question3

June 19, 2021

0.1 Defining the Polynomial

[1]:
```
var('x, a0,a1,a2,a3,a4,a5,a6')
x = polygen(ZZ)
eq = a0 + a1*x + a2*x^2 + a3*x^3 + a4*x^4 + a5*x^5 + a6*x^6
```

[2]:
```
eq.show()
```

$$a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$$

[3]:
```
eq1 = eq.subs([a0==1],[a1==-5],[a2==0],[a3==0],[a4==0],[a5==0],[a6==1])
eq1.show()
```

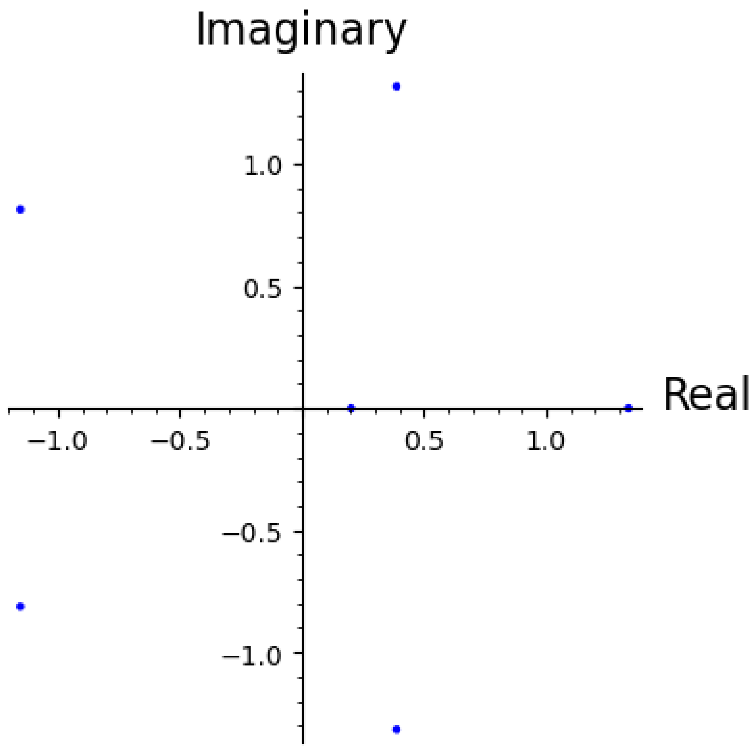
$$x^6 - 5\,x + 1$$

0.1.1 Solving for the given polynomial

[4]:
```
x = polygen(ZZ)
eq1 = x^6-5*x+1
eq1.roots(ring=CIF)
```

[4]:
```
[(0.2000128049178756?, 1),
 (1.335687296057259?, 1),
 (-1.153570400482396? - 0.812646548862507?*I, 1),
 (-1.153570400482396? + 0.812646548862507?*I, 1),
 (0.385720349994829? - 1.315729182385585?*I, 1),
 (0.385720349994829? + 1.315729182385585?*I, 1)]
```

[5]:
```
p = Graphics()
for i in eq1.roots(ring=CIF):
    p+= i[0].plot()

p.show(axes_labels=['Real','Imaginary'])
p.save("Roots.png")
```



0.2 Enclosing Circle

Iterating through two and three points at a time and finding if the circumcircle encompasses all other points

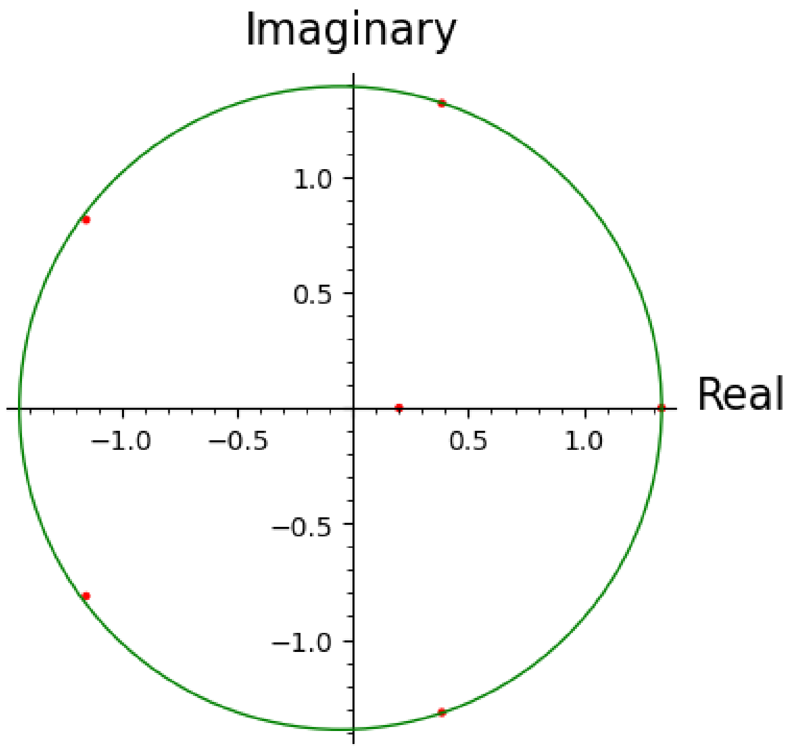
[6]:
```
# Inspired from GeeksforGeeks

def dist(a, b):
    # Returns distance between two points
    return sqrt( (a[0]-b[0])**2 + (a[1]-b[1])**2)


def is_inside(c, p):
    # Checks if point p lies inside circle
    return dist(c[0], p) <= c[1]


def get_circle_center(bx, by, cx, cy):
    # Returns the centre of circumcircle of A, B, C
    B = bx * bx + by * by
    C = cx * cx + cy * cy
    D = bx * cy - by * cx
    return [(cy * B - by * C) / (2 * D), (bx * C - cx * B) / (2 * D) ]
```

```python
def circle_from_3(A, B, C):
    I = get_circle_center(B[0] - A[0], B[1] - A[1], C[0] - A[0], C[1] - A[1])
    I[0] += A[0]
    I[1] += A[1]
    return [I, dist(I, A)]

def circle_from_2(A, B):
    C = [ (A[0] + B[0]) / 2.0, (A[1] + B[1]) / 2.0]
    return [C, dist(A, B) / 2.0]

def is_valid_circle(c, P):
    for p in P:
        if (is_inside(c, p) == False):
            return False
    return True

def get_farthest(P):
    # Returns the farthest point from origin
    n = len(P)
    far = -1
    for i in range(n):
        far = max(far, dist([0,0], P[i]))

    return far

def minimum_enclosing_circle(P):
    # Iterates through all possible circles and finds the Enclosing Circle

    INF = get_farthest(P)

    n = len(P)
    if (n == 0):
        return [[0, 0], 0]
    if (n == 1):
        return [P[0], 0]

    mec = [[0, 0], INF]

    for i in range(n):
        for j in range(i + 1, n):
            tmp = circle_from_2(P[i], P[j])
            if (tmp[1] < mec[1] and is_valid_circle(tmp, P)):
                mec = tmp

    for i in range(n):
        for j in range(i + 1, n):
```

```
            for k in range(j + 1, n):
                tmp = circle_from_3(P[i], P[j], P[k])

                if (tmp[1] < mec[1] and is_valid_circle(tmp, P)):
                    mec = tmp
    return mec

def convert_to_points(z):
    points = []
    for point in z:
        points.append([point[0].real().n(digits=2), point[0].imag().
 ↪n(digits=2)])
    return points
```

[7]:
```
root = eq1.roots(ring=CIF)
points = convert_to_points(root)
mec = minimum_enclosing_circle(points)
```

[8]:
```
p = Graphics()
for i in eq1.roots(ring=CIF):
    p+= i[0].plot(color='red')

p+= circle(mec[0], mec[1], color='green')

p.show(axes_labels=['Real','Imaginary'])
p.save("Enclosed.png")
```



0.3 Generalising the Result

Consider the monic polynomial $p(x) = a_0 + a_1x + a_2x^2 + \cdots + x^n$, with distinct roots, where $a_i \in \mathbb{C}$.

Construct a matrix $A \in \mathbb{C}^{n \times n} \equiv \begin{bmatrix} 0 & 0 & 0 & \dots & 0 & -a_0 \\ 1 & 0 & 0 & \dots & 0 & -a_1 \\ 0 & 1 & 0 & \dots & 0 & -a_2 \\ \vdots & \vdots & \vdots & \ddots & 0 & \vdots \\ 0 & 0 & 0 & \dots & 1 & -a_{n-1} \end{bmatrix}$ where n is the degree of polynomial.The *eigenvalues* of the matrix A are the roots of the polynomial $p(x) = 0$ and hence the matrix A is called as the companion matrix of the polynomial $p(x)$.

Gershgorin Circle Theorem states that for a matrix $A \in \mathbb{C}^{n \times n}$, the inequality $|\lambda - a_{ii}| \leq R_i$ holds where λ is an *eigenvalue* of A and $R_i = \sum_{i \neq j} |a_{ij}|$ is the sum of non-diagonal entries in the i-th row.

To find the circle inside which all *eigenvalues* lie, simply extend $R = \sum |a_{ij}|$ such that $|\lambda - a_{ii}| \leq R, \forall\, i \in \mathbb{N}\; and\; 1 \leq i \leq n$.

[9]:
```
# Construct coefficients array in increasing order of exponent
coefficients = [1,-5,0,0,0,0,1]
```

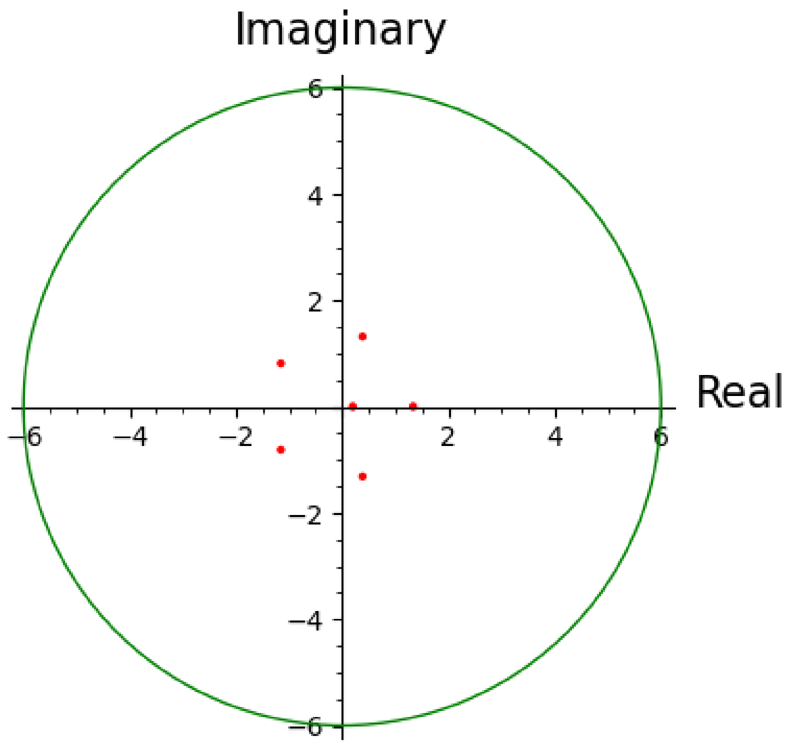
[10]:
```
def getMatrix(coefficients):
    # Returns the Companion Matrix
```

```
    degree = len(coefficients)-1

    M = matrix(CC, degree-1, degree-1, identity_matrix(degree-1))
    row = matrix(CC, 1, degree-1, [0 for t in range(degree-1)])
    column = matrix(CC, degree, 1, [-coefficients[e] for e in range(degree)])

    M = row.stack(M)
    M = M.augment(column)

    return M

def getCircle(coefficients):
    # Returns the Eigenvalues and enclosing circle
    M = getMatrix(coefficients)
    eigen = M.eigenvalues()
    max_radius = -1

    for i in range(M.nrows()):
        radius = 0
        for e in range(len(M[i])):
            radius += M[i][e].abs() # To ensure that all points lie inside the
 ↪circle

        if radius > max_radius:
            max_radius = max(max_radius, radius)
            center = M[i][i]

    return eigen, (center.real(), center.imag()), max_radius
```

[11]:
```
eigen, center, radius = getCircle(coefficients)
```

```
<ipython-input-10-7ddcc42ef7e0>:17: UserWarning: Using generic algorithm for an
inexact ring, which will probably give incorrect results due to numerical
precision issues.
  eigen = M.eigenvalues()
```

[12]:
```
p = Graphics()
for i in eigen:
    p+= i.plot(color='red')

p+= circle(center, radius, color='green')

p.show(axes_labels=['Real','Imaginary'])
p.save("General.png")
```


Imaginary
Real
6
4
2
−2
−4
−6
−6
−4
−2
2
4
6

```
[ ]:
```