

1. Jacobi and Gauss-Seidel

- a. Suppose that $A \in \mathbb{R}^{n \times n}$ is strictly diagonally dominant. Prove that the Jacobi iteration matrix T_J corresponding to A has the property that $\|T_J\|_\infty < 1$.
- b. Let the n -by- n matrix $A = D - L - U$, where L is lower triangular with zeros on its diagonal, U is upper triangular with zeros on its diagonal, and D is diagonal. If A is symmetric positive definite, show that the Gauss-Seidel iteration matrix T_G corresponding to A can be written $T_G = I - (D - L)^{-1}A$.

2. Successive over-relaxation (SOR(ω))

- a. Consider the linear system

$$\begin{bmatrix} 3 & -1 & -1 & 0 & 0 \\ -1 & 4 & -1 & -1 & 0 \\ -1 & -1 & 5 & -1 & -1 \\ 0 & -1 & -1 & 4 & -1 \\ 0 & 0 & -1 & -1 & 3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \\ v \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{bmatrix}.$$

Explain why the SOR(ω) method will converge to the solution of this linear system for any initial starting guess. (Note: it is not sufficient to simply run SOR(ω) on this system to see if it converges, you must apply some theorem.)

- b. Write a computer program (using MATLAB preferably) for solving a linear system $Ax = b$ using SOR(ω). Your program should take as input A , b , ω , and a stopping criterion ϵ . The output should be an approximate solution to the linear system. Turn in a listing of your program.
- c. For $b = [3 \ -5 \ 5 \ -5 \ 3]^T$ the solution to the linear system is $x = [1 \ -1 \ 1 \ -1 \ 1]^T$. Using your SOR(ω) program from part b with the initial guess $x^{(0)} = 0$, solve the linear system for this b using various values of ω . Consider the system “solved” when the approximate solution $x^{(k+1)}$ satisfies: $\|Ax^{(k+1)} - b\|_\infty < 10^{-10}$ (i.e. the residual is less than 10^{-10}). Approximate the optimal ω for this system (e.g. trial and error is sufficient). Compare the number of iterations it takes to solve the system using the optimal ω with the number it takes for various other values, including $\omega = 1$ (i.e. the Gauss-Seidel case).

3. Conjugate Gradient: The following MATLAB function (which is available on the course webpage) implements the conjugate gradient algorithm as described by Bradie on p. 241:

```
% Solves the symmetric positive definite linear system Ax=b
% using the conjugate gradient algorithm.
% Input:      A - the matrix (symmetric positive definite)
%            b - the RHS of the system to solve
%            x - the initial guess
%            tol - convergence tolerance
% Output:    x - the solution
%            iter - the number of iterations
```

```

function [x,iter] = conjgrad(A,b,x,tol)

n = size(A,1);           % Determine the dimension of the matrix
x = x(:);               % Ensure x and b are column vectors
b = b(:);

r = A*x - b;           % Find residual
d = -r;                % First direction is steepest descent

for m = 1:n
    lambda = -(d'*r)/(d'*A*d); % How far to move x0
    x = x + lambda*d;      % Move x appropriately
    r = A*x - b;          % Compute the residual
    if sqrt(r'*r) < tol   % If the residual is small enough...
        break;           % exit the for loop and return the current x
    end
    alpha = r'*A*d/(d'*A*d); % Value for A-conjugate vectors.
    d = -r + alpha*d;     % Optimal direction to step
end
iter = m;              % How many times the loop was run

```

- a. The code above is not as efficient as it could be. For example, inside the loop it computes 3 matrix–vector products, 4 vector–vector products, and 3 vector–vector additions. Bradie, pp. 241–243, describes how to make this function more efficient so that inside the loop only 1 matrix–vector product, 2 vector–vector products, and 3 vector–vector additions are performed (along with scalar–vector multiplications). Your goal is to change the above function according to the more efficient implementation described by Bradie.
 - b. Use your modified function to solve the linear system from problem 2c with the initial guess $x^{(0)} = 0$. Report the number of iterations required to solve system with a tolerance of 10^{-10} .
 - c. Bradie §3.9 problem 11.
4. The almost universally used algorithm to compute \sqrt{a} is the recursion

$$x_{n+1} = \frac{1}{2} \left(x_n + \frac{a}{x_n} \right), \quad (1)$$

easily obtained by means of Newton’s method.

- a. Assume you are working with a very simple processor that only supports addition, subtraction, multiplication and halving (a subtraction of one in the exponent of a base-2 number), but not a general divide. Devise a fast algorithm for this processor to directly approximate $\frac{1}{\sqrt{a}}$ (it then only remains to multiply with a to get \sqrt{a}).

Other approaches than Newton’s method are available for generating fast iterative methods for computing \sqrt{a} . One approach starts by noting that

$$\sqrt{a} = x(1-r)^{-\frac{1}{2}} \quad \text{where } r = 1 - \frac{x^2}{a}$$

is an identity for any (positive) value of x . Hence the iteration

$$x_{n+1} = x_n(1-r)^{-\frac{1}{2}} \quad \text{where } r = 1 - \frac{x_n^2}{a}$$

will converge in one step to \sqrt{a} . However, this iteration is useless since it requires the computation of a square root (we could have just as well computed \sqrt{a} directly!).

To make this iteration useful, we note that r becomes small if x_n is a good approximation to \sqrt{a} , which suggests replacing $(1-r)^{-\frac{1}{2}}$ by a truncated Taylor expansion about $r = 0$. By including different numbers of terms in the Taylor expansion, we get iterative methods of arbitrary order of convergence.

b. From the preceding discussion, derive the quadratically convergent recursion

$$x_{n+1} = \frac{x_n}{2} \left(3 - \frac{x_n^2}{a} \right). \quad (2)$$

Explain why it is quadratically convergent. Print out a table of successive iterates for $a = 2$ and $x_0 = 1$ and note how the number of correct digits increases with n .

5. Cubically Convergent Newton's Method

(a) Generalize the Taylor series argument presented in class that led to Newton's method so that it instead leads to the cubically convergent iteration

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} - \frac{[f(x_n)]^2 f''(x_n)}{2[f'(x_n)]^3}$$

Hint: Let \bar{x} be the true solution to $f(x) = 0$ and $\Delta = \bar{x} - x_n$. Consider the Taylor expansion of $f(x_n + \Delta)$ about the point x_n :

$$0 = f(x_n + \Delta) \approx f(x_n) + \Delta f'(x_n) + \frac{\Delta^2}{2} f''(x_n). \quad (3)$$

Solve for the first Δ in the RHS to get $\Delta \approx -\frac{f(x_n)}{f'(x_n)} - \Delta^2 \frac{f''(x_n)}{2f'(x_n)}$. By setting $\Delta = 0$ in this RHS, we get in the LHS the Newton approximation for Δ . Plug this approximation into the Δ^2 term in the RHS of (3) to get what we want.

(b) We can verify that the regular Newton's method is quadratically convergent using the following two lines of *Maple*

```
x1:=simplify(series(x0-f(x0)/diff(f(x0),x0),x0=0,3)):
x1:=subs(f(0)=0,x1);
```

Here we have simplified the algebra by assuming that the zero of $f(x)$ occurs at $x = 0$. This simplification is justified by the fact that Newton's method is invariant to translations in the x -direction.

Your task is to use *Maple* to similarly verify that the iteration in part a is indeed cubically convergent.