



## 9. ITERATIVE METHODS FOR NON-LINEAR SYSTEMS

### 9.1.1 Remember: what we saw during the last lesson

**Quadrature** using Gauss and Gauss-Konrod rules

**Polynomial** equations solved using `roots()`

**Methods** for non-linear equations: bisection, secant, Newton-Raphson, iterations.

### 9.1.2 Overview: what you will learn today

**Sparse linear systems** Jacobi, Gauss-Seidel

**Non-linear systems** fixed point, Picard and Newton iterations

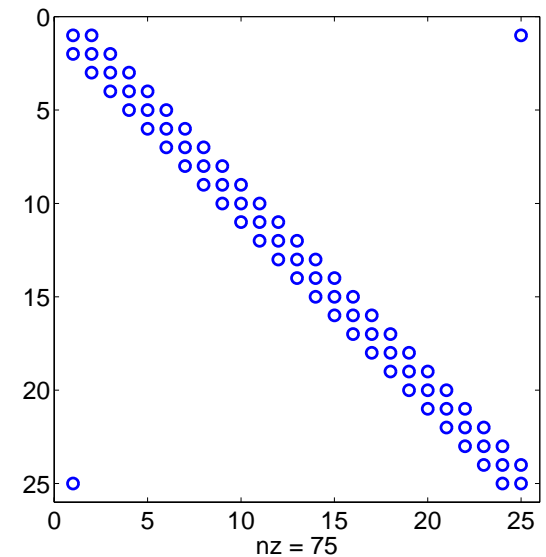
**Non-linear** interpolation, fitting

## 9.2 Iterative methods for sparse linear systems (NAM 6.7, H 11.5.2)

**Sparse matrices** have most of the elements equal to zero

In Matlab use the commands `sparse()`, `spdiags()`, `spy()`. For example, use a periodic spline interpolation through  $n = 25$  points so that  $b_k = \exp(\cos(2\pi k/n))$

```
>> n=25; h=2*pi/n; x=0:h:2*pi-h; e=h*ones(n,1);
>> b=(1:n)'; b=exp(cos(2*pi*b/n));           % RHS vector
>> % --- Sparse matrix
>> A=spdiags([e 4.*e e],-1:1,n,n);
>> A(1,n)=h; A(n,1)=h; spy(A);               % periodicity
>> x=A\b;                                     % sparse solver
>> % ---- Alt: full matrix
>> AA=diag(e(1:n-1),-1) +diag(4*e) +diag(e(1:n-1),1);
>> AA(1,n)=h; AA(n,1)=h;                    % periodicity
>> x=AA\b;                                    % LU factor
```



Only non-zero elements should be used in the sparse operations; unfortunately Gaussian elimination does not preserve sparsity  $\Rightarrow$  motivation for iterative methods!

**Split** the matrix into diagonal (**D**) and strictly lower (**E**) and upper (**F**) triangular parts  
Solve the linear system iteratively by rewriting

$$(\mathbf{E} + \mathbf{D} + \mathbf{F})\mathbf{x} = \mathbf{Ax} = \mathbf{b}$$

$$\mathbf{D}\mathbf{x}_{n+1} = \mathbf{b} - (\mathbf{E} + \mathbf{F})\mathbf{x}_n \quad \text{Jacobi iteration (D is diagonal)}$$

$$(\mathbf{E} + \mathbf{D})\mathbf{x}_{n+1} = \mathbf{b} - \mathbf{F}\mathbf{x}_n \quad \text{Gauss-Seidel (E + D is lower triangular)}$$

More advanced methods exist: type `help cgs, gmres, qmr`

## Example Compare direct LU, with iterative Gauss-Seidel, Jacobi and other solvers.

```
function x=Jacobi(A,b,tol) % --- Jacobi
n=length(b); x=zeros(n,1); dxnorm=1;
while dxnorm>tol
    xold=x;
    x(1)=(b(1)-A(1,2)*xold(2)-A(1,n)*xold(n))/A(1,1);
    for i=2:n-1
        x(i)=(b(i)-A(i,i+1)*xold(i+1)-A(i,i-1)*xold(i-1))/A(i,i);
    end
    x(n)=(b(n)-A(n,1)*xold(1)-A(n,n-1)*xold(n-1))/A(n,n);
    dxnorm=norm(x-xold,inf);
end
return
```

```
function x=GS(A,b,tol) % --- Gauss-Seidel
n=length(b); x=zeros(n,1); dxnorm=1;
while dxnorm>tol
    xold=x;
    x(1)=(b(1)-A(1,2)*xold(2)-A(1,n)*x(n))/A(1,1);
    for i=2:n-1
        x(i)=(b(i)-A(i,i+1)*xold(i+1)-A(i,i-1)*x(i-1))/A(i,i);
    end
    x(n)=(b(n)-A(n,1)*xold(1)-A(n,n-1)*x(n-1))/A(n,n);
    dxnorm=norm(x-xold,inf);
end
return
```

```
tic; x=AA\b; toc; % LU factorization
tic; x=Jacobi(A,b,tol); toc; % Jacobi
tic; x=GS(A,b,tol); toc; % Gauss-Seidel
tic; x=gmres(A,b,20,tol); toc; % GMRES(20)
tic; x=A\b; % Matlab sparse
```

% Time (sec) required to solve the sparse linear system

N	LU	Jacobi	GS	GMRES(20)	Matlab
250	0.0688	0.2353	0.1360	0.0089	0.0048
500	0.7930	0.4844	0.2866	0.0127	0.0061
1000	6.4487	1.0100	0.5772	0.0173	0.0141
2000	50.0698	2.0770	1.2275	0.1159	0.0296

The time for the LU factorization increases as  $N^3$ ; iterative solvers scaling with  $N$ . Note that you **cannot use** `tridia()` directly to solve this sparse system, because of the periodicity points.

### 9.3 Iterative methods for non-linear systems (NAM 6.8, H 5.6)

**Problem:** solve the non-linear system  $f(x) = 0$  in the neighborhood  $x_1 \approx 1, x_2 \approx 0.5$

$$\begin{aligned}2x_1 + x_2 - 2 - x_1x_2/2 &= 0 \\x_1 + 2x_2 - 1.5 - \cos x_2/2 &= 0\end{aligned}$$

**Fixed point iteration** rewrite the system in the general form

$$\mathbf{x}^{\text{new}} = \mathbf{g}(\mathbf{x}^{\text{old}})$$

$$\begin{aligned}x_1 &= (2 + x_1x_2/2 - x_2)/2 \\x_2 &= (1.5 + \cos x_2/2 - x_1)/2\end{aligned}$$

Iterate from one equation to the next; it takes 7 loops for the system to converge to  $x_1 = 0.8431, x_2 = 0.5426$  with a 4 digits accuracy. The method is generally slow and converges only if the Jacobian matrix  $G_{ij} = \partial g_i / \partial x_j$  satisfies  $\|G\| < 1$ .

**Piccard iteration** linear terms to the left / non-linear to the right

$$\mathbf{Ax}^{\text{new}} = \mathbf{b}(\mathbf{x}^{\text{old}})$$

$$\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2 + x_1x_2/2 \\ 1.5 + \cos x_2/2 \end{pmatrix}$$

The linear part of the system is solved using standard methods and the non-linear part is successively updated with newly computed values. The method converges if  $\|A^{-1}B\| < 1$  where  $B_{ij} = \partial b_i / \partial x_j$ .

**Newton's method** can be generalized to

$$\mathbf{x}^{\text{new}} = \mathbf{x}^{\text{old}} + d\mathbf{x}, \quad \mathbf{J}d\mathbf{x} = -\mathbf{f}(\mathbf{x}^{\text{old}})$$

$$\mathbf{J} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{pmatrix} = \begin{pmatrix} 2 - x_2/2 & 1 - x_1/2 \\ 1 & 2 + \sin x_2/2 \end{pmatrix}$$

where the Jacobian matrix now plays the role of what previously was a derivative.

```
>> x=[1 0.5]'; iter=0; dxnorm=1;
>> while dxnorm>0.0005 & iter<10
>>   f=[2*x(1)+x(2)-0.5*x(1)*x(2)-2 ;
>>     x(1)+2*x(2)-0.5*cos(x(2))-1.5];
>>   J=[2-x(2)/2    1-x(1)/2
>>      1          2+sin(x(2))/2];
>>   dx=-J\f; x=x+dx
>>   dxnorm=norm(dx,inf), iter=iter+1
>> end
```

Newton's method achieves also here a quadratic convergence, which is visible in the norm of the correction term  $d\mathbf{x} = \text{dxnorm} = 0.1548, 0.0021, 5.3762e-07$ . It is possible to approximate the Jacobian matrix using finite differences

$$\frac{\partial f_i}{\partial x_j} \approx \frac{f(x_1, x_2, \dots, x_j + \delta x, \dots, x_n) - f(x_1, x_2, \dots, x_j, \dots, x_n)}{\delta x}$$

but this additional evaluation of functions get prohibitive for large systems. Use **Broyden's method** instead for a generalization of the secant method (H 5.6.3).

**Example:** interpolating harmonic function  $y(t) = a + b \sin \omega(t - t_0)$

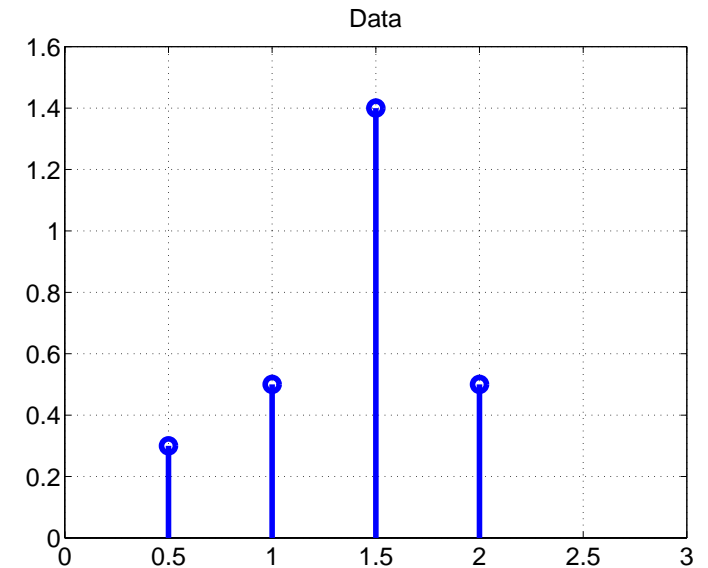
Determine the coefficients of the interpolating function that goes through the following coordinates  $t = 0.5, 1.0, 1.5, 2.0$ , with  $y = 0.3, 0.5, 1.4, 0.5$ .

$$\begin{cases} a + b \sin \omega(0.5 - t_0) = 0.3 \\ a + b \sin \omega(1.0 - t_0) = 0.5 \\ a + b \sin \omega(1.5 - t_0) = 1.4 \\ a + b \sin \omega(2.0 - t_0) = 0.5 \end{cases}$$

To apply Newton's method, we need first to calculate the Jacobian  $J_{ij} = \partial y_i / \partial x_j$ , where  $x_j$  are here the parameters  $a, b, \omega, t_0$

$$\begin{aligned} \frac{\partial y_i}{\partial a} &= 1 & \frac{\partial y_i}{\partial b} &= \sin \omega(t_i - t_0) \\ \frac{\partial y_i}{\partial \omega} &= b(t_i - t_0) \cos \omega(t_i - t_0) & \frac{\partial y_i}{\partial t_0} &= -b\omega \cos \omega(t_i - t_0) \end{aligned}$$

The starting parameter can be guessed from the plot, with  $a \approx 0.7$  (mean value),  $b \approx 0.7$  (mean to max value),  $\omega = 2\pi/T \approx 2$  (one peak) and  $t_0 \approx 1.2$  ( $\sin x \approx a$  for  $x \approx 1.2$ )



In Matlab, the solution is obtained using the Newton method:

```
t=[0.5 1 1.5 2]'; y=[0.3 0.5 1.4 0.5]';
subplot(1,2,1), stem(t,y), axis([0 3 0 1.6])
title('Data'), grid, drawnow
a=0.7, b=0.7, w=pi, t0=1.2 % First solution
% a=0.7, b=0.7, w=2*3.94, t0=0.55 % Second solution
% a=0.7, b=0.7, w=2*3.94, t0=0.6 % Third solution
c=[a b w t0]';
iter=0; dcnorm=1;
while dcnorm>0.5e-5 & iter<10
    u=w*(t-t0); f=a+b*sin(u) - y;
    J=[ones(4,1) sin(u) b*(t-t0).*cos(u) -w*b*cos(u)];
    if iter==1, J, end % print first Jacobian
    dc=-J\f; dcnorm=norm(dc,inf)
    c=c+dc; iter=iter+1;
    a=c(1); b=c(2); w=c(3); t0=c(4);
end, c, iter
tt=(0:0.05:3)'; Ft=a+b*sin(w*(tt-t0));
subplot(1,2,2), plot(t,y,'o', tt,Ft), axis([0 3 0 1.6])
```

A different short wavelength solution is obtained with different stating parameters:

