Programowanie strukturalne w Pythonie

Klasa reprezentująca funkcje podcałkowe

Kod źródłowy: integrand_with_call.py

<sxh python> # -*- coding: utf-8 -*- # vim:fenc=utf-8 # # Copyright © 2017 putanowr <putanowr@foo> # # Distributed under terms of the MIT license.

“”“ Define integrand as scalar function in R. ”“”

class Integrand:

```
  def __init__(self, expression):
      self.expression = expression
```

```
  def evaluate(self, x):
      return eval(self.expression)
```

```
  def __call__(self, x):
      return self.evaluate(x)
```

if name == 'main':

```
  f = Integrand("x**2")
  z = f(4)
  print("Integrand value %f" % z,)
```

</sxh>

Obliczanie całki

Obliczanie całki z funkcji jednej zmiennej metodą trapezów na siatce nierównomiernej

Kod źródłowy: calculate_integral.py

<sxh python> #! /usr/bin/env python # -*- coding: utf-8 -*- # vim:fenc=utf-8 # # Copyright © 2017 putanowr <putanowr@foo> # # Distributed under terms of the MIT license.

“”“ Calculate integral of scalar function in 1D ”“” import sys import os

class Integrand:

```
"""Represents integrand as 1D scalar function
"""
def __init__(self, expression):
  self.expression = expression
```

```
def evaluate(self, x):
  return eval(self.expression)
```

```
def __call__(self, x):
  return self.evaluate(x)
```

class Mesh:

```
"""Represents on dimensional mesh
"""
def __init__(self):
  self.nodes = list()
```

```
def load(self, filename):
  """Read from file a list of nodes"""
  with open(filename, 'r') as f:
    for l in f:
      self.nodes.append(tuple(float(x) for x in l.split()))
```

```
def nelem(self):
  return len(self.nodes)-1
```

class MeshIntegrator:

```
"""Calculate integral over a domain discretised by a mesh
"""
```

```
def integrate(self, mesh, integrand):
  """Integrate function fun using numerical integration on given mesh
  """
  mesh_fun = self.make_mesh_fun(mesh, integrand)
  integral = 0.0
  for i in range(mesh.nelem()):
    integral += self.integrate_element(i, mesh, mesh_fun)
  return integral
```

def make_mesh_fun(self, mesh, fun):

```
  xcoord = [ node[0] for node in mesh.nodes ]
  discrete_fun = [ fun(x) for x in xcoord ]
  return discrete_fun
```

def integrate_element(self, i, mesh, mesh_fun):

```
  x1 = mesh.nodes[i][0]
  x2 = mesh.nodes[i+1][0]
  f1 = mesh_fun[i]
  f2 = mesh_fun[i+1]
```

```
  h = x2 - x1
  integral = h * (f1+f2) / 2.0
  return integral
```

def parse_command_line(argv):

```
"""Parse command line"""
argc = len(sys.argv)
if len(sys.argv) < 2:
  print("Mesh file name must be given")
  sys.exit(22)
meshfile = sys.argv[1]
if len(sys.argv) > 2:
  fun = sys.argv[2]
else:
  print("Using default function f(x) = x^2")
  fun = "x**2"
```

return [meshfile, fun]

def main():

```
[meshfile, fun] = parse_command_line(sys.argv)
mesh = Mesh()
mesh.load(meshfile)
integrand = Integrand(fun)
integrator = MeshIntegrator()
integral = integrator.integrate(mesh, integrand)
print("Integral of %s is %g" %(fun, integral))
```

if name == 'main':

```
main()
```

</sxh>

Version with quadrature defined as a class.

<sxh python> import sys import os import itertools

class Integrand:

```
"""Represents integrand as 1D scalar function
"""
def __init__(self, expression):
  self.expression = expression
```

```
def evaluate(self, x):
  return eval(self.expression)
```

```
def __call__(self, x):
```

```
  return self.evaluate(x)
```

class Mesh:

```
"""Represents on dimensional mesh
"""
def __init__(self):
  self.nodes = list()
```

```
def load(self, filename):
  """Read from file a list of nodes"""
  with open(filename, 'r') as f:
    for l in f:
      self.nodes.append(tuple(float(x) for x in l.split()))
```

```
def nelem(self):
  return len(self.nodes)-1
```

class TrapezoidQuadrature:

```
  ref_nodes = (-1,1)
  ref_weights = (0.5, 0.5)
  def nodes(self, a, b):
      return (a,b)
  def weights(self, a, b):
     return ((b-a)*x for x in self.ref_weights)
```

class MeshIntegrator:

```
"""Calculate integral over a domain discretised by a mesh
"""
```

```
def integrate(self, mesh, integrand):
  """Integrate function fun using numerical integration on given mesh
  """
  quadrature = TrapezoidQuadrature();
  integral = 0.0
  for i in range(mesh.nelem()):
    integral += self.integrate_element(i, mesh, integrand, quadrature)
  return integral
```

def make_mesh_fun(self, mesh, fun):

```
  xcoord = [ node[0] for node in mesh.nodes ]
  discrete_fun = [ fun(x) for x in xcoord ]
  return discrete_fun
```

def integrate_element(self, i, mesh, integrand, quadrature):

```
  x1 = mesh.nodes[i][0]
  x2 = mesh.nodes[i+1][0]
  qr = quadrature
  integral = 0.0
  for (x, w) in itertools.zip_longest(qr.nodes(x1, x2), qr.weights(x1,x2)):
    integral += w * integrand(x);
  return integral
```

def parse_command_line(argv):

```
"""Parse command line"""
argc = len(sys.argv)
if len(sys.argv) < 2:
  print("Mesh file name must be given")
  sys.exit(22)
meshfile = sys.argv[1]
if len(sys.argv) > 2:
  fun = sys.argv[2]
else:
  print("Using default function f(x) = x^2")
  fun = "x**2"
```

return [meshfile, fun]

def main():

```
[meshfile, fun] = parse_command_line(sys.argv)
mesh = Mesh()
mesh.load(meshfile)
integrand = Integrand(fun)
integrator = MeshIntegrator()
integral = integrator.integrate(mesh, integrand)
print("Integral of %s is %g" %(fun, integral))
```

if name == 'main':

```
main()
```

</sxh>