**بسمه تعالی**

**Serendipity elements  
Theory and practice**

**(گزارش درس آنالیز عددی پیشرفته)**

Shape

Description automatically generated with low confidence

تابستان1403

رعنا حسنی

40120058

**مقدمه**

امروزه روش عناصر محدود (FEM)را میتوان به عنوان سنگ بنای حل عددی معادلات دیفرانسیل جزئی (PDEs)، به ویژه برای مسائل مربوط به هندسه های پیچیده و شرایط مرزی دانست. در این زمینه گسترده، عناصر سرندیپیتی به عنوان یک توسعه حیاتی ظاهر می شوند و شکاف بین کارایی محاسباتی و دقت راه حل را پر می کنند. این عناصر به دلیل توانایی آنها در کاهش هزینه های محاسباتی بدون به خطر انداختن یکپارچگی راه حل، توجه قابل توجهی را به خود جلب کرده اند.

مروری کوتاه بر عناصر Serendipity

عناصر Serendipity برای افزایش کارایی محاسباتی تحلیل عناصر محدود با کاهش استراتژیک تعداد گره‌ها در یک عنصر طراحی شده‌اند. برخلاف عناصر سنتی لاگرانژ که گره ها را به طور یکنواخت در سراسر عنصر توزیع می کنند، عناصر سرندیپیتی گره ها را عمدتاً در امتداد مرزهای عنصر قرار می دهند. این قرارگیری انتخابی گره، درجه تقریب چند جمله‌ای را حفظ می‌کند در حالی که تعداد کل درجات آزادی را به حداقل می‌رساند، که منجر به صرفه‌جویی محاسباتی قابل‌توجهی، به ویژه در مسائل با ابعاد بالاتر می‌شود.

زمینه تاریخی و توسعه

مفهوم عناصر سرندیپیتی به طور رسمی توسط ریچارد اس. گالاگر در سال 1975 معرفی شد که منعکس کننده یک کشف غیرمنتظره و در عین حال سودمند در تحلیل اجزای محدود است. گالاگر و معاصرانش تشخیص دادند که پیکربندی‌های گره خاص می‌توانند عناصری را به دست آورند که از نظر محاسباتی کارآمدتر هستند و در عین حال دقت کافی را برای کاربردهای مهندسی عملی حفظ می‌کنند. این کشف بی‌نظیر بر پتانسیل بهینه‌سازی مدل‌های عناصر محدود تاکید کرد و راه را برای پذیرش گسترده آنها در رشته‌های مختلف علمی و مهندسی هموار کرد.

پایه ریاضی

از نظر ریاضی، عناصر سرندیپیتی با استفاده از توابع شکل چند جمله ای که شرایط درون یابی در گره های عنصر را برآورده می کنند، مشخص می شوند. به عنوان مثال، در یک عنصر سرندیپیتی چهار ضلعی دو بعدی، توابع شکل معمولاً چند جمله ای درجه دوم هستند. این توابع برای درون یابی راه حل بر اساس مقادیر گرهی طراحی شده اند و به طور موثر رفتار متغیر میدان را در عنصر ثبت می کنند. درون یابی درون یک عنصر به صورت زیر بدست می آید:

  
که در آن 𝑢𝑖 مقادیر گره را در گره 𝑖 نشان می دهد. این فرمول تضمین می کند که راه حل پیوسته و قابل تمایز در سراسر عنصر است و شرایط لازم برای راه حل های دقیق PDE را برآورده می کند.

کاربردها و مزایا

عناصر Serendipity کاربردهای گسترده ای در حوزه های مختلف از جمله مکانیک ساختاری، انتقال حرارت و دینامیک سیالات پیدا می کنند. مزیت اصلی آنها در کاهش هزینه محاسباتی مرتبط با گره های کمتر است که به ماتریس های سیستم کوچکتر و زمان های حل سریع تر تبدیل می شود. این کارایی آنها را به ویژه برای شبیه سازی در مقیاس بزرگ که منابع محاسباتی یک نگرانی حیاتی هستند، مناسب می کند. علاوه بر این، دقت به‌دست‌آمده توسط عناصر سرندیپیتی اغلب برای بسیاری از کاربردهای عملی کافی است و یک رویکرد متعادل برای مدل‌سازی اجزای محدود ارائه می‌دهد.

**انگیزه مطالعه**

معادله پواسون، Δu=f ، یک معادله دیفرانسیل جزئی بنیادی (PDE) است که در کاربردهای علمی و مهندسی متعددی از جمله الکترواستاتیک، دینامیک سیالات و انتقال حرارت به وجود می‌آید. حل این معادله به صورت تحلیلی فقط برای هندسه های ساده و شرایط مرزی امکان پذیر است. در بیشتر سناریوهای دنیای واقعی، هندسه و شرایط مرزی پیچیده هستند و به روش‌های عددی برای حل دقیق نیاز دارند. روش اجزای محدود (FEM) به ویژه برای حل معادله پواسون در این سناریوهای پیچیده مناسب است. در اینجا چندین انگیزه برای اعمال FEM در معادله پواسون وجود دارد:

1. انعطاف پذیری در رسیدگی به هندسه های پیچیده

یکی از انگیزه های اولیه برای استفاده از FEM توانایی آن در مدیریت هندسه ها و حوزه های پیچیده است. معادله پواسون اغلب نیاز به حل در حوزه هایی دارد که نامنظم هستند یا دارای مرزهای پیچیده هستند، مانند دامنه های L شکل، مناطق دارای سوراخ، یا دامنه هایی با گوشه های ورودی مجدد. FEM دامنه را به زیر دامنه‌های کوچک‌تری به نام عناصر محدود گسسته می‌کند، که می‌توانند به راحتی برای هر شکل هندسی تطبیق داده شوند. این انعطاف‌پذیری برای مدل‌سازی دقیق مشکلات دنیای واقعی بسیار مهم است.

2. تطبیق پذیری در شرایط مرزی

معادله پواسون می تواند انواع مختلفی از شرایط مرزی مانند شرایط مرزی دیریکله، نویمان یا رابین را شامل شود FEM. به طور طبیعی می تواند این شرایط مرزی مختلف را در فرمولاسیون بگنجاند. این روش روشی سیستماتیک برای اعمال شرایط مرزی مستقیماً در گره‌های عناصر محدود ارائه می‌دهد و اطمینان حاصل می‌کند که راه‌حل به محدودیت‌های مشخص شده در مرزها احترام می‌گذارد.

3. تقریب محلی و کنترل خطا

FEM حل را به صورت محلی در هر عنصر محدود با استفاده از توابع شکل، که معمولاً توابع چند جمله ای هستند، تقریب می زند. این تقریب محلی در مناطقی که محلول دارای تغییرات قابل توجهی است، دقت بالایی را امکان پذیر می کند، در حالی که تقریب های درشت تر را می توان در مناطقی با راه حل های صاف تر استفاده کرد. این سازگاری برای کنترل خطای تقریب و دستیابی به راه حل های دقیق بدون هزینه محاسباتی بیش از حد مفید است.

4. مدیریت راه حل های ناپیوسته

در بسیاری از کاربردها، راه‌حل معادله پواسون ممکن است ناپیوستگی یا گرادیان‌های شدید را نشان دهد، مانند مشکلات مربوط به رابط‌های مواد یا تغییرات فاز. FEM به خوبی برای رسیدگی به چنین ناپیوستگی ها از طریق استفاده از عناصر با خواص مختلف یا تکنیک های پالایش مش مجهز است. این قابلیت تضمین می کند که راه حل حتی در صورت وجود ناپیوستگی دقیق و پایدار باقی می ماند.

5. استحکام و پایداری

FEM یک روش عددی قوی و پایدار برای حل معادله پواسون است. این روش یک چارچوب سیستماتیک برای اطمینان از پایداری حل عددی از طریق فرمول‌بندی اصول تغییرات فراهم می‌کند. با حل شکل ضعیف معادله پواسون، FEM ذاتاً تضمین می‌کند که محلول پایدار است و با پالایش مش به جواب واقعی همگرا می‌شود.

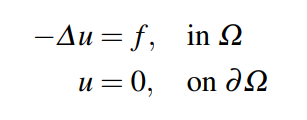
6. الگوریتم های محاسباتی کارآمد

پیشرفت در الگوریتم های محاسباتی و سخت افزار، FEM را به روشی بسیار کارآمد برای حل مسائل در مقیاس بزرگ تبدیل کرده است. بسته های نرم افزاری مدرن FEM، مانند FEniCS، deal.II، و COMSOL Multiphysics، ابزارهای قدرتمندی را برای تولید مش، مونتاژ ماتریس های سیستم و حل سیستم های خطی حاصل ارائه می دهند. این بسته‌ها از محاسبات موازی، حل‌کننده‌های کارآمد و پیش‌شرطی‌کننده‌ها برای رسیدگی کارآمد به مسائل بزرگ و پیچیده بهره می‌برند.

**چارچوب نظری برای حل معادله پواسون با عناصر Serendipity**

1 - معادله پواسون:

در دو بعد معادله پواسون به شکل پیدا کردن جواب می باشد که



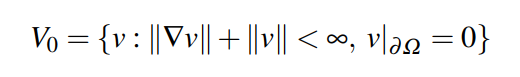
که در آن عملگر لاپلاس است و یک تابع داده شده است.  
-2 فرمول بندی Variational

برای به دست آوردن یک فرمول متغیر از معادله پواسون، را در یک تابع آزمایشی v ضرب می‌کنیم، که فرض می‌شود در مرز ناپدید می‌شود و با استفاده از فرمول گرین (یعنی انتگرال جز به جز) ادغام می‌شود.

**A group of math equations

Description automatically generated**

از آنجایی که v = 0 در . به علاوه، معرفی فضای V0، تعریف شده به صورت :



ما فرمول Variational زیر را داریم u ∈ V0 را طوری پیدا کنید که

A math equation with a number and a symbol

Description automatically generated with medium confidence

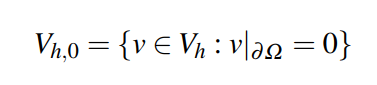
با این انتخاب فضای جواب ها و فضای آزمایشی V0، انتگرال های دو سمت معادله ی بالا

به خوبی تعریف شده اند. برای مشاهده این موضوع، توجه داشته باشید که به دلیل نابرابری کوشی-شوارتز دارای است که با فرض کمتر از بی نهایت است. خط استدلال مشابهی در مورد انتگرال دیگر صدق می کند. در واقع، V0 بزرگترین فضا با این خاصیت است که انتگرال ها در فرمول بندی Variational وجود دارند. در این زمینه ما می خواهیم به نکته ای ظریف اشاره کنیم که هنوز به آن اشاره نکرده ایم. حتی اگر حل معادله پواسون فرم قوی نیز راه‌حلی برای فرمول Variational است، برخلاف آن معمولاً درست نیست. برای درک این موضوع، توجه به این نکته کافی است که راه حل معادله Variational نباید دو بار متمایز باشد. به همین دلیل، فرمول Variational را بر خلاف شکل قوی اصلی، گاهی اوقات شکل ضعیف می نامند. اثبات اینکه یک راه حل ضعیف نیز یک راه حل قوی است می تواند مشکل باشد، زیرا به شکل دامنه و نظم ضرایب بستگی دارد.

.3 تقریب المان محدود

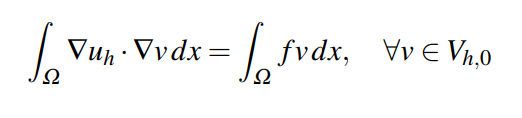
حال، اجازه دهید K یک مربع بندی روی Ω باشد، و اجازه دهید فضای خطی های پیوسته تکه ای روی K باشد. همچنین، برای ارضای شرایط مرزی قوی، اجازه دهید

فضای فرعی باشد.



با این انتخاب فضای تقریبی، روش اجزای محدود برای معادله پواسون شکل می گیرد:

را پیدا کنید به طوری که



4. استخراج یک سیستم خطی معادلات

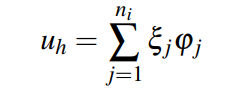
برای محاسبه تقریب المان محدود اجازه دهید پایه ای برای با توابع hat باشد. در اینجا، ni تعداد گره های داخلی در مش است، زیرا توابع در مرز ناپدید می شوند.

توجه می کنیم که روش اجزای محدود معادل است با

A black and white math symbol

Description automatically generated with medium confidence

از آنجایی که متعلق به است، می توان آن را به صورت زیر نوشت



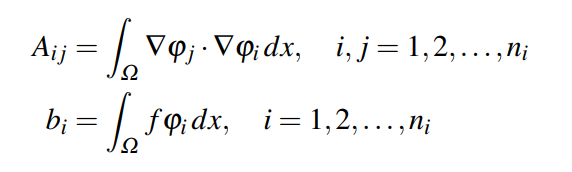
که به صورت N یعنی مجهولات ، تعیین می شود.

با ترکیب مباحث بالا داریم:

A group of math equations

Description automatically generated

با نوتیشن های زیر می توان نوشت :



که یک سیستم خطی برای مجهولات است. به صورت ماتریسی این را می نویسیم:

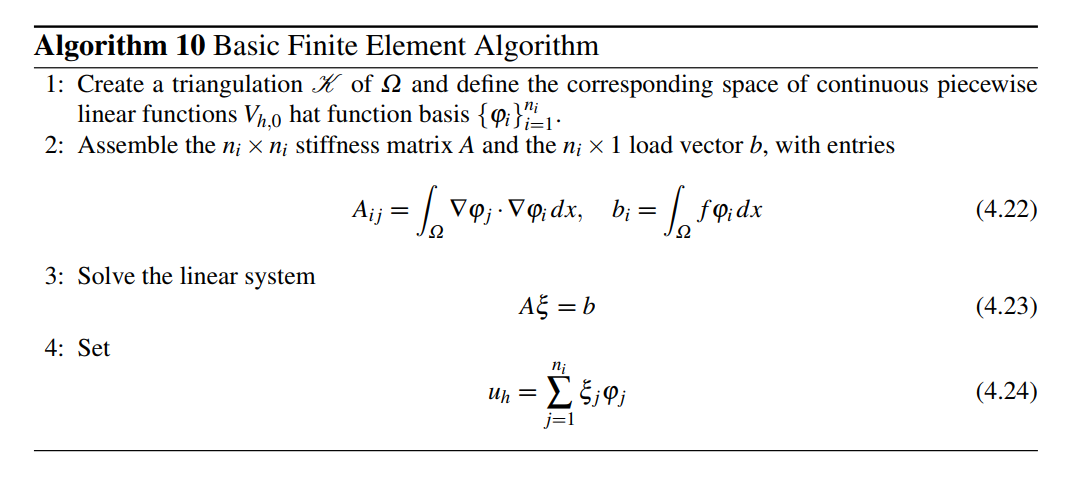
A math equation with a couple of letters

Description automatically generated with medium confidence

که در آن ورودی های ماتریس سختی A و بردار بار به ترتیب تعریف شده است. با حل سیستم خطی، مجهولات را بدست می آوریم، و بنابراین نیز بدست می آید.

5. الگوریتم اساسی برای محاسبه حل المان محدود

الگوریتم زیر مراحل اساسی برای محاسبه حل المان محدود uh را خلاصه می کند.



کدهایی که در ادامه توضیح می دهیم طبق همین اساس کلی نوشته شده اند.

5. تحلیل اساسی روش عناصر محدود

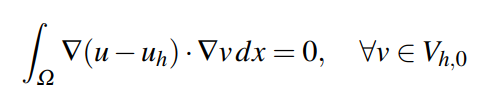
5.1 وجود و منحصر به فرد بودن حل عناصر محدود

قضیه 5.1. راه حل عناصر محدود تعریف شده توسط وجود دارد و منحصر به فرد است.

6. برآوردهای اولیه خطا :

در این بخش ما تخمین های خطای اساسی را برای تقریب عناصر محدود ارائه می کنیم.

قضیه 6.1 (Galerkin Orthogonality) تقریب عناصر محدود ، تعریف شده رابطه زیر را برآورده می کند



برای تخمین خطا، نرم زیر را به نام نرم انرژی در V0 معرفی می کنیم

A black symbols on a white background

Description automatically generated

A black and blue triangle and triangle symbols

Description automatically generated with medium confidence

قضیه 6.2. حل عناصر محدود تعریف شده توسط رابطه زیر برآورده می شود

A black and white math equation

Description automatically generated with medium confidence

این نشان می‌دهد که حل عناصر محدود نزدیک‌ترین توابع در به جواب دقیق هنگام اندازه‌گیری فاصله با استفاده از نرم انرژی است.

**توضیحات مربوط به کد :**

1. **تعریف توابع اصلی** **poisson\_serendipity:**

تابع poisson\_serendipity محرک اصلی برنامه است که کل فرآیند حل معادله پواسون را با استفاده از عناصر محدود سرندیپیتی هماهنگ می کند. در اینجا توضیح دقیق هر خط آورده شده است:

function poisson\_serendipity()

% Define geometry

L = 2; % Size of the domain

Nx = 100; % Number of elements in x direction

Ny = 100; % Number of elements in y direction

[nodes, elements] = generate\_mesh(L, Nx, Ny);

% Define serendipity shape functions and their gradients

[shape\_functions, shape\_gradients] = define\_serendipity\_elements();

% Assemble global stiffness matrix and force vector

[K, F] = assemble\_system(nodes, elements, shape\_functions, shape\_gradients);

% Apply boundary conditions

[K, F] = apply\_boundary\_conditions(K, F, nodes);

% Solve the linear system

U = K \ F;

% Plot the solution

plot\_solution(nodes, elements, U);

end

**توضیحات این بخش**

1- هندسه را تعریف می کنیم :

% Define geometry

L = 2; % Size of the domain

Nx = 100; % Number of elements in x direction

Ny = 100; % Number of elements in y direction

[nodes, elements] = generate\_mesh(L, Nx, Ny);

L = 2; : اندازه دامنه را به مربع 2\*2 تنظیم می کنیم.

Nx = 10; Ny = 10; : تعداد عناصر را در جهت x و y تعریف می کنیم.

[nodes, elements] = generate\_mesh(L, Nx, Ny); : تابع generate\_mesh را برای ایجاد یک شبکه ساختار یافته از گره ها و عناصر فراخوانی می کنیم. خروجی به این صورت است:

nodes : مختصات همه گره ها در مش.

elements : ماتریس اتصال که تعیین می کند کدام گره ها هر عنصر را تشکیل می دهند.

2- توابع شکل Serendipity را تعریف می کنیم:

% Define serendipity shape functions and their gradients

[shape\_functions, shape\_gradients] = define\_serendipity\_elements();

: define\_serendipity\_elements();تابع برای تعریف توابع شکل سرندیپیتی و گرادیان آنها با توجه به مختصات مرجع () فراخوانی می شود. خروجی به این صورت است:

: shape\_functions توابع شکل عناصر سرندیپیتی

: shape\_gradients گرادیان توابع شکل با توجه به و

3- ماتریس سختی جهانی و بردار نیرو را جمع آوری می کنیم:

% Assemble global stiffness matrix and force vector

[K, F] = assemble\_system(nodes, elements, shape\_functions, shape\_gradients);

assemble\_system : تابع برای ساختن ماتریس سختی جهانی K و بردار نیروی جهانی F فراخوانی می شود. این تابع مشارکت های هر عنصر را در سیستم جهانی جمع می کند.

4- اعمال شرایط مرزی:

% Apply boundary conditions

[K, F] = apply\_boundary\_conditions(K, F, nodes);

apply\_boundary\_conditions : تابع برای اصلاح ماتریس سختی K و بردار نیرو F برای ترکیب شرایط مرزی فراخوانی می شود. این مرحله تضمین می کند که راه حل محدودیت های داده شده در مرز دامنه را برآورده می کند.

5- حل سیستم خطی:

% Solve the linear system

U = K \ F;

سیستم خطی KU=F با استفاده از عملگر بک اسلش \ برای به دست آوردن مقادیر گرهی راه حل U حل می شود.

6- نمایش راه حل:

% Plot the solution

plot\_solution(nodes, elements, U);

plot\_solution : تابع برای رسم شکل راه حل محاسبه شده در دامنه فراخوانی می شود.

**2- تابع کمکی generate\_mesh :**

function [nodes, elements] = generate\_mesh(L, Nx, Ny)

% Generate a simple structured mesh for the domain with mid-side nodes

[x, y] = meshgrid(linspace(0, L, Nx+1), linspace(0, L, Ny+1));

nodes = [x(:), y(:)];

% Generate mid-side nodes for horizontal edges

mid\_x = (x(:, 1:end-1) + x(:, 2:end)) / 2;

mid\_y\_horizontal = y(:, 1:end-1);

mid\_nodes\_x = [mid\_x(:), mid\_y\_horizontal(:)];

% Generate mid-side nodes for vertical edges

mid\_y = (y(1:end-1, :) + y(2:end, :)) / 2;

mid\_x\_vertical = x(1:end-1, :);

mid\_nodes\_y = [mid\_x\_vertical(:), mid\_y(:)];

% Combine mid-side nodes

mid\_nodes = [mid\_nodes\_x; mid\_nodes\_y];

% Combine all nodes

nodes = [nodes; mid\_nodes];

% Add an index for the mid-side nodes to the elements array

num\_original\_nodes = (Nx + 1) \* (Ny + 1);

num\_mid\_x\_nodes = numel(mid\_x);

elements = [];

for i = 1:Nx

for j = 1:Ny

n1 = (j-1)\*(Nx+1) + i;

n2 = n1 + 1;

n3 = n2 + Nx + 1;

n4 = n1 + Nx + 1;

n5 = num\_original\_nodes + (j-1)\*Nx + i; % mid-side node on the bottom edge

n6 = num\_original\_nodes + num\_mid\_x\_nodes + (i-1)\*Ny + j; % mid-side node on the right edge

n7 = num\_original\_nodes + (j-1)\*Nx + i + Nx; % mid-side node on the top edge

n8 = num\_original\_nodes + num\_mid\_x\_nodes + (i-1)\*Ny + j - 1; % mid-side node on the left edge

elements = [elements; n1, n2, n3, n4, n5, n6, n7, n8];

end

end

end

**توضیحات این بخش**

1. تعریف تابع و ورودی‌های آن

function [nodes, elements] = generate\_mesh(L, Nx, Ny)

L: طول دامنه‌ی مستطیلی در هر جهت (x و y).

Nx: تعداد بخش‌های گره‌ای در جهت x.

Ny: تعداد بخش‌های گره‌ای در جهت y.

2. تولید نقاط گره‌ها

[x, y] = meshgrid(linspace(0, L, Nx+1), linspace(0, L, Ny+1));

nodes = [x(:), y(:)];

meshgrid: این تابع یک شبکه‌ی دوبعدی از نقاط را ایجاد می‌کند که در آن linspace(0, L, Nx+1) و linspace(0, L, Ny+1) به ترتیب برای ایجاد ابعاد x و y استفاده می‌شوند.

nodes = [x(:), y(:)]: این دو بردار x و y را به یک آرایه گره تبدیل می‌کند که هر ردیف آن نماینده‌ی یک نقطه با مختصات (x، y) است.

3. تولید گره‌های وسط‌کننده برای لبه‌ها

الف) برای لبه‌های افقی

mid\_x = (x(:, 1:end-1) + x(:, 2:end)) / 2;

mid\_y\_horizontal = y(:, 1:end-1);

mid\_nodes\_x = [mid\_x(:), mid\_y\_horizontal(:)];

mid\_x: مختصات x برای نقاط وسط‌کننده در لبه‌های افقی محاسبه می‌شود.

mid\_y\_horizontal: مختصات y برای نقاط وسط‌کننده در لبه‌های افقی، که در اینجا همان y از ماتریس اولیه است.

mid\_nodes\_x = [mid\_x(:), mid\_y\_horizontal(:)]: ترکیب مختصات x و y برای گره‌های وسط‌کننده در لبه‌های افقی به صورت یک آرایه گره.

ب) برای لبه‌های عمودی

mid\_y = (y(1:end-1, :) + y(2:end, :)) / 2;

mid\_x\_vertical = x(1:end-1, :);

mid\_nodes\_y = [mid\_x\_vertical(:), mid\_y(:)];

mid\_y: مختصات y برای نقاط وسط‌کننده در لبه‌های عمودی محاسبه می‌شود.

mid\_x\_vertical: مختصات x برای نقاط وسط‌کننده در لبه‌های عمودی.

mid\_nodes\_y = [mid\_x\_vertical(:), mid\_y(:)]: ترکیب مختصات x و y برای گره‌های وسط‌کننده در لبه‌های عمودی به صورت یک آرایه گره.

4. ادغام تمامی نقاط گره

mid\_nodes = [mid\_nodes\_x; mid\_nodes\_y];

nodes = [nodes; mid\_nodes];

mid\_nodes: ترکیب همه‌ی نقاط گره‌های وسط‌کننده در یک آرایه mid\_nodes.

nodes = [nodes; mid\_nodes]: ادغام تمامی نقاط گره اصلی و وسط‌کننده در یک آرایه nodes نهایی.

5. ایجاد شاخص برای عناصر

num\_original\_nodes = (Nx + 1) \* (Ny + 1);

num\_mid\_x\_nodes = numel(mid\_x);

elements = [];

for i = 1:Nx

for j = 1:Ny

n1 = (j-1)\*(Nx+1) + i;

n2 = n1 + 1;

n3 = n2 + Nx + 1;

n4 = n1 + Nx + 1;

n5 = num\_original\_nodes + (j-1)\*Nx + i; % mid-side node on the bottom edge

n6 = num\_original\_nodes + num\_mid\_x\_nodes + (i-1)\*Ny + j; % mid-side node on the right edge

n7 = num\_original\_nodes + (j-1)\*Nx + i + Nx; % mid-side node on the top edge

n8 = num\_original\_nodes + num\_mid\_x\_nodes + (i-1)\*Ny + j - 1; % mid-side node on the left edge

elements = [elements; n1, n2, n3, n4, n5, n6, n7, n8];

end

end

num\_original\_nodes : تعداد کل نقاط گره اصلی در دامنه محاسبه می‌شود.

num\_mid\_x\_nodes: تعداد نقاط گره وسط‌کننده در لبه های افقی محاسبه می‌شود.

حلقه‌ی for دوگانه برای ایجاد شاخص‌های n1 تا n8 برای هر عنصر، که نقاط گره اصلی و وسط‌کننده برای هر عنصر را نمایش می‌دهد.

elements = [elements; n1, n2, n3, n4, n5, n6, n7, n8];: افزودن شاخص‌های ایجاد شده به آرایه elements برای تشکیل عناصر در دامنه.

نقاط گره اصلی (Main Nodes):

n1: شاخص برای نقطه‌ی گره اصلی سمت چپ پایین هر عنصر.

n2: شاخص برای نقطه‌ی گره اصلی سمت راست پایین هر عنصر.

n3: شاخص برای نقطه‌ی گره اصلی سمت راست بالا هر عنصر.

n4: شاخص برای نقطه‌ی گره اصلی سمت چپ بالا هر عنصر.

نقاط گره وسط‌کننده (Mid-side Nodes):

n5 : شاخص برای نقطه‌ی گره وسط‌کننده در لبه‌ی پایین هر عنصر.

n6: شاخص برای نقطه‌ی گره وسط‌کننده در لبه‌ی راست هر عنصر.

n7: شاخص برای نقطه‌ی گره وسط‌کننده در لبه‌ی بالا هر عنصر.

n8: شاخص برای نقطه‌ی گره وسط‌کننده در لبه‌ی چپ هر عنصر.

توضیحات بیشتر:

محاسبه شاخص‌ها برای نقاط گره اصلی:

n1: این شاخص با استفاده از فرمول (j-1)\*(Nx+1) + i برای هر عنصر محاسبه می‌شود. این فرمول به ازای هر i و j در دو حلقه‌ی for دوگانه، شاخص را برای نقطه‌ی گره اصلی سمت چپ پایین هر عنصر مشخص می‌کند.

مانند n2, n3, و n4 که به ترتیب برای نقاط گره اصلی سمت راست پایین، سمت راست بالا و سمت چپ بالا هر عنصر محاسبه می‌شوند.

محاسبه شاخص‌ها برای نقاط گره وسط‌کننده:

n5: این شاخص برای نقطه‌ی گره وسط‌کننده در لبه‌ی پایین هر عنصر محاسبه می‌شود با استفاده از فرمول num\_original\_nodes + (j-1)\*Nx + i. این شاخص برای لبه‌ی پایین هر عنصر تعیین می‌کند که کدام نقطه‌ی گره وسط‌کننده است.

مانند n6, n7, و n8 که به ترتیب برای نقاط گره وسط‌کننده در لبه‌های راست، بالا، و چپ هر عنصر محاسبه می‌شوند.

استفاده از شاخص‌ها در تعریف عناصر:

پس از محاسبه‌ی شاخص‌ها برای هر عنصر، آن‌ها به آرایه elements اضافه می‌شوند. این آرایه elements شامل همه‌ی شاخص‌های لازم برای تعریف عناصر شبکه‌ی گره‌ها است که در ادامه برای تولید ماتریس سختی و بردار نیرو استفاده می‌شود.

**-3تابع کمکی define\_serendipity\_elements:**

function [shape\_functions, shape\_gradients] = define\_serendipity\_elements()

syms xi eta;

shape\_functions = [

0.25 \* (1 - xi) \* (1 - eta) \* (-xi - eta - 1); % Shape function 1

0.25 \* (1 + xi) \* (1 - eta) \* ( xi - eta - 1); % Shape function 2

0.25 \* (1 + xi) \* (1 + eta) \* ( xi + eta - 1); % Shape function 3

0.25 \* (1 - xi) \* (1 + eta) \* (-xi + eta - 1); % Shape function 4

0.5 \* (1 - xi^2) \* (1 - eta); % Shape function 5 (mid-side node on the bottom edge)

0.5 \* (1 + xi) \* (1 - eta^2); % Shape function 6 (mid-side node on the right edge)

0.5 \* (1 - xi^2) \* (1 + eta); % Shape function 7 (mid-side node on the top edge)

0.5 \* (1 - xi) \* (1 - eta^2); % Shape function 8 (mid-side node on the left edge)

];

shape\_gradients = [

diff(shape\_functions, xi)';

diff(shape\_functions, eta)'

];

end

1. تعریف تابع و خروجی‌های آن

function [shape\_functions, shape\_gradients] = define\_serendipity\_elements()

shape\_functions: آرایه‌ای از توابع شکل برای عناصر سرندیپیتی که در اینجا با استفاده از متغیرهای xi و eta تعریف می‌شوند.

shape\_gradients: آرایه‌ای از گرادیان توابع شکل متناظر با عناصر سرندیپیتی، که با استفاده از diff به دست می‌آید.

2. تعریف توابع شکل

syms xi eta;

shape\_functions = [

0.25 \* (1 - xi) \* (1 - eta) \* (-xi - eta - 1); % Shape function 1

0.25 \* (1 + xi) \* (1 - eta) \* ( xi - eta - 1); % Shape function 2

0.25 \* (1 + xi) \* (1 + eta) \* ( xi + eta - 1); % Shape function 3

0.25 \* (1 - xi) \* (1 + eta) \* (-xi + eta - 1); % Shape function 4

0.5 \* (1 - xi^2) \* (1 - eta); % Shape function 5 (mid-side node on the bottom edge)

0.5 \* (1 + xi) \* (1 - eta^2); % Shape function 6 (mid-side node on the right edge)

0.5 \* (1 - xi^2) \* (1 + eta); % Shape function 7 (mid-side node on the top edge)

0.5 \* (1 - xi) \* (1 - eta^2); % Shape function 8 (mid-side node on the left edge)

];

در این قسمت، توابع شکل برای هر کدام از نقاط گره در عناصر سرندیپیتی تعریف شده‌اند. هر تابع شکل با استفاده از متغیرهای xi و eta بیان می‌شود که نشان دهندهٔ مختصات محلی عنصر هستند.

3. محاسبهٔ گرادیان توابع شکل

shape\_gradients = [

diff(shape\_functions, xi)';

diff(shape\_functions, eta)'

];

diff(shape\_functions, xi): این دستور به تابع شکل مرتبط با xi (تابع شکل نسبت به xi) گرادیان را محاسبه می‌کند.

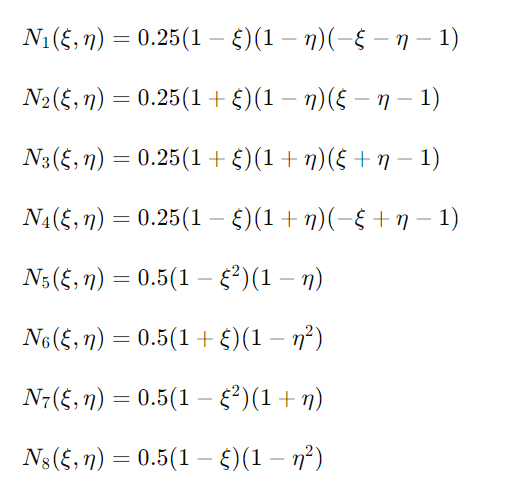
diff(shape\_functions, eta): این دستور به تابع شکل مرتبط با eta (تابع شکل نسبت به eta) گرادیان را محاسبه می‌کند.

shape\_gradients = [ ... ]: ترکیب این دو گرادیان به صورت یک آرایهٔ گرادیان، که هر ردیف آن نمایانگر گرادیان یک تابع شکل است.

**جزئیات ریاضی**

**تعریف توابع شکل (Shape Functions)**

توابع شکل در روش عناصر محدود به صورت توابعی از متغیرهای محلی (معمولاً و ) تعریف می‌شوند که به صورت خطی یا غیرخطی، توزیع میدانی متغیرهای فیزیکی را در داخل هر عنصر محدود نمایش می‌دهند. در مورد توابع شکل سرندیپیتی (Serendipity Shape Functions) که در کد ارائه شده است، می‌توانیم به صورت زیر تعریف و محاسبه کنیم:



هر تابع شکل متناظر با یک گره خاص در عناصر محدود است:

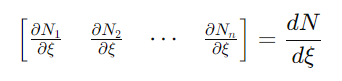
A screenshot of a computer

Description automatically generated

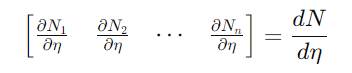
2. مشتقات توابع شکل (Shape Function Gradients) :

مشتقات توابع شکل نسبت به متغیرهای محلی 𝜉و𝜂به صورت زیر محاسبه می‌شوند:

مشتق با توجه به:



مشتق با توجه به 𝜂:



**4-تابع کمکی :assemble\_system**

function [K, F] = assemble\_system(nodes, elements, shape\_functions, shape\_gradients)

num\_nodes = size(nodes, 1);

num\_elements = size(elements, 1);

K = sparse(num\_nodes, num\_nodes);

F = zeros(num\_nodes, 1);

for e = 1:num\_elements

element\_nodes = elements(e, :);

coords = nodes(element\_nodes, :);

% Compute element stiffness matrix and force vector

[Ke, Fe] = element\_stiffness\_force(coords, shape\_functions, shape\_gradients);

% Assemble into global stiffness matrix and force vector

for i = 1:length(element\_nodes)

for j = 1:length(element\_nodes)

K(element\_nodes(i), element\_nodes(j)) = K(element\_nodes(i), element\_nodes(j)) + Ke(i, j);

end

F(element\_nodes(i)) = F(element\_nodes(i)) + Fe(i);

end

end

end

1. تعریف تابع و ورودی‌های آن

function [K, F] = assemble\_system(nodes, elements, shape\_functions, shape\_gradients)

این تابع assemble\_system نام دارد و چهار ورودی دریافت می‌کند: nodes( گره‌ها)، elements(المان‌ها)، shape\_functions (توابع شکل)، و shape\_gradients (گرادیان توابع شکل).

2. تعیین تعداد گره‌ها و المان‌ها

num\_nodes = size(nodes, 1);

num\_elements = size(elements, 1);

num\_nodes: تعداد گره‌ها را با استفاده از تابع size محاسبه می‌کند.

num\_elements: تعداد المان‌ها را با استفاده از تابع size محاسبه می‌کند.

3. مقداردهی اولیه به ماتریس سختی و بردار نیرو

K = sparse(num\_nodes, num\_nodes);

F = zeros(num\_nodes, 1);

K: ماتریس سختی را به صورت ماتریس اسپارس (پراکنده) و با ابعاد num\_nodes در num\_nodes مقداردهی اولیه می‌کند.

F: بردار نیرو را با اندازه num\_nodes و با مقادیر صفر مقداردهی اولیه می‌کند.

4. حلقه برای پردازش هر المان

for e = 1:num\_elements

element\_nodes = elements(e, :);

coords = nodes(element\_nodes, :);

یک حلقه for برای پردازش هر المان (از ۱ تا num\_elements) اجرا می‌شود.

element\_nodes: گره‌های مربوط به المان فعلی را از آرایه elements استخراج می‌کند.

coords: مختصات گره‌های المان فعلی را از آرایه nodes استخراج می‌کند.

5. محاسبه ماتریس سختی المان و بردار نیرو

[Ke, Fe] = element\_stiffness\_force(coords, shape\_functions, shape\_gradients);

Ke: ماتریس سختی المان را با استفاده از تابع element\_stiffness\_force محاسبه می‌کند.

Fe: بردار نیرو المان را با استفاده از تابع element\_stiffness\_force محاسبه می‌کند.

6. تجمیع در ماتریس سختی کلی و بردار نیرو

for i = 1:length(element\_nodes)

for j = 1:length(element\_nodes)

K(element\_nodes(i), element\_nodes(j)) = K(element\_nodes(i), element\_nodes(j)) + Ke(i, j);

end

F(element\_nodes(i)) = F(element\_nodes(i)) + Fe(i);

end

end

یک حلقه دوگانه for برای تجمیع مقادیر ماتریس سختی المان Ke در ماتریس سختی کلی K اجرا می‌شود.

بردار نیرو المان Fe نیز به بردار نیرو کلی F اضافه می‌شود.

**5-تابع کمکی : element\_stiffness\_force**

function [Ke, Fe] = element\_stiffness\_force(coords, shape\_functions, shape\_gradients)

% Numerical integration (Gauss quadrature)

gauss\_points = [-1/sqrt(3), 1/sqrt(3)];

gauss\_weights = [1, 1];

num\_nodes\_per\_element = size(coords, 1);

Ke = zeros(num\_nodes\_per\_element, num\_nodes\_per\_element);

Fe = zeros(num\_nodes\_per\_element, 1);

for i = 1:length(gauss\_points)

for j = 1:length(gauss\_points)

xi = gauss\_points(i);

eta = gauss\_points(j);

weight = gauss\_weights(i) \* gauss\_weights(j);

% Evaluate shape functions and gradients at Gauss points

N = double(subs(shape\_functions, {'xi', 'eta'}, {xi, eta}));

dN\_dxi = double(subs(shape\_gradients(1, :), {'xi', 'eta'}, {xi, eta}));

dN\_deta = double(subs(shape\_gradients(2, :), {'xi', 'eta'}, {xi, eta}));

% Jacobian matrix and determinant

J = [dN\_dxi; dN\_deta] \* coords;

detJ = det(J);

invJ = inv(J);

% Gradient of shape functions in physical coordinates

dN\_dx = invJ \* [dN\_dxi; dN\_deta];

% Element stiffness matrix

Ke = Ke + (dN\_dx' \* dN\_dx) \* detJ \* weight;

% Element force vector

Fe = Fe + N \* detJ \* weight;

end

end

end

1. تعریف تابع و ورودی‌های آن

function [Ke, Fe] = element\_stiffness\_force(coords, shape\_functions, shape\_gradients)

این تابع با استفاده از مختصات گره‌ها، توابع شکل و گرادیان توابع شکل، ماتریس سختی المان و بردار نیرو را محاسبه می‌کند.

2. تعریف نقاط و وزن‌های گاوس

gauss\_points = [-1/sqrt(3), 1/sqrt(3)];

gauss\_weights = [1, 1];

num\_nodes\_per\_element = size(coords, 1);

Ke = zeros(num\_nodes\_per\_element, num\_nodes\_per\_element);

Fe = zeros(num\_nodes\_per\_element, 1);

gauss\_points: نقاط انتگرال‌گیری گاوسی در بازه[1,1-].

gauss\_weights: وزن‌های متناظر با نقاط گاوس.

num\_nodes\_per\_element: تعداد گره‌های هر عنصر.

Ke: ماتریس سختی عناصر که به‌صورت صفر مقداردهی اولیه می‌شود.

Fe: بردار نیرو که به‌صورت صفر مقداردهی اولیه می‌شود.

3. حلقه برای محاسبه انتگرال‌ها در نقاط گاوس

for i = 1:length(gauss\_points)

for j = 1:length(gauss\_points)

xi = gauss\_points(i);

eta = gauss\_points(j);

weight = gauss\_weights(i) \* gauss\_weights(j);

این حلقه‌ها بر روی نقاط گاوس تکرار می‌شوند تا انتگرال‌ها در این نقاط محاسبه شوند.

4. ارزیابی توابع شکل و گرادیان‌ها در نقاط گاوس

% Evaluate shape functions and gradients at Gauss points

N = double(subs(shape\_functions, {'xi', 'eta'}, {xi, eta}));

dN\_dxi = double(subs(shape\_gradients(1, :), {'xi', 'eta'}, {xi, eta}));

dN\_deta = double(subs(shape\_gradients(2, :), {'xi', 'eta'}, {xi, eta}));

N: توابع شکل ارزیابی شده در نقاط گاوس.

dN\_dxi و dN\_deta: گرادیان‌های توابع شکل نسبت به مختصات طبیعی 𝜉و 𝜂 ارزیابی شده در نقاط گاوس.

5. محاسبه ماتریس ژاکوبی و دترمینان آن

% Jacobian matrix and determinant

J = [dN\_dxi; dN\_deta] \* coords;

detJ = det(J);

invJ = inv(J);

J: ماتریس ژاکوبی که از حاصل‌ضرب گرادیان‌های توابع شکل و مختصات گره‌ها به دست می‌آید.

detJ: دترمینان ماتریس ژاکوبی.

invJ: ماتریس معکوس ژاکوبی.

6. محاسبه گرادیان توابع شکل در مختصات فیزیکی

% Gradient of shape functions in physical coordinates

dN\_dx = invJ \* [dN\_dxi; dN\_deta];

dN\_dx: گرادیان توابع شکل نسبت به مختصات فیزیکی.

7 . محاسبه و اضافه کردن به ماتریس سختی عناصر

% Element stiffness matrix

Ke = Ke + (dN\_dx' \* dN\_dx) \* detJ \* weight;

Ke: محاسبه هر عضو از ماتریس سختی المان از طریق انتگرال‌گیری عددی و اضافه کردن به ماتریس سختی اولیه.

8. محاسبه و اضافه کردن به بردار نیرو

% Element force vector

Fe = Fe + N \* detJ \* weight;

Fe: محاسبه هر عضو از بردار نیرو از طریق انتگرال‌گیری عددی و اضافه کردن به بردار نیرو اولیه.

Gaussian Quadrature : ربع گاوسی (روش گاوس)

ربع گاوسی یک تکنیک یکپارچه سازی عددی است که برای تقریب انتگرال معین یک تابع استفاده می شود. ایده این است که انتگرال را با مجموع وزنی مقادیر تابع در نقاط خاص (که نقاط گاوس نامیده می شود) در حوزه انتگرال تقریب می زنیم.

در شرایط ریاضی، برای یک تابع معین f(x) که در بازه [1،1-] تعریف شده است، فرمول ربع گاوسی را می توان به صورت زیر نوشت:



که در آن xi نقاط گاوس و wi وزن های مربوطه هستند. انتخاب xi و wi به مرتبه n مربع بستگی دارد.

مراحل استفاده از کوادرچر گوسی در FEM

1. نگاشت به عناصر مرجع:

* تبدیل دامنه‌ی انتگرال به عناصر • (معمولاً [1,1-] برای یک بعد، [1,1-]×[ 1,1-] برای دو بعد و غیره).

1. تعیین نقاط گوسی و وزن‌ها:
   * برای انتگرال‌گیری یک بعدی، نقاط گوسی معمولاً

[-1/sqrt{3}, 1/sqrt{3}]با وزن‌های [1,1] هستند.

* + برای انتگرال‌گیری دو بعدی، نقاط روی مربع مشخص می‌شوند

[−1,1]×[−1,1]

1. ارزیابی توابع شکل و گرادیان‌هایشان:

* محاسبه توابع شکل (N) و گرادیان‌هایش در نقاط گاوسی فعلی با استفاده از جایگذاری نمادین.

1. محاسبه ماتریس ژاکوبین و دترمینالش:
   * ماتریس ژاکوبین J مشتقات عنصر مرجع را به مشتقات عنصر واقعی مرتبط می کند. تعیین کننده J (که با (det(J) مشخص می شود) و معکوس آن (inv(J)) برای تبدیل انتگرال ها از عنصر مرجع به عنصر واقعی استفاده می شود.
2. ساخت ماتریس سختی و بردار نیرو برای عناصر:
   * با استفاده از گرادیان توابع شکل در مختصات فیزیکی، ماتریس سختی عنصر 𝐾𝑒 و بردار نیرو را محاسبه می کند.

**6-تابع apply\_boundary\_conditions:**

این تابع ماتریس سختی کلی K و بردار نیروی جهانی F را برای اعمال شرایط مرزی دیریکله اصلاح می کند.

function [K, F] = apply\_boundary\_conditions(K, F, nodes)

tol = 1e-5;

boundary\_nodes = find(nodes(:, 1) < tol | nodes(:, 1) > 2-tol | nodes(:, 2) < tol | nodes(:, 2) > 2-tol);

for i = 1:length(boundary\_nodes)

node = boundary\_nodes(i);

K(node, :) = 0;

K(node, node) = 1;

F(node) = 0;

end

end

1. تعریف تابع و ورودی‌های آن

function [K, F] = apply\_boundary\_conditions(K, F, nodes)

تابع apply\_boundary\_conditions را تعریف می کند که ماتریس سختی کلی K، بردار نیروی جهانی F و مختصات گره ها را می گیرد.

2.تولرانس

tol = 1e-5;

tol یک مقدار تلورانس است که برای شناسایی گره های روی مرز استفاده می شود. این مقدار کوچک برای محاسبه مسائل دقت عددی استفاده می شود.

3.گره های مرزی

boundary\_nodes = find(nodes(:, 1) < tol | nodes(:, 1) > 2-tol | nodes(:, 2) < tol | nodes(:, 2) > 2-tol);

boundary\_nodes آرایه ای از شاخص های گره است که در مرز دامنه قرار دارند. این خط گره هایی را پیدا می کند که در آنها:

nodes(:, 1) < tolگره های نزدیک مرز چپ را مشخص می کند (x = 0).

nodes(:, 1) > 2 - tol گره های نزدیک مرز سمت راست را شناسایی می کند (x = 2).

nodes(:, 2) < tolگره های نزدیک به مرز پایین را شناسایی می کند (y = 0).

nodes(:, 2) > 2 – tol)گره های نزدیک مرز بالایی را شناسایی می کند y = 2.(

4. اعمال شرایط مرزی

for i = 1:length(boundary\_nodes)

node = boundary\_nodes(i);

K(node, :) = 0;

K(node, node) = 1;

F(node) = 0;

end

این حلقه بر روی هر گره مرزی تکرار می شود و ماتریس سختی کلی K و بردار نیروی جهانی F را اصلاح می کند:

K(node, :) = 0;: تمام ورودی های ردیف مربوط به گره مرزی را صفر می کند. این به طور موثر هرگونه تأثیر این گره را از معادلات حذف می کند.

K(node, node) = 1; ورودی مورب برای گره مرزی را برابر با 1 قرار می دهد. این شرط دیریکله را با ایجاد هویت ماتریس سختی برای این گره، اعمال می کند.

F(node) = 0;: ورودی مربوطه را در بردار نیرو صفر می کند و از همگن بودن شرط مرزی اطمینان می دهد (یعنی ).

**7-تابع: plot\_solution**

این تابع حل معادله پواسون را رسم می کند.

function plot\_solution(nodes, elements, U)

% Create a grid of the solution

L = 2; % Size of the domain

Nx = 100; % Number of elements in x direction

Ny = 100; % Number of elements in y direction

x = linspace(0, L, Nx+1); % Generate a vector of x coordinates

y = linspace(0, L, Ny+1); % Generate a vector of y coordinates

[X, Y] = meshgrid(x, y); % Create a grid of points (X, Y) from vectors x and y

Z = griddata(nodes(:, 1), nodes(:, 2), U, X, Y); % Interpolate U values onto the grid (X, Y)

% Plot the solution

surf(X, Y, Z); % Create a surface plot of Z values over the grid (X, Y)

title('Solution of the Poisson equation using serendipity elements'); % Set the title of the plot

xlabel('x'); % Label for the x-axis

ylabel('y'); % Label for the y-axis

zlabel('u'); % Label for the z-axis (or color scale for the surface)

view(2); % Set the view angle to 2D (view from the top)

colorbar; % Display a colorbar showing the scale of u values

end

1. تعریف تابع و ورودی های آن:

function plot\_solution(nodes, elements, U): یک تابع plot\_solution را تعریف می کند که سه آرگومان می گیرد: گره ها، عناصر و U.

: L = 2اندازه دامنه را در جهت x و 2 y واحد تعریف می کند.

: Nx = 100تعداد عناصر را در جهت x (با فرض مش یکنواخت) مشخص می کند.

: Ny = 100تعداد عناصر را در جهت y (با فرض مش یکنواخت) مشخص می کند.

2. ایجاد بردارهای مختصات:

x = linspace(0, L, Nx+1) : بردار x از مختصات 0 تا L را با نقاط Nx+1 ایجاد می کند.

y = linspace(0, L, Ny+1) : بردار y از مختصات 0 تا L را با نقاط Ny+1 ایجاد می کند.

3. ایجاد شبکه برای رسم

[X, Y] = meshgrid(x, y) : یک شبکه دو بعدی از نقاط (X, Y) از بردارهای x و y ایجاد می کند. این شبکه مشخص می کند که راه حل U در کجا رسم می شود.

4.درونیابی مقادیر راه حل

راه حل U را با استفاده از مختصات گره از گره ها (:، 1) (مختصات x) و گره ها (:، 2) (مختصات y) روی شبکه (X, Y) درون یابی می کند. این مرحله تضمین می‌کند که راه‌حل U برای ترسیم بر روی شبکه ساختاریافته (X, Y) نگاشت می‌شود..

5.رسم راه حل

surf(X, Y, Z): یک نمودار سطح سه بعدی از مقادیر Z را روی شبکه (X, Y) ایجاد می کند. این راه حل U را به عنوان یک سطح در فضای سه بعدی تجسم می کند.

title('حل معادله پواسون با استفاده از عناصر سرندیپیتی'): عنوان طرح را برای توصیف ماهیت راه حل ترسیم شده تنظیم می کند.

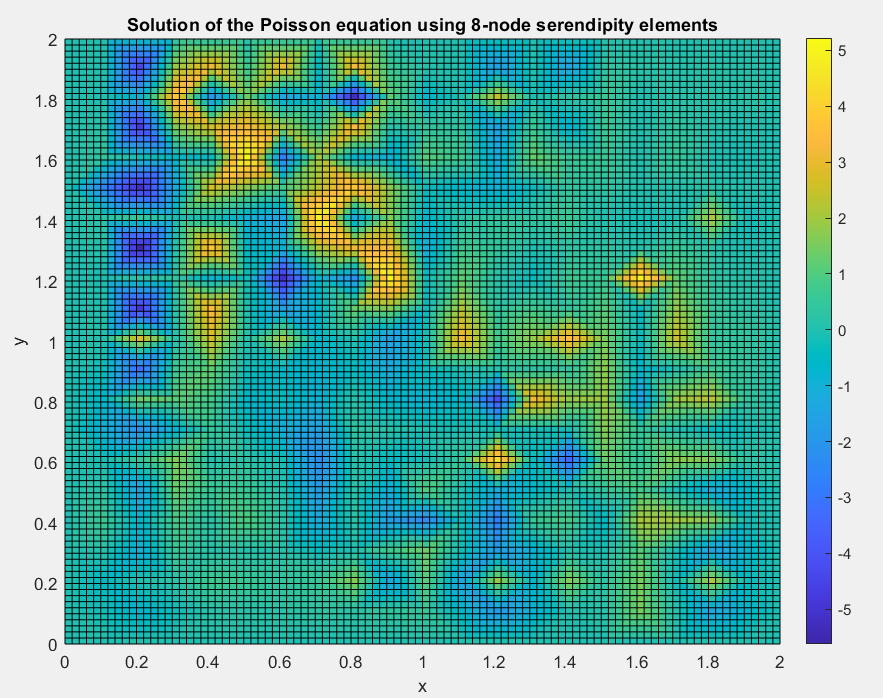
xlabel('x'); ylabel('y'); zlabel('u');: محورهای نمودار را با x و y معرف مختصات فضایی و u نشان دهنده مقادیر راه حل برچسب گذاری می کند.

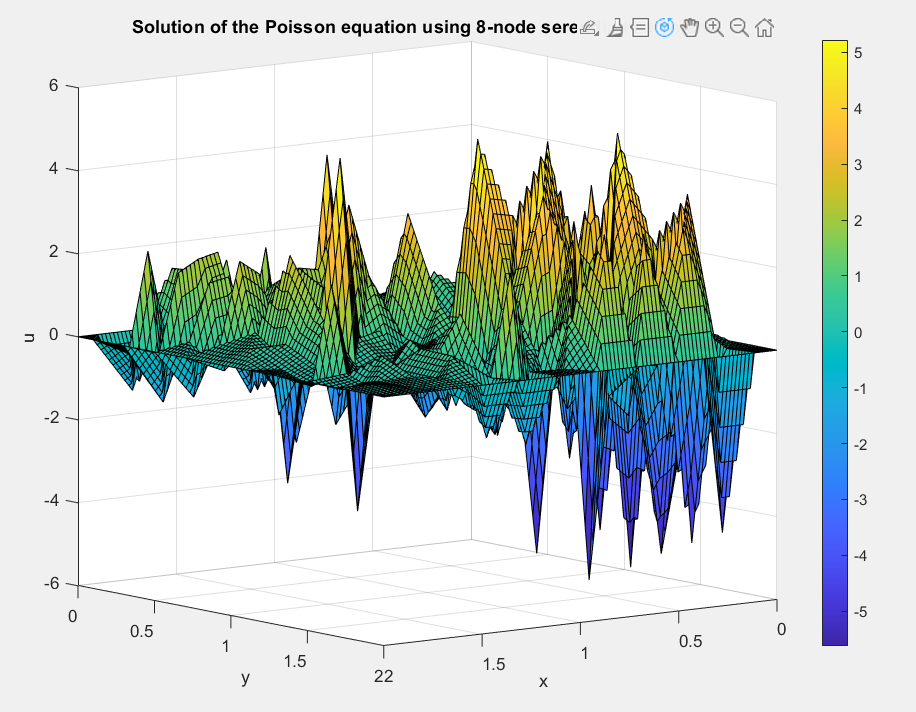
view(2): : زاویه دید را روی 2D تنظیم می کند، به این معنی که طرح از بالا مشاهده می شود.

colorbar : یک نوار رنگی به نمودار اضافه می کند که مقیاس مقادیر u مربوط به رنگ های مختلف روی سطح را نشان می دهد.

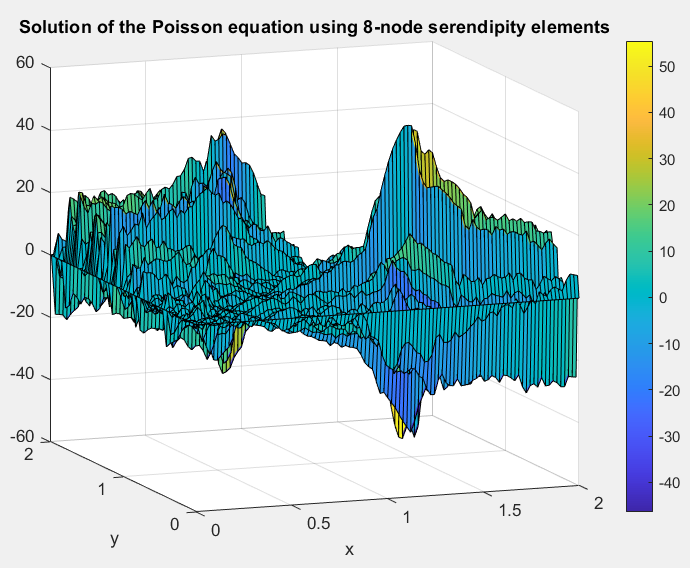
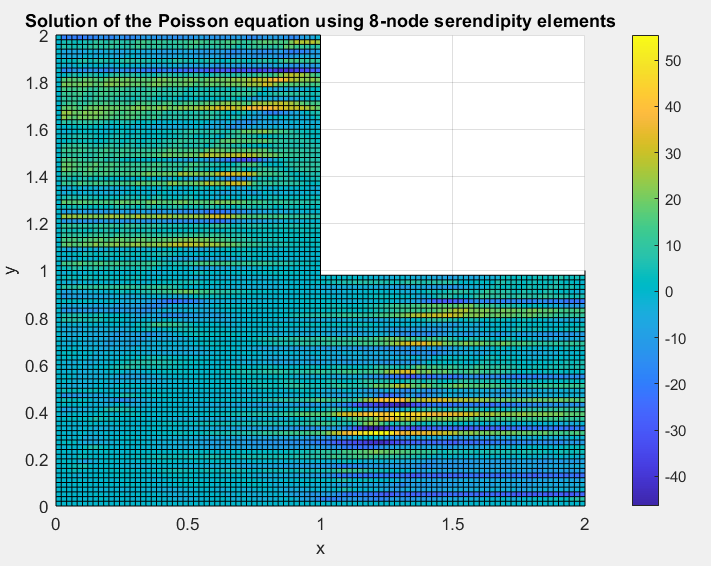
این تابع تولید مش، درون یابی راه حل و تجسم را در یک نمودار منسجم از راه حل عددی U با استفاده از تابع گفته شده انجام می دهد. بسته به اندازه دامنه خاص و وضوح مش که برای مشکل شما نیاز است، می‌توان پارامترهایی مانند L، Nx و Ny را تنظیم کرد.

**7-نتایج کد توضیح داده شده:**

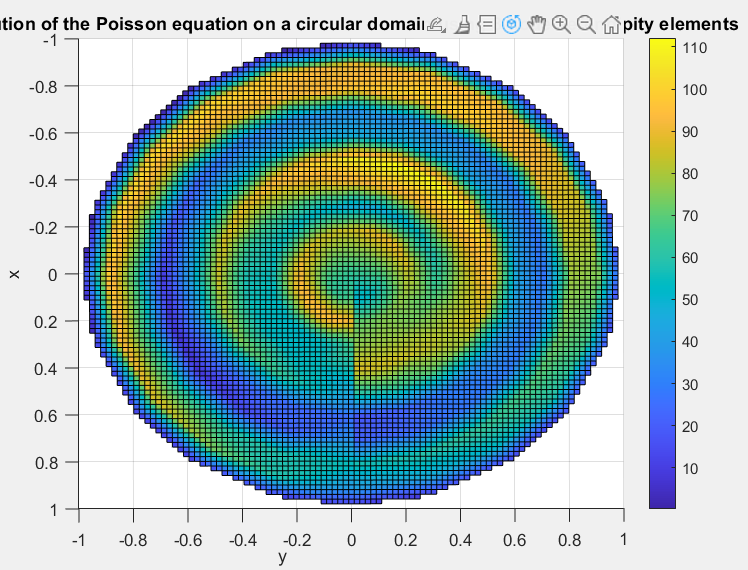


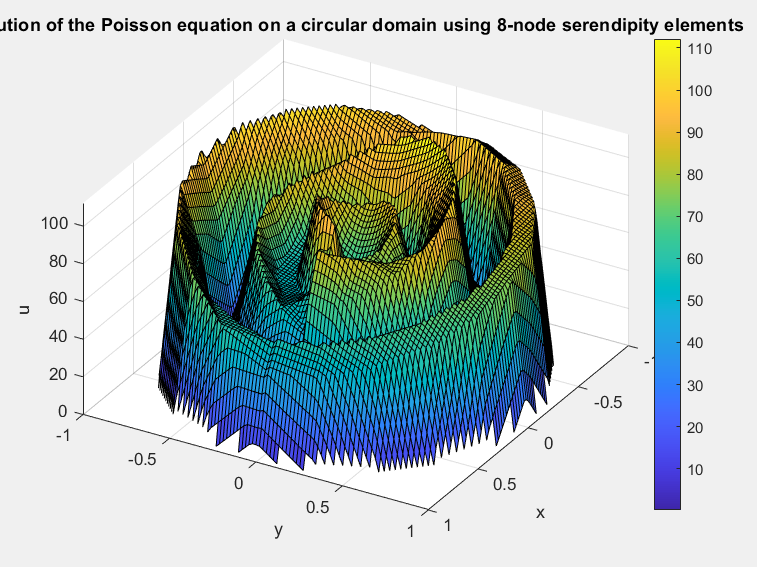


رسم نمودار جواب های معادله ی پواسون روی دامنه ی L-shaped :



رسم نمودار جواب های معادله ی پواسون روی دامنه ی دایره ای :





**lagrange element method**

در این بخش کد روش عناصر لاگرانژ را بدون توضیح جزئیات قرار می دهیم(کد مشابه حالت عناصر سرندیپیتی با اندکی تغییر در جزئیات که شامل یک گره اضافه تر ، یک تابع شکل اضافه تر و حل مشکلات بعد می باشد)

function poisson\_lagrange\_9node()

% Define geometry

L = 2; % Size of the domain

Nx = 10; % Number of elements in x direction

Ny = 10; % Number of elements in y direction

[nodes, elements] = generate\_mesh(L, Nx, Ny);

% Define Lagrange shape functions and their gradients

[shape\_functions, shape\_gradients] = define\_lagrange\_elements();

% Assemble global stiffness matrix and force vector

[K, F] = assemble\_system(nodes, elements, shape\_functions, shape\_gradients);

% Apply boundary conditions

[K, F] = apply\_boundary\_conditions(K, F, nodes);

% Solve the linear system

U = K \ F;

% Plot the solution

plot\_solution(nodes, elements, U);

end

function [nodes, elements] = generate\_mesh(L, Nx, Ny)

% Generate a structured mesh for the domain with mid-side and center nodes

[x, y] = meshgrid(linspace(0, L, Nx+1), linspace(0, L, Ny+1));

nodes = [x(:), y(:)];

% Generate mid-side nodes for horizontal and vertical edges

mid\_x = (x(:, 1:end-1) + x(:, 2:end)) / 2;

mid\_y\_horizontal = y(:, 1:end-1);

mid\_nodes\_x = [mid\_x(:), mid\_y\_horizontal(:)];

mid\_y = (y(1:end-1, :) + y(2:end, :)) / 2;

mid\_x\_vertical = x(1:end-1, :);

mid\_nodes\_y = [mid\_x\_vertical(:), mid\_y(:)];

% Generate center nodes

center\_x = (x(1:end-1, 1:end-1) + x(2:end, 2:end)) / 2;

center\_y = (y(1:end-1, 1:end-1) + y(2:end, 2:end)) / 2;

center\_nodes = [center\_x(:), center\_y(:)];

% Combine mid-side and center nodes

mid\_nodes = [mid\_nodes\_x; mid\_nodes\_y; center\_nodes];

% Combine all nodes

nodes = [nodes; mid\_nodes];

% Add an index for the mid-side and center nodes to the elements array

num\_original\_nodes = (Nx + 1) \* (Ny + 1);

num\_mid\_x\_nodes = numel(mid\_x);

num\_mid\_y\_nodes = numel(mid\_y);

num\_center\_nodes = numel(center\_x);

elements = [];

for i = 1:Nx

for j = 1:Ny

n1 = (j-1)\*(Nx+1) + i;

n2 = n1 + 1;

n3 = n2 + Nx + 1;

n4 = n1 + Nx + 1;

n5 = num\_original\_nodes + (j-1)\*Nx + i; % mid-side node on the bottom edge

n6 = num\_original\_nodes + num\_mid\_x\_nodes + (i-1)\*Ny + j; % mid-side node on the right edge

n7 = num\_original\_nodes + (j-1)\*Nx + i + Nx; % mid-side node on the top edge

n8 = num\_original\_nodes + num\_mid\_x\_nodes + (i-1)\*Ny + j - 1; % mid-side node on the left edge

n9 = num\_original\_nodes + num\_mid\_x\_nodes + num\_mid\_y\_nodes + (j-1)\*Nx + i; % center node

elements = [elements; n1, n2, n3, n4, n5, n6, n7, n8, n9];

end

end

end

function [shape\_functions, shape\_gradients] = define\_lagrange\_elements()

syms xi eta;

shape\_functions = [

0.25 \* (1 - xi) \* (1 - eta) \* (xi + eta + 1); % Shape function 1

0.25 \* (1 + xi) \* (1 - eta) \* (-xi + eta + 1); % Shape function 2

0.25 \* (1 + xi) \* (1 + eta) \* (-xi - eta + 1); % Shape function 3

0.25 \* (1 - xi) \* (1 + eta) \* (xi - eta + 1); % Shape function 4

0.5 \* (1 - xi^2) \* (1 - eta); % Shape function 5 (mid-side node on the bottom edge)

0.5 \* (1 + xi) \* (1 - eta^2); % Shape function 6 (mid-side node on the right edge)

0.5 \* (1 - xi^2) \* (1 + eta); % Shape function 7 (mid-side node on the top edge)

0.5 \* (1 - xi) \* (1 - eta^2); % Shape function 8 (mid-side node on the left edge)

(1 - xi^2) \* (1 - eta^2) % Shape function 9 (center node)

];

shape\_gradients = [

diff(shape\_functions, xi)';

diff(shape\_functions, eta)'

];

end

function [K, F] = assemble\_system(nodes, elements, shape\_functions, shape\_gradients)

num\_nodes = size(nodes, 1);

num\_elements = size(elements, 1);

K = sparse(num\_nodes, num\_nodes);

F = zeros(num\_nodes, 1);

for e = 1:num\_elements

element\_nodes = elements(e, :);

coords = nodes(element\_nodes, :);

% Compute element stiffness matrix and force vector

[Ke, Fe] = element\_stiffness\_force(coords, shape\_functions, shape\_gradients);

% Assemble into global stiffness matrix and force vector

for i = 1:length(element\_nodes)

for j = 1:length(element\_nodes)

K(element\_nodes(i), element\_nodes(j)) = K(element\_nodes(i), element\_nodes(j)) + Ke(i, j);

end

F(element\_nodes(i)) = F(element\_nodes(i)) + Fe(i);

end

end

end

function [Ke, Fe] = element\_stiffness\_force(coords, shape\_functions, shape\_gradients)

% Numerical integration (Gauss quadrature)

gauss\_points = [-1/sqrt(3), 1/sqrt(3)];

gauss\_weights = [1, 1];

num\_nodes\_per\_element = size(coords, 1);

Ke = zeros(num\_nodes\_per\_element, num\_nodes\_per\_element);

Fe = zeros(num\_nodes\_per\_element, 1);

for i = 1:length(gauss\_points)

for j = 1:length(gauss\_points)

xi = gauss\_points(i);

eta = gauss\_points(j);

weight = gauss\_weights(i) \* gauss\_weights(j);

% Evaluate shape functions and gradients at Gauss points

N = double(subs(shape\_functions, {'xi', 'eta'}, {xi, eta}));

dN\_dxi = double(subs(shape\_gradients(1, :), {'xi', 'eta'}, {xi, eta}));

dN\_deta = double(subs(shape\_gradients(2, :), {'xi', 'eta'}, {xi, eta}));

% Jacobian matrix and determinant

J = [dN\_dxi; dN\_deta] \* coords;

detJ = det(J);

invJ = inv(J);

% Gradient of shape functions in physical coordinates

dN\_dx = invJ \* [dN\_dxi; dN\_deta];

% Element stiffness matrix

Ke = Ke + (dN\_dx' \* dN\_dx) \* detJ \* weight;

% Element force vector

Fe = Fe + N \* detJ \* weight;

end

end

end

function [K, F] = apply\_boundary\_conditions(K, F, nodes)

tol = 1e-5;

boundary\_nodes = find(nodes(:, 1) < tol | nodes(:, 1) > 2-tol | nodes(:, 2) < tol | nodes(:, 2) > 2-tol);

for i = 1:length(boundary\_nodes)

node = boundary\_nodes(i);

K(node, :) = 0;

K(node, node) = 1;

F(node) = 0;

end

end

function plot\_solution(nodes, elements, U)

% Create a grid of the solution

L = 2; % Size of the domain

Nx = 100; % Number of elements in x direction

Ny = 100; % Number of elements in y direction

x = linspace(0, L, Nx+1);

y = linspace(0, L, Ny+1);

[X, Y] = meshgrid(x, y);

Z = griddata(nodes(:, 1), nodes(:, 2), U, X, Y);

% Plot the solution

surf(X, Y, Z);

title('Solution of the Poisson equation using 9-node Lagrange elements');

xlabel('x');

ylabel('y');

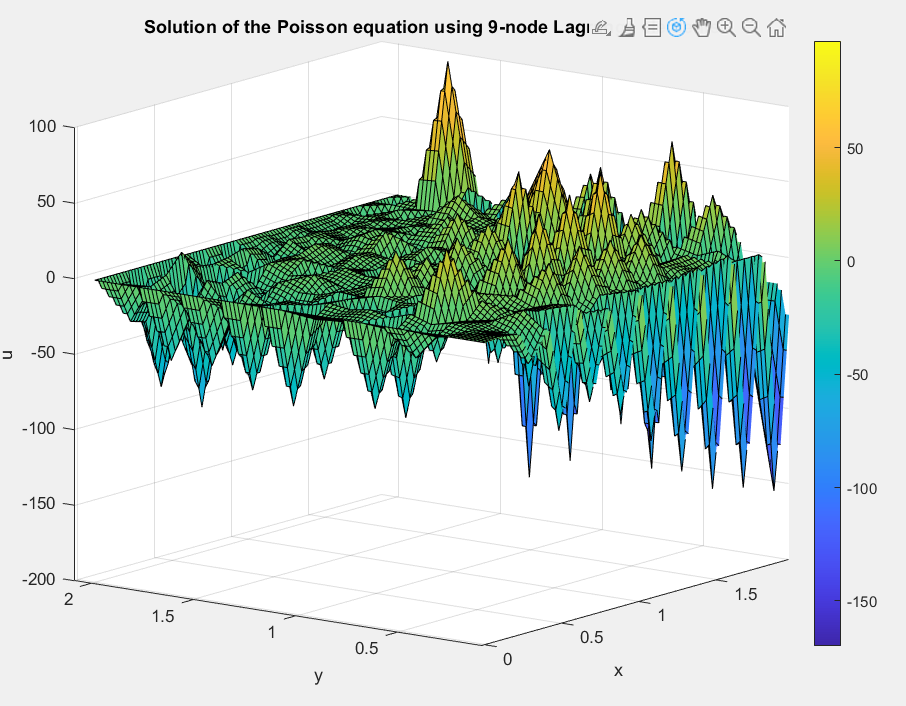
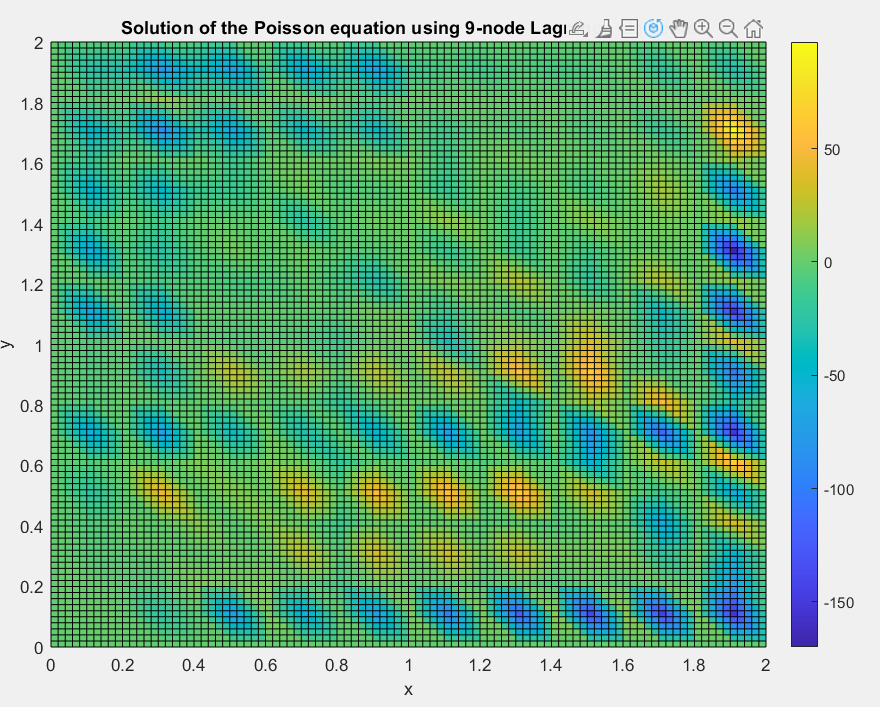
zlabel('u');

view(2); % View from the top

colorbar;

end

نتایج مربوط به این کد :

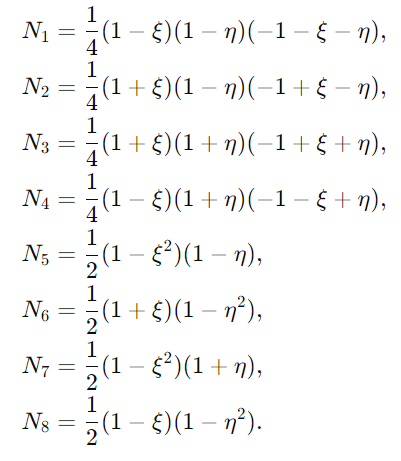


**تفاوت‌های عددی و ریاضیاتی روش عناصر لاگرانژ و سرندیپیتی**

**1. توابع شکل**

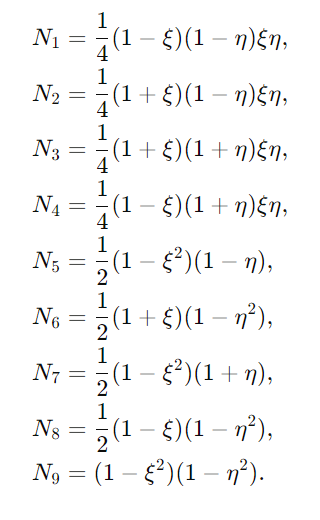
**عناصر سرندیپیتی 8 گره**

توابع شکل سرندیپیتی 8 گره معمولاً به‌صورت زیر تعریف می‌شوند:



**عناصر لاگرانژ 9 گره**

توابع شکل لاگرانژ 9 گره معمولاً به‌صورت زیر تعریف می‌شوند:



**توضیح تحلیلی**

گره ها و درجات آزادی: عناصر Serendipity دارای گره ها و درجات آزادی کمتری در مقایسه با عناصر لاگرانژ با همان نظم هستند. به عنوان مثال، یک عنصر سرندیپیتی 8 گره از گره مرکزی موجود در یک عنصر لاگرانژ 9 گره می گذرد.

درون یابی: توابع شکل برای بهبود کارایی محاسباتی و در عین حال حفظ دقت طراحی شده اند. با این حال، آنها دقیقاً در همه گره ها درون یابی نمی کنند.

کارایی: عناصر Serendipity تعادل خوبی بین دقت و هزینه محاسباتی ارائه می دهند که اغلب برای مشکلات خاص کارآمدتر هستند.

**تفاوت های کلیدی**

پیکربندی گره:

عناصر لاگرانژ: شامل گره های گوشه، نقاط میانی لبه و احتمالاً گره های داخلی است.

عناصر Serendipity: شامل گره های گوشه و نقاط میانی لبه است اما معمولاً گره های داخلی را حذف می کند.

ساختار تابع شکل:

عناصر لاگرانژ: توابع شکل بر اساس چند جمله‌ای لاگرانژ هستند که درون یابی دقیق را در همه گره‌ها ارائه می‌کنند.

عناصر Serendipity: توابع شکل ترکیبی از اصطلاحات چند جمله ای هستند که تعداد گره ها را بدون کاهش قابل توجهی در دقت کاهش می دهند.

ویژگی های درون یابی:

عناصر لاگرانژ: دقیقاً در همه گره‌ها درون‌یابی می‌شوند، که آنها را برای هندسه‌های پیچیده بسیار دقیق می‌کند.

عناصر Serendipity: دقیقاً در همه گره ها درون یابی نمیکند، اما محاسبات کارآمدتری را برای بسیاری از مشکلات عملی ارائه می دهد.

کارایی محاسباتی:

عناصر لاگرانژ: تعداد گره ها و درجات آزادی بیشتر می تواند منجر به هزینه های محاسباتی بالاتر شود.

عناصر Serendipity: گره‌ها و درجات آزادی کمتر هزینه‌های محاسباتی را کاهش می‌دهند و در عین حال سطح معقولی از دقت را حفظ می‌کنند.

دقت و کاربرد:

عناصر لاگرانژ: برای مشکلاتی که نیاز به دقت بالا و راه حل های صاف دارند ترجیح داده می شود.

عناصر Serendipity: اغلب برای مشکلاتی که بازده محاسباتی حیاتی تر است و کاهش جزئی در دقت قابل قبول است استفاده می شود.

**نتیجه‌گیری**

انتخاب بین عناصر سرندیپیتی 8 گره و لاگرانژ 9 گره بستگی به نیازهای خاص مسئله و توازنی بین دقت و پیچیدگی محاسباتی دارد. در حالی که عناصر لاگرانژ 9 گره دقت بالاتری ارائه می‌دهند، اما محاسبات بیشتری نیاز دارند و ممکن است برای مسائل بزرگ و پیچیده، زمان محاسباتی زیادی را طلب کنند. از سوی دیگر، عناصر سرندیپیتی 8 گره با پیچیدگی کمتر و محاسبات سریع‌تر، می‌توانند در مسائل ساده‌تر و با دقت کمتر مورد استفاده قرار گیرند.