Nonlinear systems

Newton's method and `SNES`

Ed Bueler
Dept. of Mathematics and Statistics, UAF

23 February 2016

outline for today

my news:

- *book contract signed with SIAM Press!*

Chapter 4 of book:

- Newton's method
  - *residual* & *Jacobian*
- a fixed-dimension `SNES` example
- a structured-grid (`DMDA`-generated) `SNES` example
- run-time options
  - finite-difference Jacobian
  - finite-difference Jacobian by coloring
  - matrix-free Newton-Krylov
- line search

linear residual

- in a linear system $A\mathbf{u} = \mathbf{b}$ the residual is a linear function:

$$\mathbf{r}(\mathbf{u}) = \mathbf{b} - A\mathbf{u}$$

- an iterative linear solver generates a sequence $\mathbf{u}_k$ which reduces the size of the linear residual $\|\mathbf{r}(\mathbf{u}_k)\|$ to zero
  - a Krylov method like Richardson, CG, or GMRES does this
  - in exact arithmetic, a direct method like LU can send the residual to zero in one step: $\mathbf{r}(\mathbf{u}_1) = 0$
    - but no such luck for nonlinear equations . . . or even higher-degree polynomials (Abel, 1823)
    - also LU might take too much time

*non*linear residual

- in a nonlinear system the residual function is general
- suppose $\mathbf{F} : \mathbb{R}^N \to \mathbb{R}^N$ is differentiable
  - input $\mathbf{x}$ and output $\mathbf{F}(\mathbf{x})$ are column vectors
  - so $\mathbf{F}$ acts like square-matrix multiplication $\mathbf{x} \mapsto A\mathbf{x}$
- to solve:

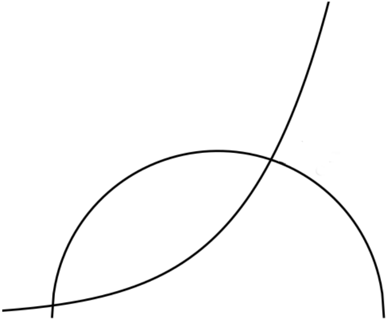
$$\mathbf{F}(\mathbf{x}) = 0$$

- reduce the nonlinear residual $\mathbf{F}$ to zero by iteration:
  - generate approximations $\mathbf{x}_k$ ... *technique needed!*
  - the goal is that $\|\mathbf{F}(\mathbf{x}_k)\|$ goes to zero ... ideally quickly!

$N = 2$ example

- find the intersection of the exponential graph $y = \frac{1}{2}e^{2x}$ and the circle $x^2 + y^2 = 1$
- that is, make this residual zero

$$\mathbf{F}(\mathbf{x}) = \begin{bmatrix} \frac{1}{2}e^{2x_0} - x_1 \\ x_0^2 + x_1^2 - 1 \end{bmatrix}$$



Jacobian = derivative of nonlinear residual

- suppose $\mathbf{x}_k$ is any vector ... perhaps an estimate of solution to $\mathbf{F} = 0$
- because $\mathbf{F}$ is differentiable, then *by definition*, for any $\mathbf{s}$,

$$\mathbf{F}(\mathbf{x}_k + \mathbf{s}) = \mathbf{F}(\mathbf{x}_k) + J_{\mathbf{F}}(\mathbf{x}_k)\mathbf{s} + o(\|\mathbf{s}\|)$$

  for some square matrix

$$J_{\mathbf{F}}(\mathbf{x}_k) = \begin{bmatrix} \frac{\partial F_0}{\partial x_0} & \dots & \frac{\partial F_0}{\partial x_{N-1}} \\ \vdots & \ddots & \vdots \\ \frac{\partial F_{N-1}}{\partial x_0} & \dots & \frac{\partial F_{N-1}}{\partial x_{N-1}} \end{bmatrix}$$

  and some quantity $o(\|\mathbf{s}\|)$ that goes to zero as $\|\mathbf{s}\| \to 0$
- the matrix $J_{\mathbf{F}}(\mathbf{x})$ is called the *Jacobian* of $\mathbf{F}$ at $\mathbf{x}$

Newton's method

- because we seek the zero of $\mathbf{F}$, we drop the $o(\|\mathbf{s}\|)$ term and seek the location of $\mathbf{x}_k + \mathbf{s}$:

$$0 = \mathbf{F}(\mathbf{x}_k) + J_{\mathbf{F}}(\mathbf{x}_k)\mathbf{s}$$

- write this linear system in form "$A\mathbf{u} = \mathbf{b}$" for unknown $\mathbf{s}$:

$$J_{\mathbf{F}}(\mathbf{x}_k)\mathbf{s} = -\mathbf{F}(\mathbf{x}_k)$$

- *Newton's method*: one iteration solves a linear system and then does a vector addition:

$$\begin{aligned} J_{\mathbf{F}}(\mathbf{x}_k)\mathbf{s} &= -\mathbf{F}(\mathbf{x}_k) \\ \mathbf{x}_{k+1} &= \mathbf{x}_k + \mathbf{s} \end{aligned}$$

easy exercise: scalar case

- if $N = 1$ (scalar case) then $\mathbf{F}(\mathbf{x}) = F(x)$ and $J_{\mathbf{F}}(\mathbf{x}) = F'(x)$
- in that case Newton's method becomes

$$F'(x_k)s = -F(x_k)$$
$$x_{k+1} = x_k + s$$

- which simplifies to the well-known formula

$$x_{k+1} = x_k - F(x_k)/F'(x_k)$$

- with well-known picture: find tangent line at $(x_k, F(x_k))$ and let x_{k+1} be the point on the x-axis where the tangent line crosses

$N = 2$ example, continued

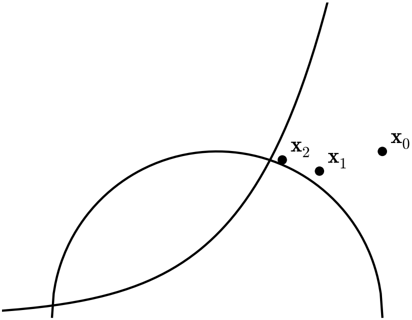
- recall residual:

$$\mathbf{F}(\mathbf{x}) = \begin{bmatrix} \frac{1}{2}e^{2x_0} - x_1 \\ x_0^2 + x_1^2 - 1 \end{bmatrix}$$

- Jacobian:

$$J_{\mathbf{F}}(\mathbf{x}) = \begin{bmatrix} e^{bx_0} & -1 \\ 2x_0 & 2x_1 \end{bmatrix}$$

- for $b = 2$, if we start from $\mathbf{x}_0 = [1 \;\; 1]^\top$, then Newton's iterates are



$$\mathbf{x}_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad \mathbf{x}_1 = \begin{bmatrix} 0.619203 \\ 0.880797 \end{bmatrix}, \quad \mathbf{x}_2 = \begin{bmatrix} 0.394157 \\ 0.948623 \end{bmatrix}, \quad \dots$$

SNES

- a `SNES` object manages Newton's method
  - "scalable nonlinear equations solvers"
- standard `Create/Set.../Destroy` sequence:

```
SNES snes;
SNESCreate(PETSC_COMM_WORLD,&snes);
SNESSetFunction(snes,r,FormFunction,&user);
SNESSetJacobian(snes,J,J,FormJacobian,&user);
SNESSetFromOptions(snes);
SNESSolve(snes,NULL,x);
SNESDestroy(&snes);
```

  - `x` is allocated `Vec` for holding solution $\mathbf{x}$
  - `r` is allocated `Vec` for holding residual $\mathbf{F}(\mathbf{x})$
  - `J` is allocated `Mat` for holding Jacobian $J_{\mathbf{F}}(\mathbf{x})$

call-backs for residual and Jacobian

it is worth saying more about "call-backs" set by

```
SNESSetFunction(snes,r,FormFunction,&user);
SNESSetJacobian(snes,J,J,FormJacobian,&user);
```

- the `SNES` calls `FormFunction()` when it needs the values (vector) $\mathbf{F}(\mathbf{x})$ and `FormJacobian()` when it needs values (matrix) $J_{\mathbf{F}}(\mathbf{x})$
- `FormFunction()` is code we write ourselves, to tell PETSc what the residual is
  - *required* because the `SNES` has to have *some* information about the equations
- `FormJacobian()` is code we write ourselves, to tell PETSc what the Jacobian is
  - *optional* because derivatives can be approximated by finite differences

finite-difference Jacobians

- if a `FormJacobian()` function is *not* provided through `SNESSetJacobian()` then the `SNES` cannot run unless you ask for certain behavior
- option 1: `-snes_fd` causes each entry of the Jacobian to be approximated by a finite difference:

$$\frac{\partial F_i}{\partial x_j} \approx \frac{F_i(x_0, \dots, x_j + \Delta x, \dots, x_{N-1}) - F_i(x_0, \dots, x_j, \dots, x_{N-1})}{\Delta x}$$

    - so there are N^2 such approximations needed (N^2 extra $\mathbf{F}$ evaluations) per Jacobian evaluation
- option 2: `-snes_mf` causes action of the Jacobian on a vector $\mathbf{v}$ to be approximated by a finite difference:

$$J_{\mathbf{F}}(\mathbf{x})\mathbf{v} \approx \frac{\mathbf{F}(\mathbf{x} + \epsilon\mathbf{v}) - \mathbf{F}(\mathbf{x})}{\epsilon}$$

    - N extra $\mathbf{F}$ evaluations per Jacobian-vector product
- typically $\Delta x = \epsilon \approx 10^{-8}$

example codes: `expcircle.c` and `ecjacobian.c`

- see `c/ch4/` and looks at these codes
- build and run:

```
$ make expcircle
$ ./expcircle                        # error
$ ./expcircle -snes_fd
$ ./expcircle -snes_fd -snes_monitor
$ ./expcircle -snes_fd -snes_monitor \
  -snes_rtol 1.0e-14
$ ./expcircle -snes_mf
```

- with Jacobian you can use no option:

```
$ make ecjacobian
$ ./ecjacobian
```