Linear Algebra, Optimization and Solving Ordinary Differential Equations Using Scilab

Deepak U. Patil
deepakp@ee.iitb.ac.in

Indian Institute of Technology, Bombay

November 8, 2009

System of Linear Equations

- Consider
  $x_1 + x_2 + x_3 = 10$
  $3x_1 + x_2 + 2x_3 = 5$
  $x_1 + x_2 - x_3 = 1.$

System of Linear Equations

- Consider

  $x_1 + x_2 + x_3 = 10$
  $3x_1 + x_2 + 2x_3 = 5$
  $x_1 + x_2 - x_3 = 1.$
- Can be represented as

  $Ax = b$

  where $A = \begin{pmatrix} 1 & 1 & 1 \\ 3 & 1 & 2 \\ 1 & 1 & -1 \end{pmatrix}$

  and $b = \begin{pmatrix} 10 \\ 5 \\ 1 \end{pmatrix}$.

System of Linear Equations

- Consider
  $x_1 + x_2 + x_3 = 10$
  $3x_1 + x_2 + 2x_3 = 5$
  $x_1 + x_2 - x_3 = 1.$
- Can be represented as
  $Ax = b$
  where $A = \begin{pmatrix} 1 & 1 & 1 \\ 3 & 1 & 2 \\ 1 & 1 & -1 \end{pmatrix}$
  and $b = \begin{pmatrix} 10 \\ 5 \\ 1 \end{pmatrix}$.
- Number of Equations may or may not be equal to number of unknowns.

Solution by Scilab

- Solve using single line code
  `x=A\b`

Solution by Scilab

- Solve using single line code
  `x=A\b`
- Or use command
  `[x,ker]=linsolve(A,b)`

Solution by Scilab

- Solve using single line code
  `x=A\b`
- Or use command
  `[x,ker]=linsolve(A,b)`
- To find Kernel(nullspace) of a system separately use
  `ker=kernel(A)`

Other useful functions

- `[D,X]=bdiag(A)      //Block Diagonalisation`

Other useful functions

- `[D,X]=bdiag(A)      //Block Diagonalisation`
- `[U,S,V]=svd(A)      //Singular Value Decomposition`

Other useful functions

- `[D,X]=bdiag(A)      //Block Diagonalisation`
- `[U,S,V]=svd(A)       //Singular Value Decomposition`
- `[L,U]=lu(A)       //Lower and Upper Traingular form Decomposition`

Other useful functions

- `[D,X]=bdiag(A)      //Block Diagonalisation`
- `[U,S,V]=svd(A)       //Singular Value Decomposition`
- `[L,U]=lu(A)       //Lower and Upper Traingular form Decomposition`
- `[Q,R]=qr(A)       //QR-Decomposition`

Examples

- Solve
  $x_1 + 4x_2 = 34$
  $-3x_1 + x_2 = 2$

Examples

- Solve
  $x_1 + 4x_2 = 34$
  $-3x_1 + x_2 = 2$
- Solve
  $2x_1 - 2x_2 + 3x_3 = 1$
  $x_1 + 2x_2 + 3x_3 = 2$

Examples

- Solve
  $x_1 + 4x_2 = 34$
  $-3x_1 + x_2 = 2$
- Solve
  $2x_1 - 2x_2 + 3x_3 = 1$
  $x_1 + 2x_2 + 3x_3 = 2$
- Use `linsolve`

Examples

- Solve
  $x_1 + 4x_2 = 34$
  $-3x_1 + x_2 = 2$
- Solve
  $2x_1 - 2x_2 + 3x_3 = 1$
  $x_1 + 2x_2 + 3x_3 = 2$
- Use `linsolve`
- Try this for previously obtained solution
  `A*x  A*(x+ker)   //In this case kernel is a line`

Nonlinear Root Finding

- Many real world problems requires us to solve $f(x) = 0$

Nonlinear Root Finding

- Many real world problems requires us to solve $f(x) = 0$
- In Scilab `fsolve` can be used.

Nonlinear Root Finding

- Many real world problems requires us to solve $f(x) = 0$
- In Scilab `fsolve` can be used.
- Function can be defined in a separate file.

Nonlinear Root Finding

- Many real world problems requires us to solve $f(x) = 0$
- In Scilab `fsolve` can be used.
- Function can be defined in a separate file.
- Function is passed as an argument.

Nonlinear Root Finding

- Many real world problems requires us to solve $f(x) = 0$
- In Scilab `fsolve` can be used.
- Function can be defined in a separate file.
- Function is passed as an argument.
- For Example:
  Solve $x^2 + 3x + 2 = 0$
  ```
  deff('y=f(x)','y=x^ 2+3*x+2')
  x=fsolve(x0,f)
  ```
  where x0 is initial guess.

Nonlinear Root Finding

- Many real world problems requires us to solve $f(x) = 0$
- In Scilab `fsolve` can be used.
- Function can be defined in a separate file.
- Function is passed as an argument.
- For Example:
  Solve $x^2 + 3x + 2 = 0$
  ```
  deff('y=f(x)','y=x^ 2+3*x+2')
  x=fsolve(x0,f)
  ```
  where x0 is initial guess.
- One can also define function $f : R^n \rightarrow R^n$ and solve it for zero locations.

Minimizing a Function

- Maximizing a function $f(x)$ is same as minimizing $-f(x)$.

Minimizing a Function

- Maximizing a function $f(x)$ is same as minimizing $-f(x)$.
- `optim` is the inbuilt function for this purpose.

Minimizing a Function

- Maximizing a function $f(x)$ is same as minimizing $-f(x)$.
- `optim` is the inbuilt function for this purpose.
- It can be used for both Constrained and Unbounded minimization Problem.

Minimizing a Function

- Maximizing a function $f(x)$ is same as minimizing $-f(x)$.
- `optim` is the inbuilt function for this purpose.
- It can be used for both Constrained and Unbounded minimization Problem.
- `[f,xopt]=optim(costf,x0)    //gradient has to be specified`

Minimizing a Function

- Maximizing a function $f(x)$ is same as minimizing $-f(x)$.
- `optim` is the inbuilt function for this purpose.
- It can be used for both Constrained and Unbounded minimization Problem.
- `[f,xopt]=optim(costf,x0)    //gradient has to be specified`

Minimizing a Function

- Maximizing a function $f(x)$ is same as minimizing $-f(x)$.
- `optim` is the inbuilt function for this purpose.
- It can be used for both Constrained and Unbounded minimization Problem.
- `[f,xopt]=optim(costf,x0)    //gradient has to be specified`
- `[f,xopt]=optim(list(NDcost,myf),x0)`

For Example

- Minimize:
  $f(x,y) = (x+y)^2 + x + y + 2$
- Gradient of the Function f
  $\nabla f = \begin{pmatrix} 2(x+y)+1 & 2(x+y)+1 \end{pmatrix}$

Numerical Differentiation

- `g=numdiff(f,x)`

Numerical Differentiation

- `g=numdiff(f,x)`
- If $f : R^n \rightarrow R$, then g is gradient of f at x.

Numerical Differentiation

- `g=numdiff(f,x)`
- If $f : R^n \rightarrow R$, then g is gradient of f at x.
- If $f : R^n \rightarrow R^m$, then g is Jacobian a $m \times n$ Matrix.

Hessian

- `[g,H]=derivative(f,x)` is the calling sequence
- for a function $f : R^n \rightarrow R$
  g is the gradient of f
  and H is Hessian matrix of f

Ordinary Differential Equations

- `y=ode(x0,t0,t,myode)` is the calling sequence.

Ordinary Differential Equations

- `y=ode(x0,t0,t,myode)` is the calling sequence.
- x0 is initial condition.
  t0 is initial time
  t is the time instants at which solution is needed.
  'myode' is external function which defines the differential equation.

Ordinary Differential Equations

- `y=ode(x0,t0,t,myode)` is the calling sequence.
- x0 is initial condition.
  t0 is initial time
  t is the time instants at which solution is needed.
  'myode' is external function which defines the differential equation.
- Higher Order Equations must be made into first order equations of form $\dot{x} = Ax + Bu$.

Examples

- Solve the differential equation
  $\frac{\mathrm{d}^2\theta}{\mathrm{d}t^2} + \frac{g}{L} sin(\theta) = 0$
- Take $g = 9.8\ m/s^2\ L = 1\ m$
- Check the plot of solution against time using `plot2d(t,x(1,:)` and `plot2d(t,x(2,:))`
- Also obtain the phase plane plot using `plot2d(x(1,:),x(2,:))`

Thank You!

- www.scilab.org

Thank You!

- www.scilab.org
- "Modeling And Simulation in Scilab/Scicos",by S.L.Campbell, J. Chancelier, R. Nikoukah.