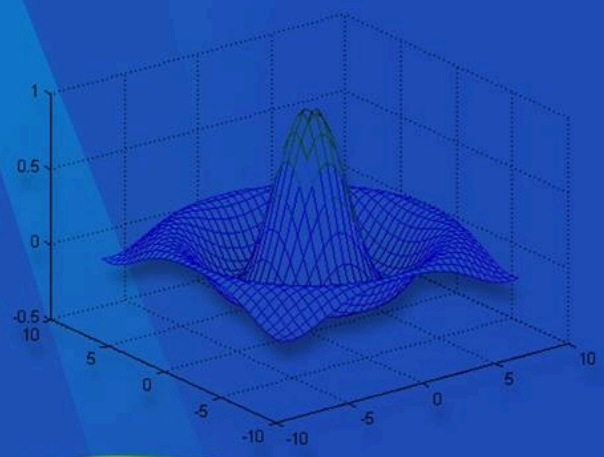
المقدمة في نظام ماتلاب



المهندس
حسن الحوري

المهندس حسن الحوري

المقدمة في نظام ماتلاب

Hasan.Alhouri@gmail.com

جدول المحتويات

جدول المحتويات 3
مقدمة الكتاب 5
التعريف بنظام ماتلاب 11
بيئة النظام وأدواته 15
العمليات على المصفوفات 61
النصوص والمتحولات النصية 115
خلاياالبيانات المتنوعة 129
بنى المعطيات المركبة 139
الأعداد العقدية (المركبة) 149
الرسم البياني 157
الأدوات البرمجية 235
البرمجة غرضية التوجه 287
كثيرات الحدود 307

المعالجة الرمزية 319

قرناء ماتلاب وبدائله 335

الملحقات 347

مقدمة الكتاب

عندما تريد كسر عود فأنت تكسرها بيدك أما عندما تريد قطع شجرة فعليك أن تستخدم الفأس أو المنشار. سيبدو غباء أن تستخدم الفأس أو المنشار لكسر العود، وسيبدو غباء وضعفاً أن تتخلى عن قطع شجرتك لأنك لا تقوى على فعل ذلك بيديك المجردتين.

عندما تحتاج إلى جمع عددين أو عدة أعداد فمن المضحك أن تفكر في شراء حاسوب وإن كان لديك الحاسوب فمن المضحك أن تبدأ في كتابة برنامج لجمع تلك الأعداد. دخلنا هنا في التعقيدات المنطقية، فأغلبنا اليوم لديهم حواسب شخصية محمولة وغير محمولة وجوالات عليها تطبيقات لمعظم الأغراض، نعم هذا صحيح ولكن هناك تراتبية في البحث عن الحل، أولا إذا كنت أستطيع فعل الأمر باليد المجردة فأفعل، ثم إن لم أكن أستطيع وكان لدي أداة فأستخدمها ولكن مع الانتباه إلى حجم الأداة وكلفة تشغيلها مقابل الثمرة المرجوة من ذلك، ثم إن لم يكن لدي أداة وكنت أستطيع صنعها فأصنعها وكذلك مع الانتباه إلى العامل السابق.

في المثال السابق عن جمع عددين يمكن العمل ذهنياً أو باستخدام ورقة وقلم أو آلة حاسبة، وعندما تزداد الأعداد المراد جمعها يصبح وجود تطبيق من تطبيقات الجداول الاكترونية أمراً مجدياً، أما الشروع في كتابة برنامج جديد فهو العمل الذي نقوم به عندما يكون لدينا الكثير من المعالجة التي لا تلبيها البرمجيات المتوفرة لدينا وعندما يكون طريق صناعة برنامج جديد أقصر من طريق إنجاز العمل بالأدوات المتوفرة، طبعا مع الانتباه إلى وجود حالة التكرار للعمل فقد يكون لدينا عمل يوم

ونقوم بصناعة تطبيق جديد ينجزه في نصف ساعة ولكن صناعة التطبيق في حد ذاتها تحتاج إلى أسبوع من العمل، في هذه الحالة سيكون من الأفضل إنجاز العمل بدون تطبيق جديد فقط إذا لم يكن من المتوقع أبداً أن نحتاج إلى عمله ثانية، أما إذا كان لدينا عمل كهذا كل فترة من الزمن فلا بد أن التطبيق الجديد سيكون مجديا.

لم يتعب العلماء أنفسهم على مدى قرون من الزمن من أجل وضع نظريات وتطوير طرق وابتكار آلات لكسر العود، بل بذلوا جهودهم وسهروا لياليهم وأحرقوا طاقاتهم لإنجاز ما لايمكن إنجازه بالطرق والأساليب والأدوات المألوفة.

اليوم نجد الكثير من التطبيقات والبرمجيات وحتى الكثير من لغات البرمجة لدرجة أنه أصبح بالإمكان بناء لغة برمجة جديدة بجهد قليل نسبيا مقارنة مع ما كان عليه الأمر في الماضي. في حقل الرياضيات والهندسة هناك الكثير من التعقيدات فيما يتعلق باستخدام الأدوات البرمجية، ويقع الرياضي أو الفيزيائي أو المهندس أو غيرهم من العاملين في الحقول العلمية في حيرة إزاء مهمات تطرح نفسها على أحدهم، هل عليه أن يستخدم التطبيقات المتوفرة؟ إنها لا تلبي الحاجات! إذن عليه أن يكون مبرمجا ويبرمج شيئا جديداً؟ ولكن لم تصبح فكرة الموضوع جاهزة بعد، إنها لم تكتمل! .... وتساؤلات عديدة أخرى.

وجدت بعض الأنظمة التي تتيح إمكانية العمل بشكل يشبه البرمجة ولكن دون التعقيدات التي يتطلبها إنشاء برنامج جديد، فهناك نافذة أوامر يمكن من خلالها كتابة الأوامر البرمجية البسيطة لإنجاز العمليات المطلوبة بسرعة ورؤية النتائج مباشرة وكأننا نكتب

البرامج ولكننا لا نفعل، بل وكأننا نستخدم التطبيقات إذن، ولكنها أوامر برمجية. بيئة المعالجة من هذا النوع هي بيئة وسط بين تطبيق المستخدم النهائي وبيئة صناعة البرمجيات، وهي تقدم إمكانات هائلة، وفي بعض هذه الأنظمة هناك إمكانية لصناعة البرامج فعلا عندما تصبح الفكرة جاهزة ومتبلورة وهي إمكانية رائعة فالنظام من هذا النوع يتحول إلى تطبيق مستخدم نهائي ولغة برمجة وبيئة معالجة شبيهة بلغة البرمجة وغير ذلك مما يضاف عليه.

في مقدمة هذه الأنظمة المتطورة يأتي ماتلاب الذي يحوي الآلاف من الأدوات المجمعة في أدوات أكبر ويمكن إنجاز العمليات فيه بسرعة كبيرة وجهد قليل بفضل مكتباته الضخمة. أما تعلم ماتلاب فهو شيء مختلف فلا يمكن لأحد أن يتعلم ماتلاب كاملاً وهو أمر غير مجد أصلاً، فكل من يعمل في حقل من حقول العلم ويريد تعلم ماتلاب عليه أن يتعلم الجزء الذي يخص عمله ويفيده فيه، ذلك أن بعض الأجزاء من نظام ماتلاب يحتاج فهمها إلى التخصص فهي تقدم خدمات لا يفهمها ويفهم الغاية منها إلا ذو خبرة في المجال الذي تخدمه تلك الأجزاء. ولكن هناك قاعدة أساسية على جميع المتعلمين أن ينطلقوا منها ثم يتوقف البعض عندها وينطلق آخرون لسبر أغوار أكثر عمقاً وتخصصاً. لنفرض وجود سوق تجاري كبير (مول)، على جميع الزبائن أن يدخلو من الباب الرئيسي ويودعوا أغراضهم في الأمانات، بعضهم يجد حاجاته في بهو السوق فيشتري وينهي زيارته وبعضهم يضطر لزيارة الطوابق الأخرى أو الأقسام الأخرى ليجد ضالته، وفي النهاية على الجميع أن يأتي بما اشترى إلى طاولة

الحساب ويدفع قيمة الفاتورة ثم يأخذ أغراضه من الأمانات وينصرف، ربما لا يكون مع أحدهم أغراضاً ليضعها في الأمانات ولكن عليه أن يعرف أن هناك أمانات وأن عليه وضع أغراضه فيها عندما يكون لديه أغراض. في ماتلاب أيضا هناك أساسيات على الجميع معرفتها حتى لو كان بعضها لا يبدو لازماً في مرحلة ما، وهناك تفاصيل ودقائق أمور وأدوات ومكتبات خاصة لا يتعلمها إلا من يحتاج إليها.

تم تأليف هذا الكتاب ليغطي تلك الأساسيات، لا ليغطي كل نظام ماتلاب، وفي الواقع ليس ماتلاب هو الهدف بعينه فهناك أنظمة بديلة ولست بصدد التسويق لماتلاب أو لغيره ولكن لا يمكن في كتاب واحد أن نشرح استخدام عدة أنظمة، ولهذا قمت بشرح ماتلاب لأنه الأقوى من وجهة نظري والأكثر استخداماً. ثم ذكرت بعض الأنظمة البديلة الأكثر شهرة وأجريت مقارنة بسيطة بينها وبين ماتلاب ليتسنى للدارس أن يختار البقاء مع ماتلاب أو الانتقال إلى نظام آخر. في حالة اختيار نظام آخر بعد دراسة ماتلاب لا يكون الأمر صعباً كالبداية من الصفر، فمن يتعلم الكتابة بقلم الرصاص لا يكون عسيرا عليه أبداً أن يكتب بقلم الحبر الناشف.

عزيزي القارئ: لن أطيل في المقدمة أكثر فالوقت من ذهب. إن وجدت في هذا الكتاب متعة وفائدة فهو من توفيق الله عز وجل وإن وجدت أخطاء فكلنا نخطئ ولا ينجو من الخطأ إلا من لا يعمل، وبالنقد البناء نصوب أخطاءنا ونطور أعمالنا، لا تبخل علينا برأيك وملاحظاتك ونقدك البناء.

والله ولي الأمر والتوفيق.

التعريف بنظام ماتلاب

سمي ماتلاب بهذا الاسم نحتا من كلمتين matrix laboratory فهو يعالج كل البيانات التي يتم إدخالها إليه على أنها مصفوفات. قامت شركة MathWorks بتطوير هذا النظام ليكون عوناً للمهندسين والعاملين في الحقول العلمية على إنجاز الأعمال التي تحتاج إلى معالجة رقمية كبيرة بسرعة أكبر ودقة. يمكن أيضا في ماتلاب إنجاز الرسوم البيانية والمخططات الرياضية بكافة أشكالها. كذلك يوفر النظام إمكانية بناء التطبيقات التي تعمل في بيئته أو في بيئة مستقلة بواجهات رسومية أو بدون، ويمكن بناء تطبيقات كمزيج من إجراءات ماتلاب ولغات برمجة أخرى مثل Java, C, C++, Fortran, …

بدأ تطوير ماتلاب في أواخر سبعينيات القرن العشرين من قبل البروفيسور Cleve Moler كأداة لإنجاز المعالجات الرقمية دون الحاجة إلى كتابة الأكواد البرمجية، ثم انضم إليه المهندس Jack Little و Steve Bangert فأعادوا كتابة ماتلاب بلغة C وقاموا معاً بتأسيس شركة MathWork في العام 19784

أصبح ماتلاب اليوم نظاماً ضخماً من حيث إمكانياته وحجم المجالات التي يمكنه أن يكون مفيداً جداً فيها، ولقي انتشاراً واسعاً في أوساط العلماء والباحثين والمدرسين، وذلك بفضل ميزاته التي نذكر منها مايلي:

يعتبر ماتلاب لغة برمجة عالية المستوى من أجل الحسابات الرقمية والرسومية وبناء التطبيقات النهائية.

لدى ماتلاب واجهة تفاعلية تمكن المستخدم من إجراء الحسابات بسرعة دون برمجة وحل المسائل بإدخالات بسيطة وسريعة.

يوفر مكتبات هائلة متنوعة تضم توابعاً مبنية مسبقا built-in لإجراء الحسابات في شتى المجالات كالجبر والتحليل الرياضي والإحصاء والتحليل العددي ومعالجة الإشارة وغير ذلك.

يوفر أيضا مكتبات من التوابع الجاهزة لإنجاز الرسوم والمخططات وتعديلها بما يناسب الهدف منها بالإضافة إلى إمكانية حفظها لاستخدامه لاحقاً أو تصديرها كصور.

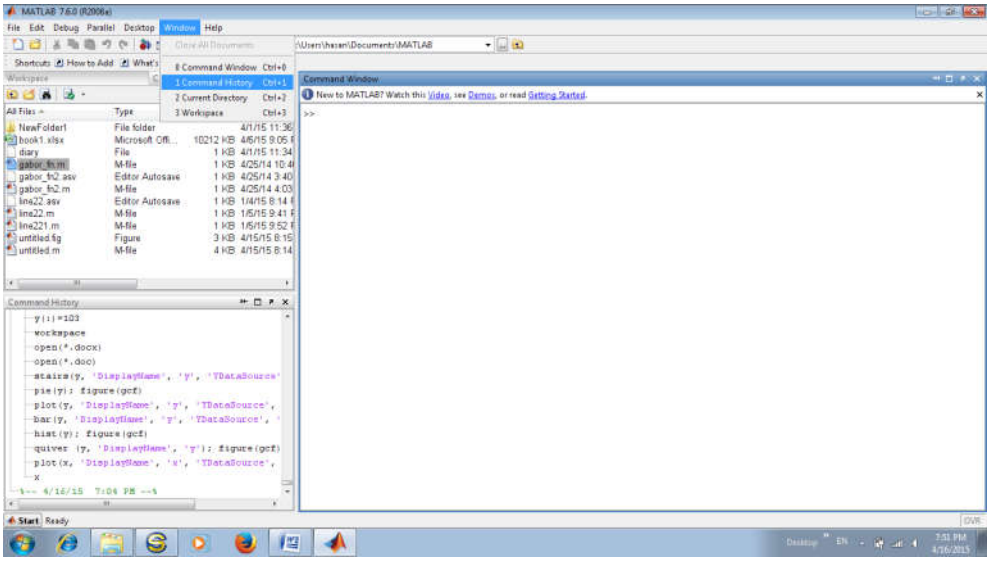
لدى ماتلاب أدوات تمكن المطور من بناء تطبيقات بمزج إمكانيات ماتلاب مع تطبيقات أخرى مثل java, C++, Excel, ..

يملك أيضا محرر خاص به لكتابة الشيفرات البرمجية يمكن من كتابة تلك الشيفرات ومراجعتها وتطويرها بسهولة لما للمحرر من ميزات وإمكانات.

بيئة النظام وأدواته

يعمل ماتلاب في بيئة نظام النوافذ عن طريق نافذة رئيسية تضم مجموعة من النوافذ الفرعية لكل منها مجموعة من الوظائف ويمكن الوصول إلى أوامر وأدوات ووظائف النظام من خلال نافذته الرئيسية والنوافذ الفرعية وقائمة ماتلاب الرئيسية وبقية القوائم في النافذة الرئيسية.

الشكل 1 يبين لقطة شاشة لواجهة النظام ويظهر عليها الوضع الافتراضي لتوضع عناصر هذه الواجهة ويمكن أن نرى فيه مايلي:



الشكل 1 الواجهة الافتراضية لنظام ماتلاب

النافذة الرئيسية: وهي الإطار الحاوي لجميع أدوات ونوافذ الواجهة الرسومية ومنها ما يظهر بشكل افتراضي ومنها ما يمكن الوصول إليه عن طريق القوائم والأوامر.

نافذة الأوامر: هذه النافذة يتم فيها إدخال التعليمات والأوامر لتنفيذها بشكل مباشر وهي تظهر أيضا نتائج تنفيذ تلك الأوامر.

نافذة مساحة العمل: في هذه النافذة يمكن معاينة أسماء المتحولات المخزنة في الذاكرة المؤقتة ومعلومات عن كل متحول منها.

نافذة المسار الحالي: تظهر هذه النافذة أسماء ومسارات الملفات المستخدمة حالياً.

نافذة تاريخ الأوامر: تظهر جميع الأوامر التي تم استخدامها في جلسة العمل الحالية وجلسات العمل السابقة.

قائمة ماتلاب الرئيسية: تشبه هذه القائمة في طريقة عملها قائمة البدء في نظام التشغيل ويندوز حيث يتوضع زر أمر إظهارها في أسفل يسار النافذة الرئيسية وبالنقر عليه تظهر القائمة وهي تحوي أوامر تم تجميعها في قوائم فرعية للوصول إلى مجموعة كبيرة من أدوات ومكتبات النظام.

مجموعة القوائم المنسدلة: وتضم القوائم المنسدلة التالية:

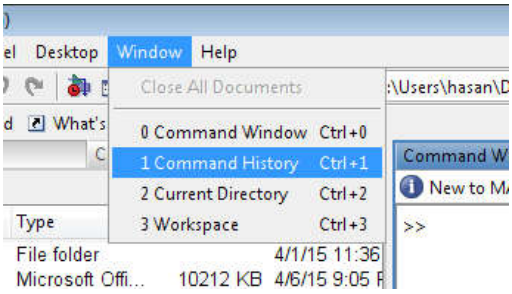
قائمة ملف File : تحوي أوامر إدارة الملفات كإنشاء ملف جديد وفتح ملف وإغلاق ملف وغير ذلك.

قائمة تحرير Edit : تحوي الأوامر المتعلقة بالنسخ والقص واللصق والاختيار وغير ذلك من أوامر التحرير.

قائمة تصحيح Debug : تحوي الأوامر المتعلقة بتعقب وتصحيح أخطاء البرامج المكتوبة في ماتلاب.

قائمة سطح المكتب Desktop : تحوي الأوامر الخاصة بضبط طريقة إظهار النوافذ الفرعية والنافذة الرئيسية ومنها يمكن تحديد ما يراد ظهوره من تلك العناصر ومالا يراد.

قائمة النوافذ Window : تظهر في هذه القائمة أسماء النوافذ المفتوحة حاليا ومنها يمكن التنقل بين تلك النوافذ بالنقر على أي منها، في حالة الوضع الافتراضي لماتلاب يكون شكل هذه القائمة كما هو مبين في الشكل 2



الشكل 2 قائمة النوافذ

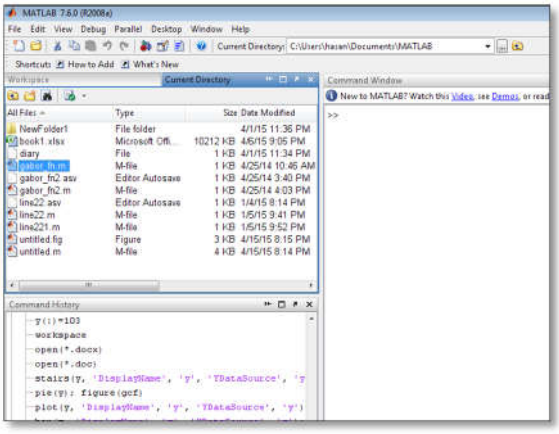
قائمة المساعدة Help : نجد فيها معلومات عن النظام والترخيص وأوامر خاصة بالحصول على المساعدة في الحصول على معلومات حول كيفية استخدام التعليمات والمكتبات.

شريط أدوات ماتلاب: يحوي اختصارات لبعض الأوامر كثيرة الاستخدام في النظام لتسهيل الوصول إليها من المستخدم.

◄ تغيير توضع النوافذ

يشبه ماتلاب الكثير من التطبيقات في طريقة العمل مع النوافذ فيه حيث يمكننا إظهار نوافذ وإخفاء أخرى وتغيير مكان توضع نافذة أو شكل ظهورها.

عند النظر إلى نافذة مساحة العمل ونافذة المسار الحالي نجد أنهما تشغلان نفس المساحة من النافذة الرئيسية ولكن إحداهما فقط تظهر والأخرى يظهر منها شريط عنوانها فقط وللتبديل بينهما يمكن النقر على شريط عنوان الأخرى (غير الظاهرة) لتصبح هي في الأعلى (ظاهرة) وتختفي الأولى التي كانت ظاهرة تحتها.



الشكل 3 نافذتان تشغلان المساحة ذاتها

لكل نافذة فرعية شريط عنوان يحوي عنوان النافذة ومجموعة من رموز الأوامر كما في الشكل 4 وهذه الأوامر هي:

أمر الإغلاق: لإغلاق النافذة، ولإعادة إظهارها يتم تفعيل ظهورها من قائمة سطح المكتب بوضع علامة الصح بجانب اسمها في تلك القائمة.

أمر إلغاء الالتصاق بمكان undock لتحرير توضع النافذة وجعلها تعوم فوق النافذة الرئيسية ليتمكن المستخدم من تغيير مكانها حيث يشاء وعند تفعيل هذه الوضعية يتغير هذا الأمر إلى الأمر تفعيل الالتصاق dock لإعادة النافذة إلى موضع محدد في النافذة الرئيسية.

أمر التكبير الأقصى: لجعل النافذة تملأ النافذة الرئيسية وعند تفعيله يتحول إلى الأمر استعادة لإعادتها إلى وضعها السابق.

أمر التصغير: لتصغير النافذة إلى زر أمر صغير يظهر إلى يسار بقية النوافذ ضمن النافذة الرئيسية ولإظهارها في هذه الحال يمكن النقر على زر الأمر ذاك ولكن عند الابتعاد عن النافذة تعود لحالة الاختصار أما لإعادتها إلى وضعها السابق فيمكن النقر على زر أمر استعادة الذي يظهر بدلا من أمر التصغير.

الشكل 4 شريط عنوان نافذة فرعية

هناك نوافذ أخرى في ماتلاب لا تظهر في الوضع الافتراضي ويمكن إظهار أي منها باختيارها من قائمة سطح المكتب.

◄ أدوات نظام ماتلاب

يعتبر نظام ماتلاب نظاما وليس مجرد تطبيق لأنه يحوي العديد من الأدوات التي لا يمكن ضمها ضمن سياق واحد كما هو الحال في التطبيقات الاعتيادية فهو يحوي بشكل رئيسي الأدوات التالية:

لغة برمجية تفسيرية (Interpreting Programming Language): حيث يمكن كتابة الأوامر البرمجية عالية المستوى في نافذة الأوامر لتنفيذها مباشرة دون تحويلها إلى ملف تنفيذي أو يمكن تخزين تلك الأوامر في ملف نصي ذو امتداد خاص بالماتلاب لتنفذها دفعة واحدة وهنا أيضا لا حاجة إلى عملية الترجمة (التحويل

إلى ملف تنفيذي). يمكن أيضا بناء تطبيق برمجي متكامل بكتابة مجموعة من تلك الملفات لكل منها وظيفة جزئية محددة وربطها ببعضها البعض. في كل الحالات السابقة لا بد ان يكون العمل ضمن بيئة ماتلاب أي أن استدعاء أي أمر لتنفيذه أو ملف ماتلاب أو تطبيق مؤلف من عدة ملفات ماتلاب لا بد أن يكون من ضمن بيئة النظام الذي يجب تنصيبه على جهاز المستخدم النهائي.

يمكن في حال الحاجة أيضا بناء تطبيق برمجي وتحويله إلى ملف تنفيذي قابل للعمل على جهاز لا يحوي ماتلاب وذلك بترجمته (بعد تجريبه وتنقيحه) بواسطة مترجم ما إلى تطبيق مستقل عن بيئة ماتلاب stand alone application.

أداة رسم بياني: في هذه الأداة يمكن رسم الخطوط والأشكال البيانية ثنائية وثلاثية الأبعاد بالإحداثيات الديكارتية أو القطبية وكذلك يمكن رسم مخططات ورسوم إحصائية كالمخططات التكرارية ومخططات الأعمدة الإحصائية وغير ذلك. يمكن أيضا رسم أشكال أكثر تعقيداً أو معالجة أشكال وصور تم أخذها بواسطة أجهزة اخرى كصور الأقمار الصناعية. كل ذلك يتم بواسطة أوامر وتعليمات معبأة في مكتبات خاصة.

أداة محاكاة: تسنخدم هذه الأداة الهامة في إنشاء النماذج الرقمية لأشياء كثيرة كالدارات الكهربائية والمحركات والآلات الصناعية الكبيرة بغية معاينتها بكلفة صغيرة جداً أو مهملة نسبياً مقارنة مع كلفة بناء نماذج حقيقية لتلك الأشياء.

◄ البدء في العمل

للبدء في العمل في نظام ماتلاب لا بد من التعرف على بعض الأوامر الأساسية في نافذة الأوامر على التوازي مع تبسيط المفاهيم المتعلقة بفلسفة العمل في النظام.

◄ نافذة الأوامر

هي النافذة الأهم في ماتلاب على الأقل بالنسبة للمبتدئين أو بالنسبة لمن يريدون إجراءات بسيطة نسبياً من النظام وهي تظهر في الوضع الافتراضي محتلة المساحة الأكبر من النافذة الرئيسية وكل ما فيها هو رمز المحث (موجه الأوامر) وهو رمز يدل على جاهزية نافذة الأوامر للكتابة فيها وتظهر بعده الوامضة للإشارة إلى موضع الكتابة.

يمكن كتابة أمر وتنفيذه بالضغط على مفتاح الإدخال Enter وعندها تظهر نتيجة التنفيذ على نافذة الأوامر نفسها أو غيرها حسب الأمر ففي حالة إسناد قيم المتحولات تظهر في نافذة الأوامر أما عند تنفيذ أمر رسم فيتم إظهار نافذة الرسم وعليها نيجة التنفيذ.

◄ المتحولات

المتحول هو مكان محجوز في الذاكرة الرئيسية (المؤقتة) ويتم تخزين قيمة ما فيه قد تكون هذه القيمة هي بيانات عددية أو نصية أو بنية معقدة من البيانات. ويتم إظهار معلومات عن المتحولات فيما يسمى بمساحة العمل `workspace` التي يمكن مشاهدتها من نافذة خاصة بها هي نافذة مساحة العمل.

لا حاجة لتعريف المتحولات في ماتلاب فهو يميز نوع المتحول من خلال القيمة التي يتم إسنادها فيه.

يجب استخدام أسماء للمتحولات بحيث تلبي بعض القواعد فالمتحول يجب أن يبدأ بحرف من أحرف الأبجدية الانكليزية كبيراً أو صغيراً ويجب الانتباه إلى أن ماتلاب حساس لحالة الأحرف فالمتحول `a` غير المتحول `A` ويمكن أن يحوي اسم المتحول بعد الحرف الأول أي عدد من الأحرف أو الأرقام ولا يمكن أن يحوي رموزا.

◄ المتحولات العددية

المتحولات العددية في ماتلاب هي مصفوفات دوما فهو يخزن أي قيمة بسيطة أو مركبة على شكل مصفوفة بالأبعاد المناسبة لتلك القيم التي تم إدخالها وهناك طرق عديدة لإدخال تلك القيم سوف نذكرها قريبا.

◄ الإدخال الأول

يمكنك كتابة الأمر a=23 في سطر الأوامر والضغط على مفتاح الإدخال لتنفيذه، سوف تظهر نتيجة التنفيذ في نفس النافذة كما هو مبين فيمايلي

```
>> a=23
a =
   23
>>
```

حيث يظهر اسم المتحول المستخدم متبوعا بالرمز = ثم قيمة المتحول على سطر جديد. بالنظر الآن إلى نافذة مساحة العمل نجد اسم المتحول a وقيمته 23 والقيمة الصغرى 23 والقيمة الكبرى 23

لا معنى الآن للقيمة الصغرى والقيمة الكبرى فالمتحول a هو قيمة بسيطة أما في حال إدخال مصفوفة في متحول فالقيمة الصغرى هي أصغر قيم تلك المصفوفة والكبرى هي أكبر قيمها وفي حال كون عدد القيم في المصفوفة كبيراً نسبيا بالنسبة للمساحة المتوفرة في نافذة مساحة العمل لا يتم إظهار القيم بل أبعاد المصفوفة.

يمكن إدخال قيم مصفوفة بين حاصرتين من الشكل [] كما في المثال التالي:

```
>> a=[1 4 -5]
```

```
a =
   1   4   -5
```

في هذا المثال تم إدخال عناصر متجه (شعاع) أفقي إلى المتحول a ويتم الفصل بين القيم بواسطة فراغات كما في المثال أو بواسطة فواصل عادية commas أما عند الحاجة للانتقال إلى سطر جديد في المصفوفة تستخدم الفاصلة المنقوطة semicolon كما في المثالين التاليين:

```
>> a=[1,4,5]
>> b=[1;4;-5]
>> c=[1,3,5;2,1,1,2]
```

حيث تم إدخال متجه أفقي في المتحول a ومتجه عمودي في المتحول b ومصفوفة من سطرين وثلاثة أعمدة في المتحول c

◄ الأمران who و whos

عند إسناد قيم ما إلى متحول مثل a ثم إعادة إسناد قيم جديدة إلى نفس المتحول يتم حذف القيم القديمة وإعادة استخدام الاسم نفسه دون إنذار من النظام، ولهذا يجب الانتباه إلى ذلك عند تسمية المتحولات. يمكن معرفة المتحولات المعرفة في مساحة

العمل بمعاينة نافذة مساحة العمل أو باستخدام أحد الأمرين who و whos حيث يظهر الأول أسماء المتحولات في نافذة الأوامر أما الثاني فيظهر أسماء المتحولات مع معلومات عنها كمايلي:

```
>> who
Your variables are:
a  b
>> whos
  Name      Size            Bytes  Class     Attributes
  a         2x3                48  double
  b         2x2                32  double
```

لرؤية محتوى متحول محدد يمكن كتابة اسمه في نافذة الأوامر والضغط على مفتاح الإدخال Enter.

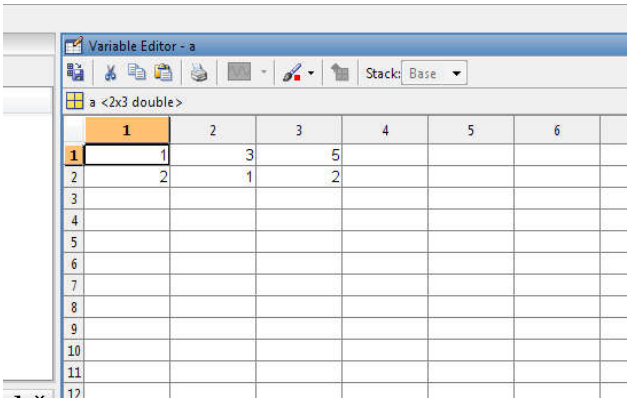
◄ الأمران clc و clear

يستخدم الأمر clc لمسح محتويات نافذة الأوامر وهو اختصار لـ clear commands أما الأمر clear فيستخدم لمسح محتويات مساحة العمل أي مسح جميع المتحولات من الذاكرة.

يمكن استخدام الأمر clear متبوعا باسم أحد المتحولات لمسح المتحول المذكور فقط من مساحة العمل.

◄ محرر المتحولات Variable Editor

يكون تغيير قيم متحول في بعض الحالات أسهل باستخدام نافذة محرر المتحولات بدلاً من نافذة الأوامر وعلى سبيل المثال عند كون المتحول a مصفوفة من سطرين وثلاثة أعمدة ونريد أن نجعلها ذات ثلاثة أسطر وخمسة أعمدة مع وضع القيمة 7 في العمود الخامس من السطر الثالث ويقية القيم الجديدة أصفار، يمكن إظهار المتحول a في نافذة تحرير المتحولات بالنقر المزدوج على رمزه في نافذة مساحة العمل فتظهر النافذة المطلوبة فوق نافذة الأوامر (تتم إزاحة الأخيرة إلى الأسفل لتفسح المجال أمام النافذة الجديدة) وتظهر كما في الشكل 5



الشكل 5 نافذة تحرير المتحولات

لإجراء التعديل المطلوب على المتحول يكفي الانتقال إلى الخلية الخامسة في السطر الثالث وكتابة القيمة 7 فيها ليتم تغيير أبعاد المصفوفة وتصفير بقية القيم في الخلايا الجديدة.

◄ الفاصلة والفاصلة المنقوطة

تستخدم الفاصلة العادية كما ذكرنا للفصل بين قيم السطر الواحد في متجه أو مصفوفة وتستخدم الفاصلة المنقوطة للفصل بين أسطر المصفوفة، ولكن هناك استخدامات أخرى لهاتين الفاصلتين خارج حاصرتي المصفوفة، حيث تستخدم المنقوطة في نهاية الأمر لمنع إظهار النتيجة على نافذة الأوامر وهذا جيد عندما لا يكون المستخدم بحاجة لرؤية تلك النتيجة بل بحاجة لرؤية مجموعة من الأوامر التي يدخلها تباعا، أما الفاصلة العادية فتستخدم في نهاية الأمر للتمكن من إتباعه بامر آخر قبل تنفيذه بالضغط على مفتاح الإدخال ثم تنفيذ الأمرين معا (في الواقع تباعا) بالضغط مرة واحدة على مفتاح الإدخال. يمكن أيضا الفصل بين الأوامر بالفاصلة المنقوطة كما في المثال التالي:

```
>> a=[1 2 4];
>> a=[1 2 4];b=23;
>> a=[1 2 4],b=23;
>> a=[1 2 4],b=23
```

في السطر الأول يتم التنفيذ دون إظهار أي نتائج بسبب الفاصلة المنقوطة في نهاية السطر، في السطر الثاني تم إتباع كل أمر بفاصلة منقوطة لمنع إظهار النتائج لكليهما أيضا، في السطر الثالث استخدمت المنقوطة في النهاية فقط والعادية بين الأمرين فتظهر النتائج للأمر الأول فقط أما في السطر الأخير فتظهر نتائج الأمرين حيث تم الفصل بينهما بالفاصلة العادية ولم تستخدم أي فاصلة في النهاية.

◄ سجل الأوامر diary

من المفيد جدا في بعض الحالات حفظ الأوامر المستخدمة في جلسة العمل في ملف نصي، يمكن عمل ذلك باستخدام الأمر `diary` متبوعا باسم الملف كاملاً مع اللاحقة والمسار ضمن إشارتي التنصيص المفردتين `single quotes` كمايلي:

```
Diary ‘d:/matlab_diary.txt’
Diary ‘matlab_diary.txt’
Diary ‘matlab_diary’
diary
```

يمكن الاستغناء عن مسار الملف في حال الرغبة بحفظه في مسار النظام (نظام ماتلاب)، ويمكن الاستغناء عن اللاحقة في حال الرغبة باستخدام ملف بدون لاحقة ويقوم ماتلاب بإنشاء ملف بالاسم المحدد في حال عدم وجوده وتفعيل خدمة حفظ

الأوامر فيه أما في حال وجود ملف بالاسم المحدد فيقوم ماتلاب بالإضافة إليه دون حذف محتوياته. في حال عدم تحديد اسم ملف يقوم ماتلاب بإنشاء ملف بالاسم diary في الدليل (المجلد) الحالي.

عند الحاجة إلى إيقاف التسجيل يمكن استخدام الأمر diary متبوعا بكلمة off وفي حال الرغبة باستئناف التسجيل تستخدم الكلمة on

للحصول على معلومات عن التسجيل يستخدم الأمر get كمايلي:

لمعرفة حالة عملية التسجيل:

get(0,'diary')

لمعرفة ملف التسجيل الحالي:

get(0,'diaryfile')

◄ التنقل بين الأوامر وإلغاء تنفيذ أمر

تستخدم مفاتيح الأسهم الشاقولية ↑ ↓ لإعادة كتابة أمر سابق حيث يتم إعادة كتابة آخر أمر باستخدام المفتاح ↑ ولإعادة كتابة الأمر السابق له يتم الضغط على ذات

المفتاح ثانية وهكذا أما للانتقال بشكل معاكس يستخدم المفتاح ↓ ولمسح أمر قبل تنفيذه يستخدم مفتاح الهروب Esc

في حال تنفيذ أمر طويل ودخول نظام ماتلاب في حالة عدم استجابة بسبب طول الأمر يمكن إيقاف تنفيذ الأمر بعد بدء تنفيذه بالضغط على Ctrl+c

◄ المتحول ans

يستخدم ملاتلاب المتحول الخاص ans لحفظ نتيجة العملية غير المسندة ويتم حجز مساحة له في مساحة العمل عند استخدامه للمرة الأولى في جلسة العمل، وشأنه شأن بقية المتحولات تتغير قيمته دون إنذار في حال استخدامه ثانية. في المثال التالي يتم إدخال مصفوفة عددية دون إسنادها إلى أي متحول وتكون النتيجة حفظها في المتحول ans ثم يسند محتوى هذا المتحول إلى المتحول newArray

```
>> [5,6 8]
ans =
   5   6   8
>> newArray=ans
newArray =
   5   6   8
```

◄ قراءة عنصر أو أكثر من مصفوفة

لنفرض أننا أدخلنا المصفوفة ثنائية الأبعاد c التالية:

```
c =
   3   5   7
   2   4   6
   1   0  11
```

يتم ترقيم عناصر هذه المصفوفة في ماتلاب بدءاً من العنصر الأول في السطر الأول نزولاً في العمود الأول حتى آخره ثم العمود الثاني وهكذا حتى العنصر الأخير في العمود الأخير كما هو موضح في الشكل 6.

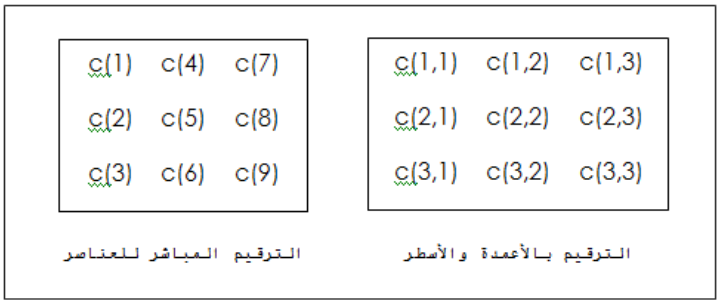
وعلى هذا يمكن الوصول إلى عناصر المصفوفة بكتابة اسم المصفوفة متبوعا برقم العنصر بين قوسين كمايلي:

```
>> c(1)
ans =
   3
>> c(5)
ans =
   4
```

يتم ترقيم عناصر المصفوفة أيضا بترقيم أسطرها وأعمدتها كما هو موضح في الشكل 6

وهكذا يمكن الوصول إلى العناصر بتحديد رقم السطر ورقم العمود للعنصر كمايلي:

```
>> c(1,1)
ans =
   3
>> c(2,3)
ans =
   6
```



الشكل 6 ترقيم عناصر المصفوفة

يمكن أيضا باستخدام ترقيم الأسطر والأعمدة الوصول إلى مجموعة من الأسطر أو الأعمدة من المصفوفة ويمكن دائما إسناد النتائج إلى متحولات جديدة كما في الأمثلة التالية:

إسناد السطر الأول من المصفوفة c إلى المصفوفة row1

```
>> row1=c(1,:)
row1 =
   3   5   7
```

إسناد السطرين الثاني والثالث من المصفوفة c إلى المصفوفة row23

```
>> row23=c(2:3,:)
row23 =
   2   4   6
   1   0  11
```

إسناد عناصر السطر الأول من المصفوفة c والواقعة في العمودين الثاني والثالث إلى المصفوفة cPartR1C23

```
>> cPartR1C23=c(1,2:3)
cPartR1C23 =
   5   7
```

حيث يستخدم المعامل : للتعبير عن المجال المطلوب من الأسطر أو الأعمدة وفي حال استخدامه بدون أرقام قبله وبعده فهو يعني أن كل الأسطر أو الأعمدة مطلوبة حسب موقعه ضمن القوسين.

◄ الدليل الخاص end

يستخدم الدليل الخاص `end` للوصول إلى العنصر الأخير في سياق محدد في متجه أو مصفوفة وهو مفيد عندما نريد الوصول إلى العنصر الأخير في مصفوفة أو سطر أو عمود أو غير ذلك دون أن يكون لدينا علم بعدد العناصر في السياق المحدد.

أمثلة:

```
>> x=magic(3)
x =
   8   1   6
   3   5   7
   4   9   2
>> x(end)
ans =
   2
>> x(end,end)
ans =
   2
>> x(end,2)
ans =
   9
```

◄ إعادة تعيين قيمة عنصر أو أكثر

تستخدم الطريقتان المذكورتان في الفقرة السابقة في ترقيم عناصر المصفوفة لإعادة تعيين عناصر في مواقع محددة من المصفوفة كما في الأمثلة التالية:

```
>> c(5)=111;
>> c(1,2)=112;
>> c(1,:)=[5 5 5];
>> c(:,1:2)=[10;10];
>> c(:,1:2)=[10 10;5 5;2 3];
```

يمكن أيضا إعادة تعيين بعض العناصر دفعة واحدة بذكر متجه الأدلة المطلوب تغيير العناصر المقابلة لها كمايلي:

```
>> x=magic(3)
x =
   8   1   6
   3   5   7
   4   9   2
>> x([2,4,7])=10
x =
   8  10  10
  10   5   7
```

```
   4   9   2
```

◄ حذف عناصر من المصفوفة

يتم حذف العناصر بإسناد مصفوفة فارغة إلى الجزء المراد حذفه من المصفوفة، ويمكن بهذه الطريقة حذف سطر أو عمود أو عدة أسطر أو أعمدة ولا يمكن حذف قيم مفردة لان ذلك يؤدي إلى نشوء فراغ في المصفوفة الناتجة لا يقبله نظام ماتلاب

أمثلة:

```
>> c(1,:)=[]
>> c(:,1)=[]
>> c(:,2:3)=[]
>> c(1,2)=[]
??? Subscripted assignment dimension mismatch.
```

في السطر الأخير نحصل على رسالة خطأ بسبب محاولة حذف قيمة مفردة.

يمكن أيضا حذف مجال من القيم باستخدام طريقة الترقيم المباشر للعناصر ولكن ماتلاب يقوم بإعادة تشكيل المصفوفة الناتجة وتحويلها إلى مصفوفة سطرية كما في المثال التالي:

```
c =
   3   10   1
```

```
   6   20    4
   9   30    7
>> c(3:6)=[]
c =
   3    6    1    4    7
```

وكما في طريقة إعادة التعيين باستخدام متجه الأدلة يمكن حذف عناصر مقابلة لمتجه أدلة مع الإنتباه هنا إلى أن حذف عناصر من أسطر أو أعمدة ربما يشوه شكل المصفوفة ولهذا فإن ماتلاب يعيد تشكيل المصفوفة إلى متجه أفقي كما في المثال التالي:

```
>> x=magic(3)
x =
   8    1    6
   3    5    7
   4    9    2
>> x([2,4,7])=[]
x =
   8    4    5    9    7    2
```

◄ تركيب مصفوفة من مصفوفات أخرى

في هذه الطريقة يتم إدخال أسماء مصفوفات في المصفوفة المراد تشكيلها بدلاً من الأرقام بشرط تحقيق أبعاد صحيحة للمصفوفة الناتجة ولتوضيح هذا المعنى يمكن ان نتأمل المثال التالي:

```
>> a=[1 3;4 2];
>> a1=[a a];
>> a2=[a ; a];
>> a3=[a1 a2];
```

في السطر الأول تم إدخال عناصر المصفوفة a المكونة من سطرين وعمودين، في السطر الثاني تم إدخال عناصر المصفوفة a1 المكونة من سطر واحد من العناصر وعمودين ولكن بدلاً من إدخال الأرقام تم إدخال اسم المصفوفة a في المصفوفة a1، للوهلة الأولى قد يظن بعض من لديهم خبرة في لغات برمجة أخرى أن هذه العملية سوف تؤدي إلى تشكيل مصفوفة من المصفوفات وفي الحقيقة فإن هذه تقنية أخرى سوف نراها لاحقاً، أما هنا فيتم توضيع عناصر a كما هي للحصول على مصفوفة عددية جديدة ناتجة عن اصطفاف تلك العناصر بالترتيب المحدد ففي حالة a1 تم الإصطفاف أفقياً وفي حالة a2 تم الإصطفاف شاقولياً أما في حالة a3 فلا يمكن إجراء الإصطفاف لعدم تطابق الأبعاد ويظهر ماتلاب رسالة خطأ.

نتائج تنفيذ السطرين الثاني والثالث ستكون كمايلي:

```
>> a1
a1 =
   1   3   1   3
   4   2   4   2
>> a2
a2 =
   1   3
   4   2
   1   3
   4   2
```

أمثلة أخرى:

```
>> a=[1 2 3];
>> b=[4 5 6 7];
>> c=[a 1;b];
>> d=[a;3 3 3];
>> e=[[1 1]];
>> f=[[[1 1]]];
>> g=[a;[1 2] 5;[5 6 7;8 9 9]];
```

نلاحظ أن هناك أساليب مختلفة لدمج المصفوفات والأرقام لتشكيل مصفوفة ناتجة عددية، ونلاحظ أن الأقواس الزائدة في إدخال المصفوفتين e و f لا تؤثر على النتيجة وذلك ليس بسبب إهمال ماتلاب لتلك الأقواس بل لأنه يعالجها كعملية دمج ففي حالة المصفوفة e تعني الأقواس الخارجية أن ماتلاب سوف يشكل مصفوفة من

المصفوفات الداخلية والتي هي هنا مصفوفة سطرية واحدة وكذلك الحال في حالة المصفوفة f حيث استخدمنا المصفوفة السطرية لوحدها لإنشاء أخرى جديدة وتلك الجديدة استخدمت ثانية لإنشاء النتيجة النهائية. هذا العمل لا يؤدي إلى نتائج مختلفة ولكنه يفسر طريقة عمل ماتلاب.

◄ إدخال مصفوفة عناصر ذات خطوة

يمكن إدخال سلسلة حسابية على شكل مصفوفة عددية بذكر القيمة الأولى والخطوة ثم القيمة الاخيرة بين قوسي مصفوفة وبحيث يفصل بين كل قيمتين المعامل : كما في الأمثلة التالية:

```
>> x=[1:3:10]
x =
   1   4   7   10
>> x=[1:3:11]
x =
   1   4   7   10
>> x=[1:-3:11]
x =
  Empty matrix: 1-by-0
```

في المثال الثاني تم استخدام الخطوة الموجبة 3 كما هو الحال في المثال الأول ولكن القيمة النهائية لم تظهر لأنها غير محتواة في تلك السلسلة وهذا يدل على أن ماتلاب يقوم بتوليد السلسلة انطلاقا من القيمة الابتدائية بإضافة قيمة الخطوة إلى كل حد حتى الوصول إلى أقصى حد يقل عن القيمة النهائية أي أننا نستطيع إعادة تسمية القيمة النهائية بقيمة التوقف لأنها (كما في المثال الثاني هنا) ليست بالضرورة قيمة نهائية.

في المثال الثالث استخدمنا خطوة سالبة وقيمة توقف أكبر من القيمة الابتدائية وهذا لم يؤدي إلى خطأ بل إلى مصفوفة فارغة.

ملاحظة: رغم كون المصفوفة فارغة فإن ماتلاب يفهمها كمصفوفة ذات سطر واحد بدون أعمدة.

في حال كون الخطوة مجهولة ونعلم عدد العناصر المطلوب والقيمة الأولى والقيمة الأخيرة يمكن استخدام الطريقة نفسها بعد حساب الخطوة ويمكن استخدام طريقة اخرى كمايلي:

```
>> x=linspace(1,10,4)
x =
   1   4   7  10
```

في هذه الحالة يستخدم التابع `linspace` لتوليد سلسلة من القيم بخطوة عددية ثابتة وتكون معاملات هذا التابع بالترتيب هي القيمة الابتدائية ثم القيمة النهائية ثم عدد

العناصر المطلوب ويقوم ماتلاب بحساب الخطوة. يمكن أيضا عدم إدخال عدد العناصر في الحالة الخاصة عندما يكون هذا العدد هو 100

هناك تابع آخر شبيه هو logspace يقوم يتوليد سلسلة من القيم بفواصل لوغاريتمية وله نفس المعاملات إلا أن المعامل الأول هو قوة العدد 10 التي تعطي القيمة الابتدائية والمعامل الأخير هو قوة العدد 10 التي تعطي القيمة النهائية وفي حال عدم ذكر المعامل الثالث (عدد العناصر) يتم توليد 50 عنصراً.

مثال:

```
>> c=logspace(2,5,4)
c =
     100     1000     10000     100000
```

◄ الأمران load و save

رأينا سابقاً أن ماتلاب يستخدم جزءاً من الذاكرة المؤقتة ودعوناه مساحة العمل وهذا الجزء يفقد جميع بياناته بانتهاء جلسة العمل الحالية التي تنتهي بإغلاق ماتلاب. في كثير من الحالات نحتاج إلى التوقف عن العمل وربما إغلاق البرنامج والعودة لمتابعة العمل لاحقا ربما في اليوم التالي وفي هذه الحالات الكثيرة لا نريد بالطبع أن نعود لتعريف جميع المتحولات وإعطائها قيما بل نريد المتابعة من حيث انتهينا، يقدم لنا ماتلاب نوعين من التسهيلات التي تمكننا من ذلك في النوع الأول نقوم بحفظ

المتحولات الموجودة في مساحة العمل إلى ملف بيانات ونستردها منه في الوقت الذي نريد لاحقاً أما في النوع الثاني فنقوم بحفظ التعليمات التي استخدمناها لتعريف وتعبئة المتحولات في ملف تعليمات ونقوم بتشغيله عند الحاجة إليه. في هذه الفقرة سوف نناقش التقنية الاولى باستخدام الأمرين save و load وفي الفقرة التالية سوف نناقش التقنية الثانية باستخدام ملفات ماتلاب m-files

يستخدم الأمر save لحفظ المتحولات الموجودة في مساحة العمل إلى ملف بيانات في الذاكرة الثانوية (أقراص التخزين) وهذا الأمر يأخذ الشكل العام التالي:

save filename content options

حيث:

Filename: اسم الملف المراد الحفظ فيه مع الامتداد والمسار الكامل. في حال عدم ذكر المسار يتم الحفظ في مجلد العمل الحالي وفي حال عدم ذكر الامتداد يستخدم الامتداد الافتراضي mat. وفي حال عدم ذكر اسم الملف يستخدم اسم الملف الافتراضي matlab ويتم إنشاؤه إن لم يكن موجوداً.

Content: لائحة بأسماء المتحولات المراد حفظها، في حال عدم ذكر أي متحولات يتم حفظ جميع المتحولات المتوفرة في مساحة العمل.

Options: خيارات الحفظ، أهم هذه الخيارات هو خيار الإضافة -append وخيار التنسيق -format

يستخدم خيار الإضافة -append لحفظ المتحولات المطلوبة في الملف المحدد دون حذف محتوياته القديمة حيث يعتبر حذف تلك المحتويات هو الإجراء الافتراضي. ويستخدم خيار التنسيق -format لتحديد تنسيق حفظ الملفات بواحدة من خمسة تنسيقات هي:

-ascii	يتم الحفظ بالتنسيق ascii بثمانية خانات.
-ascii -tab	يتم الحفظ بالتنسيق ascii بثمانية خانات مفصولة بعلامات الجدولة.
-ascii -double	يتم الحفظ بالتنسيق ascii بست عشرة خانة.
-ascii -double -tabs	يتم الحفظ بالتنسيق ascii بست عشرة خانة مفصولة بعلامات الجدولة.
-mat	يتم الحفظ بتنسيق بيانات رقمية وهو الخيار الافتراضي.

أمثلة:

```
>> save
>> save a b
>> save a b –ascii
>> save wrk.dat a b –ascii
>> save wrk.mat a*
>> save wrk.mat b -append
```

في الأمر الأول يتم حفظ جميع متحولات مساحة العمل إلى الملف الافتراضي matlab بالامتداد الافتراضي mat. إلى مجلد العمل الحالي بالتنسيق الافتراضي -mat

في الأمر الثاني تم تحديد الحفظ للمتحولين a و b فقط.

في الأمر الثالث تم تغيير التنسيق هنا إلى ascii-

في الأمر الرابع تم تغيير الملف إلى wrk.dat بدلاً من الافتراضي.

في الأمر الخامس يتم حفظ جميع المتحولات التي تبدأ أسماؤها بالحرف a

في جميع الأوامر السابقة يتم حذف محتوى الملف المحدد وحفظ المحتوى الجديد أما في الأمر الأخير فتتم إضافة المحتوى الجديد إلى القديم بسبب استخدام الخيار -append

لرؤية ما تم حفظه يمكن استخدام الأمر whos متبوعا بالخيار file- ثم اسم الملف المحفوظ شريطة أن يكون بالتنسيق mat- كما يلي:

```
>> save wrk.mat a*
>> whos -file wrk.mat
  Name      Size            Bytes  Class     Attributes
  a         2x2                32  double
  a1        1x5                40  double
  a2        3x4                96  double
```

وهذا مفيد عند الحاجة إلى استرداد جزء من الملف كما سنرى في الأمر `load`

يستخدم الأمر `load` بشكل معاكس للأمر `save` لتحميل مساحة العمل بجميع أو بعض المتحولات المخزنة في ملف بيانات بالشكل العام:

```
load filename X Y Z
```

حيث يتم ذكر اسم الملف ثم لائحة بأسماء المتحولات المراد تحميلها منه. في حالة عدم ذكر أسماء متحولات يتم تحميل كافة المتحولات إلى مساحة العمل، وفي حالة عدم ذكر اسم الملف يتم تحميل محتوى الملف الافتراضي بالامتداد الافتراضي `matlab.mat` من مجلد العمل الحالي إن كان موجوداً وفي حال عدم وجوده يظهر ماتلاب رسالة خطأ.

أمثلة توضيحية:

```
>> load
>> load wrk.mat
```

```
>> whos -file wrk.mat
  Name     Size          Bytes  Class     Attributes
  a        2x2              32  double
  a1       1x5              40  double
  a2       3x4              96  double
  b        3x3              72  double
>> load wrk a*
```

في الأمر الأول يتم تحميل كامل الملف `matlab.mat` إلى مساحة العمل، وفي الأمر الثاني يتم تحميل الملف `wrk.mat` أما الأمر الأخير فيتم فيه تحميل كافة المتحولات التي تبدأ بالحرف `a` (استخدمنا الأمر `whos -file wrk.mat` لمعرفة المتحولات المخزنة في الملف).

◄ ملفات ماتلاب m-files

ملفات ماتلاب أو سكريبتات ماتلاب هي ملفات نصية صرفة تحوي أوامر ماتلاب. يمكن حفظ هذه الملفات في الذاكرة الثانوية واستدعاؤها عند الحاجة إليها لتنفيذ ما فيها دفعة واحدة دون عناء إعادة كتابة التعليمات، وهذه الملفات مفيدة في كثير من الأشياء حيث يمكن بواسطتها بناء وظائف إضافية بسيطة أو معقدة ويمكن بناء تطبيقات كبيرة متعددة الوظائف، وفي هذه الفقرة سوف نناقش فقط تقنية إعادة تعريف وتعيين قيم المتحولات باستخدام ملفات ماتلاب `m-files`

لإنشاء ملف ماتلاب يمكن اختيار الأمر file → new →m-file بدءاً من النافذة الرئيسية للنظام ليظهر محرر الأوامر وهو محرر نصوص في جوهره إلا أنه يوفر إمكانات خاصة للتعامل مع الملفات التي تحوي تعليمات ماتلاب. يمكن أن نكتب هنا كمثال أولي بعض التعليمات لإنشاء بعض المتحولات كمايلي:

```
x=[3:3:15];
y=[5:5:45];
z=[x y];
```

بعد الإنتهاء يجب حفظ الملف ويمكن ذلك بالأمر file → save as من نافذة المحرر لتظهر نافذة حوار الحفظ حيث يمكننا فيها اختيار المسار والاسم فقط أما الامتداد فيجب أن يكون m.

قبل أن يصبح الملف جاهزا للاستخدام يجب إضافة المجلد الحاوي عليه إلى مسار ماتلاب حيث أن ماتلاب لا يمكنه تنفيذ ملف غير موجود في مسار عمله، ويقصد بمسار ماتلاب أو مسار عمل ماتلاب مجموعة المجلدات التي يقوم ماتلاب بالبحث فيها عند استدعاء ملف أوامر أو وظيفة ما. لرؤية المسار الحالي (مجموعة المجلدات) يكتب الأمر path مجرداً في نافذة الأوامر، أما لإضافة مجلد ما إلى المسار فيكتب الأمر كمايلي:

```
Path(path,'directory name')
```

وبالطبع يجب ذكر اسم المجلد المراد إضافته مع مساره الكامل ووضع هذه النص ضمن إشارتي اقتباس مفردتين.

بعد إضافة المجلد إلى مسار ماتلاب أصبح بالإمكان تنفيذ الملف بذكر اسمه فقط في نافذة الأوامر كمايلي:

```
>>xxc
```

وذلك بفرض أننا استخدمنا الاسم xxc للملف المثال.

تكمن الفائدة في هذه الطريقة في أننا نستطيع إغلاق جلسة العمل متى شئنا دون القلق بشان إعادة تعريف وتعيين قيم المتحولات فالعملية لا تتطلب سوى ذكر اسم الملف الحاوي على التعليمات اللازمة للقيام بذلك.

◄ المصفوفات متعددة الأبعاد

يوفر ماتلاب إمكانية تعريف مصفوفات ذات أكثر من بعدين وهناك طرق مختلفة لإنشاء تلك المصفوفات وتعيين عناصرها والوصول إليها. لنفرض أننا نريد إنشاء مصفوفة ذات ثلاثة أبعاد (أسطر وأعمدة وصفحات)، لعمل ذلك نقوم أولا بإنشاء الصفحة الأولى من المصفوفة كأي مصفوفة ثنائية البعد من أسطر وأعمدة ثم نضيف عناصر الصفحة الثانية والثالثة وهكذا كما في المثال التالي:

```
>> c=[1 2;3 4];
>> c(:,:,2)=[10 20;30 40]
c(:,:,1) =
   1   2
   3   4
c(:,:,2) =
  10  20
  30  40
```

في السطر الأول تم إدخال عناصر الصفحة الأولى كمصفوفة عادية ذات سطرين وعمودين وفي السطر الثاني تم تعيين عناصر الصفحة الثانية بنفس الأسلوب ولكن هنا لا يمكن الإسناد إلى اسم المصفوفة نفسه وإنما تجب الدلالة على أن العناصر المدخلة يراد تعيينها في الصفحة الثانية ولهذا يستخدم اسم المصفوفة مع قوسين يحويان معاملات تدل على ذلك `c(:,:,2)` المعامل الأول : يشير إلى الأسطر والمعامل الثاني : يشير إلى الأعمدة والمعامل الثالث 2 يشير إلى الصفحات وهو يعني هنا أننا نريد تعيين عناصر الصفحة الثانية بكل أسطرها وأعمدتها وكما نرى يعيد لنا ماتلاب المصفوفة الناتجة على مرحلتين واحدة لكل صفحة.

يمكن الوصول إلى عناصر المصفوفة ثلاثية الأبعاد بطريقتين كما في ثنائية الأبعاد، ففي الطريقة الأولى يستخدم الترقيم المباشر حيث يذكر اسم المصفوفة ورقم العنصر المراد الوصول إليه بين قوسين مع العلم أن الترقيم يبدأ بالعنصر الواقع في السطر الأول والعمود الأول والصفحة الأولى ثم يتم التحرك عموداً عموداً حتى انتهاء

الأعمدة في الصفحة الأولى ثم يتم التحرك بنفس الأسلوب في الصفحة الثانية وهكذا حتى الوصول إلى آخر عنصر في المصفوفة. في الطريقة الثانية يستخدم دليل السطر ودليل العمود ودليل الصفحة بين قوسين تاليين لاسم المصفوفة. الأمثلة التالية توضح كيفية استخدام هاتين الطريقتين:

```
>> c=[1 2;3 4];
>> c(:,:,2)=[10 20;30 40];
>> c(1) ans =1
>> c(5) ans =10
>> c(1,2,2)      ans = 20
>> c(2,2,1)      ans =4
```

يمكن أيضاً تعيين مصفوفات رباعية أو خماسية الأبعاد أو أكثر بنفس الأسلوب كما يوضح المثال التالي:

```
>> c=[1 2;3 4];
>> c(:,:,2)=[10 20;30 40];
>> c(:,:,:,2)=1
c(:,:,1,1) =
   1   2
   3   4
c(:,:,2,1) =
  10  20
  30  40
c(:,:,1,2) =
   1   1
```

```
   1   1
c(:,:,2,2) =
   1   1
   1   1
```

في السطر الثالث تم إسناد قيمة مفردة للمصفوفة المشكلة للصفحة الثانية في البعد الرابع وهذا صحيح حيث يقوم ماتلاب بجعل كل القيم مساوية لتلك القيمة.

◄ تنسيق إظهار القيم العددية للمصفوفات

عند إظهار القيم العددية على نافذة الأوامر نلاحظ تغييراً في كل مرة في شكل الأرقام وعدد الخانات المعنوية التي تظهر بعد الفاصلة العشرية، يعود هذا إلى تغير التنسيق المتبع في إظهار تلك القيم العددية حيث يقوم ماتلاب باختيار نمط التنسيق المناسب للأعداد التي يراد إظهارها حسب نوع تلك الأعداد. يمكن أيضا التحكم بالتنسيق من قبل المستخدم بواسطة الأمر `format` ذي الشكل العام:

Format type

حيث يدل المعامل `type` على نوع التنسيق المراد اتباعه في إظهار الأعداد، أما عند استخدام الأمر بدون معامل فهو يضبط طريقة التنسيق على الوضع الافتراضي وهو اختيار ماتلاب حسب نوع العدد أو المتحول المراد إظهره.

الجدول التالي يوضح القيم الممكنة للمعامل type لتغيير تنسيق إظهار الأعداد ذات الفاصلة العشرية:

قيمة المعامل	الوصف
Short	عدد ذو فاصلة عشرية ذو 4 خانات بعد الفاصلة.
Long	عدد ذو فاصلة عشرية ذو 14 أو 15 خانة بعد الفاصلة.
Short e	يظهر العدد هنا بأربع خانات بعد الفاصلة وبالشكل العلمي. مثال: 35.2501e+002
Long e	يظهر العدد هنا بـ 14 أو 15 خانة بعد الفاصلة وبالشكل العلمي.
Short g	يترك الخيار لماتلاب لاختيار الأفضل بين short و short e
Long g	يترك الخيار لماتلاب لاختيار الأفضل بين long و long e
Short eng	يظهر العدد هنا بأربع خانات بعد الفاصلة وبالشكل العلمي على ان تكون القوة من مضاعفات العدد 3 . مثال:

35.2501e+003	
يظهر العدد هنا بـ 16 خانة تحديداً بعد الفاصلة وبالشكل العلمي على ان تكون القوة من مضاعفات العدد 3	Long eng

مثال:

```
>> format short
>> x=34.567
x =
  34.5670
>> format long
>> x
x =
  34.567000000000000
>> x=34.567 e
??? x=34.567 e
          |
Error: Unexpected MATLAB expression.
```

مثال آخر:

```
>> format short e
>> x
x =3.4567e+001
```

```
>> format short eng
>> x
x =34.5670e+000
>> x*400
ans =13.8268e+003
```

لمعرفة التنسيق الحالي المتبع يمكن استخدام الأمر `get` كمايلي:

```
>> get(0,'format')
ans = shortEng
```

الجدول التالي يوضح القيم الممكنة للمعامل `type` لتغيير إظهار شكل الأرقام:

قيمة المعامل	الوصف
+	يظهر إشارة العدد فقط ويترك فراغاً في حال الصفر.
Bank	يظهر العدد برقمين فقط بعد الفاصلة.
Hex	يظهر العدد بالشكل السداسي عشري.
Rat	يظهر العدد بشكل نسبة بين عددين صحيحين. مثال: 345/548

أمثلة:

```
>> format +
>> x=[-2 3 0 4 0 0 0 -7]
x =-+ +  -
>> format bank
>> x=[3.2 1 304]
x = 3.20       1.00       304.00
>> format rat
>> x=[.5 17.4]
x = 1/2        87/5
```

هناك أيضا قيمتان أخريان للمعامل `type` هما `loose` و `compact` حيث تستخدم الأولى لجعل ماتلاب يظهر أسطر فارغة بين أسطر إظهار النتائج وفراغات زائدة قبل كل سطر لجعل الرؤية اوضح وأجمل وهي الحالة الافتراضية، أما الثانية فهي تستخدم لجعل ماتلاب يوفر تلك المساحات.

لمعرفة الوضع الحالي يمكن استخدام الأمر `get` كمايلي:

```
>> get(0,'formatspacing')
ans =
compact
```

العمليات على المصفوفات

◄ **العمليات الحسابية الأساسية على المصفوفات العددية**

هناك عمليات حسابية تجرى على المصفوفات العددية بشكل مصفوفي وأخرى بشكل غير مصفوفي بحيث تجرى العمليات بين كل عنصرين متقابلين. في مايلي نوضح العمليات الحسابية التي يمكن إجراؤها على المصفوفات:

الجمع والطرح

يعني جمع المصفوفات جمع كل عنصر من المصفوفة الأولى مع العنصر المقابل له من الثانية ولهذا يشترط كون المصفوفتين متساويتين في الأبعاد وكذلك الطرح هو طرح العناصر المتقابلة ويشترط فيه نفس الشرط. يرمز للجمع والطرح بالرمزين المشهورين + و – على التتالي.

الضرب

حالة الضرب مختلفة تماما فهناك حالتان، في الحالة الأولى يمكن ضرب كل عنصر من المصفوفة الأولى بالعنصر المقابل له في المصفوفة الثانية، وهذا الضرب الخطي يختلف عن الضرب المصفوفي لمصفوفتين حيث يتم ضرب العناصر بطريقة مختلفة وفق العلاقة التالية:

$$X_{r1,c1} \cdot Y_{r2,c2} = Z_{r1,c2}$$

$$z_{ij} = \sum_{k=1}^{c1} x_{ik} . y_{kj}$$

يشترط في الضرب المصفوفي أن يكون عدد أسطر المصفوفة الثانية يساوي عدد أعمدة الأولى وتكون المصفوفة الناتجة مساوية في الأسطر للأولى وفي الأعمدة للثانية.

مثال:

$$\begin{bmatrix} 1 & 2 & 5 \\ 3 & 9 & 8 \end{bmatrix} \cdot \begin{bmatrix} 4 & 1 \\ 5 & 0 \\ 8 & 2 \end{bmatrix} = \begin{bmatrix} 54 & 11 \\ 121 & 19 \end{bmatrix}$$

يرمز للضرب المصفوفي بالرمز * ويرمز للضرب اللامصفوفي بالرمز $\cdot$. حيث تشير النقطة السابقة لإشارة الضرب إلى أن العملية ليست مصفوفية.

أمثلة في ماتلاب:

```
>> x=[1 2 5;3 9 8];
>> y=[4 1 ;5 0;8 2];
>> x*y
ans =
   54   11
  121   19
>> x=[3 4 1 1 2];
>> y=[5 4 3 3 1];
>> x.*y
ans =
```

```
   15   16    3    3    2
```

القسمة اللامصفوفية اليمينية واليسارية

يقصد بالقسمة اللامصفوفية قسمة كل عنصر من المصفوفة الأولى على العنصر المقابل له من المصفوفة الثانية ويقصد بالقسمة اليسارية قسمة عناصر المصفوفة المذكورة ثانيا على المصفوفة المذكورة أولا واليمينية بالعكس. يرمز للقسمتين اللامصفوفيتين اليمينية واليسارية بالرمزين ./ و .\ على التتالي.

مثال:

```
>> x=[1 2 5;3 9 8];
>> y=[3 5 1;2 2 1];
>> x./y
ans =
   0.3333   0.4000   5.0000
   1.5000   4.5000   8.0000
>> x.\y
ans =
   3.0000   2.5000   0.2000
   0.6667   0.2222   0.1250
```

القسمة المصفوفية

عند تقسيم مصفومة على أخرى تنتج مصفوفة جديدة قابلة للضرب بالمصفوفة المقسوم عليها وتنتج المصفوفة المقسومة، ويمكن إجراء القسم اليمينية أو اليسارية عند تحقق الشرط.

الرفع إلى قوة

يتم رفع المصفوفة إلى قوة أو رفع عناصرها إلى قوة عنصراً عنصراً، وفي الحالة الأولى يشترط أن تكون المصفوفة مربعة لتحقيق شرط الضرب المصفوفي إذ أن الرفع إلى القوة 2 يعني ضرب المصفوفة بنفسها، وفي الحالة الثانية لا يشترط أي شرط إذ أن أي مصفوفة يمكن رفع كافة عناصرها إلى قوة واحدة.

يرمز للرفع إلى قوة للمصفوفة بالرمز ^ ولرفع عناصرها إلى قوة بالرمز .^ والمثال التالي يوضح رفع مصفوفة مربعة إلى القوة 2 في الخطوة الأولى ورفع عناصرها إلى القوة 2 في الخطوة الثانية.

```
>> c=[1 2;3 4]
c =
   1   2
   3   4
>> c^2
ans =
```

```
   7   10
  15   22
>> c.^2
ans =
   1    4
   9   16
```

◄ العمليات بين المصفوفات والأعداد المفردة

يمكن إجراء عمليات حسابية بين المصفوفات والأعداد المفردة حيث إن الأعداد المفردة في حقيقتها من وجهة نظر ماتلاب هي مصفوفات وحيدة السطر والعمود. ومن هذه العمليات ضرب جميع عناصر المصفوفة بعدد أو قسمتها عليه أو قسمة العدد على عناصر المصفوفة أو الجمع أو الطرح، ولكن يجب الانتباه إلى القسمة اليمينية واليسارية ورمز العملية ومعناها. الأمثلة التالية توضح المقصود:

```
>> c=[15 12;18 21];
>> c/3
ans =
   5    4
   6    7
>> c\3
??? Error using ==> mldivide
Matrix dimensions must agree.
>> c.\3
ans =
   0.2000   0.2500
```

0.1667 0.1429

في البداية تم إدخال المصفوفة c ذات السطرين والعمودين، في العملية الأولى تتم قسمة كل عنصر من عناصر المصفوفة c على العدد 3 وتنتج مصفوفة جديدة لها نفس أبعاد c، في العملية الثانية يراد تقسيم العدد 3 على المصفوفة c وهذا غير ممكن أما في العملية الثالثة فيتم تقسيم العدد 3 على كل عنصر من عناصر c

◄ الثوابت والقيم الخاصة المعرفة مسبقا

يوجد في ماتلاب قيم خاصة وثوابت معرفة مسبقا بحيث يمكن استخدامها مباشرة وهي موضحة فيمايلي:

يستخدم الثابت الخاص pi للتعبير عن العدد π وهو يعيد القيمة 3.1416 في الوضع الافتراضي.

يمكن استخدام هذه المتحولات لإسناد قيم أخرى ولذلك يجب الانتباه.

المثال التالي يوضح كيفية استخدام المتحول pi المعرف مسبقا ثم طريقة استعادة قيمته.

>> pi
ans =

```
   3.1416
>> pi=5
pi =
   5
>> clear pi
>> pi
ans =
   3.1416
```

حيث تم استخدام الأمر `clear` لحذف المتحول من الذاكرة.

القيمتان الخاصتان `i` و `j` تستخدمان للدلالة على العدد التخيلي $\sqrt{-1}$ وعند كتابة إحداهما فإن ماتلاب يعيد عدداً عقديا (مركباً) قسمه الحقيقي يساوي الصفر والتخيلي يساوي الواحد كمايلي:

```
>> i
ans =
     0 + 1.0000i
>> j
ans =
     0 + 1.0000i
```

العدد الكبير جداً والعدد الصغير جداً يتم التعبير عنهما بالرمزين ∞ و ∞- على الترتيب وفي ماتلاب يستخدم المتحول Inf للتعبير عن الـ ∞ والمتحول Inf ذاته مسبوقا بإشارة السالب (-) للتعبير عن الـ ∞-

يعيد ماتلاب القيمة Inf عندما تؤول نتيجة الحساب إلى عدد أكبر من قدرة ماتلاب على الحفظ في الذاكرة والنتيجة Inf- عند كون النتيجة أصغر مما يستطيع ماتلاب حفظه، هذه النتائج يمكن أن تحدث نتيجة عمليات كالتالي على سبيل المثال:

```
>> 1/0
ans =
  Inf
>> exp(Inf)
ans =
  Inf
>> 2^1000^1000
ans =
  Inf
```

القيمة الخاصة NaN تستخدم للدلالة على حالة عدم تعيين، أي أن ماتلاب لا يمكنه معرفة أو حساب القيمة وحالات عدم التعيين المعروفة تنتج عن محاولة إجراء قسمة أو طرح بين الأعداد الكبيرة جداً أو الصغيرة جداً أو محاولة تقسيم الصفر على الصفر أو ضرب الصفر باللانهاية.

$0 \div 0$ $\infty - \infty$ $\infty \times 0$ $\infty \div \infty$

في ماتلاب يمكن الحصول على هذه القيمة بإجراء عمليات كهذه:

```
>> (1/0)/(5/0)
ans =
  NaN
>> exp(Inf)/Inf
ans =
  NaN
>> (3-3)/(5-5)
ans =
  NaN
```

القيمة ans تكلمنا عنها سابقاً حيث أشرنا إلى أن المتحول ans يحوي نتيجة آخر عملية تم إجراؤها بدون إسناد.

القيمة الخاصة computer تستخدم لمعرفة معلومات حول الكمبيوتر الذي يعمل عليه ماتلاب كمايلي:

```
>> [s,m]=computer
s =
PCWIN
m =
```

2.1475e+009

حيث يتم إسناد وصف الحاسب إلى المتحول s والعدد الأعظمي للعناصر الممكنة في مصفوفة ما إلى المتحول m

القيمة التي حصلنا عليها هنا PCWIN تعني أن الحاسب من النوع الشخصي PC وعليه نظام تشغيل ويندوز وهو من فئة 32bit (القيمة PCWIN64 تشير إلى حاسب شبيه بـ 64bit)

للحصول على معلومات حول إصدار ماتلاب المثبت تستخدم القيمة الخاصة version كما في الأمثلة التالية:

>> version
ans =7.6.0.324 (R2008a)
>> version -release
ans =2008a
>> version -date
ans =February 10, 2008
>> version -java
ans =Java 1.6.0 with Sun Microsystems Inc. Java HotSpot(TM) Client VM mixed mode

استعملنا في الأمثلة السابقة عدة بارامترات إضافية كل منها يعطي معلومة مختلفة عند العمل بالتابع release

يستخدم التابعان intmax و intmin للحصول على القيمة العظمى والقيمة الدنيا الممكن استخدامهما في متحول من النوع المحدد كمايلي:

Intmax(classname)
Intmin(classname)

حيث أن القيمة النصية classname هي إحدى الأنواع المستخدمة في تعريف الأعداد الصحيحة وهي المبينة في الجدول التالي:

int8	int16	int32	int64
uint8	uint16	uint32	uint64

بشكل مشابه يستخدم التابعان realmax و realmin للحصول على القيم العظمى والدنيا الممكن تخزينها في متحولات من النوع double أو single كمايلي:

Realmax(classname)
Realmin(classname)

حيث يأخذ البارامتر classname هنا إحدى القيمتين double أو single

الأمثلة التالية توضح استخدام التوابع الأربعة السابقة الذكر

```
>> intmax('int8')
ans =127
>> intmax('int16')
ans =32767
>> intmax('uint16')
ans =65535
>> realmax('double')
ans =1.7977e+308
```

تجدر الإشارة إلى أن القيمة الافتراضية لنوع المعطيات في التابعين الأولين هي int32 وللتابعين الآخرين هي double فالكتابة intmax('int32') تكافئ الكتابة intmax والكتابة realmin('double') تكافئ الكتابة realmin ونحصل على النتائج ذاتها.

◄ عمليات المقارنة والعمليات المنطقية الأساسية

يستخدم ماتلاب القيمة العددية 1 للتعبير عن القيمة المنطقية true والقيمة العددية 0 للتعبير عن القيمة المنطقية false أما في حالة وجود قيمة ما غير هاتين القيمتين وأردنا اختبارها منطقيا فإنها تؤول إلى القيمة المنطيقة false فقط في حالة الصفر.

تنتج عمليات المقارنة بين مصفوفتين مصفوفة جديدة لها نفس أبعاد المصفوفتين الخاضعتين للمقارنة، وتكون قيم تلك المصفوفة الناتجة واحدات في مواضع تحقق

شرط المقارنة وأصفار في مواضع عدم التحقق. للمقارنة يمكن اختبار تساوي عنصرين أو عدم تساويهما أو كون أحدهما أكبر أو أصغر من الثاني أو لا يساويه كما يبين الجدول التالي:

رمز العملية	وصفها
<	أصغر من
<=	أصغر من أو يساوي
>	أكبر من
>=	أكبر من أو يساوي
==	يساوي
~=	لا يساوي

مثال:

```
>> x=[3 4 4 5 3 2];
>> y=[3 3 4 4 5 2];
>> x==y
ans =
```

```
   1   0   1   0   0   1
>> x~=y
ans =
   0   1   0   1   1   0
>> x>=y
ans =
   1   1   1   1   0   1
```

يمكن أيضا مقارنة مصفوفة بعدد مفرد وهذا يؤدي إلى مقارنة كل عنصر من عناصرها بذلك العدد وإخراج مصفوفة واحدات وأصفار بحجم المصفوفة الداخلة في المقارنة. ومن الهام الانتباه إلى وجود فرق بين عملية الإسناد = وعملية المقارنة ==

نقول عن التعبير الذي تتم الإجابة عليه بإحدى القيمتين المنطقيتين بأنه تعبير منطقي ويمكن إجراء عمليات على التعابير المنطقية البسيطة لتركيب تعابير منطقية مركبة أو معقدة بواسطة أدوات الربط المنطقي وللتذكير بهذه الأدوات نوردها في الجدول التالي:

الأداة	عملها
a & b	تعيد مصفوفة بحجم a أو b تحوي واحدات في المواضع الموافقة لقيم منطقية true (غير الصفر) في كلا المصفوفتين معا وأصفار في بقية المواضع.

a | b	تعيد مصفوفة بحجم a أو b تحوي واحدات في المواضع الموافقة لقيم منطقية true (غير الصفر) في إحدى المصفوفتين على الأقل وأصفار في بقية المواضع.
~a	تعيد مصفوفة بحجم a تحوي أصفاراً في المواضع الموافقة لقيم منطقية true (غير الصفر) في كلا المصفوفة a.

أمثلة

```
>> x=[1 2 3;4 3 0; 0 1 1];
>> y=[3 3 1;1 1 3;4 3 2];
>> x&y
ans =
   1   1   1
   1   1   0
   0   1   1
>> x|y
ans =
   1   1   1
   1   1   1
   1   1   1
>> x==3&y<=2
ans =
   0   0   1
   0   1   0
```

0 0 0

◄ طرق أخرى في توليد المصفوفات

ناقشنا سابقاً كيفية إنشاء مصفوفات عددية بالإدخال المباشر للقيم فيها أو بأخذ أجزاء من مصفوفة لإنشاء أخرى، هناك طرق أخرى تعتمد على توابع ماتلاب جاهزة أعدت مسبقا لإنشاء مصفوفات خاصة، وفيمايلي أهم هذه التوابع:

تابع توليد مصفوفة واحدات:

يقوم بتوليد مصفوفة جميع عناصرها واحدات بالأبعاد المعطاة له كبارامترات ويمكن أيضا تمرير بارامتر آخر يدل على نوع البيانات للقيم العددية وهكذا يأخذ التابع أحد الأشكال التالية:

Ones(m,n,…, classname)
Ones([m n …],clasname)

حيث تعبر القيم m و n والقيم التالية لها عن أبعاد المصفوفة المطلوبة أما القيمة الاختيارية classname فتعبر عن نوع المعطيات للقيم العددية الناتجة وفي حال غيابها يتم توليد مصفوفة بنوع المعطيات الافتراضي الذي هو عدد حقيقي ذو دقة

مضاعفة double، يمكن أن تكون أنواع المعطيات في هذا البارامتر إحدى القيم المبينة في الجدول التالي:

نوع المعطيات	قيمة البارامتر
عدد حقيقي ذو دقة مضاعفة	Double
عدد حقيقي	Single
عدد صحيح بثمانية خانات	Int8
عدد صحيح بثمانية خانات بدون إشارة	Uint8
عدد صحيح بست عشرة خانة	Int16
عدد صحيح بست عشرة خانة بدون إشارة	Uint16
عدد صحيح باثنتين وثلاثين خانة	Int32
عدد صحيح باثنتين وثلاثين خانة بدون إشارة	Uint32
عدد صحيح بأربع وستين خانة	Int64
عدد صحيح بأربع وستين خانة بدون إشارة	Uint64

أمثلة:

```
>> x=ones(2,3)
x =
   1   1   1
   1   1   1
>> x=ones(2,3,'int32')
x =
        1        1        1
        1        1        1
>> x=ones(2,3,2)
x(:,:,1) =
   1   1   1
   1   1   1
x(:,:,2) =
   1   1   1
   1   1   1
```

في حال تم تمرير قيمة واحدة للأبعاد تعتبر المصفوفة المطلوبة مربعة ولذلك فإن كتابة `ones(3)` تكافئ كتابة `ones(3,3)`

نلاحظ أيضا إمكانية كتابة البارامترات كأعداد منفصلة أو كمتجه من الأعداد فالكتابة `ones(3,4,2)` تكافئ الكتابة `ones([3 4 2])` ولا فرق إطلاقا في النتيجة الظاهرة.

لمعرفة نوع المعطيات لمصفوفة في مساحة العمل يمكن استخدام التابع `class(matrix name)` كمايلي:

```
>> class(x)
ans = double
>> class(y)
ans = int32
```

يجب الانتباه أيضا إلى النتائج الممكن حصول عليها عند إجراء العمليات الحسابية على مصفوفات من النوع الصحيح ولمعرفة المقصود يمكن إمعان النظر في المثالين التاليين:

المثال الأول:

```
>> x=ones(3);y=ones(3,'int32');
>> o.5*x
>> x*.5
ans =
   0.5000   0.5000   0.5000
   0.5000   0.5000   0.5000
   0.5000   0.5000   0.5000
>> y*.5
ans =
        1        1        1
        1        1        1
        1        1        1
```

في هذا المثال تم ضرب المصفوفة x ذات النوع الحقيقي المضاعف بالعدد 0.5 والنتائج هي مصفوفة من الـ 0.5 أما عند ضرب المصفوفة y بالعدد 0.5 فالنتائج هي مصفوفة من الواحدات لأن y من النوع الصحيح، يقوم ماتلاب بالحساب وتقريب النتائج إلى أقرب عدد صحيح دون تحذير وعلى المستخدم أن يكون واعياً ومدركاً لما يفعل.

المثال الثاني:

```
>> x=ones(3,'int8')
x =
   1   1   1
   1   1   1
   1   1   1
>> x*256
ans =
  127  127  127
  127  127  127
  127  127  127
```

في هذا المثال عرفنا المصفوفة x كمصفوفة واحدات من النوع الصحيح بثمانية خانات وهذا يعني أن مجال القيم المتاحة هو (-128 : 127) وعندما ضربنا

المصفوفة x بالعدد 256 حصلنا على مصفوفة من الـ 127 حيث لا يمكن الوصول لقيم أكبر في هذا النوع.

تابع توليد مصفوفةأصفار:

يولد هذا التابع مصفوفة جميع عناصرها أصفار ويأخذ أحد الشكلين المشابهين لشكل تابع مصفوفة الواحدات:

zeros(m,n,…, classname)
zeros ([m n …],clasname)

تصح هنا كافة المعلومات المذكورة حول التابع السابق.

تابع توليد المصفوفة الأحادية:

يولد مصفوفة جميع عناصرها أصفار عدا قيم القطر الرئيسي فهي واحدات وهو يشبه في الشكل العام التابعين السابقين عدا أنه يقبل بعدين فقط كمايلي:

eye(m,n, classname)
eye ([m n],clasname)

أمثلة:

```
>> eye(2)
ans =
   1   0
   0   1
>> eye(2,3)
ans =
   1   0   0
   0   1   0
>> eye(3,2)
ans =
   1   0
   0   1
   0   0
```

لاحظ القطر الرئيسي غير المكتمل للمصفوفات المستطيلة.

مصفوفة اللانهايات ومصفوفة اللاأعداد:

يمكن توليد مصفوفة من القيم Inf باستخدام التابع Inf(dimensions) ويمكن توليد مصفوفة من القيم NaN باستخدام التابع NaN(dimensions) كمايلي:

```
>> inf(2)
ans =
  Inf  Inf
  Inf  Inf
>> nan(1,4)
```

```
ans =
  NaN  NaN  NaN  NaN
>> nan(2)
ans =
  NaN  NaN
  NaN  NaN
>> -inf(2,3)
ans =
 -Inf -Inf -Inf
 -Inf -Inf -Inf
```

◄ الحصول على معلومات حول المصفوفات

للمصفوفات خصائص يمكن الحصول عليها باستخدام توابع معلومات المصفوفات لاستخدامها في عمليات أخرى، وفيما يلي أهم المعلومات التي يمكن الحصول عليها مع الأمثلة التوضيحية:

حجم المصفوفة:

يستخدم التابع `size` لمعرفة أبعاد المصفوفة وهو يعيد متجها أفقيا يحوي الأعداد الدالة على الأبعاد بالترتيب (البعد الاول ثم الثاني فالثالث وهكذا) ويختلف طول هذا المتجه باختلاف عدد الأبعاد وأقل قيمة له هي 2 حيث يدل على عدد الأسطر وعدد الأعمدة.

مثال:

```
>> x=[1 2 3];
>> size(x)
ans =
   1   3
```

يمكن استخدامه أيضا بطريقة أخرى للحصول على قيم منفصلة كمايلي:

```
>> [m,n]=size(x)
m =
   1
n =
   3
```

في حال تم إدخال أسماء متحولات أكبر من عدد الأبعاد يتم إسناد القيمة 1 للأبعاد الأخرى ففي الواقع تكون المصفوفة التي نقول عنها ذات بعدين (أسطر وأعمدة) مصفوفة ذات ثلاثة أبعاد عمقها يساوي الواحد وهكذا بقية الأبعاد موجودة ولكنها مساوية للواحد. أما إذا تم إدخال أسماء متحولات أقل فنميز حالتين، في الحالة الأولى عند استخدام اسم متحول وحيد فإن ماتلاب سيفهم أن هذا المتحول هو اسم للمتجه الذي يجب أن يحوي قيم الأبعاد ويعيد متجها في ذلك المتحول، أما في الحالة الثانية عندما يكون عدد أسماء المتحولات أكبر من الواحد وأقل من اللازم فيتم إسناد عدد العناصر في البعد الأول إلى المتحول الأول و عدد العناصر في البعد الثاني إلى

المتحول الثاني وهكذا وجداء عدد عناصر بقية الأبعاد إلى المتحول الأخير. الأمثلة التالية توضح الاستخدام والنتيجة.

```
>> v=[1 2 3];
>> [m,n,d3,d4]=size(v)
m =
   1
n =
   3
d3 =
   1
d4 =
     1

>> v=[1 2 3];
>> [m]=size(v)
m =
     1     3

>> x=[1 2 3];
>> x(:,:,2)=ones(size(x));
>> [m,n]=size(x)
m =
   1
n =
     6
```

طول المصفوفة:

يستخدم التابع `length` لمعرفة قيمة أطول أبعاد المصفوفة وهو يعيد قيمة واحدة كمايلي:

```
>> x=[1 1 2 3 1 4 5;3 3 2 4 8 8 7];
>> length(x)
ans =
   7
```

في المصفوفة وحيدة الاتجاه (المتجه الأفقي أو العمودي) تكافئ قيمة الطول عدد العناصر في المتجه.

نوع معطيات المصفوفة:

لمعرفة نوع المعطيات المستخدم لتخزين عناصر المصفوفة يستخدم التابع `class` كمايلي:

```
>> class(x)
ans =
double
```

القيم العظمى:

يمكن الحصول على القيم العظمى على مستوى كافة المصفوفة أو في اتجاه أحد الأبعاد باستخدام التابع max كمايلي:

```
>> x=magic(3);
>> c=max(x)
>> [c,i]=max(x,[],1)
>> [c,i]=max(x,[],2)
>> [c,i]=max(x,[],3)
```

في السطر الثاني تم استخدام التابع max لإعادة القيم العظمى في كل عمود من المصفوفة x وهو يعيد متجهاً أفقيا يحوي تلك القيم العظمى في المواضع الموافقة لمواضع أعمدة x

في السطر الثالث تم إجراء تعديلين على الأمر، أولهما إسناد النتيجة إلى متحولين c و i حيث يستخدم هنا المتحول المذكور أولاً لاحتواء القيم العظمى والمتحول المذكور ثانيا لاحتواء أدلة الأسطر التي تنتمي إليها تلك القيم العظمى، أما التعديل الثاني فهو إضافة الرمز [] كبارامتر ثاني وهذا البارامتر له وظيفة خاصة سنراها بعد قليل والعدد 1 كبارامتر ثالث، يعمل البارامتر الثالث للدلالة على البعد المراد إيجاد القيم العظمى وفقه وهو هنا البعد الأول أي أن التابع max يجب أن يعيد القيمة العظمى لكل عمود في اتجاه الأسطر (يقارن بين القيم التي تنتمي إلى أسطر مختلفة)

وهذا التعديل لا يؤثر هنا على النتيجة لأن اتجاه الأسطر هو الاتجاه الافتراضي للأمر.

في السطر الرابع تم استخدام الأمر بنفس أسلوب السطر الثالث ولكن هنا تم استخدام البعد الثاني وهو اتجاه الأعمدة ولهذا تكون النتيجة متجهاً عمودياً يحوي القيم العظمى في الأعمدة المتباينة ومتجهاً عمودياً مساوياً له في الطول يحوي أدلة أعمدة تلك القيم.

في السطر الأخير تم استخدام البعد الثالث (العمق) وهذا يعني مقارنة العناصر في اتجاه عمق المصفوفة مما يؤدي إلى الحصول على النتيجة كمصفوفة ثنائية البعد كل عنصر من عناصرها هو القيمة العظمى الممكن الحصول عليها من إحدى المصفوفات ثنائية البعد المشكلة للمصفوفة الكلية ويتم تخزين أدلة العمق الحاوي على القيم العظمى في مصفوفة ثنائية البعد أيضاً. في مثالنا هذا يتم إرجاع المصفوفة `x` نفسها كنيجة ومصفوفة واحدية مماثلة لها في الأبعاد كمصفوفة أدلة عمق لأن المصفوفة `x` ثنائية البعد فقط (عمقها يساوي الواحد).

يتم استخدام التابع `max` بطريقة أخرى للحصول على القيمة العظمى بين مصفوفتين بإدخال اسمي المصفوفتين كبارامترات بشرط كونهما متساويتي الأبعاد وتكون النتيجة مصفوفة مساوية لهما في الأبعاد تحوي القيم العظمى من إحدى المصفوفتين كما في المثال التالي:

```
>> x=magic(3);
>> y=5*ones(3);
```

```
>> c=max(x,y)
c =
   8   5   6
   5   5   7
   5   9   5
```

هذه الطريقة في الاستخدام تفسر استخدام الرمز [] كبارامتر ثاني في الطريقة السابقة حيث يشير هذا الرمز إلى أن العدد المدخل في البارامتر الثالث هو رقم البعد المطلوب العمل في اتجاهه، وفي غياب هذا الرمز الخاص سيقوم ماتلاب باعتبار ذلك العدد مصفوفة مساوية في الأبعاد للمصفوفة المدخلة ويستخدم التابع بالطريقة الأخيرة كما في المثال التالي:

```
>> x=magic(3);
>> c=max(x,3)
c =
   8   3   6
   3   5   7
   4   9   3
```

في هذه الطريقة أيضاً لا يمكن استخدام التابع max لإيجاد القيم العظمى بين أكثر من مصفوفتين بشكل مباشر ولكن يمكن استخدامه لهذا الغرض بالتحايل كما في المثالين التاليين:

```
>> x=magic(3);
>> y=5*ones(3);
>> z=3*ones(3);
>> c=max(x,max(y,z))
c =
   8   5   6
   5   5   7
   5   9   5
>> k=magic(3)-2;
>> c=max(max(x,y),max(z,k))
c =
   8   5   6
   5   5   7
   5   9   5
```

في السطر الرابع تم استخدام التابع max للمقارنة بين المصفوفتين x و y ثم تم استخدامه مرة ثانية لمقارنة النتيجة مع المصفوفة z

في السطر العاشر تم استخدامه لمقارنة المصفوفتين x و y ثم ثانية للمقارنة بين المصفوفتين z و k وثالثة للمقارنة بين النتيجتين.

تجدر الإشارة أيضاً إلى أنه من الممكن إيجاد القيمة العظمى الكلية في مصفوفة مهما كان عدد أبعادها باستخدام التابع max عدة مرات متتالية بعدد أبعاد المصفوفة والمثال التالي يبين هذه التقنية:

```
>> x=magic(3);
>> c=max(max(x))
c = 9
>> x(:,:,2)=3*ones(3);
>> c=max(max(max(x)))
c = 9
```

القيم الصغرى:

يستخدم التابع min للحصول على القيم العظمى بأسلوب مماثل تماما لأسلوب استخدام التابع max إلا أنه يعيد القيم الصغرى بدلاً من العظمى.

المجاميع:

يستخدم التابع sum لإيجاد المجاميع في أحد اتجاهات أبعاد المصفوفة حيث يمرر له اسم المصفوفة ورقم البعد المطلوب حساب المجاميع في اتجاهه كما يوضح المثال التالي:

```
>> x=magic(3);
>> s=sum(x)
s =
   15   15   15
>> s=sum(x,2)
s =
```

```
   15
   15
   15
>> s=sum(x,3)
s =
   8   1   6
   3   5   7
   4   9   2
```

لم يتم ذكر رقم البعد في السطر الثاني باعتبار ان البعد الأول (اتجاه الأسطر) هو الافتراضي أما في السطر الخامس فيراد حساب مجاميع الأسطر (باتجاه الأعمدة) ولذلك تم ذكر رقم البعد 2 والتابع هنا يعيد متجها عموديا بدلا من الأفقي في الحالة الأولى وفي السطر العاشر تم إدخال البعد الثالث للجمع باتجاه العمق والتابع هنا يعيد ذات المصفوفة لأن عمقها يساوي الواحد.

المضاريب:

يستخدم التابع `prod` لايجاد المضاريب في أحد اتجاهات المصفوفة بنفس الأسلوب المتبع في إيجاد المجاميع.

المجاميع التراكمية:

يستخدم التابع cumsum لإيجاد المجاميع التراكمية بدلاً من المجاميع، وهو يعيد مصفوفة لها نفس أبعاد المصفوفة المدخلة ولكنها تحوي المجاميع التراكمية في أحد الاتجاهات وعلى سبيل المثال إذا كان الاتجاه المطلوب هو الاتجاه الأول (نريد المجاميع التراكمية للأعمدة) نحصل على مصفوفة جديدة تحوي في السطر الأول ذات القيم في السطر الأول للمصفوفة المدخلة وفي السطر الثاني مجموع السطرين الأول والثاني وفي السطر الثالث مجموع الأسطر الثلاثة الأولى وهكذا. يمكن الحصول على مصفوفة ناتجة بطرق أخرى ولكن التابع cumsum يوفر الوقت والجهد.

مثال:

```
>> x=magic(3)
>> c=cumsum(x)
>> c=x;c(2,:)=x(1,:)+x(2,:);c(3,:)=sum(x)
c =
   8   1   6
  11   6  13
  15  15  15
```

السطر الثاني في هذا المثال يكافئ السطر الثالث ونحصل على نفس النتيجة ولكن يمكن الملاحظة أن طول الأمر المستخدم في السطر الثاني لا يتعلق بحجم وأبعاد

المصفوفة أما في السطر الثالث فيزداد الأمر تعقيدا كلما زاد حجم المصفوفة أو عدد أبعادها.

في حال الرغبة بالجمع التراكمي في اتجاه بعد غير الأول يجب إدخال رقم البعد كبارامتر ثاني بشكل مشابه لحالة الجمع بالتابع sum

محدد المصفوفة:

لحساب محدد مصفوفة مربعة يستخدم التابع det ويعيد قيمة واحدة فقط هي قيمة المحدد. تدعى المصفوفة ذات المحدد الصفري بالمصفوفة الفريدة أو المصفوفة الشاذة (unique matrix)

المضاريب التراكمية:

يعمل الأمر cumprod بشكل مماثل تماما للأمر cumsum لإيجاد المضاريب التراكمية للمصفوفة المدخلة.

مقلوب المصفوفة:

إذا كان ناتج ضرب مصفوفتين مربعتين هو مصفوفة أحادية (مصفوفة مربعة عناصر قطرها الرئيسي واحدات وبقية العناصر أصفار) عندئذ يقال أن كل من المصفوفتين هي مقلوب الأخرى. لإيجاد مقلوب مصفوفة معطاة يستخدم التابع inv الذي يقبل بارامتراً واحداً هو المصفوفة المراد حساب مقلوبها ويعيد ذلك المقلوب. في حال كانت المصفوفة فريدة ينتج التابع inv مصفوفة مساوية لها في الأبعاد جميع عناصرها تحمل القيمة الخاصة Inf ويظهر تحذيراً بأن المصفوفة فريدة.

مثال:

```
>> x=magic(3);
>> vx=inv(x);

>> x=[3 3 3;2 1 2;4 4 4];
>> vx=inv(x)
Warning: Matrix is singular to working precision.
vx =
  Inf  Inf  Inf
  Inf  Inf  Inf
    Inf    Inf    Inf
```

◄ العمليات المنطقية على المصفوفات

يقصد بالعمليات المنطقية تلك التي نحصل بموجبها على إجابة بنعم أو لا ويعبر ماتلاب عن الإجابة نعم بالعدد 1 وعن الإجابة بلا بالعدد 0 ويعيد مجموع الإجابات

في متجه أو مصفوفة حسب نوع السؤال المطروح وفيما يلي أهم العمليات المنطقية التي يمكن إجراؤها على مصفوفة:

البحث عن قيم محددة في المصفوفة:

يستخدم التابع `find` لإجراء عملية البحث عن القيم المحققة لشرط محدد في مصفوفة ما حيث يتم تمرير الشرط المطلوب وعدد مرات التكرار المطلوب البحث عنها وإحدى القيمتين النصيتين `first` أو `last` لإيجاد أدلة العناصر من المصفوفة التي تحقق الشرط بعدد التكرارات الممررة بدءاً من بداية المصفوفة أو نهايتها كما في المثال التالي:

```
>> x=[3 4 2 3 3 1 4 2];
>> find(x==3,2,'first')
ans =
   1   4
>> find(x==3,2,'last')
ans =
   4   5
>> find(x==3)
ans =
   1   4   5
```

في المثال الأول يبحث التابع find عن القيم المساوية لـ 3 في المصفوفة x ويعيد أدلة أول تكرارين وفي المثال الثاني يعيد آخر تكرارين أما في المثال الثالث عند حذف عدد التكرارات وخيار اتجاه البحث فهو يعيد كافة التكرارات، من البديهي أن عدد التكرارات هو أعظمي فلو وجد تكرار واحد فقط تتم إعادة دليله. الأدلة هنا هي الأدلة الترتيبية لعناصر المصفوفة أما إذا رغبنا في الحصول على أدلة الأبعاد (أرقام الأسطر والأعمدة) فيستخدم التابع find بالشكل التالي:

```
>> [row,col]=find(x==3)
row =
   1   1   1
col=
   1   4   5
```

حيث يعيد في هذه الحالة متجهين متساويين في عدد عناصرهما يحوي الأول أدلة الأسطر المحققة للشرط ويحوي الثاني أدلة الأعمدة.

في حال تم إدخال اسم المصفوفة فقط دون شرط تتم معاملة هذا التعبير على أنه شرط يجب أن يؤول حسابه إلى قيمة منطقية (0 أو 1) وعلى هذا يعيد التابع find مصفوفة أدلة العناصر اللاصفرية باعتبار أن أي قيمة غير الصفر تؤول منطقيا إلى القيمة المنطقية 1 (true) والقيم الصفرية فقط تؤول منطقيا إلى القيمة المنطقية 0 (false)

مثال:

```
>> x=[3 0 2 3 0 0 4 2];
>> find(x)
ans =
   1   3   4   7   8
```

في حال كانت المصفوفة قيد البحث ذات بعدين يعيد التابع `find` متجها عموديا في حالة الأدلة الترتيبية ومتجات عمودية في حالة أدلة الأسطر والأعمدة.

اختبار شرط ما في عناصر المصفوفة:

يستخدم التابع `any` لاختبار تحقق شرط ما في عناصر المصفوفة ويعيد القيمة المنطقية `1` في المواضع الموافقة لتحقق الشرط والقيمة المنطقية `0` في المواضع الأخرى، أما شكل المصفوفة الناتجة فهي تختلف حسب رقم البعد الذي يتم إدخاله كبارامتر للتابع كما توضح الأمثلة التالية:

```
>> x=[3 2 0;1 4 0;7 0 0]
x =
   3   2   0
   1   4   0
   7   0   0
>> any(x)
```

```
ans =
   1   1   0
>> any(x,1)
ans =
   1   1   0
>> any(x,2)
ans =
   1
   1
   1
>> any(x,3)
ans =
   1   1   0
   1   1   0
   1   0   0
```

في المثال التالي نريد البحث عن وجود العدد 2 في أعمدة المصفوفة x

```
>> x
x =
   3   2   0
   1   4   0
   7   0   0
>> any(x==2)
ans =
   0   1   0
```

المتجه الناتج يحوي أصفاراً في مواضع الأعمدة التي لا تحوي العدد 2 ويحوي القيمة 1 في موضع العمود الذي يحوي القيمة 2 مرة على الأقل.

يمكن استخدام التابع all بشكل مشابه تماما للتابع any مع فرق أن التابع all يحوي القيمة المنطقية 1 عند مواضع الاعمدة التي كل قيمها تحقق الشرط (عند البحث في اتجاه البعد الأول) وأذكر مرة أخرى ان إدخال اسم المصفوفة فقط دون شرط يؤول إلى اختبار كون العناصر لاصفرية. الأمثلة التالية توضح استخدام التابع all:

```
>> x
x =
   3   2   0
   1   4   0
   7   0   0
>> all(x==0)
ans =
   0   0   1
>> all(x)
ans =
   1   0   0
```

◄ العمليات الشكلية على المصفوفات

تستخدم التحويلات الشكلية على المصفوفات لتغيير ترتيب عناصر المصفوفة وفق توابع جاهزة في ماتلاب وأشهر التحويلات الممكنة هي:

إعادة تشكيل المصفوفة:

تتألف المصفوفة كما رأينا من مجموعة من القيم يتم ترتيبها بشكل محدد ضمن أسطر وأعمدة وربما بأبعاد أكثر، ويمكن إعادة ترتيب تلك العناصر بأبعاد مختلفة بواسطة التابع `reshape` الذي يقبل اسم المصفوفة كبارامتر أول والأبعاد الجديدة المطلوبة كبارامترات إضافية ويمكن تمرير هذه الأبعاد بشكل أرقام تفصل بينها فواصل أو بشكل متجه أفقي يعبر عن تلك الأبعاد كما في الأمثلة التالية:

```
>> x=[1:6]
x =
   1   2   3   4   5   6
>> x1=reshape(x,6,1)
x1 =
   1
   2
   3
   4
   5
   6
```

```
>> x=[1:6]
x =
   1   2   3   4   5   6
>> x1=reshape(x,2,3)
x1 =
   1   3   5
   2   4   6
>> x=[1:6];
>> x3=reshape(x,[2 3])
x3 =
   1   3   5
   2   4   6
```

في المثال الأول يعيد التابع reshape منقول المصفوفة وهو مكافئ للامر x' وفي المثالين الثاني والثالث نحصل على نفس النتيجة بإدخال الأبعاد الجديدة للمصفوفة الناتجة بطريقتين مختلفتين.

قلب المصفوفة أفقيا:

يمكن قلب المصفوفة ثنائية الأبعاد أفقيا باستخدام التابع fliplr أو التابع flipdim كما يلي:

```
>> x=[1 2 3;4 5 6]
x =
```

```
   1   2   3
   4   5   6
>> y=fliplr(x)
>> y=flipdim(x,2)
y =
   3   2   1
   6   5   4
```

عند استخدام التابع `fliplr` يتم تمرير اسم المصفوفة المراد قلبها فقط، أما عند استخدام التابع `flipdim` فيتم تمرير اسم المصفوفة ورقم البعد المراد القلب في اتجاهه وهو هنا 2 ليكافئ الانقلاب الأفقي من اليسار إلى اليمين.

يلاحظ أن الانقلاب الأفقي لمصفوفة عدد عناصرها فردي لا يغير مواضع العمود الأوسط حيث ينطبق محور الدوران عليه في هذه الحالة.

يلاحظ أيضا أن قلب متجه عمودي لا يغير شيئاً في المتجه.

قلب المصفوفة عموديا:

يشبه هذا الإجراء إجراء القلب الأفقي، ولكن يستخدم فيه التابع flipud بدلاً من التابع fliplr أو يستخدم التابع flipdim ويمرر له رقم البعد الأول 1 ليناسب اتجاه الإنقلاب.

يلاحظ أيضاً أن المصفوفة ذات عدد الأسطر الفردي لا يتغير فيها السطر الأوسط لأن محور الإنقلاب ينطبق عليه، وكذلك فإن المتجه الأفقي لا يتغير بالإنقلاب الرأسي.

إدارة المصفوفة بزاوية واتجاه:

يستخدم التابع rot90 لإدارة المصفوفة بعكس عقارب الساعة بزاوية من مضاعفات الزاوية 90 كما في الأمثلة التالية:

```
>> x=[1 2 3;4 5 6]
x =
   1   2   3
   4   5   6
>> rot90(x)
   3   6
   2   5
   1   4
>> rot90(x,2)
   6   5   4
   3   2   1
```

```
>> rot90(x,3)
   4   1
   5   2
   6   3
```

في الواقع لا فائدة من أي إدخالات غير الإدخالات الثلاثة المذكورة في هذا المثال، فعلى سبيل المثال فإن تدوير المصفوفة بزاوية 90 درجة عكس عقارب الساعة يكافئ تدويرها بزاوية 270 درجة في اتجاه عقارب الساعة، وكمثال آخر فإن تدويرها بزاوية 360 درجة لن يغير فيها شيئاً.

يلاحظ أيضاً أن تدوير المصفوفة بزاوية 90 درجة يكافئ إجراء قلب رأسي لمنقولها:

```
>>flipud(x')
>>rot90(x)
```

وكذلك فإن تدويرها بزاوية 180 درجة يكافئ إجراء إنقلابين متتاليين لها أحدهما رأسي والآخر أفقي بغض النظر عن الترتيب:

```
>>fliplr(flipud(x))
>>flipud(fliplr(x))
>>rot90(x,2)
```

ترتيب عناصر المصفوفة:

يستخدم التابع sort لترتيب عناصر المصفوفة تصاعدياً أو تنازلياً، وهو يعمل بطريقة مشابهة لطريقة عمل التوابع max و min و sum وغيرها حيث يقوم بالترتيب في أحد الاتجاهات ويكون الاتجاه الافتراضي له هو اتجاه الأسطر (يقوم بترتيب الأعمدة)، أما إذا أردنا ترتيب الأسطر فيجب إدخال قيمة البارامتر الثاني وهو رقم يدل على البعد المراد إجراء الترتيب فيه. من جهة أخرى فإن الترتيب الافتراضي هو الترتيب التصاعدي، أما في حال الرغبة بالترتيب التنازلي فيجب إدخال قيمة البارامتر الثالث الذي يقبل إحدى القيمتين النصيتين ascend للترتيب التصاعدي وهي الافتراضية أو descend للترتيب التنازلي. الأمثلة التالية توضح عملية الترتيب:

```
>> x=magic(3)
   8   1   6
   3   5   7
   4   9   2
>> sort(x)
   3   1   2
   4   5   6
   8   9   7
>> sort(x,2)
   1   6   8
   3   5   7
   2   4   9
```

```
>> sort(x,2,'descend')
   8   6   1
   7   5   3
   9   4   2
```

يمكن أيضا الحصول على مصفوفة الأدلة القديمة للعناصر المرتبة بإسناد نتيجة الترتيب إلى متحولين أولهما يحوي نتيجة الترتيب والثاني مصفوفة الأدلة القديمة (قبل الترتيب) كما في المثال التالي:

```
>> x=magic(3)
   8   1   6
   3   5   7
   4   9   2
>> [u i]=sort(x)
u =
   3   1   2
   4   5   6
   8   9   7
i =
   2   1   3
   3   2   1
   1   3   2
```

يمكن إجراء ترتيب لكامل عناصر المصفوفة باستخدام هذا التابع وبعض التقنيات السابقة كمايلي:

```
>> a=magic(4);
>> b=reshape(a,1,prod(size(a)));
>> c=sort(b);
>> d=reshape(c,size(a));
```

في الخطوة الأولى يتم تغيير شكل المصفوفة a لتصبح متجهاً أفقياً وفي الخطوة الثانية يتم ترتيبها تصاعديا أو تنازلياً (تصاعدياً هنا) وفي الخطوة الثالثة يعاد تشكيلها بأبعادها السابقة للحصول على المصفوفة المرتبة عموداً عموداً وإذا أردنا الترتيب سطراً سطراً يمكن إيجاد منقول المصفوفة الناتجة.

يمكن دمج الخطوات السابقة جميعاً في تعبير واحد كمايلي:

```
>> d=reshape(sort(reshape(a,1,prod(size(a)))),size(a));
```

ترتيب أسطر المصفوفة:

يختلف هذا الترتيب عن الترتيب السابق حيث يتم التعامل مع الأسطر وكأن كل سطر هو كتلة واحدة (كما في سجلات قواعد البيانات) ويتم الترتيب بالإعتماد على أحد الأعمدة حيث يتم ترتيب قيمه تصاعدياً ويتم نقل كافة بيانات الأعمدة الموافقة معه

وفي حال تساوي قيمتين في العمود المعتمد يتم الاعتماد في ترتيبهما على العمود التالي إلى اليمين وهكذا حتى الوصول إلى العمود الأخير فإذا تساوت قيمتان فيه تترك كما هي دون تبديل. يتم هذا الترتيب باستخدام التابع sortrows الذي يقبل مصفوفة ثنائية البعد أو متجه عمودي حصراً (لا معنى لترتيب أسطر متجه أفقي) كبارامتر أول ورقم العمود المعتمد في الترتيب كبارامتر ثاني خياري حيث يتم اعتماد العمود الأول في الحالة الافتراضية ويعيد التابع مصفوفة مرتبة الأسطر. المثال التالي يوضح عملية الترتيب:

```
>> a=magic(4);
>> a(1:3,1)=5
a =
   5   2   3  13
   5  11  10   8
   5   7   6  12
   4  14  15   1
>> [b ind]=sortrows(a)
b =
   4  14  15   1
   5   2   3  13
   5   7   6  12
   5  11  10   8
ind =
   4
   1
   3
   2
```

المتجه ind يحوي قيم أدلة الأسطر كما أصبحت بعد الترتيب ومن النتيجة التي حصلنا عليها هنا نلاحظ أن السطر الرابع أصبح هو السطر الأول وذلك اعتماداً على العمود الأول لأن قيمة العنصر من العمود الأول في السطر الرابع هي القيمة الأدنى ويليه السطر الأول الذي أصبح ثانيا حيث تتم المقارنة بين قيم العمود الثاني بسبب تساوي قيم العمود الأول في بقية الأسطر والسطر الثالث الذي بقي ثالثا ثم السطر الثاني الذي أصبح رابعاً.

هناك تابع مفيد يخبرنا فيما إذا كانت المصفوفة مرتبة أم لا وهو التابع issorted حيث يقبل متجهاً أفقياً أو عمودياً ليخبرنا بأنه مرتب العناصر أم لا أو يقبل مصفوفة ذات أعمدة وأسطر ليخبرنا بكونها مرتبة الأسطر أم لا وبالطبع هنا يجب إدخال بارامتر ثاني وهو القيمة النصية rows

مثال 1:

```
>>a=[2:10];
>>issorted(a)
>>ans=1
```

مثال 2:

```
>>a=magic(3);
>>issorted(a,'rows')
>>ans=0
```

```
>>a=sortrows(a)
>>issorted(a,'rows')
>>ans=1
```

من الملاحظ أنه كان بالإمكان جعل التابع `issorted` يقبل متجهات أو مصفوفات بدون القيمة النصية `rows` كبارامتر ثاني وجعله يميز لوحده فإن تم إعطاؤه متجها اختبر ترتيبه وإن أعطي مصفوفة اختبر ترتيب أسطرها، ولست أرى سبباً للبارامتر الثاني إلا أن ماتلاب أراد أن ينبه المستخدم، وهذه من المفارقات فمستخدم ماتلاب يجب أن يكون نبيها بطبيعته.

النصوص والمتحولات النصية

◄ إدخال وتخزين النصوص

يتم تخزين القيم النصية في ماتلاب بشكل أعداد صحيحة ويظهرها كنصوص باستخدام الترميز Unicode ولإخبار ماتلاب بأن القيم المدخلة هي نصوص يجب استخدام إشارتي التنصيص المفردتين '' بدلا من الحاصرتين [] ويقوم ماتلاب في هذه الحالة بتمييز القيمة المدخلة وتخزين قيمة العدد الصحيح الموافق لكل محرف لها في الترميز العالمي Unicode سواءً كانت قيمة رقمية أو حرفية. في المثال التالي توضيح لعملية إدخال النصوص:

```
>> x='matlab'
x =
matlab
>> whos
  Name      Size            Bytes  Class    Attributes
  x         1x6                12  char
```

عند استخدام الأمر whos في هذا المثال نرى أن حجم المتجه النصي هو 1×6 فهو متجه أفقي ذو ستة أعمدة كل عمود فيه محرف واحد وتم تخزينه بالنوع char ليتمكن ماتلاب من تمييزه عن النوع الرقمي الواجب التعامل معه كأرقام مثل double

يمكن التمييز بين إدخال العدد 23 كنص أو كقيمة عددية كما في المثال التالي:

```
>> x='23';n=23;
>> whos
 Name      Size            Bytes  Class     Attributes
 n         1x1                 8  double
 x         1x2                 4  char
```

نلاحظ أن المتحول `x` المدخل كنص تم تخزينه في متجه نصي يحوي عمودين بالنوع `char` أما المتحول `n` المدخل كعدد فتم تخزينه بالنوع `double` كمتحول عددي.

يمكن استخدام الكثير من التوابع المستخدمة للحصول على معلومات عن المتجه النصي بشكل مماثل لما رأيناه في المتجهات والمصفوفات العددية والمثال التالي يوضح بعض الاستخدامات ونتائجها:

```
>> c='matlab';
>> class(c)
ans =
char
>> length(c)
ans =
    6
>> size(c)
ans =
    1    6
>> sum(c)
ans =
  625
```

```
>> uint8(c)
ans =
  109   97  116  108   97   98
>> sum(ans)
ans =
  625
```

إن استخدام التوابع class و length و size في هذا المثال واضح ولا يحتاج إلى شرح أما التابع sum فهو يجمع القيم العددية الصحيحة المقابلة للمحارف المدخلة في المتجه النصي وفي الواقع هذه القيم هي ما يتم تخزينه فعلا كما أسلفنا، أما التابع uint8 فهو يظهر هذه القيم المخزنة ولذلك فإن sum(c) يكافئ sum(uint8(c))

من الجدير بالذكر أنه يمكن استخدام مصفوفات نصية متعددة الأسطر بشرط تساوي عدد الأعمدة في كل سطر ولذلك عند توزيع النصوص على الأسطر يجب أن تكون متساوية الطول وإن لم تكن فيجب إضافة مسافات حشو للنصوص الأقصر من أطول نص، كذلك فإن استخدام الفواصل العادية أو الفراغات يؤدي إلى إضافة فواصل أو مسافات نصية لأن هذه الأشياء هي محارف هنا.

الأمثلة التالية توضح بعض الأشياء بهذا الخصوص:

مثال 1:

```
>> x='matlab',y='m,a,t,l,a,b',z='m a t a l a b'
```

```
x =
matlab
y =
m,a,t,l,a,b
z =
m a t a l a b
```

مثال 2:

```
>> x='matlab','is','good',y=['matlab','is','good']
x =
matlab
ans =
is
ans =
good
y =
matlabisgood
```

في هذا المثال تم استخدام الفواصل بين المتجهات فتعامل معها ماتلاب على أنها تعليمات منفصلة وتم إسناد القيم الأولى إلى المتجه x وإسناد القيم التالية إلى المتحول الخاص ans في الثانية والثالثة. أما في حال استخدام الحاصرتين [] فتم التعامل معها على أنها تنتمي إلى متجه واحد وهي طريقة لتركيب المتجهات النصية.

مثال 3:

```
>> x=['matlab';'is   ';'good  ']
x =
matlab
is
good
```

هنا تم استخدام الفواصل المنقوطة لفصل المتجهات النصية إلى أسطر وتم الحشو بمسافات لأن عدد الأعمدة في كل سطر يجبب أن يكون موحداً.

◄ بعض التوابع الهامة في المصفوفات النصية

تستخدم بعض التوابع لتوليد المصفوفات النصية بشكل سريع ودقيق بدلاً من كتابتها يدوياً كمايلي:

يستخدم التابع `blank` لتوليد مصفوفة من الفراغات (المسافات البيضاء) حيث يمرر له عدد المسافات البيضاء المطلوبة كمايلي:

```
>> b=blanks(10);
>> s=['xx',b,'xx']
s =
xx        xx
```

تم استخدام القيمتين xx في كلا جانبي النص s لإظهار وجود المسافات البيضاء العشر بينهما.

يستخدم التابع deblank(string) لحذف كافة المسافات البيضاء التالية لآخر محرف في النص المدخل كما يلي:

```
>> b=blanks(10);
>> w=[b,'matlab',b];
>> s=['xx',w,'xx']
s =
xx        matlab        xx
>> s=['xx',deblank(w),'xx']
s =
xx        matlabxx
```

حيث تم استخدامه هنا لحذف المسافات البيضاء من النص المخزن في المتحول w

يستخدم التابع strtrim(string) لحذف كافة الفراغات السابقة للمحرف الأول والتالية للمحرف الاخير في النص المدخل كمايلي:

```
>> b=blanks(10);
>> w=[b,'matlab',b];
>> s=['xx',w,'xx']
s =
xx        matlab        xx
>> s=['xx',strtrim(w),'xx']
```

```
s =
xxmatlabxx
```

لا يوجد تابع لإزالة المسافات البيضاء مباشرة من ضمن النص ولكن توجد طريقة كمايلي:

```
>> x='m  at  l ab'
x =
m  at  l ab
>> x(x==' ')=[]
x =
matlab
```

في هذه المثال يتم إعادة تعيين قيم المتجه النصي x المحققة للشرط x=' ' بحيث يسند إليها القيمة الخالية [] حيث أن هذه القيمة الخاصة تعني حذف القيمة المحددة.

◄ الإختبارات على المصفوفات النصية

يمكن إجراء العديد من الإختبارات على النصوص باستخدام توابع مرت معنا في الاختبارات على المصفوفات العددية وتوابع أخرى خاصة بالنصوص. وفيمايلي بعض الأمثلة الهامة:

اختبار وجود محرف ما

يتم هذا الاختبار بمقارنة عادية كمايلي:

```
>> x='Memory'
x =
Memory
>> x=='M' | x=='m'
ans =
   1   0   1   0   0   0
```

اختبار وجود المسافات البيضاء

يمكن إجراء هذا الاختبار مباشرة بعملية مقارنة عادية أو باستخدام التابع `isspace(string)` كمايلي:

```
>> x='Matlab 2015'
x =
Matlab 2015
>> isspace(x)
ans =
   0   0   0   0   0   0   1   0   0   0   0
>> x==' '
ans =
   0   0   0   0   0   0   1   0   0   0   0
```

اختبار نوع المحرف

يستخدم لهذا التابع `isstrprop(string,'property')` حيث يمكن أن تكون القيمة النصية `property` إحدى القيم التالية (على سبيل المثال):

`alpha` لاختبار الحروف.

`alphanum` لاختبار الحروف والأرقام.

`digit` لاختبار الأرقام.

`upper` لاختبار كون الحروف كبيرة.

`lower` لاختبار كون الحروف صغيرة.

هناك قيم أخرى ولكن لا أود الإطالة هنا بذكرها كلها، وكذلك في بقية الكتاب لا أذكر كل التفاصيل وأترك للقارئ أن يبحث عن تفاصيل أكثر في حال الحاجة إلى ذلك، فالغاية من هذا الكتاب ليست وضع مرجع في ماتلاب بل تسليط الضوء على بعض تقنياته وإمكانياته.

أمثلة:

```
>> x='Matlab 2015'
x =
Matlab 2015
>> isstrprop(x,'alpha')
```

```
ans =
   1   1   1   1   1   1   0   0   0   0   0
>> isstrprop(x,'alphanum')
ans =
   1   1   1   1   1   1   0   1   1   1   1
>> isstrprop(x,'upper')
ans =
   1   0   0   0   0   0   0   0   0   0   0
```

◄ مقارنة النصوص

يمكن مقارنة النصوص باستخدام عمليات المقارنة العادية (==, ~=, >, >=, <, <=) إذا كان طول النص الأول يساوي طول النص الثاني حيث تتم مقارنة كل محرف من النص مع المحرف المقابل له من النص الآخر. وعند استخدام معامل مقارنة مثل > فالمقصود هو اختبار كون المحرف الأول يقابل عدداً صحيحاً أقل من العدد الصحيح المقابل للمحرف الثاني حيث ذكرنا سابقاً أن المحارف النصية تقابل أعداداً صحيحة في جدول الترميز العالمي Unicode

أمثلة:

```
>> s1='Hello';s2='help ';
>> s1==s2
ans =
   0   1   1   0   0
>> s1>s2
```

```
ans =
   0   0   0   0   1
```

في حال تساوي النصين موضوع المقارنة أو عدم تساويهما يمكن استخدام التوابع الخاصة بالمقارنة وهي لا تعيد نتائج مماثلة للمقارنة بالعمليات بل تعيد قيما منطقية مفردة (ليست متجهات من القيم المنطقية) وهذه المقارنات هي:

مقارنة نصين لاختبار تطابقهما

يستخدم لذلك أحد التابعين `strcmp` أو `strcmpi` حيث يتم إهمال حالة الأحرف في الثاني، ويعيد كل من هذين التابعين القيمة المنطقية `1` في حال تطابق النصين أو القيمة المنطقية `0` في حال عدم التطابق.

اختبار تطابق عدد محدد من المحارف بين نصين

يستخدم التابعان `strncmp` أو `strncmpi` حيث يتم إهمال حالة الأحرف في الثاني ويأخذ هذان التابعان بارامتراً إضافيا يحدد عدد المحارف المطلوب مقارنتها من النصين ويبدأ العد من بداية النص.

أمثلة:

```
>> s1='Hello';s2='help ';
>> strcmp(s1,s2)
ans =
    0
>> strcmpi(s1,s2)
ans =
    0
>> strncmpi(s1,s2,3)
ans =
    1
```

خلاياالبيانات المتنوعة

رأينا أن ماتلاب يتعامل مع القيم العددية والنصية جميعا على أنها مصفوفات، والمصفوفات كما نعلم وكما رأيناها هنا هي بنى معطيات متماثلة القوام من حيث نوع المعطيات وحجم المكونات الجزئية، بمعنى أن عدد الأعمدة في كل سطر من أسطر المصفوفة يجب أن يكون ثابتا ونوع المعطيات يجب أن يكون واحدا، وعلى هذا لا يمكن تخزين قيمة نصية وأخرى عددية في مصفوفة واحدة ولا يمكن تخزين قيمة عددية من النوع الحقيقي مع قيم من النوع الصحيح في مصفوفة واحدة. لحل هذه المشكلة وتخزين قيم مختلفة النوع ومختلفة الحجم تستخدم خلايا البيانات cell arrays حيث تعطى المصفوفة من هذا النوع اسماً ويتم تقسيمها إلى صفوف وأعمدة، وربما تحوي أبعاداً إضافيةً أيضاً ويتم تخزين مصفوفة بيانات من نوع واحد في كل موقع من مواقع مصفوفة الخلايا وهذا الموقع يسمى خلية بيانات. بعبارة أخرى يمكن اعتبار مصفوفة الخلايا كمصفوفة من المصفوفات كل عنصر من عناصرها هو مصفوفة من نوع ما.

◄ إنشاء مصفوفة الخلايا

يستخدم لإنشاء مصفوفة الخلايا طريقة مشابهة لطريقة إنشاء المصفوفات العادية حيث يستخدم اسم متحول ويسند له المحتوى المطلوب ولكن هنا بين قوسي مجموعة {} بدلا من القوسين الكبيرين [] ويتم ذكر البيانات المطلوبة كعناصر لمصفوفة

الخلايا كمصفوفات وليس كقيم مفردة. في الأمثلة التالية نوضح عدة أشكال لإنشاء مصفوفة خلايا.

مثال 1:

```
>> a=ones(1,3);b=[2 3;4 3];
>> st='Hasan';x='matlab';
>> cellArray={a st;b x}
cellArray =
   [1x3 double]    'Hasan'
   [2x2 double]    'matlab'
```

في هذا المثال تم تعريف العناصر المكونة لمصفوفة الخلايا وهي المصفوفة `a` المكونة من متجه أفقي والمصفوفة `b` المكونة من مصفوفة ثنائية الأبعاد والمصفوفتان النصيتان `st` و `x` ثم استخدمت المصفوفة `cellArray` لجمع تلك المصفوفات الأربع معا في بنية واحدة متعددة الأحجام والانواع.

مثال 2:

```
>> cellArray={[3 3 4] [2 2;1 2;3 2] ; ...
'a b c' 'cell arrays'}
cellArray =
   [1x3 double]    [3x2 double]
   'a b c'         'cell arrays'
```

في هذا المثال استخدمت المصفوفة cellArray بأسلوب مشابه للمثال السابق مع فرق أن الإدخال هنا تم بشكل مباشر في مصفوفة الخلايا.

مثال 3:

```
>> cellArray={[3 3 4] [2 2;1 2];'matlab'}
??? Error using ==> vertcat
CAT arguments dimensions are not consistent.
>> cellArray={[3 3 4] [2 2;1 2];'matlab' []}
cellArray =
   [1x3 double]    [2x2 double]
   'matlab'              []
```

يجب الانتباه إلى أن مصفوفة الخلايا شأنها شأن المصفوفة العادية يجب أن تكون ذات شكل بنيوي صحيح أي لا يمكن أن تحوي في سطرها الأول عمودين وفي سطرها الثاني عمود واحد على سبيل المثال، ولهذا فإن ماتلاب يعيد رسالة خطأ في هذا المثال عند محاولة الإدخال الأولى ولحل المشكلة تمت المحاولة ثانية مع وضع مصفوفة فارغة في الموقع الشاغر لجعل بنية مصفوفة الخلايا بنية صحيحة.

مثال 4:

```
>> c1={[1 2 1 1];['cell']}
c1 =
   [1x4 double]
   'cell'
```

```
>> c2={'matlab';[2 2;4 3]}
c2 =
   'matlab'
   [2x2 double]
>> c={c1 c2 [3 3]}
c =
   {2x1 cell}   {2x1 cell}   [1x2 double]
```

في هذا المثال تم تكوين مصفوفة الخلايا c من مصفوفتي خلايا c1 و c2 ومصفوفة عددية من عددين صحيحين وفي هذه الحالة تبقى عناصر مصفوفة الخلايا c مستقلة. أما في المصفوفات العادية فيتم دمج العناصر المكونة للمصفوفة النهائية عند استخدام أسلوب مماثل.

◄ الوصول إلى عناصر مصفوفة خلايا

يمكن الوصول إلى عناصر مصفوفة خلايا بأسلوب مماثل لأسلوب الوصول إلى عناصر مصفوفة عادية، حيث يستخدم أسلوبا الترقيم ذاتاهما، ففي الأول يتم ترقيم خلايا مصفوفة الخلايا من الواحد إلى العنصر الأخير عموداً عموداً، وفي الأسلوب الثاني يتم ترقيم الخلايا بترقيم أسطرها وأعمدتها. الفارق هنا هو وجود طريقتين مختلفتين للتعبير عن عنصر (خلية) في مصفوفة خلايا، في الطريقة الأولى عند

استخدام الأقواس العادية () يعيد ماتلاب وصفاً فقط لخلايا المصفوفة أما في حال استخدام الأقواس {} فيعيد ماتلاب محتوى تلك الخلية المشار إليها.

مثال توضيحي:

يمكن تعريف مصفوفة خلايا c كمايلي:

```
>> c={[1 2;3 7] [4 2 3 3];'matlab' 'big'}
```

شكل المصفوفة كما يعيده ماتلاب هو التالي:

```
   [2x2 double]    [1x4 double]
   'matlab'        'big'
```

يمكن الوصول إلى العنصر الأول في هذه المصفوفة بطريقتي الترقيم المذكورتين آنفاً كمايلي:

```
>> c(1)
ans =
   [2x2 double]
>> c(1,1)
ans =
   [2x2 double]
```

ولكن كما نرى تكون النتيجة هي وصف للعنصر المشار إليه فقط.

للوصول إلى محتوى العنصر يمكن استخدام أسلوبي الترقيم ذاتيهما مع استبدال الأقواس كما ذكرنا:

```
>> c{1}
ans =
   1   2
   3   7
>> c{1,1}
ans =
   1   2
   3   7
```

هكذا يعيد ماتلاب محتوى العنصر المشار إليه وهو هنا مصفوفة عددية، أما للوصول إلى عنصر من المصفوفة العددية هذه فتستخدم طريقة مركبة حيث يتبع ما سبق بقوسين يحويان ترقيماً للعنصر المفرد المراد الوصول إليه، وللمثال تكون الطرق الأربعة التالية متطابقة من حيث النتيجة:

```
>> c{1,1}(2,1)
>> c{1,1}(2)
>> c{1}(2,1)
>> c{1}(2)
ans =
   3
```

وبالتأكيد يمكن استخدام الأساليب الأخرى في الوصول إلى أكثر من عنصر كما في الأمثلة التالية:

```
>> c
c =
   [2x2 double]    [1x4 double]
   'matlab'        'big'
>> c{:,1}
ans =
    1    2
    3    7
ans =
matlab
>> c{1}(:,2)
ans =
    2
    7
```

بنى المعطيات المركبة

بنى المعطيات المركبة في ماتلاب هي طريقة لتخزين البيانات بأسلوب هرمي مرن، حيث يمكن حفظ أي نوع من البيانات في كل موقع تخزين من البنية المركبة structure

يسمى كل موقع تخزين لتلك البيانات حقل بيانات وتستخدم علامة النقطة (.) للوصول إلى حقول البيانات، يمكن للحقل أن يكون أي نوع من البيانات بما فيها بنية معطيات مركبة اخرى.

◄ إنشاء بنى المعطيات المركبة

يمكن إنشاء بنية معطيات بطريقة بسيطة كمايلي:

```
>> person.name.fst='Hasan';
>> person.name.lst='Alhouri';
>> person.age=39;
```

هذه البنية المسماة person تحوي هنا حقلين الأول هو name والثاني هو age. الأول بدوره هو بنية معطيات تحوي حقلين أولهما هو fst والثاني هو lst

◄ الوصول إلى العناصر

للوصول إلى عناصر هذه البنية يستخدم الأسلوب التالي:

```
>> person.name.fst
ans =
Hasan
>> person.name.lst
ans =
Alhouri
>> person.age
ans =
   39
```

إذا أردنا الآن إضافة مركبة جديدة لهذه البنية نستخدم أسلوب الإسناد المتبع في إسناد قيم جديدة لمصفوفة عادية كمايلي:

```
>> person.name.fst='Wisam';
>> person.name.lst='Alhouri';
>> person.age=7;
```

يمكن استخدام التابع `struct` لإدخال البيانات إلى البنية السابقة بطريقة أخرى كمايلي:

```
per=struct('name',
        {struct('fst','Hasan','lst','Alhouri'),struct('fst','Hasan','lst','Alhouri')},
```

```
'age',{39,7})
```

يتم إدخال اسم الحقل أولاً ثم بيانات الحقل لكافة عناصر البنية وبما أن الحقل `name` هنا هو بنية أيضا تم استخدام التابع `struct` مرة أخرى.

الشكل العام للتابع `struct` هو كمايلي:

```
struct('field1', values1, 'field2', values2, ...)
```

حيث يمكن أن تكون القيم قيما عادية مفردة او مصفوفات أو بنى معطيات مضمنة كما في المثال السابق.

◄ عرض أسماء الحقول

التابع `fieldname(structure name)` يستخدم لعرض أسماء الحقول في بنية المعطيات ويمكن استخدامه لمعرفة الحقول في المستوى الأول أو الثاني أو أي مستوى كما في المثال التالي:

```
>> fieldnames(per)
ans =
   'name'
   'age'
```

```
>> fieldnames(per(1).name)
ans =
   'fst'
   'lst'
```

حيث تم استخدامه هنا للمرة الأولى لمعرفة أسماء الحقول في البنية `per` ثم استخدم للمرة الثانية لمعرفة أسماء الحقول في البنية المضمنة `per(1).name`

◄ إضافة حقول وتغيير قيمها

عند الحاجة إلى إضافة حقل جديد تستخدم عملية الإسناد مباشرة للحقول المراد إضافتها مع الانتباه إلى ضرورة تحديد دليل في حال كانت البنية ذات أبعاد غير الواحد كما في المثال:

```
>> per(1).gender='male'
per =
1x2 struct array with fields:
   name
   age
   gender
>> per.gender
ans =
male
ans =
```

```
[]
```

في هذا المثال تمت إضافة الحقل gender إلى البنية per عند الدليل 1 وأعطيت له القيمة male هذا يؤدي إلى إضافة هذا الحقل لكل البنية ولكن تكون قيمته عند بقية الأدلة قيمة خالية [].

إذا أردنا تغيير تلك القيمة نقوم بالإسناد ثانية إليها كمايلي:

```
>> per(2).gender='male'
```

◄ حذف الحقول

يستخدم التابع rmfield لحذف حقل أو أكثر من بنية المعطيات، ويجب الانتباه إلى ضرورة إسناد ناتج العملية إلى اسم البنية ثانية كما في المثالين التاليين:

```
>> per=rmfield(per,'gender')
>> per=rmfield(per,{'city','country'})
```

حيث يتم حذف الحقل gender في المثال الأول ويتم حذف الحقلين city و country في المثال الثاني. ونلاحظ أن الحقول المراد حذفها هي مصفوفة خلايا نصية.

◄ التحويل من بنية إلى خلية والعكس

يمكن تحويل بنية معطيات إلى خلية بيانات بواسطة التابع `struct2cell(structure name)` ويمكن العكس بواسطة التابع `cell2struct(cell name)` كما في المثالين التاليين:

مثال 1: التحويل من بنية إلى خلية

```
>> s.name='Hasan';s.job='Engineer';s.age=39
s =
   name: 'Hasan'
    job: 'Engineer'
    age: 39
>> c=struct2cell(s)
c =
   'Hasan'
   'Engineer'
   [    39]
```

مثال 2: التحويل من خلية إلى بنية

```
>> c={'Hasan';'Engineer';39}
c =
   'Hasan'
```

```
    'Engineer'
    [    39]
>> cell2struct(c,{'name','job','age'},1)
ans =
    name: 'Hasan'
     job: 'Engineer'
     age: 39
```

في المثال الثاني نحن مضطرون لإعلام ماتلاب بأسماء الحقول المرغوبة في البنية والاتجاه المراد استخدامه في الخلية لأخذ القيم، وهو هنا اتجاه الأسطر أي أن كل سطر يحوي قيم لحقل محدد وبالطبع هنا لا ينجح استخدام الأعمدة كاتجاه لأن عدد الحقول هو ثلاثة حقول وعدد الأعمدة هو عمود واحد.

في المثال التالي نبين استخدام الاتجاه مرة في الأسطر ومرة في الأعمدة:

```
>> c={'Hasan','Engineer';'Wisam','Student'}
c =
    'Hasan'    'Engineer'
    'Wisam'    'Student'
>> s1=cell2struct(c,{'name','job'},1)
s1 =
2x1 struct array with fields:
    name
    job
>> s2=cell2struct(c,{'name','job'},2)
s2 =
```

2x1 struct array with fields:
name
job

للوهلة الأولى تبدو النتائج متطابقة، ولكن عند إمعان النظر فيما يجب فعله نرى أن الحقل `name` في الحالة الأولى سوف يأخذ القيم التي في السطر الأول وهذا خطأ، أما في الحالة الثانية فسوف يأخذ قيم العمود الأول وهو العمل الصحيح. وللتأكد يمكن محاولة الوصول إلى قيم حقل الاسم كمايلي:

>> s1.name
ans =
Hasan
ans =
Engineer
>> s2.name
ans =
Hasan
ans =
Wisam

الأعداد العقدية (المركبة)

◄ التعبير عن الأعداد المركبة وإدخالها

يعبر المتحول الخاص i أو المتحول الخاص j عن العدد التخيلي $\sqrt{-1}$ وباستخدام هذا العدد يمكن إدخال الأعداد العقدية (المركبة) إلى ماتلاب بكتابتها بنفس الطريقة التي نكتبها فيها على الورق كما في الأمثلة التالية:

```
>> x=2+1i
x =
  2.0000 + 1.0000i
>> y=3-j
y =
  3.0000 - 1.0000i
```

في حال استخدام أحد المتحولين i أو j لتخزين قيم أخرى لا يتأثر إدخال الأعداد المركبة ولكن يتأثر جواب ماتلاب على المتحولين بعينهما، ولإلغاء القيمة غير الخاصة من المتحول المستخدم يتم حذفه من مساحة العمل باستخدام التابع clear متبوعا باسم المتحول i أو j

الأمثلة التالية توضح آلية تعامل ماتلاب مع الأعداد المركبة والمتحولين الخاصين i و j

```
>> i
ans =0 + 1.0000i
>> j
```

ans =0 + 1.0000i

يعيد ماتلاب قيمة المتحول الخاص i كعدد مركب قسمه الحقيقي يساوي الصفر والتخيلي يساوي الواحد وكذلك المتحول الخاص j

>> i=9
i =9

يمكن استخدام أي من المتحولين لحفظ قيمة أخرى:

>> x=3+i
x =12

في هذه الطريقة يتعامل ماتلاب مع المتحول i على أساس القيمة المخزنة فيه من قبل المستخدم.

>> x=3+1i
x =3.0000 + 1.0000i

في هذا المثال يفهم ماتلاب أن العدد x هو عدد مركب حيث يعتبر هذا التعبير صحيحاً فقط في حالة الأعداد المركبة.

```
>> x=3+j
x =3.0000 + 1.0000i
```

بما أن المتحول j غير مستخدم من قبل المستخدم فقيمته لا تزال عدداً مركباً، ويتم جمعها هنا إلى العدد الصحيح 3 وإعادة النتيجة بشكل عدد مركب مع ملاحظة أن ماتلاب يعيد نتيجته دائماً باستخدام المتحول i وليس j رغم قبوله أي منهما.

يمكن أيضاً الحصول على متجه أو مصفوفة من الأعداد المركبة ولكن لا تصح صيغة الإدخال السابقة وإنما يتم التعامل مع المتحولين الخاصين i و j كأي قيميتن، وعلى هذا يجب أن لا يكون أحدهما (المراد استخدامه) مستخدماً في مساحة العمل، وكمثال يمكن الحصول على متجه عمودي من أربعة أعداد مركبة أقسامه الحقيقية والتخيلية في متجهين أفقيين كمايلي:

```
>> a=[1:4];
>> b=[5:8];
>> x=a'+b'*i
x =
  1.0000 + 5.0000i
  2.0000 + 6.0000i
  3.0000 + 7.0000i
  4.0000 + 8.0000i
```

يمكن أيضاً الحصول على النتيجة ذاتها باستخدام التابع complex كمايلي:

```
>> a=[1:4];
>> b=[5:8];
>> complex(a',b')
ans =
   1.0000 + 5.0000i
   2.0000 + 6.0000i
   3.0000 + 7.0000i
   4.0000 + 8.0000i
>> complex(a',5)
ans =
   1.0000 + 5.0000i
   2.0000 + 5.0000i
   3.0000 + 5.0000i
   4.0000 + 5.0000i
```

في الواقع يستخدم التابع `complex` عند كون الأقسام الحقيقية في متجه والتخيلية في متجه آخر أما العملية الأخرى فهي عامة الاستخدام حيث يمكن أن يحوي المتجه الأول أعداداً حقيقة ومركبة وكذلك المتجه الثاني.

◄ التوابع الخاصة بالأعداد المركبة:

يوفر ماتلاب مجموعة من التوابع المفيدة في التعامل مع الأعداد المركبة وفيمايلي جدول بها:

التابع	وظيفته
`Conj(x)`	يعيد مرافق العدد المركب `x`
`Real(x)`	يعيد القسم الصحيح
`Imag(x)`	يعيد القسم التخيلي
`Abs(x)`	يعيد طويلة العدد المركب
`Angle(x)`	يعيد زاوية العدد المركب بالراديان
`Isreal(x)`	يعيد القيمة المنطقية `1` إذا كان العدد `x` حقيقيا فقط والقيمة المنطقية `0` إذا كان للعدد `x` جزءا تخيلياً

الرسم البياني

تعتبر إمكانية الرسم البياني للدوال الرياضية بسهولة من الميزات الهامة في نظام ماتلاب، حيث يتيح النظام الفرصة للمستخدمين لرسم المنحنيات ثنائية وثلاثية الأبعاد بإدخال تعلميات بسيطة والحصول على نتائج مرضية جداً وقابلة للتعديل بشكل مرن جداً، ومن جهة أخرى لو قارنا الجهد المبذول في ماتلاب مع الجهد الواجب بذله في لغة برمجة ما لوجدنا فرقاً هائلاً، فالكود اللازم في لغة البرمجة الأخرى سيكون كبيراً ومعقداً وتحتاج كتابته إلى الخبرة البرمجية الكبيرة نسبياً أما إذا قارنا النتائج مع النتائج الممكن الحصول عليها من برامج أخرى لرسم المنحنيات فسنجد فرقاً كبيراً أيضاً، فتلك البرامج عادة لا تتيح فرصة كبيرة للتعديل المرن للمنحنيات المرسومة وبهذا يكون نظام ماتلاب وسطاً من حيث كونه تطبيقاً قابلاً للبرمجة.

هناك أيضا أشكال لا تندرج تحت تسمية منحنيات بيانية كالمخططات الإحصائية والأشكال الهندسية، وهي أيضاً قابلة للرسم ولها أوامر خاصة بها.

يمكن أيضا تصدير الأشكال المرسومة إلى خارج نظام ماتلاب على شكل صور لاستخدامها في أمكنة اخرى كملفات الوثائق والكتب والمقالات والعروض التقديمية.

◄ رسم المنحنيات ثنائية الأبعاد

المنحني ثنائي البعد هو الخط البياني لدالة تتبع متغيراً واحداً وتعرف كمايلي:

$$y = f(x)$$

حيث x هو المتغير، أما إذا كانت الدالة تتبع عدة متغيرات فيمكن التعبير عنها بيانياً بعدة منحنيات ثنائية البعد وفي كل مرة يعتبر أحد المتحولات متغيراً وتعطى بقية المتحولات قيماً ثابتة لتعامل كثوابت عددية وعلى سبيل المثال لتكن لدينا الدالة المعرفة كمايلي:

$$k = x^2 + xy + xz^2$$

هذه الدالة تتبع المتغرات الثلاثة x,y,z وللتعبير عنها بيانياً يجب إعطاء قيم ثابتة لمتحولين منهما ورسم الدالة بالنسبة للمتحول الثالث على أنه المتغير، ثم إعطاء قيم أخرى للمتحولين ورسم الدالة ثانيةً وثالثةً وهكذا.

لرسم دوال من هذا النوع علينا أن نستخدم مجموعة من الأدوات منها أدوات الرسم ومنها أدوات برمجية. سوف أناقش هذه المسألة لاحقاً بعد التعرف على الأدوات البرمجية أما هنا فسوف أتعرض لأدوات الرسم في المستوي للدوال المعرفة كتوابع لمتغير واحد فقط.

عندما نريد رسم دالة y تتبع متغيراً x علينا البدء أولاً بتحديد مجال قيم مجموعة المنطلق أي مجال قيم المتغير x وبما أن الرسم في ماتلاب سيكون نقطياً فهذا المجال يجب أن يكون مجموعة من القيم يتم تخزينها في متجه أفقي باستخدام القيمة الأولى والقيمة النهائية والخطوة كمايلي:

X=[-10:1:10]

تستخدم هذه الطريقة في أغلب الأحيان، أما إذا كنا مهتمين بتحديد عدد محدد من القيم فلا داعي لحساب الخطوة ونستخدم الأمر `linspace(initial,final,count)` حيث ندخل القيمة الابتدائية والقيمة النهائية وعدد القيم في المتجه المراد إنشاؤه، ويتولى ماتلاب حساب الخطوة كمايلي:

X=linspace(-10,10,10)

في هذا المثال سيتم توليد 10 قيم تبدأ بالقيمة 10- وتنهي بالقيمة 10 وبخطوة ثابتة بين كل قيمتين يقوم ماتلاب بحسابها ولهذا ندعو هذا المجال من القيم مجالاً خطياً حيث يوجد مجال آخر هو المجال اللوغاريتمي الذي يمكن إنشاؤه بواسطة الامر `logspace(initial,final,count)` حيث يعمل بطريقة مشابهة للأمر `linspae` عدا أن الخطوة هنا هي خطوة لوغاريمية وليست ثابتة كمايلي:

x=logspace(1,3,100)

في هذا المثال يتم إنشاء متجه أفقي يحوي 100 قيمة تبدأ بالقيمة 10^1 وتنتهي بالقيمة 10^3

الخطوة الثانية في الرسم هي استخدام أحد أوامر الرسم ثنائي البعد، واختيار هذا الأمر يعلق بعدة عوامل حسب ما نريد أن نرسم، هذه الأوامر هي المبينة في الجدول التالي:

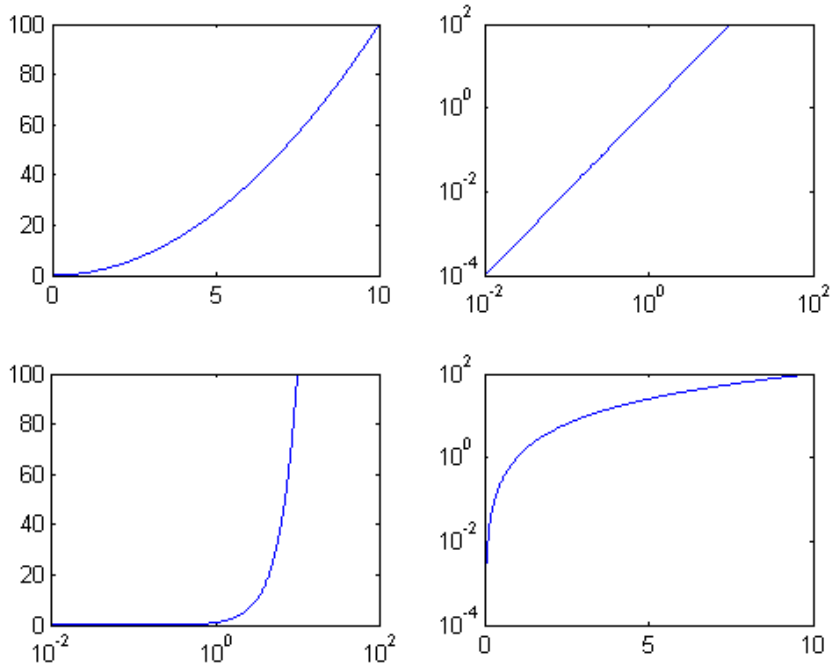
الأمر	الوصف
Plot	يستخدم لرسم منحني ثنائي البعد في مستوي ذي تدريج خطي باتجاه المحورين.
Semilogx	يستخدم لرسم منحني ثنائي البعد في مستوي ذي تدريج خطي باتجاه المحور OY ولوغاريتمي باتجاه المحور OX.
Semilogy	يستخدم لرسم منحني ثنائي البعد في مستوي ذي تدريج خطي باتجاه المحور OX ولوغاريتمي باتجاه المحور OY.
Loglog	يستخدم لرسم منحني ثنائي البعد في مستوي ذي تدريج لوغاريتمي باتجاه المحورين.

بالطبع يجب تحديد علاقة الربط بين متغير الدالة والدالة وهذه العلاقة يمكن تعريفها قبل أمر الرسم أو ضمنه كما في المثال التالي:

```
x=[0:0.1:10];
plot(x,x.^2);
loglog(x,x.^2)
```

semilogx(x,x.^2)
semilogy(x,x.^2)

الشكل 7 يظهر نتيجة استخدام الأوامر الأربعة السابقة.



الشكل 7: الرسم ثنائي البعد

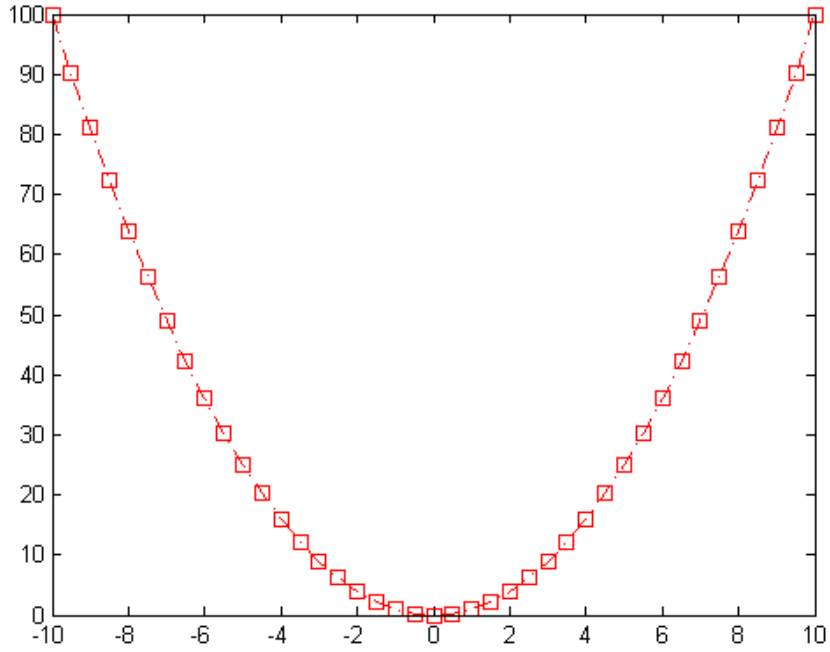
◄ خصائص خط الرسم

في الحالة الافتراضية يقوم ماتلاب برسم المنحني البياني بشكل خط مستمر أزرق اللون بين النقاط المحددة بالمتجهين المدخلين في أمر الرسم `plot` والحالة الافتراضية هذه يمكن تغييرها بواسطة إعلام ماتلاب بالخصائص المرغوبة عن طريق الأمر `plot` نفسه أو عن طريق الأمر `set`

أول الخصائص التي يمكن التحكم بها هي نمط الخط وشكل علامة النقطة ولون الخط والعلامة، وهذه الخصائص يمكن إدخال رموز خاصة بها بشكل نص مرتب كبارامتر ثالث في الأمر `plot` كما في المثال التالي:

```
x=[-10:0.5:10];y=x.^2;
plot(x,y,'-.rs')
```

في هذه الحالة سيتم رسم خط منقط باللون الأحمر وشكل علامة النقطة فيه هو المربع. إذن يجب إدخال رمز نمط الخط يليه مباشرة رمز اللون ويليهما رمز شكل علامة النقطة، كل ذلك ضمن إشارتي الاقتباس المفردتين للإدخال كنص. والنتيجة ستكون كما في الشكل 8



الشكل 8

هناك طريقة أخرى لإدخال هذه الخصائص وخصائص أخرى عن طريق إدخال سلسلة من الأزواج المرتبة المكونة من اسم الخاصية وقيمتها كما في الشكل العام التالي:

Plot(x,y,'property name','property value','property name','property value', …)

بالنسبة لأسماء الخصائص الثلاث السابقة فهي LineStyle لنمط الخط وMarker لشكل النقطة وColor للون، ويمكن الحصول على الشكل 8 بواسطة التعليمة البديلة التالية:

```
plot(x,x.^2,'LineStyle','-.','Marker','s','color','r')
```

في الجداول التالية نجد الأنماط الممكنة لخط الرسم في الجدول 1 والأشكال الممكنة للنقطة في الجدول 2 والألوان المتاحة ضمن هذه الطريقة من التعيين في الجدول 3

نمط الخط	
النمط	الرمز
مستمر (افتراضي)	-
مقطع	--
منقط	:
مقطع ومنقط	-.

الجدول 1 أنماط خط الرسم

شكل العلامة	
الشكل	الرمز
إشارة الجمع	+
دائرة	O
نجمة الضرب	*
نقطة	.
إشارة الضرب	X
مربع	S
معين	D
مثلث يشير للأعلى	^
مثلث يشير للأسفل	V
مثلث يشير لليمين	>
مثلث يشير لليسار	<
نجمة خماسية	P
نجمة سداسية	H

الجدول 2الأشكال الممكنة للنقطة

لون الخط		
اللون	الرمز	الاسم الكامل
أحمر	r	Red
أخضر	g	Green
أزرق	b	Blue
تركوازي	c	Cyan
قرمزي	m	Magenta
أصفر	y	Yellow
أسود	k	Black
أبيض	w	White

الجدول 3 الألوان

◄ الألوان الاخرى

في الطريقة السابقة لتعيين اللون استخدمنا رمز اللون وهو حرف من أحرف اسم اللون، يمكن أيضا استخدام الاسم الكامل للون كما في الجدول 3 حيث يبين الجدول أسماء الألوان الأساسية والمشهورة، في هذه الطريقة لا يمكن استخدام ألوان أخرى ولكن هناك طريقة ثانية في تعيين اللون هي طريقة مزج الألوان الأساسية والتي هي الأحمر والأخضر والأزرق، يتم المزج عن طريق إدخال نسب هذه الألوان بشكل

مرتب في متجه أفقي يحوي ثلاثة قيم بين الصفر والواحد. على سبيل المثال يمكن استخدام أحد تدرجات اللون الرمادي بإدخال القيمة 0.5 لجميع النسب كما في المثال التالي:

```
plot(x,x.^2,'LineStyle','-.','Marker','s','color',[0.5 0.5 0.5])
```

◄ خصائص أخرى

هناك خصائص أخرى عديدة من الممكن ضبطها لخط الرسم لتغيير قيمها الافتراضية عن طريق إدخالها كأزواج من أسماء الخواص والقيم، ومن أبرز تلك الخصائص مايلي:

LineWidth تحدد هذه الخاصة سماكة خط الرسم مقاسة بالنقطة point حيث تعادل واحدة النقطة جزءاً واحداً من اثنين وسبعين جزءاً من البوصة(1point=1/72 inch=2.54cm). السماكة الافتراضية لخط الرسم هي نصف نقطة.

MarkerEdgeColor تحدد هذه الخاصة لون خط المحيط للنقط المرسومة بشكل هندسي مثل الدائرة أو المثلث أي تلك التي لها خط محيط، ويمكن لهذه الخاصة أن تضبط على قيمة لونية ما باستخدام الرمز اللوني أو اسم اللون أو متجه النسب اللونية، يمكن أيضاً ضبط الخاصة على القيمة auto لتأخذ لون خط الرسم المحدد بالخاصة color

MarkerFaceColor تحدد هذه الخاصة لون التعبئة للنقط المرسومة بأشكال قابلة للتعبئة (لها خط محيط) وتأخذ قيماً لونية كما في الخاصة السابقة، أو القيمة auto لجعل التعبئة من نفس لون خلفية الرسم أو من نفس لون منطقة الرسم (المحاور) في حال كان لها لون مختلف عن خلفية الرسم.

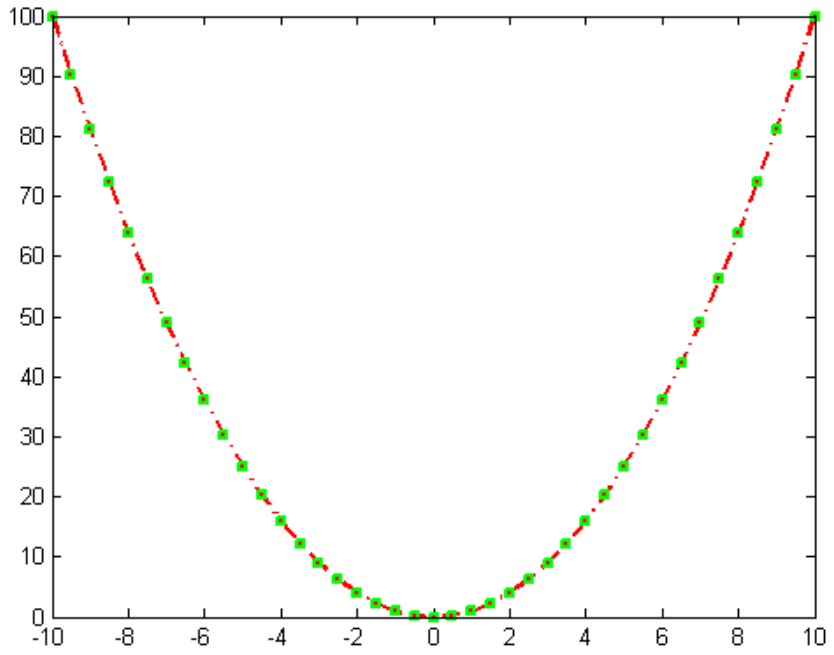
MarkerSize تحدد حجم النقطة البيانية المرسومة مقاسة بالنقطة، والقيمة الافتراضية لها هي ست نقاط.

مثال:

ليكن لدينا التعليمات التالية:

```
>> x=[-10:0.5:10];
>> y=x.^2;
>> plot(x,y,'-.rs',...
'Linewidth',2,...
'markeredgecolor','g',...
'markerfacecolor','r',...
'markersize',3)
```

هذه التعليمات سوف تنتج لنا الخط المبين في الشكل 9

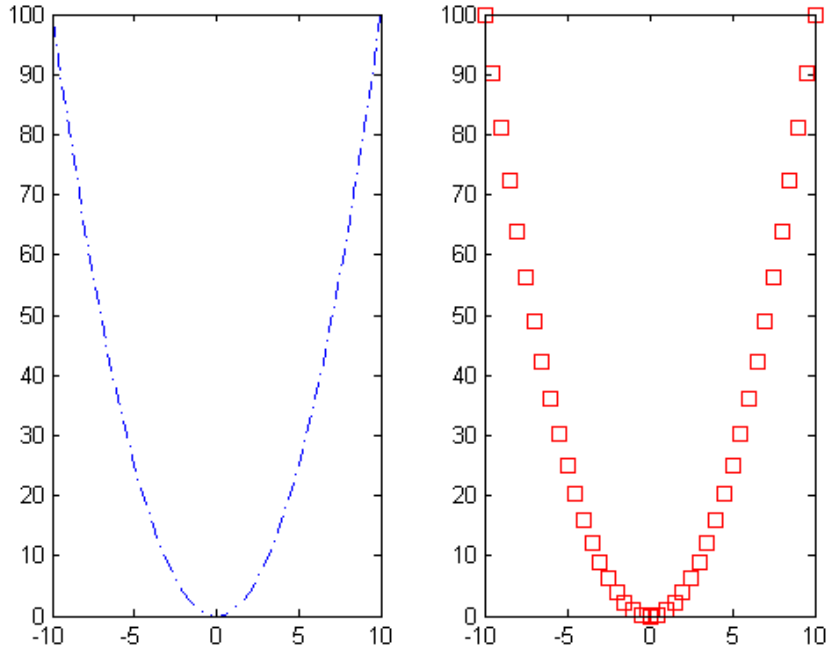


الشكل 9 خصائص الرسم البياني

يمكن أيضاً رسم الخط البياني بدون علامات النقط أو رسم علامات النقط بدون الخط الواصل بين تلك النقط، وكل ما ينبغي فعله هو حذف قيم الخصائص اللازمة كما في المثال التالي:

```
>> x=[-10:0.5:10];
>> y=x.^2;
>> plot(x,y,'-.')
>> plot(x,y,'rs')
```

في أمر الرسم الأول في السطر الثالث سيقوم ماتلاب برسم الخط البياني مقطعا ومنقطا دون علامات، أما في أمر الرسم التالي فيقوم برسم العلامات بدون الخط، والنتائج الحاصلة كما في الشكل 10



الشكل 10 رسم العلامات والرسم بدون علامات

◄ رسم عدة خطوط على شكل واحد

في بعض الحالات نحتاج لرسم عدة خطوط بيانية على رسم واحد ولعمل هذا يوجد طريقتان، في الطريقة الأولى يتم ذكر الأزواج من المتغير والدالة تباعاً في الأمر `plot` كما في المثال التالي:

```
>> x=[-10:0.5:10];
>> y=x.^2;z=x.^2+15;k=x.^2+30;
>> plot(x,y,x,z,x,k)
```

يقوم ماتلاب برسم الخطوط الثلاثة على رسم واحد وبلون مختلف لكل خط. هذه الطريقة جيدة في بعض الحالات ولكنها تصبح مزعجة في حالات أخرى فعند الرغبة في إسناد خصائص مختلفة ستكون هذه الطريقة غير مجدية، ولهذا تستخدم طريقة أخرى يتم فيها استخدام الأمر `hold` متبوعا بكلمة `on` أو كلمة `off` لمسك أو إفلات الرسم الحالي، ونعني بالمسك الاحتفاظ به كرسم حالي ليتم الرسم عليه في الأمر التالي دون حذف محتوياته، والإفلات هو تحريره من قبضة المسك ليتم الرسم عليه في الأمر التالي بعد حذف محتوياته وهي الحالة الافتراضية حيث يقوم ماتلاب عادة بحذف كافة محتويات الرسم الحالي ثم رسم الخط أو الخطوط عليه.

في التعلميات التالية سنقوم برسم الخطوط الثلاثة السابقة نفسها ثم إفلات الرسم الحالي:

```
>> x=[-10:0.5:10];
>> y=x.^2;z=x.^2+15;k=x.^2+30;
>> plot(x,y)
>>hold on
>>plot(x,z)
>>plot(x,k)
>>hold off
```

سوف نحصل على النتيجة نفسها عدا عن أن ماتلاب لا يقوم يتغيير الألوان هذه المرة.

علينا أن نعرف هنا مصطلح الرسم الحالي، هذا الرسم الحالي هو منطقة الرسم في النافذة التي يقوم ماتلاب بالرسم عليها عند تلقيه الأمر وهي النافذة المستخدمة أخيرا فإن لم تكن هناك نافذة مفتوحة للرسم وتم إعطاء أمر رسم يقوم ماتلاب بفتح نافذة جديدة للرسم ويجعلها نافذة الرسم الحالي (الفعال أو النشط) ثم ينفذ أمر الرسم عليها، أما إن كانت هناك نافذة مفتوحة تم الرسم عليها منذ قليل فيقوم ماتلاب بالرسم عليها نفسها باعتبارها نافذة الرسم الحالي الوحيدة، أما إذا كانت هناك نافذتان أو أكثر فالنافذة التي استخدمت أخيرا هي نافذة الرسم الحالي والتي سيقوم ماتلاب بالرسم عليها. هذا يقودنا إلى التساؤل التالي: كيف يمكن الرسم في نافذة غير المفتوحة إذا كان ماتلاب سيقوم بالرسم في المفتوحة الوحيدة أصلا؟ والجواب هو أن هناك أمراً يفتح نافذة رسم جديدة لتكون هي الحالية التي ستتلقى الأوامر التالية للرسم وهذا الامر هو `figure` وهنا يطرح سؤال آخر نفسه: كيف يمكن الرسم في النافذة الأولى بعد

أن تم جعل الثانية هي النشطة؟ يتم ذلك باستخدام ما يسمى مقبض منطقة الرسم ولاستخدامه يجب أن يكون معروفاً ولهذا يجب إجراء خطوة استباقية قبل فقدان إمكانية العودة للرسم القديم، هذه الخطوة الاستباقية هي حفظ مقبض منطقة الرسم في متحول وهي كمايلي:

```
>> plot(h,x,y)
>> h=gca
h =
  171.0111
>>figure
>> plot(x,z)
>> hold on
>> plot(h,x,k)
```

حيث يشير المتغير gca إلى منطقة الرسم الحالية (graphic current axes) وعند إسناده إلى المتحول h يتم الاحتفاظ به هناك (في h) إلى حين الحاجة إليه، وعند الرسم من جديد باستخدام الأمر figure يتحول gca ليدل على الرسم الحالي الجديد (لم يعد مساويا h) ثم يتم استخدام المقبض h في الأمر plot قبل بقية البارامترات جميعاً للدلالة على أن الرسم يجب أن يكون في منطقة الرسم المشار إليها بواسطة هذا المقبض h، وبالطبع يتم حذف محتويات النافذة إن لم نستخدم الأمر hold on.

يمكن رسم أكثر من خط بياني على منطقة رسم واحدة بطريقة أخرى كمايلي:

```
>>plot(x,y,x,z)
```

وفي هذه الحالة يمكن إسناد خصائص كل خط على حده أو ضبط خصائص مشتركة للخطين (يمكن رسم أكثر من خطين)، ولكن لا يصح هذا الكلام عن كل الخصائص وإنما فقط الخصائص الثلاث التي يمكن ضبطها في تعبير نصي واحد وهي نمط خط الرسم ولون الخط وشكل علامة النقطة، فإذا أردنا مثلاً جعل لون الخطين هو اللون الأحمر ندخل هذه الخاصة بعد ذكر أزواج المتحولات كمايلي:

```
>>plot(x,y,x,z,'—r')
```

أما إذا أردنا جعل لون الخط الأول أحمر والثاني أزرق فندخل الخاصة مع قيمتها المرغوبة بعد ذكر زوج المتحولات لكل خط كمايلي:

```
>>plot(x,y,'—r',x,z,'color','--b')
```

ينبغي الانتباه أيضا إلى أن متحول المقبض الذي نحصل عليه في عملية إسناد الرسم هنا هو متجه عمودي عدد عناصره يساوي عدد الخطوط البيانية والمثال التالي يوضح استخدام ذلك في ضبط الخصائص.

```
>> x=[-10:.5:10];
>> y=x.^2;z=cos(x);
>> h=plot(x,y,x,z)
h =
  172.0057
  173.0048
>> set(h(1),'color','r');
>> set(h(2),'color','b');
```

◄ الرسم على عدة أشكال وضبط خصائصها

يفيدنا المقبض الذي تكلمنا عنه في الفقرة السابقة في أمر آخر هام جداً وهو ضبط خصائص الخط البياني بعد رسمه بواسطة الأمر `set` الذي يقبل مقبض العنصر المراد تغيير خصائصه (ليس بالضرورة خط الرسم ولكننا هنا بصدد الحديث عن خط الرسم) كبارامتر أول ثم أزواج من أسماء الخصائص وقيمها.

في المثال التالي نقوم برسم خطين بيانيين على شكل واحد وإسناد كل خط بياني إلى مقبض ثم ضبط خاصية اللون لكل خط بعد تنفيذ الرسم.

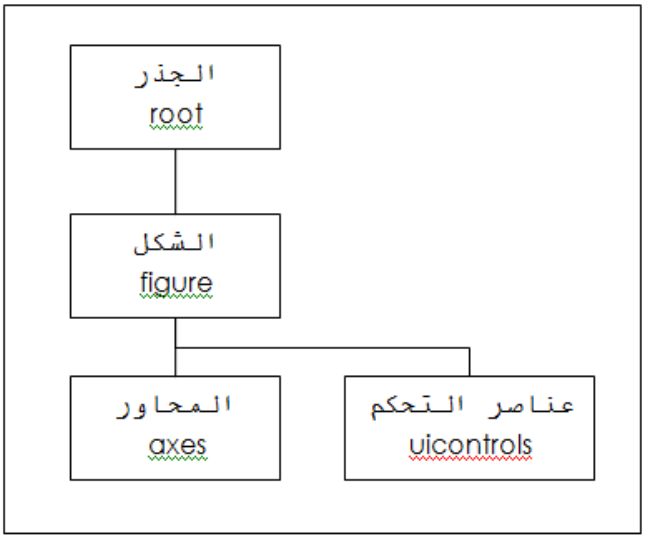
```
>> x=[-10:0.5:10];
>> y=x.^2;z=x.^2+30;
>> hy=plot(x,y);hold on;hz=plot(x,z);
>> set(hy,'color','r');set(hz,'color','g');
```

◄ المزيد عن الرسم في ماتلاب

حتى الآن رأينا كيف يمكن رسم الخطوط البيانية على شكل واحد أو على عدة أشكال، ولكي نتابع الحديث دون عقبات لا بد من توضيح بعض المفاهيم بشكل بسيط ليسهل فهم ما يفعله ماتلاب عند تنفيذ الأوامر.

في الواقع يتعامل ماتلاب مع الرسوم على أنها أغراض (سوف نتكلم عن مفهوم الغرض لاحقاً ولكن يكفي الآن أن نعرف أن الغرض هو عنصر له مواصفات خاصة به وينتمي إلى فئة معينة وهو يشبه في حياتنا أن نقول أن السيارة هي فئة من الأغراض والسيارة التي أملكها هي غرض من فئة السيارات) والأغراض الرسومية لها بنية هرمية تبدأ بالغرض الجذر root والذي هو شاشة الحاسب الذي نعمل عليه وهذا الغرض يعتبر جذراً لأنه لا ينتمي إلى أي غرض آخر فهو الأعلى في التصنيف ثم يأتي في المرتبة الثانية الغرض المسمى الشكل figure وهو يقابل النافذة في نظام التشغيل ويندوز حيث أن كل شكل هو نافذة وبهذا أصبح لدينا جذر وحيد هو الشاشة وينتمي إليه عدد من الأغراض من فئة الشكل، نقول عن الغرض الجذر هنا أنه الغرض الأب للغرض الشكل ونقول عن الشكل بأنه الغرض الابن للغرض الجذر، الأغراض الأبناء للغرض الشكل تنتمي إلى فئتين رئيستين هما فئة عناصر التحكم الرسومية uicontrols وفئة محاور الرسم البياني axes. ما يهمنا في هذا السياق هو الأخير بما أننا نتكلم عن الرسم البياني أما عناصر التحكم الرسومية فهي

تستخدم في البرمجة ذات الواجهات الرسومية. الشكل 11 يبين البنية الهرمية التي تحدثنا عنها.



الشكل 11 البنية الهرمية للرسوميات

الآن أصبح بالإمكان أن نتحدث أكثر عن إنشاء العناصر الرسومية وضبط خصائصها دون أن يحصل التباس، وفيمايلي بعض الأوامر الهامة المستخدمة في ذلك:

الأمر `figure` يستخدم لإنشاء شكل جديد ويمكن أن يمرر له بعض الخصائص مباشرة أو تترك هذه المهمة للأمر `set` لاحقاً بشرط أن نسند هذا الأمر إلى متحول

مقبض ويتم ذلك بأن نقوم بعملية الإسناد كالعادة باستخدام معامل الإسناد (=) ونكتب اسم المتحول قبله والأمر figure بعده فيقوم ماتلاب بإنشاء شكل جديد مقبضه هو المتحول المسمى على يسار معامل الإسناد، يمكن أيضاً استخدام الأمر figure لغرض آخر وهو جعل شكل محدد هو الشكل الحالي (النشط أو الفعال) وذلك بإدخال الرقم الصحيح الدال على الشكل كبارامتر للأمر.

المثال التالي يبين كيفية استخدام الأمر.

```
>>figure
>>h=figure
>>figure(2)
```

هناك طريقة أخرى لاستخدام الأمر figure لإنشاء شكل جديد وهي تمرير رقم صحيح لا يدل على شكل موجود كبارامتر.

الأمر axes يستخدم هذا الأمر لإنشاء منطقة للرسم البياني على الشكل الحالي، أما إذا كنا نريد إنشاء منطقة الرسم هذه على شكل غير الحالي فينبغي جعله حالياً أولاً.

الأمر set يستخدم هذا الأمر لضبط خصائص أي عنصر يتم إدخاله إليه كبارامتر أول ثم يتم إدخال أزواج من أسماء الخصائص وقيمها وبالطبع لا يتسع الكتاب لذكر كافة الخصائص ولكن سوف أذكر بعض الأمثلة. حتى الآن يمكننا استخدام هذا الأمر لضبط خصائص الشكل ومنطقة المحاور والخط البياني.

الأمر get يستخدم لمعرفة خصائص عنصر محدد وذلك بتمرير متحول المقبض لذلك العنصر كبارامتر أول ثم اسم الخاصة المراد معرفة قيمتها كبارامتر ثاني، يمكن أيضا معرفة كافة الخصائص بتمرير متحول المقبض فقط وهي حيلة جيدة لمعرفة الخصائص الممكن ضبطها بواسطة الأمر set دون الرجوع إلى وثائق المساعدة.

الأمر gcf يعيد لنا قيمة المقبض للشكل الحالي ويمكن إسناده لمتحول ما، وهو مفيد في حالة أننا لم نسند ذلك الشكل لمتحول عند إنشائه حيث يقوم ماتلاب بإنشاء مقبض له ولكن دون إظهاره، ويستخدم بعملية إسناد عادية.

الأمر gca يعيد لنا قيمة المقبض لمنطقة المحاور الحالية (منطقة المحاور على الشكل الحالي) ويستخدم بشكل مشابه للأمر gcf بعملية إسناد عادية.

الأمر clf يستخدم لمسح محتويات الشكل الحالي، يمكن تمرير رقم مقبض الشكل أيضاً لمسح محتوياته بدلاً من الشكل الحالي.

◄ خصائص الشكل

سوف أذكر هنا بعض أهم الخصائص للشكل مع ذكر أمثلة عن ضبطها والأمثلة هنا ليست لبيان طريقة استخدام الأمر set أو الأمر get فهي طريقة واضحة ولا تحتاج إلى بيان ولكن لإظهار الطرق المختلفة لضبط تلك الخصائص.

Children تعيد هذه الخاصة متجها عموديا يحوي أرقام (مقابض) العناصر الأبناء للشكل المعني (هذه الأبناء قد تكون محاور رسم بياني أو عناصر تحكم).

Color تستخدم لتغيير لون الشكل ويقصد بلون الشكل لون المساحة المحصورة ضمن إطار النافذة ولكن لا يؤثر هذا على لون منطقة المحاور لأن الأخيرة تقع فوق الشكل. يمكن استخدام قيمة اللون بالطرق الثلاث التي تحدثنا عنها سابقاً.

Currentaxes تستخدم هذه الخاصة عندما يكون هناك أكثر من منطقة محاور على الشكل لمعرفة أي منها هي الحالية أو لتغيير الحالية لرسم خط بياني عليها.

Name هذه الخاصة تحدد النص الذي سيظهر في شريط عنوان نافذة الشكل، وفي الحالة الافتراضية يكون هذا النص خاليا حيث تظهر كلمة figure متبوعة برقم الشكل فقط أما في حال إسناد قيمة لهذه الخاصة فتظهر هذه القيمة بعد رقم الشكل.

Nextplot تحدد هذه الخاصة الشكل الذي يجب أن يتم عليه الرسم في الأمر التالي وهي تأخذ إحدى القيم التالية:

New تعني أن أمر الرسم التالي يجب أن يتم تنفيذه على شكل جديد.

Add تعني أن أمر الرسم التالي يجب أن يستخدم الشكل الحالي ليضيف الرسم الجديد إليه، وهي القيمة الافتراضية.

Replace تعني أن أمر الرسم التالي سوف يستخدم الشكل الحالي بعد إزالة كافة محتوياته وإعادة كافة خصائصه إلى قيمها الافتراضية.

Replacechildren تعني استخدام الشكل الحالي بعد إزالة كافة محتوياته دون تغيير الخصائص.

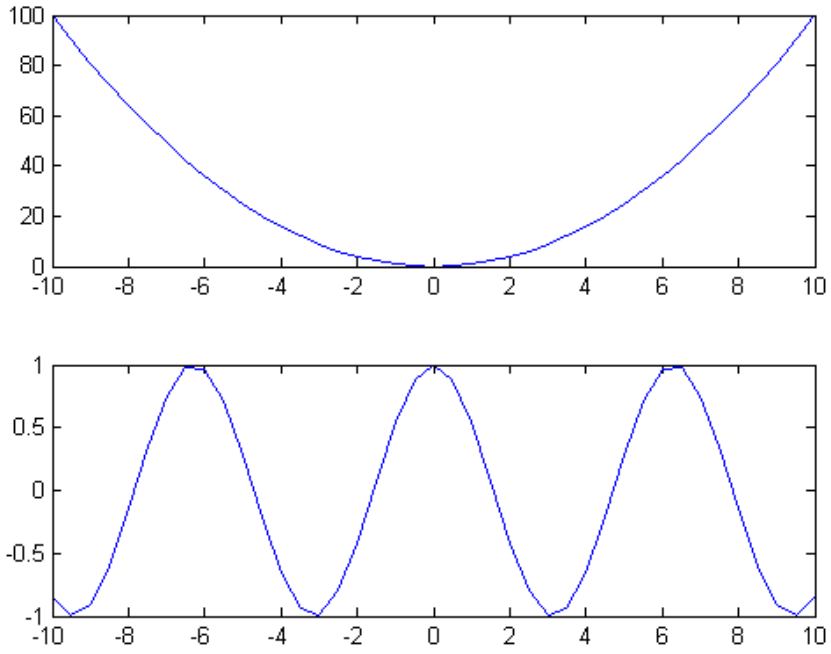
تبدو القيمتان add و replacechildren وكأنهما متماثلتان وهما كذلك عند وجود رسم واحد على الشكل أما في حال وجود أكثر من رسم فالأمر مختلف، فالأولى تضيف الرسم الجديد أما الثانية فتمحو الرسوم الموجودة وتضيف الجديد بدلاً عنها.

في المثال التالي نقوم بتعريف ثلاثة دوال ثم نرسم الخطين البيانيين للأولى والثانية على شكل واحد باستخدام الأمر subplot الذي سوف نشرحه لاحقاً (ما يهمنا الآن هو بيان الفرق بين قيمتي الخاصة) ثم نقوم برسم الخط البياني للدالة الثالثة بطريقتين كمايلي:

أولا نقوم بتنفيذ الأوامر التالية:

```
>> x=[-10:.5:10];
>> y=x.^2;z=cos(x);k=-x.^2;
>> hf=figure;
>> subplot(2,1,1);
>> plot(x,y);
>> subplot(2,1,2);
>> plot(x,z);
```

ستكون النتيجة رسم الدالتين y و z على شكل واحد كما في الشكل 12

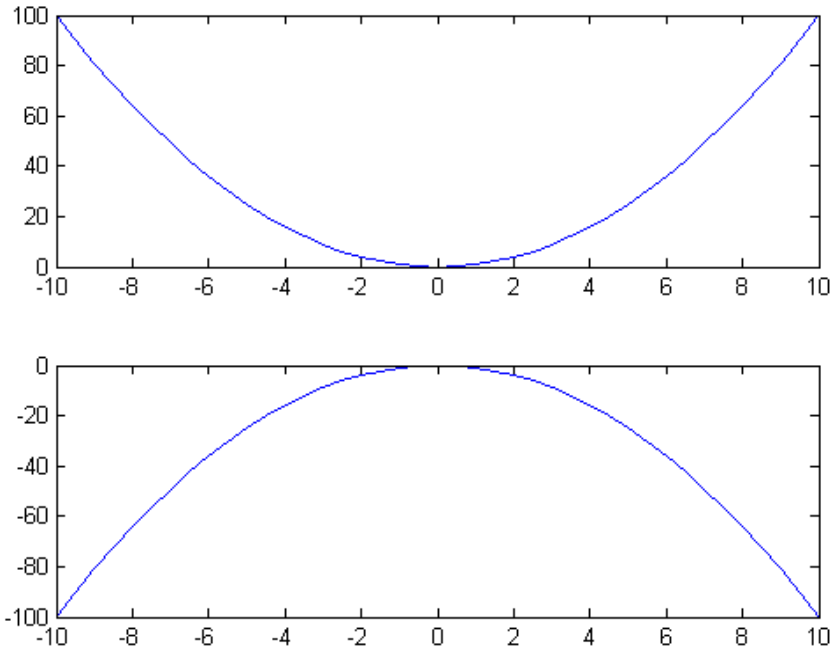


الشكل 12

بعد ذلك ندخل الأمر التالي:

>> plot(x,k);

سيقوم ماتلاب برسم الخط البياني للدالة k مكان الخط البياني للدالة z في النصف الأسفل من الشكل، وذلك لأن منطقة المحاور النشطة هي التي رسم عليها أخيراً أي تلك التي في الأسفل ولأن القيمة الافتراضية للخاصة nextplot للشكل هي add التي تعني إضافة الرسم إلى الشكل الحالي ولكن هذه الإضافة بالطبع سوف تؤدي إلى حذف الرسم من منطقة المحاور الحالية والرسم مكانه. النتيجة كما في الشكل 13



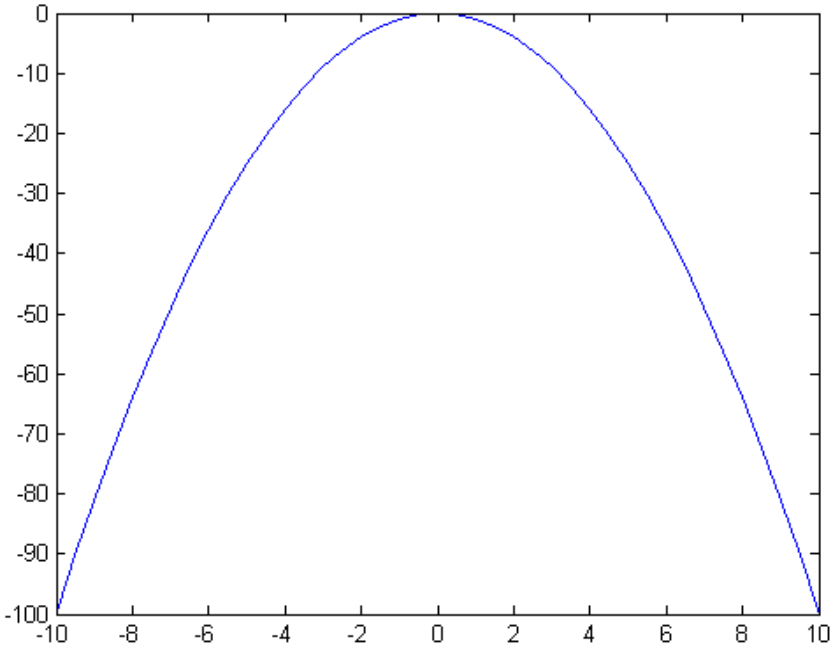
الشكل 13

الآن نغير الخاصة nextplot إلى القيمة replacechildren ثم نرسم الدالة z من جديد كمايلي:

```
>>set(gcf,'nextplot','replacechildren')

>>plot(x,z)
```

في هذه الحالة سيتم حذف كافة محتويات الشكل ورسم الخط البياني للدالة z عليه لتظهر كما في الشكل 14



الشكل 14

Position تحدد هذه الخاصة موضع نافذة الشكل وأبعادها بواسطة أربع قيم هي:

Left البعد الأفقي بين الزاوية اليسرى السفلى للشاشة والزاوية اليسرى السفلى لنافذة الشكل.

Bottom البعد العمودي بين الزاوية اليسرى السفلى للشاشة والزاوية اليسرى السفلى لنافذة الشكل.

Width عرض نافذة الشكل.

Height ارتفاع نافذة الشكل.

تقاس هذه القيم بالواحدة المحددة للقياس والواحدة الافتراضية هي البكسل ويمكن تغييرها بواسطة الخاصة units. يتم إدخال هذه القيم بشكل متجه أفقي كما في المثال التالي:

```
>> >> set(gcf,'position',[150 150 800 500])
```

◄ خصائص منطقة المحاور

منطقة المحاور أيضاً لها العديد من الخصائص وسأكتفي بذكر بعض تلك الخصائص التي يكثر استخدامها.

Color تحدد اللون المستخدم لتعبئة منطقة المحاور.

Colororder تحدد الألوان التي يجب أن تستخدم لتلوين الخطوط البيانية عند رسم أكثر من خط على منطقة محاور واحدة، وهي مصفوفة ثلاثية الأعمدة ويمكن إدخال عدد من الأسطر بحيث يعبر كل سطر منها عن لون بطريقة نسب الألوان الأساسية، يمكن الحصول على المصفوفة الافتراضية وتعديلها ثم إعادة إسنادها كما في المثال التالي:

```
>> cc=get(gca,'colororder')
cc =
         0         0    1.0000
         0    0.5000         0
    1.0000         0         0
         0    0.7500    0.7500
    0.7500         0    0.7500
    0.7500    0.7500         0
    0.2500    0.2500    0.2500
>> cc(1,1)=1;cc(1,3)=0;
>> cc(2,1)=1;cc(2,3)=1;
>> cc(3,1)=0.5;cc(3,3)=0.25;
>> set(gca,'colororder',cc)
```

Linestyleorder هذه الخاصة تحدد شكل علامة النقطة للخطوط المتعددة المرسومة على منطقة محاور واحدة وهي تدخل بشكل مصفوفة بنية structure كما في المثال التالي:

```
>> set(gca,'linestyleorder',{'-','o','*'})
```

Outerposition و position تحدد هاتان الخاصتان موضع ومساحة منطقة المحاور ضمن الشكل الحاوي عليها حيث تستخدم الثانية لتعيين حدود منطقة المحاور فقط أما الأولى فتستخدم لتعيين منطقة المحاور مع المسميات على المحاور والعنوان، وكل منهما تدخل كمتجه أفقي من أربع قيم هي كمايلي:

```
Set(h,'outerposition',[left bottom width height])
```

وتقاس بالواحدة الافتراضية (وهي هناليست البكسل بل واحدة النسب normalized) ما لم يتم تغييرها.

مثال:

```
>> figure;a1=axes;a2=axes;
>> set(a1,'outerposition',[0 0 1 0.5]);
>> set(a2,'outerposition',[0 0.5 1 0.5]);
```

في السطر الأول يتم إنشاء شكل جديد ويضاف إليه منطقتا رسم a1 و a2 وفي السطر الثاني يتم إسناد قيم الحدود الخارجية للمنطقة الأولى ثم للمنطقة الثانية في السطر الثالث، والنتيجة ستكون تقسيم نافذة الشكل عموديا إلى قسمين متساويين.

Title يتم بواسطة هذه الخاصة وضع عنوان للرسم في منطقة المحاور المعنية، ويمكن ضبط بعض الخصائص لهذا العنوان كنوع الخط وحجمه ولونه كما في المثال التالي:

```
>> title(gca,'axes title','color','r','edgecolor','b')
```

يمكن ضبط العديد من الخصائص للعنوان على أنه نص ولكن سوف أتعرض للحديث عن خصائص النص في مكان آخر.

Xlabel, ylabel, zlabel هذه الخصائص الثلاث هي تسميات المحاور الإحداثية الثلاثة وهي نصوص أيضاً ويتم استخدامها بطريقة مشابهة للخاصة title حيث نضيف النص وبعض الخصائص كما في المثال التالي:

```
>> xlabel(gca,'x values','color','r','background','y')
>> ylabel(gca,'y=x^2','color','r','background','y')
```

Xgrid, ygrid, zgrid تستخدم هذه الخصائص الثلاث لتفعيل أو إلغاء تفعيل ظهور خطوط الشبكة في أحد اتجاهات المحاور الثلاثة وهي تأخذ قيمة on أو قيمة off فقط كما يلي:

>> set(gca,'xgrid','on','ygrid','off')

هناك طريقة بديلة إذا كنا نريد تعيين قيمة كل هذه الخصائص الثلاث إلى on أو إلى off معا وهي استخدام الأمر grid فالأمر:

>>grid on

يكافئ:

>>set(gca,'xgrid','on', 'ygrid','on', 'zgrid','on')

Xlim, ylim, zlim تستخدم هذه الخصائص الثلاث لتحديد القيم الدنيا والعليا للمحاور وهذا للتحكم بظهور الرسم البياني ضمن المجال المطلوب حيث يقوم ماتلاب في الوضع الافتراضي بتعيين هذه القيم الدنيا والعليا وفقا لمجال القيم المستخدم للمتحولات في الدالة المرسومة، ومن الملاحظ أن قيم هذه الخصائص يجب أن تكون متجهات أفقية تحوي قيمتين في كل خاصة كمايلي:

>> set(gca,'xlim',[-20 20])

يمكن أيضاً تعيين القيم الدنيا والعليا للمحاور باستخدام الأمر axis كمايلي:

axis(axes handle, [xmin, xmax, ymin, ymax])

ويمكن إضافة القيم الدنيا والعليا للمحور الثالث ولكن لا يمكن الاكتفاء بمحور واحد، كذلك فعند حذف متحول المقبض لمنطقة المحاور تعتبر المنطقة الحالية كما في المثال:

>>axis(gca, [-20 20 -25 25])

بالطبع هنا يمكن استبدال gca بأي متحول مقبض آخر فالمتحول gca هو افتراضي ويمكن الاستغناء عن ذكره.

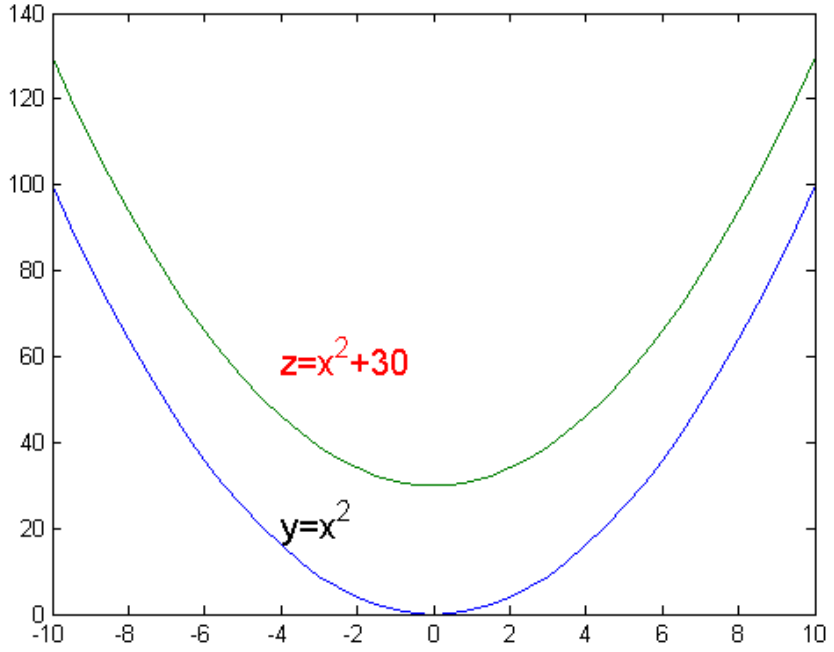
Xscale, yscale, zscale في هذه الخصائص يتم تحديد نوع التدريج المتبع على المحور المعني إن كان خطياً أو لوغاريتمياً والقيمة الافتراضية هي التدريج الخطي linear ويمكن ضبطها على القيمة اللوغاريتمية log

◄ إضافة نص على الرسم

يستخدم التابع `text` لإضافة نص على الرسم عند نقطة معينة يتم إدخال إحداثياتها الثلاث إلى هذا التابع ثم يتم إدخال النص المراد وضبط خصائصه، كما في المثال التالي:

```
>> x=[-10:.5:10];
>> y=x.^2;z=x.^2+30;
>> plot(x,y,x,z)
>> text(-4,22,'y=x^2','fontsize',15)
>> t=text(-4,60,'z=x^2+30','fontsize',15)
>> set(t,'color','r')
```

نتيجة تنفيذ هذه الأوامر ستكون كما في الشكل 15



الشكل 15 إضافة نص على الرسم البياني

استخدمنا هنا التابع `text` بطريقتين ففي الطريقة الأولى أسندنا خاصة حجم الخط فقط ولم نسند النص المستخدم إلى متحول مقبض، أما في الطريقة الثانية فقمنا بعملية الإسناد تلك لنقوم فيما بعد بتغيير خاصة اللون إلى الأحمر.

هناك خصائص أخرى عديدة للخط يمكن تغييرها وفيما يلي أهم تلك الخصائص:

Backgroundcolor لتحديد لون خلفية النص ويمكن أن يسند لهذه الخاصية أي قيمة لونية باستخدام الطرق الثلاث للقيم اللونية التي تحدثنا عنها سابقاً.

Edgecolor لتحديد لون الخط المحيط بالمستطيل الحاوي على النص وفي الحالة الافتراضية تكون قيمة هذه الخاصية هي none بمعنى أن لون الخط المحيط هو نفس لون الامتلاء backgroundcolor

Linestyle تحدد هذه الخاصية نمط الخط المحيط وهي تأخذ إحدى القيم الرمزية الأربع (- للخط المستمر، -- للخط المقطع، : للخط المنقط، .- للخط المقطع المنقط)، وبالطبع يجب أن تضبط خاصة لون الخط المحيط على قيمة ما مغايرة للون الخلفية ليصبح بالإمكان رؤية النمط.

Linewidth تحدد سماكة الخط وتأخذ قيمة عددية وهي أيضاً لا يظهر أثرها إلا بعد تحديد لون الخط المحيط. قيمة هذه الخاصة تقاس بالنقطة (1point=1/72inch)

Margin تحدد قيمة الفراغ بين النص والخط المحيط وهي تأخذ قيما عددية مقاسة بالبكسل.

مثال:

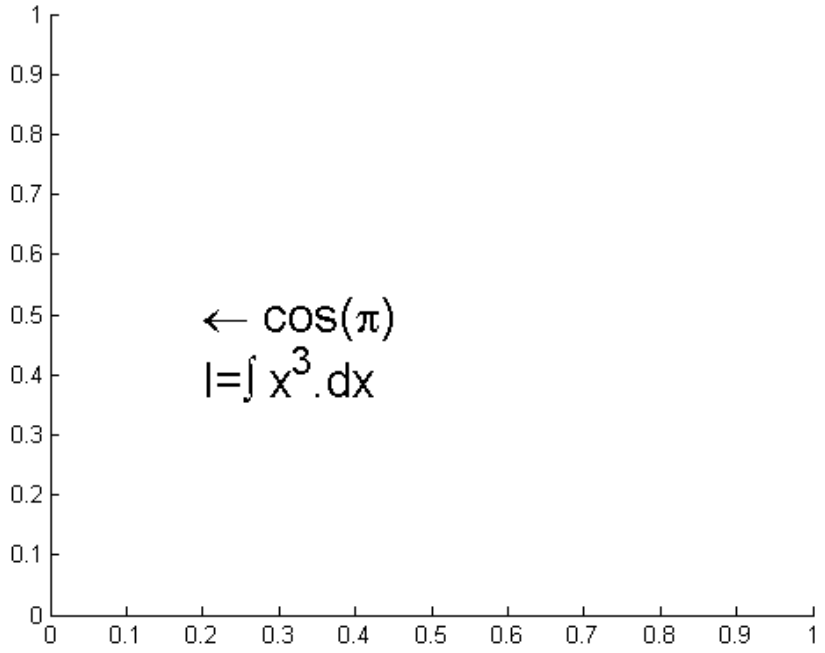
```
>> set(tt,'backgroundcolor',[0.5 0.7 0.1],...
'edgecolor', 'r',...
'linestyle',':',...
'linewidth',3,...
```

'margin',10)

هناك أحرف ورموز لا تكتب بشكل مباشر في النص وإنما لها اختصارات أو أسماء تعبر عنها كما في المثال التالي:

```
>> figure;axes;
>> text(0.2,0.5,'\leftarrow cos(\pi)','fontsize',20)
>> text(0.2,0.4,'I=\int x^3.dx','fontsize',20)
```

تظهر نتيجة تنفيذ هذه التعليمات كما في الشكل 16

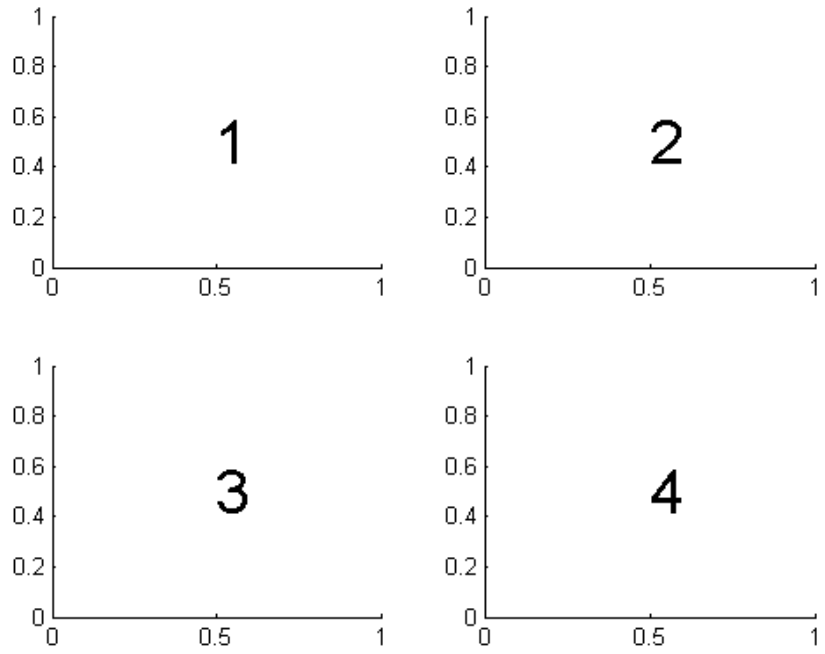


الشكل 16 كتابة الرموز على الرسم البياني

الملحق (أ) يبين الرموز المستخدمة بكثرة واختصاراتها.

◄ تقسيم نافذة الرسم

يمكن تقسيم نافذة الشكل ورسم عدة مناطق محاور عليها بأكثر من طريقة، إحدى تلك الطرق هي تحديد الخاصة `position` للمحاور عند إنشائها كما رأينا سابقاً، ولكن هناك طريقة أسهل وأسرع وهي باستخدام التابع `subplot(m, n, p)` الذي يقوم بتقسيم نافذة الشكل إلى `m` سطراً و`n` عموداً، ثم ينشئ منطقة المحاور في الخلية المحددة بالقيمة `p` أو على امتداد الخلايا المحددة بالمتجه `p` بشرط إمكانية تحقيق ذلك ويكون ترقيم الخلايا بدءاً من اليسار إلى اليمين ثم نزولاً إلى الأسفل كما يظهر الشكل 17



الشكل 17 تقسيم نافذة الشكل

تم رسم هذا الشكل في ماتلاب بواسطة الأوامر التالية:

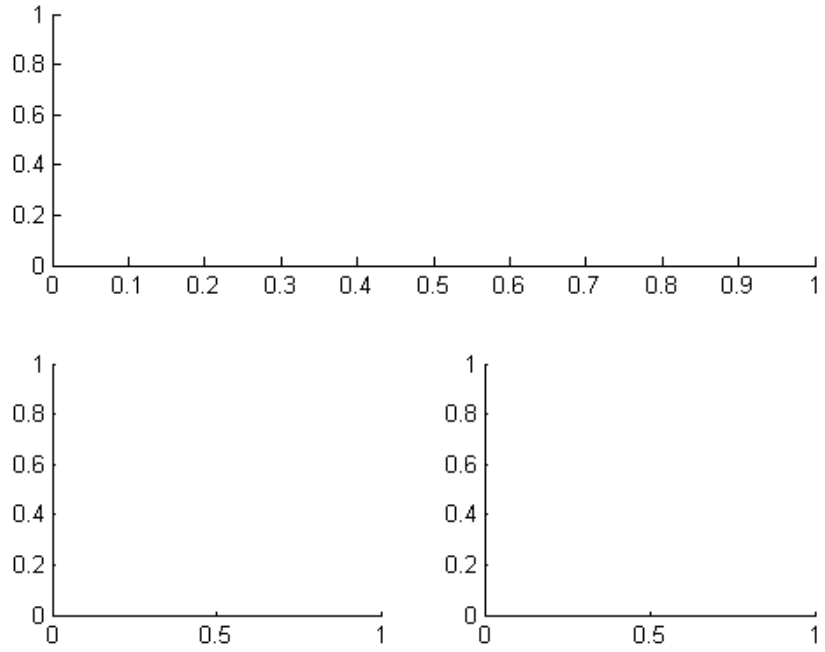
```
>> figure;
>> a1=subplot(2,2,1);
>> a2=subplot(2,2,2);
>> a3=subplot(2,2,3);
>> a4=subplot(2,2,4);
>> text(.5,.5,'1','fontsize',25,'parent',a1);
>> text(.5,.5,'2','fontsize',25,'parent',a2);
```

```
>> text(.5,.5,'3','fontsize',25,'parent',a3);
>> text(.5,.5,'4','fontsize',25,'parent',a4);
```

في السطر الأول تم إنشاء نافذة شكل جديدة، في الأسطر من الثاني إلى الخامس تم إنشاء مناطق محاور وإسنادها إلى متحولات مقابض بواسطة التابع subplot، في السطر السادس تم استخدام التابع text لكتابة الرقم 1 كنص في وسط منطقة المحاور الأولى ولتحقيق ذلك استخدمت الخاصة parent وأسندت إليها القيمة a1 وهي متحول المقبض لمنطقة المحاور الأولى ويعني هذا أن النص الذي سيتم إدراجه سيكون عنصراً ابناً لمنطقة المحاور تلك، لو لم تستخدم تلك الخاصة سيقوم ماتلاب بإدراج النص في منطقة المحاور التي تم إنشاؤها أخيراً باعتبارها منطقة المحاور الحالية (النشطة)، الأسطر الأخيرة من السابع إلى التاسع تشبه السطر السادس مع فرق أنها تكتب في مناطق المحاور الأخرى، ومن الجدير بالذكر أنه يمكن الاستغناء عن الخاصة parent في السطر التاسع لأنه يكتب في منطقة المحاور النشطة حيث أن الأسطر السابقة لا تنقل النشاط منها إلى غيرها.

يمكن أيضاً تقسيم نافذة الشكل تقسيمات أخرى كما في الأمثلة التالية:

المثال الأول: كما في الشكل 18



الشكل 18 تقسيم آخر لنافذة الشكل

يتم هذا التقسيم بواسطة إدخال التعليمات التالية:

```
>> figure;
>> a1=subplot(2,2,1:2);
>> a3=subplot(2,2,3);
>> a4=subplot(2,2,4);
```

حيث يتم إخبار ماتلاب في السطر الثاني أن منطقة المحاور a1 يراد مدها على الجزأين الأول والثاني، لا يمكن بالطبع مدها على الأجزاء الثلاثة الأولى في مثالنا هذا لان تلك الأجزاء لا تشكل إطاراً مستطيلاً، أما لمدها على الجزأين الأول والثالث فيجب إدخال الأرقام كمتجه كمايلي:

```
>> a1=subplot(2,2,[1 3]);
```

وتصلح طريقة المتجه في الحالة السابقة أيضاً، ولجعل الفكرة أكثر وضوحاً يمكنك أن تتأمل الأكواد المتكافئة التالية:

```
>> a1=subplot(2,3,1:3);
```

أو:

```
>> a1=subplot(2,3,[1 2 3]);
```

يمكن أيضاً إجراء التقسيم السابق بطريقة أخرى كمايلي:

```
>> figure;
>> a1=subplot(2,1,1);
>> a3=subplot(2,2,3);
>> a4=subplot(2,2,4);
```

في هذه المرة تم استبدال السطر الثاني فقط حيث تم تقسيم النافذة أولا إلى جزأين فقط علوي وسفلي وإنشاء منطقة المحاور في العلوي منهما، ثم أعيد تقسيم النافذة إلى أربعة أجزاء كالسابق وتم إنشاء منطقتي المحاور على الجزأين الثالث والرابع منها.

لا يؤثر هذا التقسيم المتتابع على وجود منطقة المحاور القديمة بسبب السلوك الافتراضي للتابع `subplot` حيث يبقي منطقة المحاور الموجودة دون تغيير وفي الواقع فإنه يفعل ذلك حتى لو استخدمنا نفس الجزء فهو ينشئ المنطقة الجديدة فوقه، بالطبع لا معنى لإجراء كهذا إلا إذا كنا نخطط لتغيير الخاصة `position` لاحقاً. يمكن تغيير سلوك التابع `subplot` هذا وجعله يقوم بحذف محتويات الجزء المعني إن كان عليه محتويات قبل إنشاء منطقة محاور جديدة وذلك باستخدام المعامل `reset` الذي يتم إدخاله للتابع كبارامتر كمايلي:

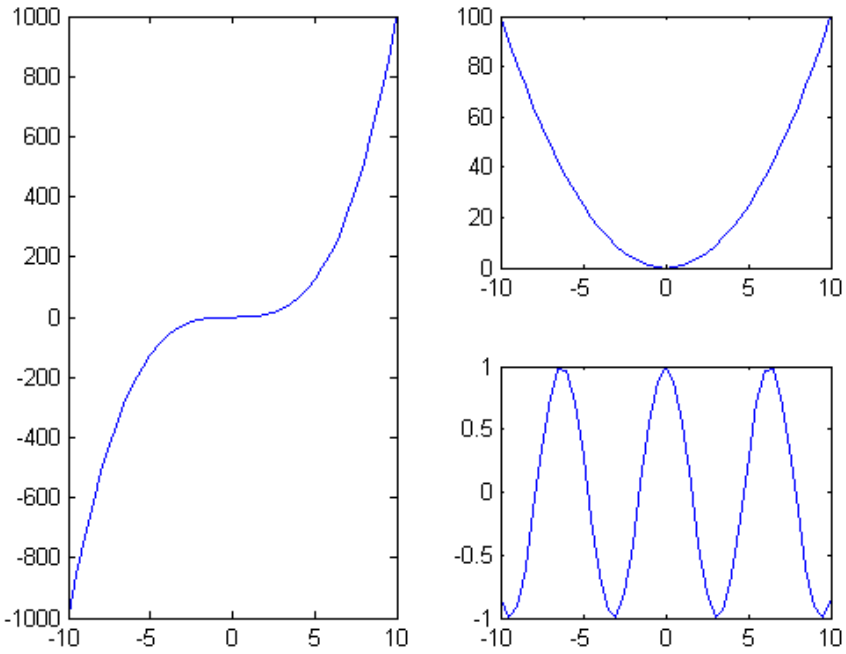
```
>> a4=subplot(2,2,4,'reset');
```

عندما نريد رسم الخطوط البيانية الآن في إحدى مناطق المحاور يجب تحديد تلك المحاور باستخدام متحول المقبض كما في مثال الشامل التالي:

```
>> x=[-10:.5:10];
>> y=x.^2;z=cos(x);k=x.^3;
>> figure;
>> a1=subplot(2,2,[1 3]);
>> a2=subplot(2,2,2);
```

```
>> a4=subplot(2,2,4);
>> plot(a1,x,k)
>> plot(a2,x,y)
>> plot(a4,x,z)
```

تظهر نتيجة التنفيذ كما في الشكل 19



الشكل 19 طريقة لرسم عدة خطوط بيانية

◄ الرسم في المستوي العقدي

يمكن استخدام التابع plot بطريقة أخرى بسيطة بأن ندخل له متحول دالة فقط دون المتحول المتغير وفي هذه الحالة سيقوم ماتلاب برسم الخط البياني للمتحول الذي تم إدخاله مقابل أدلة العناصر، وليكن لدينا على سبيل المثال متجه أفقي y يحوي مربعات الأعداد من الواحد إلى العدد 8، أدلة عناصر هذا المتجه هي الأعداد من الواحد إلى العدد 8 ولذلك فالعلاقة بين رقم الدليل وقيمة العنصر الموافق من المتجه y هي علاقة تربيعية من الشكل $y=x^2$ ويكون كل من الأمرين التاليين متكافئان:

```
>>y=[1 4 9 16 25 36 49 64];
>>plot(y)
```

أو:

```
>>x=[1:8];
>>y=[1 4 9 16 25 36 49 64];
>>plot(x,y)
```

هذا الكلام صحيح تماما عند العمل في المستوي الحقيقي وعندما تكون جميع القيم حقيقية.

عند كون المتجه y يحوي قيما عقدية ولو قيمة واحدة على الأقل يقوم ماتلاب برسم الخط البياني للدالة التي متغيرها هو القسم الحقيقي ومتغيرها التابع هو القسم التخيلي،

أي كأنه يرسم الخط البياني المعبر عن الأعداد في المستوي العقدي، وهذا عندما نستخدم التابع plot بشكله الأخير أما إذا استخدم التابع plot بأي شكل آخر كما سبق فيهمل ماتلاب القسم التخيلي.

ليكن لدينا المثال التالي:

```
>> x=[-10:.5:10];
>> y=x.^2;
>> c=complex(x,y);
>> plot(c)
```

في هذا المثال عرفنا متجهاً x يحوي القيم من 10- إلى 10 بخطوة مقدارها 0.5 والمتجه y هو متجه مربعات القيم في x ثم استخدمنا التابع complex لتكوين متجه من الأعداد العقدية يأخذ كل منها قسمه الحقيقي من المتجه x وقسمه التخيلي من y، الآن عند استخدام التابع plot(c) في السطر الرابع يقوم ماتلاب برسم الخط البياني للمتحول y كتابع للمتحول x وهذا يكافئ استخدام الأمر plot(x,y) أما عند استخدام الأمر plot(x,c) فهذا يعني رسم المتحول c كتابع للمتغير x وبما أن c هو متجه أعداد عقدية يهمل ماتلاب القسم التخيلي ويؤول الأمر إلى مكافئه plot(x,x)

◄ رسم خطين بيانيين بمحورين مستقلين

في الواقع تعني هذه العملية رسم خطين بيانيين على منطقتي محاور مختلفتين ولكنهما متراكبتان وتشغلان المساحة نفسها وتتوضعان في الموضع نفسه لكن إحداهما محورها الشاقولي في جهة اليمين والأخرى محورها الشاقولي في جهة اليسار، وعند إسناد الرسم في هذه الحالة نحصل على متحولي مقبض لمنطقتي الرسم ومتحولي مقبض للخطين البيانيين. يستخدم للرسم بهذه الطريقة التابع `plotyy` كمايلي:

[a h1 h2]=plotyy(x,y,x,z)

حيث يتم إسناد منطقتي الرسم إلى المتحول `a` الذي سيكون في هذه الحالة متجها عموديا من قيمتين ويتم إسناد الخطين البيانيين إلى المتحولين `h1` و `h2`. يمكن فيما يعد ضبط الخصائص ولكن لا يمكن فصل منطقتي الرسم.

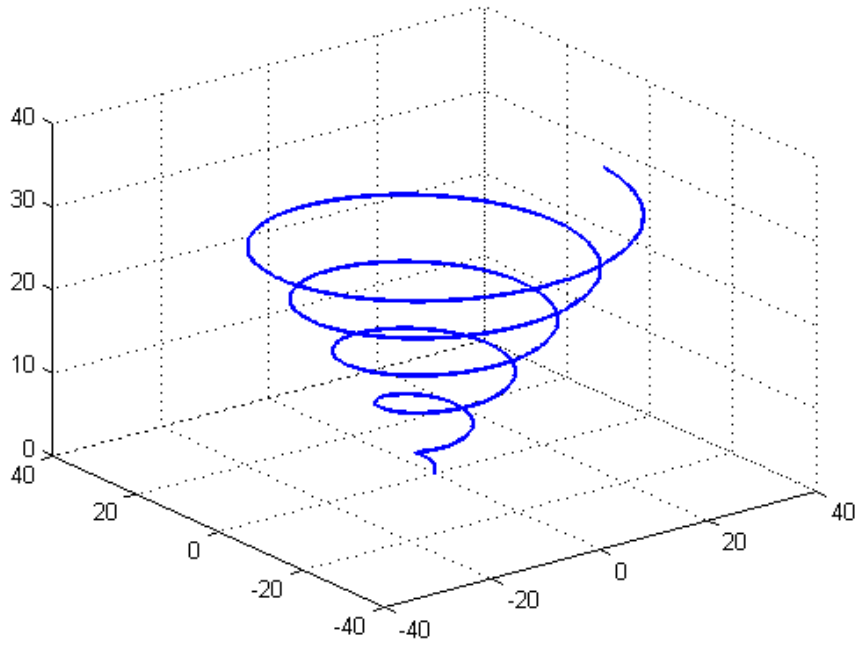
◄ رسم المنحنيات ثلاثية الأبعاد

يتم رسم المنحنيات ثلاثية الأبعاد في ماتلاب بطريقة مشابهة تماما لطريقة رسم المنحنيات ثنائية الأبعاد ويصح كل ما سبق شرحه حول نافذة الشكل ومنطقة المحاور والخطوط البيانية وضبط الخصائص، ونضيف أن التابع المستخدم للرسم في الفضاء

هو التابع plot3 الذي يحتاج إلى ثلاثة متغيرات واحداً لكل محور من المحاور الديكارتية، وعند تنفيذ الرسم يظهر الشكل ثلاثي الأبعاد من إحدى نقاط النظر كما في المثال التالي:

```
>> t=[0:0.01:10*pi];
>> x=cos(t).*t;y=sin(t).*t;
>> plot3(x,y,t,'linewidth',2);grid on;
```

عند التنفيذ سوف ينتج الرسم البياني المبين في الشكل 20

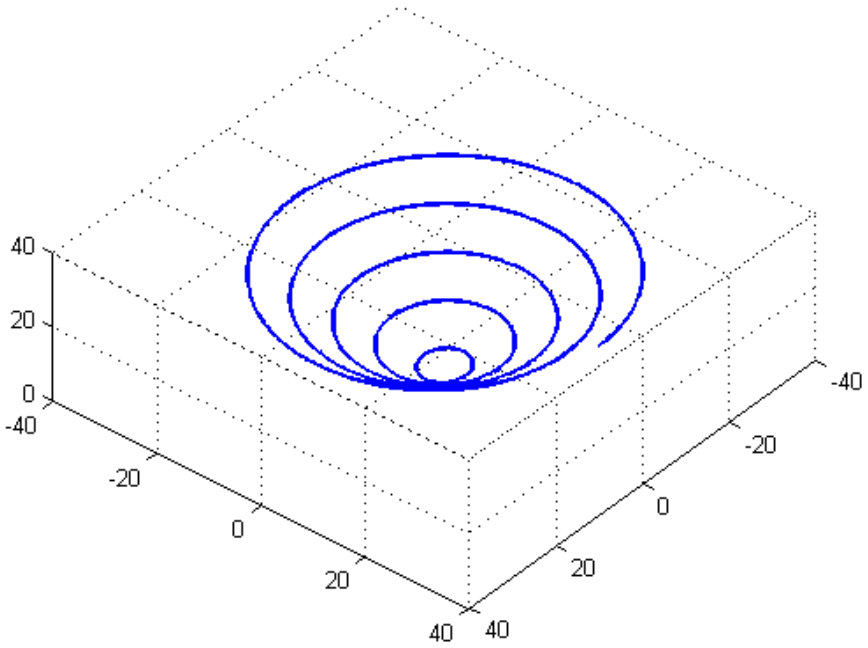


الشكل 20 الرسم في الفضاء

يمكن تغيير زاوية الرؤية بضبط خاصية نقطة توضع الكاميرا cameraposition كمايلي:

```
>> set(gca,'CameraPosition' , [-100 -120 -150])
```

والنتيجة هي تغير المشهد ليظهر وكأن عين الناظر تتمركز في النقطة التي أدخلت إحداثياتها في خاصة موضع الكاميرا كما في الشكل 21



الشكل 21 مشهد مختلف لخط ثلاثي الأبعاد

◄ التمثيل البياني للدوال التابعة لمتحولين أو أكثر

يمكن تمثيل الدالة z التابعة لمتحولين x و y بأكثر من طريقة وتتبع الطريقة المتبعة الهدف من ذلك التمثيل، وفيمايلي بعض تلك الطرق وكيفية تطبيقها في ماتلاب:

طريقة خطوط التسوية

في هذه الطريقة يتم تمثيل الدالة z في المستوي المحدد بالمحورين OX و OY بشكل خطوط كل خط منها يدل على القيم المتساوية لقيمة الدالة z ويمكن استخدام التلوين للدلالة على اختلاف الخطوط أو القيم العددية بجانب كل خط. تستخدم هذه الطريقة في الطبوغرفيا حيث يتم تمثيل التضاريس الطبيعية كالجبال والوديان بخطوط تسوية وتكون قيم الخطوط هي ارتفاعات تلك التضاريس فيما تدل القيم المتغيرة x و y على إحداثيات النقاط على الأرض.

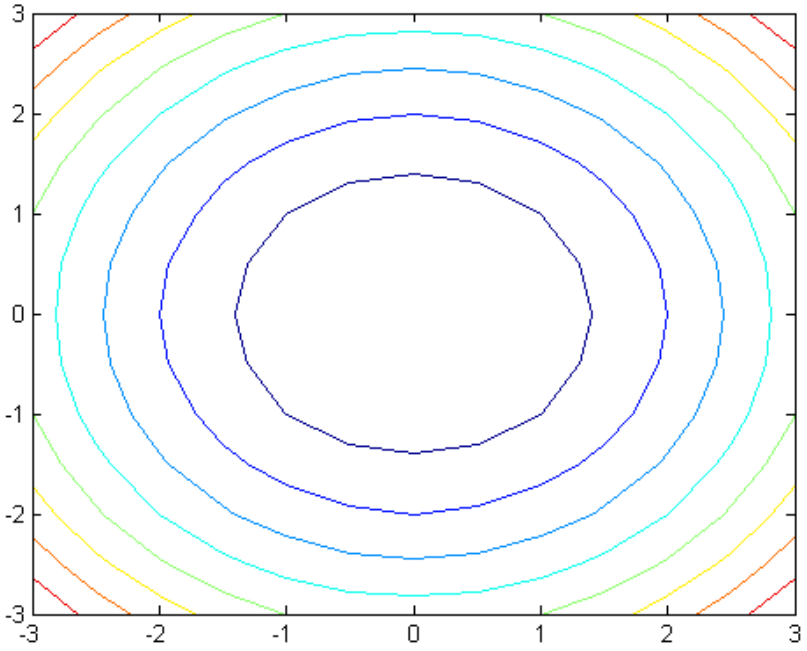
في ماتلاب يمكن استخدام هذه الطريقة بواسطة التابع contour بعد إنشاء شبكة الإحداثيات المكونة من قيم المتجهين x و y بواسطة التابع meshgrid كما في المثال التالي:

```
>>x=[-3:0.5:3];
```

```
>>[x y]=meshgrid(x);
>>z=x.^2+y.^2;
>>contour(x,y,z)
```

في هذا المثال قمنا أولا بتعريف مجال القيم للمتحول x على شكل متجه من القيم، في السطر الثاني قمنا بإنشاء شبكة الإحداثيات للمتغيرين x و y وهذا يعني أن المتحول x أصبح مصفوفة مربعة من القيم كل سطر منها هو المتجه x سابقا أما المتجه y فهو منقول المصفوفة x، هذه يعطينا شبكة من الإحداثيات وكأننا عرفنا إحداثيات كافة النقاط الواقعة ضمن المجال المستوي (ثنائي البعد) المحدد بقيم المتحولين، في السطر الثالث قمنا بتعريف العلاقة التابعية بين المتحول الدالة z والمتغيرين x و y ليتم حساب تلك القيم عند كل نقطة من المجال المستوي المعرف، ومن الملاحظ أن أبعاد كل المصفوفات الثلاث متساوية، أخيراً في السطر الرابع قمنا بإنشاء التمثيل البياني بواسطة التابع contour الذي ندخل له أسماء المتحولات الثلاثة.

الشكل الناتج سيكون كما في الشكل 22



الشكل 22 التمثيل البياني بطريقة خطوط التسوية

في هذا المثال تم الرسم في منطقة المحاور الحالية ولكن إذا أردنا الرسم في منطقة محاور أخرى علينا تحديدها بواسطة إدخال متحول مقبضها كبارامتر أول قبل أسماء المتحولات الثلاثة، وكذلك تم رسم عدد من الخطوط قام ماتلاب بتحديده حسب مجال القيم للمتحولين x و y أما إذا أردنا رسم عدد محدد من الخطوط فيجب إدخال هذا العدد كبارامتر رابع أو خامس بعد أسماء المتحولات الثلاثة، المثال التالي يوضح ترتيب إدخال هذه البارامترات:

```
>> figure
>> ax=axes
>> contour(ax,x,y,z,4)
```

في السطر الأول قمنا بإنشاء نافذة شكل ومنطقة محاور جديدتين وفي السطر الثالث قمنا برسم أربعة خطوط تسوية في منطقة المحاور الجديدة باستخدام متحول مقبضها ax

خطوط التسوية يمكن إسنادها أيضا إلى متحول مقبض لضبط خصائصها لاحقاً ولكن هنا يمكن استخدام متجه أفقي من قيمتين لمتحول المقبض كمايلي:

```
>> [c h]=contour(ax,x,y,z,4);
```

حيث يدل المتحول c على مصفوفة الخطوط وهي مصفوفة من القيم ذات سطرين يقوم ماتلاب بإنشائها لتعريف الطريقة التي سيقوم برسم الخطوط تلك بواسطتها، وبعبارة أخرى هي إحداثيات لتلك الخطوط في المستوي XOY ففي السطر الأول تكون قيم x وفي السطر الثاني قيم y وهي لا تعنينا هنا من حيث القيم لكن ماتلاب يستخدمها ليرسم الخطوط ولضبط بعض الخصائص كما سنرى، أما المتحول h فهو متحول المقبض لخطوط التسوية المرسومة وبواسطته يمكن ضبط الخصائص الأساسية لها.

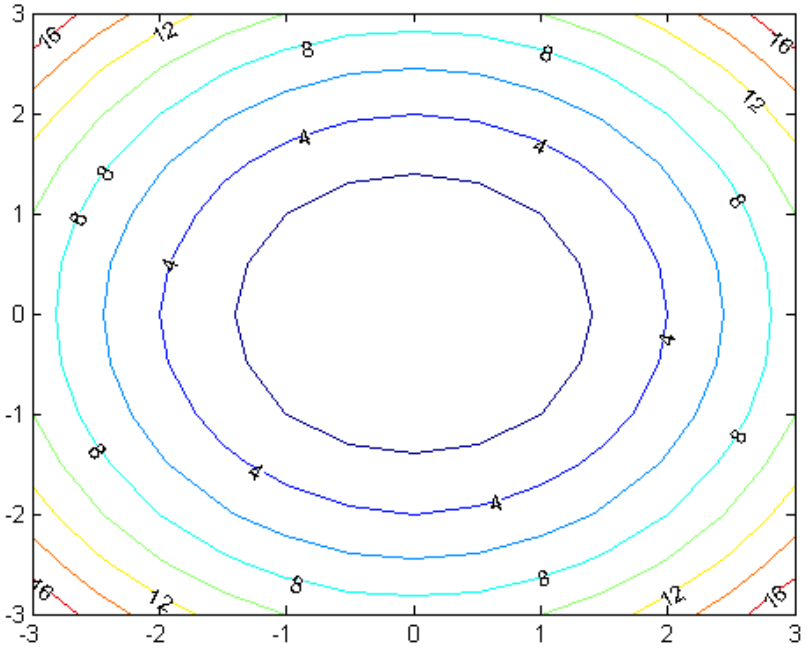
تستخدم الخاصية showtext لإظهار قيم عددية على الخطوط تدل على قيم z كمايلي:

```
>> set(h,'showtext','on');
```

ويمكن جعل تلك القيم تظهر على بعض الخطوط دون أخرى كأن تظهر على واحد من كل خطين متتاليين باستخدام الخاصية textstep والخاصية levelstep كمايلي:

```
>> set(h,'textstep',get(h,'levelstep')*2);
```

الشكل 23 يظهر كيفية ظهور تلك القيم العددية على خطوط التسوية.

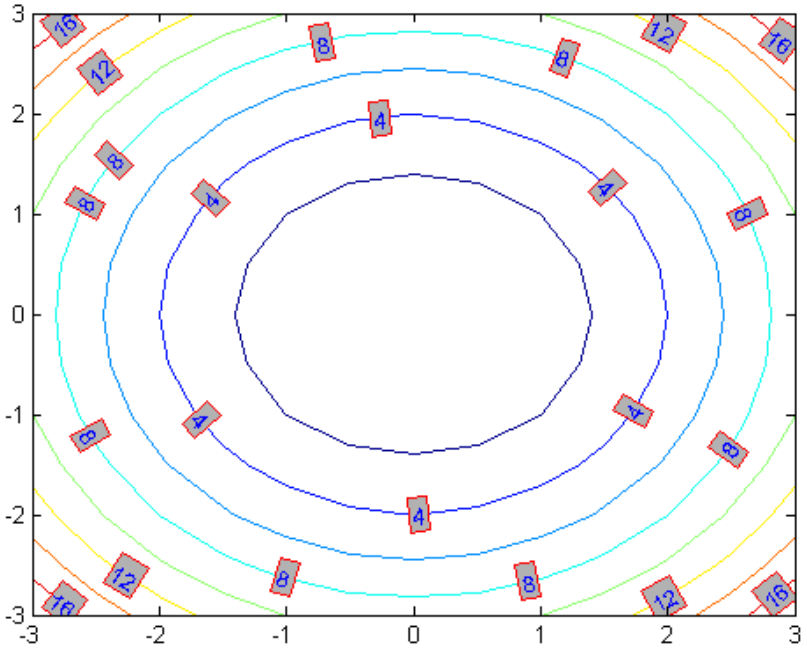


الشكل 23 الدلالة العددية على خطوط التسوية

هناك أيضاً إمكانية لتلوين خلفية هذه القيم بإسناد النص إلى متحول مقبض ثم إعطائه الخصائص المطلوبة كلون الخلفية ولون الخط المحيط ولون خط النص وغير ذلك، كما في المثال التالي:

```
>> text_h=clabel(c,h);
>> set(text_h,'backgroundcolor',[0.7 0.7 0.7],...
'edgecolor',[1 0 0],'color',[0 0 1])
```

هذه النصوص ستظهر كما في الشكل 24



الشكل 24 خطوط تسوية بخصائص متعددة

يمكن أيضاً الاطلاع على بقية خصائص خطوط التسوية وبقية خصائص نصوص قيم العنونة باستخدام الأمرين:

```
>>get(h)
>>get(text_h)
```

طريقة الرسم ثلاثي الأبعاد

في هذه الطريقة يتم رسم النقاط ثلاثية الأبعاد (x,y,z) باستخدام التابع mesh كمايلي:

```
>>x=[-3:0.5:3];
>>[x y]=meshgrid(x);
>>z=x.^2+y.^2;
>>mesh(x,y,z)
```

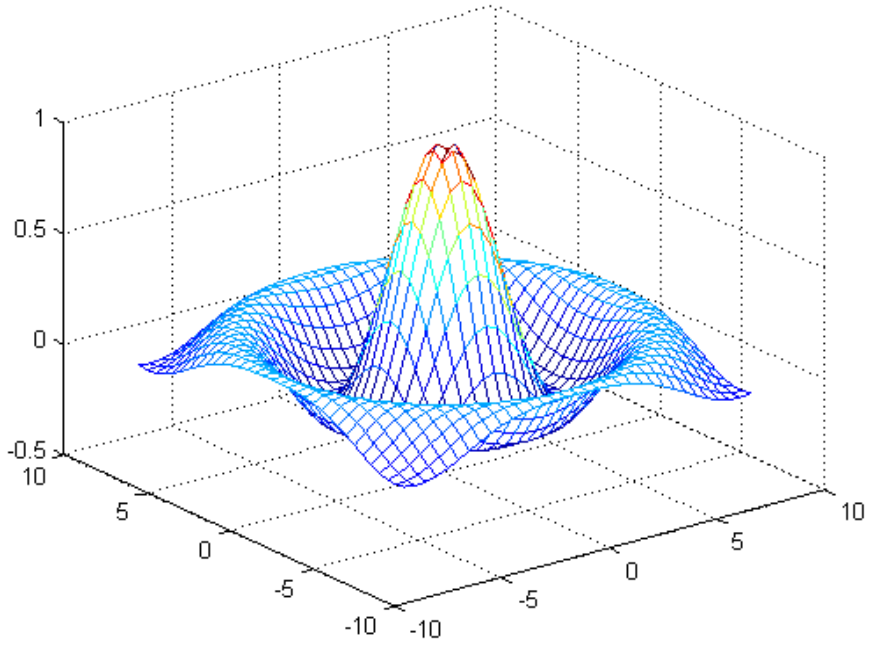
الأسطر الثلاثة الأولى في هذا المثال تم شرحها سابقاً في الطريقة السابقة، أما السطر الرابع ففيه نقوم برسم شبكة ثلاثية الأبعاد كل نقطة منها محددة بالإحداثيات (x,y,z) وذلك لتمثيل الدالة z=f(x,y) فراغياً.

يصح هنا أيضاً الكثير مما قيل سابقا في عدة مواضع، فيمكن رسم هذه الشبكة في منطقة محاور غير الحالية باستخدام متحول مقبض منطقة المحاور المطلوبة كبارامتر أول في التابع mesh، ويمكن إسناد الرسم إلى متحول مقبض لضبط الخصائص والوصول إليه لاحقا، كذلك يمكن ضبط بعض الخصائص مباشرة. المثال التالي يوضح بعض تلك الأمور:

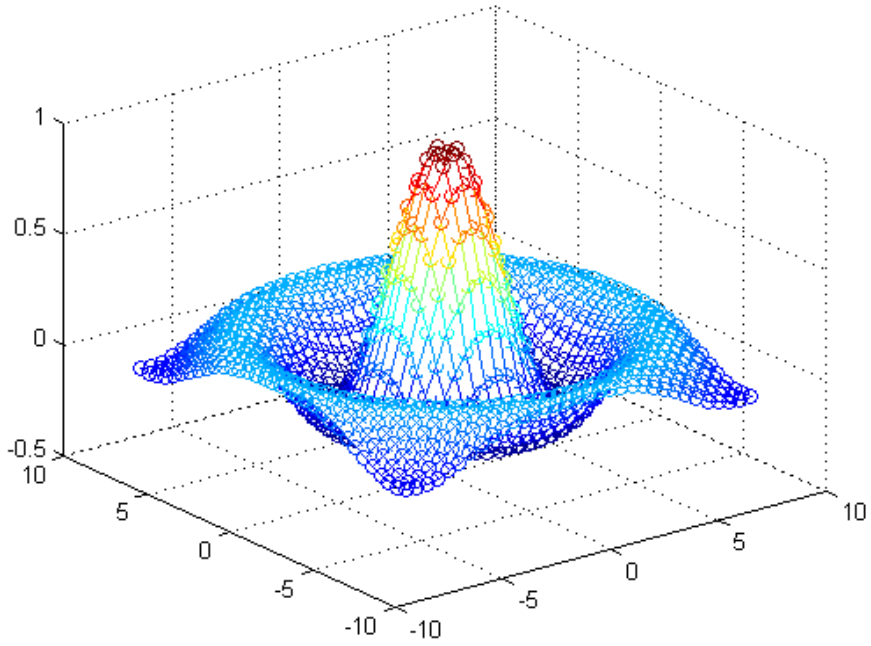
```
>> x=[-8:0.5:8];
>> [x y]=meshgrid(x);
```

```
>> z=sin(sqrt(x.^2+y.^2))./(sqrt(x.^2+y.^2));
>>ax=gca;
>> h=mesh(ax,x,y,z)
>> set(h,'marker','o')
```

في السطر الرابع أسندنا متحول مقبض منطقة المحاور الحالية إلى المتحول ax واستخدمنا الأخير في السطر الخامس لرسم شبكة التمثيل البياني الفراغي على منطقة المحاور تلك، بالطبع هذه العملية غير ضرورية لأن ماتلاب سيقوم في الحالة الافتراضية بالرسم هناك ولكن فعلنا ذلك لتوضيح إمكانية توجيه الرسم، في السطر الأخير يتم استخدام متحول مقبض الشبكة المرسومة لتغيير خاصية علامة النقطة إلى دائرة. الشكل 25 و الشكل 26 يظهران النتيجة قبل وبعد ضبط الخاصة الأخيرة.



الشكل 25 سبكة بالإعدادات الافتراضية

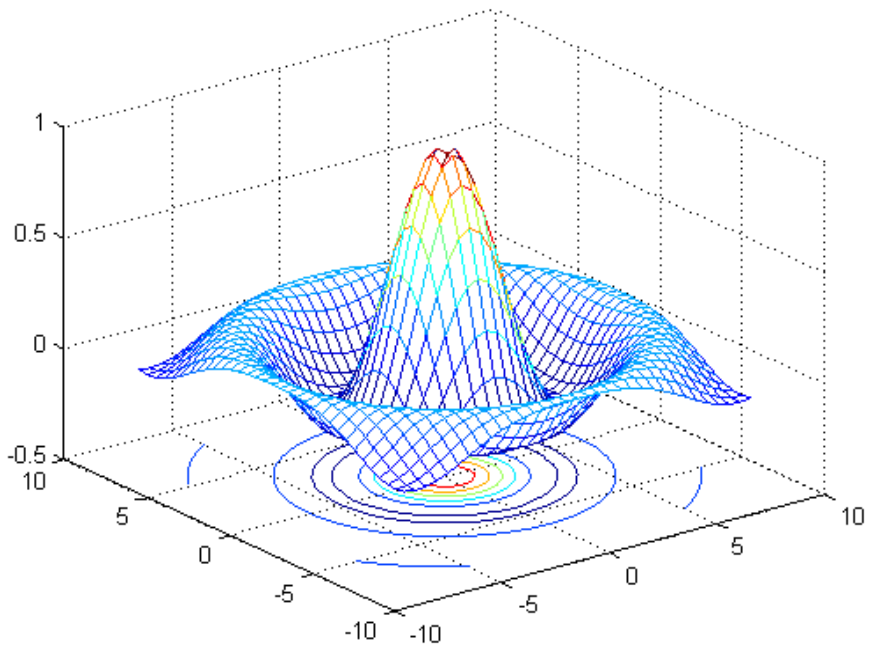


الشكل 26 شبكة تستخدم الدائرة كعلامة

يمكن ضبط بقية خصائص الشبكة بنفس الطريقة، وللاطلاع على الخصائص يمكن كتابة الأمر:

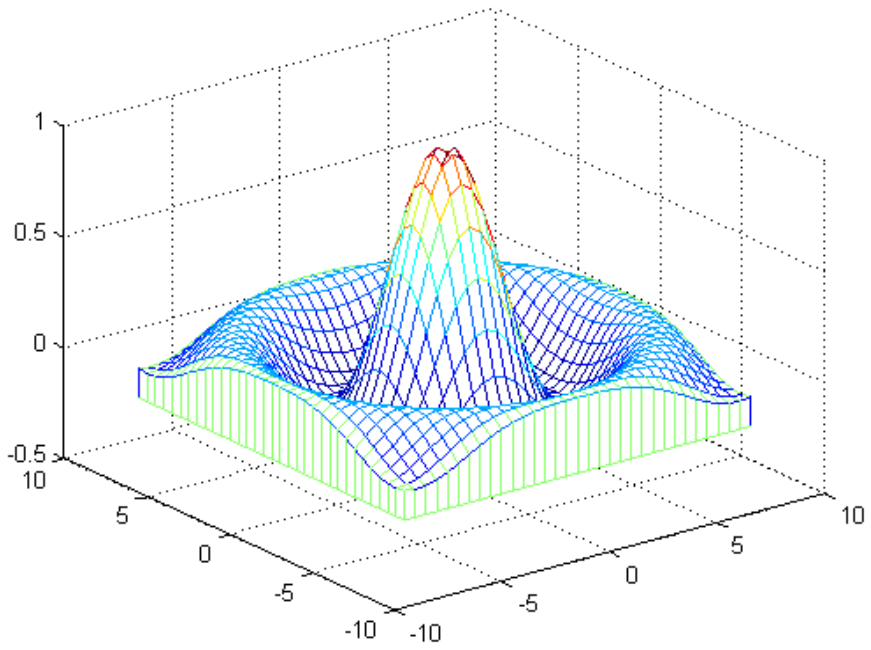
>>get(h)

يمكن في ماتلاب أيضاً رسم شبكة التمثيل البياني الفراغي وخطوط التسوية معاً على رسم واحد باستخدام التابع `meshc` بنفس أسلوب التابع `mesh` فيظهر الرسم كما في الشكل 27



الشكل 27 خطوط التسوية والرسم الفراغي معاً

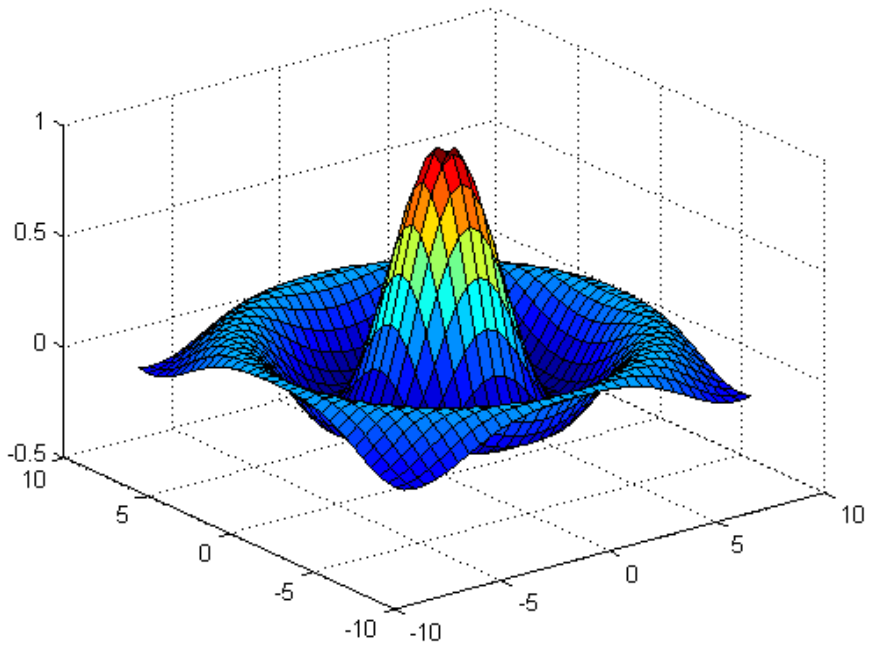
ويمكن رسم مغلف من الخطوط الموازية للمحور oz حول الشبكة الفراغية يبدأ كل خط من المستوي xoy وينتهي عند النقطة الموافقة من الشبكة فتظهر النتيجة كما في الشكل 28



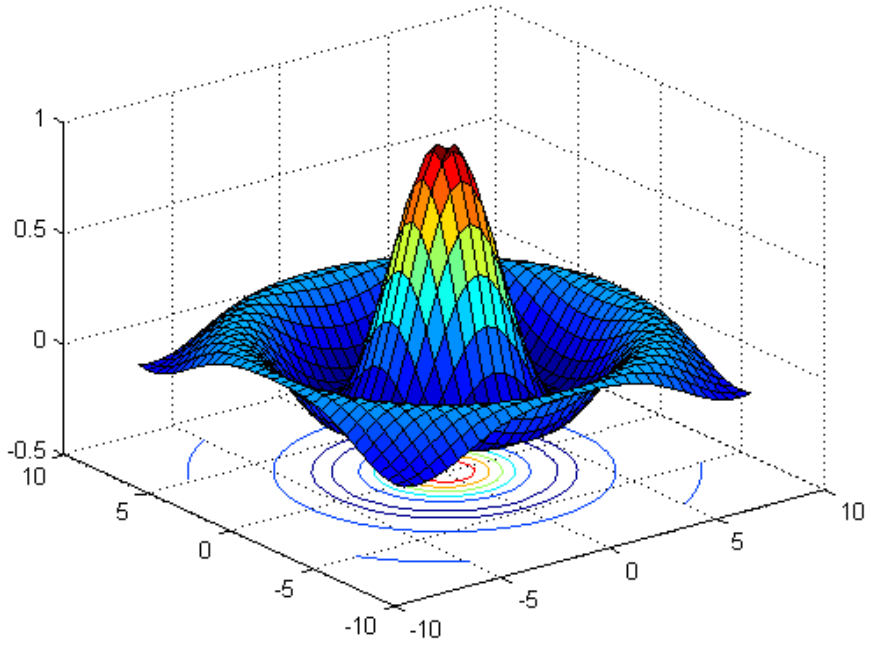
الشكل 28 الرسم الفراغي مع مغلف

هناك تابع آخر لتمثيل الدالة لمتحولين فراغياً وهو التابع surf الذي يسلك سلوكاً مشابهاً للتابع mesh إلا أنه يلون المساحات الفارغة في الشبكة لتظهر وكأنها ممتلئة،

أي لم تعد شبكة بل أصبحت سطحاً. ويمكن استخدام التابع surfc أيضاً لرسم السطح مع خطوط التسوية. تظهر النتائج كما في الشكل 29 و الشكل 30



الشكل 29 رسم الشبكة الفراغية الممتلئة



الشكل 30 الشبكة الفراغية الممتلئة مع خطوط التسوية

طريقة الرسم ثنائي البعد عدة مرات

تستخدم هذه الطريقة عندما يكون لدينا دالة معرفة تتبع أكثر من متحولين وهذه الحالات كثيرة في الهندسة والعلوم، في هذه الطريقة نقوم بإعطاء جميع المتحولات قيماً ثابتةً عدا المتحول الأول ثم نقوم برسم الدلة وكأنها تتبع المتحول الأول فقط، ثم نغير قيمة المتحول الثاني ونعيد الرسم وهكذا للحصول على مجموعة من الخطوط البيانية كل منها يمثل الدالة عند مجموعة قيم للمتحولات الأخرى غير المتحول

الأول، بالتأكيد يمكن العمل مع المتحول الثاني بدلاً من الأول وهذا يعود للغاية من التمثيل البياني.

هناك سبب آخر لاستخدام هذه الطريقة حتى لو كانت الدالة تتبع متغيرين فقط، وهو توضيح أثر تغير أحد المتحولات على الخط البياني للدالة، وعلى سبيل المثال لتوضيح أثر إضافة عدد في نهاية دالة صحيحة من الدرجة الثانية يمكن إعطاء قيمة متغيرة في كل مرة لذلك العدد ورسم الدالة الناتجة للحصول على مجموعة من الخطوط التي تمثل قطوعاً مكافئة بإزاحات شاقولية حسب قيم العدد المضاف، في الحقيقة ما تم رسمه سيكون دالة تتبع المتحول الأصلي للدالة الصحيحة من الدرجة الثانية والمتحول الجديد وهوالعدد المضاف لعمل الإزاحة.

في المثال التالي سوف نقوم بتمثيل الدالة المعرفة وفق العلاقة التابعية التالية:

$$y = \sin(x + u)$$

لتمثيل هذه الدالة بيانياً نعرف مجال القيم للمتحول x ثم مجال من القيم للمتحول u، بعد ذلك نقوم برسم الدالة:

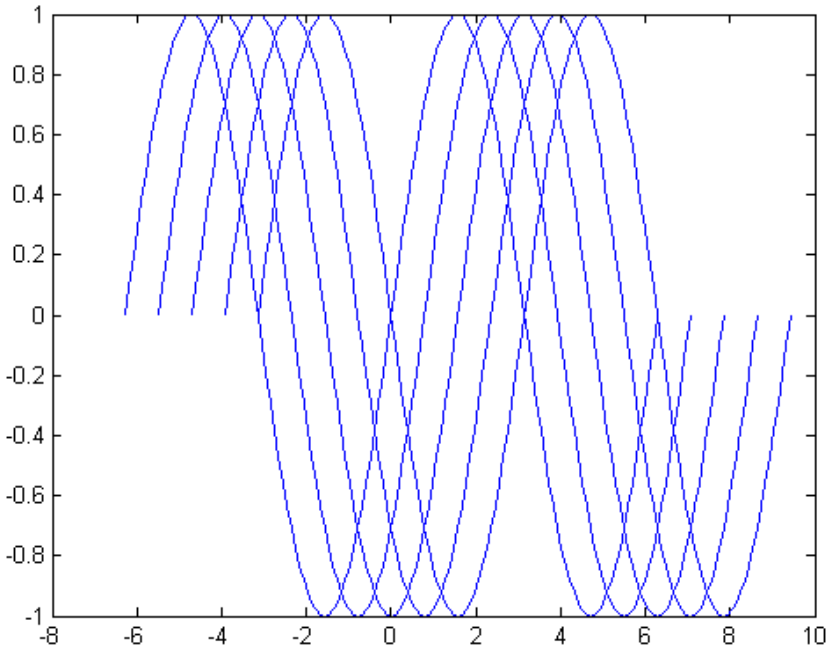
$$y = \sin(x)$$

عند كل قيمة للمتحول u باستخدام الكود التالي:

```
>> x=[-2*pi:pi/32:2*pi];u=[0:pi/4:pi];y=sin(x);
>> plot((x+u(1)),y);hold on;
```

```
>> plot((x+u(2)),y);hold on;
>> plot((x+u(3)),y);hold on;
>> plot((x+u(4)),y);hold on;
>> plot((x+u(5)),y);hold on;
```

وبالنتيجة نحصل على التمثيل البياني الموضح في الشكل 31



الشكل 31 التمثيل البياني ثنائي البعد لدالة تتبع متحولين

يمكن هنا إضافة المسميات على الخطوط لبيان قيمة المتحول u المستخدم في كل خط.

من الجدير بالذكر أن هذا العمل يمكن إجراؤه بطريقة أبسط بواسطة الحلقات التكرارية التي سوف نناقشها في بحث الأدوات البرمجية في ماتلاب، ولا سيما عندما تكون مجموعة القيم المستخدمة للمتحول u كثيرة وعندما يكون لدينا أكثر من متحول آخر غير المتحول x

◄ رسم المساحات

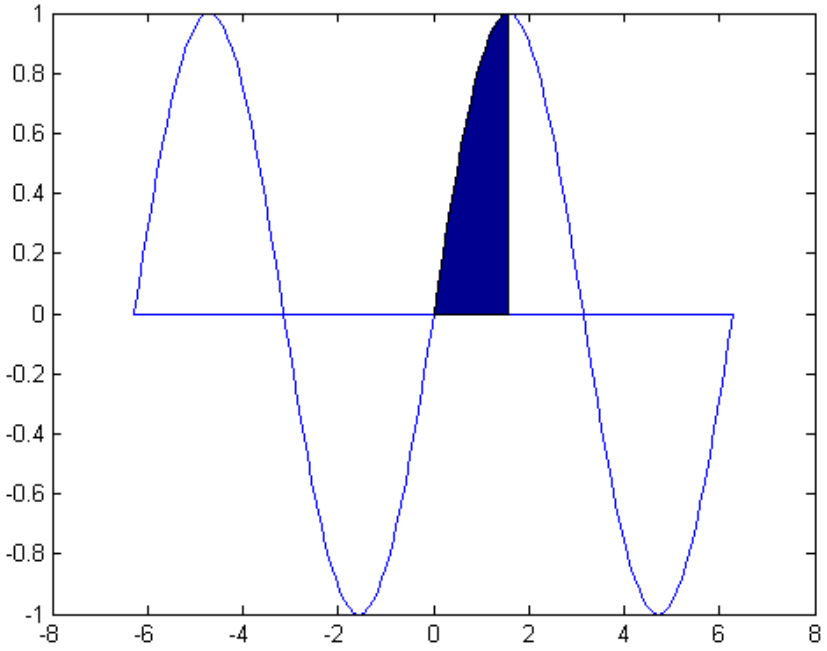
يستخدم التابع area لرسم المساحات المحصورة بين الخطوط البيانية ومحور الفواصل OX ضمن مجال محدد لقيم x بشكل مشابه لاستخدام التابع plot، ويمكن بالتأكيد استخدام التابعين معاً لرسم خط بياني ما وإظهار مساحة محددة محصورة بينه وبين المحور OX كما في المثال التالي:

```
>> x=[-2*pi:pi/32:2*pi];
>> y=sin(x);
>> plot(x,y)
>> xa=[0:pi/32:pi/2];
>> ya=sin(xa);
>>hold on
>> area(xa,ya);
>>hold on
```

>> plot(x,zeros(size(x)))

في السطر الأول تم تعريف مجال قيم المتحول x، وفي السطر الثاني تم تعريف العلاقة التابعية بين المتحولين x و y، وفي السطر الثالث تم رسم الدالة y=f(x) بواسطة التابع plot. في السطر الثالث تم تعريف مجال قيم المتحول xa ضمن مجال قيم المتحول x ولكن بنفس الخطوة للحصول على دقة جيدة في الرسم، ثم تم في السطر الخامس تعريف العلاقة التابعية بين المتحولين xa و ya بنفس علاقة المتحولين x و y لأننا نريد رسم المساحة المحصورة تحت نفس التابع، في السطر السادس تم استخدام الأمر hold on للاحتفاظ بالرسم الأول وفي السطر السابع تم استخدام الأمر area لرسم المساحة المطلوبة، في السطر الأخير تم استخدام الأمر plot لرسم المحور OX.

الشكل الناتج من هذه العملية سيكون كما في الشكل 32

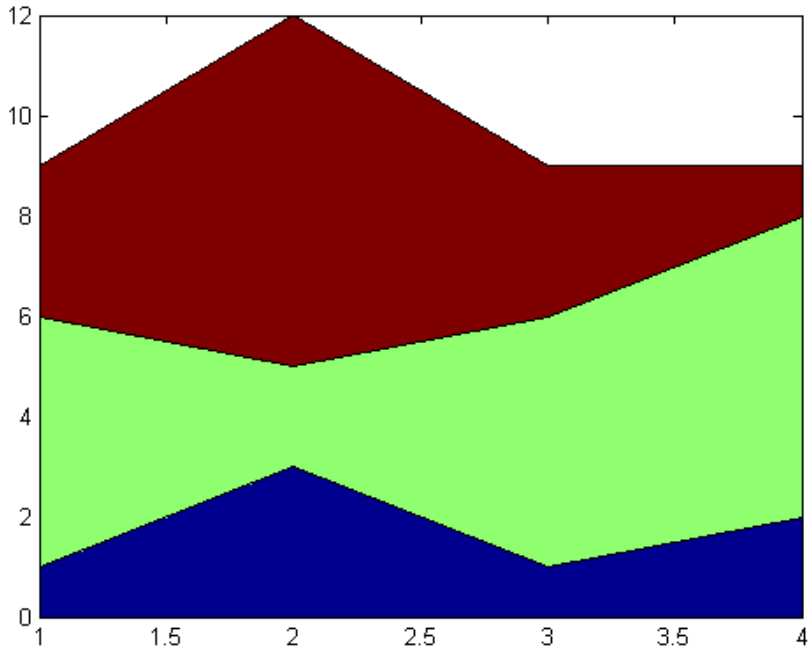


الشكل 32 رسم المساحات

يستخدم التابع area أيضاً بطريقة أخرى لرسم عدة مساحات متراكبة كل منها فوق الأخرى كما في المثال التالي:

```
>> y=[1 5 3;3 2 7;1 5 3; 2 6 1];
>> area(y)
```

في السطر الأول تم تعريف المصفوفة y التي تحوي أربعة أعمدة، في السطر الثاني استخدم التابع area لرسم المساحات المتراكبة الثلاث حيث يعامل كل عمود من المصفوفة y على أنه قيم دالة جديدة يراد رسم مساحتها وترسم مقابل قيم أدلتها بسبب غياب المتحول x والجديد المختلف هنا هو أن القيم في كل عمود تضاف إلى القيم السابقة عند الرسم وبالنتيجة نحصل على الشكل 33



الشكل 33 رسم المساحات المتراكبة

يمكن الحصول على النتيجة ذاتها بواسطة التعليمات المكافئة التالية:

```
>> y1=[1 3 1 2];
>> y2=[5 2 5 6];
>> y3=[3 7 3 1];
>> a3=area(y3+y2+y1);
>> hold on
>> a2=area(y2+y1);
>> hold on
>> a1=area(y1);
>> set(a1,'facecolor','r');
>> set(a2,'facecolor','g');
```

مع الانتباه إلى ضرورة رسم المساحات بالترتيب لأن ماتلاب يظهر تلك المساحات على شكل طبقات فالمساحة التي ترسم أولاً تختفي في الأسفل وهي تظهر في مثالنا لأنها تشغل حيزاً أقل من المساحات الواقعة فوقها، في حال لم يتم الرسم بالترتيب يمكن إعادة ترتيب طبقات المساحات باستخدام الخاصة `children` للعنصر الأب لإحدى المساحات وذلك بالطريقة التالية:

```
>> set(get(a1,'parent'),'children',[a1;a2;a3])
```

حيث استخدمنا التابع `get` للحصول على متحول المقبض للعنصر الأب للمساحة `a1` ثم مررنا هذا المتحول كبارامتر للتابع `set` لضبط الخاصة `children` للعنصر

الأب وكل ما ينبغي فعله هو إعادة ترتيب العناصر في متجه العناصر الأبناء وهنا لدينا ثلاثة أبناء هي a1 و a2 و a3

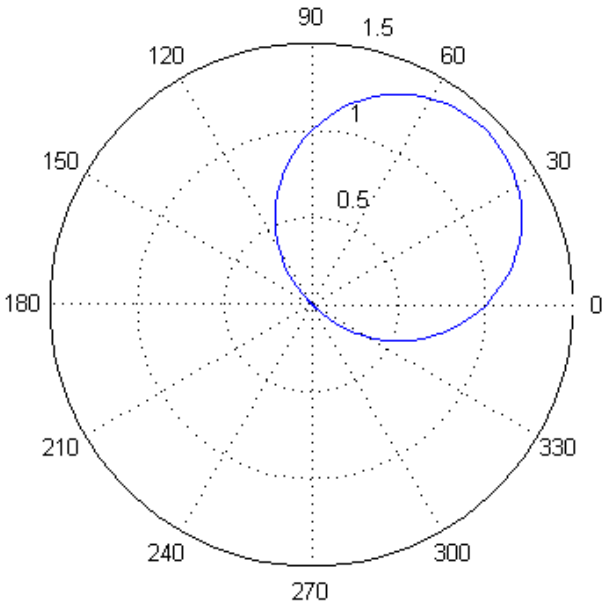
◄ رسم المنحنيات القطبية

يستخدم للرسم في المستوي القطبي تابعان أساسيان هما التابع polar والتابع compass فالأول يرسم لنا المنحني القطبي كخط بياني، أما الثاني فيقوم بريم أشعة تعبر عن القيم التابعة لقيم الزاوية القطبية.

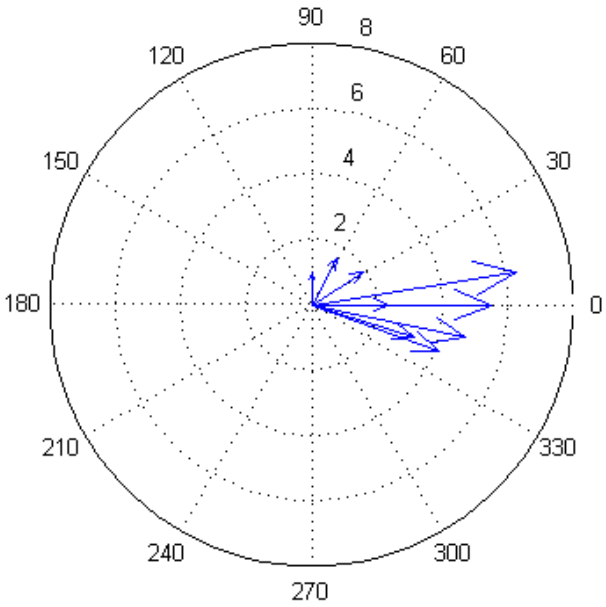
المثال التالي يقوم برسم الدالة r=sin(th)+cos(th) بالطريقتين:

```
>> th=[0:pi/64:2*pi];
>> r=sin(th)+cos(th);
>> polar(th,r)
>> th=[0:pi/4:2*pi];
>> r=sin(th)+cos(th);
>> figure,compass(th,r)
```

تظهر نتيجة تنفيذ هذين الامرين كما في الشكل 34 والشكل 35



الشكل 34 الرسم في المستوي القطبي



الشكل 35 التعبير عن الدالة القطبية بالمتجهات القطبية

الأدوات البرمجية

◄ مقدمة

سبق القول أن نظام ماتلاب هو بيئة عمل متكاملة ويوفر إمكانيات برمجية، وفي الواقع لا يمكن اعتباره لغة برمجة لأنه يحوي أدوات أخرى غير برمجية ولا يمكن اعتباره أقل من لغة برمجة لأنه يحوي كافة المواد اللاومة لبناء وتشغيل البرامج.

عند الحديث عن البرمجة هناك عدة تساؤلات ينبغي مناقشتها وفي مطلعها حزمة التطوير ونوع اللغة ومحرر الشيفرة وطريقة الترجمة والتنفيذ وإمكانية بناء التطبيقات ومنصات العمل المتاحة وأدوات التحكم بسير البرنامج والبرمجة غرضية التوجه والبرمجة الموجهة بالحدث وغير ذلك. في الفقرات التالية سوف نناقش بعض هذه المواضيع فيما يخص ماتلاب بفرض أن القارئ ملم نوعاً ما ببعض المفاهيم، أي أننا لن ننطلق من الصفر فالغاية من هذا الكتاب هي تقديم ماتلاب والغاية من هذا الفصل هي تقديم ماتلاب كلغة برمجة وليس تعليم البرمجة.

◄ ماتلاب كلغة برمجة

يملك ماتلاب حزمة تطوير خاصة به تمكن المبرمجين من كتابة الشيفرات وتنفيذها ضمن بيئة ماتلاب، ففي حزمته تلك جميع الأدوات البرمجية اللازمة لتعريف المتحولات وحجز مساحات التخزين في الذاكرة وتنظيمها وإجراء العمليات على المتحولات ويمكنه التخاطب مع التطبيقات الخارجية ونظام التشغيل، كذلك ففي

حزمته تلك الأدوات اللازمة لترجمة الأوامر ترجمة فورية دوت التحويل إلى ملفات بلغة الآلة مما يجعل العمل معه أبسط.

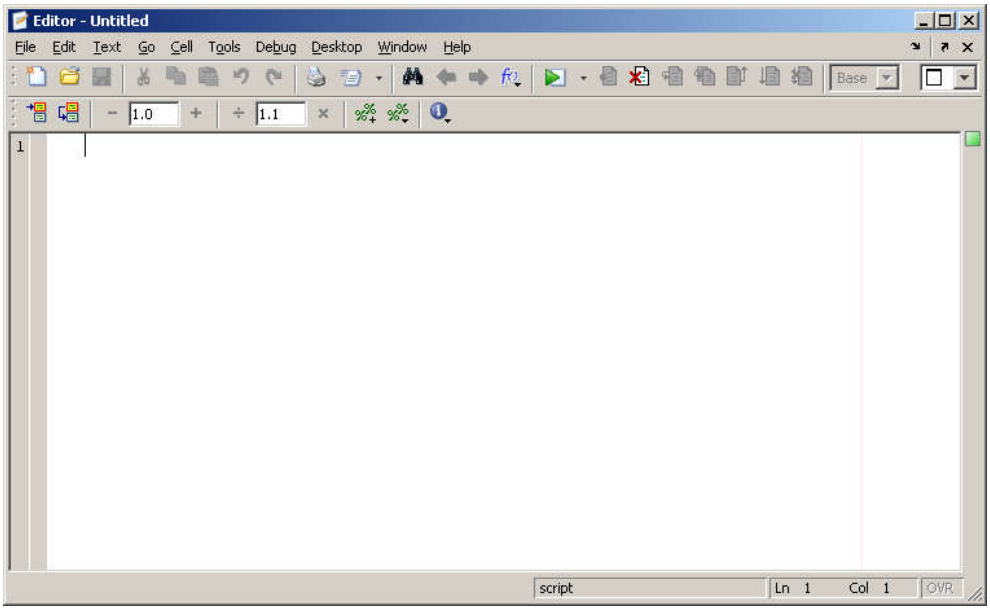
لتشغيل البرامج المكتوبة بلغة ماتلاب ينبغي وجود نظام ماتلاب على الجهاز وهذا يعني أن البرنامج المكتوب بلغة ماتلاب لا يعمل بشكل مستقل وإنما يعمل في بيئة ماتلاب نفسها، هذا الأمر لا ينفي وجود إمكانية لتطوير تطبيق مستقل يعمل دون وجود ماتلاب أو دون تشغيل ماتلاب وهذه الإمكانية متوفرة بواسطة ترجمة التطبيق بمترجم لغة c أو مترجم لغة c++ حيث لا يملك ماتلاب مترجماً خاصاً به لإنشاء التطبيقات المستقلة.

ما يميز البرمجة في ماتلاب هو وجود مكتبات هائلة من التوابع الرياضية الجاهزة وهو ما لا نجده في لغات البرمجة عادة ونضطر عند الحاجة إلى إجراء عملية رياضية ما إلى كتابة تابع لها مما يعني أننا نعيد اختراع العجلة لهذه العملية، وهو عكس الشعار الذي يرفعه دائما العاملون في حقل البرمجة وهو "لا تعد اختراع العجلة"، فينبغي على المبرمج دائما أن يبحث عن الأدوات المتوفرة لاستخدامها في عمله التطويري دون أن يصرف جهده ووقته على تطوير الأدوات التي تم تطويرها من قبل. للاطلاع على المكتبات التي يوفرها ماتلاب يمكن مراجعة وثائق المساعدة حيث لا يتسع الكتاب لذكر تلك المكتبات، وفي تلك الوثائق يتم تصنيف المكتبات حسب وظائفها إلى مجموعات فهناك مكتبة التوابع المتعلقة بالعمليات الجبرية ومكتبة التحليل الرياضي ومكتبة الحساب العددي ومكتبة الإحصاء وغيرها الكثير.

◄ أين تكتب أوامر ماتلاب؟

عند الحاجة إلى كتابة الشيفرات البرمجية لا بد من محرر نصوص بسيط أو متخصص للقيام بذلك وعادة ما يبدأ المبرمجون المبتدئون تعليمهم باستخدام محررات نصوص بسيطة معدة للأغراض العامة كبرنامج المفكرة notepad أو المفكرة notepad++ ثم ينتقل المبرمج إلى استخدام محرر شيفرة متخصص يوفر إمكانيات أكثر من مجرد الكتابة والتحرير، مثل تعقب الأخطاء والترجمة في اللغات التي تحتاج إلى الترجمة وتنظيم الملفات وغير ذلك.

يمكن كتابة شيفرات البرمجة في ماتلاب أيضاً باستخدام محرر نصوص بسيط مثل المفكرة أو غيره حيث نقوم لاحقاً بحفظ الملف بالتنسيق m. ليتم استدعاؤه في نافذة أوامر ماتلاب باسمه الكامل مع المسار أو اسمه فقط إن كان محفوظا في مسار ماتلاب (ناقشنا هذه الفقرة سابقاً في بداية الكتاب). رغم ذلك فإن ماتلاب لديه محرر نصوص خاص به لتحرير الملفات التي تكتب فيها التعليمات البرمجية ويمكن استخدامه ببساطة باختيار الأمر new من قائمة file ثم اختيار الأمر الفرعي m-file فتظهر النافذة المبينة في الشكل 36



الشكل 36 نافذة محرر الكود

وهي نافذة محرر الشيفرة البرمجية في ماتلاب. سوف نناقش إمكانيات هذا المحرر في سياق الحديث عن كتابة الشيفرات البرمجية كلا في حينه.

تجدر الإشارة أخيراً إلى أننا منذ بدأنا في الكتاب كنا نكتب أوامر برمجية وذلك في نافذة الأوامر حيث إن كل ما يمكن اعتباره أمراً برمجيا يمكن كتابته في نافذة الأوامر.

◄ المتحولات والثوابت وأنواع المعطيات

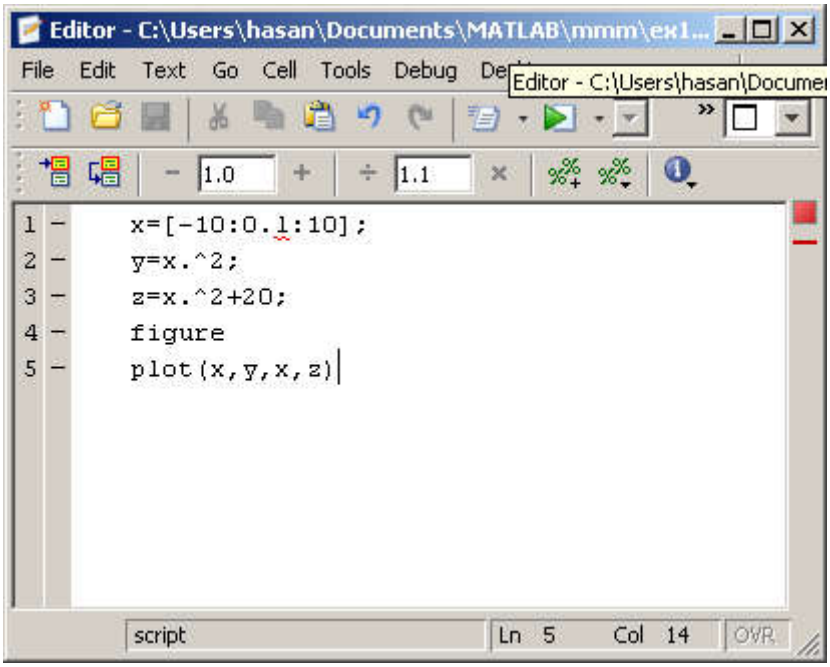
كما في أي لغة برمجة أخرى فإن ماتلاب لديه طريقة في التعامل مع المتحولات والثوابت وأنواع المعطيات، وبما أن هذا الأمر تمت مناقشته بالتفصيل سابقاً فلا داعي لإعادة الحديث عنه هنا.

◄ الإجراءات المعرفة من قبل المستخدم

في كل لغة برمجة هناك ما يسمى بالإجراء البرمجي وهو مجموعة من الأوامر والتعليمات يتم تنفيذها تباعاً. يتم تجزيء أي تطبيق يراد بناؤه إلى مجموعة من الإجراءات الجزئية لتسهيل عملية تنظيم التطبيق وصيانته وتطويره ولتحقيق إمكانية إعادة الاستخدام. في ماتلاب تكتب الإجراءات البرمجية الجزئية كملفات من النوع m-files ويتم حفظها في مجلد ضمن مسار ماتلاب وعند الحاجة إليها يتم استدعاؤها باسم الملف المحفوظ. خلافاً لبقية لغات البرمجة فإن الإجراءات في ماتلاب لا تحتاج إلى التعريف عنها فكل إجراء يحفظ في ملف مستقل.

مثال:

يمكن كتابة الشيفرة المبينة في الشكل 37 ثم حفظ الملف بالاسم ex1.m في مجلد معروف ضمن مسار ماتلاب، وعند الحاجة للتنفيذ يتم الاستدعاء من خلال نافذة الأوامر بكتابة الأمر ex1 وهو اسم الملف المحفوظ.



الشكل 37 إجراء برمجي بسيط (سكريبت)

يمكن أيضاً استدعاء أي ملف m-file من ملف m-file آخر.

◄ التوابع المعرفة من قبل المستخدم

يملك ماتلاب كما ذكرنا سابقاً العديد من المكتبات، كل منها تضم العديد من التوابع الجاهزة للاستخدام، مما يوفر على المطور الكثير من الوقت والجهد. في بعض الحالات لا يجد المطور تابعاً جاهزاً يؤدي الغرض المطلوب ولذلك يقوم بإنشاء ذلك

التابع وضمه إلى مكتبة من مكتبات ماتلاب أو إنشاء مكتبة جديدة (المكتبة في ماتلاب هي مجلد يقع في مسار ماتلاب وكل ما يلزم لإنشاء المكتبة هو إنشاء مجلد ووضعه في مسار ماتلاب، ثم كتابة ملفات التوابع الجديدة فيه).

يجب أن يبدأ أي ملف يحوي تابعاً معرفاً من قبل المستخدم بسطر التعريف عن التابع والشكل العام لهذا التعريف هو كمايلي:

Function [out1, out2, …]=functionnmae(in1, in2, …)

حيث تعبر الكلمة المفتاحية `function` عن أن الشيفرة التالية هي تابع معرف من قبل المستخدم. لكل تابع هناك مدخلات ومخرجات، يتم تعريف المخرجات هنا بالمتحولات `out1` و `out2` و ... ضمن الحاصرتين الكبيرتين ويتم تعريف المدخلات بالمتحولات `in1` و `in2` و ... ضمن الأقواس الصغيرة التالية لاسم التابع، ومن البديهي أن هذه الأسماء للتوضيح ويمكن استخدام أي أسماء متحولات اخرى.

يجب أن يحفظ التابع في ملف له ذات اسم التابع ليتم استدعاؤه بواسطته.

الأقواس الصغيرة إجبارية حتى لو لم يكن هناك مدخلات وهناك توابع كثيرة ليس لها مدخلات رغم أن أغلب التوابع تحتاج إلى المدخلات.

في حال كون المخرجات متحولاً وحيداً يمكن حذف الحاصرتين الكبيرتين ليصبح شكل سطر التعريف كمايلي:

Function out=functionnmae(in1, in2, …)

لكي يعمل التابع ويعيد نتائج صحيحة يجب إسناد قيم المخرجات بعد حسابها.

أما في حال كون التابع ليس له مخرجات فيصبح من النوع `void` وهو يكافئ إجراءاً برمجياً ويكتب سطر التعريف كمايلي:

Function functionnmae(in1, in2, …)

بعد سطر التعريف مباشرة يضاف وصف وشرح للتابع بشكل تعليقات (يبدأ سطر التعليق بالرمز `%`)، هذه التعليقات المضافة قبل بقية الشيفرة تظهر للمستخدم عند طلب المساعدة حول التابع بكتابة الأمر:

Help functionname

أما بقية التعليقات التي تكتب داخل أجزاء الشيفرة فلا تظهر في الشرح والوصف.

بعد ذلك تكتب شيفرة التابع لإجراء المطلوب.

مثال:

في هذا المثال سوف نكتب تابعاً من أجل إنشاء مصفوفة مربعة ذات خمسة أسطر وخمسة أعمدة جميع قيمها واحدات عدا القيمة الأولى والأخيرة أصفار، يمكن استخدام التابع ones لتوليد المصوفة أولاً ثم إسناد قيمة الصفر للعنصر الأول والعنصر الأخير، سوف نسمي هذا التابع بالاسم ones_ze2، وهو لا يحتاج إلى مدخلات ولكن يحتاج إلى متحول واحد للمخرجات هو اسم المصفوفة الناتجة. يمكن كتابة الشيفرة البرمجية لهذا التابع كمايلي:

```
function out=ones_ze2()
out=ones(5);
out(1)=0;
out(end)=0;
```

يمكن استدعاء هذا التابع من نافذة الأوامر بكتابة اسمه فقط ليتم إسناد النتيجة إلى المتحول الخاص ans أو بإسناده إلى متحول ما كما في أي تابع آخر جاهز.

يمكننا الآن تطوير هذا التابع ليقوم بتوليد مصفوفة جميع قيمها تساوي قيمة يختارها المستخدم وبحيث يختار المستخدم أيضاً أبعاد المصفوفة، تعامل هذه المتغيرات كمدخلات للتابع كمايلي:

```
function out=ones_ze2(rows,cols,value)
out=value*ones(rows,cols);
out(1)=0;
out(end)=0;
```

الآن أصبح من الضروري إضافة وصف وشرح للتابع لكي يتمكن المستخدم من معرفة كيفية استخدامه، نضيف الوصف فيصبح التابع بالشكل التالي:

```
function out=ones_ze2(rows,cols,value)
%ones_ze2(rows,cols,value)
%Description:
%This function generates a matrix of 'rows' rows
%and 'cols' columns,
%each value of it is 'value'
%exept the first and end which are zeros
out=value*ones(rows,cols);
out(1)=0;
out(end)=0;
```

عند كتابة الأمر

```
Help ones_ze2
```

في نافذة الأوامر سوف نحصل على النتيجة التالية:

```
>> help ones_ze2
 ones_ze2(rows,cols,value)
 Description:
 This function generates a matrix of 'rows' rows
 and 'cols' columns,
 each value of it is 'value'
```

exept the first and end which are zeros

◄ التوابع الجزئية أو المساعدة

هذه التوابع يتم تعريفها مثل أي تابع آخر ولكن ضمن ملف لتابع آخر هو الأساسي ويسلك التابع الجزئي سلوك المساعد، فهو لا يستدعى أو يستخدم من خارج ملف التابع الأساسي وإنما من قبل التابع الأساسي فقط، كما في المثال التالي:

```
function out=totalArea(width,length,height)
out=baseArea(width,length)+sideArea(width,length,height);

function out=baseArea(w,l)
out=2*l*w;

function out=sideArea(w,l,h)
out=2*(w+l)*h;
```

في هذا المثال يتم تعريف ثلاثة توابع في ملف واحد يسمى باسم التابع الأول `totalArea` الذي يقبل ثلاثة مدخلات هي الطول والعرض والارتفاع لمتوازي مستطيلات ما، التابع الثاني `baseArea` يحسب مجموع مساحتي القاعدتين باستخدام المدخلات التي هي الطول والعرض، والتابع `sideArea` يحسب المساحة الجانبية، يستقبل التابع الأول الطول والعرض والارتفاع ويمررها للتابعين التاليين ليقوما

بالحساب ويرجعا النتيجة له ليرجعها هو بدوره للكود الزبون (الكود الزبون هو الكود الذي قام باستدعاء التابع).

◄ التوابع غير المسماة

التوابع غير المسماة ليست توابعاً بدون أسماء كما يوحي اسمها للوهلة الأولى، وإنما بدون ملفات تحويها، هذه التوابع يمكن تعريفها بطريقة مختلفة قليلاً عن التوابع الأخرى في نافذة أوامر ماتلاب أو ضمن سياق تابع آخر أو إجراء، وتستخدم بطريقة مماثلة للتوابع العادية.

الشكل العام لتعريف التابع غير المسمى هو:

FunctionName=@(in1, in2, …) expression

حيث `functionname` هو اسم التابع و `expression` هو أي كود يستخدم توابعاً جاهزةً و معرفة من قبل المستخدم.

نلاحظ هنا أن اسم التابع أصبح قبل رمز الإسناد وحل محل اسم التابع في التعريف العام الرمز `@` للدلالة على عدم التسمية (عدم تسمية ملف يحوي التابع)، وكذلك لا يوجد متحولات مخرجات، فاسم التابع نفسه هو متحول الخرج وهو وحيد. لنعرف تابعاً في نافذة أوامر ماتلاب كمايلي:

```
parabola=@(x) x.^2;
```

لاستخدام هذا التابع يمكن إسناده إلى أي متحول وتمرير مصفوفة x له كبارامتر دخل كمايلي:

```
>> a1=parabola(2);
>> a1=parabola([2:10]);
```

يمكن أيضاً أن يستخدم التعبير المعرف للتابع غير المسمى قيم متحولات معرفة مسبقاً كمايلي:

```
>> a=3;b=4;
>> parabola=@(x) a*x.^2+b*x;
```

في هذه الحال يقوم ماتلاب باستخدام القيم المخزونة في المتحولين a و b في تعريف التابع parabola ولا يربط التابع بهذين المتحولين، بمعنى أنه إذا تغيرت قيمة أحد هذين المتحولين أو قيمتاهما معاً فلا يتغير التابع parabola فقد انقطعت علاقته بهما، والمثال التالي يوضح ذلك:

```
>> a=3;b=4;
>> parabola=@(x) a*x.^2+b*x;
>> parabola(4)
ans =
```

```
   64
>> a=5;
>> parabola(4)
ans =
   64
```

مثال آخر على استخدام التابع غير المسمى في تابع آخر:

```
function out=paraF(x)
para=@(k) k.^2;
out=para(x);
```

◄ التوابع السطرية

التوابع السطرية `inline functions` تماثل التوابع غير المسماة في طريقة تعريفها ضمن نافذة الأوامر أو ضمن تابع آخر دون ملف يحويها، وهي طريقة قديمة في ماتلاب تم استبدالها بالتوابع غير المسماة ولا ينصح باستخدامها. يعرف التابع السطري كمايلي:

```
FunctionName=inline('expression')
```

حيث يكتب التعبير الرياضي `expression` كنص.

مثال:

```
>> cu=inline('x.^3');
>> cu(3)
ans =
   27
```

◄ المتحولات العمومية

رأينا كيف يمكن كتابة مجموعة من الأوامر في ملف m-file لاستخدامه كإجراء برمجي، ورأينا كيف يمكن كتابة تابع معرف من قبل المستخدم دون مخرجات من النوع void ليستخدم كإجراء برمجي، ولكن ما الفرق؟

في الحالة الأولى عند كتابة الأوامر في ملف m-file فإن ماتلاب يقوم بتنفيذ تلك الأوامر وكانها تدخل في نافذة الأوامر وكل متحول يتم تعريفه في تلك الأوامر يحجز له مكان في ذاكرة مساحة العمل العامة التي تستخدمها نافذة الأوامر، أما في الحالة الثانية فالمتحولات التي يعرفها الإجراء التابعي تكون خارج مساحة العمل العامة حيث أن لكل تابع مساحة عمل خاصة به وهو لا يشاركها مع التوابع الأخرى حتى لو كانت توابعاً جزئية ضمن ملفه الخاص، بكلمات أخرى يكون المتحول محلياً بالنسبة للتابع، وهذا فرق هام بين الإجرائين التابعي وغير التابعي.

يمكن استخدام المتحول في التابع وخارج التابع أي مشاركته بين تابعين أو أكثر أو بين تابع ونافذة أوامر ماتلاب باستخدام الأمر global variablename في كل مكان نحتاج فيه للمشاركة، على سبيل المثال إذا أردنا مشاركة المتحول x بين

التابع `funSub` ونافذة الأوامر نكتب `global x` في نافذة الأوامر وفي كود التابع كمايلي:

```
function funSub()
global x;
y=x+2;
x=y;
```

في نافذة الأوامر:

```
>> global x
>> funSub
```

◄ أداة الشرط if

أداة الشرط هي أول ما ينبغي تعلمه من أدوات التحكم بسير البرنامج فبواسطتها يمكن جعل البرنامج يسلك سلوكاً مختلفاً وفقاً لشرط ما. البنية الهيكلية العامة لأداة الشرط هي:

```
if condtion
   statements
elseif condtion
   staements
elseif condtion
   statements
```

```
else
   statements
end
```

حيث يمكن أن يكون الشرط condition أي تعبير منطقي بسيطاً أو مركباً، والتعبير المنطقي هو كل تعبير يؤول حسابه إلى إحدى القيمتين المنطقيتين true أو false أما كتلة أوامر التنفيذ فهي مجموعة من الأوامر من أي شكل أو حجم فهي تعابير رياضية أو حسابية أو استدعاء لإجراءات برمجية أو حتى كتلة شرط جديدة ضمنية في الكتلة الأساسية، يتم تنفيذ كتلة أوامر التنفيذ في حال تحقق الشرط أي في حال كانت نتيجة حسابه هي القيمة المنطقية true وفي حال عدم تحققه ينتقل البرنامج لاختبار الشرط الثاني التالي لعبارة elesif الأولى دون تنفيذ كتلة الأوامر الأولى، فإن تحقق الشرط الثاني يتم تنفيذ كتلة الأوامر الثانية وإلا ينتقل البرنامج لاختبار الشرط الثالث وهكذا حتى الوصول إلى العبارة else إن لم يتحقق أي شرط يسبقها فيتم تنفيذ كتلة الأوامر التالية لها. في حال تحقق أي شرط وتنفيذ كتلة الأوامر التالية له يتم الانتقال مباشرة بعدها إلى نهاية كتلة الشرط دون النظر لبقية الشروط وكتل الأوامر المتعلقة بها.

في أي أداة شرط تكون المكونات الصغرى الممكنة هي:

```
if condtion
   statements
```

end

وهو الشكل الأبسط لأداة الشرط.

تستخدم العبارة `elseif` في حال الحاجة إليها بأي عدد من المرات، أما العبارة `else` فتستخدم مرة واحدة فقط إن كانت لها حاجة.

في المثال التالي سنقوم بكتابة كود في نافذة الأوامر يطلب من المستخدم إدخال عدد صحيح ثم يختبر إذا كان هذه العدد زوجياً أم لا ويعيد النتيجة بشكل نص على الشاشة، نستخدم لطلب الإدخال من المستخدم التابع `input` الذي يظهر رسالة للمستخدم تطلب منه الإدخال ويوقف البرنامج حتى يقوم المستخدم بالإدخال والضغط على المفتاح `Enter` ثم يسند ما أدخله المستخدم إلى المتحول المستخدم في عبارته للإسناد. يمكن كتابة هذا الكود بالشكل التالي:

```
>> x=input('Enter an integer: '),if mod(x,2)==0,
disp('your number is even'),else,disp('your number is odd'),
end
```

كما نلاحظ يمكن تجزئة الكود على عدة أسطر دون استخدام النقاط الثلاث حيث يفهم ماتلاب هنا أن العبارة غير منتهية لأنه لا يقوم بتنفيذ كتلة الشرط مالم تنتهِ بالكلمة

المفتاحية end ولكن لا يمكن الضغط على مفتاح الإدخال بعد عبارة input لأن هذا سيؤدي إلى التنفيذ كونها عبارة منتهية. يمكن كتابة الكود السابق بشكل آخر كمايلي:

```
>> x=input('Enter an integer: '),...
if mod(x,2)==0
disp('your number is even')
else
disp('your number is odd')
end
```

نلاحظ هنا وضع الفاصلة بعد عبارة input للدلالة على انتهاء العبارة، ووضع النقاط الثلاث التي تخبر ماتلاب أن ينتظر في التنفيذ حتى اكتمال الكود المطلوب تنفيذه، أما بعد العبارات الجزئية لعبارة الشرط فلا داعي لوضع أي شيء.

عند تنفيذ السابق بأحد شكليه يعرض ماتلاب الرسالة المدخلة كبارامتر في عبارة input وعندما يدخل المستخدم قيمة ما ويضغط مفتاح الإدخال Enter يسند ماتلاب القيمة المدخلة إلى المتحول x ويتابع التنفيذ وهو الاختبار هنا.

لنأخذ مثالاً آخر ولنكتب الكود هذه المرة في ملف m-file للتعامل معه كتابع يعيد قيمة نعم أو لا (قيمة منطقية)، في هذا المثال نريد اختبار العدد الدال على سنة ما لمعرفة إن كانت تلك السنة كبيسة أم لا. تكون السنة كبيسة إذا كان العدد الدال عليها قابلا للقسمة على العدد 4 وليست من سنوات رؤوس القرون (ليست آخر سنة في

قرن) أما إذا كانت رأس قرن فيجب أن يقبل العدد الدال عليها القسمة على 400 (في الواقع موضوع السنة الكبيسة أعقد قليلاً ولكن لا نريد الدخول في تعقيداته هنا حيث نهدف إلى توضيح المثال البرمجي فقط). تنطلق فكرة الحل إذن من إجراء اختبار أولي على السنة فإن كانت نهاية قرن يتم اختبار قابلية قسمتها على 400 وإن لم تكن يتم اختبار قابلية قسمتها على 4، يمكن كتابة الكود بالشكل التالي:

```
function leap=isLeap(y)
if mod(y,100)==0
   if mod(y,400)==0
      leap=true;
   else
      leap=false;
   end
else
   if mod(y,4)==0
      leap=true;
   else
      leap=false;
   end
end
```

عند طلب هذا التابع من نافذة الأوامر يتم العمل كمايلي:

```
>> isLeap(2015)
ans =
   0
```

```
>> isLeap(2016)
ans =
   1
```

طبعاً هذا الكود طويل بالنسبة لمثالنا ويمكن اختصاره بعدة طرق، أكثر الطرق اختصاراً هي كمايلي:

```
function leap=isLeap(y)
leap=false;
if mod(y,400)==0 || (mod(y,100)~=0 && mod(y,4)==0)
leap=true;
end
```

حيث أعطينا المتحول `leap` قيمة افتراضية `false` ثم قمنا باختبار الشرط المركب انطلاقاً من فكرة رياضية منطقية تقول بأنه إذا كان العدد قابلاً للقسمة على `400` فهو بالتأكيد يقبل القسمة على `100` وبالتالي فالسنة هي نهاية قرن وتحقق شرط الكبيسة فهي كبيسة (لا يهم ما تكون القيمة بعد عملية `||`) أما إن لم تكن قابلة للقسمة على `400` فيجب التأكد من كونها ليست نهاية قرن وتكون كبيسة إذا كانت لا تقبل القسمة على `100` وتقبل القسمة على `4` في آن واحد، إن لم يتحقق أي من هذا فإن كتلة الشرط لا تغير القيمة الابتدائية للمتغير `leap`.

لنأخذ مثالا آخر يستخدم العبارة elseif، في هذا المثال نريد كتابة تابع يقرأ عدداً بين الواحد والعشرة ويعيد العدد نصاً، سوف نحتاج لاختبار العدد عدة اختبارات لمعرفة قيمته ثم إظهار النتيجة المناسبة كمايلي:

```
function t=n2text(n)
if n==0
   t='zero';
elseif n==1
   t='one';
elseif n==2
   t='two';
elseif n==3
   t='three';
elseif n==4
   t='four';
elseif n==5
   t='five';
elseif n==6
   t='six';
elseif n==7
   t='seven';
elseif n==8
   t='eight';
elseif n==9
   t='nine';
elseif n==10
   t='ten';
end
```

تخيل أنك تريد تطوير التابع ليعيد العدد نصاً للاعداد حتى الألف أو العشرة آلاف! طبعا لن تكتب آلاف الأسطر المتشابهة، هناك طرق أبسط من ذلك ولكن ليس هذا الأمر ضمن هدف هذا الكتاب، سوف أعرض فقط طريقة أبسط للمثال السابق، كمايلي:

```
function t=n2text(n)
num={'zero' 'one' 'two' 'three' 'four' 'five' 'six' 'seven' 'eight' 'nine' 'ten'};
if n+1>length(num)
   t='unknown number';
else
   t=num(n+1);
end
```

◄ اختبار الدارة القصيرة

يعبر مصطلح الدارة القصيرة في الاختبارات المنطقية عن طريقة متبعة لتسريع عملية حساب النتيجة المنطقية، حيث يتم الحساب خطوة خطوة فإذا وصل البرنامج إلى النتيجة النهائية يتوقف عن متابعة الخطوات. وهذا ممكن في علم المنطق الرياضي، فكما نعلم تكون نتيجة إجراء العملية المنطقية `and` على تعبيرين منطقيين هي القيمة المنطقية `true` إذا وفقط إذا كان كل من التعبيرين المنطقيين مساويا للقيمة المنطقية `true` وعليه فإننا نقرر أن النتيجة النهائية هي `false` بمجرد معرفة أن

قيمة التعبير الأول هي false أما إذا كانت قيمة التعبير الأول هي true فيجب متابعة الحساب المنطقي والنظر في قيمة التعبير الثاني. بشكل مشابه يمكن أن نقرر أن النتيجة النهائية لعملية or بين تعبيرين منطقيين هي true بمجرد معرفة أن قيمة التعبير الأول هي true. يصلح هذا الأمر لأي عدد من التعابير المنطقية. تستخدم الدارة القصيرة الرمزين && لعملية and و || لعملية or

يمكن إعادة كتابة التابع السابق لاختبار السنة الكبيسة باستخدام الدارة القصيرة كمايلي:

```
function leap=isLeap(y)
leap= (mod(y,400)==0) || (mod(y,100)~=0 && mod(y,4)==0);
```

◄ التوابع التعاودية

يسمى كل تابع يستدعي ذاته تابعاً تعاودياً، ويمكن للتابع أن يعيد طلب ذاته عدداً لانهائياً من المرات وفق شروط محددة، في النهاية لا بد من شرط يحدد انتهاء العملية التعاودية. الأمثلة على التوابع التعاودية كثيرة وسوف نناقش هنا مثالين.

المثال الأول: تابع العاملي

يعرف التابع العاملي رياضياً بالشكل التالي:

$$n! = f(x) = \begin{cases} 1, & x = 0 \; or \; x = 1 \\ n \cdot (n-1)!, & x > 1 \end{cases}$$

من علاقة الربط هذه نجد أن التابع العاملي لأي عدد هو ذلك العدد مضروباً بقيمة التابع العاملي للعدد السابق له، مما يعني أن التابع يعيد حساب نفسه نزولاً وهذا هو مبدأ التعاودية في التوابع البرمجية سواءً في ماتلاب أو غيره. يمكن كتابة شيفرة التابع العاملي كمايلي:

```
function f=fac(n)
if n==1, f=1;return,end
f=n*fac(n-1);
```

وهي بسيطة جداً كما نرى مع الانتباه إلى ضرورة وجود شرط التوقف عن استدعاء التابع لنفسه.

يعمل الأمر `return` على إيقاف تنفيذ التابع الحالي وإعادة مؤشر التنفيذ إلى المكان الذي طلب هذاالتابع وهو ربما يكون تابعاً آخر أو أمر في نافذة الأوامر، أي أن كل ما يلي الأمر `return` لا يتم تنفيذه.

المثال الثاني: إيجاد القاسم المشترك الأكبر لعددين

وفقاً لخوارزمية اقليدس فإن القاسم المشترك الأكبر لعددين صحيحين هو أيضا قاسم مشترك أكبر للعدد الصغير منهما وباقي قسمة كبيرهما على صغيرهما إذا كان الصغير لا يقسم الكبير، وإلا فالقاسم المشترك الأكبر هو العدد الصغير ذاته. يعني هذا أننا نستطيع تعريف تابع للقاسم المشترك الأكبر لعددين بالشكل التالي:

$$\gcd(m,n) = f(x) = \begin{cases} n, & m \bmod n = 0 \\ \gcd(n, m \bmod n), & m \bmod n \neq 0 \end{cases}$$

نفترض هنا أن العدد m هو العدد الكبير، ونقصد بالعملية mod باقي القسمة. يتضح من التعريف السابق أن عملية حساب القاسم المشترك الأكبر بهذه الطريقة هي عملية تعاودية، ويمكن كتابة تابع في ماتلاب يقوم بالحساب كمايلي:

```
function f=gcd(m,n)
if m<n,temp=m;m=n;n=temp;end
if mod(m,n)==0,f=n;return,end
f=gcd(n,mod(m,n));
```

في السطر الثاني نقوم بالتبديل بين العددين في حال كون العدد الثاني أكبر من الأول.

◄ أداة الاختيار الشرطي switch

تستخدم هذه الأداة عند وجود تعبيير يؤول حسابه إلى قيمة عددية مفردة أو تعبير نصي، ويراد إجراء عمل ما ضمن البرنامج بناء على القيمة المحسوبة من ذلك التعبير. الشكل العام لهذه العبارة هو:

```
switch expression
 case case_expr
   statements
 case {case_expr1, case_expr2, case_expr3, ...}
   statements
 otherwise
   statements
end
```

حيث يتم حساب القيمة ومقارنتها بالحالة `case_exp` فإن تم التساوي يتم تنفيذ الكتلة البرمجية التالية لهذه الحالة، بعد ذلك يتم اختبار تحقق الحالة التالية وتنفيذ الكتلة البرمجية التالية لها في حال التحقق، في النهاية يتم تنفيذ الكتلة البرمجية التالية لعبارة `otherwise` وهي تقابل حالة عدم تحقق أي مما سبق.

كمثال على استخدام هذه العبارة يمكن أن نكتب تابعاً يقرأ عدداً صحيحاً ويعيد اسم اليوم الذي يعبر العدد المقروء عنه في سياق أيام الأسبوع، يمكن كتابة شيفرة هذا التابع كمايلي:

```
function d=n2day(n)
switch n
   case 1
      d='Saturday';
   case 2
      d='Sunday';
   case 3
      d='Monday';
   case 4
      d='Tuesday';
   case 5
      d='Wednesday';
   case 6
      d='Thursday';
   case 7
      d='Friday';
   otherwise
      d='unknown day';
end
```

◄ الحلقة الشرطية while

تستخدم عبارة الحلقة الشرطية `while` لتكرار عمل ما (مجموعة من الأوامر البرمجية) عدداً غير محدد من المرات يعتمد على تحقق شرط ما وفي كل مرة تقوم الحلقة باختبار تحقق الشرط وتنفذ كتلتها البرمجية في حال التحقق أما في حال عدم

التحقق فينتقل البرنامج إلى ما بعد الحلقة التي أتمت عملها. الشكل العام لحلقة while الشرطية هو:

```
while expression, statements, end
```

وكمثال عليها يمكن أن نكتب تابعاً بسيطاً يقوم بإضافة عدد محدد على كافة عناصر مصفوفة مدخلة كمايلي:

```
function d=tri(n)
i=1;
while i<=length(n)
   n(i)=n(i)+10;
   i=i+1;
end
d=n;
```

ينبغي الانتباه هنا إلى ضرورة التأكد من وصول الحلقة إلى حالة عدم تحقق الشرط وإلا فسوف نحصل على حلقة أبدية التكرار (سوف يؤدي هذا إلى تعليق التطبيق).

هناك أيضاً حالات نحتاج فيها إلى الخروج من الحلقة عند تحقق شرط ما غير شرطها ويستخدم لهذا الغرض الأمر break كما في المثال التالي:

```
function d=lcm(m,n)
lower=m;
upper=n;
```

```
if lower>upper,lower=n;upper=m;end
i=1;
while i<=lower
   upper=upper*i;
   i=i+1;
   if mod(upper,lower)==0,brea,end
end
d=upper;
```

في هذا المثال نستخدم حلقة while لحساب مضاعفات عدد من أحد عددين ثم اختبار إن كان المضاعف المحسوب هو مضاعف أيضاً للعدد الثاني، فإذا تحقق ذلك يتم الخروج من الحلقة بالأمر break وإلا فالحلقة تتابع حتى الوصول إلى المضاعف الذي هو جداء العددين ولا بد أن يكون هذا الجداء هو مضاعف للعدد الآخر، عندها تتوقف الحلقة بسبب شرطها.

في الواقع هناك حالات يجب فيها استخدام هذه الطريقة وهي حالات معقدة قليلا لن ندخل في نقاشها في هذا السياق، أما في مثالنا هذا فيمكن الاستغناء عن هذه التقنية وتركيب الشرطين ليكونا معا شرط الحلقة كمايلي:

```
function d=lcm(m,n)
lower=m;
upper=n;
if lower>upper,lower=n;upper=m;end
```

```
i=1;
while i<=lower && mod(upper,lower)~=0
   upper=upper*i;
   i=i+1;
end
d=upper;
```

◄ حلقة for

يمكن استخدام حلقة while الشرطية في أي عمل تكراري مهما كان ولا حاجة فعلية لوجود نوع آخر من الحلقات، ولكن هناك حالات يكون فيها ممكناً استخدام حلقة for التي هي حالة خاصة من الحلقة الشرطية، فهي شرطية أيضاً، ولكن تم ترتيبها بطريقة خاصة، ذلك أن التكرار يراد إجراؤه عدداً محدداً من المرات باستخدام متحول يدعى متحول الحلقة له قيمة ابتدائية وقيمة نهائية وخطوة تزايد أو تناقص، تنتهي الحلقة عندما تتجاوز قيمة المتحول مجال القيم المحصور بين القيمة الابتدائية والقيمة النهائية سواء صعوداً أو نزولاً.

الشكل العام لحلقة for هو:

```
for x=initval:stepval:endval, statements, end
```

لا يمكن استخدام حلقة for في بعض الحالات عندما لا يكون ممكناً تحديد القيمة النهائية أو الخطوة.

في المثال التالي نستخدم حلقة for للبحث عن القيم الأكبر من 5 في متجه أفقي من الأعداد الصحيحة:

```
>> x=reshape(magic(3),1,9);
>> count=0;for i=1:length(x),if x(i)==5,count=count+1,end,end
```

لم نذكر هنا خطوة الحلقة وهذا يعطيها القيمة الافتراضية المساوية للواحد. بالطبع يمكن إنجاز هذا العمل بدون حلقة كمايلي:

```
>> sum(x==5)
```

في المثال التالي نستخدم حلقتين متداخلتين للبحث عن القيم الأكبر من 5 في مصفوفة ثنائية البعد:

```
>> x=magic(3);
>> count=0;for i=1:3,for j=1:3,if x(i,j)==5,count=count+1,end,end,end
```

يمكن عمل هذا بطريقة أخرى بواسطة حلقة واحدة كمايلي:

```
>> count=0;for i=1:prod(size(x)),if x(i)==5,count=count+1,end,end
```

أو بالطريقة السابقة بدون حلقات.

لناخذ مثالاً آخر، في هذا المثال نريد كتابة تابع يوجد المضاعف المشترك الأصغر لعددين باستخدام حلقة for (كتبنا هذا التابع سابقاً باستخدام حلقة while)، يمكن التفكير بالأمر كمايلي: لإيجاد المضاعف المشترك الأصغر لعددين يمكن البدء بإيجاد مضاعفات العدد الكبير واحداً تلو الآخر بدءاً بالعدد الكبير نفسه وفي كل مرة نختبر إن كان المضاعف المحسوب مضاعفاً للعدد الصغير أيضا فإن كان نوقف عمل الحلقة ونكون قد وجدنا المضاعف المشترك الأصغر، أما إن لم يكن فنتابع حتى الوصول إلى العدد الناتج عن ضرب العددين وهو بالتأكيد مضاعف مشترك لهما.
يمكن كتابة شيفرة هذا التابع كمايلي:

```
function c=lcm4(m,n)
for c=m:m:m*n
   if mod(c,n)==0
      return
   end
end
```

يمكن استخدام حلقة for بطرق أخرى كمايلي:

حلقة ذات خطوة غير ثابتة تؤخذ من متجه، كما في المثال التالي:

```
>> s=[1 3 4 7 4 11 2 4];
>> count=0;for x=s,if x==4,count=count+1,end,end
```

في هذا المثال يأخذ متحول الحلقة قيمه من قيم المتجه s المعرف في السطر السابق.

حلقة ذات أكثر من متحول لكل منها قيمته الابتدائية والنهائية، كما في المثال التالي:

```
>> for i=1:3,j=1:4,xx(i,j)=i+j,end
```

في هذا المثال لدينا متحولان لحلقة واحدة، تقوم الحلقة بإعطاء كافة التزايدات للمتحول الثاني مقابل كل قيمة ممكنة للمتحول الأول. نتيجة تنفيذ هذا المثال ستكون مصفوفة ثنائية البعد كل عنصر منها يساوي مجموع رقمي سطره وعموده.

◄ التخطي والكسر

لنتأمل المثال التالي الذي يستخدم حلقة for لتعبئة متجه أفقي بالأعداد من الواحد إلى العشرة:

```
>> for i=1:10,x(i)=i;end
```

نتيجة التنفيذ ستكون متجها أفقيا كمايلي:

```
x = 1   2   3   4   5   6   7   8   9   10
```

إذا أردنا من الحلقة أن تتخطى القيمة 5 وتترك مكانها صفراً يمكن عمل ذلك باستخدام الأمر continue كمايلي:

```
>> for i=1:10,if i==5,continue,end,x(i)=i;end
x = 1   2   3   4   0   6   7   8   9   10
```

الأمر continue هنا يلغي تنفيذ كل الأوامر التالية له في كتلة الحلقة ويعود بمؤشر التنفيذ إلى متحول الحلقة ليأخذ تزايداً أو تناقصاً جديداً وتسنمر الحلقة، استخدمنا هنا أداة الشرط if لإجراء التخطي عند القيمة 5 فقط لمتحول الحلقة.

في حالات أخرى نريد إيقاف تنفيذ الحلقة بشكل كامل وهنا نستخدم الأمر break بدلاً من continue كمايلي:

```
>> for i=1:10,if i==5,break,end,x(i)=i;end
x = 1   2   3   4
```

◄ تعقب الأخطاء وإصلاحها

يعتبر تعقب الأخطاء وإصلاحها موضوعاً كبيراً ومتشعباً وهو يحتاج إلى خبرة ومعرفة جيدة بالأوامر والتعليمات والآلية التي تعمل بها لغات البرمجة والمفسرات والمترجمات، ليس الأمر مقتصراً على برنامج صغير مكون من عدة أسطر وإنما

هناك حالات يكون فيها البرنامج كبيراً كفايةً لكي يجعل المبرمج قليل الخبرة يصاب باليأس والإحباط عندما تبدأ الأخطاء بالظهور.

في هذا الكتاب لن نناقش تعقب الأخطاء بالتفصيل العميق إذ إن هذا الأمر يتطلب كتاباً لوحده ولكن سوف نتكلم عن أهم المواضيع بشكل مختصر.

◄ أنواع الأخطاء

تختلف الأخطاء الممكن حدوثها وتتنوع من حيث أثرها على سير البرنامج وطريقة تصحيحها وصعوبة ذلك التصحيح، ومن أبرز أنواعها:

الأخطاء النحوية: يقصد بها الأخطاء التي تتعلق بطريقة كتابة الأوامر كأن ننسى كتابة حرف من الأمر أو نضع قوساً زائداً أو نكتب حلقة بدون أمر النهاية end وأشياء مماثلة. هذه الأخطاء تمنع البرنامج من العمل نهائياً بغض النظر عن موقعها فهو لا يعمل حتى يكون الملف صحيحاً من الناحية النحوية.

الأخطاء التنفيذية: هذه الأخطاء لا تظهر إلا عند التنفيذ، كأن نحاول ضرب متجهين أفقيين ضرباً مصفوفياً أو ضرب متجهين غير متساويي الطول ضرباً خطياً وغير ذلك من الأخطاء. هذه الأخطاء تظهر رسائل خطأ عند حدوثها وتسبب توقف البرنامج عن المتابعة، يمكن معالجة الأمر بواسطة بعض التقنيات كما سنرى.

الأخطاء المنطقية: هذه هي أكثر الخطاء إزعاجاً فهي تحدث دون أن يظهر أثرها بشكل مفاجئ ويتابع البرنامج عمله حتى النهاية لكننا لا نحصل على النتائج المتوقعة بسبب خطأ منطقي لا نعرف مكان وجوده، ولجعل الأمر أكثر سوءاً فقد نحصل على النتائج الصحيحة في المثال الذي نجربه ولكن تحدث المشاكل عندما يجرب شخص آخر حالة غير تلك التي جربناها. يجب تفادي هذه الأخطاء بالتفكير جيداً بمنطقية البرنامج وآليته وجعل الكود نظيفاً ما أمكن، ثم علينا أن نجرب كل الحالات الممكنة بما فيها الحالات الحدية والحرجة، وأخيراً عند وجود خطأ منطقي يمكن الاستعانة بمتعقب الأخطاء للكشف عن موقع الخطأ أما إصلاح الخطأ فهو مهمة المبرمج.

◄ تغيير رد فعل البرنامج مع الخطأ

في الحالة العادية يقوم ماتلاب بإيقاف تنفيذ البرنامج عندما يصادف خطأ، فهو يظهر رسالة خطأ باللون الأحمر مفيداً بأن خطأ ما حدث وإنه لا يستطيع العمل هكذا وسوف يتوقف عن المتابعة! يمكن تغيير هذا السلوك باستخدام عبارة try … catch ووضع الأوامر المتوقع منها أن تسبب أخطاء ضمنها، كما يلي:

```
Try
Commands 1
Catch e
Commands 2
end
```

حيث يقوم ماتلاب أولاً بمحاولة تنفيذ كتلة الأوامر commands1 وفي حال مصادفته أي خطأ فإنه ينتقل مباشرة لتنفيذ الأوامر commands2 وإذا صادف خطأ هناك فإنه يوقف التنفيذ كعادته. في حال انتهاء تنفيذ الأوامر commands1 دون أخطاء يتخطى ماتلاب كتلة catch كاملةً. يمكن لكتلة try أو كتلة catch أن تتضمن كتلة try catch … أخرى.

لنأخذ مثالاً صغيراً، بفرض أننا كتبنا ملف m-file وحفظناه بالاسم doo.m وفي هذا الملف نكتب الأوامر التالية:

```
x=input('Enter the first vector horizontaly: ');
y=input('Enter the second vector vertically: ');
p=x*y;
```

هذا البرنامج يطلب من المستخدم إدخال متجهين، الأول أفقي والثاني عمودي ليقوم بعد ذلك بضربهما ضرباً مصفوفياً. الأخطاء المحتملة هنا كثيرة فالمستخدم قد يدخل المتجهين أفقيين أو عموديين، وقد يدخل متجهاً أطول من الآخر وقد يترك أحد المتجهين فارغاً، إذن علينا معالجة هذه الأخطاء ولتسهيل الأمور لنفرض أولاً أن خطأ محتملاً واحداً هو ما يمكن أن يحدث وهو أن يدخل المستخدم متجهين أفقيين، حل هذه المشكلة هو الضرب بمنقول المتجه الثاني بدلاً منه كمايلي:

```
x=input('Enter the first vector horizontaly: ');
y=input('Enter the second vector vertically: ');
try
   p=x*y;
catch e1
   p=x*y';
end
```

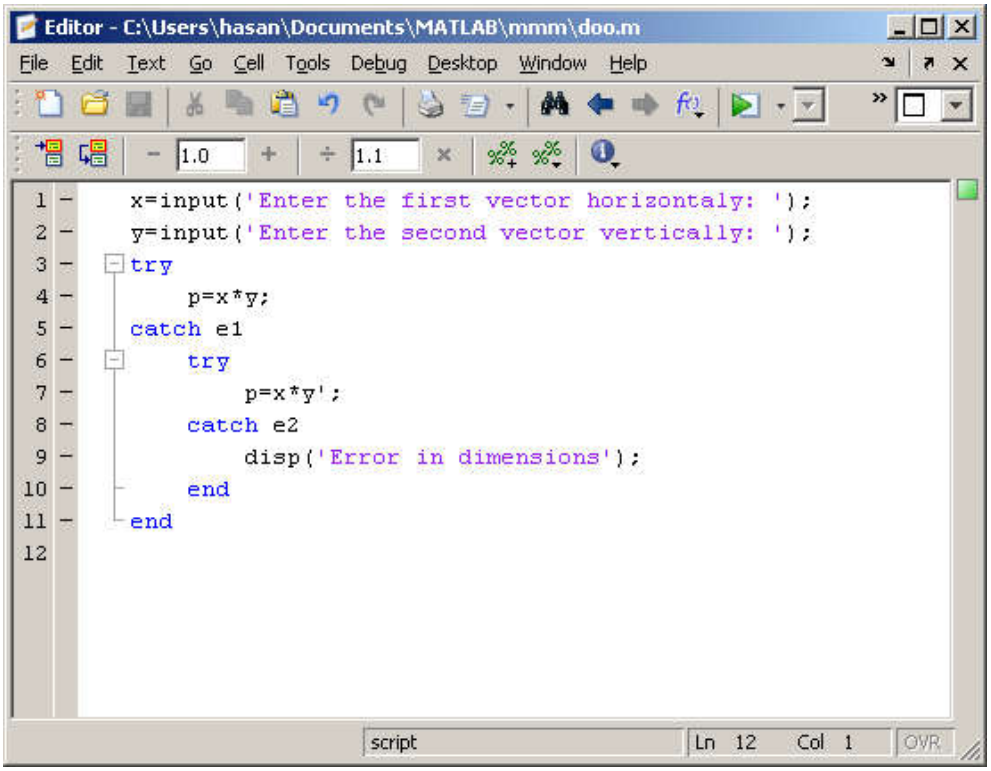
في حال حدوث هذا الخطأ سيقوم ماتلاب بتحويل التنفيذ إلى عبارة `catch` ليتم الضرب بالمنقول، ولكن في حال إدخال متجهين أحدهما أطول من الآخر يقوم ماتلاب أيضاً بنفس العمل ويحصل هنا خطأ في عبارة `catch` يؤدي إلى التوقف، لحل المشكلة يمكن إضافة كتلة `try … catch` أخرى كمايلي:

```
x=input('Enter the first vector horizontaly: ');
y=input('Enter the second vector vertically: ');
try
   p=x*y;
catch e1
   try
      p=x*y';
   catch e2
      disp('Error in dimensions');
   end
end
```

من الهام جدا اختبار صحة الإدخال وعدم الاقتصار على استخدام عبارات try … catch فهذه العبارات عادة تستخدم للاخطاء غير المتوقعة وربما نستخدم في كتلة catch أمراً يتم تنفيذه في كل الأحوال وثم يتابع ماتلاب تنفيذ كود بعد نهاية العبارة وهذا لضمان استمرار عمل البرنامج في حال حدوث خطأ لا يؤثر على الأوامر التالية لعبارة catch.

◄ ملاحظة الاخطاء النحوية على محرر ملفات ماتلاب

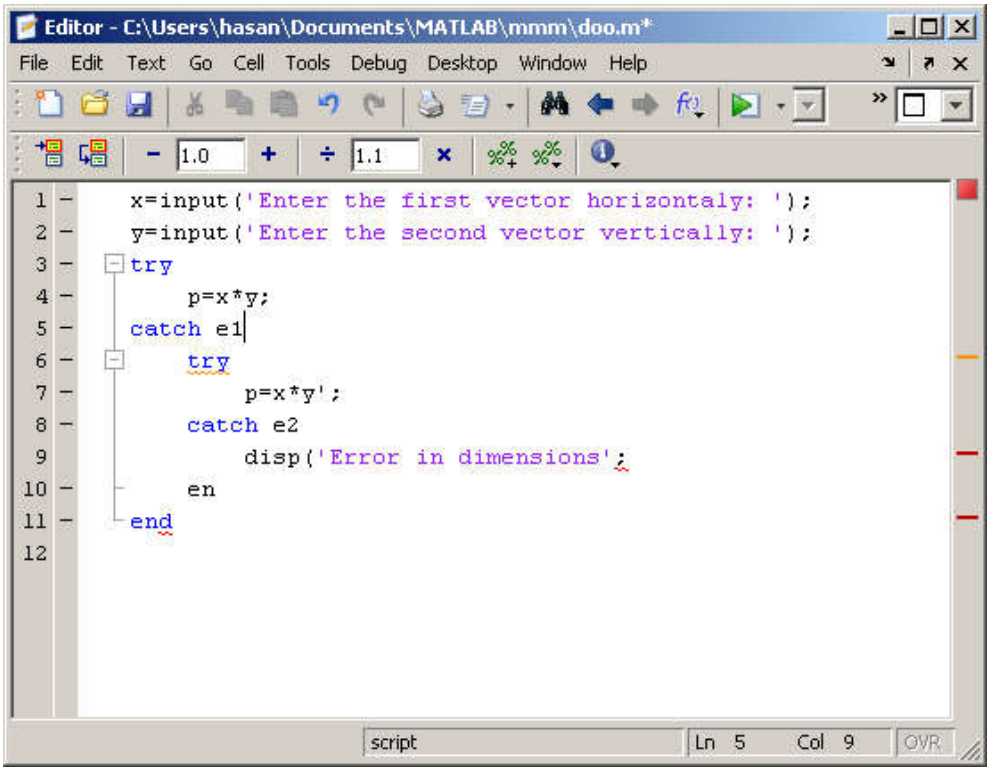
يوفر محرر ملفات ماتلاب بعض الإمكانيات لمساعدتنا في إصلاح الأخطاء النحوية قبل محاولة تشغيل البرنامج، فهو يظهر التلميحات والتحذيرات على الهامش الخاص بذلك في أيمن الشاشة كما يظهر الشكل 38



الشكل 38

تظهر العلامة الخضراء على يمين الملف أن الكود كله صحيح ولا يوجد أي أخطاء، وعند التوقف عليها بالماوس يظهر التلميح مخبراً بذلك.

لنتعمد الآن إحداث بعض الأخطاء لنرى سلوك المحرر، سنقوم بحذف أحد الأقواس وحذف حرف من آخر كلمة end، عندها سيظهر المحرر تحذيراته ويغير لون الإشارة إلى الأحمر كما في الشكل 39



الشكل 39

إذا وضعنا مؤشر الماوس فوق الإشارة البرتقالية سيظهر لنا تلميح يفيد بأن عبارة `try` ليست محاذية لعبارة `end` الموافقة لها فماتلاب يعتبر أن `end` الأخيرة تتبع `try` الثانية وأن `try` الأولى ليس لها `end`، يؤكد لنا ذلك التلميح الذي نقرؤه عند الوقوف بالماوس فوق الإشارة الحمراء الأخيرة. أما الإشارة الحمراء الأولى فهي تحذرنا من فقدان أحد الأقواس وهي الأسهل في التصحيح حيث نبحث عن القوس الضائع ونضعه في مكانه، أما الخطأ الآخر فهو مربك لو لم نكن اصطنعناه ولو كان

الملف كبيراً كفايةً. الخطأ الذي يظهر باللون البرتقالي يعني أن البرنامج قد يعمل رغم وجود الخطأ ولكن من المستحسن إصلاحه، أما الخطأ الذي يظهر باللون الأحمر فيعني أن البرنامج لن يعمل حتى يتم الإصلاح.

◄ تعقب الأخطاء المنطقية

ليكن لدينا الإجراء غير التابعي (السكريبت) التالي:

```
x=input('Enter the first number: ');
y=input('Enter the second number: ');
z=input('Enter the third number: ');
if x^2==y^2+z^2
   area=y*z/2;
elseif y^2==x^2+z^2
   area=x*z/2;
else
   area=y*z/2;
end
disp(area);
```

يطلب هذا الإجراء من المستخدم إدخال ثلاثة أعداد ثم يقوم بحساب مساحة المثلث القائم الذي أطوال أضلاعه هي تلك الأعداد ويظهر النتيجة. يبدو هذا الإجراء صحيحاً من الناحية النحوية فلا أخطاء من قبيل الأحرف الناقصة أو الزائدة أو ضياع

الأقواس، ولكنه سوف يعاني الكثير من الأخطاء التشغيلية وهذه الأخطاء سوف تسبب توقف البرنامج عن العمل كما قلنا سابقا، من هذه الأخطاء المحتملة أن يدخل المستخدم متجهاً أفقياً أو عمودياً بدلاً من القيمة المفردة في أحد المتحولات، في هذه الحالة سوف يقوم ماتلاب بإظهار الخطأ ويتوقف عن المتابعة.

هناك أخطاء تشغيلية لا تسبب توقف البرنامج هنا كأن يدخل المستخدم القيم ويترك إحداها فارغة بالضغط على مفتاح الإدخال دون إدخال قيمة، في هذه الحالة لا يظهر ماتلاب أي خطأ لأنه في الواقع لا يرى خطأ ويقوم بالحساب بشكل طبيعي لكن النتيجة ستكون غير متوقعة فالمساحة ستكون مصفوفة فارغة. هذا الخطأ يدعى خطأ منطقياً لأن ماتلاب لا يراه ويقوم بالحساب وكأن كل شيء صحيح رغم أن الخطأ يحدث بسبب خلل في الإدخال.

هناك أخطاء أخرى في مثالنا من نوع الأخطاء المنطقية، فالإجراء هنا يفترض أن الأعداد المدخلة تشكل مثلثاً قائماً بالضرورة ولهذا يقوم بالاختبار الأول لمعرفة إن كان العدد x هو الدال على طول الوتر ثم يقوم بالاختبار الثاني لمعرفة إن كان العدد y هو الدال على طول الوتر ثم يفترض في حالة النتيجة السلبية للاختبارين أن العدد z هو الدال على طول الوتر، هذا صحيح لو أدخل المستخدم أطوال أضلاع مثلث قائم فعلاً، ولكن إن لم يكن المثلث قائماً فسوف يفشل الاختباران الأول والثاني ويقوم الإجراء بحساب مساحة خاطئة ويورط المستخدم الذي يثق بالإجراء فهو لا يظهر أي علامة أو إشارة تدل على العمل الخاطئ لأنه أصلاً لا يراه.

هناك أيضاً خطأ منطقي آخر وهو أن البرنامج لا يعالج مشكلة إدخال أعداد سالبة والاختبارات التي يقوم بها لا تميز الموجب من السالب لأنها تختبر مربعات الأعداد ولكن المساحة ستظهر سالبة.

الخطأ الثالث يرتبط بالخطأ الأول ففي حالة إدخال أعداد لا تعبر أصلا عن أطوال أضلاع مثلث فالبرنامج يحسب المساحة ويظهرها، وعدا عن كونها محسوبة بعلاقة خاطئة فهي لا تعبر عن مثلث أصلا.

هناك عدة خطوات لتفادي هذه الإشكالات، فقبل كل شيء يجب دراسة الحالة جيداً قبل البدء بكتابة الكود وينبغي أخذ كل الحالات الغريبة والشاذة والحرجة بعين الاعتبار، كذلك يجب أخذ الأخطاء المحتملة من قبل المستخدم بعين الاعتبار ومعالجة ما أمكن منها.

في الواقع هناك أخطاء منطقية لا بد أن تقع فمهما بلغ ذكاء المبرمج ومهما زادت حنكته قد يصادف حالات تفوته فيها بعض الأمور لا سيما عندما يكون التطبيق الذي يعده معقداً، ولكن إن حدث ووقع خطأ ما وظهر بالتجريب أن خطأ ما حدث في مكان مجهول من البرنامج فلا بد من تتبعه وكشفه وإصلاحه. إعادة التفكير بالبرنامج ككل أمر مؤلم وشاق ولهذا ييجب أن تتوفر آليات للتبع الخطأ وهي موجودة بالفعل، وفيمايلي سوف نناقش تلك الآليات.

◄ التشغيل في وضع تعقب الاخطاء

في العادة نقوم بتشغيل الإجراء أو البرنامج بشكل طبيعي عن طريق استدعائه وهكذا يقوم ماتلاب بتنفيذ الأوامر فيه واحداً تلو الآخر دون توقف. توفر محررات لغات البرمجة ومنها محرر ملفات m-files إمكانية تشغيل ما يسمى متعقب الأخطاء عن طريق وضع نقاط توقف في البرنامج وإتاحة الفرصة للمبرمج لكي ينتقل بين تلك النقاط في الوقت الذي يشاء مما يسهل عليه معاينة ما يحدث.

لوضع نقطة توقف عند سطر ما من البرنامج يمكن فتح القائمة debug ثم اختيار الأمر set/clear breakpoint فإذا لم يكن هناك نقطة توقف في السطر المحدد يتم إدراج واحدة وإن كانت هناك واحدة أصلاً تتم إزالتها. عند وضع نقطة توقف في سطر ما تظهر بشكل دائرة حمراء على الهامش الأيسر للسطر.

عند تشغيل البرنامج الآن يتم تنفيذ كافة الأوامر قبل السطر المحدد بنقطة التوقف ويضع متعقب الأخطاء مؤشراً بشكل سهم أخضر اللون بجانب السطر مشيراً إلى أن هذا السطر هو ما سوف يتم تنفيذه تالياً، وتالياً هنا تعني عندما يريد المبرمج ذلك. قبل المتابعة يمكن للمبرمج أن يقف بالماوس على أي متحول في البرنامج ليظهر له تلميح يخبره بنوع المعطيات وقيمة المتحول، هذه العملية مفيدة جداً حيث أن المبرمج يمكنه من خلال قراءة هذه المعلومات أن يكتشف ما يحدث فعلا في أثناء التنفيذ وربما يكتشف أن ما يحدث هو غير ما توقعه فينكشف له الخطأ المنطقي أو جانب منه.

بعد استكشاف تلك المعلومات يمكن للمبرمج أن يختار مايلي:

المتابعة إلى نقطة التوقف التالية بالأمر continue من قائمة debug، وفي حال عدم وجود نقاط توقف أخرى سيتم الإجراء عمله إلى النهاية.

أو المتابعة إلى السطر التالي بالأمر step من قائمة debug

أو تخطي الإجراء الحالي إلى نهايته بالأمر step out

في حال كون مؤشر التعقب يقف عند سطر فيه استدعاء لتابع آخر يكون لدينا خياران مختلفان:

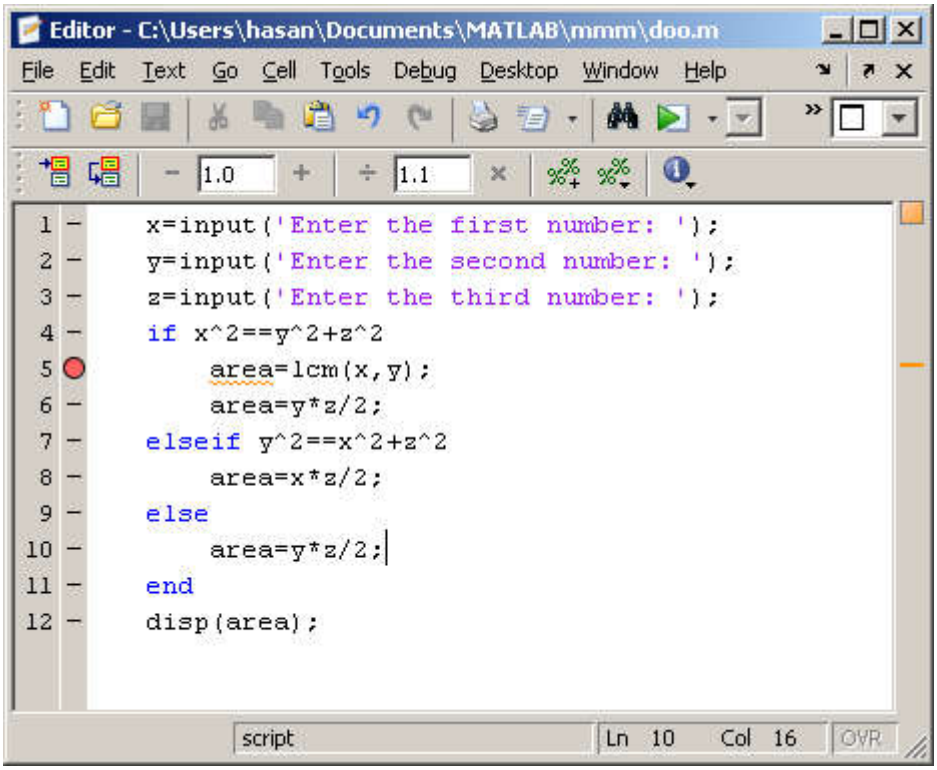
اختيار الأمر step يقوم ماتلاب بتنفيذ السطر كاملاً بما فيه استدعاء التابع وتنفيذه وإرجاع القيمة أو القيم إلى الإجراء الحالي ثم ينتقل المؤشر إلى السطر التالي.

اختيار الأمر step into يقوم ماتلاب بتنفيذ السطر الحالي حتى الوصول إلى استدعاء التابع وعندها يقوم بنقل مؤشر التعقب إلى داخل كود ذلك التابع المستدعى وكأنه جزء من الكود الأصلي، يمكن عندها المتابعة بنفس الطريقة أو استخدام الأمر step out لتنفيذ التابع دفعة واحدة والعودة إلى الإجراء الذي تم الاستدعاء منه.

للتوضيح سنضيف الأمر التالي إلى الكود السابق:

```
area=lcm(x,y);
```

ليصبح الكود الكلي كما في الشكل 40



الشكل 40

هذا الأمر الجديد لا فائدة منه في الإجراء ولكن وضعناه لتوضيح كيفية تعامل متعقب الأخطاء مع استدعاء تابع، وضعنا أيضاً نقطة توقف عند سطر الاستدعاء.

الآن عند تشغيل الإجراء (بالضغط على المفتاح F5 اختصاراً) سيتم تنفيذ الأوامر تباعاً دون توقف حتى الوصول إلى نقطة التوقف في السطر الخامس، وهنا لدينا خياران كما شرحنا سابقاً.

عند وجود مؤشر التعقب عند سطر لا يحوي استدعاء تابع فالأمران step و step into يعملان بشكل متماثل.

البرمجة غرضية التوجه

تميل معظم لغات البرمجة اليوم إلى التحول إلى البرمجة غرضية التوجه Object Oriented Programming (OOP) وماتلاب ليس استثناءً، فقد أصبحت البرمجية غرضية التوجه أساسية فيه وأصبح بالإمكان بناء تطبيقات بواسطته باستخدام هذه المفاهيم، علماً أنه لا يزال يدعم التقنية القديمة.

في هذا الكتاب سوف نشرح مفاهيم البرمجة غرضية التوجه في ماتلاب بشكل مختصر.

◄ مفهوم الصفوف والأغراض والأمثلة

أول المفاهيم التي ينبغي التعرض لها في البرمجة غرضية التوجه هي مفاهيم الصفوف والأغراض والأمثلة، فمن هنا تكون نقطة البداية. للتبسيط يمكن تشبيه العلاقة بين الصف والغرض بالعلاقة بين وصفة قالب الكاتو وقالب الكاتو نفسه، يمكن أن نحصل على وصفة قالب الكاتو التي تذكر فيها المكونات ومقاديرها وطريقة صنعها ولكن لا يمكننا تذوق طعم الكاتو حتى نقوم باتباع تلك الوصفة وصنع قالب الكاتو منها، وبالطبع يمكن صناعة أكثر من قالب كاتو من وصفة واحدة ولكل منها نكهتها الخاصة. بشكل مشابه يمكن تعريف العلاقة بين الصف والأغراض فالصف هنا هو وصفة الكاتو والأغراض هي قوالب الكاتو، يمكن تعريف الصف بأنه وصف للعناصر المراد إنشاؤها واستخدامها في التطبيق أما الأغراض فهي تلك العناصر

وعندما ننشئها يصبح بالإمكان استخدامها وتغيير بعض الصفات لكل عنصر بعينه ليصبح مختلفاً قليلاً عن الآخرين لكنه يبقى محتفظاً بصفته الأساسية من حيث أنه عنصر من ذلك الصف، من جهة أخرى لا يمكن استخدام الصف بحد ذاته فهو مجرد تعريف. يتألف تعريف الصف من الخصائص والوظائف بشكل أساسي، ولكل لغة برمجة طريقتها في تعريفه واستخدامه.

المثال هو تعبير آخر يطلق على الغرض، فنقول عن غرض ما أنه مثال من الصف الذي يتبع له، ويمكن استخدام تشبيه آخر هنا فالسيارة كوصف مجرد هي ذلك الشيء الذي يمكن الركوب فيه وتشغيله ليمشي على عدة عجلات قد تكون أربع أو ست أو ثمان أو أكثر وينتقل من مكان لآخر بسرعة تختلف حسب كمية الوقود التي تدفع إلى المحرك وللسيارات ألوان مختلفة، هذا وصف للصف المدعو سيارة، أما عندما أتحدث عن سيارتي الخاصة ذات العجلات الأربع واللون الفضي فهي غرض من الصف المدعو سيارة أو هي مثال عن الأغراض التي يمكن أن ننشئها من الصف المدعو سيارة.

◄ التخطيط للعمل وتعريف الصف

في البداية عندما نريد بناء صفوف جديدة لاستخدامها في التطبيق علينا أن نفكر ملياً بخصائص ووظائف وبقية مكونات تلك الصفوف وعلاقاتاها مع بعضها البعض.

تستخدم للتخطيط المسبق لذلك عدة طرق ومنها طريقة لغة UML التي تمكننا من بناء نموذج رسومي إلى حد ما للتطبيق المراد إنجازه، بعد ذلك يمكن تحليل النموذج وإصلاحه وتعديله عدة مرات للوصول إلى الشكل الأمثل والمرضي، في الخطوة التالية يتم بناء الصفوف والأغراض بشكل كامل بالاعتماد على النموذج الذي تم بناؤه.

هناك تقنيات عديدة ونظريات كثيرة حول أساليب التصميم الجيد للتطبيق غرضي التوجه وطريقة استخدام لغة UML من أجل التصميم وتحليل التصميم. في هذا الكتاب لن نركز على لغة UML وهندسة البرمجيات، ولكن سوف نشير إلى كيفية تمثيل الصفوف والأغراض فيها.

يتم تمثيل الصف بمستطيل مقسوم إلى ثلاثة أقسام يكتب في القسم الأول من الأعلى اسم الصف وفي القسم التالي أسماء الخصائص وفي القسم الأسفل أسماء الوظائف، كما في الشكل 41

الشكل 41 تمثيل الصف في UML

◄ الخصائص

الخصائص هي أسماء متحولات تدل على مواصفات الصف المعرف، وفي مثالنا عن قالب الكاتو يكون عدد أكواب الطحين وحجم الماء المستخدم وكمية السكر خصائصاً للصف. كمثال من الواقع البرمجي يمكن تعريف صف لإجراء حسابات على التاريخ والوقت بالاسم `nDate` ويمكن أن تكون لهذا الصف خصائص مثل اليوم والساعة والدقائق والثواني.

◄ الوظائف

الوظائف هي توابع يتم تعريفها في الصف لإنجاز مهام محددة، وفي مثالنا عن الصف `nDate` يمكن تعريف وظيفة لطباعة الوقت الحالي، ووظيفة أخرى لإضافة عدد من الساعات للوقت الحالي.

هناك وظيفة ذات طابع خاص تأخذ اسم الصف ذاته وهي تسمى بالوظيفة المنشئة `constructor` ومهمة هذه الوظيفة إعطاء القيم الأولية للخصائص عند إنشاء غرض من الصف وإعادة ذلك الغرض الذي تم إنشاؤه بالقيم المحددة للخصائص.

هناك أيضاً وظيفة أخرى خاصة تعمل على إزالة المعلومات المختزنة في الذاكرة عن خصائص الغرض الذي تم إنشاؤه من الصف، حيث إن ماتلاب لا يقوم بإزالة تلك المعلومات من الذاكرة من تلقاء نفسه عندما يقوم المستخدم باستخدام اسم المتحول نفسه لحفظ معلومات أخرى، أي أن الغرض ستتم إزالته اسمياً ولكن المعلومات التي تدل عليه تبقى في الذاكرة ولهذا تستخدم الوظيفة الخاصة هذه لإزالة تلك المعلومات عند إزالة الغرض، هذه الوظيفة الخاصة تسمى المدمرة أو الماحية destructor.

◄ بنية الصف في ماتلاب

في ماتلاب يمكن تعريف الصف في أي ملف من نوع m-file (تتم تسميته باسم الصف المراد تعريفه) بواسطة الكلمة المفتاحية classdef متبوعةً باسم الصف ثم تكتب الخصائص ثم الوظائف ثم الكلمة المفتاحية end للدلالة على انتهاء تعريف الصف، لتعريف الخصائص نبدأ بكتابة الكلمة properties يليها على أسطر جديدة الخصائص المطلوبة ثم الكلمة المفتاحية end للدلالة على انتهاء الخصائص، بعد ذلك تكتب الكلمة methods يليها تعريف الوظائف على أسطر جديدة وفي نهايتها كلمة end للدلالة على انتهاء الوظائف. المثال التالي يبين بنية الصف في ماتلاب والتوافق بينه وبين طريقة UML في عرض الصف.

```
classdef Shape
  properties
    vertices=[];
    type='';
  end %end of properties
  methods
     function s=Shape(v,t)
       s.vertices=v;
       s.type=t;
     end
     function per=perimeter()
        %code for calculating the perimeter of the shape
     end
     function ar=area()
        %code for calculating the area of the shape
     end
  end %end of methods
end %end of class
```

في هذا المثال استخدمنا متجهاً أفقياً لتعريف إحداثيات رؤوس شكل هندسي في المستوي كخاصة أولى للصف المعرف للشكل، الخاصة الثانية هي قيمة نصية تدل على نوع الشكل الهندسي (مستطيل، مربع، مثلث، ...).

الوظائف المستخدمة هي توابع لحساب المساحة والمحيط، ولم نكتب الكود اللازم هنا حالياً.

الوظيفة الخاصة التي تأخذ اسم الصف ذاته هي لإنشاء غرض من الصف وفيها يتم إسناد قيم الخصائص للغرض المثال.

بعد إنشاءالصف بهذا الشكل وحفظ الملف في مسار ماتلاب، يمكن استخدامه لإنشاء الأغراض في نافذة أوامر ماتلاب أو في أي مكان آخر شأنه في ذلك شأن أي نوع من أنواع المتحولات، وعلى سبيل المثال لإنشاء غرض من هذا الصف بالنوع `rec` والإحدائيات `(0,0), (100,0), (100,50), (0,50)` يمكن كتابة الكود التالي في نافذة أوامر ماتلاب:

```
>> s=Shape([0 0 100 0 100 50 0 50],'rec')
```

عند الضغط على مفتاح الإدخال يظهر ماتلاب نتيجة التنفيذ كمايلي:

```
s =
Shape
properties:
   vertices: [0 0 100 0 100 50 0 50]
      type: 'rec'
list of methods
```

◄ استخدام الوظائف والخصائص

سوف نضيف هنا الشيفرة اللازمة لحساب محيط الشكل الهندسي في الصف المعرف سابقاً ليصبح الكود الكامل للصف كمايلي:

```
classdef Shape
  properties
    vertices=[];
    type='';
    per=0;
  end %end of properties
  methods
     function s=Shape(v,t)
       s.vertices=v;
       s.type=t;
     end
     function obj=perimeter(obj)
        %code for calculating the perimeter of the shape
        x=obj.vertices;
        y=reshape(x,2,length(x)/2);
        x=y(1,:);y=y(2,:);
        x=[x x(1)];y=[y y(1)];
        xd=diff(x).^2;yd=diff(y).^2;
        dis=sum(sqrt(xd+yd));
        obj.per=dis;
     end
     function ar=area()
        %code for calculating the area of the shape
     end
```

```
  end %end of methods
end %end of class
```

عرفنا خاصة جديدة هي per للدلالة على محيط الشكل وقيمتها الافتراضية هي الصفر. الوظيفة perimeter تقوم بحساب المحيط، يتم تمرير المتحول obj كبارامتر لهذه الوظيفة ويكون هو متحول الخرج لها أيضاً، وبهذه الطريقة يدل المتحول obj على الغرض المثال المستخدم فعندما نكتب obj.vertices نقصد الخاصة vertices للغرض الذي تم تمريره في المتحول obj كبارامتر دخل للوظيفة perimeter وعندما نقول obj.per=dis نقصد أننا نسند قيمة المتحول المحسوب dis إلى الخاصة per للغرض المقصود بالمتحول obj وبهذا يتم تحديث معلومات الغرض.

في نافذة أوامر ماتلاب نقوم بتعريف شكل هندسي جديد وفق الصف Shape كما سبق ثم نقوم باستدعاء الوظيفة perimeter ليتم حساب المحيط وتحديث المعلومات كمايلي:

```
>> s=Shape([0 0 100 0 100 50 0 50],'rec')
s =
Shape
properties:
   vertices: [0 0 100 0 100 50 0 50]
```

```
      type: 'rec'
       per: 0
>> s.perimeter
ans =
Shape
properties:
   vertices: [0 0 100 0 100 50 0 50]
      type: 'rec'
       per: 300
```

يمكن أيضا تضمين استدعاء الوظيفة `perimeter` في الوظيفة المنشئة `Shape` لجعل الصف يحسب محيط الشكل الغرض مباشرة عند إنشائه وفي هذه الحالة نغير خرج الوظيفة `perimeter` ليصبح قيمة المحيط ونسنده في الوظيفة المنشئة إلى الخاصة `per` فيصبح الكود كمايلي:

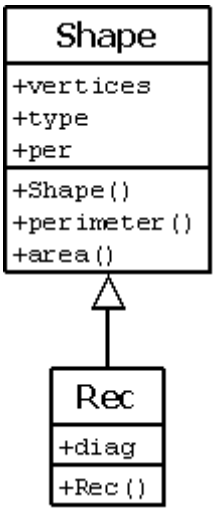
```
classdef Shape
  properties
    vertices=[];
    type='';
    per=0;
  end %end of properties
  methods
     function s=Shape(v,t)
       s.vertices=v;
       s.type=t;
       s.per=s.perimeter();
```

```
    end
    function dis=perimeter(obj)
       %code for calculating the perimeter of the shape
        x=obj.vertices;
        y=reshape(x,2,length(x)/2);
        x=y(1,:);y=y(2,:);
        x=[x x(1)];y=[y y(1)];
        xd=diff(x).^2;yd=diff(y).^2;
        dis=sum(sqrt(xd+yd));
        %obj.per=dis;
    end
    function ar=area()
       %code for calculating the area of the shape
    end
  end %end of methods
end %end of class
```

◄ الوراثة

في كثير من الحالات يمكن تعريف صفوف متشابهة في الخصائص والوظائف وبعض هذه التشابهات تصل إلى حد ان أحد الصفوف هو حالة خاصة من الآخر، وعلى سبيل المثال العدد الصحيح هو حالة خاصة من العدد الحقيقي، والمستطيل هو حالة خاصة من الشكل الهندسي الرباعي.

في مثل هذه الحالات يمكن الاستفادة من وجود صف معرف مسبقاً لتعريف صفوف جديدة هي في حقيقتها حالات خاصة من الصف المعرف أصلاً. يمكن تمثيل ذلك في لغة UML كما هو مبين في الشكل 42



الشكل 42التعبير عن الوراثة في UML

في هذا التمثيل نرى الصف Shape بخصائصه ووظائفه التي رأيناها مسبقاً، ونرى الصف الجديد Rec الذي يمثل حالة خاصة من الصف Shape، لتمثيل العلاقة بين الصفين يستخدم السهم الذي ينطلق من الصف الابن إلى الصف الأب ونقول إن الصف الابن يرث كل خصائص ووظائف الصف الأب بالإضافة إلى أنه يملك خصائصاً ووظائفا خاصة به وهي هنا الخاصة diag والوظيفة المنشئة Rec

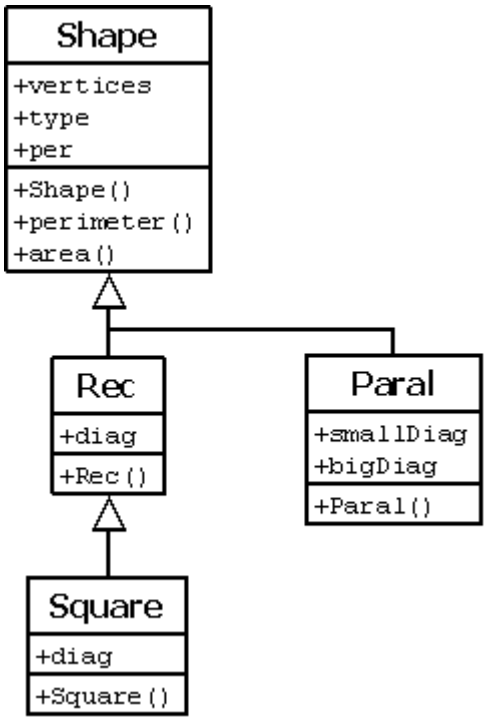
لإنشاء صف ابن من صف أب في ماتلاب تستخدم نفس طريقة تعريف الصف السابقة مع إضافة اسم الصف الأب إلى سطر التعريف بعد الإشارة < كما في في كود التعريف التالي للصف Rec

```
classdef Rec < Shape
  properties
    diag;
  end
  methods
    function r=Rec(v,t)
      r=r@Shape(v,t);
    end
  end
end
```

الوظيفة المنشئة هنا للصف Rec تعمل بطريقة مماثلة للوظيفة المنشئة للصف الأب Shape ولذلك نستخدم الوظيفة المنشئة للصف الأب لإجراء اللازم، وللوصول إليها نستخدم الرمز @ حيث نضع قبله اسم المتحول المراد الإسناد إليه وبعده اسم الوظيفة التي تنتمي إلى الصف الأب.

يمكن أن ننشئ أي عدد من الصفوف الأبناء للصف الأب الواحد، ويمكن أن ننشئ صف ابن لصف ابن بحيث يصبح الابن القديم أباً وكمثال يمكن إنشاء الصف

Square الذي يعرف مربعا في المستوي كصف ابن للصف Rec الذي هو بدوره صف ابن للصف Shape، الشكل 43 يبين تمثيل ذلك في لغة UML



الشكل 43 أبناء متعددون

يسمح ماتلاب أيضا خلافا للكثير من لغات البرمجة الأخرى بالوراثة المتعددة وهي أن يكون للصف الابن أكثر من صف أب فيرث الخصائص والوظائف منها جميعاً ولكن يجب الانتباه هنا في حال تشابه أسماء الخصائص أو الوظائف. ولا ينصح عادة باستخدام الوراثة المتعددة إلا في حالات الضرورة.

لتعريف الوراثة المتعددة في ماتلاب تذكر أسماء الصفوف الآباء جميعاً كمايلي:

classdef classname < super1 & super2 & super3

◄ الوصول المقيد إلى الخصائص والوظائف

عند تعريف الخصائص والوظائف فإنها تكون عامة `public` في الحالة الافتراضية، ويقصد بذلك أن الوصول إليها من خارج الصف غير مقيد فيمكن قراءة خاصة أو تشغيل وظيفة من أي مكان خارج الصف، هناك حالات ينبغي فيها تقييد الوصول إلى الخصائص أو الوظائف لتصبح متاحة من قبل الصف فقط فلا يمكن الوصول إليها من الخارج وهذه تسمى خصائص ووظائف خاصة `private` بالصف الذي تم تعريفها فيه. هناك أيضاً حالات نحتاج فيها إلى تقييد الوصول ولكن بحيث يمكن لكل الصفوف الأبناء الوصول إلى خصائص الأب ووظائفه، في هذه الحالة نقول عن الخصائص والوظائف أنها محمية `protected`

يمكن تعريف عدة خصائص عامة وأخرى خاصة وأخرى محمية في الصف الواحد وكذلك الوظائف.

في ماتلاب يمكن تقييد الوصول إلى مجموعة من الخصائص بكتابة `(Access='private')` بعد الكلمة المفتاحية `properties` ثم كتابة الخصائص

المراد جعلها خاصة. بشكل مشابه يتم تجميع الخصائص المحمية والوظائف الخاصة والوظائف المحمية، لتكون البنية الهيكلية العامة للصف في ماتلاب كمايلي:

```
classdef classname
   properties

   end
   properties (Access='private')

   end
   properties (Access='protected')

   end
   methods

   end
   methods (Access='private')

   end
   methods (Access='protected')

   end
end
```

◄ الوظائف التي تعمل من الصف مباشرة

عند تعريف وظيفة في صف وإنشاء غرض من ذلك الصف يمكن استخدام تلك الوظيفة من الغرض باستخدام اسم الغرض متبوعاً بعلامة النقطة ثم اسم الوظيفة، هذه الوظائف هي الوظائف الغرضية التي يجب أن يتم استدعاؤها من الأغراض وليس لها معنى بدون أغراض.

هناك وظائف أخرى يمكن استخدامها من الصف مباشرة دون وجود أغراض أو حتى لو وجدت الأغراض فالوظيفة غير معنية بها ولكن بالصف نفسه، هذه الوظائف تسمى بالوظائف الصفية `static methods` ويمكن تعريف وظائف كهذه باستخدام العبارة `(static=true)` بعد الكلمة المفتاحية `methods` لمجموعة الوظائف المراد تعريفها بهذا النوع. لاستخدام وظيفة من هذا النوع يذكر اسم الصف متبوعاً بعلامة النقطة ثم اسم الوظيفة.

◄ متى نستخدم البرمجة غرضية التوجه

عندما تكون المهمة التي نريد إنجازها برمجياً صغيرة وبسيطة في بنيتها وإمكانية تحليلها يمكن استخدام تابع واحد لكتابة البرنامج اللازم لإنجاز المهمة، أما عندما تصبح المهمة على قدر من التعقيد بحيث إن تابعاً واحداً سيصبح من الصعب تنظيمه وإصلاحه فيمكن تجزيء المهمة وكتابة البرنامج اللازم لها في عدة توابع بحيث يتم

التنفيذ بشكل سلسلة يمرر فيها كل تابع نتيجة تنفيذه إلى التابع الذي يليه وهكذا للوصول إلى النتيجة النهائية. عندما يكون لدينا تطبيق كبير نريد إنجازه ويحوي عدداً كبيراً من المهمات التي تتشارك في بعض التوابع ولا تتشارك في أخرى ويصبح تنظيم كتابة تلك التوابع وتمرير البارامترات بينها أمراً معقداً، عند ذلك ينبغي التفكير بكتابة الصفوف والأغراض لتنظيم العمل وتسهيله.

تتميز البرمجة غرضية التوجه بسهولة الصيانة لاحقاً وسهولة التصميم المسبق باستخدام بعض التقنيات مثل لغة UML قبل البدء بكتابة الكود، وكذلك تتميز بأنها تعطينا قدرة عالية في إعادة الاستخدام للصفوف المعرفة.

خلاصة القول أنه علينا التفكير في البرمجة غرضية التوجه عندما يكون لدينا تطبيقات كبيرة نسبياً وعندما يكون لدينا خطة لإعادة استخدام الكثير من التوابع الصغيرة التي تنجز مهمات بسيطة حيث نجمعها معاً في صفوف معرفة لمرة واحدة وقابلة للاستخدام فيما بعد.

كثيرات الحدود

◄ كثير الحدود وجذوره

يعرف كثير الحدود الجبري ذي المتحول الوحيد بالعلاقة العامة التالية:

$$f(x) = a_n.x^n + a_{n-1}.x^{n-1} + \cdots + a_1$$

يسمى هذ كثير حدود في x من المرتبة n وتسمى المعاملات العددية $a_n \ldots a_1$ ثوابت كثير الحدود.

يمكن التعبير عن كثير الحدود الجبري في ماتلاب باستخدام متجه أفقي يحوي ثوابت كثير الحدود مرتبة حسب القوى المتناقصة للمتحول، أي أن كثير الحدود المعطى بالعلاقة:

$$f(x) = 5.x^5 + 7.x^2 - 11.x^2 + 9$$

يعبر عنه بالمتجه الأفقي:

```
>>f=[5 0 0 7 -11 0 9]
```

يمكن الحصول على جذور كثير الحدود الذي عرفت ثوابته بواسطة التابع roots كمايلي:

```
>>roots([1 -11 30])
```

يعيد هذا التابع متجهاً عمودياً فيه جذور كثير الحدود وهي هنا 6 و 5

بشكل عكسي يمكن الحصول على ثوابت كثير حدود عرفت جذوره بواسطة التابع poly كمايلي:

```
>>poly([5 6])
```

هنا نحصل على ذات كثير الحدود السابق.

◄ ضرب كثيرات الحدود

عند إجراء ضرب كثيرات الحدود نحصل على كثير حدود جديد مرتبته هي جداء مرتبتي كثيري الحدود المضروبين، يوفر ماتلاب تابعاً لإجراء ضرب كثير حدود هو التابع conv ويستخدم بإدخال كثيري الحدود المراد ضربهما كمتجهين أفقيين يحويان ثوابت كثيري الحدود، كما في المثال التالي:

```
>>conv([1 5 6],[1 0 4])
```

نتيجة التنفيذ هي متجه أفقي يحوي أمثال كثير الحدود الناتج.

◄ تقسيم كثيرات الحدود

في عملية القسمة نحصل على ناتج وباقي قسمة، طبعاً يمكن أن يكون باقي القسمة صفراً أو عدداً أو كثير حدود، يستخدم التابع `deconv` لإجراء القسمة والحصول على الناتج والباقي كمايلي:

```
>> [q r]=deconv([1 -11 34 -44 120],[1 -11 30])
q =
   1   0   4
r =
   0   0   0   0   0
```

نلاحظ أن باقي القسمة هو متجه يساوي في الطول متجه كثير الحدود المقسوم.

◄ التعبير عن كسر حدودي

إذا كان لدينا كسر بسطه ومقامه كثيرا حدود، يمكن التعبير عنه بطريقتين مختلفتين، ففي الطريقة الأولى يكتب البسط والمقام بشكل كثيري حدود عاديين وفي الطريقة

الثانية يكتب الكسر بشكل مجموع عدة كسور كل منها مقامه هو ثنائي حد، وقد يكون هناك حد أخير مقامه عدد ثابت.

ليكن لدينا الكسر الحدودي التالي:

$$\frac{\mathrm{b(s)}}{\mathrm{a(s)}} = \frac{b_1.s^m + b_2.s^{m-1} + \ \ldots + b_{m+1}}{a_1.s^n + a_2.s^{n-1} + \ \ldots + a_{n+1}}$$

بإجراء عملية تفريق الكسور يمكن الحصول على الشكل:

$$\frac{\mathrm{b(s)}}{\mathrm{a(s)}} = \frac{r_1}{s-p_1} + \frac{r_2}{s-p_2} + \ \ldots + \frac{r_n}{s-p_n} + k(s)$$

تدعى الأعداد $p_1, p_2, \ldots, p_n$ بالأقطاب ويمكن أن تتكرر قيمة القطب فنحصل على ثنائي حد مرفوع للتربيع، ويدعى `k(s)` بالحد الثابت، والأعداد $r_1, r_2, \ldots$ r_n بالبواقي.

يمكن التعبير عن الكسر الحدودي السابق في ماتلاب بمتجهين يحويان معاملات البسط والمقام للتعبير عنه بالشكل الأول، أو بثلاثة متجهات تحوي البواقي والأقطاب ومعاملات الحد الثابت للتعبير عنه بالشكل الثاني.

إذا كان لدينا الشكل الأول يمكن الحصول على الشكل الثاني باستخدام التابع `residue` كمايلي:

```
[r,p,k]=residue(b,a)
```

أما إذا كان لدينا الشكل الثاني فيمكن الحصول على الشكل الأول باستخدام نفس التابع مع تبديل المدخلات والمخرجات كمايلي:

```
[b,a]=residue(r,p,k)
```

يمكن لماتلاب أن يفهم المطلوب بواسطة اختلاف عدد المدخلات وعدد المخرجات.

مثال

ليكن لدينا الكسر:

$$f(x) = \frac{b(x)}{a(x)} = \frac{1}{x^2 - 1}$$

يمكن إجراء تفريق الكسور للحصول على:

$$f(x) = \frac{-0.5}{x+1} + \frac{0.5}{x-1}$$

للتعبير عن هذا العمل في ماتلاب نكتب التعليمات التالية:

```
>> b=[1];
>> a=[1 0 -1];
>> [r,p,k]=residue(b,a)
```

في السطر الأول أدخلنا البسط بشكل متجه أفقي، وفي السطر الثاني أدخلنا المقام بشكل متجه أفقي أيضاً، في السطر الثالث استخدمنا التابع residue للحصول على البواقي والأقطاب والحد الثابت (لا يوجد حد ثابت في مثالنا) فنحصل على النتائج كمايلي:

```
r =
  -0.5000
   0.5000
p =
   -1
    1
k =
    []
```

باستخدام التابع residue بشكله الآخر نحصل ثانية على المتجهين b و a مايلي:

```
>> [b,a]=residue(r,p,k)
b =
    0    1
a =
    1    0   -1
```

◄ إيجاد مشتقات كثير حدود

يوفر ماتلاب تابعاً لحساب مشتقات كثير حدود بثلاثة أشكال هامة، الشكل الأول لحساب مشتق كثير حدود، والشكل الثاني لحساب مشتق جداء كثيري حدود، والشكل الثالث لحساب مشتق كسر حدودي كل من بسطه ومقامه كثيرا حدود، في الشكلين الأول والثاني يعيد التابع `polyder` متجهاً أفقيا يعبر عن المعاملات العددية لكثير الحدود الناتج، أما في الشكل الثالث فيعد متجهين أفقيين أحدهما يعبر عن بسط الناتج والآخر عن مقامه.

ليكن لدينا كثيرا الحدود التاليين:

$$f1(x) = 3.x^3 + 4.x^2 - 9.x + 16$$

$$f2(x) = 9.x^2 + x - 5$$

يمكن تعريف هذين التابعين في ماتلاب كمايلي (سبق إيضاح ذلك):

```
>>f1=[3, 4, -9, 16];
>>f2=[9, 1, -5];
```

للحصول على المشتق الأول لكثير الحدود الأول يستخدم التابع `polyder` كمايلي:

```
>>f1_1=polyder(f1);
```

للحصول على المشتق الثاني والثالث يستخدم التابع بطريقة تضمينية بعدد يناسب رتبة المشتق كمايلي:

```
>>f1_2=polyder(polyder(f1));
>>f1_3=polyder(polyder(polyder(f1)));
```

للحصول على المشتق الأول لجداء كثيري الحدود `f1, f2` يمكن إدخال كثيري الحدود كبارامترين في التابع `polyder` كمايلي:

```
>>fd=polyder(f1,f2);
```

هذا يكافئ من حيث النتيجة إدخال الأمر التالي:

```
>> fd=conv(f1,polyder(f2))+conv(polyder(f1),f2)
```

لأن:

$$(f1 \times f2)' = f1 \times f2' + f1' \times f2$$

ويتضح أن الطريقة الأولى أسهل.

في الشكل الثالث لاستخدام التابع `polyder` نوجد مشتق الكسر الحدودي الناتج عن جعل كثير الحدود الأول بسطاً وكثير الحدود الثاني مقاماً، كمايلي:

>>[q, d]=polyder(f1,f2);

حيث يعبر المتجه الناتج q عن بسط المشتق الناتج ويعبر المتجه الناتج d عن مقام المشتق الناتج.

وهذا يكافئ من حيث النتيجة:

>>[q, d]=[conv(polyder(f1),f2)-conv(f1,polyde(f2)) , conv(f2,f2)]

لأن:

$$(\frac{f1}{f2})' = \frac{f1'.f2 - f1.f2'}{f2^2}$$

المعالجة الرمزية

يوفر ماتلاب إمكانية المعالجة الرمزية للتعابير الرياضية بدلاً من معالجتها لأعداد محددة، والمعالجة الرمزية هي أن يقوم ماتلاب بحساب النتائج لتعابير رياضية دون التقيد بأرقام وأعداد، ومن أمثلة ذلك إيجاد التكاملات غير المحددة والمشتقات والنهايات.

تشمل أدوات المعالجة الرمزية في ماتلاب التحليل الرياضي والجبر الخطي والمعادلات الجبرية والتفاضلية والتحويلات العددية كتحويل لابلاس وتحويل فوريه.

◄ التعريف عن المتحولات الرمزية

يتم التعريف عن المتحولات المراد معاملتها معاملة رمزية باستخدام أحد التابعين sym و syms وفي حقيقة الأمر فإن ماتلاب يقوم بإنشاء عنصر من النوع sym (اختصارا لكلمة symbol) للتعامل معه لاحقاً كرمز.

يمكن استخدام التابع sym لتعريف رموز مفردة وإسنادها إلى أسماء كمايلي:

```
>>a=sym('a')
```

في حال وجود عدة متحولات يراد تعريفها يمكن استخدام التابع sym عدة مرات، مرة لكل متحول كمايلي:

```
>>a=sym('a');
>>b=sym('b');
>>x=sym('x');
```

وهكذا لكل متحول. يمكن أيضا اختصار العملية باستخدام التابع syms كمايلي:

```
>>syms a b x
```

◄ تعريف التعابير الرمزية وحسابها

يتم تعريف التعابير الرمزية بواسطة التابع sym بأسلوب مشابه لتعريف المتحولات ولكن يجب أن تكون المتحولات الداخلة في تركيب التعبير المعرف رمزياً قد تم تعريفها سابقا كمتحولات رمزية، كما في المثال التالي:

```
>>syms a b c x
>>f=sym('a*x^2+b*x+c');
```

في هذا المثال تم تعريف المتحولات الرمزية a و b و c و x ثم تم تعريف التعبير f على أنه كثير حدود من الدرجة الثانية في x ثوابته العددية متغيرات هي a و b و c

هنا تعامل الثوابت على أنها متحولات ويمكن حساب كثير الحدود الناتج من استبدال قيم المتحولات تلك للحصول على كثير حدود في x بعينه، ثم حساب قيم محددة لكثير الحدود ذاك بإعطاء قيم للمتحول x

يستخدم التابع subs لحساب التعبير المعرف رمزياً من أجل قيم محددة للمتحولات أو بعض المتحولات الداخلة في تركيبه، كما في الأمثلة التالية:

```
>> syms a b c x
>> f=sym('a*x^2+b*x+c');
>> f1=subs(f,a,1)
f1 =
x^2+b*x+c
>> f2=subs(f,[a b c],[1 4 5])
f2 =
x^2+4*x+5
>> y1=subs(f2,x,1)
y1 =
   10
>> y2=subs(f2,x,2)
y2 =
   17
```

في البداية تم هنا تعريف كثير الحدود f بدلالة المتحولات الرمزية a, b, c, x ثم تم حساب قيمته باستبدال المتحول الرمزي a بالقيمة 1 وأسندت النتيجة إلى المتحول f1 الذي أصبح بدوره كثير حدود معرف رمزياً بالمتحولات b, c, x، بعد ذلك تم استبدال قيم المتحولات الرمزية a, b, c بالقيم 1, 4, 5 وأسندت النتيجة إلى المتحول f2 الذي أصبح بدوره أيضا كثير حدود رمزياً في المتحول x فقط. أخيراً أسندت القيم الناتجة عن استبدال المتحول الرمزي x في التعبير الرمزي f2 إلى

المتحولين y1, y2 وبما أن f2 لا يحوي متغيرات أخرى حصلنا على قيم عددية فالمتحولات y1, y2 أصبحت من النوع العددي الافتراضي في ماتلاب وهو النوع double

◄ حساب المشتقات

يستخدم التابع diff لحساب مشتقات التوابع المعرفة رمزياً وإعداة النتائج كتوابع معرفة رمزياً أيضاً، يمكن حساب المشتقات من أي رتبة بإدخال رتبة المشتق في التابع diff كبارامتر ثاني علماً أن الرتبة الافتراضية هي الواحد.

في المثال التالي نعرف تابعاً صحيحاً من الدرجة الثالثة ونوجد مشتقاته الثلاثة كمايلي:

```
>> syms x
>> f=sym('4*x^3+2*x^2-9*x+16')
f =
4*x^3+2*x^2-9*x+16
>> f1=diff(f)
f1 =
12*x^2+4*x-9
>> f2=diff(f,2)
f2 =
24*x+4
>> f3=diff(f,3)
```

```
f3 =
24
```

يمكن استخدام التابع `diff` بطريقة تضمينية كمايلي:

```
>> syms x
>> f=sym('4*x^3+2*x^2-9*x+16')
f =
4*x^3+2*x^2-9*x+16
>> f3=diff(diff(diff(f)))
f3 =
24
```

يمكن الاستفادة من هذه الأدوات لحساب مشتقات توابع معقدة قليلاً كما في المثال التالي:

```
>> syms x
>> f=sym('(3*x^2+1)*cos(2*x^3-5)/(sin(8*x+2))')
f =
(3*x^2+1)*cos(2*x^3-5)/(sin(8*x+2))
>> diff(f)
ans =
6*x*cos(2*x^3-5)/sin(8*x+2)-6*(3*x^2+1)*sin(2*x^3-5)*x^2/sin(8*x+2)-
8*(3*x^2+1)*cos(2*x^3-5)/sin(8*x+2)^2*cos(8*x+2)
```

◄ المشتقات الجزئية

عند اشتقاق الدوال التابعة لمتحولين أو أكثر نقوم بالاشتقاق الجزئي لمتحول واحد ومعاملة بقية المتحولات على أنها ثوابت، على سبيل المثال ليكن لدينا التابع التالي:

$$f(x,t) = \sin(3 \cdot \mathrm{x}^2 + 2 \cdot \mathrm{x} \cdot \mathrm{t})$$

يمكن إيجاد المشتق الجزئي له بالنسبة لمتحول الزمن `t` كمايلي:

$$\frac{\partial f(x,t)}{\partial t} = 2 \cdot x \cdot \cos(3 \cdot x^2 + 2 \cdot x \cdot t)$$

في ماتلاب يمكن إيجاد هذا المشتق والمشتق الآخر بالنسبة للمتحول `x` باستخدام تابع `diff` مع تمرير اسم المتحول المراد الاشتقاق بالنسبة له كبارامتر ثاني، كما في المثال التالي:

```
>> syms x t
>> f=sym('sin(3*x^2+2*x*t)')
f =
sin(3*x^2+2*x*t)
>> df_dt=diff(f,t)
df_dt =
2*cos(3*x^2+2*x*t)*x
>> df_dx=diff(f,x)
df_dx =
cos(3*x^2+2*x*t)*(6*x+2*t)
```

◄ التكاملات

حساب التكامل من الأشياء التي يمكن معالجتها في ماتلاب أيضاً بواسطة المعالجة الرمزية، ويستخدم لذلك التابع `Int` (اختصاراً لكلمة `Integral`) سواء لحساب التكامل غير المحدد أو للتكامل المحدد. في حالة التكامل غير المحدد يتم تمرير التابع المراد مكاملته كبارامتر وحيد ونحصل على التابع الأصلي له رمزياً، أما في حالة التكامل المحدد فيتم تمرير مجال التكامل كبارامترين آخرين القيمة الدنيا ثم القيمة العليا ونحصل طبعاً على عدد.

مثال

```
>> syms x
>> f=sym('x*sin(x)')
f =
x*sin(x)
>> int(f)
ans =
sin(x)-x*cos(x)
>> int(f,0,pi)
ans =
pi
```

في هذا المثال قمنا بتعريف الدالة f ثم استخدمنا التابع int لإيجاد الدالة الأصلية لها، نلاحظ أن ماتلاب يهمل ذكر ثابت التكامل حيث يمكن إضافة أي عدد ليكون ثابت تكامل، ثم استخدمنا التابع int لإيجاد قيمة التكامل المحدد للدالة f في المجال [0, pi]

◄ النهايات

يتم حساب نهايات الدوال في التحليل الرياضي من أجل إيجاد مقاربات الخطوط البيانية وحساب المشتقات، وتحسب هذه النهايات عندما تقترب قيمة المتحول من قيمة ما في جوار محذوف لقيم المتغير.

في ماتلاب يستخدم التابع limit لحساب النهايات حيث يتم تمرير الدالة المراد حساب نهايتها كبارامتر أول ثم القيمة التي يراد حساب النهاية عندها كبارامتر ثاني، في حالة عدم تمرير تلك القيمة الأخيرة تستخدم القيمة الافتراضية وهي الصفر.

أمثلة:

في الأمثلة التالية نستخدم التابع limit لحساب نهايات بعض الدول بشكله البسيط كما ذكرنا.

```
>> syms x
>> limit(x^7+x^2)
```

```
ans =
0
>> limit(x^7+x^2,inf)
ans =
Inf
>> limit(sin(x)/x,inf)
ans =
0
>> limit(sin(x)/x)
ans =
1
```

في حالة وجود دالة تتبع متحولين يجب إدخال المتحول المراد حساب النهاية عنده ويعامل المتحول الثاني كأنه عدد وبالنتيجة نصل على دالة بالمتحول الثاني بدلاً من قيمة عددية، كمايلي:

```
>> syms x t
>> limit(2*x+3*t,t,0)
ans =
2*x
>> limit(2*x+3*t,x,0)
ans =
3*t
```

هناك حالات تكون فيها النهاية من اليسار مختلفة عن النهاية من اليمين ولهذا ينبغي استخدام البارامتر الرابع للدلالة على اتجاه حساب النهاية وقيمته هي 'left' أو 'right' كمايلي:

```
>> syms x
>> limit(1/x,x,0,'left')
ans =
-Inf
>> limit(1/x,x,0,'right')
ans =
Inf
```

◄ مجاميع السلاسل

يستخدم لهذا الحساب التابع symsum وهو يحسب مجموع السلسلة المدخلة كبارامتر أول بين القيمتين التاليتين كبارامتريين ثاني وثالث، كما في المثال التالي:

```
>> syms x
>> symsum(1/x,1,10)
ans =
7381/2520
```

◄ منشور تايلور

يستخدم منشور تايلور في الرياضيات لإيجاد تقريب لحساب دالة ما مثل e^x أو `sin(x)` أو غيرها بحيث يتم حساب تلك الدالة بواسطة جمع عدد من الحدود للحصول على نتيجة بدقة مقبولة. يستخدم التابع `taylor` لإيجاد السلسلة الممكنة لمنشور تايلور لدالة ما حيث يتم إدخال تلك الدالة كبارامتر أول وعدد الحدود المطلوب كبارامتر ثاني علماً أن القيمة الافتراضية لعدد الحدود هي خمسة حدود.

في المثال التالي نستخدم التابع `taylor` للحصول على سلسلة منشور تايلور للدالة `sin(x)` ثم نستخدم التابع `subs` لحساب قيمة `sin(5)`

```
>> syms x
>> taylor(sin(x))
ans =
x-1/6*x^3+1/120*x^5
>> subs(ans,5)
ans =
  10.2083
>> taylor(sin(x),50);
>> subs(ans,5)
ans =
  -0.9589
```

طبعاً نحصل على دقة أكبر في الحساب كلما زاد عدد حدود منشور تايلور.

◄ النشر والتبسيط

يقصد بالنشر إعادة كتابة الدالة بشكل آخر أكثر تفصيلاً كأن تعاد كتابة دالة مكتوبة كجداء قوسين يحوي كل منهما كثير حدود بشكل كثير حدود جديد ناتج عن ضرب كثيري الحدود، أما التبسيط فهو إعادة كتابة الدالة بشكل أبسط بعد اختصار الحدود الممكن اختصارها أو استبدال بعض الأجزاء بمكافئات أبسط.

يستخدم للنشر التابع `expand` ويستخدم للتبسيط التابع `simplify` كما في الأمثلة التالية:

أمثلة على النشر:

```
>> syms x
>> f=sym('(x+4)^2')
f =
(x+4)^2
>> expand(f)
ans =
x^2+8*x+16
```

أمثلة على التبسيط:

```
>> syms x
```

```
>> f=sym('(x+4)^2+x^2-10')
f =
(x+4)^2+x^2-10
>> simplify(f)
ans =
2*x^2+8*x+6
>> g=sym('sin(x)^2+cos(x)^2+x+10')
g =
sin(x)^2+cos(x)^2+x+10
>> simplify(g)
ans =
x+11
```

قرناء ماتلاب وبدائله

◄ **مشاكل ماتلاب**

عندما نفكر في البدائل يكون لدينا أسباب لهذا التفكير فمن يجد ماتلاب جيداً ومفيداً لا يفكر بتركه واستبداله دون سبب.

أهم الأسباب التي تجعل أحدنا يفكر بالتحول عن ماتلاب هي:

ماتلاب منتج تجاري مئة في المئة

كون ماتلاب منتجا تجاريا يعني الكثير بالنسبة لمستخدميه، ففي البدء هذا يعني أن تركيز الشركة المنتجة له سيكون على جمع المال من وراء تطوير المنتج وهدف التحسين ليس غائباً بالطبع لكنه ليس الهدف الرئيسي، هنا يوجد تناقض وهمي من حيث أن تحسين المنتج من شأنه أن يجلب المزيد من المال للشركة المنتجة وهكذا يبدو أن زيادة الربح توازي تحسين المنتج، في الواقع هذا ليس صحيحا إلا بشكل نسبي، فعندما يكون الهدف هو الربح بالدرجة الأولى، يكون التحسين المطلوب مقيداً بالربح وبالتالي مقيداً برغبة الزبائن الأكثر أهمية وهم هنا الزبائن الأكثر دفعاً للمال، ويكون مقيداً أيضاً بالزمن فهناك مدة زمنية يجب إنتاج نسخة جديدة من التطبيق خلالها لجمع المزيد من الأرباح وهو يعني إضافة ميزات جديدة أو تطوير ميزات موجودة، هذا يدفعنا للتساؤل إن كانت تلك الميزات التي تم تطويرها أو تم إيجادها مفيدة فعلاً أم لا وإلى أي حد سيكون من المفيد للزبون أن يدفع مقابل تلك الميزات.

في حالة التطبيقات غير الربحية يكون الهدف هو إيجاد تطبيق متميز وبما ان الربح غير مستهدف فالزمن غير مهم ويأخذ المبرمجون المتطوعون أو الذين يدفع لهم من قبل المتبرعين بدعم المنتج قصارى جهدهم لجعل المنتج متميزا فعلاً ضمن مدة مريحة في العمل.

من جهة أخرى يكون المنتج الربحي ذا ثمن مرتفع وفي واقع الأمر فإن ماتلاب باهظ الثمن جداً حتى بالنسبة للطلاب والمؤسسات التعليمية، يترتب على هذا أن الشركات الإنتاجية عليها أن تدرس الجدوى الاقتصادية من شراء ماتلاب وفي بعض الأوقات قد لا تكون الجدوى الاقتصادية ذات مؤشر إيجابي. أما المؤسسات التعليمية فقد تجد تعليم ماتلاب امراً مكلفاً جداً.

مصدر الشيفرة مغلق

في البرمجيات مغلقة المصدر لا يمكن الاطلاع على الطريقة التي تم بها إنجاز العمل وهي ما يدعى الخوارزمية، في بعض الأمور البسيطة مثل خوارزمية إيجاد القاسم المشترك الأكبر لعددين لا يكون هذا مهماً فالنتائج تبدو صحيحة من تجارب بسيطة، أما في الخوارزميات المعقدة والتي نجدها في بعض الأدوات الخاصة مثل أدوات المحاكاة وأدوات الشبكات العصبونية فكل ما نستطيع فعله هو أن نثق بالخوارزمية التي يتبعها ماتلاب دون أن نراها فهو مغلق المصدر. أما في البرمجيات مفتوحة

المصدر فيمكن الاطلاع على الخوارزميات المتبعة للتأكد من كونها توافق النظريات والطرق التي نريد تطبيقها.

من جهة أخرى فإن تطوير التطبيق من قبل مبرمج من خارج الشركة المنتجة غير ممكن وهذا يعني أن من يشتري التطبيق مغلق المصدر عليه أن يستخدمه كما هو وفي حال الحاجة للتطوير عليه الطلب من الشركة الصانعة التي لن تدعمه في معظم الحالات إلا في إصدارات لاحقة للتطبيق حيث عليه أن يدفع ثمن التطويرات التي يحتاجها والتي لا يحتاجها في نسخة الإصدار الجديد. في البرمجيات مفتوحة المصدر يمكن تطوير تطبيقات تتخاطب مع التطبيق الأصلي أو تندمج فيه لتصبح وكأنها ميزات إضافية تعمل في بيئته، ويمكن أيضا تطوير التطبيق نفسه ليتلاءم مع حاجات ومتطلبات الزبون، وهذه ميزة هامة فالزبون لم يعد عليه أن ينتظر إصداراً جديداً من الشركة الصانعة لتلبية احتياجاته.

◄ متى نستبدل ماتلاب ومتى نبقي عليه

هناك الكثير من البدائل التي يمكن استخدامها بدلاً من ماتلاب عند ضرورة الاستبدال، وتتنوع تلك البدائل فبعضها لا يحوي كافة الميزات المتوفرة في ماتلاب لكنه يتغلب عليه في مجال محدد، وبعضها لا يحل كل مشاكل ماتلاب لكنه يحل

مشكلة ما بشكل جيد، وبعضها يمكن أن نستخدمه كبديل لميزة في ماتلاب لا كبديل كلي.

في الخلاصة يجب أن نفكر باستبدال ماتلاب في الحالات التالية:

عندما يكون من غير المجدي اقتصاديا أن ندفع ثمن ماتلاب أو الميزات التي نحتاجها منه.

عندما يكون من المخطط له أن نقوم بتطوير التطبيق البديل ليوافق حاجاتنا التي لا يلبيها ماتلاب بالشكل المرغوب.

عندما يكون العمل الذي نريد القيام به دقيقاً كفايةً بحيث نريد أن نعرف بالضبط ما الذي نفعله بواسطة التطبيق، بمعنى أننا نريد الاطلاع على الخوارزميات المتبعة في إنجاز الأعمال.

أما الحالات التي يجب أن نبقي فيها على استخدام ماتلاب فهي:

عندما يكون لدينا فريق عمل مؤهل تأهيلاً جيداً في ماتلاب وليس لدينا الوقت الكافي لإعادة التأهيل في البدائل، رغم أن ذلك ليس صعباً إلا أن بعض المبرمجين لديهم أسباب ليجدوا الأمر مزعجاً وقد لا يكون هناك الوقت الكافي لتجربة البديل والتدريب عليه.

عندما يكون لدينا شركاء في العمل يرفضون استبدال ماتلاب لأنهم سبق أن دفعوا مبالغ باهظة مقابل الحصول عليه وتأهيل فريقهم للعمل فيه تأهيلاً عالي المستوى.

عندما تكون الميزات التي نحتاجها بكثرة في مشروعنا هي تلك الميزات التي يتفوق فيها ماتلاب، أو لا توجد إلا فيه، وعلى سبيل المثال فإن أداة المحاكاة sumulink لا نجدها في أي بديل وهي في واقعها ليست ميزة بل تطبيق مدمج في ماتلاب.

◄ **البدائل الأهم**

Octave

أوكتاف هي لغة برمجة عالية المستوى مكتوبة بلغة الـ C++، وهي تشبه ماتلاب إلى حد كبير من حيث الواجهات ونوافذ الاستخدام ومن حيث الطرق المتبعة في كتابة الشيفرات البرمجية، وتستخدم لذات الأغراض. أهم أوجه الشبه بين أوكتاف وماتلاب هي التعامل مع البيانات كمصفوفات ووجود العديد من المكتبات الهامة المبنية داخل النظام وإمكانية بناء تطبيقات أو معالجة البيانات بدون برمجة.

يمكن استخدام أكواد ماتلاب في غالب الأحيان في بيئة أوكتاف ولكن على العكس هناك الكثير من أكواد أوكتاف لا يمكن استخدامها في بيئة ماتلاب.

من الهام أن أوكتاف هي تطبيق مجاني ومفتوح المصدر، ولذلك فهي واسعة الاستخدام في المؤسسات التعليمية والصناعية والمؤسسات التي تفشل في إيجاد جدوى اقتصادية من استخدام ماتلاب.

يمكن العمل بماتلاب وأوكتاف في فريق واحد ولكن من الهام الانتباه إلى استخدام الطرق النحوية المشتركة في كتابة الشيفرات البرمجية المراد تشغيلها في غير بيئتها الأصلية، ومن الطرق الجيدة استخدام أوكتاف في الوضعية `traditional` حيث لا تقبل بيئة تطبيق أوكتاف في هذه الحالة إلا الأكواد المكتوبة بطرق يمكن تشغيلها في بيئة ماتلاب. على سبيل المثال يمكن كتابة النصوص في بيئة أوكتاف باستخدام إشارات التنصيص الأحادية أو الثنائية أما في ماتلاب فلا يمكن استخدام الثنائية، عند تشغيل الوضعية `traditional` تعيد بيئة أوكتاف رسالة خطأ إذا تم استخدام إشارات التنصيص الثنائية.

R

لغة البرمجة عالية المستوى `R` تعمل جيداً وتحقق نتائجاً مرضيةً بشكل كبير في مجال الحسابات الإحصائية والبرمجيات التي تعتمد كثيراً على الإحصائيات. تمت كتابة

الشيفرة الأساسية لـ R بواسطة لغة C وهي مجانية ومفتوحة المصدر. يمكن في R إنشاء النماذج الإحصائية الخطية وغير الخطية والرسومات البيانية والمخططات الإحصائية بشكل سهل وسريع مع العديد من الميزات التي تجعل العمل ممتعاً. يمكن للمبرمجين استخدام R مع لغات برمجة أخرى مثل C, C++, Java, .NET, python لبناء تطبيقات تستفيد من ميزات كلا الجانبين وبالأخص استخدام الأدوات الجاهزة في R مباشرة دون إعادة برمجتها في لغة البرمجة الأخرى التي يمكن أن تقدم تسهيلات أكثر في مجال مختلف، هذا يشبه استخدام أدوات ماتلاب في لغات البرمجة الأخرى.

تعمل أوامر R أيضا من خلال سطر أوامر بدون الحاجة إلى ترجمة الكود بل تتم الترجمة فورياً كما في ماتلاب.

يمكن استخدام R بدلاً من ماتلاب عند كون التطبيق المراد إنجازه من النوع الذي يعتمد على الإحصاء أو البيانات الإحصائية، في حالات أخرى يتفوق ماتلاب.

Python

تعتبر بايثون في طبيعتها لغة برمجة عالية المستوى، وهي مجانية ومفتوحة المصدر، وهذه هي أولى ميزاتها.

يمكن استخدام بايثون للعمل في مجال المعالجة الرقمية كماتلاب باستخدام حزم إضافية كثيرة توفرها بايثون لهذا الغرض، خلافا لماتلاب الذي أعد لهذا الغرض تحديداً ولديه حزمة واحدة تندرج تحتها كل الوظائف.

تستخدم بايثون العديد من الموديولات وتصنف الوظائف ضمن فضاءات أسماء لإعطاء البرامج بنية هيكيلة واضحة أما ماتلاب فهو يدرج كل وظائفه تحت حزمة واحدة، مما يجعل بايثون تتفوق في هذا المجال فهي أسرع في التشغيل والكود المكتوب فيها أوضح وأسهل في الصيانة والإصلاح.

هناك أنواع معطيات أخرى في بايثون يمكن استخدامها غير المصفوفات مثل القوائم والقواميس، هذا لا ينفي وجود أنواع معطيات مثل نوع العدد الحقيقي والعدد الصحيح والقيمة النصية في ماتلاب ولكننا نتكلم هنا عن البنية العامة التي تنتظم فيها تلك المعطيات فماتلاب ينظم كل ذلك في مصفوفات سواء كانت أعداد حقيقية أو صحيحة أو نصوص أو غير ذلك، بينما تستخدم بايثون تلك الأنواع الأخرى المذكورة.

تتطور بايثون الآن بشكل مستمر والميزات التي تجعلها أفضل من ماتلاب في نظر الكثيرين قد تزداد تقدماً، ولكن لا أحد يمكنه توقع ما سيحدث في المستقبل من تطوير لماتلاب.

◄ كلمة اخيرة

هناك الكثير من البدائل كما ذكرنا، وهناك حالات علينا التفكير فيها بالبدائل وحالات أخرى علينا أن نصرف النظر عن البدائل، ولكن هناك أمر مفيد، فكون أغلب البدائل الجيدة هي أنظمة مجانية ومفتوحة المصدر فمن وجهة نظري لا بأس من الاطلاع على تلك البدائل وتعلم القليل عنها قبل التفكير في شراء ماتلاب، هذا إن كنا نقرر التعلم كأشخاص، أما إذا كنا نتكلم عن مؤسسة تريد اتخاذ قرار بهذا الشأن فأرى أن تقوم المؤسسة بإجراء دراستها الخاصة لا أن تعتمد على دراسات الآخرين في هذا الشأن فمعظم التجارب هنا تأخذ الطابع الشخصي بامتياز، وعلى سبيل المثال يرى بعضهم أن لغة بايثون لغة جميلة وهي ميزة في نظرهم، هذا التعبير عن اللغة بأنها جميلة هو تعبير شخصي بمعنى أنه لا يستند إلى معايير علمية وأشياء قابلة للقياس، وكلنا نعلم أن ما لا يقاس لا يعبر عن كميات علمية، إذن لماذا هي جميلة؟ في الواقع ربما أفهم أنا هذا وربما تفهم أنت ولكن ربما يجد آخرون ماتلاب أجمل. من جهة أخرى فبعض التقييمات ولو كانت علمية في إجراء المقارنات إلا أن الأمر يرتبط بطريقة أو أخرى بنوعية البرمجيات التي تفضل مؤسسة ما أن تتعامل معها وتثق بها، فبعض المؤسسات تفضل التعامل مع نظام تشغيل لينكس في حين أن مؤسسات أخرى تفضل التعامل مع نظام تشغيل ويندوز، والسبب دائما هو أن ميزات أحد النظامين تكون هامة من وجهة نظر زبون معين بينما يركز زبون آخر على ميزات أخرى تكون هي الهامة لديه.

بعد هذا الكلام وكل هذه المقارنة نقول أن قرار استخدام ماتلاب أو غيره يجب أن يتخذ بعناية ويمكن إجراء دراسة خاصة بنا أو الاستعانة بجهة استشارية تقوم بهذه الدراسة بشكل مفصل ودقيق وفق معايير علمية وبارامترات قابلة للقياس، أما المقارنات التي تمت في السطور السابقة فهي لإطلاع القارئ الكريم على تلك البدائل باختصار.

الملحقات

◄ الملحق (أ): الرموز المستخدمة في النصوص واختصاراتها في ماتلاب

الاختصار	الرمز	الاختصار	الرمز	الاختصار	الرمز
\alpha	α	\upsilon	Υ	\sim	~
\beta	β	\phi	Φ	\leq	≤
\gamma	γ	\chi	Χ	\infty	∞
\delta	δ	\psi	Ψ	\clubsuit	♣
\epsilon	ε	\omega	Ω	\diamondsuit	♦
\zeta	ζ	\Gamma	Γ	\heartsuit	♥
\eta	η	\Delta	Δ	\spadesuit	♠
\theta	θ	\Theta	Θ	\leftrightarrow	↔
\vartheta	ϑ	\Lambda	Λ	\leftarrow	←
\iota	ι	\Xi	Ξ	\uparrow	↑
\kappa	κ	\Pi	Π	\rightarrow	→
\lambda	λ	\Sigma	Σ	\downarrow	↓
\mu	μ	\Upsilon	Υ	\circ	○
\nu	ν	\Phi	Φ	\pm	±
\xi	ξ	\Psi	Ψ	\geq	≥
\pi	π	\Omega	Ω	\propto	∝
\rho	ρ	\forall	∀	\partial	∂
\sigma	σ	\exists	∃	\bullet	•
\Varsigma	ς	\ni	∋	\div	÷
\tau	τ	\cong	≅	\neq	≠

\equiv	≡	\approx	≈	\aleph	ℵ
\Im	ℑ	\Re	ℜ	\wp	℘
\otimes	⊗	\oplus	⊕	\oslash	⊘
\cap	∩	\cup	∪	\supseteq	⊇
\supset	⊃	\subseteq	⊆	\subset	⊂
\int	∫	\in	∈	\o	*o*
\rfloor	ẽ	\lceil	Ė	\nabla	∇
\lfloor	û	\cdot	·	\ldots	…
\perp	⊥	\neg	¬	\prime	′
\wedge	∧	\times	×	\0	ø
\rceil	Ù	\surd	√	\mid	|
\vee	∨	\varpi	ϖ	\copyright	©
\langle	⌞	\rangle	⌞		

◄ الملحق (ب): جدول ASCII

Char	Dec	Oct	Hex	Char	Dec	Oct	Hex
(nul)	0	0000	0x00	(sp)	32	0040	0x20
(soh)	1	0001	0x01	!	33	0041	0x21
(stx)	2	0002	0x02	"	34	0042	0x22
(etx)	3	0003	0x03	#	35	0043	0x23
(eot)	4	0004	0x04	$	36	0044	0x24
(enq)	5	0005	0x05	%	37	0045	0x25
(ack)	6	0006	0x06	&	38	0046	0x26
(bel)	7	0007	0x07	'	39	0047	0x27
(bs)	8	0010	0x08	(	40	0050	0x28
(ht)	9	0011	0x09	)	41	0051	0x29
(nl)	10	0012	0x0a	*	42	0052	0x2a
(vt)	11	0013	0x0b	+	43	0053	0x2b
(np)	12	0014	0x0c	,	44	0054	0x2c
(cr)	13	0015	0x0d	-	45	0055	0x2d
(so)	14	0016	0x0e	.	46	0056	0x2e
(si)	15	0017	0x0f	/	47	0057	0x2f
(dle)	16	0020	0x10	0	48	0060	0x30
(dc1)	17	0021	0x11	1	49	0061	0x31
(dc2)	18	0022	0x12	2	50	0062	0x32
(dc3)	19	0023	0x13	3	51	0063	0x33
(dc4)	20	0024	0x14	4	52	0064	0x34
(nak)	21	0025	0x15	5	53	0065	0x35
(syn)	22	0026	0x16	6	54	0066	0x36
(etb)	23	0027	0x17	7	55	0067	0x37
(can)	24	0030	0x18	8	56	0070	0x38
(em)	25	0031	0x19	9	57	0071	0x39
(sub)	26	0032	0x1a	:	58	0072	0x3a
(esc)	27	0033	0x1b	;	59	0073	0x3b
(fs)	28	0034	0x1c	<	60	0074	0x3c
(gs)	29	0035	0x1d	=	61	0075	0x3d
(rs)	30	0036	0x1e	>	62	0076	0x3e
(us)	31	0037	0x1f	?	63	0077	0x3f

Char	Dec	Oct	Hex	Char	Dec	Oct	Hex
@	64	0100	0x40	`	96	0140	0x60
A	65	0101	0x41	a	97	0141	0x61
B	66	0102	0x42	b	98	0142	0x62
C	67	0103	0x43	c	99	0143	0x63
D	68	0104	0x44	d	100	0144	0x64
E	69	0105	0x45	e	101	0145	0x65
F	70	0106	0x46	f	102	0146	0x66
G	71	0107	0x47	g	103	0147	0x67
H	72	0110	0x48	h	104	0150	0x68
I	73	0111	0x49	i	105	0151	0x69
J	74	0112	0x4a	j	106	0152	0x6a
K	75	0113	0x4b	k	107	0153	0x6b
L	76	0114	0x4c	l	108	0154	0x6c
M	77	0115	0x4d	m	109	0155	0x6d
N	78	0116	0x4e	n	110	0156	0x6e
O	79	0117	0x4f	o	111	0157	0x6f
P	80	0120	0x50	p	112	0160	0x70
Q	81	0121	0x51	q	113	0161	0x71
R	82	0122	0x52	r	114	0162	0x72
S	83	0123	0x53	s	115	0163	0x73
T	84	0124	0x54	t	116	0164	0x74
U	85	0125	0x55	u	117	0165	0x75
V	86	0126	0x56	v	118	0166	0x76
W	87	0127	0x57	w	119	0167	0x77
X	88	0130	0x58	x	120	0170	0x78
Y	89	0131	0x59	y	121	0171	0x79
Z	90	0132	0x5a	z	122	0172	0x7a
[	91	0133	0x5b	{	123	0173	0x7b
\	92	0134	0x5c	|	124	0174	0x7c
]	93	0135	0x5d	}	125	0175	0x7d
^	94	0136	0x5e	~	126	0176	0x7e
_	95	0137	0x5f	(del)	127	0177	0x7f

- يعتبر ماتلاب لغة برمجة عالية المستوى من أجل الحسابات الرقمية والرسومية وبناء التطبيقات النهائية.
- لدى ماتلاب واجهة تفاعلية تمكن المستخدم من إجراء الحسابات بسرعة دون برمجة وحل المسائل بإدخالات بسيطة وسريعة.
- يوفر مكتبات هائلة متنوعة تضم توابعا مبنية مسبقا لإجراء الحسابات في شتى المجالات كالجبر والتحليل الرياضي والإحصاء والتحليل العددي ومعالجة الإشارة وغير ذلك.
- يوفر أيضا مكتبات من التوابع الجاهزة لإنجاز الرسوم والمخططات وتعديلها بما يناسب الهدف منها بالإضافة إلى إمكانية حفظها لاستخدامها لاحقا أو تصديرها كصور.
- لدى ماتلاب أدوات تمكن المطور من بناء تطبيقات بمزج إمكانيات ماتلاب مع تطبيقات أخرى مثل جافا أو إكسيل .
- يملك أيضا محرر خاص به لكتابة الشيفرات البرمجية يمكن من كتابة تلك الشيفرات ومراجعتها وتطويرها بسهولة لما للمحرر من ميزات وإمكانات.