A first nonlinear PDE

`DMDA` + `SNES` = lots of options

Ed Bueler
Dept. of Mathematics and Statistics, UAF

8 March 2016

outline for today

Chapter 4 of book:

- recall the fixed-dimension `SNES` example, and Newton's method, from last week
- diffusion-reaction equation in one dimension:

$$-u'' - R(u) = f(x)$$

  where $R(u)$ is nonlinear
- ... is a structured-grid `DMDA` +`SNES` example
  `c/ch4/reaction.c`
- show evidence for convergence

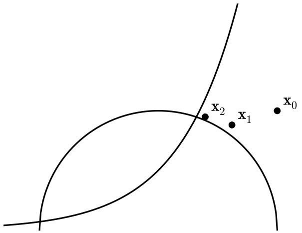
recall Newton's method (and $N = 2$ example)

- goal: find $\mathbf{x}$ so that $\mathbf{F}(\mathbf{x}) = 0$
- $J_{\mathbf{F}}(\mathbf{x})$ is the matrix of partial derivatives of $\mathbf{F}$
- Newton's method: each iteration solves a linear system and does a vector addition

$$J_{\mathbf{F}}(\mathbf{x}_k)\,\mathbf{s} = -\mathbf{F}(\mathbf{x}_k)$$
$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}$$

example: $N = 2$; to find intersection of circle and exponential:

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



Newton's method: resulting digits

- `c/ch4/ecjacobian.c` solves the example; has Jacobian
- quadratic convergence looks like this:

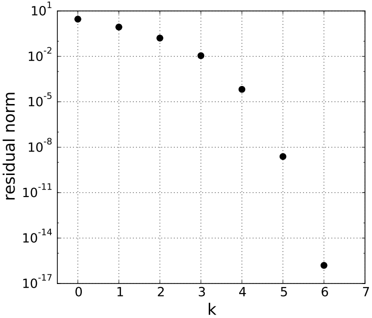
```
$ ./ecjacobian -snes_monitor -snes_rtol 1.0e-16
  0 SNES Function norm 2.874105323289e+00
  1 SNES Function norm 8.591392822370e-01
  2 SNES Function norm 1.609958166309e-01
  3 SNES Function norm 1.106891138388e-02
  4 SNES Function norm 6.618107497046e-05
  5 SNES Function norm 2.419259135755e-09
  6 SNES Function norm 0.000000000000e+00
Vec Object: 1 MPI processes
  type: seq
0.319632
0.947542
```

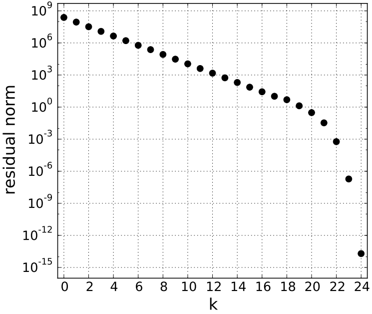
- slightly-modified version showing many digits:

```
$ ./ecdigits -snes_rtol 1.0e-16
  0:  x[0] = 1.0000000000000000,  x[1] = 1.0000000000000000
  1:  x[0] = 0.6192029220221174,  x[1] = 0.8807970779778824
  2:  x[0] = 0.3941574411297963,  x[1] = 0.9486230792698007
  3:  x[0] = 0.3251991354029324,  x[1] = 0.9481566262176070
  4:  x[0] = 0.3196650558943322,  x[1] = 0.9475469644105247
  5:  x[0] = 0.3196315386294769,  x[1] = 0.9475419149896520
  6:  x[0] = 0.3196315374042095,  x[1] = 0.9475419147967131
```

Newton's method: graph of residual

- solution $\mathbf{x}^* \approx [0.32\ 0.95]^\top$
- $\mathbf{x}_0 = [1\ 1]^\top$, closer to $\mathbf{x}^*$, gives top graph
- $\mathbf{x}_0 = [10\ 10]^\top$, far from $\mathbf{x}^*$, gives bottom graph
- once inside "good" neighborhood of $\mathbf{x}^*$, *drop doubles each iteration* on log-residual axes
  - $=$ characteristic "look" of quadratic convergence
- there is theory to support this; see Kelley (2003)





Newton's method: options and evaluations

- available options:

```
$ ./ecjacobian              # analytical Jacobian
$ ./ecjacobian -snes_fd     # Jacobian entries computed
                            #   by finite differences
$ ./ecjacobian -snes_mf     # Jacobian-vector products in
                            #   Krylov solver are computed
                            #   by finite differences
```

- ask PETSc to count evaluations with above options:

```
$ ./ecjacobian -log_view |grep SNESFunctionEval
SNESFunctionEval        6 ...
$ ./ecjacobian -snes_fd -log_view |grep SNESFunctionEval
SNESFunctionEval       21 ...
$ ./ecjacobian -snes_mf -log_view |grep SNESFunctionEval
SNESFunctionEval       21 ...
```

- …done with fixed-size systems of nonlinear equations

nonlinear diffusion-reaction equation

- $u(x)$ is the density of substance or temperature
- model (ODE) for balance of *diffusion* and *reaction* processes:
$$-u'' - R(u) = f(x) \qquad (*)$$
  - $R(u) = 0$ case is Poisson equation
- $(*)$ is steady-state of the time-dependent model (PDE)
$$w_t = w_{xx} + R(w) + f(x)$$
  - positive value of $R(w) + f(x)$ is increase of w
  - if $R(w)$ is increasing function then possible explosive reaction: $R(w) = \lambda e^w$ with $\lambda > 0$ in Bratu equation runs away at $\lambda \approx 3.5$ in 1D

particular boundary-value problem

- will solve two-point boundary value problem for ODE:

$$-u'' - R(u) = f(x), \quad u(0) = \alpha, \quad u(1) = \beta$$

- acts like an elliptic PDE BVP more than ODE IVP
  - shooting is possible ... requires Newton's method anyway and does not generalize to 2D, 3D
- example with decreasing $R(u) = -\rho\sqrt{u}$:

$$-u'' + \rho\sqrt{u} = 0$$

  - well-posed
  - exact solution known: $u(x) = M(x+1)^4$ with $M = (\rho/12)^2$
  - obtain b.c.s from exact solution: $\alpha = M$ and $\beta = 16M$.

method/plan for PETSc implementation

- discretize ODE with finite differences
  - `DMDA` manages grid in parallel
- discrete equations → residual function: $\mathbf{F}(\mathbf{u}) = 0$
  - code for residual function $\mathbf{F}(\mathbf{x})$ is a `SNES` call-back
  - Jacobian $J_{\mathbf{F}}(\mathbf{x})$, derivatives of $\mathbf{F}$, are also a `SNES` call-back
  - `SNES` does Newton's method …users don't write algorithms!
- note "universal" initial iterate for these two-point ODEs: $u_0(x) = \alpha(1 - x) + \beta x$
  - i.e. solve Poisson's equation with $f = 0$ and given b.c.s
- verify with exact solution
  - measure norm of error $\|u_k - u\|$

finite difference scheme

- N point grid x_i where $h = 1/(N-1) > 0$, $x_i = ih$ for $i = 0, 1, \dots, N-1$
- centered, $O(h^2)$ FD scheme in source-free case:

$$-\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} - R(u_i) = 0$$

- $\to$ component of residual:

$$F_i(\mathbf{u}) = -u_{i+1} + 2u_i - u_{i-1} - h^2 R(u_i)$$

  - coefficients normalized so entries of Jacobian are $O(1)$
  - sparse because F_i only depends on u_{i-1}, u_i, u_{i+1} and not all N components $u_0, \dots, u_{N-1}$

implementation

look at program `c/ch4/reaction.c`:

- a "context" `struct` holds constants:

```
typedef struct {
  double rho, M, alpha, beta;
} AppCtx;
```

- `FormFunctionLocal()` computes $F_i(\mathbf{u})$ for grid points i owned by process
- `FormJacobianLocal()` computes rows i of $J_{\mathbf{F}}(\mathbf{u})$ owned by process

implementation 2

in `main()`:

- to create the grid:

```
DMDACreate1d(COMM,DM_BOUNDARY_NONE,-9,1,1,NULL,&da);
```

- to set-up the call-backs:

```
SNESSetDM(snes,da);
DMDASNESSetFunctionLocal(da,INSERT_VALUES,
    (DMDASNESFunction)FormFunctionLocal,&user);
DMDASNESSetJacobianLocal(da,
    (DMDASNESJacobian)FormJacobianLocal,&user);
```

basic runs

- run it:

```
$ make reaction
$ ./reaction
on 9 point grid:  |u-u_exact|_inf/|u|_inf = 0.000188753
```

- refine grid:

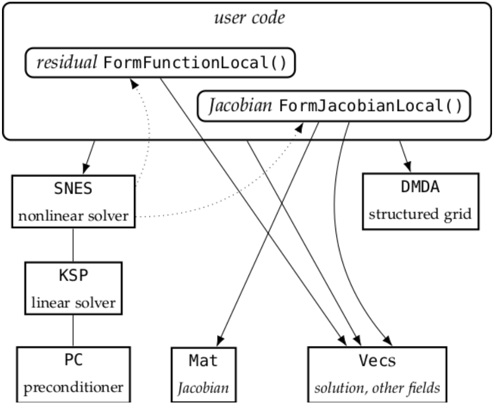
```
$ ./reaction -da_refine 4 -snes_monitor
  0 SNES Function norm 1.671129624018e-02
  1 SNES Function norm 3.609252641302e-04
  2 SNES Function norm 4.167490508953e-07
  3 SNES Function norm 4.935229190504e-13
on 129 point grid:  |u-u_exact|_inf/|u|_inf = 7.39662e-07
```

- visualize:

```
$ ./reaction -da_refine 4 -snes_monitor \
    -snes_monitor_solution draw -draw_pause 1
```

structure of `reaction.c`

- solid arrows mean "user code acts directly on"
- dotted arrows are call-backs



convergence

```
$ for N in 0 2 4 6 8 10 12 14 16; do
>   ./reaction -da_refine $N -snes_rtol 1.0e-10; done
on 9 point grid:  |u-u_exact|_inf/|u|_inf = 0.000188753
on 33 point grid:  |u-u_exact|_inf/|u|_inf = 1.1825e-05
...
on 131073 point grid:  |u-u_exact|_inf/|u|_inf = 7.05476e-13
on 524289 point grid:  |u-u_exact|_inf/|u|_inf = 6.04273e-12
```