

Software

José Luis de la Fuente O'Connor

Profesor Titular de Matemática Aplicada

jldelafuente@etsii.upm.es

joseluis.delafuente@upm.es

Mayo de 2017

En lo que sigue se listan todos los programas, o *software*, que se describen en el libro “Ingeniería de los algoritmos y métodos numéricos. Un acercamiento práctico y avanzado a la computación científica e ingenieril con MATLAB”.

Si este documento se consulta desde la web, y se desea descargar un determinado programa, sólo hay que hacer *click* en el enlace con su nombre.

1 Para MATLAB

- **Aritmética con precisión finita y operaciones algebraicas elementales.**

Horner.m	Evalúa un polinomio de grado dado con la regla de Horner.
epsilon.m	Obtiene el <i>epsilon</i> de la máquina.
expSeriesPlot.m	Obtención de e^x mediante desarrollo en serie.
Suma_de_serie_1.m	Se simula la suma de $1/k^2$ con infinitos sumandos.
wilkpoly.m	Evalúa el polinomio de Wilkinson (grado 20) en un punto dado.
Mat_operaciones.m	Incluye los códigos para calcular los productos de matrices por vectores y de dos matrices: MatVecF, MatVecC, VecMatC, VecMatF, MatMatInt, MatMatVec y MatMatExte.
strass.m	Calcula el producto de dos matrices $N \times N$, $N = 2^x$, mediante la formula de Strassen.
TiemposAxB.m	Obtiene el producto de dos matrices mediante todos los métodos expuestos: para conocer sus tiempos de cálculo.

- **Solución de ecuaciones no lineales de una variable y sistemas de varias variables (incluye mínimos cuadrados no lineales).**

Bisec_0.m	Resuelve $f(x) = 0$ mediante el método iterativo de la Bisección. Se puede probar con bisec_1.m .
fpi.m	Iteración de punto fijo para resolver $f(x)=x$.
Newton.m	Resuelve $f(x)=0$ mediante el método de Newton-Raphson. Se puede probar con Newt_1.m .
Newton_composite.m	Resuelve $f(x) = 0$ mediante el método de Newton relajado: evalúa la derivada cada dos iteraciones. Se puede probar con Newt_1.m .
HalleysMethod_log.m	Resuelve $x + \ln(x) = 0$ mediante el método de Halley.
Newton_sec.m	Resuelve $f(x)=0$ mediante el método de la secante. Se puede probar con Newt_sec_1.m .
Muller_2.m	Resuelve $f(x) = 0$ mediante el método de Muller. Se puede probar con Newt_1.m y con Muller_2_2.m .
Brent.m	Resuelve $f(x) = 0$ mediante el método de Brent.
Newt_Rap.m	Resuelve un sistema no lineal de ecuaciones mediante el método de Newton-Raphson. Se puede probar con NRP_1.m .
Newtrp_df_1.m	Resuelve un sistema no lineal de ecuaciones mediante el método de Newton-Raphson con la matriz jacobiana calculada por diferencias finitas. Se puede probar con NRP_1_dif.m .
Broyden_3.m	Resuelve un sistema no lineal de ecuaciones mediante el método de Newton-Raphson con la matriz jacobiana adaptada de iteración en iteración mediante la fórmula de Broyden. Se puede probar con NRP_1.m .
GN111.m	Resuelve un problema no lineal de mínimos cuadrados mediante el método de Gauss-Newton. Se puede probar con GaNew.m .
Levmarmar_99.m	Resuelve un problema no lineal de mínimos cuadrados mediante el método de Levenberg-Marquardt.
LevenbergMarquardt_2.m	Resuelve un problema no lineal de mínimos cuadrados mediante una variante del método de Levenberg-Marquardt y calculando la matriz Jacobiana por diferencias finitas.
LSQ_New_1.m	Resuelve un problema no lineal de mínimos cuadrados mediante el método de Newton. Se puede probar con Newt_Ls.m .

[LSQ_New_sym_2.m](#) Resuelve un problema no lineal de mínimos cuadrados mediante el método de Newton usando la potencia del cálculo simbólico de MATLAB.

- **Solución de sistemas de ecuaciones lineales: métodos directos.**

[Gauss.m](#) Obtiene la solución de un sistema de ecuaciones lineales $Ax = b$ mediante eliminación de Gauss con pivotación parcial.

[LUCrout.m](#) Calcula la factorización LU de una matriz con el método de Crout. No se destruye la matriz original.

[Crout_1.m](#) Calcula la factorización LU de una matriz con el método de Crout. Se destruye la matriz original en la que se guarda la solución.

[CroutP.m](#) Calcula la factorización LU de una matriz mediante el método de Crout con pivotación.

[Crout1lu.m](#) Calcula la factorización L_1U de una matriz mediante el método de Crout.

[CroutP1.m](#) Calcula la factorización L_1U de una matriz mediante el método de Crout con pivotación.

[InvLU_1.m](#) Calcula la inversa de una matriz a partir de su factorización LU .

[Chols_1.m](#) Calcula la factorización de Cholesky de una matriz simétrica definida positiva.

[Diagpiv_1.m](#) Calcula la factorización diagonal LDL^T de una matriz simétrica mediante el método de Bunch y Kaufman con pivotación.

- **Mínimos cuadrados lineales.**

[Grmsch_2.m](#) Resuelve un sistema de ecuaciones lineales –mínimos cuadrados– mediante el método de Gram-Schmidt clásico.

[Grmsch_3.m](#) Resuelve un sistema de ecuaciones lineales –mínimos cuadrados– mediante el método de Gram-Schmidt modificado.

[Qrdes_3.m](#) Obtiene la solución de un problema de mínimos cuadrados a través de la factorización QR de su matriz mediante transformaciones de Householder.

[Mincua_QR.m](#) Se resuelve un problema de mínimos cuadrados general, incluso de rango incompleto, mediante transformaciones de Householder.

[Givens.m](#) Se calcula la solución de un problema de mínimos cuadrados a través de la factorización QR de su matriz mediante transformaciones de Givens.

[Svdre.m](#) Se resuelve un problema de mínimos cuadrados general, incluso de rango incompleto, mediante la descomposición en valores singulares, SVD , de su matriz. Resuelve un sistema de ecuaciones lineales –mínimos cuadrados– mediante el método de Gram-Schmidt clásico.

- **Funciones de interpolación y aproximación.**

[lagrang_int.m](#) Calcula el polinomio de interpolación de Lagrange de un conjunto de pares de puntos. Se puede probar con [do_Lagrange_int.m](#) o [do_Lagrange_int_gas_1.m](#).

[Newton_int_1.m](#) Calcula el polinomio de interpolación de Newton de un conjunto de pares de puntos. Se puede probar con [do_newton_int_1.m](#).

[chebypoly.m](#) Calcula los coeficientes del polinomio de Chebyshev de un grado dado.

[chebpol_int.m](#) Calcula los coeficientes del polinomio de Chebyshev de un grado dado que mejor interpola puntos de una función dada usando puntos de Chebyshev. Se puede probar con [Chebpol_test.m](#).

[chebpol_clenshaw.m](#) Calcula los coeficientes del polinomio de Chebyshev de un grado dado de una función usando puntos de Chebyshev.

- **Métodos iterativos para sistemas de ecuaciones lineales.**

[Jacobi_2.m](#) Resuelve un sistema de ecuaciones lineales $Ax = b$ mediante el método iterativo de Jacobi.

[Jacobi_NEW_1.m](#) Resuelve un sistema de ecuaciones lineales $Ax = b$ mediante el método iterativo de Jacobi en forma compacta.

[jacobi_S_1.m](#) Resuelve un sistema de ecuaciones lineales, $Ax = b$, con matriz dispersa, mediante el método iterativo de Jacobi. Se puede probar con [sparsesetup.m](#).

[Gauss_Seidel_1_1.m](#) Calcula la solución de un sistema de ecuaciones lineales $Ax = b$ mediante el método de Gauss-Seidel.

Gauss_Seidel_NEW.m	Resuelve un sistema de ecuaciones lineales, $Ax = b$ mediante el método iterativo de Gauss-Seidel en forma compacta.
SOR.m	Resuelve un sistema de ecuaciones lineales, $Ax = b$ mediante el método iterativo de la sobrerelajación sucesiva: <i>Successive Overrelaxation</i> .
sor_1.m	Resuelve un sistema de ecuaciones lineales, $Ax = b$ mediante el método iterativo de la sobrerelajación sucesiva, <i>Successive Overrelaxation</i> , en forma compacta.
SSOR_1.m	Calcula la solución de un sistema de ecuaciones lineales $Ax = b$ mediante el método de la sobrerelajación sucesiva simétrica, <i>Symmetric Successive Overrelaxation</i> .
Steep.m	Resuelve un sistema de ecuaciones lineales $Ax = b$ mediante el método iterativo de la máxima pendiente.
CGr.m	Resuelve un sistema de ecuaciones lineales $Ax = b$ mediante el método iterativo de los gradientes conjugados.
PCGr.m	Resuelve un sistema de ecuaciones lineales $Ax = b$ mediante el método iterativo de los gradientes conjugados con preconditionamiento de la matriz A .
gmres_Netlib.m	Calcula la solución de un sistema de ecuaciones lineales $Ax = b$ mediante el algoritmo GMRES, <i>Generalized Minimal Residual</i> .

- **Cálculo de valores y vectores propios, y valores singulares.**

Gershgoring.m	Dibuja los círculos de Gerschgorin de una matriz.
Jacobi_val_12_2.m	Calcula los valores y vectores propios de una matriz simétrica mediante el método de Jacobi.
potencia.m	Método de la iteración de potencia para la obtención de un valor propio de una matriz y su vector asociado.
ItInversa_2.m	Método de la iteración inversa de potencia para la obtención de un valor propio de una matriz y su vector asociado.
ItInvRayleigh_2.m	Método de la iteración inversa con cociente de Rayleigh para la obtención de un valor propio de una matriz y su vector asociado.
defl_1.m	Se deflacta un vector propio de una matriz y se calcula el valor propio dominante de la matriz resultante y su vector asociado.
IterSimul.m	Calcula un número dado de los máximos valores propios de una matriz A simétrica mediante el método de la iteración simultánea ortogonal.
IteracionQR.m	Calcula todos los valores propios de una matriz A con valores propios reales –simétrica, hermítica, etc.– mediante la iteración QR con desplazamiento simple de Rayleigh.
Hessred.m	Reducción de una matriz a la forma de Hessenberg mediante transformaciones de Householder.
It_QR_3.m	Método de la iteración QR con desplazamiento de Wilkinson para el cálculo de todos los valores propios de una matriz simétrica.
qrstep_Francis_Bindel.m	Método de la iteración QR con doble desplazamiento implícito tipo Francis y mejoras de Bindel para el cálculo de todos los valores propios reales o complejos de una matriz, reduciéndola previamente a la forma de Hessenberg.
arnoldi_W.m	Realiza un número dado de iteraciones de Arnoldi con reortogonalización en una matriz A . Se puede probar con arngo.m .
bidiag.m	Bidiagonaliza una matriz mediante transformaciones de Householder.
svd_GolRein_1.m	Calcula la descomposición en valores singulares de una matriz mediante el algoritmo de Golub Reinsch.
svd_J.m	Calcula la descomposición en valores singulares de una matriz mediante el algoritmo de Jacobi.

- **Optimización. Programación no lineal sin condiciones.**

Maxima_pendiente_unc.m	Optimiza una función no lineal mediante el método de la máxima pendiente. Se puede probar con la rutina incluida en el listado.
--	---

Newton_unc.m	Optimiza una función no lineal mediante el método de Newton. Se puede probar con una rutina incluida en el listado.
Newton_mp.m	Optimiza una función no lineal mediante un método que combina Newton y máxima pendiente. Se puede probar con la rutina incluida en el listado.
Newton_amortiguado.m	Optimiza una función no lineal mediante el método de Newton amortiguado y paso completo. Se puede probar con la rutina incluida en el listado.
Dogleg_UBC_yo.m	Algoritmo de minimización de una función mediante el método de región de confianza DogLeg con modelo de Newton.
Grad_Conjugados_unc.m	Algoritmo de minimización de una función mediante el método de los gradientes conjugados. Se prueba con una rutina interior.
quasi_newton_1.m	Algoritmo de minimización de una función mediante diversos métodos cuasi Newton: DFP o BFGS.

- **Optimización. Programación lineal.**

ProgLineal_3.m	Resuelve un problema de Programación Lineal, Min. $c^T x$, s.a $Ax = b$, $x \geq 0$, mediante el método Simplex. Se puede probar con ProgLineal_3_Ejemplo_clase.mat .
Rsimplex_1.m	Resuelve un problema de Programación Lineal, Min. $c^T x$, s.a $Ax = b$, $l \leq x \leq u$, mediante el método Simplex en dos fases.
dual_simplex.m	Resuelve un problema de Programación Lineal, Min. $c^T x$, s.a $Ax = b$, $l \leq x \leq u$, mediante el método dual del Simplex.
PL_pd_simplex_1.m	Resuelve un problema de Programación Lineal, Min. $c^T x$, s.a $Ax = b$, $x \geq 0$, mediante los programas ProgLineal_3 o dual_simplex según la base de partida sea factible o no. Se puede dar la base o no.
IntPointLP_2.m	Resuelve un problema de Programación Lineal, Min. $c^T x$, s.a $Ax = b$, $x \geq 0$, mediante el método de punto interior primal-dual.
ProgLineal_InP_MPS_sc.m	Resuelve un problema de Programación Lineal, Min. $c^T x$, s.a $Ax = b$, $x \geq 0$, mediante el método de punto interior primal-dual pudiendo leer los datos en formato MPS.
PLip_1.m	Resuelve un problema de Programación Lineal, de cualquier tamaño, Min. $c^T x$, s.a $Ax = b$, $l \leq x \leq u$, mediante el método de punto interior predictor corrector —basado en Lipsol—. Se puede probar con cualquier fichero en formato MPS, mat de Matlab, o introduciendo los datos de matrices y vectores.

- **Optimización. Programación no lineal con condiciones.**

pcQP_gen_2.m	Resuelve un problema de Programación Cuadrática con condiciones de igualdad y desigualdad mediante un algoritmo de punto interior predictor corrector.
SQP_e.m	Resuelve un problema de Programación No Lineal con condiciones de igualdad mediante un algoritmo SQP —Programación Cuadrática Secuencial—.
SQP_ie_1.m	Resuelve un problema de Programación No Lineal con condiciones de desigualdad mediante un algoritmo SQP.
SQP_general_1.m	Resuelve un problema de Programación No Lineal con condiciones de igualdad y desigualdad mediante un algoritmo SQP.
NLP_IP_nc.m	Resuelve un problema de Programación No Lineal con condiciones de igualdad y desigualdad mediante un algoritmo de punto interior general.

- **Derivación e integración de funciones.**

trapezrule.m	Integra una función mediante la regla del trapecio.
simpson_1.m	Integra una función dada en un número de intervalos mediante la regla de Simpson.
romberg.m	Calcula una integral definida en el intervalo $[a, b]$ mediante la tabla de n filas según la fórmula de Romberg.

`lgwt.m` Calcula los N nodos y pesos de Gauss-Legendre en el intervalo $[a, b]$ para el cálculo de integrales definidas.

`quadadapt.m` Evalúa la integral de $f(x)$ en $[a, b]$ mediante cuadratura adaptativa.

- **Integración de ecuaciones diferenciales ordinarias.**

`Euler_2_S.m` Script de MATLAB para integrar $y'(t) = ty + t^3$ mediante el método de Euler.

`Heun_4.m` Integra una ecuación diferencial ordinaria mediante el método de Heun.

`test_ode45_1.m` Script de MATLAB para comparar la integración de $y'(t) = y(-2t + 1/t)$ mediante el método de Euler, Heun, Runge-Kutta de orden 4 y ODE45 de MATLAB.

`back_euler_1.m` Integra la ecuación diferencial ordinaria $y'(t) = -\alpha(y - t^2) + 2t$ mediante el método de Euler hacia atrás.

`Adams_Bas_Moul.m` Integra la ecuación diferencial ordinaria $y'(t) = 4(t - t^3)y^2$ mediante el método de Adams-Bashforth-Moulton de segundo orden.

- **Integración de problemas de contorno.**

`nlbvpfd.m` Se resuelve por diferencias finitas un problema de contorno concreto.

`Galerkin_ef_1.m` Se resuelve por elementos finitos un problema de contorno concreto.

- **Solución numérica de ecuaciones en derivadas parciales.**

`heatfd.m` Se resuelve con MATLAB la ecuación del calor por diferencias adelantadas.

`heatbd.m` Se resuelve con MATLAB la ecuación del calor por diferencias atrasadas.

`Crank_Nicolson.m` Se resuelve con MATLAB la ecuación del calor mediante el método de Crank-Nicolson.

`wavefd.m` Se resuelve con MATLAB la ecuación del onda por diferencias adelantadas.

`Poisson.m` Se resuelve con MATLAB la ecuación de Poisson por diferencias finitas.

`Poisson_FEM_1.m` Se resuelve con MATLAB la ecuación de Poisson por Elementos Finitos.

`Burgers.m` Se resuelve con MATLAB la ecuación de Burgers, en derivadas parciales no lineal, mediante diferencias atrasadas y Newton-Raphson.

2 En FORTRAN

- `bbmi.f90` Programa en Fortran para resolver problemas de Programación Lineal y Programación Entera de grandes y pequeñas dimensiones. Su ejecutable es `bbmi.exe`. Los datos del problema que hay que resolver deben estar en formato MPS, ampliado para tener en cuenta variables enteras. El manual de utilización se encuentra en el Apéndice E del libro *Técnicas de cálculo para sistemas de ecuaciones, programación lineal y programación entera*.
- `mps2mat.f` Rutina, para utilizar con Matlab, para leer un fichero en formato `.MPS` y convertirlo en `.MAT`. Ya compilado está en `mps2mat.mexw64`, para 64 bits, o en `mps2mat.mexw32` para 32. Se llama en el entorno de Matlab con `mpsAmat.m`.