Norges teknisk–
naturvitenskapelige universitet
Institutt for matematiske fag

Faglig kontakt under eksamen:
Anne Kværnø, tlf. 93542

EKSAMEN I FAG SIF5040 NUMERISKE METODER
Mandag 8. mai 2000
Tid: 09:00–14:00

Hjelpemidler: C1 – Alle trykte og håndskrevne hjelpemidler tillatt.
Alle kalkulatortyper tillatt.

Sensuren faller i uke 22.

Besvarelsen skal inneholde så mange mellomregninger at det tydelig går fram hvilke metoder og mellomresultater som er anvendt.

Bakerst i oppgavesettet finnes to vedlegg, som det kan være en idé å se over først.

Oppgave 1 Finn interpolasjonspolynomet $p(x)$ av lavest mulig grad som interpolerer funksjonen

$$f(x) = \frac{1}{(x-0.3)^2+0.1}, \qquad 0 \le x \le 0.4$$

i x-verdiene 0.0, 0.2 og 0.4.

Finn en øvre grense for feilen $|f(x) - p(x)|$.

Oppgave 2 Gitt integralet

$$I = \int_0^{0.4} \frac{1}{(x-0.3)^2+0.1} dx. \tag{1}$$

a) Bruk trapesformelen, med uniform skrittlengde $h = 0.1$ for å finne en tilnærmelse T til integralet.

Finn en øvre grense for feilen $|I - T|$.

b) Hvor liten må skrittlengden h være for at vi er garantert en feil på mindre enn $2 \cdot 10^{-2}$. Bruker vi trapesformelen med denne skrittlengden, blir den relle feilen $|I - T| \approx 7 \cdot 10^{-3}$, altså langt mindre enn hva som ble forlangt. Hva skyldes det?

c) Denne oppgaven går ut på å lage en adaptiv trapes-formel for integralet

$$I = \int_a^b f(x)dx,$$

dvs. at vi ønsker å kunne variere skrittlengden h over intervallet $[a, b]$, og samtidig sikre at den totale feilen er mindre eller omtrent lik en gitt feiltoleranse.

i). Anta at f'' varierer lite over intervallet $[a, b]$. La

$$T_1 = \frac{b-a}{2}(f(a) + f(b)), \qquad T_2 = \frac{b-a}{4}(f(a) + 2f(\frac{a+b}{2}) + f(b)).$$

Vis at

$$|I - T_2| \approx \frac{1}{3}|T_2 - T_1|$$

ii). Forklar hvordan dette kan brukes til å lage en adaptiv algoritme, dvs. en algoritme der total feil automatisk blir nogenlunde likt fordelt over delintervallene (tilsvarende adaptiv Simpsons metode, men nå basert på trapesformelen).

iii). Anvend algoritmen på integralet (1), med feiltoleransen $2 \cdot 10^{-2}$.

Oppgave 3 Gitt følgende differensialligning

$$y'' = y^3 - y \cdot y', \qquad 0 \le x \le 1 \qquad y(0) = \frac{1}{2}$$

Skriv om dette som et system av første ordens differensialligninger.

Systemet skal løses med en 2.ordens Runge-Kutta metode med skrittlengde $h = 0.2$. Tabellen under gir resultatene av dette for to ulike valg av $y'(0)$, nemlig 0 og $-1/12$:

i	0	1	2	3	4	5
x_i	0.0	0.2	0.4	0.6	0.8	1.0
y_i	0.5000	0.5025	0.5095	$--$	0.5365	0.5563
y_i'	0.0000	0.0238	0.0459	$--$	0.0884	0.1101
y_i	0.5000	0.4867	$---$	0.4760	0.4774	0.4828
y_i'	-0.0833	-0.0530	$--$	-0.0039	0.0170	0.0364

Her er $y_i \approx y(x_i)$.
Fyll ut de åpne plassene i tabellen.

Egentlig ønsker vi å løse et to punkts randverdiproblem, der $y(1) = 1/3$. Basert på resultatene i tabellen, hvordan vil du velge $y'(0)$ for å få en bedre tilnærmelse til løsningen i endepunktet?

Oppgave 4 Gitt datasettet

x_i	0	1	2	3	4
y_i	1.5	2.5	3.5	5.0	7.5

og funksjonen

$$y(x) = C \cdot \mathrm{e}^{Ax} \tag{2}$$

Oppgaven går ut på å beregne konstantene C og A slik at funksjonen best mulig tilpasses dataene.

a) Vis at (2) kan skrives som

$$\ln y = A \ln x + \ln C$$

og bruk lineær minste kvadraters metode for å beregne konstantene A og C.

b) Ved å bruke lineær minste kvadraters metode i **a)** har vi minimalisert

$$\sum_{i=0}^{4} (\ln y_i - A \ln x_i - \ln C)^2$$

Anta at vi istedet ønsker å minimalisere feilen

$$E(A, C) = \sum_{i=0}^{4} (y_i - C \cdot \mathrm{e}^{Ax_i})^2$$

direkte.

Skriv et MATLAB-program som beregner A og C ved bruk av funksjonen `fminsearch`, se vedlegg 2 for en beskrivelse.

Oppgave 5 Gitt Helmholtz' ligning

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + a \cdot u = 0, \qquad 0 \le x, y \le 1, \qquad a \text{ er en konstant} \tag{3}$$

med randbetingelser

$$\begin{aligned} u(x,0) &= \sin(\pi x), & 0 \le x \le 1 \\ u(0,y) &= \sin(\pi y), & 0 \le y \le 1 \\ \frac{\partial u}{\partial y}(x,1) &= 0, & 0 \le x \le 1 \\ u(1,y) &= 0, & 0 \le y \le 1 \end{aligned}$$

Ligningen skal løses numerisk med fem-punkts formelen, og vi bruker skrittlengde h i begge retninger. I det følgende er $x_i = ih$, $y_j = jh$ og $u_{ij} \approx u(x_i, y_j)$.

a) Skriv opp ligningene som må løses for å finne u_{ij} i alle gitterpunktene der løsningen er ukjent.

Dette systemet skal løses ved hjelp av SOR. Sett $h = 1/3$, $a = -1$ og utfør en SOR-iterasjon med relaksasjonsparameter $\omega = 1.2$. Bruk $u_{ij}^{(0)} = 0.5$ som startverdi.

b) For enkelthets skyld vil vi i resten av oppgaven bruke en Gauss-Seidel type iterasjon, modifisert slik at løsningene på samme rad oppdateres samtidig. Iterasjonskjemaet for de indre punktene blir

$$u_{ij}^{(k+1)} = \frac{1}{A}(u_{i+1,j}^{(k)} + u_{i-1,j}^{(k)} + u_{i,j+1}^{(k)} + u_{i,j-1}^{(k+1)})$$

hvor uttrykket for A burdte være kjent fra punkt **a)**.

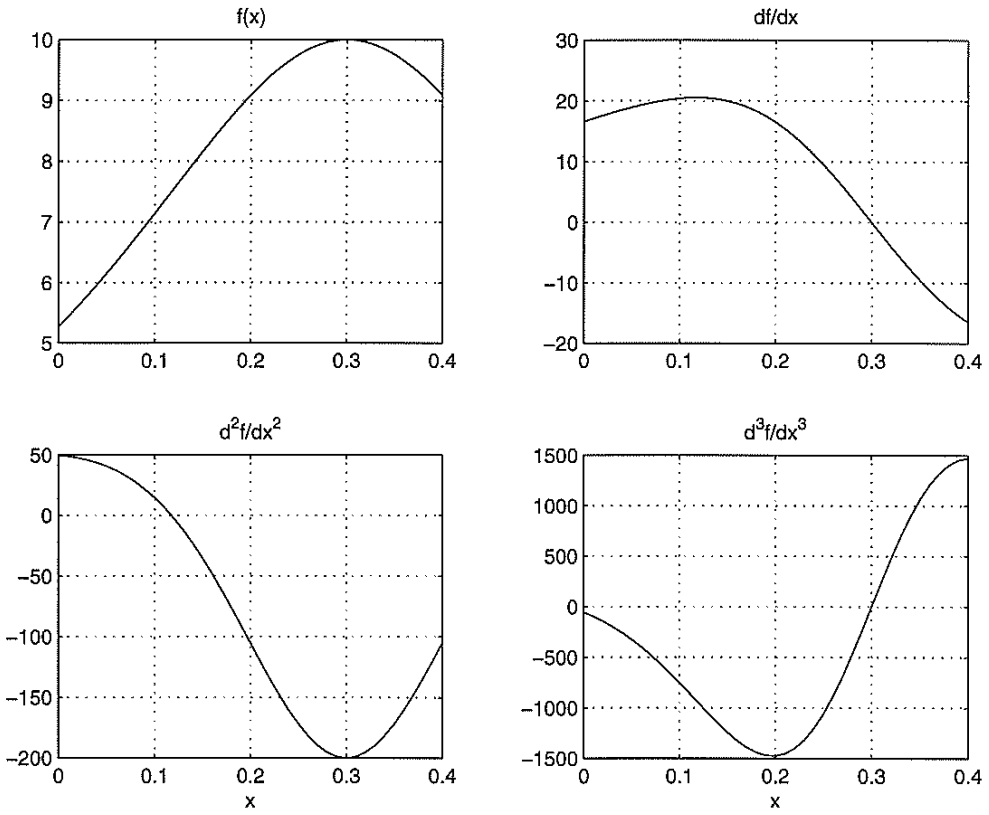
Si litt om hvilke fordeler og ulemper et slikt iterasjonsskjema har, sammenlignet med vanlig Gauss-Seidel.

Skriv et fullstendig MATLAB-program for løsning av (3) for en valgt verdi av a. Antall gitterpunkter inngår som en parameter som skal kunne velges, men bruk samme antall gitterpunkter i begge retninger. Bruk iterasjonskjemaet slik det er beskrevet ovenfor, men husk å ta hensyn til at ligningene blir litt anderledes på randa $y = 1$. Iterasjonene skal avsluttes når $|u_{ij}^{(k+1)} - u_{ij}^{(k)}| < 10^{-4}$ for alle i og j. Programmet skal også lage et plott av løsningen.

Grafer av funksjonen

$$f(x) = \frac{1}{(x-0.3)^2 + 0.1}$$

og dens deriverte:



MATLAB Function Reference **Go to function:** Search Help Desk

fminsearch

Examples See Also

Minimize a function of several variables

Syntax

```
x = fminsearch(fun,x0)
x = fminsearch(fun,x0,options)
x = fminsearch(fun,x0,options,P1,P2,...)
[x,fval] = fminsearch(...)
[x,fval,exitflag] = fminsearch(...)
[x,fval,exitflag,output] = fminsearch(...)
```

Description

fminsearch finds the minimum of a scalar function of several variables, starting at an initial estimate. This is generally referred to as *unconstrained nonlinear optimization.*

`x = fminsearch(fun,x0)` returns a vector `x` that is a local minimizer of the function described in `fun` (usually an M-file, built-in function or an inline object) near the starting vector `x0`. `fun` should return a scalar function value `f` evaluated at `x` when called with `feval`: `f=feval(fun,x)`.

`x = fminsearch(fun,x0,options)` minimizes with the optimization parameters specified in the structure `options`. You can define these parameters using the `optimset` function. `fminsearch` uses these `options` structure fields:

- `Display` - Level of display. `off` displays no output; `iter` displays output at each iteration; `final` displays just the final output.
- `MaxFunEvals` - Maximum number of function evaluations allowed.
- `MaxIter` - Maximum number of iterations allowed.
- `TolFun` - Termination tolerance on the function value.
- `TolX` - Termination tolerance on `x`.

`x = fminsearch(fun,x0,options,P1,P2,...)` passes the problem-dependent parameters `P1`, `P2`, etc., directly to the function `fun`: `feval(fun,x,P1,P2,...)`. Pass an empty matrix for `options` to use the default values.

`[x,fval] = fminsearch(...)` returns in `fval` the value of the objective function `fun` at the solution `x`.

`[x,fval,exitflag] = fminsearch(...)` returns a value `exitflag` that describes the exit condition of fminsearch:

- `> 0` indicates that the function converged to a solution `x`.
- `0` indicates that the maximum number of function evaluations was reached.
- `< 0` indicates that the function did not converge to a solution.

`[x,fval,exitflag,output] = fminsearch(...)` returns a structure `output` that contains information about the optimization:

- `output.algorithm` - The algorithm used.
- `output.funcCount` - The number of function evaluations.
- `output.iterations` - The number of iterations taken.

Arguments

`fun` is a string containing the name of the function that computes the objective function to be minimized at the point `x`. The function returns one argument, a scalar valued function `f` to be minimized, given a vector `x`. For example, if `fun='fun'`, the first line of the M-file `fun.m` is

```
f = fun(x)
```

`fun` can also be the name of a built-in function such as `fun='norm'`.(Note that `norm` takes a vector and returns a scalar.)

Alternatively, you can specify an inline object. For example,

```
fun = inline('sin(x''*x)');
```

Other arguments are described in the syntax descriptions above.

Examples

A classic test example for multidimensional minimization is the Rosenbrock banana function

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

The minimum is at `(1,1)` and has the value `0`. The traditional starting point is `(-1.2,1)`. The M-file `banana.m` defines the function.

```
function f = banana(x)
f = 100*(x(2)-x(1)^2)^2+(1-x(1))^2;
```

The statement

```
[x,fval] = fminsearch('banana',[-1.2, 1])
```

produces

```
x =

    1.0000    1.0000

fval =

    8.1777e-010
```

This indicates that the minimizer was found to at least four decimal places with a value near zero.

Move the location of the minimum to the point `[a,a^2]` by adding a second parameter to `banana.m`.

```
function f = banana(x,a)
if nargin < 2, a = 1; end
f = 100*(x(2)-x(1)^2)^2+(a-x(1))^2;
```

Then the statement

```
[x,fval] = fminsearch('banana', [-1.2, 1], ...
   optimset('TolX',1e-8), sqrt(2));
```

sets the new parameter to `sqrt(2)` and seeks the minimum to an accuracy higher than the default on `x`.

Algorithm

fminsearch uses the simplex search method of [1]. This is a direct search method that does not use numerical or analytic gradients.

If `n` is the length of `x`, a simplex in `n`-dimensional space is characterized by the `n+1` distinct vectors that are its vertices. In two-space, a simplex is a triangle; in three-space, it is a pyramid. At each step of the search, a new point in or near the current simplex is generated. The function value at the new point is compared with the function's values at the vertices of the simplex and, usually, one of the vertices is replaced by the new point, giving a new simplex. This step is repeated until the diameter of the simplex is less than the specified tolerance.

Limitations

fminsearch can often handle discontinuity, particularly if it does not occur near the solution. fminsearch may only give local solutions.

fminsearch only minimizes over the real numbers, that is, *x* must only consist of real numbers and *f(x)* must only return real numbers. When *x* has complex variables, they must be split into real and imaginary parts.

See Also

`fminbnd`, `optimset`, `inline`

References

[1] Lagarias, J.C., J. A. Reeds, M.H. Wright, and P.E. Wright, "Convergence Properties of the Nelder-Mead Simplex Algorithm in Low Dimensions," May 1, 1997. To appear in the *SIAM Journal of Optimization*.
