Jacobian-Free Newton-Krylov Methods

Nick Horelik

November 26, 2012

Massachusetts Institute of Technology

What is JFNK?

"Jacobian-free Newton-Krylov (JFNK) methods are synergistic combinations of Newton-type methods for super-linearly convergent solution of nonlinear equations, and Krylov subspace methods for solving the Newton correction equations. The link between the two methods is the Jacobian-vector product, which may be probed approximately without forming and storing the elements of the true Jacobian, through a variety of means."

- D.A. Knoll, D.E. Keyes / Journal of Computational Physics 193 (2004) 357-397

Massachusetts Institute of Technology

Outline

Massachusetts Institute of Technology

Outline

Massachusetts Institute of Technology

Newton's Method

- Iterative minimization of residual equations $\mathbf{F}(\mathbf{x}) = 0$
- Taylor expansion around the current iteration state vector $\mathbf{x}^k$

$$\mathbf{F}(\mathbf{x}^{k+1}) = \mathbf{F}(\mathbf{x}^k) + \mathbb{J}(\mathbf{x}^k)\left(\mathbf{x}^{k+1} - \mathbf{x}^k\right) + \text{higher order terms}$$

- Set RHS $= 0$, iterate with a series of linear solves with $\mathbf{x}^{k+1} = \mathbf{x}^k + \delta$

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

Massachusetts Institute of Technology

The Beast: The Jacobian

- The Jacobian $\mathbb{J}$ is the derivative matrix of the residual equations $\mathbf{F}(\mathbf{x})$ w.r.t. each variable of the state vector
  - e.g. For N equations and N unknowns

$$\mathbf{x} = \{x_1, x_2, x_3, ..., x_N\}, \mathbf{F}(\mathbf{x}) = \{F_1, F_2, F_3, ..., F_N\}$$

$$J_{ij} = \frac{\partial F_i(\mathbf{x})}{\partial x_j}$$

- Expensive to calculate, expensive to store, expensive to invert - sometimes it's not even possible to calculate analytically

Massachusetts Institute of Technology

A Note on Globalization

- Newton's method normally converges quadratically to a stationary $\mathbf{x}$, but not necessarily to a global minimizer for all initial guesses
  - e.g. For $f(x) = \arctan(x) = 0$, $f'(x) = \left(1 + x^2\right)^{-1}$, an initial guess of $x = 10$ yields iterates:

$$10 \rightarrow -138 \rightarrow 2.9 \times 10^4 \rightarrow -1.5 \times 10^9 \rightarrow 9.9 \times 10^{17}$$

- Typically we use some variation of *line searches* or *trust regions* on top of Newton's method to correct for this
  - e.g.: $\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha\delta$
  - Try $\alpha = 1$, if $\mathbf{F}(\mathbf{x}^{k+1}) \not< \mathbf{F}(\mathbf{x}^k)$ then try $\alpha = \frac{1}{2}$, etc.

Massachusetts Institute of Technology

Newton-Interative/Inexact Newton/Truncated Newton Methods

- Outer nonlinear Newton iteration
  $$\mathbf{x}^{k+1} = \mathbf{x}^k + \delta$$
  - Inner linear iterative solve
    $$\mathbb{J}(\mathbf{x}^k)\delta = -\mathbf{F}(\mathbf{x}^k)$$
  - Apply globalization scheme
- We can use any method for the inner linear solve, but for our case we use Krylov Methods

Massachusetts Institute of Technology

Krylov Methods

- Reason #1 why we like Krylov methods for the Newton inner linear solve:
  - **We only require the action of the Jacobian on a vector to carry out the solve**
- This enables us to approximate the Jacobian as a finite difference
- This also facilitates matrix-free implementation of a Jacobian matrix-vector product routine

What Are Krylov Methods?

- Krylov subspace for the jth Kyrlov iteration

$$\mathcal{K}_j = \text{span}\left\{\mathbf{r}_0, \mathbb{A}\mathbf{r}_0, \mathbb{A}^2\mathbf{r}_0, ..., \mathbb{A}^{j-1}\mathbf{r}_0\right\}$$

  for the initial linear residual $\mathbf{r}_0 = \mathbf{b} - \mathbb{A}\mathbf{x}_0$

- $\mathbf{x}_j$ is always drawn from $\mathcal{K}_k$:

$$\mathbf{x}_j = \mathbf{x}_0 + \sum_{i=0}^{j-1} \beta_i \mathbb{A}^i \mathbf{r}_0$$

  where the β_i scalars are found in a least-squares process to minimize the residual $\mathbf{b} - \mathbb{A}\mathbf{x}_j$

Massachusetts Institute of Technology

GMRES

$$\text{minimize}_{\mathbf{x}\in\mathbf{x}_0+\mathcal{K}}||\mathbf{b}-\mathbb{A}\mathbf{x}||_2$$

If we find an orthonormal projector $\mathbf{V}_k$ onto $\mathcal{K}_k$

$$\mathbf{x}-\mathbf{x}_0=\mathbf{V}_k\mathbf{y}$$

then

$$\text{minimize}_{\mathbf{r}\in\mathbb{R}^k}||\mathbf{r}_0-\mathbb{A}\mathbf{V}_k\mathbf{y}||_2$$

i.e. a standrad least-squares minimization that can be solved with QR factorization, here Gram-Schmidt (Arnoldi) process for finding the orthonormal basis

Massachusetts Institute of Technology

JFNK: Finite Differencing the Jacobian

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

For example: $F_1(x_1, x_2) = 0.$ $F_1(x_1, x_2) = 0$

$$\begin{aligned}
\frac{\mathbf{F}(\mathbf{x}+\epsilon\mathbf{v}) - \mathbf{F}(\mathbf{x})}{\epsilon} &= \begin{bmatrix} \frac{F_1(x_1+\epsilon v_1, x_2+\epsilon v_2) - F_1(x_1,x_2)}{\epsilon} \\ \frac{F_2(x_1+\epsilon v_1, x_2+\epsilon v_2) - F_2(x_1,x_2)}{\epsilon} \end{bmatrix} \\
&\approx \begin{bmatrix} \frac{F_1(x_1,x_2)+\epsilon v \frac{\partial F_1}{\partial x_1} + \epsilon v \frac{\partial F_1}{\partial x_2} - F_1(x_1,x_2)}{\epsilon} \\ \frac{F_2(x_1,x_2)+\epsilon v \frac{\partial F_2}{\partial x_1} + \epsilon v \frac{\partial F_2}{\partial x_2} - F_2(x_1,x_2)}{\epsilon} \end{bmatrix} \\
&= \begin{bmatrix} v\frac{\partial F_1}{\partial x_1} + v\frac{\partial F_1}{\partial x_2} \\ v\frac{\partial F_2}{\partial x_1} + v\frac{\partial F_2}{\partial x_2} \end{bmatrix}
\end{aligned}$$

Massachusetts Institute of Technology

Choosing The Pertubation Parameter

- Typical choice "average ϵ"

$$\epsilon = \frac{1}{N||\mathbf{v}||_2} \sum_{i=1}^{N} h|x_i| + h$$

where h is chosen on the order of machine roundoff

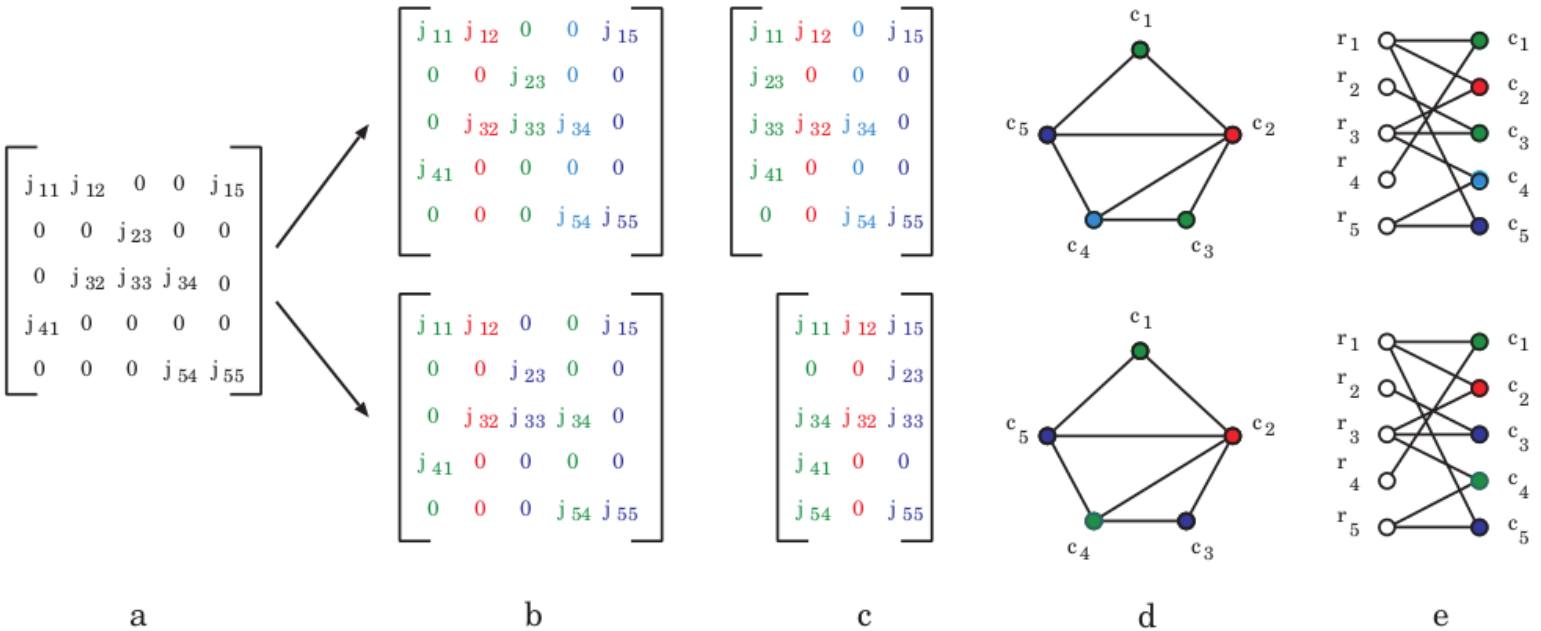
FD Works Well With Krylov Methods

- Reason #2 why we like Krylov methods
  - **The finite difference approximation of the Jacobian does not affect the performance of the Krylov solver**
- Theorem in C.T. Kelly's book "Iterative Methods for Linear and Nonlinear Equations" section 6.2.1
  - In the case where h is $\sqrt{\epsilon_{mach}}$, we maintain superlinear convergence of the Newton scheme while using Krylov methods

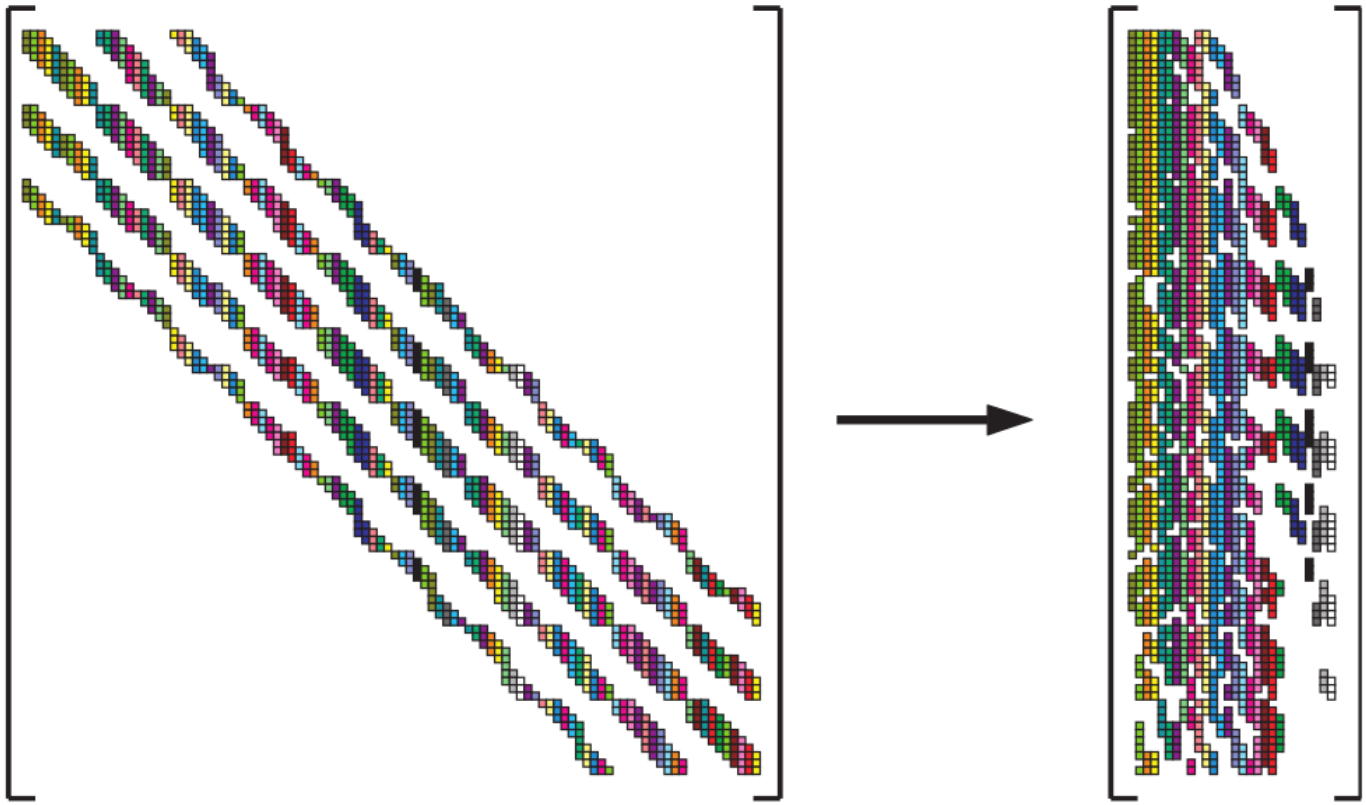
Massachusetts Institute of Technology

Implementing Jacobian Finite Differencing: Coloring

- A. H. Gebremedhin, F. Manne, AND A. Pothen "What Color Is Your Jacobian? Graph Coloring for Computing Derivatives" *Society for Industrial and Applied Mathematics*, 2005. Vol. 47, No. 4, pp. 629-705



Jacobian Coloring



Preconditioning

- Needed to reduce the number of GMRES iterations

Right-preconditioning: two step process

$$\mathbb{J}(\mathbf{x})\mathbb{P}^{-1}\mathbb{P}\delta = -\mathbf{F}(\mathbf{x})$$

$$\mathbb{J}(\mathbf{x})\mathbb{P}^{-1}\mathbf{w} = -\mathbf{F}(\mathbf{x})$$

$$\delta = \mathbb{P}^{-1}\mathbf{w}$$

e.g. LU

$$\mathbb{R} = \mathbb{L}\mathbb{U} - \mathbb{A}$$

$$\mathbb{P} = \mathbb{L}\mathbb{U}$$

Massachusetts Institute of Technology

Performance Notes

- Matrix-Free implementation: compute Jacobian on-the-fly, never store it
- Good preconditioning of the linear solve is necessary!
- Jacobian and Preconditioner Lag
  - Chord method: use only the initial Jacobian for all Newton Iterations
  - Shamanskii method: periodically recalculate the Jacobian

Outline

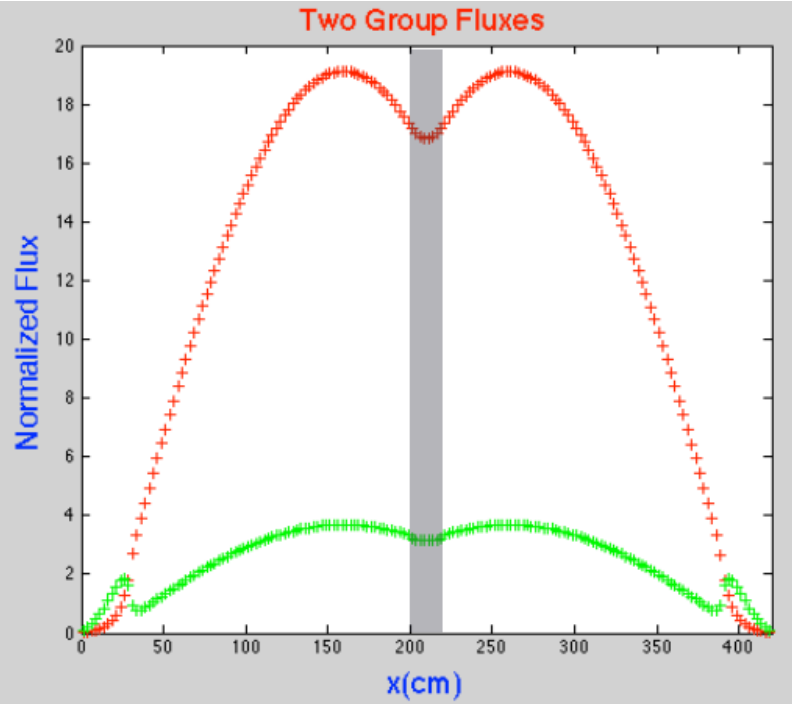
Massachusetts Institute of Technology

PETSc!

- PETSc provides a powerful interface for fine-tuned control: SNES
- SNESCreate $\rightarrow$ SNESGetKSP $\rightarrow$ KSPGetPC
  - SNES: ls, tr, ngmres, nrichardson, ...
  - KSP: gmres, cg, cgnr, richardson, ...
  - PC: jacobi, lu, ilu, icc, none, ...
- Matrix-free MatShell operations are fully supported (need $\mathbb{J}^T\mathbf{v}$ for some methods)

Massachusetts Institute of Technology

Steady State Diffusion



$$\mathbf{F}(\mathbf{x}) = \begin{bmatrix} \mathbb{M}\mathbf{\Phi} - \lambda\mathbb{F}\mathbf{\Phi} \\ -\frac{1}{2}\mathbf{\Phi}^T\mathbf{\Phi} + \frac{1}{2} \end{bmatrix}, \mathbf{x} = \begin{bmatrix} \mathbf{\Phi} \\ \lambda \end{bmatrix}$$

$$\mathbb{J} = \begin{bmatrix} \mathbb{M} - \lambda\mathbb{F} & -\lambda\mathbb{F} \\ -\mathbf{\Phi}^T & 0 \end{bmatrix}$$

Massachusetts Institute of Technology

Implementation

- Use the same 1D diffusion matrices as before
- Build PETSc routine to compute **F**

```
call MatCreateSeqAIJ(comm,N+1,N+1,Jnz,PETSC_NULL_INTEGER,mat_J,ierr)
call VecCreateSeq(comm,N+1,vec_x,ierr)
call VecCreateSeq(comm,N+1,vec_r,ierr)
call SNESSetFunction(snes,vec_r,FormFunction,ctx,ierr)
call SNESSetJacobian(snes,mat_J,mat_J,FormJacobian,ctx,ierr)
call SNESSetFromOptions(snes,ierr)
call SNESSolve(snes, PETSC_NULL, vec_x, ierr)
```

Massachusetts Institute of Technology

Implementation

Matrix-Free

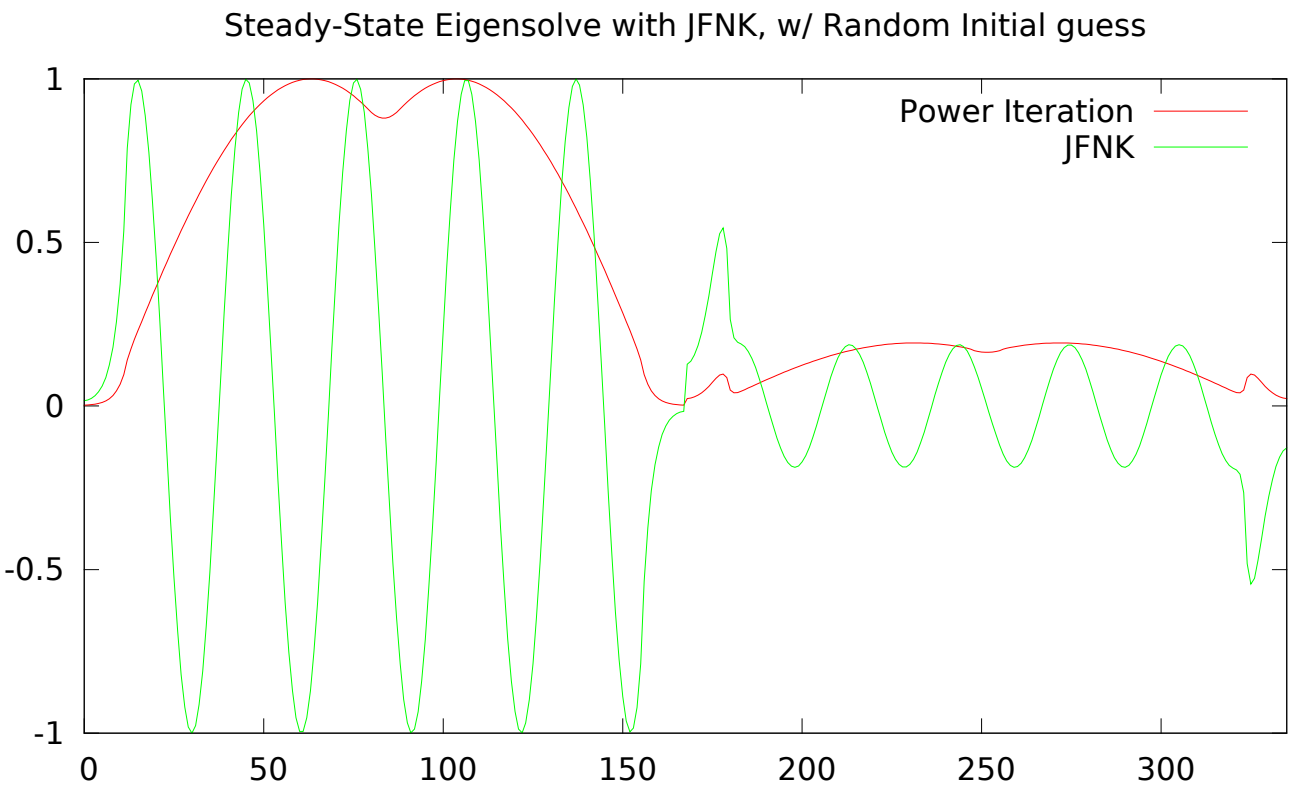
```
call MatCreateSNESMF(snes,mat_JmF,ierr)
call SNESGetKSP(snes,ksp,ierr)
call KSPGetPC(ksp,pc,ierr)
call PCSetType(pc,PCSHELL,ierr)
call PCShellSetApply(pc,MatrixFreePreconditioner,ierr)
```

Finite Difference Coloring

```
call FormJacobian(snes,x,mat_J,mat_J,flag,ctx,ierr)
call MatGetColoring(mat_J,MATCOLORINGSL,iscoloring,ierr)
call MatFDColoringCreate(mat_J,iscoloring,fdcoloring,ierr)
call MatFDColoringSetFunction(fdcoloring,jfnk_func_f,jfnk_ctx,ierr)
call MatFDColoringSetFromOptions(fdcoloring,ierr)
call SNESSetJacobian(snes,mat_J,mat_J, &
&                    SNESDefaultComputeJacobianColor,fdcoloring,ierr)
call SNESSolve(snes, PETSC_NULL, vec_x, ierr)
```

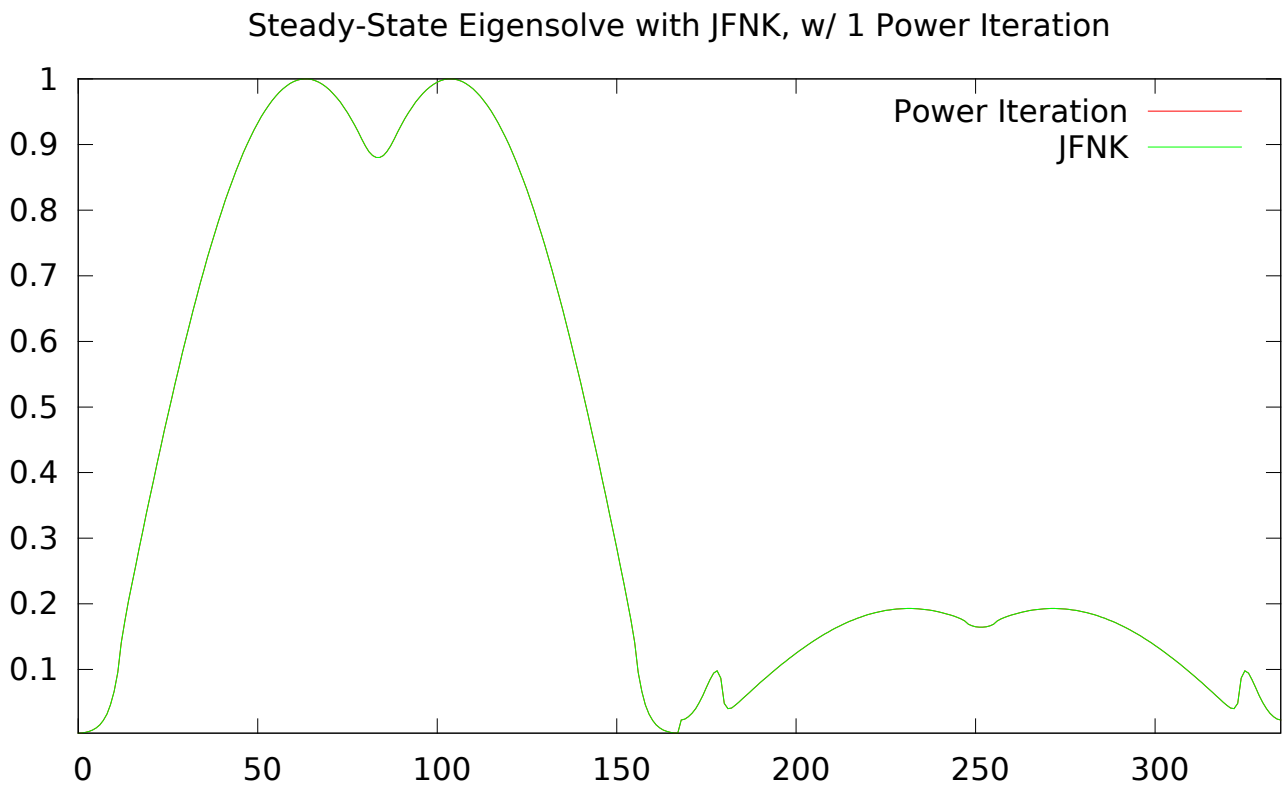
Massachusetts Institute of Technology

Random Initial Flux Guess



Massachusetts Institute of Technology

Random Guess, with 1 Power Iteration



Massachusetts Institute of Technology

Timing Results

- 1cm mesh, residual converged to 10^{-10}

Case	Outer Iters	GMRES Iters	Time (s)
Power Iteration	1821	2	0.229
Analytic $\mathbb{J}$, ilu(0)	6	$\{96, 60, 34, 34, 40, 85\}$	3.9×10^{-2}
Slow FD $\mathbb{J}$, ilu(0)	6	$\{94, 62, 40, 38, 57, 102\}$	0.275
Colored FD $\mathbb{J}$, ilu(0)	5	$\{124, 59, 38, 38, 56\}$	0.173
Colored FD $\mathbb{J}$, ilu(20)	5	$\{12, 10, 10, 10, 10\}$	0.160
Colored FD $\mathbb{J}$, lu	5	1	0.154

Massachusetts Institute of Technology

Transient Diffusion

Now λ is supplied as a constant (k_{crit})

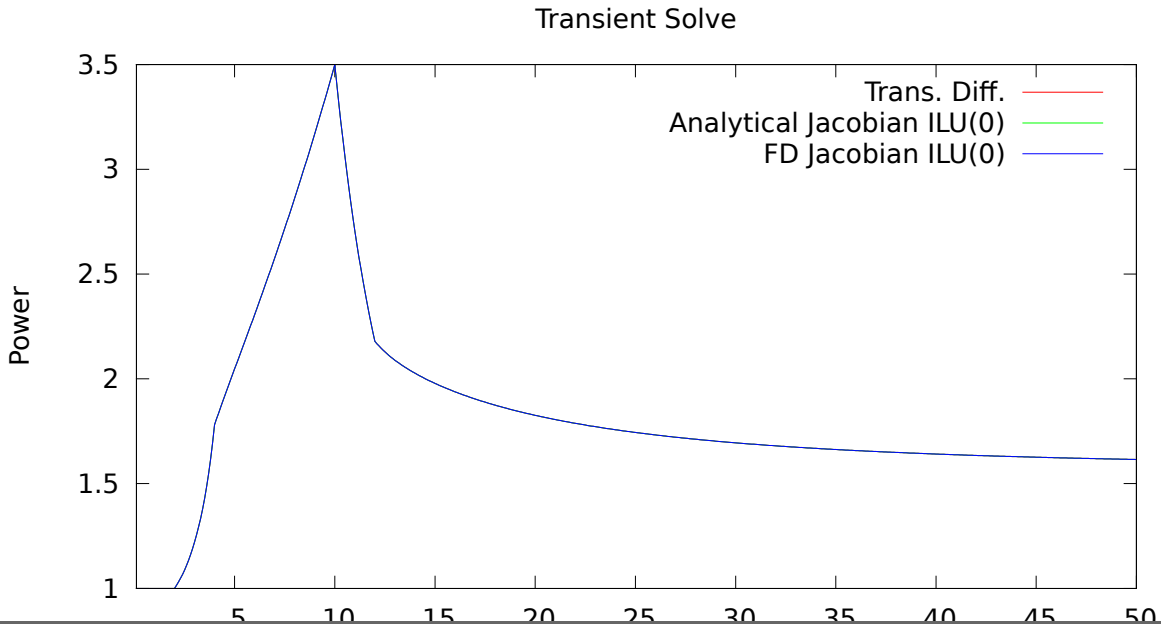
$$\mathbf{F}(\mathbf{x}) = \begin{bmatrix} \boldsymbol{\Phi}^{n+1} - \boldsymbol{\Phi}^{n} + v\Delta t\left(\mathbb{M}\boldsymbol{\Phi}^{n+1} - (1-\beta)\lambda\mathbb{F}\boldsymbol{\Phi}^{n+1} - \sum_i \lambda_i \mathbf{c}_i^{n+1}\right) \\ \mathbf{c}_1^{n+1} - \mathbf{c}_1^{n} + \Delta t\left(\lambda_1 \mathbf{c}_1^{n+1} - \beta_1 \lambda \mathbb{F}\boldsymbol{\Phi}^{n+1}\right) \\ \mathbf{c}_2^{n+1} - \mathbf{c}_2^{n} + \Delta t\left(\lambda_2 \mathbf{c}_2^{n+1} - \beta_2 \lambda \mathbb{F}\boldsymbol{\Phi}^{n+1}\right) \\ \cdots \\ \mathbf{c}_8^{n+1} - \mathbf{c}_8^{n} + \Delta t\left(\lambda_8 \mathbf{c}_8^{n+1} - \beta_8 \lambda \mathbb{F}\boldsymbol{\Phi}^{n+1}\right) \end{bmatrix}$$

$$\mathbf{x}^T = \left[\boldsymbol{\Phi}^{n+1}, \mathbf{c}_1^{n+1}, \mathbf{c}_1^{n+1}, \ldots, \mathbf{c}_8^{n+1}\right]$$

Massachusetts Institute of Technology

Transient Diffusion

$$\mathbb{J} = \begin{bmatrix} 1 + v\Delta t\,(\mathbb{M} - (1-\beta)\lambda\mathbb{F}) & \text{diag}\{-v\Delta t\lambda_1\} & \text{diag}\{-v\Delta t\lambda_2\} & \dots & \text{diag}\{-v\Delta t\lambda_8\} \\ -\Delta t\beta_1\lambda\mathbb{F} & \text{diag}\{1+\Delta t\lambda_1\} & & & \\ -\Delta t\beta_2\lambda\mathbb{F} & & \text{diag}\{1+\Delta t\lambda_2\} & & \\ \dots & & & \dots & \\ -\Delta t\beta_8\lambda\mathbb{F} & & & & \text{diag}\{1+\Delta t\lambda_8\} \end{bmatrix}$$



Massachusetts Institute of Technology

Timing Results

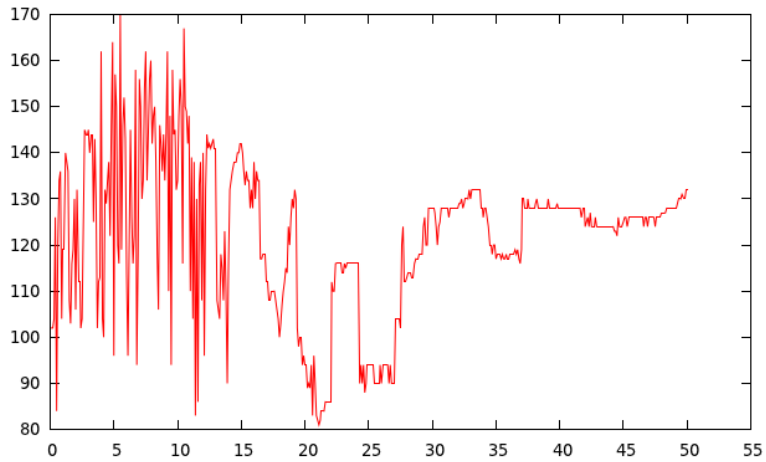
- 50s on 10cm mesh, 0.1s Δt, residual converged to 10^{-10}
- Requires 2 or 3 Newton iterations per timestep
- ILU(0) yields approx 28 GMRES iterations

Case	**Time (s)**
Transient diffusion ilu(0)	0.632
Analytical Jacobian ilu(0)	2.691
Colored FD Jacobian ilu(0)	3.612
Transient Diffusion lu	0.271
Analytical Jacobian lu	1.135
Colored FD Jacobian lu	2.002
Slow FD Jacobian lu	90.9
Transient Diffusion ilu(0) 1cm mesh	5.845
Analytical Jacobian ilu(0) 1cm mesh	64.71

Massachusetts Institute of Technology

Timing Results

Operation	Time (s)
MatMult	3.82E-001
MatSolve	5.41E-001
MatFDColorApply	2.01E+000
MatFDColorFunc	1.60E+000
VecMDot	2.31E-001
VecAXPY	2.06E-001
VecMAXPY	3.72E-001
KSPGMRESOrthog	5.97E-001
KSPSolve	1.75E+000
PCApply	5.50E-001
SNESSolve	3.95E+000
SNESLineSearch	1.55E-001
SNESFunctionEval	1.77E-001
SNESJacobianEval	2.01E+000



Massachusetts Institute of Technology

Outline

Massachusetts Institute of Technology

JFNK Overview

- Find Jacobian coloring
- Outer nonlinear Newton iteration
  $$\mathbf{x}^{k+1} = \mathbf{x}^k + \delta$$
  - Inner linear iterative solve
    - Evaluate Jacobian finite difference using coloring
    - Calculate a precoditioner (or use a "stale" one)
    - Solve linear system $\mathbb{J}(\mathbf{x}^k)\delta = -\mathbf{F}(\mathbf{x}^k)$
  - Apply globalization scheme
- Key to JFNK: only the action of the Jacobian matrix-vector product is required for Krylov methods, which allows us to use the finite difference approximation

Massachusetts Institute of Technology

Points to Remember

- Pay attention to the choice of the Krylov preconditioner
- Must color matrices to take advantage of sparsity
- *Always* monitor iteration numbers, convergence reasons
  - -snes_monitor, -ksp_monitor
  - -snes_converged_reason, -ksp_converged_reason
- Play with preconditioner and Jacobian lags
- Required reading: D.A. Knoll, D.E. Keyes / Journal of Computational Physics 193 (2004) 357-397

Massachusetts Institute of Technology