new_root, root in zip(ne</pre>
            break
   return roots
# Coefficients of the polynomial equation: x^5 - 18x^4 + 118x^3 - 348x^2 + 4
coefficients = [1, -18, 118, -348, 457, -210]
# Calculate the roots of the polynomial equation using the Durand-Kerner met
roots = durand_kerner(coefficients)
# Print the roots
print("Roots of the polynomial equation:")
for root in roots:
                             Regenerate response
   print(root)
```

Código em Pyton sugerido pelo ChatGPT

Figura 3 - Saída do programa sugerido pelo *ChatGPT*

```
Could you give me a simple code in Python to calculate the roots of the equation polynomial like this: : x^5 -18x^4 + 118x^3 - 348x^2 + 457x - 210 = 0

Sure, here is a simple code in Python to calculate the roots of the equation polynomial:
```

```
import math
def calculate roots(coefficients)
  # 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:</pre>
   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.
      (-coefficients[1] + math.sqrt(discriminant)) / (2 * coefficients[0]),
      (-coefficients[1] - math.sqrt(discriminant)) / (2 * coefficients[0])
 1
 # 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:</pre>
        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