

Lecture 15: Non-linear equations (ODE's and otherwise)

Outline

- 0) Review Stiff linear systems
- 1) Stiff non-linear systems: the van-der Pol oscillator
- 2) Implicit methods for stiff-systems and systems of non-linear equations
- 3) General solution of non-linear systems and $\underline{F}(\underline{x})=0$
 - A) A Simpler Example
 - B) Newton's method
 - C) A demo (linear vs. non-linear systems)
- 4) Software for solution of non-linear systems

All Matlab odexx routines

see help ode45, or doc ode45

Solver	Problem Type	Order of Accuracy	When to Use
ode45	Nonstiff	Medium	Most of the time. This should be the first solver you try.
ode23	Nonstiff	Low	For problems with crude error tolerances or for solving moderately stiff problems.
ode113	Nonstiff	Low to high	For problems with stringent error tolerances or for solving computationally intensive problems.
ode15s	Stiff	Low to medium	If ode45 is slow because the problem is stiff.
ode23s	Stiff	Low	If using crude error tolerances to solve stiff systems and the mass matrix is constant.
ode23t	Moderately Stiff	Low	For moderately stiff problems if you need a solution without numerical damping.
ode23tb	Stiff	Low	If using crude error tolerances to solve stiff systems.

The algorithms used in the ODE solvers vary according to order of accuracy [6] and the type of systems (stiff or nonstiff) they are designed to solve. See [Algorithms](#) for more details.

Review: Stiff linear dynamical systems**Stiff systems of non-linear ODE's:**

van Der Pol oscillator (stiffness increases with μ)

$$y'' - \mu(1-y^2)y' + y = 0; \quad y(0)=y_0, y'(0)=v_0$$

Stiff systems of non-linear ODE's:

implicit schemes for non-linear ODE's

$$y'' - \mu(1-y^2)y' + y = 0; \quad y(0)=y_0, y'(0)=v_0$$

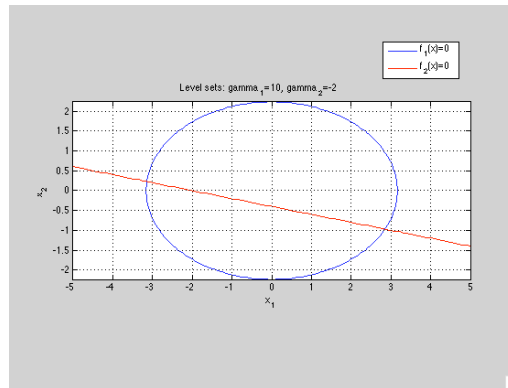
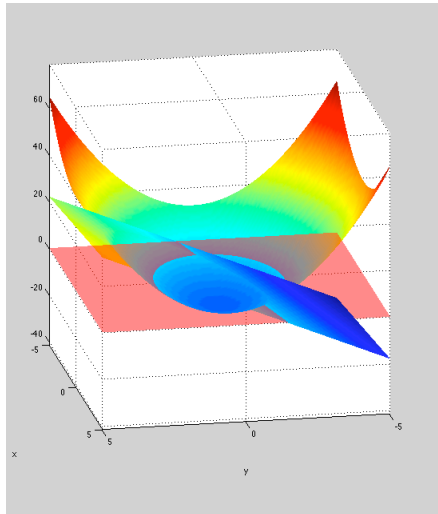
Backwards Euler:

Solving systems of non-linear equations

Example:

Solving systems of non-linear equations

Interpretation: Solve intersection of level sets



Review: solving non-linear equations of 1 variable $f(x)=0$

Example:

Available methods:

Review Newton for $f(x)=0$:

Newton's method for $\underline{F}(\underline{x})=\underline{0}$

Same basic idea:

1) given initial guess \underline{x} , find correction $\underline{\delta}$ such that

Newton's method for $\underline{F}(\underline{x})=\underline{0}$

2) Example: Linear system

Newton's method for $\underline{F}(\underline{x})=\underline{0}$

Taylor's theorem for vector valued functions.

1) consider $f(\underline{x}): \mathbb{R}^n \rightarrow \mathbb{R}$

Newton's method for $\underline{F}(\underline{x})=\underline{0}$

Taylor's theorem for vector valued functions.

2) now $\underline{F}(\underline{x}): \mathbb{R}^n \rightarrow \mathbb{R}^n$

Newton's method for $F(x)=0$

Newton's method:

Newton's method for $E(x)=0$

Some code

```
function [x, xVec, resVec] = newton(func,x0)
% function [x, xVec, resVec] = newton(func,x0)
% NEWTON - does a standard newton iteration on function starting with an
% initial guess x0. returns the final value x, such that func(x)=0 as well
% as a vector of residuals ||f(x_i)||
%
% x- solution (if converges) to func(x)=0
% xVec - vector of intermediate iterations
% resVec- vector of residuals at each iteration.
%
% func- function handle to f(x) which is assumed to return [F,J]=func(x)
%       where F is the residual (F(x) and J is the jacobian J(x)
% x0- initial guess

MAX_ITERATIONS=100;
RELATIVE_RES_TOL=1.e-14;

x = x0;
xVec = x;
for k = 1:MAX_ITERATIONS
    [f,J] = func(x);
    resVec(k) = norm(f,1);
    relResidual = resVec(k)/resVec(1);
    if (relResidual < RELATIVE_RES_TOL)
        break
    end
    x = x-J\f;
    xVec = [ xVec x];
end
resVec=resVec(:);

if (k==MAX_ITERATIONS)
    warning(sprintf('Maximum Iterations = 100, relative Residual=%g',relResidual));
end
return
```

Newton's method for $\underline{F}(\underline{x})=0$

Example: $\gamma=[10 \ -2]'$

Demo: and caution! Newton is very touchy...you need either an extremely good guess or more robust methods (e.g. Line search and/or trust region)

Stiff systems of non-linear ODE's:

van Der Pol oscillator (stiffness increases with μ)

$$y'' - \mu(1-y^2)y' + y = 0; \quad y(0)=y_0, y'(0)=v_0$$

Software with advanced algorithms for solving $\underline{F}(\underline{x})=0$

Matlab: `fsolve()` Part of the optimization toolkit (extra unfortunately):
Implements Newton with trust-region methods.

Gnu Scientific Library: GSL (C code based on MINPAK)
Newton with trust-region methods

SciPy: open source Python project for scientific computing
<http://www.scipy.org/>

PETSc: Portable Extensible Toolkit for Scientific Computation

- 1) Implements Newton + LineSearch + broad range of methods for solving large sparse linear systems in **parallel**.
- 2) Extremely useful for large sparse systems arising from numerical PDE's
- 3) Wicked steep learning curve.