CP315 Portfolio

Jasmaan Panesar (140722880)

Winter 2018

Contents

1 Horner's Scheme

Write Maple and/or Matlab code to implement the fact that a polynomial of degree d can be evaluated in d multiplications and d additions, using Horner's scheme.

Illustrate your code with the polynomials:

- $x^8 - x^7 + x^4 - x^3 + x + 1$
- $x^{400} - x^{300} + x^{200} - x^{100} + 1$
- $x^5 + x^8 + x^{11} + x^{14}$

Question: Does your Horner scheme code correctly detect the special structure of the $second^2$ and $third^3$ polynomials above?

Answer:

From the Maple code on the next two pages you can observe that for the provided sample polynomials, once they are converted to Horner's Scheme the amount of multiplications required to solve them significantly reduces whereas the number of additions stays the same. The implemented Horner's scheme was able to correctly detect the special structure of the $second^2$ and $third^3$ polynomials above.

```
> horner := proc(p::polynom)
      local polynomial, term, coefficient, exp, hornerExpression::polynom;
      polynomial := sort(p, x, ascending);
      hornerExpression::polynom := a;
      while polynomial <> 0 do
          exp := ldegree(polynomial);
          coefficient := tcoeff(polynomial);
          term := coefficient*x^exp;
          polynomial := polynomial - term;
          polynomial := collect(polynomial/x^exp, x);
          polynomial := sort(polynomial, x, ascending);
          hornerExpression := subs(a = (coefficient + a)*x^exp, hornerExpression);
      end do;
      hornerExpression := subs(a = 0, hornerExpression);
      return hornerExpression
  end proc
```

```
horner := proc(p::polynom)                                                    (1)
    local polynomial, term, coefficient, exp, hornerExpression::polynom;
    polynomial := sort(p, x, ascending);
    hornerExpression::polynom := a;
    while polynomial <> 0 do
        exp := ldegree(polynomial);
        coefficient := tcoeff(polynomial);
        term := coefficient*x^exp;
        polynomial := polynomial - term;
        polynomial := collect(polynomial/x^exp, x);
        polynomial := sort(polynomial, x, ascending);
        hornerExpression := subs(a = (coefficient + a)*x^exp, hornerExpression)
    end do;
    hornerExpression := subs(a = 0, hornerExpression);
    return hornerExpression
end proc
```

```
> f := x^8 - x^7 + x^4 - x^3 + x + 1 :
> answer := horner(f);
```

$$answer := 1 + \left(1 + \left(-1 + \left(1 + \left(-1 + x\right) x^3\right) x\right) x^2\right) x \quad (2)$$

```
> with(codegen, cost, optimize) : cost(answer);
```

$$5 \text{ additions} + 7 \text{ multiplications} \quad (3)$$

```
> with(codegen, cost, optimize) : cost(f);
```

$$5 \text{ additions} + 18 \text{ multiplications} \quad (4)$$

```
> f := x^400 - x^300 + x^200 - x^100 + 1 :
> answer := horner(f);
```

$$answer := 1 + \left(-1 + \left(1 + \left(x^{100} - 1\right) x^{100}\right) x^{100}\right) x^{100} \quad (5)$$

```
> with(codegen, cost, optimize) : cost(answer);
```

$$4 \text{ additions} + 399 \text{ multiplications} \quad (6)$$

```
> with(codegen, cost, optimize) : cost(f);
```

$$4\ additions + 996\ multiplications \tag{7}$$

```
> f := x^5 + x^8 + x^11 + x^14 :
> answer := horner(f);
```

$$answer := \left(1 + \left(1 + \left(x^3 + 1\right) x^3\right) x^3\right) x^5 \tag{8}$$

```
> with(codegen, cost, optimize) : cost(answer);
```

$$13\ multiplications + 3\ additions \tag{9}$$

```
> with(codegen, cost, optimize) : cost(f);
```

$$3\ additions + 34\ multiplications \tag{10}$$

```
>
>
>
```

2 Taylor's theorem

Write Maple and/or Matlab code to find the Taylor expansion of a $k+1$ differentiable function up to order k, around the point x_0.

Compare your code with the built-in Maple/Matlab commands for computing Taylor expansion.

Illustrate your code by computing the Taylor expansion of the functions:

- $f(x) = cos(x)$, up to degree $k = 10$, around $x_0 = 0$.
- $f(x) = e^x$, up to degree $k = 10$, around $x_0 = 0$.

Answer:

As you can see from the Maple code on the next page, the Taylor Expansion method that was programmed returns the same result as the built in taylor method in maple. When calling this built-in method an order of 11 was used instead of 10 because *taylor* returns the last coefficient as O. This way coefficients up to order 10 are solved for and then converted into a polynomial to get rid of the O term.

```
> taylorExpansion := proc(function, k, a)
    local expression, n, f, fac;
    f := function;
    expression := eval(f, x = a);
    fac := 1;
    for n from 1 to k do
      f := diff(f, x);
      fac := fac·n;
      expression := expression + eval(f, x = a)/fac·(x − a)^n;
    end do;
  return expression;
  end proc;
```

$$taylorExpansion := \mathbf{proc}(function, k, a) \quad \textbf{(1)}$$

```
  local expression, n, f, fac;
  f := function;
  expression := eval(f, x = a);
  fac := 1;
  for n to k do
    f := diff(f, x); fac := fac * n; expression := expression + (eval(f, x = a)) * (x − a)^n
    /fac
  end do;
  return expression
end proc
```

```
> taylorExpansion(exp(x), 10, 0);
```

$$1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 + \frac{1}{720}x^6 + \frac{1}{5040}x^7 + \frac{1}{40320}x^8 + \frac{1}{362880}x^9 + \frac{1}{3628800}x^{10} \quad \textbf{(2)}$$

```
> taylorExpansion(cos(x), 10, 0);
```

$$1 - \frac{1}{2}x^2 + \frac{1}{24}x^4 - \frac{1}{720}x^6 + \frac{1}{40320}x^8 - \frac{1}{3628800}x^{10} \quad \textbf{(3)}$$

```
> convert(taylor(exp(x), x = 0, 11), polynom)
```

$$1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 + \frac{1}{720}x^6 + \frac{1}{5040}x^7 + \frac{1}{40320}x^8 + \frac{1}{362880}x^9 + \frac{1}{3628800}x^{10} \quad \textbf{(4)}$$

```
> convert(taylor(cos(x), x = 0, 11), polynom)
```

$$1 - \frac{1}{2}x^2 + \frac{1}{24}x^4 - \frac{1}{720}x^6 + \frac{1}{40320}x^8 - \frac{1}{3628800}x^{10} \quad \textbf{(5)}$$

```
>
```

3 Polynomials

Consider the cubic polynomial equation

$$f(x) = x^3 - 44x^2 + 564x - 1728$$

1. Prove that equation (1) has three real roots, by exhibiting three sign change intervals, one for each root.

Answer:
According to the Intermediate Value theorem, for a function, $f(x)$ continuous on the interval $[a, b]$ if $f(a)f(b) < 0$, there exists a root r in $[a, b]$ such that $f(r) = 0$.

If we let $a = 4$ and $b = 6$

$$f(4)f(6) < 0$$
$$(-112)(288) < 0$$
$$-32256 < 0$$

Therefore by the Intermediate Value theorem there exists a root p_1 in $[4, 6]$

If we let $a = 6$ and $b = 20$

$$f(6)f(20) < 0$$
$$(288)(-48) < 0$$
$$-13824 < 0$$

Therefore by the Intermediate Value theorem there exists a root p_2 in $[6, 20]$

If we let $a = 4$ and $b = 6$

$$f(20)f(24) < 0$$
$$(-48)(288) < 0$$
$$-13824 < 0$$

Therefore by the Intermediate Value theorem there exists a root p_3 in $[20, 24]$

2. Denote by p_1, p_2, p_3 the three real roots of equation (1).

$$p_1 = 13 - \sqrt{73}$$
$$p_2 = 13 + \sqrt{73}$$
$$p_3 = 18$$

(a) Compute the *exact* values of the sum $p_1 + p_2 + p_3$ and the product $p_1p_2p_3$

$$p_1 + p_2 + p_3 = s_1$$

$$s_i = (-1)^i a_{n-i}/a_n$$
$$s_1 = (-1)a_2/a_3$$
$$s_1 = -(-44)/1$$
$$s_1 = 44$$

$$p_1p_2p_3 = s_3$$
$$s_i = (-1)^i a_{n-i}/a_n$$
$$s_3 = -a_0/a_3$$
$$s_3 = -(-1728/1)$$
$$s_3 = 1728$$

(b) Compute the *exact* values of the 2nd elementary symmetric function $p_1p_2 + p_2p_3 + p_1p_3$

$$p_1p_2 + p_2p_3 + p_1p_3 = s_2$$
$$s_i = (-1)^i a_{n-i}/a_n$$
$$s_2 = a_1/a_3$$
$$s_2 = 564/1$$
$$s_2 = 564$$

(c) Compute the *exact* values of the sums $p_1^2 + p_2^2 + p_3^2$ and $p_1^3 + p_2^3 + p_3^3$

$$(p_1 + p_2 + p_3)^2 = s_1^2$$
$$p_1^2 + 2p_1p_2 + 2p_1p_3 + p2^2 + 2p_2p_3 + p_3^2 = s_1^2$$
$$p_1^2 + p_2^2 + p_3^2 = s_1^2 - 2(p_1p_2 + p_1p_3 + p_2p_3)$$
$$p_1^2 + p_2^2 + p_3^2 = s_1^2 - 2(s_2) = 44^2 - 2(564)$$
$$p_1^2 + p_2^2 + p_3^2 = 808$$

$$(p_1+p_2+p_3)^3 = p_1^3+p_2^3+p_3^3+3(p_1^2p_2+p_1^2p_3+p_2^2p_3+p_1p_2^2+p_1p_3^2+p_2p_3^2)+6(p_1p_2p_3)$$
$$(p_1^2p_2+p_1^2p_3+p_2^2p_3+p_1p_2^2+p_1p_3^2+p_2p_3^2) = (p_1p_2+p_2p_3+p_1p_3)(p_1+p_2+p_3)-3(p_1p_2p_3)$$
$$p_1^3+p_2^3+p_3^3 = (p_1+p_2+p_3)^3-3((p_1p_2+p_2p_3+p_1p_3)(p_1+p_2+p_3)-3(p_1p_2p_3))-6(p_1p_2p_3)$$
$$p_1^3+p_2^3+p_3^3 = (s_1)^3-3((s_2)(s_1)-3(s_3))-6(s_3) = (44)^3-3((564)(44)-3(1728))-6(1728)$$
$$p_1^3 + p_2^3 + p_3^3 = 15920$$

(d) Compute the *exact* value of the quantity $\frac{1}{p_1^2} + \frac{1}{p_2^2} + \frac{1}{p_3^2}$

$$\frac{1}{p_1^2}+\frac{1}{p_2^2}+\frac{1}{p_3^2}=\frac{p_2^2p_3^2+p_1^2p_3^2+p_1^2p_2^2}{(p_1p_2p_3)^2}$$

$$s_2^2=(p_1p_2+p_2p_3+p_1p_3)(p_1p_2+p_2p_3+p_1p_3)=p_2^2p_3^2+p_1^2p_3^2+p_1^2p_2^2+2(p_1^2p_2p_3+p_1p_2^2p_3+p_1p_2p_3^2$$

$$\frac{s_2^2}{s_3^2}=\frac{(p_2^2p_3^2+p_1^2p_3^2+p_1^2p_2^2+2(p_1^2p_2p_3+p_1p_2^2p_3+p_1p_2p_3^2)}{(p_1p_2p_3)^2}=\frac{p_2^2p_3^2+p_1^2p_3^2+p_1^2p_2^2}{(p_1p_2p_3)^2}+\frac{2(p_1+p_2+p_3)}{(p_1p_2p_3)}$$

$$\frac{s_2^2}{s_3^2}=\frac{(p_2^2p_3^2+p_1^2p_3^2+p_1^2p_2^2+2(p_1^2p_2p_3+p_1p_2^2p_3+p_1p_2p_3^2)}{(p_1p_2p_3)^2}=\frac{1}{p_1^2}+\frac{1}{p_2^2}+\frac{1}{p_3^2}+\frac{2s_1}{s_3}$$

$$\frac{1}{p_1^2}+\frac{1}{p_2^2}+\frac{1}{p_3^2}=\frac{s_2^2-2s_1s_3}{s_3^2}=\frac{564^2-2(44)(1728)}{(1728)^2}=\frac{166032}{2985984}=\frac{1153}{20736}$$

4 Equation Solving

4.1 Bisection method

Write Maple and/or Matlab code to implement the bisection method as seen in class. Illustrate your code with the examples seen in class:

- $f(x) = x^3 + x - 1, x \in [0, 1]$.
- $f(x) = cos(x) - x, x \in [0, 1]$.

Use your code to find numerical approximations to each of the three real roots of equation (1), with 6 significant digits of accuracy correct. In each case, report the number of iterations that are needed in the application of the bisection method.

Answer:

As seen in the maple code on the next page

- $f(x) = x^3 + x - 1, x \in [0, 1]$. requires 19 iterations to reach root 0.682329 with 6 significant digits correct
- $f(x) = cos(x) - x, x \in [0, 1]$. requires 19 iterations to reach root 0.739084 with 6 significant digits correct

For the equation (1):

- $f(x) = x^3 - 44x^2 + 564x - 1728, x \in [4, 6]$. requires 20 iterations to reach root p1 4.45600 with 6 significant digits correct
- $f(x) = x^3 - 44x^2 + 564x - 1728, x \in [6, 20]$. requires 23 iterations to reach root p2 18.0000 with 6 significant digits correct
- $f(x) = x^3 - 44x^2 + 564x - 1728, x \in [20, 24]$. requires 21 iterations to reach root p3 21.5440 with 6 significant digits correct

```
> bisection := proc(function, aParam, bParam, precision)
      local expression, emach, c, a, b, iterations;
      a := aParam;
      b := bParam;
      emach := 10^(-precision);
      iterations := 0;
      while emach < 1/2*b - 1/2*a do
          c := evalf(1/2*a + 1/2*b);
          iterations := iterations + 1;
          if eval(function, x=c) = 0 then
              return c
          elif (eval(function, x=c)) * (eval(function, x=a)) < 0 then
              b := c
          else
              a := c
          end if
      end do;
      interface(displayprecision=precision);
      return c, iterations
  end proc
```

```
bisection := proc(function, aParam, bParam, precision)
    local expression, emach, c, a, b, iterations;
    a := aParam;
    b := bParam;
    emach := 10^(-precision);
    iterations := 0;
    while emach < 1/2*b - 1/2*a do
        c := evalf(1/2*a + 1/2*b);
        iterations := iterations + 1;
        if eval(function, x=c) = 0 then
            return c
        elif (eval(function, x=c)) * (eval(function, x=a)) < 0 then
            b := c
        else
            a := c
        end if
    end do;
    interface(displayprecision=precision);
    return c, iterations
end proc
```

(1)

```
> f := x^3 + x - 1
```

$$f := x^3 + x - 1 \tag{2}$$

```
> answer := bisection(f, 0, 1, 6)
```

$$answer := 0.682329,\ 19 \tag{3}$$

```
> f := cos(x) - x
```

(4)

$$f := \cos(x) - x \quad \textbf{(4)}$$

```
> answer := bisection(f, 0, 1, 6)
```

$$answer := 0.739084, 19 \quad \textbf{(5)}$$

```
> f := x^3 - 44*x^2 + 564*x - 1728
```

$$f := x^3 - 44\,x^2 + 564\,x - 1728 \quad \textbf{(6)}$$

```
> p1 := bisection(f, 4, 6, 6)
```

$$p1 := 4.45600, 20 \quad \textbf{(7)}$$

```
> p2 := bisection(f, 6, 20, 6)
```

$$p2 := 18.0000, 23 \quad \textbf{(8)}$$

```
p3 := bisection(f, 20, 24, 6)
```

$$p3 := 21.5440, 21 \quad \textbf{(9)}$$

4.2 Fixed-point iteration method (FPI)

Write Maple and/or Matlab code to implement the FPI method, as seen in class.

Use your FPI code to:

- compute $\sqrt{2}$ with 15 significant digits correct, using the function

$$g(x) = \frac{x + \frac{2}{x}}{2}$$

  Choose a suitable starting point for your iteration.

- find the solution of the equation

$$x^3 = 2x + 2$$

  correct to eight decimal places.

Answer:

As you can see from the Maple code on the next page, using Fixed-Point Iteration with the sample equations above the results were:

- $g(x) = \frac{x+\frac{2}{x}}{2}$: 7 Iterations taken to reach the root 1.41421356237310 = $\sqrt{(2)}$ with 15 significant digits when starting point of $x_0 = 0.5$
- $x^3 = 2x + 2$: This equation was rearranged to get $g(x) = \sqrt[3]{2x + 2}$. 12 Iterations taken to reach the root 1.7692924 with 8 significant digits when starting point of $x_0 = 1.7$

```
> fpi := proc(function, initGuess, precision)
      local a, ai, emach, i, err;
      a := initGuess;
      ai := eval(function, x = a);
      emach := 10^(-precision);
      for i from 2 to 1000 do
          a := ai; ai := eval(function, x = a); err := abs(a - ai); if err < emach then break
          end if
      end do;
      interface(displayprecision = precision);
      return ai, i
  end proc
```

```
fpi := proc(function, initGuess, precision)
    local a, ai, emach, i, err;
    a := initGuess;
    ai := eval(function, x = a);
    emach := 10^(-precision);
    for i from 2 to 1000 do
        a := ai; ai := eval(function, x = a); err := abs(a - ai); if err < emach then break
        end if
    end do;
    interface(displayprecision = precision);
    return ai, i
end proc
```

(1)

```
> f := (x + 2/x)/2;
```

$$f := \frac{x}{2} + \frac{1}{x} \tag{2}$$

```
> answer, iterations := fpi(f, 0.5, 15);
```

$$answer, iterations := 1.41421356237310, 7 \tag{3}$$

```
> f := (2*x + 2)^(1/3);
```

$$f := (2x+2)^{1/3} \tag{4}$$

```
> answer, iterations := fpi(f, 1.7, 8);
```

$$answer, iterations := 1.7692924, 12 \tag{5}$$

```
>
>
>
>
```

4.3 Newton's method

Write Maple and/or Matlab code to implement Newton's method, as seen in class.

- Using your code, illustrate the execution of Newton's method with $x_0 = 0.1$ and $f(x) = x^3 + x - 1$.
- Study the effect of the initial condition on the number of iterations it takes to obtain a numerical approximation to the root of $f(x) = x^3 + x - 1$ with 8 correct digits. Illustrate your conclusions with a graph.
- Using your code to solve the equation $f(x) = x^3 + x - 1$. Find a suitable initial guess so that Newton's method will converge to the root i. Find a suitable initial guess so that Newton's method will converge to the root $-i$.
- Using your code and suitable initial conditions, compute numerical approximations, with 10 digits of accuracy correct, to all three real roots p_1, p_2, p_3 of equation (1) using Newton's method

Answer:

- The execution of Newton's method with $x_0 = 0.1$ and $f(x) = x^3 + x - 1$ returns the root 0.68232780 with 8 significant digits in 6 iterations.
- As seen in the graph displayed along with Maple code in the next 2 pages, the general conclusion is that the number of iterations taken to reach the root increases the greater the difference between the root and the initial condition.
- When the initial guess is set to $i + 1$ the Newton's method converges to the complex root of $f(x) = x^2 + 1$ which is i. When the initial guess is set to $1 - i$ the Newton's method converges to the other complex root of $f(x) = x^2 + 1$ which is $-i$.
- For equation 1, $p_1 = 4.455996254 = 13 - \sqrt{73}$ is found in 6 iterations when the initial guess is 0.1, $p_2 = 21,5440374 = 13 + \sqrt{73}$ is found in 7 iterations when the initial guess is 20.3, $p_3 = 18.00000000$ is found in 5 iterations when the initial guess is 18.4

```
> newton := proc(function, initGuess, precision)
      local f, fprime, emach, x0, xn, i, err;
      f := function;
      fprime := diff(function, x);
      emach := 10^(-precision);
      x0 := initGuess;
      xn := eval(x0 - eval(f, x=x0) / eval(fprime, x=x0));
      for i from 2 to 1000 do
          x0 := xn;
          xn := evalf(x0 - eval(f, x=x0) / eval(fprime, x=x0));
          err := abs(xn - x0);
          if err < emach then break end if
      end do;
      interface(displayprecision=precision);
      return xn, i
  end proc
```

```
newton := proc(function, initGuess, precision)                    (1)
    local f, fprime, emach, x0, xn, i, err;
    f := function;
    fprime := diff(function, x);
    emach := 10^(-precision);
    x0 := initGuess;
    xn := eval(x0 - eval(f, x=x0) / eval(fprime, x=x0));
    for i from 2 to 1000 do
        x0 := xn;
        xn := evalf(x0 - eval(f, x=x0) / eval(fprime, x=x0));
        err := abs(xn - x0);
        if err < emach then break end if
    end do;
    interface(displayprecision=precision);
    return xn, i
end proc
```

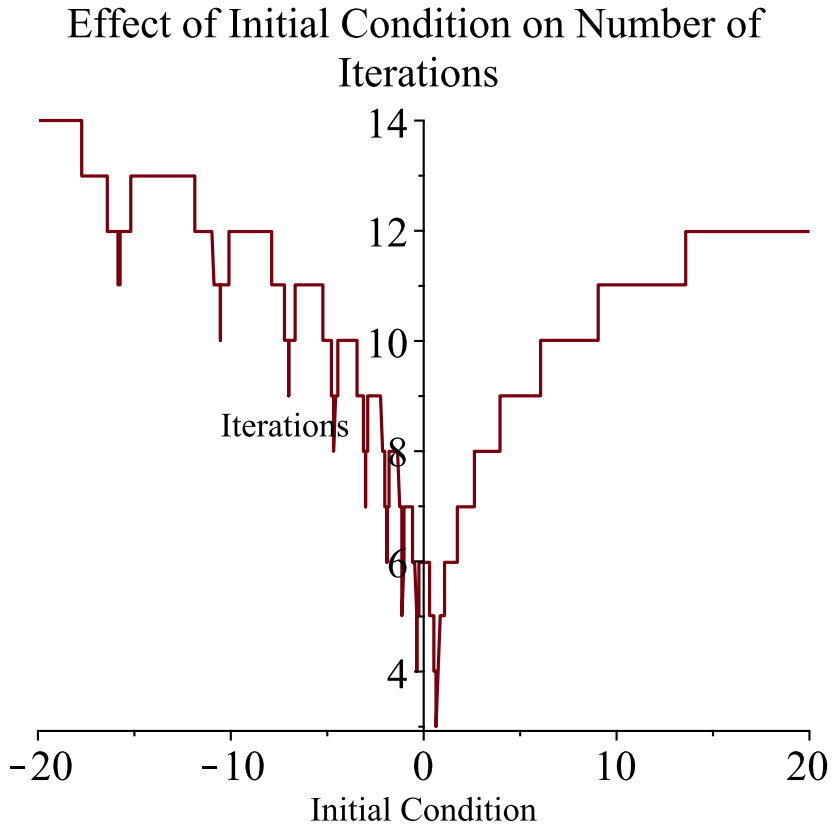
```
> f := x^3 + x - 1
```

$$f := x^3 + x - 1 \quad (2)$$

```
> ans, iterations := newton(f, 0.1, 8);
```

$$ans, iterations := 0.68232780, 6 \quad (3)$$

```
> plot([seq(i, i=-20..20, 0.01)], [seq(newton(f, i, 8)[2], i=-20..20, 0.01)], labels
      = ["Initial Condition", "Iterations"], title
      = "Effect of Initial Condition on Number of Iterations")
```



> $f := x^2 + 1;$

$$f := x^2 + 1 \quad \textbf{(4)}$$

> $ans, iterations := newton(f, 1 + I, 8);$

$$ans, iterations := I, 6 \quad \textbf{(5)}$$

> $ans, iterations := newton(f, 1 - I, 8);$

$$ans, iterations := -I, 6 \quad \textbf{(6)}$$

> $f := x^3 - 44\,x^2 + 564\,x - 1728;$

$$f := x^3 - 44\,x^2 + 564\,x - 1728 \quad \textbf{(7)}$$

> $p1, iterations := newton(f, 0.1, 10);$

$$p1, iterations := 4.455996254, 6 \quad \textbf{(8)}$$

> $p2, iterations := newton(f, 20.3, 10);$

$$p2, iterations := 21.54400374, 7 \quad \textbf{(9)}$$

> $p3, iterations := newton(f, 18.4, 10);$

$$p3, iterations := 18.00000000, 5 \quad \textbf{(10)}$$

>

>

>

4.4 Newton's Method Basins of Attraction (Bonus Question)

The question which initial guesses z_0 in Newton's method lead to convergence in which one of the roots p_1,p_2,p_3 of equation (1), is the question of computing the basins of attraction of each one of the three roots. More specifically, the three basins of attraction of the roots p_1,p_2,p_3 are defined as:

$B_{p1} = z_0 \in C$ Newton's method with initial value z_0 converges to p_1.
$B_{p2} = z_0 \in C$ Newton's method with initial value z_0 converges to p_2.
$B_{p3} = z_0 \in C$ Newton's method with initial value z_0 converges to p_3.

The objective of this question is to produce a relatively accurate drawing of the three basins of attraction B_{p1},B_{p2},B_{p3}, in the region of the complex plane defined by a square of side 60, centered around (0,0).

1. Consider the region of the complex plane, defined by a square of side 60, centered around (0,0). Consider the x-coordinate values x = -30, -29 ,... -1, 0, 1,... 29, 30 and the y-coordinate values y = -30, -29,... -1, 0, 1,...29, 30. For each pair of x,y values, make up the complex number $z_0 = x + yi$ and use it as an initial guess for Newton's method for equation (1).

2. For each Newton's method from the previous step, observe whether the method converges to p_1,$p_2$orp_3 or if it doesn't converge. Use the same accuracy"for all Newton's methods. Via these observations, classify each $z_0$into one of the sets(basins of attraction)B_{p1},B_{p2},B_{p3}, or disregard it (case of non-convergence).

3. Assign a different color to each basin of attractionB_{p1},B_{p2},$B_{p3}$and plot the points belonging to these sets, using your color scheme. This should give you an approximate picture of the required intersection.

4. Repeat the process with refining further the subdivision of the intervals on the x and y axis, to include for instance points with floating point coordinates, e.g. -30, 29.5, 29,..., 1, 0.5, 0, 0.5, 1,..., 29.5, 30. This should yield a more accurate picture.

Answer:
See the following pages for the graphs showing the Basins of Attraction for Equation (1):

- B_{p1} is coloured blue in the graphs and shows which $x + yi$ combinations converged towards root $p_1 = 13 - \sqrt{73}$
- B_{p2} is coloured red in the graphs and shows which $x + yi$ combinations converged towards $p_2 = 13 + \sqrt{73}$
- B_{p3} is coloured green in the graphs and shows which $x + yi$ combinations converged towards $p_3 = 18$

```
> newton := proc(function, initGuess, precision)
      local f, fprime, emach, x0, xn, i, err;
      f := function;
      fprime := diff(function, x);
      emach := 10^(-precision);
      x0 := initGuess;
      xn := evalf(x0 - eval(f, x=x0) / eval(fprime, x=x0));
      for i from 2 to 1000 do
          x0 := xn;
          xn := evalf(x0 - eval(f, x=x0) / eval(fprime, x=x0));
          err := abs(xn - x0);
          if err < emach then break end if
      end do;
      interface(displayprecision=precision);
      return xn, i
  end proc
```

```
newton := proc(function, initGuess, precision)                    (1)
    local f, fprime, emach, x0, xn, i, err;
    f := function;
    fprime := diff(function, x);
    emach := 10^(-precision);
    x0 := initGuess;
    xn := evalf(x0 - eval(f, x=x0) / eval(fprime, x=x0));
    for i from 2 to 1000 do
        x0 := xn;
        xn := evalf(x0 - eval(f, x=x0) / eval(fprime, x=x0));
        err := abs(xn - x0);
        if err < emach then break end if
    end do;
    interface(displayprecision=precision);
    return xn, i
end proc
```

```
> f := x^3 - 44*x^2 + 564*x - 1728;
```

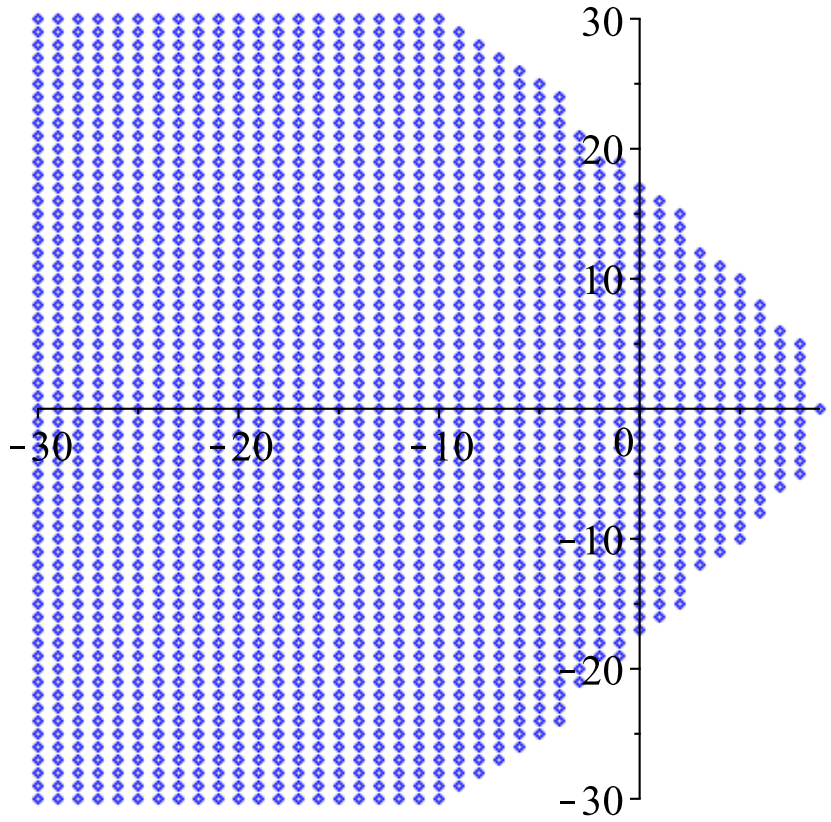
$$f := x^3 - 44\,x^2 + 564\,x - 1728 \qquad (2)$$

```
> Bp1 := [ ];
  Bp2 := [ ];
  Bp3 := [ ];
  for a from -30 to 30 do:
    for b from -30 to 30 do:
      root := Re(newton(f, a + b*I, 6)[1]);
      if root > 4.4 and root < 4.5 then
        Bp1 := [op(Bp1), [a, b]];
      elif root > 21.5 and root < 21.6 then
        Bp2 := [op(Bp2), [a, b]];
      elif root > 17.9 and root < 18.1 then
        Bp3 := [op(Bp3), [a, b]];
      end if
```

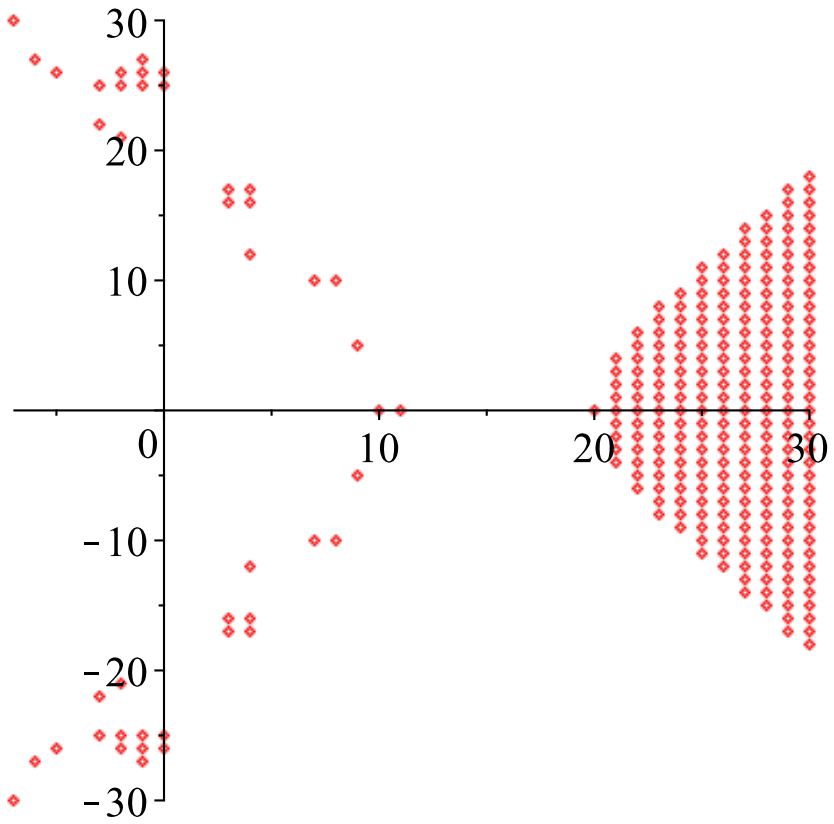
```
    end do;
  end do;
```

$$Bp1 := [\,]$$
$$Bp2 := [\,]$$
$$Bp3 := [\,] \quad \textbf{(3)}$$

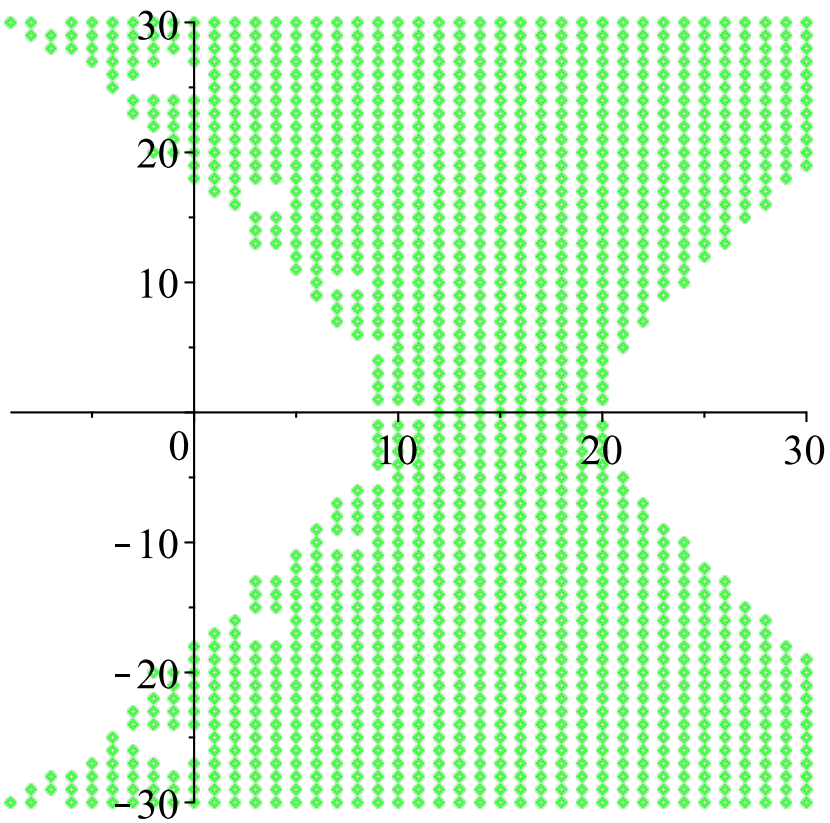
```
> g1 := pointplot(Bp1, color = blue);
```



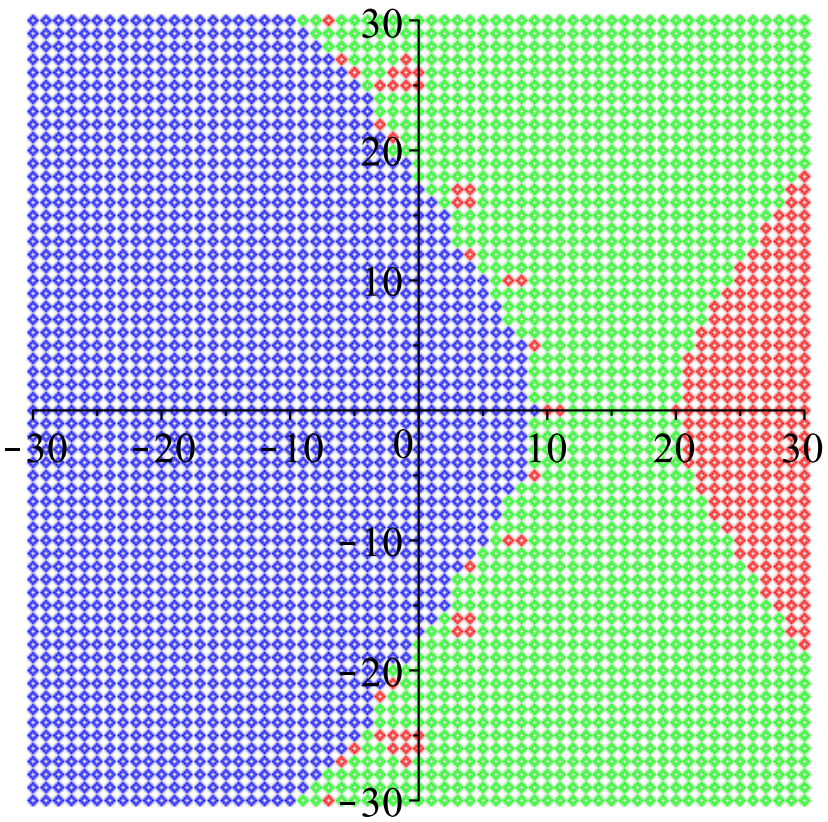

```
> g2 := pointplot(Bp2, color = red);
```




```
> g3 := pointplot(Bp3, color = green);
```



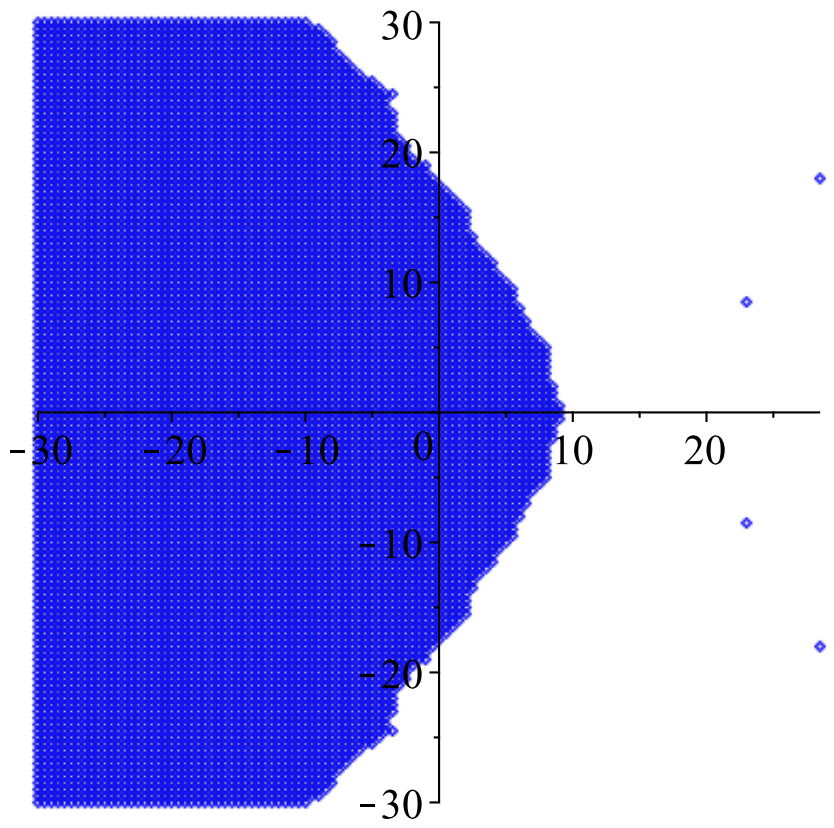

```
> display(g1, g2, g3)
```



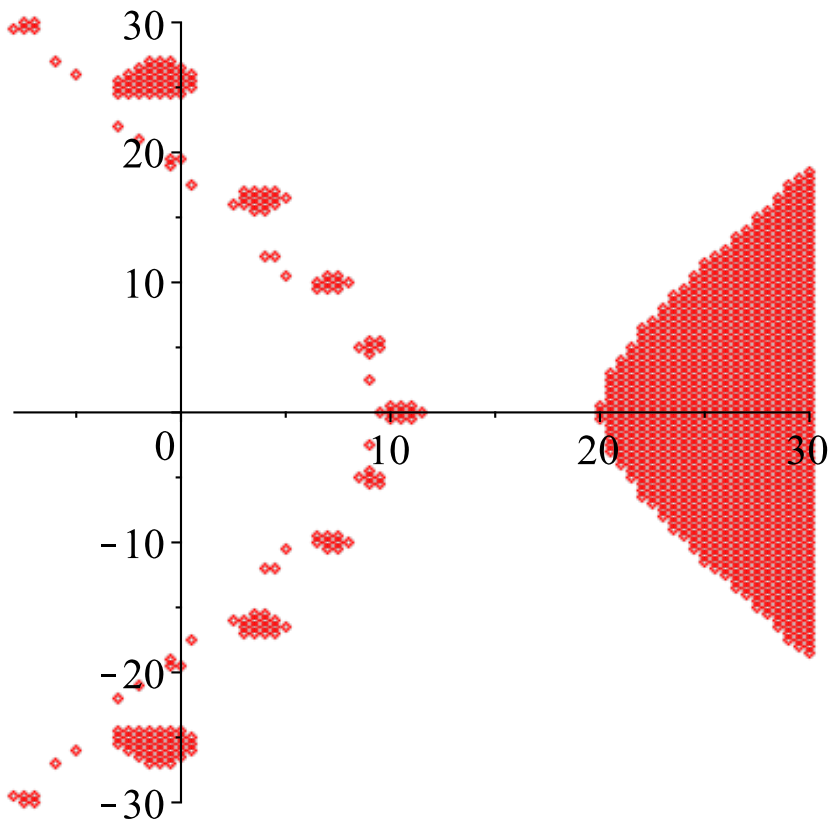

```
> Bp1 := [ ];
  Bp2 := [ ];
  Bp3 := [ ];
  for a from -30 to 30 by 0.5 do:
    for b from -30 to 30 by 0.5 do :
      root := Re(newton(f, a + b·I, 6)[1]);
      if root > 4.4 and root < 4.5 then
        Bp1 := [op(Bp1), [a, b]];
      elif root > 21.5 and root < 21.6 then
        Bp2 := [op(Bp2), [a, b]];
      elif root > 17.9 and root < 18.1 then
        Bp3 := [op(Bp3), [a, b]];
      end if
    end do;
  end do;
```

$$Bp1 := [\,]$$
$$Bp2 := [\,]$$
$$Bp3 := [\,] \qquad \textbf{(4)}$$

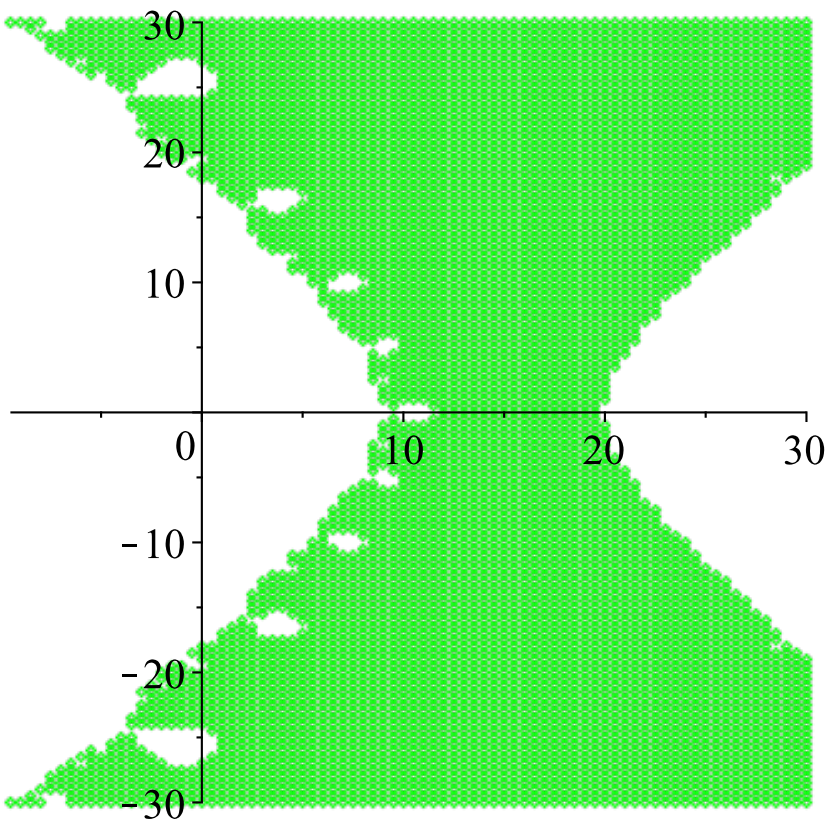
```
> g1 := pointplot(Bp1, color = blue);
```



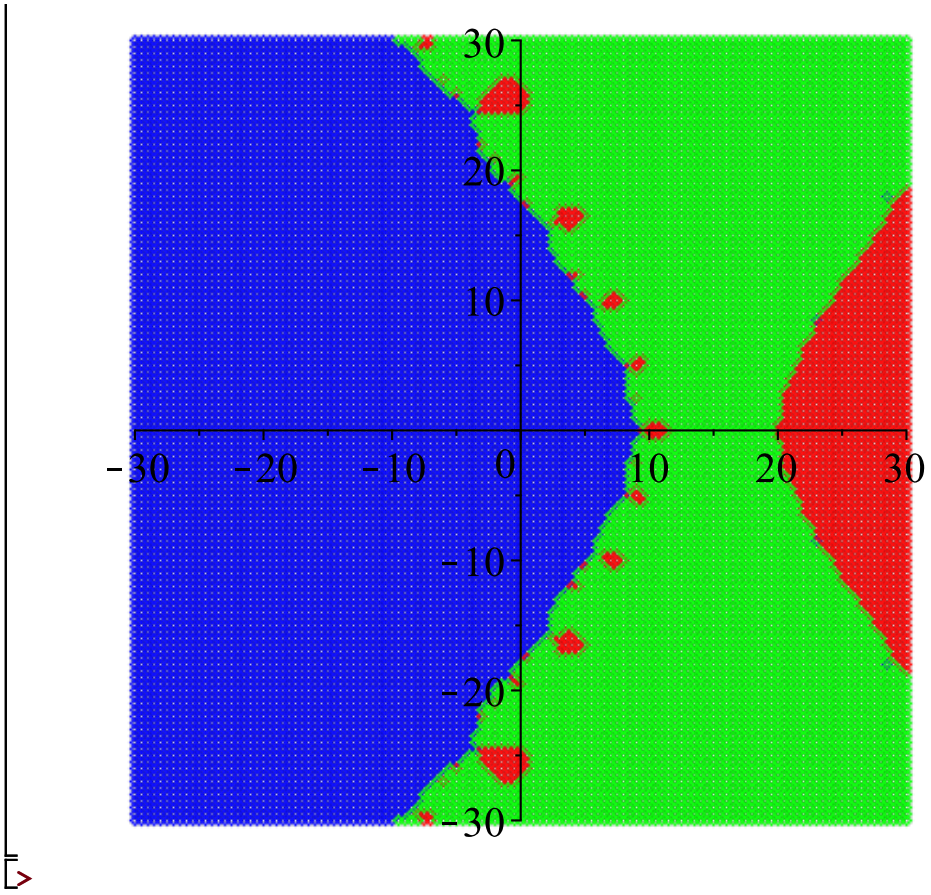

```
> g2 := pointplot(Bp2, color = red);
```




```
> g3 := pointplot(Bp3, color = green);
```




```
> display(g1, g2, g3);
```


30
20
10
-30
-20
-10
0
10
20
30
-10
-20
-30

5 Systems of Linear Equations

5.1 Gauss Elimination

Write Maple and/or Matlab code to implement Gauss elimination, as seen in class.
Answer:

See the following pages for the Maple code. The Gauss Elimination procedure was tested by utilizing the built-in GaussianElimination(m) method in Maple to ensure that the procedure I created was correct.

```
> with(LinearAlgebra);
```

(1)

```
[&x, Add, Adjoint, BackwardSubstitute, BandMatrix, Basis, BezoutMatrix, BidiagonalForm,
  BilinearForm, CARE, CharacteristicMatrix, CharacteristicPolynomial, Column,
  ColumnDimension, ColumnOperation, ColumnSpace, CompanionMatrix,
  CompressedSparseForm, ConditionNumber, ConstantMatrix, ConstantVector, Copy,
  CreatePermutation, CrossProduct, DARE, DeleteColumn, DeleteRow, Determinant,
  Diagonal, DiagonalMatrix, Dimension, Dimensions, DotProduct, EigenConditionNumbers,
  Eigenvalues, Eigenvectors, Equal, ForwardSubstitute, FrobeniusForm,
  FromCompressedSparseForm, FromSplitForm, GaussianElimination, GenerateEquations,
  GenerateMatrix, Generic, GetResultDataType, GetResultShape, GivensRotationMatrix,
  GramSchmidt, HankelMatrix, HermiteForm, HermitianTranspose, HessenbergForm,
  HilbertMatrix, HouseholderMatrix, IdentityMatrix, IntersectionBasis, IsDefinite,
  IsOrthogonal, IsSimilar, IsUnitary, JordanBlockMatrix, JordanForm, KroneckerProduct,
  LA_Main, LUDecomposition, LeastSquares, LinearSolve, LyapunovSolve, Map, Map2,
  MatrixAdd, MatrixExponential, MatrixFunction, MatrixInverse, MatrixMatrixMultiply,
  MatrixNorm, MatrixPower, MatrixScalarMultiply, MatrixVectorMultiply,
  MinimalPolynomial, Minor, Modular, Multiply, NoUserValue, Norm, Normalize,
  NullSpace, OuterProductMatrix, Permanent, Pivot, PopovForm, ProjectionMatrix,
  QRDecomposition, RandomMatrix, RandomVector, Rank, RationalCanonicalForm,
  ReducedRowEchelonForm, Row, RowDimension, RowOperation, RowSpace, ScalarMatrix,
  ScalarMultiply, ScalarVector, SchurForm, SingularValues, SmithForm, SplitForm,
  StronglyConnectedBlocks, SubMatrix, SubVector, SumBasis, SylvesterMatrix,
  SylvesterSolve, ToeplitzMatrix, Trace, Transpose, TridiagonalForm, UnitVector,
  VandermondeMatrix, VectorAdd, VectorAngle, VectorMatrixMultiply, VectorNorm,
  VectorScalarMultiply, ZeroMatrix, ZeroVector, Zip]
```

```
> gaussElimination := proc(m::Matrix)
    local matrix, rows, columns, currentRow, levelRow, levelColumn;
    matrix := m;
    levelRow := 1;
    levelColumn := 1;
    rows, columns := Student:-LinearAlgebra:-Dimension(m);
    with(Student[LinearAlgebra]);
    while levelColumn <= columns and levelRow <= rows do
        currentRow := levelRow;
        while currentRow <= rows and matrix[currentRow, levelColumn] = 0 do
            currentRow := currentRow + 1
        end do;
        if currentRow <= rows then
            matrix := Student:-LinearAlgebra:-SwapRow(matrix, currentRow, levelRow)
        end if;
        if matrix[levelRow, levelColumn] <> 0 then
            currentRow := levelRow + 1;
            while currentRow <= rows do
                matrix[currentRow] := matrix[currentRow] - matrix[levelRow]
                 * matrix[currentRow, levelColumn] / matrix[levelRow, levelColumn];
                currentRow := currentRow + 1
```

```
            end do
        end if;
        levelRow := levelRow + 1;
        levelColumn := levelColumn + 1
    end do;
    return matrix
end proc
```

```
gaussElimination := proc(m::Matrix)                                    (2)
    local matrix, rows, columns, currentRow, levelRow, levelColumn;
    matrix := m;
    levelRow := 1;
    levelColumn := 1;
    rows, columns := Student:-LinearAlgebra:-Dimension(m);
    with(Student[LinearAlgebra]);
    while levelColumn <= columns and levelRow <= rows do
        currentRow := levelRow;
        while currentRow <= rows and matrix[currentRow, levelColumn] = 0 do
            currentRow := currentRow + 1
        end do;
        if currentRow <= rows then
            matrix := Student:-LinearAlgebra:-SwapRow(matrix, currentRow, levelRow)
        end if;
        if matrix[levelRow, levelColumn] <> 0 then
            currentRow := levelRow + 1;
            while currentRow <= rows do
                matrix[currentRow] := matrix[currentRow] - matrix[levelRow] * matrix
                [currentRow, levelColumn] / matrix[levelRow, levelColumn];
                currentRow := currentRow + 1
            end do
        end if;
        levelRow := levelRow + 1;
        levelColumn := levelColumn + 1
    end do;
    return matrix
end proc
```

```
> m := Matrix([[0, 3, -6, 6, 4, -5], [3, -7, 8, -5, 8, 9], [3, -9, 12, -9, 6, 15]])
```

$$m := \begin{bmatrix} 0 & 3 & -6 & 6 & 4 & -5 \\ 3 & -7 & 8 & -5 & 8 & 9 \\ 3 & -9 & 12 & -9 & 6 & 15 \end{bmatrix} \quad (3)$$

```
> gaussElimination(m)
```

$$\begin{bmatrix} 3 & -7 & 8 & -5 & 8 & 9 \\ 0 & 3 & -6 & 6 & 4 & -5 \\ 0 & 0 & 0 & 0 & \frac{2}{3} & \frac{8}{3} \end{bmatrix} \quad \textbf{(4)}$$

```
> GaussianElimination(m)
```

$$\begin{bmatrix} 3 & -7 & 8 & -5 & 8 & 9 \\ 0 & 3 & -6 & 6 & 4 & -5 \\ 0 & 0 & 0 & 0 & \frac{2}{3} & \frac{8}{3} \end{bmatrix} \quad \textbf{(5)}$$

```
>
```

5.2 Gauss-Jordan Elimination

Write Maple and/or Matlab code to implement Gauss-Jordan elimination, as seen in class.

Answer:

See the following pages for the Maple code. The Gauss Elimination procedure was tested by utilizing the built-in ReducedRowEchelonForm(m) method in Maple to ensure that the procedure I created was correct.

```
> with(Student[LinearAlgebra]) :
> gaussJordanElimination := proc(m::Matrix)
    local matrix, rows, columns, currentRow, levelRow, levelColumn;
    matrix := m;
    levelRow := 1;
    levelColumn := 1;
    rows, columns := LinearAlgebra:-Dimension(m);
    with(Student[LinearAlgebra]);
    while levelColumn <= columns and levelRow <= rows do
        currentRow := levelRow;
        while currentRow <= rows and matrix[currentRow, levelColumn] = 0 do
            currentRow := currentRow + 1
        end do;
        if currentRow <= rows then
            matrix := Student:-LinearAlgebra:-SwapRow(matrix, currentRow, levelRow)
        end if;
        if matrix[levelRow, levelColumn] <> 0 then
            matrix[levelRow] := matrix[levelRow] / matrix[levelRow, levelColumn];
            currentRow := 1;
            while currentRow <= rows do
                if currentRow <> levelRow then
                    matrix[currentRow] := - matrix[levelRow] * matrix[currentRow,
                    levelColumn] + matrix[currentRow]
                end if;
                currentRow := currentRow + 1
            end do;
            levelRow := levelRow + 1
        end if;
        levelColumn := levelColumn + 1;
    end do;
    return matrix
end proc
```

```
gaussJordanElimination := proc(m::Matrix)
    local matrix, rows, columns, currentRow, levelRow, levelColumn;
    matrix := m;
    levelRow := 1;
    levelColumn := 1;
    rows, columns := LinearAlgebra:-Dimension(m);
    with(Student[LinearAlgebra]);
    while levelColumn <= columns and levelRow <= rows do
        currentRow := levelRow;
        while currentRow <= rows and matrix[currentRow, levelColumn] = 0 do
            currentRow := currentRow + 1
        end do;
        if currentRow <= rows then
            matrix := Student:-LinearAlgebra:-SwapRow(matrix, currentRow, levelRow)
        end if;
        if matrix[levelRow, levelColumn] <> 0 then
            matrix[levelRow] := matrix[levelRow] / matrix[levelRow, levelColumn];
            currentRow := 1;
            while currentRow <= rows do
                if currentRow <> levelRow then
                    matrix[currentRow] := - matrix[levelRow] * matrix[currentRow,
                    levelColumn] + matrix[currentRow]
                end if;
                currentRow := currentRow + 1
            end do;
            levelRow := levelRow + 1
        end if;
        levelColumn := levelColumn + 1
    end do;
    return matrix
end proc
```

(1)

```
> m := Matrix([[0, 3, -6, 6, 4, -5], [3, -7, 8, -5, 8, 9], [3, -9, 12, -9, 6, 15]])
```

$$m := \begin{bmatrix} 0 & 3 & -6 & 6 & 4 & -5 \\ 3 & -7 & 8 & -5 & 8 & 9 \\ 3 & -9 & 12 & -9 & 6 & 15 \end{bmatrix}$$

(2)

```
> gaussJordanElimination(m)
```

(3)

$$\begin{bmatrix} 1 & 0 & -2 & 3 & 0 & -24 \\ 0 & 1 & -2 & 2 & 0 & -7 \\ 0 & 0 & 0 & 0 & 1 & 4 \end{bmatrix} \quad (3)$$

> $ReducedRowEchelonForm(m)$

$$\begin{bmatrix} 1 & 0 & -2 & 3 & 0 & -24 \\ 0 & 1 & -2 & 2 & 0 & -7 \\ 0 & 0 & 0 & 0 & 1 & 4 \end{bmatrix} \quad (4)$$

>

5.3 The Hilbert Matrix

The **Hilbert matrix** $H_n = (h_{ij})$ is a square n x n symmetric matrix with $(h_{ij} = \frac{1}{i+j-1}$,for $i, j = 1, ...n$.

For example, for $n = 5$, the Hilbert matrix of order 5 x 5 is:

$$H_5 = \begin{bmatrix} 1 & 1/2 & 1/3 & 1/4 & 1/5 \\ 1/2 & 1/3 & 1/4 & 1/5 & 1/6 \\ 1/3 & 1/4 & 1/5 & 1/6 & 1/7 \\ 1/4 & 1/5 & 1/6 & 1/7 & 1/8 \\ 1/5 & 1/6 & 1/7 & 1/8 & 1/9 \end{bmatrix}$$

1. Use your Gauss Elimination and Gauss-Jordan Elimination implementations to solve the following system of linear equations:

$$H_5 \cdot \vec{x} = \vec{1}$$

2. Use your implementation to compute the determinant of H_5, using Gauss elimination. Compare your result with the determinant as computed by built-in Maple or Matlab commands.

3. Use your implementation to compute the inverse matrix H_5^{-1}, using Gauss-Jordan elimination, by appending the I_5 identity matrix to H_5. Compare your result with the inverse matrix as computed by built-in Maple or Matlab commands.

4. Compute the **condition number** of H_5 by the formula $(H_5) = ||H5|| \cdot ||H_5^{-1}||$, for two different norms: (1) the 1-norm (2) the Frobenius norm.

Answer:

1. By utilizing the Gauss-Jordan Elimination method to reduce the matrix H_5 to reduced row echelon form the following solution was found:

$$\vec{x} = \begin{bmatrix} 5 \\ -120 \\ 630 \\ -1120 \\ 630 \end{bmatrix}$$

2. The determinant of H_5 was found by multiplying the diagonal elements of the output received from conducting a Gauss Elimination on H_5. Since there were no swaps made in the elimination this multiplication did not need to be multiplied by -1. The determinant I computed was checked by using the built=in Determinant() method in Maple.

3. By appending the I_5 identity matrix to H_5 reducing this matrix to reduced row-echelon form, the inverse of H_5 was found. This was checked by utilizing the built-in Compare your result with the inverse matrix as computed by built-in MatrixInverse() method in Maple.

4. Corresponding methods were created to calculate the 1-norm and Frobenius norm of H_5. These methods were checked for correctness by utilizing the MatrixNorm(H5,Frobenius) and MatrixNorm(H5,1) methods in Maple. From these calculations it is shown that the condition number of the 1-norm of H_5 is 943656 and the condition number of the Frobenius norm of H_5 is $\frac{\sqrt{15871330}\sqrt{3700542505}}{504}$.

```
> with(Student[LinearAlgebra]) :
> gaussJordanElimination := proc(m::Matrix)
    local matrix, rows, columns, currentRow, levelRow, levelColumn;
    matrix := m;
    levelRow := 1;
    levelColumn := 1;
    rows, columns := LinearAlgebra:-Dimension(m);
    with(Student[LinearAlgebra]);
    while levelColumn <= columns and levelRow <= rows do
        currentRow := levelRow;
        while currentRow <= rows and matrix[currentRow, levelColumn] = 0 do
            currentRow := currentRow + 1
        end do;
        if currentRow <= rows then
            matrix := Student:-LinearAlgebra:-SwapRow(matrix, currentRow, levelRow)
        end if;
        if matrix[levelRow, levelColumn] <> 0 then
            matrix[levelRow] := matrix[levelRow] / matrix[levelRow, levelColumn];
            currentRow := 1;
            while currentRow <= rows do
                if currentRow <> levelRow then
                    matrix[currentRow] := - matrix[levelRow] * matrix[currentRow,
                    levelColumn] + matrix[currentRow]
                end if;
                currentRow := currentRow + 1
            end do;
            levelRow := levelRow + 1
        end if;
        levelColumn := levelColumn + 1;
    end do;
    return matrix
end proc
```

```
gaussJordanElimination := proc(m::Matrix)                                    (1)
    local matrix, rows, columns, currentRow, levelRow, levelColumn;
    matrix := m;
    levelRow := 1;
    levelColumn := 1;
    rows, columns := LinearAlgebra:-Dimension(m);
    with(Student[LinearAlgebra]);
    while levelColumn <= columns and levelRow <= rows do
        currentRow := levelRow;
        while currentRow <= rows and matrix[currentRow, levelColumn] = 0 do
            currentRow := currentRow + 1
        end do;
        if currentRow <= rows then
            matrix := Student:-LinearAlgebra:-SwapRow(matrix, currentRow, levelRow)
        end if;
        if matrix[levelRow, levelColumn] <> 0 then
            matrix[levelRow] := matrix[levelRow] / matrix[levelRow, levelColumn];
            currentRow := 1;
            while currentRow <= rows do
                if currentRow <> levelRow then
                    matrix[currentRow] := - matrix[levelRow] * matrix[currentRow,
                    levelColumn] + matrix[currentRow]
                end if;
                currentRow := currentRow + 1
            end do;
            levelRow := levelRow + 1
        end if;
        levelColumn := levelColumn + 1
    end do;
    return matrix
end proc
```

```
> gaussElimination :=proc(m::Matrix)
      local matrix, rows, columns, currentRow, levelRow, levelColumn;
      matrix := m;
      levelRow := 1;
      levelColumn := 1;
      rows, columns := Student:-LinearAlgebra:-Dimension(m);
      with(Student[LinearAlgebra]);
      while levelColumn <= columns and levelRow <= rows do
          currentRow := levelRow;
          while currentRow <= rows and matrix[currentRow, levelColumn] = 0 do
              currentRow := currentRow + 1
          end do;
          if currentRow <= rows then
```

```
            matrix := Student:-LinearAlgebra:-SwapRow(matrix, currentRow, levelRow)
        end if;
        if matrix[levelRow, levelColumn] <> 0 then
            currentRow := levelRow + 1;
            while currentRow <= rows do
                matrix[currentRow] := matrix[currentRow] - matrix[levelRow]
                 * matrix[currentRow, levelColumn] / matrix[levelRow, levelColumn];
                currentRow := currentRow + 1
            end do
        end if;
        levelRow := levelRow + 1;
        levelColumn := levelColumn + 1;
    end do;
    return matrix
end proc
```

(2)

```
gaussElimination := proc(m::Matrix)
    local matrix, rows, columns, currentRow, levelRow, levelColumn;
    matrix := m;
    levelRow := 1;
    levelColumn := 1;
    rows, columns := Student:-LinearAlgebra:-Dimension(m);
    with(Student[LinearAlgebra]);
    while levelColumn <= columns and levelRow <= rows do
        currentRow := levelRow;
        while currentRow <= rows and matrix[currentRow, levelColumn] = 0 do
            currentRow := currentRow + 1
        end do;
        if currentRow <= rows then
            matrix := Student:-LinearAlgebra:-SwapRow(matrix, currentRow, levelRow)
        end if;
        if matrix[levelRow, levelColumn] <> 0 then
            currentRow := levelRow + 1;
            while currentRow <= rows do
                matrix[currentRow] := matrix[currentRow] - matrix[levelRow] * matrix
                [currentRow, levelColumn] / matrix[levelRow, levelColumn];
                currentRow := currentRow + 1
            end do
        end if;
        levelRow := levelRow + 1;
        levelColumn := levelColumn + 1
    end do;
    return matrix
end proc
```

> $H5\_equals\_1 := \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & 1 \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & 1 \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & 1 \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & 1 \\ \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} & 1 \end{bmatrix}$

$$H5\_equals\_1 := \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & 1 \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & 1 \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & 1 \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & 1 \\ \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} & 1 \end{bmatrix} \quad \textbf{(3)}$$

> $gaussJordanElimination(H5\_equals\_1)$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 5 \\ 0 & 1 & 0 & 0 & 0 & -120 \\ 0 & 0 & 1 & 0 & 0 & 630 \\ 0 & 0 & 0 & 1 & 0 & -1120 \\ 0 & 0 & 0 & 0 & 1 & 630 \end{bmatrix} \quad \textbf{(4)}$$

> $H5 := \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} \\ \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} \end{bmatrix}$

$$H5 := \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} \\ \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} \end{bmatrix} \quad \textbf{(5)}$$

> $H5\_upper\_triangle := gaussElimination(H5)$

$$H5\_upper\_triangle := \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ 0 & \frac{1}{12} & \frac{1}{12} & \frac{3}{40} & \frac{1}{15} \\ 0 & 0 & \frac{1}{180} & \frac{1}{120} & \frac{1}{105} \\ 0 & 0 & 0 & \frac{1}{2800} & \frac{1}{1400} \\ 0 & 0 & 0 & 0 & \frac{1}{44100} \end{bmatrix} \quad \textbf{(6)}$$

> $determinant := H5\_upper\_triangle[1,1] \cdot H5\_upper\_triangle[2,2] \cdot H5\_upper\_triangle[3,3] \cdot H5\_upper\_triangle[4,4] \cdot H5\_upper\_triangle[5,5];$

$$determinant := \frac{1}{266716800000} \quad \textbf{(7)}$$

> $Determinant(H5)$

$$\frac{1}{266716800000} \quad \textbf{(8)}$$

> $H5\_appended\_inverse :=$

$$\begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & 1 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & 0 & 1 & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & 0 & 0 & 1 & 0 & 0 \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & 0 & 0 & 0 & 1 & 0 \\ \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$H5\_appended\_inverse := \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & 1 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & 0 & 1 & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & 0 & 0 & 1 & 0 & 0 \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & 0 & 0 & 0 & 1 & 0 \\ \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (9)$$

```
> H5_inverse := gaussJordanElimination(H5_appended_inverse)[1..5, 6..10]
```

$$H5\_inverse := \begin{bmatrix} 25 & -300 & 1050 & -1400 & 630 \\ -300 & 4800 & -18900 & 26880 & -12600 \\ 1050 & -18900 & 79380 & -117600 & 56700 \\ -1400 & 26880 & -117600 & 179200 & -88200 \\ 630 & -12600 & 56700 & -88200 & 44100 \end{bmatrix} \quad (10)$$

```
> MatrixInverse(H5);
```

$$\begin{bmatrix} 25 & -300 & 1050 & -1400 & 630 \\ -300 & 4800 & -18900 & 26880 & -12600 \\ 1050 & -18900 & 79380 & -117600 & 56700 \\ -1400 & 26880 & -117600 & 179200 & -88200 \\ 630 & -12600 & 56700 & -88200 & 44100 \end{bmatrix} \quad (11)$$

```
> norm1 :=proc(m::Matrix)
      local rows, columns, answer, total, matrix, i, j;
      rows, columns := Student:-LinearAlgebra:-Dimension(m);
      matrix := m;
      answer := 0;
      for i to columns do
          total := 0;
          for j to rows do total := total + abs(matrix[j, i]) end do;
          if answer < total then answer := total end if
      end do;
      return answer
  end proc
```

```
norm1 := proc(m::Matrix)                                   (12)
    local rows, columns, answer, total, matrix, i, j;
    rows, columns := Student:-LinearAlgebra:-Dimension(m);
    matrix := m;
    answer := 0;
    for i to columns do
```

```
        total := 0;
        for j to rows do total := total + abs(matrix[j, i]) end do;
        if answer < total then answer := total end if
    end do;
    return answer
end proc
```

```
> conditionNumber1 := norm1(H5)·norm1(H5_inverse);
```

$$conditionNumber1 := 943656 \tag{13}$$

```
> MatrixNorm(H5, 1)·MatrixNorm(H5_inverse, 1);
```

$$943656 \tag{14}$$

```
> frob := proc(m::Matrix)
      local rows, columns, answer, matrix, i, j;
      rows, columns := Student:-LinearAlgebra:-Dimension(m);
      matrix := m;
      answer := 0;
      for i to columns do
          for j to rows do
              answer := answer + matrix[j, i]^2
          end do
      end do;
      answer := sqrt(answer);
      return answer
  end proc
```

```
frob := proc(m::Matrix)
    local rows, columns, answer, matrix, i, j;
    rows, columns := Student:-LinearAlgebra:-Dimension(m);
    matrix := m;
    answer := 0;
    for i to columns do for j to rows do answer := answer + matrix[j, i]^2 end do end do;
    answer := sqrt(answer);
    return answer
end proc
```

(15)

```
> conditionNumberFrobenius := frob(H5)·frob(H5_inverse);
```

$$conditionNumberFrobenius := \frac{\sqrt{15871330}\,\sqrt{3700542505}}{504} \tag{16}$$

```
> MatrixNorm(H5, Frobenius)·MatrixNorm(H5_inverse, Frobenius)
```

$$\frac{\sqrt{15871330}\,\sqrt{3700542505}}{504} \tag{17}$$

```
>
```