Iterative solution methods for mesh variational inequalities

the exercise

Task 2: Contact with Friction and constraits

Picture 1: residuals from an unfunctional version of the program used to solve the problem

Lassi Miinalainen
1928453
Oulu University
28.4.2009

The task was to find the optimal iteration parameter σ for the SOR-method by iterating a problem for which an exact solution was known. The problem was formulated as following:

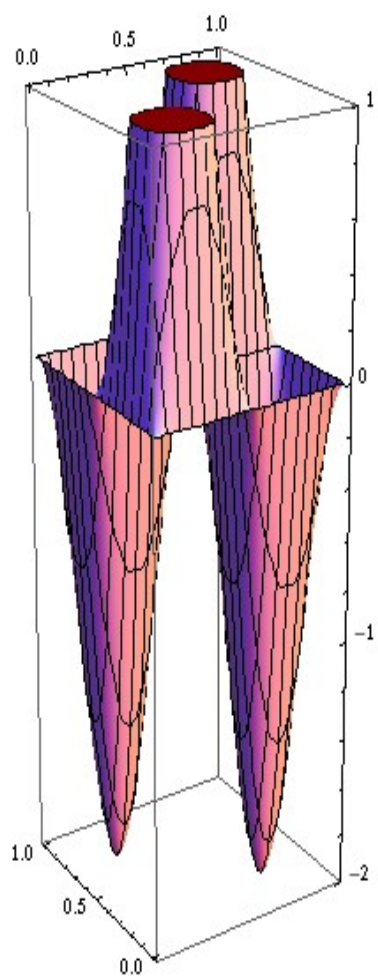
$$\int_{\Omega} \nabla u \cdot \nabla (v-u)\,dx + \int_{\Omega} (|v|-|u|)\,dx \geq \int_{\Omega} f*(v-u)\,dx \;\forall v$$

$$f\ given,\ u(\partial\Omega)=0,\ u(\Omega)\leq 1$$

which leads to a discrete approximation $f \in Au + \partial\varphi(u)$

where A is a matrix representing the Laplace operator and $\partial\varphi(u)=\begin{matrix} -1, u<0 \\ [-1,1], u=0 \\ 1\leq, u=1 \end{matrix}$.

The exact solution u was chosen to be $u=\begin{matrix} 2\,sin2\,\pi\,x \sin 2\,\pi\,y\,, z<1 \\ 1, z\geq 1 \end{matrix}$ (see the picture below). The right side was generated using this exact solution. Different iteration parameters were then used to find an iterated approximation fullfilling the stopping criterion of $\|u^k-u^*\|<h$. The actual optimal parameter is marked by σ*. There is a way to loosely predict σ* by $\sigma^0=2-ch$, where c is a constant and h the distance between grid points. In the next page are two tables showing the needed number of iterations for different values of σ.



Picture 2: a plot of exact solution

Size of mesh N=52

σ	1.0	1.3	1.6	1.7	σ* =1.755	1.8	1.9
n(ε)	306	161	69	49	42	50	54

From this sparse grid we can deduce the constant c used in predicting the optimal iteration parameter.

Size of mesh N=302

σ	1.91	1.94	σ^0 =1.96	1.97	1.98	σ* =1.9826	1.985	1.99
n(ε)	698	481	598	534	389	367	370	496

Source code of the program used in computing these iterations is attached below in case it proved itself interesting (which it surely will not do).

```
#include <stdio.h>
#include <math.h>
#define MESH 50 /* number of integral points */
#define MESHUPPER 2.0 /* Largest delta to try */
#define MESHLOWER 1.0 /* Smallest delta to try */
#define MESHSTEP 0.1 /* distance between tried iteration parameters delta */
#define PI 3.14159265358979323846264338327950288419716939937510

main(){

double exact[MESH*MESH];
double rightside[MESH*MESH];
double u[MESH*MESH];
double g[MESH*MESH];
double laplace[5][MESH*MESH];
int a;
int b;
int i=0;
int j=0;
int n=0;
double delta;
double h;
h=1.0/(((double) MESH) +1.0);

/* construct exact*/
for(a=0; a<MESH; ++a){
	for(b=0; b<MESH; ++b){
		if(2.0*sin(2*PI*((double) a+1)*h)*sin(2*PI*((double) b+1)*h) >= 1)
			{exact[a*MESH+b] = 1; }
		else{exact[a*MESH+b]=2.0*sin(2*PI*((double) a+1)*h)*sin(2*PI*((double)
								b+1)*h);};
	}
}
/* end of construct exact*/

/* construct laplace operator matrix*/
for(a=0; a<MESH*MESH; ++a){
	laplace[2][a]=4.0/(h*h);
};

for(a=0; a<MESH*MESH; ++a){
	if(a>=MESH){
		laplace[0][a]=-1.0/(h*h);
		laplace[4][MESH*MESH-1-a]=-1.0/(h*h);
	}
	else{
```

```
            laplace[0][a]=0;
            laplace[4][MESH*MESH-1-a]=0;
      };
      if(a % MESH !=0 ){
            laplace[1][a]=-1.0/(h*h);
      }
      else{
            laplace[1][a]=0;
      };

};

for(a=0; a<MESH*MESH; ++a){
      laplace[3][a]=-1.0/(h*h);
};

for(a=0; a<MESH; ++a){
      laplace[3][a*MESH+MESH-1]=0;
};
/* end of construct laplace operator matrix*/

/*construct rightside*/
for(a=0; a<MESH*MESH; ++a){
      rightside[a]=0;
      for(b=0; b<5; ++b){

            switch(b){
            case 0 :
                  if(a>=MESH){rightside[a]=rightside[a]+laplace[0][a]*exact[a-MESH];};
                  break;
            case 1 :
                  if(a>=1){rightside[a]=rightside[a]+laplace[1][a]*exact[a-1];};
                  break;
            case 2 :
                  rightside[a]=rightside[a]+laplace[2][a]*exact[a];
                  break;
            case 3 :
                  if(a<MESH*MESH-1){rightside[a]=rightside[a]+laplace[3][a]*exact[a+1];};
                  break;
            case 4 :
                  if(a<MESH*MESH-MESH-1){rightside[a]=rightside[a]+laplace[4]
                                                                  [a]*exact[a+MESH];};
                  break;
            };
      };

};

for(i=0; i<MESH*MESH; ++i){
      if(exact[i]<-0.00001){
            rightside[i]=rightside[i]-1;
            }
            else if(exact[i]>0.00001){
            rightside[i]=rightside[i]+1;
            };

};
/*end of construct rightside*/

for(delta=MESHLOWER; delta<=MESHUPPER; delta=delta+MESHSTEP){
      double k;
      k=0;
      n=0;
for(a=0; a<MESH*MESH; ++a){
      u[a]=0;
      g[a]=rightside[a];
```

```
};
while(k==0){
            /* iterate u */
            for(a=0; a<MESH; ++a){
                  for(b=0; b<MESH; ++b){
                        for(i=0; i<=1; ++i){
                              switch(i){
                                    case 0 :
                                          if(a>=1){g[a*MESH+b]=g[a*MESH+b]-
                                          laplace[0][a*MESH+b]*u[(a-1)*MESH+b];};
                                          break;
                                    case 1 :
                                          if(a*MESH+b>=1)
                  {g[a*MESH+b]=g[a*MESH+b]-laplace[1][a*MESH+b]*u[a*MESH+b-1];};
                                          break;
                              };
                        };

                        if(g[a*MESH+b]<=-1){
                        u[a*MESH+b]=(delta/laplace[2][a*MESH+b])*(g[a*MESH+b]+1);

                        }else if(g[a*MESH+b]>=1){
                        u[a*MESH+b]=(delta/laplace[2][a*MESH+b])*(g[a*MESH+b]-1);
                              if(u[a*MESH+b]>1){u[a*MESH+b]=1;};
                        }else{
                              u[a*MESH+b]=0;
                        };

                  };
            };
            /* end of iterate u */

            ++n;

            if(n>1000){
                  printf("iteration failed \n");
                  return(MESH);
            };

            /* construct new g */
            for(a=0; a<MESH; ++a){
                  for(b=0; b<MESH; ++b){
                  g[a*MESH+b]=(((1.0/delta)-1.0)*u[a*MESH+b]*laplace[2][a*MESH+b])
                                                +rightside[a*MESH+b];

                        for(i=3; i<5; ++i){
                              switch(i){
                                    case 3 :
                                          if(a*MESH+b<MESH*MESH-1)
                  {g[a*MESH+b]=g[a*MESH+b]-laplace[3][a*MESH+b]*u[a*MESH+b+1];};
                                          break;
                                    case 4 :
                                          if(a<MESH-1){g[a*MESH+b]=g[a*MESH+b]-
                  laplace[4][a*MESH+b]*u[(a+1)*MESH+b];};
                                          break;
                              };
                        };
                  };


            };
            /* end of construct new g*/

            /* check stopping criterion */
            k=0;
            for(j=0; j < MESH*MESH; ++j){
```

```
                k=k+(u[j]-exact[j])*(u[j]-exact[j]);
                };
                if(k<=1)
                {k=1;} else {k=0;};
                /* end of check stopping criterion */
        };



        printf("%0.5f : %d \n", delta, n);
};
}
```