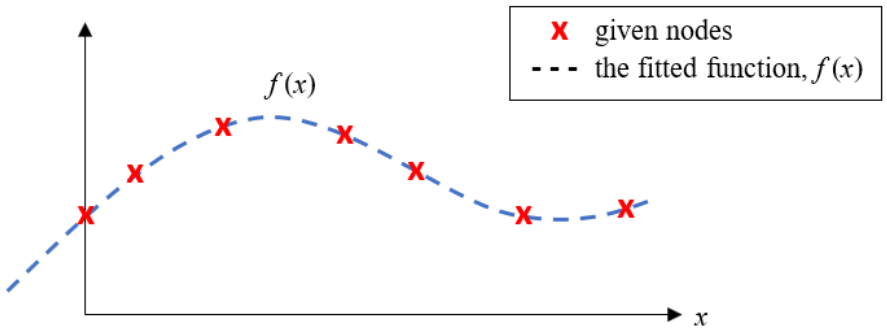
Point Interpolation Method (PIM)

INTRODUCTION

The point interpolation method (PIM) uses a polynomial function to fit a curve. It requires only a set of nodes along the curve. The PIM has the ability to fit the curve exactly, meaning that the fitted function touches all the given nodes, as illustrated in `Figure 1`.



`Figure 1`

To establish the PIM formulation, we begin with a simple statement of the polynomial representation of a function, *f*(*x*), in a 1D system, which is given by:

$$f(x) = a_1 + a_2 x + a_3 x^2 + \ldots + a_n x^{n-1} \quad \ldots(1)$$

where *x* is the spatial variable and *n* is the number of polynomial-terms required. Equation (1) can be written in a matrix form as:

$$f(x) = \begin{Bmatrix} 1 & x & x^2 & \cdots & x^{n-1} \end{Bmatrix} \begin{Bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_n \end{Bmatrix} = \mathbf{p}^T \mathbf{a} \quad \ldots(2)$$

Alternatively, it can also be written in a typical meshfree form as:

$$f(x) = \sum_{i=1}^{n} p_i(x)\, a_i \quad \ldots(3)$$

Where *i* is the running index, *pi*(*x*) is a set of monomial basis functions (1, *x*, *x*^2, ...) and *ai* is a set of constants that are unknowns. To fit a given set of nodes on a curve, these unknowns must be solved for.

EXAMPLE OF CURVE FITTING

Let say we have a set of known nodes along a curve, *uc*, given as:

$$u_c{}^T = \begin{Bmatrix} u_1 & u_2 & u_3 & \cdots & u_k \end{Bmatrix} \quad \ldots(4)$$

Where *k* is the number of available curve nodes. For the sake of simplicity in this example, let us consider only five nodes, *u1-u5*, located at specific *xc* coordinates, as shown in the code:

```
clear, clc
n = 5 % number of given nodes
```

```
n = 5
```

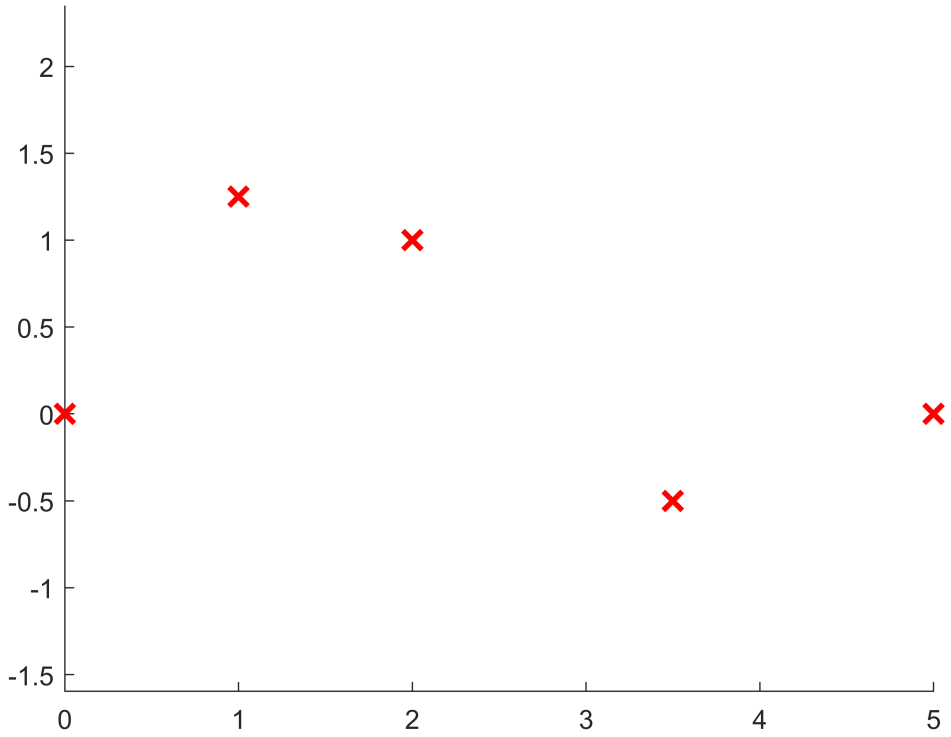
```
uc = [0 1.25 1 -0.5 0]' % nodal values of a curve
```

```
uc = 5×1
         0
    1.2500
    1.0000
   -0.5000
         0
```

```
xc = [0 1 2 3.5 5]' % location (x-axis)
```

```
xc = 5×1
         0
    1.0000
    2.0000
    3.5000
    5.0000
```

```
plot(xc,uc,'rx','MarkerSize',10,'LineWidth',2)
axis equal
box off
```



We will fit these nodes using the PIM approach. First, we link each nodal value that we have to the fitted function given in equation (2). As an example, we use a general form where, when $x = x1$ and $f(x) = u1$, which yields:

$$u_1 = \begin{Bmatrix} 1 & x_1 & {x_1}^2 & {x_1}^3 & {x_1}^4 \end{Bmatrix} \begin{Bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{Bmatrix} \quad ...(5)$$

This will give us one equation. By repeating this process for all nodes, we obtain five equations that can be put in matrix form, as shown below:

$$\begin{bmatrix} 1 & x_1 & {x_1}^2 & {x_1}^3 & {x_1}^4 \\ 1 & x_2 & {x_2}^2 & {x_2}^3 & {x_2}^4 \\ 1 & x_3 & {x_3}^2 & {x_3}^3 & {x_3}^4 \\ 1 & x_4 & {x_4}^2 & {x_4}^3 & {x_4}^4 \\ 1 & x_5 & {x_5}^2 & {x_5}^3 & {x_5}^4 \end{bmatrix} \begin{Bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{Bmatrix} = \begin{Bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{Bmatrix} \quad ...(6)$$

It is important to note that in the PIM, the number of unknowns must be equal to the number of given nodes, that is, $n = k$. Equation (6) can be simplified as:

$$\mathbf{P_c}\,\mathbf{a} = \mathbf{u_c} \quad ...(7)$$

Where **Pc** is a matrix of monomials evaluated at the five nodal locations, and **uc** is a vector of nodal values.

We can now solve for **a** using:

$$\mathbf{a} = \mathbf{P_c}^{-1}\mathbf{u_c} \quad ...(8)$$

Assuming that a is now solved, we can insert it back into equation (2) to obtain the general fitted function:

$$f(x) = \mathbf{p}^T(\mathbf{P_c}^{-1}\mathbf{u_c}) \quad ...(9)$$

Alternatively, it can be rearranged to yield:

$$f(x) = (\mathbf{p}^T\mathbf{P_c}^{-1})\mathbf{u_c} = \mathbf{\Phi}\,\mathbf{u_c} \quad ...(10)$$

Where **Φ** can be called a matrix of shape functions. A typical meshfree form for equation (10) is:

$$f(x) = \sum_{i=1}^{n}\phi_i(x)\,u_i \quad ...(11)$$

in which:

$$\phi_i(x) = \begin{Bmatrix}1 & x & x^2 & x^3 & x^4\end{Bmatrix}\begin{bmatrix}1 & x_1 & x_1^2 & x_1^3 & x_1^4\\ 1 & x_2 & x_2^2 & x_2^3 & x_2^4\\ 1 & x_3 & x_3^2 & x_3^3 & x_3^4\\ 1 & x_4 & x_4^2 & x_4^3 & x_4^4\\ 1 & x_5 & x_5^2 & x_5^3 & x_5^4\end{bmatrix}^{-1} \quad ...(12)$$

It is important to note that *x* is a spatial variable, while *x*1-*x*5 are the locations of the nodal values. We can now say that *f*(*x*) is a continuous interpolated function along *x* based on given nodal values (*u*1-*u*5) at specified locations (*x*1-*x*5).

The implementation codes for these steps are given below:

1) First, we calculate the **Pc** matrix:

```
Pc = [];
for i = 1:n
    Pci = [1 xc(i) xc(i)^2 xc(i)^3 xc(i)^4]
    Pc = cat(1,Pc,Pci) % generate the Pc matrix
end
```

```
Pci = 1×5
     1     0     0     0     0
Pc = 1×5
     1     0     0     0     0
Pci = 1×5
     1     1     1     1     1
Pc = 2×5
     1     0     0     0     0
     1     1     1     1     1
Pci = 1×5
     1     2     4     8    16
```

```
Pc = 3×5
     1     0     0     0     0
     1     1     1     1     1
     1     2     4     8    16
Pci = 1×5
    1.0000    3.5000   12.2500   42.8750  150.0625
Pc = 4×5
    1.0000         0         0         0         0
    1.0000    1.0000    1.0000    1.0000    1.0000
    1.0000    2.0000    4.0000    8.0000   16.0000
    1.0000    3.5000   12.2500   42.8750  150.0625
Pci = 1×5
     1     5    25   125   625
Pc = 5×5
    1.0000         0         0         0         0
    1.0000    1.0000    1.0000    1.0000    1.0000
    1.0000    2.0000    4.0000    8.0000   16.0000
    1.0000    3.5000   12.2500   42.8750  150.0625
    1.0000    5.0000   25.0000  125.0000  625.0000
```

2) To produce a continuous fitted curve, we create a spatial variable, x, ranging from 0 to 5 with a step of 0.1.

```
x = 0:0.1:5
```

```
x = 1×51
         0    0.1000    0.2000    0.3000    0.4000    0.5000    0.6000    0.7000 ···
```

3) We can now obtain the shape function matrix given in equation (12):

```
px = [];
for i = 1:length(x)
    pxi = [1 x(i) x(i)^2 x(i)^3 x(i)^4];
    px = cat(1,px,pxi); % generate the Pc matrix
end
phi = px/Pc
```

```
phi = 51×5
    1.0000         0         0         0         0
    0.8140    0.3165   -0.1666    0.0426   -0.0065
    0.6517    0.5702   -0.2816    0.0702   -0.0106
    0.5114    0.7670   -0.3509    0.0852   -0.0127
    0.3911    0.9126   -0.3803    0.0897   -0.0132
    0.2893    1.0125   -0.3750    0.0857   -0.0125
    0.2042    1.0718   -0.3403    0.0751   -0.0108
    0.1342    1.0956   -0.2809    0.0596   -0.0085
    0.0778    1.0886   -0.2016    0.0410   -0.0058
    0.0335    1.0553   -0.1066    0.0206   -0.0029
      ⋮
```

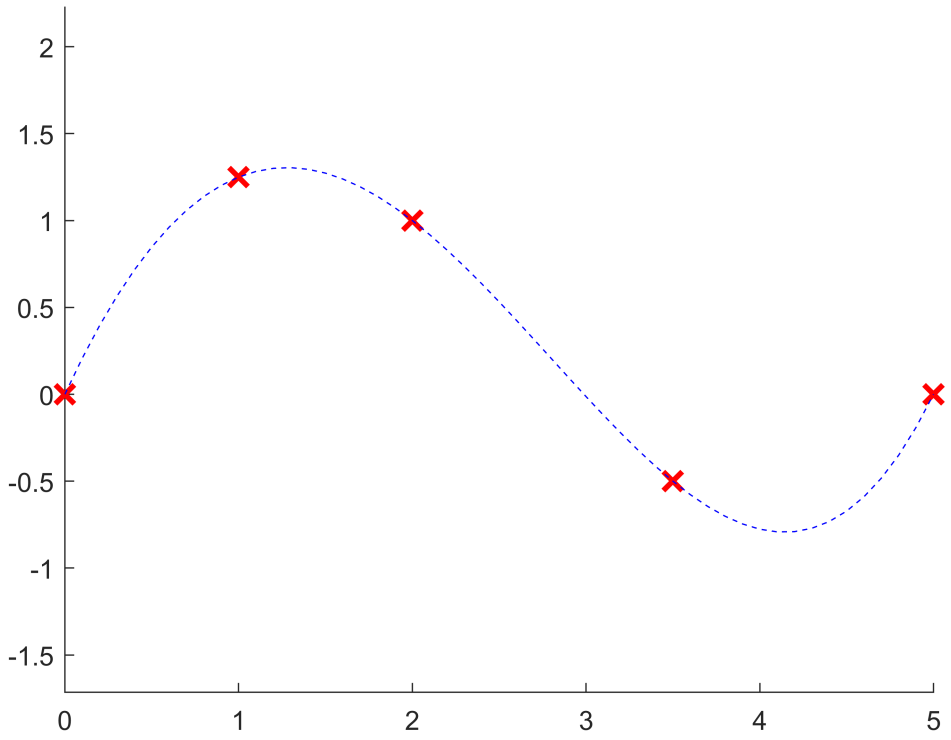
4) By multiplying the shape function matrix with the nodal values (*uc*), we obtain the fitted curve:

```
fx = phi*uc
```

```
fx = 51×1
         0
    0.2078
    0.3961
    0.5653
    0.7157
```

```
0.8478
0.9620
1.0588
1.1387
1.2023
  ⋮
```

```
cla
plot(xc,uc,'rx','MarkerSize',10,'LineWidth',2)
hold on
plot(x,fx,'b--')
hold off
axis equal
box off
```




```

```

GENERALISED CODES

A more generalized code for the PIM is given below, allowing us to try any number of given curve nodes and observe the performance of the PIM in fitting the curve.

```
clear, clc
```

```
uc = [0 -0.8 4.8 2 -1 0 0 -4.7 2 1 0]' % nodal values of a curve
```

```
uc = 11×1
         0
   -0.8000
    4.8000
    2.0000
   -1.0000
         0
         0
   -4.7000
    2.0000
    1.0000
      ⋮
```

```
n = length(uc) % number of given nodes
```

```
n = 11
```

```
xc = [0:(n-1)]' % x coordinates (assumed uniform)
```

```
xc = 11×1
     0
     1
     2
     3
     4
     5
     6
     7
     8
     9
     ⋮
```

```
Pc = [];
for i = 1:n
    Pci = xc(i).^(0:(n-1))
    Pc = cat(1,Pc,Pci) % generate the Pc matrix
end
```

```
Pci = 1×11
     1     0     0     0     0     0     0     0     0     0     0
Pc = 1×11
     1     0     0     0     0     0     0     0     0     0     0
Pci = 1×11
     1     1     1     1     1     1     1     1     1     1     1
Pc = 2×11
     1     0     0     0     0     0     0     0     0     0     0
     1     1     1     1     1     1     1     1     1     1     1
Pci = 1×11
           1           2           4           8          16          32 ···
Pc = 3×11
           1           0           0           0           0           0 ···
           1           1           1           1           1           1
           1           2           4           8          16          32
Pci = 1×11
           1           3           9          27          81         243 ···
```

```
Pc = 4×11
           1           0           0           0           0           0 ···
           1           1           1           1           1           1
           1           2           4           8          16          32
           1           3           9          27          81         243
Pci = 1×11
           1           4          16          64         256        1024 ···
Pc = 5×11
           1           0           0           0           0           0 ···
           1           1           1           1           1           1
           1           2           4           8          16          32
           1           3           9          27          81         243
           1           4          16          64         256        1024
Pci = 1×11
           1           5          25         125         625        3125 ···
Pc = 6×11
           1           0           0           0           0           0 ···
           1           1           1           1           1           1
           1           2           4           8          16          32
           1           3           9          27          81         243
           1           4          16          64         256        1024
           1           5          25         125         625        3125
Pci = 1×11
           1           6          36         216        1296        7776 ···
Pc = 7×11
           1           0           0           0           0           0 ···
           1           1           1           1           1           1
           1           2           4           8          16          32
           1           3           9          27          81         243
           1           4          16          64         256        1024
           1           5          25         125         625        3125
           1           6          36         216        1296        7776
Pci = 1×11
           1           7          49         343        2401       16807 ···
Pc = 8×11
           1           0           0           0           0           0 ···
           1           1           1           1           1           1
           1           2           4           8          16          32
           1           3           9          27          81         243
           1           4          16          64         256        1024
           1           5          25         125         625        3125
           1           6          36         216        1296        7776
           1           7          49         343        2401       16807
Pci = 1×11
10^9 ×
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0003    0.0021 ···
Pc = 9×11
10^9 ×
    0.0000         0         0         0         0         0         0         0 ···
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0001
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0003
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0001    0.0008
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0003    0.0021
Pci = 1×11
10^9 ×
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0001    0.0005    0.0048 ···
Pc = 10×11
10^9 ×
    0.0000         0         0         0         0         0         0         0 ···
```

```
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0001
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0003
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0001    0.0008
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0003    0.0021
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0001    0.0005    0.0048
Pci = 1×11
10^10 ×
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0001    0.0010 ···
Pc = 11×11
10^10 ×
    0.0000         0         0         0         0         0         0         0 ···
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0001
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0002
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0001    0.0005
     ⋮
```

```
dxc = 0.1; % step (can be changed if necessary)
x = min(xc):dxc:max(xc) % interpolation locations
```

```
x = 1×101
         0    0.1000    0.2000    0.3000    0.4000    0.5000    0.6000    0.7000 ···
```

```
px = [];
for i = 1:length(x)
    pxi = x(i).^(0:(n-1));
    px = cat(1,px,pxi); % generate the Pc matrix
end
phi = px/Pc
```

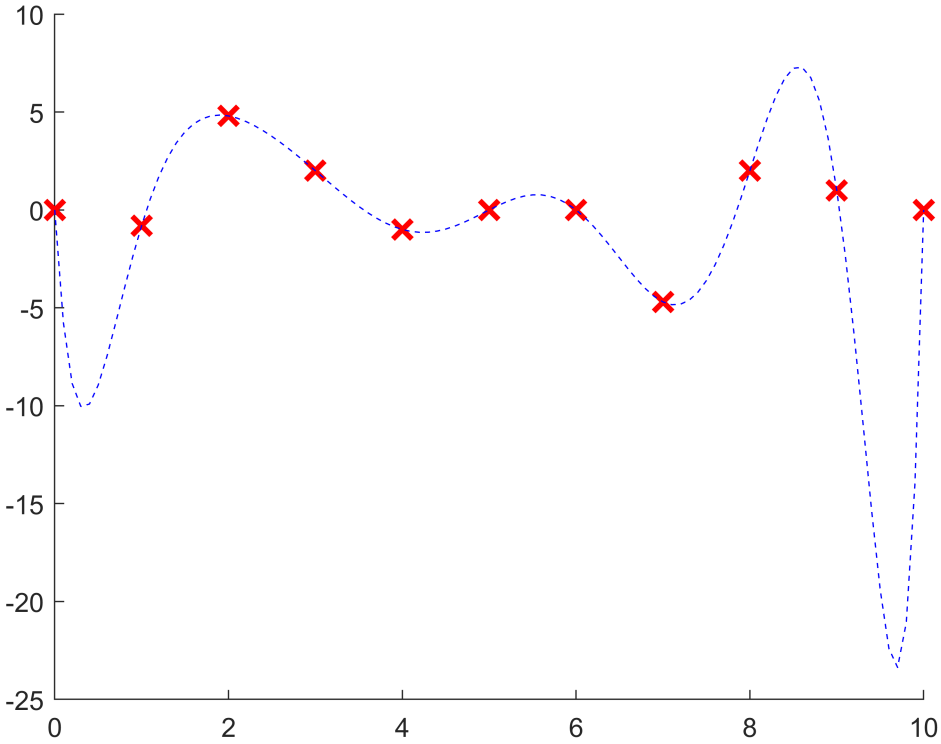
```
phi = 101×11
    1.0000         0         0         0         0         0         0         0 ···
    0.7400    0.8222   -1.7527    3.0622   -3.9847    3.8058   -2.6340    1.2870
    0.5377    1.3442   -2.6884    4.6087   -5.9428    5.6456   -3.8935    1.8977
    0.3821    1.6376   -3.0343    5.0947   -6.5061    6.1461   -4.2232    2.0531
    0.2642    1.7610   -2.9717    4.8767   -6.1636    5.7884   -3.9623    1.9211
    0.1762    1.7620   -2.6430    4.2287   -5.2859    4.9335   -3.3638    1.6264
    0.1119    1.6783   -2.1578    3.3566   -4.1464    3.8449   -2.6107    1.2587
    0.0660    1.5399   -1.5991    2.4103   -2.9398    2.7073   -1.8304    0.8799
    0.0342    1.3698   -1.0274    1.4943   -1.7979    1.6438   -1.1064    0.5302
    0.0132    1.1856   -0.4850    0.6775   -0.8031    0.7287   -0.4882    0.2332
     ⋮
```

```
fx = phi*uc
```

```
fx = 101×1
```

```
         0
   -5.7704
   -8.8573
  -10.0370
   -9.9138
   -8.9502
   -7.4932
   -5.7969
   -4.0418
   -2.3502
      ⋮
```

```
cla
plot(xc,uc,'rx','MarkerSize',10,'LineWidth',2)
hold on
plot(x,fx,'b--')
hold off
box off
```



REMARKS

Since the PIM uses polynomial interpolation, the higher the number of nodes involved, the higher the polynomial terms required. Therefore, the PIM may have instability issues when many nodes are required to be fitted,

especially when the nodes are distributed arbitrarily. Nonetheless, it is worth noting that when the distribution of nodes follows a polynomial curve, the PIM can accurately fit these nodes without difficulty.