# Problem definition

Task which has been given consisted in computing the gravitational potential across a given two-dimensional grid imitating space containing a number of point masses. Each point in space has its own gravitational potential caused by the presence of masses in space. The higher the mass and the closer to the point it is the more influence it has on the gravitational potential at a given point. Gravitational potential infinitely far away from any mass is defined as zero therefore it is negative at any point in space. The potential at a distance x from a point mass of mass M is

G is the gravitational constant. In this task it is assumed that G is equal to 1 to avoid dealing with very small numbers.

# Problem solution

In order to calculate the gravitational potential properly at a certain point one has to keep in mind that net potential of the point is a sum of potentials caused by each body, so it is necessary to iterate over each point as many times as many bodies are defined in space. The first step of solving the problem was to present the universe as a variable. I have decided to create *Cell* object containing variables for gravitational potential of a cell from each body and a net potential. *Cell* object has a function to calculate net potential by adding partial potentials. Universe consists of an array of *Cell* objects. There is no need of adding the variables for coordinates of the cell, position of a variable in the array would be used to identify the coordinates. To calculate the gravitational potential it is necessary to have the distance between the mass point and the body, distance is calculated using Pythagorean theorem. To avoid dividing by zero I had to assume that if the distance is equal to zero the potential at that point is zero as well. To calculate potential of each grid I iterated over each row and each column. The number of iterations depends on the number of the bodies. It is assumed that the number of the bodies does not exceed five, so iterating five times over would be possible, but it would not be efficient. The next step is iterating again over each cell to calculate net potential then results are saved as a .csv file.

# Code explanation

Code listing . Variables declaration

int main**(**int argc**,** char **\*\***argv**)**

**{**

Cell universe**[**DIMENSION**][**DIMENSION**];**//universe grid

Body bodyArray**[**MAX\_BODIES\_NUMBER**];** //maxBodiesNumber is defined in LoadData.h

Body **\***pbodyArray **=** bodyArray**;** //a pointer to an array of data

char **\***pFileName**;** //a pointer to string with desired file name assigned by ChooseFileToRead() function

char sFileName**[**FILE\_NAME\_LENGTH**];** //string for storing a file name

pFileName **=** sFileName**;** //assigning the address of sFileName[] to a pointer pFileName

In the first six lines of the *main()* method variables are decelerated.

Variable named *universe* is a two-dimensional array of *Cell* type which represents a grid, dimensions of the array are defined as a constant in a *Universe.h* header to allow a programmer to increase grid size and therefore accuracy of calculations if needed.

Code listing . struct Cell

struct Cell

**{**

float fPotential**[**MAX\_BODIES\_NUMBER**];** //gravitational impact of each body

float fNetPotential**;**

void CalculateNetPotential**()**

**{**

**for** **(**int i **=** 0**;** i **<** MAX\_BODIES\_NUMBER**;** i**++)**

**{**

fNetPotential **+=** fPotential**[**i**];**

**}**

**}**

Cell**()** //setting all potentials to zero to get rid of the random values

**{**

**for** **(**int i **=** 0**;** i **<** MAX\_BODIES\_NUMBER**;** i**++)**

**{**

fPotential**[**i**]** **=** 0.0**;**

**}**

fNetPotential **=** 0.0**;**

**}**

**};**

Each Cell variable consist of a float for storing net potential of a cell and an array of floats named *fPotential* for storing potential caused by each body present in the universe. Struct *Cell* also contains a method for calculating net potential which is just simple addition of values stored in *fPotential*. Struct *Cell* also contains a constructor which sets all variables to zero to avoid storing random values which interrupt proper calculations. The next variable is an array of body objects. Each body object consists of float values which represent x and y coordinates as well as its mass. Pointers are assigned to *Body* array and a string containing file name in order to enable passing these variables as a parameters to the functions. *ChooseFileToRead()* function is called to assign a file name to a variable. This file name is used afterwards to load data from a text file. *LoadData()* function requires file name and a pointer to an array of type *Body*. File name indicates which file should be opened while a pointer to an array of *Body* type is needed to assign values from a text file to a *Body* variables. Next step is loading data from a text file. Parameters for a *LoadData()* function are file name which is resolved by *ChooseFileToRead()* function and a pointer to *Body* type array for indication to which variable a data from a text file should be assigned to. Depending on what value *DetermineHowManyBodies()* returns the proper method of reading the text file is used. The next step is iterating over all cells to calculate potential of each point. The number of iterations depends on the number of the bodies included in a given text file. Then all cells are iterated over again to calculate net potential of each cell. When all cells have assigned net potential all the data from the array of *Cell* is exported to a .csv file.

struct Body

**{**

float x**,** y**,** mass**;** //data from a text file

**};**

Code listing . struct Body

Code listing . ChooseFileToRead() and LoadData() functions calls

ChooseFileToRead**(**pFileName**);**

LoadData**(**pFileName**,** pbodyArray**);** //now all the values from a chosen textfile are stored in Body bodyArray[]

# Potential improvements

Code listing . Calculating the gravitational potential at a given poin

float CalculatePointPotential**(**float x**,** float y**,** Body body**)**

**{**

//calculate the distance between a point and a body

float fDistance **=** sqrt**(**pow**(**body**.**x **-** x **\*** GRID\_SIZE**,** 2**)** **+** pow**(**body**.**y **-** y **\*** GRID\_SIZE**,** 2**));**

// GRID\_SIZE indicates the size of a single grid in AU units

//calculate the potential V = -G\*m/distance

float fPotential**;**

**if** **(**fDistance **==** 0.0**)** //to avoid division by zero

fPotential **=** 0.0**;**

**else**

fPotential **=** **-((**GRAVITATIONAL\_CONSTANT **\*** body**.**mass**)** **/** fDistance**);**

**return** fPotential**;**

**}**

However, program calculates the gravitational point at any point it can be improved to robust its efficiency and flexibility. The main issue of the program is that files which contain no more than five bodies data can be processed, if there are more the program is not capable of taking them into account while calculating the gravitational potential. There should be possibility of processing files containing any number of defined bodies. The other improvement which might be applied is a structure of a struct *Cell*. It consists of two variables, but *fNetPotential* variable is redundant since net potential can be easily figured out by adding values from *fPotenial[]*. Removing *fNetPotenial* would reduce the amount of memory used by a program and it is worth remembering that *universe* variable is made of 10000 *Cell* objects, so it would be great enhancement to decrease a size of a *Cell* struct. The other drawback in the code is that when the size of the grid was greater than 205x205 the program crashed. If this problem was solved the accuracy of the calculations might be significantly improved. I think the grid limit is related the amount of memory used, supposedly larger grid could be used after reducing a size of a *Cell* struct. Another potential solution for the grid limit would be writing each cell into a databased instead of the array of type *Cell*, size limit would be much higher. Another improvement which might be made is allowing to load a file from any location not just four files which are hard-coded. The last improvement I can think of is displaying a message after the file is processed and .csv file is exported.

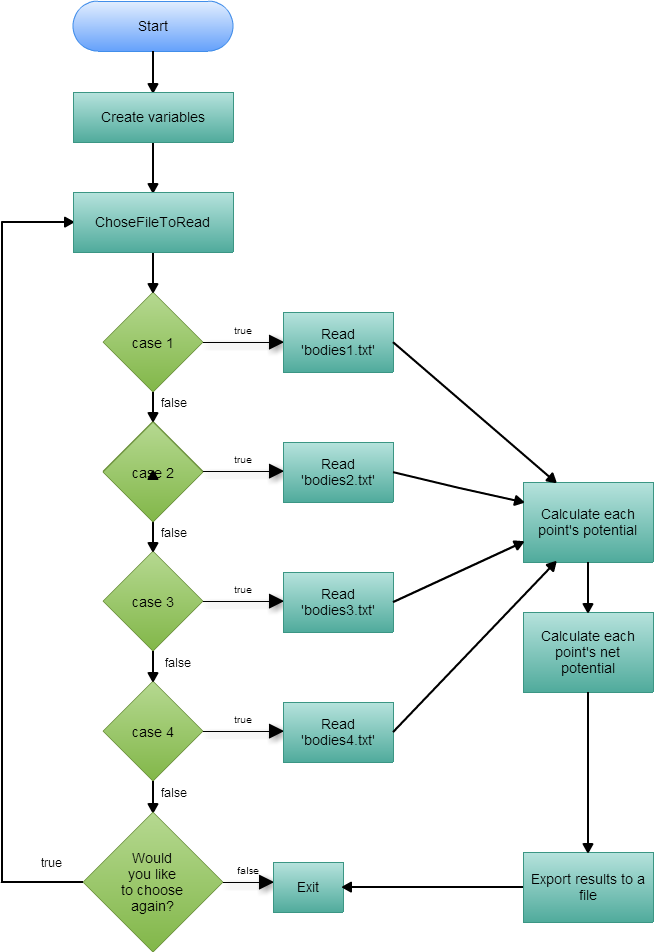


Figure . Main() function flowchart

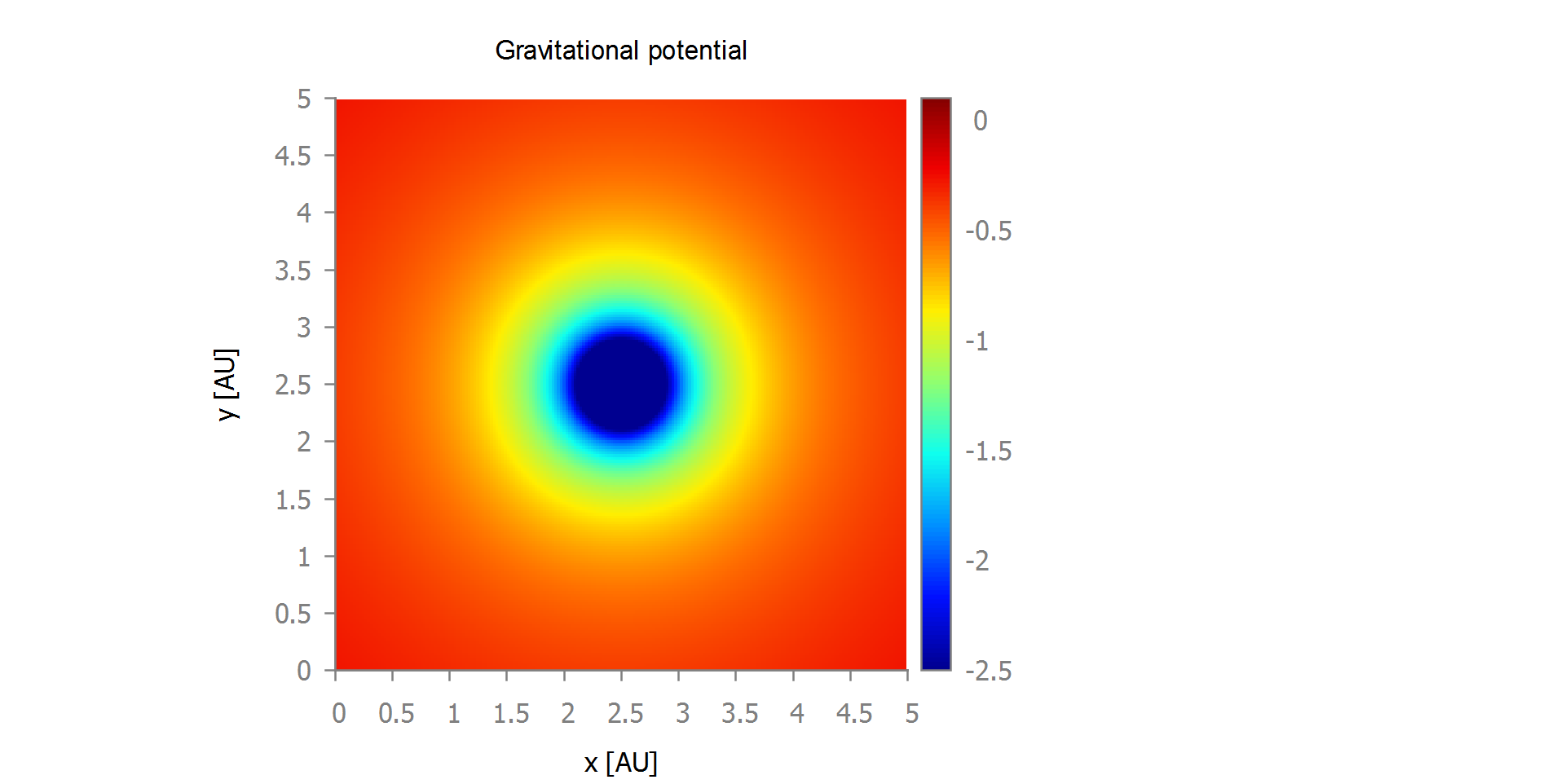


Figure . Plot of the gravitational potential of the bodies defined in bodies1.txt

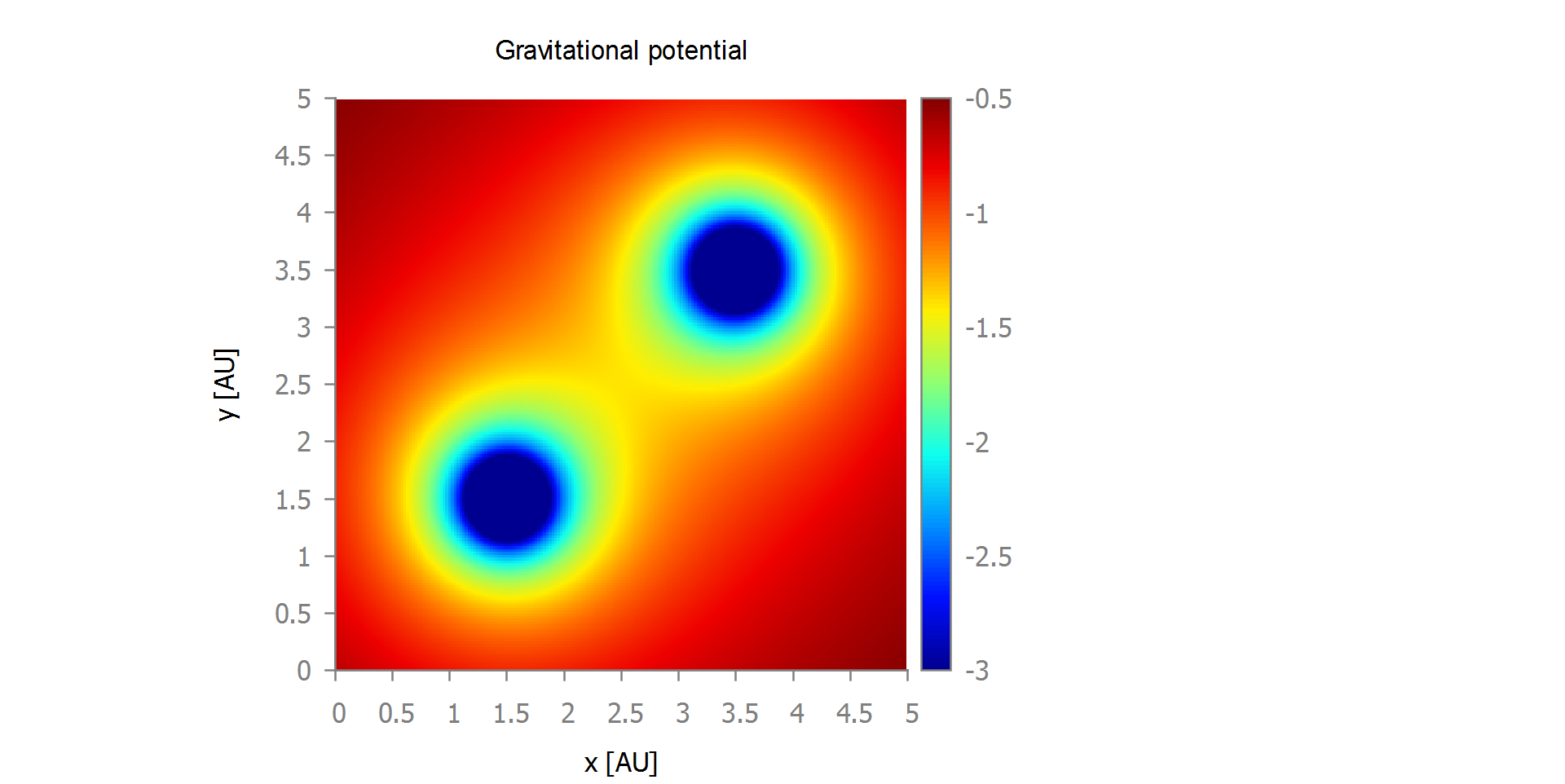


Figure . Plot of the gravitational potential of the bodies defined in bodies2.txt

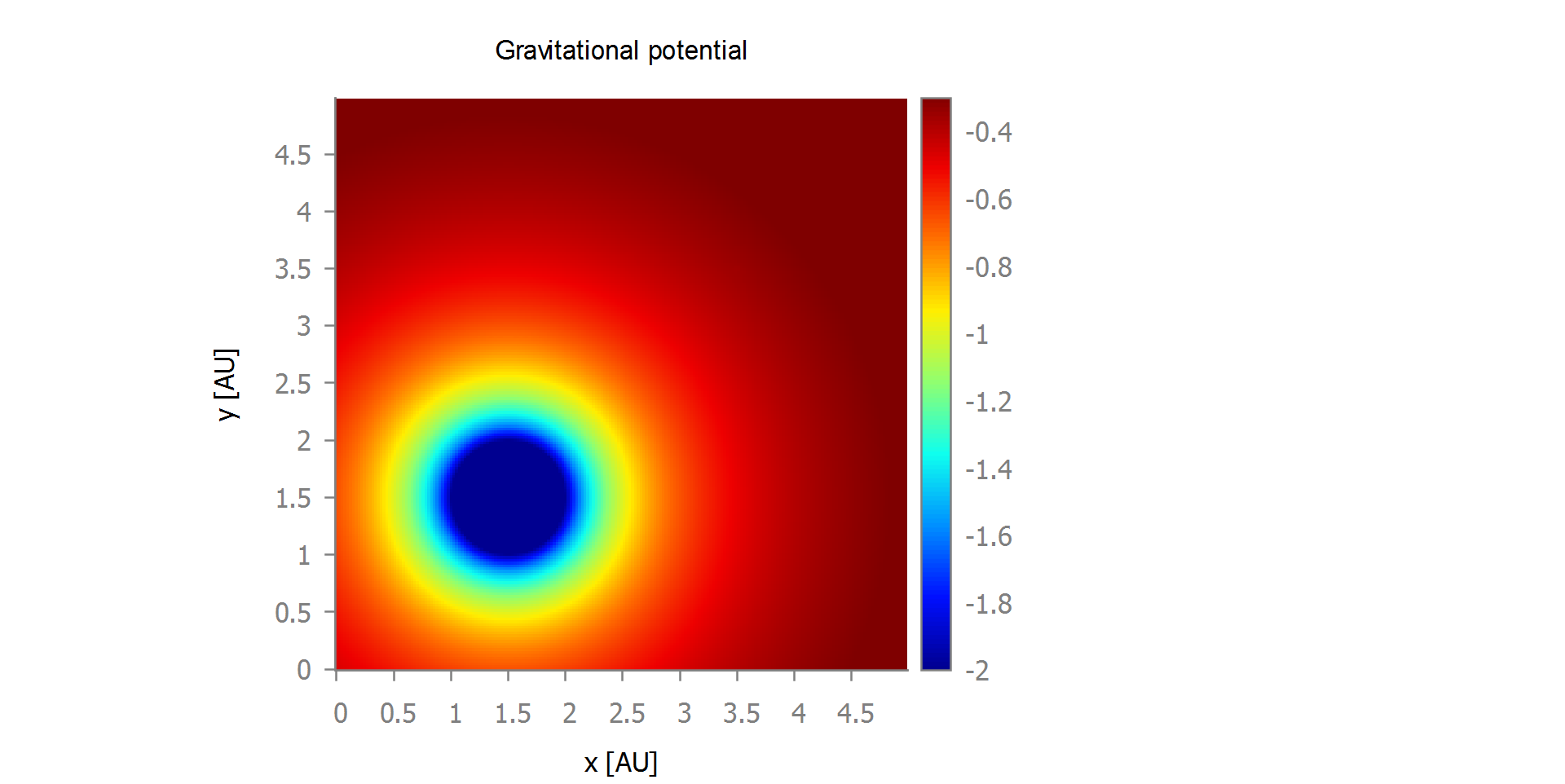


Figure . Plot of the gravitational potential of the bodies defined in bodies3.txt

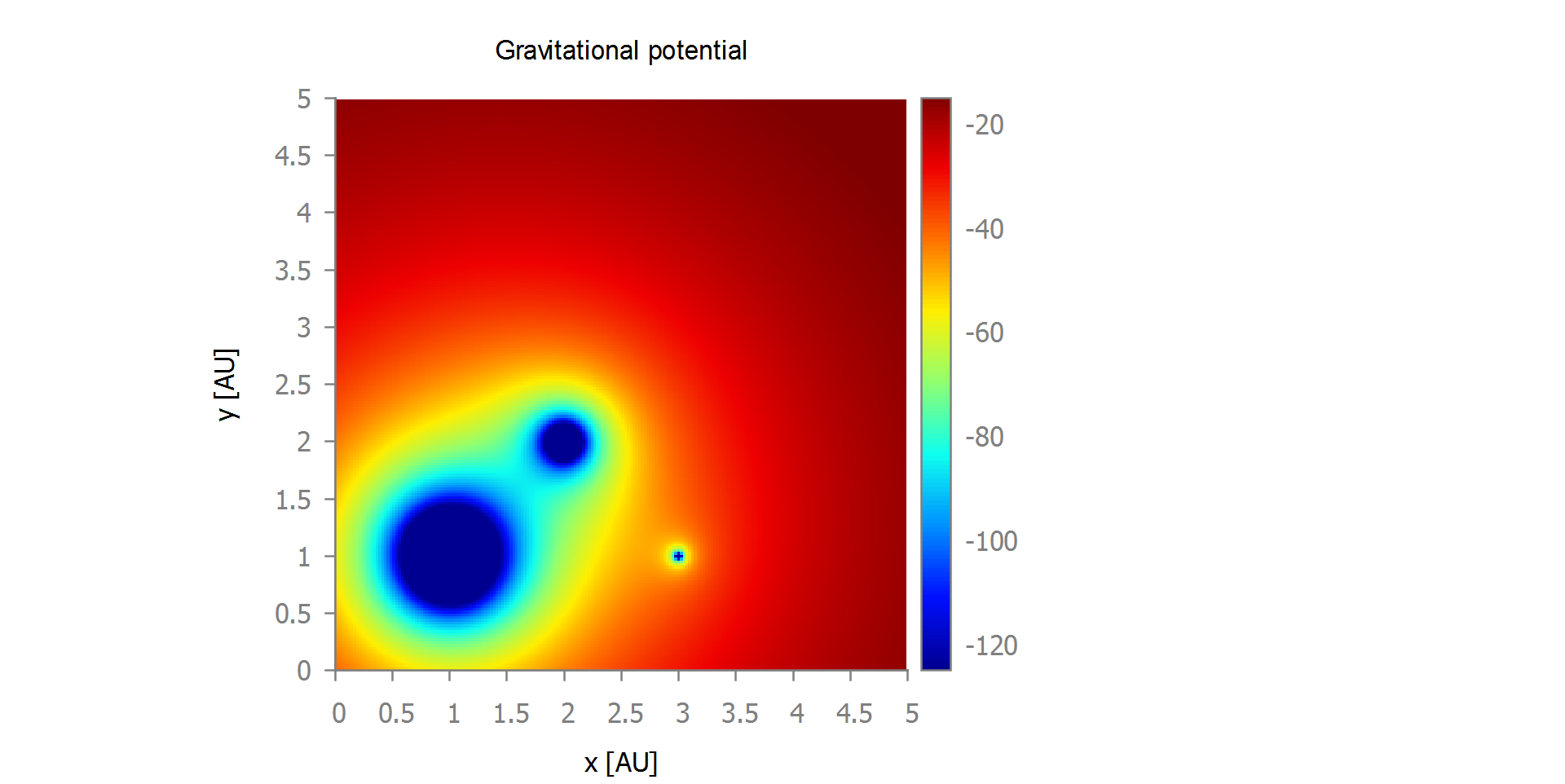


Figure . Plot of the gravitational potential of the bodies defined in bodies4.txt