Newton's Method: Equations

Newton's method solves a system of nonlinear equations of the form,

$$\mathbf{F}(\mathbf{u}) = 0,$$

by a sequence of steps including the linear problem

$$\mathbf{J^k}(\mathbf{u}^k)\delta\mathbf{u}^k = -\mathbf{F}(\mathbf{u}^k),$$

where,

$$\mathbf{J}_{i,j}(\mathbf{u}^k) = \frac{\partial \mathbf{F}_i}{\partial \mathbf{u}_j^k},$$

and,

$$\mathbf{u}^{k+1} = \mathbf{u}^k + \delta\mathbf{u}^k.$$

This is continued until

$$\|\mathbf{F}(\mathbf{u}^k)\| < tol_n \|\mathbf{F}(\mathbf{u}^0)\|$$

where tol_n is an input nonlinear tolerance.

Newton's Method: Basics

- Newton's method is a first-order Taylor series approximation

$$\mathbf{F}(\mathbf{u} + \delta\mathbf{u}) = 0,$$

$$\mathbf{F}(\mathbf{u}) + \delta\mathbf{u}\frac{\partial \mathbf{F}}{\partial \mathbf{u}} \approx 0$$

$$\mathbf{J}\delta\mathbf{u} = -\mathbf{F}(\mathbf{u}),$$

- Forming $\mathbf{J}$ can be challenging for many problems. Analytic or numerical evaluation ?

$$\mathbf{J}_{i,j} = \frac{\partial \mathbf{F}_i}{\partial \mathbf{u}_j},$$

- Radius of convergence can be small. This is a function of the Taylor series approximation. Damping / Globalization is often required. Choosing d ?

$$\mathbf{u}^{k+1} = \mathbf{u}^k + d\delta\mathbf{u}^k.$$

Newton Iteration: Implementation

1. Define $T^{k=1}$
2. Do k = 1,Nnewt
3. Evaluate $F(T^k)$ (if $\| F(T^k) \|_2 <$ tol then exit)
4. Evaluate $J(T^k)$
5. Solve $J(T^k)\delta T = -F(T^k)$
6. Update $T^{k+1} = T^k + \delta T$
7. End Do

Standard Newton-Krylov Methods

- Using a Krylov method to solve $\mathbf{J^k}\delta\mathbf{u}^k = -\mathbf{F}(\mathbf{u}^k)$
- Form both $\mathbf{J}$ and $\mathbf{F}(\mathbf{u})$
- Send $\mathbf{J}$ and $\mathbf{F}(\mathbf{u})$ to Krylov solver and use standard matrix vector multiply.
- Extract preconditioning directly from $\mathbf{J}$

Inexact Newton Methods

- Inexact Newton tolerance on Krylov solver for each Newton iteration limits excessive Krylov iterations. Does prevent quadratic convergence.

$$\| \mathbf{J}^k \delta \mathbf{u}^k + \mathbf{F}(\mathbf{u}^k) \|_2 < \gamma \| \mathbf{F}(\mathbf{u}^k) \|_2, \tag{1}$$

- "Under solving" in early Newton iterations can increase radius of convergence.

Jacobian-Free Newton-Krylov

- Using a Krylov method to solve $\mathbf{J}^{\mathbf{k}}\delta\mathbf{u}^k = -\mathbf{F}(\mathbf{u}^k)$
- Newton-Krylov methods only need the action of the Jacobian matrix to construct the m^{th} linear iterate of $\delta\mathbf{u}^k$

$$\delta\mathbf{u}_m^k = \beta_0\mathbf{r}_0 + \beta_1\mathbf{J}\mathbf{r}_0 + \beta_2\mathbf{J}^2\mathbf{r}_0 + \ldots + \beta_m\mathbf{J}^m\mathbf{r}_0$$

where

$$\mathbf{r}_0 = \mathbf{J}^k\delta\mathbf{u}_0^k + \mathbf{F}(\mathbf{u}^k)$$

- The Matrix-vector product required by the Krylov method can be approximated with a single function evaluation

$$\mathbf{J}\mathbf{v} \approx \frac{\mathbf{F}(\mathbf{u}+\epsilon\mathbf{v}) - \mathbf{F}(\mathbf{u})}{\epsilon}$$

The Jacobian-vector Product

- Consider the two coupled nonlinear equations $F_1(u_1, u_2) = 0,\ F_2(u_1, u_2) = 0$. The Jacobian matrix is

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial F_1}{\partial u_1} & \dfrac{\partial F_1}{\partial u_2} \\ \\ \dfrac{\partial F_2}{\partial u_1} & \dfrac{\partial F_2}{\partial u_2} \end{bmatrix}.$$

- Working backwards

$$\frac{\mathbf{F}(\mathbf{u} + \epsilon\mathbf{v}) - \mathbf{F}(\mathbf{u})}{\epsilon} = \begin{pmatrix} \dfrac{F_1(u_1+\epsilon v_1, u_2+\epsilon v_2) - F_1(u_1,u_2)}{\epsilon} \\ \\ \dfrac{F_2(u_1+\epsilon v_1, u_2+\epsilon v_2) - F_2(u_1,u_2)}{\epsilon} \end{pmatrix}.$$

The Jacobian-vector Product, cont.

- Approximating $\mathbf{F}(\mathbf{u}+\epsilon\mathbf{v})$ with a first-order Taylor series

$$\frac{\mathbf{F}(\mathbf{u}+\epsilon\mathbf{v})-\mathbf{F}(\mathbf{u})}{\epsilon} \approx \begin{pmatrix} \dfrac{F_1(u_1,u_2)+\epsilon v_1\frac{\partial F_1}{\partial u_1}+\epsilon v_2\frac{\partial F_1}{\partial u_2}-F_1(u_1,u_2)}{\epsilon} \\ \\ \dfrac{F_2(u_1,u_2)+\epsilon v_1\frac{\partial F_2}{\partial u_1}+\epsilon v_2\frac{\partial F_2}{\partial u_2}-F_2(u_1,u_2)}{\epsilon} \end{pmatrix},$$

- which simplifies to

$$\begin{pmatrix} v_1\frac{\partial F_1}{\partial u_1}+v_2\frac{\partial F_1}{\partial u_2} \\ \\ v_1\frac{\partial F_2}{\partial u_1}+v_2\frac{\partial F_2}{\partial u_2} \end{pmatrix} = \mathbf{J}v.$$

Jacobian-Free Newton-Krylov : Refs

- Standard "PDE motivated" references:
  P. N. Brown and Y. Saad, *SIAM J. Sci. Stat. Comput.*, **11**, pp. 450-481 (1990)
  Tony F. Chan and Kenneth R. Jackson, *SIAM J. Sci. Stat. Comput.*, **5**, pp. 533-542 (1984)
- Also see the monograph,
  C. T. Kelley, *Iterative Methods for Linear and Nonlinear Equations*, SIAM, Philadelphia, 1995
- Recent JFNK review article from the application perspective
  D.A. Knoll and D.E. Keyes, Jacobian-free Newton-Krylov methods: a survey of approaches and applications, *J. Comput. Phys.*, **193**, pp. 357-397 (2004)

Choosing ϵ

- Choosing ϵ for a numerical derivative of $f(x)$

$$\frac{\partial f(x)}{\partial x} \approx \frac{f(x+\epsilon) - f(x)}{\epsilon}$$

  $\epsilon = a(1.0 + |x|)$ where a is on order square root of round-off. $a \approx 1.0e-6$ to $1.0e-8$.

- Choosing ϵ for the JFNK matvec. Many algorithms work for choosing ϵ (see refs), one simple one is

$$\epsilon = \frac{1}{N||\mathbf{v}||_2} \sum_{i=1}^{N} a(1.0 + |u_i|)$$

  where $\mathbf{v}$ is the Krylov vector and N is the system dimension. Looks like an average of what would be done individually.

- Why we use GMRES

Jacobian Times a Vector: Implementation

- We need
$$\mathbf{Jv} = \frac{ResT(\mathbf{T} + \epsilon\mathbf{v}) - ResT(\mathbf{T})}{\epsilon}$$
- We have $ResT(\mathbf{T})$ and $\mathbf{v}$
- Must evaluate an ϵ, then evaluate $ResT(\mathbf{T} + \epsilon\mathbf{v})$, and finally evaluate $\mathbf{Jv}$.

Evaluating $y = Jv$, MatrixFreeJv

- Subroutine MatrixFreeJv(v,y,N)
  1. evaluate ϵ
  2. evaluate $\mathbf{Tpert} = \mathbf{T} + \epsilon\mathbf{v}$
  3. evaluate $\mathbf{w} = ResT(\mathbf{Tpert})$ (call ResTemp)
  4. Do i = 1,N
  5. $y(i) = \frac{w(i) - ResT(i)}{\epsilon}$
  6. End Do
- This routine / function provides the same service to GMRES as the function AFUN in Matlab. This is a matrix-free AFUN which allows you to do JFNK with GMRES in Matlab.

Standard Newton-Krylov vs JFNK

- Standard: Using a Krylov method to solve $\mathbf{J^k}\delta\mathbf{u}^k = -\mathbf{F}(\mathbf{u}^k)$
  - Form both $\mathbf{J}$ and $\mathbf{F}(\mathbf{u})$
  - Send $\mathbf{J}$ and $\mathbf{F}(\mathbf{u})$ to Krylov solver and use standard matrix vector multiply.
  - Extract preconditioning directly from $\mathbf{J}$
- JFNK: Using a Krylov method to solve $\mathbf{J^k}\delta\mathbf{u}^k = -\mathbf{F}(\mathbf{u}^k)$
  - Form only $\mathbf{F}(\mathbf{u})$
  - Send only $\mathbf{F}(\mathbf{u})$ to Krylov solver and use Matrix-free matrix vector multiply.
  - Extract preconditioning from something less costly than $\mathbf{J}$

Preconditioning: Concept and Choices

- In left preconditioning the Krylov method is solving the altered system
$$M^{-1}Ax = M^{-1}b$$
- What is the "meaning" of M^{-1} vs that of A^{-1} ?
- We have two choices to make
  1. Do we use the same discrete operator to form M as we are using to solve $Ax = b$?
  2. How do we approximately invert M ?
- GOAL: Reduce the number of Krylov iterations from N_1 to N_2 with a preconditioning process which cost less than $(N_1 - N_2)$ Krylov iterations.

Preconditioning JFNK (1 of 2)

- Preconditioning is the KEY for efficient application to multiphysics engineering problems.
- Using right preconditioning one solves

$$(\mathbf{J}\mathbf{M}^{-1})(\mathbf{M}\delta\mathbf{u}) = -\mathbf{F}(\mathbf{u}). \tag{1}$$

  $\mathbf{M}$ symbolically represents the preconditioning matrix (or process) and $\mathbf{M}^{-1}$ the inverse of preconditioning matrix.
- Actually realized through a two-step process. First solve

$$(\mathbf{J}\mathbf{M}^{-1})\mathbf{w} = -\mathbf{F}(\mathbf{u}), \tag{2}$$

  for $\mathbf{w}$ with the Krylov methods. Then solve for $\delta\mathbf{u}$,

$$\delta\mathbf{u} = \mathbf{M}^{-1}\mathbf{w}. \tag{3}$$

Preconditioning JFNK (2 of 2)

- Right-preconditioned matrix-free version is:

$$\mathbf{J}\mathbf{M}^{-1}\mathbf{v} \approx [\mathbf{F}(\mathbf{u} + \epsilon\mathbf{M}^{-1}\mathbf{v}) - \mathbf{F}(\mathbf{u})] \,/\, \epsilon. \tag{4}$$

- Required in step 1 of previous slide
- Actually done in two steps ($\mathbf{v}$ is given):
  1. Preconditioning: Solve (approximately) for $\mathbf{y}$ in $\mathbf{y} = \mathbf{M}^{-1}\mathbf{v}$.
  2. Perform matrix-free product $\mathbf{J}\mathbf{y} \approx [\mathbf{F}(\mathbf{u} + \epsilon\mathbf{y}) - \mathbf{F}(\mathbf{u})] \,/\, \epsilon$.
- Only the matrix elements required for the action of $\mathbf{M}^{-1}$ are formed.

Preconditioning JFNK: Options (1 of 2)

- Form $\mathbf{J}$ every Newton iteration, $\mathbf{M} \equiv \mathbf{J}$, and approximately invert $\mathbf{M}$ as preconditioning.
  - There is NO advantage to this !!
  - Should perform the same as standard Newton Krylov
- Form $\mathbf{J}$ every N Newton iterations, $\mathbf{M} \equiv \mathbf{J}$, and approximately invert $\mathbf{M}$ as preconditioning.
  - This can significantly reduce the cost of forming $\mathbf{J}$.
  - Strong Newton convergence maintained since GMRES is using most current version of $\mathbf{J}$ in matrix-vector multiply.
  - Use of same $\mathbf{J}$ for a number of Newton iterations is often referred to as Modified Newton Krylov (MNK).
  - We will consider this as a solver and a preconditioner.

Preconditioning JFNK: Options (2 of 2)

- Do not form **J**. Form **M** from a Picard linearization, and approximately invert **M** as preconditioning.
  - Here **M** can typically be formed with less effort.
  - Often the matrix from Picard linearization has better properties with respect to linear algebra.
  - A key concept in physics-based preconditioning.
- Do not form **J**. Form **M** from operators which control numerical stiffness (spread in eigenvalues) and approximately invert **M** as preconditioning.
  - Here **M** is formed from targeted physical processes such as diffusion, advection, reaction ...
  - This can often result optimal preconditioning cost-benefit.